# A Green, Modular and Fair Packet Scheduler for Boosting Throughput over Wireless Links

Maurizio Casoni[1], Carlo Augusto Grazia[1], Paolo Valente[2]
[1]Department of Engineering
University of Modena and Reggio Emilia
via Vignolese 905, 41125 Modena, Italy
[2]Department of Physics, Computer Science and Mathematics
University of Modena and Reggio Emilia
via Giuseppe Campi 213/a, 41125 Modena, Italy
{maurizio.casoni, carloaugusto.grazia, paolo.valente}@unimore.it

***ABSTRACT:*** *We study the problem of defining and implementing a packet scheduler able to give the high performance of a wired packet scheduler over a wireless link, i.e. a low execution time with low power consumption, high QoS guarantees and high throughput.*

*A common solution is using a single, integrated scheduler that deals both with the QoS guarantees and the wireless link issues. Unfortunately, such an approach is little flexible and does not allow any of the existing high-quality schedulers for wired links to be used without modification.*

*To address these issues, in this paper we validate a modular architecture which permits the use, as they are, of existing packet schedulers for wired links over wireless links, and at the same time allows the flexibility to adapt to different channel conditions.*

*We validate the effectiveness of the modular architecture by showing, through experimental results, that this architecture enables us to get a new scheduler with the following features, just by combining existing schedulers: execution time and energy consumption close to that of just a Deficit Round Robin, accurate fairness and delay guarantees, possibility to set, by changing one parameter, the desired trade-off between throughput-boosting level and granularity of the service guarantees. In particular, we show that this scheduler, which we named High-throughput Twin Fair scheduler (HFS), outperforms one of the most accurate and efficient integrated schedulers available in the literature.*

**Keywords:** Cross-layering, Energy Consumption, QoS, Scheduling, Throughput

**Received:** 18 August 2013, Revised 20 September 2013, Accepted 25 September 2013

© 2013 DLINE. All rights reserved

## 1. Introduction

State-of-the-art wireless technologies aim at supporting increasingly high data rates for applications such as video streaming, web browsing and file sharing, in both stationary and nomadic/mobile scenarios. Moreover, given the growing spread of smartphones and tablets, an increasing number of users access to wireless networks everyday.

This trend puts several limits on how network and/or service providers can effectively supply adequate quality of service (QoS) to their users, since over-provisioning is hard to implement in wireless contexts. Most current wireless systems directly provide QoS capabilities, in terms of traffic differentiation, traffic prioritization and so on. In this respect, one of the most important sub-systems involved in QoS provision is the packet scheduler, which properly sets the order by which packets are sent over a given interface, both in the uplink and in the downlink.

State-of-the-art solutions to provide both QoS guarantees, as well as a high throughput, are based on *cross-layering*: scheduling decisions are made by using also channel state information coming from the MAC layer [1], [2], [3], [4]. For example, per-destination channel conditions may be considered when deciding which flow to serve, in order to avoid transmission failures. In these proposals, just *one integrated* scheduler takes *all* the scheduling decisions, based on these issues and on the desired QoS.

Unfortunately, integrated solution entails a few drawback. High-quality schedulers for reliable links [5], [6], [7], [8], [9], [10], [11], [12] cannot be used and converted into cross-layer schedulers, without modifying them. After these modifications, the guarantees provided by the scheduler are likely to change and need to be recomputed. Finally, if the medium access protocol or the channel technology changes, or if we want to use new techniques for achieving an even higher throughput or saving energy, then the scheduler is likely not to fit the new technology or requirements. Hence it may need to be modified again. Especially, if we want to use the same scheduler on heterogeneous wireless technologies, we may need to define a different version of it for each technology.

In this work we validate the flexible and effective packetscheduling architecture proposed in [13]. The architecture focuses on *local* packet schedulers, which are executed inside wireless nodes, and decide (only) the order by which packets are transmitted over the outgoing links of the node. Such architecture preserves both effectiveness and flexibility, and permits to reuse existing schedulers without modification; we implement such architecture in which the scheduler just chooses the next packet to transmit according to its QoS policy, but delivers it to a *middle layer* instead of the MAC layer. The middle layer then deals with the issues of the wireless medium and reorders packet transmissions if needed. With this architecture it is easy to cherry pick from the literature and combine the best solutions in terms of service guarantees, computational cost, power consumption and throughput boosting. As for the last two goals, we note that this architecture allows a system to *profit from cross-layering* while still *preserving flexibility*.

### 1.1 Contribution

How effective is this architecture? In this paper we answer, in a sense, to this question by defining a new flexible, efficient and green packet scheduler for wireless links: High-throughput twin Fair Scheduler (HFS). To present it more thoroughly we complete the architecture description with the developed mechanism that moves the packets from the QoS layer to the *middle layer*. This mechanism is implemented in a software module called *packet prefetcher*.

HFS provides two contributions to energy reduction: first, by increasing the throughput, it increases the number of packets successfully transmitted per energy consumed (the number of retransmission is also lower), second, according to our experiments, the time and energy needed to execute HFS for each packet enqueue/dequeue are close to those of just DRR. Moreover, while still reducing energy consumption and boosting the throughput, we show also that HFS preserves high QoS guarantees.

### 1.2 Organization of the document

In Section II we describe the architecture proposed in [13]. In Section III we show the testbed deployed to validate the efficiency of the architecture. Then we show HFS, a new packet scheduler for wireless links based on such architecture and implemented in such testbed in Section IV. In Section V, we validate HFS performance by comparing it with the best high-performance schedulers for wired links and with the best integrated scheduler for wireless links, showing experimental results. Finally, in Section VI we highlight our conclusions.

## 2. Architecture

The architecture proposed in [13] is shown in Figure 1 for a system containing only one outgoing link. The architecture is made of two layers: a *QoS Provisioning Layer*, hereafter called just QoS layer for short, and a *MAC Scheduling&Abstraction Layer*, hereafter abbreviated as MAC-SAL. The purpose of the first layer is putting together, conceptually, the two most important components for providing service guarantees over a transmission link: the packet classifier and the packet scheduler. Packets are passed from the QoS layer to the MAC-SAL by a *packet prefetcher*, described in detail in Subsection II-C. A generalization of this architecture for a multi-link system is also discussed in [13].

### 2.1 QoS layer

The QoS layer is shown in the top box in Figure 1. We add the prefix *QoS* to the name of both the classifier and the scheduler in the QoS layer to highlight the goal of these component, and to distinguish them from the corresponding components in the MAC-SAL (described in the next subsection). As for the classifier, it divides packets into $N$ separate flows, so that individual bandwidth and delay guarantees can be provided to each flow. Finally, the purpose of the scheduler is to enforce a scheduling policy that does provide the desired guarantees.

To complete the description of the QoS layer, consider the path followed by a packet passed to this layer by invoking the *send*() interface function (top left corner in Figure 1). The packet first enters the QoS classifier, which determines the QoS flow it belongs to; then it is inserted into the queue associated to the flow. A possible additional intra-flow scheduler determines the order by which the packets within each flow must be served. The simplest order is certainly FIFO. The next packet to serve can be requested to the QoS layer by invoking the *get_next_pkt*() function. On the invocation of this function, the next packet is chosen among the head-of-line packets of the QoS flows. This packet is then removed from the queue and passed to the entity that invoked the function.

### 2.2 MAC Scheduling&Abstraction layer

The MAC-SAL has the same structure of the QoS layer. What changes is the purpose of the components, and of the layer as a whole. In particular, on one side the MACSAL interacts with the underlying MAC layer and deals with the details of the wireless technology at hand. The possible information received by the MAC-SAL from the MAC layer are reported on the right of the box representing the MACSAL in Figure 1. On the other side, the MAC-SAL exports an interface made by two functions: *MAC-SAL-send* () and *link_ready* (), the latter informs when the abstract link is ready.

The MAC-SAL may serve various important purposes. First, if required by the application at hand, it may make sure that packets get eventually transmitted successfully. Second, it may implement algorithms for maximizing the link throughput. As said in the introduction, to achieve the latter goal the MAC-SAL must classify packets according to their chances of successful transmission, and must be able to change the order by which packets are sent to the MAC layer. In the end, although with a different goal, the MAC-SAL must, in general, accomplish the same main sub-tasks as the QoS layer: divide packets into distinct flows, store the packets of each flow in a distinct queue and schedule the head-of-line packets of the flows according to the desired policy.

### 2.3 Packet prefetcher

The higher is the number of non-empty MAC (flow) queues, the higher is the number of head packets among which the MAC scheduler can choose the next packet to transmit. Hence, the higher is the probability that the MAC scheduler can pick good packets with respect to the goals it wants to achieve. For example, suppose that the goal of the MAC scheduler is to keep a high throughput and that some MAC queues contain packets for destinations with bad channel conditions. If many/few other queues are not empty, then the probability for the scheduler to have at its disposal better packets to transmit is high/low. In the end, to maximize the effectiveness of the MAC-SAL, its queues should be always kept as full as possible. This is exactly the purpose of the *packet prefetcher* depicted in the middle of Figure 1.

The behavior of the prefetcher depends on a parameter $Q$, measured in number of bytes. We say that the prefetcher prefetches a packet if it invokes the function *get_next_pkt* () to get the next packet, say *pkt*, from the QoS layer and then invokes *MAC-SAL-send* (*pkt*) to insert the packet in the MACSAL. We assume that a prefetched packet is never dropped by the MAC-SAL. Finally, we denote as $S(t)$ the sum of the sizes of the packets queued in the MAC-SAL at time $t$.

The prefetcher operates on each of the following two events: when a new packet arrives in the QoS layer and when a new packet is dequeued from the MAC-SAL. On the first event, let *pkt* be the new arriving packet. If $S(t) < Q$ when *pkt* arrives, then the packet prefetcher immediately prefetches *pkt*. On the other event, if $S(t)$ becomes lower than $Q$ when the packet is dequeued from the MAC-SAL, then the packet prefetcher starts prefetching new packets until $S(t) < Q$ does not hold any more. In other words, the combination of the set of queues in the MAC-SAL and of the packet prefetcher implements a *shared buffer* with

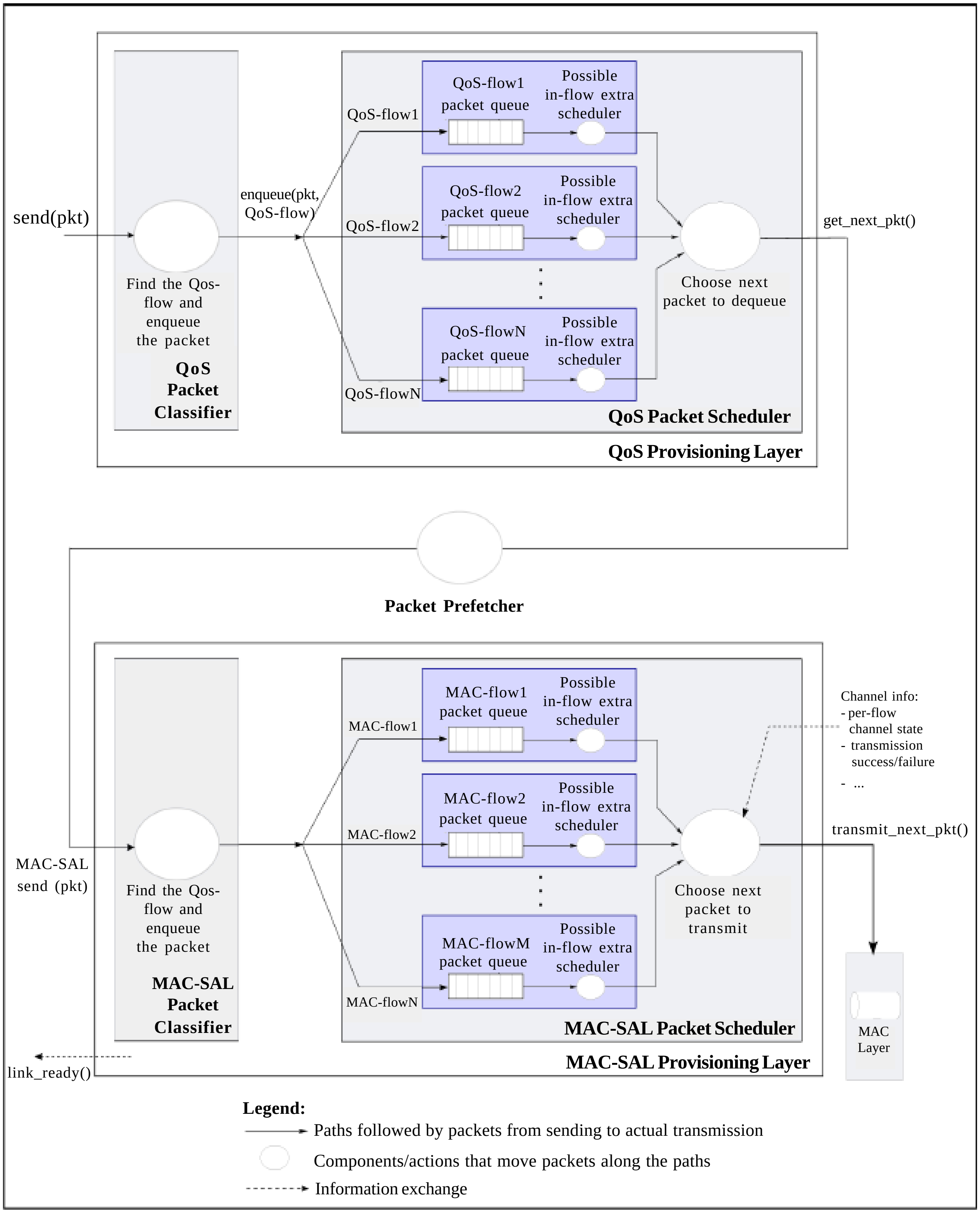

Figure 1. Modular architecture for providing QoS over a wireless link

*virtual queueing*, a device in which the memory used to store the packets is shared and the queues are only virtually separated.

We conclude by noting that this scheme is much more effective than an approach with distinct queues and with the same total memory. As for the second scheme, suppose for example that all the queues have the same length, equal to $Q/M$ plus one maximum packet size L. If all the packets prefetched in a given time interval are destined to the same MAC queue, then the queue becomes full only after $Q/M + L$ bytes have been prefetched. If the next packet to prefetch is again for the same queue, then the prefetcher must block. In the shared-buffer scheme the prefetcher must block only after $Q + L$ bytes have been prefetched, which increases the probability that more queues are not empty.

## 3. Testbed

In this section we present the test environment used to easily deploy, test, modify and validate flexible and efficient packet schedulers for wireless links as well as scenarios to model the radio channel characteristics.

We ran our experiments using the test environment [14], which is an improved version of the original test environment [15], extended to test schedulers over the modular architecture solution showed in Section II and to simulate also wireless link characteristics. With the help of this flexible tool it is possible to measure and compare one against each other the real execution time, simulated throughput and QoS performance of different schedulers. An existing packet scheduler can be easily plugged into this environment, after at most some little interface changes. Inside the environment, the scheduler can then be exercised with the desired sequence of enqueue/dequeue requests, through a *controller* that iteratively switches between two phases: an enqueue phase in which it generates fake packets by picking them from a free list, and a dequeue phase in which it dequeues packets from the scheduler and reinserts them into the free list. The switch occurs according to two configurable max-total-backlog and min-total-backlog thresholds.

### 3.1 Configuration

Each test consisted of 10M[1] events with a typical balancing of 5M packet enqueues and 5M packet dequeues, with the controller configured so as to let flows oscillate between null backlog and a backlog of 10 packets each. Such an enqueue/ dequeue pattern happened to be the most demanding one for the schedulers. Packets had a fixed size of about 1700 bytes, with no cache-line alignment. The payload of the packets was never either read or written. No migration and no packet drop occurred in any run.

### 3.2 Arrival Pattern

The controller switches between enqueue and dequeue mode to control the number of pending packets of each run. Packets arrival pattern is set in a *bursty* manner: the packets generated by the controller are sent to the flows with bursts that have random size proportional to the flow weight. Also a single threshold per-flow has been setted, if a flow drops below a certain threshold of filling, the controller refill it with a given probability.

### 3.3 Wireless scenarios

This tool can be executed to simulate a wireless link. To run our tests over the modular architecture we considered a specific scenario described here with its specific well known wireless parameters simulated, that is:

• Total bandwidth: 54Mb/s;

• 20 Subscriber Stations (SS);

• 2.7Mb/s per SS;

• For each SS, 5 flows:

– One with ~1.6Mb/s reserved for video or VoIP (20 flows with weight 20);

– Another with ~ 0.8Mb/s reserved forWEB browsing or direct downloads (20 flows with weight 10);

– The other three with ~ 0.1Mb/s reserved for download/ sharing (3.20 flows with weight 1).

As for base stations channel condition, we set for each of them a packet loss probability ($P_{loss}$) to emulate different user

[1]The package of the test environment [14] contains the script, run_test.sh, that we used to run the tests.

conditions. In fact, we let $P_{loss}$ ranging linearly from $10^0$ to $10^{-1}$ and we considered also $P_{loss}$ of $10^{-2}$, $10^{-3}$ and $10^{-4}$. We considered static channel condition in our tests to evaluate possible phenomena for different user conditions. From an user classification point of view, we define a threshold to distinguish user in *good* and *bad* channel condition. This threshold is placed at 20% of packet loss. Users/flows with less or equal to 20% of packets lost are considered as users/flows in good conditions, while users/flows with more than 20% of packets lost are considered as user/flows in bad conditions. This because above 20% most applications do not work properly; for instance, both the TCP window and the VoIP controller mechanism do not perform well under the aforementioned condition.

## 3.4 Statistics

To measure, simulate and validate schedulers' performance we implemented different well-known metrics. We used all of these indicators to easily design, test and validate different packet scheduler solutions. Here is a list of the main performance metrics and a detailed description:

• **Throughput:** To measure the throughput we simulated the *normalized throughput* achieved by the schedulers. This parameter is the amount of successfully transmitted packets for each flow divided by the amount of total sent packets. We computed the normalized throughput as

$$thr \equiv \frac{\sum_i pkt_{sent_i}(1 - P_{loss_i})}{\sum_i pkt_{sent_i}}$$

where $pkt_{sent_i}$ is the number of packets sent by the flow *i*, $P_{loss_i}$ is the packet loss probability of the flow *i* and $pkt_{sent_i}(1 - P_{loss_i})$ is the number of successfully transmitted packets by the flow *i*;

• **Execution time:** How to measure the execution time? At the end of each run, we measured the total execution time of the run and divided it by the total number of enqueues, or equivalently of dequeues executed. We obtained therefore the average total packet-processing time, i.e., the execution time of an enqueue plus a dequeue, inclusive also of the cost of generating and discarding an empty, fixed-size packet;

• **Energy consumption:** According to the models in [16], [17], lower/higher relative execution times imply also lower/higher relative energy consumptions. Moreover, the increase of the throughput imply the increase of the number of packets successfully transmitted per energy consumed. In this sense we will consider the energy consumption level according to the execution time and the throughput level;

• **Time Worst-case Fair Index (T-WFI):** Another useful QoS guarantees metric is the T-WFI, which allows to evaluate, in a single value, both fairness and delay: on one hand it shows the fairness, in terms of delay from the worst-case completion time of a packet in a perfectly fair system, while on the other hand it allows to instantly calculate the actual delays incurred by packets depending on the occupation of queues. In a perfectly fair system, the worst-case completion time of a packet is equal to its queue length (including the packet itself) divided by the packet's flow guaranteed ratio. Assuming that the link rate *R* is constant, we measured *T-WFI* $^k$ for flow *k* as:

$$T-WFI^k \equiv max\left(t_{deq} - t_{enq} - \frac{Q^k(t_{enq})}{\phi^k R}\right)$$

where $t_{deq} - t_{enq}$ is the delay experienced by a packet *pkt*, $t_{enq}$ is the number of packets dequeued by the system when *pkt* is enqueued, while $t_{deq}$ is the number of packets dequeued by the system when *pkt* is dequeued. $Q^k(t_{enq})$ is the backlog of flow *k* just after the arrival of the packet and *k* is the relative weight of the flow;

• **Bit Worst-case Fair Index (B-WFI)**
This metric indicates how much, in terms of service received, a flow remains behind what it would receive on an ideal scheduler. We measured *B-WFI* $^k$ for flow *k* as:

$$B\text{-}WFI^k \equiv \max_{[t_1,\, t_2]}\{\phi^k W(t_1, t_2) - W^k(t_1, t_2)\}$$

where $[t_1, t_2]$ is any time interval during which the flow is continuously backlogged, $\phi^k W(t_1, t_2)$ is the minimum amount of service the flow should have received according to its share of the link bandwidth, and $W^k(t_1, t_2)$ is the actual amount of service provided by the scheduler to the flow. In particular, $W(t_1, t_2)$ is measured as the number of dequeue event computed by the scheduler between $t_1$ and $t_2$, while $W^k(t_1, t_2)$ is measured as the number of dequeue event computed by the scheduler only for the flow $k$.

### 3.5 Test equipment

We ran our tests on two systems with the following software and hardware characteristics:

• Ubuntu 12.04.2, 64-bit kernel 3.2.0, Intel Core Dual-E2200 @ 2.20GHz, gcc 4.6.3 -O3;

• OS X 10.7.5, Darwin 11.4.0, Intel Core i5-2557M @ 1.8 GHz, gcc 4.2.1 -O3.

Since the relative performance of the schedulers were about the same on the two systems, we report our results only for the first system. In the next two sections we first show how we used the test environment described here to define HFS packet scheduler, and then we show how we validated its performance.

## 4. HFS

Once described the architecture, it is easy to describe HFS. We considered the best schedulers for wired links available in the literature, that is:

• **WF$^2$Q+:** An optimal service guarantees with $O(logn)$ complexity [5];

• **DRR:** A scheduler with extremely low time complexity $O(1)$, but with $O(n)$ deviation form optimal service [7];

• **QFQ+:** A quasi-optimal service guarantees scheduler with execution time close to the DRR one [18].

To develop HFS we combined these schedulers placing them at QoS layer and/or at MAC-SAL layer. For a matter of space we do not show all the results of these combinations (we have left a detailed analysis of all solutions in a different work), but we present directly our solution with QFQ+ both at QoS and MAC-SAL layer. QFQ+ achieves quasi-optimal service guarantees at QoS level while still preserving low execution time overhead, on the other side QFQ+ placed at MAC-SAL layer achieves high-throughput with, also in this case, a low time complexity. In brief, to understand how HFS is composed by, just look at the Figure 1 and place QFQ+ at the end of both layers inside the packet scheduler box. We validated HFS in Section V comparing its performance with the best existing schedulers for wired networks and with the best integrated one for wireless links, in particular we compared its execution time against DRR, its throughput level against W$^2$F$^2$Q+ and its service guarantees against WF$^2$Q+.

## 5. Results

Here we validate the efficiency of HFS packet scheduler by comparing its performance with the best packet schedulers for wired links and with the best integrated scheduler for wireless links. To validate HFS results we considered different schedulers as a benchmark:

• **W$^2$F$^2$Q**, to validate HFS throughput results against the best integrated scheduler available in the literature [19];

• **DRR**, to validate HFS execution time against the best high-performance scheduler for wired links in terms of time complexity;

• **WF$^2$Q+**, to validate HFS guarantees against the best high-performance scheduler for wired links in terms of service guarantees.

Let us call *double-SCHED* a double instance of the scheduler *SCHED* compliant with the modular architecture with *SCHED* placed both at QoS and at MAC-SAL layer (e.g. double-DRR is obtained placing DRR scheduler both at QoS level and at MAC-SAL level)[2]. Similarly, we will call *single- SCHED* a single instance of the scheduler *SCHED* compliant with the classical architecture.

[2]Following this nomenclature HFS is exactly the same of double-QFQ+.

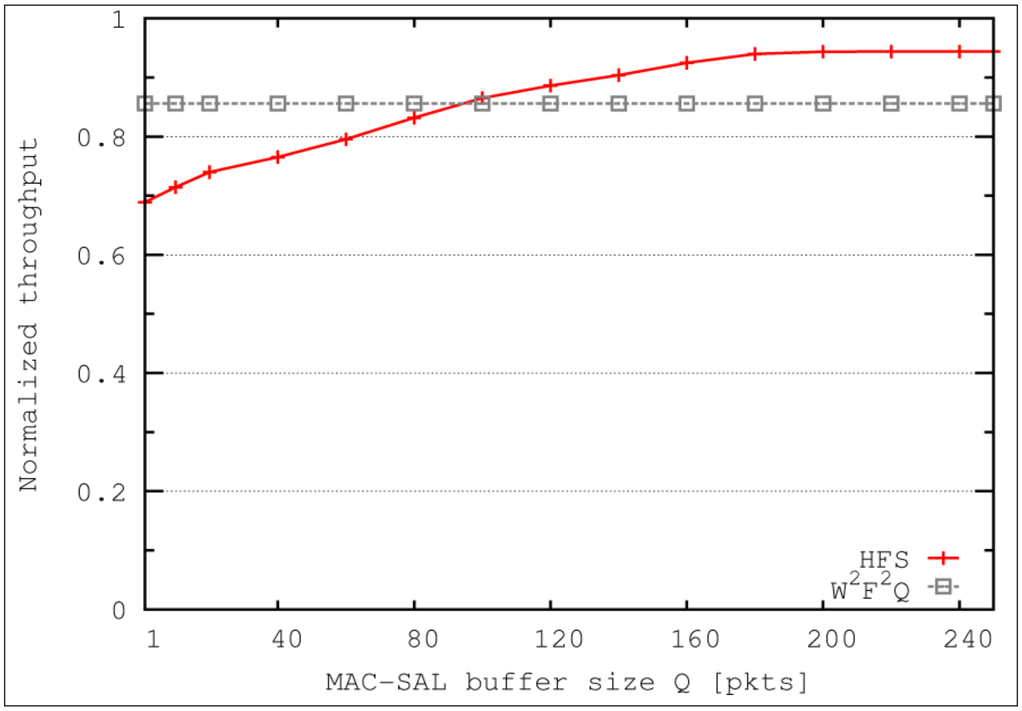

Figure 2. Normalized throughput for HFS and W$^2$F$^2$Q

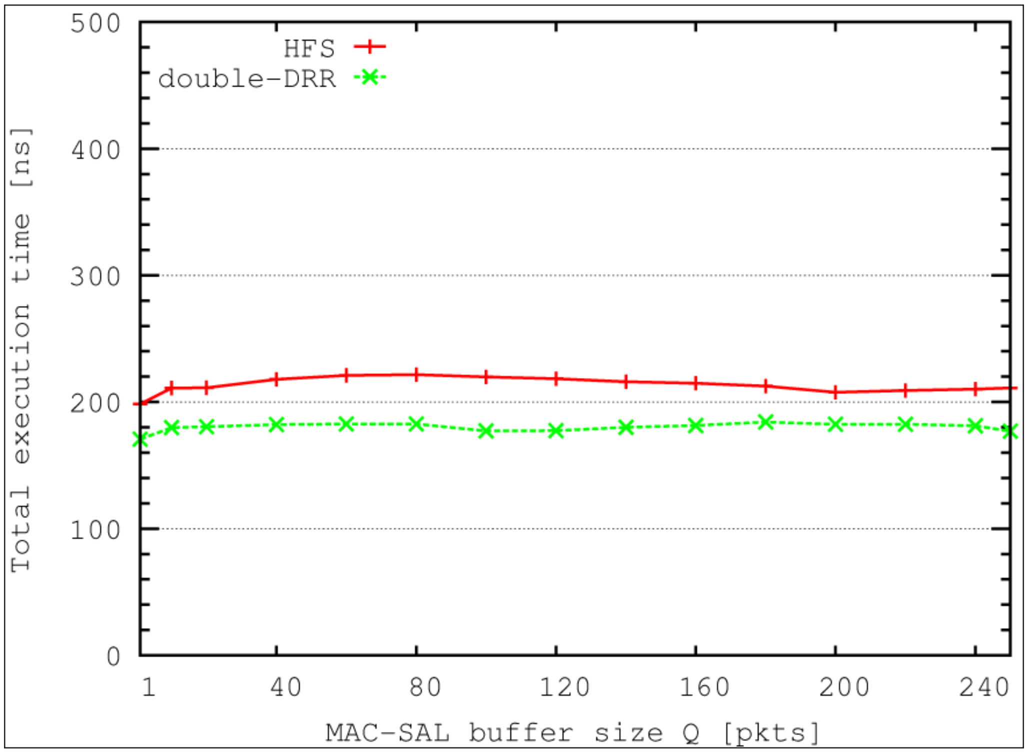

Figure 3. Execution time of HFS against double-DRR

**5.1 Throughput**

The integrated scheduler W$^2$F$^2$Q models temporary bursty channel errors of a wireless link only, with a two-state Markov chain, to simulate flows in good or bad channel conditions. Unfortunately, such a distinction does not hold in our model, since we considered a scenario in which a flow can experience long term static channel conditions as well, based for instance on the

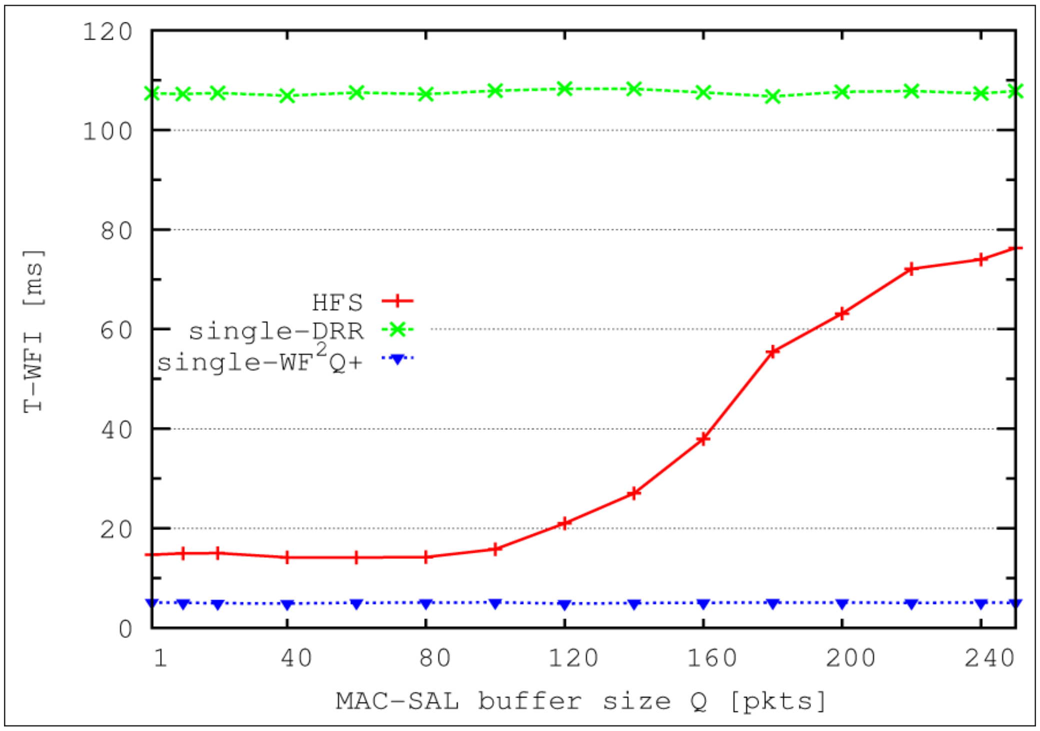

Figure 4. T-WFI for HFS, WF2Q+ and DRR, for flows in *intermediate conditions*

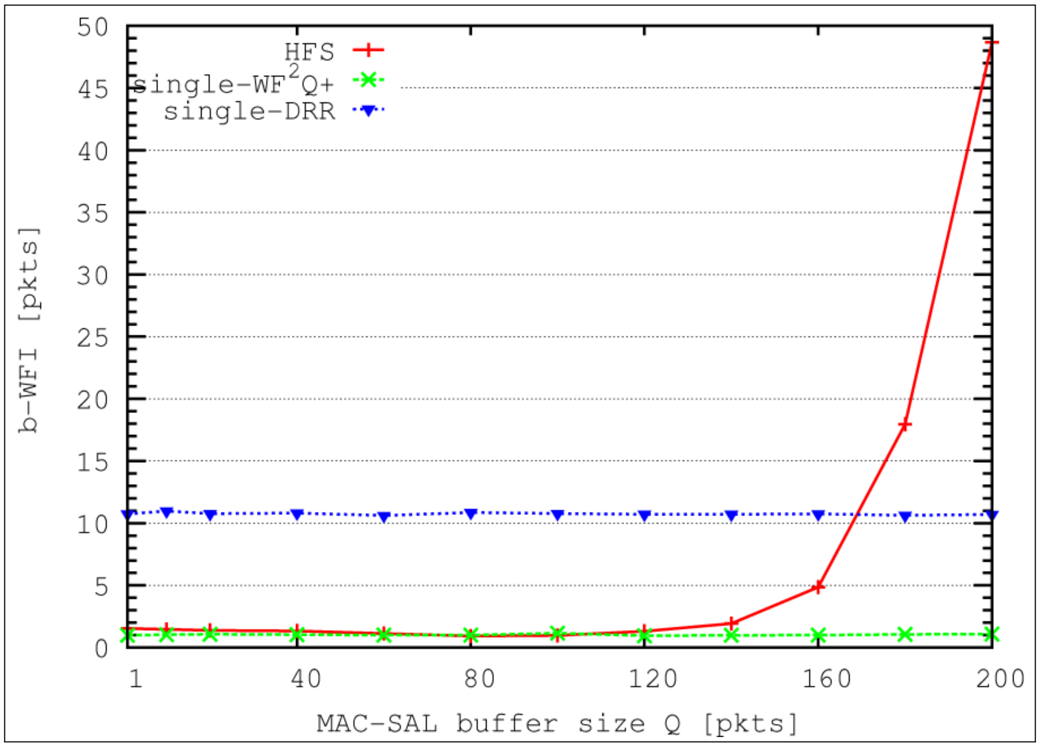

Figure 5. B-WFI for HFS, WF2Q+ and DRR, for flows in *intermediate conditions*

position of the user. Anyway, comparing $W^2F^2Q$ with HFS can benefit the first in terms of throughput, since users in good state manage the total bandwidth available leaving flows in bad state with null weight. We simulate this kind of event by using a threshold in our environment, where flows with more then 20% of packet loss are classified as flows in bad state with weight equal to zero, while the other flows are classified as flows in good state with weight equal to their own weight according to the weight distribution policy. Figure 2 shows how in this favourable case $W^2F^2Q$ scheduler obtains high normalized throughput. However, HFS achieves better throughput performance with respect to $W^2F^2Q$ for Q  100 due to his fine-grained choice among good flows, which increases with the MAC-SAL shared buffer size Q.

**5.2 Execution time**

To evaluate HFS in terms of execution time (and energy consumption) we use as benchmark the scheduler double- DRR, which is the simplest possible solution to achieve high throughput preserving QoS guarantees with a low computational cost of $O(1)$ with our modular architecture. Figure 3 shows that HFS has a really close execution time to the double instance of DRR which is the best high-performance scheduler for wired links in terms of execution time. Furthermore, the picture shows that the packet-processing time of HFS is compliant with the link rate (i.e. much lower than the packet transmission time). Why we do not considered $W^2F^2Q$ in the execution time graph? $W^2F^2Q$ scheduling complexity is $O(N)$, too high for schedulers in backbone network where N is very large and more than the HFS scheduling complexity which is close to the DRR optimal cost of $O(1)$.

**5.3 Energy consumption**

According to the models in [16], [17], lower/higher relative execution times imply also lower/higher relative energy consumptions. It is the case of HFS, as showed in Figure 3, with a really close execution time to DRR which is the best high-performance packet scheduler for wired links in terms of execution time. Moreover, in Figure 2 we show that HFS achieves also higher throughput then the best integrated packet scheduler for wireless links, which is $W^2F^2Q$. In this way, by increasing the throughput, HFS increases the number of packets successfully transmitted per energy consumed (the number of retransmission is also lower). Therefore, reducing the execution time and boosting the throughput permits to HFS to reduce the energy consumption.

**5.4 QoS guarantees**

As regards QoS metric, we show first Time Worst-case Fair Index because it allows to evaluate, in a single graph, both fairness and delay: on one hand it shows the fairness, in terms of delay from the worst-case completion time of a packet in a perfectly fair system, while on the other hand it allows to instantly calculate the actual delays incurred by packets depending on the occupation of queues. In a perfectly fair system, the worst-case completion time of a packet is equal to its queue length (including the packet itself) divided by the packet's flow guaranteed ratio. Figure 4 validates the performance of HFS in terms of QoS for flows in *intermediate conditions*, i.e. for flows/users with at least 0.8 of normalized throughput per-flow (HFS guarantees this kind of worst case performance if the station looses less or equal to 20% of packets).With a packet loss ratio above 20% most applications do not work properly; for instance, both the TCP window and the VoIP controller mechanism do not perform well under the aforementioned condition. Why we do not considered W2F2Q in QoS-guarantees graph? As we have said, the algorithm $W^2F^2Q$ model temporary bursty channel errors of a wireless link only. This way the scheduler tries to implement the *compensation* mechanism to guarantees long term fairness between different flow-condition services under the hypothesis that an error-prone flow has sufficient time to make up for their lag after recovery of channel (see [19]). Unfortunately, such assumption does not hold in our model because we considered a scenario in which a flow can experience also long term static channel condition, based on the position of the user. In this sense, comparing $W^2F^2Q$ with our model from a QoS guarantees point of view can be unfair because users/flows classified in bad state can lag forever with quasi-zero service guarantees.

Considering the same scenario described to evaluate the Time Worst-case Fair Index graph, Figure 5 shows the performance of HFS scheduler in terms of Bit Worst-case Fair Index, which is a QoS metric useful to indicate how much, in terms of service received, a flow remains behind what it would receive on an ideal GPS scheduler. Also in this case the performance of HFS is comparable to the optimal one of $WF^2Q+$ with a MAC-SAL buffer size Q lower than 160 packets. Increasing the size of the buffer of the MAC-SAL scheduler implies enhancing the fine grain choice of HFS among flows in better channel conditions, and the B-WFI parameter tends to grow. By comparing Figure 2 and 4 we can say that it is possible to choose the desired trade-off between throughput-boosting level and granularity of the service guarantees, by only setting the parameter Q. For instance, by setting a value of Q equal to 100 packets, HFS reaches a normalized throughput close to 90%, greater then $W^2F^2Q$ one, while still preserving service guarantees close to the optimal ones of $WF^2Q+$.

## 6. Conclusions

In this paper we have validated a general, modular architecture for decoupling the task of providing QoS guarantees from the task of dealing with the idiosyncrasies of a wireless link. We defined a new test-environment which easily permits to test existing packet schedulers and analyze the result in terms of execution time, QoS guarantees and also throughput achieved over wireless links. We also define HFS, a new flexible, efficient and green packet scheduler for providing low energy consumption, high throughput and QoS guarantees over wireless links. As a validation of HFS we show, through experimental results, the high performance of this new packet scheduler with execution time and energy consumption close to that of just a Deficit Round Robin, higher throughput than the best integrated scheduler Wireless Worst-case Fair Weighted Fair Queueing and accurate fairness and delay guarantees close to the optimal ones of Worst-case Weighted Fair Queueing.

## References

[1] Ng, T. S. E., Stoica, I., Zhang, H. (1998). Packet fair queueing algorithms for wireless networks with location-dependent errors, *In*: Proceedings of IEEE INFOCOMM 98, 3 (c) 1103–1111.

[2] Lu, S., Bharghavan, V., Srikant, R. (1999). Fair scheduling in wireless packet networks, *IEEE/ACM Trans. Netw.*, 7 (4) 473–489.

[3] Yi, Y., Seok, Y., Kwon, T., Choi, Y., Park, J. (2000). W2f2q: packet fair queuing in wireless packet networks, *In*: Proceedings of the 3rd ACM international workshop on Wireless mobile multimedia, ser. WOWMOM '00. New York, NY, USA: ACM, p. 2–10.

[4] Iera, A., Molinaro, A., Pizzi, S. (2007). Channel-aware scheduling for qos and fairness provisioning in ieee 802.16/wimax broadband wireless access systems, *IEEE Network*, 21 (5) 34–41.

[5] Bennet, H., Jon, C. R., Zhang. (1997). Hierarchical packet fair queueing algorithms, *IEEE/ACM Transactions on Networking*, 5 (5) 675–689.

[6] Valente, P. (2007). Exact gps simulation and optimal fair scheduling with logarithmic complexity, *IEEE/ACM Transactions on Networking*, 15 (6) 1454–1466.

[7] Shreedhar, M., Varghese, G. (1995). Efficient fair queuing using deficit round robin, *IEEE/ACM Transactions on Networking*, p. 375– 385.

[8] Guo, C. (2001). SRR: An O (1) time complexity packet scheduler for flows in multi-service packet networks, *In*: *ACM SIGCOMM*, p. 211–222.

[9] Ramabhadran, S., Pasquale, J. (2006). The stratified round robin scheduler: design, analysis and implementation, *IEEE/ACM Transactions on Networking*, 14 (6) 1362–1373.

[10] Yuan, X., Duan, Z. (2009). Fair round-robin: A low complexity packet scheduler with proportional and worst-case fairness, *IEEE Transactions on Computers*, 58 (3).

[11] Stephens, D. C., Bennett, D. C., Zhang, H. (1999). Implementing scheduling algorithms in high-speed networks, *IEEE Journal on Selected Areas in Communications*, 17 (6) 1145–1158, Jun.

[12] Karsten, M. (2010). Approximation of generalized processor sharing with stratified interleaved timer wheels, *IEEE/ACM Transactions on Networking*, 18 (3) 708–721, June.

[13] Casoni, M., Paganelli, A., Valente, P. (2013). A modular architecture for qos provisioning over wireless links, *In*: Proc. of the 8th IEEE International Workshop PAEWN, Barcelona, (Spain).

[14] http://algogroup.unimore.it/people/paolo/agg-sched/test-env.tgz.

[15] http://info.iet.unipi.it/'luigi/papers/20100210-qfq-test.tgz.

[16] Bartolini, A., Cacciari, M., Tilli, A., Benini, L. (2011). A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores, in *Design, Automation Test in Europe Conference Exhibition* (*DATE*), p. 1–6.

[17] Sadri, M., Bartolini, A., Benini, L. (2011). Single-chip cloud computer thermal model, in *Thermal Investigations of ICs and Systems* (*THERMINIC*), 17th *International Workshop on*, p. 1–6.

[18] Valente, P. (2013). Providing near-optimal fair-queueing guarantees at roundrobin amortized cost, *to appear in the Proc. of the 22nd International Conference on Computer Communications and Networks*, ICCCN.

[19] Yi, Y., Seok, Y., Kwon, T., Choi, Y., Park, J. (2000). W2f2q: packet fair queuing in wireless packet networks, in *WOWMOM*, p. 2–10.