QoS Provision in Hierarchical and non-Hierarchical Switch Architectures

Javier Cano-Cano[a,*], Francisco J. Andújar[b], Francisco J. Alfaro[a], José L. Sánchez[a]

[a] *Computing System Department, University of Castilla-La Mancha, 02071, Albacete, Spain*

[b] *Computing System Department, Universidad de Valladolid, 47011, Valladolid, Spain*

[*] Correspondence to: Instituto de Investigación en Informática de Albacete, Campus Universitario s/n, 02071, Albacete, Spain.

Email addresses: javier.cano@uclm.es (Javier Cano-Cano), fandujarm@infor.uva.es (Francisco J. Andújar), fco.alfaro@uclm.es (Francisco J. Alfaro), jose.sgarcia@uclm.es (José L. Sánchez)

Preprint submitted to Journal of Parallel and Distributed Computing *October 10, 2024*

Abstract

Quality of service (QoS) provision has become an important aspect of high-performance computing interconnection networks. Proof of that is the inclusion of mechanisms targeted to the provision of QoS by the main interconnection technologies such as Gigabit Ethernet, Infiniband (IB) and Omni-Path (OPA). A key component of QoS provision is the output scheduling algorithm, which determines when a packet should be transmitted. An ideal scheduling algorithm should satisfy two main properties: good end-to-end latency and implementation simplicity. Table-based schedulers are able to provide these two properties, and because of this, IB and OPA have implemented this approach.

In this paper, we present a comparative study in terms of QoS provision between these two dominating interconnection technologies. Those interconnection technologies are also two examples of non-hierarchical and hierarchical switch architectures, respectively, which gives the results of this study greater significance. In order to carry out the study, the Deficit Table scheduler (DTable) has been used. DTable is a table-based scheduling algorithm which offers a good balance between end-to-end latency and implementation cost.

Keywords: Quality of Service, Scheduling Algorithms, Interconnection Networks, Omni-Path, Infiniband, Simulation, Hierarchical-crossbar-switch Architecture, Modeling and simulation tool, Performance evaluation

1. Introduction

In high performance computing (HPC) systems, composed of up to thousands of nodes, several applications are executed concurrently, generating a lot of messages in the network. Since each node can produce a lot of data at a faster rate, network contention may appear, and applications performance may be affected. To reduce the impact of contention, Quality of Service (QoS) mechanisms can be used, providing applications with the necessary resources to meet their requirements in the best possible way. For this reason, the main interconnection networks technologies, InfiniBand (IB) and Omni-Path (OPA) technologies, include QoS support.

QoS is still an active research topic in the two main environments where high-speed interconnection networks are used: Internet datacenters and supercomputing platforms [1]. The increasing use of Internet has been one of the most dominant contribution to the need of QoS. New application types have appeared [2], such as on-demand video streaming, live video/voice communications, interactive applications, etc. Providing QoS is necessary to satisfy the application requirements and to provide a good user experience. In the area of HPC, QoS provides differentiate services between traffic flows, i.e. between users with different privilege levels in a cluster [3, 4], or between users with different service level agreements in a cloud. For those reasons, many studies have been carried out to provide QoS on the Internet using wired or wireless networks [5, 6], and to provide QoS on HPC, such as new output schedulers [7], techniques for meeting QoS requirements [8, 9], etc.

A key component for networks with QoS support is the output scheduling algorithm, which determines when a packet should be transmitted, on the

basis of some expected performance metrics [10, 11]. In an HPC interconnection network, packets from different applications interact with each other in every network element. Without any scheduling policy, packets from different traffic applications flows use as many resources as they need. In the worst scenario, a single flow may consume all network resources, causing a degradation of the performance of other flows or, even worst, starvation.

An ideal scheduling algorithm implemented in an HPC interconnection network should provide *fairness* between the traffic flows (depending on their required service level), should provide a *good end-to-end latency* and should be *simple* in terms of computational and implementation complexity. However, designing an output scheduling algorithm involves inevitable trade-offs among the stated properties. The simplicity is the most important property. The fairness property only affects to the short-term distribution of the service offered to the flows sharing the link. End-to-end latency implies burstiness of the flow at the output of the scheduler, thus increasing the buffering requirements in order to avoid packet contention.

A well-known scheduling algorithm family is sorted-priority schedulers, which use a global variable, called virtual time, to keep track of the server's progress and is updated when a packet is received or transmitted. Each packet has a time-stamp, computed as a function of the virtual time by the specific scheduling algorithm. Two of the most common examples of this family of schedulers are Weighted Fair Queuing (WFQ) [10] and Self-Clocked Fair Queuing (SCFQ) [12]. They provide a good fairness and a low latency, but they are not very efficient due to the complexity of computing the virtual time and maintaining a time-sorted packet list. However, some works have

shown how to reduce the complexity of WFQ and SCFQ maintaining decent QoS provisioning [13, 14, 15].

Table-based scheduling algorithms are another well-known family. They are based on a table with multiple entries assigned to one or more flows. Each entry has a weight that determines the number of packets that may be transmitted. This family of schedulers is able to provide good end-to-end latency, controlling the maximum separation between any consecutive pair of entries of the same flow [16], and presents a very low computational and implementation complexity. Generally, the table only requires a pointer indicating the last entry selected [17, 18] and a number of entries ranging from 64 to 128. Both IB and OPA have implemented this approach [19, 20, 21].

In this paper, we present a comparative study in terms of QoS provision of two representative HPC interconnection technologies such as IB and OPA. In the most powerful computer list TOP500 [22], they are well established. Up to 9.8% of these supercomputer are using OPA and up to 30.6% are using IB as their interconnection network. Looking at top 100, both technologies are dominating, 20% of these systems use OPA and 40% use IB (June, 2020). Moreover, these two interconnection technologies are also two examples of different switch architecture, hierarchical and non-hierarchical, respectively. Hierarchical switch architectures were designed to bring together the advantages of high-radix [23, 24, 25, 26, 27] and low-radix switches [28, 29, 30].

Furthermore, in order to carry out the study we will use DTable [31], a table-based scheduling algorithm which offers a good balance between performance and hardware cost. To our knowledge, there are no studies which compare the performance of non-hierarchical and hierarchical switches in terms

of QoS provision being this issue important to compare both approaches.

The structure of the paper is as follows: Section 2 briefly reviews the switch architecture of IB and OPA and describes their simulation models. Section 3 reviews the QoS support in both technologies. Section 4 explains the DTable scheduler operation, as well as how it could be adapted to IB and OPA. Section 5 includes the experimental study design and Section 6 shows and analyzes its results. Finally, Section 7 presents some conclusions.

2. Hiperion simulator

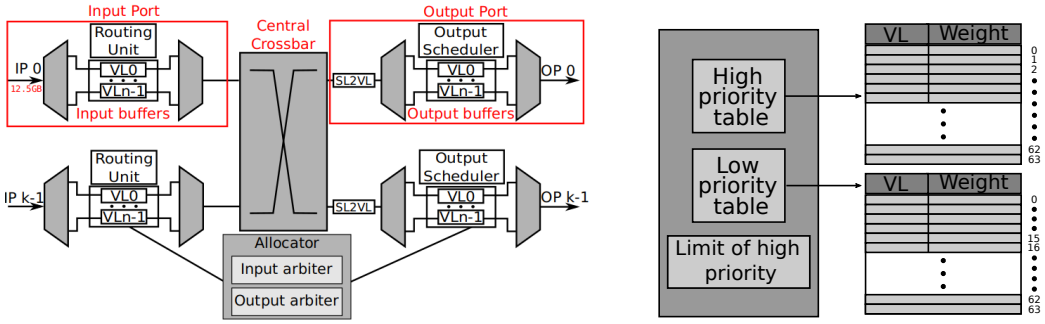
As stated in Section 1, IB and OPA interconnection technologies are used to carry out this study. This work has been carried out using simulation for being one of the most popular methodology to evaluate different techniques in HPC networks. It allows to test, compare and explore new techniques in a cheap, flexible and reproducible way. There are multiple HPC network simulators such as Garnet [32], Booksim2 [33], xSim [34], etc., focused on on-chip networks. These simulators also allow full-system simulation, feasible for on-chip networks, due to the small network sizes (rarely more than 64 switches). However, when the network size grows to hundred of elements, the computational resources needed makes the full-system simulation unapproachable. Moreover, the characteristics of the on-chip traffic and off-chip traffic are totally disparate. On the other hand, there are no public simulators modeling the OPA switch architecture. For these reasons, we have developed our IB and OPA models in a previous simulator, which has been used for years in our research group, and with multiple publications behind him [35, 36]. Our simulator Hiperion (HIgh PERformance InterconnectiOn

Network) gives us a deep knowledge of its operation and a wide flexibility regarding the techniques that can be implemented and its interoperability [37]. Hiperion is a discrete-event based network simulator that models the behavior of HPC network elements, such as switches, links and network interfaces. The simulator main goal is to perform comparative studies and it has a large range of configurable parameters, e.g. topology, routing, queue sizes, output scheduling algorithms, etc. Hiperion is capable of running simulations using synthetic traffic (e.g. random uniform, bit-reversal, bit-complement, etc.). Multiple performance metrics have been implemented such as end-to-end latency, throughput, etc.

2.1. Infiniband simulation model

This section details the main components of our IB-like simulation model and its behavior. Figure 1a shows a generic scheme of a k port IB-based switch where k is the total number of ports. The IB switch implements virtual cut-through as switching technique and a credit-based flow control. We have defined the input/output bandwidth to 12.5 GBps. The model assumed in Figure 1a includes the following elements:

- Input/output buffers: They store the flits from the input/output ports. There is one input/output buffer per input/output port. Buffer storage space is dynamically shared by virtual lanes (VLs). The dynamic buffers provides more flexibility than static buffers [38].
- Routing units: They add the routing information to header flits. There is one routing unit per input port.
- Central crossbar: It interconnects input ports to required output ports.



(a) IB-based k-port switch.

(b) IB output scheduler.

Figure 1: Diagram of IB architecture.

- Input arbiter: Given an input buffer, it selects the VL that participates in the next allocator phase. The more VLs, the bigger the arbiter is.
- Output arbiter: Given an output port, it chooses which input port is going to transmit flits on that output port.

As stated before, Hiperion is a discrete-event based simulation tool. We have defined and implemented a collection of events. Those events are:

- IB (Input Buffering): Each input buffer may receive 1 flit/cycle. Flits are stored in the corresponding VL queue, depending on the transmission VL. A header packet flits are labeled as *RT-ready* and triggers the RT event. Otherwise, the flit is just stored and labeled as *X-ready*.
- RT (RouTing): Only applied to header packet flits labeled as *RT-ready*. The routing unit determines in which output port will be placed on the entire packet. After RT, the header flit is labeled as *VA-SA-ready* and its input buffer/VL can be eligible for the VA-SA stage. Note that non-header flits always follow the header flit path. The routing function is configurable and must be according to the configured topology.

- VA-SA (Virtual Allocator / Switch Allocator): A two-staged allocator:
  - Virtual Allocator: Each input arbiter chooses a VL with at least one *VA-SA-ready* header flit. The winning VL will be allowed to deliver a packet. VLs are chosen in a round-robin order.
  - Switch Allocator: Each output arbiter chooses an input buffer with a winning VL whose packet is destined to its output port. The winning buffer will be allowed to move a packet to an output buffer. The top header flit at input buffer is tagged as *X-ready*.
- X (Xbar): Winning packets from the allocation process are moved from the input buffer to the output buffer selected in the RT stage. Flits that reach the requested output buffer are tagged as *OB-ready*.
- OB (Output Buffering): Each output buffer with *OB-ready* flits chooses which VL will send flits to the neighbor network element (i.e. switch or network interface). The VL selection process is performed by a configurable output scheduling algorithm, being this event in charge of the QoS provision. Each output scheduler selects a VL with *OB-ready* packets and available credits. When the tail packet flit is transmitted, the output scheduler releases the winning VL and selects a new VL.

2.2. Omni-Path simulation model

A generic diagram of a 48 ports OPA-like switch can be found in Figure 2, which is based in [21, 39]. As can be seen in Figure 2, it has a large range of internal link with different bandwidths. The base bandwidth is 12.5 Gbps, thereby, an x3 port means that it may deliver 3 flits/cycle, which allows us to reach those bandwidths. The number of input and output ports is represented in the figure as `INPORTS:OUTPORTS`. For instance, the MPort0

xBar has 4 input ports and 6 output ports (4:6). The model shown in this figure includes the following elements:

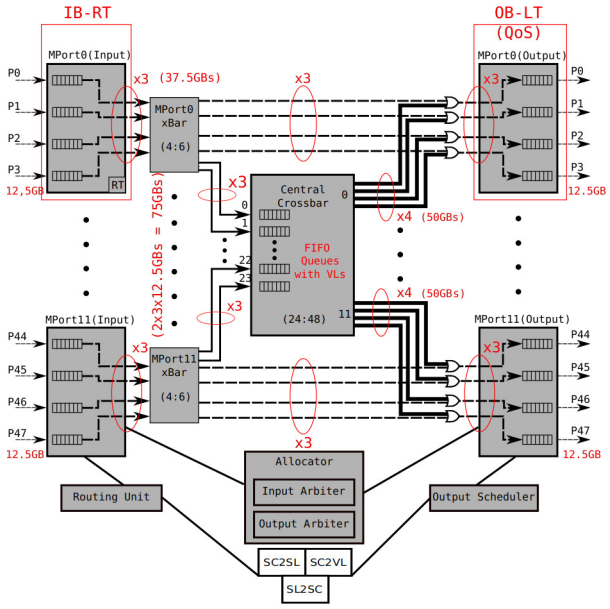


Figure 2: Diagram of the modeled OPA switch of 48 ports. For clarity, MPorts are unfolded in Input and Output buffers.

- MPort: It is a cluster of 4 input and 4 output ports and their input/output buffers, there is one buffer per input and output port. Ports belonging to the same MPort are directly connected each other through an MPort crossbar (MPort Xbar). MPorts are also connected to the rest of MPorts through the Central Crossbar.
- MPort Xbar: This crossbar has 4 input ports, one per input buffer; and 6 output ports: 4 connected to output buffers and 2 connected to the Central Crossbar. Note that the 75 Gbps link to the Central Crossbar is modelled with 2 links. Also, those links may deliver 3 flits/cycle, resulting on $2Links \times 3 \times (12.Gbps) = 75Gbps$.

- Central Crossbar: This is a buffered 24:48 crossbar. It provides connectivity for all MPorts. Note that the Central Crossbar is used when a packet requests an output port that does not belong its input MPort. For instance, a new packet arrives to the port P0, which belongs to the MPort0. The packet requests the output port P44, which belongs to the MPort11. Then, the packet will need to cross the Central Crossbar. Otherwise, if for example, the packet requests the port P2 (MPort0), it will not need to cross the Central Crossbar, only the MPort Xbar 0.

Note that the remaining elements work in the same way that the ones explained in Section 2.1. The simulation events defined in this OPA-like simulation model are the same explained in Section 2.1. However, when a packet is stored in the Central Crossbar buffers, this packet is tagged again as *VA-SA-ready* being forced to face the allocation process. After the arbitration process, the packet is tagged as *X-ready* and it can be moved to the corresponding output port at X event execution. After reaching the output buffer, the packet is tagged as *OB-ready*. Finally, the OB event explained in Section 2.1 is triggered and executed as stated. Further details about the OPA simulation model can be found at [40, 41].

3. Interconnection network QoS support

In this section we review the QoS mechanisms offered by IB and OPA to provide QoS to applications, flows, packets, etc. All the mechanisms detailed in Sections 3.1 and 3.2 have been implemented into Hiperion.

3.1. Infiniband

The IB architecture has mechanisms enabling QoS support. According to [19, 20, 42], these mechanisms are:

- Virtual lines (VLs) provide dedicated buffer space for packets. IB implements a credit-based flow control at VL-level. IB supports up to 16 VLs being the last VL reserved for management purposes.
- Service Levels (SLs) are the only QoS identifier stored in packets. SLs are used to aggregate flows with similar characteristics in order to provide applications with QoS in the network. The packet SL is set by the network interface (NIC) before its injection, and cannot be modified.

Every switch has an SL-to-VL mapping table (SL2VL) per output port, as can be seen in Figure 1a. Through these tables, the packets are assigned to a VL based on their SL, output port and input port. Note that SL2VL tables are placed in the output ports and each switch has its own tables, so that, packets may be assigned to different VLs along their route depending on the SL2VL tables at each output port.

As shown in Figure 1a, IB defines an output scheduling algorithm per output port. This scheduler is a table-based scheduler which uses two tables, as can be seen in Figure 1b. The first table schedules packets from high priority VLs while the second one schedules packets from low priority VLs. Each table has up to 64 entries and each table entry has a VL identifier and a weight in the range of 0 to 255. The weight indicates the number of units of 64 bytes to be transmitted from that VL. Entry weights are always rounded up in order to allow the transmission of an entire packet. IB output scheduling algorithm also defines a value ($LimitOfHighPriority \times 4096$) representing the maximum amount of information that may be delivered from the high priority table before transmitting a packet from the low priority table. The *LimitOfHighPriority* value is configurable by the network administrator.

When arbitration is needed, the table is cycled through sequentially until a table entry with an active VL is found, that is, a VL with stored packets and enough credits. Then the VL can transmit a maximum of as many packets as the table entry weight states. Finally, the table is cycled through again.

Moreover, if there are no high priority packets stored in an output port, the arbiter will allow to transmit low priority packets (if there is any) until a high priority packet arrives. This mechanism avoids to waste bandwidth.

3.2. Omni-Path

The OPA architecture also has some mechanisms enabling QoS support to packets, flows, applications, etc. According to [21], these mechanisms are:

- Virtual Lanes (VLs) provide dedicated receive buffer space for incoming packets at switch ports. VLs are also used for avoiding routing deadlocks. The Intel Omni-Path architecture supports up to 32 VLs.
- Service Channels (SCs) differentiate packets from different Service Levels. The SC is the only QoS identifier stored in the packet header. Each SC is mapped to a single VL, but a VL can be shared by multiple SCs. SCs are used for avoiding topology deadlocks and avoiding head-of-line blocking between different traffic classes. The Intel Omni-Path architecture supports up to 32 Service Channels, however SC15 is dedicated to in-band fabric management.
- Service Levels (SLs) are a group of SCs. An SL may span multiple SCs, but an SC is only assigned to one SL. SLs are used for separating high priority packets from lower priority packets belonging to the same application or Transport Layer, avoiding protocol deadlocks, etc. The Intel Omni-Path architecture supports up to 32 SLs.

- Traffic Classes (TCs) represent a group of SLs aimed to distinguish applications' traffic. A TC may span multiples SLs, but each SL is only assigned to one TC. The Intel Omni-Path architecture supports up to 32 TCs.
- A vFabric is a set of ports and one or more application protocols. For each vFabric, a set of QoS policies are applied. A given vFabric is associated with a TC for QoS and associated with a partition for security.

In contrast to IB, this architecture requires three mapping tables, an SL-to-SC table (SL2SC), an SC-to-VL table (SC2VL) and an SC-to-SL table (SC2SL) per network device. Packets may change of VL and SC on their route through the network, however, they cannot change of SL or TC. This fact, added to the fact that the SC identifier is the only QoS identifier stored in the packets, makes necessary to implement the SC2SL and SL2SC tables. Note that is is necessary to know the SL associated to a given SC, and vice-versa. The SC2VL table is also required to be able to store packets in the proper VL buffer. Note that NICs also require the SL2SC and SC2VL tables to assign the SC identifier and deliver packets to the proper VL.

Figure 2 shows a generic diagram of our OPA-based simulation model, where SC2SL, SL2SC and SC2VL tables are connected to the routing units and the output schedulers. Before packets can be moved from input buffers to output buffers, a mechanism is required to decide if a packet needs to change the SC. In our simulation model the SC change is performed when the packet is routed, avoiding deadlocks and/or distributing the traffic among the SCs belonging to the same SL.

OPA also includes QoS mechanisms such as *VLArbitration algorithm* and *preemption Tables*. However, there is not much information about how these mechanisms work. In our simulation model we have implemented DTable as the output scheduler as we will detail in Section 4. Note that, as far as we know, some elements of the OPA architecture are not detailed in the available public information [40, 21]. Given that, some assumptions have been done.

4. The Deficit Table scheduling mechanism

The main goal of an scheduling algorithm is to determine when packets from different SLs are delivered to satisfy the specified end-to-end latency and bandwidth requirements. Moreover, as stated in Section 1, in the context of HPC interconnection networks, output scheduling algorithms must meet two main properties: low computational complexity (the scheduler latency should be as low as possible) and low implementation complexity (the scheduler algorithm is typically implemented in hardware and a high implementation complexity implies a large silicon area). The complexity and silicon area of DTable compared with output schedulers such as Deficit Round-Robin (DRR) or SCFQ have been studied in [43]. This work has proven that in a scenario of high traffic load ($> 80\%$) and if VLs are used, DTable and SCFQ provide better packet latency [44] than DRR. It also concludes that the silicon area of DTable scheduler is twice the area of the DRR scheduler, while the silicon area of SCFQ is 4.5 times bigger. In addition, the IB architecture specification release 1.4, describes in Section 7.6.10 an enhanced QoS arbiter using an DTable-like mechanism [19, 20]. For these reasons, we have decided that DTable is a good option to perform this comparative study.

The DTable scheduler is a table-based output scheduling algorithm [31]. It uses an arbitration table with an arbitrary number of table entries, e.g. 32, 64, 128, etc. Each table entry has two fields: an SL identification number and an entry weight. The entry weight determines the maximum amount of information, measured in flow control credits, to be transmitted by a given SL each time that the entry is selected. When scheduling is required, the table is cycled through until a table entry from an active SL is found. An SL is considered active when it stores at least one packet and the flow control allows that SL to transmit packets, i.e. the buffer of the associated VL has enough flow control credits. Then, the DTable scheduler is composed of:

- Scheduling table: As stated before, it is a set of table entries, each one having an SL identifier and an associated weight.
- Deficit counters: It represents the weight that the scheduler owns to the SL. Every SL and output port have a counter. It means that there are $NumSLs \times OutputPorts$ deficit counters per switch. Each deficit counter is initialized to 0 at the system start-up.
- Accumulated weight counter: It is equal to the sum of the selected entry weight and the SL deficit counter. There is one accumulated weight counter per output port.

The selected SL may deliver as many packets as the accumulated weight allows. When a packet is transmitted, the accumulated weight is reduced by one packet size measured in flow control credits.

The next active table entry (i.e. a table entry associated to an active SL) is selected when the accumulated weight is smaller than the packet size. For instance, let us suppose an input/output port with 2 SLs and one buffer per

SL. The SL 0 buffer stores 3 packets and requires 2 flow control credits per packet. Hence, 2 weight units are required to deliver a packet. Figure 3 shows this example. The scheduler selects the table entry SL 0, being its weight of 3 units and its deficit counter of 0. Given that, the accumulated weight counter is equal to 3 units (*Round N Before delivery* in the figure). The SL 0 transmits 1 packet and the accumulated weight counter is reduced in 2 units. The final accumulated weight counter is 1 unit (*Round N After delivery*). Since the minimum weight to send a packet is 2 units, the SL 0 cannot send packets until the next arbitration round. The remaining accumulated weight (1 unit) is stored in the SL 0 deficit counter and the scheduler selects the next active table entry. Since the SL 1 has no packets, the SL 0 table entry is selected again. Adding its associated weight and its deficit counter, the SL 0 has an accumulated weight of 4 units (*Round N+1 Before delivery*), and it delivers its 2 remaining packets (*Round N+1 After delivery*).

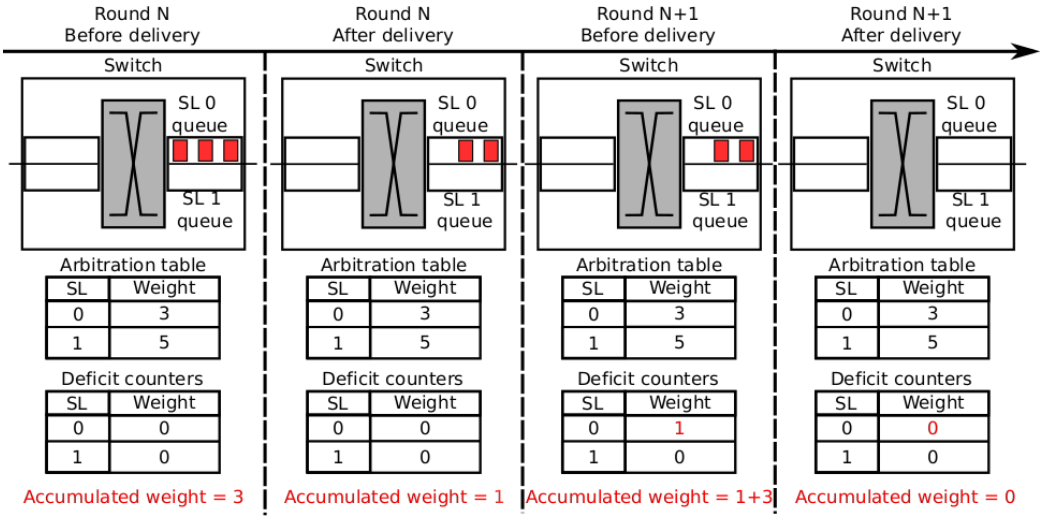


Figure 3: Example of DTable behavior.

If a given SL becomes inactive in a given output port, the SL deficit

counter is set to zero, the accumulated weight is discarded and the scheduler goes through the table to find the next active entry. In short, there are two possible scenarios: the accumulated weight becomes smaller than the packet size (it is stored in the deficit counter) or the SL becomes inactive (both counters are set to zero).

The bandwidth ϕ_i assigned to the SL_i in an N-entry arbitration table is

$$\phi_i = \frac{\sum_{j=0}^{J-1} weight_j}{\sum_{k=0}^{N-1} weight_k},$$

where J is the set of table entries assigned to SL_i and $weight$ is the entry weight assigned to a table entry. The number of table entries and the maximum distance between any pair of consecutive table entries assigned to the same SL allow to control the SL latency [16]. Let us suppose a system with a table-based output scheduling algorithm with 64 table entries. The SL 0 has 32 table entries with a distance of 2 between any pair of consecutive table entries. The SL 1 has 16 table entries with a distance of 4 entries. The weight of each SL 0 table entry is 2 units while the weight of each SL 1 entry is 4 units. Hence, $\phi_0 = \phi_1$, in other words, both SLs have been assigned the same bandwidth. However, SL 0 packets have lower end-to-end latency than SL 1 packets since the distance between table entries is lower for SL 0. The DTable configuration methodology can be found in Section 4.4 and at [31].

4.1. DTable implementation on IB-based interconnection network

In Section 3.1 we have detailed the QoS mechanisms of IB-based system and in Section 4 we have established the basic DTable behavior. This section details the implementation aspects of DTable in our IB-based simulation

model. The DTable scheduler requires a set of table entries, deficit counters and accumulated weight counters. IB-based architecture has a total of 128 table entries (64 high priority entries and 64 low priority entries) per output port and it supports up to 16 SLs.

In order to implement DTable in IB-based architectures, we have established that the high priority table entries will be the DTable set of table entries. Thereby, these table entries store the SL identifiers and the associated entry weights. It is required a deficit counter per SL, switch and output port. As stated above, IB provides an output scheduling table per output port with 64 entries. Therefore, the deficit counters have been implemented using the first 16 low priority table entries, as many entries as the maximum number of possible SLs in the system. An accumulated weight counter per output port is also required. The accumulated weight counter has been implemented using the last low priority table entry on each output port, in a similar way that we have stated for the accumulated weight counter. Note that IB table entries are 8-bit wide, and therefore, the accumulated weight could easily overflow. However, more unused adjacent entries could be used to implement the counter. In our IB-based simulation model we have used only one entry for the sake of simplicity. Figure 4 shows the arbitration table and counters implementation using the high and low priority table entries.

Using this technique to implement DTable on IB-based architectures, 47 table entries are wasted. Nevertheless, these entries could be used as part of the DTable scheduling table entries set, which gives us a total of 111 table entries. In this work we have left those 47 entries unused for the sake of simplicity. When scheduling is required, the high priority table entries will

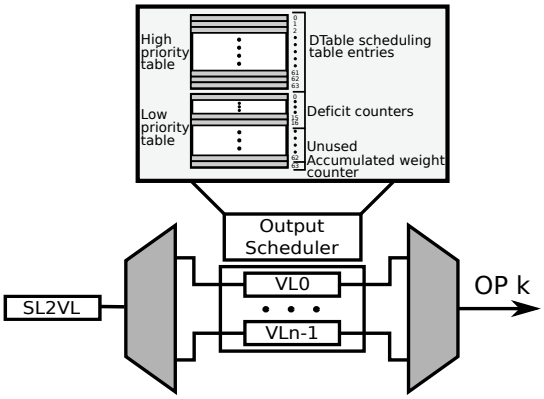


Figure 4: DTable scheduling algorithm implementation in an IB-based architecture.

be cycled through until a table entry is found.

4.2. DTable implementation on OPA-based interconnection network

Section 3.2 details the QoS mechanisms provided by OPA and Section 4 explains how the DTable output scheduling algorithm works. In this section we explain the DTable implementation details in OPA.

As mentioned in Section 3.2, there is no much details in the public available information about the output scheduler algorithm. Hence, we cannot take advantage of any existent table as we have done in IB (Section 4.1). In our OPA-based simulation model, he have implemented a scheduling table for each network device, a deficit counter per output port and SL and an accumulated weight counter per output port. OPA supports up to 32 SLs and the canonical switch has 48 ports, this means a total of 1536 deficit counters and 48 accumulated weight counters per network device.

OPA architecture defines the QoS level of the traffic flows through the SLs. Then, DTable must perform the arbitration process at SL level (i.e. arbitration entries store SL identifiers). As mentioned in Section 3.2, SCs and VLs are also involved in the QoS process, and therefore SL2SC, SC2SL and

SC2VL tables are also required. When scheduling is required, the scheduler selects the next table entry and reads the SL identifier of the entry. Using SL2SC and SC2VL tables, the scheduler determines if some VL associated to that SL stores packets and can transmit packets to the next network device (i.e. it has enough flow control credits). Finally, if these conditions are satisfied, the device delivers as many packets as the DTable scheduler allows.

4.3. The DTable scheduler and variable maximum transmission units

To fully exploit the DTable scheduler, variable maximum transmission units (MTUs) per SL are required. The MTU is the size of the biggest packet that may be delivered. A variable MTU per SL is necessary when using DTable scheduler for two main reasons:

1. In a system with fixed MTU, the deficit counter will never be used.
2. Having different packet sizes allows to decouple the bandwidth assignments from the latency requirements [31], which is very important.

Note that in the context of HPC interconnection networks, messages are composed of one or more packets, which are divided into one or more flits [45]. To that end, we have included in both architectures SL_MTU tables. These tables have as many entries as SLs in the system and only need one SL_MTU table per network device. MTU tables determine how many flits compose the packets per SL and also permit us to have different maximum packet sizes for different SLs, taking fully advantage of the DTable scheduler.

Nevertheless, neither IB nor OPA support variable packet sizes. To overtake this limitation, SL_MTU tables in our simulation model determine how many fixed size packets compose the messages, and the message generation

has been modified to always generate messages according to SL_MTU packet sizes. Moreover, before sending a message to the next network device, DTable has to ensure: i) the entire message fits onto the neighbor receiving buffer and ii) there is enough remaining weight for the selected SL.

Since the switching technique used is virtual-cut through and the mentioned delivery message system, all flits of the same message are stored, sent and received consecutively. Given that, VL buffers must have enough space (i.e. flow control credits) for storing at least the biggest MTU in the system.

4.4. DTable configuration methodology

In order to provide applications, flows, etc., specific QoS differences, DTable arbitration tables must be configured in a proper way. Note that DTable scheduling mechanisms themselves do not provide QoS without a proper configuration methodology [31].

Table 1: Arbitration table parameters.

$max\phi_i$,$min\phi_i$	Maximum/Minimum bandwidth assignable to the SL_i	MTU_i	Specific Maximum Transfer Unit of the SL_i
ϕ_i	Bandwidth assigned to the SL_i	M	Maximum weight per table entry
N	Number of entries of the arbitration table	$pool$	Bandwidth pool
n_i	Number of entries assigned to the SL_i	k	Bandwidth pool decoupling parameter
$GMTU$	General Maximum Transfer Unit	w	Maximum weight decoupling parameter

In an arbitration table configuration with N entries, a certain $GMTU$ value and n_i table entries for each SL (depending on their latency requirements), we would like to be able to assign the SL_i with a certain bandwidth ϕ_i in a flexible way. In other words, this means to keep the minimum bandwidth $min\phi_i$ that can be assigned to the SL_i as small as possible and the maximum bandwidth $max\phi_i$ assignable to the SL_i as large as possible. Table 1 shows the definition of all the parameters of the configuration methodology.

The maximum total weight that can be divided among the table entries is $M \times N$. However, we are going to fix it to a lower value called *pool*, which is determined by the k parameter. Section 4.3 explains that we can assign an specific MTU value for each SL. Then, the bandwidth ϕ_i of the SL_i is

$$\phi_i = \frac{\sum_{j=0}^{J-1} weight_j}{pool},$$

where J is the set of table entries assigned to the SL_i and $weight$ is the weight assigned to the table entry. Therefore, $min\phi_i$ and $max\phi_i$ assignable bandwidth values to the SL_i are

$$min\phi_i = \frac{n_i \times MTU_i}{pool}, \qquad max\phi_i = \frac{n_i \times M}{pool}.$$

Let us define M and *pool* using the $GMTU$ parameter and the decoupling parameters w and k:

$$M = GMTU \times w, \qquad pool = N \times GMTU \times k.$$

where $k \leq w$ because the bandwidth pool has to be smaller than $N \times M$. Hence, the maximum and minimum bandwidth depend not only on the proportion of table entries n_i, but also on the w and k parameters and the proportion between their specific MTU_i and $GMTU$:

$$min\phi_i = \frac{n_i \times MTU_i}{N \times GMTU \times k}, \qquad max\phi_i = \frac{n_i \times GMTU \times w}{N \times GMTU \times k} = \frac{n_i \times w}{N \times k}.$$

Therefore, parameters w, k and the specific MTU_i assigned to each SL allow to vary the maximum and minimum bandwidth assignable to SLs without affecting the final latency [31].

5. Performance evaluation

In this section, we compare the QoS provision performance of IB and OPA switch architectures using the DTable scheduler. We have used the following metrics:

- End-to-end latency per SL: Average message latency of all the messages labeled with the same SL measured from generation to reception. This is the latency that users of each SL will experience.
- Throughput per SL: The total amount of delivered information per SL expressed in flits/cycle/NIC transmitted through the network.

We have used Hiperion, which implements both IB and OPA simulation models detailed in Sections 2.1 and 2.2, which mimic the architectural behavior of both switches, as well as the QoS mechanisms explained in Sections 3.1 and 3.2, the DTable scheduler adapted for each architecture as stated in Sections 4.1 and 4.2, and SL_MTU tables detailed in Section 4.3.

5.1. Traffic model

Table 2 shows the traffic types considered and if they have QoS requirement of they are best-effort traffic. There are seven types of traffic flows, five of them with explicit QoS requirements such as latency and bandwidth, and three of them are best-effort with slight different levels of priority. SLs in Table 2 are sorted from the most latency sensitive to the least latency sensitive SL. For instance, NC SL is the most latency sensitive, which means that it must achieve the lowest latency of all SLs. The best-effort SLs are the least sensitive SLs. In terms of the DTable configuration, all the explicit requirements for each SL have been defined, except the best-effort flows, which have been assigned with the furthest arbitration table distances and the remaining bandwidth percentages.

Packets from each SL have been simulated using the most appropriated traffic pattern to emulate its behavior:

Table 2: Set of SLs considered in the evaluation.

Type	**SL**	**Description**	**Traffic pattern**	**Message size**
QoS	Network Control (NC)	Supports the network infrastructure	Random uniform	192B
QoS	Voice (VO)	Audio and online videogames backend traffic	CBR connections	128B
QoS	Video (VI)	Video streaming traffic	CBR connections	2048B
QoS	Controlled load (CL)	High performance computing traffic	CBR connections	2048B
Best-effort	Excellent-effort (EE)	Preferential best-effort traffic.	Bursts4	1024B
Best-effort	Best-effort (BE)	Backup protocols, email system, etc.	Bursts4	1024B
Best-effort	Background (BK)	Rest of applications and services	Bursts4	1024B

- Network Control (NC) traffic is generated using a random uniform traffic distribution. In an HPC network we can expect fabric managers deliver to all network devices control packets in an uniform way. We have chosen a payload of 192 bytes in order to represent the worst case scenario. We have assumed that this SL will use nearly 1% of the total network bandwidth.
- Voice (VO) traffic is generated using a Constant Bit Rate (CBR) distribution. VO is composed of multiple point-to-point connections. We have defined that these connections will be selected at the simulation beginning and they will be up until the simulation ends. In [46], several payload values for voice codecs algorithms are shown. These values range from 20 to 160 bytes. We have selected a payload of 128 for being the closest one with a packet size of 64 bytes due to the limitations explained in Section 4.3. We have assumed that this SL will use nearly 2% of the total network bandwidth.
- Video (VI) traffic is generated using a CBR distribution. VI traffic is generated in the same way as in the VO traffic. According with [47], payload values ranging from 100 bytes to 64 kilobytes are feasible. We

have chosen a payload value of 2048 bytes. We have assumed that this SL will use nearly 30% of the total network bandwidth.

- Controlled Load (CL) traffic is generated using a CBR distribution with payload of 2048 bytes, representing a possible average payload of many HPC application communications. We have assumed that this SL will use nearly 35% of the total network bandwidth.
- Best-effort traffic: Excellent-effort (EE), Best-effort (BE) and Background (BK), are generated using a Bursts4 distribution. This traffic is composed of bursts of 4 messages generated at the same time heading to the same destination. Note that each message is composed of multiple packets according with SL_MTU tables. The packet payload of these SLs is 1028 bytes. We have assumed that those SLs will use nearly 12% of the total network bandwidth.

For all cases, except VI and VO traffic patterns, the destination distribution is uniform in order to fully load the network. Note that we have chosen a heterogeneous scenario where multiple traffics are mixed using the above mentioned bandwidth percentages. However, our proposal is aimed to any environment where flows with different QoS requirements coexist. Nevertheless, the multimedia environment is the most straightforward one.

5.2. Network topology

We have chosen the k-ary n-tree and nD Torus interconnection topologies for being very common and well known solutions in high performance environments. We have used these topologies with two different layouts: 4-ary 3-tree, 24-ary 2-tree, 8x8 2D Torus and 8x8x8 3D Torus. The configuration on each layout is the following:

- The 4-ary 3-tree configuration has 64 end points connected or NICs. Each switch, either IB or OPA, has 8 ports.
- The 24-ary 2-tree has been configured with 576 NICs and switches with 48 ports.
- The 2D Torus configuration has 512 end points and 64 switches with 48 ports: the ports have been aggregated to create trunk links of 10x ($4 \times 10 = 40$ ports) and there are 8 NICs attached by one link per switch. Note that the trunk links are used to increase the bandwidth in each torus direction, but the ports belonging to the same trunk link works independently, i.e. the ports transmit different packets traveling in the same torus direction.
- The 3D Torus configuration has 2048 NICs and 512 switches with 28 ports: 6 trunk links of 4x and 4 NICs connected by one link.

The k-ary n-tree topology implements the valiant routing algorithm for being able to mitigate congestion and Head-on-Line contention [48]. The nD torus implements DOR for being the most common in those topologies [49]. The SL2SC and SC2VL tables configuration used in the OPA architecture scenario are shown in Table 3. In short, each SL has one SC associated and each SC has one VL assigned. For instance, the NC SL has the SC 0 and the VL 0 associated. SLs description can be found in Section 5.1. The SL2VL table configuration used in the IB architecture scenario is shown in Table 3. As in the OPA scenario, each SL has a single VL associated.

In both architectures, switches implement a credit-based flow control protocol. Given that, packets are not dropped when congestion appears and they are only transmitted if there is enough space in reception buffers. Flows from

Table 3: Configuration of IB SL2VL, OPA SL2SC and OPA SC2VL tables.

IB SL2VL	
SL	*VL*
NC	0
VO	1
VI	2
CL	3
EE	4
BE	5
BK	6

OPA SL2SC	
SL	*SC*
NC	0
VO	1
VI	2
CL	3
EE	4
BE	5
BK	6

OPA SC2VL	
SC	*VL*
0	0
1	1
2	2
3	3
4	4
5	5
6	6

different applications and similar characteristics are aggregated via SLs. The packet scheduling is performed through SLs and flow control via VLs. For the sake of comparison, we have established the flit size in 64 bytes, the packet size in 1 flit and the flow control credit unit in 64 bytes. Therefore, the final packet size is 64 bytes and each packet is 1 flow control credit sized. We have stated the GMTU in both architectures to 32 (i.e. 2048 bytes). Nevertheless, the evaluation may be performed with smaller or larger MTUs.

As stated, we have used in both cases an input, output and central (in the case of OPA) buffer queuing architecture. The same buffer capacities have been used in both architectures. The buffer capacity is 114,688 bytes at input and output ports of switches and 229,376 bytes at the network interfaces. In OPA, the central crossbar buffer capacity is 229,376 bytes for every 4 ports (MPort), e.g. in an OPA-based switch with just 8 ports, the central buffer capacity is 458,752 bytes. This central crossbar buffer capacity is available only for each Mport (i.e. each MPort may consume in total 229,376 bytes). These buffer capacities have been adjusted experimentally in order to avoid flow contention before reaching the congestion point. Both architectures dynamically manage the VL buffer storage space, i.e. there are

no independent buffers per VL and the buffer space is divided according with the traffic requirements, ensuring a minimum and a maximum space per VL. The dynamic buffers provide more flexibility than static buffers [38]. Finally, if an application wants to inject a packet into an interface network queue but it is full, we assume that the packet is stored in the application layer queue.

5.3. Simulated scenario and scheduler configurations

We have assumed a scenario where the goal is to dedicate the following egress link percentages to the SLs shown in Table 2: 1% to NC SL, 2% to VO SL, 30% to VI SL, 35% to CL and the remaining percentage to be shared among the best-effort SLs. Regarding maximum latency requirements, as mentioned in Section 4 and in [16], maximum latency values can be handled through the maximum distance between any pair of table entries assigned to an SL. The bandwidth percentages are intended to represent, as close as possible, a realistic combination of traffic and applications with QoS requirements. Distances assigned to each SL are shown in Table 4, in the *Distance* column. As can be seen, we have stated that the maximum latencies, in ascending order are: NC, VO, VI, CL and best-effort SL. Note that distances do not define an absolute end-to-end maximum value but an order. For instance, the NC SL has lower latency requirements than the VO SL, the VO SL has a lower latency requirements than the VI SL and so on. This approach permits us to provide differentiated services to the applications.

The DTable configuration process requires a defined methodology. This methodology is explained in Section 4.4 and at [31]. It allows to decouple bandwidth assignments from latency assignments. As stated before, we have established the maximum distances: a maximum distance of 2 entries for the

Table 4: Application of the decoupling methodology and DTable bandwidth configuration.

	DTable decoupling metodology						Injection		DTable scheduler configuration			
SL	**Distance**	**#entr.**	**%entr**	MTU_i	$min\phi_i$	$max\phi_i$	**Min.**	**Max.**	ϕ_i	**#entr.**	**E.W.**	**T.W.**
NC	2	32	50	192	0.094	3	0.01	0.01	0.094	32	4-3	101
VO	4	16	25	128	0.047	1.5	0.016	0.016	0.164	16	11	176
VI	8	8	12.5	2048	0.25	0.75	0.23	0.23	0.3	8	40-41	322
CL	16	4	6.25	2048	0.125	0.375	0.28	0.28	0.35	4	93-94	375
EE	32	2	3.125	1024	0.0313	0.188	0.0125	0.1525	0.04	2	21-22	43
BE	64	1	1.563	1024	0.016	0.094	0.0125	0.1525	0.036	1	39	39
BK	64	1	1.563	1024	0.008	0.047	0.0125	0.1525	0.016	1	17	17
	Total	64	100		0.57	5.95	0.5735	0.9935	1	64		1073

N = 64, GMTU = 32, w = 3, k = 0.5

NC SL and a maximum distance of 64 entries to the best-effort SLs. Table 4 shows the total number of entries (#entr.) and the proportion of table entries (%entr.) for each SL. In order to achieve the maximum flexibility possible, MTU values have been established as small as possible. Table 4 shows the specified MTUs for each SL in the MTU_i column.

Finally, we have configured proper values for w and k parameters (Section 4.4). To that end, we have used a w and k adjustment methodology. This methodology is based in a multipurpose backtracking algorithm [50]. The main condition taken into account is we want for the CL SL a bandwidth higher than the actual proportion of table entries assigned. Moreover, we want to assign the NC SL, which has a high proportion of table entries assigned, a small proportion of bandwidth. Nevertheless, it is important to keep the k parameter value as small as possible in order to obtain good latency performance. For that reason, the backtracking algorithm always tries first to adjust the k parameter, starting from the lowest value possible 0.1 being $w = k$ and increasing k in steps of 0.1 until $min\phi_i$ is lower than the

desired ϕ_i for all SLs. Finally, the algorithm starting from $w = k$ increases w in steps of 0.1 until $max\phi_i$ is higher than the desired ϕ_i for all SLs. We have finally chosen a value of 3 for w and a value of 0.5 for k. This combination of values allows us to get an assignable bandwidth range [$min\phi_i$, $max\phi_i$] for each SL, which fits with the bandwidth requirements. Table 4 shows the minimum and maximum bandwidth that may be assigned to each SL using this configuration.

Table 4 shows in the column ϕ_i the final assigned bandwidth values. These bandwidth values have been selected taking into account the expected SL bandwidth usages stated in Section 5.1. Note that the high priority SLs have more reserved bandwidth than the corresponding injection rates in order to prevent flow congestion and HoL could affect these flows. Note also that these values are within the defined ranges. The column *injection* shows the total amount of traffic expressed in flits/cycle/NIC that each SL injects. In this scenario, SLs NC, VO, VI and CL inject a fixed amount of traffic and the best-effort SLs increase gradually the amount of traffic, *Min.* and *Max.* columns show these increases. The column ($T.W.$) shows the total weight distributed among the table entries of each SL and the weight associated to each table entry ($E.W$). Note that some SL entries may have been assigned different $E.W.$ values, e.g. some NC SL table entries have been assigned with a weight of 4 and others with 3. The reason behind these table weight deviations is the original DTable configuration methodology may produce slightly bandwidth deviations, i.e. an SL gets more or less bandwidth than desired. We have developed a system able to correct these deviations.

6. Simulation results

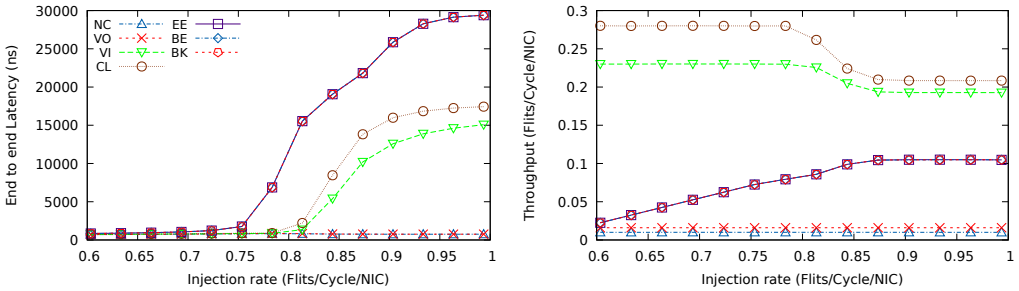
Performance metrics are described in Section 5 and the values shown for each injection rate are the average of 30 different simulations varying the seed of the random number generation.

Figures 5 and 6 show the end-to-end latency and throughput results using the k-ary n-tree and nD Torus topologies, respectively. Note that in some figures the BK SL is off the chart. We have decided to leave it outside for the sake of clarity, otherwise, the rest of lines would be too close to each other.

Total end-to-end latency using OPA switches is lower than using IB switches. For instance, the NC SL at the 0.7 injection rate is 780.41 on IB, 647.1 ns in OPA, 604.01 on IB, 446.87 ns in OPA for the 4-ary 3-tree and the 24-ary 2-tree topologies, respectively. In the same injection rate, the latency achieved by the NC SL in the nD Torus scenarios is 840.95 ns on IB, 758.1 ns in OPA, 1330.24 ns on IB and 1115.85 ns in OPA for the 2D and 3D Torus topologies, respectively.

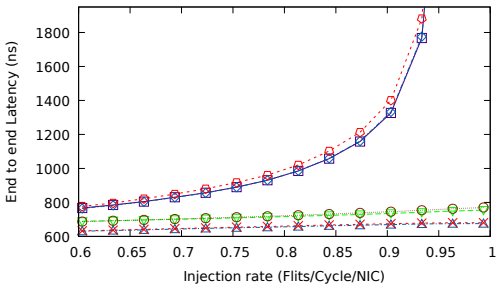
High priority SLs in OPA are less degraded than on IB. For instance, the NC SL latency difference in OPA between the first and the last injection rate shown in Figure 5c is 44.6 ns while on IB the latency difference between the first and last injection rate shown in Figure 5a for the same SL is 55.78 ns. Comparing latency increases with Figures 5e and 5g, the differences are 35.95 and 144.95 ns for OPA and IB, respectively. Regarding the nD Torus topologies, the NC SL latency difference in OPA shown in Figures 6a and 6c is 69.45 ns compared with 267.66 ns on IB. Differences in Figures 6e and 6g are 229.89 and 80.36 ns for IB and OPA, respectively.

In the OPA scenario, in terms of latency, point-to-point connections (VO,

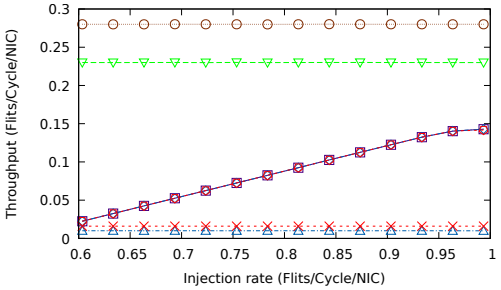


(a) IB

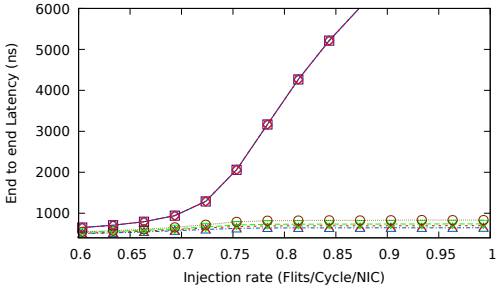
(b) IB



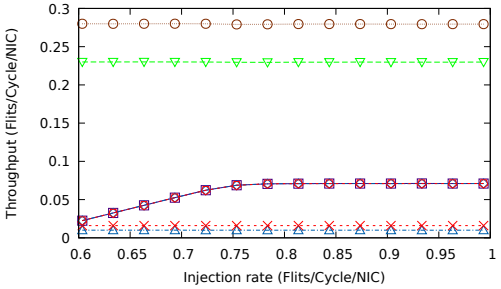
(c) OPA



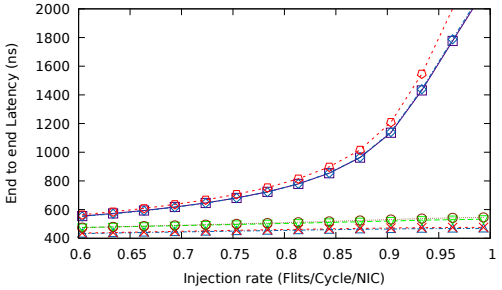
(d) OPA



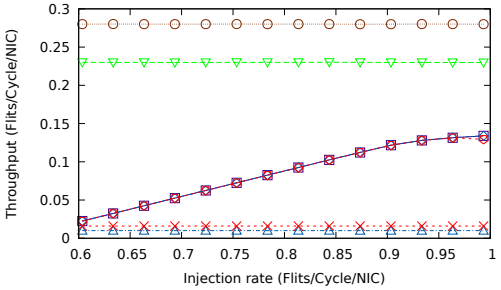
(e) IB



(f) IB

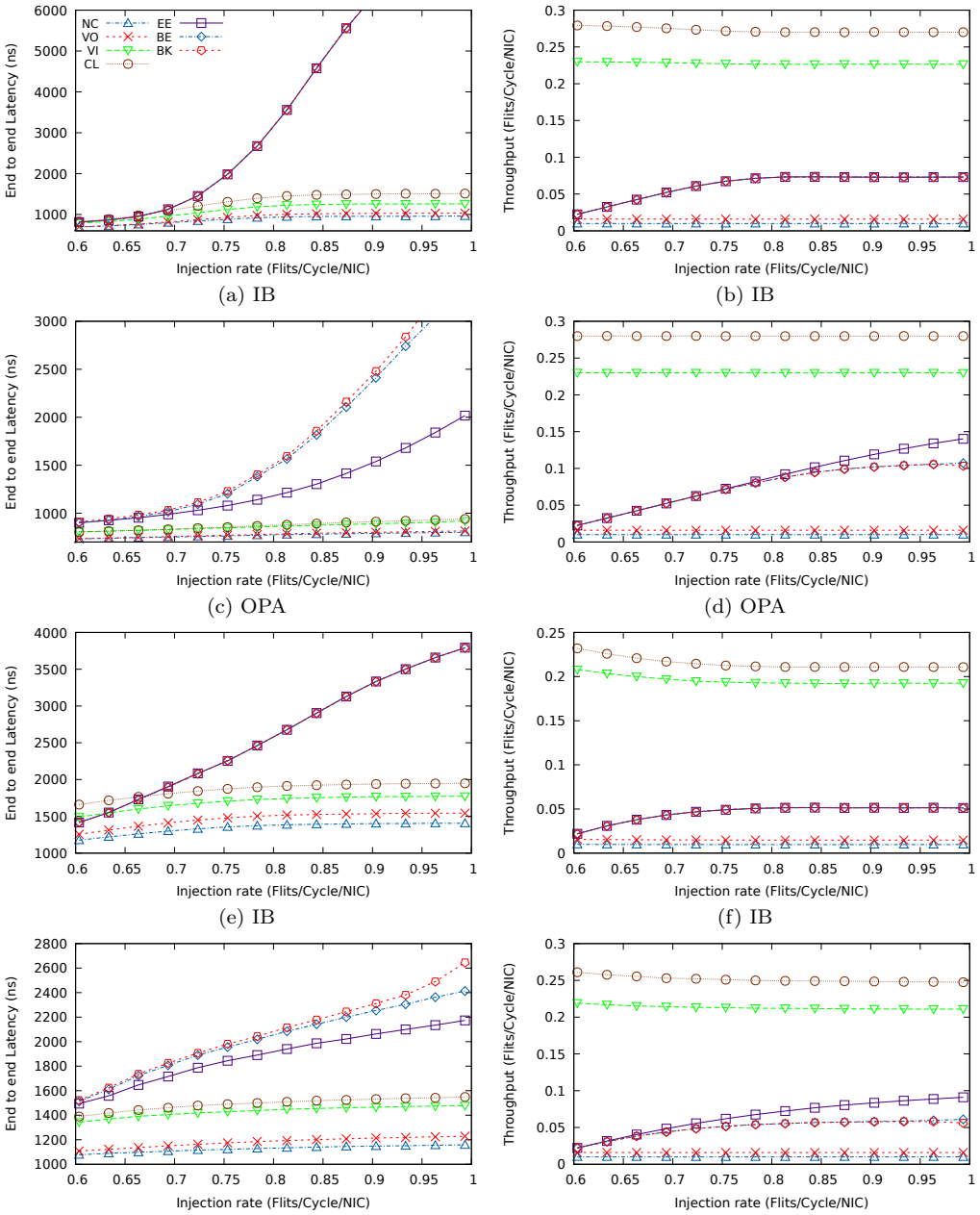


(g) OPA



(h) OPA

Figure 5: Performance of each SL using a 4-ary 3-tree (5a, 5b, 5c and 5d) and 24-ary 2-tree (5e, 5f, 5g and 5h) topologies with IB switch (5a, 5b, 5e and 5f) and OPA switch (5c, 5d, 5g and 5h). Results in Figures 5a, 5c, 5e and 5g refer to end-to-end latency and results in Figures 5b, 5d, 5f and 5h refer to throughput.



(a) IB (b) IB

(c) OPA (d) OPA

(e) IB (f) IB

(g) OPA (h) OPA

Figure 6: Performance of each SL using a 2D Torus (6a, 6b, 6c and 6d) and 3D Torus (6e, 6f, 6g and 6h) topologies with IB switch (6a, 6b, 6e and 6f) and OPA switch (6c, 6d, 6g and 6h). Results in Figures 6a, 6c, 6e and 6g refer to end-to-end latency and results in Figures 6b, 6d, 6f and 6h refer to throughput.

VI and CL SLs) are also less penalized, while on IB the point-to-point connections are more penalized than the low priority SLs. Moreover, as can be seen in Figure 5a, VO and VI latencies increase drastically when the network congestion point is reached. However, on IB the best-effort SLs increase their end-to-end latency more quickly than in OPA, when the injection ratio increases. This means that in order to meet the latency requirements on the high priority SLs, IB switches are forced to increase the low priority latencies. Whilst, this behavior can also be seen in OPA, although it is more widespread. This means OPA switches are able to provide better latency values than IB switches. Note that in Figures 5a, 5e, 5c and 5g BK SL latencies suffer from a large increase, the reason behind this fact is because the BK SL is generating packets at a higher rate than packets are injected, therefore, they are accumulated in the injection queues increasing drastically their final latency.

Referring throughput results shown in Figures 5b, 5d, 5f, 5h, 6b, 6d, 6f and 6h, achieved by each SL on each configuration. Throughput results are very close to the desired ones, even after the saturation point. The BK SL suffers of a slight throughput reduction because the BK SL is trying to use more bandwidth than the scheduler allows. Hence, the final throughput for the BK SL is slightly reduced. In both architectures, results show the scheduler using the proposed configuration is almost able to meet the bandwidth requirements in all cases. However, results of IB in Figure 5b, show the VI and VO throughput is reduced behind of the expected. Also, in the results shown by Figures 6f and 6h, the achieved throughput by the VO an VI SLs, is a bit lower than the expected. Note that in the case of OPA, the CL SL

is the only one affected. In our opinion, this is due to the fact that the IB pipeline is shorter than the OPA pipeline which encourages point-to-point SLs to suffer from increased congestion and HoL. Generally, in extreme cases, e.g. when an SL is heavily saturated, OPA is able to provide better QoS provision than IB. As commented above, in all proposed scenarios, OPA achieve better result than IB. The reason for this is that the OPA hierarchical crossbar architecture makes it possible to achieve higher throughput ratios and to maintain lower end-to-end latency values than IB, although in OPA packet could takes more time to cross the switch than on IB because some packets have to go through the central crossbar. Therefore, it seems more interesting to have high-radix switches even using more complex design to improve the achieved throughput than a more simple design providing lower latencies to all packets that cross the switch.

7. Conclusions

QoS provision is a relevant aspect of HPC interconnection networks. A key component for networks with QoS support is the scheduling algorithm, which is in charge of determining when the next packet should be transmitted. An ideal scheduling algorithm implemented in HPC networks should satisfy two main properties: good end-to-end latency and simplicity.

In this paper we have designed an IB-based and an OPA-based simulation models. They are two of the dominating technologies in the list TOP500 of the most powerful computers. Both architectures are also an example of non-hierarchical and hierarchical switch architectures, respectively. We have adapted to these architectures the DTable output scheduler which offers a

good balance between performance and hardware cost

We have evaluated the performance of the DTable scheduling algorithm in both architectures using an heterogeneous scenario where multiple traffic types coexist. We have carried out different experiments using several topology configurations and we have compared results against both architectures. End-to-end latency per SL results show that the OPA architecture is able to meet latency requirements better than the IB architecture, it is able to achieve lower latency values than IB. Also, high priority and point-to-point SLs in OPA are less degraded than on IB. Moreover, IB is forced earlier to drastically increase the latency of low priority SLs in order to keep the latency of high priority SLs. Throughput per SL results show that either IB or OPA can provide bandwidth desired values. However, on IB, in some scenarios, point-to-point SLs achieve throughput results a bit lower than expected.

Currently, we are developing a strategy to reduce the congestion, specially on IB for being the architecture most affected. We are also adapting sorted-priority scheduling algorithms.

Acknowledgment

This work has been supported by the Junta de Comunidades de Castilla-La Mancha, European Commission (FEDER funds) and Ministerio de Ciencia, Innovación y Universidades under projects SBPLY/17/180501/000498 and RTI2018-098156-B-C52 respectively. It is also co-financed by the University of Castilla-La Mancha and Fondo Europeo de Desarrollo Regional funds under project 2019-GRIN-27060. Javier Cano-Cano is also funded by the MINECO under FPI grant BES-2016-078800.

References

[1] H. T. Co., Huawei cloudcampus switch qos, Huawei Technologies Co. White Paper (2019).

[2] P. L. Montessoro, D. Pierattoni, Advanced research issues for tomorrow's multimedia networks, in: Proceedings International Conference on Information Technology: Coding and Computing, IEEE, 2001, pp. 336–340.

[3] B. Nandy, N. Seddigh, P. Pieda, J. Ethridge, Intelligent traffic conditioners for assured forwarding based differentiated services networks, in: Networking 2000 Broadband Communications, High Performance Networking, and Performance of Communication Networks, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 540–554.

[4] K. Kilkki, Differentiated services for the Internet, Macmillan Publishing Co., Inc., 1999.

[5] P. Ferguson, G. Huston, Quality of service: delivering QoS on the Internet and in corporate networks, John Wiley & Sons, Inc., 1998.

[6] D. Chalmers, M. Sloman, A survey of quality of service in mobile computing environments, IEEE Communications surveys 2 (2) (1999) 2–10.

[7] A. Souza, K. Pelckmans, J. Tordsson, A hpc co-scheduler with reinforcement learning (2020).

[8] L. Savoie, D. K. Lowenthal, B. R. De Supinski, K. Mohror, N. Jain, Mitigating inter-job interference via process-level quality-of-service, in: 2019 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2019, pp. 1–5.

[9] L. Savoie, Inter-job optimization in high performance computing (2019).

[10] A. Demers, S. Keshav, S. Shenker, Analysis and simulation of a fair queueing algorithm, ACM SIGCOMM Computer Communication Review 19 (4) (1989) 1–12.

[11] A. G. Greenberg, N. Madras, How fair is fair queuing, Journal of the ACM (JACM) 39 (3) (1992) 568–598.

[12] S. J. Golestani, A self-clocked fair queueing scheme for broadband applications, in: Proceedings of INFOCOM'94 Conference on Computer Communications, IEEE, 1994, pp. 636–646.

[13] J. W. Lee, M. C. Ng, K. Asanovic, Globally-synchronized frames for guaranteed quality-of-service in on-chip networks, ACM SIGARCH Computer Architecture News 36 (3) (2008) 89–100.

[14] B. Grot, S. W. Keckler, O. Mutlu, Preemptive virtual clock: a flexible, efficient, and cost-effective qos scheme for networks-on-chip, in: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009, pp. 268–279.

[15] H. Zhang, Service disciplines for guaranteed performance service in packet-switching networks, Proceedings of the IEEE 83 (10) (1995) 1374–1396.

[16] F. J. Alfaro, J. L. Sánchez, J. Duato, Qos in InfiniBand subnetworks, IEEE Transactions on Parallel and Distributed Systems 15 (9) (2004) 810–823.

[17] B. C. Lincoln, Scheduler utilizing dynamic schedule table, uS Patent 5,889,779 (Mar. 30 1999).

[18] D. B. Kramer, D. P. Sonnier, L. Zsohar, Processor with dynamic table-based scheduling using multi-entry table locations for handling transmission request collisions, uS Patent 7,224,681 (May 29 2007).

[19] D. Crupnicoff, S. Das, E. Zahavi, Deploying quality of service and congestion control in infiniband-based data center networks, Mellanox Technologies (2005).

[20] I. T. Association, et al., Infiniband architecture specification release 1.4, http://www.infinibandta.org (2020).

[21] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, R. C. Zak, Intel® Omni-Path Architecture: Enabling scalable, high performance fabrics, in: High-Performance Interconnects (HOTI), 2015 IEEE 23rd Annual Symposium on, IEEE, 2015, pp. 1–9.

[22] TOP500 homepage, `https://www.top500.org`, (Accessed June 12, 2020).

[23] J. Kim, W. J. Dally, B. Towles, A. K. Gupta, Microarchitecture of a high-radix router, in: ACM SIGARCH Computer Architecture News, Vol. 33, IEEE Computer Society, 2005, pp. 420–431.

[24] S. Scott, D. Abts, J. Kim, W. J. Dally, The blackwidow high-radix clos network, ACM SIGARCH Computer Architecture News 34 (2) (2006) 16–28.

[25] D. Abts, J. Kim, High performance datacenter networks: Architectures, algorithms, and opportunities, Synthesis Lectures on Computer Architecture 6 (1) (2011) 1–115.

[26] S. Scott, Rossetta: A 64-port switch for Cray's Slingshot Interconnect, in: Proceedings of the 26th Annual Symposium on High-Performance Interconnects (HOTI), 2019.

[27] G. Han, R. H. Klenke, J. H. Aylor, Performance modeling of hierarchical crossbar-based multicomputer systems, IEEE Transactions on Computers 50 (9) (2001) 877–890.

[28] EXTOLL homepage, `http://www.extoll.de`, (Accessed August 12, 2020).

[29] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, et al., The ibm blue gene/q compute chip, Ieee Micro 32 (2) (2011) 48–60.

[30] Y. Ajima, T. Kawashima, T. Okamoto, N. Shida, K. Hirai, T. Shimizu, S. Hiramoto, Y. Ikeda, T. Yoshikawa, K. Uchida, et al., The tofu interconnect d, in: 2018 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2018, pp. 646–654.

[31] R. Martinez-Morais, F. J. Alfaro-Cortes, J. L. Sanchez, Providing QoS

with the deficit table scheduler, IEEE Transactions on Parallel and Distributed Systems 21 (3) (2009) 327–341.

[32] N. Agarwal, T. Krishna, L.-S. Peh, N. K. Jha, Garnet: A detailed on-chip network model inside a full-system simulator, in: 2009 IEEE international symposium on performance analysis of systems and software, IEEE, 2009, pp. 33–42.

[33] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, W. J. Dally, A detailed and flexible cycle-accurate network-on-chip simulator, in: 2013 IEEE international symposium on performance analysis of systems and software (ISPASS), IEEE, 2013, pp. 86–96.

[34] S. Böhm, C. Engelmann, xsim: The extreme-scale simulator, in: 2011 International Conference on High Performance Computing & Simulation, IEEE, 2011, pp. 280–286.

[35] F. J. Andújar, S. Coll, M. Alonso, P. López, J.-M. Martínez, Powar: Power-aware routing in hpc networks with on/off links, ACM Transactions on Architecture and Code Optimization (TACO) 15 (4) (2019) 1–22.

[36] F. J. Andújar, J. A. Villar, J. L. Sánchez, F. J. Alfaro, J. Duato, Building 3d torus using low-profile expansion cards, IEEE Transactions on Computers 63 (11) (2013) 2701–2715.

[37] Hiperion repository homepage, `https://gitraap.i3a.info/fandujar/hiperion`, (Accessed October 23, 2020).

[38] Y. Tamir, G. L. Frazier, Dynamically-allocated multi-queue buffers for VLSI communication switches, IEEE Transactions on Computers 41 (6) (1992) 725–737.

[39] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, R. C. Zak, Enabling scalable high-performance systems with the Intel Omni-Path Architecture, IEEE Micro 36 (4) (2016) 38–47.

[40] J. Cano-Cano, G. Ferández-Martín, F. J. Andújar, J. L. Sánchez, F. J. Alfaro, G. Mora-Porta, OPASim: an OPA simulator for high-performance interconnections, Tech. rep., Campus Universitario s/n 02071 Albacete (2018).

[41] J. Cano-Cano, F. J. Andújar, G. Ferández-Martín, J. L. Sánchez, F. J. Alfaro, G. Mora-Porta, OPASim: an OPA simulator for high-performance interconnections networks, in: Proceedings of the 4nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems, ACM, 2019, pp. 1–4.

[42] G. F. Pfister, An introduction to the InfiniBand architecture, High Performance Mass Storage and Parallel I/O 42 (2001) 617–632.

[43] R. Martínez, J. M. Claver, F. J. Alfaro, J. L. Sánchez, Hardware implementation study of several new egress link scheduling algorithms, Journal of Parallel and Distributed Computing 72 (8) (2012) 975–989.

[44] R. Martínez, F. J. Alfaro, J. L. Sánchez, A framework to provide quality

of service over advanced switching, IEEE Transactions on Parallel and Distributed Systems 19 (8) (2008) 1111–1123.

[45] J. Duato, S. Yalamanchili, L. Ni, Interconnection networks, Morgan Kaufmann, 2003.

[46] A. Tyagi, J. K. Muppala, H. De Meer, VoIP support on differentiated services using expedited forwarding, in: Conference Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference (Cat. No. 00CH37086), IEEE, 2000, pp. 574–580.

[47] S. Wenger, H. 264/avc over IP, IEEE transactions on circuits and systems for video technology 13 (7) (2003) 645–656.

[48] L. G. Valiant, A scheme for fast parallel communication, SIAM journal on computing 11 (2) (1982) 350–361.

[49] W. J. Dally, B. P. Towles, Principles and practices of interconnection networks, Elsevier, 2004.

[50] H. A. Priestley, M. P. Ward, A multipurpose backtracking algorithm, Journal of Symbolic Computation 18 (1) (1994) 1–40.