The Implementation of Forwarding and Redirection Based on the Ryu Framework

1st Ruoxuan Cao
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou, China
ruoxuan.cao21@student.xjtlu.edu.cn

2nd Zhihui Wang
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou, China
zhihui.wang21@student.xjtlu.edu.cn

3rd Bowen Fang
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou, China
bowen.fang21@student.xjtlu.edu.cn

4th Zeyu Chen
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou, China
zeyu.chen21@student.xjtlu.edu.cn

5th Yiran Peng
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou, China
yiran.peng2102@student.xjtlu.edu.cn

***Abstract*—The control flow plays a critical role in the network project. As part of our mission, we successfully implemented a simple SDN network topology using Mininet. This article comprehensively presents our investigation process, which includes exploring the latest advancements in network frontier and traffic redirection, designing and implementing our SDN solution, conducting thorough tests, and analyzing the obtained results.**

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

I. Introduction

IP networks have been fully distributed and battle-hardened for the past 30 years, addressing a wide range of customer needs, and SDN today is about enabling user needs better and faster in the future. Figure out how the SDN controller can for- ward or redirect traffic without the awareness of the client. The SDN architecture brings tremendous gains in open network capabilities and improved network resource utilization, and can be applied in cloud data centers, WANs, transport networks and mobile core networks. The files Ryu redirect.py and Ryu forward.py have been successfully written to implement all the task requirements.

II. RELATED WORK

Redirection is the process by which when a user visits a web page, the server directs the user's request to another page. In this section, we review the relevant research on network traffic redirection and the current cutting-edge applications of related technologies. In a paper published in 2015, Chand et al. proposed a traffic redirection-based Congestion Control Transport Protocol (TRCCTP) for wireless sensor networks, which provides an effective mechanism for congestion control in wireless sensor networks [1]. Shih et al. published a paper in 2017 that built a true mobile edge computing platform on top of a container-based environment. It is mainly through the design of mobile terminal and traffic redirection workflow, and the realization of application awareness function in eNodeB [2].

III. DESIGN

A. The network system design

In this network system, the client is aware of the services running on server1 due to communication between the client and server1. However, there is no communication between the client and server2, which means the client is unaware of the services running on server2. An SDN controller is utilized to redirect traffic and implement traffic control measures to solve this issue. The network system design diagram is shown in Figure 1.

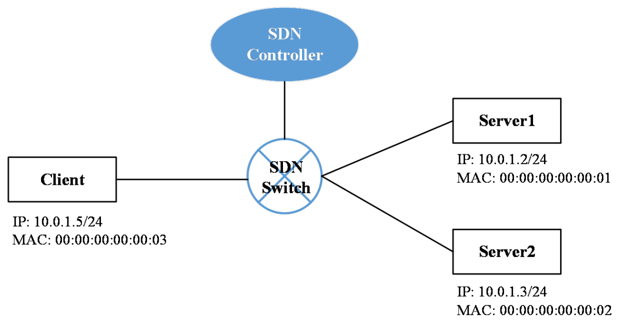

Fig. 1. A simple SDN network topology

B. The workflow

In this project, a simple network topology which contains two servers, one client, and an SDN switch and SDN controller was built. The IP addresses and MAC addresses of the three hosts were preconfigured. the SDN switch acts as an intermediary between the three hosts, executing different commands based on a built-in flow table to process packets from different ports. The SDN controller sits on the upper level of the switch and is responsible for deleting, adding, and modifying the flow table within the switch. The advantage of using SDN network is that we can manually adjust the network traffic path and perform flexible network programming

To simulate the network environment more realistically, a virtual network with hosts, switches, controllers, and links using Mininet was created, which supports OpenFlow applications and provides a flexible Python API that was very helpful in accomplishing the task. There are two main problems in this task: one is to enable the switch to forward packets sent by clients based on the destination address in the packet, and the other is to realize the redirection of packets sent by clients. The specific design scheme is as follows:

First, the client side and the server side need to be initialized. On the server side, creating TCP sockets for listening to connection requests; on the client side, creating TCP sockets to request a connection to the server. Next, the SDN controllers and switches were initialized, using the OpenFlow protocol version 1.3 to define the communication standard between the controllers and switches. In the process, we created a table entry that handles packets that do not match other table entries, sending them to the controller for further processing.

To solve the first problem, the SDN controller looks up the output port pointing to server1, then adds the appropriate entry and passes the updated flow table to the switch. In this way, all subsequent traffic from the client to server1 will be correctly forwarded to server1. however, the redirection problem is relatively more complex. All traffic sent by the client is transmitted to server2, but from the user's logical point of view, the communication is always going on with server1. Therefore, the SDN controller needs to create a table entry that includes a function that can replace the header information associated with server1 with server2 information. In addition, from the client's perspective, the communication has been with server1. This means that all packets sent by server2 need to be treated as if they were sent by server1.The SDN controller also needs to create a table entry for modifying the header information in the traffic sent by server2 to enable redirection of packets.

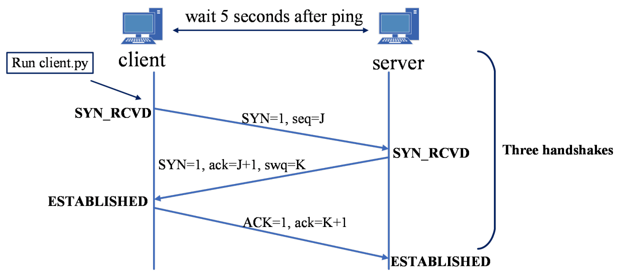

Fig. 2. Flowchart of a given program running to the SDN

C. Algorithm

The kernel pseudo codes of the network traffic redirection function as below. The pseudo code is used to implement the forwarding of data packets sent from the client to server1 to server2. When the switch receives a TCP packet, the program will determine the direction of the packet based on its source MAC address and destination MAC address, and then forward the packet to the corresponding port according to the preset forwarding rules. Specifically, the if statement in the above code is used to determine whether the source MAC address and destination MAC address of the packet comply with the preset rules. If they do, then the packet is forwarded to the corresponding port according to the rules, and the source MAC address and destination MAC address of the packet are modified. Among them, server1 and server2 are the information of the two servers, client is the information of the client, and mac_to_port is a dictionary used to record the port number corresponding to each MAC address. This achieves traffic redirection.

D. Replenishment

a) Buffer id check: Regarding the code for adding and processing flow table entries in the OpenFlow protocol, the purpose of the buffer id check here is to verify whether the switch has successfully cached the packet. If caching occurs, the controller can directly add a stream table entry to the switch without resending the packet. This practice significantly enhances data processing efficiency by eliminating unnecessary back-and-forth transmission of packets.

Additionally, this mechanism effectively reduces network congestion as it enables faster packet processing by the controller, eliminating delays caused by round-trip travel between the switch and controller. Simultaneously, it improves network stability by enabling prompt response from the controller to network changes and facilitating updates to flow table entries for seamless adaptation to new network configurations.

b) Two addflow methods: The first addflow method is utilized during the switch's initial connection to the SDN network, enabling it to add default flow table entries when reporting its own information to the controller. As there are no existing stream entries at this stage, incoming packets are considered as miss flow-entry. With idle_timeout parameter is not specified and a priority of 0, the permanent default stream entry ensures that these packets can be forwarded to the controller for processing in case of uncertainty regarding their treatment.

The second addflow1 method is employed by the controller to convey instructions on how to handle packets based on certain field characteristics. It complements the first addflow method as they work together seamlessly. Through this approach, specific flow table entries can be sent from the controller to instruct the switch on packet processing strategies tailored for particular fields. By setting idle_timeout = 5, if there are no matched packets for this flow within 5 seconds, the switch will remove the entry. These flow entries allow matching and manipulation according to distinct packet attributes, thereby facilitating a more flexible approach towards packet processing.

Algorithm 1: Traffic redirection function

```
1  if src == client's MAC address and dst == server1's MAC address then
2      if server2's MAC address in self.mac_to_port[dpid] then
3          out_port = self.mac_to_port[dpid][server2['mac']]
4      end if
5      else
6          out_port = ofproto.OFPP_FLOOD
7      end if
8      match = parser.OFPMatch($eth\_type \leftarrow ether\_types.ETH\_TYPE\_IP$, $in\_port \leftarrow in\_port$, $ipv4\_dst \leftarrow dstip$, $ip\_proto \leftarrow protocol$, $tcp\_dst \leftarrow t.dst\_port$);
9      actions = [parser.OFPActionSetField($eth\_src \leftarrow server2'sMACadress$),
10     parser.OFPActionSetField($ipv4\_src \leftarrow server2'sIPadress$),
11     parser.OFPActionOutput($port \leftarrow out\_port$)];
12 end if
13 else if src== server2's MAC adress and dst == client's MAC adress then
14     if client's MAC adress in self.mac_to_port[dpid] then
15         out_port = self.mac_to_port[dpid][client['mac']]
16     end if
17     else
18         out_port = ofproto.OFPP_FLOOD
19     end if
20     match = parser.OFPMatch($eth\_type \leftarrow ether\_types.ETH\_TYPE\_IP$, $in\_port \leftarrow in\_port$, $ipv4\_dst \leftarrow dstip$, $ip\_proto \leftarrow protocol$, $tcp\_dst \leftarrow t.dst\_port$);
21     actions = [parser.OFPActionSetField($eth\_src \leftarrow server1'sMACadress$),
22     parser.OFPActionSetField($ipv4\_src \leftarrow server1'sIPadress$),
23     parser.OFPActionOutput($port \leftarrow out\_port$)];
24 end if
25 else
26     match = parser.OFPMatch($eth\_type \leftarrow ether\_types.ETH\_TYPE\_IP$, $in\_port \leftarrow in\_port$, $ipv4\_dst \leftarrow dstip$, $ip\_proto \leftarrow protocol$, $tcp\_dst \leftarrow t.dst\_port$);
27 end if
```

IV. IMPLEMENTATION

A. The host environment

- CPU: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz;
- Memory: 32.0 GB;
- Operating system: Windows 10.

B. The development tools

The IDE used is PyCharm and the programming language is Python Version 3.6. The python libraries used in this file:"app_manager" library serves the fundamental purpose of managing essential functionalities within Ryu applications. The "ofp_event" module defines various OpenFlow event types, facilitating the handling of events associated with the OpenFlow protocol. The "CONFIG_DISPATCHER" and "MAIN_DISPATCHER" libraries specify distinct phases of the state machine for processing OpenFlow messages. The "set_ev_cls" decorator is employed to register methods as specific event handlers for OpenFlow events. The "ofproto_v1_3" library defines constants pertinent to OpenFlow protocol version 1.3, enabling interactions with the OpenFlow protocol at this specific version. The classes "packet", "ethernet", "ipv", "tcp", "icmp", and "arp" collectively contribute to the parsing and construction of network data packets. The "in_proto" library defines constants related to Internet protocols. Additionally, the "ether_types" library is employed to define constants representing different protocol types within Ethernet frames. In summary, these libraries collectively provide the necessary tools and functionalities for developing Ryu applications that engage with the OpenFlow protocol and manipulate network packets.

C. Steps of implementation

In task1, the first step is to complete networkTopo.py to create a network topology. And bind server1, server2, and client to the IP address and MAC address specified in the question requirements. In task2, run the SDN application under Ryu framework to ensure that the client and server can ping each other. And create a flow label in the created switch s1. To prevent overflow of the flow table, the idle time is set to 5 seconds. If the entry is not called within 5 seconds, it will be deleted from the flow table. In task3, run server.py on server1 and server2, run client.py on client. Confirm that all three hosts can run successfully. In task4, the SDN control program was programmed to achieve forwarding function in the SDN network. When the switch receives a packet, it stores the source MAC address and input port of the packet in the mac_to_port dictionary, for use in future forwarding. Then, the exchange opportunity checks whether the target MAC address is already in the mac_to_port dictionary, if true, forward the packet to the corresponding output port; otherwise, flood the packet to all ports. Before forwarding packets, the switch also checks the protocol type of the packet. If it is an IP protocol, it further checks the protocol type (TCP or ICMP) and the source and destination IP addresses of the packet. If it is the ARP protocol, the source MAC address and destination MAC address of the data packet will be checked. Finally, the switch will forward the data packet to the corresponding output port and add the forwarding rule to the switch's flow table for future forwarding. In task5, redirect the incoming packets from the client to server1 to server2. This is achieved by modifying the IP address and Mac address of the original target in the data packet to the IP address of server2. The remaining operations are the same as task4.

D. *Programming skills*

For this assignment, the programming approach for the SDN controller application leverages object-oriented principles to enhance code modularity, enabling increased reusability and simplified system maintenance. The transformation of the packet format from binary to a message object is implemented to streamline controller calls, and the header is acquired based on the specified protocol. Additionally, to minimize the workload involved in class creation, the Ryu application inherits from the 'ryu.base.app manager.RyuAPP' class.

E. *Actual implementation*

a) The traffic redirection function: Within the traffic redirection mechanism, first input a series of parameters and check whether the conditions to redirect traffic to server1 or server2 are met, depending on the source and destination MAC addresses. If the conditions are met, check if the MAC address of the target server is known. If known, determine the output port; otherwise, flood the traffic. Then, create an OpenFlow match condition based on the input parameters and define a set of OpenFlow actions to modify Ethernet and IPv4 addresses. Finally, call the 'add_flow1' method to add the flow entry to the switch, redirecting traffic according to the specified conditions.

b) The process of self-learning: First, obtain the message and datapath information. Retrieve the message object from the event object and extract information such as the datapath, OpenFlow protocol version, parser, and input port from the message object. Next, use 'packet' to parse the received packet, extracting Ethernet header information. Then, check the Ethernet frame type and retrieve the source MAC address and destination MAC address from the parsed Ethernet header information. Generate a datapath identifier, log information about the received packet, including datapath ID, source MAC address, destination MAC address, and input port. Store the learned mapping of source MAC addresses to input ports in the 'self.mac_to_port' dictionary. If the destination MAC address has been learned, retrieve the corresponding output port; otherwise, set the output port to OFPP_FLOOD, indicating broadcast of the packet to all ports.

F. *The difficulties and solutions*

Initially, getting started might seem daunting, but the Ryu controller's source code is actually quite straightforward. The ping function is achieved through matching entries for ICMP and ARP protocols, which are essential for checking host connectivity. Another challenge is setting the idle_timeout. If it's not configured correctly, you may encounter issues with TCP redirection after pinging.

V. TESTING AND RESULTS

A. *Testing environment*

The operation system of the computer is Windows10, with 16G memory and AMD Ryzen 75800H with Radeon Graphics. The test environment is a virtual machines equipped with Ubuntu (64-bit) operating system which uses 8 core processors and the memory size is 4096MB

B. *Testing steps*

a) Build a SDN network topology: Build a SDN network topology: A controller, a switch and 3 hosts named 'client','server1','server2' have been added to the Mininet network. Three hosts are connected to the switch and assigned IP and MAC addresses respectively. Run networkTopo.py to build SDN network topology. Then use "dump" to check if the 3 hosts' IP and MAC addresses build correctly. The SDN network topology is shown in Fig.3.

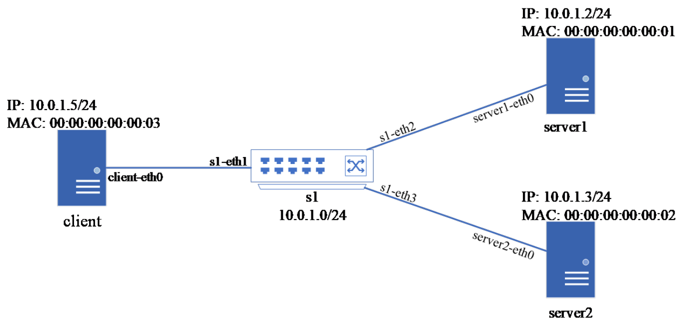

Fig. 3. SDN network topology

b) Run ryu_forward.py and ryu_redirect.py: Run them on the controller c1 respectively. Use 'pingall' to check if client, server1, and server2 can ping each other pairwise. Fig.4. is an example shows client ping server1. For both forwarding case and redirecting case, this step of testing was successful.

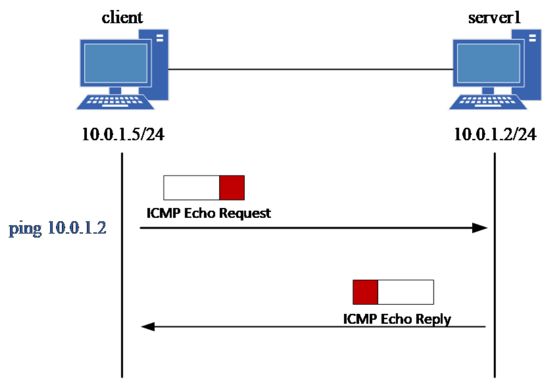

Fig. 4. client ping server1

c) (1) Run server.py and client.py: Run server.py on server1 and server2 and run client.py on client after the previous ICMP ping incurred flow entry's idle timeout. The flow tables on Switch s1 are as TABLE I. and TABLE II.

TABLE I
FORWARD CASE

cookie=0x0	cookie=0x0
duration=9.321s	duration=9.314s
table=0	table=0
n_packets=21	n_packets=11
n_bytes=1686	n_bytes=1236
idle_timeout=5	idle_timeout=5
priority=1	priority=1
tcp	tcp
in_port="s1-eth1"	in_port="s1-eth2"
nw_dst=10.0.1.2	nw_dst=10.0.1.5
tp_dst=9999	tp_dst=34088
actions=output: "s1-eth2"	actions=output: "s1-eth1"

TABLE II
REDIRECT CASE

cookie=0x0	cookie=0x0
duration=9.062s	duration=9.047s
table=0	table=0
n_packets=19	n_packets=10
n_bytes=1524	n_bytes=1119
idle_timeout=5	idle_timeout=5
priority=1	priority=1
tcp	tcp
in_port="s1-eth1"	in_port="s1-eth3"
nw_dst=10.0.1.2	nw_dst=10.0.1.5
tp_dst=9999	tp_dst=42488
actions=set_field: 00:00:00:00:00:02->eth_dst"	actions=set_field: 00:00:00:00:00:01->eth_src
set_field:10.0.1.3->ip_dst	set_field:10.0.1.2->ip_src
output:"s1-eth3"	output:"s1-eth1"

(2): For the forwarding case, server1 receives the traffic sent from client will shows on the terminal. After running successfully, server1 will print "from client (10.0.1.5.34088): seq=0 Hello, server (10.0.1.2)", client will print "from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)"

For the redirecting case, server2 receives the traffic sent from client will shows on the terminal. After running successfully, server2 will print "from client (10.0.1.5.42488): seq=0 Hello, server (10.0.1.2)", client will print "from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)".

C. Test results

A comparison was made between the forward and redirect situations, and the Three-way handshake time was tested ten times. By calculating the average, the average forwarding time was 0.002443314s and the average redirecting time was 0.0031352s. The line chart is as follows. Redirection takes more time because new IP address and MAC address need to be reset, it requires time consumption during redirection.

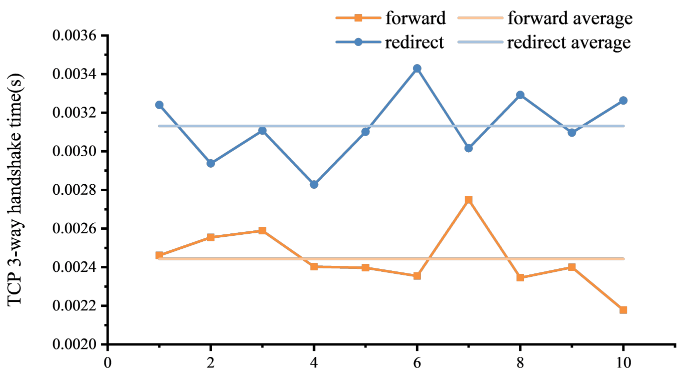

Fig. 5. Test result

VI. CONCLUSION

In this project, we created a simple SDN network topology, completed the design and implementation of the SDN controller, and tested the experimental results according to testing requirements. Based on Ryu SDN controller, the function of creating flow entries, installing flow entries into SDN switches, and sending data for traffic forwarding and traffic redirection has been implemented. During the testing phase, we tested the TCP connection establishment time for forwarding and redirection operations and found that redirection took more time. In future improvement work, we can explore the use of features such as traffic control and multipath forwarding. In addition, attention should be paid to network security and code security should be improved.

ACKNOWLEDGMENT

Ruoxuan Cao (2141986); Zhihui Wang (2144366) Bowen Fang (2144218); Zeyu Chen (2143415) Yiran Peng (2142136)

REFERENCES

[1] T. Chand, B. Sharma and M. Kour, "TRCCTP: A traffic redirection based congestion control transport protocol for wireless sensor networks," 2015 IEEE SENSORS, Busan, Korea (South), 2015, pp. 1-4, doi: 10.1109/ICSENS.2015.7370452.
[2] S. -C. Huang, Y. -C. Luo, B. -L. Chen, Y. -C. Chung and J. Chou, "Application-Aware Traffic Redirection: A Mobile Edge Computing Implementation Toward Future 5G Networks," 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), Kanazawa, Japan, 2017, pp. 17-23, doi: 10.1109/SC2.2017.11.