University of Castilla-La Mancha

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

Formalizing the Fill in of the InfiniBand Arbitration Table

by

Francisco J. Alfaro, José L. Sánchez, Manuel Menduiña, José Duato

Technical Report **#DIAB-03-02-35** March 2003

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

Contents

Formalizing the Fill in of the InfiniBand Arbitration Table [1]

Francisco J. Alfaro, José L. Sánchez, Manuel Menduiña
Dept. de Informática
Escuela Politécnica Superior
Universidad de Castilla-La Mancha
02071- Albacete, Spain
{falfaro, jsanchez, mmendu}@info-ab.uclm.es

José Duato
Dept. de Informática de
Sistemas y Computadores
Universidad Politécnica de Valencia
46071- Valencia, Spain
jduato@gap.upv.es

[1]This work was partly supported by the Spanish CICYT under Grant TIC2000-1151-C07 and Junta de Comunidades de Castilla-La Mancha under Grant PBC-02-008

Abstract

The InfiniBand Architecture (IBA) is a new industry-standard architecture for server I/O and interprocessor communication. InfiniBand is very likely to become the de facto standard in a few years. It is being 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 industry and academia. 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 on mean bandwidth requirements. Moreover, we propose a very effective strategy to compute the virtual lane arbitration tables for IBA switches. We evaluate our proposal with different network topologies. Performance results show that, with a correct treatment of each traffic class in the arbitration of the output port, every traffic class meets its QoS requirements.

1 Introduction

Input-output (I/O) buses have become a major bottleneck for disk accesses, especially in high-performance servers. While buses have the major advantage of simplicity, and have served the industry well up to this point, bus-based I/O systems do not use their underlying electrical technology well enough to provide high data transfer bandwidth out of a system to devices.

Despite recent upgrades, the most popular I/O bus (PCI bus) offers limited bandwidth, concurrency and reliability, and this limitation is unacceptable for a lot of actual applications and server systems. A single device failure can inhibit the correct operation of the bus itself, causing all the devices on the bus to become unavailable.

Several external interconnects, such as Fiber Channel, have been used to overcome these difficulties. However, they still enter the processing complex through an industry-standard bus, making it impossible to avoid the bottlenecks and low availability characteristic of standard I/O buses. As an example, the fastest PCI bus (PCI-X running at 133 MHz) does not provide enough bandwidth to feed a single 10 Gigabit Ethernet card.

This was the primary reason why a group of the most important companies joined together to develop a standard for communication between processing nodes and I/O devices as well as for interprocessor communication, known as InfiniBand [1]. The InfiniBand^{SM} Trade Association (IBTA) is a group of more than 200 companies founded in August 1999 to develop IBA. Membership is also open to Universities, research laboratories, and others. The IBTA is led by a Steering Committee whose members come from Dell, Compaq, HP, IBM, Intel, Microsoft, and Sun, co-chaired by IBM and Intel. Sponsor companies are 3Com, Cisco Systems, Fujitsu-Siemens, Hitachi, Adaptec, Lucent Technologies, NEC, and Nortel Networks. The first specification of the InfiniBand Architecture (release 1.0) was published in October 2000 [13].

On the other hand, most of the current networking products have tried to achieve maximum throughput and minimum latency, leaving aside other aspects like guarantee of bandwidth, bounded delivery deadline, bounded interarrival delays, etc. [18]. Many current applications need those characteristics that not all the current networks are able to provide. The current network that can best provide QoS is probably ATM [11, 12]. However, ATM focuses more upon wide area networks, and it can only be used with great difficulty in LAN environments. It is not suitable for connections between a processor and its devices since it introduces significant overhead.

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 [5].

Therefore, it would be important for InfiniBand to be able to satisfy both the applications that only need minimum latency and also those different applications that need other characteristics to satisfy their QoS requirements. 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 the packets with the most need for QoS.

InfiniBand distinguishes up to a maximum of 16 service levels, but it does not specify which characteristics will bear the traffic of those service levels. Nor does it specify how the arbitration table should be filled in, as it only specifies the form that this table should have, leaving the implementation of the table to the manufacturers' or users' consideration, according to the characteristics that they want to obtain.

Recently, an overview of a possible implementation of DiffServ over IBA has been described in [16]. 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.

In this report, we propose a classification of the different traffic types with QoS needs that improves the proposal made in [16], as well as a strategy to compute the arbitration tables for the IBA switch ports to obtain the QoS required by the applications.

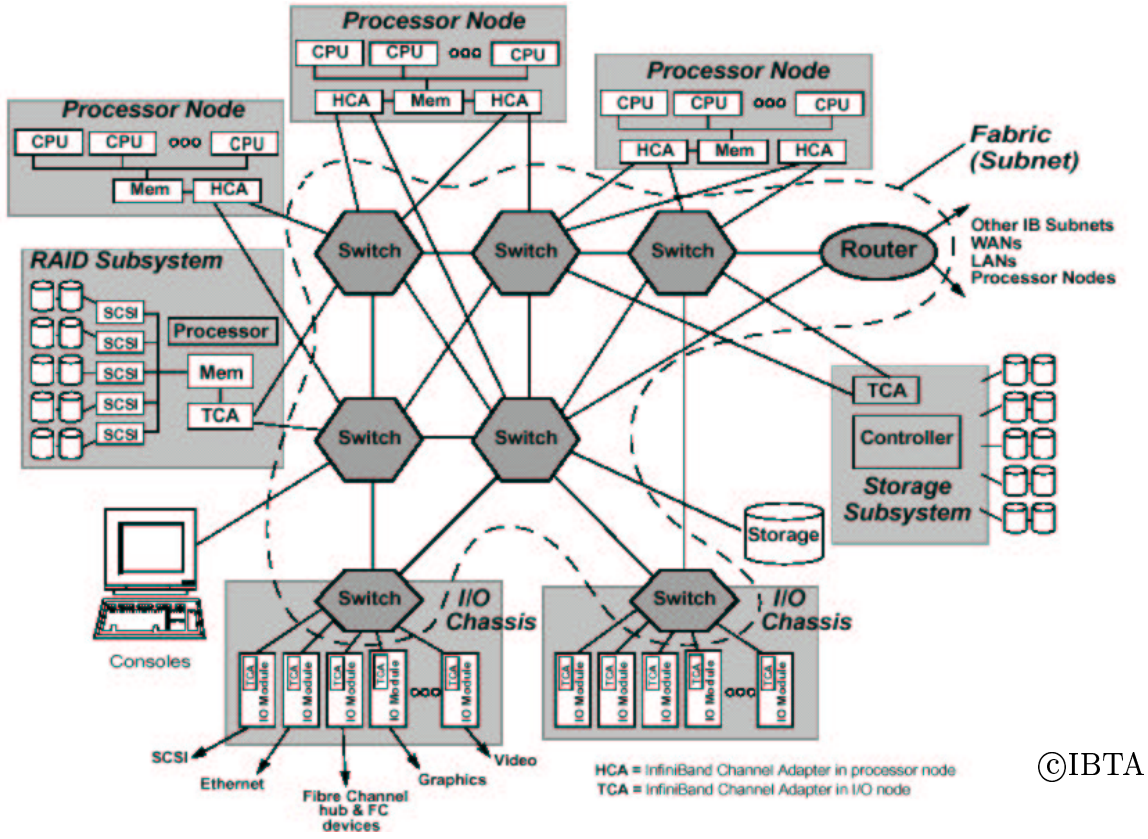


Figure 1: IBA System Area Network.

The structure of the report is as follows: Section 2 presents a summary of the general aspects in the specifications of InfiniBand. In Section 3, we explain the most important mechanisms that InfiniBand provides to support QoS. In Section 4, we study the traffic that generates the applications and its requirements. In this section, we also present our proposal to treat the different types of traffic based on its requirements.

In Section 5 we study in depth a way how of providing QoS in InfiniBand, with special emphasis laid on how to provide bandwidth and latency guarantee. Section 6 presents two different methods to look for a new sequence in the arbitration table. In Section 7 a formal model to represent a sequence of entries and its propositions is presented. In Section 8 we propose an algorithm to fill in the InfiniBand arbitration table based on the model proposed in the previous section. In this section, we also propose and prove some theorems, which show this algorithm achieves an optimal performance filling in the table when there are no connections removed from the table. In Sections 9 and 10, this previous restriction is eliminated, and the additional algorithms needed in this case are introduced. Section 11 presents the global management of the table when there are both new requests to be met in the table and requests to be released from the table, which could take place in any order. Finally, we present some conclusions and future work.

2 InfiniBand

The InfiniBand Architecture (IBA) Specification 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 communications and management infrastructure supporting both I/O and interprocessor communications for one or more computer systems (see Figure 1). The architecture is independent of the host operating system and processor platform.

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. Routing in IBA subnets is distributed, based on forwarding tables stored in each switch. IBA supports any topology defined by the user, including irregular ones, in order to provide flexibility and incremental expansion capability.

Processing nodes (either single processor or symmetric multiprocessor (SMPs)) are directly attached to a switch through a Host Channel Adapter (HCA). I/O devices can be attached to a switch through a Target Channel Adapter (TCA). While IBA Specification describes the behavior for HCA by IBA verbs, IBA does not specify the semantics of the consumer interface for a TCA. The IBA verbs are features that are defined to be available to host programs.

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 greater bandwidth. Currently, IBA 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), which correspond to 4-bit wide and 12-bit wide links, respectively. The width that will be supported by a link is vendor-specific.

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. The forwarding table can be linear, specifying an output port for each possible destination address up to a switch-specific limit, indexed by that address; or random, initialized by storing {destination, output port} pairs. The number of ports of a switch is vendor-specific, but is limited to 256 ports. Switches can be cascaded to form large networks. Switches may also optionally support multicast routing.

Routing between different subnets (across routers) is carried out 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 with Local Identifiers (LID) which allow 48K endnodes on a single subnet, the remaining 16K LID addresses being reserved for multicast.

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, 1KB, 2KB or 4KB. Each packet, even those for unreliable datagrams, contains two separate CRCs, one covering data that cannot change, and another covering data that changes in switches or routers, being 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. Obviously, for supporting QoS guarantee the applications must use reliable connections in order to be able to carry out resource allocation.

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 that discovers and initializes the network.

The interested reader is referred to the InfiniBand Specifications [13] for more details on InfiniBand. Other interesting papers that are good summaries of the official specifications are [17, 7].

3 IBA support for QoS

In this section we are going to describe the mechanisms that IBA provides to support QoS. Basically, IBA has three mechanisms that permit QoS to be supported: service levels, virtual lanes, and virtual lane arbitration for transmission over links.

3.1 Service levels

IBA defines a maximum of 16 service levels (SLs), but it does not specify what characteristics the traffic of each service level should have. Therefore, it depends on the implementation or on the administrator how to distribute the different existing traffic types among the SLs.

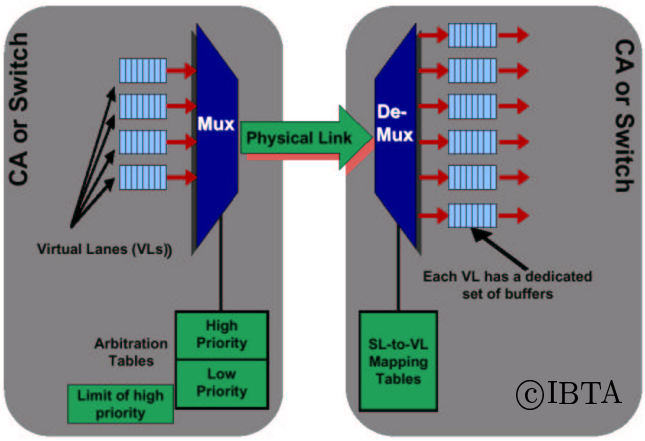


Figure 2: Operation of virtual lanes in a physical link.

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.

3.2 Virtual lanes

IBA ports support virtual lanes (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 (Figure 2). IBA ports have to support a minimum of two and a maximum of 16 virtual lanes ($VL_0 \ldots VL_{15}$). All ports support VL_{15}, which is reserved exclusively for subnet management, and must always have priority over data traffic in the other 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 Service Level (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.

3.3 Virtual lane arbitration

When more than two VLs are implemented, the priorities of the data lanes are defined by the *VLArbitrationTable*. This is for host channel adapter (HCA), input/output device channel adapter (target channel adapter, TCA), as well as for switches. This arbitration is only for data VLs, because VL_{15}, which transports control traffic, always has priority over any other VL.

The structure of the VLArbitrationTable is shown in Figure 3. Each VLArbitrationTable has 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. The arbitration tables implement weighted round-robin 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 the VL in question. This weight must be in the range of 0 to 255, and is always rounded up as a whole packet.

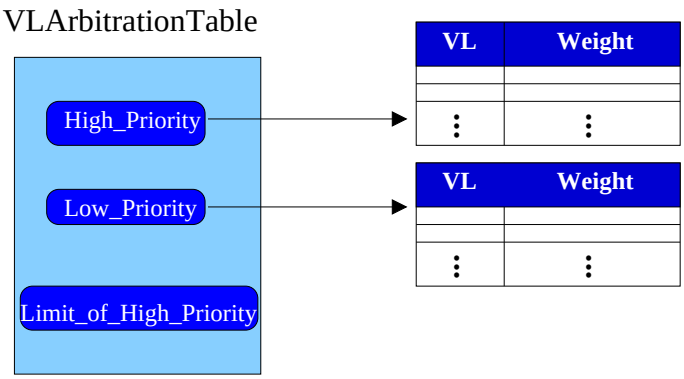


Figure 3: VLArbitrationTable structure.

A *LimitOfHighPriority* value specifies the maximum number of high-priority packets that can be sent 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 can be transmitted. If no high-priority packets are ready for transmission at a given time, low-priority packets can also be transmitted.

4 Traffic of applications

Using these mechanisms provided by IBA, and the different techniques that will be proposed (obviously, they must be compatible with the IBA specifications), the needs of QoS of the applications must be guaranteed. We are now going to study these specific necessities. For that purpose, we are first going to look at the applications that are used nowadays in these kind of environments.

4.1 Application needs

At the end of the 90's new multimedia applications emerged. These applications needed a great deal of bandwidth. This fact obliged the builders and researchers to make a great effort to achieve a large bandwidth. This may be one of the reasons why today network technologies are able to provide enough bandwidth. However, these new multimedia applications introduce new requirements for the transmission networks, which have not been considered until now.

For example, both video and audio produce a continuous traffic requiring availability of bandwidth during the entire transmission in order for the application not to be interrupted in the target node. For the video signal, an image must be shown in the target in a continuous way. Furthermore, these images after being generated in the source, must be given to the target between correct delay limits. The values of these

limits depend on the interaction level of the application. These limits can vary from 20 milliseconds to just a few seconds.

The most common technique used for this purpose is to impose delay levels according to the perceptive levels of the applications. When the delay suffered by a information packet is greater than this limit, the receiver discards it. This discarding causes a loss in the quality of the signal received (maybe increased by the dependencies among packets that introduce the signal codification). Besides, the transmission of the packet eventually discarded also supposes a waste of bandwidth [14].

The QoS requirements for the multimedia applications are based on their perceptive needs. So, the information must be transmitted by the network with some characteristics in order for the users to perceive the signal without degradation.

In the following, the perceptive requirements of some of the most representative multimedia applications will be studied [9]:

- Video-conference. This is an interactive real-time application. Its delay requirements are therefore very stringent. A delay greater than 200 milliseconds can be annoying for the users. Regarding the bandwidth requirements, the sequences usually only contain the talking head of a person. Thus, the motion is very small, and so they do not need a lot of bandwidth.

- Digital television broadcasting. These applications include movies, advertisements, news, etc.. In general, all of them have more motion than the video-conferences. This is the reason why the bandwidth requirements are greater for this application than for the video-conference. These bandwidth requirements can vary between 4 and 10 Mbps for the usual television, and from 20 to 80 Mbps for high definition television. However, the delay requirements are not as stringent as for the video-conference. These applications can admit delays of a few seconds, even for live transmissions. The only requirement for this kind of application has to do with the delay differences. Once the images start to appear in the target, they must be shown in a continuous way, without interruptions. This implies that the delay variations will be an important parameter, which must be taken into account.

- Video over demand. In this application the coded video signal is stored in a server. The signal is transmitted from the server to the user when the user demands it. So, in this case, it is the user who controls the starting of the transmission. In addition, the user could have available some interactive functions, like rewinding, forwarding, or pausing. In this case, the application is more interactive, and the requirements of delay are of hundreds of milliseconds [10].

- Telemedicine. This application is very interactive between the diagnostic center and the user. So, the delay requirements are very stringent. They can be about 100 milliseconds or even less. The resolution of the images is high, and quality degradation must be imperceptible. These applications usually need bandwidth from 1 to 20 Mbps.

Of course, there are many more multimedia applications, but the above mentioned are representative. Other multimedia applications usually have the requirements of some of these previously analyzed.

In general, we can draw the following conclusions about the needs of the multimedia applications [8]:

- The delay requirements are more stringent when the applications are more interactive.
- The bandwidth requirements depend on the quality of the signal and its characteristics (static images or with a lot of motion, image resolution, etc.).
- When the visualization of the signal in the target has begun, it is important that it should be continuous, in order for the user to perceive it without interruptions.

In the following section a traffic classification will be proposed. This traffic classification is based on the requirements of the applications of bandwidth and delay.

4.2 Traffic classification

Pelissier proposed in [16] a traffic classification for the different traffic flows. This classification is based on the requirements of the applications. We have assumed this classification, and we have introduced slight modifications that are described in [2]. Specifically, the categories we have proposed are:

- **DBTS** (Dedicated Bandwidth Time Sensitive). This category includes the kind of traffic that needs both a minimum bandwidth and a maximum delay guaranteed. As an example of this kind of traffic, we have the video-conference or the interactive audio.
- **DB** (Dedicated Bandwidth). This category includes that type of traffic only requiring a guarantee referring to the minimum bandwidth. Usually, this kind of traffic is not very sensitive to the latency. So, 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 or minimum bandwidth, but which we try to provide with a better treatment than the other best-effort traffic. This is the case, for example, with web traffic or the traffic accessing a data base. It is useful for this traffic to have priority over the other traffic without guarantees in order to improve its behavior.
- **BE** (Best-Effort). This kind of traffic does not need any type of guarantees of bandwidth or maximum latency. Usually it is enough to guarantee that, sooner or later, it will arrive at its destination. This task is usually carried out by the

upper levels of the software architecture. In today's 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. As an example of this kind of traffic, we have 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 propose 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 proposing to devote the high-priority arbitration table of InfiniBand to the DBTS traffic, and the low-priority arbitration table to the remaining categories. However, in [2] we pointed out that this proposal was problematic. If the sources that are generating the 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 in accordance with what they previously requested, no guarantees can be given to the traffic using only the low-priority table.

This behavior is caused because the $LimitOfHighPriority$ 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 of the high-priority table can transmit $LimitOfHighPriority \times 4096$ bytes before having to transmit a low-priority packet. In the best case, for the biggest packet allowed in InfiniBand (4096 bytes), this means that for a value of 1 we could transmit traffic, 50% coming from the VLs of the high-priority table, and the other 50% coming from the VLs of the low-priority table.

It is evident, therefore, that in order to provide guarantees for any kind of traffic we must put the VLs used in the high-priority table. In our case, we will place the VLs used for the DBTS traffic, but also the VLs used for DB traffic, in the high-priority table of InfiniBand ports and interfaces. Thus, we leave the low-priority table for the traffics without explicit guarantee requirements, which are PBE, BE, and CH. Furthermore, the bandwidth distribution will be performed when the connections are established.

Finally, in order to provide a connection for the required bandwidth, and as seen in [4], a weight must be assigned to the entries corresponding to the VLs that this connection is going to use. This weight depends on the required mean bandwidth of the connection.

5 Giving QoS in InfiniBand

In the previous section we have shown the mechanisms or supports provided by InfiniBand to support QoS. We have also studied today's applications requirements. So, now we can indicate the way we propose to guarantee these requirements. Obviously, our proposal is based on the mechanisms provided by the InfiniBand specifications. Specifically, the proposal is based on three points:

- Bandwidth and delay guarantee. When a host wants to establish a new connection it must decide which characteristics of bandwidth and maximum delay it wants to request from the network. This request consists of the mean bandwidth and maximum end-to-end delay which needs to be guaranteed.

- Resource reservation. To provide QoS guarantee for the applications in InfiniBand, we must make a previous resources reservation. This outline is similar to what is done with the RSVP protocol [6, 19]. This QoS guarantee can be carried out in two senses: bandwidth guarantee and/or maximum latency guarantee. To provide this guarantee the applications must use the Reliable Connection Service of the InfiniBand Transport Layer.

- Distributed control of connection establishment. The requesting host sends a message requesting a new connection with its QoS requirements. 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 specifications where the local agents are able to perform that task. The request 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, the task should be done 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 and the target host 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.

We shall now study in depth how we achieve guarantees for both bandwidth and delay, using correctly the mechanisms provided by InfiniBand.

5.1 Model to guarantee QoS in InfiniBand

Using the connection model of distributed admission, the message requesting a new connection travels through the network. This requesting message contains the require-

ments 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 step 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.

5.1.1 Bandwidth guarantee

In a distributed network admission model we can assume that both switches and hosts know the bandwidth previously reserved for other connections. So, 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.

If the network model was centralized, the only difference would be who possesses this information. In that case the Subnet Manager would have the information about the accumulated bandwidth in each port or interface. So, the Subnet Manager would study the admission of the new bandwidth request.

Anyway, according to the InfiniBand specifications, 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 [4].

However, while performing the previous weight computation, we must also do a rounding up. We would then be giving each connection more weight it needs. Repeated rounding up accumulates, and this would fill in the arbitration table with connections that do not run out the real bandwidth. 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 in 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. On the contrary, the idea is to fill in the low-priority table at the start-up. Its values would be based on the ratio priorities assigned among the different kinds of traffic without guarantees that are going to use the low-priority arbitration table. In our tests we have used the following rates: only one entry with a weight of one for the CH traffic, another entry but with a weight of 10 for BE traffic and finally 4 entries for the PBE traffic, each one with a weight of 255.

5.1.2 Delay guarantee

We perform the treatment of the latency distributing the total bounded maximum delay among the switches on the path, taking also into account the flying time on the links. So, 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 [2] 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 virtual channels 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 time 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 [2].

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 virtual lines (maximum time) that can be transmitted before one of its packets is transmitted.

5.1.3 Giving guarantee of bandwidth and delay at the same time

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

So, 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 by the bandwidth requirement ($\frac{w}{255}$) and those

resulting from the requirement of maximum distance between two consecutive entries 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, 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 $\frac{64 \times 255}{d}$.

  Therefore, each node will have a requirement table with the sequences 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 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 a new connection, but none of them can be used. The reason is that the accumulated weight for their entries does not permit this new connection to be placed 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. We only need to recompute the weights of its entries. The other two cases require a more elaborate process to find a new sequence. Besides, this should be performed in a efficient way.

6 Looking for a new sequence of entries in the table

As it was seen 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 entries that have not been used yet. In this section we propose a method that is

efficient and practical. It is efficient because this method selects the most suitable among the available sequences. It is practical because its implementation has a viable cost.

Moreover, we must take into account the management of the requests. This must be done in a dynamic way. So, in a continuous way new requests can be made and connections, previously established, can end. Thus, we could have at any moment more allocations or also releases of entries in the table. This could require the addition of new hardware or keeping some type of extra information.

It may, therefore, not be possible to design an optimal method due to its complexity. But this is usual in a lot of other topics in the area of computer architecture and engineering. In this case, perhaps, it is more useful to have an easier method with a efficiency close to the optimum.

In our case, 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 orders. 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 easy.

The distance between two consecutive entries of the sequence depends on the type of request. This is the maximum distance allowed by the application, but there would be lower distances allowed. In this case, the needs of the applications would be achieved easily, and the use of more entries that those absolutely necessary might be beneficial.

Therefore, to look for a sequence with the minimum number of entries (maximum distance allowed) or to consider more entries (distance lower than the requested) eventually leads to very different searching algorithms. So, before considering the design of the searching algorithm, we must fix the searching criterion for the entries separation.

6.1 Distance categorization

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. So, 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[1] of $\boldsymbol{d}$ between two consecutive entries of the sequence that its VL uses in the arbitration table of the switch output ports. 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.

[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.

Note, the maximum distance between two consecutive entries in the sequence is not the unique distance that can be accepted. Any lower separation would be acceptable. However, a separation lower than the maximum could use up more entries of the table than those strictly necessary, which in principle is not desirable.

According to the InfiniBand specifications, 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 we considered categorizing the traffic of the different applications based on its requirements of mean bandwidth. We did that independently of the needs that they have of bounded end-to-end delay, and therefore, of the maximum separation between two consecutive entries in the high-priority arbitration table. This forced us to join connections in the same service level, and thus in the same virtual line, the connections having a very different deadline requirement, but a similar mean bandwidth requirement.

Using this grouping we must give to all the traffic of the VL the same treatment regarding delay as the most restrictive connection requires. 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. This option is very problematic, however. We need to store all the information relating to the connections that each port is using. 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.

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, because all the connections that share that VL will have the same maximum distance requirements 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 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 specifications, 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 view of this, we will determine what is the set of different distances to be considered. We will consider two alternatives that are based on different criteria, and finally we will select one of them. The first consists in grouping the distances requiring the same number of entries. The second considers the symmetry to establish the different types of requests.

6.1.1 Categorization based on the number of entries

According to the size of the table there can be 64 different types of request. In any case, there are too many for them all to be considered and 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. Besides, a request of distance 1 would need all entries of the table, although this is non-viable. Note that this kind of request would reflect a need of latency that is almost zero, which, as we have explained in Section 4.1, is not true. So, specifically we could have the following 14 different types of maximum distances:

- 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.

- 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.
- Finally, only the requests of distance 2 require 32 entries.

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 done 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 the same distance between two consecutive entries of the sequence to be maintained. However, in the other cases it is not possible for any pair of consecutive entries to have the same distance.

6.1.1.1. Selecting the sequence

Anyway, we must select which entries exactly 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 *d* in the table is that a sequence, of length equal to or bigger than *d***, of consecutive fulfilled entries does not exist in the table.

We are going to assume the above criterion of trying to use 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 be located later.

As we can observe in both previous examples, the entries of the first (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 need entries along all the arbitration table with a distance lower than 15. So, for example, another valid sequence could be 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. For example, we could vary the distance between two consecutive entries, provided it is lower than 15. In this case, another valid sequence could be 1, 14, 24, 37, 50.

As we can see, 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 it should leave as many free holes[2] as possible. Besides, they should be in the most convenient positions in order to later locate other requests. Thus, **the algorithm that optimizes the placement 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. So, the most restrictive request is 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, we 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, which in principle is not desirable.

[2]set of free consecutive entries.

6.1.1.2. Management topics

As we know, in this model 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 SLs remaining 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 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. In order to do that, the algorithm should situate the requests as far apart as possible in order to leave a hole for later requests. As has been previously said, the necessary condition to be able to locate a request of distance d in the table is only that a fulfilled sequence, of length equal to or larger than d, of consecutive entries does not exist in the table. So, 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.

6.1.2 Categorization based on the symmetry of the entries

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 sequence of entries. In this way, the entries of the table will be distributed 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 $2^1, 2^2, 2^3, 2^4, 2^5$, and 2^6. These represent requests of type 2, 4, 8, 16, 32, and 64. The request of type 64 is when the request does not have any deadline delay or it is long enough.

With this new classification, 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. Being of type 32 we can locate them 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] we can consider them in several situations. 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 and treat them as if they were of type 16 using one additional entry to those necessary. This happens with other situations, which are discussed in the following items.

- For the request with distance in the interval [8, 15] we have the following situations:

  - The requests of type 8 and 9 need 8 entries.
  - The 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.

- The requests in the interval [4, 7] are turned into requests of type 4 with 16 entries. However, the really necessary entries are:

  - Type 5 need 13 entries (3 fewer).
  - Type 6 require 11 entries (5 fewer).
  - 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 would 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.

Maximum distance need	**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

Table 1: Each one of the 64 possible distances, the categorization based on the power of 2, and the number of entries used more than those strictly necessary.

According to the delay requirements of current applications and the technology used, these more restrictive distances will be precisely the least demanded. In fact, as seen in Section 4.1, today's applications tolerate latencies in the order of tens or hundreds of milliseconds. These latencies reach out to distances bigger than 32 in most cases, even for a very long path crossing a lot of switches [9]. So, 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.

6.1.2.1. Selecting a sequence

As we have indicated previously, 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. One version of this algorithm can be found in later sections and, as seen there, 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 latter be located.

6.1.2.2. Management topics

Using this model 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. So, for example, we could split the requests of distance 32 into two SLs of low and large mean bandwidth. The requests of distance 64 could be split up into four SLs of low, low-medium, medium-high, and high mean bandwidth. In this way, we would be using 10 SLs just for the traffic DBTS and DB, 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 distances cannot be joined in the same VL, because we will then require again the characteristics of each connection in order to use this information when the connections end. This option has already been discarded.

We would thus need a minimum of 8 VLs: 6 for the 6 distances allowed, another for the traffics PBE, BE, and CH, and yet another for the control traffic. Obviously, if we had 16 VLs per port, we would be able to use a different VL for each one of the 10 SLs previously defined. Besides, we would use the other three for each one of the user traffics without QoS needs: PBE, BE, and CH. This would allow the PBE, BE, and CH traffic to be treated differently. Finally, we would use another VL for 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, this proposal has another advantage, which allows 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 difference between them.

6.2 Comparing both models

As we have already indicated, with the proposal of grouping in powers of 2, in some cases we are using more entries in the table than those absolutely necessary. However, the other model 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 a version of the first model 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. This algorithm 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 group of consecutive entries is selected.

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.

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 algorithm using only distances that are power of 2 this is due to the rounding up. However, for the other method 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 method of rounding up to the lowest closest power of 2 uses up on average 8.78 entries. While the supposedly optimum algorithm 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 connections equally probable. However, as explained in Section 4.1, this is not 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 (like 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 method of rounding to a power of 2 wastes on average 5.68 entries. On the other hand, considering all the distances grouped in the 14 categories previously studied, the algorithm wastes 0.86 entries. Therefore, the relative difference between them has decreased significantly.

It is clear than 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 of rounding to powers of 2 does not use up a lot of entries of the table, but has other advantages.

Another topic is the complexity of the fill 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 first model proposed, depending on the decision taken for a request other requests could be later met. But, 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 first method. 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 Figure 4(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 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 Figure 4(b).

On the other hand, following the algorithm that turns requests into powers of 2,

these requests can always be met in the table. Specifically, with the algorithm that we are going to explain in Section 8, the final status of the table is shown in Figure 5. 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 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.

To sum up, our proposal regarding the treatment of the maximum latency is to turn it into the maximum distance between two consecutive entries in the arbitration table. This maximum distance is rounded to the lowest closest power of 2. In the following, we will always use this criterion to consider the distances. The next section presents in a detailed form the algorithm used to choose the sequence of entries to meet a certain request.

	45	8	53	61	60	55	24	3	9
1								3	
2						55			
3								3	
4		8							
5								3	
6							24		
7								3	
8				61					
9									
10								3	
11									
12		8							
13								3	
14									
15								3	
16			53						
17									
18								3	
19									
20		8							
21								3	
22									
23								3	
24					60				
25									
26								3	
27									
28		8							
29								3	
30							24		
31								3	
32	45								
33								3	
34						55			
35								3	
36		8							
37									
38								3	
39									
40				61					
41								3	
42									
43								3	
44		8							
45									
46								3	
47									
48			53						
49								3	
50									
51								3	
52		8							
53								3	
54							24		
55								3	
56					60				
57									
58								3	
59									
60		8							
61								3	
62									
63								3	
64	45								
Occupied entries	2	10	12	14	16	18	21	47	

(a)

	45	8	53	61	60	55	24	3	9
1								3	
2		8							
3									9
4								3	
5							24		
6	45								
7								3	
8			53						
9		8							
10								3	
11				61					
12									9
13								3	
14					60				
15						55			
16								3	
17		8							
18									
19								3	
20									
21									9
22								3	
23									
24		8							
25								3	
26									
27									
28								3	
29							24		
30									9
31								3	
32		8							
33									
34								3	
35									
36									
37								3	
38									9
39		8							
40								3	
41	45								
42			53						
43								3	
44				61					
45									9
46								3	
47		8							
48					60				
49								3	
50						55			
51							24		
52								3	
53									9
54		8							
55								3	
56									
57									
58								3	
59									
60									9
61								3	
62		8							
63									
64								3	
Occupied entries	2	11	13	15	17	19	22	44	52

(b)

Figure 4: Locating the sequence 45, 8, 53, 61, 60, 55, 24, 3, and 9 following the first method proposed.

	45	8	53	61	60	55	24	3	9
Turned into	32	8	32	32	32	32	16	2	8
1	45								
2								3	
3						55			
4								3	
5		8							
6								3	
7									9
8								3	
9				61					
10								3	
11							24		
12								3	
13		8							
14								3	
15									9
16								3	
17			53						
18								3	
19									
20								3	
21		8							
22								3	
23									9
24								3	
25					60				
26								3	
27							24		
28								3	
29		8							
30								3	
31									9
32								3	
33	45								
34								3	
35						55			
36								3	
37		8							
38								3	
39									9
40								3	
41				61					
42								3	
43							24		
44								3	
45		8							
46								3	
47									9
48								3	
49			53						
50								3	
51									
52								3	
53		8							
54								3	
55									9
56								3	
57					60				
58								3	
59							24		
60								3	
61		8							
62								3	
63									9
64								3	
Occupied entries	2	10	12	14	16	18	22	54	62

Figure 5: Locating the sequence 45, 8, 53, 61, 60, 55, 24, 3, and 9 following the method of power of 2.

7 Formal model for the InfiniBand arbitration table

As stated in the previous sections, we now propose a concrete algorithm to find a new sequence of free entries able to locate a connection request in the table. The number of entries in the table and the distance between two consecutive entries depend on the characteristics of the connections requested.

The treatment of the problem that we present basically consists in setting out an efficient algorithm able to select a set of free entries on the arbitration table for the request indicating a maximum separation between two consecutive entries. In order to develop it, we first give our initial hypothesis and definitions. This is because we need to establish the correct frame to later present the algorithm and its associated theorems.

Although the algorithm and all the treatments are focused on a specific frame such as InfiniBand, they can be generalized to any problem of finding a sequence of entries with a certain separation between two consecutive entries in a table. This general character can be achieved if we do not use the word *arbitration* when we are referring to the table, nor the word *connection* when we are talking about the origin of the requests of entries. We will therefore carry on in a general context, but obviously without forgetting the environment. We must simply know that the requests are originated by the connections, so that some requirements are guaranteed. Besides, the group of entries assigned to a request belongs to the arbitration table associated with the output ports and interfaces of the InfiniBand switches and hosts, respectively.

7.1 Definitions

For the purposes of later treatment, we define the following concepts:

- *Table*: Round list of 64 entries.
- *Entry*: Each one of the 64 parts composing a table.
- *Weight*: Numerical value assigned to each one of the entries of the table. It can vary from 0 to 255.
- *Status of a entry*: Situation of a entry of the table. The different situations can be free (weight=0) or occupied (weight$\neq$0).
- *Request*: A demand of a certain number of entries.
- *Distance*: Maximum separation allowed between two consecutive entries assigned to one request in the table.
- *Type of request*: Each one of the different kinds the request can be grouped into. They are based on the number of entries requested.

- *Group or sequence of entries*: The set of entries of the table with a fixed distance between two consecutive entries. In order to characterize a sequence of entries it will be enough to give the first entry and the distance between two consecutive entries.

7.2 Initial hypothesis

In the following, and unless stated to the contrary, the following hypothesis will be considered:

1. There are no request eliminations, so the table is filled in when new requests are received, and these requests are never removed. In other words, the entries could change from free status to occupied status, but it is not possible for an occupied entry to change to free.

2. It could be necessary to devote more than a group of entries to a set of requests of the same type.

3. The total weight associated with one request is distributed among the entries of the selected sequence so that the weight for the first entry of this sequence is always larger than or equal to the weight of the other entries of the sequence.

4. The distance d associated to one request will always be a power of 2 and it must be $1 \leq d \leq 64$. These are the different types of requests that we are going to consider. So, in the following, *distance* and *type of request* will be equivalent terms. Thus, the maximum distance d requested will be $d = 2^i$ where $i = 0, 1, 2, \ldots, 6$.

7.3 The model

For a table T, the sequence t_0, t_1, $\cdots$, t_{62}, t_{63} represents the entries of that table. According to the previous definitions, every t_i has an associated weight w_i whose value can vary from 0 to 255. We say t_i is free if $w_i = 0$, otherwise it is not free or occupied.

For convenience, and without loss of generality, consecutive entries of the table will not have identifiers with consecutive indexes. The allocation of identifiers to the entries is based on the application of the bit-reversal permutation to which it would be the usual numeration, and which would assign identifiers with consecutive indexes to consecutive entries. Remember that the bit-reversal permutation is such that $m = b_{n-1}b_{n-2}\ldots b_1 b_0$, $\Re(m) = b_0 b_1 \ldots b_{n-2}b_{n-1}$.

For the case of a table of 8 entries, Figure 6(a) shows a correlative numeration, while Figure 6(b) shows the new numeration of the entries.

It can be observed in Figure 6(b) that a request of distance 2 can only be met with the sequences $\{0, 2, 1, 3\}$ and $\{4, 6, 5, 7\}$. As among the elements of a set there is no established order[3] those sets are the same as $\{0, 1, 2, 3\}$ and $\{4, 5, 6, 7\}$, respectively.

[3] A set is a group of elements no repeated nor ordered.

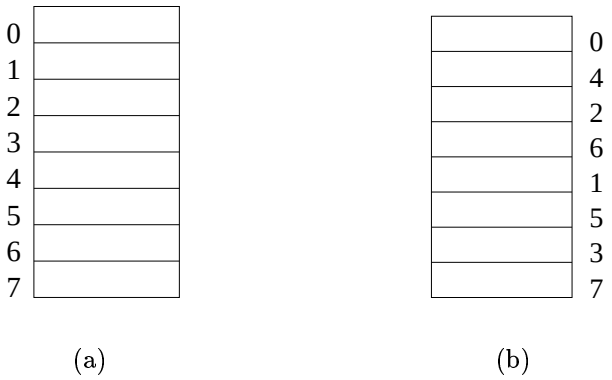


Figure 6: Table of 8 entries, (a) correlative numeration of the entries, (b) numeration based on the application of the bit-reversal permutation.

As can be seen, the entries of a set have a consecutive numeration. This is the reason why the numeration proposed has been used. Something similar happens for other distances.

The next definition of entry set uses the new numeration. Each set contains the necessary entries to be able to meet a request of a certain distance.

Definition 1 *Given a table T, for any request of type $d = 2^i$, $0 \leq i \leq 6$, we define the sets $E_{i,j}$, where $j = k \times 2^{6-i}$ and $0 \leq k < 2^i$, as*

$$E_{i,j} = \left\{ t_n \mid \quad j \leq n < j + 2^{6-i} \right\}$$

For convenience, we define the set $I_i = \{j \mid j = k \times 2^{6-i} \text{ con } 0 \leq k < 2^i\}$ that is composed by the possible values of j for a given i. So, when this index j is referred to it will be indicated as $j \in I_i$.

For the table of 8 entries of Figure 6(b), we have the following sets:

- Entries of the table with distance 1: $E_{0,0} = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$.
- Entries of the table with distance 2: $E_{1,0} = \{t_0, t_1, t_2, t_3\}$ and $E_{1,4} = \{t_4, t_5, t_6, t_7\}$.
- Entries of the table with distance 4: $E_{2,0} = \{t_0, t_1\}$, $E_{2,2} = \{t_2, t_3\}$, $E_{2,4} = \{t_4, t_5\}$ and $E_{2,6} = \{t_6, t_7\}$.
- Entries of the table with distance 8: $E_{3,0} = \{t_0\}$, $E_{3,1} = \{t_1\}$, $E_{3,2} = \{t_2\}$, $E_{3,3} = \{t_3\}$, $E_{3,4} = \{t_4\}$, $E_{3,5} = \{t_5\}$, $E_{3,6} = \{t_6\}$ and $E_{3,7} = \{t_7\}$.

Different sets for the same distance are disjoint among them, each of them having enough entries to meet a request of that distance. A binary tree structure can be established with all the sets. A set in a certain level i is the union of two disjoint sets belonging to the level $i + 1$. For the table with 8 entries shown in Figure 6(b), this tree can be shown in Figure 7.

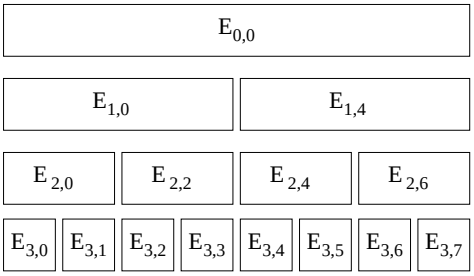


Figure 7: The set tree for the table of 8 entries in Figure 6(b).

In general, for a table of 64 entries, each set $E_{i,j}$ contains the entries of the table having a distance $d = 2^i$ between two consecutive entries. Obviously, these entries are able to locate a request of type d. In this case, the binary tree with all the sets is shown in Figure 8. The root of the tree is the set $E_{0,0}$. This set is the union of the sets $E_{1,0}$ and $E_{1,32}$, which are the roots of their two subtrees. These are also divided into two sets, and so on.

Although the basic terminology used in the rest of this paper will be based on sets, for the sake of clarity, we will, in some cases, also use terms related to binary trees.

The sets $E_{1,0}$ and $E_{1,32}$ are the only available sets for the distance $d = 2$. These sets have the following entries:

$$E_{1,0} = \{t_0 t_1 t_2 t_3 \ldots t_{30} t_{31}\} \quad \text{y} \quad E_{1,32} = \{t_{32} t_{33} t_{34} t_{35} \ldots t_{62} t_{63}\}$$

one set having the first half of the entries, and another one having the second half. Note that with the correlative numeration, these two sets correspond to a set with the even entries and another with the odd entries. These sets are disjoint between them. Each of them has enough entries to meet a request of type or distance 2. For the other types of requests, the corresponding sets $E_{i,j}$ could be obtained (Figure 8).

Definition 2 *A set $E_{i,j}$, $0 \leq i \leq 6$, and $j \in I_i$, is free if $w_k = 0 \quad \forall\ t_k \in E_{i,j}$.*

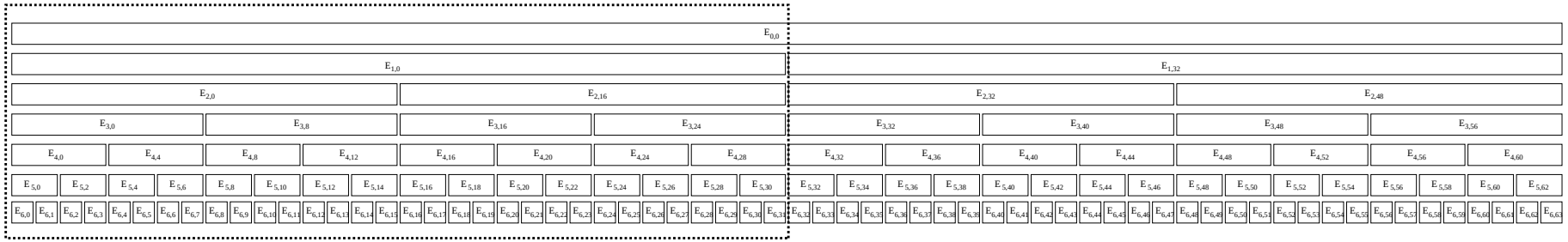
This, together with the definition of $E_{i,j}$ (Definition 1), implies that a free set $E_{i,j}$ is needed for meeting a request[4] of type $d = 2^i$, where $j \in I_i$.

Taking the previous definitions as a starting point, the following propositions can be considered:

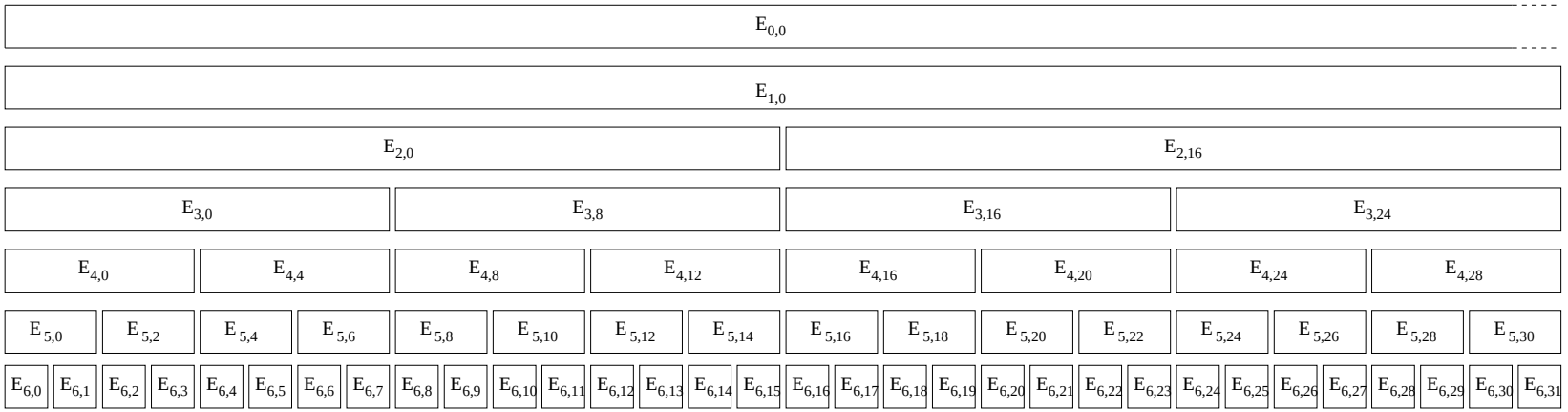
Proposition 1 *For any i and j, where $0 \leq i < 6$ and $j \in I_i$,*

$$E_{i,j} = E_{i+1,j} \cup E_{i+1,j+2^{6-(i+1)}}$$

[4]Remember that according to the previous section, we are going to consider only the case where a new request needs a new sequence of free entries.



(a)



(b)

Figure 8: Decomposition in a binary tree form of all the possible entry sets in the table, where each level corresponds to a different type of request: (a) the whole tree, (b) a zoom of the marked half in (a).

In binary tree terms, this proposition could be enunciated as *"the union of both children is the father"*.

Proof: According to the Definition 1,

$$\begin{aligned}
E_{i,j} &= \left\{t_n \mid \quad j \leq n < j + 2^{6-i}\right\} = \\
&= \left\{t_j, t_{j+1}, \ldots, t_{j+2^{6-(i+1)}-1}, t_{j+2^{6-(i+1)}}, t_{j+2^{6-(i+1)}+1}, \ldots, t_{j+2^{6-i}-1}\right\} = \\
&= \left\{t_j, t_{j+1}, \ldots, t_{j+2^{6-(i+1)}-1}\right\} \cup \left\{t_{j+2^{6-(i+1)}}, t_{j+2^{6-(i+1)}+1}, \ldots, t_{j+2^{6-i}-1}\right\} = \\
&= \left\{t_n \mid \quad j \leq n < j + 2^{6-(i+1)}\right\} \cup \left\{t_m \mid \quad j + 2^{6-(i+1)} \leq m < j + 2^{6-i}\right\} = \\
&= E_{i+1,j} \cup E_{i+1,j+2^{6-(i+1)}}
\end{aligned}$$

□

As a consequence of this proposition, with the entries that allow a request of type $d = 2^i$ to be met, two requests of type $d' = 2^{i+1}$ can be located.

From Definition 1, for a given level i, the sets $E_{i,j}$, $0 < i \leq 6$ and $j \in I_i$, are those obtained with $j = k \times 2^{6-i}$, and $0 \leq k < 2^i$. So, the values of j are:

$$0, 1 \times 2^{6-i}, 2 \times 2^{6-i}, 3 \times 2^{6-i}, \ldots, (2^i - 2) \times 2^{6-i}, (2^i - 1) \times 2^{6-i}$$

or

$$\begin{aligned}
0, 2 \times 2^{6-i}, 4 \times 2^{6-i}, \ldots, (2^i - 2) \times 2^{6-i} \quad \text{and} \\
1 \times 2^{6-i}, 3 \times 2^{6-i}, \ldots, (2^i - 1) \times 2^{6-i}
\end{aligned}$$

therefore

$$\left.\begin{array}{r} 2k \times 2^{6-i} \\ (2k+1) \times 2^{6-i} \end{array}\right\} \; 0 \leq k < 2^{i-1}$$

For a given k, both these expressions permit us to obtain the index that identifies two brother sets. The first one identifies the brother *situated on the left*, and the second one, the brother *situated on the right*.

The next function permits us to relate the identifiers of two brother sets. Specifically, when it is applied on the index that identifies whichever of the brothers it returns the index of the other brother.

$$Brother(p, i) = \left\{ \begin{array}{lll} p + 2^{6-i} & \text{if} & (p \bmod 2^{6-i+1}) = 0 \\ p - 2^{6-i} & \text{if} & (p \bmod 2^{6-i+1}) \neq 0 \end{array} \right.$$

As two brother sets are always placed on the same level, we will simplify the previous expression omitting the level. Therefore, the function is $Brother(p)$. We are going to test the validity of the function applying it to a pair of brothers. For a given level i $(0 < i \leq 6)$ the indexes of two brothers are j and $j + 2^{6-i}$.

For $2k \times 2^{6-i}$: $(2k \times 2^{6-i}) \bmod 2^{6-i+1} = (k \times 2^{6-i+1}) \bmod 2^{6-i+1} = 0$

So,

$Brother(2k \times 2^{6-i}) = (2k \times 2^{6-i}) + 2^{6-i} = (2k+1) \times 2^{6-i}$

For $(2k+1) \times 2^{6-i}$: $((2k+1) \times 2^{6-i}) \bmod 2^{6-i+1} =$

$(k \times 2^{6-i+1} + 2^{6-i}) \bmod 2^{6-i+1} = 2^{6-i} \neq 0$

So,

$Brother((2k+1) \times 2^{6-i}) = ((2k+1) \times 2^{6-i}) - 2^{6-i} = 2k \times 2^{6-i}$

A similar expression can be obtained in order to relate the identifier of a set with whichever of its ancestors. Therefore, for any set $E_{k,l}$, $0 < k \leq 6$ and $l \in I_k$, its index l is related to the index of whichever of its ancestors in level i according to the expression $(l \text{ div } 2^{6-i}) \times 2^{6-i}$.

In order to simplify the later use of this expression, we are going to define the function $Ancestor(l,i) = (l \text{ div } 2^{6-i}) \times 2^{6-i}$, as the ancestor of l in level i.

A generalization of the Proposition 1 emerges from being applied successively to each one of the descendants of a set $E_{i,j}$. This generalization indicates that with the entries of this set, that are meeting a request of type $d = 2^i$, two requests of type 2^{i+1} can be met, 4 of type 2^{i+2}, and so on. This is shown in the following proposition.

Proposition 2 *For any i and j, where $0 \leq i < 6$ and $j \in I_i$,*

$$E_{i,j} = \bigcup_{l=0}^{2^k-1} E_{i+k,j+l\times 2^{6-(i+k)}} \qquad k = 1, 2, \ldots, 6-i$$

Proof: The proof works by induction on k. For $k = 1$ we have $E_{i,j} = E_{i+1,j} \cup E_{i+1,j+2^{6-(i+1)}}$, which follows from Proposition 1. Let us now suppose it is true for $k-1$, and let's show it is also true for k. As it is true for $k-1$ then:

$$E_{i,j} = \bigcup_{l=0}^{2^{(k-1)}-1} E_{i+(k-1),j+l\times 2^{6-(i+k-1)}} \tag{1}$$

Applying Proposition 1 to each one of these sets $E_{i+k,j+l\times 2^{6-(i+k-1)}}$ we obtain:

$$\begin{aligned} E_{i+(k-1),j+l\times 2^{6-(i+k-1)}} &= E_{i+k,j+l\times 2^{6-(i+k-1)}} \cup E_{i+k,j+l\times 2^{6-(i+k-1)}+2^{6-(i+k)}} = \\ &= E_{i+k,j+l\times 2\times 2^{6-(i+k)}} \cup E_{i+k,j+(l\times 2+1)\times 2^{6-(i+k)}} \end{aligned}$$

and replacing it in Expression 1 we have:

$$E_{i,j} = \bigcup_{l=0}^{2^{(k-1)}-1} E_{i+k,j+l\times 2\times 2^{6-(i+k)}} \cup E_{i+k,j+(l\times 2+1)\times 2^{6-(i+k)}}$$

For the range of values of l, $\left[0\ ..\ 2^{(k-1)}-1\right]$, the expressions $l\times 2$ and $(l\times 2+1)$ include all the values in the range $\left[0\ ..\ 2^k-1\right]$ because:

$$\begin{aligned} l\times 2 \quad &\text{includes} \quad 0,2,4,\ldots,2(2^{(k-1)}-1) = 0,2,4,\ldots,2^k-2 \\ l\times 2+1 \quad &\text{includes} \quad 1,3,5,\ldots,2(2^{(k-1)}-1)+1 = 1,3,5,\ldots,2^k-1 \end{aligned}$$

Therefore,

$$\begin{aligned} E_{i,j} &= \bigcup_{l=0}^{2^{(k-1)}-1} E_{i+k,j+l\times 2\times 2^{6-(i+k)}} \cup E_{i+k,j+(l\times 2+1)\times 2^{6-(i+k)}} = \\ &= \bigcup_{l=0}^{2^k-1} E_{i+k,j+l\times 2^{6-(i+k)}} \end{aligned}$$

as we wanted to prove.

□

From this previous proposition it is easy to test that, if $E_{i,j} = \bigcup E_{i+k,m}$, then $m \geq j$. This means that the index of whichever of the descendant sets, in any level, of a certain set is always greater than or equal to its index.

Proposition 3 *The available sequences of entries for a distance $d = 2^i$ are independent among them. In a more formal way:*

$$E_{i,j} \cap E_{i,l} = \phi \quad \forall\ l \neq j,\ 0 < i \leq 6 \text{ and } j,l \in I_i$$

In binary tree terms this proposition could be enunciated as *"the nodes of the tree located in the same level are disjoint of each other".*

Proof: The entries corresponding to these two sets are:

$$\begin{aligned} E_{i,j} &= \{t_n \mid \ j \leq n < j+2^{6-i}\} \\ E_{i,l} &= \{t_m \mid \ l \leq m < l+2^{6-i}\} \end{aligned} \quad \left\{ \begin{array}{l} j = k\times 2^{6-i} \\ l = t\times 2^{6-i} \\ 0 \leq k,t < 2^i \end{array} \right.$$

But

$$\left.\begin{array}{l} j \leq n < j+2^{6-i} \\ l \leq m < l+2^{6-i} \end{array}\right\} \Longrightarrow \left\{ \begin{array}{l} k\times 2^{6-i} \leq n < (k+1)\times 2^{6-i} \\ t\times 2^{6-i} \leq m < (t+1)\times 2^{6-i} \end{array} \right.$$

As $j \neq l$ then $k \neq t$ and so $n \neq m$, therefore

$$E_{i,j} \cap E_{i,l} = \phi$$

□

Proposition 4 *Each available set of entries $E_{i,j}$ for a distance $d = 2^i$ is disjoint with all the other available sets of entries for a distance $d' = 2^{i+1}$ that fail Proposition 1 about $E_{i,j}$. More formally:*

$$E_{i,j} \cap E_{i+1,l} = \phi \qquad \forall\ l \text{ such that } (l \neq j)\ y\ (l \neq j + 2^{6-(i+1)}),\ \begin{cases} 0 < i < 6 \\ j \in I_i \\ l \in I_{(i+1)} \end{cases}$$

In binary tree terms, this proposition can be enunciated as *"a node is disjoint with any other that is deeper except with its descendants".*

Proof: According to Proposition 3 we know that $E_{i,j} \cap E_{i,l} = \phi, \forall j \neq l$. On the other hand, from Proposition 1, $E_{i,l} = E_{i+1,l} \cup E_{i+1,l+2^{6-(i+1)}}$. Combining both expressions we have:

$$\begin{aligned} E_{i,j} \cap E_{i,l} = \phi &\iff E_{i,j} \cap (E_{i+1,l} \cup E_{i+1,l+2^{6-(i+1)}}) = \phi \\ &\iff (E_{i,j} \cap E_{i+1,l}) \cup (E_{i,j} \cap E_{i+1,l+2^{6-(i+1)}}) = \phi \\ &\implies \begin{cases} (E_{i,j} \cap E_{i+1,l}) = \phi \\ (E_{i,j} \cap E_{i+1,l+2^{6-(i+1)}}) = \phi \end{cases} \end{aligned}$$

as we wanted to prove.

□

Proposition 5 *Let us consider a free set $E_{i,l}$ and another non-free set $E_{i,m}$, $l \neq m$, $0 < i \leq 6$ and $l, m \in I_i$, whose occupied entries are used to locate completely requests of type $d \geq 2^i$. Then, all those requests can be met completely with entries of $E_{i,l}$, thus leaving free the set $E_{i,m}$.*

Proof: If all the entries of the set $E_{i,m}$ are used to locate completely a request of type $d = 2^i$, this request could also be met with the entries of any free set $E_{i,j}$, $j \neq m$, and specifically with the entries of the set $E_{i,l}$.

If all or part of the entries of the set $E_{i,m}$ are being used to meet completely requests of type $d > 2^i$, this means that we are using some of the sets $E_{i+k,m+n\times 2^{6-(i+k)}}$, $n = 0, \ldots, 2^k - 1$ such that $E_{i,m} = \bigcup_{n=0}^{2^k-1} E_{i+k,m+n\times 2^{6-(i+k)}}$. In order to meet those requests other free sets $E_{i+k,j+n\times 2^{6-(i+k)}}$, $j \neq m$, can also be used. Specifically, the sets $E_{i+k,l+n\times 2^{6-(i+k)}}$, $n = 0, \ldots, 2^k - 1$ and $k = 1, \ldots, 6 - i$, can be used, because they are free sets as follows from $E_{i,l}$ is free and Proposition 2.

□

Definition 3 *A set $E_{i,j}$ is* **singular** *if this set is free but the set $E_{i,n}$ is not free, with $0 < i \leq 6$, $n = Brother(j)$ and $j, n \in I_i$. Therefore, a set $E_{i,j}$ is singular if it is free but its brother is not free.*

As a consequence, and using binary tree terminology, all the ancestors, including its father, of a singular set are not free.

Definition 4 *We say a table is normalized if* $\forall i, 0 \leq i \leq 6$, *at most there is a singular* $E_{i,j}$, *with* $j \in I_i$.

According to this definition, an empty table and a full table are normalized, because there is no level with a singular set.

Among the previously defined sets, a partial order relation can be defined, based on their position in the binary tree.

Definition 5 *Let* $\mathcal{E}$ *be the set made up of all the sets* $E_{i,j}$, $0 \leq i \leq 6$ *and* $j \in I_i$. *In* $\mathcal{E} \times \mathcal{E}$ *the following binary relation is defined:*

$$E_{k,l} \blacktriangleleft E_{i,j} \Longleftrightarrow \begin{cases} i \leq k \\ l < j \end{cases}$$

$E_{k,l} \blacktriangleleft E_{i,j}$ *is read as* $E_{k,l}$ *is placed more at the left and on a deeper level or the same, than the set* $E_{i,j}$ *in the binary tree of the Figure 8.*

As not all the sets $E_{i,j}$ are comparable (for example, $E_{3,8} \not\blacktriangleleft E_{4,12}$ and $E_{4,12} \not\blacktriangleleft E_{3,8}$) this relation establishes a strict partial order relation, and so, $\mathcal{E}$ is a set partially ordered.

Let us show new propositions that are based on the Definition 5.

Proposition 6 *For* $E_{k,l}$ *and* $E_{k,m}$, *where* $0 < k \leq 6$ *and* $l, m \in I_k$, *such that* $E_{k,l} \blacktriangleleft E_{k,m}$, *if* $p \neq q$, *with* $p = Ancestor(l, i)$, $q = Ancestor(m, i)$ *and* $0 < i < k$, *then* $E_{i,p} \blacktriangleleft E_{i,q}$.

That means, if two sets are related, their respective ancestors on whichever level, up to the level immediately before the level of the first ancestor in common, are also related between them.

Proof: In order for $E_{i,p} \blacktriangleleft E_{i,q}$ both conditions of the Definition 5 must be true. Therefore, $i \leq i$ and $p < q$. The first condition is trivial. If we look at the second, however, because $E_{k.l} \blacktriangleleft E_{k,m}$, then $l < m$, and so $(l \text{ div } 2^{6-i}) \leq (m \text{ div } 2^{6-i})$, and $(l \text{ div } 2^{6-i}) \times 2^{6-i} \leq (m \text{ div } 2^{6-i}) \times 2^{6-i}$, thus, $p \leq q$.

However, from definition, $p \neq q$, and so $p < q$, which means that $E_{i,p} \blacktriangleleft E_{i,q}$.

□

Proposition 7 *For the sets* $E_{k,l}$ *and* $E_{k,m}$, *with* $0 < k \leq 6$ *and* $l, m \in I_k$, *such that* $E_{k,l} \blacktriangleleft E_{k,m}$, *if* $i < k$ *and* $n \neq j$, *with* $n = Ancestor(l, i)$, $j = Ancestor(m, i)$, *then* $E_{k,l} \blacktriangleleft E_{i,j}$.

This means, if two sets are related, then the ancestor of the second one on whichever level, up to the level of the first ancestor in common, is related to the first.

Proof: As $E_{k,l} \blacktriangleleft E_{k,m}$, then $l < m$. Besides, as $i < k$, and $n \neq j$, from Proposition 6 then $E_{i,n} \blacktriangleleft E_{i,j}$.

On the other hand, as $n \neq j$, from Proposition 3, then $E_{i,n} \cap E_{i,j} = \phi$. But, as $E_{k,l} \subset E_{i,n}$ then it is also true $E_{k,l} \cap E_{i,j} = \phi$.

Let us suppose that $E_{k,l} \not\blacktriangleleft E_{i,j}$, and so $l \geq j$. It is also true that $l < m$, and so $j \leq l < m$. As $E_{k,m} \subset E_{i,j}$, and following on from the consequence of Proposition 2, which relates the indexes of a set and its descendants, we would have $E_{k,l} \subset E_{i,j}$, which is a contradiction, because $E_{k,l} \cap E_{i,j} = \phi$. Therefore, $E_{k,l} \blacktriangleleft E_{i,j}$.

□

Proposition 8 *For $E_{k,l}$ and $E_{i,j}$, where $0 < i < k \leq 6$, $l \in I_k$ and $j \in I_i$, such that $E_{k,l} \blacktriangleleft E_{i,j}$, then $E_{t,m} \blacktriangleleft E_{i,j}$, being $i \leq t < k$ and $m = Ancestor(l,t)$.*

This means, if two sets of levels k and i, where $i < k$ are related, then all the ancestors up to the level i of the set of the level k, are related with the set of the level i.

Proof: In order for $E_{t,m} \blacktriangleleft E_{i,j}$, according to Definition 5, $i \leq t$ and $m < j$. The first condition is true according to the initial hypothesis. Let us study the second condition. Because $E_{k,l} \blacktriangleleft E_{i,j}$ then $l < j$, and as a consequence

$$m = (l \text{ div } 2^{6-t}) \times 2^{6-t} \leq (j \text{ div } 2^{6-t}) \times 2^{6-t} \tag{2}$$

As $j \in I_i$, $j = p \times 2^{6-i}$, with $0 \leq p < 2^i$. As $i \leq t$, then $t = i + x$, $x \geq 0$. Thus:

$$\begin{aligned} j \bmod 2^{6-t} &= (p \times 2^{6-i}) \bmod 2^{6-t} = (p \times 2^{6-t+x}) \bmod 2^{6-t} = \\ &= (p \times 2^x \times 2^{6-t}) \bmod 2^{6-t} = 0 \end{aligned}$$

As $(j \bmod 2^{6-t}) = 0$, then $(j \text{ div } 2^{6-t}) \times 2^{6-t} = j$, and so, from (2), $m \leq j$.

Moreover, as $m = Ancestor(l,t)$, following on from the consequence of Proposition 2, which relates the indexes of a set and its descendant sets, then $m \leq l$. So, $m \leq j$ and $m \leq l$, in addition to $l < j$. If $m = j$ we would have $l < j = m$, which is a contradiction. Therefore, $m \neq j$, and so it should be true that $m < j$.

Thus, $i \leq t$ and $m < j$, and so $E_{t,m} \blacktriangleleft E_{i,j}$, with $i \leq t < k$.

□

Definition 6 *A table is ordered if $\forall E_{i,j}$ and $\forall E_{k,l}$, both singular sets, where $0 < i, k \leq 6$, $0 \leq j < 2^i$ and $0 \leq l < 2^k$, if $i \leq k$, then $E_{k,l} \blacktriangleleft E_{i,j}$.*

We have seen the model that we are using, some related definitions, and some derived propositions. In the following section, we will propose the algorithm to look for a new sequence of entries in the table that are situated with a certain distance between two consecutive entries. Firstly, we will enunciate the algorithm, and later some theorems that could be obtained from it.

8 Algorithm for filling in the table

When there is a new request of maximum distance $d = 2^i$, we must find a group of $\frac{64}{d}$ entries of the table with two consecutive entries between them at a maximum distance of d. Obviously, this is the same as finding a free set $E_{i,j}$. For that purpose, we will apply an algorithm. First we give an informal description of this algorithm in order later to formally propose it, and validate it with some theorems and their corresponding proofs.

For a request of distance $d = 2^i$, the algorithm examines, in a certain order, all the possible sets $E_{i,j}$ for this type of request. The first one of these sets having all its entries free is selected.

The order in which the sets are examined has as objective to maximize the distance between two free consecutive entries remaining in the table after the selection. In this way, the table is left in the best possible conditions to meet later the most restrictive request possible.

The algorithm examines the sets $E_{i,j}$ from left to right according to the location of the Figure 8. In a more formal way, the algorithm can be enunciated as:

For a new request of maximum distance $d = 2^i$, the algorithm selects the first free set $E_{i,j}$ of the sequence

$$E_{i,0}, E_{i,1\times 2^{6-i}}, E_{i,2\times 2^{6-i}}, E_{i,3\times 2^{6-i}}, \ldots, E_{i,(2^i-1)\times 2^{6-i}}$$

It should be noted that the apparent simplicity of this algorithm (to examine the sets $E_{i,j}$ from left to right) has been achieved because the numeration of the entries of the table uses the permutation bit-reversal, instead of the usual correlative numeration.

This algorithm has some characteristics that make it very efficient for filling in the table with a series of requests with requirements about maximum distance. We are going to show these characteristics by proving some theorems.

Theorem 1 *If there is a group of free entries in the table with the requested distance $d = 2^i$, $0 \leq i \leq 6$, between two consecutive entries, then the algorithm finds it.*

Proof: If there is a free group of entries in the table such that whichever two consecutive entries are at a distance $d = 2^i$, that means there is a free set $E_{i,k}$ with $k \in I_i$. From the definition of the algorithm, which consecutively inspects all the sets $E_{i,j}$ until a free one is found, it is guaranteed this set $E_{i,k}$ will be finally found.

□

Theorem 2 *After applying the filling in algorithm, the table is normalized. That means, the filling in algorithm leaves, on whichever level of the tree of the Figure 8, at most a singular set.*

Proof: Let us suppose this is not true, and so, $\exists i$, $1 < i \leq 6$, this level having more than one singular set. Let us suppose there are two, specifically $E_{i,k}$ and $E_{i,l}$, with $k, l \in I_i$ and $k < l$. Because $E_{i,l}$ is singular, then $E_{i,n}$ is not free, where $n = Brother(l)$. As $k < l$, and $E_{i,k}$ and $E_{i,l}$ are singular sets, then $Ancestor(k, i-1) \neq Ancestor(l, i-1)$, and so it is also true that $k < n$. Therefore, the set $E_{i,k}$ is placed at the left of the set $E_{i,n}$.

However, this situation is not possible, because the filling in algorithm would have selected the set $E_{i,k}$, or whichever of its sons in the tree according to Proposition 2, before the set $E_{i,n}$, or whichever of its sons in the tree according to Proposition 2. This is because with the first sets we could have satisfied all the requests later met by the last ones.

Any other situation that considers more than two sets, if the initial hypothesis fails, includes the case of two sets, and as a consequence none of them will be possible. So, $\forall i, 0 < i \leq 6$, if there is more than one free set $E_{i,j}$, at most one of them, it is a singular set, and so the table is normalized.

□

Theorem 3 *If after applying several times the filling in algorithm there are still n free entries in the table, it is possible to meet the most restrictive possible request, which is the type $d = 2^i$, with $\frac{64}{d} \leq n < \frac{64}{\frac{d}{2}}$. So, this means that one free set $E_{i,m}$ exists, where $m \in I_i$.*

Proof: We know there are n free entries in the table. If $\frac{64}{d}$ entries of them belong to the same set $E_{i,m}$, the proof is trivial.

Now, let us suppose there is no free set $E_{i,j}$, so the n free entries are distributed among a series of free sets $E_{k,l}$, with $i < k \leq 6$ and $l \in I_k$. According to the Theorem 2, after applying several times the filling in algorithm, the table is normalized. This means that in the level $i+1$ at most there is free singular set $E_{i+1,l}$. But this is true $\forall k$, where $i < k \leq 6$. Each one of these sets $E_{k,l}$ would have $\frac{64}{2^k} = 2^{6-k}$ entries, and so, the total number of free entries of these free sets would be

$$\sum_{k=i+1}^{6} 2^{6-k} = \sum_{k=0}^{6-i-1} 2^k = 2^0 + 2^1 + 2^2 + ... + 2^{6-i-1}$$

This is the addition of a geometric progression with difference 2, and its result is $2^{6-i} - 1$. As a consequence

$$\sum_{k=i+1}^{6} 2^{6-k} = \sum_{k=0}^{6-i-1} 2^k = 2^{6-i} - 1 < 2^{6-i} = \frac{64}{2^i} = \frac{64}{d}$$

which is a contradiction from our initial assumption that there are $n \geq \frac{64}{d}$ free entries. So, one free set $E_{i,k}$ must exist.

Therefore, if after applying the filling in algorithm several times, there are still n free entries, then $\frac{64}{2^i}$ entries ($\frac{64}{2^i} \leq n < \frac{64}{\frac{d}{2}}$) belong to the same set $E_{i,m}$, and so it is possible to meet the most restrictive possible request, which is the type $d = 2^i$.

□

Theorem 4 *Let there be a table with n free entries, the filling in algorithm is able to locate whichever set of requests that does not require more than n entries.*

Proof: Let $d_1, d_2, ..., d_m$ be the requests to locate in the table. The table has n free entries. These requests are made in the order indicated by the sequence and they satisfy

$$\sum_{i=1}^{m} \frac{64}{d_i} \leq n$$

Let n_i be the number of free entries in the table when the request d_i is made. Of course, $n_1 = n$. Let us analyze what happens when each one of the requests d_i, $1 \leq i \leq m$, is made. When a request d_i is made, it could be the most restrictive one or not. So, we have two possible cases:

- The request d_i is the most restrictive. In this case, from Theorem 3 it can be met. After meeting it, there are still $n_{i+1} = n_i - \frac{64}{d_i}$ free entries.
- The request d_i is not the most restrictive. In this case, from Theorem 3, with the n_i free entries the most restrictive possible request (d_k, so that $\frac{64}{d_k} \leq n_i < \frac{64}{\frac{d_k}{2}}$) can be met. As d_i is not the most restrictive possible request, $d_i > d_k$, and from Proposition 2, with the free entries able to locate a request of type d_k, $\frac{d_i}{d_k}$ requests of type d_i could be met, where $\frac{d_i}{d_k} \geq 2$. So, the free set required to meet the request d_i exists.

To sum up, if there are enough free entries in the table to locate a set of requests, the filling in algorithm is able to meet them.

□

Theorem 5 *After applying the filling in algorithm the table remains ordered.*

Proof: According to the Theorem 2, the filling in algorithm always leaves the table normalized. Therefore, $E_{i.j}$ and $E_{k.l}$ are the only singular sets of the levels i and k, respectively, $\forall i$ and $\forall k$ so that $0 < i < k \leq 6$, with $j \in I_i$ and $l \in I_k$.

As $E_{k,l}$ is the singular set of the level k, and due to the way the filling in algorithm functions, then $E_{k,m}$ is not free, $\forall m$ such that $m < l$. According to Proposition 2, these sets $E_{k,m}$ are included in their ancestors on the level i, so, $E_{k,m} \subset E_{i,n}$, which means the sets $E_{i,n}$ are not free.

In addition, as after applying the filling in algorithm the set $E_{k,l}$ is singular, then its ancestor in the level i is not free. So, all the ancestor sets in the level i of the set of the level k placed at the left of the set $E_{k,l}$, and also $E_{k,l}$'s ancestor in the level i, are not free. Therefore, the free sets in the level i must be placed at the right of the ancestor of l in the level i, what means that $Ancestor(l,i) < j$. As $E_{k,l} \subset E_{i,Ancestor(l,i)}$ and due to the Proposition 3, $E_{k,l} \not\subset E_{i,j}$, then $l < j$. This, together with $i < k$ implies that $E_{k,l} \blacktriangleleft E_{i,j}$. As this is true $\forall i$ and $\forall k$ such that $0 < i < k \leq 6$, then, according to Definition 6, the table remains ordered.

□

9 Insertions and eliminations in the table

The algorithm presented in the previous section has been set out based on certain initial hypotheses. The first of these is that requests can only be inserted in the table. Obviously, this is not a real situation, and so the releasing situations must also be considered. In this way, in the general behavior of the table we could have insertions of new requests and releases of previously established requests.

Now we eliminate the first initial hypothesis, so that now the elimination of requests is possible. As a consequence, the entries used for the eliminated requests will be released.

Considering the fill in algorithm of Section 8 together with this, we can achieve situations where there are enough available entries to meet a request, but these entries are not situated with a correct separation between them. As a consequence, although there are enough free entries, the request can not be met. This situation is shown in the next example.

Example 1 *We have the table filled* [5]*, and two requests of type* $d = 8$ *are eliminated. These requests were using the entries of the sets* $E_{3,16}$ *and* $E_{3,32}$*. That means that now the table has 16 free entries, and so a request of type* $d' = 4$ *could be met. However, there is no free set* $E_{2,j}$ *(see Figure 9).*

Note that in the previous example the table is no longer normalized when the release is made. In order to solve the situation shown in Example 1, the 16 entries must be situated having a distance of 4 between two consecutive entries. Therefore, it should have a free set $E_{2,j}$. This means that the table must be normalized again. This can be achieved as is indicated in Example 2. In this example, we follow a process based on leaving free entries that are occupied, which are, however, necessary in order to obtain a set of free entries able to meet the most restrictive possible request.

[5] We have selected this starting point in order to clarify the development of this example. However, the situation shown in this example and in the following ones, can be achieved starting from different status of the table.

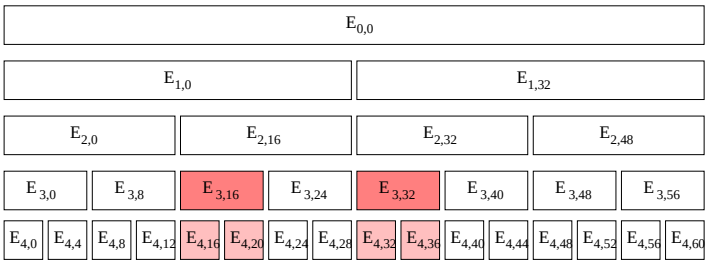


Figure 9: Situation of the arbitration table after the release of the sets $\mathbf{E_{3,16}}$ and $\mathbf{E_{3,32}}$, having started with a filled table.

Example 2 *A solution for the situation shown in Example 1, would be, for example, to release the set $E_{3,40}$, and in this way, together with the set $E_{3,32}$, which is also free, to obtain $E_{2,1}$ free (by Proposition 1). So, the request of type $d' = 4$ could be met.*

In order to make the set $E_{3,40}$ free, the entries of the free set $E_{3,16}$ can be used to meet the requests that are using the entries of the set $E_{3,40}$. This is possible according to Proposition 5. The final situation after the interchange would be the one shown in Figure 10, now leaving the table normalized.

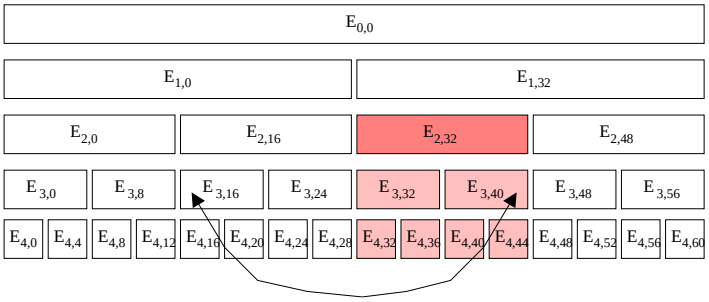


Figure 10: Situation of the arbitration table of Example 1 after the interchange between the sets $\mathbf{E_{3,16}}$ and $\mathbf{E_{3,40}}$.

The aim of this process, which we call *disfragmentation*, is to obtain the greatest free set possible with the available free entries.

Although the need for applying this disfragmentation process emerges as a consequence of considering the release of requests, it is possible that it should be applied after locating requests. This situation is shown in the following example.

Example 3 *This example starts from the situation obtained in Example 2, so the entries of the set $E_{2,32}$ are free and the others are occupied. In this situation the entries of the set $E_{3,56}$ are released. This situation can be shown in Figure 11.*

After that, there is a new request of type $d = 8$, which must be located because there are enough free entries having the correct separation between them. When the filling in algorithm is applied the set $E_{3,32}$ is selected (the first one of level 3). This creates a situation similar to that previously shown in Example 1, which can now be seen in Figure 12.

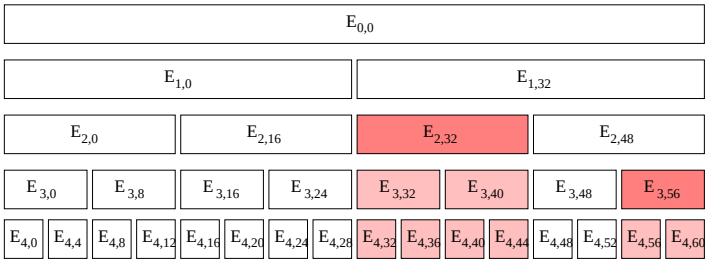


Figure 11: Situation of the table after the release of the set $E_{3,56}$.

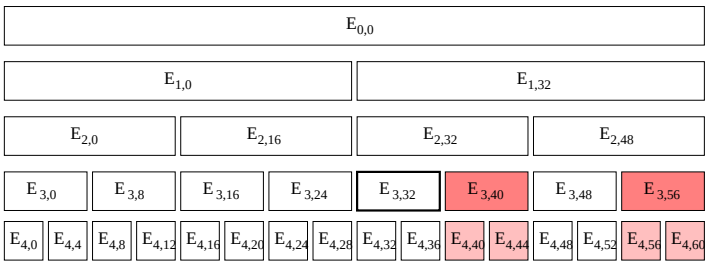


Figure 12: Final situation of the table of the Example 3.

As a consequence, and due to the fact that enough entries exist to meet a request of type $d' = 4$, but do not belong to the same set, a new disfragmentation process is needed. In this case, it could consist, for example, in ensuring that the set $E_{2,48}$ is left free, the requests that now are being used to meet the entries of the set $E_{3,48}$ meeting the entries of the set $E_{3,40}$.

In Example 3, when the set $E_{3,56}$ is released, the table is no longer ordered. When the filling in algorithm is later applied over a non-ordered table, the table does not remain normalized. The solution applied in Example 3 has consisted in applying disfragmentation after a new insertion.

Another alternative is to order the singular sets that have been generated after a release. In this way, the problem would be solved if in the situation shown in Figure 11, the singular set in the level 3 (now $E_{3,56}$) were *to the left* of the singular set of the level 2 (now $E_{2,32}$). So, we could meet the request of type $d = 8$ using the singular set of level 3, without applying disfragmentation and with a free set remaining able to meet a request of type $d' = 4$. Thus, the free sets with a smaller size, whose entries will have greater separation, are situated at the left of the larger free sets. Obviously, this means that the table is ordered.

In order to detect this kind of situations we must study the table checking if there is any free set of a certain size *to the left* of other set that has a smaller size. To solve this situation we must apply Proposition 5, but at the level of the larger set. The interchange must be carried out between the largest set and the set on the same level that includes the smallest free set. This situation is explained in the next example, Example 4.

Example 4 *We will use the final situation of Example 2 as a starting point (Figure*

10). Then the set $E_{3,56}$ is released (Figure 11). We must study all the levels of the tree, until the situation to be solved is found or until the last level is reached. When we have found the situation we must apply Proposition 5, interchanging the sets $E_{2,32}$ and $E_{2,48}$. The resulting situation has as free sets $E_{3,32}$ and $E_{2,48}$, as can be seen in Figure 13. If there is now a request of type $d = 8$, it will be met with the entries of the set $E_{3,32}$, with the free set $E_{2,48}$ remaining, in order to meet a later request of type $d = 4$.

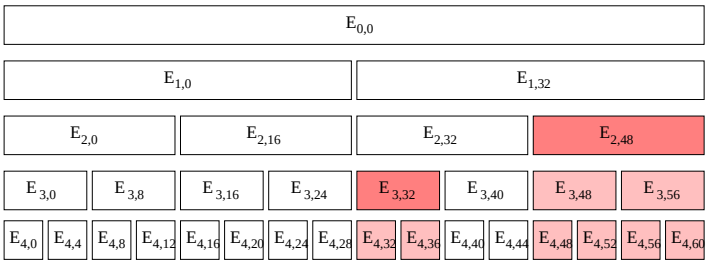


Figure 13: Situation of the arbitration table for the Example 4, after applying a reorder process.

Therefore, with the combination of the filling in algorithm, the disfragmentation algorithm and another algorithm to maintain the ordering, we can cover the general treatment of the table. We have shown two alternatives. In the first, we are able always to maintain the table ordered and normalized, while in the second one, it is only necessary to maintain the table normalized. In this case, it is not necessary for the table to be ordered.

Summing up, we have just outlined two methods to treat those situations that have arisen as a result of having eliminated the restriction imposed by the first of out initial hypotheses in Section 7.2. These methods basically are the following:

1. We must study the resulting status of the table both after meeting a new request (using free entries) and after releasing a request (releasing occupied entries). In both cases we must study if the table has enough free entries to meet a request, but this is not possible because there is no free set of the correct size. If this situation arises the disfragmentation algorithm must be applied to solve it. In other words, if after an insertion or release the table is no longer normalized, a disfragmentation algorithm must be applied to normalize it.

2. We must study the resulting status of the table only when a request is released. In this case we must check if there are free sets of a greater size than other sets situated to the left, and in this case carry out a reordering. We must also check if, there being enough free entries to meet a certain request, it cannot be met because there is no free set of the correct size. In this case, the disfragmentation algorithm must be applied. The order of these checks is not important.

   Summing up, this second method should check after a release if the table is not normalized and/or not ordered. In this case, a disfragmentation algorithm

and/or a reordering algorithm must be executed to leave the table normalized and ordered again.

Thus, there are two new algorithms to be developed: one to perform the disfragmentation and the other to perform the reordering. In the following sections these new algorithms and the generic situations for their use will be explained. We will also study some characteristics of these new algorithms by means of a series of related theorems.

9.1 Disfragmentation algorithm

The basic idea of this algorithm is to join the entries of two free sets of the same size in just one free set. This joining will be done only if the two free sets do not already belong to the same greatest free set. The algorithm is, therefore, only applied on singular sets. The goal is to have a free set of the biggest size in order to be able to meet a request of this size. For this purpose, the table has enough free entries but they belong to two small free sets that are not able to meet that request.

1: **Procedure Disfragmentation** (set $E_{i,k}$)
2: **while** $i > 1$ **do**
3: found = $Find(E_{i,l})$ // $E_{i,l}$ singular
4: **if** found **then**
5: $swap(E_{i,k}, E_{i,m})$ // $E_{i,k}$ is the $E_{i,m}$'s brother
6: **end if**
7: $i = i - 1$
8: $k = Ancestor(m, i)$
9: **end while**
10: **End Disfragmentation**

Figure 14: Disfragmentation algorithm

In Figure 14 the code of the disfragmentation algorithm is shown. Starting from a free set $E_{i,k}$, a check is carried out to see if there is another free set $E_{i,l}$ that is not $E_{i,k}$'s brother (function *Find*, line 3). If it exists, all the entries of both sets are joined in a new free set of double size. This is achieved by applying Proposition 5 (procedure *swap*, line 5). As the new free set cannot be the only one on the level $i-1$, the process must continue. It is possible that this process reaches the level 1 (the loop from the line 2 to 9). The resulting set from the joining is the $E_{i,k}$ of its level (lines 7 and 8).

The basic action of this disfragmentation algorithm is the execution of the procedure *swap*(). This procedure makes a transformation in the table through an interchange of sets. This interchange and its consequences are reflected in Theorem 6.

Theorem 6 *Let there be a table in which* $\exists E_{i,k}, E_{i,l}$ *which are singular, with* $1 < i \leq 6$ *and* $k \neq l$. *It is possible to achieve a situation where* $\exists\ E_{i-1,j}$, *which is free, such that* $E_{i-1,j} = E_{i,m} \cup E_{i,l}$, *with* $k, l, m \in I_i$, $j \in I_{i-1}$.

Proof: We have a table where $\exists\ E_{i,k}, E_{i,l}$ which are singular sets, and so $\nexists E_{i-1,j}$, such that $E_{i-1,j} = E_{i,k} \cup E_{i,l}$. Proposition 5 permits us to interchange the singular set $E_{i,k}$ and the non-free set $E_{i,m}$, where $m = Brother(l)$. So, we interchange the free set $E_{i,k}$ with the $E_{i,l}$'s brother, which is occupied. This process creates a free set $E_{i-1,j}$ such that $E_{i,m} \cup E_{i,l} = E_{i-1,j}$, which allows us to meet a request of type $d = 2^{i-1}$.

□

However, there can be situations where the problem cannot be solved applying only an interchange. In these cases the disfragmentation algorithm must perform several iterations. This situation is shown in the following example.

Example 5 *Taking as a starting point the situation shown in Figure 15(a), after the release of the set $E_{3,0}$ or the $E_{3,16}$, the disfragmentation process is applied. In the first iteration of the algorithm, the sets $E_{3,0}$ and $E_{3,24}$ are interchanged. So, now the set $E_{2,16}$ is free. However, we have 32 free entries although we do not have a free set on level 1 that is able to meet a possible request of type d=2. This is because the table is not normalized, and we have two singular sets on level 2. So, the disfragmentation algorithm must perform another iteration. When the second iteration starts we have $E_{2,16}$ and $E_{2,48}$ as singular sets. The algorithm will now interchange the sets $E_{2,16}$ and $E_{2,32}$, $E_{2,32}$ and $E_{2,48}$ staying as free sets, and so the singular set is now $E_{1,32}$. This is the situation shown in Figure 15(b). Observe that the table is now normalized. If a request of type d=2 were now made, it would be met with the entries of the set $E_{1,32}$.*

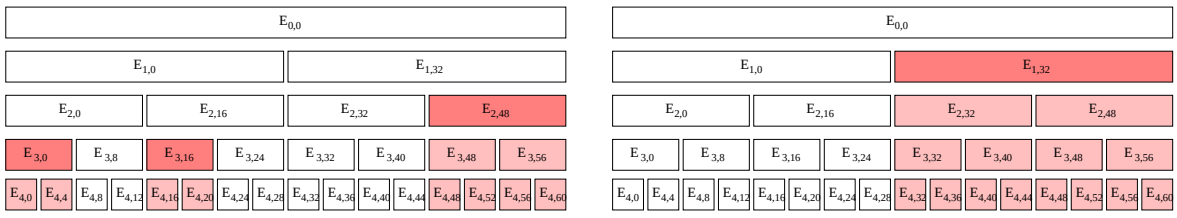


Figure 15: The situation before (a) and after (b) of applying the disfragmentation algorithm in a successive way on several levels.

Therefore, the disfragmentation algorithm consists in applying Theorem 6 to all the levels where there is more than a singular set. So, starting from a table that is not normalized, and using this algorithm, a normalized table is achieved. This is reflected in the following theorem.

Theorem 7 *The disfragmentation algorithm allows us to obtain a normalized table from a table that was not normalized. This was obtained after a release of a request in a normalized table.*

Proof: The disfragmentation algorithm applies the Theorem 6 successively on all the levels where there are two or more singular sets. Starting from the level where the release has happened, it studies all the level up to the level 1 joining a pair of free sets of the same level. In this way, after applying the algorithm there cannot be more than a singular set in each level. So, after applying the disfragmentation algorithm the table is normalized again.

□

Let us see how the disfragmentation process influences the table ordering.

Example 6 *Let us suppose as a starting point the situation shown in Figure 16(a). The only way this situation can be achieved is after the release of the set $E_{3,8}$ or the set $E_{3,16}$. Note that the table is ordered. However, the table is not normalized because there are two singular sets on level 2. When the disfragmentation process is applied the situation shown in Figure 16(b) is reached. Note that the table is still ordered, but now it is also normalized.*

If the disfragmentation algorithm were applied in a successive way to several levels, the order in the table would not be modified either. This is because if we start with an ordered table the biggest sets will always be on the right. So, the order will never be altered although the algorithm joined a singular set formed in the previous iteration, together with the previous one existing on that level.

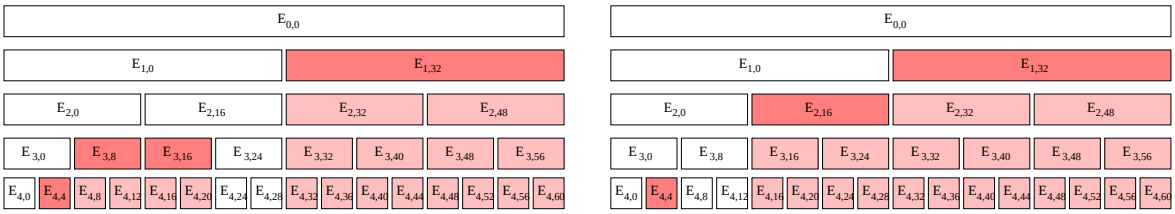


Figure 16: The situation after (a) and before (b) of applying the disfragmentation algorithm. Starting from an ordered table, the algorithm achieves an ordered and normalized table.

Note that we are not saying the disfragmentation algorithm manages to order the table. Nor is it the case that after a release the table is always ordered, as has been seen in Example 13. When we wish to reestablish the order in the table it will be necessary to apply the reordering algorithm, which will be studied in the next section. It is important to emphasize, however, that the disfragmentation algorithm does not disorder an ordered table.

Therefore, after having shown, that the disfragmentation algorithm applied on an ordered and non-normalized table achieves an ordered and normalized table, we are going to prove it by means of the following theorem.

Theorem 8 *After applying the disfragmentation algorithm to an ordered table, in which there is at most one level, i, with* $1 < i \leq 6$, *having two singular sets, the table becomes normalized and remains ordered.*

Proof: Let $E_{i,k}$ and $E_{i,l}$ be the two singular sets of the unique level i that has two singular sets, where $k < l$ y $k, l \in I_i$. Applying the Theorem 7 a free set $E_{t,u}$ is obtained, with $0 < t < i$ and $u \in I_t$.

Let $E_{p,q}$ and $E_{r,s}$ be any two singular sets, such that $0 < r < i < p \leq 6$, $q \in I_p$ and $s \in I_r$. As the table is initially ordered, then $E_{p,q} \blacktriangleleft E_{i,k} \blacktriangleleft E_{i,l} \blacktriangleleft E_{r,s}$.

With this situation as a starting point, we have two possibilities:

- If $r < t$, then we must prove that $E_{p,q} \blacktriangleleft E_{t,u} \blacktriangleleft E_{r,s}$.

  Let us first see the case when $t = i - 1$. In this case $u = Ancestor(l, t)$. As $E_{i,k} \blacktriangleleft E_{i,l}$, then from Proposition 7 $E_{i,k} \blacktriangleleft E_{t,u}$. Besides, as $E_{p,q} \blacktriangleleft E_{i,k}$ and $E_{i,k} \blacktriangleleft E_{t,u}$, due to the transitive property of the order relation, then $E_{p,q} \blacktriangleleft E_{t,u}$.

  If $r < t < i - 1$ the proof is similar, but applying in a successive way the Proposition 7 and the transitive property of the order relation. So, in any case, if $r < t$ then $E_{t,u} \blacktriangleleft E_{r,s}$.

  On the other hand, it must also be true that $E_{t,u} \blacktriangleleft E_{r,s}$, which needs $r \leq t$ and $u < s$. The first one is in the initial hypothesis. Let's study the second one for the particular case where $t = i - 1$. As $E_{i,l} \blacktriangleleft E_{r,s}$ then $l < s$. Besides, as $u = Ancestor(l, t)$, and following on from the consequence of Proposition 2, which relates the indexes of a set and its descendants, then $u \leq l$, and so $u < s$. Therefore, $E_{t,u} \blacktriangleleft E_{r,s}$.

  Again, for the general case when $r < t < i - 1$, the proof is similar doing it in a successive way. Therefore, in any case if $r < t$ then $E_{t,u} \blacktriangleleft E_{r,s}$.

  In this way, $E_{p,q} \blacktriangleleft E_{t,u} \blacktriangleleft E_{r,s}$, and so if $r < t$ the new singular set $E_{t,u}$, which the disfragmentation algorithm has generated, keeps the table ordered.

- If $r = t$, then the disfragmentation algorithm, using $E_{t,u}$ and $E_{r,s}$, will generate a singular set $E_{r-1,h}$, with $h \in I_{r-1}$, $h = Ancestor(s, r - 1)$. In this case, we only need to prove that $E_{p,q} \blacktriangleleft E_{r-1,h}$.

  This proof is similar to the proof made in the previous item where the Proposition 7 and the transitive property of the order relation should be applied in a successive way.

Therefore, in any case, after applying the disfragmentation algorithm on an ordered table, in which there is at most one level i, with $1 < i \leq 6$, with two singular sets, the table becomes normalized and remains ordered.

□

In the following section, the reordering algorithm will be proposed. We start with an informal description using easy examples. Then the algorithm's characteristics are shown using several theorems.

9.2 Reordering algorithm

The reordering algorithm proposed here consists, basically, in an order algorithm, but applying it at a level of sets. This algorithm has been designed to order an unordered table, according to Definition 6. The reordering algorithm establishes an order from lower to larger size, and from left to right. When the order fails, as a consequence of the apparition of new badly situated singular sets, the algorithm will act to set the order again.

When a set is released, the algorithm checks if this set is correctly placed with respect to the other free sets. This new free set must have *on its left* all the smaller free sets, and placed *on its right* all the larger free sets. If its position is not correct the algorithm will make the necessary movements to reestablish the order in the table. These movements will be made by interchange, in a similar way to the disfragmentation algorithm.

As has been previously indicated, the reordering algorithm will be used each time an entry release occurs. This release happens in an ordered table, so that, when the reordering algorithm is applied, there is only one set that is not ordered with respect to the other sets.

Note that the way the reordering algorithm functions is the same as the well known *direct selection* ordering algorithm [15].

```
1:  Procedure Reordering
2:  for k = 6 downto 2 do
3:    for all E_{k,l} singular do
4:      for i = k − 1 downto 2 do
5:        found = Find(E_{i,j})                    // E_{i,j} singular and E_{k,l} ◄ E_{i.j}
6:        if found then
7:          swap(E_{i,j}, E_{i,Ancestor(k,i)})
8:        end if
9:      end for
10:   end for
11: end for
12: End Reordering
```

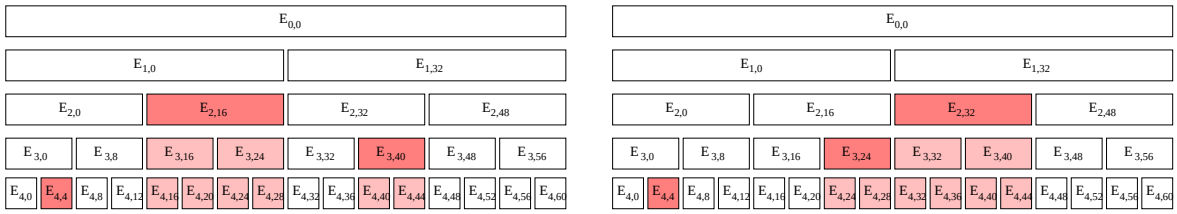
Figure 17: Reordering algorithm.

In Figure 17 the basic code that makes up the algorithm is shown. The algorithm studies all the possible levels (lines 2-11), except the two last, starting from the bottom. At each level we will have at most a singular set, $E_{k,l}$, except at maybe one level where there could be two singular sets, only one of them being badly ordered. So, in each level for each one of the singular sets $E_{k,l}$, the algorithm checks if there is a singular set $E_{i,j}$ of the upper levels situated on its left (lines 4-9). If there are several sets, the one situated furthermost to the left is selected (lines 5 and 6), and they are ordered.

For that purpose, Proposition 5 is applied over set $E_{i,j}$ and $E_{k,l}$'s ancestor at the level i (procedure *swap*, line 7).

The following easy example shows how the reordering algorithm works.

Example 7 *Let us take as a starting point the situation shown in Figure 18(a). This situation has been reached after the release of set $E_{2,16}$ or set $E_{3,40}$ in a previously ordered table. The reordering algorithm will interchange the sets $E_{2,16}$ and $E_{2,32}$. In this way, the final situation will be as shown in Figure 18(b), where the singular sets are now $E_{4,4}$, $E_{3,24}$ and $E_{2,32}$, with the table ordered once again.*



(a) (b)

Figure 18: The situation before (a) and after (b) of applying the reordering algorithm.

Therefore, with the interchange of the correct sets, we can pass from a situation when some singular sets do not maintain the order relation of Definition 5, to another situation when the new singular sets now maintain this order. This is shown in a formal way in the following theorem.

Theorem 9 *Let $E_{k,l}$ and $E_{i,j}$ be the only singular sets in the levels k and i, respectively, with $0 < i < k \leq 6$, $l \in I_k$ and $j \in I_i$, such that they do not maintain an order relation. So, $E_{k,l} \not\blacktriangleleft E_{i,j}$. However, it is possible to change to another situation with $E_{k,n} \blacktriangleleft E_{i,m}$, with $n \in I_k$ and $m \in I_i$, where $E_{k,n}$ and $E_{i,m}$ are now the only singular sets of the levels k and i, respectively.*

Proof: The Proposition 5 allows us to interchange the sets $E_{i,j}$ and $E_{i,m}$, where $m = Ancestor(l, i)$. Therefore, the singular set on level i is now $E_{i,m}$ while $E_{i,j}$ is no longer free. The same happens with the singular set on level k, which is now $E_{k,n}$, with $n = l - m + j$. What we must prove is $E_{k,n} \blacktriangleleft E_{i,m}$. For that purpose it would be enough to prove $E_{k,n} \blacktriangleleft E_{k,l}$, because if this is true, and applying Proposition 7, then $E_{k,n} \blacktriangleleft E_{i,m}$.

Let us suppose $E_{k,n} \not\blacktriangleleft E_{k,l}$, and so, $n \geq l$. But as $n \neq l$, because otherwise $E_{k,n}$ and $E_{k,l}$ will be the same set, then $n > l$. But, then $E_{k,l} \blacktriangleleft E_{k,n}$, and from Proposition 7, then $E_{k,l} \blacktriangleleft E_{i,j}$, which is false according to the initial hypothesis. So, $E_{k,l} \not\blacktriangleleft E_{k,n}$, $n < l$, and $E_{k,n} \blacktriangleleft E_{k,l}$, and from Proposition 7, $E_{k,n} \blacktriangleleft E_{i,m}$.

□

Now that we can turn two unordered singular sets into two ordered singular sets, we can achieve a totally ordered table applying this process in a successive way. This result is provided by the following theorem.

Theorem 10 *The reordering algorithm allows us to pass from a table that is not ordered to one that is ordered.*

Proof: The reordering algorithm applies successively Theorem 9 to any pair of sets $E_{i,j}$ and $E_{k,l}$ which are singular, with $0 < i < k \leq 6$, $j \in I_i$ y $l \in I_k$, such that $E_{k,l} \not\blacktriangleleft E_{i,j}$. After applying the theorem then $E_{k,n} \blacktriangleleft E_{i,m}$, with $n = l - m + j$ and $m = Ancestor(l, i)$. Therefore, after applying the reordering algorithm the table becomes ordered.

□

As mentioned earlier the algorithm's operation is similar to the direct selection algorithm. Thus the correct functioning of the algorithm can now be guaranteed, and it obtains an ordered table.

As a final step, we are now going to study what happens when this algorithm is applied to a table that is normalized. We want to prove that the table remains normalized. We will use the fact that the reordering algorithm does not generate more singular sets than those previously existing, and only changes its order. We will see this result in the following theorem.

Theorem 11 *After applying the reordering algorithm to a normalized table, the table remains normalized.*

Proof: If the table is normalized it must have at most a singular set in each level. But the reordering algorithm always interchanges a singular set $E_{i,j}$ with another set $E_{i,m}$ which is not free. So, the algorithm will never generate more free sets than those existing at each level. Therefore, we could have, at most, a singular set per level, and thus the table remains normalized.

□

Once we have presented the algorithms to perform the filling in, the disfragmentation and the reordering, we need to bring this all together and to test that the global treatment of the table is correct.

10 Global management of the table

In the previous sections we have studied the disfragmentation and reordering algorithms. However, the separate parts need to be brought together. We will prove that the final situation achieved in the arbitration table after insertions and releases of requests, with their corresponding disfragmentations and reorderings, is equivalent to the situation achieved when we only have the insertions of the requests that are left in

the table. The term equivalent refers to the capacity to meet requests, and is exactly defined in the following.

Definition 7 *We say that two tables T and T' are equivalent if they can meet the same requests.*

Independently of the concrete free sets that both tables have, they must be able to meet the same requests. As we will see, two table are equivalent if they have the same number of singular sets and are distributed on the same levels. As an example, the tables shown in Figures 15(a) and 15(b) are not equivalent, although they have the same number of free entries. This is because with the second table we can meet requests of type $d \geq 2$, while with the first we can only meet requests of type $d' \geq 4$. In the same way, the tables shown in Figures 16(a) and 16(b) are not equivalent. The following is an example where two tables are equivalents.

Example 8 *In Figure 19 we can see two equivalent tables. In the table shown in Figure 19(a) the singular sets are $E_{4,4}$, $E_{3,24}$ and $E_{2,32}$, while in Figure 19(b) the singular sets are $E_{4,8}$, $E_{3,16}$ and $E_{2,48}$, the other sets (which are not descendants of them) being occupied. However, both tables are able to meet exactly the same sequences of requests.*

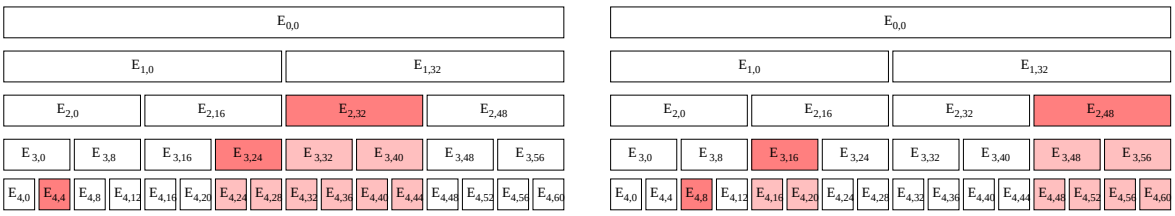


Figure 19: Two equivalent tables.

The following theorem proves that two normalized tables having the same number of singular sets have the same number of free entries.

Theorem 12 *If there are two normalized tables T and T', such that $\forall E_{i,j} \in T$, where $E_{i,j}$ is singular, $\exists E_{i,k} \in T'$, where $E_{i,k}$ is also singular, and $\forall E_{i,k} \in T'$, where $E_{i,k}$ is singular, $\exists E_{i,j} \in T$, where $E_{i,j}$ is also singular, with $0 < i \leq 6$, $j, k \in I_i$, then both tables have the same number of free entries.*

Proof: The proof is based on reductio ad absurdum. Let us suppose that both tables T and T' do not have the same number of entries. So, there must be a level l, with $0 \leq l \leq 6$, from which (up to the lower levels) both tables do not have the same number of free entries. As the number of singular sets is the same at all the levels, then one

of the tables should have a number n of free sets that are not singular, and these sets are not in the other table. In order for these free sets not to be singular the situation should be such that their brother sets are also free. But this would form one or more singular sets in some level t, with $0 < t < i$, which are in one table, but not in the other, which contradicts the initial hypothesis.

Therefore, if two tables T and T' are normalized, and $\forall E_{i,j} \in T$, where $E_{i,j}$ is singular, $\exists E_{i,k} \in T'$, where $E_{i,k}$ is also singular, and $\forall E_{i,k} \in T'$, where $E_{i,k}$ is singular, $\exists E_{i,j} \in T$, where $E_{i,j}$ is also singular, with $0 < i \leq 6$, $j, k \in I_i$, then both tables have the same number of free entries.

□

Using the previous theorem, we are going to prove that two normalized tables are equivalent, if they have the same number of singular sets and these are located at the same levels.

Theorem 13 *If there are two normalized tables T and T', such that $\forall E_{i,j} \in T$, where $E_{i,j}$ is singular, $\exists E_{i,k} \in T'$, where $E_{i,k}$ is also singular, and $\forall E_{i,k} \in T'$, where $E_{i,k}$ is singular, $\exists E_{i,j} \in T$, where $E_{i,j}$ is also singular, with $0 < i \leq 6$, $j, k \in I_i$, then both tables are equivalent.*

Proof: If T and T' are normalized they have at most a singular set per level. Besides, as $\forall E_{i,j} \in T$, where $E_{i,j}$ is singular, $\exists E_{i,k} \in T'$, where $E_{i,k}$ is also singular, with $0 < i \leq 6$, and vice versa, and by Theorem 12, both tables have the same number of free entries. Therefore, according to Theorem 4 the tables T and T' are able to meet the same number of requests, and thus are equivalent.

□

We will prove that the resultant situation of insert and release requests is equivalent to the final situation if there are only the requests that are left after the releases. So, all the work developed for the case when there are no releases is also applicable to the case when there are request releases (considering the disfragmentation algorithm and the reordering algorithm). This is because we would have an equivalent situation regarding new requests, and in this case, we can apply Theorem 4. In this way, we will have linked both parts of the theory. Let us see the theorem to prove this equivalence.

Theorem 14 *Let a series of requests d_1, d_2, $\cdots$, d_n be followed by a series of releases d'_1, d'_2, $\cdots$, d'_m, after each one the reordering and/or disfragmentation algorithms is applied, when the table needs to be ordered and/or normalized, where $\exists d_i / d'_j = d_i$, $\forall j \in [1, m]$. This sequence of insertions d_i followed by these releases d'_j produces an equivalent result to the insertion of only the following requests $\{d_1, d_2, \cdots, d_n\} - \{d'_1, d'_2, \cdots, d'_m\}$, which are the requests that remain after the releases, when all the insertions have been done.*

Proof: Let T be the resulting table after realizing the insertion of the requests d_1, d_2, $\cdots$, d_n, followed in any order by some releases d'_1, d'_2, $\cdots$, d'_m. On the other hand, let T' be the resulting table if the insertions $\{d_1, d_2, \cdots, d_n\} - \{d'_1, d'_2, \cdots, d'_m\}$ are directly performed. What we want to prove is that T and T' are equivalent tables. Thus, we are going to show that both tables have the same number of singular sets and are placed at the same levels. Thus, according to Theorem 13, both tables are able to meet exactly the same requests.

As was explained in Section 9 on page 47, we have two options to dynamically manage the table:

1. After each insertion d_i and each release d'_j the disfragmentation algorithm is applied. So, according to Theorem 7 table T becomes normalized. On the other hand, according to Theorem 2 the table T' is also normalized.

2. In this case, after each release d'_j the reordering algorithm and/or the disfragmentation algorithm are applied. Thus, according to Theorems 7, 8 and 11, the table T is normalized. For the same reason, from Theorem 2 table T' is normalized.

In both cases tables T and T' are normalized and so at most have only a singular set per level. We can see that both tables also have the same number of singular sets and are placed in the same levels.

If T and T' do not have the same number of singular sets, or having the same number are not situated at the same levels, then is because for some level i, with $0 < i \leq 6$, $\exists E_{i,j} \in T$ which is singular, $j \in I_i$, but $\nexists E_{i,k} \in T'$, with $E_{i,k}$ singular and $k \in I_i$. The sets $E_{i,l} \in T'$ which are not free, with $l \in I_i$, could be due to requests of type $d = 2^i$, or by Proposition 2, to one or some requests $d' = 2^{i+p}$, with $1 \leq p \leq 6 - i$ such that $E_{i,l} = \bigcup_{m=0}^{2^p-1} E_{i+p,j+m\times 2^{6-(i+p)}}$. This would mean that the request of type $d = 2^i$ (or in a similar way, the request of type $d' = 2^{i+p}$) in which both tables differ, would be in the sequence of requests d_1, d_2, $\cdots$, d_n and not in the sequence $\{d_1, d_2, \cdots, d_n\} - \{d'_1, d'_2, \cdots, d'_m\}$. But we know that this is false, so this situation is not possible.

Thus, we have proved that both tables T and T' have the same number of singular sets and are located at the same levels. Thus, according to Theorem 13, both tables are able to meet the same requests. Therefore, we have proved that both tables T and T' are equivalent.

□

Both parts of the theory are now brought together. Now, in whatever situation it is possible to apply Theorem 4 in order to meet new requests. The conclusion is important because whatever the request our proposal is able to meet it in the table provided simply that there are enough free entries.

11 Conclusions

In this work we have dealt with the formalizing of a scheduling model to provide QoS for the applications in a subnet InfiniBand. We have started studying the different kind of traffic and making a traffic classification based on their requirements. This classification is based on the previous one proposed in [16], under which we proposed some modifications in [2].

In this work 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 two consecutive 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, and have proposed to categorize them in the powers that are divisors of 64. So, we will have as possible maximum distances the values 2^0, 2^1, 2^2, 2^3, 2^4, 2^5 y 2^6. If the request does not match one of these categories, it is rounded up to the lowest closest category. Obviously, this makes us use up entries in the arbitration table. However, it simplifies the management of the requests and it achieves a good filling in of the arbitration table.

We have also made a formal study about the management of the arbitration table to provide QoS. We have defined a model that is based on sets of entries in the arbitration table. Each set is the sequence of entries of the arbitration table that have a certain distance between two consecutive entries. After defining the model we have enunciated and proved a series of propositions that have allowed us to enunciate the theorems. With these theorems we have proved that, if we have enough free entries in the arbitration table, with our algorithm, we can always situate the request in the table. This is because the algorithm always uses the entries in a concrete way, leaving the other free entries in the best place to meet other requests later.

Finally, we have also studied the different possible situations in which requests are met and released from the arbitration table. We have analyzed the different situations and have found some problematic situations affecting the behavior of our algorithm. In order to solve these problems, we have proposed two new algorithms. One of them is a disfragmentation algorithm that must be executed every time a request is released on the table. The idea of this algorithm is to join loose sets (sequences) in order to form other sets able to meet requests that are more restrictive than the previous ones. For that purpose, it is possible that an interchange between a free set and another occupied set can be made. This is in order to put together the two free sets, and in this way to form a large free set able to meet a more restrictive request.

The other algorithm proposed is used to maintain the order among the sets. The idea is that we could need to have the free sets in a certain order based on their size. As in the previous case, to achieve our propose, the algorithm will interchange different sets.

We have two possibilities: if we only want to make these additional operations on the table after a release, we need both the disfragmentation and the reordering

algorithms. If we apply the disfragmentation algorithm both after an insertion and after a release, it is not necessary to maintain an order among the sets, thereby making the reordering algorithm superfluous.

The algorithm to fill in the arbitration table, together with the disfragmentation and reordering algorithms, allow us to make a dynamic use of the arbitration table of InfiniBand when there are requests to be settled in the table and also requests to be released from the table.

In [3] this model has been evaluated. An InfiniBand network simulator has been used. We have modeled several kinds of applications, embracing different QoS requirements. Regarding bandwidth the application requirement varied from 8 KBps to 300 MBps, which covered almost all kind of applications having this requirement. The applications also requested a maximum end-to-end latency. This requirement could vary from the minimum distance considered between two consecutive entries in the IBA arbitration table (distance of 2) to the maximum possible distance allowed (distance of 64). Therefore, the application requirements that we have modeled in the simulator agree with both the current applications with QoS requirements, and with the theoretical model developed in this work.

The experimental results that we have obtained show that all applications achieve the QoS requirements previously requested. By making a previous resources reservation, the network is able to deal with very high workloads. These results are very good, and they confirm that our theoretical model is able to provide the QoS required for the applications.

Bibliography

[1] *InfiniBand^{TM} Trade Association*, 1999. `http://infinibandta.com`.

[2] F. J. Alfaro, J. L. Sánchez, and J. Duato. A Strategy to Manage Time Sensitive Traffic in InfiniBand. In *Proceedings of Workshop on Communication Architecture for Clusters (CAC'02)*, Apr. 2002. Held in conjunction with IPDPS'02, Fort Lauderdale, Florida.

[3] F. J. Alfaro, J. L. Sánchez, and J. Duato. A New Proposal to Fill in the InfiniBand Arbitration Tables. Submitted to the International Conference on Parallel Computing (ICPP'03), Oct. 2003.

[4] F. J. Alfaro, J. L. Sánchez, J. Duato, and C. R. Das. A Strategy to Compute the InfiniBand Arbitration Tables. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'02)*, Apr. 2002.

[5] S. Blake, D. Back, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. Internet Request for Comment RFC 2475, Internet Engineering Task Force, Dec. 1998.

[6] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RVP) – version 1 functional specification. Internet Request for Comment RFC 2205, Internet Engineering Task Force, Sept. 1997.

[7] P. by ISSG Technology Communications. Infiniband architectural technology. Technical Report TC000702TB, Compaq Computer Corporation, July 2000.

[8] M. B. Caminero. *Diseño de un Encaminador Orientado a Tráfico Multimedia en Entornos LAN*. PhD thesis, Departamento de Informática de la Universidad de Castilla-La Mancha, July 2002.

[9] P. Cuenca. *Codificación y transmisión robusta de señales de vídeo MPEG-2 de caudal variable sobre redes de transmisión asíncrona ATM*. PhD thesis, Universidad Politécnica de Valencia, 1998.

[10] I. Dalgic. *Performance of Ethernet and ATM Networks Carrying Video Traffic Based on Accurate Characteristics of Video Sources*. PhD thesis, Stanford University, Aug. 1996.

[11] A. Forum. *ATM Forum traffic management specification. Version 4.0*, May 1995.

[12] N. Giroux and S. Ganti. *Quality of Service in ATM Networks*. Prentice Hall, 1999.

[13] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1. Release 1.0*, Oct. 2000.

[14] G. Karlsson. Asynchronous transfer of video. *IEEE communication Magazine*, 24(8):118–126, August 1996.

[15] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

[16] J. Pelissier. Providing Quality of Service over Infiniband Architecture Fabrics. In *Proceedings of the 8th Symposium on Hot Interconnects*, Aug. 2000.

[17] G. Pfister. *High Performance Mass Storage and Parallel I/O*, chapter 42: An Introduction to the InfiniBand Architecture, pages 617–632. IEEE Press and Wiley Press, 2001.

[18] M. Schwartz and D. Beaumount. Quality of service requirements for audio-visual multimedia services. *ATM Forum*, ATM94-0640, July 1994.

[19] J. Wroclawski. RFC 2210: The Use of RSVP with IETF Integrated Services, Sept. 1997. Status: proposed standard.