



# 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 mentioned 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 implemented 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<sup>1</sup>.

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 sends 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 as 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 of

### 3. Simulating in virtual-environment

#### Why virtual environment?

Since the experiment in the real network environment it is hard to set up 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.

<sup>1</sup>One can think of the control plane as being the network's 'Brain', i.e., it is responsible for making all decisions, for example, how to forward data, while the data plane is what actually 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).

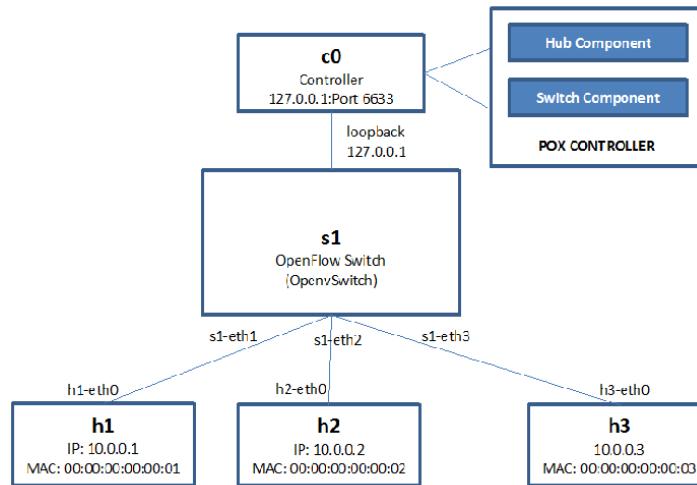


Figure 1: Simulation Scenario of topology used in week 06 experiment

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, implemented 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 controller.

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

```
터미널
○ ysho@ubuntu:~/Documents/2022-2_Network_lab/Week_06/Experiment/Experiment_01$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** 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> []
```

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

## mininet python API

By writing complete mininet scripts in python, it can be easily customized 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 polymorphism 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.

---

```
#!/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.

```

터미널
○ yscho@ubuntu:~/Documents/2022-2_Network_lab/Week_06/Experiment/Experiment_01$ sudo -E python3 basic_topo.py
Unable to contact the remote controller at 0.0.0.0:6653
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 (cfs -1/100000us) h2 (cfs -1/100000us) h3 (cfs -1/100000us)
*** Starting controller
c0 c
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> 

```

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<sup>2</sup> therefore flow table of switch does not contain any flow entry.

In **Experiment 1**, we build a topology with 1 switch with 3 hosts structure **without controller**. So we will therefore face the aforementioned problems. We are going to use 'dpctl'<sup>3</sup> utility that comes with OpenFlow reference distribution and is used to manage the flow table of switch.

We will make a comparison the case before we add 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 throughput rate according to the number of hosts. To observe the each controller's mechanism.

<sup>2</sup>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.

<sup>3</sup>Data Path Controller

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

## 1-1 Expeiment 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 :

---

```

#!/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')      # Flow table을 확인해 추가된 경로 확인
            info(sw.dpctl('dump-flows'))

    def run_scenario_expeirment_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_expeirment_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.

```

터미널 + 
bash
bash Experiment_01

● yscho@ubuntu:~/Documents/2022-2_Network_lab/Week_06/Experiment/Experiment_01$ sudo -E python3 ex1.py
Unable to contact the remote controller at 0.0.0.0:6653
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6653
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 (cfs -1/100000us) h2 (cfs -1/100000us) h3 (cfs -1/100000us)
*** Starting controller
c0 c
*** Starting 1 switches
s1 ...
----- Flow를 추가하기 전 Flow Table 출력

----- Flow를 추가하기 전 h1과 h3 사이의 ping 2개 test
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1027ms
pipe 2

----- Flow를 추가하기 전 host 전체의 Ping test
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
Adding flows to s1...----- Flow를 추가한 후 Flow Table 출력
cookie=0x0, duration=0.009s, table=0, n_packets=0, n_bytes=0, in_port="s1-eth1" actions=output:"s1-eth3"
cookie=0x0, duration=0.004s, table=0, n_packets=0, n_bytes=0, in_port="s1-eth3" actions=output:"s1-eth1"

----- Flow를 추가한 후 h1과 h3 사이의 ping 2개 test
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.445 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.061 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.061/0.253/0.445/0.192 ms

----- Flow를 추가한 후 h2과 h3 사이의 ping 2개 test
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1022ms
pipe 2

----- Flow를 추가한 후 host 전체의 Ping test
*** Ping: testing ping reachability
h1 -> X h3
h2 -> X X
h3 -> h1 X
*** Results: 66% dropped (2/6 received)
*** Stopping 2 controllers
c0 c
*** Stopping 3 links
...
*** Stopping 1 switches
s1
*** Stopping 3 hosts
h1 h2 h3
*** Done
○ yscho@ubuntu:~/Documents/2022-2_Network_lab/Week_06/Experiment/Experiment_01$ 

```

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.

```

    터미널
    yscho@ubuntu:~$ ./pox/pox.py forwarding.hub
    bash: ./pox/pox.py: No such file or directory
    ○ yscho@ubuntu:~$ ./Downloads/pox/pox.py forwarding.hub
    POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
    INFO:forwarding.hub:Proactive hub running.
    INFO:core:POX 0.3.0 (dart) is up.
    INFO:openflow.of_01:[00:00:00:00:00:01 2] connected
    INFO:forwarding.hub:Hubifying 00:00:00:00:00:01
    ┌─┐

    ○ yscho@ubuntu:~$ sh Documents/2022-2_Network_lab/Week_06/Experiment/Experiment_01/experiment01.sh
    [sudo] password for yscho:
    *** Creating network
    *** Adding controller
    Unable to contact the remote controller at 127.0.0.1:6653
    Connecting to remote controller at 127.0.0.1:6633
    *** 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> dpctl dump-flows
    *** s1 -----
    cookie=0x0, duration=84.424s, table=0, n_packets=24, n_bytes=1896, actions=FL00D
    mininet> ┌─┐
  
```

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 every 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  $h1 \rightarrow h3$  and  $h2 \rightarrow h3$  then check the input and output of the packet for each host with the result of mini-interface listening.

### 2-2 Ping Test

```

mininet> h1 ping -c2 h3
ping: -c2: Temporary failure in name resolution
mininet> h1 ping -c2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.339 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.103 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.103/0.221/0.339/0.118 ms
  
```

(a) Ping test ( $H1 \rightarrow H3$ ) with 2 packets

```

mininet> h2 ping -c2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.00 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.227 ms

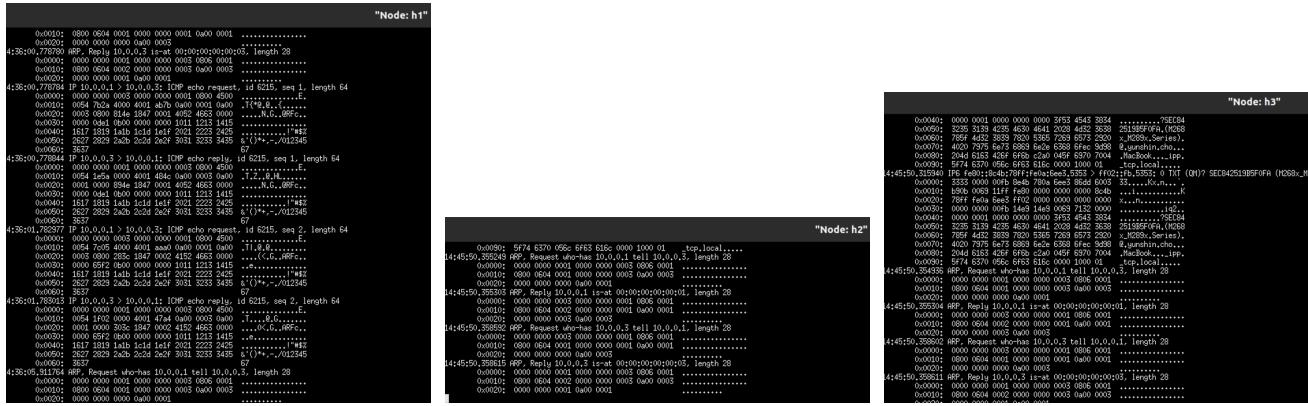
--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.227/0.615/1.003/0.388 ms
  
```

(b) Ping test ( $H2 \rightarrow H3$ ) 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 interface terminal

## 2-4 Throughput test

```
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL
+ x
bash
python2.7 pox
sudo

yosho@ubuntu:~/Downloads/pox$ ./pox.py forwarding.hub
POX 0.3.0 (dart) Copyright 2011-2014 James McCauley, et al.
INFO:forwarding.hub:Forwarding hub is running.
INFO:core.POX 0.3.0 (dart) is up
INFO:openflow.of_01:[00-00-00-00-00-01] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-01] closed
INFO:openflow.of_01:[00-00-00-00-00-01] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
[

yosho@ubuntu:~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6653
*** 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
Internet
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: [ '97.7 Gbits/sec', '97.9 Gbits/sec' ]
mininet> [
```

$N = 3 : 97.7, 97.9 \text{ Gbits / sec}$

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

$N = 300 : 5.99, 3.0 \text{ Gbits / sec}$

D•

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 Table 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 ( $h_3 \rightarrow h_1$  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

```
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL + ×

yosho@ubuntu:~/Downloads/pox$ ./pox.py forwarding.l2_learning
POX 0.3.0 (dарт) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dарт) is up.
INFO:openflow.of_01:[60:00:00:00:00:01 2] connected
[]

yosho@ubuntu:~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
[sudo] password for yosho:
*** Creating network...
*** Adding controller
Unreachable to contact the remote controller at 127.0.0.1:6633
Connecting to remote controller at 127.0.0.1:6633
*** 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> dptctl dump-flows
*** s1 -----
mininet> ■
```

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

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

```

PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL
ysho@ubuntu:~/Downloads/pox$ ./pox.py forwarding.l2_learning
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
[...]
ysho@ubuntu:~/Downloads/pox$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
[sudo] password for ysho:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6653
*** 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> dptcl dump-flows
*** s1 .....
mininet> xterm h1 h2 h3
mininet> h1 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=46.6 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.191 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.143 ms
...
-- 10.0.0.3 ping statistics --
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.143/15.631/46.559/21.869 ms
mininet>

```

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 learning processes that updates the MAC Table adding the MAC address 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.

```

PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL
ysho@ubuntu:~/Downloads/pox$ ./pox.py forwarding.l2_learning
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 4] closed
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:openflow.of_01:[00-00-00-00-00-01 6] connected
INFO:openflow.of_01:[00-00-00-00-00-01 6] closed
INFO:openflow.of_01:[00-00-00-00-00-01 8] connected
[...]
mininet> dptcl dump-flows
*** s1 .....
cookie=0x0, duration=8.356s, table=0, n_packets=1, n_bytes=42, idle_timeout=10, hard_timeout=30, priority=65535, arp,in_port="s1-eth3",vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,arp.spa=10.0.0.3,arp.tpa=10.0.0.1,arp.op=2 actions=output:"s1-eth1"
cookie=0x0, duration=3.177s, table=0, n_packets=1, n_bytes=42, idle_timeout=10, hard_timeout=30, priority=65535, arp,in_port="s1-eth3",vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,arp.spa=10.0.0.3,arp.tpa=10.0.0.1,arp.op=1 actions=output:"s1-eth1"
cookie=0x0, duration=8.354s, table=0, n_packets=2, n_bytes=196, idle_timeout=10, hard_timeout=30, priority=65535, icmp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw.tos=0,icmp.type=8,icmp.code=0 actions=output:"s1-eth1"
cookie=0x0, duration=8.351s, table=0, n_packets=2, n_bytes=196, idle_timeout=10, hard_timeout=30, priority=65535, icmp,in_port="s1-eth3",vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw.dst=10.0.0.1,nw.tos=0,icmp.type=8,icmp.code=0 actions=output:"s1-eth1"
mininet>
mininet>

```

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 forward that of after learning process.

"Node: h1"

```

0x0000: 3333 0000 0002 9a20 2d22 7e2e 86dd 6000 33.....E
0x0010: 0000 0000 0001 3af9 f809 0000 0000 0000 982b ..-.....
0x0020: 2aef Fe22 7e2e F000 0000 0000 0000 0000 ..-.....
0x0030: 0000 0000 0000 0000 0000 0000 0000 0001 ..-.....
0x0040: 9a20 2d22 7e2e ..-.....
15:17:39.044942 88P, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
0x0000: ffff FFFF FF8F 0000 0000 0001 0006 0001 ..-.....
0x0010: 0000 0004 0001 0000 0001 0000 0001 0000 ..-.....
0x0020: 0000 0000 0000 0000 0000 0000 0000 0000 ..-.....
0x0030: 0000 0000 0001 0000 0000 0003 0006 0001 ..-.....
0x0040: 0000 0004 0002 0000 0000 0003 0000 0003 ..-.....
0x0050: 0000 0000 0000 0000 0000 0000 0000 0000 ..-.....
0x0060: 3637 ..-.....
15:17:39.065250 IP 10.0.1.1 > 10.0.0.3: ICMP echo request, id 7170, seq. 1, leng
h 64
0x0000: 0000 0000 0003 0000 0000 0001 0000 4500 ..-.....
0x0010: 0004 F20 4000 4001 332d 0000 0001 0000 ..T,..0.8...
0x0020: 0003 0000 73b5 1c00 0001 025c 4663 0000 ..-.....
0x0030: 0000 5a00 0000 0000 0000 0000 1011 1213 1415 ..-.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2011 2223 2425 ..!#*...
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 ..!(*,*,-/012345
0x0060: 3637 ..-.....
15:17:39.091449 IP 10.0.0.3 > 10.0.1: ICMP echo reply, id 7170, seq. 1, leng
h 64
0x0000: 0000 0000 0001 0000 0000 0003 0000 4500 ..-.....
0x0010: 0004 4f00 0001 4001 179c 0000 0003 0000 ..T,..0.8...
0x0020: 0001 0000 81bb 1c00 0001 025c 4663 0000 ..-.....
0x0030: 0000 5a00 0000 0000 0000 0000 1011 1213 1415 ..-.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2011 2223 2425 ..!#*...
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 ..!(*,*,-/012345
0x0060: 3637 ..-.....
15:17:39.094859 IP 10.0.0.1 > 10.0.3: ICMP echo request, id 7170, seq. 2, leng
h 64
0x0000: 0000 0000 0003 0000 0000 0001 0000 4500 ..-.....
0x0010: 0004 F25 4000 4001 332d 0000 0001 0000 ..T,..0.8...
0x0020: 0003 0000 ecb3 1c02 0002 03cc 4663 0000 ..-.....
0x0030: 0000 e65b 0000 0000 0000 0000 1011 1213 1415 ..-.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2011 2223 2425 ..!#*...
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 ..!(*,*,-/012345
0x0060: 3637 ..-.....
15:17:39.094675 IP 10.0.0.3 > 10.0.1: ICMP echo reply, id 7170, seq. 2, leng
h 64
0x0000: 0000 0000 0004 0001 0000 0003 0000 4500 ..-.....
0x0010: 0004 4f4f 0001 4001 1687 0000 0003 0000 ..T,..0.8...
0x0020: 0001 0000 f4f3 1c02 0002 03cc 4663 0000 ..-.....
0x0030: 0000 e65b 0000 0000 0000 0000 1011 1213 1415 ..-.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2011 2223 2425 ..!#*...
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 ..!(*,*,-/012345
0x0060: 3637 ..-.....
15:17:40.017451 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 7170, seq. 3, leng
h 64
0x0000: 0000 0000 0003 0000 0000 0001 0000 4500 ..-.....
0x0010: 0004 F322 4000 4001 333b 0000 0001 0000 ..T,..0.5...
0x0020: 0003 0000 9451 1c02 0002 04cc 4663 0000 ..-.....
0x0030: 0000 e65b 0100 0000 0000 0000 1011 1213 1415 ..-.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2011 2223 2425 ..!#*...
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 ..!(*,*,-/012345
0x0060: 3637 ..-.....
15:17:40.017572 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 7170, seq. 3, leng
h 64
0x0000: 0000 0000 0001 0000 0000 0000 0000 4500 ..-.....
0x0010: 0004 5065 0001 4001 1641 0000 0003 0000 ..T,Pe...A...
0x0020: 0003 0000 9451 1c02 0002 04cc 4663 0000 ..-.....
0x0030: 0000 e65b 0100 0000 0000 0000 1011 1213 1415 ..-.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2011 2223 2425 ..!#*...
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 ..!(*,*,-/012345
0x0060: 3637 ..-.....
15:17:45.272190 88P, Request who-has 10.0.0.1 tell 10.0.3, length 28

```

(a) Screenshot of h1's terminal

(b) Screenshot of h2's terminal

"Node: h3"  
 4 :00000 :0000 0001 0000 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 0809 0000 4001 5843 0000 0000 0000 4500 .....I.,.1..,F.  
 :00020 :0001 0000 2801 2231 0000 2554 4683 0000 .....N..1..,F.C.  
 :00030 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:22:27,235450 IP 10.0.0.1 > 10.0.0.1; ICMP echo request, id 8755, seq. 2, length 4  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 F042 4000 4001 3863 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 0000 0000 0000 0000 0000 1011 1213 1415 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:22:27,235451 IP 10.0.0.3 > 10.0.0.1; ICMP echo reply, id 8755, seq. 2, length 4  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 0954 0000 4001 5565 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 0000 0000 0000 0000 0000 1011 1213 1415 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:22:28,235217 IP 10.0.0.1 > 10.0.0.1; ICMP echo request, id 8755, seq. 3, length 4  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 0440 0000 4001 5565 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 ed2231 0005 2454 4683 0000 .....N..1..,F.C.  
 :00030 :0000 0000 0000 0000 0000 0000 1011 1213 1415 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:22:28,235218 IP 10.0.0.3 > 10.0.0.1; ICMP echo reply, id 8755, seq. 3, length 4  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 F088 4000 4001 3565 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 d28 0300 0000 0000 1011 1213 1415 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:22:29,235364 IP 10.0.0.1 > 10.0.0.1; ICMP echo request, id 8755, seq. 4, length 4  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 F088 4000 4001 3565 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 d28 0300 0000 0000 1011 1213 1415 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:22:29,235365 IP 10.0.0.1 > 10.0.0.1; ICMP echo request, id 8755, seq. 5, length 4  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 F133 4000 4001 3520 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 0000 0000 0000 0000 0000 1011 1213 1415 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:22:29,279623 IP 10.0.0.3 > 10.0.0.1; ICMP echo request, id 8755, seq. 5, length 4  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 0954 0000 4001 Bbb 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 1644 0400 0000 0000 1011 1213 1415 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:23:13,241725 IP 10.0.0.1 > 10.0.0.1; ICMP echo request, id 8755, seq. 6, length 28  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 0954 0000 4001 0000 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.  
 E:23:13,241726 IP 10.0.0.3 > 10.0.0.1; ICMP echo reply, id 8755, seq. 6, length 28  
 :00000 :0000 0000 0001 0000 0000 0000 0000 0000 4500 .....E.  
 :00001 :0054 0954 0000 4001 Bbb 0000 0000 0000 4500 .....T,,0..,J..,F.  
 :00020 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00030 :0000 0000 0000 0000 0000 0000 0000 0000 4500 .....N..1..,F.C.  
 :00040 :1617 1819 1ab1 lcld 1lef 2013 2232 4245 .....N..1..,F.C.  
 :00050 :3637 2829 2ab2 2cd2 2d29 3013 3233 3435 L(1\*\*,-,0/2345  
 :00060 :3637 .....E.

(c) Screenshot of h3's terminal

### 3-5 Throughput test

```
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL +  
y szko@ubuntu:~/Downloads/pox$ ./pox.py forwarding.l2_learning  
POX 0.3.0 (dart) Copyright 2011-2014 James McCauley, et al.  
INFO:core:POX 0.3.0 (dart) is up  
INFO:openflow.of_01:[00:00:00:00:00:01 2] connected  
[ ]  
  
mininet> h1 ping -c 5 h3  
PING 8.0.0.3 (8.0.0.3) 56(84) bytes of data.  
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=36.7 ms  
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.210 ms  
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.063 ms  
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.068 ms  
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.134 ms  
  
... 10.0.0.3 ping statistics ...  
5 packets transmitted, 5 received, 0% packet loss, time 4054ms  
rtt min/avg/max/mdev = 0.0637/7.443/36.742/14.649 ms  
mininet> dptcl dump-flows  
*** s1 -----  
mininet> dptcl -T h3  
*** iperf -test ip bandwidth between h1 and h3  
*** Results: ['100.2 Gbits/sec', '88.4 Gbits/sec']  
mininet> [ ]
```

$N = 3 : 80.2, 80.4 \text{ Gbits / sec}$

```
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL +  
yo@ubuntu:~/Downloads/pox$ ./pox.py forwarding,l2_learning  
POX 0.3.0 (dарт) / Copyright 2011-2014 James McCauley, et al.  
INFO:core:POX 0.3.0 (dарт) is up.  
INFO:openflow.of_01:[00:00:00:00:00:01] connected  
INFO:openflow.of_01:[00:00:00:00:00:01] closed  
INFO:openflow.of_01:[00:00:00:00:00:01] connected  
[]  
+ *** Adding links:  
  (h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1) (h9, s1) (h10, s1) (h11, s1) (h12, s1) (h13, s1) (h14, s1) (h15, s1) (h16, s1) (h17, s1) (h18, s1) (h19, s1) (h20, s1) (h21, s1) (h22, s1) (h23, s1) (h24, s1) (h25, s1) (h26, s1) (h27, s1) (h28, s1) (h29, s1)  
+ *** Configuring hosts  
  h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30  
+ *** Starting controller  
c9  
+ *** Starting 1 switches  
s1 ...  
+ *** Starting CLI:  
mininet> ./mininet-h3  
+ iperf: testing TCP bandwidth between h1 and h3  
+ Results: ['84.8 gbits/sec', '85.0 Gbits/sec']  
mininet> 
```

$N = 30 : 84.8, 85.0 \text{ Gbits / sec}$

```
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL +  
yesho@ubuntu:~/Downloads/pox$ ./pox.py forwarding.l2_learning  
POX 0.3.0 (dарт) / Copyright 2011-2014 James McCauley, et al.  
INFO:POX:POX 0.3.0 (dарт) is up.  
INFO:openflow.of_01:[00:00-00-00-00-01 2] connected  
INFO:openflow.of_01:[00:00-00-00-00-01 2] closed  
INFO:openflow.of_01:[00:00-00-00-00-01 2] connected  
INFO:openflow.of_01:[00:00-00-00-00-01 4] closed  
INFO:openflow.of_01:[00:00-00-00-00-01 6] connected  
[ ]  
h153 h154 h155 h156 h157 h158 h159 h160 h161 h162 h163 h164 h165 h166 h167 h168 h169 h170 h171 h172 h173 h174 h175 h176 h177 h178 h179 h180 h181 h182 h183 h184 h185 h186 h187 h188 h189 h190 h191 h192 h193 h194 h195 h196 h197 h198 h199 h200 h201 h202 h203 h204 h205 h206 h207 h208 h209 h210 h211 h212 h213 h214 h215 h216 h217 h218 h219 h220 h221 h222 h223 h224 h225 h226 h227 h228 h229 h230 h231 h232 h233 h234 h235 h236 h237 h238 h239 h240 h241 h242 h243 h244 h245 h246 h247 h248 h249 h250 h251 h252 h253 h254 h255 h256 h257 h258 h259 h260 h261 h262 h263 h264 h265 h266 h267 h268 h269 h270 h271 h272 h273 h274 h275 h276 h277 h278 h279 h280 h281 h282 h283 h284 h285 h286 h287 h288 h289 h290 h291 h292 h293 h294 h295 h296 h297 h298 h299 h300  
*** Starting controller  
c0  
*** Starting 1 switches  
s1  
*** Starting CLI:  
mininet> iperf h1 h3  
*** Iperf: testing TCP bandwidth between h1 and h3  
*** Results: [73.6 Gbits/sec, .73.7 Gbits/sec]
```

$N = 300 : 73.6 \text{ Gbits/sec}$

**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.