Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 8354150, 16 pages
https://doi.org/10.1155/2022/8354150

Research Article

ALBLP: Adaptive Load-Balancing Architecture Based on Link-State Prediction in Software-Defined Networking

Junyan Chen,[1,2] **Yong Wang**,[2] **Xuefeng Huang**,[2] **Xiaolan Xie**,[2] **Hongmei Zhang**,[1] **and Xiaoye Lu**[2]

[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*

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

Received 11 October 2021; Accepted 20 January 2022; Published 12 February 2022

Academic Editor: Ming Yan

Load-balancing optimization in software-defined networking (SDN) has been researched for a long time. Researchers have proposed many solutions to the load-balancing problem but have rarely considered the impact of transmission delay between controllers and switches under high-load network conditions. In this paper, we propose an adaptive load-balancing architecture based on link-state prediction (ALBLP) in SDN that can solve the influence of transmission delay between controllers and switches on network load balancing. ALBLP constructs the prediction model of the network link status, adopts the long-term and short-term memory neural network (LSTM) algorithm to predict the network link-state value, and then uses the predicted value as the Dijkstra weight to calculate the optimal path between network hosts. The proposed architecture can adaptively optimize network load balancing and avoid the empty window period, in which the switch flow table does not exist by actively issuing the flow table. Under the network architecture, we collect the data set of the network link-state by simulating the GÉANT network, and we verify the effectiveness of the proposed algorithm. The experiment results show that the ALBLP proposed in this paper can optimize load balancing in SDN and solve the problem of transmission delay between controllers and switches. It has a maximum load-balancing improvement of 23.7% and 11.7% in comparison with the traditional Open Shortest Path First (OSPF) algorithm and the reinforcement learning method based on Q-Learning.

1. Introduction

In recent years, we have seen the rapid development of the Internet and an increasing growth in the prominence of its scale. The popularity of social media, high-definition movies, and online games has caused network traffic to grow rapidly. In this vast traffic data environment, traditional networks are confronted with even more severe challenges. Software-defined networking (SDN) is a new network-architecture revolution when compared to a traditional computer network. In SDN, due to the decoupling of the data plane and the control plane and due to a centralized control strategy, the artificial intelligence method can be directly deployed in the control plane and in the application plane [1].

The existing widely used open shortest path first (OSPF) or equal-cost multi-path (ECMP) routing methods in the current SDN structure calculate the shortest path or multiple optimal forwarding paths based on static local link information [2–4]. Because of lacking the intelligence to learn from previous experiences, these methods can lead to the repetition of wrong decisions when similar traffic patterns happen. The lousy path decision results in network congestion, which leads to further performance deterioration and cannot meet the global load-balancing requirements. Meanwhile, making routing decisions requires obtaining network link status information [5]. However, there are delays in the process of the controller periodically acquiring network link information. These include (i) the process delay of the switch processing the data packet; (ii) the link delay of the switch transmitting the data packet; and (iii) the command delay of the secure channel between the controller and the switch. The existence of these delays causes a difference

between the network link status information acquired by the controller and the current. After these delays, the current link-state value may have changed, causing the issued flow table not to use network resources effectively. Furthermore, it may increase network congestion under high-load conditions. Hence, it is necessary to obtain real-time network link status information to make forwarding decisions more accurate and improve the performance of network load balancing.

In this paper, we propose an adaptive load-balancing architecture based on dynamic link-state prediction (ALBLP) in SDN that can solve the influence on network load balancing of transmission delay between controllers and switches. ALBLP predicts the network link-state value through an improved long-term and short-term memory neural network (LSTM) model and then uses the predicted value as the Dijkstra weight to calculate the optimal path between network hosts and proactively issue the flow table. The data of the network status used for decision-making in the prediction model includes the transmission delay between the controllers and switches, thus reducing the effect of transmission delay.

The main contributions of this paper are as follows:

(1) Effective simulation experiments identify the influence of transmission delay between controllers and switches on network load balancing under high-load network conditions

(2) A prediction model of network link status is constructed to optimize network load balancing under high-load network conditions

(3) An adaptive load-balancing architecture based on link-state prediction in SDN is proposed that can solve the influence of transmission delay between controllers and switches on network load balancing

(4) A prototype system for the application of the proposed architecture is designed to verify its effectiveness

The rest of this paper is organized as follows. In Section 2, we discuss related research. Section 3 introduces the network architecture and optimization model. Section 4 describes the SDN load-balancing algorithm based on link-state prediction. Section 5 describes our evaluation and experimental results in detail. And finally, Section 6 concludes the paper and offers future directions for further investigation.

2. Related Research

In network management, accurate and timely network traffic data is essential for many downstream tasks. The characteristics of network traffic are that it is self-similar, multiscalar, long-term dependent, and highly nonlinear. These statistical characteristics determine the traffic predictability [6]. The traditional Poisson or Gaussian distributions cannot build the predictive model. The authors in [7] used Taylor's equation to predict network traffic changes, but its prediction effect was not effective, and its generalization ability was weak. With today's vigorous development of artificial intelligence technology, researchers have used machine learning algorithms to construct data prediction models in other fields. The authors in [8, 9] propose a position prediction model based on a weighted Markov model. The authors in [10] use an improved LSTM method to achieve better results in traffic flow prediction. The authors in [11] provided ultimate user experience to next generation mobile users by utilizing big data network analytics to predict users' service usage patterns and trajectories. The authors in [12] used the regression function of Convolutional Neural Network (CNN) to realize a polar code decoding structure.

Other researchers have used machine-learning methods in SDN. Azzouni and Pujolle [13] and Novaes et al. [14] verified the feasibility of LSTM in network traffic prediction. However, they only predicted traffic forwarding data between hosts. In an actual network, the number of hosts is often far greater than the number of switches, and so predicting traffic data between hosts plays a limited role in optimizing the data forwarding of switches. The work in [15] established LSTM models to predict the load of the SDN controller–having a significant effect on the load balance of the controller. The authors in [16] use the cache method based on LSTM to improve the cache hit rate, which can effectively improve the throughput of the data center. However, the network load-balancing problem of the data plane in SDN has not been solved.

In recent years, several studies have been carried out on network load balancing in the data plane. The authors in [17] propose a fuzzy comprehensive evaluation mechanism (FSEM), which dynamically selects paths to transmit network traffic according to changes in network status. The authors in [18] use a switch migration-based decision-making (SMDM) scheme to improve migration efficiency. The authors in [19] propose a traffic-scheduling method based on the ant colony algorithm. The authors in [20] capitalize on the merit of a fast global search of GA and efficient tracking of an optimal solution of ACO to enhance the performance of SDN. The authors in [21] propose a dynamic, multipath Dijkstra forwarding algorithm to optimize routing policy. The optimization effects and robustness of traditional algorithms for network load balancing are not ideal due to the limited perception of the network. Traditional algorithms cannot gather the network resource information from multiple dimensions such as time and space. Therefore, this often leads to local network congestion after distributing the flow table.

Methods based on deep learning (DL) or reinforcement learning (RL) can optimize network load balancing in multiple dimensions in the form of experience. The authors in [22, 23] propose a routing strategy that utilizes CNN to choose the paths combinations according to the network traffic trace in an online fashion. This strategy can avoid congested paths and balance the network traffic. However, it does not consider the influence of time dimension in a network. The authors in [24] propose the Reinforcement Learning and Software Defined Networking Intelligent Routing (RSIR) algorithm based on Q-learning, which optimizes the network in three dimensions: throughput,

delay, and packet loss rate. However, its Q-learning method requires significant time to train the Q-table between each host pair after collecting link information. It has no learning ability for historical network information and has high requirements for the controller's performance. The authors in [25] propose routing based on deep reinforcement learning (DRL-R). The authors in [26] propose a multi-controller routing planning algorithm based on a deep Q-learning network (DQN). The authors in [27] adopt the Deep Deterministic Policy Gradient (DDPG) algorithm to find the optimal scheduling scheme for flows. They achieved strong improvement in network load balancing, throughput, and QoS optimization. The authors in [28] proposed a real-time distributed learning method, which uses a multiagent deep reinforcement learning (MADRL) model to realize the independent routing decision of each edge node. This method overcomes the weakness of conventional turn-based DRL methods and increases delivered packet ratios and effective throughput. But the RL algorithm must constantly obtain network link status data as its environmental state, and so the transmission delay between the controllers and the switches may cause a difference between the collected data and the actual data.

There is still a lack of relevant research that considers the impact of transmission delay between controllers and switches on network load balancing. This delay cannot be ignored in the real network because the network load-balancing algorithm depends on the real-time network state. In our paper, we propose an adaptive load-balancing architecture based on link-state prediction in SDN that can solve this problem.

3. Network Architecture and Optimization Model

This section describes the overall architecture of the design and builds a model for the load-balancing optimization objectives.

3.1. Network Architecture. According to the application scenario of SDN, we designed a network architecture as shown in Figure 1. We used the OpenFlow 1.3 protocol to simulate the SDN network. The description of each part of the specific network architecture is as follows:

3.1.1. Data Plane. The data plane consists of hosts and SDN switches, and it uses the standard OpenFlow protocol to connect the control plane through the SDN switch.

3.1.2. Control Plane. The control plane comprises SDN controllers, and it is an intermediate network component that controls the data plane and transmits data to the application plane.

The communication between the controller and the switch includes the following:

(i) Establishing the connection between the controller and the switch

(ii) The controller requesting information from the switch

(iii) The switch sending packet_in messages to the controller without matching flow entries

(iv) The controller issuing and updating the flow entries to the data plane

In this paper, the control plane is divided into the following modules:

(i) The network topology discovery module: the controller periodically sends a connection message to the switch to detect the network topology through this module.

(ii) The flow monitoring module: this module regularly obtains the network state information and provides a fundamental basis for decisions of the load-balancing algorithm.

(iii) The flow entry distribution module: we adopt the method of actively issuing flow entries to periodically update flow entries based on the network status and the load-balancing algorithm in this module. Flow entries are actively delivered by the controller periodically generating a data packet forwarding strategy according to the acquired network status and sending it to all switches. When we compare it with passively sending flow entries, we see that active delivery will not generate a considerable window period when the flow entry does not exist. Figure 2 shows the difference between issuing a flow table actively or passively.

(iv) The message processing module: this module monitors and handles the packet_in message uploaded by the switch.

3.1.3. Application Plane. The application plane customizes different applications according to needs in various scenarios. In this article, the application plane uses the network load-balancing optimization algorithm based on LSTM to solve the problem of the transmission delay between the controller and the switch in obtaining network link status. Therefore, this plane consists of the following modules:

(i) The storage module: this module stores the network link information as the basis of the LSTM training module.

(ii) The LSTM training module: it uses the historical network link information as training data to train the LSTM model in this module. The trained model is used as the model basis of the load-balancing optimization algorithm module.

(iii) The load-balancing optimization algorithm module: this module takes the received residual bandwidth as the input of the trained LSTM model to obtain the predicted residual bandwidth. It uses the predicted value as the Dijkstra weight to calculate the optimal path between network hosts and then sends the updated flow entries to the data plane.

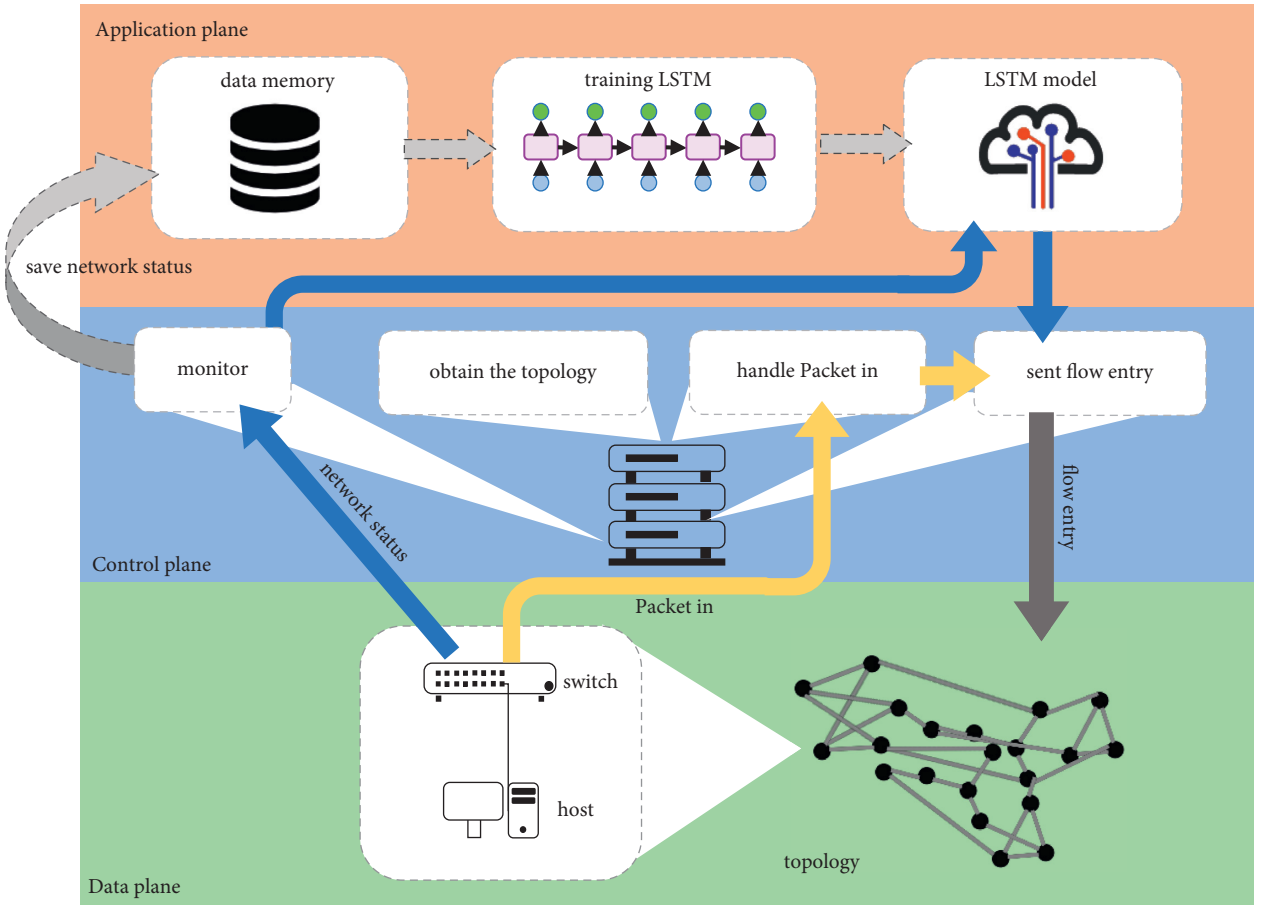

Figure 1: ALBLP network architecture. The architecture consists of the data plane, control plane, and application plane.

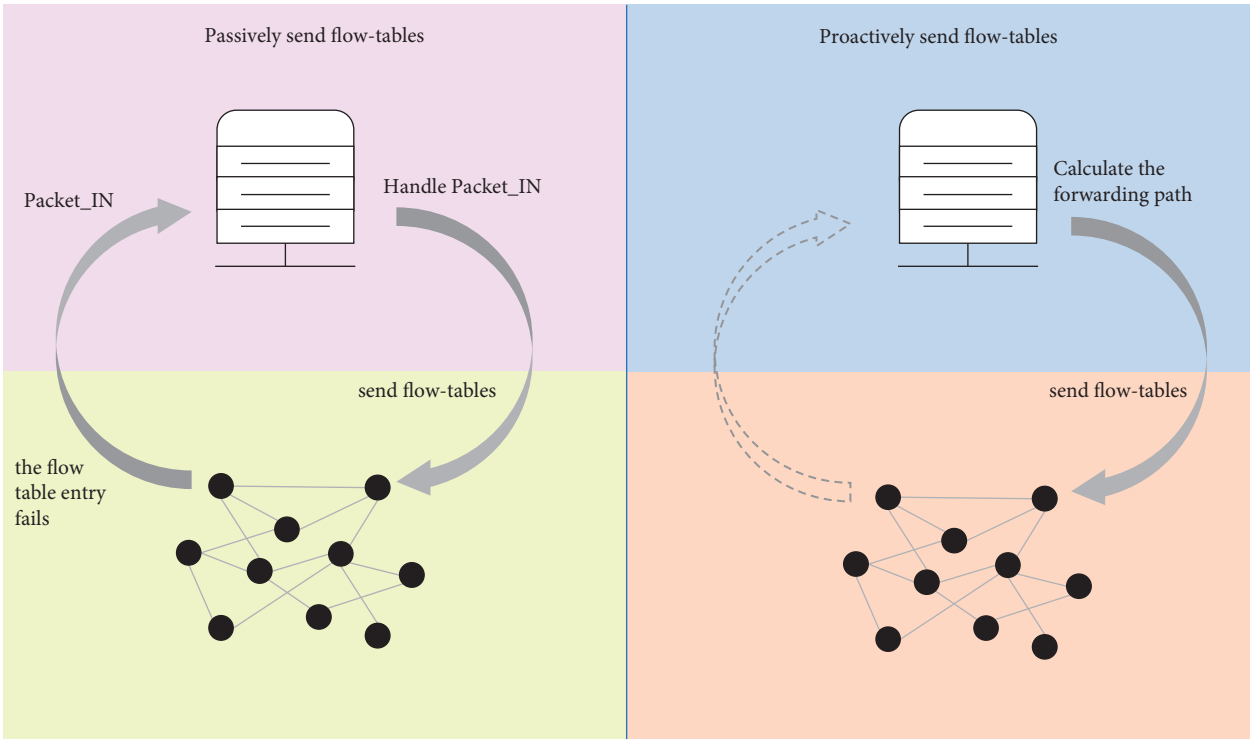

Figure 2: The difference between active and passive send-flow tables.

3.2. Load-Balancing Optimization Model. In SDN, equation (1) represents the graph structure of the data plane. We use N to denote the number of forwarding nodes; V is the set of all forwarding nodes v, that is, $V = [v_1, v_2, \ldots, v_N]$; E is the set of all links e. A link e can be defined by connecting two forwarding nodes, that is, $\forall e_{v_i v_j} \in E$, such that $v_i, v_j \in V$ and $v_i \neq v_j$. W represents the weight matrix of the link forwarding, that is, $W = [w_{ij}]_{N\times N}$, where w_{ij} represents the weight of the link between the i and j nodes. If the link exists, w_{ij} is a positive real number; otherwise, w_{ij} is 0, as shown in equation (2).

$$\text{Graph} = (V, E, W), \tag{1}$$

$$w_{ij} = \begin{cases} w_{ij} \in R^+, & e_{v_i v_j} \in E, \\ 0, & \text{else.} \end{cases} \tag{2}$$

We use B to denote the link bandwidth matrix for the entire data plane, that is, $B = [b_{ij}]_{N\times N}$, where b_{ij} represents the bandwidth of the link between the i and j nodes. If the link exists, b_{ij} is a positive real number; otherwise, b_{ij} is -1, as shown in the following equation:

$$b_{ij} = \begin{cases} b_{ij} \in R^+, & e_{v_i v_j} \in E, \\ -1, & \text{else.} \end{cases} \tag{3}$$

We use U_t to describe the used bandwidth matrix in the t time period, that is, $U_t = [u_{ij}^t]_{N\times N}$, where u_{ij}^t represents the used bandwidth between the i and j nodes in the t time period. If the link exists, u_{ij}^t is a positive real number; otherwise, u_{ij}^t is -1, as shown in the following equation:

$$u_{ij}^t = \begin{cases} u_{ij}^t \in R^+, & e_{v_i v_j} \in E, \\ -1, & \text{else.} \end{cases} \tag{4}$$

We use R_t to denote the link remaining bandwidth matrix of the data plane in the t time period, that is, $R_t = [r_{ij}^t]_{N\times N}$, where r_{ij}^t represents the remaining bandwidth between the i and j nodes in the t time period. If the link exists, r_{ij}^t is the value of b_{ij} minus u_{ij}^t; otherwise, r_{ij}^t is -1, as shown in equation (5). R_t is averaged and normalized to obtain R_t' as shown in equation (6), and the link weight can be expressed by equation (7).

$$r_{ij}^t = \begin{cases} r_{ij}^t = b_{ij} - u_{ij}^t, & r_{ij}^t \in R^+, e_{v_i v_j} \in E, \\ -1, & \text{else,} \end{cases} \tag{5}$$

$$R_t' = \left[r_{ij}^{'t} = \begin{cases} \dfrac{r_{ij}^t - \text{MIN}(R_t)}{\text{MAX}(R_t) - \text{MIN}(R_t)} & e_{v_i v_j} \in E \\ -1 & \text{else} \end{cases} \right]_{N\times N}, \tag{6}$$

$$W = \left[w_{ij} = \begin{cases} r_{ij}^{'t} & e_{v_i v_j} \in E \\ 0 & \text{else} \end{cases} \right]_{N\times N}. \tag{7}$$

Evaluating the performance of a network is essential. In this paper, we pay attention to the degree of load balancing in the network. We use the ratio of the used bandwidth to the link bandwidth to describe link utilization, as shown in equation (8), where $(\rho_t)_{ij}$ represents the link utilization rate between the i and j nodes.

$$\rho_t = \left[(\rho_t)_{ij} = \frac{u_{ij}^t}{b_{ij}} \right]_{N\times N}. \tag{8}$$

We use equation (9) to evaluate the network load-balancing factor β.

$$\beta = \frac{1}{N-1} \sum_{i=1,j=1}^{N} \left[(\rho_t)_{ij} - \text{avg}(\rho_t) \right]^2 \times b_{ij}. \tag{9}$$

In SDN, the load-balancing factor describes the relative degree of network link utilization. The forwarding of the same length of a data packet may cause a difference in network throughput due to the link bottleneck in the forwarding path [29]. Therefore, we describe the network throughput as Nt-shown in equation (10), 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 is a measure of the carrying capacity of the data plane:

$$Nt = \sum_{i=1}^{L_H} H_i. \tag{10}$$

In this paper, the optimization goal of load balancing is defined as equation (11). Equation (12) represents the limiting condition for obtaining the optimization goal:

$$\min(\beta), \tag{11}$$

$$s.\text{t} \begin{cases} \beta \geq 0, \\ b_{ij} > u_{ij}^t. \end{cases} \tag{12}$$

4. Adaptive Load-Balancing Algorithm Based on Link-State Prediction

This section describes the principle of the improved LSTM training model and the adaptive load-balancing algorithm based on link-state prediction in SDN.

4.1. Improved LSTM Training Model

4.1.1. LSTM Training Target. In this paper, the specific prediction task of the LSTM model is to predict the remaining bandwidth value between the switches in several consecutive time intervals in the future based on the remaining bandwidth observation values in several historical time intervals. The remaining bandwidth between the switches in the entire network within time interval t can be expressed as R_t. The historical time intervals of input and output are represented by W_{in} and W_{out} respectively. We use $X_t = (R_{t-W_{\text{in}}}, R_{t-W_{\text{in}}+1}, \ldots, R_t)$ to represent the historical interval observation value, and we use $Y_{t+1} = (R_{t+1}, R_{t+2}, \ldots, R_{t+W_{\text{out}}+1})$ to define the prediction interval value. The target of the model training is the mapping denoted by $f(\cdot)$ in

$$X \xrightarrow{f} (\cdot)\, Y. \tag{13}$$

4.1.2. LSTM Model. The model in this paper is composed of one LSTM layer, one Dropout layer, and one Full-connection layer. We use sliding windows to set the data. The two windows are separated by one time slice, and the input is the historical observation value X_t. The loss function uses mean squared error (MSE), and the optimizer uses the Adam optimizer. We use $x^t = X_t$ to represent the LSTM input. In LSTM, the number of memory cells is set to 5, and each memory cell contains four units-Input Gate, Forget Gate, Cell, and Output Gate-for calculation.

The Input Gate is shown in equation (14). a_l^t is calculated in equation (15), where s_c^{t-1} is the cell at the previous moment. The Forget Gate is shown in equation (16), where a_ϕ^t is calculated in equation (17). The input of the cell is x_i^t, q_l^t, and q_ϕ^t, which are, respectively, the input feature, the output of the Input Gate, and the output of the Forget Gate. The cell is calculated in equation (18). The Output Gate is shown in equation (19), where a_w^t is as shown in equation (20). The final output of the memory cell is shown in equation (21).

$$q_l^t = f\left(a_l^t\right), \tag{14}$$

$$a_l^t = \sum_{i=1}^{I} \omega_{il} x_i^t + \sum_{c=1}^{C} \omega_{cl} s_c^{t-1}, \tag{15}$$

$$q_\phi^t = f\left(a_\phi^t\right), \tag{16}$$

$$a_\phi^t = \sum_{i=1}^{I} \omega_{i\phi} x_i^t + \sum_{c=1}^{C} \omega_{c\phi} s_c^{t-1}, \tag{17}$$

$$s_c^t = q_\phi^t s_c^{t-1} + q_\iota^t g\left(\sum_{i=1}^{I} \omega_{ic} x_i^t\right), \tag{18}$$

$$q_w^t = f\left(a_w^t\right), \tag{19}$$

$$a_w^t = \sum_{i=1}^{I} \omega_{iw} x_i^t + \sum_{c=1}^{C} \omega_{cw} s_c^{t-1}, \tag{20}$$

$$q_c^t = q_w^t h\left(s_c^t\right). \tag{21}$$

4.1.3. Training Algorithm. According to the LSTM training objectives and the LSTM model, we draw the LSTM training algorithm as shown in Algorithm 1.

4.2. Adaptive Load-Balancing Optimization Algorithm. We obtain the LSTM training model as shown in the previous section. In this current section, we describe the ALBLP algorithm that optimizes the load balancing in SDN based on link-state prediction.

In SDN, the controller's decision depends on the link information obtained from the data plane. If the controller cannot receive the link information of the data plane in time, it will not be able to calculate accurately and to decide how to optimize the path. We use Delay_{cs} to represent the transmission delay between the controllers and switches in obtaining network link-state information. We calculate the time from the controller sending a link status request to the controller receiving the reply to the request as the value of Delay_{cs}. We use the forwarding state of the network to describe the impact of the transmission delay between the controllers and the switches, that is, the load-balancing factor proposed in equation (9).

According to the model obtained by LSTM, the historical matrix sequence X_t in the W_{in} interval can be input to obtain the prediction matrix sequence Y_{t+1}. However, the matrix sequence Y_{t+1} cannot be directly used as the basis for making a decision, because Delay_{cs} needs to be considered in the input time interval when placing the trained model in the network. It is necessary to select which of the prediction matrix sequences to use as the basis for eliminating the influence of Delay_{cs} as much as possible. We set the prediction time interval as T_f and the length of Delay_{cs} as T_d. The input $X_{k\times T_f}$ and output $Y_{k\times T_f+1}$ of the model in the network are expressed by equations (22) and (23), where k is the number of cycles, and the prediction interval T_f is calculated in equation (24).

$$X_{k\times T_f} = \left(R_{k\times T_f - W_{\text{in}}}, \ldots, R_{k\times T_f - W_{\text{in}}+T_d}, \ldots, R_{k\times T_f}\right), \tag{22}$$

$$Y_{k\times T_f+1} = \left(R_{k\times T_f+1}, \ldots, R_{k\times T_f+T_d}, \ldots, R_{k\times T_f+W_{\text{out}}+1}\right), \tag{23}$$

$$T_f = W_{\text{out}} - \text{Delay}_{\text{cs}}. \tag{24}$$

Figure 3 shows the input and output of the model in the network. The data used for deciding in the network is the Delay_{cs} time slice shifted backwards in the prediction matrix sequence, which is the Usable values part in Figure 3. We then use each matrix sequence in the Usable values part as the weight of the Dijkstra algorithm according to the time advancement. Finally, the flow entries are issued according to the forwarding decision output by the Dijkstra algorithm. We draw the ALBLP algorithm shown in Algorithm 2. Our ALBLP is an application-layer algorithm, and, as shown in Figure 1, the trained LSTM model will also be a part of the application plane ALBLP and will be called by ALBLP. As shown in Algorithm 2, this algorithm will periodically obtain the link information from the data plane through the controller as input, standardize the input data, and then call the LSTM model to predict the link information. The predicted link information needs to subtract part of the data that has been invalid once the time delay exists, affected by the time delay. Then, it needs to destandardize the remaining data and input the Dijkstra algorithm to obtain the forwarding path of the switch of the entire data plane. Finally, the flow entry obtained according to the forwarding path is sent to the controller and then sent to the data plane switch through the controller.

Input:
Link residual bandwidth matrix sequence X_t;
The input time series length W_{in} and the output time series length W_{out};
Training batch size *batch_size and* training times *epochs*;
Output:
model parameters after training;
(01) **for** $i = 1$ to $(n - W_{\text{in}} - W_{\text{out}} + 1)$ **do**
(02) $\mathcal{X}_i = (R_{i-W_{\text{in}}}, R_{i-W_{\text{in}}+1}, \ldots, R_i)$
(03) $\mathcal{Y}_{i+1} = (R_{i+1}, R_{i+2}, \ldots, R_{i+W_{\text{out}}+1})$;
(04) *Data_train add* $(\mathcal{X}_i, \mathcal{Y}_i)$;
(05) **end for**
(06) Initialize model *net*;
(07) **repeat**
(08) Take out *Data_train* from *Data_batch*, the size is *batch_size*;
(09) $Out = net(Data\_batch)$;
(10) $Loss = MSEloss(Out, \mathcal{Y}_i)$;
(11) Back propagation to update model parameters;
(12) $epochs = epochs - 1$;
(13) **until** $epochs = 0$

Algorithm 1: LSTM training algorithm.

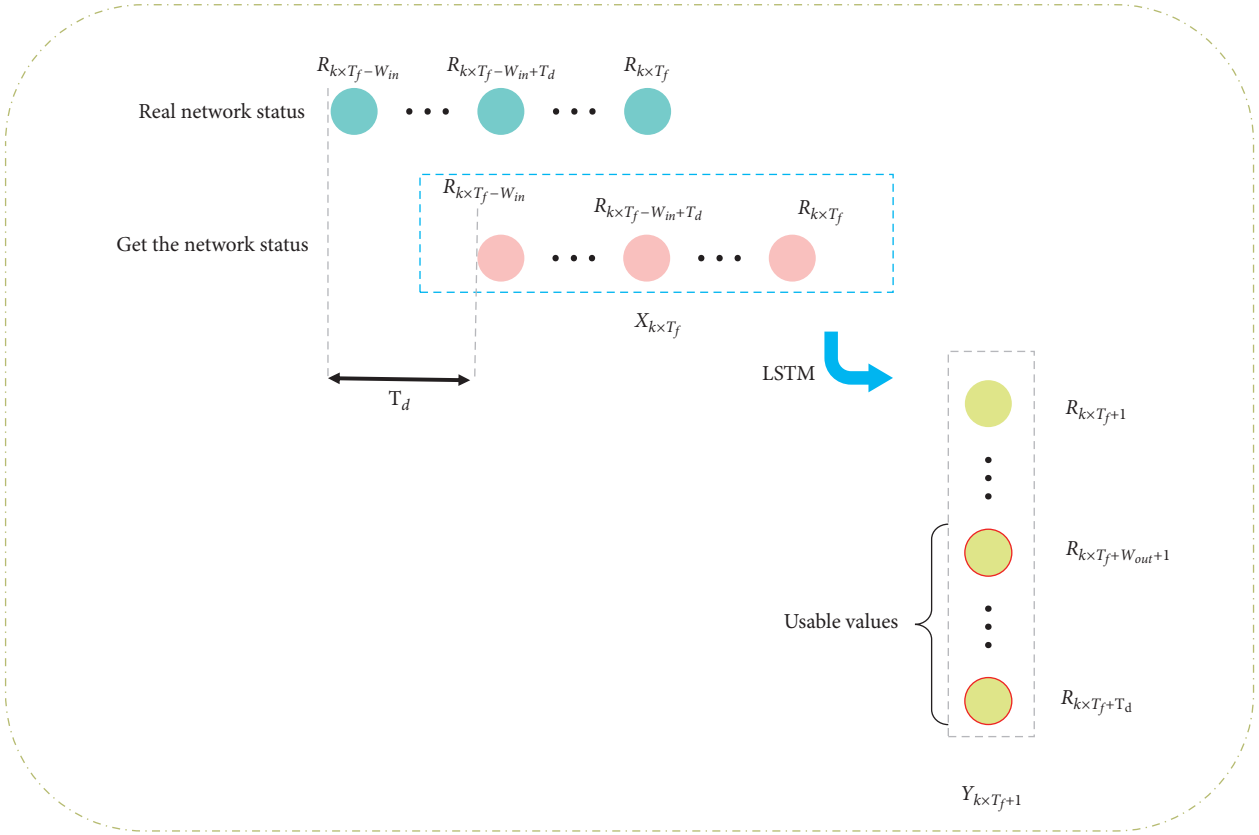

Figure 3: Input and output of the LSTM model in the experimental network. The data used for deciding on the network is the Delay_{cs} time slice shifted backwards in the prediction matrix sequence.

Input:
The network link-state value obtained periodically;
Output:
Flow entries used to control the SDN switch;
(01) Initialize the List of network link sequences $List\_in = []$;
(02) Initialize the Data prediction flag $flag\_dp$ = True;
(03) Obtain the network link-state value and assign it to R;
(04) Obtain the length of $Delay_{cs}$ and assign it to T_d;
(05) List_in.append (R);
(06) **if** size_of ($List\_in$) <= T_f **then**
(07) $Graph$ = set_graph (R);
(08) send_flow_entries (DijKstra($Graph$));
(09) **else if** size_of ($List\_in$) >= T_f and $flag\_dp$ = = True **then**
(10) X = standardization ($List\_in[-T_f, :]$);
(11) Y = de_standardization (**LSTM_forecast(X)**);
(12) $Y = Y\ [-\text{int}(T_f - T_d),:]$;
(13) $flag\_dp$ = False;
(14) **else then**
(15) $Graph$ = set_graph ($Y[0]$);
(16) Y = delete ($Y[0]$);
(17) send_flow_entries (DijKstra($Graph$));
(18) **if** size (Y) = = 0 **then**
(19) $flag\_dp$ = True;
(20) **end if**
(21) **end if**

Algorithm 2: ALBLP algorithm.

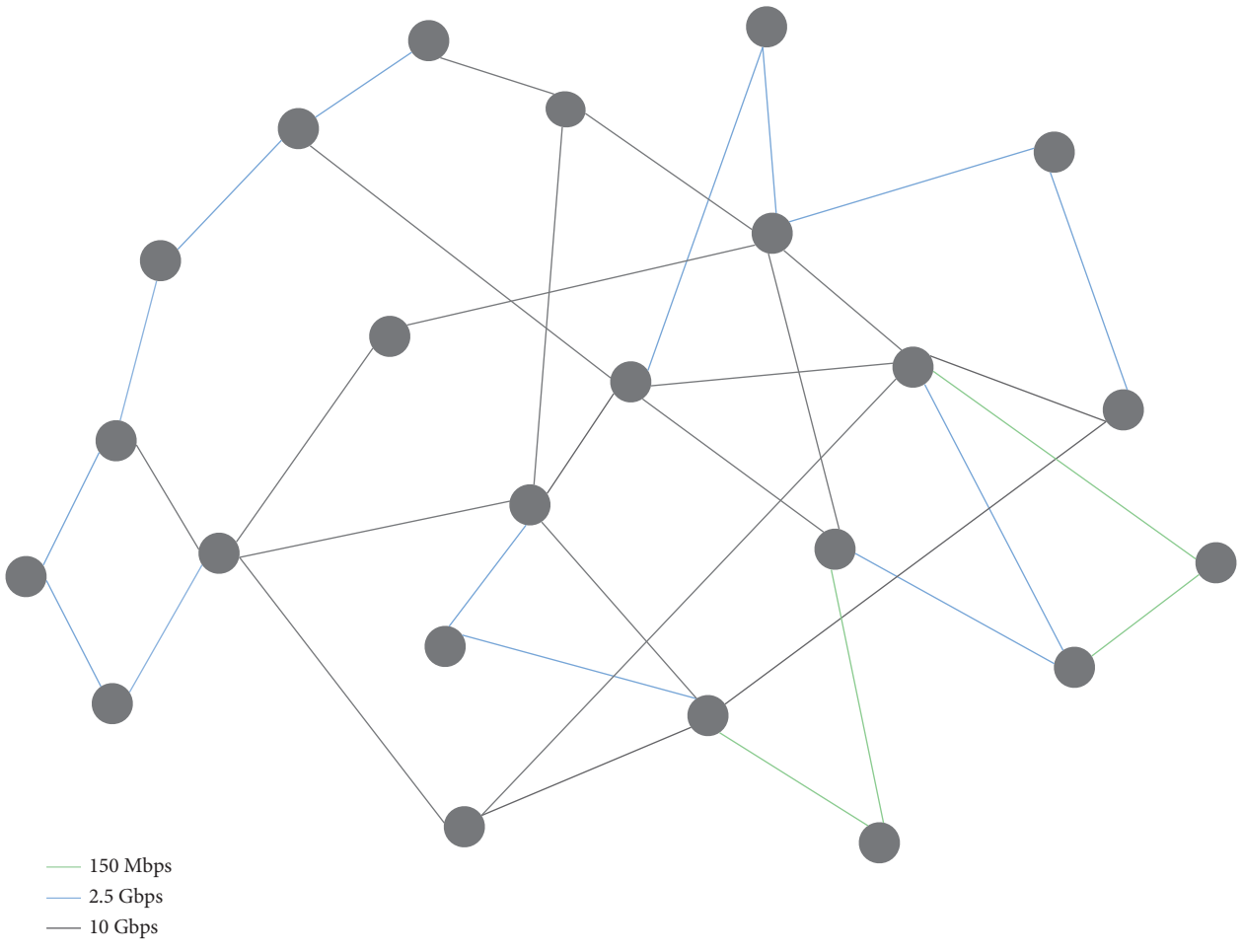

Figure 4: GÉANT network topology. The GÉANT network has 23 switches and 37 links, including 3 different bandwidth links: 10 Gbps, 2.5 Gbps, and 155 Mbps.

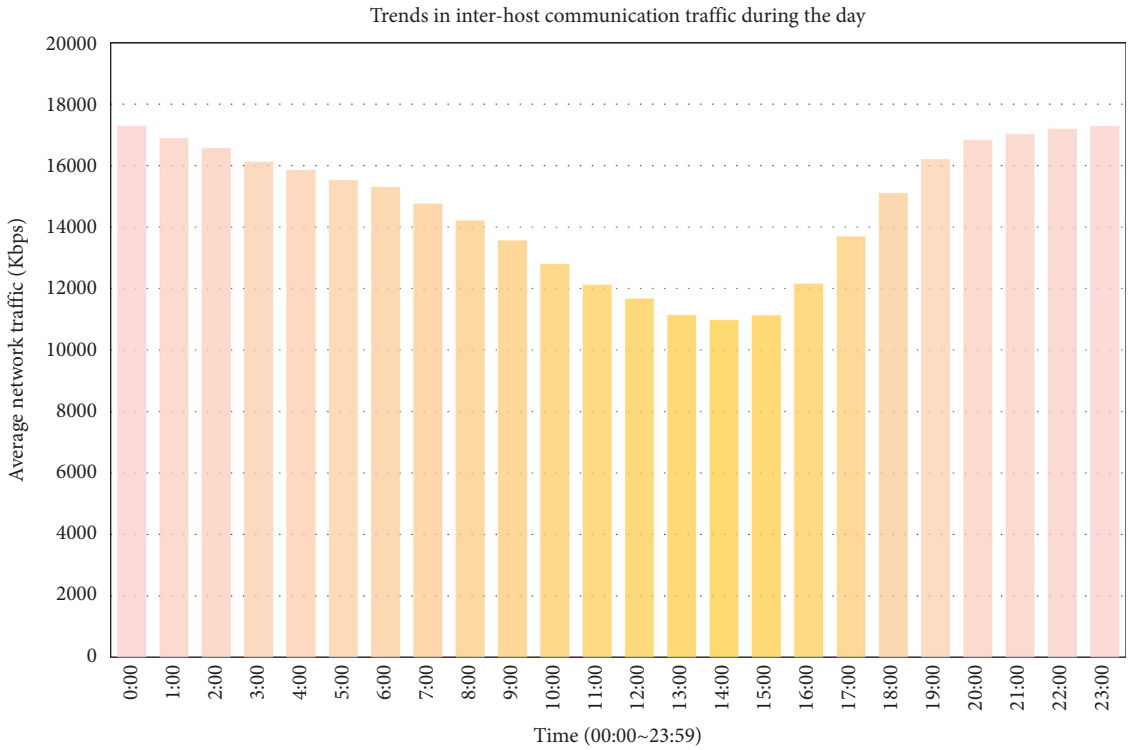

Figure 5: Trends in interhost communication traffic during the day.

5. Experiments and Analysis

For the experiment, we used the *Ubuntu* 20.4 operating system, *Mininet* network simulation, *Open vSwitch*, *Ryu* controller [30], and the OpenFlow 1.3 protocol. The computer used in the simulation experiment was equipped with an AMD Ryzen 7 2700 CPU with a base frequency of 3.2 GHz, an NVIDIA GTX1080 graphics card, and 16 GB of RAM memory.

5.1. Experimental Setup. In our experiment, we used the highly recognized GÉANT network [31], as shown in Figure 4. The GÉANT network is a data network for the European research and education community. In it, there are 23 switches and 37 links, including 3 different bandwidth links: 10 Gbps, 2.5 Gbps, and 155 Mbps. In this article, due to the performance limitations of the simulation environment, our experiment limits the link bandwidth to three bandwidths of 100 Mbps, 25 Mbps, and 1.55 Mbps. Each switch is connected to a host for receiving and sending data.

5.2. Simulation Traffic Matrix. In our experiment, we used the *Iperf* tool to send traffic. Communication traffic between hosts used the public intradomain traffic matrix [32]. The traffic matrix data was the four-month flow data in 2006, and the collection interval was 15 minutes. The average, one-day traffic packet trend is shown in Figure 5. It can be seen from Figure 5 that the peak flow is from 19 : 00 to 1 : 00 the next day in the traffic matrix. We wrote a script to send packets between hosts based on the experimental traffic matrix data.

Because the experimental link bandwidth is reduced to 1/100 of the original link bandwidth, the flow matrix between hosts is also reduced to 1/100 of the original flow matrix. We assume that the traffic matrix between the hosts is X. In order to verify the effectiveness of the algorithm under different traffic intensities, the traffic intensity set in this paper is n times of X, where n is the traffic intensity coefficient.

5.3. Experimental Results

5.3.1. LSTM Prediction Result. Our experiment used the SDN controller to collect the remaining bandwidth value of links between switches as a training data set. The experimental environment and traffic matrix are described in 5.1 and 5.2. The forwarding decision during this period is the Open Shortest Path First (OSPF) [4] algorithm. Each traffic matrix continued to send packets for 30 seconds, and the flow monitoring module collected a value every 1 second. A total of 21738240 pieces of the remaining bandwidth value were collected, of which 70% were used as a training set, 20% as a verification set, and 10% as a test set.

In our experiment, we set $W_{\text{in}} = W_{\text{out}}$, obtained from 5 to 30 link acquisition cycles in steps of 5 link acquisition cycles. The evaluation indexes are mean absolute error (MAE), root mean square error (RMSE), and symmetric mean absolute percentage error (SMAPE). The results are shown in Table 1 and Figure 6. According to the training results, we set $W_{\text{in}} = W_{\text{out}} = 20$. Other hyperparameters such as DropOut, Batch size, number of LSTM cells, and Hidden size of LSTM are shown in Table 2.

Table 1: Evaluation index of the forecast effect.

Evaluation index		MAE	RMSE	sMAPE
LSTM	$W_{in} = W_{out} = 5$	2.44	5.50	11.46
	$W_{in} = W_{out} = 10$	2.22	4.70	11.13
	$W_{in} = W_{out} = 15$	1.57	3.20	10.37
	$W_{in} = W_{out} = 20$	1.35	2.68	9.62
	$W_{in} = W_{out} = 25$	1.37	2.57	10.43
	$W_{in} = W_{out} = 30$	1.56	3.02	10.66

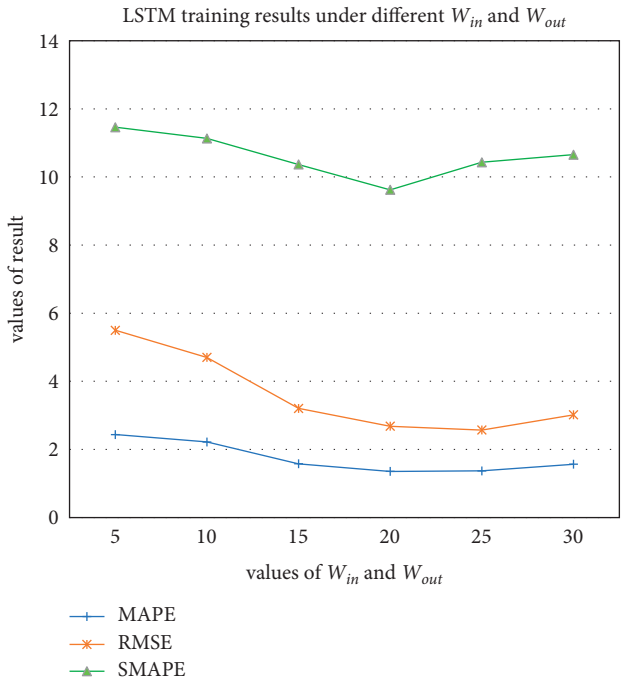

Figure 6: LSTM training results under different W_{in} and W_{out}.

Table 2: LSTM training hyperparameters.

Hyperparameters	Value
Epochs	2000
Hidden size	256
Cell	5
Batch size	128
Learning rate	0.001
DropOut	0.8
W_{in} & W_{out}	20

According to the above hyperparameters, we used the improved LSTM model to predict the remaining bandwidth of the network, as shown in Figure 7. It can be seen from Figure 7 that the predicted remaining bandwidth value is close to the actual remaining bandwidth value. The experiment results show that the improved LSTM model has a better effect in predicting the remaining bandwidth of the network link.

In the experiment, the loss of LSTM training decreases as shown in Figure 8. It essentially converges after 1000 training iterations (loss < 0.01).

5.3.2. Influence of Different Delay$_{cs}$ on the Algorithm. In order to verify the effectiveness of the Dijkstra algorithm under different Delay$_{cs}$, we experimentally observed the network load balancing of the OSPF algorithm under different Delay$_{cs}$. The unit of Delay$_{cs}$ is the link acquisition cycle. We set the link acquisition cycle as one second in the experiment. The experimental flow matrix adopted a random flow matrix in the peak flow of a day and conducted multiple experiments under the condition of the flow intensity coefficient of 1–10. Figure 9 shows a comparison of the network load-balancing factor β proposed in equation

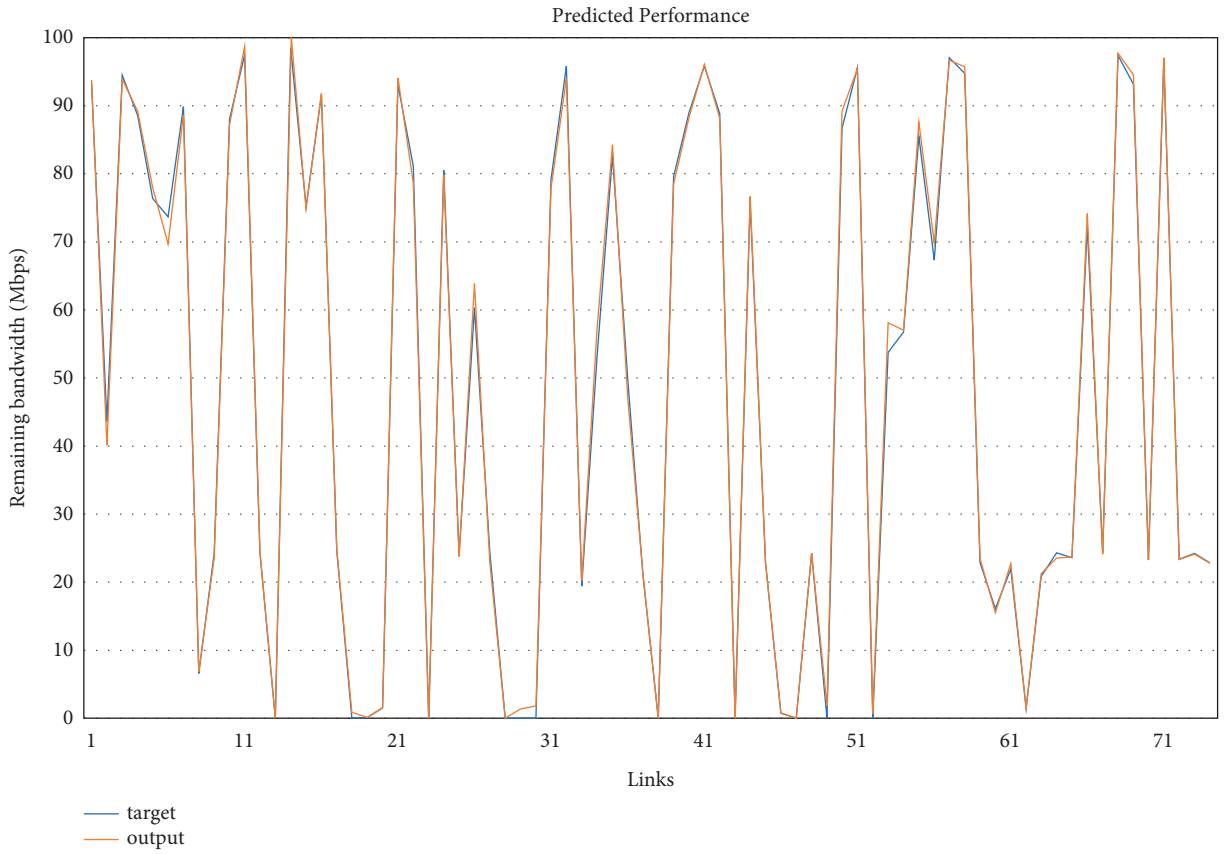

Figure 7: Display of the prediction effect of a time slice.

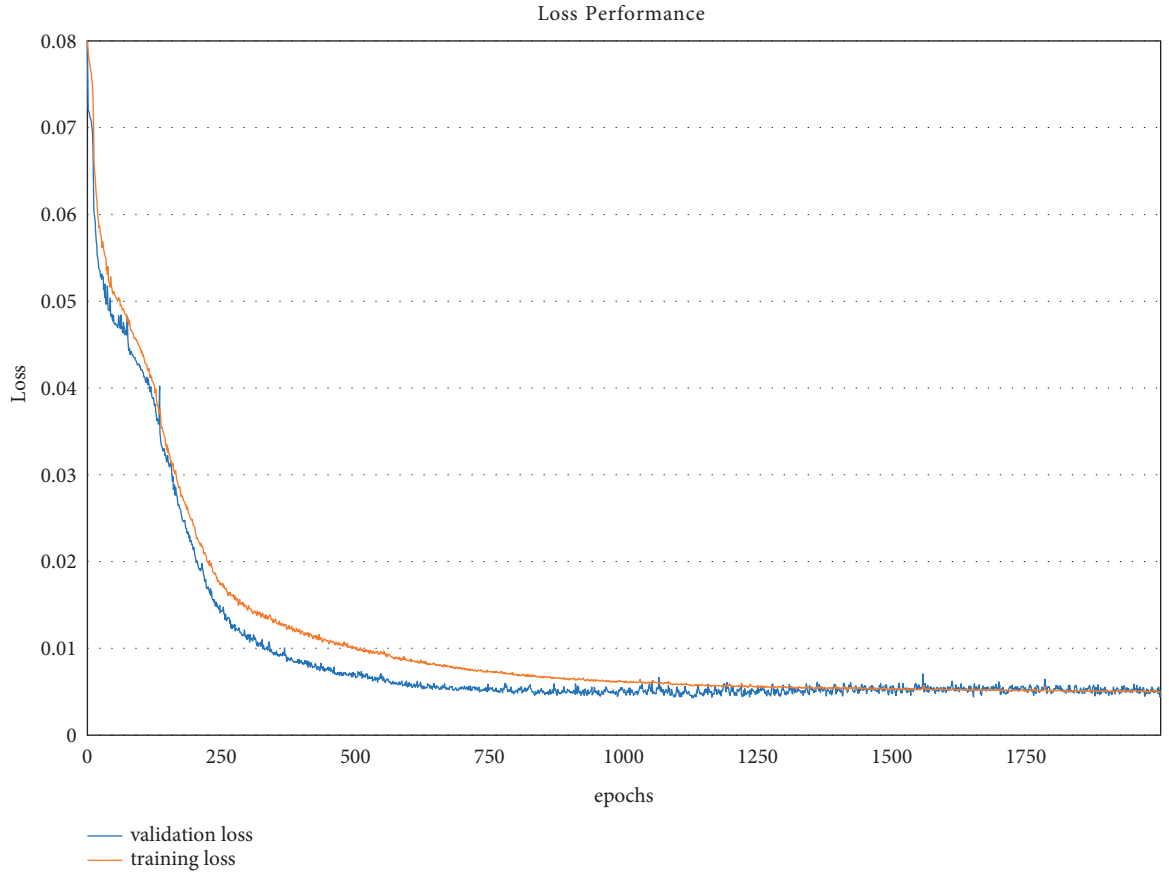

Figure 8: The loss of training and verification decreased.

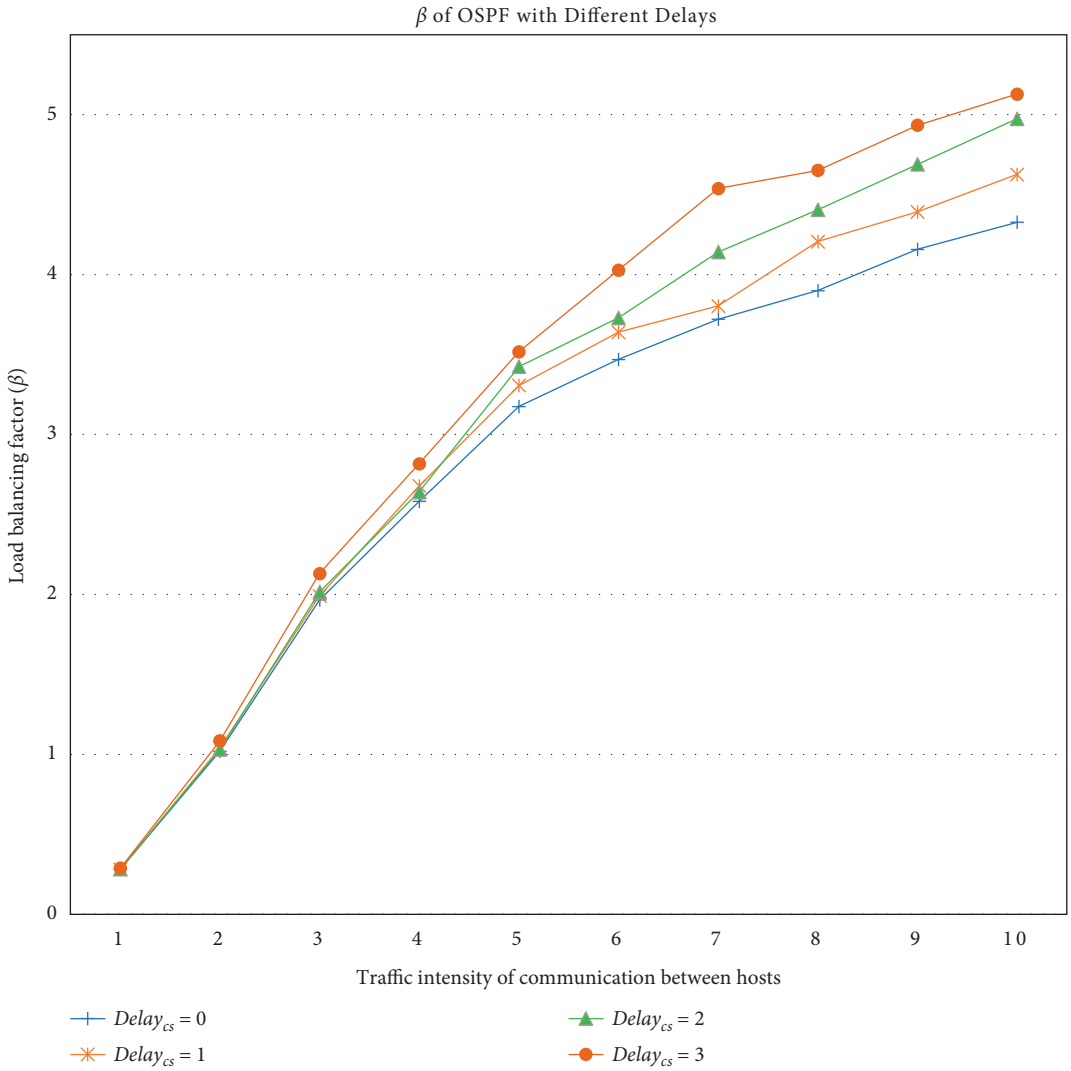

Figure 9: Load-balancing factor of OSPF with different Delay_{cs}.

(9) under different Delay_{cs}. According to the optimization goal of the load balancing in equation (11), the lower the β, the better. When the traffic intensity of communication between hosts is 1, β under each Delay_{cs} is similar. However, the higher the Delay_{cs} is, the more unbalanced the network load is with the increase in traffic intensity. Therefore, the impact of Delay_{cs} on network performance cannot be ignored.

5.3.3. Comparison of the ALBLP Algorithm, RSIR Algorithm, and OSPF Algorithm. In order to verify the effectiveness of the proposed ALBLP algorithm in solving the problem of Delay_{cs} in network performance, our experiment was designed to simulate different Delay_{cs} and conduct multiple experiments with a traffic intensity coefficient of 1–10 to compare the effectiveness of the ALBLP, RSIR [24], and OSPF [4] algorithms. The experimental results are shown in Figures 10 and 11.

Figure 10 shows a comparison of the network throughput Nt proposed in equation (10) of the three algorithms under the same traffic matrix input, where the higher the Nt, the better. The network throughput of the three algorithms is similar under low load (a traffic intensity coefficient $n < 5$). But in a high-load scenario, the network throughput of the ALBLP algorithm is greater than that of the OSPF algorithm and the RSIR algorithm. This is because, under low-load conditions, the performance of the network has been guaranteed. If the network performance is redundant, the network throughput will not fluctuate substantially. In the case of high load, if the network load-balancing problem cannot be effectively solved, part of the network will suffer from congestion, and network throughput will decrease. However, the experimental network traffic is already close to the network limit when the traffic intensity is 10. In Figure 10(a), when Delay is 0, ALBLP is not significantly improved compared to the other two algorithms, while in (b), (c), and (d), ALBLP has a higher Nt than the other two algorithms.

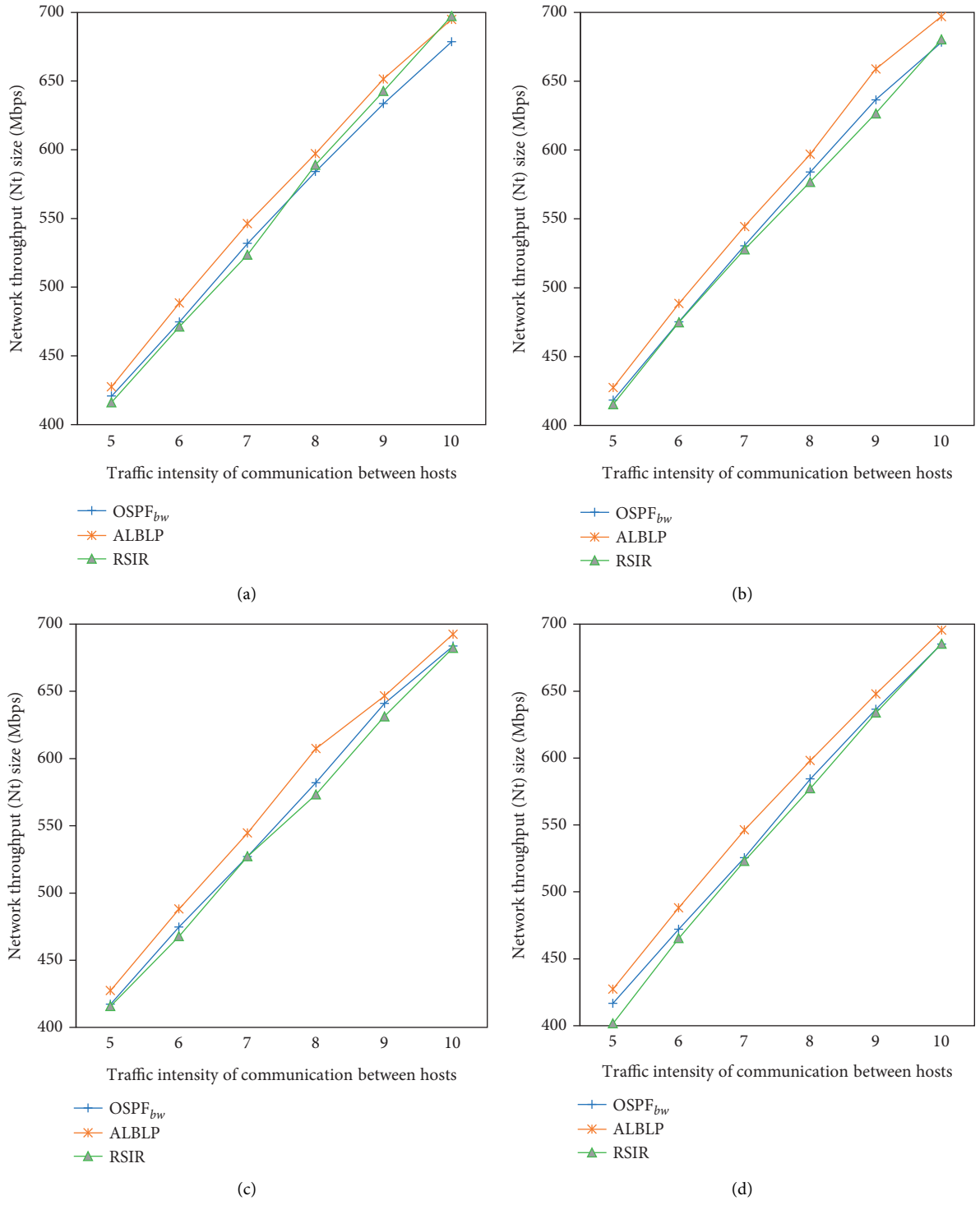

Figure 10: Network throughput of ALBLP, OSPF, and RSIR under different traffic intensities and different Delay$_{cs}$. (a) Delay$_{cs}$ = 0. (b) Delay$_{cs}$ = 1. (c) Delay$_{cs}$ = 2. (d) Delay$_{cs}$ = 3.

Figure 11 shows a comparison of the load-balancing factor β of the three algorithms under the same traffic matrix input, where the lower the β, the better. When Delay$_{cs}$ is 0, the effectiveness of the three algorithms for load balancing is similar. When Delay$_{cs}$ equals 1, 2, and 3 link acquisition cycles, respectively, the ALBLP algorithm is better than the OSPF and RSIR algorithms in load balancing under high load, and the degree of optimization increases with the increase of Delay$_{cs}$. Under low-load conditions, the network link has less occupancy rate, and the data plane has greater network redundancy. Therefore, the network load-balancing factor β does not fluctuate greatly. Under high load, the impact of time delay is greater. This is because the network is continuously transmitting a large amount of data. The

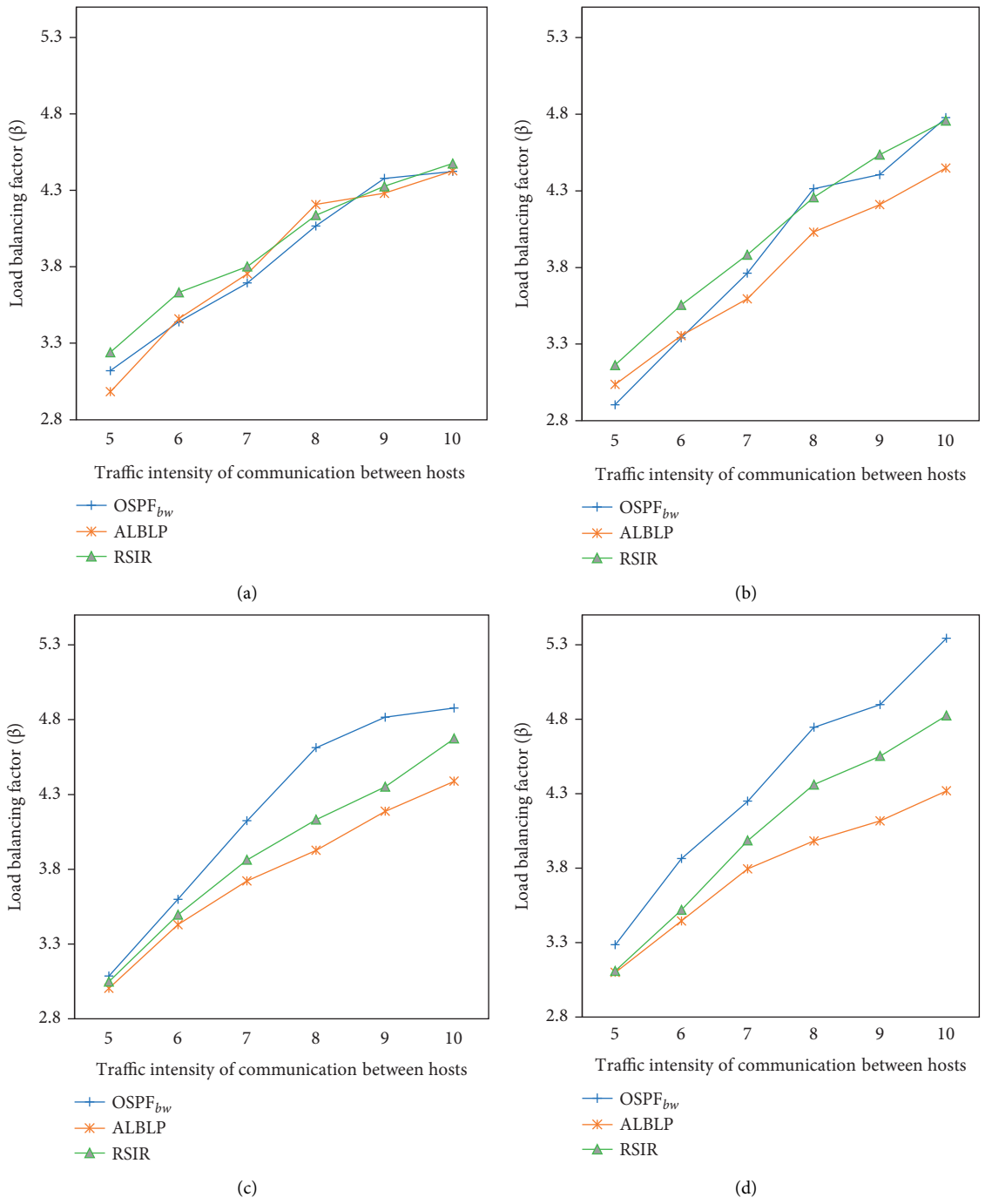

Figure 11: Load-balancing degree of ALBLP, OSPF, and RSIR under different traffic intensities and different Delay$_{cs}$. (a) Delay$_{cs}$ = 0. (b) Delay$_{cs}$ = 1. (c) Delay$_{cs}$ = 2. (d) Delay$_{cs}$ = 3.

greater the Delay$_{cs}$, the more the link information obtained by the controller lags, and the more difficult it is to make better routing decisions. If the impact of Delay$_{cs}$ cannot be solved as with the ALBLP algorithm, the network load-balancing factor β will be larger. The ALBLP algorithm uses predictive methods to ascertain the network link conditions in advance, and so the load-balancing factor β is better than that with the other two algorithms. In this experiment, the RSIR algorithm needs to retrain the Q-table and calculate the path between each host pair one by one after receiving the network link-state value. This process requires a great deal of training time, and so the network cannot make path changes in time, according to changes of network traffic, resulting in poor performance. Compared with the RSIR algorithm and the OSPF algorithm, the ALBLP algorithm has a maximum load-balancing improvement of 11.7% and 23.7%, respectively. It is worth noting that when the Delay$_{cs}$ is considerable, the RSIR and OSPF algorithms repeatedly caused

network crashes due to too much data being sent by the switch to the controller. This situation does not, however, occur in the ALBLP algorithm.

6. Conclusions and Future Work

This paper verified the influence of transmission delay between controllers and switches on network load balancing by conducting simulation experiments by setting different $Delay_{cs}$ and different network strengths. Thus, we built a network architecture based on the SDN application plane to solve the problem. Based on this construct, we designed the ALBLP algorithm to optimize network balance. The essence of the ALBLP algorithm is to use the LSTM model traffic-prediction method to offset the effect of time delay and better meet the requirements of real-time global load balancing. The experiment results show an 11.7% improvement over the RSIR algorithm in network load balancing, and 23.7% over OSPF. By comparing RSIR and OSPF, ALBLP can effectively solve the problem of transmission delay between controllers and switches under high-load network conditions. In subsequent work, we will continue to explore how to improve the efficiency of link-state prediction algorithms and how to optimize load-balancing strategies in large-scale networks.

Data Availability

The data supporting the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61861013), the major program of Guangxi Natural Science Foundation (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] Y. Yang, G. Lu, H. Zhao, and P. Li, "Overview of the application of deep learning in software defined network research," *Journal of Software*, vol. 31, no. 7, pp. 2184–2204, 2020.

[2] J. Yan, H. Zhang, Q. Shuai, B. Liu, and X. Guo, "HiQoS: an SDN-based multipath QoS solution," *China Communications*, vol. 12, no. 5, pp. 123–133, 2015.

[3] X. Kong, S. Shen, and L. Li, "A QoS routing scheme based on software-defined networking," *Journal of Computer Research and Development*, vol. 28, no. 2, pp. 102–106, 2018.

[4] Y. Bi, G. Han, C. Lin, Y. Peng, H. Pu, and Y. Jia, "Intelligent quality of service aware traffic forwarding for software-defined networking/open shortest path first hybrid industrial internet," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1395–1405, 2020.

[5] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[6] X. Tao, Z. Liu, and C. Yang, "An efficient network security situation assessment method based on AE and PMU," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 1173065, 9 pages, 2021.

[7] H. Zhong, J. Fan, J. Cui, Y. Xu, and L. Liu, "Assessing profit of Prediction for SDN controllers load balancing," *Computer Networks*, vol. 191, pp. 1–10, 2021.

[8] M. Yan, H. Yuan, Z. Li, Q. Lin, and J. Li, "Energy savings of wireless communication networks based on mobile user environmental prediction," *Journal of Environmental Protection and Ecology*, vol. 22, no. 1, pp. 206–217, 2021.

[9] M. Yan, S. Li, C. A. Chan, Y. Shen, and Y. Yu, "Mobility prediction using a weighted Markov model based on mobile user classification," *Sensors*, vol. 21, no. 5, p. 1740, 2021.

[10] B. Yang, S. Sun, J. Li, and T. Yan, "Traffic flow prediction using LSTM with feature enhancement," *Neurocomputing*, vol. 332, pp. 320–327, 2019.

[11] M. Yan, W. Li, C. A. Chan, S. Bian, C. L. I, and A. F. Gygax, "PECS: towards personalized edge caching for future service-centric networks," *China Communications*, vol. 16, no. 8, pp. 93–106, 2019.

[12] M. Yan, X. Lou, and Y. Wang, "Channel noise optimization of polar codes decoding based on a convolutional neural network," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 1434347, 10 pages, 2021.

[13] A. Azzouni and G. Pujolle, "NeuTM: a neural network-based framework for traffic matrix prediction in SDN," in *Proceedings of the 2018 IEEE/IFIP International Conference on Network Operations and Management Symposium (NOMS 2018)*, pp. 1–5, IEEE, Taipei, Taiwan, April 2018.

[14] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. L. 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.

[15] A. Filali, Z. Mlika, S. Cherkaoui, and A. Kobbane, "Preemptive SDN load balancing with machine learning for delay sensitive applications," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 15947–15963, 2020.

[16] D. Liang, J. P. Li, and C. Wang, "Applying LSTM to enable cache prefetching to optimize flow table update efficiency in SDN switches," in *Proceedings of the 7th International Conference on Information Technology: IoT and Smart City*, pp. 126–130, Shanghai, China, December 2019.

[17] J. Li, X. Chang, Y. Ren, Z. Zhang, and G. Wang, "An effective path load balancing mechanism based on SDN," in *Proceedings of the IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 527–533, IEEE, Beijing, China, September 2014.

[18] C. A. Wang, B. Hu, S. Chen, D. Li, and B. Liu, "A switch migration-based decision-making scheme for balancing load in SDN," *IEEE Access*, vol. 5, pp. 4537–4544, 2017.

[19] H. Li, G. Yang, H. Lu, X. Fu, and Z. Shen, "Research on SDN data center network elephant flow scheduling based on ant colony algorithm," *Journal of Computer Application Research*, vol. 36, no. 12, pp. 3837–3841, 2019.

[20] X. Hai, K. K. Tae, and Y. H. Yong, "Dynamic load balancing of software-defined networking based on genetic-ant colony optimization," *Sensors (Basel, Switzerland)*, vol. 19, no. 2, pp. 1–17, 2019.

[21] M. C. Nkosi, A. A. Lysko, and S. Dlamini, "Multi-path load balancing for SDN data plane," in *Proceedings of the 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*, pp. 1–6, IEEE, Mon Tresor, Mauritius, December 2018.

[22] 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.

[23] 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.

[24] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da 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.

[25] 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 ID 102865, 2021.

[26] 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. 1–26, 2020.

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

[28] B. He, J. Wang, H. Sun, and J. Liao, "RTHop: real-time hop-by-hop mobile network routing by decentralized learning with semantic attention," *IEEE Transactions on Mobile Computing*, p. 17, 2021.

[29] S. Ejaz, Z. Iqbal, P. Azmat 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.

[30] J. H. Cox, S. Donovan, R. J. Clarky, and H. L. Owen, "Ryuretic: a modular framework for RYU," in *Proceedings of the 2016 IEEE Military Communications Conference (MILCOM 2016)*, pp. 1065–1070, IEEE, Baltimore, MD, USA, November 2016.

[31] P. T. Kirstein, "European international academic networking: a 20 year perspective," in *Proceedings of the Terena Conference*, pp. 1–18, Rhodes, Greece, January 2004.

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