Received October 21, 2020, accepted November 26, 2020, date of publication December 9, 2020,
date of current version December 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3043511

A Deep Reinforcement Learning Based Switch Controller Mapping Strategy in Software Defined Network

JIA CHEN[1], **(Member, IEEE), SHIHUA CHEN**[2], **XIN CHENG**[1], **AND JING CHEN**[1], **(Graduate Student Member, IEEE)**
[1]National Laboratory of Next Generation Internet Interconnection Devices, Department of Electronic and Information Engineering, Beijing 100044, China
[2]Tencent Technology Ltd., Shenzhen, China

Corresponding author: Jia Chen (chenjia@bjtu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61471029, and in part by the National Science and Technology Major Project under Grant 2013ZX03006002.

ABSTRACT Software defined network (SDN) is a promising technology which can reduce network management complexity through the decoupling of the control plane and data plane. Due to large number of switches in the data plane, distributed and multiple controllers are necessary in the control plane for managing the switches. The switch controller mapping strategy for identifying the mapping relationships between the switch and controller is crucial in order to optimize the network performance. Considering the dynamics of the network behavior, it is quite important and challenging to develop models to reflect the network topology dynamics and to propose method for solving the long-term network performance optimization. Inspired by the recent advances in Artificial Intelligence (AI), in this paper, we propose a Deep Reinforcement Learning (DRL) based strategy for solving the switch controller mapping problem. A DRL based mapping strategy is proposed, in which Markov Decision Process (MDP) formulation is devised and Deep *Q*-network (DQN) is proposed to achieve the maximization of long-term system performance by leveraging network latency, load balancing and system stability. Extensive simulations show that the DQN based algorithm can achieve the best system stability results while maintaining moderate switch controller latency and system equilibrium performance comparing with the optimization which only considers current system performance for switch controller mapping decision, and the optimization approaches which generate mapping decisions purely based on latency or load balancing separately.

INDEX TERMS Software defined network, deep reinforcement learning, controller, switch, mapping, deep *Q*-network.

I. INTRODUCTION

Software Defined Network (SDN) is an emerging technology, in which the control plane and forwarding plane are decoupled. In the control plane, the controllers own the overall view of the network and can hence perform centralized control of the network. In the forwarding plane, packet forwarding devices (usually Openflow switches), deal with the data packets' behavior based on the forwarding tables obtained from the controller [1]. The controller can manage the switches' behavior though distributing the forwarding rules into the switches. SDN can significantly reduce the complexity for the control and management of the network and promote network innovations, thus becoming popular in modern network scenarios, such like data centers.

Due to the application of large scale of switches in the networks, single controller mode is impossible. The large amount of traffic and the latency between controller and switch poses challenge for single controller design [2]. Accordingly, multiple and distributed controllers have been proposed in the literature in order to improve the reliability and scalability of the control plane [3]. In addition, the implementation of the physically distributed design of the logically centralized control plane has also been investigated [4], [5]. For the distributed control plane, the basic problem is to solve the controller placement problem of determining the

The associate editor coordinating the review of this manuscript and approving it for publication was Longxiang Gao.

controllers' location. This problem has been investigated in the literature thoroughly, in which the focus is to decide the number of controllers required and the placement locations of the controllers [3], [6]–[8].

Moreover, after the controllers have been placed, the problem of switch controller mapping needs to be solved. The switch controller mapping identifies the mapping between switch and controller based on the network environment. Most existing works in the area of switch controller mapping solve the mapping problem considering different objectives, such as balancing the load [9], maximizing the controller resource utilization [10], minimizing the response time [11], or considering both perspectives from the controller and switch [12]. However, the above existing research works solve the switch controller mapping problem based on the instantaneous network environment information, such as the controller traffic load, working status and resources' utilization.

However, the networks' topology may be dynamic and change with time. For example, the controller may leave the system due to controller failure. Thereby, the time-varying network dynamics may affect the mapping solution. For example, when a switch is attached to a controller which may have probability of failure in future, there will be additional cost for migrating the switch. Considering this, it is better to map this newly joint switch to a controller with less failure probability. Therefore, the time-varying network dynamics can affect the mapping decision. This poses challenge to the switch controller mapping problem, since the optimization decision for the switch controller mapping should be devised not only based on the current system status but also the time-varying characteristics of the network environment. A few research works have been designed to adapt to the network traffic dynamic situation [13]–[15]. However, the above existing research work deals with the long-term time-average performance optimization for minimizing the system's operation cost by only considering traffic fluctuation. No dynamics of the controller and switch, such as the joining and leaving of the controllers and switches due to network topology dynamics have been taken into consideration.

In this paper, the objective is to tackle controller switch dynamic mapping problem for optimizing the long-term system performance (cumulative system performance in the long run) by considering the network topology dynamics including both controllers' and switches' dynamic behavior. The system performance in terms of switches' response time, controllers' load balancing and system stability in terms of controller's failure impact are considered. Since the optimization problem is a multiple-stage decision process, we apply Markov Decision Process (MDP) for modeling the problem. We decompose the multi-period decision process into sub-process which is modeled by the network environment (state), mapping decision (action) and system performance (reward) in MDP. Then we apply deep reinforcement learning (DRL) approach for solving the problem. In reinforcement learning (RL), an agent can interact with the environment and adjustify the action policy in order to achieve the optimal solution for maximizing the long-term reward (cumulative reward in the long run) of the system [16]. Moreover, DRL has recently been proposed by DeepMind to overcome the limitations of RL for solving problems with large scale of system states [17]. Through taking advantages of deep learning during the learning process, DRL can further solve the problem with high dimensional input. Inspired by the powerful tool of DRL, we aim to apply it to solve the long-term optimization problem of dynamic switch controller mapping. The system performance of the objective is modeled as the reward in MDP and the agent can devise the optimal mapping decision as the action policy for solving it. Particularly, we adopt a popular DRL algorithm-Deep *Q*-network (DQN) in the switch controller mapping design. The contributions of the paper are summarized below.

- The dynamic switch controller mapping problem is proposed to optimize the long-term (cumulative performance in the long run) overall system performance in terms of switches' response time, controllers' load balancing and system stability considering the dynamics of network topology including both controllers' and switches' dynamics.
- The mapping problem is modeled as a MDP, in which the state, action and reward in MDP are formulated. Then a DQN based switch controller mapping algorithm is proposed for solving the long-term optimization problem.
- A simulation platform is established to validate the proposed strategy with different input parameters. The proposed DQN algorithm is compared with the mapping method without taking the long-term system performance into account. In addition, the mapping approaches which consider switches' response time and controllers' load balancing respectively are also compared.

The rest of the paper is organized as follows. Section II introduces the related work. The system model and design objective are presented in Section III. Section IV presents the DQN based switch controller mapping approach. Numerical results are shown in Section V. And section VI concludes the paper.

II. RELATED WORK

In this section, related work is introduced. We first present the related work on controller placement and switch controller mapping problem. Then the application of DQN and Q-learning(QL) in network resource management is also elaborated.

A. CONTROLLER PLACEMENT

The problem of controller placement is to determine the controllers' location. In multi-controller SDN system, the controller system design and the controllers' placement have been investigated in many research works [3], [6]–[8]. In [3], the controller placement design problem has been investigated. The impact of network topology and the placement of controllers on latency were analyzed. And the paper concludes that the reaction bounds, metric choices and network topology are the three factors which affect the placement of controllers. In [6], the distributed design of controller system has been investigated, which makes use of the local algorithms in distributed computing for enhancing SDN controller coordination. In [7], the controller placement algorithm is proposed for solving the number and the location of controllers' placement while satisfying the performance metrics, such as latency and convergence time. Particularly, the problem is modeled as standard graph theory problems in forms of K-median and K-center, to which the complexity of the optimal solution is NP-hard. And a heuristic K-critical algorithm has been proposed which outperforms the traditional heuristic K-median and K-center solution. In [8], heuristic approaches have been proposed to solve the controller placement problem in large scale SDN networks. A Pareto-based optimization framework is proposed for achieving different performance metrics.

For further improving the reliability design of SDN, the controller placement considering reliability design are further investigated in [18], [19]. In [18], the problem of determining the controller placement policy is solved for satisfying reliability requirement given links and switches' failure probability. The problem is modeled as a NP-hard problem and heuristic solutions are proposed to solve the problem. Similarly, in [19], a fault tolerant controller placement problem was formulated, in which the objective is to identify the placement of controllers for minimizing the cost while satisfying reliability requirement. Heuristic solution has been proposed to solve the problem. However, the above research work focusing on the placement of the controller has not considered the dynamic mapping between controllers and switches.

B. SWITCH CONTROLLER MAPPING

The switch controller mapping problem is to dynamically assign the switch controller mapping relationship based on network environment after the controllers have been placed. For solving the switch controller dynamic mapping problem, several research works have been proposed [9]–[15], [20]. In [9], an elastic distributed controller architecture is proposed, in which the controller can be dynamically justified based on the traffic. The proposed solution monitors the network performance and dynamically migrates the switch once the load is imbalanced. In [10], [20], the switch controller mapping problem is proposed for maximizing the sum of network utility of all controllers. And a distributed algorithm has been proposed for solving the optimization problem. In [11], the switch controller mapping was modeled for minimizing the average response time. A genetic algorithm has been proposed to solve the optimization problem heuristically. In [12], the switch controller mapping is proposed by considering the perspective from both the controller and switch. The status of the controller and switch is considered at the same time. A matching list is constructed accordingly, from which the mapping decision is made through mutual selection. However, the above research works only consider the switch controller mapping problem based on the instantaneous network environment information and instantaneous system performance.

More recently, a few research works have been proposed in which long-term performance are considered. In [13], the SDN controller assignment problem is solved for minimizing the long-term operating, maintenance and switching cost considering the dynamic changing of work load. An algorithm based on randomized fixed horizon control theory has been proposed, in which the long-term optimization is divided into a set of short-term optimization problem. In [14], a distributed switch controller dynamic orchestration problem including controller activation/deactivation and switch controller mapping is solved for minimizing long-term system cost. Stochastic optimization theory is applied to solve the multiple timescale optimization problem for adapting dynamic traffic variation. In [15], a dynamic control plane is proposed to minimize the long-term flow-setup and control plane adaptation cost for adapting dynamic traffic requirement and distribution. Multi-period optimization problem is established, which is decomposed into smaller instance solved by simulated annealing. However, the above works only tackled the problem for minimizing the long-term system cost considering the traffic fluctuation. Network dynamics, such as the joining and leaving of the controllers and switches due to network failure or topology dynamic have not been considered.

C. APPLICATION OF DQN/QL IN NETWORK RESOURCE MANAGEMENT

In the process of RL, the learner (agent) interacts with the environment and learns the optimal policy for achieving the maximum long-term cumulative performance. Throughout the process, the agent intends to make decisions to maximize the expected benefit. Particularly, at each time instant, the agent gathers the state information from the environment, produces an action, and obtains a reward. Then the system state is transited to the next state, which together with the reward are used for adjustifying the action. The above process is repeated for achieving the optimal policy which can maximize the long-term reward. The process of RL can be modeled as an MDP, which is a mathematical framework describing the decision-making process of the agent. MDP contains four parts: system status obtained from the environment, action set, the transition probability from current system status to the next, and the reward which is immediately obtained when an action is performed.

The most widely used RL algorithm is the Q-learning algorithm proposed in [21]. In Q-learning algorithm, a two-dimensional (state and action) Q-table is used to record the expected reward. The value of the Q-table is updated from each iteration in the learning process. Based on the Q-table, a certain system status can have different expected reward

TABLE 1. A summary of applying DQN/QL in network resource management.

REF	Scenario	State	Action	Reward	Agent	Algorithm
[23]	Resource management at the network edge	Locations of the VM and the number of users in each base station	Base station migration	Communication cost of data transmission and VM migration	Abstract Controller	DQN
[24]	Offloading in mobile edge	Task queue, remaining energy (MU-BS association state, channel)	Offloading decision and allocated energy	task completing performance and economic cost of service	Central controller	Double DQN
[25]	Offloading in mobile ad-hoc clouds	User queue and cloudlet's queue and distance between users and cloudlets	No. of task offloaded to each cloudlet	Utility−cost	End user	DQN
[26]	Controller synchronization in distributed SDN	Time from last synchronization for each domain	Domain selected for synchronization	Reduction in APC	Domain Controller	DQN
[27]	Service placement in SDN	Decision of deleting a service in a switch	Install a new service when old service is removed	The benefit of installing new service minus old service	Controller	QL
[28]	VNF Orchestration	Collection of physical nodes	Next hop node	Node and link resource utilization Node usage frequency	NFV controller	QL

and the action which can maximize the expected reward is selected and considered as the optimal policy.

DRL [17] further improves the learning time and performance of RL by taking advantages of deep learning during the learning process. Deep learning contains a series of algorithm which can automatically learn the structure of the data. Particularly, deep learning uses deep neural networks (DNN), which is neural networks with more than one layer, for data modeling. Deep Q learning network (DQN) is the most popular DRL algorithm, which trains DNNs as the Q-network. During the training process, a DNN is trained instead of Q-table used in Q-learning. DQN contains the benefit of both RL and deep learning. The agent can learn to make decisions and deal with high dimensional input using DQN.

DQN/QL has drawn increasingly attention to various applications in communications [22]. However, the application of DQN/QL to network resource management is limited. We here list a number of work for applying DQN/QL for network resource management in Table 1. Particularly, we summarize research work in terms of the scenario, system state, action, reward, agent and the algorithm used.

The authors in [23] propose a DRL approach for managing resource at the network edge. Particularly, a case for determining the VM placement is obtained with the consideration of users' mobility. The target is to minimize the operational cost of the process. The location of the VM and the number of users associated to each base station is set as the system state, and the action is the base station where the VM is going to migrate to. The communication cost of both data transmission and VM migration are used for formulating the reward function. The agent is assumed to be an abstract controller in the system. DQN method is applied for solving the problem.

In [24], DQN is applied for solving a joint task offloading and energy allocation problem in mobile edge network. The computation task queue at the mobile user (MU), the wireless channel state, the energy of the MU, and MU-base station association are used for system state. Action set includes the task offloading decision and the energy allocation policy. Reward is set as a combination of task completing performance (execution and queuing delay, drop, failure) and the economic factors. A central controller is assumed to be the agent performing the DQN. In addition, due to large space of the action set, the authors proposed algorithm based on double-DQN.

In [25], a DQN based offloading scheme is proposed in ad-hoc mobile clouds. End user acts as the agent by learning the system information including queuing information from the users and nearby mobile cloudlets and the distance between users and cloudlets. Then the agent determines the number of tasks to be offloaded to each cloudlet. Reward is set as utility minus cost in order to maximize the sum of task rate and at the same time minimize the cost of task delay and economic expense. And DQN is used for solving the problem.

In [26], a DQN based controller synchronization in distributed SDN is proposed. The controller in each domain acts as an agent for making decisions of whether to synchronize with the controllers in other domains. Accordingly, the time slot from the last synchronization with the controller in each domain is used as system state. The domain which is selected for synchronization is set as action. The objective is to minimize the average path cost (APC), which is the link weight assignment over the constructed path. DQN is applied for solving the problem.

In [27], service placement method in SDN is investigated. The controller runs the algorithm as an agent for determining the service placement in switches. System state is the decision of deleting a service in a switch. Action set is defined as installing a new service when an old service is deleted. Reward is the benefit for replacing the old service in the switch with a new service. QL based algorithm is developed for generating the service replacing decision.

In [28], a QL based multi-objective resource allocation scheme for Network Function Virtualization (NFV) orchestration has been proposed. The NFV controller acts as the agent for deciding which node and link to use for performing virtual network function (VNF) placement and route selection. Reward is a weighted function of node usage frequency, together with node and link utilization. QL based algorithm is used for devising the VNF and route selection strategy.

III. SYSTEM MODEL AND DESIGN OBJECTIVE

A. SYSTEM MODEL

The overall system model of switch controller mapping in SDN is shown in Fig. 1. The physical environment consists of a list of controllers and switches. Each controller is connected to multiple disjoint set of switches through OpenFlow protocol. We use $C = (c_1, c_2, \cdots, c_i, \cdots, c_n)$ to represent the controller set of the network, in which n is the total number of controllers. We use $S = (s_1, s_2, \cdots, s_j, \cdots, s_m)$ to represent the switch set of the network. The switch only connects to

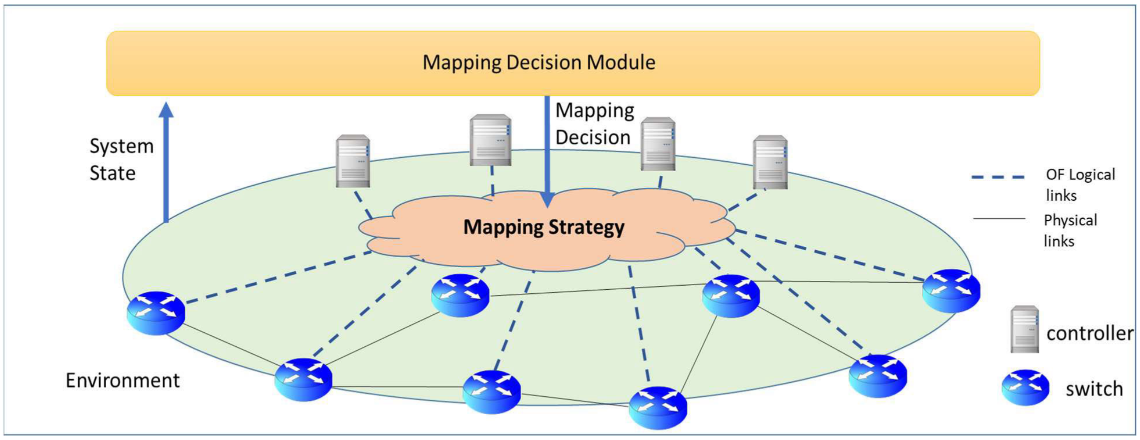


FIGURE 1. System model of switch controller mapping in SDN.

one of the controllers. And N_{c_i} is denoted as the number of switches connected to controller c_i. $d_{c_i}^{s_j}$ represents the distance for mapping the switch s_j to controller c_i. The controllers' system follows a hierarchical SDN control structure [6]. The location of the controllers has been selected and remains the same once selected. The number of controllers in the system is fixed. However, the controllers may encounter failures in which the failed controller(s) belong to set C'. In addition, we assume that each controller has fixed capacity, which limits the number of attached switches and denoted as W_{c_i}. The switch arrives at the system due to the expanding of switch equipment. The switch may also leave the system due to the switch local failure. The controller can also leave the system due to controller failure. If the controller encounters any failure, then the switches which were originally connected to the failed controller are remapped. Table 2 summarizes the notations used in the model.

The mapping decision module is assumed to be executed on a centralized management plane, e.g., a centralized controller. The mapping decision module can obtain the system state information from the network environment, i.e., the physical network. The obtained system information includes both switch features and controller features. Such information is then passed to the mapping decision module for generating mapping decision. Then, the mapping strategy is received by the control plane of the network, based on which the switch controller mapping relationship is established.

B. DESIGN OBJECTIVE

Based on the above system model, the dynamic mapping strategy is performed considering the network topology dynamic process of both switches and controllers. The objective is to devise the switch controller mapping relationship $\chi(C)$ to achieve the optimal long-term cumulative system performance including the switches' response delay, the controllers' load balancing and controllers' failure punishment. One thing to be noted is that during the optimization process, the mapping decision impacts not only the current system performance, but also the future system performance.

TABLE 2. Key notations.

Notations	Description
C	controller set
n	number of controllers
c_i	Controller i
S	switch set
m	number of switches
s_j	Switch j
N_{c_i}	number of switches connected to controller c_i
C'	failed controller set
W_{c_i}	controller capacity for controller c_i
$\chi(C)$	switch controller mapping relationship
ΔT	time duration of the multiple stage decision process
l_{c_i}	number of the left switches of controller c_i
s_{c_i}	binary variable of the working status of controller c_i
Δs_{c_i}	variation of the working status of controller c_i
q	number of arrived switches to the whole system
d_{c_i}	distance of mapping the current switch to controller c_i
l_{c_f}	number of attached switches to the failed controller c_f
p	Poisson arrival rate of switches

Accordingly, the dynamic mapping process is decomposed into multiple stage decision process with time duration ΔT for each period. At each time slot t, we denote $\mathcal{B}_1(\mathcal{D}(t))$ as the benefit of the controller response delay for each switch, $\mathcal{B}_2(\mathcal{U}(t))$ as the benefit on the controller utilization, and $\mathcal{B}_3(\mathcal{P}(t))$ as the benefit on system stability related to controller failure. And we use $\mathcal{F}()$ as the overall benefit function. Mathematically, the objective is expressed as

$$\max \left(\sum_{t=1}^{T} \mathcal{F}(\mathcal{B}_1(\mathcal{D}(t)), \mathcal{B}_2(\mathcal{U}(t)), \mathcal{B}_3(\mathcal{P}(t))) \right). \quad (1)$$

which is to maximize the accumulation of the overall benefit function $\mathcal{F}(\mathcal{B}_1(\mathcal{D}(t)), \mathcal{B}_2(\mathcal{U}(t)), \mathcal{B}_3(\mathcal{P}(t)))$ over a long-term time period T. The benefit of response delay, controller utilization and system stability denote system performance of response delay, controller utilization and system stability with specific numerical values, the overall benefit function $\mathcal{F}()$ of which the system intends to maximize. The numerical value of the system performance is generated as reward. The details of the overall benefit function $\mathcal{F}()$ and benefit functions $\mathcal{B}_1()$, $\mathcal{B}_2()$ and $\mathcal{B}_3()$ are illustrated in Section IV Eqn. (7),

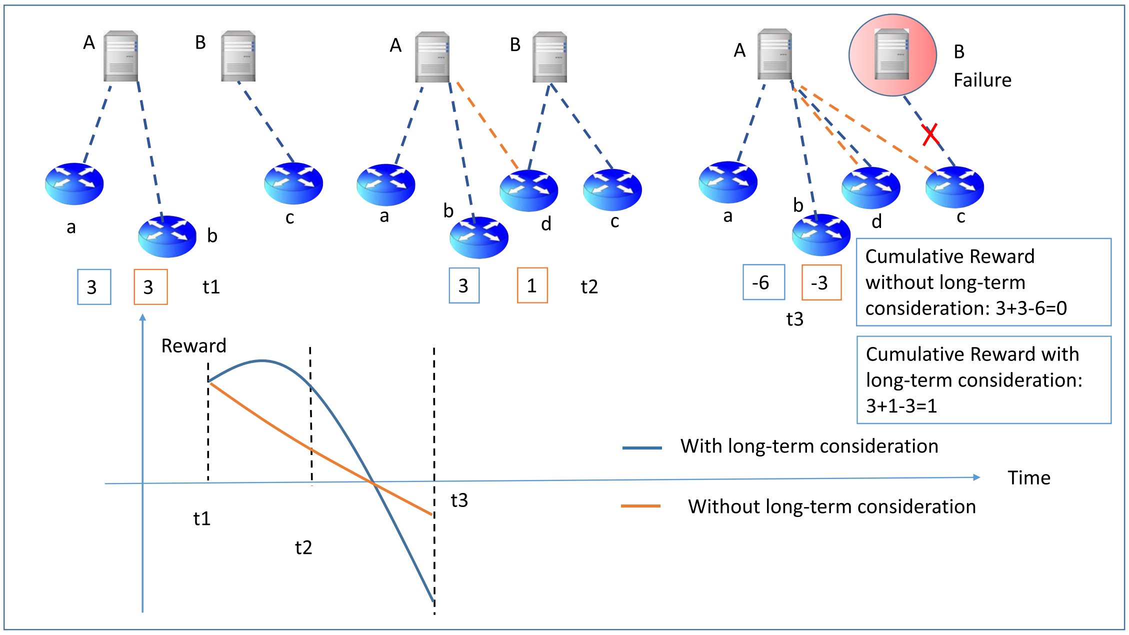

A
B
a
b
c
3
3
t1
A
B
a
b
d
c
3
1
t2
A
B
Failure
a
b
d
c
-6
-3
t3
Cumulative Reward without long-term consideration: 3+3-6=0
Cumulative Reward with long-term consideration: 3+1-3=1
Reward
With long-term consideration
Time
Without long-term consideration
t1
t2
t3

FIGURE 2. An example of the benefit of time average system performance consideration.

(4), (5) and (6). Particularly, the benefit function of response delay is declared as an inverse function of delay. The benefit function of controller utilization is an inverse function of controller utilization. The benefit function of system stability is an inverse function of affected switches due to controller failure. The overall benefit function is a weighted function of $\mathcal{B}_1()$, $\mathcal{B}_2()$ and $\mathcal{B}_3()$.

In the above problem, the network dynamics model is unknown to the decision module. The objective is to learn the network environment and obtain the optimal mapping control policy. In addition, we have the constraint that the switch can only be mapped to one controller over the process.

In the above design, we need to consider current system performance to make decision. In addition, we also make mapping decisions considering the future system performance. We illustrate the advantages considering long-term system performance in Fig. 2.

Suppose that there are two controllers A and B. At the start of the process (time instant $t1$), switch c is mapped to controller B according to the mapping strategy. Based on the overall benefit of the system, the reward at $t1$ is assumed to be $2+1$, which is the sum of reward of controller switch response delay and load balancing. At time instant $t2$, another switch d joins the system. Without long-term consideration, the system decides to map the new switch based on the instant system reward. Suppose connecting switch d to controller B can lead to a reward as $1.5 + 1.5 = 3$, which is the sum of reward of controller switch response delay and load balancing. While connecting switch d to controller A can lead to a reward as $0.5+0.5 = 1$ (shown in orange box), since the delay between switch d and controller A is larger and connecting to A will lead to worse load balancing performance. With only considering the current system reward information, the system will choose to map switch d to controller B for better reward for the current time instant.

However, for the next time instant $t3$, the network situation may change. For example, a failure may occur in controller B. Then the newly joint switch and the original connected switches need to reconnect to controller A. However, such switch migration reduces the overall system performance, leading to reduced reward of the system. Suppose at time instant $t3$, no more switch arrives at the system. Due to controller failure, switch c and d need to be remapped. The reward now is decided by the system stability, which reflects the cost of reconnecting the switches. Suppose the reward is -6, since two switches need to be remapped. On the other hand, with the consideration of long-term performance of the system, the system can learn the network dynamics. And at time instant $t2$, the system connects newly joint switch to controller A. This leads to a smaller reward than the former scheme with a value of 1. However, for the next time instant $t3$, the failure of controller B has less impact of the system, thus resulting in larger value of reward at time $t3$ with a value of -3. Through considering the long-term cumulative reward by summing over the three time instants, the latter approach can lead to better performance (reward=1) than the former approach (reward=0). As shown in Fig. 2, the orange curve (considering the long-term system performance) can lead to better long-term system performance than the blue curve in the long run.

IV. DQN BASED SWITCH CONTROLLER MAPPING APPROACH

In this section, we present the design of the DQN based switch controller mapping approach. Firstly, we present the MDP formulation of the switch controller mapping problem.

Then based on the MDP model, we present the DQN algorithm.

A. MDP FORMULATION

MDP is used for describing the optimal control problem as in discrete stochastic version [16]. In our problem for controller/switch mapping, the objective is to devise the switch controller mapping relationship to achieve the optimal long-term system performance. The dynamic mapping process is decomposed into multiple stage decision process, which is modeled as a discrete stochastic version of the optimal control problem consisting a sequential mapping decision. The mapping decision has an impact on the system current performance and also the subsequent performance. Thereby, MDP can be applied directly for modeling the controller/switch mapping problem.

To be more specific, the mapping decision module works as the agent in the MDP. The agent interacts with the SDN network continuously. Then the agent makes mapping decisions (actions) and accordingly, the network status changes and new network status information is sent back to the agent. System performance with specific numerical value is generated as rewards, which the agent intends to maximize over time by generating mapping policy.

In the following, the details of the state S, action A and reward R in the MDP model are presented.

Modeling of the system state S: The system state in the framework at time instant t includes mainly two parts: the controller features and the switch features. We use d_{c_i} to represent the distance of mapping the current switch to each controller c_i. The switch and controller are assumed to be connected through the shortest path algorithm. The switches which have already been connected to the controller may leave the system due to switch failures or mobility. The number of the left switches in controller c_i is denoted as l_{c_i}. In addition, we use q to represent the number of arrived switches to the whole system. And we use binary variable $s^t_{c_i}$ to represent the working status of the controller at t. We use $\Delta s^t_{c_i}$ as the variation of the working status of the controller at t. Particularly, we have

$$\Delta s^t_{c_i} = s^t_{c_i} - s^{t-1}_{c_i}, \tag{2}$$

in which the value of $\Delta s^t_{c_i}$ can be 0, 1 or -1. 0 represents that the status of the controller remains the same. 1 means the controller which has failed previously becomes available. And -1 means the controller which was available fails in the current time instant t.

We assume that the time duration between t and $t+1$ is small enough that only at most one switch arrival exists within the time interval. The arrival process of the switch follows Poisson process. Therefore, q is a binary variable depending on the arrival rate of the Poisson arrival process p^t, which is a time varying parameter. Note that the arrival rate refers to the arrival rate of newly joint switch which is added to the network topology. This corresponds to the network dynamics in terms of switch. When the controller l_{c_f} is failed at time instant t, the attached switch needs to be remapped. When the controller l_{c_f} fails, the attached switches which are connected to the failed controller need to be remapped to other working controllers. During the remapping process, there are still newly arrived switches joining to the whole topology. Therefore, the mapping decision module needs to process both the disconnected switches and newly arrived switches. And in the proposed system, the disconnected switches which were forced to be remapped are being processed first. Then, mapping decisions for the newly joint switches are generated by the mapping decision module.

The reason that we guarantee one switch arrives between the time interval is because for Poisson arrival process, if the sampling time is small enough, the probability that two or more switch arrivals in the interval can be neglected. In practice, the switch arrival pattern is measured. Then the interarrival between the switch arrival time can be obtained, based on which we can select the sampling time to be smaller than the minimum value of the interarrival time for guaranteeing that one switch arrives between the time interval.

Note that although OpenFlow has a mechanism to deal with failed controllers that activates a secondary controller, there are mainly two differences of the secondary controller approach comparing with the proposed scheme. First, in the design of secondary controllers, main controllers are responsible for managing the relationship of the mapping of switch to the secondary controllers. Secondary controller is activated when the main controller delegates the control to the secondary controllers based on its own working information. In the proposed approach, the mapping decision module is a centralized module, which makes mapping decision based on the whole system environment. Second, in our proposed approach, the mapping module can learn the network environment dynamics and generate the mapping decision based on the interaction with the network for achieving the long-term system performance. However, no research work in secondary controller mapping area has been designed for achieving this goal.

In summary, the system state which is obtained by the agent/mapping decision module at time instant t is presented as

$$s_t = \left(n^t, N^t_{c_i}, s^t_{c_i}, \Delta s^t_{c_i}, l^t_{c_i}, q^t, p^t, d^t_{c_i}\right). \tag{3}$$

The state of the system contains two parts. The first four parameters n^t, $N^t_{c_i}$, $s^t_{c_i}$, $\Delta s^t_{c_i}$ describe the controllers' dynamic state, including the number of controllers in the system, the number of switches attached to the controller and the working status of the controller. The last four parameters $l^t_{c_i}$, q^t, p^t, $d^t_{c_i}$ are related with the switches' dynamic state, including the leaving and arrival status of the switches and switches' distance to the controllers. The above set of system state models the dynamics of the network topology.

Modeling of the action set A: The action set of the MDP reflects the mapping relationship between the switch and the controller. In this case, the action set $a_t = \{c_1, c_2, \cdots, c_n\}$ is modeled as the controller to which the switch is mapped.

The action set has a dimension of n, which is the total number of controllers in the system t. For easy implementation, a fixed number of action set is used n. However, when failure occurs, the failed controller cannot be selected so the actual available controller set $n^t \leq n$.

Modeling of the reward R: The objectives of the switch controller mapping mainly include three parts: to reduce the response time of the switches, to balance the load for the controllers and to enhance the system stability by reducing the controller failure impact. In DQN design, the algorithm aims to maximize the long-term system reward. Therefore, the first portion of the reward which reflects the response time of the switch is mathematically presented in Eqn. (4) as

$$r^1 = \mathcal{B}_1(\mathcal{D}(t)) = \frac{1}{d_{c_i}}, \tag{4}$$

which is an inverse function of d_{c_i}. By using the above function, the reward r^1 decreases with the increasing of the distance. This can guide the DQN module to generate the switch controller mapping with smaller response time.

The second portion of the reward is for load balancing. In this part, the aim is to minimize the maximum utilization of the controller for achieving the overall balance of the controllers' load. Mathematically, the reward is represented in Eqn. (5) as

$$r^2 = \mathcal{B}_2(\mathcal{U}(t)) = \frac{1}{\gamma \max\{\frac{N_{c_1}}{W_{c_1}}, \cdots, \frac{N_{c_i}}{W_{c_i}}\}}, \tag{5}$$

where the capacity of controller c_n is denoted as W_{c_n}. γ is a punishment factor for the controller utilization. When the controller utilization goes to 1, the reward is the most punished since the value of r^2 decreases dramatically. By setting this, we avoid the situation of the overloaded controller and ensures the balance of the controllers. This is because if the case of the extreme case of the controller utilization is under control, the other controllers' utilization must be under control.

Although the switch only asks the controller for unknown packets, there are several reasons that the flow can still cause possible high processing delay in a controller. First, new arrival flow may result in high delay in a controller. It is shown that in the worst case, a network with 100 switches can have 10M new flows arrivals per second [29]. On the other hand, an SDN controller-NOX can handle around 30k flow initiation events per second [30]. Thereby, multiple controllers are needed to handle the spike of new traffic flow. Second, increasing the number of switches beyond the threshold can result in controller throughput degrading due to the increased contention across threads of controllers, TCP dynamics, and task scheduling overhead within the controller [31]. Third, the advanced functions of the controllers, such as congestion control, traffic engineering, load balancing, security and fault tolerance [32], [33], limits the number of switches a controller can manage.

The third portion of the reward involves the punishment of the failed controller. If the controller fails, the switches connected to the controller need to migrate to other controllers, resulting in the migration cost and QoS violation cost. The punishment of the migration is proportional to the total number of attached switches for the failed controllers. Mathematically, the reward of this portion is modeled in Eqn. (6) as

$$r^3 = \mathcal{B}_3(\mathcal{P}(t)) = \sum_{c_n} \min(0, \Delta s_{c_n}) \times N_{c_n}. \tag{6}$$

Here the reward is a summation over all controllers. If the value of Δs_{c_n} equals to -1, the controller c_n which was available fails. Therefore, the number of switches N_{c_n} which were attached to the controller c_n needs to be remapped. Thus, the reward of r^3 is negative, resulting in punishment on the reward. When the value of Δs_{c_n} equals to (0, 1), there is no punishment on the system since this indicates the controller status remains the same or the controllers become available. This will not change the previous mapping switches, thus resulting in neither punishment nor benefit.

To summarize, the overall reward function assuming controller c_i is selected is mathematically presented in Eqn. (7) as

$$\begin{aligned} r_t &= \mathcal{F}(\mathcal{B}_1(\mathcal{D}(t)), \mathcal{B}_2(\mathcal{U}(t)), \mathcal{B}_3(\mathcal{P}(t))) \\ &= \left(\alpha \times \frac{1}{d_{c_i}} + \beta \times \frac{1}{\gamma \max\{\frac{N_{c_1}}{W_{c_1}}, \cdots, \frac{N_{c_i}}{W_{c_i}}, \cdots, \frac{N_{c_n}}{W_{c_n}}\}}\right) \\ &\quad \times \eta(s_{c_i}) + \delta \times \sum_{c_n} \min(0, \Delta s_{c_n}) \times N_{c_n}, \end{aligned} \tag{7}$$

in which η is a function reflecting the benefit for selecting controller c_i. It is mathematically modeled in Eqn. (8) as

$$\eta = \begin{cases} \eta' & \text{when } s_{c_i} = 1 \\ \widetilde{\eta} & \text{when } s_{c_i} = 0 \end{cases} \tag{8}$$

Here, we have $\eta' > 0$ and $\widetilde{\eta} < 0$. By setting a negative reward for the failed controller, the selection of failure controller is avoided. In addition, in Eqn. (7), three adjustable weight parameters α, β and δ are used to represent the importance of the three portions. Accordingly, based on the importance of the three portions in the system design, the reward can be dynamically changed. This adds flexibility in the system design since reward function can reflect the change of the system target directly.

Based on the reward function r_t, the future discounted return R at time t is defined in Eqn. (9) as [17]

$$\begin{aligned} R_t &= \sum_{t'=t}^{T} \mu^{t'-t} r_t \\ &= \sum_{t'=t}^{T} \mu^{t'-t} (\mathcal{F}(\mathcal{B}_1(\mathcal{D}(t)), \mathcal{B}_2(\mathcal{U}(t)), \mathcal{B}_3(\mathcal{P}(t)))), \end{aligned} \tag{9}$$

which is the sum of the overall benefit $r_t = \mathcal{F}(\mathcal{B}_1(\mathcal{D}(t)),$ $\mathcal{B}_2(\mathcal{U}(t)), \mathcal{B}_3(\mathcal{P}(t)))$ discounted by μ at each time-step t until the mapping process terminates at T. μ is the discount factor where $0 \leq \mu \leq 1$ and μ reflects the trade-off between the current and future reward.

Then, the design objective of problem Eqn. (1) can be achieved by generating the policy for achieving the maximum expected return $Q^*(s, a)$ expressed as [16], [17]

$$Q^*(s, a) = \arg_\pi \max E[R_t|s_t = s, a_t = a, \pi], \tag{10}$$

which is the optimal state-action value function achieved for system state s and action a. And π^* is the switch controller mapping policy for achieving the maximum expected return $Q^*(s, a)$.

B. DQN BASED ALGORITHM

The DQN algorithm contains two parts: offline training part and online mapping part. In the offline training part, the DQN algorithm obtains the network state information with the aid of SDN controllers. The controller behavior and switch behavior are gathered, based on which a Q-network is trained by the DQN algorithm. Then during the online mapping process, the trained Q-network is used for generating the online-mapping decision. For the online mapping part, the network environment information is obtained by the SDN controller. Then such information is modeled as state parameter s_t as in MDP. Then multiple Q values correspond to different actions are generated, from which the maximum value is selected. The online mapping process of DQN is fast and efficient. Therefore, it can be used for generating real-time mapping decision.

Note that as a representative machine learning (reinforcement learning) method, DQN can learn the network environment information for producing the optimal policy for solving the MDP problem with unknown network dynamics model. By selecting properly on the training parameters, DQN algorithm can achieve the optimal policy. Therefore, the propose DQN based algorithm is the optimal control policy for solving the MDP problem.

Based on the proposed MDP model, the target of solving the switch controller mapping problem is to find the optimal policy π^* which can find $Q^*(s, a)$. The optimal policy can be generally found by repeatedly updating the Q value using the following Bellman equation [16] as

$$Q_{t+1}(s, a) = E_{s'}[r(s, a) + \mu \max_{a'} Q_t^*(s', a')|s, a], \tag{11}$$

in which s' is the state in the next time-step, a' is the possible action in the next time-step $t + 1$. Such value iteration converges to the optimal Q^* when t goes to infinity.

In the proposed strategy, we propose to use DQN other than QL for two reasons. The first reason is due to the high system status dimension. In QL approach, Q table is trained during the learning process. The Q table is a two-dimensional matrix, in which the row element corresponds to the system status and the column element represents the action space. Q-table is fast and accurate when the state and action set is relatively small. However, when the state and action set is large, Q-table is impossible to use and implement. In the problem of dynamic switch controller mapping, the system status includes both switch and controller features, and the size of the status grows exponentially with the size of the controller. Thereby, it is impossible to use such large system state in the Q-table. In DQN, a Q-network is trained through a deep learning model. The Q-network is modeled by a deep learning model, which can describe the data features and provide automatic learning from data structure [22]. The Q-network used in the DQN algorithm for switch controller mapping is DNN, which is a neural network with two or more hidden layers. Therefore, even the system state is large, the Q-network can be trained by using part of the system state. This can solve the problem of large system space.

The second reason we use DQN is because in Q learning, the algorithm requires the agent to observe all possible system status during the training process for generating the Q-table. However, in the mapping problem, the system status may be random. The switch arrival process and the controller status are unknown. And the pattern for their performance is unknown. Thereby, DQN is necessary since even the system status is partially known during the training process, action can be generated during the decision process by using Q-network.

The overall module design of the DQN-based switch controller mapping is shown in Fig. 3. In the training process shown in the red dashed line box, the objective is to generate a DNN based Q-network, in which the input is environment s and action a. The output is the value of $Q(s, a)$. The details for generating the Q-network during the training process is further illustrated in Algorithm 1.

Then once the Q-network is generated, it can be applied in the switch controller mapping process as shown in the green dashed line box. During the mapping process, by using the trained Q-network and the input network environment state s_t and available action $a_t^1, a_t^2, \cdots, a_t^n$, multiple Q values can be generated as $Q(s_t, a_t^1), Q(s_t, a_t^2), \cdots, Q(s_t, a_t^n)$. Then the optimal mapping strategy can be found by choosing the action a^* which can lead to the maximum value among the Q values. Mathematically, the mapping action is chosen based on Eqn. (12) as

$$a^* = \arg\max_a Q(s, a). \tag{12}$$

Furthermore, the training algorithm of DQN based switch controller mapping is illustrated in Algorithm 1. At the start of the algorithm, the size of the replay memory D is initialized. In addition, the initial Q-network and target Q-network is initialized with parameters θ and θ'. We use DNN as the model of Q-network. Then, the body of algorithm is repeatedly executed for a number of $\mathbb{T}$ episodes. During each episode, another loop starts with randomly generated system status. The inner loop iterates for a number of $\mathbb{S}$ steps. Then, the classical ϵ-greedy algorithm is used for finding the action set. That is, with the probability of ϵ or $1-\epsilon$, action a_t is randomly chosen among the action set or chosen by maximizing the value of $Q(s_t, a_t)$. Particularly, the value of ϵ starts at 1 at the beginning of the training step. And it decreases with the

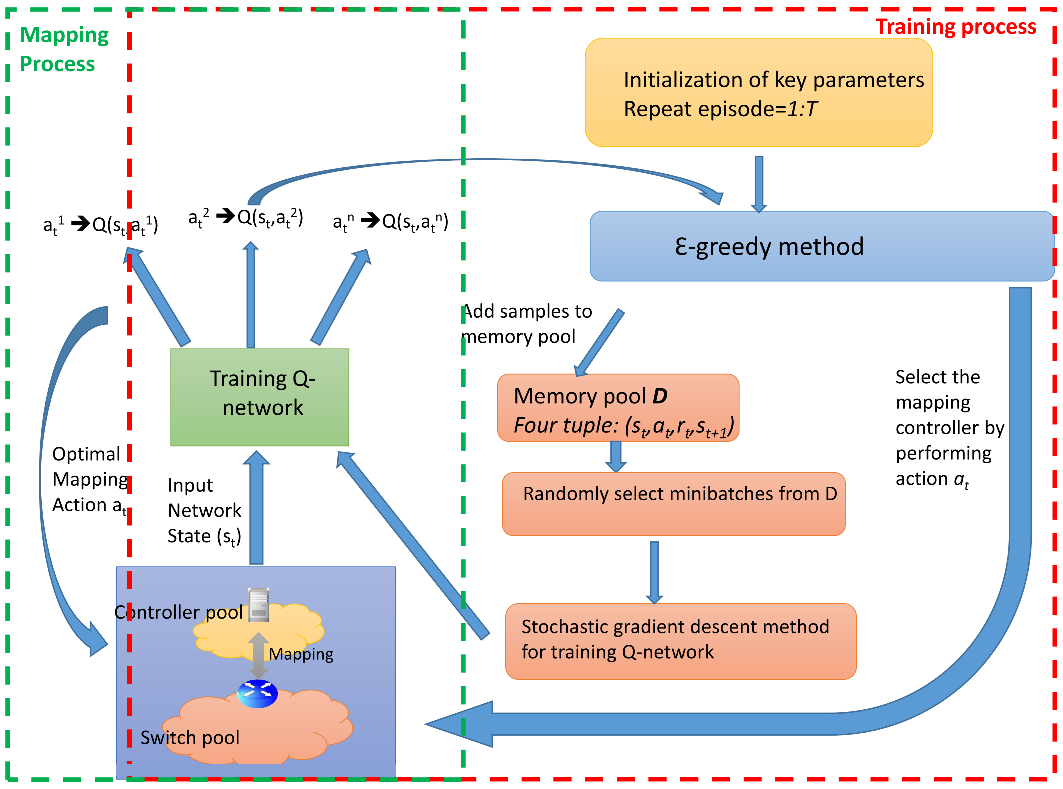

Mapping Process
Training process
Initialization of key parameters
Repeat episode=1:T
Ɛ-greedy method
$a_t^1 \rightarrow Q(s_t, a_t^1)$
$a_t^2 \rightarrow Q(s_t, a_t^2)$
$a_t^n \rightarrow Q(s_t, a_t^n)$
Add samples to memory pool
Memory pool D
Four tuple: (s_t, a_t, r_t, s_{t+1})
Select the mapping controller by performing action a_t
Training Q-network
Optimal Mapping Action a_t
Input Network State (s_t)
Randomly select minibatches from D
Stochastic gradient descent method for training Q-network
Controller pool
Mapping
Switch pool

FIGURE 3. Module design of the DQN-based switch controller mapping.

episode. The reason is because the agent knows little at the beginning of the iteration. Thereby, the agent tends to search randomly from the action set since the action leading to the best cumulative reward is uncertain. With the increasing of the iteration, the agent learns more of the network, thereby, it makes more sense to select the action which can produce the best Q value.

Then, based on the selected a_t, the next system state s_{t+1} is obtained. Accordingly, a four tuple (s_t, a_t, r_t, s_{t+1}) is generated, which is stored in the memory pool D and used for training the Q-network. For improving the independence of the data samples, minibatches of the four-tuple set are randomly selected from D. Therefore, the DNN can be trained by using both the historical data and recently generated data, which improves the independence of the training data. Then such selected data are used for training DNN parameter θ using stochastic gradient descent method through minimizing the loss function shown as

$$[r + \mu \max_{a_{k+1}} Q'(s_{k+1}, a_{k+1}; \theta') - Q(s_k, a_k; \theta)]^2. \quad (14)$$

Then for a number of M steps, the parameter θ' of the target network Q' is adjusted with Q-network parameter θ. The reason for updating Q'-network after M steps is the stability consideration [17]. The parameter of Q-network is copied to the target network Q'-network after every M steps. And for the next M steps, the Q'-network is used for performing the stochastic gradient descent method. This reduces the correlation between the target Q'-network and estimated Q-network and increases the stability of the DQN algorithm [22].

DQN based switch controller mapping can be applied into the real SDN system naturally, thanks to the separation of forwarding plane and control plane. During the training process, the local SDN controller can collect the basic network environment statistical information (switch statistics) through the southbound interface. Then such information is delivered to the mapping decision module through the northbound interface. The mapping decision module can either be a centralized server which is connected to the SDN controllers through northbound interface. Or it can connect to a root controller through the controller northbound interface in the hierarchical SDN structure [6]. In addition, the mapping decision module can obtain local controller statistically information. Then the mapping decision module acts as the agent in MDP for training the Q-network.

During the mapping process, the local controllers and the mapping decision module collect the switch and controller behavior information respectively. Then based on the trained Q-network, the mapping decision module acts as the agent for generating control policy. Then the mapping policy is distributed through the northbound interface to the local controllers. Finally, the mapping relationship is delivered through the southbound interface of the local controllers. The separation of forwarding and control plane in SDN system ensures the implementation of the proposed DQN.

Algorithm 1 Training Algorithm of DQN Based Switch Controller Mapping

```
1:  Input: MDP models
2:  Output: Switch controller mapping policy
3:  Initialize the target $Q$-network as $Q'$ with randomly chosen weight parameter $\theta'$
4:  Initialize the $Q$-network as $Q$ with randomly chosen weight parameter $\theta$
5:  Initialize the size of the memory pool $D$
6:  for Each episode $1, \cdots, \mathbb{T}$ do
7:      Randomly select an initial state $s_0$
8:      for Each step $1, \cdots, \mathbb{S}$ do
9:          With probability of $\epsilon$, randomly choose an action $a_t$
10:         With probability of $1 - \epsilon$, choose an action $a_t = \arg\max_{a_t} Q(s_t, a_t|\theta)$
11:         Based on the chosen action $a_t$, obtain current reward $r_t$ and next state $s_{t+1}$
12:         Store the $(s_t, a_t, r_t, s_{t+1})$ into the memory pool $D$
13:         Randomly select mini-batches from $D$ as samples $\{(s_k, a_k, r_k, s_{k+1})\}$
14:         Perform gradient descent regarding with parameter $\theta$ for minimizing the loss function:
                $[r + \mu \max_{a_{k+1}} Q'(s_{k+1}, a_{k+1}; \theta') - Q(s_k, a_k; \theta)]^2$.   (13)
15:         For every $M$ steps, reset $Q' = Q$.
16:     end for
17: end for
```

C. ALGORITHM COMPLEXITY

The algorithm complexity of DQN contains two parts: the off-line training part and on-line mapping part. In the off-line training part, the algorithm complexity is expressed as $O(\mathbb{T} * \mathbb{S})$, which is proportional to the overall training steps. We show that the algorithm can converge within a certain number of training steps in the next section. In the on-line mapping part, the algorithm complexity is displayed as $O(n * m)$, where n is the number of controllers and m is the number of switches.

V. NUMERICAL RESULTS

A. SIMULATION SETTINGS

A simulation platform is established for verifying the performance of the proposed framework. The simulated network consists of 4 controllers. In all scenarios, the settings of network topology are the same. Each of the controller connects to a number of switches. Initially, the switch number follows Gaussian distribution with mean and variance equaling to 15 and 12. The distance between the switch and controller distributes randomly and uniformly between $[1, 15]$. The arrival probability of the newly arrived switch in the system is assumed to be 0.8. The departure probability of each controller domain is randomly distributed between $[0.1, 0.5]$, and the number of left switches is randomly selected within the range of $[1, 4]$. The capacity of each controller is randomly selected between $[100, 300]$, considering the newly generated flow that a controller can support [29] and the controller performance degrading when the number of switches exceeds the threshold [31]. The failure probability for each controller is diverse and within the range of $[0, 0.1]$. We simulate the proposed approach in two different failure cases: Case 1 with failure probabilities set as $[0.01, 0.001, 0.0001, 0.01]$, and Case 2 with failure probabilities set as $[0.1, 0.1, 0, 0]$. The weight parameters in the reward are set to be $\alpha = 50$, $\beta = 5$ and $\delta = 50$, which are selected through testing for guaranteeing the three components in the reward in the same order. In addition, we set $\eta' = 1$ and $\widetilde{\eta} = -10$. And the punishment factor of the controller utilization is $\gamma = 2^{\max\left(\frac{N_{c_i}}{W_{c_i}}\right)}$. The values of the simulation parameters are summarized in Table 3.

Since the Q-function in DQN needs to be trained by deep learning approach, the selection of the Q-network model and training parameters is important for approaching the Q network well. In our simulations, we select DNN with two hidden layers as the Q-function model. During the training process, the learning rate is set to be 0.001. The discount factor μ is set as 0.9. The Q-function updating step M is 100. The size of memory pool D is 5000. The size of the mini batch is 24. In the ϵ-greedy method, the initial value of ϵ is set as 1, since at the beginning of the training, the agent knows little about the network and tends to select the action randomly. Then the value of ϵ reduces by 0.001 after each training step until it reaches 0.

To verify the proposed framework, the proposed DQN approach is compared with four other approaches. 1) Greedy approach. In this approach, the system selects the mapping controller based on the instantaneous system performance using Eqn. (7). 2)Random approach. In this approach, the system randomly maps the switch to the controller. 3) Lowest Latency (LL) approach. In this approach, the system generates mapping decision purely based on the distance between switch and controller. 4) Best Equilibrium (BE) approach. In this approach, the system generates mapping decision purely based on the load of controllers. Switch will select the controller which can lead to the minimum value of $\max\{\frac{N_{c_1}}{W_{c_1}}, \cdots, \frac{N_{c_i}}{W_{c_i}}\}$. For all the approaches, the mapping process in the simulation is obtained by averaging over 10 time cycles, each of which contains data set of 1000 mapping steps.

Based on the above simulation platform, we perform extensive simulations and generate results in terms of three aspects: controller switch response latency, system equilibrium and system stability. The controller switch response latency is measured using the controller switch distance. The equilibrium performance is measured as the maximum value of controller utilization $\max\{\frac{N_{c_1}}{W_{c_1}}, \cdots, \frac{N_{c_i}}{W_{c_i}}\}$. The system stability is measured as the switches affected by the failed controllers $\sum_{c_n} \min(0, \Delta s_{c_n}) \times N_{c_n}$. In the following, results in the

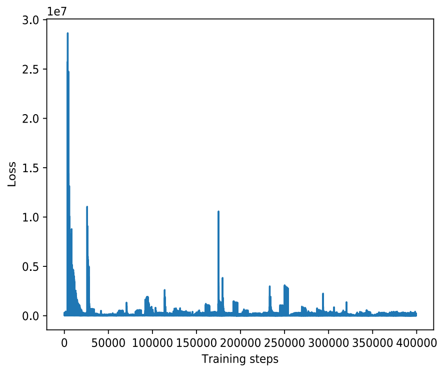
TABLE 3. Simulation parameters.

Parameters	Value
Number of controllers	4
Failure probabilities	$[0.01, 0.001, 0.0001, 0.01]$, $[0.1, 0.1, 0, 0]$
Mean of the initial number of switches connected to the controller	15
Variance of the initial number of switches connected to the controller	12
Distance between switch and controller	[1,15]
Arrival probability of newly arrived switches	0.8
Departure probability of each controller domain	$[0.1, 0.5]$
Number of left switches of each controller domain	$[1, 4]$
The capacity of each controller	$[100, 300]$
Weight parameter α	50
Weight parameter β	5
Weight parameter δ	50
Benefit of selecting working controller η'	1
Benefit of selecting failed controller $\widetilde{\eta}$	-10

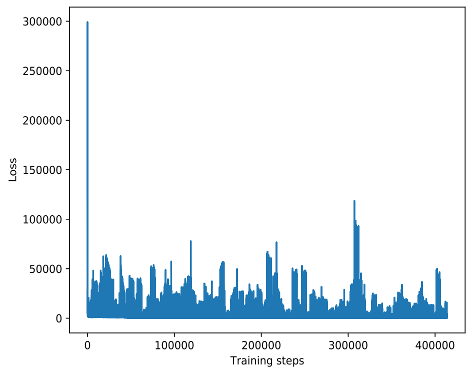
training phase of DQN approach are first plotted. Then the simulation results during the mapping process are presented and compared across different approaches.

B. RESULTS IN THE TRAINING PHASE

Fig. 4 plots the loss results versus the training steps during the training process of the DQN algorithm. It is shown that for all the cases, the loss value decreases to almost 0 with the increasing of the training steps. This demonstrates that DQN algorithm converges with the increasing of the training steps for all the scenarios. In addition, it implies that the algorithm converges within a reasonable training step. For all the simulated scenarios, the algorithm can converge within $4 * 10^5$ decision steps.

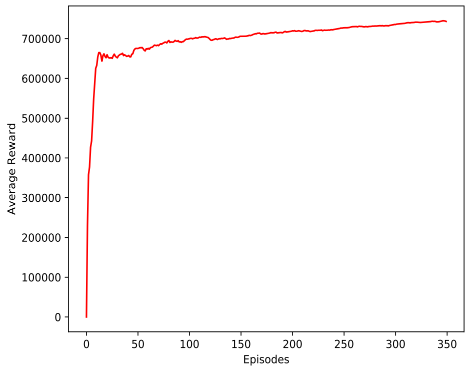


(a) Case 1

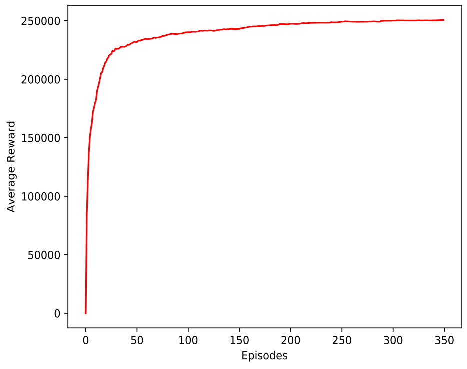


(b) Case 2

FIGURE 4. Loss versus the total training steps in the training phase.



(a) Case 1



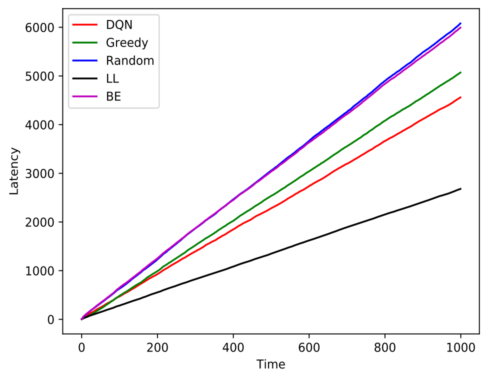
(b) Case 2

FIGURE 5. Average reward versus training episode in the training phase.

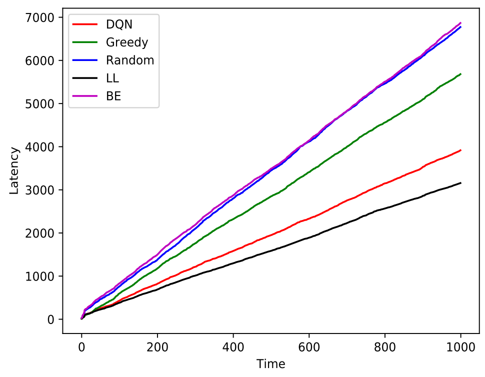
Fig. 5 displays the average reward results versus the training episode during the training process of the DQN algorithm. It can be observed that the average reward results approach to the maximum after around 150 episodes. This again indicates that the training process converges after an acceptable training episode. In addition, it can be observed that the value of the average reward in case 2 is much smaller than that of case 1 due to increasing value of failure probability.

C. CONTROLLER SWITCH RESPONSE LATENCY

The cumulative controller switch response delay versus time is presented in Fig. 6. It can be observed that for both cases,

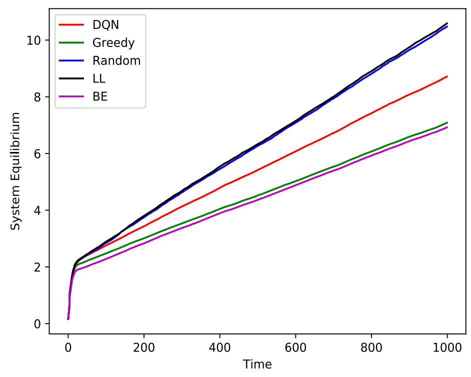


(a) Case 1

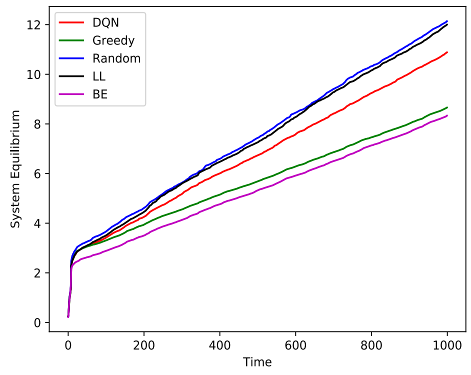


(b) Case 2

FIGURE 6. **Cumulative controller switch response latency versus time in the mapping phase.**



(a) Case 1



(b) Case 2

FIGURE 7. **Cumulative system equilibrium performance versus time in the mapping phase.**

the proposed DQN approach presented in red solid line shows better performance than greedy approach (green solid line), random approach (blue solid line) and BE approach (purple solid line). The LL approach (black solid line) shows the smallest value in latency, since switch controller mapping is decided purely on the controller switch latency. The DQN approach shows slightly better latency results than Greedy approach. The results of BE approach and random approach perform the worst and are very close, since no controller switch latency is considered in these schemes.

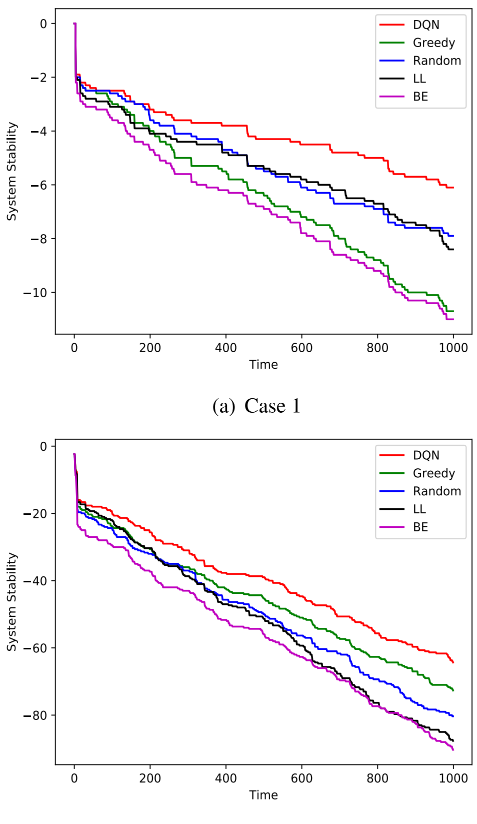
D. SYSTEM EQUILIBRIUM

The cumulative system equilibrium performance versus time is depicted in Fig. 7. The simulation results of the BE approach have the least value of system equilibrium, since mapping decision is generated by only considering system equilibrium. The Greedy approach has slightly larger system equilibrium value. The results of random approach and LL approach are very close and perform the worst, since no load balancing factor is considered during the mapping process. The results of DQN approach lies in the middle, since the DQN approach sacrifices system equilibrium for achieving system stability. For example, DQN approach may map switch to controllers which has less fail probability. Therefore, the system equilibrium performance of DQN is not as good as BE approach and Greedy approach.

E. SYSTEM STABILITY

Simulation results for the cumulative system stability versus time are plotted in Fig. 8. It can be observed that the system stability of the DQN approach performs significantly better than all other approaches. For example, during the running time, the gap of the affected switches between the DQN approach and BE approach is around 6 in case 1 and around 20 in case 2. The reason is because by generating the mapping decision based on the system long-term performance and learning the system status from the environment, the system can avoid mapping switches to possible failed controllers. This demonstrates the effectiveness of the DQN approach in protecting overall system stability. In addition, it can be observed that the system stability of case 2 is much lower than that of case 1 due to the increasing value of the failure probability in case 2.

To summarize, the compared algorithm may perform better than the proposed DQN in some specific task, such like equilibrium performance and response delay. However, the main benefit of the proposed DQN approach is to balance the overall consideration of response delay, controller utilization and system stability and to improve the cumulative system performance in the long-term consideration. The above simulation results prove that the proposed DQN approach can achieve the best system stability performance while maintaining an



(a) Case 1

(b) Case 2

FIGURE 8. **Cumulative system stability performance versus time in the mapping phase.**

acceptable latency and equilibrium performance comparing with other approaches.

VI. CONCLUSION AND DISCUSSIONS

In this paper, a DRL based approach has been proposed for solving the switch controller mapping problem in order to optimize the long-term network performance. MDP is modeled and DQN is proposed which can achieve the optimal policy for achieving long-term system performance in terms of network latency, load balancing and system stability. Through extensive simulation results, the proposed DQN method is proved to achieve the best system stability performance, while maintaining acceptable latency performance and system equilibrium performance comparing with the optimization method which performs optimization by considering only the current instantaneous system performance and the optimization method which considers network latency or load balancing separately.

With the developing trend of control and forwarding plane separation of the network, SDN provides a solid and convenient platform for applying DQN in the network optimization. The DQN based network optimization can be applied into the SDN architecture naturally, where the SDN controller and the mapping decision module can collect the basic system status and acts as an agent for generating control policy. Then the control policy can be distributed through the south/northbound interface to the forwarding engines. This ensures the implementation of the proposed DQN with a relatively low complexity. Additionally, the benefit of DQN on dealing with large and uncertain system status shed light to solve the network optimization problem in SDN system. The proposed DQN approach confirms the effectiveness of the algorithm in terms of long-term system average performance and system stability over the traditional optimization approaches. With the development of Deep RL theory, there are also other DRL algorithms, which can provide improved performance than DQN. As a future work, we will consider to apply these DRL algorithms for solving the mapping problem in SDN.

ACKNOWLEDGMENT

This article was presented at the 2020 ACM International Conference on Digital Signal Processing.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] K. Phemius and M. Thales, "OpenFlow: Why latency does matter," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2013, pp. 680–683.

[3] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 7–12.

[4] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. Internet Netw. Managment Workshop*, San Jose, CA, USA, Apr. 2010, pp. 1–6.

[5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM Sigcomm Workshop (HotSDN)*, Chicago, IL, USA, Aug. 2014, pp. 1–6.

[6] S. Schmid and J. Suomela, "Exploiting locality in distributed SDN control," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, Hong Kong, 2013, pp. 121–126.

[7] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed SDN control layer," in *Proc. IFIP Netw. Conf.*, Jun. 2014, pp. 1–9.

[8] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[9] A. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "ElastiCon: An elastic distributed sdn controller," in *Proc. 10th ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Oct. 2014, pp. 17–27.

[10] X. Ye, G. Cheng, and X. Luo, "Maximizing SDN control resource utilization via switch migration," *Comput. Netw.*, vol. 126, pp. 69–80, Oct. 2017.

[11] B. Han, X. Yang, and X. Wang, "Dynamic controller-switch mapping assignment with genetic algorithm for multi-controller SDN," in *Proc. IEEE Int. Conf Dependable, Autonomic Secure Comput., Int. Conf Pervasive Intell. Comput., Int. Conf Cloud Big Data Comput., Int. Conf Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2019, pp. 980–986.

[12] T. Hu, P. Yi, Z. Guo, J. Lan, and J. Zhang, "Bidirectional matching strategy for multi-controller deployment in distributed software defined networking," *IEEE Access*, vol. 6, pp. 14946–14953, 2018.

[13] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.

[14] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and Y. J. Guo, "Multi-timescale decentralized online orchestration of software-defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2716–2730, Dec. 2018.

[15] M. He, A. Varasteh, and W. Kellerer, "Toward a flexible design of SDN dynamic control plane: An online optimization approach," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 4, pp. 1694–1708, Dec. 2019.

[16] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[17] V. Mnih, M. G. Bellemare, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[18] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2013, pp. 672–675.

[19] F. J. Ros and P. M. Ruiz, "On reliable controller placements in software-defined networks," *Comput. Commun.*, vol. 77, pp. 41–51, Mar. 2016.

[20] G. Cheng, H. Chen, Z. Wang, and S. Chen, "DHA: Distributed decisions on the switch migration toward a scalable SDN control plane," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, Toulouse, France, May 2015, pp. 1–9.

[21] C. J. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, May 1992. [Online]. Available: https://link.springer.com/article/10.1023/A:1022676722315

[22] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[23] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo, "Resource management at the network edge: A deep reinforcement learning approach," *IEEE Netw.*, vol. 33, no. 3, pp. 26–33, May 2019.

[24] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[25] D. Van Le and C.-K. Tham, "A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 760–765.

[26] Z. Zhang, L. Ma, K. Poularakis, K. K. Leung, and L. Wu, "DQ scheduler: Deep reinforcement learning based controller synchronization in distributed SDN," in *Proc. ICC-IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[27] Z. Zhang, L. Ma, K. K. Leung, L. Tassiulas, and J. Tucker, "Q-placement: Reinforcement-learning-based service placement in software-defined networks," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1527–1532.

[28] J. Chen, J. Chen, R. Hu, and H. Zhang, "QMORA: A Q-learning based multi-objective resource allocation scheme for NFV orchestration," in *Proc. IEEE 91st Veh. Technol. Conf. (VTC-Spring)*, Antwerp, Belgium, May 2020, pp. 1–6.

[29] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, 2010, pp. 256–280.

[30] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," in *Proc. Workshop Hot Topics Netw. (HotNets)*, 2009, pp. 1–6.

[31] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services*, 2012, pp. 1–10.

[32] J. Hua, L. Zhao, S. Zhang, Y. Liu, X. Ge, and S. Zhong, "Topology-preserving traffic engineering for hierarchical multi-domain SDN," *Comput. Netw.*, vol. 140, pp. 62–77, Jul. 2018.

[33] H. Farhady, L. HyunYong, and N. Akihiro, "Software-defined networking: A survey," *Comput. Netw.*, vol. 81, pp. 79–95, Apr. 2015.

JIA CHEN (Member, IEEE) received the B.Eng. degree in communication engineering from the Beijing University of Posts and Telecommunications, in 2005, and the master's and Ph.D. degrees from the Department of Electrical and Electronic Engineering, University College London, in 2006 and 2010, respectively. She is currently an Associate Professor with the National Laboratory of Next Generation Internet Interconnection Device, Beijing Jiaotong University. Her current research interests include protocol design and optimization for the future Internet.

SHIHUA CHEN received the bachelor's and master's degrees in communication engineering from Beijing Jiaotong University, Beijing, China, in 2017 and 2020, respectively. He is currently working with Tencent Technology Ltd. His research interests include next generation Internet protocol design and applications.

XIN CHENG received the bachelor's degree in communication engineering from Beijing Jiaotong University, Beijing, China, in 2019, where he is currently pursuing the academic master's degree in communication engineering. His research interest includes applying AI in the next generation Internet protocol design.

JING CHEN (Graduate Student Member, IEEE) received the master's degree in computer application technology from the Henan University of Science and Technology, Henan, China, in 2017. She is currently pursuing the Ph.D. degree in communication engineering with Beijing Jiaotong University. Her research interests include network function virtualization (NFV) and artificial intelligence (AI).

• • •