

Ryu的应用开发（六） 网络拓扑时延探测

一：预备知识

[SDN实验---Ryu的应用开发（五）网络拓扑发现](#)

[Ryu源码之模块功能分析](#)

[Ryu源码之拓扑发现原理分析](#)

二：实验原理

三：时延探测代码实现

[（一）拓扑发现模块（已修改）](#)

[（二）模块导入](#)

[（三）数据结构](#)

[（四）协程获取链路时延](#)

[（五）获取Echo时延](#)

[（六）获取LLDP时延](#)

[（七）根据LLDP和Echo时延，更新网络拓扑图中的权值信息](#)

[（八）显示网络拓扑图和Echo、LLDP时延信息](#)

[（九）全部代码](#)

四：实验测试

[回顾：拓扑代码和时延代码](#)

[（一）启动Ryu](#)

[（二）启动mininet](#)

[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！！](#)

[（三）结果显示](#)

一：预备知识

[SDN实验---Ryu的应用开发（五）网络拓扑发现](#)

[Ryu源码之模块功能分析](#)

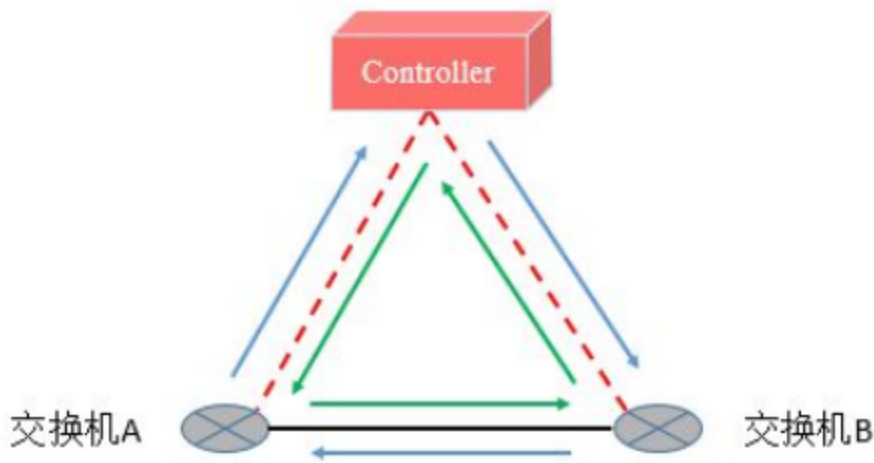
[Ryu源码之拓扑发现原理分析](#)

二：实验原理

网络时延探测应用利用了Ryu自带的Switches模块的数据，获取到了LLDP数据包从控制器下发到交换机A，然后从交换机A到交换机B，再上报给控制器的时延T1，示例见图1的蓝色箭头。

同理反向的时延T2由绿色的箭头组成。

此外，控制器到交换机的往返时延由一个蓝色箭头和一个绿色箭头组成，此部分时延由echo报文测试，分别为Ta，Tb。最后链路的前向后向平均时延 $T = (T1 + T2 - Ta - Tb) / 2$ 。



三：时延探测代码实现

(一) 拓扑发现模块（已修改）

```

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, 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): #修改, 只获取拓扑, 不主动显示!!!
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         hub.sleep(DELAY_MONITOR_PERIOD) #5秒一次
54
55
56     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
57     def switch_feature_handle(self, ev):
58         """
59         datapath中有配置消息到达
60         """
61         #print("-----XXXXXXXXXX-----%d-----XXXXXXXXXX-----swi
62 tch_feature_handle"%self.net_arrived)
63         #print("----%s-----", ev.msg)
64         msg = ev.msg
65         datapath = msg.datapath
66         ofproto = datapath.ofproto
67         ofp_parser = datapath.ofproto_parser
68
69         match = ofp_parser.OFPMatch()
70
71         actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofp
72 roto.OFPCML_NO_BUFFER)]
73
74         self.add_flow(datapath=datapath, priority=0, match=match, actions=ac
75 tions, extra_info="config infomation arrived!")
76
77     def add_flow(self, datapath, priority, match, actions, idle_timeout=0, hard
78 _timeout=0, extra_info=None):
79         #print("-----add_flow:")
80         if extra_info != None:
81             print(extra_info)
82         ofproto = datapath.ofproto
83         ofp_parser = datapath.ofproto_parser
84
85         inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTI
86 ONS, actions)]
87
88         mod = ofp_parser.OFPFlowMod(datapath=datapath, priority=priority,
89 idle_timeout=idle_timeout,
90 hard_timeout=hard_timeout,
91 match=match, instructions=inst)
92
93         datapath.send_msg(mod);
94
95     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
96     def packet_in_handler(self, ev):
97         #print("-----packet_in_handler")

```

```

93     msg = ev.msg
94     datapath = msg.datapath
95     ofproto = datapath.ofproto
96     ofp_parser = datapath.ofproto_parser
97
98     dpid = datapath.id
99     in_port = msg.match['in_port']
100
101     pkt = packet.Packet(msg.data)
102     eth_pkt = pkt.get_protocol(ethernet.ethernet)
103     dst = eth_pkt.dst
104     src = eth_pkt.src
105
106     #self.logger.info("-----Controller %s get packet, Ma
c address from: %s send to: %s , send from datapath: %s,in port is: %s"
107     #                                     ,dpid,src,dst,dpid,in_port)
108     self.get_topology(None)
109
110
111     @set_ev_cls([event.EventHostAdd])
112     def _host_add_handler(self, ev):      #主机信息单独处理, 不属于网络拓扑
113         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
114         #获取主机信息字典ip2host{ipv4:host object} ip2switch{ipv4:dpid}
115         for i, host in enumerate(self.host_list):
116             self.ip2switch["%s"%host.ipv4] = host.port.dpid
117             self.ip2host["%s"%host.ipv4] = host
118
119
120     events = [event.EventSwitchEnter, event.EventSwitchLeave,
121              event.EventSwitchReconnected,
122              event.EventPortAdd, event.EventPortDelete,
123              event.EventPortModify,
124              event.EventLinkAdd, event.EventLinkDelete]
125
126     @set_ev_cls(events)
127     def get_topology(self, ev):
128         #print("-----+++++++-----%d-----+++++++-----get
_topology"%self.net_arrived)
129
130         self.net_flag = False
131         self.net_topo = []
132
133         #print("-----get_topology")
134         #获取所有的交换机、链路
135         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
136         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

```

```

136
137
138 #获取交换机字典id2dpid{id:dpid} dpid2switch{dpid:switch object}
139 for i,switch in enumerate(self.switch_list):
140     self.id2dpid[i] = switch.dp.id
141     self.dpid2id[switch.dp.id] = i
142     self.dpid2switch[switch.dp.id] = switch
143
144
145 #根据链路信息，开始获取拓扑信息
146 self.net_size = len(self.id2dpid) #表示网络中交换机个数
147 for i in range(self.net_size):
148     self.net_topo.append([0]*self.net_size)
149
150 for link in self.link_list:
151     src_dpid = link.src.dpid
152     src_port = link.src.port_no
153
154     dst_dpid = link.dst.dpid
155     dst_port = link.dst.port_no
156
157     try:
158         sid = self.dpid2id[src_dpid]
159         did = self.dpid2id[dst_dpid]
160     except KeyError as e:
161         #print("-----Error:get KeyError with link infoma
162         tion(%s)"%e)
163         return
164         self.net_topo[sid][did] = [src_port,0] #注意：这里0表示存在链路，
165         后面可以修改为时延
166         self.net_topo[did][sid] = [dst_port,0] #注意：修改为列表，不要用
167         元组，元组无法修改，我们后面要修改时延
168
169
170 self.net_flag = True #表示网络拓扑创建成功
171
172 def show_topology(self):
173     print("-----show_topology")
174     print("-----switch network-----")
175     line_info = " "
176     for i in range(self.net_size):
177         line_info+=" "s%-5d "%self.id2dpid[i]
178     print(line_info)
179     for i in range(self.net_size):
180         line_info = "s%d "%self.id2dpid[i]
181         for j in range(self.net_size):
182             if self.net_topo[i][j] == 0:
183                 line_info+="%-22d"%0
184             else:

```

```

181             line_info+="(%d,%.12f)    "%tuple(self.net_topo[i][j]
182         )
183         print(line_info)
184
185     print("-----host 2 switch-----")
186     for key,val in self.ip2switch.items():
187         print("%s---s%d"%(key,val))

```

(二) 模块导入

Python | [复制代码](#)

```

1  from ryu.base import app_manager
2  from ryu.base.app_manager import lookup_service_brick
3
4  from ryu.ofproto import ofproto_v1_3
5
6  from ryu.controller import ofp_event
7  from ryu.controller.handler import MAIN_DISPATCHER,CONFIG_DISPATCHER,DEAD_
  DISPATCHER,HANDSHAKE_DISPATCHER #只是表示datapath数据路径的状态
8  from ryu.controller.handler import set_ev_cls
9
10 from ryu.lib import hub
11 from ryu.lib.packet import packet,ethernet
12
13 from ryu.topology.switches import Switches
14 from ryu.topology.switches import LLDPpacket
15
16 import time

```

(三) 数据结构

```

1 ECHO_REQUEST_INTERVAL = 0.05
2 DELAY_DETECTING_PERIOD = 5
3
4 class DelayDetect(app_manager.RyuApp):
5     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
6
7     def __init__(self, *args, **kwargs):
8         super(DelayDetect, self).__init__(*args, **kwargs)
9         self.name = "delay"
10
11         self.topology = lookup_service_brick("topology") #注意：我们使用look
up_service_brick加载模块实例时，对于我们自己定义的app,我们需要在类中定义self.name。
12         self.switches = lookup_service_brick("switches") #此外，最重要的是：
我们启动本模块DelayDetect时，必须同时启动自定义的模块!!! 比如：ryu-manager ./Top
oDetect.py ./DelayDetect.py --verbose --observe-links
13
14         self.dpid2switch = {} #或者直接为{}，也可以。下面_state_change_handler
也会添加进去
15         self.dpid2echoDelay = {} #记录echo时延
16
17         self.src_sport_dst2Delay = {} #记录LLDP报文测量的时延。实际上可以直接更
新，这里单独记录，为了单独展示 {"src_dpid-srt_port-dst_dpid": delay}
18
19         self.detector_thread = hub.spawn(self._detector)

```

(四) 协程获取链路时延


```
1  def _detector(self):
2      """
3      协程实现伪并发，探测链路时延
4      """
5      while True:
6          if self.topology == None:
7              self.topology = lookup_service_brick("topology")
8          if self.topology.net_flag:
9              #print("-----_detector-----")
10             self._send_echo_request()
11             self.get_link_delay()
12             if self.topology.net_flag:
13                 try:
14                     self.show_delay()
15                     self.topology.show_topology() #拓扑显示
16                 except Exception as err:
17                     print("-----Detect delay failure!!!--
-----")
18             hub.sleep(DELAY_DETECTING_PERIOD) #5秒一次
```

(五) 获取Echo时延

```

1  def _send_echo_request(self):
2      """
3      发生echo报文到datapath
4      """
5      for datapath in self.dpid2switch.values():
6          parser = datapath.ofproto_parser
7          echo_req = parser.OFPEchoRequest(datapath, data=bytes("%.12f"%time.time(), encoding="utf8")) #获取当前时间
8
9          datapath.send_msg(echo_req)
10
11         #重要! 不要同时发送echo请求, 因为它几乎同时会生成大量echo回复。
12         #在echo_reply_处理程序中处理echo_reply时, 会产生大量队列等待延迟。
13         hub.sleep(ECHO_REQUEST_INTERVAL)
14
15     @set_ev_cls(ofp_event.EventOFPEchoReply, [MAIN_DISPATCHER, CONFIG_DISPATCHER, HANDSHAKE_DISPATCHER])
16     def echo_reply_handler(self, ev):
17         """
18         处理echo响应报文, 获取控制器到交换机的链路往返时延
19
20         Controller
21         |
22         echo latency |
23         \ | /
24         Switch
25         """
26         now_timestamp = time.time()
27         try:
28             echo_delay = now_timestamp - eval(ev.msg.data)
29             self.dpid2echoDelay[ev.msg.datapath.id] = echo_delay
30         except:
31             return

```

(六) 获取LLDP时延

补充：前面我们通过lookup_service_brick("switches"), 实例化了switches模块。详细见：<https://www.cnblogs.com/ssyfj/p/14193150.html>。该模块中通过协程实现了周期0.05s发送LLDP数据包。所以我们下面可以直接获取LLDP数据报。

```

1  @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
2  def packet_in_handler(self,ev): #处理到达的LLDP报文，从而获得LLDP时延
3      """
4          Controller
5          |          /\
6          \\/          |
7          Switch----->Switch
8      """
9      msg = ev.msg
10     try:
11         src_dpid,src_outport = LLDPpacket.lldp_parse(msg.data) #获取两个相邻交换机的源交换机dpid和port_no(与目的交换机相连的端口)
12         dst_dpid = msg.datapath.id #获取目的交换机（第二个），因为来到控制器的消息是由第二个（目的）交换机上传过来的
13         dst_inport = msg.match['in_port']
14         if self.switches is None:
15             self.switches = lookup_service_brick("switches") #获取交换机
模块实例
16
17         #获得key（Port类实例）和data（PortData类实例）
18         for port in self.switches.ports.keys(): #开始获取对应交换机端口的
发送时间戳
19             if src_dpid == port.dpid and src_outport == port.port_no:
#匹配key
20                 port_data = self.switches.ports[port] #获取满足key条件的
values值PortData实例，内部保存了发送LLDP报文时的timestamp信息
21                 timestamp = port_data.timestamp
22                 if timestamp:
23                     delay = time.time() - timestamp
24                     self._save_delay_data(src=src_dpid,dst=dst_dpid,src_port=src_outport,lldpdealy=delay)
25             except:
26                 return
27
28     def _save_delay_data(self,src,dst,src_port,lldpdealy):
29         key = "%s-%s-%s"%(src,src_port,dst)
30         self.src_sport_dst2Delay[key] = lldpdealy

```

（七）根据LLDP和Echo时延，更新网络拓扑图中的权值信息

```

1  def get_link_delay(self):
2      """
3      更新图中的权值信息
4      """
5      print("-----get_link_delay-----")
6      for src_sport_dst in self.src_sport_dst2Delay.keys():
7          src,sport,dst = tuple(map(eval,src_sport_dst.split("-")))
8          if src in self.dpid2echoDelay.keys() and dst in self.dpid2
echoDelay.keys():
9              sid,did = self.topology.dpid2id[src],self.topology.dpi
d2id[dst]
10             if self.topology.net_topo[sid][did] != 0:
11                 if self.topology.net_topo[sid][did][0] == sport:
12                     s_d_delay = self.src_sport_dst2Delay[src_sport
_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDelay[dst])/2;
13                     if s_d_delay < 0: #注意：可能出现单向计算时延导致最
后小于0，这是不允许的。则不进行更新，使用上一次原始值
14                         continue
15                     self.topology.net_topo[sid][did][1] = self.src
_sport_dst2Delay[src_sport_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDe
lay[dst])/2

```

(八) 显示网络拓扑图和Echo、LLDP时延信息

```

1      @set_ev_cls(ofp_event.EventOFPStateChange, [MAIN_DISPATCHER, DEAD_DISPATCHER])
2      def _state_change_handler(self, ev):
3          datapath = ev.datapath
4          if ev.state == MAIN_DISPATCHER:
5              if not datapath.id in self.dpид2switch:
6                  self.logger.debug('Register datapath: %016x', datapath.id)
7                  self.dpид2switch[datapath.id] = datapath
8          elif ev.state == DEAD_DISPATCHER:
9              if datapath.id in self.dpид2switch:
10                 self.logger.debug('Unregister datapath: %016x', datapath.id)
11                 del self.dpид2switch[datapath.id]
12
13         if self.topology == None:
14             self.topology = lookup_service_brick("topology")
15         print("-----_state_change_handler-----")
16         print(self.topology.show_topology())
17         print(self.switches)
18
19     def show_delay(self):
20         print("-----show echo delay-----")
21         for key, val in self.dpид2echoDelay.items():
22             print("s%d----%.12f"%(key, val))
23         print("-----show LLDP delay-----")
24         for key, val in self.src_sport_dst2Delay.items():
25             print("s----%.12f"%(key, val))

```

(九) 全部代码

```

1  from ryu.base import app_manager
2  from ryu.base.app_manager import lookup_service_brick
3
4  from ryu.ofproto import ofproto_v1_3
5
6  from ryu.controller import ofp_event
7  from ryu.controller.handler import MAIN_DISPATCHER, CONFIG_DISPATCHER, DEAD
   _DISPATCHER, HANDSHAKE_DISPATCHER #只是表示datapath数据路径的状态
8  from ryu.controller.handler import set_ev_cls
9
10 from ryu.lib import hub
11 from ryu.lib.packet import packet, ethernet
12
13 from ryu.topology.switches import Switches
14 from ryu.topology.switches import LLDPpacket
15
16 import time
17
18 ECHO_REQUEST_INTERVAL = 0.05
19 DELAY_DETECTING_PERIOD = 5
20
21 class DelayDetect(app_manager.RyuApp):
22     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
23
24     def __init__(self, *args, **kwargs):
25         super(DelayDetect, self).__init__(*args, **kwargs)
26         self.name = "delay"
27
28         self.topology = lookup_service_brick("topology") #注意：我们使用loo
   kup_service_brick加载模块实例时，对于我们自己定义的app，我们需要在类中定义self.nam
   e。
29         self.switches = lookup_service_brick("switches") #此外，最重要的是：
   我们启动本模块DelayDetect时，必须同时启动自定义的模块!!! 比如：ryu-manager ./To
   poDetect.py ./DelayDetect.py --verbose --observe-links
30
31         self.dpid2switch = {} #或者直接为{}，也可以。下面_state_change_handle
   r也会添加进去
32         self.dpid2echoDelay = {}
33
34         self.src_sport_dst2Delay = {} #记录LLDP报文测量的时延。实际上可以直接更
   新，这里单独记录，为了单独展示 {"src_dpid-srt_port-dst_dpid": delay}
35
36         self.detector_thread = hub.spawn(self._detector)
37
38     def _detector(self):

```

```

39         """
40         协程实现伪并发，探测链路时延
41         """
42     while True:
43         if self.topology == None:
44             self.topology = lookup_service_brick("topology")
45         if self.topology.net_flag:
46             #print("-----_detector-----")
47             self._send_echo_request()
48             self.get_link_delay()
49             if self.topology.net_flag:
50                 try:
51                     self.show_delay()
52                     self.topology.show_topology()
53                 except Exception as err:
54                     print("-----Detect delay failure!!!-
-----")
55             hub.sleep(DELAY_DETECTING_PERIOD) #5秒一次
56
57     def get_link_delay(self):
58         """
59         更新图中的权值信息
60         """
61         #print("-----get_link_delay-----")
62         for src_sport_dst in self.src_sport_dst2Delay.keys():
63             src,sport,dst = tuple(map(eval,src_sport_dst.split("-")))
64             if src in self.dpid2echoDelay.keys() and dst in self.dpid
2echoDelay.keys():
65                 sid,did = self.topology.dpid2id[src],self.topology.dp
id2id[dst]
66                 if self.topology.net_topo[sid][did] != 0:
67                     if self.topology.net_topo[sid][did][0] == sport:
68                         s_d_delay = self.src_sport_dst2Delay[src_spor
t_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDelay[dst])/2;
69                         if s_d_delay < 0: #注意：可能出现单向计算时延导致
最后小于0，这是不允许的。则不进行更新，使用上一次原始值
70                             continue
71                         self.topology.net_topo[sid][did][1] = self.sr
c_sport_dst2Delay[src_sport_dst]-(self.dpid2echoDelay[src]+self.dpid2echo
Delay[dst])/2
72
73     def _send_echo_request(self):
74         """
75         发生echo报文到datapath
76         """
77         #print("=====send_echo_request=====")
78         #print(self.dpid2switch)
79         for datapath in self.dpid2switch.values():

```

```

80         parser = datapath.ofproto_parser
81         echo_req = parser.OFPEchoRequest(datapath, data=bytes("%.12f"%
time.time()), encoding="utf8")) #获取当前时间
82         #print("=====_send_echo_request=====2=====")
83         datapath.send_msg(echo_req)
84
85         #重要! 不要同时发送echo请求, 因为它几乎同时会生成大量echo回复。
86         #在echo_reply_处理程序中处理echo_reply时, 会产生大量队列等待延迟。
87         hub.sleep(ECHO_REQUEST_INTERVAL)
88
89     @set_ev_cls(ofp_event.EventOFPEchoReply, [MAIN_DISPATCHER, CONFIG_DISPATCHER, HANDSHAKE_DISPATCHER])
90     def echo_reply_handler(self, ev):
91         """
92         处理echo响应报文, 获取控制器到交换机的链路往返时延
93
94         Controller
95         |
96         echo latency |
97         \ | /
98         Switch
99
100         """
101         #print("=====")
102         #print(ev)
103         #print("=====")
104         now_timestamp = time.time()
105         try:
106             echo_delay = now_timestamp - eval(ev.msg.data)
107             self.dpid2echoDelay[ev.msg.datapath.id] = echo_delay
108         except:
109             return
110
111     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
112     def packet_in_handler(self, ev): #处理到达的LLDP报文, 从而获得LLDP时延
113         """
114
115         Controller
116         |          /\
117         \ | /      |
118         Switch----->Switch
119
120         """
121         msg = ev.msg
122         try:
123             src_dpid, src_outport = LLDPpacket.lldp_parse(msg.data) #获取两个相邻交换机的源交换机dpid和port_no(与目的交换机相连的端口)
124             dst_dpid = msg.datapath.id #获取目的交换机(第二个), 因为来到控制器的消息是由第二个(目的)交换机上传过来的
125             dst_inport = msg.match['in_port']

```



```

124         if self.switches is None:
125             self.switches = lookup_service_brick("switches") #获取交换机
    机模块实例
126
127             #获得key (Port类实例) 和data (PortData类实例)
128             for port in self.switches.ports.keys(): #开始获取对应交换机端口的
    发送时间戳
129                 if src_dpidd == port.dpidd and src_outport == port.port_no
    : #匹配key
130                     port_data = self.switches.ports[port] #获取满足key条件
    的values值PortData实例, 内部保存了发送LLDP报文时的timestamp信息
131                     timestamp = port_data.timestamp
132                     if timestamp:
133                         delay = time.time() - timestamp
134                         self._save_delay_data(src=src_dpidd,dst=dst_dpidd,s
    rc_port=src_outport,lldpdealy=delay)
135             except:
136                 return
137
138         def _save_delay_data(self,src,dst,src_port,lldpdealy):
139             key = "%s-%s-%s"%(src,src_port,dst)
140             self.src_sport_dst2Delay[key] = lldpdealy
141
142         @set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER, DEAD_DISP
    ATCHER])
143         def _state_change_handler(self, ev):
144             datapath = ev.datapath
145             if ev.state == MAIN_DISPATCHER:
146                 if not datapath.id in self.dpidd2switch:
147                     self.logger.debug('Register datapath: %016x', datapath.id
    )
148                     self.dpidd2switch[datapath.id] = datapath
149             elif ev.state == DEAD_DISPATCHER:
150                 if datapath.id in self.dpidd2switch:
151                     self.logger.debug('Unregister datapath: %016x', datapath.
    id)
152                     del self.dpidd2switch[datapath.id]
153
154             if self.topology == None:
155                 self.topology = lookup_service_brick("topology")
156                 #print("-----_state_change_handler-----
    -----")
157                 #print(self.topology.show_topology())
158                 #print(self.switches)
159
160         def show_delay(self):
161             #print("-----show echo delay-----
    ----")

```

```
162         for key,val in self.dpid2echoDelay.items():
163             print("s%d----%.12f"%(key,val))
164             #print("-----show LLDP delay-----")
165         ----")
166         for key,val in self.src_sport_dst2Delay.items():
            print("%s----%.12f"%(key,val))
```

四：实验测试

回顾：拓扑代码和时延代码

```

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, 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         hub.sleep(DELAY_MONITOR_PERIOD) #5秒一次
54
55
56     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
57     def switch_feature_handle(self, ev):
58         """
59         datapath中有配置消息到达
60         """
61         #print("-----XXXXXXXXXXXX-----%d-----XXXXXXXXXXXX-----swi
62 tch_feature_handle"%self.net_arrived)
63         #print("-----%s-----", ev.msg)
64         msg = ev.msg
65         datapath = msg.datapath
66         ofproto = datapath.ofproto
67         ofp_parser = datapath.ofproto_parser
68
69         match = ofp_parser.OFPMatch()
70
71         actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofp
72 roto.OFPCML_NO_BUFFER)]
73
74         self.add_flow(datapath=datapath, priority=0, match=match, actions=ac
75 tions, extra_info="config infomation arrived!")
76
77     def add_flow(self, datapath, priority, match, actions, idle_timeout=0, hard
78 _timeout=0, extra_info=None):
79         #print("-----add_flow:")
80         if extra_info != None:
81             print(extra_info)
82         ofproto = datapath.ofproto
83         ofp_parser = datapath.ofproto_parser
84
85         inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTI
86 ONS, actions)]
87
88         mod = ofp_parser.OFPFlowMod(datapath=datapath, priority=priority,
89 idle_timeout=idle_timeout,
90 hard_timeout=hard_timeout,
91 match=match, instructions=inst)
92
93         datapath.send_msg(mod);
94
95     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
96     def packet_in_handler(self, ev):
97         #print("-----packet_in_handler")

```

```

93     msg = ev.msg
94     datapath = msg.datapath
95     ofproto = datapath.ofproto
96     ofp_parser = datapath.ofproto_parser
97
98     dpid = datapath.id
99     in_port = msg.match['in_port']
100
101     pkt = packet.Packet(msg.data)
102     eth_pkt = pkt.get_protocol(ethernet.ethernet)
103     dst = eth_pkt.dst
104     src = eth_pkt.src
105
106     #self.logger.info("-----Controller %s get packet, Mac
107     # address from: %s send to: %s , send from datapath: %s,in port is: %s"
108     #                  ,dpid,src,dst,dpid,in_port)
109     self.get_topology(None)
110
111     @set_ev_cls([event.EventHostAdd])
112     def _host_add_handler(self,ev):      #主机信息单独处理, 不属于网络拓扑
113         self.host_list = get_host(self.topology_api_app) #3.需要使用pingal
114         l,主机通过与边缘交换机连接, 才能告诉控制器
115         #获取主机信息字典ip2host{ipv4:host object} ip2switch{ipv4:dpid}
116         for i,host in enumerate(self.host_list):
117             self.ip2switch["%s"%host.ipv4] = host.port.dpid
118             self.ip2host["%s"%host.ipv4] = host
119
120     events = [event.EventSwitchEnter, event.EventSwitchLeave,
121              event.EventSwitchReconnected,
122              event.EventPortAdd, event.EventPortDelete,
123              event.EventPortModify,
124              event.EventLinkAdd, event.EventLinkDelete]
125
126     @set_ev_cls(events)
127     def get_topology(self,ev):
128         #print("-----+++++++-----%d-----+++++++-----get
129         _topology"%self.net_arrived)
130
131         self.net_flag = False
132         self.net_topo = []
133
134         #print("-----get_topology")
135         #获取所有的交换机、链路
136         self.switch_list = get_switch(self.topology_api_app) #1.只要交换机
137         与控制器联通, 就可以获取
138         self.link_list = get_link(self.topology_api_app) #2.在ryu启动时, 加
139         上--observe-links即可用于拓扑发现

```

```

136
137
138 #获取交换机字典id2dpid{id:dpid} dpid2switch{dpid:switch object}
139 for i,switch in enumerate(self.switch_list):
140     self.id2dpid[i] = switch.dp.id
141     self.dpid2id[switch.dp.id] = i
142     self.dpid2switch[switch.dp.id] = switch
143
144
145 #根据链路信息，开始获取拓扑信息
146 self.net_size = len(self.id2dpid) #表示网络中交换机个数
147 for i in range(self.net_size):
148     self.net_topo.append([0]*self.net_size)
149
150 for link in self.link_list:
151     src_dpid = link.src.dpid
152     src_port = link.src.port_no
153
154     dst_dpid = link.dst.dpid
155     dst_port = link.dst.port_no
156
157     try:
158         sid = self.dpid2id[src_dpid]
159         did = self.dpid2id[dst_dpid]
160     except KeyError as e:
161         #print("-----Error:get KeyError with link infoma
162         tion(%s)"%e)
163         return
164         self.net_topo[sid][did] = [src_port,0] #注意：这里0表示存在链路，
165         后面可以修改为时延
166         self.net_topo[did][sid] = [dst_port,0] #注意：修改为列表，不要用
167         元组，元组无法修改，我们后面要修改时延
168
169
170 self.net_flag = True #表示网络拓扑创建成功
171
172 def show_topology(self):
173     print("-----show_topology")
174     print("-----switch network-----")
175     line_info = " "
176     for i in range(self.net_size):
177         line_info+=" "s%-5d "%self.id2dpid[i]
178     print(line_info)
179     for i in range(self.net_size):
180         line_info = "s%d "%self.id2dpid[i]
181         for j in range(self.net_size):
182             if self.net_topo[i][j] == 0:
183                 line_info+="%-22d"%0
184             else:

```

```
181             line_info+="(%d,%.12f)      "%tuple(self.net_topo[i][j]
182         )
183         print(line_info)
184     print("-----host 2 switch-----")
185     for key,val in self.ip2switch.items():
186         print("%s---s%d"%(key,val))
```

```

1  from ryu.base import app_manager
2  from ryu.base.app_manager import lookup_service_brick
3
4  from ryu.ofproto import ofproto_v1_3
5
6  from ryu.controller import ofp_event
7  from ryu.controller.handler import MAIN_DISPATCHER, CONFIG_DISPATCHER, DEAD
   _DISPATCHER, HANDSHAKE_DISPATCHER #只是表示datapath数据路径的状态
8  from ryu.controller.handler import set_ev_cls
9
10 from ryu.lib import hub
11 from ryu.lib.packet import packet, ethernet
12
13 from ryu.topology.switches import Switches
14 from ryu.topology.switches import LLDPpacket
15
16 import time
17
18 ECHO_REQUEST_INTERVAL = 0.05
19 DELAY_DETECTING_PERIOD = 5
20
21 class DelayDetect(app_manager.RyuApp):
22     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
23
24     def __init__(self, *args, **kwargs):
25         super(DelayDetect, self).__init__(*args, **kwargs)
26         self.name = "delay"
27
28         self.topology = lookup_service_brick("topology") #注意：我们使用loo
   kup_service_brick加载模块实例时，对于我们自己定义的app，我们需要在类中定义self.name。
29         self.switches = lookup_service_brick("switches") #此外，最重要的是：
   我们启动本模块DelayDetect时，必须同时启动自定义的模块!!! 比如：ryu-manager ./To
   poDetect.py ./DelayDetect.py --verbose --observe-links
30
31         self.dpid2switch = {} #或者直接为{}，也可以。下面_state_change_handle
   r也会添加进去
32         self.dpid2echoDelay = {}
33
34         self.src_sport_dst2Delay = {} #记录LLDP报文测量的时延。实际上可以直接更
   新，这里单独记录，为了单独展示 {"src_dpid-srt_port-dst_dpid": delay}
35
36         self.detector_thread = hub.spawn(self._detector)
37
38     def _detector(self):

```



```

39         """
40         协程实现伪并发，探测链路时延
41         """
42     while True:
43         if self.topology == None:
44             self.topology = lookup_service_brick("topology")
45         if self.topology.net_flag:
46             #print("-----_detector-----")
47             self._send_echo_request()
48             self.get_link_delay()
49             if self.topology.net_flag:
50                 try:
51                     self.show_delay()
52                     self.topology.show_topology()
53                 except Exception as err:
54                     print("-----Detect delay failure!!!-
-----")
55             hub.sleep(DELAY_DETECTING_PERIOD) #5秒一次
56
57     def get_link_delay(self):
58         """
59         更新图中的权值信息
60         """
61         #print("-----get_link_delay-----")
62         for src_sport_dst in self.src_sport_dst2Delay.keys():
63             src,sport,dst = tuple(map(eval,src_sport_dst.split("-")))
64             if src in self.dpid2echoDelay.keys() and dst in self.dpid
2echoDelay.keys():
65                 sid,did = self.topology.dpid2id[src],self.topology.dp
id2id[dst]
66                 if self.topology.net_topo[sid][did] != 0:
67                     if self.topology.net_topo[sid][did][0] == sport:
68                         s_d_delay = self.src_sport_dst2Delay[src_spor
t_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDelay[dst])/2;
69                         if s_d_delay < 0: #注意：可能出现单向计算时延导致
最后小于0，这是不允许的。则不进行更新，使用上一次原始值
70                             continue
71                         self.topology.net_topo[sid][did][1] = self.sr
c_sport_dst2Delay[src_sport_dst]-(self.dpid2echoDelay[src]+self.dpid2echo
Delay[dst])/2
72
73     def _send_echo_request(self):
74         """
75         发生echo报文到datapath
76         """
77         #print("=====send_echo_request=====")
78         #print(self.dpid2switch)
79         for datapath in self.dpid2switch.values():

```

```

80         parser = datapath.ofproto_parser
81         echo_req = parser.OFPEchoRequest(datapath, data=bytes("%.12f"%
time.time()), encoding="utf8")) #获取当前时间
82         #print("=====_send_echo_request=====2=====")
83         datapath.send_msg(echo_req)
84
85         #重要! 不要同时发送echo请求, 因为它几乎同时会生成大量echo回复。
86         #在echo_reply_处理程序中处理echo_reply时, 会产生大量队列等待延迟。
87         hub.sleep(ECHO_REQUEST_INTERVAL)
88
89     @set_ev_cls(ofp_event.EventOFPEchoReply, [MAIN_DISPATCHER, CONFIG_DISPATCHER, HANDSHAKE_DISPATCHER])
90     def echo_reply_handler(self, ev):
91         """
92         处理echo响应报文, 获取控制器到交换机的链路往返时延
93
94         Controller
95         |
96         echo latency |
97         \ | /
98         Switch
99
100         """
101         #print("=====")
102         #print(ev)
103         #print("=====")
104         now_timestamp = time.time()
105         try:
106             echo_delay = now_timestamp - eval(ev.msg.data)
107             self.dpid2echoDelay[ev.msg.datapath.id] = echo_delay
108         except:
109             return
110
111     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
112     def packet_in_handler(self, ev): #处理到达的LLDP报文, 从而获得LLDP时延
113         """
114
115         Controller
116         |          /\
117         \ | /      |
118         Switch----->Switch
119
120         """
121         msg = ev.msg
122         try:
123             src_dpid, src_outport = LLDPpacket.lldp_parse(msg.data) #获取两个相邻交换机的源交换机dpid和port_no(与目的交换机相连的端口)
124             dst_dpid = msg.datapath.id #获取目的交换机(第二个), 因为来到控制器的消息是由第二个(目的)交换机上传过来的
125             dst_inport = msg.match['in_port']

```

```

124         if self.switches is None:
125             self.switches = lookup_service_brick("switches") #获取交换机
    机模块实例
126
127             #获得key (Port类实例) 和data (PortData类实例)
128             for port in self.switches.ports.keys(): #开始获取对应交换机端口的
    发送时间戳
129                 if src_dpид == port.dpид and src_outport == port.port_no
    : #匹配key
130                     port_data = self.switches.ports[port] #获取满足key条件
    的values值PortData实例, 内部保存了发送LLDP报文时的timestamp信息
131                     timestamp = port_data.timestamp
132                     if timestamp:
133                         delay = time.time() - timestamp
134                         self._save_delay_data(src=src_dpид,dst=dst_dpид,s
    rc_port=src_outport,lldpdealy=delay)
135             except:
136                 return
137
138         def _save_delay_data(self,src,dst,src_port,lldpdealy):
139             key = "%s-%s-%s"%(src,src_port,dst)
140             self.src_sport_dst2Delay[key] = lldpdealy
141
142         @set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER, DEAD_DISP
    ATCHER])
143         def _state_change_handler(self, ev):
144             datapath = ev.datapath
145             if ev.state == MAIN_DISPATCHER:
146                 if not datapath.id in self.dpид2switch:
147                     self.logger.debug('Register datapath: %016x', datapath.id
    )
148                     self.dpид2switch[datapath.id] = datapath
149             elif ev.state == DEAD_DISPATCHER:
150                 if datapath.id in self.dpид2switch:
151                     self.logger.debug('Unregister datapath: %016x', datapath.
    id)
152                     del self.dpид2switch[datapath.id]
153
154             if self.topology == None:
155                 self.topology = lookup_service_brick("topology")
156                 #print("-----_state_change_handler-----
    -----")
157                 #print(self.topology.show_topology())
158                 #print(self.switches)
159
160         def show_delay(self):
161             #print("-----show echo delay-----
    ----")

```

```

162         for key,val in self.dpid2echoDelay.items():
163             print("s%d----%.12f"%(key,val))
164             #print("-----show LLDP delay-----")
165         ----")
166         for key,val in self.src_sport_dst2Delay.items():
            print("%s----%.12f"%(key,val))

```

(一) 启动Ryu

ryu-manager ./TopoDetect.py ./DelayDetect.py --verbose --observe-links

```

ld@ld-Lenovo-Product:~/RyuSCP$ ryu-manager ./TopoDetect.py ./DelayDetect.py --verbose --observe-links
loading app ./TopoDetect.py
require_app: ryu.topology.switches is required by TopoDetect
loading app ./DelayDetect.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ./DelayDetect.py of DelayDetect
instantiating app ./TopoDetect.py of TopoDetect
instantiating app ryu.topology.switches of Switches
BRICK delay
CONSUMES EventOFPPStateChange
CONSUMES EventOFPPacketIn
CONSUMES EventOFPEchoReply
BRICK topology
CONSUMES EventPortDelete
CONSUMES EventSwitchLeave
CONSUMES EventOFPSwitchFeatures
CONSUMES EventPortModify
CONSUMES EventSwitchReconnected
CONSUMES EventLinkDelete
CONSUMES EventHostAdd
CONSUMES EventOFPPacketIn
CONSUMES EventPortAdd
CONSUMES EventSwitchEnter
CONSUMES EventLinkAdd
BRICK ofp_event
PROVIDES EventOFPSwitchFeatures TO {'topology': {'config'}}
PROVIDES EventOFPPStateChange TO {'delay': {'dead', 'main'}, 'switches': {'dead', 'main'}}
PROVIDES EventOFPPacketIn TO {'delay': {'main'}, 'topology': {'main'}, 'switches': {'main'}}
PROVIDES EventOFPPortStatus TO {'switches': {'main'}}
PROVIDES EventOFPEchoReply TO {'delay': {'config', 'main', 'handshake'}}
CONSUMES EventOFPEchoRequest
CONSUMES EventOFPErrormsg
CONSUMES EventOFPEchoReply
CONSUMES EventOFPSwitchFeatures
CONSUMES EventOFPHello
CONSUMES EventOFPPortDescStatsReply
CONSUMES EventOFPPortStatus
BRICK switches
PROVIDES EventPortDelete TO {'topology': set()}
PROVIDES EventSwitchLeave TO {'topology': set()}
PROVIDES EventPortModify TO {'topology': set()}
PROVIDES EventSwitchReconnected TO {'topology': set()}
PROVIDES EventLinkDelete TO {'topology': set()}
PROVIDES EventHostAdd TO {'topology': set()}

```

(二) 启动mininet

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

```
-----show_topology
-----switch network-----
s1      s2      s3      s4
s1      0      (2,0.001831889153)  0      0
s2      (2,0.000893712044)  0      (3,0.000322699547)  0
s3      0      (2,0.000332951546)  0      (3,0.000593900681)
s4      0      0      (2,0.000980854034)  0
-----host 2 switch-----
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->delay EventOFPPacketIn
EVENT ofp_event->topology EventOFPPacketIn
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->delay EventOFPPacketIn
EVENT ofp_event->topology EventOFPPacketIn
-----_detector-----
EVENT ofp_event->delay EventOFPEchoReply
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->delay EventOFPPacketIn
EVENT ofp_event->topology EventOFPPacketIn
EVENT ofp_event->delay EventOFPEchoReply
EVENT ofp_event->delay EventOFPEchoReply
EVENT ofp_event->delay EventOFPEchoReply
-----get_link_delay-----
-----show echo delay-----
s1----0.001555204391
s2----0.000720500946
s3----0.001564741135
s4----0.001928567886
-----show LLDP delay-----
3-3-4----0.003864049911
2-2-1----0.002599477768
1-2-2----0.003213644028
4-2-3----0.003358840942
2-3-3----0.001875400543
3-2-2----0.004353761673
```

