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

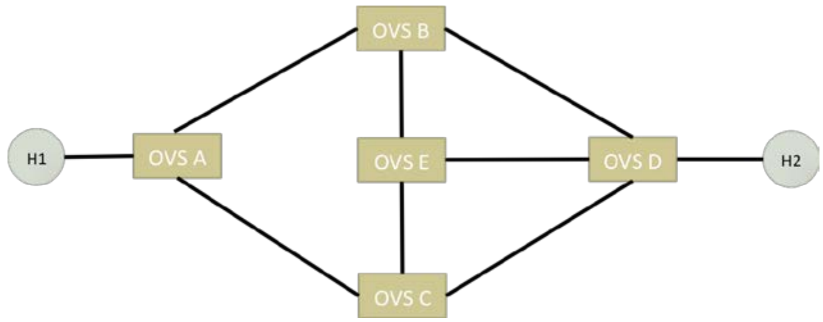
3. Experiments

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



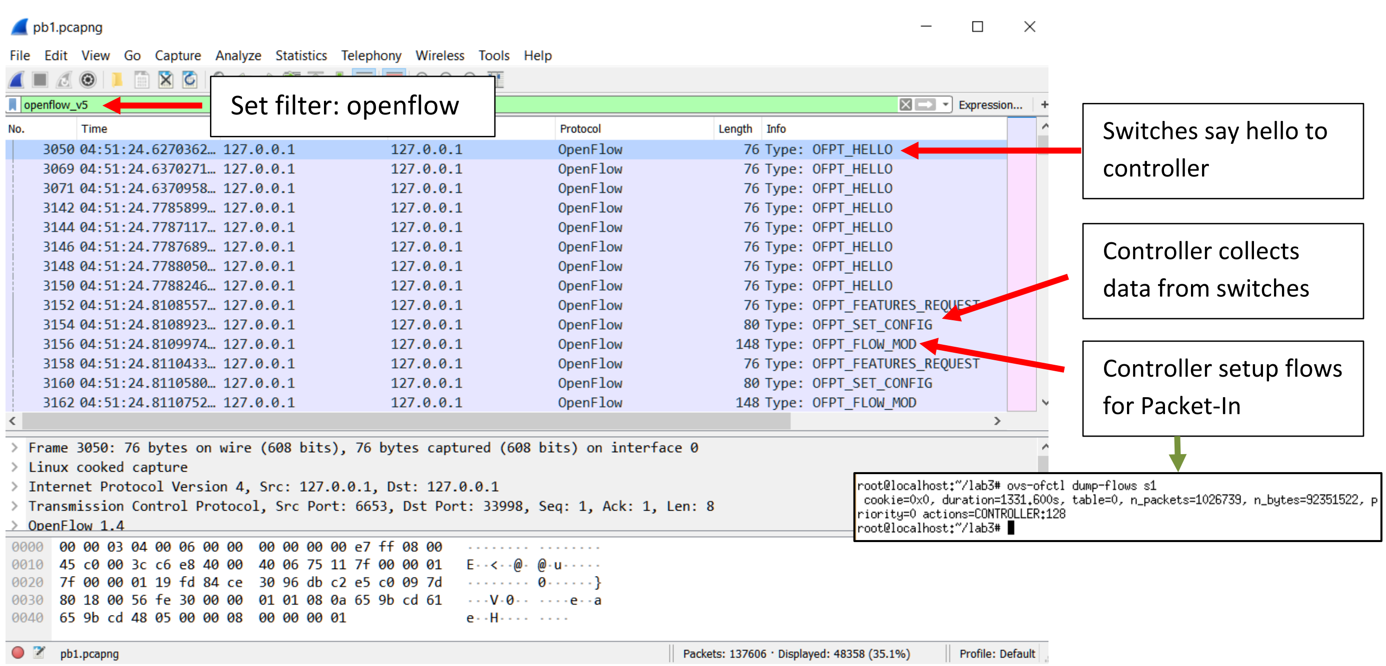
c. verify your policies by,
   i. generating corresponding traffic
   ii. capturing packets with Wireshark

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).

3. 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.)

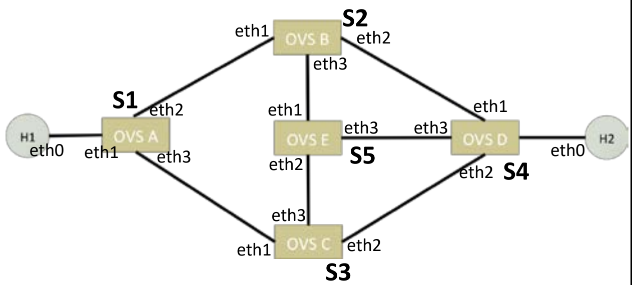
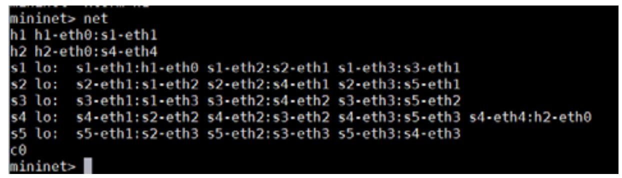
4. Reports

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



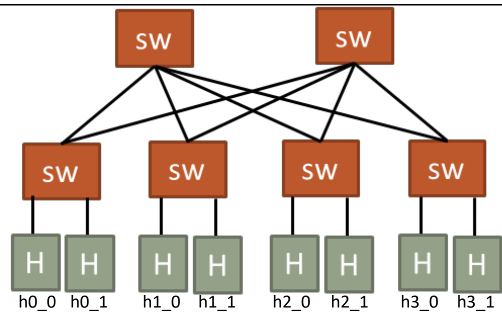
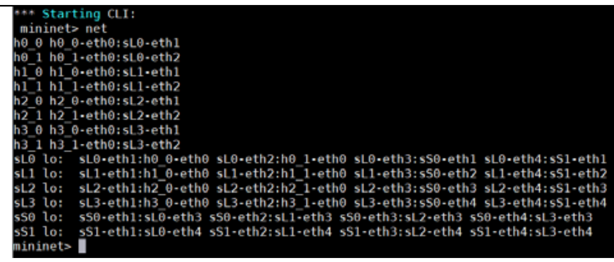
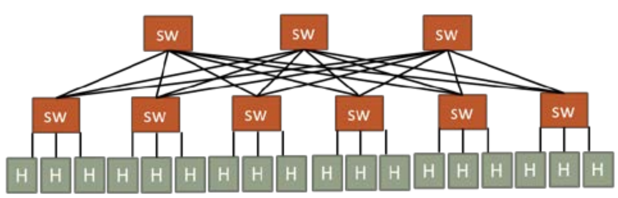
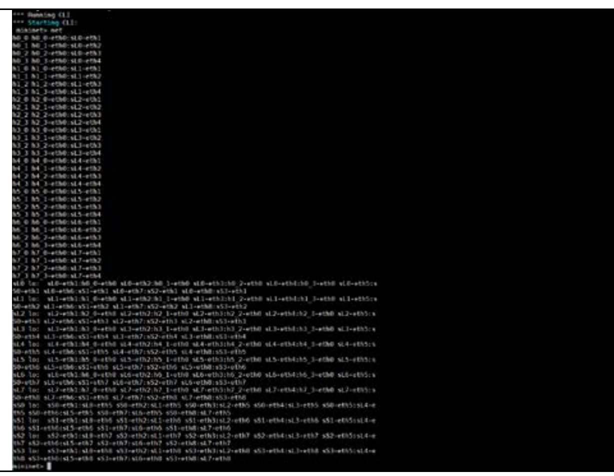
(b) Output of mininet “net” command for both topologies. (you can use any N for Fat-tree, ex: 4, 6, 8)

Part1 Topology

Logical	Mininet output
	


```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s4-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1 s1-eth3:s3-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:s4-eth1 s2-eth3:s5-eth1
s3 lo:  s3-eth1:s1-eth3 s3-eth2:s4-eth2 s3-eth3:s5-eth2
s4 lo:  s4-eth1:s2-eth2 s4-eth2:s3-eth2 s4-eth3:s5-eth3 s4-eth4:h2-eth0
s5 lo:  s5-eth1:s2-eth3 s5-eth2:s3-eth3 s5-eth3:s4-eth3
c0
mininet>
```

Part2 2-stage fat-tree topology

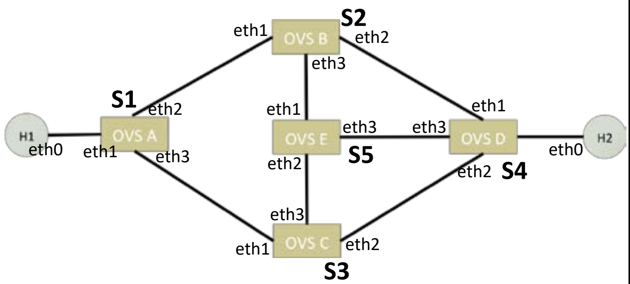
Logical	Mininet output
N=4	
	
N=6	
	

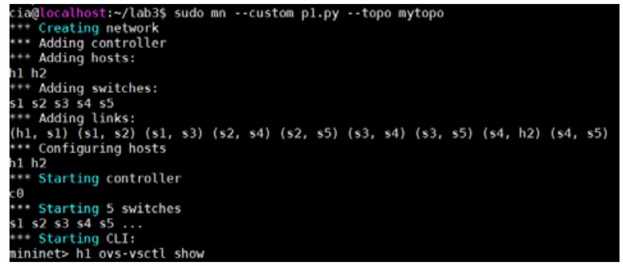

```
*** Starting CLI:
 mininet> net
h0_0 h0_0-eth0:sL0-eth1
h0_1 h0_1-eth0:sL0-eth2
h1_0 h1_0-eth0:sL1-eth1
h1_1 h1_1-eth0:sL1-eth2
h2_0 h2_0-eth0:sL2-eth1
h2_1 h2_1-eth0:sL2-eth2
h3_0 h3_0-eth0:sL3-eth1
h3_1 h3_1-eth0:sL3-eth2
sL0 lo:  sL0-eth1:h0_0-eth0 sL0-eth2:h0_1-eth0 sL0-eth3:sS0-eth1 sL0-eth4:sS1-eth1
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>
```


(c) Mininet output while creating the networks

Part1 Topology

Logical	Mininet output



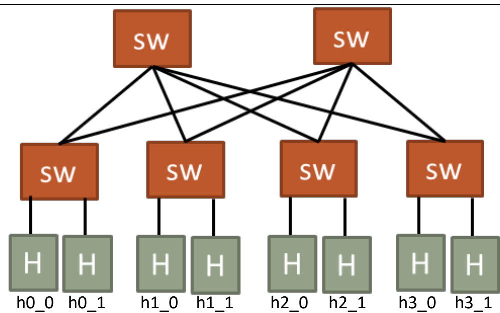


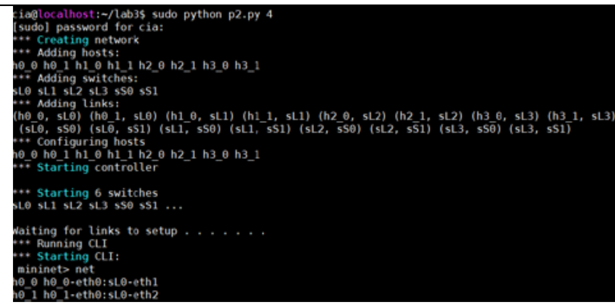
```
cia@localhost:~/lab3$ sudo mn --custom p1.py --topo mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (s1, s2) (s1, s3) (s2, s4) (s2, s5) (s3, s4) (s3, s5) (s4, h2) (s4, s5)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> h1 ovs-vsctl show
```

Part2 2-stage fat-tree topology

Logical	Mininet output

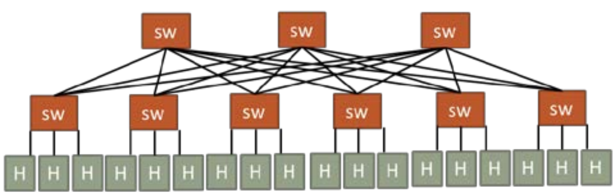
N=4

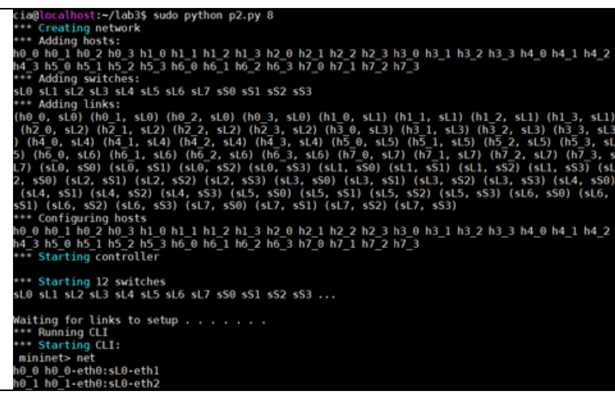




```
cia@localhost:~/lab3$ sudo python p2.py 4
[sudo] password for cia:
*** Creating network
*** Adding hosts:
h0_0 h0_1 h1_0 h1_1 h2_0 h2_1 h3_0 h3_1
*** Adding switches:
sL0 sL1 sL2 sL3 sS0 sS1
*** Adding links:
(h0_0, sL0) (h0_1, sL0) (h1_0, sL1) (h1_1, sL1) (h2_0, sL2) (h2_1, sL2) (h3_0, sL3) (h3_1, 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
```

N=6





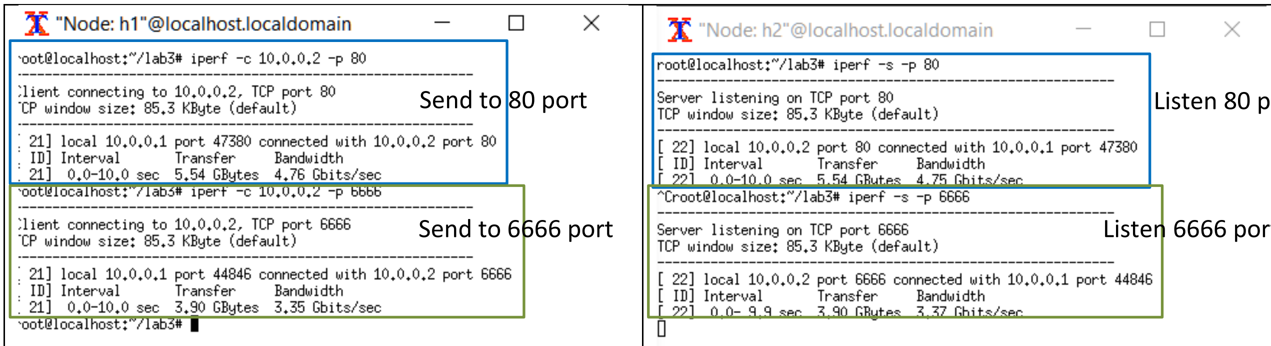
```
cia@localhost:~/lab3$ sudo python p2.py 8
*** Creating network
*** Adding hosts:
h0_0 h0_1 h0_2 h0_3 h1_0 h1_1 h1_2 h1_3 h2_0 h2_1 h2_2 h2_3 h3_0 h3_1 h3_2 h3_3 h4_0 h4_1 h4_2
h4_3 h5_0 h5_1 h5_2 h5_3 h6_0 h6_1 h6_2 h6_3 h7_0 h7_1 h7_2 h7_3
*** Adding switches:
sL0 sL1 sL2 sL3 sL4 sL5 sL6 sL7 sS0 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, sL
5) (h6_0, sL6) (h6_1, sL6) (h6_2, sL6) (h6_3, sL6) (h7_0, sL7) (h7_1, sL7) (h7_2, sL7) (h7_3, s
L7) (sL0, sS0) (sL0, sS1) (sL0, sS2) (sL0, sS3) (sL1, sS0) (sL1, sS1) (sL1, sS2) (sL1, sS3) (sL
2, 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
```

(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



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

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.

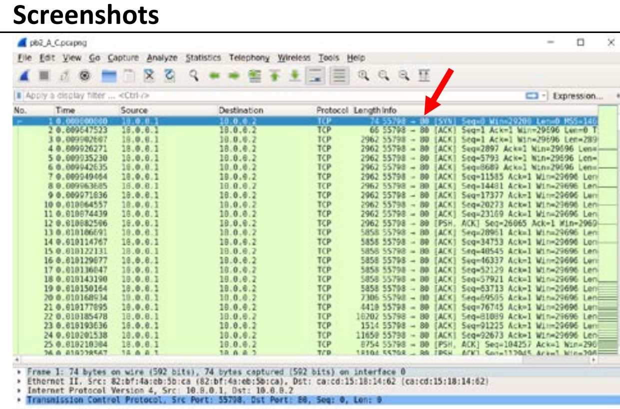
Before using Wireshark, we can first dump the flows to check if the flow setting is correct:

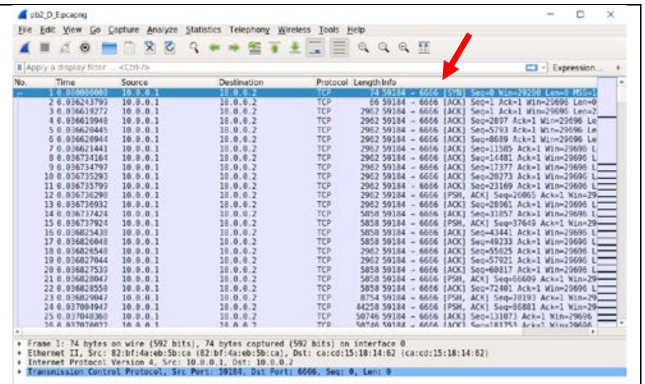
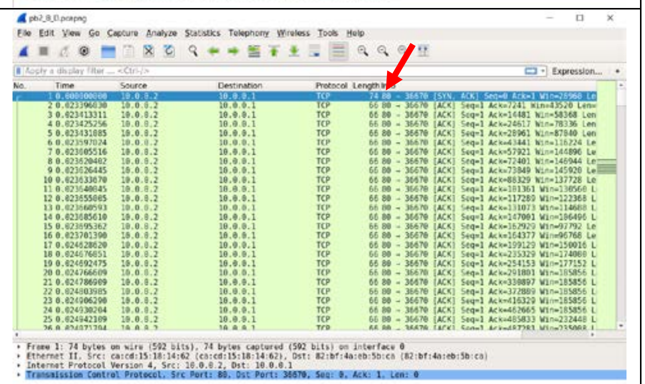
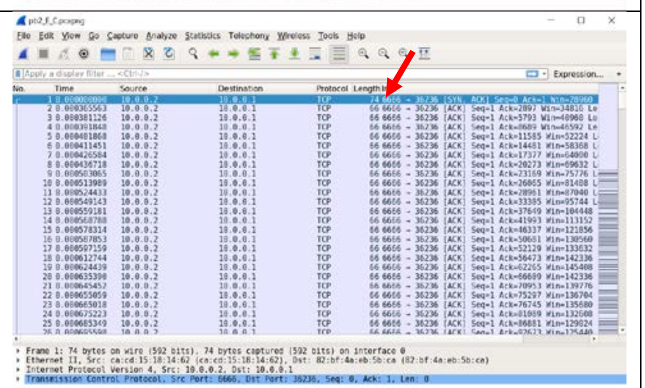
This is the flows of OVS D, which connects to host 2. From the screenshot, we can see that every flow is successfully matched.

```
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**
Traffic from H1 ➔ H2 HTTP traffic with **d_port**=80 follows path: A-**C**-D 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. Other flows does not route through this interface.	

Traffic from H1 ➔ H2 other traffic follows path: A-B-E-**D** Measure the <u>interface of switch D</u> 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.	
Traffic from H2 ➔ H1 HTTP traffic with **s_port**=80, follow path: D-**B**-A Measure the <u>interface of switch B</u> 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.	
Traffic from H2 ➔ H1 other traffic, follow path: D-C-**E**-B-A Measure the <u>interface of switch E</u> 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.	

(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 commends:

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

	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)
```

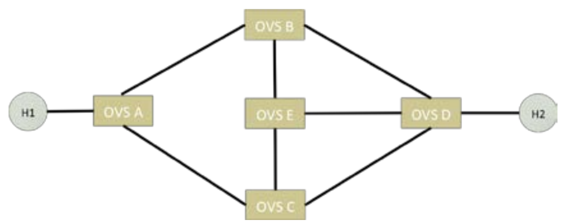
(g) Also submit all your python files used in this lab (do NOT paste code in report).

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

p1.py is the implementation of the topology.

Usage:

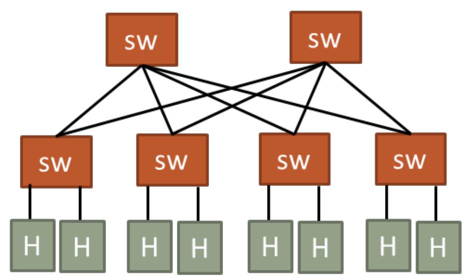
1. sudo mn --custom p1.py --topo mytopo
   (Need to manually set up the rules)
2. sudo python pb1.py
   (Automatically set up the rules)



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.