Tuning Buffer Size in InfiniBand to Guarantee QoS[*]

Francisco J. Alfaro and José L. Sánchez

Universidad de Castilla-La Mancha, 02071-Albacete, Spain
{falfaro,jsanchez}@info-ab.uclm.es

Abstract. InfiniBand (IBA) is a new industry-standard architecture both for I/O server and interprocessor communication. IBA employs a switched point-to-point network, instead of using a shared bus. IBA is being developed by the InfiniBand^{SM} Trade Association to provide present and future server systems with the required levels of reliability, availability, performance, scalability, and quality of service (QoS).
In previous papers we have proposed an effective strategy for configuring the IBA networks to provide users with the required levels of QoS. This strategy is based on the proper configuration of the mechanisms IBA carries to support QoS. In this paper, we determine the minimum virtual lines' buffer size required to guarantee QoS to the applications.

1 Introduction

The InfiniBand Trade Association (IBTA) [1] was formed in 1999 to develop a new standard for high-speed I/O and interprocessor communication. InfiniBand defines a technology for interconnecting processor nodes (hosts) and I/O devices to form a system area network [2]. In a first stage, instead of directly replacing the PCI bus with a switch-based interconnection to access I/O devices, these devices are attached to a Host Channel Adapter (HCA), which is connected to the PCI bus. In this way, the communication is switched from HCA, affording the desired reliability, concurrency and security. Moreover, it is foreseen that the PCI bus could be replaced in a near future by other advanced technologies like PCI Express Advanced Switching.

InfiniBand implements some mechanisms to provide each kind of application with the required QoS. In previous works, [3] and [4], we have developed a methodology to configure such mechanisms. The proposed methodology successfully provides applications with both bandwidth and latency guarantees. In this paper, we determine the minimum buffer size in the virtual lanes of the switch ports and host interfaces required to achieve this goal.

The structure of the paper is: Section 2 presents a summary of the most important mechanisms included in IBA to support QoS; in Section 3, we review our proposal to give QoS guarantees; Section 4 presents the evaluation methodology used to determine the buffer size required in the VLs to provide applications with QoS, as well as the obtained results; finally, some conclusions are given.

[*] This work was partly supported by the Spanish CICYT under Grant TIC2003-08154-C06 and Junta de Comunidades de Castilla-La Mancha under Grant PBC-02-008.

2 InfiniBand

InfiniBand hardware provides highly reliable, fault-tolerant communication, improving the bandwidth, latency, and reliability of the system. 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.

Specifically, IBA has three mechanisms to support QoS: Service levels (SLs), virtual lanes (VLs), and a virtual lane arbitration for transmission over links. IBA defines a maximum of 16 SLs, although it does not specify which characteristics the traffic of each service level should have. Therefore, the distribution of the different existing traffic types among the SLs may be stated by the manufacturer or the network administrator. By allowing the traffic to be segregated by categories, we will be able to distinguish between packets from different SLs and to give them a different treatment according to their needs.

IBA ports support VLs as a mechanism for creating multiple virtual links within a single physical link. Each VL must be an independent resource for flow control purposes. A VL represents a set of transmission and reception buffers in a port. IBA ports can support a minimum of two and a maximum of 16 VLs. Since systems can be constructed with switches supporting a different number of VLs, the number of VLs used by a port is configured by the subnet manager. Moreover, 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*.

When more than two VLs are implemented, the priorities of the data lanes are defined by the *VLArbitrationTable*. This arbitration affects only to data VLs, because control traffic uses its own VL, which has greater priority than any other VL. The VLArbitrationTable consists of two tables, one for scheduling packets from high-priority VLs and another 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 transmitted from that VL. This weight must be in the range from 0 to 255, and is always rounded up as a whole packet.

Moreover, a $LimitOfHighPriority$ value specifies the maximum number of high-priority packets that can be sent before a low-priority packet is transmitted. Specifically, the VLs of the high-priority table can send $LimitOfHighPriority \times 4096$ bytes before a packet from the low-priority table can be transmitted. If at a given time, no high-priority packets are ready for transmission, low-priority packets can also be transmitted.

3 Our Proposal to Give QoS Guarantees

In previous works we have proposed a simple strategy to treat the requests of latency guarantee. Specifically, when an application requests latency guarantee, the maximum distance allowed between two consecutive entries in the high-priority table must be computed in order to allocate entries on that table to that application. Moreover, the application could also request a mean bandwidth that would result in a weight to put in the entries of the arbitration table. Therefore, for a certain connection that requests a maximum delay that results in a distance d, and a mean bandwidth that results 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 requested by a connection ranges from 1 to 64. However, in order to optimize the filling up of the table, we only consider the following distances: 2, 4, 8, 16, 32, and 64 [5]. Therefore, the applications' requests of a maximum distance between two consecutive entries in the high-priority table are turned into the closest lower power of 2 [4].

Traffic is grouped in SLs according to its maximum latency. Specifically, all connections using the same SL need the same maximum distance between two consecutive entries in the high-priority table, regardless of their mean bandwidth. For the most requested distances, we could distinguish between two or four different SLs considering the mean bandwidth. In this way, if we have enough available VLs, each kind of traffic could use a different VL.

Moreover, in [4] and [5] we proposed an algorithm to select a free sequence of entries in the high-priority table to meet a new application's request. This algorithm successfully allocates a new sequence in the table if there are enough available entries. This is achieved because the available entries are always in the best situation to treat the most restrictive request. For a connection requesting $\frac{64}{d}$ entries with a maximum distance d between them, the algorithm looks for a previously established sequence, for the corresponding VL, with enough available weight. If there is no available sequence, a new free sequence with those characteristics is looked for.

In a more formal way, in a table T, let the sequence $t_0, t_1, \ldots, t_{62}, t_{63}$ represent the entries of the table. Each t_i has an associated weight w_i whose value can vary between 0 and 255. Thus, 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 spaced by an equal distance d which 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 [5].

In [4] we also presented a simple algorithm to maximize the number of requests allocated in the arbitration table. For a new request of distance $d = 2^i$, the algorithm studies all possible sets $E_{i,j}$ for this kind of request, in a certain order, and selects the first set that is free (so, all its entries are free). The order the sets are inspected is based on the application of the bit-reversal permutation to the distance values in the interval $[0, d-1]$. Specifically, for a new request of maximum distance $d = 2^i$, the algorithm selects the first free $E_{i,j}$ 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. Note that this algorithm is only applied if there is no previously allocated sequence for the same requested distance with available room in its entries.

For example, the order the sets are inspected for a request of distance $d = 2^3$ 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 later the odd entries. In this way, if we have available entries, we can always meet a request of distance 2, which is the most restrictive. The same consideration can be made for longer distances.

In [5], we have also proved several theorems showing that the algorithm can always allocate a new request if there are enough available entries. This is achieved because the algorithm always selects the sequences in the optimal way for satisfying later the most restrictive possible request.

When a connection finishes, its bandwidth is deducted from the accumulated bandwidth in the entries it was occupying. When this accumulated bandwidth is zero those entries must be released. When some entries are released, a disfragmentation algorithm must be applied to leave the table in a correct state, such that the proposed filling in algorithm can be used. This disfragmentation algorithm and its properties are also described in [5]. Basically, it puts together small free sets to form a larger free set, moving the content of some entries.

Both algorithms together permit the allocation and release of sequences of entries in the arbitration table in a optimal and dynamical way [5]. This allows us to provide applications with QoS using in an optimal way the IBA mechanisms.

4 Performance Evaluation

In [4] and [5] we have evaluated the behavior of our proposals using a large buffer size. We have shown that our proposals are able to provide applications with QoS guarantee. In this section, we are going to determine the minimum switch port and host interface VLs buffer size our proposals require to satisfy the QoS requirements. In the following points, we 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, the other 4 being used for interconnection between switches. We have evaluated networks with sizes ranging from 8 to 64 switches (so, with 32 to 256 hosts, respectively). We have also tested several packet sizes ranging from 256 to 4096 bytes, and the three link rates specified in IBA. Taking into account that all these variations present similar results and the space limitation, we have only included here results for the 16 switches-network using a packet size of 256 bytes and a link rate of 2.5 Gbps.

In the switches both at input and output ports, there are 16 VLs in order to assign a different VL to each SL. Each switch has a multiplexed crossbar. We will test several buffer sizes. As IBA uses virtual cut-through we have only considered buffer sizes that allow to store completely whole packets. Specifically, we have considered buffer sizes of 1, 2, 3 or 4 whole packets of capacity.

Table 1. Features of the SLs used.

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

4.2 Traffic Model

We have used 10 SLs for traffic needing QoS. Each SL presents a different maximum distance and bandwidth requirements. We have used CBR traffic, randomly generated among the bandwidth range of each SL. For the most requested distances several SLs have been considered using the mean bandwidth of the connections. Specifically, the SLs used and their features are shown in Table 1.

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 Table 1. Note that this is similar to requesting a maximum deadline and computing the maximum distance between two consecutive entries in the virtual lane arbitration tables.

Each request is considered by each node along its path and is accepted only if there are available resources. Connections of the same SL are grouped into the same sequence of entries. The total weight of the sequence is computed according to the accumulated bandwidth of the connections sharing that sequence. When the connection cannot be settled in a previously established sequence (or there is no previous one), our algorithm looks in the high-priority arbitration table for a new free sequence of entries 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 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 types of traffic, which would be served by the low-priority table. So, connections would only be established up to 80% of the available bandwidth.

4.3 Simulation Results

We can see in Table 2 several metrics measured for the different buffer sizes considered. Specifically, we have computed the injected and delivered traffic (in

Table 2. Traffic and utilization for different buffer sizes.

	Buffer size (in packets)			
	1	2	3	4
Injected traffic (Bytes/Cycle/Node)	0,7262	0,7262	0,7258	0,7258
Delivered traffic (Bytes/Cycle/Node)	0,6788	0,7262	0,7258	0,7258
Av. utilization for host interfaces (%)	67,88	72,63	72,58	72,58
Av. utilization for switch ports (%)	68,06	73,49	73,48	73,48
Av. reservation for host interfaces (Mbps)	1849,81	1849,81	1848,67	1848,67
Av. reservation for switch ports (Mbps)	1871,84	1877,84	1871,75	1871,75

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, we are close to the maximum utilization achievable. Obviously, we could achieve a higher utilization establishing more connections, but we have already made many attempts for each SL. We could establish other connections, but these connections would be of SLs of a small mean bandwidth because the network is already heavily loaded, and it is unlikely that these new connections would provide us with more information. So, it seems reasonable to assume 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 all the buffer sizes considered. Regardless of the buffer size used, the network reaches a similar throughput and the reservation performed in the host interfaces and switch ports is quite similar. However, there is an important difference for the buffer size of one packet. In this case, the network cannot transmit all the packets it receives, and delivered traffic is lower than injected traffic. As we will see, this causes long waitings for the packets. However, for other buffer sizes the injected traffic is equal to the delivered traffic, thereby showing that the network is able to transmit all the packets it receives.

We have also computed the percentages of packets that meet a certain deadline threshold. These thresholds are different for each connection and are related

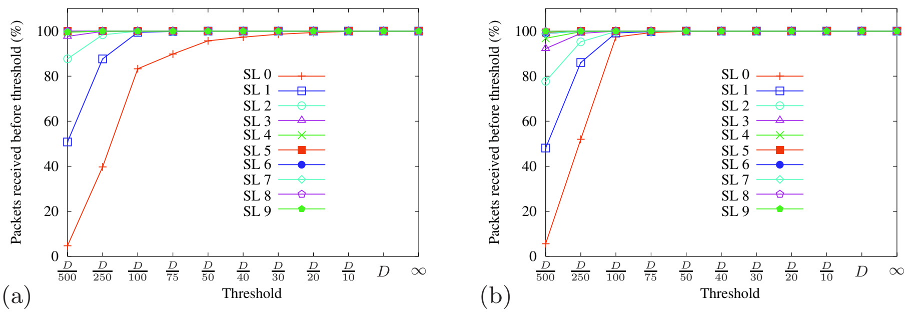

Fig. 1. Packet delay for buffer sizes of (a) 1 packet and (b) 2 packets.

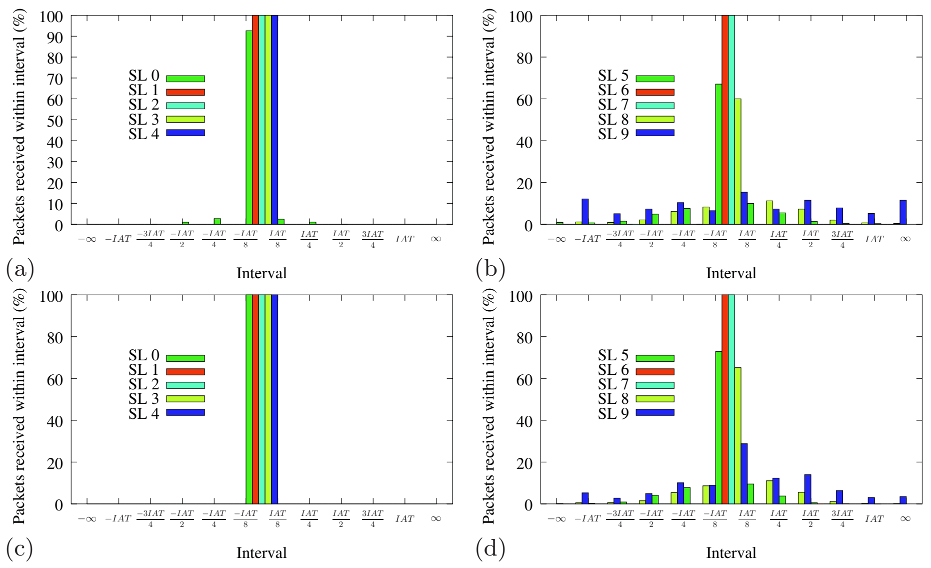

Fig. 2. Packet jitter for buffer sizes of (a) and (b) one packet, (c) and (d) two packets.

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 Fig. 1 for buffer size of one and two packets. Results for larger buffer sizes are quite similar to those obtained for two packets. In these figures, we can see that all packets of all SLs arrive at their destinations before their deadlines. However, packets of SLs with stricter deadlines arrive at their destination closer to their deadline, although in time to meet their 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 are shown in Fig. 2. We have only shown results for buffer sizes of one and two packets. We can see that results for buffer size of one packet are worse than for two packets. Results for a larger buffer size are quite similar to those of buffer size of two packets. Moreover, results for the buffer size of two packets are much better than those obtained for just one packet, although almost all packets arrive at their targets in the central interval $[\frac{-IAT}{8}, \frac{IAT}{8}]$. For other SLs with bigger mean bandwidth the jitter has a Gaussian distribution never exceeding $\pm IAT$. Note that packets of SLs with less priority (SLs 9 and 8) present a worse behavior for buffer size of one packet than for buffer size of two packets.

Finally, for a given deadline threshold, we have selected the connections that deliver the lowest and the highest percentage of packets before this 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

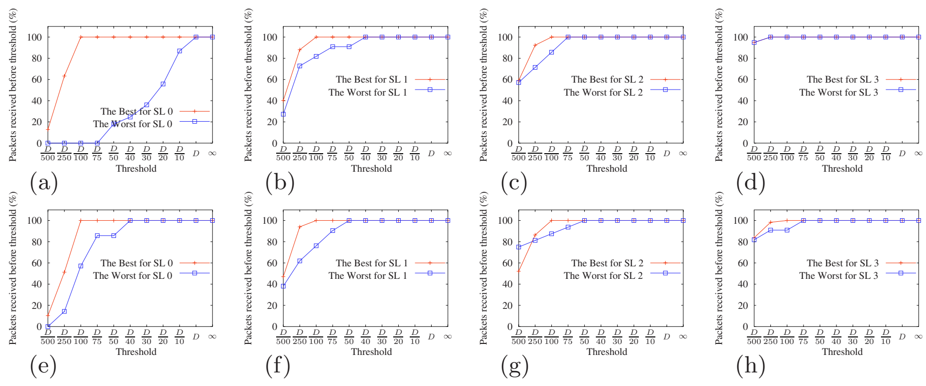

Fig. 3. The best and the worst connection for SLs with the strictest delay requirements, for buffer sizes of (a), (b), (c) and (d) one packet, and (e), (f), (g) and (h) two packets.

percentage of packets meeting the deadline was lower than 100% in Fig. 1. In particular, we have selected the threshold equal to $\frac{Deadline}{100}$. Note again that this threshold is different for each connection and depends on its own maximum deadline. Fig. 3 shows the results for buffer sizes of one and two packets and for the SLs 0, 1, 2, and 3, which are the SLs with the highest deadline requirements. The results for other SLs are even better than these shown in the figures. It is noteworthy that, in all cases, even the packets of the worst connection arrive at their destination before their deadline. We can also see that for buffer size of two packets, results are very similar in the best and the worst case. However, for buffer size of one packet, SL 0 (which has the highest deadline requirements), has a very different behavior. This is due to the long waitings of the packets in the buffers caused by the lack of available room in the next buffer of their paths.

5 Conclusions

In [4] and [5] we proposed a new methodology to provide each kind of applications with the previously required QoS level. We also proposed an algorithm to select a free sequence of entries in the arbitration table. This algorithm successfully allocates a request in the arbitration table if there are enough available entries. It manages the requests in an optimal way, being able later to satisfy the most restrictive possible request. Some formal properties and theorems derived from this algorithm are shown in [5].

In this paper, we have determined the minimum virtual lanes' buffer size required to provide applications with QoS guarantee. We have tested several buffer sizes ranging from 1 to 4 packets. The most important result may well be that for all buffer sizes our proposals meet the QoS requirements. However, important waitings for buffer size of one packet have been observed. These waitings disappear when we have a buffer size of two or more packets.

These are clearly important results, which show that our methodology does not require a great deal of space resources to achieve QoS in InfiniBand environments. We configure in an accurate way the mechanisms that InfiniBand provides to support QoS, with a minimum of buffer room in switch ports and host interfaces.

References

1. InfiniBand^{TM} Trade Association: `http://infinibandta.com`. (1999)
2. InfiniBand Trade Association: InfiniBand Architecture Specification Volume 1. Release 1.0. (2000)
3. Alfaro, F.J., Sánchez, J.L., Duato, J., Das, C.R.: A Strategy to Compute the InfiniBand Arbitration Tables. In: Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'02). (2002)
4. Alfaro, F.J., Sánchez, J.L., Duato, J.: A New Proposal to Fill in the InfiniBand Arbitration Tables. In: Proceedings of IEEE International Conference on Parallel Computing (ICPP'03). (2003) 133 – 140
5. Alfaro, F., Sánchez, J., Menduiña, M., Duato, J.: Formalizing the Fill-In of the Infiniband Arbitration Table. Technical Report DIAB-03-02-35, Dep. de Informática Universidad de Castilla-La Mancha (2003)