

# Ryu的应用开发（四） 基于跳数的最短路径转发原理

---

## 一：实现最短跳数转发

### （一）原理

推文：迪杰斯特拉算法和弗洛伊德算法

## 二：代码实现

### （一）全部代码

（二）注意：由于各种版本的不同，导致我们使用函数的时候可能有所不同，需要我们自己去调试

1.安装networkx模块，若是下载太慢或者出错，换国内源：

2.出现self.network.add\_edge(dpid, src, {'port':in\_port})使用时，会出现参数太多

3.学会代码调试和思考

## 三：代码讲解

（一）不变代码：实现初始连接处理和公共函数--下发流表

（二）实现获取网络拓扑结构

（三）实现下一跳端口获取（根据图获取最短路径，从中获取信息）

（四）实现包接收，计算最短路径，按照最短路径进行动作下发《重点》

## 三：实验演示：注意开启顺序，否则可能导致其他错误

（一）启动Ryu控制器 observe-links命令会导致控制器在运行期间会不间断地发送LLDP数据包进行链路探测

（二）启动Mininet

（三）Ryu查看信息

## 一：实现最短跳数转发

掌握基于Ryu开发最短路径转发应用：基于网络信息进行复杂网络算法实现

### （一）原理

# 基于跳数的最短路径转发原理

- 拓扑信息发现
  - 周期发送LLDP报文，发现链路信息
  - 使用Networkx来存储拓扑信息
- 根据链路信息计算最佳转发路径
  - Dijkstra + Floyd
  - 使用Networkx实现最短路径计算
- 根据最短路径，安装流表项
  - 实现批量下发流表项方法

推文：[迪杰斯特拉算法和弗洛伊德算法](#)

## 二：代码实现

### (一) 全部代码

```

1  from ryu.base import app_manager
2  from ryu.controller.handler import set_ev_cls
3  from ryu.controller import ofp_event
4  from ryu.controller.handler import MAIN_DISPATCHER, CONFIG_DISPATCHER
5  from ryu.lib.packet import packet, ethernet
6  from ryu.topology import event
7  from ryu.topology.api import get_switch, get_link
8  from ryu.ofproto import ofproto_v1_3
9
10 import networkx as nx
11
12 class MyShortestForwarding(app_manager.RyuApp):
13     '''
14     class to achive shortest path to forward, based on minimum hop count
15     '''
16     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
17
18     def __init__(self, *args, **kwargs):
19         super(MyShortestForwarding, self).__init__(*args, **kwargs)
20
21         #set data structor for topo construction
22         self.network = nx.DiGraph()          #store the dj graph
23         self.paths = {}                     #store the shortest path
24         self.topology_api_app = self
25
26     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
27     def switch_features_handler(self, ev):
28         '''
29         manage the initial link between switch and controller
30         '''
31         msg = ev.msg
32         datapath = msg.datapath
33         ofproto = datapath.ofproto
34         ofp_parser = datapath.ofproto_parser
35
36         match = ofp_parser.OFPMatch()        #for all packet first arrive, ma
37         actions = [ofp_parser.OFPActionOutput(
38             ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUF
39             FER
40             )]
41
42         self.add_flow(datapath, 0, match, actions)
43     def add_flow(self, datapath, priority, match, actions):

```

```

44         '''
45         fulfil the function to add flow entry to switch
46         '''
47         ofproto = datapath.ofproto
48         ofp_parser = datapath.ofproto_parser
49
50         inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
51
52         mod = ofp_parser.OFPFlowMod(datapath=datapath,priority=priority,match=match,instructions=inst)
53
54         datapath.send_msg(mod)
55
56
57     @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
58     def packet_in_handler(self,ev):
59         '''
60         manage the packet which comes from switch
61         '''
62         #first get event information
63         msg = ev.msg
64         datapath = msg.datapath
65         ofproto = datapath.ofproto
66         ofp_parser = datapath.ofproto_parser
67
68         in_port = msg.match['in_port']
69         dpid = datapath.id
70
71         #second get ethernet protocol message
72         pkt = packet.Packet(msg.data)
73         eth_pkt = pkt.get_protocol(ethernet.ethernet)
74
75         eth_src = eth_pkt.src      #note: mac info willn't change in network
76
77         eth_dst = eth_pkt.dst
78
79         out_port = self.get_out_port(datapath,eth_src,eth_dst,in_port)
80         actions = [ofp_parser.OFPActionOutput(out_port)]
81
82         if out_port != ofproto.OFPP_FLOOD:
83             match = ofp_parser.OFPMatch(in_port=in_port,eth_dst=eth_dst)
84             self.add_flow(datapath,1,match,actions)
85
86         out = ofp_parser.OFPPacketOut(
87             datapath=datapath,buffer_id=msg.buffer_id,in_port=in_port,
88             actions=actions,data=msg.data

```

```

88         )
89
90         datapath.send_msg(out)
91
92         @set_ev_cls(event.EventSwitchEnter, [CONFIG_DISPATCHER, MAIN_DISPATCHER
    ]) #event is not from openflow protocol, is come from switchs` state c
    hanged, just like: link to controller at the first time or send packet t
    o controller
93     def get_topology(self, ev):
94         '''
95         get network topo construction, save info in the dict
96         '''
97
98         #store nodes info into the Graph
99         switch_list = get_switch(self.topology_api_app, None) #-----
    ----need to get info, by debug
100        switches = [switch.dp.id for switch in switch_list]
101        self.network.add_nodes_from(switches)
102
103        #store links info into the Graph
104        link_list = get_link(self.topology_api_app, None)
105        #port_no, in_port -----need to debug, get diffirent
    from both
106        links = [(link.src.dpid, link.dst.dpid, {'attr_dict': {'port': link.d
    st.port_no}}) for link in link_list] #add edge, need src, dst, weight
107        self.network.add_edges_from(links)
108
109        links = [(link.dst.dpid, link.src.dpid, {'attr_dict': {'port': link.
    dst.port_no}}) for link in link_list]
110        self.network.add_edges_from(links)
111
112     def get_out_port(self, datapath, src, dst, in_port):
113         '''
114         datapath: is current datapath info
115         src, dst: both are the host info
116         in_port: is current datapath in_port
117         '''
118         dpid = datapath.id
119
120         #the first :Doesn't find src host at graph
121     if src not in self.network:
122         self.network.add_node(src)
123         self.network.add_edge(dpid, src, attr_dict={'port': in_port})
124         self.network.add_edge(src, dpid)
125         self.paths.setdefault(src, {})
126
127         #second: search the shortest path, from src to dst host
128     if dst in self.network:

```

```

129         if dst not in self.paths[src]:          #if not cache src to dst pa
th,then to find it
130
131             path = nx.shortest_path(self.network,src,dst)
132             self.paths[src][dst]=path
133
134             path = self.paths[src][dst]
135             next_hop = path[path.index(dpid)+1]
136             #print("1oooooooooooooooooooo")
137             #print(self.network[dpid][next_hop])
138             out_port = self.network[dpid][next_hop]['attr_dict']['port']
139             #print("2oooooooooooooooooooo")
140             #print(out_port)
141
142             #get path info
143             #print("6666666666 find dst")
144             print(path)
145         else:
146             out_port = datapath.ofproto.OFPP_FLOOD          #By flood, to find
dst, when dst get packet, dst will send a new back,the graph will record
dst info
147
148             #print("8888888888 not find dst")
149         return out_port

```

(二) 注意：由于各种版本的不同，导致我们使用函数的时候可能有所不同，需要我们去调试

### 1.安装networkx模块，若是下载太慢或者出错，换国内源：

在末尾加上-i <https://pypi.tuna.tsinghua.edu.cn/simple>

```
1 pip3 install networkx
2 pip3 install multiqc
```

2.出现self.network.add\_edge(dpid, src, {'port':in\_port})使用时，会出现参数太多

```
1 self.network.add_edge(dpid, src, attr_dict={'port': in_port}) attr_dict是
   对我们提供的扩展成熟
```

### 3.学会代码调试和思考

## 三：代码讲解

### (一) 不变代码：实现初始连接处理和公共函数--下发流表

```
Python | 复制代码

1  @set_ev_cls(ofp_event.EventOFPSwitchFeatures,CONFIG_DISPATCHER)
2  def switch_features_handler(self,ev):
3      '''
4      manage the initial link between switch and controller
5      '''
6      msg = ev.msg
7      datapath = msg.datapath
8      ofproto = datapath.ofproto
9      ofp_parser = datapath.ofproto_parser
10
11     match = ofp_parser.OFPMatch() #for all packet first arrive, match it successful, send it to controller
12     actions = [ofp_parser.OFPACTIONOutput(
13         ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER
14         )]
15
16     self.add_flow(datapath, 0, match, actions)
17
18     def add_flow(self,datapath,priority,match,actions):
19         '''
20         fulfil the function to add flow entry to switch
21         '''
22         ofproto = datapath.ofproto
23         ofp_parser = datapath.ofproto_parser
24
25         inst = [ofp_parser.OFPIInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
26
27         mod = ofp_parser.OFPFlowMod(datapath=datapath,priority=priority,match=match,instructions=inst)
28
29         datapath.send_msg(mod)
```

### (二) 实现获取网络拓扑结构

```

1 class MyShortestForwarding(app_manager.RyuApp):
2     '''
3     class to achive shortest path to forward, based on minimum hop count
4     '''
5     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
6
7     def __init__(self,*args,**kwargs):
8         super(MyShortestForwarding,self).__init__(*args,**kwargs)
9
10        #set data structor for topo construction
11        self.network = nx.DiGraph()          #store the dj graph    设置图结构
12        self.paths = {}                    #store the shortest path
13        self.topology_api_app = self
14
15        @set_ev_cls(event.EventSwitchEnter, [CONFIG_DISPATCHER,MAIN_DISPATCHER]
16        )    #event is not from openflow protocol, is come from switchs` state cha
17            nged, just like: link to controller at the first time or send packet to co
18            ntroller
19
20        def get_topology(self,ev):
21            '''
22            get network topo construction, save info in the dict    由于监听交换
23            机信息进入，触发，注意事件不在flow协议类，在from ryu.topology import event中
24            '''
25
26            #store nodes info into the Graph
27            switch_list = get_switch(self.topology_api_app,None)    #-----
28            ---need to get info,by debug
29            switches = [switch.dp.id for switch in switch_list]
30            self.network.add_nodes_from(switches)
31
32            #store links info into the Graph
33            link_list = get_link(self.topology_api_app,None)
34            #port_no, in_port -----need to debug, get diffirent f
35            rom both
36            links = [(link.src.dpid,link.dst.dpid,{'attr_dict':{'port':link.ds
37            t.port_no}}) for link in link_list]    #add edge, need src,dst,weightht
38            self.network.add_edges_from(links)
39
40            links = [(link.dst.dpid,link.src.dpid,{'attr_dict':{'port':link.d
41            st.port_no}}) for link in link_list]
42            self.network.add_edges_from(links)

```



补充: `event.EventSwitchEnter`---由于监听交换机信息进入, 触发, 注意事件不在flow协议类, 在`from ryu.topology import event`中

(三) 实现下一跳端口获取 (根据图获取最短路径, 从中获取信息)

```

1 def get_out_port(self, datapath, src, dst, in_port):
2     '''
3     datapath: is current datapath info
4     src, dst: both are the host info
5     in_port: is current datapath in_port
6     '''
7     dpid = datapath.id
8
9     #the first :Doesn't find src host at graph
10    if src not in self.network:      #根据src主机是否在网络中, 决定是否新添
    加进入
11        self.network.add_node(src)
12        self.network.add_edge(dpid, src, attr_dict={'port': in_port})
13        self.network.add_edge(src, dpid)
14        self.paths.setdefault(src, {})    #设置数据结构: 用于保存每个源主机
    到各个目的主机的最短路径{src1:{dst1:[],dst2:[],dst3:[]....},src2:{dst1:[],dst
    2:[],dst3:[]....},}
15
16    #second: search the shortest path, from src to dst host
17    if dst in self.network:
18        if dst not in self.paths[src]:    #if not cache src to dst pat
    h, then to find it
19            path = nx.shortest_path(self.network, src, dst)
20            self.paths[src][dst] = path
21
22            path = self.paths[src][dst]
23            next_hop = path[path.index(dpid)+1]    #根据数据结构获取下一跳data
    path信息
24            #print("10000000000000000000")
25            #print(self.network[dpid][next_hop])
26            out_port = self.network[dpid][next_hop]['attr_dict']['port']
    #根据该id和下一跳id去获取出端口, 进行数据转发
27            #print("20000000000000000000")
28            #print(out_port)
29
30            #get path info
31            #print("6666666666 find dst")
32            print(path)
33        else:    #否则是泛洪处理
34            out_port = datapath.ofproto.OFPP_FLOOD    #By flood, to find d
    st, when dst get packet, dst will send a new back, the graph will record ds
    t info
35            #print("8888888888 not find dst")
36            return out_port

```

## （四）实现包接收，计算最短路径，按照最短路径进行动作下发《重点》

Python | 复制代码

```
1  @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
2  def packet_in_handler(self,ev):
3      '''
4      manage the packet which comes from switch
5      '''
6      #first get event infomation
7      msg = ev.msg
8      datapath = msg.datapath
9      ofproto = datapath.ofproto
10     ofp_parser = datapath.ofproto_parser
11
12     in_port = msg.match['in_port']
13     dpid = datapath.id
14
15     #second get ethernet protocol message
16     pkt = packet.Packet(msg.data)
17     eth_pkt = pkt.get_protocol(ethernet.ethernet)
18
19     eth_src = eth_pkt.src      #note: mac info willn't  change in netwo
rk
20     eth_dst = eth_pkt.dst
21
22     out_port = self.get_out_port(datapath,eth_src,eth_dst,in_port)  #
这里进行获取下一跳端口
23     actions = [ofp_parser.OFPACTIONOutput(out_port)]
24
25     if out_port != ofproto.OFPP_FLOOD:
26         match = ofp_parser.OFPMATCH(in_port=in_port,eth_dst=eth_dst)
27         self.add_flow(datapath,1,match,actions)
28
29     out = ofp_parser.OFPPACKETOut(
30         datapath=datapath,buffer_id=msg.buffer_id,in_port=in_port,
31         actions=actions,data=msg.data
32     )
33
34     datapath.send_msg(out)
```

## 三：实验演示：注意开启顺序，否则可能导致其他错误

## (一) 启动Ryu控制器 observe-links命令会导致控制器在运行期间会不间断地发送LLDP数据包进行链路探测

Shell | 复制代码

```
1 ryu-manager my_shortest_forward.py --observe-links --verbose
```

```
^Cnjzy@njzy-Inspiron-5493:~/CODE/python/SDN_Controller/ryu/ryu/app$ ryu-manager my_shortest_forward.py --observe-links
loading app my_shortest_forward.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app my_shortest_forward.py of MyShortestForwarding
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
```

## (二) 启动Mininet

Shell | 复制代码

```
1 sudo mn --topo=tree,2,2 --controller=remote --mac
```

```
njzy@njzy-Inspiron-5493:~$ sudo mn --topo=tree,2,2 --controller=remote --mac
[sudo] password for njzy:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> █
```

```
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)
```

### (三) Ryu查看信息

```
njzy@njzy-Inspiron-5493:~/CODE/python/SDN_Controller/ryu/ryu/app$ ryu-manager my_shortest_forward.py --observe-links
loading app my_shortest_forward.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app my_shortest_forward.py of MyShortestForwarding
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
['00:00:00:00:00:02', 2, '00:00:00:00:00:01']
['00:00:00:00:00:01', 2, '00:00:00:00:00:02']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:04', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:02', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:04', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:02', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:04', 3, '00:00:00:00:00:03']
['00:00:00:00:00:03', 3, '00:00:00:00:00:04']
```