University of Castilla-La Mancha

A publication of the **Department of Computer Science**

Decoupling the bandwidth and latency bounding for table-based schedulers

by

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

Technical Report **#DIAB-06-02-01** February 2006

This work has been submitted to the
International Conference on Parallel Processing (ICPP'2006)

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

Decoupling the bandwidth and latency bounding for table-based schedulers[1]

Raúl Martínez, Francisco J. Alfaro, José L. Sánchez
Departamento de Sistemas Informáticos
Escuela Politécnica Superior
Universidad de Castilla-La Mancha
02071- Albacete, Spain
{Email: 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

The provision of Quality of Service (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.

Some of the latest network technology proposals define scheduling algorithms that use an arbitration table to select the next packet to be transmitted. These table-based schedulers are simple to implement and can offer good latency bounds. However, they face the problem of bounding the bandwidth and latency assignments. Moreover, this kind of scheduler does not work properly with variable packet sizes.

In this paper, we propose a methodology to decouple the bandwidth and latency assignments. We also propose a new table-based scheduler, which we call Deficit Table (DTable), that works properly with variable packet sizes.

Keywords: Quality of Service (QoS), scheduling algorithms, bandwidth and latency requirements, interconnection networks.

1 Introduction

Current high performance packet networks are required to carry traffic of different applications. Some of these applications, like real-time video or telephony, require pre-specified service guarantees. Therefore, multiservice 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 possess the following properties:

- Fairness: The fairness of a scheduling algorithm is measured as the maximum difference between the bandwidth allocation provided by the scheduling algorithm and an ideal fair queuing scheme.
- Good End-to-End Delay: The end-to-end delay 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. Thus, a good scheduling algorithm should guarantee acceptable queuing delay.
- Simplicity: 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. Among the three, the delay and implementation complexity are clearly the most important criteria for the selection of an algorithm for use in a real system. While the fairness property of the algorithm affects only the short-term distribution of service offered to the sessions sharing the link, 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 [1].

Several scheduling algorithms with different properties have been proposed. In [2], Chaskar and Madhow propose a category of scheduler called list-based Weighted Round Robin (WRR) [3], which has a simple implementation and can offer good latency bounds. In this generalization of the classical WRR discipline, a list of flow identifiers, called "service list", is maintained[1]. When scheduling is needed, the list is cycled through sequentially and a packet is transmitted from the flow indicated by the current list identifier. The same approximation is followed in two of the last high performance network interconnection proposals. In the Advanced Switching architecture [4], one of the schedulers defined in the specification is a virtual channel arbitration table scheduler. Moreover, the InfiniBand [5] scheduler also uses this kind of table but

[1]Sometimes, this list is also called table. In this paper both terms will be used alike.

adds to each entry a weight that indicates the amount of information that can be transmitted.

In this kind of table-based scheduler, we can control the latency of a flow by controling the maximum separation between any consecutive pair of entries assigned to that flow [6]. Therefore, we can provide flows with different QoS latency requirements by assigning different maximum distances. However, this way of assigning the entries of the arbitration table faces the problem of bounding the bandwidth and latency assignments [6]. If a maximum separation between any consecutive pair of entries of a flow is set, a certain number of table entries are being assigned, and hence a minimum bandwidth, to the flow in question. In this paper we propose a methodology to configure the arbitration table that partially decouples the bounding between bandwidth and latency assignments.

In the basic table-based scheduler approximation each table entry allows the transmission of a packet regardless of the packet size. Therefore, this kind of scheduler also presents the problem of not working properly with variable packet sizes, as is common in actual high performance networks. As we are going to show, the InfiniBand approximation solves this problem only in part. As far as we know, a table-based scheduler proposal that is able to handle properly variable packet sizes has not yet been proposed. In this paper we also propose a new table-based scheduling algorithm that works properly with variable packet sizes. We have called this algorithm *Deficit Table* scheduler or just DTable scheduler.

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 highlight the problems of the table-based schedulers, namely the bounding between the bandwidth and latency assignments, and the problem with variable packet sizes. In Section 4, we propose the DTable scheduler. In Section 5, we present our methodology to configure the arbitration table to decouple the bandwidth and latency assignments. Details on the experimental platform and the performance evaluation are presented in Section 6. Finally, some conclusions are given.

2 Scheduling algorithms

Many fair queuing scheduling algorithms have been proposed, among them, a family of algorithms, which we will refer to by the generic name of “sorted-priority" algorithms. This family of algorithms relies on a common-reference virtual clock, according to which arriving packets are stamped with a virtual time tag. The packets are then sorted on the basis of their increasing time tags: the smaller the time tag, the higher the priority of the corresponding packet to be transmitted.

The best known algorithm of this family is the Weighted Fair Queuing (WFQ) mechanism [7]. The WFQ algorithm is an approximation of the Generalized Processor Sharing (GPS) model [8]. GPS is a fair-queuing model based on a fluid model that provides perfect instant fairness in bandwidth allocation. This ideal model assumes

that several packets from different flows can be simultaneously transmitted. WFQ is a packet-by-packet algorithm that tries to emulate the GPS model by stamping each packet that arrives at the egress link with its departure time in a corresponding GPS system. The packets are then transmitted in an increasing order of time tag. This provides the best possible fairness and delay bounds. However, to calculate the time tags, which derive from the virtual clock, the emulation in real-time of an auxiliary GPS fluid system is required. This may make the WFQ algorithm unfeasible in high-speed interconnection technologies.

The Self-Clocked Weighted Fair Queuing (SCFQ) algorithm [9] is a variant of the WFQ algorithm which has a lower computational complexity. The SCFQ algorithm defines fair queuing in a self-contained manner and avoids using a hypothetical queuing system as reference to determine the fair order of services. This objective is accomplished by adopting a different notion of virtual time. Instead of linking virtual time to the work progress in the GPS system, it uses a virtual time function which depends on the progress of the work in the actual packet-based queuing system. This approach offers the advantage of removing the computation complexity associated with the calculation of the time tags in the WFQ algorithm. However, the price paid is in terms of the end-to-end delay bound, that grows linearly with the number of sessions sharing the egress link [1]. Thus, the worst-case delay of a session can no longer be controled just by controling its bandwidth assignment, as it is possible in WFQ.

Summing up, the sorted-priority family of algorithms suffers from two major problems. The first problem is that these algorithms require processing at line speeds for tag calculation and tag sorting. In other words, each time a packet arrives at a node, its time tag is calculated and the packet is inserted at the appropriate position in the ordered list of packets waiting for transmission. This means that these algorithms require at least the complexity of a search algorithm in the list of queued packets: $O(log(N))$, where N is the maximum number of packets at the queue, or if the buffers are not shared, $O(log(J))$, where J is the number of active flows. The second problem that may happen in the sorted-priority approach is that, since the time tag is an increasing function of the time and depends on a common-reference virtual clock, which in turns reflects the value of the time tag of previously served packets, the virtual clock cannot be reinitialized to zero until the system is completely empty and all the sessions are idle. 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 has been shown to exhibit self-similar patterns which lead to heavily tailed buffer occupancy distributions. Therefore, for practical implementation of these algorithms, very high-speed hardware needs to be designed to perform the sorting, and floating-point units must be involved in the computation of the time tags. This, of course, can be done, but at a great cost and with very limited scalability.

To avoid the complexity of the sorted-priority approach, the Deficit Round Robin (DRR) algorithm [10] has been proposed. The aim of DRR is to implement fair queuing and achieve practically acceptable complexity at the expense of other performance

metrics such as fairness and delay. The DRR algorithm is a variation of the Weighted Round Robin (WRR) algorithm [3] that works properly with variable packet sizes. In the WRR algorithm, a list of flow weights is visited sequentially, each weight indicating the number of packets from the flow in question that can be transmitted. The sum of all the weights, which is the maximum number of packets transmitted in each list cycle, is called the frame length. On the other hand, the DRR algorithm associates each queue with a deficit counter, which is set to zero at the start. The scheduler visits and serves a fixed amount of data (referred to as *quantum*) from each flow. When a packet is transmitted, the quantum is reduced by the packet size. For each flow, the scheduler transmits as many packets as its quantum allows. 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.

A well-known problem of the DRR algorithm (which is also common to other round-robin schedulers) is that the latency and fairness 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. According to the implementation proposed in [10], DRR exhibits $O(1)$ complexity provided that each flow is allocated a quantum no smaller than the Maximum Transfer Unit (MTU). As observed in [11], removing this hypothesis would entail operating at a complexity which can be as large as $O(N)$. This restriction affects not only the weight assigned to the smallest flow, but the rest of the flows in order to keep the proportions between them.

As stated before, Chaskar and Madhow [2] propose a category of scheduler called list-based WRR. 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 list identifier. 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. In [2], three ways of distributing the flow identifiers to conform the service list are proposed, resulting in three different schedulers with different characteristics. They show that the list-based WRR schemes can achieve the performance of the sorted-priority algorithms. Moreover, the proposed WRR-based schemes do not involve packet tag sorting, and hence, they have lower implementation complexity than WFQ-based schemes. These reasons make promising this kind of schedulers. The study of Chaskar and Madhow is performed with fixed packet size and they comment that weighted versions of these schemes could handle variable packet sizes. However, as far as we know, a table-based scheduler that is able to handle properly variable packet sizes has not yet been proposed.

Two recent network technology standards incorporate a scheduling algorithm similar to the list-based WRR schedulers: Advanced Switching (AS) [4] and InfiniBand [5]. These technologies use Virtual Channels (VCs) to aggregate flows with similar characteristics and the arbitration is made at a VC level. In both cases, the maximum number of unicast VCs that a port can implement is 16. AS defines a VC arbitration table scheduler which uses an arbitration table that works in the same way as the service list of the list-based WRR schedulers. This arbitration table can have 32, 64, 128, 256, 512, or 1024 entries. Each entry contains a VC identifier value. 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. On the other hand, InfiniBand defines a scheduler that uses two tables, one for scheduling packets from high-priority VCs and another one 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.

3 Problems of the table scheduler

As stated before, in [2], Chaskar and Madhow propose three list-based WRR schedulers. For a given set of flows the arbitration table of the three schedulers is going to have the same number of entries. Therefore, the difference between the three schedulers is in the way of distributing the flow identifiers among the table entries. These different forms of interleaving the flow identifiers result in different latency characteristics for the three schedulers.

In [6], 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 [6], it is shown (in that case for InfiniBand) that controling 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 in the head of a flow queue is going to wait until being transmitted. This explains the different latency properties of the list-based WRR schedulers.

When using the table-based schedulers in which one entry allows one packet to be transmitted, the way of assigning the entries of the table proposed in [6] 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 that way, 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. Moreover, we cannot assign more bandwidth to the flows than that provided for the number of table entries.

We have performed a simple test using a table scheduler with several aggregated flows, each one using a different VC. The flows present different latency requirements and thus a different distance in the arbitration table has been assigned to each VC. We have performed the test using an arbitration table of 64 entries. Table 1 shows the distance between any consecutive pair of the table entries of the flows and the number and percentage of entries that this entails. The simulated architecture is the same as that used for the performance evaluation in Section 6. The results obtained are shown in Figure 1. As can be seen, the flows inject data at the same rate but they receive a different throughput depending on the distance configuration of each flow.

Table 1: Arbitration table configuration

VC	**Distance**	**#entries**	**%entries**
D2	2	32	50
D4	4	16	25
D8	8	8	12.5
D16	16	4	6.25
D32	32	2	3.125
D64	64	1	1.5625
D64'	64	1	1.5625
	Total	64	100

Another problem of the table-based schedulers is that they do not work in a proper way with variable packet sizes. However, today networkt technologies usually use variable packet sizes. 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. Figures 2, 3, and 4 show the performance of various table-based schedulers when there are four aggregated traffic flows in the network. All these flows have the same data rate and the same number of assigned table entries (the same bandwidth reservation) but different packet size. The simulated architecture is the same as that used for the performance evaluation in Section 6.

Figure 2 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 flows obtain a very different bandwidth because each flow 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 InfiniBand's 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 3 shows the performance of a weighted

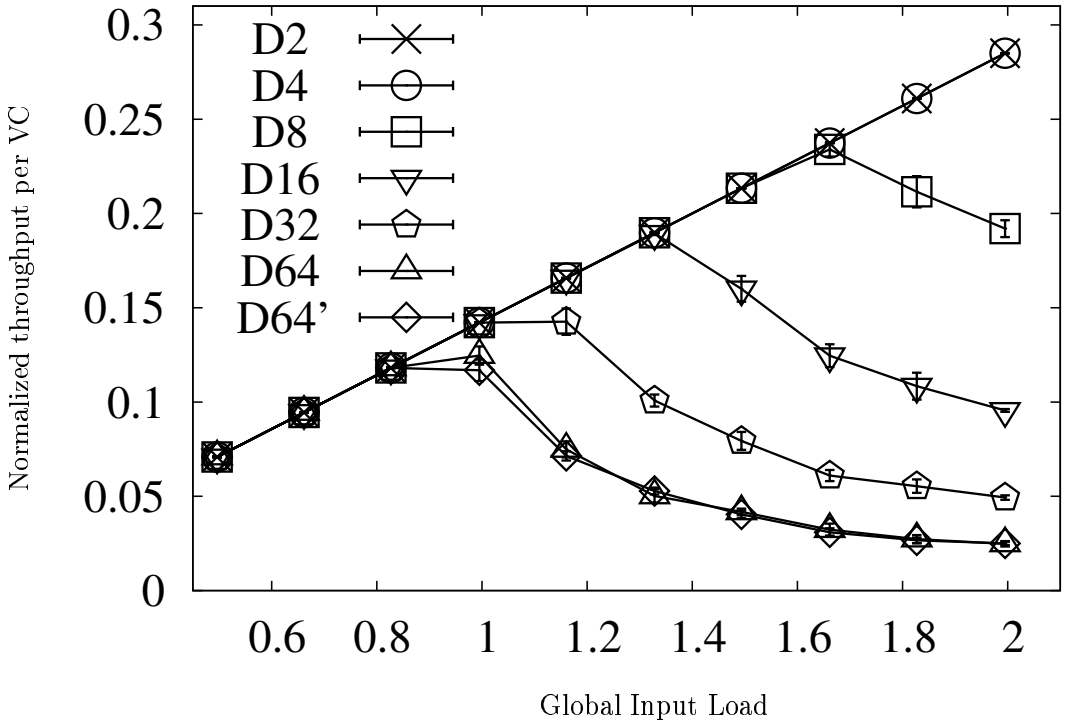

Figure 1: Performance of the basic table scheduler for flows with different distance configuration between any consecutive pair of table entries.

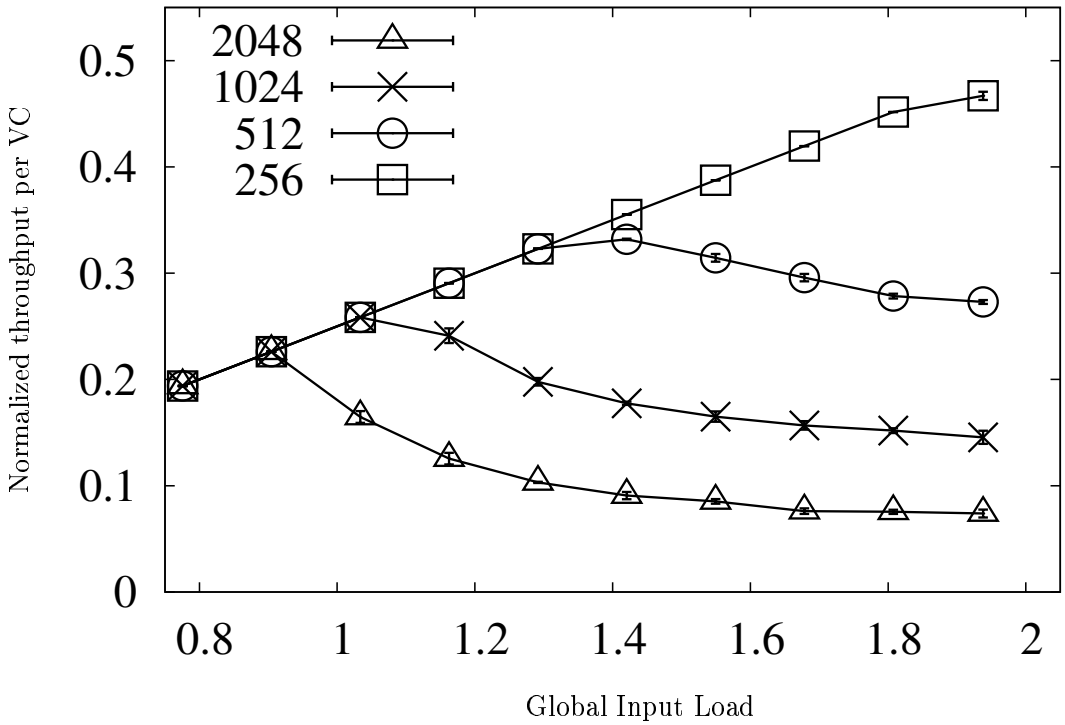

Figure 2: Performance of the basic table scheduler for flows with different packet size.

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

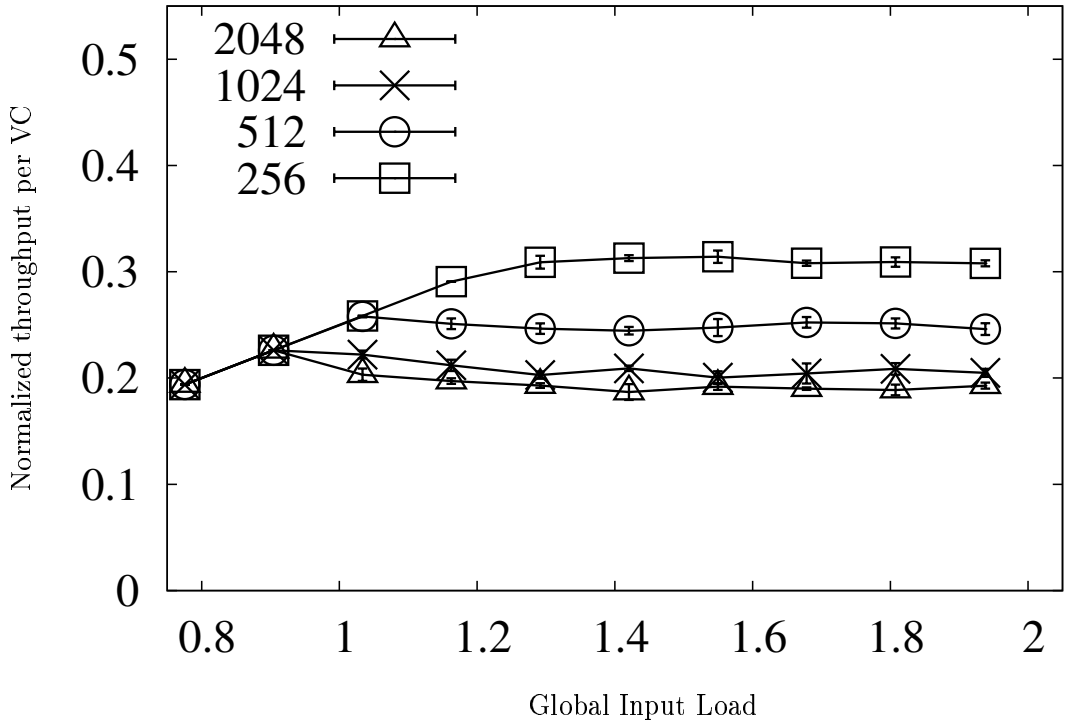

Figure 3: Performance of the weighted table scheduler for flows with different packet size.

basic table scheduler, but not an optimum performance.

4 The Deficit Table scheduler

In this section we propose a new table-based scheduling algorithm that is able to solve the above-mentioned problems. We have called this new algorithm Deficit Table scheduler, or just DTable scheduler, because it is a mix between the table-based schedulers presented before and the DRR algorithm. In the same manner that the list-based WRR schedulers distribute the service over the round-robin frame, the DTable scheduler distributes the quantum assigned to each flow among the table entries assigned to that flow. 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. If there is no flow control mechanism, the scheduler must consider that there are always enough credits to transmit a packet. Specifically, this scheduler works in the following way:

- Each table entry has associated a VC identifier and an *entry weight*. The *entry weight* is the amount of information, in flow control credits, that each entry allows to be transmitted.

- Each VC has assigned a *deficit counter*.

- A VC is active when it stores at least one packet and there are enough credits to transmit the packet that is at the head of the VC queue.

- Table entries are cycled through until an entry assigned to an active VC is found. We will call this VC the *selected VC*.

- 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 VC and the entry weight.

- Packets belonging to the selected VC 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 VC. In that case, the VC becomes inactive, and the deficit value for that VC becomes zero.
  - There are not enough flow control credits for transmitting the packet that is at the head of the VC queue. In that case, the VC becomes inactive, and the deficit value for that VC becomes zero.
  - The accumulated weight is less than the size of the packet that is at the head of the queue. The deficit value becomes equal to the accumulated weight.

We set the minimum value that a table entry can have associated to the MTU of the network. This is the smallest value that ensures that there will never be a need to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. Note that this is the same consideration as that made in the DRR algorithm [10]. Note also that the InfiniBand tables solve this problem by rounding up to a whole packet the remaining weight in a table entry.

Figure 4 shows, as in the previous cases, the performance of the DTable scheduler when four flows inject data at the same rate and have assigned the same number of table entries, but have different packet sizes. As can be seen, in this case all the flows obtain the same throughput. Therefore, the resulting DTable algorithm is a quite simple modification of the table scheduler that works properly with variable packet sizes. The memory requirements for this algorithm over the basic table scheduler are the memory needed to store the deficit counter for each VC and the weight associated to each table entry.

Summing up, we have proposed a table-based scheduler that considers the posibility of a link-level flow control mechanism and is able to deal properly with variable packet sizes.

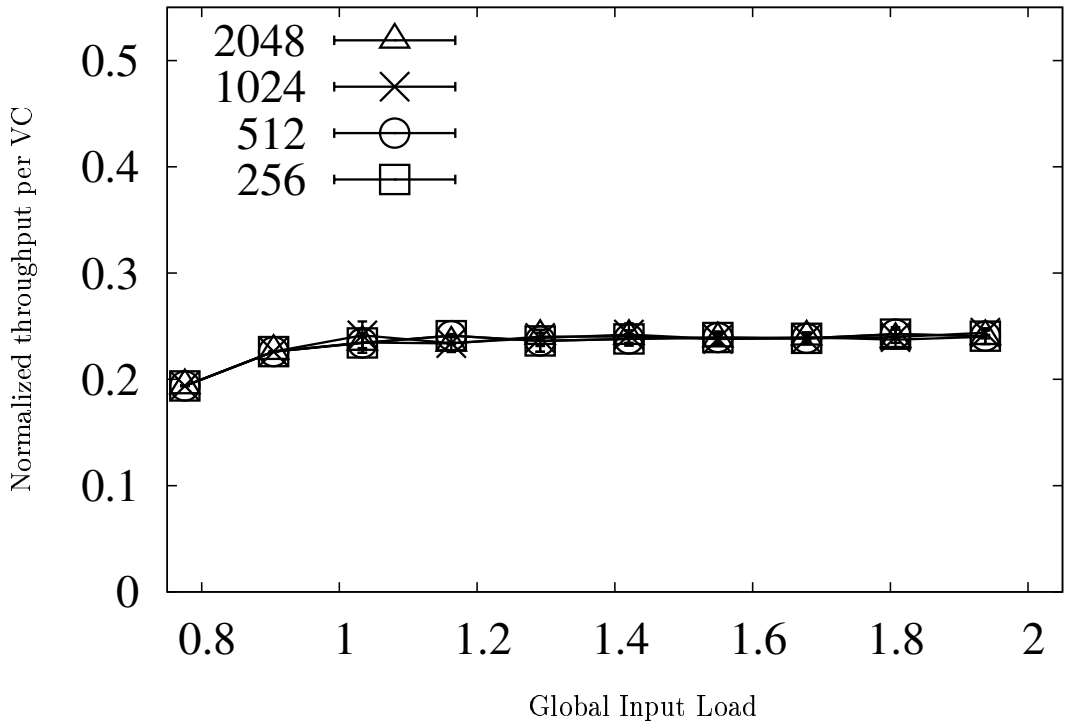

Figure 4: Performance of the DTable scheduler for flows with different packet size.

5 Decoupling the bandwidth assignment of the latency requirements

In this section we present a methodology to configure the DTable scheduler presented above to decouple, at least partially, the bounding between the bandwidth and latency assignments. This methodology is also applicable to a weighted table like the used by InfiniBand. However, in this case, it may not work properly with variable packet sizes. Our aim here is for the maximum distance between any consecutive pair of table entries for each flow, or aggregate of flows, to control the latency, and for the weights assigned to those entries likewise to control the assigned bandwidth. In that way, the latency and bandwidth allocation of a flow would be independent.

Supposing an arbitration table with N entries in a network with a certain MTU, and supposing the i^{th} flow has assigned n_i table entries, we would like to be able to assign to the i^{th} flow 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. Note that in the classical table scheduler, the number of entries (the proportion of entries over the total) fixes the bandwidth allocated to that flow. Table 2 shows all the involved parameters.

We define two decoupling table parameters: w and k. The w parameter determines the maximum weight M that can be assigned to a single table entry in function of the MTU:

$$M = MTU \times w$$

Table 2: Arbitration table parameters

N	Number of entries of the arbitration table
MTU	Maximum Transfer Unit in credits
n_i	Number of entries assigned to the i^{th} flow
$max\phi_i$	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
k	Bandwidth pool decoupling parameter
w	Maximum weight decoupling parameter
M	Maximum weight per table entry
$pool$	Bandwidth pool

The k parameter determines the total weight that can be distributed between all the table entries. We are going to call this value the *bandwidth pool*.

$$pool = N \times MTU \times k$$

The total number of weight units 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 bigger than the theoretical maximum weight among all the entries $(N \times M)$.

These two parameters fix the minimum and the maximum bandwidth that can be assigned to a flow:

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

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

Table 3 shows an example with a table of 64 entries and 7 flows with different distances between any consecutive pair of table entries. The w and k parameters have been set to 4 and 2 respectively. The table also shows the bandwidth that can be actually assigned to each flow depending on the number of entries and the w and k parameters. As we can see the flows can be assigned in a range that goes from half the percentage of assigned table entries to double that percentage.

When choosing the w and k parameters some considerations must be made. If we want to be able to assign a small amount of bandwidth to a flow with lots of entries, the k parameter must be small. However, the smaller k is, the smaller the maximum bandwidth that can be assigned. We can solve this by increasing the value of w but this 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

Table 3: Example of decoupling with N=64, k=2, w=4

Distance	**#entries**	**%entries**	$min\phi_i$	$max\phi_i$
2	32	50	25	100
4	16	25	12.5	50
8	8	12.5	6.25	25
16	4	6.25	3.13	12.5
32	2	3.13	1.56	6.25
64	1	1.56	0.78	3.13
64	1	1.56	0.78	3.13
Total	64	100	50	200

each entry is allowing more information to be transmitted, and thus the maximum time between any consecutive pair of table entries is higher.

6 Performance Evaluation

In this section, we evaluate the behavior of our decoupling methodology. Specifically, we apply it to the proposed DTable 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, however, our proposals can be applied to any interconnection network technology that uses a table-based scheduler. First, we will describe the main AS network model features. Secondly, the configuration of the table scheduler employed is specified. Thirdly, the traffic model is described. Finally, we present and analyze the results obtained.

Table 4: Table configuration. $N = 64$, $MTU = 34$, $k = 2.75$, and $w = 8$

VC	**#entries**	**%entries**	$min\phi_i$	$max\phi_i$	ϕ_i	**Total weight**
D2	32	50	18.18	145.45	18.18	1088
D4	16	25	9.09	72.72	18.18	1088
D8	8	12.5	4.54	36.36	18.18	1088
D16	4	6.25	2.27	18.18	18.18	1088
D32	2	3.13	1.14	9.09	9.09	544
D64	1	1.56	0.57	4.55	4.55	272
D64'	1	1.56	0.57	4.55	0.57	34
Total	64	100	36.36	290.90	86.93	5202
			Unassigned bandwidth		13.07	782
				Total	100	5984

6.1 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). 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. As stated before, AS uses VCs to aggregate flows with similar characteristics and the arbitration is made at VC 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 [12].

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. The MTU of an AS packet is 2176 bytes. The credit-based flow control unit is 64 bytes, and thus the MTU corresponds to 34 credits.

The buffer capacity is 34816 bytes (16×MTU) per VC at the network interfaces and 17408 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 in the application layer. Regarding the latency statistics, a packet is considered injected when it is stored in the network interface.

6.2 Table 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. 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 2.75 (the bandwidth pool is 2.75 times the MTU multiplied by the number of entries), and the w parameter a value of 8 (each table entry can be assigned a maximum weight of 8 times the MTU). These parameters determine the bandwidth pool, the minimum bandwidth per VC, and the maximum bandwidth per

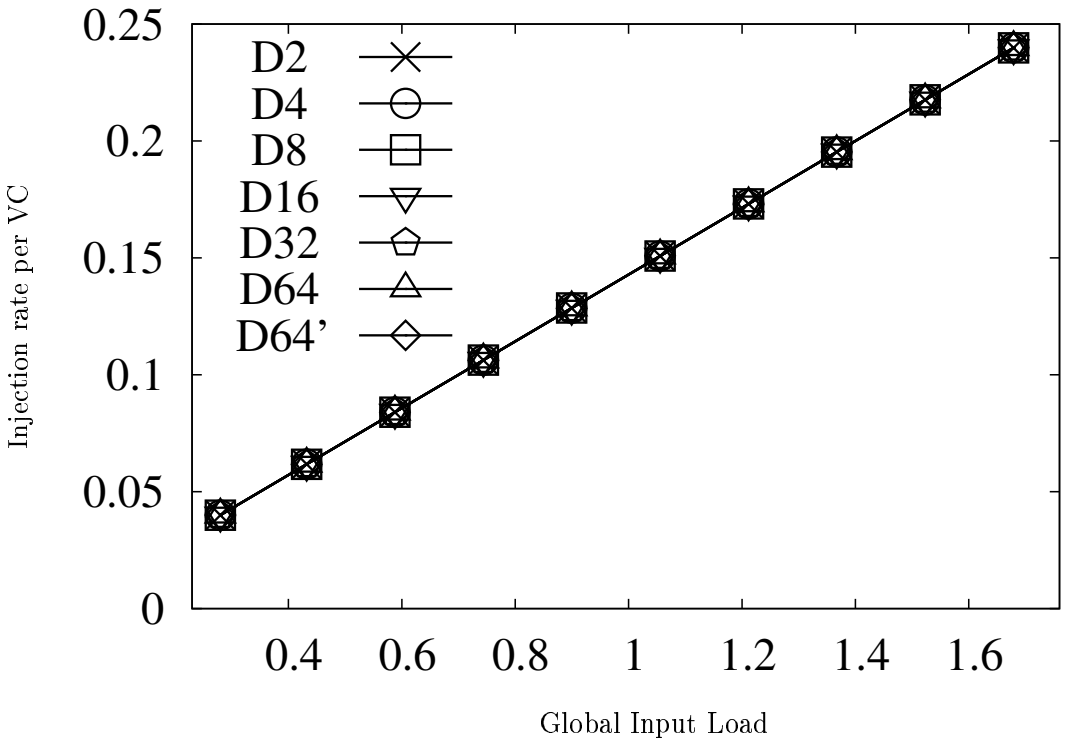

Figure 5: Normalized injection rate per VC.

VC:

$$pool = N \times MTU \times k = 64 \times 34 \times 2.75 = 5984$$

$$min\phi_i = \frac{n_i}{N} \times \frac{1}{k} = \frac{n_i}{64} \times \frac{1}{2.75}$$

$$max\phi_i = \frac{n_i}{N} \times \frac{w}{k} = \frac{n_i}{64} \times \frac{8}{2.75} = \frac{n_i}{64} \times 2.91$$

We have chosen this combination of parameter values because they allow us to assign D2, D4, D8, and D16 the same bandwidth. This is useful for two reasons. First of all, our intention is to show that we can assign bandwidth to a VC with a certain independence of the distance between consecutive entries, and thus with a certain independence of the number of entries that a VC has assigned. To do this we are going to inject traffic for all the VCs in the same proportion and we expect to obtain the same throughput for these VCs. Secondly, this allows us to study the effect of the different separation between any consecutive pair of table entries of these VCs in a fair way. Table 4 shows the number of entries assigned to each VC, the percentage of entries that this entails, the minimum and maximum bandwidth that can be assigned to each VC, the bandwidth that we have actually assigned to each VC in the simulations, and the weight that we have distributed between the table entries of each VC to configure this bandwidth.

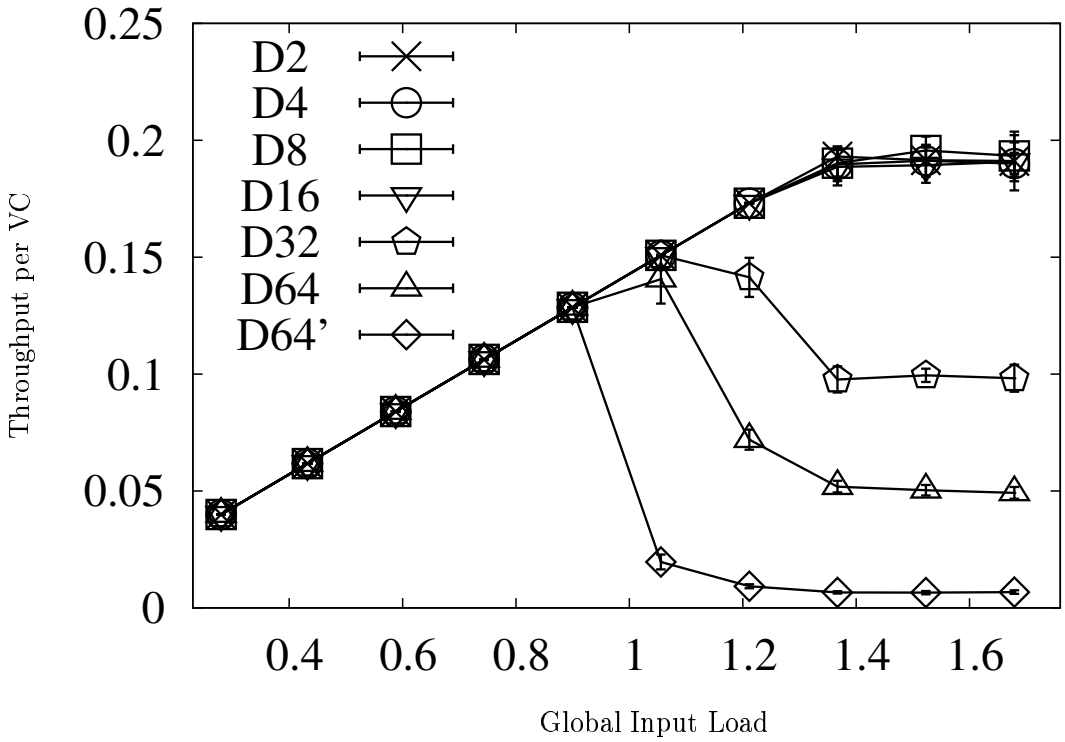

Figure 6: Normalized throughput per VC.

6.3 Traffic model

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 [13]. In this way, many small-sized packets are generated, with an occasional packet of large size. The periods between packets are modeled with a Poisson distribution.

6.4 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 average throughput, the average packet injection latency, and the maximum packet injection latency of each flow. No statistics on packet loss are given because, as has been said, AS has 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 5 shows the normalized injection rate of the aggregated of flows associated with each VC. As stated before, we inject the same amount of traffic in all the VCs. Figure 6 shows the normalized throughput results per VC. 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 band-

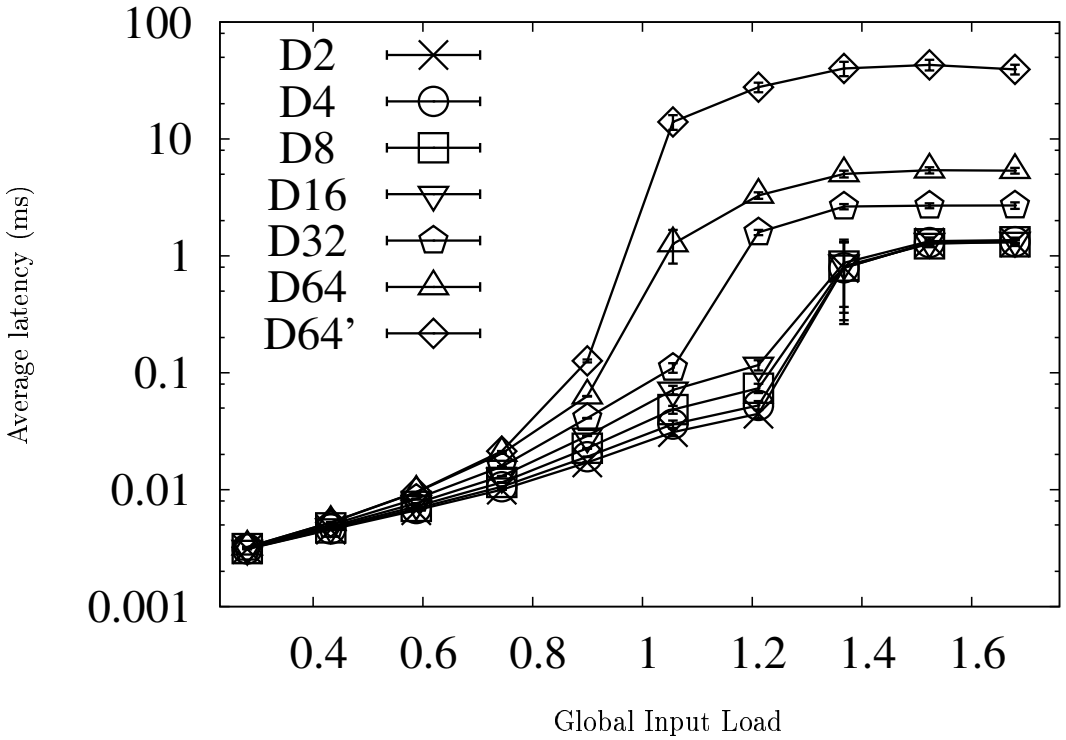

Figure 7: Average latency performance.

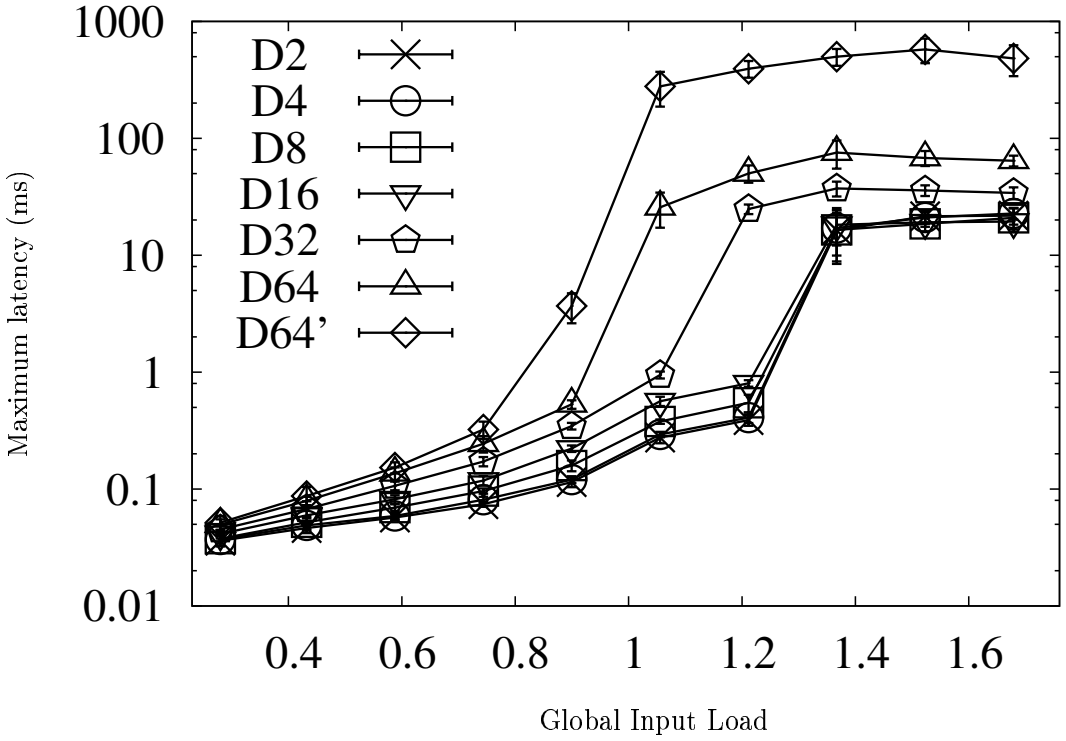

Figure 8: Maximum latency performance.

width proportional to their priority. Specifically, the D2, D4, D8, and D16 VCs obtain the same throughput although they have assigned a different number of table entries. However, the D64 and D64' VCs obtain a different throughput although they have

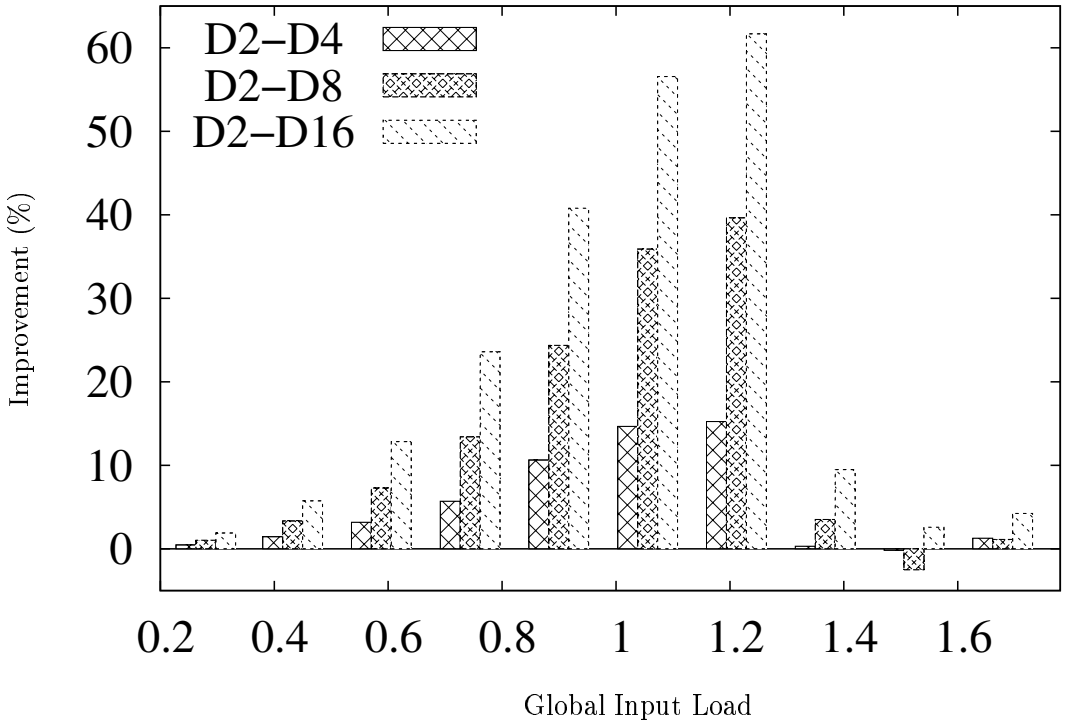

Figure 9: Average latency improvement.

assigned the same number of table entries. Therefore, this figure shows that the VCs obtain a throughput that does not depend on the number of table entries nor on the distance between any consecutive pair of entries assigned to the VCs, but on the weight assigned to their entries.

Figures 7 and 8 show the average and maximum 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. 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 that the scheduler 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 filled and the latency stabilizes at a given value which depends on the buffers' size and the bandwidth assigned to that VC.

Figures 9 and 10 show the percentage improvement on average and maximum latency of the D2 VC over the D4, D8, and D16 VCs. Note that all of these VCs inject the same traffic and obtain the same throughput. However, they 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. The percentage of improvement is very small when the load is small, but increases with the load. As stated before, this is because scheduler makes the difference when there are conflicts between packets from different VCs. However, when the load is

higher than the throughput, the buffers are almost always full. At this moment, the bandwidth that each VC obtains outweights in importance the distance of the entries in the arbitration table. This is the reason why the figures show that the percentage of improvement for those points becomes zero or does not stabilize in a clear value.

Summing up, our results show that the weight assigned to the VCs determines the proportion of bandwidth that they are going to obtain, independently of the number of table entries or the distance between any consecutive pair of entries. Moreover, the separation between any consecutive pair of the table entries of a VC determines the latency performance when the load of that VC is smaller than the throughput that it obtains.

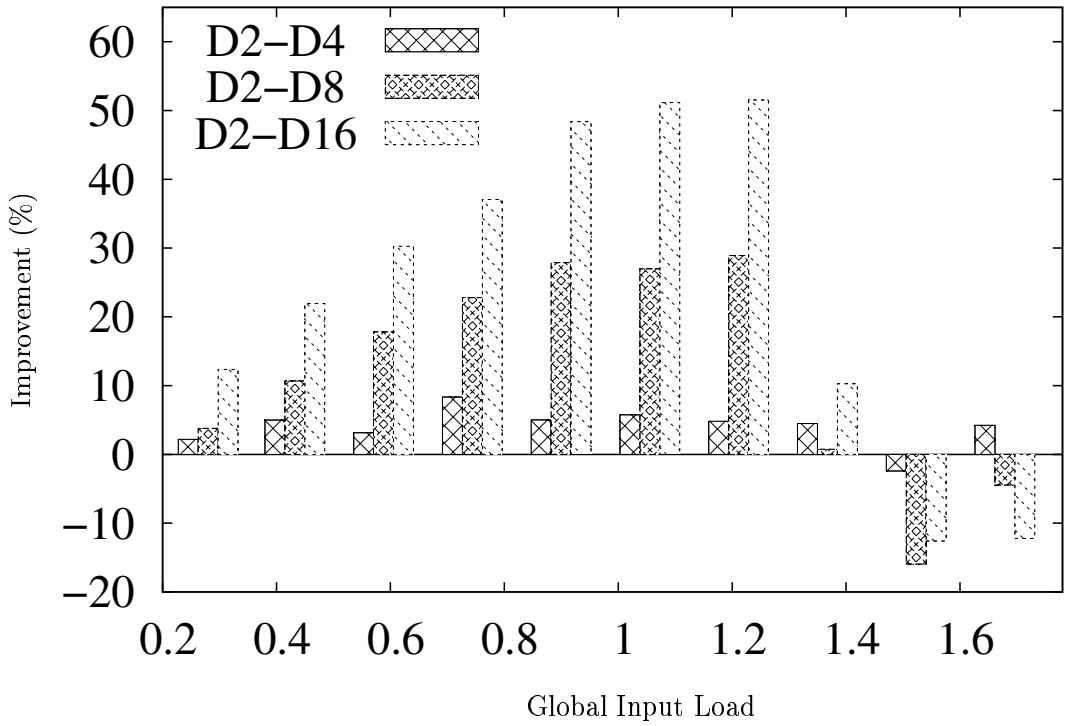

Figure 10: Maximum latency improvement.

7 Conclusions

Table-based schedulers exhibit interesting characteristics. Moreover, recent network technologies such as InfiniBand or Advanced Switching propose this kind of scheduler at the egress ports. In this paper we have shown that this kind of table-based scheduler faces the problem of bounding the bandwidth and latency assignments. We have also shown that these schedulers do not work properly with variable packet sizes. Even the weighted table of InfiniBand only partially solves the problem, and its results are far from acceptable.

In this paper we have proposed the Deficit Table scheduler, which is a new table-based scheduler that works properly with variable packet sizes. As far as we know, this

is the first table-based scheduler proposal that is able to deal properly with variable packet sizes.

Moreover, we have proposed a methodology for decoupling the bandwidth assignment from the latency requirements for table-based schedulers. 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 tested our proposals in an Advanced Switching simulator, although they can be applied to any interconnection network technology. Simulation results show that the weight assigned to the VCs fixes the bandwidth they receive, independently of the number of table entries or the maximum distance between them, while the latency performance comes from the separation between any consecutive pair of table entries assigned to the VC.

These results are extremely important because they offer us the solution to two major problems of the table-based schedulers, which can be used in current interconnection standards such as InfiniBand or Advanced Switching, or in future proposals of interconnection standards.

Bibliography

[1] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 611–624, 1998.

[2] H. M. Chaskar and U. Madhow, "Fair scheduling with tunable latency: A round-robin approach," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 592–601, 2003.

[3] M. Katevenis, S. Sidiropoulos, and C. Corcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE Journal on Selected Areas in Communications*, Oct. 1991.

[4] *Advanced Switching core architecture specification. Revision 1.0*, Advanced Switching Interconnect Special Interest Group, Dec. 2003.

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

[6] F. J. Alfaro, J. L. Sánchez, and J. Duato, "QoS in InfiniBand subnetworks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 810–823, Sept. 2004.

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

[8] A. K. Parekh and R. G. Gallagher, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," *IEEE/ACM Transactions on Networking*, 1994.

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

[10] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *SIGCOMM*, 1995, pp. 231–242.

[11] S. S. Kanhere, H. Sethu, and A. B. Parekh, "Fair and efficient packet scheduling using elastic round robin," *IEEE Transactions on Parallel and Distributed Systems*, 2002.

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

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