Topology-Custom UGAL Routing on Dragonfly

Md Shafayat Rahman
rahman@cs.fsu.edu
Department of Computer Science
Florida State University
Tallahassee, Florida

Saptarshi Bhowmik
bhowmik@cs.fsu.edu
Department of Computer Science
Florida State University
Tallahassee, Florida

Yevgeniy Ryasnianskiy
ryasnian@cs.fsu.edu
Department of Computer Science
Florida State University
Tallahassee, Florida

Xin Yuan
xyuan@cs.fsu.edu
Department of Computer Science
Florida State University
Tallahassee, Florida

Michael Lang
mlang@lanl.gov
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

The Dragonfly network has been deployed in the current generation supercomputers and will be used in the next generation supercomputers. The Universal Globally Adaptive Load-balance routing (UGAL) is the state-of-the-art routing scheme for Dragonfly. In this work, we show that the performance of the conventional UGAL can be further improved on many practical Dragonfly networks, especially the ones with a small number of groups, by customizing the paths used in UGAL for each topology. We develop a scheme to compute the custom sets of paths for each topology and compare the performance of our topology-custom UGAL routing (T-UGAL) with conventional UGAL. Our evaluation with different UGAL variations and different topologies demonstrates that by customizing the routes, T-UGAL offers significant improvements over UGAL on many practical Dragonfly networks in terms of both latency when the network is under low load and throughput when the network is under high load.

CCS CONCEPTS

• **Networks** → **Routing protocols**; Network performance analysis; • **Computer systems organization** → *Interconnection architectures*;

KEYWORDS

Interconnection network, Dragonfly, UGAL routing

ACM Reference Format:
Md Shafayat Rahman, Saptarshi Bhowmik, Yevgeniy Ryasnianskiy, Xin Yuan, and Michael Lang. 2019. Topology-Custom UGAL Routing on Dragonfly. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 12 pages.

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356208

1 INTRODUCTION

The Dragonfly topology is cost-effective for large-scale interconnection networks [1]. The Cray Cascade architecture [2] employs a variation of Dragonfly, and has been deployed in current supercomputers including Titan at Oak Ridge National Laboratory [3] and Trinity at Los Alamos National Laboratory [4]. The Cray Slingshot network, designed for future exascale computing, also uses the Dragonfly topology [5].

The routing to achieve high performance on Dragonfly is challenging: different routing schemes must be used for different traffic conditions to achieve high performance [1]. In particular, minimal routing (MIN) is suited for uniform traffic while non-minimal Valiant Load-balanced routing (VLB) is required for adversarial traffic patterns. To unify the two routing schemes in one system, the Universal Globally Adaptive Load-balanced routing (UGAL) has been developed [1] that adapts the routing decision for each packet between MIN and VLB paths based on queue lengths [6].

A Dragonfly network has a two-layer structure and consists of a number of groups. A practical Dragonfly topology typically has a fixed group topology, but can configure the number of groups and the number of global links between each pair of groups [2]. Different configurations will have very different connectivity characteristics. For example, for full-sized Dragonfly topologies, there is only one link between each pair of groups [1] while for some other topologies, there are many links between each pair of groups[2], resulting in significant path diversity using just the MIN paths. Note that path diversity means the number of different paths that can be used to transmit a packet. For an adversarial traffic pattern, having path diversity in a routing scheme is essential for the routing scheme to explore network capacity and achieve high performance. Yet, the conventional UGAL does not adapt to topologies in the sense that the method to decide MIN or VLB paths is the same across different Dragonfly topologies. Although UGAL has been shown to achieve high performance on Dragonfly, it may not achieve the best routing performance for different Dragonfly topologies.

In this work, we propose topology-custom UGAL routing (T-UGAL) that has the same routing mechanism and the same set of MIN paths as the conventional UGAL. T-UGAL uses different sets of VLB paths that are customized for each topology. We develop a general approach to determine the custom VLB paths for

each topology and compare the performance of T-UGAL with the conventional UGAL on different topologies and different UGAL variations including UGAL-L (UGAL with local information) [1], progressive adaptive routing (PAR) [7], and the theoretical UGAL-G (UGAL with global information) [1]. The results show that T-UGAL significantly improves over UGAL in terms of both latency when the network is under low load and throughput when the network is under high load for many practical topologies. Our results also show for Dragonfly topologies where there is only one link between each pair of groups, UGAL converges with T-UGAL and achieves high performance. Such networks were used in the design and evaluation of the original UGAL for Dragonfly [1, 7]. Note that T-UGAL only changes the set of candidate paths for UGAL, and thus is complement to other schemes developed to improve UGAL [8–13] and can be applied to any UGAL variations.

The rest of the paper is structured as follows. Section 2 presents the background and related work. Section 3 describes our proposed topology-custom UGAL. Section 4 reports the performance study. Finally, Section 5 concludes the paper.

2 BACKGROUND AND RELATED WORK

This section gives background and related work. We will describe the Dragonfly topology and the UGAL routing, and discuss the related work to improve UGAL on Dragonfly.

2.1 Dragonfly Topology

Details about the Dragonfly topology can be found in Kim et al.'s original paper [1]. Here, we will briefly introduce the topology for the completeness of this paper. The Dragonfly topology has a 2-layer structure. A group of routers/switches are interconnected with an intra-group topology into a *group* that can be treated as a single virtual router with a very high radix. We will use the terms router and switch interchangeably. The groups are then connected with an inter-group topology. Figure 1 shows an example of the 2-layer dragonfly topology. In this example, each group consists of 4 switches; there are a total of 9 groups in the system.

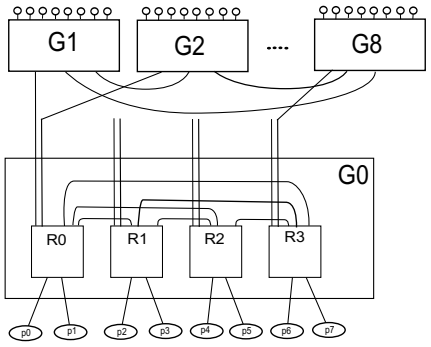

Figure 1: An example Dragonfly topology ($dfly(p = 2, a = 4, h = 2, g = 9)$)

Various topologies can be used to form the intra-group connectivity. A typical intra-group topology is a fully connected graph where all switches are directly connected to each other [1]. An example of such an intra-group topology is shown in the G0 group in Figure 1. The intra-group connectivity in the Cascade architecture is a 2-dimensional all-to-all mesh [2]. In this work, we focus on Dragonfly networks with a fully connected intra-group topology, which is used in the Cray's newer generation Slingshot network [5]. Our techniques, however, can be applied to other Dragonfly variations.

The number of groups in a Dragonfly can vary. The largest possible Dragonfly has only one *global* link connecting each pair of groups. When the intra-group topology is a fully connected graph, the largest Dragonfly is uniquely defined by three parameters: the number of links per switch connecting to local compute nodes p, the number of switches in each group a, and the number of global links per switch connecting to switches in other groups h. In such a topology, the number of ports in each switch is $p + a - 1 + h$; the number of global links from each group is $a \times h$; and the number of groups is $a \times h + 1$; the total number of switches is $(a \times h + 1) \times a$; and the total number of compute nodes is $(a \times h + 1) \times a \times p$. As discussed in [1], a load-balanced Dragonfly system should have $a = 2p = 2h$. Figure 1 illustrates a balanced Dragonfly network with $p = 2$, $a = 4$, and $h = 2$. In this case, each group has $a = 4$ switches and $a \times h = 8$ global links. The largest topology can thus have $8 + 1 = 9$ groups with each group having one global link connecting to any other group.

The number of groups (g) in a Dragonfly topology may be smaller than $a \times h + 1$. For example, Figure 2 shows a topology where $p = h = 2$, $a = 4$, and $g = 3$. As shown in the figure, 4 global links are used to connect each pair of groups. Different ways to arrange the global connectivities have been proposed [14] including *relative*, *absolute*, and *circulant-based* arrangements. In this paper, we assume the global links are connected using a minor variation of **absolute** arrangement. The variation is able to form bi-directional Dragonfly topology with different numbers of groups. Our techniques, however, do not depend on the link arrangement schemes and can be applied for Dragonfly with different link arrangement schemes. With the assumption of the global link arrangement, the Dragonfly topology can be determined with 4 parameters: p, h, a, and g. We will use the notation $dfly(p, a, h, g)$ to represent such a topology. Figure 2 shows the topology of $dfly(p = 2, a = 4, h = 2, g = 3)$ while Figure 1 is the topology of $dfly(p = 2, a = 4, h = 2, g = 9)$.

In the past, global links (the longer links between groups) incur significantly higher latency than local links (the shorter links within a group), a ratio of 1:10 [7]. However, recent technological advances have significantly reduced the latency gap between global and local links. In the Cascade architecture [2], the latency of global links is roughly 1.5 times the latency of local links (depending on the length of the global cable).

2.2 Universal Globally Adaptive Load-balanced Routing (UGAL)

The following terminology will be used to describe routing in Dragonfly. Packets are routed from a *source compute node* to a *destination compute node*. The switch that the source compute node connects to is called the *source switch*. The switch that the destination compute node connects to is called the *destination switch*. The group that the source compute node is in is called the *source group*;

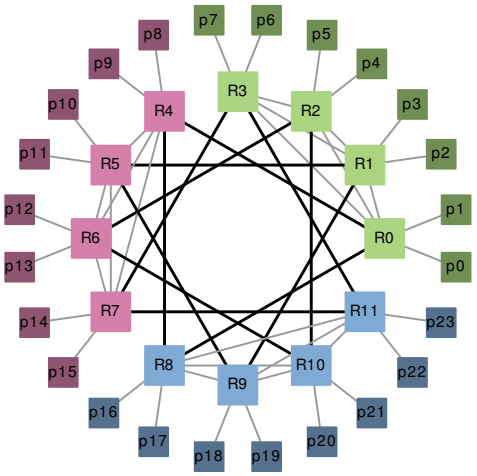

Figure 2: An example Dragonfly topology with absolute global link arrangement ($dfly(p = 2, a = 4, h = 2, g = 3)$)

the group that the destination compute node is in is called the *destination group*. We will use the notation $s \rightarrow d$ to represent a link from node s to node d.

In the Dragonfly topology, packets are routed along either a *minimal* or a *non-minimal path*. A minimal path is a path from the source compute node to the destination compute node that contains at most one global link. The thick segmented line in Figure 3 shows a typical minimal path from s to d, where the path takes one local hop in the source group from the source switch to the switch that has a global link to the destination group, then the global link to the destination group, and finally a local link at the destination group to the destination switch. Depending on the positions of the source and the destination, the minimal path may have fewer hops.

The Minimal routing (MIN) scheme routes packets only with minimal paths. It minimizes the resource usage and works well for traffic patterns where MIN can evenly distribute the load such as the random uniform traffic. However, since the number of links between each pair of groups is small (e.g. one link in $dfly(p = 2, a = 4, h = 2, g = 9)$), for traffic patterns where many nodes in one group must communicate to many nodes in another group, the MIN routing will perform poorly since all of the traffic from one group to another must use the small number of links between the two groups. Such traffic patterns are considered adversarial in Dragonfly.

To avoid congestion on global links for an adversarial traffic pattern, Valiant Load-balanced routing (VLB) [15] can be used to spread non-uniform traffic evenly over the set of available links. A VLB path can be considered as using MIN to find a path from the source to a randomly selected intermediate switch that is not in the source and destination groups, and then, from the intermediate switch to the destination. A VLB path is non-minimal as it uses two global links to route inter-group traffic packets. Figure 3 shows a 6-hop VLB path in solid thick lines. With a VLB route, a packet is first sent to an intermediate router (R_i in this example) that is randomly selected from routers not in the source and destination groups, and then to the destination.

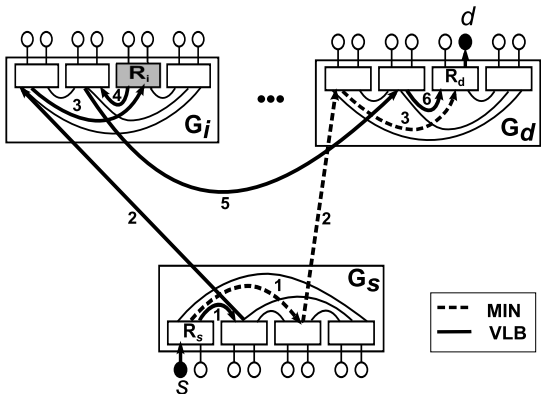

Figure 3: MIN and VLB routing on Dragonfly

The Universal Globally Adaptive Load-balanced routing (UGAL) selects among MIN and VLB paths for each packet based on the traffic condition. The traffic condition is inferred from the occupancy of packet queues of the network sensed at the source switch. For each packet, UGAL first randomly selects a small number of candidate MIN and VLB paths from all possible MIN and VLB paths for further consideration for routing. In the original UGAL proposal and its Dragonfly adaptation, the number of MIN paths is 1 and the number of VLB paths is 1 [1, 6]. After that, among the candidate paths, UGAL selects one that it anticipates would achieve the smallest packet delay.

An ideal UGAL routing, called UGAL-G (UGAL with global information) [1], assumes that the precise global network state information is available, and estimates the smallest packet delay along a path using the total queue length on all links along the path. Let TQ_{MIN} be the total queue length for the MIN path, and TQ_{VLB} be the total queue length for the VLB path. UGAL-G selects the MIN path if

$$TQ_{MIN} \leq TQ_{VLB} + T$$

and the VLB path otherwise. Here, T is an offset constant that can be tuned to decide how much the path selection will be biased toward MIN paths (a large value of T giving preference to MIN paths). Clearly, UGAL-G is a theoretical scheme that cannot be implemented under existing technological constraints. Other practical UGAL-based schemes [2, 7] use some practical methods to estimate the packet delay and approximate UGAL-G. In this work, we evaluate our topology-custom UGAL with three UGAL variations: the theoretical UGAL-G and two practical UGAL routing schemes, UGAL-L (UGAL with local information) [1] and progressive adaptive routing (PAR) [7]. UGAL-L uses the local queue length multiplied by the path length to estimate the total queue length along the path. PAR behaves like UGAL-L, but allows the routing decision for a packet to be revised in the second hop within the group when it decides to use a MIN path for the packet in the first switch.

2.3 Related Work

Many techniques have been developed to improve UGAL performance for the Dragonfly topology. When the Dragonfly network was first introduced, Kim et al. proposed selecting a random intermediate group to route non-minimally in order to load-balance adversarial traffic patterns over global channels [1]. Jiang et al. proposes several adaptive routing heuristics that approximate UGAL-G [7]. Improvements over the original UGAL-based scheme have been developed. Garcia et al. [8] are the first to address local congestion inside Dragonfly groups. They proposed using non-minimal routing on both intra- and inter-group communication in their OFAR routing scheme. OFAR-CM [9] proposes throttling packet injection at local nodes as well as routing through an escape subnetwork to mitigate congestion on OFAR routing at the cost of additional hops. Opportunistic local misrouting (OLM) [10] allows non-minimal routing on both local and global levels of the Dragonfly hierarchy and the routing decision may be updated at any hop. Traffic pattern-based adaptive routing enhances UGAL by using local counters to infer the traffic pattern and guide routing decisions in the system [13]. Improvements for load estimation with UGAL-based routing schemes have also been developed [11, 12]. This work is different from the existing research in that we develop systematic techniques to find subsets of all possible VLB paths to be used in UGAL based on the Dragonfly topology, which results in improved throughput at high load and reduced latency at low load in comparison to the traditional UGAL as shown in our evaluation. Our schemes are orthogonal to all of the existing techniques to improve UGAL performance on Dragonfly and can be combined with the techniques to further optimize the routing performance on Dragonfly. Using limited length paths for load balance routing has also been proposed in different contexts and different network topologies [6]. However, none of the existing techniques can be directly applied to the Dragonfly network.

3 TOPOLOGY-CUSTOM UGAL ROUTING (T-UGAL)

Topology-custom UGAL routing (T-UGAL) is essentially the same as UGAL except that the set of candidate VLB paths is customized for each topology. The set of candidate VLB paths in T-UGAL, which will be denoted as T-VLB, is a subset of all VLB paths in UGAL and has a smaller average path length than the average path length of all VLB paths. This allows T-UGAL to use, on average, a shorter path to deliver each packet, which results in less network resource usage for each packet (and thus less overall network load for a given user traffic) and potentially improves the overall routing performance. The challenge is **to ensure that T-VLB has sufficient path diversity for all traffic conditions**. Note that our focus is on performance. Thus, if T-UGAL that uses T-VLB has equal or better performance than the conventional UGAL for a traffic condition, we consider the T-VLB to have sufficient path diversity for the traffic condition. In the following, we will first discuss the motivation for T-UGAL, list important properties of T-UGAL, and present the algorithm to determine the custom T-VLB for any given Dragonfly topology.

3.1 Motivation

Given a topology, a routing scheme that achieves good performance should in general have two properties: (1) the routing should use a minimal amount of network resources to deliver each packet, and (2) the routing should be able to distribute traffic evenly over the network. The first property is generally achieved by using shorter paths for each packet; and the second property requires that the routing scheme can exploit path diversity to achieve load-balance. UGAL has these two properties by adapting between MIN and VLB paths: when possible, UGAL delivers packets using MIN paths, which are short; when necessary, UGAL uses the long VLB paths for path diversity and load balancing.

However, the method to decide candidate VLB paths in UGAL does not adapt to the network topology. As discussed in Section 2, for any Dragonfly topology, the VLB paths used in UGAL are obtained by randomly selecting an intermediate switch; and a VLB path consists of two MIN paths: one from the source to the intermediate switch and the other one from the intermediate switch to the destination. As shown in Figure 3, a typical MIN path has 3 hops and a typical VLB path has 6 hops. Although VLB routing provides path diversity and thus load-balancing, there is a question whether **all** the VLB paths are necessary for path diversity on all Dragonfly topologies.

In some Dragonfly topologies such as the one in Figure 2, there are many links between each pair of groups. In such a topology, there exist many 3-hop, 4-hop, and 5-hop VLB paths that are shorter than the typical 6-hop VLB paths. As will be shown later, many practical Dragonfly networks have a large number of such shorter paths that can provide sufficient path diversity even for the most demanding adversarial traffic patterns. For such topologies, if one can find the set of shorter VLB paths that can provide sufficient path diversity, and restrict the candidate VLB paths to these shorter paths, the performance of UGAL will be improved.

Using shorter paths with sufficient path diversity has clear advantages. To simplify the discussion, let us assume that the average length of MIN paths is 3; the average length of VLB paths is 6. Let us further assume that 70% of packets are delivered with MIN paths. Using UGAL, each packet on average goes through $0.7 \times 3 + 0.3 \times 6 = 3.9$ hops. If T-UGAL can reduce the average length of VLB paths to be 4.8 hops, each packet would on average go through $0.7 \times 3 + 0.3 \times 4.8 = 3.54$ hops. Assume that the load balancing property for UGAL and T-UGAL is the same, T-UGAL will enjoy a $\frac{3.9}{3.54} - 1 \approx 10\%$ reduction in packet latency when the network is under low load and 10% reduction of the network overall load when the network is under high load.

Another important question is how to make sure that T-VLB paths have sufficient path diversity (T-UGAL performs at least as good as the conventional UGAL). For a general network, this is a difficult question to answer. However, Dragonfly has a unique property that the most demanding traffic patterns that require most path diversity to support are known: the adversarial shift traffic patterns [1, 13]. Since the inter-group connectivity in any Dragonfly is a fully connected network with the same number of (one or more) direct links connecting each pair of groups, the number of direct global links between one pair of groups is always small compared to the total number of global links in the whole network: the

ratio of 1 to $g(g-1)$ remains the same regardless of the number of direct links between groups. The adversarial shift traffic patterns incur maximum traffic from one group to another throughout the network. Such a pattern can quickly consume the capacity on the direct links between the two groups and force UGAL to use VLB paths. This applies to all Dragonfly; and the adversarial shift traffic patterns are adversarial also for non-maximal Dragonfly with many links between each pair of groups. If T-UGAL can support such patterns better than or at least as well as UGAL, it should perform at least as well as UGAL for any other traffic condition on the network (having sufficient path diversity for any traffic condition).

Practical, deployable Dragonfly networks such as Cascade and Slingshot, have a fixed group structure; and the systems can be configured with different numbers of groups and different numbers of global links between each pair of groups. Our techniques can be applied to find T-VLB for any of such topologies and achieve improved routing performance.

3.2 T-UGAL Properties

Our scheme selects paths in T-VLB based on the topology in such a way that T-UGAL has the following properties:

(1) T-UGAL achieves higher or similar performance in comparison to UGAL for the most demanding adversarial traffic patterns. The most demanding adversarial traffic patterns are the ones that require most path diversity for performance. The idea is that if T-UGAL can have higher or similar performance for such patterns, it should be able to achieve higher or similar performance for any other traffic patterns, which are less demanding.
(2) The average path length of T-VLB is as small as possible. The advantage of T-UGAL over the conventional UGAL is using shorter paths for communications, which results in low packet latency as well as less average network resources to deliver a packet that can yield higher throughput at high load.
(3) T-UGAL has a similar load-balancing property as UGAL. The load balancing property is essential for any routing scheme to achieve high performance.

3.3 Computing T-VLB

3.3.1 Most Demanding Adversarial Patterns for Dragonfly. The main idea of T-UGAL is to find T-VLB with the smallest average path length while being able to provide sufficient path diversity even for the most demanding traffic patterns for the network. Thus, the first problem is to find the most demanding traffic patterns for Dragonfly. Finding representative adversarial traffic patterns for a general network is challenging if at all possible. Luckily, as explained in Section 3.1, for Dragonfly the most demanding adversarial patterns are well known [1, 13]: the shift pattern where all compute nodes from one group communicate with all compute nodes in another group. There are different forms of such adversarial patterns. Consider $dfly(p, a, h, g)$. The topology consists of g group; each group has a switches; and each routing has p compute nodes. We will use the notation (g_i, s_j, n_k) to denote the k-th compute node in the j-th switch in the i-th group, $0 \le i < g$, $0 \le j < a$, $0 \le k < p$. To ensure that different forms of adversarial traffic are considered in the design of T-UGAL, we consider two types of shift patterns.

The first type of patterns is denoted as $shift(\Delta_g, \Delta_s)$ where node (g_i, s_j, n_k) transmits to node $(g_{(i+\Delta_g) \bmod g}, s_{(j+\Delta_s) \bmod a}, n_k)$. The set of this type of patterns used in our procedure is

TYPE_1_SET = {
$shift(1,0), shift(1,1), ..., shift(1,a-1)$,
$shift(2,0), shift(2,1), ..., shift(2,a-1)$,
... ,
$shift(g-1,0), shift(g-1,1), ..., shift(g-1,a-1)$
}

The $TYPE\_1\_SET$ contains the traffic patterns where each group shifts to any other group. Additionally, nodes from each switch also shift to every other switches. This set contains $(g-1)a$ patterns.

The second type of patterns can be specified by first having a random permutation at the group level, and then having a random permutation at the switch level for each source and destination group pair in the group permutation. Consider a Dragonfly with 3 groups and each group having 4 switches. An example group-level random permutation can be $0 \to 2 \to 1 \to 0$. After the group-level permutation is generated, switch-level permutations are generated for each of the group-level communications. For example, for group-level communication $0 \to 2$, an example switch-level permutation can be $0 \to 0$ and $1 \to 2 \to 3 \to 1$. In this case, the communication pattern from nodes in group 0 to group 2 includes the following traffic: nodes (g_0, s_0, n_k) send to (g_2, s_0, n_k), $0 \le k < p$; nodes (g_0, s_1, n_k) send to (g_2, s_2, n_k); nodes (g_0, s_2, n_k) send to (g_2, s_3, n_k); nodes (g_0, s_3, n_k) send to (g_2, s_1, n_k). We include 20 of such random patterns in $TYPE\_2\_SET$.

Patterns in $TYPE\_1\_SET$ and $TYPE\_2\_SET$ are representative for different adversarial traffic patterns on Dragonfly. To support these patterns effectively, maximum path diversity is necessary. On the other hand, since any other pattern will be more "uniform" in comparison to these adversarial patterns, if a routing scheme is able to support these patterns effectively, the path diversity supported by the routing scheme should also be able to support any other pattern effectively (or at least as effectively as UGAL).

3.3.2 Deciding T-VLB: Step 1 - Coarse-Grain Estimation of T-VLB. Now that we know the most demanding traffic patterns, the next step is to find the subset VLB paths for all pairs of source-destination switches. Our scheme uses a two-step approach to obtain T-VLB. The first step, which performs coarse-grain estimation, uses a performance model to evaluate many potential data points (different subsets of VLBs) and find a small number of candidate configurations. In the second step, after some fine-tuning is performed, the final T-VLB is decided through simulation (if there is a working system, T-VLB for the system can be decided by experimentation).

The performance model that we use is a minor modification of the Model No. 3 in [16]. The model is based on linear programming and has been shown to quite accurately model the UGAL throughput for adversarial traffic patterns [16]. Additionally, the model is efficient and can obtain the modeling results for adversarial traffic patterns even for very large Dragonfly networks with tens of thousands of end points. When adopting the model for this work, we found that although the model is accurate for UGAL with all

VLB paths, the accuracy drops when a small percentage of 5-hop or 6-hop paths are used. To overcome this problem, we modified the model by adding constraints in the linear programming formulation to enforce that the data rate allocated for a longer VLB path for a source-destination pair is no more than the data rate allocated for a shorter VLB path for the same source-destination pair. This is because UGAL has an inherent tendency to prefer shorter paths over longer ones when such paths are available, and thus a model allocating higher data rate to some specific longer paths even while enough shorter paths are available will overestimate the final throughput. We validate this enhanced model, which consistently produces accurate results for different compositions of VLB paths, and use it in our coarse-grain estimation. We note that the specific performance model is not a requirement for our scheme, any mechanism that allows the probing of a large number of configurations can replace the model.

Table 1 lists the data points (or configurations) that are probed in Step 1. For any $dfly(p, a, h, g)$, the number of hops for VLB paths is between 2 and 6. Each data point is applied to all source-destination switches in a synchronized manner. For example, the point "4-hop paths" means that all pairs of switches will use all VLB paths that are 4 hops or less in length. Similarly, the configuration "20% 5-hop paths" means that all pairs of switches will use all of their VLB paths that are 4 hops long or less in addition to the 20% randomly selected 5-hop paths. In this step, the paths are randomly selected if not fully specified. For each data point, the modeled performance for the patterns in $TYPE\_1\_SET$ and $TYPE\_2\_SET$ is obtained and the average performance is used to identify the best performing data point. For example, Figure 4 shows the average modeled throughput for $dfly(4, 8, 4, 9)$. As can be seen in the figure, the best performing data point in this case is "60% 5-hop".

Notation	explanation
3-hop paths	all paths 3-hop or less
10% 4-hop paths	all paths 3-hop or less plus 10% 4-hop paths
20% 4-hop paths	all paths 3-hop or less plus 20% 4-hop paths
...	...
90% 4-hop paths	all paths 3-hop or less plus 90% 4-hop paths
4-hop paths	all paths 4-hop or less
10% 5-hop paths	all paths 4-hop or less plus 10% 5-hop paths
...	...
90% 5-hop paths	all paths 4-hop or less plus 90% 5-hop paths
5-hop paths	all paths 5-hop or less
10% 6-hop paths	all paths 5-hop or less plus 10% 6-hop paths
...	...
90% 6-hop paths	all paths 6-hop or less plus 90% 6-hop paths
6-hop paths	all VLB paths

Table 1: The data points probed in coarse-grain Step 1

Once the best performance point is identified, we consider a small number of data points in the vicinity of the best performance point and form a set of candidate data points that are considered in Step 2. For $dfly(4, 8, 4, 9)$, as shown in Figure 4, 40% 5-hop, 50% 5-hop, and 70% 5-hop all have similar performance as 60% 5-hop based on the model. We pass all four as candidate data points to be considered in Step 2.

We note that since the performance model is efficient, one may repeat this step multiple times to deal with the problem potentially caused by the randomization (e.g. a bad random seed). However, in all of our experiments, randomization has never caused a problem.

3.3.3 Deciding T-VLB: Step 2 – Finalizing T-VLB. After we obtain the candidate configurations from Step 1, we first examine the candidate sets to possibly expand the set by including some deterministic strategic choices. These deterministic choices are easier to obtain and reason. Our experiments show that such sets of paths sometimes have special characteristics that help the performance in routing schemes. For example, for $dfly(4, 8, 4, 9)$, Step 1 results in four candidate choices: 40% 5-hop, 50% 5-hop, 60% 5-hop, and 70% 5-hop. Since there are two ways to obtain a 5-hop VLB path: (1) a 2-hop MIN path (from the source to the intermediate node) followed by a 3-hop MIN path (from the intermediate node to the destination) and (2) a 3-hop MIN path followed by a 2-hop MIN path. Thus, 50% 5-hop paths can strategically be obtained either by having all 2-hop MIN paths followed by 3-hop MIN paths or by having all 3-hop MIN paths followed by 2-hop MIN paths. We include these two strategic choices in the candidate set. Hence, in Step 2, we will consider six candidate configurations for $dfly(4, 8, 4, 9)$: the four from Step 1 and the 2 strategic path choices for 50% 5-hop. Note that the paths in T-VLB may not be obtained by specifying an intermediate switch only, since T-VLB may have restrictions on the hop count of the MIN paths which need to be considered while determining a path to reach the intermediate switch.

For each of the candidate configurations, our scheme checks the load balancing properties of the set of VLB paths, performs load balancing adjustments, and then evaluates the adjusted candidates through simulation. For the traditional UGAL routing, since all intermediate routers can be selected for VLB paths, the VLB paths use the links in the network in a balanced manner due to the symmetricity of the Dragonfly topology. T-VLB uses a subset of VLB paths and can potentially result in an imbalanced use of links. The imbalance may happen in two levels: locally for each pair of switches when some links are significantly more likely to be used to carry the traffic of this pair of switches than other links, and globally for all pairs of switches when some links are significantly more likely to be used to carry traffic than others.

In our scheme, we detect local imbalance by computing the probability of link usage for each pair of switches under the assumption that all VLB paths for the pair of switches are equally likely to be used and then checking if some link usage probability is significantly higher than others. We detect the global imbalance by computing the probability of link usage of all links under the assumption that a packet between any pair of switches is equally likely. When such imbalances are detected, we perform simple load balance adjustments by removing paths that cause high link usage probability either at the local level (per pair of switches) or at the global level (all pairs of switches). In theory, imbalance can also be removed by replacing paths that use highly loaded links by paths that do not use highly loaded links. But the simple mechanism of just removing paths work sufficiently well in our experiments, so we do not explore other mechanisms. We note that UGAL can itself

tolerate some imbalanced use of paths and still achieve high performance in the presence of some level of imbalanced link usage, which has been shown in other studies [17].

After the load balancing adjustments, we simulate all candidate sets using adversarial patterns and select the highest performing candidate as the final T-VLB. In our experiments, we simulate 5 patterns from $TYPE\_2\_SET$ and use the average throughput of the 5 patterns as the final performance metric.

3.3.4 Put It All Together. Algorithm 1 shows the whole procedure to determine the T-VLB. The procedure takes a dragonfly topology $dflg(p, a, h, g)$ as input and outputs T-VLB. The procedure first computes the adversarial patterns $TYPE\_1\_SET$ and $TYPE\_2\_SET$ (Line 3). Lines 4 to 7 sort the VLB paths based on the path length and randomize the order of VLB paths of the same length. Lines 8 to 12 is the Step 1 coarse-grain estimation of the number of VLB paths needed as described in Section 3.3.2. Lines 13 to 21 is the Step 2 finalizing T-VLB described in Section 3.3.3. Finally, Line 22 outputs the results.

```
1  Input: A Dragonfly topology, dfly(p, a, h, g)
2  Output: The sets of T-VLB paths for all pairs of switches
3  Generate TYPE_1_SET and TYPE_2_SET for the topology
4  for each pair of switches do
5  |   Compute all VLB paths and randomize the order
6  |   Sort all VLB paths based on the path length
7  end
8  /* Step 1: coarse-grain */
9  Compute the modeled throughput for all patterns and all VLB
     subsets described in Table 1
10 Find the point with the largest average modeled throughput
11 Decide candidate set in the vicinity of the largest average
     throughput
12 /* end of Step 1 and begin of Step 2 */
13 Expand the candidate set if necessary
14 for each candidate configuration do
15 |   Compute per pair link usage probability
16 |   Do local load balance adjustment if necessary
17 |   Compute all-to-all link usage probability
18 |   Do global load balance adjustment if necessary
19 |   Simulate a set of traffic patterns and record the results
20 end
21 The data point with the highest average simulated
     throughput is selected
22 Output the associated VLB paths, which is T-VLB
```

Algorithm 1: Algorithm to determine T-VLB for any $dfly(p, a, h, g)$

4 PERFORMANCE STUDY

Extensive simulation studies have been carried out to study T-UGAL and compare its variations with the corresponding conventional UGAL variations. The experiments were designed to investigate the characteristics of T-UGAL as well as its routing performance on different topologies. In the following, we first describe our experimental methodology and simulation settings and then report the performance results.

4.1 Methodology

4.1.1 Topology. We present results for topologies $dfly(p = 4, a = 8, h = 4, g = 33)$, $dfly(p = 4, a = 8, h = 4, g = 17)$, and $dfly(p = 4, a = 8, h = 4, g = 9)$. These topologies are built with 15-port switches and represent a range of Dragonfly topologies with different connectivity characteristics: these topologies have the same intra-group connectivity, but different numbers of groups as well as different numbers of links connecting each pair of groups. We also report results on $dfly(p = 13, a = 26, h = 13, g = 27)$, a larger topology, to demonstrate that T-UGAL also works for larger topologies. Table 2 lists the major parameters of topologies.

Topology	No. of PEs	No. of switches	No. of groups	links per group pair
$dfly(4, 8, 4, 33)$	1056	264	33	1
$dfly(4, 8, 4, 17)$	544	135	17	2
$dfly(4, 8, 4, 9)$	288	72	9	4
$dfly(13, 26, 13, 27)$	9126	702	27	13

Table 2: Topologies used in the experiments

4.1.2 Routing Variations and Simulator Settings. We use Booksim 2.0, a cycle-accurate interconnection network simulator [18], in our experiments. The Dragonfly topology code that Booksim comes with always creates a network where g = a * h + 1. We extend the code to accommodate different network sizes and implement the absolute arrangement of global links [14]. Booksim provides a basic implementation of UGAL-L. To study the performance of other UGAL variations, we added PAR and UGAL-G and incorporated T-UGAL with the three variations: UGAL-L, UGAL-G and PAR. We will use the notations T-UGAL-L, T-PAR, and T-UGAL-G to denote the three variations of T-UGAL. UGAL-L, PAR, UGAL-G are briefly described in Section 2.2. We note that UGAL-L and PAR are practical and can be deployed while UGAL-G is an ideal-case scheme.

UGAL routing selects from one candidate MIN path and one candidate VLB path. The adaptive routing threshold (the value T described in Section 2.2) is set to zero, so the routing schemes do not bias towards MIN or VLB paths.

Booksim supports the traditional four-stage pipeline router microarchitecture. Input-queued routers can suffer from head-of-line blocking which may degrade the overall performance of the system. For the studies like ours where the focus is on the performance of the routing algorithms instead of the router microarchitecture, Booksim provides a feature to increase the speed of the router's internal pipeline. With a speedup of 2, the router pipeline runs at twice the speed of the network channels. For our simulations, we set the default speedup to 2 as in [11].

We use virtual-channels (VCs) to ensure deadlock-freedom in the network. As demonstrated by Won et el. [11], UGAL in Dragonfly requires 4 virtual channels to ensure no deadlock can occur. We follow their VC-allocation scheme and use 4 VCs per channel for UGAL-L and UGAL-G. For PAR, a flit can take one extra hop at the source group while switching from a minimal to non-minimal route. So we allocate 5 VCs for PAR to accommodate the extra hop.

In the seminal Dragonfly paper [1], Kim et al. set both the local latency and global latency in their simulations to 1 cycle. More recently, a number of researchers [7, 8, 10, 11] used 10 cycles as local channel latency and 100 cycles as global channel latency. In this work, we set the latency of local and global channel to 10 and 15 cycles, respectively. The reasoning behind these numbers is to mimic the Cascade [2] system architecture, where the ratio of global and local link latency is roughly 1.5 to 1.

Booksim uses credit-based flow control for buffer management among adjacent routers. Credits are sent back to the opposite direction when a packet reaches its destination. In order to accommodate this round-trip delay, we set each virtual channel buffer size to 32. This is in accordance with [11] and [8].

We use single-flit packets to avoid any potential flow-control issue, which is not the focus of our study. The default network parameters in the simulation are summarized in Table 3. Besides the default parameters, we also study the sensitivity of the routing performance to various network parameters including link latency, buffer length, virtual channel allocation scheme, and switch speedup.

Parameter	value
# of virtual channels	4 for UGAL-L and UGAL-G 5 for PAR
buffer size	32
link latency	10 cycles (local) 15 cycles (global)
switch speed-up	2

Table 3: Default network parameters in the simulations

The simulation results are collected over a window of 10000 cycles, and the simulator is warmed-up for 3 sample windows of same size each before result collection starts. Booksim considers a network as saturated when the average latency for the sample period crosses over 500 cycles. We note the last injection rate before saturation happens, and report it as the network throughput. In Booksim, injection rate (offered load) is specified as packets per cycle (per node). So an injection rate (offered load) of 0.1 means a node can generate 1 packet on average in 10 cycles. Throughput is also measured in unit of packets per cycle per node. For each synthetic traffic pattern, we simulate with a sufficient number of injection rates to infer the latency curve.

4.1.3 Traffic Patterns. Five different types of traffic patterns are used in the evaluation: uniform random traffic, adversarial shift traffic, random permutation traffic, a mix of uniform random and adversarial traffic in the space domain, and a mix of uniform random and adversarial traffic in the time domain. With uniform random traffic (UR), the probability of sending a packet to each destination is equal. For adversarial traffic (ADV), all nodes connecting to a router i in a group send to all nodes connecting to router i in another group to stress the global links connecting the two groups: the $shift(k, 0)$ pattern described in Section 3.3.1. In a random permutation pattern, the compute nodes perform a randomly generated permutation pattern: each node sending to and receiving from at most one destination in the pattern. The space-based mixed traffic is generated by combining UR and ADV traffic patterns. We will use the notation MIXED(UR%, ADV%) to represent the combined traffic patterns, where UR% of nodes generate UR traffic and ADV% of nodes generate ADV traffic. For example, in MIXED(25,75), 25% of processing nodes are randomly selected to perform uniform traffic while the rest 75% perform adversarial shift traffic. The last type of traffic is the time-based mixed traffic pattern. We will use the notation TMIXED(UR%, ADV%) to represent such a traffic pattern. In TMIXED(UR%, ADV%), each packet from every node has an UR% probability to have a uniform random destination and an ADV% probability to have an adversarial destination.

4.2 T-VLB for Different Topologies

Figure 4 shows the average modeled throughput for $dfly(4, 8, 4, 9)$ with different configurations probed in the Step 1 estimation for this topology. The error bar in the figure is the standard error of the mean. As can be seen from the figure, that best throughput of 0.58 for this topology is achieved at 60% 5-hop: all VLB paths that are 4-hop or less and 60% of 5-hop VLB paths. The throughput of 0.58 means that each node can communicate at 58% of its link speed when the network saturates. This throughput is higher than the throughput of 0.56 with the conventional UGAL when all VLB paths are used. With 4 global links between each pair of groups, sufficient path diversity is provided by short VLB paths; and not all VLB paths are needed to achieve the best performance for the most demanding adversarial traffic patterns. Four candidate configurations are considered in Step 2: 40% 5-hop, 50% 5-hop, 60% 5-hop and 70% 5-hop, all having very similar average modeled throughput. The final T-VLB's for T-UGAL-L, T-PAR, and T-UGAL-G are the strategic choice with all 2-hop MIN paths followed by all 3-hop MIN paths with load-balance adjustment (removing some paths). Same results are obtained for $dfly(4, 8, 4, 17)$.

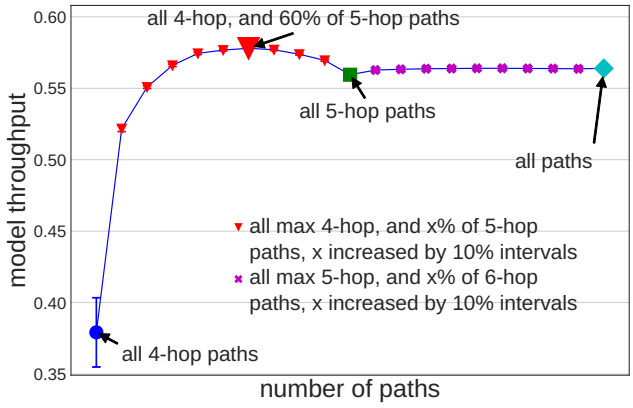

Figure 4: Average modeled throughput in Step 1 calculation for $dfly(4, 8, 4, 9)$

Figure 5 shows the average modeled throughput for $dfly(4, 8, 4, 33)$ with different configurations that are probed in the Step 1 estimation. As can be seen from the figure, the best performance for this topology is achieved when all VLB paths are used. Simulation results confirm that using any subset of VLB paths for this topology degrades the performance for the adversarial traffic patterns: T-UGAL converges with UGAL for this topology. In this topology,

there is only one link between each pair of groups. As such, all VLB paths are necessary to achieve high performance for the adversarial patterns. Note that maximum-sized Dragonfly topologies like $dfly(4, 8, 4, 33)$ with 1 link per pair of groups have been used in the design and evaluation of UGAL for dragonfly [1, 7].

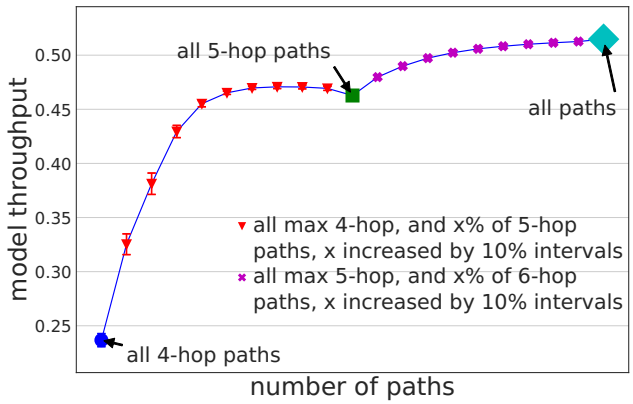

Figure 5: Average modeled throughput in Step 1 calculation for $dfly(4, 8, 4, 33)$

4.3 Simulation Results

Figure 6 shows the latency as the offered load increases for UGAL-L, T-UGAL-L, PAR, and T-PAR on $dfly(4, 8, 4, 9)$ with the adversarial traffic (shift(2, 0) pattern). The x-axis is the offered load in the unit of packets per cycle per node while the y-axis is the latency in the unit of cycles. As can be seen from the figure, T-UGAL-L improves over UGAL-L in latency when the network is not saturated and has a much higher saturation throughput. Specifically, when the offered load is 0.1, the average packet latency is 52.1 cycles for T-UGAL-L, 9.2% lower than the 56.9 cycles latency for UGAL-L. For this pattern, the saturation throughput of T-UGAL-L is 0.29, 26.1% higher than the 0.23 saturation throughput of UGAL-L. The results for PAR are similar. When the offered load is 0.2, the average packet latency for T-PAR is 59.9 cycles, 12.9% lower than the 67.6 cycles average packet latency for PAR. The saturation throughput of T-PAR is 0.38, 31.0% higher than the 0.29 saturation throughput of PAR. Figure 7 shows results for UGAL-G. T-UGAL-G improves the latency when the network is not saturated: at 0.1 offered load, the average latency for T-UGAL-G is 54.2, 12.9% lower than the 61.2 average latency for UGAL-G. The saturation throughput for T-UGAL-G is 0.3, 30% higher than the 0.23 saturation throughput for UGAL-G. For all UGAL-L, UGAL-G, and PAR, using our topology-custom scheme, significant improvements have been observed for the adversarial traffic pattern. We note that UGAL-G performs worse than PAR for this pattern: this is due to the fact that UGAL-G makes the decision at the source node. Due to the (large) link latency, as the packet moves through the network, the network information used by UGAL-G to make the routing decision becomes inaccurate. On the other hand, PAR can update the routing decision in the next hop, which results in higher performance.

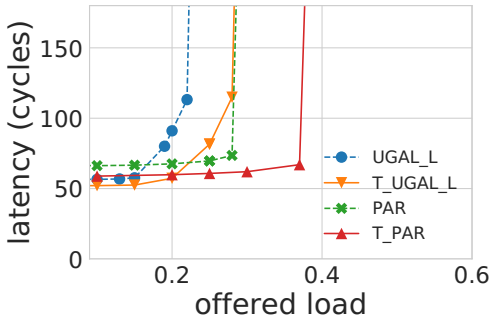

Figure 6: Latency for the adversarial shift(2, 0) pattern for UGAL-L and PAR on $dfly(4, 8, 4, 9)$

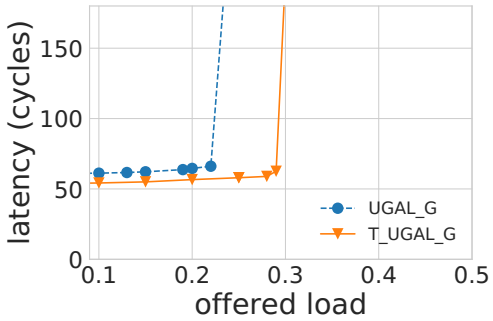

Figure 7: Latency for the adversarial shift(2, 0) pattern for UGAL-G on $dfly(4, 8, 4, 9)$

Figure 8 shows the latency as the offered load increases for UGAL-L, T-UGAL-L, PAR, and T-PAR on $dfly(4, 8, 4, 9)$ for a random permutation pattern. The results for permutation are somewhat similar to those for the adversarial traffic with less improvement. T-UGAL-L improves over UGAL-L in latency when the network is not saturated and has a higher saturation throughput. For example, when the offered load is 0.3, the average packet latency for T-UGAL-L is 43.7, 2.1% lower than the 44.6 cycles with UGAL-L. The saturation throughput of T-UGAL-L is 0.68, 7.9% higher than the 0.63 saturation throughput of UGAL-L. The reason that less improvement is observed in this experiment in comparison to the results for adversarial traffic is that a smaller percentage of packets are routed using VLB paths in the permutation pattern. Figure 9 shows results for UGAL-G. In this case, T-UGAL-G has similar average packet latency when the network is under low load. However, the saturation throughput for T-UGAL-G, 0.66, is 11.9% higher that the 0.59 saturation throughput for UGAL-G. This is due to the use of shorter paths that reduces the overall network load and improves the saturation throughput. Even with the precise information, UGAL-G is forced to use longer paths since it randomly selects one VLB path for consideration for each packet.

Figure 10 and Figure 11 show the results for MIXED(75, 25) and MIXED(25, 75) on $dfly(4, 8, 4, 17)$, respectively. T-UGAL only optimizes VLB paths. As such, its advantage can only be observed when more traffic are routed using VLB paths. This trend can be

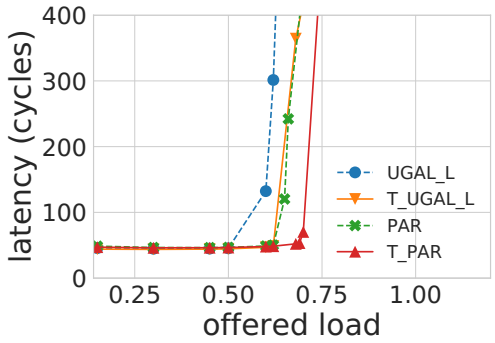

Figure 8: Latency for a random permutation pattern for UGAL-L and PAR on $dfly(4, 8, 4, 9)$

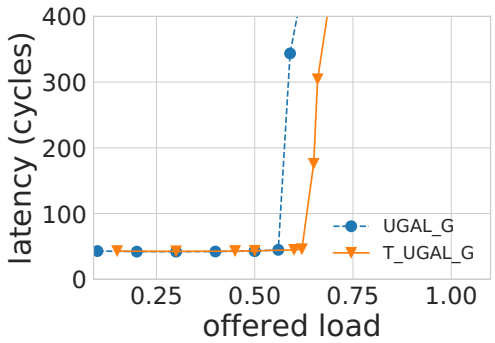

Figure 9: Latency for a random permutation pattern for UGAL-G on $dfly(4, 8, 4, 9)$

observed in the two figures: as the traffic becomes more adversarial (MIXED(25,75)), the saturation throughput decreases for all schemes, but the advantage of T-UGAL-L and T-PAR becomes larger. For example, for MIXED(75, 25), the saturation throughput for T-PAR is 0.46, 15% higher than the saturation throughput 0.40 for PAR; for MIXED(25, 75), the saturation throughput for T-PAR is 0.30, 20% higher than the saturation throughput 0.25 for PAR. The advantage of T-UGAL over UGAL is also observed for time-based mixed traffic as shown in Figure 12.

Figure 13 shows the latency as the offered load increases for UGAL-L, T-UGAL-L, PAR, T-PAR, UGAL-G, and T-UGAL-G, on a larger Dragonfly topology $dfly(13, 26, 13, 27)$ for an adversarial traffic pattern (shift(1, 0) pattern). While the specific numbers differ, the trend is very similar to that for the smaller topologies $dfly(4, 8, 4, 9)$ and $dfly(4, 8, 4, 17)$. T-UGAL-L has significant improvement over UGAL-L while T-PAR have significant improvement over PAR, at both low and high loads. Figure 14 shows the results for a mixed traffic (MIXED(50, 50)). Again, T-UGAL variations have clear advantage over their corresponding UGAL variations. The results for other traffic patterns on $dfly(13, 26, 13, 27)$ have a similar trend as the results on smaller $dfly(4, 8, 4, 9)$ and $dfly(4, 8, 4, 17)$: T-UGAL offers advantages over the corresponding UGAL for Dragonfly topologies of different sizes and shapes.

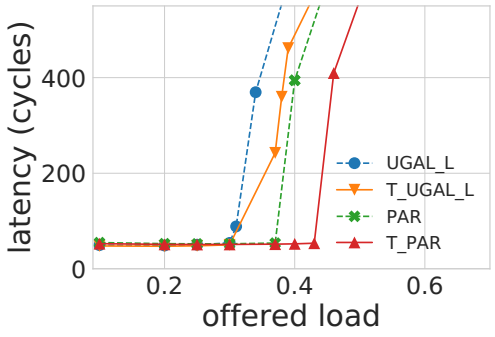

Figure 10: Mixed traffic: MIXED(75, 25) with UGAL-L and PAR on $dfly(4, 8, 4, 17)$

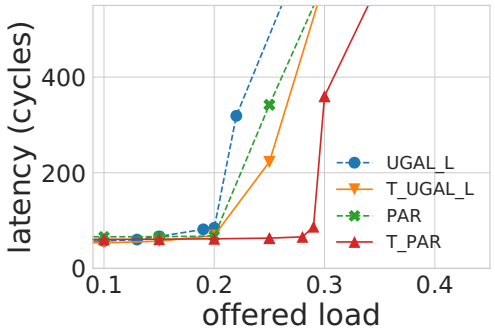

Figure 11: Mixed traffic: MIXED(25, 75) with UGAL-L and PAR on $dfly(4, 8, 4, 17)$

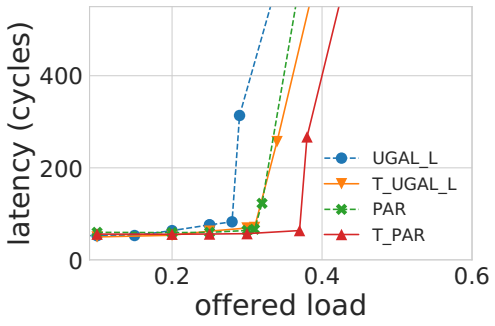

Figure 12: Time-based mixed traffic: TMIXED(50, 50) with UGAL-L and PAR on $dfly(4, 8, 4, 17)$

Figures 15, 16, 17, and 18 show the sensitivity of UGAL and T-UGAL to different network parameters. For these experiments, we alter one network parameter and keep the remaining parameters as the default (Table 3). Figure 15 shows the sensitivity to the link latency. The legend format for this figure is $routing(local\_link\_latency, global\_link\_latency)$. For example, UGAL_G(40, 60) denotes UGAL_G with local link latency of 40 cycles and global link latency of 60 cycles. Figure 16 shows the sensitivity to the buffer length. The legend format for this figure is $routing(buffer\_length)$.

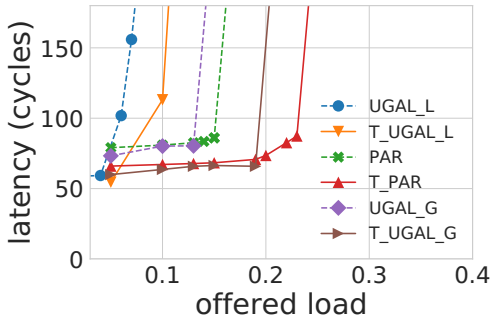

Figure 13: Adversarial traffic (shift(1, 0) pattern) for UGAL-L, PAR, and UGAL-G on $dfly(13, 26, 13, 27)$

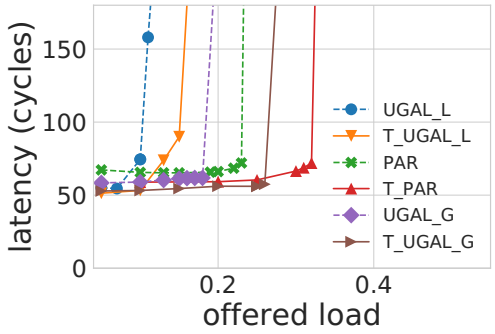

Figure 14: Mixed traffic: MIXED(50, 50) for UGAL-L, PAR, and UGAL-G on $dfly(13, 26, 13, 27)$

For example, UGAL_L(8) denotes UGAL_L with buffer size of 8 flits. Figure 17 shows the sensitivity to the switch speedup. The legend format for this figure is *routing*(*speedup*). PAR(1) denotes PAR with switch speedup of 1. Finally, Figure 18 shows the sensitivity to the virtual channel allocation scheme. The legend format for this figure is *routing*(4) or *routing*(6). *routing*(4) denotes the network with the virtual channel allocation scheme in [11] to avoid deadlock where UGAL_G takes 4 virtual channels. *routing*(6) denotes the network with a simple deadlock avoidance virtual channel scheme where each packet is sent on a new virtual channel every hop. As can be seen from the figures, all of these parameters can have significant impacts on packet latency and saturation throughput. For example, T_UGAL(4) performs worse than T_UGAL(6) in Figure 18 due to the smaller total number of buffers per link and head-of-line blocking. However, a common observation on all of these experiments is that a T-UGAL variation consistently and substantially out-performs its UGAL counterpart. This has been observed in all of the experiments that we have performed with different topologies and network parameters, which highlights the performance advantage of our proposed T-UGAL.

In summary, for all three variations of UGAL: UGAL-L, PAR, and UGAL-G, T-UGAL has a clear advantage when the adversarial traffic components are present in the network. By using a subset of shorter VLB paths, computed based on the network topology, T-UGAL reduces the packet latency when the network is not

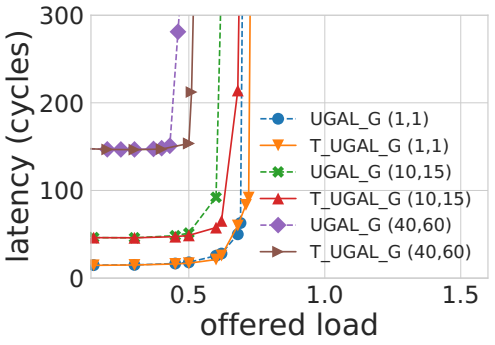

Figure 15: Effect of varying link latency on UGAL-G on $dfly(4, 8, 4, 17)$ for random permutation pattern

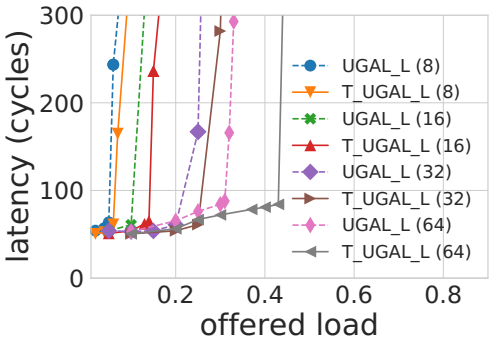

Figure 16: Effect of varying buffer length on UGAL-L on $dfly(4, 8, 4, 17)$ for MIXED(50,50) pattern

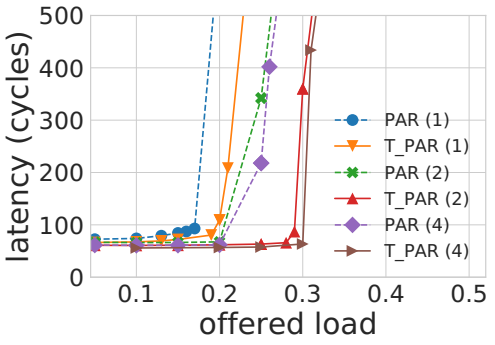

Figure 17: Effect of varying router internal speedup on PAR on $dfly(4, 8, 4, 17)$ for MIXED(25,75) pattern

saturated while improving the saturation throughput. Moreover, T-UGAL has more advantage with practical UGAL schemes such as UGAL-L and PAR than with the idealistic UGAL-G since using shorter VLB paths also helps in the estimations of path latency (such estimation is used to make routing decisions). As shown in the experiments, depending on the traffic condition, the improvement can be very significant.

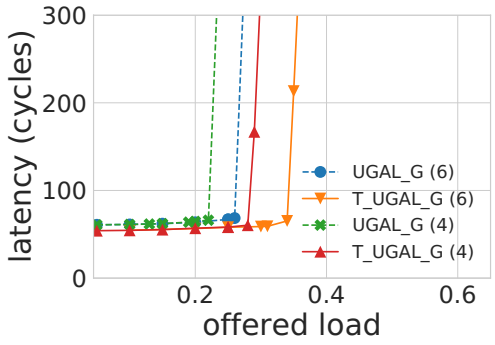

Figure 18: Effect of different virtual channel allocation schemes on UGAL-G on $dfly(4, 8, 4, 9)$ for the adversarial shift(1,0) pattern

5 CONCLUSION

We propose topology-custom UGAL routing (T-UGAL) for Dragonfly topologies. We show that by using a subset of VLB paths with shorter average path length (than the average path length of all VLB paths), T-UGAL improves over the conventional UGAL routing very significantly on many topologies, especially the ones with a small number of groups and a large number of links between each pair of groups which is common in many practical systems. We develop a general scheme that can be applied to any Dragonfly topology to obtain T-UGAL. Our scheme is orthogonal to other techniques for improving UGAL performance and can be used alone or combined with other UGAL-enhancement techniques. All the analysis and selection of candidate VLB paths happen during network designing, and the set of eligible paths does not need to change unless the network topology is altered; so it does not add any extra computational burden over existing routers.

ACKNOWLEDGMENTS

This work is supported in part by Los Alamos National Laboratory under subcontract 497357. The material is also based upon work supported by the National Science Foundation under Grants CICI-1738912 and CRI-1822737. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. This work used the XSEDE Bridges resource at the Pittsburgh Supercomputing Center (PSC) through allocation TG-ECS190004.

REFERENCES

[1] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 77–88, Washington, DC, USA, 2008. IEEE Computer Society.

[2] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. Cray cascade: A scalable hpc system based on a dragonfly network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 103:1–103:9, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[3] Oak Ridge National Laboratory. Introducing Titan—the world's #1 open science supercomputer. https://www.olcf.ornl.gov/titan/, 2012.

[4] Billy J. Archer and Manuel Vigil. The Trinity system. In *Nuclear Explosive Code Development Conference (NECDC)*, Los Alamos, New Mexico, October 20–24, 2014. Also appears as Los Alamos Technical Report LA-UR-15-20221.

[5] Cray Inc. Slingshot: the interconnect for exascale computing. White paper, Feburary 2019. available at https://www.cray.com/sites/default/files/Slingshot-The-Interconnect-for-the-Exascale-Era.pdf.

[6] Arjun Singh. *Load-Balanced Routing In Interconnection Networks*. PhD thesis, Stanford University, 2005.

[7] Nan Jiang, John Kim, and William J. Dally. Indirect adaptive routing on large scale interconnection networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 220–231, New York, NY, USA, 2009. ACM.

[8] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, G. Rodríguez, J. Labarta, and C. Minkenberg. On-the-fly adaptive routing in high-radix hierarchical networks. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 279–288, Sept 2012.

[9] M. Garcia, E. Vallejo, R. Beivide, M. Valero, and G. Rodríguez. OFAR-CM: Efficient Dragonfly networks with simple congestion management. In *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on*, pages 55–62, Aug 2013.

[10] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero. Efficient routing mechanisms for Dragonfly networks. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 582–592, Oct 2013.

[11] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott. Overcoming far-end congestion in large-scale networks. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 415–427, Feb 2015.

[12] P. Fuentes, E. Vallejo, M. Garcia, R. Beivide, G. Rodríguez, C. Minkenberg, and M. Valero. Contention-based nonminimal adaptive routing in high-radix networks. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 103–112, May 2015.

[13] Peyman Faizian, Juan Francisco Alfaro, Md Shafayat Rahman, Md Atiqul Mollah, Xin Yuan, Scott Pakin, and Michael Lang. TPR: traffic pattern-based adaptive routing for dragonfly networks. *IEEE Trans. Multi-Scale Computing Systems*, 4(4):931–943, 2018.

[14] E. Hastings, D. Rincon-Cruz, M. Spehlmann, S. Meyers, A. Xu, D. P. Bunde, and V. J. Leung. Comparing global link arrangements for dragonfly networks. In *2015 IEEE International Conference on Cluster Computing*, pages 361–370, Sep. 2015.

[15] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[16] Md Atiqul Mollah, Peyman Faizian, Md Shafayat Rahman, Xin Yuan, Scott Pakin, and Michael Lang. Modeling ugal on the dragonfly topology. In *International Workshop on Performance Modeling, Benchmarking, and Simulation on High Performance Computer Systems (PMBS'17)*, pages 136–157, 01 2017.

[17] Md Shafayat Rahman, Md Atiqul Mollah, Peyman Faizian, and Xin Yuan. Load-balanced slim fly networks. In *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, pages 41:1–41:10, New York, NY, USA, 2018. ACM.

[18] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 86–96, April 2013.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Step 1: We generated Linear Programming models for a number of Dragonfly network topologies. The models were generated as described in "Modeling Ugal on the Dragonfly Topology" by Mollah et al. We made some further modifications on the models to increase their accuracy. To be specific, we added the constraint that for a source-destination pair, a longer VLB path will never have a larger data rate allocated to it than a shorter VLB path between the same SD pair.

We wrote a tool using Python 3.6.4 which creates the topology, generates minimal and non-minimal paths, generates the linear programming constraints, runs a linear programming solver to solve the equations, and collects results.

Our implementation of the models can be found in the included git repository.

IBM CPLEX optimizer was used as the linear-programming solver.

Step 2: We used open-source Booksim 2.0 to simulate the selected Dragonfly networks and analyze their performance. For Booksim, we extended the code and implemented variations of Dragonfly topology, routing functions and traffic patterns.

The extension code we wrote for Booksim is also shared in our git repository.

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: Some author-created data artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

List of URLs and/or DOIs where artifacts are available:

```
https://github.com/YashfatHarman/SC\_2019\_TUGAL\_Dragonfly\_source\_code.git
https://github.com/booksim/booksim2
https://www.ibm.com/analytics/cplex-optimizer
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Operating systems and versions: Ubuntu 16.04.5, CentOS Linux release 7.4.1708

Compilers and versions: g+= 5.4.0, python 3.6.4

Applications and versions: Booksim 2.0, IBM CPLEX optimizer

Paper Modifications: As we mention in the previous section, we made some changes in Booksim. Please check our git repository to see the relevant modifications.

Output from scripts that gathers execution environment information.

```
XDG_SESSION_ID=309
HOSTNAME=draco4.cs.fsu.edu
SELINUX_ROLE_REQUESTED=
SHELL=/bin/bash
TERM=xterm-256color
HISTSIZE=1000
SSH_CLIENT=192.168.123.12 60174 22
SELINUX_USE_CURRENT_RANGE=
SSH_TTY=/dev/pts/0
USER=USER
LS_COLORS=rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15:pi=40;38;5;11:so=38;5;13:do=38;5;5:bd=48;5;232;38;5;11:cd=48;5;232;38;5;3:or=48;5;232;38;5;9:mi=0;48;5;232;38;5;15:su=48;5;196;38;5;15:sg=48;5;1;38;5;16:ca=48;5;196;38;5;226:tw=48;5;10;38;5;16:ow=48;5;10;38;5;21:st=48;5;21;38;5;15:ex=38;5;34:*.tar=38;5;9:*.tgz=38;5;9:*.arc=38;5;9:*.arj=38;5;9:*.taz=38;5;9:*.lha=38;5;9:*.lz4=38;5;9:*.lzh=38;5;9:*.lzma=38;5;9:*.tlz=38;5;9:*.txz=38;5;9:*.tzo=38;5;9:*.t7z=38;5;9:*.zip=38;5;9:*.z=38;5;9:*.Z=38;5;9:*.dz=38;5;9:*.gz=38;5;9:*.lrz=38;5;9:*.lz=38;5;9:*.lzo=38;5;9:*.xz=38;5;9:*.bz2=38;5;9:*.bz=38;5;9:*.tbz=38;5;9:*.tbz2=38;5;9:*.tz=38;5;9:*.deb=38;5;9:*.rpm=38;5;9:*.jar=38;5;9:*.war=38;5;9:*.ear=38;5;9:*.sar=38;5;9:*.rar=38;5;9:*.alz=38;5;9:*.ace=38;5;9:*.zoo=38;5;9:*.cpio=38;5;9:*.7z=38;5;9:*.rz=38;5;9:*.cab=38;5;9:*.jpg=38;5;13:*.jpeg=38;5;13:*.gif=38;5;13:*.bmp=38;5;13:*.pbm=38;5;13:*.pgm=38;5;13:*.ppm=38;5;13:*.tga=38;5;13:*.xbm=38;5;13:*.xpm=38;5;13:*.tif=38;5;13:*.tiff=38;5;13:*.png=38;5;13:*.svg=38;5;13:*.svgz=38;5;13:*.mng=38;5;13:*.pcx=38;5;13:*.mov=38;5;13:*.mpg=38;5;13:*.mpeg=38;5;13:*.m2v=38;5;13:*.mkv=38;5;13:*.webm=38;5;13:*.ogm=38;5;13:*.mp4=38;5;13:*.m4v=38;5;13:*.mp4v=38;5;13:*.vob=38;5;13:*.qt=38;5;13:*.nuv=38;5;13:*.wmv=38;5;13:*.asf=38;5;13:*.rm=38;5;13:*.rmvb=38;5;13:*.flc=38;5;13:*.avi=38;5;13:*.fli=38;5;13:*.flv=38;5;13:*.gl=38;5;13:*.dl=38;5;13:*.xcf=38;5;13:*.xwd=38;5;13:*.yuv=38;5;13:*.cgm=38;5;13:*.emf=38;5;13:*.axv=38;5;13:*.anx=38;5;13:*.ogv=38;5;13:*.ogx=38;5;13:*.aac=38;5;45:*.au=38;5;45:*.flac=38;5;45:*.mid=38;5;45:*.midi=38;5;45:*.mka=38;5;45:*.mp3=38;5;45:*.mpc=38;5;45:*.ogg=38;5;45:*.ra=38;5;45:*.wav=38;5;45:*.axa=38;5;45:*.oga=38;5;45:*.spx=38;5;45:*.xspf=38;5;45:
```

```
PATH=/home/USER/anaconda3/bin:/usr/local/bin:/usr/bi⌋
↪  n:/usr/local/sbin:/usr/sbin:/home/USER/.local/bi⌋
↪  n:/home/USER/bin
MAIL=/var/spool/mail/USER
_=/usr/bin/env
PWD=/home/USER/Author-Kit
LANG=en_US.UTF-8
MODULEPATH=/usr/share/Modules/modulefiles:/etc/modul⌋
↪  efiles
LOADEDMODULES=
SELINUX_LEVEL_REQUESTED=
HISTCONTROL=ignoredups
HOME=/home/USER
SHLVL=2
LOGNAME=USER
SSH_CONNECTION=192.168.123.12 60174 192.168.122.154
↪  22
MODULESHOME=/usr/share/Modules
LESSOPEN=||/usr/bin/lesspipe.sh %s
XDG_RUNTIME_DIR=/run/user/1001
BASH_FUNC_module()=() {  eval `/usr/bin/modulecmd
↪  bash $*`
}
+ lsb_release -a
./collect_environment.sh: line 10: lsb_release:
↪  command not found
+ uname -a
Linux draco4.cs.fsu.edu 3.10.0-693.11.6.el7.x86_64 #1
↪  SMP Thu Jan 4 01:06:37 UTC 2018 x86_64 x86_64
↪  x86_64 GNU/Linux
+ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 94
Model name:            Intel(R) Xeon(R) CPU E3-1220 v5
↪  @ 3.00GHz
Stepping:              3
CPU MHz:               900.234
CPU max MHz:           3500.0000
CPU min MHz:           800.0000
BogoMIPS:              6000.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              8192K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce
↪  cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪  dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪  pdpe1gb rdtscp lm constant_tsc art arch_perfmon
↪  pebs bts rep_good nopl xtopology nonstop_tsc
↪  aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
↪  ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm
↪  pcid sse4_1 sse4_2 x2apic movbe popcnt
↪  tsc_deadline_timer aes xsave avx f16c rdrand
↪  lahf_lm abm 3dnowprefetch epb invpcid_single
↪  intel_pt spec_ctrl ibpb_support tpr_shadow vnmi
↪  flexpriority ept vpid fsgsbase tsc_adjust bmi1
↪  hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx
↪  smap clflushopt xsaveopt xsavec xgetbv1 dtherm ida
↪  arat pln pts hwp hwp_notify hwp_act_window hwp_epp
+ cat /proc/meminfo
MemTotal:       65767632 kB
MemFree:        64013104 kB
MemAvailable:   64689900 kB
Buffers:           70816 kB
Cached:           794380 kB
SwapCached:            0 kB
Active:           506368 kB
Inactive:         420000 kB
Active(anon):      61448 kB
Inactive(anon):    16504 kB
Active(file):     444920 kB
Inactive(file):   403496 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:      16776188 kB
SwapFree:       16776188 kB
Dirty:                 8 kB
Writeback:             0 kB
AnonPages:         61252 kB
Mapped:            27016 kB
Shmem:             16788 kB
Slab:             414520 kB
SReclaimable:     363376 kB
SUnreclaim:        51144 kB
KernelStack:        2032 kB
PageTables:         3568 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    49660004 kB
Committed_AS:     253160 kB
VmallocTotal:   34359738367 kB
VmallocUsed:      384136 kB
VmallocChunk:   34358947836 kB
HardwareCorrupted:     0 kB
AnonHugePages:      4096 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

```
DirectMap4k:      115672 kB
DirectMap2M:     4036608 kB
DirectMap1G:    62914560 kB
+ inxi -F -c0
./collect_environment.sh: line 14: inxi: command not
↪  found
+ lsblk -a
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 465.8G  0 disk
├─sda1   8:1    0  39.2M  0 part
├─sda2   8:2    0     2G  0 part
├─sda3   8:3    0   500M  0 part /boot
├─sda4   8:4    0     1K  0 part
├─sda5   8:5    0 347.2G  0 part /home
├─sda6   8:6    0   100G  0 part /
└─sda7   8:7    0    16G  0 part [SWAP]
sr0     11:0    1  1024M  0 rom
+ lsscsi -s
[0:0:0:0]    disk    ATA      WDC WD5000AZLX-7 1A02
↪  /dev/sda    500GB
[4:0:0:0]    cd/dvd  HL-DT-ST DVD-ROM DU90N    D3C1
↪  /dev/sr0        -
+ module list
++ /usr/bin/modulecmd bash list
No Modulefiles Currently Loaded.
+ eval
+ nvidia-smi
./collect_environment.sh: line 18: nvidia-smi:
↪  command not found
+ lshw -short -quiet -sanitize
+ cat
./collect_environment.sh: line 19: lshw: command not
↪  found
+ lspci
00:00.0 Host bridge: Intel Corporation Skylake Host
↪  Bridge/DRAM Registers (rev 07)
00:01.0 PCI bridge: Intel Corporation Skylake PCIe
↪  Controller (x16) (rev 07)
00:01.1 PCI bridge: Intel Corporation Skylake PCIe
↪  Controller (x8) (rev 07)
00:14.0 USB controller: Intel Corporation Sunrise
↪  Point-H USB 3.0 xHCI Controller (rev 31)
00:14.2 Signal processing controller: Intel
↪  Corporation Sunrise Point-H Thermal subsystem
↪  (rev 31)
00:16.0 Communication controller: Intel Corporation
↪  Sunrise Point-H CSME HECI #1 (rev 31)
00:16.1 Communication controller: Intel Corporation
↪  Sunrise Point-H CSME HECI #2 (rev 31)
00:17.0 SATA controller: Intel Corporation Sunrise
↪  Point-H SATA controller [AHCI mode] (rev 31)
00:1d.0 PCI bridge: Intel Corporation Sunrise Point-H
↪  PCI Express Root Port #9 (rev f1)
00:1d.2 PCI bridge: Intel Corporation Sunrise Point-H
↪  PCI Express Root Port #11 (rev f1)
00:1f.0 ISA bridge: Intel Corporation Sunrise Point-H
↪  LPC Controller (rev 31)
00:1f.2 Memory controller: Intel Corporation Sunrise
↪  Point-H PMC (rev 31)
00:1f.4 SMBus: Intel Corporation Sunrise Point-H
↪  SMBus (rev 31)
03:00.0 Ethernet controller: Broadcom Limited
↪  NetXtreme BCM5720 Gigabit Ethernet PCIe
03:00.1 Ethernet controller: Broadcom Limited
↪  NetXtreme BCM5720 Gigabit Ethernet PCIe
04:00.0 PCI bridge: Renesas Technology Corp. SH7758
↪  PCIe Switch [PS]
05:00.0 PCI bridge: Renesas Technology Corp. SH7758
↪  PCIe Switch [PS]
06:00.0 PCI bridge: Renesas Technology Corp. SH7758
↪  PCIe-PCI Bridge [PPB]
07:00.0 VGA compatible controller: Matrox Electronics
↪  Systems Ltd. G200eR2 (rev 01)
```

ARTIFACT EVALUATION

Verification and validation studies: 1. For linear programming models, we followed the directions and reproduced some of the result graphs published in "Modeling Ugal on the Dragonfly Topology" by Mollah et al. The results were close enough to ensure that we were following their directions properly.

2. For Booksim simulation, we reproduced some of the results in "Technology-Driven, Highly-Scalable Dragonfly Topology" by Kim et al. This way we ensured that our implementation of regular Dragonfly network and UGAL routing is correct.

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: Booksim 2.0 is widely used and most of the studies using Booksim mentions the simulator settings. We ensured that we are following the standard practice. We performed extensive tests with various Booksim parameters to ensure that our results do not depend on the parameters, including but not limited to buffer size, virtual channel count, link latency, simulator speedup etc. Changing these values will change the absolute output values, but the relative performance of the existing schemes and our proposed scheme will remain same.

Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system. For each modeling and simulation run that depends on a random seed for routing/path generation/traffic generation, we performed multiple runs (8 to 20, depending on experiment) using different random seeds. Final results were presented as average of the runs, with standard error of mean, where applicable.