Traffic Control in a Simple SDN Network Topology using Mininet: Emulating SDN Flow Entry Management

1st Yufei Wu 2141453 2nd Kexin Chen 2142454 3rd Hangling Wang 2144994

4th Ziyi Qiu 2145776 5th Zhenyang Zhao 2142074

***Abstract*—Software-Defined Networking (SDN) is a network architecture which has core idea to separate the network control plane and data plane, so that the control logic of network devices is centralized in a centralized controller instead of scattering in various network devices. Therefore, with the use of SDN, network manager can manage the whole network through the controller more easily and adapt to changing requirements more quickly than the traditional network. In this coursework, our group will leverage Mininet to establish a basic SDN network topology with an SDN controller based on Ryu framework. The main goal is to manipulate SDN flow entries in order to simulate traffic control functions without client's awareness including traffic forwarding and redirection. In this report, the main design idea and workflow are shown through diagrams and the algorithms that implement the forwarding and redirection processes are shown in pseudocode. Then the steps of implementation procedure are listed clearly. The results display the flow entries for traffic forwarding to server1 and redirection to server2 created successfully. Moreover, the networking latency behaviors are demonstrated with comparison between forwarding and redirection by Wireshark capturing packets. By summarizing and analyzing, our team finds that redirection has long latency and network instability.**

***Index Terms*—Ryu Controller, TCP/IP, Mininet, Computer Network, Ethernet**

I. Introduction

A. background

Software-Defined Networking is a major shift in the architecture of computer network that changed the way networks are structured, managed and operated. SDN separates the control plane from the data plane which makes the control logic of the network centralized [1]. By decomposing the network control problem into manageable chunks, SDN makes it easier to create and introduce new abstractions into networks, simplifies management, and encourages network growth [2]. It allows the manager to administer the network dynamically and improve its behavior through software. Traffic administration, security or energy efficiency in the network can be supported through written applications [3]. In the coursework, a simple SDN network topology with an SDN controller will also be built.

B. Project tasks

In the coursework, the assignment has 5 parts. Firstly, a simple SDN network topology with the use of Mininet is required to be established. Secondly, apply Ryu framework to build two SDN controller applications and ensure that all nodes are properly connected. Thirdly, one controller application should realize the forwarding to Server1 and another should implement redirection to Server2. Fourthly, run server.py and client.py on two servers and one client respectively. Finally, Determine the network delay by using Wireshark/Tcpdump on the client to capture packets.

C. Challenges encountered

During the process, since our group is not familiar with the knowledge on Ryu framework, members take much time on it to write forward and redirection python applications well. Another challenge that our group meet is the final result representation. Firstly, our group member takes histogram to display the results on comparison of forwarding and redirection's rates. However, other members think that it is not intuitive. Finally, after discussing, all members agree to use line chart.

D. Potential application

For the SDN network topology we designed, it has some potential application. It can be applied on improving network transport performance and utilization through better management of network traffic with the SDN controller. Furthermore, with the use of tools such as Wireshark or Tcpdump, it can be used to monitor network's traffic and analyze its performance to achieve troubleshooting. Besides, SDN can adapt changes in requirements nimbly, so that it can be used to different application and traffic requirements.

II. Related Work

As the use of the Internet becomes more widespread, traffic control and redirection in distributed systems become more and more important, and many products are designed to address and improve this problem. SDN controller optimization challenges for partially deployed traffic engineering have been presented by relevant practitioners, and fast full polynomial time approximation techniques (FPTAS) have been

devised to solve these problems [4]. Moreover, the performance advantages that are possible with these techniques, even with a gradually implemented SDN, are demonstrated by both analysis and ns-2 simulations. Another innovative approach is to use reinforcement learning to solve or facilitate network traffic redirection. SDN offers centralized control with a special benefit for the application of reinforcement learning-based intelligent traffic engineering frameworks [5]. From the centralized control, network policies with matching traffic engineering rules to forwarding devices are readily generated. Reinforcement learning enables network control and management to happen more quickly by fitting the modeling of the agent's behavior on the environment with rewards into the SDN network architecture.

III. Design

The overall design of the program is illustrated in this part

A. Explanation of Architecture

Figure 1 illustrates the topology architecture of an SDN (Software-Defined Networking) network. The diagram comprises five key components: the SDN controller, SDN switches, Server1, Server2, and the Client. The SDN controller, leveraging protocols such as OpenFlow, assumes the responsibility of managing the overall network topology and traffic control. It communicates with SDN switches via southbound interfaces, disseminating flow tables to guide packet forwarding. SDN switches, in turn, employ these flow tables to forward packets. Matching the header fields of the information with the flow labels, they determine the appropriate processing method for the packets based on the **actions** specified in the flow tables.

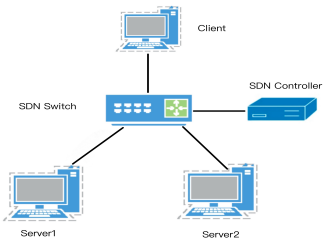


Fig. 1: The network topology

B. Workflow of Program

Fig.2. and Fig.3. respectively illustrate the processes of forwarding and redirecting. The overall framework implementation proceeds as follows: Initially, the client sends a data packet to the switch, which then parses the packet to extract Ethernet protocol information. Based on the Ethernet protocol, the source MAC address, destination MAC address, and switch ID are obtained. The source MAC address is then bound to the input port, storing the mapping relationship to prevent flooding in subsequent occurrences.

Next, based on the destination MAC address, the corresponding output port is identified. If the corresponding

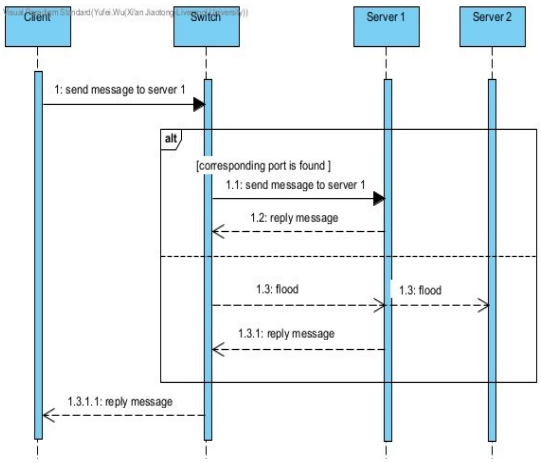


Fig. 2: The sequence diagram of forwarding

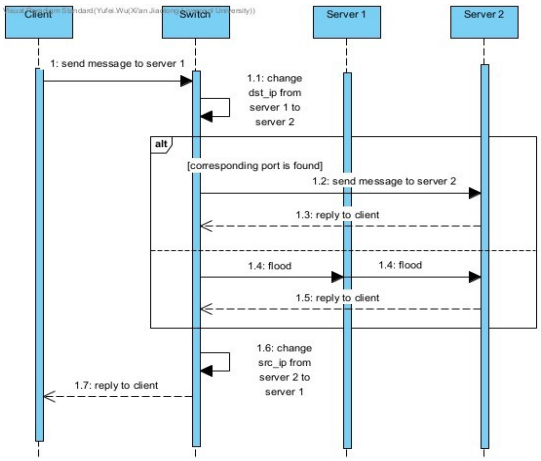


Fig. 3: The sequence diagram of redirecting

output port is not found in the mapping table, the switch will broadcast the packet to all hosts, known as flooding. If the corresponding output port is found, field matching is performed to determine the action for the packet. If the matching fails, the packet is sent to controller to handle, known as packet-in, the controller needs to add a new flow entry. If the matching succeeds, the packet is forwarded.

In the implementation of the redirect task, the switch needs to perform two IP address transformations. When the client sends a data packet to server 1, the switch needs to change the target IP from server 1 to server 2. Similarly, when server 2 responds to the client, the switch needs to change the source IP from server 2 to server 1. This will be detailed in the implementation section.

C. Algorithm

Algorithm 1 and Algorithm 2 pseudo codes describe the main steps for handling packets in SDN controller applications, specifically focusing on forwarding and redirecting functions.

.

Algorithm 1: *Forward Algorithm*

Input: *ev: message packet in event*
Output: *packet out message*

```
pkt ← packet.Packet(ev.msg);
eth ← pkt.get protocol(ethernet);
src_mac, dst_mac ← eth;
if dst_mac is known then
    out_port ← mac_to_ port[dst_mac];
else
    out_port ← FLOOD;
actions ← parser.OFPActionOutput(out_port);
if eth is IP then
    ipv4_src, ipv4_dst ← eth.get protocol(ipv4);
    if protocol is ICMP then
        match ← {eth, src_ip, dst_ip};
    if protocol is TCP then
        match ← {eth, src_ip, dst_ip, src_port,
          dst_port};
    add_flow, out_msg ← match, actions;
    out_msg.send;
```

Algorithm 2: *Redirect Algorithm*

Input: *ev: message packet in event*
Output: *packet out message while redirecting*

```
pkt ← packet.Packet(ev.msg);
eth ← pkt.get protocol(ethernet);
src_mac, dst_mac ← eth;
if eth is IP then
    src_ip, dst_ip ← eth.get protocol(ipv4);
    if src_ip is client[ip] and dst_ip is server1[ip]
     then
        if server2[mac] is known then
            out_port ← mac_to_port[server2[mac]];
        else
            out_port ← FLOOD;
        action ← Change dst_ip , dst_mac to
          server2's;
        match ← {eth, src_ip, dst_ip};
    if src_ip is server2[ip] and dst_ip is client[ip]
     then
        if client[mac] is known then
            out_port ← mac_to_port[client[mac]];
        else
            out_port ← FLOOD;
        action ← Change src_ip , src_mac to
          server1's;
        match ← {eth, src_ip, dst_ip};
    add_flow, out_msg ← match, action;
    out_msg.send;
```

IV. Implementation

A. Development Environment

The environment of development is shown in table 1.

B. Steps of Implementation

1) Take a close look at the coursework requirements to determine what has to be done.
2) To make the code's workflow clearer, create sequence diagrams depicting the forward and redirect interactions between the switch, client and servers.
3) To learn and understand how Ryu switching hubs work and operate, read the official documentation.
4) In accordance with the sequence diagram, develop the forward and redirect code.
5) Deploy the code and run tests on two virtual machines.
6) Add comments to the code to make it more readable.
7) Analyze the test results and finish the report.

C. Programming Skills

- **Modular Programming:** Follow modular programming principles when writing the code. The main idea behind modular programming is to divide a complex problem into smaller, more manageable parts, thus improving code reusability, avoiding overly lengthy code, and achieving more efficient development and better software design.
- **Debugging and Testing:** Proficiency in detecting and resolving coding errors as well as putting into practice efficient testing techniques guarantee the dependability and quality of software programs.

TABLE I: Development Environment

Field	Description
Operating System	Microsoft Windows 11
CPU	Gen Intel(R) Core(TM) i7-12700H
RAM	16G
IDE	Jetbrain Pycharm Professional 2023.1
Python Version	Python 3.9
Python Library	'socket', 'time', 'mininet', 'ryu',
SDN controller	Ryu 4.34

D. Actual Implementation

- **SDN network topology:** The SDN network topology is built using Mininet and the process is quite straightforward. Firstly, establishing a network topology object with the Mininet constructor and defining the Base IP and other pertinent information. Then, using the add method to create, link, and configure each host, switch, and Ryu controller of the network topology. After that , set the mac address and IP address of the client and the server. Finally, executing the network topology and make sure that it was constructed correctly.

- **Forward:** During the implementation of the forward process, the specific procedure is as follows: The Client sends a data packet to Server1, and the switch records the port corresponding to the Client at this time. Therefore, when Server1 responds to the data packet later, the switch can determine the port corresponding to the Client. Subsequently, as the switch is unaware of the port corresponding to Server1, it sends data packets to all switch ports except the one where the Client is located. Only Server1 responds in this process because the destination IP of the data packet is Server1. Next, the switch performs field matching on the data packet using the flow table to determine how to process the packet (in this task, the action is forwarding). It is worth noting that the idle timeout for flow table entries is set to 5 seconds to prevent flow table overflow. If the field matching fails, the data packet is handed over to the SDN controller for processing (packet-in). The controller creates a new flow table entry and sends a Packet_Out SDN message containing the TCP SYN segment Finally, after receiving the data packet, Server1 responds accordingly.
- **Redirect:** In the process of implementing redirection, the procedure becomes more complex, as all traffic sent by the client will be redirected to Server 2. From the client's perspective, it always communicates with Server 1. Therefore, the implementation idea is to have the switch change the source IP and destination IP. The implementation process is as follows: The client sends a data packet to Server 1, and at this point, the switch changes the destination IP to Server 2's IP. Similarly, during the first packet transmission, the switch is unaware of the port corresponding to Server 2, as it engages in flooding. In this process, only Server 2 responds because the destination IP of the data packet is Server 2. Next, Server 2 receives the data packet and responds. As mentioned above, from the client's perspective, it communicates with Server 1. All data packets sent by Server 2 will be treated as if sent by Server 1. Therefore, the switch changes the source IP from Server 2's to Server 1's at this point. It is important to note that during forwarding, the switch retrieves the source and destination port, as well as the source and destination IP addresses from the TCP protocol header to create matching fields and set actions. In contrast, during redirection, communication parties are differentiated based on the destination IP and source IP, and different actions are then set.

E. Difficulties and Solutions

The main difficulty we encountered is all about Ryu framework when developing the forwarding and redirecting part, because Ryu is a fully featured SDN controller framework, which can present challenges due to its complexity. To understand Ryu framework requires good understanding of network principles, and the features offered by Ryu and SDN. Further, the Ryu framework relies on the complex protocol OpenFlow, and it is critical to understand and effectively use OpenFlow concepts such as flow tables, matching fields, and other operations. To overcome this challenge, we have carefully studied the Ryu framework, OpenFlow protocol and SDN concepts to solve the difficulties that arose during the development process, based on official documentation and the support from community such as csdn.

V. Testing and Results

A. Testing Environment

TABLE II: Testing Environment

Field	Description
Operating System	Microsoft Windows 11
CPU	Gen Intel(R) Core(TM) i7-12700H
RAM	16G
IDE	Jetbrain Pycharm Professional 2023.1
Python Version	Python 3.9
Python Library	'socket', 'time', 'mininet', 'ryu',
SDN controller	Ryu 4.34

The testing environment is shown in table 2, which is identical to the development environment.

B. Test Plan

1) Make sure the SDN network topology is created as expected. Run the $networkTopo.py$ file created by Mininet python library. According to the task requirement, check the host, switch and controller meet the requirement, and the IP and MAC address are correct.

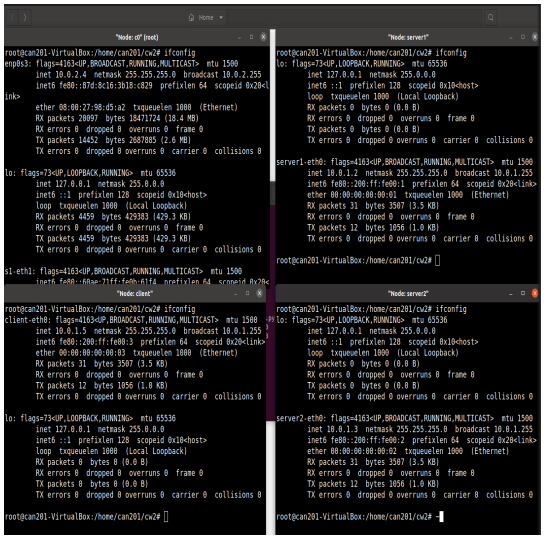


Fig. 4: Test for task 1

2) For task2, run the SDN controller and check if every node is reachable with each other using the pingall command. Moreover, make sure any flow entry has already set an idle timeout of 5 seconds by running the command $sudo\ ovs-ofctl\ -O\ OpenFlow13\ dump-flows\ s1$ in switch's terminal. The implementation of task2 are included in Fig. 5, Fig. 6, Fig. 7 and Fig. 8 below.

3) Run the $ryu\_forward.py$ program on the controller terminal according to Tasks 3, 4.1, and 4.2. Then, run the provided $server.py$ program on the terminals of server1 and server2. Finally, run the provided $client.py$ program on the client side to forward traffic to server1. After that, using pingall command to ping each other. Remember to set up Wireshark to listen, record network packets, and to calculate latency for later analysis.

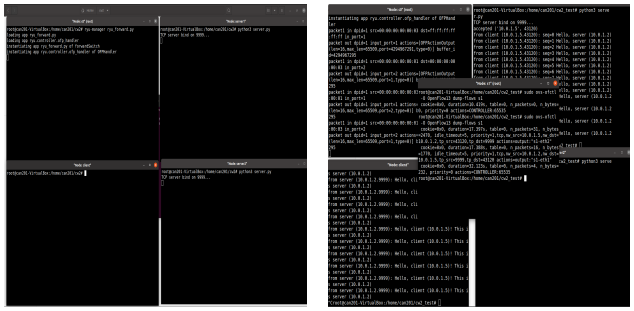


Fig. 5: results for forwarding process

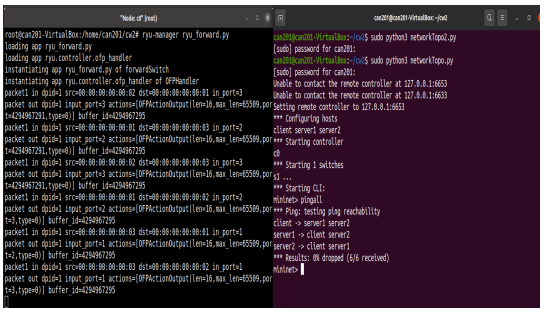


Fig. 6: pingtest for forwarding

4) In this step, run the SDN controller application $ryu\_redirect.py$ on the controller's terminal in order to redirect traffic from the client to server1 to server2. The remaining steps are the same as in step 3. It is worth mentioning that it takes five seconds after the completion of step 3 to perform step 4, because the idle timeout of flow entries is 5 seconds. Wireshark is also needed to listen and record packets.

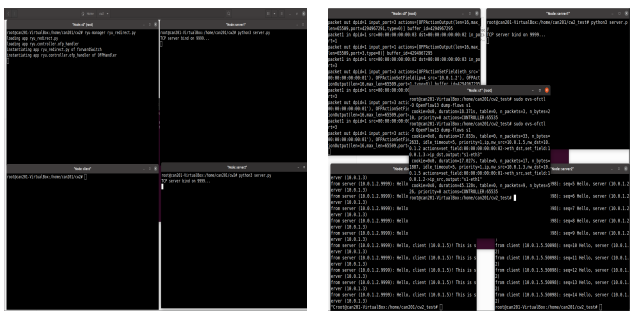


Fig. 7: results for redirecting process

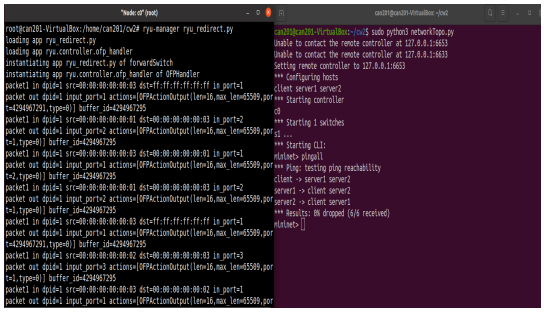


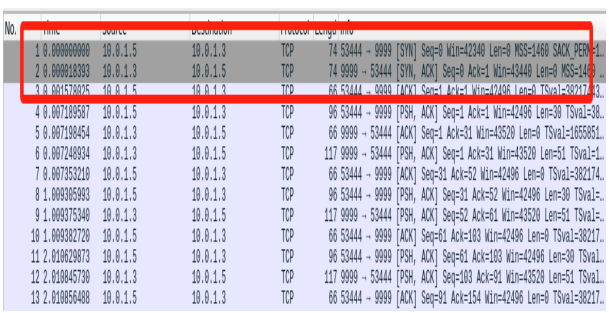
Fig. 8: pingtest for redirecting

5) Repeat steps 3 and 4 to collect more network latency data to reduce analysis errors.

C. Test Result

All the requirements in the tasks have been displayed in figures above. An SDN network topology along with one client, two servers and a controller has been established successfully and all nodes have no problem communicating with each other. Moreover, client and two server all have correct IP and MAC addresses that satisfy specified requirements which can been seen in Fig. 4. From Fig. 5 and Fig. 7, they shows flow entries with 5-seconds idle timeout are correctly set and $client.py$ and $server.py$ can run favorably on Client and Servers. Running $ryu\_forward.py$ program on the controller terminal, then in Fig. 5, traffic from Client to Server1 is shown to be forwarded correctly. On the other hand, after running $ryu\_redirect.py$ on the controller, the whole flow path from Client to Server1 is changed and redirected to Server2 without the Client's knowledge successfully. It means that the program implements the goal of disguise on packet sources, so that the Client considers that these packets are sent from Server1.

Comparing the logs in two controller terminal which are forwarding and redirection, the main difference can be seen clearly. Forwarding directly transmits packets from input port to the correct output port according to the flow table in the switch. However, in the redirection, the controller can change the switch's flow table automatically which realize the automatic adjustment on the traffic to lead the packet to Server2. This action is executed by $OFPActionSetField$ that is shown in Fig. 7.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.1.5	10.0.1.3	TCP	74	53444 → 9999 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1...
2	0.000818393	10.0.1.3	10.0.1.5	TCP	74	9999 → 53444 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 ...
3	0.001578025	10.0.1.5	10.0.1.3	TCP	66	53444 → 9999 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=382717...
4	0.007189587	10.0.1.5	10.0.1.3	TCP	96	53444 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=42496 Len=30 TSval=38...
5	0.007198454	10.0.1.3	10.0.1.5	TCP	66	9999 → 53444 [ACK] Seq=1 Ack=31 Win=43520 Len=0 TSval=1655861...
6	0.007249834	10.0.1.3	10.0.1.5	TCP	117	9999 → 53444 [PSH, ACK] Seq=1 Ack=31 Win=43520 Len=51 TSval=1...
7	0.007353219	10.0.1.5	10.0.1.3	TCP	66	53444 → 9999 [ACK] Seq=31 Ack=52 Win=42496 Len=0 TSval=382174...
8	1.000305993	10.0.1.5	10.0.1.3	TCP	96	53444 → 9999 [PSH, ACK] Seq=31 Ack=52 Win=42496 Len=30 TSval=...
9	1.000375340	10.0.1.3	10.0.1.5	TCP	117	9999 → 53444 [PSH, ACK] Seq=52 Ack=61 Win=43520 Len=51 TSval=...
10	1.000382726	10.0.1.5	10.0.1.3	TCP	66	53444 → 9999 [ACK] Seq=61 Ack=103 Win=42496 Len=0 TSval=38217...
11	2.010629873	10.0.1.5	10.0.1.3	TCP	96	53444 → 9999 [PSH, ACK] Seq=61 Ack=103 Win=42496 Len=30 TSval...
12	2.010845730	10.0.1.3	10.0.1.5	TCP	117	9999 → 53444 [PSH, ACK] Seq=103 Ack=91 Win=43520 Len=51 TSval...
13	2.010856488	10.0.1.5	10.0.1.3	TCP	66	53444 → 9999 [ACK] Seq=91 Ack=154 Win=42496 Len=0 TSval=38217...

Fig. 9: wireshark example on redirection

To calculate the networking delay, our team used Wireshark on Client to capture the TCP packets and packets with SYN, SYN ACK and ACK segments can be found in the first three rows that is shown in Fig. 9. In the test, our team uses 20 packets to transmit and apply curve chart for comparison. For these 20 packets transmitting, conduct one five-fold cross-validation to compute the average on each group and a second cross-validation is then performed on the means of these groups to obtain the overall average. As a result, the average latency of forwarding and redirection are roughly 0.0005s and 0.0010s separately which are displayed in Fig. 10.

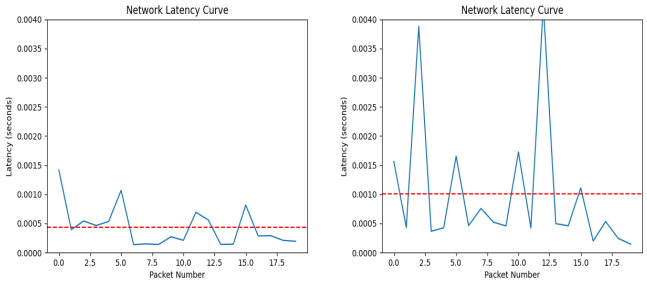


Fig. 10: curve graphs on forwarding and redirection

This result indicates that average latency in redirection is twice as long as that on forwarding. Furthermore, compared to the curve on forwarding, network latency curve has large fluctuations and variances which means that the network is less stable. This analysis results can be explained by the fact that redirection involves additional requirements and actions when handling TCP protocol, since the two communicating parties are distinguished through the source and destination IP addresses. Moreover, in order to execute redirection, SDN controller is required to modify flow entries dynamically which may cause the reinstallation delay of the flow entry. In order to improve the network latency performance on redirection, effective flow entry delivery algorithm can be applied to reduce unnecessary messages in the control plane. Moreover, when running redirection, the shortest path algorithms such as Dijkstra and Bellman-Ford algorithm can be utilized to shorten the path of the packets transmission which can also decrease latency.

VI. Conclusion

To sum up, our group successfully finished all five of the coursework's tasks and compared the latency between using redirecting and forwarding. To begin, we constructed the SDN network topology. Then, we designed two SDN controllers to achieve forwarding and redirecting function based on the Ryu framework. During testing, we make sure the whole process can run successfully in the environment of virtual machines. Moreover, we calculated the latency to get a better understanding about the performance of the two functions.

Network security becomes more and more important. Thus, in the future, we can enhance network security by altering the matching details of flow table entries or adding a filter to prevent packages from a specific IP address.

Acknowledgment

References

[1] A. Ominike Akpovi, A. Adebayo, and F. Osisanwo, "Introduction to Software Defined Networks (SDN).," International Journal of Applied Information Systems, pp. 10-14, 2016.

[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2014.

[3] S. Scott-Hayward, C. Kane, and S. Sezer, "Operationcheckpoint: Sdn application control," in 2014 IEEE 22nd International Conference on Network Protocols, 2014: IEEE, pp. 618-623.

[4] S. Agarwal, M. Kodialam and T. V. Lakshman, "Traffic engineering in software defined networks," 2013 Proceedings IEEE INFOCOM, Turin, Italy, 2013, pp. 2211-2219, doi: 10.1109/INFCOM.2013.6567024.

[5] Dake, Delali & Gadze, Dzisi & Klogo, Griffith & Nunoo-Mensah, Henry. (2021). Traffic Engineering in Software-defined Networks using Reinforcement Learning: A Review. International Journal of Advanced Computer Science and Applications. 12. 10.14569/IJACSA.2021.0120541.