Mitigating Routing Update Overhead for Traffic Engineering by Combining Destination-Based Routing With Reinforcement Learning

Minghao Ye, Yang Hu, Junjie Zhang, *Member, IEEE*, Zehua Guo, *Senior Member, IEEE*, and H. Jonathan Chao, *Life Fellow, IEEE*

***Abstract*—Traffic Engineering (TE) is a widely-adopted network operation to optimize network performance and resource utilization. Destination-based routing is supported by legacy routers and more readily deployed than flow-based routing, where the forwarding entries could be frequently updated by TE to accommodate traffic dynamics. However, as the network size grows, destination-based TE could render high time complexity when generating and updating many forwarding entries, which may limit the responsiveness of TE and degrade network performance. In this paper, we propose a novel destination-based TE solution called FlexEntry, which leverages emerging Reinforcement Learning (RL) to reduce the time complexity and routing update overhead while achieving good network performance simultaneously. For each traffic matrix, FlexEntry only updates a few forwarding entries called *critical entries* for redistributing a small portion of the total traffic to improve network performance. These critical entries are intelligently selected by RL with traffic split ratios optimized by Linear Programming (LP). We find out that the combination of RL and LP is very effective. Our simulation results on six real-world network topologies show that FlexEntry reduces up to 99.3% entry updates on average and generalizes well to unseen traffic matrices with near-optimal load balancing performance.**

***Index Terms*—Reinforcement learning, traffic engineering, destination-based routing, routing update overhead, linear programming.**

I. Introduction

TRAFFIC Engineering (TE) is a network operation adopted by service providers to improve network performance under different network conditions and service requirements. With a common optimization objective (e.g., minimizing the maximum link utilization in the network), TE computes an optimal routing strategy and configures the routing across Wide-Area Networks (WANs) to control traffic distribution accordingly [1]–[4]. One typical TE solution is flow-based routing, where each flow refers to a unique source-destination pair in the network [5], [6]. Given a set of pre-generated paths in the network, each flow can be forwarded along the paths with different traffic split ratios based on the forwarding entries installed in the routing table of each router. Upon traffic changes, the centralized controller would compute a new routing strategy to accommodate the traffic dynamics, and then update the flow-based forwarding entries to facilitate deliberate routing policies and realize fine-grained traffic control.

However, flow-based routing may suffer from scalability issues. For a network with P prefixes, each router has to store $O(P^2)$ forwarding entries in the worst case to distinguish the source and destination addresses of the packets. Over recent years, the size of the Internet routing table has increased superlinearly [7]. As reported in [7], today's routing tables could include approximately 900,000 entries, which poses a great challenge to storing a huge number of forwarding entries in a router. To support flow-based routing with high-speed packet processing, a complex Ternary Content-Addressable Memory (TCAM)-based routing table is used by routers [8]. Due to the high cost-to-density ratio and high power consumption of TCAM [9], [10], a router usually has limited TCAM resources [11] and thus cannot accommodate a huge number of forwarding entries [8], [12]. Moreover, looking up a route in a large routing table with a huge number of forwarding entries may increase the forwarding delay of the packets [13].

An alternative solution for TE is to use destination-based routing, where routers make forwarding decisions based on the destination address of the packets [3], [4], [14]. With destination-based forwarding, each router only needs to maintain a forwarding table with at most $O(P)$ entries for a network with P prefixes, which can effectively reduce the forwarding complexity compared to flow-based routing. Moreover, destination-based forwarding has been widely implemented in legacy routers with simple Random-Access Memories (RAMs) that are cheaper and more energy-efficient compared to TCAMs [14]. Traditional destination-based routing protocols (e.g., Open Shortest Path First [15]) usually forward the packets along the shortest paths with low link costs. To responsively accommodate dynamic traffic variations, destination-based TE needs to update routing frequently. For example, in Amazon's inter-region backbone network, TE operations are performed at a time scale of seconds by rerouting traffic to satisfy diverse requirements of applications [16].

However, destination-based routing faces several challenges. First, generating optimal traffic split ratios for all entries could be time-consuming since the computation complexity of solving traffic rerouting optimization problems in large networks

An earlier version of this paper was presented in part at the ACM SIGCOMM 2020 Workshop on Network Meets AI & ML (NetAI 2020).

This paper was supported by the National Natural Science Foundation of China under Grant 62002019 and the Beijing Institute of Technology Research Fund Program for Young Scholars. *(Corresponding author: Zehua Guo.)*

Minghao Ye, Yang Hu, and H. Jonathan Chao are with the Department of Electrical and Computer Engineering, New York University, New York City, NY 11201 USA (e-mail: minghao.ye@nyu.edu; yh3751@nyu.edu; chao@nyu.edu).

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

Zehua Guo is with Beijing Institute of Technology, Beijing 100081, China (e-mail: guolizihao@hotmail.com).

is generally high [4], [17]. Second, frequently updating a large number of forwarding entries at each router requests high management overhead and introduces a long delay for routing updates. One promising solution is to selectively update a few forwarding entries when traffic changes. However, given the numerous solution space and dynamic network conditions, it would be very challenging to design a heuristic algorithm for identifying such "critical" forwarding entries to improve network performance with low routing update overhead.

In this paper, we propose FlexEntry, a destination-based TE solution to achieve near-optimal network performance and mitigate routing update overhead simultaneously. FlexEntry forwards majority traffic using Equal-Cost Multi-Path (ECMP) [18] and then selectively and dynamically redistributes a small portion of total traffic by installing a few *critical entries* as traffic changes. Here, critical entries can be viewed as a set of destination-based forwarding entries that contribute the most to improving network performance. For each given Traffic Matrix (TM), FlexEntry employs Reinforcement Learning (RL) to learn a policy for efficiently and effectively selecting an appropriate number of critical entries, and then obtains the corresponding loop-free rerouting split ratios by formulating and solving a Linear Programming (LP) optimization problem. Since most of the traffic is forwarded by static ECMP, the computation complexity of TE optimization and time complexity of entry updates would be greatly reduced.

The main contributions of this paper are summarized as follows:

1) We customize a 2-stage RL approach with a carefully designed reward function to identify flexible numbers of critical entries in different network scenarios.
2) We adopt LP to produce reward signals for RL and optimize traffic split ratios for the selected critical entries to control traffic distribution. This RL + LP combined approach turns out to be effective in mitigating routing update overhead during frequent TE operations.
3) We evaluate FlexEntry by conducting extensive simulations on six real-world network topologies with both real and synthesized traffic. The simulation results show that FlexEntry achieves near-optimal performance with good generalization over unseen traffic scenarios and saves at most 99.3% forwarding entry updates on average.

The remainder of this paper is organized as follows. Section II lists the related works. Section III provides an overview of our system design and describes the workflow of FlexEntry. Section IV explains the proposed 2-stage RL model in detail. Section V presents the LP formulations for traffic rerouting optimization. Section VI evaluates the effectiveness of our scheme. Section VII concludes the paper and discusses future work.

II. Related Works

Flow-based routing has been adopted by TE to support fine-grained traffic control and improve network performance. In Multiprotocol Label Switching (MPLS) networks, a routing problem is typically formulated as an optimization problem, where explicit paths are obtained for each source-destination pair to distribute flows [1], [2]. Dynamic hybrid routing [5] realizes load balancing for multiple traffic scenarios by dynamically re-balancing traffic to accommodate traffic fluctuations with a preconfigured routing policy. SMORE [6] adopts an oblivious routing algorithm to generate a set of preconfigured paths for each flow and further solves an optimization problem to optimize the path split ratios for all flows. In the inter-datacenter WANs of Google [19] and Microsoft [20], flow-based TE solutions have been deployed to achieve high utilization. However, to distinguish source and destination addresses, flow-based TE would suffer from scalability issues with a huge number of forwarding entries installed in the routing table.

An alternative solution for TE is destination-based routing, which is widely supported by legacy routers. Using OSPF and ECMP protocols, [3], [21], [22], and [23] attempt to balance link utilization of the network by carefully adjusting the link costs to select path in ECMP. However, it has been shown that it is an NP-hard problem to optimize the link costs for a network [21], [23], while the even traffic distribution imposed by ECMP introduces additional limitations on routing optimization [4]. One recent work [24] leverages the emerging segment routing techniques to perform joint optimization of link costs and segments that specifies the intermediate way-points. However, the above-mentioned related works would update link costs at routers and wait for OSPF reconvergence, which may lead to potential network disturbance and service disruption [25]. To improve the performance of destination-based TE, weighted ECMP [4] extends ECMP by allowing weighted traffic splitting at each node to achieve significant performance improvement over ECMP. For a given TM, Zhang et al. [14] proved that an arbitrary flow-based routing can be converted to a loop-free destination-based routing without any performance penalty, which is guaranteed to achieve optimal performance. However, these existing works do not consider the adversary impact of routing updates under frequent TE operations. In contrast, by updating a small set of critical entries in the network, FlexEntry can avoid the potential network disturbance caused by OSPF reconvergence with low routing update overhead.

In recent years, machine learning techniques have been used in TE design to improve network performance. In [26], an automatic network protocol is designed for backbone networks using semi-supervised deep learning. To minimize signaling delay in large-scale networks, QoS-aware Adaptive Routing [27] uses a distributed three-level control plane architecture coupled with RL. Xu et al. [28] employ RL to optimize network performance (i.e., throughput and delay). CFR-RL [29] and DATE [30] leverage RL to identify and reroute critical flows in the network to reduce network disturbance. In multi-region networks, Geng et al. [31] adopt multi-agent RL to obtain routing decisions for each network region in a distributed manner, while FedTe [32] combines supervised learning and graph neural network to predict the optimal distributions of cross-region traffic at each network region. However, all of the above-mentioned related works do not take mitigating routing update overhead as an objective.

While many existing works adopt fully learning-based approaches to determine routing policies with a focus on

network performance, the novelty of our approach relies on the combination of the emerging RL techniques and traditional LP optimization methods to reduce routing update overhead. Indeed, it could be a good choice to leverage the agility of learning-based methods to make TE solutions more adaptive. However, LP also has a unique advantage to allocate network traffic in an efficient and optimal way. Thus, we aim at combining these two powerful techniques to address the existing routing update overhead problem that is not well-studied in the past from the perspective of TE. Moreover, we adopt a novel 2-stage RL design that pre-trains several sub-models to select different numbers of critical entries and uses a single model to select a suitable sub-model for different traffic scenarios. It is worth mentioning that our 2-stage RL design might be similar to the recently studied meta-learning [33] and ensemble learning [34] schemes. However, the sub-models in meta-learning could be trained along with the single model with different types of datasets and tasks, while ensemble learning aims at constructing a single good model by aggregating multiple sub-models trained with different splits of dataset or algorithms. On the contrary, the training procedure of FlexEntry is performed sequentially and all the sub-models are trained with the same dataset and objective function.

III. System Description

In this section, we explain our motivation, overview FlexEntry's design, and provide an illustrative example to describe the advantages of critical entry updates.

A. Motivation

To identify critical entries for a given TM, four problems should be solved:

1) How many critical entries should be installed to guarantee promising network performance?
2) At which routers should critical entries be installed?
3) To which destinations the traffic belongs should be redistributed, once the routers for installing critical entries are determined?
4) How to redistribute the selected traffic among available next hops?

Once the critical entries are identified, it is relatively simple to address the problem (4) by formulating and solving an LP optimization problem. However, it is not trivial to solve problems (1), (2), and (3) due to the large solution space. Given a network with N routers, there would be a total of $N * (N-1)$ candidate router-destination pairs (i.e., entries). With consideration of all possible critical entry numbers K, there are a total of $\sum_{K=1}^{N*(N-1)} \binom{N*(N-1)}{K}$ possible combinations in the solution space. For a network with $N = 10$ nodes[1], the solution space for selecting a combination of $K = 9$ critical entries has $\binom{90}{9} \approx 706$ billion combinations. Considering all possible combinations, it is very challenging to design a heuristic algorithm to solve problems (1)-(3) based on fixed and simple rules, as later shown in Section VI-B.

[1]Each router can be viewed as a network node. We use these two terms interchangeably.

One possible way to address the above-mentioned problems is to incorporate the selection of critical entries into traditional routing optimization problems. The objective of this joint optimization problem is to minimize the number of critical entry updates and the maximum link utilization in the network simultaneously. In addition to the destination-based routing variables to be solved, it is necessary to determine for each candidate entry whether it should be selected for routing updates or not. If yes, such entry would be treated as a critical entry and the corresponding traffic split ratios should be optimized; otherwise, the entry would forward the traffic based on a default routing scheme (e.g., ECMP). Such critical entry indicators can only take binary values, which can be viewed as integer constraints. As a result, this joint optimization problem is formulated as a Mixed-Integer Linear Programming (MILP) problem, which is proved to be NP-hard [35].

Emerging RL sheds light on problems (1)-(3) for critical entry selection, and FlexEntry receives the benefits of RL from three aspects. First, RL can model complex selection policies as neural networks and provide a scalable and expressive way to incorporate various "raw" observations (e.g., TMs) into the selection policy. Second, RL is able to train for objectives that lack precise models and are thus hard to optimize directly, as long as reward signals exist and correlate with the objective. Third, by continually learning to make better selections through reinforcement in the form of reward signals, RL can optimize its selection policy under varying network scenarios.

We use LP to solve the problem (4) because of two reasons. First, given that only a few critical entries are selected, the set of rerouting split ratios would be relatively small. LP can optimally solve the rerouting problem with fewer routing variables. Second, traffic split ratios are continuous numbers. Existing works show that the RL methods for continuous action domain [36], [37] would lead to slow and ineffective learning when the number of output parameters is relatively large [28], [38]. In other words, applying RL methods to solve the problem (4) may suffer from scalability issues.

B. Overview

For a given TM, the tasks of FlexEntry are to (1) decide how many critical entries (i.e., K) are required to achieve near-optimal performance with the lowest routing update overhead, (2) select a good combination of K router-destination pairs as critical entries, and (3) redistribute the selected traffic among *admissible next hops*[2] to balance link utilization of the network based on critical entries. Overall, """""""""a very challenging task, considering the variety of the TM and the complexity of the network topology.

To achieve the objective, we exploit an RL + LP combined approach with a two-stage RL design. In the beginning, we train multiple RL sub-models in the first training stage, where each sub-model is responsible for identifying a unique number of critical entries (e.g., $K = 10$ for one sub-model and $K = 20$ for another sub-model). The settings of K value

[2]An admissible next hop is defined as a next hop that would not cause a forwarding loop.

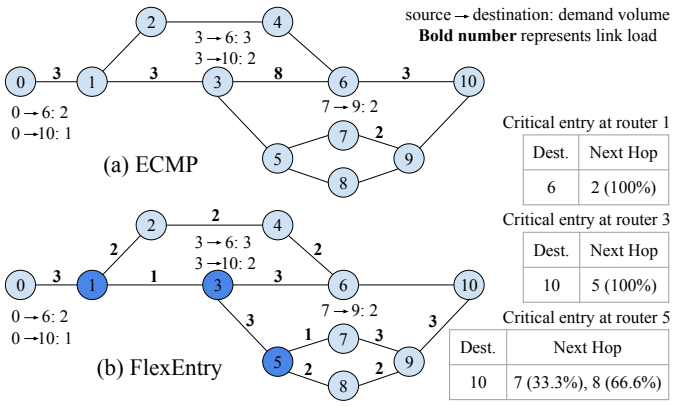


Fig. 1. Illustrative example of FlexEntry to redistribute traffic with critical entries. Each link is bidirectional with link cost and capacity equal to 10.

for different sub-models are later discussed in Section VI-B1. Each sub-model learns a selection policy over a rich variety of historical TMs, where TMs can be measured and collected by a centralized controller periodically [39]. The selection policy of each sub-model is represented as a neural network that maps a "raw" observation (e.g., a given TM) to a combination of K different critical entries.

Once the first training stage is done, we train a single model in the second training stage to find a proper K value for the task (1), which will be responsible for determining which sub-model should be used to generate the corresponding K critical entries for the task (2). This single model also learns over the same TM dataset in the first training stage, but its selection policy is represented as a different neural network from the sub-models, which maps the input to one of the sub-models to be used (i.e., p_j). For both two training stages, we formulate and solve an LP optimization problem (described in Section V-A) to generate the optimal traffic split ratios for critical entries for the task (3), and obtain the resulting maximum link utilization as part of the reward signal (see Section IV). Through reinforcement in the form of reward signal, all neural networks are trained based on REINFORCE algorithm [40] with some customizations.

The whole training procedure can take place in an offline server with a TM dataset collected in advance. After that, FlexEntry can be deployed in a centralized controller. Given a newly measured TM, FlexEntry would leverage the single model trained in the second stage to choose an appropriate sub-model, where the selected sub-model is responsible for identifying the corresponding K critical entries under the current traffic scenario. Then, the optimal rerouting split ratios for critical entries can be derived by solving the LP optimization problem (9a) presented in Section V-A, such that the centralized controller would install/update new critical entries at the corresponding routers to redistribute traffic accordingly. Note that the critical entries installed in the previous period would timeout automatically.

C. Updating Critical Entries

FlexEntry aims to effectively balance link utilization of the network by smartly installing/updating a small set of critical destination-based forwarding entries at some routers. We illustrate how FlexEntry works with a simple example in Fig. 1, where each link is bidirectional with link cost and capacity equal to 10. Assuming that node 0 sends 2 units of traffic to node 6 and 1 unit of traffic to node 10, respectively. Similarly, node 3 sends 3 units of traffic to node 6 and 2 units of traffic to node 10, respectively. In addition, node 7 sends 2 units of traffic to node 9. Initially, all traffic is distributed along the shortest paths according to default ECMP routing, and the traffic load on each link is shown in Fig. 1(a). Given that ECMP routing is static and not traffic-aware, link $\langle 3, 6\rangle$ becomes the bottleneck link with 80% link utilization under the current traffic scenario.

FlexEntry can install a few critical entries[3] at several routers to effectively improve network performance with low routing update overhead. As depicted in Fig. 1(b), FlexEntry updates a critical destination-based forwarding entry at router 1 to reroute 2 units of traffic originated from node 0 and destined to node 6 by specifying an admissible next hop router 2. Once the rerouted traffic arrives at router 2, it would be forwarded to its destination node 6 along the shortest path according to ECMP routing. In this situation, the congestion of link $\langle 3, 6\rangle$ would be alleviated by leveraging previously underutilized links. Similarly, the traffic destined to node 10 aggregated at router 3 can be rerouted to next hop router 5 instead of ECMP next hop router 6 to reduce the load of the bottleneck link $\langle 3, 6\rangle$ with a critical entry at router 3. Moreover, when the rerouted traffic reaches router 5, it can be further split between admissible next hops router 7 and router 8 with different weights to achieve better load balancing performance according to the critical entry. By forwarding 1 unit of traffic (33.3%) to router 7 and 2 units of traffic (66.6%) to router 8, the maximum link utilization in the network can be reduced to 30%, while the remaining traffic is still distributed by the static ECMP routing. Thus, the above example shows that FlexEntry can achieve load balancing by complementing ECMP routing with only three critical destination-based forwarding entries.

IV. Proposed Model

In this section, we provide the details of our proposed 2-stage RL design, including RL problem formulations and training procedures to learn a critical entry selection policy and a sub-model selection policy.

A. Learning a Critical Entry Selection Policy

1) First-Stage RL Formulation: In the first RL training stage, there are m sub-models (i.e., $p_1, p_2, ..., p_m$) to be trained with different K value settings (e.g., K_j for sub-model p_j). The detailed settings can be found in Section VI-B1. The goal of each sub-model p_j is to learn a policy π that selects a combination of K_j "right" router-destination pairs for each given TM, such that the network performance is maximized after redistributing traffic according to the selected critical entries at the corresponding nodes.

Input: An input instance s is represented as a traffic matrix TM, which consists of traffic demands for all source-destination pairs in a certain period.

[3]Critical entries take strict precedence over ECMP entries.

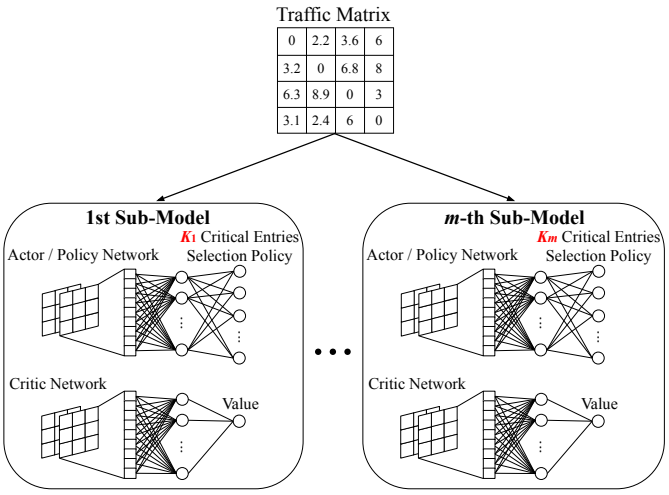


Fig. 2. First training stage with m sub-models to be trained in parallel. Each sub-model p_j samples a unique number of K_j critical entries.

Action Space: For a given instance s, each sub-model p_j needs to select K_j router-destination pairs as critical entries. For a network with N nodes, $N-1$ destination-based forwarding entries can be inserted into each router to forward the traffic to the remaining $N-1$ destination nodes, which means there are a total of $N*(N-1)$ candidate router-destination pairs. In this scenario, this RL problem would require a large action space of size $\binom{N*(N-1)}{K_j}$ for each sub-model p_j. Inspired by [41] and [42], we define the action space as {0, 1, ..., $N*(N-1)-1$} and allow each sub-model p_j to sample K_j different actions for each instance s (i.e., $a^1, a^2, ..., a^{K_j}$).

Reward: For each sub-model p_j, K_j different router-destination pairs would be sampled for a given instance s. Then, the LP optimization problem (9a) described in Section V-A can be solved for each sub-model to obtain the maximum link utilization U. To evaluate how far the performance of the derived routing is being from optimal routing, a performance ratio is applied and defined as below:

$$PR = \frac{U_{\text{optimal}}}{U}, \tag{1}$$

where U_{optimal} is the maximum link utilization achieved by the optimal destination-based routing. Such an optimal solution is derived from the LP formulations presented in [14] by optimizing the traffic split ratios for all destination-based forwarding entries, which is proved to achieve the same performance as optimal flow-based explicit routing. A higher PR indicates that the derived routing can achieve better performance and is closer to optimal routing. When $PR = 1$, it achieves the same performance as optimal routing. For the first training stage, our objective is to select the best K_j entries for each sub-model p_j to improve network performance as much as possible. Therefore, we define the reward function as $r = PR$ for each sub-model, which is set to reflect the network performance after redistributing traffic according to the selected critical entries. The greater reward r, the higher PR and thus better performance.

2) First-Stage Training Algorithm: For the first training stage, we need to train m sub-models. For each sub-model p_j, we use a neural network to represent the policy, where the policy network takes a TM as an input and outputs a probability distribution $\pi(a|s)$ over all available actions. Fig. 2 provides the details of the first training stage, including the actor-critic architecture of all m sub-models. Note that the hyperparameter settings are listed in Section VI-A3. For each sub-model p_j with K_j actions to be selected ($j = 1, 2, ..., m$), we define a solution $a_{K_j} = (a^1, a^2, ..., a^{K_j})$ as a combination of K_j sampled actions since we do not care about the order of the sampled actions. In other words, when selecting a solution a_{K_j} with a given instance s, a stochastic policy $\pi(a_{K_j}|s)$ parameterized by θ_j can be approximated as follows[4]:

$$\pi_{\theta_j}(a_{K_j}|s) \approx \prod_{i=1}^{K_j} \pi_{\theta_j}(a^i|s). \tag{2}$$

Recall that the goal of each sub-model p_j is to find a policy π_{θ_j} that maximizes the network performance over various TMs for a given K_j, i.e., maximizes the expected reward $E_{\pi_{\theta_j}(a_{K_j}|s)}[r]$. Thus, we optimize $E_{\pi_{\theta_j}(a_{K_j}|s)}[r]$ by gradient ascend, using REINFORCE algorithm with a baseline $b(s)$:

$$\nabla_{\theta_j} E_{\pi_{\theta_j}(a_{K_j}|s)}[r] = E_{\pi_{\theta_j}}[\nabla_{\theta_j} log \pi_{\theta_j}(a_{K_j}|s)(r - b(s))]. \tag{3}$$

A good baseline $b(s)$ reduces gradient variance and thus increases the speed of learning. For each sub-model p_j, we use a learned estimate of the value function $V^{\pi_{\theta_j}}(s)$ as the baseline $b(s)$. The critic network of each sub-model p_j in Fig. 2 is trained to learn an estimate of $V^{\pi_{\theta_j}}(s)$, and the critic network parameter θ_{v_j} is updated according to the following equation:

$$\theta_{v_j} \leftarrow \theta_{v_j} - \alpha_{v_j} \sum_s \nabla_{\theta_{v_j}} (r - V_{\theta_{v_j}}^{\pi_{\theta_j}}(s))^2, \tag{4}$$

where $V_{\theta_{v_j}}^{\pi_{\theta_j}}(\cdot)$ is outputted by the critic network of sub-model p_j as the estimate of $V^{\pi_{\theta_j}}(\cdot)$, and α_{v_j} is the learning rate for the critic network of sub-model p_j. Note that the critic network is only trained to estimate the expected reward r, and solely helps train the policy network. Once training is done, only the policy network is required for each sub-model to execute the action selection.

To ensure that each sub-model explores the action space adequately during training to discover good policies, we add the entropy of the policy π to Eq. (3). This technique improves exploration by discouraging premature convergence to suboptimal deterministic policies [43]. Then, the policy network parameter θ_j of each sub-model p_j is updated according to the following equation:

$$\begin{aligned}\theta_j \leftarrow \theta_j + \alpha_j \sum_s \nabla_{\theta_j} log \pi_{\theta_j}(a_{K_j}|s)(r - V_{\theta_{v_j}}^{\pi_{\theta_j}}(s)) + \\ \beta_j \nabla_{\theta_j} H(\pi_{\theta_j}(\cdot|s)),\end{aligned} \tag{5}$$

where α_j is the learning rate for the policy network of sub-model p_j, and H is the entropy of the policy (the probability distribution over actions). The hyperparameter β_j controls the

[4]To select K_j distinct actions for the j-th sub-model, we perform action sampling without replacement. The right side of Eq. (2) is the solution probability when sampling with replacement, but we still use Eq. (2) to approximate the probability of solution a_{K_j} given an instance s for simplicity.

Algorithm 1 First-Stage Training Algorithm for Sub-Model

Initialize θ_j, θ_{v_j} for the j-th sub-model with K_j actions to be sampled
for each iteration **do**
  $\Delta\theta_j \leftarrow 0$, $\Delta\theta_{v_j} \leftarrow 0$
  $\{s_i\} \leftarrow$ Sample a batch of instances with size B
  for $i = 1, ..., B$ **do**
    Sample a solution $a_{i_{K_j}}$ based on policy $\pi_{\theta_j}(a_{i_{K_j}}|s_i)$
    Receive reward r_i
  end for
  for $i = 1, ..., B$ **do**
    $\Delta\theta_j \leftarrow \Delta\theta_j + \alpha_j(\nabla_{\theta_j} log\pi_{\theta_j}(a_{i_{K_j}}|s_i)(r_i - V^{\pi_{\theta_j}}_{\theta_{v_j}}(s_i)) + \beta_j \nabla_{\theta_j} H(\pi_{\theta_j}(\cdot|s_i)))$
    $\Delta\theta_{v_j} \leftarrow \Delta\theta_{v_j} - \alpha_{v_j} \nabla_{\theta_{v_j}} (r_i - V^{\pi_{\theta_j}}_{\theta_{v_j}}(s_i))^2$
  end for
  $\theta_j \leftarrow \theta_j + \Delta\theta_j$, $\theta_{v_j} \leftarrow \theta_{v_j} + \Delta\theta_{v_j}$
end for

strength of the entropy regularization term for the j-th sub-model. Algorithm 1 shows the pseudo-code for the sub-model training algorithm in the first stage.

B. Learning a Sub-model Selection Policy

1) Second-Stage RL Formulation: Once the first training stage is completed, we can obtain several well-trained sub-models with different K settings. Then, we need to train a single model in the second stage to learn a policy π that decides how many critical entries are required (i.e., which sub-model should be selected) for each given TM to ensure a near-optimal network performance with the smallest number of critical entries. Since each well-trained sub-model p_j can be taken as a module that identifies the best K_j critical entries, the single model can focus on how many critical entries K should be selected and choose the corresponding sub-model without worrying about potential performance degradations caused by inappropriate critical entry selection from sub-models.

Input: An input instance s for the second stage is the same as the first stage, which is a traffic matrix TM.

Action Space: Given an instance s, the single model should decide how many critical entries K to be selected. In other words, it determines which sub-model should be chosen. Since there are a total of m sub-models, the action space should be $\{1, 2, ..., m\}$ that allows the single model to select one of the sub-models for each instance s.

Reward: In the second stage, the single model is responsible for selecting a K value (i.e., choosing one of the sub-models) for a given instance s. After that, the selected sub-model p_j would be used to identify the best K_j critical entries based on the input TM. Then, the LP optimization problem (9a) can be solved to obtain the optimal traffic split ratios for the critical entries as well as the maximum link utilization U. As mentioned in the previous section, we can compute the performance ratio PR according to Eq. (1). Since our goal is to achieve near-optimal network performance with the lowest routing update overhead, the single model needs to trade off

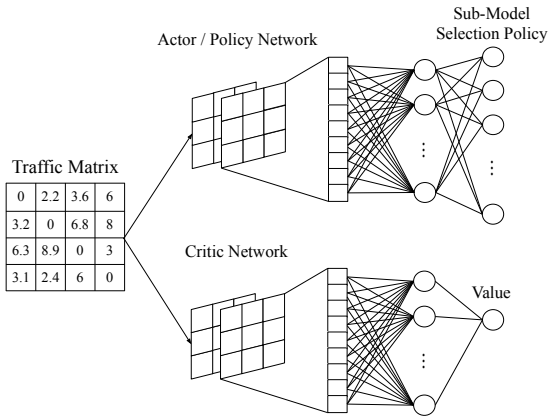


Fig. 3. Training a single model in the second stage to select an appropriate sub-model p_j with K_j critical entries to be identified.

the number of critical entries K and network performance PR. Thus, we design a reward function for the single model as follows:

$$r = \begin{cases} PR^2 & \text{if } PR < PR_{TH} \\ 1 - \lambda * \dfrac{K}{N * (N-1)} & \text{if } PR \geq PR_{TH} \end{cases} \quad (6)$$

where PR_{TH} is a preset performance target, λ is a penalty factor on the number of critical entries, and $\frac{K}{N*(N-1)}$ represents the percentage of total entries that are identified as critical entries. In our evaluation, we set the performance target PR_{TH} to 0.9 to achieve near-optimal performance (i.e., above 90% of optimal performance). For the penalty factor λ, it can be viewed as a knob that adjusts the trade-off between network performance and routing update overhead. We will discuss the impact of network topologies on λ settings in Section VI-B2.

Intuitively, the ideas of our reward function design can be explained as follows. When RL cannot achieve a satisfactory performance (i.e., $PR < PR_{TH}$), the reward signal solely relies on the performance ratio without consideration of the number of critical entries K. Based on the quadratic function of PR, RL would benefit more from better performance and thus focus on selecting an appropriate sub-model to improve network performance. Once the performance target is achieved (i.e., $PR \geq PR_{TH}$), the reward function only depends on the number of critical entries K. If a sub-model with a higher K value is selected, RL would receive a smaller reward regardless of the performance improvements since there is a penalty imposed on the number of critical entries. Therefore, RL should learn to reduce the number of critical entries as much as possible when the performance target can be satisfied.

2) Second-Stage Training Algorithm: For the second training stage, we only need to train a single model. Similar to the sub-models in the first stage, this single model also uses an actor-critic architecture and takes a TM as an input, as shown in Fig. 3. However, the policy network of the single model would output $\pi(a^0|s)$ as a probability distribution over all candidate sub-models that may be selected. Here, the stochastic policy $\pi(a^0|s)$ is parameterized by θ. We can sample an action a^0 from the action space $\{1, 2, ..., m\}$ to choose one of the sub-models p_j for generating the corresponding K_j critical entries. Similar to the first stage, the critic network parameter

Algorithm 2 Second-Stage Training Algorithm for Single Model

Initialize θ and θ_v
for each iteration **do**
  $\Delta\theta \leftarrow 0$, $\Delta\theta_v \leftarrow 0$
  $\{s_i\} \leftarrow$ Sample a batch of instances with size B
  for $i = 1, ..., B$ **do**
    Sample an action a_i^0 according to policy $\pi_\theta(a_i^0|s_i)$
    Obtain $K_{a_i^0}$ critical entries from sub-model $p_{a_i^0}$
    Receive reward r_i
  end for
  for $i = 1, ..., B$ **do**
    $\Delta\theta \leftarrow \Delta\theta + \alpha(\nabla_\theta log\pi_\theta(a_i^0|s_i)(r_i - V_{\theta_v}^{\pi_\theta}(s_i)) + \beta\nabla_\theta H(\pi_\theta(\cdot|s_i)))$
    $\Delta\theta_v \leftarrow \Delta\theta_v - \alpha_v\nabla_{\theta_v}(r_i - V_{\theta_v}^{\pi_\theta}(s_i))^2$
  end for
  $\theta \leftarrow \theta + \Delta\theta$, $\theta_v \leftarrow \theta_v + \Delta\theta_v$
end for

θ_v of the single model is updated as follows:

$$\theta_v \leftarrow \theta_v - \alpha_v \sum_s \nabla_{\theta_v}(r - V_{\theta_v}^{\pi_\theta}(s))^2, \tag{7}$$

where α_v is the learning rate for the critic network. The policy network parameter θ of the single model is updated as follows:

$$\theta \leftarrow \theta + \alpha \sum_s \nabla_\theta log\pi_\theta(a^0|s)(r - V_{\theta_v}^{\pi_\theta}(s)) + \beta\nabla_\theta H(\pi_\theta(\cdot|s)), \tag{8}$$

where α is the learning rate for the policy network, and β is the entropy factor. Algorithm 2 shows the pseudo-code for the single model training algorithm in the second stage.

V. Traffic Split Ratio Optimization

In this section, we describe how to formulate and solve the traffic split ratio optimization problem in FlexEntry. The notations used in this section are listed in Table I.

A. Rerouting Optimization Problem

As described in Section III, traffic is either distributed evenly among the default ECMP next hops or split unevenly among admissible next hops according to the critical entries. Given a network $G(V, E)$ with a TM and selected critical entries τ_K, our objective is to obtain the weighted split ratios $\{\sigma_{i,j}^d\}$ for the selected $\tau_i^d \in \tau_K$, such that the maximum link utilization U is minimized and the routing is loop-free. To achieve this objective, we first obtain the optimal destination-based traffic allocation $\{y_{i,j}^d\}$, where $y_{i,j}^d$ stands for the traffic destined to d routed on link $\langle i, j\rangle$. Then, we can derive $\{\sigma_{i,j}^d\}$ from $\{y_{i,j}^d\}$. We formulate the destination-based rerouting problem as an optimization problem as follows:

$$minimize \quad U \tag{9a}$$

subject to

$$\sum_{d\in V} y_{i,j}^d = l_{i,j} \qquad i, j : \langle i, j\rangle \in E \tag{9b}$$

TABLE I
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 $\langle i, j\rangle$ ($\langle i, j\rangle \in E$)
$t^{s,d}$	the traffic demand originated from s destined to d ($s, d \in V$, $s \neq d$)
τ_i^d	the traffic destined to d at node i ($i, d \in V$, $i \neq d$, $\|\{\tau_i^d\}\| = N * (N - 1)$)
τ_K	a combination of K selected τ_i^d ($\|\tau_K\| = K$), e.g., τ_1^6, τ_3^{10}, and τ_5^{10} in Fig. 1(b)
$\tau_{N*(N-1)-K}$	the set of remaining τ_i^d ($\|\tau_{N*(N-1)-K}\| = N * (N - 1) - K$)
ENH_i^d	the set of ECMP next hops for destination d at node i ($i, d \in V$), e.g., $\text{ENH}_1^{10} = \{3\}$, $\text{ENH}_3^{10} = \{6\}$ in Fig. 1(b)
$y_{i,j}^d$	the traffic destined to d routed on link $\langle i, j\rangle$ ($d \in V, \langle i, j\rangle \in E$)
$\sigma_{i,j}^d$	the split ratio at node i to node j for the traffic destined to node d ($d \in V, \langle i, j\rangle \in E$), e.g., $\sigma_{5,8}^{10} = 66.6\%$ in Fig. 1(b)

$$l_{i,j} \leq c_{i,j} \cdot U \qquad i, j : \langle i, j\rangle \in E \tag{9c}$$

$$\sum_{k:\langle k,i\rangle\in E} y_{k,i}^d - \sum_{k:\langle i,k\rangle\in E} y_{i,k}^d = -t^{i,d} \quad i, d : \tau_i^d \in \tau_K \tag{9d}$$

$$y_{i,k}^d = \begin{cases} \dfrac{\sum_{n:\langle n,i\rangle\in E} y_{n,i}^d + t^{i,d}}{|\text{ENH}_i^d|} & \text{if } k \in \text{ENH}_i^d \\ 0 & \text{otherwise} \end{cases} \tag{9e}$$

$$i, d : \tau_i^d \in \tau_{N*(N-1)-K},\ k : \langle i, k\rangle \in E$$

$$\sum_{k:\langle k,d\rangle\in E} y_{k,d}^d - \sum_{k:\langle d,k\rangle\in E} y_{d,k}^d = \sum_{s\in V, s\neq d} t^{s,d} \quad d \in V \tag{9f}$$

$$y_{i,j}^d \geq 0 \qquad d \in V,\ i, j : \langle i, j\rangle \in E \tag{9g}$$

(9c) is the link capacity utilization constraint. (9d), (9e), and (9f) are the flow conservation constraints for the selected τ_i^d, for the remaining τ_i^d, and at destinations, respectively. By solving problem (9) using LP solvers (e.g., Gurobi [44]), we can obtain the optimal destination-based traffic allocation $\{y_{i,j}^d\}$. In case there exist forwarding loops, we can use the techniques in [14] to eliminate the loops. Then, $\{\sigma_{i,j}^d\}$ can be derived according to the following equation:

$$\sigma_{i,j}^d = \frac{y_{i,j}^d}{\sum_{k:\langle i,k\rangle\in E} y_{i,k}^d} \quad i, d : \tau_i^d \in \tau_K,\ j : \langle i, j\rangle \in E. \tag{10}$$

Note that traffic is distributed evenly among ECMP next hops for the remaining $\tau_i^d \in \tau_{N*(N-1)-K}$.

B. Traffic Splitting

By leveraging the existing IP router's forwarding table lookup architecture, we can easily expand it to accommodate the function of forwarding traffic to each destination node with different split ratios at the output ports. IP lookup usually uses a RAM-based proprietary data structure to perform the longest prefix matching. When an incoming packet's destination IP address matches with the longest prefix, the result is a pointer pointing to an entry of another table storing next hop information (let us call the table NHIT). Each entry of the NHIT can,

for instance, store the next hop's IP address and an output port number. To facilitate our proposed TE solution, the NHIT can be slightly modified to a so-called Traffic Split Ratio Table (TSRT). It has N entries, each corresponding to a destination node, and each entry has a flag and H split ratios $\sigma_{i,j}^{d}$ (H = the number of output ports, e.g., 64). When the flag is set, up to H split ratios provided by the centralized controller are used to split traffic destined to the corresponding destination node at the output ports of the router; otherwise, traffic is evenly distributed among ECMP next hops ENH_i^d. In practice, packets belonging to a TCP (or UDP) session follow a single path to avoid packet misordering. An approximation to the traffic splitting is to hash the 5-tuple packet header fields and then allocate TCP (or UDP) flows to one of the output ports based on the hash results and split ratios (refer to RFC 2992 [45] and the standard hashing technique [46]).

VI. Evaluation

In this section, we conduct extensive simulations using six different real-world network topologies to evaluate the performance of FlexEntry and demonstrate its effectiveness in mitigating routing update overhead[5].

A. Evaluation Setup

1) Simulation Environment: In our evaluation, the actor-critic architecture for 2-stage RL is implemented using TensorFlow [47]. All the training tasks are conducted in a high-performance computing cluster to accelerate the training process. We spawn multiple actor agents (e.g., 20 agents) in parallel to experience different subsets of the training set and aggregate their (state, action, reward) tuples to a central RL agent for performing gradient updates. To train the sub-models as well as the single model, we assign a 2.9 GHz CPU core to each RL agent with a total of 32 GB memory allocated. It is worth mentioning that we train multiple sub-models with different K settings in parallel to speed up the training process during the first RL training stage. In practice, the number of CPU cores and memory allocated for RL training can be adjusted based on actual hardware specifications.

Once the training is done, we conduct all simulation tests on a Linux server with a 4-core Intel 3.4 GHz CPU and 16 GB memory. The server is running an Ubuntu 16.04.2 LTS system with Gurobi optimizer v9.1.1 [44] installed to solve LP for FlexEntry and other baseline methods mentioned in Section VI-A4. To simulate network environments, we use Python 3.8 and the NetworkX library [48] to construct networks based on the six real-world network topologies listed in Table II. Given the real TMs and synthetic TMs described in Section VI-A2, traffic flows with different demand volumes are fed into the network accordingly to simulate various traffic scenarios. Then, we can implement FlexEntry and the baselines to control traffic distributions and evaluate the load balancing performance. It is worth mentioning that the critical entries installed in the previous time period (i.e., for the last TM) would be removed when performing routing updates for the current traffic scenario, as we discussed in Section III-B. Given that FlexEntry performs routing updates when traffic changes (e.g., a new TM is measured), each entry update would last for the same time as the TM measurement interval in each network.

[5]The source codes of FlexEntry and the datasets used in our evaluation are publicly available at GitHub (https://github.com/yanghu-bit/FlexEntry).

TABLE II
Network Topologies Used in Evaluation

Topology	Nodes	Directed Links
Abilene	12	30
Nobel-Germany	17	52
EBONE (Europe)	23	76
Sprintlink (US)	44	166
Tiscali (Europe)	49	172
Germany50	50	176

TABLE III
Parameter Settings of Synthesized TMs

Parameters	Dynamic TMs	Stable TMs
Hourly Peak-to-Mean Ratio	1.5	1.05
Daily Peak-to-Mean Ratio	5	1.1
Hourly Spatial Variance	1	1
Daily Spatial Variance	3	1.5

2) Dataset: In our evaluation, we use six real-world network topologies, including the Abilene network, two networks (i.e., Nobel-Germany and Germany50) from SNDlib [49], and three service provider networks (i.e., EBONE, Sprintlink, and Tiscali) collected by Rocketfuel [50]. The numbers of nodes and directed links of all six topologies are shown in Table II.

For the Abilene network, the topology information (such as link connectivity, costs, and capacities) and measured TMs are available at [51]. Since Abilene TMs are measured every 5 minutes, there are a total of 288 TMs per day. To evaluate the performance of FlexEntry, we choose the total 2016 TMs in the first week (starting from Mar. 1st, 2004) as our training set, and then test our scheme in the following week (starting from Mar. 8th, 2004). For the Nobel-Germany and Germany50 networks, both the link capacities and costs are not provided. We refer to [31], [32] to set the link capacities and link costs. Specifically, all link costs are set to 10 and the capacity of each link is configured based on the degrees of the two directly connected nodes. If at least one of the directly connected nodes has a degree larger than three, the link capacity is set to 10 Gbps; otherwise, the link capacity would be 5 Gbps. Since there are limited real TMs measured in one day for each of these two networks [52], we take the first 85% of total TMs to train FlexEntry and use the remaining 15% TMs for evaluation.

For the 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 of Cisco routers. In other words, the link costs are inversely proportional to the link capacities. This approach is commonly adopted in literature [14], [53]. Given that there are no measured TMs available for the three service provider networks from Rocketfuel, we synthesize a series of spatiotemporal TMs for each of the three networks using the Modulated Gravity Model (MGM) [54], [55]. On the one hand, MGM can effectively construct spatial properties for synthesized TMs based on gravity-model-like constraints.

On the other hand, MGM utilizes sinusoids to reflect the cyclical nature of TMs and generates TM sequences with temporal correlations. Thus, MGM is capable of emulating the characteristics of real TMs. In our simulation, we combine daily and hourly traffic patterns to synthesize TMs. To evaluate FlexEntry in different traffic scenarios, we tune the MGM parameters to adjust traffic variations in the generated TM sequences. The detailed parameter settings are shown in Table III. Note that we also introduce an exponential model [55] to generate traffic spikes in dynamic TMs. For each of the three networks, we synthesize 50 dynamic TMs and 50 stable TMs as the training set following the parameter settings in Table III. To better evaluate the generalization of FlexEntry, we use a different set of parameters to generate a total of 100 synthetic test TMs in a similar manner (e.g., slightly increase the peak-to-mean ratios and use different random seeds).

3) RL Implementation: For the first stage, the policy neural network of each sub-model consists of three layers, where 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, and the activation function used for the previous two layers is Leaky Relu [56]. The last layer is a fully connected linear layer (without activation function) with $N * (N-1)$ neurons corresponding to all possible actions (i.e., router-destination pairs). A softmax function is applied upon the output of the final layer to generate the probabilities for all available actions. The critic network of each sub-model is similar to the policy network except that the last layer is a fully connected linear layer with only one neuron corresponding to the baseline $b(s)$. For each sub-model p_j, the learning rates α_j and α_{v_j} are initially configured as 0.0001 with a decay rate of 0.96 every 500 iterations. Additionally, the entropy factor β_j is set to 0.1.

For the single model in the second stage, the policy neural network is almost the same as sub-models except for the last layer. With an action space of $\{1, 2, ..., m\}$, the last layer of the single model is a fully connected linear layer (without activation function) with m neurons corresponding to all candidate sub-models. The critic network and hyperparameter settings in the second stage are the same as in the first stage. It is worth mentioning that we reach the above hyperparameter settings through a grid search procedure, as later shown in Section VI-B5. Additionally, the performance target PR_{TH} in the reward function Eq. (6) is configured to 0.9 with an objective to achieve at least 90% of optimal performance. As for the penalty factor settings, λ is configured with consideration of network topology effects as later shown in Section VI-B2. Except for λ, we fixed all these hyperparameters throughout our simulations. The simulation results show that FlexEntry works well on different network topologies with a single set of fixed hyperparameters.

4) Baselines: For comparison, we evaluate six different destination-based TE solutions as follows:

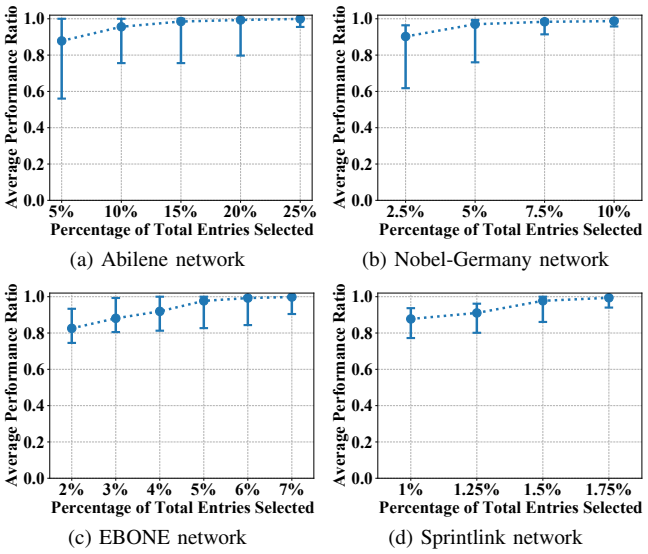
1) **FlexEntry**: leverages a combination of 2-stage RL and LP to control traffic distributions with a flexible number K of critical entries selected for each TM. This is our proposed approach to achieve near-optimal performance and low routing update overhead.
2) **SmartEntry** [57]: exploits RL to select a fixed number of critical entries (i.e., 10% of total entries) and redistribute a small portion of total traffic accordingly with LP to improve network performance.
3) **ECMP** [18]: distributes traffic evenly among available next hops along the shortest paths. The link cost setting for each network was discussed in Section VI-A2.
4) **Weighted ECMP (W-ECMP)** [4]: extends ECMP to allow weighted traffic splitting among available next hops along the shortest paths. The corresponding optimal weighted split ratios are obtained by the method proposed in [4].
5) **Top-K**: selects the top K destination-based forwarding entries that forward the most traffic under ECMP routing, and then adopts the LP formulation in (9) to optimize the traffic split ratios for these entries. This heuristic is designed based on the assumption that the entries forwarding larger traffic volumes would have a dominant impact on network performance. Note that the value of K is the same as FlexEntry.
6) **Link Cost Optimization (LCO)** [21]: uses a local search heuristic to find good link cost settings that achieve promising load balancing performance with ECMP routing, which is implemented by REPETITA [58]. Unlike the static link costs used in the above-mentioned baselines, LCO changes link costs adaptively for each TM.

B. Simulation Results

1) The Number of Critical Entries: The goal of FlexEntry is to achieve near-optimal network performance (i.e., $PR \geq PR_{TH}$) while effectively mitigating routing update overhead. Thus, we need to train multiple candidate sub-models with different K settings that provide more flexibility for the single model to adapt to different traffic scenarios. Specifically, we need to ensure that there exists at least one sub-model whose worst-case performance is above $PR_{TH} = 0.9$, such that we can guarantee near-optimal performance for all TMs. Thus, we conduct a series of simulations with different numbers of critical entries.

Fig. 4 shows the average performance ratio achieved by each sub-model with an increasing percentage of total entries selected as critical entries in the first four networks. Each data point in Fig. 4 represents a sub-model with a distinct K value. For example, in a network with N nodes, 5% of total entries selected means that $K = 5\% * N * (N-1)$. For each sub-model, we also plot an error bar that spans from the lowest to the highest performance ratio. From Fig. 4, we can see that the performance ratio is improving as the number of critical entries K increases in all four networks. This is because LP can reroute more traffic to improve network performance as K grows, but it would also incur higher routing update overhead since FlexEntry needs to update more critical entries.

To determine an appropriate set of candidate sub-models for each network, it is important for the sub-model with the lowest K value to achieve $PR_{TH} = 0.9$ for at least one TM, while the sub-model with the highest K value should achieve above



(a) Abilene network (b) Nobel-Germany network

(c) EBONE network (d) Sprintlink network

Fig. 4. Evolution of the average performance ratio of FlexEntry with an increasing number of critical entries in the first four networks.

90% of optimal performance for all TMs. The reason is that we need to provide flexible solutions to accommodate various TMs. For instance, as illustrated in Fig. 4(a), we train 5 sub-models with K ranging from 5% to 25% of total entries as the candidates in the Abilene network. For some of the TMs, it is good enough to select 5% of total entries to achieve the performance target with low routing update overhead. Meanwhile, we have to choose 25% of total entries for some TMs to maintain promising network performance. Based on the above-mentioned selection criteria, we can determine the lower bound and upper bound for the K value.

Another important aspect is to define the number of sub-models m and their corresponding K values with consideration of the training complexity and the flexibility under different traffic scenarios. If we use fewer sub-models, it would require less training time and resource consumption. However, FlexEntry might not be able to adapt to different traffic scenarios since the action space in the second stage RL is limited. Suppose we only have 3 sub-models for the Abilene network with K = 5%, 15%, and 25% of total entries, respectively. Then, it is possible that updating 5% of total entries cannot guarantee near-optimal performance for a certain TM, while 15% of entry updates would result in unnecessary routing update overhead. In contrast, if we use more sub-models, FlexEntry would be more effective in mitigating routing update overhead with a wider range of candidate K values. However, it may lead to higher training complexity with a larger solution space to explore. Throughout our simulation, we found that 4-6 sub-models would be sufficient to achieve a good trade-off between flexibility and overhead. Thus, we empirically choose several candidate sub-models with different K values for each network, as shown in Table IV.

2) Topology Effects and Penalty Factor Settings: From Fig. 4 and Table IV, one interesting observation is that less percentage of total entries (i.e., $\frac{K}{N(N-1)}$) is required to achieve promising network performance as the topology size increases. In Fig. 4(a), we can see that 25% of total entries should be installed in the Abilene network to satisfy the performance

TABLE IV
PARAMETER SETTINGS FOR DIFFERENT NETWORKS

Topology	λ	Critical Entries Percentage of Sub-Models
Abilene	0.05	5%, 10%, 15%, 20%, 25%
Nobel-Germany	0.8	2.5%, 5%, 7.5%, 10%
EBONE	1	2%, 3%, 4%, 5%, 6%, 7%
Sprintlink	5	1%, 1.25%, 1.5%, 1.75%
Tiscali	6	0.5%, 0.75%, 1%, 1.25%
Germany50	8	0.25%, 0.5%, 0.75%, 1%

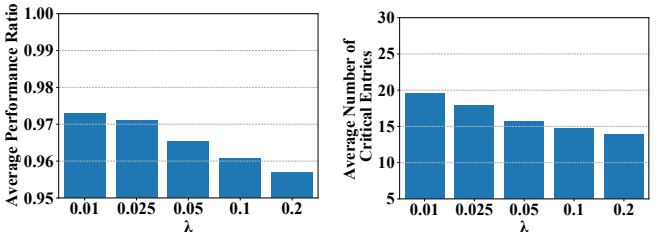
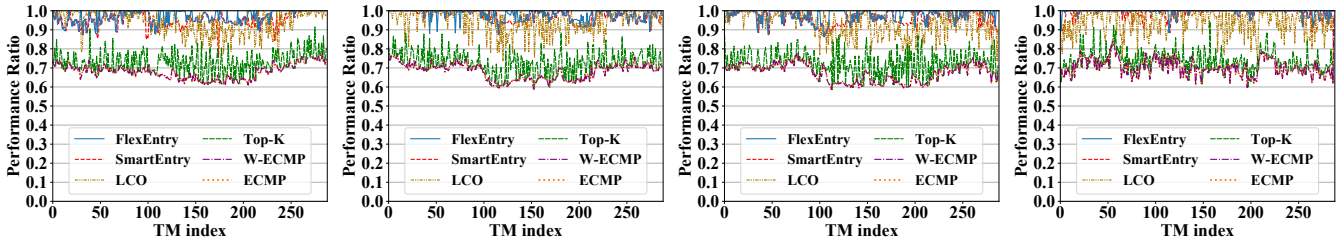


Fig. 5. Comparison of average performance ratio and average number of critical entries with different penalty factor λ settings in the Abilene network.

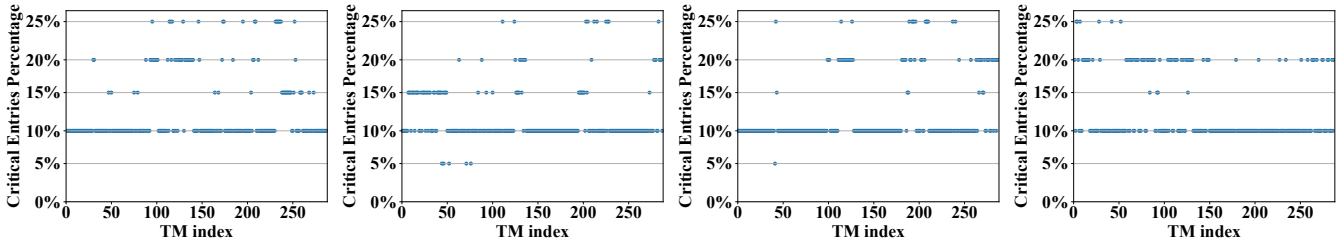
target (i.e., $PR \geq PR_{TH}$) for all TMs, while it only requires 7% of total entries for FlexEntry to achieve near-optimal performance in all cases of the EBONE network, as shown in Fig. 4(c). Such observation holds for all six real-world networks. One possible reason is that each entry would control more traffic/flows in larger networks to improve network performance more effectively. Given that a critical entry τ_i^d is installed in router i with a specified destination d, it is responsible for controlling all traffic destined to d that aggregates at router i. With consideration of all possible source nodes, a carefully selected τ_i^d could control more source-destination pairs in larger networks and thus become more effective in improving network performance. As a result, FlexEntry can achieve the performance target with less percentage of total entries selected as critical entries in larger networks.

Due to the topology effects on the number of critical entries, we have different sub-model configurations for the six real-world network topologies. For example, as shown in Fig. 4, the K interval for the sub-models of the Abilene network is set to 5%, while the sub-models in the Sprintlink network have a K interval of 0.25%. When tuning the penalty factor λ in the reward function Eq. (6) to achieve good performance and low overhead, we need to consider the above-mentioned topology effects and sub-model configurations. Moreover, λ is designed as a knob to balance network performance and routing update overhead. To investigate the influence of penalty factor λ on FlexEntry, we run several experiments in the Abilene network with 5 different λ settings (i.e., 0.01, 0.025, 0.05, 0.1, and 0.2). As depicted in Fig. 5, we can find that both the average performance ratio and average critical entry number would gradually decrease with an increasing λ in the Abilene network. This is reasonable since λ controls the strength of penalty on critical entries percentage $\frac{K}{N(N-1)}$. Given a high λ value, RL would prioritize the reduction of critical entry number over the improvement of network performance; otherwise, RL would be more tolerant to a high K value such that it can achieve better performance by rerouting more traffic. Since our objective is to achieve the performance target with consideration of mitigating routing update overhead, we believe that FlexEntry should provide near-optimal performance for at least 95% of



(a) Day 2 (b) Day 3 (c) Day 5 (d) Day 6

Fig. 6. Comparison of performance ratio PR among different destination-based TE solutions with the Abilene test TMs from days 2, 3, 5, and 6 in week 2. The higher PR, the better performance.



(a) Day 2 (b) Day 3 (c) Day 5 (d) Day 6

Fig. 7. The number of critical entries K selected by FlexEntry for each Abilene test TM from days 2, 3, 5, and 6 in week 2. A smaller K value indicates lower routing update overhead. For clarity, we use the percentage of total entries selected as critical entries to represent K values.

traffic scenarios with less critical entries selected under the λ settings. Therefore, we configure an appropriate penalty factor λ for each topology in Table IV based on the above-mentioned criterion to effectively trade off PR and K. In the following experiments, we fix the parameter settings as shown in Table IV.

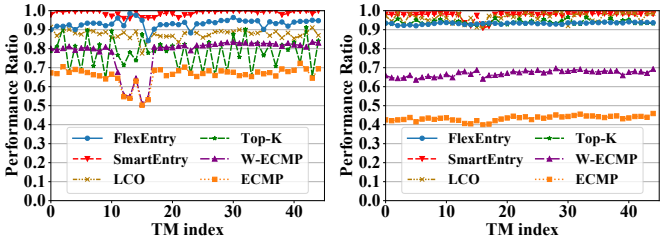
3) Performance Comparison: To demonstrate the performance of FlexEntry, we compute the performance ratio of each baseline method according to Eq. (1) for comparison. Fig. 6 shows the performance ratio achieved by each TE solution on week 2's test TMs in the Abilene network. Fig. 7 provides the number of critical entries selected by FlexEntry for each test TM in the Abilene network.

Since there is only one shortest path available for each flow in the Abilene network, W-ECMP cannot utilize multiple shortest paths with weighted split ratios to improve network performance. In Fig. 6, W-ECMP achieves the same performance as ECMP, which is far from optimal performance. By optimizing link costs for each TM, LCO can far outperform ECMP since ECMP uses static pre-defined link costs that cannot adapt to traffic changes. However, the performance of LCO is not stable and we can find severe performance degradations in many traffic scenarios. Due to the restriction of the shortest paths and equal split rules, the performance of LCO could be limited compared to the optimal solution.

For the other schemes, we can see that FlexEntry achieves the performance target in most cases with better worst-case performance compared to SmartEntry. For example, as depicted in Fig. 6(a), we can find that the performance ratio of SmartEntry is only 75.6% for the 175th TM on day 2. This is because SmartEntry selects a fixed number of critical entries (i.e., 10% of total entries) without considering topology effects and traffic dynamics. As shown in Section VI-B1, near-optimal performance cannot be guaranteed for all Abilene TMs when selecting 10% of total entries. Thus, SmartEntry cannot adapt to different traffic scenarios and suffers from performance degradation. In contrast, FlexEntry achieves optimal performance for the same TM with a performance improvement of 24.4% over SmartEntry. As shown in Fig. 7(a), FlexEntry avoids performance degradation by selecting more critical entries (i.e., 25% of total entries) for the 175th Abilene TM in day 2. Similarly, it could be too much for SmartEntry to select 10% of total entries in certain traffic scenarios. As shown in Figs. 6(b) and 7(b), even though SmartEntry achieves better performance in the 51st TM on day 3 compared to FlexEntry, FlexEntry only needs to select 5% of total entries to achieve near-optimal performance with lower routing update overhead. Therefore, FlexEntry is able to choose an appropriate sub-model for each test TM as shown in Fig. 7 to adapt to traffic dynamics.

Another interesting finding is that Top-K can only provide a marginal performance improvement over ECMP. In some traffic scenarios, Top-K achieves the same performance as ECMP after updating the top K entries that forward the most traffic, while ECMP does not require any routing updates. In other words, such entry updates from Top-K are ineffective since the network congestion cannot be alleviated. This is because a simple rule-based heuristic cannot adapt to the changes in TMs and network dynamics. On the contrary, with the exact same number of entry updates, FlexEntry is able to achieve near-optimal performance with intelligently selected critical entries, which demonstrates the effectiveness of our 2-stage RL design.

For the Nobel-Germany and Germany50 networks, Fig. 8 illustrates the performance comparison among different destination-based TE solutions. Compared to the Abilene network, there are fewer TMs in the training set of these two networks (i.e., the first 85% TMs in a single day). Therefore, it could be more difficult for RL to capture the complete daily traffic patterns with limited data. However, as shown in Fig.



(a) Nobel-Germany network (b) Germany50 network

Fig. 8. Performance ratio comparison based on the test TMs of the Nobel-Germany and Germany50 networks.

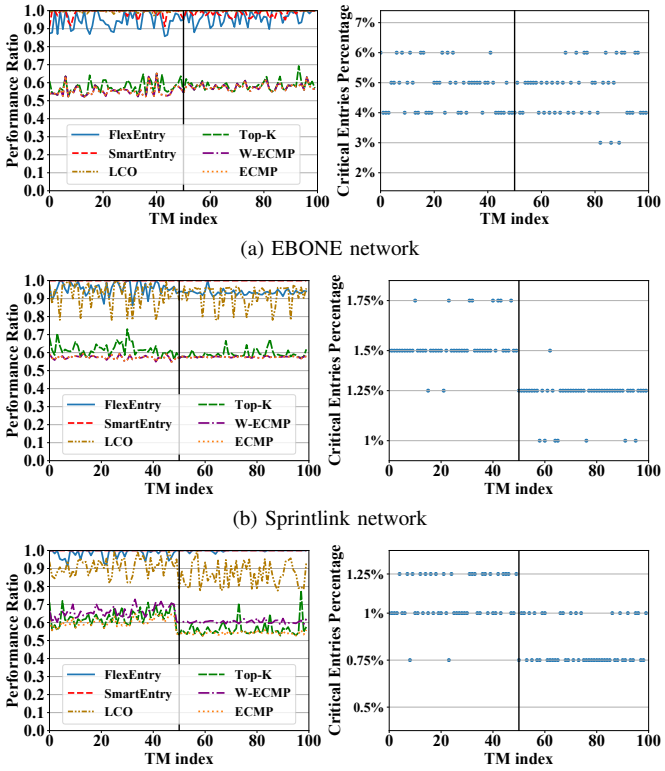
TABLE V
COMPARISON OF AVERAGE CRITICAL ENTRY NUMBERS

Topology	FlexEntry	SmartEntry
Nobel-Germany	8.2 (3% of total)	27 (10% of total)
Germany50	18 (0.7% of total)	245 (10% of total)

8, FlexEntry achieves the performance target $PR_{TH} = 0.9$ in most cases in the two networks except for two TMs in the Nobel-Germany network. Moreover, the lowest PR achieved by FlexEntry is 84.2%, which is not far from the performance target.

For the baseline methods, W-ECMP can outperform ECMP with optimized weighted traffic splitting among ECMP next hops. However, W-ECMP is still constrained by the shortest paths with 14.9% and 27.2% average performance degradation compared to FlexEntry in the Nobel-Germany and Germany50 networks, respectively. For LCO and Top-K, there exist performance fluctuations in the Nobel-Germany network, which is similar to our previous observation in the Abilene network. However, both LCO and Top-K can achieve surprisingly good performance in the Germany50 network and even slightly outperform FlexEntry. One possible reason is that FlexEntry only targets 90% of optimal performance, while LCO and Top-K benefit from this specific network topology and TM distribution by rerouting a large amount of traffic. For SmartEntry, it can achieve close-to-optimal network performance with 10% of total entries selected as critical entries. However, as shown in Table V, FlexEntry only selects 3% and 0.7% of total entries on average for the Nobel-Germany and Germany50 networks, respectively. In other words, FlexEntry achieves near-optimal performance while saving up to 99.3% entry updates on average. This is because FlexEntry aims at achieving a preset performance target with consideration of mitigating routing update overhead, while SmartEntry solely focuses on maximizing network performance. As a result, FlexEntry can generalize to unseen TMs while ensuring near-optimal performance with low routing update overhead.

4) Generalization Over Different Traffic Variations: As described in Section VI-A2, we synthesize a series of TMs for the EBONE, Sprintlink, and Tiscali networks. In the test set of each network, the first 50 TMs represent dynamic traffic scenarios with large traffic variations, and the remaining 50 TMs are relatively stable. Fig. 9 demonstrates the comparison results for each test TM of the three networks. Compared to the stable TMs, it is obvious that there are more frequent performance fluctuations for all TE solutions in the dynamic traffic scenarios (i.e., the first 50 TMs), where FlexEntry



(a) EBONE network

(b) Sprintlink network

(c) Tiscali network

Fig. 9. Comparison of performance ratio and critical entries percentage in the EBONE, Sprintlink, and Tiscali networks with different traffic variations.

tends to select more critical entries for these dynamic TMs to maintain good network performance. For the second half of TMs with small traffic variations, the performance is relatively stable and FlexEntry tends to select less critical entries to mitigate routing update overhead. As shown in Fig. 9, FlexEntry performs consistently well in both dynamic and stable traffic scenarios with an average performance ratio of 94.6% - 99.1% in the three networks. Meanwhile, FlexEntry flexibly selects different numbers of critical entries based on network conditions and effectively reduces the average entry updates by up to 99% in the three networks. For instance, FlexEntry can achieve the performance target for all Tiscali test TMs with only 1% of total entries selected on average. As for the baseline methods, the performance ratios of Top-K, W-ECMP, and ECMP are not promising while SmartEntry introduces more entry updates compared to FlexEntry. Even though LCO achieves relatively good performance compared to other traditional baseline methods, it also requires updating all link costs and waiting for OSPF reconvergence, which may lead to network disturbance and service disruption during the routing updates. Overall, FlexEntry can generalize well to different traffic variations with promising network performance as well as low routing update overhead.

5) Hyperparameters: To investigate the impact of different hyperparameter settings on FlexEntry's performance, we perform a grid search on the learning rate α and the entropy factor β. Table VI lists the average performance ratio of FlexEntry with different α and β settings in the Abilene network. Since the results for other network topologies are similar, we only present the results for the Abilene network. For each set of

TABLE VI
AVERAGE PERFORMANCE RATIO WITH DIFFERENT HYPERPARAMETERS

	$\beta = 0.01$	$\beta = \mathbf{0.1}$	$\beta = 1$
$\alpha = 0.001$ (with decay)	0.932	0.939	0.927
$\alpha = \mathbf{0.0001}$ **(with decay)**	0.948	**0.965**	0.960
$\alpha = 0.00001$ (with decay)	0.946	0.953	0.952

TABLE VII
OFFLINE TRAINING TIME FOR DIFFERENT NETWORK TOPOLOGIES

Topology	First Stage (hours)	Second Stage (hours)
Abilene	3.5	1.5
Nobel-Germany	4.5	2.5
EBONE	6	4.5
Sprintlink	13.5	7
Tiscali	17	8
Germany50	18	9

hyperparameters, we train a 2-stage RL model with 10000 iterations and evaluate the well-trained RL model in the entire test set of the Abilene network. Note that we use the same hyperparameters for both the sub-models and the single model since they have similar neural network architectures.

The results in Table VI show that FlexEntry achieves the best average performance ratio with $\alpha = 0.0001$ and $\beta = 0.1$, which are the hyperparameter settings we adopted in our simulation. If the initial learning rate α is too large (e.g., $\alpha = 0.001$), the training process would be unstable with a negative impact on RL convergence. On the contrary, a small learning rate (e.g., $\alpha = 0.00001$) may lead to slow convergence with a longer training time. Thus, we believe that $\alpha = 0.0001$ would be a good choice to balance training stability and convergence speed. Regarding the entropy factor β, it determines the trade-off between exploration and exploitation during RL training. A large entropy factor will encourage exploration for long-term benefit but at the risk of making bad decisions. In contrast, a small entropy factor can focus on exploitation and avoid potential bad decisions. However, RL might be trapped in a sub-optimal policy with a small β since it cannot discover potential better decisions. Thus, we consider the above-mentioned trade-offs and empirically reach a good set of hyperparameters for FlexEntry, which is demonstrated to work well in different network scenarios.

6) Time Complexity and Routing Update Overhead: Table VII lists the training time of FlexEntry for the first and second training stages in the six networks. From Table VII, we can observe that it is usually faster to train a single model in the second stage compared to the sub-model training in the first stage. This is because the action/solution space of the single model is much smaller than that of sub-models, given that each sub-model needs to learn a combination selection policy while the single model learns to select one of the sub-models. Besides, as the network size grows, there are more router-destination pairs in the network with larger action/solution space. Thus, the training time is positively correlated with the network topology size. For the smallest Abilene network, only a total of 5 hours are required to train FlexEntry. As for the largest Tiscali and Germany50 networks, we need approximately one day to train FlexEntry. However, it is worth mentioning that all the training costs are incurred offline. To keep FlexEntry up-to-date, the sub-models and single model can also be retrained periodically (e.g., once a week) based on newly collected TMs to absorb various traffic patterns.

TABLE VIII
ONLINE EXECUTION TIME FOR DIFFERENT NETWORK TOPOLOGIES

Topology	RL Inference (ms)		LP Optimization (ms)		Saving
	Avg.	Range	Avg.	Range	
Abilene	4.2	2.8-32.9	12.4	11.2-104.4	64.0%
Nobel-Germany	17.9	3.4-31.9	28.2	26.1-96.3	65.2%
EBONE	19.2	7.6-95.5	44.1	40.4-126.1	83.8%
Sprintlink	80.3	24.5-103.4	155.8	133.2-244.4	91.8%
Tiscali	97.5	92.2-175.7	193.3	161.4-266.9	90.0%
Germany50	114.8	89.6-243.9	259.6	214.6-342.3	86.8%

*Note: "Saving" means the average LP time savings of FlexEntry compared to the optimal destination-based routing LP method for all entries [14].

Another important aspect that makes FlexEntry a practical solution is the low time complexity and routing update overhead during online deployment. For each of the six networks, we list the average online execution time of FlexEntry along with the upper bound and lower bound of the execution time in Table VIII. From the perspective of RL, the inference time to select critical entries would be longer in larger networks. This is because the dimension of input TMs and the size of each sub-model's output layer would become larger if the number of network nodes increases. As for the LP optimization time, the size of the LP problem is also related to the network size. Since the destination-based entries would forward the traffic in a hop-by-hop manner, the degree of network nodes should be considered when measuring the size of the LP problem presented in (9). Suppose the maximum degree in a network $G = (V, E)$ is denoted as $\Delta(G)$. Then, we have at most $O(N^2\Delta(G))$ routing variables to be solved and $O(N^2\Delta(G) + M)$ constraints to be considered in the LP formulation of FlexEntry, where N and M stand for the number of nodes and links, respectively (please refer to Table II). As shown in Table VIII, the execution time for LP optimization would become longer as the network size increases. However, it is also worth mentioning that the optimal traffic split ratio for critical entries can be efficiently solved by LP under different network scenarios. In the largest Germany50 network with 50 nodes and 176 links, it only takes less than 400 ms on average for FlexEntry to identify critical entries and compute the corresponding optimal routing solution. Such low execution time should be credited to RL's critical entry selection since LP only needs to optimize the traffic split ratios for a small portion of total entries. Compared to the optimal destination-based routing LP method for all entries, FlexEntry can achieve 64.0%-91.8% of average LP time savings in the six networks, which would be more critical in large networks as the LP complexity increases. More importantly, FlexEntry only needs to install a few critical entries at some routers with much lower routing update overhead (e.g., updating 0.7% of total entries on average in the Germany50 network). Overall, FlexEntry can achieve good scalability in large networks by performing routing updates in a responsive manner with low time complexity and management overhead.

VII. CONCLUSION AND FUTURE WORK

To achieve near-optimal load balancing performance and mitigate routing update overhead, we proposed FlexEntry, a

destination-based TE solution that learns to intelligently and flexibly identify a few critical entries using RL and then redistributes the selected traffic by solving an LP optimization problem. FlexEntry is trained with a 2-stage RL approach to accommodate various traffic scenarios and realize a good trade-off between network performance and forwarding entry savings. Extensive simulation results on six real-world network topologies show that FlexEntry can effectively reduce entry updates and achieve the performance target in most cases with good generalization over unseen TMs and different traffic variations.

Yet, it would be of interest to discuss several aspects that may help improve the performance and generalization capability of the proposed FlexEntry in future work.

Objectives: The major performance metric considered by FlexEntry in this work is the minimization of the maximum link utilization in the network. As such, the critical entry selection and traffic rerouting procedure are carried out to achieve near-optimal load balancing performance with mitigated routing update overhead. However, it is possible that the resulting routing strategy would lead to higher forwarding delay since it is not considered an objective during RL training and LP solving. On the one hand, we can set a higher load balancing performance target (e.g., $PR_{TH} = 0.95$) and a smaller λ value in the reward function, such that FlexEntry will give more priority to improving load balancing performance over mitigating routing update overhead to prevent potential high forwarding delay. On the other hand, FlexEntry could be re-formulated with an objective to minimize the overall end-to-end delay in the network. This can be done by modifying the objective and constraints of LP formulation as well as the performance metric in RL's reward function. In practice, the objective of FlexEntry can be customized based on actual requirements (e.g., better load balancing, lower forwarding delay, higher throughput).

QoS Requirements: In the foreseeable future, it would be important for TE solutions to take the Quality of Service (QoS) requirements of different applications into consideration. For instance, high priority traffic could have certain requirements on end-to-end delay with low link utilization along the routing paths, while low priority traffic is relatively tolerant to higher latency. Such traffic with different priorities can be distinguished by tagging the packet headers at the sender to indicate the priority level [20], [59]. In addition to the priority queues implemented at the data plane, optimized routing decisions at the control plane are also required for good QoS provisioning. Under the assumption that FlexEntry can operate in legacy routers with slight modifications, we can further extend FlexEntry to satisfy the QoS requirements of different applications using a 2-step optimization approach. To achieve this, the original TM can be decomposed into a high priority TM and a low priority TM for optimization purposes. In general, FlexEntry would install critical destination-based entries for traffic with different priorities, and the critical entries for high priority traffic should take strict precedence over those for low priority traffic. In the first step, FlexEntry only considers the optimal allocation of high priority traffic with a specific objective designed in accordance with the QoS requirements. For example, given the high priority TM, FlexEntry would selectively install some critical destination-based entries to minimize the end-to-end delay. Once the allocation of high priority traffic is determined, the second step is to obtain the routing strategy for the remaining low priority traffic. While FlexEntry can target load balancing performance to allocate low priority traffic in this step, it is necessary to modify the link capacity constraints in the LP formulation since part of the bandwidth is already reserved for high priority traffic. As a result, FlexEntry can install some additional critical entries for low priority traffic to improve network performance with good QoS provisioning for high priority traffic.

Robustness: During the online deployment, a good TE solution should be able to provide robust routing strategies against unexpected link failures and topology changes. In the current design of FlexEntry, we only take TM as an input and assume that the topology remains unchanged. Although FlexEntry can implicitly learn how traffic flows affect each other in the network from the reward signals, it would be essential to incorporate topology information into RL design to enhance the robustness of FlexEntry. One promising solution is to adopt the emerging Graph Neural Network (GNN) techniques [60], [61] that are suitable for processing graph-structured data (e.g., network topology). GNN has strong generalization capability over link failures and topology changes [32], [62]. By leveraging the message passing framework in GNN, each network node can efficiently exchange messages with neighboring nodes to capture both topology and traffic information during the training and inference stages. In our future work, we will investigate possible directions to improve the robustness and generalization capability of FlexEntry with GNN.

REFERENCES

[1] Y. Wang and Z. Wang, "Explicit routing algorithms for internet traffic engineering," in *Proceedings Eight International Conference on Computer Communications and Networks (Cat. No. 99EX370)*. IEEE, 1999, pp. 582–588.

[2] E. D. Osborne and A. Simha, *Traffic engineering with MPLS*. Cisco Press, 2002.

[3] J. Chu and C.-T. Lea, "Optimal link weights for ip-based networks supporting hose-model vpns," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 3, pp. 778–788, 2009.

[4] J. Zhang, K. Xi, L. Zhang, and H. J. Chao, "Optimizing network performance using weighted multipath routing," in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2012, pp. 1–7.

[5] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Dynamic hybrid routing: Achieve load balancing for changing traffic demands," in *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*. IEEE, 2014, pp. 105–110.

[6] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé, "Semi-oblivious traffic engineering: The road not taken," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 157–170.

[7] T. Bates, P. Smith, and G. Huston. CIDR report. [Online]. Available: https://www.cidr-report.org/

[8] S. Q. Zhang, Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, R. Boutaba, and A. Leon-Garcia, "Tcam space-efficient routing in a software defined network," *Computer Networks*, vol. 125, pp. 26–40, 2017.

[9] A. X. Liu, C. R. Meiners, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 2, pp. 490–500, 2009.

[10] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *Distributed Computing and Networking*. Springer, 2013, pp. 439–444.

[11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[12] S. Kotachi, T. Sato, R. Shinkuma, and E. Oki, "Multicast routing model to minimize number of flow entries in software-defined network," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019, pp. 1–6.

[13] R. Biswas and J. Wu, "Traffic engineering to minimize the number of rules in sdn datacenters," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1467–1477, 2021.

[14] J. Zhang, K. Xi, and H. J. Chao, "Load balancing in ip networks using generalized destination-based multipath routing," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 6, pp. 1959–1969, 2015.

[15] J. Moy, "OSPF Version 2," *IETF RFC 2328*, April 1998.

[16] W. Reda, K. Bogdanov, A. Milolidakis, H. Ghasemirahni, M. Chiesa, G. Q. Maguire Jr, and D. Kostić, "Path persistence in the cloud: A study of the effects of inter-region traffic engineering in a large cloud provider's network," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 2, pp. 11–23, 2020.

[17] M. Moradi, Y. Zhang, Z. Morley Mao, and R. Manghirmalani, "Dragon: Scalable, flexible, and efficient traffic engineering in software defined isp networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2744–2756, 2018.

[18] D. Thaler and C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection," *IETF RFC 2991*, November 2000.

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[20] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference*, 2013, pp. 15–26.

[21] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, vol. 2. IEEE, 2000, pp. 519–528.

[22] K. Holmberg and D. Yuan, "Optimization of internet protocol network design and routing," *Networks: An International Journal*, vol. 43, no. 1, pp. 39–53, 2004.

[23] B. Fortz and M. Thorup, "Optimizing ospf/is-is weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 756–767, 2002.

[24] M. Parham, T. Fenz, N. Süss, K.-T. Foerster, and S. Schmid, "Traffic engineering with joint link weight and segment optimization," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 313–327.

[25] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional ip routing protocols," *IEEE communications Magazine*, vol. 40, no. 10, pp. 118–124, 2002.

[26] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. ACM, 2018, pp. 40–45.

[27] 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 *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 2016, pp. 25–33.

[28] 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 Conference on Computer Communications*. IEEE, 2018, pp. 1871–1879.

[29] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "Cfr-rl: Traffic engineering with reinforcement learning in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2249–2259, 2020.

[30] M. Ye, J. Zhang, Z. Guo, and H. J. Chao, "Date: Disturbance-aware traffic engineering with reinforcement learning in software-defined networks," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.

[31] N. Geng, T. Lan, V. Aggarwal, Y. Yang, and M. Xu, "A multi-agent reinforcement learning perspective on distributed traffic engineering," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. IEEE, 2020, pp. 1–11.

[32] M. Ye, J. Zhang, Z. Guo, and H. J. Chao, "Federated traffic engineering with supervised learning in multi-region networks," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–12.

[33] Q. He, A. Moayyedi, G. Dán, G. P. Koudouridis, and P. Tengkvist, "A meta-learning scheme for adaptive short-term network traffic prediction," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2271–2283, 2020.

[34] J. Xiao, Z. Xiao, D. Wang, J. Bai, V. Havyarimana, and F. Zeng, "Short-term traffic volume prediction by ensemble learning in concept drifting environments," *Knowledge-Based Systems*, vol. 164, pp. 213–225, 2019.

[35] M. Pióro and D. Medhi, *Routing, flow, and capacity design in communication and computer networks*. Elsevier, 2004.

[36] 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.

[37] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[38] 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.

[39] H. Xu, Z. Yu, C. Qian, X. Li, and Z. Liu, "Minimizing flow statistics collection cost of sdn using wildcard requests," in *IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[40] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[41] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016, pp. 50–56.

[42] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *arXiv preprint arXiv:1803.08475*, 2018.

[43] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[44] Gurobi Optimization. [Online]. Available: https://www.gurobi.com

[45] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *IETF RFC 2992*, November 2000.

[46] Z. Cao, Z. Wang, and E. Zegura, "Performance of hashing-based schemes for internet load balancing," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 1. IEEE, 2000, pp. 332–341.

[47] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[48] NetworkX. [Online]. Available: https://networkx.org/

[49] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007.

[50] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[51] Yin Zhang's Abilene TM. [Online]. Available: https://www.cs.utexas.edu/~yzhang/research/AbileneTM

[52] SNDlib. [Online]. Available: http://sndlib.zib.de/home.action

[53] M. Kodialam, T. Lakshman, J. B. Orlin, and S. Sengupta, "Oblivious routing of highly variable traffic in service overlays and ip backbones," *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, pp. 459–472, 2008.

[54] P. Tune and M. Roughan, "Spatiotemporal traffic matrix synthesis," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 579–592, 2015.

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

[56] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1. Citeseer, 2013.

[57] J. Zhang, Z. Guo, M. Ye, and H. J. Chao, "Smartentry: Mitigating routing update overhead with reinforcement learning for traffic engineering," in *Proceedings of the Workshop on Network Meets AI & ML*, 2020, pp. 1–7.

[58] S. Gay, P. Schaus, and S. Vissicchio, "Repetita: Repeatable experiments for performance evaluation of traffic-engineering algorithms," *arXiv preprint arXiv:1710.08665*, 2017.
[59] 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*. ACM, 2018, pp. 191–205.
[60] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2018.
[61] P. Veličković *et al.*, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2018.
[62] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Is machine learning ready for traffic engineering optimization?" in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–11.

Minghao Ye received the B.Eng. degree in microelectronic science and engineering from Sun Yat-sen University, Guangzhou, China, the B.Eng. degree (Hons.) in electronic engineering from The Hong Kong Polytechnic University, Hong Kong, in 2017, and the M.S. degree in electrical engineering from New York University, New York, 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, network optimization, software-defined networking, and machine learning for networking.

Yang Hu received the B.S. degree in electronic information engineering from Beijing University of Technology, China, in 2020, and the M.S. degree in computer engineering from New York University, New York, NY, USA, in 2022. He has been with Fortinet, Inc., Sunnyvale, CA, USA, since 2022. His research interests include traffic engineering, network optimization, machine learning, and software-defined networking.

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

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

Zehua Guo (Senior Member, IEEE) received B.S. degree from Northwestern Polytechnical University, Xi'an, China, M.S. degree from Xidian University, Xi'an, China, and Ph.D. degree from Northwestern Polytechnical University. He was a Research Fellow at the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, New York, NY, USA, and a Research Associate at the Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN, USA. His research interests include programmable networks (e.g., software-defined networking, network function virtualization), machine learning, and network security. Dr. Guo is an Associate Editor for IEEE Systems Journal and the EURASIP Journal on Wireless Communications and Networking (Springer), and an Editor for the KSII Transactions on Internet and Information Systems. He is serving as the TPC of several journals and conferences (e.g., Elsevier Computer Communications, AAAI, IWQoS, ICC, ICCCN, ICA3PP). He is a Senior Member of IEEE, China Computer Federation, China Institute of Communications, and Chinese Institute of Electronics, and a Member of ACM.

H. Jonathan Chao (Life 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 at New York University (NYU) from 2004 to 2014. He has been doing research in the areas of machine learning for networking, real-time communications, datacenter networks, high-speed packet scheduling/switching/routing, software-defined networking, network function virtualization, and network security. During 2000-2001, he was the Co-Founder and a 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, NY, USA. He is also the Director of the High-Speed Networking Lab. He has co-authored three networking books, *Broadband Packet Switching Technologies-A Practical Guide to ATM Switches and IP Routers* (New York: Wiley, 2001), *Quality of Service Control in High-Speed Networks* (New York: Wiley, 2001), and *High-Performance Switches and Routers* (New York: Wiley, 2007). He holds 63 patents and has published more than 280 journal and conference papers. He is a fellow of the IEEE and 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 TRANSACTION ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.