A Link Load Balancing Algorithm Based on Ant Colony Optimization in Data Center Network

Shuqing Ma[1,2(✉)], Hong Tang[1,2], and Xinxin Wang[1,2]

[1] School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
masql0l8@163.com

[2] Chongqing Key Lab of Mobile Communications Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China

Abstract. With the continuous increase of types of services and data volume in data center, the traffic loads of some links are excessive, and how to balance the link load and ensure the quality of network service have become research hotpots. However, the traditional link load balancing mechanisms ignore the complexity of network and the Quality of Service (QoS) requirement of the flow when calculating the forwarding paths. Therefore, we propose a link load balancing algorithm based on Ant Colony Optimization (LLBA) in data center network. The algorithm redefines the heuristic function according to the number of elephant flows on the link and the real-time load of links, and updates the pheromones according to the path length. Then, the algorithm customizes the optimal path determination rule according to the path transmission delay and the real-time loads, so as to find a best forwarding path for the current flow under the multiple constraints including the path length, link load, and transmission delay. The experiment results show that, the proposed algorithm improves the link utilization and network throughput effectively, and also reduces the delay and delay jitter to some extent, as compared with the traditional mechanisms.

Keywords: Ant colony optimization · Data center network · Load balancing · Quality of service

1 Introduction

With the rise of cloud computing and big data technology, data center as the information infrastructure, gradually become the core of data computing, transmission, and storage. Data center contains thousands of interconnected servers, and various services such as MapReduce, network games, and search engines and so on. Due to the uneven business distribution and the randomness of data quantity changes, and the loads of different links which resulting in the decline of the performance of data center network [1]. In the meanwhile, with the diversification of Internet business, and users' requirements for service quality are increasing day by day. In order to improve resource utilization,

H. Gao et al. (Eds.): ChinaCom 2020, LNICST 352, pp. 474–489, 2021.
https://doi.org/10.1007/978-3-030-67720-6_33

decrease response time, and ensure service quality while maintaining stable and efficient network operation, thus, a flexible link load balancing strategy is urgently needed [2].

The flow characteristics are the basic basis of making the flow scheduling strategy. Generally, flows in data centers can be categorized into two types: elephant flow and mice flow [3]. Mice flows are often generated by interactive services, such as web page traffic, search queries, and so on, so mice flows take little resource and the number of bytes is numerous. Elephant flows are often generated by bandwidth-demanding services such as massive data synchronization, distributed computing, and video streaming, which live a long time and take advantage of lots of network resource. Since the number of elephant flows is less than 10%, but it takes up more than 90% of the link bandwidth, which is the main factor that cause link load imbalance. Due to elephant flows would take lots of network resources, so elephant flows scheduling is a challenging problem in data centers which may cause congestion. Thus, it is very feasible to design effective link load balancing strategies aim at elephant flows [4].

Equal-Cost Multipath Routing (ECMP) [5] is widely used for traffic scheduling in data center networks. It regards the available paths as equal cost paths and does not distinguish between elephant flows and mice flows. ECMP allocates the corresponding path statically to forward flows by calculating a hash value which typically from transport port numbers, IP addresses of the source and destination, and protocol. But a key limitation of ECMP is that, it is possible to allocate multiple flows to the same path, which causing elephant flows collision and bring the problem of link bottleneck. The appearance of Software Defined Networking (SDN) [6] provides an opportunity for fine-grained traffic scheduling for data center networks to distinguish the size of flows and perceive the global topology, thus making effective routing strategies to balance link load dynamically. There are so many traffic scheduling strategies based on SDN. Some methods which aim at mice flows, such as MiFlO [7]. In MiFlO, the scheduling algorithm which classifies the elephant flows and mice flows in servers and ensures the priority of mice flows. Additionally, there are some link load balancing strategies which aim at elephant flows, such as Hedera [8], Nimble [9], and Mahout [10]. Flow detection is performed in edge switches (Hedera, Nimble) or terminal hosts (Mahout), the basis of judging the size of flow is according to number of bytes transmitted per unit time. If the number of bytes transmitted per unit time that beyond the threshold, we regard this flow as elephant flow. And then calculate a light- load path for elephant flow. These load-balancing strategies calculate the forwarding path are according to the real-time link loads. To some extent, those solutions can extend network's capacity of data transmission, and also improve the utilization rate of the network links, as compared with ECMP.

However, the following problems remain: 1) network flows will occupy as much transmission bandwidth as possible in the links, which resulting in a large transient load fluctuation in the case of multiple elephant flows in a link. Thus, the instantaneous load information acquired by the controller does not completely reflect the load situation of the link. 2) The more links an elephant flow passes, the more link resources it occupies. It is unreasonable to calculate the forwarding path only according to the real-time load of the path, while ignoring the number of links that is unreasonable. 3) The current business requires high quality of network services. Such as multimedia information, which not only requires abundant bandwidth to transmit mass data, but also requires timeliness of data transmission [11]. Therefore, it is necessary to consider multiple factors in traffic scheduling strategy to ensure QoS.

Considering problems of all above, this paper proposes a load balancing algorithm based on ant colony optimization (LLBA) to balance the link load of network effectively and improve the quality of flow transmission. Ant colony optimization algorithm (ACO) with the features of self-organization, positive feedback, and heuristic search. The ACO algorithm is widely used in routing problems [12]. This paper redefines the parameters and operations. Firstly, the top-k shortest path set is selected as the basis of ants routing. Then, according to the number of elephant flow and the real-time residual bandwidth, two heuristic functions are designed to guide ants routing. Additionally, LLBA adopts global pheromone updating method. Finally, the real-time link load and path delay are used as the basis to judge the QoS of the path. After repeated iterations, the best forwarding path of flow is obtained.

The rest of the paper is organized as follows. Section 2 introduces the system architecture of the algorithm. We introduce the detailed design of the algorithm in Sect. 3. The performance evaluation is presented in Sect. 4. Finally, in Sect. 5 we draw our conclusions.

2 System Architecture

The system architecture of the proposed algorithm is showed in Fig. 1, which mainly consists of the following three modules.

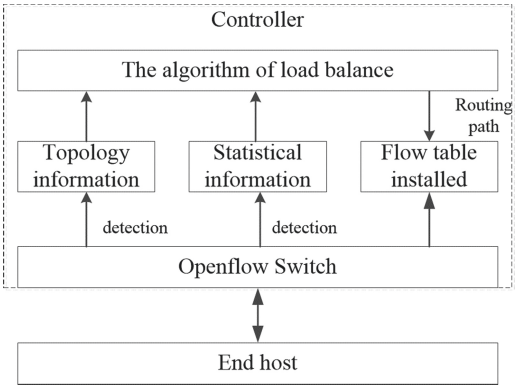

Fig. 1. The system architecture Algorithm.

2.1 Topology Information Detection

The function of this module is to get the network topology, and then obtain the shortest top-k paths between the source address and the destination address of flows. The controller gets the network topology by the link layer discovery protocol (LLDP). The specific steps are as follows:

Step 1: the controller establishes the OpenFlow channel and obtains the physical port information of the switch through the handshake principle.
Step 2: the controller periodically sends LLDP data packets to the switches, and then the switches which receive the LLDP data packets will sent the data packets to the

neighboring switches through the ports. After receiving the LLDP data packets, the neighboring switches will send a Packet-In message to the controller.
Step 3: the controller obtains the connection information of the switch according to the Packet-In message and the saved physical port information, thus getting the whole network topology information.

After obtaining the network topology information, the controller uses the Top-k shortest paths (KSP) algorithm to get the shortest path set according to the source switch address and the destination switch address of the flow.

2.2 Statistical Information Detection

The function of the module is to obtain the flow information and link status information, which mainly includes filtering the flows, counting the number of flows on the path, and calculating the real-time load and transmission delay of the path.

Filter the Flows. The controller polls each OpenFlow switch at a fixed time interval by sending OFPT_STATS_REQUEST message, and then queries and stores all of flow information and port statistics from the switches.

The elephant flows whose number of bytes transmitted per unit time exceeds the threshold that was preset in the algorithm. And we record and store the elephant flows according to the flow information. And we can distinguish the elephant flows and mice flows by formula (1).

$$\varphi = \frac{b_{t_2} - b_{t_1}}{(t_2 - t_1)B_{max}} \tag{1}$$

b_{t_1} and b_{t_2} respectively represent the number of bytes received by the switch at time t_1 and t_2, B_{max} is the maximum available bandwidth of the link. In this paper, we assume that when φ of a flow exceeds 10%, the flow is considered to be an elephant flow. And the flow needs to be dynamically scheduled [8].

Count the Number of Elephant Flows on the Path. The statistical information detection module counts the number of elephant flows on each link in real time. Assuming that the p_{th} path contains n links, and there are Num_i elephant flows on the i_{th} link, the number of elephant flows on the p_{th} path is shown as (2).

$$Num = max(Num_1, Num_2, \cdots, Num_n) \tag{2}$$

Compute the Real-Time Load of Path. The statistical information detection module monitors the traffic information both of each port and link in the network in real time. So the real-time load of path can be shown as (3).

$$Load = max(Load_1, Load_2, \cdots, Load_n) \tag{3}$$

Compute the Transmission Delay of Path. The transmission delay of path can be calculated by the delay of the LLDP packet and Echo packet.

First, as shown by the red arrow in Fig. 2, the controller sends a LLDP packet with timestamp to Switch_A, and instructs Switch_A to forward the packet to Switch_B. After receiving the data packet, switch B does not have a flow entry matching the data packet in the flow table, so the data packet is encapsulated in the Packet-In message and be send to the controller. The controller parses the sending time-stamp of the LLDP Packet from the Packet-In message. The LLDP Packet $delay_1$ can be obtained by subtracting the time stamp from the current system time. Similarly, the reverse $delay_2$ is shown by the blue arrows.

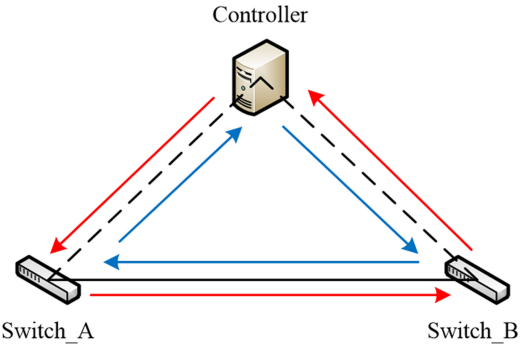

Fig. 2. Delay calculation of topology. (Color figure online)

Then, the controller sends an Echo_request message with the timestamp to Switch_A, and the controller parses the Echo_reply returned by Switch_A. The timestamp is subtracted from the current system time to obtain the round-trip time difference $delay_a$ between the controller and Switch_A. Similarly, the round-trip time difference $delay_b$ from the controller to switch B is obtained.

Then, the i_{th} link transmission delay between Switch_A and Switch_B can be calculated by (4).

$$Delay_i = (delay_1 + delay_2 - delay_a - delay_b)/2 \tag{4}$$

So the total transmission delay of the path is shown as (5).

$$Delay = \sum_{i=1}^{n} Delay_i \tag{5}$$

Flow Table Installed. Based on the topology and statistical information obtained previously, this module calculates the forwarding path for elephant flows, and then sends the new flow tables to the OpenFlow switches through the OFPT_FLOW_MOD message. The switches transmit the elephant flows according to routing information in the flow table.

3 Algorithm Design

Ant colony optimization (ACO) is a bionic algorithm that simulates ant foraging [13]. Each ant releases pheromones along the way of foraging, and the pheromones always evaporate with time. The higher pheromone intensity of path, the closer they get to the food. Other ants can perceive pheromones and tend to choose the path that contain more pheromones. Therefore, these collective behaviors of ants that present a positive feedback mechanism. The more ants pass on a certain path, the more likely the latter ants to choose the path, and finally get a closest path to the food.

The ACO algorithm with the characteristics of distributed computing, information positive feedback, and heuristic search which can solve the combination optimization routing problem [14]. In this paper, we propose a link load- balancing algorithm LLBA based on ACO. The algorithm customs the new best path determination rule. And we find the optimal transmission path for the current flow under the multiple constraints of path length, link-load, and delay, so as to achieve the link load-balancing of each link, and ensure the quality of network service.

3.1 Path Selection Strategy

There are many reachable paths between the two hosts in data center, and it would take many resources to find the optimal path from all of the reachable paths. If the more elephant flows pass through, the more resources are occupied. Therefore, in this paper, firstly, the algorithm obtains the shortest top-k paths to form the path set $SP = \{SP_1, SP_2, \cdots, SP_k\}$ from the information statistics module, and then the ant chooses an optimal path in the path set. The method of all ants select next site is roulette, and the calculation formula is given as (6).

$$p_i^j(t) = \frac{[\tau_i(t)]^{\alpha} * [\eta_i^1(t) * \eta_i^2(t)]^{\beta}}{\sum_{s \in allowed_k} [\tau_s(t)]^{\alpha} * [\eta_s^1(t) * \eta_s^2(t)]^{\beta}}, i \in allowed_k \tag{6}$$

M is the number of ants, $p_i^j(t)$ is the probability that the j_{th} ant chooses the i_{th} path at time t, and $allowed_k$ denotes the optional paths that ants can choose. $\tau_i(t)$ is the pheromone intensity on i_{th} path, $\eta_i^1(t)$ and $\eta_i^2(t)$ are the heuristic functions of path. And α and β are the weight values of pheromone and heuristics function respectively, and representing the relative influence of each factor. The higher the pheromone intensity and the value of the heuristic function, the greater the probability that path can be selected.

3.2 Heuristic Function

In order to solve the load imbalance problem, the path with lighter load should be preferentially selected. However, in actual network, the network flows will preempt the larger transmission bandwidth in the links as much as possible, resulting in large transient load fluctuation under the condition of multiple elephant flows on the link. The instantaneous load information obtained by the controller at a certain time cannot exactly reflect the load situation in the current state of the link. Therefore, the algorithm

estimates the path load by the number of elephant flows on the path, and proposes the heuristic function 1 of path as shown in (7), and then proposes the heuristic function 2 according to the real-time link load shown as shown in (8).

$$\eta_i^1(t) = \frac{1}{Num_i + 1} \tag{7}$$

$$\eta_i^2(t) = 1 - Load_i / B_{max} \tag{8}$$

Num_i is the number of elephant flows on the i_{th} path, $Load_i$ is the real-time load on path. And the larger value of $Load_i$ and B_{max}, resulting in the heavier load of the i_{th} path. The smaller the heuristic function of path, the less likely it is that this path be chosen by the ants.

3.3 Pheromone Update Strategy

After completing the path search, ants would release pheromone along the path, and the pheromone also would evaporate automatically with time. This process is called as pheromone updating.

The algorithm adopts global pheromone updating, that is, only update the pheromone increment on the optimal path after all ants complete the path selection and obtain the path solution set $AP = \{AP_1, AP_2, \cdots, AP_m\}$. And pheromone updating is defined as (9) and (10).

$$\tau_i(t+n) = (1-\rho) * \tau_i(t) + \Delta\tau_i(t) \tag{9}$$

$$\Delta\tau_i(t) = \begin{cases} \frac{Q}{Len_i} & if \quad (i = bestpath) \\ 0 & else \end{cases} \tag{10}$$

The change amount of pheromone intensity on the i_{th} path is calculated by (9), including natural volatility and pheromone increment. ρ is the speed of pheromone volatilization, and $\Delta\tau_i(t)$ is the amount pheromone increment. The pheromone increment is calculated by (10), where Len_i is the length of the best path, and Q is the constant total amount of pheromone released by the ant colony.

The purpose of the LLBA is to balance the network load and guarantee the transfer quality of flows. The delay is an important factor to measure the transfer quality of the flows. In the complex network environment, the path with the least load does not necessarily with the minimum delay, and the delay obtained in real time with randomness. To find the optimal solution, the algorithm takes the real-time load and delay of the path into consideration when choosing the path. So the path state function is proposed as (11). And μ is the weight of delay. According to (11), the smaller the path state value is, the better QoS of the path gets.

$$Cost_i = \mu * delay_i + (1-\mu) * Load_i \tag{11}$$

Thus, the best path with the minimum cost is shown as (12).

$$bestpath = \underset{i}{argmin}(Cost_1, Cost_2, \cdots, Cost_m) \tag{12}$$

3.4 Algorithm Description

The algorithm description is shown as Table 1.

Table 1. The algorithm description of LLBA.

LLBA: Link load-balancing algorithm based on ACO
Initialize parameters
Calculate the k_shortest_path
While F<Fmax
for each ant m
ant choose path from k_shortest_path
AP.append(path)
Selectbestpath(AP)
Update pheronmone
end while
for each ant m
ant choose path from k_shortest_path
AP.append(path)
selectbestpath(AP)
Print bestpath

Firstly, we initialize the basic parameters of the algorithm, and obtain the path set SP which consists of the shortest top-k paths, according to the source address and the destination address of the flow. Secondly, each ant selects a path from the path set SP according to (6), get the path set AP after all the ants complete the routing. Then, the best path is selected from the path set AP according to (12), and update the pheromone on the path by (9) and (10), the ant colony restarts the next search. Finally, when the number of iterations of the algorithm reaches a certain value, the optimal path is output.

4 Performance Evaluation

4.1 Proof Experiment

It is mentioned in the algorithm that the network flows will preempt the larger transmission bandwidth in the link as much as possible, resulting in a transient load fluctuation under the condition of multiple elephant flows on the link. The instantaneous load information obtained by the controller at a real-time cannot fully reflect the load

situation in the current state of the link. To demonstrate this, we used the topology shown in Fig. 3 to simulate the confirmatory experiment. On the 0th second, Host1 sends the first big flow to Host3 by using the Iperf command. On the 20th second, Host2 sends the second big flow to Host4. The monitoring time lasts for 60 s. Note that all links bandwidth of hosts-switches and inter-switches in the topology are set to 1Gbps because of the limited switching ability on a single machine. The experimental results are shown as Fig. 4 and Fig. 5.

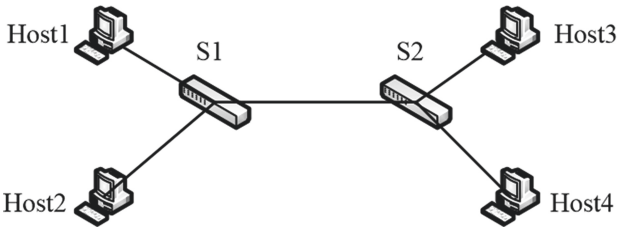

Fig. 3. Confirmatory experiment.

As shown in Fig. 4, between 0 and 20 s, the first flow takes up more than half of the bandwidth of the link. At the 20th second, the second flow is generated, and two flows begin to compete for bandwidth resources, the real-time transmission rate of the first flow gradually decreases. The real-time load of links between S1 and S2 is shown in Fig. 5, we can see that as the number of flows increases, the real-time load of links does not increase significantly, but fluctuates constantly. Therefore, the load of links cannot be determined completely according to the real-time load.

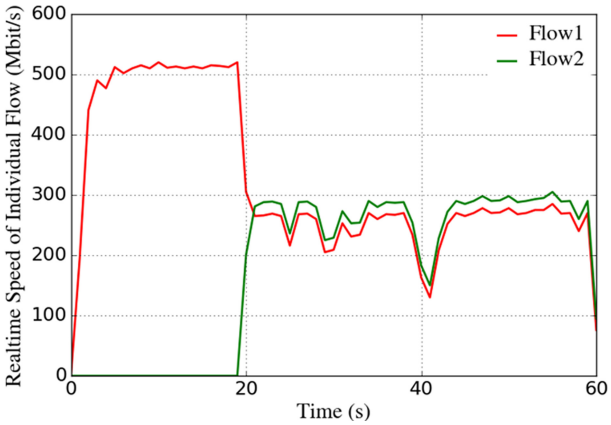

Fig. 4. Real-time speed of individual flow.

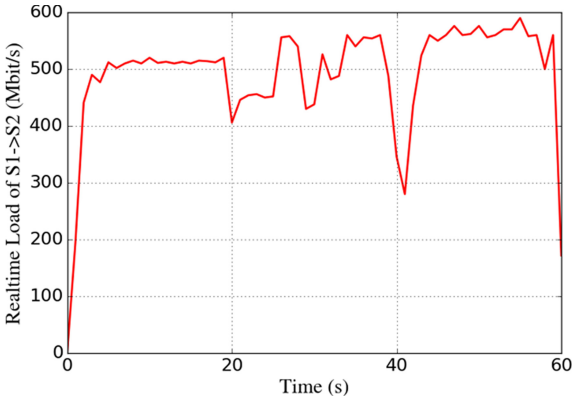

Fig. 5. Real-time load of s1- > s2.

4.2 Comparative Experiment

In order to evaluate the performance of the algorithm, we use the lightweight simulation experiment platform Mininet [15] to build the Fat-Tree network, which is implemented on a Ryu [16] controller. The Fat-Tree topology is shown as Fig. 6. It is highly accepted in data center networks due to its simple structure, ease of deployment, and scalability. In addition, we adopt the test suite of various communication modes of the data center network provided by the working group of Kandula et al. [17] as the traffic generation mode of this experiment.

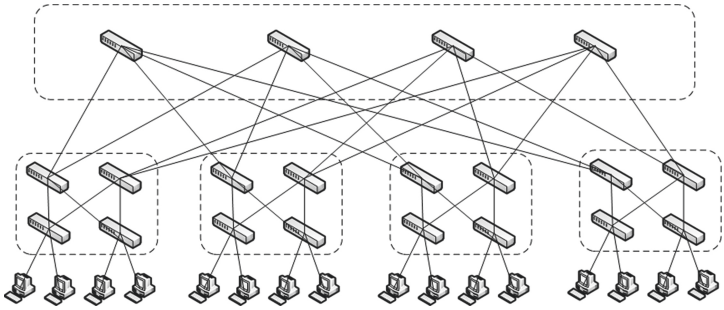

Fig. 6. Fat-tree topology with 4 pods.

Simulation Parameter. In this section, we conduct four series of experiments to show the performance of algorithms in terms of the link utilization, network throughput, round-trip delay, and delay jitter, respectively. We use the Mininet [15] to generate a Fat-tree structure of four core switches and four pods, and all of the link bandwidth between the switches are 1 Gbps. Every host sends flows to other hosts according to the probability of the traffic model, and uses Iperf command to send data to simulate the elephant flows (the default number of send is 1). Each elephant flow lasts for 60 s and occupies the bandwidth

between 100 Mbps and 1 Gbps. Mice flows are simulated with a sending interval of 0.1 s by using Ping packets (the default number of packets is 600). Each mice flow has a duration less than 0.5 s and occupies the bandwidth less than 50 Mbps.

We simulate realistic datacenter workloads with 9 types: 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0. '0.2' means that 20% of hosts in the network initiate an elephant flow, whereas the remaining hosts do not send elephant flow. Other load conditions are similar. We use the network monitoring software bwm-ng to monitor the real-time network load. The network performance test duration is 60 s. The parameters of the algorithm are shown in Table 2, which are determined after several simulation tests, so as to guarantee the convergence rate of the algorithm and the effect of solving the problem.

Table 2. Parameter setting.

Description	Parameter	Value
Num. of literation	F_{max}	5
Num. of ants	m	10
The initial value of the pheromone	t	1
The weight of pheromone	α	2
The weight value of heuristics function	β	1
The weight value of delay	μ	0.7
Volatile factor	ρ	0.1
Total pheromones	Q	1

Algorithm Effect Test 1: Link Load Balancing. The utilization of links refers to the ratio of the number of links being utilized to the total number of links. And the utilization of links also reflects the utilization of network resources. The higher the link utilization rate, the higher the return on investment of the network and the more balanced the distribution of network traffic in network. The network link is generally bidirectional. In the experiment, a network link is regarded as two opposite directions in the analysis and

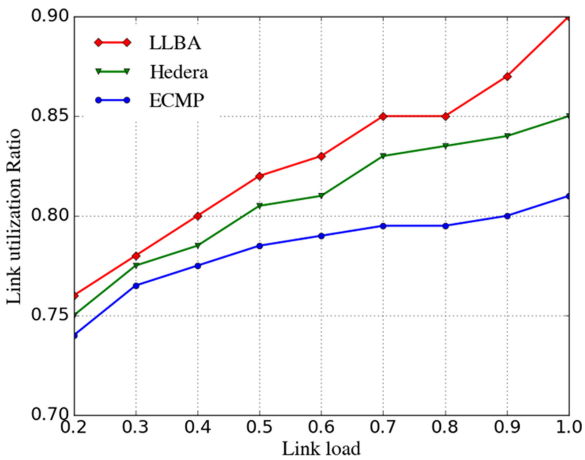

Fig. 7. Network link utilization.

calculation, and the statistical links does not include the directly connected links between the access layer switches and the end hosts. Figure 7 shows that the network link utilization performance of different algorithms as the load increases.

As shown in Fig. 7, due to the ECMP algorithm treats the available paths as the same cost paths and statically schedules all flows, the link utilization is the lowest among these algorithms. Hedera and LLBA both dynamically schedule flows according to the network real-time status and balance the link load to some extent, so the link utilization is significantly improved compared to ECMP. Hedera finds a path randomly based on the real-time load of link, and if one path meets the bandwidth requirement of the flow firstly, and then it will be selected. Although the efficiency of Hedera is high, there are some deficiencies, such as the path with less load may not be selected for a long time. The LLBA algorithm calculates the forwarding path according to the dual constraint of flow number and the real-time load, so the path with lighter load is easier to be selected. Therefore, LLBA gets the higher utilization than Hedera and ECMP. Figure 7 illustrates that the proposed algorithm has the best performance in balancing link load.

Algorithm Effect Test 2: Network Transmission Performance. The average throughput of the network refers to the throughput of per unit time obtained by the network system under the current traffic model. The standardized throughput of the network refers to the ratio of the total throughput obtained by the network system under the current traffic model and the maximum throughput obtained under the ideal traffic model. Both of them are important indicators to evaluate the network transmission performance. Figure 8 shows the average throughput and standardized throughput of different algorithms. We can see from Fig. 8, since the ECMP algorithm statically schedules traffic and it cannot make full use of network resources, the average throughput and standardized throughput are the lowest among these algorithms. Hedera and LLBA adopt flexible flow scheduling strategies, so the average throughput and standardized throughput are improved. The LLBA has a better performance in balancing link load compared with Hedera, so the network throughput and standardized throughput are higher.

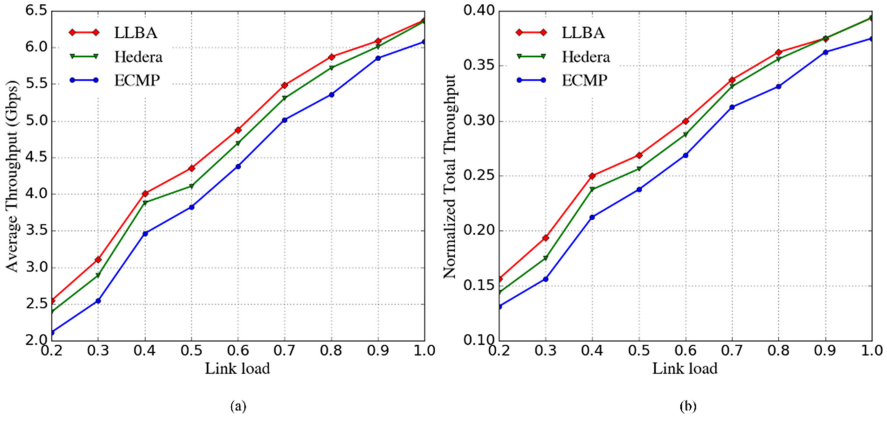

Fig. 8. The network throughput. (a) Average throughput. (b) Normalized total throughput.

Algorithm Effect Test 3: QoS. The first packet round-trip delay refers to the time between the first-packet of the flow is sent from the client to receiving the reply of the first packet from the server. The average round-trip delay refers to the average time between the sending of all data packets from the client to the receiving of the reply from the server. Both of them are important basis for detecting the service quality of the flow. Figure 9 shows that the round-trip delay and average round-trip delay of different algorithms as the load increase. When the switch forwards the non-first packet, which can match the existing flow entries in the flow table, achieving the fast forwarding, and the average round-trip delay is much lower than the first-packet round-trip delay.

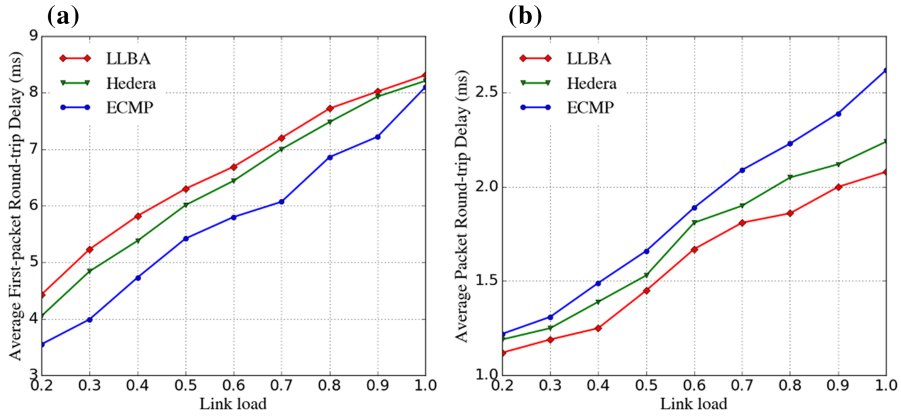

Fig. 9. The round trip delay. (a) Average delay of first-packet. (b) Average delay of packet.

As shown in Fig. 9, since the ECMP algorithm directly forwards the flows without the flow entries being sent by the controller, the round-trip delay of the first packet is the lowest. Both Hedera and LLBA need the controller to calculate the forwarding path of elephant flows and send the flow entries, so the first packet round-trip delay is higher than ECMP. Hedera forwards the to-be-scheduled flow once it finds the path which satisfies the bandwidth requirement, so the efficiency is high. The LLBA algorithm is more accurate in calculating the elephant-flow forwarding path, so the first packet has the higher round-trip delay than other packets. However, as the network load increases, the gap in first packet round-trip delay between LLBA and Hedera is gradually reduced.

LLBA algorithm preferentially selects a path with a smaller transmission delay, thereby effectively reducing the average packet round-trip delay of the flows. ECMP does not consider the link state, equivalently forwards all flows, and it is prone to network congestion due to elephant-flows collisions. Therefore, the average round-trip delay of ECMP is the highest. Hedera reduces network congestion through link load-balancing, but it does not consider the transmission delay of links, so the average round-trip delay is lower than ECMP but higher than LLBA.

The average round-trip delay deviation refers to the average value of the difference between the round-trip delay and the average round-trip delay of each packet over the

flows. The smaller the average round-trip delay deviation is, the more concentrated the round-trip delay of the packet is. The average round-trip delay deviation has some similarities with the delay jitter, and services such as audio transmission have higher requirements on these. Figure 10 shows that the average round-trip delay deviation of different algorithms as the load increases.

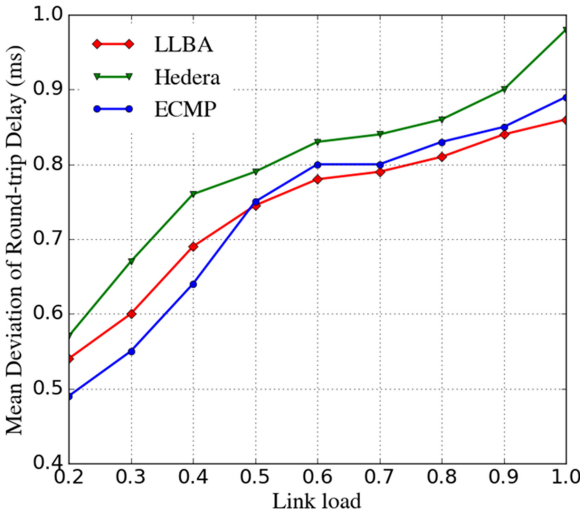

Fig. 10. The round trip delay deviation of flow.

As shown in Fig. 10, when the overall load of the network is light, the delay deviation of the ECMP algorithm is minimal. However, as the load increases, the probability of network congestion is increased due to the flow collision of ECMP, so the delay bias increases sharply. The LLBA and Hedera effectively balance the link load through dynamic scheduling, so the delay deviation increases slowly. Compared with the Hedera, LLBA has better performance in balancing link load, and it can control the transmission delay of elephant flows effectively, so the average round-trip delay deviation of the flows is minimal.

5 Conclusion

In this paper, we analyze the existing traffic scheduling algorithms in data center, and propose a link load balancing algorithm based on ant colony optimization (LLBA). Firstly, the algorithm redefines the heuristic function according to the number of elephant flows and real-time load on the link. Then, updating the pheromone according to the path length. Additionally, the optimal path determination rule is customized according to the path transmission delay and real-time load. Finally, get the optimal transmission path for the current flow after repeated iterations. Simulation experiments show that compared with other algorithms, the proposed algorithm significantly

improves link utilization and network throughput, and obtains the lower round-trip delay and the lower average round-trip delay deviation to some extent. However, the upper limit of the link bandwidth in this simulation is 1 Gbps, which cannot meet the bandwidth requirements of the actual data center network. Future work is needed to improve the upper limit bandwidth in our simulations.

Acknowledgment. This work was supported by Program for Changjiang Scholars and Innovative Research Team in university (IRT_16R72).

References

1. Wang, Y., You, S.: An efficient route management framework for load balance and overhead reduction in SDN-based data center networks. IEEE Trans. Netw. Serv. Manage. **15**(4), 1422–1434 (2018)
2. Cong, L., Yong-Hao, W.: Strategy of data manage center network traffic scheduling based on SDN. In: 2016 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS), Changsha, pp. 29–34 (2016)
3. Wang, J.M., Wang, Y., Dai, X., Bensaou, B.: SDN-based multi-class QoS guarantee in inter-data center communications. IEEE Trans. Cloud Comput. **7**(1), 116–128 (2019)
4. Wang, W., Sun, Y., Zheng, K., Kaafar, M.A., Li, D., Li, Z.: Freeway: adaptively isolating the elephant and mice flows on different transmission paths. In: 2014 IEEE 22nd International Conference on Network Protocols, Raleigh, NC, pp. 362–367 (2014)
5. Zhang, H., Guo, X., Yan, J., Liu, B., Shuai, Q.: SDN-based ECMP algorithm for data center networks. In: 2014 IEEE Computers, Communications and IT Applications Conference, Beijing, pp. 13–18 (2014)
6. Truong, T., Fu, Q., Lorier, C.: FlowMap: Improving network management with SDN. In: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, pp. 821-824 (2016)
7. Qiu, S., Yu, X., Wang, K., Gu, H.: MiFlO: a scheduling algorithm based on mice flows optimization in hybrid data center network. In: 2017 16th International Conference on Optical Communications and Networks (ICOCN), Wuzhen, pp. 1–3 (2017)
8. Al-Fares, M., et al.: Hedera: dynamic flow scheduling for data center networks. In: Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, 28–30 April 2010, San Jose, CA, USA DBLP (2010)
9. Curtis, A.R., Kim, W., Yalagandula, P.: Mahout: low-overhead datacenter traffic management using end-host-based elephant detection. In: 2011 Proceedings IEEE INFOCOM, Shanghai, pp. 1629–1637 (2011)
10. Long, L., Binzhang, F., Lixin, C.: Nimble: a fast flow scheduling strategy for OpenFlow networks. Chin. J. Comput. **38**(5), 1056–1068 (2015)
11. Hu, W., Liu, J., Huang, T., Liu, Y.: A completion time-based flow scheduling for inter-data center traffic optimization. IEEE Access **6**, 26181–26193 (2018)
12. Kanthimathi, M., Vijayakumar, D.: An enhanced approach of genetic and ant colony based load balancing in cloud environment. In: 2018 International Conference on Soft-computing and Network Security (ICSNS), Coimbatore, pp. 1–5 (2018)
13. Dorigo, M., Stützle, T.: Ant colony optimization: overview and recent advances. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. ISORMS, vol. 272, pp. 311–351. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91086-4_10

14. Wang, C., Zhang, G., Xu, H., Chen, H.: An ACO-based link load-balancing algorithm in SDN. In: 2016 7th International Conference on Cloud Computing and Big Data (CCBD), Macau, pp. 214–218 (2016)
15. Mininet. http://www.mininet.org/
16. Ryu. https://github.com/osrg/ryu
17. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. ACM SIGCOMM Comput. Commun. Rev. **38**(4), 63–74 (2008)