DEEP REINFORCEMENT LEARNING BASED OPTIMAL ROUTING WITH SOFTWARE-DEFINED NETWORKING

Yingjiang Liu[1], Jiaxin Zhang[1] , Yiling Yang[1], Ran Yang[1], Yuehui Tan[2]

[1]Beijing University of Posts and Telecommunications Beijing, China
[2]China Mobile (Suzhou) Software Technology Co., Ltd., Suzhou, China

ABSTRACT

An effective method for network congestion avoidance is to optimize the routing. We propose an optimal routing controlling that combines Deep Reinforcement learning(DRL) based policy optimization and Software Defined Network(SDN) based flow control. It uses DRL to learn the optimal network routing policy in different traffic patterns. SDN controller provides network traffic information for DRL agent to make decision, and implements flow routing control according to the decision. The results of simulation experiments show that our DRL-based route optimization has good performances in congestion avoidance.

KEYWORDS

Deep reinforcement learning, Software-Defined Networking, Routing optimization, congestion avoidance, Load balance

1. INTRODUCTION

In traditional communication networks, the quality of service (QoS) is often affected by network congestion. This is because most routing protocols in traditional networks use shortest path routing, which cannot prevent the occurrence of network congestion. To address network congestion issues, one approach is to upgrade hardware to enhance the processing capabilities of network equipment. However, a more economical and effective method is to optimize the network's routing strategies to avoid network congestion.

In traditional networks, methods to address network congestion include load balancing. However, due to issues such as uneven traffic density distribution and highly dynamic traffic changes in typical communication networks, the difficulty of achieving network load balancing is significantly increased. A traditional load-balancing solution is ECMP (Equal-Cost Multipath Routing). When there are multiple different equivalent paths between source and destination addresses within a network, ECMP-supporting network protocols can simultaneously use multiple paths for data traffic transmission between the source and destination addresses. While ECMP can help reduce network congestion to some extent, it cannot fundamentally solve the root causes of network congestion. This is because ECMP lacks awareness of the global traffic state within the network and therefore cannot effectively address congestion issues arising from uneven traffic distribution.

The emergence of the Software-Defined Networking (SDN) framework offers a new possibility for addressing the aforementioned issues. Unlike traditional network architectures, SDN advocates for the separation of the network data plane from the control plane. Communication

DOI : 10.5121/ijngn.2024.16401

between the data plane and the control plane can be achieved through an open protocol—the OpenFlow protocol. Switches based on the OpenFlow protocol not only perform standard data traffic forwarding and transmission but also upload real-time network status information to the controller in the control plane. The controller can collect and aggregate information uploaded by OpenFlow switches within its managed network area and formulate corresponding routing strategies and forwarding mechanisms based on the collected network status information. Compared with traditional networks, the SDN framework offers many advantages. One significant advantage is the ability to achieve global route control of the network via the controller. This means that we can control the routing policies and traffic distribution across the entire network from a holistic perspective, thereby addressing network congestion issues, including those caused by uneven traffic density distribution mentioned earlier. Furthermore, leveraging the SDN framework allows us to implement Network Function Virtualization (NFV), enhancing the flexibility and efficiency of network management and optimization.

Once we can grasp network status information and achieve network routing control through the SDN controller, what we need to do is analyze the collected real-time network status information and take corresponding routing control actions to achieve our objectives, such as traffic control. This analysis-decision process can be implemented either by an expert system composed of a series of rules or by precise algorithms made up of formulas. In this paper, we will use a Deep Neural Network (DNN) trained based on Deep Reinforcement Learning (DRL) to carry out this analysis-decision process.

Deep Reinforcement Learning (DRL) is considered the most suitable current machine learning method for application in network routing control. DRL originates from traditional Reinforcement Learning within machine learning methods and combines deep learning that employs deep neural networks. In recent years, it has achieved remarkable success in several domains, such as Atari video games and AlphaGo [29], defeating top human players in these fields. However, it is regrettable that DRL's success has so far been limited to a few areas and has not yet been sufficiently applied in network control, industrial control, and other fields. Reviewing successful cases of DRL, we have summarized some conditions necessary for its success across different application scenarios: an environment with clear rules and definitions, a system capable of providing accurate and timely feedback, and a reward function that can assist us in achieving our goals. We believe that network traffic control and routing decision-making meet these conditions. That is to say, using DRL to achieve intelligent network control and routing decisions is feasible. In this paper, we propose a DRL-based network routing strategy optimization scheme and demonstrate its effectiveness in achieving load balancing and avoiding network congestion.

In this paper, we first introduce the overall architecture of the network control system based on the SDN framework and DRL, along with the background of related technologies. Subsequently, we provide a detailed explanation of how to use DRL to train a Deep Neural Network (DNN) for analyzing network information to make optimal routing decisions. Finally, we validate the effectiveness of the DRL-based routing scheme in achieving network load balancing and congestion avoidance through comparative simulation experiments conducted across various network topologies.

2. BACKGROUND

2.1. Research Significance and Application Background

In recent years, due to the progress of network infrastructure, the rapid development of network communication technology and the large increase of network terminal equipment, the business

flow in the network has shown a rapid growth trend. At the same time, these fast-growing traffic has also brought new challenges to the existing communication network. The existing network traffic control scheme is not mature enough to effectively deal with the complex and changeable network conditions caused by the increasing network traffic. In order to meet this new challenge, a new intelligent network flow control scheme which can deal with complex network conditions is needed.

The hardware devices in the core of the Internet have experienced several generations of changes, but the main idea of the traditional network routing strategy is still used today. Whether it is distance vector protocol or link state protocol, the network relies on the classical shortest routing algorithm to calculate the forwarding path from the source to the destination, which leads to some specific nodes/links in the network will bear more load than other nodes/links. Therefore, when facing the network environment with explosive growth of traffic, those devices that bear more load are limited by their own processing capacity and are difficult to cope with excessive business requirements, which may lead to network congestion and the decline of network service quality. On the other hand, although operators can alleviate this problem to a certain extent by adding higher performance core routers and more physical links, this method of relying on hardware solutions to improve the core network performance, that is, simply increasing the router performance or the number of links will lead to a large-scale waste of resources in the actual network. In addition, the existing network routing algorithms often choose the optimal path based on local network information, which sometimes causes congestion in the global network. At the same time, some routing algorithms have the problem of slow convergence in large-scale networks, and so on. These shortcomings are caused by the factors of the routing algorithm itself. With the increasing scale of the network and the continuous growth of network services, the shortcomings of traditional routing algorithms are increasingly prominent. Therefore, it is necessary to optimize the routing strategy of network traffic through a reasonable network traffic control scheme to achieve load balancing in the network.

In recent years, the emergence and development of software defined Networking (SDN) architecture [1-4] provides us with a new means to achieve more flexible network traffic control. Different from the traditional network architecture, SDN adopts the idea of separating the network control plane from the data plane. The information interaction between the control plane and the data plane of the network can be realized through an open protocol - openflow protocol [5]. The switching device based on openflow protocol can not only forward and transmit ordinary data traffic, but also upload the collected real-time network status information to the controller of the control plane through openflow protocol. The controller can collect and summarize the network status information uploaded by the openflow switch in the network area it manages, and formulate the corresponding routing strategy and forwarding mechanism for the network according to the summarized information.

Compared with the traditional network architecture, SDN architecture has better flexibility. Relying on SDN architecture, we can realize network functions Virtualization (nfv) [6]. In addition, an advantage of SDN architecture is that it can use the controller to realize the flow control of the global network. This means that we can control the routing strategy and traffic allocation in the network from the overall perspective to solve the network congestion problem. When it is possible to master the network status information and realize the network flow control through the SDN controller, what we need to do is to analyze the collected real-time network status information and decide which routing strategy and forwarding operation to adopt to realize the flow control. This analysis decision process can be an expert system composed of a series of rules or an accurate algorithm composed of some formulas, but these methods are often specific to the specific network structure and cannot be applied to a variety of network

structures. In this paper, we propose a new method using deep reinforcement learning algorithm to implement the analysis decision process.

At present, the application of deep reinforcement learning is only limited to industrial automation, robot control, and gaming, and has not yet been sufficiently applied in network control. We reviewed successful cases of deep reinforcement learning and summarized three necessary conditions for its application scenarios: first, there needs to be an environment with clear rules and definitions; second, a reward function must be set for learning that can help us achieve our goals; and third, a training system that can receive correct and timely feedback is needed.

After rigorous analysis, we believe that the traffic control problem in the network meets the above conditions. Therefore, using deep reinforcement learning to achieve network traffic scheduling and routing decision optimization is feasible. To achieve this goal, this paper proposes a network traffic scheduling algorithm based on deep reinforcement learning. In the algorithm design, we first conduct a detailed analysis and modeling of the traffic scheduling optimization problem in the network, and design corresponding reward functions with the goal of improving the network's transmission capacity. We also design and implement a closed-loop feedback system that can be used for training and testing deep reinforcement learning models. Based on this, we propose a routing strategy optimization scheme based on deep reinforcement learning and SDN architecture. This solution can achieve intelligent network traffic control in networks with different topology structures. Based on the real-time traffic status information in the current network, the optimal global routing strategy is calculated, and the forwarding paths of all flows in the network are reasonably allocated to achieve load balancing. This solution can solve the congestion problem caused by uneven traffic distribution in traditional networks, improve network resource utilization and transmission capacity.

2.2. Overview of Related Technical Research

2.2.1. Deep Reinforcement Learning

Deep reinforcement learning originates from the combination of reinforcement learning (RL) in traditional machine learning [7-9] and deep neural networks (DNN) in deep learning (DL) [10-12], and has become an important research direction in the field of machine learning in recent years. In fact, the idea of combining reinforcement learning with neural networks is not a recent development. Many researchers have attempted to combine neural networks of different structures with reinforcement learning algorithms, but often for specific problems [13-16]. The success of deep reinforcement learning is marked by the Deep Q-Network (DQN) model proposed by Mnih et al. of DeepMind in 2013 [17]. DQN successfully achieved end-to-end learning of multiple Atari games using a deep Q-network composed of convolutional neural networks and fully connected networks, indicating that the DQN model is a universal deep reinforcement learning model. In recent years, deep reinforcement learning has been further developed in multiple fields, and deep learning techniques can be roughly divided into supervised, unsupervised, reinforcement, and hybrid learning based models. Some key topics in deep learning, such as transfer, federated learning, and online learning models, have also been applied in many fields[20]. The recent progress in DL covers the evolution and applications of basic models such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), as well as the latest architectures such as Generative Adversarial Networks (GAN), Capsule Networks, and Graph Neural Networks (GNN). In addition, deep reinforcement learning has also developed new training techniques, including self supervised learning, federated learning, and deep reinforcement learning, which further enhance the capabilities of deep learning models[21].

2.2.2. Research on Network Traffic Control Based on Deep Reinforcement Learning

In recent years, due to the rapid development of deep reinforcement learning and the emergence of various universal deep reinforcement learning models, deep reinforcement learning technology is increasingly being applied in fields such as industrial automation and robot control. In the field of network traffic engineering, research on network traffic control and routing strategy optimization based on deep reinforcement learning has also made some preliminary progress.

In 2018, Xu et al. proposed Experience driven Networking based on deep reinforcement learning [19]. Under the premise of using multi-path forwarding for traffic in the network, deep reinforcement learning methods are used to calculate the optimal allocation ratio on each path for traffic, in order to achieve load balancing and improve network utilization.

2.3. Main Research Content

The main research content of this article is to achieve network load balancing through intelligent network traffic scheduling. Focusing on the load balancing problem in data center networks, this article studies and analyzes the commonly used traffic scheduling algorithms in data center networks, and proposes a network traffic scheduling algorithm based on deep reinforcement learning. Specifically, the research content mainly includes the following aspects:

Firstly, this article introduces the relevant technologies and methods of deep reinforcement learning and SDN used in network traffic scheduling algorithms based on deep reinforcement learning, and investigates some common load balancing and traffic scheduling algorithms in current data center networks.

Afterwards, this article analyzed the load balancing problem and traffic scheduling problem in typical data center networks, studied the solution ideas of existing multi-path traffic scheduling algorithms in data center networks, and discussed the problems existing in traffic scheduling algorithms. Based on the above research and discussion, the network traffic scheduling problem was modeled, and the DQN algorithm in deep reinforcement learning was introduced to handle the optimization problem of network traffic scheduling. The deep neural network model structure, input-output characteristics, system reward function, etc. in the DQN algorithm were redesigned according to the specific requirements of the problem. In the design of DQN traffic scheduling algorithm, this paper proposes a traffic scheduling priority algorithm to address the issue of the order of all traffic when using the algorithm to select forwarding paths for global traffic. The algorithm determines a reasonable allocation method for the order of traffic path selection.

Finally, based on the proposed deep reinforcement learning based network traffic scheduling algorithm, this article designs a complete intelligent network traffic control system solution combined with SDN architecture. In order to verify the effectiveness of the proposed scheme, two simulation experiments were designed to test the algorithm's improvement in network transmission efficiency in two different structured simulation network environments. The comprehensive performance of the algorithm was evaluated by comparing it with various traffic scheduling algorithms.

3. DESIGN AND IMPLEMENTATION

3.1. Systems Modeling

In this section, the paper will first analyze and study the load balancing problem of data center networks based on the foundation of the previous chapter, and propose a dynamic optimization model based on network elephant flow scheduling for the problems to be solved. Then, it will analyze the existing data center network traffic scheduling technologies, discuss the problems with existing technologies and the parts that can be improved. Finally, based on the above discussion, the paper will provide a clear definition of the network traffic scheduling optimization model, the constraints of the model, and the optimization objectives. By formulating the load balancing problem of data center networks as an optimization problem, it paves the way for solving the problem using the deep reinforcement learning algorithm proposed in this chapter.

3.1.1. Analysis of Network Load Balancing Issues

Data center network traffic load balancing algorithms are divided into packet-level and flow-level based on traffic granularity. Flow-level algorithms aggregate packets into flows based on certain criteria, such as the 5-tuple in the packet header, and schedule traffic accordingly. This approach is suitable for specific applications and avoids packet reordering issues but may result in less effective load balancing compared to packet-level methods. Common flow-level algorithms include ECMP, VLB, Hedera's Global First Fit, and Mahout's Offline increasing first fit. Packet-level algorithms, in contrast, decide the forwarding path for each packet individually, offering finer granularity but potentially causing packet reordering. Solutions like SADP and SAOP have a moderate reordering effect but average load balancing, while round-robin routing can reduce link contention but leads to severe reordering. SDN networks facilitate flow-level load balancing due to their reliance on flow tables, making deployment easier. The intelligent traffic scheduling algorithm presented in this paper operates at the flow level, balancing the need for effective load balancing without causing packet reordering.

3.1.2. Analysis of Network Traffic Scheduling Algorithms

Utilizing multiple paths between source and destination nodes in data center networks can effectively achieve network load balancing. ECMP (Equal-Cost Multi-Path) is based on this idea, distributing different flows between source and destination nodes to multiple equivalent forwarding paths based on the hash result of the packet header. Although ECMP can achieve some load balancing effects in actual networks, a significant issue with ECMP is the potential for traffic collisions when different flows are assigned to multiple paths based on the hash result of the packet header. That is, two flows with high bandwidth requirements might conflict in the hash result and end up being assigned to the same forwarding path, leading to excessive link load on that path and causing network congestion. Figure 1 i1illustrates two types of traffic collisions that hash conflicts might cause. The network in the diagram is a 1Gbps Fat-tree structured network, with all link bandwidth limits set to 1Gbps, and the maximum transmission bandwidth of four TCP flows (ABCD) can reach 1Gbps. First, TCP flows A and B experience a local traffic collision at the aggregation switch Agg0 due to hash conflict, resulting in both flows only being able to utilize the 1Gbps bandwidth link of the core switch Core0. Then, due to the independent forwarding of packets by aggregation switches Agg1 and Agg2, they cannot foresee the traffic load on the upper-layer core switches, leading to a conflict between flows C and D on core switch Core2. In this example, if the path allocation is reasonable, all four TCP flows could achieve a transmission bandwidth of 1Gbps, such as forwarding flow A to Core1 and flow D to Core3. However, due to these two collisions, all

four TCP flows ultimately only reach a total bandwidth of 2Gbps, losing half of the throughput.

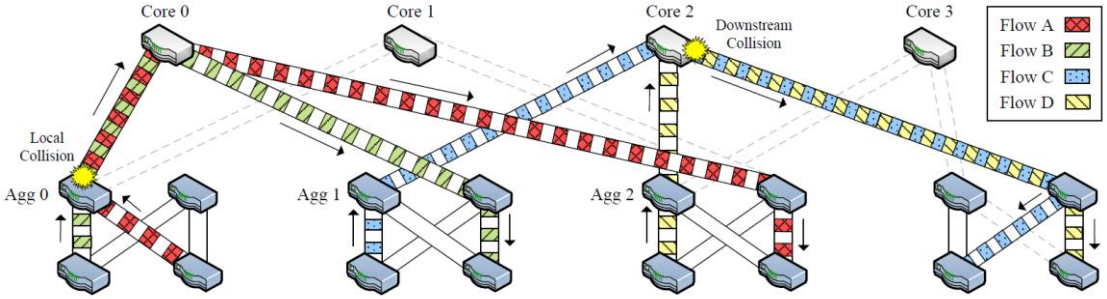

Figure 1. Two Types of Traffic Collision Issues When Using ECMP

3.2. Algorithm Design

This section introduces the algorithm design for a traffic scheduling algorithm based on deep reinforcement learning. It begins by explaining the deep neural network model used in the DQN (Deep Q-Network) algorithm and then details the method for generating optimal routing strategies based on the DQN algorithm's output. Subsequently, it addresses the issue of the order in which traffic is scheduled by proposing a flow scheduling priority algorithm. Finally, it provides a complete workflow of the algorithm.

3.2.1. Model Design in the DQN Algorithm

3.2.1.1. Deep Neural Network Architecture

The role of the Deep Q-Network neural network in DQN is to output the Q-values of all actions in the current state based on the input environmental state information, and then to select and execute the action with the highest Q-value among all actions. The structure within the deep neural network, including the number of hidden layers and the number of neurons in the hidden layers, is related to the scale of the input and output data. Generally speaking, the larger the scale and the higher the dimension of the input data, the more layers and neurons are needed in the hidden layers. If the input data is images or sequential data, a convolutional neural network with convolutional layers or a recurrent neural network with recurrent structures may be required.

3.2.1.2. Neural Network Input and Output

This paper innovatively proposes a method for representing network routing strategies in deep reinforcement learning with small-scale action outputs. This method breaks down the routing selection of all flows in the network into separate actions, where each action outputs the routing selection for one flow, and a series of actions sequentially determine the routing selections for all flows in the network. This output method controls the number of actions outputted each time within a very small range, with the number of actions equal to the number of optional paths for each flow. This approach avoids the negative impact of too many actions on the effectiveness of deep reinforcement learning. On the other hand, this method, which decomposes a single state input-action output process into a series of state input-action output processes, adopts the idea of dynamic programming. It breaks down a complex, large-scale problem into a series of consecutive simpler problems, while fully leveraging the advantages of reinforcement learning

in handling sequential decision-making problems. Experiments have proven that the Deep Q-Network neural network designed in this way shows good performance in both training and testing.

3.2.1.3. System Reward Function

In the design of this paper, the reward function consists of two parts: the episode reward after all actions are output in a turn, and the step reward after each individual action is output. The episode reward is used to evaluate all actions within a single turn, while the step reward is used to evaluate each action immediately after it is output. In fact, for most reinforcement learning problems, it is possible to optimize the strategy by providing only the episode reward. However, introducing step rewards can effectively accelerate the training process. Therefore, we use both types of rewards to evaluate the routing strategy of DQN.

Obviously, we can use the total bandwidth loss of all traffic in the network at the end of a round as the standard to evaluate all actions in that round. The definition of the episode reward is as follows:

$$\text{Episode\_reward} = \begin{cases} \min\left(\sum_{k=1,2..m}^{k} \left(\text{link}_k.\text{capacity} - \text{link}_k.\text{load} \right) \right) & \text{if flow\_bandwidth\_loss} = 0 \\ -\left(\text{flow\_bandwidth\_loss} \right) & \text{otherwise} \end{cases}$$

When the total bandwidth loss of the traffic in the network is 0, it means that the transmission rate of all flows in the network has reached the bandwidth demand. In this case, the Deep Q-Network neural network will receive a positive episode reward. The value of the episode reward is equal to the minimum available bandwidth among all network links. This is to encourage DQN to distribute the network load more evenly without losing traffic bandwidth. When the total bandwidth loss of the traffic in the network is not 0, the episode reward is the negative value of the total bandwidth loss. In other words, the more total traffic bandwidth loss caused by the actions output by the neural network in this round, the worse the episode reward given by the reward function.

Since we can only determine the episode reward at the end of a round, we cannot ascertain during the round whether each step will have a positive impact on the final outcome. Therefore, we cannot provide a positive step reward. However, after each action, we can determine whether the action has resulted in new traffic bandwidth loss. Thus, we can provide a negative step reward based on this. The definition of the step reward is as follows:

$$\text{Step\_reward} = \begin{cases} -\Delta\text{flow\_band\_widthloss} & \text{if flow\_bandwidth\_loss increase} \\ 0 & \text{otherwise} \end{cases}$$

$\Delta\text{flow\_band\_widthloss}$ represent the change in the total bandwidth loss after each action output. In a complete round, starting from the initial state, a step reward is obtained after each action output. When the round ends, the final action output will receive both the step reward for that action plus the episode reward.

3.2.1.4. Model Training and Parameter Configuration

After the neural network structure, input-output configuration, and reward function of DQN have been established, the training process for DQN can begin. The training process consists of

a large number of episodes, and at the start of each episode, we generate a set of random traffic demand data to simulate the network traffic demand state information at a particular moment in a real network. For specific network scenarios, such as data center networks, the optimal training approach is to use the actual traffic demand patterns from data center networks as the basis for generating traffic demand data, ensuring that the well-trained deep neural network performs effectively in real networks. However, due to privacy and security concerns, sometimes we cannot access publicly available real network traffic information. In such cases, we can also randomly generate traffic demand data within a specified range with equal probability. DQN may require a longer training time to achieve good results, but an additional benefit is that the trained Deep Q-Network neural network can provide superior routing strategies for all possible traffic demand states within the specified range, making the neural network's understanding and handling of various states more "comprehensive."

3.2.2. Flow Scheduling Priority Algorithm

We propose a flow priority algorithm for asymmetric networks: First, we determine all possible forwarding paths for the flows in the network. Assuming the expected bandwidth requirements for each flow in the network are known, and each flow selects its forwarding path with equal probability, we can calculate the expected load of each flow on each link in the network. By summing the expected loads of all flows on each link, we can obtain the expected total load for each link. After obtaining the expected total load on each link, we can compute the expected available bandwidth of each link based on its bandwidth capacity. The expected available bandwidth of a link can be either positive or negative. When the expected available bandwidth is negative, it indicates that the link is more likely to become the bottleneck in network communication. Next, we calculate the expected available bandwidth of each flow's possible forwarding paths. The rule for calculation is that the expected available bandwidth of a path is the minimum of the expected available bandwidths of the links on that path. After obtaining the expected available bandwidth for all the possible forwarding paths of each flow, we use the difference between the maximum and the second-largest expected available bandwidths of a flow’s forwarding paths as a criterion to evaluate the flow's priority. We believe that the larger this difference, the greater the gap between the flow's optimal and suboptimal path choices, and thus it is easier to identify the optimal forwarding path for the flow. In other words, the flow has a higher priority in the sequence of path selection. Based on this consideration, we calculate the difference between the maximum and second-largest expected available bandwidths of all possible forwarding paths for each flow. We then rank all flows from high to low according to this difference, and determine the sequence of flow selections when forwarding paths are chosen.

3.2.3. Algorithm Implementation Process

Before the algorithm starts execution, the basic information of the network needs to be determined: the link information between switches in the network, information about all possible flows in the network, the range of bandwidth demand for each flow, and the possible forwarding paths for each flow in the network. Based on this information, the flow scheduling priority algorithm proposed in this paper is used to compute the order in which paths are selected for the flows. During the algorithm's operation, this basic network information must remain consistent with the network information used during the DQN model training.

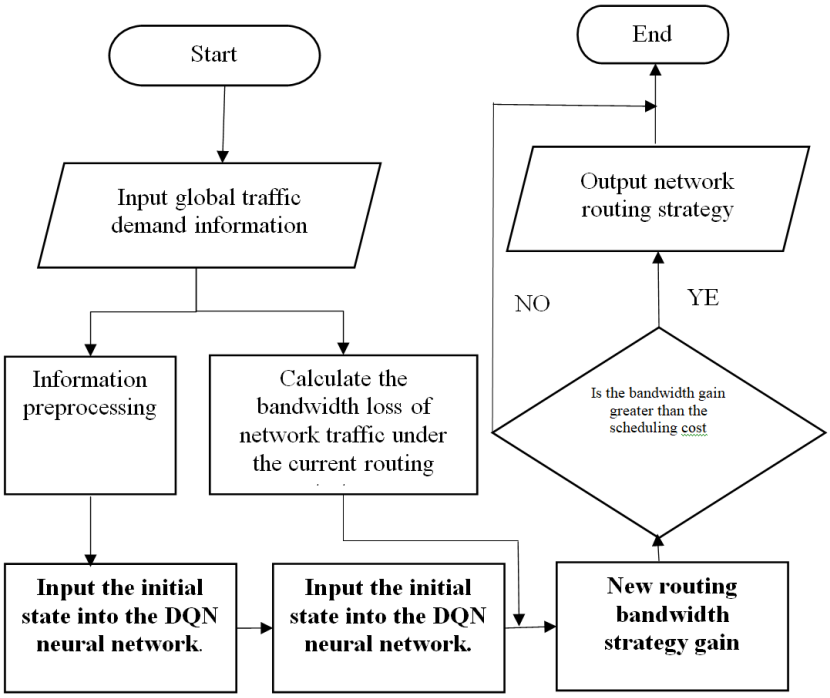

Figure 2. DQN-based network traffic scheduling algorithm process

Figure 2 shows the implementation process of the network traffic scheduling algorithm. The algorithm is executed under two conditions for global traffic scheduling: the first condition is when a new elephant flow is detected in the network, and the second condition is when it is executed periodically based on a timer. The purpose of periodically executing the algorithm is that when some flows in the network finish transmission, new available bandwidth is freed up on the links. At this point, executing the algorithm can reclaim this unused available bandwidth to increase the network throughput.

After the algorithm starts running, the global traffic demand information of the current network needs to be input. In an SDN architecture, this global traffic demand information can be provided by the SDN controller. Once the global traffic demand information is input, the algorithm preprocesses this information. The preprocessed state information is then used as the initial state input to the Deep Q-Network (DQN) neural network. The DQN outputs action Q-values to determine the forwarding path selection for the first flow and also provides the next state. This process of state input and action output is repeated until the forwarding paths for all flows in the network are determined, which results in the new traffic routing strategy.

While calculating the new routing strategy, the algorithm also computes the total traffic bandwidth loss in the network based on the current routing strategy. After obtaining the new strategy, the algorithm calculates the total traffic bandwidth loss for the new strategy and finds the difference between this and the current strategy. This difference represents the bandwidth gain of the new routing strategy. If the gain is greater than the cost of traffic scheduling, the algorithm will output the new routing strategy to update the forwarding paths for all flows in the network. Otherwise, it will maintain the current routing strategy and end, waiting for the next execution.

3.3. Intelligent Network Traffic Control Scheme Based on Deep Reinforcement Learning and SDN Architecture

The intelligent network traffic scheduling algorithm based on deep reinforcement learning proposed in this paper can determine the optimal global routing strategy in real-time based on the traffic demand in the network. This section will introduce how to deploy and implement the network traffic scheduling algorithm in a network with an SDN architecture.

3.3.1. System Architecture Framework

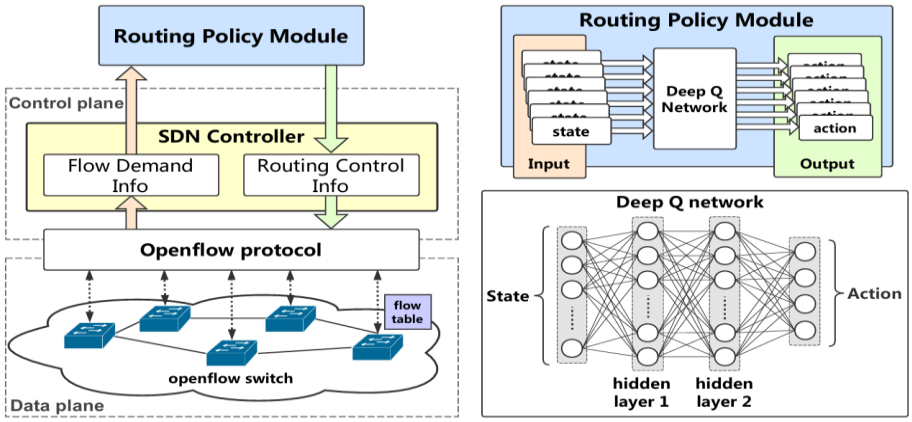

Figure 3. Intelligent Network Traffic Control System Scheme Based on SDN Architecture

Figure 3 shows the overall design scheme of the DQN-based traffic scheduling system. This scheme is based on an SDN architecture, where a DQN routing decision module, responsible for executing the traffic scheduling algorithm, is introduced into the SDN control plane. The DQN routing decision module contains a trained DQN neural network, input data preprocessing, and output information organization. Leveraging the programmability of the SDN controller, we can effectively monitor the traffic state in the network, and collect and aggregate real-time traffic demand information in the current network. This ensures the timeliness of the traffic scheduling algorithm's decision-making. The DQN routing decision module executes the traffic scheduling algorithm to derive the optimal routing strategy for the current network state, which involves determining the forwarding paths for all flows in the network. Through the SDN controller, commands are sent directly to update the flow tables on OpenFlow switches, enabling rapid traffic scheduling in the network and ensuring the timely execution of routing decisions.

3.3.2. System Operation Process

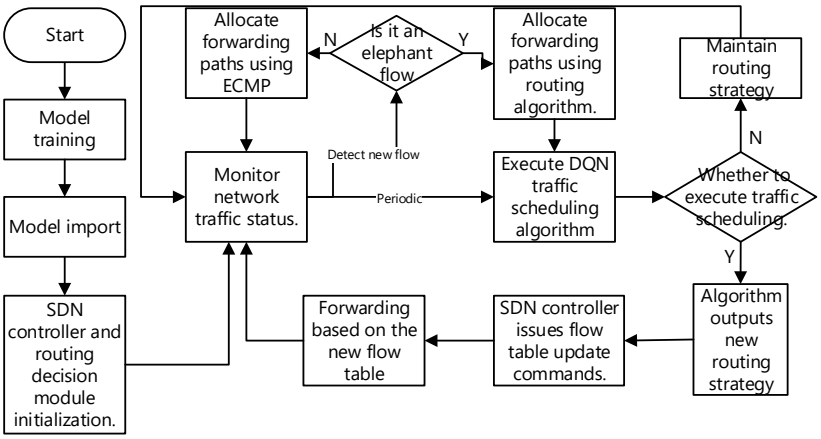

Figure 4. Operation Flowchart of the Intelligent Network Control System

Firstly, the top K shortest paths between two hosts in the network obtained through Yen's algorithm can be regarded as the optional forwarding paths for the flow between these two hosts. Once the basic network information is determined, the DQN neural network model in the DQN routing decision module can be trained based on the network structure and traffic patterns. After the above work is completed, initialize the SDN controller and the DQN routing decision module, and begin to detect and collect network traffic demand information.

Throughout the operation of the entire system, the SDN controller continuously monitors the real-time traffic status across the entire network by exchanging data with the data plane's switch devices. When a switch detects a new flow, it estimates the bandwidth requirements of the flow and determines whether the flow is an elephant flow. If it is not an elephant flow, it forwards the flow based on the ECMP routing method. If it is determined to be an elephant flow, it notifies the SDN controller. When the controller receives information that a new elephant flow has emerged in the network, it first uses default routing algorithms (such as Global First Fit, Offline increasing first fit, etc.) to select a forwarding path for the elephant flow, and then executes the DQN traffic scheduling algorithm.

The DQN-based traffic scheduling system can integrate with other routing algorithms, focusing on rerouting all elephant flows rather than individual path assignments. When a new elephant flow is added, the DQN algorithm may calculate a new routing policy that could be too costly to implement, so it defaults to the current policy and uses a predefined routing algorithm to assign a path to the new flow. The SDN controller periodically invokes the DQN algorithm based on network dynamics, with the interval depending on traffic volatility and computational capacity. After execution, the DQN module decides if the new routing strategy is beneficial enough to implement, and if so, the SDN controller updates the network's routing strategy to optimize performance.

4. EVALUATION

4.1. Experimental Simulation and Analysis

In order to evaluate the effectiveness of the DRL-based routing strategy optimization proposed in this paper, we conducted a large number of simulation experiments. In the rest of this section we will analyze the conditions and results of our comparative experiments in detail. We use

python to write all the code for the routing decision module in this scenario, including the construction of the DQN and the training and testing of the DQN. In addition, in order to accurately implement the network traffic control environment we described in the problem statement, we also wrote a stream-level network simulation environment using python to simulate the SDN controller in the SDN architecture to collect traffic information in the network. Modify the flow table to control the behavior of the traffic forwarding path. This not only ensures that our network flow control simulation environment can be perfectly matched with the DQN-based routing decision module, but also allows us not to care about the cumbersome data interaction process between the simulation environment and the decision module.

4.1.1. Network Simulation Scenario

An important feature of the intelligent network traffic scheduling algorithm proposed in this article is its universality for different networks, that is, the algorithm can not only be applied to data center networks with regular and symmetrical topology structures, but also to other networks with irregular and asymmetrical topology structures. To demonstrate the generality of the algorithm.

In the simulation experiment, we selected two different network topology scenarios to test the algorithm.

The first network scenario is a small-scale three-layer Fat tree simulation network used to simulate typical data center network environments. This network has 2 Pods, 8 hosts, 4 access layer switches, 4 aggregation layer switches, and 4 core layer switches. There are a total of 16 links between the switches in the network, with a bandwidth capacity of 1Gbps per link.
The second network scenario is a simulation network with an asymmetric structure generated using a random topology generation algorithm. The specific topology structure of the network is shown in Figure 4-3. The network has 8 hosts and 8 switches, among which the switches shown in gray in the network are access switches, and each access switch connects two hosts. There are a total of 14 links between switches in the network. In order to increase the complexity of the network structure, we no longer set the bandwidth capacity of all links to 1Gbps uniformly. Instead, we use two types of link bandwidth capacities, 1Gbps and 2Gbps, and randomly select one of the two bandwidth capacities as the link bandwidth capacity for each link in an equal probability manner. The randomly allocated bandwidth capacity of each link is shown in Figure 5.

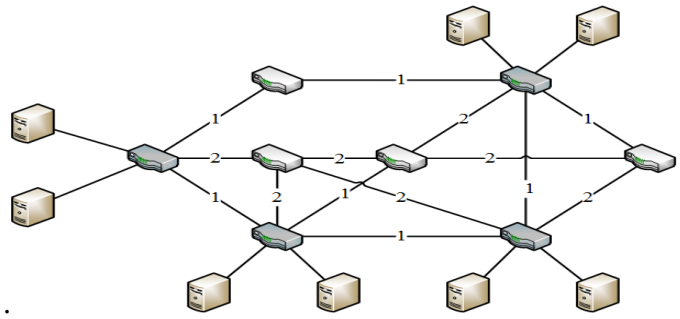

Figure 5. Operation flowchart of intelligent network control system

4.1.2. Network Traffic Mode

We define data packets with the same quintuple (source MAC address, destination MAC address, source IP address, destination IP address, Ethernet type) as a flow. Therefore, for both

Fat tree topology simulation networks and random topology simulation networks, due to the presence of 8 hosts in the network, the maximum possible number of flows in the network is 56. However, considering that the forwarding path of the flow generated between two hosts connected to the same access switch does not form a load on the links between the switches in the network, our intelligent traffic scheduling algorithm does not include the flow between two hosts connected to the same access switch. Therefore, we can conclude that the algorithm ultimately calculates the flow between the two hosts connected to the same access switch in the network. The maximum number of streams that can be scheduled is 48. In the Fat tree structure simulation network, there are four optional forwarding paths for flows between two hosts belonging to different Pods, and two optional forwarding paths for flows between two hosts belonging to the same Pod. In a random topology network, we use Yen's algorithm to calculate the first four shortest paths between two hosts and use them as optional forwarding paths for inter host flows

We simulate the traffic generated in the network based on the random traffic model. The bandwidth requirements of the flow are generated randomly within a specified range, and the random process follows a uniform distribution. The flow scheduled by the intelligent traffic scheduling algorithm is the elephant flow in the network, and we specify that all flows in the network with bandwidth requirements greater than 10% of the minimum link bandwidth capacity are elephant flows, so the minimum bandwidth requirements of the flows generated in the two simulation networks are 0.1Gbps. In order to test the performance change of the algorithm with the increase of the total traffic bandwidth in the network, we set the maximum bandwidth demand of the generated flow as a variable, which gradually increases from 0.1Gbps to 1Gbps, that is, when the maximum bandwidth demand is 0.1Gbps, the bandwidth demand of all the flows generated in the network is 0.1Gbps, while when the maximum bandwidth demand is 1Gbps, the bandwidth demand of the flows generated in the network is a random value between 0.1Gbps and 1Gbps. Because the random value follows a uniform distribution, according to the definition of the expected value of uniform distribution, it can be regarded as the expected bandwidth demand of all the flows generated in the network at this time 5.5Gbps.
Considering the network size, link bandwidth capacity and traffic bandwidth requirements, we set three traffic densities for the network simulation scenario to compare the effects of intelligent traffic scheduling algorithms under different traffic densities:

（1）random20： Each host sends traffic to other hosts in the network with a 20% probability
（2）random40： Each host sends traffic to other hosts in the network with a 40% probability
（3）random60： Each host sends traffic to other hosts in the network with a 60% probability

These three traffic densities correspond to the situation that the network traffic density is low, the traffic density is general and the traffic density is high.

After determining the bandwidth demand and traffic density of generated traffic, we will generate real-time network traffic bandwidth demand information in the simulation network environment based on this to simulate the real-time bandwidth demand state of the network when the intelligent flow control system calls the algorithm in the real network.

4.1.3. DQN Neural Network Model

In the intelligent traffic scheduling algorithm based on deep reinforcement learning proposed in this article, the DQN algorithm is used. In the simulation experiment, this article used Python and Tensorflow to implement the Deep Q-Network neural network model in DQN and related code for other parts of the algorithm. The basic DQN model used in the algorithm comes from an improved version published by DeepMind in Nature [18]. Compared with the original DQN

model [17], the Nature version of DQN adds a target Q-value neural network and an experience pool replay mechanism. In addition, this article also introduces two improved methods of DQN in the implementation of the algorithm: Dueling network and prioritized experience replay

Considering the scale of the simulation network environment, the DQN deep neural network structure constructed in the simulation experiment in this paper is a fully connected neural network with two hidden layers, and the number of neurons in each hidden layer is 120. We will use two identical deep neural networks with the above structure for training in DQN, one for calculating the current value function and the other for calculating the target Q value, in order to reduce the correlation between the target Q value and the current Q value in the Bellman equation and improve the stability of the algorithm.

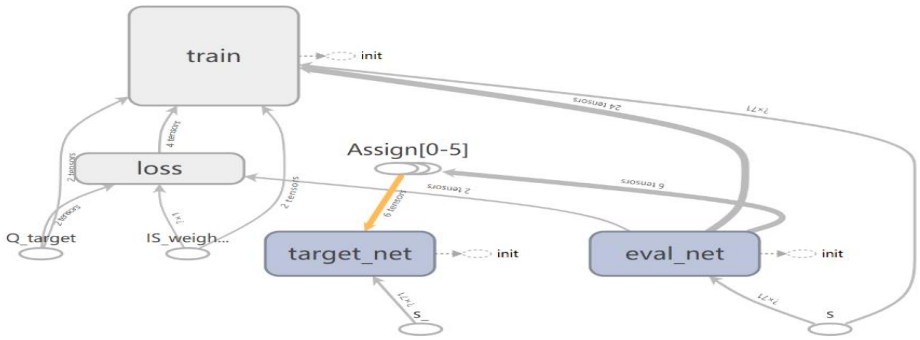

Figure 6. DQN model structure

The train module is the main body of the DQN algorithm, and during the training process, the algorithm updates parameters by calling the train module. Each time the parameters are updated, the algorithm first samples a batch size of data from the experience pool as training samples. The evale_net neural network is used to estimate the Q values under different states, and the loss is based on the difference between the training samples and the estimated values of the evale_net neural network to provide the corresponding loss function. The target_net neural network calculates the target Q value based on the training samples. After obtaining the loss function and target Q value, the evale_net neural network can update the network parameters based on the Bellman equation using gradient descent.

In addition to the structure of the neural network in DQN, there are some specific parameters that need to be set during DQN training. The specific values of these parameters in this simulation experiment are shown in the table below:

Table 1. DQN model training parameter settings.

hyper-parameters	**value**
learning rate	0.001
reward decay	0.9
ε-greed	0.9
ε-greed increment	0.000002
Q target replace interval	500
memory size	10000
batch size	32

The above parameter settings refer to the relevant parameters in the original DQN, and have been adjusted based on the traffic scheduling algorithm and simulation network environment in this article, as well as the effectiveness of the algorithm during actual training.

In both simulation network scenarios, we used the DQN model and parameter settings described above. Considering that there are three traffic density patterns in each simulation network scenario, in order to optimize the performance of the DQN algorithm, we trained three sets of DQN models for each simulation network, each corresponding to a traffic density pattern. For each set of models, we set the maximum number of rounds trained to 10 million. The hardware device used for model training is a computer with an eight core i7-6700 processor and 16GB of memory. Due to hardware limitations, there is no dedicated GPU to accelerate the training process. In the simulation experiment, the training time for each group of models is approximately 110-120 hours. After training, the DQN model takes an average of 4.2 milliseconds to output the global network routing strategy based on the input network traffic demand information each time.

4.1.4. Comparison of Evaluation Indicators and Algorithms

To evaluate algorithm performance in simulated networks, this paper employs throughput, network traffic bandwidth loss rate, and Bisection Bandwidth as metrics. Bisection Bandwidth, which measures the maximum transmission rate through a network-dividing section, is a key indicator of data center network performance, with higher values indicating stronger data transmission capabilities.

We compared our deep reinforcement learning-based traffic scheduling algorithm with four others: ECMP, VLB, Hedera's Global First Fit, and Mahout's Offline increasing first fit. Recognizing these algorithms' limitations in random topology networks, we included SPF shortest path algorithm for comparison in such scenarios.

Our evaluation method involves randomly generating flows within a specified bandwidth range for each algorithm in both Fat tree and random topology networks. The traffic density pattern sets the flow density. Each algorithm selects a forwarding path for each flow, and we calculate the resulting network throughput and bandwidth loss rate. We vary the bandwidth requirement range at a fixed traffic density to observe algorithm performance changes and average the results from 10,000 randomly generated flows to enhance accuracy.

4.2. Simulation Results and Performance Analysis

4.2.1. Fat Tree Structured NetworkFat-Tree

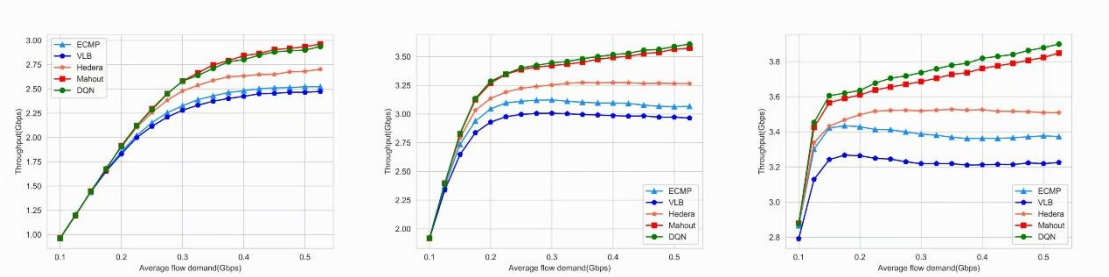

Figure 7. Throughput Comparison of Five Algorithms under Random20、Random40、Random60 Traffic Density in Fat tree Simulation Network

Figures 7 and 8 compare the DQN-based traffic scheduling algorithm's throughput and network traffic bandwidth loss rates with four other algorithms across three traffic densities: random20, random40, and random60. The x-axis represents the average traffic bandwidth demand, which increases from 0.1Gbps to 5.5Gbps, affecting throughput and loss rates.

As traffic demand rises, so does network throughput for all algorithms, eventually stabilizing due to the network's maximum transmission capacity. DQN and Mahout outperform ECMP, VLB, and Hedera in throughput across all densities. DQN's performance is close to Mahout at random20, improves slightly at random40, and exceeds it at random60. This suggests that at low traffic densities, the benefits of traffic scheduling are limited, and Mahout's Offline increasing first fit algorithm approximates optimal routing. However, DQN shows greater gains, especially as traffic density increases.

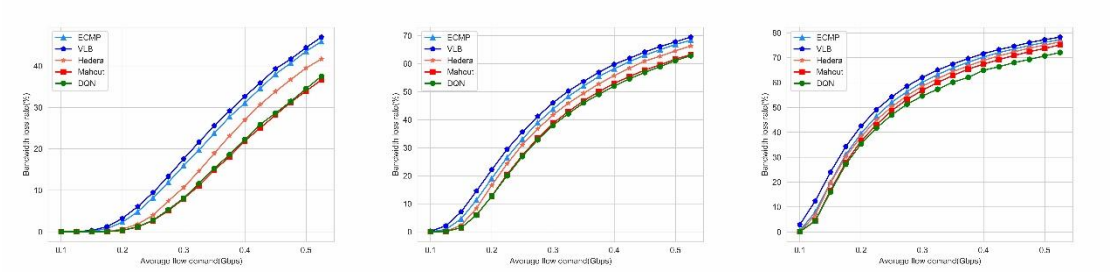

Figure 8. Comparison of Network Traffic Bandwidth Loss Rates of Five Algorithms under Random20、Random40、Random60 Traffic Density in Fat tree Simulation Network

Figures 8 show the network bandwidth loss rates of five algorithms. The lower the network bandwidth loss rate, the better the algorithm performance. Unlike network throughput, network bandwidth loss rate will continue to increase with the increase of average traffic bandwidth demand. From these three figures, it can be seen that the performance of the DQN based traffic scheduling algorithm is similar to the results in Figures 7.The algorithm can achieve lower network bandwidth loss rates than ECMP, VLB, and Hedera at all three traffic densities. At low traffic densities, the network bandwidth loss rate of the algorithm is similar to Mahout, while at high traffic densities, the algorithm can achieve a lower bandwidth loss rate than Mahout.

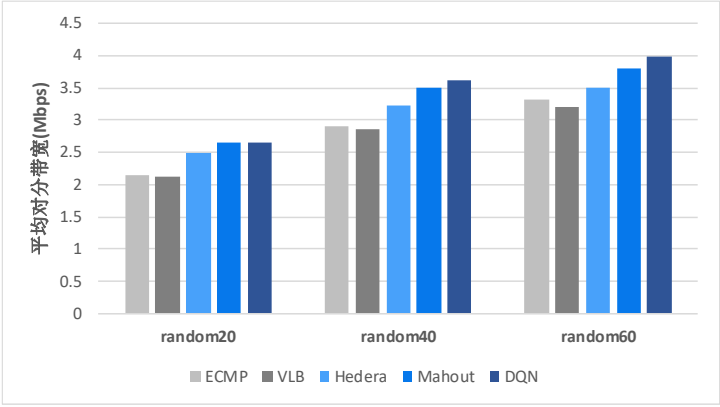

Figure 9. Comparison of Average Splitting Bandwidth of Algorithms under Three Traffic Modes

Figure 9 shows the average split bandwidth of five algorithms under three traffic modes. Due to the small scale of the Fat tree structure simulation network used in the experiment, we used the effective transmission bandwidth between two Pods in the network as an indicator of the

average split bandwidth. It can be seen that the average split bandwidth obtained by the DQN based network traffic scheduling algorithm at random 20 traffic density is approximately Mahout, higher than other algorithms. However, at random 40 and random 60 traffic densities, the average split bandwidth obtained by the DQN based network traffic scheduling algorithm is 3.2% and 4.8% higher than the best performing Mahout among other algorithms, respectively.

From the above experimental results, it can be seen that in a typical data center network with a Fat tree structure, the network traffic scheduling algorithm based on DQN can perform similarly to the Offline Increasing First Fit algorithm in Mahout at low traffic densities, and achieve better performance than the other four traffic scheduling algorithms in high traffic density network environments. This means that the intelligent network traffic scheduling algorithm based on deep reinforcement learning proposed in this article is more suitable for data center networks with high traffic density and heavy network load. Experimental results have shown that the intelligent network traffic control scheme designed in this article can meet this requirement.

4.2.2. Random Topology Network

The performance of six algorithms in a random topology simulation network is presented here, and the experimental results are shown in the following figure.

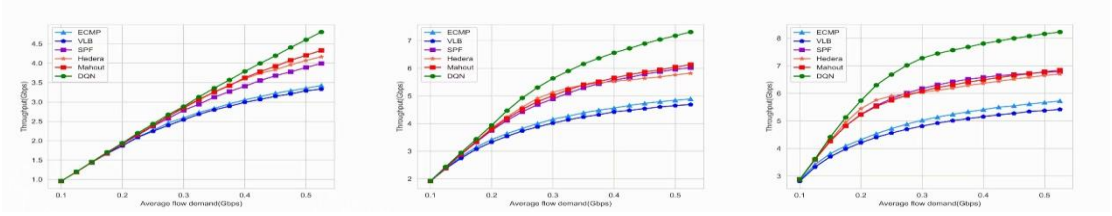

Figure 10. Comparison of throughput rates of five algorithms under Random20、Random40、Random60 traffic density in a random topology simulation network

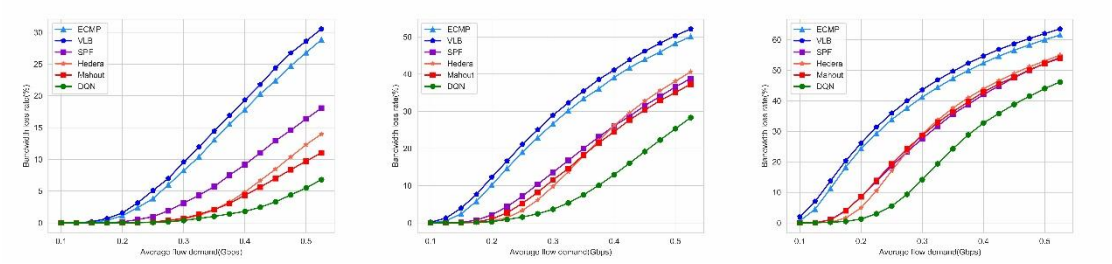

Figure 11. Comparison of Network Traffic Bandwidth Loss Rates of Five Algorithms under Random20、Random40、Random60 Traffic Density in Random Topology Simulation Network

Figure 10 compares the throughput of three different traffic density DQN based traffic scheduling algorithms with five other algorithms, while Figure 11 compares the network traffic bandwidth loss rate. The horizontal and vertical coordinates in each figure are the same as the corresponding figures 7 to 8 in the previous section.

In Figure 10, it can be seen that the throughput of the DQN based traffic scheduling algorithm is significantly higher than other algorithms under the three traffic densities. When the average traffic bandwidth requirement is the highest, the DQN traffic scheduling algorithm outperforms Mahout, which has the best performance among all other algorithms, by 10.97%, 19.16%, and

20.27% in network throughput, respectively. In Figure 11, the DQN traffic scheduling algorithm also showed a lower network bandwidth loss rate than the other five algorithms.

From the above results, in the simulated network with random topology, the DQN traffic scheduling algorithm is far superior to the other five algorithms in terms of both throughput and network bandwidth loss rate, and has a greater improvement compared to the Fat-tree structure network. This is because in the random topology network, multiple alternative paths between source and destination nodes are not all equivalent, which is different from the symmetrical relationship between paths under the standardized topology structure in data center networks. The relationship between the forwarding paths of flows in the random topology network is more complex, so it is difficult to obtain optimal routing strategies using either Hedera's Global First Fit algorithm or Mahout's Offline increasing first fit algorithm.

In addition to the above experiments, we also set up a set of experiments in a simulated network with a random topology structure. The results of this experiment are as follows:

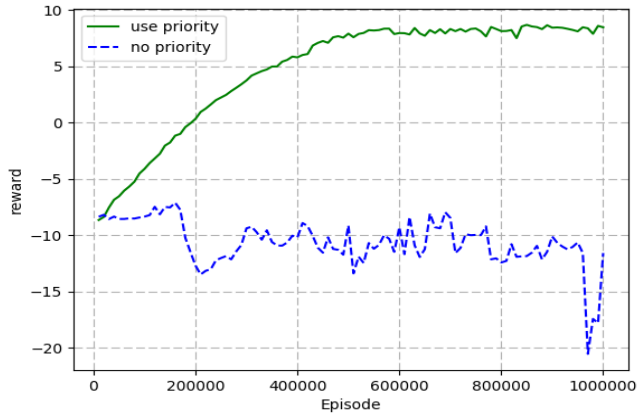

Figure 12. Comparison of training rewards between DQN models using priority algorithms and DQN models without priority algorithms

In the simulation experiment, in order to verify the necessity of the flow scheduling priority algorithm, we specially designed a set of comparative experiments. Under all other environmental conditions being the same, we compared the convergence of two sets of DQN models during the training process. One set of DQN models used a flow scheduling priority algorithm to determine the order of forwarding path selection for traffic, while the other set randomly arranged the order of forwarding path selection for traffic. We observed the convergence of the two sets of DQN models by the change in reward values obtained during the training process, and the results are shown in Figure 12.

From Figure 12, we can see that as the number of training epochs increases, the reward values obtained by the DQN model using the flow scheduling priority algorithm during training steadily improve and gradually converge; The reward values obtained by DQN models without using flow scheduling priority algorithms have been fluctuating up and down and difficult to converge. The experimental results indicate that for networks with irregular topology, if the intelligent network traffic scheduling algorithm based on deep reinforcement learning proposed in this paper is to be used in the network, the flow scheduling priority algorithm proposed in this paper needs to be used before training the DQN model to determine the order in which the DQN model selects forwarding paths for flows. Otherwise, it may result in longer training time or even inability to converge.

5. RELATED WORK

As DRL continues to achieve results in various fields, this powerful technology is getting more and more attention. As a matter of course, research on the application of DRL to traffic engineering (TE) is also rapidly increasing. In the study of [19], Zhiyuan Xu et al. proposed a DRL method based on Actor-Critic and prioritized experience replay to optimize traffic engineering to improve network utilization. In [18], the author used supervised learning on the TE to predict network traffic demand, and then used DRL to generate link weights for routing decisions. The authors in [15] propose a DNN architecture that incorporates CNN and determine whether the output route combination will cause congestion through the DNN. In addition to the research on the use of DRL to process TE in the traditional network, the optimization of the TE problem under the SDN framework is also a research hotspot. For example, et al. proposed an approximate algorithm for the TE problem in the SDN framework network in [2]. Recently, others have also proposed a method of network route optimization combined with SDN and DRL [Reference], but different from this paper, they use the network link weight as output to control the network route.

Due to the complexity of network control problems, most of the above methods based on DRL to achieve network control often use higher-dimensional output, which will undoubtedly increase the difficulty of training, and at the same time limit the scalability of the system. The use of combined action output simplifies the control of routing in the network, providing a new idea for network control problems based on DRL.。

6. CONCLUSIONS

In this paper, we propose a scheme to optimize the routing strategy of the network using the SDN framework and the DRL method. This scheme can dynamically adjust the routing strategy based on the traffic information in the communication network, thereby reducing the occurrence of network congestion. In the scheme, we use the DRL's classical method DQN to train the DNN used for network routing decision. At the same time, in order to reduce the dimension of the action space, we innovatively break down the way that one action allocates routes for all the flows in the network to each time. The action selects the route for a stream, and multiple actions complete all the route selections, so that the training method based on DQN can be realized. In addition, we also propose an algorithm that prioritizes all flows when routing decisions are made to speed up training. Finally, through simulation experiments, we prove that compared with the traditional load balancing ECMP, the DQN-based routing decision optimization scheme can more effectively achieve load balancing and avoid network congestion in asymmetric topological networks.

REFERENCES

[1] Fundation, O. N. (2012). Software-defined networking: The new norm for networks. ONF White Paper, 2, 2-6.

[2] Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications Surveys & Tutorials, 16(3), 1617-1634.

[3] 张卫峰. (2014). 深度解析 SDN: 利益, 战略, 技术, 实践. 电子工业出版社.

[4] 左青云, 陈鸣, 赵广松, 邢长友, 张国敏, & 蒋培成. (2013). 基于 OpenFlow 的 SDN 技术研究. 软件学报, 24(5), 1078-1097.

[5] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2), 69-74.

[6] ETSI, G. (2013). 001:" Network Functions Virtualisation (NFV). Architectural framework.

[7] Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.
[8] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
[9] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.
[10] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436.
[11] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.
[12] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.
[13] Cuccu, G., Luciw, M., Schmidhuber, J., & Gomez, F. (2011, August). Intrinsically motivated neuroevolution for vision-based reinforcement learning. In Development and Learning (ICDL), 2011 IEEE International Conference on (Vol. 2, pp. 1-7). IEEE.
[14] Cheng, J., Yi, J., & Zhao, D. (2005). Neural network based model reference adaptive control for ship steering system. International Journal of Information Technology, 11(6), 75.
[15] Lange, S., Riedmiller, M., & Voigtlander, A. (2012, June). Autonomous reinforcement learning on raw visual input data in a real world application. In Neural Networks (IJCNN), The 2012 International Joint Conference on (pp. 1-8). IEEE.
[16] Liu, D., & Wei, Q. (2014). Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems. IEEE Trans. Neural Netw. Learning Syst., 25(3), 621-634.
[17] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
[18] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.
[19] Xu, Z., Tang, J., Meng, J., Zhang, W., Wang, Y., Liu, C. H., & Yang, D. (2018). Experience-driven networking: A deep reinforcement learning based approach. arXiv preprint arXiv:1801.05757.
[20] Talaei Khoei, T., Ould Slimane, H. & Kaabouch, N. Deep learning: systematic review, models, challenges, and research directions. Neural Comput & Applic 35, 23103–23124 (2023). https://doi.org/10.1007/s00521-023-08957-4
[21] Mienye, I.D.; Swart, T.G. A Comprehensive Review of Deep Learning: Architectures, Recent Advances, and Applications. Information 2024, 15, 755. https://doi.org/10.3390/info15120755