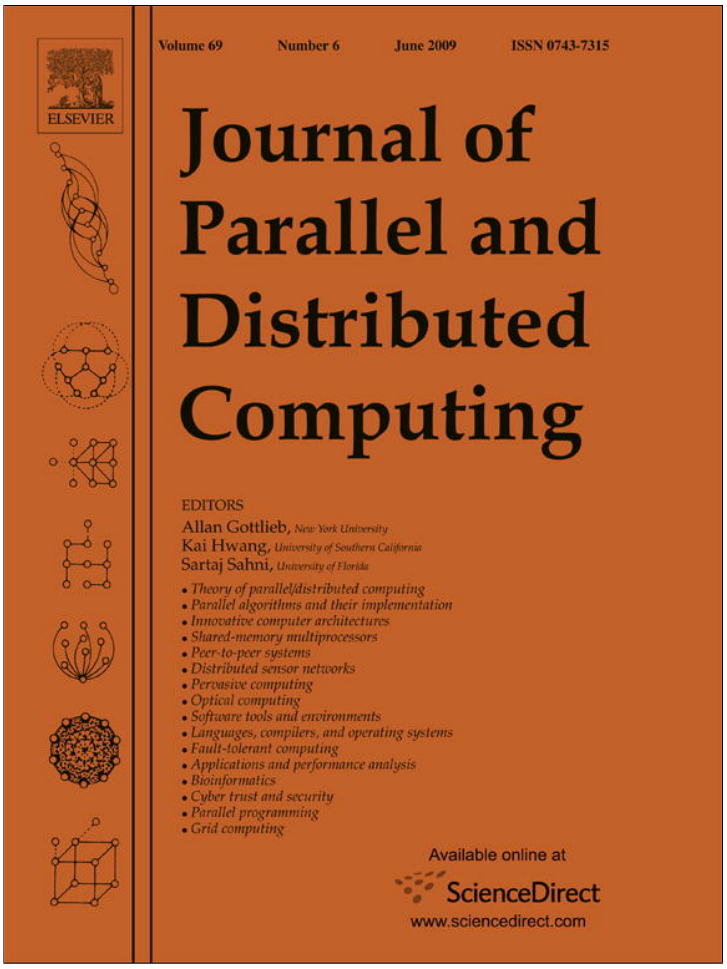

J. Parallel Distrib. Comput. 69 (2009) 508–520

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A new strategy to manage the InfiniBand arbitration tables

Francisco J. Alfaro [a,*], José L. Sánchez [a], José Duato [b]

[a] *Dept. de Sistemas Informáticos, Escuela Superior de Ingeniería Informática, Universidad de Castilla-La Mancha, 02071- Albacete, Spain*

[b] *Dept. de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, 46071- Valencia, Spain*

A R T I C L E I N F O

Article history:
Received 20 December 2006
Received in revised form
9 January 2009
Accepted 11 February 2009
Available online 6 March 2009

Keywords:
Interconnection networks
InfiniBand
QoS
Arbitration
Clusters
Connection requirements

A B S T R A C T

The InfiniBand Architecture (IBA) is an industry-standard architecture for server I/O and interprocessor communication. InfiniBand is extensively used for interconnection in high-performance clusters. It has been developed by the InfiniBand^{SM} Trade Association (IBTA) to provide the levels of reliability, availability, performance, scalability, and quality of service (QoS) necessary for present and future server systems.

The provision of QoS in data communication networks is currently the focus of much discussion and research in both industry and academia. In that sense, IBA enables QoS support with some mechanisms. In this paper, we examine these mechanisms and describe a way to use them. We propose a traffic segregation strategy based only on delay requirements. Moreover, we propose a very effective methodology to compute the virtual lane arbitration tables. Finally, we evaluate our proposal and performance results show that, with a correct traffic treatment at the output ports, every traffic class meets its QoS requirements.

1. Introduction

The InfiniBand Architecture (IBA) [9] is a standard for high-speed I/O and interprocessor communication. It has been developed by the InfiniBand Trade Association (IBTA) [10]. This association, founded in 1999, includes over 200 leading IT companies. Membership is also open to Universities, research laboratories, and others. The IBTA is led by a Steering Committee whose members come from Cisco, IBM, Intel, Mellanox, QLogic, Sun, and Voltaire. The proposed standard is a fast, highly scalable interconnect technology suitable for clusters and high-end servers executing distributed applications. The IBA specification is open to further enhancements by manufacturers and researchers.

On the other hand, many current applications have requirements, such as guarantee of bandwidth, bounded delivery deadline, bounded interarrival delays, etc., which not all the current networks are able to provide. Therefore, it is important for InfiniBand to be able to satisfy the QoS requirements of current applications. InfiniBand provides a series of mechanisms that, when properly used, are able to provide QoS to the applications. These mechanisms are mainly the segregation of traffic according to categories and the arbitration of the output ports according to an arbitration table that can be configured to give priority to certain flows.

Although IBA is currently used almost exclusively on clusters for high-performance computing, it may expand its market in the future, being implemented in clusters for different application areas that may require QoS support. We envisage two different application areas where our proposals may prove to be very useful. The first one is providing Internet services that require QoS guarantees (e.g. video on demand) to a very large number of concurrent clients. While it is true that the Internet protocols play a critical role in providing such a QoS, it is also true that those Internet servers must be highly parallel (e.g. a cluster) and will require internal QoS support when retrieving information from the disk subsystem and transmitting it through its system area network (e.g. InfiniBand) from the disks to the server nodes.

The second scenario is providing bandwidth and latency guarantees to each partition when the interconnection network of a cluster is partitioned into several virtual networks. This partitioning is quickly becoming very important as the trend toward virtualization continues, and an increasing number of companies are providing service to many customers by splitting a physical server into multiple virtual servers.

The Internet Engineering Task Force (IETF) is currently in the process of developing an architecture for providing QoS on the Internet. This effort is referred to as Differentiated Services [6]. An overview of a possible implementation of DiffServ over IBA has been described in [12]. In this study the traffic is classified into several categories and the author proposes that the arbitration tables of InfiniBand should deal with each category in a different way, but no attempt is made to indicate how to fill in those tables.

* Corresponding author.
E-mail addresses: falfaro@dsi.uclm.es (F.J. Alfaro), jsanchez@dsi.uclm.es (J.L. Sánchez), jduato@gap.upv.es (J. Duato).

0743-7315/$ – see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2009.02.002

In this paper, we propose a classification of the different traffic types with QoS needs that improves the proposal made in [12]. We describe how InfiniBand is able to provide bandwidth and/or latency guarantees. Besides, we study two different approaches to distribute the entries of the arbitration table that a connection requires. The first approach (CENRE) categorizes the requests based on the exact number of required entries, while the second approach (CESY) categorizes the requests in the powers that are divisors of 64. Finally, after a deep study we propose the CESY approach as a very effective strategy to compute the arbitration tables to obtain the QoS required by the applications.

The structure of the paper is as follows: Section 2 presents a summary of the general aspects in the specifications of InfiniBand, including the most important mechanisms that InfiniBand provides to support QoS. In Section 3, we present our proposal to treat the different types of traffic based on its requirements. In Sections 4 and 5 we study in depth how to provide QoS in InfiniBand, studying specifically how to provide bandwidth and latency guarantees. In Section 6 the performance evaluation is presented. Finally, some conclusions are given.

2. InfiniBand

The InfiniBand Architecture Specification [9] describes a System Area Network (SAN) for connecting multiple independent processor platforms (i.e. host processor nodes), I/O platforms, and I/O devices. The IBA SAN is a communication and management infrastructure supporting both I/O and interprocessor communications for one or more computer systems. IBA is designed around a switch-based interconnect technology with high-speed point-to-point links.

An IBA network is divided into subnets interconnected by routers, each subnet consisting of one or more switches, processing nodes, and I/O devices. IBA switches route messages from their source to their destination based on forwarding tables that are programmed with forwarding information during initialization and after network modification. Routing between different subnets (across routers) is done on the basis of a Global Identifier (GID) 128 bits long, modeled over IPv6 addresses. On the other hand, the addressing used by switches is based on Local Identifiers (LID) which allow 48K endnodes on a single subnet, the remaining 16K LID addresses being reserved for multicast.

IBA links are bidirectional point-to-point communication channels, and may be either copper cable, optical fiber or printed circuit on a backplane. The signaling rate on the links is 2.5 GHz in the 1.0 release, the later releases possibly being faster. The physical links may be used in parallel to achieve higher bandwidth. The IBA specification defines three link bit rates. The lowest one is 2.5 Gbps and is referred to as 1x (only one link at 2.5 GHz). Other link rates are 10 Gbps (referred to as 4x) and 30 Gbps (referred to as 12x) that correspond to 4-bit wide and 12-bit wide links, respectively. Currently, there are also versions 12x DDR (60 Gbps) and 12x QDR (120 Gbps). The width or widths that will be supported by a link is vendor-specific.

Messages are segmented into packets for transmission on links and through switches. The packet size is such that after headers are considered, the Maximum Transfer Unit (MTU) of data may be 256 bytes, 1 KB, 2 KB or 4 KB. Each packet, even those for unreliable datagrams, contains two separate CRCs, one covering data that cannot change, and another one covering data that changes in switches or routers, and it is recomputed.

The IBA transport mechanisms provide several types of communication services between endnodes. These types are connections or datagrams and both can be reliable (acknowledged) or unreliable.

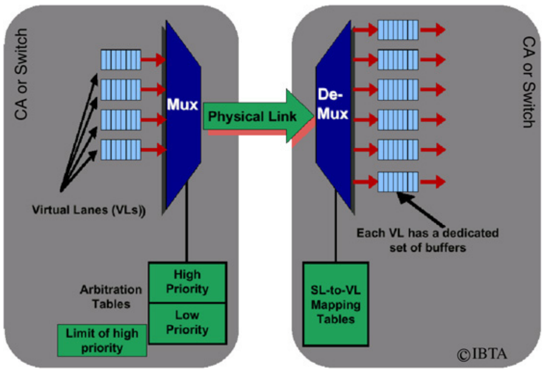

Fig. 1. Virtual lanes in a physical link.

IBA management is defined in terms of managers and agents. While managers are active entities, agents are passive entities that respond to messages from managers. Every IBA subnet must contain a single master subnet manager, residing on an endnode or a switch, which discovers and initializes the network.

2.1. IBA support for QoS

Basically, IBA defines three mechanisms that permit QoS to be supported: service levels (SLs), virtual lanes (VLs), and virtual lane arbitration for transmission over links. IBA defines a maximum of 16 SLs, but it does not specify what characteristics the traffic of each SL should have. Therefore, it depends on the implementation or on the administrator how to distribute the different existing traffic types among the SLs. By allowing the traffic to be segregated by category, we will be able to distinguish between packets from different SLs and to give them a different treatment based on their needs.

IBA ports support VLs, providing a mechanism for creating multiple virtual links within a single physical link. A VL represents a set of transmit and receive buffers in a port (Fig. 1). IBA ports must support a minimum of two and a maximum of 16 VLs ($VL_0 \dots VL_{15}$). All the ports support VL_{15}, which is reserved exclusively for subnet management, and must always have priority over data traffic in the VLs. The number of VLs used by a port is configured by the subnet manager. Since systems can be constructed with switches supporting different numbers of VLs, packets are marked with a SL, and a relation between SL and VL is established at the input of each link by means of the *SLtoVLMappingTable*. Each VL must be an independent resource for flow control purposes.

When more than two VLs are implemented, the priorities of the data lanes are defined by the *VLArbitrationTable*. This arbitration is only for data VLs, because VL_{15}, which transports control traffic always has priority over any other VL, as previously stated.

The structure of the VLArbitrationTable is shown in Fig. 2. Each VLArbitrationTable consists of two tables, one for delivering packets from high-priority VLs and another one for low-priority VLs. However, IBA does not specify what is high and low priority. Both arbitration tables implement weighted round-robin (WRR) [11] arbitration within each priority level. Up to 64 table entries are cycled through, each one specifying a VL and a weight, which is the number of units of 64 bytes to be sent from that VL. This weight must be in the range of 0 to 255, and is always rounded up in order to transmit a whole packet.

A *LimitOfHighPriority* value specifies the maximum number of high-priority packets that can be transmitted before a low-priority packet is sent. More specifically, the VLs of the High-Priority table can transmit *LimitOfHighPriority* $\times$ 4096 bytes before a packet from the Low-Priority table could be transmitted. If no high-priority packets are ready for transmission at a given time, low-priority packets can also be transmitted.

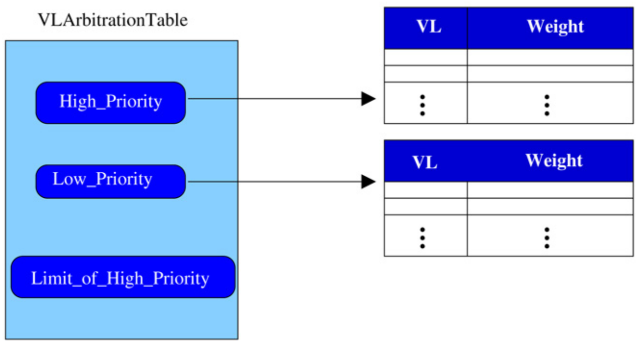

Fig. 2. VLArbitrationTable structure.

3. Traffic classification

Pelissier proposed in [12] a traffic classification for the different traffic flows. This classification is based on the applications' requirements. We have assumed this classification, and we have introduced slight modifications that are described in [1]. Specifically, the categories we have considered are the following:

- **DBTS** (Dedicated Bandwidth Time Sensitive). This category includes the kind of traffic that needs both a mean bandwidth and also a guaranteed maximum delay. Example of this kind of traffic are the video-conference or the interactive audio.
- **DB** (Dedicated Bandwidth). This category includes traffic only requiring a guarantee of mean bandwidth. Usually, this kind of traffic is not very sensitive to the latency. Thus, it is not necessary to provide it with any guarantee in this sense. An example of this class of traffic is the video visualization from a server.
- **PBE** (Preferential Best-Effort). This category embraces all kinds of traffic that do not need explicit guarantees in maximum latency nor minimum bandwidth, but it is desirable to provide it with a better treatment than other best-effort traffic. This is the case, for example, of web traffic or the traffic accessing a database. It is useful for this kind of traffic to have priority over the rest of the traffic without guarantees in order to improve its behavior.
- **BE** (Best-Effort). This kind of traffic does not need any type of bandwidth guarantees nor maximum latency. Usually it is enough to guarantee that, sooner or later, it will reach its destination. This task is usually carried out by the upper levels of the software architecture. In current networks, most of the traffic is usually of this type. As an example of this category, we can mention the transfer of files, printing services, etc.
- **CH** (Challenged). This kind of traffic is intentionally degraded so that it does not interfere with any other traffic type. An example of this kind of traffic is any type of activity of backup in a server. It is important that these activities are not carried out when there is any other type of traffic in the network. It may be a good idea to perform these tasks at night when no other traffic is using the network.

Pelissier proposed the categories DBTS, DB, BE, and CH. We proposed splitting up the BE traffic into PBE and the usual BE in order to provide different treatments for these kinds of applications that must be served without guarantees, without aiming at the same level of performance.

In our earlier papers, we agreed with Pelissier in devoting the high-priority arbitration table of InfiniBand to the DBTS traffic, and the low-priority arbitration table to the remaining categories. However, in [1] we pointed out that this proposal was problematic. If the sources generating DBTS traffic send packets exceeding the bandwidth previously requested, all the traffic using the low-priority table will be affected. Specifically, if we cannot guarantee that all the sources will have a behavior according to what they previously requested, no guarantees can be given to the traffic using only the low-priority table.

This behavior is caused because the *LimitOfHighPriority* value does not permit a "fine-grain" distribution of the bandwidth. As we mentioned above, this limit can have a value between 0 and 255. Specifically, the VLs in the high-priority table can transmit *LimitOfHighPriority* $\times$ 4096 bytes before having to transmit a low-priority packet. In the best case, for the largest packet allowed in InfiniBand (4096 bytes), this means that for a *LimitOfHighPriority* of 1 we could transmit traffic, 50% coming from the VLs in the high-priority table, and the other 50% coming from the VLs in the low-priority table.

It is evident, therefore, that in order to provide guarantees for any kind of traffic, the VLs used by this traffic must be included in the high-priority table. Specifically, we propose to place in the high-priority table of InfiniBand the VLs used for DBTS traffic, but unlike Pelissier's proposal, also the VLs used for DB traffic. Thus, we propose to leave the low-priority table for traffic flows without explicit guarantee requirements, which are PBE, BE, and CH. Furthermore, the bandwidth distribution will be performed when the connections are established.

4. Providing QoS in InfiniBand

In this section we will review the way we propose to guarantee applications' requirements [5]. As far as we know, nobody has proposed a different method to fill in the IBA arbitration table to provide applications with QoS. Obviously, our proposal is based on the mechanisms provided by the InfiniBand specification. When a host wants to establish a new connection it must decide which characteristics of mean bandwidth and maximum delay it wants to request from the network. Using this request, our methodology performs a resource reservation. This outline is similar to what is done with the RSVP protocol [7]. Therefore, the QoS guarantee can be carried out in two senses: Bandwidth guarantee and/or maximum latency guarantee. To provide these guarantees the applications must use the Reliable Connection Service of the InfiniBand Transport Layer.

Specifically, the requesting host sends a message with its QoS requirements asking for a new connection. Depending on the network model used, this requesting message will have a different destination. We are going to assume a distributed control model where the switches have enough complexity to perform this task themselves. This approach agrees with the InfiniBand specification where the local agents are able to perform that task. The requesting message is therefore sent through the destination host, and is analyzed in each switch to be found in the path toward the destination host. However, in a centralized control model the switches cannot perform this task. In this case, this task should be performed by the subnet manager. So, the requesting message would be sent to the subnet manager that would study the request and the path to establish the connection. The subnet manager should have the necessary structures to store and to manage the information of each switch and host in the network. In the following we are going to study the distributed model, where each intermediate switch can take its own decisions. However, all the analysis carried out here is applicable to the centralized model where the subnet manager performs this admission task.

Using the distributed admission model, the message requesting a new connection travels through the network. This requesting message contains the requirements necessary for the connection to be established. This message travels through the network up to the target host or up to the intermediate switch where the requirements are denied. This request is then studied in each intermediate switch and is forwarded toward the next switch in the path. If the request cannot be accepted in some intermediate switch or at the target host, it is rejected and an informative message is sent back to the predecessor switches in the path and to the source host.

We have considered two possible requirements that are bandwidth and maximum end-to-end delay. Obviously, both must be studied. We are, therefore, going to analyze separately how to achieve guarantees for both objectives. Finally, we will study how to treat both of them at the same time.

4.1. Bandwidth guarantee

In a distributed network admission model we can assume that both switches and hosts know the bandwidth previously reserved for other connections. Thus, it is easy to check if the bandwidth request can be accepted or must be rejected. The accumulated bandwidth must be added to the bandwidth requested by the new connection. If this value is lower than the maximum bandwidth of the link, the requested bandwidth could be accepted.

According to the InfiniBand specification, a weight of one unit in the arbitration table permits the transmission of a block of 64 bytes. We know the link speed and the mean bandwidth requested by the connection, and so we can easily compute the weight that a connection must have in the arbitration table. The whole process is explained in depth in [2].

However, while performing the previous weight computation, we must carry out a rounding up. We would then be giving each connection more weight which it needs. Thus, if we perform this computation regarding each connection in an individual way, repeated rounding up accumulates, and this would fill in the arbitration table with connections that do not use up the real bandwidth. However, to solve this problem, we will perform this calculation in a different way. We will compute the weight for a certain entry of the table based on the bandwidth that must be satisfied by the connections using this entry. In this way, we waste nothing with the rounding up, because each time a new connection using the same entry is accepted, we will recompute the weight for that entry, but now taking into account the new total bandwidth accumulated for this entry.

This proposal forces us to create a new structure in each port with the capacity to store this information. Specifically, we need to store a floating point value for each entry of the high-priority arbitration table. Assuming a floating point takes up 8 bytes, and as each high-priority arbitration table could have up to 64 entries, a total of $64 \times 8 = 512$ bytes per switch port or host interface are needed. Note that this structure is only needed for the high-priority table, because the low-priority table will not be modified based on the bandwidth.

4.2. Delay guarantee

We perform the treatment of the latency distributing the total bounded maximum delay among the switches on the path, also taking into account the flying time on the links. Thus, the study in each intermediate switch is easier and we also guarantee the same treatment in all of them, which we consider to be highly desirable.

In [1] we have studied how to perform this check. Each switch can compute the maximum number of packets that can be transmitted before one packet of a certain VL is transmitted. To do that, we must take into account the number of ports the switch has, the number of VLs per port, and the input and output buffer size. This computing also depends on the structure of the crossbar (multiplexed or full-crossbar) and the maximum packet size allowed. Besides, the behavior of the output port virtual line arbitration table must be taken into account. To be able to achieve the latency requirement the connection might have to use several entries in the arbitration table of the output port of the switch. These entries must have a certain maximum distance between them in order to guarantee the maximum number of packets that can be transmitted before one packet of that VL is actually transmitted. Knowing the maximum number of packets that can be transmitted before a packet of a certain VL and the link speed, it is easy to compute the maximum time that can be spent waiting for a packet to be transmitted from a switch. To find more details of how this computing is done the reader can consult [1].

Having assured this maximum distance between two consecutive entries devoted to the VL used by the connection, there will be a guarantee of the amount of information from other VLs (maximum time) that can be transmitted before one of its packets is transmitted.

4.3. Providing guarantee of bandwidth and delay at the same time

A connection, therefore, could request a certain mean bandwidth of B Mbps and a bounded end-to-end delay of t milliseconds. This request will be treated in each node as a request of a certain weight w and a maximum distance d between any consecutive pair in the arbitration table.

Therefore, the total number of entries used by the connection in the high-priority arbitration table of the output port of the switches it crosses will be the maximum between these two values: the entries needed due to the bandwidth requirement ($\frac{w}{255}$) and those resulting from the requirement of maximum distance between any consecutive pair in the table.

If several connections requiring the same distance between two consecutive entries share a sequence of entries in the table, all of them will, obviously, use the same VL. As mentioned earlier, the weight that each one of these entries of the sequence has will be based on the accumulated bandwidth for all the connections sharing that sequence of entries.

When a node is studying the maximum distance between two consecutive entries in the table, it must look for an available sequence of entries in the table to be used for that connection. Specifically, we have three possibilities:

- We can use an already used sequence with the same maximum distance (the same VL), but only if it has available weight. For a sequence of entries with maximum distance d, we have $\frac{64}{d}$ entries in the table, each one with a maximum weight of 255. So, the total weight that can be accumulated in this sequence is $\frac{64\times255}{d}$.
  Therefore, each node will have a requirement table with the sequences of entries of the arbitration table already assigned and the bandwidth accumulated. If there is already a sequence in the arbitration table of the same distance as that requested, we will try to use its entries. Specifically, if the needed weight for the accumulated bandwidth of all the connections sharing the sequence, including the bandwidth requested by the new connection, does not exceed the limit for this sequence, this new connection can use the already established sequence. Obviously, we must recompute the weight of the entries of the sequence with the new bandwidth accumulated by the connections that are using it.
- There are one or more sequences in the high-priority arbitration table with the same maximum distance as that requested by the new connection, but none of them can be used. The reason is that the accumulated weight for their entries does not permit us to place this new connection there because the maximum weight would be exceeded.
- There is no previously established sequence for this distance in the high-priority arbitration table.

The first case is very easy, and we only need to recompute the weights of the sequence of entries. The other two cases require to find a new free sequence of entries in the table. Besides, this searching should be performed in an efficient way.

5. Looking for a new sequence of entries in the arbitration table

As was stated at the end of the previous section, in order to be able to guarantee the demand of a new connection request, in some situations we need to find a sequence of free entries in the arbitration table. One aspect that has great importance for the complexity of the searching method is the distance between two consecutive entries that form the sequence of entries. Depending on the election made, we could have very different searching methods with different complexity order. Besides, depending on the selected method, the later management of the requests (insertion of new requests and release of requests already situated in the table) will be more or less complex.

As was previously stated, the high-priority arbitration table has 64 entries that must be assigned to some requests with certain requirements for the distance between two consecutive entries of the same sequence. Thus, it is the bounded delay which will mark the separation between the entries of the sequence.

We say a connection request is of type $\boldsymbol{d}$ if it needs a maximum distance $\boldsymbol{d}$ between two consecutive entries of the sequence that its VL uses in the arbitration table of the switch output ports[1]. So, a connection request of type 15 needs a separation between two consecutive entries of maximum 15 units. One connection of type 8 requires that two consecutive entries be situated at a maximum separation of 8 units. As the arbitration table has a cyclic behavior, a connection request of type d needs $\lceil \frac{64}{d} \rceil$ (rounding up function) entries of the arbitration table.

According to the InfiniBand specification, all traffic of the same VL is going to receive the same treatment in the arbitration table. This is because the arbitration process is based on the output VL of the port or interface. Note that the selection of the VL for a packet depends on the SL indicated in its header and the *SLtoVLMappingTable*. So, the application is going to mark the packet with a certain SL based on the characteristics of the traffic generated. The VLs in the switches of its path up to its target will be selected based on the SL that has been put in its header.

On the other hand, in our previous works [5] we considered categorizing the traffic of the different applications based on its mean bandwidth requirements. We did that independently of the needs that they have of bounded end-to-end delay, and thus, of the maximum separation between two consecutive entries in the high-priority arbitration table. This forced us to join in the same SL, and thus in the same VL, connections having a very different deadline requirement, but a similar mean bandwidth requirement.

Using this grouping criterion we must give to all the traffic of the VL the same treatment regarding delay as the most restrictive connection requires. Note that the maximum distance between two consecutive entries in the sequence is not the unique distance that can be accepted. Any lower separation would also be acceptable. The problem is what to do when this most restrictive connection finishes and its requirements in the arbitration table must be released. There are basically two options:

- To do nothing and maintain the current situation for this VL. This would waste resources in the arbitration table, because we are dedicating more entries to this VL than the remaining connections really need.
- To modify the entries of the arbitration table so that this VL receives the treatment of the remaining new most restrictive connection using the VL. However, this option is very problematic. We need to store all the information of the connections that each port is managing. Without this information it would be impossible to know which is the next most restrictive connection. This option is clearly not viable, because we would need to store information, taking up a lot of space.

Both alternatives pose too many problems to be considered further. To solve this problem a solution would be to use different VLs for the different traffic flows for which we want to provide a different treatment. Therefore, our proposal is to segregate the traffic so that all the connections that share a VL have the same maximum distance requirement in the arbitration table. In this way, it is not necessary to keep any kind of additional information regarding the established connections, because all the connections that share that VL will have the same maximum distance requirement between two consecutive entries. When a connection finishes we discount the bandwidth that it requested in its establishment, and we recompute the weight of the entries with the bandwidth accumulated by the remaining connections using the sequence of entries. When the bandwidth accumulated is zero, this means that there are no connections accumulated using that sequence of entries, and so it can be released.

Obviously, for a table of 64 entries there can be 64 different distances requested, and so 64 possible types of requests. As the number of SLs and VLs is limited, at most they can be used for as many types of requests as these values indicate. According to the InfiniBand specification, the number of SLs is 16, and the number of VLs can be 16, 8, 4, or 2, depending on the implementation. It is, therefore, the number of VLs that will impose the maximum number of different types of applications, and also the distances to be considered. Besides, some SLs and VLs must be devoted to the traffic that does not require QoS.

In the following, we are going to determine the set of different distances to be considered. We will present two alternatives that are based on different criteria, and finally we will select one of them. The first one is called *CENRE* and consists in grouping the distances requiring exactly the same number of entries. The second one is called *CESY* and considers the symmetry to establish the different types of requests.

5.1. Categorization based on the Exact Number of Required Entries: CENRE

According to the size of the table, there can be 64 different types of request, and therefore, some grouping criterion must be applied. However, many of the 64 possible types of distance need the same number of entries in the table, and so they could be grouped together and given the same treatment. Thus, specifically we could have the following 15 different types of maximum distance:

- Requests with low latency requirements only need one entry in the table, and so the requested maximum distance is 64.
- Requests of distance in the range [32, 63] require 2 entries in the table, and so all of them can be treated as if they were of distance 32.
- Requests of distance in the range [22, 31] need 3 entries in the table, and so all of them can be treated as if they were of distance 22.
- Requests in the range [16, 21] need 4 entries and can be located as requests of distance 16.
- Requests in the range [13, 15] require 5 entries, and so they can be treated as if they were of type 13.
- Requests of distance in the range [11, 12] both need 6 entries.

[1] As type of connection and distance have the same value and meaning for a certain connection, in the following we are going to use both terms synonymously to refer to the same concept.

- Only requests of distance 10 need 7 entries. These requests cannot therefore be grouped with other requests with different distance requirement without using more entries than those strictly necessary.
- Both types of requests in the range [8, 9] require 8 entries.
- Only requests of distance 7 need 10 entries. These requests cannot therefore be grouped with other requests with a different distance requirement without using more than the necessary entries.
- The same happens with the requests of distance 6 that need 11 entries.
- Only the requests of distance 5 require 13 entries.
- Only the requests of distance 4 need 16 entries.
- Only the requests of distance 3 need 22 entries.
- Only the requests of distance 2 require 32 entries.
- Finally, a request of distance 1 would need all entries of the table, which is non-viable and so we will not consider this distance.

Summing up, in some cases by changing a distance into a lower one, the distance between two consecutive entries is the same. This is helpful because it could simplify the later management of the requests. Specifically, this happens when the grouping is performed in a distance which is the divisor of the total number of entries of the table. Thus, the grouping where the maximum distances considered are 64, 32, 16, 8, 4, and 2, permits us to maintain the same distance between two consecutive entries of the sequence. However, in the other cases it is not possible for any pair of consecutive entries to have the same distance.

5.1.1. *Selecting the sequence*

Anyway, we must select exactly which entries are going to be used to meet a certain request, whatever their distances. For the distances previously shown, many algorithms are possible to locate a request in the table. Note that, in general, the only necessary condition to be able to locate a request of distance $\boldsymbol{d}$ in the table is that a sequence, of length equal to or bigger than $\boldsymbol{d}$, of consecutive fulfilled entries does not exist in the table.

The CENRE approach uses the minimum number necessary of entries. So, a request of type 8 would need $\lceil \frac{64}{8} \rceil = 8$ entries (for example the entries 2, 10, 18, 26, 34, 42, 50, 58). While a request of type 15 would need $\lceil \frac{64}{15} \rceil = 5$ entries (for example, the entries 1, 16, 31, 46, 61).

As has been previously indicated, our goal is to achieve an optimal situation for the requests in the table, maximizing the number of requests that can be met. We must therefore take into account three essential criteria: the request type, the number of requests, and the arrival order of the different requests. The arrival order is important because the decision for locating a request must be made when the request is made with the information that the process has at that moment. Depending on the selected positions, other requests could, or could not, be located later.

As we can observe, the entries of the first example (the request of type 8) are situated in a symmetric way in an arithmetic progression with difference 8. On the other hand, the entries of the second example (the request of type 15) loses its symmetry in the last entry because between the entries 61 and 1 (with the cyclical behavior of the table) the distance is not 15, but 4. For this second case, we have another possibility of distributing the 5 entries along all the arbitration table with a distance lower than 15. So, for example, another valid sequence for this request could consist of the entries 1, 14, 27, 40, and 53. In this case all the entries have a separation of 13, except the last one having only a separation of 12. This case corresponds to the grouping shown in the previous section, where we saw that the requests with distances in the range [13, 15] could be treated as if they were of distance 13 using 5 entries. Clearly, more possibilities to locate a request of type 15 are possible.

```
1: $max = 64$               // Size table
2: t[$max$]                 // IBA table
3: t = 0                    // All free entries
4: free_entries = $max$     // Number of free entries
5: group = 0                // Number of consecutive reserved entries
```

Fig. 3. Necessary definitions for the algorithms implemented.

As we can see, following the CENRE approach there are a lot of possibilities to locate this request. The best choice is to obtain a method optimizing the number of locations. So, when a request is located in the table it should leave as many free holes[2] as possible. Besides, the remaining free entries of the table should be in the most convenient positions in order to later locate other requests. Thus, the algorithm that optimizes the location is that which allows the placement of the most restrictive request in the next step. Obviously, the most restrictive request is that which needs the lowest maximum distance, and as a consequence the largest number of entries in the table.

Logically, the most restrictive request is the one of type 1 that needs all the entries in the table, but we have already indicated that this type is not realistic and so we are not going to consider it. Thus, the most restrictive request that we consider is of type 2, that needs 32 entries in the table, all of them situated at a maximum distance of 2 units from its neighbors in the sequence. The next most restrictive is the request of type 3 that needs 22 entries, and so on. Therefore, to be able always to put in the most restrictive request (type 2) without using more entries than those strictly necessary, the CENRE approach must always use first the even entries and when these run out, the odd, or vice-versa. Thus, we keep enough entries with the correct separation to be able to admit a later request of type 2. It is clear that the same criterion must be applied in order to maximize the number of entries of greater distances.

Of course, another possibility is to use more entries than those necessary and modify the distance between two consecutive entries when necessary. For example, let's suppose the entry number 20 is occupied by a previous request and we want to meet a request of distance 2. In this case, we could use the sequence..., 16, 18, 19, 21, 23, …, if these entries are free. Obviously, this increases the number of entries that we are using, and complicates the later management of the table.

In order to perform the comparison of Section 5.3, we have implemented the CENRE approach. The necessary definitions for this implementation are shown in the Fig. 3, and an implementation of the CENRE approach is shown in Fig. 4.

In the proposed implementation, given a request of type *d*, first of all the number of necessary entries for attending a request of type *d* is obtained by means of the function *Obtain*. Then, all the possible free entry sequences are explored, and the entry sequence generating the largest set of free consecutive entries is selected. Once the best sequence is selected, the variables and the table entries are updated.

The implementation we propose for the CENRE approach follows two criteria to select the entries:

- The selected sequence leaves the smallest group of consecutive occupied entries in the table. In this way, it is later possible to meet the most restrictive request possible.
- If there are some sequences meeting the previous criterion, the sequence maximizing the smallest of the free groups of consecutive entries is selected.

2 Set of free consecutive entries.

```
 1: Begin Procedure CENRE ( ibatable t, requests $d$)
 2: entries = Obtain($d$)      // Obtain the number of entries to attend request $d$
 3: if (entries $\leq$ free_entries) and (group $<$ $d$) then
 4:   // It's sure request $d$ can be attended
 5:   // All entry sequence candidates are explored
 6:   for $i = 0$ to $max - 1$ do
 7:     if t[i] is free then
 8:       // Find entry sequence to attend request $d$
 9:       Found = FindEntries( t, t[i], entries)
10:       if Found then
11:         // Obtain the entry group with more consecutive occupied entries
12:         greserved = BigGroupReserved(t)
13:         // Obtain the entry group with less consecutive free entries
14:         gfree = SmallGroupFree(t)
15:         // Sequence is selected if it is the best
16:         best = (greserved $<$ greservedmin) or
            ((greserved = greservedmin) and (gfree $>$ gfreemin)) or
            ((greserved = greservedmin) and (gfree = gfreemin) and
            (entries $<$ entriesmin))
17:         if best then
18:           greservedmin = greserved
19:           gfreesmin = gfree
20:           entriesmin = entries
21:         end if
22:       end if
23:     end if
24:   end for
25:   group = greservedmin
26:   entries = entriesmin
27:   free_entries = free_entries - entries
28:   Update( t, t[i], entries)          // Reserve entries for request $d$
29: end if
30: End Procedure CENRE
```

Fig. 4. An implementation of the CENRE approach.

Obviously, this algorithm is close to the optimum according to its capacity to locate requests in the table. The algorithm selects a new correct sequence if there are enough free entries in the table and there is no occupied sequence of entries greater than the distance requested. It is clear that in some cases this algorithm can use more entries than those strictly necessary. When the first entry of the sequence has been fixed, the others are selected following the distance requested, or the divisor of 64 that uses the same number of entries. For example, for a request of type 53, that needs 2 entries in the table, the latter are going to be met with a distance of 32. This is because the same number of entries is used, and in this way the distance is distributed equitably on the table. Following this distance the next entries of the sequence are selected, but if any of the following entries matches an occupied entry, the algorithm tries to use the previous one. If this is also occupied, the algorithm tries to use the second previous one, and so on.

5.1.2. *Management issues*

As we know, in the CENRE approach the possible distances can be grouped in 14 different categories. As all the traffic of the same SL uses the same VL, and therefore receives the same service, we are going to use a different SL for each one of the 14 different distances considered. In this way, there are still 2 remaining SLs that can be used for the traffic without QoS guarantee. For example, we could devote one of them to the PBE and BE traffic, and the other one to the CH traffic.

If we have 16 VLs then a different VL can be used for each SL. However, if we have less than 16 VLs some criterion must be chosen to join traffic from different SLs into the same VL. In this case, this traffic is going to receive the treatment of the most restrictive SL of all that share the VL. This is another decision about grouping maximum distances, and so fewer distances than 14 would be considered.

Another very important consideration to be taken into account to select a method for locating a request in the table is the later management of these requests. Obviously, first of all an algorithm is needed to locate a request of a certain distance in the table. That algorithm must be quick and able to make a good use of the available entries. As was stated, in order to do that, the algorithm should always leave the biggest possible holes in order to meet other later requests. Specifically, decreasing the lowest of the available holes must be avoided.

Furthermore, some information must be stored to be able to release the used entries when a connection finishes. We should try to make sure that the information needed for this task is the smallest amount possible. This is because we would need the data structures for that, and the number of connections met could be very large. If the variation of the distance between two consecutive entries of the sequence is allowed, it would be necessary to store specifically the entries which form the sequence in order to be able to release them when the connection finishes. However, if the sequence always keeps the same distance between two consecutive entries, the task is easier because it is only necessary to store the first entry and the separation of the sequence. It is therefore this characteristic that leads us to another way to classify the distances, and therefore the requests.

5.2. *Categorization based on Entries SYmmetry: CESY*

As has been indicated at the end of the previous section, considering the same distance between every couple of consecutive entries of the sequence can simplify the selection process and the later management of the entry sequence of entries. In this way, the CESY approach distributes the entries of the table according to an arithmetic progression, with the difference of the progression being the distance requested by the connection.

The arithmetic progressions that are symmetric in a table of 64 entries (2^6) are those which have, as difference of the progression, the divisors of 64. As 64 is a power of 2, the divisors are the power of 2 lower than or equal to 2^6, that are 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, and 2^6. These values represent requests of type 1, 2, 4, 8, 16, 32, and 64. The request of type 64 is when the request does not have any deadline or it is long enough. As was stated, request of distance 1 will not be considered.

With the CESY approach any request will be reduced to the corresponding power of 2 immediately below. Thus, a request of distance 11 is going to be treated as if it were of distance 8. Another request of distance 45 would be treated as if it were of distance 32, and so on. Let us analyze the possible situations, indicating the changes made and the consequences of this transformation:

- All the requests with distances between [32, 63] can be treated as if they were of type 32 because all of them need 2 entries. A request of type 32 can be located in a uniform and symmetric distribution with 2 entries separated between them for 32 units (an arithmetic progression of difference 32). In this way, locating them symmetrically with distance 32, we are treating the request as if it were of distance 32. This change does not affect the connections and we leave the table in a better situation to locate later other requests.
- For the requests with distance in the interval [16, 31] several cases can be considered. The requests with distance between 16 and 21 need 4 entries, and by treating them as a request of type 16 as in the previous case, no unfavorable situation arises. However, for the requests between 22 and 31 it will be enough with 3 entries to satisfy their demand, but we are treating them as if they were of type 16 using one additional entry to those necessary.

Table 1
Each one of the 63 distances considered, the categorization based on the power of 2 using the CESY approach, and the number of entries used more than those strictly necessary.

Maximum distance needed	Treated as	Exceeding entries used
2	2	0
3	2	10
4	4	0
5	4	3
6	4	5
7	4	6
8, 9	8	0
10	8	1
11, 12	8	2
13, 14, 15	8	3
16, . . ., 21	16	0
22, . . ., 31	16	1
32, . . ., 63	32	0
64	64	0

- For the requests with distance in the interval [8, 15] we have the following situations:
  - Requests of type 8 and 9 need 8 entries.
  - Request of type 10 needs 7 entries (one fewer than if we treat it as a request of distance 8).
  - The ones of distance 11 and 12 need 6 entries (two fewer).
  - The ones of type 13, 14, and 15 need 5 entries (three fewer).

  As can be seen, in some of these situations this approach uses more entries than those absolutely necessary. For example, changing the requests of type 13, 14, and 15 to a request of type 8, we are using 3 entries more than necessary.
- Requests in the interval [4, 7] are turned into requests of type 4 with 16 entries. However, the really necessary entries are:
  - Requests of type 5 need 13 entries (3 fewer).
  - Requests of type 6 require 11 entries (5 fewer).
  - Requests of type 7 need 10 entries (6 fewer).
- Finally, the most restrictive requests are in the interval [2, 3], where the request of type 3 (that could be located only with 22 entries) is turned into a request of type 2, and so requires 32 entries. In this case, the request would be using 10 entries more than necessary.

Summing up, all cases are shown in the Table 1. As can be seen, the most restrictive requests are a problem for the CESY approach. However, according to the delay requirements of current applications and the technology used, these more restrictive distances will be precisely the least demanded. In fact, today's applications tolerate latencies in the order of tens or hundreds of milliseconds [13]. These latencies imply distances bigger than 32 in most cases, even for a very long path crossing a lot of switches [8]. Thus, it seems that the distances most used in practice will be those in which this proposal uses up fewer entries, and thereby, making better use of the table.

5.2.1. *Selecting the sequence*

As we have indicated previously, if the CESY approach is used, the fact that the distance between every consecutive couple of entries of the sequence is the same simplifies the process to locate a certain request and the requests made later. Besides, dealing with a power of 2, the process is easier. So, the algorithm that can be implemented has a reduced complexity. When there is a new request of maximum distance $d = 2^i$, we must find a group of $\frac{64}{d}$ entries in the table such that two consecutive entries are separated at a maximum distance of d.

We have implemented a version of this algorithm, which is shown in the Fig. 5. The CESY approach is able to meet any request in the table if there are enough free entries. This is because the algorithm leaves the free entries situated in such a way that the most restrictive possible request can later be located, as is proved in [3].

```
1: Begin Procedure CESY (ibatable t, requests $d$)
2: entries = Obtain($d$)      // Obtain the number of entries to attend request d
3: if entries $\leq$ free_entries then
4:     // It's sure request $d$ can be attended
5:     first = FindFirst( t, $d$)     // Find the first entry for request $d$
6:     Update( t, first, entries) // Reserve entries for request $d$
7:     free_entries = free_entries - entries
8: end if
9: End Procedure CESY
```

Fig. 5. An implementation of the CESY approach.

The order in which the entries are examined has as objective to maximize the distance between two free consecutive entries that would remain in the table after carrying out the selection. In this way, the table remains in the optimum conditions to be able to meet later the most restrictive possible request.

In the proposed implementation, given a request of type *d*, the number of necessary entries to attend it is calculated using the function *Obtain*. The function *FindFirst* finds the first free entry in the table using an efficient search algorithm, trying to maximize the distance between two free consecutive entries that would remain in the table after carrying out the selection. In this way, the table remains in the optimum conditions to be able to meet later the most restrictive possible request. The function *Update* is used for reserving all entries needed to attend the request *d*. At the end, the number of free entries is updated.

5.2.2. *Management issues*

Using the CESY approach we could have a SL for each distance used (2, 4, 8, 16, 32, and 64). Besides, the SLs that are considered to have more connections (the ones with bigger distance) could be split up into some SLs now based on their mean bandwidth. Therefore, we are leaving the other SLs for the traffic without QoS guarantees (PBE, BE, and CH) and for the control traffic.

If there are enough VLs we can devote a different VL to each SL that we have just proposed. But, if there are not enough VLs then the SLs with the same maximum distance should be joined in the same SL. However, SLs with different maximum distance cannot be joined in the same VL, because we will then again require the features of each connection in order to use this information when the connections end, and this option has already been discarded. We would thus need a minimum of 8 VLs: 6 for the 6 distances allowed, another for the PBE, BE, and CH traffics, and yet another for the control traffic.

However, if we have fewer than 8 VLs per port, we must consider using fewer values for the maximum distance allowed. For example, if we have only 4 VLs, we would use one of them for the control traffic and another for the traffic without QoS requirement. The other two VLs could be used for the two maximum distances that we would want to consider. For example, we could use the distance with separation 64 (for the DB traffic and for the DBTS traffic with very low latency requirement) and the distance 16. The requests with lower distance should be rejected because it is not possible to admit them. Besides, the requests of distance 32 should be turned into requests of distance 16.

Furthermore, the CESY approach has another advantage, which allows us to make the later management of the requests easier, both to locate other new requests and to release a previously located request from the table. As all the consecutive entries of the sequence keep the same distance, we only need to know which is the first entry of the sequence and the distance between them.

5.3. *Comparing both models*

As we have already indicated, with the CESY proposal, in some cases we are using more entries in the table than those absolutely necessary. However, the CENRE approach can also use more entries than those strictly necessary. We are going to compare both models on the basis of the number of entries improperly used, and their complexity, taking into account their implementation and management. This comparison will serve to select one of them.

We have implemented the algorithm CENRE shown in the Fig. 4. As was stated, the CENRE approach is able to meet any request of distance *d* if there is no sequence of fulfilled entries greater than *d* in the table. This assumes that in some cases the distance between two consecutive entries should be modified in order to adapt itself to the available holes, though always respecting the maximum distance.

We have also implemented an algorithm for the CESY approach, which looks for a free sequence of entries in the table that turns any request in the closest power of 2. This algorithm is shown in the Fig. 5. This algorithm is able to locate a sequence of free entries in the table if there are enough free entries [3]. As was stated, the order in which the entries are examined has as objective to maximize the distance between two free consecutive entries that would remain in the table after carrying out the selection. In this way, the table remains in the optimum conditions to be able to later meet the most restrictive possible request.

Sequences of requests have been generated in order to try to meet them in the table using both methods, until the table is completed. These requests are of a certain distance randomly generated between 2 and 64, in this case, all of them with the same probability. If one request cannot be located in the table, it is discarded and another is generated. In principle, we are interested in computing how many entries are wasted in both cases. So, both algorithms use more entries in the table than those strictly necessary. For the CESY method this is due to the rounding up. However, for the CENRE approach it is due in some cases to lower distances being used than necessary because the entry that the algorithm wants to use is occupied.

According to the results obtained, the CESY approach uses up on average 8.78 entries, while the supposedly optimum algorithm CENRE uses up on average 1.96 entries. This results in a difference between both methods of almost 7 entries, which is significant.

These results have been obtained considering all types of connections equally probable, which is not very realistic. It is difficult to compute the real probability of each distance, because this will depend on the applications, but also on some aspects of the network (size, diameter, packet size, etc.), and on the switches (type of crossbar, number of ports, number of virtual lines, etc). As a simple approximation we have considered establishing the probability proportionally to the distance. In this way, a request of distance 64 has twice the probability of a request of distance 32, and 32 times more than a request of distance 2. Using this new scenario the test has been repeated. In this case, the CESY approach wastes on average 5.68 entries. On the other hand, the CENRE proposal wastes 0.86 entries. Therefore, the relative difference between them has decreased significantly.

It is clear that in real cases these probabilities will be even more disproportionate, the requests of big distances being more frequent. Even, for small or medium sized networks, it is quite probable that the distances lower than 16 are never requested. In this way, the proposal CESY of rounding to powers of 2 does not use up a lot of entries of the table, but has other very important advantages.

Another topic is the complexity of the filling-in algorithm. Obviously, the decision must be made based on the information the algorithm has when the request is made. Later requests are unknown at that moment. Regarding the CENRE model, depending on the decision taken for a request, other requests could be later met. But, obviously we cannot know which requests we will have in the future.

As an example, we are going to study the sequence of requests of distance 45, 8, 53, 61, 60, 55, 24, 3, and 9. Each request is located following the rules previously indicated in the CENRE approach. As a starting point, the first request (of distance 45) is met in the entries 32 and 64, thus having a separation of 32. The other requests are met following those rules. The final table is shown in Fig. 6(a). We can see that the last request (the one of distance 9) cannot be located because there is a consecutive sequence of occupied entries greater than 8. Specifically, for the situation shown in that figure, we have the sequences 28, 29, …, 36 and 63, 64, 0, 1, …, 8 that have length equal to or greater than the distance requested. So, the last request (the one of distance 9) cannot be met, although there are still enough free entries in the table (17 free entries). This is because the free entries are situated with an incorrect separation between two consecutive entries. A possible solution is to move the requests on the table when new requests are made.

However, if we know the complete sequence, we can locate the requests using also the CENRE approach in the correct entries in order for the other requests to be met. In this case, a possible final status of the table is shown in Fig. 6(b).

On the other hand, following the CESY approach which turns requests into powers of 2, these requests can always be met in the table. Specifically, using this strategy, the final status of the table is shown in Fig. 6(c). In this case there are still 2 free entries (entries number 19 and 51) with a distance of 32, that are able to meet a request later that is turned into that distance. So, in spite of the CESY algorithm using more entries than those strictly necessary, the free entries are left in a correct way in order to later meet the most restrictive possible request. A request, therefore, can always be located in the table if there are enough free entries.

After this deep study, finally, our proposal is to use the CESY approach that turns the requests into the lowest closest power of 2. We use up some entries, but we now know that this proposal has other worthy advantages. In the next section we are going to show the performance evaluation of this proposal when the network is very loaded. We will study the results that the applications with QoS requirements receive. Note that we do not show results using both categorization methods studied because once the connections are established, all of them would receive the same treatment in spite the way they were located. Thus, the only difference between both approaches is the number of connections they are able to establish, and it has already clearly been shown that the CESY approach is much better.

6. Performance evaluation

We have used simulation to evaluate the behavior of our proposals to provide QoS in InfiniBand. We start explaining the network and the traffic models we have used, and then we show the performance evaluation results.

6.1. Network model

We have used regular and irregular networks randomly generated, obtaining, in all the cases, similar results. Due to the lack of space, in this paper we are going to show only results for irregular networks. All switches have 8 ports, 4 of them having a host attached, and the other 4 are used for interconnection between switches. We have evaluated networks with sizes ranging from 8 to 64 switches. As each switch has 4 hosts attached, we have therefore used networks ranging from 32 to 256 hosts. Moreover, we have made some preliminary tests using the three link rates specified in InfiniBand. The results obtained for the main parameters are similar. This is because the link rate has no influence on the parameters studied for the QoS. Instead, the most important parameter is the load reached by the network. When the load is close to the maximum network throughput,

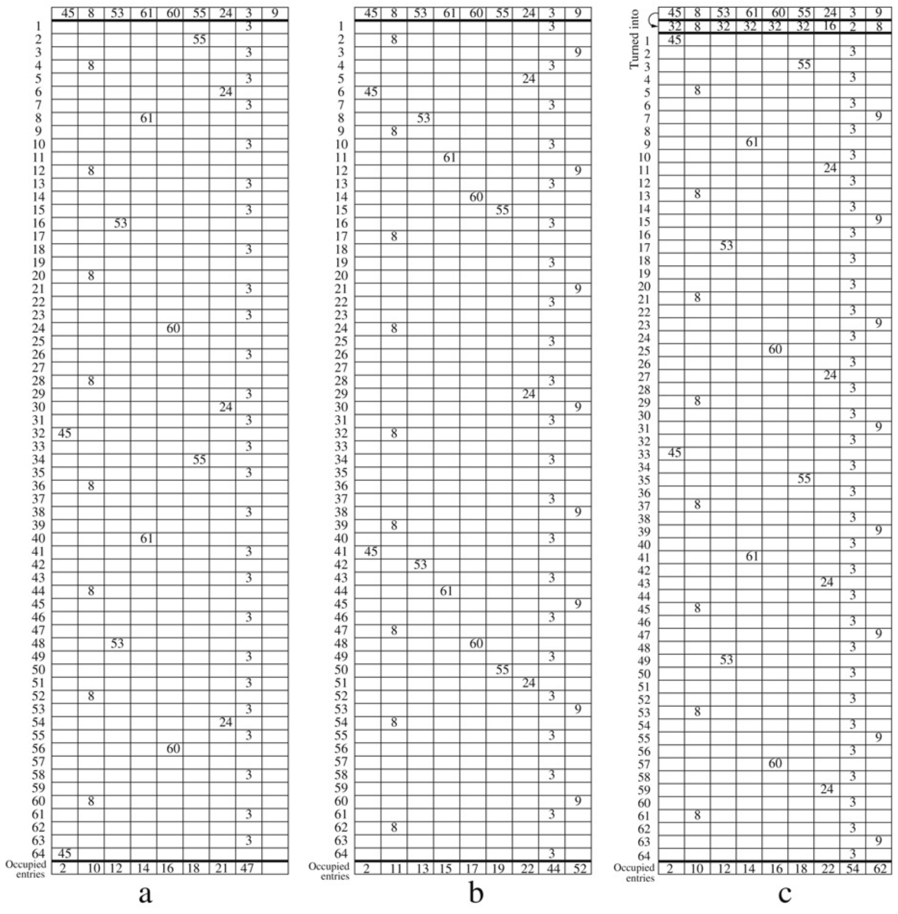

Fig. 6. Locating the sequence 45, 8, 53, 61, 60, 55, 24, 3, and 9, (a) and (b) using the CENRE approach, and (c) using the CESY approach.

a correct resource reservation and an accurate arbitration are critical regardless of link bandwidth. Without a correct arbitration, packets may suffer delays that could affect their QoS requirements. Therefore, we will only include in the paper the results for the link rate of 2.5 Gbps, but the conclusions can be extended to the other link rates specified in InfiniBand.

Both of input and output ports have 16 VLs in order to permit each SL to have its own VL. Each VL is large enough to store four whole packets, and two packet sizes have been considered: the smallest (256 bytes) and the largest (4096 bytes) allowed in InfiniBand. Each switch has a multiplexed crossbar. So, only a VL of each input (output) port can be transmitting (receiving) at the same time.

6.2. *Traffic model*

We have used 10 SLs for traffic needing QoS. Each one has different maximum delay (distance) and bandwidth requirements. The SLs used are shown in Table 2. For the most demanded distances, a division has been made based on the mean bandwidth of the connections. We have used CBR traffic, randomly generated among the bandwidth range of each SL.

In this paper we have used only CBR traffic because we want to offer guarantees. Therefore, we require the traffic to have a constant behavior. However, we also have results for VBR traffic (without offering guarantees) that are shown in [4]. Those results show that our proposal is also able to provide QoS to this kind of traffic, but obviously without guarantees.

The connections of each SL request a maximum distance between two consecutive entries in the high-priority table and a mean bandwidth in the range shown in the Table 2. Note that this is equivalent to requesting a maximum deadline and computing the maximum distance between two consecutive entries in the virtual lane arbitration tables, following the CESY approach.

Table 2
Features of the SLs used.

SL	Maximum distance	Bandwidth range (Mbps)
0	2	0.064–1.55
1	4	0.064–1.55
2	8	0.064–1.55
3	16	0.064–1.55
4	32	0.064–1.55
5		1.55–64
6	64	0.008–0.064
7		0.064–1.55
8		1.55–64
9		64–255

Each request is studied in each node in its path, and it is only accepted if there are available resources. Connections of the same SL are grouped in the same sequence of entries computing the total weight of the sequence based on the accumulated bandwidth of the connections sharing it. When the connection cannot be settled in a previously established sequence (or there is not a previous one), the algorithm looks for an empty sequence of entries in the high-priority arbitration table with the correct distance between its entries.

When no more connections can be established we start a transient period in order for the network to reach a stationary state. Once the transient period finishes, the steady state period begins,

Table 3
Traffic and utilization for different network sizes.

	Number of switches			
	8	16	32	64
Delivered traffic (Bytes/Cycle/Node)	0.7557	0.7258	0.7248	0.7227
Average utilization for host interfaces (%)	75.57	72.58	72.48	72.27
Average utilization for switch ports (%)	72.93	73.48	72.66	71.94
Average reservation for host interfaces (Mbps)	1926.85	1848.67	1834.04	1822.41
Average reservation for switch ports (Mbps)	1859.59	1871.75	1827.16	1837.62
Number of established connections	46495	111813	232854	284365

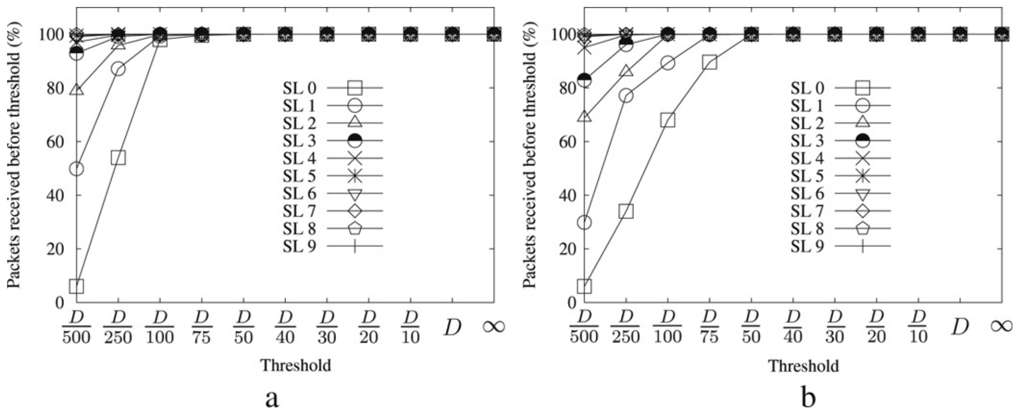

Fig. 7. Distribution of packet delay for (a) small packet size and (b) large packet size.

where we will gather results to be shown. The steady state period continues until the connection with a smaller mean bandwidth has received 100 packets.

Although the results for BE and CH traffic are not the main focus of this paper, we have reserved 20% of available bandwidth for these kinds of traffic, that would be attended by the low-priority table. So, connections would just be established up to 80% of the available bandwidth.

6.3. *Simulation results*

We can see in Table 3 the injected and delivered traffic (in bytes/cycle/node), the average utilization (in %) and the average bandwidth reserved (in Mbps) in host interfaces and switch ports, and the number of established connections. In all the cases we have used the small packet size. Note that the maximum utilization reachable is 80%, because the other 20% is reserved for BE and CH traffic. So, we are close to the maximum utilization. Obviously, we could achieve a higher utilization establishing more connections, but we have already made many attempts for each SL. Other connections that we could establish would be of SLs of small mean bandwidth because the network is already very loaded, and we think these new connections would not provide us with more information. So, we think that with this load we can study the network behavior in a quasi-fully loaded scenario.

Also note that the behavior is quite similar for all network sizes. Obviously, the number of established connection varies, but in all the cases the network reaches a high load. However, the network is not congested and it is able to deliver all the traffic that is injected. This is due to the accurate arbitration performed at the output ports and interfaces. As all the network sizes obtain similar results, in the following we will show only results for the network with 16 switches, and, thus 64 hosts.

We have also computed the percentages of packets that meet a certain deadline threshold. These thresholds are different for each connection and are related to their requested maximum deadline, which is the maximum delay that has been guaranteed to each connection. In the figures this deadline is referred to as D. The results for each SL are presented in Fig. 7, for both packet sizes. Note that results are quite similar for both packet sizes. In these figures we can see that all packets of all SLs arrive at their destination before their deadlines. However, the packets of SLs with stricter deadlines arrive at their destination close to their deadline, but in time to achieve their requirements. The results are related to the connection deadlines and so connections with lower latency requirements have much more time to achieve their deadlines.

We have also measured the average packet jitter. We have computed the percentage of packets received in several intervals related to their interarrival time. Obviously, these intervals are different for each connection. The results for each SL and small packet size are shown in Fig. 8. For large packet size results are quite similar. In all cases, we can see that almost all packets arrive in the central interval $[\frac{-IAT}{8}, \frac{IAT}{8}]$. For the SLs of the connections having the smallest mean bandwidth (SLs 0, 1, 2, 3, 4, 6, and 7), all packets arrive in the central interval. This is because these connections have a large interarrival time, large enough for all packets to arrive at their destinations. For the other SLs, with the biggest mean bandwidth, the jitter has a Gaussian distribution never exceeding $\pm IAT$.

Finally, for a given threshold, we have selected the connections having delivered the lowest and the highest percentage of packets before a threshold. In the figures these connections will be referred to as the worst and the best connections, respectively. We have selected a very tight threshold so that the percentage of packets meeting the deadline was lower than 100% in Fig. 7(a). In particular we have selected the threshold equal to $\frac{Deadline}{100}$. Note again that this threshold is different for each connection and it is based on its own maximum deadline. The results shown in Fig. 9 correspond to the small packet size and the SLs 0, 1, 2, and 3, which are the SLs with the highest deadline requirements. The results for the other SLs are even better than these shown in the figures. We can observe that, in all cases, even the packets of the worst connection arrive at their destination before their deadline. We can also see that in all cases the results are very similar for the best and the worst case. This is due to the fact that the arbitration tables are set in a correct way and all the connections receive a good and similar treatment.

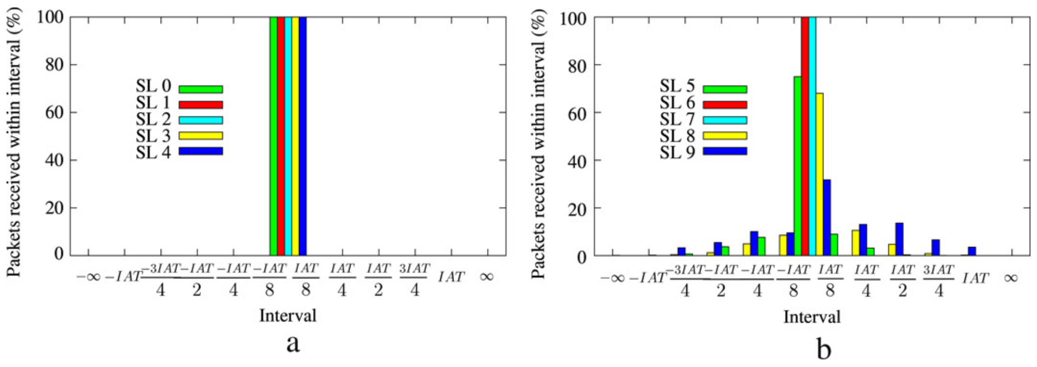

Fig. 8. Average packet jitter for small packet size (a) for SLs 0, 1, 2, 3, and 4, and (b) for SLs 5, 6, 7, 8, and 9.

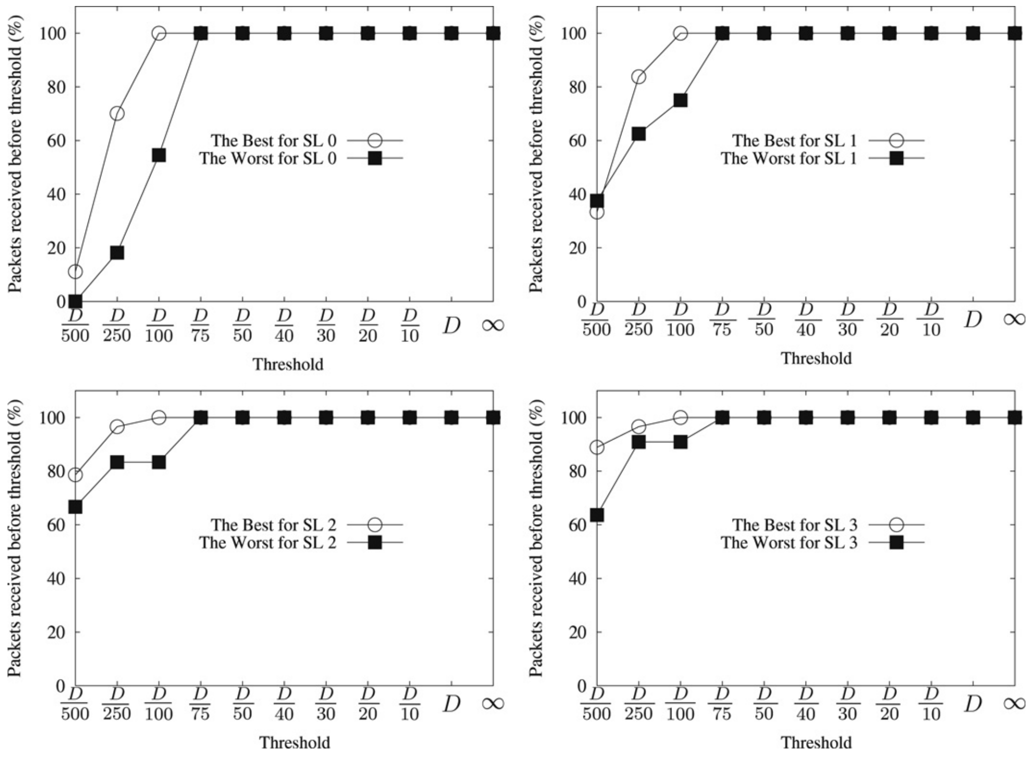

Fig. 9. The best and the worst connection for SLs with the strictest latency requirements.

7. Conclusions

InfiniBand is an industry-standard architecture for server I/O and interprocessor communication. InfiniBand has some mechanisms that, properly used, provide QoS to the applications. In [5], we proposed a new methodology to compute the virtual lane arbitration tables of InfiniBand for traffic, having both bandwidth and latency requirements. We evaluated this proposal, obtaining very good results. However, that proposal had the problem that no guarantees could be provided to traffic that uses the low-priority table if sources of traffic using the high-priority one used more bandwidth that they previously requested.

In this paper, we have presented a new approach that solves this problem. We treat all kinds of traffic in the same way, grouping it based on its latency requirements. Besides, all traffic with QoS requirements uses the high-priority table, and so, all of them have guaranteed what they have requested. If some source sends more than it has requested this will only affect the connections sharing the same VL, but the rest of the traffic in others VLs will achieve what they requested.

Moreover, we have proposed a novel way to assign the service levels of InfiniBand to the different traffic kinds. This proposal takes into account the requirements regarding maximum latency. Specifically, for a certain maximum latency, the maximum distance between any consecutive pair of entries in the arbitration table of the output ports in the switches is computed. We have studied the different possibilities to treat those maximum distances: the CENRE approach tries to use the exact number of required entries and the CESY approach categorizes the requests in the values that are divisors of 64. After studying both proposals in depth, we finally select the CESY approach due to its good use of the free entry sequences. With the CESY approach we are able to meet any request in the arbitration table if there are enough free entries because they are always placed to meet the most restrictive possible request.

Finally, we have tested this new proposal with traffic having very different requirements. We have used traffic with bandwidth and latency requirements, varying both of them in a large range. In all cases, the results obtained are very good, fulfilling the requested requirements easily. We think these are some important results, proving our methodology is very good to achieve QoS in InfiniBand environments.

Acknowledgments

This work has been jointly supported by the Spanish MEC and European Comission FEDER funds under grants "Consolider Ingenio 2010 CSD2006-00046"and "TIN2006-15516-C04-0X", and by the Junta de Comunidades de Castilla-La Mancha under Grant PBC08-0078-9856.

References

[1] F.J. Alfaro, J.L. Sánchez, J. Duato, A strategy to manage time sensitive traffic in InfiniBand, in: Proceedings of Workshop on Communication Architecture for Clusters, CAC'02, 2002, held in conjunction with IPDPS'02, Fort Lauderdale, Florida.

[2] F.J. Alfaro, J.L. Sánchez, J. Duato, C.R. Das, A strategy to compute the InfiniBand arbitration tables, in: Proceedings of International Parallel and Distributed Processing Symposium, IPDPS'02, 2002.

[3] F.J. Alfaro, J.L. Sánchez, M. Menduiña, J. Duato, Formalizing the fill-in of the InfiniBand arbitration table, Tech. Rep. DIAB-03-02-35, Dep. de Informática Univ. de Castilla-La Mancha, Mar. 2003. Available at http://www.info-ab.uclm.es/trep.php.

[4] F.J. Alfaro, J.L. Sánchez, L. Orozco, J. Duato, Performance evaluation of VBR traffic in InfiniBand, in: Proceedings of IEEE Canadian Conference on Electrical & Computer Engineering, CCECE'02, 2002, pp. 1532–1537.

[5] F.J. Alfaro, J.L. Sánchez, J. Duato, QoS in InfiniBand subnetworks, IEEE Transactions on Parallel and Distributed Systems 15 (9) (2004) 194–205.

[6] S. Blake, D. Back, M. Carlson, E. Davies, Z. Wang, W. Weiss, An architecture for differentiated services, Internet Request for Comment RFC 2475, Internet Engineering Task Force, Dec. 1998. URL http://www.ietf.org/rfc/rfc2275.txt.

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, Resource ReSerVation protocol (RVP) — version 1 functional specification, Internet Request for Comment RFC 2205, Internet Engineering Task Force, Sep. 1997. URL http://www.ietf.org/rfc/rfc2205.txt.

[8] P. Cuenca, Robust coding and transmission of variable rate mpeg-2 video signals over asynchronous transfer mode (ATM) networks, Ph.D. Thesis, Universidad Politécnica de Valencia, 1998.

[9] InfiniBand trade association, InfiniBand Architecture Specification, vol. 1. Release 1.2, Sep. 2004.

[10] InfiniBand^{TM} trade association, 1999. http://infinibandta.com.

[11] M. Katevenis, S. Sidiropoulos, C. Corcoubetis, Weighted round-robin cell multiplexing in a general-purpose ATM switch, IEEE Journal on Selected Areas in Communications.

[12] J. Pelissier, Providing quality of service over infiniband architecture fabrics, in: Proceedings of the 8th Symposium on Hot Interconnects, 2000. URL http://www.hoti.org/hoti8_thursday.html.

[13] M. Schwartz, D. Beaumount, Quality of service requirements for audio-visual multimedia services, ATM Forum ATM94-0640.

Francisco J. Alfaro received the MS degree in computer science from the University of Murcia in 1995 and the PhD degree from the University of Castilla-La Mancha in 2003. He is currently an assistant professor of computer architecture and technology in the Computer Systems Department at Castilla-La Mancha University. His research interests include high-performance local area networks, QoS, design of high-performance routers, and design of on-chip interconnection networks for multicore systems.

José L. Sánchez received the Ph.D. degree from the Technical University of Valencia, Spain, in 1998. Since November 1986 he has been a member of the Computer Systems Department (formerly Computer Science Department) at the University of Castilla-La Mancha. He is currently an associate professor of computer architecture and technology. His research interests include multicomputer systems, quality of service in high-speed networks, interconnection networks, parallel algorithms and simulation.

José Duato is Professor in the Department of Computer Engineering (DISCA) at UPV, Spain. His research interests include interconnection networks and multiprocessor architectures. He has published over 350 papers. His research results have been used in the design of the Alpha 21364 microprocessor, and the Cray T3E and IBM BlueGene/L supercomputers. Dr. Duato is the first author of the book "Interconnection Networks: An Engineering Approach". He served as associate editor of IEEE TPDS and IEEE TC, and is serving as associate editor of IEEE CAL. He was General Co-Chair of ICPP 2001, Program Chair of HPCA-10, and Program Co-Chair of ICPP 2005. Also, he served as Co-Chair, Steering Committee member, Vice-Chair, or Program Committee member in more than 55 conferences, including HPCA, ISCA, IPPS/SPDP, IPDPS, ICPP, ICDCS, Europar, and HiPC.