

Lab 3: SDN Simulation

1. Objectives

- Fully understand the operation of Openflow and observe the operations
- Master the simulation tool mininet

2. Equipment Needs

- Computers
- Internet

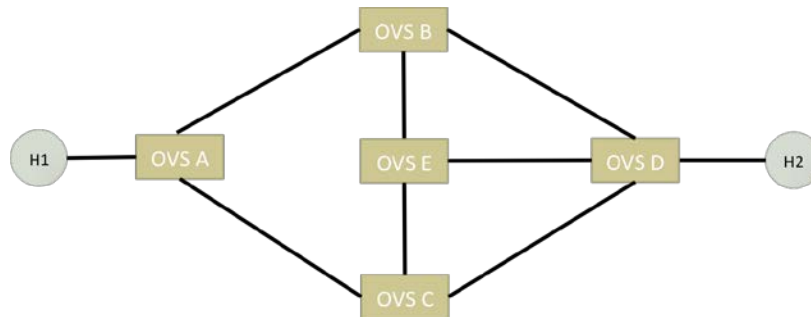
3. Experiments

3.1 Basics

1. Install Ubuntu on your computer, you can install it on a VM using either VMWare player or VirtualBox:
VMWare:
https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/6_0
VirtualBox:
<https://www.virtualbox.org/wiki/Downloads>
2. Follow the instructions to set up Mininet on your Ubuntu VM:
<http://mininet.org/download/>
3. Perform basic simulations following these steps:
<http://mininet.org/walkthrough>
4. Practice writing simulation with python scripts:
<https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>

3.2 Openflow

1. Use Mininet to create the topology below, where A, B, C, D, and E are all Openflow switches.
2. Enforce the following policies so that,
 - a. Traffic from H1 → H2
 - i. HTTP traffic with d_port=80 follows path: A-C-D
 - ii. other traffic follows path: A-B-E-D
 - b. Traffic from H2 → H1
 - i. HTTP traffic with s_port=80, follow path: D-B-A
 - ii. other traffic, follow path: D-C-E-B-A



- c. verify your policies by,
- generating corresponding traffic
 - capturing packets with Wireshark

You can use OVS-OFCTL to manually install rules on switches (preferred method), or you can install a simple controller using RYU/POX/NOX/Beacon (Not recommended for lab 3 you'll do it in lab 4).

- Using mininet, create a 2-stage Fat Tree network using N-port switches, where N is an input parameter to your python script running Mininet simulation. N should be an even number. (You don't need to check the connectivity, just create the topology.)

4. Reports

- (a) A screenshot of OpenFlow control messages you captured with WireShark.

Set filter: openflow

No.	Time	Source	Destination	Protocol	Length	Info
3050	04:51:24.6270362...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3069	04:51:24.6370271...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3071	04:51:24.6370958...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3142	04:51:24.7785899...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3144	04:51:24.7787117...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3146	04:51:24.7787689...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3148	04:51:24.7788050...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3150	04:51:24.7788246...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_HELLO
3152	04:51:24.8108557...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_FEATURES_REQUEST
3154	04:51:24.8108923...	127.0.0.1	127.0.0.1	OpenFlow	80	Type: OFPT_SET_CONFIG
3156	04:51:24.8109974...	127.0.0.1	127.0.0.1	OpenFlow	148	Type: OFPT_FLOW_MOD
3158	04:51:24.8110433...	127.0.0.1	127.0.0.1	OpenFlow	76	Type: OFPT_FEATURES_REQUEST
3160	04:51:24.8110580...	127.0.0.1	127.0.0.1	OpenFlow	80	Type: OFPT_SET_CONFIG
3162	04:51:24.8110752...	127.0.0.1	127.0.0.1	OpenFlow	148	Type: OFPT_FLOW_MOD

Annotations:

- Switches say hello to controller (points to OFPT_HELLO messages)
- Controller collects data from switches (points to OFPT_FEATURES_REQUEST message)
- Controller setup flows for Packet-In (points to OFPT_FLOW_MOD messages)

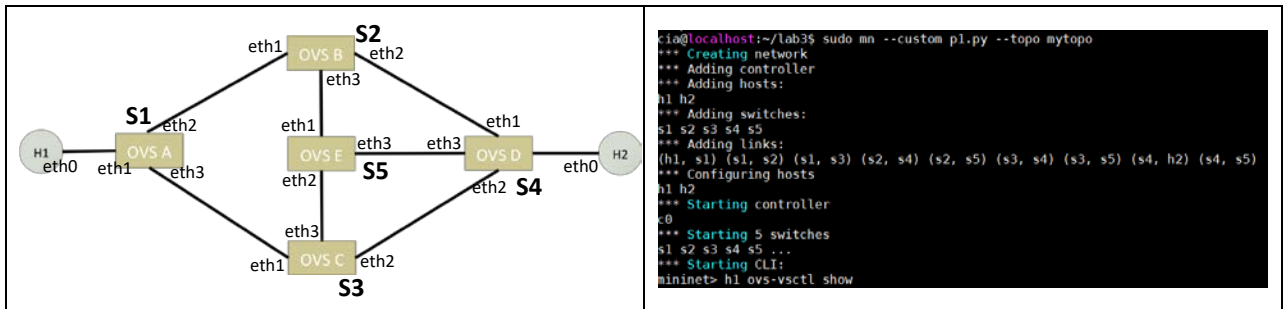
```

root@localhost:~/lab3# ovs-ofctl dump-flows s1
cookie=0x0, duration=1331.600s, table=0, n_packets=1026739, n_bytes=92351522, p
riority=0 actions=CONTROLLER:128
root@localhost:~/lab3#
  
```

- (c) Mininet output while creating the networks

Part2 2-stage fat-tree topology	
Logical	Mininet output
N=4	
<pre> graph TD SW1[SW] --- SW21[SW] SW1 --- SW22[SW] SW1 --- SW23[SW] SW1 --- SW24[SW] SW2 --- SW21 SW2 --- SW22 SW2 --- SW23 SW2 --- SW24 SW21 --- H0_0[H0_0] SW21 --- H0_1[H0_1] SW22 --- H1_0[H1_0] SW22 --- H1_1[H1_1] SW23 --- H2_0[H2_0] SW23 --- H2_1[H2_1] SW24 --- H3_0[H3_0] SW24 --- H3_1[H3_1] </pre>	<pre> *** Starting CLI: mininet> net h0_0 h0_0-eth0:sL0-eth1 h0_1 h0_1-eth0:sL0-eth2 h1_0 h1_0-eth0:sL1-eth1 h1_1 h1_1-eth0:sL1-eth2 h2_0 h2_0-eth0:sL2-eth1 h2_1 h2_1-eth0:sL2-eth2 h3_0 h3_0-eth0:sL3-eth1 h3_1 h3_1-eth0:sL3-eth2 sL0 lo: sL0-eth1:h0_0-eth0 sL0-eth2:h0_1-eth0 sL0-eth3:sS0-eth1 sL0-eth4:sS1-eth1 sL1 lo: sL1-eth1:h1_0-eth0 sL1-eth2:h1_1-eth0 sL1-eth3:sS0-eth2 sL1-eth4:sS1-eth2 sL2 lo: sL2-eth1:h2_0-eth0 sL2-eth2:h2_1-eth0 sL2-eth3:sS0-eth3 sL2-eth4:sS1-eth3 sL3 lo: sL3-eth1:h3_0-eth0 sL3-eth2:h3_1-eth0 sL3-eth3:sS0-eth4 sL3-eth4:sS1-eth4 sS0 lo: sS0-eth1:sL0-eth3 sS0-eth2:sL1-eth3 sS0-eth3:sL2-eth3 sS0-eth4:sL3-eth3 sS1 lo: sS1-eth1:sL0-eth4 sS1-eth2:sL1-eth4 sS1-eth3:sL2-eth4 sS1-eth4:sL3-eth4 mininet> </pre>
N=6	
<pre> graph TD SW1[SW] --- SW21[SW] SW1 --- SW22[SW] SW1 --- SW23[SW] SW1 --- SW24[SW] SW1 --- SW25[SW] SW1 --- SW26[SW] SW2 --- SW21 SW2 --- SW22 SW2 --- SW23 SW2 --- SW24 SW2 --- SW25 SW2 --- SW26 SW21 --- H0_0[H0_0] SW21 --- H0_1[H0_1] SW21 --- H0_2[H0_2] SW22 --- H1_0[H1_0] SW22 --- H1_1[H1_1] SW22 --- H1_2[H1_2] SW23 --- H2_0[H2_0] SW23 --- H2_1[H2_1] SW23 --- H2_2[H2_2] SW24 --- H3_0[H3_0] SW24 --- H3_1[H3_1] SW24 --- H3_2[H3_2] SW25 --- H4_0[H4_0] SW25 --- H4_1[H4_1] SW25 --- H4_2[H4_2] SW26 --- H5_0[H5_0] SW26 --- H5_1[H5_1] SW26 --- H5_2[H5_2] </pre>	<pre> *** Starting CLI: mininet> net h0_0 h0_0-eth0:sL0-eth1 h0_1 h0_1-eth0:sL0-eth2 h0_2 h0_2-eth0:sL0-eth3 h1_0 h1_0-eth0:sL1-eth1 h1_1 h1_1-eth0:sL1-eth2 h1_2 h1_2-eth0:sL1-eth3 h2_0 h2_0-eth0:sL2-eth1 h2_1 h2_1-eth0:sL2-eth2 h2_2 h2_2-eth0:sL2-eth3 h3_0 h3_0-eth0:sL3-eth1 h3_1 h3_1-eth0:sL3-eth2 h3_2 h3_2-eth0:sL3-eth3 h4_0 h4_0-eth0:sL4-eth1 h4_1 h4_1-eth0:sL4-eth2 h4_2 h4_2-eth0:sL4-eth3 h5_0 h5_0-eth0:sL5-eth1 h5_1 h5_1-eth0:sL5-eth2 h5_2 h5_2-eth0:sL5-eth3 sL0 lo: sL0-eth1:h0_0-eth0 sL0-eth2:h0_1-eth0 sL0-eth3:h0_2-eth0 sL0-eth4:h1_0-eth0 sL0-eth5:h1_1-eth0 sL0-eth6:h1_2-eth0 sL1 lo: sL1-eth1:h1_0-eth0 sL1-eth2:h1_1-eth0 sL1-eth3:h1_2-eth0 sL1-eth4:h2_0-eth0 sL1-eth5:h2_1-eth0 sL1-eth6:h2_2-eth0 sL2 lo: sL2-eth1:h2_0-eth0 sL2-eth2:h2_1-eth0 sL2-eth3:h2_2-eth0 sL2-eth4:h3_0-eth0 sL2-eth5:h3_1-eth0 sL2-eth6:h3_2-eth0 sL3 lo: sL3-eth1:h3_0-eth0 sL3-eth2:h3_1-eth0 sL3-eth3:h3_2-eth0 sL3-eth4:h4_0-eth0 sL3-eth5:h4_1-eth0 sL3-eth6:h4_2-eth0 sL4 lo: sL4-eth1:h4_0-eth0 sL4-eth2:h4_1-eth0 sL4-eth3:h4_2-eth0 sL4-eth4:h5_0-eth0 sL4-eth5:h5_1-eth0 sL4-eth6:h5_2-eth0 sL5 lo: sL5-eth1:h5_0-eth0 sL5-eth2:h5_1-eth0 sL5-eth3:h5_2-eth0 sL5-eth4:sS0-eth1 sL5-eth5:sS0-eth2 sL5-eth6:sS0-eth3 sS0 lo: sS0-eth1:sL0-eth3 sS0-eth2:sL1-eth3 sS0-eth3:sL2-eth3 sS0-eth4:sL3-eth3 sS0-eth5:sL4-eth3 sS0-eth6:sL5-eth3 sS1 lo: sS1-eth1:sL0-eth4 sS1-eth2:sL1-eth4 sS1-eth3:sL2-eth4 sS1-eth4:sL3-eth4 sS1-eth5:sL4-eth4 sS1-eth6:sL5-eth4 sS2 lo: sS2-eth1:sL0-eth5 sS2-eth2:sL1-eth5 sS2-eth3:sL2-eth5 sS2-eth4:sL3-eth5 sS2-eth5:sL4-eth5 sS2-eth6:sL5-eth5 sS3 lo: sS3-eth1:sL0-eth6 sS3-eth2:sL1-eth6 sS3-eth3:sL2-eth6 sS3-eth4:sL3-eth6 sS3-eth5:sL4-eth6 sS3-eth6:sL5-eth6 sS4 lo: sS4-eth1:sL1-eth3 sS4-eth2:sL1-eth4 sS4-eth3:sL1-eth5 sS4-eth4:sL1-eth6 sS4-eth5:sL2-eth3 sS4-eth6:sL2-eth4 sS5 lo: sS5-eth1:sL1-eth3 sS5-eth2:sL1-eth4 sS5-eth3:sL1-eth5 sS5-eth4:sL1-eth6 sS5-eth5:sL2-eth3 sS5-eth6:sL2-eth4 sS6 lo: sS6-eth1:sL1-eth3 sS6-eth2:sL1-eth4 sS6-eth3:sL1-eth5 sS6-eth4:sL1-eth6 sS6-eth5:sL2-eth3 sS6-eth6:sL2-eth4 sS7 lo: sS7-eth1:sL1-eth3 sS7-eth2:sL1-eth4 sS7-eth3:sL1-eth5 sS7-eth4:sL1-eth6 sS7-eth5:sL2-eth3 sS7-eth6:sL2-eth4 sS8 lo: sS8-eth1:sL1-eth3 sS8-eth2:sL1-eth4 sS8-eth3:sL1-eth5 sS8-eth4:sL1-eth6 sS8-eth5:sL2-eth3 sS8-eth6:sL2-eth4 sS9 lo: sS9-eth1:sL1-eth3 sS9-eth2:sL1-eth4 sS9-eth3:sL1-eth5 sS9-eth4:sL1-eth6 sS9-eth5:sL2-eth3 sS9-eth6:sL2-eth4 sS10 lo: sS10-eth1:sL1-eth3 sS10-eth2:sL1-eth4 sS10-eth3:sL1-eth5 sS10-eth4:sL1-eth6 sS10-eth5:sL2-eth3 sS10-eth6:sL2-eth4 sS11 lo: sS11-eth1:sL1-eth3 sS11-eth2:sL1-eth4 sS11-eth3:sL1-eth5 sS11-eth4:sL1-eth6 sS11-eth5:sL2-eth3 sS11-eth6:sL2-eth4 sS12 lo: sS12-eth1:sL1-eth3 sS12-eth2:sL1-eth4 sS12-eth3:sL1-eth5 sS12-eth4:sL1-eth6 sS12-eth5:sL2-eth3 sS12-eth6:sL2-eth4 sS13 lo: sS13-eth1:sL1-eth3 sS13-eth2:sL1-eth4 sS13-eth3:sL1-eth5 sS13-eth4:sL1-eth6 sS13-eth5:sL2-eth3 sS13-eth6:sL2-eth4 sS14 lo: sS14-eth1:sL1-eth3 sS14-eth2:sL1-eth4 sS14-eth3:sL1-eth5 sS14-eth4:sL1-eth6 sS14-eth5:sL2-eth3 sS14-eth6:sL2-eth4 sS15 lo: sS15-eth1:sL1-eth3 sS15-eth2:sL1-eth4 sS15-eth3:sL1-eth5 sS15-eth4:sL1-eth6 sS15-eth5:sL2-eth3 sS15-eth6:sL2-eth4 sS16 lo: sS16-eth1:sL1-eth3 sS16-eth2:sL1-eth4 sS16-eth3:sL1-eth5 sS16-eth4:sL1-eth6 sS16-eth5:sL2-eth3 sS16-eth6:sL2-eth4 sS17 lo: sS17-eth1:sL1-eth3 sS17-eth2:sL1-eth4 sS17-eth3:sL1-eth5 sS17-eth4:sL1-eth6 sS17-eth5:sL2-eth3 sS17-eth6:sL2-eth4 sS18 lo: sS18-eth1:sL1-eth3 sS18-eth2:sL1-eth4 sS18-eth3:sL1-eth5 sS18-eth4:sL1-eth6 sS18-eth5:sL2-eth3 sS18-eth6:sL2-eth4 sS19 lo: sS19-eth1:sL1-eth3 sS19-eth2:sL1-eth4 sS19-eth3:sL1-eth5 sS19-eth4:sL1-eth6 sS19-eth5:sL2-eth3 sS19-eth6:sL2-eth4 sS20 lo: sS20-eth1:sL1-eth3 sS20-eth2:sL1-eth4 sS20-eth3:sL1-eth5 sS20-eth4:sL1-eth6 sS20-eth5:sL2-eth3 sS20-eth6:sL2-eth4 sS21 lo: sS21-eth1:sL1-eth3 sS21-eth2:sL1-eth4 sS21-eth3:sL1-eth5 sS21-eth4:sL1-eth6 sS21-eth5:sL2-eth3 sS21-eth6:sL2-eth4 sS22 lo: sS22-eth1:sL1-eth3 sS22-eth2:sL1-eth4 sS22-eth3:sL1-eth5 sS22-eth4:sL1-eth6 sS22-eth5:sL2-eth3 sS22-eth6:sL2-eth4 sS23 lo: sS23-eth1:sL1-eth3 sS23-eth2:sL1-eth4 sS23-eth3</pre>

Part1 Topology	
Logical	Mininet output



Part2 2-stage fat-tree topology

<p>Logical</p>	<p>Mininet output</p>
<p>N=4</p>	<pre> ia@localhost:~/lab3\$ sudo python p2.py 4 (sudo) password for cia: *** Creating network *** Adding hosts: h0_0 h0_1 h1_0 h1_1 h2_0 h2_1 h3_0 h3_1 *** Adding switches: sL0 sL1 sL2 sL3 sS0 sS1 *** Adding links: (h0_0, sL0) (h0_1, sL0) (h1_0, sL1) (h1_1, sL1) (h2_0, sL2) (h2_1, sL2) (h3_0, sL3) (h3_1, sL3) (sL0, sS0) (sL0, sS1) (sL1, sS0) (sL1, sS1) (sL2, sS0) (sL2, sS1) (sL3, sS0) (sL3, sS1) *** Configuring hosts h0_0 h0_1 h1_0 h1_1 h2_0 h2_1 h3_0 h3_1 *** Starting controller *** Starting 6 switches sL0 sL1 sL2 sL3 sS0 sS1 ... Waiting for links to setup *** Running CLI *** Starting CLI: mininet> net h0_0 h0_0-eth0:sL0-eth1 h0_1 h0_1-eth0:sL0-eth2 </pre>
<p>N=6</p>	<pre> ia@localhost:~/lab3\$ sudo python p2.py 8 *** Creating network *** Adding hosts: h0_0 h0_1 h0_2 h0_3 h1_0 h1_1 h1_2 h1_3 h2_0 h2_1 h2_2 h2_3 h3_0 h3_1 h3_2 h3_3 h4_0 h4_1 h4_2 h4_3 h5_0 h5_1 h5_2 h5_3 h6_0 h6_1 h6_2 h6_3 h7_0 h7_1 h7_2 h7_3 *** Adding switches: sL0 sL1 sL2 sL3 sL4 sL5 sL6 sL7 sS0 sS1 sS2 sS3 *** Adding links: (h0_0, sL0) (h0_1, sL0) (h0_2, sL0) (h0_3, sL0) (h1_0, sL1) (h1_1, sL1) (h1_2, sL1) (h1_3, sL1) (h2_0, sL2) (h2_1, sL2) (h2_2, sL2) (h2_3, sL2) (h3_0, sL3) (h3_1, sL3) (h3_2, sL3) (h3_3, sL3) (h4_0, sL4) (h4_1, sL4) (h4_2, sL4) (h4_3, sL4) (h5_0, sL5) (h5_1, sL5) (h5_2, sL5) (h5_3, sL5) (h6_0, sL6) (h6_1, sL6) (h6_2, sL6) (h6_3, sL6) (h7_0, sL7) (h7_1, sL7) (h7_2, sL7) (h7_3, sL7) (sL0, sS0) (sL0, sS1) (sL0, sS2) (sL0, sS3) (sL1, sS0) (sL1, sS1) (sL1, sS2) (sL1, sS3) (sL2, sS0) (sL2, sS1) (sL2, sS2) (sL2, sS3) (sL3, sS0) (sL3, sS1) (sL3, sS2) (sL3, sS3) (sL4, sS0) (sL4, sS1) (sL4, sS2) (sL4, sS3) (sL5, sS0) (sL5, sS1) (sL5, sS2) (sL5, sS3) (sL6, sS0) (sL6, sS1) (sL6, sS2) (sL6, sS3) (sL7, sS0) (sL7, sS1) (sL7, sS2) (sL7, sS3) *** Configuring hosts h0_0 h0_1 h0_2 h0_3 h1_0 h1_1 h1_2 h1_3 h2_0 h2_1 h2_2 h2_3 h3_0 h3_1 h3_2 h3_3 h4_0 h4_1 h4_2 h4_3 h5_0 h5_1 h5_2 h5_3 h6_0 h6_1 h6_2 h6_3 h7_0 h7_1 h7_2 h7_3 *** Starting controller *** Starting 12 switches sL0 sL1 sL2 sL3 sL4 sL5 sL6 sL7 sS0 sS1 sS2 sS3 ... Waiting for links to setup *** Running CLI *** Starting CLI: mininet> net h0_0 h0_0-eth0:sL0-eth1 h0_1 h0_1-eth0:sL0-eth2 </pre>

- (d) Briefly explain how you produce different traffic to verify whether the rules installed function correctly.

To produce different traffic, we use “iperf” with specific port number.

First, we open xterm of the two hosts. On each host, we set up server and client, and then use iperf with different port number.

Host 1	Host 2
--------	--------

"Node: h1"@localhost.localdomain

```
root@localhost:~/lab3# iperf -c 10.0.0.2 -p 80
Client connecting to 10.0.0.2, TCP port 80
TCP window size: 85.3 KByte (default)
[ 21] local 10.0.0.1 port 47380 connected with 10.0.0.2 port 80
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-10.0 sec  5.54 GBytes  4.76 Gbits/sec
root@localhost:~/lab3# iperf -c 10.0.0.2 -p 6666
Client connecting to 10.0.0.2, TCP port 6666
TCP window size: 85.3 KByte (default)
[ 21] local 10.0.0.1 port 44846 connected with 10.0.0.2 port 6666
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-10.0 sec  3.90 GBytes  3.35 Gbits/sec
root@localhost:~/lab3#
```

Send to 80 port

Send to 6666 port

"Node: h2"@localhost.localdomain

```
root@localhost:~/lab3# iperf -s -p 80
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
[ 22] local 10.0.0.2 port 80 connected with 10.0.0.1 port 47380
[ ID] Interval      Transfer    Bandwidth
[ 22] 0.0-10.0 sec  5.54 GBytes  4.75 Gbits/sec
root@localhost:~/lab3# iperf -s -p 6666
Server listening on TCP port 6666
TCP window size: 85.3 KByte (default)
[ 22] local 10.0.0.2 port 6666 connected with 10.0.0.1 port 44846
[ ID] Interval      Transfer    Bandwidth
[ 22] 0.0-9.9 sec   3.90 GBytes  3.37 Gbits/sec
root@localhost:~/lab3#
```

Listen 80 port

Listen 6666 port

a. Traffic from H1 → H2

- HTTP traffic with **d_port=80** follows path: A-C-D
- other traffic follows path: A-B-E-D

b. Traffic from H2 → H1

- HTTP traffic with **s_port=80**, follow path: D-B-A
- other traffic, follow path: D-C-E-B-A

Test by: setting h1 as client, h2 as server on 80 port

(e) With the produced traffic, show the screenshots of Wireshark capture on different links (switch with interface) to verify the paths taken by different traffic are correct.

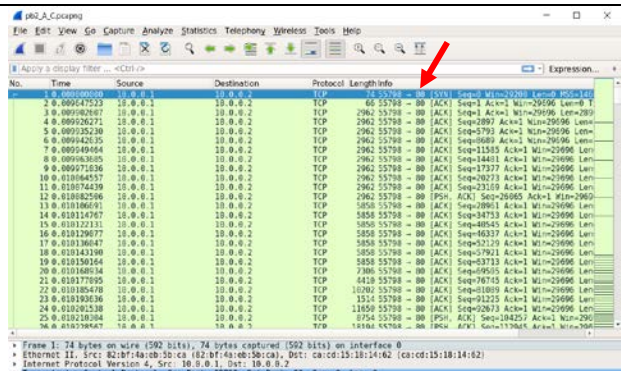
Before using Wireshark, we can first dump the flows to check if the flow setting is correct:

This is the flows of OVS D, which connects to host 2. From the screenshot, we can see that every flow is successfully matched.

```
cia@localhost:~$ sudo ovs-ofctl dump-flows s4
cookie=0x0, duration=1146.161s, table=0, n_packets=134194, n_bytes=5958347532, priority=2,tcp,in_port="s4-eth2",tp_dst=80 actions=output:"s4-eth4"
cookie=0x0, duration=1146.157s, table=0, n_packets=126445, n_bytes=8345402, priority=2,tcp,in_port="s4-eth4",tp_src=80 actions=output:"s4-eth1"
cookie=0x0, duration=1146.233s, table=0, n_packets=1591615, n_bytes=4299844866, priority=1,in_port="s4-eth3" actions=output:"s4-eth4"
cookie=0x0, duration=1146.196s, table=0, n_packets=79928, n_bytes=5275248, priority=1,in_port="s4-eth4" actions=output:"s4-eth2"
cookie=0x0, duration=1151.512s, table=0, n_packets=749551, n_bytes=52468570, priority=0 actions=CONTROLLER:128
```

Verify the paths using Wireshark:

By looking into the interfaces that only one kind of the flow route through, we can observe whether the paths taken by different traffic are correct.

Detail	Screenshots
<p>Traffic from H1 → H2</p> <p>HTTP traffic with d_port=80 follows path: A-C-D</p> <p>Measure the <u>interface of switch C</u> connecting to A. In the screenshot, every TCP flow with d_port=80 routes through this interface.</p> <p>Other flows does not route through this interface.</p>	 <p>The screenshot shows a Wireshark capture on interface eth0. A red arrow points to the filter field where 'eth0' is entered. The packet list shows multiple TCP flows with destination port 80, which are highlighted in green, indicating they match the filter. The packet details pane shows the selected packet's structure, including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol.</p>

Traffic from H1 → H2

other traffic follows path: A-B-E-D

Measure the interface of switch D connecting to E.
In the screenshot, every TCP flow with `d_port=6666` routes through this interface.

Other flows does not route through this interface.

The screenshot shows a Wireshark packet capture on interface 0. The packet list shows multiple TCP flows. The packet details pane for the selected packet shows the destination port is 6666. The packet bytes pane shows the raw data of the packet.

Traffic from H2 → H1

HTTP traffic with `s_port=80`, follow path: D-B-A

Measure the interface of switch B connecting to D.
In the screenshot, every TCP flow with `s_port=80` routes through this interface.

Other flows does not route through this interface.

The screenshot shows a Wireshark packet capture on interface 0. The packet list shows multiple TCP flows. The packet details pane for the selected packet shows the source port is 80. The packet bytes pane shows the raw data of the packet.

Traffic from H2 → H1

other traffic, follow path: D-C-E-B-A

Measure the interface of switch E connecting to C.
In the screenshot, every TCP flow with `s_port=6666` routes through this interface.

Other flows does not route through this interface.

The screenshot shows a Wireshark packet capture on interface 0. The packet list shows multiple TCP flows. The packet details pane for the selected packet shows the source port is 6666. The packet bytes pane shows the raw data of the packet.

- (f) OVS-OFCTL commands used to install the rules on switches. (If you use a controller, upload your controller program)

By using the command “`sudo mn --custom p1.py --topo mytopo`”, we need to set up the rules manually with OVS-OFCTL in CLI mode. Here are the commands:

Traffic from H1 → H2	
HTTP traffic with <code>d_port=80</code> , follow path: A-C-D	<code>ovs-ofctl add-flow s1 priority=2,in_port=1,tcp,tcp_dst=80,actions=output:3</code> <code>ovs-ofctl add-flow s3 priority=2,in_port=1,tcp,tcp_dst=80,actions=output:2</code> <code>ovs-ofctl add-flow s4 priority=2,in_port=2,tcp,tcp_dst=80,actions=output:4</code>
other traffic, follow path: A-B-E-D	<code>ovs-ofctl add-flow s1 priority=1,in_port=1,actions=output:2</code> <code>ovs-ofctl add-flow s2 priority=1,in_port=1,actions=output:3</code>

	ovs-ofctl add-flow s5 priority=1,in_port=1,actions=output:3 ovs-ofctl add-flow s4 priority=1,in_port=3,actions=output:4
Traffic from H2 → H1	
HTTP traffic with s_port=80, follow path: D-B-A	ovs-ofctl add-flow s4 priority=2 ,in_port=4,tcp,tcp_src=80,actions=output:1 ovs-ofctl add-flow s2 priority=2 ,in_port=2,tcp,tcp_src=80,actions=output:1 ovs-ofctl add-flow s1 priority=2 ,in_port=2,tcp,tcp_src=80,actions=output:1
other traffic, follow path: D-C-E-B-A	ovs-ofctl add-flow s4 priority=1,in_port=4,actions=output:2 ovs-ofctl add-flow s3 priority=1,in_port=2,actions=output:3 ovs-ofctl add-flow s5 priority=1,in_port=2,actions=output:1 ovs-ofctl add-flow s2 priority=1,in_port=3,actions=output:1 ovs-ofctl add-flow s1 priority=1,in_port=2,actions=output:1

By using the command “sudo python pb1.py”, we write a function that can automatically set up the rules:

def AutoSetFlows():

```

# Other flow (low priority) h1 -> h2
cmd = "ovs-ofctl add-flow s1 priority=1,in_port=1,actions=output:2"
os.system(cmd)
cmd = "ovs-ofctl add-flow s2 priority=1,in_port=1,actions=output:3"
os.system(cmd)
cmd = "ovs-ofctl add-flow s5 priority=1,in_port=1,actions=output:3"
os.system(cmd)
cmd = "ovs-ofctl add-flow s4 priority=1,in_port=3,actions=output:4"
os.system(cmd)

# Other flow (low priority) h2 -> h1
cmd = "ovs-ofctl add-flow s4 priority=1,in_port=4,actions=output:2"
os.system(cmd)
cmd = "ovs-ofctl add-flow s3 priority=1,in_port=2,actions=output:3"
os.system(cmd)
cmd = "ovs-ofctl add-flow s5 priority=1,in_port=2,actions=output:1"
os.system(cmd)
cmd = "ovs-ofctl add-flow s2 priority=1,in_port=3,actions=output:1"
os.system(cmd)
cmd = "ovs-ofctl add-flow s1 priority=1,in_port=2,actions=output:1"
os.system(cmd)

# Port = 80, h1 -> h2
cmd = "ovs-ofctl add-flow s1 priority=2,in_port=1,tcp,tcp_dst=80,actions=output:3"
os.system(cmd)
cmd = "ovs-ofctl add-flow s3 priority=2,in_port=1,tcp,tcp_dst=80,actions=output:2"
os.system(cmd)
cmd = "ovs-ofctl add-flow s4 priority=2,in_port=2,tcp,tcp_dst=80,actions=output:4"
os.system(cmd)

# Port = 80, h2 -> h1

```

```

cmd = "ovs-ofctl add-flow s4 priority=2,in_port=4,tcp,tcp_src=80,actions=output:1"
os.system(cmd)
cmd = "ovs-ofctl add-flow s2 priority=2,in_port=2,tcp,tcp_src=80,actions=output:1"
os.system(cmd)
cmd = "ovs-ofctl add-flow s1 priority=2,in_port=2,tcp,tcp_src=80,actions=output:1"
os.system(cmd)

```

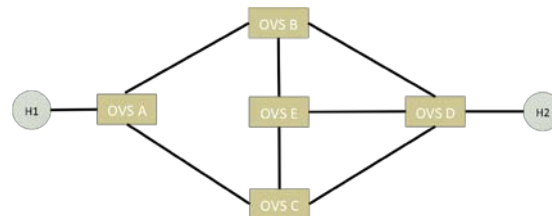
(g) Also submit all your python files used in this lab (do NOT paste code in report).

The python files include two files: **p1.py** and **p2.py**.

p1.py is the implementation of the topology.

Usage:

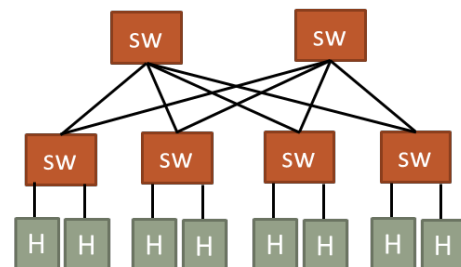
1. `sudo mn --custom p1.py --topo mytopo`
(Need to manually set up the rules)
2. `sudo python pb1.py`
(Automatically set up the rules)



p2.py is the implementation of the 2-stage fat tree topology.

Usage:

1. `sudo python pb2.py <N>`
(ex. "`sudo python pb2.py 6`")



We have zero tolerance to forged or fabricated data!! A single piece of forged/fabricated data would bring the total score down to zero.