Roracle: Enabling Lookahead Routing for Scalable Traffic Engineering with Supervised Learning

Minghao Ye[†], Junjie Zhang[‡], Zehua Guo[§*], H. Jonathan Chao[†]
[†]New York University [‡]Fortinet, Inc. [§]Beijing Institute of Technology
Email: {minghao.ye, junjie.zhang, chao}@nyu.edu, guolizihao@hotmail.com

***Abstract*—Traditional Traffic Engineering (TE) usually balances the load on network links by formulating and solving a routing optimization problem based on measured Traffic Matrices (TMs). Given that traffic demands could change unexpectedly and significantly in realistic scenarios, routing strategies optimized based on currently measured TMs might not work well in future traffic scenarios. To compensate for the mismatch between stale routing decisions and future TMs, network operators may perform routing updates more frequently, which could introduce significant network disturbance and service disruption. Moreover, given the high routing computation overhead of TE optimization in today's large-scale networks, routing updates could experience severe delay and thus cannot accommodate future traffic changes in time. To address these challenges, we propose Roracle, a scalable learning-based TE that quickly predicts a good routing strategy for a long sequence of future TMs, while the learning process is guided by the optimal solutions of Linear Programming (LP) problems using Supervised Learning (SL). We design a scalable Graph Neural Network (GNN) architecture that greatly facilitates training and inference processes to accelerate TE in large networks. Extensive simulation results on real-world network topologies and traffic traces show that Roracle outperforms existing TE solutions by up to 36% in terms of worst-case performance under future unknown traffic scenarios. Additionally, Roracle achieves good scalability by providing at least 71× speedup over the most efficient baseline method in large-scale networks.**

I. Introduction

Traffic Engineering (TE) is an efficient operation that enables network operators to optimize network performance and resource utilization by configuring the routing across their backbone networks [1], [2]. To minimize network congestion probability, traditional TE [3]–[8] usually balances the load on network links by formulating and solving a routing optimization problem with a specific objective, such as minimizing the Maximum Link Utilization (MLU).

For TE purposes, network operators can periodically measure a Traffic Matrix (TM) by sampling traffic volume over a past time interval. TE usually takes a measured TM as the input and operates periodically, e.g., every 5 minutes, to optimize and update routing. For example, a TM is measured at $t = 5$ to record the traffic volume for $t \in (0, 5]$. TE can compute an updated routing based on the measured TM, and then apply the routing for $t \in (5, 10]$. However, the traffic demands among nodes could change unexpectedly and significantly in realistic scenarios. As a result, the routing, which is optimized based on the currently measured TM, might not work well for the traffic demands in the future period and potentially introduce routing performance degradation.

To maintain good network performance, network operators may want to compensate for the mismatch between stale routing strategies and future TMs by updating the routing frequently. However, it could introduce two critical issues. First, frequent routing updates may introduce significant network disturbance and service disruption [9], [10]. A recent study [10] has shown that frequent routing updates would result in severe Quality of Service (QoS) degradation in the AWS network. This is because routing updates could temporarily affect many TCP flows' normal operation with packet loss and reordering issues, which in turn increase the flow completion time and degrade the QoS. Second, frequent routing updates pose great challenges to the scalability of TE solutions. Given that the network size has grown by 10× in the last decade, it becomes computationally intractable for traditional TE solutions to solve a routing optimization problem for large networks (e.g., hundreds of nodes and links) within a limited time budget [11]. As a result, such delayed routing decisions may not be able to accommodate traffic changes in time, which could lead to severe routing performance degradation.

Recently, emerging Machine Learning (ML) techniques provide new opportunities for TE to address the above issues. Some recent works focus on predicting TMs to adapt to future traffic scenarios [12]–[16]. However, there is no guarantee to bound the error between predicted TMs and real future TMs in a small range. Given the inaccurately predicted TMs as the input, it would be difficult for TE to make good routing decisions for future traffic scenarios. Moreover, these TM prediction-based TE solutions would suffer from high computation overhead in large-scale networks, as they still need to solve a routing optimization problem based on the predicted TMs to derive routing strategies during online deployment.

Essentially, the main problem of the above solutions arises from the fully coupled relationship between the accuracy of predicted TMs and the performance of optimized routing strategies, as well as the high complexity of routing optimization. To solve the problem, one promising solution is to leverage ML techniques to directly predict good routing strategies for future traffic scenarios. Compared to TM prediction-based TE solutions, routing prediction is more decision-focused and provides better scalability, as there is no need to perform real-

* Corresponding author.

979-8-3503-0322-3/23/$31.00 ©2023 IEEE

time routing optimization during online deployment.

Inspired by the above insight, we propose Roracle, a scalable learning-based TE solution that provides a good routing strategy for a long sequence of future TMs to maximize network performance in future TMs, mitigate network disturbance, and reduce TE computation overhead. The core concept behind Roracle is to directly predict routing strategies for future traffic scenarios using Supervised Learning (SL). Roracle can directly learn from optimal routing for future traffic scenarios during offline training, which eliminates the need to explore a large solution space. To simplify routing prediction and facilitate training and inference in large-scale networks, Roracle leverages an oblivious routing [17], [18] algorithm to compute and preconfigure a set of diverse forwarding paths [6], and then simply predicts path split ratios for each source-destination pair with less routing variables. Moreover, we design a scalable Graph Neural Network (GNN) [19], [20] architecture based on graph representation learning techniques, where each module/layer is shareable and reused by each node in parallel. We compare Roracle with existing TE solutions through simulations on real-world network topologies and traffic traces. Evaluation results show that Roracle outperforms existing TE solutions in future traffic scenarios by at most 36% in terms of worst-case MLU ratio, and also achieves good scalability with at least 71× speedup over the most efficient baseline in large-scale networks.

The contributions of this paper are summarized as follows:

- We propose a learning-based TE called Roracle to predict good routing strategies for future unknown traffic scenarios with mitigated disturbance and good scalability.
- We customize a scalable neural network architecture based on GNNs to greatly accelerate training and inference processes.
- Evaluation results show that Roracle outperforms existing TE solutions in future traffic scenarios with quick routing inference in large-scale networks.

The rest of the paper is organized as follows. In Section II, we discuss existing problems and challenges. Section III provides an overview of Roracle. Section IV details Roracle's design. Section V explains implementation details. We evaluate the performance of Roracle in Section VI. Section VII lists related work, and Section VIII concludes this paper.

II. Problem Statement and Challenges

A. Limitation of Existing TE Solutions

Traditional TE usually solves a routing optimization problem based on the network topology and a recently measured TM that records the average traffic demand volume of all source-destination pairs in the last time interval. Given a capacitated network, the task of TE optimization is to solve for optimal routing under multiple constraints to minimize the MLU in the network, thereby achieving good load balancing performance and reducing congestion probability. Here, MLU represents the utilization of the most congested link in the network, where link utilization is defined as the ratio of the traffic load on the link over the link capacity. This routing optimization problem is known as the Multi-Commodity Flow (MCF) problem [21], which can be formulated as a Linear Program (LP).

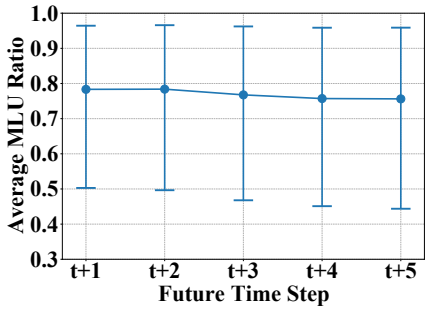


Fig. 1. Load balancing performance degradation in the future time period with traditional TE. The error bars span from the average value to the 5th and 95th percentile values achieved in each future time step.

Since future demands cannot be obtained in advance, TE can only optimize routing based on historical traffic demands, which will inevitably result in a stale routing that degrades network performance in future traffic scenarios. To evaluate the impact of the mismatch between stale routing strategies and future TMs, we conduct a series of simulation experiments based on real TMs from the large-scale BRAIN network with 161 nodes and 332 links [22], [23]. Similar to the settings of traditional TE, the routing is periodically optimized and updated every 5 minutes by solving an MCF problem [21], where the input is the currently measured 5-minute TM that records the average traffic demand volume in the last 5 minutes. Then, the computed routing is applied in the future 5-minute period for routing performance evaluation, including five fine-grained 1-minute future TMs (from TM_{t+1} to TM_{t+5}). For comparison, we also evaluate the optimal routing for each real future TM that leads to the lowest possible MLU (which is infeasible in practice) and use a performance metric called MLU ratio to demonstrate routing performance degradation, which is obtained by comparing the MLU of optimal routing to that of stale routing (see Section V-B).

1) Routing performance degradation in future TMs: Fig. 1 shows the experiment result, which depicts the degradation of the MLU ratio in each future time step of the future 5-minute period. Given the stale routing strategies from traditional TE solutions, we can observe that the network MLU ratio degrades by more than 20% on average compared to optimal routing in future TMs. In the worst-case scenarios, the MLU ratio could be dramatically degraded by more than half, which reveals the limitations of traditional TE in accommodating future traffic fluctuations. Since the stale routing is optimized based on past traffic demands, it is no longer effective when future traffic scenarios are different from past traffic scenarios.

2) Limitations of frequent routing updates: Network operators may consider increasing the frequency of routing updates (e.g., every minute) to improve network performance. However, frequent routing updates cannot fundamentally address the above issue since the routing is still optimized based on currently measured TMs without considering future traffic scenarios. As shown in Fig. 1, the MLU degradation in the

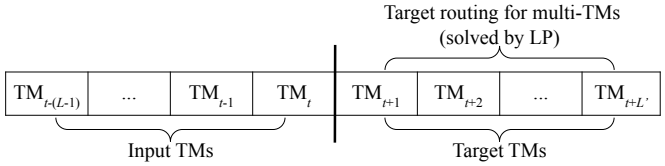


Fig. 2. Target routing to be learned by routing prediction-based TE solutions. Instead of predicting a long sequence of future TMs, these solutions directly predict a routing to accommodate future traffic scenarios and accelerate TE operations while mitigating network disturbance.

future 1-minute time step (i.e., TM_{t+1}) is still significant enough with more than 50% of MLU ratio degradation in the corner cases, which confirms the limitations of frequent routing updates. Moreover, frequent routing updates would introduce significant network disturbance and service disruption.

3) Poor scalability in large networks: Traditional TE solutions often encounter scalability issues due to the high computation overhead of routing optimization. In our experiments, it usually requires 7-8 minutes to solve an MCF routing optimization problem [21] in the large-scale BRAIN network, as later shown in Section VI-D. Obviously, such time overhead has greatly exceeded the traditional 5-minute routing update intervals, which results in a severe delay in TE's reaction to traffic changes. As shown in Fig. 1, routing performance gradually degrades when the stale routing is applied for a longer future time step (e.g. from TM_{t+1} to TM_{t+5}), which reflects the limitation of traditional TE solutions in terms of scalability as well as the importance of fast routing computation.

B. Limitation of TM Prediction-based TE Solutions

One potential idea to address the above issues (1) and (2) is to formulate and solve a multi-TMs routing optimization problem using LP (details in Section IV-C), where these given TMs should represent the future traffic scenarios to avoid performance degradation caused by traffic fluctuations. Unfortunately, future TMs are unknown beforehand during online deployment. A straightforward solution is to predict future TMs based on historically measured TMs [12], [14], [16]. By capturing temporal and spatial relationships in real TMs, the next future TM can be predicted based on historically measured TMs. To reduce the frequency of routing updates and mitigate network disturbance, it is also possible to predict the next multiple future TMs for routing optimization with a longer time interval. However, as we discussed in Section I, TM prediction-based TE solutions are prone to prediction errors due to the coupled relationship between TM prediction accuracy and optimized routing performance. Moreover, these methods cannot bypass the scalability issue (3) due to the high computation overhead of solving a routing optimization problem in large networks based on multiple predicted TMs.

C. Our Insight

Based on the above analysis, an effective TE solution should achieve three key requirements: (1) providing a lookahead routing that is capable of handling future TMs to avoid network performance degradation; (2) being suitable for a long sequence of TMs in a future period to prevent frequent routing update and thus mitigate network disturbance; (3) achieving good scalability with quick and efficient routing computation to cope with traffic changes in large-scale networks.

To achieve these requirements, one promising solution is to directly predict a good routing strategy for future traffic scenarios based on historically measured TMs. Fig. 2 shows the target routing strategy to be learned by ML. In this figure, the "ground truth" data (target routing) is obtained by solving a multi-TMs routing optimization problem for future TMs using LP, which is feasible during the training stage since we can construct training samples with historical TMs and future TMs from a pre-collected training TM dataset. Even though the computation overhead of multi-TMs routing optimization might be unacceptable for real-time routing updates in large networks, it is suitable to serve as the learning target during the offline training process. Besides, such time-consuming routing optimization can be omitted during online deployment since the well-trained ML model can directly and quickly generate routing strategies through neural network inference.

Given that the proposed routing prediction method can provide a lookahead routing to accommodate future traffic scenarios without frequent routing updates, requirements (1) and (2) can be satisfied. Moreover, routing prediction is a decision-focused approach that bypasses routing optimization during online deployment, which also enables efficient routing inference in large-scale networks to satisfy requirement (3). Therefore, routing prediction can be considered a promising paradigm to address the emerging needs of network operators.

D. Challenges

However, we still face some technical challenges when designing a routing prediction framework.

1) How to reduce routing complexity: When predicting a routing for a network topology with N nodes and E links, the size of a prediction model's output variables becomes a big concern. Take explicit routing as an example, there would be $N \times (N-1) \times E$ routing variables representing the traffic split ratios for each source-destination pair on each link. In the BRAIN network [22], [23] with $N = 161$ nodes and $E = 332$ links, the total number of routing variables would be $161 \times 160 \times 332 = 8552320$. It is very difficult, if not impossible, to accurately predict such a large number of variables. Destination-based routing is an alternative, which only requires $(N-1) \times E$ routing variables due to the hop-by-hop forwarding property. But, the probabilistic output of a prediction model is not guaranteed to be a loop-free routing.

2) How to select ML technique: Reinforcement Learning (RL) is a popular approach for designing scalable TE solutions. However, there are two limitations in terms of the training efficiency of RL-based methods. First, RL agents need to interact with an environment to learn a good policy by exploring different actions and receiving their corresponding rewards. With $N \times (N-1) \times E$ routing variables, the action space could be too large for RL agents to efficiently discover a good routing policy in today's large-scale networks. Second, since routing variables are continuous numbers, RL methods

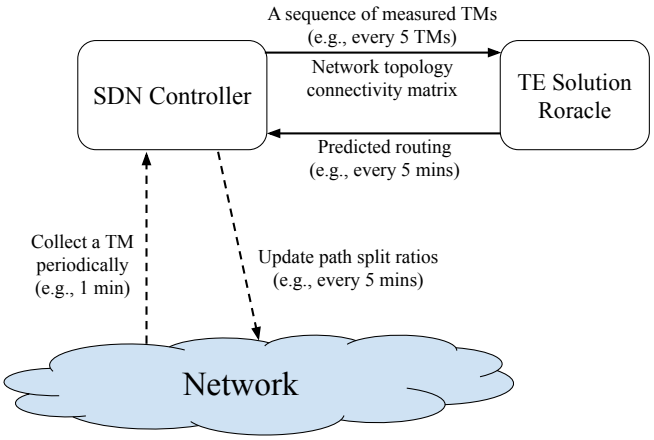


Fig. 3. Overview of Roracle. Roracle predicts a routing strategy and updates path split ratios periodically based on historically measured TMs.

for continuous action domain [24], [25] should be adopted to output continuous variables. However, when the number of output variables is relatively large, this type of RL method could lead to slow and ineffective learning [26], [27].

3) How to design a scalable neural network architecture: In our problem, the input consists of a sequence (e.g., L) of $N \times N \times L$ historical TMs and a $N \times N$ connectivity matrix of the topology, and the output could be a $N \times (N-1) \times E$ traffic split ratio vector, i.e., a lookahead routing targeting multiple future TMs. If we design a routing prediction framework with traditional neural network architectures (e.g., fully connected neural network or convolutional neural network), the prediction model would exponentially expand as the number of nodes N and links E increases. In today's large-scale networks, the high dimensional input and output of the prediction model could create a large neural network that is difficult to train.

III. Overview

A. System Overview

To address the above challenges, we propose a learning-based routing prediction framework called Roracle. Given a sequence of historically measured TMs and a topology connectivity matrix, Roracle predicts traffic split ratios for each source-destination pair on multiple paths as a lookahead routing strategy. To facilitate TE operations, Roracle can be integrated with emerging Software-Defined Networking (SDN) techniques [28], which provides a global view of the network and enables flexible deployment of routing policies. Fig. 3 illustrates an overview of Roracle, where traffic demands can be measured by SDN switches and collected by an SDN central controller periodically [29]. Once the SDN controller assembles and accumulates a sufficient number of TMs (e.g., five 1-minute TMs in the last 5 minutes), these TMs would be forwarded to Roracle along with the topology information. Then, Roracle quickly predicts a routing based on these inputs, which would be installed in the network by the SDN controller.

B. Main Techniques

We briefly introduce the main techniques we adopted in Roracle to address the technical challenges in Section II-D.

Path-based Routing. We leverage an oblivious routing [17], [18] algorithm to compute and preconfigure a set of diverse forwarding paths for each source-destination pair and then simply predict path split ratios. As a result, the number of output variables dramatically decreases compared to that of link-based routing. Given κ preconfigured paths for each source-destination pair, the number of output variables is now only $N \times (N-1) \times \kappa$, where $\kappa \ll E$. We choose 4 paths (i.e., $\kappa = 4$) with the highest weights assigned by oblivious routing to achieve a good tradeoff between performance and complexity [6]. Since these paths are low stretch, diverse, and optimized for load balancing [6], the distribution of path forwarding entries across nodes becomes more even, which can alleviate the path storage burden in large networks.

Supervised Learning. We train Roracle using SL instead of RL to improve training efficiency. It is feasible to train a prediction model using SL with the help of LP, although the size of path split ratios is not small. As long as LP can precompute the target routing for large networks in a reasonable timescale during offline training, it would be sufficient for Roracle to converge with low training overhead. Compared to the slow and ineffective learning process of RL with trial and error, SL can leverage LP to generate the target routing as an expert solution (i.e., optimal multi-TMs routing for future TMs), which can efficiently guide the learning process of the routing prediction model to accelerate convergence.

Graph Neural Networks. Compared to traditional neural network architectures, GNNs offer unique advantages in modeling graph-structured data, which is suitable for network topologies since they can be represented as graphs. Based on GNNs, we leverage graph representation learning techniques to characterize the TMs and topology information into a low-dimensional embedding space and then perform message exchange between neighboring nodes. Among the variants of GNNs, we choose Graph Attention Networks (GATs) [20] to design the routing prediction model since GAT supports flexible aggregation of neighboring features based on learnable weights. The main advantage of the GNN-based prediction model is that each module can be shared and reused by each node in parallel with lower dimensional input and output, which greatly improves training efficiency and reduces routing inference time in large networks. Moreover, GNN can efficiently capture the topology and demand information to make adaptive and lookahead routing decisions.

IV. Design

In this section, we explain the design of Roracle in detail, including the GNN-based routing prediction model and the LP formulation for generating target routing. Inspired by the graph representation learning techniques [19], [20] and the attention mechanism [20], [30], [31], we design an encoder to produce embedding for each node in the network, and a decoder to interpret path split ratios from each node's embedding. This encoder-decoder architecture can be viewed as the core design of our GNN-based routing prediction model. Moreover, we formulate a multi-TMs routing optimization problem to

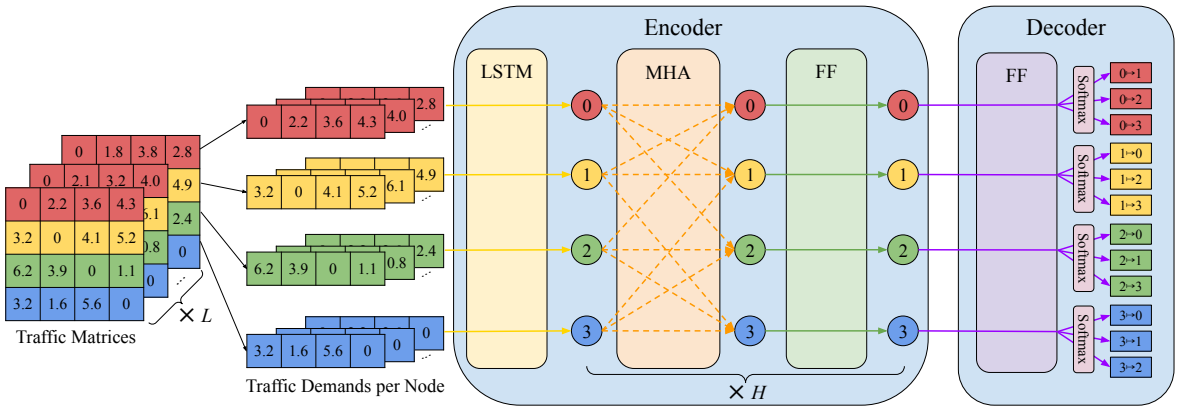


Fig. 4. The GNN architecture of Roracle. During the encoding process, the initial node embeddings are computed using a shared LSTM layer and then updated using H attention layers according to the topology connectivity matrix. After that, the decoder interprets the final node embedding as path split ratios for corresponding source-destination pairs with a node-wise fully connected FF layer and a pair-wise softmax layer.

facilitate path-based routing with reduced routing complexity, which is used to compute the target routing for future TMs during training and guide the learning process of the routing prediction model with SL.

A. Encoder

For each node $i \in \upsilon$, the input of encoder $x_i \in \mathbb{R}^{L \times N}$ is a series of $|\tau| = L$ demands $\{D_t^{i,d},\ d \in \upsilon,\ t \in \tau\}$ originated from node i, where υ is the node set ($|\upsilon| = N$). To capture complex temporal features among the series of demands, we apply a Long Short-Term Memory (LSTM) [32] layer to compute an initial d_h dimensional node embedding h_i^0 ($i \in \upsilon$), as shown in Fig. 4. We also provide spatial information by adding Positional Encoding (PE) to the initial node embedding h_i^0 to indicate the origin i of the series of demands $\{D_t^{i.d}\}$, where PE is calculated as a mix of sinusoidal functions [31]. Then, the embedding of each node is updated through message exchange with its neighboring nodes. The embedding update module consists of a stack of H identical attention layers [30], [31]. Each attention layer includes a Multi-Head Attention (MHA) layer that exchanges messages between neighboring nodes, and a node-wise fully connected Feed-Forward (FF) layer for nonlinear transformation. We also apply a skip connection [33] and layer normalization (LN) [34] to each sub-layer to facilitate training.

Single Dot-Product Attention. Given a node i, the node embedding h_i^l (layer $l \in \{1, ..., H\}$) is obtained by aggregating weighted messages from its neighboring nodes according to the attention mechanism [30], [31]. The attention weight of a message is determined by the compatibility function, i.e., the dot-product of the query $q_i^l \in \mathbb{R}^{d_k}$ of node i and the key $k_j^l \in \mathbb{R}^{d_k}$ of node j, where q_i^l and k_j^l are obtained by projecting the corresponding embedding vectors with parameter matrices $W_Q^l \in \mathbb{R}^{d_k \times d_h}$ and $W_K^l \in \mathbb{R}^{d_k \times d_h}$, respectively:

$$q_i^l = W_Q^l h_i^{l-1}, \quad k_j^l = W_K^l h_j^{l-1}. \tag{1}$$

The scaled compatibility $u_{i,j}^l \in \mathbb{R}$ is then calculated according to the following equation:

$$u_{i,j}^l = \begin{cases} (q_i^l)^\top k_j^l / \sqrt{d_k} & \text{if } i \text{ is adjacent to } j \\ -\infty & \text{otherwise.} \end{cases} \tag{2}$$

The compatibility of non-adjacent nodes is set to $-\infty$ to prevent message exchange between these nodes. Note that each node is self-adjacent to itself. According to the compatibility $u_{i,j}^l$, the attention weight $a_{i,j}^l$ is calculated using softmax:

$$a_{i,j}^l = \frac{e^{u_{i,j}^l}}{\sum_{n=0}^{N-1} e^{u_{i,n}^l}}. \tag{3}$$

Then, the node embedding h_j^{l-1} received by node i is projected with a parameter matrix $W_V^l \in \mathbb{R}^{d_v \times d_h}$ to obtain the value v_j:

$$v_j^l = W_V^l h_j^{l-1}. \tag{4}$$

As a result, the embedding vector h_i^l for node i is updated according to the following equation:

$$h_i^l = \sum_{n=0}^{N-1} a_{i,n}^l v_n^l. \tag{5}$$

Multi-Head Attention. To allow each node to jointly exchange multiple types of messages with its neighboring nodes [20], [30], [31], we employ multiple attention functions in parallel, i.e., calculate Eq. (5) for M times with different W_Q^l, W_K^l and W_V^l, and set $d_k = d_v = \frac{d_h}{M}$. For each attention head $m \in \{1, ..., M\}$, the corresponding embedding vector is updated according to the following equation:

$$h_i^{lm} = \sum_{n=0}^{N-1} a_{i,n}^{lm} v_n^{lm}. \tag{6}$$

Then, multiple h_i^{lm} are projected back to the original d_h dimensional vector h_i^l using parameter matrices $W_h^m \in \mathbb{R}^{d_h \times d_v}$:

$$h_i^l = \sum_{m=1}^{M} W_h^m h_i^{lm}. \tag{7}$$

After exchanging messages for H iterations, each node's final embedding h_i^H would include the information of the nodes that are H hops away. Therefore, H is set to the max hops between any two nodes in the network to ensure that each node can receive the information from the entire network.

B. Decoder

In Fig. 4, the decoder consists of a node-wise fully connected FF layer and a pair-wise softmax layer. It interprets each node's final embedding vector h_i^H ($i \in \{0, ..., N-1\}$) outputted by the encoder as path split ratios $\{\sigma_p^{i,d},\ p \in$

TABLE I
NETWORK TOPOLOGIES USED IN EVALUATION

Topology	Nodes	Links	TM Interval
Abilene	12	30	5 minutes
CERNET	14	32	5 minutes
GÉANT	23	72	15 minutes
Sprintlink	44	166	Synthetic TMs
BRAIN	161	332	1 minute
BRITE	204	964	Synthetic TMs

$\{0, ..., \kappa-1\}\}$ for the node pairs $\langle i, d\rangle$ ($d \in \{0, ..., N-1\}$) with source i. Given the input h_i^H, the output $o_i \in \mathbb{R}^{\kappa \times N}$ of the node-wise fully connected FF layer is given below:

$$o_i = W_O h_i^H + b_O, \tag{8}$$

where $W_O \in \mathbb{R}^{\kappa \times N \times d_h}$ and $b_O \in \mathbb{R}^{\kappa \times N}$. Then, a set of ($\kappa \times N$) values $\{\hat{o}_p^{i,d}\}$ ($d \in \{0, ..., N-1\}$, $i \neq d$, $p \in \{0, ..., \kappa-1\}$) is derived from o_i. Each $\hat{o}_p^{i,d}$ corresponds to a preconfigured path of a source-destination pair $\langle i, d\rangle$. Since $\langle i, i\rangle$ is not a valid pair, we do not derive $\hat{o}_p^{i,i}$. Note that there might be less than κ preconfigured paths for some pairs. Thus, we set $\hat{o}_p^{i,d} = -\infty$ if $p \notin P^{i,d}$. After applying the pair-wise softmax function, the final outputs of the decoder are given as follows:

$$\sigma_p^{i,d} = \frac{e^{\hat{o}_p^{i,d}}}{\sum_{p'=0}^{\kappa-1} e^{\hat{o}_{p'}^{i,d}}}. \tag{9}$$

Such outputs $\{\sigma_p^{i,d}\}$ can be viewed as the inferred routing decision from Roracle, i.e., the traffic split ratios for all source-destination pairs on its corresponding preconfigured paths.

C. Target Routing

For training purposes, we generate input features from L historical TMs in the training dataset and compute the optimal path-based routing for the next L' future TMs as the training target with low computation overhead (see Fig. 2). The routing optimization problem for target future TMs can be described as follows. Given a topology $G(\upsilon, \varepsilon)$ with nodes υ and directed links ε ($|\upsilon| = N, |\varepsilon| = E$), a set of preconfigured paths $\{P^{s,d}\}$ for each source-destination pair $\langle s, d\rangle$, and a sequence of $|\tau'| = L'$ target future TMs $\{\text{TM}_t,\ t \in \tau'\}$, our objective is to obtain the optimal path split ratios $\{\bar{\sigma}_p^{s,d},\ p \in P^{s,d}\}$ for each source-destination pair $\langle s, d\rangle$, such that the average load balancing performance among given TMs is maximized, i.e., the average MLU $\sum_{t\in\tau'} U_t/L'$ is minimized, where U_t is the MLU achieved on TM_t ($t \in \tau'$). We omit the constant $|\tau'| = L'$ for simplicity and formulate the above routing problem as an optimization problem as follows.

$$minimize \quad \sum_{t\in\tau'} U_t \tag{10a}$$

$$\text{subject to} \quad \bar{\sigma}_p^{s,d} \geq 0 \qquad p \in P^{s,d},\ s, d \in \upsilon,\ s \neq d \tag{10b}$$

$$\sum_{p\in P^{s,d}} \bar{\sigma}_p^{s,d} = 1 \qquad s, d \in \upsilon,\ s \neq d \tag{10c}$$

$$\sum_{\substack{s,d\in\upsilon \\ s\neq d}} \sum_{p\in P^{s,d}} \delta_{p,i,j}^{s,d} \cdot \bar{\sigma}_p^{s,d} \cdot D_t^{s,d} = l_{t,i,j} \quad \langle i, j\rangle \in \varepsilon,\ t \in \tau' \tag{10d}$$

$$l_{t,i,j} \leq c_{i,j} \cdot U_t \qquad \langle i, j\rangle \in \varepsilon,\ t \in \tau' \tag{10e}$$

(10b) and (10c) mention that the split ratios $\bar{\sigma}_p^{s,d}$ should be non-negative and sum up to 1 for each source-destination pair $\langle s, d\rangle$. (10d) represents the traffic load $l_{t,i,j}$ on link $\langle i, j\rangle$ under TM_t, where $D_t^{s,d}$ is the traffic demand from source s to destination d in TM_t, and $\delta_{p,i,j}^{s,d}$ is a binary indicator that equals 1 if link $\langle i, j\rangle$ belongs to the path p of the source-destination pair $\langle s, d\rangle$; otherwise, $\delta_{p,i,j}^{s,d} = 0$. (10e) is the link capacity utilization constraint for TM_t, where $c_{i,j}$ stands for the capacity of link $\langle i, j\rangle$. By solving problem (10) using LP solvers, we can obtain the set of optimal path split ratios $\{\bar{\sigma}_p^{s,d}\}$, which is the target routing that Roracle wants to learn.

V. IMPLEMENTATION

A. Training and Testing Setups

Several real-world network topologies are used in our evaluation and the statistics are listed in Table I. There are four networks with real measured TMs available, including the Abilene, CERNET, GÉANT, and BRAIN networks. For the Abilene network, the topology information and measured TMs are available at [35], which are collected every 5 minutes for 24 weeks. Similarly, the CERNET TMs are measured at 5-minute intervals for 5 weeks, where the network topology and real TMs are obtained from [36]. As for the GÉANT network, the link capacities and costs are provided by [37], and the measured TMs are available at [38], which are collected every 15 minutes for 17 weeks. For the large-scale BRAIN network with 161 nodes and 332 links, the topology connectivity and real TMs are provided in [22], [23], where the fine-grained BRAIN TMs are measured at 1-minute intervals for a total of 7 days. We assign equal link costs and configure link capacities based on the degrees of connected nodes [39]. If both degrees < 4, we only configure 5 Gbps; otherwise, 10 Gbps.

For each of these four networks, we select TMs from the first 80% of total weeks as our training/validation dataset $\mathbb{D}$, while the remaining 20% TMs are used as the test set for evaluating Roracle. As previously depicted in Fig. 2, a training sample is generated for every sequence of $L + L'$ TMs. The first L TMs is the input $X \in \mathbb{R}^{L\times N\times N}$ of a training sample, and the target $T \in \mathbb{R}^{\kappa\times N\times(N-1)}$ of a training sample is calculated based on the later L' TMs using LP (10). We randomly shuffle the training samples constructed from dataset $\mathbb{D}$ and select 80% of the samples as the training set $\mathbb{D}_T$, while the remaining 20% samples are considered as a standalone validation set $\mathbb{D}_V$, which is used for early stopping to avoid overfitting [40]. Roracle is trained on $\mathbb{D}_T$ with the objective of minimizing the Mean Absolute Error (MAE) loss between the predicted routing and the target routing using stochastic gradient descent. When training is done, we evaluate Roracle using the test set that is unseen during training. Given a sequence of L TMs, we predict a routing strategy and then apply it in the following L' TMs. We repeat the above procedure every L' TMs until the last TM of the test set. In the following experiments, we set $L = L' = 5$ for all networks.

To further evaluate the generalization capability of Roracle under different traffic variations, we introduce the Sprintlink network and a large-scale synthetic network topology generated from BRITE [41]. The statistics of these two networks are listed in Table I. The Sprintlink network is an Internet Service

TABLE II
PARAMETER SETTINGS FOR DYNAMIC AND STABLE TMS

Parameters	Dynamic	Stable
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

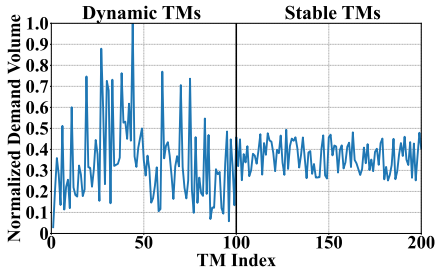


Fig. 5. Traffic variations of a source-destination pair in the Sprintlink network, whose demand volume is normalized with its maximum value in the test set.

Provider (ISP) network collected by Rocketfuel [42] with 44 nodes and 166 links. The large-scale BRITE network is generated as a 4×4 grid region-level topology with 204 nodes and 964 links, which is suitable for scalability analysis. Since the link cost settings are available for the Sprintlink network, we follow the default settings in Cisco routers to infer link capacities as the inverse of link costs. For the synthetic BRITE network, both the link costs and capacities are configured in a similar way as the BRAIN network. Given that real traffic traces are not available for these two networks, we use the Modulated Gravity Model (MGM) [43], [44] to generate a series of spatiotemporal TMs that simulate the characteristics of real-world traffic demands. Fig. 5 provides an example of traffic variations in the Sprintlink network. By controlling the MGM parameters in Table II, we generate 100 dynamic TMs with large variations and 100 stable TMs with small variations for both the training/validation set and the test set.

B. Performance Metric

In this paper, we aim to minimize the MLU to achieve good load balancing. Thus, we use the MLU ratio $\bar{U}$ to evaluate the performance of Roracle, which is defined as follows:

$$\bar{U}_{\text{Roracle}} = U_{\text{optimal}}/U_{\text{Roracle}}, \tag{11}$$

where U_{optimal} is the MLU achieved by an optimal routing that is obtained by solving the MCF problem using LP based on exact future TMs. A higher MLU ratio indicates that the performance of Roracle is closer to optimal performance.

C. Hyperparameters

For the routing prediction model, we set the embedding dimension d_h to 128 and the number of attention heads M to 8. For the FF sub-layer in each attention layer, the intermediate layer dimension is set to 256. For the decoder, the output dimension of the FF layer equals $N \times \kappa$, where N represents the number of network nodes and κ is the number of preconfigured paths ($\kappa = 4$). A constant learning rate $\alpha = 10^{-4}$ is used for training, and the batch size is set to 512 for a large training set and 64 for a small training set. Besides, we apply dropout [45] with a rate of $\rho_{drop} = 0.1$ and L2 regularization [46] with regularization parameter $\lambda = 0.001$ to avoid overfitting. All these hyperparameters are fixed throughout our experiments, which works well on different topologies and traffic traces.

VI. EVALUATION

A series of simulation experiments are conducted using real-world network topologies and traffic traces to evaluate the performance of Roracle and demonstrate its advantage by comparing it with the following baseline methods.

ACRNN [13]: Exploits an attention-based convolutional recurrent neural network model to predict a sequence of L' future TMs based on historically measured L TMs, and then solves the multi-TMs routing optimization problem (10) based on the predicted TMs, which is a state-of-the-art TE solution that considers lookahead routing and multiple TM prediction.

SMORE [6]: Generates a set of paths using an oblivious routing [17], [18] algorithm and solves a variation of the MCF problem using LP to obtain the optimal path split ratios for a given TM, which is a state-of-the-art centralized TE method that achieves good tradeoffs between routing optimization complexity and network performance.

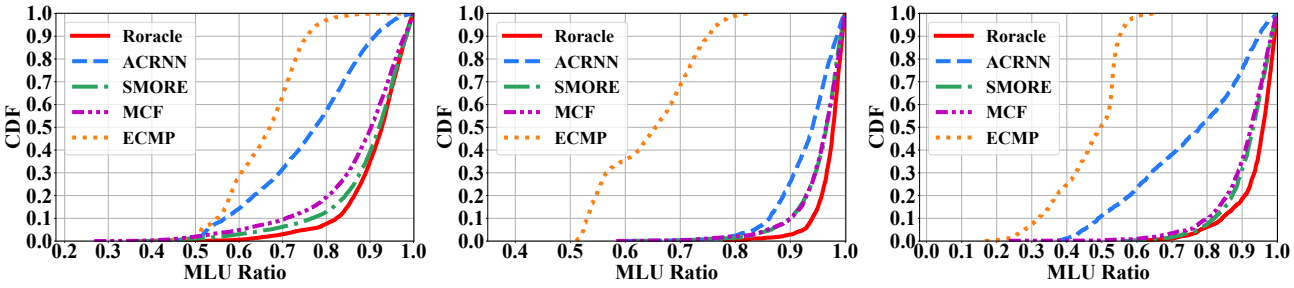
MCF [21]: Formulates an MCF problem with the objective of minimizing MLU to obtain fine-grained per-flow routing, which is a traditional TE solution that achieves optimal performance for a given TM with link-based routing.

Equal-Cost Multipath (ECMP) [47]: Distributes traffic evenly among available next hops along the shortest paths, which is widely adopted by network operators along with the OSPF routing protocol. The link cost setting for each network is shown in Section V-A.

For comparison, we also calculate the MLU ratio for the baseline methods. ACRNN optimizes routing based on a sequence of L' predicted TMs and then applies the routing on the sequence of L' real future TMs. SMORE and MCF optimize routing based on TM_t and then apply the routing on the subsequent TMs spanning from TM_{t+1} to $\text{TM}_{t+L'}$. Roracle, ACRNN, SMORE, and MCF have the same routing update frequency, while Roracle, ACRNN, and SMORE use the same preconfigured paths. ECMP is a static method that is not traffic-aware and does not require routing updates.

A. Performance Comparison in Future Traffic Scenarios

Fig. 6 illustrates the MLU ratio achieved by Roracle and the baseline methods in the test set of the first three small networks. From Fig. 6, we can observe that Roracle performs the best among all schemes. Since SMORE and MCF can achieve good performance in small networks under stable traffic scenarios, the average performance improvement of Roracle over SMORE and MCF is not very significant. However, the MLU ratios of SMORE and MCF could degrade dramatically when traffic fluctuates dynamically. To emphasize the comparison of MLU ratio in such extreme conditions, we present a box plot in Fig. 7 with customized whiskers and outliers to highlight the worst 5% scenarios. In the Abilene network, the worst-case MLU ratios of SMORE and MCF are only 36.6% and 26.8%, respectively, which leads



(a) Abilene network. (b) CERNET network. (c) GÉANT network.

Fig. 6. MLU ratio comparison in future unknown traffic scenarios of the three small networks. A higher MLU ratio indicates better performance.

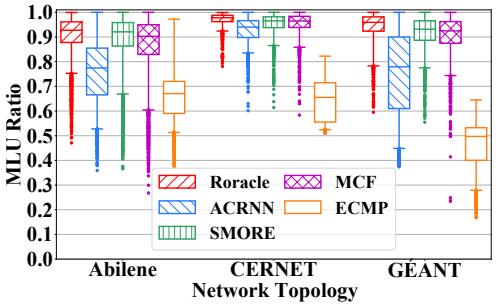


Fig. 7. MLU ratio comparison with an emphasis on worst-case performance, where the whiskers are customized as the highest & 5th percentile MLU ratio. We use outliers to reflect the MLU ratios of different TE solutions under extreme conditions (the worst 5% scenarios).

to severe network performance degradation. For the GÉANT network, MCF only achieves a 23.5% MLU ratio in the worst case, which is slightly better than ECMP. In contrast, Roracle can provide lookahead routing strategies with the best 5th percentile/worst-case MLU ratio in all three networks to prevent severe performance degradation. As shown in Fig. 7, Roracle achieves at least 92.4% MLU ratio for 95% of future TMs in the CERNET work with a worst-case performance of 78.1%, which demonstrates the strong generalization capability of Roracle to adapt to future traffic scenarios. Compared to traditional TE (e.g., SMORE and MCF), Roracle can achieve at most 14.9% and 36% performance improvements in terms of 5th percentile and worst-case MLU ratio, respectively.

As depicted in Figs. 6 and 7, the performance of ACRNN is far from optimal. While Roracle, SMORE, and MCF can achieve a median MLU ratio over 90%, ACRNN's median performance is less than 80% in the Abilene and GÉANT networks. Moreover, the worst-case performance of ACRNN could be worse than SMORE, even though SMORE is optimized for the current TM rather than future TMs. As the prediction sequence becomes longer (i.e., 5 future TMs), the TM prediction complexity would inevitably increase and lead to performance degradation. In contrast, Roracle performs consistently well in future unknown traffic scenarios. In the GÉANT network, Roracle outperforms ACRNN by 33.4% in terms of 5th percentile performance. This is because routing prediction has a main advantage over TM prediction: regardless of the length of the prediction sequence, the routing prediction complexity remains the same. Moreover, it turns out that routing prediction is more decision-focused compared to the two-step TM prediction + routing optimization approach. As a result, Roracle is able to provide a good routing strategy for a long sequence of future TMs with mitigated network disturbance and service disruption.

B. Generalization to Different Traffic Variations

Fig. 8 shows the MLU ratio comparison of different TE solutions on each future TM in the Sprintlink network, including dynamic and stable traffic scenarios. During dynamic traffic fluctuations, both SMORE and MCF experience severe performance degradation, which reflects the limitation of traditional TE solutions since they only rely on historically measured TM to make routing decisions. With the prediction capability, both Roracle and ACRNN can achieve good performance in dynamic future traffic scenarios and outperform traditional TE solutions. As for stable TMs, all TE solutions except ECMP can achieve high MLU ratios as we expected. One interesting observation is the slight performance degradation of Roracle and ACRNN compared to SMORE in stable TMs, which might be caused by prediction errors. Given that SMORE optimizes routing based on the current TM, it might work well when there are no significant future traffic variations. However, it is worth noting that Roracle can consistently achieve close-to-optimal performance in stable TMs and outperform SMORE in dynamic TMs with an average MLU ratio of 94.1% in all future traffic scenarios, which demonstrates the strong generalization capability of Roracle over different traffic variations.

C. Scalability Analysis in Large Networks

To validate the scalability of Roracle, we evaluate all TE solutions in two large-scale networks, including the BRAIN network (161 nodes, 332 links) with highly dynamic real traffic traces and the BRITE network (204 nodes, 964 links) with dynamic/stable TMs. For routing performance comparison, we assume that all TE solutions can compute an updated routing in time to catch up with traffic changes without taking computation overhead into account. In other words, the actual performance of the baseline methods could be even worse in realistic scenarios due to seriously delayed TE reactions. Since MCF requires more than 20 hours for routing optimization in the BRITE network, we omit it from the corresponding results.

Fig. 9 shows the comparison of different TE solutions in future traffic scenarios of the two large networks. Given the highly dynamic traffic patterns in the BRAIN network, the

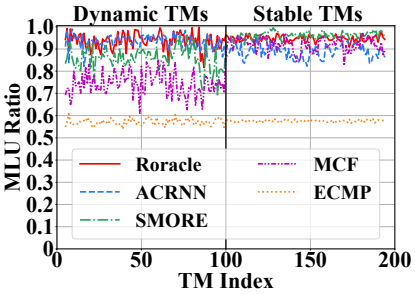
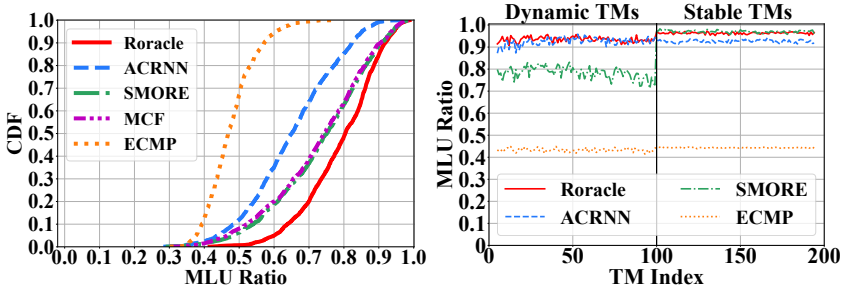


Fig. 8. MLU ratio comparison in the Sprintlink network with different traffic variations. The first 100 TMs represent dynamic scenarios, and the remaining 100 TMs are relatively stable.

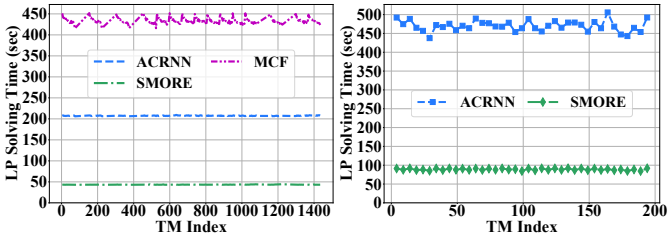


(a) BRAIN network (161 nodes, 332 links). (b) BRITE network (204 nodes, 964 links).

Fig. 9. MLU ratio comparison in the two large-scale networks. Due to scalability issues, MCF is omitted from the results of the BRITE network. Note that the impact of high computation overhead is not yet considered for performance comparison, which means the actual performance of the baseline methods could be even worse with a seriously delayed TE reaction.

performance of SMORE and MCF degrades dramatically with a worst-case MLU ratio of 28.9% and 30.3%, respectively. ACRNN also suffers from severe performance degradation with a 65.1% average MLU ratio. Compared to these baselines, Roracle can better accommodate future traffic fluctuations in the BRAIN network with up to 13.9% average performance improvement and 13.3% worst-case performance improvement. For the BRITE network, our observations are similar to the Sprintlink network in Section VI-B. As shown in Fig. 9(b), Roracle constantly achieves near-optimal performance under different traffic variations. It also outperforms SMORE and ACRNN in dynamic and stable future TMs, respectively.

In addition to routing performance comparison, it is essential to evaluate the computation overhead of different TE solutions for scalability analysis in large networks. Fig. 10 shows the LP solving time of the baseline methods in the two large-scale networks. Unfortunately, these baseline methods cannot compute an updated routing within a reasonable timescale during online deployment. In the BRAIN network, MCF requires an average of 7.2 minutes to optimize routing for a given TM, which cannot fit into regular 5-minute routing update intervals. Moreover, MCF cannot compute a routing for the BRITE network within 20 hours. As we explained in Section II-D, MCF needs to solve for a large number of $N \times (N-1) \times E$ routing variables with N nodes and E links, which introduces high routing computation complexity (considering E = 964 in the BRITE network). By leveraging κ preconfigured paths for each source-destination pair, SMORE and ACRNN only need to solve for $N \times (N-1) \times \kappa$ routing variables with reduced complexity (e.g., κ = 4), but it still requires an average of 43.4 seconds and 1.5 minutes for SMORE to compute a path-based routing strategy in the BRAIN and BRITE networks, respectively. Since ACRNN solves a path-based multi-TMs routing optimization problem (10) based on multiple predicted TMs, there are more constraints to be considered in the LP problem (10) compared to the single TM-based SMORE, which results in an average computation time of 3.5 minutes and 7.8 minutes for ACRNN in the BRAIN and BRITE networks, respectively. Such a high routing computation overhead will restrict timely routing updates of these baseline methods and lead to potential performance degradation. In contrast, Roracle can directly and quickly infer routing strategies in



(a) BRAIN network. (b) BRITE network.

Fig. 10. Comparison of LP solving time among different baseline methods in the two large-scale networks. In the BRITE network, MCF requires more than 20 hours for routing optimization, which is omitted from the comparison.

these two large networks to avoid delay in TE operations (see Section VI-D). Overall, Roracle achieves good scalability in large networks with promising routing performance and low computation overhead.

D. Overhead Analysis

In our evaluation, we use a Tesla V100 GPU to train Roracle in a high-performance computing cluster. The training time of Roracle depends on the size of the network topology and TM dataset. For the large-scale BRAIN network (161 nodes, 332 links) with thousands of 1-minute fine-grained TMs measured in one week, it takes 10.5 hours to train a Roracle model for routing prediction. For the remaining networks with smaller topology sizes or fewer training samples, we can train a Roracle model in less than one hour. It is worth noting that the above training time can be further reduced with a more powerful hardware specification, and we do not retrain the Roracle model since it can achieve good performance in all future traffic scenarios. In practice, network operators can consider retraining the model on a weekly basis or when traffic patterns significantly change.

Once the training is done, we conduct simulation experiments to evaluate the routing performance of Roracle and the baselines in future traffic scenarios. Considering the resource consumption of routing optimization caused by the baseline methods in large-scale networks, we take Gurobi [48] as the LP solver and test all TE solutions on a Linux server with a 10-core Intel 3.3 GHz CPU and 128 GB memory. Table III shows the average running time of Roracle and the baseline methods in all six networks (see Table I for topology size).

TABLE III
COMPUTATION OVERHEAD OF DIFFERENT TE SOLUTIONS

Topology	Roracle	ACRNN	SMORE	MCF	Target LP
Abilene	50.7 ms	300.9 ms	65.1 ms	142.8 ms	300.5 ms
CERNET	71.8 ms	478.1 ms	102.8 ms	254.4 ms	475.8 ms
GÉANT	162.3 ms	1.3 s	277.4 ms	1.5 s	1.3 s
Sprintlink	376.4 ms	2.5 s	547.3 ms	5.6 s	2.5 s
BRAIN	612.8 ms	3.5 min	43.4 s	7.2 min	3.5 min
BRITE	1.1 s	7.8 min	1.5 min	>20 h	7.8 min

For the first four small networks, the computation overhead of all TE solutions is negligible. However, as the network size increases to hundreds of nodes and links, we can observe a severe delay in TE operations of the baseline methods, which reveals their limitations on scalability. In contrast, Roracle can achieve efficient routing inference in the large-scale BRAIN and BRITE networks with an average running time of 618.2 ms and 1.1 s, respectively. Compared to the most efficient baseline SMORE, Roracle can achieve a speedup of at least 71× in these two large networks. Moreover, it is feasible to compute the target routing for Roracle's training with Multi-TMs routing optimization (10). As shown in Table III, it takes an average of 3.5 minutes and 7.8 minutes to solve the target LP (10) in the BRAIN and BRITE networks, respectively. By leveraging powerful machines in an offline setting, we can compute target routing strategies for multiple training samples in parallel to accelerate the procedure and then apply them for offline training. Thanks to the scalable GNN architecture design, Roracle can quickly generate a lookahead routing to accommodate future unknown traffic scenarios in today's large-scale networks with good scalability.

VII. RELATED WORK

In the early days, traditional TE solutions [3]–[5], [49] were designed based on Interior Gateway Protocols (IGPs). For instance, with OSPF [50] or IS-IS [51], routers can exchange link state information to learn about the network topology and then calculate the shortest paths to each destination according to the link costs. If there are multiple next hops for a destination, routers split traffic evenly among them according to the ECMP [47] split rule. These TE solutions aim to optimize link costs to achieve good network performance with a given TM, which has been proven to be NP-hard [4]. In [7], [8], Multi-Protocol Label Switching (MPLS) is used to support explicit routing that offers fine-grained traffic distribution control for each source-destination pair, and the explicit routes are obtained by solving an optimization problem with a given input TM. However, all the above methods assume that the traffic is stable and the routing optimized based on the input TM would be effective for a long period.

The emerging SDN techniques provide new opportunities for TE to improve network performance [28]. The control plane of SDN generates flexible routing policies based on its global view of the network and then installs these policies in the corresponding SDN switches in the network. TE solutions implemented in the control plane [2], [6], [52]–[54] obtain routing policies by formulating and solving a routing optimization problem with single or multiple given TM(s) and different constraints. For example, the scheme in [52] considers a network with partially deployed SDN switches to improve network utilization and reduce packet loss by strategically placing the controller and SDN switches. SOTE [53] focuses on minimizing the MLU in an SDN/OSPF hybrid network. SMORE [6] generates a set of paths using an oblivious routing [17], [18] algorithm and then dynamically adapts sending rates of all flows for each TM according to these preconfigured paths. Again, they make routing decisions based on the input TM(s) and have to passively react to traffic fluctuations by frequently updating the routing, which lacks the prediction capability to accommodate traffic changes in the future. Moreover, these TE solutions may suffer from scalability issues due to the high computation overhead of solving routing optimization problems in large networks.

TM prediction plays an important role in the realm of TE. In recent years, quite a few neural network-based TM prediction methods [12], [14]–[16] have been proposed. Given the high prediction complexity, most of the existing works focus on predicting the next single TM and are able to achieve state-of-the-art results. In these methods, TM prediction is typically modeled as a time-series prediction problem, which takes a series of past TMs as the input and predicts the following TM. By employing LSTM or Gated Recurrent Unit (GRU) [55] to capture the intra-flow dependencies, the methods proposed in [12], [15], [16] improve the accuracy of TM prediction. Recently, Gao et al. [13] propose an Attention-based Convolutional Recurrent Neural Network (ACRNN) to predict the next multiple TMs by capturing both intra-flow dependencies and inter-flow correlations in TMs. However, TM prediction-based TE does not necessarily lead to promising routing performance [13], [14] since the routing performance is prone to prediction errors. Besides, these methods cannot circumvent scalability issues since they need to perform routing optimization based on predicted TMs with high computation overhead.

VIII. CONCLUSION

To avoid performance degradation on future TMs and accelerate routing decisions in large networks, we propose Roracle, a scalable learning-based TE that predicts a good routing strategy for a long sequence of future TMs, while the learning is guided by the optimal solutions of LP problems using SL. Our customized scalable GNN architecture has greatly facilitated training and inference, which enables Roracle to quickly infer routing decisions in today's large-scale networks. Extensive experiments show that Roracle outperforms existing TE methods in future TMs with good scalability and mitigates network disturbance with infrequent routing updates.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China under Grant 62002019, SongShan Laboratory Fund under Grant YYJC022022009, and Zhejiang Lab Open Research Project under Grant K2022QA0AB02.

REFERENCES

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

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

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

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

[5] 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.

[6] P. Kumar *et al.*, "Semi-oblivious traffic engineering: The road not taken," in *USENIX NSDI'18*, pp. 157–170.

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

[8] Y. Wang and Z. Wang, "Explicit routing algorithms for internet traffic engineering," in *IEEE ICCCN'99*, pp. 582–588.

[9] R. Carpa, M. D. de AssunçÃo, O. Glück, L. LefÈvre, and J.-C. Mignot, "Evaluating the impact of sdn-induced frequent route changes on tcp flows," in *CNSM'17*, pp. 1–9.

[10] W. Reda *et al.*, "Path persistence in the cloud: A study of the effects of inter-region traffic engineering in a large cloud provider's network," *ACM SIGCOMM CCR*, vol. 50, no. 2, pp. 11–23, 2020.

[11] F. Abuzaid, S. Kandula, B. Arzani, I. Menache, M. Zaharia, and P. Bailis, "Contracting wide-area network topologies to solve flow problems quickly." in *USENIX NSDI'21*, pp. 175–200.

[12] A. Azzouni and G. Pujolle, "Neutm: A neural network-based framework for traffic matrix prediction in sdn," in *NOMS'18*, pp. 1–5.

[13] K. Gao *et al.*, "Incorporating intra-flow dependencies and inter-flow correlations for traffic matrix prediction," in *IEEE/ACM IWQoS'20*, pp. 1–10.

[14] Z. Liu, Z. Wang, X. Yin, X. Shi, Y. Guo, and Y. Tian, "Traffic matrix prediction based on deep learning for dynamic traffic engineering," in *IEEE ISCC'19*, pp. 1–7.

[15] S. Troia, R. Alvizu, Y. Zhou, G. Maier, and A. Pattavina, "Deep learning-based traffic prediction for network optimization," in *ICTON'18*, pp. 1–4.

[16] Q. Zhuo, Q. Li, H. Yan, and Y. Qi, "Long short-term memory neural network for network traffic prediction," in *ISKE'17*, pp. 1–6.

[17] H. Racke, "Minimizing congestion in general networks," in *IEEE SFCS'02*, pp. 43–52.

[18] H. Räcke, "Optimal hierarchical decompositions for congestion minimization in networks," in *ACM STOC'08*, p. 255–264.

[19] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2018.

[20] P. Veličković *et al.*, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2018.

[21] D. Mitra and K. G. Ramakrishnan, "A case study of multiservice, multipriority traffic engineering design for data networks," in *GLOBECOM'99*, vol. 1B, pp. 1077–1083.

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

[23] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *INOC'07*.

[24] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[25] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML'15*, pp. 1889–1897.

[26] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *ACM HotNets'17*, pp. 185–191.

[27] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM'18*, pp. 1871–1879.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[29] H. Xu, Z. Yu, C. Qian, X. Li, and Z. Liu, "Minimizing flow statistics collection cost of sdn using wildcard requests," in *IEEE INFOCOM'17*, pp. 1–9.

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

[31] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE/CVF CVPR'16*, pp. 770–778.

[34] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

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

[36] B. Zhang, J. Bi, J. Wu, and F. Baker, "Cte: Cost-effective intra-domain traffic engineering," *ACM SIGCOMM CCR*, vol. 44, no. 4, pp. 115–116, 2014.

[37] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM CCR*, vol. 36, no. 1, pp. 83–86, 2006.

[38] GÉANT. The TOTEM project. [Online]. Available: https://totem.info.ucl.ac.be/dataset.html

[39] M. Ye, J. Zhang, Z. Guo, and H. J. Chao, "Federated traffic engineering with supervised learning in multi-region networks," in *IEEE ICNP'21*, pp. 1–12.

[40] L. Prechelt, "Early stopping — but when?" in *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 53–67.

[41] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *MASCOTS'01*, pp. 346–353.

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

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

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

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[46] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *NIPS'91*, p. 950–957.

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

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

[49] Ashwin Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current ospf/is-is networks," in *IEEE INFOCOM'03*, vol. 2, pp. 1167–1177.

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

[51] D. Oran, "OSI IS-IS Intra-domain Routing Protocol," *IETF RFC 1142*, February 1990.

[52] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *IEEE INFOCOM'13*, pp. 2211–2219.

[53] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in sdn/ospf hybrid network," in *IEEE ICNP'14*, pp. 563–568.

[54] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Dynamic hybrid routing: Achieve load balancing for changing traffic demands," in *IEEE IWQoS'14*, pp. 105–110.

[55] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.