Distributed Traffic Engineering in Hybrid Software Defined Networks: A Multi-agent Reinforcement Learning Framework

Yingya Guo, Qi Tang, Yulong Ma, Han Tian and Kai Chen

✦

Abstract—Traffic Engineering (TE) is an efficient technique to balance network flows and thus improves the performance of a hybrid Software Defined Network (SDN). Previous TE solutions mainly leverage heuristic algorithms to centrally optimize link weight setting or traffic splitting ratios under the static traffic demand. Note that as the network scale becomes larger and network management gains more complexity, it is notably that the centralized TE methods suffer from a high computation overhead and a long reaction time to optimize routing of flows when the network traffic demand dynamically fluctuates or network failures happen. To enable adaptive and efficient routing in TE, we propose a Multi-agent Reinforcement Learning method CMRL that divides the routing optimization of a large network into multiple small-scale routing decision-making problems. To coordinate the multiple agents for achieving a global optimization goal, we construct an interactive environment for training the routing agents that own partial link utilization observations. To optimize credit assignment of multi-agent, we introduce the difference reward assignment mechanism for encouraging agents to take better action. Extensive simulations conducted on the real traffic traces demonstrate the superiority of CMRL in improving TE performance, especially when traffic demands change or network failures happen.

Index Terms—Traffic engineering, Software Defined Networks, Multi-agent reinforcement learning

1 INTRODUCTION

Due to the explosive growth of Internet traffic, Traffic Engineering (TE) has gained increasing attentions in achieving traffic balancing and improving network performance [1]. Nowadays, the TE performance of traditional distributed network is largely constrained by its adopted shortest-path-routing protocols. Fortunately, with the emergence of Software Defined Network (SDN) [2] architecture, the decoupling of control plane and data plane enables TE to design flexible solutions for better optimizing traffic routing. However, the full-SDN enabled network, which upgrades all legacy routers with SDN switches, encounters economical and technical problems [3]. Therefore, hybrid SDN, in which SDN switches are partially deployed in

- *Yingya Guo, Qi Tang, Yulong Ma are with the College of Computer and Data Science, Fuzhou University; the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, and also with the Key Laboratory of Spatial Data Mining & Information Sharing, Ministry of Education Fujian, P.R.China, 350003.*
- *Han Tian and Kai Chen are with the Department of Computer Science & Engineering, Hong Kong University of Science & Technology.*

the traditional distributed networks, is widely adopted by Internet Service Providers (ISPs) as a practical solution to realize a smarter TE. Many studies have shown that TE in the hybrid SDN can achieve the network performance close to the full-SDN enabled network [4].

Previous TE solutions for hybrid SDN mainly focused on developing various heuristics [4]–[7]. These heuristics are often human-designed and optimize routing policies only on a single traffic demand. As a result, the routing policies derived from these heuristics inevitably suffer a performance degradation for the inability to adapt to the dynamically-changing network environment, e.g. fluctuating traffic demands or network link failures. In addition, due to the high computation and communication overhead, it is impractical for these heuristics to promptly re-calculate and deploy the appropriate routing strategy in a dynamically-changing network environment.

As an essential branch of machine learning, Reinforcement Learning (RL) has exhibited great potential in tackling dynamic decision-making problems by enabling an experience-driven and model-free control [8]. Instead of the human-designed heuristics, RL learns an intelligent agent to adaptively and rapidly derive optimal policies according to different environments. Without any supervised information, RL can automatically accumulate a large number of valuable experience by repeatedly interacting with a virtual environment in a trial and error manner. The accumulated experience helps the intelligent agent to discover the hidden patterns in the historical data, and establish the direct relationship between dynamic environments and optimal policies.

Some pioneering studies have attempted to leverage RL technique to address the TE problems in the hybrid SDN [9], [10]. However, as the network scales increase, the action space increases rapidly and it will be intractable to make accurate online routing inference with a single agent [11]. Unlike these studies adopting the single-agent RL framework to implement a centralized TE, this paper proposes a multi-agent RL framework for achieving a distributed TE in a hybrid SDN. As shown in Fig. 1, the workflow of our proposed multi-agent RL framework exhibits two main advantages: 1) the multi-agent RL framework needs no additional communication overhead to exchange the network information when deciding the routing strategy,
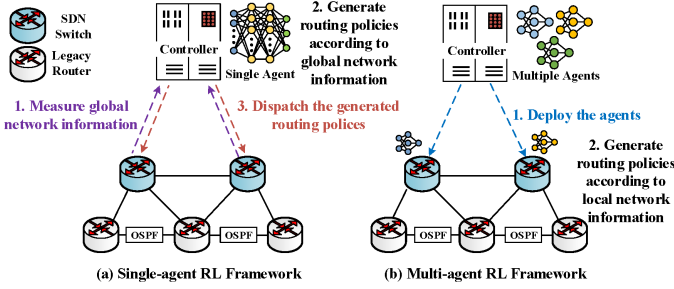
Fig. 1. The illustration of different workflows of single-agent RL framework and multi-agent RL framework for achieving TE in hybrid SDN.

since it only needs the local network information rather than the global network information, which makes the routing inference more efficient; 2) the multi-agent RL framework performs a higher potential to improve network performance and exhibits better scalability, because it decreases the problem complexity and improves the convergence by decomposing a large-scale problem into several small-scale sub-problems, each of which is solved by a relatively simple and independent agent.

However, developing an efficient multi-agent RL framework to achieve distributed TE in hybrid SDN still encounters the following challenges. First, the state, reward and action functions should be carefully designed to enable an efficient multi-agent training. Constructing an interactive environment for enabling multi-agent to collaboratively learn the map between the traffic demands and routing policies poses a great challenge. Second, in a cooperative setting, the global reward generated by the joint action makes it hard to quantify the contribution of each agent and the individual reward for each agent can hardly motivate it for taking better action. Assigning reasonable credits for different agents and designing a difference reward assignment mechanism to the agents for encouraging the better actions of agents pose another challenge.

To address the above challenges, we innovatively propose a Counterfactual-based Multi-agent Reinforcement Learning method CMRL for improving TE performance in a hybrid SDN. Specifically, we first construct an interactive environment with carefully designed state, action and reward functions. Then, we propose a Deep Deterministic Policy Gradient (DDPG)-based multi-agent reinforcement learning method with difference reward assignment for training the routing agents in the constructed environment. For the agent credit assignment, we introduce difference reward for agents and adopt a counterfactual-inspired independent reward assignment mechanism inspired by [12] for achieving a high TE performance. Finally, the learned agents intelligently and timely generate the adaptive routing policies to control the forwarding behaviors of SDN switches in the highly dynamic environment. Through extensive experimental results and evaluations, we demonstrate that our proposed method has a superior performance compared with the previous TE solutions.

In a nutshell, the main contributions of this paper can be summarized as follows:

- We propose a multi-agent reinforcement learning based approach CMRL to efficiently solve the TE

problem in the dynamic hybrid SDN, especially when the network scale or network complexity increases. Specifically, an interactive environment is first constructed for offline agent training. Next, the routing agents are trained offline in the constructed environment for collaboratively learning the map between the traffic demands and routing policies of SDN switches. Finally, the trained agents are deployed to enable a timely and intelligent routing policy inference when traffic demands change or network failures happen.
- We integrate difference reward assignment among agents into offline agent training in order to solve the multi-agent credit assignment problem. Specifically, each agent is given a reward that computes the estimated return for the current joint action to a baseline that marginalises out the agent's action and with the other agents' actions unchanged. The difference reward assignment encourages different agents to sacrifice for the better actions in the cooperative setting.
- We conduct extensive experiments on real network topologies and traffic traces to evaluate the performance of CMRL. The experimental results demonstrate the superior performance of our proposed method CMRL in improving TE performance of the hybrid SDN when traffic demands change or network failures happen.

The rest of the paper is organized as follows. Section 2 presents the related work on TE solutions. Section 3 provides the problem definition. In Section 4, we describe our proposed method CMRL, including the offline training part and online inference part. Section 5 shows the experimental results of different algorithms on real network topologies and traffic traces when traffic demands change or network failures happen. Finally, we make the conclusion and give possible future research direction in section 6.

2 RELATED WORK

In this section, we present the related work on the TE solutions in traditional distributed network and Software Defined Networks, respectively.

2.1 TE Solutions in Traditional Distributed Network

In the traditional distributed network, distributed routing protocols dominate and the traffic is constrained to route on the shortest paths between the source and destination according to link weight setting under the distributed routing protocols, such as Open Shortest Path First (OSPF) protocol [13], Intermediate System to Intermediate System (IS-IS) protocol [14]. The link weight setting under the distributed routing protocols determines the available shortest paths for routing and many related works focus on optimizing distributed link weight setting using heuristic algorithms [15]–[17] or machine learning approach based on gradient descent [18]. However, for traditional distributed networks, flow routing lacks of flexibility with the shortest-path-routing constraint, which greatly limits the TE performance.

2.2 TE Solutions in Software Defined Networks

With the prevailing of SDN architecture, the routing gains more flexibility. The SDN controller can centrally control the forwarding behavior of SDN switches through dispatching flow entries and traffic can be routed on all available paths between a source-destination pair, regardless of shortest paths constraint. Microsoft [19] and Google [20] have already built the SDN-enabled datacenter networks and have boosted the network utilization to near 100% through flexible flow routing. However, fully deploying the SDN switches to replace the legacy router is a non-trivial task and encounters various challenges. To incrementally deploy the SDN switches into traditional network, Agarwal et al. [4] propose a greedy approach to determine the placement of SDN switches. To improve the TE performance of the hybrid SDN, a Fully Polynomial Time Approximation Scheme is also introduced in [4] to optimize the traffic splitting ratio at SDN switches. To further reduce the MLU of the hybrid SDN, Guo et al. [5], [6] propose heuristic algorithms that jointly optimize the OSPF link weight setting and traffic splitting ratio at SDN switches under static and dynamic environments. To maximize the network throughput in hybrid SDN, Xu et al. [21] introduce an approximation algorithm to optimize flow routing on h available paths between a source-destination pair.

With the significant advances of reinforcement learning in various fields, more researchers begin to apply them in solving TE problems. Noting that the modern communication networks are becoming more complicated and highly dynamic, Xu et al. [22] and Chen et al. [23] propose novel experience-driven model-free Deep Reinforcement Learning (DRL) methods for solving TE problem. To mitigate the impact of network disturbance, Zhang et al. [24] introduce a reinforcement learning method CFR-RL for selecting and rerouting the critical flows to balance the network link utilization. To better optimize the link weight setting under OSPF protocol in a hybrid SRv6 network, Tian et al. [9] propose a DRL-based method for learning the optimal OSPF link weight setting under the given traffic demands in a trail and error manner.

To efficiently react to dynamic environment and obtain the routing policy in an online manner, Guo et al. [10] adopt a DRL method ROAR for learning the mapping between the traffic demands and routing polices. The trained agent can promptly infer the optimal routing policy when traffic demands change or network failures happen. To solve the TE problem in a multi-region scenario, Geng et al. [25], [26] propose a distributed TE framework based on DRL for optimizing route selection under highly dynamic traffic. To optimize route selection of traffic flows under QoS requirements, Liu et al. [27] propose an online routing algorithm DRL-OR based on DRL for computing the optimal next hop at each router and further introduce safe learning mechanism to facilitate the online learning process.

Previous TE works in a hybrid SDN either leverage heuristic algorithms or single-agent reinforcement learning for optimizing network performance. As with the rapid increasing of network scales and network complexity, previous TE solutions suffer a performance degradation when traffic demands fluctuate or network failures happen be-

cause of high delay in computing routing policies. In this paper, we innovatively leverage the multi-agent reinforcement learning with a combination of difference reward assignment mechanism to improve the TE performance in a dynamic hybrid SDN environment.

3 PROBLEM DEFINITION

In our hybrid SDN network environment, the network topology is denoted as an undirected graph $G = (V, E)$. Here, V represents the set of forwarding devices, which is composed of SDN switches V_s and legacy routers V_l. E refers to the set of links, where the capacity of each link $e \in E$ is represented as $C(e)$. Here, the link capacity refers to the maximum traffic volume that the link can accommodate. The Traffic Matrix (TM) is represented by D, which is a set of total traffic demands that need to be delivered from the source to the destination. The element $D(u, v)$ in TM D represents the total traffic demand from source node u to destination node v. The traffic routing in TE is to distribute the traffic demands onto the available candidate paths between the source and the destination to optimize maximum link utilization, network cost, or network throughput, etc. Specifically, at the legacy routers, the traffic is routed to the next hops on the shortest paths. At the SDN switches, the traffic can be flexibly routed on the multiple next hops to the destination. We refer to the common goal of TE, i.e., minimizing the Maximum Link Utilization (MLU) of a hybrid SDN without violating the link capacity constraints, as our TE optimization goal in this paper. Here, link utilization refers to the ratio of the link load to link capacity (bandwidth) and MLU indicates the largest link utilization of all links. Our approach can also be easily extended and adopted to optimize other TE goals in a hybrid SDN.

4 PROPOSED METHOD CMRL

In this section, we first provide an overview of the proposed method CMRL. Then, we exhibit the offline agents training phase of CMRL for learning the hidden patterns between the traffic demands and routing policy. Finally, with the trained agents deployed, the online routing inference phase of CMRL is elaborated on to exhibit the routing policy generation process when new traffic demands arrive.

4.1 An overview of CMRL

Our proposed method CMRL consists of two phases: offline agents training and online routing inference. An overview of CMRL is presented in Fig. 2.

In the offline agents training, to train these agents, which are denoted by Deep Neural Networks (DNNs), an interactive environment should be first constructed for the interaction with the multiple agents. Then, given the traffic and topology information, multiple routing agents collaboratively learn the mapping between the traffic demands and routing policies through the interaction with the interactive environment. The well trained agents are deployed at SDN switches for online routing inference.
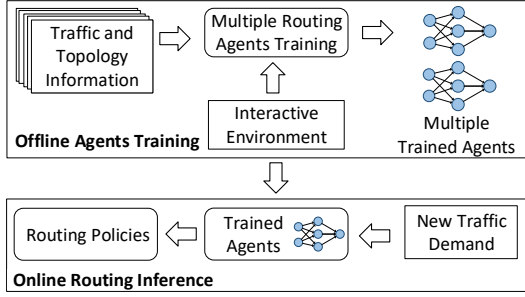
Fig. 2. An overview of CMRL.

In the online routing inference, given the new arrival traffic demands and partial observed adjacent link utilization information, the distributedly deployed trained agents can infer the routing policy promptly with the trained DNNs. In the following, we will introduce the two phases in details.

4.2 Offline Agents Training

To enable routing agents intelligently and efficiently learn the mapping between the traffic demands and routing policy, we adopt to Deep Deterministic Policy Gradient (DDPG) [8], which is an off-policy, model-free, actor-critic DRL algorithm, for training the agents through the interaction with environment. Compared to other DRL algorithms, DDPG can handle the continuous high-dimension action space and state space, which is suitable for learning the continuous splitting ratios of traffic at SDN switches. In this part, we first elaborate on the interactive environment construction for training the multiple DRL agents offline. Then, given the interactive environment, we introduce the DDPG-based offline multi-agent training process.

4.2.1 Interactive Environment

In the offline training phase, the multiple agents collaboratively learn the routing policy through interacting with the environment. Therefore, the interactive environment should be constructed first. As shown in Fig. 3 (a), in each time step t, each agent h has a partial observation of the network status o_t^h and chooses corresponding action a_t^h according to the partial observed environment state o_t^h. After agents taking the action, the environment provides a reward r_t for the joint action $\boldsymbol{u_t} = \{a_t^h\}_{h=1}^H$ and the centralized controller receives the reward r_t that indicates the performance of the joint action $\boldsymbol{u_t}$. Here, H denotes the total number of agents. To make the agents learn the better action and achieve the global TE goal, we introduce the difference reward assignment for each agent. The difference reward assignment, denoted as $A_t^h(s_t, \boldsymbol{u})$, is computed by the controller for specifying the contribution of each agent and participates in the actor network update. Then, the environment transits from s_t to the new state s_{t+1}. The past experiences, with a form of $(\{o_0^h\}_{h=1}^H, \boldsymbol{u_t}, s_t, r_t, s_{t+1})$, are restored in the replay buffer for agent learning. An optimal policy $\{\pi_h^*\}$, which maps the partial observed network state to a probability distribution over actions, maximizes the expected cumulative discounted reward $E_{\{\pi_h^*\}}[\sum_t \gamma^t r_t]$. Here, the term $\gamma \in [0, 1]$ is the discount factor which prevents an infinite sum of

accumulated rewards. To be specific, the state, action and reward functions are designed as follows.

State: The state is the input to the agents and should be carefully designed. The link utilization, which is determined by both the traffic demands and routing policy, can well reflect the network status. To better describe the network status and minimize the MLU in a hybrid SDN, we define the entire environment state s_t at time t based on the utilization of each link s_t^e, i.e., $s_t = \{s_t^e, e \in E\}$. The link utilization s_t^e can be easily computed by the link load and the link capacity shown in Eq. (1).

$$s_t^e = \frac{\sum\limits_{q \in V} f_{e,D_t}^q}{C(e)} \tag{1}$$

Here, f_{e,D_t}^q is the splitting traffic on link e destined to q under TM D_t. $\sum\limits_{q \in V} f_{e,D_t}^q$ calculates the total traffic volume on link e under TM D_t.

In a practical SDN-TE system, each SDN switch can obtain the utilization of all links through the information flooded regularly by other switches. However, leveraging the utilization information of all links leads to the slow convergence of the method. To improve the convergence speed of CMRL and better guide the training of DNNs in agents, we exploit the partial observation of each SDN switch instead of the entire link utilization in offline agents training. Here, the partial observation of the SDN switch refers to the adjacent link utilization of the switch and we use o_t^h to denote the partial observation of the SDN switch with the deployed agent h at time t, $h \in [1, H]$.

Action: The action is the output of the agent which indicates the routing policy of the deployed SDN switch. The action a_t^h of agent h at time t is a vector that consists of traffic-splitting ratios χ_{e,D_t}^h distributed to the adjacent links e of SDN switch h under TM D_t as shown in Eq. (2). It should be noted that the sum of the splitting ratios of the traffic on all the adjacent links of each SDN node should be equal to 1. Therefore, we adopt to a normalization of the output to the actor networks for guaranteeing this constraint. The joint actions $\boldsymbol{u_t}$ of the multiple agents are defined as $\boldsymbol{u_t} = \{a_t^h\}_{h=1}^H$.

$$a_t^h = \{\chi_{e,D_t}^h\}_{p_e=h} \tag{2}$$

p_e denotes the head of the link e.

Reward: The agents work collaboratively for achieving a global TE goal, which is to minimize the MLU of the hybrid network. Therefore, we design a reward function which can reflect the global TE performance of the hybrid SDN. The reward function related to MLU is shown in Eq. (3):

$$r_t = \begin{cases} -e^{2(\frac{1}{\alpha}-1)} & \text{if } \alpha > 1 \\ 0 & \text{if } \alpha = 1 \\ e^{2(\alpha-1)} & \text{if } \alpha < 1 \end{cases} \tag{3}$$

$$\alpha = \frac{U_{i,0}^{max}}{U_{i,t}^{max}} \tag{4}$$

Here, $U_{i,0}^{max}$ and $U_{i,t}^{max}$ denotes the MLU of the network under the TM D_i at time 0 and t, respectively. We treat $U_{i,0}^{max}$ as a baseline in minimizing MLU, which is computed
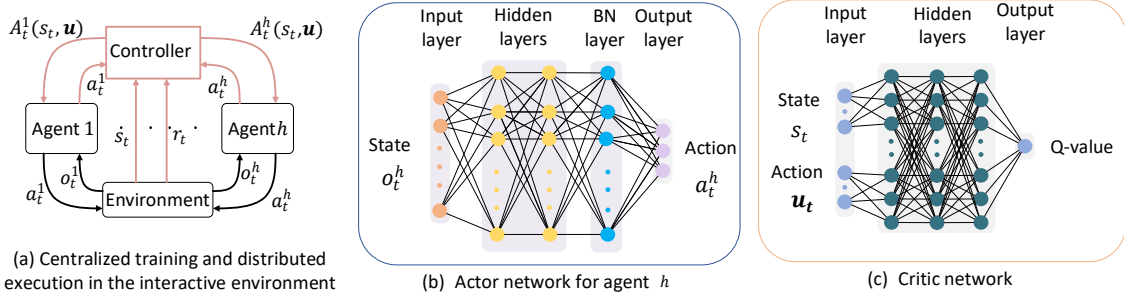
Fig. 3. The architectures of multi-agent reinforcement learning, actor network and critic network. In (a), the red arrows and components are only required in centralized training.

by routing all the flows to the next hops on the shortest paths according to the OSPF protocol. The α value, which is computed by Eq. (4), shows the improvement ratio of the network performance when the new routing policies are deployed at time t. As shown in Eq. (3), a higher positive reward r_t is given if the joint action taken by multiple agents at time t generates a lower MLU which improves network performance. Otherwise, a lower negative reward is given if the action taken generates a higher MLU that degrades the network performance.

4.2.2 Offline Training

Given the interactive environment, multiple routing agents are trained offline with DDPG for learning the mapping between the traffic demands and routing policy. Each agent maintains actor networks for approximating the mapping between the input network status to routing policy outputs. There is also a critic network for evaluating the actions taken by the agents and helping update the parameters of actor networks. Specifically, the actor network of each agent outputs an action a_t^h based on its partially observed state o_t^h, and the critic network generates a reward r_t given the joint action $\boldsymbol{u_t}$ and partially observed state o_t^h of each agent h. Both the actor network and the critic network have two sub-networks, the online network and the target network, respectively and the structures of the two sub-networks are the same.

As shown in Fig. 3(b), the actor network consists of input layer, hidden layer, Batch Normalization (BN) layer and output layer. The two hidden layers are fully-connected layers and each layer contains 1024 neurons. The activation function is ReLU. The BN layer is added to the actor network in order to speed up the training and convergence of the neural network as well as to control the gradient exploding and prevent the gradient vanishing. The output layer refers to Softmax as its activation function to ensure that the sum of the splitting ratios for a demand equals 1. For evaluating the joint action taken by agents, a critic network is kept at the centralized controller to generate a Q-value for the joint action. As shown in Fig. 3(c), the critic network consists of an input layer, three hidden layers and an output layer. Both three fully-connected hidden layers have 1024 neurons and the activation function is also ReLU. The historical data of actor network and critic network are stored in the replay buffer B for training.

Algorithm 1 presents the offline training phase of multiple agents. The inputs in Algorithm 1 contain the hybrid SDN environment G, the historical TMs D, the set of available links L for each traffic demand, and the number of agents H.

Algorithm 1 Offline agents training

Input: $G = (V, E)$, D, L, H
Output: $\boldsymbol{\mu} = \{\mu_h(\boldsymbol{o_h}|\theta_h^\mu)\}_{h=1}^H$
1: Initialize action set $\boldsymbol{u} = \{a_h\}_{h=1}^H$;
2: Initialize H online actor networks $\{\mu_h(\boldsymbol{o_h}|\theta_h^\mu)\}_{h=1}^H$ and an online critic network $Q(s, \boldsymbol{u}|\theta^Q)$ with random $\{\theta_h^\mu\}_{h=1}^H$ and θ^Q;
3: Initialize H target actor networks $\{\mu_h'(\boldsymbol{o_h}|\theta_h^{\mu'})\}_{h=1}^H$ and a target critic network $Q'(s, \boldsymbol{u}|\theta^{Q'})$ with $\{\theta_h^{\mu'}\}_{h=1}^H \leftarrow \{\theta_h^\mu\}_{h=1}^H$, $\theta^{Q'} \leftarrow \theta^Q$;
4: Initialize replay buffer B ;
5: **for** D_i in $\boldsymbol{D}$ **do**
6:     Initialize OU process $\boldsymbol{N} = \{\mathcal{N}^h\}_{h=1}^H$;
7:     $\epsilon = 1.0$;
8:     **for** episode $n = 1 \cdots N$ **do**
9:         $F_{i,0} =$get_ospf_flows(G, D_i);
10:         $(\{o_0^h\}_{h=1}^H, s_0, U_{i,0}^{max}) =get_state(F_{i,0})$;
11:         **for** step $t = 0 \cdots T - 1$ **do**
12:             **for** agent $h = 1 \cdots H$ **do**
13:                 $a_t^h = \mu_h(o_t^h|\theta_h^\mu) + \epsilon\mathcal{N}_{(n-1)T+t}^h, \epsilon \leftarrow \eta\epsilon$;
14:             **end for**
15:             $\boldsymbol{u_t} = \{a_t^h\}_{h=1}^H$;
16:             $\boldsymbol{P} =$ get_policy$(\boldsymbol{u_t}, L)$;
17:             $F_{i,t+1} =$ get_flows$(G, D_i, \boldsymbol{P})$;
18:             $(\{o_{t+1}^h\}_{h=1}^H, s_{t+1}, U_{i,t+1}^{max}) =get_state(F_{i,t+1})$;
19:             $r_t =$get_reward$(U_{i,t+1}^{max}, U_{i,0}^{max})$;
20:             **if** $t = T$ **then**
21:                 $done_t = 1$;
22:             **else**
23:                 $done_t = 0$;
24:             **end if**
25:             B.store$(\{o_t^h\}_{h=1}^H, \boldsymbol{u_t}, s_t, r_t, s_{t+1}, done_t)$;
26:             minibatch $B' = B$.sample(M);
27:             **for** $(\{o_j^h\}_{h=1}^H, \boldsymbol{u_j}, s_j, r_j, s_{j+1}, done_j) \in B'$ **do**
28:                 $y_j = r_j + \gamma(1 - done_j)Q'(s_{j+1}, \{\mu_h'(o_{j+1}|\theta_h^{\mu'})\}_{h=1}^H| \theta^{Q'})$;
29:             **end for**
30:             Update online critic and actor with Eq. (5)-(7);
31:             Update target critic and actor with Eq. (8)(9);

```
32:        end for
33:     end for
34:  end for
35: return μ ;
```

In Algorithm 1, we start with initialization (lines 1-4). The online actor networks $\{\mu_h(\boldsymbol{o_h}|\theta_h^\mu)\}_{h=1}^H$ and online critic network $Q(s, \boldsymbol{u}|\theta^Q)$ are initialized using random $\{\theta_h^\mu\}_{h=1}^H$ and θ^Q, respectively (line 1-2). The target actor networks $\{\mu_h'(\boldsymbol{o_h}|\theta_h^{\mu'})\}_{h=1}^H$ and target critic network $Q'(s, \boldsymbol{u}|\theta^{Q'})$ are initialized with the same parameters as the online networks (line 3). The replay buffer B is initialized as a circular array with a fixed size for agent learning (line 4).

Then, we begin to train the multiple agents under the historical TM set $\boldsymbol{D}$. To better explore the action space, the Ornstein-Uhlenbeck (OU) process $\boldsymbol{N}$ is initialized for action exploration and parameter ϵ is initialized to 1.0, which is used to balance action exploration and exploitation (line 6-7). At the beginning of each episode, through function get_ospf_flow and get_state, we calculate the initial state s_0 (line 9-10). Here, get_ospf_flow function derives the link load distribution $F_{i,0}$ with flows are routed on the shortest paths according to OSPF protocol and get_state function calculates the state s_0 according to Eq. (1) given the traffic distribution $F_{i,0}$.

For each step t, the agent h computes its own action a_t^h based on the actor network $\mu(o_t^h|\theta_h^\mu)$ with a local partial observation o_t^h and a discount OU noise $\epsilon\mathcal{N}_{(n-1)T+t}^h$ (line 12-14). With the joint action $\boldsymbol{u_t} = \{a_t^h\}_{h=1}^H$, we can derive the routing policy P through the get_policy function and obtain the MLU of the hybrid SDN through get_state function (line 16-18). Then, reward r_t can be computed by Eq. (3) in get_reward function (line 19). Here, $done_t$ implies whether the agent can get more reward in the rest of the steps (line 20-24). Afterwards, transitions in the form of $(\{o_t^h\}_{h=1}^H, \boldsymbol{u_t}, s_t, r_t, s_{t+1}, done_t)$ are deposited in the replay buffer B (line 25). Finally, the critic and actor networks are updated on the randomly sampled minibatch B' with M transitions from the replay buffer B (line 26-31).

Specifically, for the online critic network, the parameter θ^Q is updated by minimizing the loss between the cumulative reward y_m and the estimated reward $Q(s_m, \boldsymbol{u_m}|\theta^Q)$ shown as follows.

$$L_Q(\theta^Q) = \frac{1}{M}\sum_{m=1}^M (y_m - Q(s_m, \boldsymbol{u_m}|\theta^Q))^2 \quad (5)$$

To encourage multiple agents to explore the good action, we introduce the idea of difference reward assignment into agent training. In actor-critic approaches, the reward of each agent is denoted as $A^h(s_t, \boldsymbol{u})$ shown in Eq. (6). In Eq. (6), the first term on the right side $Q(s_t, \boldsymbol{u})$ estimates the Q-value for the joint action $\boldsymbol{u}$ taken on the state of agent h. The advantage function $A^h(s_t, \boldsymbol{u})$ is obtained by comparing the Q-value of the current action a_t^h to a counterfactual baseline that marginalizes a_t^h while keeping the actions of other agents u^{-h} fixed.

$$A^h(s_t, \boldsymbol{u}) = Q(s_t, \boldsymbol{u}) - \int Q(s_t, \boldsymbol{u'} = (\boldsymbol{u_t}, \boldsymbol{u_{t-1}^{-h}}))d\boldsymbol{u'} \quad (6)$$

Because the second integral term on the right is difficult to calculate, we adopt to Monte Carlo method [28] for sampling different actions in action space and calculate the Q value through the critic network.

Meanwhile, the parameter θ_h^μ of the online actor network for agent h is updated by the sampled policy gradient as follows.

$$\nabla J_\mu(\theta_h^\mu) \approx \frac{1}{M}\sum_{j=1}^M A^h(s_j^h, \boldsymbol{u})\nabla_{\theta_h^\mu}\mu_h(o_m|\theta_h^\mu) \quad (7)$$

For the target actor network and critic network, the parameters $\theta^{Q'}$ and $\theta_h^{\mu'}$ will be softly updated as follows.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \quad (8)$$

$$\theta_h^{\mu'} \leftarrow \tau\theta_h^\mu + (1-\tau)\theta_h^{\mu'} \quad (9)$$

When the offline training process terminates, the outputs of Algorithm 1 are the trained actor networks $\boldsymbol{\mu}$, which are the DNNs that have well learnt the mapping between the traffic demands and the routing policy.

4.3 Online Routing Inference

When offline agents training finishes, we deploy the trained routing agents on the SDN switches. When new traffic demands arrive, each agent generates the corresponding routing policy P in an online manner through the inference from the trained actor network as shown in Algorithm 2. The inputs to the algorithm are the network topology G, the set of trained actor networks $\boldsymbol{\mu}$, the set of available links L, the current TM D_i, and the number of agents H. The learnt agents can derive the routing policy P based on the input information in T steps.

Algorithm 2 Online routing inference

Input: $G = (V, E)$, $\boldsymbol{\mu}$, L, D_i, H
Output: P, U_{T-1}^{max}
1: $F_{i,0}$ =get_ospf_flows(G, D_i)
2: $(s_0, U_{i,0}^{max})$ =get_state($F_{i,0}$)
3: **for** step $t = 0$ to $T-1$ **do**
4:     $\boldsymbol{u_t} = \{\mu_h(o_t^h|\theta_h^\mu)\}_{h=1}^H$;
5:     P = get_policy($\boldsymbol{u_t}, L$);
6:     $F_{i,t+1}$ = get_flows(G, D_i, P) ;
7:     $(s_{t+1}, U_{i,t+1}^{max})$ = get_state($F_{i,t+1}$);
8:     r_t = get_reward($U_{i,t+1}^{max}, U_{i,0}^{max}$);
9: **end for**
10: **return** P, U_{T-1}^{max}

---

5 EVALUATION

To demonstrate the superior performance of CMRL, we conduct extensive experiments on different network topologies and traffic datasets. In this section, we first introduce the environmental setup, including the dataset and baseline methods in section 5.1. Then, we present the experimental results and analysis of various methods on TE performance under different traffic demands and network failure scenarios in section 5.2.

5.1 Experimental Setup

The simulation experiments are executed on a workstation with eight Intel cores of 2.4GHz, a RAM of 256 GB and a NVIDIA GeForce RTX 3090 GPU. Our method is implemented on tensorflow. During the offline learning, the size of minibatch M and the discount factor γ are set to 32 and 0.9. The size of replay buffer B and the term τ are set to 8000 and 0.001. The learning rates of the online actor and critic nets are set at 1×10^{-3} and 2×10^{-3}. In addition, we set the number of episodes N to 160.

5.1.1 Dataset

The experimental evaluation is carried out on three different network topologies: Abilene (12 nodes, 30 links), CERNET (14 nodes, 32 links) and GÉANT (23 nodes, 74 links), which are the research and education networks of America, China and European, respectively. The traffic demands datasets of Abilene and CERNET are provided by TOTEM [29] and Zhang [30], which are measured every 5 minutes. The traffic demands dataset of GÉANT is provided by Uhlig [31], which are measured every 15 minutes. In our experiments, 1024 (80%) TMs are used for offline training and 256 (20%) TMs are used for online inference.

5.1.2 Baseline

To exhibit the superiority of the proposed CMRL method, we conduct the comparative experiments with the following methods.

- **OSPF [32]:** This method routes the network traffic according to the OSPF protocol. The traffic is routed on the shortest paths between source and destination node pair.
- **ROAR [10]:** This method is a single-agent reinforcement learning approach that trains an intelligent routing agent with DDPG for minimizing the MLU of the hybrid SDN.
- **MARL [25]:** This method adopts to a general multi-agent reinforcement learning solution that learns the routing policy of each SDN switch without difference reward assignment for multiple agents.

5.2 Experimental Results

5.2.1 Parameter analysis

The number of iteration times T is an important parameter in online routing inference. We conduct experiments to evaluate the average MLU under different T values as shown in Fig. 4. As shown in Fig. 4, we can observe that with the increasing of T, the average MLU first decreases, then increases and finally becomes flat. When $T = 2$, the average MLU is the minimum. The reason is larger T influences trained agent for finding the optimal routing policy. We set T to 2 in the following experiments.

5.2.2 Convergence analysis

To demonstrate our proposed method CMRL converges, we plot the curve of training loss varies with the increasing of training episode in Fig. 5. As shown in Fig. 5, the training loss curves fluctuate with the increasing of training episodes. For the loss curve of CMRL, when the iteration
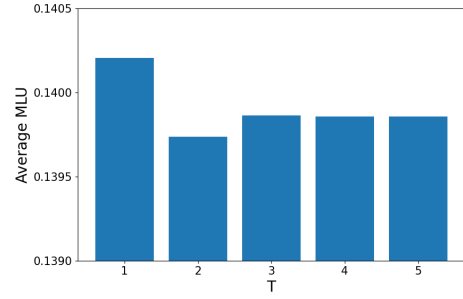

Fig. 4. The average MLU under different values of T.

time is smaller than 1000, the loss value of CMRL fluctuates violently. When the iteration time is larger than 1000, the loss curve becomes relatively flat and stays at a low value. The CMRL method converges when the iteration time reaches 1000. For the loss curve of MARL, the loss value fluctuates drastically at first. When the iteration time is larger than 3000, the loss curve becomes relatively flat and stays at a low value. The MARL method converges when the iteration time reaches 3000. The experiment demonstrates that our proposed method CMRL converges faster than the general MARL method.
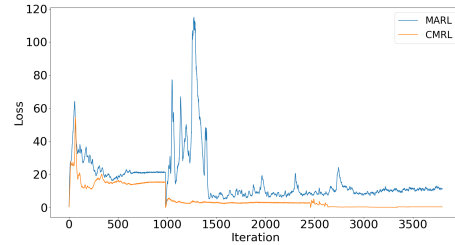

Fig. 5. The training loss varies with the increasing of iteration times.

5.2.3 Network performance under dynamic traffic demands

To comprehensively evaluate the TE performance of various methods under dynamic traffic demands, we plot the Cumulative Distribution Function (CDF) curves of MLU under different SDN deployment ratios and different network topologies in Fig. 6, Fig. 7 and Fig. 8. As shown in Figs. 6-8, we can observe that the CDF curves of CMRL stay above the CDF curves of OSPF, ROAR and MARL under Abilene, CERNET and GEANT network topologies with SDN deployment ratios set to 0.2, 0.3 and 0.4, respectively. This demonstrates that compared to the MLU derived in other methods, our proposed method CMRL can obtain a lower MLU under different network topologies and SDN deployment ratios. In addition, with the increasing of network scale, the gap between CMFL and the other methods becomes larger. This is because in large network topologies, there are more adjacent links for each switch and the agents have more flexibility in optimizing the distribution of flows through the SDN switch.

The reason why CMRL outperforms OSPF in minimizing MLU is that the traffic is constrained to routed on the shortest paths between the source-destination node pair in
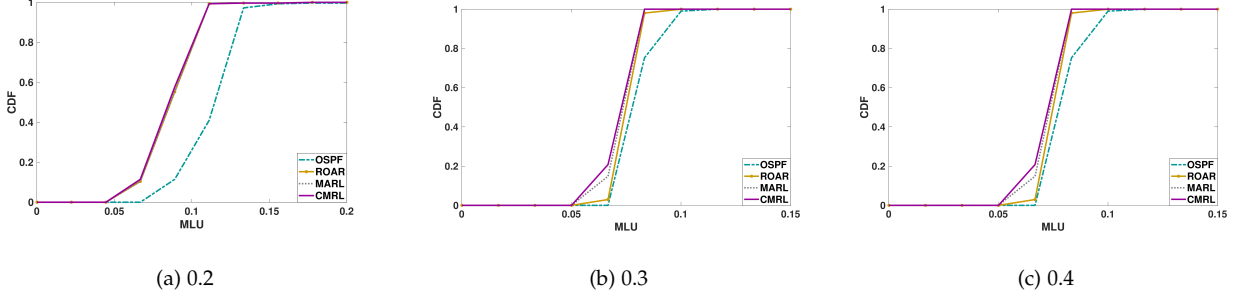
(a) 0.2   (b) 0.3   (c) 0.4

Fig. 6. The CDF curves of MLU under different SDN deployment ratios in Abilene.

(a) 0.2   (b) 0.3   (c) 0.4

Fig. 7. The CDF curves of MLU under different SDN deployment ratios in CERNET.

(a) 0.2   (b) 0.3   (c) 0.4

Fig. 8. The CDF curves of MLU under different SDN deployment ratios in GÉANT.

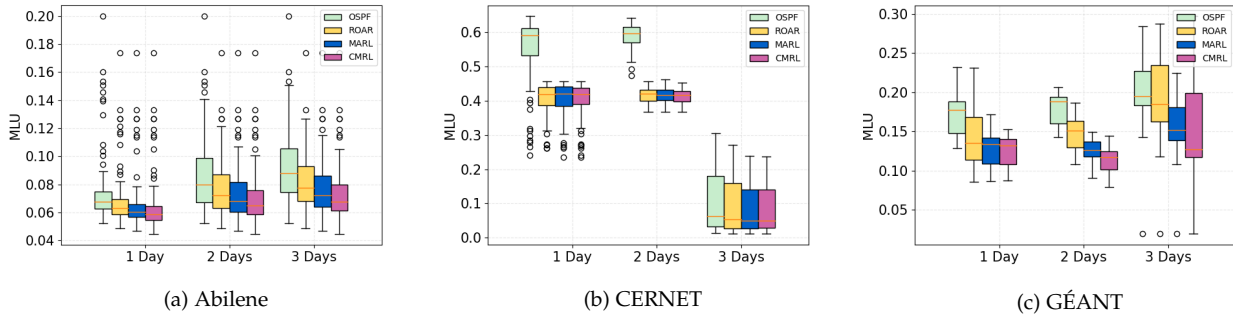(a) Abilene   (b) CERNET   (c) GÉANT

Fig. 9. The MLU under different time periods. We evaluate the performance under 288TMs (1 day), 576TMs (2 days) and 864TMs (3 days) for Abilene and CERNET topologies. We evaluate the performance under 96TMs (1 day), 192TMs (2 days) and 288TMs (3 days) for GÉANT topology.

OSPF, while in our method CMFL, traffic can be split on multiple available paths from the source to the destination, thus achieving better link load balance. Meanwhile, compared to single-agent RL method ROAR, our method has superior performance. The reason is the collaboration of multiple agents can better learn the mapping between traffic demands and routing policies. When new traffic demands arrive, the trained agents can better infer the routing policies with a higher TE performance. Additionally, compared to general MARL, the TE performance of CMRL can be greatly improved with the introduction of difference reward assignments, which encourages multiple agents to take better actions.

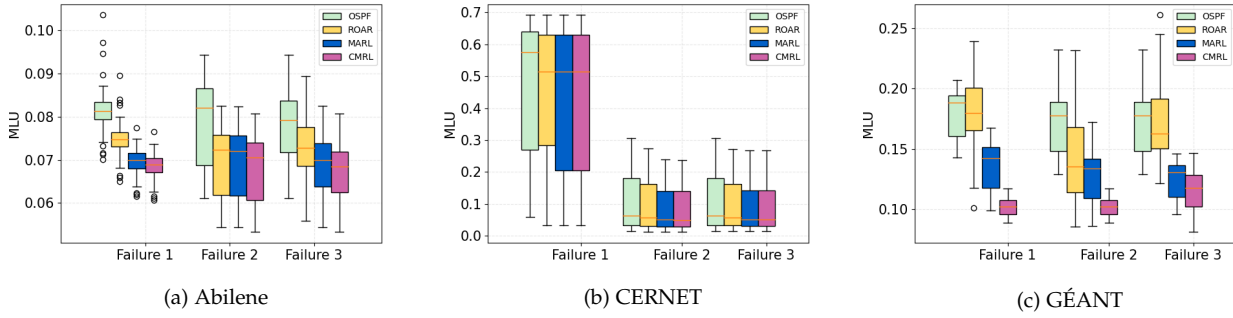To exhibit the concrete performance improvement ratio

Fig. 10. The MLU of network failure scenarios in different network topologies.

TABLE 1
Average MLU under different traffic demands

Method	Abilene	CERNET	GÉANT
OSPF	0.124(32.50%)	0.108(19.35%)	0.182 (41.21%)
ROAR	0.0848(1.29%)	0.0973(10.48%)	0.146 (26.71%)
MARL	0.0847(1.18%)	0.0989(11.93%)	0.140 (23.57%)
CMRL	0.0837	0.0871	0.107

of CMRL, we compute the average MLU of different algorithms under different network topologies in TABLE 1. As shown in TABLE 1, we can observe that compared to OSPF, our proposed method CMFL improves the network performance up to 41.21% under different network topologies; compared to ROAR, our proposed method improves the network performance up to 26.71% under different network topologies; compared to MARL, our proposed method improves the network performance up to 23.57% under different network topologies.

In addition, we conduct extensive experiments to comprehensively evaluate the TE performance under different number of TMs in Fig. 9. Each box contains the 5%-quantile, 25%-quantile, median value, 75%-quantile and the outliers. As shown in Fig. 9(a), we can observe that compared to other methods, our proposed method CMRL obtains a smaller 5%-quantile, 25%-quantile, median value, 75% MLU value under TMs of different time periods in Abilene topology. The results are similar in CERNET and GÉANT topologies. Through the extensive experiments, we can conclude that CMRL can efficiently learn the routing policies through the collaboration of multiple agents and the network performance can be better enhanced in CMRL, compared with other TE methods.

5.2.4 TE performance under network failures

TABLE 2
Average MLU under link failure scenarios

Method	Abilene	CERNET	GÉANT
OSPF	0.0790(14.94%)	0.460(7.83%)	0.173(32.95%)
ROAR	0.0689(24.67%)	0.442(4.07%)	0.170(31.76%)
MARL	0.0687(2.18%)	0.430(1.40%)	0.125(7.20%)
CMRL	0.0672	0.424	0.116

Network failures, especially single link failures, happen frequently in large networks and can lead to severe network congestion and packet loss [33]. To validate the superior performance of CMRL in handling network failures, we evaluate the MLU of the hybrid SDN under different single link failures and draw the box diagrams in Fig. 10. As shown in Fig. 10(a), we can observe that under different link failures, our proposed method CMRL can obtain a smaller MLU compared to the MLU derived from the other methods in Abilene. In CERNET and GÉANT shown in Fig. 10(b) and 10(c), we obtain the similar results. In particular, the gap between our method CMFL and other baseline methods becomes larger in the GÉANT network. In summary, our proposed method CMFL exhibits superior performance in minimizing MLU of the hybrid SDN. As shown in TABLE 2, our proposed method can reduce the average MLU up to 32.95%, 31.76% and 7.20% compared to OSPF, ROAR and MARL, respectively, under different network topologies. This demonstrates that CMRL is robust to network failures and can infer an efficient routing policy when link failures happen.

5.2.5 Online inference time

Finally, we record the online inference time of different methods in TABLE 3. As shown in TABLE 3, we can observe that CMRL has a shorter online inference time compared to ROAR and the inference time of CMRL approximates that of MARL. This demonstrates that compared to ROAR, the computation time can be reduced in CMRL by dividing a large-scale inference problem into multiple small-scale inference problems. The trained agents in CMRL can promptly react to the dynamic changing environment in an online manner when traffic demands change or network failures happen.

TABLE 3
The online inference time

Method	Abilene	CERNET	GÉANT
ROAR	0.878ms	0.891ms	1.203ms
MARL	0.534ms	0.547ms	0.996ms
CMRL	0.531ms	0.545ms	0.996ms

6 CONCLUSION

In this paper, we innovatively propose a multi-agent reinforcement learning framework CMRL for improving the TE

performance in a dynamic hybrid SDN environment. Specifically, an interactive environment is first constructed and the multiple agents are trained offline for collaboratively learning the map between traffic demands and routing policies. To solve the credit assignment of multi-agent, difference reward assignment is introduced for encouraging the agents to sacrifice for the good actions. Then, the trained agents are deployed for online routing policy inference. The extensive experiments demonstrate the superior performance of CMRL in reducing MLU of hybrid SDN when traffic demands change or network failures happen.

7 ACKNOWLEDGEMENTS

REFERENCES

[1] Y. Xiao, J. Liu, J. Wu, and N. Ansari, "Leveraging deep reinforcement learning for traffic engineering: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2064–2097, 2021.

[2] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[3] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," in *Proceedings of ACM SIGCOMM*. ACM, 2014, pp. 70–75.

[4] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFO-COM*. IEEE, 2013, pp. 2211–2219.

[5] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in sdn/ospf hybrid network," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 563–568.

[6] Y. Guo, Z. Wang, Z. Liu, X. Yin, X. Shi, J. Wu, Y. Xu, and H. J. Chao, "Sote: Traffic engineering in hybrid software defined networks," *Computer Networks*, vol. 154, pp. 60–72, 2019.

[7] 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 Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1012–1028, 2019.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[9] Y. Tian, Z. Wang, X. Yin, X. Shi, Y. Guo, H. Geng, and J. Yang, "Traffic engineering in partially deployed segment routing over ipv6 network with deep reinforcement learning," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1573–1586, 2020.

[10] Y. Guo, W. Wang, H. Zhang, W. Guo, Z. Wang, Y. Tian, X. Yin, and J. Wu, "Traffic engineering in hybrid software defined network via reinforcement learning," *Journal of Network and Computer Applications*, p. 103116, 2021.

[11] Q. Xu, Y. Zhang, K. Wu, J. Wang, and K. Lu, "Evaluating and boosting reinforcement learning for intra-domain routing," in *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2019, pp. 265–273.

[12] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[13] J. Moy *et al.*, "Ospf version 2," 1998.

[14] D. Oran, "Rfc1142: Osi is-is intra-domain routing protocol," 1990.

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

[16] M. Ericsson, M. G. C. Resende, and P. M. Pardalos, "A genetic algorithm for the weight setting problem in ospf routing," *Journal of combinatorial optimization*, vol. 6, no. 3, pp. 299–333, 2002.

[17] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[18] M. Kodialam and T. Lakshman, "Network link weight setting: A machine learning based approach," in *IEEE INFOCOM 2022*. IEEE, 2022, pp. 2048–2057.

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

[20] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *In Proceedings of the ACM SIGCOMM*. ACM, 2013, pp. 3–14.

[21] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang, "Incremental deployment and throughput maximization routing for a hybrid sdn," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1861–1875, 2017.

[22] 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*. IEEE, 2018, pp. 1871–1879.

[23] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "Rl-routing: An sdn routing algorithm based on deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3185–3199, 2020.

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

[26] N. Geng, M. Xu, Y. Yang, C. Liu, J. Yang, Q. Li, and S. Zhang, "Distributed and adaptive traffic engineering with deep reinforcement learning," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

[27] 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.

[28] A. Shapiro, "Monte carlo sampling methods," *Handbooks in operations research and management science*, vol. 10, pp. 353–425, 2003.

[29] S. Balon and G. Monfort, "The traffic matrices and topology of the abilene network," 2019.

[30] B. Zhang, J. Bi, J. Wu, and F. Baker, "Cte: cost-effective intra-domain traffic engineering," in *Proceedings of the ACM SIGCOMM*, vol. 44, no. 4. ACM, 2014, pp. 115–116.

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

[32] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 2, 2000, pp. 519–528 vol.2.

[33] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proceedings of the ACM SIGCOMM*, 2014, pp. 527–538.