CS339 lab 6

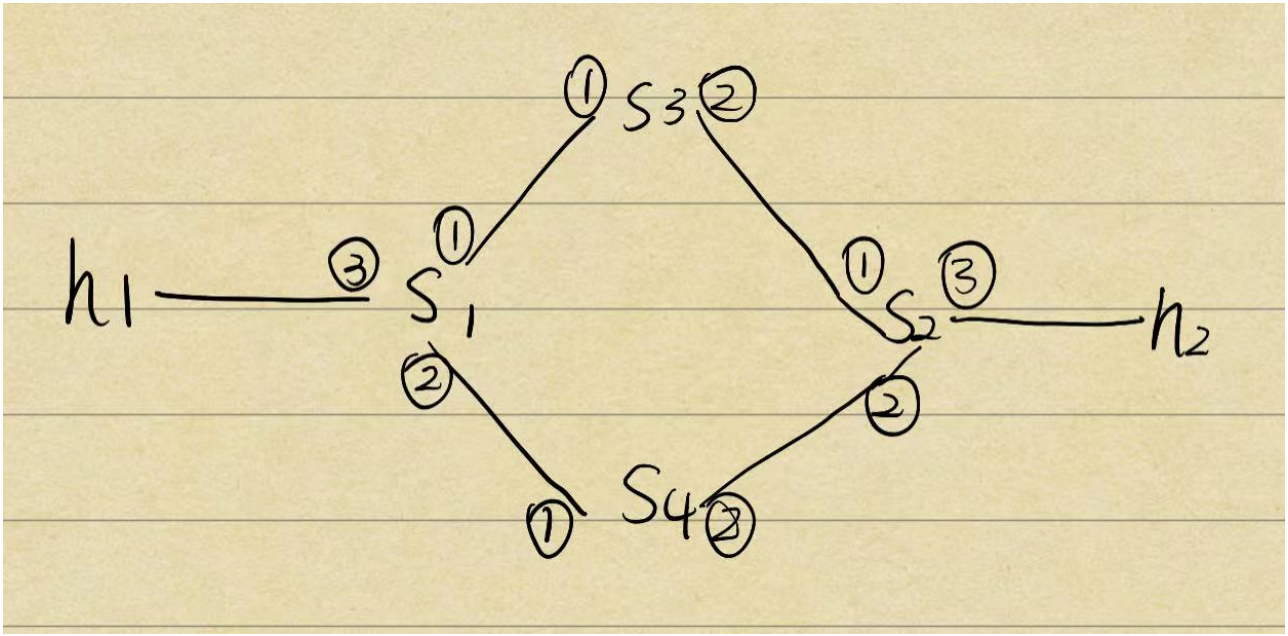
Introduction

In this lab, we are required to use RYU to implement an open flow controller. With RYU, we can regulate the flow rules of each switch. Therefore, we can avoid ARP storm in this situation.

HM1

In homework 1, it requires us to create a topology with a loop in the middle. It's simple. For the convenience of adding new flow rules, I specialized the connecting ports between each pair of switches. Since that I am able to add flow rules more easily.

Besides, I specified the mac addresses of h1 and h2 for later use.



HM2

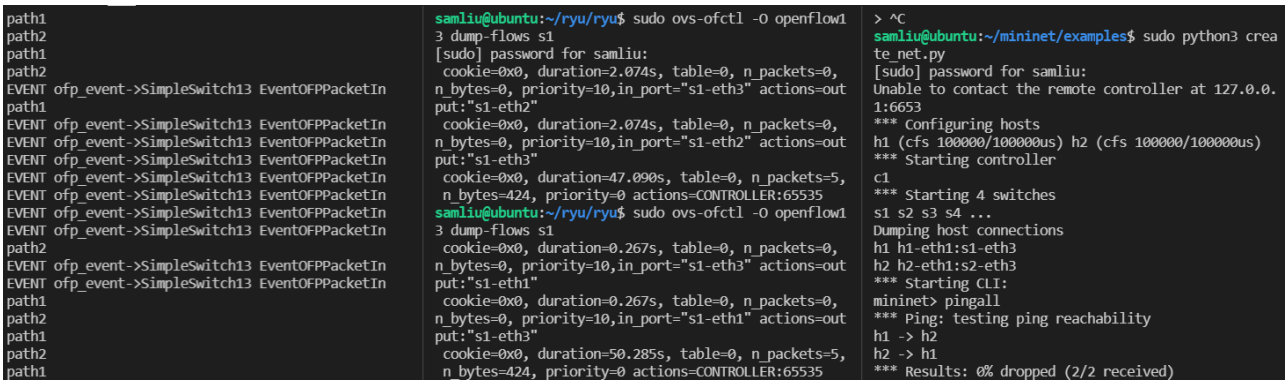
In homework 2, it requires us to switch paths(h1-s1-s3-s2-h2 or h1-s1-s4-s2-h2) between h1 and h2 every 5 seconds. Reading the code named "simple_switch_13.py", I have known that function "switch_features_handler" is used for setting the features of switches. Therefore, I considered to modify this function and add the flow rules of each switch in this function.

Further understanding the function, I have discovered that this function will be called whenever there is a new switch to be created. Therefore, this function will be called four times under our topology. However, I wish to change the path every 5 seconds. So the paths should be changed more than 4 times.

To solve this problem, I add a new variable called "self.switch={}" when initializing the class. Whenever a switch is created and the function is called, I will add the switch into this dictionary. When the length of dictionary comes to 4, all the switches have been initialized. In this case, I use a while loop in the function and change the paths every 5 seconds. Noticing that when switching the paths, I only need to modify switch 1 and switch 2.

Besides, I need to delete the path that is not used. Simply changing the command in parser.OFPFlowMod to ofproto.OFPFC_DELETE can reach the goal.

Here is the running results.



HM3

In homework 3, it requires us to use both paths to forward packets from h1 to h2. That's much easier. I only need to initialize the four switches at first.

To make the workload more balanced, I group the two paths together and give each path 50% of capacity.

```
def send_group_mod(self,datapath):
    ofproto=datapath.ofproto
    parser=datapath.ofproto_parser
    weight1=50
    weight2=50
    watch_port=ofproto_v1_3.OFPP_ANY
    watch_group=ofproto_v1_3.OFPQ_ALL
    actions1 = [parser.OFPActionOutput(1)]
    actions2=[parser.OFPActionOutput(2)]
    buckets = [parser.OFPBucket(weight1, watch_port, watch_group, actions=actions1),
               parser.OFPBucket(weight2, watch_port, watch_group, actions=actions2)]

    req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                             ofproto.OFPGT_SELECT, 50, buckets)
    datapath.send_msg(req)
```

HM4

In homework 4 , it requires us to use the first path (h1-s1-s3-s2-h2) to forward packets from h1 to h2 and the second path for backup. When the first path experiences a link failure, the network should automatically switch to the second path. The hint asks us to consider using OFPGT_FF to construct a group table.

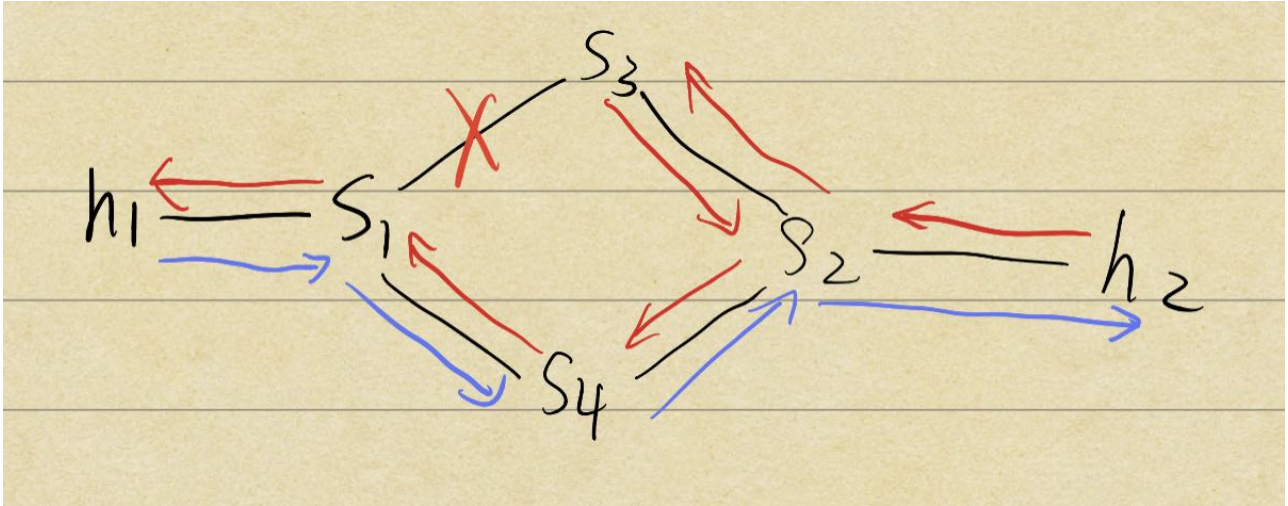
To use OFPGT_FF, we need to know the usage of it. Reading the document of ofproto_v1_3, I have known that this signal means choosing the first living bucket in the group to add flow. Based on our topology, I create a group for s1 and s2 that they will set the out_port as 1 with higher priority and set the out_port as 2 for backup using.

```
actions1 = [parser.OFPActionOutput(1)]
actions2 = [parser.OFPActionOutput(2)]
buckets = [parser.OFPBucket(watch_port=1, actions=actions1),
           parser.OFPBucket(watch_port=2, actions=actions2)]

req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                         ofproto.OFPGT_FF, 50, buckets=buckets)
datapath.send_msg(req)
```

When I make the link between s1 and s3 down and make the link between s2 and s3 down, the network can ping successfully. However, when I make the link between s1 and s3 down, the network cannot ping successfully.

To solve this problem, I wish to create a flow table as below.



In this case, the packets sent from h2 to h1 will not lose. Therefore, I need to set two groups for s3. When in_port is 1, it set 2 as out_port and itself as out_port for backup. When in_port is 2, it set 1 as out_port and itself as out_port for backup. Notice that when packets are sent out from the port they were sent in, the output port need to be set to OFPP_IN_PORT.

```
OFPP_IN_PORT=0xfffffff8
actions1 = [parser.OFPActionOutput(1)]
actions2 = [parser.OFPActionOutput(OFPP_IN_PORT)]
buckets = [parser.OFPBucket(watch_port=1, actions=actions1),
           parser.OFPBucket(watch_port=2, actions=actions2)]

req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                         ofproto.OFPGT_FF, 52, buckets=buckets)
datapath.send_msg(req)

actions1 = [parser.OFPActionOutput(2)]
actions2 = [parser.OFPActionOutput(OFPP_IN_PORT)]
buckets = [parser.OFPBucket(watch_port=2, actions=actions1),
           parser.OFPBucket(watch_port=1, actions=actions2)]
```

Here is the running results.

```
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> link s1 s3 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> link s3 s2 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Conclusion

In this lab, I have a further understanding in SDN controller and RYU. I have learnt how to add and delete flow by using RYU. Besides, I have learnt how to group different actions together to make the flow rules more reasonable. To sum up, it is a fruitful lab experience.