

# Evaluation of Topology-Aware Broadcast Algorithms for Dragonfly Networks

Matthieu Dorier,<sup>\*</sup> Misbah Mubarak,<sup>\*</sup> Rob Ross,<sup>\*</sup> Jianping Kelvin Li<sup>†</sup>, Christopher D. Carothers,<sup>‡</sup> and Kwan-Liu Ma<sup>†</sup>

<sup>\*</sup>Argonne National Laboratory {mdorier,mmubarak,rross}@anl.gov

<sup>†</sup>University of California, Davis, {kelli,klma}@ucdavis.edu

<sup>‡</sup>Computer Science Department, Rensselaer Polytechnic Institute, chrisc@cs.rpi.edu

**Abstract**—Two-tiered direct network topologies such as Dragonflies have been proposed for future post-petascale and exascale machines, since they provide a high-radix, low-diameter, fast interconnection network. Such topologies call for redesigning MPI collective communication algorithms in order to attain the best performance. Yet as increasingly more applications share a machine, it is not clear how these topology-aware algorithms will react to interference with concurrent jobs accessing the same network. In this paper, we study three topology-aware broadcast algorithms, including one designed by ourselves. We evaluate their performance through event-driven simulation for small- and large-sized broadcasts (in terms of both data size and number of processes). We study the effect of different routing mechanisms on the topology-aware collective algorithms, as well as their sensitivity to network contention with other jobs. Our results show that while topology-aware algorithms dramatically reduce link utilization, their advantage in terms of latency is more limited.

## I. INTRODUCTION

As the number of cores in petascale and post-petascale supercomputers increases, traditional low-radix networks such as torus fail to meet cost and performance requirements of HPC infrastructures. Consequently, a number of novel, high-radix topologies such as Dragonflies [1] and variants [2], [3] have been proposed.

The Dragonfly topology is a two-tiered direct topology consisting of groups of routers connected to terminals. Routers belonging to the same group form an all-to-all network, while groups themselves act as high-radix, virtual routers connected in an all-to-all manner. This design enables a cost-efficient, low-diameter network. Indeed, any two terminals in such a topology are at most five hops away from one another. Dragonfly topologies have been used in a number of machines, such as the Cori and Edison supercomputers at NERSC [4], and will be used by the future Theta machine at ANL [5].

Yet such a topology poses new challenges. High-radix routers and low network diameter make job isolation more difficult and force network resources to be shared across independent jobs. Communication interference between jobs [6], [7], [8], one of the main causes of performance variability in HPC applications [9], becomes increasingly difficult to avoid. This interference issue is further amplified by the fact that Dragonfly networks perform best under uniform random traffic [10], which motivates using random placement of processes across the network and nonminimal or adaptive routing strategies [11], [1].

While random placement of processes and appropriate routing strategies can help mitigate interference, collective

communication provides another opportunity for reducing interference. Topology-aware algorithms can minimize traffic between groups, reducing utilization of these key links. These algorithms must be evaluated not only for jobs spanning the full network and running individually but also for random allocations of any size and in the presence of background traffic.

Broadcast algorithms adapted to the Dragonfly network have been proposed in the past few years [12], [13]. Xiang and Liu [12] proposed the *group first* and *router first* algorithms, which we call GLF (global links first) and LLF (local links first) in this paper. An algorithm similar to LLF was also proposed for the PERCS topology by Jain et al. [13]. These works, however, focus on a single job spanning the full machine.

In this paper, we evaluate three topology-aware broadcast algorithms for the Dragonfly topology: LLF, GLF, and FOREST. FOREST is an algorithm designed by ourselves that mixes LLF and GLF to try to overcome the limitations of both. We compare these algorithm to the binomial tree algorithm (TREE) implemented in MPICH. Using the CODES framework [14] for discrete-event simulations of the Dragonfly network, we demonstrate their performance across a range of allocation sizes, for different types of routing methods, different data sizes, and with the presence of background traffic. Our results show that, *while topology-aware algorithms dramatically decrease link utilization, they do not necessarily decrease the execution time*. Deeper studies should be conducted to find out what makes each algorithm perform particularly well (or particularly poorly) in a given context. For example, LLF presents a high run time and a large variability for small allocations, compared with other algorithms.

The rest of this paper is organized as follows. In Section II we present the background and related work for our study. Three topology-aware algorithms, LLF, GLF, and FOREST, are presented in Section III. Section IV evaluates these algorithms through packet-level network simulations using the CODES simulation framework. Section V presents our conclusions and briefly discusses avenues for future work.

## II. BACKGROUND AND RELATED WORK

In this section we first describe the Dragonfly topology, including common problems related to routing, task placement, and communication interference. We then describe common nontopology-aware broadcast algorithms, showing their potential drawbacks on a Dragonfly network.

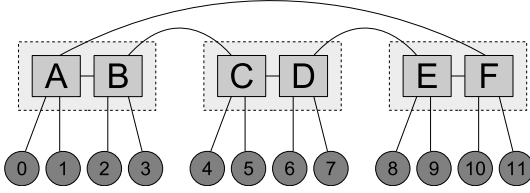


Fig. 1: Illustration of a Dragonfly topology with  $g = 3$ ,  $a = 2$ ,  $p = 2$ , and  $h = 1$ .

### A. Dragonfly networks

1) *Overview:* The Dragonfly topology is a two-tier network topology composed of  $g$  groups of  $a$  routers. Each router is connected to a number  $p$  of terminals through terminal links. Routers in a given group are all connected to one another through local links in an *all-to-all* topology. Each router in a given group is also connected through global links to  $h$  routers belonging to other groups. In our study, we consider a Dragonfly topology in which, for any two groups  $G$  and  $G'$ , there is exactly one router  $R$  belonging to  $G$  and one router  $R'$  belonging to  $G'$  such that  $R$  shares a global link with  $R'$ . This translates into  $h = \frac{g-1}{a}$  and corresponds to the architecture originally proposed by Kim et al. [1].

2) *Routing on a Dragonfly:* A message sent from a terminal  $T_1$  to a target terminal  $T_2$  connected to the same router  $R_1$  needs to cross two terminal links (for example, terminals 0 and 1 in Figure 1). If  $T_2$  is in the same group but connected to  $R_2 \neq R_1$ , the message has to cross one terminal link ( $T_1 \rightarrow R_1$ ), one local link ( $R_1 \rightarrow R_2$ ), and one terminal link ( $R_2 \rightarrow T_2$ ). This is the situation of terminals 0 and 2, for example. If the target terminal is in another group, the message first needs to be routed from  $R_1$  to a router  $R'_1$  that has a global link to a router  $R'_2$  in the target group, then from  $R'_2$  to  $R_2$  through a local link, and finally to the target terminal (note that  $R'_1$  may be equal to  $R_1$  if  $R_1$  has a global link to a router in the destination group, and similarly  $R'_2$  may be equal to  $R_2$ ). Hence in this situation the message crosses two terminal links ( $T_1 \rightarrow R_1$  and  $R_2 \rightarrow T_2$ ), one global link ( $R'_1 \rightarrow R'_2$ ), and anywhere from zero to two local links ( $R_1 \rightarrow R'_1$  if  $R_1 \neq R'_1$  and  $R'_2 \rightarrow R_2$  if  $R_2 \neq R'_2$ ). In Figure 1, for example, a message sent from terminal 0 to terminal 11 needs to cross two terminal links and one global link. A message sent from terminal 0 to terminal 4 needs to cross one local link in addition to the global link and the two terminal links. A message sent from terminal 0 to terminal 7 needs to cross one more local link.

The above routing method is called *minimal*, or *direct*. Nonminimal (or indirect) and adaptive routing strategies using Valiant's algorithm [15] and variants [16] have been proposed in order to randomize the intergroup traffic and avoid congestion [1], [17], [18]. When sending a message from a terminal to another in different groups, nonminimal routing randomly selects an intermediate group through which the packet will transit. This technique has the benefit of making the network traffic look more like a uniform random traffic pattern, for which the Dragonfly topology is well suited [10]. Adaptive routing consists of switching between minimal and nonminimal routing depending on whether congestion is detected.

Although the adaptive routing mechanism proposed by Kim et al. [1] along with the Dragonfly topology can help avoid congestion, Prisacari et al. have shown that it still has limited capabilities to really randomize the traffic [10]. Besides, nonminimal routes increase the number of hops required to transfer a packet, which increases the latency and energy consumption. Relying only on routing mechanisms to address congestion issues is therefore not sufficient; we need to develop topology-aware communication algorithms that effectively minimize global and local link utilization.

3) *Job and process placement:* The intuition to avoid sharing links across jobs would be to isolate jobs on as few routers as possible [6], [8]. Process placements strategies have been proposed by Bhatele et al. [18] that depend on the dominant communication patterns of applications. The authors show that while topology-aware task placement strategies can achieve better performance under minimal routing, nonminimal routing makes such optimization unnecessary, at the cost of higher latencies.

Jain et al. [19] have shown that random process placement, along with adaptive or nonminimal routing, helps spread the traffic across the network and avoid hot spots. Yet randomizing task placement is still not sufficient to fully randomize the network traffic, as shown by Prisacari et al. [20].

As a result of random process placement, most of the messages generated by any non-topology-aware collective communication algorithms transit through a global link. This problem is further amplified by nonminimal and adaptive routing, which select a random intermediate group through which to route messages, therefore doubling the utilization of global links. If the goal is to minimize link utilization, a key focus must be on leveraging topology information to avoid unnecessary traffic outside groups.

4) *Taking advantage of Dragonfly:* Several observations can be made from the design of the Dragonfly topology and from its routing method. First, sending data to a process in a different group requires traversing more hops than does sending data to a process in the same group. In a production system where multiple applications run concurrently, it may be desirable to keep intergroup communications and global link utilization to a minimum in order to avoid network congestion and performance variability. Doing so will also minimize the impact of routing decisions on the performance of the algorithm. Indeed, as fewer intergroup communications are performed, there will be fewer opportunities for nonminimal or adaptive routing to increase the latency by making messages transit through a randomly selected group. Consequently, it will also minimize the communication volume.

The second observation is that once a process in a group has received its data, it can broadcast that data internally within the group without using global links. Hence the number of messages traversing global links in a topology-aware broadcast can be reduced to one per group in which the application runs.

Third, provided that multiple terminals in a group have received the data, the data can be sent in parallel to terminals of other groups through distinct global links.

These three observations have driven the design of some topology-aware collective algorithms [12], [13], which attempt to minimize the transfer through global links. However, one

consideration that has been left aside in these previous works is the fact that Dragonfly networks use high-radix routers. Sending data only across local links, or only across global links during one step of a broadcast, prevents the router from leveraging its input and output ports and buffers associated with unused links. *Counterintuitively, minimizing global link utilization might therefore not be the best way to get the shortest run time of the collective algorithms on a Dragonfly network.* The evaluation we conduct in this paper precisely aims to explore this tradeoff.

### B. Non-topology-aware broadcast algorithms

1) *State of the art:* Current implementations of MPI such as MPICH [21] provide several broadcast methods that are selected based on the size of the message and the number of processes involved. For small messages ( $size \leq 12,288$  bytes), MPICH uses a binomial tree algorithm. For long messages, MPICH splits the data and performs a *scatter* across processes followed by an *allgather*. The *scatter* phase uses a binomial-tree algorithm. The *allgather* phase uses a recursive-doubling algorithm [22] for medium-size messages ( $size \leq 524,288$ ) and power-of-2 number of processes. For long messages and for medium-size messages and non-power-of-2 number of processes the *allgather* phase uses a ring algorithm. OpenMPI also provides a number of broadcast algorithms, including a binomial tree algorithm [23]. Contrary to MPICH, which selects the algorithm based on data size thresholds and number of processes independently of the machine, OpenMPI's algorithms are selected based on prior benchmarking and an encoded decision function [24]. None of these algorithms, however, take into account the network topology and job placement.

Zhou et al. [25] evaluated their new broadcast algorithm on a Dragonfly-based machine (Cray XC40). The algorithm itself is not topology-aware, however, since it does not take into account the location of processes in the topology. Topology-aware broadcast algorithms for topologies other than Dragonfly (in particular, torus) have been proposed in the past [26], [27], [28].

In our study, we chose to compare topology-aware algorithms with the binomial-tree algorithm (from this point onward, we will refer to it as TREE). The two other algorithms provided by MPICH are composed of *scatter* and *allgather* phases that could themselves be redesigned for the Dragonfly network. Such a design is left for future work.

## III. BROADCASTING IN A DRAGONFLY NETWORK

In this section, we present three algorithms —LLF, GLF, and FOREST— specifically designed for the Dragonfly topology. These algorithms aim at minimizing the utilization of global links in order to prevent congestion with other applications running on the platform. LLF and GLF have been presented by Xiang et al. [12] under the terms “router first” and “group first,” respectively. LLF has also been presented by Jain et al. [13] for the PERCS network topology, another two-tier direct network topology. FOREST is an algorithm we designed as a hybrid between GLF and LLF, aiming at overcoming the limitations of LLF in the context of small allocations or of

small number of routers per group compared with the number of groups used by the job.

### A. Terminology and topology information

1) *Notation:* In the following we use the term *root terminal* to refer to the terminal containing the process from which the broadcast is initiated. We call the router to which this terminal is connected the *root router*. Similarly, we designate *root group* the group that contains this router. We call *remote groups* the groups that do not contain the root router/terminal.

2) *Querying topology and placement information:* The algorithms presented hereafter are based on the assumption that some information about the topology and the process placement can be obtained. Each terminal, router, and group is assigned a unique terminal, router, and group *id*, respectively.

a) *Process placement:* Any process can have access to the terminal *id* of any other process belonging to the same application.

b) *Topology:* From a terminal *id*, any process can retrieve the *id* of the router connected to this terminal and the *id* of the group containing this router.

c) *Connectivity:* Given two group *ids*  $G_1$  and  $G_2$ , any process (not necessarily part of  $G_1$  or  $G_2$ ) can retrieve the *ids* of the routers  $R_1$  in  $G_1$  and  $R_2$  in  $G_2$  that share a common global link.

Such information can reasonably be obtained from tools such as the Portable Network Locality (netloc) [29].

### B. Algorithm 1: Local-Links-First (LLF)

The LLF algorithm consists of four steps.

- 1) The root process sends its data to at least one process in every router of its group that is involved in the broadcast. This first broadcast is done by using a binomial tree. The goal of this step is to have as many routers as possible having a terminal that holds the data in the root group. This step uses terminal and local links.
- 2) Processes that have the data (including the root) in the root group send it to one process in each remote group involved in the job. The receiving processes are chosen based on their distance to the sending processes. The source processes send data in serial to the destination processes. The goal of this step is to send the data to one terminal in each remote group. This step is the only one that uses global links.
- 3) Processes that have the data in remote groups broadcast it inside their group. This is done by sending it to one process in each router by using a binomial tree broadcast. The goal here is that every router in every group has at least one terminal holding the data. This step uses only terminal and local links.
- 4) Each process that has the data in each router proceeds with sending it to the rest of the processes connected to the same router, using a binomial tree broadcast. This step uses only terminal links.

This algorithm takes advantage of the parallelization across global links. After the root process has sent its data to other processes in its group in step 1, these processes can independently send their data to remote groups without interfering with one another.

**Degraded placement cases.** The worst-case placement for this algorithm appears when the root group contains a single process: the root of the broadcast. In this situation, the root process will have to send its data to one process of each group in serial by itself. This worst-case scenario can be avoided, for example, by first sending the data to a process in a group that has a large number of routers used by the job. This process will serve as the new root of the LLF broadcast. An alternative consists of first broadcasting the data across global links by using a binomial-tree broadcast and then broadcasting only inside each group. This solution is provided in the next section.

### C. Algorithm 2: Global-Links-First (GLF)

The GLF algorithm consists of three steps.

- 1) The root process broadcasts the data to one process in each group. This first step is completed by using a binomial-tree algorithm. This step uses terminal, local, and global links. The goal is to have the data present in one process of every group.
- 2) Each receiving process of the first step, as well as the root, becomes the root of an intragroup broadcast to send the data to one process in each router. This step is performed by using a binomial-tree algorithm. It uses terminal and local links. The goal is to send the data to at least one process in every router.
- 3) Each receiving process in each router becomes the root of a broadcast across processes connected to the same router. This last step is also done by using a binomial-tree broadcast. This step uses only terminal links.

**Degraded placement cases.** The GLF algorithm does not suffer from an unbalanced number of processes per group. For a given number of processes, however, assuming that it is more costly to send through global links than through local links, allocations that span fewer total groups will tend to show higher performance (i.e., lower latency). The worst case occurs when the largest number of groups is involved (forcing the deepest tree for the first broadcast). Remaining processes can be placed in one group spanning the largest number of routers (for instance by first placing one process per router) to force the deepest possible tree in the second step as well. By placing the remaining processes in all the terminals of at least one router, one can force the deepest tree for the third step as well.

### D. Algorithm 3: FOREST

FOREST is an algorithm we have proposed to overcome the limitation of LLF in situations where LLF would perform the worst, that is, when a small number of processes in the root group have to send their data to comparatively many destinations in remote groups in step 2. In this worst-case situation, the fact that source processes send data in serial with LLF makes the root group become a bottleneck and impacts performance. Instead, FOREST builds several binomial trees (hence the name of the algorithm) across the groups, one per source process in the root group, to perform this step. The next steps (sending data within each group and within each router) remain the same.

### E. Analytical estimation of link utilization

In this section, we provide an estimation of link utilization for each algorithm. We start by computing some probabilities on the number of links required to reach one terminal from another, assuming minimal routing.

1) *Probability of crossing a global link:* Given two distinct terminals  $T_1$  and  $T_2$ , the probability that they belong to the same group is  $P = \frac{ap-1}{gap-1}$ . Hence the probability that sending a message from  $T_1$  to  $T_2$  requires the use of one global link is

$$P(1 \text{ global link}) = 1 - \frac{ap-1}{gap-1}.$$

2) *Probability of crossing local links:* A message sent from  $T_1$  to  $T_2$  will cross 0, 1, or 2 local links. The probability that it crosses 0 local links is

$$P(0 \text{ local link}) = \frac{p-1}{gap-1} + \frac{hp}{gap-1}.$$

The first term corresponds to the probability that the two terminals belong to the same router. The second term corresponds to the probability that they belong to different groups but that the routers to which they are connected share a global link. The probability that the message has to cross 1 local link is

$$P(1 \text{ local link}) = \frac{(a-1)p}{gap-1} + 2\frac{h(a-1)p}{gap-1} = (2h+1)\frac{(a-1)p}{gap-1}.$$

The first term corresponds to the probability that the terminals belong to the same group but not the same router. The second term corresponds to the probability that they belong to different groups and that one local link is required in one of the groups but not in the other.

The probability that the message has to cross 2 local links is

$$P(2 \text{ local links}) = 1 - P(0 \text{ local link}) - P(1 \text{ local link}).$$

3) *Estimating link utilization for TREE:* Assuming a random job allocation of  $n$  terminals, TREE being non-topology-aware, any message sent from a terminal to another during a broadcast will have to cross an average number of global links equal to

$$E(\text{global links}) = P(1 \text{ global link}) = 1 - \frac{ap-1}{gap-1}.$$

Thus the expected total number of global links used is

$$E(\text{total global links}) = (n-1)\left(1 - \frac{ap-1}{gap-1}\right).$$

The average number of local links used by a single message is equal to

$$\begin{aligned} E(\text{local links}) &= P(1 \text{ local link}) + 2P(2 \text{ local links}) \\ &= 2 - \frac{2+p(1+2ha+a)}{gap-1}. \end{aligned}$$

Hence the expected total number messages transiting through local links is

$$E(\text{total local links}) = (n-1)\left(2 - \frac{2+p(a+2ha+a)}{gap-1}\right).$$

4) *Estimating link utilization for LLF, GLF, and FOREST:* Topology-aware algorithms attempt to make a minimal use of links. One need not rely on probabilities to compute their link utilization, and the following analysis is common to all of them. All the algorithms will use only one global link per remote group, leading to a global link utilization bounded by  $g - 1$ . During the step where data is sent to remote groups, at worst  $2(g - 1)$  messages transit in local links. Once one terminal in each group has the data, broadcasting within one group requires sending at worst  $a - 1$  messages through local links. The final step does not require any local or global links. Hence the number of messages crossing global links is bounded by  $g - 1$ , and the number of messages crossing local links is bounded by  $2(g - 1) + g(a - 1)$ .

#### IV. EVALUATION

In this section we evaluate the four algorithms presented above.

##### A. Methodology

1) *Preliminary study:* To evaluate the algorithms, we first conduct a preliminary study, looking only at the generated communication pattern. We determine the number of links used and the makespan, without taking into account the particular behavior of routers and the congestion that could occur between different processes running the algorithm, or interference with other jobs.

2) *Event-driven simulations:* We then use the CODES network simulation framework [14] to simulate the behavior of the network during the execution of each algorithm at packet-level detail. CODES is based on the ROSS parallel discrete event simulator [30]. It has already demonstrated its accuracy in modeling high-performance networks such as torus [31] and has been used to evaluate various routing strategies on Dragonfly networks [32]. CODES has also been used to evaluate the performance of MPI collective communications on torus and Dragonfly networks [33], but the collective algorithms were modeled simply by using a fan-in/fan-out communication pattern instead of point-to-point messages generated by the actual algorithms, as done in our work.

3) *Background traffic:* After evaluating the network performance with no other traffic than the collective communication, we observe how the algorithms react to background traffic in the network (Section IV-D). The background traffic is generated as follows. Each terminal not participating in the broadcast has a payload of  $x$  MB to send ( $x$  being chosen such that the background traffic lasts at least until the broadcast has completed). It sends the payload in 1024-byte messages addressed to another randomly selected terminal not participating in the broadcast. The interarrival time of such send operations follows an exponential distribution of mean 750 nanoseconds. Given that uniform random traffic enables peak performance on a Dragonfly network and that jobs are consequently allocated randomly, such a traffic pattern is representative of background traffic observed in Dragonfly networks. This methodology for generating random traffic is the same as that employed by Besta and Hoefler for the simulation of Slimfly networks [34] and by Kim et al. for the Dragonfly topology [1].

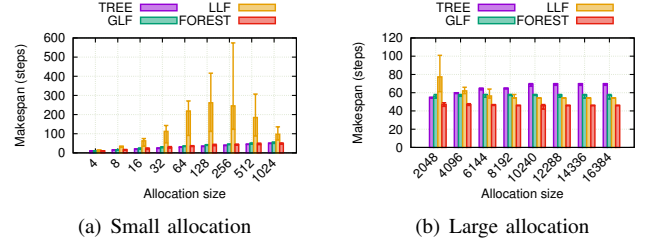


Fig. 2: Makespan of the broadcast algorithms (average over 20 random allocations, error bars representing maximum and minimum).

4) *Parameter space:* In our CODES-based simulations, we consider Dragonfly networks of 5,256 and 16,512 terminals, using minimal, nonminimal, and adaptive routing. We evaluate the broadcast of small messages (1 KB) and large messages (1 MB) using the TREE, LLF, GLF, and FOREST algorithms, with and without background traffic. Each simulation yields the total run time of the algorithm (difference between the time at which the last process receives the data and the time at which the root issues its first send), the average number of hops encountered by packets, and the average and maximum latency of packets from source to destination processes.

##### B. Preliminary study

We consider a Dragonfly network composed of  $g = 129$  groups of  $a = 16$  routers each (allowing the 16 routers in a group to share equally the connections to 128 remote groups). Each router is connected to  $p = 8$  terminals, for a total of 16,512 terminals in the network. Each router is connected to all other routers in its group and to  $h = 8$  routers in remote groups.

The makespan is here measured by the number of links. We assume that (1) any send operation lasts as many time units as the number of links the message has to traverse to reach its destination (no matter the nature of the links traversed) and (2) a sending process has to wait for the message to reach its destination before being able to send another message. Hence, for example, sending a message from a terminal to another through two routers requires 3 time units (terminal 1  $\rightarrow$  router 1  $\rightarrow$  router 2  $\rightarrow$  terminal 2). This constitutes a worst-case scenario. In practice, a *send* would complete as soon as the message has fully crossed the first link and is buffered in the first router.

For all these experiments, we run the algorithms 20 times on randomly generated allocations of given sizes. We distinguish small job allocations (spanning up to 1,024 terminals) from large job allocations (2048 and more).

1) *Makespan:* Figure 2 shows the makespan of the four algorithms. For small allocations, LLF requires a long makespan to complete. This makespan is also highly variable. This is because the root terminal is more likely to be isolated in its group and will send its data to representatives of other groups one by one in series. This problem was not highlighted in previous work where LLF was presented, because it was evaluated in full-machine allocations, a situation that allows maximum parallelism in step 2. This problem does not appear

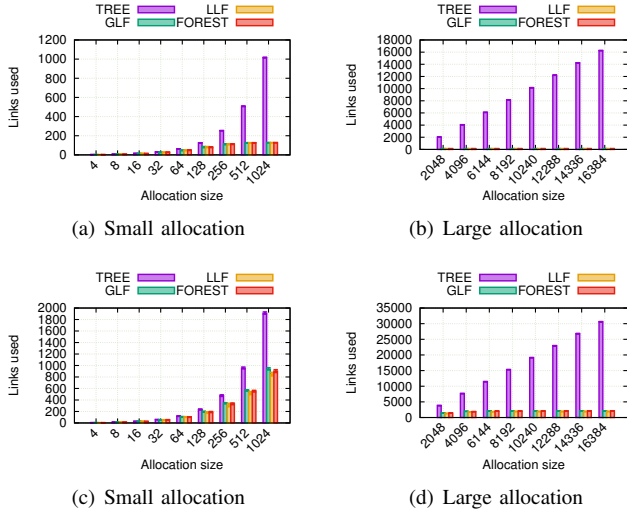


Fig. 3: Total number of messages transiting through global (a,b) and local (c,d) links during the broadcast.

with GLF, FOREST, and TREE. In larger allocations, the root group contains more processes belonging to the job, allowing better parallelism when sending from the root group to other groups. Hence, LLF becomes better than GLF and TREE. FOREST goes one step further by replacing step 2 of LLF by parallel tree-based broadcasts, which further decreases the makespan.

2) *Link utilization*: Figure 3 shows the total number of messages transiting through global and local links. Because of their design, LLF, GLF, and FOREST never use global links more than once per remote group involved in the job (128 here), since each group receives the data only once. The global link utilization is therefore bounded to 128 regardless of the allocation size with topology-aware algorithms. The TREE algorithm has a global link utilization proportional to the allocation size, as expected by the formulas provided in Section III-E. We also observe that like global link utilization, the local link usage remains small and bounded for topology-aware algorithms, while increasing proportionally to the allocation size with the TREE algorithm.

Note that the numbers of terminal links used for all the algorithms are the same, since each process has to receive the data once, an operation that requires always crossing two terminal links.

According to the formulas provided in Section III-E, the number of local links used by TREE for a 10240-terminal allocation is 19,122, and the number of global links is 10,160. Those numbers match our experimental results. The upper bounds for topology-aware algorithms are 128 global links and 2,191 local links. These upper bounds are respected in our experiments.

3) *Visual analysis*: Figure 4 displays the traffic generated during one of the runs for each algorithm (each figure has been generated for a different random allocation, and the root of the broadcast changes in each figure) for a broadcast across 1,024 processes on a 16K network. Terminals are placed on a circle, organized by router and by group. This figure

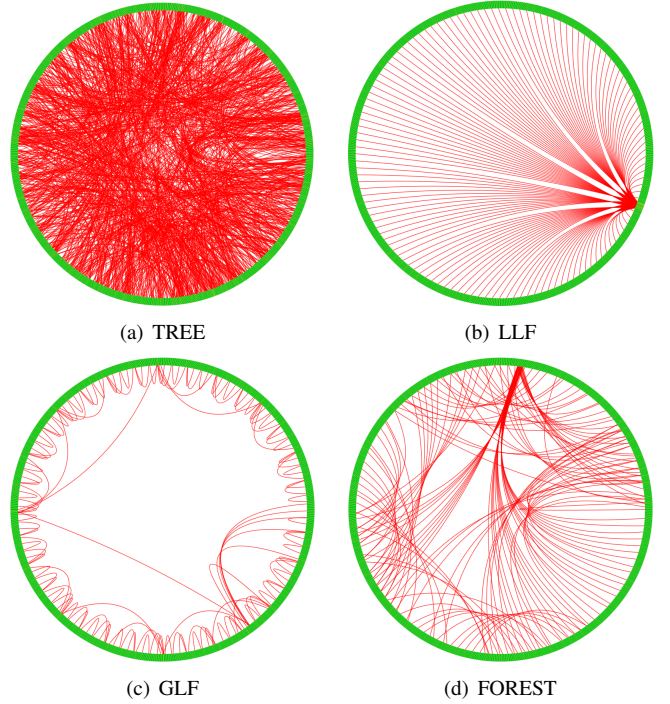


Fig. 4: Visualization of the network traffic through global links when broadcasting data from 1024 terminals on a 16K-terminal Dragonfly network. Note that LLF, GLF, and FOREST all use the same number of global links (128).

shows that as predicted in Section IV-B2, the TREE algorithm generates a lot of traffic across global links. When executing LLF, all global-link traffic originates from the root group. This unbalanced traffic pattern may, however, become a problem in the presence of background traffic. The tree formed by representative terminals of each participating group in the GLF algorithm appears clearly in Figure 4c. Trees executed in parallel and originating from the root group are also clear in Figure 4d.

4) *Preliminary observations*: The above study seems to indicate that for allocations of up to 1024 terminals, the non-topology-aware algorithm has the lowest makespan, despite using several orders of magnitude more links. For larger terminal counts, FOREST presents the lowest makespan. LLF presents a very high and variable makespan for small allocations, suggesting that contrary to what is stated in papers that have introduced it [12], [13], this algorithm has limitations that deserve further considerations.

5) *Shortcomings of the preliminary study*: The evaluation conducted in the previous subsections gives a rough idea of the behavior of each algorithm but may be far from the reality. It assumes that a *send* completes when the receiving terminal has received all the data, which corresponds to an upper bound on the transfer time for large data sizes. In practice, an *MPI\_Send* is allowed to return when the data has been pushed to a local buffer on the network interface, allowing the terminal to issue other *send* operations to other destinations without waiting. The preliminary evaluation did not take into consideration the contention for links or buffers in routers, nor the routing



strategy employed by the network, the fact that messages are further divided into packets (that may take different routes in case of nonminimal or adaptive routing), or the potential presence of other jobs contending for network resources. The next sections address these shortcomings by using the CODES network simulator to evaluate the algorithms at packet level and accurately reproduce the behavior of all network entities.

### C. Packet-level simulations using CODES

We consider the same network of 16,512 terminals as in the previous section. The bandwidth for the global, local, and terminal links in the Dragonfly are respectively 4.7 GB/s, 5.25 GB/s, and 5.25 GB/s, inspired by the CrayXC30 architecture [14], [35]. The virtual channel capacity is 16 KB for terminal links and local links and 32 KB for global links so that 32, 32, and 64 full-sized packet can fit into the terminal, local, and global virtual channels, respectively. Each MPI message gets split into 512-byte packets for transportation over the network. For MPI messages that are smaller than 512 bytes, the exact message size is transported over the network as a data packet. The application broadcasting runs alone on the network on a randomly generated allocation of terminals. Five runs (each with a different random allocation) are executed for each allocation size. We report the median execution time of these runs, as well as the minimum and maximum as error bars.

Figure 5 presents the results for the broadcast of 1 KB of data, in terms of run time (a,d,g), average number of hops per packet (b,e,h), and average and maximum latency of each packet (c,f,i). Experiments are performed with the three different types of routing mechanisms: minimal, nonminimal, and adaptive. These results show that both GLF and FOREST exhibit a lower run time than does TREE. As expected, LLF exhibits a larger run time and larger variability at small scale but becomes the best algorithm starting from 1,024 terminals. The results also show that as the allocation size grows, topology-aware algorithms take advantage of the topology and reduce the average number of hops. We can see that LLF and GLF both minimize the global link utilization by the fact that regardless of the routing strategy, the average hop count converges toward 1, while the average hop count of TREE remains at 4, 6, and 5, for minimal, nonminimal, and adaptive routing, respectively.

Figure 6 presents the results for a broadcast of 1 MB of data. The picture here differs significantly from that of a 1 KB broadcast. The TREE algorithm performs better than topology-aware algorithms for allocations of up to 2048 terminals. FOREST then becomes the best algorithm. LLF presents a much higher and more variable run time for small allocations. It then outperforms TREE for allocations of 10,240 terminals and more but remains worse than GLF and FOREST.

Going from minimal to nonminimal routing has a positive effect on LLF. Note that because of the change of scale from Figure 6a to 6c, the runtime of GLF, TREE, and FOREST may seem to have increased. This is not the case. They have in fact remain very similar. The reason why nonminimal routing boosts the performance of LLF is that the packets that constitute messages can take different routes in parallel. This comes at the price of increased link utilization. Still,

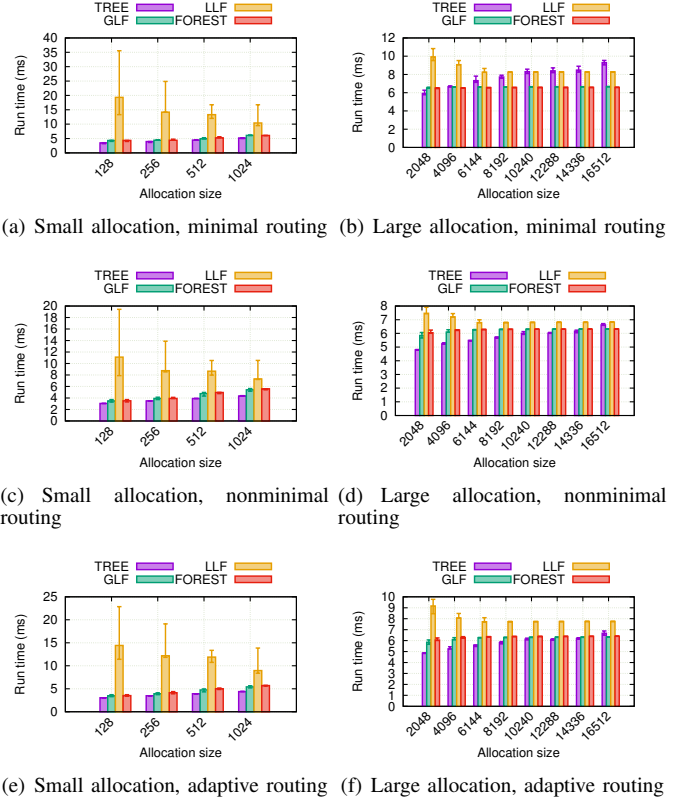


Fig. 6: Run time when broadcasting 1 MB on a 16K terminal Dragonfly network with different routing mechanisms. The job runs alone on the machine. Figures show the median, minimum, and maximum across 5 executions for each job size.

LLF remains relatively inefficient under adaptive routing. In general, our results here show no advantage of topology awareness in terms of run time. In particular, all the algorithms have similar performance for allocations of more than 2,048 terminals under nonminimal routing.

While TREE remains better (or at least competitive) in most cases for broadcasting large amounts of data, we note that in current implementations of MPI, such data sizes are not handled by a tree-based broadcast but by a scatter followed by an all-gather, which breaks the data into chunks that can be sent in parallel to multiple destinations and recomposed later. Such a strategy, particularly interesting in the context of a Dragonfly topology, could be applied with topology awareness to develop better algorithms for large data sizes. Such algorithms are left for future work.

### D. Impact of background traffic

The low diameter of Dragonfly networks, along with the random allocation strategy, makes multiple jobs running on the platform share the same links and contend for the same network resources. In this context, therefore, one must evaluate how topology-aware algorithms perform in the presence of background traffic. Because of the long run times and high memory footprint of packet-level simulations involving background traffic (up to 10 hours), we use a smaller network

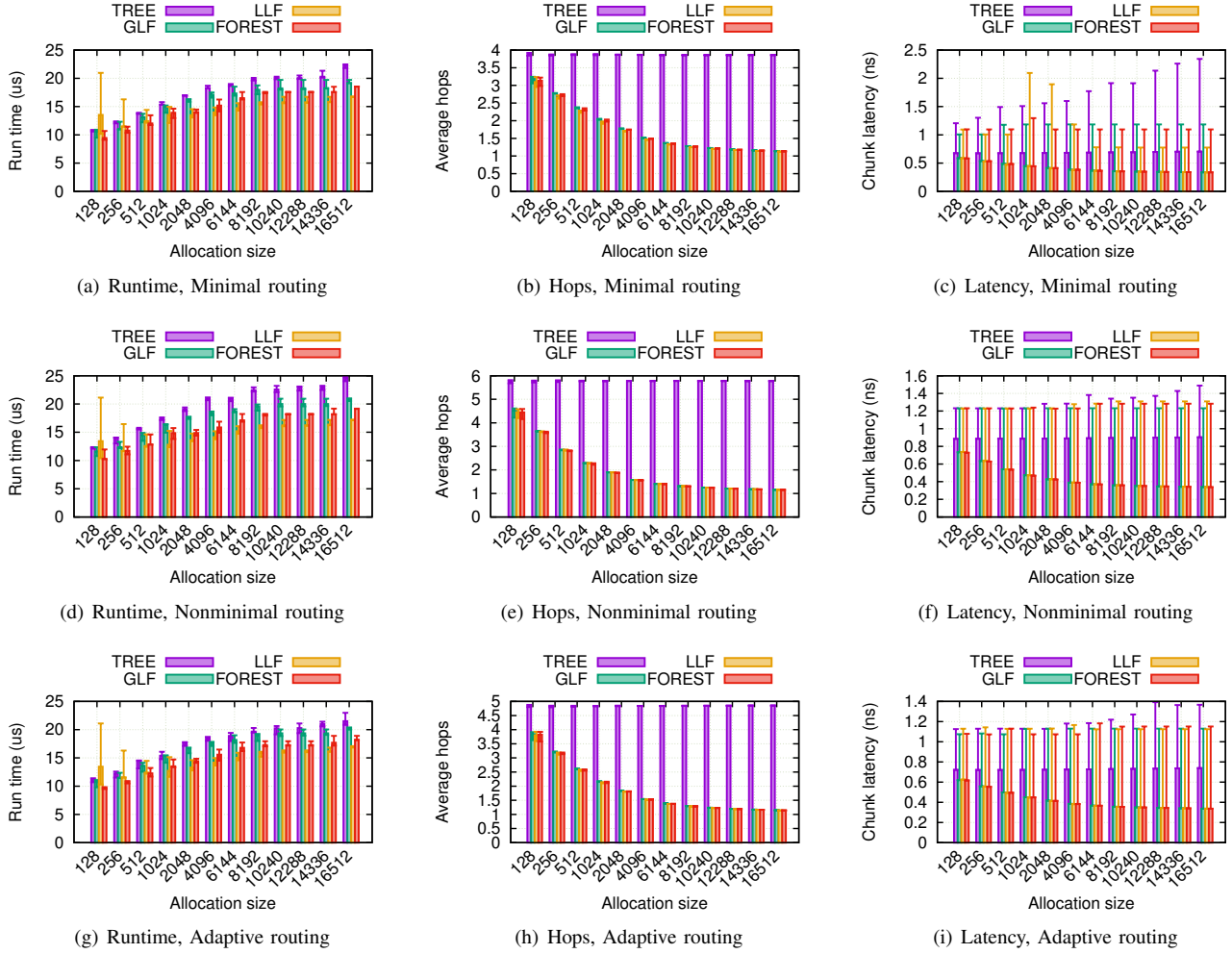


Fig. 5: Run time, average packet hops, and average and maximum packet latency when broadcasting 1 KB on a 16K terminal Dragonfly network with different routing strategies. The job runs alone on the network. Figures show the median, minimum, and maximum across 5 executions for each job size.

topology and, for broadcasts of 1 MB, execute only one run for each allocation size.<sup>1</sup> The network is composed of 73 groups of 12 routers. Each router is connected to 6 routers of remote groups and to 11 routers in its group, and to 6 terminals, for a total number of terminals of 5,256 ( $a = 12$ ,  $p = 6$ ,  $h = 6$ ,  $g = 73$ ).

Figure 7 presents the run time of each algorithm for a 1 KB broadcast. It shows that even in presence of background traffic, FOREST, LLF, and GLF perform better than TREE, with LLF outperforming the other algorithms in nearly all situations. Note that while the execution time increases with the allocation size when no traffic is added, it decreases when adding traffic. The reason is that as the allocation size increases, the number of terminals generating background traffic decreases; thus the effect of background traffic diminishes.

Figure 8 shows the run time for a 1 MB broadcast. Without traffic, the same conclusions can be made from the 5K terminal network as from the 16K terminal network: LLF

performs poorly, especially at small scale, and TREE performs systematically the best when using nonminimal and adaptive routing. When adding background traffic, the TREE algorithm exhibits a lower latency in most cases again, regardless of the routing strategy employed.

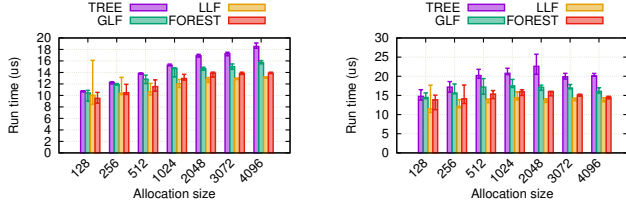
## V. CONCLUSION

In this paper, we have evaluated three topology-aware algorithms and one non-topology-aware algorithm for broadcasting on a Dragonfly network. Our experimental campaign conducted through event-driven simulations using the CODES framework explored the effect of varying the allocation size, the data size, and the routing strategy. Additionally, we studied the performance of these algorithms in the presence of background traffic.

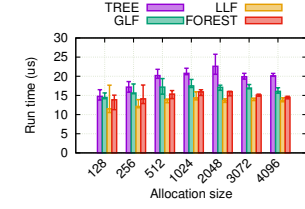
1) *Lessons learned:* The advantage of topology awareness in collective algorithms in terms of link utilization is clear. Our study shows that topology-aware broadcasts have a number of messages crossing global links that is bounded by the number of groups involved, while this number increases proportionally

<sup>1</sup>A subset of the experiments was run at larger scale, and results are similar to those observed at this scale.

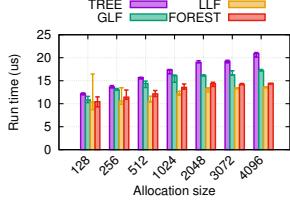




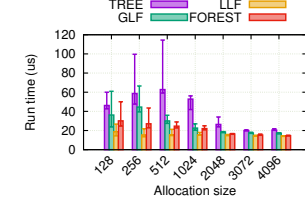
(a) Minimal routing, no background traffic



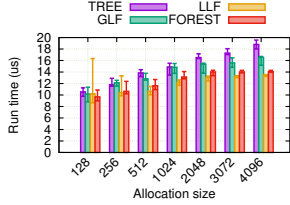
(b) Minimal routing, with background traffic



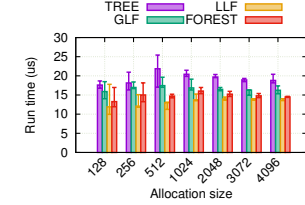
(c) Nonminimal routing, no background traffic



(d) Nonminimal routing, with background traffic



(e) Adaptive routing, no background traffic

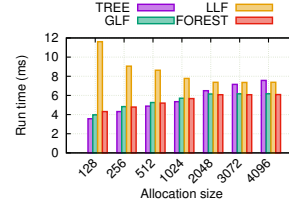


(f) Adaptive routing, with background traffic

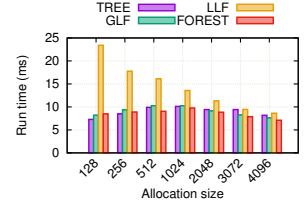
Fig. 7: Run time of a 1 KB broadcast on a 5K terminal Dragonfly network, with and without background traffic and with different routing mechanisms.

to the allocation size for a non-topology-aware broadcast. However, the fact that an algorithm is topology-aware does not necessarily makes it perform better in terms of latency than non-topology-aware ones. This is the main lesson we learned from our study with topology-aware broadcasts. Other lessons include the following.

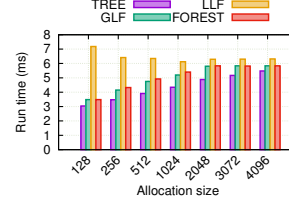
- 1) The size of the job allocation has an impact on the performance of topology-aware broadcasts. Because of the random task placement employed on Dragonfly networks to randomize traffic, small jobs end up spread across a large number of groups. In this context, trying to improve the parallelism across global links, as done with LLF, may lead to larger execution time than with state-of-the-art algorithms such as binomial trees.
- 2) For broadcasts of large data sizes, the non-topology-aware algorithm performs better than (for small jobs) or as well as (for large jobs) topology-aware algorithms. This performance is explained by the fact that non-topology-aware algorithms have a better chance at randomizing the traffic across more links.
- 3) The routing strategy plays an important role in the performance of broadcast algorithms. For example, when broadcasting 1 MB on a 16K terminal network, non-minimal routing mitigates the high overhead of LLF for small locations by randomizing the traffic across links



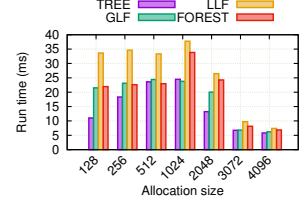
(a) Minimal routing, no background traffic



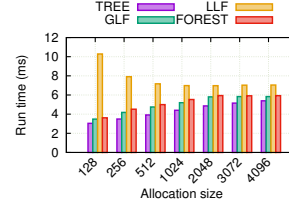
(b) Minimal routing, with background traffic



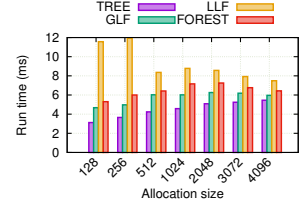
(c) Nonminimal routing, no background traffic



(d) Nonminimal routing, with background traffic



(e) Adaptive routing, no background traffic



(f) Adaptive routing, with background traffic

Fig. 8: Run time of a 1 MB broadcast on a 5K terminal Dragonfly network, with and without background traffic and with different routing mechanisms.

connected to the root router (although the run time of LLF remains much higher than that of other algorithms).

- 4) Interference with background traffic is not mitigated by the use of topology-aware algorithms. On the contrary, some topology-aware algorithms such as LLF present hot spots that could be more subject to interference.

We hope that our study will motivate further research in practical, topology-aware collective algorithms. In particular, a promising direction consists of considering more carefully the high radix offered by routers and taking into account their resources (such as buffers capacity). We think that higher-radix tree algorithms could be an important direction for further optimization.

2) *Future work:* Our work opens several directions. Other algorithms based on a scatter+allgather strategy are usually employed for large-sized broadcasts. These algorithms have not been evaluated in our study. They could benefit from topology-aware versions of scatter and allgather.

A second aspect that is worth investigating is energy consumption. While topology-aware algorithms do not necessarily perform better than non-topology-aware ones, they greatly minimize the network utilization, which in turn decreases the energy consumption of the supercomputer. Since DARPA set a 20 MW limit for exascale machines [36], such performance vs. energy tradeoff becomes worth considering.

With respect to interference between jobs, we have studied how the algorithms react to background traffic, but we have not studied how another job would react to one that executes a broadcast. If all jobs were to use topology-aware communication, this would greatly reduce the overall traffic and consequently decrease interference. Such a study could motivate a broad utilization of topology-aware algorithms to reduce link utilization at the level of the entire machine.

#### ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.

#### REFERENCES

- [1] J. Kim, W. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 77–88.
- [2] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li *et al.*, "The PERCS high-performance interconnect," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE, 2010, pp. 75–82.
- [3] 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.
- [4] NERSC, "Edison Cray XC30 supercomputer," <http://www.nersc.gov/users/computational-systems/edison/configuration/>.
- [5] T. Williams, "ALCF early science program," <https://www.alcf.anl.gov/files/HPCUserForumALCF-ESP.pdf>, April 2015.
- [6] A. Jukanovic, J. C. Sancho, G. Rodriguez, A. Lucero, C. Minkenberg, and J. Labarta, "Quiet neighborhoods: Key to protect job performance predictability," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 449–459.
- [7] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: Performance degradation due to nearby jobs," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 41.
- [8] A. Jukanovic, G. Rodriguez, J. C. Sancho, and J. Labarta, "Impact of inter-application contention in current and future HPC systems," in *18th Annual Symposium on High Performance Interconnects (HOTI)*. IEEE, 2010, pp. 15–24.
- [9] D. Skinner and W. Kramer, "Understanding the causes of performance variability in HPC workloads," in *Proceedings of the IEEE International Workload Characterization Symposium, 2005*. IEEE, 2005, pp. 137–149.
- [10] B. Prisacari, G. Rodriguez, M. Garcia, E. Vallejo, R. Beivide, and C. Minkenberg, "Performance implications of remote-only load balancing under adversarial traffic in dragonflies," in *Proceedings of the 8th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, ser. INA-OCMC '14. New York, NY, USA: ACM, 2014, pp. 5:1–5:4. [Online]. Available: <http://doi.acm.org/10.1145/2556857.2556860>
- [11] M. García, E. Vallejo, R. Beivide, C. Camarero, M. Valero, G. Rodríguez, and C. Minkenberg, "On-the-fly adaptive routing for dragonfly interconnection networks," *The Journal of Supercomputing*, vol. 71, no. 3, pp. 1116–1142, 2015.
- [12] D. Xiang and X. Liu, "Deadlock-free broadcast routing in dragonfly networks without virtual channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2015.
- [13] N. Jain, J. Lau, and L. Kale, *Collectives on Two-Tier Direct Networks*. Springer, 2012.
- [14] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Enabling parallel simulation of large-scale hpc network systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2016.
- [15] L. G. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, vol. 11, no. 2, pp. 350–361, 1982.
- [16] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott, "Overcoming far-end congestion in large-scale networks," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 415–427.
- [17] P. Fuentes, E. Vallejo, M. Garca, R. Beivide, G. Rodriguez, C. Minkenberg, and M. Valero, "Contention-based nonminimal adaptive routing in high-radix networks," in *International Parallel and Distributed Processing Symposium (IPDPS)*, May 2015, pp. 103–112.
- [18] A. Bhatele, W. D. Gropp, N. Jain, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2011, pp. 1–11.
- [19] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a dragonfly network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 336–347.
- [20] B. Prisacari, G. Rodriguez, A. Jukanovic, and C. Minkenberg, "Randomizing task placement and route selection do not randomize traffic (enough)," *Design Automation for Embedded Systems*, vol. 18, no. 3-4, pp. 171–182, 2014.
- [21] "MPICH," <http://www.mpich.org/>.
- [22] R. Thakur and W. D. Gropp, "Improving the performance of collective operations in MPICH," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2003, pp. 257–267.
- [23] J. M. Squyres and A. Lumsdaine, "The component architecture of OpenMPI: Enabling third-party collective algorithms," in *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, V. Getov and T. Kielmann, Eds. St. Malo, France: Springer, July 2004, pp. 167–185.
- [24] J. Pjesivac-Grbović, G. E. Fagg, T. Angskun, G. Bosilca, and J. J. Dongarra, "MPI collective algorithm selection and quadtree encoding," in *In Proceedings, 13th European PVM/MPI Users' Group Meeting*, Bonn, Germany, September 2006.
- [25] H. Zhou, V. Marjanovic, C. Niethammer, and J. Gracia, "A bandwidth-saving optimization for mpi broadcast collective operation," in *44th International Conference on Parallel Processing Workshops (ICPPW)*, Sept 2015, pp. 111–118.
- [26] P. Sack and W. Gropp, "Faster topology-aware collective algorithms through non-minimal communication," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPOPP '12. New York, NY, USA: ACM, 2012, pp. 45–54. [Online]. Available: <http://doi.acm.org/10.1145/2145816.2145823>
- [27] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng, "Optimization of MPI collective communication on BlueGene/L systems," in *Proceedings of the 19th annual international conference on Supercomputing*. ACM, 2005, pp. 253–262.
- [28] P. Sack and W. Gropp, "Collective algorithms for multiported torus networks," *ACM Transactions on Parallel Computing*, vol. 1, no. 2, p. 12, 2015.
- [29] Cisco Systems Inc., University of Wisconsin-La Crosse, and Inria, "Portable network locality (netloc)," <https://www.open-mpi.org/projects/netloc/>.
- [30] 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.
- [31] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "A case study in using massively parallel simulation for extreme-scale torus network codesign," in *Proceedings of the 2nd ACM SIGSIM/PADS conference on Principles of Advanced Discrete Simulation*. ACM, 2014, pp. 27–38.
- [32] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns, "Modeling a million-node dragonfly network using massively parallel discrete-event simulation," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*. IEEE, 2012, pp. 366–376.
- [33] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Using massively parallel simulation for MPI collective communication modeling in extreme-scale networks," in *Proceedings of the 2014 Winter Simulation Conference*. IEEE Press, 2014, pp. 3107–3118.
- [34] M. Besta and T. Hoefler, "Slim Fly: A cost effective low-diameter network topology," Nov. 2014, proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC14).
- [35] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc series network," *Cray Inc., White Paper WP-Aries01-1112*, 2012.
- [36] "DARPA summons researchers to reinvent computing," <http://www.extremetech.com/computing/116081-darpa-summons-researchers-to-reinvent-computing>.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. <http://energy.gov/downloads/doe-public-access-plan>.