Deep Reinforcement Learning Based Dynamic Flowlet Switching for DCN

Xinglong Diao, Huaxi Gu, Wenting Wei, *Member, IEEE,* Guoyong Jiang, and Baochun Li, *Fellow, IEEE*

***Abstract*—Flowlet switching has been proven to be an effective technology for fine-grained load balancing in data center networks. However, flowlet detection based on static flowlet timeout values, lacks accuracy and effectiveness in complex network environments. In this paper, we propose a new deep reinforcement learning approach, called DRLet, to dynamically detect flowlets. DRLet offers two advantages: first, it provides dynamic flowlet timeout values to detect bursts into fine-grained flowlets; second, flowlet timeout values are automatically configured by the deep reinforcement learning agent, which only requires simple and measurable network states, instead of any prior knowledge, to achieve the pre-defined goal. With our approach, the flowlet timeout value dynamically matches the network load scenario, ensuring the accuracy and effectiveness of flowlet detection while suppressing packet reordering. Our results show that DRLet achieves superior performance compared to existing schemes based on static flowlet timeout values in both baseline and asymmetric topologies.**

***Index Terms*—Flowlet, deep reinforcement learning, load balancing, data center networks.**

I. Introduction

DATA center networks (DCN) often employ a multi-rooted Clos topology [1], which offers multiple equal-hop paths between two hosts. With the rapid growth of traffic in DCNs, load balancing is of great significance to achieve high performance by distributing the traffic uniformly across the multiple paths [2].

Flowlet switching is a promising technique for fine-grained load balancing [3]–[5]. As Fig. 2 shows, a flowlet is a burst of packets from the same flow, separated from other bursts by a significant time gap, called flowlet timeout. Traffic can be distributed for better load balancing by sending different flowlets from the same flow across multiple paths. Furthermore, the natural time gap between bursts reduces the risk of out-of-order packet arrivals.

One challenge that remains for existing flowlet-level load balancing is the accuracy and effectiveness of flowlet detection. This has two implications. Firstly, a static flowlet timeout value is used by the switch to detect bursts into flowlets [4], [6]. However, it is hard to match the static flowlet timeout with highly dynamic and time-varying load scenarios.

More specifically, if the static flowlet timeout value is too long, it will affect the accuracy of identifying flow bursts, thus reducing the number of flowlets and the opportunities for load balancing. On the other hand, if the flowlet timeout value is too short, it is likely to over-detect, which means that packets in the same burst may be further divided into more than one flowlet, resulting in too fine a granularity and easily aggravating packet reordering.

Another implication is that the flowlet timeout value is configured based on heuristic methods [4], [5], [7] which heavily rely on historical observations or experiences. The heuristic method usually builds mathematical models based on human understandings and strict assumptions [8]. For example, the Markov chain model in [5] certainly offers some useful insights to configure flowlet timeout values. The bad news is that the final flowlet timeout value is still set to the empirical value from their testbed, which may not apply to other network scenarios.

Considering that the realistic network environment is extremely complicated, the final load balancing performance is the result of a mixture of multiple network factors. However, the relationship between the network factors and the final performance is hard to characterize. Consequently, mathematical models of heuristics may not be applicable or even valid in practice [9]. Thus, the model can only help to improve performance in the worst case scenario [10]. Moreover, once the human understanding of the network is biased, built models are extremely prone to make decisions that stray too much from the actual optimal configuration or may even lead to network errors.

To address the above challenges, this paper proposes DRLet (Deep Reinforcement Learning-based Flowlet Switching), which uses deep reinforcement learning (DRL) to achieve dynamic flowlet switching for fine-grained load balancing in data center networks.

Highlights of our design are also two-fold. On the one hand, DRLet offers dynamic flowlet timeout values to detect bursts of packets into flowlets, which provides the foundation of being able to adapt to a variety of network load scenarios. On the other hand, flowlet timeout values are configured under the guidance of the deep reinforcement learning algorithm. In particular, our DRL agent does not rely on precisely solvable mathematical models and can dynamically adjust flowlet timeout values based on feedback from the network environment, which provides DRLet with the actual ability to adapt to a variety of network load scenarios at all times.

To make DRLet work, we have built a network model to better represent the flowlet switching problem for our DRL agent. Note that there is little information on the methods of model building in the existing literature [8], [11]. In this paper, the network model is designed based on two important

X. Diao, H. Gu, W. Wei, and G. Jiang are with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, China (email: xl.diao@stu.xidian.edu.cn; hxgu@xidian.edu.cn; wtwei@xidian.edu.cn; gy.jiang.xidian@foxmail.com).
B. Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada (email: bli@ece.toronto.edu).

principles: logicality inside the model, and simplicity of model parameters. Then, we give the state and action vectors, and the reward function of our DRL agent based on the network model.

Based on the characteristics of the constructed network model, a suitable deep reinforcement learning algorithm is required to complete the mapping from input to output within the network model. This makes decisions for the flowlet switching problem. Considering that flowlet switching is a continuous control problem, Deep Deterministic Policy Gradient (DDPG) [12] would be the suitable candidate. We have designed a load-aware exploration mechanism for DDPG to better serve DRLet's network model.

The contributions of this paper are as follows:

i) We point out the drawbacks of the fixed flowlet timeout for flowlet detection and validate the flowlet timeout mismatch problem via experimental results. To solve that, we propose DRLet, a deep reinforcement learning-based dynamic flowlet switching.

ii) To fit deep reinforcement learning into the dynamic flowlet switching problem, we build a well-designed network model to better translate the dynamic flowlet switching problem for the deep reinforcement learning agent. We also design a load-aware exploration mechanism for the DDPG algorithm we used to train the deep reinforcement learning agent.

iii) We have conducted extensive large-scale ns-3 simulations under realistic traffic workloads. These simulations demonstrate that DRLet outperforms static flowlet timeout-based flowlet switching by 46% in the baseline and can work with any routing algorithm. Additionally, small-scale emulations based on two servers were also conducted to evaluate the time overhead of the centralized control loop. Emulation results show that the millisecond-level control loop delay is acceptable for DRLet.

The remainder of this paper is organized as follows. In Section II, we introduce the background and motivation. In Section III, we present the overall design of DRLet. The established network model for the flowlet switching problem is represented in Section IV, and the corresponding DRL algorithm is shown in Section V. We evaluate the performance of DRLet in Section VI. Finally, we show some related work in Section VII, and conclude this paper in Section VIII.

II. Background and Motivation

Load balancing distributes traffic evenly across multiple paths to optimize resource utilization [2], [13]. It has two aspects: routing and granularity. Routing chooses the path, while granularity defines the load balancing unit (which is the key factor for the load balancing performance limit).

A. Promise of Flowlet-Level Granularity

Load balancing has different levels of granularity: flow [14], sub-flow [15], flowcell [16], flowlet [5], and packet [17]. The finer the granularity, the more opportunities for load balancing.

At the flow-level granularity, a flow has only one chance to balance its load. As shown in Fig. 1 (a), all packets of each flow in f1, f2, f3, and f4 are assigned to the same path. This

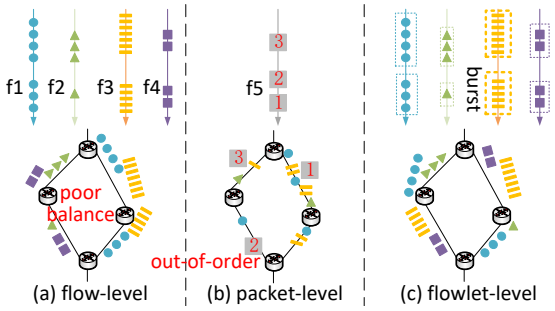

Fig. 1. Typical granularity of load balancing.

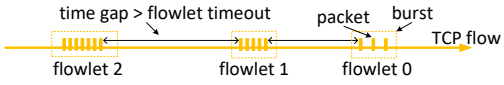

Fig. 2. The flowlet detection mechanism based on the flowlet timeout.

can cause serious problems in realistic networks, such as hash collisions [2] and head-of-line blocking.

Fine granularity improves performance by creating more opportunities for load balancing and distributing packets of the same flow over multiple paths. As shown in Fig. 1 (b), each packet of flow f5 is forwarded independently to a candidate path. Theoretically, the finest packet-level granularity can achieve the best performance of load balancing.

However, fine granularity may also cause out-of-order packet arrivals at the receiver [18]. As Fig. 1 (b) illustrates, the second packet of flow f5 arrives at the receiver earlier than the first packet of flow f5, because the former is routed to a less-loaded path, while the latter is routed to a more-loaded path. The problem is that the receiver may incorrectly assume that the out-of-order arrival of the second packet indicates packet loss of the first packet in the network. Since packet loss often implies congestion, the congestion control mechanism will be erroneously activated, resulting in the throughput degradation.

Flowcell-level load balancing [16] splits a flow into many flowcells based on bytes, and sends flowcells along different paths. Sub-flow level granularity uses the multiple parallel interfaces of the host to transmit the flow [15]. These two techniques have intermediate granularity between flow and packet, and they also suffer from severe packet reordering.

Flowlet-level granularity treats each burst of packets as a flowlet [3] and distributes flowlets of the same flow across multiple paths, as shown in Fig. 1. A flowlet is a burst of packets from the same flow, separated from other bursts by a significant time gap. A burst is detected as a new flowlet if the time gap between it and the previous burst is larger than the preset time threshold (named flowlet timeout), as shown in Fig. 2.

Flowlet technology has two benefits. First, it uses traffic bursts to create more load balancing opportunities. Second, and more importantly, it avoids packet reordering if the flowlet timeout is set to be at least the maximum delay difference between paths [4].

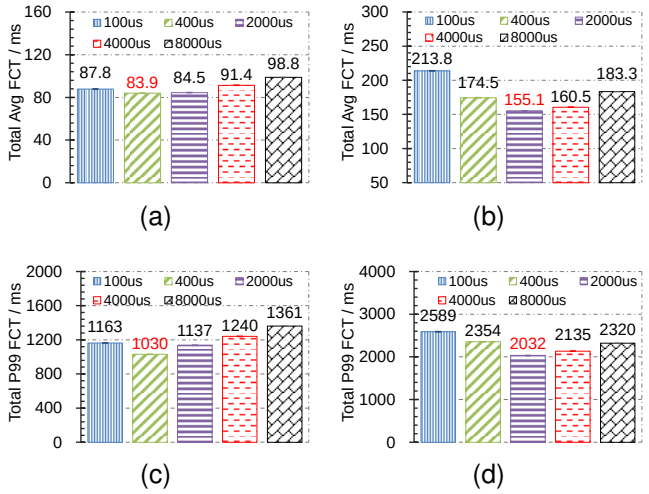

Fig. 3. Performance of LetFlow under different flowlet timeout values. (a) The average FCT under 40% load. (b) The average FCT under 60% load. (c) The P99 FCT under 40% load. (d) The P99 FCT under 60% load.

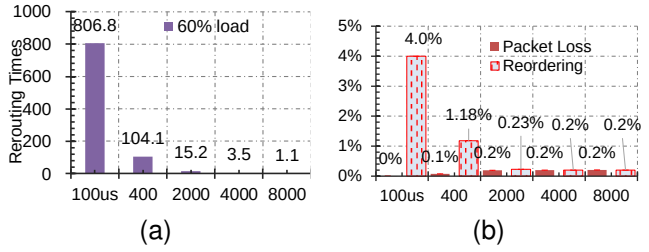

Fig. 4. LetFlow performance under different flowlet timeout values in 60% load. (a) Rerouting times (number of flowlets divided by number of flows). (b) Packet loss and reordering ratios.

B. Flowlet Timeout Mismatch

Existing flowlet-level load balancing solutions [3]–[5], [7] use a static and empirical flowlet timeout value to detect flowlets. However, the static flowlet timeout value cannot always adapt to the highly dynamic and time-varying network scenarios, causing the flowlet timeout mismatch problem.

In this section, we use simulation results to demonstrate this mismatch problem of the static flowlet timeout value under different network load scenarios. We conducted experiments using ns-3 simulator with an 8-pod FatTree with 1 Gbps link bandwidth. Fig. 3 shows the FCT performance of an existing flowlet-level approach [5] with different flowlet timeout values.

The first observation is that, using either too large or too small flowlet timeout values for flowlet detection is inappropriate and results in poor FCT (Flow Complete Time) performance. The reasons for this behavior can be analyzed from Fig. 4. With the flowlet timeout value of 8000µs, there are very few opportunities for rerouting, and the number of flowlets is only 1.1 times the original number of flows. Consequently, load balancing granularity is extremely large, which leads to the high ratio of packet drops (20 times more than that of 100µs). For the flowlet timeout value of 100µs, the opportunity for rerouting increases dramatically, which in turn optimizes the load balancing granularity and significantly reduces packet loss in the network to only 0.01%. However, it faces serious packet reordering (up to 4%), which triggers many retransmissions that drag down FCT performance. Therefore, it is difficult to make trade-offs between load balancing granularity and packet reordering when configuring flowlet timeout values.

Another observation is that, the optimal flowlet timeout value for the best FCT performance varies depending on the load. For the 60% load in Fig. 3b, the FCT decreases by 12.4%, and 15.4% when the flowlet timeout value of LetFlow changes from 8000µs to 4000 µs, 2000 µs, respectively. However, when the flowlet timeout values continue to change from 2000µs to 400µs and 100µs, the FCT increases by 12.5% and 37.8%, respectively. Compared with other values, 2000µs can be considered the optimal configuration for LetFlow's flowlet timeout value in the 60% load scenario. As observed in Fig. 3a, 400µs is the most appropriate configuration for LetFlow under 40% load, which is much different from the optimal value of 2000µs in Fig. 3b. The overall 99% FCT shown in Fig. 3c and Fig. 3d further confirms our observation.

The two observations in Fig. 3 validate the flowlet timeout mismatch problem of existing flowlet-level schemes. The mismatch problem has two main causes: first, flowlets are detected based on a static flowlet timeout value that cannot adjust to different load scenarios; and second, the flowlet timeout value is set with empirical values or heuristics [5] that may not suit the network situation. Therefore, a more efficient method is needed to achieve dynamic flowlet switching, which should be able to cope with the highly-dynamic and time-varying network environment.

Summary Flowlet switching is a promising technology for load balancing. However, static flowlet timeout value based flowlet detection faces the challenge of effectively refining granularity. In this paper, we try to use deep reinforcement learning to achieve dynamic flowlet switching for load balancing in DCN, with the aim of configuring optimal flowlet granularity in complex network environments.

III. DRLet Design

DRLet is designed to optimize load balancing performance from the perspective of flowlet granularity. In this section, we will show how DRLet provides efficient and dynamic flowlet switching with the guidance of deep reinforcement learning.

A. Overview

The deep reinforcement learning agent of DRLet lives in a centralized controller. The centralized deployment allows for convenient collection of network information and assignment of agent decisions. Through the controller, the DRLet agent interacts with the network environment periodically and optimizes its decisions during the interaction. The interaction period should be set to be relatively larger than the actual control-loop delay (which is discussed in §VI-D). For example, 30 milliseconds per interaction would be enough under a 16-pod FatTree topology.

Fig. 5 illustrates the overall workflow of the interaction between the DRLet agent and the network environment. The

network state data collected from the switches is first aggregated to the controller, where it is fed to the DRLet agent after two internal processing modules of reward calculation and state aggregation. Guided by the deep reinforcement learning algorithm, the agent outputs actions related to flowlet granularity. Then, the controller sends the actions down to the network side. In turn, the switch can identify the flowlet based on the received actions.

B. Modules in the Workflow

Next, we briefly introduce the modules involved in the interaction flow in Fig. 5.

Collect network state at the switch: The switch is required to collect and report a variety of network states to the centralized controller, including traffic injected into the network as well as the link utilization (or the queue occupancy of switches). These network state information will be further processed and used to train the DRLet agent. The reasons for choosing the above two types of network state information will be analyzed in detail in Section IV.

State aggregation: This module is responsible for processing the traffic data from each switch, and arranging them in a certain order to form a set of scalar values that will eventually be used as input to the DRLet model.

Reward calculation: This module calculates the reward value during each interaction cycle based on the network state information uploaded by the switch, resulting in a scalar value that is used to evaluate the effectiveness of the agent's action.

Network model (DRLet agent): The network model is built for the target problem of dynamic flowlet switching, which is also the core contribution of our work. The input and output of the model are elaborately designed to facilitate the agent's understanding of the target problem. The agent is responsible for completing the mapping from the input in a given format to the output in a given format, and the specific mapping rules are guided by the deep reinforcement learning algorithm. More details about the considerations behind the designed model will be displayed in Section IV.

Action assignment: The DRLet agent learns the flowlet timeout values for flowlet detection, and the network model described above will output a set of scalar flowlet timeout values. For the action assignment module, its responsibility is to distribute the flowlet timeout values to the corresponding switches on the network side. Based on the agent's action, the switch detects flowlets in the network and forwards flowlets based on a routing algorithm.

C. Flowlet Detection

Flowlet detection has been well studied in existing works [3], [4]. Like in [3], the flowlet timeout value is used by the switch to detect flowlets in this paper. When the time gap between the current incoming packet and the previous packet of the same flow exceeds the flowlet timeout value, the switch considers that the current packet starts a new flowlet.

Importantly, dynamic flowlet switching of DRLet guarantees that flowlet detection can adapt to complex network environments. This is achieved by two points. First, the flowlet timeout value used by the switch to detect flowlets is periodically updated by DRLet's controller. Second, the flowlet timeout value assigned to each switch is unique and may be different, because each switch may suffer from different network conditions.

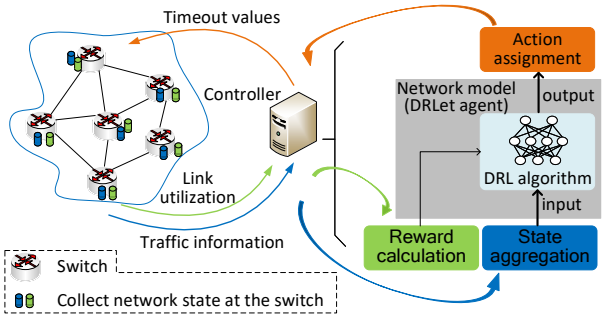

Fig. 5. System architecture of DRLet.

The flowlet timeout value assigned to a switch will be used for flowlet detection of all traffic passing through that switch, rather than being dedicated to a flow. This is because the goal of DRLet is not to accurately detect the burst of each flow to be a flowlet, but to refine the overall granularity of load balancing in the network.

D. Routing

DRLet focuses on load balancing granularity, which is independent of routing. In other words, DRLet is compatible with any routing algorithm.

Finer granularity can increase load balancing opportunities and does improve load balancing performance to some extent. Since ultimate performance is also up to the dispersion of each flowlet within the network, routing decisions are also worth exploring.

On the one hand, given the elastic nature of flowlets [5], explicit path congestion information is not necessary for routing. This allows flowlets to explore different paths and adaptively balance traffic across multiple paths [5]. Therefore, a static routing algorithm based on random or polling seems to be an appropriate choice for DRLet.

On the other hand, static routing algorithms such as hashing and random routing can achieve absolute balancing regarding the number of routing decisions. Since there is no guarantee that the granularity regarding byte size is consistent for each routing decision, absolute balancing of the number of routing decisions does not lead to load balancing of traffic in practice. Therefore, load imbalance between paths still exists, and it is necessary to introduce dynamic routing at this point.

In this paper, the different performance of static routing and dynamic routing is discussed and is displayed in Section VI-C.

IV. Network Model for Flowlet Switching

When machine learning is applied to solve the network problem, a network model with the input and the output is required to help machine learning algorithms to better understand the target network problem. Regarding deep reinforcement learning used by DRLet for dynamic flowlet

switching, the state space, action space and reward function should be specified for the DRLet agent.

A well-designed network model can improve the efficiency of the DRLet agent. Unfortunately, we notice that in the existing literature [8], [10], [11], there is little description of model design and even less analysis of the considerations behind model design.

Model design is one of the key elements in this paper. Two principles are concluded for our model design: i) The network state elements involved in each of the state, action and reward of the DRL agent should be logically related. ii) The impact of data dimension and number of parameters on the efficiency of machine learning algorithms should be fully considered. In addition, the network features we chose are commonly supported by major switch vendors [19]–[22].

A. Action Space

In this paper, the target problem of dynamic flowlet switching is to decide how to divide traffic into flowlets. Considering that traffic mostly appears in bursts and there is a relatively obvious time gap between bursts, using a flowlet timeout value to detect bursts and divide them into flowlets would be a simple and reasonable approach [3], [4]. Even so, there are many possible forms of action space for the DRLet agent.

Due to the unique transmission process of each flow in the network, the time gap and duration of bursts vary among flows. Therefore, it would be good if the DRLet agent configures a unique flowlet timeout value for each flow. However, this will induce two problems. First, the number of simultaneously existing flows in the network is not fixed, which will result in the action of the DRLet agent being a variable-length sequence (which is not supported by existing deep reinforcement learning algorithms). Second, the number of simultaneously existing flows in the network is very huge, which will lead to a very large number of parameters for the action of the DRLet agent. Unfortunately, it is not conducive to the convergence of deep reinforcement learning algorithms.

Using the same flowlet timeout value to divide flowlets for all flows in the network as in LetFlow [5] and CONGA [4] can indeed minimize the number of action parameters and avoid the problem of variable-length action sequences. However, the same flowlet timeout value obviously cannot be applied to the burst state of all flows in the network, which will reduce the accuracy of flow burst detection and the effectiveness of flowlet division, as validated in Section II-B.

With the goal of refining the overall load balancing granularity in the network, DRLet adopts a compromise approach to simultaneously balance the effectiveness of flowlet detection with the number of action parameters. In this paper, the topological features of data center networks are used to design the action space of DRLet agent. As shown in Fig. 6, for the 4-pod FatTree [1] topology, there are four equal-hop paths between Src and Dst host pairs, and the following paths are determined once the edge and aggregate switches have made their routing decisions. This indicates that flowlet detection is only required at the edge and aggregate layers. Because there is only one candidate for each hop of the second half of the path, load balancing, or flowlet detection no longer works.

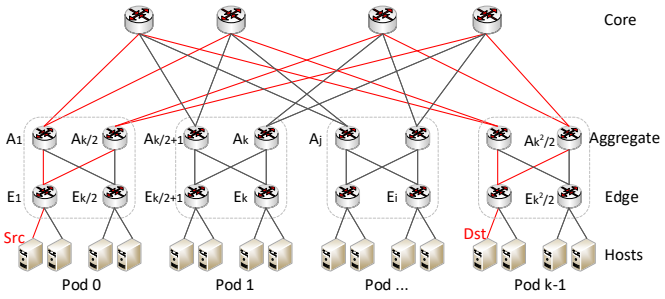

Fig. 6. The architecture of k-pod FatTree topology (k=4).

Therefore, the insight is that, the DRLet agent outputs the flowlet timeout values only for the edge and aggregate switches. All flows entering the same switch will be divided into flowlets based on the same flowlet timeout value assigned to that switch. This makes sure the effectiveness and accuracy of detecting flow bursts into flowlets on the one hand, and reduces the number of output parameters to a large extent on the other hand.

The action vector of the DRLet agent is:

$$\vec{a} = \left[a_{E_1}, \cdots, a_{E_j}, \cdots, a_{E_{\frac{k^2}{2}}}, a_{A_1}, \cdots, a_{A_j}, \cdots, a_{A_{\frac{k^2}{2}}}\right] \tag{1}$$

where a_{E_j} and a_{A_j} represent the flowlet timeout values that will be assigned to the jth edge switch and the jth aggregate switch, respectively. There are k^2 elements in the action vector under k-pod FatTree topology.

B. State Space

What needs to be considered when constructing the state space is which network state elements affect solving the target problem. Theoretically, it is traffic and node connection information that are the most primitive and fundamental influencing factors.

Detailed information about the transmission state of each flow is helpful for accurately dividing flow bursts into flowlets. However, observing the state information on a flow-by-flow basis will incur a huge measurement overhead. Moreover, simply employing the transmission state of each flow as the state of the DRLet agent will face similar problems as in the previous section: i) the state of the DRLet agent will be a variable-length sequence, and ii) the high dimension of the state. Unfortunately, these problems have a negative effect on the data processing of deep reinforcement learning.

A compromise approach for state space design was adopted in this paper. Aggregated traffic between edge switch pairs is observed and reported to the centralized controller, and will be used as the state of the DRLet agent after the process of State Aggregation. This design of the state reduces the data dimension as much as possible to enhance the efficiency of the DRLet agent while protecting the logical relationship to the action.

Deep reinforcement learning mostly uses deep neural network to fit the value function or policy. However, deep neural networks are not good at handling the input data of graph structure. Therefore, the connection relationship between each

network node is difficult to understand by deep reinforcement learning.

To this end, topological information of the network is not directly introduced into the state of the DRLet agent, which is indirectly hidden in the data sequence of the aggregated traffic between edge switches. This means that the network model designed for dynamic flowlet switching in this paper cannot adapt to real-time changes in network topology (e.g., link failure and recovery).

The state vector of the DRLet agent is:

$$\vec{s} = \begin{bmatrix} s_{(E_1,E_2)}, \cdots, s_{(E_1,E_j)}, \cdots, s_{(E_1,E_{\frac{k^2}{2}})}, \\ \vdots \quad \cdots \quad \vdots \quad \cdots \quad \vdots \\ s_{(E_i,E_1)}, \cdots, s_{(E_i,E_j)}, \cdots, s_{(E_i,E_{\frac{k^2}{2}})}, \\ \vdots \quad \cdots \quad \vdots \quad \cdots \quad \vdots \\ s_{(E_{\frac{k^2}{2}},E_1)}, \cdots, s_{(E_{\frac{k^2}{2}},E_j)}, \cdots, s_{(E_{\frac{k^2}{2}},E_{\frac{k^2}{2}-1})} \end{bmatrix} \tag{2}$$

where $s_{(E_i,E_j)}$ denotes the traffic going from the ith edge switch to the jth edge switch ($i, j \leq \frac{k^2}{2}$ & $i \neq j$). There are $\frac{k^2}{2}(\frac{k^2}{2}-1)$ elements in the state vector under k-pod FatTree topology.

C. Reward Function

Flowlet technology increases load balancing opportunities by refining the granularity, which helps to optimize the balancing performance. However, fine-grained granularity will inevitably lead to packet reordering, which in turn degrades network performance. To this end, the objectives of the DRLet agent are twofold, focusing on the performance optimization of load balancing brought by flowlet switching, and suppressing the negative impact caused by out-of-order packets.

In this paper, the objective of load balancing is converted into "minimizing the load variance between links", so that it can be understood by the DRLet agent. The corresponding reward function for load balancing is constructed based on the mean squared difference of the utilization of all links between switches:

$$r_{LB} = \log |\sum\nolimits_{i=1}^{N_{link}} \frac{(u(i)-\bar{u})^2}{N_{link}}| \tag{3}$$

where N_{link} is the number of links between the switches (i.e., $\frac{k^3}{2}$ bidirectional links under k-pod FatTree), $u(i)$ is the utilization of the ith link, and $\bar{u}$ is the mean value of the utilization of all links. It can be seen that the smaller the difference in link utilization, the larger the reward of load balancing.

Out-of-order packets could accidentally trigger the congestion control mechanism and consequently disrupt the normal transmission of the sender, which ultimately imposes a negative impact on the sender's flow rate. Considering that the flow rate is eventually reflected in the traffic injected into the network, the impact of packet reordering on the flow rate can be evaluated through the variation in traffic injected into the network.

Specifically, the impact of packet reordering is evaluated via the ratio between the traffic injected into the network of two consecutive time steps. Note that we still observe the aggregated traffic between edge switch pairs to avoid the huge measurement overhead of flow-by-flow observations. The corresponding reward function to evaluate the negative impact of packet reordering is:

$$r_{reordering} = \log \frac{\sum \vec{s_t}}{\sum \vec{s_{t-1}}} \tag{4}$$

where $\vec{s_t}$ and s_{t-1} are the observed states of two consecutive time steps, respectively. The total traffic injected into the network is roughly calculated by the sum of the elements in the observed state as shown in Equation (2). If the packet reordering has less negative impact, the flow rate of the host as well as the traffic injected into the network will gradually increase. Then the ratio between throughput of two time steps will exceed 1. Finally, a positive reward value will be obtained. Otherwise a negative reward value.

Total reward function is as follows:

$$r = \alpha \cdot r_{LB} + \beta \cdot r_{reordering} \tag{5}$$

where both α and β are configured to 0.5 to balance the optimization objectives of load balancing and packet reordering.

Finer granularity results in high reward values for load balancing but low reward values for packet reordering. Conversely, coarser granularity results in high packet reordering reward values and low load balancing reward values. Therefore, the advantage of dual objectives is that the flowlet granularity can be dynamically adjusted based on feedback from load balancing and packet reordering, avoiding the granularity to be extremely large or small.

V. Deep Reinforcement Learning Algorithm

Deep reinforcement learning employs deep learning for feature representation of states and construction of value function, such as the pioneering work named DQN in [23]. However, DQN cannot solve problems with large-scale actions or continuous action values. To this end, DDPG [12] is later proposed to use the actor network with policy-based learning to make up for the shortcoming of DQN.

For the flowlet switching problem, the established network model in Section IV involves the continuous-valued state space and the continuous-valued action space, and is therefore well suited to be solved using DDPG algorithm.

However, simply applying DDPG algorithm to the flowlet switching problem may not bring the expected results. The reason is that reinforcement learning mostly adopts online learning, while DDPG may generate unpredictable actions under random noise-based exploration. Unfortunately, this is highly likely to induce serious network threats [24], [25].

In addition, the network load is not always static and will likely change over time. Therefore, we design a load-aware exploration mechanism. Briefly, to better guide training the deep reinforcement learning algorithm, the pre-defined flowlet timeout values adapted to the network load are used as benchmarks during the exploration process. Note that the roughly-measured flowlet timeout values are enough to lead to quite good learning for DRLet, thus introducing less measurement overhead. The specific exploration mechanism is as follows:

Algorithm 1: The online learning of DDPG with load-aware exploration mechanism.

1 Randomly initialize critic network $Q(\cdot)$ and actor network $\mu(\cdot)$ with weights $\boldsymbol{\theta^Q}$ and $\boldsymbol{\theta^\mu}$ respectively;
2 Initialize target critic network $Q'(\cdot)$ with $\boldsymbol{\theta^{Q'}} \leftarrow \theta^Q$, and target actor network $\mu'(\cdot)$ with $\boldsymbol{\theta^{\mu'}} \leftarrow \theta^\mu$;
3 Initialize replay buffer R ;
4 **for** *each episode* **do**
5 Initialize a random process (noise) N ;
6 Receive the initial observed state $\boldsymbol{s_1}$;
7 **for** $\boldsymbol{t}$ = *1 to* $\boldsymbol{T}$ **do**
8 Apply the **load-aware exploration mechanism** to obtain the action:
$$\boldsymbol{a_t} = \begin{cases} a_{base} + N_t & \text{p} \\ \mu(s_t|\theta^\mu) + N_t & \text{1-p} \end{cases} \qquad p = \frac{1}{10t}$$
9 Execute action $\boldsymbol{a_t}$, observe reward $\boldsymbol{r_t}$ and new state $\boldsymbol{s_{t+1}}$;
10 Store transition sample (s_t, a_t, r_t, s_{t+1}) in R;
11 Sample a mini-batch of M transitions (s_i, a_i, r_i, s_{i+1}) from R;
12 Compute target value for critic network $Q(\cdot)$
$$y_i = r_i + \gamma \cdot \ Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$
13 Update critic network by minimizing the loss:
$$L = \frac{1}{M}\sum_{i=1}^{M}(y_i - Q(s_i, a_i)|\boldsymbol{\theta^Q})^2$$
14 Update the actor policy using the sampled policy gradient: $\nabla_{\theta^\mu} J \approx$
$$\frac{1}{M}\sum_{i=1}^{M} \nabla_{a_i} Q(s_i, a_i)|_{a_i=\mu(s_i)} \nabla_{\theta^\mu} \mu(s_i|\boldsymbol{\theta^\mu})$$
15 Update the weights of the target networks:
$$\boldsymbol{\theta^{Q'}} \leftarrow \tau\boldsymbol{\theta^Q} + (1-\tau)\boldsymbol{\theta^{Q'}}$$
$$\boldsymbol{\mu^{Q'}} \leftarrow \tau\boldsymbol{\mu^Q} + (1-\tau)\boldsymbol{\mu^{Q'}}$$
16 **end**
17 **end**

1) Based on the observed state (i.e., traffic statistics between switch pairs) of each time step, the current load intensity of the network is estimated by calculating the ratio of the average throughput of traffic to the link bandwidth.
2) According to the estimated load intensity, the optimal flowlet timeout value for each load intensity is selected as the basic action a_{base}. There are two points to note: first, the optimal flowlet timeout value is the empirical result we have obtained in our experiments; second, all the elements in a_{base} have the same value, which means that the same optimal flowlet timeout value is used for all switches.
3) At the beginning of each time step, the agent selects the basic action with p probability and the original action with $(1-p)$ probability.
4) The probability p decreases with time grows (e.g., setting $p = \frac{1}{10t}$), which means that the empirical flowlet timeout values become less and less useful as the agent continues to mature.

Algorithm 1 represents the online-learning process of the DDPG algorithm with the load-aware exploration mechanism. After the initialization of the four networks and the replay buffer (line 1-3), the algorithm starts the learning of each episode (line 4). At the beginning of each step in each episode, the agent obtains the action a_t based on the load-aware exploration mechanism (line 8). After the action a_t is executed, the resulting sample is stored into the replay buffer (line 10). Then, a batch of samples is selected from the replay buffer (line 11) to update the parameters of the actor and policy networks (lines 12-14). To improve the learning stability, the two target networks are also updated, while at a slow rate τ (line 15).

VI. Evaluation

Extensive large-scale ns-3 [26] simulations and small-scale emulations have been conducted to evaluate the performance of DRLet.

A. Simulation Methodology

To run reinforcement learning algorithms in network environment, an extension module named ns3-gym [27] is used in our ns-3 simulation, which enables the data interaction between ns-3 [26] and OpenAI Gym [28].

An 8-pod FatTree topology [1] is used in the simulation, containing 128 hosts, 80 switches. Besides, the actor and critic networks in Algorithm 1 both use the 2-layered, fully-connected neural network which includes 992 and 64 neurons in the first and second layer, respectively. Other parameter settings in our simulation include: the link bandwidth of 10 Gbps, the packet size of 1400 bytes, the RED queue with the minimal threshold of 65 packets and the maximal threshold of 250 packets, and the agent's time step of 50 ms. Note that, DCTCP [29] is used as the default congestion control algorithm in the simulation.

Traffic: In the simulation, communication between hosts is simulated by constructing flow-level traffic. Three key points are considered in our traffic generator: i) Communication pattern: All-to-all pattern was adopted in the simulation, in which each host randomly select a different host as the destination host. ii) Traffic pattern: The widely-used Web Search [29] workloads and Data Mining workload [30] are used as the traffic pattern in our simulation, which are extracted from realistic data centers. iii) Load intensity: For any source-destination pair, the communication interval obeys an exponential distribution. The smaller the interval, the more flows are generated, which in turn leads to higher load intensity. In the simulation, different load intensities are constructed by changing the value of the communication interval.

Schemes Compared: Recently proposed load balancing approaches [31], [32] mostly focus on the perspective of routing. DRLet aims to optimize the load balancing granularity with DRL-based dynamic flowlet switching. It might be unfair to directly compare DRLet with these approaches, because it's hard to tell whether the performance improvement/degradation comes from the routing or the granularity. Therefore, the control variable approach is used in our experiments.

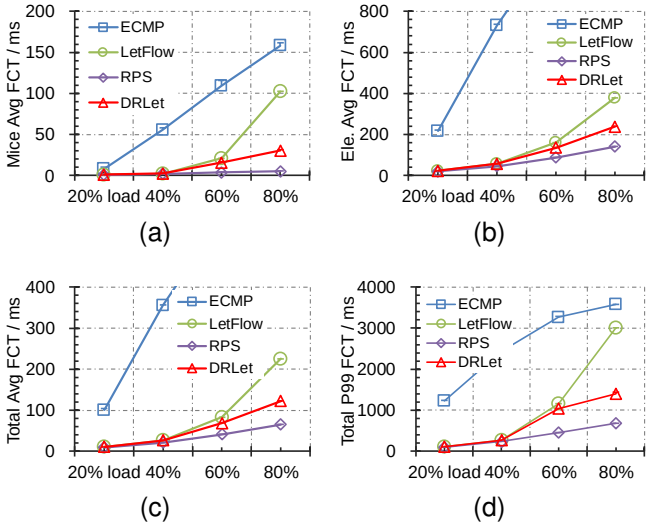

Fig. 7. FCT statistics under the Web Search workload in symmetric topology. (a) Mice avg FCT. (b) Elephant avg FCT. (c) Total avg FCT. (d) Total P99 FCT.

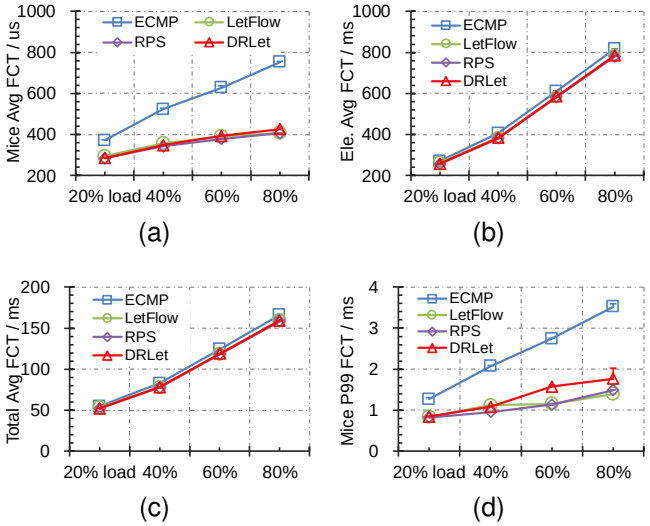

Fig. 8. FCT statistics under the Data Mining workload in symmetric topology. (a) Mice avg FCT. (b) Elephant avg FCT. (c) Total avg FCT. (d) Mice P99 FCT.

First, in Section VI-B, we evaluate the effectiveness of DRLet's granularity by setting the routing algorithm as a control variable and the schemes with different granularity as independent variables. At this point, the simplest random routing is used, and the compared schemes include ECMP [14] at flow-level granularity, LetFlow [5] based on static flowlet switching, and a packet-level approach called RPS [17].

Then, to explore the compatibility of DRLet with different routing algorithms, we use DRLet as the control variable and set different routing algorithms as independent variables in Section VI-C. At this point, DRLet will work with different routing algorithms, including the CONGA [4] routing algorithm based on global congestion awareness, the DRILL [33] routing algorithm based on local congestion awareness, and the random routing algorithm [17] without congestion awareness.

Metrics: The flow completion time is one of the important metrics. We show the average FCT for mice flows smaller than 100 KB, elephant flows larger than 100 KB, and the overall flows simulated. Additionally, the 99% FCT of mice flows and overall flows are also presented. Moreover, some essential network states inside the stack are observed in the simulation for a deep dive on the reasons behind the performance. Note that our simulation results are based on two runs.

B. Effectiveness of Dynamic Flowlet Switching

1) Baseline: We first present the performance comparison between DRLet and other different-granularity schemes under the symmetric topology.

Under the Web Search workload:

Fig. 7 shows the performance under the Web Search workload. For mice flows smaller than 100KB shown in Fig. 7a, DRLet outperforms ECMP and LetFlow by more than 81% and 70%, respectively, but is about 4.9 times inferior to RPS.

For elephant flows over 100KB shown in Fig. 7b, DRLet is superior to ECMP and LetFlow by about 88% and 37%, respectively, and is 69% inferior to RPS.

Regarding the overall average FCT shown in Fig. 7c, DRLet is better than ECMP and LetFlow by about 84% and 46%, respectively, and worse than RPS by about 88%.

Regarding the 99% FCT of all flows shown in Fig. 7d, DRLet achieves about 61% and 54% FCT reductions than ECMP and LetFlow, respectively, and underperforms RPS by about 1.3 times.

Under the Data Mining workload:

Fig. 8 presents the performance under the Data Mining workload. For mice flows less than 100KB shown in Fig. 8a, DRLet outperforms ECMP by about 43%, and underperforms LetFlow and RPS within 5%.

For elephant flows over 100 KB shown in Fig. 8b and the overall average FCT shown in Fig. 8c, DRLet is better than ECMP by about 4%, and achieves comparable performance to LetFlow and RPS.

Regarding the 99% FCT of mice flows shown in Fig. 8d, DRLet outperforms ECMP by about 50%, and inferior to LetFlow and RPS by about 26% and 19%, respectively.

Analysis:

It can be seen that all fine-grained schemes significantly outperform flow-level ECMP regarding FCT performance, because fine-grained granularity increases the chance of rerouting, thus solving the serious hash collision problem faced by flow-level granularity. Packet is the most fine-grained load balancing granularity, where each packet has the opportunity for rerouting. Therefore, compared to other schemes, RPS has the best FCT performance in almost all network scenarios.

The flowlet timeout value used for LetFlow in the simulation is configured to 200µs, which is a fair value for both workloads and different load intensities, selected from our previous extensive LetFlow measurements. As shown in Fig. 7, LetFlow achieves comparable performance to DRLet in low and medium load scenarios, and is significantly inferior to DRLet in FCT performance when the load intensity increases. This is because DRLet enables dynamic flowlet switching,

while LetFlow faces the flowlet timeout mismatch problem (as validated in Section II-B). In DRLet, the flowlet timeout values used by the switch to detect flowlets are dynamically assigned by the DRLet agent based on load awareness. This makes sure that DRLet can dynamically optimize the overall load balancing granularity of traffic in any network scenario.

Note that the performance improvement of DRLet over LetFlow is not a sweet spot that we deliberately set in the simulation, but is determined by the intrinsic problem of flowlet timeout mismatch. Even if we configure the static flowlet timeout value of LetFlow to be adapted to the higher load intensity in both workloads, it still faces the flowlet timeout mismatch problem and performs less well than DRLet in low-load scenarios.

It is also noted that the improvement in DRLet's performance in Data Mining workload is not as significant as in WEB workload. The difference between ECMP and other fine-grained schemes is obvious regarding the average FCT of mice flows, while the difference between different fine-grained levels is not significant, as shown in Fig. 8a. Regarding the average FCT of elephant flows, the performance difference between different granularity is not large, as shown in Fig. 8b.

The reason is that the long-tail characteristic is more obvious in the Data Mining workload. A very large part of the flows (more than 80%) are mice flows smaller than 100KB. Since the number of mice flows is high, granularity refinement brings some performance improvement and outperforms ECMP obviously. However, mice flows are small and survive very shortly in the network. Thus, different fine-grained granularity fail to present significant performance differences. The traffic load in Data Mining is contributed by a smaller fraction of extremely large bytes of elephant flows. Since network bandwidth resources are almost all occupied by elephant flows, no significant performance improvement is visible even with granularity refinement.

2) Asymmetry: Link failures and switch failures are very common in large-scale data centers [34], and it is reported that there are up to 40 link failures per day [35]. Unfortunately, link failures break the symmetry of the topology and will pose a serious threat to load balancing. In this section, the impact of topology asymmetry caused by link failures is investigated. In the simulation, the asymmetry scenario is constructed by decreasing the link speed of an arbitrarily chosen 5% of the in-network bidirectional links.

Given that the performance of each scheme in the asymmetric topology is similar to that in the baseline topology, Fig. 9 presents the overall average FCT metric in the asymmetric topology and omits the results of the mice average FCT and the elephant average FCT. Under the Web Search workload, DRLet outperforms ECMP, LetFlow and RPS by about 51%, 7% and 13%, respectively. Under the Data Mining workload, DRLet is superior to ECMP and RPS by about 6.5% and 2%, respectively, and is within 7% of LetFlow.

Compared to the symmetric topology, the degree of performance degradation in the asymmetric topology is different for each scheme. Fig. 10 shows the quotient of the total average FCT in asymmetric topology and that in symmetric topology, which reflects the performance degradation due to asymmetry.

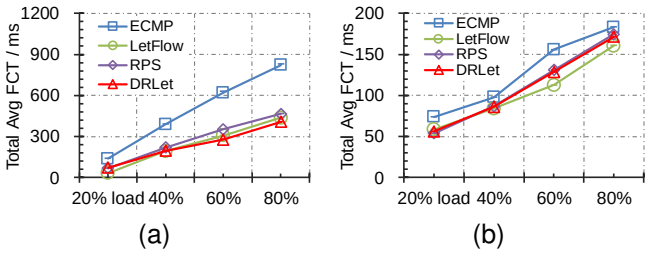

Fig. 9. Total avg FCT statistics under the asymmetric topology. (a) Web Search workload. (b) Data Mining workload.

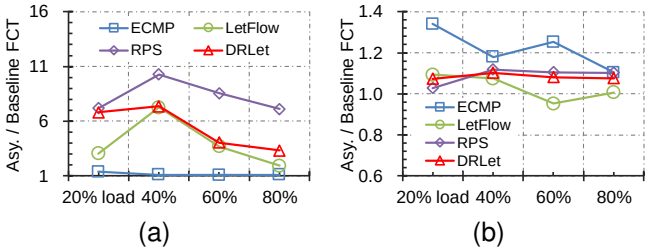

Fig. 10. Ratio of total avg FCT in the asymmetric topology to that in the symmetric topology. (a) Web Search workload. (b) Data Mining workload.

It can be observed that under the WebSearch workload, the average FCT value of RPS increases to more than six times of that under baseline. DRLet and LetFlow also show a non-negligible performance degradation when in asymmetry, but the performance change of ECMP is much smaller. Under the Data Mining workload, the performance degradation of all schemes under asymmetry is much smaller due to the less bursty traffic.

Analysis:

When packets are sent to a failed link, there is a high probability of causing congestion and resulting in packet loss, which in turn negatively affects the flow rate of the sender. Therefore, compared to baselines, the average FCT of each scheme has more or less performance degradation under asymmetry.

Congestion on the failed link will not spread to other links due to the RED queue management. ECMP employs flow-level granularity, and only a small fraction of flows will be hashed to the equal-cost path which contains the failed link. Coupled with the fact that, for ECMP in high-load scenarios, the network is saturated and heavily congested. Consequently, most flows are transmitted in a network environment that is not significantly different from that in the symmetric topology. Therefore, the impact of link failures is relatively small for ECMP, leading to a minimal variation in FCT metric.

For fine-grained load balancing, the finer the granularity, the more opportunities for load balancing, and thus the more likely that packets from each flow will encounter the failed-link paths. In other words, the negative impact of link failures on performance depends directly on the granularity of load balancing: the finer the granularity, the more significant the degradation of the FCT metric. Therefore, RPS of packet-level granularity is most negatively affected by link failures. Due to that, DRLet optimizes the overall flowlet granularity based

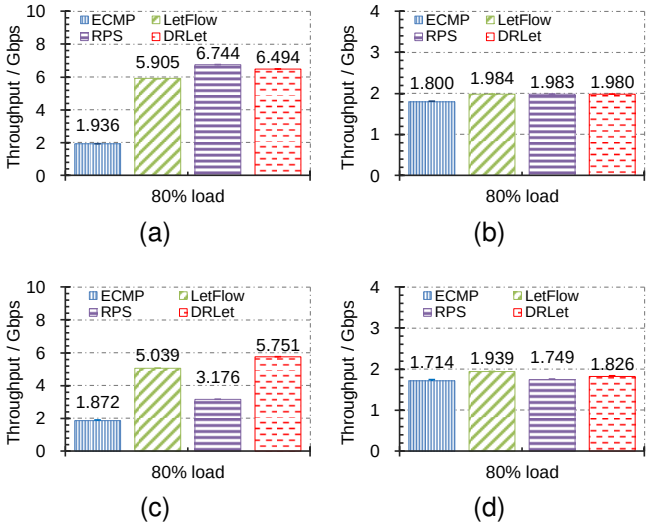

Fig. 11. The average throughput statistics at 80% load under different scenarios. (a) Under Web Search in symmetry. (b) Under Data Mining in symmetry. (c) Under Web Search in asymmetry. (d) Under Data Mining in asymmetry.

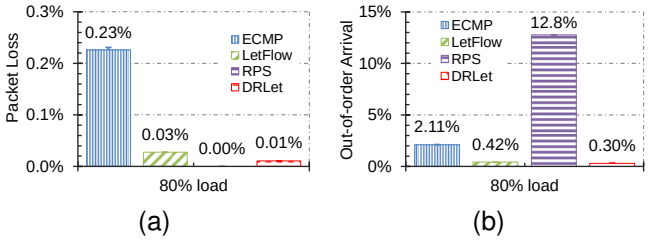

Fig. 12. Packet loss in the network and reordering at the receiver at 80% load under Web Search in symmetric topology. (a) Packet loss. (b) Out-of-order packet arrival.

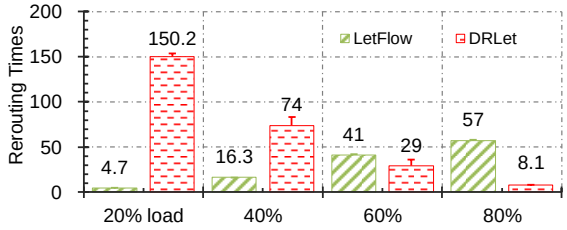

Fig. 13. Rerouting times lead by flowlets under the Web Search workload in the symmetric topology.

on dynamic flowlet timeout values. DRLet is more affected by link failures compared to the static-flowlet timeout-based LetFlow.

3) Deep Dive: **Bandwidth Utilization:** The network will reach saturation if the load is too high and the network can't keep up. Fig. 11 shows the average throughput in saturation state for different scenarios. This can be used to analyze bandwidth utilization.

ECMP suffers from severe congestion, and the average throughput is low, with bandwidth utilization not exceeding 20%. RPS has the best granularity, with an average throughput of up to 6.7 Gbps. However, in asymmetry, RPS' throughput and bandwidth utilization drop significantly due to severe packet reordering.

The flowlet granularity approximates RPS regarding bandwidth utilization, and degrades much less in asymmetry (e.g., about 15% for LetFlow) due to the tolerance of packet reordering. Thanks to dynamic flowlet switching, DRLet improves bandwidth utilization by 10% over LetFlow under Web Search in the baseline, and its performance drops by only 11% in the asymmetric topology.

Due to the obvious long-tail characteristics of the DataMining workload, fine-grained load balancing does not show performance benefits regarding bandwidth utilization, as the same analysis in Section 6.2.1 shows.

Packet Loss and Reordering: Since packet order affects packet processing latency, we show the packet loss and reordering statistics in Fig. 12. The reordering metric may be exaggerated because the statistic is counted before TCP buffer's reordering in our simulations.

As observed, ECMP faces the most severe packet loss due to the heavy congestion caused by flow-level granularity. Once the first-sent packets in flight are dropped due to congestion, subsequent in-flight packets will be recognized as out-of-order packets by our reordering counter, as shown in Fig. 12b.

The packet-level granularity results in the least congestion in the network and nearly no packet drops even at 80% load. However, delay differences between paths still exist. RPS encounters the most packet reordering, e.g., 12.8% in our simulation, because per-packet rerouting easily leads to the out-of-order arrival of packets at the receiver.

Flowlet switching treats each burst as a flowlet, which refines the granularity and leads to much less packet loss. Besides, the natural time gap between bursts offsets the delay differences between paths, which largely alleviates packet reordering. Note that DRLet does a better job at packet loss and reordering than LetFlow. This is because DRLet uses dynamic flowlet timeout values, considering both the positive rewards of granularity refinement and the negative rewards of reordering.

Rerouting Times: Fig. 13 shows the rerouting times brought about by flowlet granularity. LetFlow's rerouting times increase with load intensity. This is because traffic is more bursty in high-load scenarios. LetFlow still detects flowlets based on a static flowlet timeout value of 200 microseconds. This means that LetFlow has more opportunities to detect flowlets.

Although the number of rerouting opportunities of DRLet decreases with the increase of load intensity, its FCT performance improves as shown in Fig. 7. The reason is that DRLet is load-aware and can dynamically decide the appropriate flowlet timeout value. For example, it can assign a relatively larger flowlet timeout value to detect flowlets in high-load scenarios to optimize the overall load balancing performance of the traffic.

Different Weights for The Reward: Alpha and beta are the weights of load balancing reward and packet reordering reward in the final reward function for DRLet, respectively. We take (0.5, 0.5) as the default (α, β) combinations in our

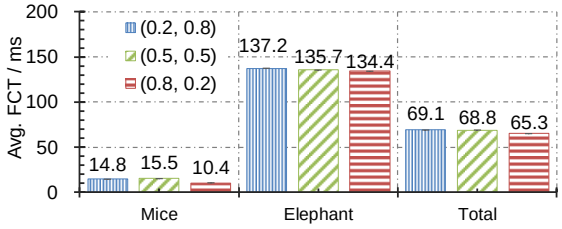

Fig. 14. Different combinations of (α, β) in the reward function of DRLet.

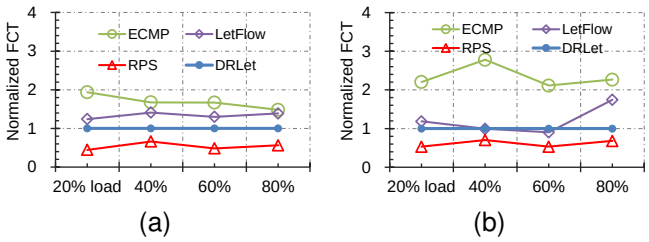

Fig. 15. FCT statistics (normalized to DRLet) under mixed workloads in symmetric topology. (a) Total avg FCT. (b) Total P99 FCT.

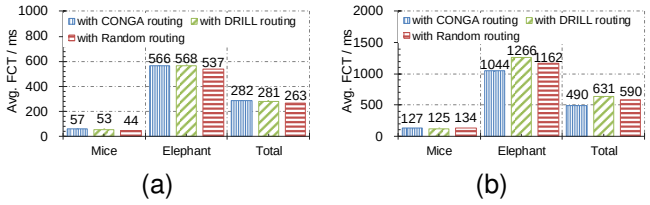

Fig. 16. FCT statistics of DRLet using different routing algorithms at 80% load under the Web Search workload. (a) Under symmetry. (b) Under asymmetry.

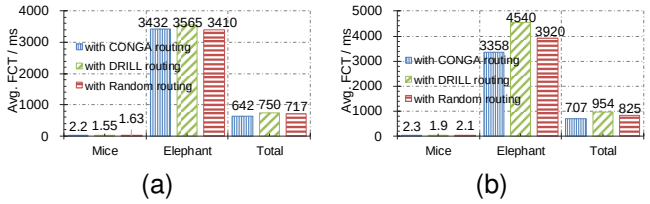

Fig. 17. FCT statistics of DRLet using different routing algorithms at 80% load under the Data Mining workload. (a) Under symmetry. (b) Under asymmetry.

simulation. Fig. 14 shows the performance comparison of DRLet under different (α, β) combinations at 60% load of Web Search workload in the baseline topology. For the combination of (0.2, 0.8), the contribution of reordering reward contributes is large in the final reward, while the contribution of load balancing reward is much smaller. As a result, the DRLet agent provides relatively large flowlet timeout values, and thus, the load balancing granularity is relatively large, which leads to slight degradation in FCT performance. Regarding the combination of (0.8, 0.2), the final reward focuses more on load balancing. Therefore, the DRLet agent will bring more fine-grained flowlet granularity, leading to about 5% improvement in FCT performance.

Mixed Workloads: We have evaluated the FCT performance of DRLet under the mixed workloads [36], which are common in realistic data center networks. The results are shown in Fig. 15. DRLet consistently outperforms other load balancing approaches, except for the packet-level RPS. As shown in Fig. 15a, DRLet reduces the overall average FCT by 30%-40% compared to LetFlow, and more than 60% compared to ECMP. However, DRLet is about 50% worse than the packet-level RPS. Fig. 15b shows a similar performance comparison as Fig. 15a.

C. Compatibility for Routing

DRLet works with any routing algorithm, so we'll look at how it performs with different options. We'll focus on three types of routing algorithms: 1) The random-based option, like RPS [17]. The router randomly selects one of the candidate output ports for routing. 2) The dynamic option based on global congestion awareness, such as CONGA [4] and Hermes [31], selecting the least-congested path for routing. 3) The dynamic option based on local congestion awareness. Like DRILL [33], the router selects the least-congested output port between two randomly selected candidate ports and the historical port.

Since Hermes is an edge-based routing algorithm, we chose CONGA as the compared routing algorithm in this section. Moreover, traffic is more bursty at high bandwidth, making it hard to find a suitable value of parameter τ for the Discounting Rate Estimator algorithm in CONGA. Therefore, the simulations in this section are performed at 1 Gbps bandwidth.

Fig. 16 and Fig. 17 show the FCT performance of DRLet with different routing algorithms in different network scenarios. Note that a highly asymmetric topology in this section is used by halving the link bandwidth for 25% of the links in the 8-pod FatTree topology.

Under the Web Search workload:

The random routing algorithm brings the best performance for DRLet in the symmetric topology, outperforming CONGA's routing algorithm and DRILL's routing algorithm by about 7% and 6.5%, respectively, regarding the overall average FCT. While for the highly asymmetric topology, CONGA's routing algorithm delivers almost the best performance for DRLet, outperforming DRILL's routing algorithm and the random routing algorithm by about 22% and 17%, respectively.

Under the Data Mining workload:

The routing algorithm of CONGA is the best for DRLet under the Data Mining workload, especially for elephant flows, which is better than DRILL's algorithm and the random routing algorithm by about 22% and 17% in the symmetric topology. While for the mice flow, the DRILL routing algorithm achieves the best performance for DRLet, outperforming CONGA algorithm and the random routing by more than 27% and more than 6% in symmetric and asymmetric topology, respectively.

Guidance for DRLet:

1) DRLet is suggested to employ congestion-aware dynamic routing for highly unbalanced scenarios (such as topology asymmetry). This will give DRLet comprehensive visibility of path congestion in the network to distribute flowlets, making

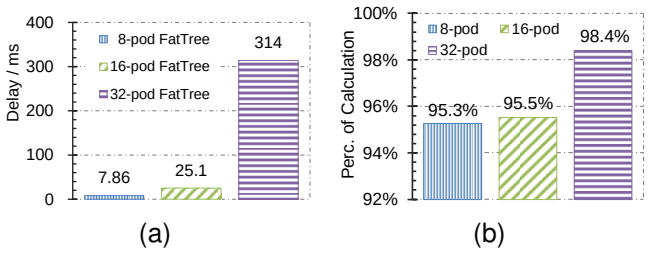

Fig. 18. Time overhead of the centralized control loop under different FatTree sizes. (a) The centralized control-loop delay under 80% load. (b) The percentage of calculation delay in control-loop delay under 80% load.

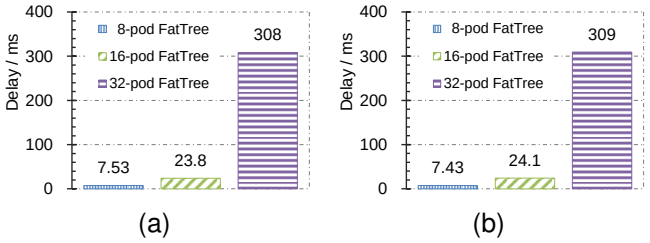

Fig. 19. The calculation delay of the DRL agent under varied load scenarios and different FatTree sizes. (a)Under the 20% load. (b)Under the mix of 40% and 60% load.

full advantage of the dynamic flowlet switching.

2) Under highly bursty network scenarios, DRLet prefers to use random routing. This ensures load balancing performance by providing many rerouting opportunities. However, dynamic routing may degrade performance instead. This is because multiple flowlets may be sent to the optimal path that was recorded earlier, but is now out-of-date and congested.

3) We observe that DRLet performs poorly in most network scenarios when it uses DRILL's routing algorithm. This is mainly because it does not have a comprehensive view of congestion, and it only routes based on the local congestion of the current hop. However, DRILL's routing algorithm performs well when using the packet-level granularity. This is because the load on each link is more balanced under the finest granularity, and the congestion difference on next hops does not affect the final performance much. This is why DRILL improves the FCT performance by 10% over random routing under the packet-level granularity.

D. Time Overhead of DRLet's Control Loop

Since DRLet adopts the DRL approach with centralized deployment, which is time-consuming, this section evaluates the time overhead of the control loop. The centralized control-loop delay is the time elapsed from when the network environment sends the network-state observation to when it receives the action from the DRLet agent.

We used two servers in the experiment: one for the emulated network environment and the other for the DRLet agent. The network environment server periodically sends network-state observations to the DRLet agent server. The DRLet agent server runs the DDPG algorithm shown in Algorithm 1 and sends the calculated actions to the network environment server. The two servers communicate through the websocket protocol and transmit data through 1 Gbps NICs. Moreover, the DRLet Agent server has the Nvidia K80 GPU.

The control-loop delay mainly consists of two parts: the communication delay of network-to-agent interactions, and the calculation delay of the DDPG algorithm in the DRLet agent. Fig. 18 shows the control-loop delays under different network sizes.

We can make three observations: i) the control loop delay is in the order of milliseconds; ii) the control-loop delay increases sharply as the network size increases; and iii) the calculation delay of the DRL agent makes up a very large fraction of the total control-loop delay, surpassing 95% even under small-scale networks.

Previous work [37] has shown that traffic demands are relatively stable over sub-second intervals due to load balancing. In addition, the average FCT is on the order of tens of milliseconds to hundreds of milliseconds according to Fig. 7. Therefore, throughout the lifespan of flows within the network, DRLet can deliver the promised performance without being affected by the millisecond-level DRL delay.

The possible issue is the large control-loop delay for the large-scale network. This is because the sizes of the action vector and the state vector will grow with the network size, leading to more stress on both the network-to-agent interaction and the calculation of the DRL agent. We are not concerned about the increased communication delay as it makes up a very small fraction of the total delay. The main concern is the calculation delay of the DRLet Agent.

In fact, the calculation delay consists of two specific parts: the computation time of neural networks and the memory access time of neural network parameters. However, existing GPUs often have very high computational ability but low memory access bandwidth, resulting in much higher memory access time than computation time. Moreover, with the high-dimensional state vector and action vector, the number of parameters of the neural network rises sharply, and the memory access time is even higher, which greatly delays the calculation of the DRL agent and consequently the control-loop delay.

To make DRLet feasible for large-scale networks, we need to reduce the enormous calculation delay of the DRL agent. A practical approach is to use distributed multi-agent reinforcement learning [38], [39], which distributes the computation and memory access to multiple agents and directly reduces the total calculation delay of the DRL agent.

Considering fluctuations in network load, we have also conducted experiments under the varied load scenarios to evaluate the calculation delay of the DRLet agent, as shown in Fig. 19. We found that DRLet ensures the timeliness of results regardless of load scenarios.

VII. Related Work

Flowlet was first proposed in FLARE [3], using packet bursts as the load balancing unit. Cisco researchers then applied flowlet-level load balancing to data center networks [4] and attracted a lot of studies.

Most existing work focuses on routing, i.e., selecting appropriate paths for flowlets to improve load balancing performance. CONGA [4], CLOVE [7], HULA [6], and Hermes [31] are the most typical work referring to congestion-aware routing for flowlets. CONGA uses piggybacking to collect the global congestion information, while CLOVE and HULA use different methods, which are probing and the traceroute mechanism, respectively. They both select the least congested path for a new flowlet. A more comprehensive set of path conditions (including congestion and link failures) are sensed in Hermes. Then, timely yet cautious rerouting decisions are made to bring performance gains for the active flow. Furthermore, CONGA and CLOVE require customized switches to collect information on path conditions. While CLOVE and Hermes are edge-based approaches that do not require switch modification.

There are other approaches to make routing decisions. CAF [40] proactively adjusts the congestion window based on measured available bandwidth before flowlets enter the network. While LetFlow [5] claims that flowlets should be allowed to explore different paths with random routing. Even if inappropriate routing decisions are made, flowlets can compensate for this due to elasticity. When a flowlet is routed to a slow path, it will lead to a large time gap for the ongoing flow, thus creating a new flowlet.

The above-mentioned work mostly uses the static flowlet timeout value to detect flowlets. DRW [41] tunes the flowlet timeout dynamically: setting a small flowlet timeout value for high-load scenarios and a large timeout value for low-load scenarios. However, the timeout value used in each load scenario is still a fixed static value. In addition, this is not feasible in realistic networks, because the load is time-varying, and the preset timeout value by DRW obviously cannot adapt to the time-varying network scenario.

VIII. Conclusion

In this paper, we propose DRLet, which applies deep reinforcement learning to dynamically configure the flowlet timeout values for achieving dynamic flowlet switching in the network. We first establish a network model to help deep reinforcement learning understand the flowlet switching problem, and then tailor and design the deep reinforcement learning algorithms according to the characteristics of the established model to better accomplish the mapping inside the model.

Simulation and emulation results have verified that: i) thanks to the effectiveness of DRL-based dynamic flowlet switching, the proposed scheme outperforms existing static flowlet timeout value-based schemes both under the baseline topology and asymmetric topology; ii) DRLet is concerned with the granularity of load balancing and is compatible with any routing algorithm, where random-based static routing helps DRLet explore more paths, while congestion-aware dynamic routing is preferred under high asymmetry or high unbalancing workloads; iii) the millisecond-level control loop delay is acceptable for DRLet, since its goal is to optimize the overall balancing granularity across the network.

In future work, we will try to design a distributed method using multi-agent reinforcement learning to achieve accurate flowlet detection in the flow-level perspective, considering that centralized control loop latency may still be a potential problem in large-scale networks.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, "Hedera: dynamic flow scheduling for data center networks." in *Nsdi*, vol. 10, no. 8, 2010, pp. 89–92.

[3] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, 2007.

[4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 503–514.

[5] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 407–420.

[6] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, 2016, pp. 1–12.

[7] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aare load balancing at the virtual edge," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, 2017, pp. 323–335.

[8] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 conference of the ACM special interest group on data communication*, 2018, pp. 191–205.

[9] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1871–1879.

[10] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proceedings of the 16th ACM workshop on hot topics in networks*, 2017, pp. 185–191.

[11] W. Wei, L. Fu, H. Gu, Y. Zhang, T. Zou, C. Wang, and N. Wang, "Grl-ps: Graph embedding-based drl approach for adaptive path selection," *IEEE Transactions on Network and Service Management*, 2023.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[13] J. Hu, C. Zeng, Z. Wang, J. Zhang, K. Guo, H. Xu, J. Huang, and K. Chen, "Load balancing with multi-level signals for lossless datacenter networks," *IEEE/ACM Transactions on Networking*, 2024.

[14] C. Hopps *et al.*, "Analysis of an equal-cost multi-path algorithm," RFC 2992, November, Tech. Rep., 2000.

[15] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 266–277, 2011.

[16] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465–478, 2015.

[17] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2130–2138.

[18] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, and T. He, "Caps: Coding-based adaptive packet spraying to reduce flow completion time in data center," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2338–2353, 2019.

[19] Broadcom, "Broadcom tomahawk 4 switch." [Online]. Available: https://docs.broadcom.com/doc/12398014

[20] Cisco, "Cisco nexus series switches." [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/data-center-analytics/nexus-dashboard/datasheet-c78-744371.html

[21] Huawei, "Huawei data center switches." [Online]. Available: https://e.huawei.com/eu/products/switches/data-center-switches

[22] Intel, "Intel tofino switches." [Online]. Available: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors.html

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] Q. Zhang, X. Wang, J. Lv, and M. Huang, "Intelligent content-aware traffic engineering for sdn: An ai-driven approach," *IEEE Network*, vol. 34, no. 3, pp. 186–193, 2020.

[25] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Load balancing for ultradense networks: A deep reinforcement learning-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9399–9412, 2019.

[26] "ns-3: Network simulator, version 3." [Online]. Available: https://www.nsnam.org/

[27] P. Gawłowicz and A. Zubow, "Ns-3 meets openai gym: The playground for machine learning in networking research," in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2019, pp. 113–120.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[29] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.

[30] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 51–62.

[31] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 253–266.

[32] J. Dong, L. Tan, C. Tian, Y. Zhou, Y. Wang, W. Dou, and G. Chen, "Meet: rack-level pooling based load balancing in datacenter networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3628–3639, 2022.

[33] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 225–238.

[34] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 conference*, 2011, pp. 350–361.

[35] N. Shelly, B. Tschaen, K.-T. Förster, M. Chang, T. Benson, and L. Vanbever, "Destroying networks for fun (and profit)," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, 2015, pp. 1–7.

[36] C. Gao, S. Chu, H. Xu, M. Xu, K. Ye, and C.-Z. Xu, "Flash: Joint flow scheduling and congestion control in data center networks," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 1038–1049, 2023.

[37] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[38] J. Hu, M. P. Wellman *et al.*, "Multiagent reinforcement learning: theoretical framework and an algorithm." in *ICML*, vol. 98. Citeseer, 1998, pp. 242–250.

[39] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3053–3062.

[40] S. Zou, J. Huang, W. Jiang, and J. Wang, "Achieving high utilization of flowlet-based load balancing in data center networks," *Future Generation Computer Systems*, 2020.

[41] F. Fan, H. Meng, B. Hu, K. L. Yeung, and Z. Zhao, "Roulette wheel balancing algorithm with dynamic flowlet switching for multipath datacenter networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 834–847, 2021.

Xinglong Diao received the B.E. degree in Communication Engineering from Xidian University, in 2017. He is currently working toward the Ph.D. degree in State Key Lab of ISN, Xidian University. His research interests include load balancing, congestion control, data center network, and machine learning for networking.

Huaxi Gu is a professor affiliated with the State Key Lab of ISN, Xidian University. Prof. Gu is the leader of the Youth Innovation Team of Shaanxi Universities. He is leading a project as the principal investigator, funded by the National Key Research and Development Program of China. He is also the principal investigator for one key, two general and one youth project funded by National Natural Science Foundation. Prof. Gu has published over 200 journal and conference papers, with his research interests being in the areas of networking technologies, network on chip, optical interconnect, etc. Prof. Gu served as the TPC member of GLOBECOM, ICC, PDCAT, etc., and the technical reviewer for multiple journals including IEEE Transactions on Computers, IEEE Transactions on VLSI, IEEE Transactions on Cloud Computing, IEEE/OSA Journal of Lightwave Technology, etc.

Wenting Wei received the M.E. and Ph.D. degrees in telecommunication and information systems from Xidian University in 2014 and 2019, respectively. Since 2019, she has been working at the State Key Lab of ISN, Xidian University. Her main research interests include data center networking, network virtualization, cloud computing and intelligent transportation.

Guoyong Jiang received the B.E. degree in Electronic Information Engineering from Qingdao University, in 2020. He is currently working toward the Master degree in State Key Lab of ISN, Xidian University. His research interests include load balancing, and data center network.

Baochun Li received the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2000. Since then, he has been with the Department of Electrical and Computer Engineering, University of Toronto, where he is currently a Professor. He holds the Bell Canada Endowed Chair in computer engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, machine learning, datacenter networking, and wireless networks. He is a member of the ACM and a fellow of the IEEE.