



Traffic Engineering in Hybrid Software Defined Network via Reinforcement Learning

Yingya Guo^{a,b}, Weipeng Wang^{a,b}, Han Zhang^{c,*}, Wenzhong Guo^{a,b}, Zhiliang Wang^c, Ying Tian^d, Xia Yin^d, Jianping Wu^d

^a College of Mathematics and Computer Science, Fuzhou University, China

^b Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, China

^c Institute for Network Sciences and Cyberspace, Tsinghua University, China

^d Department of Computer Science and Technology, Tsinghua University, China

ARTICLE INFO

Keywords:

Traffic engineering
Hybrid software defined network
Routing optimization
Reinforcement learning

ABSTRACT

The emergence of Software Defined Network (SDN) provides a centralized and flexible approach to route network flows. Due to the technical and economic challenges in upgrading to a fully SDN-enabled network, hybrid SDN, with a partial deployment of SDN switches in a traditional network, has been a prevailing network architecture. Meanwhile, Traffic Engineering (TE) in the hybrid SDN has attracted wide attentions from academia and industry. Previous studies on TE in the hybrid SDN are either traffic-oblivious or time-consuming, which causes routing schemes failed in responding to the dynamically-changing traffic rapidly and intelligently. Therefore, in this paper, we propose a Reinforcement Learning (RL) based method, which learns a traffic-splitting agent to address the dynamically-changing traffic and achieve the link load balancing in the hybrid SDN. Specifically, to rapidly and intelligently determine a routing scheme to the new traffic demands, a traffic-splitting agent is designed and learnt offline by exploiting the RL algorithm to establish the direct relationship between traffic demands and traffic-splitting policies. Once the traffic-splitting agent is learnt, the effective traffic-splitting policies, which are used to determine the traffic-splitting ratios on SDN switches, can be generated rapidly. Additionally, to meet the interactive requirements for learning a traffic-splitting agent, a reasonable simulation environment is proposed to be constructed to avoid routing loops when traffic-splitting policies are taken. Extensive evaluations on different topologies and real traffic demands demonstrate that the proposed method achieves the comparable network performance and performs superiorities in rapidly generating the satisfying routing schemes.

1. Introduction

With the increasing popularity of Internet of Everything (IoE), various emerging Internet applications have been bringing the explosive network traffic (Cisco, 2020). To meet the massive requirements of traffic transmission, Internet Service Provider (ISP) has been expanding the investment on the establishment of network infrastructure to guarantee a high-bandwidth and low-latency network. Meanwhile, there is an urgent requirement to develop the technologies on network management. Traffic Engineering (TE), as an efficient network management tool, dedicates to achieve load balancing and improves network performance by optimizing the traffic routing schemes (Mendiola et al., 2016). Therefore, the researches on TE have attracted wide attention.

Software Defined Network (SDN) is an emerging network architecture that decouples control plane from data plane (Amin et al., 2018). In this novel architecture, there are two main components, i.e. SDN controller and SDN switches. The SDN controller in the control plane has a global view of the network status by collecting traffic information from SDN switches in the data plane. Thus, it can centrally manage fine-grained forwarding behaviors of flows through dispatching flow entries to SDN switches (Open Networking Foundation, 2012). Here, the flow refers to an aggregation of the traffic identified by the source-destination address pair. According to the OpenFlow protocol (McKeown et al., 2008), the SDN controller can flexibly split the aggregated flows and set the outgoing links of the split flows by modifying or adding flow entries in the SDN switches (Liu et al., 2014; Guo

* Corresponding author. Institute for Network Sciences and Cyberspace, Tsinghua University, China.

E-mail addresses: guoyingya90@163.com, guoyy@fzu.edu.cn (Y. Guo), zhhan@tsinghua.edu.cn (H. Zhang).

et al., 2019). Compared with shortest-path-based routing in traditional distributed routing protocols, e.g. Open Shortest Path First (OSPF) (Moyet et al., 1998), SDN controller exhibits great potential in providing a more flexible, efficient and intelligent routing by customizing the forwarding rules for each flow.

Because of the benefits brought by SDN, ISPs are increasingly expected to perform the transition from a legacy network to a SDN-enabled network. However, migrating a traditional distributed network to a fully SDN-enabled network encounters technical and economic challenges (Vissicchio and VanbeverOlivier, 2014), especially for large-scale networks. Specifically, the SDN hardware and software have not been fully tested, which raises concerns on the stability and reliability of SDN devices (Xu et al., 2017a). This means that upgrading the entire network infrastructure with customized SDN devices may bring potential risks of security and instability. Moreover, upgrading to a fully SDN deployed network will inevitably impose a huge capital expenditure in purchasing SDN devices. In order to mitigate those challenges, ISPs tend to incrementally deploy the SDN devices rather than upgrading the entire network at one time. Therefore, a hybrid SDN, partially deploying SDN switches in a traditional network, is generated (Xu et al., 2017a; Hong et al., 2016; Guo et al., 2014; Agarwal et al., 2013). The hybrid SDN, where traditional legacy routers and SDN switches coexist, not only has the robustness of traditional distributed networks, but also owns the flexibility of centralized SDN-enabled networks. In recent years, the hybrid SDN has become a prevailing network architecture. Moreover, the studies on TE in the hybrid SDN have attracted widespread attention from academia and industry (Sinha Haribabuet al., 2017).

The routing schemes designed by the previous studies on TE are either traffic oblivious or centralized control. Specifically, on the one hand, in the OSPF protocol, shortest-path-based routing methods ignore the load balancing and utilization of the entire network when designing the routing schemes (Fortz and Thorup, 2000, 2002, 2004; Ericsson-Mauricio et al., 2002; Wang et al., 2001; Sridharan et al., 2005). Moreover, those methods cannot make self-adaptive adjustments to the routing schemes when encountering the fluctuating traffic demands. On the other hand, many routing methods adopt a flexible centralized-routing manner to handle various traffic demands by dynamically determining the routing schemes (Xu et al., 2017a; Hong et al., 2016; Guo et al., 2014, 2017a; Agarwal et al., 2013; Tian et al., 2020). However, the determination of the dynamic routing often requires to centrally solve a large-scale optimization problem, which is time-consuming to obtain its solution. This causes the failure of these centralized-routing methods in responding to the traffic bursts promptly.

Recent breakthrough of Reinforcement Learning (DRL) have promoted a tremendous progress in various fields, e.g. game playing (Xiang and Su, 2021), natural language processing (Sharma and Kaushik, 2017), automatic driving (Zhang et al., 2018), etc. As an essential branch of Machine Learning (ML), Reinforcement Learning (RL) attempt to solve decision-making problems through continuous learning by interacting with the environment in a trial and error manner. During the learning procedure, past experiences are generalized to new situations by learning an RL agent, which establishes a direct mapping from representations of the environment to the decision policies (Zhu et al., 2017). Once the mapping is established, the efficient decision policy can be determined rapidly. Moreover, the advent of Neural Networks (NN) enables massive data encoded in a compact and effective representation. The integration of RL and NN can better describe the complex scenarios and exhibit superior performance on realizing an adaptive and fast decision-making.

The advantages of RL exhibit its great potentials to realize intelligent TE so that the dynamic traffic demands can be handled timely in the hybrid SDN. However, leveraging RL to adaptively optimize routing schemes in the hybrid SDN still faces challenges. Firstly, because of co-existence of legacy routers and SDN switches in the hybrid SDN, different routing constraints on the different devices should be

considered when learning an RL agent. Specifically, traffic flows on legacy routers are forwarded to the next hop according to the routing table entries indicated by OSPF protocol. In order to enable a flexible traffic splitting, traffic flows on SDN switches are forwarded to the next hop according to the flow entries dispatched by SDN controllers. The routing constraints for the legacy routers and SDN switches are different, which poses a great challenge to design an effective RL-based approach to learn an RL agent. Secondly, the RL agent is gradually trained by repeatedly interacting with the environment. The flexibility of traffic flows on the SDN switches may cause unnecessary routing loops. Thus, a reasonable and correct emulation environment should be established for learning the RL agent. Thirdly, to achieve the intelligent and rapid generation of routing schemes, the design of the RL agent should be carefully considered to handle the dynamically-changing traffic demands promptly.

In this paper, we mainly focus on TE in the hybrid SDN and propose a new RL-based method named ROAR (Routing OptimizAtion with Reinforcement learning) to conduct the routing optimization for achieving link load balancing in the hybrid SDN. Specifically, we first formulate the TE problem in the hybrid SDN as a math programming problem. Then, a hybrid SDN is constructed from a traditional distributed network by incrementally upgrading legacy routers with SDN switches and determining the link weights under the distributed routing protocols. After that, we design and train a traffic-splitting agent on a set of traffic demands and a constructed simulation environment offline. Here, the traffic-splitting agent learns the direct relationship between the network states and the traffic-splitting policies. Finally, the learnt traffic-splitting agent can intelligently and rapidly generate routing schemes when the traffic demand is changed in the hybrid SDN. To demonstrate the superiority of our proposed ROAR method, extensive evaluations and comparative experiments are conducted on various topologies and real traffic datasets. The main contributions of this paper can be summarized as follows:

- To study the TE problems in the hybrid SDN, we propose a new method ROAR to intelligently and rapidly conduct the routing optimization while the traffic demands are changing dynamically. Specifically, to determine a satisfying routing scheme that adapts to new traffic demands intelligently and rapidly, an RL algorithm is introduced to learn a traffic-splitting agent in a hybrid SDN. The learnt agent is designed to establish the direct relationship between traffic demands and routing policies.
- To meet the interactive requirements from learning a traffic-splitting agent, we propose a method to construct a reasonable simulation environment. In the constructed environment, Directed Acyclic Graphs (DAGs) are built to avoid the routing loops when the traffic-splitting policies are taken by the RL agent. Additionally, the different routing constraints on legacy routers and SDN switches are satisfied in the constructed environment.

The reminder of this paper is organized as follows: Section 2 presents the related work of TE. Section 3 formulates the TE problem in the hybrid SDN. Section 4 presents the construction of a hybrid SDN from a traditional network. Section 5 details our proposed ROAR method to solve the TE problem in the hybrid SDN. Section 6 reports extensive experimental results and evaluates the performance of the proposed ROAR method on real traffic datasets and different network topologies. Finally, Section 7 gives the concluding remarks and hints at plausible future research.

2. Related work

In this section, we introduce the related works on the TE problem in different network architectures.

In the traditional distributed network, OSPF (Moyet et al., 1998) and Intermediate System-to-Intermediate System (IS-IS) (Oran, 1990) are

widely used as Interior Gateway Protocols (IGP). In the OSPF and IS-IS protocols, each link is set with a weight (or a cost), and traffic flows choose their next hops according to the shortest paths which are determined by the link weights. To improve the network performance in the traditional distributed network, many heuristic methods were proposed to optimize the link weight settings to minimize the network's Maximum Link Utilization (MLU) (Fortz and Thorup, 2000; Ericsson-Mauricio et al., 2002; Altun et al., 2013). However, the routing flexibility is greatly constrained by the shortest-path-based routing, which is not beneficial to improve the network performance.

The emergence of SDN architecture enables a flexible routing by dispatching flow entries into SDN switches. Through deploying SDN devices, the Wide Area Network (WAN) in Microsoft (Hong et al., 2013) and Google (Jain et al., 2013) have achieved near 100% network utilization by flexibly routing traffic flows according to their priorities. To upgrade a traditional ISP network to a hybrid SDN, Agarwal et al. (2013) selected a part of legacy nodes to be deployed as SDN nodes. Based on the built hybrid SDN, a FPTAS method was proposed to determine the optimal splitting ratio of SDN nodes for improving the overall network throughput. To minimize the MLU in a hybrid SDN, Hong et al. (Hong et al., 2016; Caria et al., 2015; He and Song, 2015) attempted to design heuristic routing algorithms to optimize the splitting ratio of traffic flows on SDN nodes. To achieve the throughput-maximization routing in hybrid SDN, Xu et al., 2017a, 2017b integrated a depth-first-search method with a randomized rounding mechanism to optimize the flow routing problem. To further improve the network performance in a hybrid SDN, Guo et al. (2014) proposed to jointly optimize the OSPF link weights and the splitting ratio of SDN nodes for the minimization of MLU. To reduce the abundant flow entries installed into the SDN switches, the path cardinality constraints were considered into optimizing the network performance in the hybrid SDN (Ren et al., 2020; Guo et al., 2021). To improve the network performance under different TMs in a hybrid SDN, an expected TM was calculated and exploited to reflect the fluctuating TMs over a period of time (Guo et al., 2017b).

With the advent of the big data era, Machine Learning (ML) and Deep Learning (DL) have been applied in various fields, e.g. computer vision (Wang et al., 2021), autonomous driving (Ravi Kiran et al., 2021), natural language processing (Zhang et al., 2020a), etc. In recent years, a lot of researchers have begun to attempt use ML and DL to solve TE problems in the network. To minimize the MLU of the network, Valadarsky et al. (2017) proposed to introduce Reinforcement Learning (RL) to find a strategy for setting the link weights which determine the routing scheme. To search a TE solution to improve the network performance, Zhang et al. (2020b) designed an RL-based scheme that searches a rerouting policy to intelligently route critical flows for balancing network flows. To model the high-dimensional and continuous states of the network, Deep Reinforcement Learning (DRL), which integrates RL with DL, was exploited to intelligently optimize routing schemes and exhibited the comparable performance on throughput (Xu et al., 2018; Sun et al., 2019). To reduce the transmission delay, Sun et al. (2020) leveraged the pinning control theory to select the critical links and developed a DRL-based algorithm to optimize the weight settings of the selected critical links dynamically. To guarantee the Quality of Service (QoS) in a hybrid SDN, a DRL-based algorithm was proposed to model the multi-splittable routing problem with a DRL framework (Huang et al., 2021). To achieve TE in a partially deployed Segment Routing (SR) network, Tian et al. (2020) proposed a WA-SRTE method to use DRL as an optimizer, which optimizes the link weights offline, and the online routing schemes were obtained by solving a complex Linear Programming (LP) problem. The previous studies on the use of RL to solve TE problems attempted to treat RL as an offline optimizer. However, our study exploits RL to learn a traffic-splitting agent offline, and the learnt agent, which is directly deployed on the SDN controller, is able to promptly and intelligently generate the traffic-splitting policies online.

3. Network model

In this section, we present the network model of the TE problem in a hybrid SDN with dynamic traffics. Our network topology is represented as an undirected graph $G = (V, E)$, where V is the set of forwarding devices, which are composed of SDN switches V_s and legacy routers V_l , and E is the link set. Here, the capacity of link $e \in E$ is denoted as $C(e)$. The set of historical Traffic Matrices (TMs) is denoted by $\mathbf{D} = \{D_1, D_2, \dots, D_n\}$ and r_i is the weight coefficient of TM D_i . The element $D_i(u, v)$ in D_i represents the traffic demand from device u to device v . Here, $u, v \in V$. ω is the link weight configuration under the OSPF protocol. Variable f denotes the splitting ratio of traffic flows at the SDN switches. The related notations are summarized in Table 1.

Given a set of TMs $\mathbf{D}$, the goal of TE on the hybrid SDN is to improve the network performance by minimizing the Maximum Link Utilization (MLU) for each TM D_i . In the hybrid SDN, both the link weight configuration ω and the splitting traffic flow f on each SDN switch determine the value of MLU. Therefore, we formulate the TE problem on the hybrid SDN as follows:

$$\text{minimize } \sum_{i=1}^n r_i U_i^{\max} \quad (1)$$

$$\sum_{e \in I_q(v)} f_{e,D_i}^q(\omega) + D_i(v, q) = \sum_{e \in O_q(v)} f_{e,D_i}^q(\omega) \quad (2)$$

$$\forall v \in V_s, \forall q \in V, \forall D_i \in \mathbf{D}$$

$$\sum_{q \in V} f_{e,D_i}^q(\omega) \leq U_i^{\max} C(e), \quad \forall e \in E, \forall D_i \in \mathbf{D} \quad (3)$$

$$\sum_{j=1}^n r_j = 1, \quad 0 \leq r_j \leq 1 \quad (4)$$

$$\omega(e) \in \mathbf{N}, \quad \forall e \in E \quad (5)$$

$$0 \leq U_i^{\max} \leq 1, \quad \forall D_i \in \mathbf{D} \quad (6)$$

$$f_{e,D_i}^q(\omega) \geq 0, \quad \forall e \in E, \forall t \in V, \forall D_i \in \mathbf{D} \quad (7)$$

where Eq. (1) is the optimization goal which attempts to minimize the weighted sum of MLUs under the TMs $\mathbf{D}$. This means that TE in the hybrid SDN aims to design routing schemes that performs well on different TMs. Eq. (2) is the flow conservation constraint which guarantees that the traffic flowing into and originating from a SDN node should be equal to the traffic flowing out of this node. This implies that traffic demand should be met. Eq. (3) is the link capacity constraint that the load of a link should be smaller than its capacity. Eq. (4) restricts that the sum of weight coefficient r_i should equal to 1. Eqs. (5)–(7) give the value range of the link weight, MLU and the split traffic, respectively.

Table 1
Definition of notations.

$G = (V, E)$	Undirected graph with node set V and edge set E
$C(e)$	The link capacity of edge e
$\mathbf{D} = \{D_i\}$	The set of Traffic Matrices (TMs)
$D_i(u, v)$	Traffic demand from node u to node v in D_i
r_i	The weight coefficient of TM D_i
$I_q(v)$	The set of ingoing links of node v to destination node q
$O_q(v)$	The set of outgoing links of node v to destination node q
ω	The link weight configuration of E
$\omega(e)$	The link weight of edge $e, e \in E$
$f_{e,D_i}^q(\omega)$	The split traffic on link e to destination node q when TM is D_i and the weight setting is ω
V_s	The set of SDN switches
V_l	The set of legacy routers
$U_i^{\max}$	The MLU under the TM D_i

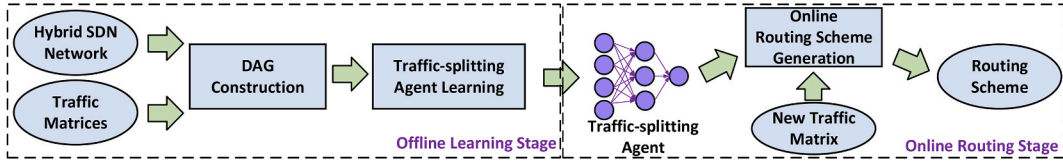


Fig. 1. The workflow of our proposed ROAR method.

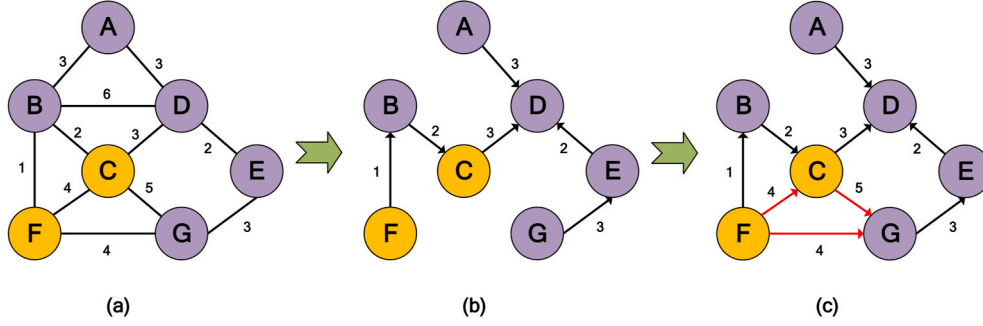


Fig. 2. An example of the procedure of DAG construction. Here, the yellow circle and purple circle represent SDN node and legacy node, respectively. Node D is the destination node. The number on the link denotes the link weight. (a) the topology of hybrid SDN; (b) the links on the shortest path to destination node D are selected into DAG; (c) the final DAG is obtained by further adding the SDN nodes' adjacent links whose addition will not form routing loop.

When the link weight configuration w is fixed, Eqs. (2)–(7) make the TE problem on the hybrid SDN a multi-commodity problem, which is NP-hard (Guo et al., 2017b).

4. Hybrid SDN construction

As a preparation for studying TE problem on the hybrid SDN, we should construct a hybrid SDN from a traditional distributed network first. The construction of a hybrid SDN consists of two steps: SDN nodes deployment and link weight optimization.

4.1. SDN nodes deployment

Given a traditional distributed network $G = (V, E)$, a set of historical TMs $\mathbf{D}$ and a SDN deployment ratio sdn_ratio , a hybrid SDN is constructed by incrementally deploying the SDN nodes (switches) V_s for upgrade. Inspired by (Cianfrani et al., (2017)), the link load is used as an indicator to select the legacy routers to be upgraded.

Specifically, for a node $v \in V$ and a TM $D_i \in \mathbf{D}$, we first calculate the node v 's Most Loaded Link (MLK) as follows:

$$MLK_{D_i}(v) = \max_{e \in E \wedge s(e)=v} U_i(e) \quad (8)$$

where $U_i(e)$ represents the link utilization of edge e when TM D_i is routed under OSPF protocol. $s(e)$ denotes the source node of directed edge e . The value of $MLK_{D_i}(v)$ means the maximum link utilization of node v 's all outgoing edges under the TM D_i .

Secondly, based on the $MLK_{D_i}(v)$, we calculate $score(v)$ to accumulate the value MLK of node v under different TMs by computing Eq. (9). The value of $score(v)$ reflects the overall maximum link utilization of node v 's all outgoing edges under different TMs.

$$score(v) = \sum_{D_i \in \mathbf{D}} MLK_{D_i}(v) \quad (9)$$

Thirdly, we prefer to select the nodes whose outgoing links have higher link utilization as SDN nodes. The reason is that the nodes whose outgoing links have high link utilization is often treated as a bottleneck node and there is a high possibility to reduce MLU with these bottleneck

nodes deployed as SDN nodes. Therefore, we greedily select k nodes with the largest $score$ value to generate the SDN node set V_s . Here, $k = |V| \times sdn_ratio$, $|V|$ is the size of set V . With the SDN node set N_s , the hybrid SDN can be generated by upgrading the nodes in N_s with SDN switches.

4.2. Link weight optimization

The construction of the hybrid SDN is conducted by upgrading the legacy routers with SDN switches incrementally. However, the forwarding behaviors on legacy routers cannot be centrally controlled by the SDN controller. Therefore, the link weights should be carefully determined to ensure a stable and effective routing scheme on legacy routers. Additionally, the stable routing scheme on legacy routers can be benefit to avoid information flooding and transient routing loops. Therefore, as our previous method proposed in (Tian et al., (2020)), a DRL method is introduced to optimize link weight offline for obtaining a stable and effective routing scheme on legacy routers.

Specifically, we first cluster multiple TMs using K-Means Clustering (Ahmed et al., 2020) and treat each cluster centroid as a representative TM. Additionally, the weight of each cluster representative TM is set to be the fraction of the number of TMs in its corresponding cluster to the number of total TMs. Then, we linearly combine the representative TMs with their weights to obtain an expected TM, which reflects the average of dynamic traffic demands. Finally, based on the expected TM, the link weight is optimized and determined by applying the DRL-based weight optimization method proposed in (Tian et al., (2020)).

5. Proposed methodology

As illustrated in Fig. 1, our proposed method ROAR for solving the TE problem of the hybrid SDN contains two stages: offline learning stage and online routing stage. At the offline learning stage, given the network topology of hybrid SDN and traffic information, we first construct the Directed Acyclic Graph (DAG) to ensure a hybrid SDN environment with a loop-free routing for the interaction of the RL agent (Section 5.1). With the constructed DAG, a traffic-splitting agent is trained to establish the direct relationship between the network environment and the routing schemes by leveraging the RL framework (Section 5.2). At the online

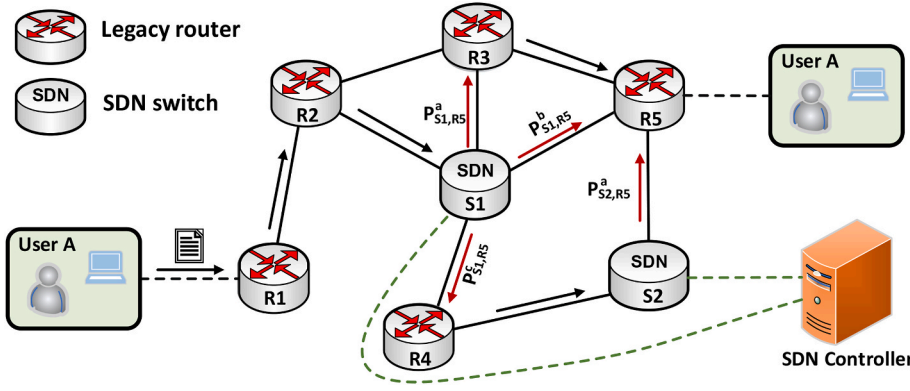


Fig. 3. Illustration of applying traffic-splitting actions in hybrid SDN. To accomplish the transmission of traffic demand and forward the traffic flows from router R1 to router R5, traffic flows on legacy routers are forwarded to next hop according to OSPF protocol (black arrows), and traffic flows on SDN switches are split according to the traffic-splitting policies (red arrows). In each traffic-splitting policy, the sum of splitting ratios on the forwarding links of a SDN node should equal to 1, i.e. $P_{S1,R5}^a + P_{S1,R5}^b + P_{S1,R5}^c = 1$ and $P_{S2,R5}^a = 1$.

routing stage, through deploying the trained traffic-splitting agent, an effective routing scheme can be quickly calculated when traffic demand changes (Section 5.3).

5.1. DAG construction

In general, RL allows agents to continuously interact with an environment to automatically learn a policy that depends on the state of the environment. A reasonable environment should be constructed to meet the interactive requirements from the RL agent. In our proposed ROAR method, considering that the agent requires to deploy traffic-splitting policies on SDN nodes, a loop-free routing should be guaranteed in the hybrid SDN. Therefore, we propose to construct DAGs for avoiding routing loop in the constructed environment.

We first give a definition of *permissible links*. We denote *permissible links* as $PL_u(v)$, which consists of node v 's adjacent links that are legally used in forwarding traffic flows to a destination node $u \in V$. Specifically, for a legacy node $v \in V$, $PL_u(v)$ contains node v 's adjacent links that are on the shortest paths from node v to the destination node u . For a SDN node v , besides node v 's adjacent links on the shortest paths to u , $PL_u(v)$ contains node v 's other outgoing links if those links will not form routing loop in DAGs. By collecting PL_u of all nodes in the hybrid SDN, the DAG to destination node u is constructed. In practice, because each node can be a destination node, the number of constructed DAGs for a hybrid SDN is $|V|$.

To illustrate the procedure of DAG construction, an example is given in Fig. 2. As shown in Fig. 2(b), for every node in the hybrid SDN, we first exploit the Floyd Algorithm to obtain their shortest paths to the destination node D . Then, the links on the shortest paths are added into DAG. Additionally, as shown in Fig. 2(c), for SDN nodes C and F , we further evaluate their remaining adjacent links iteratively. In each iteration, we examine whether there is a routing loop in the graph if the link is added into DAG. If there is no loop, the link is added to DAG. Once the *permissible links* of all nodes are found, the DAG to destination node D is constructed. Note that, in the constructed DAG, there may be multiple routing paths from a source node to a destination node. The multiple routing paths provide multiple choices for forwarding traffic flows, which assists in achieving the link load balancing. Note that the different sequences of adding the outgoing links of SDN nodes into the constructed DAG may generate different DAGs. In practice, we can exhaustively or stochastically search an optimal or satisfied sequence to guarantee the number of routing paths in the constructed DAGs.

5.2. Offline traffic splitting agent learning

With the constructed DAGs as the interactive environment, we propose to learn a traffic-splitting agent that intelligently and quickly controls the distribution of traffic flows in the Hybrid SDN. The traffic

splitting agent aims to obtain an optimal policy that splits the traffic flows on SDN nodes, which assists in achieving the link load balancing. In order to enable the agent to learn decision-making in different network scenarios, we introduce an efficient RL algorithm, i.e. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), into our proposed ROAR method. As an off-policy, actor-critic, model-free learning algorithm, DDPG is able to learn a good decision policy in the continuous action space and continuous state space (Lillicrap et al., 2015). Instead of handcrafting the training data for supervised learning, DDPG allows the agent to gradually accumulate the experience of decision-making by repeatedly interacting with and learning from the environment.

Each step of agent-environment interactions in the proposed ROAR method can be represented as tuple (s_t, a_t, r_t, s_{t+1}) , where s_t is the state of hybrid SDN network at time t , a_t is the traffic splitting action taken by the agent at time t , s_{t+1} is the resulting state of hybrid SDN network after the agent has taken the action a_t , r_t is the reward received by agent when taking the action a_t at the state s_t . A policy $\pi(a_t|s_t) \in [0, 1]$ maps each state to a probability distribution over actions. An optimal policy π^* maximizes the expected cumulative discounted reward $E_{\pi^*}[\sum_t \gamma^t r_t]$, here the term $\gamma \in [0, 1]$ is the discount factor which prevents an infinite sum of accumulated rewards. The goal of the agent is to find π^* based on its experience within the environment.

In DDPG, the traffic-splitting agent is learnt from the set of (s_t, a_t, r_t, s_{t+1}) . Therefore, in the following, we detail the notion of the state, action, and reward used in our proposed ROAR method.

5.2.1. State

To describe the traffic distribution in the hybrid SDN, we attempt to design the state s_t based on the link utilization. Therefore, we define s_t as a vector $[s_t^e | e \in E]$, where s_t^e denotes the utilization of link e at time t . The link utilization s_t^e is calculated as follows:

$$s_t^e = \frac{\sum_{q \in V} f_{e,D_t}^q(\omega)}{C(e)} \quad (10)$$

where D_t is the TM at time t , and $\sum_{q \in V} f_{e,D_t}^q(\omega)$ computes the traffic flowing on link e under the TM D_t . In practice, the distribution of traffic flows on each link can be determined when the TM is given and the traffic-splitting policies are deployed on the SDN nodes.

5.2.2. Action

The action a_t consists of a set of the traffic-splitting policies on SDN nodes. Each traffic-splitting policy can be represented as the traffic-splitting ratios on the forwarding links of a SDN node. The forwarding links of a SDN node can be determined by its adjacent links in the constructed DAG to a specific destination node. As shown in Fig. 3, two

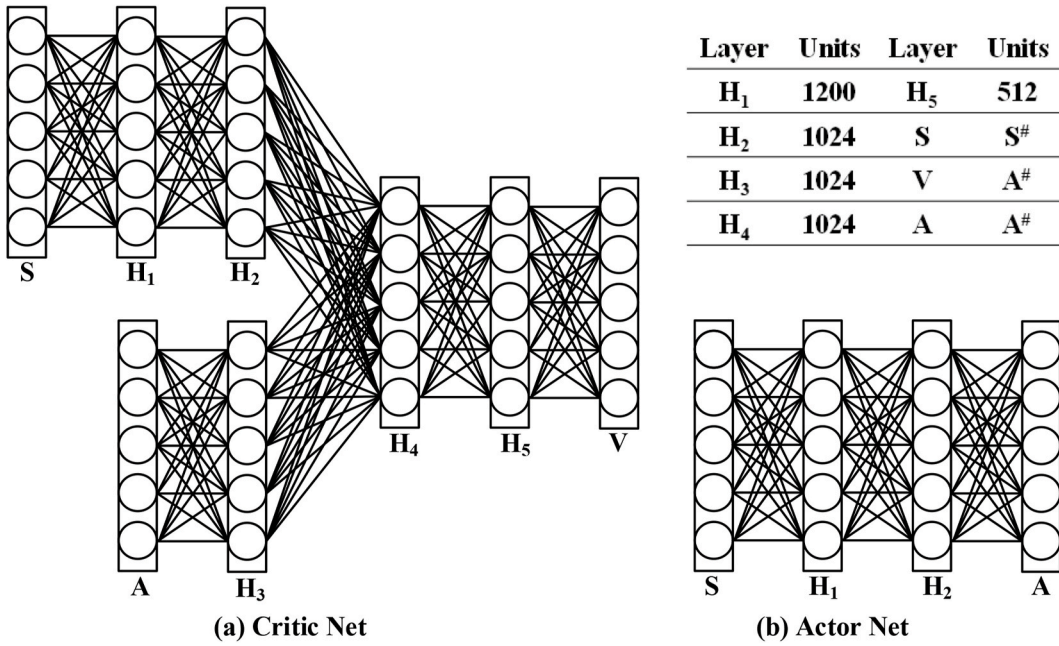


Fig. 4. The architecture of the critic net and the actor net.

traffic-splitting policies are deployed on two SDN nodes (S1, S2), respectively. According to the traffic-splitting policies, traffic flows on SDN switches are split into different forwarding links to the destination node. Formally, we denote the a_t as a vector $[P_{v,u}^e | v \in V_s, u \in V, e \in PL_u(v)]$, where $P_{v,u}^e \in [0, 1]$ is the traffic-splitting ratio on the link e from the SDN node v to the destination node u . Here, $PL_v(u)$ gives the *permissible links* of node v to the destination node u .

In each traffic-splitting policy, there is a constraint that the sum of splitting ratios on the forwarding links of a SDN node should equal to 1. However, the actor neural network, which outputs the action $a_{v,u}^e$ ranging from 0 to 1, cannot guarantee the constraint. Therefore, to meet the constraint, a necessary transformation of the output of actor neural network is made to obtain the traffic-splitting policies as follows:

$$P_{v,u}^e = \frac{\exp(\beta_1 \|a_{v,u}^e - \beta_2\|^2)}{\sum_{e' \in PL_u(v)} \exp(\beta_1 \|a_{v,u}^{e'} - \beta_2\|^2)} \quad (11)$$

where β_1 and β_2 are two parameters, which are readily set by cross-validations. In practice, because the actor neural network's output $a_{v,u}^e$ can never be 0, we clip $a_{v,u}^e$ to be 0 when its value is smaller than a threshold $\varepsilon = 10^{-5}$.

5.2.3. Reward

As shown in Eq. (1), the goal of TE in hybrid SDN is to minimize the MLU in hybrid SDN. Therefore, to enable the traffic-splitting agent to efficiently learn an optimal policy with a minimal MLU, the MLU is introduced into designing the reward r_t as follows:

$$r_t = \begin{cases} -e^{2\left(\frac{1}{\alpha}-1\right)} & \text{if } \alpha < 1 \\ 0 & \text{if } \alpha = 1 \\ e^{2(\alpha-1)} & \text{if } \alpha > 1 \end{cases} \quad (12)$$

$$\alpha = \frac{U_{i,0}^{max}}{U_{i,t}^{max}} \quad (13)$$

where $U_{i,0}^{max}$ and $U_{i,t}^{max}$ are the MLU of the hybrid SDN with the TM D_i at time 0 and t , respectively. Here, $U_{i,0}^{max}$ is considered as a baseline of minimizing MLU, and is computed in the hybrid SDN where there is no traffic-splitting policies deployed and all the traffic flows are forwarded according to the OSPF protocol. The ratio α reflects the degree of improvements on the network performance when the traffic-splitting policies are deployed at time t . Additionally, the reward r_t calculated by Eq. (12) encourages the agent with a higher positive reward if the traffic-splitting action taken at time t generates a lower MLU, and penalizes the agent with a lower negative reward if the traffic-splitting action taken at time t generates a higher MLU.

Algorithm 1 gives the main procedure of learning the traffic-splitting agent by DDPG. Specifically, the inputs of Algorithm 1 contain the hybrid SDN G , the set of historical TMs D , and the set of *permissible links* PL . In Algorithm 1, the actor-critic method adopted by DDPG has two types of neural networks, i.e. the critic net and the actor net. The actor net outputs the action a corresponding to state s . The critic net outputs the expected reward r which measures the effectiveness of action a taken at state s . Both the two nets consist of two subnets, i.e. the online net and the target net, whose architectures are the same. As shown in Fig. 4, each layer in both two nets is a sigmoid layer with many neurons. The critic net consists of two input layers S and A , five hidden layers H_1-H_5 , and an output layer V . The actor net consists of an input layer S , an output layer V and two hidden layers H_1 and H_2 . The number of neurons in S , A and V depends on the topology of a given hybrid SDN. The critic net and the actor net are iteratively trained with the transitions stored in replay buffer B . Finally, the Algorithm 1 outputs the trained online actor network $\mu(s|\theta^\mu)$, which is considered as the traffic-splitting agent in our proposed ROAR method. In the rest of this subsection, the details of the

Table 2
Three real network topologies.

Name	Nodes	Links
Abilene	12	30
Cernet	14	32
Geant	23	74

Algorithm 1 will be explained.

Algorithm 1 Learning the traffic-splitting agent by DDPG

Input: $G = (V, E), D, PL$
Output: $\mu(s|\theta^\mu)$

- 1: Initialize online actor network $\mu(s|\theta^\mu)$ and online critic network $Q(s, a|\theta^Q)$ with random θ^μ and θ^Q
- 2: Initialize target actor network $\mu'(s|\theta^{\mu'})$ and target critic network $Q'(s, a|\theta^{Q'})$ with $\theta^{\mu'} \leftarrow \theta^\mu, \theta^{Q'} \leftarrow \theta^Q$
- 3: Initialize replay buffer B
- 4: Initialize Ornstein-Uhlenbeck process $\mathcal{N}$
- 5: $\varepsilon = 1.0$
- 6: **for** D_i in D **do**
- 7: **for** episode $n = 1 \dots N$ **do**
- 8: $F_{i,0} = \text{get_ospf_flows}(G, D_i)$
- 9: $(s_0, U_{i,0}^{\max}) = \text{get_state}(F_{i,0})$
- 10: **for** step $t = 0 \dots T-1$ **do**
- 11: $a_t = \mu(s_t|\theta^\mu) + \varepsilon \mathcal{N}_{(m-1)T+t}, \varepsilon \leftarrow \eta \varepsilon$
- 12: $P = \text{get_policy}(a_t, PL)$
- 13: $F_{i,t+1} = \text{get_flows}(G, D_i, P)$
- 14: $(s_{t+1}, U_{i,t+1}^{\max}) = \text{get_state}(F_{i,t+1})$
- 15: $r_t = \text{get_reward}(U_{i,t+1}^{\max}, U_{i,t}^{\max})$
- 16: **if** $t = T$ **then**
- 17: $done_t = 1$
- 18: **else**
- 19: $done_t = 0$
- 20: **end if**
- 21: $B.\text{store}(s_t, a_t, r_t, s_{t+1}, done_t)$
- 22: minibatch $B' = B.\text{sample}(M)$
- 23: **for** transition $(s_j, a_j, r_j, s_{j+1}, done_j) \in B'$ **do**
- 24: $y_j = r_j + \gamma(1 - done_j)Q'(s_{j+1}, \mu'(s_{j+1}|\theta^{\mu'})| \theta^{Q'})$
- 25: **end for**
- 26: Update online critic and actor with Eq. (14) (15)
- 27: Update target critic and actor with Eq. (16) (17)
- 28: **end for**
- 29: **end for**
- 30: **return** $\mu(s|\theta^\mu)$

In Algorithm 1, we first execute the initialization steps (line 1–5). Specifically, the online actor network $\mu(s|\theta^\mu)$ and online critic network $Q(s, a|\theta^Q)$ are initialized with random θ^μ and θ^Q , respectively. The target actor network μ' and target critic network Q' , which assist in the stable learning, are initialized by copying the parameters of the corresponding online network. The replay buffer B , which holds the set of transition $(s_t, a_t, s_{t+1}, r_t, done)$ for agent learning, is initialized as a circular array with a fixed size. The Ornstein-Uhlenbeck (OU) noisy process (Uhlenbeck and Ornstein, 1930) $\mathcal{N}$ is initialized for action exploration and it models the velocity of a Brownian particle with friction. Here, $\mathcal{N}_t$ represents its value at time t . Additionally, to balance the action exploration and exploitation, the term ε , which ensures OU with proper noisy, is initialized to be 1.0.

After initialization, we begin to train the actor nets and the critic nets on the set of historical TM D . For each TM D_i in D , the training procedure is performed on M episodes and each episode has T steps. At the

beginning of each episode, the initial state s_0 is calculated by *get_ospf_flows* function and *get_state* function (line 8). Here, the *get_ospf_flows* function outputs the flow set $F_{i,0} = \{f_{e,D_i}^q(\omega)|q \in V, e \in E\}$ when there is no traffic-splitting policy is deployed on the hybrid SDN. The *get_state* function computes the state s_0 by Eq. (10) and the MLU of the hybrid SDN U^{max} can be obtained from s_0 . At each step t , the action a_t is first obtained by adding the output of online actor net at state s_t with a discounted OU noise. Then, the action a_t is executed on the state s_t and get the reward r_t (line 11–13); the traffic-splitting policies P are obtained by *get_policy* function which computes Eq. (11); the new state s_{t+1} is obtained after deploying traffic-splitting policies P ; the reward r_t is obtained by *get_reward* function which computes Eqs. (12) and (13). Afterwards, the transition $(s_t, a_t, r_t, s_{t+1}, done_t)$ is stored into the replay buffer B . Here, *done_t* is an indicator, which indicates whether the agent can receive more rewards in the remaining steps. Finally, the critic net and the actor net are updated on the minibatch B' , which is formed by randomly sampling M transitions from the replay buffer B (line 20–25). Specifically, for each transition in B' , the accumulated expected reward y_j is obtained by combining the reward r_j with the expected reward $\gamma Q'$, which is estimated by the target critic net at state s_{j+1} . Here, the term $\gamma \in (0, 1]$ is a discount factor. Based on the accumulated expected reward y_j , the online actor and critic net are updated. For the online critic net, the parameters θ^Q is updated by minimizing the loss between the accumulated expected reward y_j and the estimated reward $Q(s_j, a_j|\theta^Q)$ as follows:

$$L_Q(\theta^Q) = \frac{1}{M} \sum_{j=1}^M (y_j - Q(s_j, a_j|\theta^Q))^2 \quad (14)$$

For the online critic net, the parameters θ^μ are updated by sampled policy gradient as follows:

$$\nabla J_\mu(\theta^\mu) \approx \frac{1}{M} \sum_{j=1}^M \nabla_a Q(s_j, \mu(s_j|\theta^\mu)|\theta^Q) \nabla_{\theta^\mu} \Pi(s_j|\theta^\mu) \quad (15)$$

For the target actor and critic nets, the parameters $\theta^{\mu'}$ and $\theta^{Q'}$ are softly updated as follows:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \quad (16)$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \quad (17)$$

where the term $\tau \ll 1$ keeps the smooth update of the target nets.

Finally, once the offline learning procedure of DDPG terminates, the output of Algorithm 1 is the online actor $\mu(s|\theta^\mu)$, which is the trained traffic-splitting agent.

5.3. Online routing scheme generation

To obtain the routing schemes online, we deploy the trained traffic-splitting agent on the SDN controller. In practice, the SDN controller can generate the current TM of the hybrid SDN by collecting the network

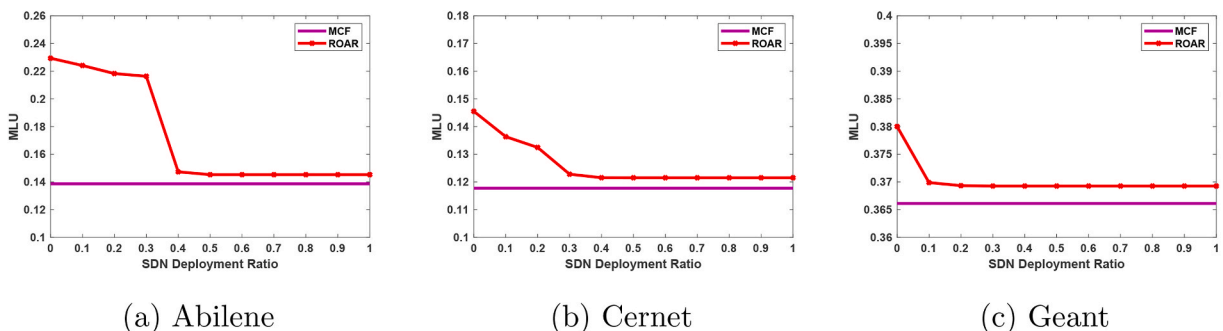


Fig. 5. The MLU of the proposed ROAR method varies with the SDN deployment ratio in different education topologies.

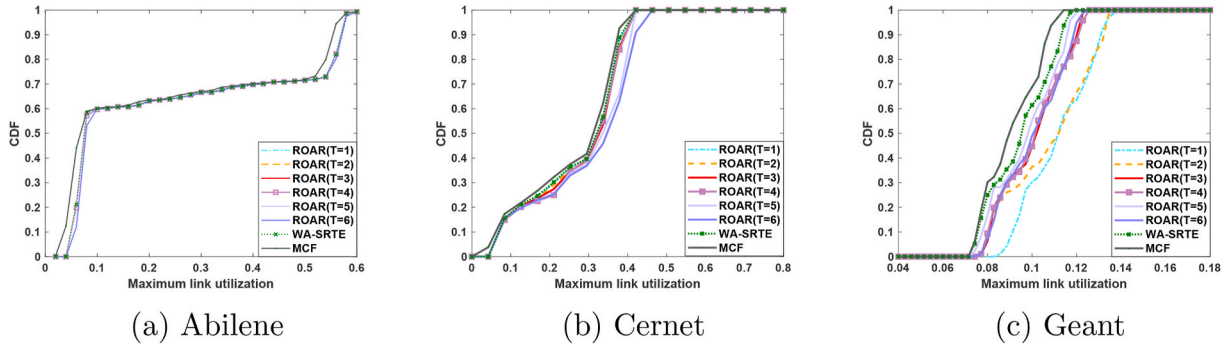


Fig. 6. CDF curves of MLU under different values of T in different topologies.

traffic reports from the SDN switches. These network traffic reports can be measured by the network monitoring tools, i.e. NetFlow (Benoit et al., 2004) and sFlow (Panchen and McKeePeter, 2001), which are installed on the SDN switches. According to the generated current TM, the trained traffic-splitting agent generates the corresponding traffic-splitting policies $\mathbf{P}$. Here, the Algorithm 2 presents the main procedure of generating the traffic-splitting policies $\mathbf{P}$. The inputs of the Algorithm 2 consist of the hybrid SDN G , the set of *permissible links* $\mathbf{PL}$, the learnt traffic-splitting agent $\mu(s|\theta^\mu)$, and the current TM D_i . As shown in Algorithm 2, with the traffic-splitting agent, we can obtain the traffic-splitting policies $\mathbf{P}$ of a new TM in T steps. Once the traffic-splitting policies $\mathbf{P}$ are obtained, the routing paths of traffic flows can be determined by (1) forwarding the flows on legacy nodes according to the OSPF protocol and (2) splitting the flows on the SDN nodes according to the traffic-splitting policies $\mathbf{P}$. Here, the OSPF protocol allows that the flows on legacy nodes are forwarding along the shortest routing paths, which are determined by the link weights. The link weights are optimized and determined by the link weight optimization described in Section 4.2.

In practice, the traffic-splitting policies can be implemented by the OpenFlow group table on the SDN switches (Openflow switch specifica, 2012). Specifically, the traffic-splitting policies provide the splitting ratios and outports of a flow. With the outports and splitting ratio, we can set the corresponding output actions and weights of OpenFlow group buckets by modifying or adding the flow entries in the OpenFlow group table. Here, different weights of OpenFlow group buckets indicate different splitting ratios of a flow. The output actions indicate the outports of a flow.

Algorithm 2 Traffic-splitting policies generation

Input: $G = (V, E), \mathbf{PL}, \mu(s|\theta^\mu), D_i$

Output: $\mathbf{P}, U_{T-1}^{\max}$

1: $F_{i,0} = \text{get_ospf_flows}(G, D_i)$

$(s_0, U_{i,0}^{\max}) = \text{get_state}(F_{i,0})$

(continued on next column)

(continued)

```

2: for step  $t = 0$  to  $T - 1$  do
3:    $a_t = \mu(s_t|\theta^\mu)$ 
4:    $\mathbf{P} = \text{get\_policy}(a_t, \mathbf{PL})$ 
5:    $F_{i,t+1} = \text{get\_flows}(G, D_i, \mathbf{P})$ 
6:    $(s_{t+1}, U_{i,t+1}^{\max}) = \text{get\_state}(F_{i,t+1})$ 
7:    $r_t = \text{get\_reward}(U_{i,t+1}^{\max}, U_{i,t}^{\max})$ 
8: end for
9: return  $\mathbf{P}$ 

```

6. Experimental evaluation

In order to demonstrate the performance of our proposed ROAR method on TE in the hybrid SDN, we conduct extensive experiments on different topologies and traffic datasets. In this section, the experimental setting is first presented. Then, through extensive experimental analysis, we determine the deployment ratio of SDN nodes used to construct the hybrid SDN and the number of steps used to generate the traffic-splitting policies. After that, we demonstrate the comparable performance of the proposed ROAR method on minimizing MLU in different datasets. Finally, we present the superiorities of the proposed ROAR method on reducing the time cost of calculating the routing schemes.

6.1. Experimental setting

6.1.1. Network topologies

Experimental evaluations are implemented on three network topologies: Abilene, Cernet, and Geant. Specifically, Abilene, Cernet, and Geant are the network topology of America Research and Education Network, China Education and Research Network, and Europe Research and Education Network, respectively. As shown in Table 2, the three networks have different numbers of links and nodes.

6.1.2. Traffic datasets

The TMs in Abilene, Cernet and Geant are real traffic datasets, which are provided by the TOTEM project (Balon and Monfort, 2019), Zhang (Zhang et al., 2014) and Uhling (Steve UhligBruno et al., 2006), respectively. Concretely, in Abilene and Cernet, the TMs are measured every 5 min, and 288 TMs are obtained in one day. Here, for Abilene and Cernet, we uniformly sample 1440 TMs and 288 TMs to form the training and testing datasets, respectively. Additionally, in Geant, the TMs are measured every 15 min, and 96 TMs are generated in one day. Here, for Geant, we uniformly sample 480 TMs and 96 TMs as training dataset and testing dataset, respectively.

6.1.3. DDPG setup

At the offline learning stage, we set the number of episodes N at 160. Additionally, the size of minibatch M and the size of replay buffer are set to 32 and 8000, respectively. The discount factor γ and the term τ are set to 0.9 and 0.001, respectively. We use Adam Optimizer (Zhang, 2018) to

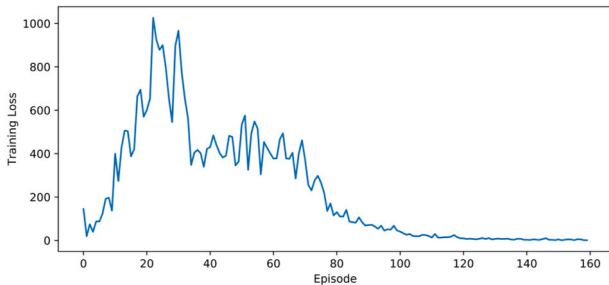
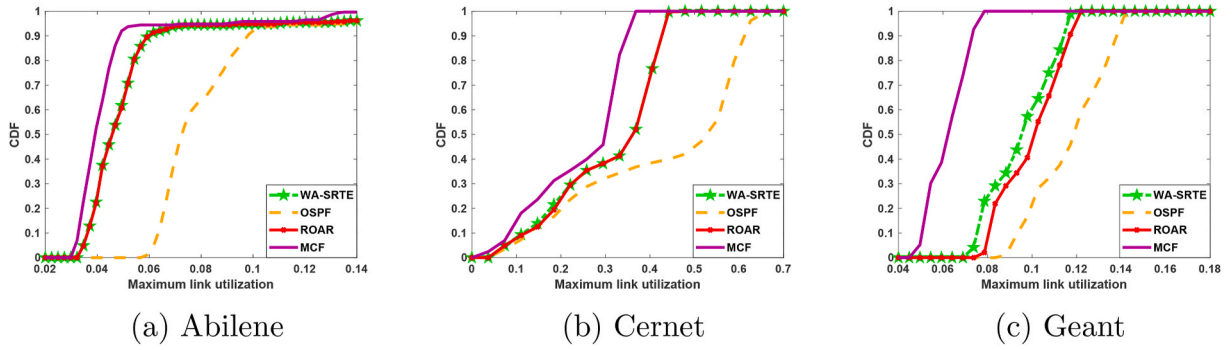
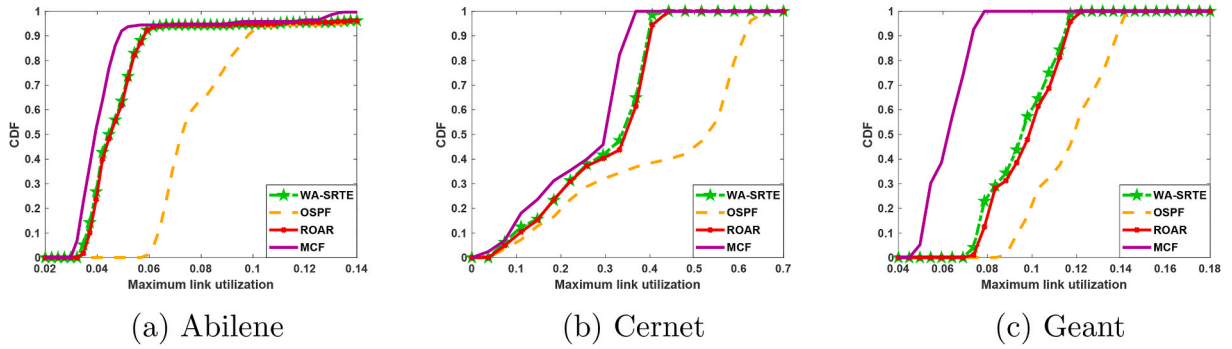


Fig. 7. The training loss of the online critic net varies with the learning episode (Geant, $s = 0.3$).

Fig. 8. The CDF curves of MLU under different topologies ($s = 0.2$).Fig. 9. The CDF curves of MLU under different topologies ($s = 0.3$).

optimize the loss function, and set the learning rates of the online actor and critic nets at 1×10^{-3} and 2×10^{-3} .

To exhibit the superiority of the proposed ROAR method, we conducted the comparative experiments using the following methods:

- **OSPF** (Fortz and Thorup, 2000): The OSPF method is a routing method often used in the traditional distributed network. Based on the OSPF protocol, the OSPF method allows that flows are routed on the shortest paths between source and destination. The shortest paths are calculated according to the setting of link weights.
- **WA-SRTE** (Tian et al., 2020): The WA-SRTE method is a routing optimization method used in a hybrid IP/SRv6 network. It uses DDPG algorithm to determine the link weight setting and deploys the SR nodes offline, and solves the LP problem to obtain the traffic-splitting ratios on SR nodes online. Although the hybrid IP/SRv6 network is different from the hybrid SDN network, for comparison, we migrated the WA-SRTE method to solve the TE problem in the hybrid SDN by changing the path-based routing to the link-based routing.
- **MCF** (Garg and Koenemann, 2007): The MCF method formulates the TE problem as a Multi-Commodity Flow (MCF) problem, which assumes that the flows can be split fractionally with arbitrary proportions on any nodes in the network. The results of the MCF method can be treated as the theoretical optimal solution to minimize MLU in the network.

6.2. Parameter analysis

6.2.1. Deployment ratio of SDN nodes

To construct a hybrid SDN from a traditional distributed network, we should first determine the SDN deployment ratio sdn_ratio , which determines the number and the placement of SDN nodes. As described in Section 4.1, we iteratively select the node with the largest value of *score* to be upgraded to the SDN node. Here, we use the TMs in the training datasets to compute each node's *score* value. The procedure of SDN

deployment terminates until the required number of nodes are upgraded.

To analyze the influence of the SDN deployment ratio on the performance of the proposed ROAR method, we plot the curves that the MLU varies with different deployment ratios for different topologies in Fig. 5. As shown in Fig. 5, the MLU of the ROAR method decreases as the SDN deployment ratio increases, which demonstrates that more SDN nodes deployed will assist the proposed ROAR method in minimizing the MLU of the hybrid SDN. The reason is that more SDN nodes can bring more flexibility to forward and split the traffic flows in the hybrid SDN. In addition, the MLU rapidly decreases when the deployment ratio ranges from 0 to 0.2, and the MLU slightly decreases when the deployment ratio ranges from 0.4 to 1. Therefore, to reduce the high investment on purchasing and deploying SDN devices, we recommend to set the deployment ratio sdn_ratio to be 0.3 for reaping the most of benefit.

6.2.2. The number of steps

The number of steps T is an important parameter in generating the traffic-splitting policies from the learnt RL agent. To analyze the impact of the number of steps T on the performance of the proposed ROAR method, the MLU of the proposed ROAR method was recorded at the following configuration: 1, 2, 3, 4, 5, and 6. Here, to better demonstrate the network performance on the testing samples, the Cumulative Distribution Function (CDF) curves of MLU on different topologies are plotted. From the CDF curves shown in Fig. 6, we can observe that as the scale of the network topology increases, the value of T increasingly affects the network performance. The reason is that the larger scale of the network topology means the larger action space for the traffic-splitting agent to search, and this results in more steps for the traffic-splitting agent to search the optimal traffic-splitting policies in a network with a larger topology.

In addition, as shown in Fig. 6(c), the network performance increases when T ranges from 1 to 5, but decreases when T equals to 6, which demonstrates that too larger T may influence the traffic-splitting agent to find an optimal policy. Moreover, the larger T will bring the more

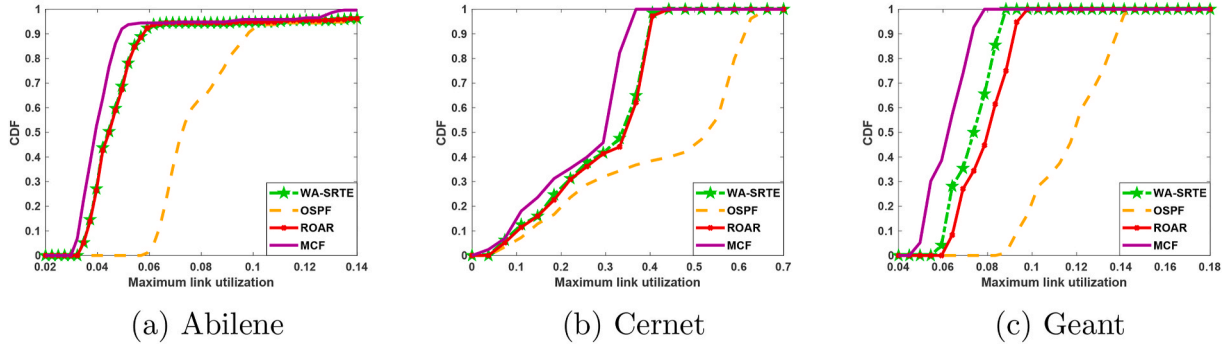


Fig. 10. The CDF curves of MLU under different topologies ($s = 0.4$).

computations for the traffic-splitting agent to search a policy. Therefore, we introduce the CDF curves of the WA-SRTE method and MCF method as references to determine the appropriate value of T in different topologies. Concretely, according to the CDF curves in Fig. 6, to make a tradeoff between the computing time and network performance, we set the values of T used in Abilene, Cernet and Geant to be 1, 2, and 3, respectively.

6.3. Network performance with ROAR

For each topology in the datasets, our proposed ROAR method first trains a traffic-splitting agent on the given training samples. To demonstrate the convergence of training procedure in the proposed ROAR method, we plots the curve of training loss of the online critic net in Geant. As shown in Fig. 7, although the loss curve fluctuates at the beginning of the training procedure, as the training episode increases, it gradually decreases and finally stabilizes at a low loss value. The loss curve demonstrates that the training procedure in the proposed ROAR method has converged. Once the training procedure converges, we can obtain a trained agent to generate the traffic-splitting policies online.

To demonstrate the performance of our proposed ROAR method on reducing the MLU in the hybrid SDN, we conduct comparisons with the following three works: OSPF (Fortz and Thorup, 2000), WA-SRTE (Tian et al., 2020), and MCF (Garg and Koenemann, 2007). Here, the link weights used in OSPF, WA-SRTE, and MCF are obtained by the method described in section 4.2. Moreover, considering that the SDN nodes are incrementally deployed in practice, we recorded the values of MLU at different deployment ratios, i.e. $s = 0.2, 0.3, 0.4$. As shown in Figs. 8–10, the CDF curves of ROAR stay above the CDF curves of OSPF. This demonstrates that compared with OSPF method, the ROAR method exhibits superior performance on achieving load link balancing. The

reason is that the traffic-splitting agent trained by the proposed ROAR method can effectively generate the traffic-splitting policies which allow traffic flow to be split to different *permissible links*. Moreover, the CDF curves of ROAR stay close to the CDF curves of WA-SRTE, and overlap with them in many cases. We can conclude that the performance of ROAR is comparable to WA-SRTE, which means that the relationship between the traffic demands and the effective traffic-splitting policies has been successfully established by the learnt traffic-splitting agent. Additionally, the CDF curves of ROAR locating below to the CDF curves of MCF indicates that there is still a gap between the proposed ROAR method and the theoretical optimal solution in optimizing network performance.

To further assess the performance of the ROAR method, we exhibit and analyze the quantitative evaluations on all built datasets when the deployment ration is set at 0.3. As shown in Table 3, compared to OSPF, the average MLU of ROAR is reduced by 60.78%, 47.40%, and 20.62% in Abilene, Cernet, and Geant, respectively. Meanwhile, in Abilene, Cernet, and Geant, the average MLU of the ROAR method deviating from the WA-SRTE method is only 1.96%, 2.08%, and 2.06%, respectively. Through the quantitative evaluations, we believe that the performance of ROAR method on the minimization of MLU is superior to the OSPF method, and approximates to the WA-SRTE method. This further proves that the ROAR method exhibits the ability in mapping the network state to the traffic-splitting policies which assists in achieving the link load balancing.

6.4. The processing time for online routing generation

All the simulation experiments are executed on a workstation with eight Intel cores of 2.1 GHz, a RAM of 125 GB and a Tesla P100 GPU. In addition, to conduct the fair comparisons, all the methods in the experiments are coded with Python and executed with a single thread. At the offline learning stage, the training time of the traffic-splitting agent for Abilene, Cernet and Geant are 10.6h, 14.2h, and 11.8h, respectively. At the online routing stage, Table 4 records the total processing time of the routing generation on the testing datasets, i.e. 288 TMs in Abilene, 288 TMs in Cernet, and 96 TMs in Geant. As shown in Table 4, the calculation time of the ROAR method for all TMs in the testing datasets on Abilene, Cernet and Geant is 2.76s, 2.49s, and 2.61s, respectively. And the average time of the ROAR method for each TM on Abilene, Cernet and Geant is only 9.5 ms, 8.6 ms, and 27 ms, respectively. This demonstrates that the proposed ROAR method can quickly adapt to the dynamically-changing traffic demands by rapidly generating the routing schemes, which exhibits the potentials of the proposed ROAR method in achieving online intelligent routing. Moreover, compared to the MCF and WA-SRTE, our proposed method requires less time in routing generation. This is because instead of solving a large-scale LP problem, the ROAR method directly maps the network state to the traffic-splitting policy by simply implementing the convolution operations. In addition, although the OSPF method needs less time, it cannot design the

Table 3

Average MLU under different topologies ($s = 0.3$).

	Abilene	Cernet	Geant
OSPF	0.082(60.78%)	0.426(47.40%)	0.117(20.62%)
WA-SRTE	0.050(-1.96%)	0.283(-2.08%)	0.095(-2.06%)
MCF	0.044(-13.73%)	0.244(-22.49%)	0.062(-36.08%)
ROAR	0.051	0.289	0.097

Table 4

The processing time of routing generation on testing datasets.

Name	Abilene	Cernet	Geant
OSPF	0.92s	1.38s	1.34s
WA-SRTE	60.09s	80.05s	81.28s
MCF	554.88s	780.74s	2067.42s
ROAR	2.76s	2.49s	2.61s

effective routing for the dynamically-changing traffic demands to improve the network performance.

7. Conclusion

In this paper, we have presented the ROAR method to implement TE in the hybrid SDN. In the proposed ROAR method, a traffic-splitting agent is trained on a set of traffic demands offline, and learns the relationship between the traffic demands and the traffic-splitting policies by the RL algorithm. Once a traffic-splitting agent is learnt, it can rapidly and intelligently generate the routing schemes when traffic demands are changing. Extensive evaluations on different topologies and real traffic datasets have demonstrated that the proposed ROAR method can achieve near-optimal network performance in the hybrid SDN. Moreover, the calculation time of routing generation for a TM have exhibited the potentials of the proposed ROAR method in achieving online intelligent routing. We believe that the proposed method provides a promising attempt that exploits the RL algorithm to directly control the forwarding behaviors of SDN switches.

Although the simulation evaluations have demonstrated the superiorities of our proposed method in implementing TE in a hybrid SDN, in the future work, we will further evaluate our proposed method in the real production networks. Actually, our proposed ROAR method can be used as a module to map the network states to the traffic-splitting policies. Because the real ISP network may adopt different strategies to deploy the SDN devices, more studies will be implemented on evaluating the generalization of our proposed method to the different strategies of the SDN deployment and the different topologies of hybrid SDN. The source codes and datasets used in this paper are available at <https://github.com/vpengwang/ROAR>.

Author statement

Guo Yingya: Investigation, Methodology, Writing. Weipeng Wang: Writing, Experiments. Han Zhang: Conceptualization, Writing. Zhiliang Wang: Reviewing, Editing. Ying Tian: Investigation, Experiments. Wenzhong Guo: Supervision, Reviewing. Xia Yin: Supervision, Resources. Wu Jianping: Supervision, Resources.

Declaration of competing interest

No conflict of interest exists in the submission of this manuscript, and manuscript is approved by all authors for publication. The work described is original research that has not been published previously, and not under consideration for publication elsewhere.

Acknowledgements

This work is partially supported by National Natural Science Foundation of China under Grant No.62002064 and No.61801121, and the Natural Science Foundation of Fujian Province under Grant 2018J07005 and 2020J05110.

References

- Agarwal, Sugam, Kodialam, Murali, Lakshman, T.V., 2013. Traffic engineering in software defined networks. In: International Conference on Computer Communications (INFOCOM). IEEE, pp. 2211–2219.
- Ahmed, Mohiuddin, Seraj, Raihan, Shamsul Islam, Syed Mohammed, 2020. The k-means algorithm: a comprehensive survey and performance evaluation. *Electronics* 9 (8), 1295.
- Altun, Aysegül, Fortz, Bernard, Thorup, Mikkel, Ümit, Hakan, 2013. Intra-domain traffic engineering with shortest path routing protocols. *Ann. Oper. Res.* 204 (1), 65–95.
- Amin, Rashid, Reisslein, Martin, Shah, Nadir, 2018. Hybrid sdn networks: a survey of existing approaches. *IEEE Communications Surveys & Tutorials* 20 (4), 3259–3306.
- Balon, S., Monfort, G., 2019. The Traffic Matrices and Topology of the Abilene Network. Benoit, Claise, Sadasivan, Ganesh, Valluri, Vamsi, Martin, Djernaes, 2004. Cisco Systems Netflow Services Export Version 9.
- Caria, M., Das, T., Jukan, A., Hoffmann, M., 2015. Divide and conquer: partitioning ospf networks with sdn. In: International Symposium on Integrated Network Management. IFIP/IEEE, pp. 467–474.
- Cianfrani, Antonio, Listanti, Marco, Polverini, Marco, 2017. Incremental deployment of segment routing into an isp network: a traffic engineering perspective. *IEEE/ACM Trans. Netw.* 25 (5), 3146–3160.
- Cisco, U., 2020. Cisco Annual Internet Report (2018–2023) White Paper.
- Ericsson, M., Mauricio, G., Resende, C., Pardalos, Panos M., 2002. A genetic algorithm for the weight setting problem in ospf routing. *J. Combin. Optim.* 6 (3), 299–333.
- Fortz, Bernard, Thorup, Mikkel, 2000. Internet traffic engineering by optimizing ospf weights. In: International Conference on Computer Communications (INFOCOM), vol. 2. IEEE, pp. 519–528.
- Fortz, Bernard, Thorup, Mikkel, 2002. Optimizing ospf/is-is weights in a changing world. *IEEE J. Sel. Area. Commun.* 20 (4), 756–767.
- Fortz, Bernard, Thorup, Mikkel, 2004. Increasing internet capacity using local search. *Comput. Optim. Appl.* 29 (1), 13–48.
- Garg, Naveen, Koenemann, Jochen, 2007. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.* 37 (2), 630–652.
- Guo, Yingya, Wang, Zhiliang, Yin, Xia, Shi, Xingang, Wu, Jianping, 2014. Traffic engineering in sdn/ospf hybrid network. In: International Conference on Network Protocols (ICNP). IEEE, pp. 563–568.
- Guo, Yingya, Wang, Zhiliang, Yin, Xia, Shi, Xingang, Wu, Jianping, 2017a. Traffic engineering in hybrid sdn networks with multiple traffic matrices. *Comput. Network.* 126, 187–199.
- Guo, Yingya, Wang, Zhiliang, Yin, Xia, Shi, Xingang, Wu, Jianping, 2017b. Traffic engineering in hybrid sdn networks with multiple traffic matrices. *Comput. Network.* 126, 187–199.
- Guo, Yingya, Wang, Zhiliang, Liu, Zhifeng, Yin, Xia, Shi, Xingang, Wu, Jianping, Xu, Yang, Chao, H Jonathan, 2019. Sote: traffic engineering in hybrid software defined networks. *Comput. Network.* 154, 60–72.
- Guo, Yingya, Luo, Huan, Wang, Zhiliang, Yin, Xia, Wu, Jianping, 2021. Routing optimization with path cardinality constraints in a hybrid sdn. *Comput. Commun.* 165, 112–121.
- He, J., Song, W., 2015. Achieving near-optimal traffic engineering in hybrid software defined networks. In: NETWORKING Conference. IFIP, pp. 1–9.
- Hong, Chi-Yao, Kandula, Srikanth, Mahajan, Ratul, Zhang, Ming, Gill, Vijay, Nanduri, Mohan, Wattenhofer, Roger, 2013. Achieving high utilization with software-driven wan. In: Proceedings of ACM SIGCOMM. ACM, pp. 15–26.
- Hong, David Ke, Ma, Yadi, Banerjee, Sujata, Mao, Z Morley, 2016. Incremental deployment of sdn in hybrid enterprise and isp networks. In: Proceedings of the Symposium on SDN Research, pp. 1–7.
- Huang, Xiaohong, Zeng, Man, Xie, Kun, 2021. Intelligent Traffic Control for Qos Optimization in Hybrid Sdns. *Computer Networks*, p. 107877.
- Jain, Sushant, Kumar, Alok, Mandal, Subhasree, Ong, Joon, Leon, Poutievski, Singh, Arjun, Venkata, Subbaiah, Jim Wanderer, Zhou, Junlan, Zhu, Min, et al., 2013. B4: experience with a globally-deployed software defined wan. In: Proceedings of the ACM SIGCOMM. ACM, pp. 3–14.
- Lillicrap, Timothy P., Hunt, Jonathan J., Alexander, Pritzel, Heess, Nicolas, Tom, Erez, Tassa, Yuval, Silver, David, Wierstra, Daan, 2015. Continuous control with deep reinforcement learning. *Computer Science* 8 (6), A187.
- Liu, Xuan, Mohanraj, Sudhir, Pioro, Michal, Deep, Medhi, 2014. Multipath routing from a traffic engineering perspective: how beneficial is it? In: 2014 IEEE 22nd International Conference on Network Protocols. IEEE, pp. 143–154.
- McKeown, Nick, Anderson, Tom, Balakrishnan, Hari, Guru, Parulkar, Peterson, Larry, Rexford, Jennifer, Scott, Shenker, Turner, Jonathan, 2008. Openflow: enabling innovation in campus networks. *Comput. Commun. Rev.* 38 (2), 69–74.
- Mendiola, Alaitz, Astorga, Jasone, Jacob, Eduardo, Higuero, Marivi, 2016. A survey on the contributions of software-defined networking to traffic engineering. *IEEE Communications Surveys & Tutorials* 19 (2), 918–953.
- Moy, John, et al., 1998. Ospf version 2.
- Open Networking Foundation, 2012. Software-defined networking: the new norm for networks. ONF White Paper 2 (2–6), 11.
- Openflow Switch Specification v1.3.1, 2012. Open Networking Foundation.
- Oran, David, 1990. Rfc1142: Osi Is-Is Intra-domain Routing Protocol.
- Panchen, Sonia, McKee, Neil, Peter, Phaal, sep 2001. sflow: a method for monitoring traffic in switched and routed networks. Technical report, RFC 3176 (Cité efinpage 46.).
- Ravi Kiran, B., Sobh, Ibrahim, Talpaert, Victor, Mannion, Patrick, Sallab, Ahmad A Al, Yogamani, Senthil, Pérez, Patrick, 2021. Deep reinforcement learning for autonomous driving: a survey. *IEEE Transactions on Intelligent Transportation Systems*.
- Ren, Cheng, Bai, Shiwei, Wang, Yu, Li, Yaxin, 2020. Achieving near-optimal traffic engineering using a distributed algorithm in hybrid sdn. *IEEE Access* 8, 29111–29124.
- Sharma, Akanksha Rai, Kaushik, Pranav, 2017. Literature survey of statistical, deep and reinforcement learning in natural language processing. In: International Conference on Computing, Communication and Automation (ICCCA). IEEE, pp. 350–354.
- Sinha, Yash, Haribabu, K., et al., 2017. A survey: hybrid sdn. *J. Netw. Comput. Appl.* 100, 35–55.
- Sridharan, Ashwin, Guerin, Roch, Diot, Christophe, 2005. Achieving near-optimal traffic engineering solutions for current ospf/is-is networks. *IEEE/ACM Trans. Netw.* 13 (2), 234–247.
- Steve Uhlig, Bruno, Quoitin, Jean, Lepropre, Balon, Simon, 2006. Providing public intradomain traffic matrices to the research community. In: Proceedings of ACM SIGCOMM. ACM, pp. 83–86.

- Sun, Penghao, Lan, Julong, Li, Junfei, Zhang, Jianpeng, Hu, Yuxiang, Guo, Zehua, 2020. A scalable deep reinforcement learning approach for traffic engineering based on link control. *IEEE Commun. Lett.* 25 (1), 171–175.
- Sun, Penghao, Li, Junfei, Guo, Zehua, Xu, Yang, Lan, Julong, Hu, Yuxiang, 2019. Sinet: enabling scalable network routing with deep reinforcement learning on partial nodes. In: *Proceedings of ACM SIGCOMM*. ACM, pp. 88–89.
- Tian, Y., Wang, Z., Yin, X., Shi, X., Guo, Y., Geng, H., Yang, J., 2020. Traffic engineering in partially deployed segment routing over ipv6 network with deep reinforcement learning. *IEEE/ACM Trans. Netw.* 28 (4), 1573–1586.
- Uhlenbeck, George E., Ornstein, Leonard S., 1930. On the theory of the brownian motion. *Phys. Rev.* 36 (5), 823.
- Valadarsky, Asaf, Schapira, Michael, Shahaf, Dafna, Aviv Tamar, 2017. Learning to route. In: *Proceedings of the ACM HotNets*. ACM, pp. 185–191.
- Vissicchio, Stefano, Vanbever, Laurent, Olivier, Bonaventure, 2014. Opportunities and research challenges of hybrid software defined networks. In: *Proceedings of ACM SIGCOMM*. ACM, pp. 70–75.
- Wang, Yufei, Wang, Zheng, Zhang, Leah, 2001. Internet traffic engineering without full mesh overlaying. In: *International Conference on Computer Communications (INFOCOM)*. IEEE, pp. 565–571.
- Wang, Wenguan, Lai, Qiuxia, Fu, Huazhu, Shen, Jianbing, Ling, Haibin, Yang, Ruigang, 2021. Salient Object Detection in the Deep Learning Era: an In-Depth Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Xiang, Guofei, Su, Jianbo, 2021. Task-oriented deep reinforcement learning for robotic skill acquisition and control. *IEEE Transactions on Cybernetics* 51 (2), 1056–1069.
- Xu, Hongli, Li, Xiang-Yang, Huang, Liusheng, Deng, Hou, Huang, He, Wang, Haibo, 2017a. Incremental deployment and throughput maximization routing for a hybrid sdn. *IEEE/ACM Trans. Netw.* 25 (3), 1861–1875.
- Xu, Hongli, Fan, Jinyuan, Wu, Jianhua, Qiao, Chunming, Huang, Liusheng, 2017b. Joint deployment and routing in hybrid sdns. In: *International Symposium on Quality of Service (IWQoS)*. IEEE, pp. 1–10.
- Xu, Zhiyuan, Tang, Jian, Meng, Jingsong, Zhang, Weiye, Wang, Yanzhi, Liu, Chi Harold, Yang, Dejun, 2018. Experience-driven networking: a deep reinforcement learning based approach. In: *Proceedings of the IEEE INFOCOM*. IEEE, pp. 1871–1879.
- Zhang, Zijun, 2018. Improved adam optimizer for deep neural networks. In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, pp. 1–2.
- Zhang, Baobao, Bi, Jun, Wu, Jianping, Baker, Fred, 2014. Cte: cost-effective intra-domain traffic engineering. In: *Proceedings of ACM SIGCOMM*. ACM, pp. 115–116.
- Zhang, Yi, Sun, Ping, Yin, Yuhua, Lin, Lin, Wang, Xuesong, 2018. Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning. In: *Intelligent Vehicles Symposium (IV)*. IEEE, pp. 1251–1256.
- Zhang, Wei Emma, Sheng, Quan Z., Alhazmi, Ahoud, Li, Chenliang, 2020a. Adversarial attacks on deep-learning models in natural language processing: a survey. *ACM Transactions on Intelligent Systems and Technology (TIST)* 11 (3), 1–41.
- Zhang, Junjie, Ye, Minghao, Guo, Zehua, Yen, Chen-Yu, Chao, H Jonathan, 2020b. Cfr-rl: traffic engineering with reinforcement learning in sdn. *IEEE J. Sel. Area. Commun.* 38 (10), 2249–2259.
- Zhu, Yuke, Mottaghi, Roozbeh, Kolve, Eric, Lim, Joseph J., Gupta, Abhinav, Fei-Fei, Li, Ali, Farhadi, 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3357–3364.

Yingya Guo received the B.S degree in computer science from Xiamen University, China, in 2013 and Ph.D. degree from Tsinghua University. She is currently an assistant professor in the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China.

Her research interests include traffic engineering, routing optimization and Software Defined Networking.

WeiPeng Wang received the B.S degree in the College of Mathematics and Computer Science, Fuzhou University in 2019 and now is a master candidate in the College of Mathematics and Computer Science, Fuzhou University. His research interests include Artificial intelligence, reinforcement learning and computer vision.

Han Zhang received the B.S degree in computer science from Jilin University, China, in 2013 and Ph.D. degree from Tsinghua University. He is currently an assistant professor in Tsinghua University, Beijing, China. His research interests include congestion control, datacenter networks and scheduling.

Wenzhong Guo received the Ph.D. degrees in the Department of Physics and Information Engineering, Fuzhou University, Fuzhou, China, in 2010. He is a professor and director of Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing. He was a postdoctoral research scholar in the Department of Computer Science, National University of Defense and Technology, Changsha, China, from 2011 to 2014, a visiting professor at the Faculty of Engineering, Information and System, University of Tsukuba, Japan, in 2013, and a visiting professor at the Department of Computer Science and Engineering, State University of New York at Buffalo, USA, in 2016. His research interests include the fields of data mining, machine learning, and artificial intelligence.

Zhiliang Wang received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University, China in 2001, 2003 and 2006 respectively. Currently he is an Associate Professor in the Institute for Network Sciences and Cyberspace at Tsinghua University. His research interests include formal methods and protocol testing, next generation Internet, network measurement.

Ying Tian received the B.E. degree in computer science from Beijing University of Posts and Telecommunications, China in 2018. She is currently a PhD candidate at the Department of Computer Science and Technology, Tsinghua University. Her research interests include traffic engineering, routing optimization, Segment Routing and Software Defined Networking.

Xia Yin received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University in 1995, 1997 and 2000 respectively. She is a Full professor in Department of Computer Science and Technology at Tsinghua University. Her research interests include future Internet architecture, formal method, protocol testing and large-scale Internet routing.

Jianping Wu received his B.S., M.S., and Ph.D. from Tsinghua University. He is a Full professor and director of Network Research Center, Ph.D. Supervisor of Department of Computer Science, Tsinghua University. From 1994, he has been in charge of China Education and Research Network (CERNET). He is a member of Information Advisory Committee, Office of National Information Infrastructure, Secretariat of State Council of China, and is also a vice president of Internet Society of China (ISC). His research interests include next generation Internet, IPv6 deployment and technologies, Internet protocol design and engineering.