Evaluation of Link Failure Resilience in Multirail Dragonfly-Class Networks through Simulation

Neil McGlohon
mcglon@rpi.edu
Rensselaer Polytechnic Institute
Troy, New York

Robert B. Ross
rross@mcs.anl.gov
Argonne National Laboratory
Lemont, Illinois

Christopher D. Carothers
chris.carothers@gmail.com
Rensselaer Polytechnic Institute
Troy, New York

Abstract

During long-term operation of a high-performance computing (HPC) system with thousands of components, many components will inevitably fail. The current trend in HPC interconnect router linkage is moving away from passive copper and toward active optical-based cables. Optical links offer greater bandwidth maximums in a smaller wire gauge, less signal loss, and lower latency over long distances and have no risk of electromagnetic interference from other nearby cables. The benefits of active optical links, however, come with a cost: an increased risk of component failure compared with that of passive copper cables.

One way to increase the resilience of a network is to add redundant links; if one of a multiplicity of links between any two routers fails, a single hop path will still exist between them. But adding redundant links comes at the cost of using more router ports for router-router linkage, reducing the maximum size of the network with a fixed router radix. Alternatively, a secondary plane of routers can be added to the interconnect, keeping the number of compute node endpoints the same but where each node has multiple rails of packet injection, at least one per router plane. This multirail-multiplanar type of network interconnect allows the overall size of the network to be unchanged but results in a large performance benefit, even with lower-specification hardware, while also increasing the resilience of the network to link failure.

We extend the CODES framework to enable multirail-multiplanar 1D-Dragonfly and Megafly networks and to allow for arbitrary link failure patterns with added dynamic failure-aware routing so that topology resilience can be measured. We use this extension to evaluate two similarly sized 1D-Dragonfly and Megafly networks with and without secondary router planes, and we compare their application communication performance with increasing levels of link failure.

CCS Concepts

• **Networks** → **Network simulations**; **Network topology types**; **Network reliability**; *Network performance evaluation*; • **Computing methodologies** → **Discrete-event simulation**.

SIGSIM-PADS '20, June 15–17, 2020, Miami, FL, USA

ACM ISBN 978-1-4503-7592-4/20/06…$15.00
https://doi.org/10.1145/3384441.3395989

Keywords

Link Failure, Interconnection Networks, High-Performance Computing, Parallel Discrete Event Simulation, Modeling

ACM Reference Format:
Neil McGlohon, Robert B. Ross, and Christopher D. Carothers. 2020. Evaluation of Link Failure Resilience in Multirail Dragonfly-Class Networks through Simulation. In *Proceedings of the SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS '20), June 15–17, 2020, Miami, FL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3384441.3395989

1 Introduction

Simulation of current and prototypical high-performance computing (HPC) interconnect networks is a useful tool for generating realistic performance expectations of proposed system acquisitions. Different interconnect designs, or network topologies, have varying strengths and weaknesses with different price-to-performance ratios. The cost of building a full-scale HPC interconnect is prohibitive for the purposes of evaluation. Using simulation, one can develop models for any arbitrary network definition and compare them with other, previously defined models.

This flexibility allows for fast, realistic predictions of how different topologies or network technologies may perform. For example, one can introduce quality of service traffic classes for bandwidth allocation or methods for the mitigation and prevention of congestion. With simulation, one can answer questions about how different network types behave at different scales, how different routing algorithms perform, or how job placement affects performance. It also allows for testing how the system might behave in states of irregularity, such as in the case of inter-router link failure.

Link failure is a common occurrence in HPC systems—so common in a large-scale system that between monthly maintenance routines, multiple faulty links may occur between routers in the network. Thus, system builders need to know that the interconnect they are purchasing is capable of still operating despite common link failure. This knowledge allows them to stick to routine maintenance of a month or more, so as to minimize downtime caused by the link failure, as opposed to immediately bringing parts of the system down for emergency system repair.

Routing algorithms for HPC interconnects conventionally take certain characteristics of networks as a given, and virtual channel (VC) assignment can be made appropriately to avoid deadlock. Introducing even a single link failure into the network can potentially reduce to zero the number of legal, deadlock-free paths for certain routing algorithms. Also, routing algorithms that could correctly explore all possible legal routes in typical topologies may fail to do so in the presence of link failure. Irregular systems—those with abnormal, arbitrary, or not uniformly defined structure—require

additional attention when routing in order to avoid cyclic channel dependency problems such as head-of-line blocking or deadlock [9, 10].

Because of the need to avoid channel dependency problems, one cannot assume that router validity is assured just because the network is connected. A legal path between two endpoints in an interconnect is any path that does not cause cyclic channel dependencies. Existing routing algorithms, such as those in 1D Dragonfly and Megafly, must be altered to ensure that only legal paths are taken in the presence of link failures. If a valid legal path does not exist, then communication (and the application making it) should fail. Determining whether a legal path exists is a computationally complex problem because the enumeration of all possible paths grows significantly as the number of endpoints (and, by association, routers) increases.

We extend the CODES [7] interconnect network simulator to allow for links in simulated networks to be marked as failed and for packets to be routed dynamically around failed links along legal paths, if and only if they exist. If all legal paths for a packet between two endpoints have at least one failed link, then communication fails, and the simulation ends.

The network types evaluated in this work are the 1D-Dragonfly and Megafly networks. 1D Dragonfly, like other Dragonfly-class networks, has groups of routers in which each group is connected to every other group with at least one global link. Within these router groups, the 1D Dragonfly uses an all-to-all configuration for local router links.

Megafly, also known as Dragonfly+, has the same global link structure as 1D Dragonfly has; but instead of all-to-all links within local router groups, it uses a 2-level fat tree configuration to form a fully connected bipartite graph. Router responsibilities are also divided differently in Megafly. Routers with global links, known as spine routers, have no compute node terminal links. The other routers, those with compute node terminal links but no global links, are known as leaf routers. This type of link configuration is designed to increase the scalability of a network with a lower cost [30].

In this work we study the effects of link failure in Dragonfly-class topologies and posit a strategy for increased resilience—multiplanar networks.

The main contributions of this work are as follows:

- Introduction of a *network manager* to the CODES framework to orchestrate link failure, and identification of legal paths given virtual channel constraints
- Extension of the CODES Dragonfly and Megafly routing algorithms— minimal (MIN), Valiant non-minimal (VAL), and progressive adaptive (PAR)—to route dynamically in the presence of network link faults
- Extenion of the CODES Dragonfly and Megafly network models to support multiplane-multirail configurations with multiple schemes for injection rail selection
- Evaluation of multirail-multiplanar inter-job interference performance and the effect of multiplanar configurations on the resilience of two example Dragonfly and Megafly networks with varying levels of link failure

2 Background

In this work, we study the effects of link failure in HPC network interconnects made up of routers or switches connected in a specific pattern known as a network topology. Also connected to these routers are compute node terminals that inject packets into the network to be routed along paths to other compute nodes. In a real-world interconnect, these packets are the carriers of information between processes in an HPC application. We can use captured real-world communication patterns of HPC applications for simulated packet generation or create our own synthetic jobs to act as benchmark workloads in simulated networks. By simulating multiple simultaneous workloads, we can analyze how varying intensities of communication from different workloads interfere with each other. Higher levels of interference reflect poorer network performance.

Simulation allows for individual packets to be injected into a modeled network of arbitrary size. Different routing algorithms can be applied, and various existing or proposed router and link technologies can be evaluated quickly and easily. But there is no free lunch: with any added complexity to what is being simulated, there is almost always a proportional increase in cost. This frequently comes in the form of increased runtime.

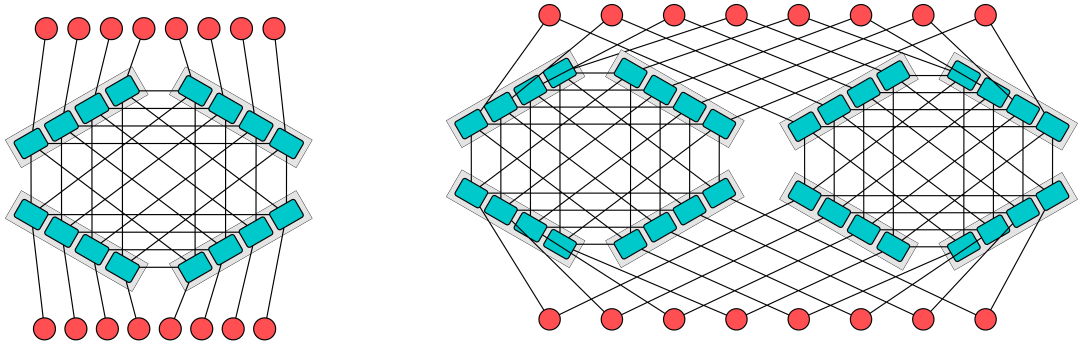
Parallelization of a simulation allows it to be spread across all available processors in a lab machine or, if implemented properly, across thousands of processors in an HPC cluster. Network packet routing is a problem well suited for the use of parallel discrete event simulation (PDES) since every interaction between entities at any specific time can easily be represented as an event in the simulation.

The CODES interconnect network simulator utilizes ROSS [4] as the underlying PDES engine and can thus be easily parallelized at large scale to yield impressive strong-scaling performance benefits for network modeling [32]. The granular elements of a ROSS simulation are logical processes (LPs). Each LP represents a single entity in the simulation. In the context of a CODES simulation, each network switch, network terminal/node, or traffic generation server is represented by a single LP. Each LP has its own state, and any interaction between LPs is modeled by an event at a specific time in the PDES simulator. LPs are grouped into abstractions called kernel processes (KPs) that serve as an organizational structure for synchronization of LPs during simulation. LPs and, by association, KPs are mapped uniquely to processing elements (PEs). Each physical MPI rank participating in the execution of the simulation has one PE.

A main tenant of PDES is its parallelism. ROSS provides a framework not only for sequential, single-PE execution but also for two modes for parallel execution. In this work we specifically use *optimistic* parallel execution.

With optimistic parallel execution, LPs can process events at their pace, allowing for drift in observed simulation time between them. This approach, however, can result in out-of-order events that threaten the validity of the simulation. ROSS resolves the issue through the use of reverse computation [5] and the *Time Warp* protocol [18] to return the simulation to a known valid state.

A general CODES simulation can be characterized by a network of *router/switch* LPs connected together via *links*. Routers generally each have *terminal node* LPs linked to them as well. Workload LPs, each representing a simulated MPI rank, are mapped to the terminal



(a) Single-Rail-Single-Plane 1D Dragonfly

(b) Dual-Rail-Dual-Plane 1D Dragonfly

Figure 1: Example layout of single-rail-single-plane (Figure 1a) and dual-rail-dual-plane (Figure 1b) 1D Dragonfly networks with 4 routers (blue squares) per group and 4 groups per plane. There are 16 compute node hosts (red circles) attached to each plane in a given network. In the single-rail-single-plane case, each node has one rail for injection to a router (blue squares) but in the dual-rail-dual-plane case, each node has two rails for injection, one per plane. The high-level connection scheme of Megafly follows similarly with the exception that routers within local groups are connected in the shape of a fully connected bipartite graph instead of all-to-all like 1D Dragonfly.

nodes and generate traffic for injection into the network. These workload LPs are the origin and final destination of messages that are transmitted through the network via packets.

CODES is designed so that specific HPC interconnection network topologies and technologies can be tested at realistic scales quickly and easily with a variety of workloads and a large degree of configuration flexibility. This simulator has previously been shown to scale well and is capable of simulating extreme-scale HPC systems [7, 23–25].

3 Network Model

The network topologies examined in this work are the 1D-Dragonfly and Megafly networks. Dragonfly-class networks have one particular characteristic in common: routers in the network are grouped together into local groups with some degree of local link connectivity between them. Global link connections, those that span between two local groups, are configured such that at least one global link exists between any two pairs of local groups. In other words, local groups themselves are connected all-to-all.

How these topologies differ lies in how local groups are configured. In 1D-Dragonfly, routers within a local group are connected in an all-to-all fashion. This makes the assured minimum number of hops between any two routers in a local group equal to one. When router groups become large, however, this results in an greater cost because of the increased number of local links. But because the groups are also mandated to have at least one level of all-to-all connectivity between them, increasing the number of groups with the aim of reducing the number of routers per group will result in an increase in the cost associated with global links.

In contrast, the Megafly network topology has routers within local groups connected in a two-level fat-tree configuration. As a result, these local router links form a fully connected bipartite graph of two types of routers: *leaf* routers, which have local router and terminal/compute node links, and *spine* routers, which have local router and global router links. No terminals are attached to spine routers, and no leaf routers own any global links.

3.1 Multirail-Multiplanar Networks

This work focuses on networks with multiple, independent planes of routers interconnecting a fixed set of compute nodes. Because compute node hosts are shared between router planes, each node has multiple rails for packet injection, at least one per plane. For simplicity, the term multirail is often used because multirail-multiplanar networks are a special case of the broader term. In this work we will use multirail and multiplanar terms interchangeably, and the number of injection rails per compute node is equal to the number of router planes in a given network. A small scale example showing the layout of a single-rail-single-plane 1D Dragonfly in comparison to a dual-rail-dual-plane configuration of the same network can be seen in Figure 1.

3.2 Benefits of Multiplanar Networks

Adding additional planes of routers has shown great ability to reduce the effect of inter-job interference even with reduced bandwidth [22]. We posit in this work that the benefits of multiplanar networks lie in the ability not only to reduce the effect of interference but also to reduce the effect of congestion caused by link failures. Additional planes of independent routers multiplies the available paths between any two endpoints in the network by a factor proportional to the number of additional planes. Thus, the odds of completely severing all legal paths between any two endpoints are reduced.

4 Link Failures

We have extended the CODES interconnection simulation framework to include an organizational class called a *network manager*. This class stores all link connection information for a network, including the failure state of the links. This data is loaded in at the start

of the simulation and is identical across all PEs in the simulation—the state of links with regard to failure is static throughout the simulation. Because the time periods simulated by CODES are generally less than a few seconds, we felt that enabling in situ link failure was not crucial. Additionally, as observed in this work, the effect of a single link failure (or any number that would reasonably be expected to occur in the course of seconds) is generally insignificant to application performance.

In a realistic mid-simulation link-failure scenario, a management communication layer may be implemented to facilitate the transmission of global knowledge. Adapting the CODES network manager to allow for this would simply require additional ROSS events to facilitate the transmission of knowledge to all LPs. Allowing for in situ link failures, however, brings in new layers of uncertainty. For example, 'what happens to a packet that was already injected prior to link failure and its only legal path became severed? How this error is handled depends on the implemented communication standard and is outside the scope of this study. Here we assume that information regarding link failure is a priori knowledge known by all terminals and routers in the network.

The network manager provides an interface for routers to query for information about the network. For example, a router has a connection manager subclass of the network manager that has information about connections specific to it and its group and can answer questions such as the following: What routers am I connected to locally, and what routers am I connected to that have a connection to a specific group? An additional parameter can be added to these queries to take into consideration whether a link has failed.

The new link failure module allows any link in the network to be failed including those between terminals and routers, local links between two routers in the same group, and global links between routers in different groups. In this work, however, we focus solely on link failure between local and global router to router links.

The addition of link failures to a network model poses challenges. In CODES, most routing algorithms are implemented to take advantage of given knowledge about the network. For 1D Dragonfly, one of these bits of design knowledge is the expectation of having at least one global link between any two groups.

Imagine a 1D-Dragonfly network configured so that each group is connected to every other group with exactly one global link. Now picture a packet being generated in a group A and destined for a router in group B. In a minimal path, it will need to traverse exactly one global link; if that global link has failed, then the only minimal path between any routers in group A and group B no longer exists. Therefore, it would need to be routed nonminimally through a router in an intermediate group. However, we also have to be aware of any failures in a potential intermediate group. The only global link between the intermediate group and group B may also have failed, and hence that group would be ineligible for a nonminimal route between groups A and B.

Unfortunately there are limitations to how packets can be routed in these interconnects so as to avoid the performance-limiting issue of head-of-line blocking or the performance-halting issue of deadlock [9]. Virtual channels, a method of virtually partitioning router ports into subunits, are commonly used in Dragonfly networks to avoid these issues by requiring that specific rules be followed about what VCs to assign a packet to when routing. To avoid deadlock, it is crucial to avoid what is referred to as a cyclic channel dependency [10, 21].

In order to avoid these channel dependency issues, rules are imposed on the routing algorithms that set the maximum number of hops allowed to be traversed within each group and how many global hops may be utilized in routing a packet from group A to group B. These constraints mean that though a router is technically connected to the network, it still may not be legally accessible by any or all other routers in the network.

If a packet is injected into a router plane and the routing algorithm fails to find a legal path or if a legal path does not exist, then the packet will reach a dead end, and the workload will fail. Optimally, then, we would like to inject a packet only onto a router plane that has a valid path to its destination. To do so, we need to be able to answer two questions.

(1) Does a legal path exist between source router x and destination router y with at most l_s local hops in group of source router, at most l_i local hops in any intermediate group, at most l_d local hops in the destination group, and exactly g global hops?
(2) Given that a legal path matching these criteria exists, what are the next stops from router x that would match any path matching the criteria?

Using constraints of the Dragonfly and Megafly networks, notably that l_s, L_i, L_d equal 1 and 2 for Dragonfly and Megafly, respectively, we can simply use l local hops per group in the problem formulation instead of three separate variables. If we had a black-box solution to the first question, we could know whether there exists a minimal path within a local group ($g = 0$), a minimal or near-minimal path with no intermediate groups ($g = 1$), or a legal nonminimal route ($g = 2$) with no need to adjust any VC assignment strategies in the network model.

4.1 Algorithmic Solution

We have developed an algorithm for solving these problems in Dragonfly-class networks leveraging the known constraints to reduce the algorithmic and computational complexity. This problem can be solved via a dynamic programming-like approach with memoization/caching to quickly return an answer to a query.

Algorithm 1 solves the problem of finding the existence (and the next hops) of a legal path defined by l allowed local hops in the current group and L maximum local hops in any group visited and with exactly g global hops. It is a modified depth-first-search algorithm to traverse a graph to find if the destination router y is reachable by x with the given restraints.

Requiring exactly g global hops allows us to narrow down what types of paths will be included in our query. If we allowed it to return paths with at most g hops instead, then we would not be able to prevent nonminimal intermediate group (2 global hops) paths from being included in the query response. For the progressive adaptive algorithm, we need to be able to compare minimal and nonminimal paths in order to decide how to route to the final destination.

The algorithm uses recursion, knowing that if there is a valid local hop from the current router with the given restraints, then

there must first be a valid hop from a locally connected router with one less local hop in the current group. Similarly, if there is a valid global hop from the current router with the given restraints, then there must first be a valid hop from a globally connected router with exactly one less global hop in the path—but the number of local hops allowed in the next current group is reset in the subproblem because a global hop was taken.

Because this algorithm only explores potential next stops if the link to the next stop is not failed, this algorithm will not return any paths that would require a failed link be traversed. If there are no valid next stops with the given constraints returned from the algorithm, then no legal path exists with those constraints.

4.2 Algorithmic Caveats

Traditional depth-first search requires a set of visited graph nodes to prevent cycles. Instead, we rely on the constraints to ensure legal paths that do not needlessly route.

With 1D Dragonfly, local group hops are limited to 1, which makes cycles or roundabout paths within a local group impossible. In Megafly, local group hops are limited to 2; but to prevent issues of VC dependency deadlock, all nonminimal paths through an intermediate group must travel through a leaf router, requiring explicitly 2 local group hops. This means that a 2-hop detour revisiting the same spine router A, as in the example, is mandated by the network model.

One might wonder, then, why not just utilize known, efficient shortest-path algorithms? Algorithms such as Flloyd-Warshall All Pairs Shortest Path[8] can find the shortest path between any two nodes in a graph quickly and optimally. But we have yet to find a modification of this algorithm that can work with the necessary constraints in order to ensure that the shortest paths found by the algorithm are also the shortest *legal* paths.

One can enumerate all shortest paths and then filter out only legal paths with this method, but doing so requires a large space and time complexity for larger networks. If we wanted to find paths of at least L hops, as is the case for nonminimal routes used by adaptive routing, we would need additional complexity.

Algorithm 1, however, is not designed to find only the shortest paths. This algorithm could be modified to enable that by encoding a current hop count as an input and to return the hop count when the destination is found. Such a modification would come at great cost to computational efficiency, however, since it would render memoization impossible: the first found legal path (and thus the memoized response) is not guaranteed to be the shortest. Another way to ensure only shortest paths would be to employ a breadth-first search approach but, again, at the sacrifice of memoization and its computational benefits.

4.3 Routing and Planar Selection

To route correctly through a network with link failures, we leverage Algorithm 1 to find legal paths with specific constraints in the path. Most importantly, we specify the exact number of global hops g in a desired path. This allows us to know whether there exist local-group-only paths, single-global-hop paths, or two-global-hop paths.

Enabling CODES to find legal paths given specific constraints allows it to dynamically determine whether a legal path exists and to route accordingly. If enough links fail, then there may not be any legal paths between a source and destination router; and, if any compute nodes attached to the routers wish to communicate with each other, the communication will fail.

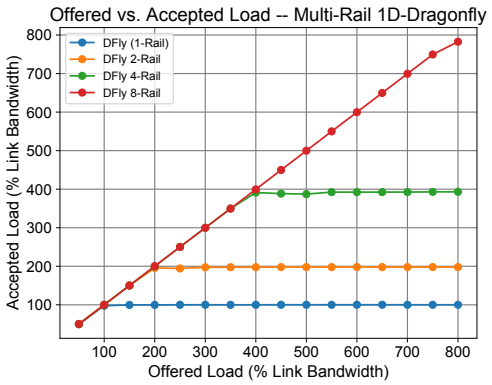
Algorithm 1 Finding Existence and Next Stops in Legal Path

```
Memoization Map of Input to Response: M(input)
Maximum Local Hops Allowed per Group: L

Input i:
    Source Router GID: x
    Destination Router GID: y
    Local Link Adjacency List for Source: A_l
    Global Link Adjacency List for Source: A_g
    Max Local Hops in Current Group : l
    Exact Global hops in path: g

GetValid(i):
V_next ← Empty set of valid next hops
if x == y and g == 0 then
    V_next ← y
    return V_next
if i ∈ M then
    return M(i)
if l > 0 then
    for link ∈ A_l do
        if link is not failed then
            V ← GetValid(link.dest, y, l − 1, g)
            if size(V) > 0 then
                V_next ← link
if g > 0 then
    for link ∈ A_g do
        if link is not failed then
            V ← GetValid(link.dest, y, L, g − 1)
            if size(V) > 0 then
                V_next ← link
M(i) ← V_next
return V_next
```

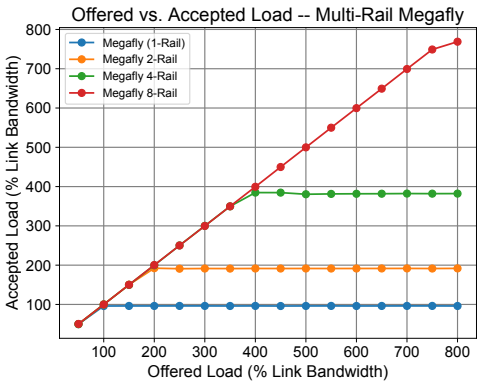
This work allows for the 1D-Dragonfly and Megafly models to feature multiple router planes servicing the same set of compute node terminals. Nodes have a choice of what rail to inject a given packet onto. To ensure greater resilience to link failure, we want to inject a packet onto a plane only if we know that there is a legal route between the source and destination compute nodes on that plane. Algorithm 1 gives us the ability to know this prior to injection.

Given a set of V valid planes for injecting a given packet, we have a variety of options for deciding the preferred plane of injection. Following similar work of enabling multirail-multiplanar Slim Fly networks in CODES [22], we have implemented the following schemes for injection plane selection, with a few modifications to account for link failures.

***PATH* Planar Selection:** The Flloyd-Warhsall shortest path evaluation to estimate the distance between two routers can be utilized to minimize the distance traveled by an injected packet. It



(a) Single-Rail-Single-Plane 1D Dragonfly



(b) Dual-Rail-Dual-Plane 1D Dragonfly

Figure 2: Single-, dual-, quad-, and octo-rail versions of 1D Dragonfly and Megafly were offered varying loads of synthetic uniform random data by sweeping the mean amount of time between when packets of fixed size were injected into the networks. As is the case for any multirail-multiplanar networks in this work, the number of router planes in each network strictly equals the number of rails. Offered and accepted loads are shown here as a percentage of the link bandwidth (12.5 GiB/s).

may not mean that a legal path with that distance exists, but it can generally determine whether significant link failures would require a great deal of routing around.

***CONGESTION* Planar Selection:** This scheme estimates the congestion being experienced by the number of packets set for injection along each rail at the originating terminal. This estimation is made with information from a credit-based flow control system that allows terminals to inject on rails with less inferred back pressure. This is the scheme utilized in the evaluation of multiplanar networks in this paper.

***RANDOM* Planar Selection:** This scheme involves trivially selecting a valid plane at random. This naively balances load across available planes but does not take into account any congestion that may be present in the network at different levels on different planes.

***DEDICATED* Planar Selection:** This scheme allows for the workload to select the injection rail during the creation of the message. The network makes no alteration. If a valid path on the given plane does not exist, then the workload will fail.

5 Validation

Without access to a physical multiplanar network, truly validating findings from simulation is difficult. We instead rely on other factors to ensure that adding additional rails and planes behaves in the way that we expect.

All routing adjustments for enabling link failure were compared with previous accepted routing implementations and return results in line with expectations.

We ran a bandwidth-testing sanity check of our implementation of multirail-multiplanar 1D Dragonfly and Megafly to ensure that the accepted load of the network scales with the number of planes in the network. If a single plane accepts some load λ, we expect 2 planes to accept $2 \times \lambda$. Specifically, we constructed 4 networks of each type, 1D Dragonfly and Megafly, with varying numbers of router planes: 1, 2, 4, and 8. We scaled synthetic uniform random traffic injected into the network at a percentage of the link bandwidth used in the network, ranging from 50% to 800%. Figures 2a and 2b display the results of the tests. We measured accepted load by counting the number of bytes received by terminals during a given time window and dividing by the length of the time window. Results are in line with expectations: greater numbers of router planes allow for greater amounts of accepted load.

6 Experiments

CODES works with DUMPI application traces through the SST DUMPI Trace module [31]. These traces are captured from communication patterns of real HPC applications and grant far greater understanding of how real-world applications might perform on given networks than what is typically achievable through basic synthetic workloads.

We used publicly available DUMPI traces provided by the Design Forward Program [11] of NERSC. All experiments in this work were performed on our 40-core Intel Xeon CPU E5-2640 laboratory server. Most simulations completed on the order of minutes run in parallel using 10 processor cores each.

6.1 Workloads

Algebraic MultiGrid Solver (AMG) Workload: AMG is an algebraic multigrid solver miniapp for unstructured mesh physics packages. Our experiments used the version with 1,728 MPI ranks. We mapped each rank to its own terminal node in the given network. The application from which this trace was created had a runtime of 2.2 seconds, including computation time, with 1,728 MPI ranks across 72 hosts. About half of the time (52%) was time spent on communication [11].

MultiGrid (MG) Workload: MultiGrid is a geometric multigrid cycle miniapp from the adaptive mesh refinement application framework BoxLib [14]. Our experiments used the version with 1,000 MPI ranks, each of which, just as we did for our AMG workload, was mapped to its own terminal node in the simulated network. The application from which this trace was created had a runtime

Table 1: Configurations of 1D-Dragonfly and Megafly networks utilized in this work for performance and resilience comparisons. Note that in some experiments dual-rail configurations were written to use reduced 7.0 GiB/s bandwidth as per InfiniBand FDR specifications.

	1D Dagonfly	1D Dragonfly - Dual Plane	Megafly	Megafly - Dual Plane
Router Radix	36	36	36	36
Planes	1	2	1	2
Rails	1	2	1	2
Groups	19	38	10	20
Node Count	3078	3078	3240	3240
Router Count	342	684	360	720
Nodes per Router	9	9	18 (Leaves only)	18 (Leaves only)
Global Connections	3078	6156	3240	6480
Global Connections / Group	162	162	324	324
Global Connections Between Groups	9	9	36	36
Link Bandwidth	12.5 GiB/s	12.5 GiB/s (EDR) 7.0 GiB/s (FDR)	12.5 GiB/s	12.5 GiB/s (EDR) 7.0 GiB/s (FDR)
Routing Algorithm	Adaptive (PAR) [19]	Adaptive (PAR)	Adaptive (PAR)	Adaptive (PAR)
Planar Selection Scheme	n/a	`CONGESTION`	n/a	`CONGESTION`

of 1.4 seconds, including computation time, with 1,000 MPI ranks across 42 hosts. The application spent very little (3.7%) of the time on communication [11].

Synthetic Workload: To simulate other random jobs in the network, we injected random traffic using a uniform random synthetic traffic generator. We fixed the mean interval time at the CODES default of 100 μs but increased the overall payload of injected messages to vary the intensity of interference.

6.2 Evaluation Metrics

Higher levels of link failure with high levels of injected traffic can result in congestion. We seek to determine how congestion-caused inter-job interference is affected by increasing levels of link failure.

To evaluate the resilience of these networks, we used various application performance metrics including the average number of hops traversed by packets, the mean latency of all packets in the simulation, and the maximum communication time (the maximum amount of time spent by any one primary workload process participating in the workload). These metrics can be measured on networks with a range of link failures and can be directly compared with networks without link failures.

6.3 Studies Performed

This work combines a standard interference study of single- and dual-plane 1D-Dragonfly and Megafly networks with a study of interference on the same networks with varying degrees of link failure.

In the link failure study, we also looked at how the same level of link failure affects a reduced-bandwidth version of the dual-planar network configurations. The aim is to show how the additional costs of building multiple router planes can be mitigated by using cheaper hardware while maintaining performance benefits similar to those of the standard multiplane configurations as noted in [22].

In our experiments we fail router-router links uniformly at random across the entire tested network until the desired percentage of failed links was met. Information about failed links was generated by using a script and was loaded by the CODES simulation. It is important to note that for our experiments, the number of failed links are specified as a percentage of the number of links in a single-plane network.

7 Discussion

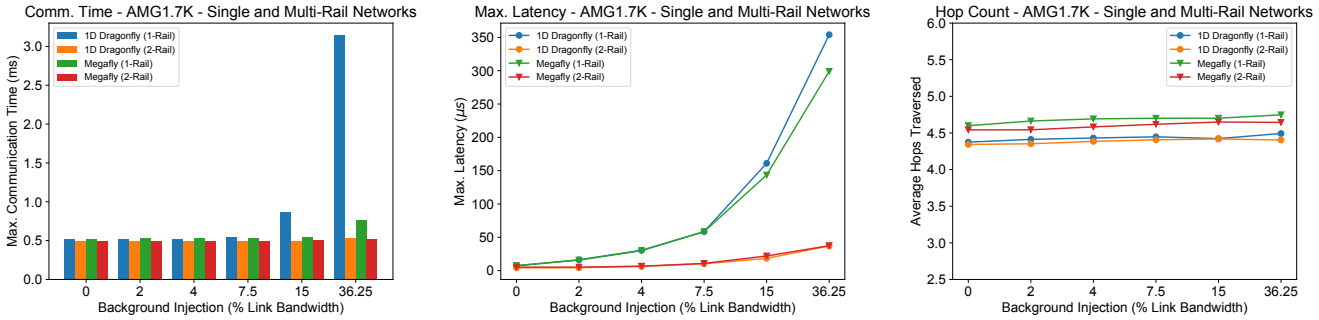
To discuss the findings of this work we first, in Section 7.1, present an inter-job interference study as a baseline comparison of single- and dual-plane 1D-Dragonfly and Megafly networks with no link failures. We then, in Section 7.2, present a network resilience study by taking the the highest level of interference observed from Section 7.1 and apply it to single- and dual-plane networks with increasing levels of link failure.

Table 1 shows the network configurations used. Our tested single-plane networks feature links with bandwidths comparable to InfiniBand EDR (12.5 GiB/s). Our tested dual-plane networks consist of two versions each, one with the same InfiniBand EDR specification links and the other using only InfiniBand FDR equivalent links (7 GiB/s).

7.1 Single-Rail to Multirail Comparison

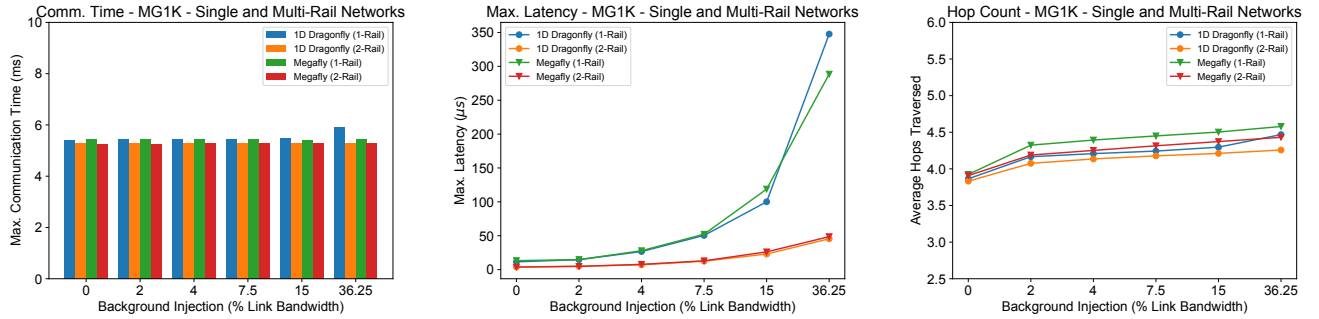
We begin by comparing single- and dual-plane 1D-Dragonfly and Megafly networks with zero-failed links. Our aim is to show a baseline standard of benefit that can be expected from a multiplanar network variation.

Because of the different balancing recommendations for generating these networks, creating a truly fair comparison of the different topologies is difficult. To create a Megafly network with approximately 3,000 compute node terminals with a 36 router radix (to utilize the same hardware specifications as the tested 1D Dragonfly), we must have 36 global connections between any two groups in the network. 1D Dragonfly, on the other hand, has only 9, and thus path diversity for inter-group messages is reduced. Nevertheless, we attempted to generate two networks of similar capabilities.



(a) Application Communication Time **(b) Latency** **(c) Hop Count**

Figure 3: Synthetic interference experiments on the AMG1728 trace workload with 1,000 synthetic background ranks. Each synthetic rank injects packets into the network at a percentage of the total link bandwidth of 12.5 GiB/s (≈InfiniBand EDR). The same workloads and job allocation maps were used across each network.



(a) Application Communication Time **(b) Latency** **(c) Hop Count**

Figure 4: Synthetic interference experiments on the MultiGrid1000 trace workload with 1,000 synthetic background ranks. Each synthetic rank injects packets into the network at a percentage of the total link bandwidth of 12.5 GiB/s (≈InfiniBand EDR). The same workloads and job allocation maps were used across each network.

Because job locality plays an important role in mitigating the effects of high network usage, we tried to choose a job-to-compute-node mapping that is "equally poor" for all networks: namely, random mapping. The same job-to-compute-node mappings were used between all evaluated networks. Unfortunately, because of the greater amount of intergroup linkage in the Megafly network, we still expect some performance bias in favor of Megafly with this mapping.

Figures 3 and 4 show the results of experiments performed with the AMG1728 and MG1000 workloads with synthetic background interference traffic on the single- and dual plane versions of 1D Dragonfly and Megafly without link failures. Background interference is scaled in these experiments from 0% (primary workload only) to 36.25%. The experiments utilize the same failure-aware routing schemes developed by using Algorithm 1.

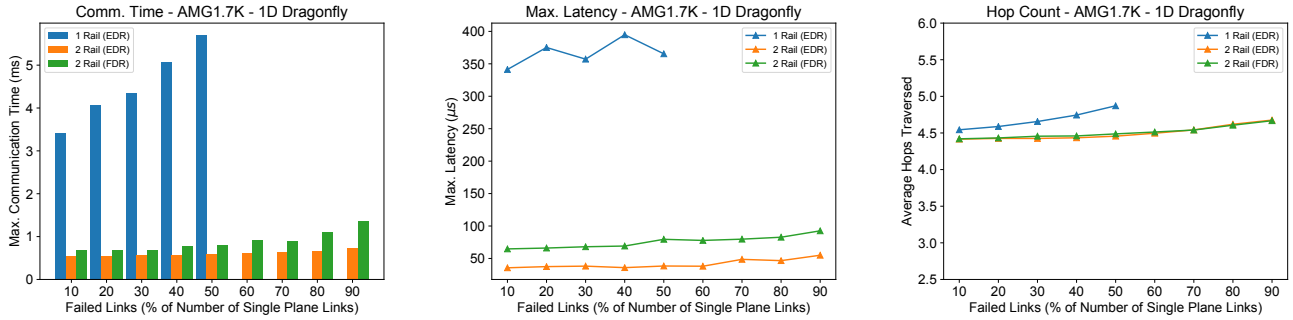
Figures 3a and 4a show that dual-rail networks have greater capacity than do single-rail networks to operate as if uncongested. The network has more places for packets to exist; and so, for a given set of packets, the odds of two packets finding themselves in the same router buffer is reduced. Thus, there is less contention for bandwidth, inter-job interference is minimized, and the maximum amount of communication time spent by any one node is decreased at higher levels of background interference.

Following a similar trend, Figures 3b and 4b show that the maximum packet latency is also significantly reduced in the multiplanar variants. The average hop count traversed by packets in the network, shown in Figures 3c and 4c, is not greatly affected but does tend to be slightly reduced in multiplanar versions.

7.2 Link Failure Study

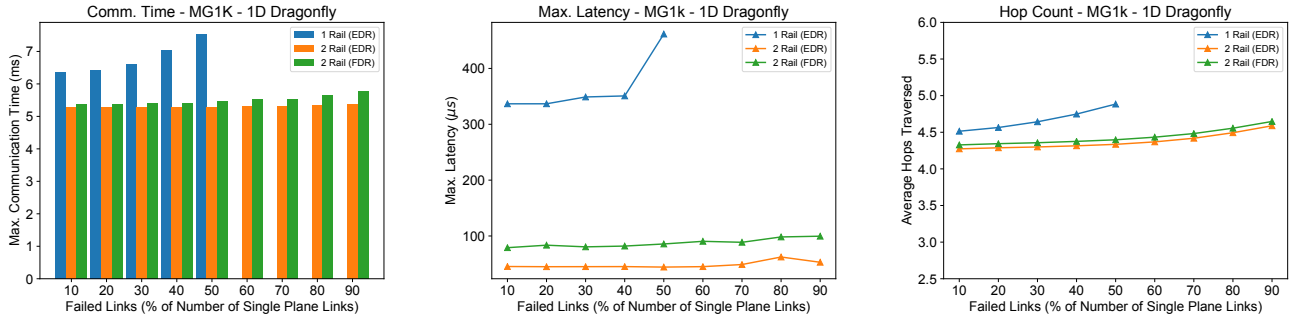
Experiments for this study use the same DUMPI trace workloads, AMG1728 and MG1000, as in Section 7.1 but with a fixed level of background interference: 36.25%. Instead of scaling the interference, we scaled the number of failed links as a percentage of the number of links in a single plane. To evaluate how resilient different versions of these networks are, we applied consistent link failure counts to each network type.

We included the FDR variants for consideration because providing an additional plane of routers on top of the first gives a distinct advantage to the multiplanar network in nearly any type of comparison except for cost. The increased cost of multiplanar networks can be mitigated by using lower-specification hardware.



(a) Application Communication Time **(b) Latency** **(c) Hop Count**

Figure 5: Link failure resilience study on the 1D-Dragonfly topology with the AMG1728 trace workload with 1,000 synthetic background ranks. Each synthetic rank injects packets into the network at 36.25% of 12.5 GiB/s. Note that the total number of links failed is based on a percentage of the number of links found in a single plane.



(a) Application Communication Time **(b) Latency** **(c) Hop Count**

Figure 6: Link failure resilience study on the 1D-Dragonfly topology with the MG1000 trace workload with 1,000 synthetic background ranks. Each synthetic rank injects packets into the network at 36.25% of 12.5 GiB/s. Note that total number of links failed is based on a percentage of the number of links found in a single plane.

Figures 5, 6, 7, and 8 show the results of these experiments. In Figures 5a, 6a, 7a, and 8a we see that as we increase the number of failed links in the network, performance suffers. Failed links reduce the number of available paths for packets to take, and thus packets are far more likely to compete for bandwidth along inter-router links.

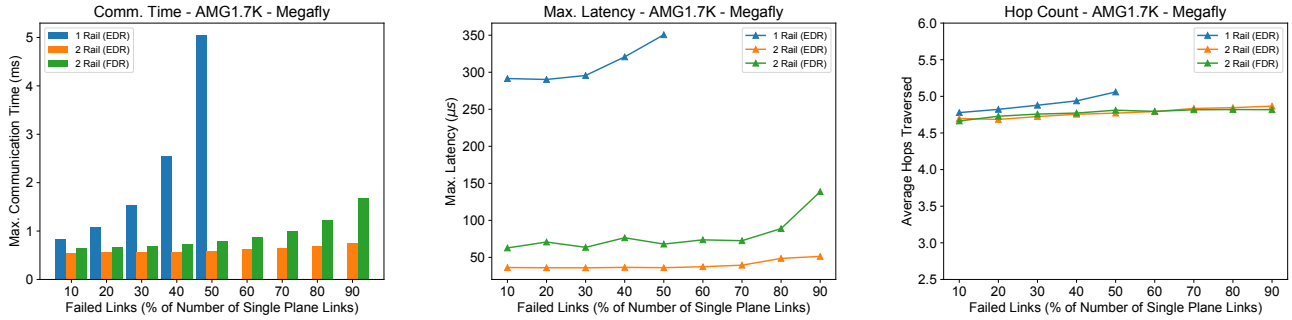
As we reached 50% failed links, however, the single-plane network configurations started to fail since there were no legal paths between select endpoints in the workload. The dual-plane configurations were able to withstand significantly more numbers of failed links before starting to show signs of congestion or interference, let alone failing communication altogether.

AMG is an application that is much more communication intensive than MG is, with a lot more collective operations that prevent the application from moving forward and injecting new packets into the network until the previous packets complete. Because AMG is collective heavy, its maximum communication time (the maximum amount of time any one process spends doing communication) is more sensitive to interference. But because new packets are not introduced until the previous collectives have completed, the maximum latency (maximum time that a single packet spends in the network) is not as affected. MG, on the other hand, mostly involves MPI_SEND and MPI_IRECV calls, with some MPI_WAITSOME's as well, and so it has fewer operations to prevent it from requesting to inject packets into an already congested network.

Effects of the failed links in dual-plane variations were felt mostly by the lower-specification simulated hardware, while the standard dual-plane versions seem mostly unperturbed by the changes to the network.

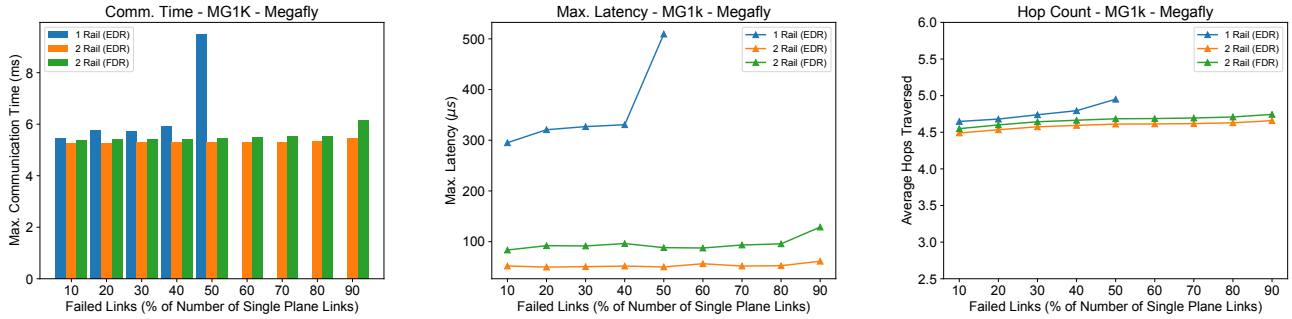
The maximum packet latencies in the experiments are also interesting. Figures 5b, 6b, 7b, and 8b show that the dual-plane versions have a distinct advantage with regard to latency. The effects of reduced bandwidth are noticeable but still significantly improved over the standard specification single-plane version.

The experiments also showed the effects of congestion on hop counts. In CODES, the hop count is the number of routers traversed in a path by a packet. We can see in Figures 5c, 6c, 7c, and 8c that the networks start to experience some level of congestion as we increase the number of failed links. This is evident by the higher number of hops traversed by packets. Because of the progressive adaptive routing scheme employed by 1D Dragonfly and Megafly, packets can be routed along longer, nonminimal routes when congestion is detected locally.



(a) Application Communication Time **(b) Latency** **(c) Hop Count**

Figure 7: Link failure resilience study on the Megafly topology with the AMG1728 trace workload with 1,000 synthetic background ranks. Each synthetic rank injects packets into the network at 36.25% of 12.5 GiB/s. Note that total number of links failed is based on a percentage of the number of links found in a single plane.



(a) Application Communication Time **(b) Latency** **(c) Hop Count**

Figure 8: Link failure resilience study on the Megafly topology with the MG1000 trace workload with 1,000 synthetic background ranks. Each synthetic rank injects packets into the network at 36.25% of 12.5 GiB/s. Note that total number of links failed is based on a percentage of the number of links found in a single plane.

We chose to fail a fixed number of links for comparing across single- and dual-plane networks in order to keep as many variables as possible constant in comparisons. Nevertheless, even if one compares, for example, the 80% dual plane data with the 40% single-plane data, the dual-plane network still wins out.

8 Related Work

Similar work observing the effects of link failure has been performed on single-planar 1D-Dragonfly networks [6]. Authors of that work tested a 8,192 node network with 1% link failure rate with three workloads using the Structural Simulation Toolkit network simulator [28]. They encountered similar issues in the realm of efficient routing computations; they precomputed a small subset of paths from a network with no link failures and filtered out paths that had failed links. A limitation of their work is that a legal path may have existed between two nodes in the network but was not included in the subset: thus, if all paths in the subset were rendered invalid by link failure, communication would fail. In contrast, our work dynamically determines legal paths to route along if they exist.

In recent years, studies have sought topologies that are particularly resilient to router link failures. Modified versions of the fat-tree network topology have been tested with varying levels of link failure and showed improvement over the default fat-tree network [1, 2].

Routing correctly in an HPC network is crucial to avoiding deadlock and performance-limiting situations, and this is even more important in irregular networks such as those with link failures. Many studies [1, 2, 15, 16, 29, 34] have been performed on routing deadlock-free within a network with link failures.

Another way to route correctly within a network with failed links is to completely ignore the type of network it originally was, treating the new, irregular network as an arbitrary graph. This type of routing is known as topology-agnostic routing. A survey of routing algorithms for general networks discusses the challenges of routing in irregular interconnection networks without deadlock [13]. Often algorithms of this type are also called oblivious because the routing decisions of one packet do not affect the routing decisions of another. Work has shown that the problem of finding an appropriate set of virtual channels to in order to route, deadlock-free, for an entirely arbitrary graph is NP-complete [12], and heuristics have been provided to solve the problem.

The CODES simulator has a large body of work associated with it. Numerous researchers have utilized the CODES simulator for job-interference and performance studies with HPC application traces. Recent works following that trend include adding and evaluating new CODES network models [20, 23, 26, 32], expanding CODES to allow for quality-of-service traffic classes in Dragonfly models [26], and extending CODES network models to allow for multirail-multiplanar configurations of fat tree and Slim Fly [17, 22, 33]. Similar to the methodology of this work, each of these other CODES-based works analyze the performance of different interconnect network models with various benchmark workloads to exhibit the features of their specific contributions.

Other recent works with simulating the HPC application communication include an analysis of different network topologies with three dimensions of variability, also performed with the CODES simulator [3]. While the CODES simulator focuses on replaying traces of HPC applications, other tools, such as PyPassT [27], analyze parallel application source code and transform it into workloads for their Performance Prediction Toolkit to generate realistic performance estimations.

9 Future Work

We would like to perform a more thorough evaluation of multiplanar variants of other topologies including fat tree and Slim Fly.

We would also like to expand on the current link failure study. Fixing the number of link failures and observing how different configurations of networks behave would be helpful for figuring out how best to provision a network.

Another line of work would be to broaden the runs to scale the percentage of failed links with the number of planes so that when comparing a dual-plane network with a single-plane network with link failures, the dual-plane network would have twice the total number of failed links. In our current results it is possible to make this comparison by comparing data points in in Figures 5, 6, 7, and 8. With that comparison, although arguably within a limited range, our findings show that application performance is less affected by link failure in multirail-multiplanar networks.

We intend to build on our current work to conduct an in-depth study of how congestion can arise in a network and to develop methods for mitigation of this congestion. Link failures are a cause of congestion in networks; and, unless these failures are dealt with, more congestion will ensue.

10 Conclusion

In this work we presented the challenges of routing packets in Dragonfly-class interconnection networks with link failures. Specifically, when link failures are present, additional care must be taken when routing packets in order to avoid problems such as head-of-line blocking or deadlock. To utilize the network models in the CODES simulator with no modification of virtual channel assignment, we needed in order to dynamically find paths that did not violate the assignment constraints. We developed an organizational structure in CODES that uses Algorithm 1 to accomplish that task. Specifically, the algorithm enumerates valid paths used by the progressive adaptive routing algorithms implemented in the CODES Dragonfly-class network models so that these algorithms are failure-aware.

We also extended the 1D-Dragonfly and Megafly CODES network models to support multirail-multiplanar configurations. We studied how single- and dual-planar Dragonfly-class networks handle increased levels of link failure using the dynamic failure-aware adaptive routing.

We showed how simulation can be used to generate predictions of the behavior of irregular networks, such as those with failed links. We found that with a fixed number of failed links, dual-planar networks maintain a distinct advantage against communication failure and tolerance of link-failure-caused congestion over single-planar configurations, even with reduced bandwidth.

Acknowledgments

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

References

[1] Mo Adda and Adamantini Peratikou. 2017. Routing and fault tolerance in Z-fat tree. *IEEE Transactions on Parallel and Distributed Systems* 28, 8 (aug 2017), 2373–2386. https://doi.org/10.1109/TPDS.2017.2666807

[2] Diego F. Bermúdez Garzón, Crispín Gómez Requena, María Engracia Gómez, Pedro López, and José Duato. 2016. A family of fault-tolerant efficient indirect topologies. *IEEE Transactions on Parallel and Distributed Systems* 27, 4 (apr 2016), 927–940. https://doi.org/10.1109/TPDS.2015.2430863

[3] Abhinav Bhatele, Nikhil Jain, Misbah Mubarak, and Todd Gamblin. 2019. Analyzing cost-performance tradeoffs of HPC network designs under different constraints using simulations. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 1–12.

[4] Christopher D Carothers, David Bauer, and Shawn Pearce. 2002. ROSS: A high-performance, low-memory, modular Time Warp system. *J. Parallel and Distrib. Comput.* 62, 11 (2002), 1648–1669.

[5] Christopher D Carothers, Kalyan S Perumalla, and Richard M Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 9, 3 (1999), 224–253.

[6] Tiffany Connors, Taylor Groves, Tony Quan, and Scott Hemmert. 2019. Simulation framework for studying optical cable failures in dragonfly topologies. In *Proceedings - 2019 IEEE 33rd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019*. Institute of Electrical and Electronics Engineers Inc., 859–864. https://doi.org/10.1109/IPDPSW.2019.00141

[7] Jason Cope, Ning Liu, Sam Lang, Phil Carns, Chris Carothers, and Robert Ross. 2011. CODES: Enabling co-design of multilayer exascale storage architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies*, Vol. 2011. ACM.

[8] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT Press.

[9] William J Dally. 1990. Virtual-channel flow control. *ACM SIGARCH Computer Architecture News* 18, 2SI (1990), 60–68.

[10] William James Dally and Brian Patrick Towles. 2004. *Principles and practices of interconnection networks*. Elsevier.

[11] US DOE. 2016. Characterization of the DOE Mini-apps. http://portal.nersc.gov/project/CAL/designforward.htm [Online; posted 14-July-2016].

[12] Jens Domke, Torsten Hoefler, and Wolfgang E. Nagel. 2011. Deadlock-free oblivious routing for arbitrary topologies. In *Proceedings - 25th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2011*. IEEE, 616–627. https://doi.org/10.1109/IPDPS.2011.65

[13] José Flich, Tor Skeie, Andrés Mejía, Olav Lysne, Pedro López, Antonio Robles, José Duato, Michihiro Koibuchi, Tomas Rokicki, and José Carlos Sancho. 2012. A survey and evaluation of topology-agnostic deterministic routing algorithms. *IEEE Transactions on Parallel and Distributed Systems* 23, 3 (mar 2012), 405–425. https://doi.org/10.1109/TPDS.2011.190

[14] Center for Computational Sciences and Engineering Lawrence Berkeley National Laboratory. [n. d.]. BoxLib Codes. https://boxlib-codes.github.io/ [Online; retrieved 14-March-2019].

[15] Patrick T. Gaughan and Sudhakar Yalamanchili. 1995. A family of fault-tolerant routing protocols for direct multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems* 6, 5 (1995), 482–497. https://doi.org/10.1109/71.382317

[16] María Engracia Gómez, Nils Agne Nordbotten, José Flich, Pedro López, Antonio Robles, Jose Duato, Tor Skeie, and Olav Lysne. 2006. A routing methodology for achieving fault tolerance in direct networks. *IEEE Trans. Comput.* 55, 4 (apr 2006),

400–415. https://doi.org/10.1109/TC.2006.46

[17] Nikhil Jain, Abhinav Bhatele, Louis H Howell, David Böhme, Ian Karlin, Edgar A León, Misbah Mubarak, Noah Wolfe, Todd Gamblin, and Matthew L Leininger. 2017. Predicting the performance impact of different fat-tree configurations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[18] David R Jefferson. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7, 3 (1985), 404–425.

[19] Nan Jiang, John Kim, and William J Dally. 2009. Indirect adaptive routing on large scale interconnection networks. In *Proceedings of the 36th annual international symposium on Computer architecture*. 220–231.

[20] Yao Kang, Xin Wang, Neil McGlohon, Misbah Mubarak, Sudheer Chunduri, and Zhiling Lan. 2019. Modeling and Analysis of Application Interference on Dragonfly+. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 161–172.

[21] German Maglione-Mathey, Pedro Yebenes, Jesus Escudero-Sahuquillo, Pedro Javier Garcia, Francisco J. Quiles, and Eitan Zahavi. 2018. Scalable deadlock-free deterministic minimal-path routing engine for infiniband-based dragonfly networks. *IEEE Transactions on Parallel and Distributed Systems* 29, 1 (jan 2018), 183–197. https://doi.org/10.1109/TPDS.2017.2742503

[22] Neil McGlohon, Noah Wolfe, Misbah Mubarak, and Christopher D Carothers. 2019. Fit Fly: A Case Study on Interconnect Innovation through Parallel Simulation. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 137–148.

[23] Misbah Mubarak, Christopher D Carothers, Robert Ross, and Philip Carns. 2012. Modeling a million-node dragonfly network using massively parallel discrete-event simulation. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 366–376.

[24] Misbah Mubarak, Christopher D Carothers, Robert B Ross, and Philip Carns. 2014. Using massively parallel simulation for MPI collective communication modeling in extreme-scale networks. In *Proceedings of the Winter Simulation Conference 2014*. IEEE, 3107–3118.

[25] Misbah Mubarak, Christopher D Carothers, Robert B Ross, and Philip Carns. 2016. Enabling parallel simulation of large-scale HPC network systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2016), 87–100.

[26] Misbah Mubarak, Neil McGlohon, Malek Musleh, Eric Borch, Robert B Ross, Ram Huggahalli, Sudheer Chunduri, Scott Parker, Christopher D Carothers, and Kalyan Kumaran. 2019. Evaluating quality of service traffic classes on the Megafly network. In *International Conference on High Performance Computing*. Springer, 3–20.

[27] Mohammad Abu Obaida, Jason Liu, Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. 2018. Parallel Application Performance Prediction Using Analysis Based Models and HPC Simulations. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '18)*. Association for Computing Machinery, New York, NY, USA, 49–59. https://doi.org/10.1145/3200921.3200937

[28] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, Rolf Risen, Jeanine Cook, Paul Rosenfeld, Elliot Cooper-Balis, et al. 2011. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review* 38, 4 (2011), 37–42.

[29] Frank Olaf Sem-Jacobsen, Tor Skeie, Olav Lysne, and José Duato. 2011. Dynamic fault tolerance in fat trees. *IEEE Trans. Comput.* 60, 4 (2011), 508–525. https://doi.org/10.1109/TC.2010.97

[30] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi. 2017. Dragonfly+: Low cost topology for scaling datacenters. In *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE, 1–8.

[31] Sstsimulator. 2018. sstsimulator/sst-dumpi. https://github.com/sstsimulator/sst-dumpi

[32] Noah Wolfe, Christopher D Carothers, Misbah Mubarak, Robert Ross, and Philip Carns. 2016. Modeling a million-node slim fly network using parallel discrete-event simulation. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 189–199.

[33] Noah Wolfe, Misbah Mubarak, Nikhil Jain, Jens Domke, Abhinav Bhatele, Christopher D Carothers, and Robert B Ross. 2017. Preliminary performance analysis of multi-rail fat-tree networks. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 258–261.

[34] Dong Xiang, Bing Li, and Yi Fu. 2019. Fault-Tolerant Adaptive Routing in Dragonfly Networks. *IEEE Transactions on Dependable and Secure Computing* 16, 2 (mar 2019), 259–271. https://doi.org/10.1109/TDSC.2017.2693372