University of Castilla-La Mancha

A publication of the **Computing Systems Department**

Improving the flexibility of the Deficit Table scheduler

by

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

Technical Report **#DIAB-06-06-1** June 2006

This work has been submitted to the
International Conference on High Performance Computing 2006 (HiPC'06)

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

Contents

Improving the flexibility of the Deficit Table scheduler[1]

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

Departamento de Sistemas Informáticos

Universidad de Castilla-La Mancha

02071 - Albacete, Spain

{raulmm, falfaro, jsanchez}@info-ab.uclm.es

[1] This work was partly supported by the Spanish CICYT under Grant TIC2003-08154-C06-02, by the Junta de Comunidades de Castilla-La Mancha under Grant PBC-05-005-1, and by the Spanish State Secretariat of Education and Universities under FPU grant.

Abstract

A key component for networks with Quality of Service (QoS) support is the egress link scheduling algorithm. The table-based schedulers are simple to implement and can offer good latency bounds. Some of the latest proposals of network technologies, like Advanced Switching and InfiniBand, define in their specifications one of these schedulers. However, these 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 that partially decouples the bandwidth and latency assignments.

In this paper we propose a method to improve the flexibility of the decoupling methodology. Moreover, we compare the latency performance of this strategy with two well-known scheduling algorithms: the Self-Clocked Weighted Fair Queuing (SCFQ) and the Deficit Round Robin (DRR) algorithms.

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 [11]. The best-effort service model, though suitable for the first type of applications, is not so for applications of the other type [12]. 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. 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 end-to-end delay and simplicity. The end-to-end delay (also called latency) is defined as the sum of the transmission delay, the propagation delay, and the queuing delay experienced at each network node. The last component is by far the most significant. In some applications, if a packet suffers a latency higher than a certain value, the packet information can be worthless for the application. Moreover, a larger delay bound implies increased burstiness of the session at the output of the scheduler, thus increasing the buffering needed at the switches to avoid packet losses [15]. Thus, a good scheduling algorithm should guarantee acceptable queuing delay. Moreover, in order to achieve a good performance, the processing overheads must be some orders of magnitude smaller than the average packet transmission time. Thus, when the switching node operates at high speed, a simple scheduling algorithm is mandatory.

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 [15]. 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 [13] has been proposed. In the DRR algorithm, a list of flow[1] *quantums* is visited sequentially, each quantum indicating the amount of data that can be transmitted from the flow in question. The sum of all the quantums is called the frame length. The main problem of this algorithm is that its delay and fairness depend on the frame length. Depending on the situation, the frame can be very long, and thus, the latency and fairness would be very bad. On the other hand, in the table-based schedulers instead of serving packets of a flow in a single

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

visit per frame, the service is distributed throughout the entire frame. This approach is followed in [3] and in two of the last high performance network interconnection proposals: Advanced Switching [1] and InfiniBand [6]. These table-based schedulers can provide a good latency performance with a low computational complexity [3, 10, 2]. However, these schedulers do not work properly with variable packet sizes and face the problem of bounding the bandwidth and latency assignments [10, 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 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. As we stated in [8], the latency performance of the DTable scheduler depends on the maximum amount of data that is allowed to be transmitted per table entry. The more information is allowed to be transmitted the worse latency performance we get. Therefore, this maximum amount of data should be kept as small as possible. However, one of the parameters that our configuration methodology uses to increase the decoupling between the bandwidth and the latency assignments is indeed this value.

In this paper we propose and evaluate a method to increase the decoupling between the bandwidth and the latency assignments without increasing too much the maximum amount of data that is allowed to be transmitted per table entry. Moreover, we compare the latency performance of the DTable scheduler with the latency performance of two well-known scheduling algorithms: the Self-Clocked Weighted Fair Queuing (SCFQ) and the DRR algorithms. We have chosen the SCFQ algorithm as an example of "sorted-priority" algorithm and the DRR algorithm as one of the simplest scheduling mechanism proposed in the literature.

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. In Section 3, we propose a method to improve the latency performance of the DTable scheduler. Details on the experimental platform and the performance evaluation are presented in Section 4. Finally, some conclusions are given.

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. 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 throughout the entire frame. For this, an arbitration table is maintained.

Each table entry has associated a flow identifier and an *entry weight*. The *entry weight* is the amount of information, in flow control credits, that each table entry allows to be transmitted. Moreover, each flow has assigned a *deficit counter* that works in the same way as in the DRR scheduler. When scheduling is needed, the table is cycled through sequentially and a certain number of packets is transmitted from the flow indicated by the current table identifier.

We define the DTable scheduler taking into account a link-level flow control mechanism in the network, as in the case of AS or InfiniBand. Thus, if the credits for a given flow have been exhausted, the scheduler treats the corresponding queue as if it were empty. Thus, a flow is considered active when it stores at least one packet and there are enough credits to transmit the packet that is at the head of the flow queue. Specifically, this scheduler works in the following way:

- Table entries are cycled through up to an entry assigned to an active flow is found. We will call this flow the *selected flow*.
- 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 entry weight.
- Packets belonging to the selected flow are transmitted. The accumulated weight is reduced after sending each packet in an amount equal to the number of flow control credits required by the transmitted packet.
- The next table entry is selected when any of the following conditions occurs:
  - There are no more packets from the selected flow or there are not enough flow control credits for transmitting the packet that is at the head of the flow queue. In that case, the flow becomes inactive, and the deficit value for that flow becomes zero.
  - The accumulated weight is less than the size of the packet that is at the head of the queue. In that case, the deficit value becomes equal to the accumulated weight.

We set the minimum value that a table entry can have associated to 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 [13].

In [2], we explained how to configure table-based schedulers (in that case for InfiniBand) 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. By fixing this separation, it is possible to control the maximum latency of a network element crossing. 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. Therefore, given a maximum number of hops, we can control the maximum end-to-end latency.

In [8] we have also 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 set the maximum distance between any consecutive pair of entries assigned to a flow depending on its latency requirement. Moreover, 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 (a weight unit is equivalent to a flow control credit) from the bandwidth pool that the table entries of a flow have assigned fixes the bandwidth that the flow has actually assigned.

Note that $k, w \geq 1$ because, as stated before, the minimum weight that can be assigned to a table entry is the MTU. Note also that $k \leq w$ because the bandwidth pool cannot be larger than the theoretical maximum weight among all the entries ($N \times M$).

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 sizes and considers the possibility of a link-level flow control mechanism. Moreover, with our configuration methodology we can provide a flow with latency and bandwidth requirements in a partially independent way.

3 Improving the latency performance of the DTable scheduler

As stated before, using the DTable scheduler and our methodology, we can assign each flow a bandwidth between a minimum and a maximum that depends on the number of table entries and the two decoupling table parameters. 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 that can be assigned to a flow and to increase the maximum bandwidth assignable in order to be as flexible as possible. In order to be able to assign a small amount of bandwidth the k parameter must be small. However, the smaller 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.

Table 1 shows two different scenarios, each one with a different pair of values for the w and k parameters: DTable4 ($k = 2$, $w = 4$) and DTable8 ($k = 4$, $w = 8$). Note that we refer the different DTable scenarios according to the w value used in each case. This table shows the minimum and maximum bandwidth that can be assigned to seven flows (referred to as Virtual Channels, VCs) with different number of table entries. As we can see, when we increase the k parameter, the minimum bandwidth decreases. However, to maintain the same maximum bandwidth in the two scenarios, we have had to increase the w parameter in the same proportion.

Table 1: Table configuration. $N = 64$, $MTU = 32$

			DTable4		**DTable8**	
			$k = 2$, $w = 4$		$k = 4$, $w = 8$	
VC	**#entries**	**%entries**	$min\phi_i$	$max\phi_i$	$min\phi_i$	$max\phi_i$
D2	32	50	25	100	12.5	100
D4	16	25	12.5	50	6.25	50
D8	8	12.5	6.25	25	3.125	25
D16	4	6.25	3.125	12.5	1.5625	12.5
D32	2	3.125	1.5625	6.25	0.78125	6.25
D64	1	1.5625	0.78125	3.125	0.390625	3.125
D64'	1	1.5625	0.78125	3.125	0.390625	3.125
Total	64	100	50	200	25	200

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. Secondly, 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.

Our objective is to have a good flexibility when assigning the bandwidth to the flows but without increasing too much the w parameter. In order to achieve this we propose to use different MTUs for the different flows, instead of considering the general network MTU that the technology fixes for all the flows. Note that this means that some flows are going to have a specific MTU smaller than the general MTU. The use of different MTUs for different flows can be done at the communication library level.

The advantage of having a flow with a specific MTU smaller than the general MTU is that we can assign a table entry a minimum weight equal to the new MTU. When we use the general MTU for all the flows we cannot do this. As stated before, in this case, the general MTU 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. Being able to assign the table entries of a flow with a weight smaller than the general MTU allows to decrease the minimum bandwidth that can be assigned to that flow. If the i^{th} flow uses a specific MTU of size MTU_i, the maximum bandwidth that can be assigned to that flow is the same, but the minimum bandwidth depends on the proportion between the specific MTU and the general MTU values:

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

Note that varying the w and k parameters affect the minimum and maximum bandwidth that can be assigned to all the flows. However, assigning a specific MTU to a flow only affects that flow minimum bandwidth.

Note that with this method we can achieve small minimum bandwidths with a low value for the k parameter. Note also that now k can be even lower than 1. This allows to use a small w and still getting big maximum bandwidths. As stated before this decreases the hardware requirements and improves the latency performance.

Table 2 shows two different scenarios, each one with a different pair of values for the w and k parameters: DTable1 ($k = 0.5$, $w = 1$) and DTable2 ($k = 1$, $w = 2$). This table shows the specific MTU per flow and the minimum and maximum bandwidth that can be assigned to the flows. If we compare these values with the values in Table 1, we can see that now we can assign a small amount of bandwidth to those flows with lots of entries with a small w parameter.

Table 2: Table configuration with different MTUs. $N = 64$, $MTU = 32$

				DTable1		**DTable2**	
				$k = 0.5$, $w = 1$		$k = 1$, $w = 2$	
VC	**#entries**	**%entries**	MTU_i	$min\phi_i$	$max\phi_i$	$min\phi_i$	$max\phi_i$
D2	32	50	$MTU/32$	3.125	100	1.5625	100
D4	16	25	$MTU/32$	1.5625	50	0.78125	50
D8	8	12.5	$MTU/16$	1.5625	25	0.78125	25
D16	4	6.25	$MTU/8$	1.5625	12.5	0.78125	12.5
D32	2	3.125	$MTU/4$	1.5625	6.25	0.78125	6.25
D64	1	1.5625	$MTU/2$	1.5625	3.125	0.78125	3.125
D64'	1	1.5625	MTU	3.125	3.125	1.5625	3.125
Total	64	100		14.0625	200	7	200

When a message from a given flow arrives at the network interface, if its size is greater than its specific MTU, the message is splitted in several packets of a maximum size given by the specific MTU of the flow, as can be seen in Figure 1. A possible disadvantage of assigning specific MTUs smaller than the general MTU could be that the bandwidth and latency overhead of fragmenting the original message in several packets could probably affect performance of the flows. However, most restrictive latency flows (for example network control or voice traffic) usually present low bandwidth requirements, and small packet size. For example, in [16] several payload values for voice codec algorithms are shown. These values range from 20 bytes to 160 bytes. In that way, if we fix a small MTU for these flows, no fragmentation will be usually necessary because, in fact, the packets of those flows are already smaller than the new MTU. Therefore, the cornerstone of this proposal is to tune the specific MTU of each flow according to the specific characteristics of the flows.

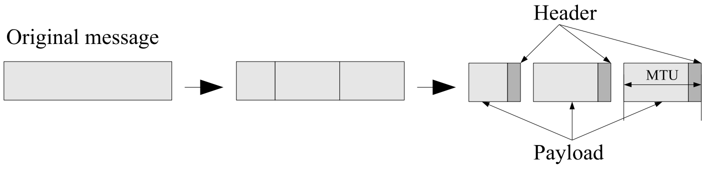


Figure 1: Process of message fragmentation into packets.

In the performance evaluation section we are going to use the same kind of traffic (with the same average packet size) for all the flows in order to make a fair comparison. Moreover we are going to assign smaller specific MTUs to those flows with more table entries in order to decrease the minimum bandwidth that can be assigned. Therefore, results are going to show the negative effect of an excessive packetization.

4 Performance evaluation

In this section, we evaluate the latency performance of the DTable scheduler. We compare the performance of the different scenarios with a different w parameter showed in the previous section (DTable1, DTable2, DTable4, and DTable8) and the SCFQ and DRR schedulers. For this purpose, we have developed a detailed simulator that allows us to model the network at the register transfer level, following the Advanced Switching (AS) specification. Note, however, that our proposals can be applied to any interconnection network technology.

4.1 The SCFQ and DRR algorithms

The Self-Clocked Weighted Fair Queuing (SCFQ) algorithm [5] is a variant of the Weighted Fair Queuing (WFQ) mechanism [4] which has a lower computational complexity. It defines fair queueing in a self-contained manner and avoids using a hypothetical queueing system as reference to determine the fair order of services. Instead, it uses a virtual time function which depends on the progress of the work in the actual packet-based queueing system. We have chosen the SCFQ algorithm as an example of “sorted-priority" algorithm because it is an implementable, but still quit complex, possibility with only slightly worse latency performance than the WFQ algorithm.

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 start. 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. We have chosen 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 [9] for being used in networks with a link-level flow control network like AS.

4.2 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). 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.

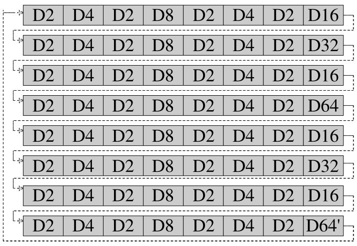
In our tests, the link bandwidth is 2.5 Gb/s but, with the AS 8b/10b encoding scheme, the maximum effective bandwidth for data traffic is only 2 Gb/s. We are assuming some internal speed-up (x1.5) for the crossbar, as is usually the case in most commercial switches. AS gives us the freedom to use any 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 [14].

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×MTU) per VC at the network interfaces and 16384 bytes (8×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. Regarding the latency statistics, a packet is considered injected when it is stored in the network interface.

4.3 Simulated scenario and scheduler configuration

We have defined 7 VCs with different distances between any pair of 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 stricter latency requirements than D4, D4 than D8, and so on. A table of 64 entries has been used in the simulations. Figure 2 shows the distribution of the VC identifiers in the arbitration table that we have set for this evaluation.



D2	D4	D2	D8	D2	D4	D2	D16
D2	D4	D2	D8	D2	D4	D2	D32
D2	D4	D2	D8	D2	D4	D2	D16
D2	D4	D2	D8	D2	D4	D2	D64
D2	D4	D2	D8	D2	D4	D2	D16
D2	D4	D2	D8	D2	D4	D2	D32
D2	D4	D2	D8	D2	D4	D2	D16
D2	D4	D2	D8	D2	D4	D2	D64'

Figure 2: Arbitration table.

As stated before, we are going to compare the performance of the DTable scheduler using different values for the w parameter (DTable1, DTable2, DTable4, and DTable8) with the performance of the SCFQ and DRR algorithms. Note that all the scenarios have the same maximum bandwidth values, differing only in the minimum bandwidth values (see Tables 1 and 2). We have assigned to each VC the amount of bandwidth ϕ_i shown in Table 3. This table also 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 the DTable1 case, the bandwidth pool is 1024 credits ($k = 0.5$), and thus, in order to assign 25% of bandwidth to this VC, 256 credits must be assigned to it. Therefore, 8 credits have been assigned to each one of its 32 table entries. The configuration of the SCFQ and the DRR schedulers is shown in Table 4.

We are going to inject an increasing amount of traffic of all the VCs and study the throughput and latency performance of the different possibilities 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. Figure 3 shows the normalized injection rate of the aggregated of flows associated with each VC.

Table 3: Bandwidth configuration of the DTable scheduler scenarios.

		DTable1		**DTable2**		**DTable4**		**DTable8**	
VC	ϕ_i	E. w.	T. w.	E. w.	T. w.	E. w.	T. w.	E. w.	T. w.
D2	25	8	256	16	512	32	1024	64	2048
D4	25	16	256	32	512	64	1024	128	2048
D8	25	32	256	64	512	128	1024	256	2048
D16	12.5	32	128	64	256	128	512	256	1024
D32	6.25	32	64	64	128	128	256	256	512
D64	3.125	32	32	64	64	128	128	256	256
D64'	3.125	32	32	64	64	128	128	256	256
Total	100		1024		2048		4096		8196

Table 4: Bandwidth configuration of SCFQ and DRR schedulers.

		SCFQ	**DRR**
VC	ϕ_i	Weight	Quantum
D2	25	0.25	256
D4	25	0.25	256
D8	25	0.25	256
D16	12.5	0.125	128
D32	6.25	0.625	64
D64	3.125	0.3125	32
D64'	3.125	0.3125	32
Total	100	1	1024

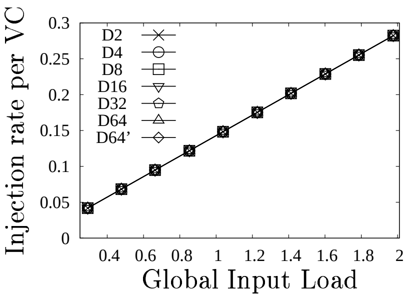
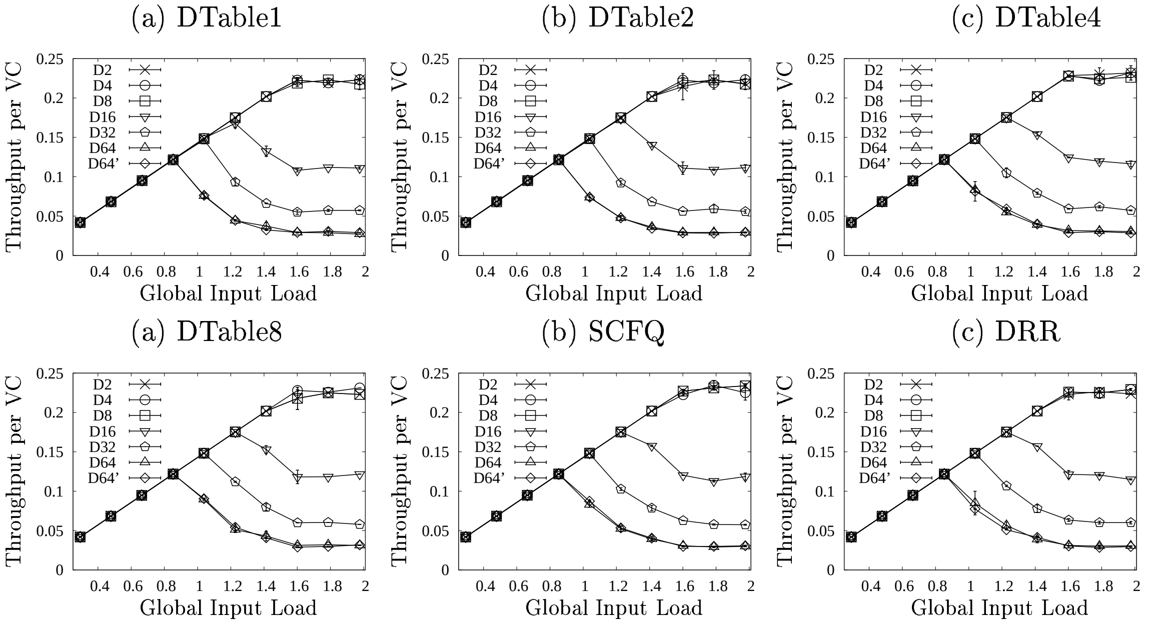


Figure 3: Normalized injection rate per VC.

4.4 Simulation results

The figures of this section 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 each flow. Note that in the DTable1 and DTable2 scenarios we use specific MTUs for the VCs that are smaller than the general MTU. Therefore, in these cases, a message can be splitted in several packets. In the rest of cases (DTable4, DTable8, SCFQ, and DRR) a mesage is going to be transmitted in only one packet. Note that in the DTable1 and DTable2 scenarios we consider the latency of the message as a whole. This means that, in these cases, to calculate the latency of a message we consider the time since we inject the first packet of a message into the network interface up to the last packet of the message arrives at its destination. Note that this may suppose a certain overhead. No statistics on packet loss are given because, as has been said, we assume a credit-based flow control mechanism to avoid dropping packets. We obtain statistics per VC aggregating the throughput of all the flows of the same VC, obtaining the average value of the average latency, and the maximum latency of all the flows. Note that the maximum latency shows the behavior of the flow with the worst performance.

Figure 4 shows the normalized throughput results per VC of the different schedulers. As we can see, when the load is low, all the VCs obtain the bandwidth they inject. However, when the load is high (around 95%) the VCs do not yield a corresponding result, obtaining a bandwidth proportional to their assigned bandwidth. Note that the VCs do not obtain all the bandwidth that they were supposed to have assigned because the network is not able to provide 100% throughput. Note also that all the schedulers provide a similar throughput performance.


(a) DTable1 (b) DTable2 (c) DTable4

(a) DTable8 (b) SCFQ (c) DRR

Figure 4: Normalized throughput per VC of the different scheduling scenarios.

Figure 5 shows the average latency performance. 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 scheduler configuration. However, when the load of the VC begins to outstrip its throughput, the latency of the scheduler 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.

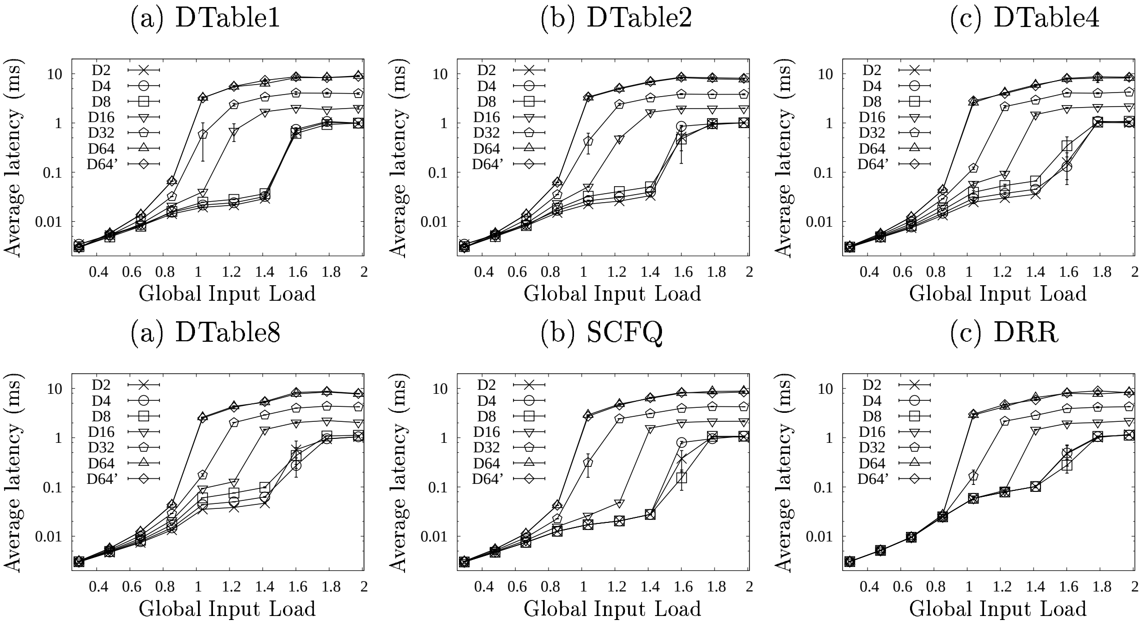

(a) DTable1 (b) DTable2 (c) DTable4
(a) DTable8 (b) SCFQ (c) DRR

Figure 5: Average latency per VC of the different scheduling scenarios.

Note that when using the SCFQ algorithm those VCs that have assigned the same bandwidth (in this case the D2, D4, and D8 VCs, and the D64 and D64' VCs) obtain the same latency performance. In the case of 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. 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.

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 SCFQ algorithm, this time is proportional to the assigned bandwidth. 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 this way, we provide some VCs with a better latency performance and other VCs with a worse latency performance. 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.

Finally, Figure 6 shows the percentage of improvement on average latency of the SCFQ algorithm over the four possibilities of the DTable scheduler and the DRR algorithm. Analyzing this figure we can compare the DTable performance comparing it not only with the SCFQ scheduler, but also with the DRR scheduler. Moreover, we can compare the difference between using the same general MTU for all the VCs or using specific MTUs for the VCs. This figure shows that in general, the SCFQ algorithm provides a better latency performance than the DTable scheduler in all the cases. However, this algorithm is the most complex. The DRR provides a worse performance for the most latency restrictive VCs and better for the less latency restrictive VCs than the DTable scheduler. This is because with the DTable scheduler we can provide a different level of latency performance to the VCs, priorizing those VCs with higher latency requirements. This is not possible with the DRR algorithm. Regarding the different scenarios of the DTable scheduler we can see that DTable1 provides a better latency performance than DTable2, and DTable4 than DTable8. This is because in general, the higher the value of the w parameter, the worse the latency performance. However, the effect of splitting the messages in several packets must also be taken into account.

Table 5 shows the bandwidth overhead per VC that is produced by using smaller specific MTUs than the general MTU. This packetization also has effect on the latency of the message. Note that each packet must be processed by the network elements (routing, scheduling, etc.). Moreover, if a table entry allows us to transmit a small number of packets of the new MTU size, it is possible that in order of transmitting all the packets belonging to the same message more than one table entry must be used, and thus, the latency increases. Figure 6 shows clearly the first effect when considering a low load for the D2 and D4 VCs. In this case, the latency of the DTable1 and Dtable2 scenarios is rather worse than for the others cases. We obtain a better latency for DTable1 and Dtable2 than DTable4 and Dtable8 when the latency is high for the D2, D4, and D8 VCs. However, for the rest of VCs we obtain a worse latency because the specific MTUs are higher and the weight assigned to the table entries lower. Note that this bad effect of the excessive packetization would disappear in a real case if the MTU of each VC is selected on the basis of the specific average message size of the flows that would use the VC.

Summing up, the DTable scheduler, which has a quite good computational complexity, 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. Our proposal of using different specific MTUs increments the flexibility of our decoupling methodology without the need of increasing the w parameter too much. Note that increasing this parameter would entail more hardware requirements to store and process the table entries and a worse latency performance. However, the excesive packetization of the messages may produce a negative effect on the performance of the flows. Therefore, the specific MTUs should be assigned taking into account the characteristics of the traffic, specifically, the size of the packets.

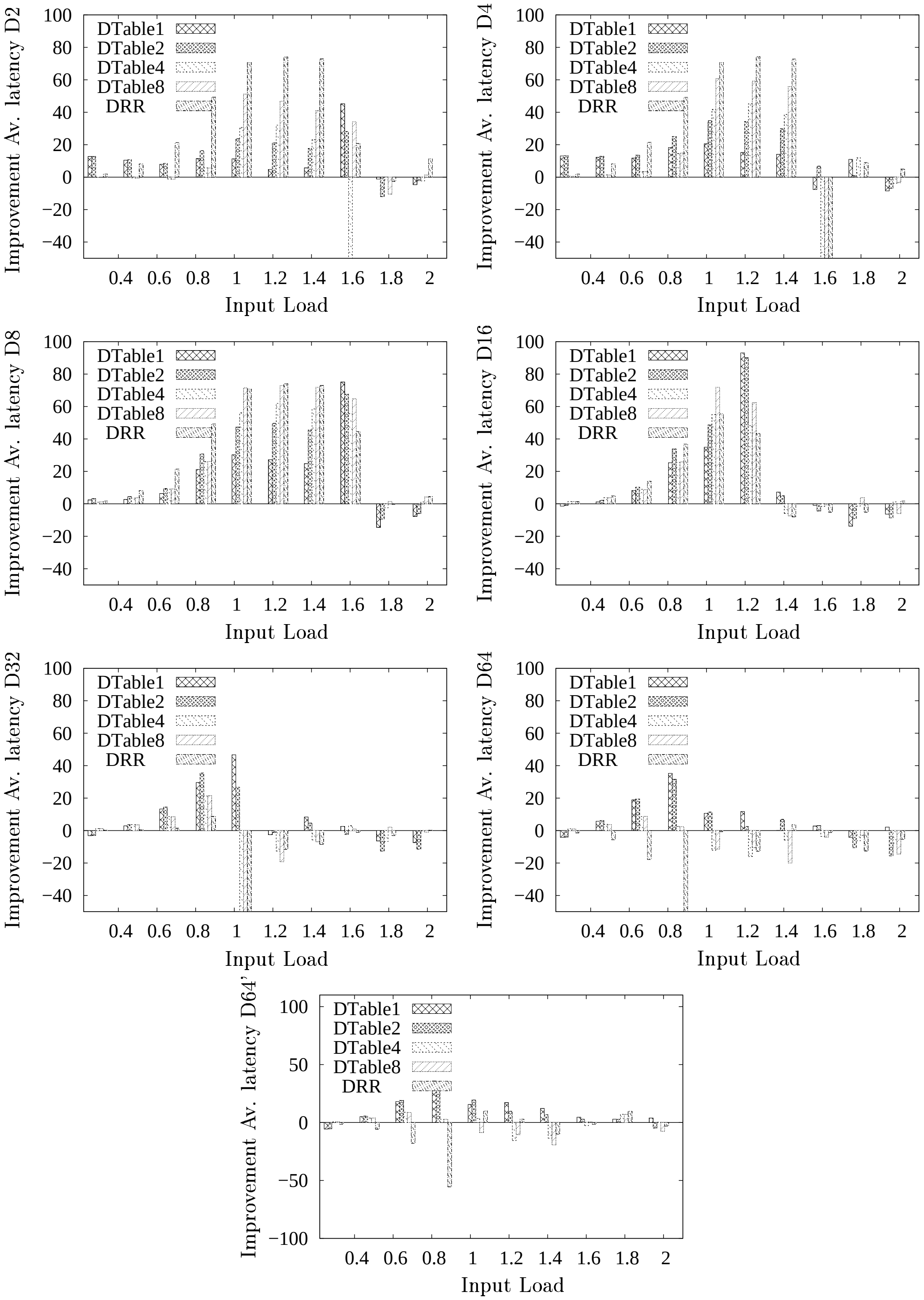


Figure 6: Average latency improvement of the SCFQ algorithm over the other schedulers considered.

5 Conclusions

A key component for networks with QoS support is the output scheduling algorithm, which selects the next packet to be transmitted. An ideal scheduling algorithm should

Table 5: Packetization bandwidth overhead per VC with average packet size of 176 bytes.

VC	MTU_i (bytes)	Overhead (%)
D2	64	11.7
D4	64	11.7
D8	128	3.82
D16	256	1
D32	512	0.4
D64	1024	0.06
D64'	2048	0

satisfy two main properties: good end-to-end delay and simplicity. Table-based schedulers try to address these two characteristics. However, they have several problems that we try to solve with a new table-based scheduler, the DTable scheduler, and several proposals to configure it.

The DTable scheduler is a simple algorithm that properly configured may provide the flows with different levels of latency performance. Moreover, given a flow or aggregated of flows with some latency requirements, we can assign a certain amount of traffic to that flow in a flexible way. The decoupling methodology that allows us to do this relies on two table configuration parameters. One of these parameters, the w parameter, determines the maximum weight that can be assigned to a single table entry, and thus, the maximum data that can be transmitted per table entry. The other, the k parameter, determines the maximum weight that can be distributed among all the table entries.

In this paper we have proposed a method to increase this flexibility. This method consists in using different MTUs for the different flow. This allows us to employ smaller values for the k parameter, and thus, for the w parameter. This is quite positive because a high value for the w parameter entails higher hardware requirements and worse latency performance. However, the specific flow MTU must be assigned taking into account the characteristics of the traffic flow. A too small specific MTU may decrease the latency performance of the flow.

In a real case the w parameter is probably going to be fixed by the network technology.The network manager should then choose an appropiate value for the k parameter and the specific MTUs. To do this the characteristics of the traffic and the proportion of each kind of traffic must be taken into account. Note that different flows can exhibit very different message sizes. As future work we are focusing our attention on applying our proposals to a multimedia environment with different kind of traffics.

Bibliography

[1] Advanced Switching Interconnect Special Interest Group. *Advanced Switching core architecture specification. Revision 1.0*, December 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, September 2004.

[3] H. M. Chaskar and U. Madhow. Fair scheduling with tunable latency: A round-robin approach. *IEEE/ACM Transactions on Networking*, 11(4):592–601, 2003.

[4] A. Demers, S. Keshav, and S. Shenker. Analysis and simulations of a fair queuing algorithm. In *SIGCOMM*, 1989.

[5] S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM*, 1994.

[6] InfiniBand Trade Association. *InfiniBand architecture specification volume 1. Release 1.0*, October 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.J. Alfaro, and J.L. Sánchez. Decoupling the bandwidth and latency bounding for table-based schedulers. *International Conference on Parallel Procesing (ICPP)*, August 2006.

[9] R. Martínez, F.J. Alfaro, and J.L. Sánchez. Implementing the advanced switching minimum bandwidth egress link scheduler. *IEEE International Symposium on Network Computing and Applications (IEEE NCA06)*, July 2006.

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

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

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

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

[14] StarGen. *StarGen's Merlin switch*, 2004. `http://www.stargen.com/products/merlin_switch.shtml`.

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

[16] A. Tyagi, J. K. Muppala, and H. de Meer. VoIP support on differentiated services using expedited forwarding. In *IEEE International Performance, Computing, and Communications Conference (IPCCC)*, February 2000.