

# Final Project Report

## Open Flow

EE 555

Group member

Huang, hui (huih@usc.edu)

Zhou, wenyan ([wenyanzh@usc.edu](mailto:wenyanzh@usc.edu))

Lu, wanyun (wanyunlu@usc.edu)

### Abstract:

Open flow is an advance communication protocol, which can use the forwarding plane of a network switch or router. In this project, our group follow the instruction of open flow wiki to get our virtual network. Our group choose pox which is a python based SDN controller platform to modify our network. From this project, all of us get a better understanding of ARP and ICMP protocol.

### Part 1:

#### 1. Learning switch

It is the important basic knowledge for whole project. We use the tutorial material to create a learning switch.

- a. Create a mac-to-port table
- b. Switch receive a packet and check its destination mac address in table  
Controller will record the source mac address and the port number that packet comes in.
- c. If the table doesn't have a port number to match this mac address, then controller will flood this packet.
- d. If the table have the matching, controller will send this packet to destined port.

Following the instruction, we use the `ofp_flow_mod` OpenFlow message to improve the performance of switch.

## Testing controller:

```
root@mininet-vm:~# ping -c 10,0,0,2
Usage: ping [-aAbBdDfHhLnOqRrUVv] [-c count] [-i interval] [-I interface]
[-m mark] [-M mtu] [-n n] [-p pattern] [-q tos]
[-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
[-w deadline] [-W timeout] [hop1 ...] destination
root@mininet-vm:~# ping -c 10,0,0,2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=45.9 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/ndev = 45.903/45.903/45.903/0.000 ms
root@mininet-vm:~# ping -c 10,0,0,5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data:
From 10.0.0.1: icmp_seq=1 Destination Host Unreachable

--- 10.0.0.5 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@mininet-vm:~#
```

```
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""#$%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 .....67
01:14:59.083283 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0002 0a00 0002 .....
0x0020: 0000 0000 0000 0a00 0001 .....
01:14:59.136729 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0002 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0002 0a00 0002 .....
01:16:16.770330 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
01:16:17.743934 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
01:16:18.770292 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
```

```
mininet@localhost's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Mon Apr 24 01:08:48 2017 from 10.0.2.2
mininet@mininet-vm:~$ cd pox
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.Switch_1
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.Switch_1:Controlling [00-00-00-00-00-01 1]
DEBUG:misc.Switch_1:Installing flow 2, 1
DEBUG:misc.Switch_1:Installing flow 1, 2
DEBUG:misc.Switch_1:Installing flow 2, 1
DEBUG:misc.Switch_1:Installing flow 2, 1
DEBUG:misc.Switch_1:Installing flow 1, 2
DEBUG:misc.Switch_1:Installing flow 1, 2
```

```
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
01:14:54.084660 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0002 .....
01:16:16.770328 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
01:16:17.743932 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
01:16:18.770290 ARP, Request who-has 10.0.0.5 tell 10.0.0.1, length 28
0x0000: ffff ffff ffff 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0000 0a00 0005 .....
```

According to the figure, we can clearly see the ARP request and Reply message. At first try, we use h1 to ping h3. Due to the empty table, all the ports except input port will receive the packet, but only h3 will reply.

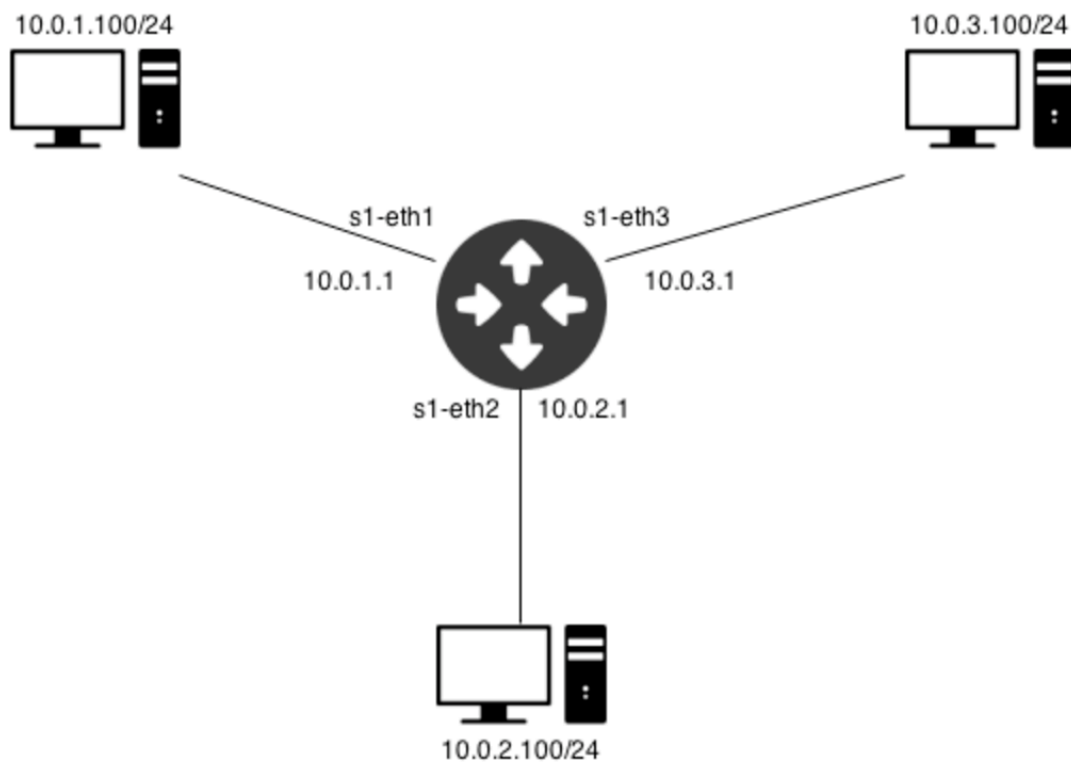
```
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['21.8 Gbits/sec', '21.8 Gbits/sec']
mininet>
```

```
DEBUG:misc.Switch_1:Installing flow 2, 1
DEBUG:misc.Switch_1:Installing flow 3, 1
DEBUG:misc.Switch_1:Installing flow 1, 3
DEBUG:misc.Switch_1:Installing flow 3, 1
DEBUG:misc.Switch_1:Installing flow 2, 1
DEBUG:misc.Switch_1:Installing flow 1, 2
DEBUG:misc.Switch_1:Installing flow 3, 2
DEBUG:misc.Switch_1:Installing flow 2, 3
DEBUG:misc.Switch_1:Installing flow 3, 2
DEBUG:misc.Switch_1:Installing flow 3, 1
DEBUG:misc.Switch_1:Installing flow 1, 3
DEBUG:misc.Switch_1:Installing flow 2, 3
DEBUG:misc.Switch_1:Installing flow 2, 3
DEBUG:misc.Switch_1:Installing flow 1, 3
DEBUG:misc.Switch_1:Installing flow 3, 1
DEBUG:misc.Switch_1:Installing flow 1, 3
DEBUG:misc.Switch_1:Installing flow 3, 1
DEBUG:misc.Switch_1:Installing flow 3, 1
```

According the figure, we use the “pingall” to test all the hosts and the result shows every host can get the packet. After that, we use “iperf” to test the bandwidth and we can see it is 21.8 Gbits/sec.

## 2. Router Exercise

First of all, we build the topology which contains one router and 3 hosts.



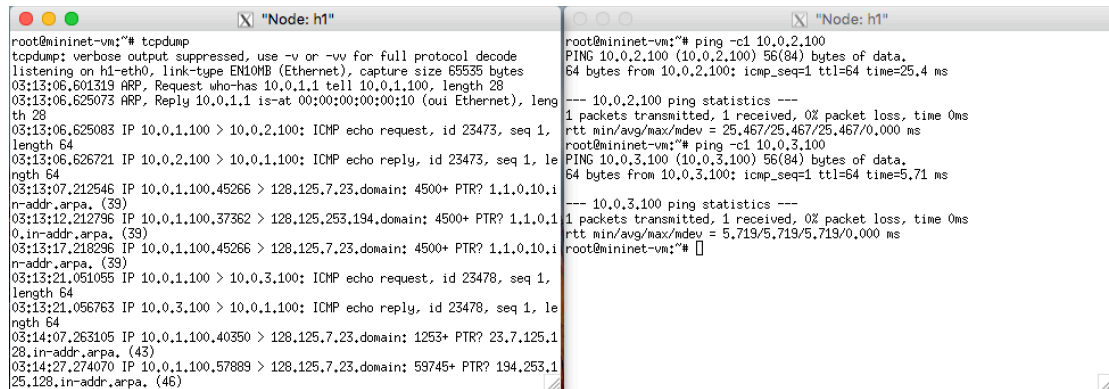
- a. Router gets a packet
- b. If this packet is an ARP packet
  - We get two type of ARP packets
    1. This is an ARP request packet
      - Router will send back an ARP reply packet to tell the source that router is the next node when it need to send a packet.
      - The source will get the router's mac address.
    2. This is an ARP reply packet
      - Router will forward this packet to its destination.
- c. If this is an ICMP packet
  - We get two type of ICMP packet
    1. This is an ICMP echo request
      - This packet's destination IP address is this router. The router will send back an echo packet to source.
      - This packet's destination IP address is a host. The router will create a new frame and forward this frame to its destination.
      - This packet's destination IP address is as unknown address. The router will send back a ICMP unreachable packet.
    2. This is an ICMP reply packet

Router just send this packet to the destination.

d. If this is a normal IP packet

Router will just send it to its destination IP address.

Testing Controller:



```
root@mininet-vx:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
03:13:06.601319 ARP, Request who-has 10.0.1.1 tell 10.0.1.100, length 28
03:13:06.625073 ARP, Reply 10.0.1.1 is-at 00:00:00:00:00:10 (oui Ethernet), length 28
03:13:07.212546 IP 10.0.1.100 > 10.0.2.100: ICMP echo request, id 23473, seq 1, length 64
03:13:06.626721 IP 10.0.2.100 > 10.0.1.100: ICMP echo reply, id 23473, seq 1, length 64
03:13:07.212546 IP 10.0.1.100,45266 > 128.125.7.23.domain: 4500+ PTR? 1.1.0.10.1 n-addr.arpa. (39)
03:13:12.212796 IP 10.0.1.100,37362 > 128.125.253.194.domain: 4500+ PTR? 1.1.0.10.1 n-addr.arpa. (39)
03:13:17.218296 IP 10.0.1.100,45266 > 128.125.7.23.domain: 4500+ PTR? 1.1.0.10.1 n-addr.arpa. (39)
03:13:21.051095 IP 10.0.1.100 > 10.0.3.100: ICMP echo request, id 23478, seq 1, length 64
03:13:21.056763 IP 10.0.3.100 > 10.0.1.100: ICMP echo reply, id 23478, seq 1, length 64
03:14:07.263105 IP 10.0.1.100,40350 > 128.125.7.23.domain: 1253+ PTR? 23.7.125.128.in-addr.arpa. (43)
03:14:27.274070 IP 10.0.1.100,57889 > 128.125.7.23.domain: 59745+ PTR? 194.253.125.128.in-addr.arpa. (46)
```

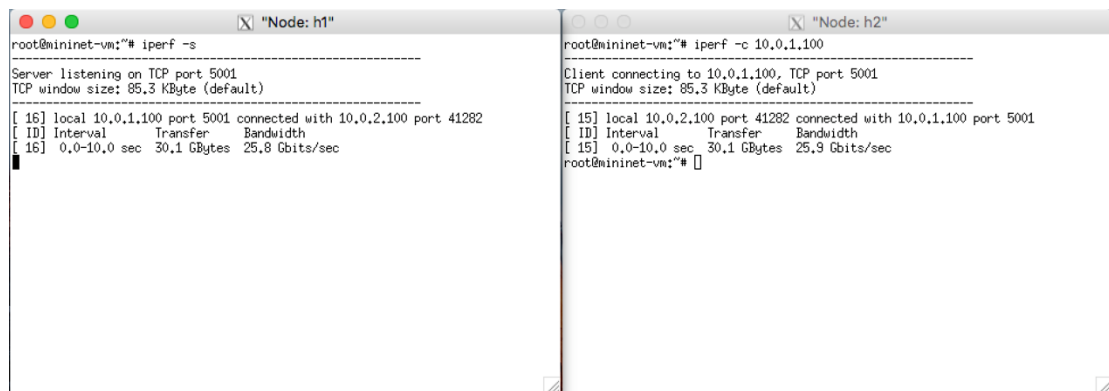
```
root@mininet-vx:~# ping -c 1 10.0.2.100
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data:
64 bytes from 10.0.2.100: icmp_seq=1 ttl=64 time=25.4 ms

--- 10.0.2.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.467/25.467/25.467/0.000 ms
root@mininet-vx:~# ping -c 1 10.0.3.100
PING 10.0.3.100 (10.0.3.100) 56(84) bytes of data:
64 bytes from 10.0.3.100: icmp_seq=1 ttl=64 time=5.71 ms

--- 10.0.3.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.719/5.719/5.719/0.000 ms
root@mininet-vx:~#
```

We first send packet to “10.0.2.100”.

According to figure, host 1 want to send packet to “10.0.2.100”. it needs know router’s mac address, so there are ARP request and ARP reply message. H1 also need to know that if the router can reach the “10.0.2.100” or not, so there exist ICMP echo request and reply.



```
root@mininet-vx:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 16] local 10.0.1.100 port 5001 connected with 10.0.2.100 port 41282
[ ID] Interval Transfer Bandwidth
[ 16] 0.0-10.0 sec 30.1 GBytes 25.8 Gbits/sec
```

```
root@mininet-vx:~# iperf -c 10.0.1.100
Client connecting to 10.0.1.100, TCP port 5001
TCP window size: 85.3 KByte (default)

[ 15] local 10.0.2.100 port 41282 connected with 10.0.1.100 port 5001
[ ID] Interval Transfer Bandwidth
[ 15] 0.0-10.0 sec 30.1 GBytes 25.9 Gbits/sec
root@mininet-vx:~#
```

According to figure, we use the host 1 as sever and host 2 as client.

This is an example for normal IP packet. From the figure, the bandwidth and port number show very clear.

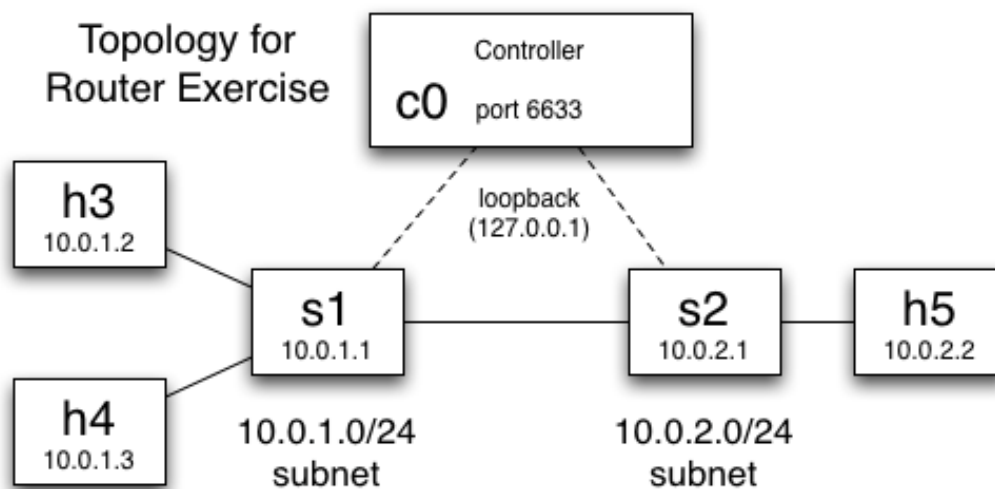
The image shows two terminal windows. The left window displays the Mininet setup process, including adding hosts (h1, h2, h3), switches (s1), and links. It shows the controller (c0) starting, and the CLI being started. Ping tests show 0% dropped packets, and an iperf test shows a bandwidth of 24.5 Gbits/sec. The right window shows the logs for the Mininet environment, including the last login, the user's name (Wenyans), and the version of the Mininet environment (POX 0.2.0).

According to figure, we can check that the router is pingable. With the setting of the flow mod, the bandwidth will be increased to 24.5 Gbits/sec.

## Part 2:

### 1. Advance Topology

First of all, we build the topology which contains two routers and 3 hosts.



This part is very similar with the router exercise, but we encounter with some problem in this part.

#### a. We need to find the correct port number for two routers

At the beginning of programming, our group think the router's port number can be changed by ourselves. We just set the port number like "1, 2, 3, 4, 5", but we can't get a feasible connection. Then we use "log.dug" method to help us print the correct port number which is "1, 2, 3, 1, 2".

#### b. We need to specify each router's behavior

##### 1. ARP part

If host 3 want host 4's mac address, it is very simple. The router 1 just forward this request packet to host 4.

If destination IP address of the ARP request packet is router's address, router will send back a reply packet and tell them router's mac address.

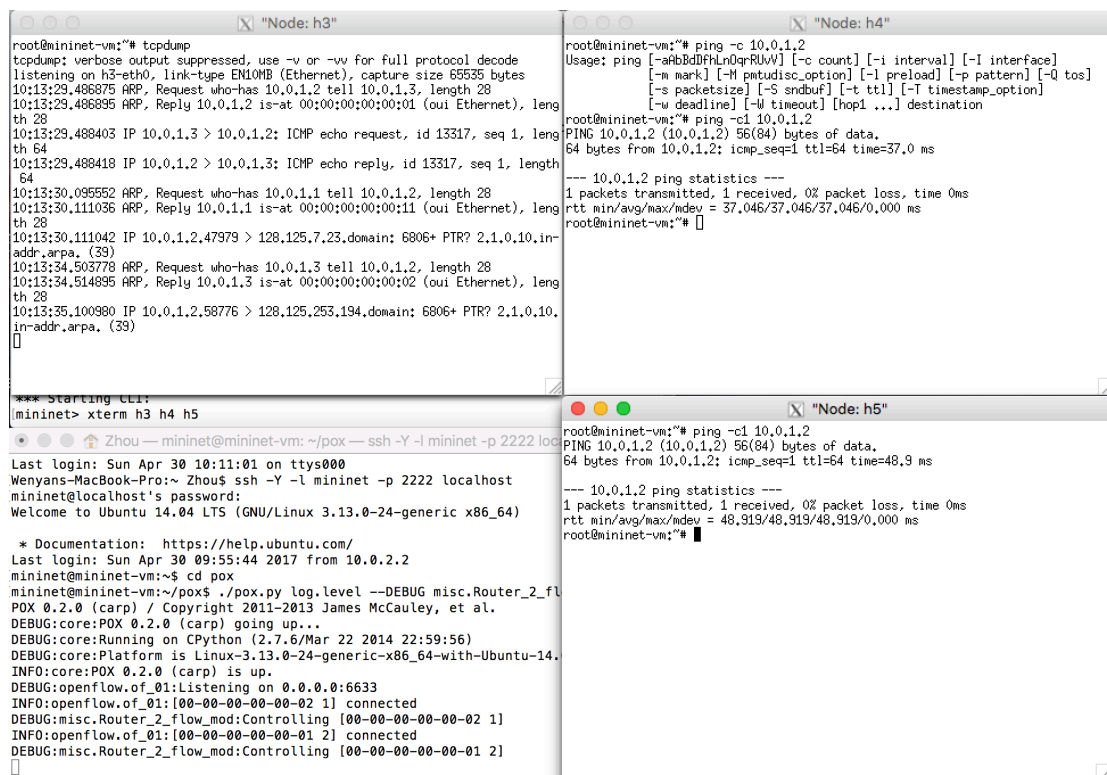
ARP reply packet is same with part 1.

## 2. IP part

ICMP request packet. If the packet's destination mac address is router1 and destination IP address is router 1, router 1 will send a reply packet. If the destination IP address is in subnet 1, then send it to the distinct port. If the destination IP address is in subnet 2, then send it to router 2.

ICMP reply and Normal IP packet should also follow the same way that has mentioned above.

## Testing Controller:



```
root@mininet-vm:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
10:13:29.486875 ARP, Request who-has 10.0.1.2 tell 10.0.1.3, length 28
10:13:29.486895 ARP, Reply 10.0.1.2 is-at 00:00:00:00:00:01 (oui Ethernet), length 28
10:13:29.488403 IP 10.0.1.3 > 10.0.1.2: ICMP echo request, id 13317, seq 1, length 64
10:13:29.488418 IP 10.0.1.2 > 10.0.1.3: ICMP echo reply, id 13317, seq 1, length 64
10:13:30.095552 ARP, Request who-has 10.0.1.1 tell 10.0.1.2, length 28
10:13:30.111036 ARP, Reply 10.0.1.1 is-at 00:00:00:00:00:11 (oui Ethernet), length 28
10:13:30.111042 IP 10.0.1.2,47979 > 128.125.7.23.domain: 6806+ PTR? 2.1.0.10.in-addr.arpa. (39)
10:13:34.503778 ARP, Request who-has 10.0.1.3 tell 10.0.1.2, length 28
10:13:34.514895 ARP, Reply 10.0.1.3 is-at 00:00:00:00:00:02 (oui Ethernet), length 28
10:13:35.100980 IP 10.0.1.2,58776 > 128.125.253.194.domain: 6806+ PTR? 2.1.0.10.in-addr.arpa. (39)
[]

*** Starting CL1:
mininet> xterm h3 h4 h5

Last login: Sun Apr 30 10:11:01 on ttys000
Wenyans-MacBook-Pro:~ Zhou$ ssh -Y -l mininet -p 2222 localhost
mininet@localhost's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun Apr 30 09:55:44 2017 from 10.0.2.2
mininet@mininet-vm:~$ cd pox
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.Router_2_flow_mod
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-02 1] connected
DEBUG:misc.Router_2_flow_mod:Controlling [00-00-00-00-00-02 1]
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.Router_2_flow_mod:Controlling [00-00-00-00-00-01 2]
[]

root@mininet-vm:~# ping -c 10.0.1.2
Usage: ping [-sAbBdDfHlNOpRUVw] [-c count] [-i interval] [-I interface]
          [-m mark] [-M mtu] [-n mtu] [-l preload] [-p pattern] [-Q tos]
          [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
          [-w deadline] [-W timeout] [hop1 ...] destination
root@mininet-vm:~# ping -c 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data:
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=37.0 ms

--- 10.0.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 37.046/37.046/37.046/0.000 ms
root@mininet-vm:~#

root@mininet-vm:~# ping -c 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data:
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=48.9 ms

--- 10.0.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 48.913/48.913/48.913/0.000 ms
root@mininet-vm:~#
```

To the text our router, we make host 3 to listen the network. In host 4 and host 5 which can represent the same and different network, we type the “ping 10.0.1.2”. That is to say, host 4 and 5 send packet to host 3. According to the figure, we can clear see the ARP request and reply packet. All the packets are sent successfully.

```
root@mininet-vm:~# ping -c1 10.0.5.5
PING 10.0.5.5 (10.0.5.5) 56(84) bytes of data.
From 10.0.5.5 icmp_seq=1 Destination Net Unreachable

--- 10.0.5.5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@mininet-vm:~#
```

In this figure, we make host 5 to a unknown IP address and the packet was lost.

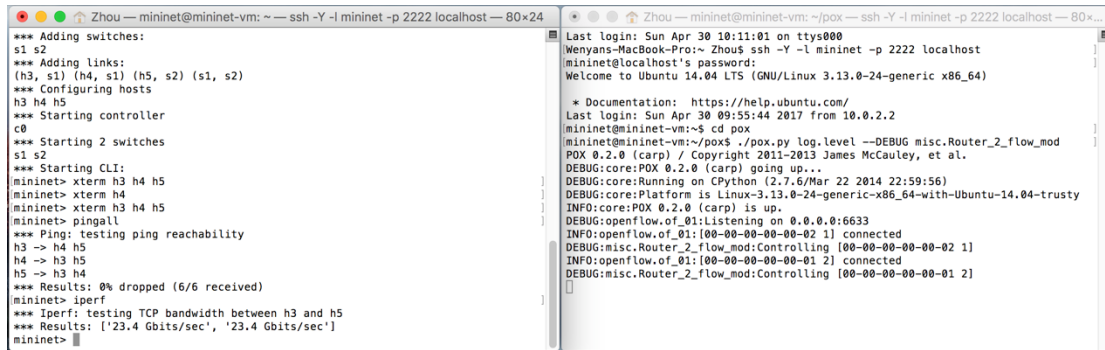
```
root@mininet-vm:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 18] local 10.0.1.2 port 5001 connected with 10.0.2.2 port 53333
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  19.8 GBytes 17.0 Gbits/sec

root@mininet-vm:~# iperf -c 10.0.1.2
Client connecting to 10.0.1.2, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 17] local 10.0.2.2 port 53333 connected with 10.0.1.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  19.8 GBytes 17.0 Gbits/sec

root@mininet-vm:~#
```

In this figure, we test the bandwidth between host 3 and host 5.



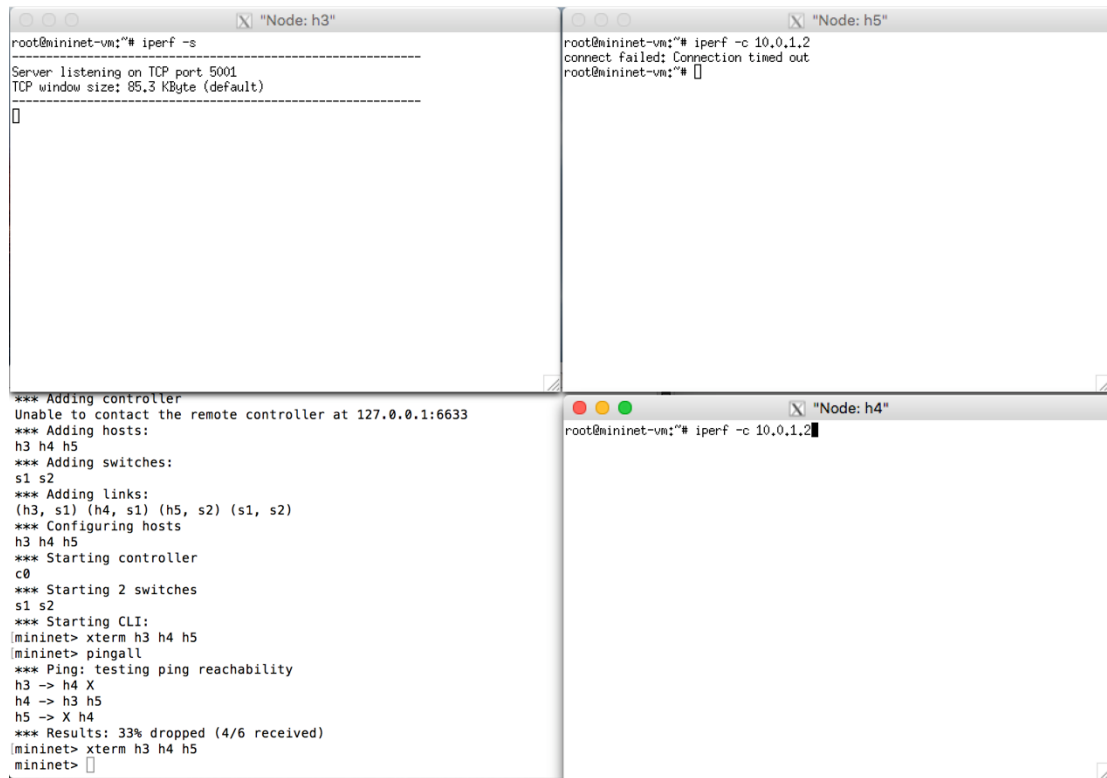
The image shows two terminal windows side-by-side. The left window is a Mininet VM terminal with the prompt 'mininet@mininet-vm: ~'. It shows the process of adding switches (s1, s2), links (h3, s1), (h4, s1), (h5, s2), (s1, s2), configuring hosts h3, h4, h5, starting a controller c0, and starting two switches s1 and s2. It then starts the CLI, runs 'xterm h3 h4 h5', 'xterm h3 h4 h5', and 'pingall'. It also shows ping results for h3 to h4, h4 to h5, and h5 to h3, all successful. Finally, it runs 'iperf' and shows results: '23.4 Gbits/sec'. The right window is a POX VM terminal with the prompt 'mininet@mininet-vm: ~/pox'. It shows the last login, the password, and the documentation URL. It then shows the command 'pox' being run, which starts the POX 0.2.0 (carp) process. It shows the core running on Python 2.7.6, the platform is Linux-3.13.0-24-generic-x86\_64-with-Ubuntu-14.04-trusty, and the core is up. It also shows the openflow of\_01 listening on 0.0.0.0:6633, and the openflow of\_01 connected to the POX core. It shows the core controlling the openflow of\_01 and the openflow of\_01 connected to the core. It shows the core controlling the openflow of\_01 and the openflow of\_01 connected to the core.

According to this figure, it shows all the result for our work. Both of two routers are pingable. With the help of flow mod, we can send large amount of packet at the same time, which give our network a 23.4 Gbits/sec bandwidth.

## 2. Firewall

To realize the function of firewall, we created an array to store the information of the source mac address and destination mac address which should be cut off in the communication. In this experiment, in the self.deny array, we recorded the mac address of h3 and the mac address of the default router of h3. As a result, we can cut of the communication between the host3 and the router1. Hosts from the outside network can no longer send message to h3 because the packet must go through the default router. In this situation, we also cut off the ICMP reply packet destined to host 3. So, h3 can not send packets out of the network either. Because, communication between h3 and h4 does not need to the help of router, so this cut of does not influent the packets transmission between h3 and h4. In the terminal, we can see that only h3 and h5 line has lost their packets.

## Testing Controller:



```
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[]

*** Adding controller
*** Adding hosts:
h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h3, s1) (h4, s1) (h5, s2) (s1, s2)
*** Configuring hosts
h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2
*** Starting CLI:
[mininet> xterm h3 h4 h5
[mininet> pingall
*** Ping: testing ping reachability
h3 -> h4 X
h4 -> h3 h5
h5 -> X h4
*** Results: 33% dropped (4/6 received)
[mininet> xterm h3 h4 h5
[mininet> []

root@mininet-vm:~# iperf -c 10.0.1.2
connect failed: Connection timed out
root@mininet-vm:~# []

root@mininet-vm:~# iperf -c 10.0.1.2
```

According to this figure, we first make the host 3 as a server. Then at host 5, we need to get connection to host 3. However, we waited for a minute, it shows that connection time out.

```

root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 18] local 10.0.1.2 port 5001 connected with 10.0.1.3 port 59597
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  23.5 GBytes 20.2 Gbits/sec
root@mininet-vm:~#

*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h3, s1) (h4, s1) (h5, s2) (s1, s2)
*** Configuring hosts
h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2
*** Starting CLI:
mininet> xterm h3 h4 h5
mininet> pingall
*** Ping: testing ping reachability
h3 -> h4 X
h4 -> h3 h5
h5 -> X h4
*** Results: 33% dropped (4/6 received)
mininet> xterm h3 h4 h5
mininet>

root@mininet-vm:~# iperf -c 10.0.1.2
connect failed: Connection timed out
root@mininet-vm:~#

root@mininet-vm:~# iperf -c 10.0.1.2
-----
Client connecting to 10.0.1.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 17] local 10.0.1.3 port 59597 connected with 10.0.1.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  23.5 GBytes 20.2 Gbits/sec
root@mininet-vm:~#

```

In this figure, we can see that host 4 can get connection to host 3, because they are in the same network.

Both of two figures, we can see that host 3 has build a fire wall to deny all the packet which comes from other network.

```

*** Creating network
*** Adding controller
*** Adding hosts:
h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h3, s1) (h4, s1) (h5, s2) (s1, s2)
*** Configuring hosts
h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2
*** Starting CLI:
mininet> xterm h3 h4 h5
mininet> xterm h5
mininet> pingall
*** Ping: testing ping reachability
h3 -> h4 X
h4 -> h3 h5
h5 -> X h4
*** Results: 33% dropped (4/6 received)
mininet>

Last login: Sun Apr 30 10:36:48 on ttys000
Wenyans-MacBook-Pro:~ Zhou$ ssh -Y -l mininet -p 2222 localhost
mininet@localhost's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun Apr 30 10:11:37 2017 from 10.0.2.2
mininet@mininet-vm:~$ cd pox
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.FireWall
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.FireWall:Controlling [00-00-00-00-00-01 2]
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
DEBUG:misc.FireWall:Controlling [00-00-00-00-00-02 3]

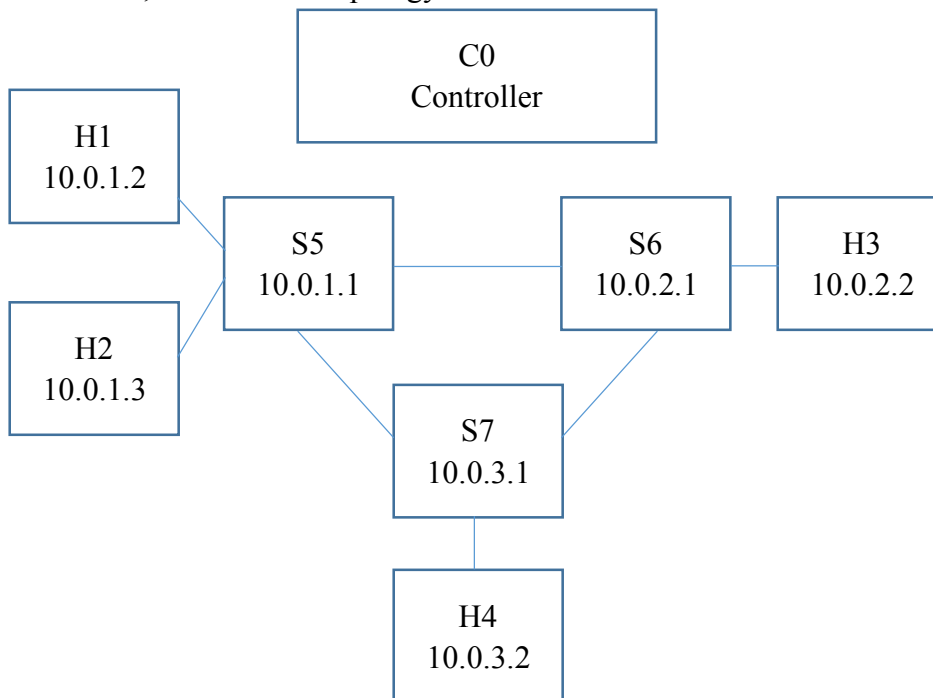
```

We can use “pingall” to show the result directly. Host 5 can’t connect to host 3.

### Part 3:

#### 1. Bonus

First of all, we build the topology which contains 3 routers and 4 hosts.



This part is very similar with the advance topology and we only need to do a little modify.

#### Testing Controller:

```
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s5 s6 s7
*** Adding links:
(h1, s5) (h2, s5) (h3, s6) (h4, s7) (s5, s6) (s5, s7) (s6, s7)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s5 s6 s7
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['20.5 Gbits/sec', '20.5 Gbits/sec']
mininet>
```

According to this figure, we can see that our topology can work successfully and has a 20.5 Gbits/sec bandwidth.

## Conclusion:

From this project, according to the simulation of SDN, we have a deeper understanding of the principle and mechanism of Software Defined Network. Inside the controller, from the detailed information of routing table, we need to react to the different information from the router. This kind of mechanism can reduce the load in the router and increase the efficiency of the routing mechanism. Plus, in the project, we know more about the ARP, ICMP messages. We know how these messages work and some details about the transmission. This experiment can help us to establish the solid basic knowledge about the SDN and benefit us a lot.