Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

TOWARD RELIABLE VALIDATION OF HPC NETWORK SIMULATION MODELS

Misbah Mubarak

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Ave. Lemont, IL 60439, USA

Nikhil Jain

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
7000 East Ave. Livermore, CA 94550, USA

Jens Domke

Matsuoka Laboratory
Tokyo Institute of Technology
Tokyo, Japan

Noah Wolfe

Computer Science Department
Rensselaer Polytechnic Institute
110 8th St, Troy NY 12180, USA

Caitlin Ross

Computer Science Department
Rensselaer Polytechnic Institute
110 8th St, Troy NY 12180, USA

Kelvin Li

Computer Science Department
University of California, Davis
1 Shields Ave., Davis, CA 95616

Abhinav Bhatele

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
7000 East Ave. Livermore, CA 94550, USA

Christopher D. Carothers

Computer Science Department
Rensselaer Polytechnic Institute
110 8th St, Troy NY 12180, USA

Kwan-Liu Ma

Computer Science Department
University of California, Davis
1 Shields Ave., Davis, CA 95616

Robert B. Ross

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Ave. Lemont, IL 60439, USA

ABSTRACT

While the high performance computing (HPC) community is relying on simulations increasingly to co-design and optimize HPC interconnects, the simulation community lacks a coherent set of practices to be followed when validating the simulators and network models. Validation of HPC network simulation models is a multi-step process starting with the selection of representative communication patterns, configuring the network model, followed by designing the set of experiments, and finally, documenting the outcome for reproducibility. In this paper, we present a set of recommended practices for each of these steps in the

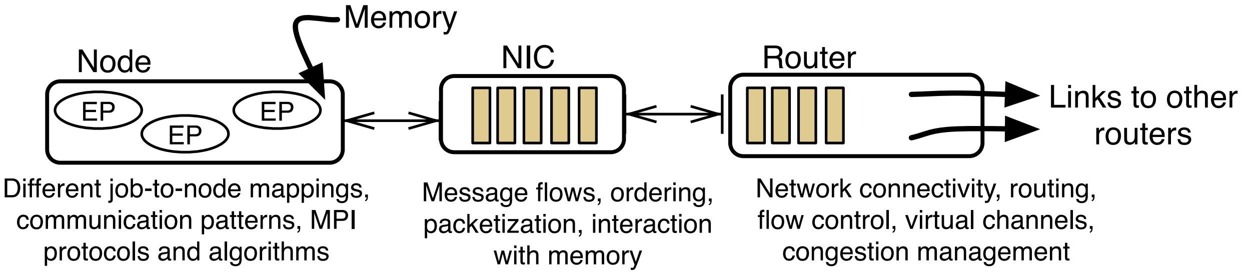

Memory
Node
EP
EP
EP
NIC
Router
Links to other routers
Different job-to-node mappings, communication patterns, MPI protocols and algorithms
Message flows, ordering, packetization, interaction with memory
Network connectivity, routing, flow control, virtual channels, congestion management

Figure 1: Factors impacting communication on HPC systems (EP - communication end-point).

validation process. If the recommendations are followed, the end result should be a *validated* network model that can make reasonably accurate predictions and convince the community about the correctness of the model.

1 INTRODUCTION

High-performance computing (HPC) interconnect simulations are being increasingly used to drive important design decisions for HPC systems such as structure of the topology, choice of routing algorithms, and packet flow control (Won et al. 2015, Kim et al. 2008, Adiga et al. 2005). Additionally, HPC facilities are exploring the use of these interconnect simulations for future system procurements. This situation poses a critical concern for the simulation developers and their users as to whether the model is correct and whether is has been validated. The term “validation” may have different interpretations in the context of HPC network simulation models for different people. In the majority of cases, if specific metrics from the simulation agree with the metrics for a corresponding experiment on the target system (e.g., the predicted runtime matches with the observed runtime), the model is assumed to be validated. However, different metrics may be used in different scenarios. While a group of simulation developers may be interested in validating details of what data is being transmitted at what time and capturing data dependencies, others may be interested in validating predicted runtime only. In this paper, we focus on metrics that can be measured on real HPC systems.

As observed in (Sargent 2005), it is not feasible to validate a model for all of its possible applications throughout its lifetime. However, Sargent emphasizes performing evaluations to validate simulators or models for their intended application or purpose. In the context of HPC networks, for example, if an interconnect simulator is intended to predict the performance of HPC applications for different routing policies or resource allocation policies, then it is important to perform a detailed validation of the simulator against the existing system first before using it for making any policy decisions. However, the simulation community currently lacks a set of general practices that should be followed when validating HPC network simulation models.

This paper presents a set of recommended practices to follow when validating HPC simulators for a particular network topology. We present a four step approach to validate interconnect simulation models by selecting the right combination of benchmarks, configuring the network model, designing validation experiments and simulations, and reporting the results. The methodology is described for validating network models against real systems but it is also applicable when validating against existing simulation frameworks or a prototype of a future system.

2 BACKGROUND

The flow of traffic on networks and its impact on metrics of interest, such as the execution time of an application, depend on a wide range of factors. Thus, it is critical for simulations to mimic the effects of these factors in order to reproduce the targeted real-world scenarios. Figure 1 presents some of the well-known factors modeled in existing network simulations, which we loosely categorize based on their point of origin in the life cycle of data communication.

Multiple applications are expected to run simultaneously on production HPC systems. These applications may have significantly different communication graphs and thus may generate different types of traffic flow on the system (Klenk and Fröning 2017). For example, some applications produce a uniform traffic pattern in which each process in the application behaves similarly, while in other applications the presence of a master entity results in an all-to-one traffic pattern. From the point of view of an end user (e.g., a scientist or application developer), the network communication is with respect to the nodes allocated to the user's job. Thus, the scheme used for allocating nodes to various jobs (job-to-node mapping) and applications running in the job are critical to the characteristics of the traffic originating and ending at different nodes in the system (Agarwal et al. 2006). Further, the number of end points (e.g., MPI processes) that produce and consume traffic on a node may also depend on the application.

Implementations of the MPI standard (MPI Forum 2015) are the prevalent libraries used for communication in HPC. In order to improve functionality and performance, these libraries often support multiple protocols, different error-checking mechanisms, and algorithms for collective operations. These implementation details are important because they affect the observed traffic patterns on a system. For example, the MPI eager protocol may result in an early send of data when *MPI_Send* is invoked, whereas the rendezvous protocol may require explicit synchronization among communicating processes (Brightwell and Underwood 2003).

On most current systems, the task of reading data from the memory and pushing it onto the network is delegated to the Network Interface Controller (NIC). NICs operate in parallel with the computation units in the nodes (such as cores), they can decide the ordering in which messages are packetized and pushed onto the network. As they access memory, they may also contend with computation units for memory bandwidth. Thus, the working of NICs and overheads induced by it can have a significant impact on traffic flows on current systems.

From NICs, the packets are typically transferred to a router or a switch. Depending on their destinations, these packets are then forwarded from one router to another, until they reach their destination NIC. As one would expect, the hardware and schemes responsible for the packet forwarding are significant for the overall communication flow. A majority of simulators focus on these aspects that include the network topology, the delay and bandwidth of the hardware, the routing policies, and the flow-control and congestion control mechanisms. Among the many network topologies that have been proposed and studied for HPC interconnects, three network topologies have been widely used in recent systems: torus (Adiga et al. 2005), fat-tree (Petrini and Vanneschi 1997), and dragonfly (Kim et al. 2008).

To demonstrate the use and impact of the validation methodology presented, we present results. Detailed validation process and results are available as a technical report at http://www.ipd.anl.gov/anlpubs/2017/05/135666.pdf. collected with the CODES network simulation framework (Mubarak et al. 2017). CODES is a general-purpose toolkit for packet-level simulations of traffic flows on HPC networks. CODES is built on top of ROSS (Carothers, Bauer, and Pearce 2002), a parallel discrete event simulation engine. CODES has been successfully deployed for studying various scenarios for HPC networks (Jain et al. 2016, Yang et al. 2016). The results presented in this paper are generated by using either synthetic patterns or CODES's MPI trace replay layer. The purpose of these results is not to validate CODES itself but to exemplify the best practices in validating HPC network simulations.

3 OVERVIEW OF THE VALIDATION METHODOLOGY

In Section 2, we discussed several distinct and often unrelated factors that affect the observed behavior on a network. As a result, a robust validation study requires enumeration and selection of several parameters. One possible workflow, which has been used in recent work (Jain et al. 2016), for systematically designing simulation experiments is presented in Figure 2. In this workflow, the first step is to select communication patterns of interest. This step includes identifying benchmarks and their inputs for simulation. Next, a

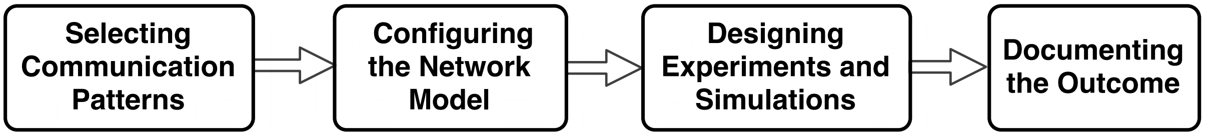

Selecting Communication Patterns
Configuring the Network Model
Designing Experiments and Simulations
Documenting the Outcome

Figure 2: Workflow for conducting validation studies.

prototype for the target system is designed in the simulation framework. Parameters configured in this step include network topology, routing schemes, bandwidth, and overheads of various network components.

Based on the benchmarks and prototype system, experiments that should be conducted are decided; parametric choices here include the jobs to be run, job-to-node mappings, length of the jobs, and environment settings such as interference and MPI protocols. Experiments enumerated in this step are then executed as simulations and real runs as required. Finally, the outcome from the experiment executions is reported after post-processing.

We find the workflow of Figure 2 adequate for conducting validation studies. The first three steps in the workflow are geared to enumerating various factors that affect traffic on HPC networks and can be mapped to different categories shown in Figure 1. The third and fourth steps can be defined to systematically conduct the experiments and report the results. In Sections 4–7, we elaborate on these steps from the point of view of conducting a validation study, and we describe the tasks that should be performed. We highlight the practices that should be followed for a reliable validation study, and we discuss the common pitfalls.

4 SELECTING COMMUNICATION PATTERNS

One of the recurring challenges encountered in a validation study is identification of benchmarks with various communication patterns to be used for validation. In order to do reliable validation, one should select a set of benchmarks that can test different aspects of the simulation framework. For example, relying only on a basic pairwise point-to-point communication pattern is unlikely to reveal the performance differences that can surface when many end points communicate simultaneously. Moreover, to build confidence in applicability of simulations for practical scenarios, the communication patterns selected should be representative of real HPC applications. In this section, we first discuss the different point-to-point and collective communication benchmarks that model developers can use for validation. We then discuss common pitfalls when selecting benchmarks for network model validation.

4.1 Pattern Categories

Communication patterns and benchmarks can be categorized in several ways, e.g. based on the number of communicating partners assigned to each end point or the ordering of communication among sets of end points. We discuss two categorizations, which can be applied to select benchmarks for validating different aspects of network simulation models.

Latency-bound and Bandwidth-bound: Our first categorization of patterns is based on the bottleneck(s) that determine a benchmark's performance. The most common division in this category w.r.t to communication is a binary assignment between latency-bound and bandwidth-bound. We call a benchmark latency-bound if constant, startup, or one-time overheads such as router delay, wire delay, etc. determine the performance of a benchmark. The simplest example of this case is a ping-pong benchmark for small messages. On the other hand, if bandwidth of network resources such as NIC or link determines the performance, we identify a benchmark as bandwidth-bound. A ping-pong benchmark with large messages falls in this category. Coverage along this categorization is important because it validates functioning of distinct aspects of a network simulation models of critical important in practical scenarios.

Simple and Adversarial: Our next categorization is focused on distribution of traffic flow induced by a pattern and the network mechanisms that are critical to the performance in presence of such flows. We categorize a pattern as a *simple* pattern if it results in a somewhat uniform flow throughout the network. For example, consider a uniformly random pattern; in this pattern, all nodes repeatedly perform the following steps: (1) randomly select the next destination according to a uniform random distribution, and (2) send a message to this destination. This is likely to result in similar traffic flow throughout the network, and tests network functioning under high throughput. Often, simple patterns are either manually tractable or their outcome can be predicted with simple models.

Adversarial patterns are designed to create network hot-spots that stress the flow-control and congestion-control mechanisms in the system. Thus, in simulations, these patterns are good candidates for testing the validity of models used for the same purposes. Many-to-one pattern in which several end points communicate with the same end point is an example of adversarial pattern that applies to several network topologies. Network specific adversarial patterns can also be created to further stress test the simulation models for individual networks (Besta and Hoefler 2014).

In practice, it is anticipated that individual patterns may represent combinations of above categories. For a reliable validation, benchmarks that represent all meaningful combinations should be used. In addition, depending on the use case for the simulation models, careful selection of benchmarks used for representing these patterns is also needed. For example, consider the scenario in which simulations are to be used for comparing performance of collective-heavy applications on different networks. Collective communication operations, such as barriers and broadcasts, entail complex dependencies among messages exchanged among a group of nodes. As a result, their performance can depend heavily on the algorithms used in their implementations and how the traffic associated with individual messages flows on the network. Hence, validation using benchmarks with collective operations may expose shortcomings in simulators that are not captured by using point-to-point benchmarks only. In particular, these tests can identify whether the collective algorithms modeled in the simulators match the algorithms used on the real systems.

Another important use case of network simulations can be performance prediction for production applications. Communication patterns in applications typically consist of both point-to-point and collective patterns. Thus, in order to match the observed performance of applications on real systems, in addition to accurately simulating individual patterns, a simulator needs to account for inter-pattern dependence. As a result, in this case, reliable validation would also include production applications as benchmarks to test if higher-level dependence among operations performed in an application are correctly captured. Note that if a simulator is only focused on evaluating best- or worst-case communication characteristics of a network, then validation using applications is not required.

4.2 Common Pitfalls

Benchmark selection is the source of origin for several problems encountered in interconnect simulation validation. Having discussed the benchmark categories that should be used, we now list the common pitfalls when selecting benchmarks.

Infinite messages – Synthetic traffic benchmarks, such as uniform random injection, are helpful in determining the generic network characteristics (e.g., throughput). However, flooding the network while ignoring message dependencies, coupled with independent processing capabilities of the network through network redundancy or virtual channels, may hide bottlenecks or deadlocks in subsections of the network caused by an incorrect model. In order to alleviate the possibility of these issues going unnoticed, tests should also be conducted either by using a scaled back injection rate or by using patterns whose performance is impacted by message dependencies. Dropping the injection rate to a fraction of the node injection link speed can help shed light on incorrect model functionality that can be overshadowed by high message volume congestion in the network.

Computation-intensive benchmark – Using benchmarks in which the majority of the runtime is spent in computation and a small percentage is spent in communication does not reveal the performance differences between the real architecture and the simulation. These benchmarks typically result in low network utilization and fail to stress the limits of network performance capabilities. Instead, benchmarks in which a large portion of runtime is spent in communication makes it much easier to identify differences between the real scenarios and simulations.

Allocating time to develop a complete understanding of available benchmarks as well as testing with multiple different benchmarks helps avoid these common benchmark pitfalls and prevent non-system-related issues from propagating into the next step of the workflow.

5 CONFIGURING THE NETWORK MODEL

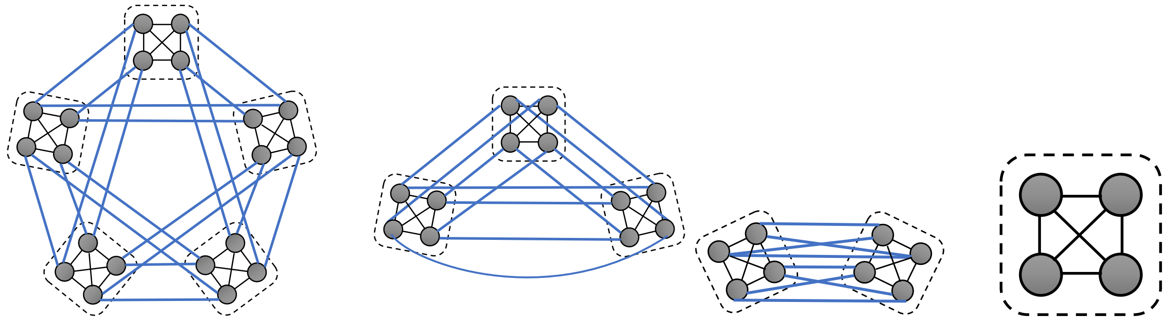
The network topology and its wide range of configurable parameters have a significant impact on the flow of traffic over the network. For instance, it is vital to choose network structural parameters that accurately represent real world system values to provide useful validation of simulation network results with real-world system runs. Further, overheads of various components, e.g. NIC and router, need to be modeled as closely as possible in the simulation to their real world counterparts. As a result, the primary focus for the model designer is establishing an accurate and consistent validation sub-system for comparing results between simulation and real hardware.

5.1 Accounting for Topology Structure

In most cases, the network system size established for validation studies is much more modest in scale as compared to the size of production systems. The use of smaller scale configurations is partly driven by logistical constraints such as the physical unavailability of large/futuristic systems to generate real results for comparison with simulation. The desire to keep the studies tractable for manual analysis can also be a driving factor for the use of smaller networks in validation. A very common issue resulting from the use of excessively small or simplified system configurations for validation is the failure to capture the complexities associated with the large-scale system counterpart. For example, validating a dragonfly model meant for simulating large multi-group systems using a setup comprising only one group can be misleading since it fails to capture the effect of inter-group communication.

For a reliable validation study, we suggest running experiments on a system that is *proportionately* smaller and still maintains *functional similarity* to the full-scale production system. The smaller system should exhibit characteristics of the larger network and should not be devoid of properties that define the larger network (e.g. number of connection hierarchies). Since many of the network topologies used in HPC are regular in shape, it is relatively easy to identify smaller systems that are suitable representatives of larger systems. We list a few examples below.

Dragonfly Networks: In a dragonfly network, routers are divided into groups and connected in a hierarchical manner (Kim et al. 2008). An example dragonfly network is shown in 3a consisting of five groups with four routers per group. Communication within groups traverses intra-group links (shown in black), while inter-group communication can utilize both intra-group and inter-group links (shown in blue), depending on the system connectivity and the location of the source and destination nodes. As shown in Figure 3d, validation using a reduced dragonfly network containing only one router group fails to include the influence of inter-group connections on network performance. Instead, a strong validation of the dragonfly network requires the utilization of a system composed of multiple groups enabling the study of both inter-group and intra-group link connections. Using two groups, as shown in figure 3c, incorporates the inter-group links but increases the impact of those links as both of the inter-group connections from each router connect to the same group, doubling the router-to-group bandwidth compared to the larger five group dragonfly configuration. Overall, the best small scale validation setup is shown in Figure 3b where a minimum of three groups are needed to observe both inter-group link activity and performance that mimics the performance of the larger scale five group system.



(a) Full Five group dragonfly network (b) Three group representation (c) Two group representation (d) One group representation

Figure 3: Example small scale representations for validation of a full five group dragonfly network with five routers per group and two global connections per router. Black lines indicate intra-group connections and blue lines indicate inter-group connections.

Torus Networks: In the case of a torus network, the structure of the topology stays the same throughout the system. The nodes connect to one another in a uniform n-dimensional manner such as the 5-dimensional tori of IBMs Blue Gene systems (Adiga et al. 2005). Thus, a smaller n-dimensional grid is a good representative of the expected behavior of a much larger grid.

Fat-tree Networks: Fat-trees are highly regular topologies which can be grown either horizontally by utilizing free switch ports or vertically by adding additional layers (Petrini and Vanneschi 1997). Validation performed between real and simulated systems that have different numbers of tree layers is likely to influence the reliability of the comparison, since more layers may result in congestion characteristics that cannot be reproduced with few layers. In contrast, horizontal contraction of large systems (either by reducing the number of leaf-level groups or size of the groups) is unlikely to restrict the traffic flow characteristics that can be produced with a large system.

5.2 Accounting for Routing Algorithms

The routing algorithm plays a critical role while performing reliable network model validation. The chosen algorithm varies from system to system depending on the preferences of the administrator, supported algorithms of the deployed networking technology, as well as the topology used to connect the nodes. For example, on the Blue Gene torus systems, both dimension-order (static) routing and dynamic routing protocols are available (Adiga et al. 2005). Similarly, dragonfly-based systems support multiple routing algorithms, such as minimal, non-minimal, adaptive, and progressive adaptive routings (Jiang, Kim, and Dally 2009). Unlike these systems, most InfiniBand-based configurations use a fat-tree topology and are limited to the choice between a few static routing algorithms (Domke, Hoefler, and Matsuoka 2014).

Often, any optimizations to the routing algorithms are hidden from the users of HPC systems which makes the validation process challenging. In this case, the simulation framework needs to use the closest known routing protocol, and a discussion of the routing algorithm utilized during validation needs to be reported. Additionally, if there are significant performance differences due to the routing algorithm, it is useful to perform additional simulated experiments to isolate the problem and ensure that the differences are due to the routing.

5.3 Accounting for General Parameters

Finalizing the validation framework requires establishing a complete set of precise system metrics across both the simulation and real system. These metrics describe the speeds, latencies, buffer sizes and other components of the network system that are vital to replicating the performance of physical systems with simulation models. Many of these parameters such as link bandwidth and router delay are determined according to the deployed network technology. For example, with InfiniBand networks, product specifications for physical switches provide link speeds, buffer sizes, switch traversal latencies, and more needed to validate the simulation model with a real system accurately. Other metrics, such as max packet size, number of virtual channels, and eager/rendezvous protocol thresholds can often be configured by the system administrator or with environment variables. Finally, metrics that cannot be determined from written specifications need to be collected experimentally. For instance, delays associated with MPI and NIC overheads can be extrapolated by measuring the round trip time of small message transfers between MPI processes on the same node. Finally, any and all differences or potential sources of error must be reported. In this case, experiments should also be performed to measure the effect of the error.

5.4 Common Pitfall

The large quantity and typically hidden nature of parameters relating to system interconnection networks translates to increased chances of error in validation. Here we discus one such setback associated with the system design process.

System Dependencies – Caution is advised with respect to the simulated network topology. Especially for statically routed networks, any minor discrepancy between the actual HPC system and the simulated topology and routing combination may yield heavily diverging results, as reported by (Domke, Hoefler, and Matsuoka 2014). For example, only two link failures in an HPC system with a few hundred nodes interconnected by a two-level fat-tree can show significant performance degradation (up to 30% in overall throughput) compared to the simulation which assumes a fault-free configuration. Hence, if possible, then the validation should be based on the exact same topology configuration as the real system.

The system design component of the validation workflow can be a tedious process. There are a number of network parameters to quantify and replicate, resulting in a number of possible sources of error within the model. However, constructing a proper system configuration is a fundamental component for instilling confidence and establishing accuracy necessary to move on to designing successful experimental simulations.

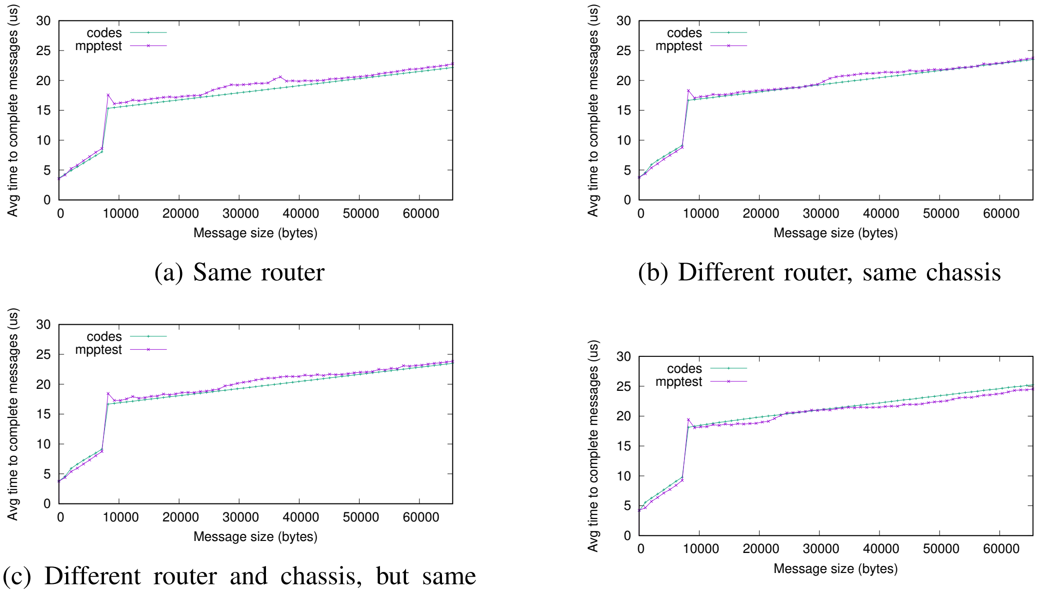
6 DESIGNING EXPERIMENTS AND SIMULATIONS

The validation of network simulation with a real system or another simulation framework often constitutes a small part of the study being conducted, and therefore commonly insufficient information is reported on the experimental design of the validation experiments. For example, whether network interference was taken into account, which MPI protocols were being simulated, and which ones were supported by the system. We discuss the techniques to design the experiments, including the elements that should be covered so that the validation process leads to plausible conclusions.

6.1 Benchmark Parameters and Placement

The selected benchmarks must be configured so that they can test various aspects of the simulated network system such as varying distance between source and destination nodes, various message sizes etc. We demonstrate with an example using a ping-pong benchmark for evaluating the CODES dragonfly network model against the MPI performance measurements on the Theta Cray XC system (Faanes et al. 2012). The performance measurements on the Cray XC system are done by using the mpptest (Gropp and Lusk 1999) performance benchmark, which measures the performance of MPI operations in various scenarios such as ping pong, bisection pairing, and ghost cell.

The dragonfly network topology of the Cray XC system is a hierarchical network with groups connected to each other via all-to-all global links. Within each group, the routers are spread out in a two-dimensional 6x16 mesh, and four compute nodes are attached to each router. Because of this multilevel structure, the validation tests should record performance overheads in different placement scenarios, for example where



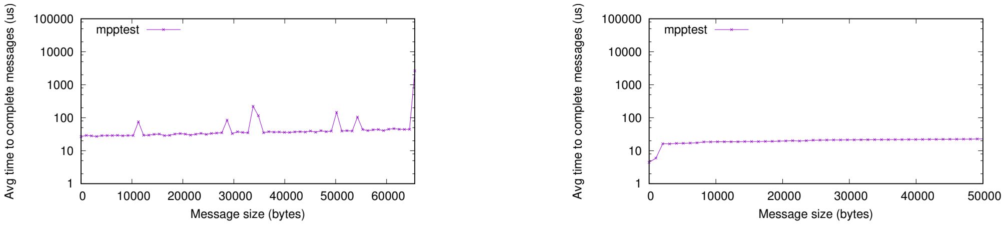
(a) Same router

(b) Different router, same chassis

(c) Different router and chassis, but same group

(d) Different groups

Figure 4: Validation of CODES dragonfly network model against Theta Cray XC system using the ping-pong benchmark. The tests on Cray XC were done on a quiet system with no other jobs running on the system.



(a) With network interference

(b) Without network interference

Figure 5: Interference effects as recorded by the mpptest benchmark on the Cray XC system.

(i) processes on different nodes and same router communicate, (ii) processes on different nodes, different routers but same group communicate with each other, and (iii) processes in different groups communicate with each other.

In the ping-pong benchmark used on Cray XC, the performance of MPI blocking messages was recorded with message sizes varying from 0 bytes to 65,536 bytes in increments of 1,024 bytes. Figure 4 shows the validation results of the dragonfly model against the Theta Cray XC platform. Note that the MPI software overhead and NIC delays are significant in this case as compared with the network traversal delays, which is why a small difference is observed when comparing cases that traverse multiple network hops. For most of the message sizes, the simulation results match the performance recorded on Theta. In some cases, however, the simulator slightly underpredicts the results; we conjecture this is due to the operating system noise and memory interference on the real system, which were not captured in the simulation framework.

6.2 Accounting for Network Interference

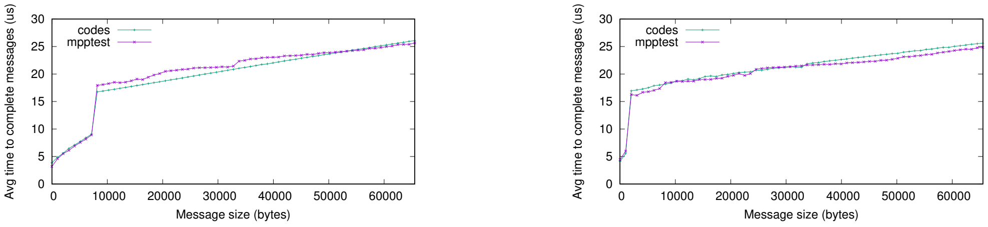
Network interference can be a source of considerable performance jitter and leads to nondeterministic results (Petrini, Kerbyson, and Pakin 2003). Validation studies often tend to ignore the effect of interference and its impact on the simulation performance being reported. Specifically, the studies do not report whether the experiments were performed on a quiet system or whether other jobs were running on the system. Hence, network interference may be misinterpreted for unintended model behavior, or severe model errors may be written off merely as background noise in the network.

To illustrate how much effect interference can have on performance, we present MPI performance measurements on a Cray XC system with and without network noise. Figure 5a shows the case when performance was recorded while other jobs were running in the system, while Figure 5b shows the case when the performance measurements were done on a quiet system with no other job running. In addition to the performance difference, one can see the spikes that are present only when there is interference in the system.

Hence, comparing measurements on an HPC system that executes multiple jobs simultaneously, with the simulations that run a job in a controlled environment while disregarding background noise from unused network nodes, unavoidably results in an apples-to-oranges comparison. However, multiple workarounds exist to manage the interference effect in the validation experiments. One simple solution is to run the experiment to measure the system performance on a quiet system when no other jobs are running. We have followed this strategy for reporting the Cray XC performance data in this paper. Another approach to deal with the problem is to quantify the interference on the target system and reproduce the interference effects in the simulation by artificially creating background noise. This would ensure that an adequate comparison is performed since the simulated system will not have idle network nodes.

6.3 Accounting for MPI Eager and Rendezvous Protocols

Most of the simulation frameworks use benchmarks to measure the MPI performance on the target systems, but the validation data does not provide details on the MPI protocols that were modeled in the simulation. Furthermore, HPC systems tend to send small messages eagerly, and large messages are typically sent by using a rendezvous protocol (Brightwell and Underwood 2003). In case of the rendezvous protocol, a handshake occurs prior to the actual data transfer, and it does not have the buffer copying overheads that are present in the eager protocol. The threshold at which the transition from the MPI eager protocol to the MPI rendezvous protocol occurs varies from system to system (Adiga et al. 2005), thus raising several questions in the validation process. For example, were the MPI protocols being modeled in the simulation the same as those on the target system? If different MPI message sizes were being used in the validation, what was the transition point between the MPI eager and rendezvous protocols?

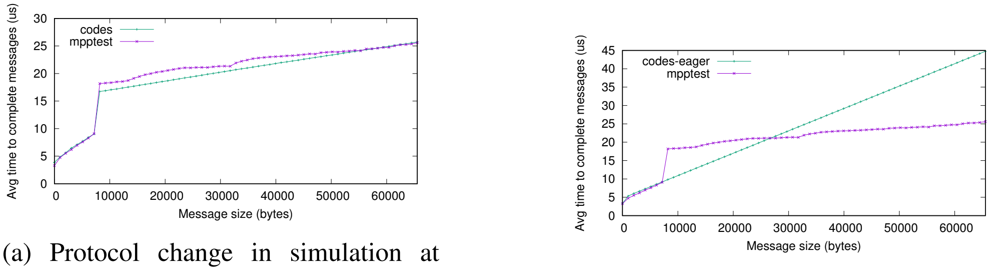


(a) For 64 nodes, protocol change occurs at 8 KiB (simulator is configured accordingly).

(b) For 1.024 nodes, protocol change occurs at 2 KiB (simulator is configured accordingly).

Figure 6: Validation experiments accounting for transition from the eager to the rendezvous protocol. The CODES dragonfly model is compared with MPI performance measurements with the mpptest bisection pairing benchmark on the Theta Cray XC system while no other jobs were running on the system.

Figure 6 shows how the protocol transition varies for different node counts on the Cray XC system. For smaller node counts (e.g., 64 nodes, as shown in Figure 6a) the transition occurs at 8 KiB, whereas for larger node counts the transition occurs at 2 KiB; as shown in Figure 6b. This figure showcases how



(a) Protocol change in simulation at 8 KiB.

(b) No protocol change.

Figure 7: Validation experiments comparing the effect of a MPI protocol change on the Theta Cray XC system on 16 nodes using the mpptest bisection pairing benchmark. No other jobs were present on the Theta system during the benchmarking.

our simulator implements both the eager and rendezvous protocols and illustrates the point at which the transition occurs is configurable. Therefore, to accurately mimic the protocol transition of the system, we configured the simulations accordingly by analyzing and identifying the right transition points from the collected system performance data. The importance of supporting both protocols becomes clear when analyzing Figure 7, which shows the noticeable performance differences in the simulation when the protocol is not switched to rendezvous at 8 KiB for the dragonfly model. In this case, the buffer copying overhead of the eager protocol increases the overall cost for the message transfer, resulting in much higher latency for large messages; see Figure 7b.

6.4 Estimating System Overheads

Another component of validation involves estimating various network overheads accurately and then representing them in the simulation. We discuss the overheads that should be taken into account when performing validation, and we describe the ways these overheads can be measured on a real system.

MPI software-level overhead, NIC delays, and router processing overheads: On a real system, these overheads can be derived by sending messages of zero bytes to a destination process mapped to the neighboring node. In order to discount the impact of operating system interference, multiple messages should be sent, and the resulting mean latency for these messages should be used in the simulation. Ideally, the deviation from the mean value should also be reported, which will give an idea of the degree of noise on the target system. On a torus network that has one router per compute node, the message will also include the sender/receiver router processing overhead. On a dragonfly network, this overhead will include only a single router overhead since routers are shared resources on high-radix networks.

Copy per byte overhead: This represents the copying cost for eager messages. It increases on a per byte basis, and it can be derived by eagerly sending messages in the range of zero to several kilobytes (possibly to a neighboring node for avoiding interference by other jobs).

Energy-saving features and wake-up delay: Modern HPC interconnection technologies, such as InfiniBand, contain multiple energy-saving features, for example, disabling unused ports in a router, frequency scaling, or link speed reduction (Mellanox 2013). Hence, the wake-up delay—the time between throttled performance and the full performance after the first packet arrives—adds additional overhead, not only in the local NIC, but potentially also in the rest of the network between the source and destination. This overhead can be estimated in a manner similar to that for the general NIC overhead. When sending multiple messages, however, the delay between the messages must be large enough to trigger the energy saving. Alternatively, the provider of the networking technology must be consulted to inquire about estimated delays.

Data locality for applications: When applications on the real system and application traces for the simulator are being compared for the validation study, an important yet mostly overlooked influence is caching effects and data locality. This issue is discussed in the next section.

6.5 Accounting for Memory and Caching Effects

Using application traces for network model validation can be an important part of the validation process; but comparing actual application performance to simulated runs, which are based on idealized/optimal MPI latencies for certain messages size, can show significant variance. For real benchmarks the locality of the data that needs to be transmitted through the network actually matters. An example is the memory bypass of the Tofu 2 interconnect, which allows the nodes/NICs to directly read and write from and into the L2 cache of the CPU (Ajima et al. 2014). Furthermore, our investigation has shown that not only the data locality but also the cache misses, when accessing MPI-internal data structures, attribute to the increased latency. To exemplify this behavior, we modified the latency benchmark of the Ohio State OSU benchmark suite to enforce cache misses at the different levels of the CPU, and to report improved statistics, as shown in Figure 8.

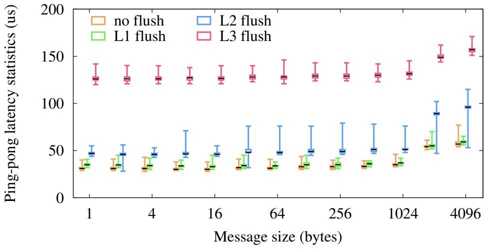


Figure 8: Influence of data locality on the ping-pong latency between nodes using IPoIB on an FDR InfiniBand fabric (2 nodes on same router). Collected latencies are represented by whisker plots showing the 0$^{\text{th}}$ to 4$^{\text{th}}$ quartiles for the 128 performed communications per message size. These message sizes range from 1 byte to 4,096 bytes. Prior to sending the ping message, the MPI process flushes its CPU data caches, resulting in different latencies depending on the flushed cache levels.

For demonstration purposes, we configured MPI to send the messages via the IP-over-InfiniBand interface of an FDR fabric instead of the native IB interface, which highlights the problem well for small messages in the range 1 byte to 4096 bytes—the predominant message sizes for HPC applications (Klenk and Fröning 2017). Similar effects, while potentially less detrimental, can be observed for other interconnection technologies and vary based on messages size, chosen MPI implementation, cache sizes and levels, and communication protocols. If the application used for the validation shows unfortunate data locality and its performance is latency sensitive, these cache effects may need to be accounted for in the network model.

6.6 Common Pitfalls

Having discussed the techniques for designing experiments, in this section we describe some of the common issues associated with the experimental design phase of the validation workflow.

Restricted message sizes – Experiments done with only a few message sizes (e.g., $<$ 1 KiB or $>$ 1 MB) fail to validate the network model for different performance regimes in which tradeoffs between constant delay and bandwidth determine the outcome. Thus, we recommend conducting experiments and simulations for a wide range of message sizes and total communication volume.

Ignoring traffic interaction – Benchmarks that are restricted to two network nodes only (e.g., ping pong) do not cover the network interference where multiple network nodes are generating traffic. Therefore, using a ping-pong benchmark alone is not sufficient for a complete network model validation. Additional benchmarks invoking communication on a substantial portion of nodes in the network need to be included in the experimental design in order to enable a comprehensive comparison of network performance in response to traffic.

On-node communication – Application traces occasionally require the execution of multiple application processes on the same node, which potentially communicate node-internally. Hence, oversimplification by assuming the equivalent behavior of on-node and (real) network communication or assuming the instantaneous data transmission within a node can diminish the accuracy of the network simulations. Thus, it is important to quantify an accurate delay per byte latency value to mitigate the effect. This can be accomplished by establishing the cost per byte for on-node message transferring by timing the roundtrip delay of two processes on the same node sending messages varying in size from 1 byte up to the desired maximum message size. This delay can then be implemented in the simulation system.

Avoiding the common pitfalls will go a long way to improving the correctness of the future network simulation results. Including all relevant aspects of the experiment such as memory/caching latencies or on-node communication establishes a foundation for performing the final step of the validation process—generating and reporting reasonable results from simulations.

7 DOCUMENTING THE OUTCOME

Adequate documentation and reporting of the collected validation data not only ensure repeatability for the validation studies but also are “critical in convincing users of the ‘correctness’ of the model and its results” (Sargent 2005). In the following, we discuss recommended practices for performing measurements and simulations for network model validation and for reporting the acquired data. The data presented in this paper on validating caching effects, using bisection pairing and ping-pong benchmarks, is available on the public gitlab repository: git@xgitlab.cels.anl.gov:mubarak/validation-data.git.

For space-saving reasons, documenting the production of the results can be kept to a higher level, such as network technology, topology type, executed benchmarks, underlying simulation framework, and algorithms used—provided that a publicly accessible repository or long-term archive is referenced with in-depth details, source code, and raw data (Collberg and Proebsting 2016). The in-depth information should include the following items and any additional data potentially affecting the results.

Hardware Specification – Detailed hardware specification of network and attached nodes: switches, links, and NICs, topology configuration, and node configurations, such as CPU and sizes of the memory/storage levels.

Software Details – Deployed software versions and flags: operating system, CPU frequencies, network firmware versions, compiler version and flags, MPI implementation and flags, source code of new or patches for existing benchmarks, and execution commands.

Environment Details – Environment for the performed benchmarks: exclusive access or details about the kind of background noise in the network through resource sharing.

Complementing this “how” information in an archive, we advocate presenting the following information in scientific publications.

Input – Parameter configurations (even if kept at the default) should be reported for benchmarks and simulations.

Metrics – The benchmarks and network model should measure and reports the same metrics; if not, then the deviations—for example, omitting acknowledge messages for simulated communication protocols—must be stated.

Repetitions – A sufficient number of runs is required to allow for statistical analysis, and potential seeds for pseudo-random-number generators should be documented.

Result Compression – The statistical analysis and filtering methods used for the data should be discussed.

Result Presentation – Outlier obfuscation, such as relative comparisons or logarithmic scales, should be avoided. (Hoefler and Belli 2015)

However, extensive input data, the raw data from benchmarks and simulations, data manipulation scripts, and any results omitted but mentioned in the research publication should be included in the public archive. Such information improves confidence in the validation and opens up the possibility for further processing of the data by other researchers. While this list is comprehensive and adequate for the purpose of network model validation studies, it is certainly not complete, given the extent of this topic. For further information, we recommend several works: (Sargent 2005, Hoefler and Belli 2015, Collberg and Proebsting 2016).

8 CONCLUSION

Widespread use of simulations in exploring next-generation HPC interconnects and procurement process of HPC systems is contingent upon the confidence that the end users of simulations have in the outcome predicted by the simulations. Robust and reliable validation of the simulations plays a key role in building this confidence. To this end, we examined a workflow for conducting validation of HPC interconnect simulations and presented ways for making it robust. For each step of the workflow, we described simulation considerations that can improve the results and strengthen the validation study. We also used results generated by the CODES simulation framework as examples to substantiate these claims. Further, we highlighted common mistakes that negatively impact the reliability of validation studies. We hope that these observations and guidelines will assist researchers in conducting convincing validation studies of their simulations.

ACKNOWLEDGMENT

This work was performed under the auspices of the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, under contract number DE- AC02-06CH11357 and Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-733848). The work has used resources from the Argonne Leadership Computing Facility (ALCF).

REFERENCES

Adiga, N. R., M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. 2005, March. “Blue Gene/L Torus Interconnection Network”. *IBM Journal of Research and Development* 49 (2): 265–276.

Agarwal, T., A. Sharma, and L. V. Kalé. 2006, April. “Topology-Aware Task Mapping for Reducing Communication Contention on Large Parallel Machines”. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*.

Ajima, Y., T. Inoue, S. Hiramoto, S. Ando, M. Maeda, T. Yoshikawa, K. Hosoe, and T. Shimizu. 2014, August. “The Tofu Interconnect 2”. In *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*, 57–62.

Besta, M., and T. Hoefler. 2014. “Slim Fly: A Cost Effective Low-diameter Network Topology”. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 348–359.

Brightwell, R., and K. Underwood. 2003. “Evaluation of an Eager Protocol Optimization for MPI”. In *European Parallel Virtual Machine/Message Passing Interface Users? Group Meeting*, 327–334. Springer.

Carothers, C. D., D. Bauer, and S. Pearce. 2002, November. “ROSS: A High-Performance, Low-Memory, Modular Time Warp system”. *Journal of Parallel and Distributed Computing* 62 (11): 1648–1669.

Collberg, C., and T. A. Proebsting. 2016, February. “Repeatability in Computer Systems Research”. *Communications of the ACM* 59 (3): 62–69.

Domke, J., T. Hoefler, and S. Matsuoka. 2014. “Fail-in-Place Network Design: Interaction Between Topology, Routing Algorithm and Failures”. In *Proceedings of the International Conference for High*

Performance Computing, Networking, Storage and Analysis, SC '14, 597–608. Piscataway, NJ, USA: IEEE Press.

Faanes, G., A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard. 2012. "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*, SC '12, 103:1–103:9. Los Alamitos, CA, USA: IEEE Computer Society Press.

Gropp, W., and E. Lusk. 1999. "Reproducible Measurements of MPI Performance Characteristics". In *Proc. of the 6th Eur. PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 11–18.

Hoefler, T., and R. Belli. 2015, November. "Scientific Benchmarking of Parallel Computing Systems". In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'15*, 73:1–73:12: ACM.

Jain, N., A. Bhatele, S. White, T. Gamblin, and L. V. Kale. 2016. "Evaluating HPC Networks via Simulation of Parallel Workloads". In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16. Piscataway, NJ, USA: IEEE Press.

Jiang, N., J. Kim, and W. J. Dally. 2009. "Indirect Adaptive Routing on Large Scale Interconnection Networks". In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, 220–231. New York, NY, USA: ACM.

Kim, J., W. J. Dally, S. Scott, and D. Abts. 2008, June. "Technology-Driven, Highly-Scalable Dragonfly Topology". *ACM SIGARCH Computing Architecture News* 36 (3): 77–88.

Klenk, B., and H. Fröning. 2017. "An Overview of MPI Characteristics of Exascale Proxy Applications". In *32th International Conference on High Performance Computing, ISC'17*: Springer International Publishing.

Mellanox 2013. "Whitepaper: Power Saving Features in Mellanox Products". Technical report.

MPI Forum 2015, June. "MPI: A Message-Passing Interface Standard Version 3.1".

Mubarak, M., C. D. Carothers, R. B. Ross, and P. Carns. 2017, January. "Enabling Parallel Simulation of Large-Scale HPC Network Systems". *IEEE Transactions on Parallel and Distributed Systems* 28 (1): 87–100.

Petrini, F., D. J. Kerbyson, and S. Pakin. 2003. "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q". In *Supercomputing, 2003 ACM/IEEE Conference*, 55–55. IEEE.

Petrini, F., and M. Vanneschi. 1997, April. "k-ary n-trees: High Performance Networks for Massively Parallel Architectures". In *11th International Parallel Processing Symposium*, 87–93.

Sargent, R. G. 2005. "Verification and Validation of Simulation Models". In *Proceedings of the 37th Conference on Winter Simulation*, edited by S. Jain, R. Creasey, J. Himmelspach, K. White, and M. Fu, WSC '05, 130–143.

Won, J., G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott. 2015. "Overcoming Far-End Congestion in Large-Scale Networks". In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 415–427. IEEE.

Yang, X., J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan. 2016. "Watch Out for the Bully!: Job Interference Study on Dragonfly Network". In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, 64:1–64:11. Piscataway, NJ, USA: IEEE Press.

AUTHOR BIOGRAPHIES

Misbah Mubarak is a postdoctoral researcher in the Mathematics and Computer Science Research Division at Argonne National Laboratory. Dr. Mubarak received her Ph.D. in computer science from Rensselaer Polytechnic Institute in 2015. Her email address is mmubarak@anl.gov

Nikhil Jain is a Sidney Fernbach postdoctoral fellow in the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. Dr. Jain received his Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 2016. His email address is jain6@llnl.gov

Jens Domke is a research associate of the Matsuoka Laboratory at the Tokyo Institute of Technology. Mr. Domke received his Diplommathematiker degree from Technische Universität in 2010. His email address is domke.j.aa@m.titech.ac.jp

Noah Wolfe is a graduate student in the Department of Computer Science at Rensselaer Polytechnic Institute. Mr. Wolfe received his B.S. in computer engineering from the University of New Mexico in 2012. His email address is wolfen@rpi.edu

Caitlin Ross is a graduate student in the Department of Computer Science at Rensselaer Polytechnic Institute. Ms. Ross received her B.S. in computer science from the University of North Carolina at Greensboro in 2014. Her email address is rossc3@rpi.edu

Kelvin Li is a graduate student in the Computer Science Department at the University of California, Davis. Mr. Li received his B.S. in computer engineering from the University of California, Davis, in 2009. His email address is kelli@ucdavis.edu

Abhinav Bhatele is a computer scientist in the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. Dr. Bhatele received his Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 2010. His email address is bhatele@llnl.gov

Christopher D. Carothers is a professor of computer science at Rensselaer Polytechnic Institute. Professor Carothers received his Ph.D. from Georgia Institute of Technology in 1997. His email address is chrisc@cs.rpi.edu

Kwan-Liu Ma is a professor of computer science at the University of California, Davis. Professor Ma received his Ph.D. in computer science from the University of Utah in 1993. His email address is ma@cs.ucdavis.edu

Robert B. Ross is a senior computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory. Dr. Ross received his Ph.D. in computer engineering from Clemson University in 2000. His email address is rross@mcs.anl.gov