Efficient Deadline-Based QoS Algorithms for High-Performance Networks

Alejandro Martínez, George Apostolopoulos, *Member*, *IEEE*, Francisco J. Alfaro, José L. Sánchez, and José Duato, *Member*, *IEEE*

Abstract—Quality of service (QoS) is becoming an attractive feature for high-performance networks and parallel machines because, in those environments, there are different traffic types, each one having its own requirements. In that sense, deadline-based algorithms can provide powerful QoS provision. However, the cost associated with keeping ordered lists of packets makes these algorithms impractical for high-performance networks. In this paper, we explore how to efficiently adapt the Earliest Deadline First family of algorithms to high-speed network environments. The results show excellent performance using just two virtual channels, FIFO queues, and a cost feasible with today's technology.

Index Terms—Quality of service, high-speed interconnection networks.

1 Introduction

Modern supercomputers and parallel machines put a lot of pressure on the interconnection network. Nowadays, low-latency and contention-free interconnection networks are demanded for the execution of parallel applications. Moreover, high bandwidth is also required to access storage devices. In addition to these, there is also a need for administration traffic used to configure and manage the machine. Finally, some low-priority traffic-like backup copies are needed. Therefore, there is a great variety of application requirements in such environments.

The usual solution to cope with this variety of communication necessities is to overprovision the network. The designers provide more resources than needed to ensure meeting traffic requirements [1]. Besides, to provide the different classes of traffic with their requirements, it is common to settle on a network for each traffic class (TC). For instance, the recently built supercomputer MareNostrum [2] implements a Myrinet network for parallel applications, a Gigabit Ethernet for storage access, and a regular Ethernet for management purposes. Although Ethernet interfaces are quite cheap nowadays, keeping three times the wires is complex, costly, and power consuming.

A subtler approach could be taken in the design of the interconnection for such machines. A single network with some quality-of-service (QoS) support could be used to provide each kind of traffic with its specific requirements. In fact, some of the latest high-performance interconnection proposals incorporate some support for QoS. In the next section, we will introduce the InfiniBand and PCI AS interconnection standards, which include some QoS mechanisms.

The two main types of QoS support are per-traffic-class and per-flow support. The first approach requires the classification of the traffic in TCs and the assignment of one virtual channel (VC) per TC. The network switches offer a traffic differentiation based on these TCs by applying different scheduling algorithms at the VC level. On the other hand, per-flow QoS support requires a flow identifier to be associated with each packet and per-flow information to be kept at each switch of the network. This second approach is much more powerful, but it is also so complex that it has never been implemented in a high-performance environment, with perhaps the exception of ATM [3].

In this paper, we discuss how to obtain most of the benefits of the per-flow QoS approach within the restrictions of high-performance switches. More specifically, we will propose a novel strategy to emulate the Earliest Deadline First (EDF) family of algorithms by using just a pair of FIFO queues.

The remainder of this paper is structured as follows: In Section 2, the related work is presented. In Section 3, we present our strategy to offer QoS support. Details on the experimental platform are in Section 4 and the performance evaluation is presented in Section 5. Finally, Section 6 summarizes the results of this study.

2 Related Work

In this section, we will review the state of the art in QoS support in wired interconnects with a special attention to the characteristics of per-flow QoS algorithms.

- *A. Martínez is with the Intel Barcelona Research Center, Intel Labs-Universitat Politècnica de Catalunya, C/ Jordi Girona, 29 Edificio Nexus II, 3a planta, 08034 Barcelona, Spain. E-mail: alejandrox.martinez@intel.com.*
- *G. Apostolopoulos is with Redback Networks, 350 Holger Way, San Jose, CA 95134. E-mail: georgeap@redback.com.*
- *F.J. Alfaro and J.L. Sánchez are with the Escuela Politécnica Superior de Albacete, Campus universitario, s/n, 02071 Albacete, Spain. E-mail: {falfaro, jsanchez}@dsi.uclm.es.*
- *J. Duato is with ETS Informática Aplicada, Camino de Vera, s/n, 46022 Valencia, Spain. E-mail: jduato@disca.upv.es.*

Manuscript received 19 Jan. 2007; revised 4 Jan. 2008; accepted 24 Jan. 2008; published online 25 Feb. 2008.
Recommended for acceptance by F. Lombardi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0030-0107.
Digital Object Identifier no. 10.1109/TC.2008.39.

2.1 QoS Support in Packet Networks

Over the last few years, as networks have had to carry more diverse types of traffic, there has been extensive work on how to schedule the resources of a packet switch[1] to provide guaranteed performance to traffic. Performance guarantees usually include bounds on packet loss, delay, jitter (delay variation), and transmission rate or throughput. Achieving these performance guarantees requires both the *scheduling* of the network resources and avoiding their *oversubscription*. Scheduling is implemented through a *service discipline* and oversubscription is avoided through a *Connection Admission Control (CAC)* policy, usually tightly coupled with the service discipline. The network resources that need to be scheduled are buffer space (usually at the outgoing port) at the packet switches and link bandwidth.

Over time, a variety of scheduling disciplines have been proposed, each one targeted at providing certain types of guarantees and representing a different trade-off between implementation complexity and performance guarantees. Scheduling policies can be classified into two broad classes depending on whether they provide *aggregate* or *per-flow* guarantees. A flow is a sequence of packets that belong to the same logical stream and should be treated similarly by the network, for example, a single TCP connection or all of the traffic destined to a certain host. A popular class of service disciplines that can provide per-flow guarantees is based on the concept of *Weighted Fair Queuing (WFQ)* [4]. In this case, individual flows receive a ratio of the available link capacity based on their weight, approximating Generalized Processor Sharing (GPS), a theoretical service discipline that schedules traffic following the fluid model, i.e., scheduling each bit of incoming packets independently. The packetized version of GPS known as PGPS [5] or WFQ [4] has been shown to be able to provide tight end-to-end rate and delay guarantees while remaining fair in how it handles traffic. The downside is that the implementation of this class of scheduling disciplines is complex since they need to approximate the operation of GPS. A number of variants have been introduced (see [6] for a brief review) that not only simplify implementation but also reduce the fairness or the quality of guarantees provided.

The *Earlier Deadline First (EDF)* discipline is a different approach, based on a deadline scheduler. Packets are assigned deadlines and the packet with the earliest deadline is selected for transmission. An advantage of this discipline over the WFQ variants is that EDF can separate the rate and delay guarantees provided; the WFQ policies can only provide a rate guarantee and, indirectly through it, a delay guarantee: A flow that needs very low delays would have to reserve a high rate. In order to provide end-to-end guarantees with EDF schedulers, it is necessary to provide additional shaping at each node. With this addition, it has been shown that the EDF discipline is the most efficient when guaranteeing end-to-end delays [7]. Still, both the simplified WFQ disciplines as well as the EDF algorithm are too difficult to implement since they require separate queues for each flow. Large packet switches may have to handle hundreds of thousands of flows, making fast hardware implementations of these policies impractical.

The above limitation has led to the development of scheduling disciplines that can provide aggregate guarantees. In these disciplines, flows are mapped to a small number of *TCs* that are then scheduled; each flow receives a less accurate aggregate guarantee. Scheduling among classes can be implemented using WFQ-like or EDF disciplines as above. A scheduling discipline specifically targeted to classes is Class-Based Queuing (CBQ) [8], which uses a hierarchy of schedulers for providing different guarantees to different classes of traffic. Aggregate guarantees will not be as hard (i.e., deterministic) as with per-flow scheduling and QoS provision will depend more on CAC. For example, if we can guarantee a given rate for a TC, we will have to be very careful how many flows are admitted in this class so as to ensure some meaningful per-flow guarantee. In certain cases, in order to achieve a relatively tight guarantee, we may have to implement a very strict CAC reducing the utilization of the network.

Based on the above scheduling disciplines, a number of QoS architectures have been proposed. Examples of per-flow-oriented QoS architectures are the QoS architecture of ATM [3] and the Integrated-Services model [9] that was proposed for Internet in the mid-1990s. An example of aggregate-QoS architecture is Differentiated Services [10], which is nowadays used to provide limited QoS in parts of the Internet.

We should note here that, in packet networks, there is another important dimension to scheduling, that of buffer management. Given that there are limited buffers available, service disciplines usually are complemented with a buffer management policy that decides what packets to drop when buffers are exhausted. We will not look further into this aspect of scheduling since high-speed interconnects typically use flow control to throttle senders when there are no available buffers and they never drop packets. Besides, in QoS architectures in packet networks, one has to deal with the potential heterogeneity of the network, which may be using equipment from different vendors with different scheduling discipline implementations. Providing end-to-end aggregate guarantees in such a network can be challenging. On the other hand, there is no such issue when we consider high-speed interconnection networks.

2.2 QoS Support in New High-Speed Interconnects

When compared with a generic packet switch, high-speed interconnect switches exhibit some important differences, mostly because of their much simpler and compact implementation. First, flow control is commonly used to throttle the incoming traffic and, thus, there are usually no packet drops due to running out of buffer space. Buffers themselves are smaller than what one would expect from a generic packet switch. Furthermore, access to these buffers is more restricted, and random access is not possible due to the strict time limitations. Similarly, the number of different queues is limited.

InfiniBand was proposed in 1999 by the most important IT companies to provide server systems with the required levels of reliability, availability, performance, scalability, and QoS [11]. Specifically, the InfiniBand Architecture (IBA)

1. The terms *packet switches* and *packet networks* will be used to refer to general networking technologies.

proposes three main mechanisms to provide the applications with QoS [12]. These are traffic segregation with service levels, the use of up to 16 VCs, and arbitration at output ports according to an arbitration table. Although IBA does not specify how these mechanisms should be used, some proposals have been made to provide applications with QoS in InfiniBand networks [13].

On the other hand, the PCI Express Advanced Switching (AS) architecture has recently been proposed as the natural evolution of the traditional PCI bus [14]. It is a switch fabric architecture that supports high availability, performance, reliability, and QoS. AS ports incorporate up to 20 VCs (16 unicast and 4 multicast) that are scheduled according to some QoS criteria. It is also possible to use a CAC implemented in the fabric management software.

These proposals, therefore, permit us to use several VCs to provide QoS support. However, implementing a great number of VCs would require a significant fraction of silicon area and would make packet processing slower. Moreover, there is a trend toward increasing the number of ports instead of increasing the number of VCs per port [15]. In general, the number of queues per port can have a significant effect on the overall complexity and cost of the interconnect switch. Therefore, it is important to attempt to provide effective QoS with a number of queues as small as possible. Indeed, our proposal addresses this very effectively.

3 Efficient Architecture for Per-Flow QoS Support

In a typical cluster environment, we have a set of hosts connected through the type of high-speed interconnects described above. The hosts have enough resources to maintain per-flow information for the traffic flows they originate. On the other hand, the switches that make up the interconnect have drastically fewer resources and usually cannot maintain more than a few different VCs. Unavoidably, the interconnect can only support aggregate QoS despite the host's ability to keep track of per-flow information. In this work, we explore whether it is possible to maintain reasonable traffic differentiation through a very simple interconnect that uses only two VCs. As discussed above, it is essential to keep the cost of interconnect switches as low as possible as even a few VCs may push the cost to unacceptable levels. The hosts implement a per-flow EDF type of scheduling discipline and all of the traffics are collapsed into the two VCs used by the interconnect. The interconnect switches perform a very simple sorting operation that can be implemented without the queue management complexity of the EDF scheduling. The effectiveness of the sorting operation depends on the fact that packets arrive from the hosts already sorted in priority. A previous version of this work can be found in [16].

We focus on the following TCs: 1) high-priority low-bandwidth control traffic, which is traffic that has to be delivered as fast as possible, 2) high-priority real-time traffic, which has to be delivered so that it does not violate its deadlines and, thus, requires a certain amount of bandwidth availability, and 3) best-effort low-priority traffic. We also want to provide further differentiation for different types of best-effort traffic. We do not aim at providing specific QoS guarantees such as delay of rate bounds, but we focus on providing sufficient traffic differentiation beyond the limitations of the two VCs in the interconnect. We will show how, thanks to the scheduling done at the hosts, we can effectively differentiate among multiple classes of traffic.

We want to efficiently adapt the EDF family of algorithms to a high-speed network. More specifically, we will use a variation of the Virtual Clock [17] algorithm. In our architecture, each packet will carry one tag, the deadline, which is the cycle in which it is supposed to be delivered to the final destination host. In order to compute this, the sender host is responsible for keeping some information about the flows with origin in that host.

A flow would be a single connection, like a TCP connection or traffic from a single application. Each flow would have the following parameters: source, destination, a fixed route, and the information necessary to compute deadlines, which would usually be average bandwidth, but may vary depending on the type of flow, as we will see later.

In addition to deadline, packets have another tag while they are at the sender host: the eligible cycle. This tag indicates the earliest cycle in which a packet is allowed to get into the network. Since it is not used in the switches, this tag is not transmitted in the packet header.

A cornerstone of our proposal is to avoid any bookkeeping of the flows at the switches. For scheduling, only the information in the header of packets is used: the deadline and the routing information.

We use an admission control similar to what is proposed for InfiniBand or PCI AS. Bandwidth reservation is performed at a centralized point and no record is kept in the switches. This makes the use of fixed routing mandatory so that packets use the route they have reserved.

On the other hand, we have to provide a connectionless or unregulated service like UDP or ATM's UBR for best-effort traffic. In this case, we still propose using fixed routing to avoid out-of-order delivery, which may happen with adaptive routing. Although we use fixed routing, admission control can ensure load balancing when assigning paths, as opposed to deterministic routing, where there is only a single path between a given pair of hosts.

For unregulated traffic, a generic flow record is kept in the end-hosts, with the necessary parameters. In this case, there is no bandwidth reservation and there is no guarantee of delivery. However, if we want to support several classes of best-effort traffic, we can configure several aggregated flows, each one with a different bandwidth to compute deadlines.

Summing up, we propose injecting packets from hosts using an unmodified EDF algorithm. However, at the switches, we use just two VCs, implemented as FIFO queues, and apply a scheduling based on the deadlines packets include in their header.

3.1 Calculus of Deadline

Taking into account the flow parameters, the packets are stamped in the end-hosts with the deadline tag. In addition, an eligible time tag is also used while the packet remains in the interface. For most flows, the deadline of packet P_i is

$$D(P_i) = maximum(D(P_{i-1}), T_{now}) + \frac{L(P_i)}{BW_{avg}},$$

where $L(P_i)$ is the length of the packet P_i, T_{now} is the host's clock when the packet arrives from the application level, and BW_{avg} is the reserved average bandwidth. This computation does not consider the number of hops that a packet needs to reach its destination. However, this is fine in high-performance networks, where base latency is very short.

Some specialized types of traffic require a different method to compute bandwidth. Control traffic needs a latency that is as short as possible but takes almost no bandwidth. For this type of traffic, we would use no connection admission and BW_{avg} would be the link bandwidth. In this way, control traffic gets the maximum priority.

Multimedia traffic usually consists of bursts of packets followed by silence periods. Let us assume that we want to transmit an MPEG video sequence with average bandwidth of 400 Kbyte/s. Moreover, we know that the video sequence consists of one video frame each 40 ms and the frame size can be between 1 and 120 Kbytes. For this kind of traffic, an average bandwidth assignation is not enough because, during peak-rate periods, it would introduce intolerable delays. We could use the maximum bandwidth (based on maximum frame size) to generate deadlines, but two problems arise: First, if the frame to be transmitted is short, we are introducing unnecessary bursts of packets. Second, the latency of each frame will vary a lot since it will depend on the size of frames.

We propose using the following strategy: The user fixes a desired latency per frame, for instance 10 ms. Upon reception of a new frame, we compute the number of network level packets it will generate. For instance, if the frame size is 80 Kbytes and the maximum transfer unit (MTU) is 2 Kbytes, it will generate 40 packets. For each packet P_i, the deadline is

$$D(P_i) = maximum(D(P_{i-1}), T_{now}) + \frac{10\text{ ms}}{Parts(F_i)},$$

where $Parts(F_i)$ is the number of packets generated by the frame to which P_i belongs. In this way, every frame will have a latency close to 10 ms, independently of frame size, and a smooth distribution of packets. This is good both for avoiding unnecessary bursts in the network and for preventing a lot of variability in frame latency.

Another central element of our proposal is that the deadline of the packets is not recomputed at the switches. The main reason is that the ideal implementation of a high-speed switch is a single chip to minimize delays, which means that silicon area is limited and there is no space for recording information regarding all of the flows traversing the switch. Moreover, recomputing the deadline would introduce additional delay and would not introduce any significant benefit in a high-performance network.

The use of eligible time in the end-nodes is optional since some TCs do not tolerate being smoothed. When it is used, typically for multimedia traffic, we propose computing the eligible time of a packet as its deadline minus a fixed value, the *eligibility factor*. We have found in our tests that 20 μs works well. In this way, when a traffic flow is smoothed, packets leave the end-node at most 20 μs before their deadline, not earlier (they can leave later due to competition for the link). This strategy, together with the aforementioned method to compute deadlines for multimedia traffic, produces almost constant latency for multimedia frames, which in turn reduces jitter and also produces low burstiness since packets are more evenly distributed.

3.2 Packet Scheduling

Ideally, each switch would schedule packets implementing an EDF algorithm. However, searching for the packet with the minimum deadline through all of the buffers is not practical. An alternative is to implement a heap buffer, which always keeps the packet with the lowest deadline at the top of the queue. A design for this is discussed in [18]. However, the associated cost is not practical for high-speed switches with high radix (number of ports).

On the other hand, we have observed that, when traffic is regulated (no oversubscription of the links), the switches can just take into account the first packet at each input buffer in arrival order. The idea is that traffic coming from the interfaces has already been scheduled and this traffic is coming in an ascending order of deadline. This being so, it is possible to just consider the first packet at each queue with confidence that packets coming afterward have higher deadlines.

The behavior of the switch would be analogous to a sorting algorithm: If the switch has as input ordered chains of packets and has to produce at the output an ordered sequence, it only needs to look at the first packet of each input.

The main limitation of this algorithm is that packets may not always come ordered from the interfaces. It may happen that, when no more low-deadline packets are available, a high-deadline packet is transmitted, especially if eligible time is not being used. If the high-deadline packet has to wait in a switch input queue and other packets with lower deadline are transmitted from the network interface, they would be stored after the high-deadline packet in the same queue. Thus, the arbiter would penalize the low-deadline packets because they would have to wait until the high-deadline packet is transmitted.

The order error situation, along with the eligibility time feature, is illustrated with an example in Fig. 1. Let us assume that, at a given host, there are two applications injecting traffic: a multimedia application and a control application. At a certain point, there are several multimedia packets waiting for injection and no control packet at the host. In $T = 4$, the first multimedia packet can proceed because the actual time, 4, plus the eligibility factor, 3, is less or equal than the packet's deadline, 7. In the same way, in $T = 9$, the next multimedia packet can proceed. However, in $T = 11$, a control packet is generated and immediately forwarded. In that case, there is an order error because the deadline of the control packet is smaller than the deadline of the preceding multimedia packet in the switch buffer. Also note that the maximum magnitude of order errors is the eligibility factor 3 in this example. In our simulations, we use 20 μs obtaining good performance.

Order errors will violate our assumptions and degrade the service offered to the low-deadline packets. We will analyze this problem and how to attenuate it later. Note that

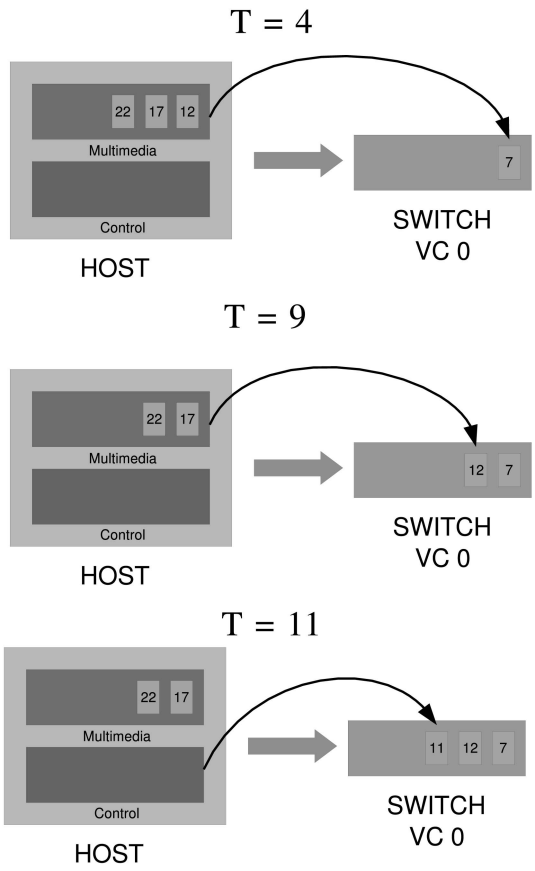

Fig. 1. Example of order error.

there is no chance for starvation since traffic is regulated and there is enough bandwidth guaranteed.

On the other hand, there is also unregulated (best-effort) traffic that could interfere with the regulated traffic. This is the reason why we propose using two different VCs: one for regulated traffic and the other for nonpoliced traffic. The regulated traffic has absolute priority over the best-effort traffic. Therefore, we can guarantee that regulated traffic will not be delayed by congestion and still accept best-effort traffic to make use of the remaining bandwidth.

There are also two VCs at the end-hosts, but packets are always in deadline order. In the regulated traffic VC, there are two queues, one feeding the other. In the first queue, packets are stored in ascending eligible time. As soon as the first packet in the queue is eligible, it goes to another queue, where packets are sorted according to ascending deadlines. Packets are injected from this queue as soon as the link is available and there are enough credits. On the other hand, packets in the best-effort VC are also sorted by deadline. They are injected only when the link is available, there are credits, and the regulated traffic VC has no packets ready to inject (there might be packets waiting for eligible time).

3.3 Clock Synchronization

Precise clock synchronization between the end-hosts would be needed for the viability of our proposal. If packets carry deadline tags in absolute values, then the performance would be affected by clock synchronization. Since clock synchronization protocols (like NTP [19]) would be implemented in the network with maximum priority, we can argue that clocks would be quite synchronized. Skew between clocks would be much less than 1 ms and, thus, this would be the maximum penalization.

However, we can avoid this situation with a simple strategy. The deadlines we are computing consist of a base time value, linked to a local clock, plus some additional time to reach the destination. By subtracting local clocks, we would have the time to reach the final destination (TTD). This value has the advantage of not needing any synchronization of clocks. The host indicates that a packet has to reach its destination before n milliseconds instead of the absolute time to do the same.

The problem is that this value would change every clock cycle, which is undesirable. However, we can reconstruct a packet's deadline by adding the local clock again. Therefore, our strategy would be the following: Packets receive a deadline in the hosts and are stored as usual. When a packet is about to leave a host or switch, the TTD is computed:

$$TTD_i = D_i - T_{local},$$

where T_{local} is the local clock at the host or switch the packet is leaving. When the packet arrives at the next hop, the deadline is reconstructed, adding to the TTD of the packet the new local clock. This deadline is used for scheduling locally and, when the packet is chosen to be transmitted, a new TTD is computed and put in the packet header.

Regarding eligibility time, when this feature is active, no smoothed packet can leave the interface with a TTD higher than the eligibility factor. For instance, in the simulations, multimedia packets have a maximum TTD of 20 μs. This also bounds the number of bits necessary to store the TTD at the packet header.

The only drawback of this proposal is that the CRC of the header would need to be recomputed at each hop. This is because a packet's TTD will be changing with each hop. Other fields of the header also change with each hop. Note that fast CRC circuits are already present in the network elements and this is not a restrictive requirement. For instance, Myrinet [20] requires that CRC be recomputed at the output links of network elements.

3.4 Reducing Order Errors

We have observed through simulation that the performance achieved by the previous proposal is similar to having full-ordered queues. However, the latency of the most demanding flows may be increased as much as 25 percent due to order errors. To attenuate this effect, we propose an improvement to this proposal.

The key idea consists of splitting, in the switches, the regulated traffic VC into two FIFO queues (Fig. 2). One of these queues is the *ordered queue* and the other is the *take-over queue*. When a packet arrives, its deadline is compared with the deadline of the packet in the last position of the *ordered queue*. If the new packet has a higher deadline, it is put in the *ordered queue*. If the deadline is smaller, the packet goes to the *take-over queue*.

The dequeuing algorithm is very simple: The packet chosen to be transmitted is always the one with the smallest deadline of both queue heads. In this way, we give

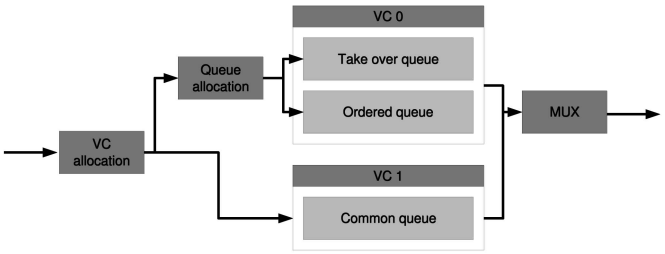

Fig. 2. New buffer structure for the switch ports.

low-deadline packets a chance to advance over packets with a high deadline. With this algorithm, the amount of order errors is not completely eliminated, but is greatly diminished, as we will see in the experiments.

This improvement to the buffer ports does not introduce out-of-order delivery. The demonstration can be found in the Appendix. Note that *out-of-order* delivery means that packets from a particular flow do not arrive in the same order in which they were injected. As opposed to when we talk about *order errors*, it means that packets from different flows are put in a queue out of order of deadline tags. Therefore, out-of-order delivery has to be avoided because it would require reordering buffers at the end-nodes, which is expensive. On the other hand, *order error* simply means that sometimes the scheduler will not choose the packet with the earliest deadline. In the latter case, some packets would be delayed, but no special hardware would be needed.

4 Simulation Conditions

In this section, we will explain the simulated network architecture. We will also give details on the parameters of the network and the load used for the evaluation.

4.1 Simulated Architecture

The network used to test the proposals is a butterfly multistage interconnection network (MIN) with 128 end-points. The actual topology is a folded (bidirectional) perfect-shuffle. We have chosen a MIN because it is a common topology for clusters. Although not included here, we have also tested direct networks (torus and meshes), obtaining similar performance. The switches use a combined input and output buffer architecture, with a crossbar to connect the buffers. We use virtual output queuing at the switch level, which is the usual solution to avoid head-of-line blocking.

We will evaluate five different architectures:

- A traditional switch architecture with some QoS support. This is based on the PCI AS specification and provides two VCs to distinguish between two broad traffic categories. It also has CAC, but no traffic smoothing. This architecture will be denoted in the figures as *Traditional 2 VCs*.
- A more advanced switch architecture, also based in PCI AS specification. In this case, there is a VC per TC (four in our study). Also, we have implemented Token Buckets for multimedia and best-effort traffic in order to alleviate the problems with bursty traffic. This architecture will be denoted in the figures as *Traditional 4 VCs*.
- An ideal switch architecture based on our EDF algorithm. This implements two VCs (regulated and unregulated traffic), but each one is a heap queue which always keeps the packet with the highest deadline at the top. In the figures, it is called *Ideal*. Order errors would not happen in this case but the implementation of this architecture would be unfeasible due to the kind of buffers.
- A simple switch architecture based on our first proposal. This emulates the previous ideal architecture, but, since order errors are possible, performance will be degraded. This will be called *Simple 2 VCs*.
- The improved version of the previous architecture. This implements the take-over queue proposed in Section 3.4. In the figures, it will appear as *Advanced 2 VCs*.

In all of the cases, the switches implement 16 ports and 8 Kbytes of buffer per VC. In our tests, the link bandwidth is 8 Gbps. The remaining parameter values are picked from the AS specification. In general, all of the parameters used in the simulations are quite typical for high-speed interconnects.

The CAC we have implemented for QoS-requiring traffic is a simple one, based on average bandwidth requirements. Each connection is assigned a path where enough bandwidth is assured. The CAC guarantees that less than 70 percent of bandwidth is used by QoS traffic at any link. The other 30 percent of available bandwidth will be used by unregulated traffic. We also use a load-balancing mechanism when a QoS connection is established, which consists of assigning the least occupied route among the possible paths.

4.2 Traffic Model

Table 1 presents the characteristics of the traffic injected in the network. We follow the recommendations of The Network Processing Forum Switch Fabric Benchmark Specifications [21]. We have considered a mix of QoS-requiring traffic flows and best-effort flows. In this way, the workload is composed of two different TCs: two QoS TCs and two best-effort TCs. Note that there are many individual flows of the four categories, each one with the characteristics shown in the table.

Control traffic models traffic from applications that demand a latency as short as possible. Since it has a connectionless nature, it is not subject to the CAC or load balancing.

Multimedia traffic consists of actual MPEG video sequences, transmitted through the network. Best-effort TCs, *Best-effort* and *Background*, are not subject to CAC or smoothing. For this reason, they can saturate network links and impact performance. These TCs are modeled with self-similar traffic, which is composed of bursts of packets heading to the same destination. The packet size is governed by a Pareto distribution, as recommended in [22].

5 Simulation Results

In this section, the performance of our proposals is shown. We have considered three traditional QoS indices for this performance evaluation: throughput, latency, and jitter.

TABLE 1
Traffic Injected per Host

Name	% BW	Application frame	Notes
Control	25	[128 bytes, 2 Kbytes]	Small control messages
Multimedia	25	[1 Kbyte, 120 Kbytes]	3 Mbyte/s MPEG-4 traces
Best-effort	25	[128 bytes, 100 Kbytes]	Self-similar internet-like traffic
Background	25	[128 bytes, 100 Kbytes]	Self-similar low-priority traffic

Note that packet loss is not considered because no packets are dropped due to the use of credit-based flow control. We also show the cumulative distribution function (CDF) of latency, which represents the probability of a packet achieving a latency equal to or lower than a certain value.

In the following, we will see two experiments. First, we evaluate an initial scenario where the input QoS load is equal to the best-effort load. Afterward, we will study which amount of QoS traffic can be allowed at each architecture.

5.1 Initial Scenario

In Fig. 3, we can see the performance of the most delay-sensitive TC we are considering, i.e., *Control* traffic. The most important result is that the EDF-based architectures (*Ideal*, *Simple 2 VCs*, and *Advanced 2 VCs*) offer much better results in terms of latency. The CDF results are obtained at an input load of 100 percent.

The best results correspond to the *Traditional 4 VCs* case. This is because, in this case, *Control* has its own VC and maximum priority. On the other hand, EDF architectures also have good performance. We can see at the bottom of the figure that our simplest proposal, *Simple 2 VCs*, increases maximum latency up to 15 percent compared with the *Ideal* case. On the other hand, the *Advanced 2 VCs* proposal has almost the same behavior as the *Ideal* case. Finally, the *Traditional 2 VCs* case has very bad performance.

Fig. 4 shows the performance of *Multimedia* traffic. Using the method we propose to compute deadlines, the average

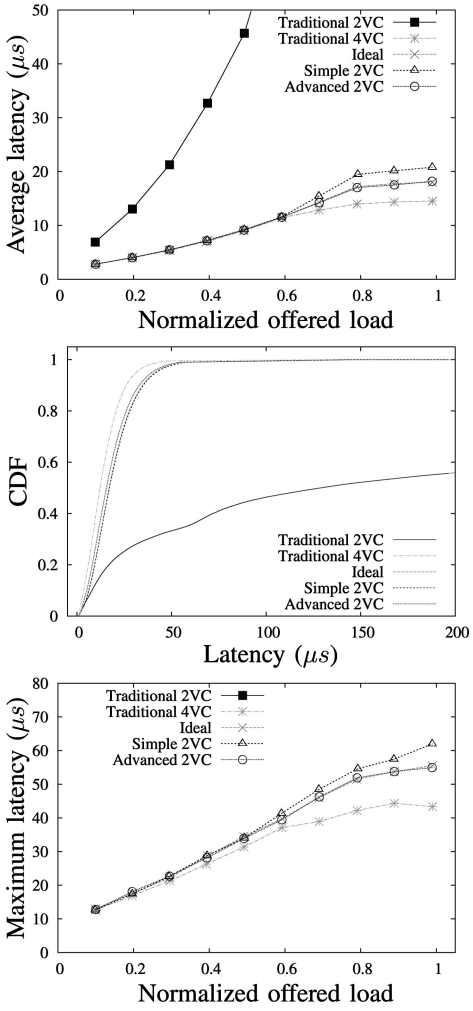

Fig. 3. Latency of control traffic.

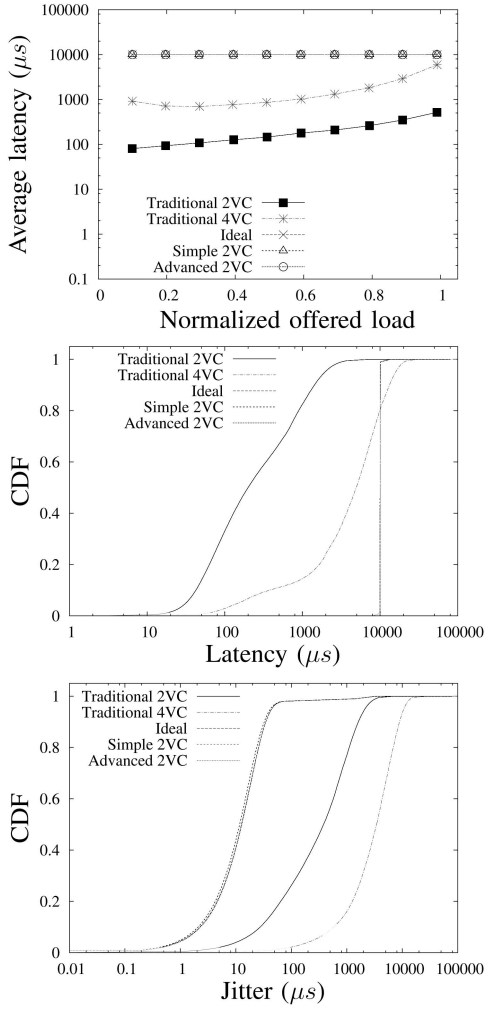

Fig. 4. Latency video traffic.

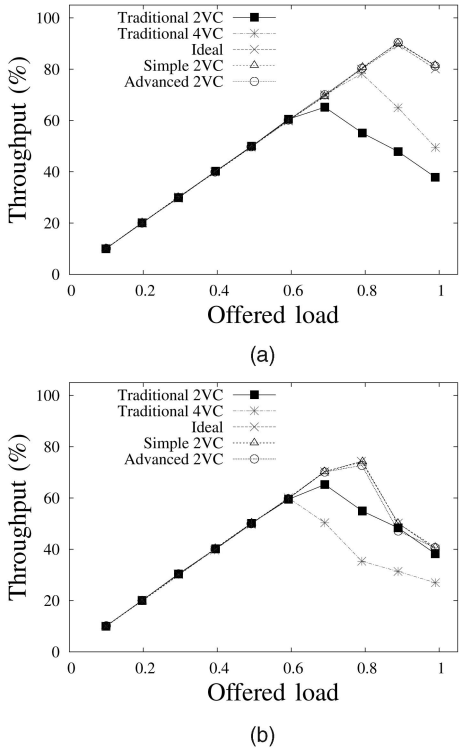

Fig. 5. Throughput of best-effort TCs. (a) Best-effort. (b) Background.

latency of video frames is almost exactly 10 ms (the value configured as desirable latency). Note that latency results refers to full transfers and not to individual packets (i.e., latency is for each frame of the video sequence). Looking at the CDF of latency, we notice that there is little variation in latency for EDF-based architectures (the probability of a latency of 10 ms is more than 99 percent). On the other hand, latency can vary considerably when using traditional architectures, which would introduce a lot of jitters, as can be seen at the bottom of the figure. Note that the *Traditional 4 VCs* case increases latency compared with the *Traditional 2 VCs* case due to the use of token buckets.

Finally, we show in Fig. 5 the performance in terms of throughput of the two best-effort classes we have considered. For the *Traditional 2 VCs* case, both classes look the same (they share VC 1) and, thus, receive the same performance. On the other hand, the EDF-based architectures can label each packet with deadlines according to the reserved bandwidth of each flow. In this way, not only can we differentiate multiple classes within a single VC, but we can also guarantee minimum bandwidth if we are careful in assigning weights to the different best-effort flows. Finally, the *Traditional 4 VCs* case is able to offer differentiation, but the achieved throughput is not as good as with the EDF architectures. The reason is that, although token buckets are being used, they have to be configured to allow peak rate of video transfers and, therefore, they alleviate but do not completely eliminate the burstiness of *Video* traffic.

We can conclude that EDF-based architectures are clearly superior to a traditional two classes QoS. Note that the cost of these architectures is similar, except for the *Ideal* architecture. Using our proposals, the only difference is a slight increase of latency for *Control* traffic compared with the *Traditional 4 VCs* architecture.

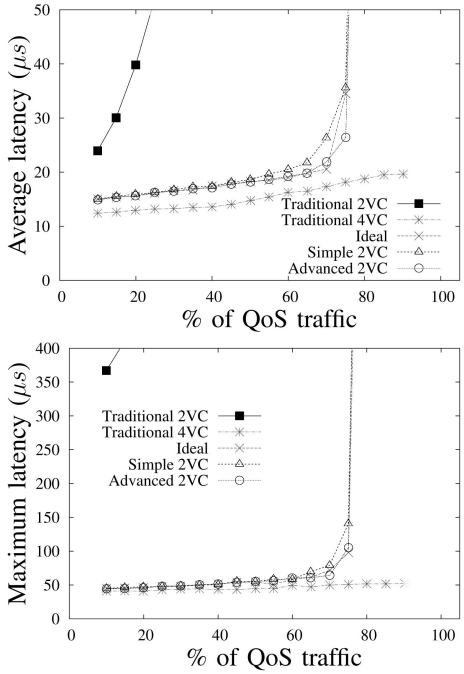

Fig. 6. Latency of control traffic varying QoS load.

5.2 QoS Traffic Acceptance

In this section, we evaluate which amount of QoS traffic can be accepted by each architecture before QoS requirements are not satisfied. Moreover, we can see the behavior of the three EDF variants we are considering under stressing QoS load.

In this scenario, we vary the proportion of QoS traffic, from 10 percent to 90 percent of the total available network load. We fill in the remaining bandwidth with best-effort traffic. Therefore, input links are injecting at 100 percent of their capacity. We can see that the different TCs saturate at different points when using the five architectures. In this way, QoS requirements are satisfied only up to a certain QoS load.

In Fig. 6, we can see *Control* latency results. For the *Traditional 2 VCs* case, results are very bad for almost any QoS load. On the other hand, the *Traditional 4 VCs* case offers good performance at any amount of QoS traffic because *Control* traffic has its own VC and maximum priority. Finally, the three EDF architectures, including our two proposals, have good performance up to a QoS load of 75 percent. Note that the *Advanced 2 VCs* case reduces the latency of this TC up to 20 percent compared with the *Simple 2 VCs* case.

Regarding *Video* traffic, we see in Fig. 7 that latency is 10 ms for every video frame using the three EDF architectures up to a load of 75 percent. On the other hand, both traditional architectures start losing[2] *Video* packets at a QoS load of 60 percent.

Finally, we can observe in Fig. 8 the throughput of the two best-effort TCs we have considered. In this case, the EDF architectures offer the best performance for *Best-effort*. For *Background* traffic, results are similar in all of the cases.

2. Packets are discarded at the end-nodes after waiting 100 ms for injection.

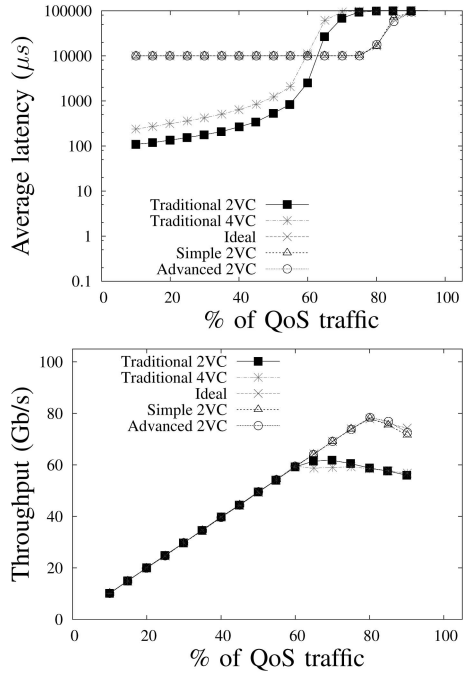

Fig. 7. Latency and throughput of video traffic varying QoS load.

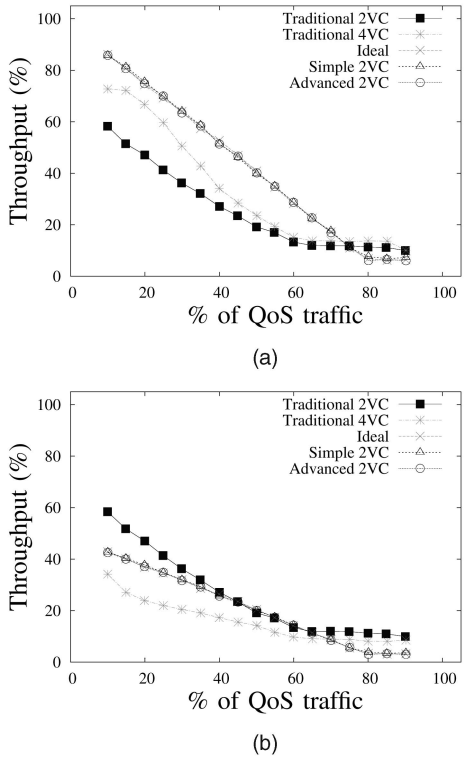

Fig. 8. Throughput of best-effort TCs varying QoS load. (a) Best-effort. (b) Background.

When using the EDF algorithms, more best-effort traffic can be injected in the gaps left by QoS traffic.

Summing up, EDF architectures provide very good performance for all TCs up to 75 percent of QoS traffic (100 percent network load). On the other hand, traditional architectures are only able to handle up to 60 percent of QoS load with similar results. Our proposals emulate very well the behavior of an ideal architecture, even with a lot of QoS traffic.

6 Conclusions

In this paper, we have explored how to efficiently adapt the EDF family of algorithms to high-speed network environments. As far as we know, no similar attempt has been made since some adaptations of ATM due to the cost of using ordered buffers. On the other hand, our proposal is an architecture which, using FIFO buffers, offers almost the same performance, even for the most delay-sensitive traffic.

We have also compared our proposals with typical VC-based QoS, such as can be found in recent standards such as InfiniBand or PCI AS. We have seen that, for similar cost in terms of the silicon area, our proposals offer a much better performance. Achieving similar performance in a traditional multi-VC architecture would require a larger number of VCs, which would significantly increase the cost of the switches and would limit the applicability to high-speed interconnects.

Appendix

In Section 3.4, we have introduced a two-queue system which models the high-priority VC of an input or output buffer in the switches. Now, we are going to prove that this system for the QoS VC does not introduce out-of-order delivery. Note that this is an important issue: In many high-speed standards, out-of-order delivery is explicitly forbidden (for instance, in PCI AS). For that purpose, first, we indicate the notation in Table 2, introduce several initial hypotheses, and present the enqueuing and dequeuing algorithms. Finally, we prove some theorems:

Initial hypotheses: Two conditions are accomplished by every flow:

$$D(F_k^j) < D(F_{k+1}^j)\ 1 \le j \le n_F,\ 1 \le k < np_j, \tag{1}$$

$$I(F_k^j) < I(F_{k+1}^j)\ 1 \le j \le n_F,\ 1 \le k < np_j. \tag{2}$$

Intuitively, the previous expressions say that packets from a flow arrive ordered at the system and they have increasing deadlines.

Now, we will formally define the enqueuing and dequeuing algorithms:

Definition 1 (Enqueuing algorithm). *Enqueuing of an incoming packet p works as follows:*

- *If both queues are empty, store p in the L queue.*
- *If there are m_L packets in the L queue:*
  - *If $D(p) \ge D(L_{m_L})$ store p in the L queue.*
  - *Else, store p in the U queue.*

Note that incoming packets always have space in the system due to the credits flow control. Also note that the two queues can dynamically take all of the memory allowed for the VC and, therefore, it is not possible for a queue to become full while there is space in the other queue.

With respect to the flow control mechanism, the following possibility could arise: If two packets are available and $D(U_1) < D(L_1)$ but L_1 is smaller (in bytes)

TABLE 2
Notation

U	Upper queue (Take over queue)
L	Lower queue (Ordered queue)
U_i	Packet at position i in the U queue. ($i = 1$, packet at the front of the queue)
L_i	Packet at position i in the L queue. ($i = 1$, packet at the front of the queue)
m_U	Number of packets in the U queue at a given moment
m_L	Number of packets in the L queue at a given moment
$D(p)$	Deadline function. $D(L_i)$ is the deadline of the ith packet of the L queue
n_F	Number of packet flows
$F^1, ..., F^{n_F}$	Packet flows
np_j	Number of packets of the jth flow
$F^j_{np_j}, ..., F^j_1$	Individual packets of the flow F^j in the generation order and, thus, in arrival order. F^j_1 is the packet at the first position
$Dep(p)$	Time at which packet p leaves the system
$I(p)$	Arrival time of the packet p

than $D(U_1)$ and there are only credits for L_1, the latter would be forwarded out of order. This would corrupt the dequeuing discipline, but we prevent this by imposing the condition that only the packet with the smallest deadline of the potential two available is checked for credits and, thus, for transmission.

Definition 2 (Dequeuing algorithm). *The algorithm for removing packets works as follows:*

- *If both queues are empty, there is no packet to choose.*
- *If there are packets only in the L queue, L_1 is chosen.*
- *If there are packets in both queues, the packet with the smallest deadline between $D(L_1)$ and $D(U_1)$ is chosen.*
- *A situation where there are only packets in the U queue is not possible (Lemma 1).*

Lemma 1. *A situation where there are only packets in the U queue is not possible.*

Proof. The empty L queue and the U queue with packets cannot be obtained from the two empty queues since the enqueuing algorithm indicates that, if the two queues are empty and a packet arrives, the latter is stored in the L queue.

Hence, an empty L and U with packets could only arise from a situation in which both queues have packets and dequeuing takes place in both. However, from Theorem 2, the packet with the highest deadline is in the L queue and, thus, all of the packets in the two queues will leave before the former and U will become empty before L. □

Definition 3 (Out-of-order delivery). *As we mentioned earlier, there would be out-of-order delivery if packets from an individual flow were to leave the system in a different order from arrival order. Therefore, there is out-of-order delivery iff*

$$\exists j, k \,/\, Dep(F^j_k) > Dep(F^j_{k+1})$$
$$1 \le j \le n_F, \quad 1 \le k < np_j.$$

In the following, we are going to prove several theorems, some of them more or less intuitive, which will permit us to prove that, given the previous enqueuing and dequeuing algorithms, out-of-order delivery is not possible in our proposed queue system.

Theorem 1. *Packets in the L queue are in deadline order:*

$$D(L_i) \le D(L_{i+1}) \qquad 1 \le i < m_L.$$

Proof. By reductio ad absurdum: If packets in the L queue are not in deadline order,

$$\exists i \,/\, D(L_{i+1}) < D(L_i) \qquad 1 \le i < m_L,$$

but that would contradict the enqueuing algorithm, which only stores a packet in the L queue when its deadline is higher than or equal to that of the last packet in the queue. Since packets can only leave the queue in a FIFO discipline, this order is preserved by the dequeuing algorithm. □

Theorem 2. *The packet with the highest deadline in the two queues is always the last packet in the L queue:*

$$D(L_i) \le D(L_{m_L}) \qquad 1 \le i < m_L,$$
$$D(U_j) < D(L_{m_L}) \qquad 1 \le j < m_U,$$

where m_L is the number of elements in the L queue.

Proof. The first part of the theorem follows from Theorem 1.

On the other hand, packets in the U queue always have a smaller deadline than the last element of the L queue. Following the enqueuing algorithm, in the event of a new packet arriving with a larger deadline than the maximum, it would be stored in the last position of the L queue and would become the new maximum deadline.

Finally, L_{m_L} is always the last element to leave the system. No packet L_i, $i \neq m_L$, in the L queue can leave earlier due to the FIFO discipline. On the other hand, since all of the packets in the U queue have a smaller deadline than L_{m_L} (as we have proven in the previous paragraph), they cannot leave earlier than it with the

dequeuing algorithm given, which always chooses the packet with the minimum deadline. □

Theorem 3. *There is no out-of-order delivery. Formally,*

$$Dep(F_k^j) < Dep(F_{k+1}^j) 1 \leq j \leq n_F, 1 \leq k < np_j.$$

Proof. Since the arrival of packets is ordered, conflicts can only arise if F_{k+1}^j manages to overcome F_k^j while it is still waiting at the system. That means that we have to study the cases where both packets are stored in the queues. Let us analyze the different possible cases:

- When they arrive, both F_k^j and F_{k+1}^j go to the same queue, either L or U. In this case, they leave in arrival order because both the U and L queues are FIFO queues. Since they arrived ordered (2), they leave ordered.
- Upon arrival, F_k^j goes to the L queue and, later, F_{k+1}^j goes to the U queue. This may happen if $D(F_k^j)$ is the maximum deadline at the arrival time, but before the arrival of F_{k+1}^j at least one packet p arrives with $D(p) > D(F_{k+1}^j)$.

  From Theorem 1, we know that the L queue is ordered and, when F_{k+1}^j is ready to leave, it means that its deadline is smaller than that of any packet in the L queue. Since $D(F_k^j) < D(F_{k+1}^j)$ (1), it is sure that packet F_k^j already left and the order is preserved.
- When they arrive, F_k^j goes to the U queue and, later, F_{k+1}^j goes to the L queue. It may happen if $D(F_k^j)$ is smaller than the maximum but $D(F_{k+1}^j)$ is larger.

  Let L_{m_L} be the last packet in the L queue when F_k^j is stored in the U queue. From Theorem 2, L_{m_L} has a higher deadline than any packet in the U queue at that moment, including F_k^j. Therefore, it will leave later than all of those packets: To leave earlier, it would need to be compared with a packet from the U queue with a higher deadline, but this is not possible. As a consequence, it is true that

$$Dep(F_k^j) < Dep(L_{m_L}).$$

  Since F_{k+1}^j is positioned in the L queue behind L_{m_L} (maybe with other packets between), it cannot leave earlier (FIFO queuing) and, therefore, it has to leave after F_k^j:

$$Dep(L_{m_L}) < Dep(F_{k+1}^j),$$

$$Dep(F_k^j) < Dep(L_{m_L}) < Dep(F_{k+1}^j)$$

$$\Rightarrow Dep(F_k^j) < Dep(F_{k+1}^j).$$

□

ACKNOWLEDGMENTS

This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under Grants Consolider Ingenio-2010 CSD2006-00046 and TIN2006-15516-C04-0X, and by Junta de Comunidades de Castilla-La Mancha under Grant PBC08-0078-9856. This research was performed while Alejandro Martí was working at the University of Castilla-La Mancha.

REFERENCES

[1] S. Reinemo, T. Skeie, T. Sodring, O. Lysne, and O. Trudbakken, "An Overview of QoS Capabilities in InfiniBand, Advanced Switching Interconnect, and Ethernet," *IEEE Comm. Magazine,* vol. 44, no. 7, pp. 32-38, July 2006.

[2] G. Rodgers and P. Morjan, "Blade Cluster Architecture," *IBM Systems Group—Barcelona Supercomputing Center,* technical report, http://www.bsc.es/publications/documentation/pdf/Greg Rodgers_Presentation.pdf, Sept. 2005.

[3] A. Forum, *ATM Forum Traffic Management Specification,* version 4.0, May 1995.

[4] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. ACM Symp. Comm. Architectures and Protocols,* pp. 1-12, http://portal.acm.org/citation.cfm?id=75248, 1989.

[5] A.K. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Trans. Networking,* vol. 1, no. 3, pp. 344-357, 1993.

[6] R. Guerin and V. Peris, "Quality-of-Service in Packet Networks: Basic Mechanisms and Directions," *Computer Networks,* vol. 31, no. 3, pp. 169-189, 1999.

[7] L. Georgiadis, R. Guerin, and A.K. Parekh, "Optimal Multiplexing on a Single Link: Delay and Buffer Requirements," *Proc. IEEE INFOCOM '94,* vol. 2, pp. 524-532, 1994.

[8] S. Floyd and V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks," *IEEE/ACM Trans. Networking,* vol. 3, no. 4, pp. 365-386, citeseer.ist.psu.edu/floyd93linksharing.html, 1995.

[9] R. Braden, D. Clark, and S. Shenker, *Integrated Services in the Internet Architecture: An Overview,* Internet Eng. Task Force, Internet Request for Comment RFC 1633, http://www.ietf.org/rfc/rfc1633.txt, June 1994.

[10] S. Blake, D. Back, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Services,* Internet Eng. Task Force, Internet Request for Comment RFC 2475, http://www.ietf.org/rfc/rfc2275.txt, Dec. 1998.

[11] *InfiniBand Architecture Specification,* vol. 1, Release 1.0, InfiniBand Trade Assoc., Oct. 2000.

[12] J. Pelissier, "Providing Quality of Service over InfiniBand Architecture Fabrics," *Proc. Eighth Symp. High-Performance Interconnects,* http://www.hoti.org/hoti8_thursday.html, Aug. 2000.

[13] F.J. Alfaro, J.L. Sánchez, and J. Duato, "QoS in InfiniBand Subnetworks," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 9, pp. 810-823, Sept. 2004.

[14] ASI SIG, *Advanced Switching Core Architecture Specification,* 2005.

[15] C. Minkenberg, F. Abel, M. Gusat, R.P. Luijten, and W. Denzel, "Current Issues in Packet Switch Design," *ACM SIGCOMM Computer Comm. Rev.,* vol. 33, pp. 119-124, Jan. 2003.

[16] A. Martínez, F. Alfaro, J. Sánchez, and J. Duato, "Deadline-Based QoS Algorithms for High-Performance Networks," *Proc. 21st Int'l Parallel and Distributed Processing Symp.,* http://investigacion.uclm.es/portali/documentos/fi_1169052300-IPDPS07.pdf, Mar. 2007.

[17] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks," *Computer Comm. Rev. (Proc. ACM SIGCOMM '90),* vol. 20, no. 4, pp. 19-29, Sept. 1990.

[18] A. Ioannou and M. Katevenis, "Pipelined Heap (Priority Queue) Management for Advanced Scheduling in High Speed Networks," *Proc. IEEE Int'l Conf. Comm.,* 2001.

[19] D.L. Mills, *RFC 958: Network Time Protocol (NTP),* Sept. 1985.

[20] N. Boden, D. Cohen, and R. Felderman, "Myrinet—A Gigabit per Second Local Area Network," *IEEE Micro,* pp. 29-36, Feb. 1995.

[21] I. Elhanany, D. Chiou, V. Tabatabaee, R. Noro, and A. Poursepanj, "The Network Processing Forum Switch Fabric Benchmark Specifications: An Overview," *IEEE Network,* pp. 5-9, Mar. 2005.

[22] R. Jain, *The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, 1991.

Alejandro Martínez received the MS degree in computer science and the PhD degree from the University of Castilla-La Mancha in 2003 and 2007, respectively. He is currently with the Intel Barcelona Research Center. His research interests include high-performance interconnections, quality of service, high-performance computing, and processor microarchitecture.

George Apostolopoulos received the BSc degree in computer engineering from the University of Patras, Greece, and the MSc and PhD degrees in computer science in 1994 and 1999, respectively. He has been a researcher with the IBM T.J. Watson Research Center and at ICS-FORTH, Greece. He is currently a principal engineer with Redback/Ericsson. His interests include routing, QoS, scalable and robust implementation of software for networking systems, system support for router software, and router architectures. He is a member of the IEEE.

Francisco J. Alfaro received the MS degree in computer science from the University of Murcia in 1995 and the PhD degree from the University of Castilla-La Mancha in 2003. He is currently an assistant professor of computer architecture and technology in the Department of Computer Systems at the University of Castilla-La Mancha. His research interests include high-performance local area networks, quality of service, design of high-performance routers, and design of on-chip interconnection networks for multicore systems.

José L. Sánchez received the PhD degree from the Technical University of Valencia, Spain, in 1998. Since November 1986, he has been with the Department of Computer Systems (formerly the Computer Science Department) at the University of Castilla-La Mancha, where he is currently an associate professor of computer architecture and technology. His research interests include multiprocessor architectures, quality of service in high-speed networks, interconnection networks, and parallel algorithms.

José Duato is a professor in the Department of Computer Engineering (DISCA) at the Universidad Politécnica de Valencia, Spain. His research interests include interconnection networks and multiprocessor architectures. He has published more than 350 papers. His research results have been used in the design of the Alpha 21364 microprocessor, and the Cray T3E and IBM BlueGene/L supercomputers. He is the first author of the book *Interconnection Networks: An Engineering Approach*. He has served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He is currently an associate editor of *IEEE Computer Architecture Letters*. He was the general cochair of ICPP '01, the program chair of HPCA-10, and a program cochair of ICPP '05. He has also been a cochair, a steering committee member, a vice chair, and a program committee member for more than 55 conferences, including HPCA, ISCA, IPPS/SPDP, IPDPS, ICPP, ICDCS, Europar, and HiPC. He is a member of the IEEE.