Size-based Scheduling to Improve the Performance of Short TCP Flows

Idris A. Rai, Ernst W. Biersack, Guillaume Urvoy-Keller
Institut Eurecom
2229, route des Crêtes
06904 Sophia-Antipolis, France
{rai,erbi,urvoy}@eurecom.fr

November 2, 2004

Abstract

The Internet today carries different types of traffic that have different service requirements. A large fraction of the traffic is either Web traffic requiring low response time or peer-to-peer traffic requiring high throughput. Meeting both performance requirements in a network where routers use droptail or RED for buffer management and FIFO as service policy is an elusive goal. It is therefore worthwhile to investigate alternative scheduling and buffer management policies for bottleneck links. We propose to use the *least attained service* (LAS) policy to improve the response time of Web traffic. Under LAS, the next packet to be served is the one belonging to the flow that has received the least amount of service. When the buffer is full, the packet dropped belongs to the flow that has received the most service. We show that under LAS, as compared to FIFO with droptail, the transmission time and loss rate for short TCP flows are significantly reduced, with only a negligible increase in transmission time for the largest flows. The improvement seen by short TCP flows under LAS is mainly due to the way LAS interacts with the TCP protocol in the slow start phase, which results in shorter round-trip times and zero loss rates for short flows.

1 Introduction

1.1 QoS building blocks

Offering service guarantees to existing and emerging applications in the Internet has been a big challenge to Internet designers. Most of these applications are either interactive such as Web applications, have delay constraints such as multimedia streaming applications, or require high throughput such as file transfer. A number of mechanisms were proposed in the past to enable the Internet to support applications with varying service requirements. Examples of these mechanisms include *admission control*, *active queue management*, and *scheduling*.

Admission control is meant to prevent network overload so as to assure the service of the applications already admitted [14]. Admission control is deployed at the network edges where a new flow is either admitted when the network resources are available or rejected otherwise. A flow is defined as a group of packets with a common set of attributes such as (source address, destination address, source port, destination port). Although it seems plausible to use admission control in the Internet, it has not been deployed so far.

Active queue management has been proposed to support end-to-end congestion control in the Internet. The aim of active queue management is to anticipate and signal congestion before it actually occurs. One of the most important active queue management approaches is *random early discard* (RED) [5]. RED allows for dropping packets before the buffer overflows and was proposed to achieve various goals such as to assure fairness among sources with different round trip times, to reduce average transfer delays, to punish unresponsive flows, and to assure a better interaction with TCP congestion control. A flow is declared unresponsive if its source does not respond to network congestion. However, in practice RED fails to offer most of these features. For example, RED cannot protect TCP flows against unresponsive flows. Similarly, for Web flows, FIFO with RED was shown to yield similar flow transfer times as FIFO with droptail.

One of the most important mechanisms to provide service guarantees is scheduling. Scheduling determines the order in which the packets from different flows are served. *Packet* scheduling in routers has been an active area of research in the last two decades, and most of the attention has focused on *Processor sharing* (PS) type of scheduling algorithms [17]. Processor sharing has many interesting properties: It assures a max-min fair allocation of bandwidth among competing flows, provides protection against unresponsive flows, and allows to establish end-to-end delay bounds for individual flows [11]. Despite all these properties, PS scheduling has not been widely deployed in the Internet where routers still implement FIFO scheduling.

The above mechanisms are building blocks out of which *Quality of Service* (QoS) architectures can be assembled. Internet QoS allows to either guarantee the required service for all applications or to provide better service to certain applications, for instance, by satisfying their minimum service requirements. The performance guarantees in QoS architectures are expressed in terms of delay, jitter, throughput, and loss. Examples of the QoS architectures include *integrated services* (Intserv), *differentiated services* (Diffserv), or *traffic engineering* [16]. Unfortunately, like their basic building blocks, the QoS architectures also have not been dwidely eployed in the Internet mainly due to lack of scalability and backward compatibility. As a consequence, the Internet today still offers only a *best-effort* service with TCP being the most widely used transport protocol for end-to-end data transfer.

1.2 Variability of Internet traffic

Studies have shown that Internet traffic exhibits a *high variability property*: Most of the TCP flows are short, while more than 50% of the bytes are carried by less than 5% of the largest flows [12, 4]. Short flows are mainly Web data transfers triggered by user interactions. As the use of the Web remains popular, Web traffic will continue not only to make up a significant fraction of the Internet traffic but also to play a major role in the variability of the overall traffic distribution. The largest flows seen in the Internet originate from peer-to-peer file sharing services that have recently become very popular. The variability of Internet traffic can negatively affect the performance experienced by the end users. The existing Internet uses TCP as a transport protocol and FIFO as a scheduling discipline in routers. As we will see later, both of them penalize short flows.

The goal of this work is to study an alternative link scheduling policy to FIFO scheduling with droptail in packet networks in order to improve the overall user perceived performance by favoring short flows without penalizing the performance of long flows too much. The class of *size-aware scheduling* policies that give service priority to packets from short flows are an ideal candidate to achieve these goals.

2 Size-aware scheduling policies

2.1 Background

Size-aware scheduling policies have been extensively studied in the area of operating systems, where they are used to decide which *job* to service next. Examples of size-aware scheduling policies are *Shortest Job First* (SJF), *Shortest Remaining Processing Time* (SRPT), and *Least Attained Service* (LAS) [9]. These policies have been known for several decades, but they have not been proposed so far for packet networks. The reason being that when analyzed under the M/M/1 queue, where the service times do not exhibit the high variability property, these policies are known to favor short jobs but also to penalize the largest jobs a lot. The high variability property of the flow sizes observed in the Internet prompted researchers to revisit size-aware policies and study their performances for workloads exhibiting the high variability property.

SJF is a non-preemptive scheduling policy that gives service to the shortest job. A newly arriving shortest job must wait for the job in service to complete before it can receive the service. SRPT is a preemptive version of SJF, which favors short jobs by giving service to the job that has the shortest remaining processing time. In order to know the remaining processing time though, one needs to know the total service required by the job (the size of the job). Fair sojourn protocol (FSP) [6] is a very recent scheduling discipline that combines some elements of PS and SRPT. FSP services next the job that would complete service first among all jobs when the service discipline would be PS. It has been shown [6] that FSP assures that the response time is never higher than the one achieved by PS. However, FSP requires to know job sizes as does SRPT.

Suitable policies for link scheduling in the Internet must not require any knowledge about flow sizes. *Least attained service* (LAS) is a size-aware scheduling policy that does not need to know the flow size. In this paper, we study the interaction of LAS with TCP congestion control and use simulation to show the benefits of LAS in packet networks.

2.2 LAS scheduling

LAS is a preemptive scheduling policy that favors short jobs without prior knowledge of the job sizes. To this end, LAS gives service to the job in the system that has *received the least amount of service*. In the event of ties, the set of jobs having received the least service shares the processor in a processor-sharing mode. A newly arriving job always preempts the job currently in service and retains the processor until it departs, or until the next arrival occurs, or until it has obtained an amount of service equal to that received by the job preempted on arrival, whichever occurs first. In the past, LAS has been believed to heavily penalize large jobs. However, it has recently been shown that the mean response time of LAS highly depends on the job size distribution [13]. LAS is also known as *Foreground-Background* (FB) [9] or *Shortest Elapsed Time* (SET) first scheduling.

In operating systems scheduling, the term *job* is commonly used to denote a piece of workload that arrives to the system all *at once*. Given this definition, a *flow* in a packet network cannot be considered as a job since a flow does not arrive at the link at once. Instead, the source transmits a flow as a sequence of packets, possibly spaced out in time, that are in turn statistically multiplexed with packets from other flows. Also, when TCP is used as transport protocol, due to the closed-loop nature of TCP, the treatment given to an individual packet may affect the temporal behavior when transmitting the remaining packets in the flow. For these reasons, we are not sure if the analytical results on LAS that have been derived for jobs can be directly be applied to flows. Therefore, we use simulation to investigate the performance of

LAS for flows.

In the context of flows, the next packet serviced under LAS is the one that belongs to the flow that has received the *least amount* of service. By this definition, LAS will serve packets from a newly arriving flow until that flow has received an amount of service equal to the amount of least service received by a flow in the system before its arrival. If two or more flows have received an equal amount of service, they share the system resources fairly. In packet networks, LAS is not only a scheduling discipline as in operating systems but also comprises a buffer management mechanism: When a packet arrives to a queue that is full, LAS first inserts the arriving packet at its appropriate position in the queue and then drops the packet that is at *the end of the queue*. Therefore, LAS gives buffer space priority to short flows as the packet discard mechanism under LAS biases against flows that have utilized the network resources the most, which differs from droptail or RED buffer management.

Simulation results in this paper show that LAS significantly reduces the mean transmission time and loss rate of short TCP flows as compared to a FIFO scheduler with droptail, with a small increase in mean response time of large flows. The results also show that under moderate load values a long-lived ftp flow under LAS is not locked out when competing with short TCP flows. However, the performance of the long-lived flow under LAS deteriorates severely when competing at high load against short flows. For this reason, we propose another class-based LAS policy that is able to guarantee service to long-lived TCP flows if needed.

3 The impact of TCP congestion control on short TCP flows

3.1 TCP congestion control

The main task of TCP congestion control is to adjust the sending rate of the source in accordance with the state of the network. For this purpose, TCP limits the amount of outstanding data. The congestion window (`cwnd`) represents the maximum amount of data a sender can have sent and for which no acknowledgment was yet received. In particular, when the source starts sending data, TCP conservatively initializes `cwnd` to one packet and then doubles `cwnd` for every window worth of ACKs received until congestion is detected. This behavior is referred to as *slow start* (SS). Network congestion is identified by packet losses. When loss is detected, `cwnd` is reduced to react to congestion and to reduce the risk of loosing more packets. The *congestion avoidance* algorithm is then used to slowly increase `cwnd` in a linear manner by one packet for every window worth of packets that are acknowledged. This is called the congestion avoidance (CA) phase.

TCP uses two error recovery mechanisms. The first one is timeout based and relies on the *re-transmission timeout* (RTO) to decide when to retransmit. At the beginning, the RTO is initialized to a conservatively chosen value *Initial Timeout* (ITO), which is often set to a value as high as 3 seconds, and is later updated using the measured *round trip time* (RTT) samples. When a source does not receive during a period RTO the ACK for a packet, the timer expires. The source then first retransmits the packet and goes into slow start and sets `cwnd` to one packet. The second error recovery mechanism in TCP is called *fast-retransmit*: A packet i is considered lost if the sender receives four times in a row the acknowledgment for packet $i-1$. The sender then immediately retransmits packet i, sets `cwnd` to half its previous value, and goes into congestion avoidance. The details of the subsequent behavior of TCP depend on the version of TCP.

3.2 The impact of LAS to short TCP flows

A closer look at the behavior of the TCP congestion control reveals a number of factors that negatively affect the performance of short flows. Short flows complete their data transfer during the SS phase. However, even if the network is completely uncongested, the duration of a transfer during SS is dominated by the round trip time (RTT) of a connection, which is in general much longer than the time to actually transfer the packets of short flows. A short flow that has to send $n = 2^{k+1}$ packets and does not experience any loss of packets or acknowledgments will need $R(n) = k \times RTT$, where the round-tip time RTT is defined as $RTT = 2(T_{e2e} + T_Q)$ and T_{e2e} denotes the end-to-end propagation and transmission delay and T_Q the queueing delay. Under LAS, the first packets in a flow will have service priority and see no or little queuing delay T_Q, which reduces the transfer time $R(n)$ for short flows as compared to FIFO.

The following three figures compare the qualitative behavior of TCP under LAS and under FIFO with drop tail for different scenarios. Figure 1 plots the sequence number of received packets versus transfer time. It illustrates the impact of LAS and FIFO with droptail on the transfer time $R(n)$ for the case that there is no loss of packets. We see that a short flow of L packets transmits its packets faster under LAS than under FIFO. The difference is mainly due to a lower RTT under LAS than under FIFO. As a result, the transmission time of the same flow under LAS ($T^{LAS}_{no\ loss}$) is lower than the transmission time under FIFO ($T^{FIFO}_{no\ loss}$).

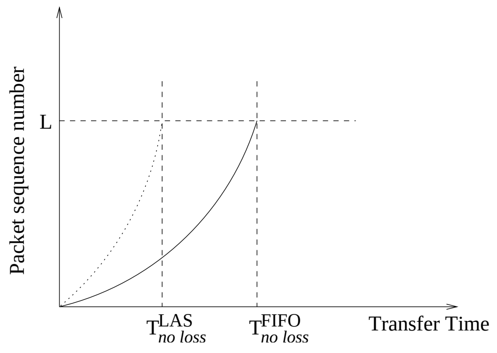


Figure 1: Slow start behavior for short TCP flows with no packet loss under both polices. Dotted line: LAS, solid line: FIFO.

Short TCP flows also have poor loss recovery performance since they often do not have enough packets in transit to trigger the fast retransmit mechanism. Instead, they must rely on the retransmission timeout to detect losses, which can significantly prolong the transmission time of a short TCP flow; TCP uses its own packets to sample the round trip times and to estimate the appropriate RTO value. When a lost packet is among the first packets of a flow, which is very likely for short flows, TCP uses the ITO as timeout value to detect the loss. Thus, losing packets in the SS phase can significantly prolong the transmission time of a short flow.

The packet discard mechanism under LAS significantly reduces packet losses during the initial slow start phase for all flows, regardless of their sizes. Instead, LAS mainly drops packets from large flows that have already transmitted a certain amount of data (have low service priority). Hence under LAS there is hardly a need for packet error recovery to rely on the retransmission timeout to detect packet loss. This is an important feature of LAS for short flows that FIFO scheduling cannot offer. Figure 2 shows the sequence number versus time plot for a short flow of size L packets that does not experience losses under LAS but looses packet $(L_l + 1)$ under FIFO. The short flow under FIFO retransmits the packet after time RTO^{FIFO}, which prolongs the transmission time T^{FIFO}_{loss} of the flow.

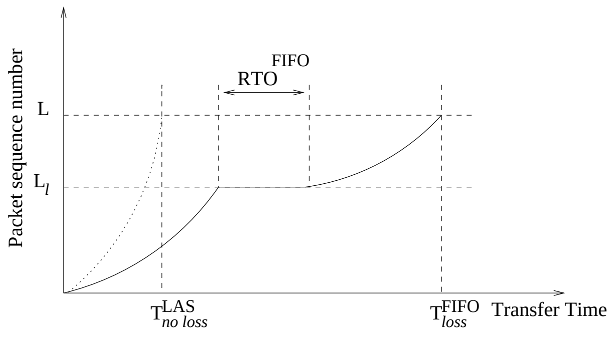


Figure 2: Slow start behavior for short TCP flows with packet loss under FIFO. Dotted line: LAS, solid line: FIFO.

Consider also the case when the short flow under both policies loses packet $(L_l + 1)$. Figure 3 shows that the response time under LAS is still lower than the response time of the flow under FIFO. It is important to note here that RTO^{FIFO} is bigger than the RTO^{LAS}. This is because the computations of RTO in TCP at any time depends directly on previous RTT sample values. These values are smaller under LAS than under FIFO due to the absence of queueing delay for the packets transmitted during SS under LAS.

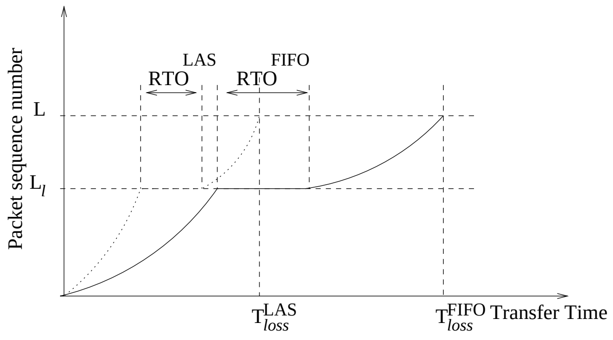


Figure 3: Slow start behavior for short TCP flows with packet loss under both policies. Dotted line: LAS, solid line: FIFO.

In practice, packet losses are likely to occur sooner under FIFO with droptail than under LAS. In this case, FIFO is very likely not to have enough RTT samples to compute the retransmission timeout, and TCP must use the high initial timeout value ITO to detect the losses. This again lengthens the transmission time of short flows under FIFO.

To summarize, we saw that the two main factors that allow LAS to improve the performance of short TCP flows are the (i) reduction of the RTT during slow start phase and (ii) the reduction of the losses seen by short flows. Note that the arguments presented in this section apply not only to short flows but also to long TCP flows during their initial slow start phase.

4 Related work

Closely related to our work are network-based approaches that improve the performance of short flows. A number of two-queue based policies have been proposed [7, 3, 8, 1] that give service priority to short flows. These policies use a variable `th` to differentiate between short and long flows: The first `th` packets of a flow are enqueued in the first queue and the remaining ones of a flow are enqueued in the second queue. Each queue is served in FIFO order and packets from the second queue are only served if the first queue is empty. Two-queue based disciplines reduce the mean transfer times of the short flows compared to FIFO scheduling. For example, simulation results show that the proposed *short flows differentiating* (SFD) policy reduces the mean transmission time of short flows by about 30% [3]. The work in [8] shows a reduction of the mean transfer time by about 80% for medium sized flows between 50-200 packets compared to simple FIFO with RED. A recent scheme called RuN2C [1] proposes a two-queue policy that is similar to the policy proposed in [3]. However, LAS offers lower transfer times to short flows than RuN2C because packets of short flows under RuN2C are serviced using FIFO and they must always wait behind any packets they meet in the first queue upon their arrival. In addition, a practical difficulty with all threshold-based schemes is the proper tuning of the threshold value `th` to capture all short flows.

Williamson and Wu [15] proposed a Context Aware Transport/Network Internet Protocol (CATNIP) for Web documents. This work is motivated mainly by the fact that loss of the first packets of a TCP flow has a much more negative impact on the transmission time of a flow than loss of other packets. CATNIP uses application layer information, namely the Web document size, to provide explicit context information to the TCP and IP protocols; Using CATNIP, IP routers identify and mark packets of a flow differently to avoid that the most important packets get dropped. Experiments with CATNIP in routers show that it can reduce the mean Web document retrieval time by 20%–80%. In contrast to CATNIP, LAS does not require the knowledge of flow sizes.

5 Simulation results

To evaluate LAS, we simulate it with the ns-2 network simulator for the case where a pool of clients request Web objects from a pool of servers. The simulated Web traffic profile is obtained by setting the distributions of inter-session, inter-page and inter-object time (all in seconds), session size (in web pages), page size (in objects) and flow size (in packets) as seen in Table 1. The parameters used in the table result in a total load for Web traffic of $\rho = 0.7$. $Exp(\mu)$ denotes exponential distribution with mean μ. We consider a Pareto *object size* distribution denoted as $P(a, \alpha, \bar{x})$, where a is the minimum possible object size, α is the shape parameter, and $\bar{x}$ is the mean object size. The Pareto distribution is used since it exhibits a high variability property and models well flow size distributions as empirically observed in the Internet. Note that the transfer of each object will result in a new flow.

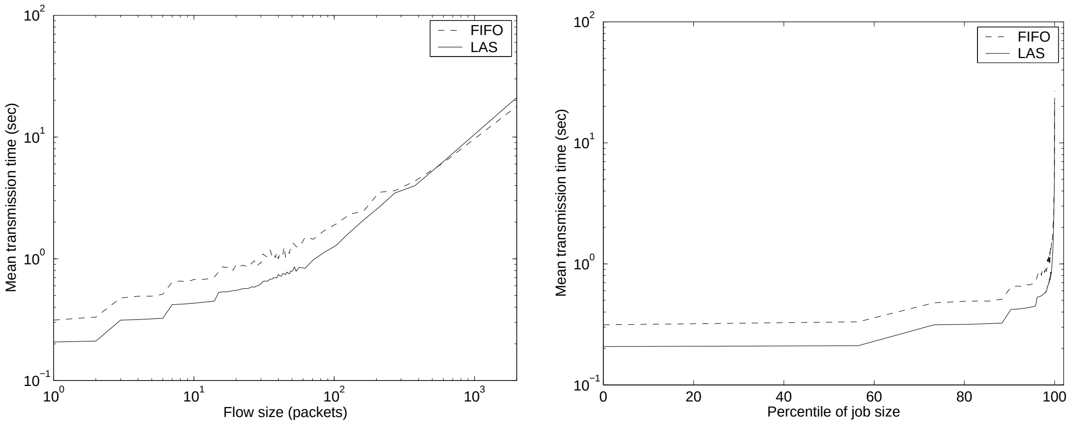
pages/session	**objs/page**	**inter-session**	**inter-page**	**inter-obj**	**obj size**
Exp(60)	*Exp(3)*	*Exp(8)*	*Exp(5)*	*Exp(0.5)*	*P(1,1.2,12)*

Table 1: Web traffic profile

In this section we use simulations to show the benefits of LAS in terms of reducing the mean transfer times and loss rates for short flows as compared to FIFO scheduling with droptail.

5.1 Mean transmission time

The transmission time of a flow is the time interval starting when the first packet of a flow leaves a server and ending when the last packet of the flow is received by the corresponding client. Figures 4(a) and 4(b) show the mean transmission time for LAS and FIFO as a function of flow size and percentile of object size respectively. The pth percentile is the flow size x_p such that $F_x(x_p) = p/100$, where $F_x(x)$ is the cumulative distribution function of x. We see that LAS significantly reduces the mean transmission time with a negligible penalty to large flows. More than 99% of the flows benefit from LAS.



(a) *Flow sizes* (b) *Percentiles*

Figure 4: Mean transmission time of Pareto Web flow sizes, load $\rho = 0.7$

We also observe that the mean transmission time of large flows under LAS are only slightly higher than under FIFO. When experiencing losses, large flows are likely to have large congestion windows that permit the use of fast retransmission to recover the losses, which helps reduce the transfer time of large flows as compared to recovery via timeout. We also investigated the performance of LAS at other load values and observed that short flows also benefit under LAS at lower load values like $\rho = 0.5$. However, the performance of short flows under LAS as compared to FIFO improves with increasing load values.

5.2 Loss rate

The overall performance of TCP depends to a certain degree on how efficiently it can detect and reduce network congestion. In general, short TCP flows are not capable of reacting to congestion. When a short flow with very few packets experiences loss(es), the congestion window is too small that reducing its size will have no impact on reducing congestion. On the other hand, the network congestion is alleviated when a source with a large congestion window detects loss. In this case, reducing the congestion window by halving the value of `cwnd` reduces the overall network load and therefore the congestion. Hence, reducing the congestion windows of a few flows with large congestion windows can be sufficient to react to network congestion.

LAS speeds up the slow start phase of all flows (including largest ones) and enables sources to reach the congestion avoidance (CA) phase faster. This improves the throughput performance of TCP as TCP flows under LAS, especially large flows, are likely to detect packet losses during the CA phase. In routers with FIFO scheduling, short flows are as likely as large flows to see their packets dropped, which deteriorates the throughput performance of short flows without alleviating congestion. Under LAS, when the queue is full, LAS first inserts an incoming packet at its position in the queue, and then drops the packet that is at the tail of the queue. It turns out that LAS very much protects flows from losing packets during the initial slow start phase and most likely drops packets from large flows. Apparently, speeding up the slow start phase and avoiding packet losses in this phase make LAS a very effective policy for short flows.

Figure 5 presents simulation results that illustrate the positive impact of LAS on the packet losses seen by short flows. The simulation was performed with parameters of Table 1 for Pareto distributed flow sizes. We observe that no short flow of size less than 40 packets experiences packet loss under LAS, whereas short flows of up to 40 packets make about 80% of the flows that experience packet losses under FIFO. In general, it is essentially the short flows that benefit from LAS in terms of significantly reduced packet losses. The overall loss rate under LAS is about 3.3% whereas under FIFO with droptail it is about 3.9%. The number of flows that experience one or more packet loss under FIFO is 12212, almost twice the number of flows that experience loss under LAS, which is 6763 out of about 47800 flows.

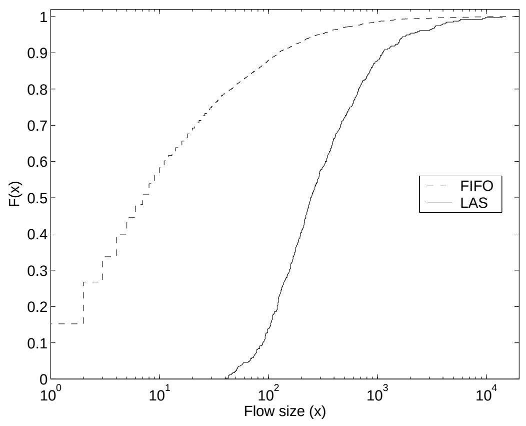


Figure 5: CDF of flows that experience loss for Pareto distributed Web flow sizes, load $\rho = 0.7$

6 Single long-lived flow competing with Web background traffic

We saw that the mean transmission times of the largest Web flows under LAS and under FIFO are quite similar. Similarly, we observed that the loss rates of the largest Web flows under LAS and FIFO do not differ too much. In this section we investigate the performance of long-lived TCP flows that are larger than the largest Web flows. One expects that a long-lived TCP flow, when competing against short TCP flows, must starve at some point as LAS favors short flows. However, simulation results show that this is true only at high load close to overload. If the Web traffic makes up for a high load, a long-lived TCP flow under LAS will experience starvation. This is clearly a drawback of LAS that can heavily affect mission-critical long-lived flows under high network load. To protect these flows from starvation, we propose a modified LAS policy called **LAS-log(k)** that provides a service differentiation for flows that require improved service. To illustrate this point, we consider a two-class policy with ordinary and priority flows. Assume a flow that has received x amount of service: If this flow belongs to the ordinary class, its service priority $P(x)$ under LAS-log(k) is x. If the flow belongs to the priority class, its service priority $P^H(x)$ is $(log_2(x))^{1/k}$, with $k > 0$. The packets of flows from the two classes are serviced just as under simple LAS policy based on their service priority. If $(log_2(x))^{1/k} < x$, (which is for instance the case for $k = 0.5$ and $x > 4$) a priority flow will receive priority service as compared to an ordinary flow that has received the same amount of service.

We studied the performance of a long-lived ftp flow under LAS and LAS-log(k) using a set-up where the long-lived flow competes against Web traffic across a bottleneck link. In case of LAS-log, the ftp flow will be in the priority class and all the Web flows in the ordinary class. All access links have a capacity of 10Mb/s except the access link for the ftp source, which has a capacity of 1Mb/s. Otherwise, when we study the performance under LAS-log, the fact that the ftp flow is in the priority class will allow this flow to hog all the bottleneck link bandwidth, which is not realistic.

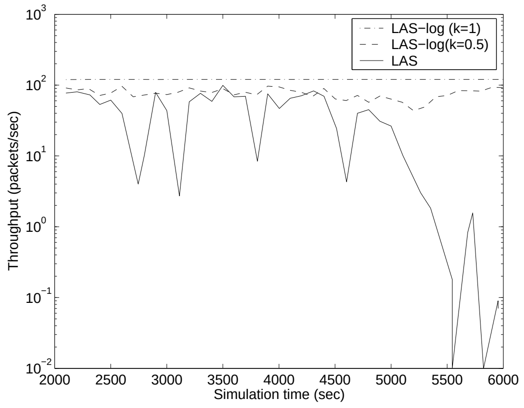


Figure 6: Throughput of the long-lived ftp flow under LAS and LAS-log (k): $k = 0.5$ and $k = 1$

Figure 6 compares the throughput of the long-lived ftp flow under plain LAS and LAS-log with $k = 0.5$ and $k = 1$ as a function of simulation time when the load due to Web traffic is about $\rho = 0.7$. We first note that the throughput under plain LAS is highly variable and decreases to zero after a long simulation time. In contrast, the LAS-log discipline improves the throughput of the flow. For $k = 0.5$, the ftp flow achieves an almost constant throughput of around 90 packets/sec. For $k = 1$, the performance of the ftp flow reaches about 120 packets/sec and does not deteriorate in time. Thus, we conclude that LAS-log improves the service of long-lived priority flows to the extent that these flows are not affected by the ordinary flows.

7 Implementation issues

LAS for link scheduling is non-preemptive and the packet to be served next will be taken from the head of the queue. To implement LAS link scheduling, we need a *priority assignment unit* whose functions include computing the service priority of each arriving packet and inserting the packet to its appropriate position in the priority queue. The priority assignment unit needs to track for each flow f the amount of bytes S_f served so far. S_f is used to compute the service priority of a packet of flow f, which is needed to determine where to insert this packet into the priority queue. Recall that the lower the value of the service priority, the higher the priority, and the closer to the head of the queue the packet is inserted. S_f is initialized with 0. When a packet of size P for flow f arrives, the priority assignment unit executes the following operations:

- `Use` S_f `to compute the service priority and insert packet into priority queue`
- `Update` S_f `to` $S_f := S_f + P$

There are a number of methods to implement a priority queue for high speed packet networks. The work in [2] presents a hardware implementation called *pipelined priority queue architecture* of a priority queue that scales well with respect to the number of priority levels and the queue size. In particular, the implementation can support data rates up to 10 Gb/s and over 4 billion priority levels.

Keeping per-flow state is known to be a difficult task in the core of the Internet due to the presence of a large number of simultaneous active flows. We do not envision to deploy LAS in the *core* of the Internet.

This is also not necessary, since the core of the Internet is not congested. Congestion in the Internet typically occurs at the edges of the Internet [10], where the number of flows simultaneously active is moderate. Thus, deploying LAS at the edges should be sufficient to reap the benefits of LAS in terms of reducing the response time of short TCP flows.

8 Conclusion

LAS scheduling for jobs has been known for decades, but it was never proposed before for packet networks. This paper argues that LAS could be deployed in packet networks at the bottleneck links. Using a realistic traffic scenario that captures the high variation of the flow sizes observed in the Internet, we saw that LAS deployed in the bottleneck link (i) significantly reduces the mean transmission time and loss rate of short TCP flows, (ii) does not starve long-lived TCP flows except at load values close to overload. The performance degradation under LAS seen by long TCP flows can be avoided if we use a modified version called LAS-log.

Other scheduling policies such as PS need to be deployed in every router along the path from the source to the destination. On the other hand, incremental deployment of LAS at a bottleneck router is effective as short flows that pass through that bottleneck will see their response time reduced. As LAS gets deployed in more bottleneck routers, more short flows will benefit.

There exist various proposals on how to improve the performance of short TCP flows by changing the mechanisms deployed in a router. However, our proposal has distinctive advantages: LAS is conceptually very simple as there is no parameter that needs to be tuned. The priority mechanism of LAS improves the transmission time of short flows much more than schemes that propose to use of two *separate FIFO* queues for short and long flows.

Acknowledgments

We would like to thank the reviewers and the editor-in-chief for their constructive comments.

References

[1] K. Avrachenkov, U. Ayesta, P. Brown, and N. Nyberg, "Differentiation between Short and Long TCP flows: Predictability of the response time", In *Proc. IEEE INFOCOM*, March 2004.

[2] R. Bhagwan and B. Lin, "Fast and Scalable Priority Queue Architecture for High-Speed Network Switches", In *Proc. IEEE INFOCOM*, pp. 538–547, 2000.

[3] X. Chen and J. Heidemann, "Preferential Treatment for Short Flows to Reduce Web Latency", *Computer Networks: International Journal of Computer and Telecommunication Networking*, 41(6):779–794, 2003.

[4] K. Claffy, G. Miller, and K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone", In *Proceedings of INET*, July 1998.

[5] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[6] E. J. Friedman and S. G. Henderson, "Fairness and Efficiency in Web Servers", In *Proc. ACM SIGMETRICS*, pp. 229–237, June 2003.

[7] L. Guo and I. Matta, "The War between Mice and Elephants", In *Proc. 9th IEEE International Conference on Network Protocols (ICNP)*, Riverside, CA, 2001.

[8] L. Guo and I. Matta, "Differentiated Control of Web Traffic: A Numerical Analysis", In *SPIE ITCOM'2002: Scalability and Traffic Control in IP Networks*, August 2002.

[9] L. Kleinrock, *Queuing Systems, Volume II: Computer Applications*, Wiley, New York, 1976.

[10] V. N. Padmanabhan, L. Qiu, and H. J. Wang, "Server-based Inference of Internet Link Lossiness", In *Proc. IEEE INFOCOM*, 2003.

[11] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approch to Flow Control In Intergated Services Networks-The Single Node Case", In *Proceedings of Infocom*, pp. 914–924, IEEE, 1992.

[12] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modelling", *IEEE/ACM Transactions on Networking*, 3:226–244, June 1995.

[13] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack, "Analysis of LAS Scheduling for Job Size Distributions with High Variance", In *Proc. ACM SIGMETRICS*, pp. 218–228, June 2003.

[14] J. Roberts, "Internet Traffic, QoS, and Pricing", *Proceedings of the IEEE*, 92(9):1389–1399, August 2004.

[15] C. Williamson and Q. Wu, "A Case for Context-Aware TCP/IP", *Performance Evaluation Review*, 29(4):11–23, March 2002.

[16] X. Xiao and L. Ni, "Internet QoS: A Big Picture", *IEEE Network*, pp. 8–18, March/April 1999.

[17] H. Zhang, "Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks", *IEEE Proceedings*, October 1995.