CFR-RL: Traffic Engineering With Reinforcement Learning in SDN

Junjie Zhang, *Member, IEEE*, Minghao Ye, Zehua Guo, *Senior Member, IEEE*, Chen-Yu Yen, and H. Jonathan Chao, *Fellow, IEEE*

***Abstract*— Traditional Traffic Engineering (TE) solutions can achieve the optimal or near-optimal performance by rerouting as many flows as possible. However, they do not usually consider the negative impact, such as packet out of order, when frequently rerouting flows in the network. To mitigate the impact of network disturbance, one promising TE solution is forwarding the majority of traffic flows using Equal-Cost Multi-Path (ECMP) and selectively rerouting a few *critical flows* using Software-Defined Networking (SDN) to balance link utilization of the network. However, critical flow rerouting is not trivial because the solution space for critical flow selection is enormous. Moreover, it is impossible to design a heuristic algorithm for this problem based on fixed and simple rules, since rule-based heuristics are unable to adapt to the changes of the traffic matrix and network dynamics. In this paper, we propose CFR-RL (Critical Flow Rerouting-Reinforcement Learning), a Reinforcement Learning-based scheme that learns a policy to select critical flows for each given traffic matrix automatically. CFR-RL then reroutes these selected critical flows to balance link utilization of the network by formulating and solving a simple Linear Programming (LP) problem. Extensive evaluations show that CFR-RL achieves near-optimal performance by rerouting only 10%-21.3% of total traffic.**

***Index Terms*— Reinforcement learning, software-defined networking, traffic engineering, load balancing, network disturbance mitigation.**

I. Introduction

THE emerging Software-Defined Networking (SDN) provides new opportunities to improve network performance [1]. In SDN, the control plane can generate routing policies based on its global view of the network and deploy these policies in the network by installing and updating flow entries at the SDN switches.

Traffic Engineering (TE) is one of important network features for SDN [2]–[4], and is usually implemented in the control plane of SDN. The goal of TE is to help Internet Service Providers (ISPs) optimize network performance and resource utilization by configuring the routing across their backbone networks to control traffic distribution [5], [6]. Due to dynamic load fluctuation among the nodes, traditional TE [7]–[12] reroutes many flows periodically to balance the load on each link to minimize network congestion probability, where a flow is defined as a source-destination pair. One usually formulates the flow routing problem with a particular performance metric as a specific objective function for optimization. For a given traffic matrix, one often wants to route all the flows in such a way that the maximum link utilization in the network is minimized.

Although traditional TE solutions can achieve the optimal or near-optimal performance by rerouting as many flows as possible, they do not consider the negative impact, such as packet out of order, when rerouting the flows in the network. To reach the optimal performance, TE solutions might reroute many traffic flows to just slightly reduce the link utilization on the most congested link, leading to significant network disturbance and service disruption. For example, a flow between two nodes in a backbone network is aggregated of many micro-flows (e.g., five tuples-based TCP flows) of different applications. Changing the path of a flow could temporarily affect many TCP flows' normal operation. Packets loss or out-of-order may cause duplicated ACK transmissions, triggering the sender to react and reduce its congestion window size and hence decrease its sending rate, eventually increasing the flow's completion time and degrading the flow's Quality of Service (QoS). In addition, rerouting all flows in the network could incur a high burden on the SDN controller to calculate and deploy new flow paths [4]. Because rerouting flows to reduce congestion in backbone networks could adversely affect the quality of users' experience, network operators have no desire to deploy these traditional TE solutions in their networks unless reducing network disturbance is taken into the consideration in designing the TE solutions.

To mitigate the impact of network disturbance, one promising TE solution is forwarding majority of traffic flows using Equal-Cost Multi-Path (ECMP) and selectively rerouting a few *critical flows* using SDN to balance link utilization of the network, where a critical flow is defined as a flow with a dominant impact on network performance (e.g., a flow on the most congested link) [4], [13]. Existing works show that critical flows exist in a given traffic matrix [4]. ECMP reduces the

Manuscript received October 1, 2019; revised February 15, 2020; accepted March 31, 2020. Date of publication June 5, 2020; date of current version September 16, 2020. The work of Zehua Guo was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003700 and in part by the Beijing Institute of Technology Research Fund Program for Young Scholars. *(Corresponding author: Zehua Guo.)*

Junjie Zhang is with Fortinet Inc., Sunnyvale, CA 94086 USA (e-mail: junjie.zhang@nyu.edu).

Minghao Ye, Chen-Yu Yen, and H. Jonathan Chao are with the Department of Electrical and Computer Engineering, New York University, New York City, NY 11201 USA (e-mail: my1706@nyu.edu; cyy310@nyu.edu; chao@nyu.edu).

Zehua Guo is with the Beijing Institute of Technology, Beijing 100081, China (e-mail: guo@bit.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2020.3000371

congestion probability by equally splitting traffic on equal-cost paths while critical flow rerouting aims to achieve further performance improvement with low network disturbance.

The critical flow rerouting problem can be decoupled into two sub-problems: (1) identifying critical flows and (2) rerouting them to achieve good performance. Although sub-problem (2) is relatively easy to solve by formulating it as a Linear Programming (LP) optimization problem, solving sub-problem (1) is not trivial because the solution space is huge. For example, if we want to find 10 critical flows among 100 flows, the solution space has $C_{100}^{10} \approx 17$ trillion combinations. Considering the fact that traffic matrix varies in the level of minutes, an efficient solution should be able to quickly and effectively identify the critical flows for each traffic matrix. Unfortunately, it is impossible to design a heuristic algorithm for the above algorithmically-hard problem based on fixed and simple rules. This is because rule-based heuristics are unable to adapt to the changes of the traffic matrix and network dynamics and thus unable to guarantee their performance when their design assumptions are violated, as later shown in Section VI-B.

In this paper, we propose CFR-RL (Critical Flow Rerouting-Reinforcement Learning), a Reinforcement Learning-based scheme that performs critical flow selection followed by rerouting with linear programming. CFR-RL learns a policy to select critical flows purely through observations, without any domain-specific rule-based heuristic. It starts from scratch without any prior knowledge, and gradually learns to make better selections through reinforcement, in the form of reward signals that reflects network performance for past selections. By continuing to observe the actual performance of past selections, CFR-RL would optimize its selection policy for various traffic matrices as time goes. Once training is done, CFR-RL will efficiently and effectively select a small set of critical flows for each given traffic matrix, and reroute them to balance link utilization of the network by formulating and solving a simple linear programming optimization problem.

The main contributions of this paper are summarized as follows:

1) We consider the impact of flow rerouting on network disturbance in our TE design and propose an effective scheme that not only minimizes the maximum link utilization but also reroutes only a small number of flows to reduce network disturbance.
2) We customize a RL approach to learn the critical flow selection policy, and utilize LP as a reward function to generate reward signals. This RL+LP combined approach turns out to be surprisingly powerful.
3) We evaluate and compare CFR-RL with other rule-based heuristic schemes by conducting extensive experiments on different topologies with both real and synthesized traffic. CFR-RL not only outperforms rule-based heuristic schemes by up to 12.2%, but also reroutes 11.4%-14.7% less traffic on average. Overall, CFR-RL is able to achieve near-optimal performance by rerouting only 10%-21.3% of total traffic. In addition, the evaluation results show that CFR-RL is able to generalize to unseen traffic matrices.

The remainder of this paper is organized as follows. Section II describes the related works. Section III presents the system design. Section IV discusses how to train the critical flow selection policy using a RL-based approach. Section V describes how to reroute the critical flows. Section VI evaluates the effectiveness of our scheme. Section VII concludes the paper and discusses future work.

II. Related Works

A. Traditional TE Solutions

In Multiprotocol Label Switching (MPLS) networks, a routing problem has been formulated as an optimization problem where explicit routes are obtained for each source-destination pair to distribute traffic flows [7], [8]. Using Open Shortest Path First (OSPF) and ECMP protocols, [9]–[11] attempt to balance link utilization as even as possible by carefully tuning the link costs to adjust path selection in ECMP. OSPF-OMP (OMP, Optimized Multipath) [14], a variation of OSPF, attempts to dynamically determine the optimal allocation of traffic among multiple equal-cost paths based on the exchange of special traffic-load control messages. Weighted ECMP [12] extends ECMP to allow weighted traffic splitting at each node and achieves significant performance improvement over ECMP. Two-phase routing optimizes routing performance by selecting a set of intermediate nodes and tuning the traffic split ratios to the nodes [15], [16]. In the first phase, each source sends traffic to the intermediate nodes based on predetermined split ratios, and in the second phase, the intermediate nodes then deliver the traffic to the final destinations. This approach requires IP tunnels, optical-layer circuits, or label switched paths in each phase.

B. SDN-Based TE Solutions

Thanks to the flexible routing policy from the emerging SDN, dynamic hybrid routing [4] achieves load balancing for a wide range of traffic scenarios by dynamically rebalancing traffic to react to traffic fluctuations with a preconfigured routing policy. Agarwal *et al.* [2] consider a network with partially deployed SDN switches. They improve network utilization and reduce packet loss by strategically placing the controller and SDN switches. Guo *et al.* [3] propose a novel algorithm named SOTE to minimize the maximum link utilization in an SDN/OSPF hybrid network.

C. Machine Learning-Based TE Solutions

Machine learning has been used to improve the performance of backbone networks and data center networks. For backbone networks, Geyer and Carle [17] design an automatic network protocol using semi-supervised deep learning. Sun *et al.* [18] selectively control a set of nodes and use a RL-based policy to dynamically change the routing decision of flows traversing the selected nodes. To minimize signaling delay in large SDNs, Lin *et al.* [19] employ a distributed three-level control plane architecture coupled with a RL-based solution named QoS-aware Adaptive Routing. Xu *et al.* [20] use RL to optimize the throughput and delay in TE. AuTO [21] is

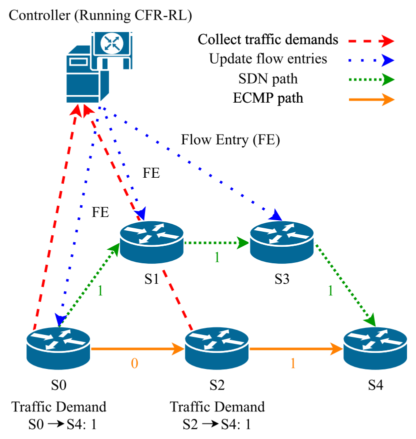

Fig. 1. An illustrative example of CFR-RL rerouting procedure. Each link capability equal to 1. Best viewed in color.

developed to optimize routing traffic in data center networks with a two-layer RL. One is called the Peripheral System for deploying hosts and routing small flows, and the other one is called the Central System for collecting global traffic information and routing large flows.

However, all of the above works do not consider mitigating the impact of network disturbance and service disruption caused by rerouting.

III. System Design

In this section, we describe the design of CFR-RL, a RL-based scheme that learns a critical flow selection policy and reroutes the corresponding critical flows to balance link utilization of the network.

We train CFR-RL to learn a selection policy over a rich variety of historical traffic matrices, where traffic matrices can be measured by SDN switches and collected by an SDN central controller periodically [22]. CFR-RL represents the selection policy as a neural network that maps a "raw" observation (e.g., a traffic matrix) to a combination of critical flows. The neural network provides a scalable and expressive way to incorporate various traffic matrices into the selection policy. CFR-RL trains this neural network based on REINFORCE algorithm [23] with some customizations, as detailed in Section IV.

Once training is done, CFR-RL applies the critical flow selection policy to each real time traffic matrix provided by the SDN controller periodically, where a small number of critical flows (e.g., K) are selected. The evaluation results in Section VI-B.1 show that selecting 10% of total flows as critical flows (roughly 11%-21% of total traffic) is sufficient for CFR-RL to achieve near-optimal performance, while network disturbance (i.e., the percentage of total rerouted traffic) is reduced by at least 78.7% compared to rerouting all flows by traditional TE. Then the SDN controller reroutes the selected critical flows by installing and updating corresponding flow entries at the switches using a flow rerouting optimization method described in Section V. The remaining flows would continue to be routed by the default ECMP routing. Note that the flow entries at the switches for the critical flows selected in the previous period will time out, and the flows would be routed by either default ECMP routing or newly installed flow entries in the current period. Figure 1 shows an illustrative example. CFR-RL reroutes the flow from S0 to S4 to balance link load by installing forwarding entries at the corresponding switches along the SDN path.

There are two reasons we do not want to adopt RL for the flow rerouting problem. Firstly, since the set of critical flows is small, LP is an efficient and optimal method to solve the rerouting problem. Secondly, a routing solution consists of a split ratio (i.e., traffic demand percentage) for each flow on each link. Given a network with E links, there will be total $E * K$ split ratios in the routing solution, where K is the number of critical flows. Since split ratios are continuous numbers, we have to adopt the RL methods for continuous action domain [24], [25]. However, due to the high-dimensional, continuous action spaces, it has been shown that this type of RL methods would lead to slow and ineffective learning when the number of output parameters (i.e., $E * K$) is large [20], [26].

IV. Learning a Critical Flow Selection Policy

In this section, we describe how to learn a critical flow selection policy using a customized RL approach.

A. Reinforcement Learning Formulation

1) Input / State Space: An agent takes a state $s_t = TM_t$ as an input, where TM_t is a traffic matrix at time step t that contains information of traffic demand of each flow. Typically, the network topology remains unchanged. Thus, we do not include the topology information as a part of the input. The results in Section VI-B show that CFR-RL is able to learn a good policy π without prior knowledge of the network. It is worth noting that including additional information like link states as a part of input might be beneficial for training the critical flow selection policy. We will investigate it in our future work.

2) Action Space: For each state s_t, CFR-RL would select K critical flows. Given that there are total $N * (N-1)$ flows in a network with N nodes, this RL problem would require a large action space of size $C^K_{N*(N-1)}$. Inspired by [27], we define the action space as $\{0, 1, \ldots, (N*(N-1))-1\}$ and allow the agent to sample K different actions in each time step t (i.e., $a^1_t, a^2_t, \ldots, a^K_t$).

3) Reward: After sampling K different critical flows (i.e., f_K) for a given state s_t, CFR-RL reroutes these critical flows and obtains the maximum link utilization U by solving the rerouting optimization problem (4a) (described in the following section). Reward r is defined as $1/U$, which is set to reflect the network performance after rerouting critical flows to balance link utilization. The smaller U (i.e., the greater reward r), the better performance. In other words, CFR-RL adopts LP as a reward function to produce reward signals r for RL.

B. Training Algorithm

The critical flow selection policy is represented by a neural network. This policy network takes a state $s_t = TM_t$ as

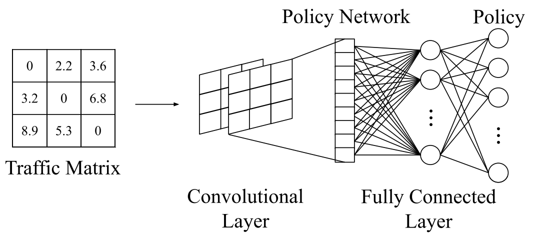

Fig. 2. Policy network architecture.

an input as described above and outputs a probability distribution $\pi(a_t|s_t)$ over all available actions. Figure 2 shows the architecture of the policy network (details in Section VI-A.1). Since K different actions are sampled for each state s_t and their order does not matter, we define a solution $a_{t_K} = (a_t^1, a_t^2, \ldots, a_t^K)$ as a combination of K sampled actions. For selecting a solution a_{t_K} with a given state s_t, a stochastic policy $\pi(a_{t_K}|s_t)$ parameterized by θ can be approximated as follows[1]:

$$\pi(a_{t_K}|s_t;\theta) \approx \prod_{i=1}^{K} \pi(a_t^i|s_t;\theta). \tag{1}$$

The goal of training is to maximize the network performance over various traffic matrices, i.e., maximize the expected reward $E[r_t]$. Thus, we optimize $E[r_t]$ by gradient ascend, using REINFORCE algorithm with a baseline $b(s_t)$. The policy parameter θ is updated according to the following equation:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log\pi(a_{t_K}|s_t;\theta)(r_t - b(s_t)), \tag{2}$$

where α is the learning rate for the policy network. A good baseline $b(s_t)$ reduces gradient variance and thus increases speed of learning. In this paper, we use an average reward for each state s_t as the baseline. $(r_t - b(s_t))$ indicates how much better a specific solution is compared to the "average solution" for a given state s_t according to the policy. Intuitively, Eq.(2) can be explained as follows. If $(r_t - b(s_t))$ is positive, $\pi(a_{t_K}|s_t;\theta)$ (i.e., the probability of the solution a_{t_K}) is increased by updating the policy parameters θ in the direction $\nabla_\theta log\pi(a_{t_K}|s_t;\theta)$ with a step size of $\alpha(r_t - b(s_t))$. Otherwise, the solution probability is decreased. The net effect of Eq. (2) is to reinforce actions that empirically lead to better rewards.

To ensure that the RL agent explores the action space adequately during training to discover good policies, the entropy of the policy π is added to Eq. (2). This technique improves the exploration by discouraging premature convergence to suboptimal deterministic policies [28]. Then, Eq(2) is modified to the following equation:

$$\theta \leftarrow \theta + \alpha \sum_t (\nabla_\theta log\pi(a_{t_K}|s_t;\theta)(r_t - b(s_t)) + \beta\nabla_\theta H(\pi(\cdot|s_t;\theta))), \tag{3}$$

[1]To select K distinct actions, we do the action sampling without replacement. The right side of Eq. (1) is the solution probability when sampling with replacement, we use Eq. (1) to approximate the probability of the solution a_{t_K} given a state s_t for simplicity.

Algorithm 1 Training Algorithm

```
Initialize $\theta$, $v = \{\}$ (keep track the sum of rewards for each state), $n = \{\}$ (keep track the visited count of each state)
for each iteration do
  $\Delta\theta \leftarrow 0$
  $\{s_t\} \leftarrow$ Sample a batch of states with size $B$
  for $t = 1, \ldots, B$ do
    Sample a solution $a_{t_K}$ according to policy $\pi(a_{t_K}|s_t)$
    Receive reward $r_t$
    if $s_t \in v$ and $s_t \in n$ then
      $b(s_t) = \frac{v[s_t]}{n[s_t]}$ (average reward for state $s_t$)
    else
      $b(s_t) = 0$, $v[s_t] = 0$, $n[s_t] = 0$
    end if
  end for
  for $t = 1, \ldots, B$ do
    $\Delta\theta \leftarrow \Delta\theta + \alpha(\nabla_\theta log\pi(a_{t_K}|s_t;\theta)(r_t - b(s_t)) + \beta\nabla_\theta H(\pi(\cdot|s_t;\theta)))$
    $v[s_t] = v[s_t] + r_t$
    $n[s_t] = n[s_t] + 1$
  end for
  $\theta \leftarrow \theta + \Delta\theta$
end for
```

where H is the entropy of the policy (the probability distribution over actions). The hyperparameter β controls the strength of the entropy regularization term. Algorithm 1 shows the pseudo-code for the training algorithm.

V. Rerouting Critical Flows

In this section, we describe how to reroute the selected critical flows to balance link utilization of the network.

A. Notations

$G(V,E)$	network with nodes V and directed edges E ($\|V\| = N, \|E\| = M$).
$c_{i,j}$	the capacity of link $\langle i,j \rangle$ ($\langle i,j \rangle \in E$).
$l_{i,j}$	the traffic load on link $\langle i,j \rangle$ ($\langle i,j \rangle \in E$).
$D^{s,d}$	the traffic demand from source s to destination d ($s, d \in V$, $s \neq d$).
$\sigma_{i,j}^{s,d}$	the percentage of traffic demand from source s to destination d routed on link $\langle i,j \rangle$ ($s, d \in V$, $s \neq d$, $\langle i,j \rangle \in E$, $\langle s,d \rangle \in f_K$).

B. Explicit Routing for Critical Flows

By default, traffic is distributed according to ECMP routing. We reroute the small set of critical flows (i.e., f_K) by conducting explicit routing optimization for these critical flows $\langle s,d \rangle \in f_K$.

The critical flow rerouting problem can be described as the following. Given a network $G(V,E)$ with the set of traffic demands $D^{s,d}$ for the selected critical flows ($\forall \langle s,d \rangle \in f_K$) and the background link load $\{\bar{l}_{i,j}\}$ contributed by the remaining flows using the default ECMP routing, our objective is to obtain the optimal explicit routing ratios $\{\sigma_{i,j}^{s,d}\}$ for each critical flow, so that the maximum link utilization U is minimized.

To search all possible under-utilized paths for the selected critical flows, we formulate the rerouting problem as an optimization as follows.

$$minimize\ U + \epsilon \cdot \sum_{\langle i,j\rangle \in E} \sum_{\langle s,d\rangle \in f_K} \sigma_{i,j}^{s,d} \tag{4a}$$

$$\text{subject to } l_{i,j} = \sum_{\langle s,d\rangle \in f_K} \sigma_{i,j}^{s,d} \cdot D^{s,d} + \bar{l}_{i,j}\ i,j : \langle i,j\rangle \in E \tag{4b}$$

$$l_{i,j} \le c_{i,j} \cdot U i,j : \langle i,j\rangle \in E \tag{4c}$$

$$\sum_{k:\langle k,i\rangle \in E} \sigma_{k,i}^{s,d} - \sum_{k:\langle i,k\rangle \in E} \sigma_{i,k}^{s,d} = \begin{cases} -1 & \text{if } i = s \\ 1 & \text{if } i = d \\ 0 & \text{otherwise} \end{cases}$$

$$i \in V, \quad s,d : \langle s,d\rangle \in f_K \tag{4d}$$

$$0 \le \sigma_{i,j}^{s,d} \le 1 s,d : \langle s,d\rangle \in f_K, \quad i,j : \langle i,j\rangle \in E \tag{4e}$$

$\epsilon \cdot \sum_{\langle i,j\rangle \in E} \sum_{\langle s,d\rangle \in f_K} \sigma_{i,j}^{s,d}$ in (4a) is needed because otherwise the optimal solution may include unnecessarily long paths as long as they avoid the most congested link, where ϵ ($\epsilon > 0$) is a sufficiently small constant to ensure that the minimization of U takes higher priority [29]. (4b) indicates the traffic load on link $\langle i,j\rangle$ contributed by the traffic demands routed by the explicit routing and the traffic demands routed by the default ECMP routing. (4c) is the link capacity utilization constraint. (4d) is the flow conservation constraint for the selected critical flows.

By solving the above LP problem using LP solvers (such as Gurobi [30]), we can obtain the optimal explicit routing solution for selected critical flows $\{\sigma_{i,j}^{s,d}\}$ $(\forall\langle s,d\rangle \in f_K)$. Then, the SDN controller installs and updates flow entries at the switches accordingly.

VI. Evaluation

In this section, a series of simulation experiments are conducted using real-world network topologies to evaluate the performance of CFR-RL and show its effectiveness by comparing it with other rule-based heuristic schemes.

A. Evaluation Setup

1) Implementation: The policy neural network consists of three layers. The first layer is a convolutional layer with 128 filters. The corresponding kernel size is 3×3 and the stride is set to 1. The second layer is a fully connected layer with 128 neurons. The activation function used for the first two layers is Leaky ReLU [31]. The final layer is a fully connected linear layer (without activation function) with $N * (N - 1)$ neurons corresponding to all possible critical flows. The softmax function is applied upon the output of final layer to generate the probabilities for all available actions. The learning rate α is initially configured to 0.001 and decays every 500 iterations with a base of 0.96 until it reaches the minimum value 0.0001. Additionally, the entropy factor β is configured to 0.1. We found that the set of above hyperparameters is a good trade-off between performance and computational complexity of the model (details in Section VI-B.5). Thus, we fixed them throughout our experiments. The results in the following experiments show CFR-RL works well on different network topologies with a single set of fixed hyperparameters. This architecture is implemented using TensorFlow [32].

TABLE I
ISP Networks Used in Evaluation

Topology	Nodes	Directed Links	Pairs
Abilene	12	30	132
EBONE (Europe)	23	74	506
Sprintlink (US)	44	166	1892
Tiscali (Europe)	49	172	2352

2) Dataset: In our evaluation, we use four real-world network topologies including Abilene network and 3 ISP networks collected by ROCKETFUEL [33]. The number of nodes and directed links of the networks are listed in Table I. For the Abilene network, the measured traffic matrices and network topology information (such as link connectivity, weights, and capacities) are available in [34]. Since Abilene traffic matrices are measured every 5 minutes, there are a total of 288 traffic matrices each day. To evaluate the performance of CFR-RL, we choose a total 2016 traffic matrices in the first week (starting from Mar. 1st 2004) as our dataset. For ROCKETFUEL topologies, the link costs are given while the link capacities are not provided. Therefore, we infer the link capacities as the inverse of link costs, which are based on the default link cost setting in Cisco routers. In other words, the link costs are inversely proportional to the link capacities. This approach is commonly adopted in literature [4], [13], [15]. Besides, since traffic matrices are also unavailable for the ISP networks from ROCKETFUEL, we use a traffic matrix generation tool [35] to generate 50 synthetic exponential traffic matrices and 50 synthetic uniform traffic matrices for each network. Unless otherwise noted, we use a random sample of 70% of our dataset as a training set for CFR-RL, and use the remaining 30% as a test set for testing all schemes.

3) Parallel Training: To speed up training, we spawn multiple actor agents in parallel, as suggested by [28]. CFR-RL uses 20 actor agents by default. Each actor agent is configured to experience a different subset of the training set. Then, these agents continually forward their (state, action, advantage (i.e, $r_t - b(s_t)$)) tuples to a central learner agent, which aggregates them to train the policy neural network. The central learner agent performs a gradient update using Eq(3) according to the received tuples, then sends back the updated parameters of the policy network to the actor agents. The whole process can happen asynchronously among all agents. We use 21 CPU cores to train CFR-RL (i.e., one core (2.6GHz) for each agent).

4) Metrics:

a) Load balancing performance ratio: To demonstrate the load balancing performance of the proposed CFR-RL scheme, a load balancing performance ratio is applied and defined as follows:

$$PR_U = \frac{U^{\text{optimal}}}{U^{\text{CFR-RL}}}, \tag{5}$$

where U^{optimal} is the maximum link utilization achieved by an optimal explicit routing for all flows[2]. $PR_U = 1$ means that the proposed CFR-RL achieves load balancing as good as the optimal routing. A lower ratio indicates that the load balancing performance of CFR-RL is farther away from that of the optimal routing.

b) End-to-end delay performance ratio: To model and measure end-to-end delay in the network, we define the overall end-to-end delay in the network as $\Omega = \sum_{\langle i,j\rangle \in E} (\frac{l_{i,j}}{c_{i,j} - l_{i,j}})$ as described in [12]. Then, an end-to-end delay performance ratio is defined as follows:

$$PR_{\Omega} = \frac{\Omega^{\text{optimal}}}{\Omega^{\text{CFR-RL}}}, \tag{6}$$

where Ω^{optimal} is the minimum end-to-end delay achieved by an optimal explicit routing for all flows with an objective[3] to minimize the end-to-end delay Ω. Note that the rerouting solution for selected critical flows is still obtained by solving (4a). The higher PR_{Ω}, the better end-to-end delay performance achieved by CFR-RL. $PR_{\Omega} = 1$ means that the proposed CFR-RL achieves the minimum end-to-end delay as the optimal routing.

c) Rerouting disturbance: To measure the disturbance caused by rerouting, we define rerouting disturbance as the percentage of total rerouted traffic[4] for a given traffic matrix, i.e.,

$$RD = \frac{\sum_{\langle s,d\rangle \in f_K} D^{s,d}}{\sum_{s,d \in V, s \neq d} D^{s,d}}, \tag{7}$$

where $\sum_{\langle s,d\rangle \in f_K} D^{s,d}$ is the total traffic of selected critical flows that need to be rerouted and $\sum_{s,d \in V, s \neq d} D^{s,d}$ is the total traffic of all flows. The smaller RD, the less disturbance caused by rerouting.

5) Rule-Based Heuristics: For comparison, we also evaluate two rule-based heuristics as the following:

1) **Top-K**: selects the K largest flows from a given traffic matrix in terms of demand volume. This approach is based on the assumption that flows with larger traffic volumes would have a dominant impact on network performance.
2) **Top-K Critical**: similar to Top-K approach, but selects the K largest flows from the most congested links. This approach is based on the assumption that flows traversing the most congested links would have a dominant impact on network performance.

[2]The corresponding LP formulation is similar to (4a), except that the objective becomes obtaining the optimal explicit ratios $\{\sigma_{i,j}^{s,d}\}$ for all flows. Note that the background link load $\{\bar{l}_{i,j}\}$ would be 0 for this problem.

[3]The objective of this LP problem is to obtain the optimal explicit routing ratios $\{\sigma_{i,j}^{s,d}\}$ for all flows, such that Ω is minimized.

[4]Although partial of traffic flows might still be routed along the original ECMP paths, updating routing at the switches might cause packets drop or out-of-order. Thus, we still consider this amount of traffic as rerouting traffic.

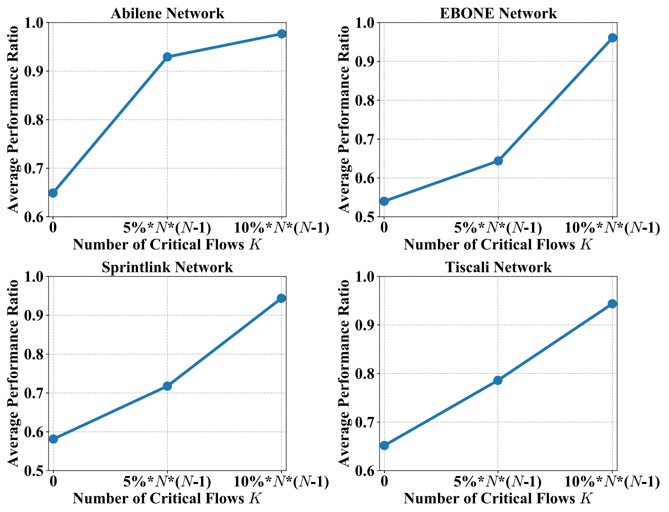

Fig. 3. Average load balancing performance ratio of CFR-RL with increasing number of critical flows K on the four networks.

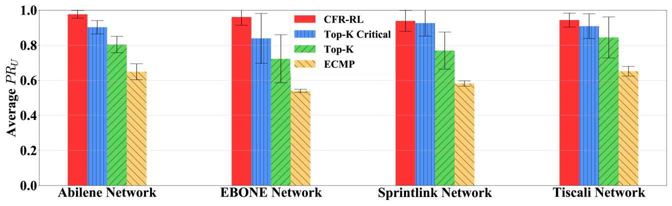

Fig. 4. Comparison of average load balancing performance ratio where error bars span ± one standard deviation from the average on the entire test set of the four networks.

B. Evaluation

1) Critical Flows Number: We conduct a series of experiments with different number of critical flows selected, and fix other parameters throughout the experiments.

Figure 3 shows the average load balancing performance ratio achieved by CFR-RL with increasing number of critical flows K. The initial value with $K = 0$ represents the default ECMP routing. The results indicate that there is a considerable room for further improvement when flows are routed by ECMP. The sharp increases in the average load balancing performance ratio for all four networks shown in Fig. 3 indicate that CFR-RL is able to achieve near-optimal load balancing performance by rerouting only 10% flows. As a result, network disturbance would be much reduced compared to rerouting all flows as traditional TE. For the subsequent experiments, we set $K = 10\% * N * (N-1)$ for each network.

2) Performance Comparison: For comparison, we also calculate the performance ratios and rerouting disturbances for Top-K, Top-K critical, and ECMP according to Eqs. (5), (6) and (7). Figure 4 shows the average load balancing performance ratio that each scheme achieves on the entire test set of the four networks. Figure 5 shows the load balancing performance ratio on each individual traffic matrix for the four networks. Note that the first 15 traffic matrices in Figs. 5(b)-5(d) are generated by an exponential model and the remaining 15 traffic matrices are generated by an uniform model. CFR-RL performs significantly well in all networks.

TABLE II
COMPARISON OF AVERAGE REROUTING DISTURBANCE

Topology	CFR-RL	Top-K Critical	Top-K
Abilene	21.3%	32.7%	42.9%
EBONE (Exponential / Uniform)	11.2% / 10.0%	25.9% / 11.5%	32.9% / 11.7%
Sprintlink (Exponential / Uniform)	11.3% / 10.1%	23.6% / 13.8%	33.2% / 14.6%
Tiscali (Exponential / Uniform)	11.2% / 10.0%	24.5% / 12.0%	32.7% / 12.2%

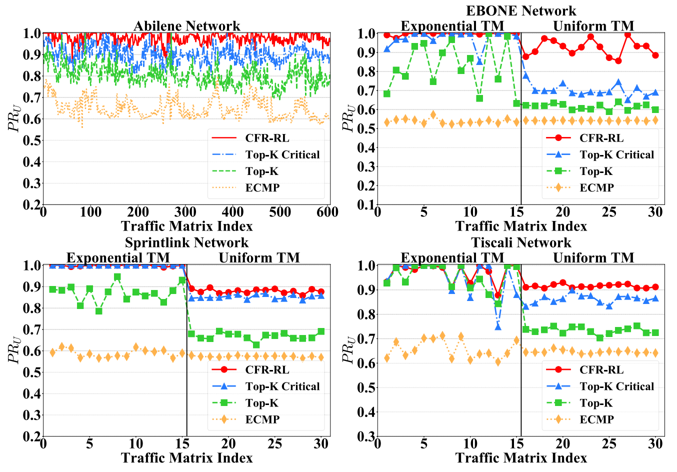

Fig. 5. Comparison of load balancing performance in the four networks on each test traffic matrix.

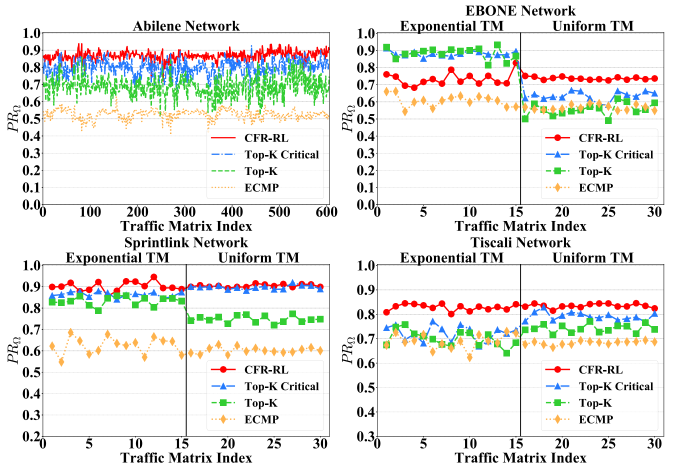

Fig. 7. Comparison of end-to-end delay performance in the four networks on each test traffic matrix.

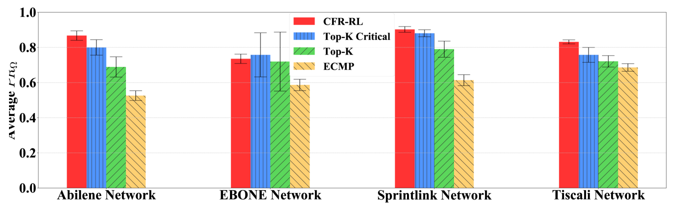

Fig. 6. Comparison of average end-to-end delay performance ratio where error bars span ± one standard deviation from the average on the entire test set of the four networks.

For example, for the Abilene network, CFR-RL improves load balancing performance by about 32.8% compared to ECMP, and by roughly 7.4% compared to Top-K critical. For the EBONE network, CFR-RL outperforms Top-K critical with an average 12.2% load balancing performance improvement. For Sprintlink and Tiscali networks, CFR-RL performs slightly better than Top-K critical by 1.3% and 3.5% on average, respectively. Moreover, Figure 6 shows the average end-to-end delay performance ratio that each scheme achieves on the entire test set of the four networks. Figure 7 shows the end-to-end delay performance ratio on each test traffic matrix for the four networks. It is worth noting that the rerouting solution for selected critical flows is still obtained by solving (4a) (i.e., minimize maximum link utilization), though the end-to-end delay performance is evaluated[5] for each scheme.

[5]For the Abilene network, the real traffic demands in the measured traffic matrices collected in [34] are relatively small, and thus the corresponding end-to-end delay would be very small. To effectively compare end-to-end delay performance of each scheme, we multiply each demand $D^{s,d}$ in a real traffic matrix TM_t by $\frac{0.9}{U_t^{\text{ECMP}}}$, where U_t^{ECMP} is the maximum link utilization achieved by ECMP routing on the traffic matrix TM_t.

By effectively selecting and rerouting critical flows to balance link utilization of the network, CFR-RL outperforms heuristic schemes and ECMP in terms of end-to-end delay in all networks except the EBONE network. In the EBONE network, heuristic schemes perform better with the exponential traffic model. It is possible that rerouting the elephant flows selected by heuristic schemes further balances load on non-congested links and results in achieving smaller end-to-end delay. In addition, Tab. II shows the average rerouting disturbance, i.e., the average percentage of total traffic rerouted by each scheme (except ECMP) for the four networks. CFR-RL greatly reduces network disturbance by rerouting at most 21.3%, 11.2%, 11.3%, and 11.2% of total traffic on average for the four networks, respectively. In contrast, Top-K critical reroutes 11.4% more traffic for the Abilene network and 14.7%, 12.3%, and 13.3% more traffic for the EBONE, Sprintlink, and Tiscali networks (for exponential traffic matrices). Top-K performs even worse by rerouting more than 42% of total traffic on average for the Abilene network and 32%, 33%, and 32% of total traffic on average for the other three networks (for exponential traffic matrices). It is worth noting that there are no elephant flows in uniform traffic matrices shown in Fig. 5(b)-5(d). Thus, all three schemes reroute similar amount of traffic for uniform traffic matrices. However, CFR-RL is still able to perform slightly better than the two rule-based heuristics. Overall, the above results indicate that CFR-RL is able to achieve near-optimal load balancing performance and greatly reduce end-to-end delay and network disturbance by smartly selecting a small number of critical flows for each given traffic matrix and effectively rerouting the corresponding small amount of traffic.

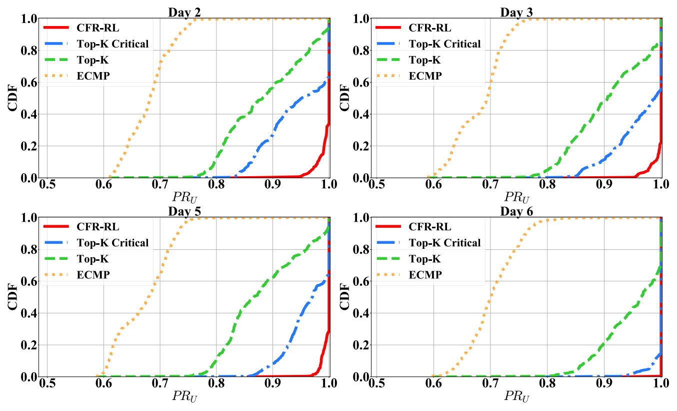

Fig. 8. Comparison of load balancing performance ratio in CDF with the traffic matrices from Tuesday, Wednesday, Friday and Saturday in week 2.

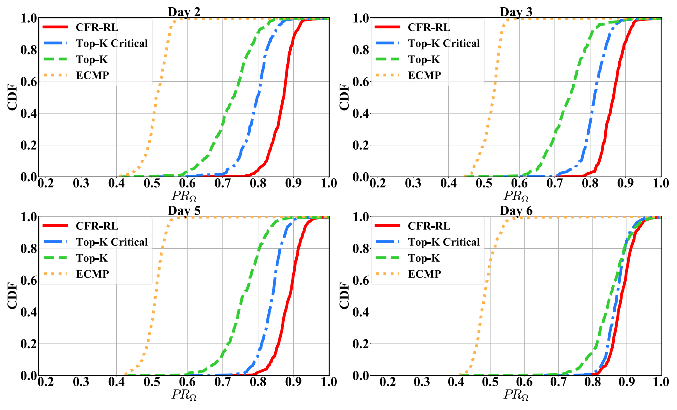

Fig. 9. Comparison of end-to-end delay performance ratio in CDF with the traffic matrices from Tuesday, Wednesday, Friday and Saturday in week 2.

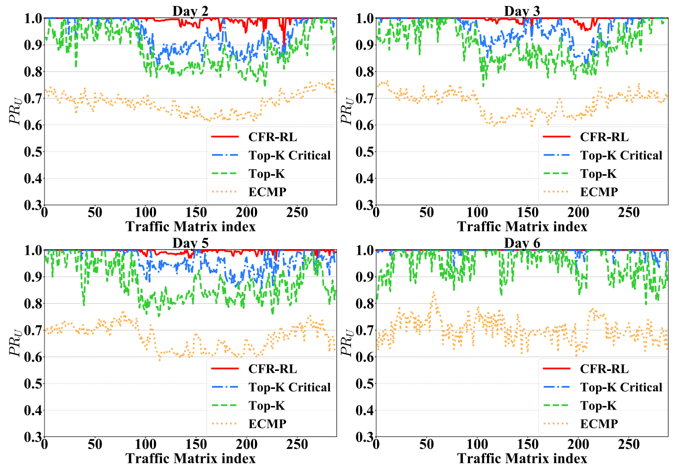

Fig. 10. Comparison of load balancing performance ratio with the traffic matrices from Tuesday, Wednesday, Friday and Saturday in week 2.

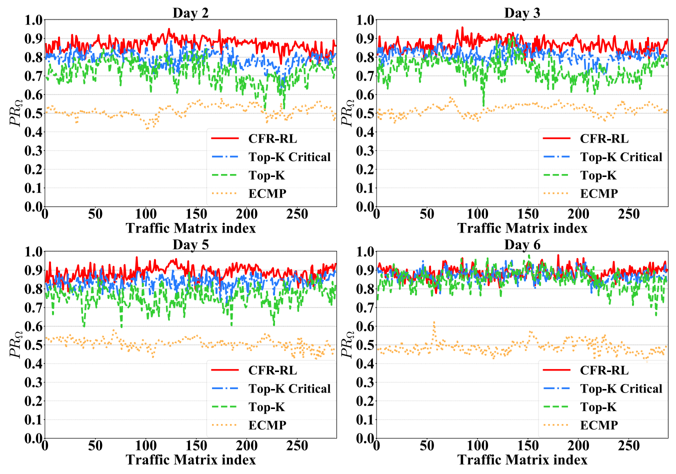

Fig. 11. Comparison of end-to-end delay performance ratio with the traffic matrices from Tuesday, Wednesday, Friday and Saturday in week 2.

As shown in Figs. 5(b)-5(d), Top-K critical performs well with the exponential traffic model. However, its performance is degraded with the uniform traffic model. One possible reason for the performance degradation of Top-K critical is that all links in the network are relatively saturated under the uniform traffic model. Alternative underutilized paths are not available for the critical flows selected by Top-K critical. In other words, there is no much room for rerouting performance improvement by only considering the elephant flows traversing the most congested links. Thus, fixed-rule heuristics are unable to guarantee their performance, showing that their design assumptions are invalid. In contrast, CFR-RL performs consistently well under various traffic models.

3) Generalization: In this series of experiments, we trained CFR-RL on the traffic matrices from the first week (starting from Mar. 1st 2004) and evaluate it for each day of the following week (starting from Mar. 8th 2004) for the Abilene network. We only present the results for day 2, day 3, day 5 and day 6, since the results for other days are similar. Figures 8 and 9 show the full CDFs of two types of performance ratio for these 4 days. Figures 10 and 11 show the load balancing and end-to-end delay performance ratios on each traffic matrix of these 4 days, respectively. The results show that CFR-RL still achieves above 95% optimal load balancing performance and average 88.13% end-to-end delay performance, and thus outperforms other schemes on almost all traffic matrices. The load balancing performance of CFR-RL degrades on several outlier traffic matrices in day 2. There are two possible reasons for the degradation: (1) The traffic patterns of these traffic matrices are different from what CFR-RL learned from the previous week. (2) Selecting $K = 10\% * N * (N-1)$ is not enough for CFR-RL to achieve near-optimal performance on these outlier traffic matrices. However, CFR-RL still performs better than other schemes. Overall, the results indicate that real traffic patterns are relatively stable and CFR-RL generalizes well to unseen traffic matrices which are not explicitly trained.

4) Training and Inference Time: Training a policy for the Abilene network took approximately 10,000 iterations, and the time consumed for each iteration is approximately 1 second. As a result, the total training time for Abilene network is approximately 3 hours. Since the EBONE network is relatively larger, it took approximately 60,000 iterations to train a policy. Then, the total training time for EBONE network is approximately 16 hours. For larger networks like Sprintlink and Tiscali, the solution space is even larger. Thus, more iterations (e.g., approximately 90,000 and 100,000 iterations)

TABLE III

COMPARISON OF AVERAGE LOAD BALANCING PERFORMANCE RATIO WITH DIFFERENT SETS OF HYPERPARAMETERS

	filters / neurons = 128, β = 0.1
α = 0.01 (with decay)	0.761
α = **0.001** (with decay)	**0.970**
α = 0.0001[*]	0.963

[*] without decay, since the initial learning rate is equal to the minimum learning rate.

(a)

	α = 0.001 (with decay), β = 0.1
filters / neurons = 64	0.928
filters / neurons = **128**	**0.970**
filters / neurons = 256	0.837

(b)

	filters / neurons = 128, α = 0.001 (with decay)
β = **0.1**	**0.970**
β = 0.01	0.958

(c)

should be taken to train a good policy, and each iteration takes approximately 2 seconds. Note that this cost is incurred offline and can be performed infrequently depending on environment stability. The policy neural network as described in Section VI-A.1 is relatively small. Thus, the inference time for the Abilene and EBONE networks are less than 1 second, and they are less than 2 seconds for the Sprintlink and Tiscali networks.

5) Hyperparameters: Table III shows that how hyperparameters affect the load balancing performance of CFR-RL in the Abilene network. For each set of hyperparameters, we trained a policy for the Abilene network by 10,000 iterations, and then evaluated the average load balancing performance ratio over the whole test set. We only present the results for the Abilene network, since the results for other network topologies are similar. In Tab. III(a), the number of filters in the convolutional layer and neurons in the fully connected layer is fixed to 128 and entropy factor β is fixed to 0.1. We compare the performance with different learning rate α. The results show that training might become unstable if the initial learning rate is too large (e.g., 0.01), and thus it cannot converge to a good policy. In contrast, training with a smaller learning rate is more stable but might require longer training time to further improve the performance. As a result, we chose $\alpha = 0.001$ to encourage exploration in the early stage of training. We compared the performance with different sizes of filters and neurons in Tab. III(b). The results show that too few filters/neurons might restrict the representation that the neural network can learn and thus Cause under-fitting. Meanwhile, too many neurons might cause over-fitting, and thus the corresponding policy cannot generalize well to the test set. In addition, more training time is required for a larger neural network. In Tab. III(c), the results show that a larger entropy factor encourages exploration and leads to a better performance. Overall, the set of hyperparameters we have chosen is a good trade-off between performance and computational complexity of the model.

VII. CONCLUSION AND FUTURE WORK

With an objective of minimizing the maximum link utilization in a network and reducing disturbance to the network causing service disruption, we proposed CFR-RL, a scheme that learns a critical flow selection policy automatically using reinforcement learning, without any domain-specific rule-based heuristic. CFR-RL selects critical flows for each given traffic matrix and reroutes them to balance link utilization of the network by solving a simple rerouting optimization problem. Extensive evaluations show that CFR-RL achieves near-optimal performance by rerouting only a limited portion of total traffic. In addition, CFR-RL generalizes well to traffic matrices which are not explicitly trained.

Yet, there are several aspects that may help improve the solution that we proposed in this contribution. Among them, we determine how CFR-RL can be updated and improved.

A. Objectives

CFR-RL could be formulated to achieve other objectives. For example, to minimize overall end-to-end delay in the network (i.e., $\Omega = \sum_{\langle i,j\rangle \in E} (\frac{l_{i.j}}{c_{i,j}-l_{i,j}})$) described in Section VI-A.4(2), we can define reward r as $1/\Omega$ and reformulate the rerouting optimization problem (4a) to minimize Ω.

Table II shows an interesting finding. Although CFR-RL does not explicitly minimize rerouting traffic, it ends up rerouting much less traffic (i.e., 10.0%-21.3%) and performs better than rule-based heuristic schemes by 1.3%-12.2%. This reveals that CFR-RL effectively searches the whole set of candidate flows to find the best critical flows for various traffic matrices, rather than simply considering the elephant flows on the most congested links or in the whole network as rule-based heuristic schemes do. We will consider minimizing rerouting traffic as one of our objectives and investigate the trade-off between maximizing performance and minimizing rerouting traffic.

B. Scalability

Scaling CFR-RL to larger networks is an important direction of our future work. CFR-RL relies on LP to produce reward signals r. The LP problem would become complex as the number of critical flows K and the size of a network increase. This would slow down the policy training for larger networks (e.g., the Tiscali network in Section VI-B.4), since the time consumed for each iteration would increase. Moreover, the solution space would become enormous for larger networks, and RL has to take more iterations to converge to a good policy. To further speed up training, we can either spawn even more actor agents (e.g., 30) in parallel to allow the system to consume more data at each time step and thus improve exploration [28], or apply GA3C [36] to offload the training to a GPU, which is an alternative architecture of A3C and emphasizes on an efficient GPU utilization to increase the number of training data generated and processed per second. Another possible design to mitigate the scalability issue is adopting SDN multi-controller architectures. Each controller takes care of a subset of routers in a large network, and one CFR-RL agent is running on each SDN controller. The corresponding problem naturally falls into the realm of Multi-Agent Reinforcement Learning. We will evaluate if a multi-SDN controller architecture can help provide additional improvement in our approach.

C. Retraining

In this paper, we mainly described the RL-based critical flow selection policy training process as an offline task. In other words, once training is done, CFR-RL remains unmodified after being deployed in the network. However, CFR-RL can naturally accommodate future unseen traffic matrices by periodically updating the selection policy. This self-learning technique will enable CFR-RL to further adapt itself to the dynamic conditions in the network after being deployed in real networks. CFR-RL can be retrained by including new traffic matrices. For example, the outlier traffic matrices (e.g., the 235th-240th traffic matrices in Day 2) presented in Fig. 10 should be included for retraining, while the generalization results shown in Section VI-B.3 suggest that retraining frequently might not be necessary. Techniques to determine when to retrain and which new/old traffic matrix should be included/excluded in/from the training dataset should be further investigated.

The above examples are some key issues that are left for future work.

Acknowledgment

The authors would like to thank the editors and reviewers for providing many valuable comments and suggestions.

References

[1] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[2] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE Int. Conf. Comput. Commun.*, Apr. 2013, pp. 2211–2219.

[3] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in SDN/OSPF hybrid network," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 563–568.

[4] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Dynamic hybrid routing: Achieve load balancing for changing traffic demands," in *Proc. IEEE 22nd Int. Symp. Qual. Service (IWQoS)*, May 2014, pp. 105–110.

[5] J. Zhang, K. Xi, and H. J. Chao, "Load balancing in IP networks using generalized destination-based multipath routing," *IEEE/ACM Trans. Netw.*, vol. 23, no. 6, pp. 1959–1969, Dec. 2015.

[6] Z. Guo, W. Chen, Y.-F. Liu, Y. Xu, and Z.-L. Zhang, "Joint switch upgrade and controller deployment in hybrid software-defined networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1012–1028, May 2019.

[7] Y. Wang and Z. Wang, "Explicit routing algorithms for Internet traffic engineering," in *Proc. IEEE Int. Conf. Comput. Commun. Netw.*, Oct. 1999, pp. 582–588.

[8] E. D. Osborne and A. Simha, *Traffic Engineering With MPLS*. Indianapolis, IN, USA: Cisco Press, 2002.

[9] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756–767, May 2002.

[10] K. Holmberg and D. Yuan, "Optimization of Internet protocol network design and routing," *Networks*, vol. 43, no. 1, pp. 39–53, Jan. 2004.

[11] J. Chu and C.-T. Lea, "Optimal link weights for IP-based networks supporting hose-model VPNs," *IEEE/ACM Trans. Netw.*, vol. 17, no. 3, pp. 778–788, Jun. 2009.

[12] J. Zhang, K. Xi, L. Zhang, and H. J. Chao, "Optimizing network performance using weighted multipath routing," in *Proc. 21st Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2012, pp. 1–7.

[13] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Load balancing for multiple traffic matrices using SDN hybrid routing," in *Proc. IEEE 15th Int. Conf. High Perform. Switching Routing (HPSR)*, Jul. 2014, pp. 44–49.

[14] C. Villamizar, *OSPF Optimized Multipath (OSPF-OMP)*, draft-ietf-ospf-omp-02, IETF Internet-Draft, 1999.

[15] M. Kodialam, T. V. Lakshman, J. B. Orlin, and S. Sengupta, "Oblivious routing of highly variable traffic in service overlays and IP backbones," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 459–472, Apr. 2009.

[16] M. Antic, N. Maksic, P. Knezevic, and A. Smiljanic, "Two phase load balanced routing using OSPF," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 1, pp. 51–59, Jan. 2010.

[17] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Proc. Workshop Big Data Anal. Mach. Learn. Data Commun. Netw. (Big-DAMA)*, 2018, pp. 40–45.

[18] P. Sun, J. Li, Z. Guo, Y. Xu, J. Lan, and Y. Hu, "SINET: Enabling scalable network routing with deep reinforcement learning on partial nodes," in *Proc. ACM SIGCOMM Conf. Posters Demos SIGCOMM Posters Demos*, 2019, pp. 88–89.

[19] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2016, pp. 25–33.

[20] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2018, pp. 1871–1879.

[21] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 191–205.

[22] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of SDN using wildcard requests," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[23] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.

[24] T. P. Lillicrap *et al.* (2016). *Continuous Control With Deep Reinforcement Learning*. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.htmlLillicrapHPHETS15

[25] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trustregion policy optimization," vol. 37, pp. 1889–1897, Jul. 2015. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html

[26] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proc. ACM Workshop Hot Topics Netw.*, 2017, pp. 185–191.

[27] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[28] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," vol. 48, pp. 1928–1937, Jun. 2016. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html

[29] Y. Wang, Z. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *Proc. IEEE Int. Conf. Comput. Commun.*, Apr. 2001, pp. 565–571.

[30] Gurobi Optimization. (2019). *Gurobi Optimizer Reference Manual*. [Online]. Available: http://www.gurobi.com

[31] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. Workshop Deep Learn. Audio, Speech Lang. Process. (ICML)*, 2013, pp. 1–6.

[32] M. Abadi and *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Conf. Oper. Syst. Design Implement.*, 2016, pp. 265–283.

[33] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 133–145, 2002.

[34] *Yin Zhang's Abilene*. Accessed: Apr. 22, 2019. [Online]. Available: http://www.cs.utexas.edu/yzhang/research/AbileneTM/

[35] *TMgen: Traffic Matrix Generation Tool*. [Online]. Available: https://tmgen.readthedocs.io/en/latest/

[36] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "GA3C: GPU-based A3C for deep reinforcement learning," *CoRR*, vol. 1611.06256, 2016, [Online]. Available: http://arxiv.org/abs/1611.06256

Junjie Zhang (Member, IEEE) received the B.S. degree in computer science from the Nanjing University of Posts and Telecommunications, China, in 2006, and the M.S. degree in computer science and the Ph.D. degree in electrical engineering from New York University, New York City, NY, USA, in 2010 and 2015, respectively.

He has been with Fortinet Inc., Sunnyvale, CA, USA, since 2015. He holds two U.S. patents in the area of computer networking. His research interests include network optimization, traffic engineering, machine learning, and network security.

Minghao Ye received the B.E. degree in microelectronic science and engineering from Sun Yat-sen University, Guangzhou, China, and the second B.E. degree (Hons.) in electronic engineering from The Hong Kong Polytechnic University, Hong Kong, in 2017, the M.S. degree in electrical engineering from New York University, New York City, NY, USA, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering. His research interests include traffic engineering, software-defined networks, mobile edge computing, and reinforcement learning.

Zehua Guo (Senior Member, IEEE) received the B.S. degree from Northwestern Polytechnical University, the M.S. degree from Xidian University, and the Ph.D. degree from Northwestern Polytechnical University. He was a Research Fellow with the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, a Post-Doctoral Research Associate with the Department of Computer Science and Engineering, University of Minnesota Twin Cities, and a Visiting Associate Professor with the Singapore University of Technology and Design. He is currently an Associate Professor at the Beijing Institute of Technology. His research interests include software-defined networking, network function virtualization, data center networks, cloud computing, content delivery networks, network security, machine learning, and Internet exchange. He was the Session Chair of the IEEE International Conference on Communications 2018 and the Technical Program Committee Member of *Computer Communications* (Elsevier). He is an Associate Editor of the IEEE ACCESS and *EURASIP Journal on Wireless Communications and Networking* (Springer), and an Editor of *KSII Transactions on Internet and Information Systems*.

Chen-Yu Yen received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2014, and the M.S. degree in electrical engineering from Columbia University in 2018. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, New York University, New York City, NY, USA. His research interests include reinforcement learning, congestion control, and practical machine learning for networking.

H. Jonathan Chao (Fellow, IEEE) received the B.S. and M.S. degrees in electrical engineering from National Chiao Tung University, Taiwan, in 1977 and 1980, respectively, and the Ph.D. degree in electrical engineering from The Ohio State University, Columbus, OH, USA, in 1985. He was the Head of the Electrical and Computer Engineering (ECE) Department, New York University (NYU), New York City, NY, USA, from 2004 to 2014. From 2000 to 2001, he was the Co-Founder and the CTO of Coree Networks, Tinton Falls, NJ, USA. From 1985 to 1992, he was a member of Technical Staff at Bellcore, Piscataway, NJ, USA, where he was involved in transport and switching system architecture designs and application-specified integrated circuit implementations, such as the world's first SONET-like framer chip, ATM layer chip, sequencer chip (the first chip handling packet scheduling), and ATM switch chip. He is currently a Professor of ECE at NYU, New York City. He is also the Director of the High-Speed Networking Laboratory. He has been doing research in the areas of software-defined networking, network function virtualization, datacenter networks, high-speed packet processing/switching/routing, network security, quality-of-service control, network on chip, and machine learning for networking. He has coauthored three networking books, *Broadband Packet Switching Technologies: A Practical Guide to ATM Switches and IP Routers* (Wiley, 2001), *Quality of Service Control in High-Speed Networks* (Wiley, 2001), and *High Performance Switches and Routers* (Wiley, 2007). He has published more than 260 journal and conference papers and holds 63 patents. He is a Fellow of the National Academy of Inventors. He was a recipient of the Bellcore Excellence Award in 1987. He was a co-recipient of the 2001 Best Paper Award from the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.