Comparing the latency performance of the DTable and DRR schedulers

Raúl Martínez, Francisco J. Alfaro, José L. Sánchez

University of Castilla-La Mancha
Computing Systems Department
02071 - Albacete, Spain
{raulmm, falfaro, jsanchez}@dsi.uclm.es

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. The Deficit Round Robin (DRR) algorithm is known to have a very little implementation complexity. However, depending on the situation, its latency performance can be very bad.

On the other hand, 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 and face the problem of bounding the bandwidth and latency assignments. 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 to decouple the bounding of bandwidth and latency assignments.

In this paper, we review these proposals and present simulation results that show that the DTable scheduler is able to provide a better latency performance than the DRR scheduler, with only a slightly higher implementaion and computational complexity.

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 Introduction

Current packet networks are required to carry not only traffic of applications such as e-mail or file transfer, which does not require pre-specified service guarantees, but also traffic of other applications that require different performance guarantees, like real-time video or telephony [10]. The best-effort service model, though suitable for the first type of applications, is not so for applications of the other type [11]. 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 [4]. Therefore, high performance packet networks need to enable Quality of Service (QoS) provisioning. The provision of QoS in computing and communication environments is currently the focus of much discussion and research in industry and academia. 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, on the basis of some expected performance metrics.

An ideal scheduling algorithm implemented in a high performance network with QoS support should satisfy two main properties: good delay and implementation simplicity. The design of a traffic scheduling algorithm involves an inevitable trade-off among these properties. Many scheduling algorithms have been proposed. Among them, the "sorted-priority" family of algorithms are known to offer very good delay [13]. However, their computational complexity is very high, making their implementation in high-speed networks rather difficult. In order to avoid the complexity of the sorted-priority approach, the Deficit Round Robin (DRR) algorithm [12] has been proposed.

The DRR algorithm associates each flow[1] with a *quan-*

[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.

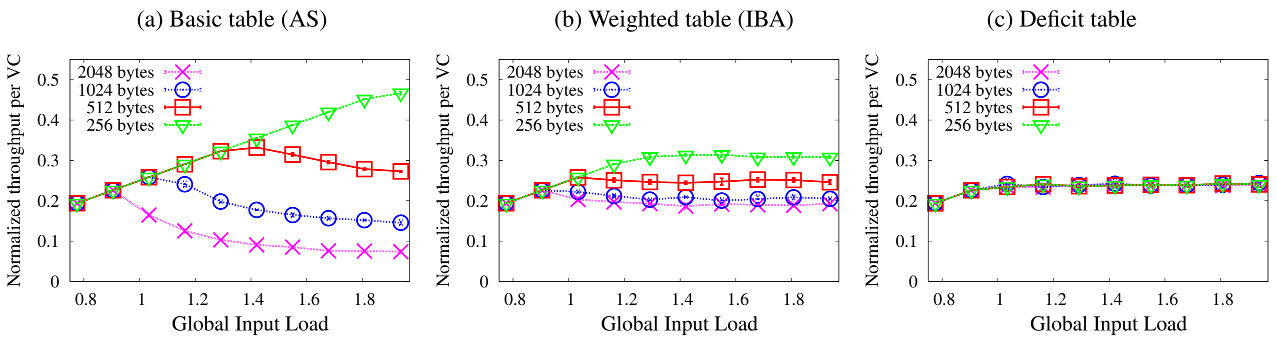

Figure 1. Performance of several table-based schedulers for flows with different packet size.

tum and a *deficit counter*. The quantum assigned to a flow is proportional to the bandwidth assigned to that flow. The sum of all the quantums is called the frame length. The deficit counter is set to 0 at the beginning. The scheduler visits sequentially each flow. For each flow, 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 flow. 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. The main advantage of the DRR scheduler is its computational simplicity. Recent research in the DiffServ area [3] proposes the DRR as a feasible solution for implementing the Expedited Forwarding Per-hop Behavior [5]. However, the main problem of this algorithm is that its delay depends on the frame length. Depending on the situation, the frame can be very long, and thus, the latency would be very bad.

On the other hand, in the table-based schedulers instead of serving packets of a flow in a single visit per frame, the service is distributed throughout the entire frame. This approach is followed by two of the last high-performance network interconnection proposals: Advanced Switching (AS) [1] and InfiniBand (IBA) [6]. These table-based schedulers can provide a good latency performance with a low computational complexity. However, these schedulers do not work properly with variable packet sizes and face the problem of bounding the bandwidth and latency assignments [9, 2]. In [8] we reviewed these problems and proposed a new table-based scheduler, that works properly with variable packet sizes. Moreover, we proposed a decoupling methodology to configure this table-based scheduler without the bounding between bandwidth and latency. In this paper, we compare the latency performance of the DTable scheduler with the latency performance of the DRR scheduler.

The structure of the paper is as follows: In Section 2, we review the DTable scheduler and our methodology to decouple the bandwidth and latency assignments. Details on the experimental platform and the performance evaluation are presented in Section 3. Finally, some conclusions are given and future work is proposed.

2 The Deficit Table scheduler

In [8] we proposed a new table-based scheduling algorithm that works properly with variable packet sizes. 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. In this section, we show how the DTable scheduling algorithm works and how it can be configured to decouple the bandwidth and latency assignments. Moreover, we make some complexity considerations.

2.1 The DTable scheduling mechanism

The main problem of the AS and IBA table-based schedulers is that they do not work in a proper way with variable packet sizes, which is common in actual traffic. If the average packet size of the different flows is different, the bandwidth that the flows obtain may not be proportional to the number of table entries [8].

Figure 1 shows the performance of various table-based schedulers when there are four Virtual Channels (VCs) in the network. Note that we use VCs to aggregate flows with similar characteristics and make 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 each VC according to the packet size that the flows using that VC use. The simulated architecture is the same as that used for the performance evaluation in Section 3.

Figure 1(a) 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). Figure 1(b) shows the performance of a weighted table that works in this way. We have assigned all the entries the same weight: 2176 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 [8] we proposed a new table-based scheduling algorithm that works properly with variable packet sizes (as can be seen in Figure 1(c)). In order to do so, the DTable scheduler defines an arbitration table in which each table entry has associated a flow identifier and an *entry weight*. 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 found. A flow is considered active when its queue has at least one packet and the link-level flow control, if exists, 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 packets from the selected flow until the accumulated weight is smaller than the size of the packet at the head of the selected flow or the selected flow becomes inactive. In the first case, the unused accumulated weight is saved in the deficit counter, representing the amount of weight that the scheduler owes the queue. In the second case, the remaining accumulated weight is discarded and the deficit counter is set to zero. Each time a packet is transmitted, the accumulated weight is reduced by the packet size. The weights are usually expressed in flow control credits. Note that the 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 service is distributed all along the entire frame.

In order to keep the computational complexity low, the minimum value that a table entry can have associated is the Maximum Transfer Unit (MTU) of the network. This is the smallest value that ensures that there 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. Note that this consideration is also made in the DRR algorithm definition [12].

2.2 Configuring the DTable scheduler

In [2], we explained how to configure table-based schedulers (in that case for IBA) to provide bandwidth and latency guarantees. In order to provide a flow with a minimum bandwidth, the number of table entries assigned to that flow must accomplish with the proportion of desired egress link bandwidth. In order to provide a flow with maximum latency requirements, the maximum separation between two consecutive table entries devoted to that flow must be fixed.

This way of assigning the entries of the table faces the problem of bounding the bandwidth and latency assignments. If a maximum separation between any consecutive pair of table entries of a flow is set, a certain number of them are being assigned, and hence a minimum bandwidth, to the flow in question.

In [8] we proposed a methodology to configure the DTable scheduler to decouple, at least partially, the bounding between the bandwidth and latency assignments. 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, but also on two table configuration parameters. We have called these parameters w and k. Supposing an arbitration table with N entries in a network with a certain MTU, the w parameter determines the maximum weight M that can be assigned to a single table entry in function of the MTU: $M = MTU \times w$. The k parameter determines the total weight that can be distributed between all the table entries. We call this value the *bandwidth pool*: $pool = N \times MTU \times k$. The total number of weight units[2] from the bandwidth pool that the table entries of a flow have assigned fixes the bandwidth that the flow has actually assigned.

The w and k parameters fix the minimum bandwidth $min\phi_i$ and the maximum bandwidth $max\phi_i$ that can be assigned to the i^{th} flow depending on the number of table entries n_i that it has assigned:

$$min\phi_i = \frac{n_i \times MTU}{pool} = \frac{n_i \times MTU}{N \times MTU \times k} = \frac{n_i}{N} \times \frac{1}{k}$$

$$max\phi_i = \frac{n_i \times M}{pool} = \frac{n_i \times MTU \times w}{N \times MTU \times k} = \frac{n_i}{N} \times \frac{w}{k}$$

Summing up, the DTable scheduler is a table-based scheduler that is able to deal properly with variable packet

[2]A weight unit is equivalent to a flow control credit

sizes and, using our configuration methodology, allows us to provide a flow with latency and bandwidth requirements in a partially independent way.

2.3 Implementation considerations

Note that, in the case of the DRR algorithm, provided that each flow is allocated a quantum no smaller than the MTU and a list of active flows is maintained, the algorithm can cycle through the list knowing that it is always possible to transmit at least one packet from each flow. 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. If a new active flow is selected the scheduler must only save the remaining quantum in the appropiate deficit counter and compute the new total amount of information to be transmitted from the selected flow. This computation can be performed with simple integer units.

The requirements to implement the DTable scheduler is quite similar, however, in the case of this scheduler, a list of active flows would not be as simple to maintain as in the DRR case, because flows 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 flow 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 this procedure 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.

However, even with this consideration, the DTable scheduler has not the problem of the increasing tag value and does not require the very complex hardware, including floating-point units, to manage the time tags, which is needed to implement the schedulers of the “sorted-priority” family of algorithms [8].

3 Performance evaluation

In this section, we compare the performance of the DTable scheduler with the performance of the DRR scheduler. 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 [1]. Note, however, that our proposals can be applied to any interconnection network technology.

3.1 Simulated architecture

We have used a perfect-shuffle Bidirectional Multi-stage Interconnection Network (BMIN) with 64 end-points connected using 48 8-port switches (3 stages of 16 switches). The switch model uses a combined input-output buffer architecture with a crossbar to connect the buffers. 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 some internal speed-up (x1.5) for the crossbar, as is usually the case in most commercial switches. 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.

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. AS uses Virtual Channels (VCs) 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 2176 bytes, but we are going to use 2048 bytes (a power of two) for simplicity, but without loosing generality. The credit-based flow control unit is 64 bytes, and thus, the MTU corresponds to 32 credits.

The buffer capacity is 32768 bytes (16$\times$MTU) per VC at the network interfaces and 16384 bytes (8$\times$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, the packet is stored in a queue of pending packets at the application layer.

3.2 Simulated scenario and scheduler configuration

We have defined 7 VCs with different distances between consecutive entries in the arbitration table. In a real case we would assign the traffic flows to these VCs depending on their latency requirements. Note that we are going to consider the requirements of a VC as the requirements of the traffic that is going to be transmitted using that VC. We have called these VCs D2, D4, D8, D16, D32, D64, and D64’, indicating the distance between any pair of consecutive table entries. Therefore, D2 has more strict latency requirements than D4, D4 than D8, and so on. A table of 64 entries has been used in the simulations.

In order to allow the decoupling between the latency requirements of the VCs and the bandwidth assigned to them, we have used our methodology, assigning to the k parameter a value of 2 (the bandwidth pool is 2 times the MTU multiplied by the number of entries), and the w parameter a value of 4 (each table entry can be assigned a maximum weight of 4 times the MTU). Table 1 shows the number of

Table 1. Configuration of the DTable and DRR schedulers.

		DTable						DRR
VC	ϕ_i	**#entries**	**%entries**	$min\phi_i$	$max\phi_i$	Entry weight	Total weight	Quantum
D2	25	32	50	25	100	32	1024	256
D4	25	16	25	12.5	50	64	1024	256
D8	25	8	12.5	6.25	25	128	1024	256
D16	12.5	4	6.25	3.125	12.5	128	512	128
D32	6.25	2	3.125	1.5625	6.25	128	256	64
D64	3.125	1	1.5625	0.78125	3.125	128	128	32
D64'	3.125	1	1.5625	0.78125	3.125	128	128	32
Total	100	64	100	50	200		4096	1024

entries assigned to each VC, the percentage of entries that this entails, and the minimum and maximum bandwidth that can be assigned to each VC using the decouplig methodology.

Table 1 also shows the actual bandwidth configuration of the DTable and DRR schedulers. This table shows the amount of bandwidth ϕ_i that we have assigned to each VC. Regarding the DTable scheduler, it 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. For example, in order to assign 25% of bandwidth to the D2 VC, 1024 of the 4096 credits of the bandwidth pool must be assigned to it. Therefore, 32 credits have been assigned to each one of its 32 table entries. Regarding the DRR scheduler, it shows the quantum assigned to each VC. The D64 and D64' VCs, which are the VCs with the minimum bandwidth requirement, are assigned a quantum that corresponds to 32 credits (the MTU), which ensures that at least one packet is going to be transmitted when a VC is selected. The rest of VCs have assigned a proportional quantum.

Note that the DRR frame length for this scenario is 1024 flow control credits. In the DTable scheduler that length is 4096 flow control credits. However, in the DTable case, the quantum assigned to each VC is distributed all along the frame length.

We have injected an increasing amount of traffic of all the VCs and study the performance of both schedulers at different network load levels. The traffic load is composed of self-similar point-to-point flows of 1 Mb/s. The destination pattern is uniform in order to fully load the network. The packets' size is governed by a Pareto distribution, as recommended in [7]. In this way, many small-sized packets are generated, with an occasional packet of large size. The minimum payload size is 56 bytes, the maximum 2040 bytes, and the average 176 bytes, which represents enough packet size variability. The AS packet header size is 8 bytes. The periods between packets are modelled with a Poisson distribution.

3.3 Simulation results

Figures 2 and 3 show the average values and the confidence intervals at 90% confidence level of ten different simulations performed at a given input load. For each simulation we obtain the normalized average throughput, the average message injection latency, and the maximum message injection latency of the flows using each VC. Figure 4 shows a zoom on the latency performance shown in the previous figures, which allows to perceive in a better way the diferences among both schedulers. Finally, Figure 5 shows the percentage of improvement on the average and maximum latency provided by the DTable scheduler over the performance provided by the DRR algorithm for each VC.

Regarding the throughput performance, Figures 2 and 3 show the normalized throughput results per VC of both schedulers. Note that both schedulers provide a similar throughput performance. As we can see, when the load is low, all the VCs obtain the bandwidth they inject. However, when the load is high (around 90%) the VCs do not yield a corresponding result, obtaining a bandwidth proportional to their assigned bandwidth.

Regarding the latency performance, the figures show that when the load is very low, all the VCs present a similar low latency. This is because at this load level there are few packets being transmitted through the network, and thus, there are few conflicts between them. However, when the load increases, the latency also increases because some packets must wait in the buffers until others have been transmitted. It is at this point when the scheduling algorithm assumes an important role and the VCs obtain a different latency depending on the scheduling algorithm. However, when the load of a VC begins to outstrip its throughput, the latency starts to grow very fast. This is because the buffers used for that VC begin to be full. Finally, the buffers become completely full and the latency stabilizes at a given value which depends on the buffers' size and the bandwidth assigned to that VC, but not on the scheduling algorithm. This is the

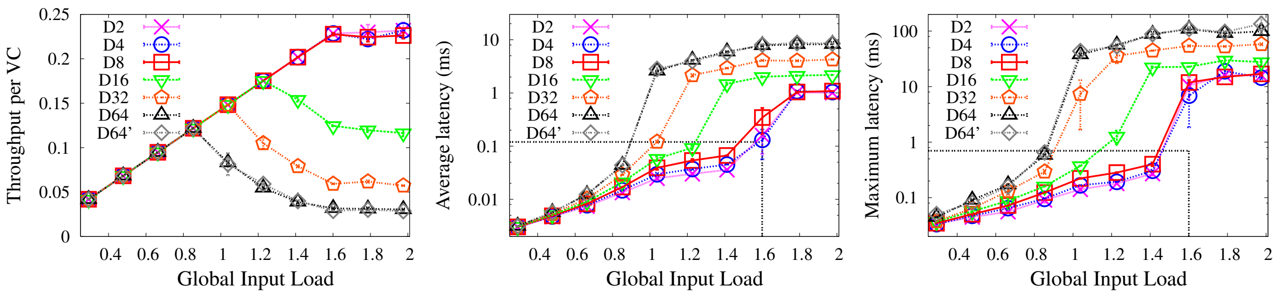

Figure 2. Normalized throughput per VC provided by the DTable scheduler.

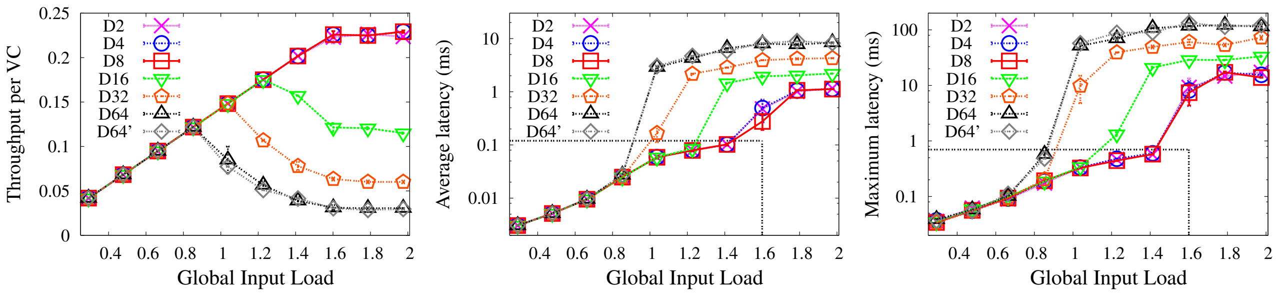

Figure 3. Normalized throughput per VC provided by the DRR scheduler.

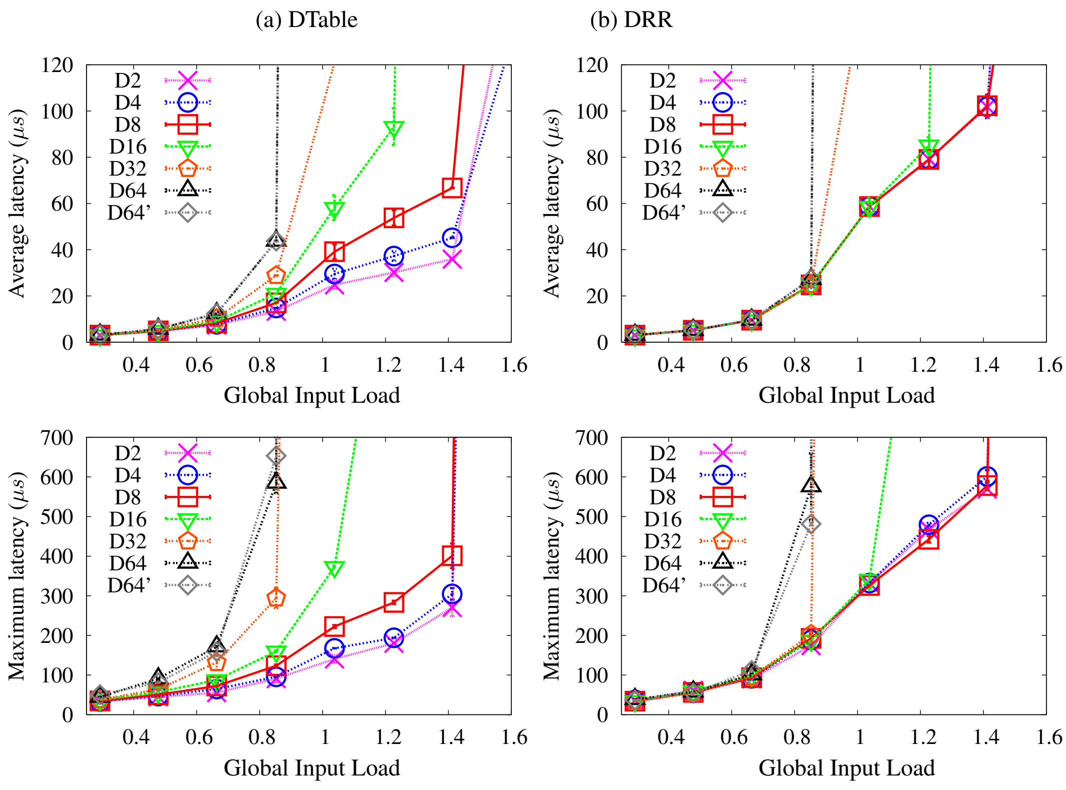

Figure 4. Latency performance comparison.

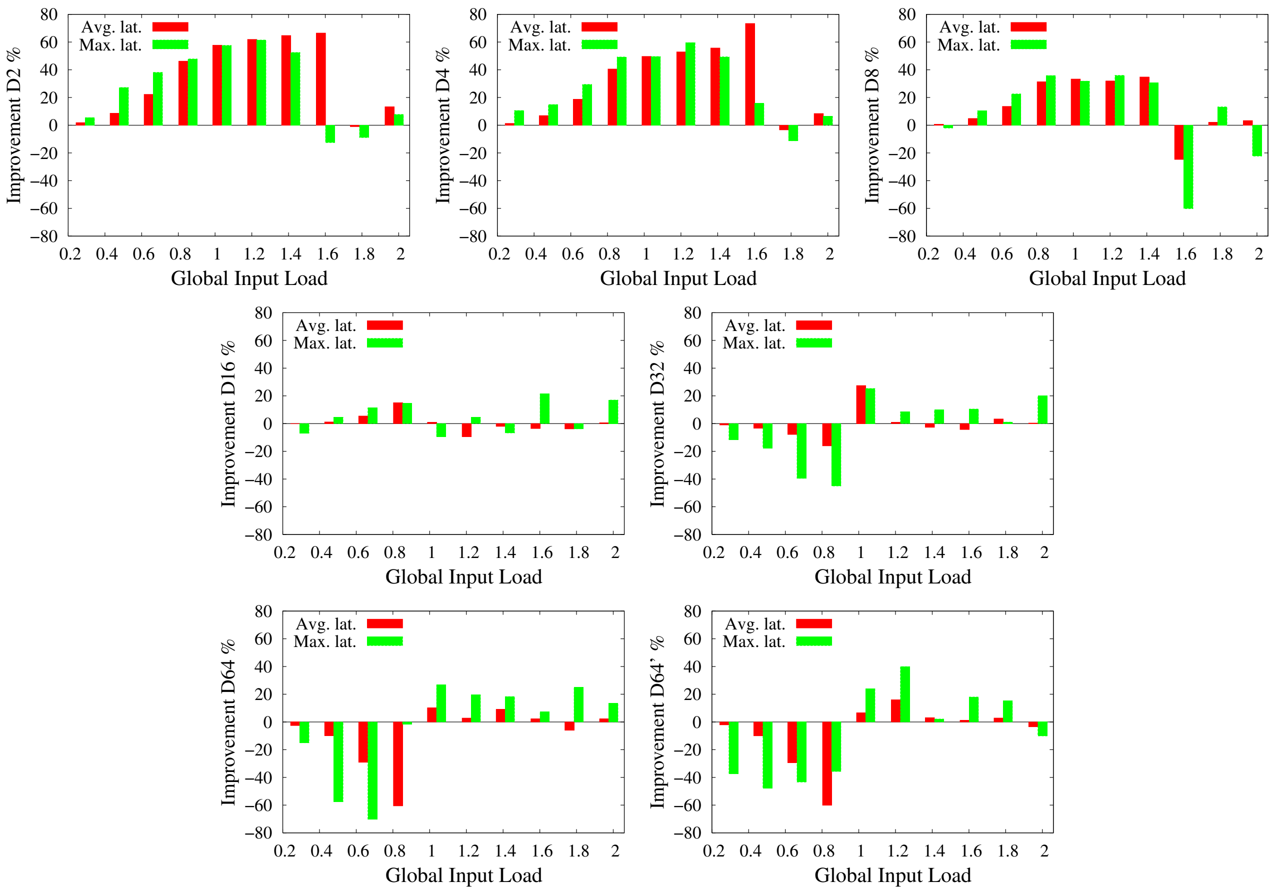

Figure 5. Latency improvement of the DTable scheduler over the DRR scheduler.

reason why Figure 5 shows that the percentage of improvement for those points becomes zero or does not stabilize in a clear value.

Note that when using the DRR algorithm, all the VCs obtain a similar latency performance until a VC reaches the point when its load begins to outstrip its throughput. In that point, the latency of that VC grows very fast and obtains a different latency performance. This happens for all the VCs as load grows. However, when using the DTable scheduler, all the VCs, including those with the same bandwidth assignment, obtain a different latency performance depending on the separation between any consecutive pair of their table entries. The smaller the distance, the better latency performance they obtain.

Note that the relevant load level interval in order to compare both schedulers is from the point where the load starts to be high until the point where the VC saturates. Analyzing this load interval, Figure 5 shows that the DTable scheduler provides a quite better average and maximum latency for the D2, D4, and D8 VCs than the DRR scheduler. It provides a slightly better latency for the D16 VC and a worse latency peformance for the D32, D64, and D64' VCs.

These different latency performance behaviors are explained by the fact that the maximum time that a packet at the head of a VC queue is going to wait until being transmitted is different depending on the scheduler algorithm. In the case of the DTable scheduler, we can control this time by controlling the maximum separation between any consecutive pair of entries assigned to the same VC. In the case of the DRR algorithm, the latency performance depends more on the frame length than on the quantum that each VC has been assigned. This is because when the quantum for a VC has been expended sending packets, all the frame must be cycled through before sending more packets of the same VC.

Summing up, the DTable and the DRR scheduling algorithms provide the same throughput performance, but a different latency performance. The DTable scheduler, which has a computational complexity just slightly higher than the DRR algorithm, provides the most preferential VCs (those which have been assigned a shorter distance between any consecutive pair of entries) with a better latency performance than the DRR algorithm. However, it provides the least preferential VCs with a worse latency than the DRR algorithm. These are good news because the DTable scheduler is providing each traffic class with a different treatment according to its requirements.

4 Conclusions and future work

The main problem of the DRR, which is known to have a very simple computational complexity, is that its delay depends on the frame length. Therefore, if the frame is very long the latency would be very bad. Moreover, we cannot differentiate the flows taking into account their latency requirements because all the flows obtain a similar latency performance.

We have proposed a new scheduling algorithm, the DTable scheduler, that solves in a high degree these DRR problems with only a slightly higher computational complexity than the DRR algorithm. With the DTable scheduler we can define several categories of traffic, with not only different bandwidth requirements but also different latency requirements. In this paper we have compared by similation the performance of both schedulers. Simulation results show that the DTable scheduler provides the most preferential flows with a better latency performance than the DRR algorithm and the least preferential VCs with a worse latency than the DRR algorithm. Therefore, with the DTable scheduler, we can provide a different level of latency performance to the VCs, priorizing those VCs with higher latency requirements, which is not possible with the DRR algorithm.

In this paper we have studied the computational and implementation complexity of the DRR and DTable schedulers 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 required to implement the schedulers, and the arbitration time that they would require.

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 Transactions on Parallel and Distributed Systems*, 15(9):810–823, Sept. 2004.

[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] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter. Interconnect-aware coherence protocols for chip multiprocessors. In *ISCA*, pages 339–351. IEEE Computer Society, 2006.

[5] A. C. et al. Supplemental information for the new definition of EF PHB (Expedited Forwarding Per-Hop-Behavior). RFC 3247, Mar. 2002.

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

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

[8] R. Martínez, F. Alfaro, and J. Sánchez. Decoupling the bandwidth and latency bounding for table-based schedulers. *International Conference on Parallel Procesing (ICPP)*, Aug. 2006.

[9] R. Martínez, F. Alfaro, and J. Sánchez. Providing Quality of Service over Advanced Switching. *International Conference on Parallel and Distributed Systems (ICPADS)*, July 2006.

[10] P. L. Montessoro and D. Pierattoni. Advanced research issues for tomorrow's multimedia networks. In *International Symposium on Information Technology (ITCC)*, 2001.

[11] K. I. Park. *QoS in Packet Networks*. Springer, 2005.

[12] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM*, pages 231–242, 1995.

[13] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 1998.