Analysis of the Early Flow Discard (EFD) Discipline in 802.11 Wireless LANs

Jinbang Chen
Eurecom, France
Jinbang.Chen@eurecom.fr

Martin Heusse
LIG CNRS UMR 5217, France
Martin.Heusse@imag.fr

Guillaume Urvoy-Keller
I3S CNRS UMR 6070, France
Guillaume.Urvoy-Keller@unice.fr

***Abstract*—Size-based scheduling improves data transfer response times by favoring flows at an early stage. Although appealing, these techniques raise concerns as they require to keep track of the volume of data sent by each and every ongoing connections and they may starve long-lived flows even if they use up limited bandwidth. Early Flow Discard (EFD) scheduling addresses these issues and we present its adaptation to infrastructure 802.11 networks where the access point downlink queue naturally builds up. To deal with this problem, EFD needs to take into account bi-directional traffic, so that it effectively controls uploads and downloads even though EFD applies to the downlink buffer only. It appears that even with limited buffers, which translates into limited memory of flows for EFD, the most simple flavor of bidirectional EFD –a simple pair of FIFO queues and tracking flow transferred volumes with a packet granularity– enables to rip the full benefit of size-based scheduling, without any of the aforementioned drawbacks.**

***Index Terms*—Performance, EFD, Wireless LAN.**

I. Introduction

We consider the typical infrastructure-based WLAN where mobile stations equipped with 802.11 interface communicate with an Access Point (AP) on a wireless channel and the AP relays traffic to and from the wired network. In many cases, the wireless LAN is the performance bottleneck, *e.g.* companies or labs frequently use access links to the Internet with 100 Mbit/s or higher capacity.

The TCP transport protocol is used for controlling the vast majority of data transfers in volume (bytes sent) and the majority of flows. When TCP traffic is relayed over an 802.11 network, a key performance problem, known as “TCP Unfairness”, occurs. It happens when the downloads data packets, from the wired network, and TCP level acknowledgments from the uploads compete to access the access point downlink buffer. The buffer at the access point tends to fill up because the Distributed Coordination Function (DCF) at the MAC layer does not grant enough priority to the Access Point as compared to the other stations in the cell [14]. Several solutions have been investigated at different levels of the protocol stack (MAC, IP, Transport) to address the TCP unfairness problem [2], [11], [7], [18] .

In this paper, we consider solutions at the IP level (leaving the MAC layer unchanged), based on size-based scheduling. Size-based scheduling is a specific type of priority scheduling family where the priority of a flow is a function of the amount of service it has received so far. The Least Attained Service (LAS) policy implements this approach in a straightforward manner by granting full priority to the flow that has received the least amount of service so far [9].The key idea when size-based scheduling is applied in a networking context is to give priority to short transfers that are, for most of them, due to interactive applications like Web browsing or mail checking. Several size-based scheduling solutions have been proposed and analyzed to improve the performance of short transfers [1], [16] and more recently to solve the TCP unfairness problem, e.g, LASACK [18], which is a variant of LAS for 802.11 networks.

In spite of the appealing property of improving end user interactivity, size-based scheduling suffers from a number of shortcomings. In particular, it is a potential threat to long transfers which can face starvation; it is originally unable to account for the rate of transfers – a long transfer might be long only because it has been active for a long time, even though at a small rate – and finally, size-based scheduling requires to keep one state for each active flow. Some size-based scheduling policies have addressed some of these issues, *e.g.*, starvation is addressed by Run2C [1], accounting for rates and accumulated volume simultaneously is addressed by LARS [8]. LARS is a variant of the original LAS that applies a temporal decay to the accumulated volume of service received by a flow, thus accounting simultaneously for volumes and rates.

To the best of our knowledge, only EFD [3] manages to tackle all aforementioned concerns. EFD uses two virtual queues, one for the high priority packets and one for the low priorty packets. Each queue is serviced in a FIFO manner while the low priority queue is drained only if the high priority queue is empty. This is similar to the Run2C proposal. However, a key distinction between EFD and Run2C is the flow management. Run2C needs to keep one state per flow[1]. The flow states are efficiently managed in EFD by dropping flow records from the flow table as soon as the last packet of a flow in the flow table leaves the queue. In [3], EFD is investigated in wired network and using some pretty large buffer of 300 packets. In this paper, we investigate the performance of EFD (Early Flow Discard) policy in 802.11 networks, where buffer sizes tend to be smaller as they typically range between 30

[1]In the original Run2C paper, the authors propose a smart modification of the algorithm to assign the initial sequence number of TCP to have a stateless scheduler. Modifying all TCP stacks is however difficult at the Internet scale.

and 100 packets.

Our contributions are as follows:

- We propose two adaptations of EFD in WLAN networks, EFDACK and PEFD, that aim at mitigating the TCP unfairness problem. EFDACK keeps track of the amount of bytes sent by each flow in both the upload and download directions, which requires reading TCP segments (the acknowledgment number field) within IP packets. This is the same idea as the one of LASACK [18]. In contrast, PEFD keeps track of the number of packets and does not distinghuish between uploads and downloads.
- We compare EFDACK and PEFD to state-of-the-art size scheduling policies, Run2C, LASACK, LARS and also FIFO and SCFQ.
- We demonstrate that the two modifications of EFD either outperform other scheduling policies or perform similarly but with a lower overhead in terms of flow bookkeeping[2].
- We demonstrate that PEFD, that requires no inspection of TCP packets achieves similarly to EFDACK, except when the buffer size becomes too small.
- We extend the original design of EFD by considering alternative scheduling policies for the low and high priority queues and discuss their impact.

The remainder of this paper is organized as follows. We introduce new variants of EFD to be analyzed in an 802.11 context in Section II. In Section III, we detail our evaluation methodology. Sections IV and V present the evaluation results of the different scheduling disciplines. Section VI reviews prior related work. Section VII concludes the paper.

II. Scheduling disciplines

SCFQ [6] is known to be a good approximation of Processor Sharing (PS) in practice for packet networks. As the extensions of LAS, LASACK [18] bases its decision on the total amount of bytes sent so far by each flow in both directions, whereas LARS [8] applies a temporal decay to the service obtained by a flow - accounting for volumes and rates simultaneously. Run2C [1] is in essential a two level FIFO+FIFO method, that uses a threshold to differentiate short and long transfers.

EFD [3] belongs to the family of multi-level processor sharing (MLPS - see [9]) policies. It features two (virtual) queues called the high-priority and low-priority queues respectively. Both are drained using the FIFO discipline.The low priority queue is serviced only if the high priority queue is empty.

The key difference between EFD and other MLPS scheduling disciplines is the way flow states are handled. The EFD scheduler keeps track of flows only as long as they have at least one packet present in the queue. At the creation of the flow record, the packets of a fresh flow are serviced by the high priority queue. If, at some point in time, the record of a flow maintained by the scheduler exceeds a threshold th, where th is typically 20 packets or equivalently 30 KB (considering MSS[3] packets), its subsequent packets are directed to the low priority queue. For this to happen, the flow must have had continuously at least one packet in the queue.

Let us consider the example of a TCP flow in its early infancy. Assuming that delayed-ack is turned off, and neglecting the interaction with other flows and the connection set-up, the scheduler will create a record for the first data packet of this flow, delete it upon its departure from the queue, create a new record for the second flight of 2 packets, delete it upon departure, etc.

Now if we consider the example of a bursty UDP flow that sends batches of n packets per RTT, as soon as $n \geq th$, th packets will be serviced by the high priority queue and $n-th$ in the low priority queue (for simplicity, we consider MSS-long packets here).

The above flow management process has a key advantage of constraining the size of the flow table to the physical size of the queue[4]. More generally, it has been observed that [3]:

- EFD keeps track of a number of flows that is orders of magnitude smaller than the other size-based scheduling disciplines. It is often much smaller than the physical queue.
- EFD avoids lock-outs between long flows and starvation of long flows, similarly to Run2C or LARS.
- EFD accounts for both volumes and rates in its scheduling decision, though not as explicitly as LARS that applies a temporal decay to the accumulated of service received by a flow.

The original work on EFD [3] considered the applicability of EFD in wired networks. In the present paper, our focus is on 802.11 networks, which feature two key properties that lead to the TCP performance problem: (i) the protocol is half-duplex, meaning that uploads and downloads share the wireless medium and (ii) the Access Point is not granted a high enough priority to access the medium under DCF, which means that its queue, which is typically 30 to 100 packets, tends to build up.

EFD was designated with quite large buffers of typically 300 packets in mind, which is not unusual for routers. In a wireless context, 300 packets seems like a big buffer, although high speed access points (802.11n) typically store hundreds of packets when a station temporarily leaves the network to scan for other access points. When this temporary buffer is cleared (once the station comes back) the AP reverts to its normal operational mode where it typically uses a buffer (shared by all stations) that is always smaller. Hence, we explore how reducing the buffer size impacts EFD's behavior.

A. *Adapting EFD to half-duplex links*

The original EFD policy accounts for volumes in bytes. An alternative is to count volumes in terms of number of packets.

[2]The benefit of EFD concerning the overhead has been clearly justified in [3]. To avoid redundancy, we don't discuss the memory consumption in this paper as the two modifications of EFD naturally inherit this good property from EFD.

[3]Maximum Segment Size (MSS) is equal to 1460 bytes by Ethernet standard. We use MSS packet in this paper to denote the data packet with the maximum size allowed.

[4]Remember that most if not all networking devices generally limit the size of their queues by the number of packets they can hold as opposed to the number of bytes the packets are worth.

In the remainder of the paper, we refer to these two EFD flavors as BEFD (Byte-based EFD) and PEFD (Packet-based EFD) respectively. To illustrate the difference between these two options, consider the case of a WLAN with a single upload and a single download. At the buffer of the AP, one observes, in the downstream direction, the data packet stream from the download and the ACK packet stream from the upload. As data packets are generally MSS packets while ACKs are 40 bytes packets, one clearly sees that counting volumes in bytes or packets will significantly impact the priority granted to the ACK stream: when counting in bytes, its priority will consistently be maximum whereas the competition between the upload and download will be more fair when counting in packets.

In addition to BEFD and PEFD, we introduce a variant of EFD that accounts for the half-duplex nature of MAC layer protocol. It attributes a virtual service size to TCP ACK packet by accounting for the total amount of data traffic that has been transferred by the flow so far, obtained through the TCP acknowledgment number in the TCP header. We call EFDACK this scheduling policy. Considering the same example as above of a WLAN cell with a single upload and a single download, and assuming that the flows are continuously tracked by the scheduler, the priority of an ACK packet is related to the total amount of bytes sent by the upload. We compare EFDACK, BEFD and PEFD extensively in Sections IV and V.

Essentially, the original EFD and its adaptation for 802.11 network - EFDACK, are FIFO+FIFO schemes since packets within each (virtual) queue are drained using the FIFO discipline at packet level. We also investigate in this paper the impact of alternative scheduling disciplines in the EFD scheme. In particular, we consider two candidates, FIFO and LAS, which leads to four combinations: FIFO+FIFO, LAS+FIFO, FIFO+LAS, LAS+LAS. We explore the relative merits of these flavors of EFD in Section V-1.

A last point to mention is that each of the scheduling policies that we consider in this paper are paired with a buffer management scheme. For FIFO or SCFQ (an implementation of Processor Sharing for packet networks [6]), this is drop tail. In contrast, for the size-based scheduling policies, when the queue is full, the newly arriving packet is assigned a priority according the scheduling policy and this is the packet with the smallest priority that is discarded.

III. Evaluation Methodology

In this section, we provide a high level overview of the evaluation methodology we apply to compare the variants of EFD that we introduced in the previous section to state-of-the-art scheduling policies.

A. Network Configuration

In this paper, we consider a simple network configuration with 10 wired hosts and 10 wireless stations associated to a single access point, as depicted in Figure 1. We use the 802.11a protocol with nominal bit rate of 54Mb/s, with RTS/CTS disabled. Good and fair radio transmission conditions are guaranteed as the 10 wireless stations are at the same physical distance from the access point and in line of sight of each other. The 10 wired hosts are connected to a router with an output rate 10 times larger than its input rate, so that its output queue never builds up. With such a configuration, the bottleneck is the access point. We use QualNet 4.5 to obtain all simulation results. TCP NewReno is used with delayed ACK enabled in the simulations.

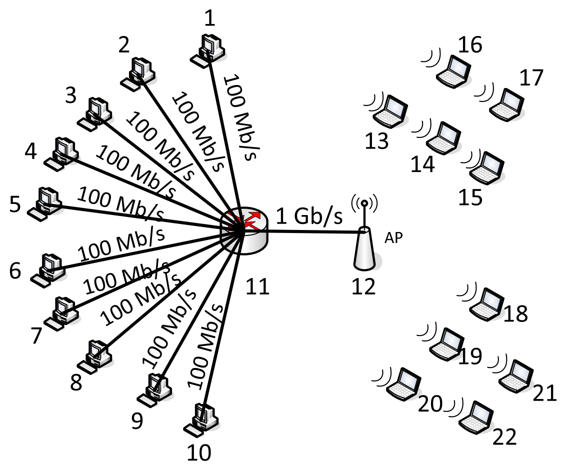


Figure 1. Network Set-up, with one way delay of 2ms in wired part

B. Workload

A key point in our evaluation is the choice of workload. We consider essentially two workloads. First, we use only long-lived flows: while unrealistic, results obtained under such a workload enable to pinpoint easily some fundamental characteristics of a scheduling policy, due to the relative simplicity of the scenario.

Secondly, we consider a more realistic case of a mix of short and long flows. We generate the workload with the assumption that TCP connections arrive according to a Poisson process with rate λ and adjust λ so as to obtain two regimes: a medium load of 10 Mbit/s and a high load of 20 Mbit/s. These loads have to be considered relatively to the maximum throughput of a single TCP transfer over 802.11a at 54 Mbit/s, which is merely 27.3 Mbit/s [4]. The workload consists of bulk TCP transfers of varying size, generated from a bounded Zipf distribution with an average size of about 60 Kbytes (40 packets with size of 1500 bytes each), which is in line with flow sizes observed on typical campus WLANs [12]. The minimum transfer size is 6 MSS, and the maximum transfer volume corresponds to 10 MB with a coefficient of variation[5] of 6, which controls how the mass of the distribution is split between short and long transfers. Note that bounded Zipf is a discrete equivalent of a continuous (bounded) Pareto distribution, and Pareto is a long tailed distribution usually adopted for modeling flows in the Internet. Each packet has a fixed size of 1500 bytes in our simulations.

[5]The CoV is defined as the ratio of the standard deviation to the mean of a distribution. The larger it is, the more skewed the distribution.

A last important parameter of the workload, in a 802.11 scenario where the medium is managed in a half-duplex manner, is the ratio of download to upload traffic. We denote by λ_d and λ_u the arrival rate of TCP downloads and uploads respectively. We considered initially three scenarios: $\frac{\lambda d}{\lambda u}$=1 for symmetric load, $\frac{\lambda d}{\lambda u}$=10 and $\frac{\lambda d}{\lambda u}$=100 for two asymmetric loads respectively. Those three scenarios are related to real use cases. The case $\frac{\lambda d}{\lambda u}$=10 corresponds to a typical residential user browsing the Web with no heavy P2P nor HTTP streaming (YouTube, DailyMotion, etc.) activity [13]. Clients that rely heavily on P2P tend to produce more symmetric ratios, corresponding to $\frac{\lambda d}{\lambda u}$=1. On the other side of the spectrum, a trend in residential network is to see more and more heavy hitters characterized by a heavy HTTP streaming activity[13]. In such a scenario, almost all bytes flow from the server to the client, leading to ratios close to 100.

To gain insights about the typical traffic within an enterprise network, we captured one full day of traffic within the Eurecom network, which comprises about 600 machines and 60 servers. We analyzed the ratio of download to upload traffic for intranet traffic and Internet traffic of each host and found that Internet traffic corresponds to an average ratio of 10, as users mostly browse the Internet, without heavy HTTP streaming activity. In contrast, intranet traffic (SMB, LDAP, etc.) is larger in volume and highly symmetric, *i.e.* characterized by ratio close to 1. A reason why the ratio of the latter is symmetric is that p2p traffic is banned from the network, as from most enterprise networks in general.

In Section V, we consider the cases $\frac{\lambda d}{\lambda u}$=1 for symmetric load, $\frac{\lambda d}{\lambda u}$=10 for asymmetric load as the case $\frac{\lambda d}{\lambda u}$=100 is less frequent in enterprise networks and degenerates to the pure download case, where the TCP unfairness problem typically vanishes. We sum up the simulation parameters in Table I.

Table I
SIMULATION PARAMETERS

Simulator		QualNet 4.5		
MAC protocol		802.11a@54Mbit/s		
Workload	long-lived cnxs	buffer size	10-70 MSS	
		composition	5 uploads vs. 5 downloads	
	mixed workload	buffer size	30MSS / 300 MSS	
		transfer size distr.	bounded Zipf	
		load regimes	medium	10 Mbit/s
			high	20 Mbit/s
		traffic ratio	sym.	$\lambda_d/\lambda_u = 1$
			asym.	$\lambda_d/\lambda_u = 10$

C. Performance Metrics

We focus on two performance metrics in our study. First, the global volumes uploaded and downloaded. It is important to keep an eye on this metric to assess the ability of a scheduling policy to effectively use the available network capacity. Secondly, the conditional response times in each flow direction as they allow to observe how the scheduling discipline treats each flow size and also if unfairness exists between uploads and downloads or between flows of various sizes.

IV. THE CASE OF LONG-LIVED CONNECTIONS

In this section, we evaluate the fairness of the following disciplines: FIFO, BEFD, PEFD, EFDACK, LASCAK, LARS, Run2C and SCFQ for the case of long lived TCP transfers, in order to highlight the impact of half-duplex nature of 802.11 wireless links. In the case of Run2C, we use a variant that takes into account the volume transferred in both directions (by tracking ACK number progress), as otherwise it would only worsen the unfairness. We refer to it as Run2CACK.

Each Qualnet simulation lasts 100 seconds. We consider a scenario with 5 uploads and 5 downloads. The TCP unfairness problem gets more pronounced with decreasing buffer size [14]. This is because the root of the problem lies in the competition to access the buffer of the AP. Conversely, unfairness drops and eventually vanishes for all scheduling disciplines when buffer size increases, although at the cost of extreme queueing delays for *e.g.* FIFO. In our simulations, we considered buffer sizes from 10 to 500 packets. We observed that losses are not observed any more when the buffer reaches around 300 packets. Indeed, since the receiver's advertised window is set to 65 KB, which is equivalent to 43 MSS, at most 5×43 outstanding data packets for the 5 downstream flows and $5 \times (43/2)$ outstanding ACK packets for the 5 upstream flows can be in the buffer at any time (with delayed ACK). For values larger than 300 packets, all policies are fair, although response time explodes for FIFO.

We report below on results for small buffer sizes from 10 to 70 packets. Figure 2 depicts the aggregate long term throughput of the uploading and downloading flows, by taking the average of 30 independent simulations.

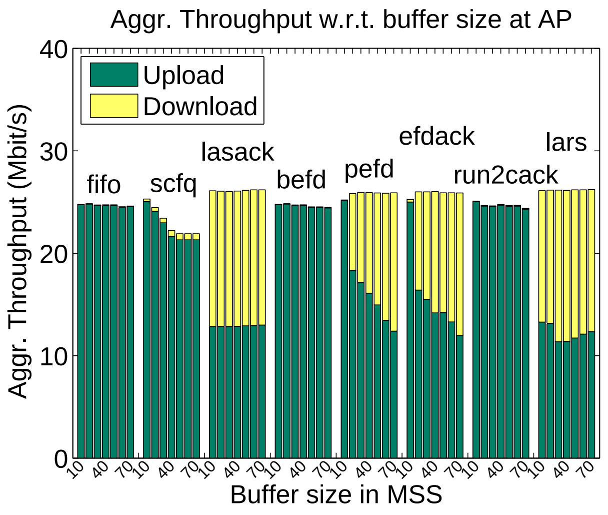


Figure 2. Long-lived connections: 5 uploads against 5 downloads

The pronounced unfairness between uploads and downloads experienced by legacy FIFO is clearly illustrated by Figure 2 when the buffer size is small. Moreover, we observe from the ratio of upload to download aggregate throughputs that, the original EFD (*i.e.* BEFD) is even less fair than FIFO, as uploads highly restrain downloads and achieve throughput 2 to 3 orders of magnitude larger than that of downloads when the buffer size is small. This is due to the high priority granted to ACKs as mentioned in section II-A. With small buffer, this low priority translates into high loss rates for downloads under BEFD and Run2C. In contrast, the loss rates experienced under LASACK, PEFD, EFDACK and LARS are negligible (with a buffer larger than 20 packets). Although Run2CACK keeps track of bidirectional traffic, long lived connections quickly end up in the low priority queue, so that this policy degenerates to FIFO in this setup.

Figure 2 further demonstrates that the network capacity is fairly shared between uploads and downloads under LASACK[7] and under LARS[8]. Meanwhile, PEFD and EFDACK are able to enforce a good level of fairness – far better than FIFO, SCFQ, and BEFD but not as perfect as LASACK or LARS – when the buffer size is larger than 20 packets. An interesting point is that fairness is not obtained at the expense of performance degradation as the aggregate throughputs under PEFD and EFDACK are larger than the ones of FIFO and SCFQ.

In an attempt to better understand the modus operandi of BEFD and EFDACK, we have computed the mean value of the two metrics: RTT and congestion window, both for the uploads and the downloads, as a function of the buffer size at the access point, which are represented in Figure 3, by collecting the samples in 30 independent simulations.

A scheduling policy might impact both the congestion window of a flow and its RTT. It can impact the congestion window by creating losses. Controlling the RTT is simply obtained by varying the priority of the packet of the flow at the scheduler. In a sense, losses can be seen as an extreme case of the delay (an infinite delay), hence the RTT is the primary variable through which a scheduler controls a TCP connection. Furthermore, if the scheduler considers only the direction in which ACKs travel, then keeping the ACKs is the only control variable as dropping them has only a limited impact on *cwnd* growth.

We observe first that RTTs are similar between uploads and downloads when the queuing policy does not differentiate between up and down directions. This is the case for FIFO and BEFD. This confirms the fact that there is a single bottleneck (the buffer of the AP) that governs all RTTs. When its size grows, the RTT grows. Second, it is clear that for FIFO, the download congestion windows do not significantly grow, so that these connections throughput remains low. With BEFD things are even worse. With EFDACK, uploads and downloads are effectively decoupled by the scheduler that inflates the RTT to compensate congestion window increase. The result with EFDACK is that throughputs of uploads and downloads are eventually similar, *i.e.* the TCP unfairness problem vanishes. We observed a similar effect with LARS, and to a lesser extent with PEFD.

One of the lessons of the above evaluation is that SCFQ and BEFD are clearly ineffective when the traffic consists of both uploads and downloads. This is why we rule them out from further investigation bellow. One can argue that this is also the case for FIFO. However, as FIFO is the legacy scheduling discipline, we keep it as a reference point hereafter.

V. Performance Evaluation using Realistic Workloads

In this section, we first investigate the impact of varying the scheduling discipline for EFD like schemes. We consider 4 combinations of disciplines: FIFO+FIFO, LAS+FIFO, FIFO+LAS, LAS+LAS in two different flavors corresponding to a threshold either in byte like in EFDACK or in packets like PEFD. We conclude that the original FIFO+FIFO is a good candidate and thus focus only on the original PEFD and EFDACK in subsequent analyses.

We next compare PEFD and EFDACK to FIFO, LARS, LASACK and Run2CACK. We examine the conditional response time of uploads and downloads, assuming a highly skewed (as the coefficient of variation is 6) flow size distribution. Finally, we discuss the impact of the buffer size at the AP on the performance of scheduling policies in 802.11 networks.

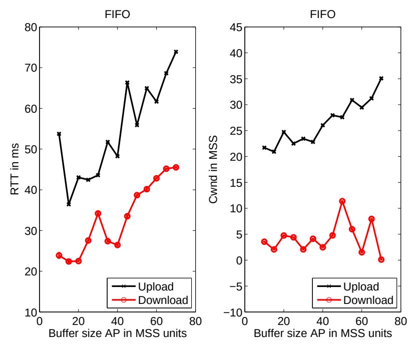
The simulation parameters are given in Table I, and each simulation lasts 5000s. Some connections are unfinished at the end of a simulation due to the premature end of simulation; however, under high load and for long enough simulations as in our case, the main reason is that they were set aside by the scheduler. We report performance results only for the connections that have completed a transfer. In this section, we do not represent on the figures the confidence intervals (for each flow size) as, given the number of curves per figure, they tend to obscure the graphs. Still, they enabled us to check that the simulations were long enough to draw conclusions based on the conditional mean response times.

1) Comparison of EFD Variants: In this part, we consider four variants of EFD: LAS+FIFO, FIFO+LAS, LAS+LAS as well as FIFO+FIFO itself. For each variant, we have two flavors, depending on the bookkeeping option which is either in bytes like EFDACK or packets as PEFD.

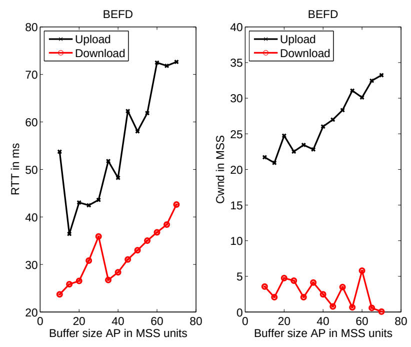
Before going into the details, we need to explicit the way LAS is used here. This is the global EFD scheduler that assigns the volumes, either in packets or bytes depending on the strategy. Each packet is thus marked with an associated volume and, when LAS is used, it manages the queue where it is applied in such a way that packets are always sorted in ascending order of their associated volume.

We conducted simulations for a symmetric load and 10 Mbit/s (moderate load) and 20 Mbit/s (high load) respectively. The buffer size is set to 30 packets. Average conditional response times of byte-based schemes are depicted in Figure 4 while the case for the packet-based schemes are illustrated in Figure 5. Results with an asymmetric load are qualitatively similar and we do not present them here.

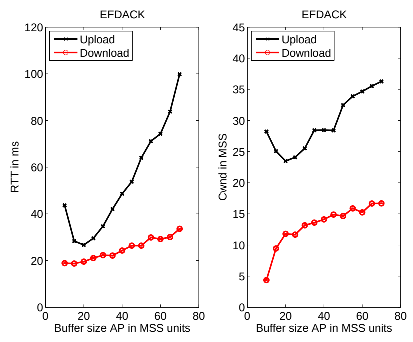
We observe from Figure 4(a) that the 4 schemes perform similarly. They all offer lower response time to short flows as compared to FIFO, but at the cost of a slight increase of completion time for long flows when the offered load is moderate at 10 Mbit/s. A similar effect for the case of packet-based scenario is visible in Figure 5(a). When the load is



(a) FIFO

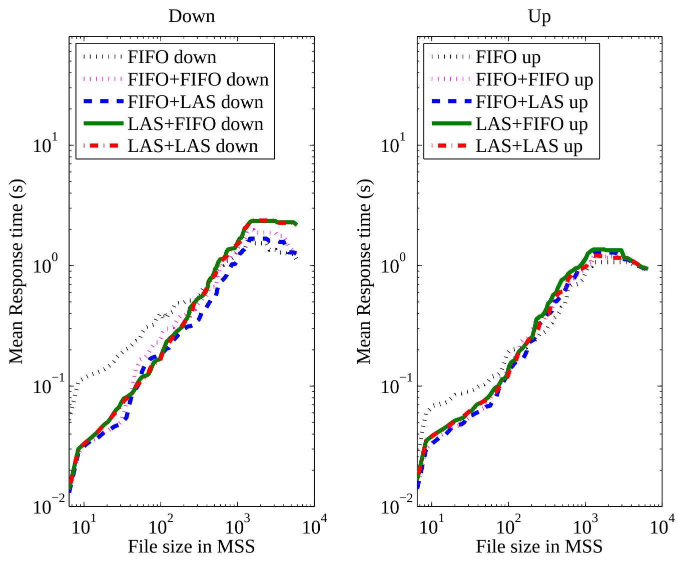


(b) BEFD

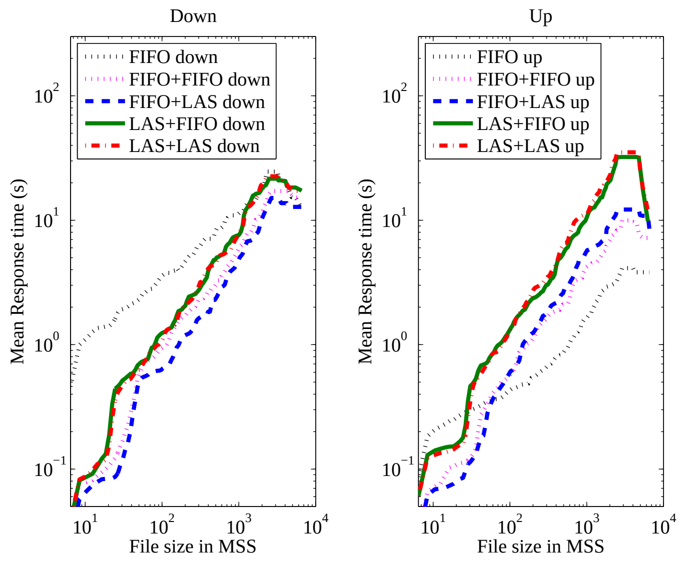


(c) EFDACK

Figure 3. How does the scheduler control the connection throughputs? RTT and Cwnd w.r.t. buffer size at AP



(a) Workload of 10Mbit/s



(b) Workload of 20Mbit/s

Figure 4. Comparison between various queueing policies in EFD queues – Average response time, symmetric load, byte-based

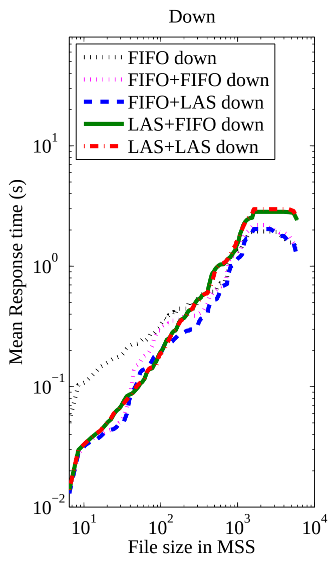
high, the behavior of the 4 different schemes differ especially for the byte-based scenario. FIFO+LAS basically offers the best response time for both scenarios, as illustrated in Figure 4(b) and Figure 5(b). FIFO+FIFO performs quite close to FIFO+LAS for the byte-based scenario. Using LAS in the high priority queue seems detrimental. Though the use of LAS is different from the original LAS policy that has a full knowledge of the history of each flow, we believe that the bad performance obtained when LAS is used in the high priority queue is a consequence of the bad performance of LAS when the distribution has a low variability - see [9]. This is the case in the high priority queue perspective here, since the flow sizes in this queue range between 1 and 30 MSS only, and the distribution is much less skewed (CoV close to 1) than the overall distribution (CoV of 6).

In conclusion, modifying the queuing discipline of each individual queue in an EFD scheduler (reasoning on packet or bytes) appear beneficial only for the low priority queue and can have a detrimental effect in the high priority. Overall, the benefit of LAS in the low priority queue seems limited in comparison to the increased complexity. We thus consider only the original FIFO+FIFO flavors, namely PEFD and EFDACK in the rest of this section.

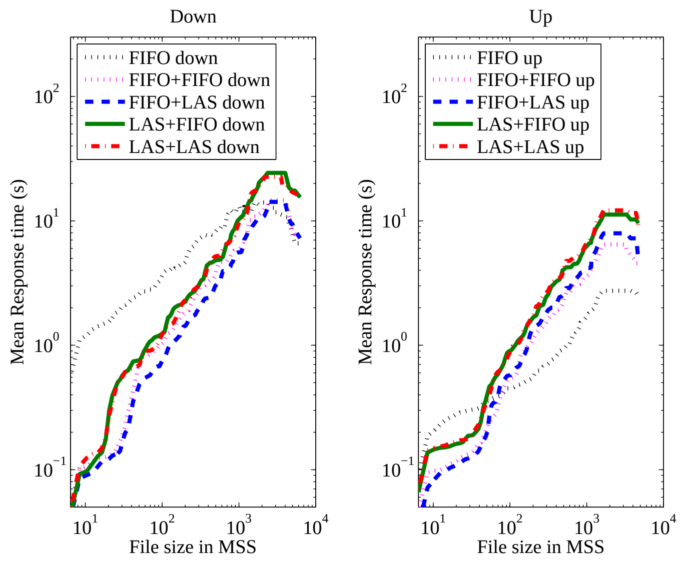
2) Impact of Load and Symmetry Ratio: We present simulations results for 10 and 20 Mbit/s and for symmetric ($\frac{\lambda d}{\lambda u}$=1) and asymmetric ($\frac{\lambda d}{\lambda u}$=10) scenarios. The buffer size is set to 30 packets. Conditional response times of uploads and downloads are depicted in Figures 6 and 7 respectively. The response time is defined as the time required for a TCP connection of a given size to complete its transfer (set-up, data transfer and tear-down).

We first observe that under FIFO, for all the scenarios and all load condition - even a moderate load - the TCP unfairness problem is visible. It is thus a performance problem for any operational 802.11 network.

In contrast, we observe that all size-based scheduling policies mitigate the TCP unfairness problem, while granting a high priority to short flows, whose performance significantly

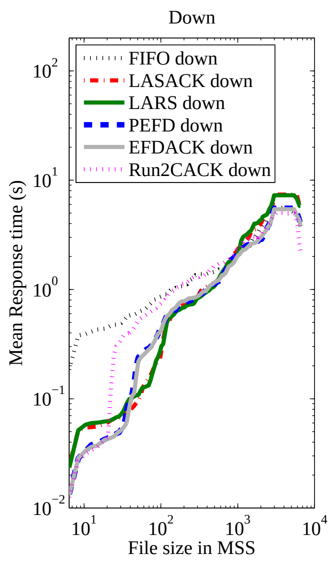


(a) Workload of 10Mbit/s

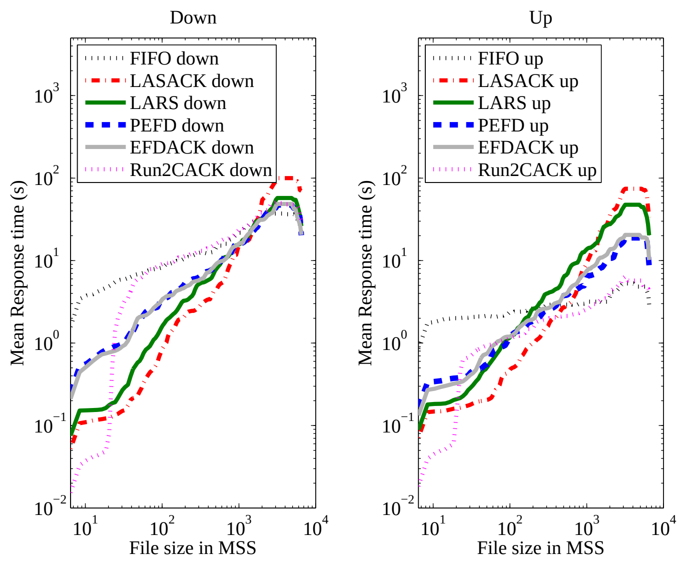


(b) Workload of 20Mbit/s

Figure 5. Comparison between various queueing policies in EFD queues – Average response time, symmetric load, packet-based



(a) Workload of 10Mbit/s



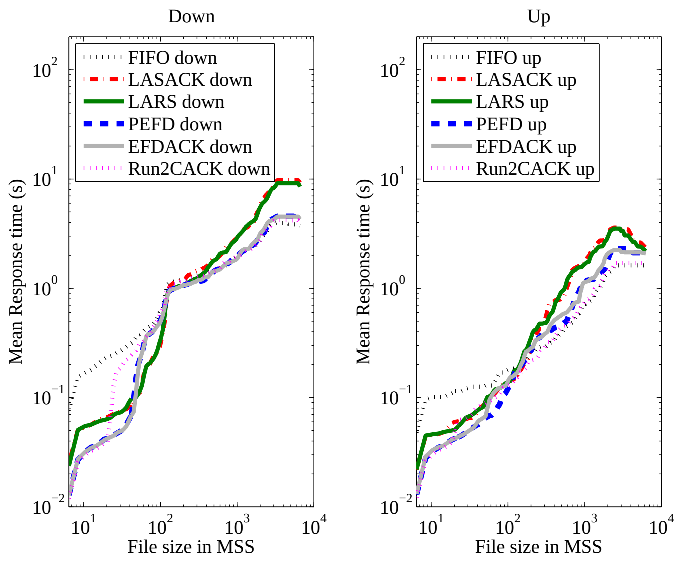
(b) Workload of 20Mbit/s

Figure 6. Comparison of EFD variants for a symmetric workload: average response time – AP buffer of 30MSS

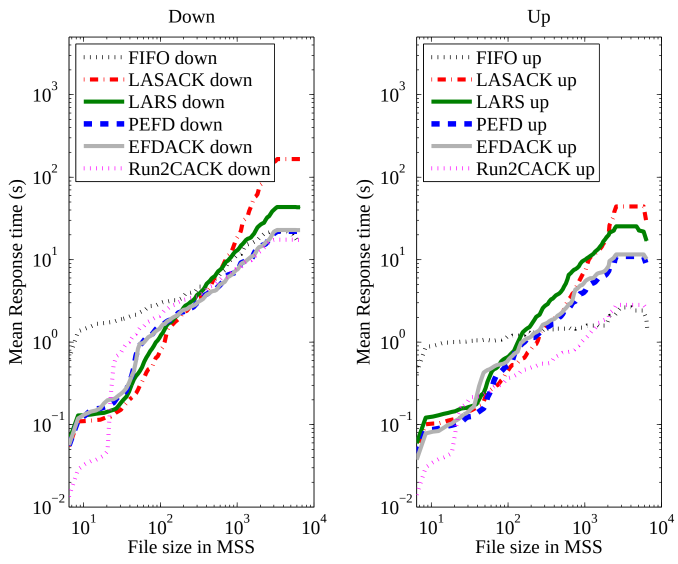
improve as compared to FIFO. These are obtained at the cost of a negligible increase of the response time of long flows.

An important remark is that we present conditional response times as a function of flow size so as to see the impact of the scheduling disciplines on each flow size. However, with a point of view that would perhaps better account for user experience, one could have considered the percentiles of flow size on the x-axis. This would have magnified the left side of each plot because short flows represent the majority of flows, *e.g.*, the 90-th quantile is less than approximately 50 packets, meaning that 90% of the flows experience a significant improvement with the size-based scheduling policies we consider.

The figures show that LASACK performs slightly better than PEFFD and EFDACK, especially for mid-size-flows. This is a side-effect of the threshold used in PEFD and EFDACK. Overall, the take-away message is that PEFD and EFDACK are able to achieve almost as well as state-of-the-art size-based scheduling policies that keep a full memory of each flow (in contrast to EFD like policies that have a memory "limited to the buffer"). Here, Run2CACK uses the same size threshold as EFD to decide in which queue a packet should go. But due to its infinite memory, flows go earlier in the low priority queue, following the expected behavior described in Section II. In fact, Run2CACK gives a more marked transition than



(a) Workload of 10Mbit/s



(b) Workload of 20Mbit/s

Figure 7. Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 30MSS

EFD, with a pronounced protection of short flows detrimental to even mid-size ones, so that it is in fact more sensitive to the transition threshold setting.

3) The Impact of Buffer size at AP: We considered buffer sizes ranging from 10 to 500 packets. Due to space limit, we pick two representative values: 30 and 300 packets. Simulations are conducted in an asymmetric load scenario. Results are presented respectively in Figures 7 and 8.

When the buffer size is large - 300 MSS for instance, there is no more unfairness between uploads and downloads even with FIFO regardless of the load, as the queue rarely overflows. Nevertheless, this is obtained at the cost of very long times spent in the AP downlink queue.

Comparing with figure 7, PEFD, EFDACK and LASACK do not suffer nor benefit from larger buffer space. This is in line with our previous results and the results obtained in the original EFD paper [3], although the buffer size is directly linked to the scheduler "memory". This confirms that, unlike FIFO, (some) size-based scheduling policies are much less sensitive to the actual buffer size.

VI. Related work

Classically, size-based scheduling policies are divided into blind and non-blind scheduling policies. A blind size-based scheduling policy is not aware of the job[6] size while a non-blind one is. Non blind scheduling policies are applicable to servers [17] where the job size is related to the size of the content to transfer. A typical example of non blind policy is the Shortest Remaining Processing Time (SRPT) policy, which is optimal among all scheduling policies, in the sense that it minimizes the average response time.

[6]Job is a generic entity in queueing theory. In the context of this work, a job corresponds to a flow.

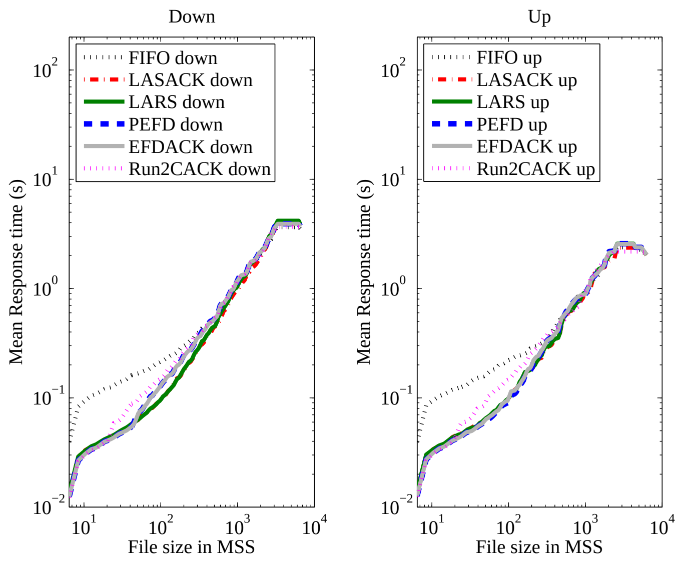
For the case of network appliances (routers, access points, etc.) the job size, i.e. the total number of bytes to transfer, is not known in advance. Several blind size-based scheduling policies have been proposed. The Least Attained Service (LAS) policy [15] bases its scheduling decision on the amount of service received so far by a flow. LAS is known to be optimal if the flow size distribution has a decreasing hazard rate (DHR) as it becomes, in this context, a special case of the optimal Gittins policy [5]. Some representatives of the family of Multi-Level Processor Sharing (MLPS) scheduling policies [10] have also been proposed to favor short flows. An MLPS policy consists of several levels corresponding to different amounts of attained service of jobs, with possibly a different scheduling policy at each level. Run2C, which is a specific case of MLPS policy using FIFO+FIFO scheduling, has already been compared to LAS [1].

Run2C and LAS share a number of drawbacks. Flow bookkeeping is complex. LAS requires to keep one state per flow. Run2C needs to check, for each incoming packet, if it belongs to a short or to a long flow. Moreover, both LAS and Run2C classify flows based on the accumulated number of bytes they have sent, without taking the flow rate into account.

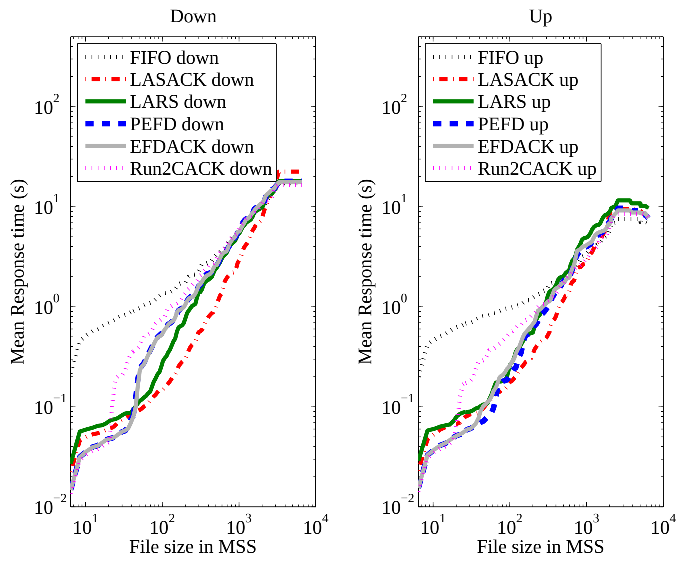
LARS is a size-based scheduling designed to account for rates [8]. It consists in a variant of LAS, Least Attained Recent Service (LARS), where the amount of bytes sent by each flow decays with time according to a fading factor β. LARS is able to handle differently two flows that have sent a similar amount of bytes but at different rates and it also limits the lock out duration of one long flow by another long flow to a maximum tunable value.

VII. Conclusion

This paper presents the adaptation and evaluation of EFD to the case of IEEE 802.11 networks, the most common half



(a) Workload of 10Mbit/s



(b) Workload of 20Mbit/s

Figure 8. Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 300MSS

duplex links effectively in use. There are basically two ways to do this adaptation: keep track of the volumes exchanged in both directions or simply count packets in a single direction. In fact, as long as the workload does not consist of flows with very disparate MSS, PEFD is a much simpler approach.

Compared to size-based scheduler with infinite flow states memory, EFD is marginally less efficient in combatting the TCP unfairness problem than LARS or LASACK; this is especially evident for long lived flow experiments. Nevertheless, for a more realistic workload, this difference vanishes even for relatively short buffers. In brief, the EFD variants presented in this paper are simple, low overhead schedulers that can effectively improve performance in wireless networks, without the usual drawbacks associated to size-based schedulers.

Acknowledgements

This work was supported by the French National Research Agency (ANR) project ELAN under contract ANR-08-VERS-008.

References

[1] Konstantin Avrachenkov, Urtzi Ayesta, Patrick Brown, and Eeva Nyberg. Differentiation between Short and Long TCP Flows: Predictability of the Response Time. In *IEEE INFOCOM*, 2004.

[2] Marco Bottigliengo, Claudio Casetti, Carla-Fabiana Chiasserini, and Michela Meo. Short-term Fairness for TCP Flows in 802.11b WLANs. In *IEEE INFOCOM*, 2004.

[3] Jinbang Chen, Martin Heusse, and Guillaume Urvoy-Keller. EFD: An Efficient Low-Overhead Scheduler. In *NETWORKING*, 2011.

[4] Matthew Gast. When Is 54 Not Equal to 54? A Look at 802.11a, b, and g Throughput. Website, 2003. http://tecweb.georgebrown.ca/jolenewa/courses/COMP2102/Files/A%20Look%20at%20802.11a,%20b,%20and%20g%20Throughput.pdf.

[5] J.C. Gittins. *Multi-armed bandit allocation indices*. Wiley-Interscience, 1989.

[6] S. Jamaloddin Golestani. A self-clocked fair queueing scheme for broadband applications. In *IEEE INFOCOM*, pages 636–646, 1994.

[7] Martin Heusse, Guillaume Urvoy-Keller, and Andrzej Duda. Layer 2 vs. Layer 3 Mechanisms for Improving TCP Performance in 802.11 Wireless LANs. In *ITC*, 2008.

[8] Martin Heusse, Guillaume Urvoy-Keller, Andrzej Duda, and Timothy X. Brown. Least Attained Recent Service for Packet Scheduling over Wireless LANs. In *IEEE WoWMoM*, 2010.

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

[10] Leonard Kleinrock. *Computer Applications, Volume 2, Queueing Systems*. Wiley-Interscience, 1 edition, April 1976.

[11] Elena Lopez-Aguilera, Martin Heusse, Yan Grunenberger, Franck Rousseau, Andrzej Duda, and Jordi Casademont. An Asymmetric Access Point for Solving the Unfairness Problem in WLANs. *IEEE Transactions on Mobile Computing*, 7(10):1213–1227, October 2008.

[12] Xiaoqiao Meng, Starsky H. Y. Wong, Yuan Yuan, and Songwu Lu. Characterizing flows in large wireless data networks. In *ACM MOBICOM*, pages 174–186, 2004.

[13] Marcin Pietrzyk, Louis Plissonneau, Guillaume Urvoy-Keller, and Taoufik En-Najjary. On profiling residential customers. In *TMA*, 2011.

[14] Saar Pilosof, Ramachandran Ramjee, Danny Raz, Ran Ramjee, Yuval Shavitt, and Prasun Sinha. Understanding TCP fairness over Wireless LAN. In *IEEE INFOCOM*, pages 863–872, 2003.

[15] Idris A. Rai, Ernst W. Biersack, and Guillaume Urvoy-keller. Size-based scheduling to improve the performance of short tcp flows. *IEEE Network*, pages 12–17, vol.19, 2004.

[16] Idris A. Rai, Guillaume Urvoy-Keller, Mary K. Vernon, and Ernst W. Biersack. Performance analysis of LAS-based scheduling disciplines in a packet switched network. In *ACM SIGMETRICS*, volume 32, pages 106–117, New York, NY, USA, June 2004. ACM Press.

[17] Bianca Schroeder and Mor Harchol Balter. Web servers under overload: How scheduling can help. *ACM Trans. Internet Technol.*, (1):20–52, vol.6, 2006.

[18] Guillaume Urvoy-Keller and André-Luc Beylot. Improving flow level fairness and interactivity in WLANs using size-based scheduling policies. In *ACM MSWiM*, pages 333–340. ACM, 2008.