Providing QoS with the Deficit Table Scheduler

Raúl Martínez-Moráis, Francisco J. Alfaro-Cortés, and José L. Sánchez

Abstract—A key component for networks with Quality of Service (QoS) support is the egress link scheduling algorithm. An ideal scheduling algorithm implemented in a high-performance network with QoS support should satisfy two main properties: good end-to-end delay and implementation simplicity. Table-based schedulers try to offer a simple implementation and good latency bounds. Some of the latest proposals of network technologies, like Advanced Switching and InfiniBand, include in their specifications one of these schedulers. However, these table-based schedulers do not work properly with variable packet sizes, as is usually the case in current network technologies. We have proposed a new table-based scheduler, which we have called Deficit Table (DTable) scheduler, that works properly with variable packet sizes. Moreover, we have proposed a methodology to configure this table-based scheduler in such a way that it permits us to decouple the bounding between the bandwidth and latency assignments. In this paper, we thoroughly review the provision of QoS with the DTable scheduler and our configuration methodology, and evaluate the performance of our proposals in a multimedia scenario. Simulation results show that our proposals are able to provide a similar latency performance than more complex scheduling algorithms. Moreover, we show the advantages of our decoupling configuration methodology over the usual ways of configuring this kind of table-based schedulers.

Index Terms—Quality of Service (QoS), scheduling algorithms, table-based schedulers, latency requirements, interconnection networks, performance evaluation.

1 Introduction

The evolution of interconnection network technology has been constant along the previous decades. The speed and capacity of various components in a communication system, such as links, switches, memory, and processors, have increased dramatically. Moreover, network topologies have become more flexible, and the efficiency of switching, routing, and flow control techniques have been improved.

The advent of high-speed networking has introduced opportunities for new applications. Current packet networks are required to carry not only traffic of applications, such as e-mail or file transfer, which does not require prespecified service guarantees, but also traffic of other applications that requires different performance guarantees, like real-time video or telecommunications [23]. The best effort service model, though suitable for the first type of applications, is not so for applications of the other type [25]. Even in the same application, different kinds of traffic (e.g., I/O requests, coherence control messages, synchronization and communication messages, etc.) can be considered, and it would be very interesting that they were treated according to their priority [5].

These are the reasons because the provision of QoS in computing and communication environments has been the focus of much discussion and research in academia during the last decades. This interest in academia has been renewed by the growing interest on this topic in industry during the last years. A sign of this growing interest in industry is the inclusion of mechanisms intended to provide QoS in some of the last network standards like Gigabit Ethernet [32], InfiniBand (IBA) [13], or Advanced Switching (AS) [1]. An interesting survey with the QoS capabilities of these network technologies can be found in [30].

A key component for networks with QoS support is the output (or egress link) scheduling algorithm (also called service discipline) [8], [11], [40]. In a packet-switching network, packets from different flows[1] will interact with each other at each switch. Without proper control, these interactions may adversely affect the network performance experienced by clients. The scheduling algorithm, which selects the next packet to be transmitted and decides when it should be transmitted, determines how packets from different flows interact with each other. Therefore, the scheduling algorithm plays an important role in providing the traffic differentiation that is necessary to provide QoS.

Apart from providing a good performance in terms of, for example, good end-to-end delay (also called latency) and fair bandwidth allocation, an ideal scheduling algorithm implemented in a high-performance network with QoS support should satisfy other important property which is to have a low computational and implementation complexity [34]. This is because in order to achieve a good performance, the time required to select the next packet to be transmitted must be smaller than the average packet transmission time. This means that the scheduler computation time must be very small, if we consider the high speed of high-performance networks. Moreover, a low complexity is required in order to be able to implement the scheduler in a small silicon area

- *R. Martínez-Moráis is with the Intel-UPC Barcelona Research Center, Campus Nord, Building NXII (Nexus II), C. Jordi Girona, 29, Barcelona 08034, Spain. E-mail: raulmm@dsi.uclm.es.*
- *F.J. Alfaro-Cortés and J.L. Sánchez are with the Computer Systems Department, University of Castilla-La Mancha, Escuela Superior de Ingeniría Informática, Campus Universitario s/n, Albacete 02071, Spain. E-mail: {falfaro, jsanchez}@dsi.uclm.es.*

Manuscript received 21 May 2008; revised 2 April 2009; accepted 9 April 2009; published online 22 April 2009.
Recommended for acceptance by S. Rangarajan.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-05-0199.
Digital Object Identifier no. 10.1109/TPDS.2009.75.

1. In this paper, we will use the term *flow* to refer both to a single flow or to an aggregated of several flows with similar characteristics.

(note that high-performance switches are usually implemented in a single chip). Note that the scenario that we are addressing here is very different than, for example, IP routers with QoS support, where these algorithms are usually implemented by software instead of hardware.

The design of a traffic scheduling algorithm involves an inevitable trade-off among the above properties. Many scheduling algorithms have been proposed for that. Among them, the "sorted-priority" family of algorithms is known to offer very good delay [37]. However, their computational complexity is very high, making their implementation in high-speed networks rather difficult. Table-based schedulers are intended to provide a good latency performance with a low computational complexity. This approach is followed in [4] and in two of the last high-performance interconnection network proposals: AS [1] and IBA [13]. However, as we will see, these schedulers do not work properly with variable packet sizes, as is usually the case in current network technologies.

In [20], we proposed a new table-based scheduler that works properly with variable packet sizes. Moreover, we proposed a methodology to configure this scheduler in such a way that it permits us to decouple partially the bounding between the bandwidth and latency assignments. We called this new scheduler Deficit Table scheduler, or just DTable scheduler.

In this paper, we review this new scheduling algorithm and largely expand the analysis of its behavior and the discussion on the way of providing QoS. Furthermore, we present a whole framework to provide QoS in any network technology that implements a DTable scheduler as the egress link scheduling algorithm. This framework supports the coexistence of traffic with explicit requirements based on bandwidth and latency at the same time that allows several levels of best effort traffic. Finally, in Section 6, we not only compare the throughput and latency performance of this scheduler with the one provided by the well-known Self-Clocked Weighted Fair Queuing (SCFQ) scheduler [10], as an example of a "sorted-priority" algorithm, and the Deficit Round Robin (DRR) scheduler [33], because of its simplicity, but also we compare our decoupling configuration methodology with the one proposed in [4]. For this performance evaluation, we have assumed a multimedia scenario using the IEEE standard 802.1D-2004 [12] traffic types.

The structure of the paper is as follows: Section 2 presents a summary of the best known scheduling algorithms and introduces the table-based schedulers. In Section 3, we present the DTable scheduling mechanism. In Section 4, we present our proposal to decouple the bandwidth and latency assignments. Section 5 presents our proposal to provide QoS requirements with the DTable scheduler and its decoupling configuration methodology. Details on the experimental platform and the performance evaluation are presented in Section 6. Finally, some conclusions are given and future work is proposed.

2 Frame-Based Schedulers

Scheduling disciplines can be categorized in many ways. Traditionally, they have been divided into work-conserving and non-work-conserving disciplines [40]. Another possible classification is based on their internal structure, according to which there are two main architectures: Sorted priority and frame based [36].

Sorted-priority scheduling disciplines, like Weighted Fair Queuing (WFQ) [8], packet-by-packet Generalized Processor Sharing (GPS) [24], Self-Clock Fair Queuing SCFQ [10], and Worst Case Weighted Fair Queuing (WF2Q) [3], use a global variable, often called virtual time (to distinguish it from real time), associated with the server. The purpose of this variable is to keep track of the progress of the server and it is usually updated at packet arrival and departure instants. For each packet in the system, a time stamp is computed as a function of this variable. Packets are then sorted based on these time stamps and served in this order. They differ in the manner in which they calculate the global virtual time function. They generally provide good fairness and a low latency but are not very efficient due to the complexity involved in computing the virtual time and the complexity of maintaining a sorted list of packets based in their time stamps.

On the other hand, frame-based scheduling disciplines use a frame of fixed or variable length which is divided among different connections/classes based on the reservations of the connections/resources allocated for the class. The more resources are allocated for a connection/class, the larger part of the frame it receives. The frame is split among the connections/classes in a similar way in each service round. The simplest examples of this category of schedulers are the round-robin schemes. They have $O(1)$ per-packet work complexity, but are well known for their output burstiness and short-term unfairness. Weighted Round Robin (WRR), DRR [33], Elastic Round Robin (ERR) [15], and Carry-Over Round Robin (CORR) [31] are typical round-robin schedulers. In these kinds of round-robin schedulers, the schedulers will serve a flow for a continuous period of time in proportion to the weight of the flow, resulting in a highly burst scheduling output for each flow. The latency and fairness of these algorithms depend on the frame length. The longer the frame is, the higher the latency and the worse the fairness. In order for DRR to exhibit lower latency and better fairness, the frame length should, therefore, be kept as small as possible. Unfortunately, given a set of flows, it is not possible to select the frame length arbitrarily.

Therefore, these kinds of round-robin schedulers are considered not suitable to provide QoS in packet networks and thus, other frame-based scheduling algorithms have been proposed in order to overcome the latency and burstiness problem of round-robin schemes, while maintaining a low complexity. Examples of these approaches are the Smoothed Round Robin (SRR) [7], the Nested Deficit Round Robin (Nested-DRR) [16], and the List-based WRR [4] scheduling algorithms.

SRR codes the weights of the flows into binary vectors to form a Weight Matrix, then uses a Weight Spread Sequence, which is specially designed to distribute the output more evenly, to schedule packets by scanning the Weight Matrix. The traversal of these two structures results in a sequence of slots assigned to each Virtual Channel (VC) that continually repeats. This sequence tries to emulate the GPS. The Nested-DRR scheduler has the same objective as the SRR, overcoming the burstiness problem of the DRR scheduler.

In this case, the scheduler modifies the frame-based DRR scheduler, by creating a nested set of multiple frames inside each DRR frame. The basic principle of the Nested-DRR is to split each round in DRR, henceforth called an outer round, into one or more smaller inner rounds, and then execute a version of the DRR algorithm over these inner rounds. The Preorder Deficit Round Robin (Preorder DRR) [38], as stated in [17], can be interpreted as a Nested-DRR with a limited and fixed number of outer rounds.

The List-based WRR algorithm can be considered a generalization of the two previous approaches and other similar algorithms. In this generalization of the classical WRR discipline, instead of serving packets of a flow in a single visit per frame, the service is distributed throughout the entire frame. For this, a list of flow identifiers, called "service list," is maintained. When scheduling is needed, the list is cycled through sequentially and a packet is transmitted from the flow indicated by the current entry. The number of times that a flow identifier appears in the service list is proportional to its weight, but these appearances are not necessarily consecutive as in the classical WRR algorithm. Note that, the list-based WRR, as the original WRR, is intended for environments with fixed packet size.

In [4], three ways of distributing the flow identifiers to conform the service list are proposed: Simply Interleaved WRR, Uniformly Interleaved WRR, and WF2Q Interleaved WRR. These three possible ways of distributing the flow identifiers result in three different schedulers. In order to compute any service list for a list-based WRR scheduler, let be w_i the integer weight assigned to each flow i, J the number of flows, and $N = \sum_{i=1}^{J} w_i$ the number of entries of the service list. The three possibilities presented in [4] are:

1. **Simply Interleaved WRR.** In order to compute the service list of this approach, which is the simplest one, we divide the service list in $W_{max} = max\{w_i\}_{i=1}^{J}$ sets of entries (bins). Session with weight w_i registers itself in the first w_i bins. Each bin will have at maximum one entry assigned to any given flow. A service list is then computed by listing all the sessions in the first bin, followed by all those in the second bin, and so on, up to the W_{max}th bin. Note that this distribution of the service list entries is the same that would be obtained with the Nested-DRR algorithm.
2. **Uniformly Interleaved WRR.** In this approach, the number of bins equals the least common multiple (denoted by W_{LCM}) of $\{w_i\}_{i=1}^{J}$. Session i registers itself in every $(n \times (W_{LCM}/w_i))$th bin for $1 \leq n \leq w_i$. A service list is then computed, by listing the sessions bin after bin. Note that this distribution of the service list entries is the same that would be obtained with the SRR.
3. **WF2Q Interleaved WRR.** In this approach, the service list is computed by assuming that all sessions are always backlogged and determining the sequence in which the packets are transmitted in the WF2Q scheme. The service list is then set equal to this sequence.

Note that, in all the cases the proportion of entries associated with each flow indicates the bandwidth assigned to each flow. Therefore, the difference between the three schedulers is in the way of distributing the flow identifiers among the entries. These different forms of interleaving the flow identifiers result in different latency characteristics for the three schedulers. All the approaches are able to improve the performance of the classical WRR. However, the WF2Q Interleaved WRR approach offers the best properties while the Simply Interleaved WRR offers the worst ones. Two of the last high-performance network interconnection proposals: IBA [13] and AS [1] define in their specifications table-based scheduling mechanisms that can be used to implement any of the three list-based WRR approaches.

InfiniBand uses VCs to aggregate flows with similar characteristics and the arbitration is made at a VC level. The maximum number of unicast VCs that a port can implement is 16. InfiniBand defines a scheduler that uses two tables, one for scheduling packets from high-priority VCs and another for low-priority VCs. The maximum amount of data that can be transmitted from high-priority VCs before transmitting a packet from the low-priority VCs can be configured. Each table has up to 64 entries. Each entry contains a VC identifier and a weight, which is the number of units of 64 bytes to be transmitted from that VC. This weight must be in the range of 0 to 255, and is always rounded up as a whole packet. When arbitration is needed, the table is cycled through sequentially, and a certain number of packets is transmitted from the VC indicated by the VC identifier depending on the entry weight.

The AS table-based scheduler employs an arbitration table that consists in a register array with fixed-size entries of 8 bits. Each entry contains a field of 5 bits with a VC identifier value and a reserved field of 3 bits. When arbitration is needed, the table is cycled through sequentially and a packet is transmitted from the VC indicated in the current table entry regardless of the packet size. If the current entry points to an empty VC, that entry is skipped. The number of entries may be 32, 64, 128, 256, 512, or 1,024.

Other interesting algorithm is Stratified Round Robin [28]. The objective of this scheduling mechanism is to manage a high number of individual flows with a feasible complexity in an Internet high-speed router. In order to do so, those flows with a similar bandwidth assignation are grouped into the same flow class. Given this, the algorithm proposes a two-step scheduler. While the first step uses a deadline mechanism similar to many time-stamp schedulers, the second step is essentially a round-robin scheme. Therefore, Stratified Round Robin may be considered a hybrid between sorted-priority and frame-based packet schedulers. This scheduling algorithm has the advantage of reducing the complexity of managing a large set of flows to the complexity of a sorted-priority scheduler and several DRR schedulers that manage a much reduced set of queues each one. However, the management of a single queue per flow is not feasible in the switches of high-performance networks, which are usually implemented in a single chip. In fact, real high-performance technologies with QoS support only consider a reduced number of VCs. With this, what we state is that the Stratified Round Robin is intended for a quite different environment than the DTable scheduler that is presented in this paper. Even so, the DTable

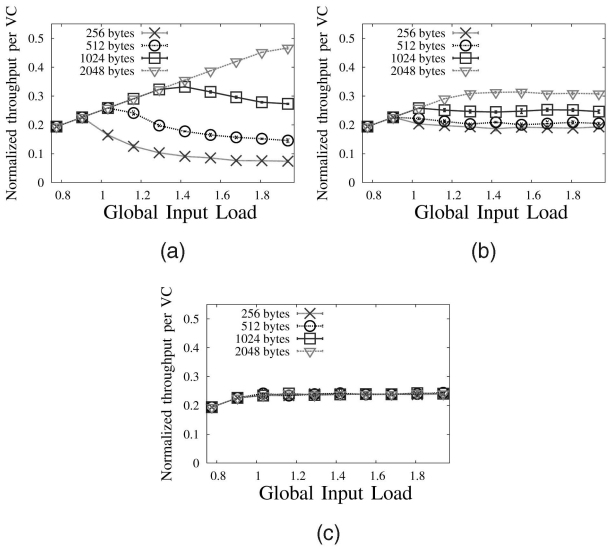

Fig. 1. Performance of several table-based schedulers for flows with different packet size. (a) Basic table (AS). (b) Weighted table (IBA). (c) Deficit table (DTable).

scheduler could be used to implement the interclass scheduling mechanism in a simpler manner.

As a final remark, we must say that, except the Stratified Round Robin that addresses other issues, the cited frame-based scheduling algorithms are conceptually more complex than a table-based scheduler, like that proposed in AS and IBA, but at the end, what they obtain is a distribution of slots among the VCs that conforms a scheduling frame that can be emulated with these table-based schedulers. Therefore, table-based schedulers are able to obtain at least the same latency performance, if not better when emulating the WF2Q algorithm, than any of the previous frame-based schedulers. However, as we will see, both the AS and IBA table-based schedulers do not work well in networks with variable packet sizes. Moreover, in all these scheduling algorithms, the latency provided to the different flows is completely dependent on the bandwidth assigned to each flow, which is the main problem that we address in this paper with the DTable and our decoupling configuration methodology.

3 The DTable Scheduling Mechanism

The main problem of the table-based schedulers mentioned in the previous section is that they do not work in a proper way with variable packet sizes, as is common in actual traffic. If the average packet size of the flows is different, the bandwidth that the flows obtain may not be proportional to the number of table entries [20].

Fig. 1 shows the performance of various table-based schedulers when there are four VCs in the network. Note that we use VCs to aggregate flows with similar characteristics performing the arbitration at a VC level, as it is the case in AS and IBA technologies. The four VCs have the same number of assigned table entries (the same bandwidth reservation). Moreover, we inject an increasing amount of traffic at the same rate in all the VCs. However, the traffic injected in each VC has a different packet size. Note that in the figures we refer to each VC according to the packet size that the flows associated to that VC use. The simulated architecture is the same as that used for the performance evaluation in Section 6.

Fig. 1a shows the case of a basic table scheduler similar to the AS table scheduler, which is cycled through and when a table entry is selected, a packet from the VC indicated in that entry is transmitted regardless of the packet size. As can be observed, when using the basic table scheduler, the VCs obtain a very different bandwidth because the traffic that traverses each VC has a different packet size. Therefore, although the same number of packets from each flow will be transmitted, the amount of information will not be the same.

The IBA's arbitration table works in a similar way than the AS table. However, it adds a weight to each entry. This weight indicates the amount of information to be transmitted from the VC associated to the table entry each time that the entry is selected. This weighted table solves the problem only partially because it allows a packet to be transmitted that requires even more weight than the remainder of a given table entry (exhausting them). Fig. 1b shows the performance of a weighted table that works in this way. We have assigned all the entries the same weight: 2,176 bytes (34 units of 64 bytes). As can be seen, it presents a better performance than the basic table scheduler, but not an optimum performance.

In [20], we proposed a new table-based scheduling algorithm that works properly with variable packet sizes (as can be seen in Fig. 1c). We called this algorithm Deficit Table scheduler, or just DTable scheduler, because it is a mix between the already proposed table-based schedulers and the DRR algorithm. The DRR algorithm [33] is a variation of the WRR algorithm that works on a proper way with variable packet sizes. In order to handle properly variable packet sizes, the DRR algorithm associates each queue with a *quantum* and a *deficit counter*. The quantum assigned to a queue is proportional to the bandwidth assigned to that queue. The deficit counter is set to 0 at the beginning. The scheduler visits sequentially each queue. For each queue, the scheduler transmits as many packets as the quantum allows. When a packet is transmitted, the quantum is reduced by the packet size. The unused quantum is saved in the deficit counter, representing the amount of quantum that the scheduler owes the queue. At the next round, the scheduler will add the previously saved quantum to the current quantum. When the queue has no packets to transmit, the quantum is discarded, since the flow has wasted its opportunity to transmit packets. Our scheduler works in a similar way than the DRR algorithm but instead of serving packets of a flow in a single visit per frame, the quantum associated to each flow is distributed throughout the entire frame.

This new table-based scheduler defines an arbitration table in which each table entry has associated a flow identifier and an *entry weight*, which is usually expressed in flow control credits in networks with a credit-based link-level flow control (like AS and IBA). Moreover, each flow has assigned a *deficit counter* that is set to 0 at the beginning.

When scheduling is needed, the table is cycled through sequentially until an entry assigned to an active flow is

```
while (There is at least one active flow)
        if ((selectedFlow is not active) or (selectedFlow_sizeFirst > accumulatedWeight))
                deficitCounter_selectedFlow ← accumulatedWeight
                tableEntry ← Next table entry assigned to an active flow
                selectedFlow ← tableEntry.flowIdentifier
                accumulatedWeight ← deficitCounter_selectedFlow + tableEntry.weight
        endif
        accumulatedWeight = accumulatedWeight − selectedFlow_sizeFirst
        Transmit packet from selectedFlow
        if ((There are no packets in the queue of selectedFlow) or
                (The flow control does not allow transmitting from selectedFlow))
                accumulatedWeight ← 0

        endif
endwhile
```

Fig. 2. Pseudocode of the DTable scheduler.

found. A flow is considered active when it stores at least one packet and the flow control allows that flow to transmit packets. When a table entry is selected, the *accumulated weight* is computed. The accumulated weight is equal to the sum of the deficit counter for the selected flow and the current entry weight. The scheduler transmits as many packets from the active flow as the accumulated weight allows. When a packet is transmitted, the accumulated weight is reduced by the packet size.

The next active table entry is selected if the flow becomes inactive or the accumulated weight becomes smaller than the size of the packet at the head of the queue. In the first case, the remaining accumulated weight is discarded and the deficit counter is set to zero. In the second case, the unused accumulated weight is saved in the deficit counter, representing the weight that the scheduler owes the queue.

This behavior is represented in the pseudocode shown in Fig. 2. Note that when using the scheduling algorithm, the bandwidth assigned to the ith flow ϕ_i with an arbitration table of N entries is

$$\phi_i = \frac{\sum_{j=0}^{J} weight_j}{\sum_{n=0}^{N} weight_n},$$

where J is the set of table entries assigned to the ith flow and $weight$ is the entry weight assigned to a table entry.

3.1 Complexity Considerations

As stated before, the complexity of a sorted-priority algorithm is determined by the complexity of calculating the time stamp, updating the priority list and selecting the highest priority packet for transmission. The complexity of time-stamp calculation is dependent on the specific scheduling discipline. For example, in WFQ, the updating of virtual time is considerably more complex than in the SCFQ. Moreover, a common problem in the sorted-priority approach is that tags cannot be reinitialized to zero until the system is completely empty and all the sessions are idle. The reason is that these tags depend on a common-reference virtual clock and are an increasing function of the time. In other words, it is impossible to reinitialize the virtual clock during the busy period, which, although statistically finite (if the traffic is constrained), can be extremely long, especially given that most communication traffic exhibits self-similar patterns which lead to heavily tailed buffer occupancy distributions. Therefore, for a practical implementation of sorted-priority algorithms, very high-speed hardware is required to perform the sorting, and floating-point units must be involved in the computation of the time tags.

On the other hand, the complexity of the DRR algorithm is quite small. According to the implementation proposed in [33], DRR exhibits $O(1)$ complexity provided that each flow is allocated a quantum no smaller than the Maximum Transfer Unit (MTU). As observed in [18], removing this hypothesis would entail operating at a complexity which can be as large as $O(N)$. Note that this restriction affects not only the weight assigned to the smallest flow, but to the rest of the flows in order to keep the proportions between them. Each time a packet is transmitted, the algorithm must compute if more packets from the same flow can be transmitted or it must change to the next active flow. This computation can be performed with simple integer units.

In the case of the DTable scheduler, in order to keep the computational complexity low, we set the minimum value that a table entry can have associated to the MTU of the network. In that case, this is also the smallest value that ensures that it will never be necessary to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. This means that each time an entry from an active flow is selected, at least one packet is going to be transmitted from that flow. Note that, this consideration is also made in the DRR algorithm definition. However, in [22], we proposed to use different MTUs for the different flows. This means that each flow has a specific MTU equal to or lower than the general MTU of the network and that we can assign each table entry a minimum weight equal to the specific MTU of the flow associated with that table entry. We can assign each flow a specific MTU by hardware or at the communication library

TABLE 1
Complexity Comparison Summary

Algorithm	Tag calculation	Tag shorting	Tag overflow	Computation units	Other considerations
WFQ	High	Yes	Yes	+, /, comparison	tag per packet
SCFQ	Low	Yes	Yes	+, /, comparison	tag per packet
DRR	-	No	No	+, -	-
DTable	-	No	No	+, -	Arbitration table searching process

level. Note that in IBA this issue is solved by rounding up to a whole packet the remaining weight in a table entry.

In the case of the DTable scheduler, a list of active VCs would not be as simple to maintain as in the DRR case, because VCs must be visited not in a sequential way but in the order indicated by the table scheduler. Therefore, in this case, the table must be looked over searching for the next active entry and skipping those entries that refer to a VC without packets or credits to transmit. Although the checking of each entry can be made with very simple computational units, in the worst case all the table must be looked over in order to find the next active entry. This kind of mechanism probably requires very little silicon area to be implemented, but may last too much time. In order to make the process faster several entries of the table can be read simultaneously at the expense of increasing the silicon area requirements. The size of the table effectively affects the number of VCs that we can accommodate with an appropriate degree of flexibility and of course has a big impact on the area requirements of the scheduling mechanism. Therefore, a trade-off must be made among the number of VCs supported and the size dedicated to the egress link scheduler when implementing this algorithm.

Note that, apart from the higher flexibility, the DTable has the advantage over the Smoothed Round Robin of not having to traverse the positions of the weight matrix that are equal to zero because, in the DTable, each entry that must be traversed has assigned a VC. On the other hand, it could be interesting to study an approach similar to the one followed in the SRR scheduler that identifies repeated patrons on the DTable in order to reduce the number of table entries required by the scheduler.

Summing up, the DTable algorithm has not the problem of the increasing tag value and does not need complex floating point units like in the case of the "sorted-priority" algorithms. However, the management of the arbitration table is more complex than the management of the DRR quantums. Therefore, taking all these things into account, which are summarized in Table 1, we can say that the DTable algorithm is simpler than the "sorted-priority" algorithms but more complex than the DRR algorithm.

4 Decoupling the Bandwidth Assignment from the Latency Requirements

The easiest way of employing the DTable scheduler is by assigning all the table entries the same weight. This weight is the general MTU of the network. In this case, the bandwidth assigned to the ith flow what has assigned n_i table entries is $\phi_i = n_i/N$, where N is the total number of entries of the table. Therefore, if we want to provide bandwidth requirements, we must assign each flow a number of table entries proportional to the bandwidth that we want to assign to that flow. Note that if we distribute all the entries belonging to the same flow in a consecutive way in the arbitration table, the performance of the scheduler is going to be similar to the DRR scheduler. As stated before, the DRR algorithm is known to offer a bad latency performance. Therefore, if we want to improve the latency performance provided by this scheduler, we can distribute the table entries as the WF2Q variant of the list-based Weighted Round Robin proposed by Chaskar and Madhow [4].

However, following the Chaskar and Madhow approach we cannot differentiate among different levels of latency requirements. The WF2Q emulation tries to provide the best latency performance for all the flows given the amount of bandwidth that each flow has assigned. On the other hand, in [2], the approach is different. Instead of having a set of flows with different bandwidth requirements and trying to provide all of them with the best possible latency, flows present different latency requirements and the table is filled in such a way that their requirements are achieved. In [2], it is shown (in that case for InfiniBand) that controlling the maximum separation between any consecutive pair of entries assigned to the same flow, it is possible to control the latency of that flow. This is because this distance determines the maximum time that a packet at the head of a flow queue is going to wait until being transmitted. Note that this explains the different latency properties of the list-based WRR schedulers.

However, setting the distances among the table entries depending on the latency requirements faces the problem of bounding the bandwidth assignment to the latency requirements. If a maximum separation between any consecutive pair of table entries of a flow (or aggregated of flows with the same maximum separation requirement) is set, a certain number of them are being assigned, and hence a minimum bandwidth, to the flow in question. If the flow requires more bandwidth, we can assign more entries. However, to assign to the most latency-restrictive flows, a small amount of bandwidth is not possible because lower distances must be used for them. This can be a problem because the most latency-restrictive traffic does not usually present a high-bandwidth requirement.

Therefore, both approaches have the problem of bounding the bandwidth and latency assignments. In [20], we proposed a methodology to configure the DTable scheduler to decouple, at least partially, this bounding, and being able, until a certain degree, to provide bandwidth and latency requirements with a certain independence among them. With this methodology, we set the maximum distance between any consecutive pair of entries assigned to a flow

TABLE 2
Arbitration Table Parameters

$max\phi_i$	Maximum bandwidth assignable to the i^{th} flow
$min\phi_i$	Minimum bandwidth assignable to the i^{th} flow
ϕ_i	Bandwidth actually assigned to the i^{th} flow
N	Number of entries of the arbitration table
n_i	Number of entries assigned to the i^{th} flow
$GMTU$	General Maximum Transfer Unit
MTU_i	Specific Maximum Transfer Unit of the i^{th} flow
M	Maximum weight per table entry
$pool$	Bandwidth pool
k	Bandwidth pool decoupling parameter
w	Maximum weight decoupling parameter

depending on its latency requirement. Moreover, we set the weights of the table entries assigned to a flow depending on its bandwidth requirement. With this methodology, we can assign the flows with a bandwidth varying between a minimum and a maximum value that depends not only on the number of table entries assigned to each flow, but also on other configuration parameters.

Supposing an arbitration table with N entries in a network with a certain general MTU $GMTU$, and supposing the ith flow has assigned n_i table entries in order to fulfill its latency requirements, we would like to be able to assign the ith flow with a certain bandwidth ϕ_i in the most flexible possible way. This means that we would like the minimum bandwidth $min\phi_i$ that can be assigned to that flow to be as small as possible, and the maximum bandwidth $max\phi_i$ that can be assigned to that flow to be as large as possible. Table 2 shows all the involved parameters in the following statements.

Given the maximum weight M that can be assigned to a single table entry, the maximum total amount of weight that can be distributed among all the table entries is $M \times N$. However, we are going to fix in advance this total weight to a lower value. We are going to call this value *bandwidth pool*, or just *pool*, and will be determined by one of the decoupling configuration parameters as we will see later. Note that the value of M is going to be given by the size of the table entry weight field, which is hardware implementation dependent. However, we can always reduce this value by software, in order to accommodate it to our requirements. Moreover, as shown in [22], we can assign each flow a specific MTU MTU_i. In this situation, the bandwidth assigned to the ith flow is

$$\phi_i = \frac{\sum_{j=0}^{J} weight_j}{pool},$$

where J is the set of table entries assigned to the ith flow and $weight$ is the entry weight assigned to a table entry. Therefore, the minimum and maximum bandwidth that can be assigned to the ith flow is

$$min\phi_i = \frac{n_i \times MTU_i}{pool},$$
$$max\phi_i = \frac{n_i \times M}{pool}.$$

Let's define M and $pool$ in function of the GMTU and two configuration parameters w and k:

$$M = GMTU \times w,$$
$$pool = N \times GMTU \times k.$$

Note that $k \leq w$ because the bandwidth pool cannot be larger than $N \times M$. In this way, we can see that the minimum and maximum bandwidth that can be assigned to a flow depends not only on the proportion of table entries n_i that it has assigned, but also on the w and k parameters and the proportion between its specific MTU and $GMTU$:

$$min\phi_i = \frac{n_i \times MTU_i}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{MTU_i}{GMTU} \times \frac{1}{k},$$
$$max\phi_i = \frac{n_i \times GMTU \times w}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{w}{k}.$$

Note that varying the w and k parameters affects the minimum and maximum bandwidth that can be assigned to all the flows. However, assigning a specific MTU to a flow smaller than the $GMTU$ only affects the minimum bandwidth of that flow. When choosing the value of these parameters some considerations must be made. Note that the objective for this methodology is to decrease the minimum bandwidth and to increase the maximum bandwidth that can be assigned to a flow. In order to be able to assign a small amount of bandwidth to a flow with a high proportion of table entries, we can use a high value for the k parameter or decrease the MTU for that flow. However, the higher k is, the smaller the maximum bandwidth that can be assigned, and thus, the flexibility to assign the bandwidth decreases. We can solve this by increasing the value of w. However, increasing the value of the w parameter has two disadvantages. First of all, the memory resources to store each entry weight are going to be higher. Second, the latency of the flows is going to increase because each entry is allowing more information to be transmitted, and thus, the maximum time between any consecutive pair of table entries is higher. Using different MTUs for the different flows allows us to assign a smaller amount of bandwidth to those flows with a smaller specific MTU than the general MTU. However, as we show in [22], each specific MTU must be assigned taking into account the characteristics of the traffic flow in order to not worsen the performance of that flow.

Summing up, with this decoupling configuration methodology we can configure the DTable scheduler in order to provide a flow with latency and bandwidth requirements in a partially independent way. Depending on the characteristics and bandwidth and latency requirements of the different flows, the network manager must choose the most appropriate k, w, and specific MTU values, and distribute properly the bandwidth pool among the table entries, in order to provide the flows with their latency and bandwidth requirements in the most efficient way.

5 Providing QoS in High-Performance Networks with the DTable Scheduler

In this section, we propose how to configure the DTable scheduler, which would be located in each egress link of the network, of both network interfaces and switches, in order

to provide QoS in a high-performance network. Note that the trend in the last high-performance technologies proposed, like IBA or AS, is to employ link-level flow control mechanisms that make the network lossless and to aggregate flows with similar characteristics into a reduced number of VCs.

5.1 Traffic Classification

In order to provide QoS in a high-performance network, a set of Service Classes (SCs) with different requirements must be specified. Specifically, we must specify one SC per available VC. When various flows obtain access to the network, they will be assigned an SC depending on their characteristics. From a networking perspective, the general QoS provisioning parameters are throughput, latency, jitter, and loss rate. The degree of sensitivity to each of these parameters varies widely from one application to another. For example, multimedia applications are usually sensitive to latency and jitter, but many of them can tolerate packet losses to some extent. For a further discussion about different applications and their requirements, see [9]. Note that the loss rate is not taken into account in lossless networks.

In order to define the different SCs, we propose a traffic classification based on three network parameters: Bandwidth, latency, and jitter. In this way, this classification is similar to the one presented by Pelissier [26]. We distinguish between three broad categories of traffic:

- Network Control traffic: High-priority traffic to maintain and support the network infrastructure. One SC will be dedicated to this kind of traffic.
- QoS traffic: This traffic has explicit minimum bandwidth, maximum latency, and/or jitter requirements. Various QoS SCs can be defined with different specific requirements. This category can be divided into two groups:
  - Traffic which requires a given minimum bandwidth and must be delivered with a maximum latency and/or jitter in order for the data to be useful. Examples of such data streams include video conference, interactive audio, and video on demand.
  - Traffic which requires a given minimum bandwidth but is not particularly sensitive to latency or jitter.
- Best effort traffic: This traffic accounts for the majority of the traffic handled by data communication networks today, like file and printing services, web browsing, disk backup activities, etc. This traffic tends to be bursty in nature and largely insensitive to both bandwidth and latency. Among the best effort traffic, we can set different SCs which are only characterized by the differing priority among each other.

The schedulers must be properly configured at the different network elements to provide the different SCs with a differentiated treatment. Specifically, we are going to configure the schedulers in order to provide just bandwidth or bandwidth and latency simultaneously. Note that, although they are not totally correlated, if we limit the maximum latency performance, we are indirectly limiting the maximum jitter performance and thus, we can translate any maximum jitter requirement into a maximum latency requirement.

5.2 Configuring the DTable Scheduler

As stated in Section 4, the simplest way of implementing the DTable scheduler, and solving the AS table problem with variable packet sizes, is to assign each table entry a fixed constant weight. In this case, the minimum bandwidth assigned to an SC is proportional to the number of entries assigned to that SC. We can improve the latency performance provided in this way by distributing the table entries as the WF2Q variant of the list-based Weighted Round Robin proposed by Chaskar and Madhow [4]. This approach tries to improve the latency performance of all the SCs by emulating the order of transmission if the WF2Q would be implemented. Employing our decoupling configuration methodology, we can optimize the resources by providing a better latency to those SCs that really require it and at the same time providing them their bandwidth requirements.

5.2.1 Providing Latency Requirements

As a general rule, in order to provide each SC its latency requirements, we must first assign the table entries taking into account the maximum distance between any consecutive pair of entries devoted to the SCs with latency requirements (network control and QoS SCs with latency requirements) [2], assigning a lower maximum distance to those SCs with higher latency requirements. The rest of table entries can be distributed among those SCs that do not have latency requirements. We can assign those entries consecutively in the remaining gaps or can interleave the entries of the various SCs like in the list-based Weighted Round Robin in order to improve the latency performance.

However, although it is possible to obtain the maximum distance between consecutive table entries assigned to a VC required to provide its SC with specific hard latency requirements (taking into account the maximum weight assigned to each table entry, the network diameter, the routing and switching time, the packet size, the waiting time in intermediate switches, etc.), the objective of this paper has been to provide soft latency requirements based on a preestablished set of SCs with latency properties provided by maximum distance between consecutive entries assigned "a priory." Specifically, in Section 6, the approach that we have followed has been to assign each SC maximum distances in a power of two steps. With this "a priory" table entry distribution in mind, we can assign each flow to the most appropriate VC based on the flow latency requirements. After that, the bandwidth assignment is performed assigning each entry the appropriate weight.

5.2.2 Providing Bandwidth Requirements

The bandwidth that each VC should be assigned depends on the requirements of the SC it has assigned. We should provide the network control SC with enough bandwidth to manage the maximum expected amount of control traffic. QoS VCs should be assigned at least a bandwidth equal to the minimum bandwidth requirements of the QoS SCs. Finally, the bandwidth intended for the best effort SCs

should be assigned among them according to their different priority in order to provide them with a differentiated performance. This bandwidth assignment to VCs can be established statically or dynamically (as the different flows are established).

However, note that, as it is well known, interconnection networks are unable to achieve 100 percent global throughput. Therefore, not all the bandwidth can be distributed among the VCs, thereby requiring a certain bandwidth to be left unassigned. We propose to assign the network control VC with this bandwidth that should be left unassigned. Moreover, we propose not to assign best effort VCs with all the bandwidth that is intended for this class of traffic. We propose instead to assign them only a small amount of bandwidth proportional to their relative priority. The rest of the best effort bandwidth will also be assigned to the network control VC. In this way, the network control VC will have been assigned more bandwidth than it actually requires. However, by doing so, we achieve a better performance of the network control traffic. We also achieve a better performance and a better resilience against unexpected transient congestion due to bursty traffic of the QoS VCs. Note that the bandwidth unused by the control and QoS VCs is redistributed by the scheduler among the rest of VCs, including the best effort VCs, and thus they are going to take advantage of the bandwidth left over by the other VCs.

In order to assign a given SC with a minimum bandwidth, the amount of weight units from the bandwidth pool assigned to the SC table entries must accomplish with the proportion of desired egress link bandwidth. Therefore, when we know the maximum distance between two consecutive table entries, and thus, the number of entries, and the amount of bandwidth that we want to assign to each SC, we must choose the w and k parameters that make possible that distribution of bandwidth among the various SCs. Moreover, we can limit the MTU of some VCs in order to have a smaller minimum bandwidth for those SCs and for being able to use smaller k values. Note that those SCs that have high latency requirements and, thus, require more table entries, usually have small bandwidth requirements and use small packets. An example of these configuration process can be found in Section 6.

5.3 Admission Control

In a lossless network, congested packets are not thrown away and as such the loss rate due to congestion is zero. This has the advantage of avoiding retransmissions that would severely affect the latency and jitter performance of the flows. On the case of applications with packet loss resilience, it would allow to reduce the overhead due to the encoding techniques used to minimize the impact of errors. On the other hand, lossless networks have other problems, being the most important the formation of congestion (or saturation) trees [27]. These congestion trees may produce dramatic network performance degradation, affecting not only the flows traversing the original point of congestion, but other flows that share common upstream links.

A common approach to avoid this problem is by using an admission control (AC) mechanism. The AC decides whether a new connection is accepted or rejected and ensures that the entry of additional traffic into a network cannot create congestion. Many AC schemes have been proposed. In [29], the implementation and comparison of several possible approaches is presented.

We propose to apply the AC mechanism only to those VCs employed by the QoS SCs and not to the control SC or the best effort SCs. Note, that the QoS SCs are the ones which actually have specific QoS requirements. In addition, the latency constraints of the control traffic are not so clear. Moreover, we can assume that the amount of control traffic that is going to traverse the network is going to be quite small. And thus, taking into account the maximum amount of expected control traffic, the scheduling algorithm can assign the network SC with an *a priory* amount of bandwidth. As a first approximation, a bandwidth broker based on the average bandwidth value required per each flow [19] is used in Section 6.

6 Performance Evaluation

In this section, we evaluate the latency performance of the DTable scheduler in a multimedia scenario. For this purpose, we have developed a detailed simulator that allows us to model the network at the register transfer level, following the AS specification. Note, however, that we only use AS for assuming the AS network parameters, and that our proposals can be applied to any interconnection network technology.

Our purpose is to compare the DTable scheduling mechanism employing our decoupling configuration methodology with the one proposed in [4], specifically the WF2Q emulation approach, which is the best scheduler of the algorithms proposed in [4]. Moreover, we compare the throughput and latency performance of our proposal (the DTable scheduler employing our decoupling configuration methodology) with the one provided by the well-known SCFQ algorithm, and the DRR scheduler. We have chosen the SCFQ algorithm as an example of "sorted-priority" algorithm, and the DRR algorithm because of its very small computational complexity. In order to simulate these algorithms we use the credit aware versions of both algorithms (SCFQ Credit Aware and DRR Credit Aware, respectively) that we proposed in [21] for being used in networks with a link-level flow control network like AS. For this performance evaluation we have assumed a multimedia scenario using the IEEE standard 802.1D-2004 [12] traffic types.

6.1 Simulated Architecture

We have used a perfect-shuffle Bidirectional Multistage Interconnection Network (BMIN) with 64 end points connected using 48 eight-port switches (three stages of 16 switches). In AS any topology is possible, but we have used a MIN because it is a common solution for interconnection in current high-performance environments. The switch model uses a combined input-output buffer architecture with a crossbar to connect the buffers. Virtual output queuing has been implemented to solve the head-of-line blocking problem at switch level.

In our tests, the link bandwidth is 2.5 Gb/s but, with the AS 8b/10b encoding scheme, the maximum effective bandwidth for data traffic is only 2 Gb/s. We are assuming

TABLE 3
Set of SCs Considered

	IEEE 802.1D-2004 traffic types suggestion		Simulated traffic pattern	
Type	**SC**	**Description**	**Traffic pattern**	**Packet size**
Control	Network control (NC)	Supports the network infrastructure.	Bursts1	up to 256B
QoS	Voice (VO)	Limit of 10 ms for latency and jitter.	64 Kb/s CBR connections	160B
QoS	Video (VI)	Limit of 100 ms for latency and jitter.	3 Mb/s MPEG-4 traces	up to 2048B
QoS	Controlled load (CL)	Explicit bandwidth requirements.	750 kb/s CBR connections	2048B
Best-effort	Excellent-effort (EE)	Preferential best-effort traffic.	Bursts60	up to 2048B
Best-effort	Best-effort (BE)	LAN traffic as we know it today.	Bursts60	up to 2048B
Best-effort	Background (BK)	It should not impact other flows.	Bursts60	up to 2048B

some internal speedup ($\times 1.5$) for the crossbar, as is usually the case in most commercial switches. AS gives us the freedom to use any algorithm to schedule the crossbar, so we have implemented a round-robin scheduler. The time that a packet header takes to cross the switch without any load is 145 ns, which is based on the unloaded cut-through latency of the AS StarGen's *Merlin* switch [35].

A credit-based flow control protocol ensures that packets are only transmitted when there is enough buffer space at the other end to store them, making sure that no packets are dropped when congestion appears. VCs are used to aggregate flows with similar characteristics and the flow control and the arbitration is made at VC level. The MTU of an AS packet is 2,176 bytes, but we are going to use 2,048 bytes (a power of two) for simplicity but without losing generality. The credit-based flow control unit is 64 bytes, and thus, the MTU corresponds to 32 credits.

The buffer capacity is 32,768 bytes ($16 \times \mathrm{MTU}$) per VC at the network interfaces and 16,384 bytes ($8 \times \mathrm{MTU}$) per VC both at the input and at the output ports of the switches. If an application tries to inject a packet into the network interface but the appropriate buffer is full, we suppose that the packet is stored in a queue of pending packets at the application layer.

6.2 Traffic Model

The IEEE standard 802.1D-2004 [12] defines seven traffic types, or SCs, at Annex G, which are appropriate for this study. Table 3 shows each traffic type and its requirements. In this classification, we can differentiate one SC for control traffic, three SCs with explicit QoS requirements, and three SCs for best effort traffic with different levels of priority among each other.

The packets from each traffic type are simulated according to different distributions, as can be seen in Table 3. VO, VI, and CL SCs are composed of point-to-point connections of the given bandwidth. VO and CL SCs are generated following a Constant Bit Rate (CBR) distribution. In [39], several payload values for voice codec algorithms are shown. These values range from 20 bytes to 160 bytes. We have selected a payload of 160 bytes for the VO SC traffic. In the case of VI SC, MPEG-4 traces are used to generate the size of each frame. Each frame is injected into the network interfaces every 40 ms. If the frame size is bigger than the MTU, the frame is split into several packets which are injected all along the frame time. The traffic of the best effort SCs is generated according to a Bursts60 distribution [6]. This traffic is composed of bursts of 60 packets heading to the same destination. The packets' size is governed by a Pareto distribution, as recommended in [14]. In this way, many small size packets are generated, with an occasional large size packet. The periods between bursts are modeled with a Poisson distribution. The Bursts60 pattern models worst-case real traffic scenarios. The NC SC is generated in the same way than the Burst60 traffic but with only one packet burst. For all the cases, the destination pattern is uniform in order to fully load the network.

Note that the traffic model that we use in this performance evaluation is based on a multimedia environment. Our proposal is intended for being used in any environment where flows with different QoS requirements coexist in the network. However, the multimedia environment is the most straightforward one. In any case, in this scenario, we use a wide range of traffic behaviors, and thus the results obtained with these kinds of traffic can be generalized to other environments with other kind of traffic with QoS requirements.

6.3 Simulated Scenario and Scheduler Configuration

We suppose a scenario in which the goal is to dedicate around 2-8 percent of the egress link bandwidth to voice traffic (a lot but low-bandwidth requiring connections), 40-50 percent of bandwidth to video traffic (a lot and high-bandwidth requiring connections), around 20-25 percent of bandwidth to controlled load, and 10-20 percent bandwidth to best effort traffic. Moreover, we expect that the maximum network control bandwidth to be around 1 percent. These percentages are intended to represent a multimedia scenario with a realistic combination of traffic from applications with very different requirements. In order to provide a differentiated treatment to the considered SCs, we are going to configure the schedulers according to the different SCs requirements.

As stated in the previous section, we are going to suppose an admission control mechanism that ensures that the VO, VI, and CL VCs are not oversubscribed. This means that the sum of the average injection rate of the flows that traverse these VCs is equal to or smaller than the bandwidth that these VCs have reserved. In the case of the VI VC we are going to allow a smaller amount of bandwidth than it has reserved because of the high degree of burstiness of the video traffic. On the other hand, we do not make any assumption about best effort traffic.

TABLE 4
Application of the Decoupling Methodology

VC	Dist.	#entr.	%entr.	MTU_i	$min\phi_i$	$max\phi_i$
NC	2	32	50	192	0.093	2
VO	4	16	25	192	0.046	1
VI	8	8	12.5	2048	0.250	0.5
CL	16	4	6.25	2048	0.125	0.25
EE	32	2	3.125	1024	0.031	0.125
BE	64	1	1.562	1024	0.015	0.062
BK	64	1	1.562	1024	0.015	0.062
	Total	64	100		0.578	4

$N = 64, GMTU = 32, w = 2, k = 0.5$

In the case of the DTable scheduler configured with our decoupling methodology, which is referred just as DTable scheduler, we are going to distribute the table entries among the VCs according to their different levels of latency requirements. In this way, we have assigned a maximum distance of two to the VC that accommodates the NC SC, and a maximum distance of 64 to the BK VC. Table 4 shows the maximum distances that we have assigned to each VC. This table also shows the number of table entries and the proportion of table entries that these maximum distances entail for a table of 64 entries. In order to have a higher level of flexibility to distribute the bandwidth among the VCs, we have assigned each VC an specific MTU as small as the expected packet sizes of each SC allow. Specifically, we have assigned an MTU of 192 bytes for VCs that accommodate the NC and VO SCs traffic, an MTU of 2,048, which is the maximum, for VI and CL VCs, and an MTU of 1,024 for the best effort VCs.

The next step to configure the DTable scheduler is to choose a proper value for the w and k parameters. We have chosen the value of these parameters taking into account mainly that we want to assign the VI VC a bandwidth several times higher than the actual proportion of table entries assigned. Moreover, we want to assign the NC VC, which has assigned a very high proportion of table entries, a quite small proportion of bandwidth. However, we want to assign a value to the w parameter as small as possible in order to obtain a good latency performance. We have finally chosen a value of 2 for k and a value of 0.5 for w. This combination allows assigning each VC a bandwidth in the desired range, except for the NC VC to which we must assign at least 9 percent bandwidth. However, note that this amount of bandwidth is not going to be wasted because it is well known that interconnection networks are unable to achieve 100 percent global throughput. Therefore, we could change the w and k parameters to be able to assign this VC a lower amount of bandwidth, but it is not necessary because we assume 8 percent to be an appropriate percentage of unused bandwidth. In any case, the bandwidth left by the network control traffic is going to be distributed among the rest of VCs, specifically, the best effort SCs. Table 4 shows the minimum and maximum bandwidth that we can assign to each VC with this configuration.

In order to configure the DTable scheduler employing the WF2Q emulation, which we are going to refer as DTable-WF2Q, the DRR scheduler, and the SCFQ scheduler, we only can specify the bandwidth assigned to each VC. In the case of the DTable-WF2Q scheduler, all the entries must have the same weight assigned, in this case 32. This means that each time an entry is selected, the corresponding VC will be able to transmit 32×64 bytes, which corresponds with the AS MTU. The proportions of assigned bandwidth entail a number of entries per VC that are distributed in the table emulating the WF2Q algorithm. Table 5 shows the amount of bandwidth ϕ_i that we have actually assigned to each VC. This table also shows the configuration of the two DTable possibilities and the SCFQ and the DRR schedulers. Specifically, in the case of the DTable scheduler, this table shows the total weight (T. W.) that we have distributed among the table entries of each VC and the weight assigned to each table entry (E. W.) of each VC. In the case of the SCFQ algorithm, we have assigned each VC a weight proportional to the bandwidth allocation. To configure the DRR algorithm, we have assigned the VC with the minimum bandwidth requirement a quantum in credits equal to its specific MTU, and a proportional quantum to the rest of VCs. Note that in this way all the VCs have been assigned a quantum equal to or greater than their specific MTUs.

Finally, in the scenario that we are simulating, we are going to inject a fixed amount of control traffic (NC) and QoS traffic (VO, VI, and CL) all the time, and we gradually increase the amount of best effort traffic (EE, BE, and BK). The amount of QoS traffic to be injected is the maximum allowed by the admission control mechanism. Table 5 and Fig. 3 show the normalized injection rate of each VC. Note that we can obtain the specific injection rates by multiplying this normalized values by 2.5 Gb/s, which is, as stated

TABLE 5
Bandwidth Configuration of the Different Scheduling Algorithms

				Scheduler configuration							
		Injection		**DTable**			**DTable-WF2Q**			**SCFQ**	**DRR**
VC	ϕ_i	Min.	Max.	# entries	E. W.	T. W.	# entries	E. W.	T. W.	Weight	Quantum
NC	0.093	0.015	0.015	32	3	96	6	32	192	0.093	96
VO	0.046	0.046	0.046	16	3	48	3	32	96	0.046	48
VI	0.500	0.453	0.453	8	64	512	32	32	1024	0.500	512
CL	0.203	0.203	0.203	4	52	208	13	32	416	0.203	208
EE	0.093	0.015	0.093	2	48	96	6	32	192	0.093	96
BE	0.046	0.015	0.093	1	48	48	3	32	69	0.046	48
BK	0.015	0.015	0.093	1	16	16	1	32	32	0.015	16
Total	100	0.76	1	64		1024	64		2048	1	1024

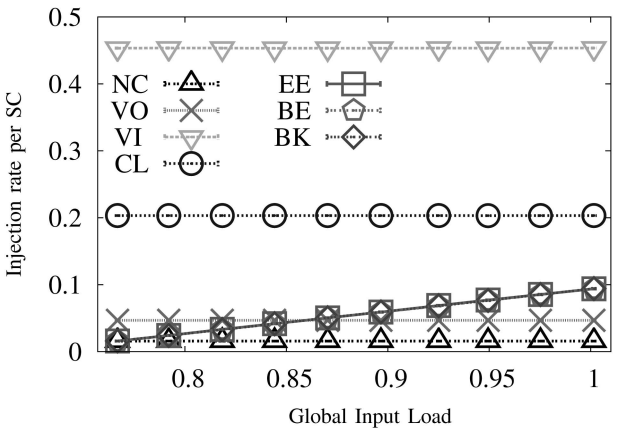

Fig. 3. Normalized injection rate per SC.

before, the assumed link bandwidth. Note also that the injection value of 0.453 assigned to VI VC traffic (lower than its reserved bandwidth) is due to the option of allowing a smaller amount of bandwidth for VI because of the high burstiness degree exhibited by video traffic.

6.4 Simulation Results

In this section, the throughput and latency performance of our proposals is shown. No statistics on packet loss are given because, as it has been said, we assume a credit-based flow control mechanism to avoid dropping packets. For each simulation we obtain the throughput, the average packet latency, and the cumulative distribution function (CDF) of latency of each SC. The CDF represents the probability of a packet achieving a latency equal to or lower than a certain value. The figures of this section show the average values and the confidence intervals at 90 percent confidence level of 10 different simulations performed at a given input load. Note that although the maximum latency statistic is valuable for some kind of applications, the maximum latencies obtained may vary a lot and, thus, are not very useful. For that reason, we use a quantile, the 99th percentile.

Note also that we do not take into account the computational complexity in the simulation infrastructure (all the schedulers take the same time to process a packet when it arrives at an egress queue and have the same arbitration time). Therefore, we are being in fact pessimistic in our evaluation of the DTable scheduler when compared with the SCFQ scheduler.

Fig. 4 gives a general overview of the performance when using the DTable scheduler. It shows the throughput, average latency, and the 99th percentile of latency. We do not show similar figures for the rest of schedulers due to lack of space. The throughput performance is the same for all the schedulers. However, although the specific latency values are different, the general tendencies for the other mechanisms are the same, and thus, the comments that we are going to make based on this figure can be generalized to the rest of schedulers. If we compare the injection (Fig. 3) and the throughput results, we can see that the NC and the QoS SCs obtain all the bandwidth they inject. However, when the network load is high (around 85 percent), the best effort SCs do not yield a corresponding result. From that input load, these SCs obtain a bandwidth proportional to their priority.

Regarding the latency performance, Fig. 4 shows that the latency (average 99th percentile) of the NC and QoS SCs grows with the load until they reach a certain value. Once this value is reached, the latency remains more or less constant. However, the average latency of best effort SCs continually grows with the load. Furthermore, it can be seen that best effort SCs obtain different average and maximum latency according to their different priority. In that sense, for example, the BK SC obtains a worse latency and starts to increase its latency sooner than the BE and EE SCs. The behavior of the best effort traffic is the same in all the schedulers and thus, no more comments are going to be done regarding these SCs.

Figs. 5, 6, 7, and 8 show a more detailed latency performance comparison between the different schedulers for the NC and QoS SCs. These figures show statistics on average latency, the 99th percentile of the latency, and the CDF of latency for the point of maximum load. We do not show the performance comparison for the best effort SCs because the relevant aspect of these SCs is not the actual latency values but that they obtain a differentiated performance among them. In this sense, the four schedulers provide this differentiation.

Regarding the NC SC, Fig. 5 shows that the SCFQ scheduler provides the best performance. However, the DTable scheduler provides a very similar performance. The DRR scheduler provides the worst performance and the DTable-WF2Q scheduler provides a better performance than the DRR scheduler, but worse than the SCFQ and DTable schedulers. Regarding the VO SC, Fig. 6 shows that the SCFQ and DTable schedulers and the DTable-WF2Q and DRR schedulers provide almost the same performance. The

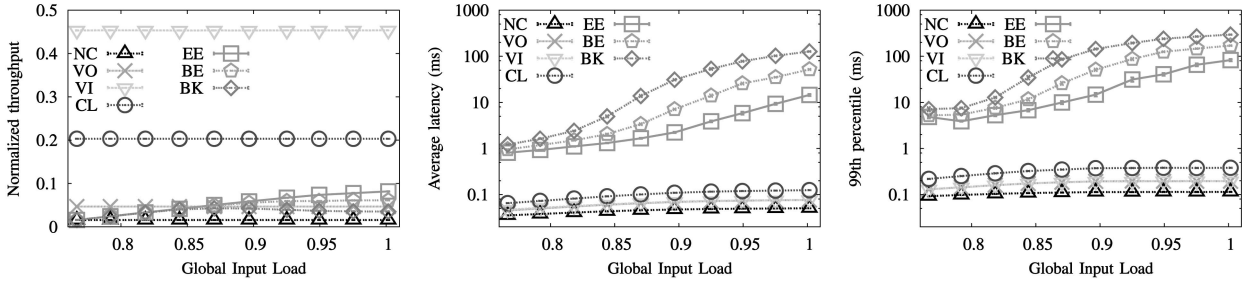

Fig. 4. Performance per SC of the DTable scheduler.

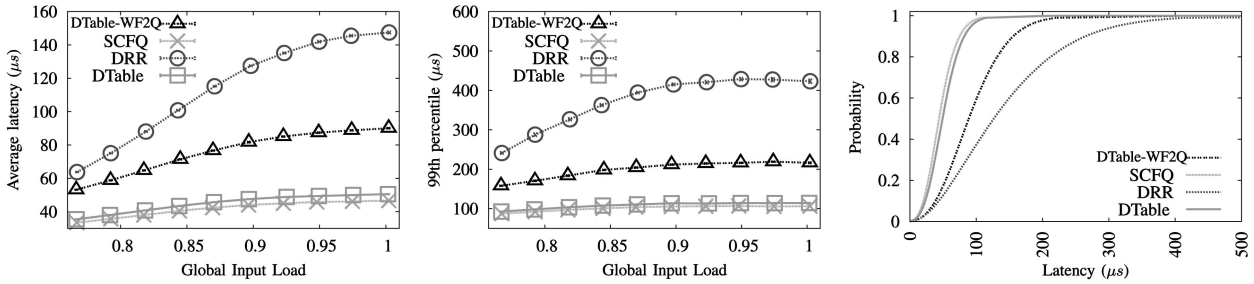

Fig. 5. Latency performance comparison for the network control SC.

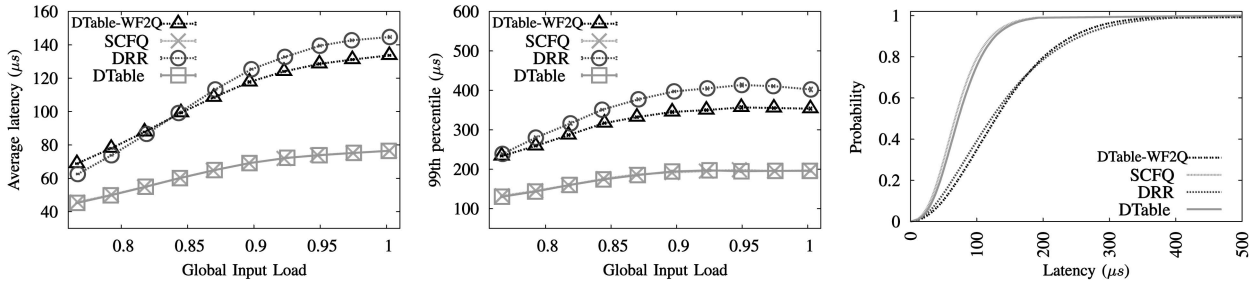

Fig. 6. Latency performance comparison for the voice SC.

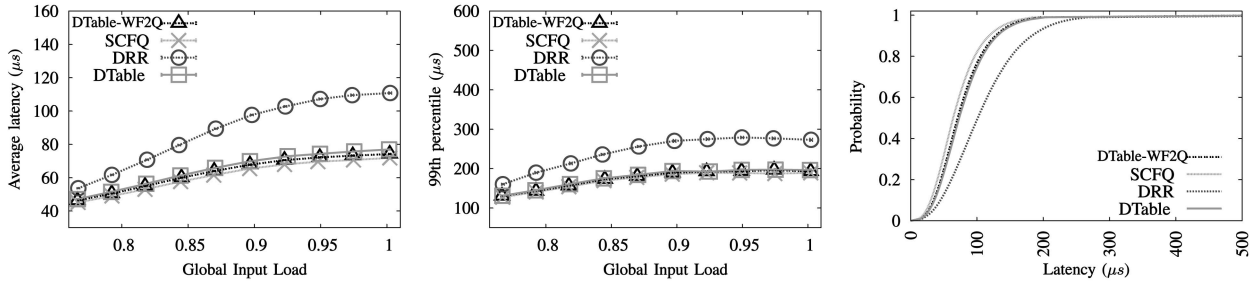

Fig. 7. Latency performance comparison for the video SC.

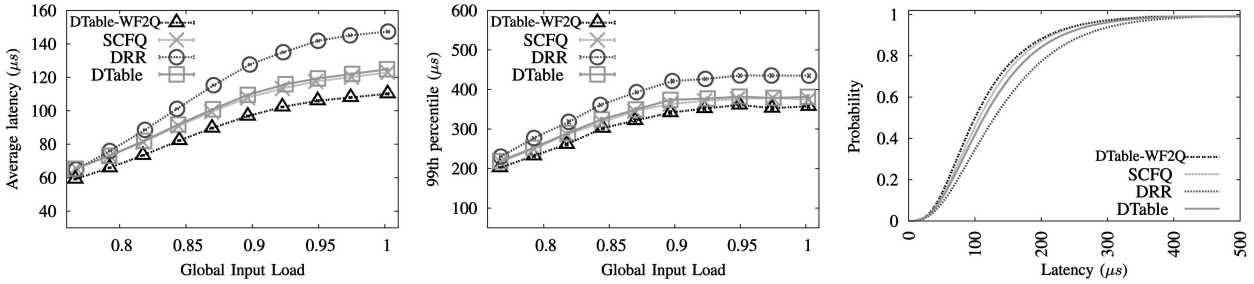

Fig. 8. Latency performance comparison for the controlled load SC.

latency provided by the SCFQ and DTable scheduler is better than the latency provided by the other two schedulers. Regarding the VI SC, Fig. 7 shows that all the schedulers except the DRR scheduler provide a similar performance, offering the DRR mechanism the worst performance. Regarding the CL SC, Fig. 8 shows that the DTable-WF2Q scheduler provides the best performance and the DRR the worst. The DTable and the SCFQ schedulers provide a similar and intermedium performance. Summing up, the DRR scheduler provides the worst latency performance, the SCFQ scheduler provides the best latency performance except for a slightly worse latency in the CL case, and the two DTable possibilities provide a better performance than the DRR scheduler.

Figs. 9, 10, and 11 show the percentage of improvement in the latency, at three different levels of load, for the NC and QoS SCs of the DTable scheduler over the DRR, SCFQ, and DTable-WF2Q schedulers. Specifically, Fig. 9 shows that the DTable scheduler provides a much better latency than the DRR scheduler. Note that the level of improvement can be up to 190 percent on average latency and up to 260 percent on the 99th percentile, which indicates that the maximum latency in the DTable case is much more limited than in the DRR case. Note also that the performance of the DRR scheduler depends on the frame length and, thus, in other scenarios, the performance could be even worse. Therefore, although the DRR scheduler has a lower complexity than the DTable

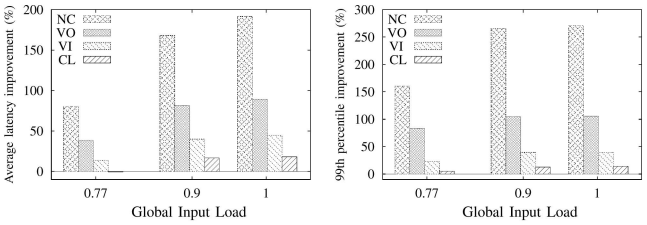

Fig. 9. Improvement of the DTable scheduler over the DRR scheduler.

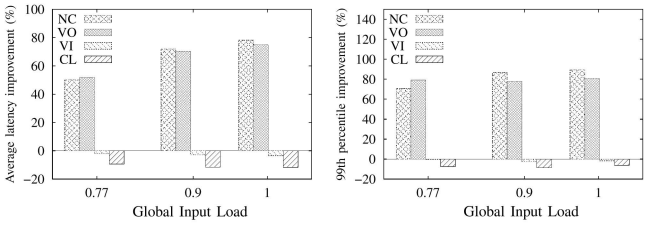

Fig. 11. Improvement of the DTable scheduler over the DTable-WF2Q scheduler.

scheduler, in our opinion, is not appropriate for providing QoS based on latency requirements.

Fig. 10 shows that the performance provided by the SCFQ scheduler is, in general, always slightly better than that provided by the DTable scheduler. Specifically, the latency degradation of the DTable scheduler over the SCFQ scheduler is less than 10 percent. Therefore, in our opinion, this slightly better performance does not justify employing the much more complex SCFQ scheduling algorithm.

Finally, Fig. 11 shows the comparison on latency performance that provides our decoupling configuration methodology over the WF2Q emulation configuration. Our methodology approximately provides up to 80 percent improvement over the WF2Q emulation variant for the NC and VO SCs, which are the most latency demanding. Moreover, it only entails up to 10 percent degradation for the VI and CL, which have less strict latency requirements. Therefore, with our decoupling methodology, we can provide a better latency performance to those applications that really need it.

7 Conclusions and Future Work

A key component for networks with QoS support is the output scheduling algorithm, which selects the next packet to be sent and determines when it should be transmitted. An ideal scheduling algorithm implemented in a high-performance network with QoS support should satisfy two main properties: good latency and simplicity. In this paper, we have thoroughly addressed the issue of providing QoS employing the DTable scheduler. This new table-based scheduler, which works properly with variable packet sizes, has a hardware complexity higher than the DRR scheduler, but lower than the SCFQ and other "sorted-priority" algorithms. We have presented a configuration methodology that allows us to decouple at least partially the bandwidth and latency assignments. This allows us to make an optimized use of the resources. Moreover, we have presented a general framework that employs this scheduling mechanism to support the coexistence of traffic with explicit requirements based on bandwidth and latency at the same time that allows several levels of best effort traffic.

Finally, in Section 6, we have evaluated the performance of our proposals in a multimedia scenario using the IEEE standard 802.1D-2004 traffic types. We have not only compared the throughput and latency performance of this scheduler with the one provided by the SCFQ and the DRR scheduler, but also we have compared our decoupling configuration methodology with the one proposed in [4]. Simulation results show that our proposal provides a much better latency performance than the DRR scheduler and a similar performance to the one provided by the more complex SCFQ algorithm. Moreover, we have shown that our decoupling methodology is able to provide a better performance to those applications that really need it, independently of their bandwidth requirements. In this way, we have shown the advantages of our configuration methodology over the emulation of some "sorted-priority" algorithm like the WF2Q algorithm. Therefore, we can conclude that the DTable scheduler configured with our decoupling methodology is able to provide a good latency performance with a low hardware complexity.

In this paper, we have considered the complexity of the DTable scheduler and the rest of schedulers considered in a rather general way. As future work, we are focusing our attention on performing a deeper hardware study in order to offer estimates about the silicon area and the arbitration time that they would require. Moreover, we are also working on obtaining mathematical expressions to characterize the QoS properties such as latency bounds of the DTable scheduler. We can find in the literature such expression for many well-known scheduling algorithms. However, completely new expressions should be obtained for the DTable scheduler, because, as far as we know, there is no formal study on the properties of table-based schedulers.

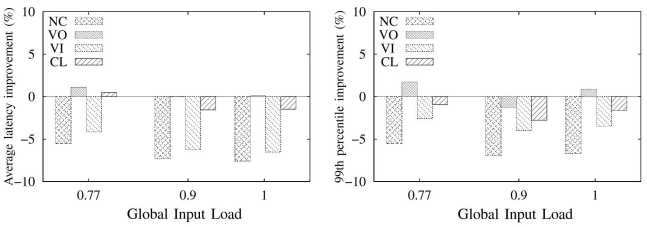

Fig. 10. Improvement of the DTable scheduler over the SCFQ scheduler.

Acknowledgments

This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grants "Consolider Ingenio 2010 CSD2006-00046" and "TIN2009-14475-C04-03," and by the Junta de Comunidades de Castilla-La Mancha under Grant PCC08-0078-9856. Raúl Martínez was with the University of Castilla-La Mancha when the main ideas of the paper were developed.

References

[1] Advanced Switching Interconnect Special Interest Group, *Advanced Switching Core Architecture Specification, Revision 1.0*, Dec. 2003.

[2] 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.
[3] J. Bennett and H. Zhang, "WF2Q: Worst-Case Fair Weighted Fair Queueing," *Proc. IEEE INFOCOM,* 1996.
[4] H.M. Chaskar and U. Madhow, "Fair Scheduling with Tunable Latency: A Round-Robin Approach," *IEEE/ACM Trans. Networking,* vol. 11, no. 4, pp. 592-601, Aug. 2003.
[5] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J.B. Carter, "Interconnect-Aware Coherence Protocols for Chip Multiprocessors," *Proc. Int'l Symp. Computer Architecture (ISCA),* pp. 339-351, 2006.
[6] N. Chrysos and M. Katevenis, "Multiple Priorities in a Two-Lane Buffered Crossbar," *Proc. IEEE Globecom '04,* Nov. 2004.
[7] G. Chuanxiong, "SRR: An O(1) Time Complexity Packet Scheduler for Flows in Multi-Service Packet Networks," *Proc. ACM SIGCOMM,* pp. 211-222, 2001.
[8] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulations of a Fair Queuing Algorithm," *Proc. ACM SIGCOMM,* 1989.
[9] M.A. El-Gendy, A. Bose, and K.G. Shin, "Evolution of the Internet QoS and Support for Soft Real-Time Applications," *Proc. IEEE,* vol. 91, no. 7, pp. 1086-1104, July 2003.
[10] S.J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," *Proc. IEEE INFOCOM,* 1994.
[11] A.G. Greenberg and N. Madras, "How Fair is Fair Queuing," *J. ACM,* vol. 39, no. 3, pp. 568-598, 1992.
[12] IEEE, 802.1D-2004: Standard for Local and Metropolitan Area Networks, http://grouper.IEEE.org/groups/802/1/, 2004.
[13] InfiniBand Trade Association, *InfiniBand Architecture Specification Volume 1, Release 1.0,* Oct. 2000.
[14] R. Jain, *The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling.* John Wiley and Sons, Inc., 1991.
[15] S.S. Kanhere, A. Parekh, and H. Sethu, "Fair and Efficient Packet Scheduling in Wormhole Networks," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 623-631, May 2000.
[16] S.S. Kanhere and H. Sethu, "Fair, Efficient and Low-Latency Packet Scheduling Using Nested Deficit Round Robin," *Proc. IEEE Workshop High Performance Switching and Routing,* pp. 6-10, May 2001.
[17] S.S. Kanhere and H. Sethu, "On the Latency and Fairness Characteristics of Pre-Order Deficit Round Robin," *Computer Comm.,* vol. 27, no. 7, pp. 664-678, 2004.
[18] S.S. Kanhere, H. Sethu, and A.B. Parekh, "Fair and Efficient Packet Scheduling Using Elastic Round Robin," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 324-336, Mar. 2002.
[19] E.W. Knightly and N.B. Shroff, "Admission Control for Statistical QoS: Theory and Practice," *IEEE Network,* vol. 13, no. 2, pp. 20-29, Mar./Apr. 1999.
[20] R. Martínez, F.J. Alfaro, and J.L. Sánchez, "Decoupling the Bandwidth and Latency Bounding for Table-Based Schedulers," *Proc. Int'l Conf. Parallel Processing (ICPP),* Aug. 2006.
[21] R. Martínez, F.J. Alfaro, and J.L. Sánchez, "Implementing the Advanced Switching Minimum Bandwidth Egress Link Scheduler," *Proc. IEEE Int'l Symp. Network Computing and Applications (NCA '06),* July 2006.
[22] R. Martínez, F.J. Alfaro, and J.L. Sánchez, "Improving the Flexibility of the Deficit Table Scheduler," *Proc. Int'l Conf. High Performance Computing (HiPC),* Dec. 2006.
[23] P.L. Montessoro and D. Pierattoni, "Advanced Research Issues for Tomorrow's Multimedia Networks," *Proc. Int'l Symp. Information Technology (ITCC),* 2001.
[24] 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, June 1993.
[25] K.I. Park, *QoS in Packet Networks.* Springer, 2005.
[26] J. Pelissier, "Providing Quality of Service over Infiniband Architecture Fabrics," *Proc. Eighth Symp. Hot Interconnects,* Aug. 2000.
[27] G. Pfister and A. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Trans. Computers,* vol. 34, no. 10, pp. 943-948, Oct. 1985.
[28] S. Ramabhadran and J. Pasquale, "Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay," *Proc. ACM SIGCOMM,* Aug. 2003.
[29] S.A. Reinemo, F.O. Sem-Jacobsen, T. Skeie, and O. Lysne, "Admission Control for Diffserv Based Quality of Service in Cut-Through Networks," *Proc. 10th Int'l Conf. High Performance Computing,* Dec. 2003.
[30] S.A. Reinemo, T. Skeie, T. Sødring, 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.
[31] D. Saha, S. Mukherjee, and S. Tripathi, "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks," *IEEE/ACM Trans. Networking,* vol. 6, no. 6, pp. 779-796, Dec. 1998.
[32] R. Seifert, *Gigabit Ethernet: Technology and Applications for High-Speed LANs.* Addison-Wesley Longman Publishing Co., Inc., 1998.
[33] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," *Proc. ACM SIGCOMM,* pp. 231-242, 1995.
[34] V. Sivaraman, "End-to-End Delay Service in High Speed Packet Networks Using Earliest Deadline First Scheduling," PhD thesis, Univ. of California, 2000.
[35] StarGen, *StarGen's Merlin Switch,* http://www.stargen.com/products/merlin_switch.shtml, 2004.
[36] D. Stiliadis, "Traffic Scheduling in Packet-Switched Networks: Analysis, Design, and Implementation," PhD thesis, Univ. of California, 1996.
[37] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Trans. Networking,* vol. 6, no. 5, pp. 611-624, Oct. 1998.
[38] S-C. Tsao and Y-D. Lin, "Pre-Order Deficit Round Robin: A New Scheduling Algorithm for Packet-Switched Networks," *Computer Networks,* vol. 35, nos. 2/3, pp. 287-305, 2001.
[39] A. Tyagi, J.K. Muppala, and H. de Meer, "VoIP Support on Differentiated Services Using Expedited Forwarding," *Proc. IEEE Int'l Performance, Computing, and Comm. Conf. (IPCCC),* Feb. 2000.
[40] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks," *Proc. IEEE,* vol. 83, no. 10, pp. 1374-1396, Oct. 1995.

Raúl Martínez-Moráis received the MS degree in computer science in 2003 and the PhD degree in 2007 from the University of Castilla-La Mancha. He is currently a researcher at the Intel Barcelona Research Center. His research interests include high-performance local area networks, QoS, design of high-performance switches, multicore architectures, thread-level parallelism, and dynamic binary optimization.

Francisco J. Alfaro-Cortés 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 a professor of computer architecture and technology in the Computer Systems Department at the Castilla-La Mancha University. His research interests include high-performance local area networks, QoS, 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 is a member of the Computer Systems Department (formerly Computer Science Department) at the University of Castilla-La Mancha. He is currently an associate professor of computer architecture and technology. His research interests include parallel architectures and parallel programming, QoS in high-speed networks, and networks on chip.