Contents lists available at ScienceDirect

e-Prime - Advances in Electrical Engineering, Electronics and Energy

journal homepage: www.elsevier.com/locate/prime

Deep Reinforcement Learning- based load balancing strategy for multiple controllers in SDN

Min Xiang [a], Mengxin Chen [a,*], Duanqiong Wang [b], Zhang Luo [b]

[a] Key Laboratory of Industrial Internet of Things and Networked Control, Chongqing University of Posts and Telecommunications, Ministry of Education, Chongqing 400065, China

[b] PowerChina Chongqing Engineering Co., Ltd., Chongqing 400060, China

ARTICLE INFO

Keywords:
Software-Defined Network (SDN)
Controller load balancing
Deep Reinforcement Learning (DRL)
Switch immigration

ABSTRACT

In Software-Defined Network (SDN) with multiple controllers, static mapping relationship between switches and controllers may cause some controllers to be overloaded, while some controller resources are underutilized. A Deep Reinforcement Learning-based switch migration strategy (DRL-SMS) is proposed to solve the load imbalance problem in the multi-controller control plane. Based on Markov Decision Process (MDP), modeling analysis is performed for SDN to obtain system state, migration action set, and system reward. Q-values of switch migration actions are obtained by fitting approximate function using Double Deep Q-Network (DDQN), and then the DDQN is trained by using the experience replay mechanism to optimize Q-Network parameters. After training, the DRL-based strategy calculates the Q-value in the current system state and selects the migration action corresponding to the maximum Q-value to perform switch migration. Simulation experiments show that DRL-SMS can effectively balance the controller load and significantly reduce the balance time.

1. Introduction

With the high-speed construction of 4G and 5G networks, network traffic and services are growing rapidly, and the expansion of networks such as data centers and wide area networks is limited by traditional network architectures. As a new network architecture, Software-Defined Network (SDN) has successfully solved the rigidity problem of traditional network by decoupling the control plane from the data plane [1–3], B4 [4] and Software-Defined WAN (SD-WAN) [5] have been proposed successively. However, as the network continues to expand, the centralized controller may be overloaded when managing too many switches. Moreover, a failure of the controller may cause the entire network to go down due to the problem of a single point of failure. Recent studies have proposed a series of multi-controller architectures, which can be divided into distributed architecture such as HyperFlow [6], Onix [7], Ethane [8], and vertical architecture such as Kandoo [9].

The scalability and reliability of the network can be improved through multiple controllers, but there is a new problem that the static mapping relationship between switches and controllers prevents the control plane from adapting to changes in traffic. As mentioned in [10], real networks may behave with great variability in temporal and spatial dimensions. Therefore, dynamic traffic will result in uneven traffic of switches in the distributed network. If the mapping relationship between switches and controllers is static, when network traffic changes suddenly, it is likely that some controllers will be overloaded, while some will have underutilized resources.

In this paper, aiming to solve the controller load imbalance problem of SDN, a Deep Reinforcement Learning-based switch migration strategy (DRL-SMS) is proposed. In the DRL-SMS model, a DRL framework is designed to interact with the network and learn how to migrate switches to get the maximum reward. After proper training, Switch migration actions can be determined quickly and accurately. Simulation results demonstrate that the DRL-based load balancing strategy can effectively balance the controller load and significantly reduce the balance time.

The main contributions of this paper are summarized as follows:

A Deep Reinforcement Learning (DRL) architecture is introduced for SDN to solve SMP, where the network state is formalized as a two-dimensional matrix. The DRL takes the two-dimensional matrix as input and generates migration decisions as outputs.

To solve the problem of over-estimation and model oscillation caused by using Q-Network, a Double Deep Q-Network (DDQN) model is designed, and the DDQN is trained by using the experience replay mechanism. After the training phase, the DDQN model can quickly and accurately decide how to migrate switches between controllers.

We conducted simulation experiments on the DDQN model, and the results show that the DDQN-based solution can efficiently improve the load balancing effect and significantly reduces the balancing time.

* Corresponding author.
E-mail address: cqupt_cmx@163.com (M. Chen).

https://doi.org/10.1016/j.prime.2022.100038
Received 25 October 2021; Received in revised form 25 February 2022; Accepted 14 April 2022

2. Related works

In recent years, the research on controller load balancing can be categorized into the optimization of controllers and the development of switch migration strategies. The controller optimization strategy concentrates on optimizing the number and location of controllers to realize a balanced distribution of controller load. Sallahi and Hilaire [11] determined optimal numbers, location and connection relationships between switches and controllers within the network by introducing a mathematical model. Yao et al. [12] modeled the controller optimization problem as an integer linear programming problem and obtained the minimum number of controllers and the optimal position. Liao et al. [13] proposed a controller deployment algorithm based on density clustering to obtain the optimal number of controllers. Lin et al. [14] improved the original artificial bee colony algorithm to determine the number of controllers and deployment locations, reducing the number of controllers and network latency. Shi et al. [15] proposed a controller deployment strategy based on genetic algorithm, which can effectively improve the load balance of controllers in large-scale SDN. Hu et al. [16] considered the transmission delay between controllers and switches in case of link failure and established a controller deployment model to minimize the transmission delay. However, these methods did not apply to the scenario where the traffic changes in real-time. In addition, Guo et al. [17] proposed a load variance-based Synchronization mechanism to achieve controller load balancing by controlling the forwarding path of flows, which effectively reduces the synchronization overhead of the controllers.

The introduction of the OpenFlow1.3 [18] provided the possibility to balance the controller load through switch migration. When a controller is overloaded, the switches controlled by the overloaded controller can be migrated to other controllers with underutilized resources to achieve load balancing. The Switch Migration Problem (SMP) is often expressed as an optimization problem [19,20], and its complexity is confirmed to be NP-hard. Cheng et al. [21] dynamically migrated switches from the overloaded controller to controllers with underutilized resources and provided a game decision mechanism to maximize resource utilization, but the strategy lacked a game trigger mechanism, and the whole process generated a large additional network overhead. Kim et al. [22] proposed a heuristic algorithm called Progressive-Harmony Search (P-HS) to solve load imbalance in the control plane. The algorithm solved the dynamic controller configuration problem and minimized the communication delay between switches and controllers. However, the cost of migration, the migration efficiency and new problems caused by migration were not considered comprehensively. Liang et al. [23] realized load balancing by constructing a controller cluster and dynamically moving switches, but the operation increased controller response time. Li et al. [24] proposed a dynamic switching migration strategy based on fuzzy multi-objective particle swarm optimization to solve the load balancing problem in the control plane by solving multi-objective optimization problems. The algorithm comprehensively considered cost, efficiency and balance, but the calculation volume of the algorithm and the storage volume of data were very large, and it also had some negative impacts on the stability of the network. Ali et al. [25] proposed the maximizing resource utilization migration algorithm (MUMA). When a controller was overloaded, the controller randomly selected a switch for migration. The algorithm did not consider that the new controller may overload after migration. Most of the above schemes rely heavily on iterative optimization algorithms or solutions that convert SMP to other models, which may result in either time-consuming or less satisfactory in terms of performance.

The development of artificial intelligence provides a new solution for SMP. To develop an artificial learning system, Kober et al. [26] proposed a Reinforcement Learning (RL) model, which mainly described that the agent interacts with the current environment in the trial and error method, and learned from the environment feedback. Li et al. [27] advanced a Reinforcement Learning-based load balancing mechanism (RL-LBM) to solve the problem of switch migration conflicts and achieve controller load balancing. However, using the Reinforcement Learning framework was not very effective for learning tasks that need to deal with unlimited spatial states. The proposal of Deep Reinforcement Learning (DRL) presents a new solution for SMP. Sun et al. [28] used a DRL model to solve the switch migration problem, which effectively improves resource utilization and flow request processing capability. However, all controllers running the DRL model increase the controller burden and the controller performance bottleneck may affect the network performance. To summarize the above description, the methods to achieve controller load balancing are shown in Table 1.

The DRL framework combines the powerful perceptual capabilities of deep neural networks with the learning capabilities of Reinforcement Learning, which can handle huge input states and efficiently generate control behaviors for target systems. When solving NP-hard problems such as Switch Migration Problem (SMP), the Deep Reinforcement Learning model interacts with the environment to learn switch migration action. At the same time, the system reward is defined according to the requirements for training the model. After training, the model can output effective switch migration actions. The DRL framework has been applied in the fields of grid technology [29], robot control [30] and autonomous driving [31]. Therefore, we intend to use the DRL framework to solve the Switch Migration Problem in SDN.

3. Proposed work

3.1. Problem description

The context of our study is the SDN with multiple controllers. In Fig. 1, there are three controllers, and each controller is connected with several switches to form a subdomain. When switches connected to controller c_2 generate a large number of flow requests, c_2 may be overloaded and the resources of other controllers may not be fully utilized. According to OpenFlow1.3, switches controlled by the overloaded controller can be migrated to other controllers to meet load requirements. As shown in Fig. 1, v_4 is selected to be migrated and c_1 is selected as the target controller. After migration, v_4 is managed by c_1, and the entire network works properly. Therefore, the controller load imbalance problem can be solved through switch migration. The main challenge is how to migrate efficiently, so a switch migration strategy needs to be designed to efficiently implement controller load balancing.

3.2. System model

The SDN is analyzed as a system whose state at the current moment is only related to the state at the previous moment and is irrelevant to the state before the previous moment. Considering the Markov property of the system and the time-varying characteristics of the flow, the Markov Decision Process is used to analyze the system formally, and the model can be described as $M = \{S_t, A, P, R_t, \eta\}$, where S_t represents the system state at time t; A represents the switch migration action set; P represents the probability of state transitions; R_t represents the system reward and describes the environment feedback; η is the attenuation factor, $\eta \in [0, 1]$. Moreover, the migration strategy $\pi(a|S_t)$ is defined as the probability of taking action $a(a \in A)$ in the S_t. To analyze system state, switch migration actions and system reward by using the Markov Decision Process, the relevant definitions can be described as follows:

3.2.1. System state

The network can be described as an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ represents the switch set, and $n(n \geq 1)$ represents the number of switches. $C = \{c_1, c_2, ..., c_m\}$ represents the controller set, and $m(m \geq 1)$ represents the number of controllers. $U = \{u_1, u_2, ..., u_m\}$ represents the controller load capacity set, $u_h(u_h \in U)$ represents the maximum load of the controller $c_h(c_h \in C)$. The packet-in message request rate that the switch $v_i(v_i \in V)$ sends to the controller

Table 1
The methods to achieve controller load balancing.

type	method	advantages	disadvantages
The Optimization of Controllers	Controller Placement Strategy [11]	The strategy simultaneously determined the optimal number, location, and type of controllers to minimize deployment costs.	It took too much time to run in large-scale networks, so this strategy was not suitable for large-scale networks.
	Capacitated K-center Strategy [12]	The strategy could significantly reduce the number of controllers and the maximum controller load.	Inability to adapt to real-time changes in traffic.
	Genetic Algorithm (GA) [15]	The strategy could effectively reduce the propagation delay and queuing delay of flow requests.	Inability to adapt to real-time changes in traffic.
Switch Migration	GAME-Switch Migration (GAME-SM) Strategy [21]	The strategy could reduce traffic setup time and overhead.	The strategy lacked a game trigger mechanism, and the whole process generated a large additional network overhead.
	Progressive-Harmony Search (P-HS) Strategy [22]	The algorithm solved the dynamic controller configuration problem and minimized the communication delay between switches and controllers.	The cost of migration, the migration efficiency and new problems caused by migration were not considered comprehensively.
	Dynamic Load Rebalancing method [23]	The strategy could improve throughput.	It increased controller response time.
	Dynamic Switching Migration Strategy based on Fuzzy Multi-objective Particle Swarm Optimization [24]	The strategy could effectively reduce network transmission delay and migration cost.	The calculation volume of the algorithm and the storage volume of data were very large, and it also had some negative impacts on the stability of the network.
	Maximizing Resource Utilization Migration(MUMA) Algorithm [25]	The migration cost is very low, and the migration time is fast.	The algorithm did not consider that the new controller may overload after switch migration.
	Reinforcement Learning-based Load Balancing Mechanism (RL-LBM) [27]	The strategy could effectively reduce migration costs and improved the response time of flow requests.	Since using the Reinforcement Learning framework was not very effective for learning tasks that need to deal with unlimited spatial states, the load balancing performance is not good after switch migration.
	Dynamic Controller Workload Balancing Scheme based on Multi-agent Reinforcement Learning [28]	The strategy effectively improved the control plane's flow request processing capabilities.	The strategy did not consider model oscillations, and the controller performance bottleneck might affect the network performance.

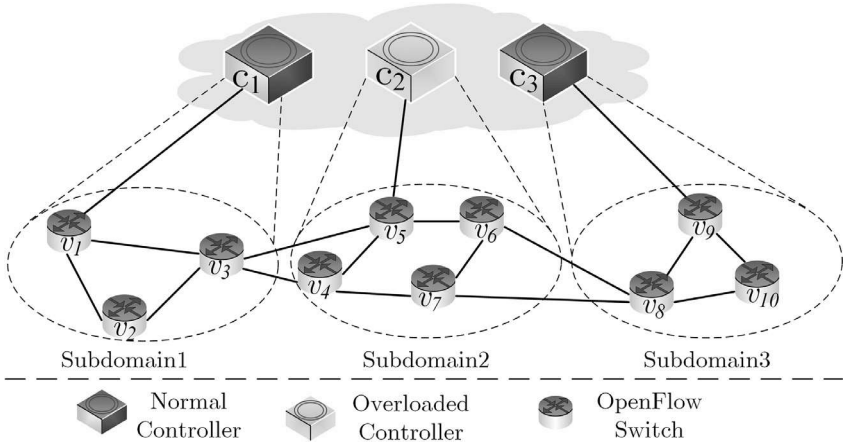


Fig. 1. Multiple controller deployment architecture.

$c_h (c_h \in C)$ at time t is expressed as $l_{hi}(t)$. The mapping relationship between controller c_h and switch v_i can be defined as:

$$g_{hi}(t) = \begin{cases} 1, & v_i \text{ is managed by } c_h \\ 0, & \text{others} \end{cases} \tag{1}$$

When the packet-in message request rate generated by switch $v_i (v_i \in V)$ is obtained, the system state S_t can be described as:

$$S_t = \begin{bmatrix} l_{11}(t)g_{11}(t) & l_{12}(t)g_{12}(t) & \dots & l_{1n}(t)g_{1n}(t) \\ l_{21}(t)g_{21}(t) & l_{22}(t)g_{22}(t) & \dots & l_{2n}(t)g_{2n}(t) \\ \dots & \dots & \dots & \dots \\ l_{m1}(t)g_{m1}(t) & l_{m2}(t)g_{m2}(t) & \dots & l_{mn}(t)g_{mn}(t) \end{bmatrix} \tag{2}$$

3.2.2. Action set

For an SDN with m controllers and n switches, there are $m \times n$ types of switch migration actions, of which n types are "non-migration" actions. Migration actions are numbered from 1 to $m \times n$, that is $a \in A$, $A = \{1, 2, ..., m \times n\}$. The target controller number w and the switch number e to be migrated can be described as:

$$e = a\%n \tag{3}$$

$$w = ceil(a/n) \tag{4}$$

where $ceil$ represents the round-up function. When executing the migration action a, v_e is migrated to c_w. Fig. 2 is a schematic diagram of switch migration. Consider an SDN with three controllers and ten switches, $m = 3$, $n = 10$, and $A = \{1, 2, ..., 30\}$. If $a = 4$, the number of the switch which needs to be migrated is 4, and the number of the target controller is 1. When executing the migration action a, v_4 is migrated to c_1.

3.2.3. Controller load

There are various sources of SDN controller load, including receiving packet-in messages, sending flow entries and communicating with

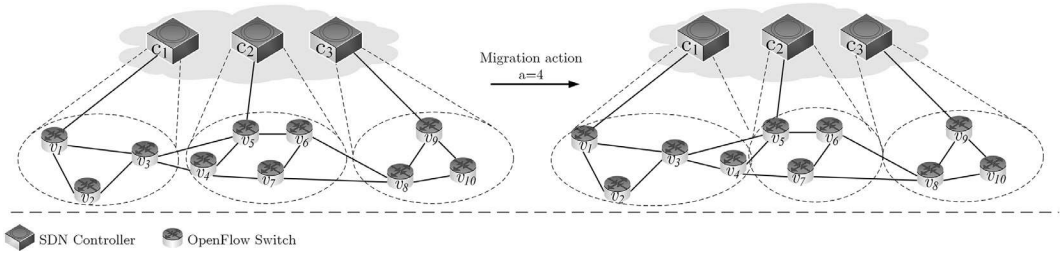


Fig. 2. Schematic diagram of the switch migration.

other controllers. The main source of SDN controller load is processing packet-in messages [27]. The controller load is simplified as packet-in message request rates received by controller c_h. The controller load can be described as:

$$L_h(t) = \sum_{i=1}^{n} l_{hi}(t) g_{hi}(t), \quad (1 \le h \le m) \tag{5}$$

3.2.4. Controller load ratio

Since different controllers have various load capacities, it is inaccurate to use the dispersion of controller load to reflect the effect of load balancing. The load ratio $\beta_h(t)$ is defined to reflect the ratio of controller resource consumption, which can be expressed as:

$$\beta_h(t) = \frac{L_h(t)}{u_h}, \quad (1 \le h \le m) \tag{6}$$

where u_h indicates the load capacity of c_h, and $\beta_h(t) \ge 0$. The controller c_h does not manage any switches when $L_h(t) = 0$, and c_h is named unloaded. The average load ratio of controllers can be described as:

$$\overline{\beta}(t) = \frac{\sum_{h=1}^{m} \beta_h(t)}{m} \tag{7}$$

where all controllers will not be unloaded in practical applications, so $\overline{\beta}(t) > 0$.

3.2.5. Controller load balancing rate

The controller load balancing rate $D(t)$ is designed to represent the dispersion of the controller load ratio and the average load ratio of controllers, which can be defined as:

$$D(t) = \frac{\sqrt{\sum_{h=1}^{m} \left(\beta_h(t) - \overline{\beta}(t)\right)^2 / m}}{\overline{\beta}(t)} \tag{8}$$

where $D(t)$ is used to measure the degree of controller load balancing, in other words, it can measure the performance of the load balancing strategy.

3.2.6. Switch migration cost

The switch migration cost mainly comes from two aspects: (1) The controller sends flow-mod messages to the switches which need to be migrated; (2) The switch sends migration requests to the target controller. The switch migration cost $F_i(t)$ can be expressed as:

$$F_i(t) = \sum_{h=1}^{m} \left[\sigma_{hi}(t) g_{hi}(t) f_{hi}\right] + \sum_{k=1}^{m} \left[\Phi_{ki}(t) g_{ki}(t+1) f_{ki}\right], \quad (k \ne h) \tag{9}$$

where $\sigma_{hi}(t)$ represents the number of flow-mod messages sent by the overload controller c_h to switch v_i, and f_{hi} represents the number of hops from switch v_i to controller c_h. $\Phi_{ki}(t)$ represents the number of request messages sent by switch v_i to the target controller c_k, $g_{ki}(t+1)$ represents the management relationship between controller c_k and switch v_i after migration, f_{ki} represents the number of hops from v_i to c_k. As shown in Fig. 2, the flow request generated by switch v_7 needs to go through at least v_7, v_6 and v_5 to reach controller c_2. Therefore, the number of hops from v_7 to c_2 is three.

3.2.7. System reward

After switch migration, the quality of the strategy is reflected by whether the system state becomes better or worse, that is, to define the system reward. To avoid frequent migration of switches only considering the improvement of controller load after migration, and to avoid migrating switch to a remote controller, the switch migration cost should be considered. System reward can be described as:

$$R_t = \begin{cases} \frac{D(t) - D(t+1)}{\sum_{i=1}^{n} F_i(t)}, & \sum_{i=1}^{n} F_i(t) \ne 0 \\ 0, & \sum_{i=1}^{n} F_i(t) = 0 \end{cases} \tag{10}$$

where $D(t)$ and $D(t+1)$ denote the controller load balancing rate before and after the switch migration, $\sum_{i=1}^{n} F_i(t)$ is the switch migration cost during the migration. When executing the "non-migration" actions, $\sum_{i=1}^{n} F_i(t) = 0$. It is defined that $R_t = 0$ when the "non-migration" actions are performed.

3.2.8. Optimal strategy

The optimal strategy $\pi^*(a|S_t)$ is defined to maximize the system reward after taking action $a(a \in A)$ in system state S_t. The optimal strategy can be defined as:

$$\pi^*(a|S_t) = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \eta^t \cdot R_t\right] \tag{11}$$

where η is the attenuation factor, and $\eta \in [0, 1]$. When η is closer to 0, the agent cares more about the immediate system reward; When η is closer to 1, the agent cares more about the long-term system reward.

3.3. Deep Q-Network model of SDN

In SDN with multiple controllers, the system state is random due to the time-varying flow. Traditional reinforcement learning, such as Q-learning, cannot be processed directly. Deep Q-Network (DQN) [32] is a deep neural network combined with Q-learning. The DQN can handle unlimited spatial states so that it can handle complex system states for SDN. It applies the approximate function $Q(S_t, a|\theta_t)$ $(a \in A)$ to indicate the output obtained by inputting the system state S_t, where θ_t represents the parameter of the DQN. The process of DQN fitting approximate function and updating Q-value through value iteration can be expressed as:

$$Q(S_t, a|\theta_t) = Q(S_t, a|\theta_t) + \alpha\left[R_t + \eta \max_{a'} Q(S_{t+1}, a'|\theta_{t+1}) - Q(S_t, a|\theta_t)\right], \quad (a, a' \in A) \tag{12}$$

where α represents the learning rate. Through formula (12), the optimal Q-value can be obtained by convergence, and the target Q-value y_t is defined as:

$$y_t = R_t + \eta \max_{a'} Q(S_{t+1}, a'|\theta_{t+1}), \quad (a' \in A) \tag{13}$$

DQN can get the Q-value of all migration actions by processing the SDN system state. Fig. 3 demonstrates the processing method of the DQN.

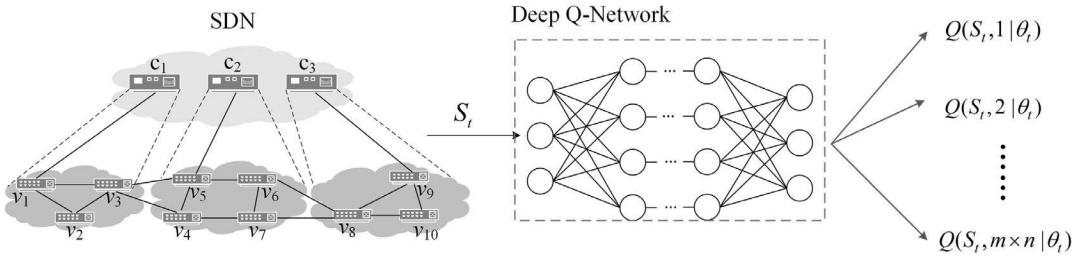


Fig. 3. Processing method of the Deep Q-Network.

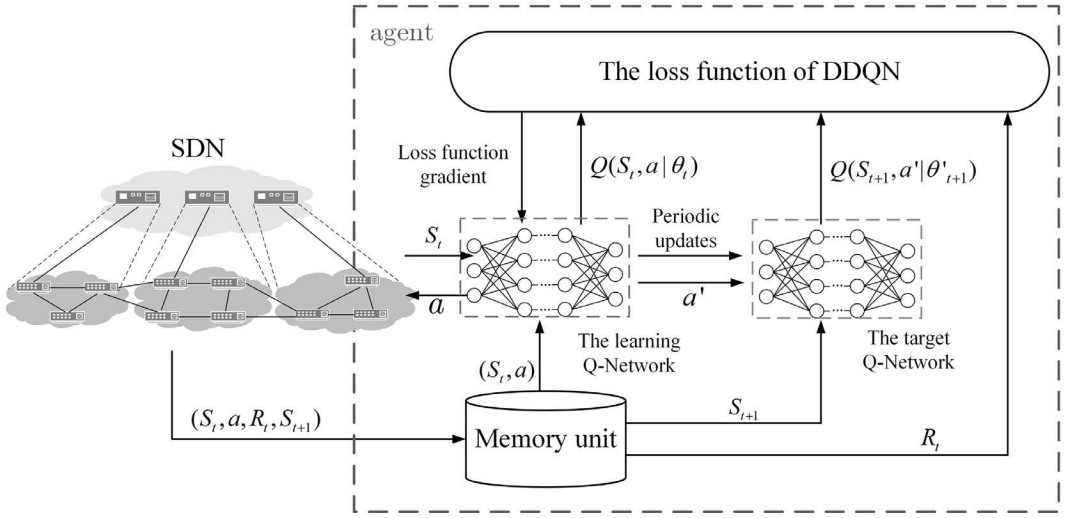


Fig. 4. Overall system architecture for switch migration strategy.

Further, the loss function is defined to optimize the parameters of the DQN by performing back-propagation. The loss function can be expressed as:

$$Y(\theta_t) = \left[y_t - Q(S_t, a|\theta_t)\right]^2, \quad (a \in A) \tag{14}$$

3.4. Switch migration strategy based on DDQN

The switch migration strategy based on DQN trains the neural network to obtain the optimal Q-value of the approximate function. However, using one Q-Network to select the migration action and calculate the target Q-value may cause $Q(S_t, a|\theta_t)$ and $Q(S_{t+1}, a'|\theta_{t+1})$ large or small simultaneously, which may lead to model oscillation. Beside, using Eq. (13) to calculate the target Q-value may lead to overestimation. Therefore, two identical Q-Networks are utilized, namely the learning Q-Network and the target Q-Network. The learning Q-Network selects migration actions and updates model parameters. The target Q-Network calculates the target Q-value. Then the target Q-Network is periodically updated with the learning Q-Network parameters to speed up the training. The calculation of the target Q-value and the selection of actions use different Q-Networks to avoid overestimation. In addition, since deep learning requires independent samples distribution, the experience replay is used to put the samples (S_t, a, R_t, S_{t+1}) $(a \in A)$ gained by interacting with the environment into the memory unit. During the training phase, a part of the samples is randomly selected to optimize the learning Q-Network parameters. The overall framework of the switch migration strategy based on Double Deep Q-Network (DDQN) [33] is shown in Fig. 4.

The target Q-value of the DDQN is expressed as:

$$y_t = R_t + \eta Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a'|\theta_{t+1})|\theta'_{t+1}), \quad (a' \in A) \tag{15}$$

where θ'_{t+1} represents the target Q-Network parameters, θ_{t+1} represents the learning Q-Network parameters.

In SDN with multiple controllers, a global controller is added to connect to domain controllers. The global controller periodically collects SDN state messages and interacts with the environment as an agent. In the training phase, the SDN state is formalized into a two-dimensional matrix according to Eq. (2) and fed to the learning Q-Network. The learning Q-Network uses convolutional neural networks and fully connected layers to get the Q-value for migration actions. The action is selected according to the ε-greedy algorithm to obtain the system reward and the next state. Then, the state, action, system reward, and next state are stored in the memory unit, from which random samples are taken to train the learning Q-Network, and the target Q-Network is updated periodically. In the online phase, the trained Q-Network parameters are loaded. When the global controller detects that the domain controller is overloaded, the SDN state is formalized as a two-dimensional matrix and fed into the model to obtain the Q-value. The switch migration action corresponding to the maximum Q-value is then output. The switch migration strategy based on DDQN is shown in Algorithm 1.

3.5. Complexity analysis

The complexity analysis of the DRL-SMS consists of the following two main aspects:

1. In the training phase, the two-dimensional matrix S_t is input to the neural network, and the Q-value of each migration action is output after calculation. The migration actions are selected and the neural network parameters are updated. The time complexity of each neural network training and update is $O(m \times n^2)$, where m represents the number of controllers, and n represents the number of switches. Due to the number of iterations of the algorithm is T, the time complexity is $O(T \times m \times n^2)$.

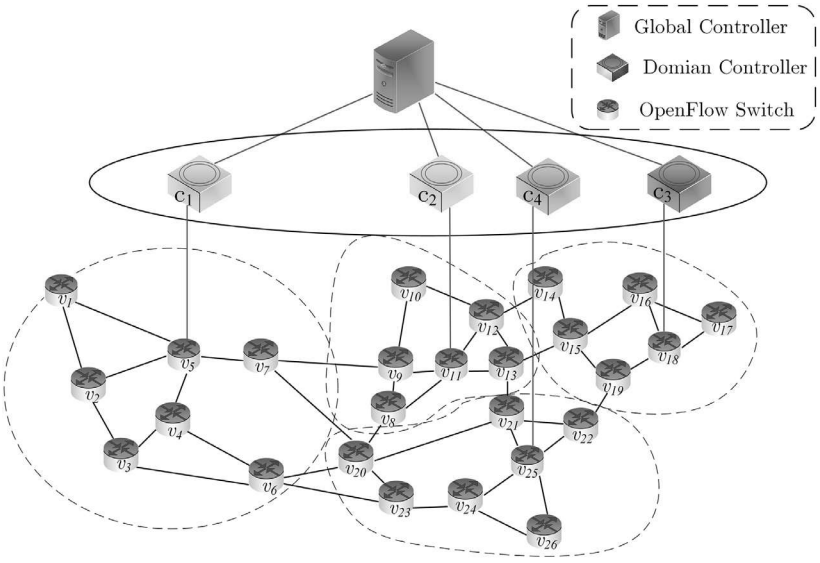


Fig. 5. The JANOS-US topology.

2. In the online phase, the Q-value of each migration action is calculated through the neural network, and the action corresponding to the maximum Q-value is selected for migration. Thus, the time complexity is $O(m \times n^2)$.

To ensure that the parameters obtained from the neural network training are effective and reliable, the number of training iterations must be large enough, so the T is a large positive integer. In real scenarios, the number of controllers m and the number of switches n are relatively small due to the limitation of network size. Therefore, the DRL-SMS algorithm has a relatively large time overhead in the training phase and a small time overhead in the online phase. In summary, the DRL-SMS algorithm sacrifices a certain amount of training time and reduces deployment time.

4. Evaluation and results discussion

4.1. Simulation environment

The platform for the simulation experiments of this paper is Ubuntu 16.04. Mininet is used to build the network topology. The controller of the control plane is Ryu. To test the performance of this strategy in a real network, the actual network topology JANOS-US is selected from SNDlib [34] to evaluate the switch migration strategy. The JANOS-US is a network with 26 nodes and 84 links. In fact, the strategy can fit most Software-Defined Networks with multiple controllers through extensive simulation experiments. The network is divided into four regions, and each area deploys a domain controller. Meanwhile, all domain controllers are connected to the global controller. The JANOS-US topology is shown in Fig. 5.

The Cbench is used to test the load capacity of 4 domain controllers as 14259flows/s, 12530flows/s, 12395flows/s and 14376flows/s, set the load threshold to 12000flows/s, 10000flows/s, 10000flows/s and 12000flows/s. The traffic characteristics are very complex in real network environments, which makes theoretical analysis intractable. According to literature [35], flow requests follow a Poisson process. Moreover, to ensure that the proposed strategy can be applied to random flow requests, flow requests follow a Poisson process added to a periodical fluctuation [28]. Use the iperf to simulate packet-in messages with different rates. The attenuation factor η is 0.2, which means that the system cares more about the immediate reward. The learning rate α is 0.01. ε is initialized to 0.9 and gradually increases over time until it becomes 1.

Table 2
Three switch migration strategies.

Strategy	description
DRL-SMS(this paper)	DRL-SMS runs the DDQN at global controller. The global controller decides switch migration based on the network state.
RL-LBM	RL-LBM designs the RL model to solve the switch migration problem and achieve controller load balancing. The RL model can learns switch migration actions by interacting with the environment.
DPCLB	DPCLB builds a distributed migration decision domain model, determines the migration switch according to the migration cost, and selects the target controller based on the greedy algorithm.

4.2. Simulation analysis

4.2.1. Migration effect

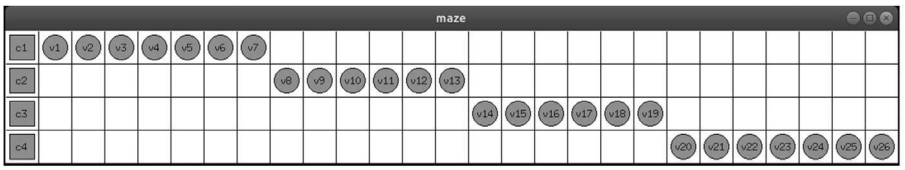
Test results visualize the mapping relationship between controllers and switches to facilitate the observation of the switch migration effect, and the results are shown in Fig. 6. Fig. 6a shows that in the initial state, c_1 manages $v_1 - v_7$, c_2 manages $v_8 - v_{13}$, c_3 manages $v_{14} - v_{19}$, and c_4 manages $v_{20} - v_{26}$, which corresponds to the management relationships in Fig. 5. Due to dynamic changes in flow, some controllers in the network may be overloaded. The five figures from Fig. 6b to 6e show the migration actions of the switch when the controller is overloaded. Fig. 6b indicates that when c_1 is overloaded, switch v_7 is migrated from c_1 to c_2. Fig. 6c shows that when c_4 is overloaded, switch v_{22} is migrated from c_4 to c_3. Fig. 6d and 6e shows that when c_2 is overloaded, switch v_6 is migrated from c_2 to c_4, and then switch v_9 is migrated from c_2 to c_1.

The result shows that the proposed strategy DRL-SMS can execute various switch migration actions according to different system states. Moreover, the strategy can migrate switches controlled by the overload controller to other controllers that are not overloaded.

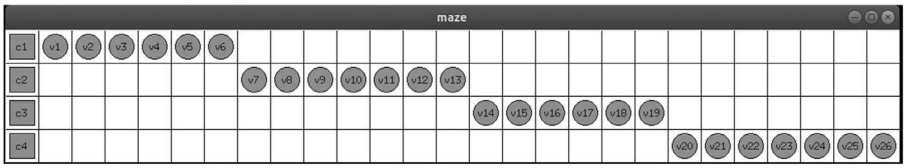
To evaluate the performance of DRL-SMS, two typical switch migration strategies are used for comparative analysis. They are the Reinforcement Learning-based load balancing mechanism (RL-LBM) [27] and the load balancing strategy based on distributed policy (DPCLB) [36]. The descriptions of three switch migration strategies are shown in Table 2.

4.2.2. Controller load balancing rate

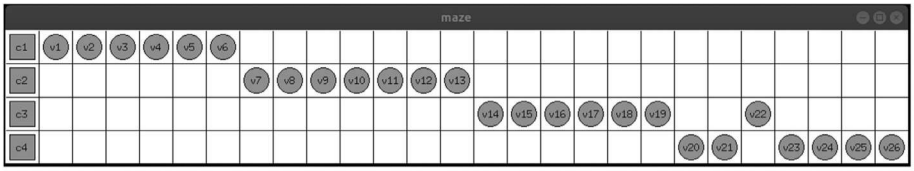
The load balancing rate is a crucial indicator to measure the quality of the strategy when comparing load balancing strategies. The purpose



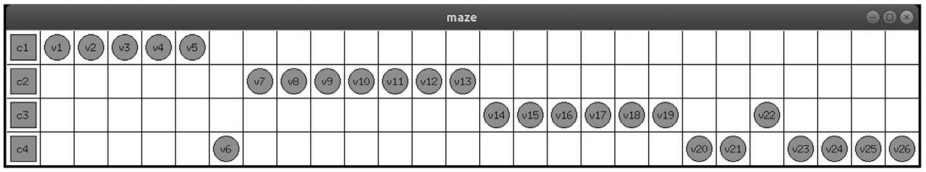
(a) Initial state



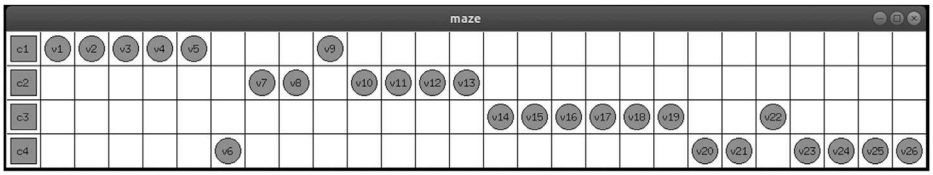
(b) when c_1 is overloaded, v_7 is migrated to c_2



(c) when c_4 is overloaded, v_{22} is migrated to c_3



(d) when c_2 is overloaded, v_6 is migrated to c_4



(e) when c_2 is overloaded, v_9 is migrated to c_1

Fig. 6. Schematic diagram of migration effect.

of the three strategies is to reduce the load balancing rate of the control plane. Flow requests with 24 h Poisson distribution are generated using iperf. The average load balance rate of each period after different strategies are applied is shown in Fig. 7.

In Fig. 7, the load balancing rate of the RL-LBM is higher than the other two strategies. There are two main reasons for this phenomenon, one of which is that the reinforcement learning uses matrices to record the system states and actions, it is suitable for situations where the environment states and actions are discrete and limited in number. However, in real networks, the system states are infinite, using reinforcement learning is not effective. The other reason is that the reinforcement learning used by RL-LBM lacks the generalization ability, so the load balancing rate is higher after migration. DRL-SMS uses the neural network to fit the Q-value, which dramatically improves the generalization ability of the strategy, and the load balance rate is at a low level. DPCLB considers the minimum migration cost and optimizes the solution through a greedy algorithm, and the load balancing rate is also kept at a low level.

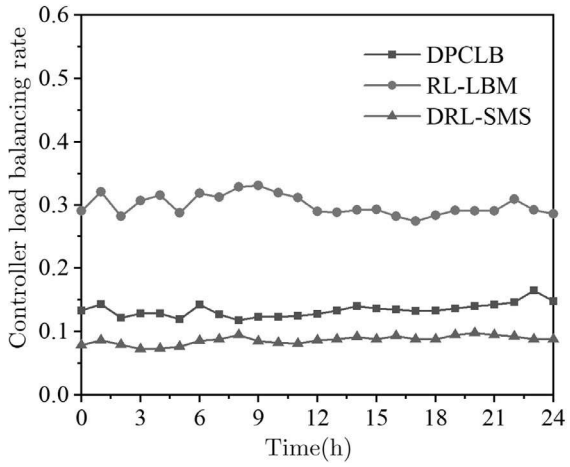


Fig. 7. The load balancing rate variation in different load balancing strategies.

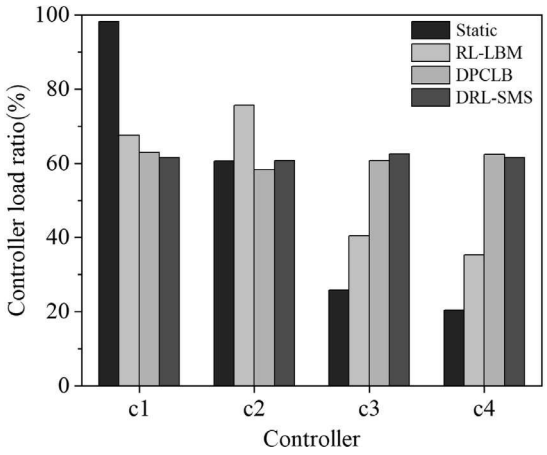


Fig. 8. The controller load ratio variation in different load balancing strategies.

Table 3
The average load ratio of controllers.

Strategy	The average load ratio of controllers
Static	0.508
RL-LBM	0.543
DPCLB	0.605
DRL-SMS	0.610

4.2.3. Controller load ratio

The controller load ratio is also an important indicator to measure the quality of load balancing strategies, and the experimental result is shown in Fig. 8. In static conditions, the switches cannot migrate, and the controller load ratio varies significantly. RL-LBM uses the reinforcement learning strategy to migrate the switches controlled by c_1. The controller load ratio is still quite different after the migration due to the lack of generalization ability. DRL-SMS uses the DDQN to extract network features, which reduces the amount of calculation and improves the generalization ability of the strategy. The controller load ratio difference is the least. DPCLB considers the minimum migration cost and effectively enhances the migration efficiency, so the controller load ratio has a slight difference.

The average load ratio of controllers is shown in Table 3. The average load ratio of DPCLB and DRL-SMS has little difference and is at a high level.

4.2.4. Balance time

The balance time represents the time consumed to execute the migration strategy. In the simulation experiment, the load of controller c_1 exceeds the threshold by increasing the request rate. After implementing three load balancing strategies, the balance time and the controller load are compared to measure the quality of the strategy.

Fig. 9 shows how the load of controller c_1 changes over time when executing DRL-SMS, RL-LBM and DPCLB. In 30 s, c_1 received a large number of requests from switches, and the controller load exceeded the threshold. DRL-SMS completed the migration operation and reduced the controller load to a normal level in 40 s. RL-LBM completed the migration operation in about 50 s. The balance time of DPCLB was the longest, it completed the migration operation in about 60 s, but the controller load was closer to the normal level than RL-LBM. From this simulation experiment, we can conclude that DRL-SMS can effectively reduce the balance time.

From the above analysis, it is clear that DRL-SMS can effectively improve the load balancing effect of SDN with multiple controllers and reduce the balance time. In addition, since the DRL-based strategy can adapt to the time-varying flow because of its strong generalization capability, DRL-SMS can be applied to networks of different sizes after proper training.

Algorithm 1
The switch migration strategy based on DDQN.

1: BEGIN
2: Input: Network graph $G = (V, E)$, Controller set C, Number of iterations T, Action set A, Exploration rate ε, Learning rate α, Attenuation factor η, Number of samples x, Target Q-Network parameter update frequency z
3: Output: Switch migration action a
4: Initialization:initialize the memory unit
initialize the learning Q-Network and the target Q-Network parameters
5: Training phase:
6: Periodically obtain the network state of SDN and formalize the network state as a two-dimensional matrix S_t
7: For $t = 1, 2, ..., T$ do
8: S_t is input to the learning Q-Network, and the learning Q-Network uses convolutional neural networks and fully connected layers to get the Q-value for migration actions.
9: The action is selected according to the ε-greedy algorithm, as follows:
ith probability $1 - \varepsilon$ select a random action a
otherwise select $a = \arg\max_a Q(S_t, a|\theta_t)$
10: Action a is executed in the simulator, the reward R_t and the state S_{t+1} are obtained
11: Store transition (S_t, a, R_t, S_{t+1}) in memory unit
12: Sample random x sets of data from memory unit to train the learning Q-Network
13: Set
$$y_t = \begin{cases} R_t + \eta Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a'|\theta_{t+1})|\theta\prime_{t+1}) & , \ S_{t+1} \ is \ not \ final \ state \\ R_t & , \ S_{t+1} \ is \ final \ state \end{cases}$$
14: According to the loss function $(y_t - Q(S_t, a|\theta_t))^2$, network parameters θ_t are updated by gradient back propagation of the neural network
15: if $t\%z == 0$ then
16: Update the target Q-Network parameters $\theta\prime_t = \theta_t$
17: End if
18: Update network state $S_t = S_{t+1}$
19: End for
20: Online phase:
17: Load the Q-Network parameters
18: Formalize the current system state into a two-dimensional matrix and input it into the learning Q-Network
19: Get the switch migration action a corresponding to the maximum Q-value
20: END.

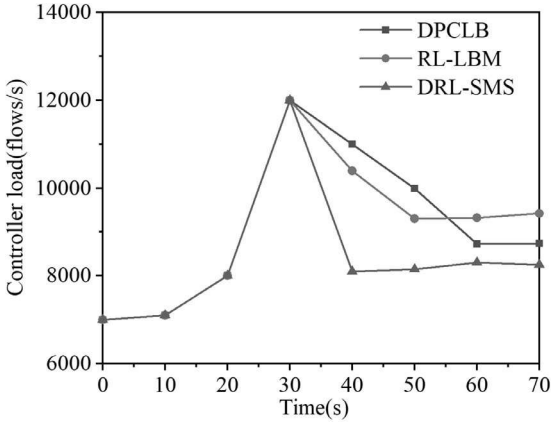


Fig. 9. The balance time variation in different load balancing strategies.

5. Conclusions

The research on the problem of load imbalance for multi-controller in SDN clarifies that the static connection between switches and controllers is a key factor leading to controller load imbalance, and proposes the necessity of switch migration. A Markov decision process is used to formally describe the system state, migration action set and system reward of the SDN. Combined with Deep Reinforcement Learning,

a DRL-based switch migration strategy is proposed. According to our simulation experiments, the DRL-based strategy has good performance in balancing controller load, increasing average load rate and reducing balance time. This proves that artificial intelligence technology can be used to solve NP-hard challenges like the switch migration problem. Since the DRL-SMS algorithm has a relatively large time overhead in the training phase, our future research will optimize training methods to reduce training time.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.prime.2022.100038.

References

[1] M.C. Chuang, C.C. Yen, C.J. Huang, Bandwidth-aware rescheduling mechanism in SDN-based data center networks, Electronics 10 (15) (2021) 1774.
[2] A. Guo, C. Yuan, Network intelligent control and traffic optimization based on SDN and artificial intelligence, Electronics. 10 (6) (2021) 700.
[3] H. Lee, J.Y. Choi, Static equivalence checking for OpenFlow networks, Electronics 10 (18) (2021) 2207.
[4] S. Jain, A. Kumar, S. Mandal, et al., B4: experience with a globally-deployed software defined WAN, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 3–14.
[5] Z. Yang, Y. Cui, B. L, et al., Software-defined wide area network (SD-WAN): architecture, advances and opportunities, in: Proceedings of the 28th International Conference on Computer Communication and Networks, Valencia, Spain, 2019, pp. 1–9.
[6] A. Ferraiuolo, M. Zhao, A.C. Myers, et al., HyperFlow: a processor architecture for nonmalleable, timing-safe information flow security, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), Toronto, Canada, 2018, pp. 1583–1600.
[7] T. Koponen, M. Casado, N. Gude, et al., Onix: a distributed control platform for large-scale production networks, in: Proceedings of the 9th Usenix Symposium on Operating Systems Design and Implementation, Vancouver, Canada, USENIX Association, 2010, pp. 351–364.
[8] M. Casado, N. McKeown, S. Shenker, From ethane to SDN and beyond, ACM SIGCOMM Comput. Commun. Rev. 49 (5) (2019) 92–95.
[9] S.H. Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, New York, USA, 2010, pp. 19–24.
[10] A. Dixit, F. Hao, S. Mukherjee, et al., Towards an elastic distributed SDN controller, in: Proceedings of the Second ACM SIGSAC Workshop on Hot topics in Software Define Networking, New York, NY, USA, 2013, pp. 7–12.
[11] A. Sallahi, M.S. Hilaire, Optimal model for the controller placement problem in Software Defined Networks, IEEE Commun. Lett. 19 (1) (2015) 30–33.
[12] G. Yao, J. Bi, Y. Li, On the capacitated controller placement problem in Software Defined Networks, IEEE Commun. Lett. 18 (8) (2014) 1339–1342.
[13] J. Liao, H. Sun, J. Wang, et al., Density cluster based approach for controller placement problem in large-scale Software Defined Networkings, Comput. Netw. 112 (15) (2016) 24–35.
[14] N. Lin, Q. Zhao, L. Zhao, et al., A novel cost-effective controller placement scheme for software-defined vehicular networks, IEEE Internet Things J. 8 (18) (2021) 14080–14093.
[15] J. Shi, Y. Xie, L. Sun, et al., Multi-controller placement strategy based on latency and load in Software Defined Network, J. Electron. Inf. Technol. 41 (8) (2019) 1869–1876.
[16] Y. Hu, L. Wang, T. Hu, et al., Placement strategy of controller based on control delay reliability in SDN, J. Commun. 39 (S2) (2018) 144–150.
[17] Z. Guo, M. Su, Y. Xu, et al., Improving the performance of load balancing in Software-Defined Networks through load variance-based synchronization, Comput. Netw. 68 (2014) 95–109.
[18] S. Liao, X. Hong, C. Wu, et al., Prototype for customized multicast services in Software Defined Networks, in: Proceedings of the 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2014, pp. 315–320.
[19] H. Chen, G. Cheng, Z. Wang, A game-theoretic approach to elastic control in Software-Defined Networking, China Commun. 13 (5) (2016) 103–109.
[20] X. Ye, G. Cheng, X. Luo, Maximizing SDN control resource utilization via switch migration, Comput. Netw. 126 (2017) 69–80.
[21] G. Cheng, H. Chen, H. Hu, et al., Dynamic switch migration towards a scalable SDN control plane, Commun. Syst. 29 (9) (2016) 1482–1499.
[22] S. Kim, S.K. Ebay, B. Lee, et al., Load Balancing for Distributed SDN with Harmony Search, in: Proceedings of the 16th IEEE Annual Consumer Communications and Networking Conference, Las Vegas, NV, USA, 2019, pp. 1–2.
[23] C. Liang, K. Ryota, M. Hiroshi, et al., Scalable and crash-tolerant load balancing based on switch migration for multiple OpenFlow controller, in: Proceedings of the Second International Symposium on Computing and Networking, Shizuoka, Japan, 2014, pp. 171–177.
[24] J.M. Li, X.L. Hu, M.Q. Zhang, Research on dynamic switch migration strategy based on FMOPSO, in: Proceedings of the IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), IEEE, 2018, pp. 913–917.
[25] N. Ali, N. Nima, H. Mehdi, Load balancing mechanisms in the Software Defined Networks: a systematic and comprehensive review of the literature, IEEE Access 6 (2018) 14159–14178.
[26] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: a survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.
[27] Z. Li, X. Zhou, J. Gao, et al., SDN controller load balancing based on reinforcement learning, in: Proceedings of the IEEE 9th International Conference on Software Engineering and Service Science, Beijing, China, 2018, pp. 1120–1126.
[28] P. Sun, Z. Guo, G. Wang, J. Lan, Y. Hu, MARVEL: enabling controller load balancing in Software-Defined Networks with multi-agent reinforcement learning, Comput. Net. 177 (2020).
[29] S. Yao, J. Gu, P. Zhang, et al., Resilient load restoration in microgrids considering mobile energy storage fleets: a Deep Reinforcement Learning approach, in: Proceedings of the IEEE Power & Energy Society General Meeting (PESGM), Montreal, Canada, 2020, pp. 1–5.
[30] F. Liu, C. Chen, Z. Li, et al., Research on path planning of robot based on Deep Reinforcement Learning, in: Proceedings of the 39th Chinese Control Conference (CCC), Shenyang, China, 2020, pp. 3730–3734.
[31] J.D. Liao, T. Liu, X.L. Tang, et al., Decision-making strategy on highway for autonomous vehicles using Deep Reinforcement Learning, IEEE ACCESS 8 (2020) 177804–177814.
[32] V. Mnih, K. Kavukcuoglu, D. Silver, et al., Human-level control through Deep Reinforcement Learning, Nature 518 (7540) (2015) 529–533.
[33] P. Lv, X. Wang, Y. Cheng, et al., Stochastic double Deep Q-Network, IEEE Access 7 (2019) 79446–79454.
[34] S. Orlowski, R. Wessaly, M. Pioro, et al., SNDlib1.0-survivable network design library, Networks. 55 (3) (2010) 276–286.
[35] T. Wang, F. Liu, H. Xu, An efficient online algorithm for dynamic SDN controller assignment in data center networks, IEEE/ACM Trans. Netw. 25 (5) (2017) 2788–2801.
[36] T. Hu, J. Zhang, J. Wu, et al., Controller load balancing mechanism based on distributed policy in SDN, Chin. J. Electron. 46 (10) (2018) 2316–2324.

The first author information, Full name: Min Xiang, Position (Prof., Assoc. Prof., Asst. Prof., PHD. Candidate., Dr., Mr. Ms. etc.): Prof, Affiliation (Organization or University): Chongqing University of Posts and Telecommunications, Research Interests: smart grid, industrial internet and automation control, Contact address: Chongqing University of Posts and Telecommunications

The second author and corresponding author information.Full name: Mengxin Chen, Position (Prof., Assoc. Prof., Asst. Prof., PHD. Candidate., Dr., Mr. Ms. etc.):Mr. (Organization or University): Chongqing University of Posts and Telecommunications, Research Interests: Software-Defined Network, Contact address: Chongqing University of Posts and Telecommunications