

Critical Flow Rerouting Based on Policy Gradient algorithm

Qiong He^{1,*†}, Caixiao Ouyang^{1,†}, Chunzhi Wang^{2,†}, Lingyu Yan^{2,†}

¹Wuhan Vocational College of Software and Engineering, Wuhan 430025, China

²Hubei University of Technology, Wuhan 430068, China

Abstract

To address the problems that traditional load balancing algorithms are not ideal for real-time network optimization as well as easy to cause network interference and low performance. In this paper, we propose a Critical Flow Rerouting Based on Policy Gradient (CFRPG) algorithm, which can automatically select a few critical flows that have a decisive impact on network performance and reroute these flows to improve network performance. By combining Equal Cost Multi-path (ECMP) algorithm to forward most of the remaining traffic, CFRPG can achieve load balancing of the network. Experiments are conducted on the Mininet network simulation platform, and the experimental results show that only 10% of the total traffic needs to be rerouted to achieve the best network performance, and the CFRPG algorithm is more effective in improving network performance than other load balancing algorithms.

Keywords

Network traffic prediction, Load balancing, Network in SDN data center

1. Introduction

With the vigorous development of Internet technology and the continuous expansion of network scale, data centers have become an integral part of modern Internet infrastructure. However, the growth of data center scale has also brought the problem of exponential growth in data traffic within data center networks. The aggregation of massive network traffic has made data center network management quite complex, leading to a severe impact on overall network performance [1, 2]. Therefore, it has become a pressing challenge for current data center networks to reduce network congestion probability, achieve load balancing of network links, and improve data center network performance while maintaining constant transmission latency.

In order to address the performance challenges of data center networks, load balancing technology is widely applied in data center networks and plays a crucial role in optimizing their performance [3].

CITY2024: 2nd International Workshop on Computer Information Technologies in Industry 4.0, June 12–14, 2024, Ternopil, Ukraine

* Corresponding author.

† These authors contributed equally.

✉ qionghe@whvcse.edu.cn (Q. He); ouyangcaixiao@whvcse.edu.cn (C. Ouyang); chunzhiwang@hbut.edu.cn (C. Wang); yanranyaya2024@163.com (L. Yan)

ORCID 0009-0001-6698-0414 (Q. He); 0009-0004-9837-7136 (C. Ouyang); 0000-0002-6742-3644 (C. Wang); 0000-0002-8434-5473 (L. Yan)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

This effectively avoids network congestion and improves network stability and reliability. However, in real-life scenarios, network conditions change rapidly, and the network management system requires a certain amount of time to analyze the network state and make corresponding routing decisions. By the time the routing decisions are deployed to the underlying switches, network congestion may have already occurred. Therefore, analyzing and predicting network traffic, forecasting future traffic volume, and assessing the probability of link congestion can be beneficial. These predictions can then be incorporated into the corresponding traffic scheduling strategies, allowing for proactive load balancing and ultimately enhancing network performance.

2. Related works

Load balancing has always been one of the classic problems extensively studied in the field of networking. Depending on its application context, load balancing can be classified into two types: server load balancing and link load balancing [4]. Server load balancing typically involves setting up multiple servers and intercepting client requests to distribute data flows to available servers, thereby avoiding server overload and achieving load balancing objectives [5]. On the other hand, link load balancing is implemented by evenly distributing network traffic across multiple network links to improve data transmission rates and prevent single-point failures caused by overloaded individual links [6].

The literature [7] proposes the Equal Cost Multi-Path (ECMP) load balancing algorithm, which is currently the primary method used to address load balancing issues in data centers. The ECMP algorithm is based on the premise that network devices simultaneously maintain multiple equivalent paths and uses a hash algorithm to randomly distribute traffic for load balancing and improved network performance. However, this routing algorithm requires more network device resources to calculate and maintain multiple equivalent paths, increasing the burden on network devices. As network load increases and network fluctuations intensify, the load balancing effectiveness of the network may decrease.

The literature [8] introduces the Dynamic Load Balancing (DLB) algorithm, which achieves load balancing by dynamically updating the weights of servers and allocating client requests to servers with lower weights. Compared to static load balancing algorithms, DLB can achieve load balancing through local dynamic routing decisions. However, when selecting paths, the DLB algorithm only considers local states, and the adjustment of server weights is not flexible enough. This can result in a few links being heavily loaded while most links remain idle, leading to network congestion.

The literature [9-11] employs an SDN architecture and proposes a Load Balancing based on Flow Classification (LBFC) mechanism to address load balancing issues in fat-tree topology networks. The LBFC mechanism dynamically calculates flow classification thresholds based on network link status and traffic characteristics. It adopts different forwarding strategies for large and small flows, effectively improving load balancing performance. However, as network load increases, the decrease in dynamic thresholds may result in a large number of significantly different flows being classified as large flows.

Consequently, the LBFC algorithm randomly distributes these large flows, which can lead to the problem of fragmented remaining bandwidth.

The literature [12] presents a dynamic load-balanced path optimization algorithm (DLPO) that achieves load balancing between links by altering the flow transmission paths. It also utilizes a priority-based flow table update strategy to avoid packet loss caused by flow path changes, thereby improving network throughput and bandwidth utilization. However, due to the use of longer policy paths, this algorithm introduces increased transmission latency for flows.

3. The Load Balancing Algorithm based on Clustered Fuzzy Random Particle Grouping (CFRPG)

3.1. Problem Analysis and Modeling

The goal of network routing optimization is to control the distribution of traffic by configuring routing on the network topology, thereby helping Internet Service Providers (ISPs) optimize network performance and resource utilization. Network routing optimization algorithms can be broadly categorized into three types: flow-level, Flowlet-level, and packet-level routing algorithms. Among them, flow-level routing algorithms treat packets with the same source and destination addresses as a data flow, which can also be defined based on specific scenarios. Compared to packet-level and Flowlet-level routing algorithms, flow-level routing algorithms have a coarser granularity but can avoid issues such as packet misordering and difficulty in Flowlet partitioning. Existing routing optimization algorithms are mainly based on flow-level routing, which achieves load balancing on each link by periodically rerouting all flows in the network topology, thereby reducing network congestion. Although rerouting all flows in the network topology can achieve near-optimal network performance, it imposes a heavy computational burden on the SDN controller, which may cause severe network interference or service interruption, leading to a significant impact on user experience. Therefore, network operators are reluctant to adopt these existing routing optimization algorithms when deploying networks, and there is an urgent need to design a rerouting optimization algorithm that can achieve load balancing while reducing the routing computational burden and network interference.

The rerouting problem described in this paper can be formulated as follows: First, the SDN data center network, consisting of N nodes and M links, is abstracted as an undirected graph $G = (V, E, F)$, Where $v_i \in V$ represents node i in the graph, which corresponds to an SDN switch; $e_{i,j} = (v_i, v_j) \in E$ represents the link between nodes i and j in the SDN; In the SDN, $f_{sd} = (v_s, v_d) \in F$ represents the critical flow, where s denotes the source node of the critical flow, and d represents the destination node of the critical flow.

For convenience in the subsequent discussion, we define $c_{i,j}$ as the link capacity of link $e_{i,j}$; define $l_{i,j}$ as the traffic load on link $e_{i,j}$; define $D^{s,d}$ as the traffic request of the critical flow f_{sd} ; define $\sigma_{i,j}^{s,d}$ as the proportion of traffic requests of the critical flow f_{sd}

allocated for rerouting on link $e_{i,j}$; define $l_{i,j}'$ as the traffic load on link $e_{i,j}$ attributed to the remaining majority of flows that are forwarded using the default Equal-Cost Multi-Path (ECMP) algorithm. Based on the definitions provided above, the link utilization in the network can be represented as follows:

$$u_e = \sum_{\langle i,j \rangle \in E} \frac{l_{i,j}}{c_{i,j}}, \quad (1)$$

The maximum link utilization in the network, defined as the maximum value among the link utilizations, can be represented as:

$$U = \max_{\langle i,j \rangle \in E} u_e, \quad (2)$$

In the context of network load balancing problems, specific performance metrics in the network are typically used as the objective functions for optimization. In this paper, the optimization objective based on load balancing is to minimize the maximum link utilization in the network. Therefore, the problem of rerouting critical flows in an SDN data center network can be modeled as follows:

$$\min U, \quad (3)$$

In addition, the problem of rerouting optimization should also satisfy the following constraints:

$$l_{i,j} = \sum_{\langle s,d \rangle \in F} \sigma_{i,j}^{s,d} \cdot D^{s,d} + l_{i,j}', \quad (4)$$

$$\forall \langle i,j \rangle \in E, \quad (5)$$

$$l_{i,j} \leq c_{i,j} \cdot U \quad \forall \langle i,j \rangle \in E, \quad (5)$$

$$\sum_{\langle k,i \rangle \in E} \sigma_{k,i}^{s,d} - \sum_{\langle i,k \rangle \in E} \sigma_{i,k}^{s,d} = \begin{cases} -1, & \text{if } i = s \\ 1, & \text{if } i = d \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

$$0 \leq \sigma_{i,j}^{s,d} \leq 1 \quad \forall \langle i,j \rangle \in E, \quad \forall \langle s,d \rangle \in F, \quad (7)$$

Formula (4) represents the traffic load on link $\langle i,j \rangle$, which is composed of the traffic demand routed by the critical flows and the traffic demand routed by the default ECMP algorithm. Formula (5) represents the capacity utilization constraint of the link. Formula (6) represents the traffic conservation constraint for the selected critical flows.

3.2. Problem Analysis and Modeling

To address the aforementioned problem, this section presents a key flow re-routing optimization algorithm called CFRPG, which is based on policy gradient. This algorithm does not rely on any domain-specific heuristics. Instead, it utilizes the policy gradient algorithm to learn a key flow re-routing policy. The network performance serves as a reward signal that is fed back to the CFRPG agent, driving the agent to gradually learn better network performance strategies. By continuously observing the actual performance of past policies, the CFRPG algorithm optimizes its routing strategy for various traffic matrices over time. Once trained, the CFRPG algorithm efficiently and effectively selects a small set of key flows that have a significant impact on network performance for a given traffic matrix. By re-routing these key flows, it balances the link utilization in the network and achieves optimization of network performance.

As shown in Figure 1, this is the architecture diagram of the CFRPG algorithm designed in this section, which includes five components: SDN data center network environment, state space, agent, reward function, and action space. The agent's policy network consists of three layers of neural networks. The first layer is a convolutional layer with 128 convolutional kernels, each of size 3x3 and a stride of 1. The second layer is a fully connected layer with 128 neurons. The activation functions used in the first two layers are Leaky ReLU and ReLU, respectively. The final layer is a linear fully connected layer with neurons, where corresponds to all possible critical flows. The SoftMax function is applied to the output of the final layer to generate the probabilities of all available actions.

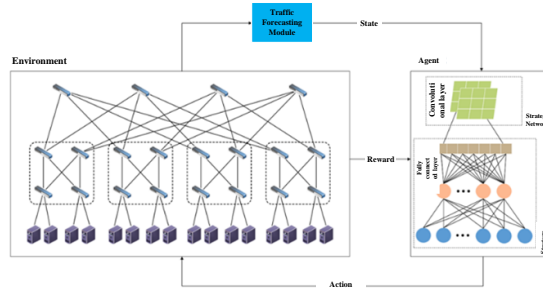


Figure 1: CFRPG algorithm architecture

To apply the CFRPG algorithm effectively to optimize the load balancing performance of SDN data center networks, it is necessary to determine the size of the state space and action space based on the current network environment. Additionally, the reward function needs to be designed according to the load balancing requirements of SDN data center networks to enhance the optimization capability of the CFRPG algorithm. The following sections will describe the state space, action space, and reward function of the CFRPG algorithm, providing a detailed explanation of how the CFRPG algorithm achieves load balancing in SDN data center networks.

Status space s_t : The intelligent agent of CFRPG takes the traffic matrix M_t , predicted by the traffic forecasting module at the next time step t as the input state s_t . This traffic matrix contains information about the traffic demand of each flow. Typically, the network

topology remains unchanged, so the network topology information is not included as input for the intelligent agent.

The action space a_t : The intelligent agent of CFRPG will automatically learn an optimal key flow rerouting policy function π from each input state s_t , and generate actions based on the policy function π , selecting K key flows for rerouting. Considering a network with N nodes and a total of $N \cdot (N-1)$ flows, the key flow rerouting optimization problem leads to a significantly large action space of size $C_{N \cdot (N-1)}^K$, making the learning process extremely challenging.

Therefore, in this paper, the intelligent agent of CFRPG is allowed to sample K different actions, denoted as $a_t^1, a_t^2, \dots, a_t^K$, simultaneously at each time step t . This reduces the size of the action space, defining it as $\{0, 1, \dots, (N \cdot (N-1))\}$.

The reward function r_t : After sampling K different key flows from the given state s_t , the intelligent agent of CFRPG reroutes these key flows and achieves optimal network performance by solving the rerouting optimization problem as described in Equation (3). The reward function formula for the CFRPG algorithm can be defined as follows:

$$r_t = \begin{cases} 1/U, & \text{when the constraints are satisfied} \\ 0, & \text{when the constraints are not satisfied} \end{cases}, \quad (8)$$

In this case, U represents the maximum link utilization. Since the objective of the rerouting optimization problem is to achieve more balanced load distribution on network links, the reward is greater when U is smaller. Therefore, in this paper, the reciprocal of the maximum link utilization is directly used as the immediate reward. For cases where the constraints are not satisfied, the immediate reward is set to 0.

3.3. Problem Analysis and Modeling

The key flow rerouting policy is represented by a neural network that takes the state s_t as input and outputs a probability distribution $\pi(a_t | s_t)$ over all possible actions. Since the intelligent agent of CFRPG samples K different actions, denoted as $a_t^1, a_t^2, \dots, a_t^K$, simultaneously and in an unordered manner for each state s_t , the random policy $\pi(a_t | s_t)$ parameterized by θ can be approximated as follows:

$$\pi(a_t | s_t; \theta) = \prod_{i=1}^K \pi(a_t^i | s_t; \theta), \quad (9)$$

The objective of training the CFRPG algorithm is to maximize network performance, specifically maximizing the expected reward $E[r_t]$ across various traffic matrices. Therefore, a reinforcement learning algorithm with a baseline $b(s_t)$ can be used to

optimize $E[r_t]$ through gradient ascent. The parameters θ of the policy function can be updated based on the following equation:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_{\theta} \log \pi(a_t | s_t; \theta) (r_t - b(s_t)) \quad (10)$$

Here, α represents the learning rate of the policy network. The baseline $b(s_t)$ represents the average reward obtained after sampling K different actions for each state s_t . It serves as a baseline in reinforcement learning to reduce the variance of gradients, improve training stability, and speed up learning. $(r_t - b(s_t))$ indicates how much better the reward based on the key flow rerouting policy is compared to the average reward for a given state s_t . If $(r_t - b(s_t))$ is positive, the policy parameters θ are updated in the direction of the gradient $\nabla_{\theta} \log \pi(a_{ik} | S_t; \theta)$ with a step size of $\alpha(r_t - b(s_t))$, thereby increasing the probability of the random policy $\pi(a_t | s_t; \theta)$. Otherwise, the magnitude of the random policy will decrease. The effect of Equation (10) is to strengthen the probability of actions that have historically produced better rewards.

To ensure that the intelligent agent of CFRPG adequately explores the action space during training and prevents premature convergence to suboptimal deterministic policies, this paper incorporates the entropy of the policy π into Equation (10). This improvement promotes exploration to discover better policies. Therefore, Equation (10) is modified to the following form:

$$\theta \leftarrow \theta + \alpha \sum_t (\nabla_{\theta} \log \pi(a_t | s_t; \theta) (r_t - b(s_t)) + \beta \nabla_{\theta} H(\pi(\cdot | s_t; \theta))) \quad (11)$$

Where H represents the entropy of the policy, and a higher entropy indicates that the policy has a more "even" distribution of probabilities for selecting different actions, which can improve the training effectiveness to some extent. The hyperparameter β controls the strength of entropy regularization.

4. Experimental simulation and result analysis

To validate the feasibility of the proposed CFRPG-based load balancing algorithm, this experiment will conduct simulation analysis based on a fat-tree data center network topology. The proposed routing scheme will be compared with ECMP and DLB algorithms. The performance of the CFRPG algorithm will be evaluated based on three metrics: throughput, load balancing degree, and average transmission delay.

4.1. Experimental Environment and Parameter Configuration

The experiment was conducted on the Ubuntu operating system using the Mininet network simulation platform to build the data center network. The open-source Ryu controller was employed as the controller for the entire network. Mininet is a lightweight network simulation platform that comes with built-in switches supporting the OpenFlow protocol. Using Python commands, a complete network topology can be constructed in Mininet, and the code developed on this platform can be easily transferred to real networks composed of physical hardware devices. Ryu is currently one of the most popular open-source SDN controllers, providing rich APIs for centralized network management and simplifying network administration. The CFRPG algorithm was implemented in Python using the TensorFlow framework and deployed on the Ryu controller. The specific experimental environment settings are presented in Table 1. A four-tier fat-tree topology architecture was utilized in the experiment, as shown in Figure 2. This topology consists of a core layer, aggregation layer, and edge layer, with a total of 16 terminal hosts and 20 switches. Each switch supports the OpenFlow protocol, and the link transmission between nodes is set to full-duplex to enable bidirectional data transfer and ensure reliable and stable transmission. Additionally, to ensure network performance and stability, the bandwidth of each link was set to 10 Mbps to meet the requirements of the experiment.

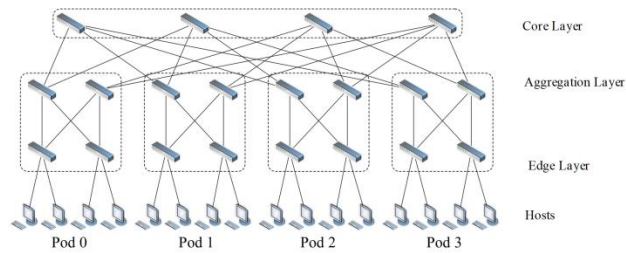


Figure 2: Four-tier fat-tree network topology

Table 1

Specific experimental environment

Software and Hardware Environment	Environment Configuration
CPU	Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz
Memory	16.00 GB
Operating System	Ubuntu 20.04
GPU	NVIDIA GeForce GTX1050Ti
Programming Language	Python 3.8
Python Libraries	Pandas、Matplotlib
Deep Learning Framework	TensorFlow 2.2.0
Network Simulation Platform	Mininet 2.3.0
SDN Controller	Ryu 3.16

Due to the confidentiality of business information in data centers, most data centers do not disclose their real traffic information. Therefore, this experiment uses the Iperf flow generation tool to simulate real network traffic based on the internal traffic characteristics of the data center network. Two traffic patterns, namely random mode and staggered mode, are used. Iperf is a network performance testing tool included in the Mininet simulation platform. By making simple code modifications to the internal files of Mininet, it can easily integrate with the Fat-Tree network topology and generate these two traffic patterns. Iperf is a network performance testing tool that is integrated with the Mininet simulation platform. It can conveniently generate random and staggered network traffic patterns that are in line with actual scenarios in the Fat-Tree network topology.

(1) Random mode (Random(p)) indicates that host i randomly sends data to host j with equal probability p . In this experiment, p is set to 0.25.

(2) Staggered mode (Staggered(p_e, p_p)) indicates that host i sends data to the upper-layer hosts belonging to the same edge switch with a probability of p_e , to the hosts belonging to the same pod with a probability of p_p , and to the hosts in other pods with a probability of $1 - (p_e + p_p)$. In this experiment, p_e and p_p are set to 0.2 and 0.5 respectively.

During the model training process, in this experiment, 70% of the generated network traffic in both modes is used as the training set, 20% as the test set, and 10% as the validation set. The model adopts the Adam gradient optimization algorithm with an initial learning rate manually set to 0.001. The learning rate is decayed by a factor of 0.96 every 500 iterations until it reaches the minimum value of 0.0001. Additionally, the entropy factor β is configured as 0.1, the number of epochs is set to 20, the batch size is set to 128, and the Dropout rate is set to 0.5. The settings of Dropout, batch size, and entropy factor β are aimed at preventing overfitting of the model. These hyperparameter values are set as the default values for implementing CFRPG.

To verify the feasibility of the algorithm, the CFRPG algorithm is mainly compared and analyzed with two other algorithms, ECMP and DLB. During the experimental process, each traffic model is repeated 5 times, and the average value of the experimental results is taken. Additionally, in order to evaluate the performance of the CFRPG algorithm, average throughput, load balancing degree, and average transmission delay are selected as the evaluation metrics for the experiments.

(1) Throughput (Φ): Throughput is typically used to measure the amount of data transmitted in a network per unit of time. It is calculated using the formula shown in Equation (12):

$$\Phi = \frac{r}{t}, \quad (12)$$

In the equation, Φ represents throughput, r represents the amount of data successfully transmitted within a certain time period, and t represents the time required for data

transmission. Throughput is one of the important metrics for measuring network performance, and a higher value indicates better load balancing effectiveness.

(2) Load Balancing Degree η : The load balancing degree in a network can be represented by the difference between the maximum and minimum link utilization rates among all links. The calculation formula is shown as (13).

$$\eta = \max_{\langle i,j \rangle \in E} u_e - \min_{\langle i,j \rangle \in E} u_e, \quad (13)$$

In the equation, η represents the load balancing degree, u_e represents the utilization of each link, and E represents the set of links in the network. When η is smaller, it indicates a more balanced load distribution among the links.

(3) The average transmission delay Ω : The average transmission delay refers to the average time taken for all data flows in the network to travel from the sender to the receiver. The calculation formula is shown as (14):

$$\Omega = \frac{\sum_{i=1}^n t_{ri} - t_{si}}{n}, \quad (14)$$

In the equation, Ω represents the average transmission delay, t_{ri} represents the end time of data flow received at the receiver, and t_{si} represents the start time of data flow sent from the sender. The average transmission delay is an effective measure to evaluate the network performance. A smaller value indicates a lower likelihood of congestion in network links and better load balancing effect.

4.2. Experimental Environment and Parameter Configuration

(1) Determining the Number of Critical Flows, K

By fixing the parameters other than the number of critical flows, K , this study investigates the impact of K on load balancing in SDN data center networks to determine its optimal value. Figure 3 illustrates the load balancing achieved by the CFRPG algorithm with varying numbers of critical flows, K , under two traffic patterns. Initially, K is set to 0, indicating the default ECMP algorithm for routing. The results show that there is significant room for improvement when using the ECMP algorithm for routing all network traffic in the SDN data center. As K increases, the network's load balancing, denoted by η , decreases rapidly, indicating that rerouting critical flows can effectively enhance network performance and greatly improve the load balancing of links. When $K = 10\% * N(N-1)$, the load balancing

index η is already less than 0.1, demonstrating that the CFRPG algorithm can achieve near-optimal load balancing performance by rerouting only 10% of flows. Therefore, in subsequent experiments, K will be set to $10\% * N(N-1)$ to train the CFRPG algorithm.

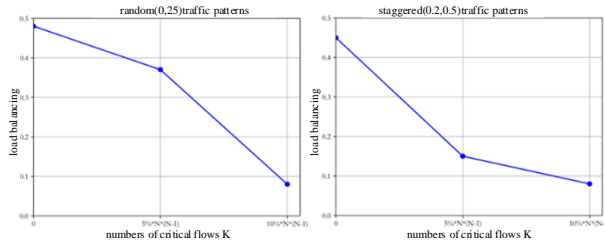


Figure 3: Impact of Different Numbers of Critical Flows, K, on Load Balancing

(2) Load Balancing Index

Figure 4 illustrates the comparison of load balancing index among ECMP, DLB, and CFRPG algorithms under two traffic patterns as the sending bandwidth varies. From the graph, it can be observed that CFRPG algorithm achieves a significantly lower load balancing index compared to DLB and ECMP algorithms. The load balancing performance of DLB algorithm is slightly better than that of ECMP algorithm. The reason behind this is that ECMP algorithm only evenly distributes the traffic across the links without considering the network's overall state information and the magnitude of traffic load. This can lead to network congestion, resulting in the highest load balancing index and the poorest load balancing performance. On the other hand, DLB algorithm makes routing decisions based on the current network state but lacks global optimization of the entire network. Therefore, it has a slightly higher load balancing index but better load balancing performance compared to the ECMP algorithm.

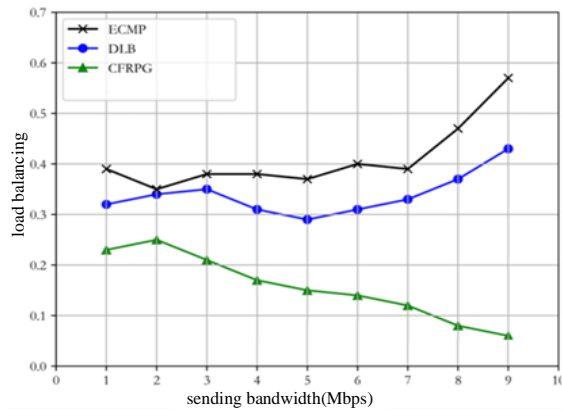


Figure 4: The graph compares the load balancing index of four algorithms under the Staggered (0.2, 0.5) traffic pattern.

Summary

This chapter proposed a key flow re-routing algorithm based on policy gradient, called CFRPG. The algorithm redefines the input state, action space, and reward function of the policy network. It can select a small number of key flows that have a significant impact on network performance and re-route these flows to improve network performance.

Additionally, the CFRPG algorithm incorporates a baseline function and policy entropy in the training process, enhancing the stability and learning speed of the CFRPG algorithm.

References

- [1] Zhang J, Yu F R, Wang S, et al. Load balancing in data center networks: A survey [J]. *IEEE Communications Surveys and Tutorials*, 2018, 20(3): 2324-2352.
- [2] Mann Z A. Allocation of Virtual Machines in Cloud Data Centers—A Survey of Problem Models and Optimization Algorithms [J]. *ACM Computing Surveys*, 2015, 48(1): 1-34.
- [3] Li Li, Wang Shuo, Huang Tao, et al. Overview of the four-layer load balancing technology of the data center network [J]. *Computer Engineering and Science*, 2022,44 (01): 48-59.
- [4] Neghabi A A, Navimipour N J, Hosseinzadeh M, et al. Energy-aware dynamic-link load balancing method for a software-defined network using a multi-objective artificial bee colony algorithm and genetic operators [J]. *IET communications*, 2020, 14(18): 3284-3293.
- [5] Zhou Yinglian, Liu Fu. Research on server load balancing technology [J]. *Computer and Digital Engineering*, 2010,38 (4): 11-14.
- [6] Zakia U, Yedder H B. Dynamic load balancing in SDN-based data center networks [C]// *Proc of the 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference*. Piscataway, NJ: IEEE Press, 2017: 242-247.
- [7] Chiesa M, Kindler G, Schapira M. Traffic engineering with Equal-Cost-Multipath: An algorithmic perspective [J]. *IEEE/ACM Transactions on Networking*, 2016, 25(2): 779-792.
- [8] Li Y, Pan D. OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support [C]// *Proc of the 12th IEEE International Conference on Communications*. Budapest, Hungary: IEEE Press, 2013: 1-5
- [9] Wang Jun, Wang Menglin, Wang Yue, et al. Load balancing scheme of SDN data center network based on flow classification [J]. *Computer Engineering and Application*, 2019,55 (24): 75-83.
- [10] Beshley M, Kryvinska N, Beshley H, Kochan O, Barolli L. Measuring End-to-End Delay in Low Energy SDN IoT Platform. *Computers, Materials & Continua*, 2022,67(1): 19-41.
- [11] Kochan O, Beshley M, Beshley H, Shkoropad Y, Ivanochko I, Seliuchenko N. SDN-based Internet of Video Things platform enabling real-time edge/cloud video analytics [C] // *Proc. of the 2023 17th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, 2023, Vol. 1: 1-5.
- [12] Lan Yuanliang, Wang Kuochen and Hsu Y H. Dynamic load-balanced path optimization in SDN-based data center networks [C]// *Proc of the 10th International Symposium on Communication Systems, Networks and Digital Signal*, 2016: 1-6.