Implementation and test of forwarding and redirection of network system based on ryu + mininet

1st Fanyun Xu
ID: 2033366

2nd Xie Xiang
ID: 2035100

3rd Yanjie Xu
ID: 2034363

***Abstract*—This project plans to design a SDN (Software Defined Network) and achieve the function of forwarding and redirecting. Moreover the network topology simulation and performance test were carried out through the mininet.**

***Index Terms*—ryu + mininet, forward, sdn, switch**

I. INTRODUCTION

This task implements traffic forwarding and traffic redirection through SDN. It uses the idea of hierarchy to separate data plane and control plane. SDN controller holds the global network information and it is responsible for the control of various forwarding rules. Switch is used to forward and process data. The main problems that we need to solve are the initialization of SDN controller and forwarding rules. This proposal can also be used to solve load balancing problems based on SDN architectures. For this coursework, we completed the design and implementation of SDN controller to achieve the functions of forwarding and redirecting data packets which are sent by the host.

II. RELATED WORK

For traffic forwarding and redirection, we wrote a Ryu controller, which is a fundamental SDN controller [1]. SDN controller introduces a new abstraction layer on the basis of the traditional controller, separating the data plane and the control plane [2], so as to solve the limitations of the traditional network. It has more functions than the traditional controller, such as packet modification, flooding and other operations. The function of redirection is to modify the packet. By evaluating capacity, delayed packet loss, and other criteria, the performance of Ryu is better than some of other controller in most of cases [3]. The whole process uses a network simulator to achieve, namely, mininet, which is a suitable and management tool to configure the whole topology [4].

III. DESIGN

A. The network system design

The network system design diagram is shown in Figure 1.

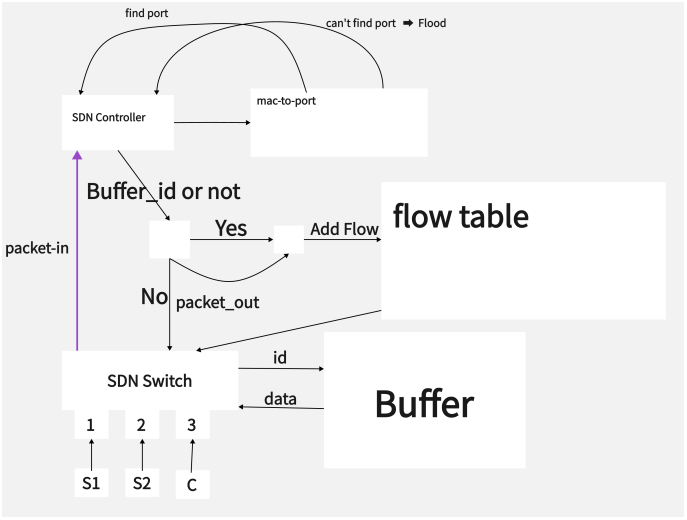

Fig. 1. The network system design

B. The workflow

In this project we built a simple network topology. It consists of two servers, a client, as well as a SDN switch and a SDN controller. In addition, we configure the IPaddresses and MAC addresses of the three hosts in advance. The SDN switch serves as the medium to connect the three hosts and executes different commands for packets coming in from different ports according to the built-in flow table. As the upper layer of the switch, the SDN controller deletes, adds and changes the flow table inside the switch. The advantages of using SDN network are that the network traffic path can be adjusted manually and the network can be programmed. Moreover, we also used Mininet to create a virtual network containing hosts, a switch, a controller and links to simulate the real network situation, so as to improve the authenticity of the later test results. In addition, Mininet supports Openflow application and provides extensible python API, which can be a good help to complete this task. In this project, we need to solve two problems. One is that the switch forwards the packet according to the target address defined in the packet sent by the client. The other is to redirect the data packets sent from the client. The specific design is as follows: First, the client and servers should be initialized, the TCP socket should be created on the server to listen. The client should create a TCP socket to apply for establishing a connection with the server. The next step is to initialize the SDN controller and switch. In this project we used the OpenFlow protocol version 1.3 to define the communication standard between

the controller and the switch. We first need to create a table-miss entry, initialize it, and set the priority to 0 (the lowest priority). It can deal with all match fields and send packet to the controller. In this case, it means that the packet does not match other switch entries. The controller will first parses the packet and retrieves the information in the packet, such as the source Mac address, destination Mac address, and the switch id. The operations of switch towards the packet are defined based on this information. For example, to implement forwarding operations, you need to obtain the output port number based on the switch id and the target Mac address. If the out port number cannot be found in the original mapping table, the system will send the packet to all hosts, that is, flood. If it has a specific output port, a match field should also be defined. The matching field contains multiple optional matching entries, such as the source IP address, the destination IP address, the source Mac address, and the destination Mac address. At last, configure the operation and matching field information into the flow table, and then deliver a new flow table to the switch. For the first problem, it just like the example we mentioned above, the SDN controller will search the output port that points to the server1. Then it will add the items and deliver the flow table to the switch. Whereby all the following traffic sent from Client to Server1 is forwarded to Server1. However the redirection problem may be more complex. All the traffic that sent by the client will be transmitted to the server2 while from the user logic perspective, it always communicates with server1. Therefore, the SDN controller will create a table item which include a function can replace the header information related to server1 with the information for server2. Moreover, as we mentioned above, in the client's point of view, it communicates with server1. So that all the packets sent by the server2 will treated as sent by server1. The SDN controller will create a item which will modify the header information in the traffic sent by the server2.

C. Algorithm

The kernel pseudo codes of the network traffic redirection function as follows.

Algorithm 1 data_packet()

```
if src ← Client's MAC address and dst← Server1's MAC address then
  if Server2's MAC address in self.mac_to_port[dpid]: then
    out_port = self.mac_to_port[dpid][Server2's MAC address]
  else
    out_port = ofproto.OFPP_FLOOD
  end if
  actions = [parser.OFPActionSetField(ipv4_dst ← Server2's IP address), OFPActionSetField(eth_dst ← Server2's MAC address), parser.OFPActionSetField(out_port)]
end if
if src ← Server2's MAC address and dst← Client's MAC address then
  if Client's MAC address in self.mac_to_port[dpid]: then
    out_port = self.mac_to_port[dpid][Client's MAC address]
  else
    out_port = ofproto.OFPP_FLOOD
  end if
  actions = [parser.OFPActionSetField(ipv4_dst ← Server1's IP address), OFPActionSetField(eth_dst ← Server1's MAC address), parser.OFPActionSetField(out_port)]
end if
```

IV. IMPLEMENTATION

A. The host environment and the development tools

The development environment for this project is shown in Figure 2. The IDE used is pycharm Version 3. The python

CPU: AMD Ryzen 9 5900X 12-Core Processor

Memory: G.skill 4000Mhz 16G x 2

Disk: Samsung SSD 980 1TB

Mother board: MSI B550i Gaming EDGE WIFI

operating system: Windows11 22H2

Fig. 2. Environment of project

libraries used in the file are ryu.base, from which to import app_manager to centrally manage ryu's applications, for example loading ryu applications, providing contexts for applications, and routing messages between applications. Moreover, we use the controller's libraries to receive Switch-features messages and send set-config messages, and to negotiate versions and send feature-request messages. And decorate a method as a data handler. We also use ofproto library to import Openflow1.3 protocol data, and regard it as an API between switch and controller. The ryu packet library is also used to help parse and construct various protocol packets, such as packet, ipv4, icmp, tcp, etc. The SDN controller software we use here is Ryu. This is a single-threaded entity to implement the various functions in it, and there is a queue for receiving events in the Ryu APP in the order of FIFO (First in, First out), with one thread for each Ryu for event processing.

B. steps of implementation

In task1, two servers and a client are bound to their IP and Mac addresses respectively. They are all in the same sub-net, and the sub-net mask is /24. In task2, run the application under the Ryu's framework to ensure that client, server1, and server2 are reachable to each other, which means the they can ping each other. Furthermore, create a flow table (for example, when server1 pings server2, the controller sends two entries to the switch. One is to help the server1 to send the ICMP packet to server2. One is when the server2 gets the ICMP

packet, it will reply to server1, and then the controller will create another flow entry for it.) Moreover, to prevent flow table overflow, the idle time is set to be 5 seconds, meaning that if the entry is not called within 5 seconds, the entry will be removed from the flow table. In task3, the given server and client files are applied to two servers and one client and enable them to run successfully. In task4, the packet sent by the client is forwarded to the target address defined by the client. First, it will create a mac-to-port table to record the mapping between Mac addresses and physical ports. If each switch is identified with a unique switch_id, we can obtain a mapping table from (switch_id, Mac) to port_num. After receiving a packet, the switch can find the port based on the destination mac address for forwarding. If the out_port cannot be found in the mapping table, the flood operation is performed. The packets are sent to all ports to find the corresponding ports and recorded in the hash table for the convenience of obtaining the out_port next time. In task5, the task is to redirect the incoming data packets from the client to server1 to server2. It is achieved by modifying the IP address and Mac address of the original target to that of server2 in the packet. The remaining operations are the same as task4.

C. programming skills

In this task, the programming of SDN controller application uses the idea of object orientation to make the code modular, with higher re-usability and easier maintenance of the system. The packet is changed from binary to message object in order to facilitate the call of the controller, and the header is obtained according to the protocol. And the ryu application inherits the ryu.base.app_manager.RyuAPP class to reduce the effort of creating classes.

D. actual implementation of the traffic redirection function

In the traffic redirection function, packets that are intended to be sent from the client to server1 are forwarded directly to server2. First, if the mapping relationship of server2 exists in the mac-to-port mapping table, obtain the corresponding out port of server2. Otherwise, flood is used to obtain it. Then, when the packet is transmitted to the switch, the destination mac address in the packet is modified to complete traffic redirecting. The OFPActionSetField method in the parser package is used to change the header field information in the packet, so as to control traffic redirecting. In addition, when server2 responds to the client, it also uses OFPActionSetField to change the IP address and Mac address of server2 in the packet to server1's, and then sends the packet to the client.

E. the difficulties and solutions

Two difficulties were encountered during the implementation of the project. The first was that in the task5 we did not know how to modify the header field in the packet so that the packet sent to server1 was redirected to server2. The solution is to find the OFPActionSetField method in the official website of RYU the Network operating system and modify the target address in the packet. The second is that the client refuses to receive the response from server2. The solution is to change the source address in the server2 response packet to the IP address and Mac address of server1, so that the client can successfully accept it.

V. TESTING AND RESULT

A. Testing

The operation system of the project is Windows10, the memory is 16G, the CPU is i5-10210U, and the testing environment is a virtual machine equipped with Ubuntu operating system, which uses one core processor and the memory size of 1024MB.

1) Establish the network topology and add a controller, a switch and three hosts to the Mininet network. The three hosts are assigned to IP addresses and Mac addresses respectively and are all connected to the switch. The following 3 shows the network topology.

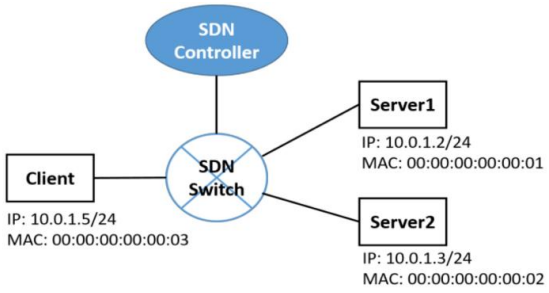

Fig. 3. Network topology

2) Check whether each node can reach each other. Use the pingall function to see if all the packets are received and whether there are any packets lost.

```
*** Starting controller
c1
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
server1 -> server2 client
server2 -> server1 client
client -> server1 server2
*** Results: 0% dropped (6/6 received)
```

Fig. 4. Ping

3) Apply the given server file and client file to the corresponding host, so that two servers and one client can run successfully, and the table_miss entry can be correctly displayed in the controller.
4) SDN controller application needs to be programmed so that it can normally deliver the flow table. First, create a table-miss entry to match all packets. The priority is set to be 0. Then the data structure in the packet is parsed to obtain the source port, path and switch parameters. Create a mac-to-port mapping table and bind (switch id, Mac address) to port numbers. Before forwarding or other actions, check whether the destination Mac address is mapped in the table. If yes, the out_port

is obtained. If the destination MAC address cannot be found, packets are sent to all ports. At last, the flow table is then sent to avoid triggering the packet_in event the next time. Different matching conditions are defined for different protocols. For example, if the protocol is TCP, you can cancel the matching between the IPv4 source address and the tcp source address to prevent the syn segment using the random source cause millions of entries generating in the table flow which is overflow. The matching entries are then created and a flow_mod message is sent to the switch to configure the flow table. There are some flow_mod messages with buffer_id. If there is a buffer_id, a flow is normally sent and the data packet with the buffer_id in the controller is sent to the switch to match the flow. There is no additional packet_out messages from the controller are required. On the other hand, send the packet_out packet and flow table if there is no buffer_id.

B. Results

To test the performance of forwarding and redirecting, we recorded the 3-way handshake time for these two different operations. They were all counted for 10 times respectively, and their average time was calculated to compare the performance. The corresponding bar figure is as follows: According to the

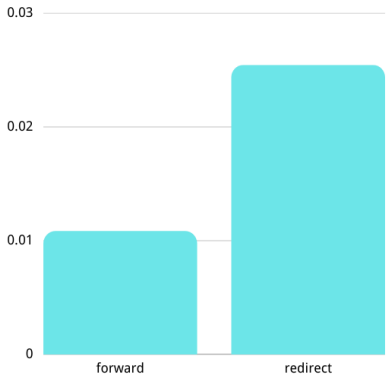

Fig. 5. Results

analysis in the figure, the average forwarding time is 0.01083s, and the average redirecting time is 0.02541s. The redirecting time is about twice as long as the forward time.

VI. CONCLUSION

In this project, we completed the design, implementation of SDN controller and tested the whole network topology according to the requirements. We designed the SDN controller based on Ryu and implemented the functions of flow table delivery and packet header modification to achieve traffic forwarding and traffic redirection. In the test phase, we tested the TCP connection establishment time of both forwarding and redirecting operations to evaluate the efficiency. As a result, redirection takes about twice as long as forwarding. In the future improvement work, the forwarding path can be defined in combination with the shortest path algorithm in the traditional routing method. In addition, we can also change the matching information of flow table entries to resist DOS attacks and improve network security.

REFERENCES

[1] Saleh Asadollahi, Bhargavi Goswami, and Mohammed Sameer. Ryu controller's scalability experiment on software defined networks. In *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, pages 1–5, 2018.

[2] Dennis Tatang, Florian Quinkert, Joel Frank, Christian Röpke, and Thorsten Holz. Sdn-guard: Protecting sdn controllers against sdn rootkits. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 297–302, 2017.

[3] Ravindra Kumar Chouhan, Mithilesh Atulkar, and Naresh Kumar Nagwani. Performance comparison of ryu and floodlight controllers in different sdn topologies. In *2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing Communication Engineering (ICATIECE)*, pages 188–191, 2019.

[4] Müge Erel, Emre Teoman, Yusuf Özçevik, Gökhan Seçinti, and Berk Canberk. Scalability analysis and flow admission control in mininet-based sdn environment. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 18–19, 2015.