Aim: Using Mininet as an Emulator and POX controller, build your own internet router. Write simple outer 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 understand the concept of Software Defined Networking (SDN).
2. To use Mininet and POX controller to build an internet router.

Requirements:

1. Computer with any Linux OS installed
2. Python runtime environment
3. Mininet Emulator
4. POX controller

Theory:

Router:

- A router is a device that connects different computer networks together. It helps to direct traffic on the internet by sending data packets between networks. The primary function of a router is to route the incoming data packets to the correct destination.
- When a data packet is sent on the internet, it contains information about the sender and the recipient, as well as the actual data being sent. The router receives the packet and reads the information about the recipient. It then checks its own routing table to see where the recipient is located.
- The routing table is a list of destinations and the corresponding paths that the router should use to send the data packets to the correct destination. Once the router has determined the correct path, it sends the packet to the next router or directly to the destination if it is located on the same network.
- Routers are used to connect different types of networks together, such as local area networks (LANs) or wide area networks (WANs). They are also used to connect devices within a network, such as computers, printers, and other devices.
- Routers can perform other functions besides routing data packets. For example, they can act as firewalls to protect the network from unauthorized

access, or they can perform network address translation (NAT) to hide the internal network addresses from the outside world.

Ethernet Frames:

- Ethernet frames are the basic units of data in Ethernet networks. They contain important information about the data being transmitted and how it should be forwarded to its destination. Here are some key points to explain Ethernet frames:
- Ethernet frames are composed of several fields that contain information about the data being transmitted. Some of these fields include:
- Destination MAC address: The MAC address of the device that the frame is being sent to.
- Source MAC address: The MAC address of the device that sent the frame.
- Type/length: Indicates the type of data contained in the frame (e.g., IP packets).
- Payload: The data being transmitted.
- Ethernet frames are used to transmit data between devices on an Ethernet network, such as a local area network (LAN).
- The size of Ethernet frames is limited to a maximum of 1518 bytes, including the header and trailer.
- Ethernet frames are typically transmitted using a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol, which helps to avoid collisions when multiple devices are transmitting data simultaneously.
- When an Ethernet frame is transmitted, it is first sent to the switch or router closest to the source device. The switch or router then examines the destination MAC address in the frame to determine which interface the frame should be forwarded to.
- If the destination MAC address is not known, the switch or router may broadcast the frame to all devices on the network to try to find the device with the correct MAC address.

Routing Tables:

- Routing tables are used by routers to determine the next hop for packets based on their destination IP address.
- A static routing table is one that is manually configured and does not change dynamically.

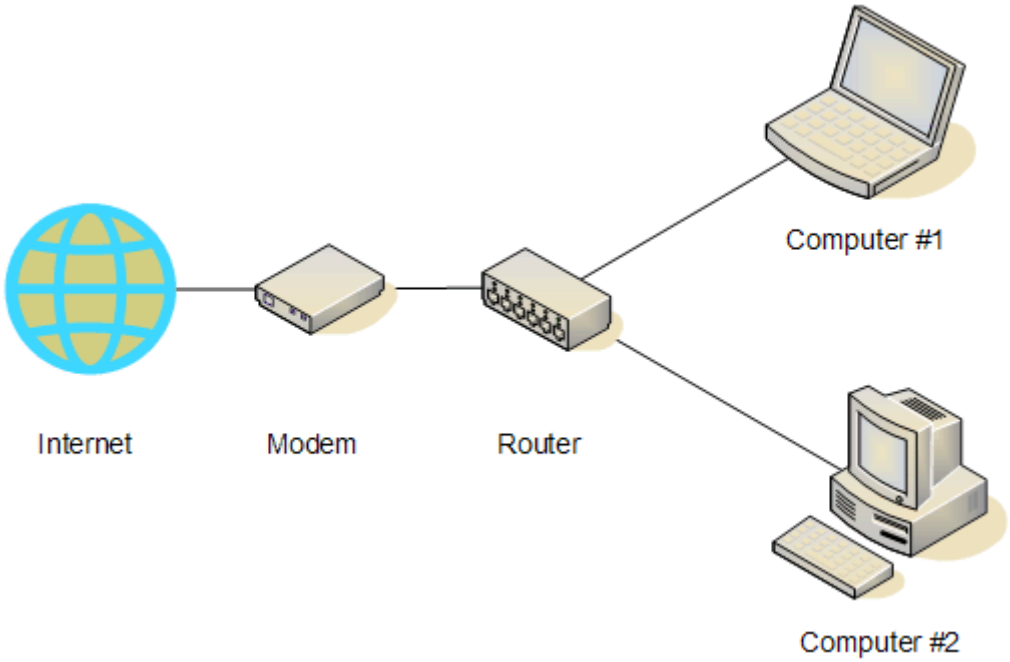

Fig. 1: Overview of Router's Role in a Network

Procedure:

Check Out Starter Code

```
cd ~
git clone https://huangty@bitbucket.org/huangty/cs144_lab3.git
cd krish/
git checkout --track remotes/origin/standalone
```

Install Simple Router POX module

```
cd ~/krish
./config.sh
```

Configuration Files

There will be two configuration files:
* `~/krish/IP_CONFIG`: Listed out the IP addresses assigned to the emulated hosts.
* `~/krish/router/rtable` (also linked to `~/krish/rtable`): The static routing table used for the simple router.

The default `IP_CONFIG` and `rtable` should look like the following:

```
> cat ~/krish/IP_CONFIG
server1 192.168.2.2
server2 172.64.3.10
client    10.0.1.100
sw0-eth1 192.168.2.1
sw0-eth2 172.64.3.1
sw0-eth3 10.0.1.1
> cat ~/krish/rtable
10.0.1.100  10.0.1.100  255.255.255.255 eth3
```

```
192.168.2.2 192.168.2.2 255.255.255.255 eth1
172.64.3.10  172.64.3.10  255.255.255.255 eth2
```

Test Connectivity of Your Emulated Topology:

Configure the environment by running the config.sh file

```
> cd ~/krish/
> ./config.sh
```

Start Mininet emulation by using the following command

```
> cd ~/krish/
> ./run_mininet.sh
```

You should be able to see some output like the following:

```
*** Shutting down stale SimpleHTTPServers
*** Shutting down stale webservers
server1 192.168.2.2
server2 172.64.3.10
client 10.0.1.100
sw0-eth1 192.168.2.1
sw0-eth2 172.64.3.1
sw0-eth3 10.0.1.1
*** Successfully loaded ip settings for hosts
 {'server1': '192.168.2.2', 'sw0-eth3': '10.0.1.1', 'sw0-eth1':
'192.168.2.1', 'sw0-eth2': '172.64.3.1', 'client': '10.0.1.100', 'server2':
'172.64.3.10'}
*** Creating network
*** Creating network
*** Adding controller
*** Adding hosts:
client server1 server2
*** Adding switches:
sw0
*** Adding links:
(client, sw0) (server1, sw0) (server2, sw0)
*** Configuring hosts
client server1 server2
*** Starting controller
*** Starting 1 switches
sw0
*** setting default gateway of host server1
server1 192.168.2.1
*** setting default gateway of host server2
server2 172.64.3.1
*** setting default gateway of host client
client 10.0.1.1
*** Starting SimpleHTTPServer on host server1
*** Starting SimpleHTTPServer on host server2
*** Starting CLI:
mininet>
```

Keep this terminal open, as you will need the mininet command line for debugging. Now, use another terminal to continue the next step. (Do not press ctrl-z.)

Mininet requires a controller, which we implemented in POX (revision `f95dd1a81584d716823bbf565fa68254416af603`). To run the controller, use the following command:

```
> cd ~/krish/
> ln -s ../pox
> ./run_pox.sh
```

You should be able to see some output like the following:

```
POX 0.0.0 / Copyright 2011 James McCauley
DEBUG:.home.ubuntu.krish.pox_module.cs144.ofhandler:*** ofhandler:
Successfully loaded ip settings for hosts
 {'server1': '192.168.2.2', 'sw0-eth3': '10.0.1.1', 'sw0-eth1':
'192.168.2.1', 'sw0-eth2': '172.64.3.1', 'client': '10.0.1.100', 'server2':
'172.64.3.10'}

INFO:.home.ubuntu.krish.pox_module.cs144.srhandler:created server
DEBUG:.home.ubuntu.krish.pox_module.cs144.srhandler:SRServerListener
listening on 8888
DEBUG:core:POX 0.0.0 going up...
DEBUG:core:Running on CPython (2.7.3/Aug 1 2012 05:14:39)
INFO:core:POX 0.0.0 is up.
This program comes with ABSOLUTELY NO WARRANTY.  This program is free
software,
and you are welcome to redistribute it under certain conditions.
Type 'help(pox.license)' for details.
DEBUG:openflow.of_01:Listening for connections on 0.0.0.0:6633
Ready.
POX>
```

Please note that you have to wait for Mininet to connect to the POX controller before you continue to the next step. Once Mininet has connected, you will see the following output:

```
INFO:openflow.of_01:[Con 1/249473472573510] Connected to e2-e5-11-b6-b0-46
DEBUG:.home.ubuntu.krish.pox_module.cs144.ofhandler:Connection [Con
1/249473472573510]
DEBUG:.home.ubuntu.krish.pox_module.cs144.srhandler:SRServerListener catch
RouterInfo even, info={'eth3': ('10.0.1.1', '86:05:70:7e:eb:56', '10Gbps',
3), 'eth2': ('172.64.3.1', 'b2:9e:54:d8:9d:cd', '10Gbps', 2), 'eth1':
('192.168.2.1', '36:61:7c:4f:b6:7b', '10Gbps', 1)}, rtable=[]
```

Keep POX running. Now, open yet another terminal to continue the next step. (Don't press ctrl-z.)

Now you are ready to test out the connectivity of the environment setup. To do so, run the binary file of the solution, `sr_solution`:

```
> cd ~/krish/
> ./sr_solution
```

You should be able to see some output like the following:

```
Loading routing table from server, clear local routing table.
Loading routing table
```

```
-------------------------------------------
Destination     Gateway         Mask    Iface
10.0.1.100              10.0.1.100      255.255.255.255 eth3
192.168.2.2             192.168.2.2     255.255.255.255 eth1
172.64.3.10             172.64.3.10     255.255.255.255 eth2
-------------------------------------------
Client ubuntu connecting to Server localhost:8888
Requesting topology 0
successfully authenticated as ubuntu
Loading routing table from server, clear local routing table.
Loading routing table
-------------------------------------------
Destination     Gateway         Mask    Iface
10.0.1.100              10.0.1.100      255.255.255.255 eth3
192.168.2.2             192.168.2.2     255.255.255.255 eth1
172.64.3.10             172.64.3.10     255.255.255.255 eth2
-------------------------------------------
Router interfaces:
eth3    HWaddr86:05:70:7e:eb:56
        inet addr 10.0.1.1
eth2    HWaddrb2:9e:54:d8:9d:cd
        inet addr 172.64.3.1
eth1    HWaddr36:61:7c:4f:b6:7b
        inet addr 192.168.2.1
 <-- Ready to process packets -->
```

In this particular setup, 192.168.2.2 is the IP for `server1`, and 172.64.3.10 is the IP for `server2`. You can find the IP addresses in your `IP_CONFIG` file.

Now, back to the terminal where Mininet is running. To issue an command on the emulated host, type the host name followed by the command in the Mininet console. For example, the following command issues 3 pings from the `client` to `server1`.

```
mininet> client ping -c 3 192.168.2.2
```

You should be able to see the following output.

```
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_req=1 ttl=63 time=66.9 ms
64 bytes from 192.168.2.2: icmp_req=2 ttl=63 time=49.9 ms
64 bytes from 192.168.2.2: icmp_req=3 ttl=63 time=68.8 ms
```

You can also use traceroute to see the route between client to server1.

```
mininet> client traceroute -n 192.168.2.2
```

You should be able to see the following output.

```
traceroute to 192.168.2.2 (192.168.2.2), 30 hops max, 60 byte packets
 1  10.0.1.1  146.069 ms  143.739 ms  143.523 ms
 2  192.168.2.2  226.260 ms  226.070 ms  225.868 ms
```

Finally, to test the web server is properly working at the server1 and server2, issue an HTTP request by using *wget* or *curl*.

```
mininet> client wget http://192.168.2.2
```

You should be able to see the following output.

```
--2012-12-17 06:52:23--  http://192.168.2.2/
Connecting to 192.168.2.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 161 [text/html]
Saving to: `index.html'

     0K                                                       100% 17.2M=0s

2012-12-17 06:52:24 (17.2 MB/s) - `index.html' saved [161/161]
```

Conclusion: Thus, we learned how to build our own internet router using Mininet as an emulator and POX controller.