SDN Controller Applications Based on Ryu Framework

Yizhen Cao, Yuzhen Chen and Bowen Li
School of Advanced Technology
Xi'an Jiaotong-Liverpool University
Suzhou, P.R.China
{Yizhen.Cao20,Yuzhen.Chen20,Bowen.Li20}@student.xjtlu.edu.cn

***Abstract*—Software Defined Networking (SDN) is an innovative network architecture, which can be considered as an effective approach to implement load balancing and traffic control as compared to traditional networking. In this project, we are motivated to develop two controller applications realizing traffic forwarding and redirection. Therefore, we initially prototyped a simple SDN topology in a virtual testbed, after which designed system architecture and core algorithms of two functionalities aimed to be implemented. Then, we wrote practical Python codes in the virtual machine to implement the design of controller applications. In addition, we tested the feasibility as well as workability of programs developed and the performance was also tested. Relevant results suggest that the applications can work normally and efficiently.**

***Index Terms*—SDN, redirection, Ryu, forward**

I. Introduction

Different from traditional networking architecture, SDN separates control plane from data plane, providing flexible programmability and enhancing the central control capability as well as adaptability to cloud computing. Besides, some urgent and complicated networking problems has been resolved based on SDN approach [1].

In this project, we are required to construct a simple SDN network topology using a virtual network testbed named Mininet. Then, program two controller applications based on Ryu framework to implement the function of network traffic forwarding and redirection, respectively. Two major challenges need to be overcome to accomplish the project. One is that developers are unfamiliar with the principle of SDN network. The other one is that team members never used Ryu framework before to complete a self-defined networking traffic manipulation application. Under the circumstance of modern websites that process huge number of requests in an extremely short period of time, overburden may occur leading to the crash of single server. Load balance can efficiently resolve this practical problem by distributing networking traffics to a set of servers. The redirection application developed redistributes traffics to another server that is not originally appointed by the client, which can be considered as a simplified version to implement load balancing.

The contributions of our work are listed below:

- First, we constructed a SDN network topology with one client and two servers (server1 and server2) connected by a SDN switch controlled by a SDN controller in Mininet. The reachability of every node in the topology is checked and ensured.
- Second, we programed two controller applications using Ryu framework to realize traffic forwarding and redirection.
- Third, Wireshark is used to test the respective performance of two applications developed and compare the testing results.

The rest of the contents are organized as follows. Section II presents some related work. Section III concentrates on design of the project and vital pseudo codes as well as corresponding explanations. Section IV elaborates practical implementation environment and shows part of core source code. Section V demonstrates the testing steps plus a figure showing performance comparison. Section VI summarizes the report and envisions the future work.

II. Related work

In this section, we review the related work about the network traffic redirection from classical redirection to SDN-based redirection.

A. Classical network traffic redirection

The network traffic redirection is one of the ubiquitous network forwarding requirements in cloud computation environment. From the perspective of virtual machine migration, several works have been proposed. For example, F. Travostino et al. [2] presented to utilize the traditional tunnel technology to realize network redirection after VM migration. K. Onoue et al. [3] also invented a kind of data transponder which ensures the original IP communication between virtual machines during redirection in a single network. B. Borisaniya et al. [4] further installed transponders in all networks to achieve cross-zone redirection. On the other hand, other works are more specific on the redirection of Content Delivery Networks. For instance, T. Boros al. [5] proposed using proxy servers to implement network redirection through rewriting the domain name of original servers according to Domain Name System (DNS) protocol. M. Ghaznavi et al. [6] also used TCP_REPAIR flag to dump the state of a current TCP session and loaded it on the desired host to implement network redirection.

Our work can be distinguished from the above mentioned papers, since we mainly focused on handling TCP and ICMP packets, and SDN is applied to network redirection.

B. SDN-based network traffic redirection

SDN provides a new architecture for the network which improves the programmability of the network communication model. Y.pu et al. [7] proposed a topology migration model based on Open vSwitch which realized the network topology migration by using the Open vSwitch control agent based on Linux. Their research provides a significant hint for solving the network redirection in the cloud environment. C.Liu et al. [8] also mentioned a new real-time traffic redirection method for edge network based on load balancing which optimizes the network traffic based on the statistics collected by controller.

The aforementioned related work provided the theoretical basis for the implementation of network redirection of this project, and helped us to clarify the methodology of building SDN network.

III. Design

In this section, we design basic network architecture and algorithms to implement network traffic redirection.

SDN architecture is a creative network architecture which has become a new approach of network virtualization implementation [9]. The design concepts of this project is highly dependent on the working principle of SDN network architecture which is the central challenge of algorithm design.

A. SDN network system architecture

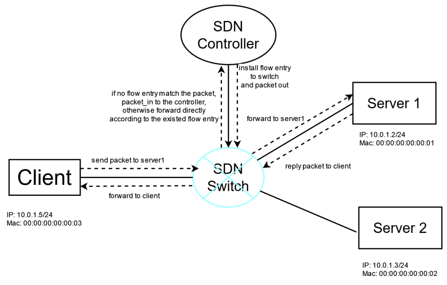


Fig. 1. forwarding function of SDN network architecture

According to Fig. 1. and Fig. 2., we can abstract the components of the SDN network into four classes: client , server, SDN switch and SDN controller. The two servers are connected with client through a single switch and SDN switch can communicate with SDN controller through a specific data_path. The controller could determine the actions of each packet going through SDN_switch by flow entries installed in the switch which will control the whole network traffic. In addition, each host connects with switch through a specific interface and a port number is allocated to each interface.

The actions of SDN controller can be distilled into two functions: adding flow entry and handling packet_in packet.

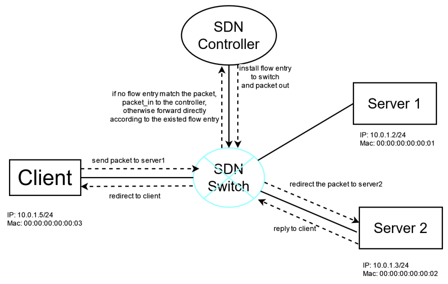


Fig. 2. redirection function of SDN network architecture

Once a packet is sent to a server, the switch will first match the existed flow entries in the flow table. If each field matches the information of the packet, the packet will forward or redirect according to the rules. Otherwise, packet_in packet will be sent to the SDN controller. After the SDN controller receives the packet, it will create flow entry. Then, packet_out packet will be sent to SDN_switch and forwarded to the right output port according to the network rules provided by the network administrator. Finally the flow entry will be installed in switch and actions provided by the controller will be executed (forwarding or redirection).

B. The workflow of solution

Although the functionality mainly focuses on adding flow entry and handling packet_in packet. However, since the packet_in handling is similar to creating flow entry, we will just concentrate on creating flow entry and installing flow entry in this subsection. Firstly, we will define the specific implementation flow of each functionality and then abstract out the specific functions each functionality needed. In addition, the communication between client and server are based on TCP connection.

Before the establishment of TCP connection between client and server, several initialization work should be completed in order to ensure the normal running of the SDN network. First of all, the controller should allocate a data_path_id through which controller can send message to switch. Secondly, a table-miss flow entry should be installed to the switch in order to match packets that should be sent to the controller. Thirdly, an empty mac_to_port table should be defined to record the mapping relationship between the port number of SDN switch and MAC address of each host.

1) Creating the flow entry: In advance of TCP connection between client and server, ARP request and reply should be sent in order to generate the mapping relationship between the port number and MAC address in mac_to_port table. These information will be used in network redirection task later.

While the first TCP packet sent from client to the switch, its original destination is server1. Since there are no flow entries matching the packet, it will be sent to the controller according to the table-miss flow entry. Once the packet enters into the SDN controller, the information in network protocol of this packet will be extracted including source_ip, destination_ip and protocols above the network layer. After the controller determines the packet as a TCP packet and the in_port is the port connecting to client, it will define the port connecting to server2 as out_port and set an action including changing packet fields of out_port, source_ip and source_mac which helps instruct such kind of packet to redirect to the correct direction. The match field will also be set to help switch recognize such kind of packets. In addition, the priority should also be set since all the flow entry should have higher priority than table-miss flow entry. Finally, the controller will create a flow entry which includes match, action, priority and data_path.

The process of creating flow entry of TCP packet from server2 to client is quite similar to the precious one. However, the source_ip and source_mac should be server1 since TCP is connection-oriented protocol. Actually, the SDN switch deceives the client that it is connected with server1.

2) Installing the flow entry: After the creation of flow entry, the controller will install it to switch through data path which is a bridge between switch and controller. However, an idle time should be added as a field of flow entry which helps remove useless flow entry. Once the flow entry has been installed to switch, packets with the same properties will be redirected directly without triggering packet_in event.

C. Algorithm Design

Here are the kernel pseudo code of networking traffic redirection function. Algorithm1 is the pseudo code for redirecting client to server2 and algorithm2 is for redirecting server2 back to client.

Algorithm 1: redirection from client to server2

```
if in_port is connected to client then
    server2_mac is '00:00:00:00:00:02'
     server2_ip is '10.0.1.3
     if server2 exists in mac_to_port table then
         out_port is the port connect to server2
    else
         out_port is flood
    end
    start:
     match :
     configure match rules of flow entry
     match rules are source ip and destination ip etc.
     actions :
     change header field: set destination MAC to
     server2's MAC
     change header field: set destination ip to server2's
     ip
     output the packet to out_port
     end
end
```

Algorithm 2: redirection from server2 to client

```
if in_port is connected to server2 then
    server1_mac is '00:00:00:00:00:01'
     server1_ip is '10.0.1.2
     if client exists in mac_to_port table then
         out_port is the port connect to client
    else
         out_port is flood
    end
    start:
     match :
     configure match rules of flow entry
     match rules are source ip and destination ip etc.
     actions :
     change header field: set source MAC to server1's
     MAC
     change header field: set source ip to server1's ip
     output the packet to out_port
     end
end
```

IV. Implementation

In this section, we use Python to program redirection application in the virtual machine.

A. Implementation Environment

The physical host running virtual machine (VM) is HUAWEI MateBook D 14 having CPU 6-core AMD Ryzen 5 45000U with Radeon Graphics with speed 2.3GHz and 16 GB main memory with speed 2667 MHz. The operating system installed is Windows 10 Home. VM is created by VirtualBox, where the version is VirtualBox 6.1.40. VM has the same memory speed and CPU power as the physical host. However, the main memory of VM is 4 GB. Ubuntu-20.04 64-bit is the operating system of VM and Python 3 is installed to develop the application. The development tool is gedit 3.36.2, an Ubuntu built-in text editor, no specific Python IDE is used. All actual coding work is extended by the framework of a SDN controller software Ryu that has been developed using Python, where the version installed is Ryu 4.34. The Python module used is only ryu in redirection application.

B. Actual Implementation

Main steps to implement redirection function is represented by a program flow chart in figure 2. Actions will be set to only forward incoming packet to the correct out_port when packet_in event occurs, after which determine whether the out_port existed in the mac_to_port table. If existed, check ethernet type field in the frame. If it is IP, further check the upper layer protocol. If TCP is detected, identify which port the packet comes in. If the incoming port connects with client, configure rules of match and action. If the incoming port connects with server2, set distinct rules of match and action. OOP is the programming skill exploited to develop this application since the Ryu framework uses the idea of OOP.

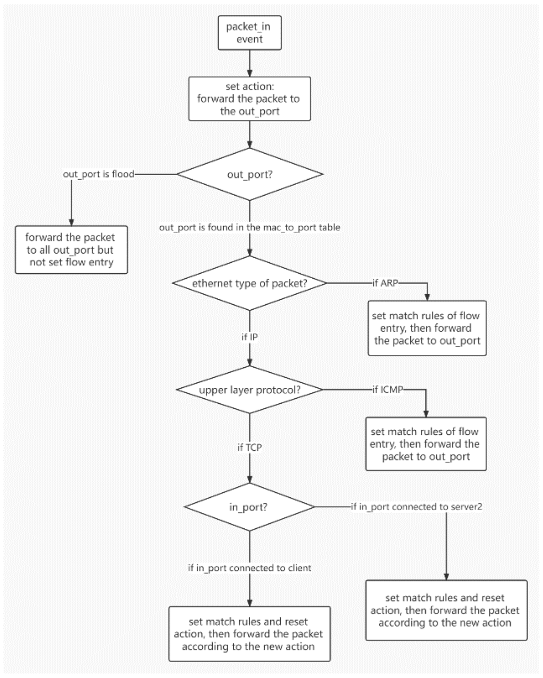

packet_in event
set action: forward the packet to the out_port
out_port?
out_port is flood
forward the packet to all out_port but not set flow entry
out_port is found in the mac_to_port table
ethernet type of packet?
if ARP
set match rules of flow entry, then forward the packet to out_port
if IP
upper layer protocol?
if ICMP
set match rules of flow entry, then forward the packet to out_port
if TCP
in_port?
if in_port connected to server2
set match rules and reset action, then forward the packet according to the new action
if in_port connected to client
set match rules and reset action, then forward the packet according to the new action

Fig. 3. steps of implementation

Only part of core Python code is shown below. The following code aims to implement match plus action when detecting TCP packet sent from client to server1. match = parser.OFPMatch() is used to set match rules in the flow entry. The most critical rules contain source and destination IP address. Header field of packet only contain IP and MAC address of server1 since client designates to send traffic to server1. Hence, parser.OFPActionSetField() is for modifying IP and MAC address in the packet header to the address of server2. Forwarding packet to the port that connects with server2 is implemented using parser.OFPActionOutput().

```
if server2_mac in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][server2_mac]
else:
    out_port = ofproto.OFPP_FLOOD

match = parser.OFPMatch
(eth_type=ether_types.ETH_TYPE_IP,
in_port=in_port,ipv4_src=srcip, ipv4_dst=dstip,
ip_proto=protocol,tcp_src=t.src_port,
tcp_dst=t.dst_port)

actions =
[parser.OFPActionSetField(eth_dst=server2_mac),
parser.OFPActionSetField(ipv4_dst=server2_ip),
parser.OFPActionOutput(port=out_port)]
```

The code below implements match plus action when detecting TCP packet replied by server2 to client. The flow entry set by match = parser.OFPMatch() is the same as the previous one. Client originally desires to send traffic to server1 and TCP is connection-oriented. Thus, parser.OFPActionSetField() is for replacing source IP and MAC address in the packet header with address of server1 so that client can receive the traffics. Then, parser.OFPActionOutput() messages switch which port to send packets out.

```
if server2_mac in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][server2_mac]
else:
    out_port = ofproto.OFPP_FLOOD

match = parser.OFPMatch
(eth_type=ether_types.ETH_TYPE_IP,
in_port=in_port, ipv4_src=srcip,
ipv4_dst=dstip, ip_proto=protocol,
tcp_src=t.src_port, tcp_dst=t.dst_port)

actions =
[parser.OFPActionSetField(eth_dst=server2_mac),
parser.OFPActionSetField(ipv4_dst=server2_ip),
parser.OFPActionOutput(port=out_port)]
```

V. Testing and results

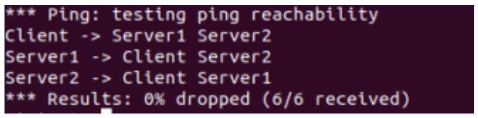
In this section, we conduct a series of tests on workability and performance of the controller applications based on the topology constructed in Mininet.

A. Testbed environment

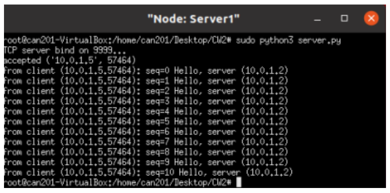
Tests are conducted in the same virtual machine (VM) and physical host where implementation is finished. The environment of VM and host is provided in detail in section IV. Moreover, the whole test is carried out in Mininet of which the version is 2.3.0.dev6. The network packet analysis software used for testing performance is Wireshark 3.2.3.

B. Test steps and results

1) Functional testing: During the testing process, we simulate a simple SDN networking topology using Mininet. We first run networkTopo.py (the topology file) to initiate the virtual topology. Then run ryu_forward.py (the controller applications) on controller's terminal to activate SDN network. To check whether each node in this topology connects with each other successfully, we use pingall to verify it. The snapshot is shown below which means The SDN topology we built can work as normal. Next, to check if the network can conduct TCP connection, do not need to close ryu-forward.py just


```
*** Ping: testing ping reachability
Client -> Server1 Server2
Server1 -> Client Server2
Server2 -> Client Server1
*** Results: 0% dropped (6/6 received)
```

wait for 5 seconds. First run server.py on both server2's and server1's terminal and then run client.py on client's terminal to check if all the packets are forwarded to Server1. The result shown below in the snapshot indicates that the TCP packets can be forwarded to server1 successfully.



```
"Node: Server1"
root@can201-VirtualBox:/home/can201/Desktop/CW2# sudo python3 server.py
TCP server bind on 9999...
accepted ('10.0.1.5', 57464)
from client (10.0.1.5.57464): seq=0 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=1 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=2 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=3 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=4 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=5 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=6 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=7 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=8 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=9 Hello, server (10.0.1.2)
from client (10.0.1.5.57464): seq=10 Hello, server (10.0.1.2)
root@can201-VirtualBox:/home/can201/Desktop/CW2# █
```

2) Performance comparison testing: Further, Wireshark is used to capture the network packets. Actually, we mainly focus on TCP segment. We run redirection and forwarding controller applications 10 times each and calculate the latency caused by TCP 3-way handshake (from the first SYN till the last ACK). Finally, we draw a curve diagram below to help us make a comparison.

According to the diagram, except the sixth time, the time consumed of traffic redirection is larger than simple forwarding. However, the only exception is caused by packet loss of SYN. Overall, the traffic network redirection will cause larger latency.

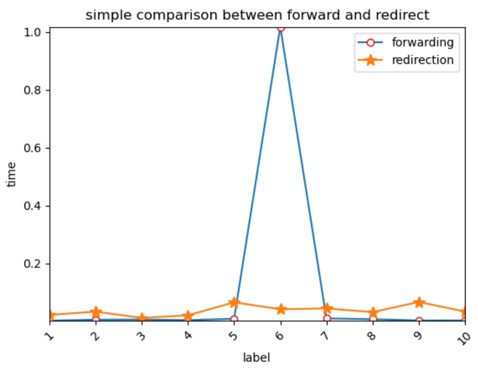


Fig. 4. simple comparison between forwarding and redirection

VI. Conclusion

SDN provides flexibility and programmability to networking management and it gains more popularity in recent years. In this project, we initially analyzed the task requirement document. Then, we prototyped a SDN network topology and designed algorithms for completing forwarding and redirection function. Furthermore, we programed two controller applications in the virtual machine on the basis of algorithm devised previously to implement the functionality required. Ultimately, we tested the developed controller application. The results show that applications can work. The networking latency obtained by Wireshark provide the performance comparison between the two applications, which suggested the high efficiency of the controller programs. The redirection program developed by our team can only distribute all traffics to another single server, which may result in failure of that server if excessive traffic occurs concurrently. Thus, future work can upgrade the functionality in redirection application, e.g., dynamically redistribute traffics to several servers.

Acknowledgement

Yuzhen Chen (2036328) contributes 33.33% to the project, Bowen Li (2033827) contributes 33.33% to the project, and Yizhen Cao (2034764) contributes 33.33% to the project.

References

[1] D. Tatang, F. Quinkert, J. Frank, C. Röpke, and T. Holz, "Sdn-guard: Protecting sdn controllers against sdn rootkits," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017, pp. 297–302.

[2] F. Travostino, "Seamless live migration of virtual machines over the man/wan," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 290–es. [Online]. Available: https://doi-org.ez.xjtlu.edu.cn/10.1145/1188455.1188758

[3] K. Onoue, Y. Oyama, and A. Yonezawa, "A virtual machine migration system based on a cpu emulator," in *First International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*, 2006, pp. 3–3.

[4] B. Borisaniya and D. Patel, "Towards virtual machine introspection based security framework for cloud." *Sādhanā: Published by the Indian Academy of Sciences*, vol. 44, no. 2, 2019.

[5] T. Boros, R. Bencel, and I. Kotuliak, "Transparent redirections in content delivery networks," *Applied Sciences*, vol. 9, no. 24, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/24/5418

[6] M. Ghaznavi, E. Jalalpour, M. A. Salahuddin, R. Boutaba, D. Migault, and S. Preda, "Content delivery network security: A survey," *IEEE Communications Surveys Tutorials*, vol. 23, no. 4, pp. 2166–2190, 2021.

[7] Y. Pu, Y. Deng, and A. Nakao, "Cloud rack: Enhanced virtual topology migration approach with open vswitch," in *The International Conference on Information Networking 2011 (ICOIN2011)*, 2011, pp. 160–164.

[8] C. Liu, S. Liu, N. Xing, S. Jin, Y. Ji, N. Zhang, and J. Tang, "A load balancing-based real-time traffic redirection method for edge networks," in *Simulation Tools and Techniques*, H. Song and D. Jiang, Eds. Cham: Springer International Publishing, 2021, pp. 34–43.

[9] P. Zanna, P. Radcliffe, and D. Kumar, "Preventing attacks on wireless networks using sdn controlled ooda loops and cyber kill chains." *Sensors (14248220)*, vol. 22, no. 23, p. 9481, 2022.