Lab3 Report: SDN Open Virtual Switches

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

Name: Ziyan Qiu ID: zq2048 Date: 11.7.2021

1. Objectives

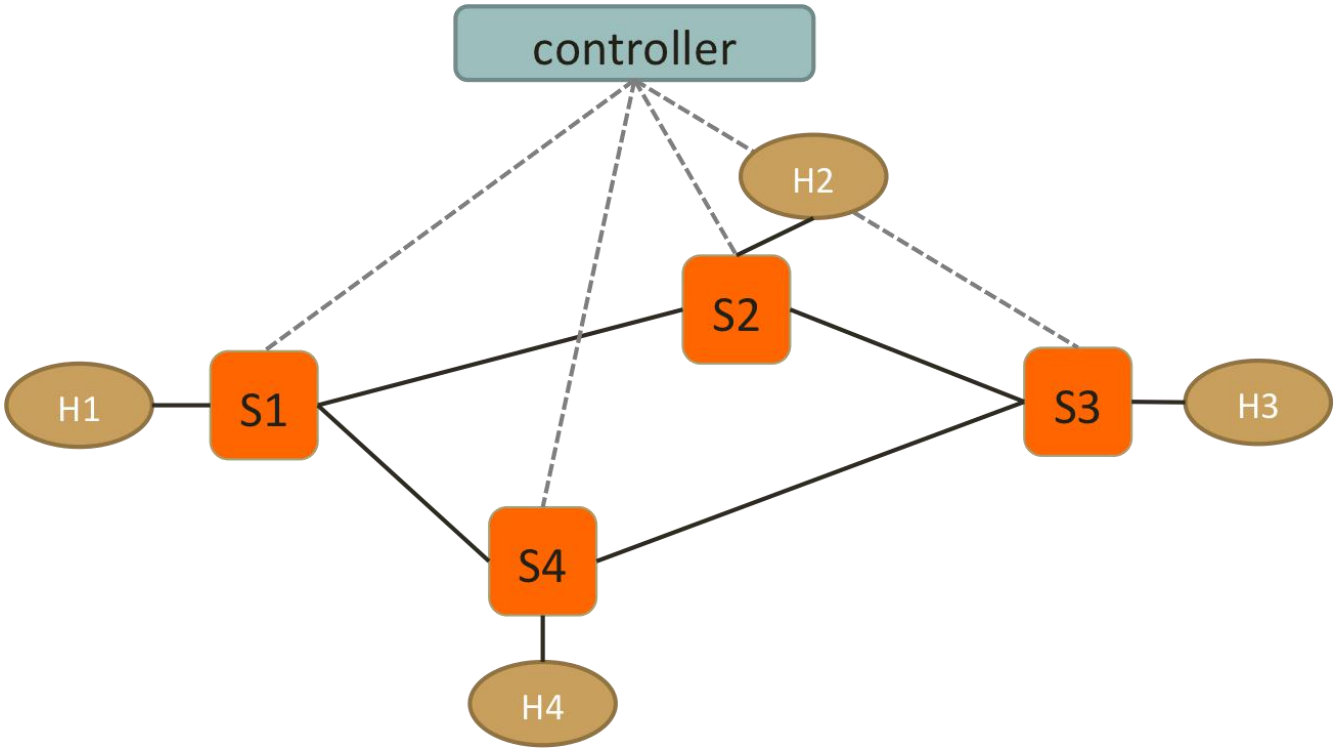
- Understand SDN and get familiar with controllers.

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
            - e.g. S1-S2-S3, S2-S3-S4
        - UDP packets take the counterclockwise path
            - e.g. S1-S4-S3, S2-S1-S4
        - H2 and H4 cannot send HTTP traffic (TCP with dst_port:80)
            - New connections are dropped with a TCP RST sent back to **H2 or H4**
            - 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.
        - H1 and H4 cannot send UDP traffic
            - simply drop packets at switches

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**.

Before:

```
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)
```

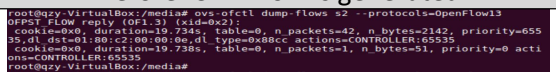
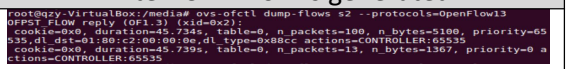
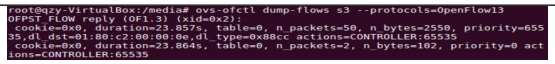
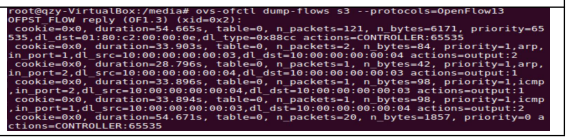
After:

```
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)
```

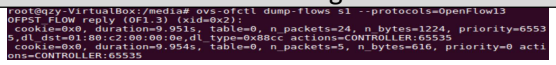
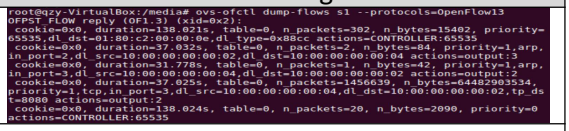
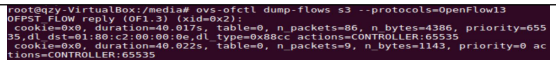
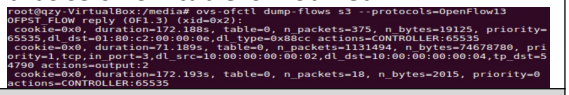
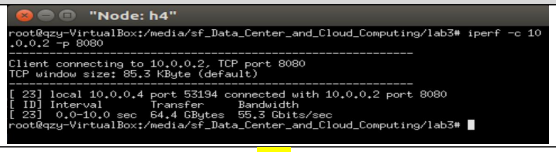
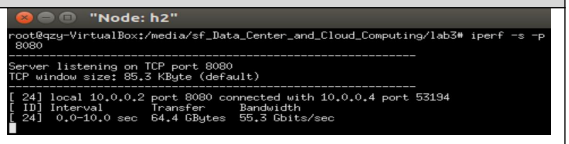
(b) How do you generate different traffic? Which tools do you use to generate: ICMP, TCP, UDP and HTTP traffic?

ICMP: ping
TCP: iperf
UDP: iperf
HTTP: iperf

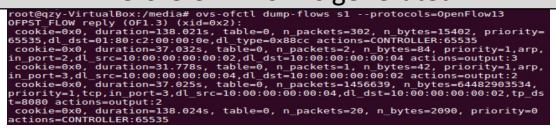
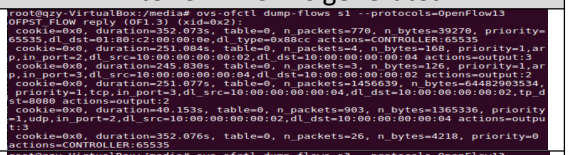
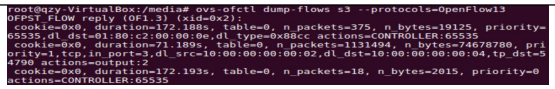
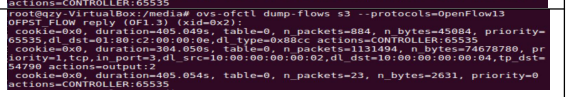
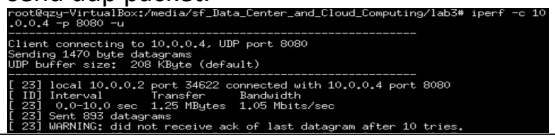
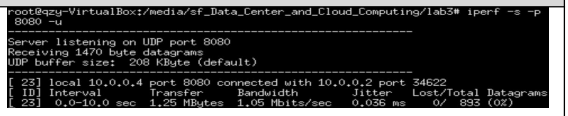
(c) Generate ICMP flows from **H4 to H3**, and take **screenshots** of the flow table on **S2** and S3 before and after the flow is generated to show that your flow follow the right path. (ovs-ofctl dump-flows)

	Before ICMP flow is generated	After ICMP flow is generated
S2		
S3		

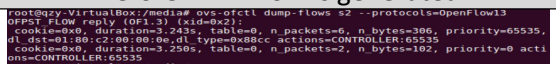
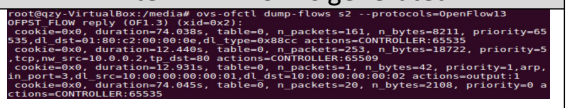
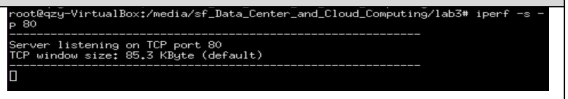
(d) Generate TCP flows (dst_port: 8080) from **H4 to H2**, and take **screenshots** of the flow table on S1 and S3 before and after the flow is generated. (ovs-ofctl dump-flows) Also, the screenshot of your Mininet or host that generates/receives the TCP traffic.

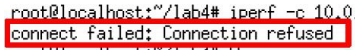
	Before TCP flow is generated	After TCP flow is generated
S1		
S3		TCP packet will be sent back to h4 from h2, thus s3's flow table is modified 
	Generates TCP traffic	Receives TCP traffic
Mininet or hosts		

(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.

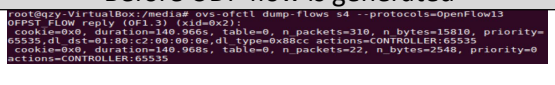
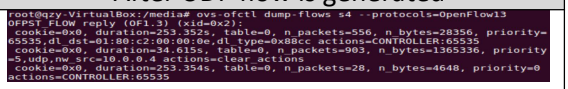
	Before UDP flow is generated	After UDP flow is generated
S1		
S3		
	Generates UDP traffic	Receives UDP traffic
Mininet or hosts	iperf in receiver will send upd packet back to the sender to ack, but h4 is not allowed to send udp packet. 	

(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.

	Before HTTP flow is generated	After HTTP flow is generated
S2		
	Generates HTTP traffic	Receives HTTP traffic
Mininet or hosts		

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., 

(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.

	Before UDP flow is generated	After UDP flow is generated
S4		
	Generates UDP traffic	Receives UDP traffic

Mininet or hosts	

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

Spanning tree is a connected acyclic undirected graph, and is the sub-graph of the the original graph that contains all the vertex.
Spanning tree protocol helps to find the spanning tree of a ethernet network and turn down all other links not within the spanning tree, to avoid broadcast storm and provide failure recovery.
The purpose of this protocol is to create a loop free ethernet network, so as to avoid broadcast storm triggered by MAC address look up.

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

It is unnecessary to implement STP in SDN, because we can have control plane to instruct the switch how to forward the packets manually, but STP is still useful in SDN.

(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?

The STP in SDN network will be implemented by the controller. The controller can first probe the layout of all the switches in the network, and then it can determine the spanning tree and instruct the switches to forward the broadcast packets only within this spanning tree. However, links not within the spanning tree can still be used to transmit packets that are not broadcast, unlike the traditional STP, where those links are invalidated.

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

1. SDN allows fine-grain central control over the network functionality.
2. SDN allows network setting automation.
3. SDN’s central controlling makes the network more efficient and guarantees content delivery.

(l) Include the controller’s code.

(Upload with your report or attach a sharable link)
Upload together.

(m) Include the topology file

(Upload with your report or attach a sharable link)
Upload together.

(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.

1. ARP proxy and avoid broadcast storm
2. Find shortest path using networkx, with --observe-links, and when there are multiple paths, choose the one that meet the need.
3. Set up the flow table to drop specific UDP packet simply and send specific TCP packet to controller to

generate a TCP_RST.
4. Match packets with ARP, ICMP, UDP, TCP and HTTP specifically.
5. Much higher work load than previous lab.
6. Caught a cold during these days.

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