Lab3 Report: SDN Open Virtual Switches

** Please **fill in the report** and submit the **pdf** to NYUClasses*

Name: ________________________ ID: ___________________ Date: _____________________

1. Objectives

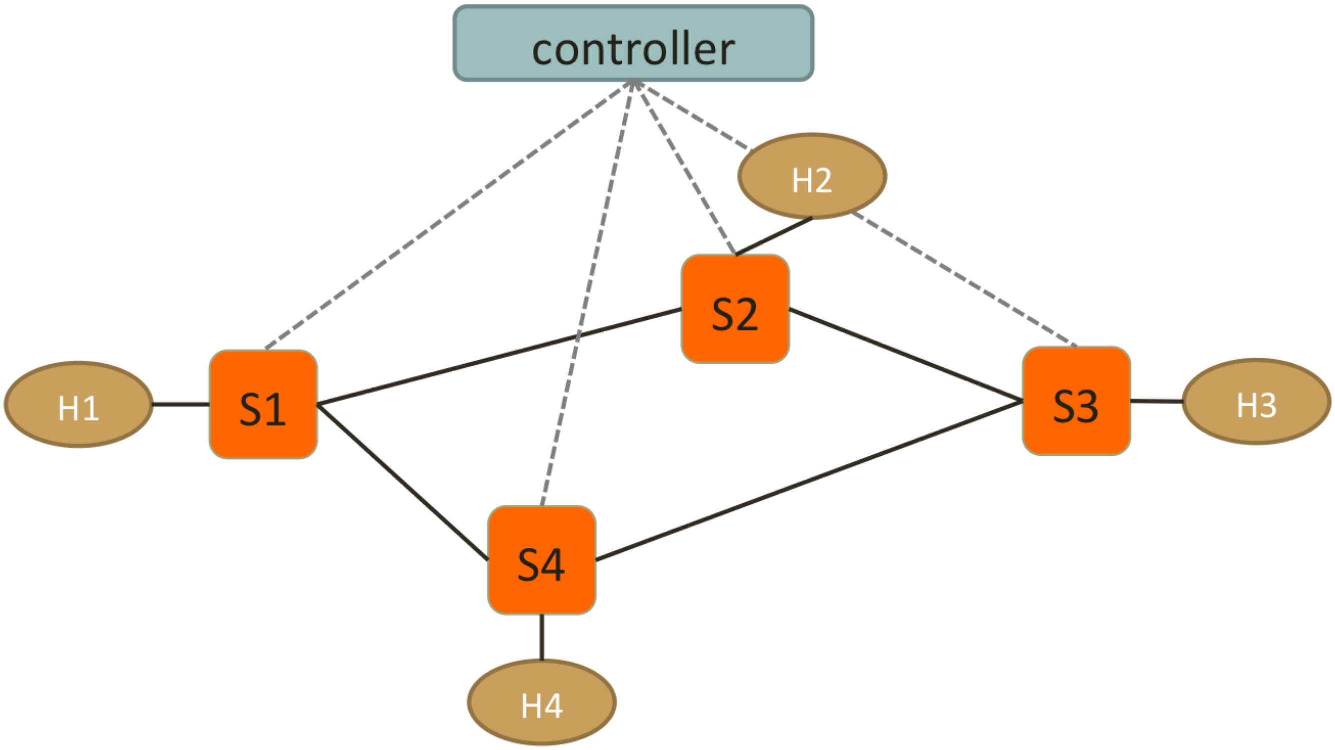
- Understand SDN and get familiar with controllers.

2. References

- https://github.com/faucetsdn/ryu/blob/master/ryu/app/simple_switch_13.py
- https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html
- Slides

3. Experiments

1. Use Mininet to create the following topology: (4 Hosts, 4 OVSes) with a remote controller
2. Use RYU to implement the controller (you can use other controller such as BEACON, POX, etc...)



3. Test Connectivity using ping. (Hint: take care of ARP packets in the controller and install proper rules for them.)
4. Enforce *these policies*:
   - **Everything follows shortest path**
   - **When there are two shortest paths with equal costs available**
     - ICMP and TCP packets take the clockwise path OK
       - e.g. S1-S2-S3, S2-S3-S4 OK
     - UDP packets take the counterclockwise path OK
       - e.g. S1-S4-S3, S2-S1-S4 OK
     - H2 and H4 cannot send HTTP traffic (TCP with dst_port:80) OK
       - New connections are dropped with a TCP RST sent back to **H2 or H4** OK
       - To be more specific, when the first TCP packet (SYN) arrives **S2 or S4**, forwarded it to controller, controller then create a RST packet and send it back to the host. OK
     - H1 and H4 cannot send UDP traffic OK
       - simply drop packets at switches OK

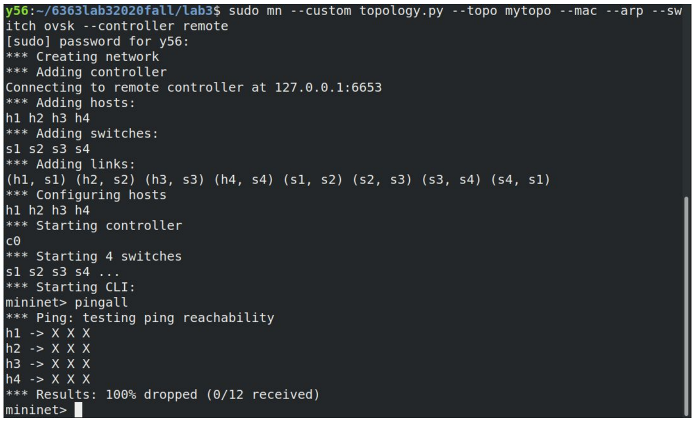
Important! Handle the flow rules in Packet-In and let the controller handles the rules dynamically.

If you use static rules for those policies or handle them in SwitchFeatureHandler, your lab score will be removed.

4. Reports

(a) Screenshots of your mininet with "pingall", **before** and **after starting the controller**.

```
sudo mn --custom topology.py --topo mytopo --mac --arp --switch ovsk --controller remote
```



```
y56:~/6363lab32020fall/lab3$ sudo mn --custom topology.py --topo mytopo --mac --arp --sw
itch ovsk --controller remote
[sudo] password for y56:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s1, s2) (s2, s3) (s3, s4) (s4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
mininet>
```

```
sh ./flow.sh
```

where flow.sh is

```
sudo ovs-vsctl set Bridge s1 protocols=OpenFlow13
sudo ovs-vsctl set Bridge s2 protocols=OpenFlow13
sudo ovs-vsctl set Bridge s3 protocols=OpenFlow13
sudo ovs-vsctl set Bridge s4 protocols=OpenFlow13
```

```
ryu-manager --verbose sample_code.py
```

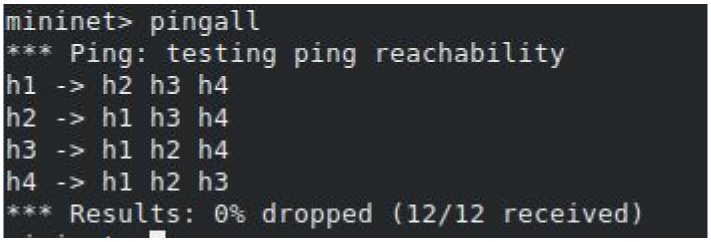
got
OSError: [Errno 98] Address already in use
according to Internet it

> if get error:
> `OSError: [Errno 98] Address already in use`
> then:
> `sudo lsof -i :6653`
> `sudo kill that_pid`

wait on

> move onto main mode
> move onto main mode
> move onto main mode

for a while



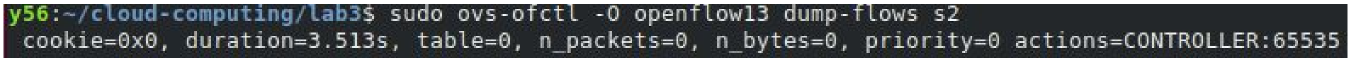
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

(b) How do you generate different traffic? Which tools do you use to generate: ICMP, TCP, UDP and HTTP traffic?

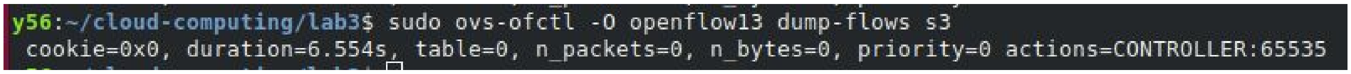
- arp: arping
    - h1 arping h2
    - h1 arp -a // show arp table

- icmp: ping/pingall
  - pingall
  - h1 ping h2
- tcp/udp: nc (netcat) or iperf
  - iperf
    - in xterm of host A: as server: iperf -s (-u) -p
    - in xterm of host B: as client: iperf -c dst_IP (-u) -p
  - netcat
    - nc -l (-u) 22 // to listen as server
    - nc -z -v -u dst_IP port_number // -v for verbose // -z for not containing data // as client
- http: (ie tcp@8080)
  - iperf
    - in xterm of host A: as server: iperf -s (-u) -p
    - in xterm of host B: as server: iperf -c dst_IP (-u) -p

(c) Generate ICMP flows from H4 to H3, and take **screenshots** of the flow table on **S2** and S3 before and after the flow is generated to show that your flow follow the right path. (ovs-ofctl dump-flows)

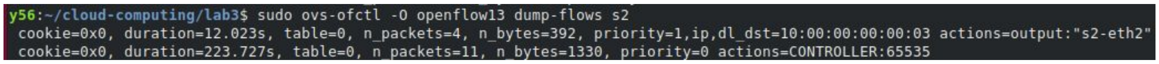


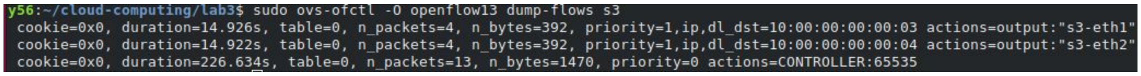
```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s2
 cookie=0x0, duration=3.513s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
```



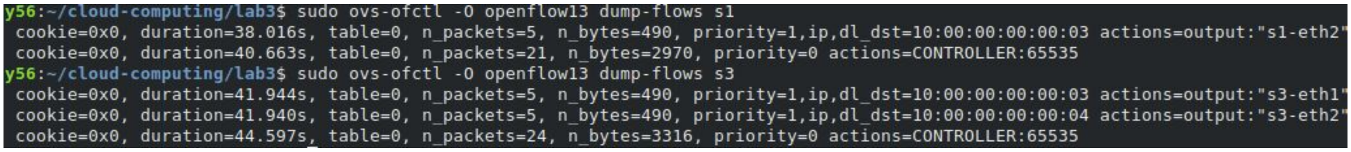
```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s3
 cookie=0x0, duration=6.554s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
```

	Before ICMP flow is generated	After ICMP flow is generated

S2	see above	
		y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s2
		cookie=0x0, duration=12.023s, table=0, n_packets=4, n_bytes=392, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s2-eth2"
		cookie=0x0, duration=223.727s, table=0, n_packets=11, n_bytes=1330, priority=0 actions=CONTROLLER:65535

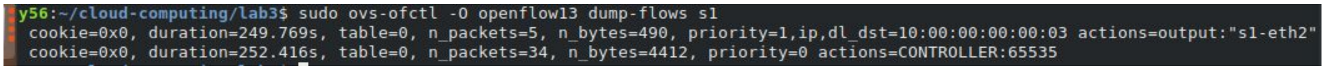
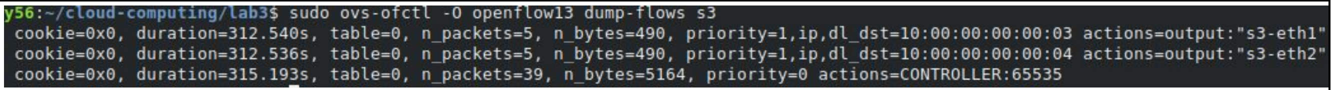
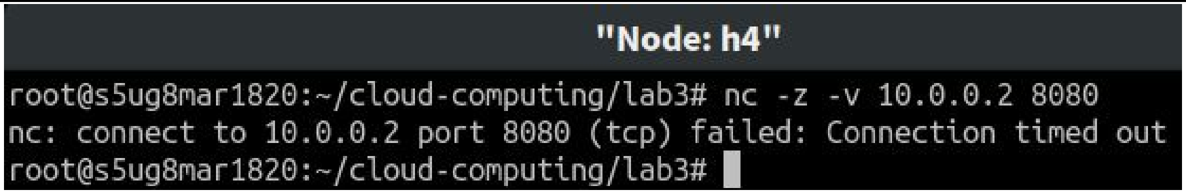
S3	see above	
		y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s3
		cookie=0x0, duration=14.926s, table=0, n_packets=4, n_bytes=392, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s3-eth1"
		cookie=0x0, duration=14.922s, table=0, n_packets=4, n_bytes=392, priority=1,ip,dl_dst=10:00:00:00:00:04 actions=output:"s3-eth2"
		cookie=0x0, duration=226.634s, table=0, n_packets=13, n_bytes=1470, priority=0 actions=CONTROLLER:65535

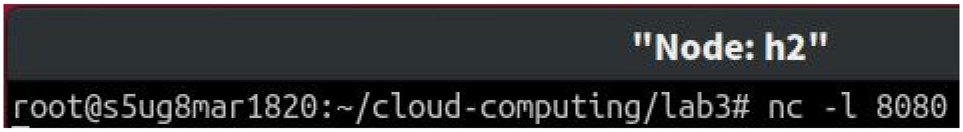
(d) Generate TCP flows (dst_port: 8080) from H4 to H2, and take **screenshots** of the flow table on S1 and S3 before and after the flow is generated. (ovs-ofctl dump-flows) Also, the screenshot of your Mininet or host that generates/receives the TCP traffic.



```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s1
 cookie=0x0, duration=38.016s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s1-eth2"
 cookie=0x0, duration=40.663s, table=0, n_packets=21, n_bytes=2970, priority=0 actions=CONTROLLER:65535
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s3
 cookie=0x0, duration=41.944s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s3-eth1"
 cookie=0x0, duration=41.940s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:04 actions=output:"s3-eth2"
 cookie=0x0, duration=44.597s, table=0, n_packets=24, n_bytes=3316, priority=0 actions=CONTROLLER:65535
```

	Before TCP flow is generated	After TCP flow is generated

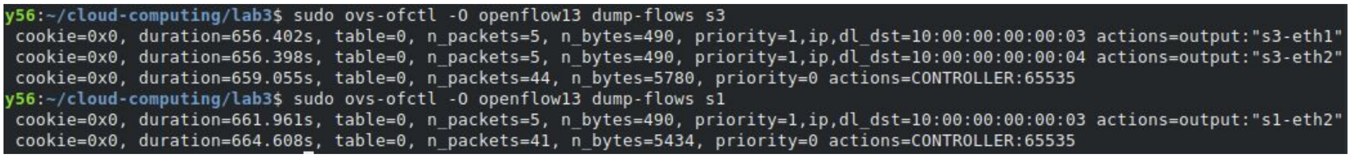
S1	see above	 y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s1 cookie=0x0, duration=249.769s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s1-eth2" cookie=0x0, duration=252.416s, table=0, n_packets=34, n_bytes=4412, priority=0 actions=CONTROLLER:65535
S3	see above	 y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s3 cookie=0x0, duration=312.540s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s3-eth1" cookie=0x0, duration=312.536s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:04 actions=output:"s3-eth2" cookie=0x0, duration=315.193s, table=0, n_packets=39, n_bytes=5164, priority=0 actions=CONTROLLER:65535
	Generates TCP traffic	Receives TCP traffic
Mininet or hosts	see below	 "Node: h4" root@s5ug8mar1820:~/cloud-computing/lab3# nc -z -v 10.0.0.2 8080 nc: connect to 10.0.0.2 port 8080 (tcp) failed: Connection timed out root@s5ug8mar1820:~/cloud-computing/lab3#



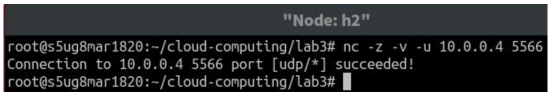
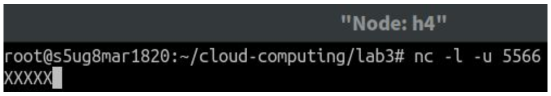
"Node: h2"

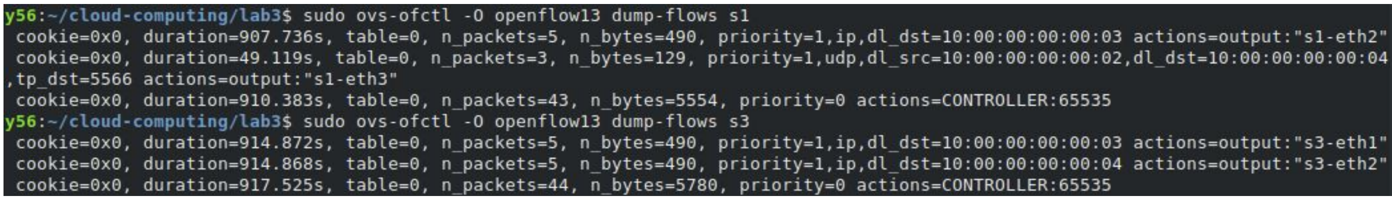
```
root@s5ug8mar1820:~/cloud-computing/lab3# nc -l 8080
```

(e) Generate UDP flows from H2 to H4, and take **screenshots** of the flow table on S1 and S3 before and after the flow is generated. (ovs-ofctl dump-flows) Also, the screenshot of your Mininet or host that generates/receives the UDP traffic.



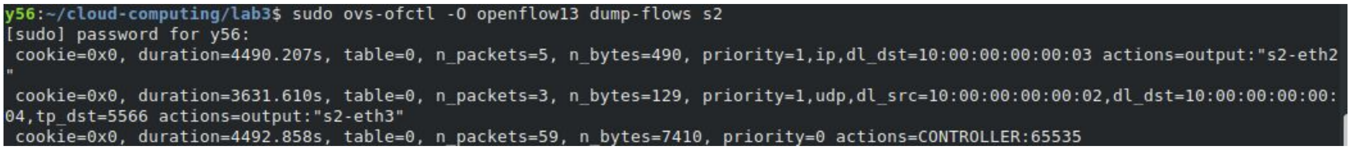
```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s3
 cookie=0x0, duration=656.402s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s3-eth1"
 cookie=0x0, duration=656.398s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:04 actions=output:"s3-eth2"
 cookie=0x0, duration=659.055s, table=0, n_packets=44, n_bytes=5780, priority=0 actions=CONTROLLER:65535
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s1
 cookie=0x0, duration=661.961s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s1-eth2"
 cookie=0x0, duration=664.608s, table=0, n_packets=41, n_bytes=5434, priority=0 actions=CONTROLLER:65535
```

	Before UDP flow is generated	After UDP flow is generated
S1	see above	see below
S3	see above	see below
	Generates UDP traffic	Receives UDP traffic
Mininet or hosts	 "Node: h2" root@s5ug8mar1820:~/cloud-computing/lab3# nc -z -v -u 10.0.0.4 5566 Connection to 10.0.0.4 5566 port [udp/*] succeeded! root@s5ug8mar1820:~/cloud-computing/lab3#	 "Node: h4" root@s5ug8mar1820:~/cloud-computing/lab3# nc -l -u 5566 XXXXX



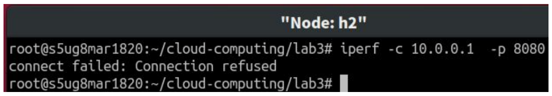
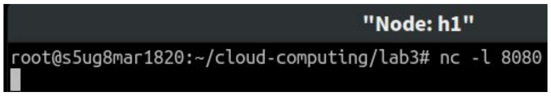
```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s1
 cookie=0x0, duration=907.736s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s1-eth2"
 cookie=0x0, duration=49.119s, table=0, n_packets=3, n_bytes=129, priority=1,udp,dl_src=10:00:00:00:00:02,dl_dst=10:00:00:00:00:04
,tp_dst=5566 actions=output:"s1-eth3"
 cookie=0x0, duration=910.383s, table=0, n_packets=43, n_bytes=5554, priority=0 actions=CONTROLLER:65535
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s3
 cookie=0x0, duration=914.872s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s3-eth1"
 cookie=0x0, duration=914.868s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:04 actions=output:"s3-eth2"
 cookie=0x0, duration=917.525s, table=0, n_packets=44, n_bytes=5780, priority=0 actions=CONTROLLER:65535
```

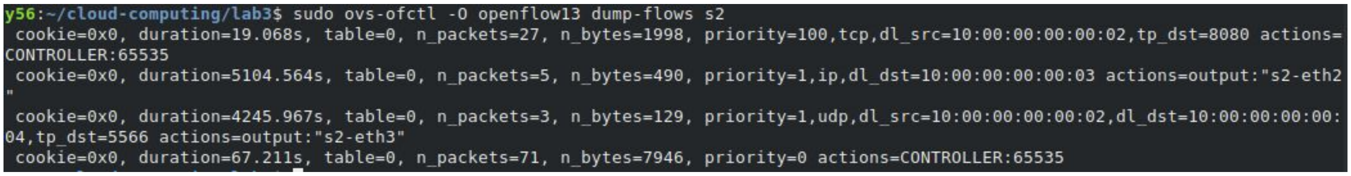
(f) Generate HTTP traffic from H2 to H1, and take **screenshots** of the flow table on S2 before and after the flow is generated. (ovs-ofctl dump-flows) Also, the screenshot of your Mininet or host that generates/receives the HTTP traffic.



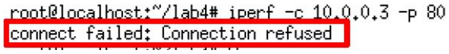
```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s2
[sudo] password for y56:
 cookie=0x0, duration=4490.207s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s2-eth2
"
 cookie=0x0, duration=3631.610s, table=0, n_packets=3, n_bytes=129, priority=1,udp,dl_src=10:00:00:00:00:02,dl_dst=10:00:00:00:00:
04,tp_dst=5566 actions=output:"s2-eth3"
 cookie=0x0, duration=4492.858s, table=0, n_packets=59, n_bytes=7410, priority=0 actions=CONTROLLER:65535
```

	Before HTTP flow is generated	After HTTP flow is generated
S2	see above	see below
	Generates HTTP traffic	Receives HTTP traffic

Mininet or hosts	
	"Node: h2"
	`root@s5ug8mar1820:~/cloud-computing/lab3# iperf -c 10.0.0.1  -p 8080`
	`connect failed: Connection refused`
	`root@s5ug8mar1820:~/cloud-computing/lab3#`
		
		"Node: h1"
		`root@s5ug8mar1820:~/cloud-computing/lab3# nc -l 8080`

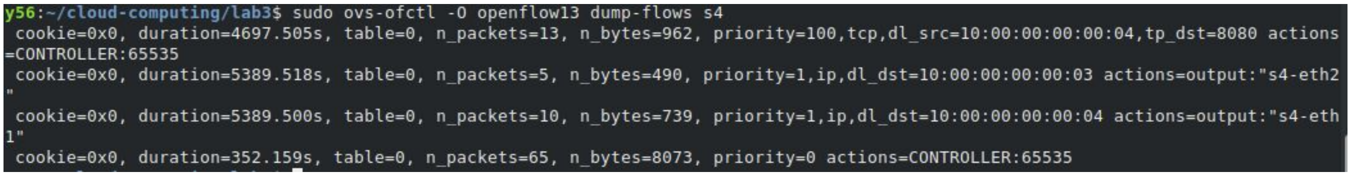


```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s2
 cookie=0x0, duration=19.068s, table=0, n_packets=27, n_bytes=1998, priority=100,tcp,dl_src=10:00:00:00:00:02,tp_dst=8080 actions=
CONTROLLER:65535
 cookie=0x0, duration=5104.564s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s2-eth2
"
 cookie=0x0, duration=4245.967s, table=0, n_packets=3, n_bytes=129, priority=1,udp,dl_src=10:00:00:00:00:02,dl_dst=10:00:00:00:00:
04,tp_dst=5566 actions=output:"s2-eth3"
 cookie=0x0, duration=67.211s, table=0, n_packets=71, n_bytes=7946, priority=0 actions=CONTROLLER:65535
```

Note: “**Connection refused**” means the RST packets is successfully sent back to S2. Otherwise, you need to check if your RST packets is correct. e.g., 

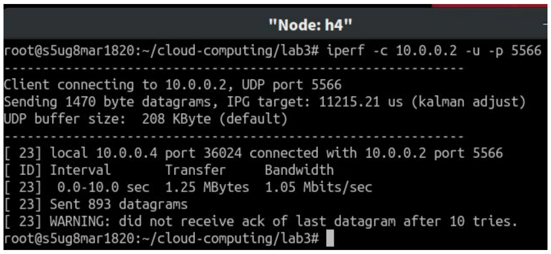
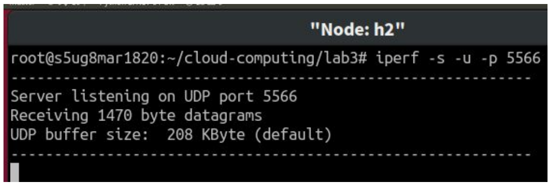
```
root@localhost:~/lab4# iperf -c 10.0.0.3 -p 80
connect failed: Connection refused
```

(g) Generate UDP traffic from H4 to H2, and take **screenshots** of the flow table on S4 before and after the flow is generated. (ovs-ofctl dump-flows) Also, the screenshot of your Mininet or host that generates/receives the UDP traffic.



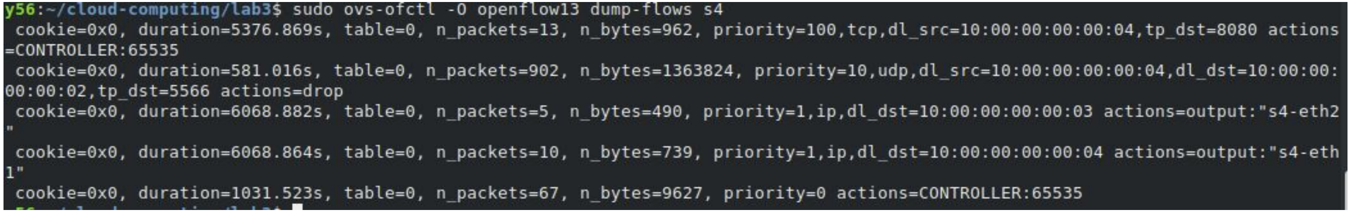
```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s4
 cookie=0x0, duration=4697.505s, table=0, n_packets=13, n_bytes=962, priority=100,tcp,dl_src=10:00:00:00:00:04,tp_dst=8080 actions
=CONTROLLER:65535
 cookie=0x0, duration=5389.518s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s4-eth2
"
 cookie=0x0, duration=5389.500s, table=0, n_packets=10, n_bytes=739, priority=1,ip,dl_dst=10:00:00:00:00:04 actions=output:"s4-eth
1"
 cookie=0x0, duration=352.159s, table=0, n_packets=65, n_bytes=8073, priority=0 actions=CONTROLLER:65535
```

	Before UDP flow is generated	After UDP flow is generated
S4	see above	see below
	Generates UDP traffic	Receives UDP traffic

Mininet or hosts		

```
"Node: h4"
root@s5ug8mar1820:~/cloud-computing/lab3# iperf -c 10.0.0.2 -u -p 5566
------------------------------------------------------------
Client connecting to 10.0.0.2, UDP port 5566
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 23] local 10.0.0.4 port 36024 connected with 10.0.0.2 port 5566
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 23] Sent 893 datagrams
[ 23] WARNING: did not receive ack of last datagram after 10 tries.
root@s5ug8mar1820:~/cloud-computing/lab3# 
```

```
"Node: h2"
root@s5ug8mar1820:~/cloud-computing/lab3# iperf -s -u -p 5566
------------------------------------------------------------
Server listening on UDP port 5566
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```



```
y56:~/cloud-computing/lab3$ sudo ovs-ofctl -O openflow13 dump-flows s4
 cookie=0x0, duration=5376.869s, table=0, n_packets=13, n_bytes=962, priority=100,tcp,dl_src=10:00:00:00:00:04,tp_dst=8080 actions
=CONTROLLER:65535
 cookie=0x0, duration=581.016s, table=0, n_packets=902, n_bytes=1363824, priority=10,udp,dl_src=10:00:00:00:00:04,dl_dst=10:00:00:
00:00:02,tp_dst=5566 actions=drop
 cookie=0x0, duration=6068.882s, table=0, n_packets=5, n_bytes=490, priority=1,ip,dl_dst=10:00:00:00:00:03 actions=output:"s4-eth2
"
 cookie=0x0, duration=6068.864s, table=0, n_packets=10, n_bytes=739, priority=1,ip,dl_dst=10:00:00:00:00:04 actions=output:"s4-eth
1"
 cookie=0x0, duration=1031.523s, table=0, n_packets=67, n_bytes=9627, priority=0 actions=CONTROLLER:65535
```

(h) Please find what is “Spanning Tree” and “Spanning Tree Protocol”? What’s the purpose of the protocol?

- “Spanning Tree”
  - A spanning tree T of an undirected graph G is a subgraph that is a tree which includes all of the vertices of G, with a minimum possible number of edges.
- “Spanning Tree Protocol”
  - To avoid loops which will echo too much when flood((broadcast radiation)). STP is a network protocol that builds a loop-free logical topology for Ethernet networks (L2 network) .
- What’s the purpose of the protocol?
  - To avoid loops which will echo too much when flood(broadcast radiation).

(i) Is it necessary to implement spanning tree in SDN for packet forwarding? Why?

NO, control plane should make sure that the flow tables in the data plane will not form loops. In SDN, switches obey control plane, having no needs to find spanning trees by themselves.

(j) If you want to find spanning tree in SDN, how will you implement and what is the difference between traditional “Spanning Tree Protocol” and the one in SDN?

I will let the controller collect traffic and calculate MST for us (using Kruskal's Algorithm). Then let the controller input the flow tables to all switches.

Using SDN can ease the burden of switches and use the spanning tree w/ minimum path cost

https://osrg.github.io/ryu-book/en/html/spanning_tree.html

(k) List three advantages of using OpenVSwitch and SDN controller compared to IP networks. Briefly explain why

1. easy control traffic if we have privilege.
2. can use different priority to send data (can be according to service type or fee)
3. can do more service (firewall, security, etc) w/ white boxes
4. easy to version up and configure switches remotely

(l) Include the controller's code.

(Upload with your report or attach a sharable link)
https://github.com/y56/cloud-computing/blob/master/lab3/controller-lab3.py

(m) Include the topology file

(Upload with your report or attach a sharable link)
https://github.com/y56/cloud-computing/blob/master/lab3/topology.py

(n) Challenges you've encountered while doing this experiment, and explain how you manage to solve them. If you do not experience any problem, simply say no problem.

so difficult
little explanation of ryu
little clue

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