Computer Communications 47 (2014) 16–33

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

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

ARTICLE INFO

Article history:
Received 16 July 2013
Received in revised form 20 February 2014
Accepted 18 April 2014
Available online 29 April 2014

Keywords:
Packet scheduling
QoS
Fair queueing
Computational cost

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.

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

E-mail address: paolo.valente@unimore.it

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

http://dx.doi.org/10.1016/j.comcom.2014.04.009
0140-3664/

traffic, is to guarantee the latter 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 [5–7]; 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]

1.1. 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.

1.2. 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. Modification scheme

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 Fig. 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 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 kth 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

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

[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.