A Tunable Implementation of Quality-of-Service Classes for HPC Networks

Kevin A. Brown[1(✉)], Neil McGlohon[2], Sudheer Chunduri[1], Eric Borch[3], Robert B. Ross[1], Christopher D. Carothers[2], and Kevin Harms[1]

[1] Argonne National Laboratory, Lemont, USA
kabrown@anl.gov
[2] Rensselaer Polytechnic Institute, Troy, USA
[3] Hewlett Packard Enterprise, Houston, USA

Abstract. High-performance computer (HPC) networks are often shared by communication traffic from multiple applications with varying communication characteristics and resource requirements. These applications contend for shared network buffers and channels, potentially resulting in significant performance variations and slowdown of critical communication operations such as low-latency MPI collectives. In order to ensure predictable communication performance, network resources must be allocated relative to the communication requirements of applications.

Quality of Service (QoS) solutions can regulate the allocation of resources by defining traffic classes with specified resource allocations and assigning applications to these classes, thus improving application performance predictability. However, it is difficult to accomplish facility-level goals of ensuring efficient application communication when constrained to a limited number of classes.

We propose a practical QoS implementation for large-scale, low-diameter networks, such as the dragonfly topology, using flexible bandwidth shaping along with traffic prioritization to reduce the impact of interference on communication performance. Our design gives facilities more control over tuning QoS class to meet application- and site-specific performance guarantees. The results show that our solution effectively eliminates the slowdown of high-priority traffic due to interference with lower-priority traffic, significantly reducing run-to-run variability. We also demonstrate how port counters can be used to detect when a job-to-class assignment is inappropriate for a given system and when a workload is exceeding the bandwidth limits of its class.

Keywords: Interconnect network · 1D dragonfly topology · QoS · Traffic class

1 Introduction

Most high-performance computer (HPC) systems are shared by multiple applications with varying communication characteristics and bandwidth/latency

B. L. Chamberlain et al. (Eds.): ISC High Performance 2021, LNCS 12728, pp. 137–156, 2021.
https://doi.org/10.1007/978-3-030-78713-4_8

requirements. The interconnection networks of large HPC systems use high-speed switches to route application traffic across the system. These applications compete for bandwidth and can oversubscribe the links when the available bandwidth is less than the total required by the competing traffic flows.

Heavy traffic flows on low-diameter networks, such as fat tree and dragonlfy, have been shown to unfairly monopolize link bandwidth when each flow is given equally unregulated access to the network channels [20]. Without access constraints, network contention becomes an issue that can result in reduced or delayed network access for different applications. This may severely harm the performance of certain types of application traffic. For example, this situation can lead to significant performance degradation for latency-sensitive communication traffic, such as small message MPI collectives, while potentially posing negligible impact on more latency-tolerant patterns such as checkpointing [1].

HPC networks use a variety of techniques to deliver high system throughput and good application performance. Adaptive routing can be employed to improve communication performance by re-routing packets around high-traffic areas of the network, balancing traffic load across network links [20]. Congestion management is aimed at diagnosing and treating network congestion by temporarily reducing the rate at which packets are injected into the network, when necessary, to reduce the total number of packets queued in network buffers [19]. However, adaptive routing and congestion management techniques cannot allocate network resources, such as buffers and channel bandwidth, to specific applications or classes of traffic based on their respective performance targets. Quality-of-service (QoS), on the other hand, can differentiate how resources are allocated to different types of traffic to better manage resource contention and interference on heavily loaded networks [6].

Numerous studies exist on QoS for wireless networks, data centers, and the internet [15]. However, these solutions and supporting hardware are typically designed to throttle injection at the source or to drop packets in transit when the flow does not conform to QoS policies. Unfortunately, for HPC, throttling is not ideal unless the overall network is congested and dropping packets increases latency and reduces overall system throughput since dropped packets need to be retransmitted.

QoS solutions for HPC differentiate between different types of traffic by placing them in different network-defined traffic classes. Each traffic class is allocated separate network buffers and a guaranteed fraction of the channel bandwidth, based on the performance requirements of the traffic assigned to the class [13]. A small number of traffic classes are usually shared by multiple applications since it is impractical to define a separate class for each of the myriad of traffic patterns. The effectiveness of the QoS solutions therefore depends on how accurately the classes can be tuned to consistently guarantee an appropriate fraction of the resources to their assigned workload, even when the resource availability varies rapidly as in typical production systems.

Studies up until now have mainly focused on priority-driven QoS or QoS based mainly on simple, course-grained bandwidth allocations [16,18,22]. Such solutions do not have the flexibility to effectively configure classes that account for the variations in workloads, interference patterns, and site-specific priorities in HPC facilities. There is need for a QoS solution that can simultaneously balance the needs of multiple competing applications and reallocate bandwidth in a controlled manner as requirements change. However, in addition to the lack of appropriate solutions, there is also limited knowledge available on how to precisely evaluate the suitability of class configurations for HPC workloads with multiple distinct classes of traffic.

Our work aims to address these issues with the contributions as follows:

- We describe a practical method of implementing QoS classes on large-scale, low-diameter networks to enable traffic differentiation, prioritization, and shaping. Our solution allows for better control of bandwidth allocations when traffic load varies by employing two rate limiters per QoS class: one that sets an assured amount of bandwidth, and one that sets a maximum amount of bandwidth at its priority level;
- We propose a scheme for configuring and deploying QoS classes in production to match the varying application performance requirements of mixed workloads on production HPC systems; and
- We evaluate the ability of our scheme to satisfy the relative performance goals of multiple traffic flows sharing a large-scale 1D dragonfly network.

Our solution successfully regulates the traffic flows of dynamic communication workloads to more consistently meet the performance targets of the respective flows. The two rate limiters enable us to tune classes so as to better match the performance targets of dynamic workloads compared to prior work with single rate QoS solutions.

2 Background and Related Work

2.1 Communication Characteristics and Performance Targets

Communication operations can be characterized as being either *latency-bound* or *bandwidth-bound*. Latency-bound transfers have low injection rates, or offered loads, and the resulting communication time depends on individual packet latencies. Small message MPI collectives, such as MPI_Allreduce, are implemented using algorithms that rely on structured communication to minimize message count and data volume. However, long tail latencies can disrupt the structure of the communication and thus severely degrading collective performance. On the other hand, bandwidth-bound operations move relatively large amounts of data through the network, and the overall communication time depends on the throughput instead of individual packet latencies. Bulk data I/O transfers are examples of bandwidth-bound operations.

To meet their respective performance targets, different types of traffic need access to the network resource in different manners. Keeping packet latencies low in latency-bounded flows requires reducing the time spent queuing in the network by providing higher priority to network channels. Bandwidth-bounded flows, however, require high injection rates or longer access to the shared channels in order to move a large amount of data through the network.

HPC facilities may further classify some traffic as having higher priority than others based on site-specific goals. For example, facility administrators may deem that certain latency-sensitive collective operations such as MPI_Allreduce should not be impeded and instead receive highest priority access to network resources, regardless of the source application. In contrast, they may decide that other types of traffic – such as network-monitoring data – can be delayed if the traffic does not require any performance guarantees.

2.2 Managing Contention for Shared Channels on HPC Networks

Contention for shared resources such as channel bandwidth causes interference to communication performance. Interference over the network significantly degrades the performance of many HPC applications that are characterized by latency-sensitive collective communication patterns [7,8]. The main cause of significant slowdown due to interference is increased queuing delays resulting from head-of-line (HoL) blocking, i.e. when fast-draining messages get stuck behind slow-draining messages in shared buffers on congested ports [22].

Adaptive Routing and Congestion Management. Interconnect congestion can be classified into two categories: *intermediate* and *endpoint* congestion [3]. Intermediate congestion occurs when multiple input ports on a router try to use the same router output port, causing packets to get backed up in the buffers of the router. Endpoint congestion occurs due to application incasts – traffic from multiple source endpoints target the same destination endpoint, overwhelming the endpoint's ability to accept all the incoming traffic. Adaptive routing can effectively address intermediate congestion by routing incoming packets to different fabric output ports to avoid oversubscribing any single output port. However, adaptive routing is ineffective against endpoint congestion because there are no alternative paths to the endpoint at the destination switch. With endpoint congestion, as traffic backs up from the target endpoint, adaptive routing will spread incoming traffic to less busy paths and potentially cause traffic to get backed up on those paths across the network as well. Congestion management schemes that appropriately abate the incast flows are essential for handling endpoint congestion. These solutions strive to curtail the injection volume at the congestion-causing sources based on how many packets can be consumed at the endpoints [19].

2.3 QoS Solutions for HPC

QoS mechanisms are not designed to address endpoint or intermediate congestion. QoS is used to decide which packet to send based on priority and bandwidth specifications. This is complementary to both adaptive routing, which determines the path a packet should take, and congestion management, which determines if a packet should be injected into the network based on the state of congestion in the network.

Several QoS solutions have been proposed [13,16,18,22] for low-diameter networks like dragonfly and fat-tree networks, which are very susceptible to inter-application interference [20]. These solutions use separate buffers for each traffic class in order to prevent HoL blocking across classes and reduce packet latencies. Most of these vary the arbitration priority in some manner to reduce packet latencies, or regulate the bandwidth allocation to different flows, or both. Savoie et al. [18] proposed grouping application traffic flows into separate QoS classes and used only priority as a constraint on the QoS classes. However, Jakanaovic et al. [13] noted that this approach has the potential to degrade overall system performance since a bandwidth-intensive workload in the high-priority class can cause prolonged starvation of other workloads. The bandwidth consumption of the high-priority class should be constrained to prevent unintended starvation.

Wilke and Kenny [22] proposed using four different traffic classes with 25% of the bandwidth allocated to each class. Their solution includes two classes that use minimal routing in order to reduce the number of required buffers – minimal routing requires less buffers than adaptive routing [14]. This limits the flexibility of their solution since two of the four classes cannot be re-purposed to carry bandwidth-intensive traffic if the facility requires this.

Mubarak et al. [16] demonstrated that managing the bandwidth allocated to traffic classes may guarantee that important applications can perform well despite interference from lower-priority jobs. They proposed each class using a single rate limiter that must be tuned relative to the other class in order to assure a fraction of the bandwidth. That is, increasing the bandwidth allocated to one class will reduce the bandwidth available to other classes. Unfortunately, their allocations must be tuned to a fixed set of traffic loads while the load on the network usually varies. It is non-trivial to define a static configuration for all classes that consistently match the performance requirements of their assigned workloads given the variability in available resource.

In deploying QoS classes on production systems, multiple studies [16,18,22] have proposed grouping application traffic flows into a few QoS classes to reduce the number of classes in required[1]. The different flows need to be grouped based on performance requirements and characteristics, and their assigned classes must now be tuned to match their collective requirements.

3 Design of a Tunable QoS Solution

An ideal QoS solution should be able to simultaneously (i) ensure low packet latencies for latency-bound traffic, (ii) guarantee high bandwidth for bandwidth-bound traffic, (iii) prevent unintended starvation, and (iv) provide these assur-

[1] Switch hardware can only support a limited number of actives classes due to resource limitations.

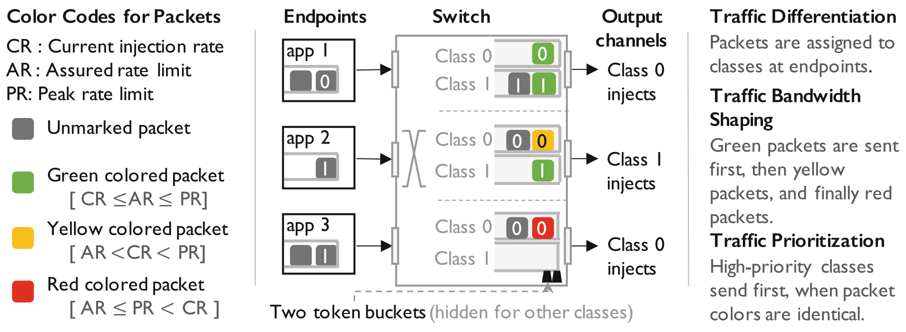
Color Codes for Packets
CR : Current injection rate
AR : Assured rate limit
PR: Peak rate limit
Unmarked packet
Green colored packet [CR ≤ AR ≤ PR]
Yellow colored packet [AR < CR < PR]
Red colored packet [AR ≤ PR < CR]
Endpoints
Switch
Output channels
app 1
app 2
app 3
Class 0
Class 1
Class 0 injects
Class 1 injects
Class 0 injects
Two token buckets (hidden for other classes)
Traffic Differentiation
Packets are assigned to classes at endpoints.
Traffic Bandwidth Shaping
Green packets are sent first, then yellow packets, and finally red packets.
Traffic Prioritization
High-priority classes send first, when packet colors are identical.

Fig. 1. Illustration of our QoS solution design. Packets are assigned to classes at compute node endpoints, placed in the appropriate class buffers on switches, and then are colored and compete for access to output channels based on the QoS policy. (Color figure online)

ances while network load varies. Since throughput and/or packet latencies affect a traffic flow's ability to meet its relative performance targets, QoS mechanisms should allow for regulating resources that affect the resulting packet latency and bandwidth available to traffic flows. Per-class buffers and priority-based arbitration can be used to prevent HoL blocking and expedite packet forwarding, thereby reducing packet latencies. Bandwidth guarantees (or assured injection rates) can be defined on each class to ensure a fraction of channel bandwidth is available to traffic using that class when channels are oversubscribed, thereby preventing starvation. Importantly, QoS should also manage the reallocation of bandwidth when traffic load changes such that more important flows get priority access to bandwidth released by other flows. This provides more useful resource partitioning for dynamic workloads.

3.1 Flexible Traffic Shaping Using Two Rate Limits

In a production system, network bandwidth usage varies dynamically as the traffic load changes. However, most QoS solutions use only a single assured rate limit to allocate bandwidth and cannot accommodate changing load requirements. When flows reduce their injection rates, their *unused* bandwidth is left unregulated for other traffic to consume, regardless of their importance. Controlling how unused bandwidth gets reallocated to specific workloads can improve the performance of important flows. By using an additional *peak rate* limit, we can give priority access to a portion of the unused bandwidth.

We propose a QoS mechanism that can be more easily tuned to satisfy the needs of HPC workloads and reallocate unused bandwidth more efficiently compared to other solutions. This solution allows for configuring an arbitrary number of traffic classes[2] with independent buffers and unique relative priorities to enable traffic prioritization. To achieve bandwidth shaping, each class is configurable with two rate limits: an *assured rate (AR) limit* and a *peak rate (PR) limit*, where $AR \leq PR \leq 100\%$, based on the Two Rate Three Color Marking design [21] for metering packet streams. The AR provides guaranteed bandwidth allocations and $\sum_{i=1}^{n} AR_i \leq 100\%$, where n is the number of classes. PR controls how excess/unused bandwidth is reallocated for controlled traffic shaping as the load changes. In our design, as illustrated in Fig. 1, packets at the front of switch port buffers are marked as either green, yellow, or red, depending on the current injection rate of its respective class. A packet is marked as red if the class exceeds its PR (and hence AR); it is marked as yellow if only the AR (but not PR) has been exceeded; or it is marked as green if its class does not exceed its AR (thus not PR either). Marking is done at each injection cycle for the purpose of output port arbitration, and stalled packets are re-marked based on the new injection rate in subsequent cycles. The packet content is unchanged and marking information is not communicated downstream.

Output port arbitration is priority-based within the constraints of the classes' rate limits. That is, green packets are sent first from higher priority classes; otherwise, yellow packets are sent in a similar priority order when there are no green packets to send be sent. If neither green nor yellow packets can be sent, a red packet will be chosen from any class by round-robin – priorities are ignored and each class has an equal chance of getting access to the output port. Note that flow control can stop any class from sending if downstream buffers are unavailable, in which case a packet from another class is sent.

We use token buckets, to meter each of the two rate limits per class [21]. Tokens accumulate in each bucket at the rate of the limit it meters, i.e., the assured rate bucket will accumulate tokens at the assured rate limit defined on the class, etc. Whenever a green or yellow packet is sent from a class, a token is removed from each of the two buckets with available tokens. No token can be removed when a red packet is sent because the peak rate limit has been exceeded, at which point both buckets are empty. Empty buckets means the traffic has completely consumed the bandwidth allocated to the class.

3.2 Defining QoS Classes for HPC Traffic

QoS classes for HPC traffic should be configured based on the traffic flow they are assigned. Additionally, systems should use the minimum number of classes required for their workloads to prevent resource fragmentation. Classes have strict priorities relative to each other, so we first consider the traffic flow's priority relative to that of other flows when deciding traffic-to-class assignments. When all other factors are equal, the priority will determine which flow progresses first and achieves lower latency. Inline with industry standards and recommendations [9,17], we argue that the following traffic classes and class assignments

[2] The number of traffic classes that can be configured on a given switch will be limited by how many class buffers and rate limiting counters are supported by that switch hardware.

are relevant for the majority of workloads on shared HPC systems and can be efficiently supported by our solution:

Low-Latency Class: Guarantees low packet latencies. This class has the highest arbitration priority to reduce queuing delays and a low assured rate limit to prevent starving other classes.

Suitable Traffic: important traffic that is primarily latency-bound and does not require high throughput, such as small message collectives.

Bulk Data Class: Guarantees high communication throughput. This class is typically allocated bandwidth commensurate with the I/O throughput of the system and the importance of I/O performance to the system workloads.

Suitable Traffic: traffic that moves a lot of data at once, requires high throughput and is not latency-sensitive, such as bulk I/O transfers to network file systems.

Scavenger Class: Guarantees minimal progression of traffic and minimal interference to other classes. This class has the lowest priority and a low assured rate limit to prevent it from impacting the performance of traffic in other classes.

Suitable Traffic: traffic that can be temporarily ignored without significant impact to overall productivity and user experience, such as scraping network counters.

Best-Effort Class: Guarantees best-effort progress of traffic with mixed latency-sensitivity and bandwidth requirements. This class is given a relatively high priority and allocated sufficiently high injection rates based on the high volume of data transferred by its combined expected workload.

Suitable Traffic: traffic that does not strongly map to any other class. Most application traffic will use this class.

Our QoS solution supports these and other class definitions by tuning the dual rate limits and relative arbitration priority on each class. For example, a system may need to support streaming real-time data, in which case such streams may require a high-priority class with the highest bandwidth allocation. One main requirement of an effective traffic-class assignment is that traffic sharing the same class are not adversarial to each other in terms of latency and bandwidth.

The following section demonstrates how traffic shaping with our dual-rate solution provides more consistent communication performance with dynamic workloads than other single-rate QoS solutions. We also show how QoS classes with dual-rate limits can be more easily tuned to simultaneous satisfy multiple performance targets and regulate diverse traffic loads.

4 Evaluation of QoS Solution

4.1 CODES Simulation Toolkit

To collect the data evaluated in this work, we use the CODES HPC interconnection network simulator [5] since HPC hardware does not yet support dual-rate

QoS. CODES is a Parallel Discrete Event Simulation (PDES) toolkit built on top of the Rensselaer Optimistic Simulation System (ROSS) [2] PDES engine. CODES allows for fine-grained, link-level simulations of packets moving across high-performance networks. Additionally, these simulations allow for testing and evaluation of different mechanisms such as adaptive routing algorithms, congestion management, and, as demonstrated in this work, QoS techniques. We implemented our QoS solution in CODES based on the design outlined in the previous section.

4.2 Network Setup

We simulate a tapered 1D dragonfly network with 8320 node endpoints. The network interconnect consists of 1040 routers with 16 routers per group. Each router has eight terminal channels, 15 local channels, and four global channels. The ratio of terminal channels to global channels results in a 2:1 taper of the global network bandwidth, similar to systems such as Theta, Edison, Malbec, and Shandy [10], which increases the potential for contention among competing traffic flows. We use 25 GB/s injection bandwidth for all channels, 10 ns delay for terminal and local channels, and 100 ns delay for global channels. The simulated router delay is 300 ns and the network packet size is set to 160 bytes. These taper and delay configurations are representative real-world dragonfly systems [10]. We use a progressive-adaptive routing algorithm for the network and a random job-to-node allocation scheme. Studies show that this random node allocation strategy improves job throughput for dragonfly systems [23] such as the ones listed above.

4.3 Workload Setup

We use the uniform random traffic (UR) pattern to generate interference on our network because other synthetic patterns, such as random-permutation, can cause congestion hotspots [3] for which QoS is not the appropriate solution. Additionally, unlike real application traffic, this synthetic traffic pattern (i) provides more precise control for managing when and how the traffic load changes and (ii) is less sensitive to the topology, routing, and congestion management capabilities of the systems. This allows us to succinctly capture the difference in traffic shaping capabilities of the dual-rate scheme versus the single-rate scheme.

The UR jobs use 640 B messages and vary the injection load by varying the delay between injecting successive messages, representative of loads recorded on a production HPC system [11]. These small messages allow us to minimize local incasts and evenly spread load across the system. We also use a Scalable Workload Model (SWM) [12] of MPI_Allreduce – a common operation on HPC systems [4] – to simulate latency-sensitive traffic. SWMs are skeletons of applications and benchmarks that capture the communication patterns of the workload that they model. Each allreduce SWM job performs at least 15 calls to

Class Configurations

single-rate

Class	Priority	AR limit (%)
0	P0	20
1	P1	45
2	P2	35
3	P3	0

dual-rate

Class	Priority	AR limit (%)	PR limit (%)
0	P0	20	100
1	P1	45	100
2	P2	35	100
3	P3	0	0

AR: Assure Rate | PR: Peak Rate
P0 is the highest priority and P3 is the lowest.

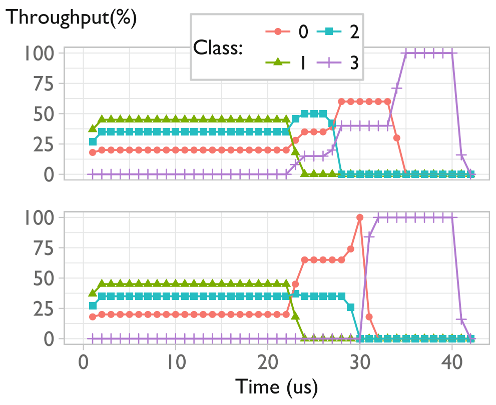

Fig. 2. Partitioning a single port bandwidth between four classes using single-rate and dual-rate QoS solutions. The injection rate in each class 100% of the bandwidth. The peak rate limit on the dual-rate QoS classes ensure that unused bandwidth is reallocated to the highest-priority class when the traffic load changes as the flow in class 1 completes around 23 µs.

MPI_Allreduce, reducing 8 bytes of data across all ranks of the job and requiring very low bandwidth. Our early evaluation survey of different load levels and message sizes produced similar results to experiments presented in this paper.

4.4 Bandwidth Shaping for Dynamic Workloads

The workloads on large production systems often exhibit variations in traffic load as applications start and stop communication operations and vary the volume or frequency sending traffic.

Reallocating Unused Bandwidth with Dual Rate Limits: When a QoS class has an assured bandwidth allocation, it is guaranteed a fraction of the bandwidth of all channels in the system. If traffic in this class does not use all of its allocation on a channel, the *unused* portion of the allocation can be consumed by flows from other classes. Controlling how the unused bandwidth gets consumed can improve the performance of more important flows over less important ones.

To demonstrate that our dual-rate solution allows for controlling the reallocation of unused bandwidth, we simulate four traffic flows sharing the bandwidth of a 16 B/ns channel. Each flow attempts to use 100% of the injection bandwidth to stream 1000 packets over the shared channel. The port is configured with four traffic classes (0, 1, 2, and 3), with one flow assigned to each class. Classes are assured a fraction of the link bandwidth relative to a designated minimum required rate of its assigned flow. The flow in class 3 is designated as having little importance and should not interfere with the other flows; therefore,

Table 1. Configuration for workload with variations in traffic load. UR jobs inject uniform random traffic.

Job	Nodes	QoS class	Initial rate (%)	New rate (%)
all_reduce32_1	32	0 - low latency	<0.08	–
all_reduce32_2	32	0 - low latency	<0.08	–
all_reduce256_1	256	0 - low latency	<0.08	–
all_reduce256_2	256	0 - low latency	<0.08	–
UR-LL	64	0 - low latency	3	–
UR-BE	4160	1 - best effort	50	85
UR-IO	1760	2 - bulk data	80	20
UR-S	1760	3 - scavenger	20	60

class 3 is assured none of the link bandwidth. We compare our QoS solution – which uses two rate limits – to another design that uses a single-rate limit [16].

The class configurations and results for both QoS solutions are shown in Fig. 2. For **single-rate QoS**, classes 0, 1, and 2 are able to share the port's bandwidth at their respective assured rates of 20%, 45%, and 35% from the start of the run. Class 3 is starved and unable to send because it is not assured a fraction of the bandwidth and the port is fully utilized by the other flows. As traffic in classes 1 and 2 complete after 22 µs, the remaining active flows equally share the *unused* bandwidth that becomes available. The flow in class 3 is able to compete for – and consume – a fraction of the unused bandwidth, partially blocking the higher-priority flow in class 0. However, **dual-rate QoS** uses peak rate limits to regulate access to the used bandwidth based on class priority, up until the class's peak rate limit. Our dual-rate solution could also be tuned to reallocate excess bandwidth to other classes besides class 0 by reducing the peak rate limit of class 0.

Maintaining Performance Despite Changing Network Loads: Properly tuned QoS classes should maintain the relative performance targets of their assigned traffic flows regardless of changes in the network load. If a flow requires more than its allocation and unused (or unallocated) bandwidth becomes available, the class should (i) be able to use the available bandwidth if it has sufficiently high priority or (ii) be blocked by another class if the other class is carrying more important traffic as done in the previous experiment.

To demonstrate the effects of system-wide network load variations on performance predictability, we evaluate a workload comprised of eight jobs with multiple changes in traffic load over time. Table 1 describes the jobs and their class assignments. Four allreduce jobs of two different job sizes and a uniform random (UR) job are placed in the low-latency class. The other classes are each assigned one UR jobs with different injection load intensities. The class configurations and UR job injection loads were selected to reflect their class's expected

Injection Rate Variations

Class	Initial	T1	T2	T3
0	3	-	-	-
1	50	-	85	-
2	80	20	-	-
3	20	-	-	60

Dual-rate Classes

Class	Priority	Assured %	Peak %
0	P0	5	10
1	P1	30	80
2	P2	20	60
3	P3	5	20

Single-rate Classes

Class	Priority %	Assured %
0	P0	10
1	P1	50
2	P2	35
3	P3	5

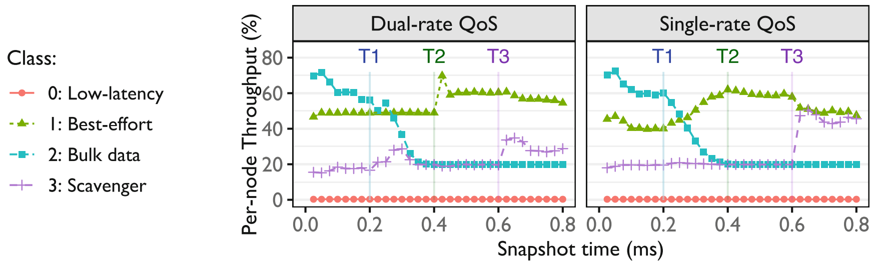

Fig. 3. Change in class throughput over time as the injection rates vary at times T1, T2, and T3, as indicated in the *Injection Rate Variations* table.

workloads that were discussed in Sect. 3. That is, the *low-latency class* will guarantee low packet latencies for traffic with a light injection load; the *best-effort class* will carry most application traffic and should guarantee high throughput; the *bulk data* class will carry I/O data and should have sufficiently high bandwidth without interfering with the low latency and best effort classes; and the *scavenger class* should ensure progress of its traffic while causing minimal interference to other flows. Details of the class configurations will be discussed in the following subsection. We create these classes for both dual-rate and single-rate QoS schemes to study each scheme's ability to maintain performance predictability as the traffic load varies. Figure 3 shows the class configurations and the resulting injection throughput during the run. The plots report the average per-node throughput of traffic in each class for both QoS schemes. If traffic flows from two classes that never share a channel, there will be no interference and both flows can theoretically be injected at 100% of the peak node injection rate simultaneously. However, on large systems, flows from multiple classes will contend for shared channels and potentially reduce the throughput that each flow can sustain. The QoS solutions manage this contention to improve workload throughput.

The results in Fig. 3 show that dual-rate QoS classes guarantee consistently high throughput for class 1 (best effort) traffic throughout the experiment. The flow in class 1 maintains its initially desired 50% rate until its injection rate is increased to 85% at 0.4 ms (T2 in the plot), exceeding the peak rate limit of 80% for class 1. From that point, it could only sustain a 60% injection rate due to the heavy load on the network causing the excess packets to be stalled and reducing the *effective* available global bandwidth. The increased network load and stalls caused ≈8% more packets to be routed non-minimally in class

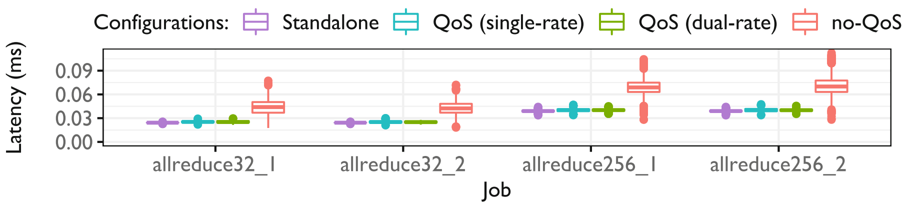

Fig. 4. Distribution of MPI_AllReduce operation latency across the ranks of each allreduce jobs. We achieve near baseline (Standalone) latencies when allreduce jobs use the low latency class of the dual-rate and single-rate QoS configurations.

1 between 0.4 ms and 0.8 ms. Non-minimally routed packets take two global hops on dragonfly networks, and the additional hop reduces the effective global bandwidth [14]. With single-rate QoS, class 1 could sustain only a 40% injection rate even though it desires 50% and it is assured 50% of the system bandwidth. This flow is able to increase its throughput after the load from class 2 is reduced at 0.2 ms (T1). The flow in class 2 has high throughput at start of the simulation when class 1 was not very loaded. After its load is reduced to 20%, which is within its assured rates for both solutions, it sustains this throughput for the rest of the run.

At 0.6 ms (T3) when the load in class 3 increases to 60%, the dual-rate solution is able to prevent traffic in class 3 from severely affecting the flow in class 1. The peak rate for class 1 was set to 80%, allowing this class to claim more of the unused bandwidth and reduce the interference from class 3: class 3 is only able to use more than it's assured rate after the other classes have exceeded their peak rates. On the other hand, the single-rate solution assured 50% of the bandwidth to class 1, which is now carrying 85% load, resulting in both class 1 and class 3 competing for the available bandwidth. Single-rate QoS shapes traffic as required only when the load distribution among the classes matches the class configurations, as shown between times T2 and T3 on the single-rate QoS plot in the figure. Otherwise, network load and interference from lower-priority classes can degrade performance. The dual-rate solution is able to provide more consistent throughput for class 1 regardless of load changes from other workloads in the network.

While maintaining high throughout for traffic in class 1 (best-effort), dual-rate QoS is also able to meet the latency targets of the allreduce jobs in the low-latency class. Figure 4 shows the MPI_Allreduce performance when using the dual-rate and single-rate QoS configurations, with both cases yielding near *Standalone* performance – where each allreduce job is ran on an idle system without background traffic. Performance is much worse in the *no-QoS* case when allreduce jobs run concurrently with the UR jobs without using separate QoS classes, i.e., traffic from all jobs share a single class. These results confirm that the dual-rate solution can also facilitate performance repeatability for low-latency traffic despite variations in network load.

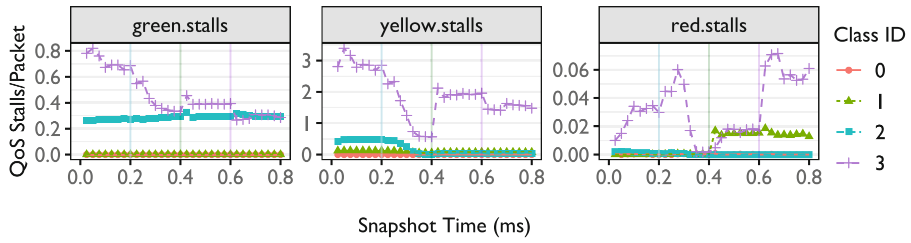

Fig. 5. Breakdown of system-wide QoS stalls per class as the traffic load changes. At 0.2 ms, load in class 2 is reduced; at 0.4 ms, load in class 1 is increased; at 0.6 ms, load in class 3 is increased. Figure 3 show the injection load and class configuration details.

Monitoring QoS Stalls to Understand Class Configuration and Behavior: Class configurations should be tuned to prevent some flows from being unintentionally delayed while simultaneously ensuring other flows are appropriately stalled. However, dynamic workloads present a challenge since they do not have static injection rates to properly guide bandwidth allocations. With dual rate limits per class, each class can be tuned to support a range of traffic loads. Additionally, monitoring how packets get stalled by classes expose how traffic is shaped and the appropriateness of the class configuration for the workloads.

As discussed in Sect. 3.2, traffic in a class is shaped by marking its packet red if the class exceeds its peak rate, yellow if only the assured rate is exceeded, or green if neither rate has been exceeded. Our solution reports three types of stalls based on these colors to expose traffic shaping:

Green Stall: The class is blocked from injecting if it has a green packet *and* a higher priority class also has a green packet ready to inject.

Yellow Stall: The class is blocked from injecting if has a yellow packet *and* either (i) a higher-priority class has a yellow packet ready to inject or (ii) any other class has a green packet ready to inject.

Red Stall: The class is blocked from injecting if has a red packet *and* either (i) another class has a green or yellow packet ready to inject or (ii) it loses to another class in round-robin arbitration.

Figure 5 reports the per-packet stall rates of each stall type over all switch-to-switch channels for the traffic described in Fig. 3. A value of 1 QoS stall/packet means that one packet was stalled for each injected packet, increasing packet latency and potentially reducing class throughput. By analyzing the type of stalls, we can determine which rate limit caused the stall and how its tuning may affect traffic shaping.

Classes 0 and 1 experience overall low QoS stall rates, confirming that their overalls flows were not being delayed. The assured rate limit of class 0 is slightly above the low injection rate of its assigned traffic, guaranteeing sufficient bandwidth to progress quickly, and the peak rate limit is high enough to accommodate momentary bursts. Class 1 is mostly stalled after its injection rate exceeds its

Table 2. Configuration for workload with mission critical traffic along with traditional HPC workload. UR jobs inject uniform random traffic.

Job	Nodes	Class	Injection rate (%)
UR-MC	**832**	**0 - mission critical**	**80**
allreduce512	256	1 - low latency	0.8
allreduce512	256	1 - low latency	0.8
UR-LL	512	1 - low latency	4
UR-BE	4160	2 - best effort	80
UR-IO	1760	3 - bulk data	90
UR-S	512	4 - scavenger	5

peak rate at 0.4 ms. Stalled packets block minimal routing paths, causing the increased use of non-minimal routing paths with extra global hops to reduce the effective available global bandwidth as mentioned earlier.

The relatively high green and yellow stall rates for class 2 confirm that its flow was regulated to limit its effect on class 1 or class 0, as intended. The yellow stall rate of class 2 is reduced when its injection rate drops to match its assured rate at 0.2 ms. The relatively low network load between 0.2 ms - 0.4 ms allowed the throughout of class 3 to be increased, signalled by the eventual reduction in its rate of yellow and red stalls. However, the initial spike in class 3 red stalls from 0.2 ms - 0.3 ms is due to the previously blocked packets being streamed into the network, causing the class to exceed its 20% peak bandwidth allocation. Overall, class 3 has the highest stall rates because it has the lowest priority and a low bandwidth allocation, and is prevented from unduly affecting more important flows.

The changes in the stall rates indicate how traffic shaping is being triggered by the composition of the network load. Having appropriately configured these HPC-oriented QoS classes to control the reallocation of bandwidth to the higher-priority flows, the stalls confirm that traffic is being shaped as intended.

4.5 Supporting Specially Defined QoS Classes

Workload configurations and requirements vary across HPC centers. While we contend that the traffic class configurations defined in Sect. 3.2 should be appropriate for most HPC workloads, centers may also need to define other classes for special workloads. When these special workloads run along the traditional HPC workloads, the QoS mechanism must satisfy the relative performance targets of both sets of workloads. We demonstrate how our QoS solution can support the creation of a site-specific *mission critical* class to carry traffic that must never be delayed by other HPC workloads. The mission critical class is assigned the highest arbitration priority and assured 100% of the system bandwidth to minimize the delay from traffic in other classes. With dual-rate QoS, other classes can be configured with peak rate limits to provide priority-ordered access to

Per-class Injection Rate

Class	Injection rate
0	80 *
1	4
2	80
3	90
4	5

Dual-rate Classes

Class	Priority	Assured %	Peak %
0	P0	100	100
1	P1	0	10
2	P2	0	50
3	P3	0	35
4	P4	0	5

Single-rate Classes

Class	Priority %	Assured %
0	P0	100
1	P1	0
2	P2	0
3	P3	0
4	P4	0

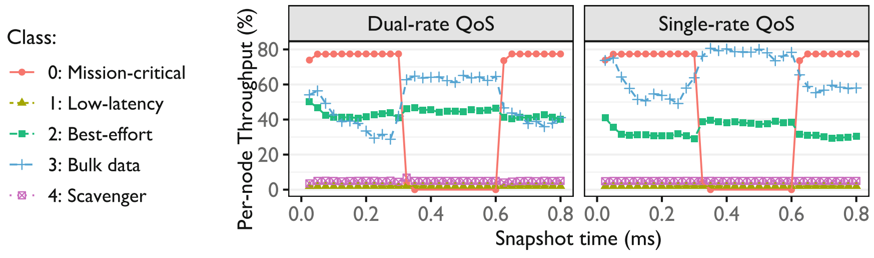

Fig. 6. Change in class throughput over time as mission critical traffic is transferred. *The mission-critical job suspends sending traffic at 0.3 ms and resumes at 0.6 ms. Workload and traffic-to-class assignments details are provided in Table 2. Traffic in class 2 (best effort) is able to sustain higher throughput with the dual-rate solution, even when the mission critical traffic is present.

unused bandwidth, respecting the relative importance of the different types of HPC traffic. However, defining the mission critical class with single-rate QoS does not allow for any regulation of HPC traffic.

We ran the workload setup in Table 2 using both single-rate and dual rate-QoS. Figure 6 shows the class configurations and throughput results for both QoS solutions. With the dual-rate solution, we ensure that class 2 has priority access to 50% of the unused bandwidth, allowing it to sustain high throughput despite heavy interference from traffic in class 3. For the single-rate QoS classes, however, the traffic in class 3 is unregulated and reduces the throughput of the higher priority class 2 traffic.

We allocate peak bandwidths to reduce the likelihood of a class being blocked when the mission critical traffic is not occupying a channel. These rates could be tuned differently depending on the goals of the system administrator. The assured and peak rate limits in our dual-rate QoS solution can be tuned independently to satisfy the performance requirements of traffic using the class while respecting the performance targets of traffic in other classes.

While it is possible to create a single-rate QoS solution using peak rates instead of assured rates, it would still be challenging to tune such a solution for dynamic workloads, as highlighted with these results. Using only a peak rate, the dual-rate QoS classes were unable to increase the allocation of bandwidth available to class 2 when the mission critical job was not injecting. Furthermore, the aggregate peak rate allocations can exceed the link bandwidth, meaning allocations are not guaranteed and lower-priority classes can be starved.

5 Discussion

5.1 Tuning Class Configurations to Match Workload Requirements

Proper QoS tuning requires accurately matching class configurations to their expected traffic load, which requires accurate knowledge of the system's expected workloads and performance targets. Our QoS solution regulates network resource allocation under varying traffic load using dual rate limits. For the assured rate limits, we recommend starting with the minimum required rate needed to attain acceptable throughput and/or latency for traffic assigned to the class. Peak rate limits can be set to the maximum expected traffic load while being mindful of the requirements of other classes. These limits can then be tuned using the QoS stall metrics as guides. Increases in yellow and red stall rates are indicators that constraints rate limit constraints are being applied since these packets get stalled only when the assured/peak rate has been exceeded. Additionally, green stall rates are indicators of priority constraints being applied priority since packets are marked green when the class has not exceeded its assured rate. The acceptable stall rates for a configuration will depend on the workload and the desired traffic shaping outcome. QoS stall rates can therefore be used to flag inappropriate traffic-to-class assignment when unexpected shaping is observed. High stall rates indicate that the traffic has exceeded its class's expected load and will experience increased packet latencies as well as potentially reducing the effective available global bandwidth.

5.2 Production Deployment

Our QoS design can be deployed on any interconnect architecture that supports network traffic classes with independent switch buffers and programmable output port arbitration, as most modern architectures do. Hence, this solution can also be used on other low-diameter topologies such as fat-tree, hyper-x, Slim Fly, and megafly, similar to other solutions [16,22]. While comparison of the different topologies is not the focus of this work, our solution will still provide more flexible control of network resources than the other solutions across the different topologies. Furthermore, being able to tune classes using stall counters will ease the integration of QoS in HPC centers.

The network drivers will provide APIs for assigning different messages to QoS classes. Communication libraries like MPI or parallel I/O libraries can be extended to utilize these APIs and automatically assign messages to different classes based on pre-defined message size/rate thresholds. Such an approach would be transparent to system users while allowing administrators to define system-wide configurations, preventing inappropriate traffic-to-class assignment. Another approach is for the user to choose class assignments for the different operations within their applications. Otherwise, a combination of the these two approaches may also be used when rolling out the QoS solution.

6 Conclusions

HPC networks often run multiple applications with differing communication patterns that compete for network resources. Because different applications may be running at any given time, network contention can result in large run-to-run performance variations for communication-sensitive applications.

Our QoS proposal classifies application traffic into one of several QoS classes, based on performance requirements, and effectively allocates resources among these classes. Each class's arbitration priority, assured bandwidth limit, and peak bandwidth limit can be tuned to match the traffic load assigned to the class. Using this solution, we can define a limited number of QoS classes – *Low-latency*, *Best-effort*, *Bulk data*, and *Scavenger* – to effectively support the diverse traffic loads on HPC systems. Our solution can ensure consistent, low-latency performance for latency-sensitive traffic, achieving near-baseline performance for MPI_Allreduce operations. It also provides the ability to maintain the high throughput required by a best-effort class, securing sufficient bandwidth for applications in order to guarantee overall system throughput.

Our solution's flexibility in provisioning multiple QoS classes with explicit, tunable assured and peak rate limits allows individual HPC sites to tailor class settings to their needs. The dual-rate limits support controlled bandwidth reallocation as traffic load changes, ensuring relative performance targets can be more effectively met in dynamic environments. Furthermore, the use of QoS stall metrics can isolate adversarial traffic-to-class assignments and help tune the configuration, deployment, and management of QoS in production. Future work will consider how to automatically assign and, and potentially reassign, traffic to classes while the workload in running. We will also investigate the interaction of our dual-rate QoS with different adaptive routing and congestion management solutions.

Acknowledgement. This work was supported by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, and by the Exascale Computing Project – learn more at https://www.exascaleproject.org/. We also gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

References

1. Brown, K.A., Jain, N., Matsuoka, S., Schulz, M., Bhatele, A.: Interference between I/O and MPI traffic on fat-tree networks. In: Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018, pp. 1–10. Association for Computing Machinery, New York, August 2018
2. Carothers, C.D., Bauer, D., Pearce, S.: ROSS: a high-performance, low memory, modular time warp system. In: Proceedings Fourteenth Workshop on Parallel and Distributed Simulation, pp. 53–60 (2000)

3. Chunduri, S., et al.: GPCNeT: designing a benchmark suite for inducing and measuring contention in HPC networks. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC 2019. Association for Computing Machinery, New York (2019)
4. Chunduri, S., Parker, S., Balaji, P., Harms, K., Kumaran, K.: Characterization of MPI usage on a production supercomputer. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. SC 2018. IEEE Press (2018)
5. Cope, J., Liu, N., Lang, S., Carns, P., Carothers, C., Ross, R.: CODES: enabling co-design of multilayer exascale storage architectures (2011)
6. Dordal, P.L.: An Introduction to Computer Networks, August 2020
7. Grant, R.E., Pedretti, K.T., Gentile, A.: Overtime: a tool for analyzing performance variation due to network interference. In: Proceedings of the 3rd Workshop on Exascale MPI, ExaMPI 2015, pp. 1–10. Association for Computing Machinery, New York, November 2015
8. Groves, T., Gu, Y., Wright, N.J.: Understanding performance variability on the aries dragonfly network. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp. 809–813, September 2017. iSSN 2168-9253
9. Hewlett Packard Enterprise: Shasta Software Workshop (2019). https://cug.org/proceedings/cug2019_proceedings/includes/files/inv113s1-file1.pdf. Accessed 19 Oct 2020
10. Hewlett Packard Enterprise: Measuring Network Performance to Better Manage IT. Technical White Paper a50002193ENW, August 2020
11. Jha, S., Brandt, J., Gentile, A., Kalbarczyk, Z., Iyer, R.: Characterizing supercomputer traffic networks through link-level analysis. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 562–570, September 2018. https://doi.org/10.1109/CLUSTER.2018.00072, iSSN: 2168-9253
12. John Thompson: Scalable Workload Models for System Simulations (2014). https://hpc.pnl.gov//modsim/2014/Presentations/Thompson.pdf. Accessed 19 Oct 2020
13. Jokanovic, A., Sancho, J.C., Labarta, J., Rodriguez, G., Minkenberg, C.: 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, June 2012. https://doi.org/10.1109/HPCC.2012.86
14. Kim, J., Dally, W.J., Scott, S., Abts, D.: Technology-driven, highly-scalable dragonfly topology. In: Proceedings - International Symposium on Computer Architecture, pp. 77–88 (2008)
15. Li, F., Niaki, A.A., Choffnes, D., Gill, P., Mislove, A.: A large-scale analysis of deployed traffic differentiation practices. In: Proceedings of the ACM Special Interest Group on Data Communication, Beijing China, pp. 130–144. ACM, August 2019
16. Mubarak, M., et al.: Evaluating quality of service traffic classes on the Megafly network. In: Weiland, M., Juckeland, G., Trinitis, C., Sadayappan, P. (eds.) ISC High Performance 2019. LNCS, vol. 11501, pp. 3–20. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20656-7_1
17. OFI Working Group: Libfabric Programmer's manual (2020). https://ofiwg.github.io/libfabric/master/man/fi_endpoint.3.html. Accessed 19 Oct 2020
18. Savoie, L., Lowenthal, D.K., de Supinski, B.R., Mohror, K., Jain, N.: Mitigating inter-job interference via process-level quality-of-service. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–5 (2019)

19. Sensi, D.D., Girolamo, S.D., McMahon, K.H., Roweth, D., Hoefler, T.: An in-depth analysis of the slingshot interconnect. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC20), November 2020
20. Smith, S.A., et al.: Mitigating inter-job interference using adaptive flow-aware routing. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 346–360, November 2018
21. Society, T.I.: A Two Rate Three Color Marker (1999). https://tools.ietf.org/html/rfc2698. Accessed 01 June 2020
22. Wilke, J., Kenny, J.: Opportunities and limitations of quality-of-service in message passing applications on adaptively routed dragonfly and fat tree networks. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER) (2020)
23. Zhang, Y., Tuncer, O., Kaplan, F., Olcoz, K., Leung, V.J., Coskun, A.K.: Level-spread: a new job allocation policy for dragonfly networks. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1123–1132 (2018)