

**CEG5101 Modern Computer Networking**  
**Graduate Assistant: Mr. Binghui Wu and Ms. Divya D Kulkarni,**  
**Prepared by: Ms. Anum Talpur (Ex-Graduate Assistant)**  
**Lecturer: Assoc. Prof. Mohan Gurusamy**  
**ECE Department, CDE., National University of Singapore**

**Laboratory 03**  
**Software-Defined Network (SDN) with POX Controller**

**LEARNING OBJECTIVES**

Upon successful completion of this laboratory, the students will be able:

- i. To become familiar with the concept of SDN and Controller.
- ii. To download and install the POX controller.
- iii. To invoke the POX controller as a layer-2 learning switch.
- iv. To invoke the POX controller as a forwarding hub.
- v. To invoke the POX controller as a load balancer.

**EQUIPMENT**

- i. *Hardware* – Laptop/Computer with internet access
- ii. *Software* – Mininet, Linux OS

**DISCUSSION**

***Task 1: To become familiar with the concept of SDN and Controller***

- 1.1 SDN or software-defined networking is a concept in which application programming interfaces (APIs) are used to enable intelligence within the network.
- 1.2 It helps to control the network centrally and consistently regardless of the underlying topology.
- 1.3 Centralized intelligence is achieved by separating the control plane and the data plane to get control over the network behaviour.
- 1.4 The data plane is the part of the network which handles traffic forwarding and the control plane is the part that instructs or configures the data plane on the traffic forwarding process.
- 1.5 In a traditional network setup, the control plane and data plane lie together on a single device. However, SDN decouples the control plane from the data plane and implements it as software instead to enable flexibility within a network through programmatic access.
- 1.6 In this lab activity, we will learn to use a controller to update data forwarding rules on the Open vSwitch (OVS) to enable communication between different hosts.

\*<https://noxrepo.github.io/pox-doc/html/#stock-components>  
<http://mininet.org/walkthrough/>

## Task 2: To download and install POX SDN Controller

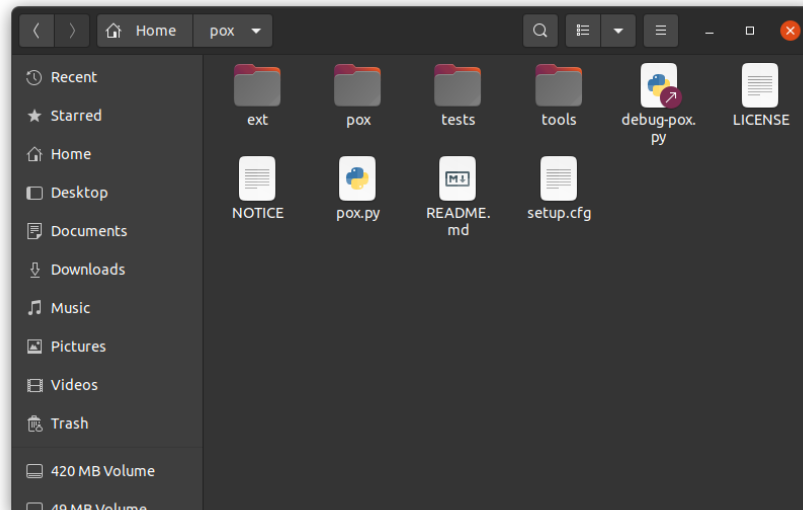
- 2.1 In this activity, we will install the POX controller.
- 2.2 POX controller is a python-based controller platform that allows fast prototyping and controlling of the network using stock components that are pre-programmed.
- 2.3 The stock components provide different functionalities and features and are listed on GitHub\*. The code for each component can be found in the installed directory of the POX.
- 2.4 In this activity, we will start by downloading and installing the POX controller which will be used as a remote controller in our further lab tasks.
- 2.5 To download and get started with the POX controller, simply issue the command “mininet/util/install.sh -p” on your home directory location.



```
anum@anum-OptiPlex-5050: ~  
anum@anum-OptiPlex-5050:~$ mininet/util/install.sh -p  
Detected Linux distribution: Ubuntu 20.04 focal amd64  
sys.version_info(major=2, minor=7, micro=18, releaselevel='final', serial=0)  
Detected Python (python2) version 2  
Installing POX into /home/anum/pox...  
Cloning into 'pox'...  
remote: Enumerating objects: 13036, done.  
remote: Counting objects: 100% (261/261), done.  
remote: Compressing objects: 100% (95/95), done.  
remote: Total 13036 (delta 161), reused 252 (delta 161), pack-reused 12775  
Receiving objects: 100% (13036/13036), 5.01 MiB | 4.25 MiB/s, done.  
Resolving deltas: 100% (8410/8410), done.  
anum@anum-OptiPlex-5050:~$
```

- 2.6 To verify successful installation, you will see a folder named POX must be created in the home directory with contents as shown below.

\*<https://noxrepo.github.io/pox-doc/html/#stock-components>  
<http://mininet.org/walkthrough/>

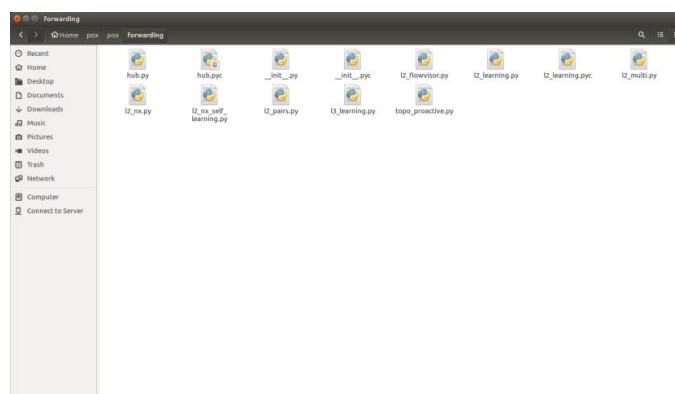


- 2.7 Besides, we need to use “curl” to get data from a server. Therefore, download “curl” in your terminal by type this in your terminal

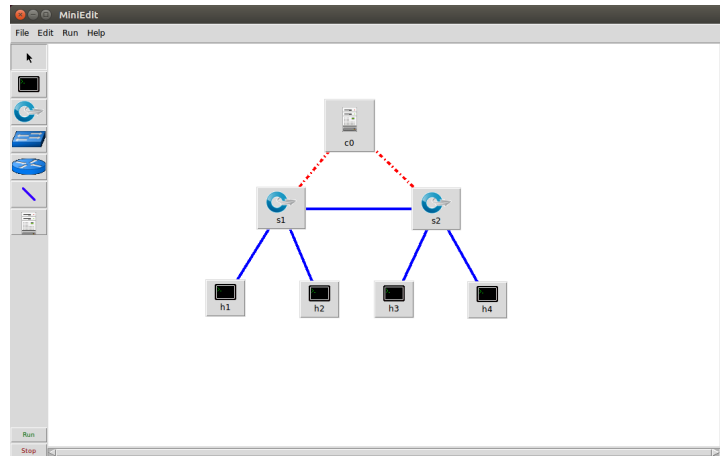
```
sudo apt update
sudo apt upgrade
sudo apt install curl
```

**Task 3: Invoke the POX controller as a layer-2 learning switch.**

- 3.1 In this activity, we will use the stock component and start POX with the basic layer-2 forwarding switch for the controller.
- 3.2 To implement layer-2 forwarding, we will use the component named “forwarding.l2\_learning”. The code for this component can be found at `~/pox/pox/forwarding/l2_learning.py`.



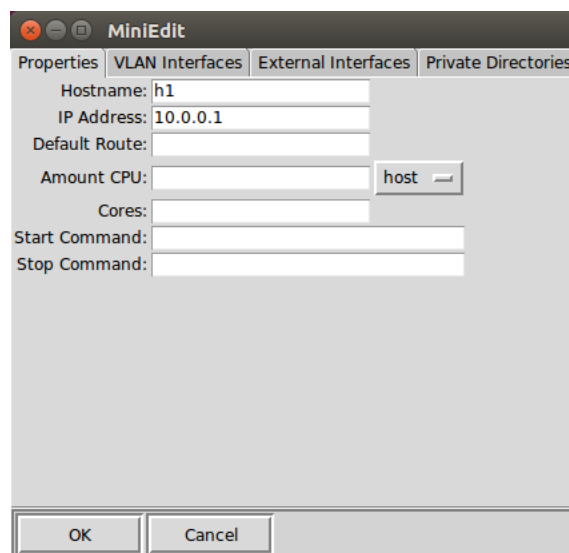
- 3.3 This POX component will enable traffic forwarding over the connected OVSS.
- 3.4 We will start with creating a below-given topology in the miniedit.



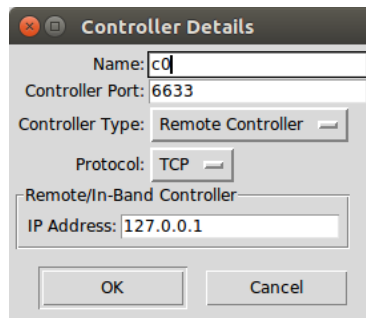
- 3.5 Next, assign IP addresses to the hosts and an IP address and port number to the POX controller according to the table given below.

Device	IP Address	Port Number
c0	127.0.0.1	6633
h1	10.0.0.1	-
h2	10.0.0.2	-
h3	10.0.0.3	-
h4	10.0.0.4	-

- 3.6 Use the host property window, to assign IP addresses to the hosts.



- 3.7 Use the controller property window, to assign an IP address and port number to the controller. NOTE: make sure the controller type is chosen as “remote controller”.



3.8 Next, on a different terminal, open the pox directory with the command “cd pox” and start the pox controller by issuing the following command “sudo python3 pox.py forwarding.l2\_learning” to enable layer-2 traffic forwarding. (if you are using python2, use “sudo ./pox.py forwarding.l2\_learning” )

3.9 Once the POX controller is up, your window should like below.

Python2

```
anumtalpur@anumtalpur-Precision-T3600: ~/pox
(base) anumtalpur@anumtalpur-Precision-T3600:~$ cd pox
(base) anumtalpur@anumtalpur-Precision-T3600:~/pox$ sudo ./pox.py forwarding.l2_
learning
[sudo] password for anumtalpur:
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
```

Python3

```
ubuntu@sdnlab:~/pox$ sudo python3 pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.
8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
```

3.10 Next, **run** the network from miniedit window.

3.11 The POX controller terminal will show the connected OVS switches.

```
ubuntu@sdnlab:~/pox$ sudo python3 pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

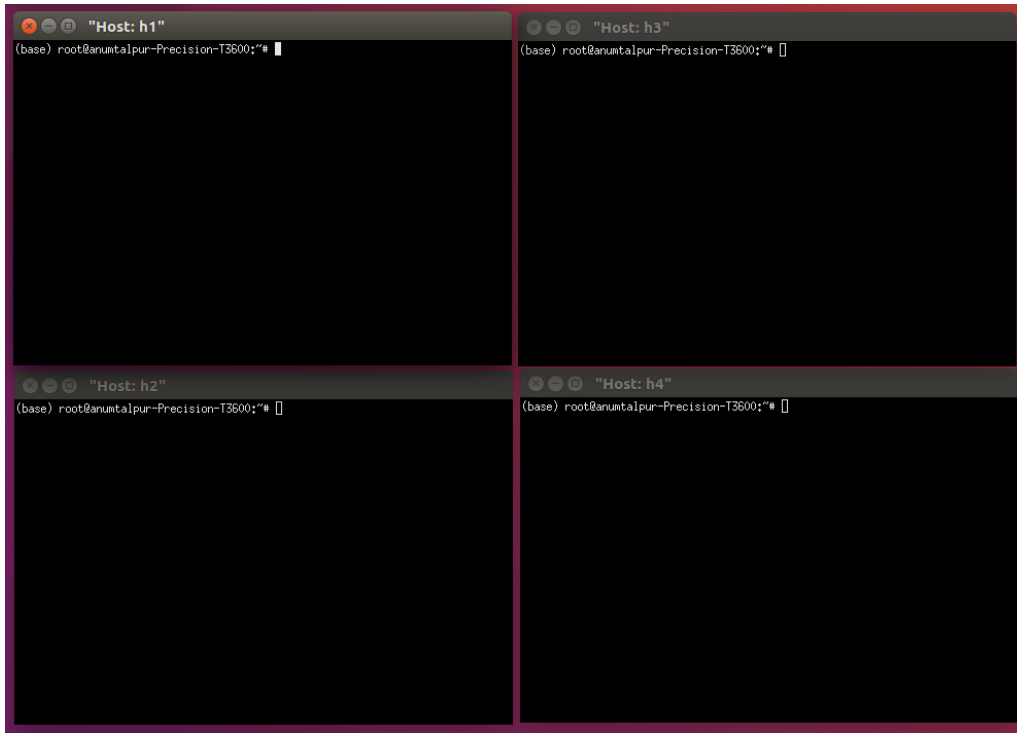
3.12 Here, the POX controller starts the basic layer 2 forwarding over the connected switches.

3.13 At this stage, the flow tables over switches will be empty until the hosts start to send packets.

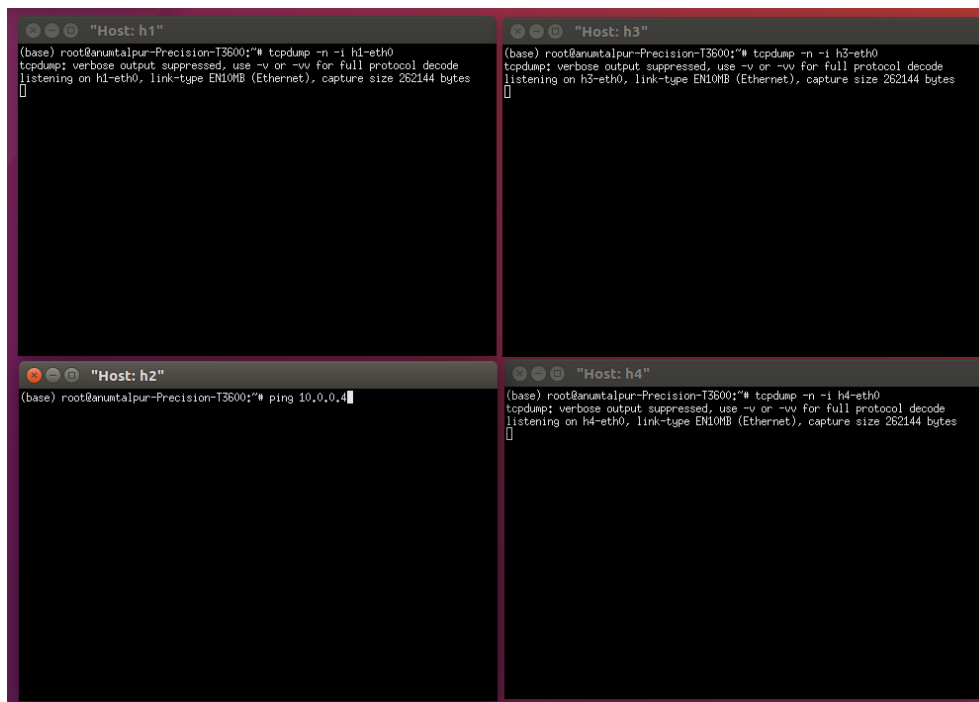
```
anumtalpur@anumtalpur-Precision-T3600: ~
(base) anumtalpur@anumtalpur-Precision-T3600:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
(base) anumtalpur@anumtalpur-Precision-T3600:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
(base) anumtalpur@anumtalpur-Precision-T3600:~$
```

3.14 To verify the successful functioning, we will start sending packets from one of the hosts and monitor the flow of traffic on all connected hosts.

3.15 Next, open terminals for all four hosts by right-clicking on the host at the miniedit workspace or by issuing the command “xterm h1 h2 h3 h4” over the Mininet CLI.



- 3.16 Next, issue the command `tcpdump -n -i hX-eth0` for all hosts except one because that would be used to send the ping command. Here, **X** is the number of the host.
- 3.17 In this activity, we issue `tcpdump` over h1, h3, and h4, and we use h2 to send a ping command toward h4.



- 3.18 Once the ping starts, we will notice that since at first switch doesn't have any information regarding h4 therefore the first packet is sent to all hosts as per the layer-2 forwarding rule. However, only h4 will send a reply. NOTE: at h1 and h3 you can notice a line within TCP traffic says "Request who-has 10.0.0.4 tell 10.0.0.2, length 28".

```

Host: h1
22:16:24.509306 IP6 fe80::4877:97ff:fe7e:a544 > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:26.521351 IP6 fe80::f41a:5bfff:fe72:8665 > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:26.524992 IP6 fe80::4c59:6fff:fe95:ca0a > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:35.209654 IP6 fe80::f41a:5bfff:fe72:8665.5353 > ff02::fb.5353: 0 [9q] PTR (
QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (
QM)?_ftp_tcp.local, PTR (QM)?_webdav_tcp.local, PTR (QM)?_webdav_tcp.lo
cal, PTR (QM)?_sftp_ssh_tcp.local, PTR (QM)?_smb_tcp.local, PTR (QM)?_afpov
ertcp_tcp.local, (141)
22:16:35.568435 IP6 fe80::b484:e8fff:fe53:295a.5353 > ff02::fb.5353: 0 [9q] PTR (
QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (
QM)?_ftp_tcp.local, PTR (QM)?_webdav_tcp.local, PTR (QM)?_webdav_tcp.lo
cal, PTR (QM)?_sftp_ssh_tcp.local, PTR (QM)?_smb_tcp.local, PTR (QM)?_afpov
ertcp_tcp.local, (141)
22:16:38.007262 IP6 fe80::b484:e8fff:fe53:295a > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:41.441201 ARP, Request who-has 10.0.0.4 tell 10.0.0.2, length 28
22:16:46.529437 IP6 fe80::88e5:f4ff:fe8a:ddde > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:06.313004 IP6 fe80::20a4:e8fff:fe67:e582 > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:21.214584 IP6 fe80::4877:97ff:fe7e:a544 > ff02::2: ICMP6, router solicitat
ion, length 16

Host: h3
(base) root@anumtalpur-Precision-T3600:~# tcpdump -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:16:20.389906 IP6 fe80::20a4:e8fff:fe67:e582 > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:24.504472 IP6 fe80::4877:97ff:fe7e:a544 > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:26.513178 IP6 fe80::4c59:6fff:fe95:ca0a > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:35.833108 IP6 fe80::303f:e7ff:fe0e:66c5.5353 > ff02::fb.5353: 0 [9q] PTR (
QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (
QM)?_ftp_tcp.local, PTR (QM)?_webdav_tcp.local, PTR (QM)?_webdav_tcp.lo
cal, PTR (QM)?_sftp_ssh_tcp.local, PTR (QM)?_smb_tcp.local, PTR (QM)?_afpov
ertcp_tcp.local, (141)
22:16:35.439668 IP6 fe80::43e6bfff:fe28:ec12.5353 > ff02::fb.5353: 0 [9q] PTR (
QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (QM)?_ftp_tcp.local, PTR (
QM)?_ftp_tcp.local, PTR (QM)?_webdav_tcp.local, PTR (QM)?_webdav_tcp.lo
cal, PTR (QM)?_sftp_ssh_tcp.local, PTR (QM)?_smb_tcp.local, PTR (QM)?_afpov
ertcp_tcp.local, (141)
22:16:40.579445 IP6 fe80::43e6bfff:fe28:ec12 > ff02::2: ICMP6, router solicitat
ion, length 16
22:16:41.442763 ARP, Request who-has 10.0.0.4 tell 10.0.0.2, length 28
22:16:47.011339 IP6 fe80::88e5:f4ff:fe8a:ddde > ff02::2: ICMP6, router solicitat
ion, length 16

Host: h2
64 bytes from 10.0.0.4: icmp_seq=152 ttl=64 time=0.089 ms
64 bytes from 10.0.0.4: icmp_seq=153 ttl=64 time=0.103 ms
64 bytes from 10.0.0.4: icmp_seq=154 ttl=64 time=0.048 ms
64 bytes from 10.0.0.4: icmp_seq=155 ttl=64 time=0.077 ms
64 bytes from 10.0.0.4: icmp_seq=156 ttl=64 time=0.095 ms
64 bytes from 10.0.0.4: icmp_seq=157 ttl=64 time=0.074 ms
64 bytes from 10.0.0.4: icmp_seq=158 ttl=64 time=0.058 ms
64 bytes from 10.0.0.4: icmp_seq=159 ttl=64 time=0.073 ms
64 bytes from 10.0.0.4: icmp_seq=160 ttl=64 time=0.080 ms
64 bytes from 10.0.0.4: icmp_seq=171 ttl=64 time=0.068 ms
64 bytes from 10.0.0.4: icmp_seq=172 ttl=64 time=0.074 ms
64 bytes from 10.0.0.4: icmp_seq=173 ttl=64 time=0.051 ms
64 bytes from 10.0.0.4: icmp_seq=174 ttl=64 time=0.054 ms
64 bytes from 10.0.0.4: icmp_seq=175 ttl=64 time=0.081 ms
64 bytes from 10.0.0.4: icmp_seq=176 ttl=64 time=0.071 ms
64 bytes from 10.0.0.4: icmp_seq=177 ttl=64 time=0.074 ms
64 bytes from 10.0.0.4: icmp_seq=178 ttl=64 time=0.068 ms
64 bytes from 10.0.0.4: icmp_seq=180 ttl=64 time=0.081 ms
64 bytes from 10.0.0.4: icmp_seq=181 ttl=64 time=0.053 ms
64 bytes from 10.0.0.4: icmp_seq=182 ttl=64 time=0.050 ms
64 bytes from 10.0.0.4: icmp_seq=183 ttl=64 time=0.072 ms
64 bytes from 10.0.0.4: icmp_seq=184 ttl=64 time=0.067 ms

Host: h4
64
22:13:43.479310 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 3771, seq 179, lengt
h 64
22:13:44.503286 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 3771, seq 180, len
gth 64
22:13:44.503306 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 3771, seq 180, lengt
h 64
22:13:45.527222 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 3771, seq 181, len
gth 64
22:13:45.527239 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 3771, seq 181, lengt
h 64
22:13:46.551213 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 3771, seq 182, len
gth 64
22:13:46.551228 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 3771, seq 182, lengt
h 64
22:13:47.628100 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 3771, seq 183, len
gth 64
22:13:47.628133 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 3771, seq 183, lengt
h 64
22:13:48.576672 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 3771, seq 184, len
gth 64
22:13:48.576894 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 3771, seq 184, lengt
h 64

```

- 3.19 Once the first ping is done, the OVS table will have a new flow entry based on the performed ping command.

```

anumtalpur@anumtalpur-Precision-T3600:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=19.612s, table=0, n_packets=21, n_bytes=2058, idle_timeout
=10, hard_timeout=30, idle_age=0, priority=65535,icmp,in_port=3,vlan_tci=0x0000,
dl_src=22:a4:ef:67:e5:82,dl_dst=4a:77:97:7e:a5:44,nw_src=10.0.0.2,nw_dst=10.0.0.
4,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:1
 cookie=0x0, duration=19.605s, table=0, n_packets=20, n_bytes=1960, idle_timeout
=10, hard_timeout=30, idle_age=0, priority=65535,icmp,in_port=1,vlan_tci=0x0000,
dl_src=4a:77:97:7e:a5:44,dl_dst=22:a4:ef:67:e5:82,nw_src=10.0.0.4,nw_dst=10.0.0.
2,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:3
(base) anumtalpur@anumtalpur-Precision-T3600:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=22.018s, table=0, n_packets=22, n_bytes=2156, idle_timeout
=10, hard_timeout=30, idle_age=0, priority=65535,icmp,in_port=1,vlan_tci=0x0000,
dl_src=22:a4:ef:67:e5:82,dl_dst=4a:77:97:7e:a5:44,nw_src=10.0.0.2,nw_dst=10.0.0.
4,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:3
 cookie=0x0, duration=22.016s, table=0, n_packets=22, n_bytes=2156, idle_timeout
=10, hard_timeout=30, idle_age=0, priority=65535,icmp,in_port=3,vlan_tci=0x0000,
dl_src=4a:77:97:7e:a5:44,dl_dst=22:a4:ef:67:e5:82,nw_src=10.0.0.4,nw_dst=10.0.0.
2,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
(base) anumtalpur@anumtalpur-Precision-T3600:~$

```

- 3.20 Now, during the second ping, the switch doesn't need to forward the packet to all other hosts and this time the traffic will only be forwarded to h4. You can verify this by ending the current tcpdump and ping commands and issuing the same commands again.

- 3.21 You can notice in the below window that this time the traffic is only sent to h4. The hosts h1 and h3 do not receive any information regarding the ping request from h2.

```

Host: h1
(base) root@anumalpur-Precision-T3600:~# tcpdump -n -i h1-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:21:55.794719 IP6 fe80::204:effff:fe67:e582 > ff02::2: ICMP6, router solicitat
ion, length 16
[]

Host: h3
(base) root@anumalpur-Precision-T3600:~# tcpdump -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:21:55.795479 IP6 fe80::204:effff:fe67:e582 > ff02::2: ICMP6, router solicitat
ion, length 16
[]

Host: h2
(base) root@anumalpur-Precision-T3600:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=21.2 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.565 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.065 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.095 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.063 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.084 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=0.083 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.084 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=0.088 ms
64 bytes from 10.0.0.4: icmp_seq=11 ttl=64 time=0.090 ms
64 bytes from 10.0.0.4: icmp_seq=12 ttl=64 time=0.087 ms
[]

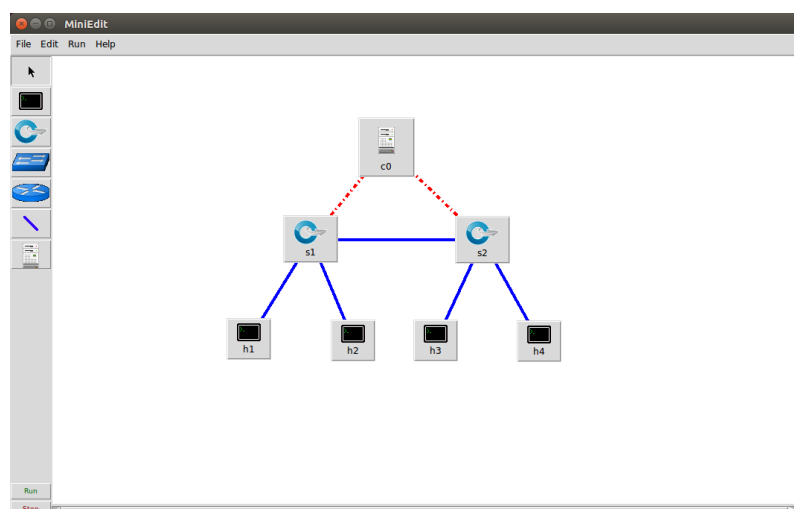
Host: h4
h4
22:21:44.247306 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 4041, seq 7, length
64
22:21:45.271299 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 4041, seq 8, lengt
h 64
22:21:45.271332 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 4041, seq 8, length
64
22:21:46.295281 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 4041, seq 9, length
h 64
22:21:46.295313 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 4041, seq 9, length
64
22:21:47.319290 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 4041, seq 10, leng
th 64
22:21:47.319322 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 4041, seq 10, length
64
22:21:48.343305 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 4041, seq 11, leng
th 64
22:21:48.343337 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 4041, seq 11, length
64
22:21:49.367294 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 4041, seq 12, leng
th 64
22:21:49.367327 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 4041, seq 12, length
64
[]

```

- 3.22 This way the controller has successfully enabled layer-2 forwarding within the network without any manual entries of flows over the switches.

#### Task 4: Invoke the POX controller as a forwarding hub.

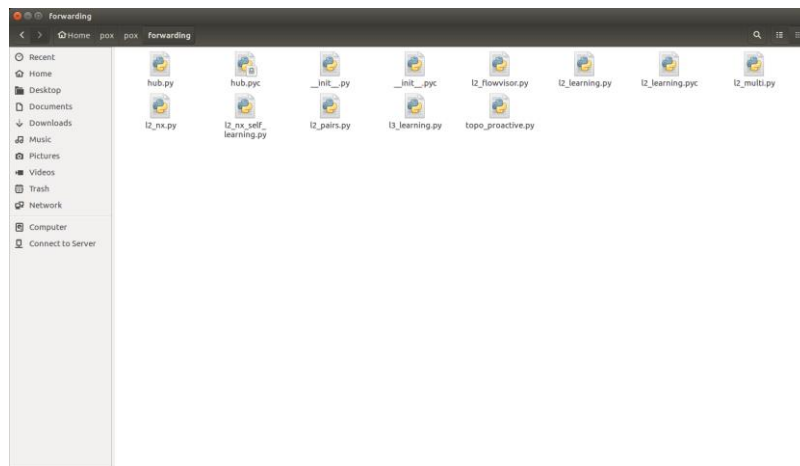
- 4.1 In this activity, we will use another stock component and start POX with the hub functioning.
- 4.2 This will enable flooding rules on every connected switch.
- 4.3 We will use the same topology and network configuration in this task also.



- 4.4 To start the pox controller as a hub, we will issue the command : “sudo python 3 pox.py forwarding.hub”  
(Python 2 : “sudo ./pox.py forwarding.hub”).

```
(base) anumtalpur@anumtalpur-Precision-T3600:~$ cd pox
(base) anumtalpur@anumtalpur-Precision-T3600:~/pox$ sudo ./pox.py forwarding.hub
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:forwarding.hub:Proactive hub running.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-02
```

- 4.5 The python code for forwarding.hub can be found from the same forwarding folder of the POX directory.



- 4.6 Next, to verify the successful flooding among hosts, we will start sending a ping from one of the hosts and monitor the flow of traffic on all connected hosts. To do so, open terminals for all four hosts and issue a ping from one host and tcpdump on all three remaining hosts.

```
(base) root@anumtalpur-Precision-T3600:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:28:20.781245 ARP, Request who-has 10.0.0.4 tell 10.0.0.2, length 28
13:28:20.781534 ARP, Reply 10.0.0.4 is-at da:4c:d9:fd:ff:43 (oui Unknown), length 28
13:28:20.781745 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 1, length 64
13:28:20.782053 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 1, length 64
13:28:21.782201 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 2, length 64
13:28:21.782252 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 2, length 64
13:28:22.807257 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 3, length 64
13:28:22.807302 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 3, length 64
13:28:23.831282 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 4, length 64
13:28:23.831329 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 4, length 64
13:28:24.855268 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 5, length 64
13:28:24.855272 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 5, length 64

[1]+  Stopped                  ping 10.0.0.4
(base) root@anumtalpur-Precision-T3600:~#
```

```
(base) root@anumtalpur-Precision-T3600:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:28:20.781404 ARP, Request who-has 10.0.0.4 tell 10.0.0.2, length 28
13:28:20.781546 ARP, Reply 10.0.0.4 is-at da:4c:d9:fd:ff:43 (oui Unknown), length 28
13:28:20.781835 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 1, length 64
13:28:20.781961 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 1, length 64
13:28:21.782205 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 2, length 64
13:28:21.782250 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 2, length 64
13:28:22.807261 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 3, length 64
13:28:22.807301 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 3, length 64
13:28:23.831285 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 4, length 64
13:28:23.831327 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 4, length 64
13:28:24.855272 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 5, length 64
13:28:24.855274 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 5, length 64
```

```
(base) root@anumtalpur-Precision-T3600:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.723 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.139 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.124 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.113 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.121 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.121 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.055 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=0.100 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.115 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=0.071 ms
64 bytes from 10.0.0.4: icmp_seq=11 ttl=64 time=0.097 ms
64 bytes from 10.0.0.4: icmp_seq=12 ttl=64 time=0.120 ms
64 bytes from 10.0.0.4: icmp_seq=13 ttl=64 time=0.103 ms
64 bytes from 10.0.0.4: icmp_seq=14 ttl=64 time=0.110 ms
^C
[1]+  Stopped                  ping 10.0.0.4
(base) root@anumtalpur-Precision-T3600:~#
```

```
(base) root@anumtalpur-Precision-T3600:~# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h4-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:28:20.781404 ARP, Request who-has 10.0.0.4 tell 10.0.0.2, length 28
13:28:20.781546 ARP, Reply 10.0.0.4 is-at da:4c:d9:fd:ff:43 (oui Unknown), length 28
13:28:20.781835 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 1, length 64
13:28:20.781961 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 1, length 64
13:28:21.782205 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 2, length 64
13:28:21.782243 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 2, length 64
13:28:22.807262 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 3, length 64
13:28:22.807293 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 3, length 64
13:28:23.831287 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 4, length 64
13:28:23.831319 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 4, length 64
13:28:24.855274 IP 10.0.0.2 > 10.0.0.4: ICMP echo request, id 11352, seq 5, length 64
13:28:24.855274 IP 10.0.0.4 > 10.0.0.2: ICMP echo reply, id 11352, seq 5, length 64
```

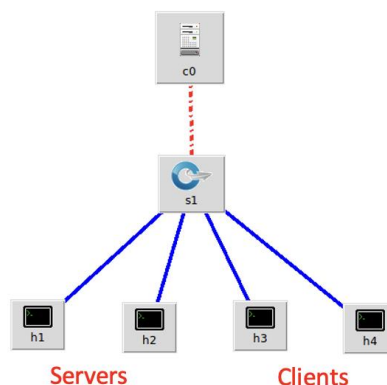
- 4.7 This time you will notice that the ping command has been flooded and received at all of the connected hosts.
- 4.8 You can also print the tables of connected OVS switches to verify the addition of flows with flooding action.

```
anumtalpur@anumtalpur-Precision-T3600: ~  
(base) anumtalpur@anumtalpur-Precision-T3600:~$ sudo ovs-ofctl dump-flows s1  
[sudo] password for anumtalpur:  
NXST_FLOW reply (xid=0x4):  
  cookie=0x0, duration=392.008s, table=0, n_packets=177, n_bytes=17463, idle_age=122, actions=FLOOD  
(base) anumtalpur@anumtalpur-Precision-T3600:~$ sudo ovs-ofctl dump-flows s2  
NXST_FLOW reply (xid=0x4):  
  cookie=0x0, duration=396.699s, table=0, n_packets=178, n_bytes=17553, idle_age=94, actions=FLOOD  
(base) anumtalpur@anumtalpur-Precision-T3600:~$
```

- 4.9 This verifies the successful functioning of switches as a flooding hub.

***Task 5: Invoke the POX controller as a load balancer.***

- 5.1 In this activity, we will use the POX controller to implement load balancing.
- 5.2 We will start with creating a below-given topology in the miniedit where single OVS connects four machines and out of these two machines, h1 and h2 will be used as HTTP servers, and h3 and h4 will be used as HTTP clients.

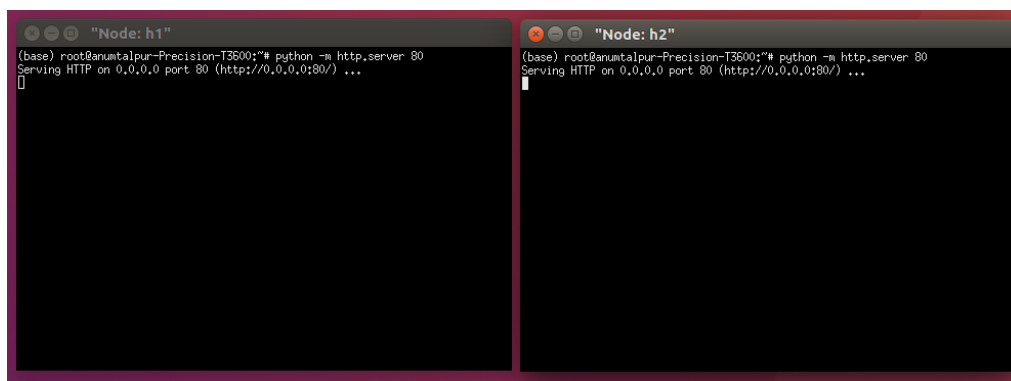


- 5.3 Idea is to balance the load among servers (i.e., h1 and h2) which is generated from clients named h3 and h4.

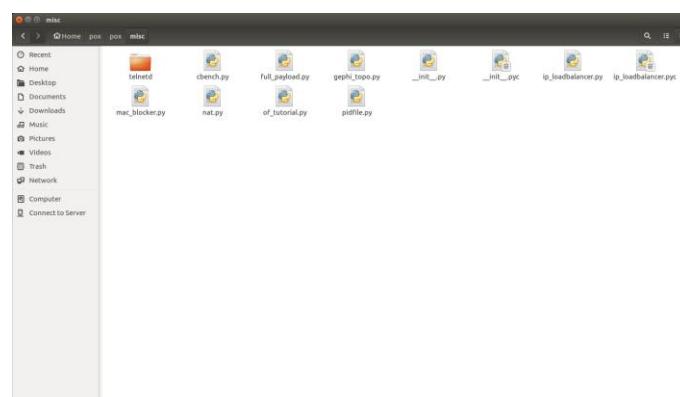
- 5.4 We will use the same IP address and port number as used in the previous configuration. NOTE: make sure the controller type is chosen as “remote controller”.

Device	IP Address	Port Number
c0	127.0.0.1	6633
h1	10.0.0.1	-
h2	10.0.0.2	-
h3	10.0.0.3	-
h4	10.0.0.4	-

- 5.5 Next, we will run the network and start the HTTP server at h1 and h2. To do so, open the h1 and h2 terminals and start a server with port number 80 by issuing the command “python -m http.server 80”.



- 5.6 Now, we already have the servers running, the next step is to start the load balancing mechanism at the POX controller.
- 5.7 To implement load balancing, we will use the stock component named “misc.ip\_loadbalancer”. The code for this component can be found at ~/pox/pox/misc/ip\_loadbalancer.py.



- 5.8 Open the pox directory in the terminal and write the command “sudo ./pox.py misc.ip\_loadbalancer --ip=10.0.10.1 --servers=10.0.0.1,10.0.0.2”. This command starts the pox controller with the load balancing function, and we also define that the devices with IP address 10.0.0.1 and 10.0.0.2 will run as a server, and

the IP address to access that server would be 10.0.10.1. Once the pox controller is up, your window should look like below.

```

anumtalpur@anumtalpur-Precision-T3600: ~/pox
(base) anumtalpur@anumtalpur-Precision-T3600:~/pox$ sudo ./pox.py misc.ip_loadba
lancer --ip=10.0.10.1 --servers=10.0.0.1,10.0.0.2
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:tplb:IP Load Balancer Ready.
INFO:tplb:Load Balancing on [00-00-00-00-00-01 1]
INFO:tplb.00-00-00-00-00-01:Server 10.0.0.1 up
INFO:tplb.00-00-00-00-00-01:Server 10.0.0.2 up

```

5.9 Next, to verify the successful load balancing among the servers by the controller, we will start sending a request from clients named h3 and h4.

5.10 To do so, we will use the Linux command “curl” to send the traffic to the server at IP address 10.0.10.1. At first, we will curl the server once from client h3 and once from client h4.

```

"Node: h3"
(base) root@anumtalpur-Precision-T3600:~# curl 10.0.10.1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01/EN" "http://www.w3.org/TR/html4/st
ric.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="/.adobe/">.adobe/</a></li>
<li><a href="/.anaconda/">.anaconda/</a></li>
<li><a href="/.bash_history">.bash_history</a></li>
<li><a href="/.bash_logout">.bash_logout</a></li>
<li><a href="/.bashrc">.bashrc</a></li>
<li><a href="/.bashrc-anaconda3.bak">.bashrc-anaconda3.bak</a></li>
<li><a href="/.cache/">.cache/</a></li>
<li><a href="/.compiz/">.compiz/</a></li>
<li><a href="/.conda/">.conda/</a></li>
<li><a href="/.condarc">.condarc</a></li>
<li><a href="/.config/">.config</a></li>
<li><a href="/.dbus/">.dbus/</a></li>

```

```

"Node: h4"
(base) root@anumtalpur-Precision-T3600:~# curl 10.0.10.1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01/EN" "http://www.w3.org/TR/html4/st
ric.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="/.adobe/">.adobe/</a></li>
<li><a href="/.anaconda/">.anaconda/</a></li>
<li><a href="/.bash_history">.bash_history</a></li>
<li><a href="/.bash_logout">.bash_logout</a></li>
<li><a href="/.bashrc">.bashrc</a></li>
<li><a href="/.bashrc-anaconda3.bak">.bashrc-anaconda3.bak</a></li>
<li><a href="/.cache/">.cache/</a></li>
<li><a href="/.compiz/">.compiz/</a></li>
<li><a href="/.conda/">.conda/</a></li>
<li><a href="/.condarc">.condarc</a></li>
<li><a href="/.config/">.config</a></li>
<li><a href="/.dbus/">.dbus/</a></li>

```

5.11 We will observe that controller forwards both requests to the server h1

```

"Node: h1"
(base) root@anumtalpur-Precision-T3600:~# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.3 -- [12/Oct/2022 14:12:52] "GET / HTTP/1.1" 200 -
10.0.0.4 -- [12/Oct/2022 14:12:55] "GET / HTTP/1.1" 200 -

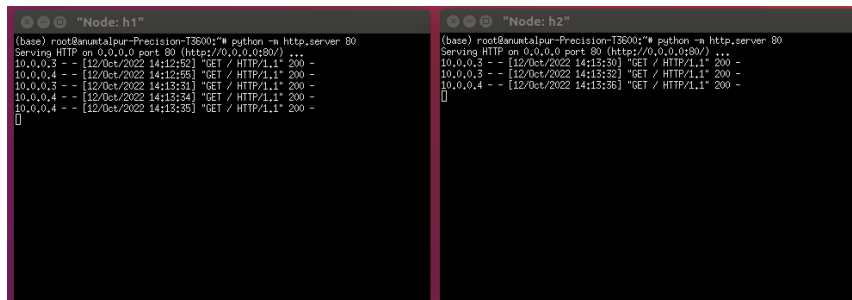
```

```

"Node: h2"
(base) root@anumtalpur-Precision-T3600:~# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...

```

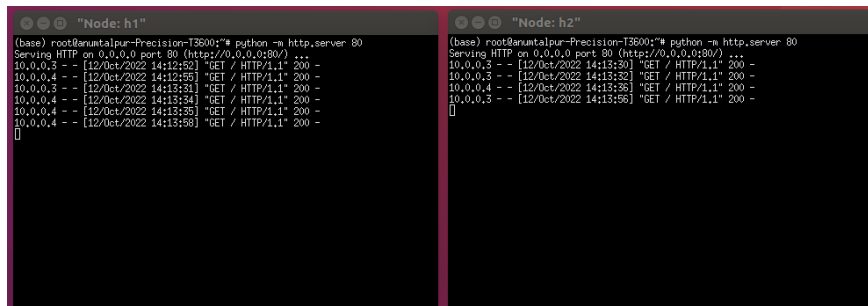
- 5.12 Next, issue 3 more curl requests from each client i.e., h3 and h4. You will observe that this time load is distributed in a way that some clients are forwarded to server h1 and some are forwarded to server h2.



```
(base) root@anumtalpur-Precision-T3600:~# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.3 - - [12/Oct/2022 14:12:52] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:12:56] "GET / HTTP/1.1" 200 -
10.0.0.3 - - [12/Oct/2022 14:13:31] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:13:34] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:13:36] "GET / HTTP/1.1" 200 -
[]

(base) root@anumtalpur-Precision-T3600:~# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.3 - - [12/Oct/2022 14:13:30] "GET / HTTP/1.1" 200 -
10.0.0.3 - - [12/Oct/2022 14:13:32] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:13:36] "GET / HTTP/1.1" 200 -
[]
```

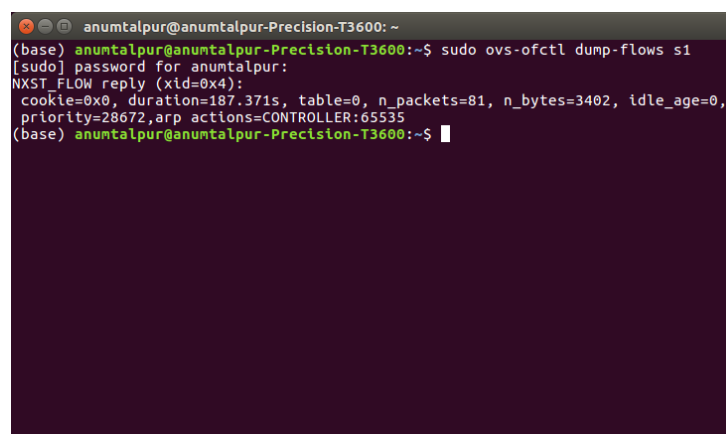
- 5.13 Again, issue one more curl request from each client and observe the pattern on the server side.



```
(base) root@anumtalpur-Precision-T3600:~# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.3 - - [12/Oct/2022 14:12:52] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:12:56] "GET / HTTP/1.1" 200 -
10.0.0.3 - - [12/Oct/2022 14:13:31] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:13:34] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:13:36] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:13:58] "GET / HTTP/1.1" 200 -
[]

(base) root@anumtalpur-Precision-T3600:~# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.3 - - [12/Oct/2022 14:13:30] "GET / HTTP/1.1" 200 -
10.0.0.3 - - [12/Oct/2022 14:13:32] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [12/Oct/2022 14:13:36] "GET / HTTP/1.1" 200 -
10.0.0.3 - - [12/Oct/2022 14:13:56] "GET / HTTP/1.1" 200 -
[]
```

- 5.14 Keep sending more curls and you will notice that the controller is distributing the incoming traffic from clients among both servers to maintain load balance within the network.
- 5.15 You can also observe the flow table over OVS to see how the switch is working by using the dump-flows function of the ovs-ofctl utility.



```
(base) anumtalpur@anumtalpur-Precision-T3600:~$ sudo ovs-ofctl dump-flows s1
[sudo] password for anumtalpur:
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=187.371s, table=0, n_packets=81, n_bytes=3402, idle_age=0,
 priority=28672,arp actions=CONTROLLER:65535
(base) anumtalpur@anumtalpur-Precision-T3600:~$
```

- 5.16 You will notice that as per the flow table entry, OVS is forwarding traffic to the controller where the controller uses a load balancing mechanism to evenly distribute traffic within the network and avoid bottlenecks over any single server.