Deadline-based QoS Algorithms for High-performance Networks*

Alejandro Martínez[1], Francisco J. Alfaro[1], José L. Sánchez[1], José Duato[2]

[1]DSI - Univ. of Castilla-La Mancha
02071 - Albacete, Spain
{alejandro, falfaro, jsanchez}@dsi.uclm.es

[2]DISCA - Tech. Univ. of Valencia
46071 - Valencia, Spain
jduato@disca.upv.es

Abstract

Quality of service (QoS) is becoming an attractive feature for high-performance networks and parallel machines because it could allow a more efficient use of resources. Deadline-based algorithms can provide powerful QoS provision. However, the cost associated with keeping ordered lists of packets makes them impractical for high-performance networks. In this paper, we explore how to adapt efficiently the Earliest Deadline First family of algorithms to the high-speed networks environments. The results show excellent performance using just two virtual channels, FIFO queues, and a cost feasible with today's technology.

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 is 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. Besides, to provide the different classes of traffic with their requirements, it is common to settle a network for each traffic class. For instance, the recently-built supercomputer MareNostrum [12] 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 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 traffic classes (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 put 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, perhaps with the exception of ATM [6].

In this paper, we discuss how to obtain most of the benefits of the per-flow QoS 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 a pair of FIFO queues.

The remainder of this paper is structured as follows. In the following section 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.

*This work was partly supported by the Spanish CICYT under CSD2006-46 and TIN2006-15516-C04-02 grants, by Junta de Comunidades de Castilla-La Mancha under grant PBC-05-005, and by the Spanish State Secretariat of Education and Universities under FPU grant.
1-4244-0910-1/07/$20.00 ©2007 IEEE.

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.

2.1 QoS support in packet networks

Over the last few years there has been extensive work on how to schedule resources of a packet switch* to provide guaranteed performance to traffic. The switch resources that need to be scheduled are buffer space (usually at the outgoing port) and link bandwidth. Both resources are managed through a *service discipline*. Typically, packet switch buffers are fairly large and support random access. When buffers become full, packets are dropped. Thus, general packet switches can introduce packet loss when their resources are oversubscribed. Performance guarantees usually include bounds on packet loss, delay, jitter, and transmission rate or throughput. A large number of service disciplines have been proposed (see [7] for an overview), each one specifically targeted for providing certain types of guarantees.

The service disciplines operate at the flow level and consequently can provide different QoS guarantees to individual flows. Examples of such flow oriented QoS architectures are the QoS architecture of ATM [6] and the Integrated-Services model [4] that was proposed for Internet in the mid-90s. Since such per-flow scheduling can prove a bottleneck as the number of flows grows, aggregate-QoS architectures have been proposed where QoS is provided collectively to all flows belonging to a certain class of service. In these environments, there are only a few such classes of service, but flows now get only aggregate and not individual QoS. An example of such a QoS architecture is Differentiated Services [3], which is nowadays used to provide limited QoS in parts of the Internet.

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. Firstly, flow control is commonly used to throttle the incoming traffic and, thus, usually there are 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 [8]. Specifically, the InfiniBand Architecture (IBA) proposes three main mechanisms to provide the applications with QoS. These are traffic segregation with service levels, the use of up to 16 VCs and the 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 [1].

On the other hand, the PCI Express Advanced Switching (AS) architecture has been recently proposed as the natural evolution of the traditional PCI bus [2]. 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. Is is also possible to use a connection admission control 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 [11]. In general, the number of queues per port can have a significant effect on the overall complexity and cost of the interconnect switch. 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

We want to adapt efficiently the Earliest Deadline First family of algorithms to a high-speed network. More specifically, we will use a variation of the Virtual Clock [13] 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 to keep 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 usually would 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 indicates

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

the earliest cycle in which a packet is allowed to get into the network and it is not used in the switches. Therefore, it is not transmitted in the 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 to use fixed routing to avoid out-of-order delivery, which may happen with adaptive routing. Although we use fixed routing, the 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.

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, 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 packet comes 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 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 in bursts of packets followed by silence periods. Let us assume that we want to transmit a MPEG sequence. The average bandwidth is 400 Kbyte/s. Moreover, we know that the video sequence consists in one video frame each 40 milliseconds and the frame size can be between 1 and 120 Kbytes. 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. Secondly, the latency of each frame will vary a lot, since it will depend on the size of frames.

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

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

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 milliseconds, independently of frame size, and a smooth distribution of packets.

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. That means that silicon area is limited and there is no space for recording information regarding all the flows traversing the switch. Moreover, recomputing the deadline would introduce additional delay.

The use of eligible time is optional, since some traffic classes do not tolerate being smoothed. When it is used, typically for multimedia traffic, we propose to compute eligible time of a packet as its deadline minus a fixed amount. We have found that 20 microseconds works well in our tests. In this way, we eliminate the bursts of packets that appear when packets are injected as soon as they are available.

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 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 [9]. 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 over-subscription 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 ascending order of deadline. This being so, it is possible to just consider the first packet at each queue, in the confidence that packets coming afterwards 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. This 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 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 to use two different virtual channels: One for regulated traffic and the other for non-policed 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.

The organization of end-hosts also uses the structure of two VCs. 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. However, we can avoid synchronization with a simple strategy. The deadlines we are computing consist in a base time value, linked to 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 header.

The only drawback of this proposal is that the CCR of the header would be needed to be recomputed at each hop. This is because a packet's TTD will be changing with each hop. However, other fields of the header also change with each hop, for instance the pointer to the next hop in PCI AS's source routing, and so, recomputing of the CCR of the header would be necessary anyway.

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% due to order errors. To attenuate this effect, we propose an improvement to this proposal.

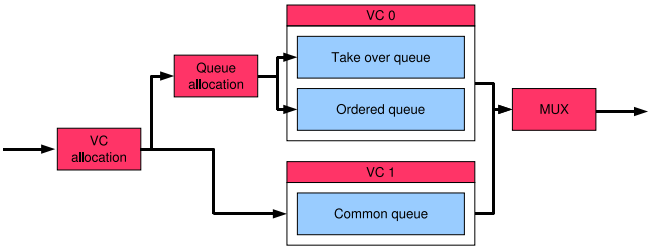


Figure 1. New buffer structure.

The key idea consists in splitting the regulated traffic VC into two FIFO queues (Figure 1). One of these queues is the

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 internet-like traffic

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 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 greatly diminished: the observed increase in average latency is now only 5%.

This improvement to the buffer ports does not introduce out-of-order delivery, the demonstration can be found in an appendix at the end of the paper. 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, 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 re-order buffers at the end-nodes, which are 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 multi-stage 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. The switches use a combined input and output buffer architecture, with a crossbar to connect the buffers. We use virtual output queuing (VOQ) at the switch level, which is the usual solution to avoid head-of-line blocking.

We will evaluate four 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. More VCs are possible with this specification, but they are very unlikely in a final implementation. This architecture will be denoted in the figures as *Traditional 2 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 at the top the packet with the highest deadline. 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 buffers.

- A simple switch architecture based on our first proposal. This emulates the previous, 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 the cases, the switches implement 16 ports and 8 Kbytes of buffer per VC. In our tests, the link bandwidth is 8 Gb/s. The remaining parameter values are picked from the AS specification. In general, all the parameters used in the simulation are quite typical for high-speed interconnects.

4.2 Traffic model

Table 1 presents the characteristics of the traffic injected in the network. We have considered a mix of QoS-requiring traffic flows and best-effort flows. In this way, the workload is composed of 2 different TCs: two QoS TCs and two best-effort TCs.

We follow the recommendations of The Network Processing Forum Switch Fabric Benchmark Specifications [5]. *Control* traffic models traffic from applications that demand a latency as short as possible. *Multimedia* traffic consists

in actual MPEG video sequences, transmitted through the network. The self-similar traffic is composed of bursts of packets heading to the same destination. The packet size is governed by a Pareto distribution, as recommended in [10].

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. 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.

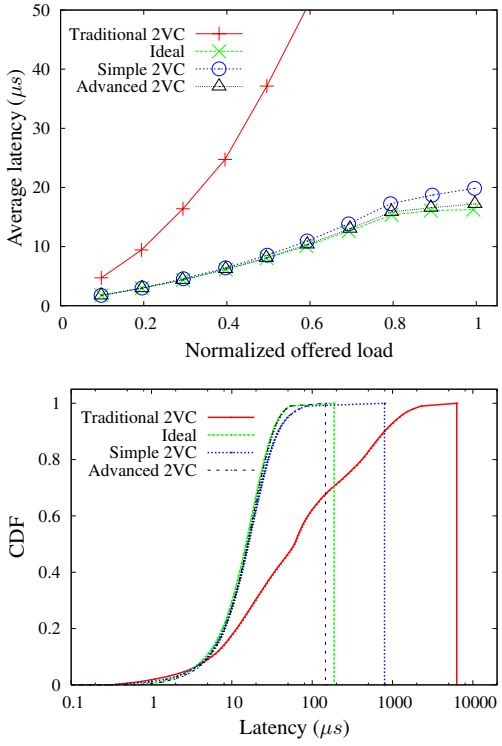


Figure 2. Results for Control traffic.

In Figure 2, we can see the performance of the delay-sensitive traffic class we are considering, *Control* traffic. The most important result is that the EDF-based architectures offer much better results in terms of latency. The CDF results are obtained at an input load of 100%.

Obviously, the best results correspond to the *Ideal* architecture. Our simplest proposal, *Simple 2 VCs*, has an average increase of around 25% in average latency. On the other hand, when using our *Advanced 2 VCs* proposal, the difference is only 5%. It is also remarkable that maximum latency values (the closing vertical line in the CDF figure) are almost the same for *Ideal* and *Advanced 2 VCs* cases.

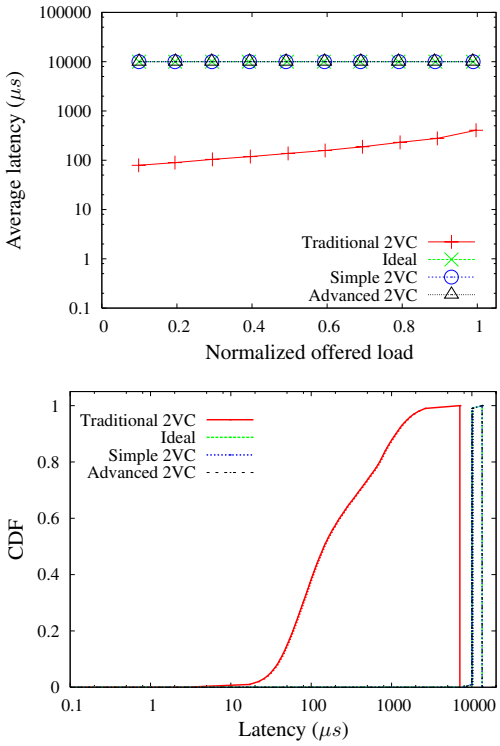
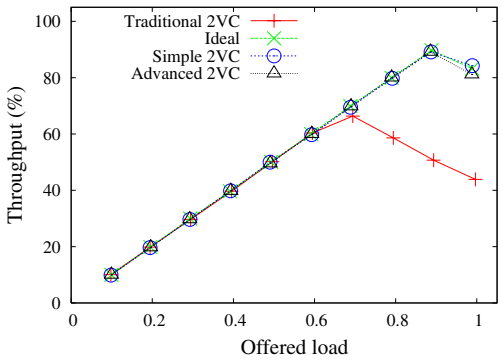


Figure 3. Results for Video traffic.

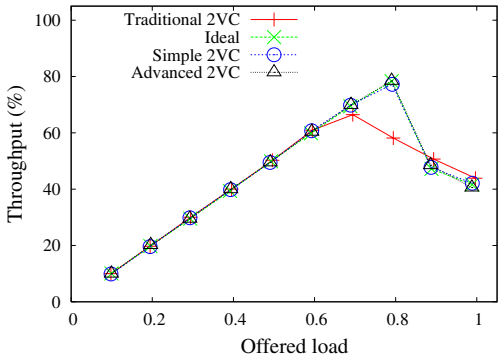
Figure 3 shows the performance of *Multimedia* traffic. Using the method we proposed to compute deadlines, the average latency of video frames is almost exactly 10 milliseconds (the value configured as desirable latency). Note that latency results refer to full transfers and not to individual packets (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 milliseconds is more than 99%). On the other hand, latency can vary considerably when using *Traditional 2 VCs*, which would introduce a lot of jitter (not shown due to lack of space).

Finally, we show in Figure 4 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 (traffic for 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 guarantee minimum bandwidth if we are careful assigning weights to the different best-effort flows.

We can conclude that EDF-based architectures are clearly superior to a traditional two classes QoS. Note that



(a) Best-effort



(b) Background

Figure 4. Results for best-effort traffic classes.

the cost of these architectures is similar, except the *Ideal* architecture. Using our proposals, the only difference is an increase of 5% average latency for *Control* traffic.

6 Conclusions

In this paper, we explore how to adapt efficiently the Earliest Deadline First family of algorithms to the high-speed networks 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 also compare our proposals with typical VC-based QoS, as can be found in recent standards such as InfiniBand or PCI AS. We see that, for similar cost in terms of the silicon area, our proposals offer a much better performance. In order to achieve something similar, it would be necessary to implement many more VCs, but because this is not affordable almost no final implementation includes them.

References

[1] F. J. Alfaro, J. L. Sánchez, and J. Duato. QoS in InfiniBand subnetworks. *IEEE Transactions on Parallel Distributed Systems*, 15(9):810–823, Sept. 2004.

[2] ASI SIG. *Advanced switching core architecture specification*, 2005.

[3] S. Blake, D. Back, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. Internet Request for Comment RFC 2475, Internet Engineering Task Force, Dec. 1998.

[4] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. Internet Request for Comment RFC 1633, Internet Engineering Task Force, June 1994.

[5] I. Elhanany, D. Chiou, V. Tabatabaee, R. Noro, and A. Poursepanj. The network processing forum switch fabric benchmark specifications: An overview. *IEEE Network*, pages 5–9, Mar. 2005.

[6] A. Forum. *ATM Forum traffic management specification. Version 4.0*, May 1995.

[7] R. Guerin and V. Peris. Quality-of-service in packet networks: basic mechanisms and directions. *Comput. Networks*, 31(3):169–189, 1999.

[8] InfiniBand Trade Association. *InfiniBand architecture specification volume 1. Release 1.0*, Oct. 2000.

[9] A. Ioannou and M. Katevenis. Pipelined heap (priority queue) management for advanced scheduling in high speed networks. In *Proceedings of the IEEE International Conference on Communications (ICC'2001)*, 2001.

[10] R. Jain. *The art of computer system performance analysis: techniques for experimental design, measurement, simulation and modeling*. John Wiley and Sons, Inc., 1991.

[11] C. Minkenberg, F. Abel, M. Gusat, R. P. Luijten, and W. Denzel. Current issues in packet switch design. In *ACM SIGCOMM Computer Communication Review*, volume 33, pages 119–124, Jan. 2003.

[12] G. Rodgers and P. Morjan. Blade cluster architecture. Technical report, IBM Systems Group - Barcelona Supercomputing Center, Sept. 2005.

[13] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *In Proceedings ACM SIGCOMM '90*, pages 19–29, Sept. 1990. Published as Special Issue of Computer Communication Review, vol. 20, n. 4.

Appendix

In section 3.4 we have introduced a two-queue system which models the high priority VC of an input or output buffer. 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, firstly we indicate the notation in Table 2, introduce several initial hypotheses, and present the enqueuing and dequeuing algorithms. Finally, we prove some theorems.

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

Initial hypotheses Two conditions are accomplished by every flow:

$$D(F^j_k) < D(F^j_{k+1}) \quad 1 \leq j \leq n_F\ , \quad 1 \leq k < np_j \quad (1)$$

$$I(F^j_k) < I(F^j_{k+1}) \quad 1 \leq j \leq n_F\ , \quad 1 \leq k < np_j \quad (2)$$

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 $D(p) \geq D(L_{m_L})$ 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 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 possibility could arise that if two packets are available and $D(U_1) < D(L_1)$ but L_1 is smaller (in bytes) 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 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*

$$\exists\, j,k \,/\, Dep(F_k^j) > Dep(F_{k+1}^j)\ 1 \le j \le n_F\ ,\ 1 \le k < np_j$$

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 the packets in the U queue have a smaller deadline than L_{m_L} (as we have proved 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,*

$$Dep(F_k^j) < Dep(F_{k+1}^j) \qquad 1 \le j \le n_F\ ,\quad 1 \le k < np_j$$

Proof: Since 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's 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 U and L queues are FIFO queues. Since they arrived ordered (Equation 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 of 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 $D(F_k^j) < D(F_{k+1}^j)$ (Equation 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 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. In 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:

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

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

□