Reducing the Execution Time of Fair-Queueing Packet Schedulers

Paolo Valente

Department of Physics, Computer Science and Mathematics, University of Modena and Reggio Emilia, Modena, Italy

Abstract

Deficit Round Robin (DRR) is probably the most scalable fair-queueing packet scheduler. Unfortunately, it suffers from high delay and jitter with respect to a perfectly fair (and smooth) service. Schedulers providing much better service guarantees exist, but have a higher computational cost.

In this paper we deal with this issue by proposing a modification scheme for reducing the amortized execution time of the latter, more accurate schedulers. Modified schedulers preserve guarantees close to the original ones, and can also handle seamlessly both leaves and internal nodes in a hierarchical setting.

We also present Quick Fair Queueing Plus (QFQ+), a fast fair-queueing scheduler that we defined using this scheme, and that is now in mainline Linux. On one hand, QFQ+ guarantees near-optimal worst-case deviation with respect to a perfectly fair service. On the other hand, with QFQ+, the time and energy needed to process packets are close to those needed with DRR, and may be even lower than with DRR exactly in the scenarios where the better service properties of QFQ+ make a difference.

Keywords: Packet scheduling, QoS, Fair queueing, Computational cost

1. Introduction

Most network applications require or benefit from their packet flows being guaranteed a given share of the link bandwidth. In addition, packet delays and packet delay variations play a critical role with time-sensitive applications. In this respect, a well-known figure of merit related to delay variations is packet *jitter*, commonly, but not uniquely, defined as the average deviation from mean delay.

In many contexts, packet schedulers are a necessary component for providing applications with service guarantees exactly in terms of the above figures of merit. As a first example, consider the public Internet. On one side, both the total IP traffic and, in particular, the component of time-sensitive traffic, are growing and expected to continue to grow at a high rate [1]. On the other side, to reduce costs, operators strive to achieve their target service levels with as little resources as possible. In other words, operators tend to provide available bandwidths close to offered loads, often by keeping active only the minimum number of links needed.

Considering also the magnitude of the fluctuations of arrival rates in a complex network like Internet, it follows that the links of, e.g., DiffServ-compliant Internet routers [2] are likely to experience (increasing) saturation periods. Being the offered load of EF queues an important, and growing, portion of the total load on these links [1], EF queues may then become long. In the end, some unlucky EF flows may easily experience high packet delays/jitters if, e.g., some other flows have a bursty behavior and no intra-class scheduling is performed[1].

Besides, given the continuous growth of time-sensitive traffic, and hence of its variance, a fixed inter-class bandwidth distribution may be a too rigid scheme. It might be instead more appropriate to let the cumulative fraction of the bandwidth devoted to time-sensitive applications dynamically grow with the actual cumulative demand of these applications. A simple way to achieve this goal is to use just a flat weighted *fair-queueing* discipline, i.e., a discipline where each flow is assigned a weight, and that tries to provide each flow with a fraction of the link bandwidth proportional to the weight of the flow. Flows in the EF class could then be simply assigned a higher weight than the other flows.

There are then contexts where stronger, and often tighter, bandwidth and packet-delay/jitter guarantees must be provided by contract. Examples are managed networks for implementing IPTV services, or, at a smaller scale, automotive and avionics local communication networks. In particular, the latter local networks are being used more and more to carry both safety-critical and infotainment traffic. In this respect, the ideal way both to prevent infotainment traffic from interfering dangerously with safety-critical traffic, and to meet the typical requirements of infotainment traffic, is to guarantee the latter

Email address: paolo.valente@unimore.it (Paolo Valente)

[1]Packet-dropping disciplines are of paramount importance as well, but they are out of the scope of this paper.

both a minimum and a maximum (capped) bandwidth, plus a small-enough packet delay/jitter.

Also in these contexts with stronger requirements, bandwidth over-provisioning may not be an option, because of costs, power consumption and other technical issues. As a consequence, the use of schedulers providing tight bandwidth and packet-delay guarantees may make the difference between feasibility and unfeasibility of the target time-sensitive applications.

Unfortunately, systems with a low computational power may have difficulty executing (also) packet schedulers at line rate, whereas, on high-speed systems, scalability issues arise when the number of flows and the speed of the outgoing link grow. On the other hand, recent results on fast packet I/O [3] reduce packet-processing time at such an extent that millions of packets per second can be easily processed on commodity hardware. Such a dramatic improvement leaves more time for packet scheduling. Only very efficient schedulers can however comply with the extremely short packet transmission times of a 1-10 Gbit/s link.

Deficit Round Robin (DRR) [4] is probably one of the best candidates to keep up with line rate, both on slow and high-speed systems. In fact, DRR is one of the simplest fair-queueing schedulers providing strong service guarantees. Unfortunately, if packet sizes and/or flow weights vary, then DRR suffers from a high worst-case packet delay and jitter with respect to an ideal, perfectly fair (and smooth) service. As shown in Section 6, in simple, realistic scenarios where packet sizes and flow weights vary, both packet delay and delay variation can easily be so high to make time-sensitive applications unfeasible, even if their bandwidth requirements are fully met.

On the opposite side, several accurate fair-queueing schedulers do not suffer from this problem, as they guarantee near-optimal deviation with respect to the above ideal service (with any packet-size and flow-weight distribution). Some of these schedulers are also quite efficient [5, 6, 7]; but even the fastest of them, namely Quick Fair Queueing (QFQ) [5], is at least twice as slow as DRR. In this paper we try to fill this gap with a solution that allows near-optimal service guarantees to be provided at a computational cost comparable or even lower than that of DRR[2].

Contribution

We propose a general modification scheme for fair-queueing schedulers, in which packet flows are grouped into *aggregates*, and the original, costly operations of the schedulers are used to schedule aggregates and not single flows. Inside aggregates, flows are scheduled with DRR. Modified schedulers are also ready to schedule internal nodes in a hierarchical setting (see Section 7 for a comparison against classical hierarchical schedulers).

Denoted as M the maximum number of flows in an aggregate, the scheme that we propose enjoys the following key property: during full-load periods, the higher M is, the longer each aggregate is served before the costly operations of the original scheduler are executed. Hence the closer the amortized execution time of the modified scheduler becomes to that of DRR.

Of course, the higher M is, the more service guarantees deviate from the original ones. In this respect, in this paper we also compute the guarantees provided by the modified scheduler in case the original one belongs to the family of the fair-queueing schedulers providing optimal or near-optimal worst-case service guarantees [9, 10, 6, 7, 5]. In particular, we show how little the QoS degradation is, even for values of M for which the execution-time cut is close to its maximum.

In practical terms, the main information conveyed by the formulas reported in this paper is that both the original and the modified schedulers always guarantee a smooth service, even in a possible scenario where the theoretical bounds are actually reached. On the opposite side, we also show bounds for DRR, which highlight that DRR suffers from serious packet-delay and jitter problems. Our experimental results confirm all these facts.

Finally, we describe Quick Fair Queueing Plus (QFQ+), a new scheduler that we have defined by applying this scheme to QFQ [5], and that we have implemented in Linux (QFQ+ replaced QFQ in mainline Linux from 3.8.0). Through QFQ+, we complete the analysis of QoS degradation with a concrete example, by comparing the service guarantees of QFQ+ against those of QFQ and DRR, analytically and experimentally. We also compare the efficiency (execution time, energy consumption, ...) of these schedulers experimentally. The gist of our results is that with QFQ+ the time and the overall system energy needed to process packets is definitely lower than with QFQ, and may be even lower than with DRR exactly in the scenarios where the accurate service guarantees of QFQ+ make a difference with respect to DRR.

The last, apparently surprising result is a consequence of a more general fact highlighted by our experiments: the order in which a scheduler dequeues packets influences the execution time and the energy consumption of both the scheduler itself and the other tasks involved in processing packets.

Organization of this document

In Section 2 we describe the modification scheme in detail, whereas in Section 3 we introduce QFQ+. In Section 4 we show the general service guarantees provided by modified schedulers, which we prove then in Section 5. In Section 6 we instantiate these guarantees for QFQ+. Finally, after comparing the contributions provided in this paper against related work in Section 7, we report our experimental results in Section 8.

[2]Part of the material presented in this paper can also be found in our preliminary work [8].

2. Modification scheme

Symbol	Meaning
$*^k$	A superscript indicates a quantity related to an aggregate
$*_i$	A subscript indicates a quantity related to a flow
N	Total number of flows
M	Maximum number of flows in an aggregate
k	Aggregate index
m^k	Current number of flows in the k-th aggregate
L, L^k	Max size of any packet in the system or in the k-th aggregate, in bits
Q	Size of the transmit queue in bits, plus L
R	Rate of the link for a constant-rate link
ϕ_i	Weight the i-th flow
ϕ^k	Weight of each flow in the k-th aggregate
ϕ_{min}	Minimum weight among all flows
$*(t_1, t_2)$	Given a generic function $*(t)$, the notation $*(t_1, t_2) \equiv *(t_2) - *(t_1)$ indicates the difference between the values of the function in t_2 and t_1
$W(t)$, $W^k(t)$, $W_i(t)$	Number of bits transmitted (globally, for the k-th aggregate, or for the i-th flow) in $[0, t]$
$\overline{W}(t)$, $\overline{W}^k(t)$, $\overline{W}_i(t)$	Number of bits dequeued, i.e., sum of the sizes of the packets dequeued from the scheduler (globally, for the k-th aggregate, or for the i-th flow) in $[0, t]$
$B_i(t_a)$	Backlog of the i-th flow right after time t_a
ΔS^k	$\Delta S^{k-} + \Delta S^{k+}$, where ΔS^{k-} and ΔS^{k+} are the two timestamp-error terms defined in Appendix A.

Table 1: Definitions of the symbols used in the paper.

In this section we show the modification scheme and discuss its main benefits. For the reader convenience, the notations used in this paper are also reported in Table 1. Besides, for ease of presentation, hereafter we call SCHED a generic fair-queueing scheduler to which our modification scheme is applied, and SCHED+ the resulting new scheduler. The modification scheme described in the rest of this section is depicted in Figure 1. As already said, SCHED+ groups flows into aggregates, where an aggregate is a set of at most M flows, all with the same maximum packet size and the same weight, and M is a free parameter[3].

We define as *backlog of a flow* the sum of the sizes of the packets currently stored in the flow queue, and as *backlog of an aggregate* the sum of the backlogs of its flows. Finally, we say that a flow/aggregate is *backlogged* if its backlog is higher than 0.

SCHED+ iteratively chooses the aggregate to serve. Once selected an aggregate for service, SCHED+ serves

[3]In an actual instance of SCHED+, such as the Linux implementation of QFQ+, aggregates can be created, and flows can be added to/removed from them, when new flows are created or when flow parameters are changed.

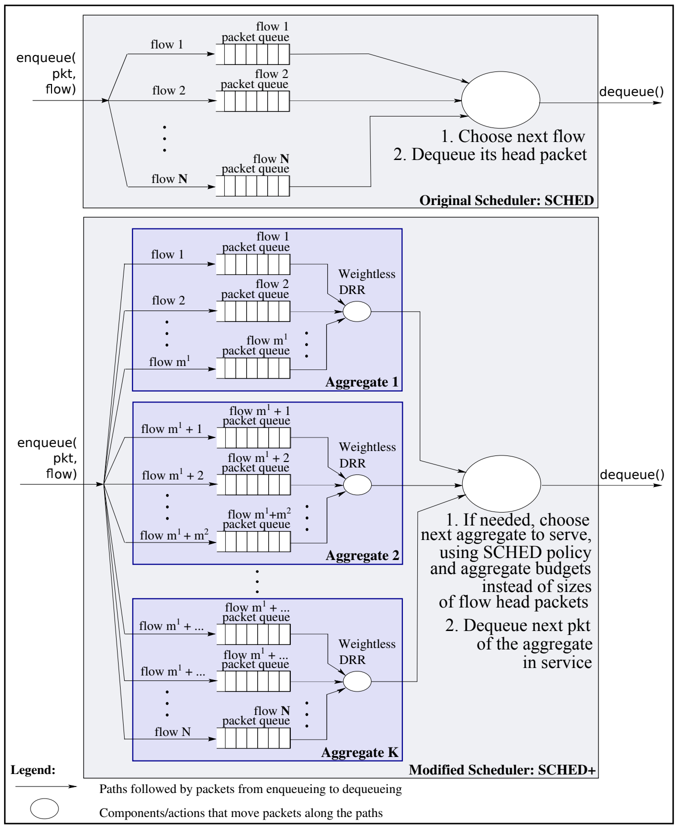

Figure 1: Graphical representation of the modification scheme.

only the flows in that aggregate *for a while*, then it chooses the next aggregate and so on. SCHED+ establishes the maximum amount of service that each aggregate can receive once selected for service, by assigning to each backlogged aggregate, say the k-th aggregate, a *budget* equal to $m^k L^k$ bits, where m^k is the current number of flows in the aggregate and L^k is the maximum packet size for the flows in that aggregate. On each invocation of the function *dequeue()* (Figure 1), which dequeues and returns the next packet to transmit, SCHED+ decreases the budget of the in-service aggregate by the size of the packet. SCHED+ selects a new aggregate either when the budget of the in-service aggregate, say the k-th aggregate, finishes, or when the backlog of the aggregate finishes. In the first case, SCHED+ assigns again a budget equal to $m^k L^k$ to the deselected aggregate and reschedules it.

SCHED+ uses DRR [4] to schedule flows inside aggregates, hence we describe briefly how DRR works before we continue to describe SCHED+ itself. DRR serves flows in rounds: in each round, DRR allows each flow to transmit a maximum number of bits, called *quantum*, equal to the sum of a fixed component—proportional to the weight of the flow—and a variable component (*deficit counter*), equal to the portion of the quantum not used by the flow in the previous round.

For each invocation of the function *dequeue()* during the service of an aggregate, SCHED+ chooses the next flow to serve, among the ones of the aggregate, through a weightless DRR (all the flows in an aggregate have the same weight), using the maximum packet size of the flows as the fixed component of the quantum. Then SCHED+ dequeues and returns the head packet of the flow.

Finally, as for the selection of the next aggregate, SCHED+ associates each aggregate also with a weight equal to the sum of the weights of the flows it contains, and chooses the next aggregate according to the SCHED policy, using the budgets of the aggregates in place of the sizes of the head packets of the flows, and the weights of the aggregates in place of the weights of the flows.

We compare SCHED+ against a classical hierarchical scheduler in Section 7. We conclude instead this section by discussing a useful property of SCHED+, on which the benefits of the modification scheme depend. These benefits are then reported in Section 2.1.

SCHED+ does not use any information about the head packet of a flow before it is the turn of the flow to be served (at that point SCHED+ uses the size of the head packet to decide whether the aggregate has enough budget and whether the flow has enough credit, as well as, if it is the case, to decrement both the credit of the flow and the budget of the aggregate). As a consequence, even if the head packet of some already backlogged flow changes, SCHED+ does not need however to update either the schedule of the flows inside the aggregate or the overall schedule of the aggregates. SCHED+ needs to change the schedule only when it has to re-schedule the in-service aggregate, or when it has to add a newly backlogged aggregate. In what follows we refer to this property as *head-packet independence*.

2.1. Benefits

Computational cost. The frequency at which the original SCHED operations are executed to schedule aggregates can be easily controlled in a system with many flows and under heavy load, i.e., exactly where efficiency is important. In fact, in such a system, the number of different weights and packet sizes is likely to be lower than the number of flows. It follows that $m^k \sim M$ and, almost always, most flows are backlogged for each aggregate. This fact, combined with the head-packet independence of SCHED+, has two consequences. First, the costly operation of choosing a new aggregate and updating the schedule is performed only after $\sim ML$ bits are served. Second, the schedule of the aggregates almost never needs to be updated on packet enqueues. In the end, the higher M is, the closer the amortized computational cost is to the one of a weightless DRR. This fact is confirmed by our experimental results (Section 8).

Compliance with non-FIFO queues and hierarchical settings. If the queue of a flow is not FIFO, then its head packet may change after the aggregate to which the flow belongs has been already scheduled. This may be the case, e.g., if the flow represents an internal node in a hierarchical setting. In fact, an internal node schedules packets of other internal nodes or leaves (flows), and hence its next packet to dequeue may change when a new packet arrives for a leaf in the sub-tree rooted at the node. SCHED+ does not need to perform any schedule change after these head-packet changes, thanks to head-packet independence. Hence SCHED+ does not need further (logical) modifications to handle flows with non-FIFO queues or internal nodes in a hierarchical setting.

The only requirement for the internal nodes is that their interface includes a *peek_next_packet()* function. This is, e.g., the case for the *classes* that implement both leaves and internal nodes in Linux [11].

3. Quick Fair Queueing Plus (QFQ+)

Quick Fair Queueing Plus (QFQ+) is a new fair-queueing packet-scheduling algorithm, obtained by applying the scheme reported in Section 2 to QFQ [5]. A certain familiarity with the non-trivial operations of QFQ is needed to understand the steps of QFQ+, which we describe in an extended version of this paper [12]. For the reader interested also in full details and in all corner cases, a working C implementation of QFQ+ can be found in the test environment [12] that we have used for our experiments. A fully functional, production-quality implementation of QFQ+ can instead be found in the Linux kernel (from 3.8.0). This implementation is basically just a super-set, in terms of functionalities, of the above test implementation. We provide more details on the Linux-kernel implementation of QFQ+ in the next section.

3.1. Linux implementation

The main difference between, on one side, the algorithm described in [12] and the QFQ+ implementation in the test environment, and, on the other side, the Linux-kernel implementation of QFQ+, is that in the latter the value of M is not free, but automatically computed as $\min(8, Q_{pkts})$, where Q_{pkts} is equal to one plus the size, in number of packets, of the transmit queue of the interface to which QFQ+ is attached (e.g., the size of the buffer ring in typical modern NICs). Hence, even if the transmit queue is large, M is however limited to a small value.

As discussed in Section 4, this fact guarantees that QFQ+ always provides service guarantees close to those of QFQ. In particular, basing on our experimental results, we have chosen 8 as maximum possible value of M because it provides a convenient trade-off between service guarantees and performance (Section 8.1). We derive the exact guarantees provided by QFQ+ in Section 6, by instantiating for QFQ+ the general service guarantees reported in Section 4.

4. Service guarantees of SCHED+

In this section we show and discuss the worst-case service guarantees that SCHED+ provides in case SCHED is either WF^{2}Q [9] or WF^{2}Q+ [10], or else SCHED is any approximate variant [6, 7, 5] of any of the previous two fair-queueing schedulers. To get simpler formulas, hereafter we assume that the sum of the weights of the flows, say Φ, is not higher than 1. The corresponding guarantees for the case $\Phi > 1$ can be obtained by just replacing ϕ_i and ϕ^k with $\frac{\phi^k}{\Phi}$ and $\frac{\phi_i}{\Phi}$ in the next statements, where ϕ_i and ϕ^k are the weights of, respectively, the generic i-th flow and any flow of the k-th aggregate.

We summarize our results before reporting exact formulas. To this purpose, we introduce the following symbols: L_i, L^k, L and R, where the first three symbols denote the maximum packet size for, respectively, the generic i-th flow, any flow of the k-th aggregate, and all the flows, while R denotes the link rate if constant. Besides, we focus on the following reference quantity for measuring the accuracy of the time guarantees of a scheduler.

Definition 1. *We define* packet service time *for the i-th flow, the transmission time of one maximum-size packet of the flow at the rate reserved to the flow.*

On a constant-rate link, since the minimum fraction of the link bandwidth reserved to the i-th flow is equal to $\phi_i R$, the packet service time for the flow is equal to $\frac{L_i}{\phi_i R}$, and, in particular, to $\frac{L^k}{\phi^k R}$ if the flow belongs to the k-th aggregate.

To highlight the importance of this quantity, consider the packet flow generated by a time-sensitive application. For the application to be feasible, the packet service time for the flow must be low enough to meet the latency requirements of the application. This implies that packet delays, or delay variations, in the order of that service time are likely to be tolerated or even negligible for the application. For example, in a VoIP application, a delay equal to the service time of a few packets would be much shorter than the maximum latency allowed for an intelligible conversation, while short audio gaps could be easily concealed through small playback buffers. A summary of our results follows.

Time guarantees. On a constant-rate link, SCHED+ guarantees, for any flow of the k-th aggregate, the same worst-case packet completion times as SCHED, plus an additional packet delay variation[4]. This variation is loosely upper-bounded by $4\frac{L^k}{\phi^k R} + (M-1)\frac{L}{R}$, i.e., by the sum of four packet service times for the flow, and of the transmission time of $M-1$ maximum-size packets at line rate.

Bit guarantees. Over any time interval, SCHED+ guarantees that any flow of the k-th aggregate receives at least the same worst-case amount of service (number of bits transmitted) as under SCHED, minus a *lag* loosely upper bounded by $4L^k + (M-1)L$.

[4]We assume that positive variations correspond to increased delays.

What is the actual impact of the above degradation of the service guarantees? As for the terms $4\frac{L^k}{\phi^k R}$ and $4L^k$, a packet-delay-variation component ranging from $\frac{L^k}{\phi^k R}$ to $6\frac{L^k}{\phi^k R}$ is already present in the worst-case packet completion times guaranteed by the SCHED schedulers considered in this section, while a lag component ranging from $2L^k$ to $7L^k$ is already present in the guaranteed service lag[5] (Appendix A). In practice, for both SCHED and SCHED+, these components are low enough to guarantee that both schedulers always provide a smooth service, and, even in a possible scenario where the theoretical bounds are actually reached, they can still meet the requirements of time-sensitive applications.

Regarding instead the terms $(M-1)\frac{L}{R}$ and $(M-1)L$, the extent to which they worsen guarantees for a given M depends on the presence and the size of a *FIFO transmit queue*, such as a buffer ring in a modern NIC. These queues are typically used to drive communication devices and to absorb link-feeding latencies. Both the packet completion times and the lag guaranteed by any scheduler happen to always contain a variation component equal, respectively, to $\frac{Q}{R}$ and Q (Appendix A), where Q is equal to the size of the transmit queue, in bits, plus the maximum possible packet size. In the end, if $ML \sim Q$ or $ML \ll Q$, then both the bit and the time guarantees provided by SCHED+ and SCHED are comparable or close to each other. In Section 6 we provide a concrete example of the degradation of the service guarantees for the pair QFQ/QFQ+ as a function of Q.

As for the possible values of Q in a real system, the sizes of the transmit queues vary greatly from system to system, ranging from 1-2 packets to even one thousand of packets. Fortunately, considering, e.g., QFQ and QFQ+, and according to our experiments, $M = 2$ is enough for QFQ+ to achieve a lower execution time than the original scheduler, while with $M = 8$ the execution-time cut achieved by QFQ+ is already close to its maximum possible value (Section 8).

The service guarantees reported so far derive from the more accurate and informative guarantees that we show in sections 4.1 and 4.2, namely the upper bounds that SCHED and SCHED+ guarantee to two special indexes: the Time and the Bit Worst-case Fair Indexes (T-WFI and B-WFI). We compute, intentionally, slightly loose upper bounds, to get simpler formulas. In sections 4.1 and 4.2 we also discuss the additional useful figures of merit that these indexes measure, such as packet delay variation. We prove instead the service guarantees of SCHED+ in Section 5. Finally, for the reader convenience, we also report

[5]WF^{2}Q and WF^{2}Q+ provide an optimal service in that they guarantee the minimum possible values, $\frac{L^k}{\phi^k R}$ and $2L^k$, for these components. The other instances of SCHED+ are instead near optimal in that these components, in the service guarantees of these schedulers, are equal to small multiples of the above minimum possible values.

full proofs of the original service guarantees of SCHED (T-WFI and B-WFI) in Appendix A.

4.1. Time Worst-case Fair Index (T-WFI)

The T-WFI of a scheduler for a given flow, defined assuming that flow queues are FIFO and the link rate is constant [5], is equal to the difference (packet delay) between the worst-case transmission time guaranteed by the scheduler to any packet of the flow, and the maximum time by which the packet should be transmitted, *ideally*, according to the weight and the backlog of the flow [9]. In particular, this ideal time is equal to the maximum time by which the packet may be completed in case, after the arrival of the packet, the flow was guaranteed its reserved share, $\phi_i R$, of the link bandwidth.

Before reporting the formula of the T-WFI, we point out that the worst-case packet delay with respect to the above ideal completion time is not only a measure of fairness. This delay, and hence the T-WFI itself, happens to be also equal to the worst-case delay variation with respect to the ideal completion time: in fact, in any realistic scenario, any fair-queueing scheduler usually transmits most packets no later than their ideal completion times. Finally, using the T-WFI we can compute actual worst-case packet completion times as well, by just summing the T-WFI itself to ideal completion times.

To introduce the formal definition of the T-WFI of a scheduler, consider that, since any flow has a FIFO queue, the ideal worst-case completion time for a packet of the i-th flow arriving at time t_a is equal to $t_a + \frac{B_i(t_a)}{\phi_i R}$, where $B_i(t_a)$ is the backlog of the flow just after time t_a (because, defined as t_c the completion time of the packet, and considering that the packets of the flow are dequeued in FIFO order, it follows that the packets transmitted during $[t_a, t_c]$ are exactly the packets that happen to be backlogged after time t_a). Accordingly, the T-WFI guaranteed by a scheduler to the i-th flow can be defined as follows:

$$\text{T-WFI}_i \equiv \max_{p \in i\text{-th flow}} \left[t_c - \left(t_a + \frac{B_i(t_a)}{\phi_i R} \right) \right] \qquad (1)$$

where p is any packet of the i-th flow, t_a is the arrival time of p, and t_c is the completion time of p under the scheduler at hand. Note that the order in which the scheduler dequeues packets determines also the value of $B_i(t_a)$.

T-WFI of SCHED

The packet size L_i and the weight ϕ_i of a flow happen to be, for any instance of SCHED, the only per-flow quantities on which the service guarantees provided to that flow depend (Appendix A). In this respect, all the flows that would belong to the same aggregate in SCHED+, say the k-th aggregate, have $L_i = L^k$ and $\phi_i = \phi^k$. Hence they all share the same T-WFI under SCHED, which we denote as T-WFI^k_{SCHED}. Hereafter we focus on T-WFI^k_{SCHED} instead of the T-WFI of SCHED for the generic i-th flow, because it is easier to compute the T-WFI of SCHED+ starting from the former than from the latter. From Theorem 5 in Appendix A we have (substituting the equality $\Delta W = Q$ in the statement of the theorem)

$$\text{T-WFI}^k_{SCHED} \le \frac{L^k}{\phi^k R} + \frac{\Delta S^k + Q + L - L^k}{R}. \qquad (2)$$

ΔS^k is equal to $\Delta S^{k-} + \Delta S^{k+}$, where ΔS^{k-} and ΔS^{k+} are two timestamp-error terms, as defined in Appendix A (see Table 1 for the other, already introduced symbols).

T-WFI of SCHED+

Theorem 1. *SCHED+ guarantees the following T-WFI to any flow in the k-th aggregate*

$$\textit{T-WFI}^k_{SCHED+} \le \left(5 - \frac{1}{m^k}\right) \frac{L^k}{\phi^k R} + + \frac{\Delta S^k + Q + ML - m^k L^k}{R}. \qquad (3)$$

From this theorem, which we prove in Section 5.5, and (2), we get that SCHED+ guarantees the same T-WFI as SCHED, plus the following extra delay:

$$\left(4 - \frac{1}{m^k}\right) \frac{L^k}{\phi^k R} + \frac{(M-1)L - (m^k - 1)L^k}{R}. \qquad (4)$$

4.2. Bit Worst-case Fair Index (B-WFI)

Consider any time interval $[t_1, t_2]$ during which the i-th flow is continuously backlogged. Denoted as $W(t_1, t_2)$ the total number of bits transmitted by the system during $[t_1, t_2]$, and according to the weight ϕ_i of the flow, in a perfectly fair system the flow should receive at least a minimum amount of service (number of bits transmitted) equal to $\phi_i W(t_1, t_2)$. The B-WFI of a scheduler for the i-th flow is equal to the maximum difference between this minimum service and the actual amount of service, $W_i(t_1, t_2)$, provided by the scheduler to the flow [10].

We extend the standard definition of the B-WFI to take into account the following fact too: the number of bits of the i-th flow that can be transmitted during $[t_1, t_2]$ is upper-bounded by: the backlog $B_i(t_1)$ plus the sum of the sizes, say $A_i(t_1, t_2)$, of the packets of the i-th flow arriving during the open interval (t_1, t_2). Defined $B_i(t_1, t_2) \equiv B_i(t_1) + A_i(t_1, t_2)$, we define the B-WFI as follows:

$$\text{B-WFI}_i \equiv \max_{[t_1,\, t_2]} \{ \min[\phi_i W(t_1,\, t_2), B_i(t_1,\, t_2)] - W_i(t_1,\, t_2) \}. \qquad (5)$$

The B-WFI is an accurate measure of both the short- and the long-term fairness of a scheduler. Besides, the B-WFI is defined independently of the bandwidth of the link and of whether the latter fluctuates.

B-WFI of SCHED

Basing on the same arguments used to introduce T-WFI$^k_{SCHED}$ in Section 4.1, we focus on the B-WFI guaranteed by SCHED to any flow that would belong to the k-th aggregate under SCHED+, and denote this metric as B-WFI$^k_{SCHED}$. From Theorem 4 in Appendix A (substituting again the equality $\Delta W = Q$), we have

$$\text{B-WFI}^k_{SCHED} \leq \phi^k Q + \phi^k \Delta S^k + (1-\phi^k)L^k + L. \quad (6)$$

B-WFI of SCHED+

Theorem 2. *SCHED+ guarantees the following B-WFI to any flow in the k-th aggregate:*

$$\begin{aligned} &\textit{B-WFI}^k_{SCHED+} \leq \\ \phi^k Q + \phi^k \Delta S^k &+ (5 - \frac{1}{m^k} - m^k\phi^k)L^k + \frac{M}{m^k}L. \end{aligned} \quad (7)$$

Comparing (7) against (6), we have that the B-WFI guaranteed by SCHED+ coincides with that guaranteed by SCHED, except for an additional term

$$\left(4 - (m^k-1)\phi^k - \frac{1}{m^k}\right)L^k + \left(\frac{M}{m^k} - 1\right)L. \quad (8)$$

5. Proof of the service guarantees of SCHED+

In this section we prove the upper bound to the T-WFI of SCHED+ reported in Theorem 1. The proof set provides also most of the steps needed to prove the upper bound (7) to the B-WFI. For the interested reader, the remaining steps for proving also this bound can be found in Appendix B. In this section we compute an upper bound to the T-WFI of DRR too. We use the latter bound to better put the service provided by QFQ+ into context in Section 6.

For brevity, hereafter we refer to the upper bound in Theorem 1 as just the T-WFI of SCHED+. Our strategy for computing the T-WFI of SCHED+ moves from the strong similarities, highlighted in Section 5.1, between SCHED and SCHED+. Thanks to these similarities, we can compute the service guaranteed by SCHED+ to the generic k-th aggregate *as a whole* in two steps. The first step consists basically in multiplying by m^k or M some terms in the T-WFI of SCHED. This yields a sort of *baseline* per-aggregate guarantees (Section 5.2), to which we have to add, as a second step, a further term, which accounts for a peculiarity of only SCHED+ (Section 5.3). Finally, we can get the T-WFI of SCHED+ from per-aggregate guarantees, as a function of how DRR distributes the service provided to an aggregate among the flows of the aggregate (Section 5.4). In Section 5.5 we do apply all of these steps to compute the T-WFI of SCHED+.

5.1. Similarities between SCHED and SCHED+

SCHED is a timestamp-based scheduler: it associates both packets and flows with *virtual start* and *finish* timestamps. SCHED computes the timestamps of a packet as a function of various parameters, among which the size of the packet itself. All the other parameters being equal, the larger a packet is, the larger its virtual finish time is. Finally, SCHED timestamps flows in such a way that the timestamps of a flow coincide with the timestamps of its current head packet. Leaving out *eligibility* [5] for simplicity, SCHED does its best to serve flows in virtual-finish-time order.

Consider now SCHED+. To highlight the similarities between SCHED and SCHED+, hereafter we pretend that the batch of packets dequeued while an aggregate is in service is a sort of head *super packet* of the aggregate. Accordingly, and considering that the weight of the k-th aggregate is equal to $m^k\phi^k$, we define the following *per-aggregate T-WFI* for SCHED+:

$$\text{T-}\widehat{\text{WFI}}^k_{SCHED+} \equiv \max_{\hat{p} \in k\text{-th aggr.}} \left[\hat{t}_c - \left(\hat{t}_a + \frac{\hat{B}^k(\hat{t}_a)}{m^k \phi^k R} \right) \right] \quad (9)$$

where $\hat{p}$ is any super packet of the k-th aggregate, $\hat{t}_a$ is the arrival time of the super packet, defined as the minimum among the arrival times of the packets in the super packets, $\hat{t}_c$ is the completion time of the super packet, defined as the completion time of the last packet in the super packet, and $\hat{B}^k(\hat{t}_a)$ is the sum of the sizes of the super packets of the k-th aggregate that are transmitted during $[\hat{t}_a, \hat{t}_c]$. We had to define $\hat{B}^k(\hat{t}_a)$ in this more complex way, with respect to just the backlog of the k-th aggregate, because super packets are not necessarily transmitted in the same order as they arrive (they are just a device to group the packets dequeued during each service period).

An upper bound to T-$\widehat{\text{WFI}}^k_{SCHED+}$ could be easily derived from the bound (2) to T-WFI$^k_{SCHED}$, on condition that SCHED+ timestamped super packets with the same rules with which SCHED timestamps packets, and scheduled aggregates as a function of the timestamps of their head super packets, with the same policy with which SCHED schedules flows as a function of the timestamps of their head packets.

In this respect, SCHED+ does schedule aggregates with the SCHED policy, but uses budgets to timestamp aggregates (Section 2). Hence both the above requirements would be met (only) if the budget of every aggregate scheduled for service *always* coincided with the size of the current head super packet of the aggregate. We assume that this does hold true, as a *simplifying assumption* for computing baseline guarantees in the next section.

5.2. Baseline guarantees

Lemma 1. *If the simplifying assumption reported in Section 5.1 holds, then we have*

$$T\text{-}\hat{WFI}^{k}_{SCHED+} \leq \frac{L^k}{\phi^k R} + \frac{\Delta S^k + Q + ML - m^k L^k}{R}. \tag{10}$$

Proof. Consider the following facts:

1. the service of each super packet is non-preemptive in the same way as the service of a single packet is;
2. the size of a super packet of the k-th aggregate is equal to at most $m^k L^k$ bits instead of just L^k bits;
3. the term L in (2) takes into account the fact that, under SCHED, at most one out-of-order, maximum-size packet p' may be dequeued between the arrival and the dequeueing time of any packet p [5][6]: the same problem occurs therefore under SCHED+, in terms of super packets, but the maximum size of a super packet is now ML bits instead of just L bits;
4. the weight of an aggregate is equal to m^k times the weight ϕ^k of the flows it contains.
5. for any instance of SCHED, the cumulative error term ΔS^k is a multiple of $\frac{L^k}{\phi^k}$, hence its value does not change if L^k and ϕ^k are replaced with $m^k L^k$ and $m^k \phi^k$.

Combining these facts with the simplifying assumption in Section 5.1, we get that (2) can be turned into the bound (10) by replacing the maximum packet sizes L^k and L with $m^k L^k$ and ML, and the weight ϕ^k with $m^k \phi^k$. □

In the next lemma we increase the bound (10) so as to hold also in case the simplifying assumption does not hold.

5.3. Removing the simplifying assumption

Lemma 2. *Regardless of whether the simplifying assumption in Section 5.1 holds, $T\text{-}\hat{WFI}^{k}_{SCHED+}$ is upper-bounded by the right-hand side of* (10) *plus* $\frac{L^k}{\phi^k R}$.

Proof. If the backlog of an aggregate finishes before the aggregate consumes all of its budget, then the timestamps of the aggregate happen to be higher than they would have been if computed as a function of the actual size of the head super packet of the aggregate. This causes the service of the aggregate to be unjustly postponed.

To compute a loose but simple upper bound to how long the service of the aggregate may be postponed, suppose that the size of the head super packet of the aggregate tends to 0. The aggregate is scheduled as if the size of its head super packet was instead $m^k L^k$ bits. Since SCHED+ emulates the service of an ideal, perfectly fair system (Appendix A), and such a system would serve the aggregate at a rate at least equal to its reserved fraction of the link bandwidth, namely $m^k \phi^k R$, it follows that the aggregate is scheduled as if its head super packet was completed in the ideal system, in the worst case, $\frac{m^k L^k}{m^k \phi^k R} - 0 = \frac{L^k}{\phi^k R}$ time units later than it actually would be. This yields a service delay equal to $\frac{L^k}{\phi^k R}$. □

5.4. Service guarantees of DRR

In this section we compute both the T-WFI of DRR, and the guarantee provided by DRR, in terms of bits dequeued, to the flows in a SCHED+ aggregate. We use the latter guarantee to compute the T-WFI of SCHED+.

We derive both guarantees from a special lower bound to the number of bits dequeued by DRR for the i-th flow, in a time interval $[t_1, t_2]$ during which the flow is continuously backlogged. To write this bound, we need to introduce some additional notations: we denote as $\overline{W}(t_1, t_2)$ and $\overline{W}_i(t_1, t_2)$ the sum of the sizes of all the packets dequeued during $[t_1, t_2]$, and of all the packets of the i-th flow dequeued during $[t_1, t_2]$, respectively.

We define as *transmission opportunity* for the i-th flow every maximal sub-interval of $[t_1, t_2]$ during which the flow is continuously at the head of the DRR queue. Let h be the number of transmission opportunities for the i-th flow during $[t_1, t_2]$. To get a simpler formula, we assume that t_1 is lower or equal to the beginning of the first transmission opportunity. From [4, Lemma 2], we can derive the following loose upper bound:

$$\begin{aligned} \overline{W}(t_1, t_2) = \sum_{j=1}^{N} \overline{W}_j(t_1, t_2) < \\ h \sum_{j=1}^{N} \frac{\phi_j}{\phi_{min}} L + (N-1)L = \\ h \frac{\sum_{j=1}^{N} \phi_j}{\phi_{min}} L + (N-1)L \end{aligned} \tag{11}$$

where ϕ_j is the weight of the j-th flow and ϕ_{min} is the minimum weight among all flows.

This bound is certainly loose if the maximum size of the packets of some flow is lower than L. We use this loose bound instead of a tighter one to avoid even longer formulas. As for the minimum service guaranteed to the i-th flow, we have, again from [4, Lemma 2],

$$\overline{W}_i(t_1, t_2) \geq (h-1) \frac{\phi_i}{\phi_{min}} L - L \tag{12}$$

where ϕ_i is the weight of the i-th flow and the factor $h-1$ in the first term is not equal to h because t_2 may be, in the worst case, equal to the beginning of the h-th transmission opportunity, and hence the i-th flow may have not yet used such opportunity at all by time t_2.

We want now to upper-bound $\overline{W}(t_1, t_2)$ as a function of $\overline{W}_i(t_1, t_2)$. To this purpose, solving (12) for h, we can

[6]Namely, a packet p' that, according to its timestamps, should have been dequeued after p.

write

$$h \leq \frac{\phi_{min}}{\phi_i}\left(\frac{\overline{W}_i(t_1,t_2)}{L} + \frac{\phi_i}{\phi_{min}} + 1\right). \tag{13}$$

Then, replacing this inequality in (11), we get the following bound to the total number of bits that might be dequeued in a time interval during which $\overline{W}_i(t_1, t_2)$ bits of the i-th flow are dequeued:

$$\begin{aligned} &\overline{W}(t_1,t_2) \leq \\ &\left(\frac{\overline{W}_i(t_1,t_2)}{L} + \frac{\phi_i}{\phi_{min}} + 1\right)\frac{\sum_{j=1}^{N}\phi_j}{\phi_i}L + (N-1)L = \\ &\frac{\sum_{j=1}^{N}\phi_j}{\phi_i}\overline{W}_i(t_1,t_2) + \\ &\left(\frac{\sum_{j=1}^{N}\phi_j}{\phi_{min}} + \frac{\sum_{j=1}^{N}\phi_j}{\phi_i} + N - 1\right)L. \end{aligned} \tag{14}$$

The term $\frac{\sum_{j=1}^{N}\phi_j}{\phi_i}\overline{W}_i(t_1,t_2)$ is equal to the total number of bits that would be dequeued, in an ideal, perfectly fair system, while $\overline{W}_i(t_1,t_2)$ bits if the i-th flow are dequeued. Hence, the term in the last line of (14) represents the extra service that the real system has to provide, in the worst-case, to guarantee that the i-th flow receives the same amount of service as in the ideal system.

This extra service leads to a high T-WFI for flows with a low packet service time (Definition 1), as we discuss after proving the following theorem.

Theorem 3. *Denoted as T-WFI$_{i,DRR}$ the T-WFI guaranteed by DRR to the i-th flow, we have*

$$T\text{-}WFI_{i,DRR} \leq \left(\frac{\sum_{j=1}^{N}\phi_j}{\phi_{min}} + \frac{\sum_{j=1}^{N}\phi_j}{\phi_i} + N - 1\right)\frac{L}{R} + \frac{Q}{R}. \tag{15}$$

Proof. We get the thesis by: 1) replacing (instantiating) t_1 and t_2 in (14) with the time instants t_a and t_c at which a packet of the i-th flow is enqueued and dequeued, 2) turning the resulting bound into an upper bound to $t_2 - t_1$, basing on the fact that bits are dequeued at a constant rate R, 3) adding a further worst-case delay $\frac{Q}{R}$ due to the transmit queue. □

The following negative property unfortunately holds for (15): being all the other parameters equal, the smaller the packet service time for the i-th flow is, the larger the bound (15) is with respect to the packet service time itself. In particular, the terms $\frac{L}{R}$ and N are independent of the packet service time, and N is of course very large in a system serving many flows. Even worse, $\frac{\sum_{j=1}^{N}\phi_j}{\phi_{min}} \geq N$ always holds. It follows that, as we show with numerical examples in Section 6, a T-WFI in the order of the bound (15) may easily be unbearable for flows with a short packet service time, i.e., the most time-sensitive flows.

Unfortunately, the inequality we have started from to prove (15), namely (11), has been in its turn proved in a rather likely scenario, i.e., in the case a packet of the i-th flow arrives after the flow has already been served for the current round. Hence, as confirmed by our experimental results in Section 8, DRR is likely to actually exhibit a T-WFI in the order of the upper bound (15) in realistic scenarios.

Finally, to get the service guaranteed by the internal DRR scheduler to the flows in an aggregate, we have just to instantiate (14) for the flows of the aggregate. In particular, we have to substitute the following equalities in (14): $N = m^k$, $\phi_i = \phi^k$, $\phi_{min} = \phi^k$, $\sum_{j=1}^{N}\phi_j = m^k\phi^k$, $L = L^k$ and $\overline{W}(t_1,t_2) = \overline{W}^k(t_1,t_2)$, where $\overline{W}^k(t_1,t_2)$ is the sum of the sizes of the packets of the flows of the k-th aggregate dequeued during $[t_1, t_2]$. By doing so, we get

$$\overline{W}^k(t_1,t_2) \leq m^k\overline{W}_i(t_1,t_2) + \left(3m^k - 1\right)L^k. \tag{16}$$

5.5. Proof of Theorem 1 (T-WFI)

From Lemma 2, and defined, for brevity: $\Delta \equiv \Delta S^k + ML - m^kL$, we get the following upper bound to the difference between the completion time $\hat{t}_c$ of a super packet $\hat{p}$ of the k-th aggregate under SCHED+, and the arrival time $\hat{t}_a$ of the super packet:

$$\hat{t}_c - \hat{t}_a \leq \frac{\overline{W}^k(\hat{t}_a, \hat{t}'_c)}{m^k\phi^kR} + 2\frac{L^k}{\phi^kR} + \frac{\Delta + Q}{R} \tag{17}$$

where $\hat{t}'_c$ is the dequeueing time of the last packet of the super packet $\hat{p}$, and $\overline{W}^k(\hat{t}_a, \hat{t}'_c)$ is the sum of the sizes of the super packets of the k-th aggregate dequeued during $[\hat{t}_a, \hat{t}'_c]$.

To simplify next derivations, we turn (17) into a bound to $\hat{t}'_c - \hat{t}_a$. According to the derivations in Appendix A, the term $\frac{Q}{R}$ in (17) comes from the obvious bound $\hat{t}_c \leq \hat{t}'_c + \frac{Q}{R}$. Substituting this inequality in (17), we get

$$\hat{t}'_c - \hat{t}_a \leq \frac{\overline{W}^k(t_a, t'_c)}{m^k\phi^kR} + 2\frac{L^k}{\phi^kR} + \frac{\Delta}{R}. \tag{18}$$

We turn now (18) into a guarantee on the queueing delay of the packets of the generic i-th flow of the k-th aggregate under SCHED+. To this purpose, we consider a packet p of the i-th flow that belongs to the super packet $\hat{p}$. We denote as t_a and t'_c the arrival and the dequeueing times of p. The packet p experiences its maximum queueing delay if it is the first packet of the super packet to arrive, and the last one to be dequeued, i.e., if $t_a = \hat{t}_a$ and $t'_c = \hat{t}'_c$.

Super packets are just a device for highlighting the similarities between SCHED and SCHED+, hence we can imagine, without affecting the packet service order, to break the super packet to which p belongs in such a way that the last two equalities hold. Substituting these equalities in (18), we get

$$t'_c - t_a \leq \frac{\overline{W}^k(t_a, t'_c)}{m^k\phi^kR} + 2\frac{L^k}{\phi^kR} + \frac{\Delta}{R}. \tag{19}$$

Since (1) is a function of $\frac{B_i(t_a)}{\phi^k R}$, our next step to get the T-WFI of SCHED+ for the i-th flow (belonging to the k-th aggregate) is to turn also (19) into a bound that is a function of $\frac{B_i(t_a)}{\phi^k R}$. To this purpose, we consider that, since the i-th flow has a FIFO queue, the packets dequeued during $[t_a, t'_c]$ are exactly the packets that are present in the flow queue just after time t_a. Hence, using the same notation as in (16), we have $B_i(t_a) = \overline{W}_i(t_a, t'_c)$. From this equality and (16), it follows that $\overline{W}^k(t_a, t'_c) \le m^k B_i(t_a) + (3m^k - 1)L^k$. Replacing this inequality in (19),

$$\begin{aligned} t'_c - t_a \le \\ \frac{m^k B_i(t_a) + (3m^k - 1)L^k}{m^k \phi^k R} + 2\frac{L^k}{\phi^k R} + \frac{\Delta}{R} \le \\ \frac{B_i(t_a) + \left(3 - \frac{1}{m^k}\right) L^k}{\phi^k R} + 2\frac{L^k}{\phi^k R} + \frac{\Delta}{R} = \\ \frac{B_i(t_a)}{\phi^k R} + \left(5 - \frac{1}{m^k}\right) \frac{L^k}{\phi^k R} + \frac{\Delta}{R}. \end{aligned} \tag{20}$$

Finally, consider that, if t_c is the completion time of p, then $t_c \le t'_c + \frac{Q}{R}$. Replacing this inequality in (20),

$$t_c - \left(t_a + \frac{B_i(t_a)}{\phi^k R}\right) \le \left(5 - \frac{1}{m^k}\right) \frac{L^k}{\phi^k R} + \frac{\Delta + Q}{R}. \tag{21}$$

6. Service guarantees of QFQ+

In this section we show an upper bound to the T-WFI guaranteed by QFQ+ to any flow of the k-th aggregate. We denote this T-WFI as T-WFI$^k_{QFQ+}$. To put the bound into context, we also instantiate it for a possible real flow set, and compare it against the bounds guaranteed by QFQ and DRR. We report instead an experimental comparison among the T-WFIs of these schedulers in Section 8. For brevity, we do not show also the bound to the B-WFI guaranteed by QFQ+: this bound can be derived from (7) with the same simple steps through which we derive the bound to T-WFI$^k_{QFQ+}$ below, and the relative degradation of this bound, when moving from QFQ to QFQ+, is about the same as for the bound to T-WFI$^k_{QFQ+}$.

To upper-bound T-WFI$^k_{QFQ+}$, we instantiate (3) for QFQ+. As for the term ΔS^k in (3), from [5] we have that, if SCHED = QFQ, then $\Delta S^{k-} = 2\frac{m^k L^k}{m^k \phi^k} = 2\frac{L^k}{\phi^k}$ and $\Delta S^{k+} = 4\frac{m^k L^k}{m^k \phi^k} = 4\frac{L^k}{\phi^k}$. Considering also that $M = \min(8, Q_{pkts})$ in QFQ+, and that $Q = Q_{pkts} L$, we get

$$\begin{aligned} \text{T-WFI}^k_{QFQ+} < \\ \left(11 - \frac{1}{m^k}\right) \frac{L^k}{\phi^k R} + \frac{Q + \min(8L, Q) - m^k L^k}{R} \end{aligned} \tag{22}$$

According to (22), (2) and (15), the near-optimal guarantees of QFQ+/QFQ and the guarantees of DRR differ significantly for the flows with shorter packet service times (Definition 1) than $N\frac{L}{R}$. Hence, the higher N is and the more differentiated packet service times are, the more the difference between the quality of service provided by QFQ/QFQ+ and the quality of service provided by DRR is evident.

Flow sets can easily be significantly skewed in terms of packet service times. Consider for example some VoIP flows sharing a transmission link with other non time-sensitive flows. A possible weight distribution for guaranteeing both a higher bandwidth and a lower delay to the VoIP flows might be, e.g., assigning each VoIP flow ten or even twenty times the weight of any of the non time-sensitive flows. It follows that, if the size of: the packets of the non time-sensitive flows and the VoIP packets are, respectively, about 1.5kB and 200B, then the ratio between the packet service time for the VoIP flows and the packet service time for the other flows ranges between 1/150 and 1/75. This ratio may become up to 42 times as low if TSO/GSO is used, because in that case the maximum size of the packets, as seen by the scheduler, becomes 64kB.

To highlight, with a simple numerical example, the difference in the quality of service provided by QFQ/QFQ+ and DRR in such a skewed scenario, consider a flow set made of 1000 flows, with a weight sum equal exactly to 1. For simplicity, suppose that all packets have the same size L. Hence, to get differentiated packet service times, assume that one of the flows, say the i-th flow, has weight 0.25, and belongs to the k-th aggregate, whereas all other flows have the same weight, equal to $0.75 \cdot 10^{-3}$. The resulting minimum ratio between packet service times is then about 1/333.

In the above scenario, and according to (22) and (4), T-WFI$^k_{QFQ+} \le \frac{39L+Q+\min(8L,Q)}{R}$, and the extra delay (and variation) of QFQ+ with respect to QFQ is $\frac{11L+\min(8L,Q)}{R}$. Regarding DRR, from (15) we have instead T-WFI$_{i,DRR} \le \left(\frac{1000}{0.75} + 4 + 1000 - 1\right) \frac{L}{R} + \frac{Q}{R} = \frac{2336.\overline{3}L+Q}{R}$.

Table 2 reports the values of the bounds for increasing sizes of the transmit queue. Values are measured in multiples of $4\frac{L}{R}$, i.e., of the packet service time for the i-th flow. The T-WFIs, and hence the packet delay variations (Section 4), guaranteed by QFQ and QFQ+ are comparable. More importantly, for small values of Q, they are equal to at most a few tens of packet service times for the flow. In this respect, in Section 8 we also compare T-WFIs experimentally for this scenario, and we show that the T-WFIs actually recorded for QFQ and QFQ+ are 4-6 times as small as these bounds. In the end, even in such a skewed scenario, both QFQ and QFQ+ easily meet the delay requirements of time-sensitive applications (according to the discussion about packet service times in Section 4).

Things change dramatically with DRR, whose T-WFI bounds are much higher than those of QFQ/QFQ+, and hundreds of times as high as the packet service time, even with short transmit queues (for the reasons highlighted after the proof of Theorem 3). This negative property holds also for the T-WFI of DRR that we have recorded experimentally (Section 8). Finally, as for long transmit queues, even with a size of 3000 packets, the T-WFIs guaranteed by QFQ and QFQ+ are still 1.8 times as low as the one guaranteed by DRR.

	Size of the transmit queue (packets)					
Sched	1	10	100	1000	2000	3000
QFQ+	10.5	14.5	37	262	512	762
QFQ	**7.5**	**9.7**	**32.2**	**272**	**507**	**757**
DRR	584	587	609	834	1084	1334

Table 2: Bounds to T-WFIs, and hence to packet delay variations with respect to a perfectly fair service, guaranteed to the highest-weight flow, for increasing sizes of the transmit queue (lower is better). Measured in multiples of the packet service time for the flow.

7. Related work

To the best of our knowledge, no other general modification scheme like the one we propose in this paper is available in the literature. However, both hierarchical schedulers and two sets of $O(1)$-cost schedulers have something in common with our solution. The first set of schedulers is the family of efficient, approximate variants of WF^{2}Q+ [6, 7, 5], whereas the other set is the family of *enhanced* round-robin schedulers [13, 14, 15, 16, 17].

Hierarchical schedulers and SCHED+. As well as a classical hierarchical, two-level scheduler, SCHED+ is made of:

- a *root* scheduler—the per-aggregate variant of SCHED—which schedules internal nodes (aggregates);
- one DRR *local* scheduler for each internal node, which schedules the leaves of the node (individual flows).

But there is an important difference between SCHED+ and a classical hierarchical scheduler (such as the one described in [10]). In the latter, the root scheduler may choose a different internal node, and hence needs to be invoked, on each packet dequeue. Besides, the data structure of the root scheduler may need to be updated on each packet enqueue involving a non-backlogged flow. In contrast, the key property of SCHED+ is that it can invoke the costly SCHED operations *at a much lower frequency*, as discussed in Section 2.1.

Approximate variants of WF^{2}Q+. These schedulers share with our scheme the use of flow grouping for reducing the computational cost of the original algorithm. The most costly packet enqueues or dequeues for these schedulers are the ones in which either the scheduling order of the flow groups must be changed or the next group to serve must be found. Fortunately, more than one packet is usually enqueued or dequeued, on average, before these operations need to be executed. But there is no further control over the frequency of these operations. In contrast, one of the peculiarity of SCHED+ is that it is tilted exactly toward serving the flows in the same aggregate, and hence avoiding costly operations, for a configurable amount of time. Finally, as for a performance comparison, QFQ is the lowest-cost approximate variant of WF^{2}Q+ [5], and, as a shown in Section 8, QFQ+ outperforms QFQ.

Enhanced round-robin schedulers. In a sense, these schedulers adopt the opposite strategy than the previous set of schedulers: instead of starting from an accurate scheduling policy and reducing its cost through flow grouping and other techniques, they use DRR or a slightly-modified DRR as an efficient building block to realize more accurate policies. Differently from, e.g., QFQ+, and like DRR, all of these schedulers suffer from a T-WFI and a B-WFI that are independent of the packet service time for the flow, and that grow linearly with the number of competing flows. As shown in sections 6 and 8, such a deviation may be problematic for flows with a shorter packet service time than other competing flows. The only exception is FRR [17], whose T-WFI is however several times as high as that of QFQ [5], and hence of QFQ+.

8. Experimental results

In this section we compare QFQ+, DRR and QFQ against each other experimentally, in terms of both efficiency and service guarantees[7]. As for efficiency, we show our experimental results about number of instructions, execution time, cache misses and energy consumption. Regarding guarantees, we report our results on T-WFI (and hence on packet delay variation, Section 4.1). As for B-WFI, the relative performance of the schedulers in terms of B-WFI, in our experiments, has been about the same as in terms of T-WFI.

Test environment. We have run our experiments using the test environment [12], which is a slightly improved version of the original test environment [18]. 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.

Configuration. Each run has consisted of 50M packet enqueues plus 50M packet dequeues[8], with the controller configured so as to let flows oscillate between null backlog

[7]We have measured the performance of WF^{2}Q+ too, but the overall cost of processing packets with WF^{2}Q+ has happened to be from 3 to 4 times as high as with the other schedulers in any scenario. We do not report these results for brevity.

[8]See [12] for the script, *run_test.sh*, that we have used to run the tests.

and a backlog of 30 packets each. Such an enqueue/dequeue pattern has 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 has never been either read or written. No migration and no packet drop have occurred in any run.

Flow sets. We have considered four flow sets:

1k-w1: 1k flows all with weight 1;

1k-wmix: $500 + 250 + 125$ flows with weights 1, 2, 8;

32k-w1: 32k flows all with weight 1;

32k-wmix: $16k + 8k + 4k$ flows with weights 1, 2, 8.

Note that weights are integers, as this is the type of the weights in existing implementations of QFQ+, QFQ and DRR. In contrast, we have assumed that the sum of the weights was not higher than 1 when computing service guarantees, just to simplify formulas.

The number of competing flows in *1k-w1* and *1k-wmix* is low enough that a negligible number of cache misses occurs (Section 8.2). This helps us to isolate the net execution time of the schedulers from other factors. The other two flow sets represent instead a possible heavy-load period for a high-speed system. Finally, we have considered *1k-wmix* and *32k-wmix* because the QoS guaranteed by an accurate fair-queueing scheduler differs from that guaranteed by DRR only if flows have different packet service times (Definition 1), as shown in sections 6 and 7.

To complete the analytical comparison carried out in Section 6 with experimental results, only in the experiments about service guarantees we have considered also the flow set described in that section[9]. We denote this very skewed flow set as *1k-highw*.

Efficiency measurement. For each run we have measured: total number of instructions, total execution time, total number of cache misses and total energy consumption of the system (we have used `perf` [19] to measure number of instructions and cache misses). Then we have divided the above quantities by the number of enqueues, obtaining the cumulative average, for each quantity, for one packet generation, enqueue, dequeue and disposal. One of the shortcomings of this approach is that we could not measure, e.g., the net execution time of each scheduler. We have opted however for this method because of the precision problems related to measuring directly the above quantities for very short time periods: in some scenarios the packet-processing time has been even lower than 100ns. As for the net execution time of the schedulers, we can still get an indication of it *by difference* with respect to the packet-processing time with a FIFO scheduler (see below).

Statistics. We have repeated each run ten times and computed inter-run statistics for each value of interest. In particular, for each batch of ten runs, the variance among the recorded values has been negligible, so hereafter we report only inter-run averages.

Arrival patterns. Including also the cost of generating and discarding packets in our statistics has helped us to highlight an important point: the order in which a scheduler dequeues packets, with respect to the order in which packets arrive, influences the execution time not only of the scheduler but also of the other tasks involved in processing a packet (Section 8.2). To investigate this point in more depth we have considered two extreme and opposite packet arrival (generation) patterns. In the first, *smooth* pattern, the controller, in the enqueue phase, iteratively generates one packet for each flow with a lower backlog than the other flows (i.e., it fills flow queues in a round-robin fashion). In the second, *bursty* pattern, the controller generates the packets for each flow in bursts of random size, ranging from 1 to $16w$, where w is the weight of the flow.

Schedulers. We have computed the statistics mentioned so far for the implementations of QFQ+, DRR and QFQ contained in the test environment. These implementations are just ports, with minimal interface changes, of the original, production-quality versions of the same schedulers under Linux or FreeBSD. Only for QFQ+, the version in the test environment differs also in that M is not computed automatically, but is set manually.

We have also considered, as a baseline case, a FIFO scheduler. Given its negligible overhead, this scheduler allows us to measure, by difference, the net execution time of the other schedulers (unfortunately only in some of the scenarios, as explained in detail in Section 8.2).

Test equipment. We have run our tests on two systems with the following software and hardware characteristics:

- Ubuntu 12.04, kernel 3.6.0,
  Intel Core i7-2760QM @ 2.40GHz, gcc 4.6.3 -O3

- OS X 10.7.4, Darwin 11.4.0,
  Intel Core i5-2557M @ 1.7 GHz, gcc 4.2.1 -O3

Since the relative performance of the schedulers has been about the same on the two systems, we report our results only for the first system.

In the next sections, first we show how we have tuned the maximum size M of the aggregates in QFQ+, then we compare the performance of QFQ+ against the one of the other schedulers.

8.1. Tuning the size of the aggregates

Figure 2 shows the average execution time of QFQ+ for smooth arrivals, as a function of M. The relative decrease of the execution time as a function of M has been about the same also for bursty arrivals (not shown). As can be seen, the relative decrease is quite small when passing from $M = 8$ to $M = 16$. This is the main reason why we have chosen 8 as the maximum possible value for M in the Linux implementation of QFQ+ (Section 3).

[9]More precisely, to implement the same weight ratios of this scenario in the test environment, we assigned 333 to the weight of the only high-weight flow, and 1 to the weight of each of the other flows.

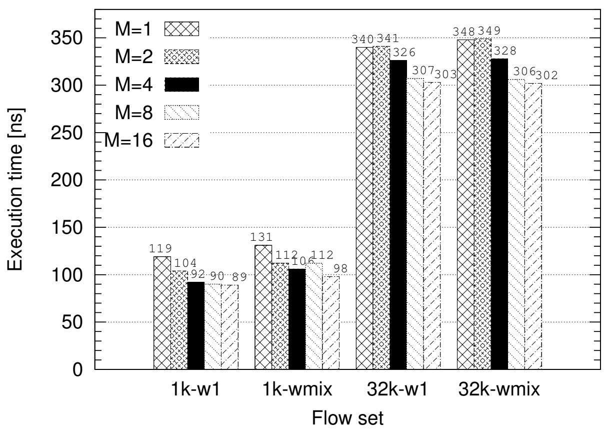

Figure 2: Execution times of QFQ+ as a function of M, with smooth arrivals (lower is better).

The performance of QFQ+ reported in the next sections is the one for $M = 8$ too. In addition, by comparing the execution times shown in the next section (Figure 4) with the ones in Figure 2, one can have an idea of the relative performance of QFQ+ with respect to the other schedulers, also for the other values of M. For example, with $M = 1$, the execution times with QFQ+ are about the same as with QFQ.

8.2. Efficiency comparison

Number of instructions. We start our comparison from the (average) number of instructions executed by the schedulers, shown in Figure 3 for smooth arrivals. We have got about the same figures with bursty arrivals, which we do not report. By computing differences with respect to FIFO, we can see that the number of instructions executed by QFQ+ ($M = 8$) is about 40% lower than QFQ, and at most 50% higher than DRR. The number of instructions does not increase dramatically when moving from 1k to 32k flows.

Execution time and cache misses - Smooth arrivals. Things do change with execution times and cache misses, shown in Figure 4 and 5 for smooth arrivals (recall that we consider the cumulative execution time of a packet allocation, enqueue, dequeue and disposal). First, execution times with 32k flows are much higher than with 1k flows, as the number of cache misses with 32k flows is much higher too, and cache misses severely impact execution times.

Second, the execution time with QFQ+ is from 0.3 to 0.9 times as high as that with QFQ, and it even happens to be lower than that with DRR with *32k-wmix*. The main reason for this inversion is that QFQ+ *handles* cache misses better than DRR with this large, mixed-weight flow set, as discussed below.

QFQ+ provides a smoother service than DRR: 1) thanks to *eligibility* [9], QFQ+ interleaves smoothly the service of high-weight aggregates with the one of low-weight ones, 2) the flows in each aggregate are served in a round-robin fashion. In contrast, the service of DRR is more bursty: DRR serves each backlogged flow repeatedly until the quantum of the flow finishes. And the quantum of a flow is proportional to its weight.

As a consequence of the smoother service of QFQ+, the packet-dequeue order of this scheduler is closer to the packet arrival order than that of DRR (as we have also verified by tracing packet arrivals and dequeues). And the order in which packets are generated tends to be close to the order in which packets are stored in memory. This improves the effectiveness of caches, basically thanks to prefetching.

With 1k flows the number of cache misses is so low that it has no influence on the relative execution times of the schedulers, which basically match the relative performance of the schedulers in terms of number of instructions executed. With 32k flows the number of cache misses is instead much higher, and hence influences execution times at a larger extent.

Along this line, with 32k flows the difference between the execution time with QFQ+, QFQ or DRR and the execution time with FIFO is no more a reliable measure of the net execution time of any of the three schedulers. In fact, cache misses may affect the execution time of the packet generation and discard tasks too, and the number of cache misses with FIFO differs from that with the other schedulers.

Execution time and cache misses - Bursty arrivals. Our results with a bursty arrival pattern are shown in figures 6 and 7. With *1k-w1* and *1k-wmix*, the relative execution times of the schedulers are about the same as in Figure 4, because cache misses are really few.

The effects of a different arrival pattern come into play with 32k flows. First, the total execution time of all the schedulers is higher than in Figure 4, because bursty arrivals conflict, differently from smooth arrivals, with the smooth service order provided by QFQ+, DRR and QFQ to low-weight flows (which are the major part in every flow set). Besides, the execution time of QFQ+ is not lower than that of DRR with *32k-mix* any more, because the bursty service of DRR for high-weight flows is now closer to the order in which packets are generated for those flows. Finally, we can note that the total execution time of FIFO decreases with respect to smooth arrivals, because with bursty arrivals FIFO causes a lower number of cache misses.

Energy consumption. QFQ+ causes less cache misses than DRR and QFQ in both scenarios with 32k flows, i.e., where the number of cache misses is not negligible (figures 5 and 7). According to the models in [20, 21], this fact should guarantee that the relative performance of QFQ+ with respect to DRR and QFQ, in terms of energy consumption, is about the same as, or is even better than, the relative performance of QFQ+ in terms of execution time (figures 4 and 6). We have verified that this property does

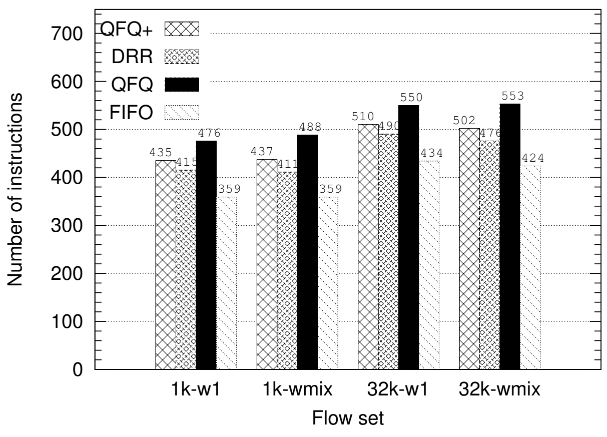

Figure 3: Number of instruction executed with smooth arrivals (lower is better).

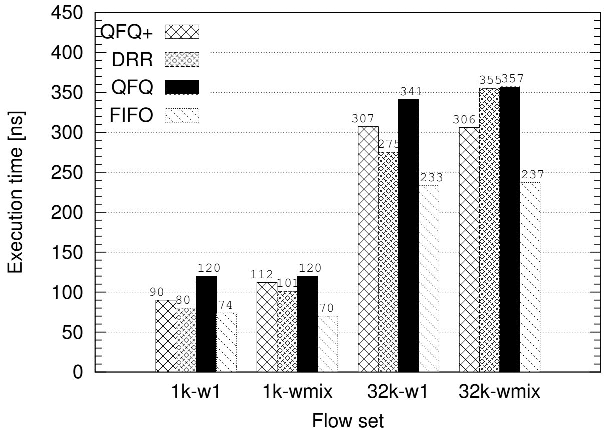

Figure 4: Execution times with smooth arrivals (lower is better).

hold on our system using a power meter.

8.3. Service-guarantee comparison

To make it harder for the schedulers to preserve low packet delays and service lags, in our experiments about service guarantees we have also let the controller randomly switch to the enqueue phase, with probability 0.5, every time a flow queue got empty. Besides, in the simulated environment, the size of the transmit queue is equal to just one packet, i.e., $Q_{pkts} = 2$ (Section 3.1). This is the configuration where, according to (2), (4) and, e.g., Table 2, the degradation of the guarantees of QFQ+ with respect to QFQ is more evident. For coherence with the experiments about efficiency, we have also set $M = 8$ for QFQ+, even if M would have been equal to 2 in the Linux implementation of QFQ+ (Section 3.1).

Figure 8 shows the T-WFIs recorded over all the runs, for only the flows with maximum weight, i.e., with minimum packet service time (Definition 1). For each flow set,

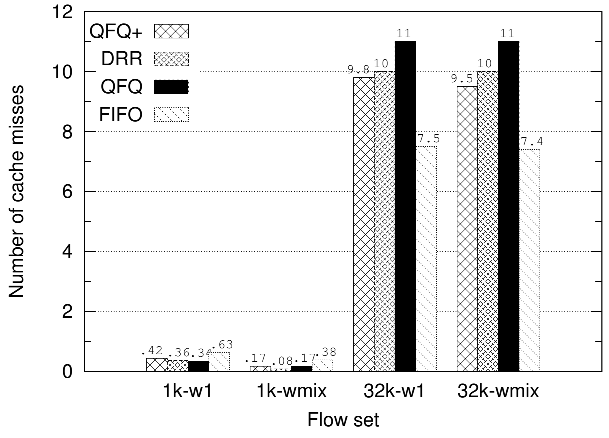

Figure 5: Number of cache misses with smooth arrivals (lower is better).

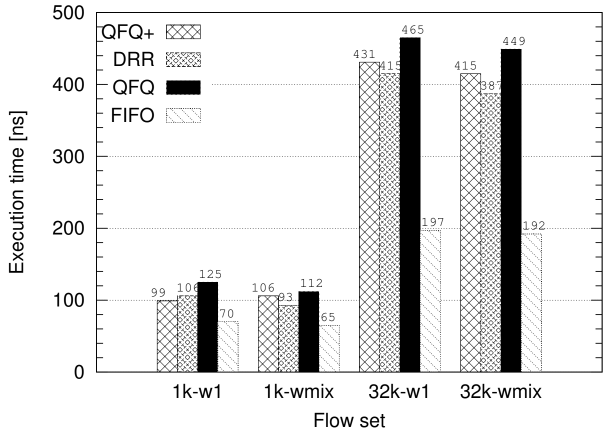

Figure 6: Execution times with bursty arrivals (lower is better).

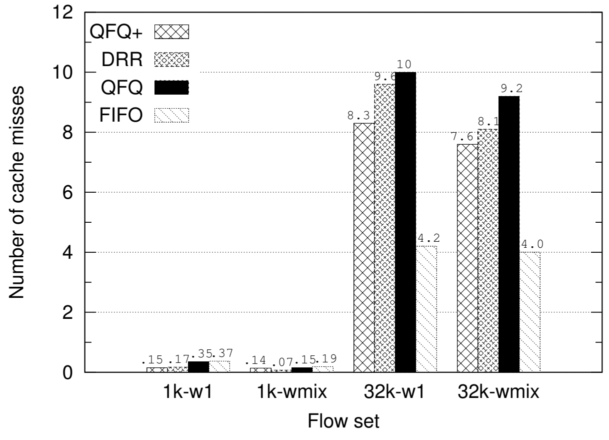

Figure 7: Number of cache misses with bursty arrivals (lower is better).

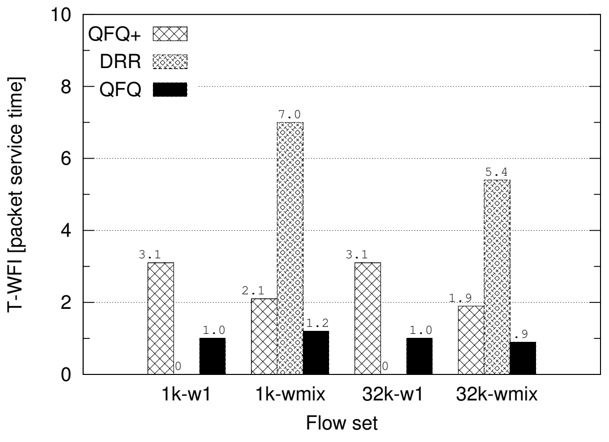

Figure 8: T-WFIs, and hence maximum packet delay variations, for the flows with the shortest packet service times (lower is better). Measured in packet service time for the maximum-weight flow.

the T-WFIs are measured in multiples of the packet service time for the maximum-weight flows. As highlighted in Section 6, these are the flows for which the accuracy of the service guarantees matters most. In this respect, in our experiments we have considered also a more skewed scenario, namely *1k-highw*, but, for scale issues, the figure does not show T-WFIs for this scenario. We report them apart below.

According to Figure 8, the T-WFIs guaranteed by QFQ+ and QFQ are close to each other, and are closer to packet service times than the worst-case bounds in Table 2. In particular, they are so low to make most time-sensitive applications feasible (we remind that T-WFIs are a measure of delay variations as well). It is worth noting that the T-WFI of DRR is instead 0 in the perfectly symmetric scenarios, whereas it is much higher than those of QFQ+ and QFQ in the other scenarios. In particular, it grows as the ratio between the maximum and the minimum packet service time grows (which happens when moving from *32k-wmix* to *1k-wmix*).

This property is confirmed by our results with the much more skewed flow set *1k-highw*, where the T-WFIs of QFQ+, DRR and QFQ, measured in packet service times for the highest-weight flow, are: 2.744, 249.0 and 1.244. The T-WFI of DRR becomes absolutely unbearable, while QFQ and QFQ+ preserve an accurate service.

9. Conclusions

In this paper we have presented a modification scheme for reducing the execution time of the family of the most accurate packet fair-queueing schedulers, and have showed how the service guarantees of the modified schedulers may change with respect to those of the original ones. Applying this scheme to QFQ, we have defined QFQ+, a new scheduler that is definitely more efficient than QFQ, and that has replaced the latter in mainline Linux.

Appendices

A. Proofs of the B-WFI and T-WFI of SCHED

In this appendix we prove the B-WFI and the T-WFI of SCHED in case SCHED is either WF^{2}Q or WF^{2}Q+, or else any approximate variant of any of the previous two schedulers. In this respect, the worst-case guarantees provided WF^{2}Q+ or by one of approximate variants of WF^{2}Q+ considered in this paper [6, 7, 5] are, respectively, equal to those provided by WF^{2}Q or by the same approximate variant, if applied to WF^{2}Q. For this reason, in this appendix we outline only the WF^{2}Q+ algorithm. Besides, for brevity, hereafter we use the generic name AFQ (Approximated Fair Queueing), to refer to any of these variants and to WF^{2}Q+ itself.

For ease of notation, only in this appendix, we move flow indexes from subscripts to superscripts. We use instead subscripts for packet indexes. For convenience, all symbols used in this appendix are listed in Table A.3.

Symbol	Meaning
$*^i$	A superscript indicates a quantity related to a flow
$*_m$	A subscript indicates a quantity related to a packet
N	Total number of flows
ΔW	capacity of the FIFO in bits
h, i	Flow index
L, L^i	Max length of any packet in the system/flow
ϕ^i	Weight of flow i
l^i	Length of the head packet in flow i; $l^i = 0$ when the flow is idle
$*(t_1, t_2)$	Given a generic function $*(t)$, the notation $*(t_1, t_2) \equiv *(t_2) - *(t_1)$ indicates the difference between the values in t_2 and t_1
$W(t)$, $W^i(t)$	The "work function", i.e. number of bits transmitted (globally, or for flow i) in $[0, t]$
$V(t)$, $V^i(t)$	System/flow virtual time, see Eq. (A.3)
S^i, F^i, S_m, F_m	Exact virtual start and finish times of flow i or packet m, see Eq. (A.2)
$\hat{S}^i, \hat{F}^i$, $\hat{S}_m, \hat{F}_m$	Approximated flow/packet timestamps, see Section A.3
$\overline{W}(t)$, $\overline{W}^i(t)$	The "work function" describing the input to the FIFO
$\overline{V}(t)$, $\overline{V}^i(t)$	System/flow virtual time corresponding to $\overline{W}(t)$
$\overline{S}^i, \overline{F}^i$	Counterparts of S^i and F^i obtained using $\overline{V}(t)$ instead of $V(t)$ in (A.2)
$\tilde{S}^i, \tilde{F}^i$	Counterparts of $\hat{S}^i$ and $\hat{F}^i$ obtained using $\overline{V}(t)$ instead of $V(t)$ in Eq. (A.2)
$B(t)$	The set of backlogged flows at time t
$Q^i(t)$	Backlog of flow i at time t

Table A.3: Definitions of the symbols used in this appendix.

Besides, we often use the notation

$$f(t_1, t_2) \equiv f(t_2) - f(t_1)$$

where $f(t)$ is a function of the time. We assume that any discontinuous function of the time is left-continuous, i.e., if t_0 is a discontinuity point for a function $f(t)$, then $f(t_0) = \lim_{\epsilon \to 0} f(t_0 + |\epsilon|)$, and $f(t_0^-) = \lim_{\epsilon \to 0} f(t_0 - |\epsilon|)$.

A.1. System model

Consider a system as in Figure A.9 A, in which N packet flows (defined in whatever meaningful way) share a common transmission link serving one packet at a time. The link has a time-varying rate, with $W(t)$ being its "work function", or the total number of bits transmitted in $[0, t]$. A system is called *work conserving* if the link is used at full capacity whenever there are packets queued. A scheduler (the AFQ block in the figure) sits between the flows and the link: arriving packets are immediately enqueued, and the next packet to serve is chosen and dequeued by the scheduler when the link is ready.

In our model, each flow i is assigned a fixed weight $\phi^i > 0$. Without losing generality, we assume that $\phi = \sum_{i=1}^{N} \phi^i \leq 1$.

A flow is defined *backlogged/idle* if it owns/does not own packets not yet completely transmitted. We call $B(t)$ the set of flows backlogged at time t. Each flow uses a FIFO queue to hold the flow's own backlog.

We call *head packet* of a flow the packet at the head of the queue, and l^i its length; $l^i = 0$ when a flow is idle. We say that a flow is *receiving service* if one of its packets is being transmitted. Both the amount of service $W^i(t_1, t_2)$ received by a flow and the total amount of service $W(t_1, t_2)$ delivered by the system in the time interval $[t_1, t_2]$ are measured in number of bits transmitted during the interval.

The analysis of the schedulers considered in this paper uses the concept of *corresponding systems* [9, Definition 1]: two systems are corresponding if they have the same work function $W(t)$, serve the same set of flows with the same weights in both systems, and are subject to the same arrival pattern.

A.2. WF^{2}Q+

In this section we outline the WF^{2}Q+ algorithm for the case of a variable-rate link (see [10, 22] for a complete description). WF^{2}Q+ is a *packet scheduler* that approximates, on a packet-by-packet basis, the service provided by a corresponding work-conserving *ideal fluid system* that delivers the following, almost perfect bandwidth distribution over any time interval during which a flow is continuously backlogged:

$$W^i(t_1, t_2) \geq \phi^i W(t_1, t_2) - (1 - \phi^i)L \qquad \text{(A.1)}$$

The fluid and the packet system differ in that the former may serve multiple packets in parallel, whereas the latter has to serve one packet at a time, and is non preemptive. To define the scheduling policy of WF^{2}Q+, we need to introduce the concept of *eligibility*, first defined in [9, Section 3]: a packet is defined as *eligible* at a given time instant if it has already started in the fluid system by that time. Accordingly, we define a flow as eligible if its head packet is eligible.

WF^{2}Q+ operates as follows. Each time the link is ready, the scheduler starts to serve, among the eligible packets, the next one that would be completed in the fluid system; ties are arbitrarily broken. WF^{2}Q+ is a work-conserving on-line algorithm, hence it succeeds in finishing packets in the same order as the ideal fluid system, except when the next packet to serve arrives after one or more out-of-order packets have already started.

A.2.1. Virtual Times

The WF^{2}Q+ policy is efficiently implemented by considering, for each flow, a special *flow virtual time* function $V^i(t)$ that grows as the *normalized* amount of service received by the flow (i.e., actual service received, divided by the flow's weight). Besides, when the flow turns from idle to backlogged, $V^i(t)$ is set to the maximum between its current value and the value of a further function, the system virtual time $V(t)$, defined below.

In addition to $V^i(t)$, each flow is conceptually[10] associated with a virtual time $V^i_{fluid}(t)$ also in the fluid system. $V^i(t)$ and $V^i_{fluid}(t)$ are computed with the same rules, but their values differ as the instantaneous distribution of work is different in the packet and in the corresponding fluid system.

For every packet of flow i, we define the virtual *start* and *finish time* of the packet as the value of $V^i(t)$ when the packet starts and finishes to be served in the fluid system. We then define the *virtual start* and *finish time* of flow i, $S^i(t)$ and $F^i(t)$, as the virtual start and finish times of its head packet at time t. These timestamps need to be updated only when the flow becomes backlogged, or when its head packet is dequeued. On these events $S^i(t)$ and $F^i(t)$ are updated as follows:

$$\begin{aligned} S^i(t_p) &\leftarrow \begin{cases} \max(V(t_p),\ F^i(t_p^-)) & \text{on newly backlogged flow;} \\ F^i(t_p^-) & \text{on packet dequeue;} \end{cases} \\ F^i(t_p) &\leftarrow S^i(t_p) + l^i/\phi^i \end{aligned} \qquad \text{(A.2)}$$

where t_p is the time when a packet enqueue/dequeue occurs, and l^i is the packet size. $V(t)$ is the *system virtual time* function defined as follows (assuming $\sum \phi^i \leq 1$):

$$V(t_2) \equiv \max \left\{ V(t_1) + W(t_1, t_2),\ \min_{i \in B(t_2)} S^i(t_2) \right\} \qquad \text{(A.3)}$$

Note how the instantaneous link rate needs not be known to update $V(t)$: just $W(t_1, t_2)$ (the amount of data transferred in $[t_1, t_2]$) suffices. At system start up, $V(0) = 0$, $S^i(0) \leftarrow 0$ and $F^i(0) \leftarrow 0$. The scheduling policy of WF^{2}Q+ is implemented using only $V(t)$, and the virtual start and finish times of the flows, as detailed in the next paragraph.

[10]This parameter is not needed in the implementation but we use it to prove Lemma 4.

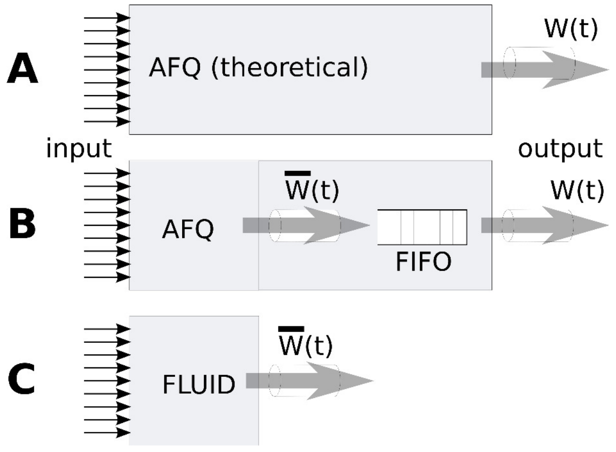

Figure A.9: A: the system model used in the literature, ignoring the presence of the FIFO and assuming $W(t)$ is known exactly within the scheduler. B: a real system, made of an AFQ scheduler feeding a dequeue unit with work function $\overline{W}(t)$, followed by a FIFO and the output link. C: the corresponding fluid system for the first part of B), serving multiple packets at a time, with the same work function $\overline{W}(t)$.

A.2.2. Scheduling decisions

In terms of virtual times, flow i is *eligible* at time t if $V(t) \geq S^i(t)$. In addition, the fluid system serves flows so as to complete packets in virtual finish time order [10, 22]. WF^{2}Q+ can then be implemented as follows: each time the next packet to transmit is requested, the scheduler (dequeues and) returns the head packet of the eligible flow with the smallest virtual finish time. The second argument of the max operator in Eq. (A.3) guarantees that the system is work-conserving.

A.3. Approximate variants of WF^{2}Q+

The exact WF^{2}Q+ algorithm as described above, has $\Omega(\log N)$ complexity in the number of flows [23]. In order to implement the same policy in $O(1)$ time, the approximate variants in the AFQ family [6, 7, 5] label flows with approximated virtual start and finish times $\hat{S}^i(t)$ and $\hat{F}^i(t)$, in addition to the exact values defined in (A.2). The approximated values help reducing the complexity of certain sorting stages in the algorithm, making them constant-time operations.

The way approximations are computed varies, but in all cases we can write:

$$\begin{aligned} S^i(t) - \Delta S^{i-} \leq \hat{S}^i(t) \leq S^i(t) \leq \\ F^i(t) \leq \hat{F}^i(t) \leq S^i(t) + \Delta S^{i+} \end{aligned} \tag{A.4}$$

where ΔS^{i-} and ΔS^{i+} are non-negative quantities ($\Delta S^{i-} = 0$ and $\Delta S^{i+} = \frac{L^i}{\phi^i}$ in WF^{2}Q+).

AFQ uses the approximated timestamps to compute the virtual time, i.e., it uses $\hat{S}^i(t_2)$ instead of $S^i(t_2)$ in (A.3), and to choose the next packet to transmit (Sec A.2.2), while it uses the exact timestamps to charge flows for the work received (Eq. (A.2)).

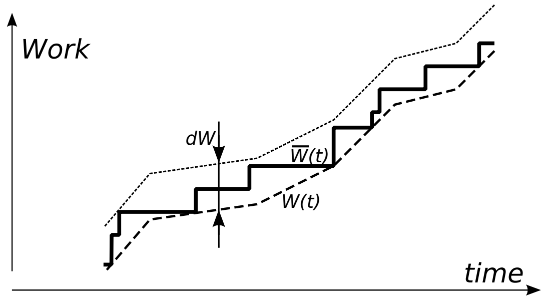

Figure A.10: $\overline{W}(t)$, the number of bits extracted from the scheduler, is within a band of height ΔW (the size of the FIFO) above $W(t)$, the number of bits transmitted by the link.

A.4. Introducing an output queue

The service guarantees of packet schedulers are generally computed on the model in Figure A.9 A, which assumes that i) the exact value of $W(t)$ is known when updating $V(t)$, and ii) the link requests a new packet to transmit only once the previous one has been fully transmitted. Unfortunately, these two assumptions are almost never true in a real system.

First, network interfaces (commonly called *NICs*) operate on a packet-by-packet basis, and do not export a real-time indication of the number of bits transmitted. The data rate is not constant due to framing, link contention and link level flow control; even the notification of transmission completions, available through memory mapped registers or interrupts, can be delayed by several microseconds, corresponding to tens/hundreds of packets on high speed links. Secondly, to make sure that the link does not remain idle while the scheduler (firmware or software) provides the next packet to send, NICs typically implement a *ring buffer*, where the operating system can push outgoing transmissions, and an internal FIFO that drains the ring buffer and drives the link.

A realistic model of a communication device is thus the one in Figure A.9 B, where the scheduler is drained by a dequeue unit that takes care of inserting packets in the FIFO queue[11]. We denote as $\overline{W}(t)$ the sum of the sizes of the packets dequeued from AFQ during $[0, t]$. In other words, $\overline{W}(t)$ is the amount of work *delivered to the FIFO* up to time t. The FIFO is often filled and drained in bursts, as traffic arrive or low/high water marks are reached, so the work function $\overline{W}(t)$ has a stepwise shape (Figure A.10), and lies in a band of height ΔW on top of $W(t)$, where ΔW equals the maximum capacity of the FIFO.

$$\Delta W = \max_t \overline{W}(t) - W(t) \geq 0. \tag{A.5}$$

The presence of the FIFO and the different work functions used to drain the scheduler may change the order in which packets are queued between the ideal (Figure A.9 A) and the actual (Figure A.9 B) systems.

[11]For brevity we will use the term *FIFO* to indicate all the buffering that is present between the scheduler and the link.

Our goal is to evaluate the service properties of systems modeled as in Figure A.9 B, hence taking into account the impact of FIFOs and the uncertainty on the work function $W(t)$. We achieve this goal in two steps:

1. We first prove a few preliminary lemmas (Section A.7) used in the proofs of the main theorems;
2. We then prove the main results (Section A.10), namely the B-WFI and T-WFI for a generic AFQ scheduler in presence of a FIFO and approximated work function.

A.5. Proof machinery

We compute both the B-WFI and the T-WFI of AFQ schedulers by lower-bounding the amount of service $W^i(t_1, t_2)$ given by the system in Figure A.9 B to a flow during any time interval in which the flow is continuously backlogged in the scheduler[12]. To lower-bound $W^i(t_1, t_2)$ we model our system as the cascade shown in Figure A.9 B. The scheduler uses $\overline{W}(t)$ as the work function, and Equations A.2 and A.3 to compute the virtual time $\overline{V}(t)$ and the timestamps $\overline{S}^i(t)$ and $\overline{F}^i(t)$. We call $\tilde{S}^i(t)$ and $\tilde{F}^i(t)$ the approximate versions of the flows' timestamps. The virtual time $\overline{V}(t)$ then becomes

$$\overline{V}(t_2) \equiv \max\left\{\overline{V}(t_1) + \overline{W}(t_1, t_2), \min_{i \in B(t_2)} \tilde{S}^i(t_2)\right\} \quad \text{(A.6)}$$

and the relation between timestamps is

$$\begin{aligned} \overline{S}^i(t) - \Delta S^{i-} \le \tilde{S}^i(t) \le \overline{S}^i(t) \le \\ \overline{F}^i(t) \le \tilde{F}^i(t) \le \overline{S}^i(t) + \Delta S^{i+} \end{aligned} \quad \text{(A.7)}$$

For some derivations, we compare the behaviour of the (packet) AFQ scheduler on the left of Figure A.9 B, with its *corresponding fluid system* shown in Figure A.9 C.

While it may seem counterintuitive, we reduce the lower bound to $W^i(t_1, t_2)$ to one on $\overline{W}^i(t_1, t_2)$, and then in Section A.9 we compute two slightly different bounds to $\overline{W}^i(t_1, t_2)$. From the first bound we immediately get the T-WFI. Using the second bound we compute instead the B-WFI and the T-WFI by a few simple steps.

A.6. Problem reduction

The bound on $W^i(t_1, t_2)$ can be derived by one on $\overline{W}^i(t_1, \hat{t})$, for a suitable interval $[t_1, \hat{t}]$. To this purpose, let $I = [\hat{t}, t_2]$ be the interval such that all packets that are in the FIFO or under service[13] at time t_2 have been dequeued during I. By definition, the sum of the sizes of all these packets is $\overline{W}^i(\hat{t}, t_2)$, and we have

$$0 \le \overline{W}^i(t_2) - W^i(t_2) \le \overline{W}^i(\hat{t}, t_2) \quad \text{(A.8)}$$

Using these two inequalities, we can then write:

$$\begin{aligned} W^i(t_1, t_2) = W^i(t_2) - W^i(t_1) \ge \\ \max\{0, W^i(t_2) - \overline{W}^i(t_1)\} \ge \\ \max\{0, \overline{W}^i(t_2) - \overline{W}^i(\hat{t}, t_2) - \overline{W}^i(t_1)\} = \\ \max\{0, \overline{W}^i(t_2) - \overline{W}^i(t_2) + \overline{W}^i(\hat{t}) - \overline{W}^i(t_1)\} = \\ \max\{0, \overline{W}^i(\hat{t}) - \overline{W}^i(t_1)\} = \overline{W}^i(t_1, \max\{t_1, \hat{t}\}) \end{aligned} \quad \text{(A.9)}$$

Intuitively, this equation relates the output service in $[t_1, t_2]$ with the service given to the scheduler during a shorter interval which excludes packets still staged in the FIFO or in the link.

In Section A.9, Lemma 7 we compute an upper bound to $\overline{W}^i(t_1, t_2)$ for any time interval $[t_1, t_2]$ during which flow i is continuously backlogged. In case $\hat{t} > t_1$, from this bound we get immediately an upper bound to $\overline{W}^i(t_1, \hat{t})$ by just replacing t_2 with $\hat{t}$ (in fact, also $[t_1, \hat{t}]$ is a time interval during which flow i is continuously backlogged).

A.7. Supporting lemmas

The following lemmas are intermediate results needed to prove our service-guarantee bounds. We present them in bottom-up order.

Notation: For the reader's convenience, on top of various equality or inequality signs we write the reason (typically a reference to one equation) why the relation holds.

We start by comparing the completion time of packet transmissions in the packet and the fluid systems. Let p_m be the $m-$th packet served in the packet system (the two systems do not necessarily complete packets in the same order) and call t_m^p and t_m^f their completion times in the two systems.

Lemma 3. *If $\overline{F}_m^i = \tilde{F}_m^i$ then $t_m^p \le t_m^f + \Delta T_L$, where ΔT_L is such that $\overline{W}(t_m^f, t_m^f + \Delta T_L) = L$ (if the exact and the approximated virtual finish time of packet p_m are equal, then p_m will complete in the packet system no later than in the fluid one, plus a worst-case delay equal to ΔT_L).*

Intuition for the proof: the packet system serves packets in strict finish time order, except when packets are not eligible or not arrived. For all in-order bursts immediately after an out-of-order packet, the fluid system cannot have started serving any of the packets in the burst before the beginning of the burst in the packet system, so it must finish the burst no earlier than the packet system.

Proof. Let o be the smallest index for which all packets have an approximated finish time no greater than p_m, $\tilde{F}_i \le \overline{F}_m \forall i \in [o..m]$. Since $\overline{F}_i \le \tilde{F}_i$, and the fluid system (using exact timestamps) completes packets in finish time order,

[12]The quantity of interest is $W^i(t_1, t_2)$, i.e. packets actually exiting the link, and not $\overline{W}^i(t_1, t_2)$, which only refers to packets entering the FIFO.

[13]We will often refer to these packets in the next sections. These packets have been fully served as far as AFQ is concerned, but have not yet emerged from the link, so from the user's perspective they are not served yet.

all packets $p_o..p_m$ must also be completed not earlier than t^f_m in the fluid system.

If $o = 1$ then from the origin of time the packet system has served only packets $p_1..p_m$, while the fluid system might have already started service for some subsequent packet. Remembering that both systems have the same work function, the fluid system cannot be ahead of the packet system, hence $t^p_m \le t^f_m$.

If $o > 1$, then packet p_{o-1} has a higher finish time than $p_o \dots p_m$, none of which has started in the packet system before t^p_{o-1}. This means that at t^p_{o-1} either they had arrived yet, or they were not eligible ($\tilde{S}^i(t^P_{o-1}) > \overline{V}(t^P_{o-1})$), so even the fluid system cannot have started serving any of those packets before t^p_{o-1} (the fluid system starts to serve a flow at time t only if $\overline{S}^i(t) \le \overline{V}(t)$ and $\overline{S}^i(t) \ge \tilde{S}^i(t)$ holds). However, some of the packets $p_o \dots p_m$ may become eligible while the fluid system works at time t^p_{o-1}, and hence also these packets may receive some service in the fluid system at time t^P_{o-1}. Besides, the work function of the fluid system increases by at most L at time t^p_{o-1}. As a consequence, between t^p_o and t^p_m the fluid system must have done at least the same amount of work as the packet system, minus at most L. Hence t^f_m cannot precede t^p_m by more than ΔT_L.

$\square$

A.8. Globally Bounded Timestamps

The flow timestamps cannot deviate too much from the system's virtual time $\overline{V}(t)$. This is the "Globally Bounded Timestamp" property (GBT) defined in [7, Definition 3]. Here we compute a variant of this property that comes in handy to upper-bound $\overline{W}^i(t_1, t_2)$.

Lemma 4 (Lower bound for $\tilde{F}^i(t)$)**.** *For all times t at which flow i is backlogged,*

$$\overline{V}(t) - \tilde{F}^i(t) \le L \tag{A.10}$$

Proof. Let $\bar{t}$ be a generic time instant at which flow i is backlogged, with p_m at its head. We know that $\overline{F}^i(\bar{t}) \le \tilde{F}^i(\bar{t})$. If $\overline{F}^i(\bar{t}) = \tilde{F}^i(\bar{t})$, Lemma 3 tells us that the transmission completion times in the packet and fluid systems are $t^P_m \le t^F_m + \Delta T_L$. Besides, denoted as $\overline{V}^i_{fluid}(t)$ the virtual time of flow i in the fluid system, the latter guarantees that $V(t) \le \overline{V}^i_{fluid}(t)$ holds at all times. Thus

$$\begin{aligned}\overline{V}(\bar{t}) \le \overline{V}^i_{fluid}(\bar{t}) &\overset{\bar{t} \le t^P_m}{\le} \overline{V}^i_{fluid}(t^P_m) \overset{t^P_m \le t^F_m + \Delta T_L}{\le} \\ &\overline{V}^i(t^P_m) + L \overset{(A.2)}{=} \tilde{F}^i(\bar{t}) + L\end{aligned} \tag{A.11}$$

The case $\overline{F}^i(\bar{t}) < \tilde{F}^i(\bar{t})$ can be handled by considering what happens if packet p_m is artificially extended so $\overline{F}^i(\bar{t}) = \tilde{F}^i(\bar{t})$. The larger packet would still satisfy (A.11). Besides, whether or not the original packet p_m is replaced with a larger one, the values of $\overline{V}(\bar{t})$ and $F^i(\bar{t})$ are the same, because 1) the value of t^p_m does not depend on the size of p_m, 2) $\bar{t} < t^p_m$, and 3) the size of p_m does not influence either any timestamp or the packet service order up to time t^p_m. Hence the thesis holds also in this case. $\square$

Lemma 5 (Upper bound for $\overline{S}^i(t)$)**.** *At all times t*

$$\overline{S}^i(t) \le \overline{V}(t) + \Delta S^{i-} + \frac{L^i}{\phi^i} - L^i \tag{A.12}$$

Note: differently from the previous bound, this bound applies to the exact timestamp, as this is what we need in the proof of subsequent Lemma 6.

Proof. Given any time instant t, we consider the smallest time instant t_p such that $\overline{S}^i(t_p) = \overline{S}^i(t)$, and we denote as p_m the packet served at time t_p, and l_m its size. According to (A.2), either $\overline{S}^i(t_p) = \overline{V}(t_p) \le \overline{V}(t)$ or $\overline{S}^i(t_p) = \overline{F}^i(t^-_p)$). In the first case the thesis holds trivially. For the other case to hold, at least one packet of flow i must have been already served before time t_p. Let t'_p be the largest time instant, with $t'_p < t_p$, at which a packet of flow i is served. Flow i has to be eligible at time t'_p, i.e., $\tilde{S}^i(t^-_p) = \tilde{S}^i(t'_p) \le \overline{V}(t'_p) \le V(t^-_p)$ has to hold. Besides, we can note that the virtual time advances by at least the size of p_m at time t_p, thus $\overline{V}(t^-_p) \le \overline{V}(t_p) - L_m \le \overline{V}(t) - L_m$ holds. In the end, $\tilde{S}^i(t^-_p) \le \overline{V}(t) - L_m$. Using this inequality, we can write

$$\begin{aligned}\overline{S}^i(t_p) = \overline{F}^i(t^-_p) &\overset{(A.2)}{=} \overline{S}^i(t^-_p) + \frac{L_m}{\phi^i} \overset{(A.4)}{\le} \\ &\tilde{S}^i(t^-_p) + \Delta S^{i-} + \frac{L_m}{\phi^i} \le \\ &\overline{V}(t) - L_m + \Delta S^{i-} + \frac{L_m}{\phi^i} \overset{\phi^i \le 1}{\le} \\ &\overline{V}(t) - L^i + \Delta S^{i-} + \frac{L^i}{\phi^i}\end{aligned} \tag{A.13}$$

$\square$

A.9. Lower bounds for $\overline{W}^i(t_1, t_2)$

The second lemma in this section contains a lower bound[14] to $\overline{W}^i(t_1, t_2)$, expressed in terms of the the work function $W(t)$. We substitute this bound in (A.9) to compute the B-WFI of AFQ schedulers.

Lemma 6.

$$\begin{aligned}&\overline{W}^i(t_1, t_2) \ge \\ &\phi^i \overline{V}(t_1, t_2) + \\ &-\phi^i \left(\frac{L^i}{\phi^i} + \Delta S^{i-} + \Delta S^{i+} + L - L^i \right)\end{aligned} \tag{A.14}$$

[14]Remembering that by definition $\overline{W}^i(t_1, t_2) \ge 0$, we could derive tighter bounds by writing $\overline{W}^i(t_1, t_2) \ge \max\{0, ...\}$. However this would make the result even less readable, and it is hardly useful given that the equation is later used in a context where we take the maximum over any flow and/or time intervals.

Proof. Recalling the meaning of the virtual time $\overline{V}^i(t)$ of flow i, we can write the following equalities, where the last equality follows from summing and subtracting $\overline{V}(t_2) - \overline{V}(t_1)$ to $V^i(t_2) - \overline{V}^i(t_1)$:

$$\begin{aligned} \overline{W}^i(t_1, t_2) &= \\ \phi^i \overline{V}^i(t_1, t_2) &= \\ \phi^i \left[\overline{V}^i(t_2) - \overline{V}^i(t_1)\right] &= \\ \phi^i \left[(\overline{V}(t_2) - \overline{V}(t_1)\right] &+ \\ \phi^i \left[\overline{V}^i(t_2) - \overline{V}(t_2) - (\overline{V}^i(t_1) - \overline{V}(t_1))\right] & \end{aligned} \tag{A.15}$$

We can therefore prove the thesis by computing lower bounds to the two terms $\overline{V}^i(t_2) - \overline{V}(t_2)$ and $-(\overline{V}^i(t_1) - \overline{V}(t_1))$. Remembering that by definition $\overline{V}^i(t) = \overline{S}^i(t)$, for the first term we have

$$\begin{aligned} \overline{V}^i(t_2) - \overline{V}(t_2) = \overline{S}^i(t_2) - \overline{V}(t_2) &\overset{(A.7)}{\geq} \\ \tilde{F}^i(t_2) - \Delta S^{i+} - \overline{V}(t_2) &\overset{(A.10)}{\geq} \\ -\Delta S^{i+} - \frac{L^i}{\phi^i} - L & \end{aligned} \tag{A.16}$$

As for the second term, we have

$$\begin{aligned} -\left[\overline{V}^i(t_1) - \overline{V}(t_1)\right] &= \\ -\left[\overline{S}^i(t_1) - \overline{V}(t_1)\right] &\overset{(A.12)}{\geq} \\ -\left[\overline{V}(t_1) + \Delta S^{i-} + \frac{L^i}{\phi^i} - L^i\overline{V}(t_1)\right] &= \\ -\left[\Delta S^{i-} + \frac{L^i}{\phi^i} - L^i\right] & \end{aligned} \tag{A.17}$$

Replacing the two bounds in (A.15), and rearranging terms, we get the thesis. □

Lemma 7.

$$\begin{aligned} \overline{W}^i(t_1, t_2) \geq \phi^i \left(\overline{W}(t_2) - W(t_1)\right) &+ \\ -\phi^i \left(\frac{L^i}{\phi^i} + \Delta S^{i-} + \Delta S^{i+} + \Delta W + L - L^i\right) & \end{aligned} \tag{A.18}$$

Proof. We prove the thesis by upper-bounding the term $\overline{V}(t_1, t_2)$ in (A.14) as follows:

$$\begin{aligned} \overline{V}(t_2) - \overline{V}(t_1) \geq \overline{W}(t_2) - \overline{W}(t_1) &\overset{(A.5)}{\geq} \\ \overline{W}(t_2) - W(t_1) - \Delta W. & \end{aligned} \tag{A.19}$$

□

A.10. Service properties

We are now ready to compute the B-WFI and the T-WFI of the AFQ family of schedulers.

A.10.1. B-WFI

The B-WFI^i for a flow i is defined as:[15]

$$\text{B-WFI}^i \equiv \max_{[t_1, t_2]} \left\{\phi^i W(t_1,\, t_2) - W^i(t_1,\, t_2)\right\} \tag{A.20}$$

where $[t_1, t_2]$ is any time interval during which the flow is continuously backlogged, $\phi^i W(t_1,\, t_2)$ is the minimum amount of service the flow should have received according to its share of the link bandwidth, and $W^i(t_1,\, t_2)$ is the actual amount of service provided by the scheduler to the flow.

Theorem 4 (B-WFI)**.** *For a flow i, AFQ guarantees*

$$\begin{aligned} \text{B-WFI}^i \leq \phi^i \Delta W + \phi^i \left(\Delta S^{i+} + \Delta S^{i-}\right) &+ \\ + (1 - \phi^i) L^i + L & \end{aligned} \tag{A.21}$$

Proof. By substituting (A.9) in (A.20), we get

$$\begin{aligned} \text{B-WFI}^i &\overset{(A.9)}{\leq} \\ \phi^i W(t_1,\, t_2) - \overline{W}^i(t_1, \max\{t_1, \hat{t}\}) &\overset{(A.18)}{\leq} \\ \phi^i W(t_1,\, t_2) - \phi^i \left(\overline{W}(\max\{t_1, \hat{t}\}) - W(t_1)\right) &+ \\ +\phi^i \left(\frac{L^i}{\phi^i} + \Delta S^{i-} + \Delta S^{i+} + \Delta W + L - L^i\right) &\leq \\ \phi^i W(t_1,\, t_2) - \phi^i \left(W(t_2) - W(t_1)\right) &+ \\ +\phi^i \left(\frac{L^i}{\phi^i} + \Delta S^{i-} + \Delta S^{i+} + \Delta W + L - L^i\right) & \end{aligned} \tag{A.22}$$

□

A.10.2. T-WFI

For a link with a constant rate R, the T-WFI^i for flow i is defined as

$$\text{T-WFI}^i \equiv \max\left(t_c - t_a - \frac{Q^i(t_a)}{\phi^i R}\right) \tag{A.23}$$

where t_a and t_c are, respectively, the arrival and completion time of a packet, and $Q^i(t_a)$ is the backlog of flow i just after the arrival of the packet.

Theorem 5 (T-WFI)**.** *For a flow i, AFQ guarantees*

$$\text{T-WFI}^i \leq \frac{L^i}{\phi^i R} + \frac{\Delta S^{i-} + \Delta S^{i+} + \Delta W + L - L^i}{R} \tag{A.24}$$

Proof. Given a packet p arriving at time t_a, we prove the thesis in two steps: first we compute an upper bound to the time that elapses from t_a to when p is dequeued from AFQ, say time $\overline{t}_c$, then we add to this upper bound the maximum time that may elapse from time $\overline{t}_c$ to the time instant t_c at which p is finally transmitted.

[15] This definition is slightly more general than the original one in [10], where t_2 was constrained to the completion time of a packet.

As for the first step, by definition of $\overline{W}^i(t)$ and $\overline{t}_c$, we have that $\overline{W}^i(t_a, \overline{t}_c) = Q^i(t_a)$. Using this equality and (A.18), and recalling that the link works at constant speed R, we can write

$$\begin{aligned}
\overline{t}_c - t_a \le \frac{W(t_a, \overline{t}_c)}{R} &= \\
\frac{W(t_a, \overline{t}_c) + \overline{W}(\overline{t}_c) - \overline{W}(\overline{t}_c)}{R} &= \\
\frac{\overline{W}(\overline{t}_c) - W(t_a) + W(\overline{t}_c) - \overline{W}(\overline{t}_c)}{R} &\overset{(A.18)}{\le} \\
\frac{W(\overline{t}_c) - \overline{W}(\overline{t}_c)}{R} + \frac{\overline{W}^i(t_a, \overline{t}_c)}{\phi^i R} &+ \\
\frac{\frac{L^i}{\phi^i} - L^i + \Delta S^{i-} + \Delta S^{i+} + \Delta W + L}{R} &= \\
\frac{W(\overline{t}_c) - \overline{W}(\overline{t}_c)}{R} + \frac{Q^i(t_a)}{\phi^i R} &+ \\
+\frac{\frac{L^i}{\phi^i} - L^i + \Delta S^{i-} + \Delta S^{i+} + \Delta W + L}{R}
\end{aligned} \tag{A.25}$$

The thesis follows from considering that, since the FIFO is emptied and the packet on the link is served at a constant rate R, then $t_c - \overline{t}_c = \frac{\overline{W}(\overline{t}_c) - W(\overline{t}_c)}{R}$. □

B. Proof of Theorem 2

Using the same conventions as in (9), we define the *per-aggregate B-WFI* of SCHED+ as follows:

$$\begin{aligned}
&\text{B-}\hat{\text{WFI}}^k_{SCHED+} \equiv \\
&\max_{[t_1,t_2]} \left\{ \min\left[m^k \phi^k W(t_1,t_2), \hat{B}^k(t_1,t_2) \right] - W^k(t_1,t_2) \right\}
\end{aligned} \tag{B.1}$$

where $[t_1, t_2]$ is any time interval during which the k-th aggregate is continuously backlogged, and $\hat{B}^k(t_1, t_2)$ is the sum of the sizes of the super packets of the k-th aggregate that either are pending at time t_1, or arrive in the open interval (t_1, t_2).

We can now enunciate the counterpart of Lemma 1 in terms of B-WFI.

Lemma 8. *If the simplifying assumption reported in Section 5.1 holds, then we have*

$$\begin{aligned}
&\text{B-}\hat{WFI}^k_{SCHED+} \le \\
&m^k\phi^k Q + m^k\phi^k \Delta S^k + (1 - m^k\phi^k) m^k L^k + ML.
\end{aligned} \tag{B.2}$$

Proof. Identical to that of Lemma 1. □

Then we can generalize the previous inequality so as to hold also in case the simplifying assumption does not hold.

Lemma 9. *Regardless of whether the simplifying assumption in Section 5.1 holds, B-$\hat{WFI}^k_{SCHED+}$ is upper-bounded by the right-hand side of* (B.2) *plus* $m^k L^k$.

Proof. If the simplifying assumption always holds for the k-th aggregate, then the lemma trivially holds. Suppose instead that the simplifying assumption may not hold. According to (B.1) and (B.2), the lemma still holds if and only if the following upper bound holds for every time interval $[t_1, t_2]$ during which the k-th aggregate is continuously backlogged:

$$\begin{aligned}
&\min\left[m^k\phi^k W(t_1,t_2), \hat{B}^k(t_1,t_2) \right] - W^k(t_1,t_2) \le \\
&m^k\phi^k Q + m^k\phi^k \Delta S^k + (2 - m^k\phi^k) m^k L^k + ML.
\end{aligned} \tag{B.3}$$

The term $m^k\phi^k Q$ comes from the corresponding term $\phi^k Q$ in (6). It follows that, according to how the term $\phi^k Q$ in (6) is derived in Appendix A, the term $m^k\phi^k Q$ disappears from (B.3) if we replace $W(t_1, t_2)$ and $W^k(t_1, t_2)$ with the sum of the sizes $\overline{W}(t_1, t_2)$ and $\overline{W}^k(t_1, t_2)$ of, respectively, all the packets dequeued during $[t_1, t_2]$ and the packets of the k-th aggregate dequeued during $[t_1, t_2]$ (intuitively, we remove the effects of the transmit queue). In the end, (B.3) holds if and only if the following bound holds:

$$\begin{aligned}
&\min\left[m^k\phi^k \overline{W}(t_1,t_2), \hat{B}^k(t_1,t_2) \right] - \overline{W}^k(t_1,t_2) \le \\
&m^k\phi^k \Delta S^k + (2 - m^k\phi^k) m^k L^k + ML.
\end{aligned} \tag{B.4}$$

For the simplifying assumption to not hold, there must exist at least one super packet $\hat{p}$ such that the aggregate finishes its backlog but not its budget when the last packet in the super packet is dequeued. For this violation of the simplifying assumption to influence the service of the k-th aggregate during $[t_1, t_2]$, the following inequalities must hold: $\hat{t}_a \le t_2$ and $\hat{t}'_c \ge t_1$, where $\hat{t}_a$ and $\hat{t}'_c$ are the arrival time of $\hat{p}$ and the dequeueing time of the last packet in $\hat{p}$ (we use the more cumbersome notation $\hat{t}'_c$, instead of just $\hat{t}_c$, because we use the latter symbol for the completion time of super packets).

We consider two alternatives: 1) $\hat{t}'_c \in [t_1, t_2]$, 2) $\hat{t}'_c \notin [t_1, t_2]$. In the first case, since the aggregate finishes its backlog at time $\hat{t}'_c$, the time interval $[t_1, t_2]$ itself necessarily ends exactly at time $\hat{t}'_c$. This implies that $W^k(t_1, t_2) = \hat{B}^k(t_1, t_2)$, and hence that (B.4) trivially holds.

In the second case, suppose first that the following assumption holds: as of time t_2, at least another super packet needs to be dequeued before dequeueing the first packet in $\hat{p}$. It follows that the service provided by SCHED+ to the k-th aggregate during $[t_1, t_2]$ is exactly the same, regardless of whether $\hat{p}$ even arrived. Hence (B.4) holds by Lemma 1.

If, instead, the assumption does not hold, then $\hat{p}$ is the only super packet of the k-th aggregate that has to be completely dequeued as of time t_2 (because the backlog of the aggregate will finish when the last packet in $\hat{p}$ is dequeued). Since the size of $\hat{p}$ is at most $m^k L^k$, it follows that $W^k(t_1, t_2) \ge \hat{B}^k(t_1, t_2) - m^k L^k$, which, substituted in (B.4), proves the thesis. □

We can finally prove Theorem 2.

Proof. Thanks to Lemma 9, (B.4) holds for any time interval $[t_1, t_2]$ during which the i-th flow, belonging to the k-th aggregate, is continuously backlogged. Dividing both sides of (B.4) by m^k, replacing (16) in the resulting inequality, and rearranging terms, we get

$$\begin{aligned}&\phi^k \min\left[\overline{W}(t_1,\, t_2), B_i(t_1, t_2)\right] - \overline{W}_i(t_1,\, t_2) \leq \\ &\phi^k \Delta S^k + \left(5 - \frac{1}{m^k} - m^k \phi^k\right) L^k + \frac{M}{m^k} L\end{aligned} \tag{B.5}$$

Finally, turning back to the actual number of bits transmitted and considering again the effect of the transmit queue as shown in Appendix A, we have

$$\begin{aligned}&\phi^k \min\left[W(t_1,\, t_2), B_i(t_1, t_2)\right] - W_i(t_1,\, t_2) \leq \\ &\phi^k Q + \phi^k \Delta S^k + \left(5 - \frac{1}{m^k} - m^k \phi^k\right) L^k + \frac{M}{m^k} L\end{aligned} \tag{B.6}$$

□

References

[1] http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/VNI_Hyperconnectivity_WP.html.

[2] L. L. Peterson, B. S. Davie, Computer Networks - A systems Approach, Morgan Kaufmann Publishers, 2010.

[3] L. Rizzo, Netmap: a novel framework for fast packet i/o, in: Proceedings of the 2012 USENIX conference on Annual Technical Conference, USENIX ATC'12, USENIX Association, Berkeley, CA, USA, 2012, pp. 9–9.
URL http://dl.acm.org/citation.cfm?id=2342821.2342830

[4] M. Shreedhar, G. Varghese, Efficient fair queuing using deficit round robin, IEEE/ACM Transactions on Networking 4 (3) (1996) 375–385.

[5] F. Checconi, L. Rizzo, P. Valente, QFQ: Efficient packet scheduling with tight guarantees, Networking, IEEE/ACM Transactions on 21 (3) (2013) 802–816. doi:10.1109/TNET.2012.2215881.

[6] M. Karsten, Approximation of generalized processor sharing with stratified interleaved timer wheels, IEEE/ACM Transactions on Networking 18 (3) (2010) 708–721. doi:10.1109/TNET.2009.2033059.

[7] D. C. Stephens, J. C. Bennett, H. Zhang, Implementing scheduling algorithms in high-speed networks, IEEE Journal on Selected Areas in Communications 17 (6) (June 1999) 1145–1158. doi:10.1109/49.772449.

[8] P. Valente, Providing near-optimal fair-queueing guarantees at round-robin amortized cost, in: Proceedings of the 22nd International Conference on Computer Communication and Networks (ICCCN 2013), ICCCN 2013, 2013.

[9] J. C. R. Bennett, H. Zhang, WF^{2}Q: Worst-case fair weighted fair queueing, Proceedings of IEEE INFOCOM '96 (March 1996) 120–128.

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

[11] http://www.lartc.org/.

[12] http://algogroup.unimore.it/people/paolo/agg-sched/.

[13] C. Guo, SRR: An O(1) time complexity packet scheduler for flows in multi-service packet networks, Proceedings of ACM SIGCOMM 2001 (August 2001) 211–222.

[14] C. Guo, G-3: An $O(1)$ Time Complexity Packet Scheduler That Provides Bounded End-to-End Delay, Proceedings of IEEE INFOCOMM 2007 (May 2007) 1109–1117.

[15] L. Lenzini, E. Mingozzi, G. Stea, Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers, IEEE/ACM Transactions on Networking 12 (4) (2004) 681–693. doi:http://dx.doi.org/10.1109/TNET.2004.833131.

[16] S. Ramabhadran, J. Pasquale, The stratified round robin scheduler: design, analysis and implementation, IEEE/ACM Transactions on Networking 14 (6) (2006) 1362–1373. doi:http://dx.doi.org/10.1109/TNET.2006.886287.

[17] X. Yuan, Z. Duan, Fair round-robin: A low complexity packet scheduler with proportional and worst-case fairness, IEEE Transactions on Computers 58 (3) (2009) 365–379.

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

[19] https://perf.wiki.kernel.org/index.php/Main_Page.

[20] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, 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), 2011, 2011, pp. 1 –6.

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

[22] D. Stiliadis, A. Varma, A general methodology for designing efficient traffic scheduling and shaping algorithms, Proceedings of IEEE INFOCOM '97 (April 1997) 326–335doi:10.1109/INFCOM.1997.635151.

[23] J. Xu, R. J. Lipton, On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms, IEEE/ACM Transactions on Networking 13 (1) (2005) 15–28. doi:http://dx.doi.org/10.1109/TNET.2004.842223.
URL http://dx.doi.org/10.1109/TNET.2004.842223