SmartEntry: Mitigating Routing Update Overhead with Reinforcement Learning for Traffic Engineering

Junjie Zhang[★], Zehua Guo[†], Minghao Ye[‡], H. Jonathan Chao[‡]
Fortinet, Inc.[★] Beijing Institute of Technology[†] New York University[‡]

ABSTRACT

Traffic Engineering (TE) has been used by Internet service providers to improve their network performance and provide better service quality to users. While flow-based TE is an alternative, destination-based TE is a more readily deployed solution. This is because destination-based forwarding is ubiquitously supported by today's routers. A challenge faced by state-of-the-art destination-based TE solutions is considerable time taken by a centralized controller to update traffic split ratios for each entry of the forwarding table of each router. This could impose a fundamental limitation on how responsively the network can react to dynamic changes of traffic demands. In this paper, we propose SmartEntry, a destination-based routing solution coupled with Reinforcement Learning (RL) to reduce the number of the forwarding entries that need to be updated to respond to dynamic change of traffic demands. SmartEntry forwards majority traffic on Equal-Cost Multi-Path (ECMP) and redistributes a small portion of traffic using our proposed RL algorithm. SmartEntry adopts Linear Programming (LP) to produce reward signals. This RL + LP combined approach turns out to be surprisingly effective. We evaluate SmartEntry by conducting extensive experiments on different network topologies with both real and synthesized traffic. The simulation results show that SmartEntry achieves near-optimal performance with a saving of 90% forwarding entry updates, and generalizes well to unseen traffic matrices.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Networks** → **Wide area networks**; **Network management**.

KEYWORDS

Reinforcement Learning, Traffic Engineering, Routing Update Overhead, Linear Programming

ACM Reference Format:
Junjie Zhang, Zehua Guo, Minghao Ye, H. Jonathan Chao. 2020. SmartEntry: Mitigating Routing Update Overhead with Reinforcement Learning for Traffic Engineering. In *Workshop on Network Meets AI & ML (NetAI '20), August 14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3405671.3405809

The work of Zehua Guo was supported by National Key Research and Development Program of China under Grant 2018YFB1003700 and Beijing Institute of Technology Research Fund Program for Young Scholars. The corresponding author is Zehua Guo.

NetAI '20, August 14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-8043-0/20/08…$15.00
https://doi.org/10.1145/3405671.3405809

1 INTRODUCTION

Traffic Engineering (TE) aims to optimize network performance (e.g., minimizing the maximum link utilization in the network) by configuring the routing across Internet Service Providers (ISPs)' backbone networks to control traffic distribution. Flow-based routing is a typical routing solution of TE. It offers fine-grained traffic distribution control by distributing traffic flows of each source-destination pair along a set of pregenerated paths. However, flow-based routing in IP networks suffer from a scalability issue as it normally requires to use Ternary Content-Addressable Memory (TCAM) for the flow table of each router. In the worst case, to distinguish source and destination addresses of packets, each router has to store $O\left(P^2\right)$ flow entries for a network with P IP routes (i.e., prefixes). Over recent years, the size of the Internet routing table is increasing exponentially [2]. Due to the high cost-to-density ratio and high power consumption of TCAM [15], routers usually have limited TCAM resources [5] and cannot accommodate a huge number of source-destination pairs in the network. An alternative solution for TE is to use destination-based routing, where routers make forwarding decisions solely based on the destination address of the packets. Destination-based routing has a lower forwarding complexity since each router maintains a forwarding table with, at most, $O(P)$ entries for a network with P IP routes. Moreover, instead of using TCAM as flow-based forwarding, destination-based forwarding has been widely implemented with simple Random-Access Memories (RAMs).

Most destination-based routing TE methods aim to optimize Interior Gateway Protocol (IGP) link costs to achieve good network performance [6, 11, 14]. Given link costs, shortest paths to each destination are calculated and forwarding entries are installed in each corresponding router, according to the Open Shortest Path First (OSPF) [22] or Intermediate System to Intermediate System (IS-IS) [23] protocol. Routers then evenly distribute traffic among multiple next hops (if exist) for a given destination, in terms of the Equal-Cost Multipath (ECMP) [27] split rule. A good link cost setting leads to significant performance improvement. However, it has been shown that it is an NP-hard problem to optimize the link costs for a network [6], and even traffic distribution imposed by ECMP introduces additional limitations on routing optimization [36]. Several recent works [34, 36] have been proposed to address the above issues. Weighted ECMP [36] extends ECMP to allow weighted traffic splitting at each node. However, the performance gain is limited by link costs and topologies. Zhang et al. [34] proved that an arbitrary flow-based routing can be converted to a loop-free destination-based routing without any performance penalty for a given traffic matrix. The scheme in [34] is guaranteed to achieve optimal performance. However, a centralized controller

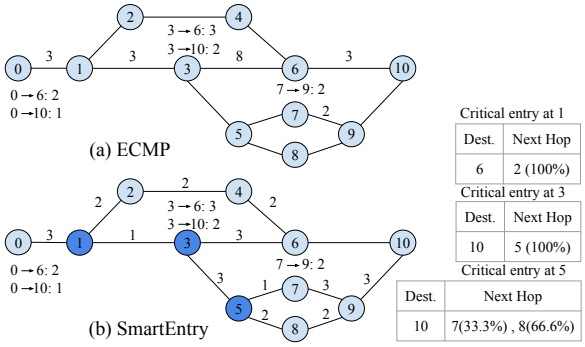

Figure 1: An illustrative example of SmartEntry. Each link is bidirectional with link weight and capability equal to 1.

has to update traffic split ratios for each entry of the forwarding table of each router in the network, which could take considerable time and impose a fundamental limitation on how responsively the network can react to dynamic changes of traffic demands.

To achieve good performance and low routing update overhead, one promising solution is forwarding the majority of flows using ECMP routing and then selectively and dynamically redistributing a small portion of traffic by installing a few critical destination-based forwarding entries as traffic changes. A simple example shown in Figure 1 demonstrates the advantages of this routing solution. Assume that router 0 sends 2 units of traffic to router 6 and 1 unit of traffic to router 10, respectively. Similarly, router 3 sends 3 units of traffic to router 6 and 2 units of traffic to router 10, respectively. In addition, router 7 sends 2 units of traffic to router 9. Figure 1(a) shows the traffic distribution under ECMP routing. Since ECMP routing is static and not traffic aware, no routing update is required, but link $\langle 3, 6\rangle$ becomes a bottleneck link. However, as shown in Figure 1(b), traffic from router 0 to router 6 can be rerouted to next hop 2 by installing a critical destination-based entry[1] at router 1, and thereafter be forwarded to destination 6 along the shortest path according to ECMP routing. The traffic destined to router 10 aggregated at router 3 can be rerouted to next hop 5 instead of ECMP next hop 6, and then be unevenly split among next hops 7 and 8, according to the other two critical destination-based entries. Note that the remaining traffic is still distributed by the static ECMP routing. The above routing solution achieves load balancing by complementing default ECMP routing with only three critical destination-based entries.

To introduce critical entries for a given traffic matrix, three problems should be addressed:

(1) At which critical routers should critical entries be installed?
(2) To which destinations the traffic belong should be redistributed, once critical routers are determined?
(3) How to redistribute the selected traffic among available next hops?

Although problem (3) is relatively simple and can be solved by formulating it as a Linear Programming (LP) optimization problem, solving problems (1) and (2) is not trivial. Given a network with N routers, there would be total $N*(N-1)$ candidate router-destination pairs for problems (1) and (2). The solution space $\binom{N*(N-1)}{K}$ would be enormous, even if we just want to introduce a small number K of critical entries. For example, when $N = 10$ and $K = 9$, the solution space has $\binom{90}{9} \approx 706$ billion combinations. Thus, it would be very difficult, if not impossible, to design a heuristic algorithm for the above problem based on fixed and simple rules, because rule-based heuristics are unable to adapt to the changes of the traffic matrix and network dynamics. Thus, we resort to Reinforcement Learning (RL) approaches that are well-suited to this combination selection problem. First, RL can model complex selection policies as neural networks, which provide a scalable and expressive way to incorporate various "raw" observations into the selection policy. Second, RL is able to train for objectives that lack of precise models and 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 signal, RL can optimize its selection policy under varying scenarios.

In this paper, we propose SmartEntry, a scheme that employs RL to learn a policy to efficiently and effectively select a combination of K router-destination pairs for each given traffic matrix, and obtains the corresponding loop-free rerouting split ratios by formulating and solving a rerouting optimization problem. The main contributions of this paper are summarized as follows:

(1) We customize a RL approach to learn the combination selection policy.
(2) We adopt Linear Programming (LP) as a reward function to produce reward signals, which reflect network performance for each combination selection. This RL + LP combined approach turns out to be surprisingly powerful for solving networking problems.
(3) We evaluate SmartEntry by conducting extensive experiments on different topologies with both real and synthesized traffic. The simulation results show that SmartEntry achieves near optimal performance and saves 90% forwarding entry updates. In addition, SmartEntry generalizes well to unseen traffic matrices.

2 PROPOSED MODEL

2.1 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 ($\|\tau_K\| = K$).
$\tau_{N*(N-1)-K}$	the set of remaining τ_i^d ($\|\tau_{N*(N-1)-K}\| = N * (N-1) - K$).
ENP_i^d	the set of ECMP next hops for destination d at node i ($i, d \in V$, e.g., $\text{ENP}_1^{10} = \{3\}$, $\text{ENP}_5^{10} = \{7, 8\}$ in Figure 1(b)).
$y_{i,j}^d$	the traffic destined to d routed on link $\langle i, j\rangle$ ($d \in V, \langle i, j\rangle \in E$).
$\beta_{i,j}^d$	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., $\beta_{5,8}^{10} = 66.6\%$ in Figure 1(b)).

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

2.2 Overview

Problem Statement: For a given traffic matrix, the task of SmartEntry is to (1) select a combination of K node-destination pairs (i.e., $\tau_i^d \in \tau_K$) and (2) redistribute the selected traffic among available next hops to balance link utilization of the network. It is a very challenging task, considering the variety of the traffic matrix and the complexity of the network topology.

For (1), we train SmartEntry to learn a selection policy over a rich variety of historical traffic matrices, where traffic matrices can be measured and collected by a centralized controller periodically [32]. The selection policy is represented as a neural network that maps a "raw" observation (e.g., a given traffic matrix) to a combination of τ_i^d (e.g., τ_1^6, τ_3^{10}, and τ_5^{10} in Figure 1(b)). For each selected combination, we formulate and solve an LP optimization problem for (2) to obtain a reward signal. Through reinforcement in the form of reward signal, the neural network is trained based on REINFORCE algorithm [31] with some customizations. Then, from the LP solution of (2), we can derive the optimal rerouting split ratios $\beta_{i,j}^d$. The centralized controller installs new critical entries at the critical routers according to the results of (1) and (2). Note that the installed critical entries in the previous period time out automatically.

There are two reasons we do not want to continuously adopt RL for (2). Firstly, since τ_K is small, the set of rerouting split ratios $\{\beta_{i,j}^d\}$ would be relatively small. LP is an efficient and optimal method to solve the rerouting problem. Secondly, split ratios are continuous numbers. Thus, we have to adopt the RL methods for continuous action domain [18] [24]. However, it has been shown that this type of RL methods would lead to slow and ineffective learning when the number of output parameters are relatively large [30][33]. In other words, compared to LP methods, applying RL methods to (2) may suffer from scalability issues.

2.3 Learning A Combination Selection Policy

The goal of RL is to learn a policy π that selects a combination of K "right" τ_i^d for each given traffic matrix, such that the network performance is maximized after redistributing traffic for each τ_i^d.

2.3.1 RL Formulation.

Input: An input instance s is represented as a traffic matrix TM, which contains information of traffic demand for each source-destination pair (i.e., $t^{s,d}$). Assume that the given network $G(V, E)$ remains unchanged, we do not include the topology information as a part of the input. The results in Section 3 show that RL is able to learn a good policy π without prior knowledge of the network.
Action Space: For a given instance s, SmartEntry wants to select K τ_i^d. Given that there are total $N*(N-1)$ candidate τ_i^d in a network with N nodes, this RL problem would require a large action space of size $\binom{N*(N-1)}{K}$. Inspired by [20][17], we define the action space as {0, 1, ..., $N*(N-1)-1$} and allow the RL agent to sample K different actions for each instance s (i.e., $a^1, a^2, ..., a^K$).
Reward: After sampling K different τ_i^d for a given instance s, SmartEntry solves the LP optimization problem (5a) (described in the following section) to obtain the maximum link utilization U_{max}. We define reward r as $\frac{1}{U_{max}}$, which is set to reflect the network performance after redistributing traffic for each τ_i^d. The greater reward r (i.e., the smaller U_{max}), the better performance.

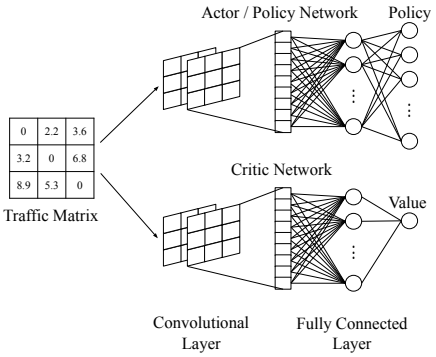

Figure 2: Actor-Critic architecture.

2.3.2 Training Algorithm.

We use a neural network to represent the policy. This policy network takes a TM as an input and outputs a probability distribution $\pi(a|s)$ over all available actions. Figure 2 shows the architecture of the policy (actor) network (details in Section 3.1.1). We define a solution $a_K = (a^1, a^2, ..., a^K)$ as a combination of K sampled actions, since we do not care the order of the sampled actions. For selecting a solution a_K with a given instance s, a stochastic policy $\pi(a_K|s)$ parameterized by θ can be approximated as follows[2]:

$$\pi_\theta(a_K|s) \approx \prod_{i=1}^{K} \pi_\theta(a^i|s). \tag{1}$$

Recall that the goal of learning is to find a policy π_θ that maximizes the network performance over various traffic matrices, i.e., maximizes the expected reward $E_{\pi_\theta(a_K|s)}[r]$. Thus, we optimize $E_{\pi_\theta(a_K|s)}[r]$ by gradient ascend, using REINFORCE algorithm with a baseline $b(s)$:

$$\nabla_\theta E_{\pi_\theta(a_K|s)}[r] = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a_K|s)(r - b(s))]. \tag{2}$$

A good baseline $b(s)$ reduces gradient variance and thus increases speed of learning. In this paper, we use a learned estimate of the value function $V^{\pi_\theta}(s)$ as the baseline $b(s)$. The critic network in Figure 2 is trained to learn an estimate of $V^{\pi_\theta}(s)$. The critic network parameter θ_v is updated according to the following equation:

$$\theta_v \leftarrow \theta_v - \alpha_v \sum_s \nabla_{\theta_v}(r - V_{\theta_v}^{\pi_\theta}(s))^2, \tag{3}$$

where $V_{\theta_v}^{\pi_\theta}(\cdot)$ is the estimate of $V^{\pi_\theta}(\cdot)$, output by the critic network, and α_v is the learning rate for the critic network. 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 to execute the action selection.

To ensure that the RL agent explores the action space adequately during training to discover good policies, we add the entropy of the policy π to Eq. (2). This technique improves exploration by discouraging premature convergence to suboptimal deterministic

[2]To select K distinct actions, we do the action sampling without replacement. The right side of Eq. (1) is the solution probability when sampling with replacement, but we still use Eq. (1) to approximate the probability of solution a_K given an instance s for simplicity.

Algorithm 1 Training Algorithm

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 a solution a_{i_K} according to policy $\pi_\theta(a_{i_K}|s_i)$
    Receive reward r_i
  end for
  for $i = 1, ..., B$ **do**
    $\Delta\theta \leftarrow \Delta\theta + \alpha(\nabla_\theta log\pi_\theta(a_{i_K}|s_i)(r_i - V^{\pi_\theta}_{\theta_v}(s_i)) + \beta\nabla_\theta H(\pi_\theta(\cdot|s_i)))$
    $\Delta\theta_v \leftarrow \Delta\theta_v - \alpha_v\nabla_{\theta_v}(r_i - V^{\pi_\theta}_{\theta_v}(s_i))^2$
  end for
  $\theta \leftarrow \theta + \Delta\theta, \theta_v \leftarrow \theta_v + \Delta\theta_v$
end for

policies [21]. Then, the policy network parameter θ is updated according to the following equation:

$$\theta \leftarrow \theta + \alpha \sum_s \nabla_\theta log\pi_\theta(a_K|s)(r - V^{\pi_\theta}_{\theta_v}(s)) + \beta\nabla_\theta H(\pi_\theta(\cdot|s)), \quad (4)$$

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

2.4 Rerouting Optimization Problem

As described in Section 1, traffic is either distributed evenly among the default ECMP next hops or split unevenly among available next hops according to the critical entries. Given a network $G(V, E)$ with a traffic matrix TM and τ_K, our objective is to obtain the weighted split ratios $\{\beta^d_{i,j}\}$ for the selected $\tau^d_i \in \tau_K$, so that U_{max} is minimized and the routing is loop-free. To achieve this objective, we first obtain the optimal destination-based traffic allocation $\{y^d_{i,k}\}$ (where $y^d_{i,k}$ stands for the traffic destined to d routed on link $\langle i, j\rangle$), then derive $\{\beta^d_{i,j}\}$ from $\{y^d_{i,k}\}$. We formulate the destination-based rerouting problem as an optimization problem as follows:

$$minimize \quad U_{max} + \epsilon \cdot \sum_{\langle i,j\rangle \in E} \sum_{d \in V} y^d_{i,j} \quad (5a)$$

subject to

$$\sum_{d \in V} y^d_{i,j} = l_{i,j} \qquad i, j : \langle i, j\rangle \in E \quad (5b)$$

$$l_{i,j} \le c_{i,j} \cdot U_{max} \qquad i, j : \langle i, j\rangle \in E \quad (5c)$$

$$\sum_{k:\langle k,i\rangle \in E} y^d_{k,i} - \sum_{k:\langle i,k\rangle \in E} y^d_{i,k} = -t^{i,d} \quad i, d : \tau^d_i \in \tau_K \quad (5d)$$

$$y^d_{i,k} = \begin{cases} \dfrac{\sum_{n:\langle n,i\rangle \in E} y^d_{n,i} + t^{i,d}}{|\mathrm{ENH}^d_i|} & \text{if } k \in \mathrm{ENH}^d_i \\ 0 & \text{otherwise} \end{cases} \quad (5e)$$

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

$$\sum_{k:\langle k,d\rangle \in E} y^d_{k,d} - \sum_{k:\langle d,k\rangle \in E} y^d_{d,k} = \sum_{s \in V, s \ne d} t^{s,d} \quad d \in V \quad (5f)$$

$$y^d_{i,j} \ge 0 \qquad d \in V,\ i, j : \langle i, j\rangle \in E \quad (5g)$$

$\epsilon \cdot \sum_{\langle i,j\rangle \in E} \sum_{d \in V} y^d_{i,j}$ in (5a) makes sure the routing is loop-free [34], where ϵ ($\epsilon > 0$) is a sufficiently small constant to ensure that the minimization of U_{max} takes higher priority. (5c) is the link capacity utilization constraint. (5d), (5e), (5f) are the flow conservation constraints for the selected τ^d_i, for the remaining τ^d_i, and at destinations, respectively.

By solving problem (5) using LP solvers (e.g., Gurobi [10]), we can obtain the optimal $\{y^d_{i,k}\}$, then $\{\beta^d_{i,j}\}$ can be derived according to the following equation:

$$\beta^d_{i,j} = \frac{y^d_{i,j}}{\sum_{k:\langle i,k\rangle \in E} y^d_{i,k}} \quad i, d : \tau^d_i \in \tau_K,\ j : \langle i, j\rangle \in E. \quad (6)$$

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

2.5 Traffic Splitting

By leveraging 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 longest prefix matching. When an incoming packet's destination IP address matches with a 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 next hop's IP address and an output port number. To facilitate our proposed TE method, 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 $\beta^d_{i,j}$ (H = the number of output ports, e.g., 64). When the flag is set, up to H split ratios provided by the centralized control 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^d_i. In practice, packets belonging to a TCP (or UDP) session follow a single path to avoid packet mis-order. An approximate 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 [13] and the standard hashing technique [3]).

3 EVALUATION

In this section, we conduct extensive experiments using different network topologies to evaluate the performance of SmartEntry and demonstrate its effectiveness.

3.1 Evaluation Setup

3.1.1 Implementation. The policy neural network consists of three layers. The first layer is a convolutional layer with 128 filters, The corresponding kernel size is 3×3 and the stride is set to 1. The

Table 1: ISP networks used in evaluation

Topology	Nodes	Directed Links
Abilene	12	30
GÉANT	23	74
EBONE	23	76
Tiscali	49	172

second layer is a fully connected layer with 128 neurons. The activation function used for previous two layers is Leaky Relu [19]. The last layer is a fully connected linear layer (without activation function) with $N \times (N-1)$ neurons corresponding to all possible actions. The softmax function is applied upon the output of final layer to generate the probabilities for all available actions. The critic network is similar to policy network except that the last layer is a fully connected linear layer with only one neuron corresponding to the baseline $b(s)$. The learning rates α and α_v are configured to decay from 0.001 to 0.0001 over 0.5×10^5 iterations. Additionally, the entropy factor β is configured to 0.01. We fixed all these hyperparameters throughout our experiments. The results in the following experiments show SmartEntry works well on different network topologies with a single set of fixed hyperparameters.

3.1.2 Dataset. In our evaluation, we use four real-world network topologies, including Abilene network, GÉANT network and two European ISP networks (i.e., EBONE and Tiscali) collected by ROCKETFUEL [25]. The numbers of nodes, directed links and source-destination pairs of the topologies are shown in Table 1. For Abilene network, the topology information (such as link connectivity, costs, and capacities) and measured traffic matrices are available at [1]. Since Abilene traffic matrices are measured every 5 minutes, there are total 288 traffic matrices per day. To evaluate the performance of SmartEntry, we choose the total 2016 traffic matrices in the first week (starting from Mar. 1st 2004) as our dataset. For GÉANT network [29], the link capacities and costs are provided by the authors, and the measured traffic matrices are available at [8], which are collected every 15 minutes for a continuous period of 4 months. Similarly, we select the total 672 traffic matrices in the first week (starting from Jan. 1st 2005) as our dataset. For ROCKETFUEL topologies, the link costs are given while the link capacities are not provided. Therefore, we infer the link capacities as the inverse of link costs, which are based on the default link cost setting of Cisco routers. In other words, the link costs are inversely proportional to the link capacities. This approach is commonly adopted in literature [16][34]. Traffic matrices are unavailable for the two ISP networks from ROCKETFUEL. Thus, we synthesize 50 traffic matrices using exponential model [28] and 50 traffic matrices using uniform model for each network. Unless otherwise noted, we use a random sample of 70% of our dataset as a training set for SmartEntry and use the remaining 30% as a test set to evaluate the effectiveness of SmartEntry.

3.1.3 Performance Ratio. To evaluate the performance of SmartEntry, a performance ratio is applied and defined as $PR = \frac{U_{\text{optimal}}}{U_{\text{SmartEntry}}}$, where U_{optimal} is the maximum link utilization achieved by the optimal flow-based routing. $PR = 1$ means that SmartEntry performs as good as the optimal routing. A lower ratio indicates that the performance of SmartEntry is further away from that of the optimal routing.

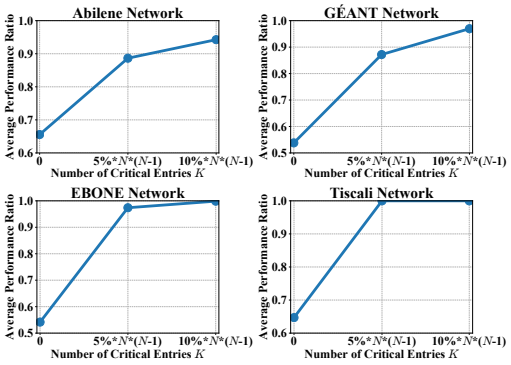

Figure 3: Average performance ratio of SmartEntry with increasing number of critical entries K in four networks.

3.1.4 Baselines. For comparison, we also evaluate two other schemes:

(1) **ECMP**: distributes traffic evenly among available next hops along the shortest paths. The link cost setting for each network was discussed in Section 3.1.2.
(2) **Weighted ECMP**: 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 [36].

3.2 Experiments

3.2.1 Number of Critical Entries. We conduct a series of experiments with different number of critical entries introduced by SmartEntry, and fix other parameters throughout the experiments.

Figure 3 shows the average performance ratio achieved by SmartEntry with increasing number of critical entries K on the four networks. Note that there are total $N*(N-1)$ candidate τ_i^d in a network with N destination nodes. For each network, we compare the performance with 0%, 5% and 10% of $N*(N-1)$ critical entries, as represented in the number of K on the x axis. The initial value with $K = 0$ represents the default ECMP routing. The results indicate that there is a considerable room for further improvement when flows are routed by ECMP. When we only introduce 5% of $N*(N-1)$ critical entries, there is at least 23.1% performance improvement over ECMP, which demonstrates the effectiveness of SmartEntry. Besides, the sharp increases in the average performance ratio for the four networks shown in Figure 3 indicate that SmartEntry is able to achieve near optimal performance by introducing only 10% of $N*(N-1)$ critical entries. For the subsequent experiments, we set $K = 10\% * N*(N-1)$ for each network.

3.2.2 Performance Gain. To demonstrate the performance gain introduced by SmartEntry's critical entries, we also calculate the performance ratios for ECMP and weighted ECMP. Figure 4 shows the performance ratio on each individual traffic matrix for four networks. As illustrated in Figure 4(c)(d), the first 15 traffic matrices for EBONE and Tiscali networks are sampled from the 50 exponential traffic matrices in the dataset, and the remaining 15 traffic matrices are sampled from the 50 uniform traffic matrices. All the results show that SmartEntry performs consistently well in all four networks under various traffic models. For example, in Abilene and GÉANT networks, SmartEntry improves performance

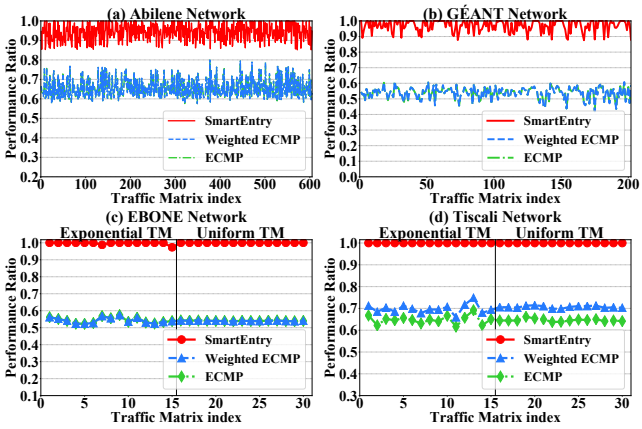

Figure 4: Comparison of performance ratio in four networks on each test traffic matrix. The higher, the better.

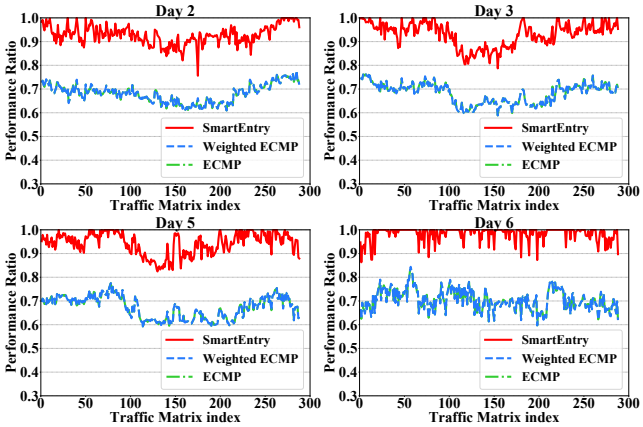

Figure 5: Comparison of performance ratio with the Abilene traffic matrices from Days 2, 3, 5, and 6 in week 2. The higher, the better.

by at least 20% and 40% compared to weighted ECMP, respectively. SmartEntry achieves optimal performance in most of the traffic matrices and outperforms weighted ECMP with at least 40% and 20% performance improvement in EBONE and Tiscali networks. It is worth noting that there might be only one shortest path for most of source-destination pairs in the small networks (e.g., Abilene, GÉANT, and EBONE networks). Thus, the performance improvement of weighted ECMP is limited, compared to ECMP. Overall, the results indicate that SmartEntry is able to effectively balance link utilization of the network by smartly installing a few critical entries in some critical routers.

3.2.3 Generalization. In this series of experiments, we train SmartEntry on the traffic matrices in the first week (starting from Mar. 1st 2004) and evaluate it for each day of the following week (starting from Mar. 8th 2004) for Abilene network. For GÉANT network, we conduct similar experiments by training SmartEntry with the traffic matrices in the first week (starting from Jan. 1st 2005) and testing its performance on each day of the second week (starting from Jan. 8th 2005). Due to the limited space, we only present the results of days 2, 3, 5, and 6 for the two networks. The results of other days are

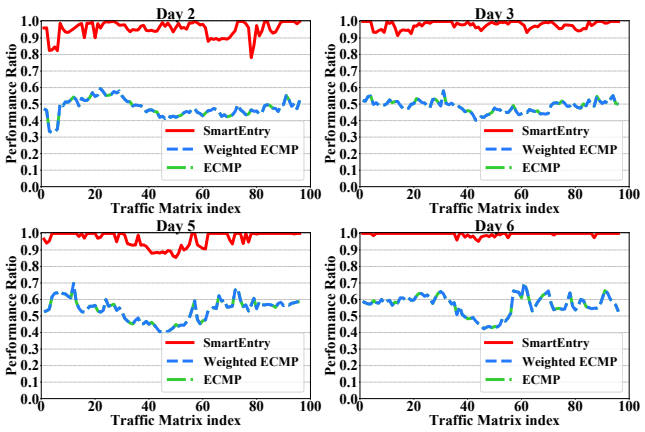

Figure 6: Comparison of performance ratio with the GÉANT traffic matrices from Days 2, 3, 5, and 6 in week 2. The higher, the better.

very similar. Figures 5 and 6 show the performance ratio on each traffic matrix of these 4 days for Abilene and GÉANT networks. SmartEntry still achieves near-optimal performance in almost all traffic matrices. The performance of SmartEntry degrades on several outlier traffic matrices. It is possible that the traffic patterns of these traffic matrices are different from what SmartEntry learned from the previous week. Overall, the results indicate that real traffic patterns are relatively stable, and SmartEntry generalizes well to unseen traffic matrices for which it was not explicitly trained.

4 RELATED WORK

Existing works use OSPF and ECMP protocols to evenly balance link utilization by carefully tuning the link costs or traffic splitting ratio to adjust path selection [4][7][12][36]. Dynamic hybrid routing [35] achieves load balancing for a wide range of traffic scenarios by dynamically rebalancing traffic to react to traffic fluctuations with a preconfigured routing policy. Machine learning has been used to realize the TE. In [30], the authors show deep reinforcement learning is a promising solution to improve the performance of TE. In [9], the authors use semi-supervised deep learning to design an automatic network protocol. Sun et al. [26] selectively control a set of nodes and use a RL-based policy to dynamically change the routing decision of flows traversing the selected nodes. Xu et al. [33] use reinforcement learning to optimize the throughput and delay in TE.

5 CONCLUSION

With an objective of minimizing maximum link utilization in a network and mitigating routing update overhead, we proposed SmartEntry, a scheme that learns a combination selection policy automatically using reinforcement learning, without any domain-specific rule-based heuristic. SmartEntry smartly selects a combination of K node-destination pairs for each given traffic matrix and reroutes the selected traffic to achieve load balancing of the network by solving a rerouting optimization problem. Extensive evaluations show that SmartEntry achieves near-optimal performance and generalizes well to traffic matrices for which it was not explicitly trained.

REFERENCES

[1] Abilene. [n.d.]. Yin Zhang's Abilene TM. http://www.cs.utexas.edu/~yzhang/research/AbileneTM/

[2] Tony Bates, Philip Smith, and Geoff Huston. [n.d.]. CIDR report. http://www.cidr-report.org/

[3] Zhiruo Cao, Zheng Wang, and E. Zegura. 2000. Performance of hashing-based schemes for Internet load balancing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 1. 332 –341 vol.1. https://doi.org/10.1109/INFCOM.2000.832203

[4] Jian Chu and Chin-Tau Lea. 2009. Optimal link weights for IP-based networks supporting hose-model VPNs. *IEEE/ACM Transactions on Networking (TON)* 17, 3 (2009), 778–788.

[5] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. DevoFlow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2011), 254–265. https://doi.org/10.1145/2043164.2018466

[6] B. Fortz and M. Thorup. 2002. Optimizing OSPF/IS-IS weights in a changing world. *Selected Areas in Communications, IEEE Journal on* 20, 4 (May 2002), 756 –767. https://doi.org/10.1109/JSAC.2002.1003042

[7] Bernard Fortz and Mikkel Thorup. 2002. Optimizing OSPF/IS-IS weights in a changing world. *IEEE journal on selected areas in communications* 20, 4 (2002), 756–767.

[8] GÉANT. [n.d.]. The TOTEM Project. https://totem.info.ucl.ac.be/dataset.html

[9] Fabien Geyer and Georg Carle. 2018. 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, 40–45.

[10] LLC Gurobi Optimization. 2019. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[11] Kaj Holmberg and Di Yuan. 2004. Optimization of Internet Protocol network design and routing. *Networks* 43, 1 (2004), 39–53. https://doi.org/10.1002/net.10102

[12] Kaj Holmberg and Di Yuan. 2004. Optimization of internet protocol network design and routing. *Networks: An International Journal* 43, 1 (2004), 39–53.

[13] C. Hopps. 2000. Analysis of an Equal-Cost Multi-Path Algorithm. *IETF RFC 2992* (November 2000).

[14] Chu Jian and Lea Chin-Tau. 2009. Optimal link weights for IP-Based networks supporting Hose-Model VPNs. *Networking, IEEE/ACM Transactions on* 17, 3 (June 2009), 778 –788.

[15] Kalapriya Kannan and Subhasis Banerjee. 2013. Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN. In *Distributed Computing and Networking*. Springer, 439–444.

[16] Murali Kodialam, TV Lakshman, James B Orlin, and Sudipta Sengupta. 2008. Oblivious routing of highly variable traffic in service overlays and IP backbones. *IEEE/ACM Transactions On Networking* 17, 2 (2008), 459–472.

[17] Wouter Kool, Herke van Hoof, and Max Welling. 2018. Attention, Learn to Solve Routing Problems! arXiv:stat.ML/1803.08475

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

[19] Andrew L. Maas. 2013. Rectifier Nonlinearities Improve Neural Network Acoustic Models.

[20] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, New York, NY, USA, 50–56. https://doi.org/10.1145/3005745.3005750

[21] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. *CoRR* abs/1602.01783 (2016). arXiv:1602.01783 http://arxiv.org/abs/1602.01783

[22] J. Moy. 1998. OSPF Version 2. *IETF RFC 2328* (April 1998).

[23] D. Oran. 1990. OSI IS-IS Intra-domain Routing Protocol. *IETF RFC 1142* (February 1990).

[24] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. Trust Region Policy Optimization. *CoRR* abs/1502.05477 (2015). arXiv:1502.05477 http://arxiv.org/abs/1502.05477

[25] Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with Rocketfuel. In *ACM SIGCOMM Computer Communication Review*, Vol. 32. ACM, 133–145.

[26] Penghao Sun, Junfei Li, Zehua Guo, Yang Xu, Julong Lan, and Yuxiang Hu. 2019. Sinet: Enabling scalable network routing with deep reinforcement learning on partial nodes. In *ACM SIGCOMM'19 Posters and Demos*. 88–89.

[27] D. Thaler and C. Hopps. 2000. Multipath Issues in Unicast and Multicast Next-Hop Selection. *IETF RFC 2991* (November 2000).

[28] Paul Tune and Matthew Roughan. 2015. Spatiotemporal traffic matrix synthesis. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 579–592.

[29] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. 2006. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communication Review* 36, 1 (2006), 83–86.

[30] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. 2017. Learning to Route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets-XVI)*. ACM, New York, NY, USA, 185–191. https://doi.org/10.1145/3152434.3152441

[31] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3 (01 May 1992), 229–256. https://doi.org/10.1007/BF00992696

[32] H. Xu, Z. Yu, C. Qian, X. Li, and Z. Liu. 2017. Minimizing flow statistics collection cost of SDN using wildcard requests. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 1–9. https://doi.org/10.1109/INFOCOM.2017.8056992

[33] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. 2018. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1871–1879.

[34] Junjie Zhang, Kang Xi, and H Jonathan Chao. 2015. Load balancing in IP networks using generalized destination-based multipath routing. *IEEE/ACM Transactions on Networking (TON)* 23, 6 (2015), 1959–1969.

[35] Junjie Zhang, Kang Xi, Min Luo, and H Jonathan Chao. 2014. Dynamic hybrid routing: Achieve load balancing for changing traffic demands. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*. IEEE, 105–110.

[36] Junjie Zhang, Kang Xi, Liren Zhang, and H Jonathon Chao. 2012. Optimizing network performance using weighted multipath routing. In *2012 21st International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–7.