Report of SDN Project

Background

About SDN, the control plane create the devices' network and maintain the graph. Every time a device is added or deleted, the control plane updates the graph. Update means that use LA algorithm to get the minimum distance from every node to other nodes in the altered graph. In this way can we get the flow table through the link of ports. And in this way we can get the shortest path from every two nodes in the graph after LA algorithm.

RYU is a package for simulating the control plane and data plane, and make changes in Mininet. Mininet is used to create the SDN network rapidly, and support the network like real network.

Implement And Test

Every time update the flow table. First, run the Dijkstra algorism to get new flow table.

```
def dijkstra(self):
    self.list = [{} for i in range(len(self.switches) + 1)]
    self.flow_table = {}
    for sw in self.switches:
        sID = sw.get_dpid()
        heap = []
        heapq.heappush(heap, (sID, 0))
        visited = []
        dist = self.init_distance(sID)
        while len(heap) > 0:
            switch, distance = heapq.heappop(heap)
            visited.append(sID)
            neighbor = [i for i in self.switches if i.get_dpid() == switch]
[0].neighbors
            for node in neighbor:
                dst = node.dpid
                if dst not in visited and distance + 1 < dist[dst]:
                    dist[dst] = distance + 1
                    heapq.heappush(heap, (dst, distance + 1))
                    self.list[sID][dst] = switch
                    self.flow_table[(dst, sID)] = node.port_no
```

After LA algorism in `shortest_python.py` use `self.add_forwarding` to implement the flow table. There are two situations:

1. If host is connected to this switch the port should be the port connected to host of the switch.
2. Otherwise, the port should be the port in the flow table.

```
for i in self.tm.switches:
    for j in self.tm.hosts:
        if i.get_dpid() == j.switch_id:
            self.add_forwarding_rule(i.get_dp(), j.get_mac(), j.switch_port)
    else:
        if (i.get_dpid(), j.switch_id) in self.tm.flow_table:
            self.add_forwarding_rule(i.get_dp(), j.get_mac(),
            self.tm.flow_table[(i.get_dpid(), j.switch_id)])
```

Then show the flow table so that we can check the result.

```
Host Added:  00:00:00:00:00:05 (IPs:  ['10.0.0.5']) on switch5/1 (02:6d:d2:55:38:62)
@@@ FLOW TABLE START @@@
Device  2-> 1: Go Port  2;  Device  3-> 1: Go Port  2;  Device  4-> 1: Go Port  2;
Device  5-> 1: Go Port  2;  Device  4-> 3: Go Port  2;  Device  2-> 3: Go Port  3;
Device  1-> 3: Go Port  2;  Device  5-> 3: Go Port  2;  Device  5-> 4: Go Port  2;
Device  3-> 4: Go Port  3;  Device  2-> 4: Go Port  3;  Device  1-> 4: Go Port  2;
Device  1-> 2: Go Port  2;  Device  3-> 2: Go Port  2;  Device  4-> 2: Go Port  2;
Device  5-> 2: Go Port  2;  Device  4-> 5: Go Port  3;  Device  3-> 5: Go Port  3;
Device  2-> 5: Go Port  3;  Device  1-> 5: Go Port  2;  @@@ FLOW TABLE END @@@
```

Then show the shortest path from every device to other devices. Use the result of Dijkstra algorism, first append the goal node to list. Second, get the device `self.list [root] [goal]` which is the input device from root to goal in Dijkstra algorism. Do this loop until the input device is root.

```
def shortest_path(self, root):
    List = [[] for i in range(len(self.switches) + 1)]
    for sw in self.switches:
        sID = sw.get_dpid()
        if (root, sID) not in self.flow_table.keys() or (sID, root) not in
self.flow_table.keys():
            continue
        if sID != root:
            List[sID].append(sID)
            tmp = sID
            if root >= len(self.list):
                continue
            x = tmp in self.list[root].keys()
            count = 0
            while x and self.list[root][tmp] != root:
                count += 1
                tmp = self.list[root][tmp]
                List[sID].insert(0, tmp)
            List[sID].insert(0, root)
    return List
```

Finally show the graph.

```
%%% SHORTEST PATH BEGIN %%%
Switch 1:
 * To Switch  2: [1, 2]
 * To Switch  3: [1, 2, 3]
 * To Switch  4: [1, 2, 3, 4]
 * To Switch  5: [1, 2, 3, 4, 5]
Switch 3:
 * To Switch  1: [3, 2, 1]
 * To Switch  2: [3, 2]
 * To Switch  4: [3, 4]
 * To Switch  5: [3, 4, 5]
Switch 4:
 * To Switch  1: [4, 3, 2, 1]
 * To Switch  2: [4, 3, 2]
 * To Switch  3: [4, 3]
 * To Switch  5: [4, 5]
Switch 2:
 * To Switch  1: [2, 1]
 * To Switch  3: [2, 3]
 * To Switch  4: [2, 3, 4]
 * To Switch  5: [2, 3, 4, 5]
Switch 5:
 * To Switch  1: [5, 4, 3, 2, 1]
 * To Switch  2: [5, 4, 3, 2]
 * To Switch  3: [5, 4, 3]
 * To Switch  4: [5, 4]
%%% SHORTEST PATH END %%%
&&& TOPOLOGY BEGIN &&&
 1 <->  2  3 <->  2  5 <->  4  4 <->  5
 2 <->  1  2 <->  3  4 <->  3  3 <->  4

&&& TOPOLOGY END &&&
```

Test

Create a network loop:

```
qhurc@qhurc-vb:~/p$ sudo python run_mininet.py linear 5
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (s2, s1) (s3, s2) (s4, s3) (s5, s4)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Sending ARPing from host h1
*** Sending ARPing from host h2
*** Sending ARPing from host h3
*** Sending ARPing from host h4
*** Sending ARPing from host h5
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

disable one link ：

```
Deleted Link:  switch2/3 (4a:34:b4:bd:b3:41) -> switch3/2 (9a:bc:01:1d:8c:b4)
@@@ FLOW TABLE START @@@
Device  2-> 1: Go Port  2;  Device  4-> 3: Go Port  2;  Device  5-> 3: Go Port  2;
Device  5-> 4: Go Port  2;  Device  3-> 4: Go Port  3;  Device  1-> 2: Go Port  2;
Device  4-> 5: Go Port  3;  Device  3-> 5: Go Port  3;  @@@ FLOW TABLE END @@@
%%% SHORTEST PATH BEGIN %%%
Switch 1:
 * To Switch  2: [1, 2]
Switch 3:
 * To Switch  4: [3, 4]
 * To Switch  5: [3, 4, 5]
Switch 4:
 * To Switch  3: [4, 3]
 * To Switch  5: [4, 5]
Switch 2:
 * To Switch  1: [2, 1]
Switch 5:
 * To Switch  3: [5, 4, 3]
 * To Switch  4: [5, 4]
%%% SHORTEST PATH END %%%
&&& TOPOLOGY BEGIN &&&
 1 <->  2  5 <->  4  4 <->  5  2 <->  1
 4 <->  3  3 <->  4
&&& TOPOLOGY END &&&
Port Changed:  switch2/3 (4a:34:b4:bd:b3:41):  UP
Port Changed:  switch2/3 (4a:34:b4:bd:b3:41):  DOWN
Port Changed:  switch3/2 (9a:bc:01:1d:8c:b4):  DOWN
```

Contribution

We code together and discuss together to implement SDN. Each one does half of the work in this project.

Conclusion

During this lab, we learnt how to implement a SDN, how to connect two device, how to implement Dijkstra algorism, how to implement the flow table and how to get the shortest path between two devices.

Problems that we have encountered:

1. When get the shortest path between two devices cost too much to search the network again. And we find we can use a way that from the goal back to the root device.
2. Host not has neighbor so we change the value neighbor the switch from device.
3. we should unify the value to stand for the `switch`, `id(int)` or `switch(device)`
4. the graph is bidirectional so the two ends should add neighbor the other end.
5. when delete a switch we need to delete the corresponding neighbor of the neighbors of deleted switch.