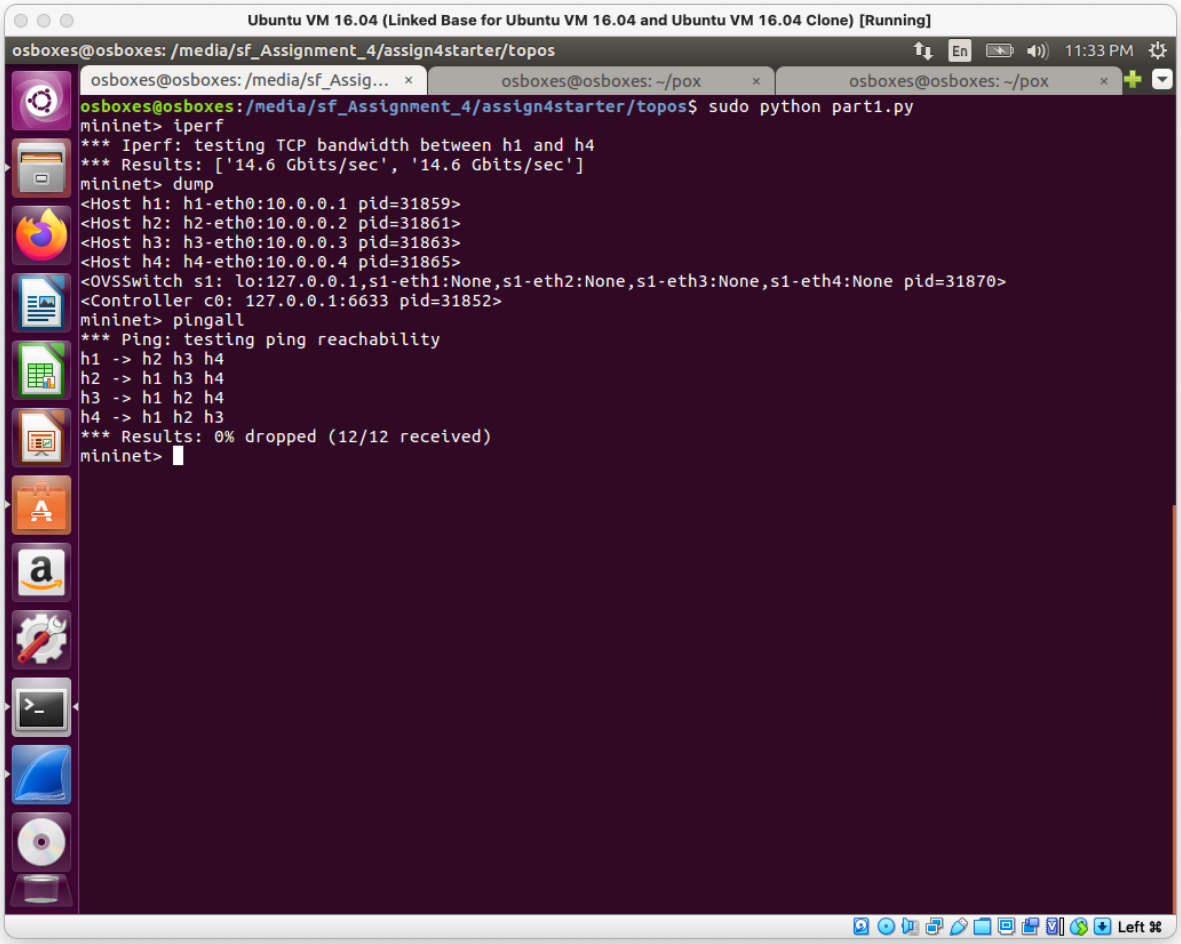
Erik Macik
Computer Networks

Assignment 4 – SDN

Task-1: Programming Mininet Topologies

1. In this task of creating a network topology with one switch connected to four unique hosts, I edited the provided part1.py file. I edited the build() method inside of the part1_topo() class to create the switch and four hosts, then add a link from each host to the switch.
2. Screenshot proof of working topology with iperf, dump, and pingall command ouputs:

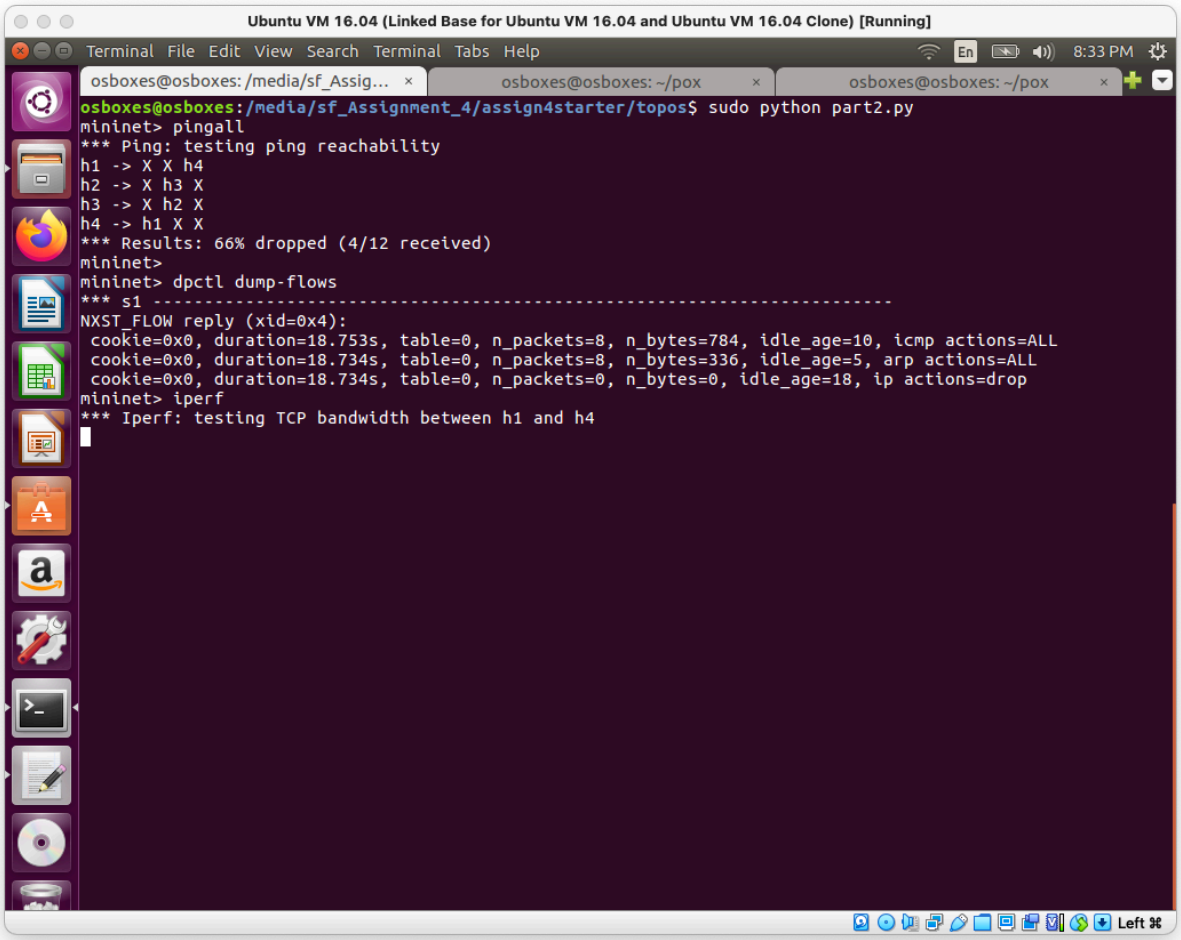


Task-2: SDN Controller using POX

1. In order to create a new controller, I edited the __init__() function of the provided Firewall() class in part2controller.py. I created rules that are of type of.ofp_flow_mod() and eventually send them to the switch.
    a. Rule 1: accept icmp traffic of ipv4
    b. Rule 2: accept arp traffic of any kind
    c. Rule 3: drop any other packet of ipv4
2. In short, I deployed the controller to pox by putting it inside the ext directory inside of the pox repository from GitHub, and ran the pox controller with the command "./pox.py

part2controller". After the controller was up and running, I then ran the provided topology file with "sudo python part2.py".

3. Screenshot proof of working edited controller with provided topology file with pingall, iperf, and dpctl dump-flows command outputs:



Erik Macik
Computer Networks

Source Code

Part 1

topos/part1.py (edited)

```
#!/usr/bin/python

# Edited by Erik Macik

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.cli import CLI

class part1_topo(Topo):

    def build(self):
        s1 = self.addSwitch('s1')
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')

        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s1)
        self.addLink(h4, s1)


topos = {'part1' : part1_topo}

if __name__ == '__main__':
    t = part1_topo()
    net = Mininet (topo=t)
    net.start()
    CLI(net)
    net.stop()
```

Part 2

topos/part2.py (unedited)

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import RemoteController
```

```
class part2_topo(Topo):
  def build(self):
    s1 = self.addSwitch('s1')
    h1 =
self.addHost('h1',mac='00:00:00:00:00:01',ip='10.0.1.2/24')
    h2 =
self.addHost('h2',mac='00:00:00:00:00:02',ip='10.0.0.2/24')
    h3 =
self.addHost('h3',mac='00:00:00:00:00:03',ip='10.0.0.3/24')
    h4 =
self.addHost('h4',mac='00:00:00:00:00:04',ip='10.0.1.3/24')
    self.addLink(h1,s1)
    self.addLink(h2,s1)
    self.addLink(h3,s1)
    self.addLink(h4,s1)

topos = {'part2' : part2_topo}

def configure():
  topo = part2_topo()
  net = Mininet(topo=topo, controller=RemoteController)
  net.start()

  CLI(net)

  net.stop()


if __name__ == '__main__':
  configure()
```

topos/part2controller.py (edited)

```
# Edited by Erik Macik

from pox.core import core
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

class Firewall (object):
  """
  A Firewall object is created for each switch that connects.
  A Connection object for that switch is passed to the __init__
function.
  """
  def __init__ (self, connection):
```

```
    # Keep track of the connection to the switch so that we can
    # send it messages!
    self.connection = connection

    # This binds our PacketIn event listener
    connection.addListeners(self)

    # add switch rules here
    # src and dst: ipv4
    # protocol: icmp
    # action: accept
    # Create rule and send to controller
    rule1 = of.ofp_flow_mod()
    rule1.match.dl_type = 0x800
    rule1.match.nw_proto = 1
    rule1.actions.append(of.ofp_action_output(port =
of.OFPP_ALL))
    self.connection.send(rule1)

    # src and dst: any
    # protocol: arp
    # action: accept
    # Create rule and send to controller
    rule2 = of.ofp_flow_mod()
    rule2.match.dl_type = 0x0806
    rule2.actions.append(of.ofp_action_output(port =
of.OFPP_ALL))
    self.connection.send(rule2)

    # src and dst: ipv4
    # protocol: -
    # action: drop
    # Create rule and send to controller
    rule3 = of.ofp_flow_mod()
    rule3.match.dl_type = 0x800
    self.connection.send(rule3)

  def _handle_PacketIn (self, event):
    """
    Packets not handled by the router rules will be
    forwarded to this method to be handled by the controller
    """

    packet = event.parsed # This is the parsed packet data.
    if not packet.parsed:
      log.warning("Ignoring incomplete packet")
      return
```

```
    packet_in = event.ofp # The actual ofp_packet_in message.
    print ("Unhandled packet :" + str(packet.dump()))

def launch ():
  """
  Starts the component
  """
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Firewall(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
```

References

A. Al-Shabibi, “POX Wiki,” Stanford, 2013. [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki.html#POXWiki-Output. [Accessed: 2020].