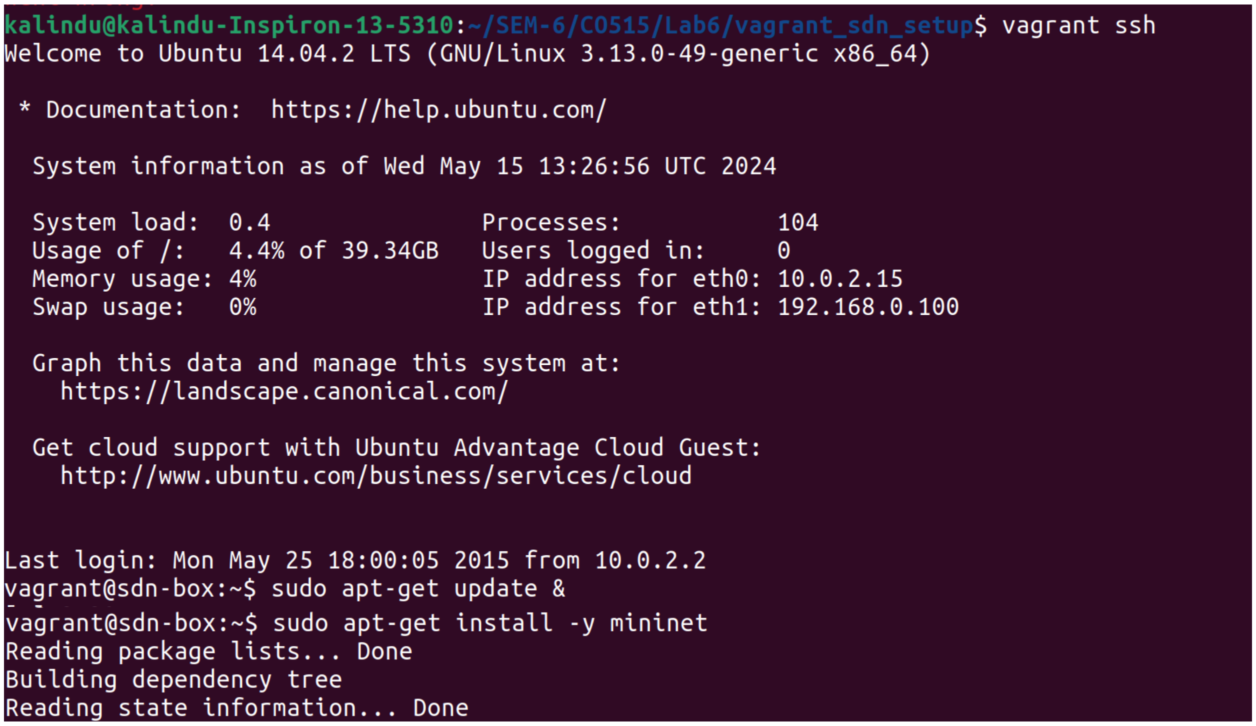
# CO515: Advances in Computer Networks: Selected Topics – Lab06 (Mininet Lab 03)

E/19/446: Wijerathna I.M.K.D.I.

09/05/2024

## Understanding Packet Switching in Mininet OpenFlow

### Install Mininet



```
kalindu@kalindu-Inspiron-13-5310:~/SEM-6/CO515/Lab6/vagrant_sdn_setup$ vagrant ssh
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-49-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Wed May 15 13:26:56 UTC 2024

  System load:  0.4              Processes:           104
  Usage of /:   4.4% of 39.34GB  Users logged in:     0
  Memory usage: 4%               IP address for eth0: 10.0.2.15
  Swap usage:   0%               IP address for eth1: 192.168.0.100

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud


Last login: Mon May 25 18:00:05 2015 from 10.0.2.2
vagrant@sdn-box:~$ sudo apt-get update &
vagrant@sdn-box:~$ sudo apt-get install -y mininet
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

*Figure 1: Update System and Ensure Installation of Mininet*

### Lab Environment Setup

*I used the same setup as the previous lab(Mininet Lab1) which is a lightweight ubuntu os setup which runs over vagrant+virtual box. The following is the lab environment structure.*

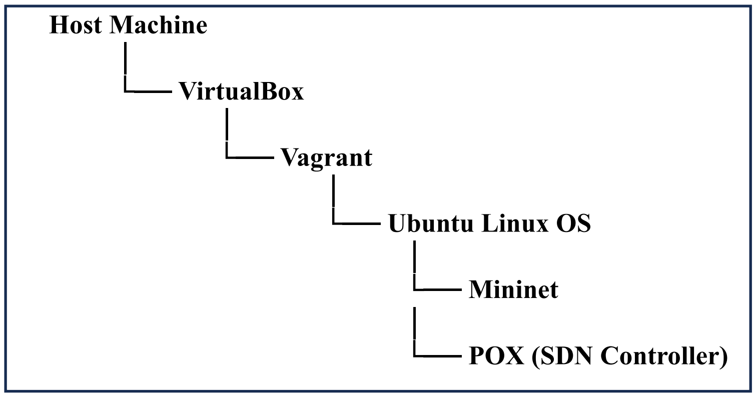
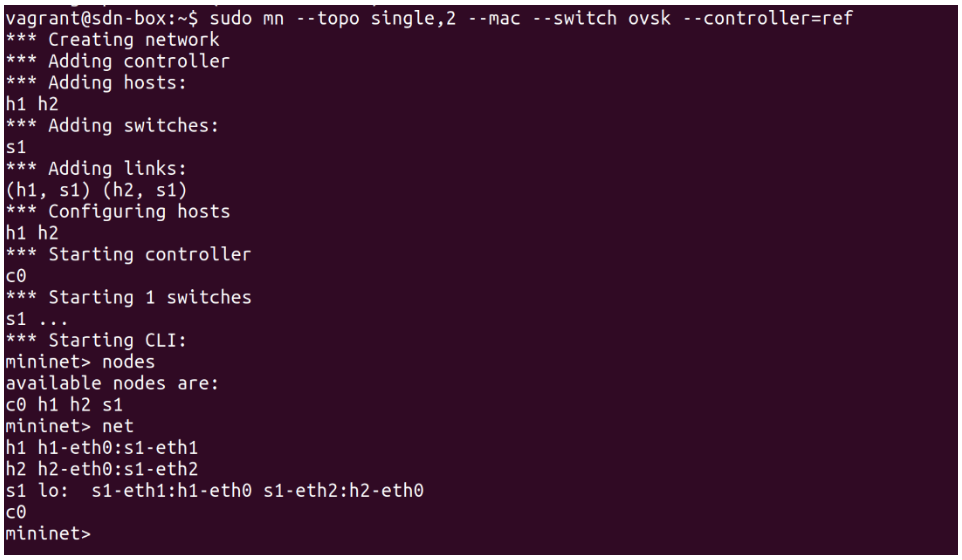


*Figure 2: Lab Environment Setup*

## Create a Basic Mininet Topology



```
vagrant@sdn-box:~$ sudo mn --topo single,2 --mac --switch ovsk --controller=ref
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet>
```

*Figure 3: Basic Topology with Mininet's Default OpenFlow Switch*

*According to the obtained details from verification commands (nodes, net, .. etc.) the following should be the topolgy.*

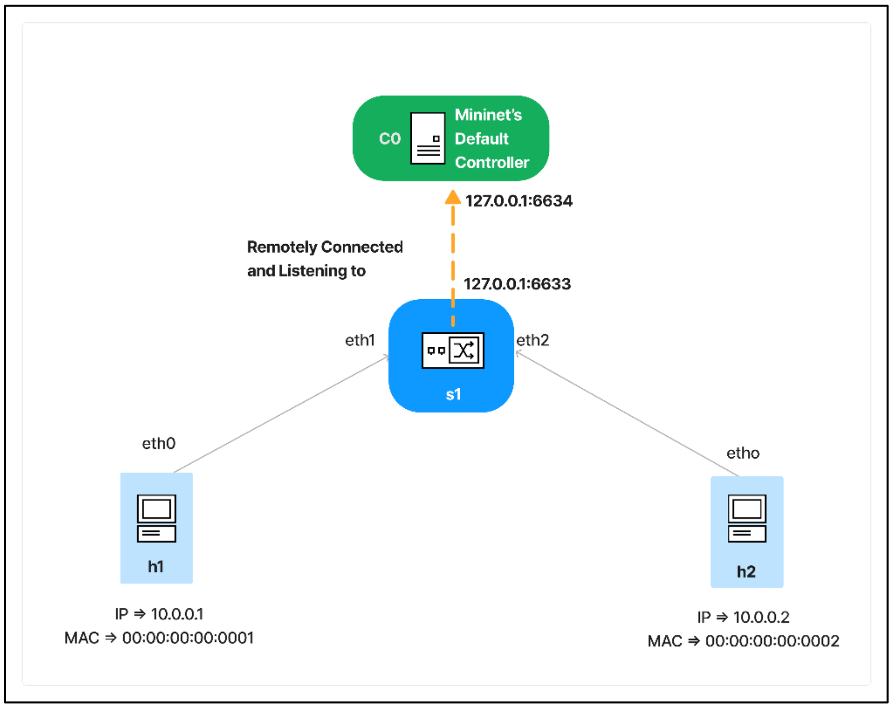
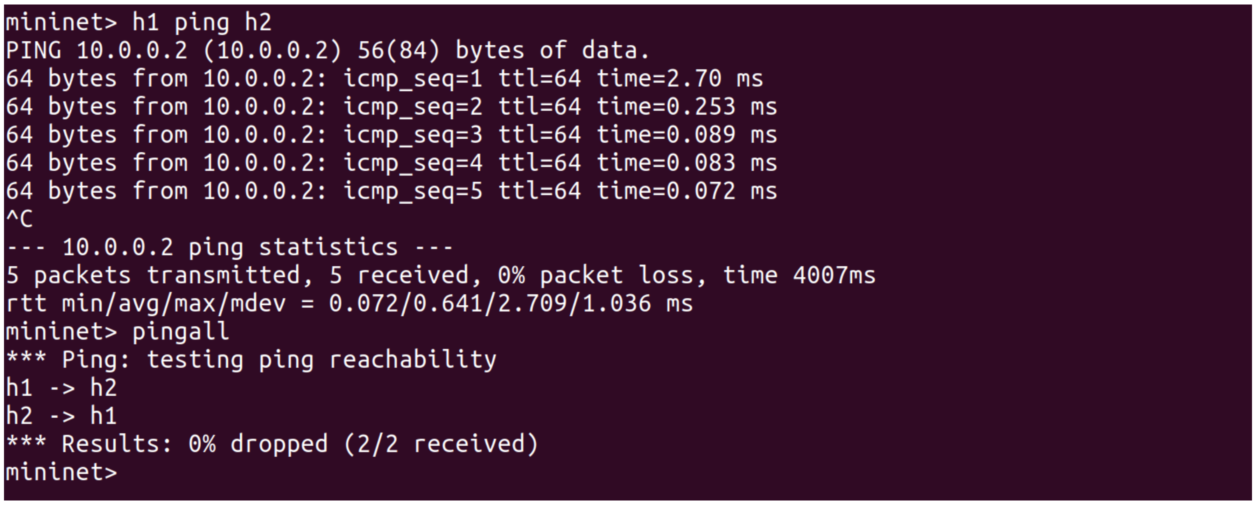

CO
Mininet's Default Controller
127.0.0.1:6634
Remotely Connected and Listening to
127.0.0.1:6633
eth1
eth2
s1
eth0
etho
h1
h2
IP ⇒ 10.0.0.1
MAC ⇒ 00:00:00:00:00:0001
IP ⇒ 10.0.0.2
MAC ⇒ 00:00:00:00:00:0002

*Figure 4: The Basic Mininet Topology*

## <u>Test Connectivity</u>

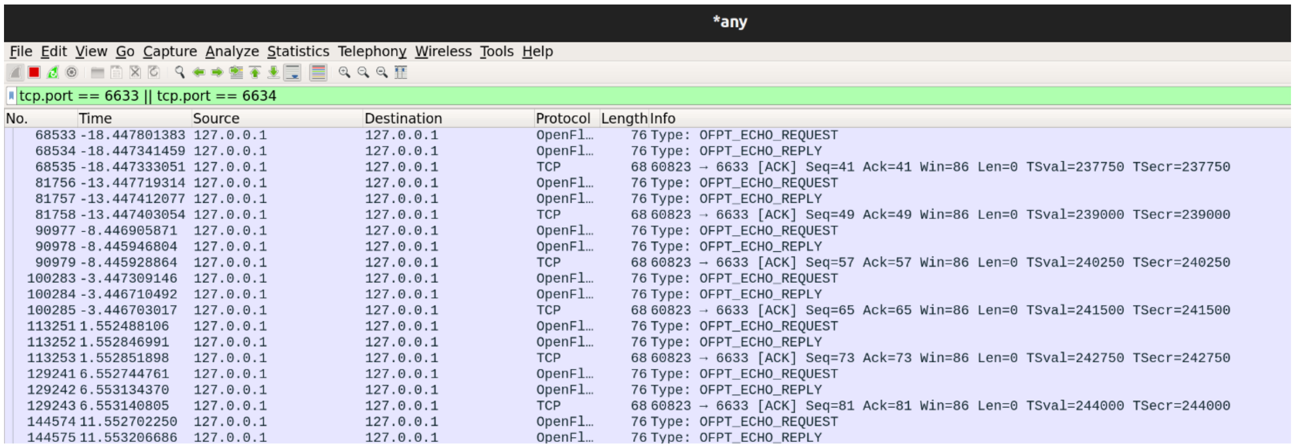
*Since, the default behavior of the Mininet's reference controller (Open vSwitch's inbuilt controller or a simple learning switch) is to handle basic packet forwarding, all the pings should be successful.*



```
mininet> h1 ping h2
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=2.70 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.253 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.072 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 0.072/0.641/2.709/1.036 ms
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

*Figure 5: pingall the Basic Mininet Topology*

*After and before the above pingall, it was observed that the c0 (SDN controller) and the s1 (OpenFlow Switch) was communicating with each other with OpenFlow echo packets to make sure each other is up and running.*


*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 6633 || tcp.port == 6634
No.     Time          Source     Destination  Protocol  Length Info
 68533 -18.447801383 127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REQUEST
 68534 -18.447341459 127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REPLY
 68535 -18.447333051 127.0.0.1  127.0.0.1    TCP       68 60823 → 6633 [ACK] Seq=41 Ack=41 Win=86 Len=0 TSval=237750 TSecr=237750
 81756 -13.447719314 127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REQUEST
 81757 -13.447412077 127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REPLY
 81758 -13.447403054 127.0.0.1  127.0.0.1    TCP       68 60823 → 6633 [ACK] Seq=49 Ack=49 Win=86 Len=0 TSval=239000 TSecr=239000
 90977 -8.446905871  127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REQUEST
 90978 -8.445946804  127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REPLY
 90979 -8.445928864  127.0.0.1  127.0.0.1    TCP       68 60823 → 6633 [ACK] Seq=57 Ack=57 Win=86 Len=0 TSval=240250 TSecr=240250
100283 -3.447309146  127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REQUEST
100284 -3.446710492  127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REPLY
100285 -3.446703017  127.0.0.1  127.0.0.1    TCP       68 60823 → 6633 [ACK] Seq=65 Ack=65 Win=86 Len=0 TSval=241500 TSecr=241500
113251 1.552488106   127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REQUEST
113252 1.552846991   127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REPLY
113253 1.552851898   127.0.0.1  127.0.0.1    TCP       68 60823 → 6633 [ACK] Seq=73 Ack=73 Win=86 Len=0 TSval=242750 TSecr=242750
129241 6.552744761   127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REQUEST
129242 6.553134370   127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REPLY
129243 6.553140805   127.0.0.1  127.0.0.1    TCP       68 60823 → 6633 [ACK] Seq=81 Ack=81 Win=86 Len=0 TSval=244000 TSecr=244000
144574 11.552702250  127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REQUEST
144575 11.553206686  127.0.0.1  127.0.0.1    OpenFl…   76 Type: OFPT_ECHO_REPLY
```

*Figure 6: Wireshark Capture of the OpenFlow Packets Exchanged between the Reference Controller and the OpenFlow Switch*

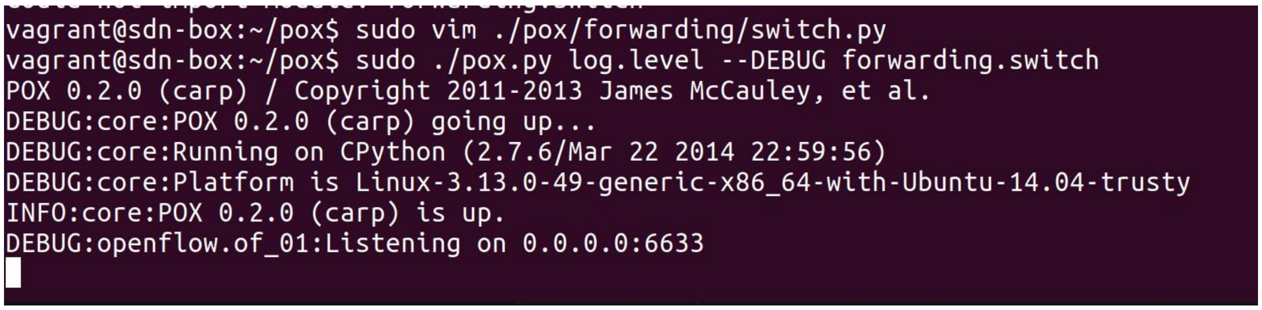
## **Creating a Simple SDN Controller that Uses Flow Rules Switch Packets between Hosts**

*I am using Pox as my SDN controller. (instead of Ryu)*

*The following is the Python script for switch.py*

```
# switch.py

# import the necessary POX components
from pox.core import core
import pox.openflow.libopenflow_01 as of

# Create a logger for this component
log = core.getLogger()

class SimpleSwitch(object):
    def __init__(self, connection):
        # Store a reference to the connection object
        self.connection = connection
        # Bind this object to listen for PacketIn messages
        connection.addListeners(self)

    def _handle_PacketIn(self, event):
        """
        Handles PacketIn messages from the switch.
        This method is called whenever the switch sends a packet to the
controller.
        """
        packet = event.parsed  # Parse the incoming packet

        # Learn the source MAC address to avoid flooding next time.
        # Store the port where the packet came from
        self.mac_to_port[packet.src] = event.port

        # Check if the destination MAC address is known
        if packet.dst in self.mac_to_port:
            # Destination MAC is known, retrieve the output port
            out_port = self.mac_to_port[packet.dst]
            log.debug("Sending packet to port %s", out_port)
            # Create a PacketOut message to send the packet out the correct port
            msg = of.ofp_packet_out()
            msg.data = event.ofp  # Include the original packet data
            action = of.ofp_action_output(port=out_port)  # Specify the output
port
            msg.actions.append(action)  # Add the action to the message
            self.connection.send(msg)  # Send the message to the switch
        else:
            # Destination MAC is unknown, flood the packet out all ports
            log.debug("Flooding packet")
            msg = of.ofp_packet_out()
            msg.data = event.ofp  # Include the original packet data
            msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))  #
Specify flooding action
            self.connection.send(msg)  # Send the message to the switch

    def _handle_ConnectionUp(self, event):
        """
        Handles ConnectionUp messages from the switch.
        This method is called when a switch connects to the controller.
        """
        log.info("Connection %s" % (event.connection,))
        # Initialize a dictionary to map MAC addresses to switch ports
        self.mac_to_port = {}

def launch():
    """
    Starts the SimpleSwitch controller.
    This function is called when the module is run.
    """
    def start_switch(event):
        log.info("Controlling %s" % (event.connection,))
        # Create an instance of SimpleSwitch for each new switch connection
        SimpleSwitch(event.connection)

    # Add an event listener for when switches connect to the controller
    core.openflow.addListenerByName("ConnectionUp", start_switch)
```

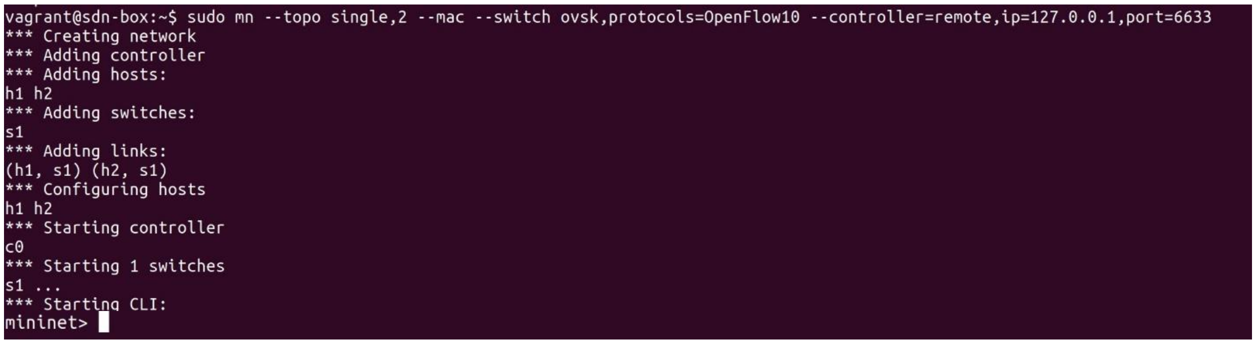
*creating the switch.py inside the correct directory under ./pox and starting the Pox controller*



```
vagrant@sdn-box:~/pox$ sudo vim ./pox/forwarding/switch.py
vagrant@sdn-box:~/pox$ sudo ./pox.py log.level --DEBUG forwarding.switch
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-3.13.0-49-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
```

*Figure 7: Starting the Pox SDN Controller with switch.py*

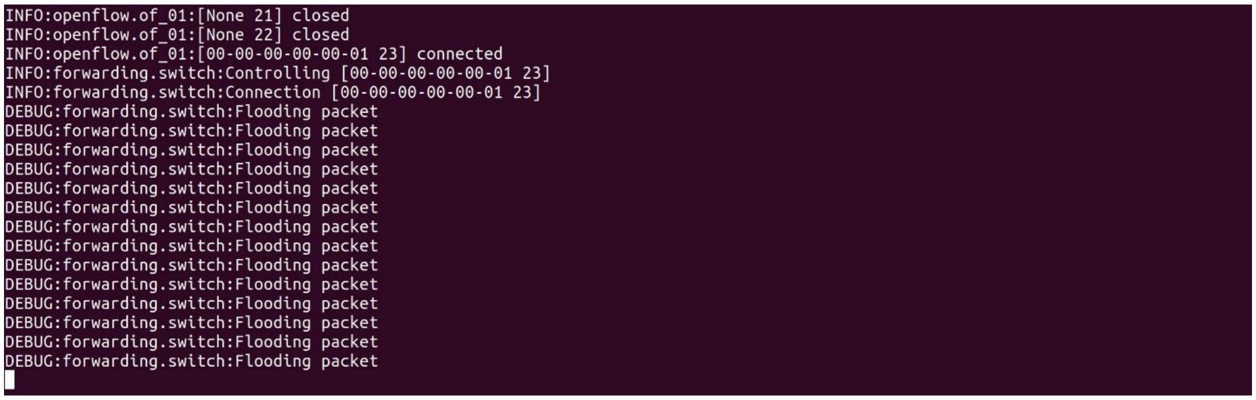
**Starting the Mininet Topology with the Custom Controller**



```
vagrant@sdn-box:~$ sudo mn --topo single,2 --mac --switch ovsk,protocols=OpenFlow10 --controller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> 
```

*Figure 8: Starting the Mininet Topology with the Custom Controller*

*Right after starting the mininet topology, it was observed at the c0 controller's debug messages, switch learning has been occurring with being detected the FLOOD packets by the c0.*



```
INFO:openflow.of_01:[None 21] closed
INFO:openflow.of_01:[None 22] closed
INFO:openflow.of_01:[00-00-00-00-00-01 23] connected
INFO:forwarding.switch:Controlling [00-00-00-00-00-01 23]
INFO:forwarding.switch:Connection [00-00-00-00-00-01 23]
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
DEBUG:forwarding.switch:Flooding packet
```

*Figure 9: FLOOD Packets Observed Right-After Starting the Mininet Topology*

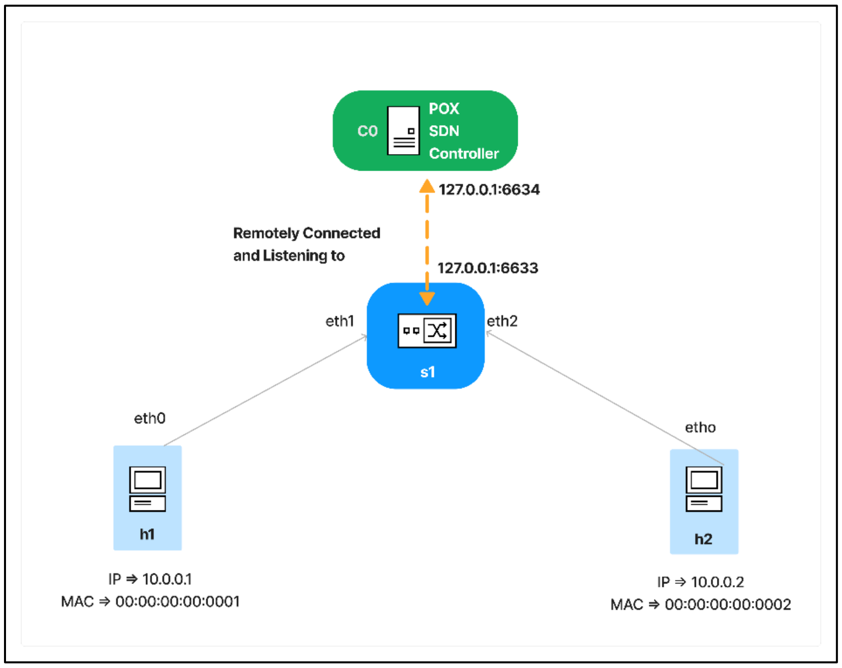
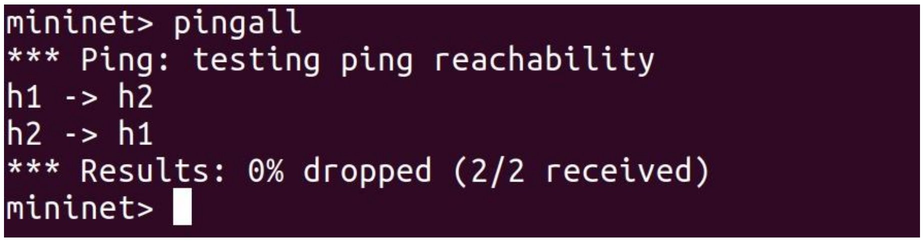


*Figure 10: The Topology Created with the Custom POX SDN Controller*

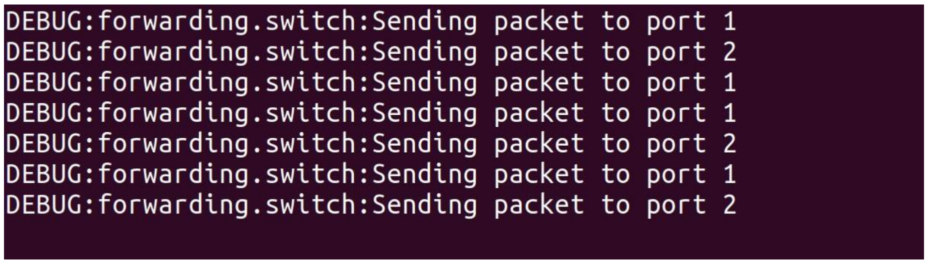
## Testing the Packet Switch



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

*Figure 11: pingall the New Topology*

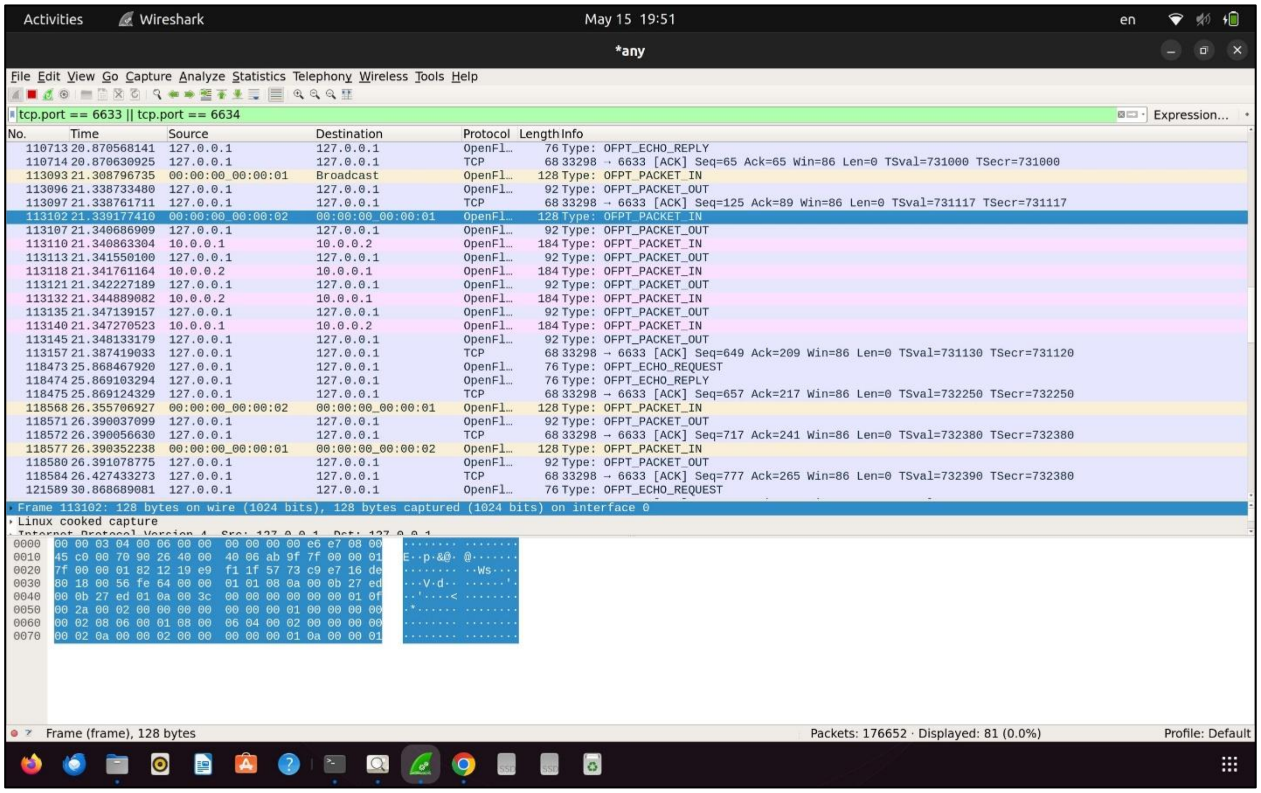
*The debug terminal of the pox controller with switch.py has detected the packet exchanges between the 2 hosts (h1 and h2)*



```
DEBUG:forwarding.switch:Sending packet to port 1
DEBUG:forwarding.switch:Sending packet to port 2
DEBUG:forwarding.switch:Sending packet to port 1
DEBUG:forwarding.switch:Sending packet to port 1
DEBUG:forwarding.switch:Sending packet to port 2
DEBUG:forwarding.switch:Sending packet to port 1
DEBUG:forwarding.switch:Sending packet to port 2
```

*Figure 12: Debug Terminal of switch.py During the Pingall*

*During the pingall the OpenFlow packets PACKET_IN and PACKET_OUT has been captured and observed using wireshark*

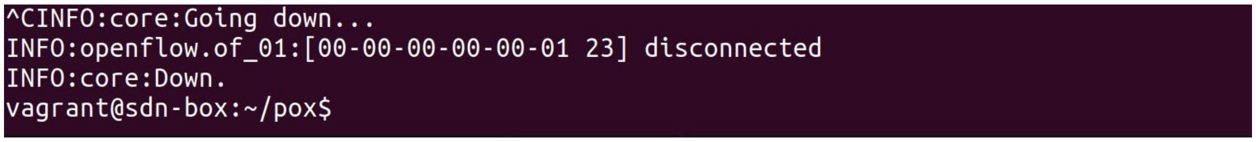


Activities
Wireshark
May 15 19:51
en
*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
tcp.port == 6633 || tcp.port == 6634

No.    Time               Source              Destination         Protocol  Length Info
110713 20.870568141  127.0.0.1           127.0.0.1           OpenFl…    76 Type: OFPT_ECHO_REPLY
110714 20.870630925  127.0.0.1           127.0.0.1           TCP        68 33298 → 6633 [ACK] Seq=65 Ack=65 Win=86 Len=0 TSval=731000 TSecr=731000
113093 21.308796735  00:00:00_00:00:01   Broadcast           OpenFl…   128 Type: OFPT_PACKET_IN
113096 21.338733480  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
113097 21.338761711  127.0.0.1           127.0.0.1           TCP        68 33298 → 6633 [ACK] Seq=125 Ack=89 Win=86 Len=0 TSval=731117 TSecr=731117
113102 21.339177410  00:00:00_00:00:02   00:00:00_00:00:01   OpenFl…   128 Type: OFPT_PACKET_IN
113107 21.340686909  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
113110 21.340863304  10.0.0.1            10.0.0.2            OpenFl…   184 Type: OFPT_PACKET_IN
113113 21.341550100  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
113118 21.341761164  10.0.0.2            10.0.0.1            OpenFl…   184 Type: OFPT_PACKET_IN
113121 21.342227189  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
113132 21.344889082  10.0.0.2            10.0.0.1            OpenFl…   184 Type: OFPT_PACKET_IN
113135 21.347139157  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
113140 21.347270523  10.0.0.1            10.0.0.2            OpenFl…   184 Type: OFPT_PACKET_IN
113145 21.348133179  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
113157 21.387419033  127.0.0.1           127.0.0.1           TCP        68 33298 → 6633 [ACK] Seq=649 Ack=209 Win=86 Len=0 TSval=731130 TSecr=731120
118473 25.868467920  127.0.0.1           127.0.0.1           OpenFl…    76 Type: OFPT_ECHO_REQUEST
118474 25.869103294  127.0.0.1           127.0.0.1           OpenFl…    76 Type: OFPT_ECHO_REPLY
118475 25.869124329  127.0.0.1           127.0.0.1           TCP        68 33298 → 6633 [ACK] Seq=657 Ack=217 Win=86 Len=0 TSval=732250 TSecr=732250
118568 26.355706927  00:00:00_00:00:02   00:00:00_00:00:01   OpenFl…   128 Type: OFPT_PACKET_IN
118571 26.390037099  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
118572 26.390056630  127.0.0.1           127.0.0.1           TCP        68 33298 → 6633 [ACK] Seq=717 Ack=241 Win=86 Len=0 TSval=732380 TSecr=732380
118577 26.390352238  00:00:00_00:00:01   00:00:00_00:00:02   OpenFl…   128 Type: OFPT_PACKET_IN
118580 26.391078775  127.0.0.1           127.0.0.1           OpenFl…    92 Type: OFPT_PACKET_OUT
118584 26.427433273  127.0.0.1           127.0.0.1           TCP        68 33298 → 6633 [ACK] Seq=777 Ack=265 Win=86 Len=0 TSval=732390 TSecr=732380
121589 30.868689081  127.0.0.1           127.0.0.1           OpenFl…    76 Type: OFPT_ECHO_REQUEST

› Frame 113102: 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits) on interface 0
› Linux cooked capture

0000  00 00 03 04 00 06 00 00  00 00 00 00 e6 e7 08 00   ········ ········
0010  45 c0 00 70 90 26 40 00  40 06 ab 9f 7f 00 00 01   E··p·&@· @·······
0020  7f 00 00 01 82 12 19 e9  f1 1f 57 73 c9 e7 16 de   ········ ··Ws····
0030  80 18 00 56 fe 64 00 00  01 01 08 0a 00 0b 27 ed   ···V·d·· ······'·
0040  00 0b 27 ed 01 0a 00 3c  00 00 00 00 00 00 01 0f   ··'····< ········
0050  00 2a 00 02 00 00 00 00  00 00 00 01 00 00 00 00   ·*······ ········
0060  00 02 08 06 00 01 08 00  06 04 00 02 00 00 00 00   ········ ········
0070  00 02 0a 00 00 02 00 00  00 00 00 01 0a 00 00 01   ········ ········

Frame (frame), 128 bytes          Packets: 176652 · Displayed: 81 (0.0%)          Profile: Default
```

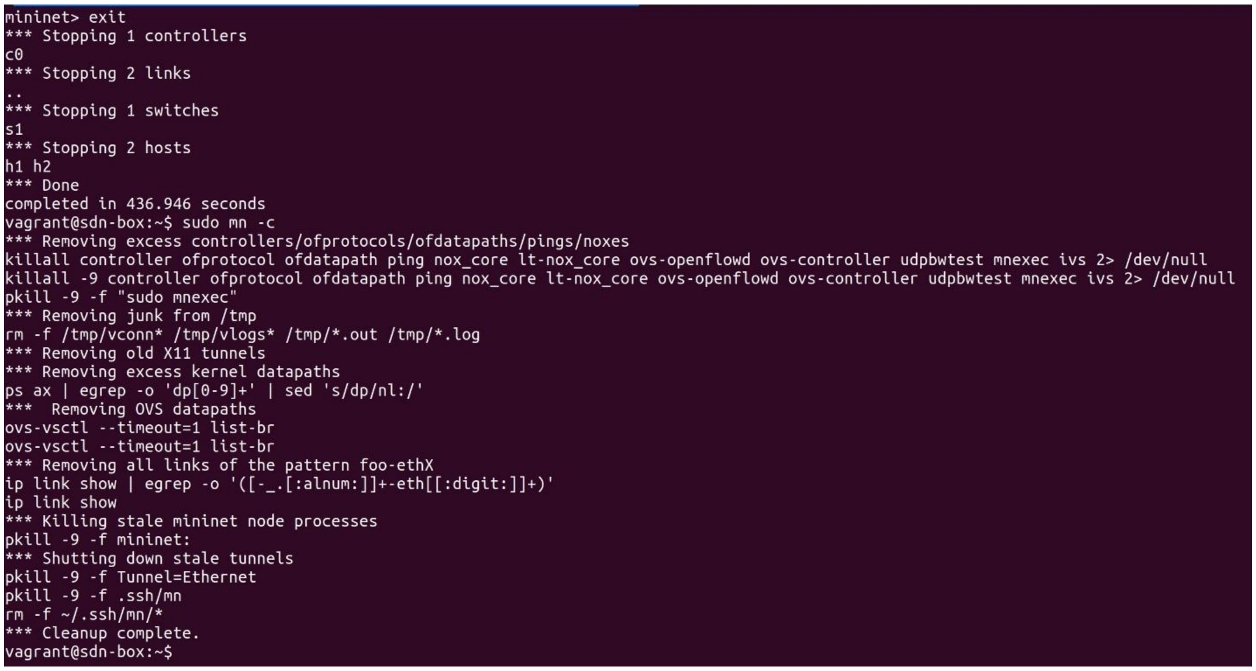
*Figure 13: Wireshark Output of pingall in the New Topology*

## Cleaning Up the Setup



```
^CINFO:core:Going down...
INFO:openflow.of_01:[00-00-00-00-00-01 23] disconnected
INFO:core:Down.
vagrant@sdn-box:~/pox$
```

*Figure 14: Cleaning Up the c0 SDN controller with switch.py*



```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 436.946 seconds
vagrant@sdn-box:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
vagrant@sdn-box:~$
```

*Figure 15: Cleaning Up the Mininet Topology*