Least Attained Recent Service for Packet Scheduling over Wireless LANs

Martin Heusse*, Guillaume Urvoy-Keller†, Andrzej Duda* and Timothy X Brown ‡
* Grenoble Informatics Laboratory, Grenoble, France
† Eurecom, Nice, France
‡ University of Colorado, Boulder, USA

***Abstract*—Wireless LANs suffer from performance problems caused by insufficient medium access opportunity given to the access point. Consequently, the downlink buffer fills up, which often leads to packet losses. We propose to address this problem by using a size-based scheduling approach, which is known to favor short flows and the start up of new ones—a very appealing property from the user's perspective as interactive applications and new flows are serviced quickly. Still, size-based scheduling policies have a well-known Achilles heel: large flows can block each other for long periods of time and low rate multimedia transfers may end up with a low priority when their accumulated transferred volume becomes large. To solve the above deficiencies, we propose a new packet scheduling scheme called *Least Attained Recent Service* (LARS) that applies a temporal decay to the volume of data associated with each flow. In this way, its priority depends more on what has happened recently. With this strategy, LARS can bound the impact of a new arriving flow on ongoing flows, thus limiting lock out durations. It can also efficiently protect low rate multimedia transfers irrespectively of the load conditions.**

I. Introduction

In 802.11 networks in the infrastructure mode, all entities contending in the cell including the access point have equal access opportunities to the medium. However, in a typical usage, the access point requires a greater access share corresponding to its central role in the cell. Tackling this imbalance by implementing a suitable queueing policy at the network layer of the access point is appealing, because it is independent of the link layer so it can be easier to deploy in access points. Nevertheless, designing a adequate policy is challenging given the set of objectives in this type of access networks.

Performance problems in 802.11 wireless networks arise when traffic consists of TCP downloads during which the access point typically transmits two TCP segments per station while each of them sends back only one TCP acknowledgment. It also persists in more intricate cases in which traffic is a mix of uploads and downloads. At moderate loads, one can already observe jitter and high set-up times for new connections as the queue can temporarily build up at the access point. At high loads, the *TCP unfairness problem* [1] arises and TCP uploads hinder the progress of downloads. This is due to TCP data packets from downloads and TCP acknowledgments from uploads that are lost with the same probability at the access point buffer[1] and losing data packets is detrimental to TCP transfers whereas losing acknowledgments has little impact on TCP progress, because they are cumulative.

Many studies have tackled the above problem by proposing modification of the 802.11 MAC layer [2], [3], [4], [5], [6], [7]. In the present work, we take a stance of applying modifications to the IP layer only leaving the 802.11 protocol unchanged. In addition, the modifications we propose only affect the access point. We view the above problem as the management issue of IP packets over paths consisting of a single bottleneck (the access point). We investigate the use of size-based scheduling techniques at the bottleneck of the path [8], [9], [10], [11], [12], [13]. The core idea behind size-based scheduling policies is to give priority to the jobs[2] that have sent the least amount of data so far when the job size is not known in advance, e.g. at an access point, or the jobs closest to completion when the job size is known, e.g. at a Web server. In all cases, smaller jobs receive higher priority than larger ones, which is a desirable property in the networking context in which short jobs correspond to short flows generated by interactive applications (e.g. Web browsing, mail checking) whereas large jobs often correspond to bulk data transfers whose actual response time is of lesser importance to users (e.g. P2P transfers, system updates). In our WLAN context, the job size is not known in advance. In this situation, the most popular scheduling policy is the Least Attained Service (LAS) [8] that assigns to jobs a priority inversely proportional to the amount of service received so far.

Our starting point is LASTOTAL [12], an extension of LAS that offers equal opportunity to uploads and downloads in a WLAN configuration, i.e. uploads and downloads of the same size achieve similar response times under LASTOTAL. In addition, LASTOTAL offers similarly to LAS low response times to short flows and more generally to each flow in its early infancy, which results in small set-up times. However, LASTOTAL as LAS, bears a number of deficiencies that we aim at solving:

- LASTOTAL bases its decision on the total amount of bytes sent so far by a flow, which can lead to highly detrimental lock outs when a long flow interrupts other ongoing long flows. The extent of this lock out depends

[1]Buffers in network devices are generally managed in packets and not in bytes.

[2]Jobs are scheduled entities in the queueing theory. Depending on the situation, they can be flows in a networking context or objects in a content delivery context.

on the ability of the newly arriving flow to monopolize the bandwidth of the bottleneck, but leads in general to a high variance of the response time for large flows;

- LASTOTAL does not take time into account in its decision. As such, it similarly considers two flows f_1 and f_2 that have sent the same amount of bytes even though f_1 might have sent data at a very high rate, e.g. a high speed download using RapidShare, and f_2 might be a relatively low rate voice over IP conversation that started a long time ago.

To address the above problems, we propose the *Least Attained Recent Service* (LARS) policy. LARS applies a *temporal decay* to the volume of data associated with each flow so that its priority depends more on what has happened recently by forgetting about the attained service in the distant past. LARS has two configuration parameters. By choosing different values we can transform LARS into LAS [14], FIFO, or Fair Queueing [15].

In this paper, we evaluate the performance of LARS by means of a theoretical analysis and extensive simulations in WLAN configurations, and compare it with other scheduling disciplines. Our objectives are to:

- assess the ability of LARS to retain the most desirable property of LAS, i.e. a low response time of short flows.
- demonstrate that LARS indeed reduces lock outs among long flows. We use the variance of the response time to capture this effect.
- check that LARS efficiently protects low-rate multimedia flows, results in small jitter and in the absence of losses.

The paper is organized as follows. Section II presents the proposed scheduling scheme. The limit behavior of LARS and its impact on parameter setting is analyzed in Section III. Then, we use simulation to evaluate its performance in an 802.11 WLAN setting and compare with other scheduling schemes in Section IV. Section V overviews the related work. Section VI concludes the paper.

II. LARS, Least Attained Recent Service Scheduling

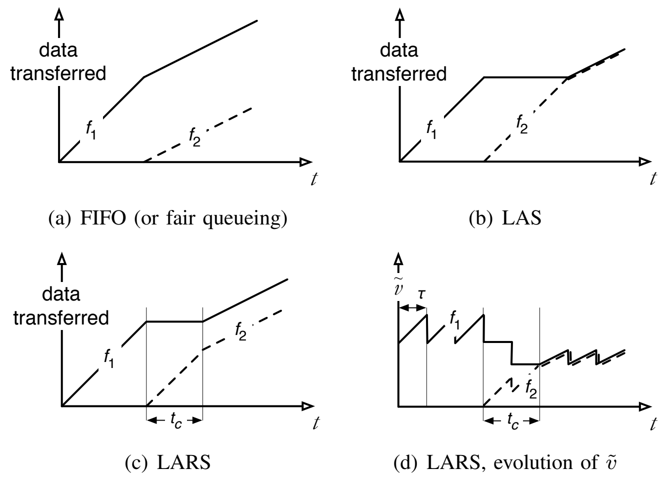
In this section, we first introduce the general principle of LARS and then describe the mechanism it borrows from LASTOTAL to handle flows in both directions in a typical WLAN setting.

A. Decayed Volume in LARS

LARS serves packets in the order that depends on the data volume transferred by ongoing flows. More specifically, it adds up the amount of data transferred by a flow (attained service) and uses it to set the priority of a packet in a queue. The scheduler serves packets in the order of the smallest volume of transferred data first. LARS periodically applies temporal decay to the data volume: every τ units of time, it multiplies it by factor $0 < \mu \leqslant 1$. In this way, the volume of data transferred by a given flow at some instant has less and less weight as time goes by. We denote the *decayed volume* by $\tilde{v}$. The decayed volume of flow j in decay period n is defined as follows:

$$\tilde{v}_j(t) = S_j(t) - S_j(n\tau) + \mu\ \tilde{v}_j(n\tau),\ \forall t \in [n\tau, (n+1)\tau[$$

where $S_j(t)$ is the volume of data transferred up to time t by flow j. In other words, $\tilde{v}$ is equal to the total traffic that arrived since the end of the last decay period plus the decayed value of $\tilde{v}$ at the end of the last decay period. The *priority* of flow j at time t decreases with increasing $\tilde{v}_j(t)$: the next packet to send belongs to the flow with the *lowest* decayed volume.



(a) FIFO (or fair queueing)
(b) LAS
(c) LARS
(d) LARS, evolution of $\tilde{v}$

Fig. 1. Behavior of FIFO, LAS, and LARS when a new flow starts. FIFO and Fair Queueing behave similarly if the flow arrival rates are the same.

Figure 1(c) presents the behavior of LARS when a new flow starts sending data. A new flow under LARS will start with a decayed volume of zero and thus with the highest priority. If its rate is greater or equal to the link capacity, it initially benefits from the whole capacity of the link. As the volume of data sent increases and $\tilde{v}$ of other flows decays (by a factor of μ every τ units of time), eventually its $\tilde{v}$ will catch up with the decayed volume of other flows and the scheduler will split the capacity of the link among the flows.

LARS has two parameters: period τ of applying temporal decay and decay factor μ. Their values determine the behavior of the scheduling scheme ranging from LAS to FIFO. Indeed, as $\mu \to 1$, the decayed volume of a flow becomes equal to its absolute volume, which means that LARS becomes LAS [14]. When μ goes to zero, then LARS behaves like the Deficit Round Robin discipline [16] in which the deficit of each flow is reset to zero at the end of each decay period. When both μ and τ go to zero, decayed volume $\tilde{v}$ of all flows becomes 0, which means that all packets will have the same priority so LARS degenerates into FIFO. We discuss the details of parameter tuning in the following sections.

Note that the units of $\tilde{v}$ are the same as the units of variable $S(t)$. These units can be chosen at our convenience or according to the need for accuracy. In this paper, we count $S(t)$ in bytes of the IP packet.

B. *LARS for bi-directional traffic control*

An interesting feature of LAS, which led to the definition of LASTOTAL, is that we can easily extend it to take into account both directions of flows over half-duplex links such as 802.11 WLANs. LARS borrows this feature from LASTOTAL that we detail below.

Let us first note that to offer similar performance to uploads and downloads, LARS, which is deployed at the downlink queue of the access point, can control both directions of flows only by adequately scheduling the packets going from the access point to the wireless stations. This is possible with TCP transfers that use a feedback loop consisting of a TCP acknowledgment stream. So, even a pure TCP upload (from a wireless station) can be indirectly controlled through an appropriate scheduling of its TCP level acknowledgments. This is not the case for UDP transfers that do not use any feedback loop. So LARS will be able to control all TCP transfers and all UDP downloads. Performance of UDP uploads is only a function of the 802.11 MAC layer. Still, in general TCP traffic represents the majority of transferred data and also the majority of flows. In Section IV-C, we further discuss the performance of low rate UDP real time traffic in the down direction.

The main idea behind LASTOTAL, which LARS capitalizes on, is to look up the ACK number in the TCP header and add its progress since the previous segment to the virtual volume $\tilde{v}$ of the corresponding connections. So the volume $\tilde{v}$ of a TCP connection takes into account its utilization of the link in both directions.

C. *LARS Algorithm*

To control flows, LARS keeps the following information for each flow:

- the source and destination IP addresses, IP header protocol field and ports if applicable,
- the last ACK number seen, ACK flag of previous packet,
- the instant at which the last packet was seen for garbage collection,
- current $\tilde{v}$.

In both LASTOTAL and LARS, to effectively favor low-volume connections during overload conditions, i.e. when the buffer is full, one must first insert the new packet in the priority queue and discard the one at the end, i.e. the one with the largest $\tilde{v}$ value. This buffer management policy differs from the legacy drop tail policy in which incoming packets are discarded when the queue is full. So we mark each packet in the queue with the volume of its flow when the packet leaves the buffer. In this way, packets are kept in the ascending order of the volume and there is no need for multiple queues nor for separating responsive and non responsive traffic: if a flow sends more packets than its share, its packets are simply dropped before they are inserted in the queue or they are pushed back out by packets of other flows. When the scheduler discards the packet, it updates the corresponding flow descriptor, so that the next packet bears the proper volume. Dequeuing is trivial.

The storage requirement for LARS is linear with the number of concurrent flows, like all per-flow queueing schemes. The scheduler can identify the flow descriptor for an incoming packet by using a hash function or maintaining an ordered set. Finally, the complexity of inserting the packet in the queue is also of the order of the log of the maximum number of packets in the queue.

III. Limit Behavior of LARS and Parameters Setting

In this section, we first analyze the limit behavior of LARS for a long-lived (infinite) flow. We next derive the maximum theoretical lock-out[3] duration of a long-lived flow caused by another long-lived flow. The case in which a newly arriving flow is able to monopolize the link bandwidth provides a key insight for choosing the parameters of LARS. We also discuss the case when an ongoing flow is a low rate multimedia flow.

A. *Steady-State Behavior of the Decayed Volume*

Let us assume a flow sending at a constant rate of x bytes per second. We analyze its $\tilde{v}$ value at the end of decay periods. During each decay period it will send $x\tau$ bytes and its $\tilde{v}^{(n)}$ at the end of decay period n is the following:

$$\begin{aligned}\tilde{v}(n\tau) = \tilde{v}^{(n)} &= x\tau + \mu\tilde{v}^{(n-1)} \\ &= \sum_{i=0}^{n-1} x\tau\mu^i = x\tau\frac{1-\mu^n}{1-\mu}\end{aligned} \tag{1}$$

$\tilde{v}^{(n)}$ is an increasing function of n. The limit as n goes to infinity is:

$$\tilde{v}^{\infty} = \frac{x\tau}{1-\mu}. \tag{2}$$

B. *Catch-up Interval*

When a new flow starts sending data, it can capture link capacity during a catch-up interval before other existing flows resume transmitting. The maximum catch-up time t_c, is a measure of the disruption time caused by the new flow. In LAS, this interval may be as long as the time of residence of an older flow, whereas in LARS it is limited: when a new flow, f_2, blocks an older one, f_1, the decayed volume $\tilde{v}$ of f_1 will gradually decrease while that of f_2 grows, so that their decayed volumes become equal and the flows start sharing link capacity (cf. Figure 1(d)). LARS can theoretically behave as close to fair queueing as desired simply by reducing time t_c during which full priority is given to the entering flow.

To find the maximum catch-up interval t_c, let us consider a long-lived flow f_1, one of several other flows existing in the system for a long time. If its rate is a steady x, Eq. 2 gives the limit of the decayed volume $\tilde{v}^{\infty}$. Assume that at time $t = 0$ a new flow, f_2, able to saturate the link starts sending data, joins the system. In this case, its initial flow rate will be equal to link capacity C and its decayed volume $\tilde{v}$ will increase according to Eq. 1:

$$\tilde{v}_2^{(n)} = C\tau\,\frac{1-\mu^{(n)}}{1-\mu}.$$

[3]We call also catch-up time from the newly arriving flow perspective.

The old flow f_1 will stop sending, so its decayed volume becomes after n consecutive periods of duration τ:

$$\tilde{v}_1^{(n)} = \tilde{v}^{\infty}\ \mu^n = \frac{x\tau\mu^n}{1-\mu}.$$

Decayed volumes $\tilde{v}$ of both flows become equal after time:

$$t_{\rm c}(x) = \tau\ \frac{\ln(x/C+1)}{-\ln(\mu)} \leqslant \frac{\tau x/C}{1-\mu} \qquad (3)$$

This expression is maximized when $x = C$ and we define the maximal catch-up interval as $t_{\rm c}^{\infty} = t_{\rm c}(C)$. Interestingly, $t_{\rm c}^{\infty}$ depends only on τ and μ, and is independent of link capacity C. This is of particular interest for wireless networks or any random access network in which the estimation of the available throughput is difficult. We can summarize the above results as follows:

For two large flows that enter the network at different times, $t_{\rm c}^{\infty}$ bounds the impact of the priority given to the new flow on f_1.

In a fair scheduling scheme two flows would immediately obtain equal capacity shares (cf. Fig. 1(a)). For $\mu = 1$, we can see that $t_{\rm c} = \infty$, because LARS degenerates to LAS and LAS would indefinitely delay f_1 until the new flow catches up (cf. Fig. 1(b)).

We adopt the values of $\mu = 0.99$ and decay period $\tau =$ 10ms in simulations presented in the next section. This leads to maximal catch-up interval $t_{\rm c}^{\infty} = 690$ms. Note again that the maximal catch-up interval is smaller if either the new flow is not able to saturate the link or if multiple flows are active. If we consider the case of TCP traffic, this value of $t_{\rm c}^{\infty}$ has the advantage of being less than one second, which is the smallest recommended timeout value for TCP [17]. Clearly, the choice of τ influences the frequency at which the LARS scheduler computes decayed volumes. Even though current networking devices such as 802.11 access points are not as powerful as hosts in terms of processing power, we believe that re-evaluating the volume of active flows every 10ms is a reasonable choice.

C. *Impact on delay sensitive traffic*

In the previous section, we have considered the maximal interval during which LARS can block a flow. In this section, we analyze the effect of the arrival of a new flow on an ongoing delay sensitive transfer. For the sake of simplicity, we assume a CBR delay sensitive flow.

As LARS gives a high priority to new flows, this may introduce delay between consecutive packets of the CBR source. This delay is unbounded with LAS, while in LARS it is less than $t_{\rm c}^{\infty}$. However, the smaller throughput x_0 of the CBR flow, the shorter the time it will take for a new flow to catch up with its $\tilde{v}$ (cf. Eq. 3) so that LARS approximates the behavior of a fair queueing scheduler.

For instance, over an 802.11b network, a high-quality H.264 encoded video at 384kb/s uses less than 7% of the resources and it will not be interrupted for more than 50ms.

IV. Evaluation

In this section, we evaluate the performance of LARS for an 802.11 WLAN operating at the nominal bit rate of 54Mb/s. We have used the Qualnet tool [18] to simulate an 802.11 wireless network in the infrastructure mode with several stations under good radio transmission conditions (stations are within 10 m of the access point).

A. *Long-Lived Flows*

At first, we ran a simulation that involves TCP uploads and downloads as well as intense UDP traffic. Each wireless station is a target or a source of a single flow and there is no traffic between wireless stations.

Figure 2 presents a comparison of various scheduling disciplines: FIFO, SCFQ, LASTOTAL, and LARS. SCFQ (Self-Clocked Fair Queueing) is a low overhead fair queueing scheme [19].

The workload varies over time with flows progressively beginning to send data: five TCP downloads start one every other second from 10s to 18s, five TCP uploads start from 30s to 38s, and two high rate UDP downloads generate one MTU packet every 0.5ms during the periods from 22s to 26s and from 42s to 46s.

We can observe that under FIFO scheduling uploads suffer less from the presence of UDP traffic than TCP downloads (consider for instance Figure 2(a) between times 42 and 46 where all five downloads are blocked). These results for FIFO corroborate the observations of Pilosof et al. [1]. We can also notice that the progress of connections is extremely variable (cf. Figure 2(a)). Moreover, UDP downloads have strong impact on the throughput of TCP downloads. SCFQ improves the overall behavior, however it fails at solving the unfairness problem. Interestingly, the second UDP download is also shut down with SCFQ, as the limited downlink bandwidth is evenly split between UDP, TCP downloads, and upload (TCP) ACKs.

Figure 2(c) illustrates the infinite memory effect of LASTOTAL: new incoming connections block older ones, in some cases for nearly 30 seconds of the 60 second simulation. Once newer flows have transferred as much data as earlier connections the latter ones do not restart right away, because the TCP senders now use very large retransmission values. We can observe in Figure 2(d) the beneficial effect of temporal decay—LARS gives higher short term priority to new flows, while sharing the available capacity on a long term scale in a fair manner. We observe that all ten TCP flows advance at approximately the same rate.

B. *Realistic Synthetic Workload*

To evaluate the benefits of LARS compared to LASTOTAL, FIFO, and SCFQ we have considered a realistic synthetic workload consisting of bulk TCP transfers of varying size. In this simulation experiment, ten wireless stations communicate with ten servers in the wired part of the network across an access point.

TCP connections use 1460 bytes MSS and arrive according to a Poisson process with a rate chosen so that the total

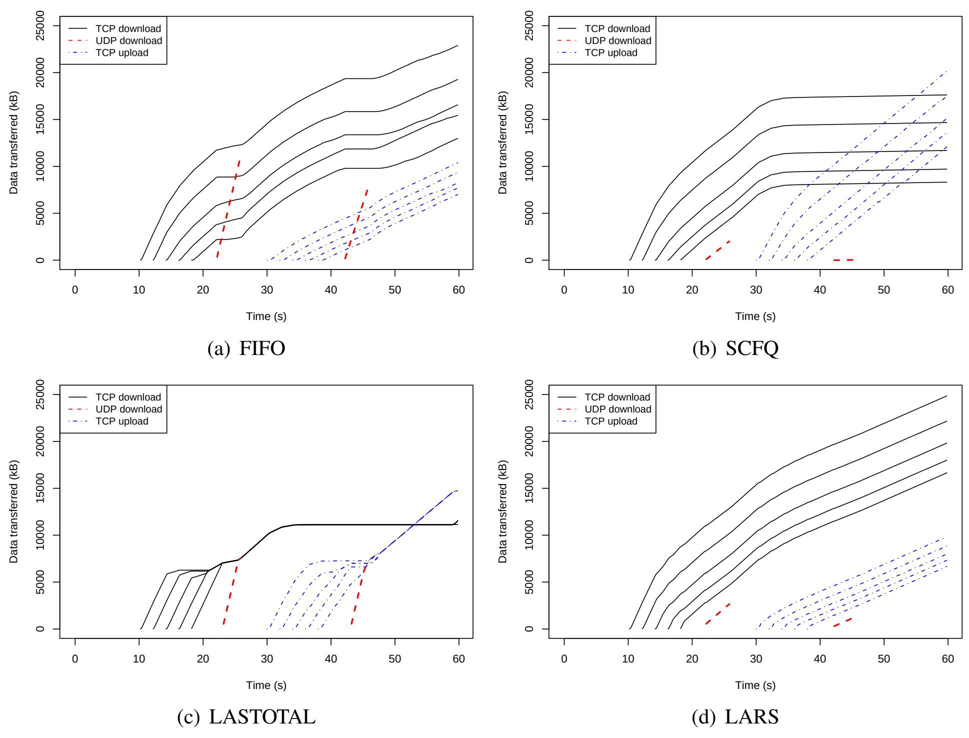


Fig. 2. Long-lived flows incoming into the network: five TCP downloads start from 10s to 18s, one every other second, five TCP uploads start from 30s to 38s, and two high rate UDP downloads last from 22s to 26s and from 42s to 46s.

generated throughput is either 10Mb/s (moderate load) or 20Mb/s (high load)—recall that a single TCP connection can achieve 24.3Mb/s goodput over 54Mb/s 802.11a.

We draw the number of MSS packets, i.e. the volume of data to transfer on a single TCP connection from a bounded Zipf distribution, which is a discrete equivalent of a continuous (bounded) Pareto distribution. This distribution has the following parameters: the minimum transfer size is 6 MSS, the maximum transfer volume corresponds to 10MB with a coefficient of variation (CoV[4]) of about 6. The latter controls how the mass of the distribution (the total amount of bytes) is split between short and long transfers. We have chosen the value of 6, which is in line with actual measurements performed on large WLANs [20]. Such parameters result in an average connection volume of about 60KB. Two third of connections are downloads and we consider 5000s of channel activity.

The objective of this experiment is to test the extent to which LARS has the same beneficial effect as LASTOTAL by granting short flows low response times while limiting lock-outs of large flows.

We present results for short and large flows each time, where short flows are defined as flows smaller than the 95-th quantile of the flow size distribution. This rather high cutoff point conforms with the heavy-tailed distribution of flow size, as with our definition and choice of distribution, approximately 60% of the mass is carried by large flows.

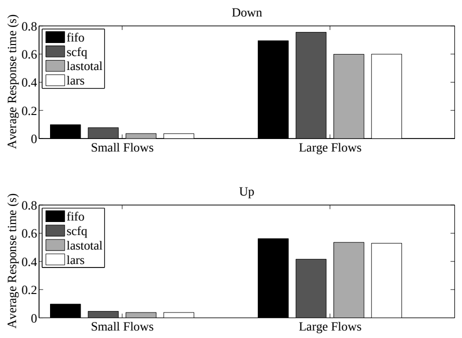


Fig. 3. Average response time, realistic synthetic workload of 10Mb/s

Figures 3 and 4 present the average TCP connection response times (the time to finish a connection) for all scheduling disciplines. We first observe that LARS and LASTOTAL obtain similar fairly short response times for short and also large flows in both directions. For the 10Mb/s workload, the response time of short flows under LASTOTAL and LARS is smaller than under FIFO and SCFQ, while for large flows they are comparable. At 20 Mb/s, LASTOTAL and LARS keep obtaining significantly smaller response times for short

[4]The CoV is formally defined as the ratio of the standard deviation to the mean of a distribution.

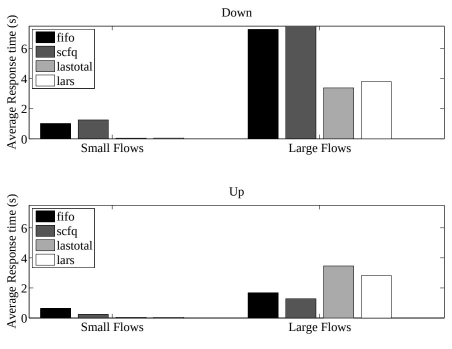


Fig. 4. Average response time, realistic synthetic workload of 20Mb/s

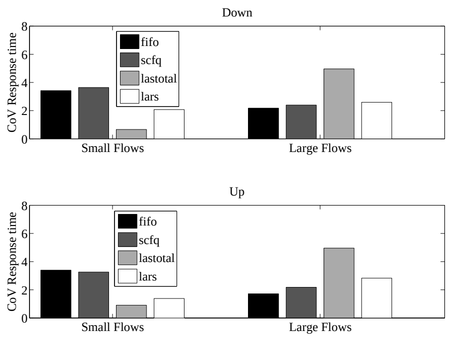


Fig. 6. CoV of response time, realistic synthetic workload of 20Mb/s

flows. Note the response time of short downloads under FIFO and SCFQ jumped to values close to 1 second, which can be detrimental for user perception. The price paid when using LARS and LASTOTAL, which throttle uploads while FIFO and SCFQ do not behave like this, is a marginal increase for the response time of large uploads. The gain is of at least one order of magnitude for downloads and is a less marked one for small uploads.

The behavior of SCFQ appears more extreme than FIFO when the load increases, because SCFQ tries to equally share the capacity of the access point between competing flows, which consist here of data segments in downloads and of ACKs in uploads. The latter being smaller in size, SCFQ serves them quicker, which results in smaller response times for uploads. When load becomes high at 20 Mb/s, the performance of SCFQ deteriorates as it tries to share capacity between an ever increasing number of flows. As a consequence, the response time skyrockets for large flows: the actual response time of large downloads under SCFQ is 21s (we deliberately cut the SCFQ bar off to better observe the difference between the analyzed policies in Figure 4).

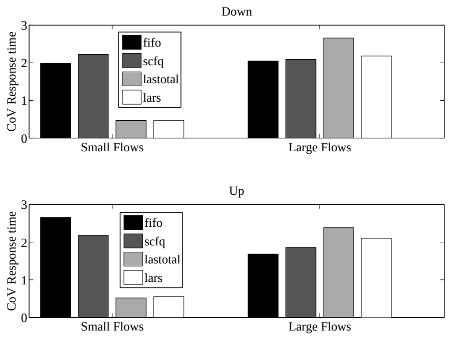


Fig. 5. CoV of response time, realistic synthetic workload of 10Mb/s

In addition to obtaining similar response times to LASTOTAL, LARS also tries to limit lockout times of large flows. We can observe this effect in our simulations by measuring the variability of the response time for large flows under LARS as compared to LASTOTAL. We use the coefficient of variation as a metric to assess variability in Figures 5 and 6 for respective loads of 10 and 20 Mb/s. We can observe that for large flows, LARS obtains smaller CoVs than LASTOTAL under all load conditions. For short flows, the effect is reversed—LASTOTAL gives full priority to short flows, which lowers both their response times and their variation. As LARS dedicates more resources to larger flows than LASTOTAL (depending on the values of the decayed volume), short flows experience higher, but still small, variability in the response times.

SCFQ and FIFO obtain CoVs in line with the applied strategy: SCFQ tends to similarly serve all downlink flows, both short and large flows obtain similar CoVs especially when the load increases. Under FIFO, on the other hand, only large flows tend to obtain small CoVs since FIFO tends to favor large flows at the expense of short ones.

The main conclusion of this section is that under realistic traffic conditions, LARS exhibits similar performance as LASTOTAL—it favors short flows at the expense of large flows, while limiting lockouts of large flows.

C. Multimedia Traffic

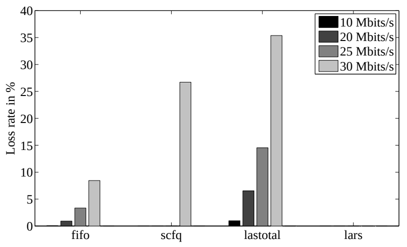


Fig. 7. Loss rate experienced by a CBR flow for various background loads

As shown in Section III-C LARS gives high priority to packets of long lasting low intensity flows. To evaluate this scenario, we have added a single CBR flow representing a typical VoIP stream to the random traffic defined in the previous section (TCP transfers distributed according to the Zipf law). We have also considered another workload case: overload conditions with 25Mb/s and 30Mb/s throughput in addition to 10Mb/s and 20Mb/s. The CBR flow generates 92 bytes every 10ms (64kb/s compressed voice with RTP overhead). The infinite memory of LASTOTAL quickly leads to discarding packets from this long flow as soon as the network becomes congested (cf. Figure 7). Conversely, LARS efficiently protects this flow, because there are no losses.

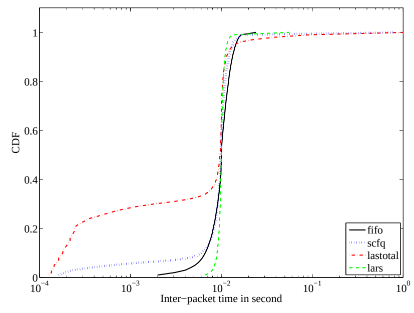


Fig. 8. *CDF* of inter-departure times for a CBR flow with 20Mb/s background traffic

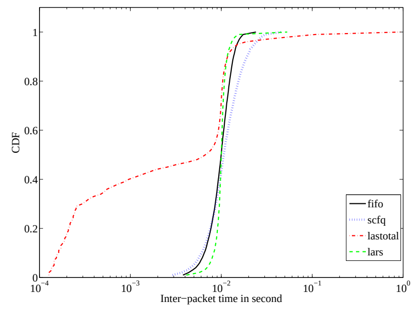


Fig. 9. *CDF* of inter-departure times for a CBR flow with 30Mb/s background traffic

To analyze further this behavior, we can look at the inter-departure time distribution presented in Figure 8 (cumulative distribution function *CDF*). We can observe that LASTOTAL releases (and drops) packets in batches (many packets have short delay between them), while LARS forwards them in a much more regular way with respect to the presence of background traffic. The jitter grows slightly under LARS only when the load significantly exceeds link capacity (cf. Figure 9). Moreover, the comparison with FIFO and SCFQ should take into account the fact that there are no packets lost under LARS. Thus, LARS can successfully protect long lived delay sensitive flows under any load condition.

V. Related Work

FIFO scheduling combined with the droptail buffer management policy has proven to be ineffective at high load in a number of cases. Suter et al. investigated the use of a per flow scheduling scheme (Fair Queuing) and various buffer management policies (Longest Queue Drop, Random, or Random Early Discard (RED)) to actually provide per flow isolation in high speed routers [15]. They concluded that scheduling alone is not sufficient to provide isolation from misbehaving flows. This is in line with our observations using SCFQ in Section IV.

Several authors put forward the idea of favoring short flows in TCP/IP networks using a two queue approach or using a marking scheme to differentiate packets from small and large flows. Guo et al. proposed an alternative use of the DiffServ architecture in which packets are marked at the edge of the network depending on the amount of bytes sent so far by the corresponding connection [21]. Active Queue Management (RIO - Red with In and Out) is used in the core router to give preferential treatment to short flows when the load increases. Nourredine et al. proposed a similar approach [22] based on different packet marking schemes combined with packet scheduling (Weighted Round Robin) and dropping policies (RIO) to enforce low response time to short connections.

Avrachenkov *et al.* have proposed Run2C [23], which aims at keeping the good features of LAS while avoiding its shortcomings. Run2C uses two Processor Sharing (PS) schedulers with different priorities. When a new flow appears, a high priority scheduler takes care of its packets. Once the flow has transmitted a given amount of data, it moves to a low priority queue. A fixed threshold is chosen to declare a job as short or large. Run2C is simpler to analyze than LARS and the authors derive its mean response time as a function of the traffic distribution. However, similarly to LAS, Run2C only takes into account the global volume of a connection. Consequently, long lived low throughput flows end up being handled by the low priority queue under Run2C. Compared to LARS, we can observe that, as more and more flows enter the network, their impact on other flows under LARS will decrease linearly. Under Run2C, all flows benefit from the same amount of data to send with a higher priority regardless of the link state at this time.

Sun et al. proposed a two class mechanism to differentiate packets belonging to short and large flows similarly to the Run2C approach [24]. They use a fixed threshold to classify flows. The Weighted Deficit Round Robin (WDRR) policy handles packets with a weight that should be set in proportion to the relative share of small and large flows. Similarly to Run2C, the proposed mechanisms accounts for the total volume of a connection without taking into account its history.

More complex (than a two queue approach) size-based scheduling approaches have also been proposed to favor short flows at the expense of long ones. LAS, which does not require to know the job size in advance, has been investigated in the context of 3G networks [11]. LAS has been also proposed to schedule flows at the bottleneck of wired links consisting of symmetric and full duplex links [10]. Another popular size-based scheduling policy, which can be used when the job size is known in advance, is *Shortest Remaining Processing Time* (SRPT). SRPT schedules the job that is closest to completion. Using this strategy, SRPT is known to be optimal (in terms of minimizing the average job response time) among all scheduling policies. SRPT has been proposed in the context of Web servers [9], [13].

To the best of our knowledge, only a single study has proposed to modify the way volumes are computed in LAS [25]. Many variants of LAS were proposed in which the priority index (equivalent of the decayed volume in LARS) is a linear or a logarithmic function of the amount of bytes sent by a flow. This technique can be used to offer different quality of service (through different functions) to various classes of flows. However, unlike LARS, it does not take the actual transfer rates into account.

VI. Conclusion

In this paper, we have proposed a new packet scheduling scheme called *Least Attained Recent Service* (LARS) that schedules packets by giving higher priority to packets of flows with less transferred traffic so far. In contrast to LAS, LARS regularly decays the volume associated with each flow.

With this strategy, LARS can bound the impact of a new flow on ongoing ones and accounts not only for the volumes, but also the rates of flows. These are key properties to avoid lock outs, which is a classical weakness of size-based scheduling approaches, and also to protect low rate multimedia flows, e.g. VoIP calls. Moreover, LARS still inherits from LAS its ability to offer low response times to short flows and more generally to all flows in their infancy. The proposed discipline only depends on two parameters that are independent of link characteristics, which makes LARS particularly well suited for fluctuating capacity or shared links like WLANs. LARS is not more complex than any other fair scheduling discipline that requires per flow bookkeeping. We believe that the progress of edge devices processing capabilities make it relevant to consider such advanced scheduling techniques.

Future directions of research on LARS could be to assign various weights to flows and possibly use various parameter settings among flows. We also believe that it is possible to bound the complexity of LARS with a low impact on the resulting performance.

References

[1] S. Pilosof, R Ramjee, D. Raz, Y Shavitt, and P. Sinha. Understanding TCP Fairness over Wireless LAN. In *Proc. of IEEE INFOCOM 2003*, volume 2, pages 863–872, March-April 2003.

[2] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, and M. Sooriyabandara. TCP performance implications of network path asymmetry. RFC 3449, Internet Engineering Task Force, December 2002.

[3] D.J. Leith, P. Clifford, D. Malone, and A. Ng. TCP Fairness in 802.11e WLANs. *IEEE Communications Letters*, 9(11), November 2005.

[4] X. Wang and S. A. Mujtaba. Performance Enhancement of 802.11 Wireless LAN for Asymmetric Traffic Using an Adaptive MAC Layer Protocol. In *Proc. of IEEE VTC-Fall 2002*, volume 2, September 2002.

[5] M. Bottigliengo, C. Casetti, CF. Chiasserini, and M. Meo. Smart Traffic Scheduling in 802.11 WLANs with Access Point. In *Proc. of IEEE VTC-Fall 2003*, volume 4, October 2003.

[6] A. Banchs, A. Azcorra, C. Garcia, and R. Cuevas. Applications and Challenges of the 802.11e EDCA mechanism: an experimental study. *IEEE Network*, 19(4):54–58, July/August 2005.

[7] E. Lopez-Aguilera, M. Heusse, Y. Grunenberger, F. Rousseau, A. Duda, and J. Casademont. An Asymmetric Access Point for Solving the Unfairness Problem in WLANs. *IEEE Transactions on Mobile Computing*, 7(10), October 2008.

[8] Misja Nuyens and Adam Wierman. The foreground-background queue: A survey. *Perform. Eval.*, 65(3-4), 2008.

[9] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.*, 21(2), 2003.

[10] Idris A. Rai, Guillaume Urvoy-Keller, and Ernst W. Biersack. Analysis of las scheduling for job size distributions with high variance. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 218–228, 2003.

[11] Pasi Lassila and Samuli Aalto. Combining opportunistic and size-based scheduling in wireless systems. In *MSWiM '08: Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 323–332, New York, NY, USA, 2008. ACM.

[12] Guillaume U. Keller and André L. Beylot. Improving flow level fairness and interactivity in wlans using size-based scheduling policies. In *MSWiM '08: Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 333–340, New York, NY, USA, 2008. ACM.

[13] Mingwei Gong and C. Williamson. Simulation evaluation of hybrid srpt scheduling policies. pages 355–363, 2004.

[14] Idris A Rai, Guillaume Urvoy Keller, and Ernst W Biersack. Analysis of LAS scheduling for job size distributions with high variance. In *ACM SIGMETRICS 2003, San Diego, USA*, Jun 2003.

[15] B. Suter, T. V. Lakshman, D. Stiliadis, and A. K. Choudhury. Buffer management schemes for supporting tcp in gigabit routers with per-flow queueing. *IEEE Journal on Selected Areas in Communications*, 17, 1999.

[16] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. *SIGCOMM Comput. Commun. Rev.*, 25(4), 1995.

[17] V. Paxson and M. Allman. Computing TCP's retransmission timer. RFC RFC 2988, Internet Engineering Task Force, November 2000.

[18] Qualnet 4.5. Scalable Networks.

[19] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. *IEEE INFOCOM '94; Networking for Global Communications*, pages 636–646 vol.2, Jun 1994.

[20] Xiaoqiao (George) Meng, Starsky H. Y. Wong, Yuan Yuan, and Songwu Lu. Characterizing Flows in Large Wireless Data Networks. In *Proc. of MobiCom '04*, 2004.

[21] Liang Guo and Ibrahim Matta. The war between mice and elephants. In *ICNP '01: Proceedings of the Ninth International Conference on Network Protocols*, Washington, DC, USA, 2001. IEEE Computer Society.

[22] Waél Noureddine and Fouad Tobagi. Improving the performance of interactive TCP applications using service differentiation. *Comput. Netw.*, 40(1), 2002.

[23] Konstantin Avrachenkov, Patrick Brown, and Eeva Nyberg. Differentiation between short and long TCP flows: Predictability of the response time. In *In Proc. IEEE INFOCOM*, 2004.

[24] Changhua Sun, Lei Shi, Chengchen Hu, and Bin Liu. Drr-sff: A practical scheduling algorithm to improve the performance of short flows. In *ICNS '07: Proceedings of the Third International Conference on Networking and Services*, Washington, DC, USA, 2007. IEEE Computer Society.

[25] I. A. Rai, G. Urvoy-Keller, M. Vernon, and E. W. Biersack. Performance models for las-based scheduling disciplines in a packet switched network. In *ACM SIGMETRICS-Performance*, June 2004.