School of Electrical and Electronic of Enginnering
eee 4474-01 : Experiments on Communication Networks
2조 2016142096 **조윤신**, 2017142043 **김재민**

Result Report Week 06

mininet (1) : sdn-based switch and hub

Simulation of virtual network topology by using mininet & Pox controller

In week 06 experiment, we employ the use of software defined networks (SDN) with the pox controller in a virtualized network environment with mininet.

1. Introduction

The main topic of this week's experiment, SDN, and OpenFlow, is the new proposal to overcome the internet's current structural problem based on the end-to-end transfer service and protocol stack TCP / IP.

In the current internet architecture, the intelligence systems are located at the ends of the network, with the core containing minimal information. Since the core network nodes do not provide information about its operation, it implies that the users get troubled when the network does not work. The node does not contain any information about where the error occurred in. We figured out those IP protocols work in the 'Week 02 Experiment: TCP IP Protocols.

The other example of the simple and transparent core the exact reason is the significant overhead of manual configuration, debugging, and designing new applications. We also checked those in the last experiments, That the TCP's overhead contains the other extra offset data area about 1~2 times compared to the pure socket.

2. SDN & Openflow

To overcome the limitations metioned above, SDN(software defined) and Openflow were introduced.
In short those problems in traditional networks are that both the control and data planes are tightly integrated and implementes in the forwarding devices that comprise a network. In this respect SDN is a networking paradigm where the data and control planes are decoupled from one another[1]

The SDN control plane is implemented by the 'controller' and the data plane by 'switches'. The controller acts as the "Brain" of the network, and send commands("Rules") to the switches on how to handle traffic. Openflow has emerged as the de facto SDN standard and specifies how the controller and the switches communicate s well as the rules controllers install on switches.

In this experiment, we simulate the how SDN and its standard, openflows does work as setting the sample scenario with SDN based Hub and L2 learning switch. The mention og

3. Simulating in virtual-environment

Why virtual environment?

Since the experiment in the real network environment it is hard to setting the component we posed to observing their work in given scenario, also can hardly be evaluated. So we implement the virtual network environment by using the mininet & POX controller.

[1] One can think of the control plane as being the netwokrs 'Brain', i.e., it is responsible making all decisions, for example, how to forward data, while the data plane is what aactually moves the data.

Mininet & POX controller

Mininet is an emulator for deploying large networks on the limited resources of a simple single Computer or Virtual Machine. Mininet can simulates SDN networks, can run a controller for experiment. It allows emulating real network scenarios Couple of SDN controllers are inclued with in Mininet virtual topology. And POX controller used for the SDN network works.

4. Build the virtual network environment by using the mininet & POX controller

So we are going to use The Simulation scenario consists of a single OpenFlow Switches (s1) connected to three hosts (h1, h2, h3).

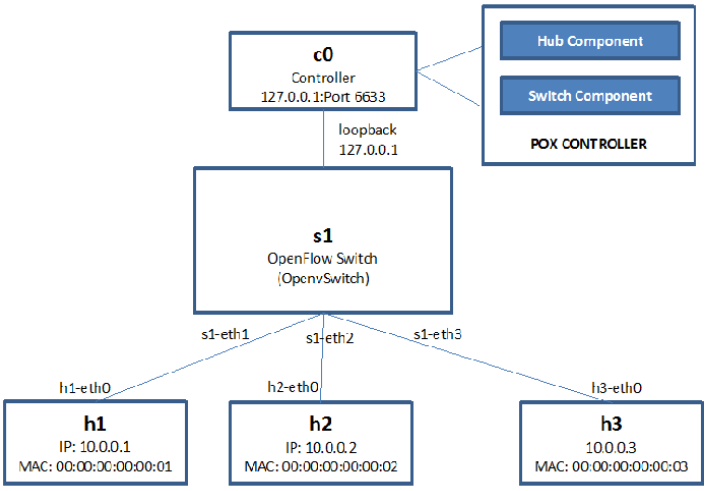


Figure 1: Simulation Scenario of topology used in week 06 experiment

There are two ways to implement the given scenario topology by mininet.
The one is the mentioned in the Experiment lecture material, that command the CLI in bash terminal in ubuntu environment, and the other is writing a custom python code since the mininet is builded by python API and we can easily make the scenario instead of writing a command line by line in bash terminal. We have implemented the basic topology of mininet to be used in Experiments 1, 2, and 3 in the two methods mentioned.

mininet CLI

In the mininet terminal it was used commands of code below to build a topology as shown in Figure 1 :

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

The parameter used in the code are described below (all of those a family of mininet CLI commands, implimented in "./mininet/topo/__main__.py"):

- **mn** : it starts the CLI mininet in bash terminal.
- **– topo single,3** : it creates a topology with 3 virtual hosts, each one containing a separated IP address.
- **– mac** : it sets the MAC address of each host equal to its IP. As a standard the hosts and switches begin at random MAC addresses. This can make it difficult to debug network applications, since each new execution Mininet generates MAC addresses distinct from their hosts and switches. To solve this problem we can use the option –mac where each node has a simple and easy address to read.
- **– switch ovsk** : it creates a single OpenFlow switch in software (OpenvSwitch) kernel with 3 doors.

- **– controller remote** : it sets the Openflow switch to connect to a remote contreller.

The figure below is the terminal output after we execute the mininet CLI code in bash terminal.

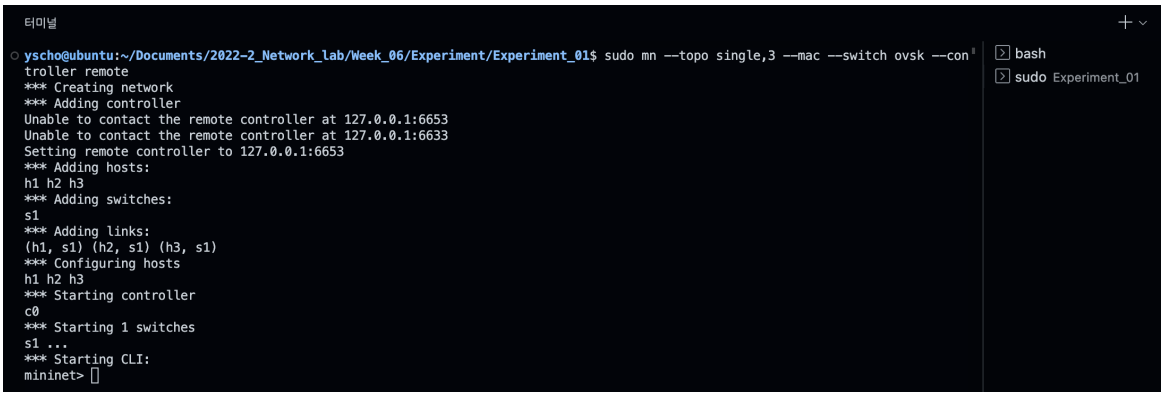


Figure 2: Terminal out screenshot : After executing the mininet CLI command in bash terminal

mininet python API

By writting complete mininet scripts in python, it can be esaily cusomized the mn command line tool using —custom options. That all options command in CLI like '—topo', defined as member of dictionary's keys. So as we can use the simple polomorphism of class that implemented in mininet API to build a custom scenario scripts. The python scripts below just same as the given mininet CLI command.

```python
#!/usr/bin/python
__author__ = "yunshin.cho"
from mininet.node import CPULimitedHost
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel, info
from mininet.node import RemoteController
from mininet.cli import CLI

class Simple3PktSwitch(Topo):
    def __init__(self, **opts):
        super(Simple3PktSwitch, self).__init__(**opts)
        # Add hosts and switches
        h1 = self.addHost('h1',mac='00:00:00:00:00:01',ip='10.0.0.1')
        h2 = self.addHost('h2',mac='00:00:00:00:00:02',ip='10.0.0.2')
        h3 = self.addHost('h3',mac='00:00:00:00:00:03',ip='10.0.0.3')
        opts = dict(protocols='OpenFlow13')
        # Adding switches
        # instead of use the --mac command in CLI, i set the ip address manually
        s1 = self.addSwitch('s1', ip = '127.0.0.1',port = '6633', opts=opts)
        # Add links
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s1)

    def run():
        c = RemoteController('c', '0.0.0.0', 6653)
        net = Mininet(topo=Simple3PktSwitch(), host=CPULimitedHost, controller = RemoteController)
        net.addController(c)
        net.start()
        CLI(net)
        net.stop()

# if the script is run directly (sudo custom/optical.py):
if __name__ == '__main__':
    setLogLevel('info')
    run()
```

mininet API python code 1: basic_topo.py, implement basic topology with 1 switch and 3 host

The figure below is the terminal output after we execute the custom python topology code in bash terminal.

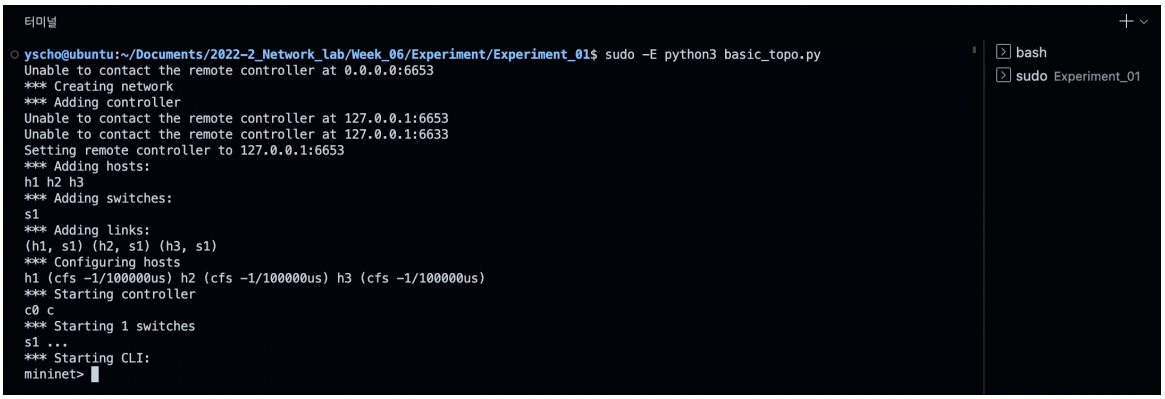


Figure 3: Terminal out screenshot : After executing the custom python topology code in bash terminal

5. Overview of Experiment

The original plan of this experiment was to take advantage of the benefits of mininet python script, which allows the progress of the entire scenario and the output to be written at once, as experiments 1, 2, and 3 use a topology using one switch and three hosts of the same structure.

However, in Experiments 2 and 3, the version of Python implemented by the POX controller used to simulate SDN behavior is 2.7 so an error occurred when importing the class from the script of the mininet written in an environment of 3.x, so we ran the POX controller on a separate terminal, and ran the topology using the minet CLI.

Overview of each experiment 1, 2, and 3

There are two methods to add flow entries into flow table of switch, remote controller and 'dpctl'utility.
As we build the simple topology with the option **'remote'** that is to be used to connect switch to remote controller. But when the remote controller is not running, hosts can not ping with each other[2] therefore flow table of switch does not contain any flow entry.

In **Experiment 1**, we build a topology with 1 swith with 3 hosts structure **without controller**. So we will therefore face the aforementioned problems. We are goint to use 'dpctl[3]'utility that comes with OpenFlow reference distribution and is used to manage the flow table of switch.
We will make a comparsion the case before we adds the flow between the host **'h1'** and host **'h2'**, and after adding flow by using the 'dpctl' commands.

In **Experiment 2 and 3** , we build the same topology of the experiment 1's with openflow based controller. The one is **HUB** and the other is **'L2 Switch'**.
We are going to have same ping test while the openflow controller is running. Then we check the dump flows before and after the ping test we observe the ping table that how does the each OpenFlow controller works. And also check the througput rate according to the number of hosts. To observe the each controller's mechanism.

[2]Because the openflow controller (here we set the remote controller to use POX controller) is responsible for deciding which action is to be performed by the switch.

[3]Data Path Controller

Experiment 1 : Adding Flow by Data Path Control (dpctl) command

1-1 Expeimtent Results

1-1-1 Simulation scenario python API code

We set the simulation scenario as followed by the given lecture material:

1. Setting the 1 host 3 switch topology as the basic topology in Figure 1
2. After implementing the whole topology then add flow between Host1 h1 and h2 by using 'dpctl' command, then check the pingtest and print out the flow table that doed the host 1 & host 2 were connected.

The whole scenario implementef by the python scripts below :

```python
#!/usr/bin/python
__author__ = "yunshin.cho"
from mininet.node import CPULimitedHost
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel, info
from mininet.node import RemoteController

class Simple3PktSwitch(Topo):
    def __init__(self, **opts):
        super(Simple3PktSwitch, self).__init__(**opts)
        # Add hosts and switches
        h1 = self.addHost('h1',mac='00:00:00:00:00:01',ip='10.0.0.1')
        h2 = self.addHost('h2',mac='00:00:00:00:00:02',ip='10.0.0.2')
        h3 = self.addHost('h3',mac='00:00:00:00:00:03',ip='10.0.0.3')
        opts = dict(protocols='OpenFlow13')
        # Adding switches
        # instead of use the --mac command in CLI, i set the ip address manually
        s1 = self.addSwitch('s1', ip = '127.0.0.1',port = '6633', opts=opts)
        # Add links
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s1)

def installStaticFlows(net):
    for sw in net.switches:                              # h1 h3 사이 flow 설정하기
        info('Adding flows to %s...' % sw.name)
        sw.dpctl('add-flow', 'in_port=1,nw_dst=10.0.0.3,actions=output=3')
        sw.dpctl('add-flow', 'in_port=3,nw_dst=10.0.0.1,actions=output=1')
        info(sw.dpctl('dump-flows'))                     # Flow table을 확인해 추가된 경로 확인

def run_scenario_expeiriment_01():
    '''
        Scenario : Compare the ping test before and after adding the flow between host 1 & 3
    '''
    c = RemoteController('c', '0.0.0.0', 6653)
    net = Mininet(topo=Simple3PktSwitch(), host=CPULimitedHost, controller = RemoteController)
    net.addController(c)
    net.start()
    h1, h2, h3, s1 = net.get('h1'), net.get('h2'), net.get('h3'), net.get('s1')
    s1.dpctl('dump-flows')
    print(h1.cmd('ping -c2',h3.IP())) # ping test between h1 and h3 with sending 2 pings
    net.pingAll()                     # Ping test between all hosts before adding the flow
    installStaticFlows( net )         # Adding flow between h1 and h3
    net.pingAll()                     # Ping test between all hosts after adding the flow
    print(h1.cmd('ping -c2',h3.IP())) # ping test between h1 and h3 with sending 2 pings
    print(h2.cmd('ping -c2',h3.IP())) # ping test between h2 and h3 with sending 2 pings
    net.end()

if __name__ == '__main__':
    setLogLevel('info')
    run_scenario_expeiriment_01()
```

mininet API python code 2: ex1.py, Experiment 01 simulation scenario python scripts

1-1-2 Terminal out screenshot

The figure below captures the output of the terminal after executing the Python script code.

By configuring the whole scenario in the python script, it is possible to check the operation of outputting the ping test and flow table before and after adding flow at once.

Figure 4: Terminal out screenshot : Simulation Scenario of Experiment 01

Experiment 2 : HUB controller

2-1 Flow Table

In Figure, the above terminal runs a POX controller on python2 and the forwarding of hub is activating. And the terminal below is the printed out flow-table. Flooding action is being performed through the flow table, so it can be confirmed that transmission is possible between all hosts.

Figure 5: Terminal out screenshot : Flow table with activating forwarding HUB

Discussion

HUB is the device of OSI layer 1. This is because the MAC address existing in layer 2 does not exist in layer 1, so data transmission and reception cannot be distinguished.

HUB's flooding function, also known as broadcasting, performs the function of sending evety input data from HOST to all three connected data entering the HUB regardless of the sender of the data. As a result, the HUB performs the flooding function, which is received by all hosts at the same time, resulting in unnecessary data loss and the number of data transmitted on the network is more than necessary.

Let's take an additional ping test with h1 → h3 and h2 → h3 then check the input and output of the packet for each host with the result of mini-interface listening.

2-2 Ping Test

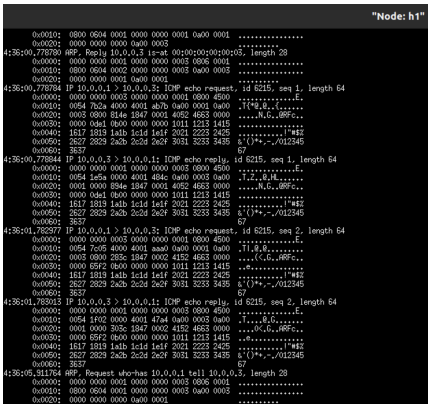
(a) Ping test (H1 → H3) with 2 packets

(b) Ping test (H2 → H3) with 2 packets

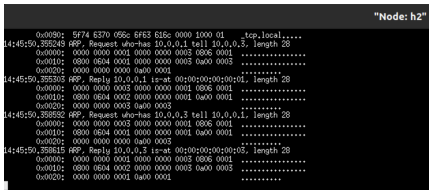
Figure 6: Terminal out screenshot : Ping Test with activating forwarding HUB

2-3 Interface Listening

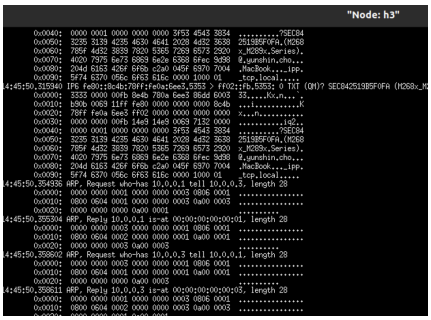
Because all packets are flooded, H1, H2, and H3 can check the broadcasting of the Hub that sends and receives ICMP and ARP packets.



(a) Screenshot of h1's terminal



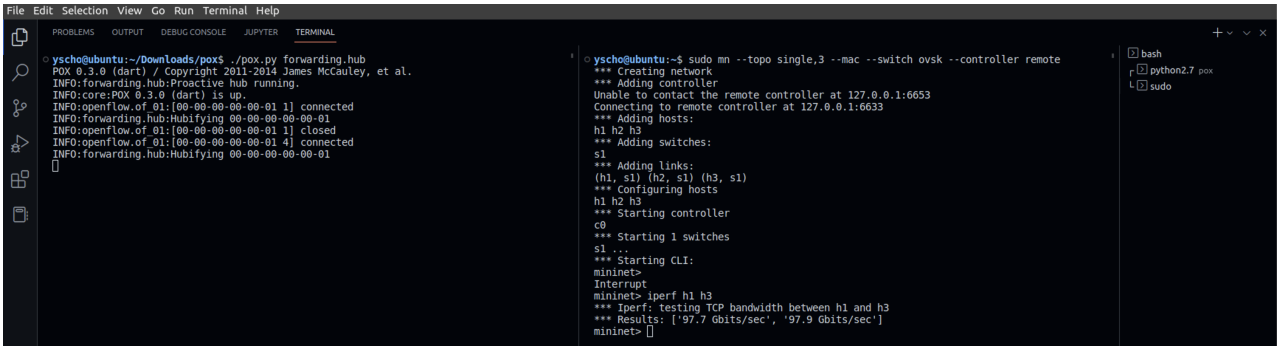
(b) Screenshot of h2's terminal



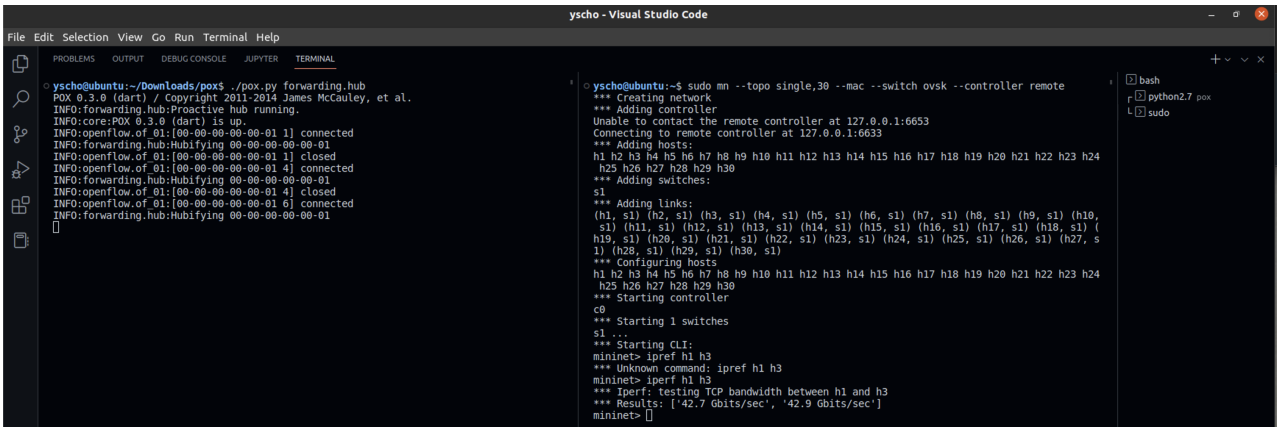
(c) Screenshot of h3's terminal

Figure 7: Terminal out screenshot : Each host's interaface terminal

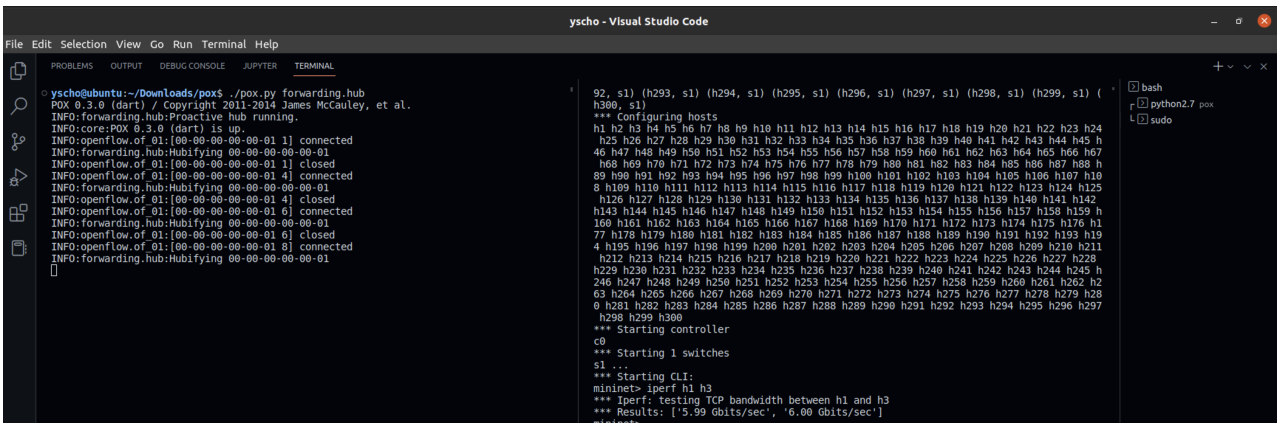
2-4 Throughput test



$N = 3$: 97.7, 97.9 Gbits / sec



$N = 30$: 42.7, 42.9 Gbits / sec



$N = 300$: 5.99, 3.0 Gbits / sec

Figure 8: Terminal out screenshot : The throughput rate of h3 correspods the number of hosts

Discussion

When HUB receives data, it transmits data to all connected at the same time, regardless of the sender and receiver, so it is usually set by dividing the bandwidth proportionally to the number of connected hosts from the total available. As a result of the experiment, since the total bandwidth is evenly divided by Host, it can be confirmed that the throughput decreases as the number of Hosts increases in proportion thereto.

Experiment 3 : L2 Switch Controller

In Experiment 3, we confirm the operation of the L2 switch. The L2 switch is an OSI two-layer MAC address-based network device that forwards frames to destinations to improve the defects of the HUB that cause bottlenecks.The operation of the L2 switch is as follows as example between h1 and h3:

1. Send packets from Host h1 to Host h3
2. **Learning-1** : Store the MAC address and port number of the origin h1 Host in the MAC table
3. **Flooding** : Destination h3 Host's MAC address is not in the MAC table, so it is sent to all ports
4. Send packets to Host h1 using MAC address received from flooding from Host h3
5. **Learning-2** : Store the origin MAC address and port number (Host h3) in the MAC Rable in process 4
6. **Forwarding** : Prior to the process 5, the MAC table has only the MAC address and port number of Host h1, the first sender, so only the stored Host h1 can be transferred by h3 (h3 → h1 forwarding)

In the following experiment, we are going to compare the flow table before and after transmission of the packet to check how the example between h1 and h3's process proceeds.

3-1 Flow table before sending the packet

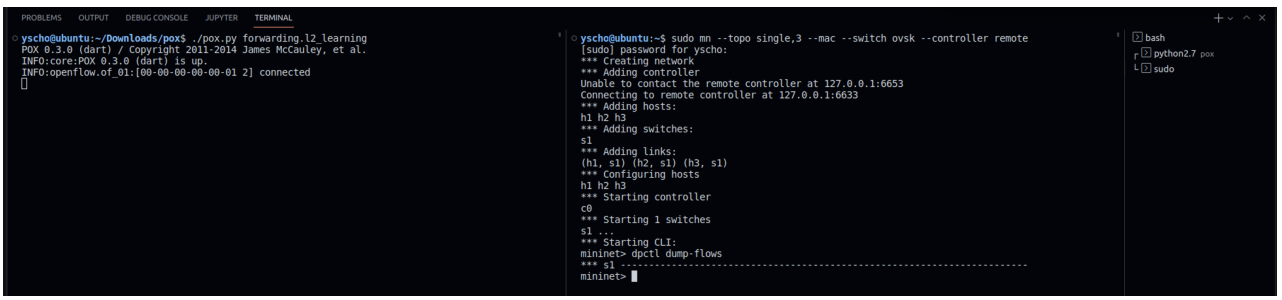


Figure 9: Terminal out screenshot : Flow table with activating forwarding L2 switch

3-2 Sending Packet : Ping test results (H1 → H3)

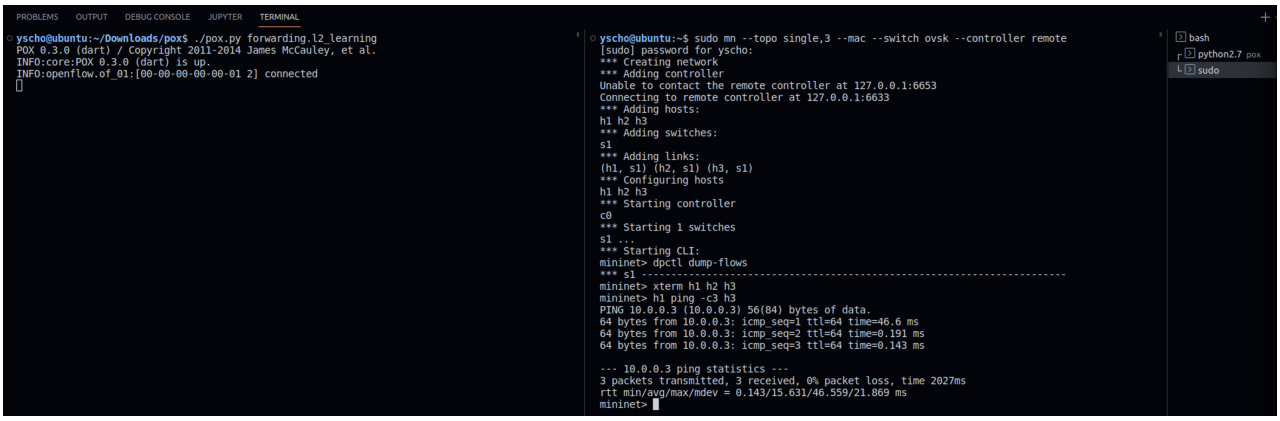


Figure 10: Terminal out screenshot : Sending the two packets (h1 → h3)

3-3 Flow table after sending the packet

In the figure we can find out the 2 of learing processes that updates the MAC Table adding the MAC addtrss of h1 and h3 in first four log line and then forwarding process that sends packet from h3 to h1 in last fifth log line.

Figure 11: Terminal out screenshot : Flow table with activating forwarding L2 Switch

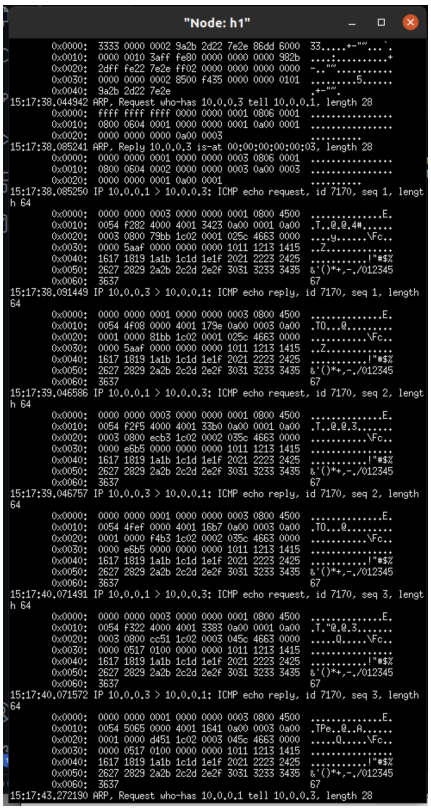
Discussion

In the previous example, when transmitting a packet from Host h1 to Host h3, the process of forwarding the L2 switch was checked based on the change in the MAC table.

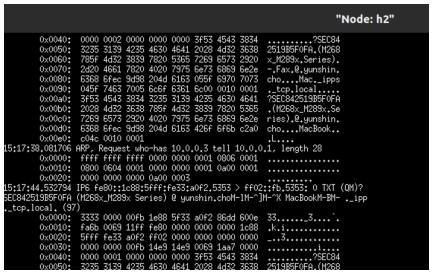
Through the learning process between h1 and h3 corresponding to stage 2 and 5, the results in which the forwarding corresponding to stage 6 is from h3 to h1 may be confirmed through the flow table after sending the packet.

3-4 Interface Listening

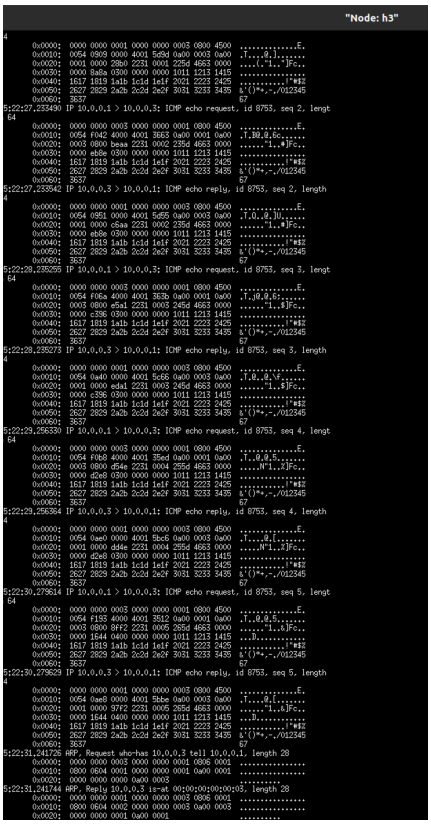
The detailed process of packets exchanged between h1 and h3 could be confirmed by interface listening experiment.
As the previous series of example shows, in the case of h2 the only once transmission of packets in flooding process that is to learn the address of h3 in h1 while they change the packet in forwding that of after learning process.



(a) Screenshot of h1's terminal



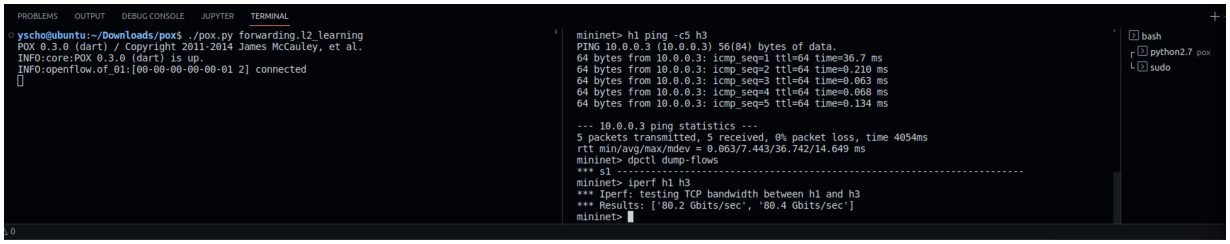
(b) Screenshot of h2's terminal



(c) Screenshot of h3's terminal

Figure 12: Terminal out screenshot : Each host's interaface terminal

3-5 Throughput test



$N = 3$: 80.2, 80.4 Gbits / sec

$N = 30$: 84.8, 85.0 Gbits / sec

$N = 300$: 73.6, 73.7 Gbits / sec

Discussion

Unlike Hubs, L2 switches do not force bandwidth allocation per host because the flooding action of transmitting throughout the connected host is performed only in the first case of adding an address to the MAC table.

However, the slight decrease in bandwidth as the number of hosts increases is considered to the possibility of collision in the first flooding step is proportional increasing to the number of hosts.