Exploration of Congestion Control Techniques on Dragonfly-class HPC Networks Through Simulation

Neil McGlohon*, K. Scott Hemmert†, Kevin A. Brown‡, Michael Levenhagen†,
Sudheer Chunduri‡, Robert B. Ross‡, and Christopher D. Carothers*
*Rensselaer Polytechnic Institute, Troy, NY, USA
†Sandia National Laboratories, Albuquerque, NM, USA
‡Argonne National Laboratory, Lemont, IL, USA

***Abstract*—Ensuring optimal communication latency in High Performance Computing (HPC) networks is of critical importance to the efficient operation of facilitated applications. Different application operations and types of tasks, such as IO operations, can create a variety of traffic patterns across the system interconnect. Some communication patterns, however, can be problematic for overall system performance.**

One traffic pattern of particular concern is the many-to-one or incast. When packets sent from many different endpoints target a singular, or small number of destinations, it can overwhelm the receiving endpoints' ability to process the traffic, resulting in a cascading effect of induced congestion. This can have broad-reaching, detrimental effects to other applications as their data streams encounter induced congestion.

The concept of congestion control has been explored in various HPC system technologies and is an important feature in state-of-the-art networks such as Infiniband and the Cray Slingshot interconnect. Because access to physical, full-scale interconnects of bleeding-edge design can be challenging and the exact mechanisms of operation not publicly known, we look to simulation to explore techniques for congestion control with a fine level of flexibility not available on real-world systems.

We present and explore a mechanism for congestion control which seeks to detect network congestion, identify its cause, and abate it by throttling injection of identified aggressor endpoints. Our work proposes, discusses and evaluates two similar mechanisms for congestion control in two different network simulators and their capabilities at mitigating the effects of congestion on application communication performance and general system packet latencies.

I. Introduction

The performance of applications on a High Performance Computing (HPC) system is dependent on a number of factors. While specifications and design details play a significant role, one factor that is difficult to predict is the effect of application interference as a result of contention on the underlying communication network resources. As HPC innovation drives toward the threshold of 'exascale' computing [1], new technologies for managing system resource contention must be developed.

A single system may facilitate hundreds of independent jobs simultaneously, each vying for network resources. When network switches are overwhelmed with traffic, congestion can arise, leading to increased packet latency and decreased application communication performance. Making matters worse, once network congestion manifests, it can spread. Analogous to vehicles trying to avoid heavy congestion on a highway by taking alternate routes on side-streets, secondary congestion will consequently form.

Once a hot spot forms, if nothing is done to actively treat the congestion, there is little hope of the situation resolving on its own. The best way to combat congestion is to take measures to avoid it in the first place. One solution would be reducing the overall network allocation, running fewer applications at a time. This might result in improved performance and reduced interference but comes at a cost of a lower total application throughput.

To combat congestion once it has already been discovered on the network, a process called congestion abatement can be used to treat it. The process of congestion abatement is rather simple: reduce the injection rate of nodes contributing to the congestion in the network. This process is maintained until the congestion is naturally resolved as the existing in-transit packets are routed and ejected from the network.

To effectively utilize congestion abatement, congestion must first be detected and the causal ranks – the endpoints responsible for creating the problematic traffic pattern – identified. The three mechanisms: congestion detection, causation, and abatement have been implemented in two HPC network simulators, CODES [2] and the Structural Simulation Toolkit (SST) [3]. By observing the results of congestion management techniques used in two independent simulators, we can strengthen confidence in the efficacy of said techniques.

In order to estimate the capabilities of the latest developments in network architecture, we simulate networks with the same underlying topology as the HPE-Cray "Slingshot" interconnect [4], the 1D Dragonfly [5], using these two simulation platforms. Workloads applied to these simulated networks include online workload models designed to emulate common real world HPC applications and traffic patterns using the Scalable Workload Models (SWM) [6] library and SST's Ember motifs [7]. We create problematic, aggressor, incast traffic patterns which overwhelm exiting port buffers and induce network congestion as a result.

In this work we present simulated implementations for rate-limiting congestion control in two HPC interconnect simulation platforms. We explore the effects of congestion caused by problematic traffic patterns such as many-to-one, including the inadequacy of just using adaptive routing for handling this problem. Finally we demonstrate the efficacy

of the presented congestion control techniques at restoring network performance.

II. Background

We simulate an underlying HPC communication network topology known as the 1D Dragonfly. The interconnection scheme of Dragonfly networks features local *groups* of switches with some degree of connectivity between switches in each group. These groups of switches are also connected to each other with global links providing one or more connections between each pair of groups.

Dragonfly networks, dating back to the late 2000s, were originally designed as a highly-scalable, cost-efficient, innovation in HPC topology research. Notable real world systems featuring a Dragonfly interconnect include the HPE-Cray "Cascade" XC series of supercomputers [8] and latest "Slingshot" systems [4].

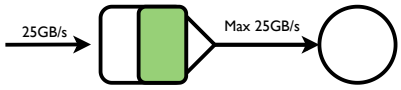
A. The Many-to-One Problem

Consider a single endpoint targeting another specific endpoint injecting packets using all of its available bandwidth. As long as the targeted endpoint's maximum ejection bandwidth is at least as much as what is being injected and it is not targeted by any other source endpoints, its output buffer will likely not become overwhelmed (Figure 1a). This is because its capacity to eject packets from the network is roughly equivalent to the rate with which packets are arriving from the source endpoint.

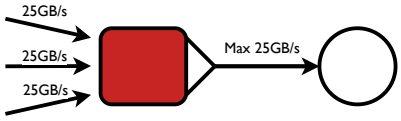
On the other hand, if *multiple* endpoints target a single specific endpoint and inject packets using all of their available bandwidth, it's easy for the targeted endpoint's output buffer to become overwhelmed (Figure 1b). This is the *many-to-one* or *incast* problem.

As packets queue on the output port, waiting to be ejected from the network, more and more arrive, eventually filling up the buffer. Once packets fill up the buffer, new packets destined for the same target will have to wait in buffers on other switches due to lack of buffer space. Congestion is now accumulating in the network.

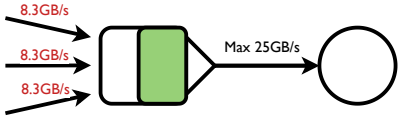
The effects of congestion are not limited to the applications that cause it; any applications sending packets that traverse congested resources will also be negatively impacted. To make matters worse, as long as the endpoints responsible for creating the problematic traffic continue their behavior, the problem will likely spread. This effect is shown in Figure 2, which shows the percentage of total simulation time each port connecting routers in the network is actively transmitting data, sitting idle, or stalled waiting on credits for the cases of with and without an active incast. In the figure, the higher the point on the diagram, the more stalled the port. Figure 2a shows the case for a 8000-node nearest neighbor communication pattern (Halo3d26) with no incast on a 8192-node dragonfly network using adaptive routing. Figure 2b shows this same pattern running concurrently with a 63-to-1 incast pattern. Note how many of the ports in the network spend a significant amount of time stalled waiting for credits.



(a) Incoming traffic matches ejection bandwidth: healthy output buffer.



(b) Incoming traffic exceeds ejection bandwidth: overwhelmed output buffer.



(c) Incoming traffic limited to ejection bandwidth: healthy output buffer.

Fig. 1: Example visualizations of switch output buffers (rounded rectangle) connected to an endpoints by a single 25GB/s link. Each output buffer is supplied with aggregated traffic sourced from elsewhere in the network. The maximum amount of sustained aggregated traffic that this buffer can handle without becoming overwhelmed is 25GB/s.

This is the problem that congestion control looks to solve. It seeks to reduce the rate of injection of endpoints who, as a group, target individual output ports. By reducing the collective rate of injection to be equivalent to the rate of ejection of the targeted output port (Figure 1c), then said port will not become overwhelmed resulting in significantly reduced network congestion.

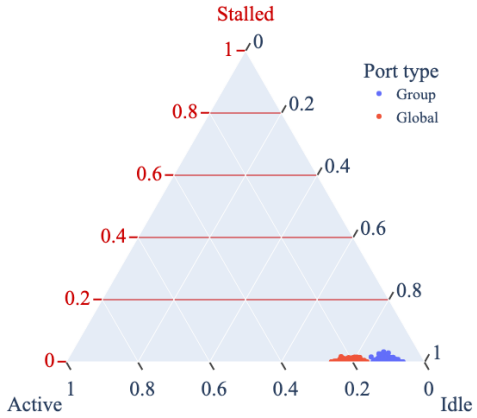
B. Congestion Control

The process of congestion control, also referred to as congestion management, employed by the works in this paper follow a three-phase approach. Congestion is *detected*, its *cause* is identified, and the source is *abated*. These three modules work together to identify problematic traffic patterns and address the problem at the source without adversely affecting non-aggressor endpoints and applications.

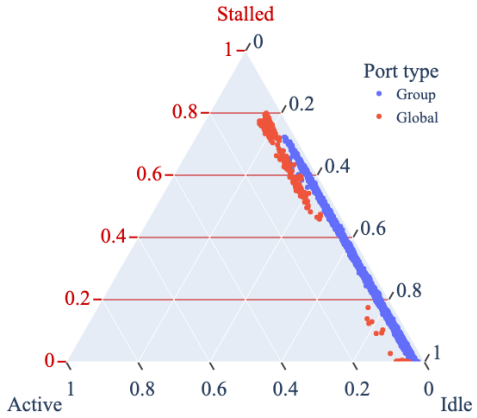
1) Congestion Detection: Before network congestion can be addressed, it must first be detected by the system. Without an accurate and responsive mechanism to accomplish this, the congestion issue will continue to grow as the aggressor ranks continue to inject traffic without restriction.

To reduce the amount of time that problematic traffic can continue to propagate unabated, a congestion control system might employ a policy to detect warning signs of congestion. If a system waits until the congestion has become problematic to enact its abatement policy, then it will be that much harder to address. In the mean time, other applications in the network will be experiencing communication interference as network resource contention remains high until the abatement policy can return the network to a normal operative state.

2) Congestion Causation: Detecting when and where congestion exists in the network is only one part of the pre-



(a) 8000-node Halo3d26



(b) 8000-node Halo3d26 plus 63-to-1 Incast

Fig. 2: Ternary plots of the effects of incast traffic on port activity in a SST simulation of an 8000-node halo job with and without a 63:1 incast. The plots show the percentage of time each intra-group and global port spends idle, active or stalled. The most pertinent metric is the stalled percentage, and points higher in the plot correspond to higher percentage of stalled time (percentages shown in red on the left). The halo job creates practically no stalls in the network, whereas adding the incast causes a large number of ports to have significant stall time.

abatement control phase. The system needs to determine *what endpoints* are responsible for creating the buffer backlog. Without this information, the congestion control mechanism might target non-problematic ranks and adversely affect non-aggressor applications.

3) Congestion Abatement: The method of action utilized in congestion control implementations for this work is utilization of the process of abatement. Aggressor endpoints identified by the congestion causation module receive an abatement signal, ordering them to throttle their rate of injection.

If the rate of injection of all identified aggressor ranks is low enough that they, collectively, cannot overwhelm the ejection capabilities of any one endpoint, then the negative effects of the otherwise problematic traffic pattern can be mitigated.

C. Simulation

As each simulator leveraged in this work, CODES and SST, operate on a PDES engine, they have similarities. Most importantly, any activity in the simulation is represented by an event which modifies the state of entities in the simulation.

1) CODES: CODES is built up on top of the Rensselaer Optimistic Simulation System (ROSS) [9] PDES engine. This provides significant flexibility in how different events in the simulation can be represented. It has been frequently utilized to generate performance results and analysis of numerous HPC network designs and technologies in the past [10]–[13].

This work utilizes the SWM models for the generation of traffic and the 1D Dragonfly network model for switch and endpoint terminal behavior.

2) SST: The Structural Simulation Toolkit (SST) is a PDES platform specifically designed to simulate computer architectures. SST uses a highly-scalable conservative synchronization scheme to ensure that causality in the simulator is not violated (i.e. events don't arrive in the past). SST has been widely used for both node- and system-level architectural simulations [14]–[16].

SST has several packages to simulate networks. This work uses the Merlin network models and Ember/Firefly workload/interface models. Merlin includes models for the interconnect switches and endpoint network protocol layers, while Ember/Firefly provides the application models and network stack model.

III. Implementation

This work analyzes the effects of congestion control techniques on HPC network interconnects with problematic traffic patterns such as an incast. Mechanisms for congestion control have been implemented in both the CODES and SST interconnection network simulation platforms. The design of the congestion control features between these two systems are based on the same concepts discussed in Section II but have slight variances in how the final product is implemented as they were largely developed independently from each other.

A. CODES Congestion Control

The CODES Congestion Control system (referred to as `CODES_CC`) and its implementation was designed to make the control loop of detection, causation, and the resulting abatement action as short as possible. The longer the duration of this control loop, the more challenging it can be to abate problematic traffic patterns.

The way that `CODES_CC` detects congestion is by monitoring the output buffer usage. If this usage crosses a configurable 'threshold of congestion', then the switch considers that port to be congested. Setting this threshold low will allow the system to detect congestion early enough to effectively enact its abatement policy. If this threshold is set too high, then congestion can be detected too late and port buffers can become overwhelmed before abatement signals can be sent.

The switch will consider that port congested until its output buffer usage crosses a configurable 'threshold of decongestion'

which is strictly lower than the one for detecting congestion. Once a port buffer becomes decongested, normal signals are sent by the switch to any ranks that it had previously sent abatement signals to.

CODES_CC determines the cause of congestion by monitoring the source endpoint of every packet in all of its buffers. When buffer congestion is detected, it identifies the endpoints that have packets in a congested buffer and sends abatement signals to these identified ranks.

When abatement signals are received by endpoints, they keep track of what routers they have abatement orders from. As long as there exists one active abatement order on a given terminal, it will remain in a throttled injection state.

When the switch has sent out abatement signals from detected port buffer congestion, it will calculate a minimum amount of time before a normal signal will be sent based on an estimation of how long it would take to reach the threshold of decongestion. Every packet received for an abated port will proportionally increment the time of expiration. An abatement signal will be returned if the source endpoint had not yet been sent one by the receiving switch.

CODES_CC tunes the strength of the abatement mechanism in a similarly distributed manner. Network endpoints monitor the aggregated rate of ejection of all packets that they have injected into the network based on acknowledgement messages received. When ordered to throttle by an abatement signal, endpoints limit their maximum rate of injection to be equal their current known rate of ejection. If N endpoints target a single destination then the total ejection bandwidth will be split amongst them with an average of $1/N$. By limiting each incast participant's injection bandwidth to their known ejection bandwidth, the target will not be overwhelmed.

This tuning has the advantage that if a rank is potentially misidentified as an aggressor, but that their throughput is generally high – and thus not encountering/contributing-to congestion – then they will not be as harshly affected.

Conversely, however, it's possible that an endpoint's packets, due to congestion, have been starved from ejection and thus the endpoint's aggregate ejection rate is zero. This would result in it's abatement-limited rate of injection as also being zero leading to injection starvation. To avoid this, there should be a minimum amount of guaranteed injection rate for endpoints while abated. This minimum can be communicated by the switches ordering the abatement by calculating the source-endpoint cardinality of the identified problematic traffic pattern.

B. SST Congestion Management

The Congestion Management system for SST (referred to as SST_CM) is primarily focused on mitigating the effects of incast traffic patterns on the rest of the network. The system detects congestion by having each switch monitor all packets targeting endpoints connected to it and using that data to determine when likely incast traffic is occurring and which sources are participating. This is done by notifying a central tracking unit whenever a packet arrives in the switch targeting one of the network endpoints attached to that switch. The unit tracks the src, size and arrival time of each packet. These items are then used to determine when to enable and disable congestion management protocols.

To decide when abatement should start, SST_CM monitors the amount of data and the number of sources that are currently targeting each endpoint. When both of these values cross a user defined threshold, the ranks currently participating in the incast will be notified to reduce their injection bandwidth to the target node (traffic to any other node will not be abated). The abatement request has two parts. The first is a request to stop sending all traffic to the affected target node for a specified period (the start-up pause); this value is sent as a number of bytes, which is translated to a time at the endpoint based on bandwidth. The number of bytes is determined by multiplying the number of outstanding bytes in the router targeting the targeted endpont multiplied by a user settable scaling factor. This value is set to allow the congestion that has already built up in the system to start to clear. The second value is a request to limit injection to $1/N$, where N is the number of participants in the incast. This ensures that the target node has sufficient bandwidth to receive the data being sent. As more incast participants are found, the N value is updated for each participant.

Abatement continues until the number of streams participating in the incast falls below the abatement threshold. A stream is considered to timeout after not receiving a packet from that source host within a user tunable timeout period, which is based on the current number of incast participants (i.e. based on the current rate at which that node is injecting into the network). The N value is updated in each source node as streams timeout, allowing the incast to always proceed at the ejection rate of the target. In all cases the CM control mechanisms take a dedicated path through the router, therefore only incurring the latency of an unloaded router.

IV. Environment

This work focuses on understanding the influence that many-to-one-traffic-based network congestion can have on other applications and how techniques for congestion control can be utilized to mitigate it.

We subject simulated 1D Dragonfly networks to different sets of communication workloads and traffic patterns in order to gain insight into the capabilities of a congestion control system. Specifically, we employ the Scalable Workload Models (SWM) [6] and SST Ember motif online MPI traffic generation suites to supply both well behaved and aggressor workloads. We can subject the same workloads with and without congestion control and observe the benefits across a number of metrics. Examples of metrics that we can record include the total communication time of an application from its first message sent to its last message received as well as the end-to-end latency of each packet.

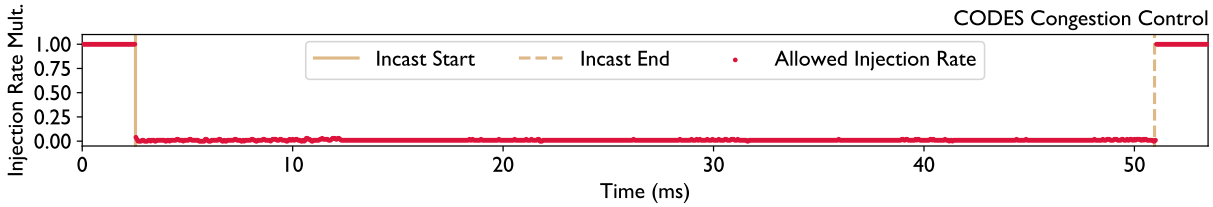


Fig. 3: Allowed injection rate over time of rank 0 in a `PeriodicAggressor` workload which has several iterations of an aggressive 100 rank incast between two iterations of a non-aggressive 2048 rank `LAMMPS` pattern. `CODES_CC` features are activated immediately upon detection of the problematic pattern, throttling by $\approx 1/N$, and the return to normal once it ceases.

A. Workloads and Traffic Patterns

1) LAMMPS: The SWM `LAMMPS` workload features a variety of types of communication. There are a number of messages sent both with and without waiting for acknowledgement of receipt and Allreduce exchanges. The usage of iterative blocking sends is a traffic pattern that is particularly sensitive to congestion.

2) MILC: The SWM `MILC` workload has less variety than `LAMMPS` but is still sensitive to congestion. It posts a number of non-blocking sends and receives, waits for them all to complete and then completes two Allreduce exchanges.

3) Incast: The SWM `INCAST` workload is a many-to-one traffic pattern. One endpoint in the workload is the designated recipient and each other endpoint sends a number of non-blocking messages to it. Its usage as an SWM is designed to be iteratively repeated; all communication from each iteration will be completed before the next is allowed to start.

4) Periodic Aggressor: The SWM Periodic Aggressor workload is a combination of the SWM `LAMMPS` workload and the SWM `INCAST` workload. All ranks will participate in a single iteration of the `LAMMPS` workload, then a subset of them will enter a phase of a number of `INCAST` iterations. Once the `INCAST` iterations have completed, the workload will complete another iteration of `LAMMPS`. This workload is useful for analyzing the responsiveness of a proposed congestion control system; how quickly it can respond to a problematic traffic pattern as well has how quickly it can return to a normal state once the problem has ceased.

5) Fixed Pairs: The SWM `FixedPairs` workload features blocking sends from one half of the workload's ranks to the other half – a simple bisection-bandwidth bound pattern. We can apply several iterations of this simple workload over the course of the simulation. If congestion is encountered by the resulting streams of data, it will very noticeably be impacted. This workload has been configured such that the senders in each pair will send significant amounts of sustained data but cannot exceed the ejection capabilities of the receivers. This traffic can, however cause interference due to resource contention on intermediate switches.

6) Halo3d26: The Halo3d26 workload is simulated using the Ember motif model. The motif is designed to mimic the communication pattern of many nearest neighbor computations. Each MPI rank communicates with its 26 nearest neighbors in the 3x3x3 cube surrounding it. The pattern creates a variety of message sizes, from very large messages on 2 faces, to single word messages at the corners. Each communication phase is preceded by a compute phase and followed by an Allreduce.

B. Network Configuration

This work features simulated 1D-Dragonfly networks of two sizes. The smaller with 3,078 compute nodes is used for simple demonstrations of the concept of congestion control and features 342 36-port switches, arranged in 18-switch groups each with 9 compute node and global links. This arrangement provides 9 global links between each switch group.

The larger, highly inter-connected, 8,192 compute node network is used for more realistic-scale experiments. This network features 512 switches with 64-ports, providing sixteen 32-switch groups with 32 global connections between each switch group.

Specific interconnection patterns may differ depending on the simulator used but high level inferences between generated results are unlikely to be drastically affected as the specific application rank to endpoint mapping is randomized for each workload set to remove potential job placement bias, except where noted otherwise. Of note, all simulations performed with SST utilize an implementation of the Universal Globally Adaptive Load-balancing (UGAL) [17] routing algorithm while CODES uses an implementation of Progressive Adaptive Routing (PAR) [18].

C. Congestion Control Configuration

`CODES_CC` was configured to attempt and recognize congestion before it becomes a problem to minimize the resulting impact on other workloads. The threshold for which `CODES_CC` detects congestion on a port buffer was set to 30% and a decongestion threshold of 10%. For the purpose of establishing an upper bound of congestion control capability, congestion notification messages for the administration of the subsystem are delivered instantaneously. The consequences of relaxing that restriction are explored in Section V-E.

The `SST_CM` studies use the following parameters: Input throttling will occur when a switch detects 8 or more streams with a total outstanding data count of 32kB. Once congestion is detected, the router will request an initial pause of 4

times the number of outstanding bytes. The expiration time of the stream is computed as 1.5 times the current number of active streams times the serialization time of a network MTU (maximum transfer unit). This gives a conservative estimate on the maximum time to expect between packets in a stream.

V. Exploration

To evaluate the performance of a proposed congestion control mechanism we present several experiments featuring well-behaved workloads sharing a network with varying numbers of aggressively competing many-to-one incast traffic patterns.

These experiments are meant to show the capabilities of these proposed congestion control systems but, more importantly, show how simulation can be a helpful tool in understanding the dynamics of a given technology. Simulation is not, by itself, a replacement for experiments with real-world interconnects but is instead a way to explore novel or existing behaviors and the dynamics of the system they operate in.

A. Responsiveness

To be able to adequately address congestion once it is determined to exist in the network, a system for congestion control must react quickly and with adequate strength to throttle the identified aggressors. The faster that the system can respond to a problematic traffic pattern, the less impact that the resulting congestion will have on nearby application traffic.

To demonstrate the responsiveness of the `CODES_CC` module, a `PeriodicAggressor` workload was run in isolation on a simulated 3,078 node dragonfly network. This workload by itself will perform an aggressive incast pattern between two well-behaved iterations of `LAMMPS` with 2,048 ranks. After the first `LAMMPS` iteration has finished, the first 100 ranks perform several iterations of a strong 99-to-1 `INCAST`. After those have all finished, the application returns to a well-behaved `LAMMPS` behavior.

Figure 3 shows the results of this experiment. Specifically it depicts the allowed injection rate of rank 0, which participates in both the `LAMMPS` phases as well as one of the 99 sending ranks in the `INCAST` phase. Near instantaneously, as the `INCAST` phase begins, `CODES_CC` begins throttling the participants of the many-to-one pattern down to one-hundredth of their maximum injection bandwidth. Similarly, very quickly after the `INCAST` phase ends, `CODES_CC` returns the previously abated participants to their normal rate of injection.

B. Effects of Adaptive Routing

In an entirely unoccupied network, the fastest path between any two endpoints is also the path that contains the fewest number of visited switches. This path and any of the same total length are referred to as minimal routes. In an occupied network, however, the shortest path may not always be the fastest. Pockets or hot-spots of localized congestion dispersed throughout the network can cause delays along given minimal routes. Consequently, it can sometimes be beneficial to take

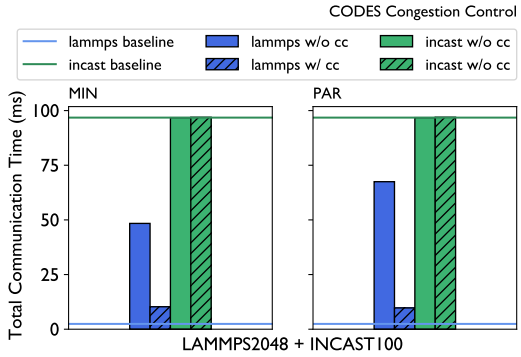
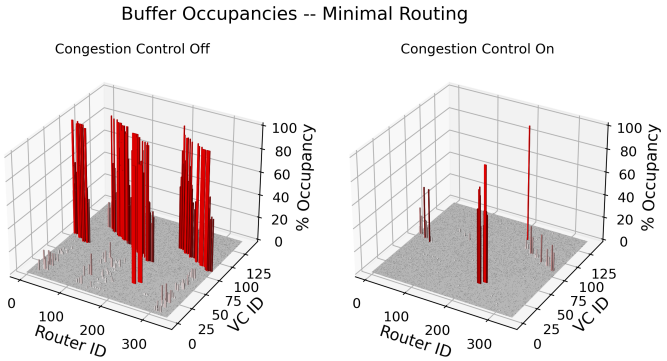
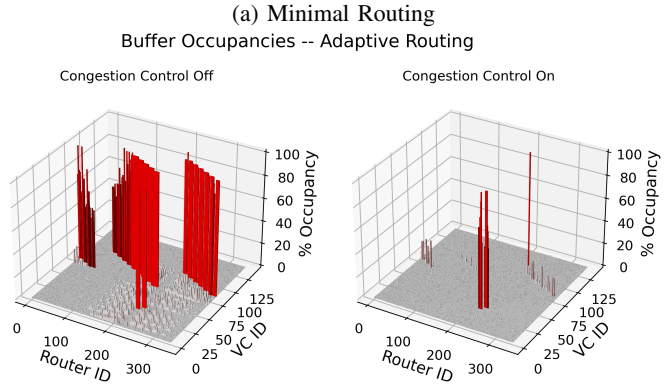


Fig. 4: Communication time of a 2,048 rank `LAMMPS` subjected to an aggressive 99-to-1 `INCAST` on a CODES simulated 3,178 node 1D dragonfly.



(a) Minimal Routing



(b) Adaptive Routing

Fig. 5: Snapshot of CODES network buffer occupancies 2ms (virtual time) into the simulation shown in Figure 4.

a *non-minimal* route [18] and attempt to re-route around observed congestion.

This can, however, have an unintended consequence with congestion causing traffic patterns [19]. By nature, congestion causing patterns such as many-to-one become problematic because they overwhelm the capacity of a single output buffer. They then, through a cascading effect, consume more and more buffer space further away from the ejection port and closer to the sources until the built-up back-pressure eventually causes the sources to self-limit. By enabling adaptive routing and al-

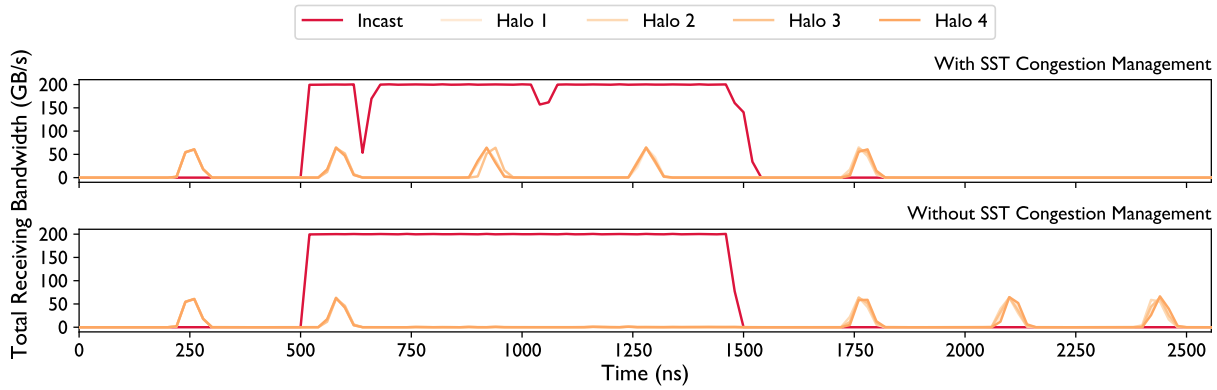


Fig. 6: The figure shows the average bandwidth used by the various workloads overtime. For the incast motifs, the receive bandwidth of the target is shown. For the halo jobs, the average receive bandwidth across all nodes are shown.

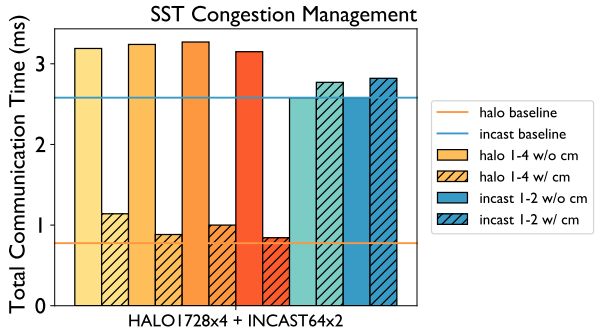


Fig. 7: Effects of SST_CM on four 1728-node Halo3d26 jobs with two 63-to-1 incasts.

lowing non-minimal paths as possible alternative routes, there are then more available lanes for congestion causing traffic to occupy which spawns more opportunities for application communication interference and resource contention.

This effect can be observed with a simple experiment on a 3,078 node dragonfly in Figures 4 and 5. We subject a 2,048 rank LAMMPS workload to an aggressive 99-to-1 INCAST both with and without CODES_CC and also with and without adaptive routing. With adaptive routing, the performance of the LAMMPS workload was indeed degraded. Figure 5 shows the buffer occupancy of each router in the network; adaptive routing opens up many more lanes of traffic to the aggressive incast pattern which was previously limited to minimal routes. This has the effect of spreading traffic around the network, impacting data streams that otherwise would not have been. With congestion control enabled, it reigns in the aggressor and adaptive routing can be effectively utilized again.

C. Throughput Restoration – SST

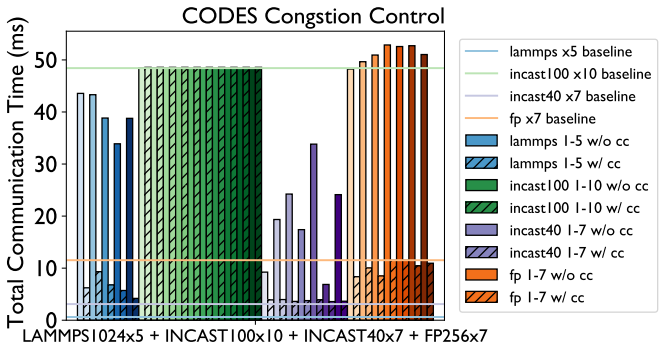
Aggressor traffic can become so problematic that it can prevent applications from making substantial progress. We have conducted an experiment using SST_CM to show its capabilities in restoring system throughput. Figure 6 shows a workload of four 1,728-node Halo3s26 motifs, each running 5 iterations on the 8,192 node dragonfly system. The incast workload starts after the 2nd iteration starts, but before it completes.

Without congestion management, we see that the ability for the HALO workloads to effectively transmit their desired exchanges is significantly limited by the presence of the incast. The second iteration completes because the data is already well underway when the incast begins, but all subsequent iterations are delayed until the incast completes and network congestion clears. A similar experiment was run

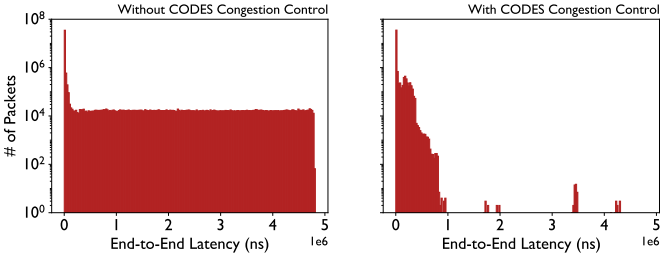
With SST_CM enabled, the system is able to find an equilibrium where the incast ranks are able to inject only what is effectively able to be ejected by the target node. This prevents the occurrence of congestion from the overwhelmed output port buffer. This returns the motif jobs to close to their original performance. The performance of the incast job, however is slightly affected. The first dip seen in the figure is due to the start-up pause requested to help clear existing congestion. The second dip is due to some streams timing out and having CM disabled, then later re-enabled.

A similar workload set with the same Halo3d26 exchange patterns but with two 63-to-1 incasts was also explored. Figure 7 shows the time for each workload to complete its communication. We observe that without SST_CM enabled, the performance of the non-aggressor HALO workloads is limited but when SST_CM is enabled, throughput is largely restored.

Variances in the completion times of each workload are the result of workload specific spatial rank-to-node mappings but are consistent between experiments with and without congestion control features. In particular, this experiment uses linear mapping of ranks to nodes for the halo jobs to take advantage of the nearest neighbor communication pattern. The incast jobs are mapped randomly to the network before the halo allocations are made, and the two jobs that run slightly longer than the others with CM active, have the incast target nodes inside their allocation.



(a) Per-application Communication time.



(b) End-to-end packet latency distribution; Log-y scale.

Fig. 8: Effects of CODES_CC on 29 simultaneously executed workloads on a 8,192 1D dragonfly network: five 1,024-rank LAMMPS, ten aggressive 99-to-1 INCAST, seven minimally aggressive 39-to-1 INCAST, and seven 256-rank latency-sensitive and interfering FixedPairs.

D. Throughput Restoration – CODES

This same occurrence is also observed in CODES. Problematic patterns can definitely impact applications co-existing in the network and due to the effects shown in Section V-B, adaptive routing can prove detrimental to overall network performance despite the goal of improving it.

1) Throughput Study 1: In Figure 8, we show results of a workload set of 8,192 allocated endpoints out of an 8,192 node dragonfly network. 5,120 total ranks are comprised of five independent LAMMPS workloads; 1,792 total ranks are comprised of seven independent FixedPairs workloads. Two different types of INCAST workloads are also utilized in this workload set. Ten jobs are comprised of independent aggressive INCAST patterns each with 100 ranks (individually identical to the one used in Section V-B). Four jobs are comprised of smaller and less intense INCAST patterns of 40 ranks each.

We record that in Figure 8a, without congestion control, each job is significantly impacted by the aggressive many-to-one patterns – including the weaker INCAST workload. The FixedPairs workload, which is designed to be very sensitive and indicative of congestion should it exist in the network is particularly affected, unable to complete until after the aggressive INCAST completes. The FixedPairs workload, which features repeated synchronized traffic streams from specific ranks to other specific ranks, sends randomized chords of sustained traffic across the network. Because no one rank can continue to its next step until all have completed their sends, if any one stream encounters significant delays due to congestion, then the overall performance will be affected. LAMMPS, which has several phases of blocking collective operations is also significantly impacted.

Conversely, with congestion control, each workload apart from the congestion causing INCAST aggressors is able to complete with times much closer to their baseline "empty-network" communication times. These baseline numbers are the maximum communication time of any one workload per type operating in the network as a group: e.g. the FixedPairs baseline is the *maximum* communication time over all seven 256-rank FixedPairs workloads operating in the network collectively without any other types of workloads sharing network resources.

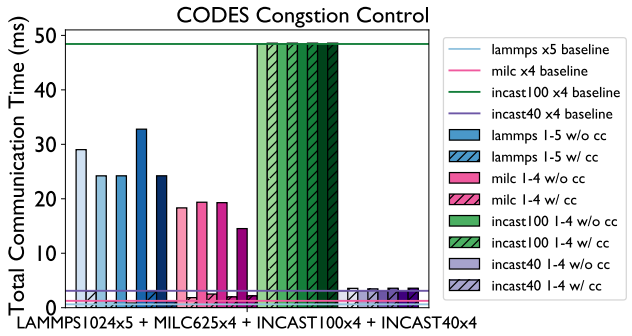
The less intense INCAST workloads performance disparity is particularly interesting. This implies that the 40 rank INCAST patterns – which also have significantly reduced payload intensity – are not ejection bound but that their communication time is significantly impacted by any delay in transmission. In contrast, even without congestion control being active, the aggressive INCAST patterns in green are already at their baseline. Furthermore, when their injection rate reduced to a hundredth of their maximum, they are only minorly affected. This means that the green INCAST patterns are significantly ejection bound and even large delays in transmission or injection do not affect their ultimate performance.

As in Section V-C, variations in per-job communication times within each type of application are the result of the randomly generated rank-to-node mappings.

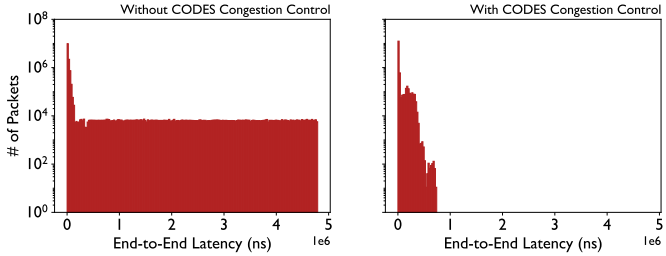
Figure 8b shows distributions of the end-to-end packet latencies for all packets injected into the network across all applications. Without congestion control, the aggressive applications traffic is spread throughout the network by the adaptive routing, impacting the queuing times for packets in nearly every buffer they encounter. When congestion control is enabled, the adaptive routing is able to provide short end-to-end latency times for the vast majority of packets. Outliers in the data appear to be the result of poor choices in adaptive routing as these are made based on local, not global, information on each switch.

2) Throughput Study 2: While the experiment performed in Section V-D1 showed good performance of CODES_CC, the FixedPairs workload, used as an indicator for congestion due to its sensitivity, can itself cause delays to other application packets because of its sustained traffic. We, thus, also simulate another workload set with in a similar manner shown in Figure 9. This figure shows the effects of CODES_CC on a workload set consisting of the same five 1,024-rank LAMMPS workloads, four 625-rank MILC, four aggressive 99-to-1 INCAST, and four weakly aggressive 39-to-1 INCAST totalling 8,180 allocated endpoints in a 8,192 node 1D Dragonfly network.

We observe in Figure 9a that the comparatively small INCAST patterns are still able to significantly reduce the overall performance of the network. This follows from the



(a) Per-application Communication time.



(b) End-to-end packet latency distribution; Log-y scale.

Fig. 9: Effects of CODES_CC on 29 simultaneously executed workloads on a 8,192 1D dragonfly network: five 1,024-rank latency-sensitive LAMMPS, four 625-rank latency-sensitive MILC, four aggressive 99-to-1 INCAST, and four minimally aggressive 39-to-1 INCAST.

findings shown in Figures 4, 6, and 7 where a single or a couple incast traffic patterns can cause significant network throughput problems. CODES_CC, when enabled, is able to bring all workloads much closer to their baseline by limiting the amount of traffic that these INCAST patterns can inject. We also note that the weaker 40-rank INCAST workload did not experience the same amount of interference that it had encountered in the previous study.

Figure 9b shows that without CODES_CC, the adaptive routing performs similar actions as before and spreads the traffic around, increasing the thickness and length of the tail end-to-end latencies distribution. With CODES_CC, more packets are able to be routed minimally and the mean and maximum packet latencies are significantly reduced.

This study demonstrates that when a network is given just well-behaved applications that themselves do not create significant amounts of interference combined with the single congestion-type causing INCAST patterns, CODES_CC is capable of restoring near baseline network performance.

E. Exploring Effects of Congestion Notification Latency

One limitation of the implementation of CODES_CC is the usage of instantaneous, out-of-band, communication of congestion control management messages. These include the ejection acknowledgement notifications and the abatement/normal signals from switches that instruct endpoints of congestion control orders.

Instantaneous communication was first used to establish a best-case scenario for the performance of the CODES_CC system. If it could not effectively abate the congestion by identifying and limiting the injection of problematic traffic patterns with instantaneous communication, it would be unlikely to do any better once delay is added.

This, however, is not realistic in real-world production systems and these messages are likely subject to their own latencies depending on how far, and over what switches and buffers, the messages must travel.

One advantage of simulation is the fine granularity with which developers have control. The congestion control notifications can be set to arrive with any time, including those which are physically impossible to establish best-case scenario performance results. In Figure 10 we do exactly that and relax the expectation of congestion control notification arrival.

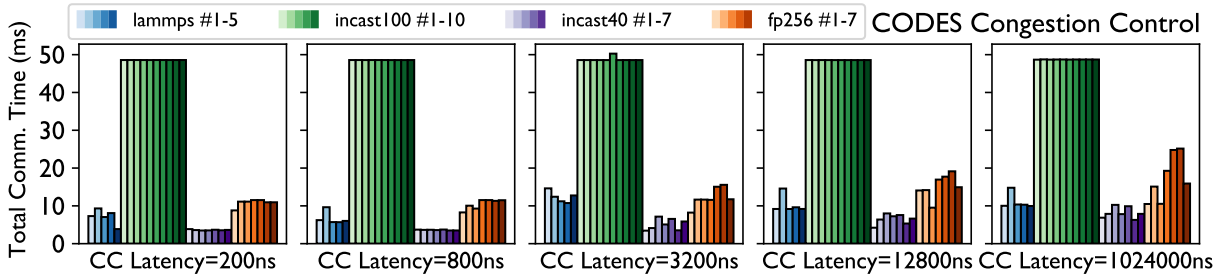
Specifically Figure 10 applies the same experiment exhibited in Section V-D1 but with varying degrees of congestion notification latency. We scale the latency from 200ns to 12,800ns and finally an extreme case of 1,024,000ns. In each of these cases, we see a weakening of the performance of the CODES_CC features. The FixedPairs workload which acts as a 'canary in the coal mine' for congestion indicates that congestion is steadily increasing in overall severity as the congestion control notification latency increases. This is corroborated by the similarly steady increasing length in the tail of the end-to-end latency distributions. The sharp drop-off forming at the end of the tail is an artifact of the fact that congestion control is still operating in some capacity but is not capable of completely eliminating it.

However, there is still an overall reduction in the ability of the INCAST patterns to be able to dominate the network. Even in the worst case with significant delays in congestion control notification arrival times, the FixedPairs workload shows significant improvement ($2-3\times$) over the case without congestion control shown in Figure 8a. Similarly, the LAMMPS workload observes a $3-4\times$ improvement over its no-congestion-control counterpart.

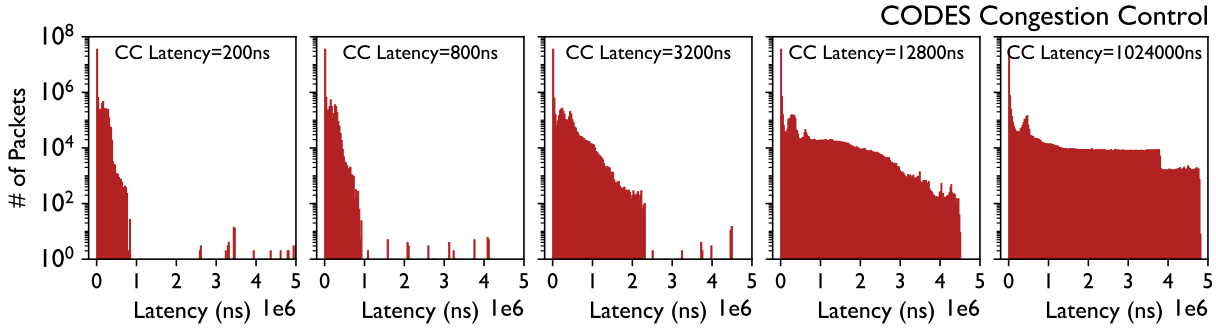
Similar trends were observed when this experiment style were applied to the workload set used in Section V-D2 but is excluded from this work for brevity.

VI. Related Work

The problem of congestion in the modern understanding of network switches has existed since the dawn of IP/TCP protocols [20]. Numerous solutions have been proposed and surveyed for general switching networks [21]. TCP, the protocol acting as the backbone for most data transmitted between nodes in the public internet has historically been the main focus for development of congestion control mechanisms. There are many actions employed by TCP/IP networks to prevent and mitigate congestion, surveys of these techniques can be found in [22], [23]. Common actions by which TCP networks commonly recognize and address congestion is through Explicit Congestion Notification (ECN) [24] and dropping packets.



(a) Total communication time



(b) End-to-end packet latency distribution; Log-y scale.

Fig. 10: Effects of increasing the latency between the sending and receipt of congestion control notifications in `CODES_CC`.

HPC networks, however, operate in a different space. Some difficulties of TCP/IP networking, like relying on transmission of packets to and from devices in vastly different locales and through switches owned and operated by neither the source nor destination, have no analogue in high-performance interconnects. Many HPC interconnect fabrics, like Mellanox Infiniband [25], try to avoid the dropping of packets and when they do it's usually as a last resort to resolve an error. So while in TCP networks, congestion can be, however aggressively, removed by simply dropping all problem packets, this is not preferred in HPC interconnects and is generally avoided. There are mechanisms proposed, however, such as Speculative Reservation Protocol (SRP), Small-Message SRP (SMSRP), and Last-Hop Reservation Protocol (LHRP) which do rely on the ability to drop problematic messages [26], [27] that could be employed in HPC interconnects.

Many networks rely on Infiniband which, in contrast, is considered a loss-less interconnect and must assure that packets not be dropped except in the case of component failure and their Infiniband Congestion Control Architecture (CCA) [28] relies on the communication of ECN messages to and from endpoints to throttle their injection and slowly ease up over time. The effects and tuning of the CCA system have been explored by numerous works [29]–[32].

The authors of [33] propose an interesting solution by introducing flow isolation in addition to injection throttling, bringing in QoS-like techniques for reducing the impact and severity of congestion.

The CODES approach to congestion control was inspired from publicly available patents from HPE Cray [34]–[36], from a keynote [37] and in the short description of the Slingshot interconnect congestion control system [38].

VII. Conclusion

We have presented two independent implementations of an injection-throttling-based congestion control mechanism for the abatement and prevention of congestion in high-performance computing interconnects.

Leveraging these implementations in the CODES and SST network simulation platforms, This work explores the results of experiments from two separate high performance network simulation platforms, CODES and SST, with similar independently developed congestion control mechanisms.

In both simulators the problem of the uninhibited many-to-one traffic pattern and its impact on overall network performance are observed. By identifying the aggressor ranks and rate-limiting their injection, any congestion induced by their traffic can be abated naturally as switches successfully deliver packets to their destination endpoints. This prevents the secondary problem of cascading, spreading, congestion. Similar performance benefits are observed between the two simulators, corroborating their findings.

Finally, this work makes the case for simulation as a valuable tool for the exploration of new techniques toward increasing overall system performance. The ability for fast, fine-grained experimentation of proposed network interconnect technologies can accelerate scientific discovery and innovation; contributing progress along the path toward exascale high-performance computing.

REFERENCES

[1] "Exascale computing project," accessed on 05-24-2021. [Online]. Available: https://www.exascaleproject.org/

[2] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "CODES: Enabling co-design of multilayer exascale storage architectures," in *Proceedings of the Workshop on Emerging Supercomputing Technologies*, vol. 2011. ACM, 2011.

[3] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis *et al.*, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.

[4] "Slingshot interconnect – high performance network for hpe cray supercomputers," accessed on 05-24-2021. [Online]. Available: https://www.hpe.com/us/en/compute/hpc/slingshot-interconnect.html

[5] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *2008 International Symposium on Computer Architecture*. IEEE, 2008, pp. 77–88.

[6] J. Thompson, "Scalable workload models for system simulations," in *Workshop on Modeling & Simulation of Systems and Applications 2014*, 2014.

[7] S. D. Hammond, K. S. Hemmert, M. J. Levenhagen, A. F. Rodrigues, and G. R. Voskuilen, "Ember: Reference communication patterns for exascale." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2015.

[8] G. Faanes, A. Bataineh, D. Roweth, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, J. Reinhard *et al.*, "Cray cascade: a scalable hpc system based on a dragonfly network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 103.

[9] C. D. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low-memory, modular time warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[10] N. Jain, A. Bhatele, L. H. Howell, D. Böhme, I. Karlin, E. A. León, M. Mubarak, N. Wolfe, T. Gamblin, and M. L. Leininger, "Predicting the performance impact of different fat-tree configurations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–13.

[11] M. Mubarak, N. McGlohon, M. Musleh, E. Borch, R. B. Ross, R. Huggahalli, S. Chunduri, S. Parker, C. D. Carothers, and K. Kumaran, "Evaluating quality of service traffic classes on the megafly network," in *International Conference on High Performance Computing*. Springer, 2019, pp. 3–20.

[12] N. Wolfe, M. Mubarak, C. D. Carothers, R. B. Ross, and P. H. Carns, "Modeling large-scale slim fly networks using parallel discrete-event simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 28, no. 4, pp. 1–25, 2018.

[13] N. McGlohon, R. B. Ross, and C. D. Carothers, "Evaluation of link failure resilience in multirail dragonfly-class networks through simulation," in *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2020, pp. 105–116.

[14] K. D. Underwood, M. Levenhagen, and A. Rodrigues, "Simulating red storm: Challenges and successes in building a system simulation," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1–10.

[15] A. Rodrigues, E. Cooper-Balis, K. Bergman, K. Ferreira, D. Bunde, and K. S. Hemmert, "Improvements to the structural simulation toolkit," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS '12. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, p. 190–195.

[16] T. A. Connors, T. Groves, T. Quan, and S. Hemmert, "Simulation framework for studying optical cable failures in dragonfly topologies," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2019, pp. 859–864.

[17] A. Singh, "Load-balanced routing in interconnection networks," Ph.D. dissertation, Stanford University, 2005.

[18] N. Jiang, J. Kim, and W. J. Dally, "Indirect adaptive routing on large scale interconnection networks," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 220–231.

[19] G. Kim, C. Kim, J. Jeong, M. Parker, and J. Kim, "Contention-based congestion management in large-scale networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[20] J. Nagle, "Congestion control in ip/tcp internetworks," *ACM SIGCOMM Computer Communication Review*, vol. 14, no. 4, pp. 11–17, 1984.

[21] C.-Q. Yang and A. V. Reddy, "A taxonomy for congestion control algorithms in packet switching networks," *IEEE network*, vol. 9, no. 4, pp. 34–45, 1995.

[22] J. Widmer, R. Denda, and M. Mauve, "A survey on tcp-friendly congestion control," *IEEE network*, vol. 15, no. 3, pp. 28–37, 2001.

[23] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," *IEEE control systems magazine*, vol. 22, no. 1, pp. 28–43, 2002.

[24] S. Floyd, "Tcp and explicit congestion notification," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, 1994.

[25] M. Technologies, "The official store," Mar. 2019, [Online; retrieved 14-March-2019]. [Online]. Available: https://store.mellanox.com/

[26] N. Jiang, D. U. Becker, G. Michelogiannakis, and W. J. Dally, "Network congestion avoidance through speculative reservation," in *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, 2012, pp. 1–12.

[27] N. Jiang, L. Dennison, and W. J. Dally, "Network endpoint congestion control for fine-grained communication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[28] D. Crupnicoff, S. Das, and E. Zahavi, "Deploying quality of service and congestion control in infiniband-based data center networks," *Mellanox White Paper, Rev*, vol. 1, 2005.

[29] J. R. Santos, Y. Turner, and G. Janakiraman, "End-to-end congestion control for infiniband," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, vol. 2. IEEE, 2003, pp. 1123–1133.

[30] M. Gusat, D. Craddock, W. Denzel, T. Engbersen, N. Ni, G. Pfister, W. Rooney, and J. Duato, "Congestion control in infiniband networks," in *13th Symposium on High Performance Interconnects (HOTI'05)*. IEEE, 2005, pp. 158–159.

[31] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using infiniband architecture congestion control," *Proceedings HP-IPC 2005*, p. 6, 2005.

[32] E. G. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L. P. Huse, and G. Shainer, "First experiences with congestion control in infiniband hardware," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–12.

[33] J. Escudero-Sahuquillo, E. G. Gran, P. J. Garcia, J. Flich, T. Skeie, O. Lysne, F. J. Quiles, and J. Duato, "Combining congested-flow isolation and injection throttling in hpc interconnection networks," in *2011 International Conference on Parallel Processing*. IEEE, 2011, pp. 662–672.

[34] L. S. Kaplan, E. L. Froese, C. B. Johns, M. P. Kelly, A. F. Godfrey, and B. T. Shields, "Congestion detection in a network interconnect," U.S. Patent 9 391 899 B2, Jul., 2016.

[35] ——, "Congestion causation in a network interconnect," U.S. Patent 9 674 091 B2, Jun., 2017.

[36] E. L. Froese, C. B. Johns, A. F. Godfrey, L. S. Kaplan, M. P. Kelly, and B. T. Shields, "Congestion abatement in a network interconnect," U.S. Patent 9 674 092 B2, Jun., 2017.

[37] M. Kagan, "The datacenter is a computer," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*. Los Alamitos, CA, USA: IEEE Computer Society, aug 2020, pp. i–ii.

[38] D. Sensi, S. Girolamo, K. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the slingshot interconnect," in *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2020, pp. 481–494.