The Journal of Supercomputing (2025)81:601
https://doi.org/10.1007/s11227-025-07069-1

# Quality-of-service provision for BXIv3-based interconnection networks

**Miguel Sánchez de la Rosa[1] · Gabriel Gomez-Lopez[1] · Francisco J. Andújar[2] · Jesús Escudero-Sahuquillo[1] · José L. Sánchez[1] · Francisco J. Alfaro-Cortés[1] · Pierre-Axel Lagadec[3]**

Accepted: 13 February 2025

## Abstract
Supercomputers (SCs) enable advanced research for a variety of scientific fields, and data centers (DCs) power our day-to-day services. These two massive systems work at scales, in terms of storage and computing power, which are not comparable to our everyday devices. As such, they require state-of-the-art technology to constantly evolve and meet our increasing demand. The interconnection network is the backbone of these systems, since it must provide efficient communication among the nodes that compose the whole system, otherwise becoming the entire system bottleneck. As multiple applications and services may use subsets of the system at the same time, interconnection networks must prevent excessive degradation for latency-sensitive applications. To this end, differentiated services are used to provide fair network access that considers bandwidth and latency requirements for each application. In this paper, we extend the switch architecture of next-generation BXI networks (hereafter called BXIv3) to incorporate arbitration tables so these networks can provide quality of service (QoS) to applications and services running on both SCs and DCs. Our proposal has been implemented in a network simulator, which models the behavior of a BXIv3 network. We have used several traffic patterns and arbitration table configurations to conduct a set of simulation experiments for the evaluation of our solution. The obtained results show that our proposal achieves accurate bandwidth allocation with differentiated latencies. Moreover, a study of memory requirements shows that our solution is quite feasible for hardware implementation.

**Keywords** Bull Exascale Interconnect (BXI) · High-performance ethernet · Networks · Quality of service (QoS) · Network requirements

Pierre-Axel Lagadec in ATOS while working on this paper.

Extended author information available on the last page of the article

Published online: 11 March 2025

## 1 Introduction

Supercomputers (SCs) and data centers (DCs) are nowadays fundamental infrastructures for the digital transformation of our personal and professional lives. They drive breakthroughs in medicine, climate research, and technology, accelerating innovation and solving complex problems beyond the capability of ordinary computers. Indeed, they are a fundamental pillar to enforce our ability to unravel mysteries, find cures, and engineer a more intelligent and healthier world. The interconnection network is a critical component of the computing systems at such scales, which must provide specific requisites such as high performance to the communication operations of thousands of nodes and reduced cost through much simpler designs while maintaining effectiveness.

In the past, interconnection networks in SCs and DCs have differed in multiple ways. Nonetheless, in recent years, their network architecture has converged [1] because the applications running in these systems have standard communication requirements. The trend nowadays is to design high-performance interconnection networks interchangeable between SCs and DCs. Indeed, the networks for these systems share critical network design aspects, such as network topology, routing algorithms, flow control, quality of service (QoS), power management, or congestion control. Although the research and innovations in the interconnection network have been pervasive to overcome these design aspects, the post-Exascale era and generative AI models pose new challenges to the interconnection network design. Therefore, interconnects need novel and disruptive approaches to deal with these tasks.

Over the last decades, computing power has increased so significantly that communication networks have not been able to improve at the same pace. The sheer increase in computing power has been brought by general-purpose computing on graphics processing units (GPUs) and, more recently, neural processing units (NPUs). Nodes may include multiple of these devices, requiring over-dimensioned intra-node communication speed for optimal performance. It has come to a point that technologies such as PCIe and NVLink communicate devices on several nodes forming a network alongside the traditional interconnect by using switches. However, regular non-GPU and non-NPU traffic are relevant for inter-node networks, and new challenges arise as our computing demands grow.

Another important aspect of designing high-performance interconnection networks relies on characterizing the communication requirements of applications and services run in SCs and DCs. This communication consists of various traffic flows (each composed of a set of packets), which servers inject through the network interfaces (NICs) into the network, where packets traverse a collection of switches on their way toward their destination. Note that several applications and services commonly used in DCs and SCs, such as cloud services in multi-tenant platforms, deep learning or generative IA training, online data-intensive (OLDI) services or inference, and distributed storage, may share the network resources. Therefore, the traffic flows from different applications and services may need specific provision of network resources to guarantee their latency and throughput demands.

QoS allows the differentiation of traffic flows from different applications according to their specific requirements (e.g., latency or throughput). Network administrators should configure the QoS provision a priori if this functionality is available, avoiding one or a few traffic flows consuming all the network resources; otherwise, users may experience poor system performance, even if the system is not overloaded. QoS is applied in multi-tenant and cloud systems to provide specific network performance to certain traffic flows. Indeed, QoS is gaining attention due to the explosion of cloud services.

Figure 1 shows the outcome of a Google Search routine (in blue) running on a system that runs on our network simulation model for BXIv3. The interconnection network is shared with one application (in purple) with a constant load and bursty traffic with a pattern that resembles the micro-bursts found in DCs (in red) [2]. We can see that every millisecond burst greatly degrades the efficiency of the Google Search pattern traffic and the total network efficiency as well. This is the result of an experiment we have carried out on our SAURON [3] simulator, which models the current state of BXIv3. More details on this scenario will be given in Sect. 4. Note that the most popular interconnection network technologies include support for providing QoS [4–6].

An example of an application that can be found in an HPC environment is GROMACS [7]. This application simulates molecular dynamics and can employ MPI for communication. Figure 2a shows the traffic generation throughout its execution time, with 240 MPI ranks spread across 10 nodes. Figure 2 shows the number of messages generated, with Fig. 2b displaying the corresponding traffic in bytes. We can see that network activity is not constant over time, and unused traffic can be used for other applications running on the system. Figure 2c and d shows the directionality of messages and bytes, respectively. Note that the matrix has been reduced to half its size for visibility reasons. We can see that the program employs multiple communicators to exploit network locality and/or intra-node

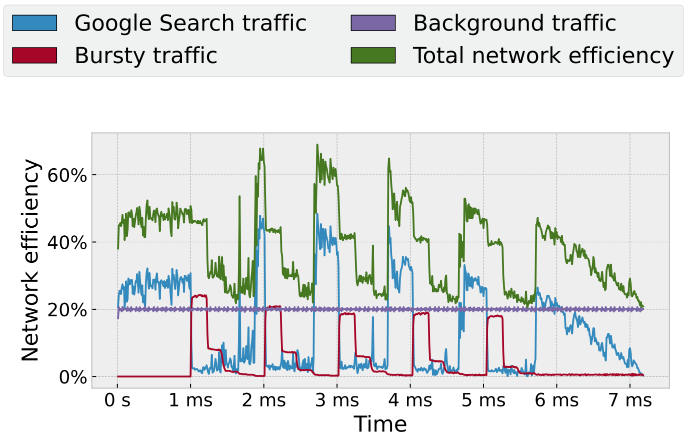
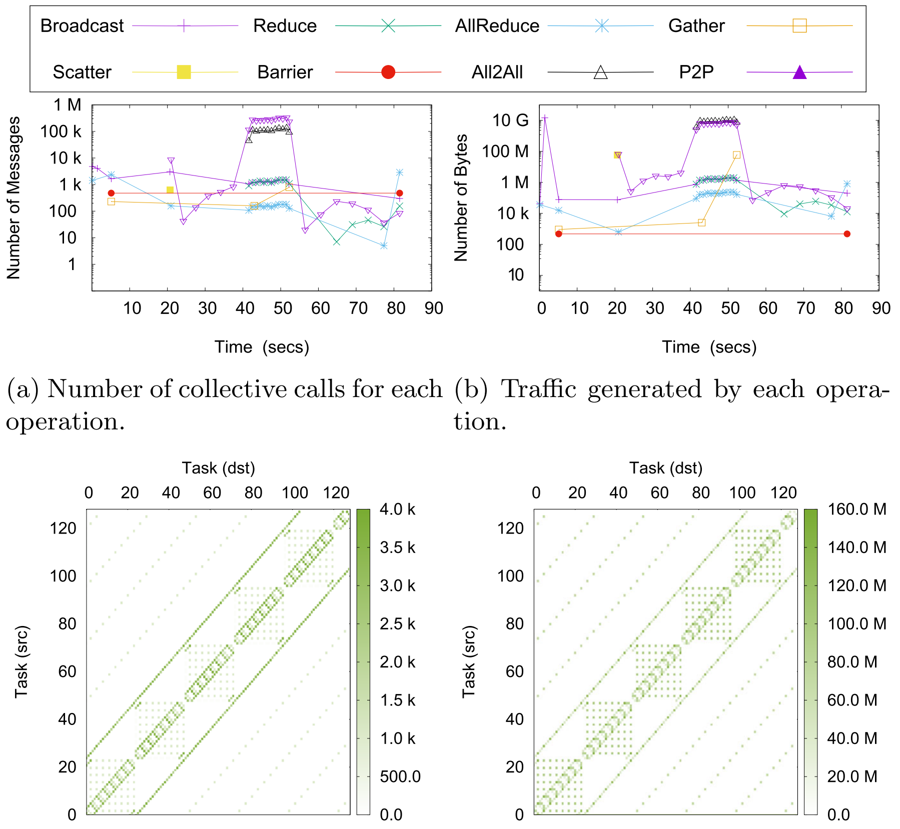


**Fig. 1** Network performance degradation of different applications due to traffic bursts in a network without QoS provision



(a) Number of collective calls for each operation.

(b) Traffic generated by each operation.

(c) Number of exchanged messages.

(d) Number of exchanged bytes.

**Fig. 2** Behavior of GROMACS in terms of messages and bytes

communication. Even so, network communication eventually occurs. If these message exchanges are degraded due to other flows, the application will remain idle waiting for messages to arrive.

Unfortunately, the most many network technologies (e.g., BXIv3) can do in this situation is to isolate each application in separate queues (or virtual channels), and schedule link transmission for each queue using round-robin (RR) arbitration. Still, RR does not guarantee equal bandwidth with variable packet sizes.

This is the problem statement that we aim to solve. Being unaware of any proposals for providing QoS to BXIv3 yet, the main contributions of this work are:

- Improve the arbitration tables previously proposed for a more accurate bandwidth and latency assignment.
- A proposal for using arbitration tables on BXIv3, providing bandwidth allocation as well as differentiated latencies for top-priority flows.
- Experiments with realistic traffic patterns, using the VEF-Traces framework [8].

The rest of the paper is organized as follows: Sect. 2 overviews the background of QoS provision in interconnection networks. Section 3 describes our proposal for QoS in BXIv3 networks. Section 4 shows the experiment results and analyses them. Finally, in Sect. 5, some conclusions are drawn.

## 2 Background

### 2.1 Interconnection networks

Interconnection networks are the backbone of SCs and DCs, ensuring the correct communication between elements such as computing and storage nodes. As mentioned above, interconnection networks need to be conveniently designed so they are not the bottleneck in the communication of applications and services. Among the critical design issues, we can find the network topology, the routing algorithm, or the switch architecture.

The topology of a network determines how devices are interconnected. State-of-the-art efficient topologies such as Parallel Port Generalized Fat Trees (PGFTs) [9] or Megafly [10] offer multiple and short paths between any pair of server nodes and are composed of fewer elements to save cost. Topologies are paired with routing algorithms to send packets with as few hops as possible or avoid the most congested paths to their destination, increasing throughput. The bare minimum for a routing algorithm is to establish a path from an origin to a destination that packets follow. However, more sophisticated algorithms can take advantage of multiple paths to diverge traffic or dynamically adapt packets based on the network status.

Figure 3 shows a simplified version of the PGFT topology using 64-port switches. In this topology, L1 switches employ half of their ports for hosts and the other half for connecting to L2 switches. L2 switches employ their uplink ports for connecting to other PGFTs, allowing the topology to scale up to 512 PGFTs, or up to 262,144 nodes. An example of an algorithm tailored to Fat Trees and their variants is D-Mod-K [11]. This routing algorithm provides minimal and static routing. It exploits the presence of multiple paths by using the packet destination to choose among those paths. This means when any node *a* sends a packet to other nodes *b* and *c*, each followed route differs, even if both packets went through the same hop count.

Switches use buffers to store packets before forwarding them to their destination. Figure 4 shows two common architectures used in HPC networks.

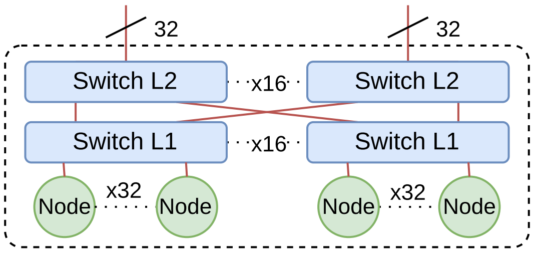
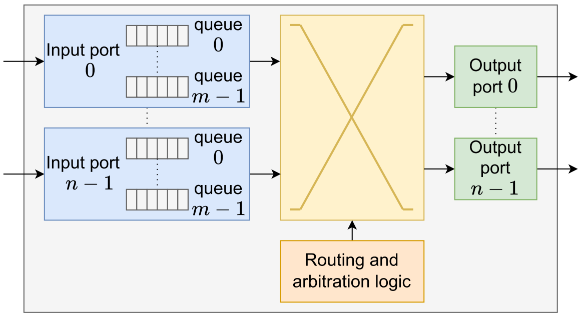
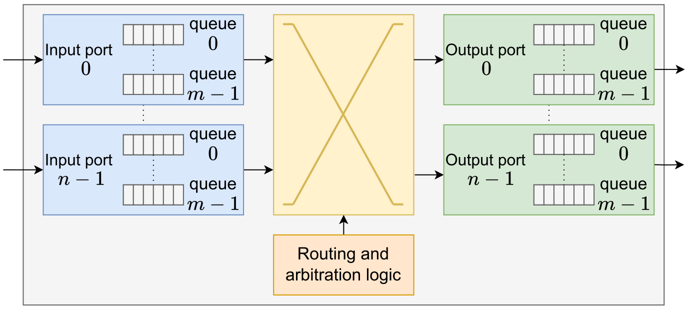


**Fig. 3** Two-stage PGFT topology with 64-port switches, connecting up to 512 nodes



(a) IQ architecture.



(b) CIOQ architecture.

**Fig. 4** Comparison of input-queued (IQ) and combined input–output-queued switch architectures. Switch radix is $n$, and queue count is $m$

In input-queued (IQ) architectures, as displayed in Fig. 4a, buffers only store incoming packets. Combined input- and output-queued (CIOQ) switches, on the other hand, do so for outbound packets as well, as shown in Fig. 4b. Buffers can be partitioned into separate queues, and queuing schemes assign packets to different queues to maximize link usage.

In the case of CIOQ switches, packets arrive at input queues and then request to be copied to a matching output buffer (based on the routing algorithm), and stay there waiting for transmission. On IQ switches, however, the packet requests to traverse the crossbar and is transmitted right away. For transmission, ports must choose a suitable packet among all the queues. These queues are often referred to as virtual channels (VCs) or virtual lanes (VLs). Ports search among the several VCs for available packets to send at any given time. VCs can be used for deadlock avoidance, isolation of congesting traffic, and queuing schemes aiming to improve throughput or traffic classification.

Another crucial aspect in the design of high-performance interconnection networks is the flow control, which must guarantee that packet discarding is not allowed, otherwise, the retransmission of dropped packets would introduce additional latency in the communication. Some network technologies, such as InfiniBand, achieve lossless communication thanks to a link-level flow control mechanism based on credits, avoiding packet dropping and retransmission. Ethernet, on the other hand, was not conceived as a lossless interconnection technology. Lossy communication may cause packet dropping and retransmissions, which is detrimental to network performance. PAUSE (MAC) frames were proposed in standard IEEE 802.3x to provide Ethernet with stop-and-go flow control. A port can send a PAUSE frame within a certain time to request the port at the other end of the link to inhibit transmission temporarily. However, this inhibition affects all the VCs, regardless of their particular occupation or priority. Thus, priority-based flow control (PFC) [12] was proposed to handle flow control of each queue separately. PFC employs PAUSE and UNPAUSE messages to request to halt or continue with transmission on certain queues, based on their occupation of input ports.

Inside the switch, arbitration or scheduling is needed when multiple flows require access to components like crossbars (when these flows head to be forwarded from the input ports to the output ones) or output ports (when flows stored at output port queues are forwarded to the next switch). Such arbitrations must ensure that access to shared elements is eventually granted to any request. However, arbiters can still favor some flows, increasing their bandwidth and/or reducing their latency. The prioritization of certain flows over others is the cornerstone of QoS policies. Network devices break ties when several packets are competing for transmission.

## 2.2 Quality of service

SC and DC applications may be highly sensitive to latency or bandwidth allocation or naive scheduling policies at network switches, risking performance decline. Therefore, network devices need to distinguish and arbitrate among their flows, based on each application's specific requirements, so that each traffic flow is scheduled at every which according to the requirements (or priority) of the applications these flows belong to. This approach is referred to as differentiated services [13].

To deal with these traffic flows, network hardware provides virtual channels/ virtual lanes (VCs/VLs) to logically divide the switch port buffers into independent queues, and packets can be stored into specific VCs based on service levels (SLs) defined by application requirements. Using this, arbitration among queues schedules the transmission of packets so that treatment is different based on their SL, thus providing applications with QoS. Avoiding the performance degradation of real-time or latency-sensitive applications in multi-tenant environments under heavy workloads is the end goal of QoS.

For example, Ethernet and InfiniBand (IB), the most predominant technologies in the TOP500 list [14] as of November 2024, have specifications for enabling differentiated services as part of IEEE 802.1p and InfiniBand Trade Association (IBTA), respectively. Next-generation BXI networks, being Ethernet-based, define virtual channels and output scheduling. So does IB, with their analogous counterparts, such as virtual lanes (VLs) and SL-to-VL mapping [15]. Intel's OmniPath (OPA) also has support for QoS [16].

Jokanovic et al. [17] proposed mapping applications to VLs to minimize inter-application contention. Then, arbiters schedule VL selection to apply the desired bandwidths. However, this proposal is targeted toward packets of fixed size. There have been proposals to provide QoS using arbitration tables on OmniPath [18] and IB [6]. Enhancements to those proposals such as deficit counters [19] have been carried out to improve bandwidth assignment accuracy. We previously proposed a mechanism to provide applications with QoS on BXIv3-based networks [20]. In this work we have built upon to create enhancements such as deficit counters, which we will explore in Sect. 3.3.

## 2.3 BXI-based networks

Bull Exascale Interconnect (BXI [21]) is an interconnection network tailored to HPC developed by Atos (now Eviden). The current versions V1.2 [22] and V2 [21] of BXI are already present in some TOP500 supercomputers such as CEA-HF and Tera-1000–2 [14].

The next-generation BXI (from now on, BXIv3 [23]) is an Ethernet-based technology under development, which is expected to be used in future European HPC systems. Several projects have recently performed efforts to develop this technology, such as the RED-SEA project [24]. Follow-up projects such as Net4EXA and UltraEthernet aim to continue with the development of post-Exascale era interconnects.

Considerable effort has been devoted to reducing latency by enabling direct access to processor cores, bypassing PCI Express (PCIe) to eliminate overheads. Providing a Remote Direct Memory Access (RDMA) engine for network offloading is also crucial. For lossless communication, PFC was chosen as a per-queue flow control. Tailored to HPC workloads, the software environment for the upcoming BXIv3 has been based on Portals [25], which provides communication primitives for offloading Message Passing Interface (MPI) and Partitioned Global Address Space (PGAS) communication from hosts. Network interfaces will feature a link bandwidth of 400 Gbps. The target endpoint count for this technology is 100k nodes, under topologies such as Fat Trees and Megafly/Dragonfly+ [10].

BXIv3 switches will comprise 64 ports, following a CIOQ architecture. The lack of output buffers in BXIv2 means that its QoS implementations are not directly portable to BXIv3. Moreover, BXIv3 will also introduce packets of variable size, possible length values below the maximum transfer unit (MTU), as opposed to BXIv2's fixed packet size. Consequently, traffic measurement shifts from packets to smaller units like bytes or flits. In the context of QoS, managing variable packet size impacts switch and NIC architecture. Note that shared cloud and multi-tenant systems on future SCs and DCs using BXIv3 will need QoS provisioning to function properly. At this development stage for next-gen BXI, we know of no other QoS provision proposals. In this work, we describe adding this feature to BXIv3 and its potential benefits.

## 3 Our proposal provisioning QoS on BXIv3

In this section, we elaborate on our proposal of arbitration tables to perform accurate bandwidth assignment to traffic flows of different applications and reduce latency on top-priority flows. We also describe the additional hardware requirements for our proposal.

### 3.1 Arbitration tables at output ports

The arbitration tables may be the main mechanism that current network technologies include to provide applications with QoS. Arbitration tables are structures present on network devices, at both NICs and switch ports. Based on the entry count and the weight for each entry, they define the priority during arbitration access to shared resources at any given time.

In the case of the BXIv3 network technology, we assume that any port in the network can transmit at most one flit at time, even if ports are configured with several output queues. The weight present for a given table entry in the table is used to calculate a turn length (or quantum) for the traffic flow associated to that entry. In our case, the quantum is measured in flits. We measure the quantum in flits because BXIv3 switches operate with variable-size packets. This is fundamental for ensuring that variations in packet size do not affect bandwidth allocation and latency. This is particularly important when packets from different flows with arbitrary sizes coexist.

We propose using arbitration tables to determine how much traffic can be sent from the queue associated with each table entry, constituting a service level (SL). As previously commented, the arbitration table is present in every switch and NIC, and the output ports use this table to select the packets to transmit. Network administrators may configure arbitration tables so that values are loaded once switches startup. During network operation, arbitration table values are read-only. It could be technically possible to recalculate table weights and values while executing tasks. However, this will happen unusually since table values may be derived from service level agreements (SLAs). A common example of this is tiered cloud services. For consistency among the whole network, arbitration table values of different devices should not differ. Indeed, table values will depend on the QoS policy adopted across the whole system, therefore they are not expected to change dynamically.

From now on, we will assume that one SL corresponds to a queue (also known as VC or VL, depending on the network technology) and vice versa. Note that linking several queues to the same SL is also possible using an SL-to-VL translation table. This data structure is present in network technologies such as InfiniBand. Only one SL-to-VL table per switch will be necessary, which would contain one entry per queue, indicating its SL. For any given SL, one of its queues can be selected using round-robin, preventing certain queues from overloading. However, for the sake of simplicity, we will consider a one-to-one mapping between queues and SLs.

**Table 1** An example of arbitration table with the weight assigned to each SL

| Index | SL | Weight | Flits ($k = 100$) | Bandwidth (%) |
|---|---|---|---|---|
| 0 | 0 | 5 | 500 | 50 |
| 1 | 1 | 3 | 300 | 30 |
| 2 | 2 | 2 | 200 | 20 |

Table 1 describes an example of how to conform an arbitration table. Each row of this table corresponds to an SL (the `Index` column). Each entry has a `Weight` associated with it, used to calculate the maximum number of flits to send until switching to the next entry. The `SL` and `Flits` columns contain the actual arbitration table values on switches and NICs, accessed by their indexes (0–2). This way, we can relate a SL to a table entry. The factor $k$ translates weights to the entry total quantum in flits. The bandwidth of every table entry will be the ratio between its weight and the sum of all weights. In the example of the table, the last column (`Bandwidth`) shows the percentage of bandwidth assigned to this SL.

## 3.2 A mechanism for bandwidth allocation

The arbitration mechanism selects which queue is the following to transmit information using the output link and how many bits it can send. A fair output port arbitration mechanism ensures that all queues with packets are eventually selected for transmission. Output ports need one register to store the currently selected table entry and another one for the number of remaining flits to be sent until exhausting its turn, often named *quantum* in literature. From now on, we will call them the *current entry* and the *remaining flits* registers, respectively. Note that the remaining flits registers are initialized according to the flits field shown in Table 1. The objective of our arbitration technique is to find one packet from the desired queue without exceeding its turn. Every time the quantum is to be exceeded or the queue pointed by the *current entry* register is empty, its turn is skipped.

Our mechanism is described in detail in the pseudocode of Algorithm 1.

**Algorithm 1** Algorithm for arbitration table usage

```
 1: for $e \leftarrow 0$ to numEntries do
 2:     notEmtpy $\leftarrow$ true if no OutQueues[current_entry.Queue].empty
 3:     noPFCStop $\leftarrow$ true if no PFC stop
 4:     queueActive $\leftarrow$ true if notEmtpy and noPFCStop
 5:     headPacket $\leftarrow$ first packet of OutQueues[current_entry.Queue]
 6:     enoughWeight $\leftarrow$ true if headPacket.flits $\leq$ remaming_flits
 7:     if queueActive and enoughWeight then              ▷ Substract flits from quantum
 8:         selectedQueue $\leftarrow$ current_entry
 9:         remaining_flits $\leftarrow$ remaining_flits - headPacket.flits
10:         if $remaming\_flits = 0$ then                     ▷ Check quantum depletion
11:             current_entry $\leftarrow$ (current_entry + 1) mod $n_{entries}$
12:             remaining_flits $\leftarrow$ weights[current_entry] $\times$ k
13:         end if
14:         return selectedQueue                                    ▷ Queue found
15:     end if
16:     current_entry $\leftarrow$ (current_entry + 1) mod $n_{entries}$
17:     remaining_flits $\leftarrow$ weights[current_entry] $\times$ k
18: end for
19: return -1                                          ▷ No valid queue was found
```

Initially, output ports set *current entry* to 0, and *remaining flits* to its corresponding total quantum. On subsequent arbitrations, the *current entry* value read will be the last entry with a successful arbitration. The *remaining flits* register will show the quantum after subtracting the number of flits of the previous sent packet.

First, the output queue pointed by the current table entry must satisfy two conditions: it must be active (non-empty and flow control, if present, allows packet transmission, see lines 2–4). The length (in flits) of the head packet of the queue must be lower or equal to the *remaining flits*, as shown in lines 5–6. As shown in line 7, every condition must be met for a successful arbitration. Otherwise, the entry is skipped regardless of the quantum left, as per lines 8–9. When a queue is selected, the length in flits of that packet is subtracted from the *remaining flits* counter, as line 9 shows. If the *remaining flits* register reaches 0 (this queue has completely exhausted its quantum) the *current entry* and *remaining flits* registers are prepared for the next arbitration (lines 11–12). Regarding the *remaining flits*, multiplying by *k* converts the queue weight to its maximum number of flits.

This routine is repeated for every queue until a packet can be sent or none is eligible. In the latter case, no packets are sent, and the arbitration routine returns an invalid queue index (see line 19). When a packet to be sent is found, the routine returns a valid queue index to select a packet, as shown in line 14. At most, we check once per queue for every arbitration, as described by the loop in the Algorithm. Note

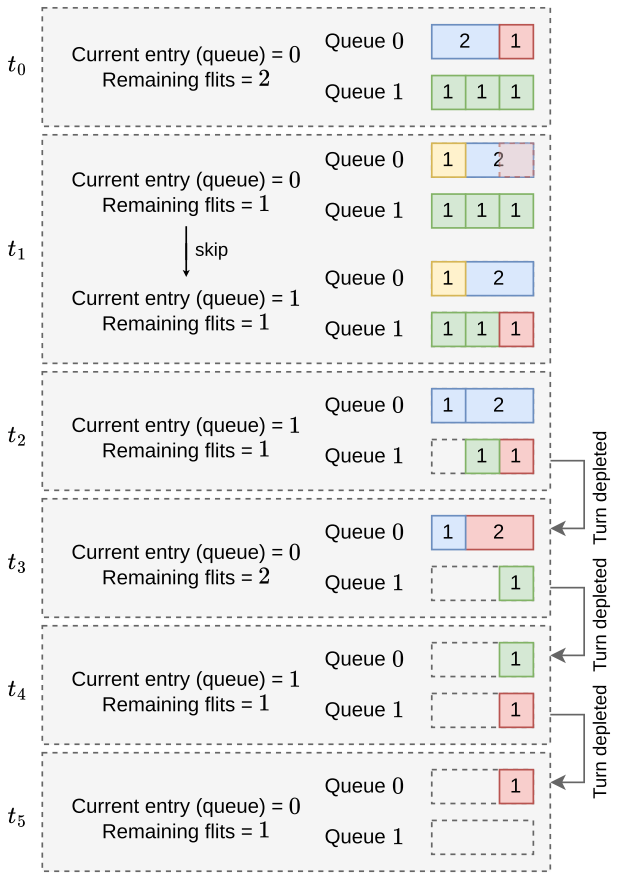


**Fig. 5** Graphical representation for our queue arbitration

that if all output queues are empty, no arbitration is required, and the *current entry* and *remaining flits* register values persist for the next arbitration.

Figure 5 shows an arbitration sequence using two entries.

Each entry corresponds to a queue, and packets are one and two flits long, for the sake of simplicity. Blue and green packets have been at output buffers, while red ones are selected at that round. Yellow packets have just arrived at the output queue at the current time. As seen at $t_1$, the 2-flits-long packet cannot be sent because its size exceeds the quantum for queue 0 (marked partially in red). Therefore, the queue is skipped, with one flit not sent, and a packet from queue 1 is sent instead. Even if a new packet arrives at $t_1$, it will have to wait until the one at the head of the queue is sent first. Arbitration will select the packet from queue 0 on $t_2$, depleting the turn. At $t_3$ the same happens to queue 1. At $t_4$ and $t_5$, packets from queue 1 and 0 will be selected, respectively.

### 3.3 Improving the mechanism with deficit counters

Due to variable packet sizes, arbiters can deny packet transmission if its length exceeds the remaining quantum. When this occurs, the entry is skipped, its quantum is discarded, and it loses part of its assigned bandwidth. For example, if an entry has a maximum turn of 1500 bytes but its queue holds 1000-byte packets, it could lose 33% of its effective bandwidth, discarding 500 bytes every two arbitrations. This potential bandwidth loss is present in our previous scheme, as explained before. Moreover, if we look back at Fig. 5, this is what occurs at $t_1$.

To overcome this flaw in the arbitration mechanism, and as BXIv3 allows variable-size packets, we propose to store the remaining quantum for that entry when skipped. Therefore, a register for each table entry is needed to track unused quantum, which we call the entry *deficit* ($d_e$, where $e$ is the entry identifier). The next time any queue is searched for packets, its entry*deficit* value is added to the *remaining flits* register. However, these extra flits should only be saved for active queues for two reasons: to avoid starvation in inactive queues when sudden traffic bursts happen and secondly to prevent overflows on deficit counters for on queue inactivity. In the same scenario as the previous example, those 500 remaining bytes would be saved for the next arbitration turn, allowing the queue to send two consecutive packets.

The mechanism for packet scheduling using the arbitration table with deficits is shown in Algorithm 2. The main difference with Algorithm 1 is that when the entry is skipped due to inactivity, the deficit is discarded, as seen in line 19. However, if the sole reason to skip is exceeding the quantum, the deficits are saved for later, as per line 17. This way, we prevent bandwidth loss for active queues. The *deficit* register for any queue is set to 0 when a queue runs out of packets to send before fully consuming its quantum.

**Algorithm 2** Algorithm for arbitration table usage, with deficits added

```
 1: for $e \leftarrow 0$ to numEntries do
 2:     notEmpty ← true if OutQueues[current_entry.Queue] is not empty
 3:     noPFCStop ← true if not PFC stop
 4:     queueActive ← true if notEmpty and noPFCStop
 5:     headPacket ← first packet of OutQueues[current_entry.Queue]
 6:     enoughWeight ← true if headPacket.flits $\leq$ remaining_flits
 7:     if queueActive and enoughWeight then
 8:         selectedQueue ← current queue
 9:         remaining_flits ← remaining_flits - headPacket.flits
10:         if $remaming\_flits = 0$ then                    ▷ Quantum completely depleted
11:             $current\_entry \leftarrow (current\_entry + 1) \bmod n_{entries}$
12:             $remaming\_flits$ ← weights[$current\_entry$] × k
13:         end if
14:         return selectedQueue
15:     else                                   ▷ Decide whether to save deficits or not
16:         if queueActive then
17:             deficits[$current\_entry$] ← remaming_flits
18:         else
19:             deficits[$current\_entry$] ← 0        ▷ Discard if queue becomes inactive
20:         end if
21:     end if
22:     $current\_entry \leftarrow (current\_entry + 1)$ mod numEntries
23:     remaming_flits ← (weights[$current\_entry$] × k) + deficits[$current\_entry$]
24: end for
25: return -1                                         ▷ No valid queue was found
```

Figure 6 shows the same initial state as the example of Fig. 5. The first difference is that now at $t_1$, the remaining flit for queue 0 is saved on the corresponding deficit register. On $t_2$, queue 0 can send up to 3 flits, thanks to the stored deficit. This results on two consecutive selections at $t_2$ and $t_3$ before turn depletion. Note that using deficits reduces the number of arbitration turns.

## 3.4 Latency tuning by spreading table entries

The solutions we have presented are solely meant to distribute the bandwidth among the different queues by using arbitration tables. The simplest implementation uses one table entry per output queue. While this is enough for bandwidth allocation, its effects on latency are not directly proportional to bandwidth allocation, as it is shown in Fig. 7, where entries with more bandwidth assignment are not only assigned more flits on an arbitration sequence but also being checked more often. If entries associated with the same queue are present several times in the table, the maximum distance between entries determines their maximum latency relative to other queues. To do this, the arbitration table must have its entries sorted so that the more priority a queue has, the more often table entries point to it. In turn, packets in more recurrent queues should have their latency reduced. We still maintain that one entry has one

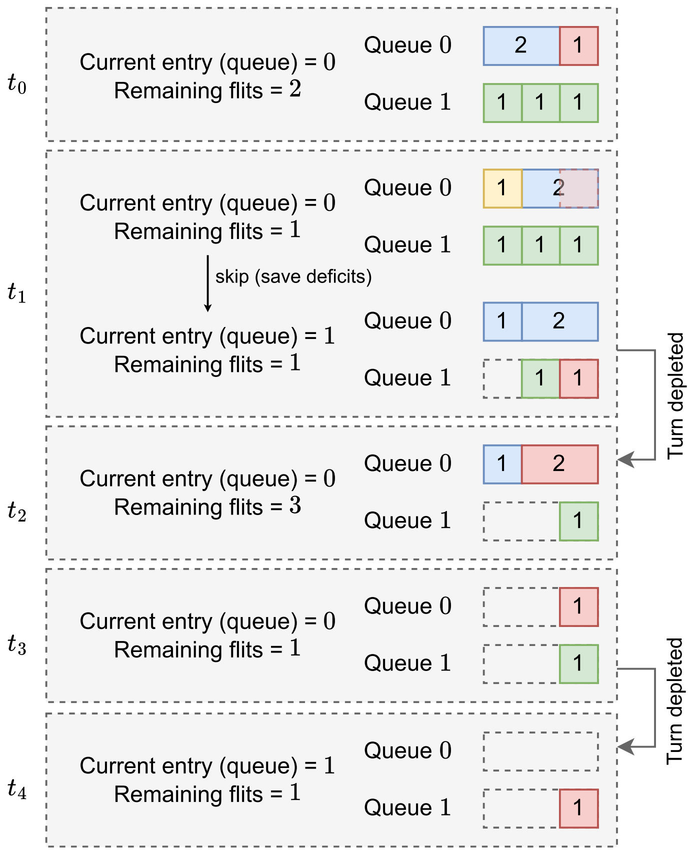


**Fig. 6** Graphical representation for our queue arbitration using the solution with deficits

queue associated with it, but a queue can be pointed at by several table entries. One way to do this is to give each item a stride to move from one similar entry to another, using a geometrical progression with ratio 2. Thus, the top-priority queue should be found every 2 entries, the next one every 4, and so on. Each entry weight must be adjusted proportionally to maintain the same bandwidth allocation. When a turn is skipped, the following queue to be searched will be the one pointed at by the next table entry. In other words, for any arbitration table entry that points to queue $q$, the next table entry may not refer to queue $q + 1$. Other than that, the arbitration logic remains the same.

Table 2 shows how table entries can be distributed for latency tuning. The proportion among every value on the second column defines how much bandwidth is granted to every SL. The stride for each entry type is obtained by the geometrical progression, defining the entry count for each SL.

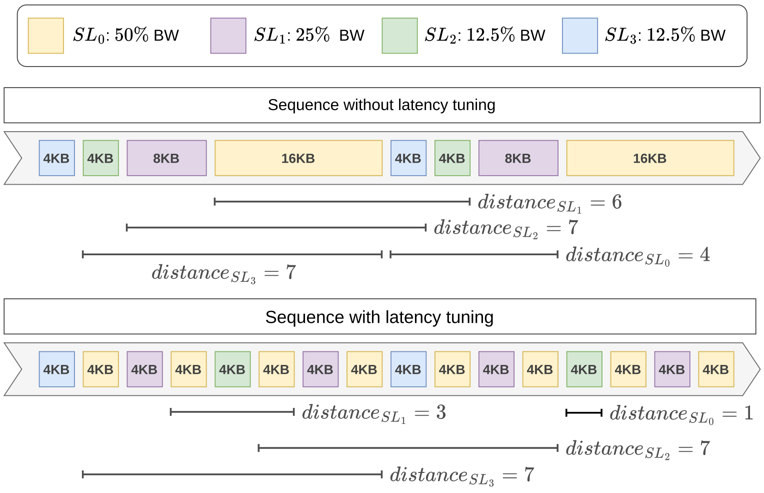


**Fig. 7** Scheduling with and without latency tuning with the same bandwidth allocation. Each square represents a turn with a specific flit length

**Table 2** Example of latency decoupling for 5 SLs

| SL | Total flits | Stride | Entries | Flits/entry | % entries | BW % |
|---|---|---|---|---|---|---|
| 0 | 5120 | 2 | 16 | 320 | 51.61 | 33.33 |
| 1 | 4096 | 4 | 8 | 512 | 25.81 | 26.67 |
| 2 | 3072 | 8 | 4 | 768 | 12.90 | 20 |
| 3 | 2048 | 16 | 2 | 1024 | 6.45 | 13.33 |
| 4 | 1024 | 32 | 1 | 1024 | 3.23 | 6.67 |

## 3.5 Space requirements for the solutions

For the solutions we have described, we assume that every switch or NIC has one read-only arbitration table and several registers for each port. First, we will show the space requirements for the arbitration table alone. Afterward, we will examine the per-port requirements and the total sum for NICs and switches. As each table entry needs to be associated with a specific SL, then we need additional space to encode that SL and its corresponding weight. Thus, the total space for the arbitration table will be:

$$\text{Total table space} = (\text{width}_{\text{SL}} + \text{width}_{\text{weight}}) \times n_{\text{entries}}$$

As an example, we assume 16 different SL values (so, we need 4 bits to encode the SL value) and a 32-entry table with a weight encoded using 16 bits, which are common values for current network technologies. The space for the considered arbitration table is $(4 + 16) \times 32 = 640$ bits. Table 3 shows an example of a 32-entry arbitration table.

For ports to use the arbitration table, the *remaining flits* register must be as wide as the table entry weight, since that value is copied to this register when a turn is over.

**Table 3** Contents and physical structure of an example of a 32-entry arbitration table

| Index | SL (4 bits) | Weight (16 bits) |
|---|---|---|
| 0 | 2 | 500 |
| 1 | 8 | 300 |
| 2 | 1 | 200 |
| .. | .. | .. |
| 31 | 3 | 250 |

The width for the *current entry* one is given by the number of table entries, which may not equate to the number of SLs.

Table 4 shows a breakdown of the port space requirements for our solution.

The length for every item is determined by the maximum value it can store: the *current entry* as an index is well within reasonable limits, up to 256 values. The *remaining flits* register could have a different width if storing bytes instead of flits. For the solution with deficits, each deficit counter must be as wide as the table entry weight. The space required for every item is a trade-off between the possible values it can hold and its byte length. It is up to the specific implementation to decide on other register sizes. With the values we have just shown, the *current entry* and the *remaining flits* registers (items 1 and 2 in Table 4, respectively), add up to 24 bits in total. This size is constant for the basic solution, regardless of the arbitration table entry count. Adding deficit counters, however, does require as many deficit counters (item 3 in Table 4) as table entries. With the example values mentioned, the calculus for space requirements per port, including deficit counters, would be:

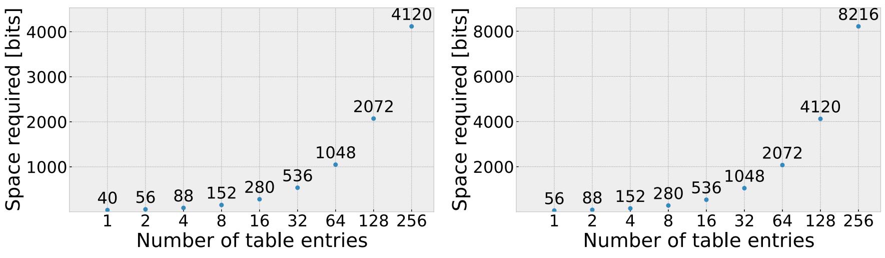
$$\text{Space per port} = \underbrace{16}_{\text{Item1}} + \underbrace{8}_{\text{Item2}} + (\underbrace{16}_{\text{Item3}} \times \underbrace{32}_{\text{table entries}}) = 536 \text{ bits}$$

Figure 8 shows the calculi for NIC space requirements with several arbitration table entry counts, with both the arbitration table and the port registers added. We can see that the space requirements nearly double when deficit counters are used.

On switches, the space requirements increase with both radix and table entry count. The total space requirements are as follows:

**Table 4** Register space for output ports, using a 32-entry table

| # | Description | Width | Count | Total space |
|---|---|---|---|---|
| 1 | Remaining quantum | 16 bits | 1 | 16 bits |
| 2 | Current entry | 8 bits | 1 | 8 bits |
| 3 | Deficit (1 entry) | 16 bits | 32 | 512 bits |



(a) Without deficit counters. (b) With deficit counters.

**Fig. 8** NIC space requirements depending on the arbitration table entry count

$$\text{Space for a switch}$$

$$=$$

$$\underbrace{(\text{width}_{\text{SL}} + \text{width}_{\text{entry}}) \times n_{\text{entries}}}_{\text{Arbitration table}} + \underbrace{(\text{width}_{\text{item 1}} + \text{width}_{\text{item 2}})}_{\text{Port registers}} \times \text{switch\_radix}$$

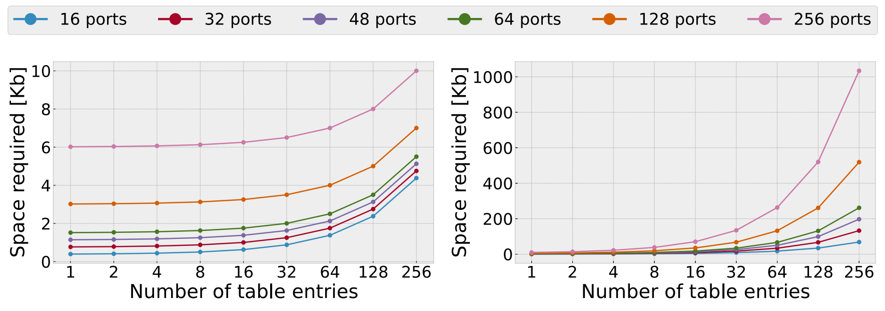
The extra space for deficit counters can be calculated the same way, multiplying by the switch radix:

$$\text{width}_{\text{item 3}} \times n_{\text{entries}} \times \text{switch\_radix}$$

Implementing the basic solution with a 32-entry arbitration table on a BXIv3 switch with 48 ports requires:

$$\underbrace{32 \times (16 + 4)}_{\text{Arbitration table}} + \underbrace{(8 + 16)}_{\text{Port registers}} \times 48 = 1792 \text{ bits}$$

Adding deficit counters would require $(32 \times 16) \times 48 = 24$ *kilobits* more to implement.



(a) Without deficit counters. (b) With deficit counters.

**Fig. 9** Switch total space requirements for several arbitration table sizes and switch radix

Figure 9 shows the space needed for a switch to store the arbitration table and all the registers, based on the number of table entries and the port count. Each series represents a different switch radix. Figure 9a illustrates that the space requirements increase proportionally with the product of the number of ports and the table entry count. However, each port maintains a separate deficit counter for each arbitration table entry when deficits are used. As a result, in this case, space requirements scale with the product of table entries and radix, as shown in Fig. 9b. If the expansion of arbitration table space is a concern, we can map multiple SLs to a single table entry [26, 27].

Still, these space requirements are very reasonable for switch and NIC designs so they can provide QoS to applications.

## 4 Evaluation

### 4.1 Network model

To evaluate our QoS proposal in BXIv3 networks, we have extended the SAURON simulator [3] to include the arbitration tables and the proposed scheduling algorithms. This simulator models the BXIv3 architecture that considers CIOQ switches (see Sect. 2.3) with buffers at both the input and output ports. Regarding network topology, we have assumed a 512-node parallel generalized fat tree (PGFT) with $D$-Mod-$K$ routing [11]. Ports have been configured with five queues, 768 flits long each. The link bandwidth is 400Gbps and the link delay is 25 ns using 5-meter-long optical fiber links. We also assume that five service levels (SLs) or traffic classes are used in the network so that different applications can be mapped to any of these SLs. In these scenarios, we have modeled different arbitration table configurations, called *fair*, *step*, and *2powers*. The details for each scenario will be described in the following sections. We analyze their results compared to the *round-robin* (RR) scheme (i.e., no QoS provision is used).

Concerning the traffic pattern, we have used synthetic traffic and trace-based traffic. First, we have modeled different applications based on synthetic traffic. Each application generates a set of traffic flows addressed to random destinations based on a uniform pattern and a specific generation rate. We assume that the maximum generation rate is the amount of traffic flows the network can process assuming the traffic pattern will not generate persistent congestion situations. Note that, in a 512-node PGFT, the maximum traffic rate the network can process (i.e., network bandwidth) is given by $512 \times Link\_bandwidth$. In our experiments, we assume that the network is generating at the maximum generation rate, the maximum traffic rate the network can process (that is, the network throughput). Apart from the synthetic traffic, we also assume trace-based traffic patterns, as described in Sect. 4.3.

Regarding simulation metrics, in our experiments, we will use the *network throughput* per SL, expressed as a percentage, which depicts the average throughput achieved by the traffic flows of an application assigned with a specific SL. We also use the *packet latency*, which measures the time from packet generation (packetization) at the source host to its arrival at the destination host (also known as end-to-

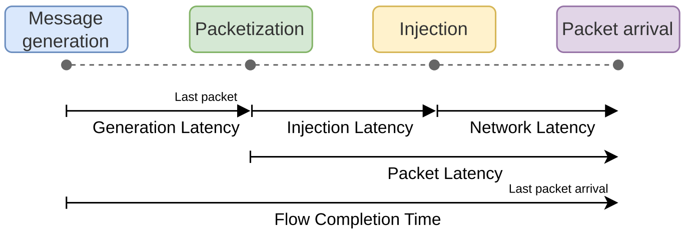


**Fig. 10** The model for the assumed latency metrics

end latency), and *network latency* measures the time from packet injection to arrival at destination. As shown in the figure, the difference between network and packet latency is the injection latency. This is the time that any packet spends in an output queue before being injected. It can be the largest contributor to network latency if congestion trees reach the nodes generating the traffic. Figure 10 shows the different latency metrics. As we are dealing with variable packet sizes, the MTU is only an upper bound for packet size. Messages below the MTU generate packets of the same size.

## 4.2 Experiments with synthetic traffic

In this section, we present the evaluation results with synthetic traffic for three different scenarios. Each scenario has an arbitration table configuration (i.e., *fair*, *step*, and *2powers*), a specific generation rate, and a different message size for every SL. Each message destination is randomly chosen among the available nodes in the network. Our synthetic traffic workload consists of messages of a given size so that, on average, every host generates messages for a percentage of time equal to the target load. Generated messages of the same workload are evenly spaced. Thus, the bigger the messages are, the less and further between in time they are generated. This is important because timing can make a difference in arbitration. SLs are assigned to workloads with different message sizes for our synthetic scenarios to study variable-sized packets. In this case, we generate messages below the MTU for simplicity.

**Table 5** Bandwidth assignment and generation rate for the *fair* arbitration table

| SL | Entries | Weight | Bandwidth (%) | Gen. rate (%) | Msg. size (KiB) |
|---|---|---|---|---|---|
| 0 | 16 | 1 | 20 | 20 | 1 |
| 1 | 8 | 1 | 20 | 20 | 2 |
| 2 | 4 | 1 | 20 | 20 | 3 |
| 3 | 2 | 1 | 20 | 20 | 4 |
| 4 | 1 | 1 | 20 | 20 | 5 |

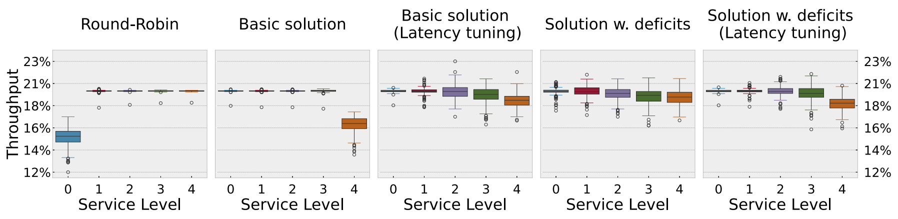
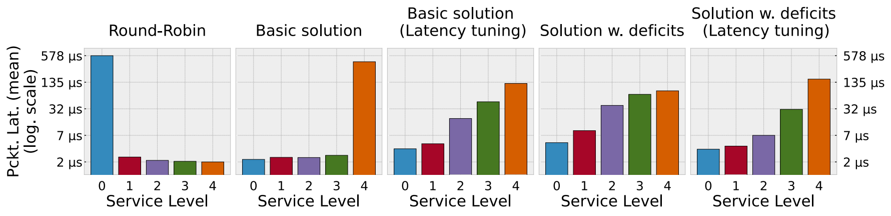
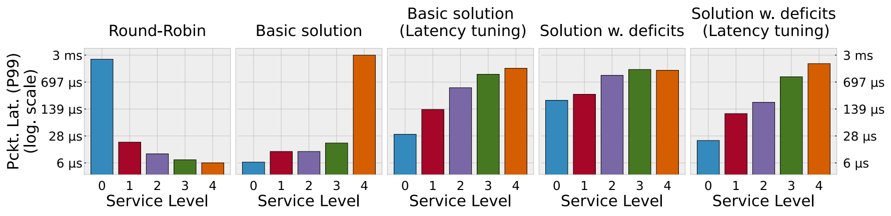


**Fig. 11** Bandwidth allocation results for the *fair* arbitration table for each SL



(a) Average values.



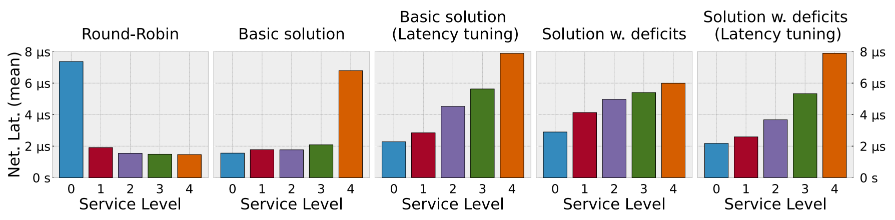
(b) 99th percentile values.

**Fig. 12** Packet latency results for the *fair* arbitration table for each SL

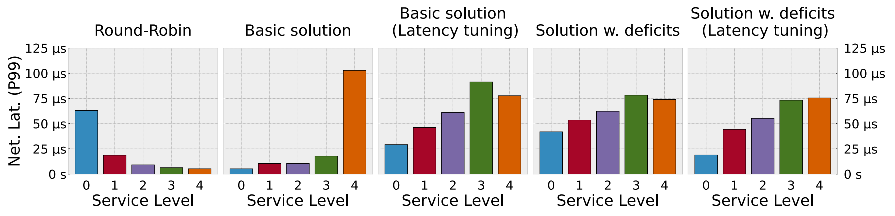
### 4.2.1 Results for the *fair* arbitration table configuration

In this section, we will show the bandwidth distribution obtained for each SL employing our *fair* arbitration table configuration and the network and packet latency, comparing our arbitration proposals to an RR arbitration. Table 5 shows the description of the scenario. The table columns show the number of table entries, the weight of each entry set in the arbitration table, the target bandwidth the table wants to achieve, the generation rate of messages per SL, and the message size defined per SL.

Figure 11 shows the results for this arbitration table configuration. RR arbitration provides equal allocation on a packet basis, but every packet is not of the same bit length. Even if we can see that SL 1–4 medians are similar, minimum values are lower as packet size decreases. This is because, at the same generation rate, those SLs generate fewer and longer packets more frequently. In other words, the longer the packets, the fewer and farther between they are; and vice versa. SL 0 generates more



(a) Average values.



(b) 99th percentile values.

**Fig. 13** Network latency results for the *fair* arbitration table for each SL

**Table 6** Bandwidth assignment and generation rate for the *step* arbitration table

| SL | Entries | Weight | Bandwidth (%) | Gen. rate (%) | Msg. size (KiB) | Gen./BW (%) |
|---|---|---|---|---|---|---|
| 0 | 16 | 5 | 33.33 | 35 | 1 | 105 |
| 1 | 8 | 4 | 26.67 | 28 | 2 | 105 |
| 2 | 4 | 3 | 20 | 21 | 3 | 105 |
| 3 | 2 | 2 | 13.33 | 15 | 4 | 112.5 |
| 4 | 1 | 1 | 6.67 | 8 | 5 | 120 |

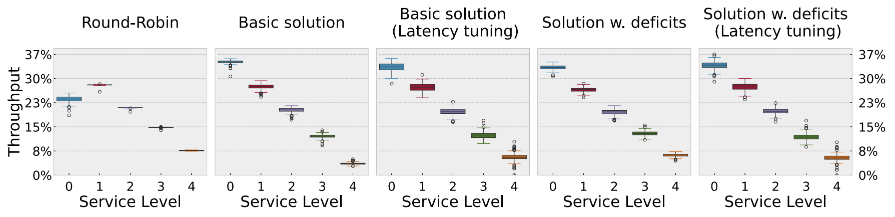
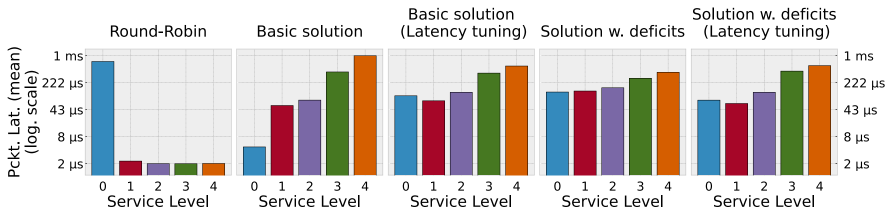


**Fig. 14** Bandwidth allocation results for the *step* arbitration table for each SL

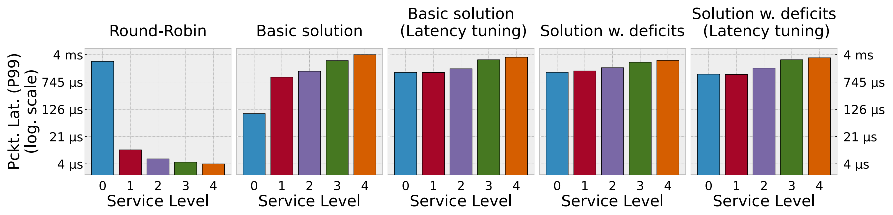
packets per unit of time than any other SL, but unequal packet size results in different bandwidth allocations.

The throughput results of our basic proposal show the opposite effect, with SL 4 receiving less bandwidth than expected. Since SL 4 is only assigned one table entry, it must either exhaust all of its quanta at once or skip its turn, which would result in bandwidth loss. Being the SL with the highest generation time between messages, its turn is usually skipped with some quantum left, losing bandwidth. In other words, bandwidth loss on SL 4 is due to the timing of packet arrival to buffers. In addition, note that RR favors SLs that employ larger packets, whereas our basic solution does the opposite. If we apply latency tuning to our basic solution, this trend for bandwidth loss is minimized. Using shorter turns means that quantum discard is done in smaller quantities compared to not using it. The bigger variation in bandwidth assignment, compared to not using latency tuning, is caused by timing factors, as queues with bigger packets will empty more often. If we add the deficit counters, throughput medians are even among every SL, within 1%. This is because SLs with longer and further between packets are more prone to quantum discarding because of timing. With latency tuning, values for SL 4 are generally lower because of its queue becoming inactive more often.

Packet latency results are shown in Fig. 12a. When no QoS is provided, SL 0 has the highest average value. This is caused by higher packet accumulation on its queue. Other queues with lower packet count, even if they reach the same total length in bytes, do not show such a difference in packet latency. However, the packet latency results are inverted using the default solution. Now that the scheduling algorithm is aware of the different packet sizes, smaller packets do not accumulate on output buffers as much. Higher packet latency for SL 4 is a result of quantum discarding. If a long packet exceeds its remaining quantum, it cannot be transmitted until the remaining SLs have exhausted or skipped their respective turns. If we add our latency tuning technique, mean values are now different among every SL, but the drastic increase for SL 4 is greatly reduced. Figure 12b shows the 99th percentile values for packet latency, which correspond to packets that suffer the most degradation. Using both our basic solutions, latency tuning reproduces the bias toward small packets.

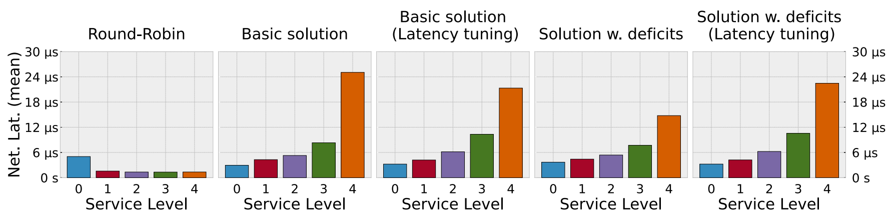


(a) Average values.

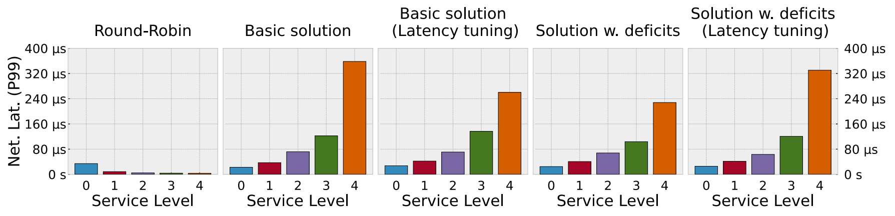


(b) 99th percentile values.

**Fig. 15** Packet latency results for the *step* arbitration table for each SL



(a) Average values.



(b) 99th percentile values.

**Fig. 16** Network latency results for the *step* arbitration table for each SL

If we add deficit counters, the longer packets are, the higher their packet latency. Smaller packets are less likely to stall on output buffers, more so when the scheduling algorithm is aware of packets of variable size. Latency tuning, on average, produces a remarkable reduction in packet latency, particularly in SLs 0–2. If we look at the 99th percentile values, we can see that the differences are even bigger.

As shown in Fig. 13a, values follow a similar trend to packet latency, meaning that most of the contribution to network latency comes from packets stalling on their way to their destinations. Regarding network latency, both schedulers exhibit similar behavior when latency tuning is used. If we look at Fig. 13b, we can see that latency tuning behaves as the previous configuration.

### 4.2.2 Results for the *step* arbitration table configuration

This section analyzes how our solutions impact bandwidth and latency for different applications, using the *step* arbitration table configuration, in contrast to a regular RR

**Table 7** Bandwidth assignment and generation rate for the *2powers* arbitration table

| SL | Entries | Weight | Bandwidth (%) | Gen. rate (%) | Msg. size (KiB) | Gen./BW (%) |
|---|---|---|---|---|---|---|
| 0 | 16 | 16 | 51.61 | 64 | 1 | 124 |
| 1 | 8 | 8 | 25.81 | 32 | 2 | 124 |
| 2 | 4 | 4 | 12.9 | 16 | 3 | 124 |
| 3 | 2 | 2 | 6.45 | 8 | 4 | 124 |
| 4 | 1 | 1 | 3.23 | 4 | 5 | 124 |

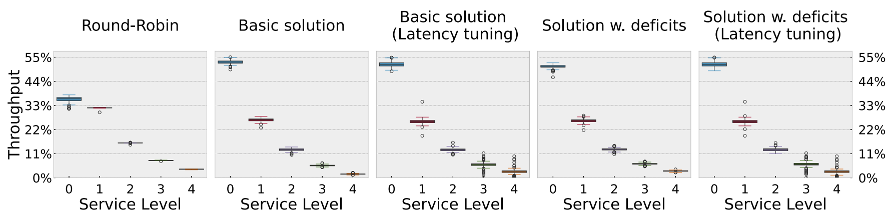


**Fig. 17** Bandwidth allocation results for the *2powers* arbitration table for each SL

arbitration. Table 6 shows the generation rates and the bandwidth assignment of this arbitration table for each service level. This time, we added another column including the ratio between the generation rate and the assigned bandwidth.

The throughput results for this arbitration table configuration are displayed in Fig. 14. As we can see, SLs 2–4 get as much bandwidth as they generate. SLs 0–1 generate more packets, but unequal packet sizes again lead to different bandwidth assignments. With this arbitration table configuration, our basic proposal provides a more accurate bandwidth allocation. As with the previous configuration, SLs that generate larger packets discard quantum more often, causing a loss in effective bandwidth. If latency tuning is enabled, median values are closer to the expected bandwidth. Moreover, previous outliers are now within the first and third quantiles. Adding the deficit counters results in a more accurate bandwidth allocation. As shown in Fig. 14, we have avoided the quantum discarding to a point where bandwidth allocation is not greatly affected by packet size. If we combine the deficits with latency tuning, we can see that throughput values present a slight variance.

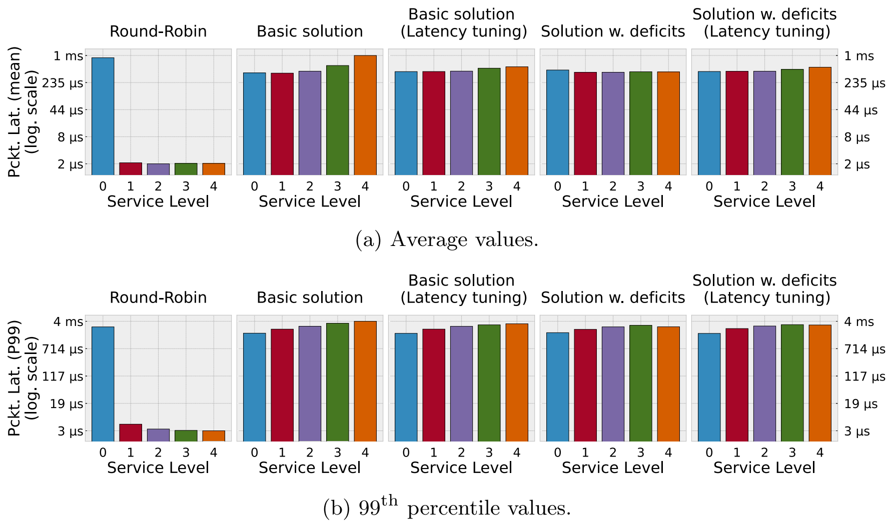
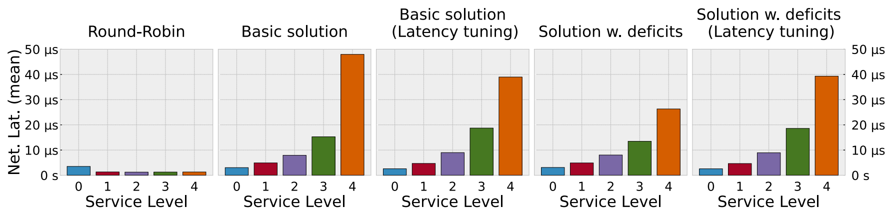


**Fig. 18** Packet latency results for the *2powers* arbitration table for each SL

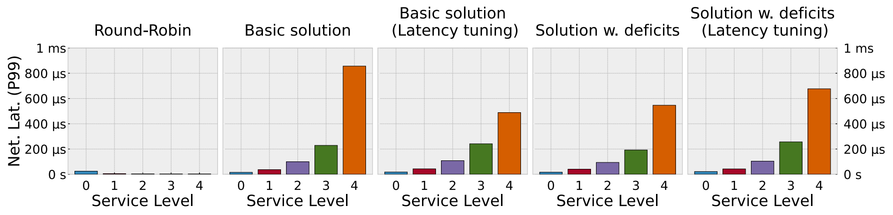
However, values above the third quantile and upper outliers are higher, compared to not using it.

Packet latency results are shown in Fig. 15a. RR scheduling causes packets marked as SL 0 to be stalled more than packets of any other SL. Moreover, SL 0 now has a higher generation rate than the others, so even more packets are accumulating on buffers. Once our basic solution is used, SL 0 is favored. As we explained earlier, smaller packets are less likely to discard quantum, lowering the chances for packets to stall. Moreover, the advantage of SL 0 in packet latency is a consequence of their extra bandwidth assignment at the cost of SLs 2–4. Using latency tuning, values are brought closer (at the expense of SL 0) and the maximum latency is lowered. However, the average for SL 0 average rises. If we compared its throughput values to not using latency tuning, we could see that its first quantile and lower values were inferior. The amplitude of throughput values indicates that shorter turns cause all SLs to discard quantum more often. Once we add deficit counters to our scheduler, the difference between average packet latency results is reduced by orders of magnitude. This is because deficits allow SLs to use the quantum that would be otherwise discarded. Latency tuning does not improve the packet latency of SL 0 over the others on either scheduler. Even if SL 0 is seemingly less favored than SL 1 when using latency tuning with either scheduler, Fig. 15b shows otherwise.

Regarding network latency, Fig. 16a shows that SL 0 is affected by the arbitration algorithm. In the same way as packet latency, the extra bandwidth assigned to SLs 0–1 lowers their latency. Adding deficit counters provides a more accurate bandwidth assignment, and does not affect SLs 2–4 that much. Employing latency tuning reduces the tendency for the basic solution to favor smaller packets. However, on the deficits solution, latency tuning raises the network latency for SLs 2–4, but SLs 0–1 do not benefit significantly. Figure 16b shows the 99th percentile values with this



(a) Average values.



(b) 99th percentile values.

**Fig. 19** Network latency results for the *2powers* arbitration table for each SL

**Table 8** Summary of arbitration tables used for the real traffic experiments

| SL | # Entries | BW (fair) (%) | BW (step) (%) | BW (2powers) (%) |
|---|---|---|---|---|
| 0 | 4 entries | 33.33 | 50.00 | 57.14 |
| 1 | 2 entries | 33.33 | 33.33 | 28.57 |
| 2 | 1 entry | 33.33 | 16.67 | 14.29 |

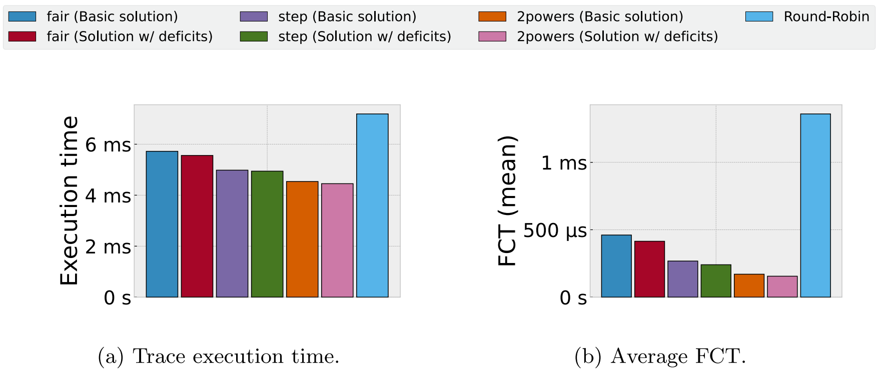


**Fig. 20** Execution time and average FCT for the SL0 application

configuration. The values follow the same pattern as the averages for each SL, meaning that latency tuning is not relevant for this metric under this arbitration table configuration.

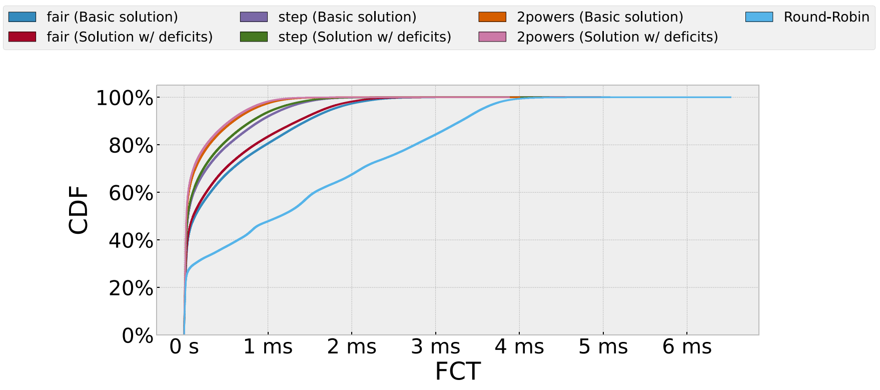


**Fig. 21** CDF for FCT

### 4.2.3 Results for the *2powers* arbitration table configuration

This section will study how bandwidth and latency are affected by our link schedulers when using the *2powers* arbitration table configuration. Table 7 summarizes the bandwidth assignment and generation rates for each SL.

Results for bandwidth allocation are shown in Fig. 17. The behavior is similar to the previous scenarios: more than 20 % bandwidth is only guaranteed if other SLs generate below that value. The default solution behaves similarly, favoring SLs 0–1 over the others. Using latency tuning causes a slight variation in bandwidth assignment, specially on SLs that use larger packets. SLs 3–4 present the most outliers, but once deficit counters are used, the accuracy of bandwidth allocation increases. Latency tuning on this scheduler also causes high variation in results with no improvement in bandwidth assignment.

Packet latency scales for SL 0 to various orders of magnitude above the others when using RR arbitration, as shown in Fig. 18a. Using the default solution increases latency because our basic solution favors smaller packets. Once we add deficit counters, packet latency is very similar among every SL. This is because using this arbitration table configuration, weight and table entries for each SL follows the same geometrical progression. Latency tuning has the same results for both schedulers. Figure 18b shows the 99th percentile values of this arbitration table. Under this arbitration table configuration, the bandwidth assignment alone is responsible for the differentiated latencies. This explains why the worst possible latency results for each SL present relevant differences between using either scheduler or latency tuning.

If we examine the results for network latency shown in Fig. 19a, we see that RR causes SL 0 packets to stall more often. Our basic solution results in differentiated latencies, but favors short packets, hindering SL 4 packets especially. Latency tuning

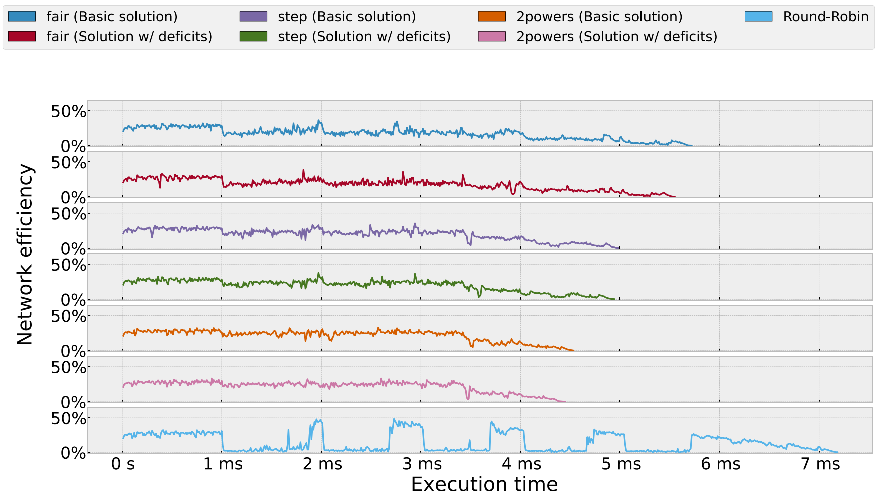


**Fig. 22** Evolution of network efficiency for the trace application

**Table 9** Absolute values and relative performance gains for the trace application when provided with QoS

| Arb. Table | Arbiter | Execution time | | Flow completion time | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 50th perc. | | 75th perc. | | 90th perc. | | 99th perc. | |
| | | Value | Gain | Value | Gain | Value | Gain | Value | Gain | Value | Gain |
| – | Round-Robin | 7.19 ms | – | 1.22 ms | – | 2.56 ms | – | 3.40 ms | – | 6.53 ms | – |
| *fair* | Basic sol | 5.72 ms | 20.41 % | 150.29 $\mu$s | 87.70 % | 827.86 $\mu$s | 67.71 % | 1.58 ms | 53.61 % | 5.08 ms | 22.11 % |
| *fair* | Sol. w. Deficits | 5.56 ms | 22.67 % | 123.80 $\mu s$ | 89.87 % | 710.22 $\mu$s | 72.30 % | 1.48 ms | 56.45 % | 4.98 ms | 23.79 % |
| *step* | Basic sol | 4.98 ms | 30.70 % | 54.05 $\mu s$ | 95.58 % | 424.35 $\mu$s | 83.45 % | 1.00 ms | 70.50 % | 4.41 ms | 32.40 % |
| *step* | Sol. w. Deficits | 4.94 ms | 31.24 % | 47.88 $\mu s$ | 96.08 % | 365.94 $\mu$s | 85.73 % | 900.70 $\mu$s | 73.48 % | 4.36 ms | 33.23 % |
| *2powers* | Basic sol | 4.54 ms | 36.89 % | 31.84 $\mu s$ | 97.39 % | 228.94 $\mu$s | 91.07 % | 665.34 $\mu$s | 80.41 % | 4.01 ms | 38.58 % |
| *2powers* | Sol. w. Deficits | 4.45 ms | 38.07 % | 28.85 $\mu s$ | 97.64 % | 197.92 $\mu$s | 92.28 % | 618.22 $\mu$s | 81.80 % | 3.89 ms | 40.49 % |

**Table 10** Percentage of improvement of using deficit counters over our basic solution in execution time and FCT

| Arb. Table | Execution time | Flow completion time | | | |
|---|---|---|---|---|---|
| | | 50th perc. (%) | 75th perc. (%) | 90th perc. (%) | 99th perc. (%) |
| *fair* | 2.83 | 2.83 | 14.21 | 6.11 | 2.16 |
| *step* | 0.78 | 0.78 | 13.77 | 10.10 | 1.23 |
| *2powers* | 1.86 | 1.86 | 13.55 | 7.08 | 3.10 |

mitigates these effects using this scheduler. Using deficit counters reduces network latency for SL 2–4 packets with no meaningful impact on the other SLs. Latency tuning has its effect reduced under this arbitration table configuration because the bandwidth allocation is responsible for the latency reduction in the first place.

### 4.3 Experiments with a real traffic pattern

To test our model's behavior with real application traffic patterns, we ran our simulator using the VEF-TraceLib [8] library and injecting a traffic pattern based on that of a Google search data center [28], assigned to SL 0. To imitate bursts often found in Datacenters [2], an application assigned to SL 1 generates many-to-one messages every millisecond from half of the nodes to node 0 in bursts of 8 MiB. Lastly, a random synthetic traffic load using 20% total bandwidth runs in the background. It generates jumbo frames (MTU-sized packets) to introduce noise. These experiments aim to observe any performance degradation in the application set to SL 0. We will use the same arbitration tables as before with synthetic traffic using 3 SLs and latency tuning. The summary for each arbitration table configuration is shown in Table 8.

To measure trace application improvement in performance, we will examine the execution time and the flow completion time (FCT), which is the time passed from message generation to its complete arrival at the destination (see Fig 10). Figure 20a shows that the application's execution time decreases as more bandwidth is assigned. The highest speed-up is achieved using the *2powers* arbitration table configuration, reducing the execution time by approximately a third compared to no QoS provision. The average values for FCT are shown in Fig. 20b.

Figure 21 shows the cumulative distributed function (CDF) of FCT for SL 0. We can see that using the default RR arbitration provides the worst results by far, with approximately 50% of messages reaching their destination in 1 ms or more. With QoS provision, however, almost 80% of FCT values are below this value regardless of the assigned bandwidth.

We can achieve a significant reduction in FCT using deficits, and even more with a higher bandwidth allocation. The significant latency reduction is caused by SL 0 receiving more than half of the arbitration table entries assigned. Figure 22 shows the evolution of throughput for the trace application with several configurations. We can see the degradation occurrences coincide with each traffic burst. Moreover, when the trace application starts to recover, a new spike in traffic happens. This is why the application requires more time to deliver all its packets. Every arbitration table mitigates the throughput drop as more bandwidth is granted to the application.

Table 9 provides an insight into the performance gains, in both absolute and relative values, achieved by every combination of QoS solution and arbitration table. We have chosen the execution time and FCT. Executing a program within a deadline can be a dealbreaker for certain applications, and FCT distribution provides a good idea about how much degradation an application may suffer latency-wise. The 99th percentile of FCT, often referred to as Tail Latency, can be used as a value for worst-case scenarios in terms of latency.

Table 10 shows the improvement of the solution with deficits over the basic solution for every arbitration table configuration regarding FCT and execution time. We can see that using deficit counters has improved all of our experiments.

## 5 Conclusions

In this paper, we have shown the necessity to provide applications with QoS in current interconnection networks. Today SC and DC applications present some QoS requirements that IT providers should satisfy. Specifically, we have proposed two different approaches to provide applications with QoS in a BXIv3 (Ethernet-based) high-performance interconnection network. We have described a simple approach based on arbitration tables and an enhancement to that approach based on adding deficit counters. Our initial proposal for the target bandwidth assignment was seemingly accurate. However, once studied thoroughly, it showed its bias toward short packets. Once we improved it with deficit counters, we reverted this favoritism, increasing bandwidth assignment accuracy. Even if space requirements for latency tuning seem steep, all realistic possibilities add up to KiBs in scale. The main outcome of our purely synthetic scenarios is that arbitration tables are perfectly valid for bandwidth allocation and differentiated latencies, more so if we add deficit counters. Moreover, in our more realistic traffic scenarios, far from reducing the degradation of the Google search application, our proposal has provided a speed-up over the original scenario. The experiments with real traffic show that the effect of our proposal on applications is palpable and not a marginal improvement. For instance, reducing the execution time for our top-priority application could be decisive if a deadline is present. Time-constraint responses in applications can also be critical, particularly under network congestion. Apart from the latency and throughput studies, we have thoroughly evaluated the hardware requirements of both proposals, whose results show that they are quite reasonable to implement in current NICs and switches.

Future work involves experimenting with larger networks representing realistic HPC environments and combining our QoS implementation with other techniques such as congestion management or adaptive routing. We would also like to study different values for weight to quantum calculi to minimize bandwidth loss due to quantum discarding.

**Acknowledgements** This project has been funded by the Junta de Comunidades de Castilla-La Mancha and FEDER funds, under the project SBPLY/21/180225/000103; the Spanish Ministry of Science and Universities, under the project PID2021-123627OB-C52; Ministerio de Transformación Digital y Función Pública, under grant Cátedras PERTE Chip (TSI-069100-2023-0014); and Ministerio de Ciencia e Innovación/AEI and European Union (NextGenerationEU/PRTR), under project TED2021—130233B—C31 (grant 10.13039/501100011033). Moreover, this work is supported by RED-SEA, a project that receives funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955776. For this project, the EuroHPC-JU receives support from the European Union's Horizon 2020 research and innovation programme and the Ministry of Science, Innovation and Universities of Spain under grant PCI2021-121976.

**Author contributions** M.S wrote the main manuscript and prepared the figures. JE-S provided technical simulation details and support. FA, JS, and FA-C provided technical guidance. P-AL provided information about the underlying technology. All authors reviewed the manuscript.

**Funding** Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

**Data availability** No datasets were generated or analyzed during the current study.

## Declarations

**Conflict of interest** The authors declare no conflict of interest.

## References

1. Hoefler T, Hendel A, Roweth D (2022) The convergence of hyperscale data center and high-performance computing networks. Computer 55(7):29–37. https://doi.org/10.1109/MC.2022.3158437
2. Shan D, Ren F, Cheng P, Shu R, Guo C (2018) Micro-burst in data centers: observations, analysis, and mitigations. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP), pp. 88–98. https://doi.org/10.1109/ICNP.2018.00019
3. Yébenes P, Escudero-Sahuquillo J, García PJ, Alfaro-Cortés F-J, Quiles FJ (2017) Providing differentiated services, congestion management, and deadlock freedom in dragonfly networks with adaptive routing. Concurr Comput: Pract Exp 29(13):4066
4. Savoie L (2019) Mitigating inter-job interference via process-level quality-of-service. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–5
5. Guo L, Congdon P (2021) IEEE 802 nendica report: intelligent lossless data center networks. In: IEEE SA Industry Connections IEEE 802 Nendica Report: Intelligent Lossless Data Center Networks, pp 1–44
6. Cano-Cano J (2021) A methodology to enable QoS provision on InfiniBand hardware. J Supercomput 77:1–13
7. Páll S, Zhmurov A, Bauer P, Abraham M, Lundborg M, Gray A, Hess B, Lindahl E (2020) Heterogeneous parallelization and acceleration of molecular dynamics simulations in gromacs. J Chem Phys 153(13):134110. https://doi.org/10.1063/5.0018516
8. Andújar FJ, Villar JA, Sánchez JL, Alfaro FJ, Escudero-Sahuquillo J (2016) An open-source family of tools to reproduce MPI-based workloads in interconnection network simulators. J Supercomput. https://doi.org/10.1007/s11227-016-1757-0

9. Zahavi E (2011) Fat-trees routing and node ordering providing contention free traffic for MPI global collectives, vol. 72. https://doi.org/10.1109/IPDPS.2011.219
10. Flajslik M, Borch E, Parker M (2018) Megafly: a topology for exascale systems, pp. 289–310. https://doi.org/10.1007/978-3-319-92040-5_15
11. Zahavi E (2010) D-Mod-K routing providing non-blocking traffic for shift permutations on real life fat trees. CCIT Report 776:840
12. Zhu Y, Eran H, Firestone D, Guo C, Lipshteyn M, Liron Y, Padhye J, Raindel S, Yahia MH, Zhang M (2015) Congestion control for large-scale RDMA deployments. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, pp 523–536
13. Black DL, Wang Z, Carlson MA, Weiss W, Davies EB, Blake SL (1998) An architecture for differentiated services. RFC Editor. Issue: 2475 Num Pages: 36 Series: Request for Comments Published: RFC 2475. https://doi.org/10.17487/RFC2475
14. Top500.org: TOP500 list. June 2024 list. Last accessed: 2024-11-08. https://www.top500.org/lists/top500/
15. Crupnicoff D, Das S, Zahavi E (2005) Deploying Quality of Service and congestion control in InfiniBand-based data center networks. Mellanox White Paper, Rev **1**
16. Birrittella MS, Debbage M, Huggahalli R, Kunz J, Lovett T, Rimmer T, Underwood KD, Zak RC (2015) Intel®omni-path architecture: enabling scalable, high performance fabrics. In: 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, pp 1–9. https://doi.org/10.1109/HOTI.2015.22
17. Jokanovic A, Sancho JC, Labarta J, Rodriguez G, Minkenberg C (2012) Effective quality-of-service policy for capacity high-performance computing systems. In: 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, pp. 598–607. https://doi.org/10.1109/HPCC.2012.86
18. Cano-Cano J, Andújar FJ, Alfaro-Cortés FJ, Sánchez JL, Mora G (2022) Providing quality of service in Omni-Path networks. J Supercomput 78(10):12310–12343
19. Martinez-Morais R, Alfaro-Cortes FJ, Sanchez JL (2010) Providing qos with the deficit table scheduler. IEEE Trans Parallel Distrib Syst 21(3):327–341. https://doi.org/10.1109/TPDS.2009.75
20. De La Rosa MS, Gomez-Lopez G, Andújar FJ, Escudero-Sahuquillo J, Sánchez JL, Alfaro FJ, Lagadec P-A (2024) Quality-of-service provision for BXI3-based interconnection networks. In: 2024 IEEE Symposium on High-Performance Interconnects (HOTI), pp. 20–23. https://doi.org/10.1109/HOTI63208.2024.00015
21. Atos: high performance interconnect for extreme HPC workloads. Last accessed: 2022-10-05. https://atos.net/wp-content/uploads/2020/10/Factsheet_BXI_V2.pdf
22. Derradji S, Palfer-Sollier T, Panziera J-P, Poudes A, Atos FW (2015) The BXI interconnect architecture. In: Proceedings of the 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects. HOTI '15, pp. 18–25. IEEE Computer Society, USA. https://doi.org/10.1109/HOTI.2015.15
23. BullSequana eXascale Interconnect V3: Intelligent Network Management Accelerates GPU Performance in AI-HPC. Technical report, Eviden (November 2024). https://eviden.com/wp-content/uploads/2024/11/Eviden_White-Paper_BXI-V3.pdf
24. Biagioni A (2022) RED-SEA: network solution for exascale architectures. In: 2022 25th Euromicro Conference on Digital System Design (DSD), pp 712–719. https://doi.org/10.1109/DSD57027.2022.00100
25. Barrett B, Brightwell RB, Grant R, Pedretti K, Wheeler K, Underwood KD, Riesen R, Maccabe AB, Hudson T, Hemmert S (2017) The portals 4.1 network programming interface. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States)
26. Martinez A, Alfaro FJ, Sanchez JL, Quiles FJ, Duato J (2007) A new cost-effective technique for qos support in clusters. IEEE Trans Parallel Distrib Syst 18(12):1714–1726. https://doi.org/10.1109/TPDS.2007.1108
27. Martínez A, Martínez R, Alfaro FJ, Sánchez JL (2007) A low-cost strategy to provide full QoS support in advanced switching networks. J Syst Arch 53(7):355–368

28. Montazeri B, Li Y, Alizadeh M, Ousterhout J (2018) Homa: a receiver-driven low-latency transport protocol using network priorities. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. SIGCOMM '18, pp. 221–235. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3230543.3230564

## Authors and Affiliations

**Miguel Sánchez de la Rosa[1] · Gabriel Gomez-Lopez[1] · Francisco J. Andújar[2] · Jesús Escudero-Sahuquillo[1] · José L. Sánchez[1] · Francisco J. Alfaro-Cortés[1] · Pierre-Axel Lagadec[3]**

✉ Miguel Sánchez de la Rosa
miguel.sanchez@uclm.es

Gabriel Gomez-Lopez
gabriel.gomez@uclm.es

Francisco J. Andújar
fandujarm@infor.uva.es

Jesús Escudero-Sahuquillo
jesus.escudero@uclm.es

José L. Sánchez
joses.garcia@uclm.es

Francisco J. Alfaro-Cortés
fco.alfaro@uclm.es

Pierre-Axel Lagadec
p.lagadec@atos.net

[1] Department of Computing Systems, Universidad de Castilla-La Mancha, Avda. de España, Albacete 02071, Castilla-La Mancha, Spain

[2] Departamento de Informática, Universidad de Valladolid, Plaza de Santa Cruz, 8, Valladolid 47002, Castilla y León, Spain

[3] ATOS, Quai Voltaire, Bezons 95870, France