A New Proposal to Fill in the InfiniBand Arbitration Tables *

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

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

Abstract

The InfiniBand Architecture (IBA) is a new industry-standard architecture for server I/O and inter-processor 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.

In this paper, we propose a simple and effective strategy for configuring the IBA networks to provide the required levels of QoS. This is a global frame that allows one to do a different treatment to each kind of traffic based on its QoS requirements. It is based on the correct configuration of the mechanisms IBA provides to support QoS. We also propose a simple algorithm to maximize the number of requests to be allocated in the arbitration table that the output ports have. This proposal is evaluated and the results

*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

show that every traffic class meets its QoS requirements.

Keywords: *InfiniBand, Arbitration, QoS, NOWs, Application requirements.*

Technical areas most relevant to this paper: *Network-based computing, Networking and protocols.*

1. Introduction

Current network interconnection standards and technologies are built around outmoded shared-bus I/O architectures that are inadequate to handle the increasing data demands of today's Web and network-powered businesses. The inherent bandwidth limitations of the shared-bus architecture require network I/O requests to compete with other attached devices to access to the PCI bus and CPU.

In response to these limitations, the InfiniBand Trade Association (IBTA) was formed in 1999. In the fall of 2000, with the support of well over 200 member companies, IBTA unveiled a new, open and inter-operable I/O specification called InfiniBand. This new standard has been developed for communication between processing nodes and I/O devices as well as for interprocessor communication.

InfiniBand replaces the PCI bus and provides the next generation of I/O connectivity to PC and server platforms. However, it is a revolutionary, rather than an evolutionary change to the traditional PC bus based architecture, as InfiniBand defines a switch based serial I/O fabric.

InfiniBand completes the disaggregation of the server moving I/O connections out of the box. InfiniBand breaks the conventional one-to-one relationship between server and I/O elements and allows for a many-to-many server-to-I/O relationship. This changes the fault granularity of a highly available system based on a disaggregated clustered architecture.

InfiniBand has been developed from the ground up to provide a cost effective, high performance

solution with reliability, availability, and serviceability support for the Internet Data Center. InfiniBand provides some mechanisms that provide the required QoS for each kind of application.

In previous works [3, 2] we have developed different proposals to configure IBA in order to provide QoS. We successfully provided both bandwidth and latency guarantees. However, these proposals included some limitations if the sources did not have a behavior according to what they previously requested. In this paper, we solve these limitations and we propose a global frame to provide the required QoS for each possible kind of application traffic. We also propose a traffic classification based only on the latency requirements.

The rest of the paper has the following structure: Section 2 presents a summary of the general aspects of the IBA specification, mainly, the most important mechanisms provided by IBA to support QoS. In Section 3, we present our proposal to manage traffic with QoS requirements, and its performance is evaluated in Section 4. Finally, some conclusions are given.

2. InfiniBand

InfiniBand incorporates the proven message-passing, memory-mapping, and point-to-point link technologies of mainframe-based networks. InfiniBand hardware provides highly reliable, fault-tolerant communication, improving the bandwidth, latency, and reliability of the system. The InfiniBand network of connections between servers, remote networking, and storage devices provides a scalable system without encountering the delays and physical limitations of bus-based architectures (see Figure 1).

The InfiniBand architecture offers a new approach to I/O. It simplifies and speeds server-to-server connections and links to other server-related systems, such as remote storage and networking devices, through a message-based fabric network.

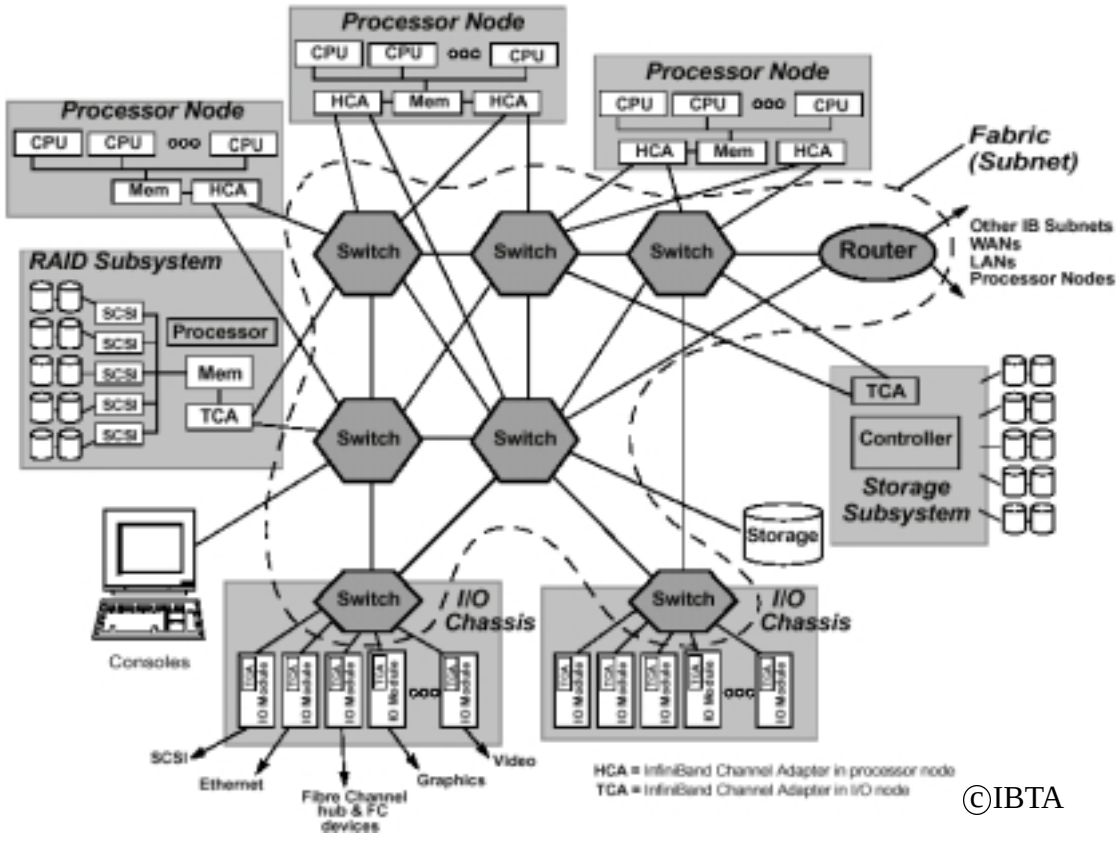

Figure 1. IBA System Area Network.

IBA has the flexibility to be used for multiple technologies that include server-to-server communication (IPC for clustering), switching, and storage as well as in-band management functions across the InfiniBand wire. One key to this flexibility is that only InfiniBand architecture has the necessary mechanisms to support the integration of the System Area Network by enabling “virtual fabrics” (through virtual lanes) to carry each type of traffic.

IBA links are full-duplex point-to-point communication channels. Signaling rate on the links is 2.5 GHz in the 1.0 release, the later releases possibly being faster. 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. Other link rates are 10 Gbps (referred to as 4x) and 30 Gbps (referred to as 12x) that

correspond to 4-bit wide and 12-bit wide links, respectively.

IBA segments messages 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.

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 the usual QoS requirements (guarantee of bandwidth, maximum latency deadline, interarrival delays, etc.) applications must use reliable connections in order to be able to do resource allocation.

The interested reader is referred to the InfiniBand Specifications [6] for more details on InfiniBand. Other interesting papers that are good summaries of the official specifications are [8, 5].

2.1 IBA support for QoS

IBA provides three mechanisms that permit QoS to be supported: Service levels, virtual lanes, and virtual lane arbitration for transmission over links. 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 the administrator how to distribute the different existing traffic types among the SLs. By allowing the traffic to be segregated by category, we will be able to distinguish between packets from different SLs and to give them a different treatment based on their needs.

IBA ports support virtual lanes (VLs), providing a mechanism for creating multiple virtual links within a single physical link. As we can see in Figure 2, a VL represents a set of transmit and receive buffers in a port. IBA ports can 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. Since systems can be constructed with switches supporting different numbers of VLs, the number of VLs used by a port is configured by the subnet manager. Also, 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 a SLtoVLMappingTable. Each VL must be an independent resource for flow control purposes.

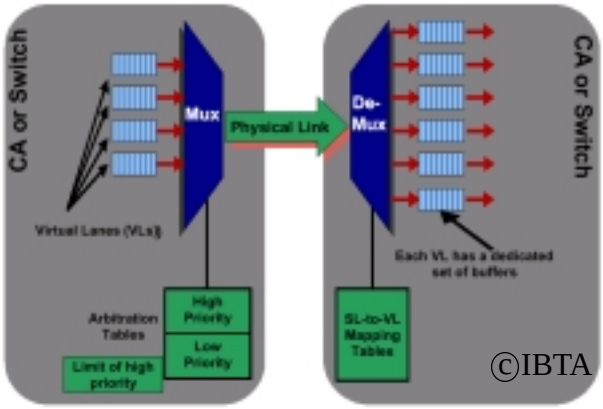

Figure 2. Operation of virtual lanes in a physical link.

When more than two VLs are implemented, the priorities of the data lanes are defined by the VLArbitrationTable. This arbitration is only for data VLs, because VL_{15}, which transports control traffic, always has priority over any other VL. The structure of the VLArbitrationTable is shown in Figure 3. The VLArbitrationTable has two tables, one for scheduling packets from high priority VLs and another for low priority VLs. However, IBA does not specify between 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 transmitted from that VL. This weight must be in the range of 0 to 255, and is always rounded up as a whole packet.

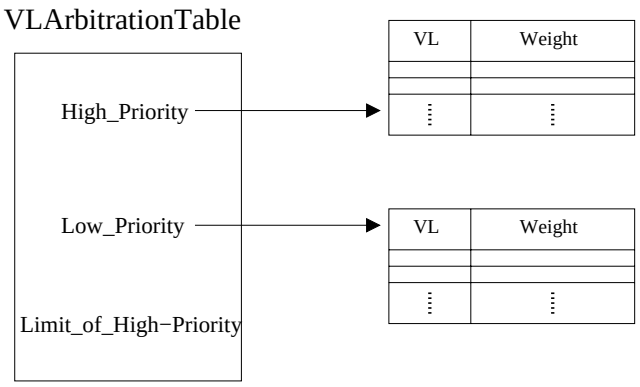

Figure 3. Structure of the virtual lane arbitration table.

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 could be transmitted. If no high priority packets are ready for transmission at a given time, low priority packets can also be transmitted.

3. Filling in the virtual lane arbitration table

In previous works we proposed a strategy to compute the IBA arbitration tables both for traffic only with bandwidth requirements and also traffic with latency requirements. But our proposals had problems that we have solved. In this section we are going to explain the traffic classification proposed, how to establish the SLs, and a new algorithm for optimizing the filling in of the arbitration table.

3.1 Traffic classification

Pelissier proposed a traffic classification that contains four distinguishing categories [7]: DBTS (Dedicated Bandwidth Time Sensitive), DB (Dedicated Bandwidth), BE (Best Effort) and CH (Challenged).

In [2], we proposed to extend that classification considering two priority levels for Best Effort traffic: PBE (Preferential Best Effort: for example web or data base accesses) and BE (the usual Best Effort like mail, ftp, etc.). This allows for preferential treatment of some kinds of Best Effort traffic over the rest, but without any type of guarantees.

He also proposed to use the high-priority table for DBTS traffic and the low-priority table for the rest of the traffic, including DB traffic. We used this model to test our proposals in [3, 2, 4]. However, the problem that exists in this model is that no bandwidth can be guaranteed to traffic that uses the low-priority table if sources of traffic using high-priority table use more bandwidth that they previously requested. So, with this model no guarantee can be done to DB traffic.

To solve this problem we propose to put all traffic with guarantee requirements (DBTS and DB) in the high-priority table. Thus, we would be mixing traffic with bandwidth and latency requirements with traffic having just bandwidth requirements. But note that DB traffic could be considered as DBTS traffic with a big enough time deadline.

In [2] we studied how to provide latency guarantees in IBA. Several switch models were studied. In all cases we computed the maximum distance allowed between two consecutive entries in the high-priority table to guarantee the maximum latency per switch. Note that DBTS traffic could need several entries in the high-priority table while DB traffic will just need one entry in this table. This is just to meet the maximum deadline requested. Besides, the connection also requests a mean bandwidth that turns into a weight to put in the entries of the table. So, for a certain connection that requests a maximum distance d and a mean bandwidth that turns in a weight w, the number of entries needed is $max\{\frac{64}{d}, \frac{w}{255}\}$.

Obviously, the maximum distance between two consecutive entries in the high-priority table allowed by a connection could be between 1 and 64. In order to maximize the number of requests attended, the

best way of setting the entries consists of distributing them in an arithmetic progression with a difference equal to the distance required by the request. But the arithmetic progressions that are symmetric in a table of 64 entries (2^6) are only those that have difference of the progression the divisors of 64. Since 64 is a power of 2 the divisors are just the powers of 2 lower than or equal to 2^6, which are 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, and 2^6. We think the distance equal to 1 is too much strict to be considered in a practical way. So, in order to optimize the way of filling in the table, we will only consider the following distances: 2, 4, 8, 16, 32, and 64.

This approach has the problem that not all requests are power of 2. So, the requests must be considered in terms of the closest lower power of 2, perhaps using more entries than needed. But we must note that for optimal placing it is not enough to consider the number of entries used with a certain sequence, but also, all of them that we will not be able to use later. You can find a deeper analysis of this topic in [1]. With these distances, and with the algorithm that we are going to propose, we will succeed in allocating a new request if there are enough available entries. This is because the available entries are always situated in order to meet the most restrictive request (the one with the minimum distance between its entries).

3.2 Establishing the SLs

When a connection is going to be established it will request a bandwidth and a maximum latency. In [3] we studied how to guarantee bandwidth. We showed that a request of a certain bandwidth was treated in each switch as a request of the corresponding weight in the arbitration table.

To provide a maximum latency guarantee, the entries used in each arbitration table must be arranged with a maximum distance between them. This maximum distance could be different in each switch

crossed. However, in order to give a connection the same treatment in each switch, we will consider the same maximum distance in the arbitration table of each switch crossed. So, in order to fill in the arbitration tables of the switches on the path, to request a maximum latency is equivalent to requesting a sequence with a maximum distance between two consecutive entries in their high-priority tables.

In [3] we proposed that several connections, with the same VL, shared the entries in the arbitration tables. This was in order to accept connections based on the available bandwidth, and not being limited by the 64 available entries in the high-priority table. As we know, each entry in the table can have a maximum weight of 255. So, with this approach we could put together several connections until they fill in the maximum weight of their entries.

In our previous works, we classified the traffic in SLs based on the mean bandwidth requested by the connection. This approach has the problem that different connections could have similar mean bandwidth but different maximum latency requirements. Having different latency requirements they would need different maximum distances between consecutive entries in the high-priority table.

According to the InfiniBand Specifications [6], a correspondence between SLs and VLs is done with the SLtoVLMappingTable. So, with our previous proposal, connection with very different latency requirements could share the same VL. If connections with different distances must share these entries we would have to select the distance most restrictive, and to arrange the sequence of entries according to this distance. This solution has the problem of what to do when the most restrictive connection finishes and we are using more entries that we need. In order to make best use of the table, we would have to store the connection requirements to modify dynamically the entries allocated. We think this introduces too much complexity.

In this paper we propose a new approach to solve this problem. We will classify the traffic in SLs

based on the maximum latency. Specifically, all the connections using the same SL will need the same maximum distance between two consecutive entries in the high-priority table, independently of their mean bandwidth. If we can use a VL for each SL, all the connections sharing a VL will need the same distance between their entries. If several SLs must share a VL, connections with different latency requirements will coexist in the same VL. In this case we could use less SLs or enforce more restrictive requirements for some SLs.

Thus, all connections in the same SL will request the same distance between two consecutive entries in the high-priority table, so having similar latency requirements. For the most used distance values, we will distinguish two or four different SLs based on the mean bandwidth. The most used distance values will usually be those that have the lower latency requirements (which request higher distances: 32 and 64). In this way, if we have enough available VLs, each kind of traffic could use a different VL.

Of course, the problem that arises with this new approach is that we are dedicating VLs in exclusivity to some kinds of traffic that maybe will not actually exist in the network. However, if there are some distances that will never be used, the network administrator could implement the SLtoVLMappingTable in order to ensure that no VLs are assigned in exclusivity to these SLs never used.

With this new approach we solve the problem of guaranteeing their requirements to all kind of connections. Specifically, if some source sends more than it previously requested this will affect only the connections sharing the same VL, but the rest of the traffic in others VLs will achieve what they requested. Besides, both kind of traffic with latency requirements and with mean bandwidth requirements (and, of course, having both of them), will have their requirements guaranteed. We think this is an important improvement that will permit InfiniBand to provide the level of QoS required for each kind of traffic.

3.3 Algorithm

In this section an algorithm to select a sequence of entries in the high-priority table is proposed. This algorithm successfully allocates a new sequence in the table if there are enough available entries. This is because the available entries are always in the best situation to treat the most restrictive request. Due to lack of space, we have not included in this paper the formal demonstration of the properties and theorems derived from this algorithm. However, they are available in [1].

As we have already explained in the previous section, for a certain connection that requests a maximum distance d and a mean bandwidth, that turns in a weight w, the number of entries needed is $max\{\frac{64}{d}, \frac{w}{255}\}$. We also know that this number will be rounded up to the closest permitted value. So, a request for entries in the table could just be of value 32, 16, 8, 4, 2, or 1 entries. This number determines the SL to which this connection corresponds. So, when a request could be met in the table, they will be assigned a sequence of equally spaced entries for that request.

Due to the correspondence established by the SLtoVLMappingTable, when a connection does a request of a certain SL it must use the corresponding VL. So, for a connection requesting $\frac{64}{d}$ entries situated at a distance d among them, the algorithm will look for a previously established sequence, for the corresponding VL, with enough weight available. If there is not an available sequence, a new sequence with these characteristics will be looked for. So, the algorithm to look for a new free sequence must be applied only if there is not available a previously established sequence.

We have developed an algorithm to maximize the number of requests to be allocated in the high-priority table that can be accepted. This algorithm successfully allocates a request in the table if there are enough available entries. It sets the requests in an optimal way being able, later, to put in the most restrictive possible request.

Let a table T, the sequence $t_0, t_1, \ldots, t_{62}, t_{63}$ represents the entries of this table. Each t_i has an associated weight w_i whose value can vary between 0 and 255. We say an entry t_i is *free* if and only if $w_i = 0$.

For a table T and a request of distance $d = 2^i$, we define the sets $E_{i,j}$ with $i = \log_2 d$ and $0 \leq j < d$, as

$$E_{i,j} = \left\{ t_{j+n\times 2^i} \quad n = 0, \ldots, \frac{64}{2^i} - 1 \right\}$$

Each $E_{i,j}$ contains the entries of the table T separated by equal distance d, that are able to meet a request of distance $d = 2^i$ starting with the entry t_j. We say a set $E_{i,j}$ is free if $\forall t_k \in E_{i,j}$, t_k is free. Other properties derived from this definition are available in [1].

For a new request of distance $d = 2^i$, our algorithm inspects all possible sets $E_{i,j}$ for this kind of request, in a certain order, and selects the first one that is free (so, it has all its entries free). The order in which the sets are inspected is based on the application of the bit-reversal permutation to the values in the interval $[0, d-1]$. Specifically, **for a new request of maximum distance $d = 2^i$, the algorithm selects the first $E_{i,j}$ free in the sequence**

$$E_{i,{}_iR_0}, E_{i,{}_iR_1}, \ldots, E_{i,{}_iR_{d-1}}$$

where ${}_iR_j$ is the bit-reversal function applied to j codified with i bits.

For example, the order to inspect the sets for a request of distance $d = 8 = 2^3$ is $E_{3,0}$, $E_{3,4}$, $E_{3,2}$, $E_{3,6}$, $E_{3,1}$, $E_{3,5}$, $E_{3,3}$, and $E_{3,7}$. Note that this algorithm first fills in the even entries and the odd entries later. In this way, if we have available entries, we can always attend a request of distance 2, the most restrictive. The same consideration can be done for other longer distances.

In [1] we have demonstrated several theorems to prove that with this algorithm we can always attend a new request if there are enough available entries. This is due to the fact that our algorithm always selects the sequences in the most optimal way to be able to later meet the most restrictive possible request. Note that this algorithm is only applied if there is not a previously allocated sequence for the same distance requested with available room in its entries.

When a connection finishes, its bandwidth is deducted from the accumulated bandwidth in the entries that it was occupying. When this accumulated bandwidth is zero those entries must be freed. When some entries are freed, a disfragmentation algorithm must be applied to leave the table in a correct way, such that the proposed filling in algorithm can be used. This disfragmentation algorithm and its properties are also described in [1]. Basically, it puts together free small sets to form a larger free set.

As we have demonstrated in [1], both algorithms together permit the meeting and release of sequences in a optimal and dynamical way. This permits us to provide QoS to the applications using in a optimal way the mechanisms provided by the InfiniBand architecture.

4. Performance evaluation

In this section, we have used simulation to evaluate the behavior of our proposals. In the following, we are going to explain the network and the traffic models we have used.

4.1. Network model

We have used irregular networks randomly generated. All switches have 8 ports, 4 of them having a host attached, and the other 4 are used for interconnection between switches. We have evaluated networks with sizes ranging from 8 to 64 switches (with 32 to 256 hosts, respectively), and, for all cases,

SL	Maximum Distance	Bandwidth Range (Mbps)
0	2	0.064 - 1.55
1	4	0.064 - 1.55
2	8	0.064 - 1.55
3	16	0.064 - 1.55
4	32	0.064 - 1.55
5		1.55 - 64
6	64	0.008 - 0.064
7		0.064 - 1.55
8		1.55 - 64
9		64 - 255

Table 1. Features of the SLs used.

the results are similar. Due to space limitation, we will only include here results for the network with 16 switches and 64 hosts. For the same reason, only results for the link rate of 2.5 Gbps will be shown.

Both of input and output ports have 16 VLs in order to permit each SL to have its own VL. Each VL is large enough to store four whole packets. Each switch has a multiplexed crossbar. So, only a VL of each input (output) port can be transmitting (receiving) at the same time.

4.2. Traffic model

We have used 10 SLs for traffic needing QoS. Each one has different maximum distance and bandwidth requirements. The SLs used are shown in Table 1. For the most demanded distances a division has been made based on the mean bandwidth of the connections. We have used CBR traffic, randomly generated among the bandwidth range of each SL.

The connections of each SL request a maximum distance between two consecutive entries in the high-priority table and a mean bandwidth in the range shown in the Table 1. Note that this is equivalent to requesting a maximum deadline and computing the maximum distance between two consecutive entries in the virtual lane arbitration tables.

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

When no more connections can be established we start a transient period in order for the network to reach a stationary state. Once the transient period finishes, the steady state period begins, where we will gather results to be shown. The steady state period continues until the connection with a smaller mean bandwidth has received 100 packets.

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

4.3. Simulation results

We can see in Table 2 the injected and delivered traffic (in bytes/cycle/node), the average utilization (in %) and the average bandwidth reserved (in Mbps) in host interfaces and switch ports. Note that the maximum utilization reachable is 80%, because the other 20% is reserved for BE and CH traffic. So,

	Packet size	
	Small	Large
Injected traffic (Bytes/Cycle/Node)	0.7183	0.7011
Delivered traffic (Bytes/Cycle/Node)	0.7183	0.7011
Av. utilization for host interfaces (%)	71.832	70.123
Av. utilization for switch ports (%)	73.096	72.350
Av. reservation for host interfaces (Mbps)	1829.699	1822.367
Av. reservation for switch ports (Mbps)	1861.969	1860.269

Table 2. Traffic and utilization for different packet sizes.

we are close to the maximum utilization. Obviously, we could achieve a higher utilization establishing more connections, but we have already made many attempts for each SL. Other connections that we could establish would be of SLs of small mean bandwidth because the network is already very loaded, and we think these new connections would not provide us more information. So, we think that with this load we can study the network behavior in a quasi-fully loaded scenario.

Note also that the behavior is quite similar for both packet sizes. Besides, for small packet size the network reaches a slightly higher throughput. This is due to the fact that the overhead introduced by packet headers is more important for small packet size and more packets must be transmitted.

We have also computed the percentages of packets that meet a certain deadline threshold. These thresholds are different for each connection and are related to their requested maximum deadline. This maximum deadline is the maximum delay that has been guaranteed to each connection. In the figures this deadline is referred to as D. The results for each SL are presented in Figure 4, for both packet sizes considered. Note that results are quite similar for both packet sizes. In these figures we can see that all

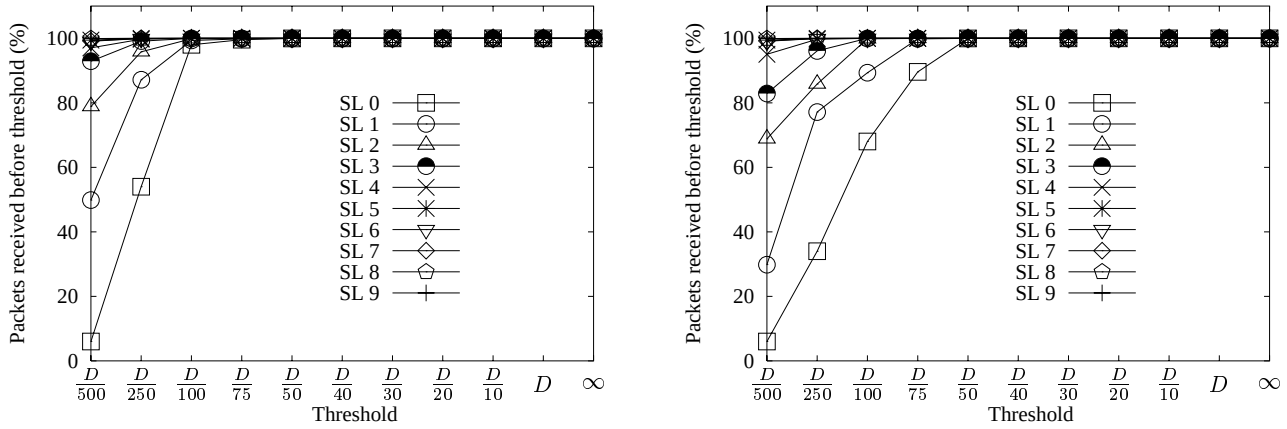

Figure 4. Distribution of packet delay for (a) small packet size and (b) large packet size.

packets of all SLs arrive to their destinations before their deadlines. However, the packets of SLs with stricter deadlines arrive to their destination near to their deadline, but in time to achieve its requirements.

We have also measured the average packet jitter. We have computed the percentage of packets received in several intervals related to their interarrival time. Obviously, these intervals are different for each connection. The results for each SL and small packet size are shown in Figure 5. For large packet size results are quite similar. In all cases, we can see that almost all packets arrive in the central interval $[\frac{-IAT}{8}, \frac{IAT}{8}]$. For the SLs of which connections have the smallest mean bandwidth (SLs 0, 1, 2, 3, 4, 6, and 7), all packets arrive in the central interval. This is because these connections have a large interarrival time, large enough for all packets to arrive to their destinations. For the other SLs, with the biggest mean bandwidth, the jitter has a Gaussian distribution never exceeding $\pm IAT$.

Finally, for a given threshold, we have selected the connections having delivered the lowest and the highest percentage of packets before a threshold. In the figures these connections will be referred to as the worst and the best connections, respectively. We have selected a very tight threshold so that the

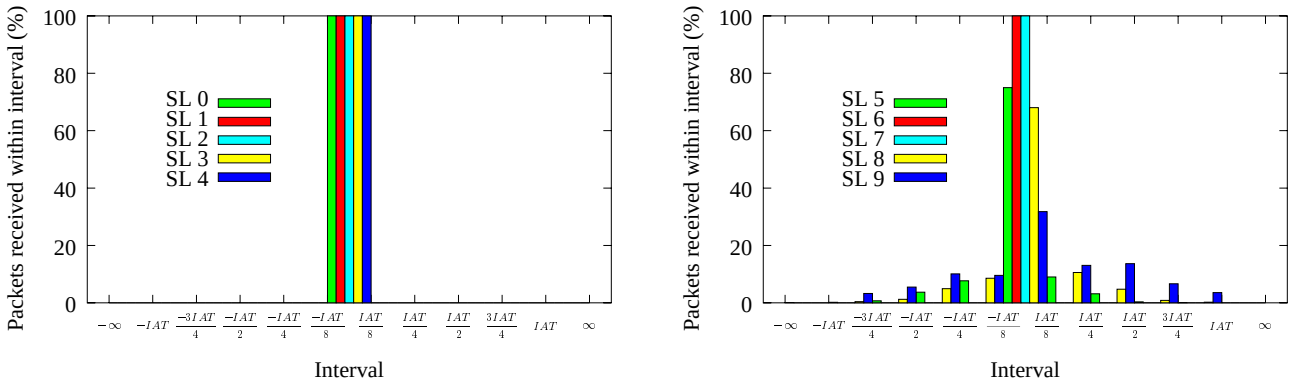

Figure 5. Average packet jitter for small packet size (a) for SLs 0, 1, 2, 3, and 4, and (b) for SLs 5, 6, 7, 8, and 9.

percentage of packets meeting the deadline was lower than 100% in Figure 4a. In particular we have selected the threshold equal to $\frac{Deadline}{100}$. Note again that this threshold is different for each connection and it is based on its own maximum deadline. The results shown in the Figure 6 correspond to the small packet size and the SLs 0, 1, 2, and 3, which are the SLs with the highest deadline requirements. The results for the other SLs are even better than these shown in the figures. We can observe that, in all cases, even the packets of the worst connection arrive to their destination before their deadline. We can also see that in all cases the results are very similar for the best and the worst case. This is due to the fact that the arbitration tables are set in a correct way and all the connections receive a good and similar treatment.

5. Conclusions

InfiniBand is a new industry-standard architecture for server I/O and interprocessor communication. InfiniBand has some mechanisms that properly used provide QoS to the applications. In previous works we have already explained these mechanisms.

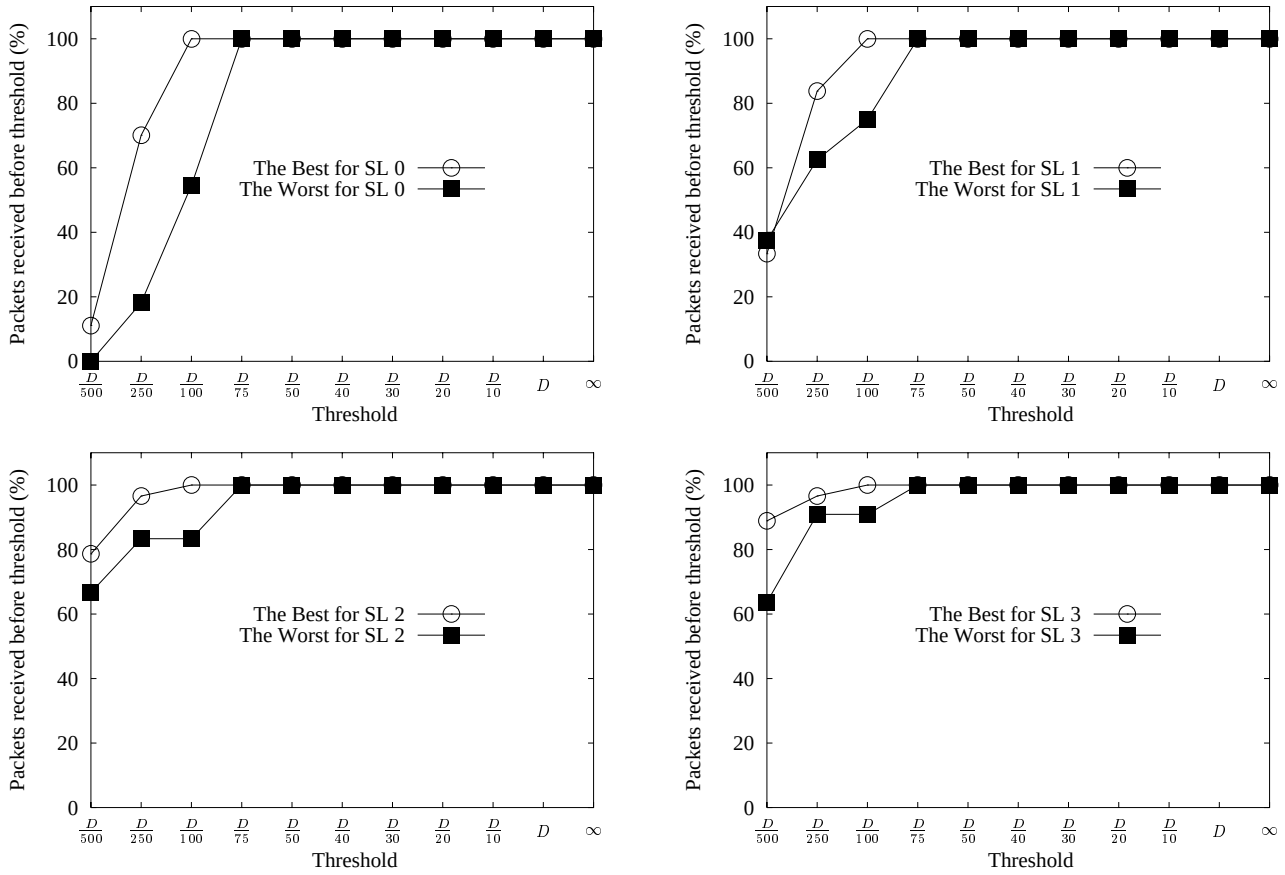

Figure 6. The best and the worst connection for SLs with the strictest latency requirements.

In [3] we proposed a new methodology to compute the virtual lane arbitration tables of InfiniBand for traffic having only bandwidth requirements. In [2] we extended this proposal to also include traffic with latency requirements. We evaluated this proposal obtaining very good results.

However, our proposal had the problem that no guarantees could be done to traffic that uses the low-priority table if sources of traffic using the high-priority one used more bandwidth that they previously requested. In this paper we have presented a new approach that solve this problem. We treat all kinds of traffic in the same way grouping it based on its latency requirements. Besides, all traffic with QoS requirements will use the high-priority table, and so, all of them have guaranteed what they requested. If some source sends more than it requested this will only affect the connections sharing the same VL,

but the rest of the traffic in others VLs will achieve what they requested.

We have also proposed an algorithm to select a sequence of entries in the table. This algorithm successfully allocates a request in the table if there are enough available entries. It sets the requests in an optimal way being able later to put the most restrictive request that we could set according to the available entries. This algorithm has some formal properties and theorems derived from it, that we have showed in [1].

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

Finally, as future work we want to test this methodology in an InfiniBand commercial product as soon as we have one available, in order to verify the practicality of our model.

References

[1] F. Alfaro, J. Sánchez, M. Menduiña, and J. Duato. Formalizing the Fill-In of the Infiniband Arbitration Table. Technical Report DIAB-03-02-35, Departamento de Informática Universidad de Castilla-La Mancha, Mar. 2003.

[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, 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)*, April 2002.

[4] F. J. Alfaro, J. L. Sánchez, L. Orozco, and J. Duato. Performance Evaluation of VBR Traffic in InfiniBand. In *Proceedings of IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'02)*, pages 1532 – 1537, May 2002.

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

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

[7] J. Pelissier. Providing quality of service over Infiniband architecture fabrics. In *Proceedings of the 8th Symposium on Hot Interconnects*, Aug. 2000.

[8] 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.