Adaptive and Low-cost Traffic Engineering: A Traffic Matrix Clustering Perspective

Yi Liu, Nan Geng, Mingwei Xu, Yuan Yang, Enhuan Dong, Chenyi Liu, Qiaoyin Gan, Qing Li, Jianping Wu

***Abstract*—Traffic engineering (TE) has attracted extensive attention over the years. Operators expect to design a TE scheme that accommodates traffic dynamics well and achieves good TE performance with little overhead. Some approaches like oblivious routing compute an optimal static routing based on a large traffic matrix (TM) range, which usually leads to much performance loss. Many approaches compute routing solutions based on one or a few representative TMs obtained from observed historical TMs. However, they may suffer from performance degradation for unexpected TMs and usually induce much overhead of system operating. In this paper, we propose ALTE, an adaptive and low-cost TE scheme based on TM classification. We develop a novel clustering algorithm to properly group a set of historical TMs into several clusters and compute a candidate routing solution for each TM cluster. A machine learning classifier is trained to infer the proper candidate routing solution online based on the features extracted from some easily measured statistics. We implement a system prototype of ALTE and do extensive simulations and experiments using both real and synthetic traffic traces. The results show that ALTE achieves near-optimal performance for dynamic traffic and introduces little overhead of routing updates.**

***Index Terms*—Traffic engineering, traffic matrix classification, machine learning.**

I. Introduction

Traffic engineering (TE) is essential to networks, especially with the wide deployment of technologies such as cloud, IoT, 5G, etc. Extensive efforts have been made to route dynamic traffic properly to improve resource utilization and network performance [2][3][4][5][6][7]. Operators expect that good TE performance can be achieved, e.g., maximizing the throughput or minimizing the maximum link utilization ratio, with little overhead while traffic dynamics, including bursts, can be accommodated.

It is challenging for TE to satisfy both adaptiveness and low overhead. Typically, optimal routing solutions are computed based on traffic matrices (TMs). However, computing the optimal routing solution is time-consuming, especially for large networks, even though the problem can be solved in polynomial time theoretically [8]. Researchers make efforts to compute routing solutions efficiently while preserving good TE performance, leveraging oblivious routing [2][3][9][10], pre-computed paths[4], representative TMs [5][11], etc. Some recent approaches also leverage machine learning (ML) techniques [6][7] to learn from historical TMs and infer the proper routing for upcoming traffic. Most existing approaches achieve good TE performance if the upcoming TMs have similar traffic characteristics but may face significant performance degradation for certain traffic patterns. One possible remedy is to monitor network status frequently and update the routing adaptively for unexpected TMs in time. Nevertheless, such a means inevitably induces much overhead (e.g., network measurement and routing computation) and impacts the quality of service (QoS) (e.g., jitter due to frequent routing updates) [12]. Besides, obtaining TMs in real time is unaffordable, especially in a high frequency [13], so approaches based on TM measurement still suffer from high overhead.

To enable adaptive and low-cost TE, we take a different method in this paper. We propose to divide all possible TMs into clusters (categories), each of which includes TMs sharing similar characteristics. Then, we compute one proper routing solution for all TMs belonging to the same TM category and achieve a good performance. These routing solutions are called candidate routing solutions, and we can achieve adaptive TE by selecting the best candidate routing solution. Such an online routing decision can be made by inferring the category of the currently observed TM quickly. Since the candidate routing solutions are pre-computed and a different routing will be enabled only when the TM category changes, the operating overhead is small, and frequent routing updates can be avoided.

Such an idea follows the main motivation behind oblivious routing, which is to cope with uncertainty in estimated TM that is likely to lie in some subset, and the TM space can be partitioned into multiple subsets and one routing strategy

Manuscript received XXX; accepted XXX. Date of publication XXX; date of current version XXX. A previous version of this work [1] has been presented in IEEE ICCCN 2020. This work was supported by the National Key Research and Development Program of China under Grant 2022YFB2901300, the National Natural Science Foundation of China under Grant 62221003 and 62132004. (*Corresponding author: Yuan Yang, Mingwei Xu.*)

Yi Liu, Nan Geng, Mingwei Xu, Yuan Yang, Chenyi Liu and Jianping Wu are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. Mingwei Xu and Jianping Wu are also with the Institute for Network Science and Cyberspace, Tsinghua University, Beijing, China (e-mail: liuy22@mails.tsinghua.edu.cn; nan_geng@sina.com; {xumw, yangyuan_thu}@tsinghua.edu.cn; liucheny19@mails.tsinghua.edu.cn; jianping@cernet.edu.cn).

Qiaoyin Gan is with Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China (email: ganqiaoyin23s@ict.ac.cn).

Enhuan Dong is with the Institute for Network Science and Cyberspace, Tsinghua University, Beijing, China (email: dongenhuan@tsinghua.edu.cn).

Qing Li is with the Peng Cheng Laboratory, Shenzhen, China (email: liq@pcl.ac.cn).

can be selected for each subset. Nevertheless, there are few studies on how to partition the TM space to the best of our knowledge. One relevant approach is described in [14], which treats the TM space as a Euclidean space and partitions the TM space into subsets orthogonally. However, such a method fails to encapsulate the fundamental characteristics of the TM space. TMs belonging to the same subset do not necessarily have similar TE performance under the routing computed for this cluster, and TMs that have optimal or near-optimal TE performance under the same routing may be partitioned into different subsets by the method in [14], for example, TMs D and λD. The essence of the TM clustering-based TE remains unrevealed.

To fill the gap, we first formalize the TM clustering-based TE problem. We observe that the TMs in the same cluster should have near-optimal TE performance under the routing computed for this cluster. Thus, we transform the performance difference between two TMs to define the “distance” between the TMs. TMs with small “distances” from each other should belong to the same cluster, and we can cluster the historical TMs based on TM distances among them. To reduce the overhead of computing TM distances, we further propose to use a neural network to estimate TM distances efficiently and with high accuracy. With properly selected TM categories, candidate routings can be optimized, so these routings have better tolerance of unexpected dynamic traffic than approaches that use only a few representative TMs [3][4][11].

We propose ALTE, an Adaptive and Low-cost TE scheme based on TM classification. First, we propose a novel clustering algorithm to efficiently and stably divide historical TMs into TM clusters. In particular, ALTE finds a fixed number of center TMs, so that the maximum distance between a TM and the nearest center TM is minimized. Second, we compute the suitable routing for each TM cluster by solving a modified oblivious routing problem. Then, we must overcome the challenge of quickly and efficiently inferring the TM category to decide the best one from candidate routing solutions in an online manner. We use traffic statistics that can be easily obtained to classify current TM and infer the best routing solution online to avoid obtaining TMs in real time, which is costly. We leverage a supervised learning algorithm, AdaBoost [15], to map such traffic statistics to the TM category. We implement the ALTE prototype system based on Ryu [16] and OpenFlow [17]. We do extensive simulations and experiments based on real topologies. Evaluation results show that ALTE outperforms other approaches greatly and achieves near-optimal TE performance for both real and synthetic TMs with little cost of routing updates.

The rest of the paper is organized as follows. Section II introduces the related works. Section III describes the overview of the ALTE approach. Then, we formalize the TM clustering-based TE problem and present the ALTE algorithms in Sections IV and V, respectively. Results of simulations and experiments are shown respectively in Section VI and Section VII, and the conclusion is summarized in Section VIII.

II. Related Work

In this section, we briefly introduce existing Traffic Engineering (TE) approaches by categorizing them into optimization model-based routing and Machine Learning (ML)-based routing.

Optimization model-based routing usually computes routing solutions using optimization methods. Some approaches focus on optimizing TE with respect to one single TM. Xu et al. [8] propose to transform the optimal TE problem into the shortest paths in terms of a set of non-negative link weights. Parham et al. [18] jointly optimize link weights and waypoints under the segment routing paradigm. To improve the adaptiveness to real traffic demands, some other approaches rely on more TMs. For instance, Leconte et al. [4] measure the real-time TM at each TE interval. The routing is optimized specifically for the measured TM by solving a mathematical optimization problem. Kumar et al. [3] propose to obtain optimal routing solutions for predicted TMs. Zhang et al. [11], Casas et al. [19], and Zhang et al. [5] focus on a set of TMs that represent current TM trends. However, measuring real-time TMs as the input of the optimization model burdens network management. Also, computing and enabling new routing solutions at each TE interval may induce some unnecessary overhead and impact QoS. Applegate et al. [2] propose oblivious routing (OR). A static routing is optimized for a wide range of TMs by solving linear programming. OR avoids routing updates so the cost is very low, while it may lose much TE performance in practice. Rétvári et al. [14] propose to partition the TM space into subsets heuristically based on computing an OR solution for each subset. As discussed above, the method does not capture the essence of the TM space. Furthermore, the computation overhead is unaffordable because linear programming problems for OR need to be solved several times for each dimension of a TM, while a TM has $O(n^2)$ dimensions for a network with n nodes. Thus, the method can only be used in very small topologies, with TMs that have only a few flows.

Our proposed scheme, ALTE, uses an optimization model to define TM distances and computes candidate routing solutions by solving a modified OR model. However, ALTE needs neither online TM measurement nor online solving optimization models, so little overhead is introduced. On the other hand, we use a set of candidate routing solutions to achieve adaptiveness to traffic dynamics, further differentiating it from most existing methods. Different from the TM space partitioning method proposed by [14], our scheme introduces TM distance and contributes a much more effective and efficient TM clustering algorithm. In fact, ALTE can be considered to be a generalization of OR.

Recently, ML techniques have garnered increasing attention for routing computation in TE. Some approaches [7][20][21] leverage reinforcement learning (RL) or deep reinforcement learning (DRL) to learn the mapping from the real-time TM to a routing solution. Valadarsky et al. [6] propose a DRL-based TE scheme that computes routing solutions with a continuous sequence of historical TMs as the inputs. Cong et al. [22] propose both centralized and distributed routing schemes based on DRL, and find that the centralized scheme is suitable for dynamic networks while the distributed one is better for coping with large-scale networks. Liu et al. [23] propose DRL-OR, which utilizes multi-agent reinforcement learning to generate hop-by-hop routing decisions in a distributed fashion to satisfy multiple QoS requirements. Zhang et al. [24] propose CFR-RL, which learns RL policies to select critical flows to reroute. Singh et al. [25] propose Trailnet, which leverages DRL to predict an output port of a router based on the destination IP address. Ye et al. [26] propose to use supervised learning to generate optimal inter-region TMs. There are also approaches [27][28][29][30] which compute routing solutions with other learning techniques such as deep belief network (DBN) and graph neural network (GNN), etc. Our scheme leverages ML in a different manner compared with existing studies. Instead of computing routing solutions directly, we use ML to estimate TM distances and choose the best routing solution from several computed candidates by inferring TM categories. This approach offers adaptiveness to traffic dynamics without introducing substantial overhead.

In a recent study, Ahuja et al. [31] propose to choose a set of dominating TMs which cover the main characteristics of the network topology and are used to solve a network planning problem efficiently. To this end, TM similarity is defined, and the TMs which share the least similarity are computed as the dominating TMs. In contrast, our scheme defines TM distance in a way that considers routing performance, with the objective of minimizing the distances from a set of center TMs to other TMs, rather than identifying TMs with the largest distance.

III. Overview

In general, *demand* is defined as an aggregation of traffic that enters the network from an ingress node and leaves the network from an egress node. The set of all the demands in the network is called TM. A routing solution of TE decides how to deliver the traffic of TM through the network properly for a TE objective. In this paper, we consider the objective of minimizing the maximum link utilization ratio (MLU) which is a typical and common TE objective [3][4][6]. Generally, a routing solution includes: 1) the forwarding paths carrying each demand from the ingress node to the egress node, and 2) the splitting ratios of each demand over these paths. For brevity, we use "routing" to represent "routing solution".

ALTE is an adaptive TE scheme that periodically adopts the best among several candidate routings. In particular, time is divided into small TE intervals. At each TE interval, a controller collects traffic statistics, i.e., the total traffic entering/leaving each ingress/egress node, which can be obtained easily by reading the port counts of routers. Then, an ML classifier is used to select the best routing solution from the

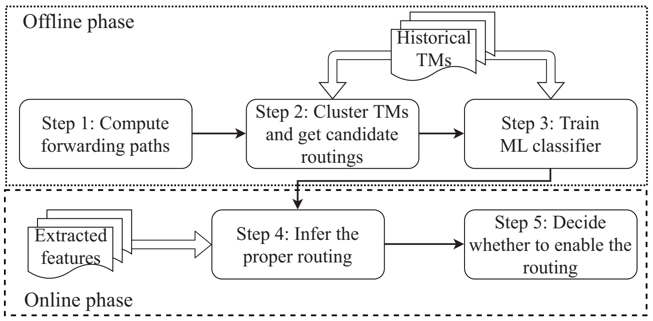

Offline phase
Historical TMs
Step 1: Compute forwarding paths
Step 2: Cluster TMs and get candidate routings
Step 3: Train ML classifier
Extracted features
Step 4: Infer the proper routing
Step 5: Decide whether to enable the routing
Online phase

Fig. 1: The workflow of ALTE.

set of candidates. In the next interval, ALTE would change the routing solution used only if the classification result differs from the current one. In other words, ALTE monitors network traffic at each TE interval, but the routing is changed only when needed. Therefore, ALTE can efficiently perceive TM category changes with a relatively small interval and without worrying about route oscillation. Meanwhile, frequent but low-cost traffic perceiving enables ALTE to accommodate unexpected traffic changes quickly.

ALTE aims to minimize the MLU for various TMs. To achieve good TE performance online, some preparatory steps should be taken carefully in the offline phase. Fig. 1 shows the workflow of ALTE. In the offline phase, we properly cluster historical TMs into several categories and obtain a candidate routing solution fitting each TM category. We develop the ML classifier to infer the proper routing solution (also the TM category) in the online phase. We use traffic features that can be easily measured for classification. We use labeled data to train our ML classifier in the offline phase. The offline phase takes place periodically to adapt to new traffic trends.

IV. TM Clustering-Based TE Problem

A. Formulation

We gradually define our problem, starting from the classic multi-commodity flow-based optimal routing problem, which considers a certain TM. We extend the problem to the optimal routing for a set of TMs. Then, we present the TM clustering-based TE problem.

1) Optimal routing for a TM: Formally, a network can be modeled as an undirected graph $G(V, E)$ with node set V and link set E. Each traffic demand (s, t) with demand size d_{st} enters the network from ingress node $s \in V$ and leaves from egress node $t \in V$. Let r_l^{st} (r_{ij}^{st}) be the fraction of traffic demand (s, t) that traverses link $l \in E$ ($(i, j) \in E$). A valid routing solution is the collection of r_l^{st} that satisfies routing consistency. In particular, we have

$$\sum_{\text{link}(s,j)\in E} r_{sj}^{st} = 1, \sum_{\text{link}(i,t)\in E} r_{it}^{st} = 1, \tag{1}$$

for any traffic demand (s, t); and for any node $k \in V$ that satisfies $k \neq s, t$, we have

$$\sum_{\text{link}(i,k)\in E} r_{ik}^{st} = \sum_{\text{link}(k,j)\in E} r_{kj}^{st}. \tag{2}$$

We say that a routing solution is valid if constraints (1) and (2) are satisfied. Note that the definition is independent of traffic demands. We organize a routing solution into matrix R, which includes $|E|$ rows corresponding to the links, and each row has $|V| \times |V|$ elements corresponding to the traffic demands.

Let $\vec{D}$ be the vector that consists of all elements in the traffic matrix $D = (d_{st})$. $R \times \vec{D}$ is then the vector that consists of traffic amounts on all links. Let c_l be the capacity of link $l \in E$, and $\vec{C} = (c_l)$ be the vector that consists of capacities of all $|E|$ links. For given TM D, the optimal routing solution R_D minimizes the maximum link utilization ratio (MLU). R_D can be obtained by solving the well-known multi-commodity flow model (P1).

$$\min \ u \tag{P1}$$

$$\begin{aligned} \text{s.t.} \quad & R_D \text{ is valid}, && (3) \\ & R_D \times \vec{D} \leqslant u\vec{C}. && (4) \end{aligned}$$

Constraint (4) means that the traffic amount on each link cannot exceed MLU u. It is well-known that P1 is a linear programming problem and can be solved in polynomial time.

2) Optimal routing for a set of TMs: With the basic model in mind, we define the optimal routing solution for a set of TMs. Let $\boldsymbol{D}$ be a TM set. For $D \in \boldsymbol{D}$ and routing solution R, let u_D^R be the MLU when R is used for D. Let u_D^0 be the optimal (minimal) MLU with traffic matrix D as the input. u_D^R/u_D^0 is the *performance ratio* of routing solution R with respect to D. If we scale the TM to $\frac{1}{u_D^0}D$, the optimal routing solution does not change, and the minimal MLU becomes 1, so the performance ratio of routing solution R equals MLU u_D^R. The optimal routing solution $R_{\boldsymbol{D}}$ is a routing solution that minimizes the maximum performance ratio over all $D \in \boldsymbol{D}$. Formally, $R_{\boldsymbol{D}}$ can be obtained by solving P2.

$$\min \ u \tag{P2}$$

$$\begin{aligned} \text{s.t.} \quad & R_{\boldsymbol{D}} \text{ is valid}, && (5) \\ & \forall D \in \boldsymbol{D} : R_{\boldsymbol{D}} \times \frac{1}{u_D^0}\vec{D} \leqslant uC. && (6) \end{aligned}$$

P2 is a general form of P1 because P1 has only one TM in $\boldsymbol{D}$. It is clear that P2 is also a linear programming problem.

Discussion: Note that we scale each TM D to $\frac{1}{u_D^0}D$ and minimize the performance ratio in P2, instead of directly minimizing the MLU for a set of TMs. This is because our target is to compute routings that minimize the MLUs under both existing TMs and upcoming TMs. Our definition of P2 is consistent with the oblivious routing model [2] if the TM set is continuous.

3) TM Clustering-Based TE: We intend to partition $\boldsymbol{D}$ into K TM clusters $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_K$ for given integer K and compute K routing solutions $R_1, R_2, \ldots, R_K$, s.t. each TM cluster uses the corresponding routing solution, and the maximum performance ratio is minimized. The problem is formulated as P3.

$$\min \ u \tag{P3}$$

$$\begin{aligned} \text{s.t.} \quad & R_1, R_2, \ldots, R_K \text{ are valid}, && (7) \\ & \forall D \in \boldsymbol{D}_i (0 \leqslant i \leqslant K) : R_i \times \frac{1}{u_D^0}\vec{D} \leqslant uC, && (8) \\ & \forall 1 \leqslant i, j \leqslant K \text{ and } i \neq j : \boldsymbol{D}_i \cap \boldsymbol{D}_j = \phi, && (9) \\ & \boldsymbol{D}_1 \cup \boldsymbol{D}_2 \cup \cdots \cup \boldsymbol{D}_K = \boldsymbol{D}. && (10) \end{aligned}$$

Note that optimal routing solutions $R_1, R_2, \ldots, R_K$ can be obtained by solving P2 K times if $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_K$ are given. In fact, the difficulty of P3 mainly lies in partitioning $\boldsymbol{D}$ into K clusters.

B. Modeling TM Clustering

We transform the TM set partitioning problem into a TM clustering problem by specifying the "distance" between two TMs. Recall that given TM set $\boldsymbol{D}$ in P2, we optimize the MLU of each TM D under the optimal routing solution $R_{\boldsymbol{D}}$, divided by the optimal MLU of D, i.e., u_D^0, as shown in Eq. (6). Note that $R_{\boldsymbol{D}}$ can also be the optimal routing solution of some TM D', which may be in $\boldsymbol{D}$ or not. In other words, P2 can be considered as finding TM D', to which each TM in $\boldsymbol{D}$ has the minimum distance. This naturally defines the distance from TM D to TM D'. The same observation can be made when we consider P3.

Formally, the distance from TM D to TM D', denoted by dis(D, D'), is the MLU of D under the optimal routing solution of D', divided by the optimal MLU of D. We have

$$\text{dis}(D, D') = \frac{u_D^{R_{D'}}}{u_D^0}. \tag{11}$$

We can obtain the distance between any pair of TMs according to the distance definition. In particular, we first solve a multi-commodity flow (MCF) problem for the optimal routing solution of D' and obtain $R_{D'}$. We apply $R_{D'}$ on TM D and compute the MLU, i.e $u_D^{R_{D'}}$. Then, we solve another MCF problem for the optimal routing solution of D and obtain u_D^0. Finally the distance from TM D to TM D', dis(D, D'), can be figured out from (11). Clearly, dis(D, D') is not less than 1 because $R_{D'}$ may be unoptimal for D. A smaller dis(D, D') indicates that D and D' share more similarities with regard to routing performance because the optimal routing solution of D' performs well on TM D.

We emphasize that the definition of distance is not restricted to Maximum Link Utilization (MLU). In fact, the proposed distance definition can be naturally extended to other practical performance metrics, such as end-to-end delays. Formally, we define R_D as a routing solution that optimizes $f(\cdot, D)$ under a certain TM D, where $f(R, D)$ is a performance metric function of routing solution R and TM D. We have

$$\text{dis}(D, D') = \frac{f(R_{D'}, D)}{f(R_D, D)}. \tag{12}$$

This adaptability demonstrates the versatility of our approach in accommodating various network performance objectives and addressing diverse optimization requirements.

C. Estimating TM Distances

The TM distance can be measured by solving MCF problems as discussed above. An MCF problem can be solved in polynomial time by linear programming theoretically, but in practice, the time cost increases significantly when considering large topologies, because the linear programming problem becomes massive and solving them is computationally expensive. Since our TM clustering-based TE strongly relies on TM distances, it is necessary to estimate TM distances by an efficient means.

To address the issue, we observe that given the network topology, a TM distance is a mapping from two TMs to a real number greater than or equal to 1. In other words, it is not necessary to compute the optimal routing solution if the mapping can be figured out. Note that the solution space of an MCF problem is a polytope [32]. Thus, according to Eq. (11), we can infer that the solution space of the TM distance mapping is also a polytope, which can be well estimated by a neural network (NN) efficiently. This method avoids the time-consuming process of solving linear programming problems.

1) Neural Network Structure: We develop the neural network to estimate the TM distances. The input of the neural network is a vector that consists of the concatenation of two flattened TMs, i.e., $\vec{D}$ and $\vec{D}'$. The output is the estimated distance $\mathrm{dis}(D, D')$. The neural network consists of multiple fully connected layers. The input features of an intermediate layer are the output features of the previous layer. The activation function used for intermediate layers is Rectified Linear Unit (ReLU) [33].

The input and output nodes of a neural network are determined by the target network topology. In particular, for a network topology with n nodes, the neural network has $2n(n-1)$ input nodes and precisely one output node. As for the intermediate layers, we empirically design the structure for each target network topology. In particular, for a topology with n nodes, the first intermediate layer has $\lfloor n(n-1) \rfloor$ nodes, the second layer has $\lfloor n(n-1)/2 \rfloor$ nodes, the k-th layer has $\lfloor n(n-1)/2^{k-1} \rfloor$ nodes, and so on. When the number falls below a certain threshold of 30, it is set to 30, and these nodes are directly connected to the output layer. For instance, we use a neural network with layers of 220, 110, 55, 30, and 1 nodes for the Abilene topology [34], with 11 nodes.

We also try neural networks with different intermediate layers. Table I summarizes some results of the minimum mean squared error (MSE) between the ground truth and the estimated TM distances on Abilene. We find that the estimation accuracy does not change much with different neural network structures. This implies that the solution space of the TM distance mapping is not so complicated and can be well captured by an extensive range of neural networks.

Note that this simple NN cannot generalize to other networks, because it does not take graph information as input, and the input size is fixed. It is an open and interesting question to design NNs to estimate TM distances with good generalizability, e.g., by using GNN. In this paper, we focus on the efficiency of solving the TE problem, and a simple NN can best meet our requirement because the solution space polytope is simple. We show below the accuracy and time overhead of our TM distance estimation.

2) Estimation Accuracy and Time Overhead: We build neural networks using PyTorch [35], a well-known library for deep learning. The objective of training is to minimize the minimum mean squared error (MSE). We employ the early stopping rule [36] to avoid overfitting and record MSE before the training procedure eventually stops.

We use four topologies (Abilene, GEANT, NSFNET and Colt) with real and synthetic TMs to show the effectiveness of neural networks. Details about the topologies and TMs are given later in Section VI, and we omit them here for the sake of simplicity. The TMs for a topology are split into a training set that has 20% of the TMs and a testing set that has 80% of the TMs. We also enlarge the training set to 80% of the TMs to compare the accuracy. We compute the TM distance between each TM pair according to Eq. (11). We use these TM distances from the training set to train our neural networks for each topology before testing.

Fig. 2 shows the CDF of the absolute error. We can observe that the absolute error is smaller than 0.1 for more than 90% of TM pairs, and the maximum absolute error is about 0.15 for NSFNET. These results show that our neural network can estimate TM distances with small errors. Note that we utilize a small proportion of TMs (20%) as training sets but still achieve relatively high accuracy, compared with using 80% of TMs for training. The training time is 2min31s, 2min8s, 1min1s, and 12min4s for Abilene (20% of 2,016 TMs), GEANT (20% of 672 TMs), NSFNET (20% of 1,400 TMs), and Colt (20% of 1,400 TMs), respectively. The time to solve TM distances for training datasets is 52s, 2min30s, 1min52s and 2.9h, respectively.

Discussion: Labeling of training datasets for TM distance estimation requires the solution of linear programming, which could be a little computationally expensive but necessary. However, compared to computing the distances between all TM pairs, NN-based TM distance estimation greatly reduces the complexity by labeling only a small number of TMs. Other TM distances are estimated by NNs in a much more efficient way, and this is just the advantage of employing NNs. Furthermore, network operators can run the entire TE scheme regularly, so the amortized offline cost would be acceptable.

Fig. 3 compares the average time needed to compute and estimate one TM distance. We observe that for Abilene and NSFNET, the average times of the two methods are similar to each other. However, for the Colt topology with a larger scale, computing one TM distance needs 37,800 ms on average, and the neural network needs only 565 ms, a 98.5% reduction. Actually, the computation time of estimating a TM distance does not change much for different topologies. This is because the neural network leverages deterministic mathematical operations, and does not need to solve MCF problems. As a result, our neural networks can accelerate

TABLE I: Minimum MSE under different network structures.

Intermediate layer size	(55)	(110,55)	**(110,55,30)**	(110,55,110)	(110,110,110)	(110,55,110,55)	(110,55,110,220)
Minimum MSE	0.0183	0.0174	**0.0134**	0.0144	0.0144	0.0135	0.0136

the process of computing TM distances greatly on large topologies, with high accuracy.

V. Algorithms

We develop algorithms for TM clustering-based TE. We first develop a clustering algorithm in Section V.A to partition historical TMs into proper clusters $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_K$, based on TM distance and estimated TM distance defined above. Second, we compute optimal routing solutions $R_1, R_2, \ldots, R_K$ for the TM clusters efficiently in Section V.B. Then, we present how to select the best routing solution online without measuring exact TM, i.e. online TM classification. Section V.C shows the features and the classification algorithm used.

A. TM Clustering

Clustering is a well-studied topic and there are a number of advanced algorithms. However, we still need to develop our TM cluster-based TE carefully from the routing point of view. Recall that for a TM cluster $\boldsymbol{D}$, there is a "center" TM D', to which each TM in $\boldsymbol{D}$ has the minimum distance, and D' may be in $\boldsymbol{D}$ or not. Although we can compute the optimal routing solution without being aware of D' by solving P2, the process is excessively time-consuming for large networks when $\boldsymbol{D}$ includes many TMs. Thus, we select one TM from each TM cluster $\boldsymbol{D}$ as the center TM and compute the routing solution by leveraging this center TM. So we need to find K center TMs to minimize the maximum distance between any TM and the nearest center TM. The process is similar to the partition-based clustering [37].

We tried to develop an exploration-based algorithm according to K-Medoids [37]. The main idea is to try a set of center TMs at each iteration and store the center TM set which gets a better objective value. The algorithm stops after a given maximum iteration number. However, we find this algorithm inefficient, and it is not easy to get a good result and bad results may be obtained sometimes, because the search space is large. To address the issue, we observe that the clustering process will be accelerated greatly if the K-clustering problem can be derived from the (K-1)-clustering problem. In particular, suppose some feasible solutions of the (K-1)-clustering problem have been obtained, and each solution contains (K-1) center TMs. An approximate solution of the K-clustering problem can be obtained by appending one more center TM to the most appropriate feasible solution. Inspired by the idea, we propose a clustering algorithm that divides the original problem into several subproblems. Each subproblem can also be divided further. By solving subproblems firstly, we can tackle their parent problems easily.

Assume there are N historical TMs. We index the TMs (i.e., $D_1, D_2, ..., D_N$) and compute the distance matrix Θ as shown in Fig 4. Each item $\Theta[y][x]$ is the value of $\mathrm{dis}(D_y, D_x)$ which is the distance (or estimated distance) between D_y and D_x, and we denote it by $\mathrm{dist}(y, x)$ in Fig 4. We define the variable $C[k][x]$ as the optimal set of k center TMs under two conditions: 1) D_x must be selected as a fixed center TM and 2) the other $k-1$ center TMs must be selected from $\{D_{\overline{x}} | \overline{x} = 1, 2, ..., x-1\}$. We can find $C[k][x]$ corresponds to a clustering subproblem which is determined by both k and x. We then define $d[k][x][y]$ as the distance between D_y and its center TM with respect to $C[k][x]$. Obviously, $\max_{y=1,2,...,N}(d[k][x][y])$ is the objective value of the clustering subproblem corresponding to $C[k][x]$. Since $1 \leqslant k \leqslant K$ and $1 \leqslant x \leqslant N$, we get total of $K \cdot N$ clustering subproblems/problems.

When $k = 1$, $C[k][x]$ and $d[k][x][y]$ can be obtained directly for any x since the unique center TM is determined by the specific value of x. When $k > 1$, $C[k][x]$ and $d[k][x][y]$ can be updated according to Eq. (13) and Eq. (14).

$$C[k][x] = C[k-1][\overline{x}^*] + \{D_x\}, \tag{13}$$

$$d[k][x][y] = \min(d[k-1][\overline{x}^*][y], \Theta[y][x]) \tag{14}$$

$$\text{where } \overline{x}^* = \underset{\overline{x}=k-1,...,x-1}{\operatorname{argmin}} \ \theta(\overline{x}), \text{ and} \tag{15}$$

$$\theta(\overline{x}) = \max_{y=1,2,...,N} \min(d[k-1][\overline{x}][y], \Theta[y][x]). \tag{16}$$

Here, θ represents the objective value of the corresponding clustering subproblem. $\overline{x}^*$ indicates the most appropriate $C[k-1][\overline{x}]$ which results in the smallest objective value of the k-clustering problem. The final result we want is $C[K][x^*]$ where

$$x^* = \underset{x=K,...,N}{\operatorname{argmin}} \max_{y=1,2,...,N} (d[K][x][y])), \tag{17}$$

and $\max_{y=1,2,...,N}(d[K][x^*][y])$ is the optimized objective value of the original K-clustering problem. Note that, $C[K][x^*]$ may not be the optimal solution since optimal subproblem solutions may not result in optimal solutions of parent problems. The details of the approximation algorithm of clustering are shown in Algorithm 1.

The input includes distance matrix Θ and the target number of clusters K. The output is the solution of K-clustering problem C_K. Lines 1 to 4 initialize C and d when $K = 1$. Lines 5 and 6 traverse all the combinations of cluster number k and the fixed center TM D_x. Lines 7 to 10 update $d[k][x][y]$ and $C[k][x]$, which is the core of the algorithm. Lines 11 to 12 obtain the final result, i.e., C_K. The time complexity of Algorithm 1 is $O(K \cdot N^3)$.

We compare our algorithm with Brute Force and the exploration algorithm proposed in [37]. We only consider 150 TMs so that we can obtain the Brute Force results within acceptable running time. For the exploration algorithm, we set 100,000 iterations to guarantee enough explorations. We

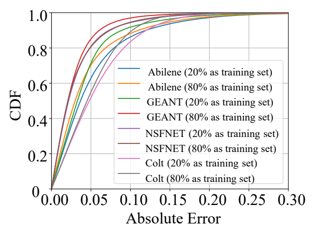


Fig. 2: CDF of absolute distance errors.

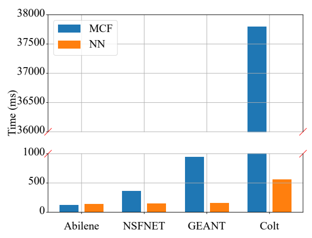


Fig. 3: Average running time to compute/estimate TM distances.

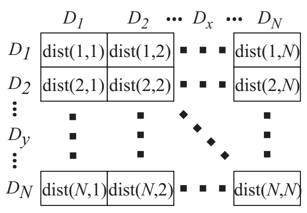


Fig. 4: Distance matrix.

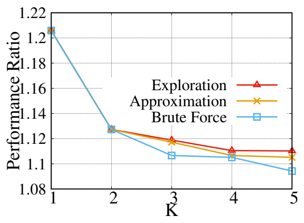


Fig. 5: The objective value of different algorithms.

Algorithm 1 Approximation Algorithm of Clustering

Input: Distance matrix Θ, the target number of clusters K;
Output: The solution of K-clustering problem C_K;

```
1:  for $x = 1$ to $N$ do
2:      $C[1][x] = \{D_x\}$;
3:      for $y = 1$ to $N$ do
4:          $d[1][x][y] = \Theta[y][x]$;
5:  for $k = 2$ to $K$ do
6:      for $x = k$ to $N$ do
7:          Get $\overline{x}^*$ through Eq. (15) and Eq. (16);
8:          for $y = 1$ to $N$ do
9:              $d[k][x][y] = \min(d[k-1][\overline{x}^*][y], \Theta[y][x])$;
10:         $C[k][x] = C[k-1][\overline{x}^*] + \{D_x\}$;
11: Get $x^*$ through Eq. (17);
12: $C_K = C[K][x^*]$;
13: return $C_K$
```

implement algorithms using Python and do simulations on a server with an 8-core Intel 3.6 GHz CPU.

Fig. 5 shows the results. The performance ratio gets smaller when K becomes larger because more TM clusters naturally result in a smaller link utilization ratio. We also find that our approximation algorithm outperforms the exploration algorithm and gets closer results to Brute Force. Note that, the performance of the exploration algorithm will decrease greatly when N and K increase because of the huge search space. Table II shows the running time of different clustering algorithms. The time of Brute Force increases rapidly as K increases. In contrast, the exploration and approximation algorithms reduce running time significantly, and the approximation algorithm is the best. When $K = 5$, our approximation algorithm takes only 1.91 seconds, but Brute Force and the exploration algorithm take 17.5 hours and nearly 2 minutes, respectively. Overall, our algorithm makes good results and works more efficiently and stably.

Discussion: It is important to choose a proper value of K for our TM clustering algorithm. We will later show by simulations that using a small K (e.g., 3 or 4) can achieve a good performance ratio (less than 1.1), and using a greater K will not make much improvement. In practice, we can run the clustering algorithm multiple times with different choices of K, compare the performance ratios of each TM cluster,

TABLE II: Running time of different clustering algorithms.

K	1	2	3	4	5
Brute Force	11ms	873ms	47s	32min	17.5h
Exploration	31s	1min20s	1min30s	1min40s	2min
Approximation	2ms	464ms	933ms	1.40s	1.91s

and then select an appropriate K value. Note that, running the algorithm with varying K values is not time-consuming, because pre-computed TM distances can be reused, and the clustering algorithm has a low complexity. This approach allows for the selection of an optimal K value that best suits the specific network conditions and requirements.

Non-parametric clustering algorithms may be an alternative approach. We attempted to employ non-parametric clustering algorithms such as DBSCAN [38], a density-based clustering algorithm. However, DBSCAN either identifies most sample TMs as noise or assigns the majority of sample TMs to a single cluster. So DBSCAN is unable to partition the TM space satisfactorily, let alone achieve comparable performance to our approach. Thus, the proposed parametric approach remains a more effective solution for this specific problem.

B. Routing Computation

We compute a suitable routing solution of the center TM of each TM cluster. The suitable routing solutions of the center TMs are treated as candidate routing solutions finally. A candidate routing solution should be able to achieve good TE performance (i.e., small performance ratio) for the TMs belonging to the corresponding cluster. That is to say, these solutions should have the potential to fit other TMs sharing similar traffic characteristics.

The simplest way is to compute an optimal routing solution for each center TM. However, such a solution usually overfits the TM and limits the generality [3]. In fact, the solution computed by oblivious routing [2] has high generality because it optimizes the routing with respect to a range of TMs. Inspired by this, we apply oblivious routing to compute the suitable routing of each center TM. Next, we describe the modified oblivious routing problem used in our design as well as how to compute suitable routing solutions.

Note that to enable a new routing solution, forwarding rules need to be updated on routers/switches to reconstruct forwarding paths and execute new splitting ratios. However, reconstructing forwarding paths on the fly may cause transient network performance degradation [3], e.g., packet loss and jitter. In contrast, modification of splitting ratios is a relatively inexpensive operation in practice. Previous works [3][4] and our tests have validated that a limited number of forwarding paths have the potential to balance various TMs well if the paths are selected carefully. Thus, we choose to pre-compute a small number (e.g., three) of forwarding paths for every demand in advance, using the method in [3]. When enabling a new routing solution, only splitting ratios need to be updated.

Let D be the target center TM. Each demand $(i, j) \in D$ with demand size $d_{i,j}$ enters the network from ingress node i and leaves from egress node j. Let $\boldsymbol{D}^*$ be the TM set where $d_{i,j}$ is restricted to $[a_{i,j}, b_{i,j}]$ (i.e., $0 \leqslant a_{i,j} \leqslant d_{i,j} \leqslant b_{i,j}$). Let $e(u, v)$ and $e(v, u)$ be the edges of link $l(u, v)$. Let $P_{i,j}$ be the pre-computed forwarding paths of demand $(i, j) \in D$. Let $f_{i,j}(p)$ be the fraction of $d_{i,j}$ that traverses path $p \in P_{i,j}$, and $\sum_{p \in P_{i,j}} f_{i,j}(p) = 1$. Let r denote the objective value (i.e., performance ratio) that we want to minimize with respect to $\boldsymbol{D}^*$. The following shows the oblivious routing problem with path limitations (denoted as P4), derived from [2]. The main difference of P4 against the problem in [2] is that we limit the forwarding paths of each demand.

$$\min \quad r \tag{18}$$

$$\text{s.t. } \forall (i,j) \in D : \sum_{p \in P_{i,j}} f_{i,j}(p) = 1, \tag{19}$$

$$\forall l \in E : \sum_{m \in E} c_m \pi(l,m) \leqslant r, \tag{20}$$

$$\forall l \in E, \forall (i,j) \in D :$$

$$\sum_{p: l \in p, p \in P_{i,j}} f_{i,j}(p)/c_l - g_l^+(i,j) + g_l^-(i,j) = q_l(i,j), \tag{21}$$

$$\forall l \in E, \forall i \in V, \forall m : e(u,v) \in m, m \in E :$$

$$\pi(l,m) + q_l(i,u) - q_l(i,v) \geqslant 0, \tag{22}$$

$$\forall l \in E : \sum_{i,j} (b_{ij} g_l^+(i,j) - a_{i,j} g_l^-(i,j)) \leqslant 0, \tag{23}$$

$$\forall (i,j) \in D, \forall p \in P_{i,j} : f_{i,j}(p) \geqslant 0, \tag{24}$$

$$\forall l, m \in E : \pi(l,m) \geqslant 0, \tag{25}$$

$$\forall l \in E, \forall (i,j) \in D : q_l(i,j) \geqslant 0, q_l(i,i) = 0, \tag{26}$$

$$\forall l \in E, \forall (i,j) \in D : g_l^+(i,j) \geqslant 0, g_l^-(i,j) \geqslant 0. \tag{27}$$

The decision variables include $f_{i,j}(p)$ which is the oblivious routing solution, $\pi(l, m)$ which is the weight of link m with respect to link l, $q_l(i, j)$ which is the path weight of the shortest path from i to j by taking $\pi(l, m)$ as the link weight of each link m on the path, and slack variables $g_l^+(i, j)$ and $g_l^-(i, j)$ for the lower and upper bound of $d_{i,j}$. Eq. (19) means all traffic amount of each demand needs to be delivered. Eq. (20) and Eq. (21) imply that r is the performance ratio. Eq. (22) guarantees $q_l(i, j)$ is the path weight of the shortest path. Eq. (23) means that $d_{i,j}$ is restricted to $[a_{i,j}, b_{i,j}]$. Note that P4 can be solved in polynomial time.

To apply P4 to compute the suitable routing solution for a given TM, firstly, we transfer the center TM to TM set $\boldsymbol{D}^*$. Specifically, given a center TM, we range each demand with a small positive constant Δ, i.e., $d_{i,j}$ is restricted to $[\max(0, d_{i,j} - \Delta), d_{i,j} + \Delta]$. The suitable routing solution of the given center TM is the optimal oblivious routing obtained by solving P4 with respect to the TMs within the range. Compared with the optimal routing solution which overfits the TM, the optimal oblivious routing fits the TMs within the range and thus has better generality.

Note that routing computation in ALTE is uncoupled with other components of the entire framework. Alternative oblivious routing models can also be employed to generate candidate routing solutions. For instance, SMORE [3], a semi-oblivious routing model, can serve as an alternative to improve the scalability. We call this approach ALTE-SMORE, which will be used in the simulations when the network size is large.

C. Classifier Selection

Given TM clusters, the best routing solution for a coming TM can be obtained by computing the TM distance between the coming TM and the center TMs. However, it is difficult and costly to obtain a coming TM exactly in real time. We need to determine the best routing solution based on traffic features that can be obtained easily. In other words, we classify a coming TM without knowing the exact traffic demand values. We leverage a classifier to map traffic features to the choice of candidate routing solutions (also TM categories).

Features, the input of ML algorithms, should contain the necessary information of TMs so that ML algorithms can classify TMs accurately. We consider two kinds of traffic statistics that can be easily obtained, i.e., the total traffic amount entering each ingress node denoted as d^{in} and the total traffic amount leaving each egress node denoted as d^{out}. For each ingress node (*resp.* egress node) we can get a d^{in} (*resp.* d^{out}). The features can be extracted from d^{in} and d^{out}. Table III shows the features selected for our ML algorithm.

We use the Gini Coefficient [39] to assess the importance of the features. Table III shows the importance ranking of the features with respect to the TM set of a network (see Section VII) when K equals three. We can see the top three features are InMin, InVar, and InMaxAveRatio in this table. We discover that the features have different importance rankings for different TM datasets and different K values. This is because different TM datasets have diverse traffic patterns. On the other hand, different K values may result in two TMs belonging to the same cluster for some K settings and belonging to different clusters for other K settings. Thus, the common features shared by TMs in the same cluster may differ under different K values. To use supervised learning algorithms, we need to label the data for training. Features are extracted from each TM in a training set. We compute the performance ratios for the TM under different candidate routing solutions and then label this TM using the index of the best candidate routing solution for the TM.

TABLE III: The description of the selected features.

Feature	Description	Gini
InAve	The average of d^{in}	11
InVar	The variance of d^{in}	2
InMax	The maximum of d^{in}	10
InMin	The minimum of d^{in}	1
OutVar	The variance of d^{out}	9
OutMax	The maximum of d^{out}	7
OutMin	The minimum of d^{out}	5
InMaxAveRatio	The ratio of InMax and InAve	3
InMinAveRatio	The ratio of InMin and InAve	8
OutMaxAveRatio	The ratio of OutMax and InAve	6
OutMinAveRatio	The ratio of OutMin and InAve	4

TABLE IV: The mean accuracy of different classifiers.

Classifier	K			
	2	3	4	5
Decision Tree	89.2%	86.5%	81.8%	69.8%
Random Forest	89.8%	87.1%	**85.9**%	73.1%
AdaBoost	**90.3**%	**87.3**%	84.7%	**75.2**%
Naive Bayes	85.0%	79.6%	69.5%	47.4%

* Accuracy: the number of samples classified correctly / the total number of samples for validation.

Next, we choose an ML algorithm as the classifier. We hope it is effective, even when labeled samples for training are few. Thus, a neural network which needs a large training set is not suitable. We compare the performance of some classification algorithms, including Decision Tree (DT) [40], Random Forest (RF) [41], AdaBoost [15], and Naive Bayes (NB) [42]. Parameters of classifiers are fine-tuned for good performance. We use 400 TMs of a network (see Section VII) for computing candidate routing solutions. We also prepare 1,000 TM samples for training and validating the classifiers.

Table IV shows the mean accuracy of different classifiers by 10-fold cross-validation [43]. We can see AdaBoost gets the highest accuracy for most K settings. DT and NB overfit the training samples, which induces low accuracy for validation sets. RF and AdaBoost have the ability to avoid the overfitting problem and are better than DT and NB, but AdaBoost outperforms RF in most cases. Besides, the training time of every classifier is no larger than 10 seconds because our training set in our test (also in practice) is small. Therefore, we choose AdaBoost as our classifier.

VI. Simulations

A. Simulation Setup

We implement ALTE using Python 3. In particular, we implement the scheme which computes TM distances by linear programming (denoted by **ALTE-MCF**), and the scheme which uses neural networks to estimate TM distances (denoted by **ALTE-NN**). For the convenience of discussion, ALTE-NN and ALTE-MCF are collectively referred to as **ALTE**. We have to point out that in the routing computation phase, the oblivious routing (OR)-based algorithm is unable to scale to large networks. So we substitute the algorithm with a more efficient one, i.e., SMORE [3], for large network topologies. This variation of ALTE is denoted by **ALTE-SMORE-MCF and ALTE-SMORE-NN**, respectively.

Some default parameters are set as follows: We precompute three forwarding paths for each demand. Δ is set to 4 Mbps for the computation of suitable routing solutions. K is set to three. The AdaBoost classifier contains 35 RF classifiers, and each RF classifier contains 35 DT classifiers whose maximum depth is set to 8. The AdaBoost classifier is developed with scikit-learn [44], an open-source ML tool. Simulations are conducted on a server equipped with an Intel i7-9700K CPU and a GeForce RTX 2080 Ti GPU.

For comparison, we implement three model-based routing schemes, i.e., the shortest path routing (SP), oblivious routing (OR) [2], and SMORE [3]. OR computes an optimal static routing solution for all possible TMs. SMORE precomputes a set of forwarding paths and obtains traffic-splitting ratios over these paths by solving linear programming based on a predicted TM. The predicted TM is computed by the exponentially weighted moving average of a continuous sequence of TMs. We also compare ALTE with an ML-based approach proposed in [6], which is one of the state-of-the-art schemes. For brevity, we name this scheme as TRPO.

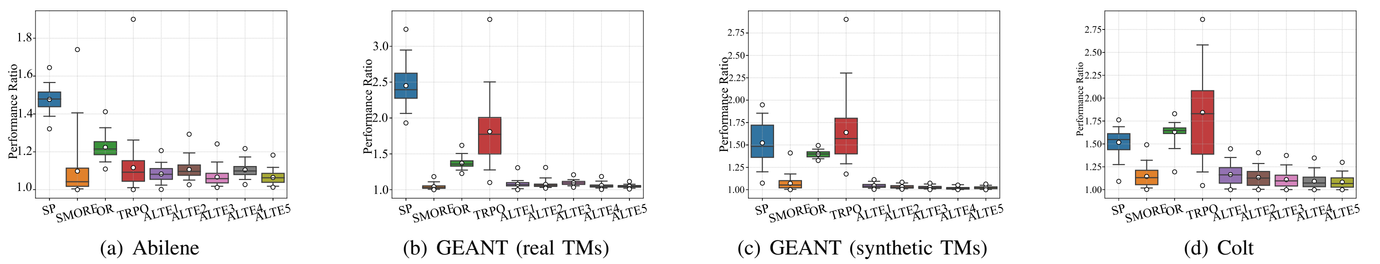
1) Topologies and TMs: Numerical simulations are conducted on four real topologies from Topology Zoo [45]. First, we adopt the Abilene topology with 11 nodes and 14 links. The link capacities and weights are set according to [34]. We use real TMs of seven days captured from Mar. 1st, 2004 [34] as a TM dataset. In particular, each TM is collected every 5 minutes, so there are 2,016 TMs in total.

Second, we use the GEANT topology with 23 nodes and 37 links. We use the link weights and TMs provided by the author of [46]. The link capacity is set to 2.5 Gbps identically. We use real TMs of seven days starting from Jan. 8th, 2005, and each TM is summarized every 15 minutes, so there are 672 TMs in total. Additionally, we generate 1,400 synthetic TMs as a supplementary dataset. We employ the Gravity Model [47] to synthesize TMs and introduce uniformly distributed random noise to each traffic demand. Both the initial dataset and this supplementary dataset are evaluated separately to assess the performance of our proposed method.

The third one is the NSFNET topology with 14 nodes and 21 links. We set identical link weights (i.e., 1) and identical link capacities (i.e., 1,000 Mbps). We utilize 1,400 synthetic TMs generated using the same method as employed for the GEANT topology. For brevity, we consider the 1,400 TMs as seven-day TMs (200 TMs for each day).

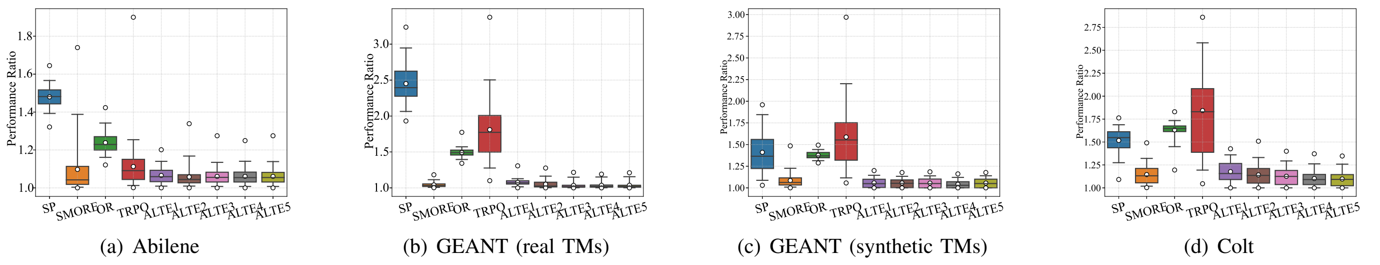
The last one is a large topology, Colt, with 92 nodes and 116 links. We also set identical link weights (i.e., 1) and identical link capacities (i.e., 1,000 Mbps). 1,400 TMs are synthesized using the aforementioned method. We employ ALTE-SMORE on Colt due to the large network size.

For all four topologies, the first 400 TMs are used for TM clustering to get K candidate routing solutions. TMs of the first five days are used to train our ML classifier. The rest two-day TMs are used as test datasets for the ML classifier.



(a) Abilene (b) GEANT (real TMs) (c) GEANT (synthetic TMs) (d) Colt

Fig. 6: The performance ratio of different schemes including ALTE-MCF. Each result contains 5%-quantile, 25%-quantile, 50%-quantile, 75%-quantile, and 95%-quantile. The maximum value, minimum value, and average value are marked in the form of circles.



(a) Abilene (b) GEANT (real TMs) (c) GEANT (synthetic TMs) (d) Colt

Fig. 7: The performance ratio of different schemes including ALTE-NN. Each result contains 5%-quantile, 25%-quantile, 50%-quantile, 75%-quantile, and 95%-quantile. The maximum value, minimum value, and average value are marked in the form of circles.

2) Metrics: First, we evaluate performance ratios which reflect the effectiveness of schemes. Second, we compare the number of routing updates (i.e., the number of switching current routing to another one) to evaluate the overhead of enabling routing solutions. Then, we observe the classification accuracy of our classifier. We also explore the effect of some parameter settings.

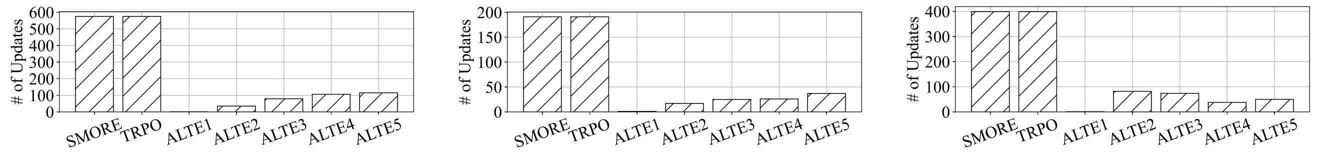
B. Simulation Results

1) Performance ratio: Fig. 6(a) shows the effectiveness of ALTE-MCF for the Abilene topology. ALTE1 to ALTE5 represent ALTE-MCF with K being from 1 to 5, respectively. We find that the median performance ratio of our scheme is much better than SP and OR, and close to SMORE and TRPO. Meanwhile, our scheme performs more stably than SMORE and TRPO. This is because SMORE and TRPO suffer from significant performance degradation if they do not predict upcoming TMs accurately. We see that the performance of ALTE-MCF is closer to the optimal as K gets larger, because more candidate routing solutions are more likely to accommodate various TMs, and good enough TE performance can be achieved when $K = 3$. Similar results can be found for the GEANT topology shown in Fig. 6(b)(c). However, TRPO in Fig. 6(b)(c)(d) performs worse than that in Fig. 6(a). This may be because TMs of GEANT and Colt change more irregularly than those of the Abilene topology, which makes the DRL agent hard to converge well.

Fig. 7(a) shows the performance of ALTE-NN for the Abilene topology. Similar to ALTE-MCF, we find ALTE-NN achieves good and stable enough performance when $K = 3$, and it achieves performance closer to the optimal as K increases. ALTE-NN also achieves more stable performance than SMORE and TRPO. Compared to SP and OR, ALTE3 can reduce the median performance ratio by 28.7% and 13.9% respectively. We can see similar results for the other topologies in Fig. 7(b)(c)(d). By comparing Fig. 6 and Fig. 7, we find ALTE-MCF and ALTE-NN achieve close performance under the same K setting, implying that neural networks in ALTE-NN can estimate TM distances accurately.

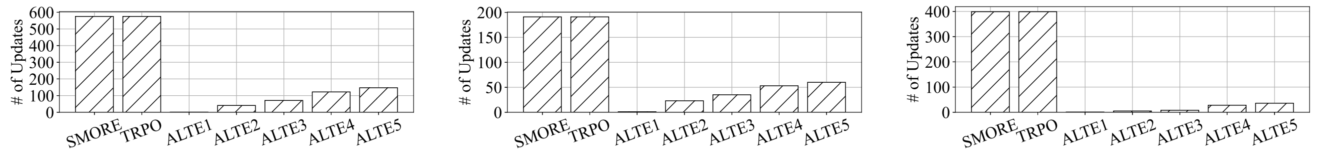
2) The number of routing updates: Fig. 8 shows the number of routing updates of ALTE-MCF, compared with different traffic-adaptive schemes. We see that ALTE-MCF induces little update overhead because the routing is altered only when the TM category changes. We also find ALTE-MCF needs more updates as K increases. This is mainly because more candidate routing solutions naturally provide more choices to adapt to TM changes. In contrast, SMORE and TRPO introduce a relatively large number of routing updates because they update routings at every TE interval. For the Abilene topology, ALTE3 reduces the updates by 84.3% compared to SMORE and TRPO when all schemes have the same TE interval. Fig. 9 shows the number of routing updates of ALTE-NN and other schemes. ALTE-NN induces update overhead similar to ALTE-MCF.

3) Classification accuracy: Fig. 10 and 13 show the accuracy of the ML classifier of ALTE-MCF and ALTE-NN, respectively. We observe that as K increases, the accuracy tends to decrease because it becomes harder to distinguish TMs when the total number of categories gets larger. We also find the accuracy for GEANT is lower than that for Abilene and NSFNET. This may be because there are many TMs located at the boundaries of two categories. Because both categories have candidate routing solutions that can achieve good



(a) Abilene (b) GEANT (c) NSFNET

Fig. 8: The number of routing updates under different schemes including ALTE-MCF.



(a) Abilene (b) GEANT (c) NSFNET

Fig. 9: The number of routing updates under different schemes including ALTE-NN.

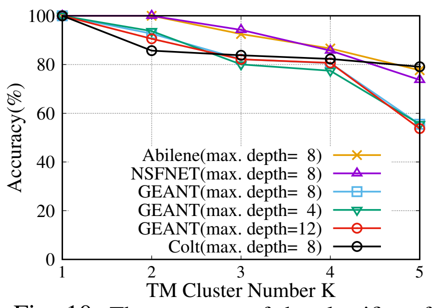


Fig. 10: The accuracy of the classifier of ALTE-MCF.

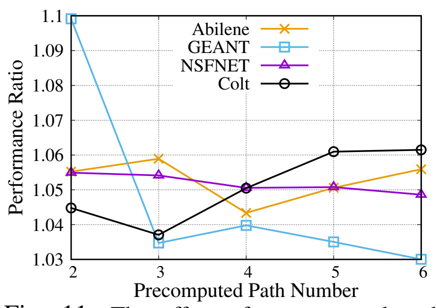


Fig. 11: The effect of precomputed path number of ALTE-MCF.

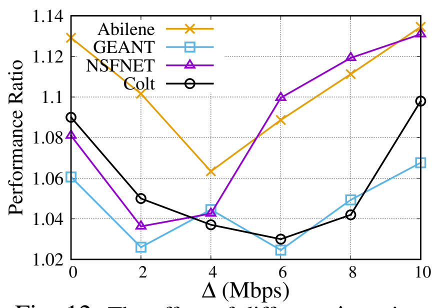


Fig. 12: The effect of different Δ settings of ALTE-MCF.

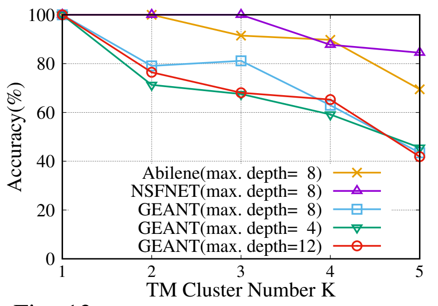


Fig. 13: The accuracy of the classifier of ALTE-NN.

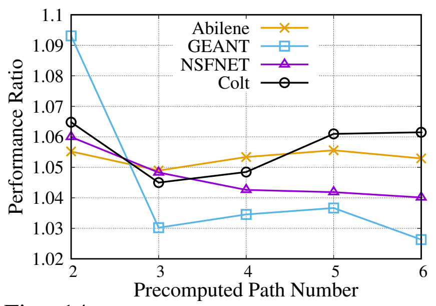


Fig. 14: The effect of precomputed path number of ALTE-NN.

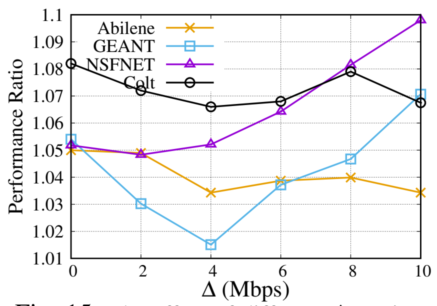


Fig. 15: The effect of different Δ settings of ALTE-NN.

performance, ALTE-MCF and ALTE-NN can still achieve small performance ratios even though the accuracy is low, as shown in Fig. 6(b) and Fig. 7(b).

4) Performance ratio under different parameter settings: We study how parameter settings affect the performance of ALTE. When we explore one parameter, other parameters are set to their default values throughout the experiments.

First, we test the effect of the decision tree's maximum depth on the accuracy of the classifier on GEANT topology. Results in Fig. 10 and Fig. 13 show that the classifier is not sensitive to the parameter and gets similar results for different settings. We set the maximum depth to 8 after a comprehensive empirical study.

Second, we study the effect of pre-computed forwarding path numbers. Fig. 11 and Fig. 14 show the median performance ratios under different path number settings. As the path number increases, the result value tends to become smaller because more paths provide more flexible routing solutions, leading to performance improvement. We also discover that more paths do not necessarily result in better performance, although more paths offer a larger solution space of problem P4 and lead to a smaller optimal value theoretically. To balance performance and overhead, we set the default path number to three.

Third, we explore the effect of different Δ settings. Δ is used in computing suitable routing solutions. Fig. 12 shows the median performance ratios under different Δ settings. We can see the result value decreases first and then increases as Δ gets larger for all three topologies. When Δ equals 0, the optimal routing solution of the center TM is used as the candidate routing solution, which lacks generality, as mentioned in Section V.B. When Δ is too large, the

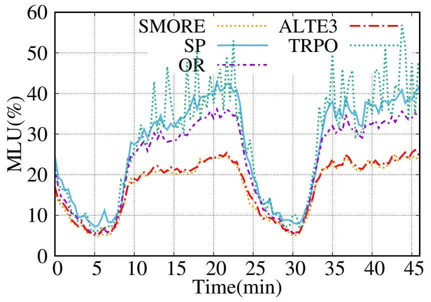


Fig. 16: The MLU over time.

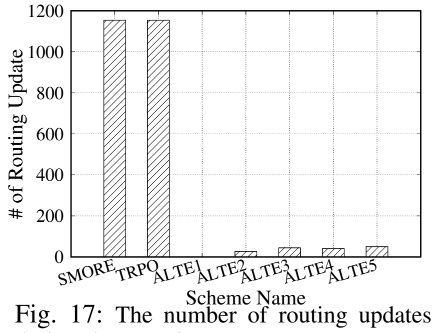


Fig. 17: The number of routing updates during the experiment.

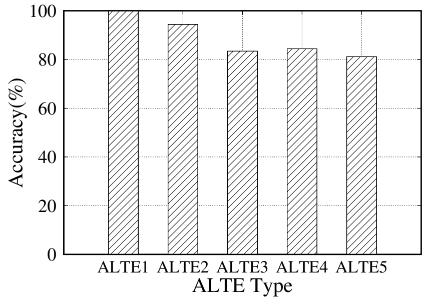


Fig. 18: The accuracy of the classifier with different K settings.

TABLE V: Offline running time of ALTE.

Topology	Offline Phase				
	Training NNs + Estimating TM Distances	Calculating TM Distances	Clustering TMs	Computing Suitable Routing Solutions	Training Classfiers
Abilene	2min2s+58s	48s	19s	3s	3s
GEANT	9min33s+1min4s	6min18s	21s	14s	4s
NSFNET	3min40s+1min	2min26s	20s	4s	4s
Colt	24min25s+3min46s	4.2h	25s	20s	5s

candidate routing solution loses much performance, just like OR. For ALTE-MCF, $\Delta = 4, 2, 2, 4$ are the best settings for Abilene, GEANT, NSFNET, and Colt, respectively. For ALTE-NN, the best settings are $\Delta = 4, 4, 2, 4$. We observe that for NSFNET, a small Δ value is better because the traffic amount is relatively small. The traffic amount of Abilene is relatively large, so a slightly larger Δ value will be better. For simplicity, we set Δ to 4 Mbps in ALTE-MCF and ALTE-NN.

5) Offline running time: Offline running time refers to the total time spent in the offline phases. The average running times of 10 simulations under default parameters are presented in Table V. For the convenience of comparison, 400 TMs are used to train NNs for each topology. The running time increases with the scale of the topology, and using neural networks to estimate TM distances is more efficient than computing distances by solving MCF problems.

VII. Experiments

A. Experiment Setup

We implement a prototype of ALTE using the Ryu [16] controller with OpenFlow [48]. We use Mininet [49] to construct the network. Forwarding rules of the pre-computed forwarding paths are installed into switches in advance. Besides, each ingress switch maintains a few *select* group tables to enable the splitting ratios of routing solutions. In particular, *select* group tables tag flows (also packets) with hash values, and flows with the same tag value will be delivered through the same forwarding path. At each TE interval, the controller collects traffic statistics from switches. Then, the classifier infers the proper routing solution based on the extracted features. If this routing solution is the same as the one in the last TE interval, nothing needs to be done. Otherwise, the splitting ratios of the new routing solution will be updated.

Due to the performance limitation of the physical machine, we choose a small-scale real network and capture TMs for the experiment, so that the replayed TMs can keep the original features. We set link capacities to 1,000 Mbps. Each TM is summarized every 5 minutes from Nov. 1st to 7th, 2018. We scale down TMs to fit the performance limit. The first 400 TMs are used to do TM clustering and get K candidate routing solutions. The TMs of the first five days are used to train our ML classifier. The rest two-day TMs are used as the test dataset. In the experiments, each TM is replayed for 5 seconds. The TE interval is 2.5 seconds. Other default values are the same as the simulations. We consider MLU to evaluate the effectiveness of ALTE. Then, we observe the number of routing updates and the accuracy of our classifier.

B. Experiment Results

For the sake of simplicity, we omit the results of ALTE-MCF and only present the results of ALTE-NN, which are very similar to those of ALTE-MCF.

Fig. 16 shows the evolution of MLU as time goes by under different routing schemes, including ALTE-NN. We can see ALTE3 (i.e., ALTE-NN with $K = 3$) and SMORE achieve almost the same performance. This is because there are few bursts in the captured TMs, and SMORE can adapt to traffic dynamics well. We also find ALTE3 outperforms TRPO, SP, and OR, especially when the traffic amount is large. Specifically, at the 20th minute, ALTE3 reduces the MLU by 33.7%, 37.2%, and 26.5% compared to TRPO, SP, and OR, respectively. However, TRPO achieves very unstable TE performance.

Fig. 17 presents the number of routing updates. ALTE induces more routing updates as K increases, but the cost is still small (e.g., only 81 updates during 1,152 TE intervals for ALTE5). SMORE and TRPO introduce more routing updates. Results of Fig. 16 and Fig. 17 show that ALTE achieves good load balancing and needs fewer routing updates.

Fig. 18 shows the accuracy of the classifier. Similar to the results of the simulations, the accuracy decreases as K increases. When $K = 3$, the accuracy is 82%. In fact, ALTE3 can achieve good enough performance, as shown in Fig. 16.

VIII. Conclusion

In this paper, we proposed ALTE. We developed a novel clustering algorithm to group TMs and computed candidate routing solutions for each TM category. We trained an ML classifier to infer the best candidate routing solution online based on easily measured statistics. Evaluation results showed both schemes of ALTE perform well for various TMs with negligible overhead.

References

[1] N. Geng, M. Xu, Y. Yang, E. Dong, and C. Liu, "Adaptive and low-cost traffic engineering based on traffic matrix classification," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020, pp. 1–9.

[2] D. Applegate and E. Cohen, "Making routing robust to changing traffic demands: algorithms and evaluation," *IEEE/ACM TON*, vol. 14, no. 6, pp. 1193–1206, 2006.

[3] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé, "Semi-oblivious traffic engineering: The road not taken," in *USENIX NSDI*, 2018.

[4] M. Leconte, A. Destounis, and G. Paschos, "Traffic engineering with precomputed pathbooks," in *IEEE INFOCOM*, 2018.

[5] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Load balancing for multiple traffic matrices using SDN hybrid routing," in *IEEE HPSR*, 2014.

[6] D. A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route with deep RL," in *NIPS*, 2017.

[7] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM*, 2018.

[8] K. Xu, M. Shen, H. Liu, J. Liu, F. Li, and T. Li, "Achieving optimal traffic engineering using a generalized routing framework," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 51–65, 2015.

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

[10] H. Racke, "Minimizing congestion in general networks," in *FOCS*, 2002.

[11] C. Zhang, Y. Liu, W. Gong, J. Kurose, R. Moll, and D. Towsley, "On optimal routing with multiple traffic matrices," in *INFOCOM*, 2005.

[12] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," *Elsevier Ad hoc networks*, vol. 2, no. 1, pp. 1–22, 2004.

[13] A. Soule, A. Nucci, R. L. Cruz, E. Leonardi, and N. Taft, "Estimating dynamic traffic matrices by using viable routing changes," *IEEE/ACM TON*, vol. 15, no. 3, pp. 485–498, 2007.

[14] G. Rétvári and G. Németh, "Demand-oblivious routing: distributed vs. centralized approaches," in *2010 Proceedings IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[15] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[16] "Ryu," https://osrg.github.io/ryu, 2018.

[17] 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 computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

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

[19] P. Casas, L. Fillatre, and S. Vaton, "Multi hour robust routing and fast load change detection for traffic engineering," in *IEEE ICC*, 2008.

[20] Q. Xu, Y. Zhang, K. Wu, J. Wang, and K. Lu, "Evaluating and boosting reinforcement learning for intra-domain routing," in *IEEE MASS*, 2019.

[21] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2020.

[22] P. Cong, Y. Zhang, W. Wang, K. Xu, R. Li, and F. Li, "A deep reinforcement learning-based routing scheme with two modes for dynamic networks," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[23] C. Liu, M. Xu, Y. Yang, and N. Geng, "Drl-or: Deep reinforcement learning-based online routing for multi-type service requirements," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

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

[25] A. Singh, S. Sharma, and A. Gumaste, "Using deep reinforcement learning for routing in ip networks," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–9.

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

[27] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *ACM Big-DAMA*, 2018.

[28] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of Graph Neural Networks for network modeling and optimization in sdn," in *ACM SOSR*, 2019.

[29] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE TOC*, vol. 66, no. 11, pp. 1946–1960, 2017.

[30] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Is machine learning ready for traffic engineering optimization?" in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–11.

[31] S. S. Ahuja, V. Gupta, V. Dangui, S. Bali, A. Gopalan, H. Zhong, P. Lapukhov, Y. Xia, and Y. Zhang, "Capacity-efficient and uncertainty-resilient backbone network planning with hose," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 547–559.

[32] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. USA: Prentice-Hall, Inc., 1993.

[33] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, p. 807–814.

[34] "Abilene," http://www.cs.utexas.edu/%7Eyzhang/research/AbileneTM.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[36] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[37] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for K-medoids clustering," *Expert systems with applications*, 2009.

[38] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[39] "Gini coefficient," https://en.wikipedia.org/wiki/Gini_coefficient/, 2020.

[40] B. Kamiński, M. Jakubczyk, and P. Szufel, "A framework for sensitivity analysis of decision trees," *Central European journal of operations research*, vol. 26, no. 1, pp. 135–159, 2018.

[41] L. Breiman, "Random forests," *Machine learning*, 2001.

[42] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *ACM ICML*, 2006.

[43] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, 1995.

[44] "scikit-learn," https://scikit-learn.org/stable, 2019.

[45] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[46] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," in *ACM SIGCOMM*, 2006.

[47] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 93–96, 2005.

[48] "Openflow v1.3," https://www.opennetworking.org/product-certification.

[49] "Mininet," http://mininet.org, 2018.

Yi Liu received his B.Sc. degree in automation from Tsinghua University in 2022. He is currently pursuing a Ph.D. degree at the Department of Computer Science and Technology of Tsinghua University. His main research interests include network routing and network security.

Nan Geng received the Ph.D. degree from Tsinghua University in 2021. He is currently an Internet Routing Protocol Researcher at Huawei Technology.

Mingwei Xu (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees from Tsinghua University. He is currently a Full Professor with the Department of Computer Science, Tsinghua University. His research interests include computer network architecture, high-speed router architecture, and network security.

Yuan Yang (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University. He was a Visiting Ph.D. Student with The Hong Kong Polytechnic University from 2012 to 2013. He is currently an Assistant Researcher with the Department of Computer Science and Technology, Tsinghua University. His research interests include computer network architecture, routing protocol, and green networking.

Enhuan Dong received the B.E. (2013) degree from Harbin Institute of Technology, Harbin, China, and the Ph.D. (2019) degree from Tsinghua University, Beijing, China. He was a visiting Ph.D. student at the University of Goettingen in 2016-2017. He is an Assistant Research Professor in the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network security, network operations and network transport.

Chenyi Liu received his B.Sc. degree in computer science and technology from Tsinghua University in 2019. He is currently a PhD candidate at the Department of Computer Science and Technology of Tsinghua University. His main research interests include traffic engineering and machine learning.

Qiaoyin Gan received his B.Sc. degree in computer science from Tsinghua University in 2023. He is currently a master's student at the Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include smart network interface cards (SmartNICs) and RDMA transmission.

Qing Li (Senior Member, IEEE) received the B.S. degree in computer science and technology from the Dalian University of Technology, Dalian, China, in 2008, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. He is currently an Associate Researcher with the Peng Cheng Laboratory, China. His research interests include reliable and scalable routing of the internet, software defined networks, network function virtualization and in-network caching/computing.

Jianping Wu (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University. He is currently a Full Professor and the Director of the Network Research Center and a Ph.D. Supervisor with the Department of Computer Science and Technology, Tsinghua University. Since 1994, he has been in charge of the China Education and Research Network. His research interests include the next-generation internet, IPv6 deployment and technologies, and internet protocol design and engineering.