

Ryu的应用开发（五）网络拓扑发现

一：实验简介

- （一）网络拓扑信息：
- （二）用邻接矩阵展示
- （三）主机信息展示

二：代码实现

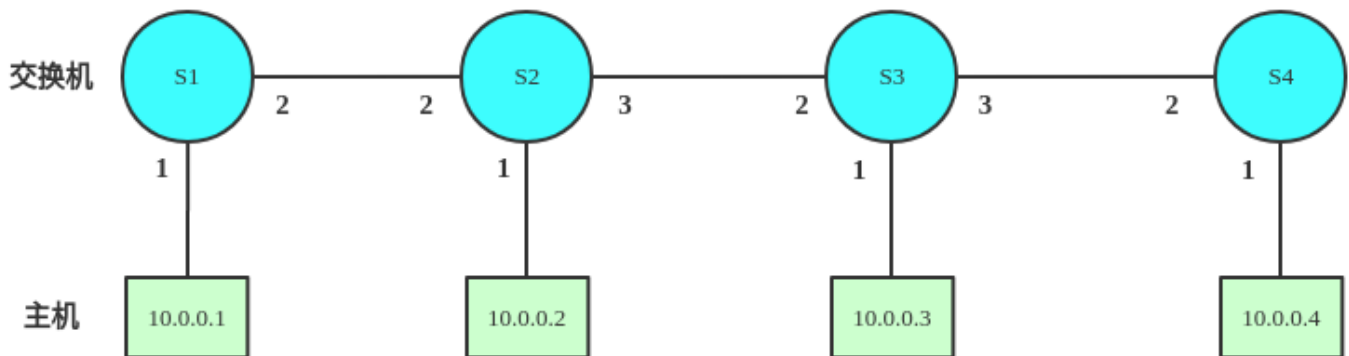
- （一）导入模块
- （二）数据结构
- （三）实现基本openflow消息处理
- （四）实现拓扑发现功能
- （五）全部代码

三：实验验证

- （一）启动Ryu控制器
- （二）启动mininet
- （三）结果显示

一：实验简介

（一）网络拓扑信息：



其中1,2,3表示该交换机对应的端口号！！

(二) 用邻接矩阵展示

出节点 \ 入节点	S1	S2	S3	S4
S1	0	(2,0)	0	0
S2	(2,0)	0	(3,0)	0
S3	0	(2,0)	0	(3,0)
S4	0	0	(2,0)	0

其中左侧列S1,S2,S3,S4表示出节点，----->，上面S1,S2,S3,S4表示入节点。

(m,0)，m表示出节点的端口--->入节点，0暂时表示两个节点之间的时延信息！

(三) 主机信息展示

IP	交换机
['10.0.0.1']	-----s1
['10.0.0.2']	-----s2
['10.0.0.3']	-----s3
['10.0.0.4']	-----s4

二：代码实现

(一) 导入模块

```
1 from ryu.base import app_manager
2
3 from ryu.ofproto import ofproto_v1_3
4
5 from ryu.controller import ofp_event
6 from ryu.controller.handler import MAIN_DISPATCHER, CONFIG_DISPATCHER, DEAD_
  DISPATCHER #只是表示datapath数据路径的状态
7 from ryu.controller.handler import set_ev_cls
8
9 from ryu.lib import hub
10 from ryu.lib.packet import packet, ethernet
11
12 from ryu.topology import event, switches
13 from ryu.topology.api import get_switch, get_link, get_host
14
15 import threading #需要设置线程锁
```

(二) 数据结构

```

1  DELAY_MONITOR_PERIOD = 5
2  LOCK = threading.RLock() #实现线程锁
3
4  class TopoDetect(app_manager.RyuApp):
5      OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
6
7      def __init__(self, *args, **kwargs):
8          super(TopoDetect, self).__init__(*args, **kwargs)
9          self.topology_api_app = self #用于保持对象本身，后面get_switch等方法需
          要（我们也可以直接传入self）
10         self.link_list = None #保存所有的link信息，由get_link获得
11         self.switch_list = None #保存所有的switch信息，由get_switch获得
12         self.host_list = None #保存所有的host信息，由get_host获得
13
14         self.dpid2id = {} #获取交换机dpid,以及自定义id--->{dpid:i
            d}
15         self.id2dpid = {} #对应上面的self.dpid2id, 翻转即可，因为我
            们使用id进行建立邻接矩阵，这两个结构方便查找
16         self.dpid2switch = {} #保存dpid和对应的交换机全部信息---->通过
            矩阵获得id, 然后获得dpid, 最后获得交换机对象信息
17
18         self.ip2host = {} #根据ip, 保存主机对象信息--->{ip:host}
19         self.ip2switch = {} #根据ip, 获取当前主机是连接到哪个交换机--->
            {ip:dpid}
20
21         self.net_size = 0 #记录交换机个数（网络拓扑大小）
22         self.net_topo = [] #用于保存邻接矩阵
23
24         self.net_flag = False #标识：用于表示拓扑网络拓扑self.net_topo
            是否已经更新完成
25         self.net_arrived = 0 #标识：用于表示网络中交换机消息到达，每当一
            个交换机到达以后，我们设置+1
26         self.monitor_thread = hub.spawn(self._monitor) #协程实现定时检测网络
            拓扑

```

（三）实现基本openflow消息处理

```

1     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
2     def switch_feature_handle(self, ev):
3         """
4         datapath中有配置消息到达
5         """
6         LOCK.acquire()
7         self.net_arrived += 1 #表示有1个交换机到达
8         LOCK.release()
9         #print("-----XXXXXXXXXXXX-----%d-----XXXXXXXXXXXX-----swit
ch_feature_handle"%self.net_arrived)
10        msg = ev.msg
11        datapath = msg.datapath
12        ofproto = datapath.ofproto
13        ofp_parser = datapath.ofproto_parser
14
15        match = ofp_parser.OFPMatch()
16
17        actions = [ofp_parser.OFPACTIONOutput(ofproto.OFPP_CONTROLLER, ofpr
oto.OFPCML_NO_BUFFER)]
18
19        self.add_flow(datapath=datapath, priority=0, match=match, actions=act
ions, extra_info="config infomation arrived!!")
20
21
22     def add_flow(self, datapath, priority, match, actions, idle_timeout=0, hard_
timeout=0, extra_info=None):
23         #print("-----add_flow:")
24         if extra_info != None:
25             print(extra_info)
26             ofproto = datapath.ofproto
27             ofp_parser = datapath.ofproto_parser
28
29             inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIO
NS, actions)]
30
31             mod = ofp_parser.OFPFlowMod(datapath=datapath, priority=priority,
32                                         idle_timeout=idle_timeout,
33                                         hard_timeout=hard_timeout,
34                                         match=match, instructions=inst)
35             datapath.send_msg(mod);
36
37     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
38     def packet_in_handler(self, ev):
39         #print("-----packet_in_handler")
40         msg = ev.msg

```

```

41     datapath = msg.datapath
42     ofproto = datapath.ofproto
43     ofp_parser = datapath.ofproto_parser
44
45     dpid = datapath.id
46     in_port = msg.match['in_port']
47
48     pkt = packet.Packet(msg.data)
49     eth_pkt = pkt.get_protocol(ethernet.ethernet)
50     dst = eth_pkt.dst
51     src = eth_pkt.src
52
53     #self.logger.info("-----Controller %s get packet, Ma
54 c address from: %s send to: %s , send from datapath: %s,in port is: %s"
55     #                                     ,dpid,src,dst,dpid,in_port)
56     #self.get_topology(None)

```

注意：对于packet_in消息，我们没有处理，所以整个网络（交换机之间的链路）是无法工作通信的，

但是各个交换机可以与控制器通信（switch_feature_handle实现），主机也可以和边缘交换机通信，

所以控制器可以获取网络拓扑信息！！！！

（四）实现拓扑发现功能

Python | 复制代码

```

1  def _monitor(self):
2      """
3      协程实现伪并发，探测拓扑状态
4      """
5      while True:
6          #print("-----_monitor")
7          self._host_add_handler(None) #主机单独提取处理
8          self.get_topology(None)
9          if self.net_flag:
10             try:
11                 self.show_topology()
12             except Exception as err:
13                 print("Please use cmd: pingall to detect topology and
14 wait a moment")
15             hub.sleep(DELAY_MONITOR_PERIOD) #5秒一次

```

```

1     @set_ev_cls([event.EventHostAdd])
2     def _host_add_handler(self, ev):      #主机信息单独处理，不属于网络拓扑
3         self.host_list = get_host(self.topology_api_app)  #3.需要使用pingal
l, 主机通过与边缘交换机连接，才能告诉控制器
4         #获取主机信息字典ip2host{ipv4:host object} ip2switch{ipv4:dpid}
5         for i, host in enumerate(self.host_list):
6             self.ip2switch["%s"%host.ipv4] = host.port.dpid
7             self.ip2host["%s"%host.ipv4] = host
8
9
10        events = [event.EventSwitchEnter, event.EventSwitchLeave,
11                  event.EventSwitchReconnected,
12                  event.EventPortAdd, event.EventPortDelete,
13                  event.EventPortModify,
14                  event.EventLinkAdd, event.EventLinkDelete]
15        @set_ev_cls(events)
16        def get_topology(self, ev):
17            if not self.net_arrived:
18                return
19
20            LOCK.acquire()
21            self.net_arrived -= 1
22            if self.net_arrived < 0:
23                self.net_arrived = 0
24            LOCK.release()
25
26            self.net_flag = False
27            self.net_topo = []
28
29            print("-----get_topology")
30            #获取所有的交换机、链路
31            self.switch_list = get_switch(self.topology_api_app)  #1.只要交换机
与控制器联通，就可以获取
32            self.link_list = get_link(self.topology_api_app)      #2.在ryu启动
时，加上--observe-links即可用于拓扑发现
33
34            #获取交换机字典id2dpid{id:dpid} dpid2switch{dpid:switch object}
35            for i, switch in enumerate(self.switch_list):
36                self.id2dpid[i] = switch.dp.id
37                self.dpid2id[switch.dp.id] = i
38                self.dpid2switch[switch.dp.id] = switch
39
40
41            #根据链路信息，开始获取拓扑信息
42            self.net_size = len(self.id2dpid) #表示网络中交换机个数

```

```

43         for i in range(self.net_size):
44             self.net_topo.append([0]*self.net_size)
45
46         for link in self.link_list:
47             src_dpid = link.src.dpid
48             src_port = link.src.port_no
49
50             dst_dpid = link.dst.dpid
51             dst_port = link.dst.port_no
52
53         try:
54             sid = self.dpid2id[src_dpid]
55             did = self.dpid2id[dst_dpid]
56         except KeyError as e:
57             print("-----Error:get KeyError with link infomati
58 on(%s)"%e)
59             return
60             self.net_topo[sid][did] = [src_port,0] #注意：这里0表示存在链路，
        后面可以修改为时延
61             self.net_topo[did][sid] = [dst_port,0] #注意：修改为列表，不要用元
        组，元组无法修改，我们后面要修改时延
62
63         self.net_flag = True #表示网络拓扑创建成功

```

Python | 复制代码

```

1 def show_topology(self):
2     print("-----show_topology")
3     print("-----switch network-----")
4     line_info = " "
5     for i in range(self.net_size):
6         line_info+="          s%-5d          "%self.id2dpid[i]
7         print(line_info)
8     for i in range(self.net_size):
9         line_info = "s%d          "%self.id2dpid[i]
10    for j in range(self.net_size):
11        if self.net_topo[i][j] == 0:
12            line_info+="%-22d"%0
13        else:
14            line_info+="(%d,%.12f)          "%tuple(self.net_topo[i][j])
15            print(line_info)
16
17    print("-----host 2 switch-----")
18    for key,val in self.ip2switch.items():
19        print("%s---s%d"%(key,val))

```


(五) 全部代码

```

11
12 from ryu.topology import event, switches
13 from ryu.topology.api import get_switch, get_link, get_host
14
15 import threading, time, random
16
17 DELAY_MONITOR_PERIOD = 5
18
19 class TopoDetect(app_manager.RyuApp):
20     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
21
22     def __init__(self, *args, **kwargs):
23         super(TopoDetect, self).__init__(*args, **kwargs)
24         self.topology_api_app = self
25         self.name = "topology"
26         self.link_list = None
27         self.switch_list = None
28         self.host_list = None
29
30         self.dpid2id = {}
31         self.id2dpid = {}
32         self.dpid2switch = {}
33
34         self.ip2host = {}
35         self.ip2switch = {}
36
37         self.net_size = 0
38         self.net_topo = []
39
40         self.net_flag = False
41         self.net_arrived = 0
42
43         self.monitor_thread = hub.spawn(self._monitor)
44
45     def _monitor(self):
46         """
47         协程实现伪并发，探测拓扑状态
48         """
49         while True:
50             #print("-----_monitor")
51             self._host_add_handler(None) #主机单独提取处理
52             self.get_topology(None)
53             if self.net_flag:
54                 try:
55                     self.show_topology()
56

```

```

56         except Exception as err:
57             print("Please use cmd: pingall to detect topology and wait a moment")
58             hub.sleep(DELAY_MONITOR_PERIOD) #5秒一次
59
60
61     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
62     def switch_feature_handle(self, ev):
63         """
64         datapath中有配置消息到达
65         """
66         #print("-----XXXXXXXXXXXX-----%d-----XXXXXXXXXXXX-----switch_feature_handle"%self.net_arrived)
67         #print("-----%s-----", ev.msg)
68         msg = ev.msg
69         datapath = msg.datapath
70         ofproto = datapath.ofproto
71         ofp_parser = datapath.ofproto_parser
72
73         match = ofp_parser.OFPMatch()
74
75         actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
76
77         self.add_flow(datapath=datapath, priority=0, match=match, actions=actions, extra_info="config information arrived!!")
78
79
80     def add_flow(self, datapath, priority, match, actions, idle_timeout=0, hard_timeout=0, extra_info=None):
81         #print("-----add_flow:")
82         if extra_info != None:
83             print(extra_info)
84         ofproto = datapath.ofproto
85         ofp_parser = datapath.ofproto_parser
86
87
88         inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
89
90         mod = ofp_parser.OFPFlowMod(datapath=datapath, priority=priority, idle_timeout=idle_timeout, hard_timeout=hard_timeout, match=match, instructions=inst)
91
92         datapath.send_msg(mod);
93
94
95     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
96     def packet_in_handler(self, ev):
97
98

```

```

98         #print("-----packet_in_handler")
99         msg = ev.msg
100         datapath = msg.datapath
101         ofproto = datapath.ofproto
102         ofp_parser = datapath.ofproto_parser
103
104         dpid = datapath.id
105         in_port = msg.match['in_port']
106
107         pkt = packet.Packet(msg.data)
108         eth_pkt = pkt.get_protocol(ethernet.ethernet)
109         dst = eth_pkt.dst
110         src = eth_pkt.src
111
112         #self.logger.info("-----Controller %s get packet, Ma
113         c address from: %s send to: %s , send from datapath: %s,in port is: %s"
114         #                                     ,dpid,src,dst,dpid,in_port)
115         self.get_topology(None)
116
117     @set_ev_cls([event.EventHostAdd])
118     def _host_add_handler(self, ev):      #主机信息单独处理，不属于网络拓扑
119         self.host_list = get_host(self.topology_api_app) #3.需要使用pingal
120         l,主机通过与边缘交换机连接，才能告诉控制器
121         #获取主机信息字典ip2host{ipv4:host object} ip2switch{ipv4:dpid}
122         for i, host in enumerate(self.host_list):
123             self.ip2switch["%s"%host.ipv4] = host.port.dpid
124             self.ip2host["%s"%host.ipv4] = host
125
126         events = [event.EventSwitchEnter, event.EventSwitchLeave,
127                 event.EventSwitchReconnected,
128                 event.EventPortAdd, event.EventPortDelete,
129                 event.EventPortModify,
130                 event.EventLinkAdd, event.EventLinkDelete]
131     @set_ev_cls(events)
132     def get_topology(self, ev):
133         #print("-----+++++++-%d-----+++++++-----get
134         _topology"%self.net_arrived)
135
136         self.net_flag = False
137         self.net_topo = []
138
139         #print("-----get_topology")
140         #获取所有的交换机、链路
141         self.switch_list = get_switch(self.topology_api_app) #1.只要交换机
与控制器联通，就可以获取
142         self.link_list = get_link(self.topology_api_app) #2.在ryu启动时，加

```

```

141 上--observe-links即可用于拓扑发现
142
143 #获取交换机字典id2dpid{id:dpid} dpid2switch{dpid:switch object}
144 for i,switch in enumerate(self.switch_list):
145     self.id2dpid[i] = switch.dp.id
146     self.dpid2id[switch.dp.id] = i
147     self.dpid2switch[switch.dp.id] = switch
148
149
150 #根据链路信息，开始获取拓扑信息
151 self.net_size = len(self.id2dpid) #表示网络中交换机个数
152 for i in range(self.net_size):
153     self.net_topo.append([0]*self.net_size)
154
155 for link in self.link_list:
156     src_dpid = link.src.dpid
157     src_port = link.src.port_no
158
159     dst_dpid = link.dst.dpid
160     dst_port = link.dst.port_no
161
162     try:
163         sid = self.dpid2id[src_dpid]
164         did = self.dpid2id[dst_dpid]
165     except KeyError as e:
166         #print("-----Error:get KeyError with link infoma
167         tion(%s)"%e)
168         return
169         self.net_topo[sid][did] = [src_port,0] #注意：这里0表示存在链路，
170         后面可以修改为时延
171         self.net_topo[did][sid] = [dst_port,0] #注意：修改为列表，不要用
172         元组，元组无法修改，我们后面要修改时延
173
174 self.net_flag = True #表示网络拓扑创建成功
175
176 def show_topology(self):
177     print("-----show_topology")
178     print("-----switch network-----")
179     line_info = " "
180     for i in range(self.net_size):
181         line_info+=" "s%-5d "%self.id2dpid[i]
182     print(line_info)
183     for i in range(self.net_size):
184         line_info = "s%d "%self.id2dpid[i]
185         for j in range(self.net_size):
186             if self.net_topo[i][j] == 0:
187                 line_info+="%-22d"%0

```

```

186         else:
187             line_info+="(%d,%.12f)    "%tuple(self.net_topo[i][j]
188     )
189     print(line_info)
190     print("-----host 2 switch-----")
191     for key,val in self.ip2switch.items():

```

三：实验验证

(一) 启动Ryu控制器

```

▼ Shell | 复制代码
1 ryu-manager TopoDetect.py --verbose --observe-links

```

其中--observe-links用于拓扑发现，添加之后用于链路的信息获取！！

```

ld@ld-Lenovo-Product:~/RyuSCP$ ryu-manager TopoDetect.py --verbose --observe-links
loading app TopoDetect.py
require_app: ryu.topology.switches is required by TopoDetect
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app ryu.topology.switches of Switches
instantiating app TopoDetect.py of TopoDetect
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK TopoDetect
CONSUMES EventPortModify
CONSUMES EventSwitchReconnected
CONSUMES EventLinkDelete

```

(二) 启动mininet

```

▼ Shell | 复制代码
1 sudo mn --topo=linear,4 --switch=ovsk --controller=remote --link=tc

```

```

ld@ld-Lenovo-Product:~/openvswitch/openvswitch-2.11.4$ sudo mn --topo=linear,4 -
-switch=ovsk --controller=remote --link=tc
[sudo] password for ld:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:

```

注意：需要在mininet中使用pingall，才能使得交换机获得host存在，从而使得控制器获取host消息！！

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
```

(三) 结果显示

```
-----switch network-----
              s1              s2              s3              s4
s1      0      (2,0.000000000000)      0      0
s2      (2,0.000000000000)      0      (3,0.000000000000)      0
s3      0      (2,0.000000000000)      0      (3,0.000000000000)
s4      0      0      (2,0.000000000000)      0
```

```
-----host 2 switch-----
['10.0.0.1']---s1
['10.0.0.3']---s3
['10.0.0.2']---s2
['10.0.0.4']---s4
```