SDNs Report

CS305 final project

11712738 武羿
11712121 胡玉斌
11612003 彭可

1. Background

The understanding for SDN

As we all know, in SDN, or software-defined network, a distinct (typically remote) controller interacts with local control agents in switches/routers to compute forwarding tables. Unlike difficult traditional routing who implements a distributed algorithm in every switch/router, a logically centralized control plane is easier to manage the whole network, avoiding router misconfigurations. In our understanding, if we could “extract” the control plane out of the routers, then it would be easier to add, modify and delete the routing logic, and enable the network to adapt to more complex usage scenarios. If there are some changes need to be made, we could just modify the remote controller since no algorithm is implemented in routers/switches. It’s just like a kind of “design pattern” intending to reduce the coupling degree of the network, and the best solution of which is to separate the "model layer" and "controller layer".

The understanding for Ryu

As far as we are concerned, Ryu controller is an open-source SDN / OpenFlow controller developed by NTT company in Japan. The controller is named "flow" in Japan, so it's called Ryu. Ryu controller is completely written in Python language, similar to POX. Ryu controller now supports OpenFlow version 1.0, 1.2, 1.3 and 1.4, and supports the combination of OpenStack and cloud computing. Ryu adopts the Apache license open-source protocol standard. In this project, we use Ryu with OpenFlow 1.0 to control the whole virtual network.

The understanding for mininet

In our understanding, mininet is a network simulator connected by some virtual terminal nodes, switches, and routers. It uses lightweight virtualization technology to make the system comparable to the real network.

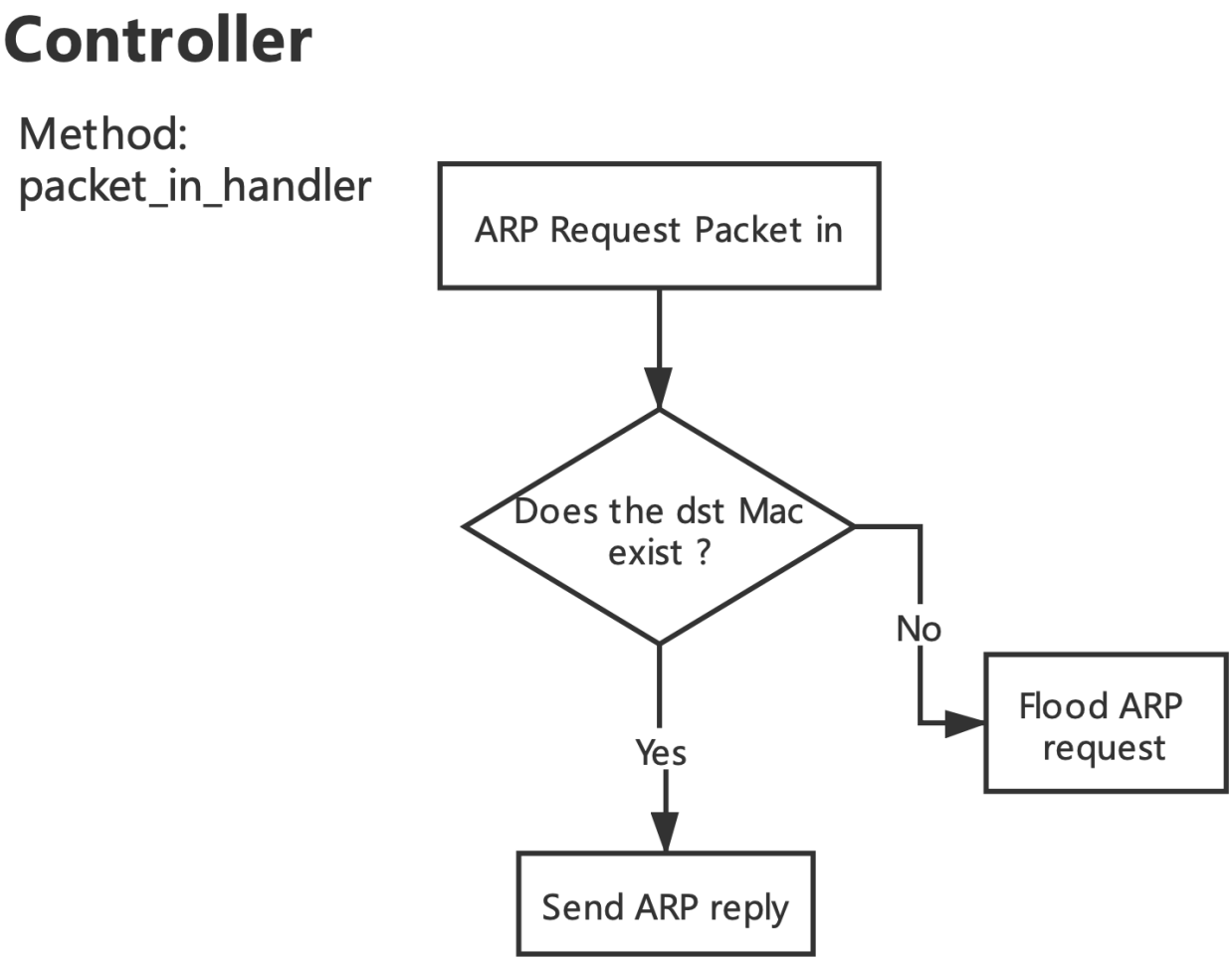
Mininet can easily create a network supporting SDN: host works as a real computer, can log in Using SSH, start the application program, the program can send data packets to the Ethernet port, and the

data packets will be received and processed by the switch and router. With this network, we can flexibly add new functions to the network and conduct relevant tests, and then easily deploy to the real hardware environment. In this project, we used some topologies provided by mininet to simulate the real network environment.

2. Implementation

2.1: Function 1 Controller handle ARP packet

Code Logic and Struture :



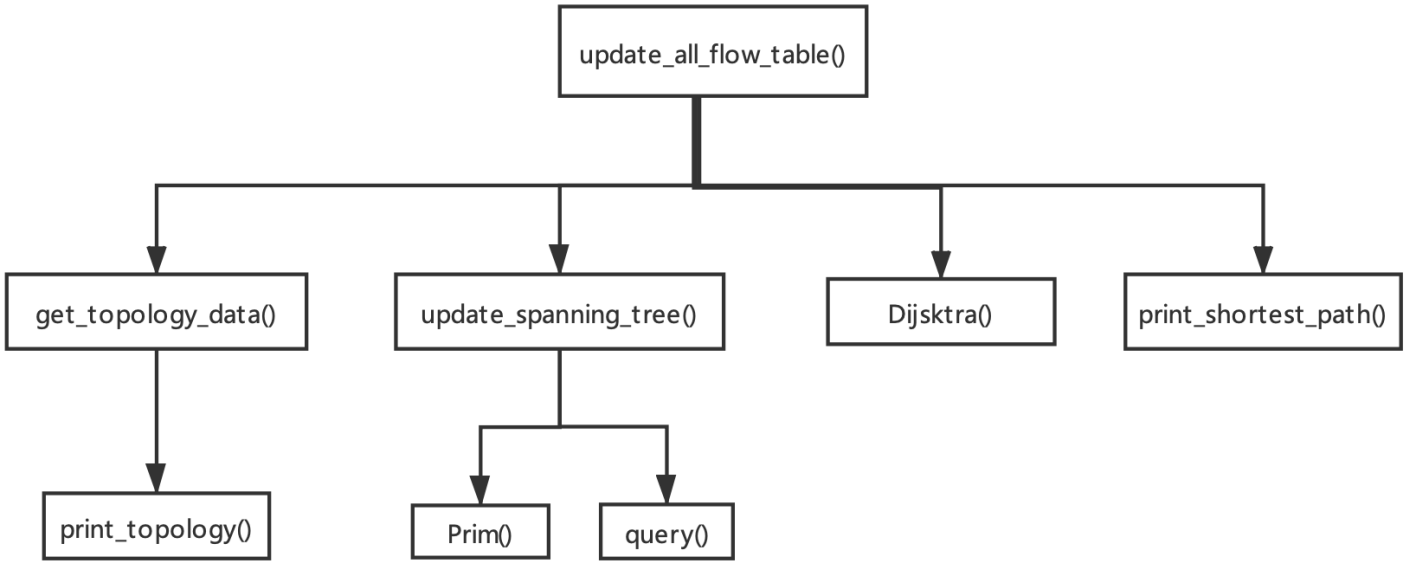
Code detail :

```
# TODO:  Generate a *REPLY* for this request based on your switch state
mac_answer = 0
# search for the mac address
for ip in self.tm.ip_host_mac:
    if ip == arp_msg.dst_ip:
        mac_answer = self.tm.ip_host_mac[ip]
        break
if mac_answer != 0:
    # if the mac address exists, controller send the ARP reply
    ofctl.send_arp(arp_opcode=arp.ARP_REPLY, vlan_id=VLANID_NONE,
                   dst_mac=arp_msg.src_mac,
                   sender_mac=mac_answer, sender_ip=arp_msg.dst_ip,
                   target_ip=arp_msg.src_ip, target_mac=arp_msg.src_mac,
                   src_port=ofctl.dp.ofproto.OFPP_CONTROLLER,
                   output_port=in_port
                   )
else:
    # if the mac address does not exist, the ARP request will flood.
    data = msg.data
    ofproto = dp.ofproto
    ofp_parser = dp.ofproto_parser
    actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
    out_flood = ofp_parser.OFPPacketOut(
        datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,
        actions=actions, data=data)
    dp.send_msg(out_flood)
```

Notice that after we implement the spaninng tree to forward arp packets in swtiches, there is no arp packets delivered to the controller any more.

2.2 Function2: Flooding without loops and Shortest path

code structure



Method Details

update_all_flow_table(): This method is called whenever the network topology changes. Calling other methods to update the flow table in switches. Especially, the update for shortest path in flow table is implemented in this method.

```
# 最短路径，流表更新
for k in s_dic:
    if s_dic[k] == i.dp.id:  # 等于本身意味着，一步就可以到达
        next_port = link_port_dict[s_dic[k]][k]
    else:
        next_port = link_port_dict[i.dp.id][s_dic[k]]
    ## 用目的地的mac地址进行match
    for host_mac in self.switch_host_mac[k]:
        ofc.set_flow(dl_dst=host_mac, cookie=0, priority=0,
                     actions=[ofp_parser.OFPActionOutput(next_port)])
    # 交换机直接连的主机也要明确端口
    for host_mac in self.switch_host_mac[i.dp.id]:
        port = self.mac_host_port[host_mac]
        ofc.set_flow(dl_dst=host_mac, cookie=0, priority=0,
                     actions=[ofp_parser.OFPActionOutput(port)])
```

get_topology_data(): get the topology infomation of the current network, including how switches are connected and through which ports.

Shortest Path:

Dijsktra(): Given a source node S, caculate the shortest path form S to all the other nodes in the current network topology graph.

```
def Dijkstra(self,n: int, S: int, para_edges: list) -> (dict, list):

    Graph = [[] for i in range(n + 1)]
    for edge in para_edges:
        u, v = edge
        Graph[u].append(node(v, 1))

    dis = [INF for i in range(n + 1)]
    dis[S] = 0

    via = {}
    pre = [0 for i in range(n + 1)]
    paths = [[] for i in range(n + 1)]
    paths[S].append(S)

    pq = PriorityQueue()
    pq.put(node(S, 0))
    while not pq.empty():
        top = pq.get()
        if dis[top.id] < top.w:
            continue
        if top.id != S:
            paths[top.id] = paths[pre[top.id]].copy()
            paths[top.id].append(top.id)
        for i in Graph[top.id]:
            if dis[i.id] > dis[top.id] + i.w:
                dis[i.id] = dis[top.id] + i.w
                pre[i.id] = top.id
                if top.id != S:
                    via[i.id] = via[top.id]
                else:
                    via[i.id] = i.id
                pq.put(node(i.id, dis[i.id]))
    return via, paths
```

print_shortest_path() : Print shortest paths for every switch.

Flooding without loops:

update_spanning_tree() :To implement flooding without loops. Update the switches flow table to specify the direction in which Arp packects are forwarded, based on a spanning tree structure of the current network topology.

```
# 指定当前交换机要output到其他switch的所有port，添加到action_set中
for each_child in relationship[father.dp.id]:
    port = link_port_dict[father.dp.id][each_child]
    action_set.append(ofp_parser.OFPActionOutput(port))
    print(" Switch_{}/Port_{} -> Switch_{}".format(father.dp.id, port, each_child))

# 指定当前交换机要output到其他host的所有port，添加到action_set中
for host_port in self.switch_host_port[father.dp.id]:
    action_set.append(ofp_parser.OFPActionOutput(host_port))

# 更新流表，ARP包通过广播地址和source address的ip地址来match
for host_ip in self.switch_host_ip[i.dp.id]:
    for each_ip in host_ip:
        ofc.set_flow(nw_src=each_ip, dl_dst="ff:ff:ff:ff:ff:ff", cookie=0, priority=0,
                     dl_type=ether_types.ETH_TYPE_ARP,
                     actions=action_set)
```

Prim(): generate a minimum spanning tree for the current network topology.

```
def Prim(self, n: int, S: int, para_edges: list) -> list:
    '''Return a list that contains the edges in the spanning tree.
    'n' is the total number of nodes, S is an arbitrary start point
    in the graph, para_edges is the list of the graph edges.

    '''
    print("Start Spanning Tree Algorithm...")

    Graph = [[] for i in range(n + 1)]
    for edge in para_edges:
        u, v = edge
        Graph[u].append(node(v, 1))

    dis = [INF for i in range(n + 1)]
    dis[S] = 0

    pre = [-1] * (n + 1)

    pq = PriorityQueue()
    pq.put(node(S, 0))
    while not pq.empty():
        top = pq.get()
        for i in Graph[top.id]:
            if dis[i.id] > i.w:
                dis[i.id] = i.w
                pre[i.id] = top.id
                pq.put(node(i.id, dis[i.id]))

    tree_edges = []
    for i in range(n + 1):
        if pre[i] != -1:
            tree_edges.append((i, pre[i]))
            tree_edges.append((pre[i], i))

    return tree_edges
```

query(): Given a root node, run BFS on the spanning tree, return a dictionary that contains the children sets of every node .

2.3 Obtain Network topology

Evey time when the topology of the network is changed, we get the topology data of the current network by used function of app_manager.RyuApp.

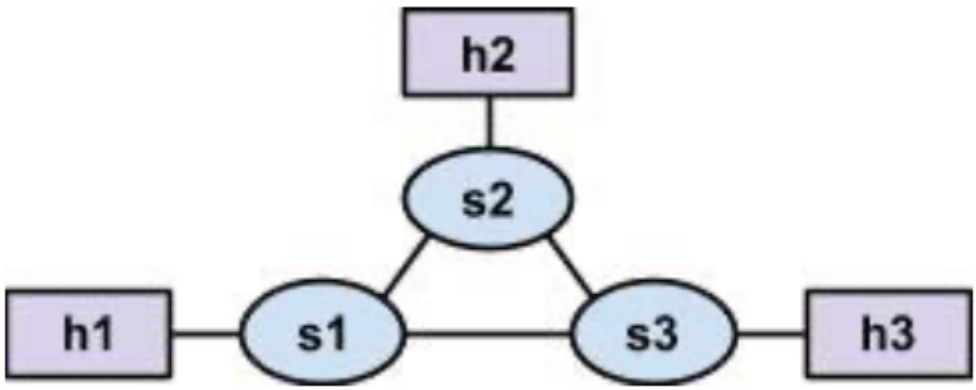
```
def get_topology_data(self):
    switch_list = topo.get_switch(self.topology_api_app, None)
    switches = [switch.dp.id for switch in switch_list]
    links_list = topo.get_link(self.topology_api_app, None)
    links = [(link.src.dpid, link.dst.dpid) for link in links_list]
    link_port_dict = defaultdict(dict)

    for link in links_list:
        link_port_dict[link.src.dpid][link.dst.dpid] = link.src.port_no

    self.print_topology(link_port_dict, switches)
    return links, link_port_dict, switches, switch_list
```

3. Test

3.1 Mininet triangle



1. **Test command 'pingall'**

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

2. **ping an unexist ip address 10.0.0.100**

```
mininet> h1 ping 10.0.0.100 -c 1
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.100 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Use TCPDUMP to capture arp packet

No flood ,only 3 ARP request packtes captured.

```
vagrant@vagrant:~$ sudo tcpdump -i s1-eth2 arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
14:39:25.194976 ARP, Request who-has 10.0.0.100 tell 10.0.0.1, length 28
14:39:26.223486 ARP, Request who-has 10.0.0.100 tell 10.0.0.1, length 28
14:39:27.247392 ARP, Request who-has 10.0.0.100 tell 10.0.0.1, length 28
```

3. **Print Shortest Path**

```
_____Start Printing Shortest Path_____
_
* For Switch_1 :
> Switch_1 to Switch_1
[1]
> Switch_1 to Switch_2
[1, 2]
* For Switch_3 :
> Switch_3 to Switch_1
[3, 1]
> Switch_3 to Switch_2
[3, 2]
* For Switch_2 :
> Switch_2 to Switch_1
[2, 1]
> Switch_2 to Switch_2
[2]
_____End Printing Shortest Path_____
```

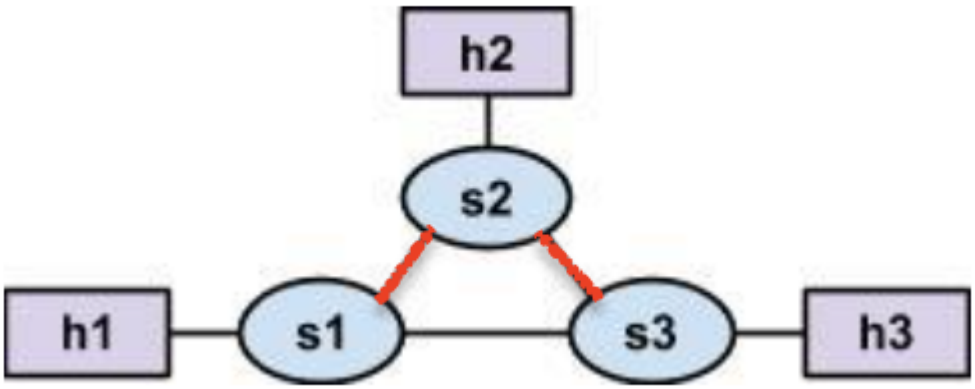
4. **Print topology graph**

```
_____________________Start Printing Topology_____________________
* For Switch_3 --------------------
> Connected Switches :
Edge: switch_3/port_3 <-> switch 1/port_3
Edge: switch_3/port_2 <-> switch 2/port_3
> Connected  Hosts:
Edge: switch_3 <-> host_ip_['10.0.0.3']
* For Switch_2 --------------------
> Connected Switches :
Edge: switch_2/port_2 <-> switch 1/port_2
Edge: switch_2/port_3 <-> switch 3/port_2
> Connected  Hosts:
Edge: switch_2 <-> host_ip_['10.0.0.2']
* For Switch_1 --------------------
> Connected Switches :
Edge: switch_1/port_2 <-> switch 2/port_2
Edge: switch_1/port_3 <-> switch 3/port_3
> Connected  Hosts:
Edge: switch_1 <-> host_ip_['10.0.0.1']
_____________________END Printing Topology_____________________
```

5. **Print Spanning Tree**

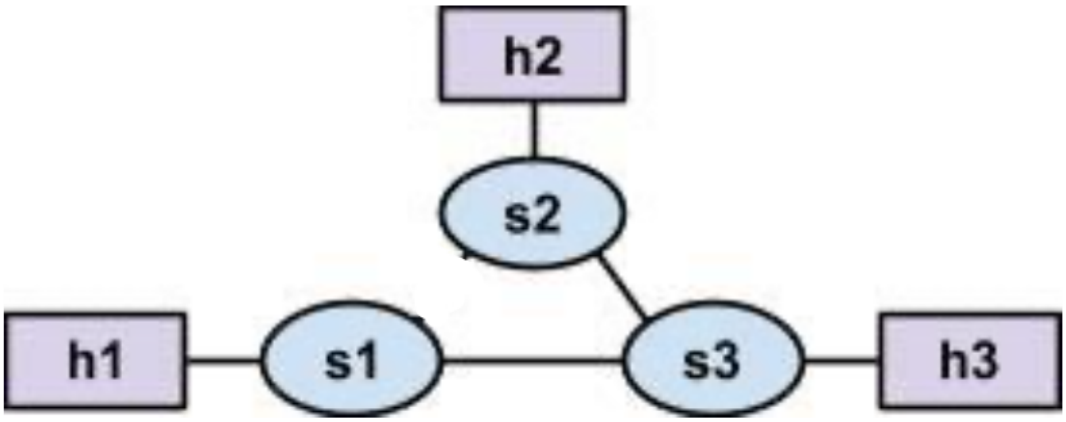
   Notice that for each edge, we print bidirectionally.

```
************ Spanning Tree **************
[(2, 1), (1, 2), (3, 1), (1, 3)]
```



6. **Change topology**

```
mininet> link s1 s2 down
```



Shortest Path after change

```
_________________________Start Printing Shortest Path_____________________________
* For Switch_1 :
> Switch_1 to Switch_1
[1]
> Switch_1 to Switch_2
[1, 3, 2]
> Switch_1 to Switch_3
[1, 3]
* For Switch_3 :
> Switch_3 to Switch_1
[3, 1]
> Switch_3 to Switch_2
[3, 2]
> Switch_3 to Switch_3
[3]
* For Switch_2 :
> Switch_2 to Switch_1
[2, 3, 1]
> Switch_2 to Switch_2
[2]
> Switch_2 to Switch_3
[2, 3]
_________________________End Printing Shortest Path_____________________________
```

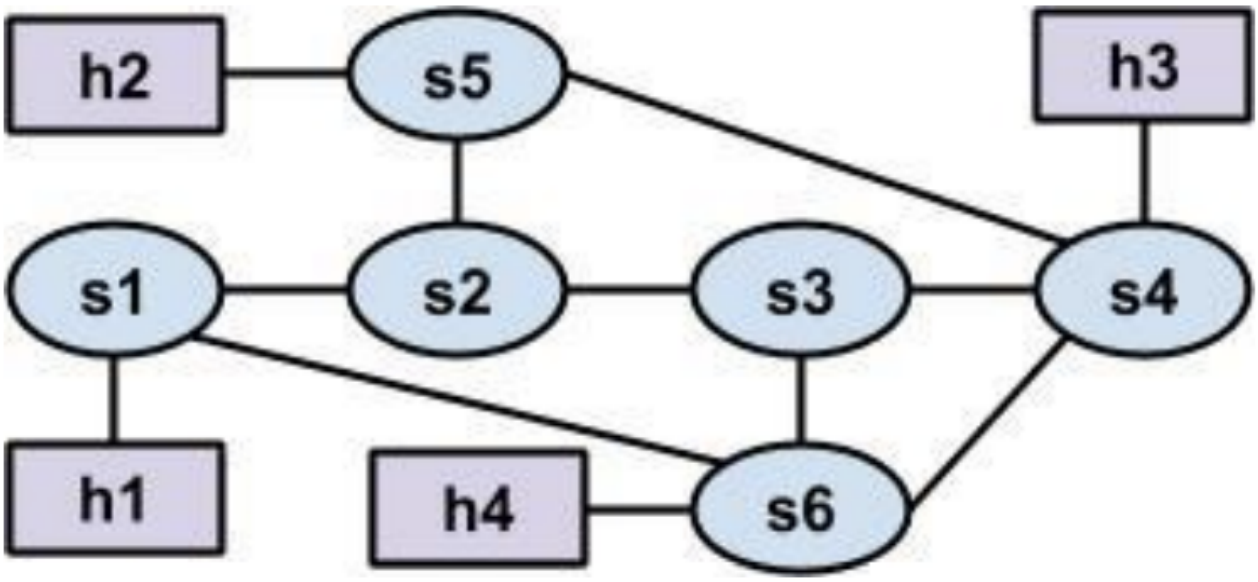
Spanning tree and topology after change

```
Start Spanning Tree Algorithm...
************ Spanning Tree ************
[(2, 3), (3, 2), (3, 1), (1, 3)]
*****************************************
@ Root: Switch_1 ---------------------------------
> For Switch_1 :
 Switch_1/Port_3 -> Switch_3
> For Switch_3 :
 Switch_3/Port_2 -> Switch_2
> For Switch_2 :
@ Root: Switch_3 ---------------------------------
> For Switch_1 :
> For Switch_3 :
 Switch_3/Port_2 -> Switch_2
 Switch_3/Port_3 -> Switch_1
> For Switch_2 :
@ Root: Switch_2 ---------------------------------
> For Switch_1 :
> For Switch_3 :
 Switch_3/Port_3 -> Switch_1
> For Switch_2 :
 Switch_2/Port_3 -> Switch_3
```

Flow table after change

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=591.822s, table=0, n_packets=1121, n_bytes=67260, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:60
 cookie=0x0, duration=175.003s, table=0, n_packets=3, n_bytes=238, priority=0,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=175.003s, table=0, n_packets=4, n_bytes=280, priority=0,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth3"
 cookie=0x0, duration=175.003s, table=0, n_packets=8, n_bytes=560, priority=0,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=175.003s, table=0, n_packets=1, n_bytes=42, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.1 actions=output:"s1-eth3",output:"s1-eth1"
 cookie=0x0, duration=175.003s, table=0, n_packets=0, n_bytes=0, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.3 actions=output:"s1-eth1"
 cookie=0x0, duration=175.003s, table=0, n_packets=1, n_bytes=42, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.2 actions=output:"s1-eth1"
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=591.821s, table=0, n_packets=1124, n_bytes=67440, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:60
 cookie=0x0, duration=175.008s, table=0, n_packets=3, n_bytes=238, priority=0,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth3"
 cookie=0x0, duration=175.007s, table=0, n_packets=4, n_bytes=280, priority=0,dl_dst=00:00:00:00:00:01 actions=output:"s2-eth3"
 cookie=0x0, duration=175.007s, table=0, n_packets=8, n_bytes=560, priority=0,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth1"
 cookie=0x0, duration=175.007s, table=0, n_packets=1, n_bytes=42, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.1 actions=output:"s2-eth1"
 cookie=0x0, duration=175.007s, table=0, n_packets=0, n_bytes=0, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.3 actions=output:"s2-eth1"
 cookie=0x0, duration=175.007s, table=0, n_packets=1, n_bytes=42, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.2 actions=output:"s2-eth3",output:"s2-eth1"
*** s3 ------------------------------------------------------------------------
 cookie=0x0, duration=591.832s, table=0, n_packets=1316, n_bytes=78960, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:60
 cookie=0x0, duration=175.013s, table=0, n_packets=8, n_bytes=560, priority=0,dl_dst=00:00:00:00:00:01 actions=output:"s3-eth3"
 cookie=0x0, duration=175.013s, table=0, n_packets=8, n_bytes=560, priority=0,dl_dst=00:00:00:00:00:02 actions=output:"s3-eth2"
 cookie=0x0, duration=175.013s, table=0, n_packets=6, n_bytes=476, priority=0,dl_dst=00:00:00:00:00:03 actions=output:"s3-eth1"
 cookie=0x0, duration=175.013s, table=0, n_packets=1, n_bytes=42, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.1 actions=output:"s3-eth2",output:"s3-eth1"
 cookie=0x0, duration=175.013s, table=0, n_packets=0, n_bytes=0, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.3 actions=output:"s3-eth2",output:"s3-eth3",output:"s3-eth1"
 cookie=0x0, duration=175.013s, table=0, n_packets=1, n_bytes=42, priority=0,arp,dl_dst=ff:ff:ff:ff:ff:ff,arp_spa=10.0.0.2 actions=output:"s3-eth3",output:"s3-eth1"
```

3.2 Mininet someloops



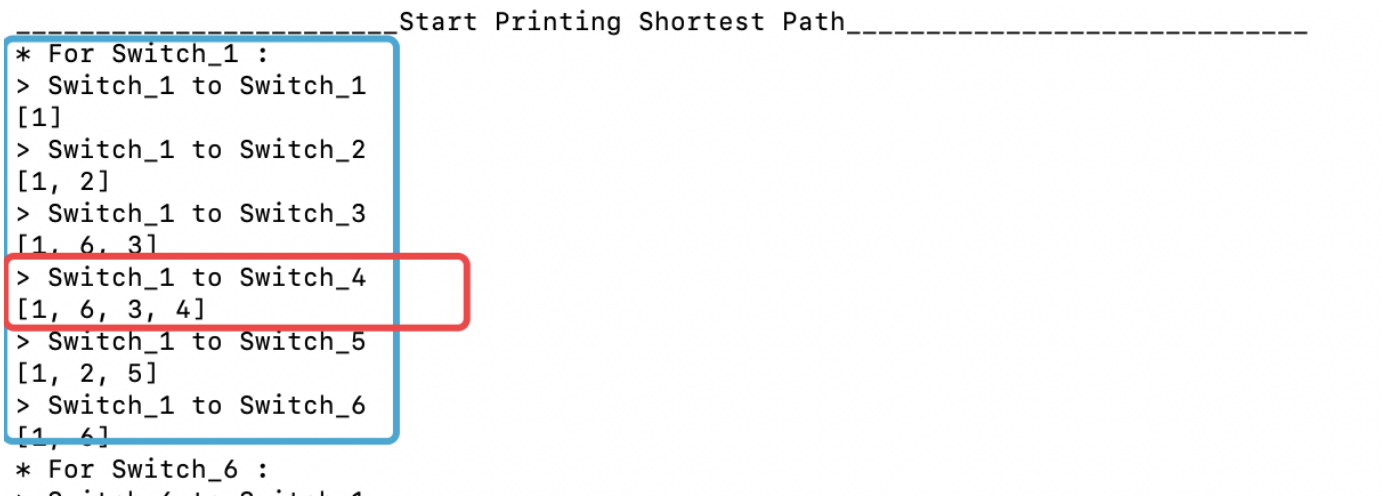
1. Print Shortest path from s1 to other switches.

```
_____Start Printing Shortest Path_______
* For Switch_1 :
> Switch_1 to Switch_1
[1]
> Switch_1 to Switch_2
[1, 2]
> Switch_1 to Switch_3
[1, 6, 3]
> Switch_1 to Switch_4
[1, 6, 4]
> Switch_1 to Switch_5
[1, 2, 5]
> Switch_1 to Switch_6
[1, 6]
* For Switch_6 :
> Switch_6 to Switch_1
[6, 1]
> Switch_6 to Switch_2
[6, 1, 2]
> Switch_6 to Switch_3
```

2. **Change topology**

```
mininet> link s4 s6 down
```

3. **Print changed Shortest path from s1 to other switches.**



4. Contribution

Id	name	Percentage	Task
11712738	武羿	33.3%	Implement shortest path and flooding without loops, Controller ARP packet handle
11712121	胡玉斌	33.3%	Configure the environment，implement flooding without loops
11612003	彭可	33.3%	Implement shortest path and alogrithms :Dijskta and Prim

5. Conclusion

Through this project, we have a deeper understanding of the SDN. We learned how to code with RYU and how to use Mininet. It is very conveninet to control the whole network using just one controller. We also learned how to implement the shortest path and flooding without loops in practice.

The problem we met during the project: it is very important to "sudo mn -c ", that is to clean the previous mininet network every time you want to start a new one. Otherwise, the function of your code may not work well unexpectedly, even your codes are totally right.