SDN-based traffic control implementation project

*CAN201-CW-Part-II-RuijiaTian-ShuranZhang-ShuChenYuan

1st Ruijia Tian
dept. ICS:
Information and Computing Science
2035825
Ruijia.Tian20@student.xjtlu.edu.cn

2nd Shuran Zhang
dept. ICS:
Information and Computing Science
2033612
Shuran.Zhang20@student.xjtlu.edu.cn

3rd Shuchen Yuan
dept. ICS:
Information and Computing Science
2036501
Shuchen.Yuan20@student.xjtlu.edu.cn

I. ABSTRACT

This project's primary objective is to implement SDN-based features for traffic control. In this project, a network emulator called 'Mininet' which allows to create a virtual network topology with switches, hosts, and links, and then run network protocols and applications on top of this virtual network is applied to design a network topology that satisfies the specifications. Based on this, a software-defined networking (SDN) framework called Ryu was constructed, which provides a python-based interface for interacting with OpenFlow networks and constructs two servers, a client and a controller as topology nodes. In this case, the project is design to implement two functionalities. One is for the packet forwarding which can forward the traffic sent from the client to specified server correctly, and the other one is the packets redirection which enables the redirection of traffic sent from client to a specified server to another server. In addition, Wireshark is used to evaluate the network latency of direct forwarding and redirection scheme.

II. INTRODUCTION

A. Background and Contribution

SDN technology is built on the concept of decoupling the control plane from the forwarding plane, allowing network managers to easily govern and modify network behavior using enhanced interfaces, making SDN network management more flexible and dynamic. SDN has emerged as a new avenue in the development of dynamic networks in recent years. This report implements the SDN network topology and simple traffic control functions based on a given client and server. Each node in this system is capable of communicating with each other, and traffic sent to server1 may be delivered directly to server1 or redirected to server2 without the client being aware of the redirection. The paradigm has promising applications for improving network throughput and enhancing network security. Previous research has supported these applications; for example, studies [1] and [7] illustrate how to increase TE performance with hybrid SDN and how to separate aberrant traffic from typical traffic with dynamic traffic management functions.

B. Challenge

The main challenge for this project is building up the controllers for the forwarding case and the redirect case. For the forwarding scheme, the challenge is to create implementation that a controller that can instruct the switcher to define the port to send back based on the port of the received packet so that the packet sent by server1 can be forwarded back to it. To solve this problem, after checking the official Ryu documentation, a switching hub was created based on Ryu's source tree. Similarly, for the redirection implementation, a small modification was made to the rules after applying the lab11 code. However, the problem arose that the redirected code would run and respond much slower, with a delay of 5-10s each time, if the code for the forward scenario had been run previously. We checked the state of the flow table on the VM to resolve this issue and found that the default mac address was unreasonable. After correcting the default mac address, the code ran smoothly and the response time was negligibly rapid.

III. RELATED WORK

Thanks to the flexibility of SDN, many solutions to improve response efficiency and reduce response time have been proposed. The study [6] proposes an optimization algorithm based on SDN flow redirection, which minimizes the maximum response time on the controller. This algorithm can reduce the maximum controller response time by about 50 percent-80 percent compared to the traditional static/dynamic method. Study [4] then does a summary work on the recent progress of SDN flow engineering, focusing on four major priorities: flow management, fault tolerance, topology update, and flow analysis/characterization. [3] is a survey on distributed SDNs. In addition to traffic engineering, another focus of SDN research is security measures. Study [5] introduces a virtual firewall ACLSwitch based on SDN features. Compared to traditional firewalls erected at the border, this new design can better detect attacks from inside the network and allow different switches to deploy different filtering configurations. [2] presents a scheme to prevent network attacks against controllers, which provides an important new perspective because once a controller is attacked, the whole network will be manipulated.

IV. Design

This project is divided into three main parts in designing:

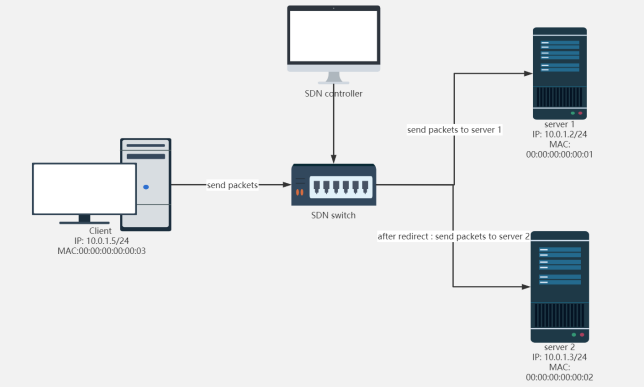

Fig. 1. C/S network architecture diagram

- *PART 1*
  Create a network topology that connects the clients, the 2 servers and the switch.
  The first part is easy to solve, we use mininet to build a network topology that meets the conditions, first join the nodes (controller, client, server, switch), connect the nodes, build the network and assign a mac and IP to each node, and finally assign a virtual terminal to each node.
- *PART 2*
  Write the Ryu controller program so that each node can be connected to each other and the controller can generate the correct flow entries so that all traffic sent from the client to server1 is forwarded to server1.
  The second part is to design a SDN controller based on Ryu framework. when the client first request sends data to server1, the controller orders the switch to broadcast the request to all hosts (here including server1 and server2), at this time the controller does not generate any flow entries, only learns the client mac. When server1 receives the request, it will reply to the client, and when the reply goes through the switch and controller, it finds that the mac of the destination (client) have been learned, at this time, the controller generates a flow entry, adds it to the flow table and forwards the data directly to the client. When the client requests to send a message to server1 again, the mac of server1 have been learned in the previous reply step, so the controller generates another flow entry to the switch and forwards the data directly to server1. In addition, a five-second idle timeout is set while the flow entry is installed. If no packets match within five seconds, this will cause the flow entry to expire.
- *PART 3*
  Modified the Ryu controller program so that each node can be connected to each other and the controller can generate the correct flow entries in the switch and all traffic originally sent from the client to server1 is redirected to server2, which results in two types of effects:
  (1) When packets sent from the client to server1 is detected, the destination IP address and mac address are altered to server2.
  (2) When a reply from server2 to the client is detected, the source IP and mac of the message are changed to server1, misleading the client into thinking the reply is from server1.
  This way, the message that should have been sent to server 1 is correctly transferred to server 2. We modify the server 2 response so that the client is unaware that the message was sent to server 2 rather than server 1.

V. Implementation

A. Test Host Environment

TABLE I
Test Host Environment Introduction

CPU	Memory	Operating System
Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz	64-bit operating system x64 based processor	RAM 16.0 GB

Object-oriented programming (OOP) was utilized in our work, which made the code more flexible, portable, and manageable. OOP not only makes code easier to read, but it also facilitates teamwork.

B. Concrete Application

- *PART 1*
  For creating such a topology, the “Mininet” library is used in this case. Based on the given function in the “Mininet” library, we first specify the network prefix of all IP addresses in the virtual network. Then, a remote controller was added and given the name ”c1” . Similarly, three host nodes were added to the network, each with an assigned name, IP address and MAC address according to the requirement. Links are added between the switch and the three host nodes. These links represent virtual Ethernet connections between the nodes. Based on the above, a simple SDN network topology has been created.
- *PART 2*
  This section will focus on the basics of simple switch 13.py. In the design section, we have briefly described the basic behavior of this controller application using a sending and receiving example. However, it is not clear how this set of behaviors occurs, i.e., how the controller’s internal code reacts differently to different packets (e.g., when to add flow entries, etc.). This will be explained in more detail in this section. The core judgment structure of the code will be explained first, followed by a detailed explanation of each of the main cases in the sending and receiving process.
  Two core judgments of the code: whether the buffer id is available and whether the destination address has been learned, determine how the controller can reasonably respond to different data and requests. As shown in Figure 2 and 3, the former determines whether the controller needs

to redirect the data portion down to the switch, or simply specify the buffer id. The latter determines whether the request needs to be flooded to every host connected to the switch, or delivered only to the target host. In addition to the judgment module that classifies messages, there are three key parts of the code that implement the underlying functionality: match and addflow implement the tasks of defining match rules and adding flow entries, respectively, while the self-learning part automatically records the address information of packets arriving at the controller. The main possible scenarios in the real application are shown in Figure 4.

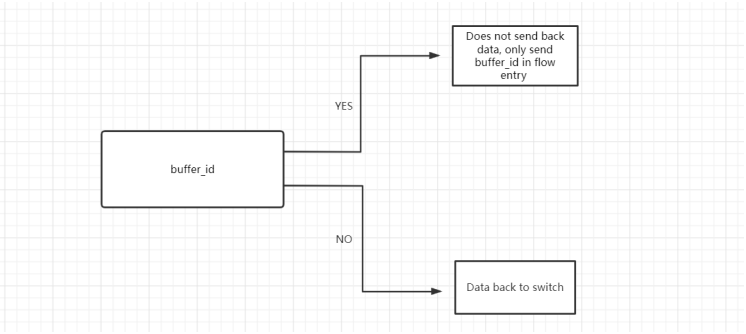

Fig. 2.

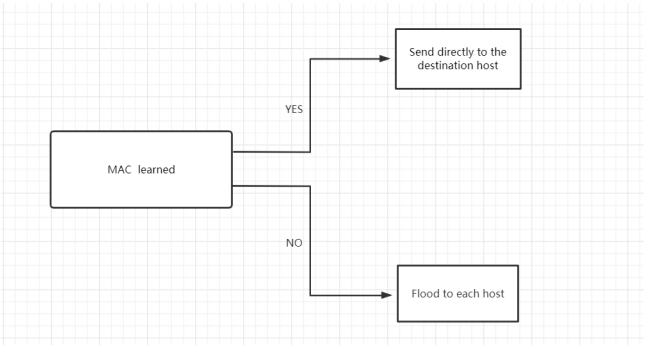

Fig. 3.

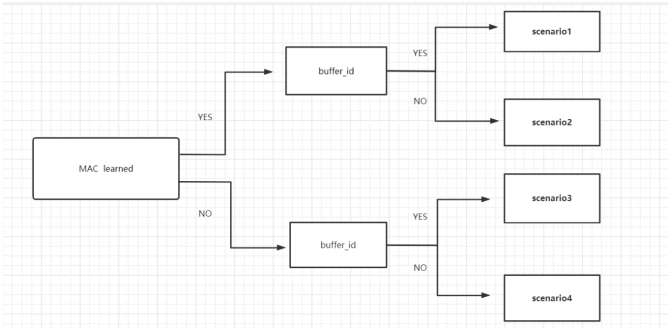

Fig. 4.

In case 1, the address is learned and the buffer id is used. adding a flow entry and marking the buffer id, the controller does not need to pass the data back to the switch and the data is sent directly to the destination host.

In case 2, the address is learned and no buffer id is used. buffer id adds a flow entry and does not mark the buffer id, the controller has to redirect the data back to the switch and the data is sent directly to the destination host.

In case 3, the address is not learned, buffer id is used. no flow entry is added, only the address is recorded, the controller does not have to pass the data back to the switch, and the data is flooded to all hosts.

In case 4, the address is not learned and no buffer id is used. no flow entry is added, only the address is recorded, and the controller has to redirect the data back to the switch and flood the data to all hosts.

Note that the above scenario is what happens when packets are presented to the controller, i.e. there are no flow entries available in the switch flow table.

- *PART 3*
  As mentioned before, the situation for the redirection case is not much different from the packets forwarding case. In this scenario, after receiving the packets from the client, the switch will check the packet headers to determine the source and destination MAC addresses and the input port, and then a new traffic entry is installed in the switch's traffic table to send all packets with the same source and destination MAC addresses out the output port to the controller. After receiving TCP packets, the controller will change the destination IP fields of all packets whose destination IP fields match the Mac address of server1 to the Mac address and IP of server2. And for replying packets from server 2 to the client, the source IP and Mac address of the message are changed to the Mac address of server1 again. This completes a redirection of the traffic smoothly.

VI. Testing and Result

A. Test Host Environment

For the test, the environment is different from the host environment for which we intend to maximize testing of program compatibility and accuracy in different system environments. The test environment is shown as below.

TABLE II
Test Host Environment Introduction

CPU	Memory	Operating System
Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz	64-bit operating system x64 based processor	RAM 16.0 GB

B. Testing Procedures

For the test case of forwarding, it is needed to first open the written SDN network topology file by typing the command

line ”sudo python3 networkTopo.py” in the terminal of the virtual machine. After successfully running it, four xterm windows will open for the controller, client and two server nodes. After that, in the window of controller c1, the command ”ryu-manager ryu_forward.py” is typed to run the written controller with forwarding function, and then enter the command line ”sudo python3 server.py” in the Server1 and Server2 windows to start the server. Finally, type ”sudo python3 client.py” to start the client in the Client window. After starting the client, the packets-forwarding program is running successfully as the figure below shows.

And the test procedure for the redirection case is basically the same as the test procedure for the forwarding case, the only difference is that the command line entered in the controller c1 window is ”sudo ryu-manager ryu_redirect.py” instead of ”sudo ryu-manager ryu_forward.py” The result for both test case is shown as following.

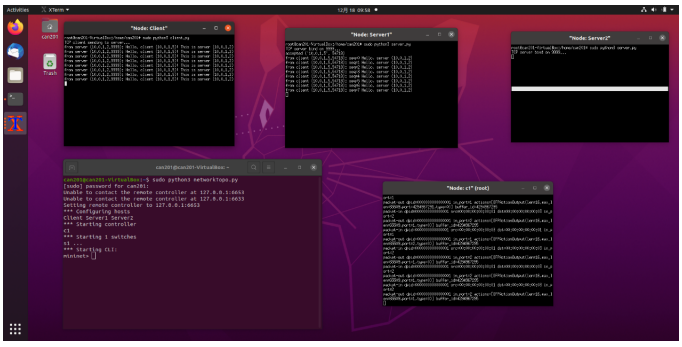

Fig. 5. the result for task 1- 4 and 4.2

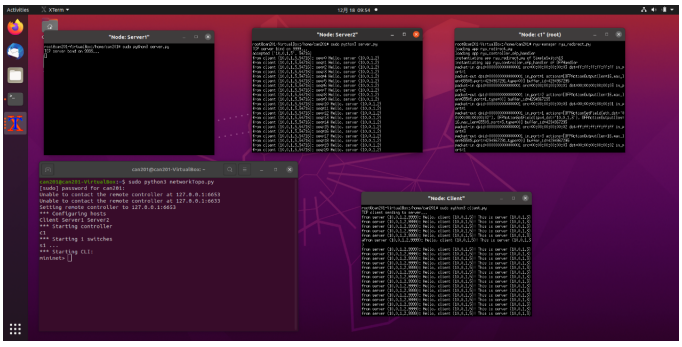

Fig. 6. the result for task 1-4 and 5.2

Figure 5 proves that T1-T4 was successfully completed, and Figure 6 proves that T5 was successfully completed. We tested 7 forwarding and redirection operations and made the following line graph. Compared with forward, redirect takes more time and it also has more significant network latency on average.

It is clear that the in both case that the servers, client and controller is successfully built and are able to communicate with each other. The given line chart is shown the latency of those two cases. It is clear that the average latency of the redirection scheme will be lower than the average latency of

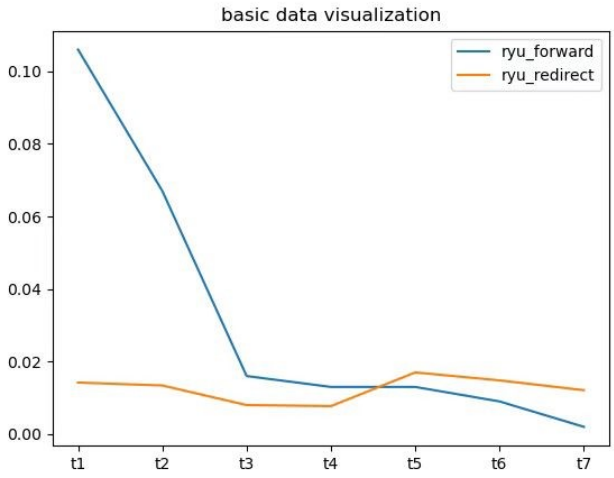

Fig. 7. networking latency

the forwarding scheme. Another noticeable result is that in the test environment, the first result in the packets-forwarding case is the highest one, the performance of the computer itself is considered to be the cause of that high latency, which might be further verified with in-depth research.

VII. Conclusion

We have built a network topology and two controller applications based on the given client and server programs to implement SDN-based traffic control functionality. This system has passed performance tests and performance comparison experiments. It has been experimentally proven to be able to accomplish direct forwarding and redirection of messages.

Since matching takes the rule of unconditional trust on ip, there is a risk of being attacked, leading to resource exhaustion. As an improvement measure, some security rules can be deployed to avoid such attacks, the simplest being improved matching rules to filter anomalous traffic, as demonstrated in the experimental task in Lab 11. In addition, many attacks against controllers have emerged due to their strong administrative privileges, and appropriate security rules should be explored to improve the security of SDN network links.

Acknowledgment

All three of us have made a serious effort to contribute the same percentage to this project. Meaning, Ruijia Tian (2035825) contributed 1/3, Shuran Zhang (2033612) contributed 1/3, and Shuchen Yuan (2036501) contributed 1/3 to this project.

References

[1] C. Ren, S. Bai, Y. Wang and Y. Li, ”Achieving Near-Optimal Traffic Engineering Using a Distributed Algorithm in Hybrid SDN,” in IEEE Access, vol. 8, pp. 29111-29124, 2020, doi: 10.1109/ACCESS.2020.2972103.

[2] D. Tatang, F. Quinkert, J. Frank, C. Röpke and T. Holz, "SDN-Guard: Protecting SDN controllers against SDN rootkits," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 297-302, doi: 10.1109/NFV-SDN.2017.8169856.

[3] F. Bannour, S. Souihi and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges," in IEEE Communications Surveys and Tutorials, vol. 20, no. 1, pp. 333-354, Firstquarter 2018, doi: 10.1109/COMST.2017.2782482.

[4] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, Wu Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," Computer Networks, Volume 71, 2014, Pages 1-30,ISSN 1389-1286, doi: https://doi.org/10.1016/j.comnet.2014.06.002.

[5] J. N. Bakker, I. Welch and W. K. G. Seah, "Network-wide virtual firewall using SDN/OpenFlow," 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2016, pp. 62-68, doi: 10.1109/NFV-SDN.2016.7919477.

[6] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang and Y. Sun, "Minimizing Controller Response Time Through Flow Redirecting in SDNs," in IEEE/ACM Transactions on Networking, vol. 26, no. 1, pp. 562-575, Feb. 2018, doi: 10.1109/TNET.2017.2786268.

[7] S. Shin, L. Xu, S. Hong and G. Gu, "Enhancing Network Security through Software Defined Networking (SDN)," 2016 25th International Conference on Computer Communication and Networks (ICCCN), 2016, pp. 1-9, doi: 10.1109/ICCCN.2016.7568520.