SDN Report

Background

The understanding for SDN

Software Defined Network is a kind of network structure. It divides the network into two parts, the controll plane and the data plane. Data plane consists of switches and hosts, which have specific rules to decide how to forwarding the packets. And the Control plane decides when and how to change the flow rules in each switch. SDN uses Openflow as its protocol.

In this project, we design the control plane. We build a virtual topology graph in the control plane, and use them to manage the flow table in each switch and ARP table in each host.

The understanding for RYU

RYU is a python library that can solve the messages and packets of Openflow protocol. We can use it to monitor the mininet events and collect the information which helps us to build the Graph mapping the topology in mininet. But RYU can not influence the topology in mininet.

In this project, we use RYU to collect information that can help us build the virtual topology graph.

The understanding for mininet

mininet is a python extension library that can build and change a network topology in the mininet plane. It provides a CLI for user to change and display the topology.

In this project, mininet plays a role as data plane. And our controller can design how it works.

Implementation

Topological graph

To store the virtual network structure in mininet as the topology which can be run Dijikastra algorithm on, we use the adjacency list to record the information of all switches, and the dictionaries to record the links of each switch and the corresponding ports. Additionally, we implement the function of changing the topology structure accordingly when adding or deleting a switch or link in mininet .

The class of the topology:

```
class graph():
    def __init__(self):
        self.switchs_list = []

    def add_switch(self, s):
        self.switchs_list.append(s)

    def add_link(self, src, src_port, dst_id, dst_port):
        src.add_neighbors(src_port, dst_id, dst_port)
```

```
    def delete_switch(self, s):
        self.switchs_list.remove(s)

    def delete_link(self, src, src_port):
        src.delete_neighbors(src_port)
```

The class of each switch:

```
class TMSwitch(Device):
    def __init__(self, name, switch):
        super(TMSwitch, self).__init__(name)
        self.neighbors = dict()
        self.switch = switch
        self.ports = switch.ports

    def get_dpid(self):
        return self.switch.dp.id

    def get_ports(self):
        return self.switch.ports

    def get_dp(self):
        return self.switch.dp

    def add_neighbors(self, src_port, dsr_id, dst_port):
        self.neighbors[src_port] = (dsr_id, dst_port, 0)

    def delete_neighbors(self, src_port):
        self.neighbors[src_port] = None

    def get_neighbors(self):
        return self.neighbors
```

Forwarding table

After the topology of all switches and links being set up, we apply the Dijikastra algorithm to every switch and get D(v) dictionary which stores the length of shortest path from any other switches to the source and P(v) dictionary which stores the precursor switch of any other switches along the shortest path to the source. According to P(v), we can get the forwarding table of every switch which stores the destination switch and the corresponding output port.

The method that converts P(v) to forwarding table:

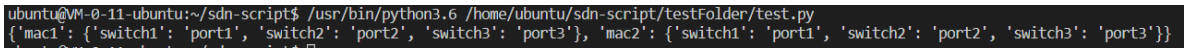
```
for s in self.graph.switchs_list:
                s_id = s.get_dpid()
                final_id = s_id
                if s_id != src:
                    if P[s_id] != None:
                        pre = P[s_id][1]
                        while pre != src:
                            s_id = pre
                            pre = P[s_id][1]
                        fw_dst = final_id
                        fw_port = P[s_id][2]
                        fw_tb[fw_dst] = fw_port
            self.forward_table[switch.get_dpid()] = fw_tb
```

The dictionary format of P(v) is: (v_port, pre_id, pre_port)

flow table

In order to store the result of Dijikastra algorithm, l use a dictionary whose value is another dictionary. The key of the first dictionary is the mac address of the packet destination. The key of the second dictionary is the switch.dp.id, the value is port id.

The sample of this dictionary:




```
def add_forwarding_rule(self, datapath, dl_dst, port):
        ofctl = OfCtl.factory(datapath, self.logger)
        actions = [datapath.ofproto_parser.OFPActionOutput(port)]
        ofctl.set_flow(cookie=0, priority=0, dl_type=ether_types.ETH_TYPE_IP,
                       dl_vlan=VLANID_NONE, dl_dst=dl_dst, actions=actions)
```

- datapath represents the switch
- dl_dst represents the mac address of the packet destination
- port represents the id of port

This function will set the flow table of a switch. Tell it when you receive a packet to a mac address, to which port should you forwarding.

```
def changeFlowTable(self):
        for mac in self.tm.macList:
            if self.tm.macList[mac] is not None:
                for switch in self.tm.macList[mac]:
                    self.add_forwarding_rule(
                        self.tm.all_switch[switch].dp, mac, self.tm.macList[mac]
[switch])
```

This function shows how to use the dictionary to set the flow table of all the switches.

ARP reply

```
send_arp(arp_opcode=2, vlan_id=VLANID_NONE, dst_mac=arp_msg.src_mac,
sender_mac=arp_table[arp_msg.dst_ip], 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)
```

usage: when a host need to ping another host by ip but doesn't know the mac address of it, it will send a arp request to the switch. The switch will send a arp reply.

parameter illustration:

arp_opcode=2----ARP REPLY

send_ip, send_mac, target_ip, target_mac----the address of ARP reply's sender and reciver

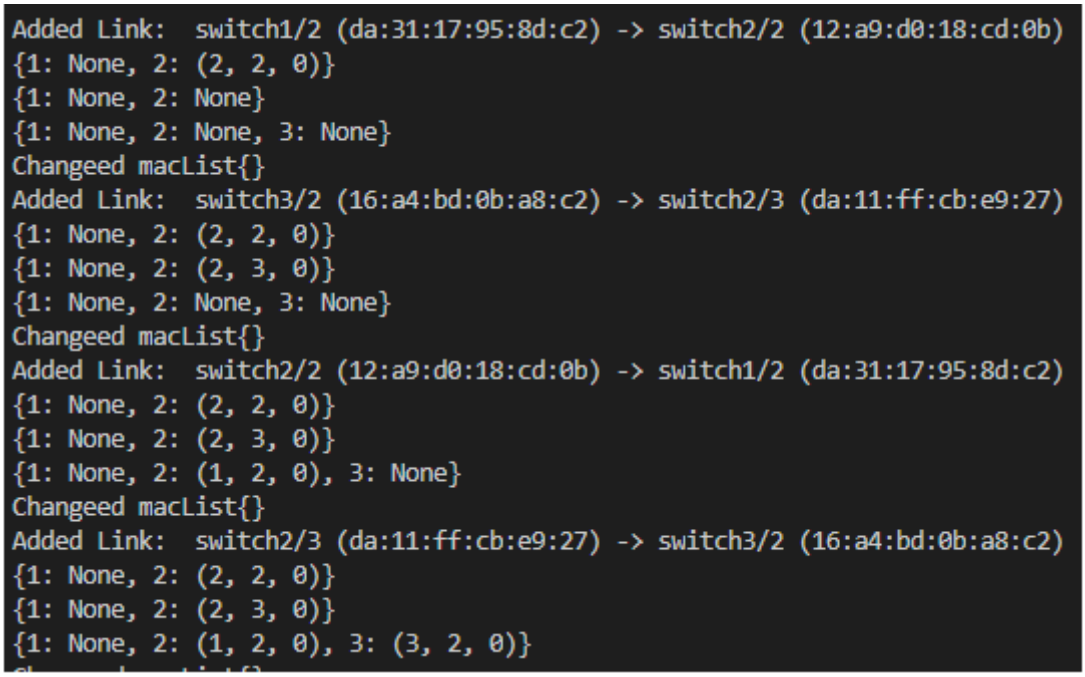
src_port, output_port ----the port number of switch

Test

Topological graph

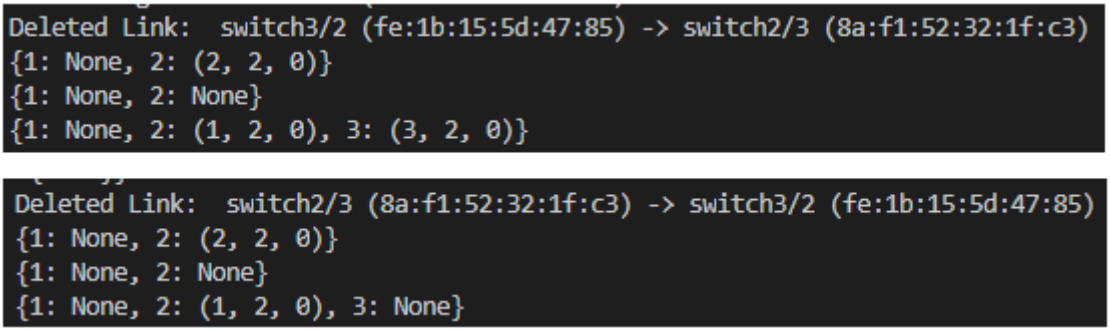
I test the structure of the topology by printing out the neighbors of every switch when a link is added:

When I set up a "linear 3" network in mininet, the output is:



As the figure shows, every time a link is added, the neighbors information of the three switches in the order of "switch 1, switch 2, switch 3" will be printed in the form of {port: (neighbor_id, neighbor_port, port_state)}, and state 0 represents the port is up.

When I execute "link s2 s3 down":



As the figures show, switch 2 deletes neighbor switch 3 and switch 3 deletes neighbor switch 2.

Forwarding table

I test the shortest path algorithm and the forwarding table by printing the final forwarding table information to check if all paths are recored correctly when a link is added.

When I set up a "linear 3" network in mininet, the output is:

```
Added Link:  switch2/2 (c2:33:88:27:46:24) -> switch1/2 (ea:f1:ee:f9:05:3f)
forwardTable{1: {2: 2}, 3: {}, 2: {}}
Changeed macList{}
Added Link:  switch2/3 (9a:e5:06:0d:ba:15) -> switch3/2 (8e:3e:f8:bc:fd:5c)
forwardTable{1: {3: 2, 2: 2}, 3: {1: 2, 2: 2}, 2: {}}
Changeed macList{}
Added Link:  switch1/2 (ea:f1:ee:f9:05:3f) -> switch2/2 (c2:33:88:27:46:24)
forwardTable{1: {3: 2, 2: 2}, 3: {1: 2, 2: 2}, 2: {1: 2}}
Changeed macList{}
Added Link:  switch3/2 (8e:3e:f8:bc:fd:5c) -> switch2/3 (9a:e5:06:0d:ba:15)
forwardTable{1: {3: 2, 2: 2}, 3: {1: 2, 2: 2}, 2: {1: 2, 3: 3}}
```

As the figure shows, when a link is added, the forwarding table will update.

When I execute "link s2 s3 down":

```
Deleted Link:  switch3/2 (8e:3e:f8:bc:fd:5c) -> switch2/3 (9a:e5:06:0d:ba:15)
forwardTable{1: {3: 2, 2: 2}, 3: {1: 2, 2: 2}, 2: {1: 2}}
```

```
Deleted Link:  switch2/3 (9a:e5:06:0d:ba:15) -> switch3/2 (8e:3e:f8:bc:fd:5c)
forwardTable{1: {2: 2}, 3: {}, 2: {1: 2}}
```

As the figures show, switch 2 deletes switch 3 item from its forwarding table and switch 3 deletes neighbor switch 2 item from its forwarding table.

flow table

To test the function of this module, l statically set a dictionary of sample topology single 3, and call changeFlowTable() function each time when a host is added into the network to check the result.

```
{'00:00:00:00:00:01': {1: 1}}
Host Added:  00:00:00:00:00:02 (IPs:  ['10.0.0.2']) on switch1/2 (ea:
1d:8d:d4:fc:cc)
add host host_00:00:00:00:00:02 in control
{'00:00:00:00:00:01': {1: 1}, '00:00:00:00:00:02': {1: 2}}
Host Added:  00:00:00:00:00:03 (IPs:  ['10.0.0.3']) on switch1/3 (3a:
50:00:b9:a4:d5)
add host host_00:00:00:00:00:03 in control
{'00:00:00:00:00:01': {1: 1}, '00:00:00:00:00:02': {1: 2}, '00:00:00:
00:00:03': {1: 3}}
```

Then here is the flow table:

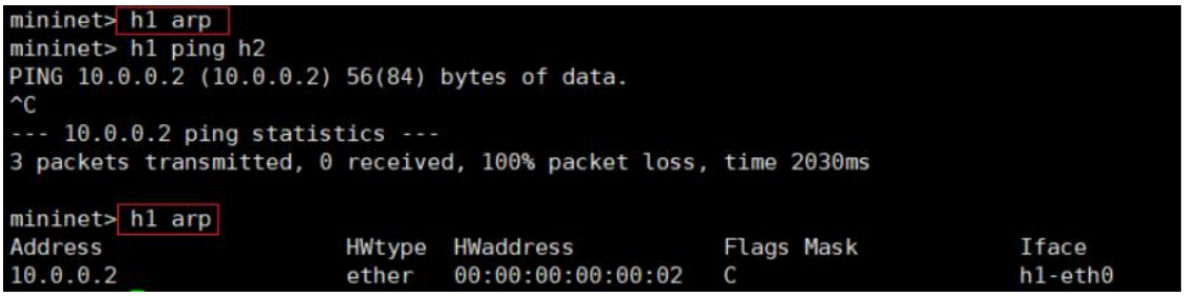
```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------
----------
 cookie=0x0, duration=161.114s, table=0, n_packets=0, n_bytes=0, prio
rity=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER
:60
 cookie=0x0, duration=160.100s, table=0, n_packets=4, n_bytes=392, pr
iority=0,ip,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=160.100s, table=0, n_packets=4, n_bytes=392, pr
iority=0,ip,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=160.100s, table=0, n_packets=4, n_bytes=392, pr
iority=0,ip,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
```

It prove that the function of set-flow is the same as l expect.

ARP reply

Without ping, host arp table is empty.

After ping, the host know the mac address of another host.



Contribution

Li Ya chen:

- Build the Graph
- Run the Dijikastra algorithm.

Xu Tian yuan:

- Set the Flow table of all the switches based on the algorithm.
- Make sense of the operational principle of SDN and all the example codes provided by SA.

Lin ao:

- Design the function to send the ARP response packets
- Make sense of the operational principle of SDN and all the example codes provided by SA.

Conclusion

What have we learned in this project?

A: we have learned the operational principle of SDN and design some functions of control plane by ourselves.

What problems have we met?

A: At some cases, even if we have set all the ARP table and flow table well, the probability of loss is still very high at the first several PING. Then with the help of classmates, we found the problem and fixed it by delete one parameter in a set_flow().