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

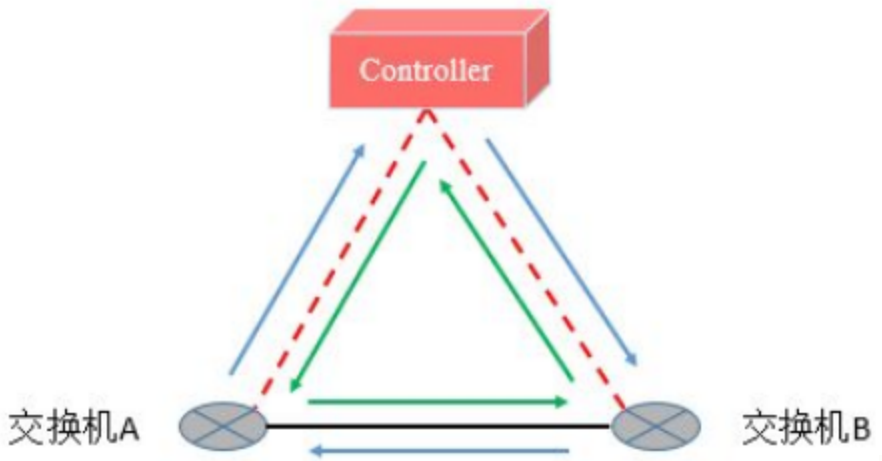
Ryu源码之拓扑发现原理分析

二：实验原理

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数据发送时的时间戳，然后和收到的时间戳进行相减，得到了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）

Python | 复制代码

```python
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER,CONFIG_DISPATCHER,DEAD_DISPATCHER #只是表示datapath数据路径的状态
from ryu.controller.handler import set_ev_cls

from ryu.lib import hub
from ryu.lib.packet import packet,ethernet

from ryu.topology import event,switches
from ryu.topology.api import get_switch,get_link,get_host

import threading,time,random

DELAY_MONITOR_PERIOD = 5

class TopoDetect(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(TopoDetect,self).__init__(*args,**kwargs)
        self.topology_api_app = self
        self.name = "topology"
        self.link_list = None
        self.switch_list = None
        self.host_list = None

        self.dpid2id = {}
        self.id2dpid = {}
        self.dpid2switch = {}

        self.ip2host = {}
        self.ip2switch = {}

        self.net_size = 0
        self.net_topo = []

        self.net_flag = False
        self.net_arrived = 0

        self.monitor_thread = hub.spawn(self._monitor)


    def _monitor(self):    #修改，只获取拓扑，不主动显示！！！
        """
        协程实现伪并发，探测拓扑状态
        """
        while True:
```

```
            #print("-----------------_monitor")
            self._host_add_handler(None) #主机单独提取处理
            self.get_topology(None)
            hub.sleep(DELAY_MONITOR_PERIOD) #5秒一次


    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,CONFIG_DISPATCHER)
    def switch_feature_handle(self,ev):
        """
        datapath中有配置消息到达
        """
        #print("------XXXXXXXXXX------%d------XXXXXXXXXX------------switch_feature_handle"%self.net_arrived)
        #print("----%s----------",ev.msg)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        match = ofp_parser.OFPMatch()

        actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]

        self.add_flow(datapath=datapath,priority=0,match=match,actions=actions,extra_info="config infomation arrived!!")


    def add_flow(self,datapath,priority,match,actions,idle_timeout=0,hard_timeout=0,extra_info=None):
        #print("-----------------add_flow:")
        if extra_info != None:
            print(extra_info)
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]

        mod = ofp_parser.OFPFlowMod(datapath=datapath,priority=priority,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    match=match,instructions=inst)
        datapath.send_msg(mod);

    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev):
        #print("-----------------packet_in_handler")
```

```
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        dpid = datapath.id
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        dst = eth_pkt.dst
        src = eth_pkt.src

        #self.logger.info("-----------------Controller %s get packet, Mac address from: %s send to: %s , send from datapath: %s,in port is: %s"
        #                  ,dpid,src,dst,dpid,in_port)
        self.get_topology(None)


    @set_ev_cls([event.EventHostAdd])
    def _host_add_handler(self,ev):    #主机信息单独处理，不属于网络拓扑
        self.host_list = get_host(self.topology_api_app) #3.需要使用pingall,主机通过与边缘交换机连接，才能告诉控制器
        #获取主机信息字典ip2host{ipv4:host object}  ip2switch{ipv4:dpid}
        for i,host in enumerate(self.host_list):
            self.ip2switch["%s"%host.ipv4] = host.port.dpid
            self.ip2host["%s"%host.ipv4] = host


    events = [event.EventSwitchEnter, event.EventSwitchLeave,
              event.EventSwitchReconnected,
              event.EventPortAdd, event.EventPortDelete,
              event.EventPortModify,
              event.EventLinkAdd, event.EventLinkDelete]
    @set_ev_cls(events)
    def get_topology(self,ev):
        #print("------++++++++++++------%d------++++++++++-----------get_topology"%self.net_arrived)

        self.net_flag = False
        self.net_topo = []

        #print("-----------------get_topology")
        #获取所有的交换机、链路
        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
        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             #获取交换机字典id2dpid{id:dpid} dpid2switch{dpid:switch object}
138             for i,switch in enumerate(self.switch_list):
139                 self.id2dpid[i] = switch.dp.id
140                 self.dpid2id[switch.dp.id] = i
141                 self.dpid2switch[switch.dp.id] = switch
142
143
144             #根据链路信息，开始获取拓扑信息
145             self.net_size = len(self.id2dpid) #表示网络中交换机个数
146             for i in range(self.net_size):
147                 self.net_topo.append([0]*self.net_size)
148
149             for link in self.link_list:
150                 src_dpid = link.src.dpid
151                 src_port = link.src.port_no
152
153                 dst_dpid = link.dst.dpid
154                 dst_port = link.dst.port_no
155
156                 try:
157                     sid = self.dpid2id[src_dpid]
158                     did = self.dpid2id[dst_dpid]
159                 except KeyError as e:
160                     #print("--------------Error:get KeyError with link infoma
    tion(%s)"%e)
161                     return
162                 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
163                 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
164
165
166             self.net_flag = True #表示网络拓扑创建成功
167
168         def show_topology(self):
169             print("-----------------show_topology")
170             print("----------switch network----------")
171             line_info = "         "
172             for i in range(self.net_size):
173                 line_info+="        s%-5d        "%self.id2dpid[i]
174             print(line_info)
175             for i in range(self.net_size):
176                 line_info = "s%d      "%self.id2dpid[i]
177                 for j in range(self.net_size):
178                     if self.net_topo[i][j] == 0:
179                         line_info+="%-22d"%0
180                     else:
```

```
                    line_info+="(%d,%.12f)    "%tuple(self.net_topo[i][j]
)
            print(line_info)

        print("----------host 2 switch----------")
        for key,val in self.ip2switch.items():
            print("%s---s%d"%(key,val))
```

（二）模块导入

Python | 复制代码

```python
from ryu.base import app_manager
from ryu.base.app_manager import lookup_service_brick

from ryu.ofproto import ofproto_v1_3

from ryu.controller import ofp_event
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
from ryu.controller.handler import set_ev_cls

from ryu.lib import hub
from ryu.lib.packet import packet,ethernet

from ryu.topology.switches import Switches
from ryu.topology.switches import LLDPPacket

import time
```

（三）数据结构

Python | 复制代码

```python
ECHO_REQUEST_INTERVAL = 0.05
DELAY_DETECTING_PERIOD = 5

class DelayDetect(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(DelayDetect,self).__init__(*args,**kwargs)
        self.name = "delay"

        self.topology = lookup_service_brick("topology") #注意：我们使用lookup_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。
        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 ./TopoDetect.py ./DelayDetect.py --verbose --observe-links

        self.dpid2switch = {} #或者直接为{}，也可以。下面_state_change_handler也会添加进去
        self.dpid2echoDelay = {} #记录echo时延

        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}

        self.detector_thread = hub.spawn(self._detector)
```

（四）协程获取链路时延

Python | 复制代码

```python
    def _detector(self):
        """
        协程实现伪并发，探测链路时延
        """
        while True:
            if self.topology == None:
                self.topology = lookup_service_brick("topology")
            if self.topology.net_flag:
                #print("------------------_detector------------------")
                self._send_echo_request()
                self.get_link_delay()
                if self.topology.net_flag:
                    try:
                        self.show_delay()
                        self.topology.show_topology()    #拓扑显示
                    except Exception as err:
                        print("------------------Detect delay failure!!!----------------")
            hub.sleep(DELAY_DETECTING_PERIOD) #5秒一次
```

（五）获取Echo时延

Python | 复制代码

```python
    def _send_echo_request(self):
        """
        发生echo报文到datapath
        """
        for datapath in self.dpid2switch.values():
            parser = datapath.ofproto_parser
            echo_req = parser.OFPEchoRequest(datapath,data=bytes("%.12f"%time.time(),encoding="utf8")) #获取当前时间

            datapath.send_msg(echo_req)

            #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。
            #在echo_reply_处理程序中处理echo reply时，会产生大量队列等待延迟。
            hub.sleep(ECHO_REQUEST_INTERVAL)

    @set_ev_cls(ofp_event.EventOFPEchoReply,[MAIN_DISPATCHER,CONFIG_DISPATCHER,HANDSHAKE_DISPATCHER])
    def echo_reply_handler(self,ev):
        """
        处理echo响应报文，获取控制器到交换机的链路往返时延

              Controller
                  |
       echo latency |
                  `|'
                   Switch
        """
        now_timestamp = time.time()
        try:
            echo_delay = now_timestamp - eval(ev.msg.data)
            self.dpid2echoDelay[ev.msg.datapath.id] = echo_delay
        except:
            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。

Python 复制代码

```python
    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev): #处理到达的LLDP报文，从而获得LLDP时延
        """
                        Controller
                       |        /|\
                      \|/         |
                    Switch----->Switch
        """
        msg = ev.msg
        try:
            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)
            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
            dst_inport = msg.match['in_port']
            if self.switches is None:
                self.switches = lookup_service_brick("switches") #获取交换机模块实例

            #获得key（Port类实例）和data（PortData类实例）
            for port in self.switches.ports.keys(): #开始获取对应交换机端口的发送时间戳
                if src_dpid == port.dpid and src_outport == port.port_no: #匹配key
                    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
                    timestamp = port_data.timestamp
                    if timestamp:
                        delay = time.time() - timestamp
                        self._save_delay_data(src=src_dpid,dst=dst_dpid,src_port=src_outport,lldpdealy=delay)
        except:
            return

    def _save_delay_data(self,src,dst,src_port,lldpdealy):
        key = "%s-%s-%s"%(src,src_port,dst)
        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

Python | 复制代码

```python
    def get_link_delay(self):
        """
        更新图中的权值信息
        """
        print("--------------get_link_delay-----------")
        for src_sport_dst in self.src_sport_dst2Delay.keys():
                src,sport,dst = tuple(map(eval,src_sport_dst.split("-")))
                if src in self.dpid2echoDelay.keys() and dst in self.dpid2echoDelay.keys():
                    sid,did = self.topology.dpid2id[src],self.topology.dpid2id[dst]
                    if self.topology.net_topo[sid][did] != 0:
                        if self.topology.net_topo[sid][did][0] == sport:
                            s_d_delay = self.src_sport_dst2Delay[src_sport_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDelay[dst])/2;
                            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
                                continue
                            self.topology.net_topo[sid][did][1] = self.src_sport_dst2Delay[src_sport_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDelay[dst])/2
```

（八）显示网络拓扑图和Echo、LLDP时延信息

```python
    @set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if not datapath.id in self.dpid2switch:
                self.logger.debug('Register datapath: %016x', datapath.id)
                self.dpid2switch[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.dpid2switch:
                self.logger.debug('Unregister datapath: %016x', datapath.id)
                del self.dpid2switch[datapath.id]

        if self.topology == None:
            self.topology = lookup_service_brick("topology")
        print("----------------------_state_change_handler----------------")
        print(self.topology.show_topology())
        print(self.switches)

    def show_delay(self):
        print("------------------------show echo delay----------------------")
        for key,val in self.dpid2echoDelay.items():
            print("s%d----%.12f"%(key,val))
        print("------------------------show LLDP delay----------------------")
        for key,val in self.src_sport_dst2Delay.items():
            print("%s----%.12f"%(key,val))
```

（九）全部代码

Python | 复制代码

```python
from ryu.base import app_manager
from ryu.base.app_manager import lookup_service_brick

from ryu.ofproto import ofproto_v1_3

from ryu.controller import ofp_event
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
from ryu.controller.handler import set_ev_cls

from ryu.lib import hub
from ryu.lib.packet import packet,ethernet

from ryu.topology.switches import Switches
from ryu.topology.switches import LLDPPacket

import time

ECHO_REQUEST_INTERVAL = 0.05
DELAY_DETECTING_PERIOD = 5

class DelayDetect(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(DelayDetect,self).__init__(*args,**kwargs)
        self.name = "delay"

        self.topology = lookup_service_brick("topology") #注意：我们使用lookup_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。
        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 ./TopoDetect.py ./DelayDetect.py --verbose --observe-links

        self.dpid2switch = {} #或者直接为{}，也可以。下面_state_change_handler也会添加进去
        self.dpid2echoDelay = {}

        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}

        self.detector_thread = hub.spawn(self._detector)

    def _detector(self):
```

```
        """
        协程实现伪并发，探测链路时延
        """
        while True:
            if self.topology == None:
                self.topology = lookup_service_brick("topology")
            if self.topology.net_flag:
                #print("------------------_detector------------------")
                self._send_echo_request()
                self.get_link_delay()
                if self.topology.net_flag:
                    try:
                        self.show_delay()
                        self.topology.show_topology()
                    except Exception as err:
                        print("------------------Detect delay failure!!!-
------------------")
            hub.sleep(DELAY_DETECTING_PERIOD) #5秒一次

    def get_link_delay(self):
        """
        更新图中的权值信息
        """
        #print("--------------get_link_delay-----------")
        for src_sport_dst in self.src_sport_dst2Delay.keys():
                src,sport,dst = tuple(map(eval,src_sport_dst.split("-")))
                if src in self.dpid2echoDelay.keys() and dst in self.dpid
2echoDelay.keys():
                    sid,did = self.topology.dpid2id[src],self.topology.dp
id2id[dst]
                    if self.topology.net_topo[sid][did] != 0:
                        if self.topology.net_topo[sid][did][0] == sport:
                            s_d_delay = self.src_sport_dst2Delay[src_spor
t_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDelay[dst])/2;
                            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
                                continue
                            self.topology.net_topo[sid][did][1] = self.sr
c_sport_dst2Delay[src_sport_dst]-(self.dpid2echoDelay[src]+self.dpid2echo
Delay[dst])/2

    def _send_echo_request(self):
        """
        发生echo报文到datapath
        """
        #print("==========_send_echo_request==============")
        #print(self.dpid2switch)
        for datapath in self.dpid2switch.values():
```

```
            parser = datapath.ofproto_parser
            echo_req = parser.OFPEchoRequest(datapath,data=bytes("%.12f"%time.time(),encoding="utf8")) #获取当前时间
            #print("==========_send_echo_request=========2=====")
            datapath.send_msg(echo_req)

            #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。
            #在echo_reply_处理程序中处理echo reply时，会产生大量队列等待延迟。
            hub.sleep(ECHO_REQUEST_INTERVAL)

    @set_ev_cls(ofp_event.EventOFPEchoReply,[MAIN_DISPATCHER,CONFIG_DISPATCHER,HANDSHAKE_DISPATCHER])
    def echo_reply_handler(self,ev):
        """
        处理echo响应报文，获取控制器到交换机的链路往返时延

              Controller
                  |
     echo latency |
                 `|'
                  Switch
        """
        #print("================================")
        #print(ev)
        #print("================================")
        now_timestamp = time.time()
        try:
            echo_delay = now_timestamp - eval(ev.msg.data)
            self.dpid2echoDelay[ev.msg.datapath.id] = echo_delay
        except:
            return


    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev): #处理到达的LLDP报文，从而获得LLDP时延
        """
                      Controller
                    |        /|\
                   \|/         |
                 Switch----->Switch
        """
        msg = ev.msg
        try:
            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)
            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
            dst_inport = msg.match['in_port']
```

```
            if self.switches is None:
                self.switches = lookup_service_brick("switches") #获取交换机模块实例

            #获得key（Port类实例）和data（PortData类实例）
            for port in self.switches.ports.keys(): #开始获取对应交换机端口的发送时间戳
                if src_dpid == port.dpid and src_outport == port.port_no: #匹配key
                    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
                    timestamp = port_data.timestamp
                    if timestamp:
                        delay = time.time() - timestamp
                        self._save_delay_data(src=src_dpid,dst=dst_dpid,src_port=src_outport,lldpdealy=delay)
        except:
            return

    def _save_delay_data(self,src,dst,src_port,lldpdealy):
        key = "%s-%s-%s"%(src,src_port,dst)
        self.src_sport_dst2Delay[key] = lldpdealy

    @set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if not datapath.id in self.dpid2switch:
                self.logger.debug('Register datapath: %016x', datapath.id)
                self.dpid2switch[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.dpid2switch:
                self.logger.debug('Unregister datapath: %016x', datapath.id)
                del self.dpid2switch[datapath.id]

        if self.topology == None:
            self.topology = lookup_service_brick("topology")
        #print("----------------------_state_change_handler----------")
        #print(self.topology.show_topology())
        #print(self.switches)

    def show_delay(self):
        #print("----------------------show echo delay----------------------")
```

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

Python | 复制代码

```python
from ryu.controller.handler import MAIN_DISPATCHER,CONFIG_DISPATCHER,DEAD_DISPATCHER #只是表示datapath数据路径的状态
from ryu.controller.handler import set_ev_cls

from ryu.lib import hub
from ryu.lib.packet import packet,ethernet

from ryu.topology import event,switches
from ryu.topology.api import get_switch,get_link,get_host

import threading,time,random

DELAY_MONITOR_PERIOD = 5

class TopoDetect(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(TopoDetect,self).__init__(*args,**kwargs)
        self.topology_api_app = self
        self.name = "topology"
        self.link_list = None
        self.switch_list = None
        self.host_list = None

        self.dpid2id = {}
        self.id2dpid = {}
        self.dpid2switch = {}

        self.ip2host = {}
        self.ip2switch = {}

        self.net_size = 0
        self.net_topo = []

        self.net_flag = False
        self.net_arrived = 0

        self.monitor_thread = hub.spawn(self._monitor)


    def _monitor(self):
        """
        协程实现伪并发，探测拓扑状态
        """
        while True:
```

```
            #print("-----------------_monitor")
            self._host_add_handler(None) #主机单独提取处理
            self.get_topology(None)
            hub.sleep(DELAY_MONITOR_PERIOD) #5秒一次


    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,CONFIG_DISPATCHER)
    def switch_feature_handle(self,ev):
        """
        datapath中有配置消息到达
        """
        #print("------XXXXXXXXXX------%d------XXXXXXXXXX------------switch_feature_handle"%self.net_arrived)
        #print("----%s----------",ev.msg)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        match = ofp_parser.OFPMatch()

        actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]

        self.add_flow(datapath=datapath,priority=0,match=match,actions=actions,extra_info="config infomation arrived!!")


    def add_flow(self,datapath,priority,match,actions,idle_timeout=0,hard_timeout=0,extra_info=None):
        #print("-----------------add_flow:")
        if extra_info != None:
            print(extra_info)
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]

        mod = ofp_parser.OFPFlowMod(datapath=datapath,priority=priority,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    match=match,instructions=inst)
        datapath.send_msg(mod);

    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev):
        #print("-----------------packet_in_handler")
```

```
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        dpid = datapath.id
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        dst = eth_pkt.dst
        src = eth_pkt.src

        #self.logger.info("-----------------Controller %s get packet, Mac address from: %s send to: %s , send from datapath: %s,in port is: %s"
        #                  ,dpid,src,dst,dpid,in_port)
        self.get_topology(None)


    @set_ev_cls([event.EventHostAdd])
    def _host_add_handler(self,ev):    #主机信息单独处理，不属于网络拓扑
        self.host_list = get_host(self.topology_api_app) #3.需要使用pingall,主机通过与边缘交换机连接，才能告诉控制器
        #获取主机信息字典ip2host{ipv4:host object}  ip2switch{ipv4:dpid}
        for i,host in enumerate(self.host_list):
            self.ip2switch["%s"%host.ipv4] = host.port.dpid
            self.ip2host["%s"%host.ipv4] = host


    events = [event.EventSwitchEnter, event.EventSwitchLeave,
              event.EventSwitchReconnected,
              event.EventPortAdd, event.EventPortDelete,
              event.EventPortModify,
              event.EventLinkAdd, event.EventLinkDelete]
    @set_ev_cls(events)
    def get_topology(self,ev):
        #print("------++++++++++++------%d------++++++++++-----------get_topology"%self.net_arrived)

        self.net_flag = False
        self.net_topo = []

        #print("-----------------get_topology")
        #获取所有的交换机、链路
        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
        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         #获取交换机字典id2dpid{id:dpid} dpid2switch{dpid:switch object}
138         for i,switch in enumerate(self.switch_list):
139             self.id2dpid[i] = switch.dp.id
140             self.dpid2id[switch.dp.id] = i
141             self.dpid2switch[switch.dp.id] = switch
142
143
144         #根据链路信息，开始获取拓扑信息
145         self.net_size = len(self.id2dpid) #表示网络中交换机个数
146         for i in range(self.net_size):
147             self.net_topo.append([0]*self.net_size)
148
149         for link in self.link_list:
150             src_dpid = link.src.dpid
151             src_port = link.src.port_no
152
153             dst_dpid = link.dst.dpid
154             dst_port = link.dst.port_no
155
156             try:
157                 sid = self.dpid2id[src_dpid]
158                 did = self.dpid2id[dst_dpid]
159             except KeyError as e:
160                 #print("--------------Error:get KeyError with link infoma
    tion(%s)"%e)
161                 return
162             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
163             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
164
165
166         self.net_flag = True #表示网络拓扑创建成功
167
168     def show_topology(self):
169         print("-----------------show_topology")
170         print("----------switch network----------")
171         line_info = "         "
172         for i in range(self.net_size):
173             line_info+="        s%-5d        "%self.id2dpid[i]
174         print(line_info)
175         for i in range(self.net_size):
176             line_info = "s%d      "%self.id2dpid[i]
177             for j in range(self.net_size):
178                 if self.net_topo[i][j] == 0:
179                     line_info+="%-22d"%0
180                 else:
```

```
                    line_info+="(%d,%.12f)    "%tuple(self.net_topo[i][j]
)
            print(line_info)

        print("----------host 2 switch----------")
        for key,val in self.ip2switch.items():
            print("%s---s%d"%(key,val))
```

Python | 复制代码

```python
from ryu.base import app_manager
from ryu.base.app_manager import lookup_service_brick

from ryu.ofproto import ofproto_v1_3

from ryu.controller import ofp_event
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
from ryu.controller.handler import set_ev_cls

from ryu.lib import hub
from ryu.lib.packet import packet,ethernet

from ryu.topology.switches import Switches
from ryu.topology.switches import LLDPPacket

import time

ECHO_REQUEST_INTERVAL = 0.05
DELAY_DETECTING_PERIOD = 5

class DelayDetect(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(DelayDetect,self).__init__(*args,**kwargs)
        self.name = "delay"

        self.topology = lookup_service_brick("topology") #注意：我们使用lookup_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。
        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 ./TopoDetect.py ./DelayDetect.py --verbose --observe-links

        self.dpid2switch = {} #或者直接为{}，也可以。下面_state_change_handler也会添加进去
        self.dpid2echoDelay = {}

        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}

        self.detector_thread = hub.spawn(self._detector)

    def _detector(self):
```

```
        """
        协程实现伪并发，探测链路时延
        """
        while True:
            if self.topology == None:
                self.topology = lookup_service_brick("topology")
            if self.topology.net_flag:
                #print("------------------_detector------------------")
                self._send_echo_request()
                self.get_link_delay()
                if self.topology.net_flag:
                    try:
                        self.show_delay()
                        self.topology.show_topology()
                    except Exception as err:
                        print("------------------Detect delay failure!!!-
------------------")
            hub.sleep(DELAY_DETECTING_PERIOD) #5秒一次

    def get_link_delay(self):
        """
        更新图中的权值信息
        """
        #print("--------------get_link_delay-----------")
        for src_sport_dst in self.src_sport_dst2Delay.keys():
                src,sport,dst = tuple(map(eval,src_sport_dst.split("-")))
                if src in self.dpid2echoDelay.keys() and dst in self.dpid
2echoDelay.keys():
                    sid,did = self.topology.dpid2id[src],self.topology.dp
id2id[dst]
                    if self.topology.net_topo[sid][did] != 0:
                        if self.topology.net_topo[sid][did][0] == sport:
                            s_d_delay = self.src_sport_dst2Delay[src_spor
t_dst]-(self.dpid2echoDelay[src]+self.dpid2echoDelay[dst])/2;
                            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
                                continue
                            self.topology.net_topo[sid][did][1] = self.sr
c_sport_dst2Delay[src_sport_dst]-(self.dpid2echoDelay[src]+self.dpid2echo
Delay[dst])/2

    def _send_echo_request(self):
        """
        发生echo报文到datapath
        """
        #print("==========_send_echo_request==============")
        #print(self.dpid2switch)
        for datapath in self.dpid2switch.values():
```

```
            parser = datapath.ofproto_parser
            echo_req = parser.OFPEchoRequest(datapath,data=bytes("%.12f"%time.time(),encoding="utf8")) #获取当前时间
            #print("==========_send_echo_request=========2=====")
            datapath.send_msg(echo_req)

            #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。
            #在echo_reply_处理程序中处理echo reply时，会产生大量队列等待延迟。
            hub.sleep(ECHO_REQUEST_INTERVAL)

    @set_ev_cls(ofp_event.EventOFPEchoReply,[MAIN_DISPATCHER,CONFIG_DISPATCHER,HANDSHAKE_DISPATCHER])
    def echo_reply_handler(self,ev):
        """
        处理echo响应报文，获取控制器到交换机的链路往返时延

              Controller
                  |
     echo latency |
                 `|'
                  Switch
        """
        #print("================================")
        #print(ev)
        #print("================================")
        now_timestamp = time.time()
        try:
            echo_delay = now_timestamp - eval(ev.msg.data)
            self.dpid2echoDelay[ev.msg.datapath.id] = echo_delay
        except:
            return


    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev): #处理到达的LLDP报文，从而获得LLDP时延
        """
                      Controller
                     |        /|\
                    \|/         |
                  Switch----->Switch
        """
        msg = ev.msg
        try:
            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)
            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
            dst_inport = msg.match['in_port']
```

```
            if self.switches is None:
                self.switches = lookup_service_brick("switches") #获取交换机模块实例

            #获得key（Port类实例）和data（PortData类实例）
            for port in self.switches.ports.keys(): #开始获取对应交换机端口的发送时间戳
                if src_dpid == port.dpid and src_outport == port.port_no: #匹配key
                    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
                    timestamp = port_data.timestamp
                    if timestamp:
                        delay = time.time() - timestamp
                        self._save_delay_data(src=src_dpid,dst=dst_dpid,src_port=src_outport,lldpdealy=delay)
        except:
            return

    def _save_delay_data(self,src,dst,src_port,lldpdealy):
        key = "%s-%s-%s"%(src,src_port,dst)
        self.src_sport_dst2Delay[key] = lldpdealy

    @set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if not datapath.id in self.dpid2switch:
                self.logger.debug('Register datapath: %016x', datapath.id)
                self.dpid2switch[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.dpid2switch:
                self.logger.debug('Unregister datapath: %016x', datapath.id)
                del self.dpid2switch[datapath.id]

        if self.topology == None:
            self.topology = lookup_service_brick("topology")
        #print("----------------------_state_change_handler----------------------")
        #print(self.topology.show_topology())
        #print(self.switches)

    def show_delay(self):
        #print("----------------------show echo delay----------------------")
```

```
        for key,val in self.dpid2echoDelay.items():
            print("s%d----%.12f"%(key,val))
        #print("----------------------show LLDP delay-------------------
----")
        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 EventOFPStateChange
  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 EventOFPStateChange 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()}
```

（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
```

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

```
-----------------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
```