Q-adaptive: A Multi-Agent Reinforcement Learning Based Routing on Dragonfly Network

Yao Kang
ykang17@hawk.iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

Xin Wang
xwang149@hawk.iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

Zhiling Lan
lan@iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

ABSTRACT

High-radix interconnects such as Dragonfly and its variants rely on adaptive routing to balance network traffic for optimum performance. Ideally, adaptive routing attempts to forward packets between minimal and non-minimal paths with the least congestion. In practice, current adaptive routing algorithms estimate routing path congestion based on local information such as output queue occupancy. Using local information to estimate global path congestion is inevitably inaccurate because a router has no precise knowledge of link states a few hops away. This inaccuracy could lead to interconnect congestion. In this study, we present Q-adaptive routing, a multi-agent reinforcement learning routing scheme for Dragonfly systems. Q-adaptive routing enables routers to learn to route autonomously by leveraging advanced reinforcement learning technology. The proposed Q-adaptive routing is highly scalable thanks to its fully distributed nature without using any shared information between routers. Furthermore, a new two-level Q-table is designed for Q-adaptive to make it computational lightly and saves 50% of router memory usage compared with the previous Q-routing. We implement the proposed Q-adaptive routing in SST/Merlin simulator. Our evaluation results show that Q-adaptive routing achieves up to 10.5% system throughput improvement and 5.2x average packet latency reduction compared with adaptive routing algorithms. Remarkably, Q-adaptive can even outperform the optimal VALn non-minimal routing under the ADV+1 adversarial traffic pattern with up to 3% system throughput improvement and 75% average packet latency reduction.

CCS CONCEPTS

• **Networks** → **Routing protocols**; **Network performance evaluation**; *Topology analysis and generation.*

KEYWORDS

HPC; interconnect network; Dragonfly; routing; multi-agent reinforcement learning

ACM Reference Format:
Yao Kang, Xin Wang, and Zhiling Lan. 2021. Q-adaptive: A Multi-Agent Reinforcement Learning Based Routing on Dragonfly Network. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '21), June 21–25, 2021, Virtual Event, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3431379.3460650

HPDC '21, June 22–25, 2021, Virtual Event, Sweden.

ACM ISBN 978-1-4503-8217-5/21/06.
https://doi.org/10.1145/3431379.3460650

1 INTRODUCTION

Interconnect network is a critical component in high-performance computing (HPC) systems. It serves as a "central nervous system" for data exchange between computer nodes [1]. Recent high-radix interconnect such as Dragonfly [17] and its variants [10][35] become a dominant interconnect topology in Top500 supercomputers [40]. The Slingshot interconnects [34], which will power future exascale HPC systems, also adopt Dragonfly topology for its cost-effective, low-diameter, and highly scalable features.

Dragonfly networks have a hierarchical design, in which system nodes are divided into several fully connected, identical groups. A dragonfly system is highly scalable due to the use of high-radix routers and a fully connected group structure. New groups can be flexibly added to the system while maintaining its low-diameter quality. With the fully connected groups, Dragonfly provides *high path diversity* such that packets can be forwarded either minimally between source and destination groups or non-minimally through any intermediate group in the system. High path diversity allows routing to take diverse paths under different network patterns: *minimal routing* is optimal for balanced distributed traffic such as uniform random traffic pattern, whereas *Valiant non-minimal routing (VAL routing)* is advantageous in case of unbalanced traffic such as adversarial traffic pattern.

Adaptive routing is often deployed in Dragonfly systems by dynamically delivering packets either minimally or non-minimally according to real-time network conditions [17][15]. Universal Globally-Adaptive Load-balanced routing (UGAL) is widely used in production dragonfly systems [10][34]. It allows the source router to choose between minimal or non-minimal routing paths with the least estimated congestion. Progressive Adaptive Routing (PAR) enhances UGAL by enabling source group routers to re-evaluate previous routing decisions based on the current estimated congestion [15]. Because no system-wide global information is shared among routers, both UGAL and PAR rely on local information such as output queue occupancy to estimate routing path congestion. As a switched fabric network, source and destination routers are often connected through several hops. Thus, local information can only be a good indicator for near-end congestion and often fails to infer far-end congestion [46] or downstream congestion [15] at neighbor and downstream hops. A simple example is illustrated in Figure 1 to show the limitation of UGAL and PAR. In this example, router s prefers $path1$ over $path2$ because the former has a less occupied output queue at router s. However, selecting $path1$ in this scenario is inappropriate because of the congestion at downstream routers

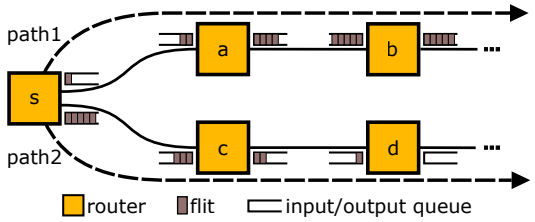

Figure 1: Issue of existing adaptive routing on Dragonfly, where local information fails to estimate global path congestion. Although path2 is the least congested path, existing adaptive routing methods typically choose path1 over path2.

of *a* and *b*. This illustrative example clearly demonstrates that adaptive routing based on local information (e.g., local output queue occupancy) could cause wrong decision making, hence leading to network congestion, delayed packet delivery, and limited system throughput.

In this work, we present *a learning-based routing method to autonomously learn network conditions on Dragonfly systems by leveraging advanced machine learning technology.* Specifically, we choose reinforcement learning (RL) over supervised learning as the latter requires human effort to provide true labels, or desirable routing behaviors during the training phase, which can be practically impossible [42]. Unlike supervised learning, RL agents learn to behave by interacting with an environment for an objective, which can be mathematically expressed to maximize a cumulative reward. RL has been successfully applied for car driving system [20], human-level gaming [24], etc. Recent studies have shown great promise of reinforcement learning in successfully solving computer system problems such as job scheduling [11][21], memory management [23], circuit design [43], NoC arbitration [48], etc.

A pioneering work in applying reinforcement learning to network routing was done by Boyan et al. in 1993 [3]. They proposed Q-routing, a multi-agent reinforcement learning (MARL) method that each router works as an independent agent and is guided by a state-action lookup table, Q-table, for routing decisions. In their study, Q-routing was evaluated on an irregular 6×6 grid network. Later, several enhancements were presented to improve the quality of Q-routing [4][26]. These studies mainly focused on similar irregular grid networks at a small scale (e.g., 15–36 routers).

Computer systems have been evolved dramatically over the past twenty years in terms of interconnect topology as well as the system scale. Little is known about the feasibility and efficiency of applying MARL for routing in a large-scale Dragonfly system. Previous MARL studies, e.g., Q-routing, were focused on grid networks [3][4][26], which are significantly different from the hierarchical, high-radix Dragonfly networks. Furthermore, modern supercomputers typically contain hundreds or thousands of routers. Practically training a RL model with a large number of independent agents to a converged state is much more challenging than the previous small-scale case studies. HPC network traffic is constantly changing with different patterns and loads. This demands the training to converge in a reasonable time under both uniform and adversarial traffic patterns whereas the previous studies only considered the uniform random traffic pattern [3][26][39][27]. Moreover, considering computational resources on routers are limited, we must avoid any RL methods that have a high computational requirement, e.g., deep learning methods using neural networks [31][49].

To tackle the above challenges, we present *Q-adaptive routing*, a fully distributed MARL routing scheme for Dragonfly networks. Inspired by the Q-routing studies, Q-adaptive routing adopts table-based RL for decision making. Table-based RL methods are computationally efficient as compared to deep neural network based methods [31][49], hence being more realistic for practical use. Distinguishing from Q-routing, Q-adaptive utilizes a novel two-level Q-table. The two-level Q-table not only provides more learning information, but also mitigates outdated Q-value issue commonly experienced in large-scale systems due to spare updates. Moreover, our two-level Q-table only requires half of the memory usage of the Q-table used in Q-routing. New techniques are adopted for Q-adaptive routing to ensure timely update of values in the two-level Q-table, hence guarantee a fast and stable model convergence. Finally, Q-adaptive assures packets to be delivered within five hops, which guarantees a routing livelock and deadlock free design for Dragonfly networks (details in Section 4).

We implement our design in SST/Merlin, a flit-level event-driven simulation toolkit [32]. We evaluate Q-adaptive routing on a 1,056-node Dragonfly system consisting of 264 routers and further demonstrate its scalability on a 2,550-node system with 510 routers. Extensive experiments are conducted to evaluate Q-adaptive routing with several routing methods presented for Dragonfly systems under various traffic patterns and loads. Our results show that Q-adaptive routing outperforms the existing adaptive routing methods by up to 10.5% in network throughput and over five times of reduction in packet latency. Moreover, Q-adaptive routing outperforms the optimal routing (e.g., non-minimal routing) under adversarial traffic patterns. For system convergence, our results show that Q-adaptive is guaranteed to converge under various traffic patterns and network loads within 500 µs.

For the rest of the paper, Section 2 presents background and related work. Section 3 discusses the challenges of designing MARL based routing for Dragonfly systems. Section 4 gives the details of our work, followed by evaluations in Section 5 and Section 6. Section 7 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 Dragonfly Topology

Dragonfly [17] is a high-radix interconnect topology featuring high-bandwidth, low-diameter, and extreme scalability at a reasonable cost. As shown in Figure 2, Dragonfly arranges network resources (i.e., routers and links) into multiple identical groups such that the network topology is organized in a three-tiered hierarchy: nodes connections, intra-group connection through local links, and inter-group connection through global links. Depending on the connected link type, router ports can be classified into host port, local port, or global port for different connection levels in the topology hierarchy. Dragonfly uses all-to-all inter-group connection such that packets generated in the source group can be minimally forwarded to any destination groups using only one global link hop.

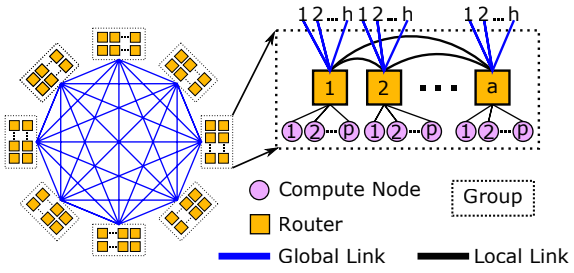

Figure 2: Dragonfly Topology

In contrast to a fully connected inter-group connection, the intra-group connection is not limited to a single form. For example, Figure 2 shows a 1-dimensional fully connected intra-group network as presented in [17]. Additionally, the intra-group connection can also be configured with a 2-dimensional partially all-to-all network in Cray Cascade systems [10], or a bipartite-graph tree network used in Megafly [12] and Dragonfly+ [35]. In this work, we focus on the Dragonfly topology with all-to-all inter-/intra-group connection as shown in Figure 2, because this structure will be deployed on the Slingshot interconnects to support the next-generation exascale-computing systems [34]. We shall point out that the proposed Q-adaptive routing is not limited to a single form and can also be applied to other Dragonfly variants. In the rest of the paper, we simply refer to this all-level fully connected Dragonfly as Dragonfly and use the nomenclature in Table 1.

Table 1: Dragonfly configurations

Parameter	Note	Systems	
N	Number of nodes	1056	2550
p	compute nodes per router	4	5
a	routers per group	8	10
h	global links per router	4	5
$k = p + h + a - 1$	ports per router	15	19
$g = ah + 1$	number of groups	33	51
$m = g * a$	routers in the system	264	510

With the hierarchical design, Dragonfly is a diameter-3 topology that packets can be minimally delivered within three hops: one local link in the source group, one global link crossing groups, and one local link in the destination group. Because a minimal path uses twice as much local link bandwidth as that of global link, a proper Dragonfly should be configured with $a = 2p = 2h$ for load balancing purposes [17].

In production, Cray's Aries interconnects adopt *credit-based flow control*, which guarantees a packet lossless Dragonfly system. In such a system, each router maintains a counter, named credits, to record the number of free buffer slots in its downstream routers. A flit can only be forwarded to the next router when this number is positive. Whenever a flit is forwarded and leaves a router, the router will send back a credit to its upstream router indicating that a new free buffer slot becomes available.

2.2 Dragonfly Routing

Dragonfly routing is commonly evaluated under the best and the worst case traffic patterns [17][15]:

Uniform Random (UR): Each node communicates with a randomly selected node on a per message basis. UR traffic pattern balances network traffic through the randomly chosen source-destination pair, thus is considered as the best case on Dragonfly topology. Minimal routing typically provides the optimal performance and is expected to achieve 100% system throughput.

Adversarial (ADV+i): Each node in group G sends packets to a random node in group $G+i$. This is the worst case as network traffic is extremely unbalanced such that packets generated in one group all target another group. As a result, the limited number of global links between groups becomes the bottleneck. Non-minimal routing via intermediate groups is typically suitable for this traffic pattern and is expected to achieve up to 50% system throughput. Note that ADV+i may also lead to local link congestion in intermediate groups [13]. Figure 3 illustrates the potential local link congestion under ADV+4 traffic. For the 1,056-node Dragonfly system described in Table 1, the ADV+4 traffic pattern has the most local link congestion, whereas ADV+1 has the least congestion.

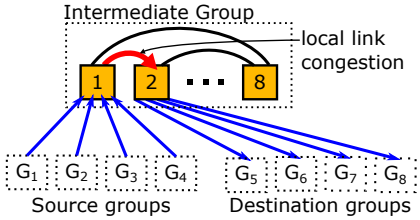

Figure 3: Local link congestion under ADV+4. In this case, G1 send packets to G5, G2 to G6, etc. When the packets are routed non-minimally, the local link between router 1 and router 2 in the intermediate group becomes bottleneck (red arrow).

Dragonfly routing mechanisms can be broadly classified as non-adaptive and adaptive methods. *Non-adaptive routing* includes minimal and Valiant routing:

Minimal routing (MIN): Packets are minimally forwarded to their destinations within three hops. Minimal routing is the optimal routing policy for the UR traffic pattern, but the worst choice for the ADV+i traffic pattern. Minimal routing requires two virtual channels (VCs) to avoids routing deadlock [7][8].

Valiant routing (VALg, VALn): Valiant non-minimal routing (VAL) is an optimal solution for the ADV+i traffic pattern. Valiant-global (**VALg**) routes packets minimally to a random intermediate *group*, then minimally to the destination group [17]. A VALg non-minimal path contains up to five hops and requires three VCs to avoid deadlock. Valiant-node (**VALn**) solves the local link congestion problem by rerouting packets to reach a specific random *router* in an intermediate group [46]. As a result, packets are not minimally forwarded in intermediate groups at the cost of an additional local link making VALn uses up to six hops and requires four VCs to avoid deadlock.

Since both MIN and VAL are only good for one traffic pattern, *adaptive routing* is typically adopted in production systems to let routers dynamically choose between minimal and non-minimal paths [17][15]. The dynamic decisions depend on network conditions deduced from a router's local queue occupancy. If the local queue occupancy of a candidate minimal path is less than twice of a candidate non-minimal path, the router will forward the packet minimally. In practice, a bias can be added to give preference for minimal paths. There are two widely used adaptive routing methods:

UGALg, UGALn: Universal Globally-Adaptive Load-balanced routing (UGAL) [17] lets source routers make adaptive routing decisions. **UGALg** chooses between a minimal path and a VALg non-minimal path, whereas **UGALn** considers the VALn non-minimal path.

PAR: Progressive Adaptive Routing (PAR) chooses between minimal paths and VALn non-minimal paths [15][46]. In contrast to UGAL, PAR allows source group routers to re-evaluate routing decisions if a packet is being minimally routed. PAR can achieve better performance than UGAL because it enables source group routers to adjust routing strategies depending on the local congestion, which can be difficult to be perceived by the source router. However, switching from a minimal path to a non-minimal path costs an additional local link making PAR use up to seven hops. To avoid routing deadlock, PAR requires five VCs.

There are several other routing related studies. Jiang et al. introduced different indirect adaptive routing algorithms to improve the original UGAL [15]. Garcia et al. proposed the OFAR routing to mitigate local link congestion and Won et al. proposed window-based PAR routing to overcome far-end congestion [13][46]. Rahman et al. customized UGAL by limiting the set of candidate non-minimal paths according to Dragonfly configuration [30]. Chunduri et al. studied run to run performance variability on a Dragonfly production system [5], and simulation studies analyzed network interference with different intra-group connections [47][16][44]. Mubarak et al. and Wilke et al. proposed QoS based approaches to mitigate network interference [25][45]. Jain et al. proposed to reduce network hot-spots through random job placement [14]. De Sensi et al. proposed to reduce network noise using dynamic minimal routing bias [9]. Fundamentally, existing Dragonfly routing methods are based on heuristics by using local queue occupancy for decision making. This work leverages multi-agent reinforcement learning (MARL) to train routers to learn beyond queue occupancy for routing.

2.3 Reinforcement Learning based Routing

Reinforcement learning is a subarea of machine learning that automatically learns to maximize cumulative reward through interaction with the environment [37]. The environment can be formulated as a Markov Decision Process (MDP), where an agent observes the current environment state, selects an action, observes the environment move to a new state, and receives a feedback reward. The whole process iterates until a terminal state. The agent is expected to find an optimal policy that maximizes the cumulative reward. *Q-learning* is a model-free RL algorithm with the expected cumulative reward named Q-value and uses ϵ-greedy policy to balance the exploration-exploitation dilemma [37]. In practice, Q-values can be either stored in Q-table or be approximated by a deep neural network [24]. When there is more than one agent in the system, it falls into the domain of *multi-agent reinforcement learning (MARL)*. Depending on the objective, MARL can be either competitive or cooperative [6]. Routing on large-scale interconnect networks is by nature a MARL problem because routers in the system can be considered as independent agents. In this work, we consider our Dragonfly routing problem as *a cooperative independent MARL process* that routers behave as independent learners for a common objective, that is, to deliver messages in the shortest time.

2.3.1 ***Q-routing.*** Boyan et al. proposed *Q-routing*, a MARL routing method based on Q-learning [3]. In Q-routing, each router maintains a Q-table of size $m \times (k-p)$, where $m = g \times a$ is the total number of routers in the system, k is the router radix and p is the number of host ports on each router. Table 2 illustrates a Q-table where each Q-value is the estimated packet delivery time from the current router to a destination router through a corresponding port. For example, when router R_x receives a packet whose destination is router R_d, R_x will read row d of the Q-table to send the packet through the port with the minimum estimated delivery time i.e., the smallest Q-value. Q-routing explores the solution space with ϵ-greedy policy, such that a random port with ϵ probability may be selected as the final outbound port instead of the best port. All downstream routers forward the packet in the same way.

Table 2: A Q-table example, where port 2 will be selected for routing to destination R_1.

To Dest.	By Port			
	1	2	...	$k-p$
R_1	320ns	300ns	...	480ns
R_2	260ns	310ns	...	310ns
...				
$R_m(R_{g\times a})$	330ns	330ns	...	330ns

As a packet is passed to its destination, routers on the routing path update their Q-tables. Assuming the previous packet destined for R_d is forwarded from R_x to R_y. After R_y selects next hop, its smallest Q-value Q_y and a reward r will be sent back to R_x. In Q-routing, the immediate reward is defined as the packet traveling time from R_x to R_y. After receiving the feedback, R_x updates its Q-value Q_x for destination R_d with the Q-learning algorithm shown in Equation (1), where α is the learning rate.

$$\Delta Qx = \alpha(r + Q_y - Q_x) \quad x, y \in N \tag{1}$$

In Q-routing, routers are independent agents because routing decisions are purely made based on the individual Q-table and there is no explicit shared information between routers.

Nowe et al. enhanced Q-routing with periodic Q-value broadcasting to improve the precision of Q-table at the expense of extra network traffic [26]. Tao et al. and Peshkin et al. proposed to use policy gradient and gradient ascent method for MARL routing [39][27]. However, both require a system-wide global reward, which causes extra network traffic and is impractical for large systems. Prashanth

et al. theoretically applied an actor-critic algorithm for routing in a centralized manner [29]. Deep learning routing approaches were presented in several studies [31][49]; however deep neural networks require heavy computation resources, which makes them unrealistic for practical usage.

None of the above studies target Dragonfly topology, hence little is known about the feasibility and the performance of MARL routing in dragonfly systems, in particular those deployed with hundreds of routers.

2.3.2 ***Issues of Q-Routing on Dragonfly.*** While Q-routing is a good starting point, it was presented in 1993 and evaluated on a 6×6 irregular grid network, which is very different from modern Dragonfly systems. The original Q-routing method does not limit routing path length, which prevents it from being applicable on Dragonfly topology for two reasons: (1) *routing livelock*, a packet travels in the network without advancing towards its destination; (2) *routing deadlock*, Dragonfly relies on distributing packets on different virtual channels (VCs) to avoid deadlock. When a packet traverses in the network, it increases its VC index to break the channel dependency loop [8]. Since the number of VCs is limited by the hardware, a packet must reach its destination before exhausting all available VCs.

To address the above issues, a naive approach would be the use of a hop count threshold denoted as *maxQ*. Once a packet has visited *maxQ* routers, it will be routed minimally to the destination. Since Dragonfly is a diameter-3 topology, packets are assured to be delivered within *maxQ* + 3 hops such that livelock can never happen and deadlock can be resolved with a limited number of VCs.

We analyze this naive method by using SST/Merlin simulation [32]. Our results indicate that there is no one-size-fits-all value for *maxQ*. While UR prefers small *maxQ* to mimic a minimal routing, ADV+i favors larger *maxQ* values to bypass the highly utilized global links between source and destination groups. Moreover, extremely large *maxQ* can be detrimental such that packets spend a long time in the network visiting unnecessary routers, resulting in both large packet latency and low network throughput. Most importantly, regardless of the value of *maxQ*, performance under the ADV+4 pattern is always worse than the ADV+1 pattern. This indicates that Q-routing is not capable of handling local link congestion. As a result, it is difficult, if not impossible, to set an appropriate *maxQ* for both UR and ADV+i traffic patterns. Additionally, since the size of the Q-table is linear to the system size, in the case of large systems, Q-values could be outdated when a destination is rarely visited. As such, we observed instability and slow convergence of the enhanced Q-routing method in our experiments. In summary, the original Q-routing and the naive Q-routing enhancement do not work for Dragonfly networks.

3 TECHNICAL CHALLENGES

Exploring MARL routing for Dragonfly systems poses several key challenges:

Topology uniqueness. Dragonfly is a hierarchical topology with all-to-all connected router groups. As a result, the topology provides a large number of candidate routing paths that are either minimal or non-minimal through any intermediate group. This *high path diversity* makes it challenging to select the most proper path based on a router's local information. When proposing the Dragonfly topology, Kim et al. emphasized the importance of studying routing performance under two extreme traffics: the best scenario of UR pattern and the worst scenario of ADV+i pattern [17]. In reality, system-scale traffic patterns can be any case between these two extremes. Furthermore, a good routing algorithm must mitigate *global link congestion* as well as potential *local link congestion* shown in Figure 3. Because of high path diversity and various link congestions, designing an efficient routing algorithm is complicated and challenging.

Routing livelock and deadlock. On Dragonfly systems, when a packet traverses in the network, it relies on increasing its VC index to break the channel dependency loop. Since the number of VCs is limited by the hardware, packets must be delivered under certain hops. Directly applying existing RL routing (e.g., Q-routing) designed for other network topologies could lead to routing livelock and deadlock discussed in Section 2.3.2. An efficient MARL routing method must take these into account and deliver packets in limited hops.

Distributed learning. A key challenge in multi-agent learning is non-stationary. Since all agents are learning simultaneously, the environment is constantly changing from the perspective of any agent. Hence directly applying a single agent algorithm in a multi-agent environment could lead to unstable and unsatisfactory performance. An efficient MARL routing must enforce certain coordination between independent agents.

Implementation cost. A good MARL routing must be lightweight in terms of both computational requirement and hardware requirement (i.e, memory consumption). As a result, deep reinforcement learning is unfavorable due to the heavy computation posed by neural networks, and tabular RL method is preferred.

4 Q-ADAPTIVE ROUTING

We present Q-adaptive routing in this section. It consists of three key components: (1) two-level Q-table, (2) routing with two-level Q-table, and (3) Q-table update. At the end, we discuss how these techniques address the challenges listed in Section 3.

Two-level Q-table. Table 3 depicts the newly designed two-level Q-table. Compared with the original Q-table (Table 2) that is accessed according to packet destination, two-level Q-table uses both packet source and destination information. For a Dragonfly configured with g groups, p nodes per k-radix router, each router maintains a $(g \times p) \times (k - p)$ two-level Q-table. When routing a packet that is generated on node n for group j, Q-values on the row of $j \times p + n$ will be evaluated for the port with the minimum estimated delivery time. Compared with the original Q-table, the two-level Q-table is smaller by 50%. The original Q-table contains $g \times a$ rows, whereas the two-level Q-table has $g \times p$ rows, where a is the number of routers per group. A balanced Dragonfly should be configured with $p = a/2$ [17], thus halving the table size. The advantages of using a smaller table are multifold: *(i)* Q-tables are stored on routers' memory, which has limited capacity. Our two-level Q-table only requires half the memory space compared with the original Q-table, thus improves system scalability. *(ii)* The smaller table alleviates the outdated Q-value issue caused by spare updates. In the two-level Q-table, Q-values of rarely visited destinations

Table 3: Two-level Q-table

To Group	From Node	By Port			
		1	2	···	k-p
G_1	N_1	Q-values: Estimated Delivery Time			
	N_2				
	...				
	N_p				
G_2	N_1				
	N_2				
	...				
	N_p				
...	...				
G_g	N_1				
	N_2				
	...				
	N_p				

can also be updated along with the ones that are more frequently accessed because value update for any destination router of the same group is on the same row under the "`To Group`" column. *(iii)* Differentiating Q-values for the same group by packet sources provides more learning information for the RL agent to take advantage of high path diversity brought by Dragonfly topology.

Routing with two-level Q-table. Figure 4 presents a flow chart of the Q-adaptive routing. When a router receives a packet, it goes through the following steps to select an outbound port:

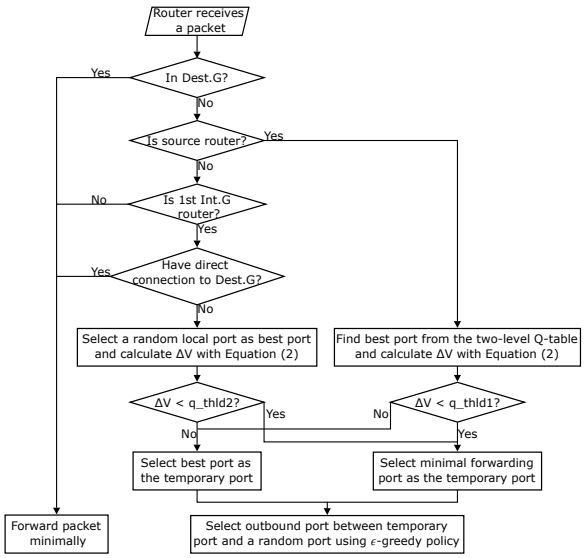

Figure 4: Flow chart for Q-adaptive routing. Dest.G and Int.G stand for destination group and intermediate group respectively.

(1) If the router is in the packet's destination group, it minimally forwards the packet.
(2) If the router is the packet's source router, it selects the *best_path_port* with the smallest Q-value (Q_{best_path}). Next, it selects a minimal forwarding port (*min_path_port*) and reads the port's Q-value (Q_{min_path}) from the two-level Q-table. Then, a temporary port (*temp_port*) is set using Equation (2) with `threshold` = q_thld1. Finally, the packet's outbound port is selected between *temp_port* and a random port using ϵ-greedy policy.
(3) If the router is the first intermediate group router visited by the packet, and has a direct connection to the packet's destination group, it forwards the packet minimally. If the router is not connected to the destination group, a port for a minimal forwarding path (*min_path_port*) and a random local port noted as *best_path_port* are selected. Both ports' Q-values (Q_{min_path} and Q_{best_path}) are read from the two-level Q-table. Next, *temp_port* is set using Equation (2) with `threshold` = q_thld2. The final outbound port for the packet is selected between *temp_port* and a random port with ϵ-greedy policy.
(4) Otherwise, the router minimally forwards the packet towards its destination.

$$\Delta V = (Q_{min\_path} - Q_{best\_path})/Q_{min\_path}$$
$$temp\_port = \begin{cases} min\_path\_port & \text{if } \Delta V < \texttt{threshold} \\ best\_path\_port & \text{otherwise} \end{cases} \quad (2)$$

With the above steps, most of the routers on a routing path forward a packet minimally except at two locations: the source router and the first intermediate group router visited by a packet. When a packet is injected into the network, the source router selects its routing path according to the two-level Q-table. The selected path could be a minimal path or a non-minimal path that traverses an intermediate group. When the packet is routed non-minimally and arrives at its first intermediate group router, it can be dynamically rerouted through a random router in the same group to bypass local link congestion based on the Q-values between the minimal forwarding path and the selected rerouting path. When the routing path is chosen under the guidance of a two-level Q-table, we add a `threshold` in Equation (2) to give a bias towards the minimal forwarding path. Two thresholds (q_thld1 and q_thld2) are used for the source router and the first intermediate group router respectively. Both are tunable parameters.

As a result, Q-adaptive routing behaves similarly to the adaptive routing algorithms by forwarding a packet either minimally or non-minimally. However, instead of always rerouting packets in the intermediate group as UGALn or PAR, Q-adaptive does it dynamically according to the network condition. Since Dragonfly is a diameter-3 topology, and only two routers on the routing path can make dynamic decisions, all packets are guaranteed to be delivered within five hops, which solves the livelock problem. To avoid routing deadlock, Q-adaptive uses five VCs and a packet increments its VC index at every hop.

Q-value update. The next challenge is how to initialize and update Q-table. Lauer et al. proposed a *distributed Q-learning* algorithm [19]. Basically, it allows each agent to update Q-values only in a positive direction and ignores penalties from other agents. As such, for each router, its routing strategy is not affected by other routers' mistakes. While the authors proved the convergence of the

distributed Q-learning algorithm in deterministic multi-agent environments, the algorithm requires initializing Q-values to very large numbers causing the downside of long convergence time. In [22], Matignon et al. pointed out that totally ignoring penalties could stick the system at a sub-optimal equilibrium and hence presented *hysteretic Q-learning*. Hysteretic Q-learning uses two learning rates to update the policy in positive and negative directions separately. The advantages of hysteretic Q-learning are that it does not pose any requirement for Q-value initialization and separate learning rates make the system more stable. Inspired by these studies, we propose to update Q-values with following equation:

$$\delta = r + Q_y - Q_x, x \text{ and } y \in N$$
$$Q_x = \begin{cases} Q_x + \alpha * \delta & \text{if } \delta < 0 \\ Q_x + \beta * \delta & \text{otherwise} \end{cases} \quad (3)$$

Here, α and β are the positive and negative learning rates, and r is the reward, defined as the packet traveling time between neighbor routers of x and y.

In summary, Q-adaptive routing is a fully distributed MARL algorithm. Each router acts as an independent agent that learns to route by interacting with the environment using its two-level Q-table. Not having any shared information between routers saves link bandwidth, which makes Q-adaptive routing easy to be implemented and scaled. Meanwhile, Q-adaptive routing addresses the challenges discussed in Section 3:

Topology uniqueness. The existing adaptive routing method relies on local information to choose among up to four random routes (two minimal and two non-minimal) without exhaustively considering all possible paths [2]. In contrast, Q-adaptive routing addresses Dragonfly's high path diversity feature by using the two-level Q-table to select the best path among all viable candidates and find underestimated solutions with ϵ-greedy exploration. Moreover, the corresponding table updating techniques refresh Q-values in a timely fashion such that a router can learn to reroute packets either locally or globally to bypass network congestion. Furthermore, learning performance can be flexibly tuned with two learning rates (α and β), and a proper bias toward minimal path can be controlled with two thresholds (q_thld1 and q_thld2).

Routing livelock and deadlock. Q-adaptive routing guarantees a packet to be delivered efficiently within five hops. Therefore, routing livelock never happens. Limited routing path length also makes the system deadlock free with five VCs.

Distributed learning. While each router is an independent agent, Q-adaptive routing coordinates them through two-level Q-table updating with two learning rates. By leveraging hysteretic Q-learning, Q-adaptive routing guarantees a fast learning convergence mainly controlled by α and maintains the converged state stable with β.

Implementation cost. Q-adaptive routing requires no specific hardware implementations nor heavy computational resources. The proposed two-level Q-table halves the original Q-table size. The small-sized Q-table improves the algorithm's scalability, as well as alleviates the outdated Q-value problems inherent in the original Q-routing algorithm. Q-adaptive routing can be easily deployed on current high-radix Dragonfly routers by replacing the routing table with the proposed two-level Q-table of size $O(g*p*k)$ [33]. The two-level Q-table updating only involves basic arithmetic operations and the reward transferred between neighbor routers could be embedded into the credit-based flow control flits or other network control packets.

5 EVALUATION

We use and expand the open-source, community-built simulation toolkit SST/Merlin [32]. SST/Merlin is a high-fidelity network modeling toolkit supporting several interconnect topologies including Dragonfly. It provides a detailed flit-level input-output queued high-radix router model with credit-based flow control. In this work, we enhance the toolkit by implementing different traffic patterns. We implement our Q-adaptive routing, PAR, UGALg, and UGALn in the toolkit. In our evaluation, we compare Q-adaptive with five routing methods: minimal routing (MIN), VALn non-minimal routing, UGALg, UGALn, and PAR. Note that MIN and VALn represent optimal routing strategies under UR and ADV+i traffic patterns respectively. UGAL and PAR are the existing adaptive routing mechanisms deployed on production Dragonfly systems and serve as the baseline for comparison study.

We evaluate Q-adaptive routing on two systems: 1,056-node and 2,550-node Dragonfly systems. In this section, we examine the results achieved on the 1,056-node system. In Section 6, we will present performance results on the scale-up 2,550-node case to demonstrate the scalability and feasibility of the Q-adaptive routing.

5.1 Experimental Setup

Each system is configured using 128B single flit packets to avoid the effect of flow control on routing results with single packet messages. Routers are configured with VC buffer size of 20 packets, and all the link bandwidth is 4GB/s with local link latency of 30ns [2], and global link latency of 300ns to keep the 1:10 ratio as mentioned in previous works [17][15][46]. All adaptive routing algorithms are set with zero bias towards minimal routing and use local output queue occupancy plus the used credit count to estimate routing path congestion.

For the 1,056-node system, all results are evaluated with $\alpha = 0.2$, $\beta = 0.04$, $\epsilon = 0.001$, $q\_thld1 = 0.2$, $q\_thld2 = 0.35$ in this section. The values are selected to balance the trade-off between convergence speed and network peak performance. Q-values are initialized to the theoretical packet delivery time without any congestion through a minimal routing path.

In our experiments, different routing methods are compared under various traffic patterns. For each pattern, messages are generated at an adjustable interval to create different network conditions, ranging from a lightly loaded system to a fully loaded system. We simulate network usage according to *offered load*, which is defined as the ratio between the message generation rate and the system-wide total injection bandwidth. Thus, offered load is between 0 and 1, where 0 means an empty network and 1 means a fully loaded system.

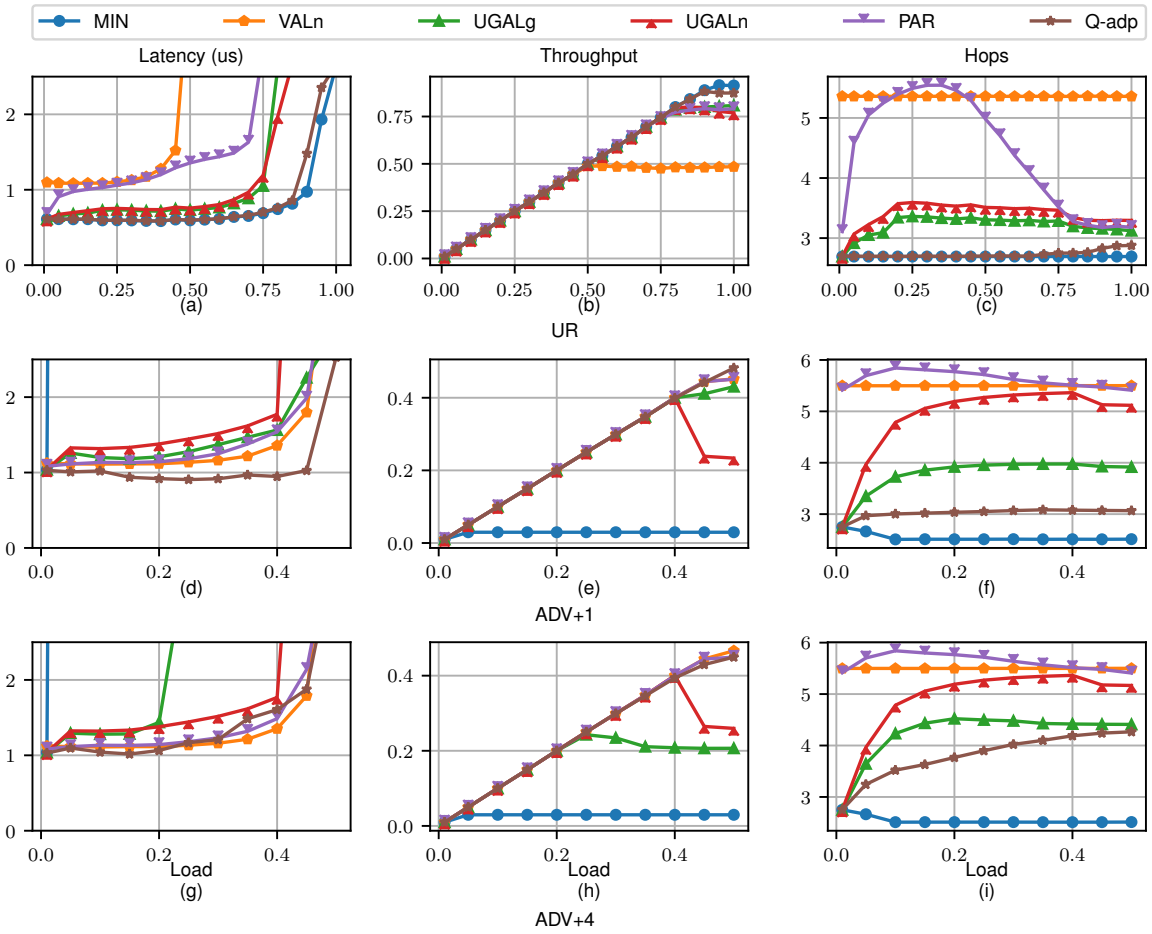

Figure 5: Q-adaptive on the 1056-node Dragonfly

5.2 Evaluation metric

We use two evaluation metrics for performance comparison: (1) *packet latency*, which denotes the packet traveling time from its generation to delivery; and (2) *system throughput*, which is measured as the aggregated message receiving rate across the system. Therefore, system throughput is always smaller than the *offered load*, with 0 means all packets are congested or blocked on the network.

5.3 Routing under Different Loads

Figure 5 depicts Q-adaptive routing results for the UR and ADV+1, ADV+4 traffic patterns. For each traffic pattern, three plots show the packet latency, system throughput, and packet hop count under different offered loads. The results shown in Figure 5 are collected as the arithmetic average over 100 µs after the system becomes stable. Adaptive and minimal, non-minimal routing results are also plotted for comparison.

UR pattern: Under the UR traffic pattern, Q-adaptive outperforms all the adaptive routing algorithms regarding system throughput shown in Figure 5(b). Under the maximum load, Q-adaptive reaches 88.25% system throughput, which is 6.60%, 10.51%, and 8.32% higher than UGALg, UGALn, and PAR respectively. Meanwhile, system throughput of Q-adaptive is only 3.29% smaller than that of the optimal minimal routing.

Q-adaptive also outperforms adaptive routing algorithms regarding packet latency. In Figure 5(a), the packet latency of Q-adaptive routing is always smaller than that of adaptive routing methods. All the adaptive routing algorithms show obvious network congestion starting from the offered load of 0.8 with a rapid packet latency increase. On the other hand, Q-adaptive routing can handle this load with an average packet latency of 0.76 µs, which is 3.43x, 2.59x, and 5.22x smaller than that of UGALg, UGALn, and PAR respectively. Furthermore, the average latency of Q-adaptive is close to the optimal minimal routing's 0.74 µs under the load of 0.8.

ADV+1 pattern: Figure 5(e) shows that Q-adaptive routing can achieve a maximum of 48.20% system throughput, which is 5.15%, 8.20%, and 3.09% higher than UGALg, UGALn, and PAR. Noticeably, under the maximum load, Q-adaptive routing even improves system throughput by 3.0% compared with the optimal VALn non-minimal routing. Q-adaptive outperforms VALn routing because it only forwards packets non-minimally in intermediate groups when it is necessary. Since ADV+1 introduces the least local

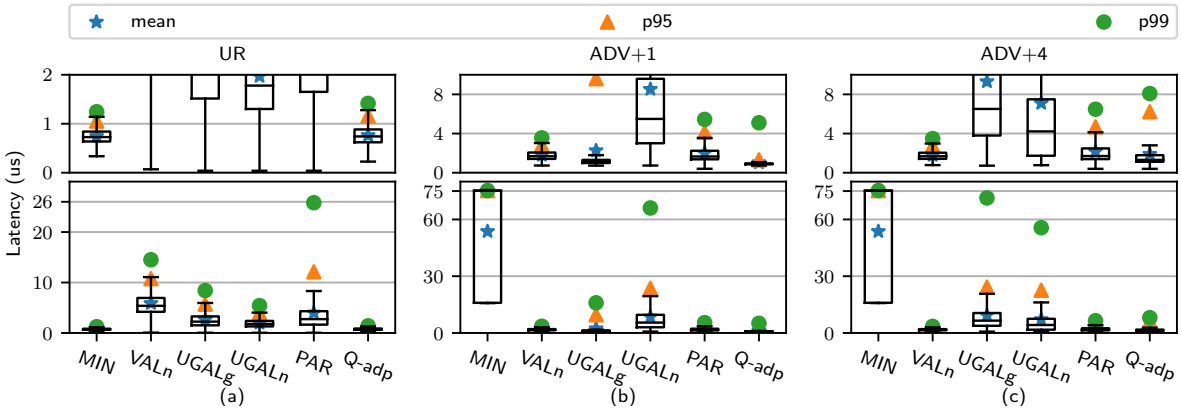

Figure 6: Packet latency distribution on the 1,056-node Dragonfly with zoom-in plot shown on top. The central box presents the latency range from $Q1$ (first quartile) to $Q3$ (third quartile) with the median shown as a black line. The upper and lower whiskers outside of the central box are $1.5 \times IQR$ above and below $Q3$ and $Q1$ respectively. Average packet latency and 95th, 99th percentile latency are also plotted.

link congestion, most of the packets should be minimally forwarded in intermediate groups, whereas VALn always reroutes packets through a random router causing network bandwidth waste. As a result, Q-adaptive uses the network more efficiently by sending packets with an average of 3.06 hops shown in Figure 5(f), which is 1.80x smaller than that of VALn routing.

Besides achieving the highest throughput, using shorter routing paths makes Q-adaptive routing deliver packets in the shortest time. In Figure 5(d), all routing methods except Q-adaptive show network congestion at the offered load of 0.45 with a sharp average packet latency increase. Under this load, the average packet latency of Q-adaptive routing is only 1.03 µs.

ADV+4 pattern: The ADV+4 traffic pattern introduces the heaviest local link congestion. Figure 5(i) shows that Q-adaptive routes packets with an average of 4.27 hops at the offered load of 0.5, which is larger than the 3.06 hops under the ADV+1 traffic pattern. This shows that Q-adaptive can reroute packets in intermediate groups with an additional hop to bypass local link congestion. As a result, Q-adaptive achieves up to 44.93% system throughput, which is only 1.69% less than the optimal VALn non-minimal routing. Q-adaptive and PAR have similar average packet latency performance shown in Figure 5(g). Both Q-adaptive and PAR routing deliver packets using less time than UGALg and UGALn.

To summarize, Q-adaptive outperforms adaptive routing algorithms with near-optimal performance. It learns to route minimally when traffic is uniform and to use intermediate groups and routers when the traffic pattern is adversarial.

5.4 Tail Latency

Figure 6 shows the packet latency distribution when the load is set to 0.8 under UR and 0.45 under ADV+i. These plots show the average packet latency with the 95th and 99th percentile latency.

UR pattern: In Figure 6(a), Q-adaptive has the smallest 99th percentile latency of 1.42 µs, which is 5.92x, 3.82x, and 18.18x smaller than that of UGALg, UGALn, and PAR respectively. Further, this 99th percentile latency is very close to the theoretic optimum, i.e., 1.25 µs achieved by the minimal routing.

ADV+1 pattern: In Figure 6(b), the 99th percentile latency of Q-adaptive routing is 5.10 µs, which is 3.12x, and 12.95x smaller than that of UGALg and UGALn respectively. UGALn performs poorly because the ADV+1 pattern introduces the least local link congestion. Under this pattern, packet rerouting in intermediate groups performed by UGALn wastes local link bandwidth and delays packet delivery. PAR achieves the 99th percentile latency of 5.44 µs, which is close to the 5.10 µs obtained by Q-adaptive. However, the latency distribution achieved by Q-adaptive is much more compact than that of PAR. In other words, Q-adaptive outperforms PAR and UGAL in terms of the overall latency distribution and tail latency.

ADV+4 pattern: In Figure 6(c), the 99th percentile latency of Q-adaptive is 8.08 µs, which is 8.83x and 6.89x smaller than that of UGALg and UGALn. While the 95th and 99th percentile latencies achieved by PAR are slightly less than those obtained by Q-adaptive, we shall point out that Q-adaptive results in a more concentrated packet latency distribution. When using Q-adaptive, 80.99% of packets have a latency of less than 2 µs, whereas only 63.69% of packets have such a low latency when using PAR.

Put together, when comparing different routing algorithms under UR and ADV+i patterns, Q-adaptive outperforms UGALg, UGALn, and PAR in terms of average latency and tail latency distribution.

5.5 Convergence

Figure 7 demonstrates how the packet latency evolves with a system starting from an empty network.

For the UR traffic pattern in Figure 7(a), Q-adaptive routing can nearly handle the lower offered load case directly with a negligible learning period. For the higher offered load case, network traffic generated at the system start time (0 µs) can quickly congest the system and leads to a dramatic packet latency surge. However, routers are capable of learning an optimal routing policy within approximately 200 µs to bring the system to a stable state with an

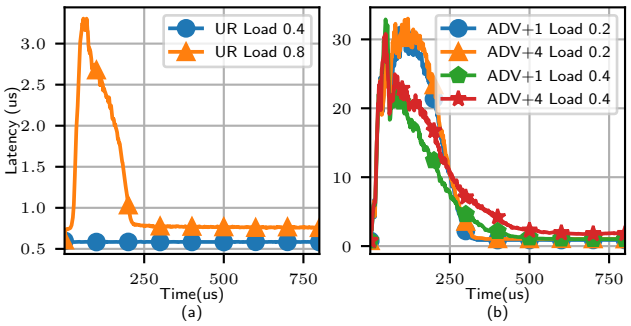

Figure 7: Convergence of Q-adaptive. Q-adaptive routing can converge under 500 µs under different traffic patterns and loads.

average packet latency of 0.75 µs. Compared with the UR pattern, ADV+1 and ADV+4 patterns require longer convergence time for routers to learn to route non-minimally as shown in Figure 7(b). Furthermore, compared with the ADV+1 traffic pattern, ADV+4 requires routers to spend some additional time to learn to alleviate local link congestion through packet rerouting in intermediate groups. Nevertheless, Q-adaptive routing can still find its optimal equilibrium for both traffic patterns under different loads within 500 µs.

In summary, Q-adaptive routing can converge within a small amount of time under both UR and ADV+i traffic patterns.

5.6 Dynamic Loads

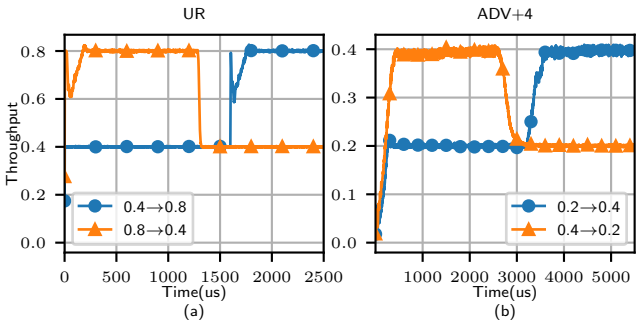

Figure 8: Q-adaptive under varying loads

Figure 8 depicts how system throughput evolves starting with an empty network and a traffic pattern with varying offered load.

In Figure 8(a), the UR traffic pattern offered load is increased from 0.4 to 0.8 at 1600 µs, and Q-adaptive needs to go through a learning period for about 156 µs to gradually adapt to the higher load. Note that the 156 µs learning period is shorter than the 200 µs learning period when the system starts from an empty network shown in Figure 7(a). When the UR offered load is decreased from 0.8 to 0.4 at 1280 µs, the system instantly responds to the change. In Figure 8(b), when the ADV+4 traffic pattern increases its offered load from 0.2 to 0.4 at 3215 µs, 455 µs is required to let Q-adaptive adjust to this change. Similarly, 440 µs is required when the pattern decreases its load from 0.4 to 0.2 at 2610 µs. Both cases require less time than the 500 µs learning period shown in Figure 7(b).

6 CASE STUDY ON 2,550-NODE SYSTEM

Now we present a scale-up study of Q-adaptive routing on a 2,550-node Dragonfly system (configuration shown in Table 1). In this case study, we evaluate Q-adaptive routing under both system-wide extreme traffic patterns (UR and ADV+1) and three common HPC communication patterns:

- 3D Stencil: stencil computation is an important class of algorithms in scientific computing and 3D Stencil is a representative one-to-many pattern [38][41]. Under this pattern, the system is arranged in a 5×10×51 3D grid and each node communicates with its six neighbors along three dimensions.
- Many to Many: this pattern is representative in applications that perform parallel Fast Fourier Transforms such as pF3D [36], NAMD [28], and VASP [18]. Under this pattern, the system is arranged in a 5×10×51 3D grid. Nodes along Z-axis are grouped into the same communicator of 51 nodes performing ***all-to-all*** operations.
- Random Neighbors: the pattern mimics computation-aware load-balancing operation performed by HPC applications such as NAMD [28]. Under this pattern, each node uniformly spreads communications with 6–20 randomly selected targets.

In this section, all the patterns are evaluated with the same hyperparameter values of $\alpha = 0.2$, $\beta = 0.04$, $\epsilon = 0.001$, $q\_thld1 = 0.05$, $q\_thld2 = 0.4$. Figure 9 shows the packet latency distribution under different traffic patterns.

UR pattern: As shown in Figure 9(a), the average packet latency of Q-adaptive routing is 0.84 µs, which is 3.24x, 2.40x, 4.67x smaller than that of UGALg, UGALn, and PAR respectively. The 99th percentile latency of Q-adaptive routing is 1.67 µs, which is significantly smaller than the adaptive routing methods. Additionally, Q-adaptive has the near-optimal performance compared with the optimal minimal routing, whose average packet latency and 99th percentile latency are 0.77 µs and 1.28 µs.

ADV+1 pattern: In Figure 9(b), the average packet latency of Q-adaptive is 0.96 µs, which is 2.33x, 12.35x, 2.35x smaller than that of UGALg, UGALn, and PAR respectively. Q-adaptive also has the smallest 99th percentile latency of 3.17 µs among all other routing methods. As a result, same as the 1,056-node system case, Q-adaptive outperforms the optimal VALn non-minimal routing, whose average packet latency is 1.75 µs and 99th percentile latency is 3.29 µs.

3D Stencil pattern: Figure 9(c) shows that Q-adaptive has the smallest average packet latency of 0.62 µs, which is 1.77x smaller than the second-best UGALg's average latency. Additionally, the 99th percentile latency of Q-adaptive, 3.08 µs, is also the smallest among all routing algorithms and is 1.31x smaller than the second-best PAR's 99th percentile latency.

Many to Many pattern: Figure 9(d) shows that Q-adaptive has the smallest average packet latency of 1.15 µs, which is 2.10x smaller than the second-best UGALg's average packet latency. The 99th percentile latency of Q-adaptive is 1.50 µs, which is 2.50x smaller than the second-best minimal routing's 99th percentile latency.

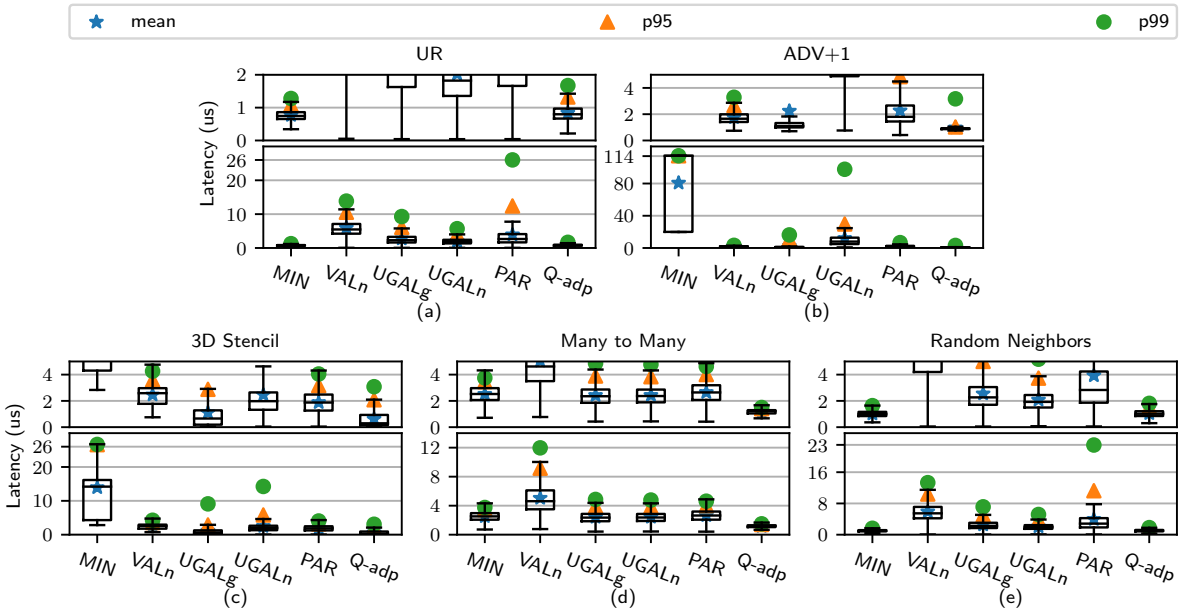

Figure 9: Packet latency distribution on the 2,056-node Dragonfly with zoom-in plot shown on top.

As a result, Q-adaptive outperforms all other routing algorithms regarding both average and 99th percentile packet latency.

Random Neighbors pattern: Under this pattern, the traffic is uniformly spread across the network. Thus minimal routing is the optimal routing solution with the smallest average packet latency of 1.01 µs and the smallest 99th percentile latency of 1.64 µs shown in Figure 9(e). However, Q-adaptive offers near-optimal performance with 1.04 µs average packet latency and 1.81 µs 99th percentile latency. Although UGALn performs best among adaptive routing algorithms, its average and 99th percentile latency are 1.99x and 2.86x larger than that of Q-adaptive routing.

Overall, we make two key observations from this scale-up case study. First, Q-adaptive routing is capable of achieving better performance than existing dragonfly routing algorithms under system-wide extreme traffic patterns and three representative HPC communication patterns. Second, Q-adaptive is capable of scaling on larger systems.

7 CONCLUSION

Dragonfly is a promising high-radix interconnect topology for large-scale HPC systems. The high path diversity and potential link congestion inherent in Dragonfly pose significant challenges when routing packets among many candidate paths. Existing adaptive routing methods use local router information (i.e., local queue occupancy) and select a path among a limited number of paths, which can lead to inefficient link usage and low system throughput. In this work, we have presented Q-adaptive routing, a multi-agent reinforcement learning based scheme, for dragonfly systems. The proposed Q-adaptive routing features by its small yet effective two-level Q-table design, along with its value updating and routing mechanisms. These techniques enable Q-adaptive routing to effectively learn to choose a proper path among all the viable paths for packet routing on Dragonfly. We have evaluated Q-adaptive routing on both 1k and 2k systems through extensive simulations. Our results indicate that Q-adaptive outperforms existing adaptive routing methods by up to 10.5% in network throughput and more than 5x reduction in packet latency. Moreover, Q-adaptive routing achieves better tail latency than the existing adaptive routing methods.

In this study, we have focused on analyzing Q-adaptive routing under system-wide traffic patterns. Given the promising results achieved in this study, part of our future work is to analyze Q-adaptive on individual applications and further to investigate inter-job interference by using Q-adaptive routing.

ACKNOWLEDGMENTS

We thank our shepherd and the HPDC reviewers for their valuable feedback and insightful comments. This work is supported in part by US National Science Foundation grants CNS-1717763, CCF-1618776.

REFERENCES

[1] Dennis Abts and Bob Felderman. 2012. A guided tour of data-center networking. *Commun. ACM* 55, 6 (2012), 44–51.
[2] Bob Alverson, Edwin Froese, Larry Kaplan, and Duncan Roweth. 2012. Cray XC series network. *Cray Inc., White Paper WP-Aries01-1112* (2012).
[3] Justin Boyan and Michael Littman. 1993. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems* 6 (1993), 671–678.
[4] Samuel PM Choi and Dit-Yan Yeung. 1996. Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *Advances in Neural Information Processing Systems*. 945–951.
[5] Sudheer Chunduri, Kevin Harms, Scott Parker, Vitali Morozov, Samuel Oshin, Naveen Cherukuri, and Kalyan Kumaran. 2017. Run-to-run variability on Xeon Phi based Cray XC systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.
[6] Caroline Claus and Craig Boutilier. 1998. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI* 1998, 746-752 (1998), 2.
[7] William J Dally et al. 1992. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed systems* 3, 2 (1992), 194–205.

[8] William J Dally and Charles L Seitz. 1988. Deadlock-free message routing in multiprocessor interconnection networks. (1988).

[9] Daniele De Sensi, Salvatore Di Girolamo, and Torsten Hoefler. 2019. Mitigating network noise on Dragonfly networks through application-aware routing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–32.

[10] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. 2012. Cray cascade: a scalable HPC system based on a Dragonfly network. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–9.

[11] Yuping Fan, Taylor Childers, Paul Rich, William Allcock, Michael Papka, and Zhiling Lan. 2021. Deep Reinforcement Agent for Scheduling in HPC. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

[12] Mario Flajslik, Eric Borch, and Mike A Parker. 2018. Megafly: A topology for exascale systems. In *International Conference on High Performance Computing*. Springer, 289–310.

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

[14] Nikhil Jain, Abhinav Bhatele, Xiang Ni, Nicholas J Wright, and Laxmikant V Kale. 2014. Maximizing throughput on a dragonfly network. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 336–347.

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

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

[17] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. 2008. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*. IEEE, 77–88.

[18] Georg Kresse and Jürgen Hafner. 1993. Ab initio molecular dynamics for liquid metals. *Physical Review B* 47, 1 (1993), 558.

[19] Martin Lauer and Martin Riedmiller. 2000. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer.

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[21] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 270–288.

[22] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. 2007. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 64–69.

[23] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. *arXiv preprint arXiv:1706.04972* (2017).

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

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

[26] Ann Nowe, Kris Steenhaut, Mohamed Fakir, and Katja Verbeeck. 1998. Q-learning for adaptive load based routing. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, Vol. 4. IEEE, 3965–3970.

[27] Leonid Peshkin and Virginia Savova. 2002. Reinforcement learning for adaptive routing. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, Vol. 2. IEEE, 1825–1830.

[28] James C Phillips, Gengbin Zheng, Sameer Kumar, and Laxmikant V Kalé. 2002. NAMD: Biomolecular simulation on thousands of processors. In *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE, 36–36.

[29] LA Prashanth, HL Prasad, Shalabh Bhatnagar, and Prakash Chandra. 2016. A constrained optimization perspective on actor–critic algorithms and application to network routing. *Systems & Control Letters* 92 (2016), 46–51.

[30] Md Shafayat Rahman, Saptarshi Bhowmik, Yevgeniy Ryasnianskiy, Xin Yuan, and Michael Lang. 2019. Topology-custom UGAL routing on dragonfly. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[31] Joao Reis, Miguel Rocha, Truong Khoa Phan, David Griffin, Franck Le, and Miguel Rio. 2019. Deep Neural Networks for Network Routing. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

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

[33] Steve Scott, Dennis Abts, John Kim, and William J Dally. 2006. The blackwidow high-radix clos network. *ACM SIGARCH Computer Architecture News* 34, 2 (2006), 16–28.

[34] Daniele Sensi, Salvatore Girolamo, Kim McMahon, Duncan Roweth, and Torsten Hoefler. 2020. An In-Depth Analysis of the Slingshot Interconnect. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 481–494.

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

[36] Charles H Still, RL Berger, AB Langdon, DE Hinkel, LJ Suter, and EA Williams. 2000. Filamentation and forward brillouin scatter of entire smoothed and aberrated laser beams. *Physics of Plasmas* 7, 5 (2000), 2023–2032.

[37] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[38] Allen Taflove and Susan C Hagness. 2005. *Computational electrodynamics: the finite-difference time-domain method*. Artech house.

[39] Nigel Tao, Jonathan Baxter, and Lex Weaver. 2001. A multi-agent, policy-gradient approach to network routing. In *In: Proc. of the 18th Int. Conf. on Machine Learning*. Citeseer.

[40] top500.org. 2020. *Top500 list*. https://www.top500.org/lists/top500/2020/11/

[41] Didem Unat, Xing Cai, and Scott B Baden. 2011. Mint: realizing CUDA performance in 3D stencil methods with annotated C. In *Proceedings of the international conference on Supercomputing*. 214–224.

[42] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. 2017. Learning to route. In *Proceedings of the 16th ACM workshop on hot topics in networks*. 185–191.

[43] Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. 2018. Learning to design circuits. *arXiv preprint arXiv:1812.02734* (2018).

[44] Xin Wang, Misbah Mubarak, Yao Kang, Robert B Ross, and Zhiling Lan. 2020. Union: An Automatic Workload Manager for Accelerating Network Simulation. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 821–830.

[45] Jeremiah J Wilke and Joseph P Kenny. 2020. Opportunities and limitations of Quality-of-Service in Message Passing applications on adaptively routed Dragonfly and Fat Tree networks. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 109–118.

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

[47] Xu Yang, John Jenkins, Misbah Mubarak, Robert B Ross, and Zhiling Lan. 2016. Watch out for the bully! job interference study on dragonfly network. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 750–760.

[48] Jieming Yin, Subhash Sethumurugan, Yasuko Eckert, Chintan Patel, Alan Smith, Eric Morton, Mark Oskin, Natalie Enright Jerger, and Gabriel H Loh. 2020. Experiences with ML-Driven Design: A NoC Case Study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 637–648.

[49] Xinyu You, Xuanjie Li, Yuedong Xu, Hui Feng, Jin Zhao, and Huaicheng Yan. 2020. Toward Packet Routing With Fully Distributed Multiagent Deep Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2020).