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 SDN networks using the 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 the week06 experiment, SDN and openflow , are the new proposal to overcome the nowdays structual problem of internet that it is based on the transfer service end-to-end and protocol stack TCP / IP.

So that current architerture of internet, the intelligence systems are located in the ends of the network with core contains very simple information as we can figured out those IP protocols work in **week 01, 02 experiment** with wireshark.The nodes of the core network do not provide information about its operation, and it implies that the user get troubled when the network does not work the node does not contain the any information where the error occured in.

The other example of the simple and transparent core with the same reason is a large overhead of manual configuration, debugging, and design new applications. We also checked those in the last **week 02 experiment**, the TCP protocol was used to perform the aforementioned functions. That the TCP's overhead contains the extra offset data area about amount of $1 \sim 2$ times compared to the pure socket has.

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.

The objectives of this week, we learn the how SDN and its standard, openflows does work as setting the sample scenario with SDN based Hub and L2 learning switch.

3. Experiment 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, 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.

POX controller & Mininet

In this network simulation expeiment employing the use of SDN with POX controllers in a virtualized environment with Minitnet in which the results will be evaluated.

4. Simulation of SDN networks using the POX controller

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

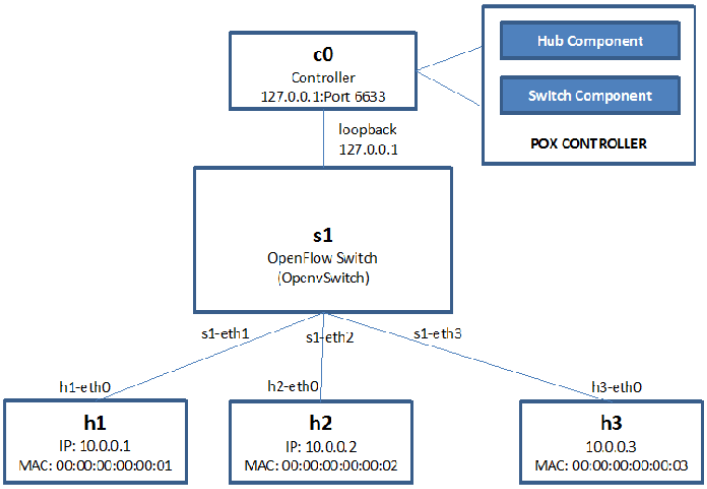


Figure 1: Simulation Scenario of topology used in week 06 experiment

And there are two ways to implemnt 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.

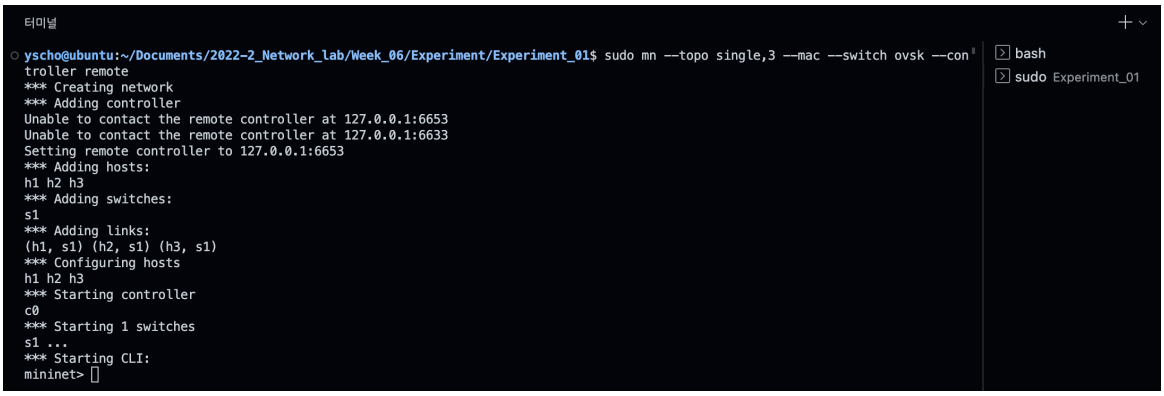


Figure 2: Terminal out screenshot : After executing the mininet CLI command in bash terminal

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.

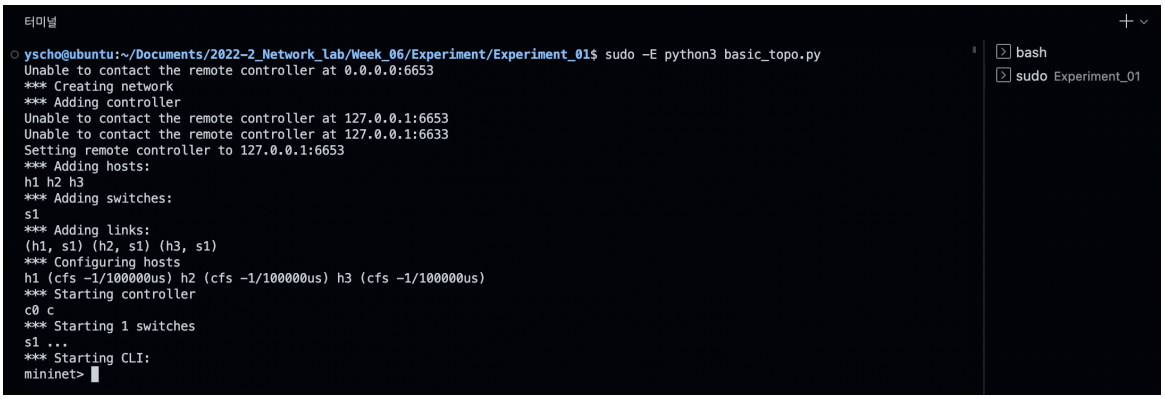


Figure 3: Terminal out screenshot : After executing the custom python topology code in bash terminal

4. Overview of Experiment 1, 2, 3

In **Experiment 1**, we build a topology with

Experiment 1 : Adding Flow in Flow Table

1-1 Expeimtent Results

1-1-1 Simulation scenario python API code

```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()
```

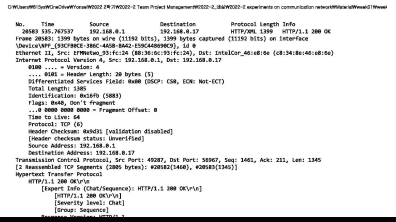
mininet API python code 2: ex1.py, Experiment 01 simulation scenario python scripts

1-1-2 Terminal out screenshot

Figure 4: Simulation Scenario of topology used in week 06 experiment

Experiment 2 : Adding Flow in Flow Table

2-1 Overview



blabla

2-2 Experiment Results

2-2-1 Flow Table

Figure 5: Simulation Scenario of topology used in week 06 experiment

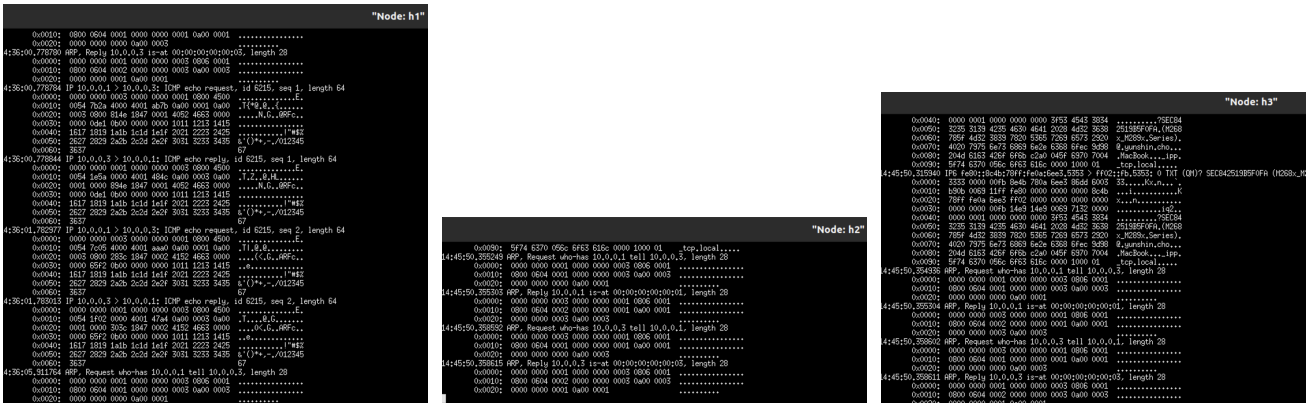
2-2-2 Ping Test

(a)

(b)

Figure 6

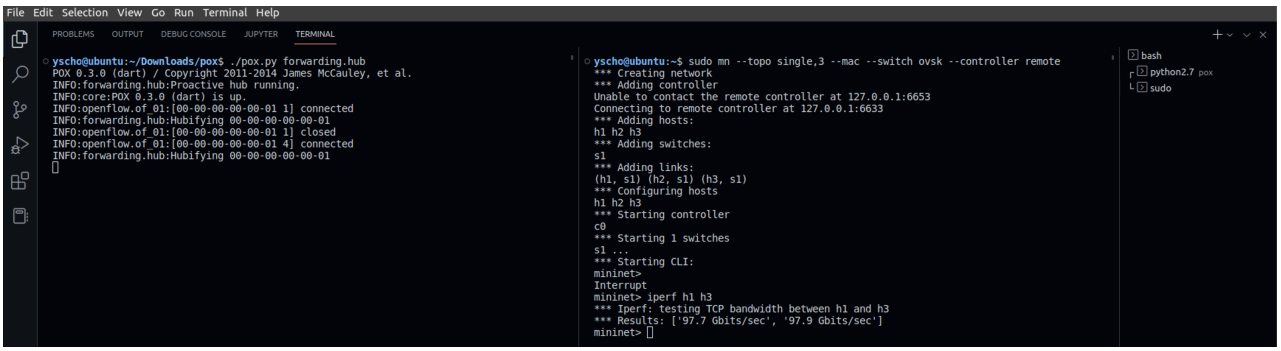
2-2-3 Interface Listeing



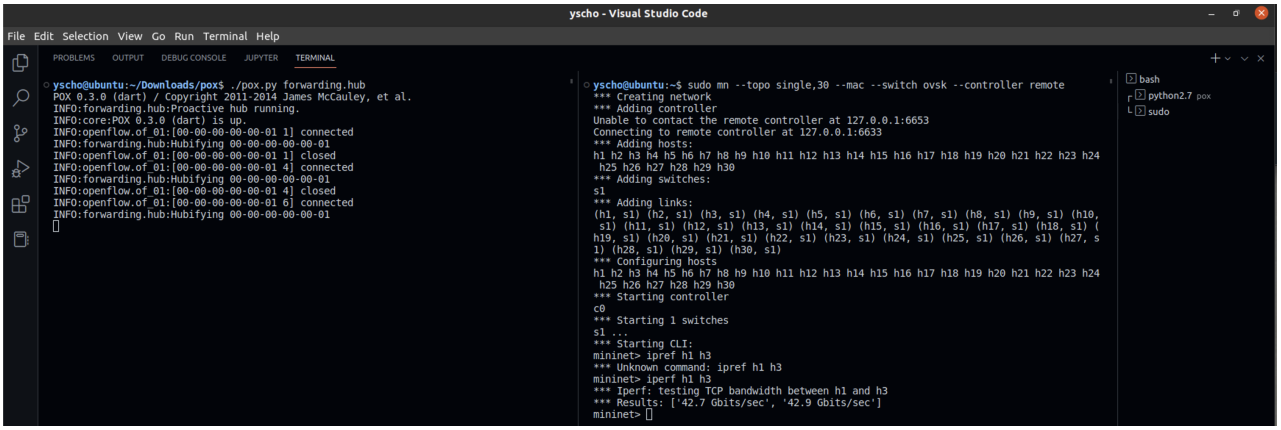
(a) Screenshot of h1's terminal (b) Screenshot of h2's terminal (c) Screenshot of h3's terminal

Figure 7

2-2-4 Bandwidth between network hosts


File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL
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 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
yscho@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 127.0.0.1:6633
*** 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>
Interrupt
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['97.7 Gbits/sec', '97.9 Gbits/sec']
mininet>
bash
python2.7 pox
sudo

(a)


yscho - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL
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 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-01 4] closed
INFO:openflow.of_01:[00-00-00-00-00-01 6] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
yscho@ubuntu:~$ sudo mn --topo single,30 --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 127.0.0.1:6633
*** Adding 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
*** Adding switches:
s1
*** 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, s
1) (h28, s1) (h29, s1) (h30, 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
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> ipref h1 h3
*** Unknown command: ipref h1 h3
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['42.7 Gbits/sec', '42.9 Gbits/sec']
mininet>
bash
python2.7 pox
sudo

(b)


yscho - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL
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 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-01 4] closed
INFO:openflow.of_01:[00-00-00-00-00-01 6] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-01 6] closed
INFO:openflow.of_01:[00-00-00-00-00-01 8] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
92, s1) (h293, s1) (h294, s1) (h295, s1) (h296, s1) (h297, s1) (h298, s1) (h299, s1) (
h300, 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 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h
46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h65 h66 h67
h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81 h82 h83 h84 h85 h86 h87 h88 h
89 h90 h91 h92 h93 h94 h95 h96 h97 h98 h99 h100 h101 h102 h103 h104 h105 h106 h107 h10
8 h109 h110 h111 h112 h113 h114 h115 h116 h117 h118 h119 h120 h121 h122 h123 h124 h125
h126 h127 h128 h129 h130 h131 h132 h133 h134 h135 h136 h137 h138 h139 h140 h141 h142
h143 h144 h145 h146 h147 h148 h149 h150 h151 h152 h153 h154 h155 h156 h157 h158 h159 h
160 h161 h162 h163 h164 h165 h166 h167 h168 h169 h170 h171 h172 h173 h174 h175 h176 h1
77 h178 h179 h180 h181 h182 h183 h184 h185 h186 h187 h188 h189 h190 h191 h192 h193 h19
4 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 h
246 h247 h248 h249 h250 h251 h252 h253 h254 h255 h256 h257 h258 h259 h260 h261 h262 h2
63 h264 h265 h266 h267 h268 h269 h270 h271 h272 h273 h274 h275 h276 h277 h278 h279 h28
0 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: ['5.99 Gbits/sec', '6.00 Gbits/sec']
bash
python2.7 pox
sudo

(c)

Figure 8

Experiment 3 : Adding Flow in Flow Table

3-1 Overview

3-2 Result - Screenshots

3-2-1 Before adding flows to switch

3-2-2 After addubg flows to switch