Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 3866143, 17 pages
https://doi.org/10.1155/2022/3866143

Research Article

ALBRL: Automatic Load-Balancing Architecture Based on Reinforcement Learning in Software-Defined Networking

Junyan Chen,[1,2] **Yong Wang**,[2] **Jiangtao Ou**,[3] **Chengyuan Fan**,[3] **Xiaoye Lu**,[2] **Cenhuishan Liao**,[2] **Xuefeng Huang**,[2] **and Hongmei Zhang**[1]

[1]*School of Information and Communication, Guilin University of Electronic Technology, Guilin 541004, China*
[2]*School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China*
[3]*AI Sensing Technology, Chancheng District, Foshan 528000, China*

Correspondence should be addressed to Yong Wang; ywang@guet.edu.cn

Received 2 March 2022; Revised 26 March 2022; Accepted 11 April 2022; Published 2 May 2022

Academic Editor: Junjuan Xia

Copyright © 2022 Junyan Chen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Due to the rapid development of network communication technology and the significant increase in network terminal equipment, the application of new network architecture software-defined networking (SDN) combined with reinforcement learning in network traffic scheduling has become an important focus of research. Because of network traffic transmission variability and complexity, the traditional reinforcement-learning algorithms in SDN face problems such as slow convergence rates and unbalanced loads. The problems seriously affect network performance, resulting in network link congestion and the low efficiency of inter-stream bandwidth allocation. This paper proposes an automatic load-balancing architecture based on reinforcement learning (ALBRL) in SDN. In this architecture, we design a load-balancing optimization model in high-load traffic scenarios and adapt the improved Deep Deterministic Policy Gradient (DDPG) algorithm to find a near-optimal path between network hosts. The proposed ALBRL uses the sampling method of updating the experience pool with the SumTree structure to improve the random extraction strategy of the empirical-playback mechanism in DDPG. It extracts a more meaningful experience for network updating with greater probability, which can effectively improve the convergence rate. The experiment results show that the proposed ALBRL has a faster training speed than existing reinforcement-learning algorithms and significantly improves network throughput.

1. Introduction

Due to the rapid development of network communication technology and the significant increase in network terminal equipment, service traffic in networks shows rapid growth. At the same time, these fast-growing service flows also bring new challenges to existing communication networks. The new-intelligent network architecture represented by software-defined networking (SDN) has led to an effective change in the Internet. In SDN, addressing network throughput is a substantial issue in traffic engineering (TE) [1–3]. Managing a large number of network devices and significant network traffic in high-load network conditions is complicated, and traffic forwarding decisions in complex SDN network situations have proven to be NP-hard issues [4].

Traditional routing-optimization schemes always forward traffic based on the open shortest path first (OSPF) [5], equal-cost multipath routing (ECMP) [6], or optimized heuristic algorithms [7]. The OSPF protocol routes all flow requests singularly to the shortest path. The ECMP protocol uses multiple links simultaneously to increase transmission bandwidth. However, based on fixed forwarding rules, these approaches are prone to link congestion, and they cannot meet the demands of the exponential growth of heavy traffic. Due to a lack of historical experience in data learning, heuristic algorithms can only establish models for fixed networks. It is challenging to determine network parameters when a network changes, and scalability is limited. In recent years, because of its robust learning algorithm and excellent performance, deep learning has been gradually applied to

computer networks. Many studies have used supervised learning methods in SDN to realize intelligent network management [8–14]. However, supervised learning requires a great many datasets for training, and slow decision-making in dynamic network scenarios also poses a problem. Compared with supervised learning, reinforcement learning (RL) is an online learning that can change its actions to obtain optimal rewards through continuous exploration, learning, and trial. Reinforcement learning opens a new way for solving complex network problems [15]. Some researchers have used traditional algorithms of reinforcement learning such as deep Q-learning network (DQN) [16–19], proximal policy optimization (PPO) [20], deep deterministic policy gradient (DDPG) [21–28], and twin delayed deep deterministic policy gradient (TD3) [29–31]. The DQN algorithm uses Q-tables to store value functions, but it leads to excessive memory overhead and sizeable computational complexity when the network size increases. The deterministic policy gradient algorithms such as DDPG and TD3 still encounter the problems of a slow convergence rate and of instability. They heavily affect network performance before convergence.

In this paper, we propose an automatic load-balancing architecture based on reinforcement learning (ALBRL) to solve the TE problem of SDN in terms of throughput. ALBRL adapts the improved DDPG algorithm to find the near-optimal path between network hosts by receiving information about the network links in the data plane. Subsequently, it generates an inter-link-weight matrix and creates the forwarding path flexibly using Dijkstra's algorithm. In this process, ALBRL evaluates the merit of the action by the link evaluation information at the next moment and updates the network based on the evaluation results. Compared with the DDPG algorithm, the proposed ALBRL uses a SumTree binary tree structure to update the experience pool sampling. The probability of sampling experiences with a more significant temporal-difference error (TD-error) and network load-balancing-rate factor becomes more prominent. ALBRL selects those more critical experiences to the network update, so the network parameters such as throughput can reach convergence values faster. The impact of the convergence of the reinforcement-learning model can be reduced when the network is changed.

The main contributions of our paper are as follows:

(1) An automatic load-balancing optimization algorithm in SDN based on an improved, DDPG, reinforcement-learning model is proposed, which uses a SumTree binary tree structure to optimize the extraction method of the empirical-playback mechanism in the traditional DDPG algorithm. It can effectively solve the problems of a long trial time and the unstable performance of SDN networks caused by the slow convergence rate of the DDPG algorithm

(2) A mathematical model for evaluating network load balancing in high-load traffic scenarios is proposed as the SDN network performance evaluation and is used as a reward for the ALBRL algorithm

(3) An experimental system using the proposed architecture is designed to verify the effectiveness of the ALBRL algorithm

The rest of this paper is organized as follows: Section 2 gives the related literature; Section 3 describes the general architecture of the network; Section 4 establishes the load-balancing optimization model; Section 5 describes the ALBRL algorithm; Section 6 verifies the algorithm through simulation experiments; and finally, our conclusion and future directions for work given in Section 7.

2. Related Research

The current network environment is characterized by its complexity, colossal data traffic, and diversified service requirements. A traffic scheduling strategy based on a traditional network cannot meet the needs of the existing network environment and cannot handle increasingly complex tasks. The advent of SDN with programmability has led an increasing number of researchers to combine machine learning algorithms with SDN to optimize network routing. There are many solutions to the problem of optimizing TE throughput in SDN networks using traditional methods. For example, the authors in [5–7] use traditional methods such as the OSPF algorithm, the EMCP algorithm, and ant-colony optimization algorithms to optimize the network. However, traditional machine learning algorithms cannot grasp the global network information from multiple dimensions and often cause network congestion.

The deep learning (DL) method can optimize network routing in multiple dimensions using historical experience. Some researchers have studied network traffic prediction methods for network load balancing and bandwidth optimization. Azzouni and Pujolle [8] and Novaes et al. [9] used the long- and short-term memory (LSTM) algorithm to predict traffic forwarding data between network hosts. However, the number of hosts in a virtual network is often much larger than the number of switches. Therefore, this method has a limited effect in optimizing the forwarding strategy of the switches. The authors in [10] use a gated recurrent unit neural network (GRU-NN) to achieve better accuracy in predicting network traffic. The authors in [11, 12] propose a routing strategy that employs the convolutional neural networks (CNN) to select routing paths according to the online network traffic. The authors in [13] propose an active controller selection algorithm to schedule packets between the switch and the controller. The authors in [14] use the predicted link-state value of the LSTM algorithm to optimize network load balancing. These methods based on supervised learning require a large number of data sets for training and face the problem of slow decision-making in dynamic network scenarios.

In this paper, we use RL-based methods to consider the optimization of TE throughput using historical experience. Compared with supervised learning, reinforcement learning is an online learning that can change its actions to obtain optimal rewards through continuous exploration, learning, and trial. The authors in [16] propose a Q-learning and

SDN intelligent routing (RSIR) algorithm. The authors in [17] propose a routing-optimization strategy based on deep RL (DRL-R). The authors in [18] propose a multicontroller route-planning algorithm based on DQN. These three methods show a better improvement of the network. Q-learning can generate paths between node pairs, which avoids relying on switch link information in traditional decision-making. However, Q-learning methods require a lot of time to train the Q-table between each host pair in large-scale networks. Therefore, they have significant requirements for the controller's performance. Chen et al. [19] propose an algorithm based on dueling double DQN to solve the TE problem. Compared with the OSPF and least load (LL) routing algorithms, it achieved higher performance in well-known network topologies such as fat-tree and NSFNet. However, the problem of its convergence rate is not considered. The authors in [20] propose a routing-optimization algorithm based on the PPO model. Because of the adoption of the strategy method, the prediction of routing strategy in PPO is more accurate than Q-Learning. However, because the PPO algorithm is an on-policy algorithm, the sample utilization rate of the model in training is low. In order to address the problems of PPO and DQN algorithms, some researchers have used the DDPG algorithm to optimize the SDN network. DDPG is a model-free, deterministic-strategy, gradient algorithm based on an actor-critical principle [21]. It uses the offline learning method and Q-network for training and takes samples from the replay buffer to minimize correlation between samples. The authors in [22] adapt the DDPG algorithm to find the optimal scheduling scheme for flows. The authors in [23, 24] present a QoS optimization algorithm based on DDPG that ultimately improves the load-balancing degree and throughput rate to ensure delay and packet-loss rate. The authors in [25] propose an intelligent traffic-sampling system based on DDPG that can maintain the load balance of multiple traffic analyzers. The authors in [26] propose an intelligence-driven experiential network architecture for automatic routing (EARS) to optimize a network intelligently. EARS adapts the DDPG algorithm to select the routing paths of elephant and mouse flow through different action strategies and obtain near-optimal routing decisions. The authors in [27] employ DQN and DDPG to build DRL-based routing, which has better performance than OSPF. The authors in [28] optimize DDPG and proposed a reliable and time-sensitive DDPG algorithm to find the optimal path of network stream and solve the problem of continuity. However, DDPG is usually not robust enough for hyperparameters and other fine tuning. One of the main reasons for failure is that the Q-function overestimates Q-values, resulting in the collapse of the policy due to the use of errors in the Q-function. TD3 uses double DQN to solve the overestimating action Q-value that is critical in DDPG [29]. The authors in [30, 31] use TD3 and pinning control theory to generate routing policies automatically, thus ensuring that the routing policy can dynamically adapt to changes in network traffic. Although the TD3 algorithm is better than DDPG in network performance, it also increases the complexity of the routing algorithm and the burden on the controller, which leads to sizeable computational complexity when the network size increases. In addition, deterministic policy gradient algorithms such as DDPG and TD3 face the problems of a slow convergence rate and instability. This will heavily affect network performance before convergence.

As we know, an algorithm based on reinforcement learning uses online learning, while the network is running and needs to retrain when the network traffic changes significantly. Our paper proposes an automatic load-balancing architecture based on the improved DDPG algorithm. It uses the SumTree structure to improve the experience-pool sampling method of DDPG, which can speed up the algorithm convergence rate. The ALBRL we propose can make routing decisions in a continuous action space and has a perfect convergence. When the network traffic changes significantly, it can quickly converge, reducing the network performance fluctuation caused by the convergence delay of the reinforcement-learning model.

3. Network Architecture

In this paper, we describe our designed network architecture based on deep reinforcement learning, as shown in Figure 1. Each plane of the specific network architecture is described as follows:

(1) Forwarding plane. The forwarding plane is made up of programmable forwarding devices (SDN switches) and hosts. The forwarding devices connect to the control plane with southbound interfaces and receive the policies delivered by the control plane, such as the OpenFlow flow table

(2) Control plane. The control plane consists of SDN controllers that connect the forwarding plane and the management plane. The control plane obtains the topology of the forwarding plane and distributes control policies through the southbound interfaces. Meanwhile, it provides data network resources to the management plane through the northbound interfaces. The controller function is divided into the following four modules:

(i) The topology-building module. This module periodically obtains network information from the forwarding plane and builds the network topology.

(ii) The link information-acquisition module. This module periodically obtains the network status information of the data plane through the south interface. It sends the state s_t and reward r_t to the RL agent according to the acquiring information from the forwarding plane.

(iii) The packet_in handling module. This module handles the packet_in message uploaded by the switch.

(iv) The flow-entry distribution module. The controller updates flow entries based on the action a_t and the

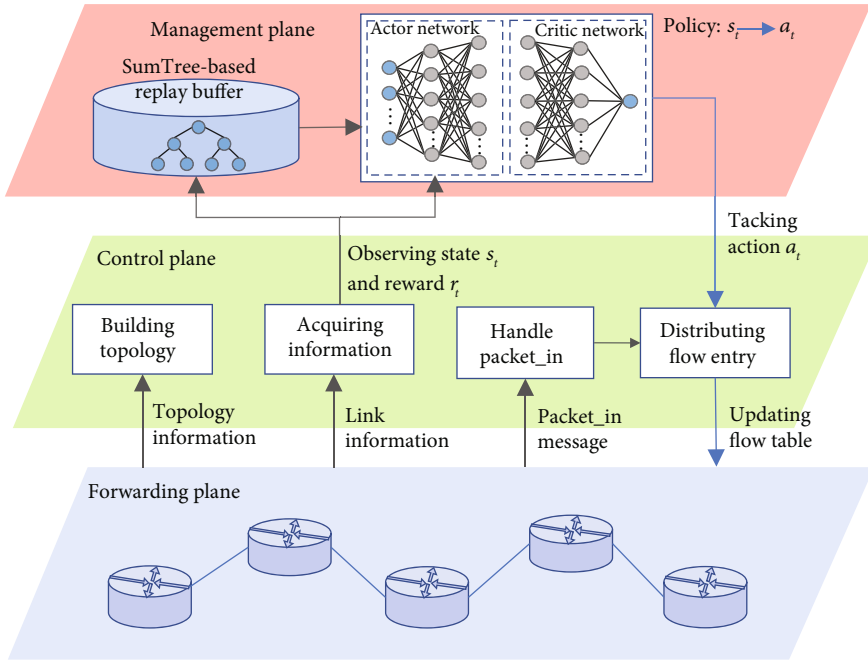


Figure 1: ALBRL network architecture. The architecture consists of the forwarding plane, the control plane, and the management plane.

packet_in message handling module and then actively issues flow entries.

(3) Management plane. The management plane runs a RL agent responsible for network load balancing. The agent adapts the DDPG algorithm, which uses SumTree-based replay buffers to find the near-optimal path between network hosts and generates link weights through the topology of the data obtained by the control plane and the network resource information of the forwarding plane. Then, the control plane updates flow entries based on link weight and send them to the forwarding plane.

4. Load-Balancing Optimization Model

This section describes the mathematical model for evaluating network load balancing in high-load traffic scenarios as the SDN network performance evaluation. We use $Graph = (V, E)$ to represent the undirected graph structure of the forwarding plane. We use M to denote the number of switches and N to denote the number of links. V is the set of all switches v, that is, $V = [v_1, v_2, \cdots, v_M]$. E is the set of all links e, that is, $E = [e_1, e_2, \cdots, e_N]$. A link e is defined by two connected forwarding nodes.

We use w_i to represent the weight value of the i-th link e_i, and we define W as the set of all link-weight values:

$$W = [w_1, w_2, w_3, \cdots, w_N]. \tag{1}$$

We use B to denote the set of link-bandwidth, as shown in Equation (2), where b_i represents the bandwidth of the link e_i:

$$B = [b_1, b_2, b_3, \cdots, b_N]. \tag{2}$$

We use U to describe the used-bandwidth set of all links, as shown in Equation (3), where u_i represents the used bandwidth of the link e_i:

$$U = [u_1, u_2, u_3, \cdots, u_N]. \tag{3}$$

It is critical to evaluate the performance of a network. This paper is concerned with the degree of load-balancing in the network. We sum the absolute value of the difference between the used bandwidth of each link and the average used bandwidth to describe the link-load-balancing factor φ:

$$\varphi = \sum_{i=1}^{N} |u_i - avg(U)|. \tag{4}$$

We define link utilization as the ratio of used bandwidth to link bandwidth, as shown in Equations (5) and (6), where r_i represents the link utilization rate of the link e_i.

$$R = [r_1, r_2, r_3, \cdots, r_N], \tag{5}$$

$$r_i = \frac{u_i}{b_i}. \tag{6}$$

In SDN, the load-balancing-rate factor ρ shows the relative utilization of the network links, which is calculated below:

$$\rho = \sum_{i=1}^{N} |r_i - avg(R)|. \tag{7}$$

Due to link bottlenecks in the forwarding path, forwarding packets of the same length may differ in network throughput [32]. Therefore, we define the network throughput as TP, shown in Equation (8), where H_i is the amount of traffic received by host i in the data plane and L_H is the number of hosts. The network throughput effectively measures the carrying capacity of the data plane:

$$TP = \sum_{i=1}^{L_H} H_i. \tag{8}$$

In this paper, our optimization objective is to maximize the network carrying capacity and minimize the link-load-balancing factor φ, as shown in Equation (9). Equation (10) represents the limiting condition for obtaining the optimization objective:

$$\max\ (TP - \varphi), \tag{9}$$

$$s.t. \begin{cases} 0 < w_i < 1 \\ u_i < b_i \end{cases}. \tag{10}$$

5. Automatic Load-Balancing Algorithm Based on Reinforcement Learning

This section describes the ALBRL routing-optimization algorithm and details the settings of the ALBRL parameters.

5.1. DDPG Algorithm. The RL model includes mainly the following components: the system environment, the RL agent, the reward, and the policy action. The RL agent continuously explores the surrounding environment through the learning scenario, a series of actions taken to transition from the initial state to the target state. Each step includes choosing and executing an action, changing the state, and acquiring a reward.

DDPG is a DRL algorithm based on deterministic policy gradient (DPG). It combines DQN and DPG into an actor-critic (AC) framework to realize efficient and stable continuous action control. During training, DDPG updates the policy through the off-policy method, selects the deterministic action, and maximizes the reward. DDPG stores the interactive information with the environment in the replay buffer and obtains the learning samples of the neural network by randomly sampling them through the empirical-playback mechanism.

The DDPG algorithm consists of the actor network, the critic network, and empirical-playback mechanism, as shown in Figure 2. The function of each component of DDPG is as follows:

(1) Actor network. The actor network adopts the deterministic policy function to output the policy action. It contains two neural networks: an actor online network for training and learning and an actor target network for preventing the correlation of training data. The actor online network selects the action a_t according to the state s_t and policy function μ and then interacts with the environment to get s_{t+1} and r_t. It uses the policy gradient to update the actor online network parameter θ^{μ}:

$$\nabla_{\theta^{\mu}} J = \frac{1}{N} \sum_i \nabla_a Q\left(s, a; \theta^Q\right)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s; \theta^{\mu})\Big|_{s_i}, \tag{11}$$

where N is the number of mini-batch samples.

The actor target network generates the optimal action a_{t+1} of the next state s_{t+1} randomly sampled from the experience pool. The network parameters $\theta^{\mu'}$ are periodically updated from θ^{μ}:

$$\theta^{\mu'} \longleftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \tag{12}$$

where the target update rate τ is a hyperparameter that is much less than 1.

(2) Critic network. The critic network adopts Deep Q-learning function to estimate the policy action. It contains two neural networks: a critic online network and a critic target network. The critic online network is responsible for calculating the current Q value $Q(s_i, a_i; \theta^Q)$. It uses loss function L to update the critic online network parameter θ^Q, as shown in Equation (13), where y_i is the target Q value, as shown in Equation (14):

$$L = \frac{1}{N} \sum_i \left(y_i - Q\left(s_i, a_i; \theta^Q\right)\right)^2, \tag{13}$$

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1}; \theta^{\mu'}\right); \theta^{Q'}\right). \tag{14}$$

In Equation (14), where γ is the reward discount factor, $\gamma \in [0, 1]$.

The critic target network uses Equation (14) to calculate the target Q value $Q'(s_{i+1}, a_{t+1}; \theta^{Q'})$. The network parameters $\theta^{Q'}$ are periodically updated from θ^Q:

$$\theta^{Q'} \longleftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{15}$$

where the target update rate τ is a hyperparameter that is much less than 1.

(3) Empirical-playback mechanism. The actor and critic networks process the transferred samples obtained during model training through the empirical-playback mechanism. They randomly extract a mini-batch of samples from the replay buffer each time for training the model. This paper adapts the priority empirical-playback mechanism based on the SumTree storage model to optimize the empirical-playback mechanism (see Section 5.2 for more details).

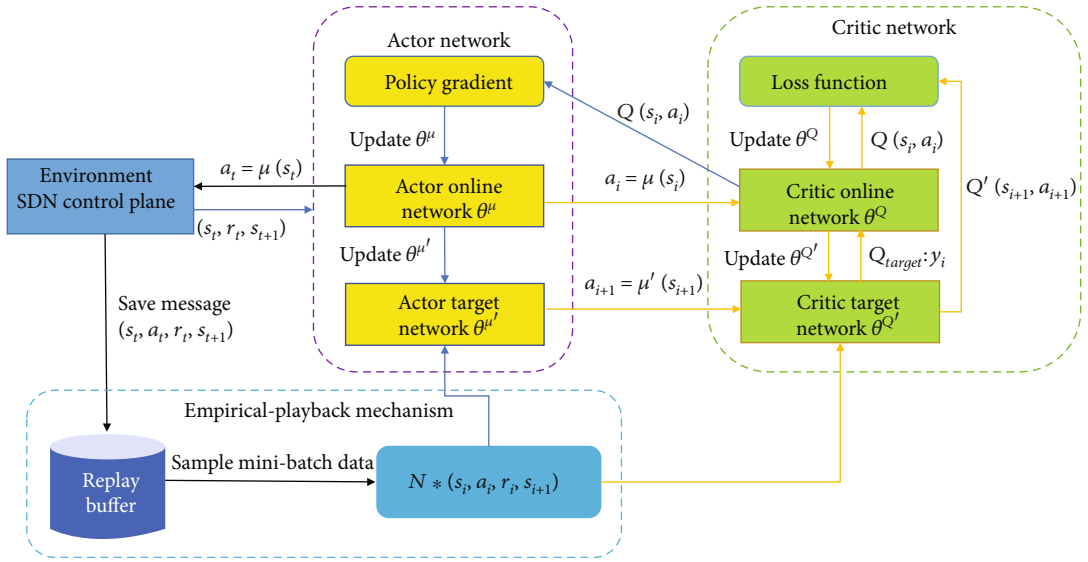


Figure 2: The DDPG framework. The framework consists of the actor network, the critic network, and empirical-playback mechanism.

5.2. The Improved DDPG Based on SumTree. In a real network scenario, the action space of the RL agent grows synchronously, increasing RL training time with the improvement in network scale. This puts the SDN network in the trial state of low performance for a long time before the model converges. Therefore, the learning convergence rate of the model is the key figure for measuring the model's performance. In addition, the network traffic state is random and diverse. When the network traffic fluctuates significantly, a model with good convergence performance can quickly restore the network to a stable convergence state.

A traditional DDPG algorithm processes the transfer samples obtained in model training through an empirical-playback mechanism. This mechanism randomly extracts a mini-batch of samples from the sample pool each time for training the model, causing the agent to be unable to distinguish the critical difference between different transferred samples. This leads to problems such as the low utilization of valuable samples, the coverage of some essential samples, and the reuse of samples. Therefore, it dramatically reduces the convergence rate of the model.

In our paper, we adapt the priority empirical-playback mechanism based on the SumTree storage model to optimize the empirical playback of DDPG. We use priority sampling to replace the original random sample, which increases the probability of significant reward experience extracting to speed up the agent learning the optimal strategy and improving the convergence rate of the model. The sample priority value p_i in the experience pool is mainly based on the TD-error δ_i of the value network and the network load-balancing-rate factor ρ:

$$p_i = |\delta_i| + \rho, \tag{16}$$

where ρ is shown in Equation (7), and δ_i is shown below:

$$\delta_i = r_i + \gamma Q'\left(s_{i+1}, a_{i+1}; \theta^{Q'}\right) - Q\left(s, a; \theta^{Q}\right). \tag{17}$$

The network load-balancing-rate factor ρ shows the relative utilization of network link. The TD-error δ_i indicates how far the value is from its next-step bootstrap estimate [33]. The larger the TD-error δ_i and the factor ρ, the higher the priority. The priority sample has a larger rising space of prediction accuracy, enhancing the model's effectiveness.

In the ALBRL we propose, the experience pool data is stored in the transition, including current state s_t, action a_t, reward value r_t, and updated state s_{t+1}. The transition uses the priority value p_i as the corresponding index. We store the priority value p_i in the SumTree leaf node. The parent node above the leaf node stores the sum of the priority values of the left and right child nodes, and the root node stores the sum of all leaf nodes, as shown in Figure 3. During data sampling, the SumTree model divides the sum of priority value p_i by the number of samples to obtain the number of intervals. Then, the model randomly selects a number in each interval. It starts from the root node of SumTree and searches downward according to specific rules. Finally, the model obtains the corresponding sample data through the priority value p_i conveyed by the previous search.

The data extraction steps of SumTree are as follows:

(1) Uniformly sample data in the root node, assuming that it is p

(2) Take the root node as the parent node, and traverse its child nodes

(3) If the left child node is more significant than p, take it as the parent node, and record its child nodes

(4) If the value of the left child node is less than p, subtract the value of the left child node from p, select the right child node as the parent node, and traverse its children

(5) Until the leaf node is traversed, the value of the leaf node is the priority. The corresponding transition can be found in the value stored in the leaf node

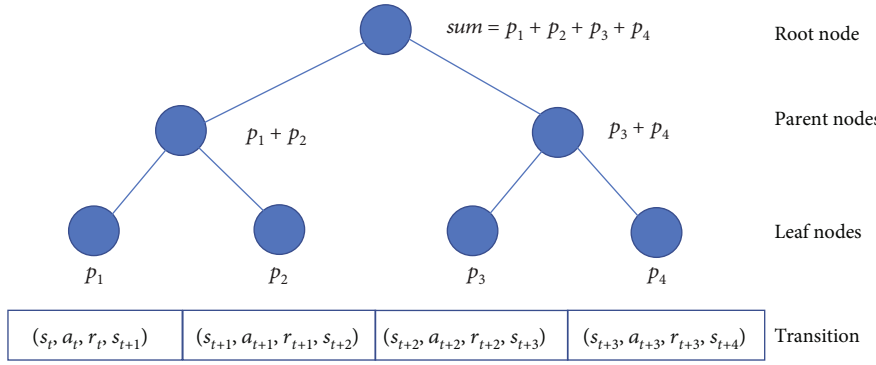


Figure 3: SumTree storage model.

According to the above data extraction method, the greater the priority value p_i, the greater is the probability of extracted data. The sampling probability and the importance-sampling weight are shown below:

$$P(i) = \frac{p_i}{\sum_j p_j}, \tag{18}$$

$$\omega_i = (N * P(i))^{-1} / \max_j (\omega_j). \tag{19}$$

We add the importance-sampling weight ω_i to update the actor online network parameter θ^μ:

$$\nabla_{\theta^\mu} J = \frac{1}{N} \sum_i \omega_i \left(\nabla_a Q\left(s, a; \theta^Q\right) \Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu\left(s; \theta^\mu\right) \Big|_{s_i} \right). \tag{20}$$

Regarding comparing the complexity of ALBRL and DDPG algorithms, since the SumTree-DDPG algorithm adopted by ALBRL is consistent with the DDPG algorithm in terms of the main body of the model, we focus on the complexity of the experience pool sampling. In the sampling process, the original DDPG uses unified random sampling to extract samples from the experience pool sequence, while ALBRL uses priority sampling based on the SumTree storage model. The difference between priority sampling and random sampling is the generation of the sampling sequence number. The random sampling method directly generates random numbers, and the time complexity is $O(1)$. In contrast, the priority sampling method we proposed stores the priority in a full binary tree structure, and we first need to find the leaf node as the sampling sequence number when sampling. Compared with the random sampling method, the time complexity of the priority sampling process is $O(logN)$, where N is the size of replay buffer [33]. In terms of space complexity, ALBRL needs to use one more binary tree structure, and the number of leaf nodes is equal to the size of the traditional experience pool. Therefore, the spatial complexity of ALBRL sampling is the same as DDPG, which is still $O(N)$. Compared with DDPG, the time complexity of ALBRL increases by $O(logN)$. Still, the ALBRL algorithm performance optimization is considerable, and it can reach the convergence state in less time.

5.3. The Network Load-Balancing Algorithm Based on ALBRL. The specific design of the state, action, and reward of the ALBRL model we propose in SDN is as follows:

(1) Reward. The reward r_t evaluates the pros and cons of traffic scheduling for the revenue obtained from the previous action. In the mathematical model described in Section 4, the optimization objective of the SDN load-balancing model is to maximize the network carrying capacity and minimize the link-load-balancing factor φ. We define the reward r_t as the optimization goal:

$$r_t = TP - \varphi. \tag{21}$$

(2) State. The state s_t is the network status information obtained from the environment. In our model, each state corresponds to a traffic matrix (flow request information in the network). We define $s_t = [D_t]$, where D_t represents the flow request matrix in the network at time t

(3) Action. The action a_t is the set of all link-weight values (as shown in Equation (1)) generated by the reinforcement-learning agent at time t according to the state s_t in the actor network.

The ALBRL agent interacts with the SDN environment through the abovementioned variables (state, action, and reward). First, the data plane's network status and performance indicators collected by the SDN controller can provide the state s_t for the ALBRL agent. Subsequently, ALBRL determines a set of link weights $a_t = [w_1, w_2, \cdots, w_N]$ according to the state s_t. The SDN controller uses the Dijkstra algorithm to generate a new flow path and to update the flow table according to the link weight. After the flow-table update, the reward r_t and the next state s_{t+1} are acquired through subsequent network measurement, optimizing

Input:
Reward discount factor γ, target update rate τ, target network parameter update frequency C, the number of mini-batch samples N, the number of iterations T.
Randomly initialize the actor-online-network parameter θ^{μ} and the critic-online-network parameter θ^{Q};
Initialize the actor target network μ' and the critic target network Q' with $\theta^{\mu'} \longleftarrow \theta^{\mu}$ and $\theta^{Q'} \longleftarrow \theta^{Q}$;
Initialize replay buffer B;
Initialize the data buffer of SumTree S, set the priority p_i of all leaf nodes to 0;
Initialize a random process $\mathcal{N}$ for action exploration;
Initialize state s_1 with n the collected information from the SDN controller and acquire its feature vector $\mu(s_1; \theta^{\mu})$;

1) **for** t =1, T **do**
2) Select the action weight $a_t = \mu(s_t) + \mathcal{N}$ according to the state s_t in the actor online network;
3) Deploy a_t on SDN controller;
4) Recalculate paths and issue the flow table;
5) Get the reward r_t, the new state s_{t+1}, and the terminate flag is_end_t from the SDN controller;
6) Store (s_t, a_t, r_t, s_{t+1}) in B;
7) Calculate the sample priority value: $p_i = |\delta_i| + \rho$;
8) Update all nodes of S;
8) Use the SumTree model to extract N samples from B;
10) Calculate the importance-sampling weight: $\omega_i = (N * P(i))^{-1} / \max_j(\omega_j)$;
11) Calculate the target Q value:
12) $$y_t = \begin{cases} r_t & \text{is\_}end_t \text{ is } true \\ r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}; \theta^{\mu'}); \theta^{Q'}) & \text{is\_}end_t \text{ is } false. \end{cases}$$
13) Use loss function L to update the critic online network parameter θ^{Q}:
14) $L = 1/N \sum_i (y_i - Q(s_i, a_i; \theta^{Q}))^2$
15) Use the policy gradient to update the actor online network parameter θ^{μ}:
16) $\nabla_{\theta^{\mu}} J = 1/N \sum_i \omega_i (\nabla_a Q(s, a; \theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s; \theta^{\mu})|_{s_i})$
17) **if** $t\%C = 0$ **then**
18) update the Critic-target-network and Actor-target-network parameters:
$\theta^{\mu'} \longleftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
$\theta^{Q'} \longleftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
19) **end if**
20) **end for**

Algorithm 1: ALBRL training algorithm.

network performance through iteration. We describe the ALBRL-training algorithm as shown in Algorithm 1.

6. Experiments and Analysis

In our experiments, the network simulation environments are the *Mininet* simulation, *Ryu* controller, and *OpenFlow 1.3* protocol. The computer is equipped with an AMD Ryzen 7 2400G CPU with a base frequency of 3.0 GHz, a 500GB SSD, and 32 GB RAM. In this paper, *Pytorch* was used to implement reinforcement-learning algorithms for SDN.

6.1. Experimental Setup. This experiment uses *iperf* to send packets and takes active flow-table distribution for issuing flow tables. The controller periodically computes the packet-forwarding policy and then sends it down to the corresponding switch through the southbound interface. Compared with passive flow-table distribution, active flow-table distribution can avoid the problem of the overloading or even the crashing of the controller due to a mismatch between the new path and the old flow table.

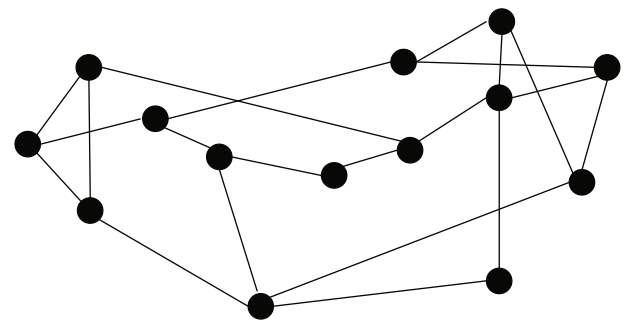
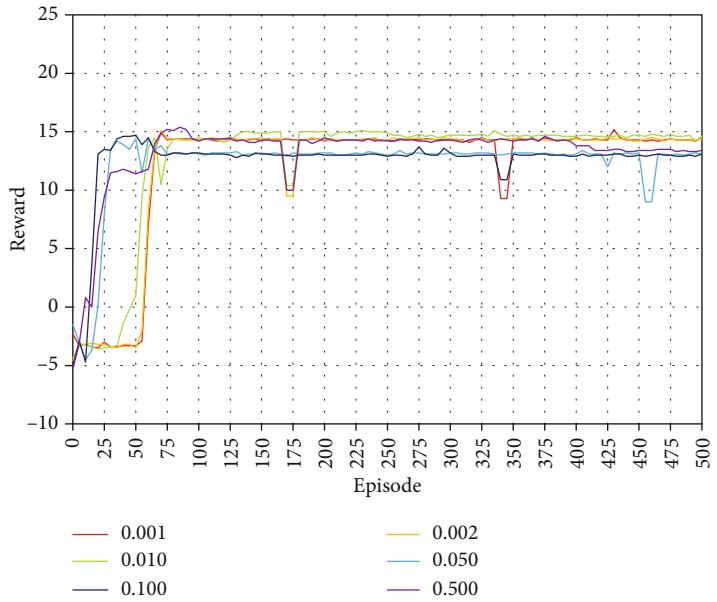
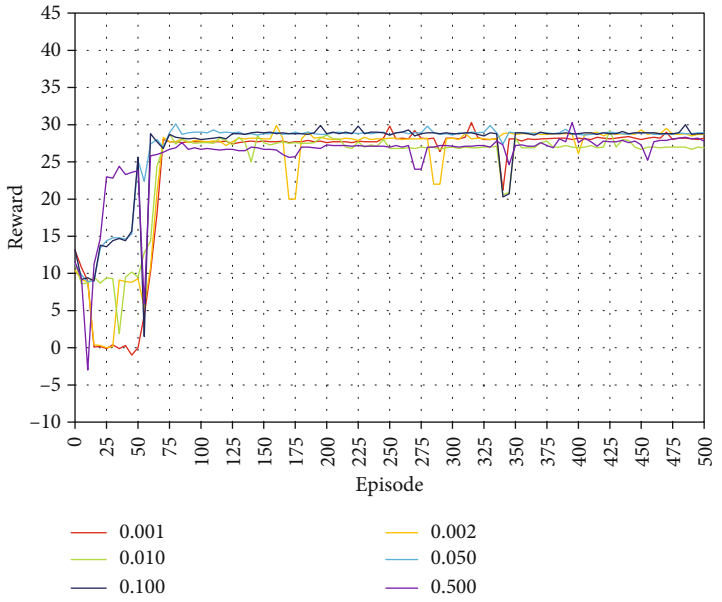


Figure 4: NSFNET topology. It consists of 14 switches with 21 links, all with a link bandwidth of 10 Mbps.

In our experiments, we used the National Science Foundation Network (NSFNet) as the experimental network topology, as shown in Figure 4. The topology consists of 14 switches with 21 links, all with a link bandwidth of 10 Mbps. Each switch connects to a host machine. This paper sets four different levels of traffic intensity for the NSFNet to simulate a realistic network situation, accounting for



(a)



(b)

Figure 5: Continued.

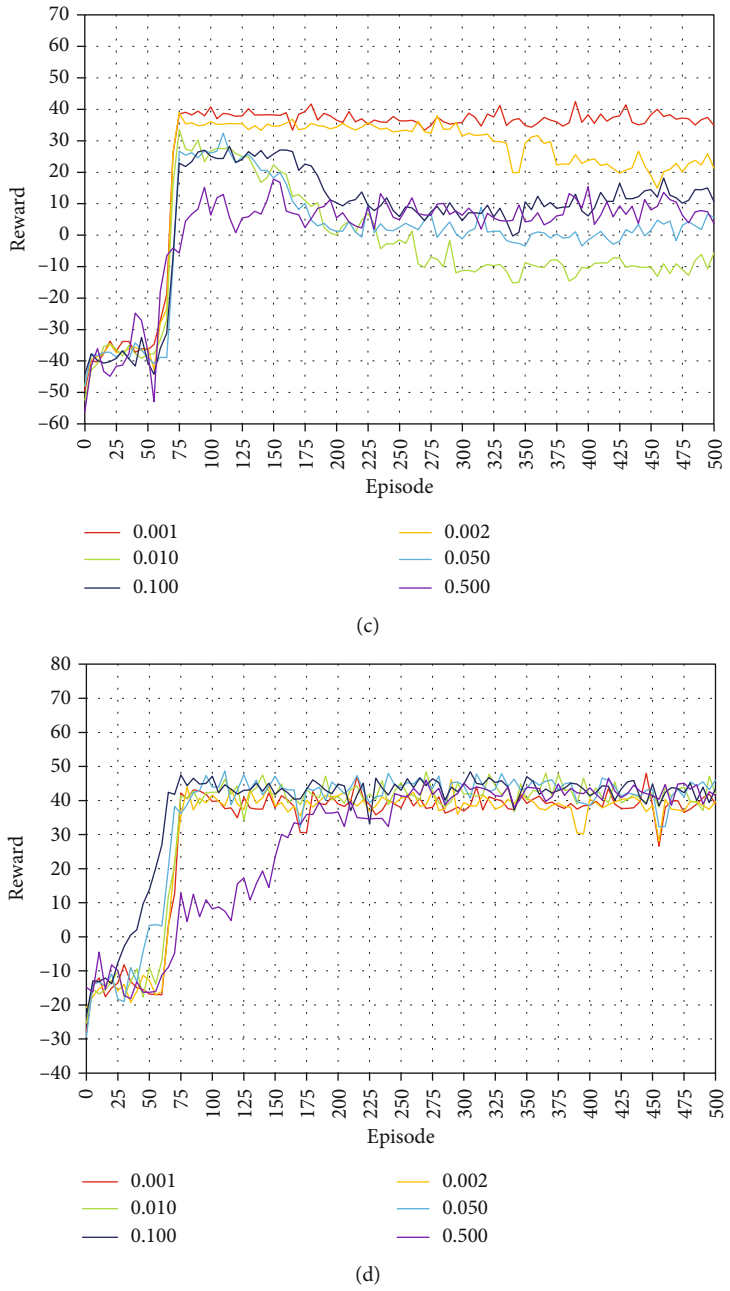


Figure 5: Effect of learning parameters: (a) traffic intensity is 25%; (b) traffic intensity is 50%; (c) traffic intensity is 75%; and (d) traffic intensity is 100%.

25%, 50%, 75%, and 100% of the entire network bandwidth, respectively. We use the gravity model [34] separately for each traffic intensity to randomly generate a traffic matrix for training.

In this section, we compare the convergence rate and network performance of ALBRL with the following algorithms:

(1) OSPF [5]: OSPF routes all flow requests singularly to the shortest path

(2) RSIR [16]: RSIR adapts the Q-learning algorithm to optimize the network in three dimensions: throughput, delay, and packet-loss rate. We only use throughput for optimization

(3) EARS [26]: EARS adapts the DDPG algorithm as the automatic routing decision algorithm to select the forwarding paths through action strategy and obtain near-optimal routing decisions

6.2. Parameter Setting. As an essential parameter in deep learning, the appropriate learning rate can quickly make the objective function converge to a local minimum. In the network, the actor learning rate is not involved in the RL

agent training process because the traffic is random, and there is no next state after each training. We change the critic-learning rate to observe the ABLRL convergence effect and determine the optimal value of the DRL critic-learning rate by the convergence rate and the maximum reward. The critic-learning rate was set to 0.001, 0.002, 0.01, 0.05, 0.1, and 0.5. The model's performance in the training environment was tested every five training steps in this experiment. The training results are shown in Figure 5.

Figure 5 shows that the reward curves obtained for different critic-learning rates are different. Moreover, the optimal critic-learning rate varies with varying intensities of traffic. The reward curve is best at a critic-learning rate of 0.1 when the traffic intensity is 25%. The highest reward value achieved is 15, as shown in Figure 5(a). When the traffic intensity is 50%, the reward curve is optimal at a critic-learning rate of 0.05, and the highest reward value achieved is 29, as shown in Figure 5(b). Although the reward curve at a critic-learning rate of 0.5 in this traffic intensity performed better than at 0.1 in the first 50 episodes, the reward convergence value of 27 was not as good as the reward convergence value at a critic-learning rate of 0.1. When the traffic intensity is 75%, the routing algorithm adjusts the link-weight values more frequently because some links have reached saturation. The reward convergence values are higher at lower critic-learning rates. The best reward curve at this flow intensity is with a critic-learning rate of 0.001, and the highest reward value achieved was 40, as shown in Figure 5(c). When the traffic intensity is 100%, most links are saturated, and so the maximum reward value reached by different critic-learning rates is the same. With a critic-learning rate of 0.1, the reward curve converges fastest. Thus, the best critic-learning rate is 0.1, and the highest reward value achieved is 45, as shown in Figure 5(d).

Based on the above results, in our experiments, the critic-learning rate of the parameter critic of ALBRL was set at a traffic intensity of 25%, 50%, 75%, and 100% with 0.1, 0.05, 0.001, and 0.1, respectively. Under these parameters, ALBRL can reach the convergence rate the fastest and most effectively.

Other hyperparameters set in the experiment are shown in Table 1.

Table 1: ALBRL training hyperparameters.

Hyperparameters	Value
Actor learning rate	0.001
Optimizer	Adam
Target update rate τ	0.01
Target network parameter update frequency C	1
Number of iterations T	500
Replay buffer B	32
Batch size N	8
Reward discount factor γ	0.99
Exploration noise	$\mathcal{N}(0,0.1)$

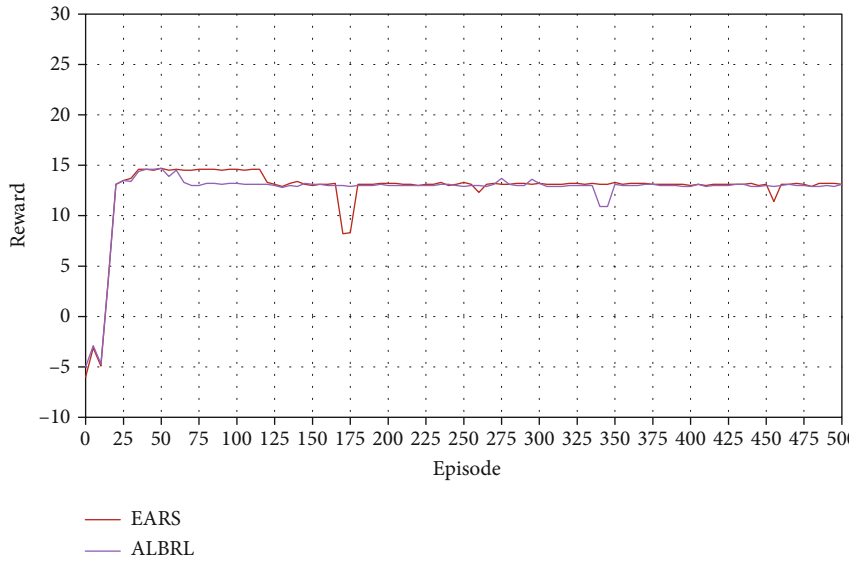
6.3. Algorithm Convergence Performance Comparison. After determining the critical learning rate, we compared the convergence performance of ALBRL and EARS algorithms under different traffic intensities. We trained 500 episodes for each traffic intensity and tested the model's performance in the training environment every five training steps. The experimental results are shown in Figure 6.

The reward value is used in reinforcement learning to measure the outcome of an action. The larger the reward value, the better the network performance is. When the traffic intensity is 25%, the ALBRL reward convergence value is the same as that of EARS, but the convergence rate of EARS is slightly faster, and the performance is somewhat better than that of ALBRL, as shown in Figure 6(a). When the traffic intensity is 50%, ALBRL converges at the same speed as EARS, but the ALBRL reward convergence value is higher than that of EARS, as shown in Figure 6(b). When the traffic intensity is 75%, the ALBRL reward convergence value is close to that of EARS, but the convergence rate is higher than that of EARS, which is improved by 47%, as shown in Figure 6(c). When the traffic intensity is 100%, the convergence value of the ALBRL reward is close to that of EARS, but the convergence rate is significantly higher than that of EARS, which is increased by nearly 70%. In addition, due to dynamic changes in network traffic, the reward value of EARS will fluctuate after convergence, while the reward value of ALBRL is more stable.

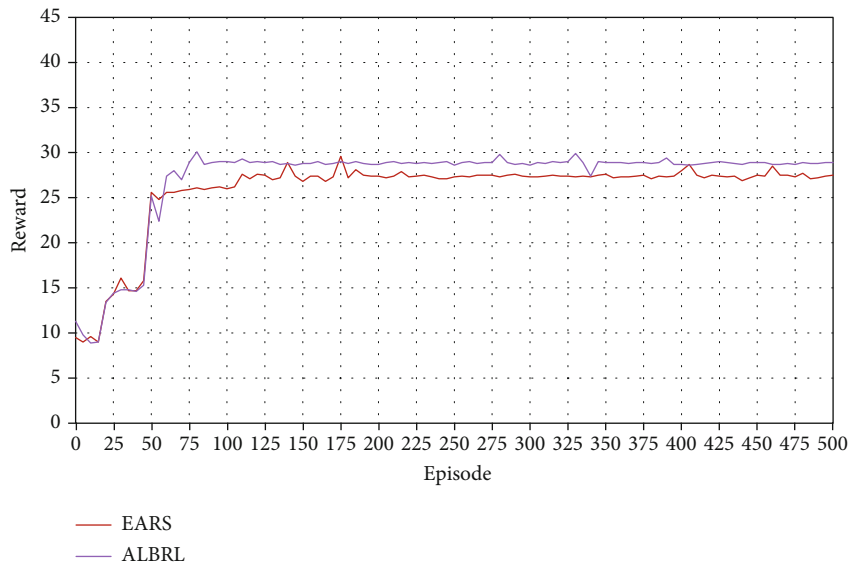
The results in Figure 6 show that the performance of ALBRL is close to that of EARS in a low-load network (traffic intensity less than 50%). Moreover, ALBRL can reach a convergence state faster than EARS in a high-load network (traffic intensity more than 50%), and after convergence, the reward value is more stable. Under low traffic intensity, the algorithm can reach the convergence state with more minor training episodes because of the lower traffic in the network. In the early stage of network operation, the priority value of the stored samples is close, and so the role of the priority-based empirical-playback mechanism is not apparent. Under high traffic intensity, the algorithm needs more training episodes to reach the convergence state. When the episode value is greater than 50, the priority values of stored samples begin to differ significantly. ALBRL causes the model to reach the convergence state quickly through the advantages of the priority playback mechanism in a high-load network.

6.4. Load-Balancing Performance Comparison. Throughput is an important indicator to measure the performance of load balancing. This paper tests the performance differences of four algorithms—ALBRL, EARS, RSIR, and OSPF—under different traffic intensities.

The throughput result of 25% traffic intensity is shown in Figure 7(a). The throughput of OSPF fluctuates significantly due to the dynamic change of network traffic. The throughput of RSIR is more stable than OSPF, but the values are lower than 40 Mbps. The performance of ALBRL and EARS is close, and the throughput value after convergence is 47 Mbps, which is better than that of OSPF and RSIR.

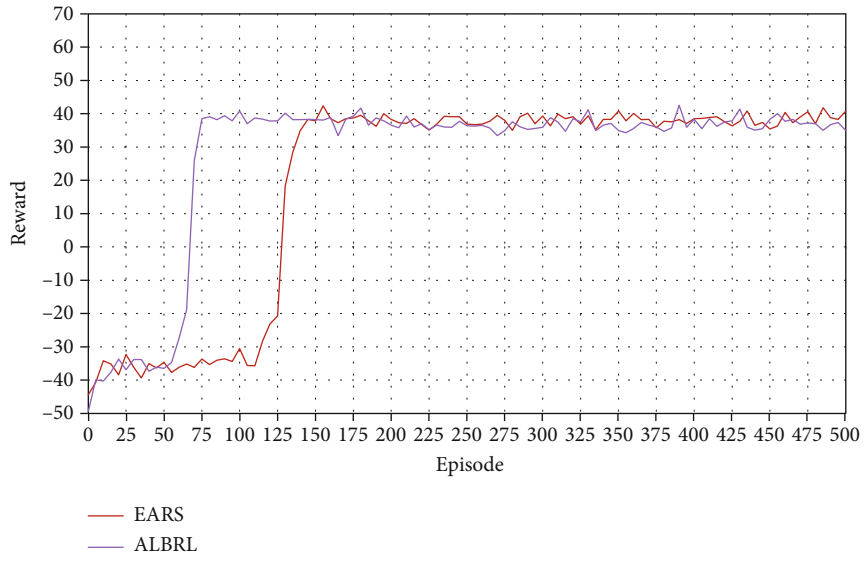


(a)

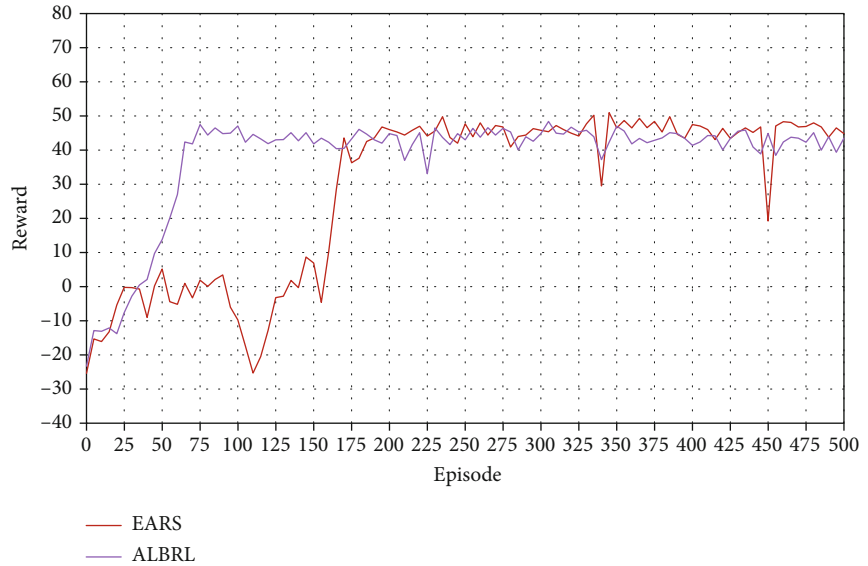


(b)

Figure 6: Continued.



(c)



(d)

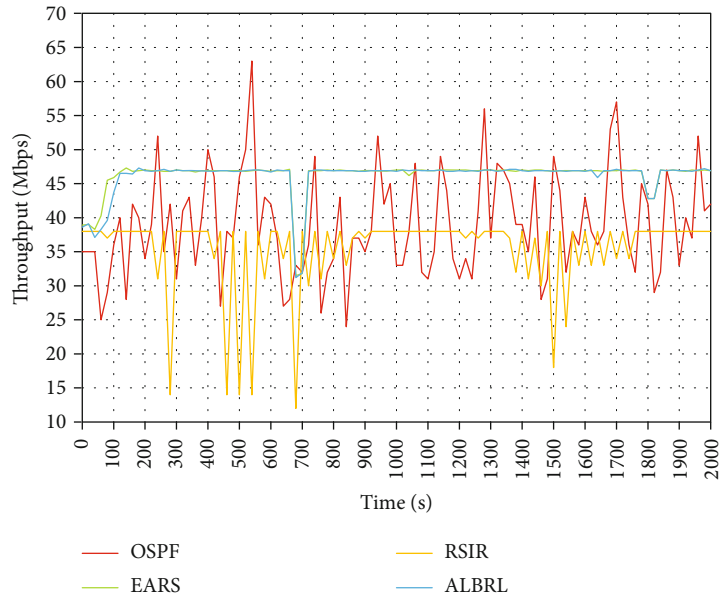
Figure 6: Convergence performance comparison: (a) traffic intensity is 25%; (b) traffic intensity is 50%; (c) traffic intensity is 75%; and (d) traffic intensity is 100%.

The throughput result of 50% traffic intensity is shown in Figure 7(b). The convergence rate of ALBRL is slightly higher than that of EARS, but the throughput during convergence is 87.5Mbps, which is higher than that of EARS. The throughput of OSPF and RSIR fluctuates wildly and is lower than that of ALBRL and EARS when they converge.

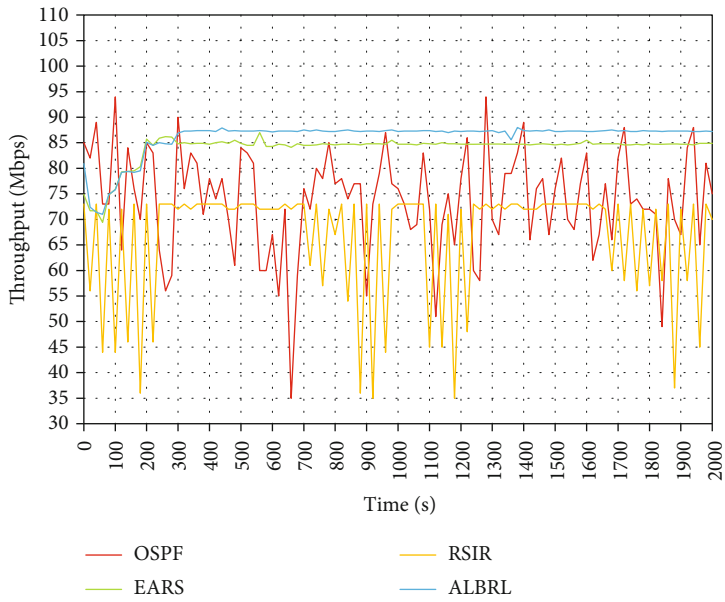
The throughput result of 75% traffic intensity is shown in Figure 7(c). The convergence throughput value of ALBRL is about 109 Mbps, which is close to EARS, but the convergence time is reduced by about 47%. The throughput of OSPF and RSIR are both lower than that of ALBRL and EARS when they converge.

The throughput result of 100% traffic intensity is shown in Figure 7(d). The convergence throughput of ALBRL is about 121 Mbps, which is lower than that of EARS, but the convergence time is reduced by about 36%. At the same time, there are two large fluctuations after the convergence of the EARS algorithm. The throughput of OSPF and RSIR is lower than that of ALBRL and EARS when they converge. Further, in the RSIR algorithm's later stage, severe congestion occurs on multiple links, resulting in a high packet-loss rate and a low total throughput value.

In a low-load network scenario, the network traffic does not reach the network forwarding bottleneck, and there is less congestion due to excessive traffic. Therefore, both ALBRL and the original DDPG algorithm can quickly reach the convergence state. The performance of ALBRL under low traffic intensity is close to EARS, but is better than OSPF and RSIR, as shown in Figures 7(a) and 7(b). However, when the network traffic scale exceeds the network forwarding



(a)



(b)

Figure 7: Continued.

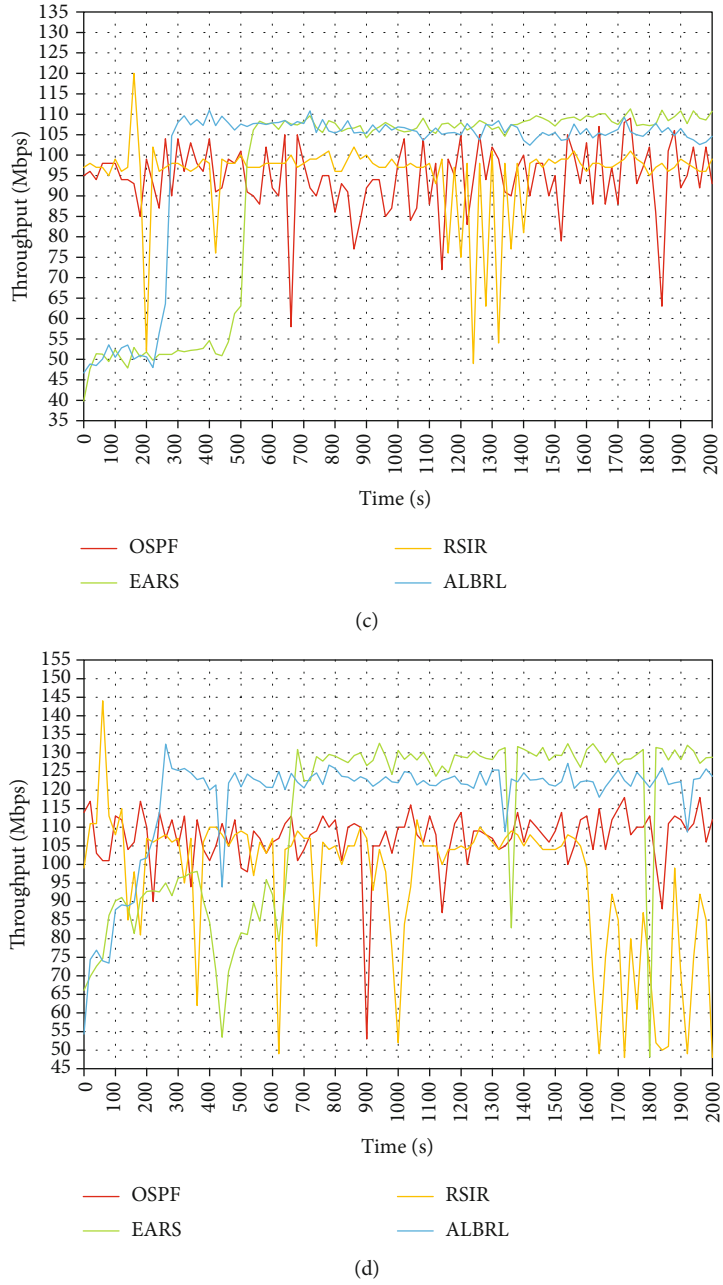


Figure 7: Overall throughput performance comparison: (a) traffic intensity is 25%; (b) traffic intensity is 50%; (c) traffic intensity is 75%; and (d) traffic intensity is 100%.

bottleneck in terms of reward convergence value since ALBRL can reuse those more valuable data, it can be closer to the optimal strategy than the other three algorithms. The experiments in Figure 7 show that the ALBRL algorithm can reach the convergence state faster than the other three algorithms, it is more stable, and it does not experience large fluctuations.

Our paper compares the four algorithms' average throughput rates with different traffic intensities in Figure 7, and the experimental results are shown in Figure 8.

Compared with EARS, ALBRL shows similar performance under low traffic intensity (less than 50%), but it is better than EARS under high traffic intensity (more than 50%). The throughput of ALBRL is 4~6% higher than that of EARS under high traffic intensity. Both ALBRL and EARS are based on the DDPG algorithm. Different from EARS, ALBRL uses the SumTree binary tree structure to optimize the extraction method of the experience pool of the traditional DDPG algorithm. Under low traffic intensity, the algorithm can reach the convergence state with more minor training episodes, and so the role of the priority-based, empirical-playback mechanism is not apparent. Under high traffic intensity, the algorithm needs more training episodes to reach the convergence state. When the episode value is greater than 50, the ALBRL extracts more meaningful

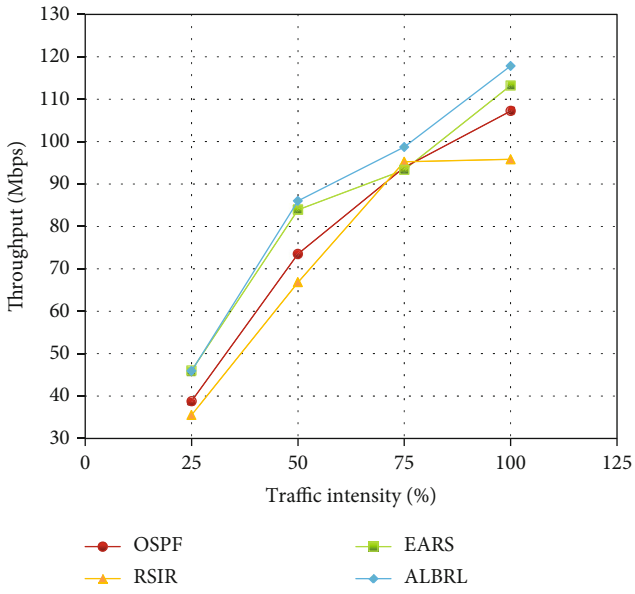


Figure 8: Model effective comparison.

experience for the network update with greater probability. Therefore, ALBRL can achieve convergence faster than EARS at a high traffic intensity.

Compared with RSIR, ALBRL is better in all traffic intensities. The throughput of ALBRL is 5~18% higher than that of RSIR. Each time the RSIR algorithm is trained, the Q table between each host pair needs to retrain for each instance of collected link information. However, the data between each host pair is irrelevant. Therefore, RSIR is likely to cause overload in the controller. At the same time, when RSIR outputs actions, due to the limitations of the value base, it can only output a single action when controlling routing, and it cannot realize fine control over the network as a whole. Thus, RSIR has multiple links with severe congestion in the later stages, resulting in high packet loss and large fluctuations in throughput. The ALBRL algorithm considers the entire network to create action vectors related to network routing for more refined control and thus for better optimization performance.

Compared with OSPF, the throughput of ALBRL is 10~14% higher than that of OSPF in all traffic intensities. OSPF is a static scheme based on fixed forwarding rules, which routes all flow requests singularly to the shortest path. Therefore, OSPF is prone to link congestion and cannot accommodate dynamic network environments.

Our experimental results show that ALBRL outperforms other algorithms under various traffic intensities. The ALBRL algorithm has a faster convergence rate and shows better performance and stability.

7. Conclusions and Future Work

This paper proposes an automatic load-balancing architecture based on an improved DDPG algorithm. The proposed ALBRL adapts the sampling method of updating the experience pool with the SumTree structure to improve the random extraction strategy of the empirical-playback mechanism in DDPG. It extracts more meaningful experiences for a network updating with greater probability, which can effectively improve the convergence rate. ALBRL can make routing decisions in a continuous action space and has a perfect convergence. The experimental comparison shows that the SDN routing algorithm based on ALBRL has a faster convergence rate and better performance and stability.

In the future, the specific measures on which we will focus are as follows: First, we will further enhance the convergence rate of deep reinforcement-learning algorithms by extending the existing single agent to multi-agent, and second, we will extend the network topology environment to test the load-balancing performance of the algorithm in large-scale networks.

Data Availability

The authors confirm that the data supporting the findings of this study are available within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No.61861013), the major program of Natural Science Foundation of Guangxi (No.2020GXNSFDA238001), the Natural Science Foundation of Guangxi (No.2018GXNSFAA281318) and the Guangxi Project to Improve the Scientific Research Basic Ability of Middle Aged and Young Teachers (No.2020KY05033).

References

[1] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Dynamic hybrid routing: achieve load balancing for changing traffic demands," in *IEEE 22nd International Symposium of Quality of Service (IWQoS 2014)*, pp. 105–110, Hong Kong, China, May 2014.

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software defined networks," *IEEE Network*, vol. 30, no. 3, pp. 52–58, 2016.

[3] R. A. Ammal, P. C. Sajimon, and S. S. Vinodchandra, "Termite inspired algorithm for traffic engineering in hybrid software defined networks," *PeerJ Computer Science*, vol. 6, article e283, 2020.

[4] T. Hartman, A. Hassidim, H. Kaplan, D. Raz, and M. Segalov, "How to split a flow?," in *Proceedings IEEE INFOCOM*, pp. 828–836, Orlando, USA, March, 2012.

[5] A. Ali Khan, M. Zafrullah, M. Hussain, and A. Ahmad, "Performance analysis of OSPF and hybrid networks," in *International Symposium on Wireless Systems and Networks (ISWSN 2017)*, Lahore, Pakistan, Nov. 2017.

[6] M. Chiesa, G. Kindler, and M. Schapira, "Traffic engineering with equal-cost-multipath: an algorithmic perspective," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 779–792, 2017.

[7] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Traffic engineering enhancement by progressive migration to SDN," *IEEE Communications Letters*, vol. 22, no. 3, pp. 438–441, 2018.

[8] A. Azzouni and G. Pujolle, "NeuTM: a neural network-based framework for traffic matrix prediction in SDN," in *IEEE/IFIP International Conference on Network Operations and Management Symposium (NOMS 2018)*, Taipei, China, April 2018.

[9] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. Proenca, "Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment," *IEEE Access*, vol. 8, pp. 83765–83781, 2020.

[10] S. A. Patil, L. A. Raj, and B. K. Singh, "Prediction of IoT traffic using the gated recurrent unit neural network- (GRU-NN-) based predictive model," *Security and Communication Networks*, vol. 2021, Article ID 1425732, 7 pages, 2021.

[11] B. Mao, F. Tang, Z. M. Fadlullah et al., "A novel non-supervised deep learning based network traffic control method for software defined wireless networks," *IEEE Wireless Communications*, vol. 25, no. 4, pp. 74–81, 2018.

[12] B. Mao, F. Tang, Z. M. Fadlullah, and N. Kato, "An intelligent route computation approach based on real-time deep learning strategy for software defined communication systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1554–1565, 2021.

[13] H. Yu, K. Li, and H. Qi, "An active controller selection scheme for minimizing packet-in processing latency in SDN," *Security and Communication Networks*, vol. 2019, Article ID 1949343, 11 pages, 2019.

[14] J. Chen, Y. Wang, X. Huang, X. Xie, H. Zhang, and X. Lu, "ALBLP: adaptive load-balancing architecture based on link-state prediction in software-defined networking," *Wireless Communications and Mobile Computing*, vol. 2022, Article ID 8354150, 16 pages, 2022.

[15] D. K. Dake, G. S. Klogo, J. D. Gadze, and H. Nunoo-Mensah, "Traffic engineering in software-defined networks using reinforcement learning: a review," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, pp. 330–345, 2021.

[16] D. Velasco, O. Rendon, and N. Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2021.

[17] W. Liu, J. Cai, Q. Chen, and Y. Wang, "RL-R: deep reinforcement learning approach for intelligent routing in software-defined data-center networks," *Journal of Network and Computer Applications*, vol. 177, pp. 1–16, 2021.

[18] M. Manel, A. Kamel, and Y. Habib, "DQR: an efficient deep Q-based routing approach in multi-controller software defined WAN (SD-WAN)," *Journal of Interconnection Networks*, vol. 20, no. 4, pp. 2150002–2150026, 2020.

[19] Y. Chen, A. Rezapour, W. Tzeng, and S. Tsai, "RL-routing: an SDN routing algorithm based on deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3185–3199, 2020.

[20] X. Che, W. Kang, Y. Ouyang, K. Yang, and J. Li, "SDN routing optimization algorithm based on reinforcement learning," *Computer Engineering and Applications*, vol. 57, no. 12, pp. 93–98, 2021.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations (ICLR 2016)*, pp. 187–200, San Juan, Puerto Rico, May 2016.

[22] Z. Yao, Y. Wang, L. Meng, X. Qiu, and P. Yu, "DPG-based energy-efficient flow scheduling algorithm in software-defined data centers," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 6629852, 10 pages, 2021.

[23] J. Lan, C. Yu, Y. Hu, and Z. Li, "A SDN routing optimization mechanism based on deep reinforcement learning," *Journal of Electronics and Information Technology*, vol. 41, no. 11, pp. 2669–2674, 2019.

[24] J. Lan, X. Zhang, Y. Hu, and P. Sun, "Software-defined networking QoS optimization based on deep reinforcement learning," *Journal on Communications*, vol. 40, no. 12, pp. 60–67, 2019.

[25] S. Kim, S. Yoon, and H. Lim, "Deep reinforcement learning-based traffic sampling for multiple traffic analyzers on software-defined networks," *IEEE Access*, vol. 9, pp. 47815–47827, 2021.

[26] Y. Hu, Z. Li, J. Lan, J. Wu, and L. Yao, "EARS: intelligence-driven experiential network architecture for automatic routing in software-defined networking," *China Communications*, vol. 17, no. 2, pp. 149–162, 2020.

[27] W. Liu, J. Cai, Q. Chen, and Y. Wang, "DRL-R: deep reinforcement learning approach for intelligent routing in software-defined data-center networks," *Journal of Network and Computer Applications*, vol. 177, article 102865, 2021.

[28] M. Ibrar, L. Wang, G. Muntean, J. Chen, N. Shah, and A. Akbar, "IHSF: an intelligent solution for improved performance of reliable and time-sensitive flows in hybrid SDN-based FC IoT systems," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3130–3142, 2021.

[29] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *35th International Conference on Machine Learning (ICML 2018)*, vol. 4, pp. 1587–1596, 2018.

[30] P. Sun, J. Lan, J. Shen, and Y. Hu, "An intelligent routing technology based on deep reinforcement learning," *Acta Electronica Sinica*, vol. 48, no. 11, pp. 2170–2177, 2020.

[31] P. Sun, J. Lan, J. Shen, and Y. Hu, "Pinning control-based routing policy generation using deep reinforcement learning," *Journal of Computer Research and Development*, vol. 58, no. 7, pp. 1563–1572, 2021.

[32] S. Ejaz, Z. Iqbal, P. Shah, B. H. Bukhari, A. Ali, and F. Aadil, "Traffic load balancing using software defined networking (SDN) controller as virtualized network function," *IEEE Access*, vol. 7, pp. 46646–46658, 2019.

[33] T. Schaul, J. Quan, L. Antonoglou, and D. Silver, "Prioritized experience replay," in *4th International Conference on Learning Representations (ICLR 2016)*, pp. 1–21, San Juan, Puerto Rico, May 2016.

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