2018

Design and Evaluation of Networking Techniques for the Next Generation of Interconnection Networks

Peyman Faizian

FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

DESIGN AND EVALUATION OF NETWORKING TECHNIQUES FOR THE NEXT GENERATION OF INTERCONNECTION NETWORKS

By

PEYMAN FAIZIAN

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2018

Peyman Faizian defended this dissertation on April 18, 2018.
The members of the supervisory committee were:

Xin Yuan

Professor Directing Dissertation

Fengfeng Ke

University Representative

Ashok Srinivasan

Committee Member

Gary Tyson

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Numbers 0000219853 and DE-SC0016039.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

High performance computing(HPC) and data center systems have undergone rapid growth in recent years. To meet the current and future demand of compute- and data-intensive applications, these systems require the integration of a large number of processors, storage and I/O devices through high-speed interconnection networks. In massively scaled HPC and data centers, the performance of the interconnect is a major defining factor for the performance of the entire system. Interconnect performance depends on a variety of factors including but not limited to topological characteristics, routing schemes, resource management techniques and technological constraints.

In this dissertation, I explore several approaches to improve the performance of large-scale networks. First, I investigate the topological properties of a network and their effect on the performance of the system under different workloads. Based on detailed analysis of graph structures, I find a well-known graph as a potential topology of choice for the next generation of large-scale networks.

Second, I study the behavior of adaptive routing on the current generation of supercomputers based on the Dragonfly topology and highlight the fact that the performance of adaptive routing on such networks can be enhanced by using detailed information about the communication pattern. I develop a novel approach for identifying the traffic pattern and then use this information to improve the performance of adaptive routing on dragonfly networks.

Finally, I investigate the possible advantages of utilizing emerging software defined networking technology in the high performance computing domain. My findings show that by leveraging the use of SDN, we can achieve near-optimal rate allocation for communication patterns in an HPC cluster, which can remove the necessity for expensive adaptive routing schemes and simplify the control plane on the next generation of supercomputers.

CHAPTER 1

INTRODUCTION

The current generation of supercomputers contain up to 40k processing nodes and reach a computing power up to 93 petaflops. By considering this as a rough guide, we can expect the exascale machines in a few years to require 100k-200k processing nodes. The same trend can be seen among data centers. At this scale, interconnection networks are a critical component of modern computer systems and significantly impact the overall performance and cost of the system. Designing such complex systems is a challenging task and spans over various domains of research in scope.

One of the main contributing factors in performance and cost of an interconnection network is the network topology: which is the way network elements are arranged and connected. The topology determines the most important charcteristics of an interconnection network including capacity, diameter, number of network elements and etc. On the other hand, it is the routing scheme which defines how network resources are used to route packets throughout the network. A good routing algorithm can distribute packets among the available paths while avoiding bottlenecks, minimzing latency and maximizing throughput under different traffic conditions. Therefore, designing and improving routing schemes also plays an important role in achieving high performance networks. Another important factor in designing an interconnection network is utilizing new networking technologies to overcome existing challenges. Traditionally, networks have been designed around the distributed control concept, with deterministic or adaptive routing methods. However, *Software Defined Networking*(SDN) [39] has been recently proposed, which can leverage simpler and more flexible control over networks by decoupling data and control plane functionalities. This can potentially alleviate the inevitable complexity of large-scale networks and lead to simpler interconnect network designs. However, it may also require revisiting some of the well-known problems in the field including routing and traffic engineering techniques.

The main goal of my PhD research is to explore different approaches that lead to more efficient interconnection networks, with a focus on (1) network topology design and evaluation, (2) network routing design and evaluation, and (3) exploring new networking technolgies. In each of these domains, I investigate and

evaluate the current networking techniques and propose new designs and schemes which can significantly improve the current and future interconnection networks.

For network topology, I have performed a comprehensive study [20] [18] on the performance bounds of the Jellyfish topology [43], which has recently been proposed for large-scale data center networks. I have also shown that *Generalized de Bruijn Graph* (GDBG) [9], a deterministic near-optimal directed regular graph, outperforms Jellyfish and any other topology in the family of *Directed Regular Graphs*(DRG) as its performance is very close to the optimal bounds. This result is significant, because DRGs are used in many interconnection network designs. Therefore, GDBG can be used as a benchmark to evaluate the performance of such networks.

My research on network routing has led to designing a new routing scheme for intra-group and inter-group communications on HPC and data center networks which use the Dragonfly topology [21]. This new routing scheme improves the existing adaptive routing methods on such networks by employing a novel technique to identify the traffic pattern at any time and to change the routing decisions accordingly, based on the inferred pattern information. Since adaptive routing is required to achieve acceptable performance on Dragonfly networks, the ability to distinguish different traffic patterns directly improves the observed run-time of applications on such networks, by decreasing routing latency.

As a part of my PhD research, I have also studied SDN-based infrastructures which have shown great promises and have been deployed in data centers and campus networks. Specifically, I have investigated several features of SDN networks including dynamic and flexible reconfiguration of network resources, failure resilience and scalability. I have also developed routing and load distribution techniques [19] based on SDN technology for Dragonfly networks such that by using the global view of the network and some HPC communication characteristics, these techniques can compete or even out-perform the current adaptive routing schemes on Dragonfly networks.

The rest of this document is structured as follows: In Chapter 2, I introduce the concepts used in later chapters such as network topologies, routing functions, modeling techniques and the SDN paradigm. Chapters 3 and 4 include my existing work on designing and evaluating topologies and routing schemes on large-scale interconnection networks. In Chapter 3, I study the performance characteristics of Jellyfish topology and compare it to GDBG. In Chapter 4, an adaptive routing scheme is proposed for Dragonfly networks, which can distinguish different traffic conditions to achieve higher throughput and lower latency in compare to available schemes. Finally, Chapter 5 includes my work in leveraging the use of SDN in HPC

environments to achieve better communication performance by using application communication patterns to calculate the near-optimal rate-allocation for flows in an interconnection network.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Topology and Routing

In this section I will introduce topologies and the respective routing schemes associated with each topology, which will be used in subsequent chapters.

2.1.1 Random Regular Graph (Jellyfish) Topology and Routing

Random Regular Graph or the Jellyfish topology is proposed recently by Singla et al. [43] as a flexible and high-capacity topology for large scale interconnects. Unlike the deterministically constructed networks currently deployed in HPC and data centers, switches in an RRG are interconnected randomly. In a network with an RRG topology, switches have the same line rate and use the same number of ports to connect to other switches. A network with an RRG topology is characterized by three parameters: the number of switches (N), the radix of each switch (x), and the number of ports used by each switch to connect to other switches (r). In this dissertation, we focus on switch-level topologies with parameters N and r, and denote the topology as $RRG(N,r)$. A sample RRG(x,y) is depicted in Figure 2.1. RRGs have been shown to have good topological properties such as small diameter [12] and high bisection bandwidth [11].

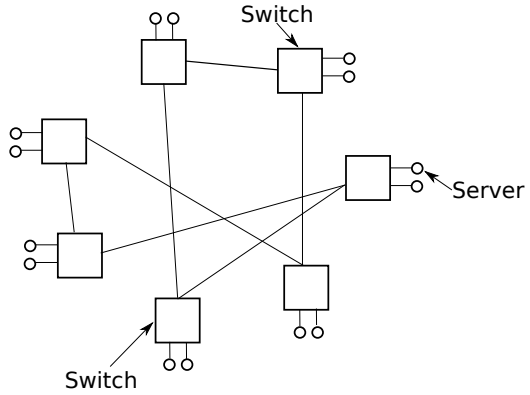


Figure 2.1: Jellyfish(Random Regular Graph) topology

***k*-Shortest Path Routing.** Singla et al. showed that shortest-path routing and standard Equal Cost Multi-Path routing can not sufficiently utilize the path diversity available in RRG topologies and in order to achieve that, longer-than-shortest paths need to be used [43]. Therefore, they propose using k-shortest path routing, in which for each flow, up to k shortest paths are used.

2.1.2 Dragonfly Topology and Routing

The dragonfly can be seen as a three-level network: router, group, and system as shown in Figure 2.2. At the first level, each router is connected to p processing nodes, $a-1$ routers in the same group and h routers in other groups. Hence the radix of each router is $k = p+a+h-1$. A group consists of a routers connected via an intra-group interconnection network formed from local channels. Each group has ap connections to processing nodes and ah connections to global channels, and all of the routers in a group collectively act as a virtual router with radix $k' = a(p+h)$. This very high radix, $k' >> k$ enables the system level network to be realized with very low global diameter (the maximum number of expensive global channels on the minimum path between any two nodes). Up to $g = ah+1$ groups ($N = ap(ah+1)$ processing nodes) can be connected with a global diameter of one. In contrast, a system-level network built directly with radix k routers would require a larger global diameter [33].

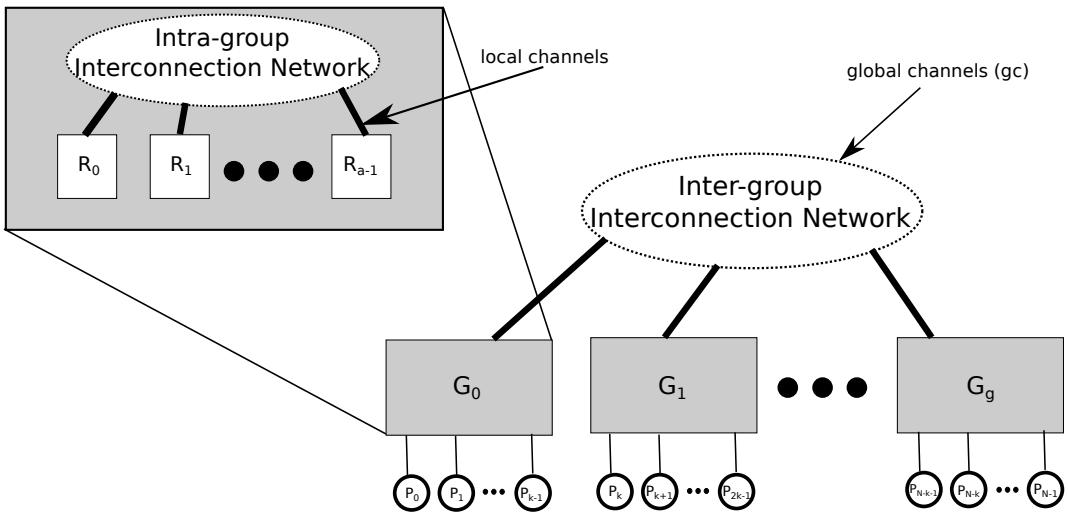


Figure 2.2: High-level block diagram of a dragonfly topology [33]

As seen in Figure 2.2, dragonfly can be considered a hierarchical topology with independent intra and inter-group interconnection networks. Although any arbitrary topology can be used for each of these networks, as mentioned above, we assume an all-to-all inter-group connection at system level, to maintain the

minimal global diameter. Figure 2.3a shows a dragonfly network with fully connected intra and inter-group networks. However, to achieve higher levels of scalability, higher-dimensional topologies can be used for intra-group networks. For example as shown in Figure 2.3b, by using 3-D flattened butterfly topology, we can increase the number of processing nodes in a group from 8 to 16 without changing the router radix.

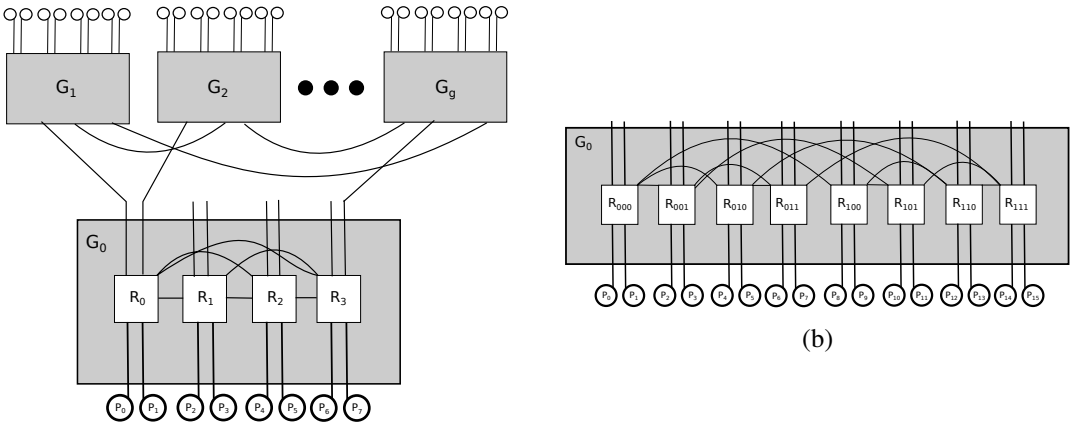


Figure 2.3: (a) A fully connected dragonfly topology with $N = 72$ (b) Using 3-D flattened butterfly for intra-group topology increases the group radix by a factor of 2 [33]

Cray Cascade. The Cray Cascade architecture employs a variation of Dragonfly as it's interconnect topology. The architecture has a fixed structure for each group, but allows variable numbers of groups to form a system. The intra-group topology is well-defined, but the total number of groups and inter-group bandwidth are flexible to allow expansion in the future.

Every group in Cascade is formed of a pair of cabinets. Each cabinet houses three chassis. Each chassis contains 16 blades. Each blade connects a single router and four processing elements. So in total, each group contains 6 chassis, where each have 16 blades with a total of 96 routers. The Cascade system uses Aries routers, which is an Application Specific Integrated Circuit (ASIC) developed by Cray. Each chassis backplane provides all-to-all connections among sixteen Aries routers. Each router is also connected to five other routers in the remaining five chassis within the same group using electrical cables. The inter-chassis connections are made with corresponding slots. For example, a router in slot 1 in one chassis will have direct links to the five slot 1 routers in the other five chassis within the same group. Each inter-chassis link is equivalent to three intra-chassis links in terms of bandwidth. An Aries router has a total of 48 ports: 8 ports for local processing nodes, 15 ports connecting to 15 routers in the same chassis, 15 ports to 5 routers

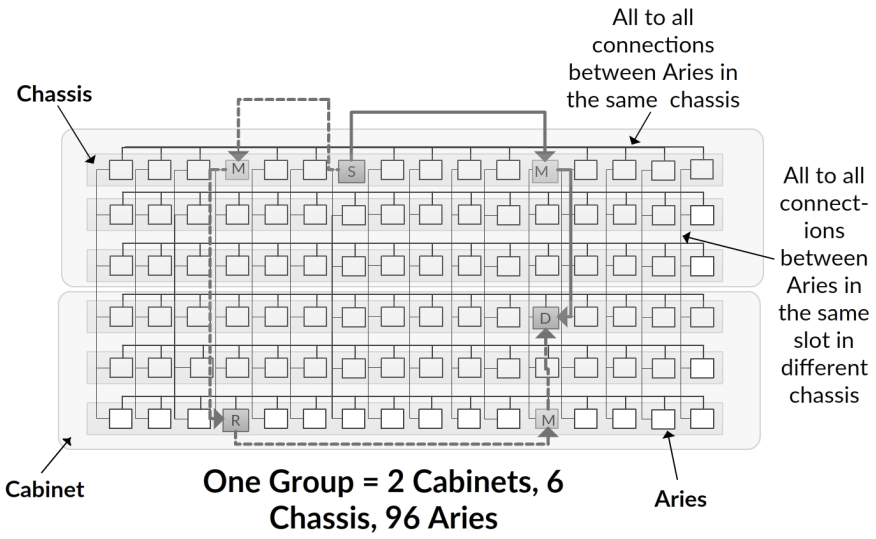


Figure 2.4: Cray Cascade intra-group topology

in the same slot but different chassis, and 10 ports to other groups. [17] Figure 2.4 shows the intra-group topology of a Cascade group. Logically, a cascade group consists of a 6 x 16 mesh with fully connected X and Y dimensions. Each pair in the same row is connected by one link while each pair in the same column is connected by three links.

The optical connections in Cascade use a 24-channel optical cable which handles 4 links. Therefore, there are up to 240 optical connections in a group. The global optical links are used to connect the groups in an all-to-all manner. The number of connections between each pair of groups varies depending on the number of groups and can be as few as 1 optical cable for a maximum-size Cascade. In this dissertation we will focus on a 30-cabinet(15-group) installation of Cray Cascade which is also known as Edison.

Minimal Routing in a Dragonfly Network. Minimal routing chooses the shortest path to route a packet between a given source and destination. A minimally routed packet in a Dragonfly network always traverses at most a single global channel, given the fact that there is an all-to-all connection between groups. Also inside the source (destination) group, if the source (destination) router is not directly connected to the destination (source) group, the packet needs to be routed to a router with such a connection [33]. Depending on the intra-group topology, this will add extra local hops to the minimal path. For example, in the Dragonfly network of figure 2.5, the minimal path between a source s connected to router R_s and a destination d

connected to router R_d, traverses at most 1 local channel at the source group, 1 global channel and 1 local channel at the destination group.Therefore minimal routing algorithm can be expressed as follows:

Step 1: If $G_s \neq G_d$ and R_s does not have a connection to G_d

route within G_s from R_s to R_a, a router that has a global channel to G_d.

Step 2: If $G_s \neq G_d$

traverse the global channel from R_a to reach router R_b in G_d.

Step 3: If $R_b \neq R_d$

route within G_d from R_b to R_d.

Where G_s, G_d, R_s and R_d are source and destination groups and routers respectively and R_a and R_b are the end routers of the global channel which connects G_s and G_d.

Although minimal routing works well under uniform traffic, its performance suffers when encountering adversarial traffic patterns where most of the generated traffic is sent to a single destination group. For example in the network shown in Figure 2.5, if all nodes in G_s send packets to nodes in G_d, using minimal routing will result in congestion on the global link from R_a to R_b, because all of the generated packets have to take this link to reach G_d. In the next section we will discuss a load-balancing scheme to avoid congestion in similar situations.

Valiant Load-Balanced Routing. To balance the load under adversarial traffic, routing decisions should consider using non-minimal paths to avoid creating bottlenecks in the network. Originally, Valiants load-balanced(VLB) routing was proposed to be used at system level: to route each packet, first to a randomly selected intermediate group and then to its final destination [33]. Although this randomizes the global

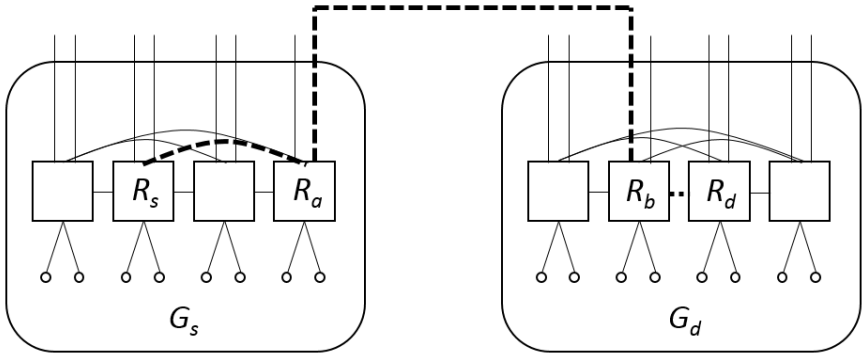


Figure 2.5: Minimal Routing in a dragonfly network

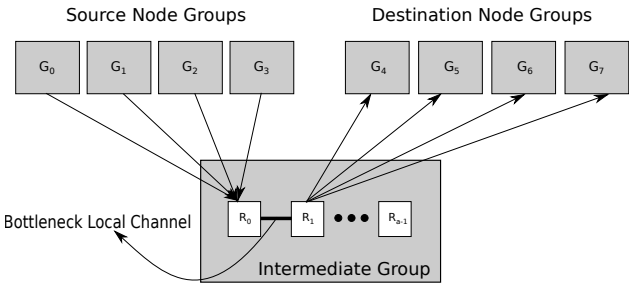


Figure 2.6: Local channel saturation under adversarial traffic in a dragonfly network using original VLB

channel usage, it can lead to congestion in the intermediate group as shown in figure 2.6 where G_0 sends traffic to G_4, G_1 sends traffic to G_5, etc. Since minimal routing is used inside the intermediate group, the particular local channel becomes the bottleneck. As a result this version of VLB only load-balances the global channels but does not necessarily load-balance local channels [46] [24]. In order to mitigate this problem, instead of choosing a random intermediate group, a random intermediate router needs to be selected. In this case, as shown in figure 2.7, the packet is routed to the intermediate router minimally, from where it will be routed minimally to its final destination. Using this mechanism, the local channel congestion showed in figure 2.6 can be avoided [46]. Therefore VLB routing can be specified as follows:

Step 1: Select a random intermediate router R_i

Step 2: If $G_s \neq G_i$ and R_s does not have a connection to G_i

route within G_s from R_s to R_a, a router that has a global channel to G_i.

Step 3: If $G_s \neq G_i$

traverse the global channel from R_a to reach router R_x in G_i.

Step 4: If $R_x \neq R_i$ and $G_i \neq G_d$

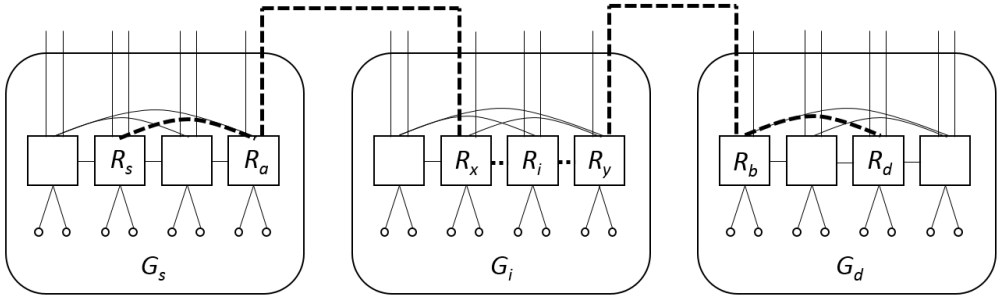


Figure 2.7: VLB Routing in a dragonfly network

route within G_i from R_x to R_i.

Step 5: If $G_i \neq G_d$ and R_i does not have a connection to G_d

route within G_i from R_i to R_y, a router that has a global channel to G_d.

Step 6: If $G_i \neq G_d$

traverse the global channel from R_y to router R_b in G_d.

Step 7: If $R_b \neq R_d$

route within G_d from R_b to R_d.

Although VLB routing achieves good load balancing under any traffic pattern, this comes with the price of using additional network resources to route each packet. Particularly under uniform traffic pattern, because of the load balanced nature of the traffic pattern, performance of VLB is considerably lower than that of minimal routing. As a conclusion, neither MIN nor VLB are suitable to be used under all traffic patterns, but it is desirable to design a scheme to adaptively decide on using either of them based on the current state of the network. In the following sections, I will discuss adaptive routing schemes on Dragonfly networks in which, the decision to route minimally or non-minimally is taken based on the current state of the network.

Universal Globally-Adaptive Load-Balanced Routing. As discussed earlier minimal routing on Dragonfly networks can result in very low bandwidth for adversarial traffic patterns and VLB routing can at most achieve half of the observed performance for minimal routing under uniform traffic. Therefore it is clear that none of these schemes can be applied to all traffic patterns that an HPC system may encounter and an adaptive routing scheme seems desirable. Ideally, an adaptive routing scheme should be able to accurately choose between MIN and VLB depending on some predefined criteria.

It has been shown [33] that congestion information is a good indication of network state at any given time and can be used to decide between MIN and VLB routing schemes on a packet-by-packet basis. Queue length is the metric that is used to identify congestion at each channel. The basic idea behind Universal Globally-Adaptive Load-balanced(UGAL) routing is that a packet is routed using MIN as long as the delay incurred by using the minimal path is less than that of using the VLB path. Otherwise, non-minimal path is preferred. The delay of a path can be calculated by summing up the queue lengths of all links throughout the path from a source to a destination. Assuming such information is available at the source router, UGAL will choose the minimal path to route the packet only if the following inequality holds:

$$Q_{minpath} \leq Q_{vlbpath}$$

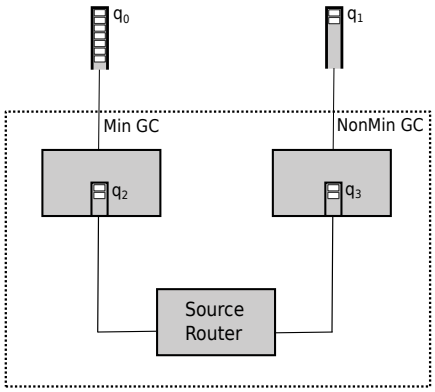
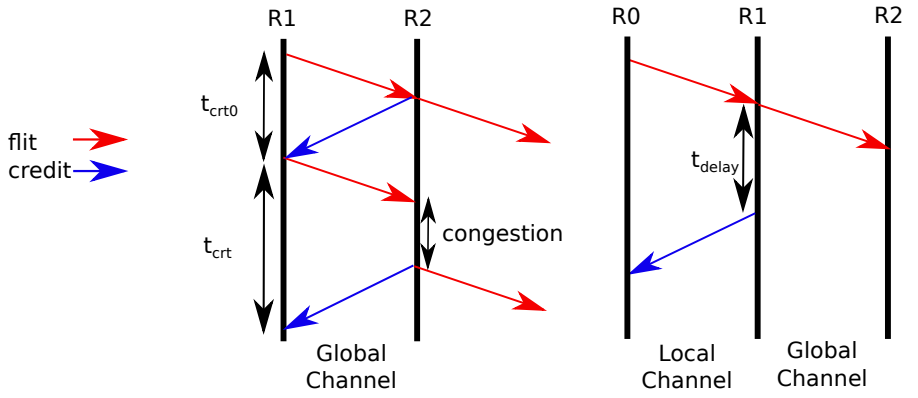


Figure 2.8: Congestion on global channels not directly connected to source router, must be inferred through local channels

where $Q_{minpath}$ and $Q_{vlbpath}$ are the sum of queue lengths throughout the MIN and VLB paths respectively. If the inequation does not hold, VLB path will be used. This variation of UGAL which requires any router to have global information about the current state of all links throughout the network at any time, is called UGAL-G. Practically such a scheme can not be implemented because: 1- Broadcasting global link state information incurs a huge load on network links which can degrade network performance; and 2- Even if such a mechanism exists, the link state can change between the time of routing decision and when the packet actually arrives at a specific link. Therefore, UGAL-G is mostly considered as a theoretical upper bound for the performance of any UGAL-like scheme. In practice, the routing decision can be made by estimating the delay of a route using the information that is locally available at the source router. UGAL-L estimates the delay by calculating the product of local queue length(Q) and path hop count(H) [33]. In other words, UGAL-L routes a packet minimally only if the following inequality holds:

$$Q_{min} \times H_{min} \leq Q_{vlb} \times H_{vlb} + T \tag{2.1}$$

where Q_{min} is the queue length on local channel for the MIN path; H_{min} is the hop count for the MIN path; Q_{vlb} is the queue length of the non-minimal path; H_{vlb} is the hop count of the non-minimal path; and 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). By default, adaptive routing decision is made on a packet-by-packet basis, by comparing one random minimal path and one random VLB path. However, in Cray Cascade machine, the routing pipeline selects up to four possible routes at random: two minimal and two non-minimal, and decides the path for the packet based on estimated link loads which are computed



(a) Definition of t_{crt}, t_{crt0} and t_d (b) Local credits are delayed based on t_d

Figure 2.9: Credit round-trip timeline [33]

using a combination of downstream link load, estimated far-end, and near-end link load. In all cases, if the inequality does not hold, VLB path will be used to route the packet. In the dragonfly network, since the global channels limit the network bandwidth and dominate network latency, we can simplify the above equation by using only the number of global hops: one for MIN and two for VLB making the inequality:

$$Q_{min} \leq 2 \times Q_{vlb} + T$$

Since global channels limit network performance, ideally we would use the global channel queue length in the routing equation. However, the source router does not always have direct access to the global channel queue length and hence uses the length of its local queues as proxies for the global channels. This approximation works well in networks with stiff backpressure but leads to high latency in networks with soft backpressure. This scenario is shown in figure 2.8, where q2 and q3 are used to approximate q0 and q1, respectively. In order for the source router to sense congestion on the minimal global channel, q0 must be filled before q2 reflects this congestion. The packets used to fill q0 experience very high latency resulting in high average latency on adversarial patterns. In the next four sections we will discuss indirect adaptive routing schemes which enable routers to sense congestion quickly and hence avoid the high latency for UGAL-L.

Credit Round Trip Routing. In [33], credit round trip (CRT) delay was introduced to reduce the latency observed when using UGAL on a dragonfly network. They showed that the congestion on global channels can be identified through delaying credits that are returned over local channels. Figure 2.9a shows the timing of the typical credit-based flow control. A flit departs router R1 and once it arrives at R2, advances

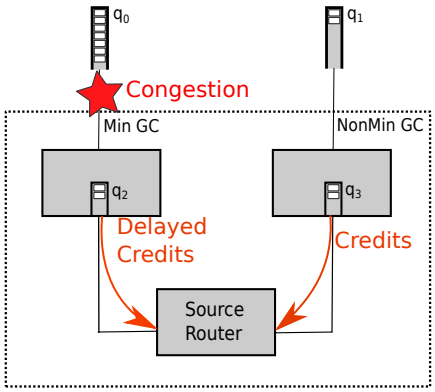


Figure 2.10: CRT routing: delayed credits on minimal route are associated with congestion

immediately at router R2. The credit for this flit returns to R1 in t_{crt0}, the zero-load credit round trip latency. If a flit is delayed by t_d because of congestion at R2, the credit round trip latency for this flit (t_{crt}) is increased by t_d, and hence the congestion, can be measured by subtracting t_{crt0} from the observed t_{crt}.

Based on the congestion sensed, credit round trip latency can be used to signal congestion as shown in Figure 2.9b. For each global channel GC, we measure t_d(GC) as described above to quantify the congestion on GC. Each time a flit is released to GC, the credit corresponding to this flit is delayed by t_d(GC) min(t_d), where min(t_d) is the minimum delay measured for global channels on that router. Delaying the return credit creates the illusion of shallower buffers and stiffer backpressure to the upstream router. As a consequence, the UGAL algorithm in the upstream router is able to adapt properly in making its routing decision. We use min(t_d) as the delay threshold to ensure the credit delay is adjusted dynamically based on the load of the network.Credit delay is only applied to credits traveling over local channels. Credits returning on the global channels are not delayed to avoid degrading the utilization of global channels.

An abstract representation of CRT routing is shown in Figure 2.10. Each time a flit is sent through minimal global channel, the credit is delayed and therefore, the source router senses the congestion on the minimal route faster.

Progressive Adaptive Routing. In progressive adaptive routing (PAR), the routing decision can be re-evaluated at each hop in the source group [28]. In other words while the flit is still in the source group, it can be re-routed to a VLB path at any hop based on the congestion at the global channels. However, any decision to route non-minimally at the source router or any of the subsequent hops, can not be changed afterwards. Using this mechanism, PAR is able to overcome the lack of minimal global channel information

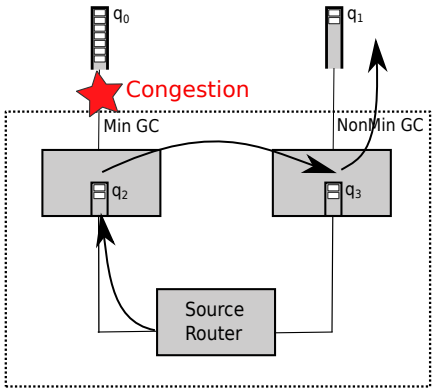


Figure 2.11: PAR routing: routing decision can be changed by observing congestion at subsequent local hops

on the source router, by allowing the packet to route minimally until it encounters congestion, where it can re-route to a non-minimal path. It should be noted that because the routing re-evaluation can not happen outside of the source group, the number of re-evaluations is at most equal to the number of hops at the source group. Figure 2.11 shows a situation where the source router decides to route the packet minimally based on its local queue information, but this decision changes at the next hop because of the congestion of the minimal global link. Therefore the packet is redirected to the VLB path.

There are 2 downsides to the use of PAR: first, PAR wastes the network resources when a minimal routing decision is later reversed. In this case all the minimal hops taken before taking the non-minimal path are wasted resources. Second, PAR also needs an addidtional virtual channel to avoid deadlocks which can be created because of the dependency between local minimal channels before misrouting happens and local non-minimal channels after the packet is decided to take the non-minimal path.

Piggyback Routing. As mentioned in previous sections, UGAL-G achieves the theoretical upper-bound for any adaptive routing scheme which decides to route using MIN or VLB only based on queue length information because, it is assumed to have a global knowledge of all link states throughout the network. The piggyback (PB) method tries to mimic this behavior by broadcasting link state information of the global channels to all adjacent routers [28]. A link state bit vector is piggybacked on every packet and broadcast on idol channels. Thus, each router maintains the most recent link state information for every global channel in its group. This information is used along the local queue information to make the routing decision. Figure 2.12 shows a situation in which piggybacked information tell the source router

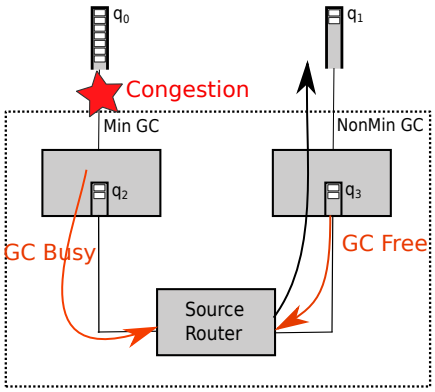


Figure 2.12: PB routing: global channel congestion information is sent back to the source router

that the global channel on the minimal path is congested while the global channel on the VLB path is free. Therefore the source router decides to send the packet non-minimally.

It should be clear that using PB in practice results in added load on local links in a group. To minimize this effect, the congestion information of each global channel (S_{GC}) is compressed into a single bit according to the following equation:

$$S_{GC} = Q_{GC} > 2 \times \overline{Q} + T$$

If S_{GC} is true, global channel GC is congested, otherwise it is uncongested. To compute congestion we compare Q_{GC} to $\overline{Q}$, the average queue length of other global channels on the same router. A routing threshold T is added to the comparison to smooth out transient load imbalances on the minimal channels.

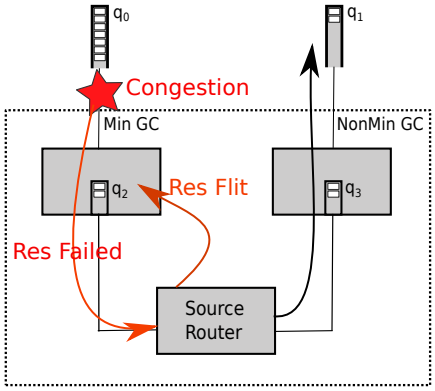


Figure 2.13: RES routing: failed reservation, indicates congestion on minimal path

Reservation Routing. The reservation (RES) routing starts by attempting to reserve the required bandwidth on the minimal global channel before actually sending the packet minimally [28]. Only if the reservation is successful, it sends the packet through the minimal path. Otherwise, VLB path will be selected. In each router, a reservation counter R needs to be maintained for each global channel in the corresponding group, to count the number of flits that have reserved a global channel but not yet traversed it. To make the reservation, the source router sends a *reservation flit* to the router connected to the minimal global channel, while storing the packet that is being routed in a buffer until the reservation flit returns. The reservation flit carries the average of the reservation counters ($\overline{R}$) of the source router. The minimal global channel's reservation level is compared to the average using following inequality:

$$R_{GC} \leq 2 \times \overline{R} + T$$

If the inequality holds, R_{GC} is incremented by the number of flits in the packet, and a positive indication is returned to the source router causing the packet to route minimally. If the inequality is false, R is left unchanged, and a negative indication is returned to the source router causing the packet to route non-minimally. When a minimally routed flit traverses a global channel, the reservation counter is decremented by 1. Figure 2.13 shows a situation where reservation fails due to congestion on the minimal global channel and the packet is routed non-minimally.

Although RES can make accurate routing decisions based on the reservation counts and also does not suffer from delay in updating link state information because of reserving the global channel before sending the packet, it requires a considerable amount of buffer space in the source router to hold pending packets. Also packet latency is increased by the amount of time needed for the reservation flit to go to the minimal global channel router and return to the source router (reservation flit round-trip time).

On-the-fly Adaptive Routing. In [25] authors present two deadlock-free routing mechanisms, opportunistic (OLM) and local (RLM) misrouting, which support in-transit adaptive routing where packets can circumvent congested links via local and global misrouting. Each router tries to forward traffic minimally. Nevertheless, if minimal routes are congested, packets will be misrouted off their shorter paths. The longest allowed route includes one global misrouting and one local misrouting per each visited group (source, intermediate and destination groups).

Adaptivity is exploited on-the-fly, the routing decision can be revisited on each hop. Routing chooses between the minimal output and one of the possible non-minimal outputs using a misrouting trigger based on the credits count of the output ports. If the minimal output is not available, a non-minimal output is randomly

chosen among those with an occupancy lower than a given threshold. This threshold is a percentage of the occupancy of the minimal queue and its value can be set empirically. Global misrouting will be performed, if necessary, in the source group, either at the source router or after the first minimal hop as in PAR.

The only difference between these two routing schemes is the deadlock avoidance mechanism. RLM eliminates the possibility of creating a deadlock by restricting the non-minimal paths that can be chosen for a packet, in a fully connected Dragonfly group. This is achieved by assigning parity and sign values to each hop and preventing some combinations to be chosen as non-minimal paths. On the other hand in OLM, packets can freely circulate on a network, even creating cyclic dependencies, as long as they always have a deadlock-free route to their destination that allows breaking such dependency.

OFAR [24] takes the same approach as OLM and RLM to allow the misrouting decision to be taken at any hop throughout the path, but with a different approach to avoid deadlocks. The authors propose using an escape subnetwork in the form of a Hamiltonian ring, which can be implemented physically or virtually through virtual channels. A packet enters the escape subnetwork only if the minimal route is deemed as congested and misrouting is not possible (due to misrouting threshold or previously taken non-minimal hops).

Contention-based Adaptive Routing. In [23] authors claim that traditional congestion detection through queue lengths and credit counts suffer from some limitations. Among those, the amount of time needed for a credit counter to be updated after a packet has been forwarded in the downstream router, contributes to the uncertainty while making the routing decision. Also when encountering traffic pattern changes, a significant amount of time is needed to fill the buffers so that an alternate path is selected.

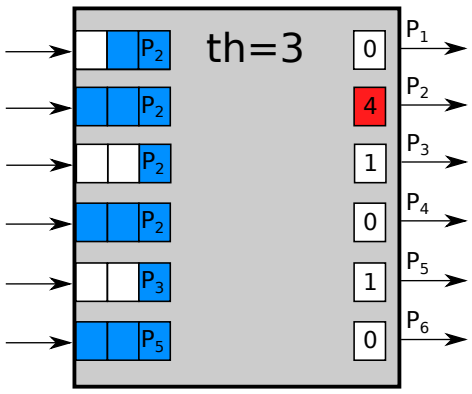


Figure 2.14: Contention detected in port P_2 since its counter exceeds the threshold [23]

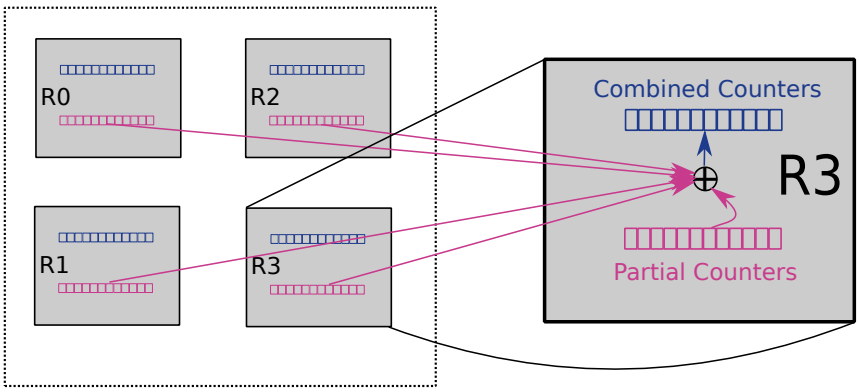


Figure 2.15: Combination of contention counters in router A in EC_tN [23]

The idea behind the contention-based adaptive routing is to decide the path to follow based on the contention level of each port, estimated from flows in the input queues that would proceed minimally through each output port. When many packets want to go on a given output, such output suffers from contention. In such case, packets will be diverted to alternative paths using non-minimal routing, without requiring the queues to be full. Hence, the mechanism decouples the buffer capacity from the misrouting trigger mechanism. Figure 2.14 shows a case in which 4 flows are competing for the same output port, whereas the contention threshold is set to 3. Therefore, a non-minimal path will be utilized to route the packet.

Three variations of contention-based routing have been introduced in [23]. The *Base* method employs one counter per output port,as depicted in Figure 2.14. When the header of a packet reaches its input buffer head, the routing mechanism determines its minimal output path and increases the corresponding contention counter. Alternative (nonminimal) routing is triggered only when the contention counter in the minimal path of the packet exceeds a given threshold. This contention counter remains increased until the packet is completely forwarded, even though the packet might be transmitted through a different output port. Thus, counters are decremented only when a packet tail is removed from the input buffer. The second method introduced is a *Hybrid* method which combines the contention and congestion levels to make the routing decision. The traffic is routed non-minimally when any contention or congestion levels are deemed to be high on the minimal path. In the third mechanism which is called Explicit contention notification (EC_tN), contention information is distributed among several routers to help them make a more accurate routing decision. As shown in Figure 2.15, every router maintains two arrays of global contention counters labeled as combined and partial counters(one counter per each global link in the group). The partial counters are incremented when an injected packet is being sent to a remote group and decremented when the packet

leaves the input queue. Periodically, the routers broadcast their partial arrays, which upon reception is used to update the combined arrays.

Among the three methods, the complexity to implement the *Base* and *Hybrid* mechanisms is very low: several parallel counters need to be updated and compared for every packet being sent, which is not significant. Additionally, the update of contention counters is not very time-sensitive, since a slight delay does not significantly harm performance. By contrast, the cost of EC_tN can be significant. EC_tN requires two additional sets of counters (partial and combined) plus the required memory to hold the partial values received from other routers. This makes EC_tN not suitable to be used in large scale networks with high radix.

2.2 Software Defined Networking

A computer network can be seen as a three-layer structure. We call each of these layers of functionality a *plane*, and divide the network functionality into *data*, *control* and *management* planes. The data plane is responsible for forwarding data and can be refered to as networking devices. The control plane is responsible for forwarding and routing mechanisms on network devices and the management plane is in charge of monitoring and configuring the control functionality. Generally, network policies are defined at management layer, enforced on control plane and executed on the data plane.

Traditionally, control and data planes are tightly coupled, meaning that they are embedded in the same networking devices in a highly decentralized manner. This is known to be crucial in the design of highly distributed networks like Internet, because it guarantees network resilience. However, this decentralized structure has also resulted in very complex and static architectures, where innovation and flexibility are hard to achieve.

The term SDN(Software Defined Networking) is used to describe an architecture in which the data plane functionality is decoupled from the control logic and handled by a remote entity [35]. An SDN architecture can be formally defined by the following criteria:

- The networking devices are simple forwarding elements without further knowledge of control structure. In other words, the control and data planes are decoupled.
- IN SDN context, a $flow$ is defined as a sequence of packets from a source to a destination, which receive identical forwarding services at all networking devices. Each data plane device applies a set of predefined instructions known as *actions* for each flow.

- The control plane functionality is moved to an entity known as *controller* or Network Operating System. The controller typically runs on a commodity server and is responsible for configuring forwarding policies on networking devices. SDN controller has a logically centralized and global view of all network.
- The network behavior including controller functionality can be *programmed* through implementing and running software applications on top of the network operating system.

An abstract SDN architecture is depicted in Figure 2.16. As mentioned earlier, networking devices are simple forwarding elements and the control logic has been moved to the external SDN controller. On top of the logically centralized controller, network services which are commonly accomodated by middleboxes, can be programmed as applications. This includes but not limited to some common network functionality like routing and MAC learning.

A simple OpenFlow-enabled forwarding device is shown in Figure 2.17 which includes a pipeline of flow tables. Each flow table entry specifies a matching rule, an action and counters that keep statistics for the matching flow. When a new packet arrives, either a match happens at one of the flow tables in which case the corresponding action is executed, or there are no rules to handle the packet(miss). If a miss happens, the packet will be discarded however, usually a default rule is installed to send the missed packets to the controller for final decision. There are four allowed actions: (1)forward the packet to outgoing port(s), (2)forward to controller, (3)drop the packet, and (4)send the packet to normal processing pipeline.

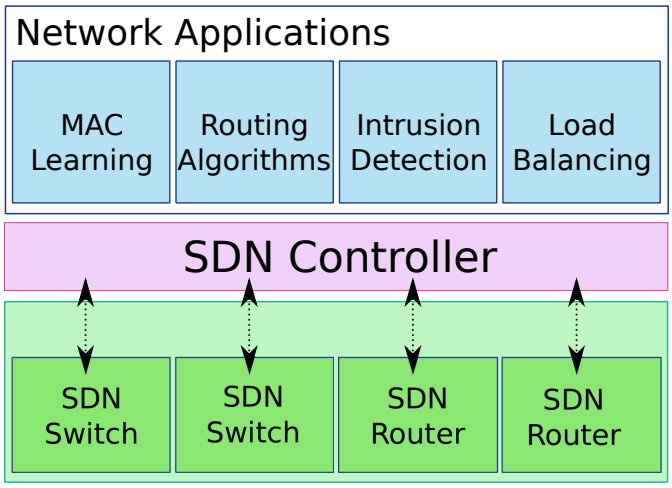


Figure 2.16: SDN architecture [35]

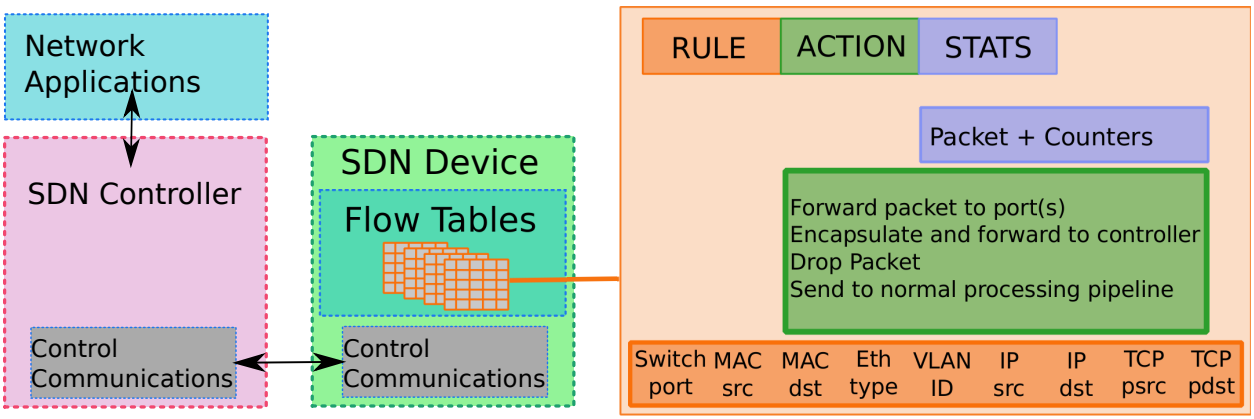


Figure 2.17: OpenFlow-enabled SDN devices [35]

One of the major strengths of OpenFlow-based SDN is its ability to manage network resources using the logically centralized global network view [35]. I use the term *SDN routing* to refer to this capability. Under the assumption that the set of flows is known, SDN can examine the global traffic demand of the whole network, manage resources according to the underlying network topology, and aim for global optimality in terms of resource utilization. Hence, even without the fine-grained control of packets at the switch or router level, SDN may achieve high performance by managing network resources more effectively.

Since the concepts of SDN and OpenFlow were introduced, SDN has been widely accepted in industry and the research community. Extensive research and development has been carried out in this area. Most related to this work is the research to explore SDN capability in the HPC domain. Arap [7] investigated techniques to explore SDN capability for efficient MPI collective communications; Takahashi [45] evaluated the performance of the MPI-allreduce operation on an SDN cluster. The MPI-Reduce operation on an SDN cluster has also been developed [38]. Research effort is underway to leverage the SDN capability to build new MPI libraries that can take advantage of SDN capabilities [40, 44].

2.3 Throughput Modeling

The scale of current high-end HPC systems and data centers is in the range of hundreds of thousands of processing cores [16] and as we are moving towards exa-scale computing, million-core systems are expected. At these scales, interconnect networks are becoming critical components of modern computer systems. The design of such networks, including architecture design space exploration and performance prediction is challenging. Specifically, cycle accurate simulation of such systems often requires distributed

systems with HPC-scale capabilites. Therefore, using performance models with higher levels of abstraction is gaining more attraction.

Given a traffic pattern, there are various models to quantify the aggregate throughput performance of an interconnect network. In this section I introduce two well-known throughput models, which are used in subsequent sections, to predict and evaluate the throughput performance of studied networks. First, I will introduce notations used in the throughput models.

Let A be a set. $|A|$ is the size of the set. Let a network be represented as a graph $G = (V, E)$, where V is the set of nodes and E is the set of links in the network. $V = PE \cup S$ contains two types of nodes. PE is the set of compute nodes; and S is the set of switches. The nodes are numbered from 0 to $|V| - 1$. For each link $e \in E$, C_e is the link capacity. Let $s \in PE$ and $d \in PE$ be two compute nodes. A flow from s to d is denoted as (s, d). A traffic pattern F is a set of flows to be considered. The traffic in a flow is carried over a set of paths for the flow. Each path p is represented as a set of links. For a flow (s, d), $P_{s,d}$ is the set of all paths available for the flow; Let $e \in E$ be a link. If a path p uses a link e, we say that $e \in p$. For a flow (s, d), $P_{s,d}(e)$ is the set of paths that use link e; Table 2.1 summarizes the notation.

Table 2.1: Notation used in the models

$G = (V, E)$	the topology with node set V and edge set E
C_e, $e \in E$	link capacity
(s, d)	a flow from s to d
F	the set of flows to be routed
$P_{s,d}$	the set of all available paths for (s, d)
$P_{s,d}(e)$	$\{p \mid e \in p \text{ and } p \in P_{s,d}\}$

2.3.1 Max-Min Fair (MMF) Throughput Model

The load distribution problem is formulated as an optimization problem using linear programming (LP): given the set of flows to be routed, how to allocate rate for each path so that the pattern achieves maximum aggregate throughput under max-min fairness. Note that max-min fairness is commonly assumed in the modeling of network performance and is the base for many network rate allocation and resource management schemes. The LP formulation is based on a formulation for multi-commodity flows given by Nace et al. [41], which assumes that all possible paths can be used to route traffic (multi-commodity flow). The

solution to the LP program gives the rate allocation for all flows and all paths that maximizes the throughput for the pattern under max-min fairness.

1 Set $k = 0$ and $L_0 = \phi$

2 Set $\lambda_{sd} = 0 \; \forall (s,d) \in F$;

3 **while** $L_k \neq F$ **do**

4 Set k = k+1; $L_k = L_{k-1}$;

5 Solve the Linear Programming problem OPT_k:

6 Maximize α

7 Subject to:

8 $\alpha - (\sum_{p \in P_{s,d}} x_{s,d}^p) \leq 0, \; \forall (s,d) \in F - L_{k-1}$ (1)

9 $\sum_{p \in P_{s,d}} x_{s,d}^p = \lambda_{sd}, \; \forall (s,d) \in L_{k-1}$ (2)

10 $\sum_{p \in P_{s,d}(e),(s,d) \in F} x_{s,d}^p \leq C_e, \forall e \in E$ (3)

11 **For** each binding constraint among LP_k Constraints (1)

12 Let (s,d) be the corresponding flow ;

13 $\lambda_{sd} = \alpha$;

14 $L_k = L_k \cup \{(s,d)\}$;

15 **end**

16 **end**

17 output the $x_{s,d}^p$ for all $(s,d) \in F$.

Figure 2.18: Algorithm for computing optimal rate allocation for a set of flows with max-min fairness

Given the set of flows to be routed, the optimal rate allocation considers all available paths for each flow and allocates a rate to each path such that the aggregate throughput is maximized under max-min fairness. The algorithm to compute the rate allocation, which is shown in Figure 2.18, is an extension of the algorithm for multi-commodity flow model in [41]. In the algorithm, k is the iteration number; and L_k is the set of flows that have been assigned rate at or before iteration k; λ_{sd} is the rate assigned to flow (s,d). The algorithm iteratively solves a linear programming problem OPT_k. In each iteration, the maximum rate that is possible for all flows under consideration is computed by solving OPT_k. After the LP problem is solved, the most congested flows, which are the flows whose rates are saturated at this iteration, are identified and their rates are recorded (Lines 11 to 15). The details about how to identify the most congested flows can be found

1 Solve the Linear Programming problem MCF:

2 Maximize α

3 Subject to:

4 $\alpha - (\sum_{p \in P_{s,d}} x^p_{s,d}) \leq 0, \ \forall (s,d) \in F$ (1)

5 $\sum_{p \in P_{s,d}(e),(s,d) \in F} x^p_{s,d} \leq C_e, \forall e \in E$ (2)

6 output the $x^p_{s,d}$ for all $(s,d) \in F$.

Figure 2.19: Algorithm for computing MCF rate allocation for a set of flows

in [41]. When the rates for all flows are assigned, the algorithm will come out of the loop at Line 16. The solution of the last LP problem contains the rate allocated to each path for each flow ($x^p_{s,d}$) that can achieve the optimal rate allocation for the whole pattern under max-min fairness.

In the linear program formulation OPT_k, for each flow (s,d), a variable $x^p_{s,d}$ is assigned to each possible path $P_{s,d}$. After the final iteration of the algorithm, these variables provide rate allocation for individual paths while the set λ provides rate allocation for all flows. $\sum_{p \in P_{s,d}} \{x^p_{s,d}\}$ is the rate allocated for flow (s,d). Constraints (1) at Line 8 ensure that at any iteration, the rates for all flows yet to be allocated are no less than the solution flow rate of that iteration. Constraints (2) at Line 9 assign the rate for flows whose rates have been determined in the earlier iterations. $\sum_{p \in P_{s,d}(e),(s,d) \in F} x^p_{s,d}$ is the accumulated rate on link e; and Constraints (3) at Line 10 are link capacity constraints that ensure that the rate on each link is no more than its capacity.

2.3.2 Maximum Concurrent Flow (MCF) Throughput Model

MCF rate allocation adds another constraint to MMF rate allocation, in which all flows can attain an identical rate. In other words, the maximum achievable rate by any flow in the network is bound by the flow(s) that are passing the most congested link(s) in the network. As a result, MCF can be seen as the first iteration of an MMF rate alocation after which, the algorithm stops and assigns the calculated rate to all flows in the network.

Figure 2.19 is the algorithm for computing MCF rate allocation, which is similar to Figure 2.18. In this case, once we get the rate allocation for the first saturated flow, the algorithm stops and assigns the calculated rate to all flows.

CHAPTER 3

NETWORK TOPOLOGY DESIGN AND EVALUATION

3.1 Introduction

In this chapter I investigate the performance bounds of a recently proposed topology and find a near-optimal topology that outperforms any topology in this line of design [18, 20]. The quest for low latency and high bandwidth interconnects has led to the idea of using random topologies for future extreme scale data centers and HPC clusters [34,42,43,48]. In a *random regular graph* (RRG), which is also known as the Jellyfish topology [43], all nodes have the same degree and links are bidirectional and connected randomly. An RRG is a special case of a *directed regular graph* (DRG) where links are unidirectional and the incoming and outgoing degrees of all nodes are the same. RRGs offer low diameter and high bisection bandwidth. In addition, they also provide high path diversity such that using k-shortest path routing (or its variants) is sufficient to exploit the network capacity [43]. It has been shown that RRGs achieve higher performance than similar cost fat-trees that are widely deployed in the current data centers and HPC clusters, which argues for using an RRG as a design alternative for the future extreme-scale interconnects [42, 43, 48].

The RRG topology represents a class of high capacity topologies with sufficient diversity among short paths such that k-shortest path routing and its variants are sufficient to explore the network capacity. This is different from other recent interconnect proposals such as SlimFly [10] and Dragonfly [33] that rely on Valiant Load Balance (VLB) routing to exploit network capacity for some traffic patterns. For a network topology to be effective with k-shortest path routing, we propose that it must have the following properties.

- **Property 1 (large aggregate capacity)**: The topology must have high aggregate capacity. This property is historically measured by bisection bandwidth.
- **Property 2 (minimal resource usage)**: The diameter and the average path length of the k-shortest paths between source-destination (SD) pairs must be small. By using shorter paths to communicate data, less network resources are needed on average to transmit each packet, which results in more efficient network resource utilization and higher aggregate throughput.
- **Property 3 (load balance)**: For k-shortest path routing to fully exploit the network capacity, the paths must be evenly distributed over the network.

From graph theory, the bisection bandwidth of an RRG is close to optimal with very high probability [11]. However, it is unclear how close an RRG is to the optimal in terms of diameter, average k-shortest path length, or load balance with k-shortest path routing. Existing studies of RRGs either compare their performance to other topologies such as fat trees [43, 48] or perform a coarse-grained upper-bound performance estimation [42]. Despite those studies, RRGs are still not well understood. For example, it is unclear if there exists any type of traffic that can result in poor performance on an RRG.

Because an RRG is a special case of a DRG its performance is bounded by the bounds for a DRG. In this work, we establish the bounds for DRGs on diameter, average k-shortest path length, and a load balancing property with k-shortest path routing that is quantified by the maximum link load for all-to-all communication. These bounds are then used to evaluate RRGs. The results show that for most values of k, RRGs perform well for average k-shortest path length, but not for diameter and load balancing.

We further study the *Generalized de Bruijn Graph* (GDBG) [9], a deterministic DRG. We identify a form of k-shortest path routing, which we call *hop-limited all path routing* (ALLPATH(H)), for GDBGs. With ALLPATH(H), all short paths that are no more than H hops are used to carry traffic. We prove that with ALLPATH(H), for most network configurations, a GDBG is near-optimal in diameter, average k-shortest path length, and load balance. Hence, GDBGs can achieve near optimal performance among all DRGs when ALLPATH(H) routing is used. This indicates that a GDBG is a highly effective topology with k-shortest path routing and can be directly applied in many situations such as the logical topology for a distributed peer-to-peer network where k-shortest path routing is used. Moreover, a GDBG can be used in simulation and modeling as a benchmark to evaluate the performance of other regular graph or directed regular graph topologies such as RRGs to investigate their performance for some specific traffic patterns. We compare the performance of a GDBG with that of an RRG through modeling and simulation and identify the strengths and weaknesses of RRGs. Our results refine Singla et al.'s earlier conclusion that the performance of an RRG is close to optimal [42]. We show that RRGs are close to optimal for diverse traffic in which each switch communicates with many other switches but that they are much less effective than GBDGs (let alone the optimum) in dealing with concentrated traffic patterns such as switch-level permutations and shift communication in which each switch communicates only with one other switch.

The contributions of this work include the following.

- We identify a near optimal topology for k-shortest path routing: the generalized De Bruijn graph (GDBG). We prove that with hop-limited all path routing—a form of k-shortest path routing—GDBGs achieve near-optimal performance for most network configurations.

- We show that the RRG with k-shortest path routing is not ideal in terms of either diameter or load balancing.
- We use modeling and simulation results to confirm that RRGs with k-shortest path routing achieve a considerably lower performance in compare to GDBGs with hop-limited all path routing.

3.2 Bounds of DRG Topological Metrics

Each link in an RRG topology can be represented as two unidirectional links, one in each direction. Hence, $RRG(N,r)$ belongs to the class of *directed r-regular graphs*, in which links are unidirectional and each node has r incoming and r outgoing links. $DRG(N,r)$ denotes an arbitrary directed r-regular graph with N nodes.

Some special DRG topologies are considered in this paper: the Kautz graph [31] and the generalized De Bruijn graph (GDBG) [15, 26, 37]. The Kautz graph achieves the optimal diameter given a degree and a network size. A GDBG is also a low-diameter and high-connectivity topology. While many results for these topologies have been obtained, our work focuses on k-shortest path routing properties of GDBGs, which is a new contribution.

In order for a $DRG(N,r)$ that uses k-shortest path routing to achieve high performance the topology must have the three properties listed in Section 3.1. Since it is known that the bisection bandwidth for an RRG is likely to be close to the optimal [11], we will study Properties 2 and 3 and derive bounds on diameter, average length of k-shortest paths, and load balancing.

3.2.1 Diameter

Lemma 1 (diameter): The diameter of any $DRG(N,r)$ is at least

$$\lceil log_r(N(r-1)+1) \rceil - 1.$$

Proof: Consider any source node. From Moore's bound, we know that the node can reach at most $1+r+r^2+\cdots+r^H$ nodes in H hops. Let D be the diameter of the $DRG(N,r)$. We have $1+r+r^2+\cdots+r^D \geq N$. Simplifying the inequality, we obtain $D \geq \lceil log_r(N(r-1)+1) \rceil - 1$. □

3.2.2 k-shortest Path Properties

There are different ways for k-shortest path routing to work. One is to fix k and select k-shortest paths for each source-destination (SD) pair [43]. In this case, the average path length of k-shortest paths for all

SD pairs is an important performance metric because it directly reflects the amount of resources used to send a packet. Note that the average shortest path length is a special case when $k = 1$. The second method is to fix a hop limit (H), and use all paths between an SD pair whose hop counts are no more than H (assuming that all SD pairs have at least one path that is H-hop or shorter) [48]. We will call this variant of k-shortest path routing *hop-limited all path routing* (ALLPATH(H)). Later in the paper we will show that a GDBG with a form of ALLPATH achieves near-optimal performance for most network configurations. With ALLPATH(H), the number of short paths whose hop count are no more than H between each SD pair is an important parameter. Next, we will derive bounds on these two metrics for $DRG(N,r)$.

The average k-shortest path length is a direct measurement of the path quality for k-shortest path routing. Cerf et al. show that the lower bound on the average shortest path length of any r-regular network of size N is $D \geq \frac{\sum_{j=1}^{h-1} jr(r-1)^{j-1}+hR}{N-1}$ where $R = N-1-\sum_{j=1}^{h-1} r(r-1)^{j-1} \geq 0$ and h is the largest integer such that the inequality holds [13]. Note that the result is for single path routing on an undirected r-regular graph. We extend this bound for k-shortest path routing on $DRG(N,r)$.

Lemma 2: For any $DRG(N,r)$, the average path length of all k-shortest paths between all SD pairs, $AKH(N,r,k)$, is

$$AKH(N,r,k) \geq \frac{\sum_{j=1}^{h-1} jr^j + hR}{k(N-1)}$$

where

$$R = k(N-1) - \sum_{j=1}^{h-1} r^j \geq 0$$

and h is the largest integer such that the inequality holds.

Proof: Consider the paths from any source node. Because the node degree is r, the number of 1-hop paths is at most r; the number of 2-hop paths is at most r^2; ...; the number of i-hop paths is at most r^i. Let h be the largest integer such that

$$R = k(N-1) - \sum_{j=1}^{h-1} r^j \geq 0 \quad .$$

Consider the $k(N-1)$ k-shortest paths from the source node to all other $N-1$ nodes. Among the $k(N-1)$ paths, at most r paths can be 1-hop paths, r^2 paths can be 2-hop paths, and r^i i-hop paths for all $i = 1,2,\ldots,h-1$. The remaining $R = k(N-1) - \sum_{j=1}^{h-1} r^j$ paths have at least h-hops. Hence, the average path length for the $k(N-1)$ shortest paths from the source is at least

$$\frac{\sum_{j=1}^{h-1} jr^j + hR}{k(N-1)} \quad .$$

Because this applies to all source nodes,

$$AKH(N,r,k) \geq \frac{\sum_{j=1}^{h-1} jr^j + hR}{k(N-1)} \quad . \square$$

We will use $LK(N,r,k) = \frac{\sum_{j=1}^{h-1} jr^j + hR}{k(N-1)}$ to denote the lower bound of average k-shortest path length for any $DRG(N,r)$. Given a fixed H, the following lemma gives the upper bound of the number of paths for each SD pair whose length is no more than than H in $DRG(N,r)$.

Lemma 3: For a given H, the upper bound of the smallest number of paths whose length is no more than H between each SD pair in any $DRG(N,r)$ is $\lfloor \frac{r+r^2+\cdots+r^H}{N-1} \rfloor$.

Proof: Consider any source, there are at most r 1-hop paths, at most r^2 2-hop paths, ..., at most r^i i-hop paths. Hence, the total number of paths whose length is no more than H is at most $r + r^2 + ... + r^H$. Because there are $N-1$ destinations from the source node, at least one SD pair will have no more than $\lfloor \frac{r+r^2+\cdots+r^H}{N-1} \rfloor$ paths of H or less hops. $\square$

3.2.3 Load Balancing

Another property to ensure that a DRG topology performs well with k-shortest path routing and its variants is how evenly paths are distributed among all links in the topology. Ideally, for an SD pair that is uniform randomly selected among all SD pairs, all links in the network should have an equal probability to be used by its paths. This property is reflected in the maximum link load for the all-to-all communication that is calculated as follows. The load on each link is initialized to 0. For each SD pair (s,d) that uses X paths to carry traffic, if a link l is used by one of its paths, the load on the link is increased by $\frac{1}{X}$. The maximum link load is the load on the link with the largest load after all SD pairs are considered. Clearly, if each path for an SD pair is equally likely to be selected to carry traffic for the SD pair, the load of a link, as it is calculated, directly reflects the possibility that the link is selected to carry traffic. If the link load for the all-to-all communication is not evenly distributed among all links, some links will have much higher loads (based on the calculation) than others. These links will have a higher chance of being used to carry traffic; and this can lead to network hot spots. On the other hand, if the traffic is evenly distributed across the network, the maximum link load should be very similar to the average link load; all links will be selected to carry traffic with similar probability. The next theorem establishes the lower bound of the maximum link load for all-to-all communication.

Lemma 4: For any $DRG(N,r)$, the lower bound of the maximum link load for all-to-all communication is

$$ML(N,r) \geq \frac{LK(N,r,1) \times (N-1)}{r}.$$

Proof: All-to-all communication has $N(N-1)$ SD pairs. Consider an arbitrary SD pair (s, d). Let $shortest(s,d)$ be the hop count of the shortest path for SD pair (s,d). Let there be X paths between the SD pair. Since all of the paths have no less than $shortest(s,d)$ hops, the total load contributed by this SD pair to all links in the network is no less than $shortest(s,d) \times \frac{1}{X} \times X = shortest(s,d)$. As defined earlier, $LK(N,r,1)$ is the lower bound of the average shortest path length for any $DRG(N,r)$. The total load contributed by all SD pairs in the all-to-all communication is no less than $N(N-1) \times LK(N,r,1)$. The bound of the maximum link load is achieved when this lower bound of contributed load is evenly distributed among all $N \times r$ links in the network. Hence the lower bound of the maximum link load for all-to-all communication is $ML(N,r) \geq \frac{N \times (N-1) \times LK(N,r,1)}{N \times r} = \frac{LK(N,r,1) \times (N-1)}{r}$. □

3.3 Generalized De Bruijn Graph and Its Properties

In this section, we will formally prove that for most network configurations (values of N and r), the Generalized De Bruijn Graph (GDBG) with a specific hop-limited all path routing is near optimal for diameter, average k-shortest path length, and load balancing, and is thus a near optimal topology for k-shortest path routing. We will first describe the construction of the Generalized De Bruijn Graph (GDBG), and then prove the properties of GDBGs with the k-shortest path routing scheme.

3.3.1 GDBG Construction and Routing

An N-node r-regular GDBG, denoted as $GDBG(N,r)$, consists of N nodes, numbered from 0 to $N-1$. Each node has r incoming links and r outgoing links. The nodes are connected as follows. The r outgoing links of node 0 are connected to nodes $0, 1, \ldots, r-1$; the r outgoing links of node 1 are connected to nodes $r, r+1, \ldots, r+r-1$; the r outgoing links of node i are connected to nodes $(i \times r) \bmod N$, $(i \times r+1) \bmod N$, $\ldots$, $(i \times r+r-1) \bmod N$ for all $i = 0, 1, \ldots, N-1$. Basically, the outgoing links from the nodes are arranged in a round-robin fashion, cycling through all N nodes repeatedly. From this construction it can easily be seen that the outgoing degree of each node is r. Because the outgoing links cycle through N nodes in a round-robin fashion, the $N \times r$ incoming links connect to the N nodes, and each node has $\frac{Nr}{N} = r$ incoming links.

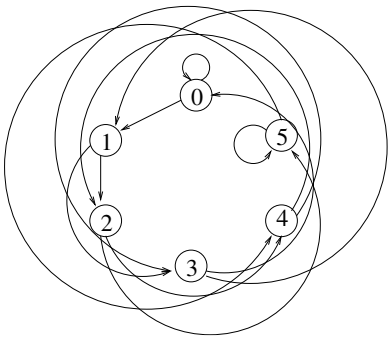


Figure 3.1: A GDBG(6,2) topology

A node may sometimes connect to itself through a loop-back link. For example, node 0 connects to node 0. It is proved in [37] that the number of such loop-back links is usually small as summarized in the following Lemma.

Lemma 5: The number of loop-back links in $GDBG(N,r)$ is at most $2r$, when $r \geq 2$. □

Regardless of the network size, the number of potential loop-back occurrences in $GDBG(N,r)$ is at most $2r$ when $r \geq 2$. When $r \ll N$, the impact of such links is negligible. Such loop-back links may be re-wired to increase the network capacity. However, since the number is small, we will not consider re-wiring in this paper. Figure 3.1 visualizes an example $GDBG(6,2)$.

For $GDBG(N,r)$, let H be the smallest integer such that $r^H \geq N$, the hop-limited all path routing will use all paths that are no more than H hops to route traffic. We will use the notion ALLPATH(H) to denote the routing scheme for GDBGs. Note that the value of H can be derived from N and r. For example, for $GDBG(N = 3000, r = 40)$, $H = 3$. As we will show in the following, when $r^H \geq N$, at least one such path exists between each SD pair. However, in most configurations, the number of short paths is sufficiently large to provide path diversity. For example, for $GDBG(3000,40)$, $H = 3$, the topology guarantees that there are at least 21 paths between each SD pair that are no longer than 3 hops.

3.3.2 Properties of the GDBG Topology

We will show that a GDBG with ALLPATH(H) is a near-optimal topology in terms of average k-shortest path length (Lemma 8) and load balancing (Lemmas 10 and 11). For the completeness of this paper, we also show the bound for the diameter (Lemma 7), which has previously been proven [26, 37]. The proofs of these lemmas are based on Lemma 6, in which a concept called *raw path*—also known as *walk* in the literature—is used. In contrast to a simple path that does not contain loops, a raw path may be a simple path

or may be a path that contains loops. A raw h-hop path from a source node contains h links starting from the source node, potentially including loop-back links. It can be realized by either an h-hop simple path when there is no loop or a shorter simple path when the raw path contains loops, which are removed in the shorter simple path. **Lemma 6**: For any source node i in $GDBG(N,r)$, its raw h-hop paths reach r^h continuously numbered nodes (with wrap around due to the modular operation) $i \times r^h \bmod N, \ldots (i \times r^h + r^h - 1) \bmod N$. *Proof*: We prove by induction. Base case: when $h = 1$, from the construction of the topology, node i connects to nodes $i \times r \bmod N$, $(i \times r + 1) \bmod N, \ldots (i \times r + r - 1) \bmod N$.

Induction case: assuming that when $h = n$, node i in its raw n-hop paths reaches r^n continuously numbered nodes $i \times r^n \bmod N$, $(i \times r^n + 1) \bmod N, \ldots, (i \times r^n + r^n - 1) \bmod N$. Consider the case when $h = n+1$. Extending from node $i \times r^n \bmod N$ one more hop can reach r nodes $((i \times r^n \bmod N) \times r) \bmod N$ (which is node $i \times r^{n+1} \bmod N$), $(i \times r^{n+1} + 1) \bmod N, \ldots, (i \times r^{n+1} + r - 1) \bmod N$. Extending from node $(i \times r^n + 1) \bmod N$ one more hop can reach r nodes $((i \times r^n + 1) \bmod N) \times r) \bmod N$ (which is node $(i \times r^{n+1} + r) \bmod N$), $(i \times r^{n+1} + r + 1) \bmod N, \ldots, (i \times r^{n+1} + 2r - 1) \bmod N$. Hence, the r^n continuously numbered nodes $i \times r^n \bmod N, \ldots, (i \times r^n + r^n - 1) \bmod N$ from the raw n-hop paths are extended into r^{n+1} continuously numbered nodes starting from node $i \times r^{n+1} \bmod N$. The lemma is also true when $h = n+1$. □

Built upon Lemma 6, we can derive the bound for the diameter of a GDBG topology .

Lemma 7 (GDBG Diameter): The diameter of $GDBG(N,r)$ is no more than D such that $r^D \geq N$.

Proof: See [26, 37].

Lemma 1 shows that the optimal diameter for a $DRN(N,r)$ is the smallest integer D_{opt} such that $1 + r + \cdots + r^{D_{opt}} \geq N$. In practice, only the Kautz graph is known to achieve the bound for some very specific values of N and r. Lemma 7 indicates that for a vast majority of values of N and r, the diameter of $GDBG(N,r)$ is optimal because in most cases hen $1 + r + \cdots + r^{D_{opt}} \geq N$, $r^{D_{opt}} \geq N$. Consider the specific case in which $r = 20$. Table 3.1 shows the theoretical optimal diameters and the diameters achieved by GDBGs for 8420 configurations ($r = 20$, and $N = 2, \ldots, 8421$). It can be seen from the table that among the 8420 networks, 7974 GDBG networks (94.7%) (N = 2–20, 22–400, 422–8000) achieve the optimal diameter. GDBG's diameter is one more than optimal in fewer than 5.3% of the cases.

Lemma 8 (average k-shortest path length): In $GDBG(N,r)$, for k-shortest path routing where $\frac{r^{H-1}}{N} < k \leq \frac{r^H}{N}$, the average k-shortest path length is near optimal.

Proof: Since $\frac{r^{H-1}}{N} < k \leq \frac{r^H}{N}$, from Lemma 6, all of the k shortest simple paths (for all pairs of SD nodes) can be realized by paths whose length is no more than H. Consider all k-shortest paths from one source node. There are at least $r-1$ 1-hop simple paths (r raw 1-hop paths with at most one loop-back path that is not counted in the simple paths). Similarly, there are at least r^2-r-1 2-hop simple paths (r^2 raw 2-hop paths with at most $r+1$ being 1-hop or 0-hop paths). In general, there are at least $r^i - r^{i-1} - \cdots - r - 1$ i-hop simple p aths for all $i = 1,2,\ldots,H-1$. The rest of the paths will all be H-hop paths since all of the k paths are no more than H hops. The number of such paths is $k(N-1) - \sum_{i=1}^{H-1}(r^i - r^{i-1} - \cdots - r - 1)$. Let $RR = k(N-1) - \sum_{i=1}^{H-1}(r^i - r^{i-1} - \cdots - r - 1)$, the average k-shortest path length is at most

$$
\begin{aligned}
&\frac{\sum_{i=1}^{H-1} j(r^j - r^{j-1} - \cdots - r - 1) + H \times RR}{k(N-1)} \\
&\leq \frac{\sum_{i=1}^{H-1} j \times r^j + H \times RR}{k(N-1)} \\
&= \frac{\sum_{i=1}^{H-1} j \times r^j + H \times (k(N-1) - \sum_{j=1}^{H-1} r^j)}{k(N-1)} + \frac{H \times \sum_{i=1}^{H-1}(r^{i-1} + \cdots + r + 1)}{k(N-1)} \\
&= LK(N,r,k) + \frac{H \times \sum_{i=1}^{H-1}(r^{i-1} + \cdots + r + 1)}{k(N-1)} \\
&= LK(N,r,k) + \frac{H \times \sum_{i=1}^{H-1} \frac{r^i - 1}{r-1}}{k(N-1)} \\
&\leq LK(N,r,k) + \frac{H \times (r^H - 1)}{(r-1)^2 \times k \times (N-1)}
\end{aligned}
$$

Since $\frac{r^H-1}{N} < k \leq \frac{r^H}{N}$, the additional term $\frac{H \times (r^H-1)}{(r-1)^2 \times k \times (N-1)}$ is roughly in between $\frac{H}{(r-1)^2}$ and $\frac{H}{r-1}$. In practical networks, H is usually a small number such as 2, 3, or 4 while r is a reasonably large number such as 20, 30, or 40. Hence, the average k-shortest path length in $GDBG(N,r)$ is very close to the optimal. □

The following lemma states that a GDBG distributes its short paths evenly among all SD pairs.

Lemma 9: For a given H, there are at least $\lfloor \frac{r^H}{N} \rfloor$ paths between any SD pair in $GDBG(N,r)$ whose length is no more than H.

Table 3.1: Number of nodes for a given degree ($r = 20$) and a given diameter

Diameter	Number of Nodes (Optimal)	Number of Nodes (GDBG)
1	2 to 21	2 to 20
2	22 to 421	21 to 400
3	422 to 8421	400 to 8000

Proof: From Lemma 6, the raw H-hop paths from an arbitrary source node reach r^H continuously numbered nodes. Thus, each node will be the destination of a raw H-hop path $\lfloor \frac{r^H}{N} \rfloor$ times. Hence, For a given H, there are at least $\lfloor \frac{r^H}{N} \rfloor$ paths between any SD pair in $GDBG(N,r)$ whose length is no more than H. □

Lemma 3 states that the upper bound of the smallest number of paths whose length is no more than H between any SD pair in any $DRG(N,r)$ is $\lfloor \frac{r+r^2+\cdots+r^H}{N-1} \rfloor$. Lemma 9 shows that $GDBG(N,r)$ guarantees to provide $\lfloor \frac{r^H}{N} \rfloor$ paths that are no more than H hops for each SD pair. When r is the degree in practical networks (e.g. r = 16, 24, 36), $\lfloor \frac{r^H}{N} \rfloor$ is very close to $\frac{r+r^2+\cdots+r^H}{N-1}$ (asymptotically the same). Thus, a GDBG distributes its short paths evenly among all SD pairs.

Lemma 10 (load balance): For $GDBG(N,r)$ with ALLPATH(H), each SD pair will at least have $\lfloor \frac{r^H}{N} \rfloor$ paths, and the maximum link load for all-to-all traffic is no more than

$$\frac{(H)r^H - \frac{r^H-1}{r-1}}{r-1} \times \frac{1}{\lfloor \frac{r^H}{N} \rfloor} \quad .$$

Proof: By the definition of ALLPATH(H), H is the smallest integer such that $r^H \geq N$. Since all raw H-hop paths are used in ALLPATH(H), from Lemma 9, each SD pair will have at least $\lfloor \frac{r^H}{N} \rfloor$ paths. Clearly, all of these paths can be derived from raw paths that are H hops or less by removing loop-backs. Hence, the number of links used in the raw paths are strictly no less than those used in the actual simple path: we can bound the maximum link load by counting all potential load contributions from the links in raw paths that are no more than H hops away.

In order to count the number of times each link is used in all 2-hop raw paths, consider 2-hop raw paths from each node. There are r first hop-paths, each will be used r times since there are r second hop branches. There are r^2 second hop links. From the construction of $DRG(N,r)$, all links in each hop are evenly distributed among all links in the network: all first hop links in the 2-hop raw paths are even distributed among all links in the network; all second hop links in the 2-hop raw paths are evenly distributed among all links in the network. The example for $DRG(6,2)$ is shown in the Figure 3.2. There are a total of $N \times r$ first hop links evenly distributed among all $N \times r$ links, each being used r times; and there are a total of $N \times r^2$ second hop links evenly distributed among all $N \times r$ links. Hence, to realize all 2-hop raw paths, each link is used $\frac{N \times r}{N \times r} \times r + \frac{N \times r^2}{N \times r} = 2r$ times. Following the similar logic, to realize all 3-hop raw paths, each link is used $3 \times r^2$ times; and to realize all i-hop raw paths, each link is used $i \times r^{i-1}$ times. Hence, the total number of times each link is used in all of the raw paths that are no more than H-hop is

$$1 + 2r + 3r^2 + \cdots + (H)r^{H-1} = \frac{(H)r^H - \frac{r^H-1}{r-1}}{r-1} \quad .$$

Since the number of paths between every SD pair is at least $\lfloor \frac{r^H}{N} \rfloor$, each time a link is used, the load contribution to the link is at most $\frac{1}{\lfloor \frac{r^H}{N} \rfloor}$. Hence, the link load for all-to-all traffic for any link is at most

$$\frac{(H)r^H - \frac{r^H-1}{r-1}}{r-1} \times \frac{1}{\lfloor \frac{r^H}{N} \rfloor} \quad . \qquad \square$$

the proof environment

Lemma 11: Let H be the smallest integer such that $r^H \geq N$, for a GDBG with ALLPATH(H), the maximum link load for all-to-all communication is near optimal.

Proof: Since H is the smallest integer such that $r^H \geq N$, $LK(N,r,1) \approx H-1$ for most of the network configurations. From Lemma 4, the lower bound on the maximum link load for all-to-all traffic is

$$ML(N,r) \geq \frac{LK(N,r,1) \times (N-1)}{r} \approx \frac{(H-1)(N-1)}{r}$$

.

From Lemma 10, for $GDBG(N,r)$, the maximum link load is at most

$$\frac{(H)r^H - \frac{r^H-1}{r-1}}{r-1} \times \frac{1}{\lfloor \frac{r^H}{N} \rfloor} \approx \frac{H \times N}{r-1} \quad .$$

Since $\frac{H \times N}{r-1}$ is very close to $\frac{(H-1)(N-1)}{r}$, the maximum link load for all-to-all communication on $GDBG(N,r)$ with ALLPATH(H) is near optimal. This lemma is confirmed in our numerical study (Figures 3.7a and 3.7b).
□

3.4 Empirical Comparison of Topological Properties

We empirically evaluate the topological properties for random regular graph (RRG), generalized De Bruijn graph (GDBG), Kautz graph, and the theoretical bounds (optimal). The empirical results confirm the theoretical findings in the previous section.

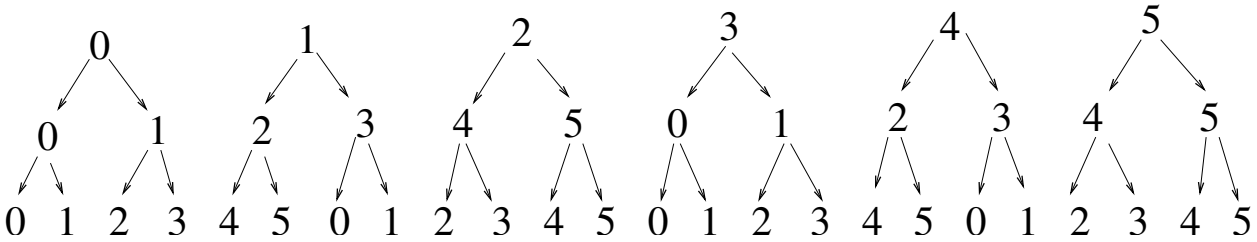
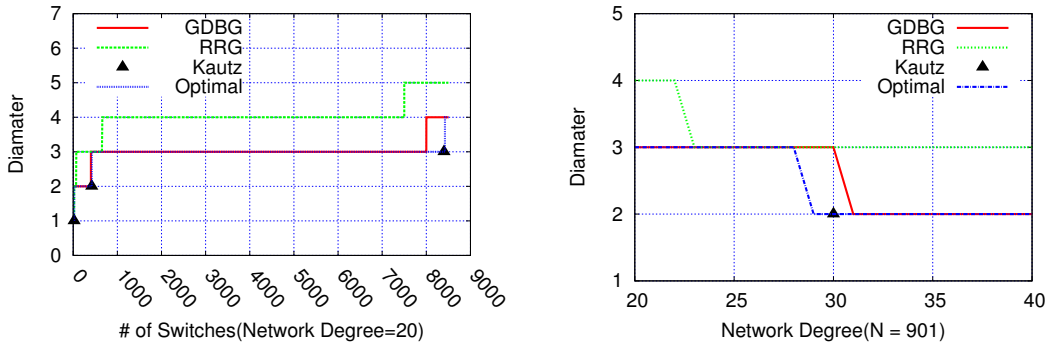


Figure 3.2: All 2-hop raw paths in $DRG(6,2)$: all first hop links are evenly distributed; and all second hop links are evenly distributed.



(a) $r = 20$ (b) $N = 901$

Figure 3.3: Change in diameter versus (a)network size and (b)network degree

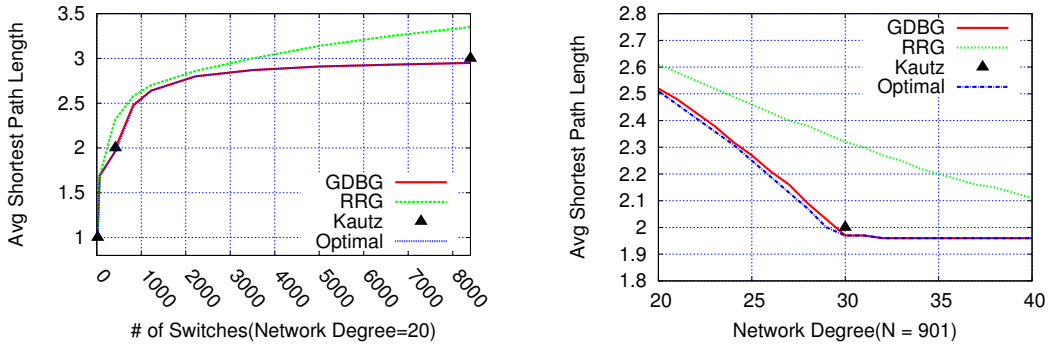
3.4.1 Diameter

Figure 3.3a shows the results for topologies with size ranging from 1 to 8500 and a fixed nodal degree ($r = 20$) while Figure 3.3b shows the results for a fixed network size ($N = 901$) but different nodal degrees. As can be seen from the figures, Kautz graphs are optimal, but only apply to a very limited number of network configurations. The diameter for an RRG is mostly 1 or 2 more than the optimal. In most of the cases when the optimal diameter is 3 (Figure 3.3a), the diameter of the respect ive RRG is 4, which is 33.3% worse than the optimal. On the other hand, GDBGs achieve optimal diameter for the majority of the network configurations. This reaffirms the conclusion in Table 3.1 that RRGs are not ideal in terms of network diameter in most cases and GDBGs are optimal in most cases.

3.4.2 Average k-shortest Path Length

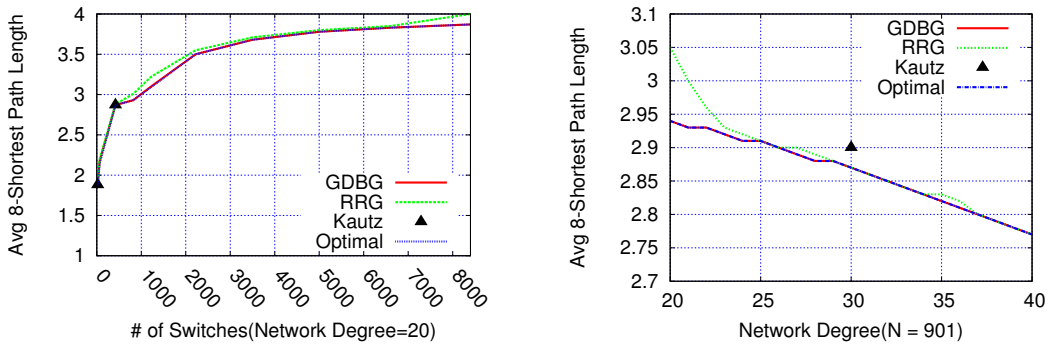
Let us now examine the average k-shortest path length. In all cases, the optimal average k-shortest path length is caculated using Lemma 2. We first show the average shortest path length ($k = 1$). Figure 3.4a shows the results for topologies with size ranging from 1 to 8500 and a nodal degree of 20 while Figure 3.4b shows the results for a fixed network size ($N = 901$) but different nodal degrees. As can be seen from the figures, depending on the network configuration, the average shortest path length for RRGs can be up to more than 15% worse than the optimal. In all of the configurations, GDBGs have very similar average shortest path length to the optimal. Kautz graphs are always optimal but only work for a few configurations.

Figure 3.5a shows the average 8-shortest path lengths for topologies with size ranging from 1 to 8500 and a fixed nodal degree ($r = 20$) while Figure 3.5b shows the results for a fixed network size ($N = 901$)



(a) $r = 20$ (b) $N = 901$

Figure 3.4: Change in average shortest path length versus (a)network size and (b)network degree

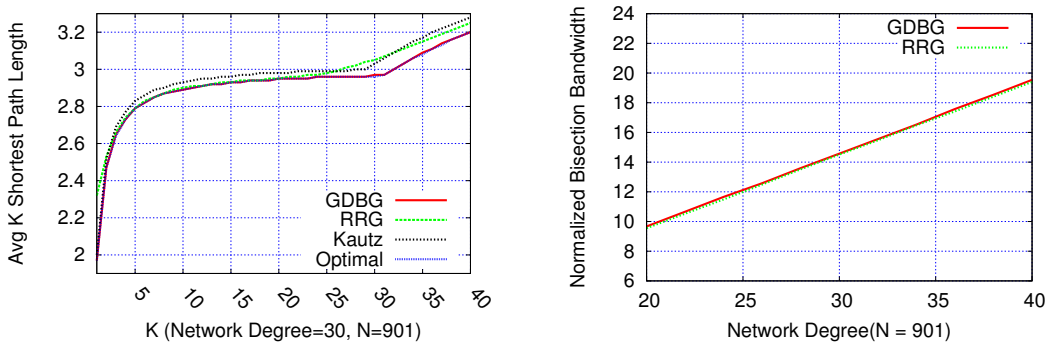


(a) $r = 20$ (b) $N = 901$

Figure 3.5: Change in average 8-shortest path length versus (a)network size and (b)network degree

but different nodal degrees. RRGs and Kautz graphs are slightly worse than the optimal (less than 3% in all cases) while GDBG virtually has an identical average 8-shortest path length as the optimal. When the number of paths used is not 1, the gap between the RRG and the optimal is small. For Kautz graphs, even though it achieves optimal average shortest path length for single path routing, when multiple paths are used, the topology is no longer optimal.

Figure 3.6a shows the average k-shortest path length for different topologies with $N = 901$, $r = 30$ and varying k. The trend in this figure is representative for other network configurations. As demonstrated in the figure, when $k = 1$, RRGs are noticeably worse than the other topologies (GDBG, Kautz, optimal) in average shortest-path length. For higher values of k, the gap among different topologies is small. Depending



(a) (b)

Figure 3.6: (a) Average k-shortest path length ($N = 901$, $r = 30$) (b) Bisection Bandwidth of RRGs vs GDBGs with increasing network degree($N = 901$)

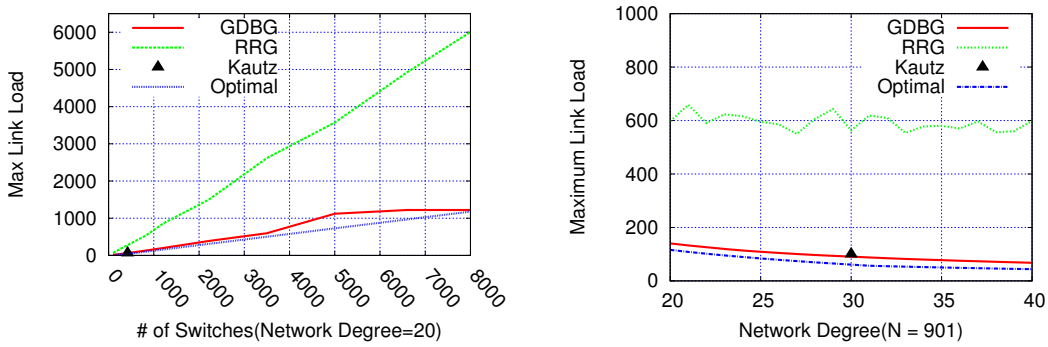
on the value of k, for some configurations, all topologies have similar average k-shortest path length; for some other configurations, RRGs have up to 5% longer average k-shortest path length than other topologies. GDBGs consistently have near optimal average k-shortest path length for all network configurations.

3.4.3 Load Balancing

In the comparison of the load balancing property, we assume ALLPATH(H) for GDBGs. For other topologies, we first compute the average number (A) of paths for each SD pair for GDBGs. All other topologies assume k-shortest path routing with $k = \lceil A \rceil$.

Figure 3.7a shows the changes in the maximum link load while increasing the network size, whereas, Figure 3.7b changes the nodal degree while keeping the number of switches fixed. The method for computing the maximum link load is described in Section 3.2.3. In both cases, there is a significant gap between maximum link loads in an RRG and the optimal case. The GDBG distributes the load much more evenly among all links in the network and therefore, reaches almost optimal maximum link load in all instances.

Figure 3.6b shows the normalized bisection bandwidth of RRGs against GDBGs for a fixed network size ($N = 901$) and different nodal degrees. The normalized bisection bandwidth is the total number of links in the minimum cut divided by $\frac{N}{2}$, the size of each partition. As can be seen in the figure, RRGs and GDBGs maintain almost identical bisection bandwidths. This is observed for all other network configurations.



(a) $r = 20$ (b) $N = 901$

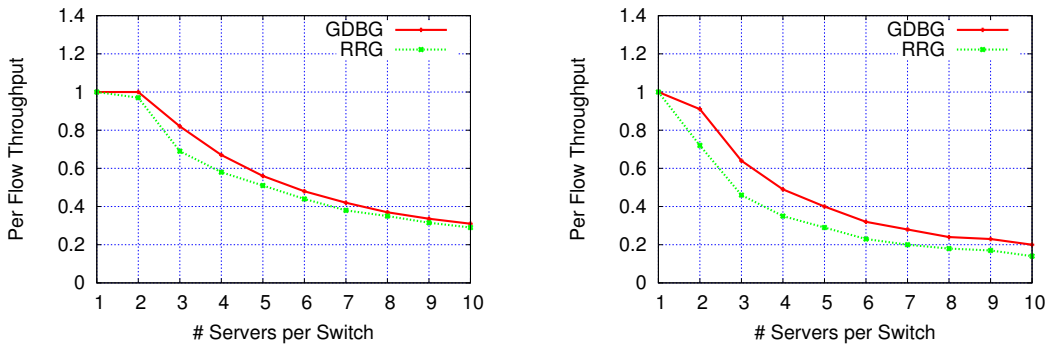
Figure 3.7: Change in maximum link load for all-to-all communication versus (a)network size and (b)network degree

3.5 GDBG Versus Related Topologies

Here we summarize the GDBG's topological properties. The key properties are the following

- The diameter of $GDBG(N,r)$ is near optimal among all $DRG(N,r)$ (Lemma 7).
- For any k, the average k-shortest path length of $GDBG(N,r)$ is near optimal (Lemma 8).
- For any value H, $GDBG(N,r)$ evenly distributes its short paths whose length is no more than H among all SD pairs (Lemma 9).
- With ALLPATH(H), $GDBG(N,r)$ achieves near perfect load balancing for most network configurations (Lemma 11).

In comparison to the Kautz graph that achieves the optimal diameter for very specific values of N and r, $GDBG(N,r)$ can be constructed for any values of N and r and achieves optimal diameter for a vast majority of network configurations. In addition, the GDBG is able to distribute short paths among all of its SD pairs evenly and achieves almost perfect load balancing with ALLPATH(H). Kautz graphs do not have these properties that are important to achieve high performance when k-shortest path routing is used. In comparison to RRGs, GDBGs are better in almost all important topological properties including diameter, average k-shortest path length, short path distribution, and load balancing.



(a) Permutation Traffic Pattern

(b) Shift Traffic Pattern

Figure 3.8: Average per flow throughput for compute node level traffic patterns $N = 150$, $r = 8$

3.6 Comparison Between RRG and GDBG

Since GDBGs are near optimal in terms of most of the topological properties discussed so far, they can be used as a benchmark to study the performance characteristics of RRGs. In this study, we used a modeling scheme as well as a cycle-accurate simulator to compare RRGs and GDBGs under different traffic conditions.

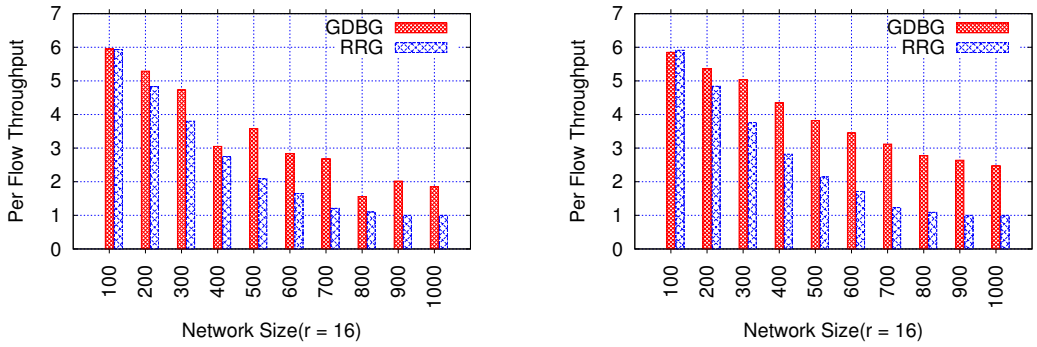
The ALLPATH(H) routing is used for GDBGs in all experiments. For RRGs, we initially planned to use the same hop-limited all-path routing. Interestingly, the experiments failed because the RRG fails to provide connectivity among all SD pairs with the same value of H as the corresponding GDBG that is of the same size and nodal degree; some SD pairs do not have any path that is no more than H hops. To use ALLPATH(H), an RRG will require one or more additional hops than a similar GDBG just in order to provide connectivity. Such a comparison would be unfair to an RRG since traffic would take longer paths. This observation itself is an indication that a GDBG is superior. To address this issue and quantitatively compare the two topologies, we compute the average number (A) of paths for all SD pairs with the ALLPATH(H) routing on the GDBG. We then compare the GDBG using ALLPATH(H) with an RRG using k-shortest path routing, where $k = \lceil A \rceil$ or $k = \lfloor A \rfloor$, whichever yields the better results. The total number of paths available for communication in an RRG is slightly more or less than that in a GDBG: the experiments slightly favor RRGs. Note that k-shortest path routing was suggested for RRGs [43].

3.6.1 Modeling Results

Two traffic patterns from LANL–FSU Throughput Indices (LFTI) [47] are used in the evaluation: the random permutation pattern and the random shift pattern. In a random shift pattern, each node i communicates with node $(i+a)$ mod *nprocs*, where *nprocs* is the number of processing nodes and a is a random value from 1 to *nprocs*. The two patterns are applied in two different ways: at the compute node level with multiple compute nodes connecting to each switch, and at the switch level. When a communication pattern is applied at the compute node level, it is assumed that the links connecting compute nodes to switches have the same speed as the links between switches. For the permutation pattern, the traffic is diverse since different nodes in one switch may communicate to different nodes at different switches. For the shift pattern, the traffic is more concentrated: all nodes in each switch communicate to nodes in one or two other switches. When the traffic patterns are at the switch level, we assume that one compute node is attached to each switch and has infinite bandwidth to and from the switch. This allows for evaluating the potential throughput between switches. The traffic is concentrated with switch level traffic patterns since each switch only communicates with another switch.

For all traffic patterns, the aggregate throughput is computed using a linear programming formulation that maximizes the minimum concurrent flow, the same metric used by Singla et al. [42]. This represents the theoretical upperbound for the aggregate throughput that any routing scheme can achieve. The linear programming formulation is solved using IBM CPLEX. Each point in the figure is the average of 20 to 200 random samples; the data collection stops when the 95% confidence interval is less than 1% of the average: with high confidence, the reported numbers are close to the theoretical average.

Figure 3.8 shows the results for compute-node level traffic pattern on RRG(150, 8) and GDBG(150, 8) with the number of compute nodes on each switch ranging from 1 to 10. The per flow throughput is normalized to the link speed: 1 means 100% of the link speed. As can be seen from the figure, when the number of nodes in each switch is 1, both RRGs and GDBGs can support the full speed of each compute node for both patterns. As the number of compute nodes per switch (*persw*) increases, both cannot support full bandwidth from the compute nodes. For both patterns, GDBGs consistently perform noticeably better than RRGs. For the permutation pattern, when the number of nodes per switch is between 3 and 10, GDBGs are between 3% ($persw = 10$) and 20% ($persw = 3$) better; for shift patterns, GDBGs are better by up to 43% (achieved when $persw = 10$). The figure also shows that when the number of nodes per switch is large (e.g. $persw = 10$), RRGs have similar performance as GDBGs for permutation patterns. In this case, the



(a) Permutation Traffic Pattern

(b) Shift Traffic Pattern

Figure 3.9: Average per flow throughput for switch-level traffic patterns ($N = 100, .., 1000$ $r = 16$).

traffic pattern at the switch level is diverse as each switch communicates with many other switches. This confirms earlier results [42] that RRGs can achieve near optimal performance when traffic is diverse. On the other hand, for the more concentrated shift patterns, RRGs are less effective.

Figure 3.9 shows the results for the switch-level patterns where traffic is concentrated. In the experiments, the network size ranges from 100 to 1000 and the nodal degree is fixed at 16. In the figure, the per-flow throughput is normalized to the (between switch) link speed. Since there are multiple links between switches, the per-flow throughput is usually more than 1. For both patterns, GDBGs perform significantly better than RRGs, especially as the network grows in size. For example, on the 900-node system, the GDBG achieves a per-flow throughput of 2.64 for the shift pattern, 2.64 times that of the RRG. As the network size increases, the advantages of GDBGs manifest.

These results refine the current understanding of RRGs when Singla concluded that RRGs are near optimal in general [42]. Our results indicate that RRGs are near optimal for diverse traffic when each switch communicates with many other switches. However, for concentrated traffic where each switch only communicates with one other switch, the performance of RRGs are far from GDBGs, let alone the optimal.

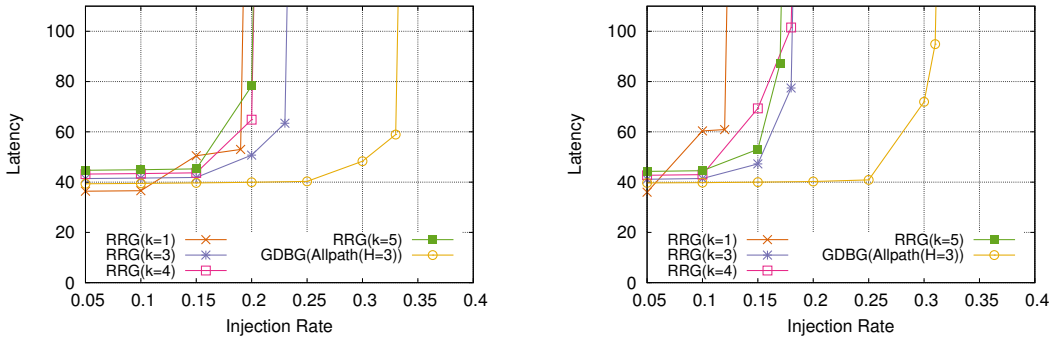
3.6.2 Simulation Results

To gain further understanding of RRGs, we also compare RRGs and GDBGs through cycle-accurate simulation. We implemented the RRG and the GDBG to pologies as well as k-shortest path routing and hop-limited all path routing in *Booksim* [27], a cycle-accurate interconnection network simu lator, which has been used to evaluate the performance of topologies and routing schemes such as the Dragonfly topology [28,33].

The experiments were carried out to investigate the performance differences between corresponding RRG and GDBG networks under different traffic conditions.

The simulation configurations are similar to those used to evaluate routing schemes on Dragonfly [28]. We assumed single-flit packets and a 2x speedup for router crossbar over network links. The latency of each network link is set to 10 cycles. To avoid deadlocks, we used up to 5 VCs with a 32-entry buffer size. For each data point, the network was warmed up for 15000 cycles, and network statistics were collected for another 15000 cycles. For both k-shortest path routing and ALLPATH(H), multiple paths are available for each SD pair. When a packet arrive s, in our implementation, one of the multiple paths is randomly selected for the p acket.

The experimental settings are similar to those in the previous section. Two traffic patterns are used in the evaluation: the random permutation pattern and the shift pattern. In our simulations, under shift pattern, each node i communicates with node $(i+p)$ mod *nprocs*, where *nprocs* is the number of processing nodes and p is the number of processing nodes connected to each switch. In other words, each processing node only sends packets to a corresponding processing node connected to the next switch. This effectively creates a switch-level shift pattern.



(a) Permutation Traffic Pattern

(b) Shift Traffic Pattern

Figure 3.10: Latency vs offered load with Allpath(3) and Kpath($k = 1,3,4,5$) ($N = 150$, $r = 8$, $p = 4$)

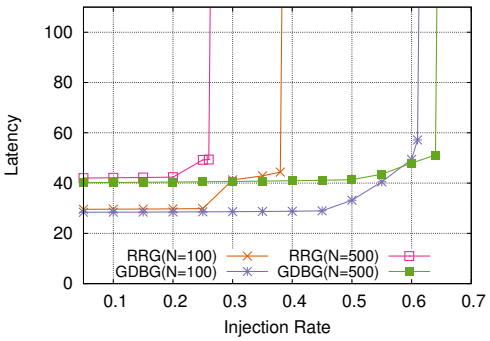
The first experiment compares the performance of k-shortest path routing on $RRG(N = 150, r = 8)$ with different values for $k = 1,3,4,5$ and Allpath(3) routing on a corresponding $GDBG(N = 150, r = 8)$. For both topologies, the number of compute nodes attached to each switch is 4 (p=4). For this network configuration, ALLPATH(3) for the GDBG provides 3.77 paths on average for each SD pair. Figure 3.10a shows the latency-throughput results for a representative random permutation traffic. The results for other

permutations have a similar trend. It is clear that a GDBG with ALLPATH(3) achieves a higher throughput performance in compare to an RRG with k-shortest path routing for any given value of k. Since ALLPATH(3) has on average 3.77 paths for each SD pair, it is expected tha t k-shortest path routing with $k = 4$ will achieve a good performance for the RR G topology. However, as can be seen in the figure, k=3 slightly outperforms k=4. Although this seems counter-intuitive, RRGs do not evenly distribute the ir short paths as discussed earlier. Hence, when $k = 4$, more longer paths are used for some SD pairs, which lowers the performance.

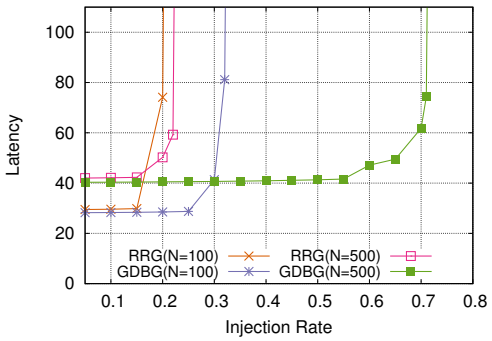
Figure 3.10b shows the results for the shift traffic. The GDBG not only achieves higher throughput performance than a corresponding R RG, but also yields lower latency under almost any offered load. This is because of the fact that GDBGs provide a much better path diversity in compare to RRGs as proven earlier. RRGs only achieve lower latency under very low load and with k=1, which is because of using a single shortest path between any source-destination pair. However, single path routing can not accomodate higher offered loads and the links get saturated much faster. This re-affirms Singla's conclusion that single path routing does not provide sufficient path diversity in RRGs [43]. A GDBG with ALLPA TH(H) is more effective than a corresponding RRG with k-shortest path routing both at low loads and high loads for different traffic patterns as shown in this experiment.

Figure 3.11, shows another set of experiments in which we investigate the effect of an increase in network size on the performance of RRGs and GDBGs. We consider two network configurations with N=100 and N=500 switches and fix the number of switch-to-switch links at r=16. For both topologies, each switch connects to 4 compute nodes. When N=100, since the network diameter is 2 for the GDBG, we use ALLPATH(2), which provides an average 2.65 paths per SD pair. For the RRG, k-shortest path routing with k=2 performs slightly better than k=3, therefore we only include k=2 results in Figure 3.11. When N=500, we use Allpath(3) for GDBGs which provides an average 8.63 paths per source-destination pair. To make the comparisons fair, we use k-shortest path routing with k=8 for RRGs.

Figure 3.11a depicts the latency-throughput results for representative random permutations on the topologies of different sizes (same permutation is compared for the corresponding GDBGs and RRGs). GDBGs with ALLPATH consistently perform better than RRGs with k-shortest path routing for both high and low loads, regardless of the network sizes. Figure 3.11b shows the results for the shift pattern, which is equivalent to a switch-level shift pattern. For both traffic patterns, when the network size and the number of paths are small, the throughput performance of the RRG is relatively closer to that of the GDBG. However, path diversity and load balancing properties of t he GDBG, create a considerable performance gain when



(a) Permutation Traffic Pattern



(b) Shift Traffic Pattern

Figure 3.11: Latency vs offered load for different network sizes ($r = 16, p = 4$)

network size increases. The s ame trend is also observed in the modeling of the throughput upper bound in Figure 3.9.

Figure 3.11b also shows that with our implementation of ALLPATH(H) and k-shortest path routing, $RRG(N = 500, r = 16)$ achieves an aggregate throughput of 0.23 per flow while $GDBG(N = 500, r = 16)$ achieves an aggregate throughput of 0.72 per flow. Here the per flow throughput is normalized to the link speed: 1 means 100% of the link speed. Since each switch connects to 4 compute nodes, $RRG(N = 500, r = 16)$ achieves a per-switch throughput of $4 \times 0.23 = 0$ .92 while $GDBG(N = 500, r = 16)$ achieves a per-switch throughput of $4 \times 0.72 = 2.88$. For the particular pattern, using the performance modeling, it is derived that the achi evable per-switch throughput for ALLPATH(3) on the GDBG is 3.84 and $k = 8$-shortest pat h routing on the RRG is 2.14. Thus, the GDBG achieves $\frac{2.88}{3.84} = 75.4\%$ of the theo retical optimum while $k = 8$-shortest path routing on the RRG only achieves $\frac{0.92}{}2.14 = 46.7\%$ of the theoretical optimum. ALLPATH(H) on the GDBG distributes traffic evenly among links and is relatively easy to exploit its capacity while k-shortest path routing on the RRG is not balanced as shown in the earlier analysis. As a result, it is more difficult to achieve high performance on RRG. Hence, although $GDBG(N = 500, r = 16)$ is $\frac{3.84}{2.14} = 1.79$ times better than $RRG(N = 500, r = 16)$ in terms of the theoretical throughput upper bound, it is $\frac{2.88}{0.92} = 3.13$ times better in the simulation.

In summary, the simulation results indicate that GDBGs with ALLPATH(H) offer be tter performance than the corresponding RRGs with k-shortest path routing both in latency at low loads and throughput at high loads. Additionally, it is more difficult to achieve high pe rformance with RRGs (in comparison with

GDBGs) in routing implementaion due to the uneven distribution of traffic in RRGs with k-shortest path routing.

3.7 Summary

The Random regular graph (RRG) has recently been proposed as an interconnect topology for future large scale data centers and HPC clusters. An RRG is a special case of *directed regular graph* (DRG) where each link is unidirectional and all nodes have the same number of incoming and outgoing links. In this work, We derive bounds for DRGs on diameter, average k-shortest path length, and load balance with k-shortest path routing and show that RRGs are not ideal for diameter and load balancing. We identify a near optimal topology for k-shortest path routing with the associated k-shortest path routing scheme: the generalized de Bruijn graph (GDBG) with the hop-limited all-path routing (ALLPATH(H)). Not only is the GDBG an effective topology for k-shortest path routing, but it can also serve as a near-optimal benchmark to evaluate other topologies when k-shortest path routing is appropriate. We show that while random regular graphs perform well for diverse traffic, they are not effective in dealing with concentrated traffic patterns such as switch-level permutation and shift patterns.

CHAPTER 4

ROUTING SCHEME DESIGN AND EVALUATION

4.1 Introduction

The Dragonfly topology features a cost-effective interconnect design and provides high aggregate bandwidth for a diverse set of traffic patterns [33]. It has been deployed in the Cray Cascade architecture and is the interconnect topology in a number of current and near-term supercomputers. In this chapter, I introduce a novel routing technique for such networks, which improves the communication performance of existing adaptive routing schemes by identifying the traffic pattern and taking smarter routing decisions [21].

One unique characteristic of the Dragonfly network is that the routing performance is very sensitive to traffic pattern. To achieve high performance, different routing schemes must be used for different traffic patterns [33]. For example, minimal routing (MIN) should be used for uniform traffic while Valiant Load Balance routing (VLB) should be used for other traffic patterns. To unify the two routing algorithms in one system, Universal Globally Adaptive Load-balanced routing (UGAL) [33] adapts the routing decision between MIN and VLB paths based on the link load information derived from queue length. The theoretical UGAL with perfect global link state information (UGAL-G), which cannot be implemented in practice, performs similarly as MIN for uniform traffic and as VLB for adversarial traffic. Various schemes that approximate the theoretical UGAL-G have been developed [28].

An adaptive routing scheme that makes routing decisions based on link loads fundamentally optimizes for network load balancing by distributing the traffic such that link loads on different links are similar. However, load balancing alone is insufficient for achieving high performance in Dragonfly. VLB can achieve load balancing for almost any traffic pattern, yet its performance for random uniform traffic is significantly lower than MIN. An adaptive routing scheme such as UGAL that makes routing decisions based solely on link/path loads has inherent limitations as the traffic pattern is not directly considered. Recent studies have shown that UGAL makes inappropriate routing decisions in various situations, which degrades its performance and that the problem is more severe with imprecise global link state information [23].

In this work, we propose traffic pattern-based adaptive routing (TPR) for Dragonfly. TPR is based on UGAL, but enhances it by explicitly incorporating a traffic pattern-based adaptation mechanism.

The proposed scheme is motivated by several observations. First, Dragonfly is developed for HPC systems; and HPC applications often have repetitive communication patterns that can easily be identified at different levels [29, 49]. Second, modern routers for HPC systems maintain an extensive number of performance counters. The statistics collected in these counters can be used to infer traffic pattern. Finally, UGAL works well under many traffic conditions, but not all [23]. Incorporating the traffic-pattern information in making routing decisions can alleviate the issues with UGAL for the traffic conditions in which it does not perform well.

TPR maintains counters for traffic in each router and infers local and global traffic condition from the counters. It has an underlying practical UGAL scheme. Based on the inferred traffic conditions, TPR identifies nine different operational regions for intra-group and inter-group communications in which different biases toward MIN and VLB paths are appropriate. In this way, TPR adapts traffic based on the current traffic pattern in addition to link loads. We performed extensive simulation with a diverse set of traffic conditions. The simulation results indicate that (1) TPR performs significantly better than its underlying practical UGAL scheme in all situations, (2) under low load, TPR (based on the practical UGAL scheme) performs better or on par with the theoretical UGAL with perfect global link state information (UGAL-G) in terms of packet latency; and (3) under high load, TPR performs comparable to UGAL-G in terms of aggregate bandwidth. We conclude that TPR is a robust and highly effective adaptive routing scheme for Dragonfly.

The rest of this chapter is structured as follows. Section 4.2 presents our proposed routing algorithm. Section 4.3 reports the results of the performance study. Finally, Section 4.4 concludes the chapter.

4.2 Traffic Pattern-based Adaptive Routing

The link load based adaptive routing mechanisms introduced in chapter 2.1.2 cannot reach the observed performance of MIN under uniform random traffic and VLB under worst-case adversarial traffic because they try to adapt to the network congestion only based on available link load information. As revealed in recent studies [23], these adaptive routing schemes suffer from the following limitations.

- Fluctuations in queue lengths due to temporary load-imbalance can result in suboptimal routing decisions made by the adaptive routing scheme, even when the overall traffic pattern remains unchanged.

- Longer queue lengths can occur for multiple reasons, including high load or the adversarial nature of the traffic. But these routing schemes treat all these situations alike by choosing the least loaded path, which is not always an optimal choice.
- The performance of UGAL (UGAL-G or UGAL-L) depends on the congestion offset (the T in Equation 2.1), which needs to be tuned empirically based on the traffic pattern. Fixing its value represents a trade-off: the routing will favor either minimal paths, which would degrade the routing performance for adversarial traffic, or non-minimal paths, which would result in low performance for uniform traffic. There does not exist a single value for T that can achieve high performance under all traffic conditions.

Considering the above limitations, our scheme is built upon the idea that assuming a UGAL-like routing scheme that selects between MIN and VLB paths, each router can observe the traffic that passes through the router and infer useful traffic pattern information to make better routing decisions. With a UGAL-based routing scheme, even when a router is not on the minimal path of a communication, it can still observe the communication, as the router can be selected as an intermediate router when a non-minimal path is used.

TPR has an underlying UGAL routing scheme. We assume it is UGAL-L in this paper. In practice, any practical UGAL implementation can be used as the underlying routing scheme for TPR. To obtain the traffic-pattern information, TPR maintains a number of counters and infers the local and global pattern information from the counters. The Cray Aries provides an extensive set of performance counters so maintaining the counters needed by our scheme is not an issue on that platform [1]. By explicitly inferring and using the traffic pattern information, custom adaptive routing mechanisms can be tailored for the observed traffic patterns. TPR improves the underlying UGAL by having nine operational regions for both intra-group and inter-group communications based on the inferred traffic-pattern information. For each operational region, either a different offset value (T) in the underlying UGAL is used to give different biases toward MIN and VLB paths, or MIN or VLB routing is directly used when appropriate. In other words, the adaptive routing mechanisms used in our scheme range from pure MIN routing to UGAL-L with different biases to pure VLB routing.

In the following, we first introduce the counters used in our routing scheme. We then discuss how the traffic pattern is inferred. Finally, we present our traffic pattern-based adaptive-routing algorithm.

4.2.1 Traffic Counters

In each router, our scheme maintains the following counters, which count the number of packets during a window of time (e.g., the last 50 cycles):

- $DestC_i$: These counters record the number of packets sent to the router i in the same group, from local compute nodes connected to this router, during the window period.
- $Port\_thr_i$: For each port i in the router, $Port\_thr_i$ records the number of through packets (packets originated from other routers in the same group) that use this port.
- $DestGrpC_i$: These counters record the number of packets sent to the group i, from local compute nodes connected to this router, during the window period.
- Thr_GrpC_i: For each destination group i, Thr_GrpC_i records the number of packets originated from other routers in the network with a destination in group i that pass through this router.

$DestC_i$ and $Port\_thr_i$ are used to infer intra-group traffic pattern while $DestGrpC_i$ and Thr_GrpC_i are use to infer inter-group traffic pattern. To support intra-group communication in Cascade, each router must have 96 $DestC_i$ counters and 30 $Port\_thr_i$. To accomodate inter-group communication, the number of $DestGrpC_i$ and Thr_GrpC_i counters per router is equal to the number of groups. These numbers of counters are not significant relative to the number of performance counters provided by the Aries router [1].

4.2.2 Inferring Intra-Group Traffic Pattern

Quantifying Intra-Group Local Traffic Pattern. Intra-group local traffic consists of packets generated from the endpoints attached to the router to other endpoints in the same group. From earlier studies of Dragonfly, it is known that the traffic is benign when it spreads evenly among all possible destination routers (uniform traffic), and adversarial when the traffic concentrates on a small number of destination routers. To make effective routing decisions, it is important to know how adversarial or how uniform the local traffic pattern is. For intra-group communication, such information can be obtained with the $DestC_i$ counter. The basic idea is that if the local traffic is uniform, the values in $DestC_i$ are similar and relatively small. If the local traffic is adversarial, the values in $DestC_i$ will exhibit a small number of spikes (implying traffic is concentrated on a small number of destination routers)whose values can be very large, depending on the injection rate. In our scheme, we quantify local traffic to each destination switch as benign or adversarial by examining $DestC_i$.

We first consider an experiment with the Booksim simulation infrastructure [28]. In this experiment we simulate one group of a Cascade machine with 96 routers each connecting to 18 traffic endpoints, to focus on intra-group communications. The routing algorithm is UGAL-L. The observation window size is 50 cycles. The traffic pattern is either uniform random (UR) or an adversarial shift pattern (ADV) in which all traffic

from endpoints on router i is sent to endpoints on router $i+1$. We observe the counter values ($DestC_1$) in router 0. The results are shown in Table 4.1. The injection rate is the number of packets generated per traffic endpoint per cycle. As can be seen from the table, for uniform traffic, the number of packets sent to each destination is much smaller than the number of packets sent to the same destination under adversarial traffic and same injection rate. Each Cascade group has 96 routers. With uniform traffic, each router should receive 1/95 of the total local traffic. On the other hand, for the worst case adversarial traffic, all of the local traffic are sent to one destination router. As such, differentiating these two types of traffic to each destination can be done by examining $\frac{DestC_i}{h}$, where h is the window size. We call this value *localimpact_intra*.

Table 4.1: The counter values for uniform and adversarial intra-group traffic ($h = 50$)

Injection rate	Pattern	$DestC_1$	$localimpact\_intra$
0.1	UR	1	0.02
	ADV	90	1.80
0.44	UR	4	0.08
	ADV	396	7.92
0.9	UR	8	0.16
	ADV	∞	∞

Assuming that at most one packet can be generated from a processing node per cycle, the value for *localimpact_intra* can range from 0 to p, where p is the number of processing nodes attached to each router. As can be seen from Table 4.1, the values for *localimpact_intra* differ significantly for the benign traffic (UR) and the adversarial traffic (ADV). For UR traffic with 0.9 injection rate (very heavy load), the *localimpact_intra* value is only 0.16 while for ADV traffic with 0.1 injection rate (not heavy load), the *localimpact_intra* value is 1.80. Hence, *localimpact_intra* is a good indicator for the locally generated traffic pattern. In our scheme, we classify the local traffic to each destination router into three types, benign, mixed, and adversarial using two threshold values $intralow_l$ and $intrahigh_l$ as follows. The intra-group local traffic to destination router i is deemed

- **benign** when $localimpact\_intra = \frac{DestC_i}{h} < intralow_l$,
- **adversarial** when $localimpact\_intra > intrahigh_l$, and
- **mixed** when $intralow_l \leq localimpact\_intra \leq intrahigh_l$.

In our experiments, we set $intralow_l = 0.8$ and $intrahigh_l = 4$. These thresholds should be tuned for each operational system.

Quantifying Intra-Group Global Traffic Pattern. Global traffic consists of packets generated by endpoints connected to other routers. Here, since we are only concerned with intra-group communication global traffic is any packet that has been generated by endpoints connected to other routers in the same group. For the global traffic pattern, the most important information for making routing decisions is how the global traffic would affect the local router. The impact of global traffic on a router can be quantified by the amount of through traffic on each port of the current router, which is the $Port\_thr_i$ counter. Table 4.2 shows the average through traffic on each port of a router under uniform random (UR) and adversarial traffic (ADV) with different loads. The simulation setting is the same as that used in Table 4.1. As can be seen from the table, there are distinguishable differences between the through traffic counts for uniform and adversarial traffic. In particular, very large $Port\_thr_i$ (e.g. > 30) is only observed for adversarial traffic with heavy loads.

Table 4.2: The average $Port\_thr_i$ values for uniform and adversarial intra-group traffic ($h = 50$)

Injection rate	Pattern	$Port\_thr_i$	$globalimpact\_intra$
0.1	UR	2.24	0.04
	ADV	5.45	0.11
0.44	UR	9.86	0.20
	ADV	33.7	0.67
0.9	UR	20.5	0.41
	ADV	∞	∞

Our scheme uses $\frac{Port\_thr_i}{h}$ to quantify the global traffic on a particular port (port i) of a router and we call this value $globalimpact\_intra$. The scheme also classifies the intra-group global traffic on the port to be benign, mixed, and adversarial using two threshold values $intralow_g$ and $intrahigh_g$. The global traffic on port i is deemed

- **benign** when $globalimpact\_intra = \frac{Port\_thr_i}{h} < intralow_g$,
- **adversarial** when $globalimpact\_intra > intrahigh_g$, and
- **mixed** when $intralow_g \leq globalimpact\_intra \leq intrahigh_g$.

In our experiments, we set $intralow_g = 0.33$ and $intrahigh_g = 0.6$. These parameters should be tuned for each particular system. As discussed earlier, an adversarial global traffic pattern as defined implies a mostly adversarial global traffic pattern with heavy load. On the other hand, benign or mixed global-traffic patterns can be interpreted as either uniform (with a higher load) or adversarial global traffic patterns (with a lower load).

4.2.3 Inferring Inter-group Traffic Pattern

Quantifying Inter-group Local Traffic Pattern. Similar to the intra-group communication, our goal is to identify the locally generated traffic, but this time we are interested in traffic which is generated from the endpoints attached to the router and is destined to routers in the other groups. Since the cost of maintaning a counter for each destination router in the whole system is prohibitive, we aggregate this information and only keep track of the packets sent to each destination group. In other words, for each locally generated packet which is destined to group i, we increment the value of the corresponding $DestGrpC_i$ counter. As discussed in Subsection 4.2.2, if the local traffic is uniform, the values in $DestGrpC_i$ are similar and relatively small, while for adversarial local traffic, we should observe a few outliers with a relatively large $DestGrpC_i$ value. Again, we quantify locally generated traffic to each destination group as benign or adversarial by examining $DestGrpC_i$.

In order to get a better understanding of the range of values for $DestGrpC_i$ this time we simulate the Edison, which is a Cray Cascade machine with 15 groups, using Booksim. Each group in Edison has 96 routers with 4 processing nodes connected to each router. This configuration is also used in our experiments to evaluate TPR. The observation windows size is 50 cycles and the routing algorithm is UGAL-L. The observed values for $DestGrpC_1$ in router 0 of group 0, under UR and adversarial traffic patterns are shown in Table 4.3.

As can be seen from the table, the trend in the counter values for the inter-group local traffic is very similar to that for the intra-group local traffic. For uniform traffic, the number of packets sent to each destination is much smaller than the number of packets sent to the same destination under adversarial traffic and same injection rate. As such, we can differentiate the traffic pattern using $\frac{DestGrpC_i}{h}$ values. We call this value *localimpact_inter*.

Table 4.3: The counter values for uniform and adversarial inter-group traffic ($h = 50$)

Injection rate	Pattern	$DestGrpC_1$	$localimpact\_inter$
0.1	UR	1	0.02
	ADV	20	0.4
0.44	UR	6	0.12
	ADV	88	1.76
0.9	UR	12	0.24
	ADV	∞	∞

Similar to the case for intra-group communication, we classify locally generated inter-group traffic into three types, benign, mixed, and adversarial using two threshold values $interlow_l$ and $interhigh_l$ as follows. The local traffic to destination group i is deemed

- **benign** when $localimpact\_inter = \frac{DestGrpC_i}{h} < interlow_l$,
- **adversarial** when $localimpact\_inter > interhigh_l$, and
- **mixed** when $interlow_l \leq localimpact\_inter \leq interhigh_l$.

In our experiments, we set $interlow_l = 0.35$ and $interhigh_l = 0.9$.

Quantifying Inter-group Global Traffic Pattern. Inter-group global traffic consists of packets generated by endpoints connected to other routers anywhere in the network. In order to quantify the impact of global traffic on a router, for each destination group i, we keep track of all packets that pass through the router and are headed towards group i. We save this information in Thr_GrpC_i counter. Table 4.4 shows the average through traffic for destination group 1 on router 0 of group 0 under uniform random (UR) and adversarial traffic (ADV) with different loads. The simulation setting is the same as that used in Table 4.3.

Table 4.4: The average Thr_GrpC_1 values for uniform and adversarial inter-group traffic ($h = 50$)

Injection rate	Pattern	Thr_GrpC_1	$globalimpact\_inter$
0.1	UR	2.3	0.05
	ADV	34.0	0.68
0.44	UR	10.1	0.20
	ADV	150.0	3.00
0.9	UR	20.7	0.41
	ADV	∞	∞

Our scheme uses $\frac{Thr\_GrpC_i}{h}$ to quantify the global traffic towards a destination group i of a router and we call this value *globalimpact_inter*. We classify the inter-group global traffic as benign, mixed, and adversarial using two threshold values $interlow_g$ and $interhigh_g$. The global traffic passing through a router towards group i is deemed

- **benign** when $globalimpact\_inter = \frac{Thr\_GrpC_i}{h} < interlow_g$,
- **adversarial** when $globalimpact\_inter > interhigh_g$, and
- **mixed** when $interlow_g \leq globalimpact\_inter \leq interhigh_g$.

In our experiments, we set $interlow_g = 0.8$ and $interhigh_g = 3.1$. These parameters can be tuned for a particular system. Similar to the intra-group case, an adversarial inter-group global traffic pattern as defined implies a mostly adversarial inter-group global traffic pattern with a heavy load. On the other hand, benign or mixed inter-group global-traffic patterns can be interpreted as either uniform (with a higher load) or adversarial inter-group global traffic patterns (with a lower load).

In the next sub-section, we will discuss how the local and global traffic pattern information of intra- and inter-group communication can be used to improve routing effectiveness.

4.2.4 The Routing Algorithm

Let p be the packet to be routed. We will denote $src(p)$ as the source router for the packet, $dst(p)$ as the destination router, $dstgrp(p)$ as the destination group and $MINP(p)$ as the local output port in the source router for the MIN path. Let $loc\_intra(p)$ be the *localimpact_intra* factor at $src(p)$ for $dst(p)$: $loc\_intra(p) = \frac{DestC_{dst(p)}}{h}$. Similarly, let $loc\_inter(p)$ be the *localimpact_inter* factor at $src(p)$ for destination group $dstgrp(p)$: $loc\_inter(p) = \frac{DestGrpC_{dstgrp(p)}}{h}$. Let $glo\_intra(p)$ be the *globalimpact_intra* factor at $src(p)$ for p, which is the *globalimpact_intra* for the local output port for the MIN path: $glo\_intra(p) = \frac{Port\_thr_{MINP(p)}}{h}$. Also, let $glo\_inter(p)$ be the *globalimpact_inter* factor at $src(p)$ for destination group $dstgrp(p)$: $glo\_inter(p) = \frac{Thr\_GrpC_{dstgrp(p)}}{h}$. The routing decision is made at $src(p)$ that has all of the needed information.

For each packet p, the source router first determines if the destination router is in the same group (intra-group or inter-group communication). Then it will decide the path based on the current traffic pattern and the link loads. For intra-group communication, the current intra-group traffic condition is represented by $loc\_intra(p)$ and $glo\_intra(p)$. Similarly, for inter-group communication, $loc\_inter(p)$ and $glo\_inter(p)$ are used to identify the traffic pattern. Based on the values of the parameters, the routing algorithm operates in nine operating regions, for intra- and inter-group communication, each with a different routing mechanism.

TPR uses a practical UGAL as its underlying routing scheme. We assume the underlying routing for TPR is UGAL-L in this paper. UGAL-L is an effective adaptive routing algorithm whose performance is largely influenced by the offset value. With a large offset value, the routing favors minimal paths [33]. By using a range of offset values, progressively decreasing, UGAL-L progressively biases towards the VLB paths. We will use the notion *UGAL(T)* to denote UGAL-L with offset T, where T may be positive or negative. By using the traffic pattern information derived from the counters, TPR identifies different operating regions. For each region, TPR either directly uses MIN or VLB when appropriate or uses UGAL-L with a different offset value. The VLB routing used in our scheme, which we call path-length oblivious VLB, is slightly

different from the VLB used in UGAL: VLB used in UGAL favors shorter VLB paths as discussed in Section 2.1.2 while VLB used in TPR is oblivious to path length. In TPR, VLB is only used for the extreme adversarial condition. In such a condition, preferring shorter non-minimal paths does not perform as well as path-length oblivious VLB. Note that it is straight-forward to replace UGAL-L by another practical UGAL-based scheme as the underlying routing for TPR.

Table 4.5 shows the nine operational regions and their routing mechanisms for intra-group communication and Table 4.6 shows the same for inter-group communication. In general, for the region when both local and global traffic are identified as benign, MIN or UGAL with a large offset (strongly favoring MIN paths) should be used; conversely when both local and global traffic indicates adversarial traffic, VLB should be used. In between, UGAL-L with tuned offset values should be used. Consider for example, the third row and third column of Table 4.5 where both $loc\_intra(p)$ and $glo\_intra(p)$ are benign: this identifies largely uniform traffic or very low load adversarial traffic when MIN achieves high performance. In this case, TPR either selects MIN or uses UGAL-L with a very large offset (64) that strongly favors MIN paths. The mechanism to differentiate these two routing schemes in the region is to check whether the combined local and global traffic exceeds a threshold. Specifically, when $loc\_intra(p) + glo\_intra(p) < thr$, MIN is used; otherwise $UGAL(64)$ is used. Similar mechanism is also used for the case when $loc\_intra(p)$ is benign and $glo\_intra(p)$ is mixed. When both local traffic and global traffic are adversarial, pure VLB routing is used. From the tables, the following can also be observed. First, for the same local traffic, as the global traffic becomes more adversarial, the algorithm biases more towards a non-minimal path with a decreasing offset value for UGAL-L. Second, for the same global traffic, as the local traffic becomes more adversarial, the algorithm biases more towards a non-minimal path. The adaptive mechanism for inter-group communication is set in a similar manner. The absolute offset values for inter-group communication are larger since inter-group communication has longer path lengths. Note that the offset values in the tables are tuned for the Edison configuration (used in our experiments) and they should be tuned for each operational system that uses TPR.

The effectiveness of the proposed scheme can be further refined by using smaller categories of local and global traffic and tailoring the offset value for each category. However, as will be shown in the next section, our algorithm is already more effective, sometimes significantly, than the traditional UGAL-L scheme. Note that like UGAL-L, our scheme only uses local information to make routing decisions. This demonstrates that

Table 4.5: Operation regions and the corresponding routing mechanisms for intra-group packet p

		loc_intra(p)		
		benign	mixed	adv.
glo_intra(p)	benign	MIN/UGAL(64)	UGAL(-4)	UGAL(-48)
	mixed	MIN/UGAL(64)	UGAL(-20)	UGAL(-64)
	adv.	UGAL(48)	UGAL(-40)	VLB

Table 4.6: Operational regions and the corresponding routing mechanisms for inter-group packet p

		loc_inter(p)		
		benign	mixed	adv.
glo_inter(p)	benign	MIN	UGAL(-30)	UGAL(-80)
	mixed	UGAL(900)	UGAL(-50)	UGAL(-95)
	adv.	UGAL(500)	UGAL(-70)	VLB

using traffic pattern to filter situations and facilitate different adaptive mechanisms is an effective approach for improving routing performance.

4.3 Performance Study

4.3.1 Simulation Methodology

The proposed scheme has been implemented in *BookSim* [28], an interconnection network simulator, which is used to evaluate the routing algorithms in this study. The experiments were designed to investigate the intra- and inter-group performance of different routing schemes. For intra-group communication we used a single group of a Cray Cascade machine with 96 routers and 18 traffic generating end-points connected to each router. For inter-group communication we used a 15-group Cray Cascade architecture [17], which is similar to the Edison machine, as described in Section 2.1.2.

The simulation configurations are similar to Jiang et al.'s [28]. We assumed single-flit packets and a 2x speedup for router crossbar over network links. The latency of each network link is set to 10 cycles. To avoid deadlocks, we used 4 VCs with a 32-entry buffer size. For each data point, the network was warmed up for 3,000 cycles, and network statistics were collected for 10,000 cycles.

4.3.2 Routing Schemes

We compare the performance of 5 routing schemes under various traffic patterns. This includes MIN, VLB, UGAL-L (UGAL with local information), UGAL-G (the theoretical UGAL with global information),

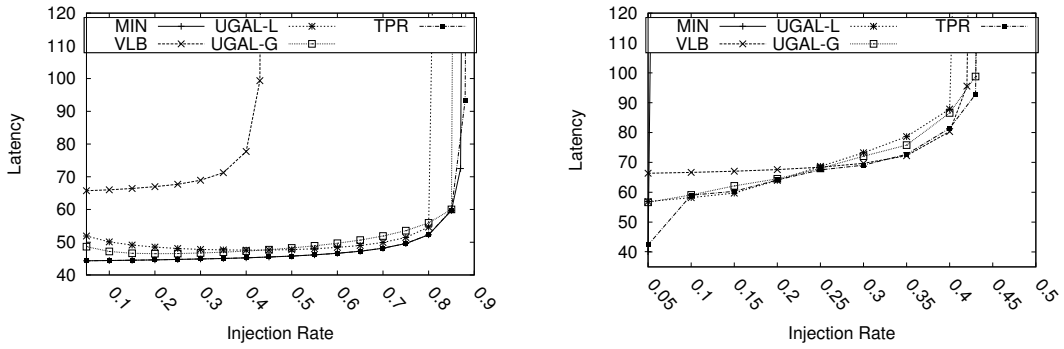
and TPR with the underlying UGAL-L. In TPR the windows size for the counters is 50 cycles. All adaptive routing decisions are made at the source router after a packet is injected into the network.

4.3.3 Traffic Patterns

Intra- and inter-group communications are evaluated separately under four different families of traffic patterns. To examine the extreme case performance, we used a uniform-random traffic pattern (UR) and an adversarial shift traffic pattern (ADV). With UR, the probability of sending a packet to each destination is equal regardless of the scope of traffic (intra-group or inter-group).Whereas ADV traffic is defined differently for intra- and inter-group communications. In a intra-group ADV, each node connected to a given router i sends all of its traffic to nodes connected to router $i+1$, while under inter-group ADV all nodes connected to a given router in group i send all of their traffic to nodes connected to router $i+96$ to stress the global links connecting the two neighbor groups. In addition, two families of mixed traffic patterns are considered by combining UR and ADV traffic patterns at processing node or router levels. In node-level combined traffic patterns, each packet is sent as either UR traffic or ADV traffic with a certain probability. Thus, the traffic from each router contains both UR and ADV components. We will use the notation NLC_URADV(UR%, ADV%) to represent the node-level combined traffic patterns, where UR% is the percentage of UR traffic and ADV% is the percentage of ADV traffic. For example, in NLC_URADV(20,80), each node sends UR traffic with 20% probability and ADV traffic with 80% probability. The other mixed traffic pattern is the router-level combined traffic pattern. In this case, all processing nodes connected to a specific router have the same traffic pattern (either UR or ADV). However, nodes at different routers may have different traffic patterns. We will use the notation RLC_URADV(UR%, ADV%) to represent the router-level combined traffic patterns. For example, in RLC_URADV(20,80), 20% of the routers generate UR traffic and 80% of the routers generate ADV traffic.

4.3.4 Results for Intra-group Communication

Uniform Random Traffic. Figure 4.1a shows the latency-throughput results for UR traffic. As expected, MIN achieves the best throughput and the lowest latency among all evaluated routing schemes due to the load-balanced nature of the UR traffic pattern. VLB achieves approximately half of the minimal routing throughput and with a higher latency because with VLB, each packet uses twice the network resources as MIN. UGAL-L achieves 92% of MIN throughput under UR, albeit at a higher latency, which stems from the fact that it routes 10–38% of the traffic non-minimally (38% under low load). This is due to the

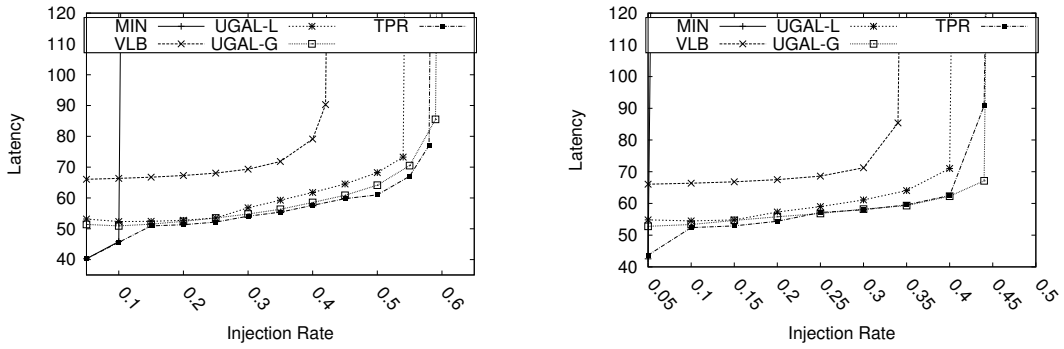


(a) (b)

Figure 4.1: Latency vs offered load under intra-group (a)Uniform Random traffic and (b)Adversarial shift traffic

fluctuation of the queue length even with the UR traffic, which Won et al. also observed [46]. Even with the perfect global information, in comparison to MIN, UGAL-G still has noticeable higher latency at low load and achieves 98% of MIN throughput. By incorporating traffic-pattern based adaptation mechanism, TPR minimizes the inappropriate decisions to route non-minimally by distinguishing UR traffic pattern early on. For all traffic injection rates, TPR routes no more than 1% of the traffic through VLB paths and achieves almost identical throughput and latency as MIN.

Adversarial Traffic. Results for ADV traffic are shown in Figure 4.1b. MIN fails to balance the load for this traffic, and, as a result, the network saturates at a very low injection rate. In contrast, VLB distributes the traffic evenly among all intermediate nodes and achieves 90% of the theoretically achievable throughput. UGAL-L starts by sending 28% of the traffic through minimal paths under low load and gradually decreases this amount to as low as 1% at the saturation point. UGAL-G reaches roughly 10% higher throughput than UGAL-L by exploiting the global information and keeps sending about 12% of the traffic minimally even at high loads. TPR uses minimal routing under very low load which explains very low latency at 0.05 injection rate. Then as the load increases, *localimpact* and *globalimpact* increase as well. Therefore, TPR uses smaller offsets while using UGAL-L to balance the adversarial load and eventually switches to our path-oblivious VLB at very high load. This results in a lower latency for TPR at high load in compare to UGAL-L and UGAL-G, as well as a higher throughput than UGAL-L and traditional VLB, which is mainly due to the adoption of path oblivious VLB.

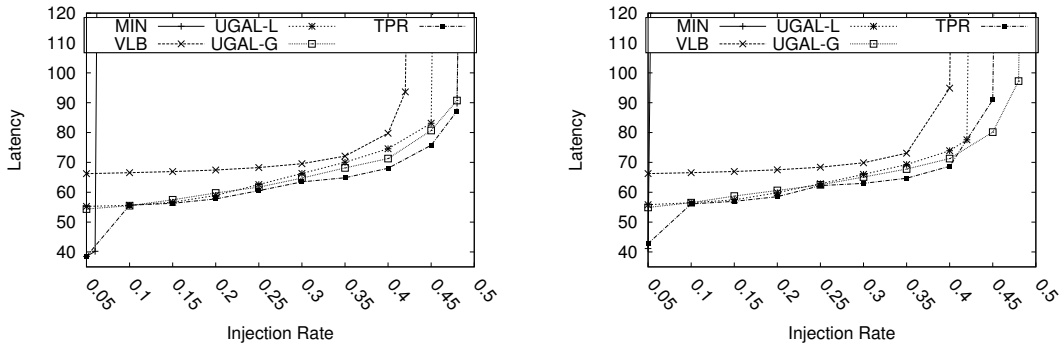


(a) (b)

Figure 4.2: Latency vs offered load under intra-group (a)NLC_URADV(50, 50) and (b)SLC_URADV(50, 50)

Node-level Combined Traffic. Figures 4.2a, 4.3a, and 4.4a show the results for three types of node-level combined traffic: NLC_URADV(50,50), NLC_URADV(20,80), and NLC_URADV(80,20), respectively. Note that NLC_URADV(20,80) means that the traffic is 20% uniform and 80% adversarial. These simulation studies are designed to test the performance of the proposed routing scheme under mixed and more complex traffic patterns. As can be seen from the figures, in all cases, TPR outperforms UGAL-L in terms of throughput and latency. Also TPR achieves up to 25% decrease in latency in compare to UGAL-G under low loads in all different cases. In Figure 4.4a, under low load (injection rate < 0.2), TPR routes at least 99% of the traffic using minimal paths, in contrast to UGAL-L, which sends less than 71% using minimal paths. This explains the lower latency. As the load increases and minimal paths get congested, TPR swiftly shifts towards using UGAL-L with smaller offset values. This results in a 4% increase in throughput and 9% improvement in latency in comparison to UGAL-L. Figures 4.3a and 4.4a also verify that TPR correctly distinguishes the local and global traffic patterns and chooses the appropriate paths to route the traffic accordingly. This creates a noticeable gain in terms of latency, even relative to UGAL-G (up to 25% improvement). TPR also achieves about 98% of the throughput performance of UGAL-G.

Router-level Combined Traffic. For all traffic patterns presented so far, every router in the network has the same behavior. With router-level combined traffic (RLC), traffic from different routers may have different characteristics. The network is logically partitioned into two interleaving parts with each part having a different traffic pattern. This creates an imbalance between locally generated traffic and through



(a) (b)

Figure 4.3: Latency vs offered load under intra-group (a)NLC_URADV(20, 80) and (b)SLC_URADV(20, 80)

traffic observed at each router, which helps us to understand if our routing scheme can handle cases where different parts of the network behave differently.

Figures 4.2b, 4.3b, and 4.4b show the results for three types of router-level combined traffic: RLC_URADV(50,50), RLC_URADV(20,80), and RLC_URADV(80,20), respectively. For RLC, a router will see the local traffic pattern (e.g. UR) to be different from the global traffic pattern (e.g. ADV). Under such a condition, TPR is able to make appropriate routing decisions by correctly identifying the local and global traffic impacts as shown in all three figures. As shown in Figure 4.3b, except when injection rate = 0.05, global traffic is known to be **mixed** or **adversarial**. Therefore, the routing decision can be taken with regards to the local traffic impact. If the generated traffic is uniform (20% of the routers), a large offset value will be used to prefer minimal paths, whereas for other routers which are generating adversarial traffic, a very small offset value is selected. Regardless of the traffic combination, TPR always performs better than UGAL-L (and very close to UGAL-G) by identifying the traffic pattern at each router and routing traffic accordingly. This demonstrates that traffic pattern-based routing can be effective even when local traffic is very different from global traffic.

4.3.5 Results for Inter-group Communication

Uniform Random Traffic. Under uniform inter-group traffic, MIN still achieves the best performance in terms of latency and throughput, as shown in Figure 4.5a, while UGAL-G acheieves the same throughput performance as MIN with up to 5% higher latency at low load. VLB only achieves about 42% of the

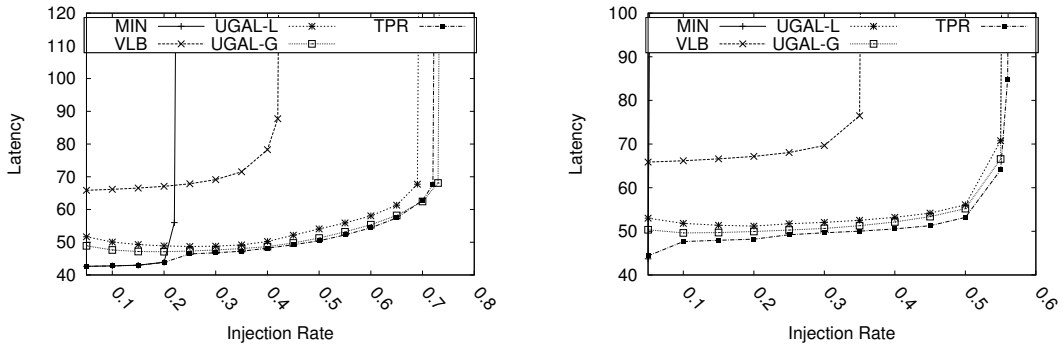
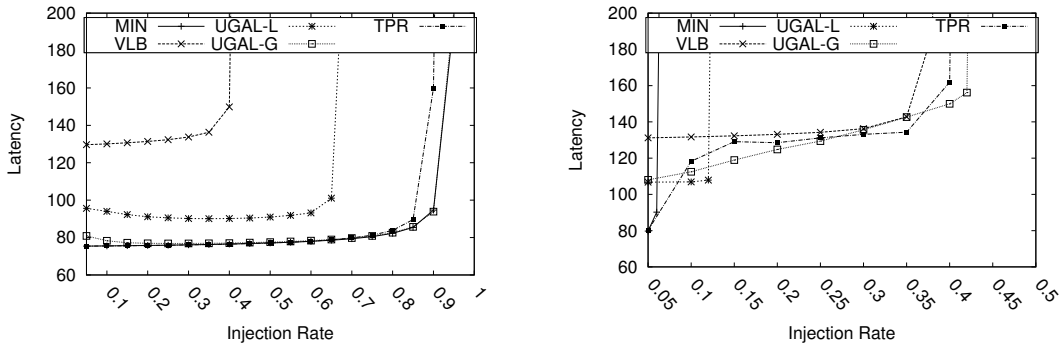


Figure 4.4: Latency vs offered load under intra-group (a)NLC_URADV(80, 20) and (b)SLC_URADV(80, 20)

throughput of MIN under uniform traffic which is consistent with the theoretical maximum throughput possible(43%) for VLB routing on Edison. Unlike Figure 4.1a, UGAL-L only achieves 70% of the throughput performance of MIN, which verifies that as the number of hops increase in the network, UGAL-L's performance drops due to the inaccurate local queue length information. Also, UGAL-L incurs up to 25% higher latency in comapre to MIN under low load. TPR with its underlying UGAL-L performs significantly better than UGAL-L at both low load and high load, achieving lower latency than UGAL-G at low load and 95% throughput as MIN and UGAL-G at high load. Note that although TPR uses MIN routing when both $local\_inter(p)$ and $glo\_inter(p)$ are benign, TPR does not always use MIN routing even for the uniform traffic pattern since both local and global traffic for a router may be classified as mixed or adversarial for a short period of time.

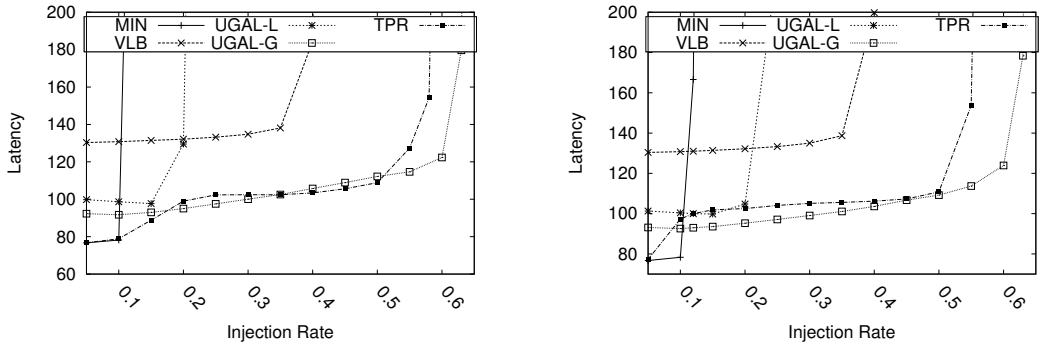
Adversarial Traffic. As discussed in Section 2.1.2, MIN is unable to balance the load under ADV traffic and a few global links get saturated at low injection rates. On the other hand, VLB distributes the traffic evenly among global links and achieves 88% of possible throughput under ADV as shown in Figure 4.5b. Interestingly, UGAL-G achieves the best throughput among all routing schemes by sending up to 16% of the traffic through minimal paths even at the saturation point. Unlike UGAL-G, UGAL-L lacks the global link state information and can only serve packets at a very low injection rate(12%). This is because of the fact that the minimal global links are generally 1 or 2 hops away from the source router, and the source router is only able to sense the congestion on these links after the back-pressure hits the local ports, which is too late.



(a) (b)

Figure 4.5: Latency vs offered load under inter-group (a)Uniform Random traffic and (b)Adversarial shift traffic

TPR uses minimal routing under very low load which explains very low latency at the 0.05 injection rate. Then as the load increases, *localimpact_inter* and *globalimpact_inter* increase as well. Unlike the case for intra-group communications, here the values for *localimpact_inter* are closer and even some times overlapping for high load UR and low load ADV. As a result, TPR is not able to out-perform UGAL-G in latency which was possible under ADV intra-group traffic. Nonetheless, TPR significantly out-performs UGAL-L and performs very similar to UGAL-G for the inter-group ADV traffic pattern.



(a) (b)

Figure 4.6: Latency vs offered load under inter-group (a)NLC_URADV(50, 50) and (b)RLC_URADV(50, 50)

Node-level Combined Traffic. Figures 4.6a, 4.8a, and 4.7a show the results for three types of node-level combined traffic: NLC_URADV(50,50), NLC_URADV(20,80), and NLC_URADV(80,20), respectively. In all cases, TPR outperforms UGAL-L in terms of throughput and latency by a large margin. This verifies the fact that in a large dragonfly network, UGAL-L with only local queue length information is not sufficient to achieve a good performance. In contrast, TPR achieves up to 25% decrease in latency in compare to UGAL-L under low loads in all different cases, while also improving the througput by a large margin.

Here also UGAL-G performs better than all other routing schemes in terms of throughput by using the instantaeous global link load information, however TPR achieves at least 90% of UGAL-G's througput with better latency under low load. Note that for inter-group communications, the gap between TPR and UGAL-G has widened due to the fact that TPR is using UGAL-L as its base, which performs poorly under all inter-group communications.

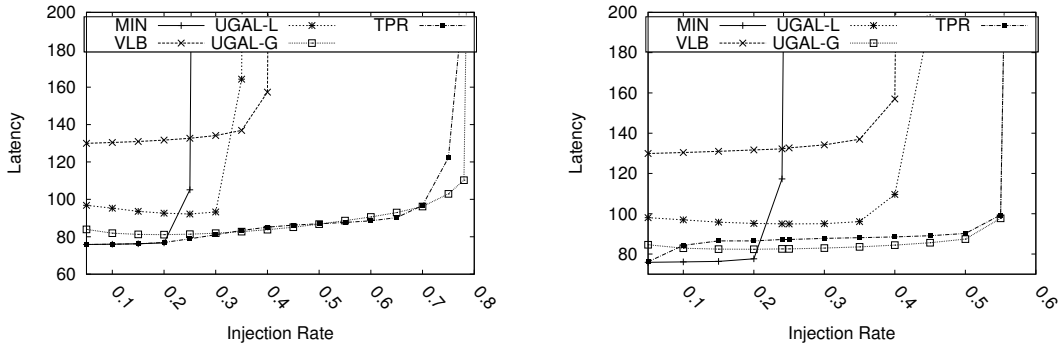
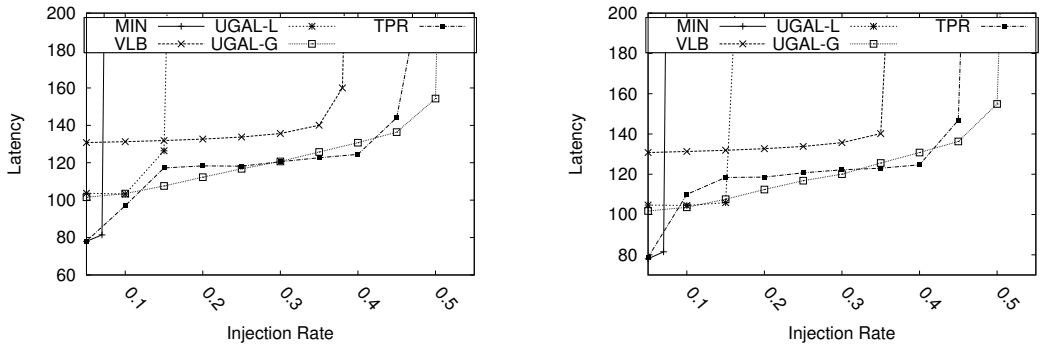


Figure 4.7: Latency vs offered load under inter-group (a)NLC_URADV(80, 20) and (b)RLC_URADV(80, 20)

Router-level Combined Traffic. With router-level combined traffic (RLC), where traffic from different routers may have different characteristics, TPR and UGAL-G still perform much better than other routing schemes as shown in Figures 4.6b, 4.8b, and 4.7b. For NLC_URADV(20, 80), at very low load, TPR identifies local and global traffic as benign and achieves very low latency similar to MIN by using 99% minimal paths. However, as the load increases, TPR categorizes the local and global traffic as mixed and uses a UGAL-L offset which is biased towards using more non-minimal paths. Therefore, TPR is able to overcome global link congestion that prevents default UGAL-L from achieving better throughput perfor-



(a) (b)

Figure 4.8: Latency vs offered load under inter-group (a)NLC_URADV(20, 80) and (b)RLC_URADV(20, 80)

mance, while also using enough minimal paths to achieve lower latency in compare to VLB. At high load TPR reaches up to 88% of the throughput performance of UGAL-G.

4.4 Summary

Existing adaptive routing schemes for the Dragonfly topology make adaptive decisions based on link loads and do not consider the traffic pattern on the network that plays an important role in effective routing in this topology. In this work, we proposed a traffic-pattern-based adaptive routing scheme and showed that by incorporating the traffic-pattern based adaptive scheme, a more effective adaptive routing scheme for intra-group and inter-group communication in Dragonfly can be obtained.

CHAPTER 5

LEVERAGING USE OF SDN TECHNOLOGY ON HPC ENVIRONMENTS

5.1 Introduction

The standardization of OpenFlow [22] stimulates the development of Software Defined Networking (SDN). OpenFlow-style SDN has shown great promises and has been deployed in campus networks, data centers, and wide-area networks. OpenFlow-style SDN originates from the Internet-scale communication infrastructure and is designed for scale. At the same time, it provides dynamic per-flow resource management using the global network view, which is significantly more flexible than traditional networks. There is a significant interest in the HPC community to adopt the SDN technology, and in this chapter I introduce a new SDN-enabled resource management scheme for HPC systems, which can compete and even outperform the current expensive and complex adaptive routing schemes on such systems [19]. It is known that SDN has different forms such as network function virtualization. This chapter focuses on OpenFlow-style SDN; and the term SDN in this chapter will refer to OpenFlow-style SDN. The term *SDN routing* refers to the routing (resource management) in SDN networks that is based on the global network view.

Although SDN has shown great promises, one major feature of SDN that is of concerns to the HPC community is that it does not have the adaptive routing capability. In an SDN network, packets in a "flow" follow the same path: SDN networks do not allow packets to adapt their routes at intermediate routers based on the network condition as in adaptive routing. Moreover, adding the adaptive routing capability may not be an option for SDN since doing so will significantly increase the network complexity. Yet, in the HPC domain, adaptive routing is widely deployed: local adaptive routing where per packet adaptive decisions are made based on the information local to each router is currently used on fat-trees [32] and low dimensional tori [6, 14] while global adaptive routing where per packet adaptive decisions are made based on estimated global network information is used in the Dragonfly networks [17, 28, 33]. Both local and global adaptive routing schemes allow the path for a packet to adapt at intermediate routers based on the traffic condition. Hence, adaptive routing has finer control over packets than SDN. This gives rise to the question whether

SDN can achieve the performance that HPC systems expect with adaptive routing. The answer to this question will affect the adoption of SDN in the HPC environment.

In this work, I consider Dragonfly networks, develop techniques to apply SDN technology on such networks, and compare SDN performance with that of adaptive routing. I select Dragonfly in this study for two main reasons. First, the Dragonfly topology is the current generation HPC interconnect topology that is deployed in current supercomputers such as Cori [2] and Trinity [8]. It features a cost-effective interconnect design and supports high aggregate throughput capacity at a lower cost in comparison to other alternatives such as fat-trees [33]. Second, effective routing on Dragonfly is challenging [33]. To achieve high performance in the Dragonfly topology, different routing schemes must be used for different traffic patterns. In particular, minimal routing (MIN) is suited to 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 [33] that adapts the routing decision for each packet between MIN and VLB paths based on the traffic condition. By using the Dragonfly network as the underlying topology, I address the challenges for applying SDN on the current generation HPC networks and obtain results that are relevant to the current HPC systems.

My results indicate that adaptive routing is in general a powerful method that yields high performance for practical HPC workloads due to its ability to react instantly to the network traffic condition. The results also indicate that using the global network view, it is possible for SDN to compete with or even out-perform adaptive routing by allocating network resources more effectively. The study also highlights some challenges for effectively applying SDN on Dragonfly or in HPC domain in general. The challenges include more accurately estimating the traffic demand, more efficient methods for computing resource allocation based on the global network view, and better support for rate allocation and network reconfiguration.

The rest of this chapter is organized as follows. In Section 5.2 I discuss the characteristics of HPC workloads, and some background information about adaptive routing in compare to SDN routing. Section 5.3 describes the techniques to apply SDN in Dragonfly networks. Section 5.4 presents the results of our evaluation. Finally, in Section 5.5 the chapter is concluded with a summary.

5.2 Background

5.2.1 HPC Communication Characteristics

To effectively apply SDN techniques, the system must be able to determine the traffic demand in the system [4]. SDN often distinguishes between *elephant* flows that are bandwidth-bound large flows and *mice* flows that are latency-bound small flows. When the system can identify such flows, SDN routing can significantly improve the communication performance by using custom routing for such flows [4, 5].

In HPC systems, the traffic demand can be identified at different levels. For example, for many HPC applications with regular communication patterns, the communication demands or elephant flows of an application are relatively easy to identify using compiler or runtime analysis [29, 49]. Moreover, elephant flows in an HPC system in general change slowly: the changes happen mostly when a new job starts running or when a job completes. Hence, I will assume that elephant flows that will be routed through SDN routing are known a priori, and investigate under such a condition, whether SDN can compete with adaptive routing.

5.2.2 Adaptive Routing Versus SDN Routing

Adaptive routing such as UGAL on Dragonfly allows the interconnection network to route packets based on the current traffic conditions. From the perspective of resource allocation, one strength of adaptive routing is that it optimizes the resource usage for any traffic condition. However, adaptive routing makes routing decisions based on either local information or imprecise global information in addition to having other constraints. Although it has a fine-grained control over the packet, adaptive routing does not attempt to optimize for global optimality. For example, for the UGAL on Dragonfly that is discussed in this chapter, although all MIN and VLB paths are available to route each packet in theory, only a small number (2 in the original proposal [33]) of MIN and VLB paths are actually considered for each packet. This limits adaptivity that can be achieved. In addition, as discussed in [17], the precise global network state information cannot be maintained; and some approximation of UGAL-G is used in practice. Due to these restrictions, even though UGAL has a fine-grained control over packets, it does not achieve globally optimal resource utilization.

SDN networks support dynamic per-flow resource management and routing using the global network view. While SDN routing is significantly more flexible than the deterministic single path routing that is widely deployed in the current Internet and data centers, as well as in InfiniBand-based HPC clusters, SDN routing requires all packets in each *flow* to follow the same path, and thus, does not have the ability to control the route for each packet at the router level to adapt based on the traffic condition: OpenFlow-style SDN

does not have adaptive routing (local or global) capability where packets in a flow may follow different paths. The strength of SDN is that it can manage network resources using the global network view, which potentially achieves globally optimal network resource allocation.

Both adaptive routing and SDN routing have strengths and weaknesses. Table 5.1 contrasts the two mechanisms for network control. On the one hand, adaptive routing does not require the knowledge of traffic pattern, automatically optimizes for any traffic condition, but cannot achieve global optimality in terms of network resource allocation. On the other hand, SDN routing requires the knowledge of the traffic demand, and cannot effectively deal with dynamically changing traffic condition. But with sufficient information, SDN can potentially achieve global optimality by using the global network view. It is thus interesting to see whether SDN can compete with adaptive routing in the HPC environment. In this work, I consider options for applying SDN techniques on the current generation HPC interconnects with the Dragonfly topology, and for evaluating the feasibility for SDN routing to achieve comparable or better performance than adaptive routing.

Table 5.1: Comparison of adaptive routing and SDN routing

	adaptive routing	SDN
requires knowledge of traffic pattern	No	Yes
handles dynamically changing traffic pattern	Yes	No
global optimality	Not possible	Possible

5.3 Applying SDN Technology on Dragonfly

As discussed earlier, the Dragonfly topology relies heavily on its adaptive routing scheme to achieve high communication performance. Without the adaptive routing capability, it is challenging for SDN to achieve high performance on Dragonfly. In the following, I will discuss options to apply SDN technology on Dragonfly, that is, how to use the global network view for managing network resources to achieve high performance. Both single-path routing and multi-path routing schemes are considered. In the discussion, I will assume (1) that a flow is a communication from a source node to a destination node, (2) that the set of elephant flows to be routed through SDN routing is known a priori, and (3) that other mice flows are handled

by a default routing mechanism. Since other flows do not communicate much data, how they are handled does not significantly affect the overall communication performance.

5.3.1 Single Path Routing (SDN-single)

Using the global network state information to optimize single path routing performance in SDN network has been proposed in various contexts such as data centers [4] and InfiniBand clusters [36]. The basic idea is to place each flow that must be routed through SDN routing on a path that minimizes the network contention [4,36]. Such techniques can be directly extended to the Dragonfly topology. I consider a straightforward application of the routing scheme in [36] for the fat-tree topology to Dragonfly. Specifically, given the set of flows that must be routed through SDN routing, the paths for the flows are computed as follows. Each flow is assigned a weight of 1; and each link in the Dragonfly network is initialized with a usage value of 0. The flows to be routed are considered one by one. For each flow, the least congested path, which is defined as the path whose maximum link usage value among all links on the path is the smallest is computed and selected. For paths with the same maximum link usage value, the shorter path is selected. Once the path for a flow is selected, the flow weight is added to the usage value of each link along the path; and the process continues until paths are selected for all of the flows. The term *SDN-single* denotes this technique.

5.3.2 Multi-path Routing

While SDN single path routing scheme is often more effective than other single path routing schemes as it uses the global network information [4,36], constraining the number of paths for each flow to a single path limits the performance, resulting in worse performance than multi-path routing schemes such as UGAL. To understand the relative strengths of adaptive routing and SDN routing, I also consider SDN routing with multiple paths for each flow.

Given the set of flows to be routed through SDN routing, to use multi-path routing in the SDN environment, two issues must be addressed. First, for each flow, what paths can be used to route traffic? Second, how to distribute the load among the paths? To have a fair comparison between SDN routing and UGAL, I assume that the set of paths for SDN for each flow is the same as that in UGAL, that is, both MIN paths and VLB paths for a flow are considered by SDN routing. For the second issue, finding a good load distribution with the MIN and VLB paths for SDN networks is challenging. Here, I consider three methods to obtain effective load distribution, which are named *SDN-ugal*, *SDN-optimal*, and *SDN-model*.

SDN-ugal. Since UGAL is considered an effective routing scheme on Dragonfly, in the first approach, SDN emulates UGAL by using the same load distribution as UGAL. For a given set of flows to be routed with SDN, the load distribution among paths is obtained by first simulating the network operations using UGAL and collecting the load distribution statistics. This method will be called *SDN-ugal*. Note that this scheme is mainly used as a reference point for comparison. Essentially, with *SDN-ugal*, SDN emulates the traffic distribution of UGAL at the source node. In this case, SDN routing is different from UGAL because UGAL's load distribution is the result of real-time adaptation of the network state while in SDN, loads are assigned to a path with a probability without considering the network traffic condition. As such, it is expected that *SDN-ugal* will result in lower performance than UGAL. However, the performance difference between UGAL and *SDN-ugal* will give a good indication of the effectiveness of adaptive routing on Dragonfly. If the performance of *SDN-ugal* can be close to that of UGAL, then the complexity of adaptive routing may not be necessary. In other words, if one can determine a good resource allocation scheme for SDN, SDN can be competitive to UGAL.

SDN-optimal. In the second method, I formulate the load distribution problem as an optimization problem using linear programming (LP): given the set of flows to be routed by SDN, how to allocate rate for each path so that the pattern achieves maximum aggregate throughput under max-min fairness with the constraint that only MIN and VLB paths can be used to carry traffic for each flow. Note that max-min fairness is commonly assumed in the modeling of network performance and is the base for many network rate allocation and resource management schemes. My LP formulation is based on a formulation for multi-commodity flows given by Nace et al. [41], which assumes that all possible paths can be used to route traffic (multi-commodity flow). Although my formulation is more scalable and practical as the paths considered in this model are limited only to MIN and VLB paths, it is still not sufficiently scalable to compute rate allocation for large networks. Nevertheless, the solution to the LP program gives the rate allocation for all flows and all paths that maximizes the throughput for the pattern under max-min fairness, and can be used as a performance upper bound for SDN schemes introduced in this paper. Next, I will first introduce notations and then describe the LP formulation.

Let A be a set. $|A|$ is the size of the set. Let a Dragonfly network be represented as a graph $G = (V, E)$, where V is the set of nodes and E is the set of links in the network. $V = PE \cup S$ contains two types of nodes. PE is the set of compute nodes; and S is the set of switches. The nodes are numbered from 0 to $|V| - 1$. For each link $e \in E$, C_e is the link capacity. Let $s \in PE$ and $d \in PE$ be two compute nodes. A flow from s to

d is denoted as (s,d). A traffic pattern F is a set of flows to be considered. The traffic in a flow is carried over a set of paths for the flow. Each path p is represented as a set of links. For each flow, our SDN routing considers all MIN paths and all VLB paths just like UGAL. For a flow (s,d), $P_{s,d}^{MIN}$ is the set of all MIN paths for the flow; $P_{s,d}^{VLB}$ is the set of all VLB paths. Let $e \in E$ be a link. If a path p uses a link e, we say that $e \in p$. For a flow (s,d), $P_{s,d}^{MIN}(e)$ is the set of MIN paths that use link e; $P_{s,d}^{VLB}(e)$ is the set of VLB paths that use link e; $P_{s,d}(e)$ is the set of all paths for flow (s,d) that use link e. $P_{s,d}(e) = P_{s,d}^{MIN}(e) \cup P_{s,d}^{VLB}(e)$. Table 5.2 summarizes our notation.

Table 5.2: Notation used in the model

$G = (V,E)$	the topology with node set V and edge set E
C_e, $e \in E$	link capacity
(s,d)	a flow from s to d
F	the set of flows to be routed
$P_{s,d}$	the set of all MIN and VLB paths for (s,d)
$P_{s,d}^{MIN}$	the set of MIN paths for (s,d)
$P_{s,d}^{VLB}$	the set of VLB paths for (s,d)
$P_{s,d}(e)$	$\{p \mid e \in p \text{ and } p \in P_{s,d}\}$
$P_{s,d}^{MIN}(e)$	$\{p \mid e \in p \text{ and } p \in P_{s,d}^{MIN}\}$
$P_{s,d}^{VLB}(e)$	$\{p \mid e \in p \text{ and } p \in P_{s,d}^{VLB}\}$
$\lambda_{s,d}$	the rate assigned to (s,d)
$x_{s,d}^{p}$	rate allocated to path p for (s,d)
L_k	set of flows with allocated rate at or before iteration k

Given the set of flows to be routed, the optimal rate allocation considers all MIN and VLB paths for each flow and allocates a rate to each path such that the aggregate throughput is maximized under max-min fairness. The algorithm to compute the rate allocation, which is shown in Figure 2.18, is an extension of the algorithm for multi-commodity flow model in [41]. In the algorithm, k is the iteration number; and L_k is the set of flows that have been assigned rate at or before iteration k; $\lambda_{s,d}$ is the rate assigned to flow (s,d). The algorithm iteratively solves a linear programming problem OPT_k. In each iteration, the maximum rate that is possible for all flows under consideration is computed by solving OPT_k. After the LP problem is solved, the most congested flows, which are the flows whose rates are saturated at this iteration, are identified and their rates are recorded (Lines 11 to 15). The details about how to identify the most congested flows can be found in [41]. When the rates for all flows are assigned, the algorithm will come out of the loop at Line

1 Set $k = 0$ and $L_0 = \phi$

2 Set $\lambda_{s,d} = 0, \forall (s,d) \in F$;

3 **while** $L_k \neq F$ **do**

4 Set k = k+1; $L_k = L_{k-1}$;

5 Solve the Linear Programming problem OPT_k:

6 Maximize α

7 Subject to:

8 $\alpha - (\sum_{p \in P_{s,d}} x^p_{s,d}) \leq 0, \ \forall (s,d) \in F - L_{k-1}$ (1)

9 $\sum_{p \in P_{s,d}} x^p_{s,d} = \lambda_{s,d}, \ \forall (s,d) \in L_{k-1}$ (2)

10 $\sum_{p \in P_{s,d}(e),(s,d) \in F} x^p_{s,d} \leq C_e, \forall e \in E$ (3)

11 **For** each binding constraint among LP_k Constraints (1)

12 Let (s,d) be the corresponding flow ;

13 $\lambda_{s,d} = \alpha$;

14 $L_k = L_k \cup \{(s,d)\}$;

15 **end**

16 **end**

17 output the $x^p_{s,d}$ for all $(s,d) \in F$.

Figure 5.1: Algorithm for computing optimal rate allocation for a set of flows with max-min fairness

16. The solution of the last LP problem contains the rate allocated to each path for each flow ($x^p_{s,d}$) that can achieve the optimal rate allocation for the whole pattern under max-min fairness.

In the linear program formulation OPT_k, for each flow (s,d), a variable $x^p_{s,d}$ is assigned to each possible path $P_{s,d} = P^{MIN}_{s,d} \cup P^{VLB}_{s,d}$, which includes both MIN paths and VLB paths. After the final iteration of the algorithm, these variables provide rate allocation for individual paths while the set λ provides rate allocation for all flows. $\sum_{p \in P_{s,d}} \{x^p_{s,d}\}$ is the rate allocated for flow (s,d). Constraints (1) at Line 8 ensure that at any iteration, the rates for all flows yet to be allocated are no less than the solution flow rate of that iteration. Constraints (2) at Line 9 assign the rate for flows whose rates have been determined in the earlier iterations. $\sum_{p \in P_{s,d}(e),(s,d) \in F} x^p_{s,d}$ is the accumulated rate on link e; and Constraints (3) at Line 10 are link capacity constraints that ensure that the rate on each link is no more than its capacity.

This algorithm computes the optimal rate allocation for a set of flows with max-min fairness assuming

that only MIN and VLB paths can be used to carry traffic for a flow. The linear programming formulation in Lines 5 to 10, however, can require a large number of variables since each MIN and VLB path will require a variable: the number of variables in the formulation is $O(|F| \times num\ of\ paths\ per\ flow)$. This prevents the algorithm from being used to compute rate allocation for reasonably large problems.

SDN-model. Since SDN-optimal cannot be used to compute max-min fairness rate allocation for large problems, I propose an efficient model, called *SDN-model* that can be applied to compute rate allocation for large problems. *SDN-model* focuses on how the traffic should be split among MIN paths and VLB paths, and assumes that different rates can be allocated to different MIN paths while same rate must be allocated to all VLB paths, and computes the rate distribution that maximizes the overall throughput for the pattern under max-min fairness with the stated condition.

The algorithm for computing the rate allocation for *SDN-model* is the same as that in Figure 5.1 except that the LP formulation in Lines 6 to 10 is replaced by the LP formulation $model_k$ in Figure 5.2. In this model, for each flow (s,d), a variable $x^p_{s,d}$ is assigned to each MIN path $p \in P^{MIN}_{s,d}$. In addition, another variable $x^{VLB}_{s,d}$ is assigned for all VLB paths, that is, each of the VLB paths is assumed to have the same rate $x^{VLB}_{s,d}$. At the final iteration of the algorithm, these variables provide rate allocation for individual paths while the set λ provides rate allocation for all flows. $\sum_{p \in P^{MIN}_{s,d}} \{x^p_{s,d}\} + |P^{VLB}_{s,d}| \times x^{VLB}_{s,d}$ is the rate allocated for flow (s,d). Constraints (1) in the LP problem ensure that at any iteration, the rates for all flows yet to be allocated are no less than the solution flow rate of that iteration. Constraints (2) assign the rate for flows whose rates have been determined in the earlier iterations. $\sum_{p \in P^{MIN}_{s,d}(e),(s,d) \in F} x^p_{s,d} + \sum_{P^{VLB}_{s,d}(e) \neq \phi,(s,d) \in F} |P^{VLB}_{s,d}(e)| \times x^{VLB}_{s,d}$ is the total rate allocated over link e; and Constraints (3) are link capacity constraints that ensure that the rate on each link is no more than its capacity.

1 Maximize α

2 Subject to:

3 $\alpha - (\sum_{p \in P^{MIN}_{s,d}} x^p_{s,d} + |P^{VLB}_{s,d}| \times x^{VLB}_{s,d}) \leq 0, \qquad \forall (s,d) \in F - L_{k-1}$ (1)

4 $(\sum_{p \in P^{MIN}_{s,d}} x^p_{s,d} + |P^{VLB}_{s,d}| \times x^{VLB}_{s,d}) = \lambda_{s,d}, \ \forall (s,d) \in L_{k-1}$ (2)

5 $\sum_{p \in P^{MIN}_{s,d}(e),(s,d) \in F} x^p_{s,d} + \sum_{P^{VLB}_{s,d}(e) \neq \phi,(s,d) \in F} |P^{VLB}_{s,d}(e)| \times x^{VLB}_{s,d} \leq C_e, \forall e \in E$ (3)

Figure 5.2: The LP formulation $model_k$ for *SDN-model*

This formulation only uses one variable for all VLB paths. Hence, for Dragonfly topologies that only have one MIN path between each pair of compute nodes, each flow only has 2 variables in this model, one for the MIN paths and one for all VLB paths. As such, the model is a very efficient model in terms of finding solutions and can be applied to solve large problems with tens of thousands or even hundreds of thousands of flows. Notice that with the model constraints, this model does not give the optimal rate allocation. However, as will be seen in the performance study section, this model can perform on par with UGAL on certain traffic conditions when all flows remain active continuously.

5.4 Experiments

I performed experiments to study the effectiveness of SDN on Dragonfly and to compare SDN with adaptive routing. The experiments are done using SST/macro [3], an open-source coarse-grained packet-level simulator for large parallel high-performance applications and machines. The experiments were designed to investigate the performance of different single-path and multi-path routing schemes discussed in earlier sections, under various synthetic and real application workloads, on small and large instances of the Dragonfly architecture.

I utilize built-in functionality in SST/macro including the Dragonfly topologies, MIN, VLB, and UGAL routing. To compare SDN with UGAL, I implement SDN-single, SDN-ugal, SDN-optimal, and SDN-model in SST/macro. For SDN-ugal, I instrument SST code to collect and output path usage statistics with UGAL routing. The paths and their usage statistics are then used as inputs in the SDN-ugal simulation. For SDN-optimal and SDN-model, a separate program is developed that takes the traffic pattern and the network topology as inputs, runs the algorithm in Figure 5.1 to compute rate allocation by generating LP formulations and using IBM CPLEX solver to solve the LP formulations, and produces an output file with paths and rate allocation information. In the simulation of SDN-ugal, SDN-optimal, and SDN-model, I assume no rate regulation in the communication although the information is available to do so: rate allocation information on different paths is only used to determine the probability that a path is used when a packet arrives, but not used to limit the rate of a flow. Note that communication rate regulation has not been deployed in production supercomputers.

Throughout the simulations, I set the injection, ejection, link and crossbar bandwidth equal to 4.7GB/s and the maximum packet size is set to 1500B. Also the injection and ejection latency are set to $1\mu s$ whereas

the link latency is set to 20ns. The total memory bandwidth is set to 32GB/s and the memory latency was set to 15ns. Finally, the buffer size is set to 64KB in all experiments.

5.4.1 Topology and Routing

Two instances of the Dragonfly topology are used in the experiments to study the effects of the proposed schemes under small and larger topologies. The small Dragonfly used in the experiments accommodates 64 processing nodes, distributed in 8 groups. Therefore, the Dragonfly topology parameters for the small networks are (p=2, a=4, h=2, g=8). In contrasts, for larger networks, we use a 1024 node Dragonfly topology (p=4, a=8, h=4, g=32). In all network instances, intra-group switches have all-to-all connectivity. For both topologies, the number of MIN paths between two compute nodes in different groups is one.

Six routing schemes are compared in the experiments: minimal routing(MIN), UGAL-L, SDN single-path routing (SDN-single), SDN with UGAL rate allocation (SDN-ugal), SDN with the optimal rate allocation for max-min fairness (SDN-optimal), and SDN with practical rate allocation model (SDN-model).

5.4.2 Benchmarks

SST/macro allows using offline mode simulation by replaying DUMPI traces. I use two types of traces in our experiments: synthetic traces and benchmark/application traces. Two types of synthetic DUMPI traces are used, one for random permutation patterns and the other for shift patterns. In the synthetic traces, I align the timestamps such that all nodes start communication at the same time; and each rank sends and receives 400MB data in one MPI message. For the random permutation pattern, the random pattern is first decided; and the communication is performed based on the pattern. Each shift pattern has a parameter i, which means that each compute node s sends to node $(s+i)$ *mod* N and receives from $(s-i)$ *mod* N, where N is the total number of processing nodes in the network. As discussed in [33], the shift pattern is the adversarial traffic on Dragonfly that leads to heavily congested global links when minimal routing is utilized.

Benchmark/application traces used in the experiments are from the DOE design-forward website [1] and the NAS benchmark suite. All of the traces are publicly available. The traces I have used are NAS benchmarks IS, CG, FT, SP, and MG and a number of DOE applications/miniapps Crystal Router (CR), Fill Boundary (FB), CMC, CNS and BigFFT. I have mostly selected communication-intensive applications to highlight the differences between routing schemes in terms of communication time.

[1] http://portal.nersc.gov/project/CAL/designforward.htm

For synthetic traces, all flows are elephant flows. For application traces, I define elephant flows to be flows that communicate more than 100KB data in the whole application. Elephant flows are routed through SDN routing and packets in other flows communicate using the default MIN routing in SST.

5.4.3 Results for Synthetic Traces

The performance results for each workload are reported based on the total simulated communication time in seconds. For synthetic traffic patterns I use 5 randomly generated permutation workloads as well as 5 shift workloads, on 64-rank and 1024-rank Dragonfly topologies. The synthetic traces represent the situation where SDN has accurate knowledge of the traffic condition since the assumption in the SDN rate allocation matches the communication in the experiments.

Table 5.3 shows the results for the five random permutation patterns on the 1024-node Dragonfly. There are notable observations from this table. First, the SDN single path routing (SDN-single) is noticeably more efficient than the MIN routing, which basically is the shortest path routing in this topology. This observation is in-line with other studies that indicate that SDN can improve single path routing performance [36]. Second, while SDN-single significantly improves over MIN, its performance is much worse than all of the multi-path routing schemes. This indicates that single-path routing is insufficient for Dragonfly to achieve high performance even with SDN; and multi-path routing is essential for the performance on Dragonfly. Third, comparing the performance of UGAL and SDN-ugal, both have the same rate allocation. However, the communication time with the two schemes differ significantly, up to 44.9% for *Perm*1. This shows that rate allocation alone is insufficient for high performance, dynamically adjusting the routes for packets in adaptive routing is effective in improving communication performance. Finally, SDN-optimal consistently achieves higher performance than UGAL for all five patterns while SDN-model is more effective than UGAL on average. This demonstrates that when SDN has the knowledge of communication pattern, it potentially can use the global network view to obtain a more effective resource allocation scheme and achieve higher performance than adaptive routing.

Table 5.4 shows the results for the five arbitrarily selected shift patterns, shift-32, shift-50, shift-230, shift-513, and shift-790. The general observation is similar to the results for the permutation communications: SDN-single very significantly improve the performance of MIN, but still much worse than all of the multi-path routing schemes. SDN-ugal still performs significantly worse than UGAL. SDN-optimal still achieves the highest performance for all of the traffic. The only major difference is that SDN-model is on average 8.9% worse than UGAL for the shift patterns (it was on average 6.6% better for the permutation

Table 5.3: Communication time (in seconds) for different routing schemes for synthetic random permutation patterns on 1024-node Dragonfly

	Perm1	Perm2	Perm3	Perm4	Perm5	Ave.
MIN	435.7	481.9	448.4	522.8	522.8	482.3
SDN-single	416.5	418.4	347.3	424.4	423.1	405.9
UGAL	234.0	247.5	249.0	236.6	243.7	242.2
SDN-ugal	339.3	352.1	300.7	302.6	318.7	322.7
SDN-optimal	201.2	198.9	205.2	209.3	202.7	203.6
SDN-model	221.6	219.2	221.0	251.1	223.0	227.2

pattern). This is likely due to the fact that UGAL was designed to effectively deal with shift patterns [33]. Overall, it is still possible to obtain a better resource allocation since SDN (SDN-optimal) out-performs adaptive routing.

Table 5.4: Communication time (in seconds) for different routing schemes for synthetic shift patterns on 1024-node Dragonfly

	Sh-32	Sh-50	Sh-230	Sh-513	Sh-790	Ave.
MIN	2788	1568	2265	2701	1917	2248
SDN-single	602	506	593	436	380	503
UGAL	317	291	296	270	289	292
SDN-ugal	374	355	329	349	330	347
SDN-optimal	244	251	263	244	251	250
SDN-model	335	333	298	323	302	318

Tables 5.5 and 5.6 show the results on the smaller 64-node Dragonfly. The trend is similar to that in the larger 1024-node network except that the performance differences for different routing schemes are smaller since in the smaller network, the routing choice is very limited. One distinguished observation in the 64-node cases is that SDN-optimal no longer achieves the best performance for all patterns. Although SDN-optimal uses the theoretical "optimal" rate allocation for max-min fairness for the traffic patterns and is more likely to be a more effective rate allocation as demonstrated in Tables 5.3 and 5.4, there are multiple factors that can contribute to its non-optimal performance. First, the model used to compute rate allocation only considers bandwidth allocation. In the experiments, SST/macro includes other practical parameters such as link latency and packet injection/ejection latency that can also affect the communication performance: the theoretical optimal may not be optimal in the experiments. Second, the optimal rate allocation assumes max-min fairness while practical communication schemes such as UGAL do not always honor max-min fairness. Third, SDN-optimal is only optimal when rate allocation is enforced. In the experiments, we

do not regulate rate for each path. Instead, rate allocation is approximated with per path rate distribution. Nonetheless, for the cases in the experiments, SDN-optimal still achieves the best average performance for both permutation and shift patterns, which indicates that its rate distribution is more effective than others for the 64-node Dragonfly.

Table 5.5: Communication times (in seconds) for different routing schemes under synthetic random permutation patterns and 64-node Dragonfly topology

	Perm1	Perm2	Perm3	Perm4	Perm5	Avg.
MIN	24.0	18.7	19.3	16.2	20.3	19.7
SDN-single	23.8	17.0	21.8	15.5	19.5	19.5
UGAL	15.3	13.5	16.8	11.7	19.6	15.4
SDN-ugal	15.0	13.0	16.8	12.1	19.6	15.3
SDN-optimal	12.0	11.7	18.1	11.8	19.5	14.6
SDN-model	13.9	12.6	17.2	12.5	19.5	15.1

Table 5.6: Communication times (in seconds) for different routing schemes under synthetic shift patterns on 64-node Dragonfly

	Sh-8	Sh-12	Sh-26	Sh-43	Sh-55	Avg.
MIN	43.6	21.8	32.7	27.2	38.1	32.7
SDN-single	26.7	21.3	21.3	24.0	22.1	23.1
UGAL	13.2	14.7	13.5	13.1	12.7	13.4
SDN-ugal	14.2	13.3	13.8	14.2	13.5	13.8
SDN-optimal	13.1	13.4	13.2	12.3	13.5	13.1
SDN-model	13.4	14.0	14.2	13.2	13.4	13.6

5.4.4 Results for Benchmarks/Applications

When an MPI application executes, its elephant flows are not always active since the application must also perform computation. Hence, the communication scenarios in application traces are very different from that in the synthetic traces and are much more realistic as the communication timing in programs becomes more important for the communication performance. This can significantly affect SDN performance since the rate allocation in SDN-model optimizes for the situation when all elephant flows are active at all times.

Table 5.7 shows the communication time for IS, Crystal Router (CR), Fill Boundary (FB), CMC, CNS, and BigFFT. Since CR and FB traces were collected from 1000 ranks, I only used the first 1000 nodes

Table 5.7: Communication time (in seconds) for different routing schemes for application traces on the 1024-node Dragonfly

	IS	CR	FB	BigFFT	CMC	CNS
MIN	339.2	173.5	504.1	564.1	18174.3	20330.3
SDN-single	296.8	111.5	504.3	504.4	18171.6	20259.8
UGAL	279.2	112.4	497.0	269.1	18170.0	20233.7
SDN-ugal	281.6	115.2	490.5	288.6	18170.5	20245.4
SDN-model	283.3	111.0	491.5	290.7	18170.8	20247.7

on the 1024-node Dragonfly topology. Other benchmarks are collected on 1024 ranks. Since the number of elephant flows in these benchmarks are much larger than a permutation, the model in SDN-optimal is too large to be solved. Hence, I only compare the performance of five routing schemes: MIN, SDN-single, UGAL, SDN-ugal, and SDN-model. Among these applications, the communication times for CMC and CNS, although very large, are insensitive to routing schemes. This is because the communications in these applications do not cause network contention regardless of the routing scheme. For other three programs, IS, CR, and BigFFT, we observe that SDN-single is much better than MIN. Figure 5.3a shows the relative performance of the multi-path routing schemes: the performance of SDN-ugal and SDN-model is normalized with UGAL. As can be seen from the figure, among the multi-path routing schemes, for all 6 benchmarks, SDN-ugal and SDN-model perform very similar to UGAL on average: SDN-ugal is only 1.5% worse than UGAL while SDN-model is 1.2% worse. This is achieved under the condition of a naive application of SDN's imprecise global network view. This indicates that SDN can be competitive to adaptive routing for HPC applications.

Adversarial Traffic. Table 5.8 shows the communication time for a number of NAS benchmarks on a 64-node Dragonfly. For this experiment, SDN-single in general improves over MIN, but performs worse than multi-path routing. Figure 5.3b shows the relative performance of the multi-path routing schemes: the performance of SDN-ugal and SDN-model is normalized with UGAL. As can be seen from the figure, among the multi-path routing schemes, for all 5 benchmarks, SDN-ugal is 3.7% worse than UGAL and SDN-model is 10.9% worse. The effectiveness of the rate allocation in SDN-model is significantly affected by the accuracy of the traffic demand. In our experiments, we have a cutoff of 100KB data for elephant flows. These elephant flows are not active all the time. Hence, the rate allocation for SDN-model may be calculated based on a pattern that is very different from the reality when the program executes. This is a challenge facing all SDN based schemes: for SDN to be effective, it must be able to obtain realistic traffic

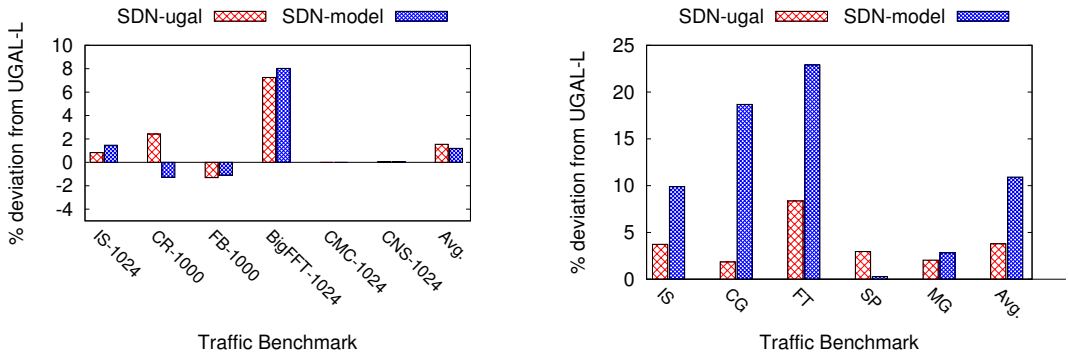


Figure 5.3: Relative performance of SDN-ugal and SDN-model with respect to UGAL for the programs on the (a)1024-node Dragonfly and (b) 64-node Dragonfly(less is better)

demand in order to perform effective resource management. Since SDN-ugal is only on average 3.7% worse than UGAL, SDN can be competitive with UGAL.

Table 5.8: Communication time (in seconds) for different routing schemes for NAS benchmarks on a 64-node Dragonfly

	IS	CG	FT	SP	MG
MIN	43.2	41.4	251.1	63.0	4.8
SDN-single	36.2	42.2	175.9	59.2	4.3
UGAL	25.5	32.6	114.2	52.0	4.1
SDN-ugal	26.5	33.2	123.8	53.5	4.2
SDN-model	28.1	38.7	140.4	52.2	4.2

5.4.5 Discussion

The experiments with synthetic traces and real applications show different pictures regarding the relative performance of SDN and UGAL. For the synthetic traces, SDN-optimal is not only competitive with UGAL but it also noticeably out-performs UGAL. For the application traces, UGAL out-performs SDN schemes on average. This highlights the strength of adaptive routing (UGAL). The ability of instantly reacting to traffic condition allows adaptive routing to achieve good performance for realistic HPC workloads without knowing the traffic demand.

For SDN, overall, the experiments demonstrate that SDN can be competitive with adaptive routing on Dragonfly. The results also highlight a number of challenges for applying SDN in the HPC domain. First,

methods to accurately estimate the traffic demand must be developed. In the experiments, a coarse-grained estimation that identifies elephant flows for the whole application is assumed. As can be seen from the NAS results on 64-node case, SDN-model is on average 10% worse than UGAL for this reason. This would indicate that more fine-grained estimation that identifies elephant flows in program segments may be necessary to increase the estimation accuracy. Second, efficient and effective rate allocation scheme that can be applied to large scale systems must be developed. As in the experiments, SDN-optimal clearly performs better than other SDN schemes for synthetic traces by having a better rate allocation scheme. Unfortunately, the algorithm for rate calculation in SDN-optimal is not scalable. Finally, improving the accuracy for demand estimation may require the network to be reconfigured more frequently. This can only happen when the support for network reconfiguration in SDN is more efficient than it is today. This is an issue that is being investigated by the OpenFlow community [30].

5.5 Summary

I consider schemes to apply SDN on Dragonfly networks and compare performance of SDN with that of adaptive routing. I conclude that adaptive routing is an effective scheme that performs well for realistic HPC workloads. SDN, with its ability to perform routing using the global network view, can be competitive with adaptive routing, and has the potential to out-perform adaptive routing when some issues are addressed. These issues include better methods to identify traffic demands, more efficient rate allocation algorithms that can be applied to large scale systems, and effective support for rate allocation/reallocation and network reconfiguration.

CHAPTER 6

CONCLUSION

In this dissertation, I explored three different areas toward designing more efficient interconnection networks. In Chapter 3 I focused on regular topologies as an important line of design for large-scale networks and proved that Jellyfish topology which is a random regular topology, is far from the optimal bounds with regards to several key topological metrics. Then I identified De-bruijn Graph as a near-optimal regular topology and showed that it has very good topological properties which enables it to achieve better communication performance in compare to Jellyfish under a variety of traffic patterns and network configurations.

In Chapter 4 I investigated the impact of designing better routing techniques for large-scale Dragonfly networks. The existing adaptive routing schemes for the Dragonfly topology rely on instantaneous queue lengths to take routing decisions. I showed that by identifying and incorporating the traffic pattern information in the routing process, a more effective routing scheme can be obtained. This new routing scheme significantly improves the communication performance on Dragonfly systems as verified by our simulation results.

Finally, in Chapter 5, I focused on utilizing state-of-the-art networking technologies in new domains. Specifically, I designed an SDN-enabled resource management scheme for HPC systems which can achieve comparable performance to currently deployed adaptive mechanisms on Dragonfly systems. This SDN routing technique removes the necessity of complex and expensive adaptive logic on Dragonfly networks, by using the global view of the network available in SDN architecture.

BIBLIOGRAPHY

[1] Aries hardware counters. Technical Report S-0045-30, Cray Inc., November 2015.

[2] NERSC Cori supercomputer. http://www.nersc.gov/users/computational-systems/cori/, 2017.

[3] Helgi Adalsteinsson, Scott Cranford, David A. Evensky, Joseph P. Kenny, Jackson Mayo, Ali Pinar, and Curtis L. Janssen. A simulator for large-scale parallel computer architectures. *Int. J. Distrib. Syst. Technol.*, 1(2):57–73, April 2010.

[4] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[5] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 253–266, San Jose, CA, 2012. USENIX.

[6] Robert Alverson, Duncan Roweth, and Larry Kaplan. The gemini system interconnect. In *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, HOTI '10, pages 83–87, Washington, DC, USA, 2010. IEEE Computer Society.

[7] Omer Arap, Geoffrey Brown, Bryce Himebaugh, and D. Martin Swany. Software defined multicasting for mpi collective operation offloading with the netfpga. In *Euro-Par*, 2014.

[8] 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.

[9] Jorgen Bang-Jensen and Gregory Z. Gutin. *Digraphs Theory, Algorithms and Applications*. Springer-Verlag, 2009.

[10] Maciej Besta and Torsten Hoefler. Slim Fly: A cost effective low-diameter network topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 348–359, 2014.

[11] Béla Bollobás. The isoperimetric number of random regular graphs. *Eur. J. Comb.*, 9(3):241–244, May 1988.

[12] Béla Bollobás and W. Fernandez de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2):125–134, 1982.

[13] V. G. Cerf, D. D. Cowan, R. C. Mullin, and R. G Stanton. A lower bound on the average shortest path length in regular graphs. *Networks*, 4(4):335–342, 1974.

[14] Dong Chen, Noel A. Eisley, Philip Heidelberger, Robert M. Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David L. Satterfield, Burkhard Steinmacher-Burow, and Jeffrey J. Parker. The ibm blue gene/q interconnection network and message unit. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 26:1–26:10, New York, NY, USA, 2011. ACM.

[15] N. G. de Bruijn. A combinatorial problem. *Nederl. Akad. Wetensh. Proc. Ser.*, A 49:758–764, 1946.

[16] J Dongarra, M Meuer, H Simon, and E Strohmaier. Top500 list.(november 2017). *URL http://www.top500.org/lists/2017/11/*, 2017.

[17] 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.

[18] P. Faizian, M. A. Mollah, X. Yuan, Z. Alzaid, S. Pakin, and M. Lang. Random regular graph and generalized de bruijn graph with *k*-shortest path routing. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):144–155, Jan. 2018.

[19] Peyman Faizian, Md Atiqul Mollah, Zhou Tong, Xin Yuan, and Michael Lang. A comparative study of sdn and adaptive routing on dragonfly networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 51:1–51:11, New York, NY, USA, 2017. ACM.

[20] Peyman Faizian, Md Atiqul Mollah, Xin Yuan, Scott Pakin, and Michael Lang. Random regular graph and generalized de bruijn graph with k-shortest path routing. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 103–112. IEEE, 2016.

[21] Peyman Faizian, Md Shafayat Rahman, Md Atiqul Mollah, Xin Yuan, Scott Pakin, and Mike Lang. Traffic pattern-based adaptive routing for intra-group communication in dragonfly networks. In *2016 24th Symposium on High-Performance Interconnects*. IEEE, 2016.

[22] Open Networking Foundation. Openflow switch specification, version 1.5.0 (protocol version 0x06), December 2014. available at https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[23] Pablo Fuentes, Enrique Vallejo, Marina García, Ramón Beivide, Germán Rodríguez, Cyriel Minkenberg, and Mateo Valero. Contention-based nonminimal adaptive routing in high-radix networks. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 103–112. IEEE, 2015.

[24] Marina Garcia, Enrique Vallejo, Ramon Beivide, Miguel Odriozola, Cristobal Camarero, Mateo Valero, Jesús Labarta, Cyriel Minkenberg, et al. On-the-fly adaptive routing in high-radix hierarchical networks. In *2012 41st International Conference on Parallel Processing*, pages 279–288. IEEE, 2012.

[25] Marina García, Enrique Vallejo, Ramón Beivide, Miguel Odriozola, and Mateo Valero. Efficient routing mechanisms for dragonfly networks. In *2013 42nd International Conference on Parallel Processing*, pages 582–592. IEEE, 2013.

[26] M. Imase and M. Itoh. Design to minimize a diameter on building block network. *IEEE Transactions on Computers*, C-30:439–443, 1981.

[27] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James D. Balfour, Brian Towles, David E. Shaw, John Kim, and William J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *ISPASS*, pages 86–96. IEEE Computer Society, 2013.

[28] 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.

[29] Gregory Johnson, Darren J. Kerbyson, and Mike Lang. Optimization of InfiniBand for scientific applications. In *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8. IEEE Computer Society Press, 2008.

[30] K. Karenos, V. Kalogeraki, and S. V. Krishnamurthy. A rate control framework for supporting multiple classes of traffic in sensor networks. In *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 11 pp.–297, Dec 2005.

[31] W. H. Kautz. Bounds on directed (d,k) graphs. *Theory of Cellular Logic Networks and Machines, AFCRL-68-0668 Final Report*, pages 20–28, 1968.

[32] J. Kim, W. J. Dally, J. Dally, and D. Abts. Adaptive routing in high-radix clos network. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 7–7, Nov 2006.

[33] 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.

[34] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D. Frank Hsu, and Henri Casanova. A case for random shortcut topologies for HPC interconnects. *SIGARCH Comput. Archit. News*, 40(3):177–188, June 2012.

[35] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[36] Jason Lee, Zhou Tong, Karthik Achalkar, Xin Yuan, and Michael Lang. Enhancing infiniband with openflow-style sdn capability. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 36:1–36:12, Piscataway, NJ, USA, 2016. IEEE Press.

[37] M. Imase and M. Itoh. A design for directed graph with minimum diameter. *IEEE Transactions on Computers*, C-33:782–784, 1983.

[38] P. Makpaisit, K. Ichikawa, P. Uthayopas, S. Date, K. Takahashi, and D. Khureltulga. Mpi_reduce algorithm for openflow-enabled network. In *2015 15th International Symposium on Communications and Information Technologies (ISCIT)*, pages 261–264, Oct 2015.

[39] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[40] Baatarsuren Munkhdorj, Keichi Takahashi, Dashdavaa Khureltulga, Yasuhiro Watashiba, Yoshiyuki Kido, Susumu Date, and Shinji Shimojo. Design and implementation of control sequence generator for sdn-enhanced mpi. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, NDM '15, pages 4:1–4:9, New York, NY, USA, 2015. ACM.

[41] Dritan Nace, Linh Nhat Doan, Olivier Klopfenstein, and Alfred Bashllari. Max-min fairness in multi-commodity flows. *Computers and Operations Research*, 35(2):557–573, February 2008.

[42] Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. High throughput data center topology design. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 29–41, 2014.

[43] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, page 17, 2012.

[44] K. Takahashi, D. Khureltulga, B. Munkhdorj, Y. Kido, S. Date, H. Yamanaka, E. Kawai, and S. Shimojo. Concept and design of sdn-enhanced mpi framework. In *2015 Fourth European Workshop on Software Defined Networks*, pages 109–110, Sept 2015.

[45] K. Takahashi, D. Khureltulga, Y. Watashiba, Y. Kido, S. Date, and S. Shimojo. Performance evaluation of sdn-enhanced mpi allreduce on a cluster system with fat-tree interconnect. In *High Performance Computing Simulation (HPCS), 2014 International Conference on*, pages 784–792, July 2014.

[46] Jongmin Won, Gwangsun Kim, John Kim, Ted Jiang, Mike Parker, and Steve Scott. Overcoming far-end congestion in large-scale networks. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 415–427. IEEE, 2015.

[47] Xin Yuan, Santosh Mahapatra, Michael Lang, and Scott Pakin. LFTI: A new performance metric for assessing interconnect designs for extreme-scale HPC systems. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, pages 273–282, 2014.

[48] Xin Yuan, Santosh Mahapatra, Wickus Nienaber, Scott Pakin, and Michael Lang. A new routing scheme for Jellyfish and its performance with HPC workloads. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 36:1–36:11, 2013.

[49] Xin Yuan, R. Melhem, and R. Gupta. Compiled communication for all-optical TDM networks. In *Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE Conference on*, pages 25–25, 1996.

BIOGRAPHICAL SKETCH

Peyman Faizian received his BS degree in computer science and MS degree in algorithms and computation from University of Tehran, Iran, in 2006 and 2009, respectively. He also received another MS degree in Computer Science from Florida State University, in 2016. During his PhD years at Florida State University, he focused on various approaches to evaluate and improve the performance of large-scale interconnection networks. His research interests include distributed systems, interconnection networks, high performance computing and software defined networking.