SAND2020-9475C

Opportunities and limitations of Quality-of-Service (QoS) in Message Passing (MPI) applications on adaptively routed Dragonfly and Fat Tree networks

Jeremiah J. Wilke
Scalable Modeling and Analysis
Sandia National Laboratories
Livmore, CA, USA
jjwilke@sandia.gov

Joseph P. Kenny
Scalable Modeling and Analysis
Sandia National Laboratories
Livmore, CA, USA
jpkenny@sandia.gov

***Abstract*—Avoiding communication bottlenecks remains a critical challenge in high-performance computing (HPC) as systems grow to exascale. Numerous design possibilities exist for avoiding network congestion including topology, adaptive routing, congestion control, and quality-of-service (QoS). While network design often focuses on topological features like diameter, bisection bandwidth, and routing, efficient QoS implementations will be critical for next-generation interconnects. Even if adaptive routing makes hotspots rare events, scaling to thousands of endpoints can make "rare" events become likely. HPC workloads are dominated by tightly-coupled mathematics, making delays in a single message manifest as delays across an entire parallel job. QoS can spread traffic onto different virtual lanes (VLs), lowering the impact of network hotspots by providing priorities or bandwidth guarantees that prevent starvation of critical traffic. Two leading topology candidates, Dragonfly and Fat Tree, are often discussed in terms of routing properties and cost, but the topology can have a major impact on QoS. While Dragonfly has attractive routing flexibility and cost relative to Fat Tree, the extra routing complexity requires several VLs to avoid deadlock. Here we discuss the special challenges of Dragonfly, proposing configurations that use different routing algorithms for different service levels (SLs) to limit VL requirements. We provide simulated results showing how each QoS strategy performs on different classes of application and different workload mixes. Despite Dragonfly's desirable characteristics for adaptive routing, Fat Tree is shown to be an attractive option when QoS is considered.**

***Index Terms*—interconnects, topology, adaptive routing, message-passing (MPI), quality-of-servce (QoS), dragonfly**

I. Introduction

Network communication remains a critical challenge for extreme-scale science and engineering applications. High-performance interconnects provide both high bandwidth and low latency. Much of system interconnect design is dominated by topology and adaptive routing. High bandwidth is provided by scattering traffic along idle non-minimal paths, increasing the available bandwidth between two endpoints. Low latency is similarly provided by avoiding long queuing delays along congested paths. Network delays can harm performance even in latency-sensitive applications which only exchange small amounts of data and are not generally considered "communication-bound." Tightly-coupled simulations that are common in science and engineering applications often only proceed as fast as the slowest process [1]. Intermittent network delays in a small subset of processes can quickly propagate to delays across the entire simulation [2].

Adaptive routing is not the only tool for mitigating long queuing delays. Quality-of-Service (QoS) has been available in Ethernet [3] and Infiniband [4] networks and is a prominent feature of the new Cray Slingshot interconnects [5]. QoS can assign priority or bandwidth guarantees to different service levels, providing low latency to traffic even along highly-congested network paths. While QoS cannot generally increase total network throughput (this falls to adaptive routing), QoS can prevent starvation of latency-sensitive traffic. QoS and adaptive routing are therefore highly complementary in optimizing latency for large HPC workloads.

Message-passing (MPI) is the dominant network programming model for HPC applications [6] (although the lessons here are equally applicable to non-MPI frameworks, e.g. GASNet [7]). Within a single application, MPI has several traffic types than can benefit from prioritization. Collectives exchange data amongst all processes in a set, creating large synchronization points. Collectives with many processes and small data (e.g. the ubiquitous all reduce of a single element) can be prioritized, guaranteeing low latency regardless of current network contention. For both collective and point-to-point traffic, data exchange protocols also involve a mix of small control messages and bulk RDMA transfers. Again, these small control messages could be given higher priority with lower priority given to bulk transfers.

In mixed workloads with multiple MPI applications, a network congested from one application might temporarily starve traffic from another application. QoS also can mitigate such network bottlenecks through bandwidth guarantees rather than priorities. Even if queues are completely full for one

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

service level, the other application can still make progress with 50% of the bandwidth.

When designing the network topology, features like hardware cost, routing, diameter, and bisection bandwidth are usually the main focus. Dragonfly, for example, has a low diameter and excellent path diversity for adaptive routing [8] while Fat Tree has higher diameter and requires extra core or director switches. Topology and routing, however, can have an enormous impact on the QoS implementation. The most effective Dragonfly routing schemes allow mis-routing at each hop rather than just selecting routes at the source [9]. This adaptive routing can require several virtual channels (VCs) to avoid deadlock cycles in the network [8], [10]. These virtual channels must be mapped to independent virtual lanes (VLs) provided by the network switches. Service levels must also be mapped onto virtual lanes (VLs), with each VL having different priorities and bandwidth shares. Assigning a unique VL to each combination of QoS and deadlock VC could quickly exceed the number of VLs available in the system. While Dragonfly has more complex deadlock avoidance rules, Fat Tree may only require a single VC. Despite Fat Tree's larger diameter and extra switches, QoS implementation can shift the cost/benefit tradeoffs of the topology.

The interaction of MPI implementation, routing and topology presents a complex set of choices for how to leverage QoS. Here we map out this design space, demonstrating the cost/benefit tradeoffs of different QoS approaches for different topologies. Although we consider only Dragonfly and Fat Tree, these can be considered extreme cases in terms of cost and number of VCs required. The lessons should therefore be transferrable to other topologies like Dragonfly+ [11], HyperX [12] or SlimFly [13] We begin by proposing different QoS strategies that could be employed by existing MPI implementations [6], [14], [15]. We examine these strategies across a set of single application and mixed workload scenarios. Our results illustrate the challenge of selecting a "universally beneficial" QoS strategy that can prioritize traffic from one application without harming the performance of others. We show topology-specific requirements for these QoS strategies. In particular, we show for Dragonfly that if different routing algorithms are used for difference service levels then the number of VLs required can be greatly reduced. Fat Tree is shown to be an attractive option when QoS is considered, even when tapering to reduce the total cost.

II. Related Work

Several QoS studies have considered inter-job interference, separating applications into different service levels [16], [17] to reduce contention. Intra-job QoS strategies prioritizing small control traffic were considered in [18], along with more coarse-grained inter-job traffic differentiation. These studies inform the QoS strategies in the current work, but do not consider the broader implications of topology and routing on QoS. A recent study on Megafly considered different prioritizing collectives over point-to-point transfers [19], but did not consider more fine-grained prioritization of control messages within the point-to-point protocols. Megafly was considered as a Dragonfly alternative requiring fewer VCs, which could better support QoS. We instead propose mixed routing methods for Dragonfly that still enable QoS within a limited number of virtual lanes (VLs). We further consider tapered Fat Tree as a cost-effective alternative with even fewer VCs required than Megafly.

QoS has been available on Infiniband networks [3], [4], which support two priorities (low and high) and bandwidth weights for different service levels. Cray Slingshot, although not widely available yet in Top 500 systems, will support QoS [5]. Ethernet (and its RDMA interface RoCE [**?**]) have long supported flexible QoS configurations [3], although it is less common in HPC systems. Blue Gene systems provided a collective tree network, which was implemented using virtual lanes within the main torus topology [20].

Beyond QoS, several studies have examined runtime variability or inter-job interference on production systems [21], [22] or via simulation [2]. These have considered both regular application traffic and I/O traffic [23]. Subramoni et al. considered using virtual lanes, without service differentiation, to reduce head-of-line (HoL) blocking in networks [24]. Samuel et al. used routing keys for application-specific routing to avoid contention, which shares some similarities with QoS [25]. Software-defined networks (SDN) with application-specific routing tables have been considered – notably for Dragonfly [26] – which again avoids inter-job contention.

We do not perform a detailed survey of Dragonfly routing algorithms. Our Dragonfly routing allows misrouting at each step similar to On-the-Fly Adaptive Routing (OFAR) [27], although we do not use escape networks. More detailed treatments of adaptive routing on Dragonfly can be found in [27], [28], [29], [30]. The original source-based UGAL algorithm [8] often fails to detect congestion at the source, which has led to proposals for piggybacking congestion information on packets [9].

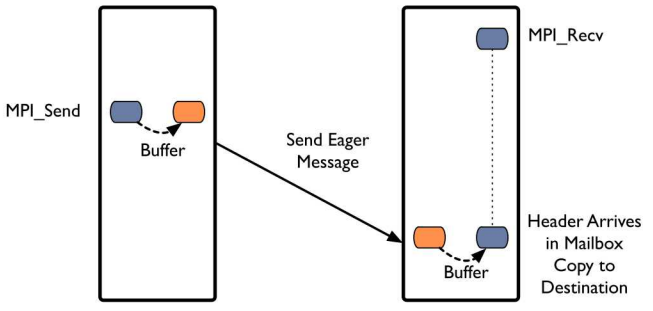


Fig. 1: Mailbox (eager) protocol that can asynchronously send to a pre-allocated buffer on the receiver. This protocol is generally reserved for small messages.

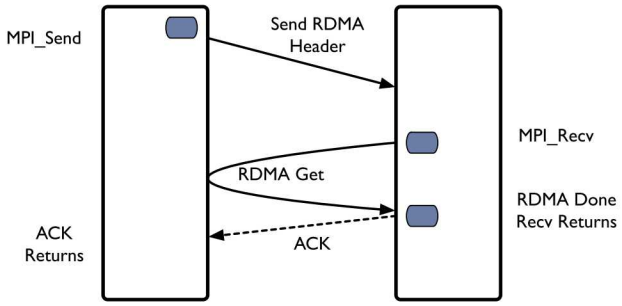


Fig. 2: Rendezvous RDMA protocol for zero-copy transfer from send buffer to receive buffer. A control message exchanging RDMA metadata is sent to the receiver, which then issues an RDMA get. A control message acknowledging completion is sent back to the sender.

III. QoS and MPI

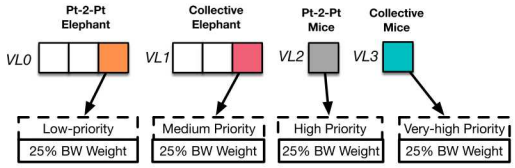
A. Message-Passing Implementation

There are numerous traffic types in the MPI implementations that could be prioritized. At a coarse-grained level, collective traffic could be prioritized over point-to-point traffic – or at least given its own virtual lane (VL) with minimum bandwidth weight. More fine-grained QoS could be used within each point-to-point send. Depending on message size, different protocols for exchanging data are used. For small messages (Figure 1), data can be first copied into a send buffer and then sent into a pre-allocated "mailbox" on the receiver, making the transfer asynchronous and only requiring a single network trip. This "mice" traffic can likely be prioritized without harming throughput of larger messages. For larger messages, a rendezvous protocol is used to enable zero-copy transfers without intermediate buffers (Figure 2). This requires first sending an RDMA header followed by a bulk RDMA transfer and then ACK to the sender. The initial header and ACK are high-priority mice traffic while the bulk RDMA get can be a low-priority "elephant" flow. This fine-grained use of QoS also applies to individual messages within collectives, which are often implemented as a sequence of point-to-point sends [31].

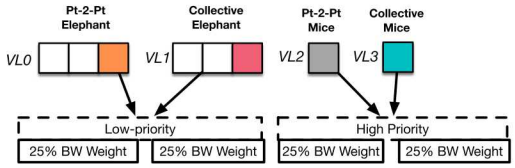
B. Quality-of-Service (QoS)

Quality-of-Service (QoS) allows network packets to be assigned to different traffic classes (TCs) or service levels (SLs). These service lvels can be mapped to different priorities or bandwidth weights within a priority. Small latency-sensitive traffic (mice traffic), for example, can be given high-priority while large bandwidth-sensitive flows (elephant traffic) can be given lower priority to prevent it starving other traffic. Service levels with the same priority can each be assigned a bandwidth weight, e.g. guaranteeing each SL 50% of the available bandwidth, which again prevents starvation of other application traffic. Infiniband provides two priorities (low and high) and implements bandwidth weights with weighted round robin (WRR) arbitration [32].

(a) QoS = None: All traffic uses a single FIFO on all ports



(b) QoS = Split-Priority: Mice has priority over elephant, collective priority over pt-2-pt. Each VL is assigned 25% bandwidth weight.



(c) QoS = Bandwidth-Weight: Mice has priority over elephant. Collective and pt-2-pt have same priority, but collectives are assigned 25% bandwidth guarantee.

Fig. 3: Different QoS strategies for prioritizing mice/elephant or collective/pt-2-pt flows.

Figure 3 shows the main QoS strategies considered in the current work. The first (QoS = None) puts all traffic into the same SL, which effectively makes all buffer queues a single FIFO. QoS=Split-Priority distinguishes point-to-point, collective, and control traffic using 4 different priorities. RDMA headers and ACKs (mice traffic) are given high-priority while large data flows (elephant traffic) are given lower-priority. Within each traffic type (mice and elephant), collectives are given higher priority than point-to-point. The third strategy (QoS=Bandwidth-Weight) assumes hardware only supports 2 different priorities. Mice traffic is again given higher priority than elephant flows. While collectives and point-to-point have the same priority, they are sent on different virtual lanes. The collective VL is given a minimum bandwidth weight to avoid starvation. Not shown in Figure 3, we also consider QoS=Isolate which gives each application its own VL. All applications have the same priority, but each is given a bandwidth weight proportional to its share of the machine. All messages within an application - both small control messages and large data flows - use the same priority and same VL.

IV. Topologies

A. Dragonfly

The Dragonfly topology [8] arranges switches into all-to-all connected groups (Figure 4). At least one inter-group link connects each pair of groups. Inter-group pairs are selected based on a fixed pattern like circulant or palm tree [28]. Intra-group flows require only a single hop while inter-group flows will require two or three hops for minimal routing. Minimal routes can quickly become congested. Adaptive routing or

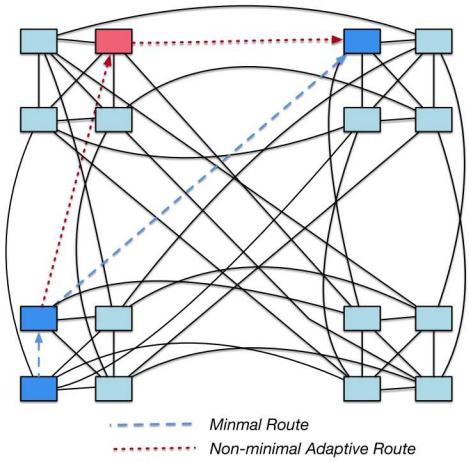


Fig. 4: Depiction of Dragonfly topology and adaptive routing. Topology is composed of 8-port switches in 4 groups of 4. Each switch has 2 connected endpoints (omitted for clarity).

misrouting schemes are can be source-based (misroute at initial switch) or progressive (misrouting allowed on each hop) [9]. UGAL is the most common source-based [8] routing while OFAR [27] allows more flexible misrouting. Minimal routing requires two virtual channels (VC) to avoid deadlock cycles between intra- and inter-group traffic. UGAL routing requires a third virtual channel to avoid deadlock cycles from non-minimal inter-group traffic. More flexible mis-routing may require as many VCs as the maximum number of hops (usually 5-7). OFAR tries to reduce VC requirements with escape networks, which adds extra complexity. In the current work we consider only Progressive Adaptive Routing (PAR), which allows misrouting at each step without escape networks.

Despite the overloaded terms in the literature, we refer to virtual lanes (VLs) on the switches that provide independent buffers and flow control while virtual channels (VCs) are abstract channels in a routing algorithm. To implement a particular routing algorithm, a VC must be mapped to a particular VL in the switch hardware. This, however, overlaps with traffic classes (TCs) or service levels (SLs) in QoS. Each SL is assigned a particular priority and bandwidth weight, which then also requires mapping SLs to independent VLs on the switches. Combining deadlock channels and service levels now requires mapping (VC,SL) pairs to VLs. Each (VC,SL) pair should ideally be independent from all other VCs to avoid deadlock and all other SLs to ensure priority/bandwidth. This potentially requires a full VCxSL cross-product. For just 4 SLs with adaptive routing, this could be as many as 24 VLs!

We test two Dragonfly implementations, each assuming 4 SLs. Dragonfly-PAR uses adaptive routing for all traffic, and we assume a theoretical switch hardware supporting a large number of VLs. Dragonfly-MIN/PAR uses progressive adaptive routing for low-priority SLs 0 and 1 and minimal routing for high-priority SLs 2,3, which limits the number of required VLs to 16.

Topology	No. Switches	Port Config
FatTree	640	32 up/32 down
FatTree-TAPER	480	27 up/36 down
Dragonfly-PAR	512	16 injection, 32 intra-group, 16 inter-group
Dragonfly-MIN/PAR	512	16 injection, 32 intra-group, 16 inter-group

TABLE I: Topology configurations

B. Fat Tree

Fat Tree arranges switches into rows of leaf, aggregation, and core switches (Figure 5). In the diagram, each leaf and aggregation switch has 4up/4down ports connecting the rows. Tapered configurations are possible, with more down than up ports and fewer aggregation/core switches [33]. Tapered designs are feasible when there is far more traffic on leaf ports than core ports.

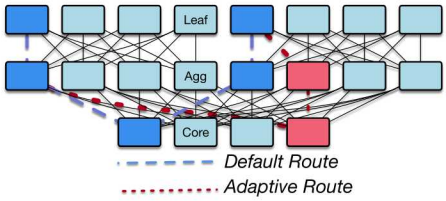


Fig. 5: Depiction of Fat Tree topology and adaptive routing. Topology is arranged in leaf, aggregation, and core switches. Each leaf switch has 4 connected endpoints (omitted for clarity).

Certain leaf and aggregation switches comprise "subtrees" within the topology. Minimal routes are 2-4 hops, depending on whether they are intra- or inter-subtree. While minimal routes are longer than Dragonfly, Fat Tree has numerous "minimal" routes between endpoints. We therefore do not refer to minimal and non-minimal for Fat Tree, but instead "default" and "adaptive" routes. If all traffic follows a simple up-down pattern (never changing direction), there are no deadlock cycles even with adaptive routing. Fat Tree therefore only requires a single VC for deadlock avoidance.

Fixed routing schemes like D-Mod-K are often used with Fat Tree [34]. However, we assume Fat Tree has equivalent adaptive routing capabilities as Dragonfly. We therefore implement a "progressive adaptive" Fat Tree that adaptively routes on each hop. Traffic still only routes up/down and requires no extra VCs for deadlock avoidance, which allows a direct correspondence between SL and VL. Recall that we test a QoS=Isolate strategy that assigns each application a unique SL and VL. Although we assume a Dragonfly-PAR with a huge number of VLs for a theoretical comparison, it may only be practical to implement QoS=Isolate on a Fat Tree.

V. Methodology

A. Topologies

We assume 64-port switches and a total system size of 8192 nodes (endpoints). Each Dragonfly implementation has 512

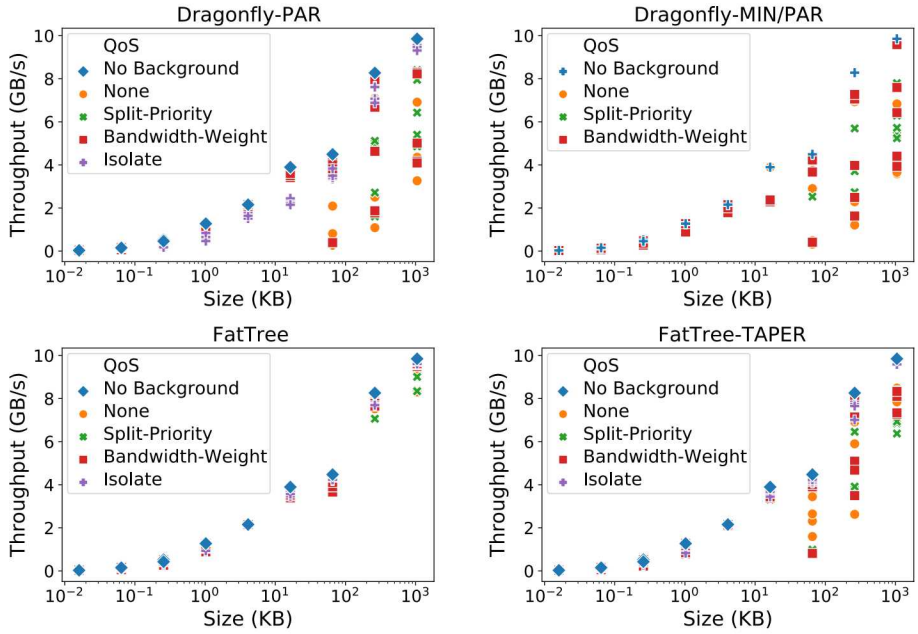


Fig. 6: Ping-Pong throughput for varying message sizes for different QoS strategies with Halo3D background.

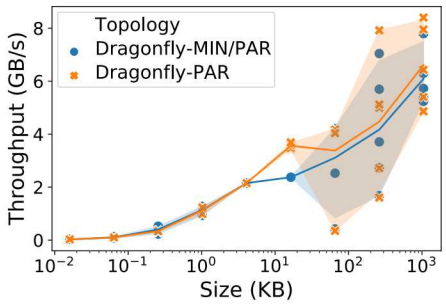


Fig. 7: MPI Ping-Pong throughput varying sizes on Dragonfly topologies for QoS=Priority with Halo3D background.

switches and concentration 16 (Table I). The 512 switches are organized into 16 groups of size 32. At this scale, every switch has at least one connection to every other group and the Dragonfly actually has a minimal diameter of 2. The full Fat Tree has 640 total switches. 256 leaf switches each have concentration 32, which connect to 256 aggregation and 128 core switches. A tapered Fat Tree is also considered for a fairer comparison to Dragonfly with fewer switches. The tapered configuration has 228 leaf switches with concentration 36, which tapers to 171 aggregation switches and 81 core switches.

Workloads were run either as a foreground application or background traffic. Each workload is selected to cover a range of use cases: bandwidth vs. latency sensitive and collective vs. point-to-point traffic. Background traffic was spawned on the system to create a fragmented allocation [22] and run for a warmup period. The foreground application was then launched on the remaining nodes in the system. While the job launcher exploits as much locality as possible in placing individual ranks on the system, the foreground ranks will still be scattered on whatever fragmented allocation is available. Foreground apps were run on either 1024, 2048, or 4096 nodes of the 8192 total. Each application assumes MPI+X parallelism with one MPI rank per node. For each run, a total of 4-7 background apps of size 1024 were launched on the system.

1) Halo3D: Halo3D is a bandwidth-intensive application dominated by point-to-point traffic between neighbor processes. Large halo regions (2MB) are exchanged along shared faces in a 3D grid.

2) Fast Fourier Transform (FFT): FFT is a proxy for a bandwidth-intensive collective. The FFT application performs all-to-all exchanges amongst processors in the same "row" in a 3D grid. Eeach rank in, e.g., row $(X, Y, 0), ..., (X, Y, N)$ forms a subcommunicator on which all-to-all communication is performed. Each MPI rank contributes 32-64 KB to each all-to-all in the current study.

3) Sweep3D: Sweep3D is a proxy for a more latency-sensitive point-to-point pattern. Sweep3D models wavefront propagation across a 3D space. Like Halo3D, data is exchanged between neighboring processes. Processes cannot send to neighbors, though, until they receive data from their predecessor wavefront. Sweep3D sends much smaller messages (10-40KB) than Halo3D, which makes Sweep3D far more latency-sensitive than Halo3D.

4) Global Allreduce: The allreduce performs a global sum of a single element. Each rank exchanges data with its partners in a spanning tree. Performance is entirely latency-bound.

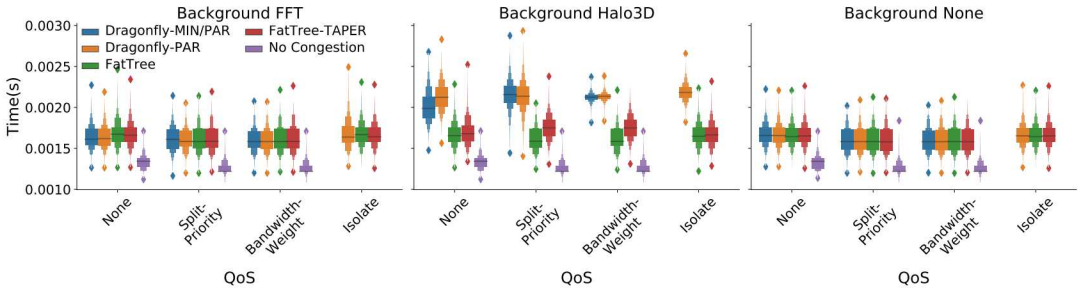
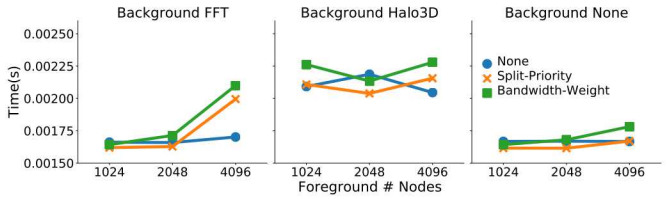


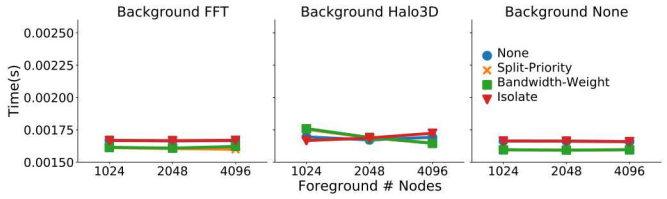
Fig. 8: Distribution of iteration times for Halo3D foreground app running on 1024 nodes on system with 8192 nodes with different background traffic.

B. Simulation

All simulations were performed using the Structural Simulation Toolkit 10.0 release [35] . Application codes were "skeletonized" to communication-only benchmarks using the Clang-based auto-skeletonzier [36] in the SST/macro element library [37]. Switch contention models are implemented in the SNAPPR component of SST/macro (*S*imulator *N*etwork for *A*daptive *P*riority *P*acket *R*outing). Background apps were started first and run continuously for an unlimited number of iterations. Foreground apps were started after a brief warmup period. Simulations were terminated when the foreground application completed a fixed number of iterations.



(a) Halo3D/Dragonfly-MIN/PAR



(b) Halo3D/FatTree-TAPER

Fig. 9: Performance scaling of Halo3D foreground with different QoS strategies and backgrounds.

VI. Results and Discussion

A. MPI Ping-Pong

Before proceeding into application use cases, it is instructive to consider a ping-pong benchmark. Figure 6 shows observed throughputs for the different QoS strategies on the Dragonfly-PAR network. The spread over all repeated iterations is shown. QoS=None, QoS=SplitPriority and QoS=Bandwidth-Weight all show a high degree of scatter and similar performance. By far the highest and most stable throughput is QoS=Isolate which assigns all ping-pong traffic to a unique VL with bandwidth guarantee.

A more subtle (and critical point to later discussion of Dragonfly) is seen comparing Dragonfly-PAR to Dragonfly-MIN/PAR. Figure 7 shows throughputs for QoS=Split-Priority, which now shows the average performance and 95% confidence interval ranges. Despite using minimal routing for small traffic, Dragonfly-MIN/PAR provides essentially the same performance as Dragonfly-PAR for most cases. There is one notable region around 15KB messages, however, where Dragonfly-PAR provides consistently higher throughputs. Despite RDMA headers having high priority in the rendezvous protocol (Figure 2), adaptive routing still seems to provide a benefit. High-priority messages can still experience some delay on congested minimal paths. The weighted round-robin arbitrator cannot immediately respond to a high-priority request. Even if a high-priority packet is the very next selected, it still has to wait for the the already arbitrated flits or packets to send. As such, there may still be situations where taking a quieter, non-minimal path can have lower latency.

B. Halo3D

Halo3D is primarily bandwidth-bound, exchanging large (several MB) messages between neighboring processes. Rather than a single median or average, we plot the complete distribution of iteration times across all MPI ranks in Figure 8. Since tail latency is a critical performance driver, a QoS strategy should ideally reduce not only the median/mean iteration time for all MPI ranks, but also the outliers. Figure 8 includes several important baselines for comparison.

- Performance with no service levels (QoS = None). All traffic fills a single FIFO queue on each port.
- Performance with no background traffic. All contention is within the application itself.
- Performance with no contention. Queue delays are ignored in network switches, simulating "ideal" performance where each packet sees a quiet network (NIC queue delays are still included).

The box plots show quantiles out to 99% while the remaining outliers are shown as points. Across topologies, Dragonfly-MIN/PAR is competitive with Dragonfly-PAR, often showing

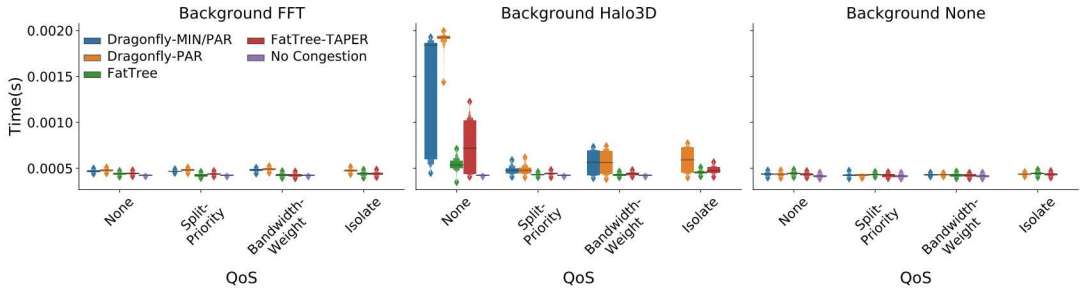
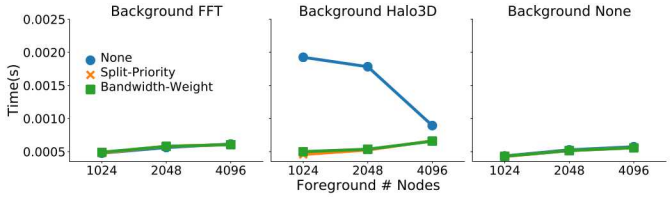
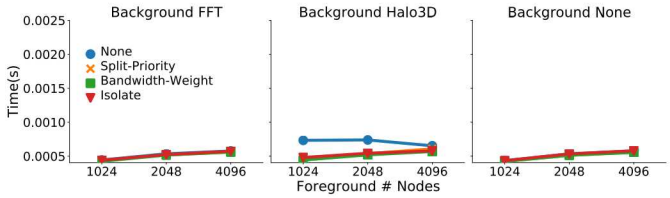


Fig. 10: Distribution of iteration times for FFT foreground app running on 1024 nodes with different background traffic.



(a) FFT/Dragonfly-MIN/PAR



(b) FFT/FatTree-TAPER

Fig. 11: Performance scaling of FFT foreground with different QoS strategies and backgrounds

slightly improved performance. Fat Tree generally outperforms Dragonfly, even in the tapered configuration whether or not service levels are used. Although the FFT benchmark sends some large messages, Halo3D seems relatively insensitive to FFT background traffic even with QoS=Split-Priority that prioritizes collective traffic – consistent with previous results [19]. Halo3D foreground performance degrades with Halo3D background since both bandwidth-intensive applications contend for limited bandwidth.

Prioritizing control traffic provides little obvious performance benefit for Dragonfly - with or without background traffic. Prioritizing control traffic has the effect of increasing injection rate, cutting down the delay between MPI_Send being called and the RDMA transfer beginning. For an already bandwidth-limited application, this provides no benefit. In fact, this could accomplish a "reverse congestion control" by allowing RDMA transfers to begin faster. For Fat Tree (with more switches and more bandwidth), prioritizing control traffic does reduce median iteration times, but the outliers still remain. The QoS=Isolate strategy gives each application its own service level with bandwidth guarantee $\frac{\#nodes\ job}{\#nodes\ system}$. Giving bandwidth-weights and removing HoL-blocking with separate VLs provides little benefit, though, since performance is entirely limited by aggregate bandwidth.

Scaling of the approach is illustrated in Figure 9 for a subset of the topologies. The results remain fairly consistent as the foreground app is increased to 4096 nodes. For Dragonfly, QoS becomes less helpful as the foreground app occupies a larger fraction of the system and the app begins to experience more intra-job contention and less inter-job contention.

C. FFT

FFT performs subcommunicator all-to-all collectives, exchanging medium-sized (10-100KB) messages. The FFT benchmark shows obvious contention effects when contending with Halo3D background (Figure 10). Self-interference with no background or interference from other FFT jobs seems to have a limited impact on performance. For Fat Tree, performance even approaches the no congestion baseline. Fat Tree again outperforms Dragonfly, even for tapered configurations. Dragonfly-MIN/PAR is again competitive with and often performs better than Dragonfly-PAR. Prioritizing control traffic shows no appreciable benefit when FFT runs with no background or other FFT jobs as background.

QoS=Split-Priority shows the largest improvement for FFT, as expected, since collective traffic is prioritized. However, the Bandwidth-Weight strategy also drastically reduces FFT outliers for all topologies. Large collective messages are only given bandwidth weight guarantees, not special priority. Likewise, for QoS=Isolate collectives do not have increased priority but guaranteed bandwidth share. While prioritizing collective traffic obviously reduces FFT iteration times, the most important reduction comes just from minimal bandwidth weighting and avoiding HoL-blocking.

Scaling for FFT shows a similar story to Halo3D scaling in Figure 11. Performance remains stable scaling to 4096 nodes while differences in QoS become less significant as the foreground app occupies a larger fraction of the system.

D. Global Allreduce

We show performance of a global all-reduce collective for a single element. Again, only bandwidth-intensive Halo3D traffic provides significant network contention. While the vast majority of observed iteration times are relatively low, the outliers are an order of magnitude slower. The Fat Tree topologies actually manage to keep contention low enough to

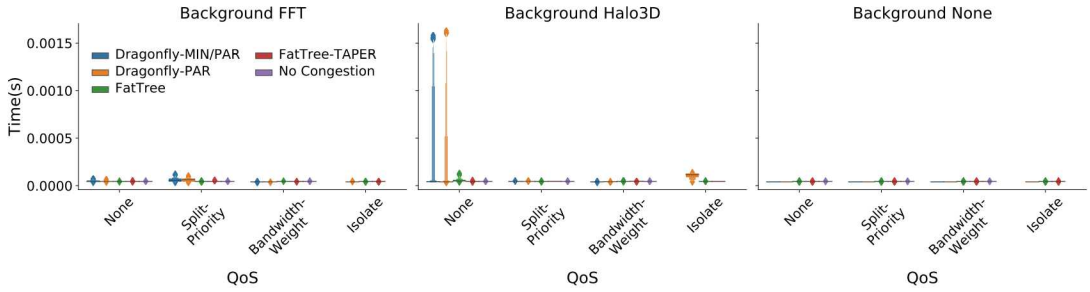


Fig. 12: Distribution of iteration times for Allreduce foreground app running on 1024 nodes with different background traffic

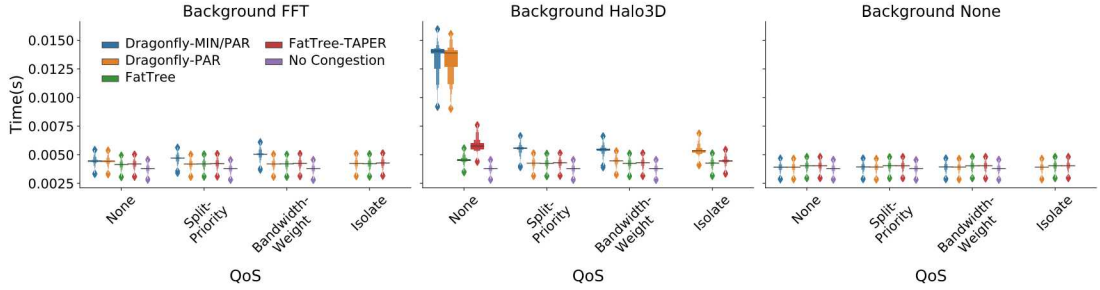


Fig. 13: Distribution of iteration times for Sweep3D foreground app running on 1024 nodes with different background traffic

completely avoids these outliers. Dragonfly, however, produces distant outliers. All QoS strategies remove the outliers, with QoS=Split-Priority (which prioritizes collectives) providing the best performance.

E. Sweep3D

Sweep3D sends small and medium-sized messages (10-50KB) between neighbor processes. Unlike Halo3D, Sweep3D simulates a propagation from one corner of a domain across the whole domain. While in Halo3D all processes can begin immediately, in Sweep3D processes can only being when their preceeding wavefront is complete. This makes Sweep3D much more latency-sensitive than Halo3D. Sweep3D shows little sensitivity to FFT background traffic. Halo3D background, however, has a major impact on Sweep3D performance. While contention in the full Fat Tree remains relatively low, both Dragonfly and tapered Fat Trees show serious contention slowdowns without QoS.

Interestingly, all QoS strategies largely reduce the performance degradation. For QoS=Isolate, this is achieved by giving Sweep3D minimal bandwidth weights and its own VL. For QoS=Split-Priority, in contrast, this achieved only by giving small messages and control traffic priority. In all cases so far, Dragonfly-MIN/PAR has matched or even outperformed the Dragonfly-PAR routing. For Sweep3D with QoS=Split-Priority or QoS=Bandwidth-Weight, Dragonfly-MIN/PAR now shows larger iteration times. Whereas Halo3D sends large messages (several MB), Sweep3D sends medium messages in the 10KB-40KB range. Recalling Section VI-A, there were particular cases where non-minimal routing of small messages improved throughput.

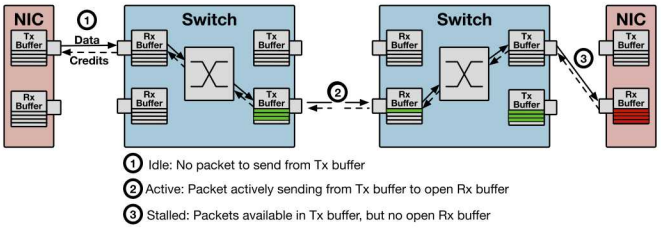


Fig. 14: Illustration of three different states of network ports: active, idle, and stalled. Simulator counts total time (number of cycles) spent in each state for every port.

F. Performance Counters

Fat Tree – even the tapered configuration – performed surprisingly well compared to Dragonfly for all use cases considered. We examine detailed performance counters to better understand where the performance gain comes from. Figure 14 illustrates the performance counters we consider. Every port in the system is either 1) idle (nothing to send), 2) actively sending, or 3) stalled (packets queued, but waiting on credits). The simulator logs the number of cycles spent by each port in a given state. Stalled cycles indicate ports overwhelmed by contention. Adaptive routing ideally reduces the number of idle cycles (traffic should be distributed to underutilized ports). Ports that spend too much time active indicate network hotspots. Aggregate activity is also lowest when all packets

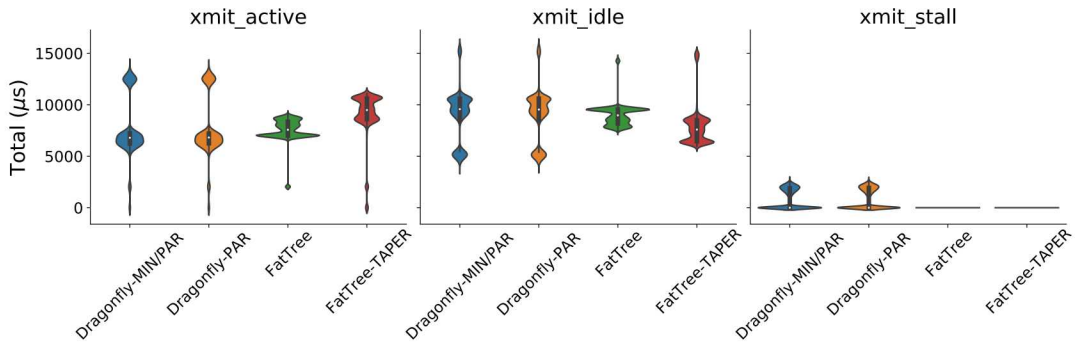


Fig. 15: Distribution plots (violin histograms) showing time spent by ports in active, idle, or stalled states in each topology. Lower is better for each performance counter. Each violin is a rotated histogram binning all ports in the system by time spent in a given state. Result shows Halo3D foreground with Halo3D background for QoS=Split-Priority.

route minimally since mis-routing increases the number of hops required.

Figure 15 shows vertical histograms (violins) for all the ports in the network. Distribution width indicates the number of ports in a given bin (time spent in a given state). Dragonfly sees a bimodal distribution of active ports. The upper tail corresponds to hotspot ports more active than the rest of the network. The Fat Tree topologies – even the tapered version – avoid this "hotspot tail" in the distribution. Although stalls occur only a small fraction of the time in Dragonfly, they are almost completely absent in the Fat Tree topologies.

We can consider again the routing diagrams in Figures 4 and 5. When Dragonfly routers detect congestion, they must choose a non-minimal path to avoid the hotspot. Selecting which non-minimal path to take is non-trivial as the router may have incomplete congestion information. Misrouting also increases the path length, which can make contention worse. In fact, for uniform random traffic, minimal routing is actually the best possible scheme. Fat Tree can adaptively route along many equivalent minimal paths. Even oblivious, round-robin schemes can distribute load across all these minimal paths. Selecting adaptive routes in Fat Tree is therefore much simpler and does not increase the path length.

We emphasize here that the results only apply to our particular implementation of Dragonfly. More complex schemes involving, e.g. congestion piggy-backing could potentially improve Dragonfly's adaptive routing [9]. Our study instead shows that 1) Fat Tree enables simple QoS and routing schemes with excellent performance and 2) QoS greatly improves Dragonfly performance over its baseline.

VII. Conclusions

Quality-of-Service (QoS) can be a powerful mechanism for preventing starvation of certain application traffic on highly-contended networks. QoS, however, has important interactions with topology and routing. Dragonfly and Fat Tree, considered here, represent two extremes of interconnects common with high-radix switches. Dragonfly has low diameter and excellent adaptive routing potential, but requires extra virtual lanes. Fat Tree has larger minimal diameter, but with much simpler routing rules and fewer virtual lanes required. Because QoS is provided by differentiating priority and bandwidth share across different virtual lanes, QoS becomes more difficult on Dragonfly. We explored a scheme selectively mixing adaptive and minimal routing (Dragonfly-MIN/PAR) that shows generally good performance, allowing multiple different service levels for Dragonfly even with limited virtual lanes. While QoS strategies that prioritize different types of MPI traffic can improve performance, the most consistent benefit is seen with QoS=Isolate where every application receives a dedicated virtual lane and bandwidth share. Even with the reduced VL requirements for Dragonfly-MIN/PAR, this QoS=Isolate strategy likely cannot provide enough unique service levels on Dragonfly. Such a strategy is likely only feasible on a Fat Tree, which can implement adaptive routing on a single virtual lane.

Fat Tree provides consistently better performance, even when QoS and background traffic are not considered. This performance benefit is seen even for tapered trees with fewer total switches than Dragonfly. Given that a more complete exploration of Dragonfly routing algorithms is not done here this should not be considered definitive evidence that tapered Fat Tree is better than Dragonfly – only that tapered Fat Tree should be considered in future work as a simpler and competitive alternative.

Acknowledgment

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. Specials thanks to Drew Lewis and Scott Hemmert for useful discussion.

References

[1] F. Petrini, D. J. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the

8,192 Processors of ASCI Q," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, 2003, p. 55.

[2] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Watch Out for the Bully! Job Interference Study on Dragonfly Network," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 750–760.

[3] S. Reinemo, T. Skeie, T. Sodring, O. Lysne, and O. Trudbakken, "An overview of QoS capabilities in infiniband, advanced switching interconnect, and ethernet," *IEEE Commun. Magazine*, vol. 44, pp. 32–38, 2006.

[4] J. Pelissier, *Providing quality of service over Infiniband architecture fabrics*, 2000.

[5] (2019) SLINGSHOT: THE INTERCONNECT FOR THE EXASCALE ERA. [Online]. Available: https://www.cray.com/sites/default/files/Slingshot-The-Interconnect-for-the-Exascale-Era.pdf

[6] W. Gropp, "MPICH2: A New Start for MPI Implementations," in *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2002, p. 7.

[7] U. Consortium, "GASNet Specification, Lawrence Berkeley National Lab Tech Report LBNL-6623E," 2013.

[8] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology," in *ISCA '08: International Symposium on Computer Architecture*, 2008, pp. 77–88.

[9] N. Jiang, J. Kim, and W. J. Dally, "Indirect Adaptive Routing on Large Scale Interconnection Networks," in *ISCA 2009: 36th Annual International Symposium on Computer Architecture*, 2009, pp. 220–231.

[10] W. J. Dally, "Virtual-Channel Flow-Control," *IEEE Transactions on Parallel Distrib. Syst.*, vol. 3, pp. 194–205, 1992.

[11] A. Shpiner, Z. Haramaty, S. Eliad, V. Zdornov, B. Gafni, and E. Zahavi, *Dragonfly+: Low Cost Topology for Scaling Datacenters*, 2017.

[12] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–11.

[13] M. Besta and T. Hoefler, "Slim Fly: A Cost Effective Low-Diameter Network Topology," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 348–359.

[14] R. L. Graham, T. S. Woodall, and J. M. Squyres, "Open MPI: A Flexible High Performance MPI," 2006, pp. 228–239.

[15] J. Liu, J. Wu, and D. K. Panda, "High performance RDMA-based MPI implementation over infiniBand," *Int. J. Parallel Program.*, vol. 32, pp. 167–198, 2004.

[16] L. Savoie, D. K. Lowenthal, B. R. D. Supinski, K. Mohror, and N. Jain, "Mitigating Inter-Job Interference via Process-Level Quality-of-Service," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–5.

[17] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, G. Rodriguez, J. Labarta, and C. Minkenberg, "On-the-Fly Adaptive Routing in High-Radix Hierarchical Networks," in *Parallel Processing (ICPP), 2012 41st International Conference on*, 2012, pp. 279–288.

[18] H. Subramoni, P. Lai, and D. K. Panda, "Designing QoS Aware MPI for InfiniBand - Techinical Report."

[19] M. Mubarak, N. McGlohon, M. Musleh, E. Borch, R. Ross, R. Huggahalli, S. Chunduri, S. Parker, C. Carothers, and K. Kumaran, "Evaluating Quality of Service Traffic Classes on the Megafly Network," 2019.

[20] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels, "MPI Collective Communications on The Blue Gene/P Supercomputer: Algorithms and Optimizations," in *2009 17th IEEE Symposium on High Performance Interconnects*, 2009, pp. 63–72.

[21] S. Chunduri, K. Harms, S. Parker, V. Morozov, S. Oshin, N. Cherukuri, and K. Kumaran, "Run-to-run variability on Xeon Phi based cray XC systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, p. Article 52.

[22] A. Jokanovic, J. C. Sancho, J. Labarta, G. Rodriguez, and C. Minkenberg, "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*, 2012, pp. 598–607.

[23] M. Mubarak, P. Carns, J. Jenkins, J. K. Li, N. Jain, S. Snyder, R. Ross, C. D. Carothers, A. Bhatele, and K. Ma, "Quantifying I/O and Communication Traffic Interference on Dragonfly Networks Equipped with Burst Buffers," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, 2017, pp. 204–215.

[24] H. Subramoni, P. Lai, S. Sur, and D. K. Panda, "Improving Application Performance and Predictability Using Multiple Virtual Lanes in Modern Multi-core InfiniBand Clusters," in *2010 39th International Conference on Parallel Processing*, 2010, pp. 462–471.

[25] A. Samuel, E. Zahavi, and I. Keslassy, "Routing Keys," in *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*, 2017, pp. 9–16.

[26] P. Faizian, M. A. Mollah, Z. Tong, X. Yuan, and M. Lang, "A comparative study of SDN and adaptive routing on dragonfly networks," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–11.

[27] M. Garcia, E. Vallejo, R. Beivide, M. Valero, and G. Rodriguez, "OFAR-CM: Efficient Dragonfly Networks with Simple Congestion Management," in *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on*, 2013, pp. 55–62.

[28] C. Camarero, E. Vallejo, and R. Beivide, "Topological Characterization of Hamming and Dragonfly Networks and Its Implications on Routing," *ACM Trans. Archit. Code Optim.*, vol. 11, pp. 1–25, 2014.

[29] M. Garcìa, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero, "Efficient Routing Mechanisms for Dragonfly Networks," in *2013 42nd International Conference on Parallel Processing*, 2013, pp. 582–592.

[30] D. Xiang, B. Li, and Y. Fu, "Fault-Tolerant Adaptive Routing in Dragonfly Networks," *IEEE Transactions on Dependable Secure Comput.*, vol. 16, pp. 259–271, 2019.

[31] R. Thakur and W. Gropp, "Improving the Performance of Collective Operations in MPICH," in *10th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2003)*, 2003.

[32] Z. Xue, N. Guiqiang, J. Fenglin, and L. Bin, *A New WRR Algorithm Based on Ideal Packet Interval Time*, 2011.

[33] G. Michelogiannakis, K. Z. Ibrahim, J. Shalf, J. J. Wilke, S. Knight, and J. P. Kenny, "APHiD: Hierarchical Task Placement to Enable a Tapered Fat Tree Topology for Lower Power and Cost in HPC Networks," in *Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 228–237.

[34] J. K. Jacobs, "D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees," 2010.

[35] G. Voskuilen *et al.* (2016) SST/macro GitHub. [Online]. Available: https://github.com/sstsimulator

[36] J. J. Wilke, J. Kenny, S. Knight, and S. Rumley, "Compiler-assisted Source-to-Source Skeletonization of Application Models for System Simulation," in *(To appear) International Conference on Supercomputing (ISC '18)*, 2018.

[37] J. J. Wilke, J. Kenny, S. Knight, and G. Michelogiannakis, "The Pitfalls of Provisioning Exascale Networks: A Trace Replay Analysis for Understanding Communication Performance," in *(To appear) International Conference on Supercomputing (ISC '18)*, 2018.