Hindawi
Scientific Programming
Volume 2020, Article ID 8888375, 8 pages
https://doi.org/10.1155/2020/8888375

Research Article

Elephant Flow Detection Mechanism in SDN-Based Data Center Networks

Qian Tang, Huan Zhang, Jun Dong, and Lianming Zhang

College of Information Science and Engineering, Hunan Normal University, Changsha, China

Correspondence should be addressed to Jun Dong; jdongcn@outlook.com and Lianming Zhang; zlm@hunnu.edu.cn

Received 14 April 2020; Accepted 14 August 2020; Published 1 September 2020

Academic Editor: Antonio J. Peña

Aiming at the problem of network congestion and unbalanced load caused by a large amount of data capacity carried in elephant flow in the data center network, an elephant flow detection method based on SDN is proposed. This method adopts the autodetect upload (ADU) mechanism. ADU is divided into two parts: ADU-Client and ADU-Server, in which ADU-Client runs in the host computer and ADU-Server runs in the SDN controller. When the host sends the elephant flow, the ADU-Client generates a packet with forged source IP address and triggers the Packet_in message of the edge switch to report the information of the elephant flow to the SDN controller and the ADU-Server completes the elephant flow identification. Experimental results show that the ADU elephant flow detection mechanism can effectively detect elephant flow in the data center network, reduce detection time, and improve network performance.

1. Introduction

In the data center network, the problem of network link congestion and uneven load caused by elephant flow is becoming increasingly serious. In the process of elephant flow forwarding, a large number of data packets are easily congested to a link node, thus causing problems such as the collision of transmission links, network congestion, and transmission delay. At the same time, the mouse flow cannot obtain sufficient bandwidth, which increases the transmission delay of the mouse flow. Although the number of elephant flow accounts for about 2% of the total flow, the 2% data flow carries 90% of the total flow [1]. Wang et al. [2] proposed a novel flow scheduling scheme, which uses path diversity in DCN topology to ensure that mouse flows are completed within the deadline and the network utilization rate is high. Hu et al. [3] designed a coding-based adaptive packet spraying. It spreads short-stream packets to all paths, while elephant flow is limited to a few paths by equal-cost multipath. To solve the resource competition problem between mixed flows, Liu et al. [4] proposed an adaptive traffic isolation scheme. Based on the grouping spraying scheme in multipath transmission, it dynamically separates the elephant flow from the mouse flow on different paths, thereby providing low latency for the mouse flow.

To effectively solve the problems of low link utilization and unbalanced link load in the data center network [5, 6], it is particularly important to identify and perform reasonable scheduling for elephant flows. Hedera [7] is a dynamically adaptive network traffic scheduling system based on business flow. By dynamic scheduling traffic flows with large traffic requirements, it avoids collision problems of large equal-cost multipath flows. In the Hedera architecture, the business flow on the switch is monitored. When the flow demand increases to a set threshold, the flow is marked as a large flow. On the edge switch, a periodic polling scheduler is used to collect flow statistics. The large flow is monitored every 5 seconds to achieve an accurate balance between the large business flow demand and the minimum scheduler overhead. Mahout [8] detects the elephant flow directly on the terminal, uses the "clip lamellar" added on the terminal to identify the elephant flow, and then reports it to the elephant flow management server. The terminal monitors the socket sending buffer to determine whether there is an elephant flow that needs to be transmitted in the data center network. When the terminal detects the elephant flow, it will mark a specific identification in the differentiated services code

point (DSCP) field of the IP header of the first packet of the elephant flow. The controller will send a flow table with the lowest priority to all switches after startup. When an identified elephant flow data packet is sent to the network, the switch will match the flow table of the DSCP field and send this packet to the controller through the Packet_in message to report the elephant flow. This method can effectively reduce the detection time, but it needs to deploy a dedicated server to collect the elephant flow information reported by the clip lamellar, which has a high deployment cost. Fincher [9] is based on a stable matching elephant flow scheduling algorithm. According to the global network information counted by the data center, it generates a prior table of network traffic and switches to provide a stable match between switches and elephant flows and a suitable forwarding path for all network traffic. The algorithm can effectively use network space, decrease data transmission time, reduce transmission delay, and prevent network link blockage caused by elephant flow. Liu et al. [10] proposed a traffic load balancing scheme based on software-defined networking (SDN) technology, using flow engineering to identify and manage elephant flows and using a weighted multipath routing algorithm to dispatch elephant flows to multiple roads. Forwarding improves network throughput and link usage efficiency. This method solves the scheduling of elephant flow well, but it cannot avoid the forwarding conflict between elephant flow and mouse flow, which is easy to cause transmission delay of mouse flow in the scheduling process.

To reduce the detection time of the elephant flow, reduce the transmission delay of the network, and improve the link utilization while taking advantage of the SDN technology, this paper studies the elephant flow detection mechanism of the data center network. Compared with the existing literature, this paper has the following major contributions:

(1) According to the fat-tree topology of the data center network and the characteristics of the SDN network, a frame for SDN-based central data networks is proposed.

(2) An autodetect upload (ADU) detection mechanism based on the OpenFlow protocol is proposed. This mechanism reduces the time consumption by self-reporting the elephant flow from the terminal.

(3) The reporting process of ADU depends on the Packet_in mechanism of the OpenFlow protocol. The switch can directly send information to the controller through the Packet_in message, which realizes a low-consumption and high-efficiency elephant flow detection mechanism.

This paper is organized as follows. We introduce data center networks and SDN networks and propose a frame for SDN-based data center networks in Section 2. Section 3 presents a mechanism for ADU elephant flow detection in the SDN-based data center network. The effectiveness of the algorithm is verified by a simulation experiment in Section 4. Finally, the conclusion is given in Section 5.

2. SDN-Based Data Center Network

Traditional network technology cannot solve the problem of poor network performance existing in the current data center network. The emergence of an SDN, which has the characteristics of visualization and centralized management, can realize the programmable operation of the end-to-end virtual paths to the data center network. Through the SDN controller, multiple core devices can be aggregated together for high-speed forwarding management.

2.1. Data Center Network. With the increasing popularity of cloud computing and virtualization technology [11–16], a large number of servers are connected to switches through high-speed links. Therefore, the data center carries a huge amount of data and an efficient flow scheduling strategy is needed to achieve load balancing of current network flow, improve link utilization, and reduce link congestion [17]. The network congestion caused by the elephant flow in the data network center is becoming increasingly prominent. To ensure efficient flow scheduling, the current network situation must be monitored. In other words, first, detect and identify the elephant flow and then reasonably schedule the elephant flow to avoid link congestion and flow transmission delay caused by the elephant flow. At the same time, the centralized exchange of data in the network and a large amount of east-west traffic are the characteristics of the data center network [18]. Consequently, the data center network must have efficient network communication protocols, flexible topology, and agile link capacity control. Meanwhile, it also needs to meet large-scale, highly extensible, low configuration overhead, and high bandwidth between servers. Not only should green energy be saved, but also the cost should be reduced [19].

Fat-tree topology is the most common topology in data center networks. It is divided into three layers: core layer, convergence layer, and border access layer [20]. The switches in the edge access layer and convergence layer form a Pod, and the switches in the two layers form a 1:1 bandwidth convergence topology. The convergence switch in the Pod will be connected with the switches in the core layer so that each edge switch can establish a connection with the core switch to achieve maximum utilization. In a Pod, there are k switches, of which $(k/2)$ are located in the convergence layer and $(k/2)$ are located in the access layer, while the core switches have $(k/2)^2$ connected to the k convergence layer switches.

This one-to-many design can enable each node to have multiple paths and can better handle the conflict problem of lateral traffic. This kind of design can disperse the traffic and is beneficial to the load balance of the network. When one of the links fails, it can switch quickly and continue transmission on other paths. The redundancy performance of links is improved, greatly reducing the impact of a single point of failure on the network.

2.2. Software-Defined Networking. SDN is a new type of network architecture, and it is derived from the idea of network abstraction [21–28], which separates the network and consists of the control layer and data layer so that the controller can obtain global views, topology structure, network state information, etc. The data layer is mainly responsible for forwarding, decoupling the traditional tightly coupled network equipment, providing centralized management and dynamic maintenance of the distributed network, optimizing the network management, and making the network framework more flexible and agile. According to ONF's standardized definition of SDN, the software-defined network can be divided into three functional layers from top to bottom [29]: application layer, control layer, and forwarding layer. SDN separates the control plane from the forwarding plane, which is centrally controlled and managed by the control plane. In the application plane of SDN, through the API interface, the required network behavior in the switch can be submitted to the upper-layer controller, including a plurality of northbound interface drivers. At the same time, it can abstract and encapsulate its functions to provide the northbound proxy interface to the outside. The forwarding layer is mainly responsible for forwarding data as well as processing data and reporting network information. According to the OpenFlow protocol, when a new flow reaches the OpenFlow switch [30], the matching information is first found in the forwarding table. If the match is successful, it is forwarded according to the rules. If the flow table does not match, it is reported to the controller.

The OpenFlow protocol is a standard SDN communication protocol. It is the standard communication protocol between the control layer and the forwarding layer and is also the most important southbound interface protocol at present. OpenFlow switch is the lowest-level forwarding device in the SDN architecture. Through OpenFlow switches, SDN controllers can monitor the network and formulate forwarding rules for the data plane. The SDN controller can control the flow table in the switch and send all decisions to each switch through the OpenFlow switch. In SDN networks, each OpenFlow switch has a secure channel that connects the switch to a remote controller, allowing commands and data packets sent between the controller and the switch to pass through. The OpenFlow protocol uses the form of a flow table. The main idea is that the controller is mainly responsible for controlling the network, while the switch is mainly responsible for forwarding. Once the data packet [21] is received, the OpenFlow switch will look up the corresponding flow entry from the first flow table, with the highest priority, parse the packet header field, and send it to the system for data package match. If the corresponding entry of the flow is found, the instruction set contained in the flow executes the forwarding rules of the entry. These instructions will include changes to data packets, operations, and pipeline processing. When the flow table entry set by the instruction does not contain the go-to table instruction, the pipeline stops processing and operating the execution data packets. If the packet does not have a matching flow table, the OpenFlow switch will send a Packet_in message to the controller [31]. Meanwhile, the data packet is encapsulated and forwarded to the controller. The controller will reply with a flow table and the original data packet, and the switch will set up the flow table entry. Also, when the switch receives a data packet if the flow table has multiple levels, the header field of the data packet and the flow table entry will be matched one by one according to the priority. The data packet will match the entry with the highest priority as the matching result. The data packet will be processed according to the specified operation in the flow entry [32], but if the match is unsuccessful, it will be sent to the SDN controller through the secure channel of the switch.

The main function of the Packet_in message is to send the data packets in the OpenFlow switch to the controller. In general, the conditions under which a Packet_in message is triggered are as follows: (1) the corresponding flow entry cannot be matched; (2) the match is successful, but the matched flow entry has a "send message to the controller" behavior.

2.3. SDN-Based Data Center Network. The scale of the data center network is continuously expanding, and its traffic has many large business flows, strong traffic bursts, and strong traffic periodicity. The data center network needs centralized management and monitoring, and multipath forwarding is required to achieve load balancing of transmission links. Traditional network technology limits the flexibility of resource provision and the real-time management of the network in the cloud era, making the network performance unable to improve and falling into a bottleneck. To solve the problem of the flexible allocation of network resources in the data center, SDN technology is introduced. SDN separates control from forwarding, realizes flexible control of the network, and has a global view of the network. In the meantime, introducing SDN into the data center network can logically centralize the management of the network. Multipath forwarding can be achieved through the SDN controller. The openness and virtualization of SDN network capabilities can fully meet the needs of open data center capabilities, intelligent deployments and migration of virtual machines, and mass virtual tenants. The SDN-based central data network is shown in Figure 1.

SDN divides the network into the control layer and forwarding layer. The SDN controller performs centralized monitoring on the control plane, and the OpenFlow switch is deployed on the forwarding plane. The two establish a connection through the standard OpenFlow southbound interface protocol. ADU detection mechanism is deployed in the SDN-based data center network. The ADU-Client runs in the host and the ADU-Server runs in the controller. Report the elephant flow information to the controller by triggering the Packet_in message of the edge switch. This method not only can achieve ultraefficient elephant flow detection but also does not need to consume additional deployment costs.

3. ADU Elephant Flow Detection

This paper proposes a new elephant flow detection method—ADU, which is divided into two parts: ADU-Client and

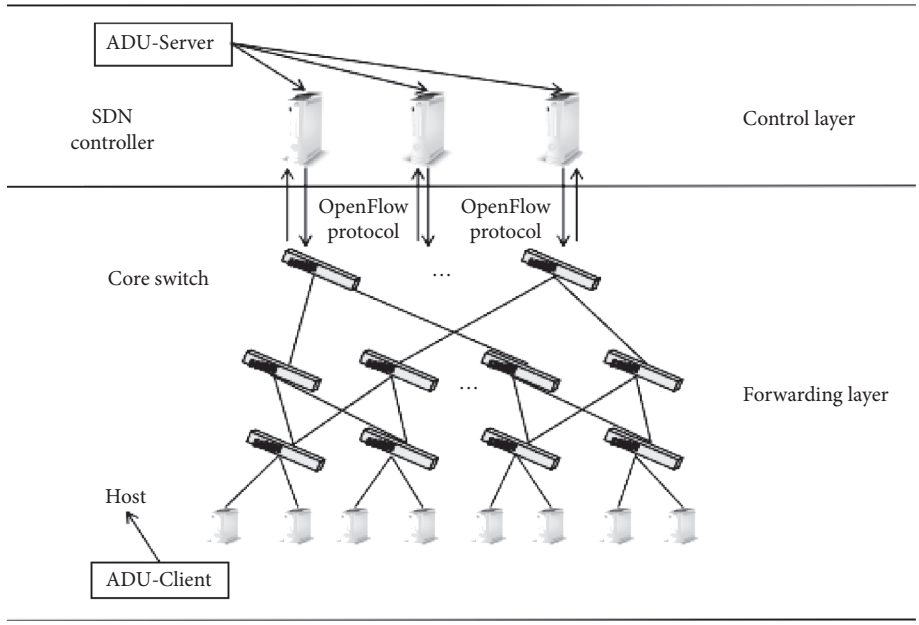

Figure 1: SDN-based central data network.

ADU-Server. ADU-Client runs in the host, and ADU-Server runs in the controller. When a host is about to send out an elephant flow, ADU-Client will generate a data packet with a forged source IP address and report the information of the elephant flow to the controller by triggering the Packet_in message of the edge switch. An elephant flow detection mechanism with low consumption and high efficiency is realized.

3.1. Principle of the Method. ADU-Server runs as a module in the controller. After the OpenFlow switch is connected to the controller, the ADU-Server will issue a flow table with the highest priority to all edge switches. The matching item is the source IP address == ADU-IP, and the action is to forward the flow table to the controller. And let the flow table take effect permanently, that is, the Hard_timeout of the flow table is set to 0. ADU-IP is a variable, which indicates the unique identifier of the message reported by the ADU. ADU-IP must be set to an IP address that does not exist in the current data center network; otherwise, it will affect the normal operation of the network. After issuing the flow table, the ADU-Server will continuously monitor the Packet_in message received by the controller. When the controller receives the Packet_in message, the ADU-Server module preferentially filters and compares the data fields of the Packet_in message. If it is found that the data field of the Packet_in message contains the packet header of the network layer IP protocol and the source IP address is ADU-IP, then the data packet is identified as an elephant flow detection report data packet, and the information of the elephant flow is analyzed according to relevant rules. The algorithm running in ADU-Server is shown in in Algorithm 1.

The main function of the ADU-Client is to monitor the network data transmission buffer queue of the host computer. When the amount of data to be transmitted for a certain connection exceeds the judgment threshold of the elephant flow, the flow is judged to be an elephant flow, and the elephant flow information is encapsulated according to the reporting rule. Then, it constructs a data packet whose source IP address is ADU-IP and the destination address of the data packet is the destination address IP of the elephant flow, and the packet is called an ADU message. Then, send the ADU message. The process of ADU-Client is shown in Figure 2.

3.2. Reporting Process of ADU Detection. The reporting process of ADU depends on the Packet_in of OpenFlow protocol. The switch can send information to the controller through the Packet_in message. Generally, there are two mechanisms to trigger Packet_in messages. One is that the switch does not match the flow table corresponding to the current data to be forwarded. The other is that the action specified by the flow table matching the data to be forwarded is to report to the controller. The reporting process of ADU is based on the second trigger mechanism of Packet_in mentioned above. ADU-Server will send a flow table to all edge switches that trigger Packet_in messages. The ADU report message is a network layer IP data packet, which is mainly composed of ADU-IP, elephant flow starting IP, elephant flow destination IP, and elephant flow size. Figure 3 shows an ADU message whose transport layer is TCP protocol.

The ADU-IP is used as the unique identifier for the controller to identify the report message. The destination address of the elephant flow is the destination IP address of this IP packet, which is a part of the elephant flow information. ADU-DATA is a data segment. The first 4 bytes are the hexadecimal data of the source IP address of the elephant stream. The content of the remaining data

Input: Packet_in
Output: elephant flow info
(1) When wait for Packet_in
(2) **If** package.src_ip == ADU-IP
(3) data = getTcpData (package)
(4) elephant_flow.src_ip = data [0: 3]
(5) elephant_flow.size = data [4:]
(6) return elephant_flow
(7) **Else**
(8) return NULL

Algorithm 1: Process of ADU-Server.

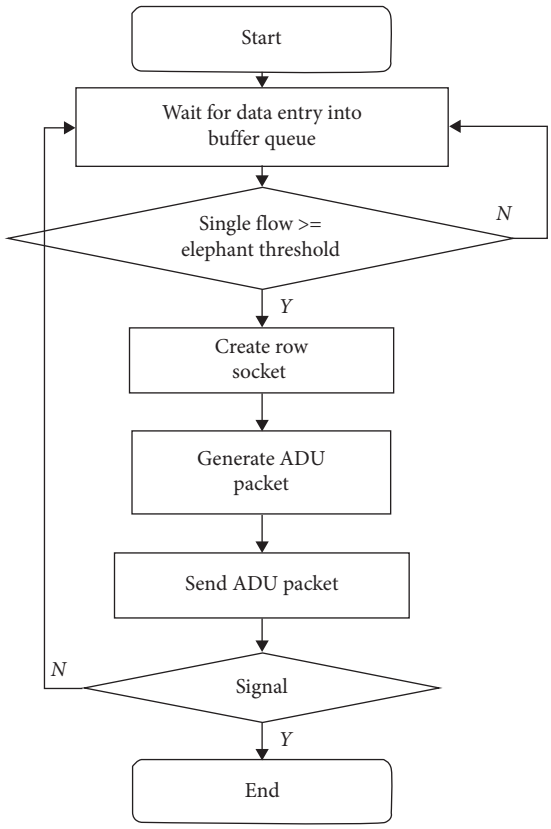

Figure 2: Process of ADU-Client.

segment represents the size of the elephant stream (in KB as a unit). In Figure 3, the content of ADU-DATA is 0*x*2800000107E2, the first 4 bytes are 0*x*28000001, which is converted to IP address: 40.0.0.1, and 0*x*07E2 is converted to decimal system equal to 2018. From this, it can be parsed that the information reported in Figure 3 is an elephant flow with a size of 2018 KB sent from the host 40.0.0.1 to the host 40.0.0.16.

4. Experiments and Results

4.1. Experimental Setup. The experiment uses two physical hosts PC1 and PC2, both running Ubuntu 16.04 operating system. The experimental steps are as follows. (1)

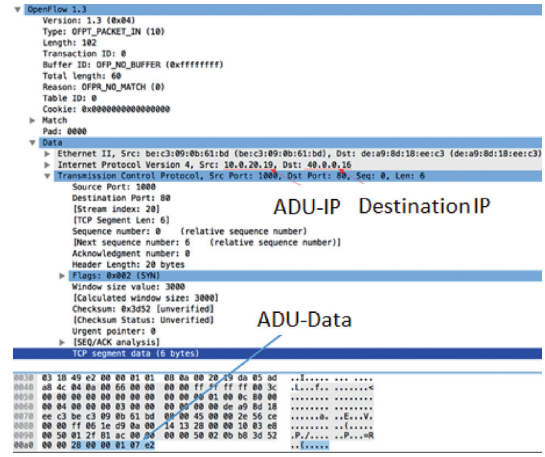

Figure 3: Packet_in of elephant flow.

OpenFloodlight is deployed on PC1 as an SDN controller, and the ADU-Server module runs in OpenFloodlight. (2) Mininet is deployed on PC2 to simulate the experimental topology and configure the network switch to connect it to the OpenFloodlight controller. (3) The ADU-Client is deployed to simulate the network card driver on each host simulated by Mininet and provides the data transmission interface in the manner of HTTP. At the same time, each host runs a data stream generation program, and the length of the data stream obeys exponential distribution, and the generation time of the data stream obeys Poisson distribution, calling the ADU-Client sending interface to send data.

To verify the efficiency of ADU, we compare the ADU elephant flow detection algorithm with Hedera's elephant flow detection algorithm under various elephant flow determination thresholds. The detection time and the detection accuracy rate of elephant flow are taken as evaluation indexes. Elephant flow detection time refers to the time required for an elephant flow to enter a network link to obtain information about this elephant flow. The correct detection rate of elephant flow is divided into two parts: the missing rate and the false reject rate. The missing rate refers to the ratio of the number of elephant flows that have not been correctly detected to the total number of elephant flows. False reject rate refers to the proportion of the number of elephant flows misjudged by the mouse flow as a proportion of all elephant flows.

In the experiment, the data flow generating program generates data flows greater than or equal to the elephant flow decision threshold and less than the elephant flow decision threshold with a probability of 1:1. Considering that the Hedera algorithm detects elephant flow as a link-state polling method, the communication mode uses the Staggered mode (0.33, 0.33), which makes the number of flows sent to the same edge switch, Pod, and other hosts similar. Take one minute for each different elephant flow decision threshold and record the timestamp of all elephant flows issued within one minute and the timestamp of the elephant flow identified by the controller program, which will eventually be different. Finally, the average value of the

detection time of all elephant flows under different decision thresholds is obtained.

4.2. Results. Figure 4 shows the time required for elephant flow detection. As can be seen from Figure 4, the time required by the ADU elephant flow detection method is far less than that required by the Hedera method, both at the millisecond level. This is because ADU adopts host network buffer monitoring mode. When a packet with the same destination address appears in the send buffer of a host that is larger than the elephant flow decision threshold, the ADU detection method can immediately detect the elephant flow and report it to the controller. But the Hedera elephant flow detection method takes dozens of times longer than the ADU method. This is because the main idea of the Hedera method is to periodically poll the samples and then analyze whether the flow of each link exceeds the elephant flow determination threshold. When an elephant flow is sent from the host, Hedera cannot detect it immediately but waits for several polling periods to determine that flow is an elephant flow.

Figure 5 shows the false reject rate. As can be seen from Figure 5, ADU and Mahout do not have false detection under various elephant flow thresholds, and the false reject rate is always 0. This is because the elephant flow detection methods used by ADU and Mahout both judge the elephant flow or mouse flow from the socket buffer queue detected by the terminal, so it is difficult to cause false detection. Hedera uses flow to poll the number of packets in each switch to identify the elephant flow. However, the number of data packets cannot obtain an accurate value, so there is a certain probability of misjudgment.

Figure 6 shows the missing rate of elephant flow. It can be seen from Figure 6 that the missed detection rate of the ADU is always 0 under various decision thresholds, while the missed detection rate of Mahout, which also adopts the terminal detection and reporting mechanism, exceeds 0.1%. This is due to a defect in Mahout's elephant flow reporting mechanism. The priority of the matching flow table of the elephant flow reported by the switch is the lowest. The controller's default forwarding flow table has idle timeout duration and forced timeout duration. When host A sends a mouse flow Fm to host B for the first time, the switch will ask the controller for the forwarding path through the Packet_in message because it cannot find the forwarding flow table, and then the controller will send the default forwarding path to the relevant switch to establish the forwarding path from host A to host B. In the future, a forwarding path flow table from host A to host B will temporarily exist in the network. Before this flow table times out, the terminal can correctly identify the DSCP field in the first IP packet header of Fe. At this time, the forwarding flow table from host A to host B already exists in the network, and the priority of the DSCP flow table is lower than that of the forwarding flow table. If host A sends an elephant flow Fe to host B again, the packet of the elephant flow Fe cannot trigger the Packet_in message of the switch after reaching the switch, thus causing the elephant flow Fe to miss detection. Hedera caused a small amount of missed detection due to certain uncertainty in the detection

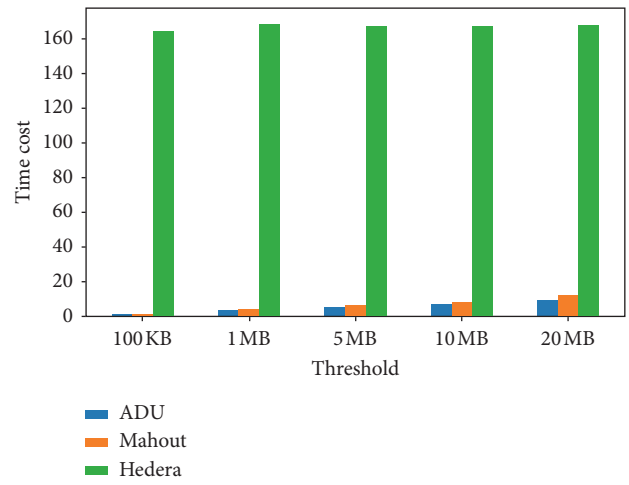

Figure 4: Time required for elephant flow detection.

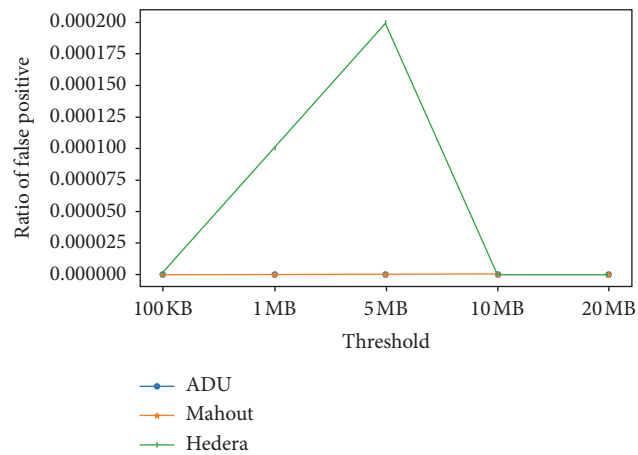

Figure 5: False reject rate of elephant flow.

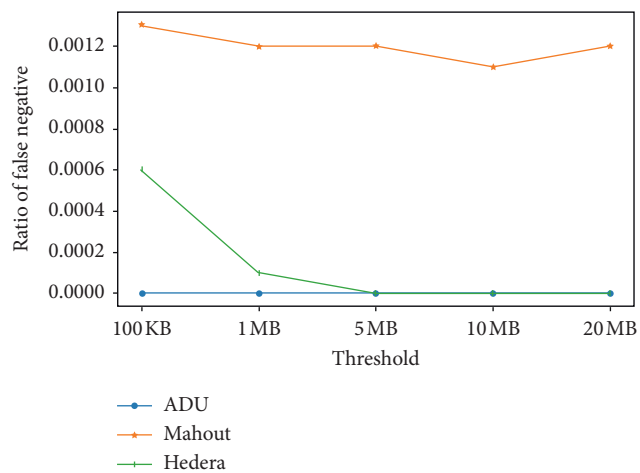

Figure 6: Elephant missing rate.

process; especially when the threshold of elephant flow is low, the discrimination between elephant flow and mouse flow is low.

5. Conclusion

Based on the SDN framework and OpenFlow technology, this paper proposes a new ADU elephant flow detection method for elephant flow in the data center network. ADU detection relies on the Packet_in mechanism of the OpenFlow protocol. The switches can send information to the controller directly through the Packet_in message. Experiments show that the self-reported ADU detection mechanism implemented at the terminal can detect elephant flows with low consumption and high efficiency, thereby reducing network delay, improving network link utilization, and improving network performance.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This study was supported by the National Natural Science Foundation of China (no. 61572191) and Humanities and Social Sciences Research Program Youth Fund Project of the Ministry of Education of China (no. 20YJCZH020). The authors would like to thank Jing Zhang from Beihua University for polishing the paper.

References

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[2] W. Wang, Y. Sun, K. Salamatian, and Z. Li, "Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 5–18, 2016.

[3] J. Hu, J. Huang, W. Lv, Y. Zhou, and T. He, "CAPS: coding-based adaptive packet spraying to reduce flow completion time in data center," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2338–2353, 2019.

[4] J. Liu, J. Huang, W. Lv, and J. Wang, "APS: adaptive packet spraying to isolate mix-flows in data center network," *IEEE Transactions on Cloud Computing*, p. 1, 2020.

[5] S. Liu, J. Huang, Y. Zhou, J. Wang, and T. He, "Task-aware TCP in data center networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 389–404, 2019.

[6] J. Huang, Y. Huang, J. Wang, and T. He, "Adjusting packet size to mitigate TCP incast in data center networks with COTS switches," *IEEE Transactions on Cloud Computing*, p. 1, 2019.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks," in *Proceedings of the 7th Usenix Symposium on Networked Systems Design and Implementation*, San Jose, CA, USA, April 2010.

[8] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: low-overhead datacenter traffic management using end-host-based elephant detection," in *Proceedings of INFOCOM*, Shanghai, China, April 2011.

[9] Y. Zhang, L. Cui, and Q. Chu, "Fincher: elephant flow scheduling based on stable matching in data center networks," in *Proceedings of IEEE 34th International Performance Computing and Communications Conference*, Nanjing, China, December 2015.

[10] J. Liu, J. Li, G. Shou, Y. Hu, Z. Gou, and W. Dai, "SDN based load balancing mechanism for elephant flow in data center networks," in *Proceedings of International Symposium on Wireless Personal Multimedia Communications*, Sydney, Australia, September 2014.

[11] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, "Deep-learning-based joint resource scheduling algorithms for hybrid MEC networks," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6252–6265, 2020.

[12] K. Gao, F. Han, P. Dong, N. Xiong, and R. Du, "Connected vehicle as a mobile sensor for real time queue length at signalized intersections," *Sensors*, vol. 19, no. 9, p. 2059, 2019.

[13] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep Q-learning," *Information Sciences*, vol. 512, pp. 1170–1191, 2020.

[14] H. Ma, H. Zhu, K. Li, and W. Tang, "Collaborative optimization of service composition for data-intensive applications in a hybrid cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1022–1035, 2019.

[15] J. Long, W. Liang, K.-C. Li, D. Zhang, M. Tang, and H. Luo, "PUF-based anonymous authentication scheme for hardware devices and IPs in edge computing environment," *IEEE Access*, vol. 7, pp. 124785–124796, 2019.

[16] J. Mei, K. Li, Z. Tong, Q. Li, and K. Li, "Profit maximization for cloud brokers in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 190–203, 2019.

[17] L. Zhang, K. Wang, D. Xuan, and K. Yang, "Optimal task allocation in near-far computing enhanced C-RAN for wireless big data processing," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 50–55, 2018.

[18] Y. Qiao, Z. Hu, and J. Luo, "Efficient traffic matrix estimation for date center networks," in *Proceedings of IFIP International Conference on Networking*, Brooklyn, NY, USA, May 2013.

[19] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proceedings of INFOCOM*, Turin, Italy, April 2013.

[20] Y. Li and D. Pan, "OpenFlow based load balancing for fat-Tree networks with multipath support," in *Proceedings of 12th IEEE International Conference on Communications*, Budapest, Hungary, June 2013.

[21] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: a low cost network monitoring framework for software defined networks," in *Proceedings of IEEE Network Operations and Management Symposium*, Krakow, Poland, May 2014.

[22] Z. Chen, Z. Luo, X. Duan, and L. Zhang, "Terminal handover in software-defined WLANs," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, p. 68, 2020.

[23] S. Zhu, Z. Sun, Y. Lu, L. Zhang, Y. Wei, and G. Min, "Centralized QoS routing using network calculus for SDN-

based streaming media networks," *IEEE Access*, vol. 7, no. 1, pp. 146566–146576, 2019.

[24] X. Zhong, L. Zhang, and Y. Wei, "Dynamic load-balancing vertical control for a large-scale software-defined internet of things," *IEEE Access*, vol. 7, no. 1, pp. 140769–140780, 2019.

[25] D. Yin, L. Zhang, and K. Yang, "A DDoS attack detection and mitigation with software-defined internet of things framework," *IEEE Access*, vol. 6, no. 1, pp. 24694–24705, 2018.

[26] Y. Hu, T. Peng, and L. Zhang, "Software-defined congestion control algorithm for IP networks," *Scientific Programming*, vol. 2017, Article ID 3579540, 8 pages, 2017.

[27] L. Zhang, Q. Deng, Y. Su, and Y. Hu, "A box-covering-based routing algorithm for large-scale SDNs," *IEEE Access*, vol. 5, no. 1, pp. 4048–4056, 2017.

[28] K. Wang, K. Yang, H.-H. Chen, and L. Zhang, "Computation diversity in emerging networking paradigms," *IEEE Wireless Communications*, vol. 24, no. 1, pp. 88–94, 2017.

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

[30] ONF, "Software-defined networking: the new norm for networks," *ONF White Paper*, 2012.

[31] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, Moscow, Russia, October 2013.

[32] M. Koerner and O. Kao, "Multiple service load-balancing with OpenFlow," in *Proceedings of IEEE 13th International Conference on High Performance Switching and Routing*, Belgrade, Serbia, June 2012.