

ASSINGMENT NO. 1

AIM : Prepare setup for Mininet network emulation environment with the help of Virtual box and Mininet. Demonstrate the basic commands in Mininet and emulate different custom network topology (Simple, Linear, and Tree).View flow tables.

THEORY :

Mininet is one type of network emulator used to simulate the network with collection of end-hosts, switches, routers and link on a single Linux kernel. The Mininet tool uses lightweight virtualization to create a single system look like a complete network which running the same kernel, system, and user code. Mininet is commonly used by the open source SDN community as a simulation, verification and testing tool.

In SDN network simulation is done by using lightweight Mininet software. The platform like Mininet is providing all required infrastructure to simulate network. Some of the requirements are

1. Speed of deployment
2. Predictable Output
3. Ease of use

For network simulation many tools are available. Three main options were discovered during research such as Cisco Virtual Internet Routing Lab (VIRL), Mininet and virtualizing devices using images.

The Cisco proprietary software VIRL consists of real networking equipment operating systems in the form of modifier images. Defining link parameters and graphical topology editor are some features of VIRL. The experimental result with VIRL was conclude that it is unstable and requires significant memory resources and difficulty for linking SDN controller to the devices.

The working experience with virtualized device images was quite good but problem with this method is consumed high memory and slow start. Also additional configuration requires each device in order to work with OpenFlow.

The Open source flexible Mininet software was created with support of OpenFlow. It is de facto standard for making experiments with OpenFlow and SDN. It supports different types underlying controller technologies. For this lab open vSwitch was chosen as open source, popular, reliable and efficient solution.

After consideration, Mininet was chosen as network simulation software. Also there some advantages of Mininet such as low resource requirements and fast startup time. Device, topologies and links behavior can be easily changed by scripting. More, Mininet interfaces can be easily listened by network sniffer.

The Mininet is switch oriented software so layer three functions such as routing protocols and MPLS should be implemented on the controller. As the labs will be OpenFlow focused features of the SDN controllers such as BGP-speaker, VPN tunneling or MPLS configuration are out of the main focus of the lab and their lack is not significant.

Following figure shows a simple Mininet network comprised of three hosts, a virtual OpenFlow switch, and an OpenFlow controller. All components are connected over virtual Ethernet links that are then assigned private net-10 IP addresses for reachability. Mininet supports very complex topologies. Mininet allows you to create custom topologies for that reason it mostly used for experimentation purpose.

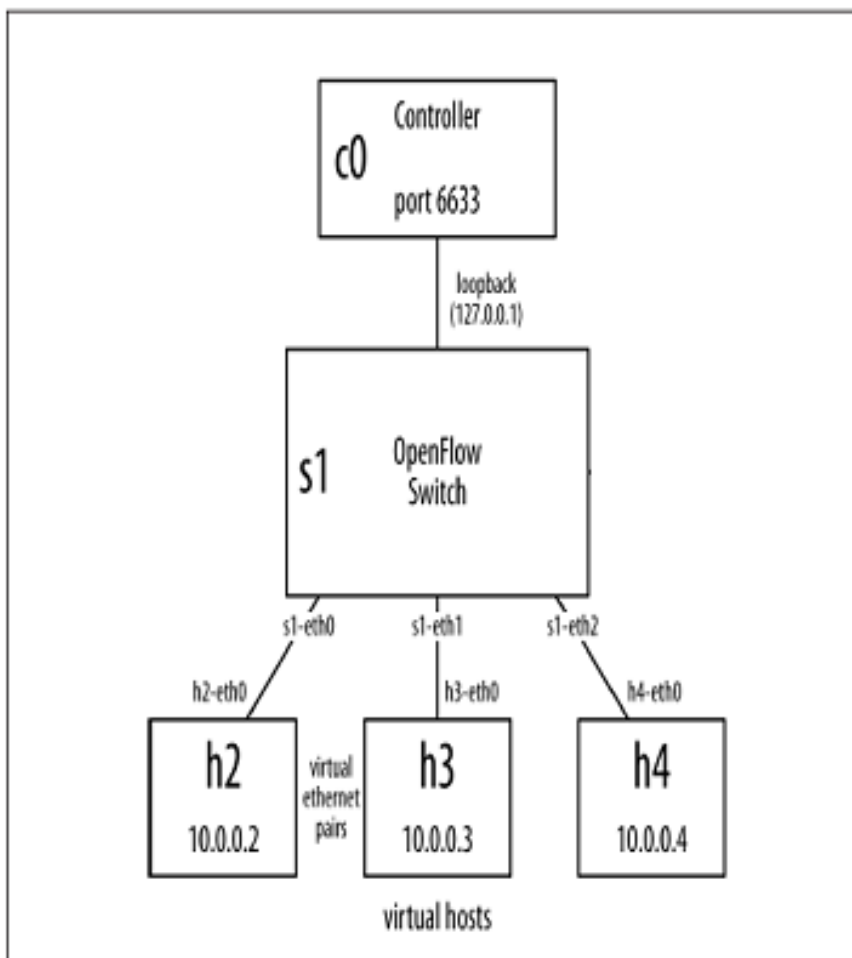


Figure: Simple Mininet Network

Mininet Installation:

Native Installation from Source

This option works well for local VM, remote EC2, and native installation. It assumes the starting point of a fresh Ubuntu, Debian, or (experimentally) Fedora installation.

We strongly recommend more recent Ubuntu or Debian releases, because they include newer versions of Open vSwitch. (Fedora also includes recent OvS releases.)

To install natively from source, first you need to get the source code:

```
git clone https://github.com/mininet/mininet
```

Note that the above `git` command will check out the latest and greatest Mininet (which we recommend!) If you want to run the last tagged/released version of Mininet - or any other version - you may check that version out explicitly:

```
cd mininet
git tag # list available versions
git checkout -b mininet-2.3.0 2.3.0 # or whatever version you wish to install
cd ..
```

Once you have the source tree, the command to install Mininet is:

```
mininet/util/install.sh [options]
```

Typical `install.sh` options include:

- `-a`: install everything that is included in the Mininet VM, including dependencies like Open vSwitch as well the additions like the OpenFlow wireshark dissector and POX. By default these tools will be built in directories created in your home directory.
- `-nfv`: install Mininet, the OpenFlow reference switch, and Open vSwitch
- `-s mydir`: use this option before other options to place source/build trees in a specified directory rather than in your home directory.

So, you will probably wish to use one (and only one) of the following commands:

```
To install everything (using your home directory): install.sh -a
To install everything (using another directory for build): install.sh -s mydir -a
To install Mininet + user switch + OvS (using your home dir): install.sh -nfv
To install Mininet + user switch + OvS (using another dir:) install.sh -s mydir -nfv
```

You can find out about other useful options (e.g. installing the OpenFlow wireshark dissector, if it's not already included in your version of wireshark) using

```
install.sh -h
```

After the installation has completed, test the basic Mininet functionality:

```
sudo mn --switch ovsbr --test pingall
```

Displaying Custom Topology on Miniedit

```
1-> sudo apt update
2-> sudo apt install python3-pip
3-> sudo pip3 install ryu
4-> sudo apt install git
5-> git clone https://github.com/mininet/mininet
6-> ls (to check the file mininet)
7-> cd mininet/
8-> ls (check file util)
9-> cd util/
10-> sudo ./install.sh -a (it will take long time 10 minutes)
11-> sudo mn
    1->(mininet> pingall )
    2->exit
12->sudo mn -c
14->sudo mn --topo single,3 --mac --switch ovsk --controller remote
    1->pingall
    2->exit
15->cd mininet/
16->sudo apt install python3
17->cd mininet/
18->cd examples/
19->cd sudo python3 ./miniedit.py
```

Mininet Commands

Interact with Hosts and Switches

Start a minimal topology and enter the CLI:

```
$ sudo mn
```

The default topology is the `minimal` topology, which includes one OpenFlow kernel switch connected to two hosts, plus the OpenFlow reference controller. This topology could also be specified on the command line with `--topo=minimal`. Other topologies are also available out of the box; see the `--topo` section in the output of `mn -h`.

All four entities (2 host processes, 1 switch process, 1 basic controller) are now running in the VM. The controller can be outside the VM, and instructions for that are at the bottom.

If no specific test is passed as a parameter, the Mininet CLI comes up.

Display Mininet CLI commands:

```
mininet> help
```

Display nodes:

```
mininet> nodes
```

Display links:

```
mininet> net
```

Dump information about all nodes:

```
mininet> dump
```

You should see the switch and two hosts listed.

If the first string typed into the Mininet CLI is a host, switch or controller name, the command is executed on that node. Run a command on a host process:

```
mininet> h1 ifconfig -a
```

You should see the host's `h1-eth0` and loopback (`lo`) interfaces. Note that this interface (`h1-eth0`) is not seen by the primary Linux system when `ifconfig` is run, because it is specific to the network namespace of the host process.

In contrast, the switch by default runs in the root network namespace, so running a command on the “switch” is the same as running it from a regular terminal:

```
mininet> s1 ifconfig -a
```

This will show the switch interfaces, plus the VM's connection out (`eth0`).

For other examples highlighting that the hosts have isolated network state, run `arp` and `route` on both `s1` and `h1`.

It would be possible to place every host, switch and controller in its own isolated network namespace, but there's no real advantage to doing so, unless you want to replicate a complex multiple-controller network. Mininet does support this; see the `--innamespace` option.

Note that *only* the network is virtualized; each host process sees the same set of processes and directories. For example, print the process list from a host process:

```
mininet> h1 ps -a
```

This should be the exact same as that seen by the root network namespace:

```
mininet> s1 ps -a
```

It would be possible to use separate process spaces with Linux containers, but currently Mininet doesn't do that. Having everything run in the "root" process namespace is convenient for debugging, because it allows you to see all of the processes from the console using `ps`, `kill`, etc.

Test connectivity between hosts

Now, verify that you can ping from host 0 to host 1:

```
mininet> h1 ping -c 1 h2
```

If a string appears later in the command with a node name, that node name is replaced by its IP address; this happened for h2.

You should see OpenFlow control traffic. The first host ARPs for the MAC address of the second, which causes a `packet_in` message to go to the controller. The controller then sends a `packet_out` message to flood the broadcast packet to other ports on the switch (in this example, the only other data port). The second host sees the ARP request and sends a reply. This reply goes to the controller, which sends it to the first host and pushes down a flow entry.

Now the first host knows the MAC address of the second, and can send its ping via an ICMP Echo Request. This request, along with its corresponding reply from the second host, both go to the controller and result in a flow entry pushed down (along with the actual packets getting sent out).

Repeat the last `ping`:

```
mininet> h1 ping -c 1 h2
```

You should see a much lower `ping` time for the second try (< 100us). A flow entry covering ICMP `ping` traffic was previously installed in the switch, so no control traffic was generated, and the packets immediately pass through the switch.

An easier way to run this test is to use the Mininet CLI built-in `pingall` command, which does an all-pairs `ping`:

```
mininet> pingall
```

Changing Topology Size and Type

The default topology is a single switch connected to two hosts. You could change this to a different topology with `--topo`, and pass parameters for that topology's creation. For example, to verify all-pairs ping connectivity with one switch and three hosts:

Run a regression test:

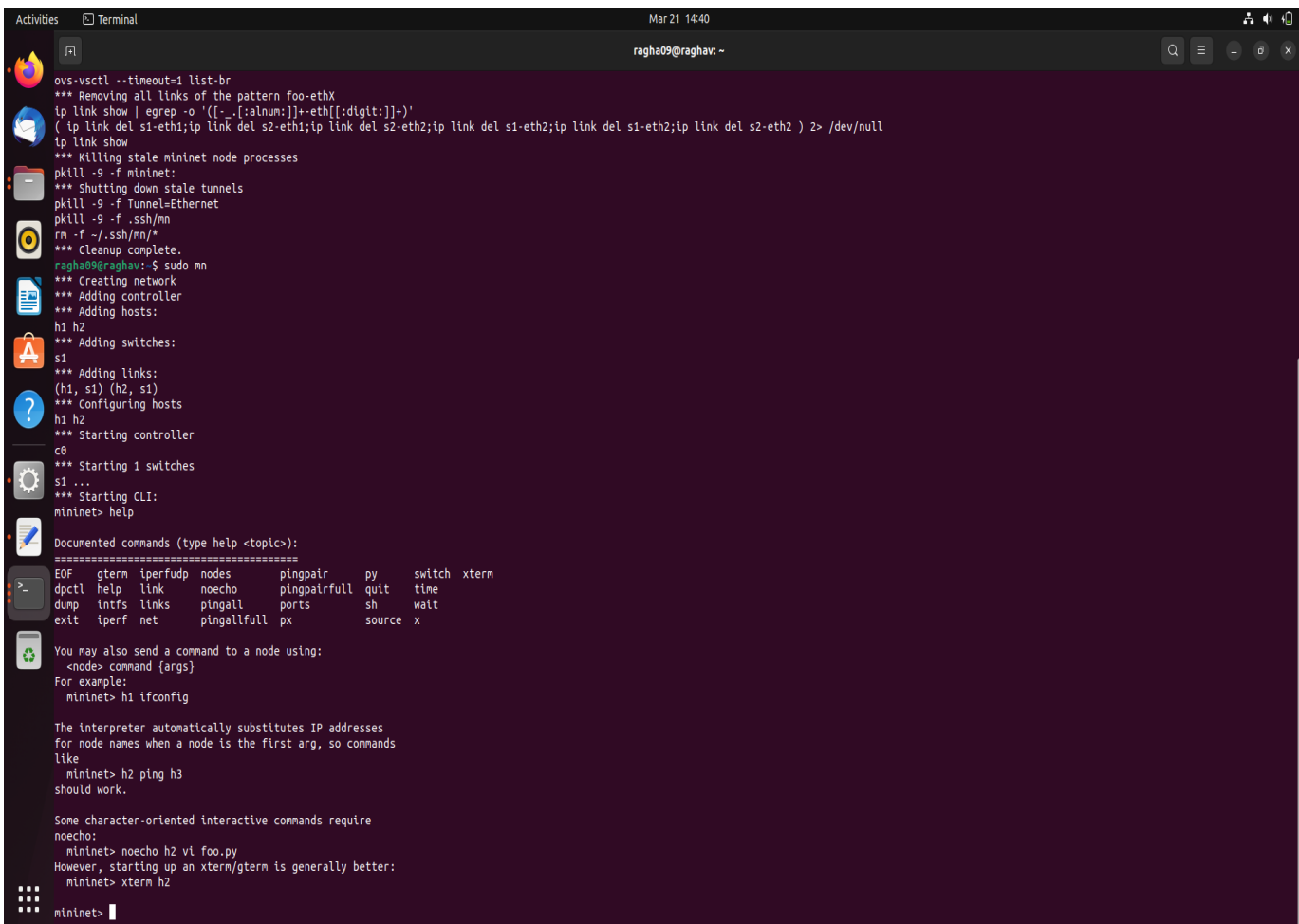
```
$ sudo mn --test pingall --topo single,3
```

Another example, with a linear topology (where each switch has one host, and all switches connect in a line):

```
$ sudo mn --test pingall --topo linear,4
```

Parametrized topologies are one of Mininet's most useful and powerful features.

OUTPUT:



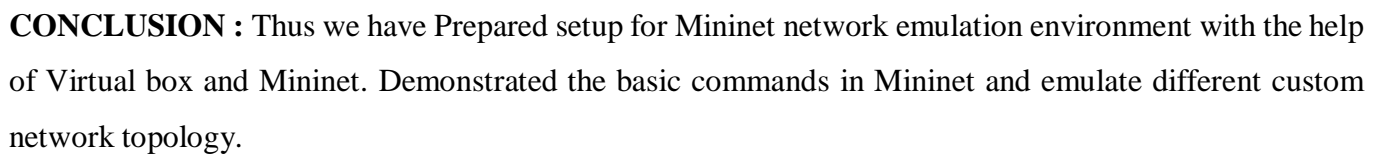
```
Activities Terminal Mar 21 14:40
ragha09@raghav: ~
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([[:alnum:]]+-eth[[:digit:]]+)'
( ip link del s1-eth1; ip link del s2-eth1; ip link del s2-eth2; ip link del s1-eth2; ip link del s1-eth2; ip link del s2-eth2 ) 2> /dev/null
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.
ragha09@raghav:~$ sudo mn
*** 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> help
Documented commands (type help <topic>):
=====
EOF  gterm  iperfudp  nodes  pingpair  py  switch  xterm
dctl  help  link  noecho  pingpairfull  quit  time
dump  intfs  links  pingall  ports  sh  wait
exit  iperf  net  pingallfull  px  source  x

You may also send a command to a node using:
<node> command [args]
For example:
mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
mininet> xterm h2

mininet>
```



ASSIGNMENT NO: 2

AIM: After studying open source POX and Floodlight controller, Install controller and run custom topology using remote controller like POX and floodlight controller. Recognize inserted flows by controllers.

THEORY

FLOODLIGHT

The Floodlight is very popular Open SDN Controller which is Java based, Apache-licensed an Enterprise-class controller. It is supported by community of developers including a number of engineers from Big Switch Networks. Big Switch Networks is a market leader in Software-Defined Networking and a strong proponent of OpenFlow technology. Floodlight SDN controller that used Beacon controller as its foundation, as beacon is an open source controller developed at Stanford.

Before we dig deep into Floodlight Controller, let like to clarify the term ‘Floodlight’ as it is used by the Big Switch’s opensource program. The term ‘Project Floodlight’ is used as an umbrella-term to cover multiple projects such as Floodlight Controller, Indigo, LoxiGen, and OFTest.

When we run the controller, both northbound and southbound operations of the controller become active and the controller as well as set of configured module applications will run too. The northbound REST APIs exposed by all the running modules becomes available via the specified REST port. To retrieve the information and invoke the services, the applications can interact with each other by sending the http REST command. At the southbound, the floodlight provider module will start start listening on the OpenFlow-specified TCP-port for connections from the OpenFlow switches. Floodlight supports OpenFlow version 1.0, 1.3 and 1.4 are in the pipeline process. It is robust and easy to use controller.

POX

POX provides a framework for communicating with SDN switches using either the OpenFlow or OVSDB protocol. Developers can use POX to create an SDN controller using the Python programming language. It is a popular tool for teaching about and researching software defined networks and network applications programming.

POX can be immediately used as a basic SDN controller by using the stock components that come bundled with it. This is the scenario we will cover in this tutorial.

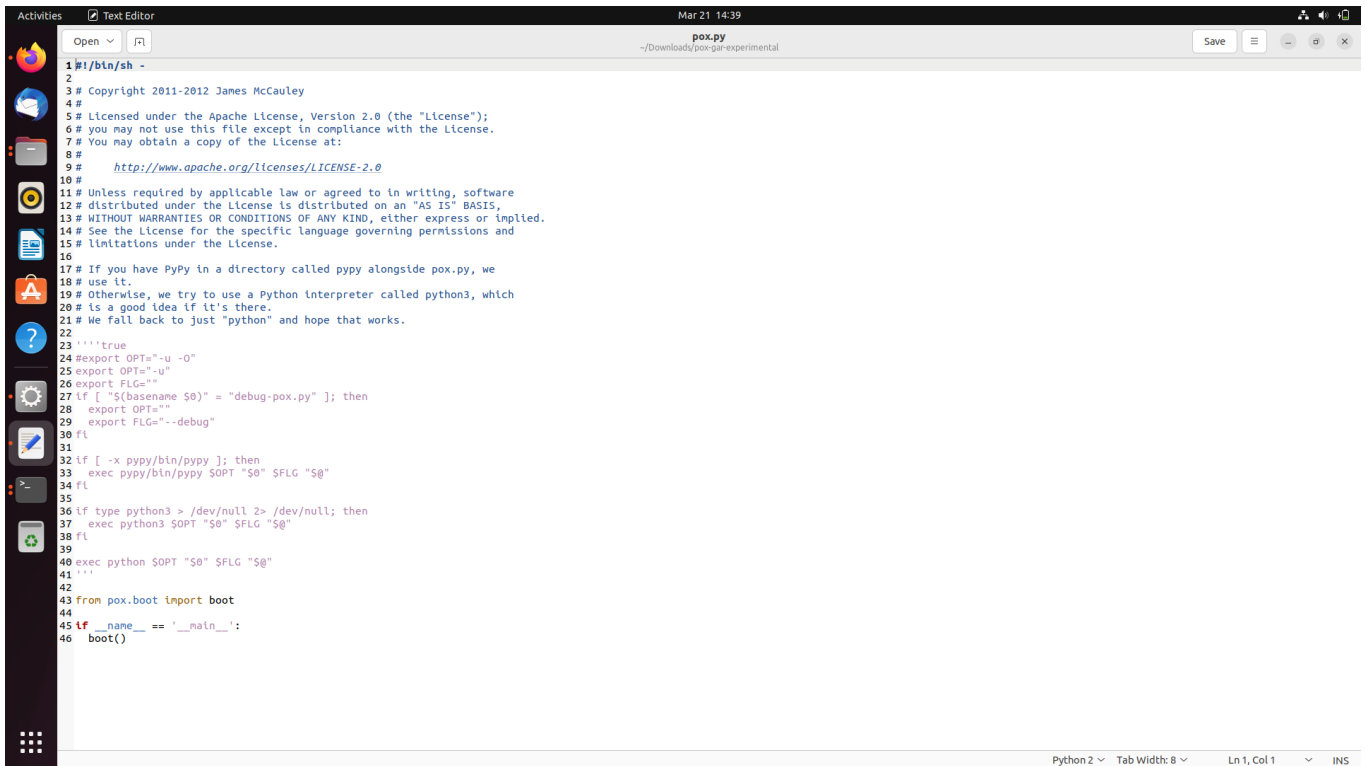
Developers may create a more complex SDN controller by creating new POX components. Or, developers may write network applications that address POX's API. In this tutorial, we only briefly discuss programming for POX.

POX components

POX components are additional Python programs that can be invoked when POX is started from the command line. These components implement the network functionality in the software defined network. POX comes with some stock components already available.

The POX stock components are documented in the POX Wiki and the code for each component can be found in the `~/pox/pox` directory on the *Mininet 2.2* VM image.

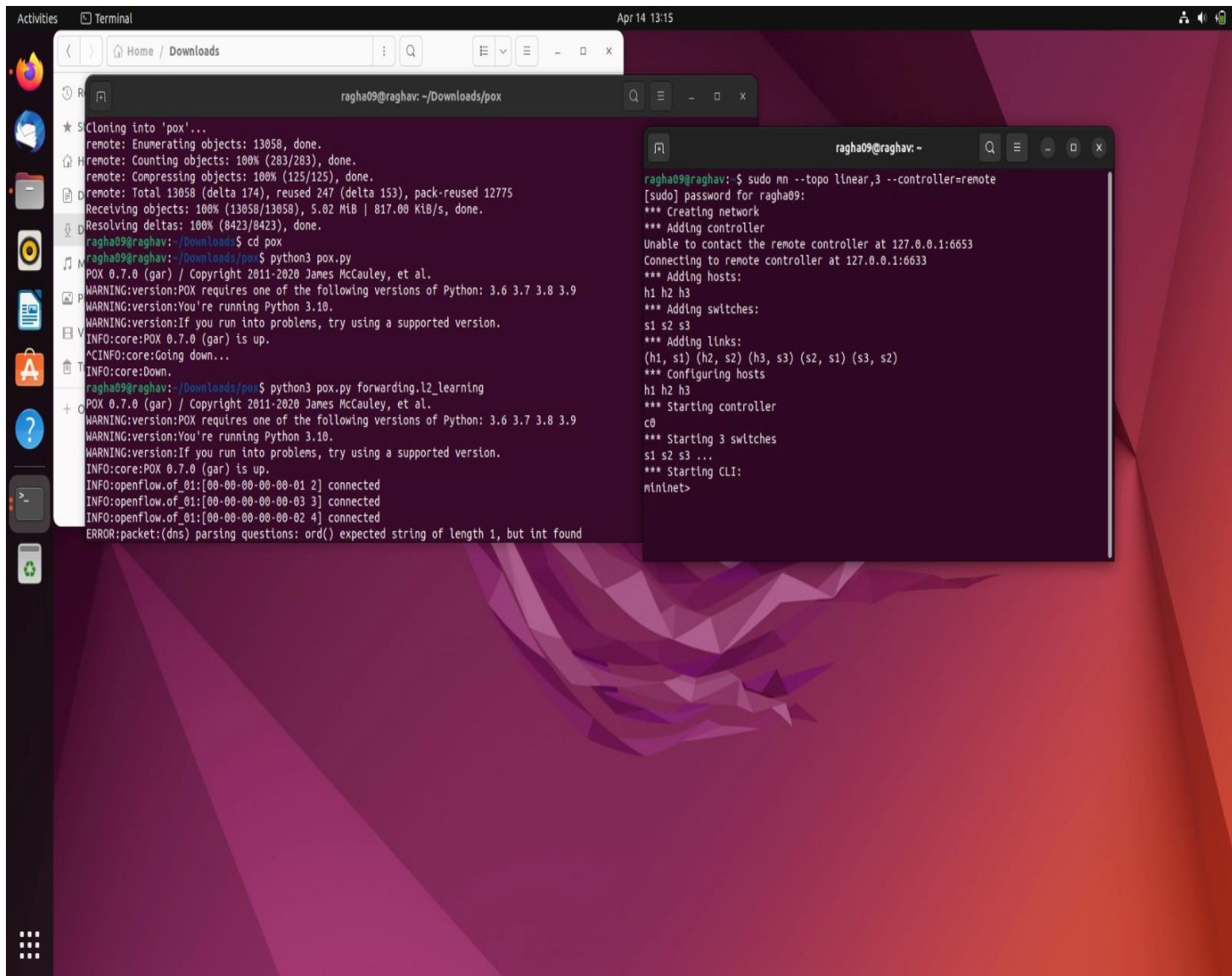
OUTPUT:

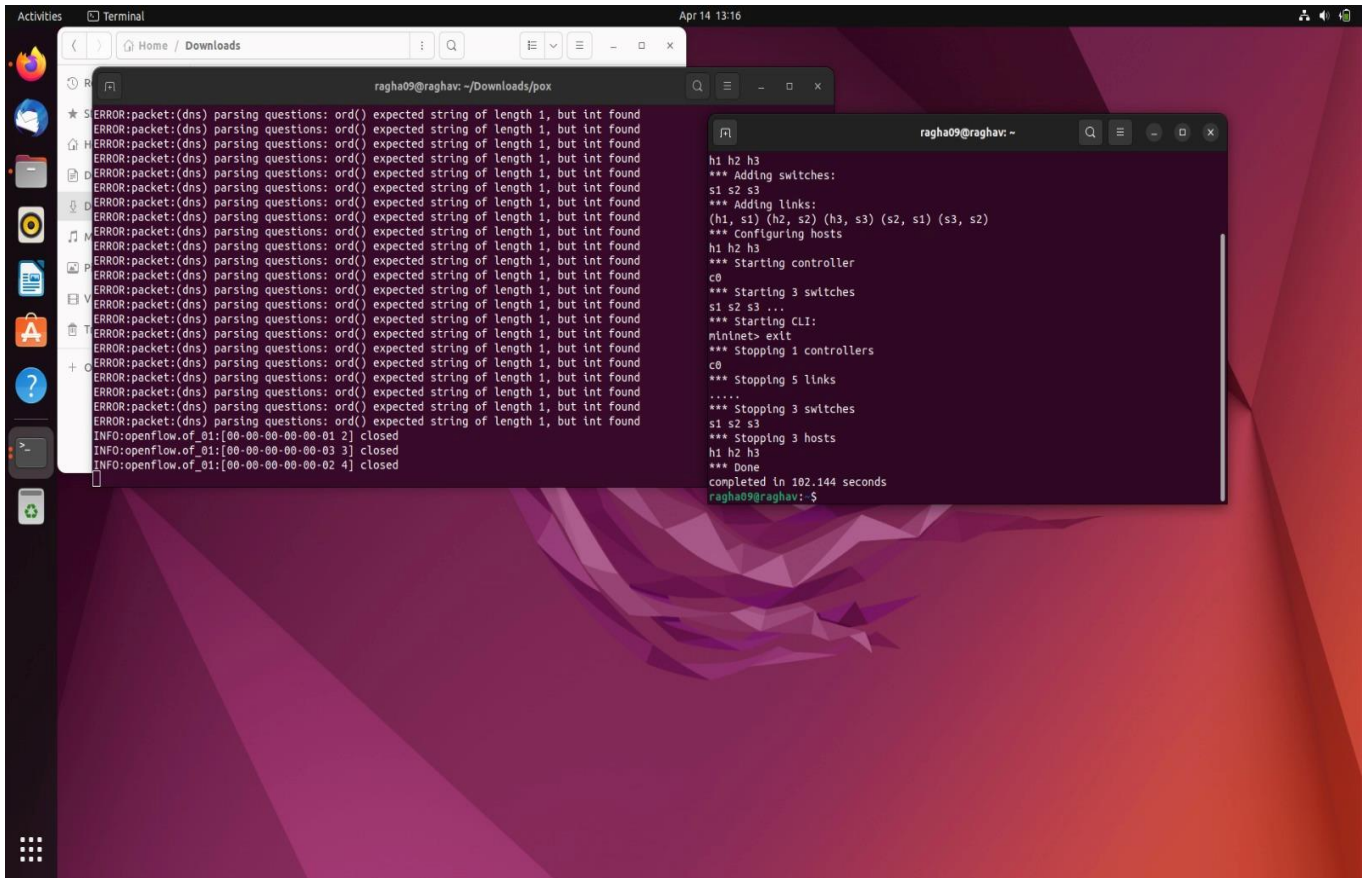


The screenshot shows a Linux desktop with a dark theme. On the left is a vertical dock with icons for Firefox, a messaging app, a file manager, a terminal, a web browser, a question mark, a settings gear, a notepad, a terminal, and a trash can. The main window is a text editor titled "Text Editor" with a subtitle "Mar 21 14:39". The editor shows a file named "pox.py" located at "~/Downloads/pox.py-experimental". The code is a Python script for a malware framework, starting with a shebang and a copyright notice. It includes license information and logic to set up the environment, including checking for PyPy and Python3, and setting environment variables like OPT and FLG. The script ends with a call to a boot function.

```
1#!/bin/sh -
2
3# Copyright 2011-2012 James McCauley
4#
5# Licensed under the Apache License, Version 2.0 (the "License");
6# you may not use this file except in compliance with the License.
7# You may obtain a copy of the License at:
8#
9#     http://www.apache.org/licenses/LICENSE-2.0
10#
11# Unless required by applicable law or agreed to in writing, software
12# distributed under the License is distributed on an "AS IS" BASIS,
13# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14# See the License for the specific language governing permissions and
15# limitations under the License.
16
17# If you have PyPy in a directory called pypy alongside pox.py, we
18# use it.
19# Otherwise, we try to use a Python interpreter called python3, which
20# is a good idea if it's there.
21# We fall back to just "python" and hope that works.
22
23'''true
24#export OPT="-u -O"
25export OPT="-u"
26export FLG=""
27if [ "$(basename $0)" = "debug-pox.py" ]; then
28    export OPT=""
29    export FLG="--debug"
30fi
31
32if [ -x pypy/bin/pypy ]; then
33    exec pypy/bin/pypy $OPT "$@" $FLG "$@"
34fi
35
36if type python3 > /dev/null 2> /dev/null; then
37    exec python3 $OPT "$@" $FLG "$@"
38fi
39
40exec python $OPT "$@" $FLG "$@"
41'''
42
43from pox.boot import boot
44
45if __name__ == '__main__':
46    boot()
```

Python 2 ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS





The screenshot shows a Linux desktop environment with a terminal window and a file manager window. The terminal window displays a series of error messages related to DNS parsing, followed by information about OpenFlow connections. The file manager window shows a directory listing of files and folders.

```
ragha09@raghav: ~/Downloads/pox
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found
INFO:openflow.of_01:[00-00-00-00-00-01 2] closed
INFO:openflow.of_01:[00-00-00-00-00-03 3] closed
INFO:openflow.of_01:[00-00-00-00-00-02 4] closed

ragha09@raghav: ~
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 3 switches
s1 s2 s3
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 102.144 seconds
ragha09@raghav: $
```

CONCLUSION : Thus we have studied open source POX and Floodlight controller and installed controller and implemented program to run custom topology using remote controller like POX and floodlight controller.

ASSIGNMENT NO: 3

AIM: Create a SDN environment on Mininet and configure a switch to provide a firewall functionality using POX controller.

THEORY:

Firewalls:

A Firewall is a necessary part of any security architecture and takes the guesswork out of host level protections and entrusts them to your network security device. Firewalls, and especially Next Generation Firewalls, focus on blocking malware and application-layer attacks, along with an integrated intrusion prevention system (IPS), these Next Generation Firewalls can react quickly and seamlessly to detect and react to outside attacks across the whole network. They can set policies to better defend your network and carry out quick assessments to detect invasive or suspicious activity, like malware, and shut it down.

Need Firewalls

Firewalls, especially Next Generation Firewalls, focus on blocking malware and application-layer attacks. Along with an integrated intrusion prevention system (IPS), these Next Generation Firewalls are able to react quickly and seamlessly to detect and combat attacks across the whole network. Firewalls can act on previously set policies to better protect your network and can carry out quick assessments to detect invasive or suspicious activity, such as malware, and shut it down. By leveraging a firewall for your security infrastructure, you're setting up your network with specific policies to allow or block incoming and outgoing traffic.

SDN firewall on POX controller:

Software Defined Network (SDN) is a revolutionary networking paradigm system in which network control plane is separated from data plane and assigned to a devoted software

program called controller running at a control layer. SDN makes networks wholly controlled through software applications and gives a hope to change the limitations of current networks infrastructures. Since the emergence of SDN, it has introduced a radical change in network architecture which simplified the control of networks, but on the other hand many challenges have arisen. One of the fundamental issues which exposed due to the new architecture of SDN is the security risks. Network firewall is an essential component for securing traffic by enforcing security policies. This study aims to implement some firewall functionalities on SDN through writing some firewall applications that run on the top of the SDN POX controller. The firewall application which is working at layer2, layer3 and layer4 capable of detecting the traffic and enforcing specified rules. Our firewall filters packets based on their headers and matches them against the predefined policies. If there is matching found, the packet is blocked otherwise it is forwarded. We have selected POX a python-based SDN controller and Open vSwitch for the experiment.

CODE:

```
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.link import Intf
from mininet.log import setLogLevel, info
from mininet.node import Controller, OVSKernelSwitch, RemoteController

def myNetwork():

    net = Mininet( topo=None,controller=RemoteController)

    info( '*** Adding controller\n' )
    c0 = net.addController('c0', controller=RemoteController, ip="172.16.87.155")
    info( '*** Add switches\n')
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
```

```
s3 = net.addSwitch('s3')
```

```
info( '*** Add hosts\n')
```

```
h1 = net.addHost('h1', ip="10.0.0.1")
```

```
h2 = net.addHost('h2', ip="10.0.0.2")
```

```
h3 = net.addHost('h3',ip="10.0.0.3")
```

```
h4 = net.addHost('h4',ip="10.0.0.4")
```

```
h5 = net.addHost('h5',ip="10.0.0.5")
```

```
h6 = net.addHost('h6',ip="10.0.0.6")
```

```
info( '*** Add links\n')
```

```
net.addLink(h1, s1)
```

```
net.addLink(h2, s1)
```

```
net.addLink(h3, s1)
```

```
net.addLink(s1, s3)
```

```
net.addLink(s2, s3)
```

```
net.addLink(h4, s2)
```

```
net.addLink(h5, s2)
```

```
net.addLink(h6, s2)
```

```
info( '*** Starting network\n')
```

```
net.start()
```

```
CLI(net)
```

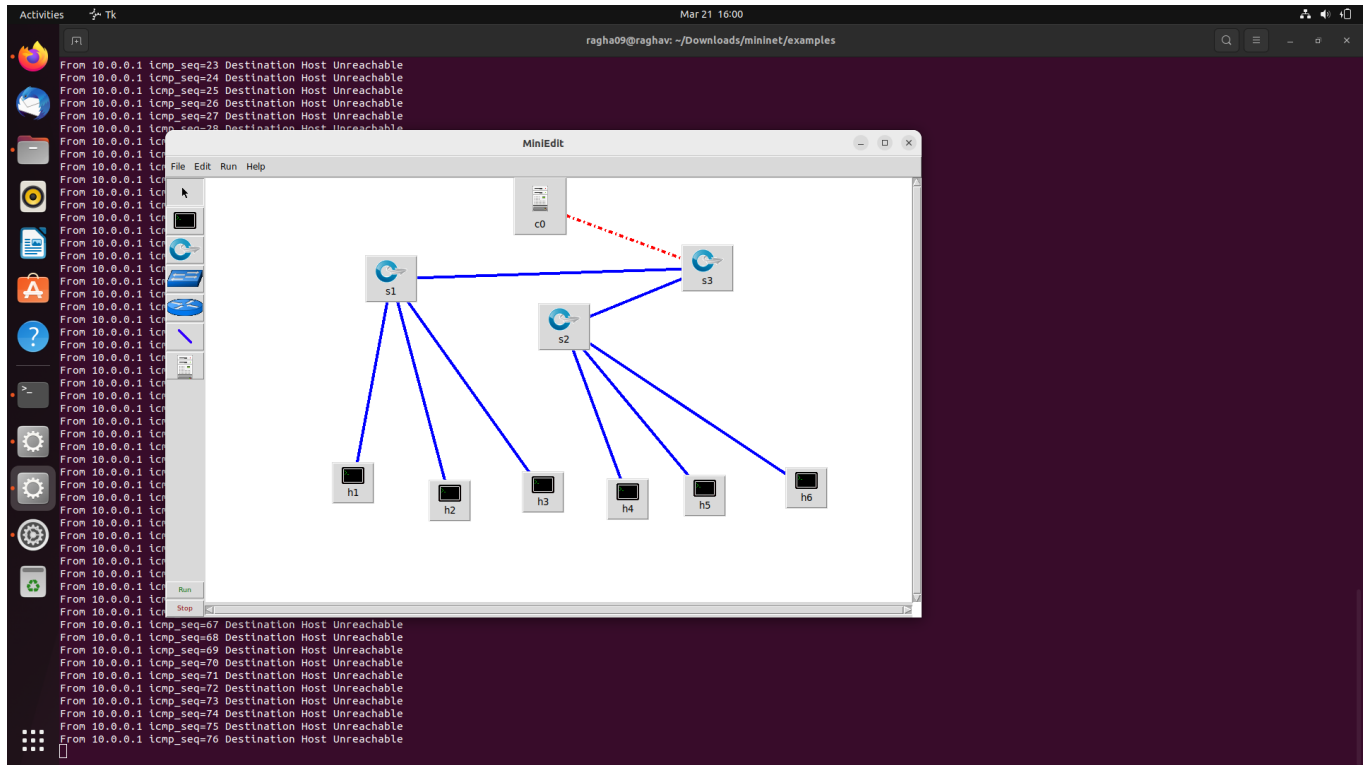
```
net.stop()
```

```
if __name__ == '__main__':
```

```
    setLogLevel( 'info' )
```

```
    myNetwork()
```


OUTPUT:



```
Activities Terminal Mar 21 15:59 ragha09@raghav: ~/Downloads/mininet/examples

rm -f ~/.ssh/mn/*
*** Cleanup complete.
ragha09@raghav:~/Downloads/mininet/examples$ sudo python3 firewall.py
*** 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
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> sh ovs-ofctl add-flow s1 action=normal
mininet> sh ovs-ofctl add-flow s2 action=normal
/bin/sh: 1: ovs-ofctl: not found
mininet> sh ovs-ofctl add-flow s2 action=normal
/bin/sh: 1: ovs-ofctl: not found
mininet> sh ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:04,actions=drop
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 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
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
From 10.0.0.1 icmp_seq=10 Destination Host Unreachable
From 10.0.0.1 icmp_seq=11 Destination Host Unreachable
From 10.0.0.1 icmp_seq=12 Destination Host Unreachable
From 10.0.0.1 icmp_seq=13 Destination Host Unreachable
From 10.0.0.1 icmp_seq=14 Destination Host Unreachable
From 10.0.0.1 icmp_seq=15 Destination Host Unreachable
From 10.0.0.1 icmp_seq=16 Destination Host Unreachable
From 10.0.0.1 icmp_seq=17 Destination Host Unreachable
From 10.0.0.1 icmp_seq=18 Destination Host Unreachable
From 10.0.0.1 icmp_seq=19 Destination Host Unreachable
From 10.0.0.1 icmp_seq=20 Destination Host Unreachable
From 10.0.0.1 icmp_seq=21 Destination Host Unreachable
From 10.0.0.1 icmp_seq=22 Destination Host Unreachable
From 10.0.0.1 icmp_seq=23 Destination Host Unreachable
From 10.0.0.1 icmp_seq=24 Destination Host Unreachable
From 10.0.0.1 icmp_seq=25 Destination Host Unreachable
From 10.0.0.1 icmp_seq=26 Destination Host Unreachable
From 10.0.0.1 icmp_seq=27 Destination Host Unreachable
From 10.0.0.1 icmp_seq=28 Destination Host Unreachable
```

```
Activities Terminal Mar 21 15:59
GitHub - noxrepo/pox: T... Firefox Privacy Notice - GitHub - mininet/mininet: simple mininet custom to Building firewall as a serv Exception: Error creating x +
ragha09@raghav: ~/Downloads/mininet/examples
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> sh ovs-ofctl add-flow s1 action=normal
mininet> sh ovs-ofctl add-flow s2 action=normal
/bin/sh: 1: ovs-ofctl: not found
mininet> sh ovs-ofctl add-flow s2 action=normal
/bin/sh: 1: ovs-ofctl: not found
mininet> sh ovs-ofctl add-flow s3 d1_src=00:00:00:00:01,d1_dst=00:00:00:00:04,actions=drop
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 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
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
From 10.0.0.1 icmp_seq=10 Destination Host Unreachable
```

```
Activities Terminal Mar 21 16:07
ragha09@raghav: ~/Downloads/mininet/examples
*** 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.
ragha09@raghav: ~/Downloads/mininet/examples$ sudo python3 firewall.py
*** 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
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> sh ovs-ofctl add-flow s1 action=normal
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=2.16 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.110 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.108 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3049ms
rtt min/avg/max/ndev = 0.081/0.014/2.158/0.891 ms
mininet> sh ovs-ofctl add-flow s1 action=drop
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
From 10.0.0.1 icmp_seq=17 Destination Host Unreachable
From 10.0.0.1 icmp_seq=18 Destination Host Unreachable
From 10.0.0.1 icmp_seq=19 Destination Host Unreachable
From 10.0.0.1 icmp_seq=20 Destination Host Unreachable
From 10.0.0.1 icmp_seq=21 Destination Host Unreachable
From 10.0.0.1 icmp_seq=22 Destination Host Unreachable
```

CONCLUSION : Thus we have studied and Created a SDN environment on Mininet and configured a switch to provide a firewall functionality using POX controller.

ASSIGNMENT NO: 4

AIM: Using Mininet as an Emulator and POX controller, build your own internet router. Write simple router with a static routing table. The router will receive raw Ethernet frames and process the packet forwarding them to correct outgoing interface. You must check the Ethernet frames are received and the forwarding logic is created so packets go to the correct interface

THEORY:

OpenFlow Controller

Modern Internet switches make millions of decisions per second about whether or not to forward an incoming packet, to what set of output ports it should be forwarded, and what header fields in the packet may need to be modified, added, or removed. This is a very complex task.

The OpenFlow control plane (controller) differs from the legacy control plane in three key ways.

- 1) It can program different data plane elements with a common, standard language such as OpenFlow.
- 2) It exists on a separate hardware device than the forwarding plane, unlike traditional switches, where the control plane and data plane are instantiated in the same physical box. This separation is made possible because the controller can program the data plane elements remotely over the Internet.
- 3) The controller can program multiple data plane elements from a single control plane instance.

The OpenFlow controller is responsible for programming all the packet-matching and forwarding rules in the switch. Whereas a traditional router would run routing algorithms to determine how to program its forwarding table, that function or an equivalent replacement to it is now performed by the controller.

Any changes that result in recomputing routes will be programmed onto the switch by the controller.

Routing Table –

A routing table is a set of rules, often viewed in table format, that's used to determine where data packets traveling over an Internet Protocol (IP) network will be directed. This table is usually stored inside the Random Access Memory of forwarding devices, such as routers and network switches.

Static Routing –

static routing is a form of routing that occurs when a router uses a manually-configured routing entry, rather than information from dynamic routing traffic. In many cases, static routes are manually configured by a network administrator by adding in entries into a routing table, though this may not always be the case

Basic Packet Forwarding

After receiving the packet, switch performs the functions as shown in figure 2.9 The switch starts by performing a table lookup in the first flow table and may perform table lookup in other flow tables based on pipeline processing.

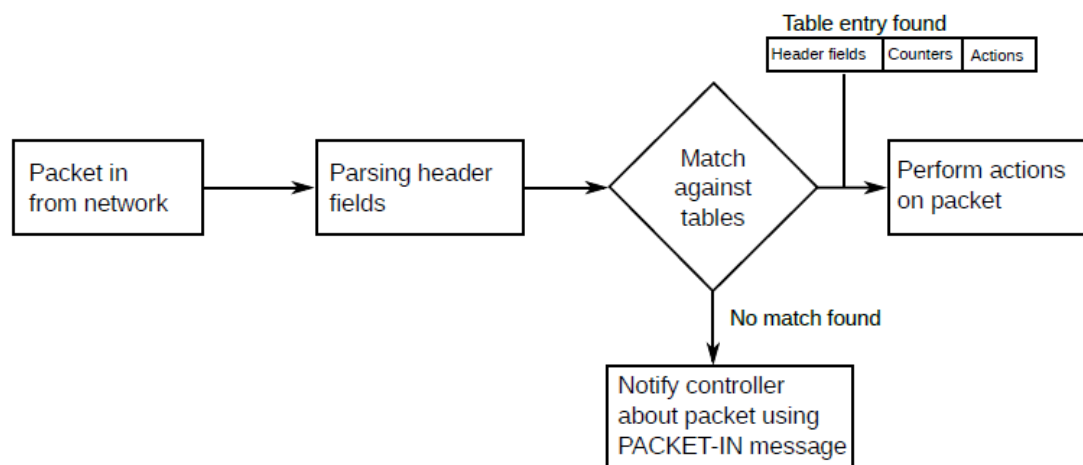


Figure2.9 Basic Packet forwarding with OpenFlow Switch

There can be two types of flow entry

- 1) **Exact match** – when all the parameters of the flow entry are uniquely defined and there is no wild card field. They always have highest priority.
- 2) **Wildcard match** - When any one parameter of the Flow entry is wildcard then it is known as wild card flow entry. A wild card flow entry can match more than one type of packets

If a packet satisfy the matching condition of both exact and wildcard flow entries than it will hit the exact flow entry.

When a switch receives a packet, it parses the packet header, which is matched against the flow table. If

a flow table entry is found where the header field wildcard matches the header, the entry is considered. If several such entries are found, packets are matched based on prioritization, i.e., the most specific entry or the wildcard with the highest priority is selected. Then, the switch updates the counters of that flow table entry. Finally, the switch performs the actions specified by the flow table entry on the packet, e.g., the switch forwards the packet to a port.

Otherwise, if no flow table entry matches the packet header, the switch generally notifies its controller about the packet, which is buffered when the switch is capable of buffering. To that end, it encapsulates either the unbuffered packet or the first bytes of the buffered packet using a PACKET-IN message and sends it to the controller; it is common to encapsulate the packet header and the number of bytes defaults to 128. The controller that receives the PACKET-IN notification identifies the correct action for the packet and installs one or more appropriate entries in the requesting switch. Buffered packets are then forwarded according to the rules; this is triggered by setting the buffer ID in the flow insertion message or in explicit PACKET-OUT messages. Most commonly, the controller sets up the whole path for the packet in the network by modifying the flow tables of all switches on the path.

CODE:

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node
from mininet.log import setLogLevel, info
from mininet.cli import CLI
class LinuxRouter( Node ):
    """A Node with IP forwarding enabled.
    Means that every packet that is in this node, communicate freely with its interfaces."""
    def config( self, **params ):
        super( LinuxRouter, self ).config( **params )
        self.cmd( 'sysctl net.ipv4.ip_forward=1' )
    def terminate( self ):
        self.cmd( 'sysctl net.ipv4.ip_forward=0' )
        super( LinuxRouter, self ).terminate()
```

```

class NetworkTopo( Topo ):
    def build( self, **_opts ):
        r1=self.addNode("r1",cls=LinuxRouter,ip=None)
        r2=self.addNode("r2",cls=LinuxRouter,ip=None)
        r3=self.addNode("r3",cls=LinuxRouter,ip=None)

        self.addLink(r1,r2,params1={ 'ip' : '10.0.0.1/24' },params2={ 'ip' : '10.0.0.2/24' })
        self.addLink(r2,r3,params1={ 'ip' : '10.0.1.1/24' },params2={ 'ip' : '10.0.1.2/24' })

topo = NetworkTopo()
net = Mininet( topo=topo )
net.start()
#ip route add ipA via ipB dev INTERFACE
#every packet going to ipA must first go to ipB using INTERFACE
net["r1"].cmd("ip route add 10.0.1.2 via 10.0.0.2 dev r1-eth0")
net["r3"].cmd("ip route add 10.0.0.1 via 10.0.1.1 dev r3-eth0")
#this command is just to r3 ping r2 work, because it will use the correct ip
net["r3"].cmd("ip route add 10.0.0.2 via 10.0.1.1 dev r3-eth0")
net.pingAll()
CLI( net )
net.stop()

```

OUTPUT

```
Activities Terminal Mar 21 14:42 ragha09@raghav: ~  
*** 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> help  
  
Documented commands (type help <topic>):  
=====EOF gterm lperfudp nodes pingpair py switch xterm  
dpctl help link noecho pingpairfull quit time  
dump intf links pingall ports sh wait  
exit lperf net pingallfull px source x  
=====
```

You may also send a command to a node using:
<node> command [args]
For example:
mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
mininet> xterm h2

```
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> dump  
<Host h1: h1-eth0:10.0.0.1 pid=3570>  
<Host h2: h2-eth0:10.0.0.2 pid=3572>  
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3577>  
<Controller c0: 127.0.0.1:6653 pid=3563>  
mininet>
```



```
Activities Terminal Mar 21 15:14
ragha09@raghav: ~/Downloads

s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 64.189 seconds
ragha09@raghav:~/Downloads$ ls
mininet mininet-master.zip pox-gar-experimental pox-gar-experimental.zip simplrouter.py
ragha09@raghav:~/Downloads$ python3 simplrouter.py
*** Mininet must run as root.
ragha09@raghav:~/Downloads$ sudo python3 simplrouter.py
*** Ping: testing ping reachability
r1 -> r2 r3
r2 -> r1 r3
r3 -> r1 r2
*** Results: 0% dropped (6/6 received)
mininet> r1 ping r1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.120 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.084 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.098 ms
^C
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2061ms
rtt min/avg/max/mdev = 0.084/0.100/0.120/0.014 ms
mininet> r1 ping r2
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=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.133 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.122 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.209 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.061 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.066 ms
^C
--- 10.0.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10257ms
rtt min/avg/max/mdev = 0.039/0.080/0.209/0.049 ms
mininet> r1 ping r3
PING 10.0.1.2 (10.0.1.2) 56(124) bytes of data.
^C
--- 10.0.1.2 ping statistics ---
15 packets transmitted, 0 received, 100% packet loss, time 15024ms

mininet> r3 ping r1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=63 time=0.057 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=63 time=0.119 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=63 time=0.045 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=63 time=0.057 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=63 time=0.078 ms
```

CONCLUSION : Thus we have studied and Using Mininet as an Emulator and POX controller, build our own internet router. The router receives raw Ethernet frames and process the packet forwarding them to correct outgoing interface.

ASSIGNMENT NO. 5

AIM: Study in details Cloud seeds automates IAAS using SDN and a high-performance network from Juniper SDN Framework.

THEORY:

Juniper Networks, Inc. is an American multinational corporation. The company develops and markets networking products, including routers, switches, network management software, network security products, and software-defined networking technology.

To realize its vision of a new class of automated IaaS services, CloudSeeds needed to create a new network platform, one that provided seamless, high-performance routing, switching, SDN, and security.

CloudSeeds has developed A.C.R.E. (Advanced Cloud Resource Elements), which uses standard prebuilt cloud components to create a highly dynamic Infrastructure-as-a-Service (IaaS) layer combining deep automation and complete virtualization to create turnkey solutions for its customers.

To realize its vision of a new class of automated services, CloudSeeds needed to create a new network platform, providing seamless, high-performance routing, switching and intense security measures. Most critically, it needed an open architecture that could be controlled and configured by software. CloudSeeds, based in Hamburg, Germany, was founded in 2013 to help establish virtualized infrastructures and IT services to companies that are growing rapidly and need scalability for their business objectives.

Kevin Fibich, founder and managing director at Cloud Seeds, had worked in a number of operational IT environments and recognized the need for a new class of software-defined IT services based on a highly scalable, flexible, and automated platform. CloudSeeds developed its new approach, known as A.C.R.E. (Advanced Cloud Resource Elements), using standard pre-built cloud components creating a

highly dynamic IaaS (Infrastructure-as-a-Service) layer-combining deep automation and complete virtualization to create turnkey solutions for its customers. The platform delivers new data

center and IT infrastructure that can scale as its customers' needs change and grow, without customers having to worry about the day-to-day management of IT hardware. CloudSeeds operates dedicated cloud setups for their customers-managed and operated on its premises-as data security is a key topic in the German commercial enterprise sector with customers pressing for data sovereignty.

Kevin Fibich said, "Our customers can sometimes be overwhelmed by their own rapid success and as a result they need to rapidly deploy new infrastructure. We call this a 'friendly DDoS' (distributed denial of service) attack, as their existing network may be overwhelmed by legitimate customer demand. It is a luxury problem for them to have, and our automated software-defined services help them overcome it."

CloudSeeds is enjoying high growth as it takes on new customers, attracted by the business flexibility and scale it offers, and is currently expanding its team.

Business Challenge To help create virtualized infrastructures and provide IT services to its customers, CloudSeeds needed an open architecture that could be controlled and configured by third-party software. It also needed a network platform that could scale ahead of its requirements, and provide a highly resilient service to enable its customers' businesses to grow.

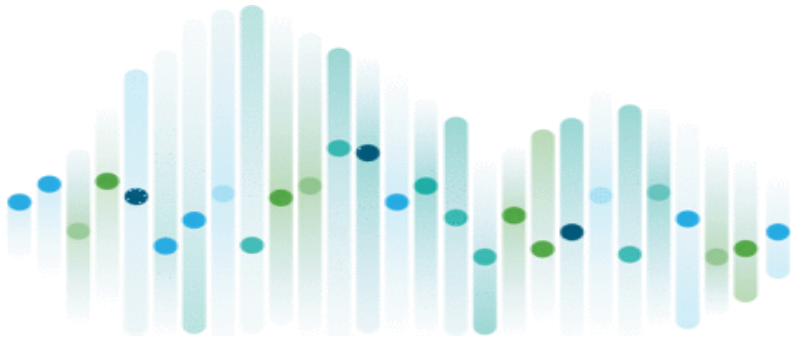
Technology Solution CloudSeeds used a number of Juniper systems to create its new software-defined A.C.R.E. platform. These include the QFX5100 Ethernet Switch, MX80 3D Universal Edge Router, SRX1400 and vSRX Services Gateways, and Contrail Networking to orchestrate the network and create different virtual networks.

Business Results The new network has made CloudSeeds' vision of zero-touch provisioning a reality. At the same time, the Juniper solution allows the rapid uptake of IaaS services by CloudSeeds' customers, making their businesses far more agile and responsive."

PRODUCT FAMILY

Technology	Product families
Routing	<ul style="list-style-type: none">• T-series: Multichassis IP/MPLS Core Routers• MX Series: Edge routers• M Series: Combined IP/MPLS edge routers• PTX Series: Packet transport routers• ACX Series: Universal access routers
Switching	<ul style="list-style-type: none">• EX Series: Enterprise Ethernet switches• WLAN Products: Controllers, access points and software• QFX Series: Datacenter switches
Security	<ul style="list-style-type: none">• SRX Series: Security products for data centers and branch locations
Software	<ul style="list-style-type: none">• Junos Operating System• Junos Space: Service Oriented Architecture development environment for network applications• Contrail: Brand of software defined networking software and networking controllers• Marvis: Mist's AI Network Assistant that is also compatible with Juniper's switches through its Wired Assurance feature.
WLAN	<ul style="list-style-type: none">• AP41: The most popular enterprise-grade Access Point available through Mist. Tailored for WiFi, BLE, and IoT.• AP43: An upgraded AP41 with WiFi 6• AP61: A long-range access point ideal for outdoor use like college campuses

Contrail



Contrail offers intelligent networking, increased security, and advanced analytics—all with automation—for multicloud and telco cloud.

MX Series Routers



A robust portfolio of SDN-enabled routers, the MX Series provides industry-leading system capacity, density, security, and performance with unparalleled longevity.

QFX Series Switches

QFX network switches deliver industry-leading throughput and scalability, a comprehensive routing stack, the open programmability of Junos OS, and the broadest set of EVPN-VXLAN and IP fabric capabilities. Find your solution for data center spine and leaf switches, campus distribution and core, or data center gateway and interconnect.



CONCLUSION : Thus we have studied in details Cloud seeds automates IAAS using SDN and a high-performance network from Juniper SDN Framework.