

410252: Software Defined Network
Group A
Assignment 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.

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

The Mininet VM is meant to speed up Mininet installation, plus make it easy to run on non-Linux platforms. The VM works on Windows, Mac, and Linux, through VMware, VirtualBox, QEMU and KVM.

After downloading the VM, you'll run a few steps to customize it for your setup. This won't take long.

VM Setup:

Download the Mininet VM

Download the Mininet VM from <https://github.com/mininet/mininet/wiki/Mininet-VM-Images> . The VM comes out to 1GB compressed and ~2GB uncompressed. It is an OVF (Open Virtualization Format) virtual machine image which can be imported by most virtual machine monitors.

Download and install a virtualization program such as: VMware Workstation for Windows or Linux, VMware Fusion for Mac, VirtualBox (**free!**, GPL) for any platform, or qemu (**free!**, GPL) for Linux. If you already have VMware, we find that it runs Mininet somewhat faster

than VirtualBox. However, VirtualBox is free to download and distribute, which is a definite advantage!

Boot VM:

Add the VM and start it up, in the virtualization program of your choice:

VirtualBox:

1. Usually you can just double-click on the .ovf file and import it.
2. If you get errors importing the .ovf file, you can simply create a new VM of the appropriate type (e.g. Linux, Ubuntu 64-bit) and use the .vmdk file as the virtual hard disk for the new VM.
3. Select “settings,” and add an additional *host-only network adapter* that you can use log in to the VM image. Start the VM.
4. For more information on setting up networking in VirtualBox, you may wish to check out these VirtualBox specific instructions

VMware: Import the OVF file, then start the VM. VMware may ask you to install VMware tools on the VM - if it asks, decline. Everything graphical in the tutorial is done via X forwarding through SSH (in fact, the VM doesn’t have a desktop manager installed), so the VMware tools are unnecessary unless you wish to install an X11/Gnome/etc. environment in your VM.

Qemu/KVM:

For Qemu, something like the following should work:

```
qemu-system-x86_64 -m 2048 mininet-vm-disk1.vmdk -net nic,model=virtio -net user,net=192.168.101.0/24,hostfwd=tcp::8022-:22
```

For KVM:

```
sudo qemu-system-x86_64 -machine accel=kvm -m 2048 mininet-vm-disk1.vmdk -net nic,model=virtio -net user,net=192.168.101.0/24,hostfwd=tcp::8022-:22
```

The above commands will set up ssh forwarding from the VM to host port 8022. For a 32-bit VM image, use qemu-system-i386.

Parallels: Use Parallels Transporter to convert the .vmdk file to an .hdd image that Parallels can use, and then create a new VM using that .hdd image as its virtual drive.

Start the VM.

Log in to VM

Log in to the VM, using the following name and password:

```
mininet-vm login: mininet
```

```
Password: mininet
```

(some older VM images may use openflow/openflow instead) The root account is not enabled for login; you can use sudo to run a command with superuser privileges.
SSH into VM

First, find the VM's IP address, which for VMware is probably in the range 192.168.x.y.

In the VM console:

```
ifconfig eth0
```

Note: VirtualBox users who have set up a host-only network on eth1 should use

```
sudo dhclient eth1 # make sure that eth1 has an IP  
address ifconfig eth1
```

You may want to add the address to your host PC's /etc/hosts file to be able to SSH in by name, if it's Unix-like. For example, add a line like this for OS X:

```
192.168.x.y mininet-vm
```

where 192.168.x.y is replaced by the VM's IP address.

SSH into the VM. We assume the VM is running locally, and that the additional precautions of ssh -X are unnecessary. ssh -Y also has no authentication timeout by default.

```
ssh -Y mininet@mininet-vm
```

If you're running the VM under QEMU/KVM with -net user and the hostfwd option as recommended above, the VM IP address is irrelevant. Instead, you tell SSH to connect to port 8022 on the host:

```
ssh -Y -p 8022 mininet@localhost
```

Optional VM Customization

These commands are optional, and may be useful for your setup:

Set up SSH auto-login

These steps let you log in via ssh without needing to enter a password. If you use the console from your virtualization software natively, then this step isn't needed. Check for `~/.ssh/id_rsa` or `~/.ssh/id_dsa`. If you can't find either of these files, then you'll want to generate an SSH key. On a unix-like system (OS X or Linux - you'll need other instructions for Windows) - on the host, not the VM:

```
ssh-keygen -t rsa
```

To speed up future SSH connections, add your host's public key to the new VM. Also on the host, not the VM:

```
scp ~/.ssh/id_rsa.pub openflow@openflow:~/
```

Now, on the VM (SSH in first):

```
cd ~/ && mkdir -p .ssh && chmod 700 .ssh && cd .ssh && touch authorized_keys2 && chmod 600  
authorized_k
```

Conclusion:

Thus, we learnt Mininet network emulation environment with the help of Virtual box and 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.

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

Using the POX SDN controller 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.

For example, the *forwarding.l2_learning* component is in the `~/pox/pox/forwarding` directory, as seen below:

```
$ cd pox/pox
$ ls
boot.py datapaths info log proto tk.py
boot.pyc forwarding __init__.py messenger py.py topology
core.py help.py __init__.pyc misc py.pyc web
core.pyc host_tracker lib openflow samples
$ cd forwarding
$ ls
```

```
hub.py l2_flowvisor.py l2_nx.py l3_learning.py  
__init__.py l2_learning.py l2_nx_self_learning.py l3_learning.pyc  
__init__.pyc l2_multi.py l2_pairs.py topo_proactive.py
```

Programming for POX:

The general purpose of all SDN controllers, including POX, is to allow users to write their own applications that use the controller as an intermediary — or abstraction layer — between network applications and the network equipment.

To learn how to write applications for POX, developers may study the stock POX components as examples that show how to write their own components or they may review the POX API documentation to learn how to write networking applications that use the POX Python API.

Installing POX:

POX comes already installed on the Mininet 2.2 VM image. In this tutorial, we will use the VM image. See my previous post about setting up the Mininet 2.2 VM.

If you wish to install Mininet and POX on your own Linux system — either hardware or a virtual machine — you may use the Mininet install script, which also installs the POX controller when it installs Mininet.

If you wish to install POX by itself, follow the POX installation instructions from the POX documentation.

Running POX:

Start POX by running the *pox.py* program, and specifying the POX components to use. For example, to run POX so it makes the switches it controls emulate the behavior of Ethernet learning switches, run the command:

```
mininet@mininet-vm:~$ sudo ~/pox/pox.py forwarding.l2_learning
```

The POX Console:

The *POX console* is the terminal session from which we run the POX controller. After POX starts, it will display log information and may optionally display an interactive Python command line, if POX is started with the *py* component.

Quit POX:

To quit POX, use the *control-C* key combination in the POX console.

Floodlight Controller:

The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller and intended to run with standard JDK tools and ant.

Highlights:

- Easy to set up with minimal dependencies
- Supports a broad range of virtual and physical OpenFlow switches
- Can handle mixed OpenFlow and non- OpenFlow networks.
- It can manage multiple “islands” of OpenFlow hardware switches
- Designed to be high-performance Installation:

Floodlight master has been updated (on 04/30/16) to Java 8.

Prerequisites:

- Java development kit
- JDK 8 for Floodlight master and above
- JDK 7 for Floodlight v1.2 and below
- Ant to build
- Python development package

-To download dependencies for Floodlight master and above:

```
sudo apt-get install build-essential ant maven python-dev
```

-To download dependencies for Floodlight v1.2 and below:

```
sudo apt-get install build-essential openjdk-7-jdk ant maven python-dev eclipse
```

- Download And Build:

Floodlight is simple to download from Github and install. The “git clone” step below uses the master version of Floodlight. Type the below command in SDNHub Terminal.

```
$ git clone git://github.com/floodlight/floodlight.git
```

If You are having JDK 7 use a specific version, specify the version branch in the “git clone”

```
$ git clone -b v1.2 git://github.com/floodlight/floodlight.git
```

After installation complete follow the steps to build floodlight controller.

```
$ cd floodlight
```

```
$ git submodule init
```

```
$ git submodule update
```

```
$ ant
```

After build completed successful. Make a directory floodlight and set root permission for it.

```
$ sudo mkdir /var/lib/floodlight
```

```
$ sudo chmod 777 /var/lib/floodlight
```

Running Floodlight in the Terminal:

Assuming java is in your path, you can directly run the floodlight.jar file produced by ant from within the floodlight directory:

```
$ java -jar target/floodlight.jar
```

To Create mininet topology with floodlight remote controller, another terminal run the below command,

```
$ cd floodlight
```

```
$ sudo mn --controller=remote,ip=127.0.0.1,port=6653 --topo=single,3
```

We created single topology with 3 host 1 switch and one controller. This can be view in the following url. Go to web browser and type,

<http://127.0.0.1:8080/ui/index.html>

In topology tab we can see our topology. Switches can be view in switches tab. Hosts are listed in Hosts tab. we can view host separately by xterm in mininet

```
mininet>xterm h1 h2
```

Ping each other host by ping command with host IP In node h1 \$ ping 10.0.0.2
In node h2 \$ ping 10.0.0.1

Conclusion:

Thus, we learnt Symmetric and Asymmetric implementation of Particle Swarm Optimization for Traveling Salesman Problem.

Assignment No.: 3

Aim:

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

Objectives:

1. To learn SDN environment on Mininet .
2. To learn and configure a switch to provide a firewall functionality using POX controller.

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

1. **Setup Mininet:** The easiest way to get started is to use a pre-packaged Mininet/Ubuntu Virtual Machine (VM). This VM contains the Mininet itself, all OpenFlow binaries and controllers, and tweaks to the kernel configuration to support larger Mininet networks.

- a. **Download Virtualization System:**

You can use any virtualization system of your choice, but we recommend installing VirtualBox. It's free and runs on Windows, Linux and OS X. VirtualBox (Recommended): <https://www.virtualbox.org/wiki/Downloads>

- b. **Download Mininet VM:**

We recommend using the latest Mininet image, which is the package file we provide. You can also go to www.mininet.org for more information. It comes with the latest version of Mininet and two OpenFlow Controller (POX and Pyretic). The download will take some time. It's ~800MB, compressed.

- c. **Setup Virtual Machine:**

- Start VirtualBox
- Select File> Import Appliance and select the .ova file
- Press the "Import" button.
 - This will unpack and import the VM in your local machine. It will take a while, as the unpacked image is about 3 GB.

d. Boot VM:

Now, you are ready to start your VM. Press the “Start” arrow icon or double-click your VM within the VirtualBox Windows.

In the VM console window, log in with the user name and password for your VM. The username and password for this VM are:

- User name – **mininet**
- Password – **mininet**

Note that this user is a sudoer, so you can execute commands with root permissions by typing sudo command, where command is the command you wish to execute with root permission.

- 2. Setup POX and POXDesk:** POX should be installed and run on another Virtual Machine. You need to download and install an Ubuntu system (we recommend version 14.04) on that machine. POXDesk is an extensible web-based GUI for POX, which makes it more convenient to monitor the status of the network and switches. This is an optional add-on; you can decide whether to use it. It helps you quickly visualize the topology you create.

a. Download POX:

Go to the root directory and pull the POX repository:

Git clone <https://github.com/noxrepo/pox>

b. Download and Install POXDesk:

Follow these commands to install POXDesk:

```
cd pox/ext
git clone https://github.com/MurphyMcAfee/poxdesk
cd poxdesk
wget http://downloads.sourceforge.net/qooxdoo/qooxdoo-2.0.2-sdk.zip
unzip qooxdoo-2.0.2-sdk.zip
mv qooxdoo-2.0.2-sdk qx
cd poxdesk
./ generate.py
```

When mininet is started and switches are connected with the POX controller, you can access the manage page of POXDesk at:

127.0.0.1:8000/poxdesk

3. Mininet:

Mininet is a powerful network emulation tool. It creates a realistic virtual network, running real kernel, switch and application code on a single machine. You can easily interact with your network using Command Line Interface (CLI). Notice that to test connectivity, you need to run a POX controller that has switch functionality.

Mininet Basic:

Display Mininet Command Line Interface (CLI) commands:

```
mininet > help
```

Display nodes:

```
mininet > nodes
```

If the first string of the CLI command is a host, switch or controller name, the command is executed on that node. For instance, to show the interface of host h1:

```
mininet > h1 ifconfig
```

Test connectivity between hosts. For example, test the connectivity between h1 and h2:

```
mininet > h1 ping h2
```

Alternatively, you can test the connectivity between all hosts by typing:

```
mininet > pingall
```

Exit mininet:

```
mininet > exit
```

Clean up:

```
$ sudo mn -c
```

Mininet Python API:

A more powerful way to construct the network topology is to use Mininet Python APIs. 'mininet.topo.Topo' is the class that defines the network topology. We can define a child class of 'mininet.topo.Topo' and initialize the network topology with the help of the class methods.

addHost(name, opts): add a host to the network with the name being 'name' parameter. For our case, hosts are named as h1, h2, h3, and h4. 'opts' is a dictionary that includes the parameters for the host.

addSwitch(name, opts): add an Openflow vSwitch to the network with the name being 'name' parameter. For our case, switches are named as s1,s2,s3,s4. 'opts' is a dictionary that includes the parameters for the switch.

addLink(node1, node2, port1, port2, opts): add a bidirectional link between port1 of node1 and port2 of nodes. 'opts' is a dictionary that includes the parameter for the link. 'bw' is one of the parameters that stands for bandwidth, and unit is Mbps.

Mininet(topo,link,controller) is the initialization function of the Mininet network emulation environment. Using the defined topology, we can set the remote controller and the communication protocol between data plane and control plane.

Conclusion:

Thus we learnt a SDN environment on Mininet and configure 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.

Objectives:

1. To learn Using Mininet as an Emulator and POX controller, build your own internet router

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

Step 1: Launch a Linux terminal by holding the Ctrl + Alt + T keys or by clicking on the Linux terminal icon.



Figure 3. Shortcut to open a Linux terminal.

The Linux terminal is a program that opens a window and permits you to interact with a command-line interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system for execution.

Step 2: To start a minimal topology, enter the command shown below. When prompted for a password, type **password** as you type it.

```
sdn@admin: ~  
File Actions Edit View Help  
sdn@admin: ~  
sdn@admin:~$ sudo mn  
[sudo] password for sdn:  
*** 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:  
containernet> |
```

Figure 4. Starting Mininet using the CLI.

The above command starts Mininet with a minimal topology, which consists of a switch connected to two hosts as shown below.

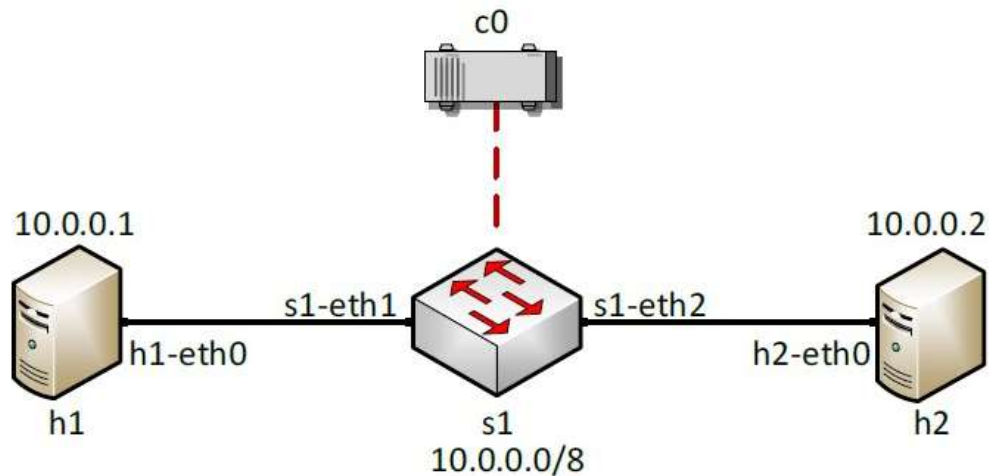


Figure 5. Mininet's default minimal topology.

When issuing the `sudo mn` command, Mininet initializes the topology and launches its command line interface which looks like this:

```
mininet>
```

Step 3: To display the list of Mininet CLI commands and examples on their usage, type the following command:

```
sdn@admin: ~
File Actions Edit View Help
sdn@admin: ~
containernet> help
Documented commands (type help <topic>):
=====
EOF    gterm  iperfudp  nodes    pingpair  py    switch
dpctl  help   link      noecho   pingpairfull  quit  time
dump   intfs  links     pingall  ports     sh    x
exit   iperf  net       pingallfull  px      source xterm

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

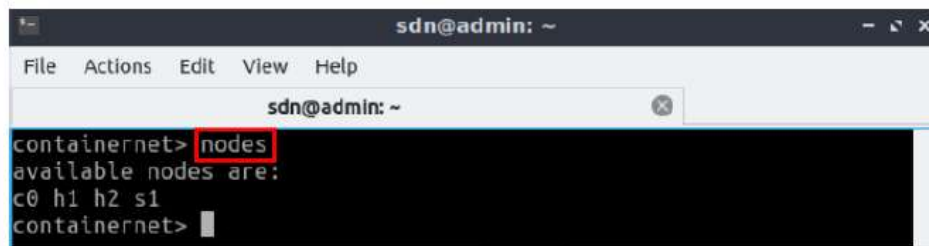
The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
mininet> h2 ping h3
should work.

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
containernet>
```

Figure 6. Mininet's `help` command.

Step 4: To display the available nodes, type the following command:

```
nodes
```

A screenshot of a terminal window titled 'sdn@admin: ~'. The terminal shows the command 'nodes' being entered at the 'containernetwork>' prompt. The output is 'available nodes are: c0 h1 h2 s1'. The word 'nodes' in the command is highlighted with a red box.

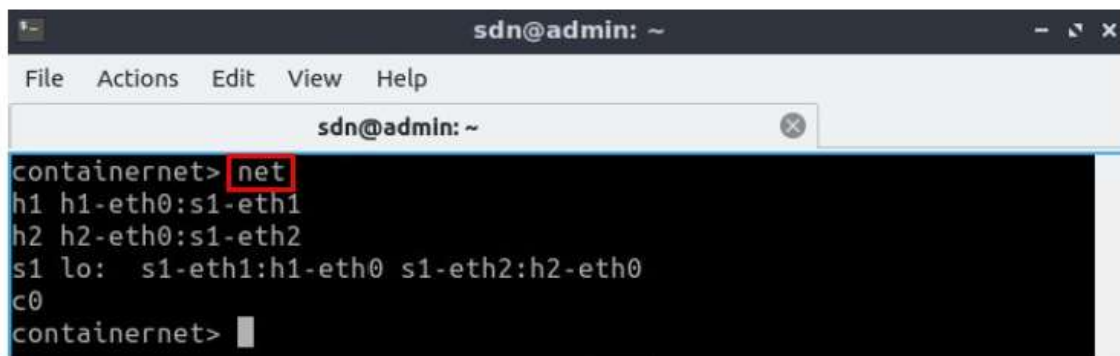
```
sdn@admin: ~  
File Actions Edit View Help  
sdn@admin: ~  
containernetwork> nodes  
available nodes are:  
c0 h1 h2 s1  
containernetwork> 
```

Figure 7. Mininet's `nodes` command.

The output of this command shows that there is a controller, two hosts (host h1 and host (h2), and a switch (s1).

Step 5: It is useful sometimes to display the links between the devices in Mininet to understand the topology. Issue the command shown below to see the available links.

```
net
```

A screenshot of a terminal window titled 'sdn@admin: ~'. The terminal shows the command 'net' being entered at the 'containernetwork>' prompt. The output lists the connections: 'h1 h1-eth0:s1-eth1', 'h2 h2-eth0:s1-eth2', 's1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0', and 'c0'. The word 'net' in the command is highlighted with a red box.

```
sdn@admin: ~  
File Actions Edit View Help  
sdn@admin: ~  
containernetwork> net  
h1 h1-eth0:s1-eth1  
h2 h2-eth0:s1-eth2  
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0  
c0  
containernetwork> 
```

Figure 8. Mininet's `net` command.

The output of this command shows that:

1. Host h1 is connected using its network interface h1-eth0 to the switch on interface s1-eth1.
2. Host h2 is connected using its network interface h2-eth0 to the switch on interface s1-eth2.
3. Switch s1:
 - a. has a loopback interface lo.
 - b. connects to h1-eth0 through interface s1-eth1.

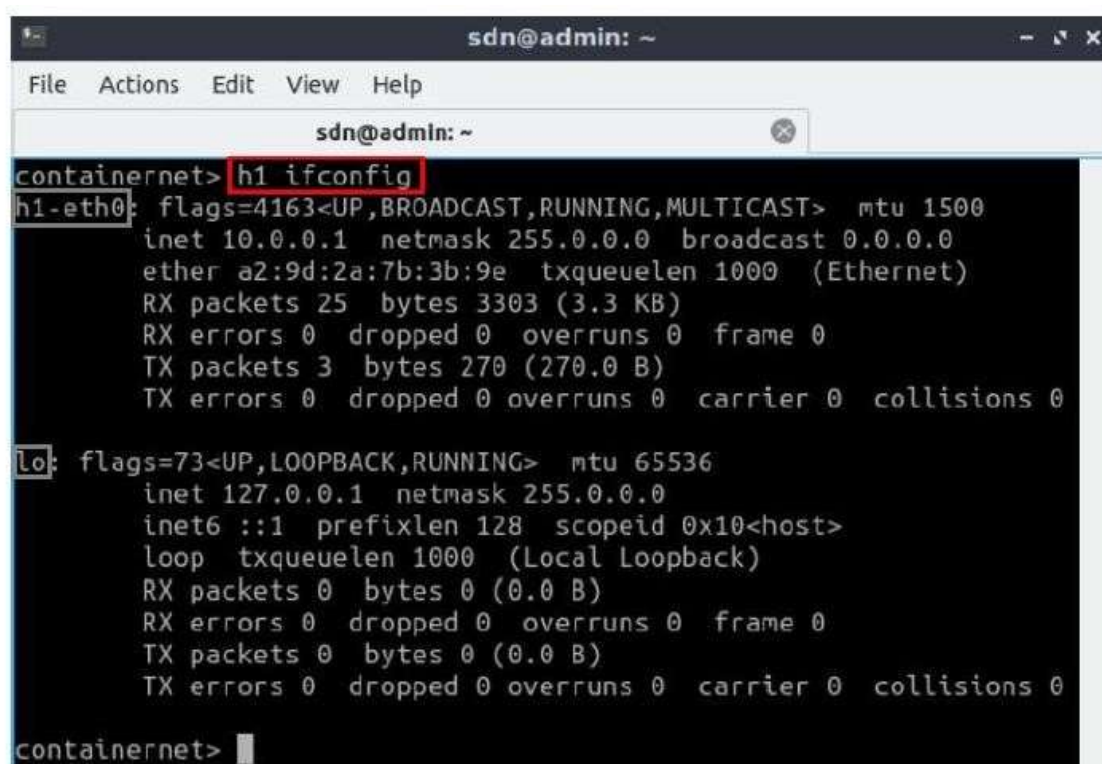
c. connects to h2-eth0 through interface s1-eth2.

4. Controller c0 is the brain of the network, where it has a global knowledge about the network. A controller instructs the switches on how to forward/drop packets in the network.

Mininet allows you to execute commands on a specific device. To issue a command for a specific node, you must specify the device first, followed by the command.

Step 6: To proceed, issue the command:

```
h1 ifconfig
```



```
sdn@admin: ~  
File Actions Edit View Help  
sdn@admin: ~  
containernetwork> h1 ifconfig  
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0  
    ether a2:9d:2a:7b:3b:9e txqueuelen 1000 (Ethernet)  
    RX packets 25 bytes 3303 (3.3 KB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 3 bytes 270 (270.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 0 bytes 0 (0.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
containernetwork> |
```

This command executes the [ifconfig Linux command on host h1. The command shows host h1's interfaces. The display indicates that host h1 has an interface h1-eth0 configured with IP address 10.0.0.1, and another interface lo configured with IP address 127.0.0.1 (loopback interface).

Conclusion:

Thus we learnt Using Mininet as an Emulator and POX controller, build your own internet router.

Assignment No.: 5

Aim:

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

Objectives:

1. To learn Cloud seeds automates IAAS using SDN and a high-performance network from Juniper SDN Framework.

Theory:

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 CloudSeeds, 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.

Challenges 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 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. Selection Criteria CloudSeeds' technical team had already worked with Juniper Networks technology in the past and felt that by comparison to other vendors it gave them more flexibility and future proofing, the configuration and management features were more advanced, and its open APIs had the best potential for scripting and automating tasks.

Kevin Fibich said, "Our entire architecture is a software-defined structure—and everything we do must work over the network. We knew that Juniper's technical philosophy fitted what we wanted to do, and how we wanted to do it." In particular, CloudSeeds felt that the ability to control its architecture using OpenStack software, and to put the Puppet software agent directly onto devices running Junos OS, set Juniper apart. Puppet is third-party software from Puppet Labs that is used for configuration management. It provides an efficient and scalable solution for managing the disparate configuration of large numbers of devices.

High-performance networking was also a critical requirement for CloudSeeds, along with granular support for quality of service (QoS) and high availability, including the ability to perform in- service software upgrades to ensure full business continuity for CloudSeeds' customers. CloudSeeds also had a preference to source all of its networking equipment from a single vendor to streamline building, maintenance and support. Solution CloudSeeds used various Juniper systems to create its new software defined A.C.R.E. platform.

It created the physical infrastructure using Juniper Networks® QFX5100 switches to provide it with a high- performance, high-density platform which can support 1GbE, 10GbE, and 40GbE connections. It also used Juniper Networks MX80 Universal Routing Platforms—flexible, full-featured routers that offer 80 Gbps of system throughput—and Juniper Networks SRX1400 Service Gateways for secure customer aggregation. Juniper's vSRX Virtual Firewall was used to create smaller virtual firewalls for CloudSeeds' customers, enabling CloudSeeds to deploy scalable firewall protection in a highly dynamic environment CloudSeeds used Juniper Networks Contrail Networking together with OpenStack to orchestrate the software-defined overlay networks, creating virtual networks, service chains, and using the powerful Contrail network analytics engine and API.

CloudSeeds highly values the development of Contrail Networking in the Tungsten Fabric (formerly, OpenContrail open source project and the open community around it. Contrail Networking is decoupled from the physical network underlay but made to interoperate and have visibility into any IP underlay network. The automation available in the Contrail Networking solution and Juniper data center underlay fabric solutions pairs perfectly to streamline the entire data center network at the speed of cloud. Furthermore, Contrail Networking's open approach to federation with routers was employed to peer with the MX routers to extend

and connect virtual networks between data centers, truly realizing the Juniper Networks' data center vision. Kevin Fibich said, "SDN is well used for cloud infrastructure. Using Contrail has allowed us to create a software-defined network that seamlessly integrates disparate locations into a single unified cloud." "Using Contrail has allowed us to create a software-defined network that seamlessly integrates disparate locations into a single unified cloud." - Kevin Fibach, Founder and Managing Director, CloudSeeds Results The new network has delivered CloudSeeds' vision of zero-touch provisioning. In turn, this allows the rapid uptake of IaaS services by CloudSeeds' customers, making their businesses far more agile and responsive. Kevin Fibich said, "Our customers need less staff to operate and run their infrastructure and provision new instances or infrastructure. In addition, the high amount of automation reduces the potential for human errors. They simply pick or receive a new device and cable it up, or we'll do it for them. It's up to the customer whether to operate the new device using their own IT crew, or to leave the management and operation of the device to CloudSeeds.

Once the equipment is physically in place and the software overlay is ready to go, it only takes minutes to bring up new servers or we can even deploy a whole new data center for customers in a few hours." CloudSeeds has also used SDN to create a highly robust and resilient infrastructure. CloudSeeds' approach has completely eliminated traditional Layer 2 issues, such as broadcast storms or switching loops, enabling it to use all of its available capacity to maximize operational efficiency and customer satisfaction. CloudSeeds also has a very high degree of control over its cloud network. Kevin Fibich said, "I need real-time visibility into any performance issues. Using Juniper technology helps me identify any problems or symptoms in the virtual space and isolate any errors in the underlying physical network.

This is only possible because of Juniper's open API, as it enables us to put our own software onto the devices. We see a lot of industry discussions around whether or not networking devices should become a simple commodity. We believe that it is far better to have a richly featured system that is supported by the vendor while still open for easy integration, as it gives us the best of both worlds—open and supported with new features." "The Juniper Networks architecture is unique in the way it has helped us create a software-defined network, and in turn an entire software-defined cloud. It has opened up a new world of possibilities for us and our customers." - Kevin Fibach, Founder and Managing Director, CloudSeeds CloudSeeds also valued Juniper's ability to perform in-service software upgrades, which has allowed CloudSeeds to automate the rollout of new software images without network disruption and removed the need to have technicians on site.

Kevin Fibich said, "Instead of rebooting the switch, we simply boot up a second Junos OS virtual machine, then this new machine takes over from the first one without any impact on packet forwarding, so our customers do not even notice software upgrades at all. Downtime is no longer accepted nor necessary." The network also helps CloudSeeds reduce costs. Kevin Fibich said, "Using Juniper equipment in conjunction with our automation approach means we have less operational expense. It also means we don't need that many specialized IT employees—in fact they are hard to find. This approach frees up our specialists to put their valuable skills to more creative, productive use elsewhere, for example, with customers." Next Steps

and Lessons Learned CloudSeeds now plans to add even greater scale into its infrastructure with Juniper's QFX10000 line of Ethernet switches, which offer platform support from 3 to 96 Tbps of throughput, all of which deliver the industry's highest 100GbE port density and support up to 480 ports in a single chassis.

This will enable CloudSeeds to continue building on its vision of a highly scalable and reliable physical network. CloudSeeds is also exploring the creation of an open marketplace for third-party software developers to build on Network Functions Virtualization (NFV) solutions orchestrated with Contrail Networking layer. Kevin Fibich concluded, "The Juniper Networks architecture is unique in the way it has helped us create a software-defined network, and in turn an entire software-defined cloud. It has opened up a new world of possibilities for us and our customers." About Juniper Networks Juniper Networks brings simplicity to networking with products, solutions and services that connect the world. Through engineering innovation, we remove the constraints and complexities of networking in the cloud era to solve the toughest challenges our customers and partners face daily. At Juniper Networks, we believe that the network is a resource for sharing knowledge and human advancement that changes the world. We are committed to imagining groundbreaking ways to deliver automated, scalable and secure networks to move at the speed of business.

Conclusion:

Thus we learnt Cloud seeds automate IAAS using SDN and a high-performance network from Juniper SDN Framework.