CAN201 Coursework Part II Report

1st
Student Name: Yifei Du
Student ID: 2035567

2nd
Student Name: Hao Zhong
Student ID: 2037707

3rd
Student Name: Xinyi Liu
Student ID: 1928951

***Abstract*—Software-Defined Network (SDN) technology is a network management approach which improves network performance and management efficiency through dynamic, programmable network configuration. This project aims to emulate SDN forwarding and redirection traffic control functions using Python Ryu framework, and test them using a Mininet-based network topology as the virtual testbed. Besides, Wireshark is also used as a testing tool to evaluate the function performance. Furthermore, a handful of relevant papers are also reviewed to investigate potential application areas for the proposed design and similar implementations of redirection that have been proposed. Finally, based on the results of this project and knowledge from the reviewed papers, we also summarize the limitations and shortcomings of our proposed controllers, and make recommendations for future improvements to this project.**

***Index Terms*—SDN, Ryu Framework, Traffic Forwarding, Traffic Redirect, Mininet, OpenFlow, Network Topology**

I. Introduction

Software-defined networking (SDN) technology is a network management approach, which enables dynamic and programmable network configuration to improve network performance and management efficiency [1]. This project aims to simulate two SDN traffic control functions (forward and redirect) on a simple SDN network topology by using SDN flow entry. To accomplish this goal, several challenges should be resolved. First, a SDN network topology needs to be constructed using Mininet, which will be used to run the emulated SDN traffic control functions. Second, based on this topology, two controllers using Ryu framework need to be designed and implemented to perform forwarding and redirection traffic control functions respectively.

These proposed traffic control functions of Ryu controllers may have numerous potential applications: First, the redirection function has utility in load-balancing applications [2]. Ryu SDN controller can redirect overloaded network traffic to less loaded path so that it can reduce networking latency and improve user experience [2]. In addition, Ryu controllers also have a great advantage in secure traffic control. This means that the controller could redirect and alert administrators when suspicious packets are detected. This can be accomplished using a condition comparison; if this condition meets the defined rules, it will allow access to the network; otherwise, the system will redirect these packets and alert the administrator [3].

We design and implement a SDN controller based on Ryu framework, which emulates a forwarding traffic control function by managing the flow entry on the SDN switch. Moreover, a SDN controller with redirection traffic control function is designed and implemented by us based on the previous forwarding one. And then, these two SDN controllers are tested on a SDN topology built on Mininet. The given server and client Python files are used to test if the forwarding and redirection functions works well. Furthermore, Wireshark is also used as one testing tool to capture the packets and calculate the networking latency.

II. Related Work

Literature on networking redirection was review, and among them, one similar application that related to improving the quality of network services and one application that facilitate the detection of malicious redirection are studied as follows.

A. Improving the quality of network services

First, Hwang and Liem et al [4] proposed a new architecture for local traffic redirection based on ONU mechanism, which improves EPON performance for P2P multimedia services, and with this architecture and their proposed REDIRECT DBA scheme, it is able to keep the video traffic delay below 5ms and also reduce the packet loss probability.

B. Facilitating the detection of malicious redirection

Second, the detection of malicious redirection (e.g., provide users with advertisements instead of what they want) has become an important technique [5]. Hans et al.'s [5] proposed five new factors to facilitate the detection of redirected spam. And according to their test results, it is more effective than traditional spam filtering after applying the new elements.

III. Design

An overview of the SDN network system design is presented in Figure 1. This system is comprised of a SDN controller, a SDN switch, a client, two servers (Server1 and Server2). The client and servers can exchange network packets with given IP and MAC addresses (addresses are shown in Figure 1) through the SDN switch, which forwards their packets to the corresponding ports of the destined hosts. In addition, the SDN controller manipulates network traffic by managing the configuration of the SDN switches.

Figure 2 shows the workflow of the designed SDN controller. The SDN controller could manage the SDN switch according to the presented workflow, and use it to instruct the switch on the forwarding process. The detailed description of this workflow is shown below.

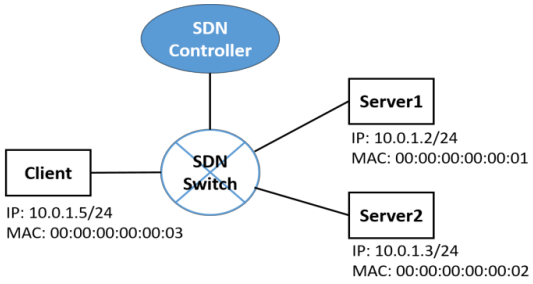

Fig. 1. SDN network system design diagram.

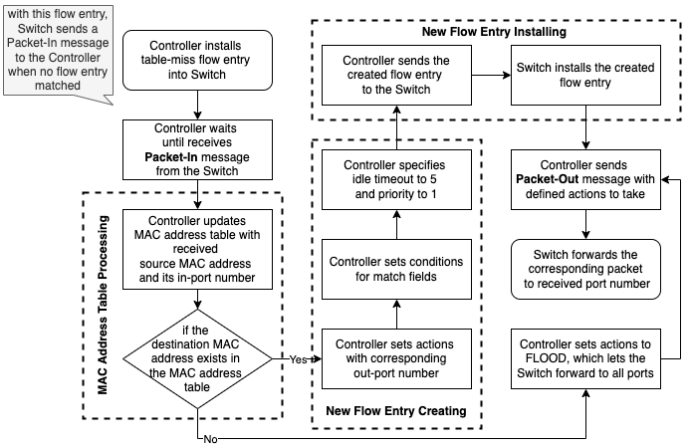

Fig. 2. The workflow of uploading operation.

A. Table-miss entry installing

This controller first installs a table-miss flow entry with the lowest priority to the SDN switch. With it, the SDN switch can send a Packet-In message to the SDN controller to request forwarding instructions when there is no flow entry match.

B. MAC address table processing

Meanwhile, the SDN controller maintains a MAC address table; whenever the SDN controller receives a Packet-In message, it first adds its source MAC address, switch ID, and receive port to the MAC address table, and then checks whether the destination MAC address is available in its MAC address table for the given switch ID. If yes, it first installs a new flow entry to the SDN switch, and then specifies the port number to forward by sending a Packet-out message back to the SDN switch; otherwise, it instructs the SDN switch to forward using flood mode by sending a Packet-out message back to the SDN switch. The flood mode instructs the SDN switch to forward that packet to all available ports.

C. New flow entry creating and installing

To install a new flow entry to the SDN switch, the controller needs to first creates a new flow entry: it sets the match fields and actions to take when the packet matches, and then assigns a priority of 1 to ensure that this entry will be used before table-miss entry; and finally, idle timeout should be set to 5 as the task specification. After creating a new flow entry, the SDN controller sends and installs it to the corresponding SDN switch.

D. Redirection function

In addition to instructing switches to forward, in this project the SDN controller is also designed to implement redirection functions for specific network flows by managing SDN switches. This redirection function redirects TCP segments sent from Client to Server1 to Server2 and does not interfere with network traffic in other directions or protocols.

Algorithm 1 shows the kernel pseudo codes of the redirection function design. It consists of two parts:

Algorithm 1 Network traffic redirection function

Require: SDN Controller and SDN Switch
SDN Controller Initialisation:
1: SDN switches are configured with table-miss entry
SDN SDN Controller processes PacketIn messages:
{Processing MAC address table}
2: **if** EtherType is IP **then**
3: **if** pktProto is TCP **and** srcMAC = cltMAC **and** dstMAC = se1MAC **then**
4: **if** se2MAC in MAC address table **then**
5: *outPort* ← corresponding port number in MAC address table
6: **else**
7: *outPort* ← FLOOD
8: *actions* ← set dstMAC to se2MAC, set dstIP to se2IP, set out-port number as *outPort*
9: **end if**
10: **else if** pktProto = TCP **and** srcMAC = se2MAC **and** dstMAC = cltMAC **then**
11: **if** cltMAC in MAC address table **then**
12: *outPort* ← corresponding port number in MAC address table
13: **else**
14: *outPort* ← FLOOD
15: *actions* ← set srcMAC to se1MAC, set srcIP to se1IP, set out-port number as *outPort*
16: **end if**
17: **end if**
18: **end if**
{Creating and installing new flow entry}
19: **if** *outPort*! = FLOOD **then**
20: **if** pktProto = TCP **and** srcMAC = cltMAC **and** dstMAC = se1MAC **then**
21: *match* ← EtherTypeIP, srcIP, dstIP
22: *actions* ← set dstMAC to se2MAC, set dstIP to se2IP, set out-port number as *outPort*
23: **else if** pktProto = TCP **and** srcMAC = se2MAC **and** dstMAC = cltMAC **then**
24: *match* ← EtherTypeIP, srcIP, dstIP
25: *actions* ← set srcMAC to se1MAC, set srcIP to se1IP, set out-port number as *outPort*
26: **else**
27: *match* ← EtherTypeIP, srcIP, dstIP
28: **end if**
29: Add a new flow entry to corresponding SDN Switch
30: **end if**
31: Send a Packet-Out message to corresponding SDN Switch

Part I (line 2-line 18) is part of the processing MAC address table. This part only handles TCP packets from Client to Server1 or Server2 to Client. For the former, it instead uses Server2's MAC address instead of Server1 to find a match in the MAC address table, and if there is a match, uses the matched port as the forwarding port; if there is no match, uses FLOOD mode and changes the destination MAC and IP address to Server2's address. For the latter, it uses the Client's

MAC address to find a match in the MAC address table, and if there is a match, uses the matched port as the forwarding port; if there is no match, uses FLOOD mode and changes the source MAC and IP address to Server1's address.

Part II (line 19-line 31) is part of creating and installing new flow entry. This part only handles non-FLOOD TCP packets from Client to Server1 or Server2 to Client. Non-FLOOD means that there is an explicit port to forward these packets and we could utilize their information to create flow entries for SDN switches to avoid sending Packet-In message next time. We handle different situations respectively: Client to Server1, Server2 to Client and others. This ensures different actions when the flow entry matches. Finally, this new flow entry is sent to the SDN switch and installed. And a Packet-Out message is also sent to instruct the forwarding port of the current packet.

IV. Implementation

A. Implementation Environment

For purpose of implementing the designed SDN traffic control functions, one virtual machine is used as testing prototype with its specification: Ubuntu 20.04.4 LTS 64-bit Operating System, AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30GHz CPU, 2GB Memory and VirtualBox Network Address Translation Network Card. And its corresponding host environment is Window 10 Operating System, AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30GHz CPU, 8GB Memory and Intel PRO/1000 MT desktop Network Card. In addition, Visual Studio Code is used as our IDE and two Python3 libraries (Ryu and Mininet) are used for implementing SDN traffic control functions and SDN network topology respectively. Moreover, Object-Oriented Programming skill is also used in this project. The specific program flow chart shows the brief ideas of forwarding implementation steps in Figure 1.

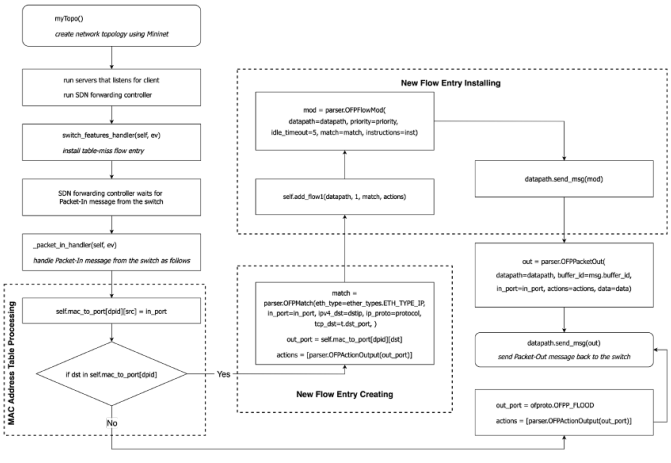

Fig. 3. The flow chart of implementing uploading operation.

B. Traffic forwarding function

The underlying virtualization testbed is implemented using Python Mininet based on the pre-designed system topology, and this network topology need to be executed firstly. And then, the controller application can be implemented based on it. When the forwarding controller application is executed, it first installs a table-miss flow entry to the SDN switch and waits for Packet-In messages. Further, the servers and client are executed sequentially; thus, client tries to establish a TCP connection with server1, and the SDN switch sends Packet-In messages when it has no matched flow entry to forward these TCP packets. When the SDN controller receives a Packet-In message from the SDN switch, it updates the MAC address table with the received source MAC address, switch ID and corresponding receiving port number. After that, it checks if the destination MAC address of the corresponding switch ID is in the MAC address, creates and installs a new flow table to that SDN switch if it exists, and sets the out-port number to the found one; otherwise, it specifies the out-port as FLOOD to send that packet to all available ports. This out-port is instructed via a Packet-Out message sent by the SDN controller after all previous processes. Therefore, the corresponding SDN switch can then forward that packet to port(s) according to the Packet-Out message.

C. Traffic redirection function

In this project, we also implement an SDN controller function that redirects TCP segments sent from the client to server1 to server2. That is to say, the client does know the existence of server2, and it only knows server1 and communicates with it. The implemented SDN controller could perform the redirection function to redirect this traffic to server2. This redirection workflow can be summarized as follows:

1) The controller chooses those TCP packets from client to server1 or server2 to client.

2) The controller determines if there is a match in the controller's MAC address table for the redirected destination MAC address. Allocate the corresponding out-port number if there is, otherwise, use flood mode and change destination from server1 to server2 if sent from client to server1 or change source from server2 to server1 if sent from server2 to client.

3) The controller creates a new flow entry and installs it to the switch if a specific out-port number can be found. Before creating it, the controller needs to set the match fields and actions to change the source or destination just as step 2.

4) The controller send a Packet-Out message to specify the out-port of that packet and actions to take (change source or destination).

D. Difficulties and our Solutions

One major problem we met is how to redirect the reply packets sent from server2 to client. At the beginning, we only implemented redirection function that redirect the packets sent from client to server1 to server2. This led to a TCP connection error when we tested it on the network topology. We found that the SYN packet sent from client can be correctly redirected

to server2, and server2 can reply the corresponding SYN-ACK. However, the TCP connection cannot be established. To solve this problem, we found that the destination of the SYN segment does not match the source of SYN-ACK segment. Therefore, it is found that the reply TCP segment from server2 to client needs also to be considered, so we added a redirection to modify the source MAC and IP address from server2 to server1. After this situation is considered, the TCP connection can be successfully established and traffic can be successfully sent.

V. Testing and Results

The same testing environment as host implementation is used to test the implementation and show the results.

A. Forwarding case

1) Run networkTopo.py in the terminal of the Virtual Machine to create the SDN topology. Five terminal windows pop up at the same time, namely, Controller, Client, Server1, Server2, and the Switch s1. The correct IP and MAC addresses for Client, Server1, and Server2 are shown in Figure 4.

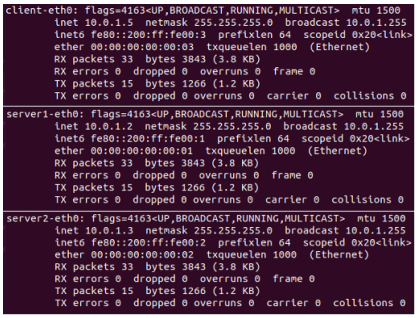

Fig. 4. The IP and MAC addresses of Client, Server1, Server2.

2) Run ryu_forward.py in the controller and ping Server1 and Server2 using Client. The result presented in Figure 5 shows that the client can ping Server1 and Server2 successfully.

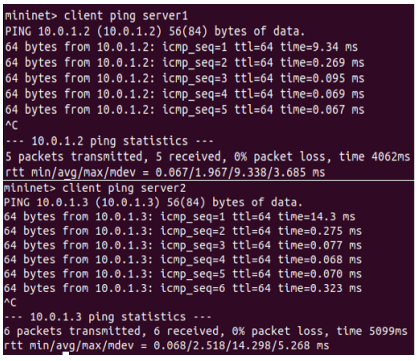

Fig. 5. The result of authorization obtaining.

3) After waiting for 5 seconds, the previously caused flow entry's idle timeout, run server.py on Server1 and Server2, and run client.py on Client. The flow table in Switch s1 is shown in Figure 6. The result of the traffic sent from the Client to Server1 is shown in Figure 7.

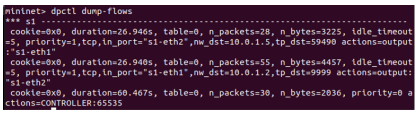

Fig. 6. The result of the flow table in Switch s1.

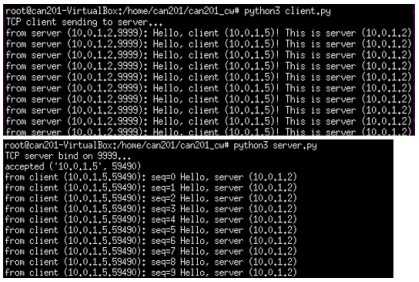

Fig. 7. The result of the traffic sent from Client to Server1.

B. Redirection case

1) Redirection case use the same network topology as forwarding case.

2) Run ryu_redirect.py on the Controller, and ensure the Client can ping server1 and server2 successfully. The result of this step is shown in Figure 8.

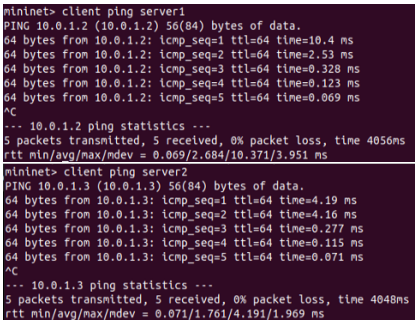

Fig. 8. The result of the Client ping Server1 and Server2

3) Repeat step 3 in forwarding case, and gain the flow table in Switch shown in Figure 9, and the results of the traffic sent from Server2 and Client in Figure 10.

Fig. 9. The result of the flow table in Switch s1.

C. Performance testing

In this part, Wireshark is used to capture packets and calculate the network delay caused by the TCP three-way handshake (from the first SYN segment till the last ACK

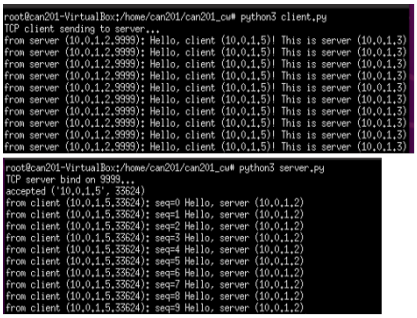

Fig. 10. The traffic sent from Client redirected to Server2.

segment) for these two cases. Further, we want to evaluate the average performance of them with the help of Wireshark.

1) Forwarding case: The obtained network delay is presented as a curve in Figure 11. According to this curve, the overall distribution of the latency remains stable, and most of them are concentrated between 8-9ms.

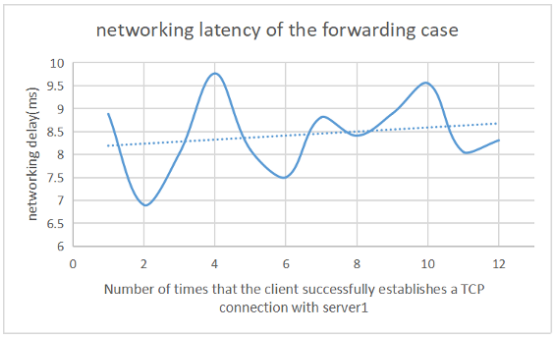

Fig. 11. The networking delay of forwarding case

2) Redirection case: The obtained network delay is presented as a curve in Figure 12. According to the image, network delay shows a continuous upward trend. The networking delay of this part of the network is mostly above 15ms, and can reach 24ms at the highest during the test. This average latency is much larger than the forwarding case. One possible reason for the relatively long network latency in this part might be that it includes the time to capture the packet and ask the controller to send the address.

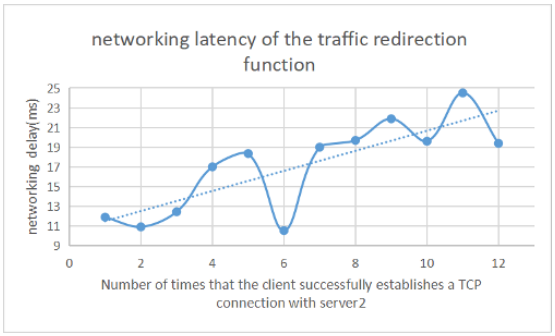

Fig. 12. The networking delay of redirection case

VI. Conclusion

To conclude, in this coursework project, we first designed two traffic control functions (forwarding and redirection) and a network topology to test them. And then, we constructed two SDN controllers using Ryu framework to implement those traffic control functions respectively. Furthermore, we tested the traffic control functions of these two SDN controllers on the Mininet-based network topology we implemented using the given client and server file. Besides, Wireshark was also used to carry a performance test; it was used to capture packets and calculate the average latency caused by TCP 3-way handshake. According to the results, the latency of redirection is much higher than forwarding function, and one possible reason was given. Moreover, we also reviewed related literature to understand the application of its related technologies in various fields.

However, there are still some shortcomings and limitations. First, one of the limitations is the small volume of tests, and network fluctuations may lead to errors in the experimental results. In the future, the test volume can be increased by extending the test time to minimize the error. Second, one of the problems of the designed SDN controllers is that it does not support simultaneous multitasking, therefore, the design of the SDN controllers could be further improved. Multi-threading function can be added to support simultaneous processing of requests from multiple switches.

VII. Acknowledgment

All group members contribute the same percentage to this project.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[2] Y. Xu et al., "Dynamic Switch Migration in Distributed Software-Defined Networks to Achieve Controller Load Balance," in IEEE Journal on Selected Areas in Communications, vol. 37, no. 3, pp. 515-529, March 2019, doi: 10.1109/JSAC.2019.2894237.

[3] Y. Gautam, B. P. Gautam and K. Sato, "Experimental Security Analysis of SDN Network by Using Packet Sniffing and Spoofing Technique on POX and Ryu Controller," 2020 International Conference on Networking and Network Applications (NaNA), 2020, pp. 394-399, doi: 10.1109/NaNA51271.2020.00073.

[4] I. -S. Hwang and A. T. Liem, "A multimedia services architecture supporting local traffic redirection in passive optical network," The 2012 11th International Conference on Optical Communications and Networks (ICOCN), 2012, pp. 1-4, doi: 10.1109/ICOCN.2012.6486245.

[5] K. Hans, L. Ahuja and S. K. Muttoo, "Significant factors for detecting malicious redirections," 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), 2017, pp. 1-4, doi: 10.1109/TEL-NET.2017.8343584.