# CAN201 Assignment 2 Technical Report

Chengze Liu† 2142808

Gaojie Hou† 2144673

Jinai Ge† 2035693

Ruixuan Chen† 2144955

Zihan Deng† 2144196

***Abstract*—This network project aims to utilize the Mininet and Ryu frameworks to implement a simple Software-Defined Network (SDN) scenario. The project involves creating an SDN network topology, simulating traffic control functions, and implementing an SDN controller application to manipulate traffic. Specifically, the first part of the project involves creating flow entries after receiving packetpackets through SDN to achieve targeted forwarding of specific types of network traffic. The second part involves redirecting the traffic from a client to server 1 to server 2 without the client's awareness. This includes creating flow entries, capturing and analyzing network traffic, and calculating network latency.**

***Index Terms*—: SDN, buffer, mininet, redirection, forwarding**

## I. Introduction

Specification: Software Defined Networking (SDN) has emerged as a promising network management and control model, providing flexibility, programmability, and centralized management. This project aims to utilize SDN principles, using Mininet and Ryu frameworks to create a simple network scenario. The main goal is to achieve traffic control and manipulation in an SDN environment, such as dynamically managing specific types of traffic, capturing packets to calculate network latency, and redirecting traffic from clients to different servers without the clients' awareness.‘

Challenge: The key challenge addressed in this project is to develop and implement an SDN controller application that can dynamically manipulate traffic within the network. This involves addressing the complexity of flow entry creation, installation, and management to achieve seamless traffic redirection.

Practice Relevance: The proposed SDN scenario holds practical significance in various network applications, such as load balancing, secure traffic control, and enhancing network security. It involves dynamically managing network traffic by creating flow entries through packetin_SDN packets and redirecting traffic from one server to another without the client's knowledge. This project explores the potential for efficient resource utilization and enhanced network security.

Contributions: The primary contributions of this project include the design and implementation of an SDN network topology, the development of an SDN controller application using the Ryu framework, and the analysis of network latency in different traffic manipulation scenarios. Additionally, the project aims to gain a deeper understanding of the practical significance of SDN-based traffic control and its potential applications in real-world network environments.

†Equal contribution

## II. Related Work

Software Defined Networking (SDN) has emerged as a new networking paradigm that decouples the network control (management) plane which is the core of SDN architecture from the network data plane [1]. Therefore, the performance of the SDN architecture is largely determined by the performance of the controller. In this context, Ryu is regarded as a typical controller that performs well compared to other controllers such as OpenDaylight, Rosemary, Open Network Operating System (ONOS), and so on [2]. Ryu represents an open-source Python implementation of an SDN controller featuring a modular, component-based architecture. It offers a set of clearly defined APIs for the development of network applications and has gained significant adoption within the research community. In addition, Ryu provides support for various protocols in the Southbound Interface (SBI) for seamless hardware integration and configuration [3]. Request redirection is a typical tradeoff-based mechanism since each redirection consumes resources of the initially contacted server and contributes to the network time component of the user response time. The request redirection method can be categorized as either client-side or server-side redirection based on the point in a request's execution where the redirection decision is made [4].

## III. Design

### A. Task 1

Initially, a Mininet network topology was set up, comprising three hosts and one switch. More precisely, each host is connected to the switch, and the switch is linked to a remote controller. Through the use of the ‘markTerm' function, a terminal window was created for them which allows users to interact through the Mininet CLI. The configuration details for the entire network, including MAC addresses and IP addresses, were also set during this process. Finally, users can launch the Mininet CLI with ‘CLI(net) ‘and engage in interactive operations through the command-line interface.

### B. Task 2

In the ‘init ‘method, the application is initialized by setting the OpenFlow version, initializing the MAC address table ‘mac_to_port', and then invoke the initialization method of the parent class. Then a method for handling switch feature events is created and then install a flow entry which can forward the packet to the controller. After several steps, packet information can be extracted and analyzed. The main purpose of this application is to dynamically manage flow entries by learning source MAC addresses, determining output ports, and installing flow entries. The goal is to enhance the efficiency of the switch and minimize network congestion.

### C. Task 3

Task 3 requires the implementation of information transmission functionality based on the provided client-server code. It can be observed that the client utilizes TCP sockets to handle and send data, so TCP-related principles need to be considered in the program design. Figure 1 illustrates the process of TCP three-way handshake.

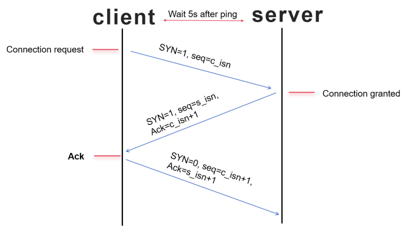


Fig. 1. TCP three-way handshake connection

Additionally, as described in Task 2, a five-second timeout callback is set; therefore, to ignore ICMP ping flow entries, it is necessary to wait for their refresh when performing this task.

### D. Task4

Task 4.1 requires the program to handle Packet_in SDN packets sent by the controller to the host. Specifically, when the switch receives a packet for which there is no corresponding flow table entry, it sends a PacketIn message to the controller. To control traffic, the controller needs to install new flow table rules on the switch upon receiving the first Packet_in, and subsequently send packet_out packets to the switch based on this flow table entry. To correctly and efficiently set up flow table entries, the controller first needs to analyze the key contents of the packet upon receiving Packet_in. Next, to prevent flooding in subsequent communications, it is necessary to design a dictionary data structure to store key-value pairs of MAC addresses and ports for the hosts that have been received. After learning the MAC addresses, flow table entries can be installed based on the information contained in the packet_in, preventing subsequent communications with the same attributes from causing packet_in and traffic congestion. Finally, the program should be designed to have the capability to construct and send packet_out packets. It is important to note that the PacketOut message should include an action, which the switch executes upon receiving PacketOut.

### E. Task5

Unlike Task 4, Task 5 involves redirecting all content sent from the clients to Server 1 to Server 2 through controller logic. The design algorithm for redirection is shown in Algorithm 1.

**Algorithm 1** Framework for controller handling of PacketIn packets for redirection

```
1:  if Packet is not TCP then
2:      if Destination is learned then
3:          Get the outport
4:      else
5:          Set flood action
6:      end if
7:      Install a flow to avoid packet_in next time.
8:      Construct packet_out message and send it.
9:  else
10:     if  Destination is Server1 and source is Client then
11:         Learn the MAC address
12:         Modify the action and argument to redirection.
13:         Install a flow and send packet_out
14:     else if Destination is client and source is server then
15:         Learn the MAC address
16:         Modify the action and argument to redirection.
17:         Install a flow and send packet_out
18:     else
19:         Log error message
20:     end if
21: end if
```

## IV. Implementation

The operating system of the computer is Windows 10, with Intel® Core™ i5-11400H @ 2.70GHz CPU and 16GB RAM. The program was developed on the Windows 22H2 operating system using Python 3.8.10 and JetBrains Pycharm as the IDE. Meanwhile, the program uses the Ryu and Mininet package as the development environment, which is version 4.34 and 2.3.0.dev6, respectively.

The implementation of the program is divided into three parts: networkTopo.py, which realizes the required network topology structure based on Mininet, and ryu_forward.py and ryu_redirect.py, which implement the controller logic based on Ryu. This section will discuss the specific implementation logic when using Mininet and Ryu, respectively.

### A. Mininet

The networkTopo.py implements a network topology structure based on Mininet, which includes a switch (switch1), a controller (controller), a client (client1), two servers (server1 and server2), and the corresponding links.

Before adding the topology structure, initialization is required for the hosts, controller, and switch. It is noteworthy that 'switch1' uses the Open vSwitch kernel switch and is set with a fault mode of 'secure'. Additionally, in line with practical considerations, the IP addresses and MAC addresses of the hosts need to be set effectively after building and starting the Mininet network. After establishing the required topology structure, for the convenience of allowing users to interact with the network, the program creates terminals for all objects.

## B. Ryu

Although the controller designs for the two tasks are not identical, their implementations are based on a similar framework. These two programs include the following key methods:

1) Initialization Constructor: This method is used to initialize the object. It calls the parent class constructor and initializes a dictionary (‘`self.mac_to_port`‘) to maintain the mapping of MAC addresses to switch ports.
2) Event Handlers: The program consists of two types of event handlers. In the Ryu framework, event handlers are distinguished using the ‘`@set_ev_cls`‘ decorator. Firstly, the ‘`switch_features_handler`‘ is responsible for handling the switch features event. The controller receives a FEATURE_REPLY message sent by the switch, and thus the event name is ‘EventOFPSwitchFeatures‘. Regarding the controller’s state, as this event requires waiting to receive the SwitchFeatures message, the state is set to ‘CONFIG_DISPATCHER‘. This method installs a table-miss flow entry to send packets to the controller.
   Secondly, the ‘`_packet_in_handler`‘ is used to handle the packet-in event in the OpenFlow protocol. When the switch cannot determine how to handle a received packet, it sends the packet to the controller, triggering this event. Therefore, the event name is ‘ofp_event.EventOFPPacketIn‘. ‘MAIN_DISPATCHER‘ signifies the main dispatch phase, where the event handler runs within the main dispatch loop. In this phase, the handler can perform more complex operations without blocking the main loop. This method is the main implementation body for forwarding requests of different tasks, and based on the designed programming logic, the primary packet handling functionality is implemented here.
3) Flow Table Entry Addition: The ‘`add_flow`‘ method constructs and sends a flow modification message to the switch, installing a flow entry. It constructs a flow_mod message and sends it to the switch.

## C. Difficulties

In the implementation of Task 5, some development difficulties were encountered during the redirection. Initially, despite observing that the flow table rules were defined as required in the switch outputs, the information was still not being correctly forwarded. After discussions and inspections, it was suspected that there might be an issue with the TCP three-way handshake process. Specifically, although the first handshake was successful in the redirection, the subsequent two handshakes did not get accepted correctly, resulting in the messages not being correctly forwarded. In the subsequent implementation, modifications were made to the actions in the OpenFlow protocol, and the communication between the client and server during redirection was handled differently based on specific scenarios. Finally, the task was successfully implemented.

## V. Testing and Results

The tests will be performed on a virtual machine with Ubuntu 20.04.4 LTS (64-bits) installed. The VM is assigned 4096MB of RAM. The virtual machine is still running on an Intel® Core™ i5-11400H @ 2.70GHz laptop.

### A. Testing Steps

First, the testing steps are as follows:

1) Run the following terminal command in the program path: `sudo python3 networkTopo.py`, The network structure will be built through Mininet and five terminal windows for controlling the nodes will be displayed.
2) For task 4, first run command ‘`ryu-manager ryu_forward.py`’ in the controller‘s terminal, then set up the client and server to send and reply to the messages. Figure 4 in the appendix shows the task4 running demonstration.
3) For task5, run command ‘`ryu-manager ryu_redirect.py`’ in the controller‘s terminal, the result is shown as Figure 5 in the appendix.
4) To check the flow table, run `sudo ovs-ofctl -O Openflow13 dump-flows switch1` code in switch1 terminal. It’s worth noting that the stream table can be refreshed on demand due to the 5-second idle being set.

### B. Testing Results

The test results for the two tasks will be demonstrated through the analysis of the delay in the TCP three-way handshake. Network latency refers to the time elapsed from the initiation of data transmission until the receiver receives the data. In the context of TCP connections, latency typically begins from the moment the sender sends the SYN segment and continues until the receiver sends the final ACK segment, indicating the completion of the TCP three-way handshake. In the testing environment, each task is set for three attempts, and the average of the three delays is taken for comparison between tasks. The delay test results for the tasks are shown in Figure 2.

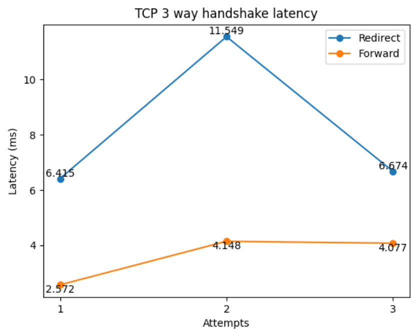


Fig. 2. The latency test results for Tasks 4.2 and 5.2.

It can be observed that, overall, the redirection latency is slightly higher than that of direct forwarding. Additionally, it is noteworthy that there are some latency fluctuations, which may be attributed to factors such as traffic congestion. The average latency illustrated in Figure 3 further confirms that the redirection latency is higher than the latency associated with forwarding.

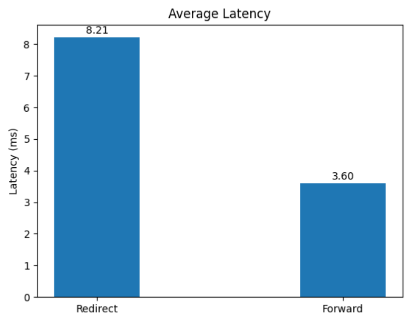


Fig. 3. The average latency for Tasks 4.2 and 5.2.

In conclusion, the involved traffic controllers have achieved the expected effectiveness.

## VI. Conclusion

Through a series of design, implementation, and testing operations, our team has built an SDN network architecture with a controller capable of efficient flow control operations. Additionally, direct sending and redirection of client messages were implemented as required. In terms of testing, to more accurately estimate the network's performance, each task underwent multiple tests, and the average values were taken. Moreover, some considerations for future work are necessary. First, during the development process, we did not emphasize the robustness of the system. In other words, it is essential to consider optimizing the constraints on flow table entries to prevent flooding and to enhance the system's resilience against attacks such as DoS. Secondly, security considerations for the system are also a future optimization path to ensure that the SDN network is not easily vulnerable to unauthorized access and malicious operations during runtime. Research and optimization in these aspects will ensure the robust operation of SDN networks in various environments.

## VII. Acknowledgement

Gaojie Hou (2144673) contributes 20% to the project, Zihan Deng (2144196) contributes 20% to the project, Chengze Liu (2142808) contributes 20% to the project, Ruixuan Chen (2144955) contributes 20% to the project, and Jinai Ge (2035693) contributes 20% to the project.

## References

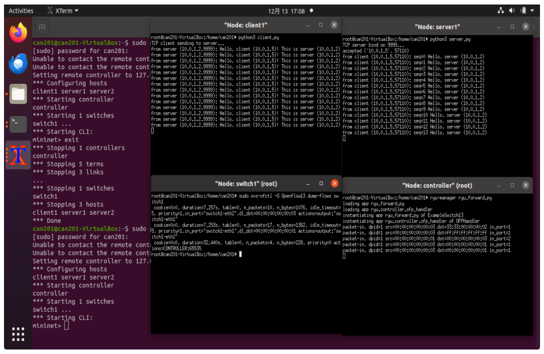
[1] R. Amin, M. Reisslein, and N. Shah, "Hybrid sdn networks: A survey of existing approaches," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3259–3306, 2018.

[2] Y. Li, X. Guo, X. Pang, B. Peng, X. Li, and P. Zhang, "Performance analysis of floodlight and ryu sdn controllers under mininet simulator," in *2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. IEEE, 2020, pp. 85–90.

[3] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, "Security analysis of opendaylight, onos, rosemary and ryu sdn controllers," in *2016 17th International telecommunications network strategy and planning symposium (Networks)*. IEEE, 2016, pp. 37–44.

[4] V. Cardellini, M. Colajanni, and P. S. Yu, "Request redirection algorithms for distributed web systems," *IEEE transactions on parallel and distributed systems*, vol. 14, no. 4, pp. 355–368, 2003.

## VIII. Appendix



```
can201@can201-VirtualBox:~$ sudo
[sudo] password for can201:
Unable to contact the remote cont
Unable to contact the remote cont
Setting remote controller to 127.
*** Configuring hosts
client1 server1 server2
*** Starting controller
controller
*** Starting 1 switches
switch1 ...
*** Starting CLI:
mininet> exit
*** Stopping 1 controllers
controller
*** Stopping 5 terms
*** Stopping 3 links
...
*** Stopping 1 switches
switch1
*** Stopping 3 hosts
client1 server1 server2
*** Done
can201@can201-VirtualBox:~$ sudo
[sudo] password for can201:
Unable to contact the remote cont
Unable to contact the remote cont
Setting remote controller to 127.
*** Configuring hosts
client1 server1 server2
*** Starting controller
controller
*** Starting 1 switches
switch1 ...
*** Starting CLI:
mininet> 
```

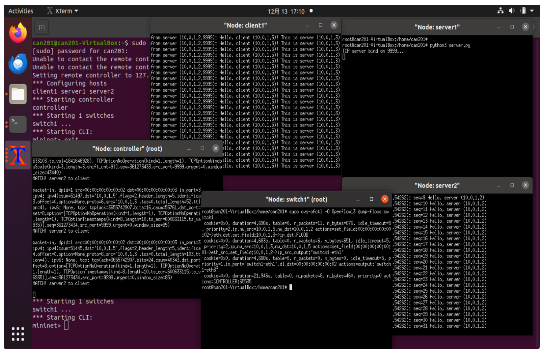
```
root@can201-VirtualBox:/home/can201# python3 client.py
TCP client sending to server...
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.2)
```

```
root@can201-VirtualBox:/home/can201# python3 server.py
TCP server bind on 9999...
accepted ('10.0.1.5', 57110)
from client (10.0.1.5.57110): seq=0 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=1 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=2 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=3 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=4 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=5 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=6 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=7 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=8 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=9 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=10 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=11 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=12 Hello, server (10.0.1.2)
from client (10.0.1.5.57110): seq=13 Hello, server (10.0.1.2)
```

```
root@can201-VirtualBox:/home/can201# sudo ovs-ofctl -O Openflow13 dump-flows sw
itch1
 cookie=0x0, duration=7.257s, table=0, n_packets=10, n_bytes=1076, idle_timeout=
5, priority=1,in_port="switch1-eth2",dl_dst=00:00:00:00:00:03 actions=output:"sw
itch1-eth1"
 cookie=0x0, duration=7.253s, table=0, n_packets=17, n_bytes=1362, idle_timeout=
5, priority=1,in_port="switch1-eth1",dl_dst=00:00:00:00:00:01 actions=output:"sw
itch1-eth2"
 cookie=0x0, duration=32.440s, table=0, n_packets=4, n_bytes=228, priority=0 act
ions=CONTROLLER:65535
root@can201-VirtualBox:/home/can201# 
```

```
root@can201-VirtualBox:/home/can201# ryu-manager ryu_forward.py
loading app ryu_forward.py
loading app ryu.controller.ofp_handler
instantiating app ryu_forward.py of ExampleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet-in, dpid=1 src=00:00:00:00:00:03 dst=33:33:00:00:00:02 in_port=1
packet-in, dpid=1 src=00:00:00:00:00:03 dst=ff:ff:ff:ff:ff:ff in_port=1
packet-in, dpid=1 src=00:00:00:00:00:01 dst=00:00:00:00:00:03 in_port=2
packet-in, dpid=1 src=00:00:00:00:00:03 dst=00:00:00:00:00:01 in_port=1
packet-in, dpid=1 src=00:00:00:00:00:03 dst=00:00:00:00:00:01 in_port=1
```

Fig. 4. Task4.1 running results



```
can201@can201-VirtualBox:~$ sudo
[sudo] password for can201:
Unable to contact the remote cont
Unable to contact the remote cont
Setting remote controller to 127.
*** Configuring hosts
client1 server1 server2
*** Starting controller
controller
*** Starting 1 switches
switch1 ...
*** Starting CLI:
mininet> exit
*** Starting 1 switches
switch1 ...
*** Starting CLI:
mininet> 
```

```
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
from server (10.0.1.2.9999): Hello, client (10.0.1.5)! This is server (10.0.1.3)
ello, client (10.0.1.5)! This is server (10.0.1.3)
ello, client (10.0.1.5)! This is server (10.0.1.3)
ello, client (10.0.1.5)! This is server (10.0.1.3)
ello, client (10.0.1.5)! This is server (10.0.1.3)
ello, client (10.0.1.5)! This is server (10.0.1.3)
```

```
root@can201-VirtualBox:/home/can201#
root@can201-VirtualBox:/home/can201# python3 server.py
TCP server bind on 9999...
```

```
633103,ts_val=1841646928), TCPOptionNoOperation(kind=1,length=1), TCPOptionWindo
wScale(kind=3,length=3,shift_cnt=9)],seq=361273433,src_port=9999,urgent=0,window
_size=43440)
MATCH! server2 to client

packet-in, dpid=1 src=00:00:00:00:00:02 dst=00:00:00:00:00:03 in_port=3
ipv4: ipv4(csum=51497,dst='10.0.1.5',flags=2,header_length=5,identifica
3,offset=0,option=None,proto=6,src='10.0.1.3',tos=0,total_length=52,ttl
on=4), ipv6: None, tcp: tcp(ack=3655742907,bits=16,csum=55761,dst_port=
set=8,option=[TCPOptionNoOperation(kind=1,length=1), TCPOptionNoOperati
,length=1), TCPOptionTimestamps(kind=8,length=10,ts_ecr=600633115,ts_va
935)],seq=361273434,src_port=9999,urgent=0,window_size=85)
MATCH! server2 to client

packet-in, dpid=1 src=00:00:00:00:00:02 dst=00:00:00:00:00:03 in_port=3
ipv4: ipv4(csum=51445,dst='10.0.1.5',flags=2,header_length=5,identifica
4,offset=0,option=None,proto=6,src='10.0.1.3',tos=0,total_length=103,tt
ion=4), ipv6: None, tcp: tcp(ack=3655742907,bits=24,csum=40343,dst_port
fset=8,option=[TCPOptionNoOperation(kind=1,length=1), TCPOptionNoOperat
1,length=1), TCPOptionTimestamps(kind=8,length=10,ts_ecr=600633115,ts_v
6935)],seq=361273434,src_port=9999,urgent=0,window_size=85)
MATCH! server2 to client

```

```
root@can201-VirtualBox:/home/can201# sudo ovs-ofctl -O Openflow13 dump-flows sw
itch1
 cookie=0x0, duration=4.696s, table=0, n_packets=11, n_bytes=876, idle_timeout=5
, priority=2,ip,nw_src=10.0.1.5,nw_dst=10.0.1.2 actions=set_field:00:00:00:00:00
:02->eth_dst,set_field:10.0.1.3->ip_dst,FLOOD
 cookie=0x0, duration=4.683s, table=0, n_packets=6, n_bytes=651, idle_timeout=5,
 priority=2,ip,nw_src=10.0.1.3,nw_dst=10.0.1.5 actions=set_field:00:00:00:00:00:
01->eth_src,set_field:10.0.1.2->ip_src,output:"switch1-eth1"
 cookie=0x0, duration=4.688s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, p
riority=1,in_port="switch1-eth1",dl_dst=00:00:00:00:00:02 actions=output:"switch
1-eth3"
 cookie=0x0, duration=21.946s, table=0, n_packets=8, n_bytes=460, priority=0 act
ions=CONTROLLER:65535
root@can201-VirtualBox:/home/can201# 
```

```
.54262): seq=9 Hello, server (10.0.1.2)
.54262): seq=10 Hello, server (10.0.1.2)
.54262): seq=11 Hello, server (10.0.1.2)
.54262): seq=12 Hello, server (10.0.1.2)
.54262): seq=13 Hello, server (10.0.1.2)
.54262): seq=14 Hello, server (10.0.1.2)
.54262): seq=15 Hello, server (10.0.1.2)
.54262): seq=16 Hello, server (10.0.1.2)
.54262): seq=17 Hello, server (10.0.1.2)
.54262): seq=18 Hello, server (10.0.1.2)
.54262): seq=19 Hello, server (10.0.1.2)
.54262): seq=20 Hello, server (10.0.1.2)
.54262): seq=21 Hello, server (10.0.1.2)
.54262): seq=22 Hello, server (10.0.1.2)
.54262): seq=23 Hello, server (10.0.1.2)
.54262): seq=24 Hello, server (10.0.1.2)
.54262): seq=25 Hello, server (10.0.1.2)
.54262): seq=26 Hello, server (10.0.1.2)
.54262): seq=27 Hello, server (10.0.1.2)
.54262): seq=28 Hello, server (10.0.1.2)
.54262): seq=29 Hello, server (10.0.1.2)
.54262): seq=30 Hello, server (10.0.1.2)
.54262): seq=31 Hello, server (10.0.1.2)
```

Fig. 5. Task5.1 running results