COMP550705 软件定义网络
实验报告

第 3 次

姓名

班级

学号

日期

一、 实验目的

- 用 ryu.topology.api 发现网络拓扑。
- 学习利用 LLDP 和 Echo 数据包测量链路时延。
- 学习计算基于跳数和基于时延的最短路由。
- 学习设计能够容忍链路故障的路由策略。
- 分析网络集中式控制与分布式控制的差异，思考 SDN 的得与失。

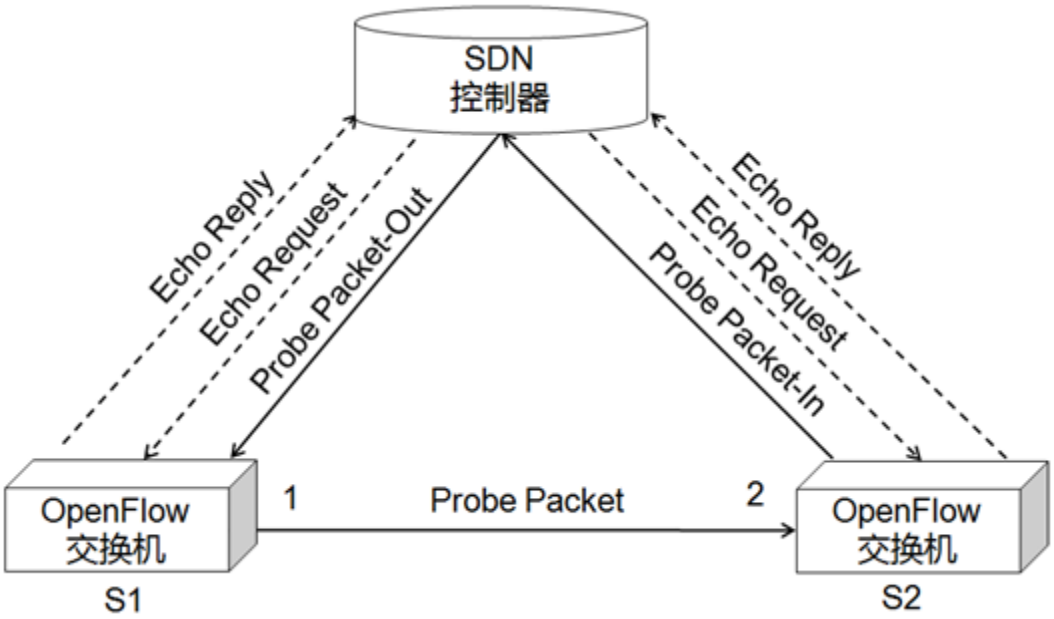
二、实验任务

（一） 必做题：最小时延路径

跳数最少的路由不一定是最快的路由，链路时延也会对路由的快慢产生重要影响。请实时地（周期地）利用 LLDP 和 Echo 数据包测量各链路的时延，在网络拓扑的基础上构建一个有权图，然后基于此图计算最小时延路径。具体任务是，找出一条从 SDC 到 MIT 时延最短的路径，输出经过的路线及总的时延，利用 Ping 包的 RTT 验证你的结果。

测量原理：链路时延

测量链路时延的思路可参考下图



控制器将带有时间戳的 LLDP 报文下发给 S1 ， S1 转发给 S2 ， S2 上传回控制器，根据收到的时间和发送时间即可计算出控制器经 S1 到 S2 再返回控制器的时延，记为 lldp_delay_s12，反之，控制器经 S2 到 S1 再返回控制器的时延，记为 lldp_delay_s21 交换机收到控制器发来的 Echo 报文后会立即回复控制器，我们可以利用 Echo Request/Reply 报文求出控制器到 S1 、S2 的往返时延，记为 echo_delay_s1 ，echo_delay_s2 则 S1 到 S2 的时延 delay = (lldp_delay_s12 + lldp_delay_s21 - echo_delay_s1 -echo_delay_s2) / 2。

（二） 选做题：容忍链路故障

1970 年的网络硬件发展尚不成熟，通信链路和交换机端口发生故障的概率较高。

请设计 Ryu app ，在任务一的基础上实现容忍链路故障的路由选择：每当链路出现故障时，重新选择当前可用路径中时延最低的路径；当链路故障恢复后，也重新选择新的时延最低的路径。请在实验报告里附上你计算的（1）最小时延路径（2）最小时延路径的 RTT （3）链路故障/恢复后发生的路由转移。

三、实验内容与分析

3.1 最小时延路径

为实现目标，需要对 Ryu 做如下修改：

1. ryu/topology/Switches.py 的 PortData/__init__()

PortData 记录交换机的端口信息，我们需要增加 self.delay 属性记录上述的 lldp_delay

self.timestamp 为 LLDP 包在发送时被打上的时间戳，具体发送的逻辑查看源码

```
class PortData(object):
def __init__(self, is_down, lldp_data):
super(PortData, self).__init__()
self.is_down = is_down
self.lldp_data = lldp_data
self.timestamp = None
self.sent = 0
self.delay = 0
```

2. ryu/topology/Switches/lldp_packet_in_handler()

lldp_packet_in_handler() 处理接收到的 LLDP 包，在这里用收到 LLDP 报文的时间戳减去发送时的时间戳即为 lldp_delay ，由于 LLDP 报文被设计为经一跳后转给控制器，我们可将 lldp_delay 存入发送 LLDP 包对应的交换机端口

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def lldp_packet_in_handler(self, ev):
# add receive timestamp
recv_timestamp = time.time()
if not self.link_discovery:
return
msg = ev.msg
try:
src_dpid, src_port_no = LLDPPacket.lldp_parse(msg.data)
except LLDPPacket.LLDPUnknownFormat:
# This handler can receive all the packets which can be
# not-LLDP packet. Ignore it silently
return
# calc the delay of lldp packet
for port, port_data in self.ports.items():
if src_dpid == port.dpid and src_port_no == port.port_no:
send_timestamp = port_data.timestamp
if send_timestamp:
port_data.delay = recv_timestamp - send_timestamp ...
```

完成上述修改后需重新编译安装 Ryu ，在安装目录下运行

```
sudo python setup.py install
```

可以利用 lookup_service_brick 获取到正在运行的 switches 的实例（即步骤 1、2 中被我们修改的类），相关代码见附件。值得主意的是，if self.switches is None 需要改为 if not self.switches，否则程序段总是进入 exception 段。

在任务 1 中，主要进行以下工作（具体的代码见附件）：
对 network_awareness.py：
- 完善 packet_in_hander()获取到正在运行的 switches 的实例，记录 lldp_delay
- 补充了 echo_hander()和 echo_send_requests()，记录 echo_delay
- 完善了_get_topology()，发出 echo 消息测量 echo 延迟，并计算总延迟 delay

对 shortest_forward.py：
- 完善了 handle_ipv4()，计算 path delay

拓扑连接如下所示：

```
# ... and now hosts
h1 = self.addHost( 'HARVARD' )
h2 = self.addHost( 'SRI' )
h3 = self.addHost( 'UCSB' )
h4 = self.addHost( 'UCLA' )
h5 = self.addHost( 'RAND' )
h6 = self.addHost( 'SDC' )
h7 = self.addHost( 'UTAH' )
h8 = self.addHost( 'MIT' )
h9 = self.addHost( 'BBN' )
```

```
# add edges between switches
self.addLink( s1 , s9, bw=10, delay='10ms')
self.addLink( s2 , s3, bw=10, delay='11ms')
self.addLink( s2 , s4, bw=10, delay='13ms')
self.addLink( s3 , s4, bw=10, delay='14ms')
self.addLink( s4 , s5, bw=10, delay='15ms')
self.addLink( s5 , s9, bw=10, delay='29ms')
self.addLink( s5 , s6, bw=10, delay='17ms')
self.addLink( s6 , s7, bw=10, delay='10ms')
self.addLink( s7 , s8, bw=10, delay='62ms')
self.addLink( s8 , s9, bw=10, delay='17ms')
```

任务 1 实现了最小实验路径。下面给出测试：

SDC ping MIT

```
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=66.8 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=129 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=129 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=130 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=128 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=128 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=129 ms
path: 10.0.0.5 -> 10.0.0.3
10.0.0.5 -> 1:s6:2 -> 4:s5:3 -> 3:s9:4 -> 3:s8:1 -> 10.0.0.3
```

RTT 理论值 $time = (17 + 29 + 17) * 2 = 126ms$

UTAH ping MIT

```
mininet> UTAH ping MIT
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=196 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=126 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=125 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=125 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=126 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=125 ms
path: 10.0.0.9 -> 10.0.0.3
10.0.0.9 -> 1:s7:3 -> 2:s8:1 -> 10.0.0.3
delay= 63.34329ms
time=126.68657ms
```

RTT 理论值 $time = 62 * 2 = 124ms$

SRI ping UTAH

```
mininet> SRI ping UTAH
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=115 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=116 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=115 ms
64 bytes from 10.0.0.9: icmp_seq=4 ttl=64 time=113 ms
64 bytes from 10.0.0.9: icmp_seq=5 ttl=64 time=115 ms
64 bytes from 10.0.0.9: icmp_seq=6 ttl=64 time=113 ms

path: 10.0.0.6 -> 10.0.0.9
10.0.0.6 -> 1:s2:3 -> 2:s4:4 -> 2:s5:4 -> 2:s6:3 -> 2:s7:1 -> 10.0.0.9
```

RTT 理论值 $time = (13 + 15 + 17 + 10) * 2 = 110ms$

3.2 容忍链路故障

链路状态改变时，链路关联的端口状态也会变化，从而产生端口状态改变的事件，即 EventOFPPortStatus ，通过将此事件与处理函数绑定在一起，就可以获取状态改变的信息，执行相应的处理，从而实现容忍链路故障。任务需要借助 OFPFC_DELETE 消息和 Packet_In 消息实现。

（1）OFPFC_DELETE 消息

与向交换机中增加流表的 OFPFC_ADD 命令不同，OFPFC_DELETE 消息用于删除交换机中符合匹配项的所有流表。由于添加和删除都属于 OFPFlowMod 消息，因此只需稍微修改 add_flow() 函数，即可生成 delete_flow() 函数。

（2）Packet_In 消息的合理利用

基本思路是在链路发生改变时，删除受影响的链路上所有交换机上的相关流表的信息，下一次交换机将匹配默认流表项，向控制器发送 packet_in 消息，控制器重新计算并下发最小时延路径。

任务 2 主要是修改 network_awareness.py，具体而言，即补充事件处理函数 port_status_handler 和 delete_flow() 函数（具体代码见附件）。

下面给出结果示例（为显示方便，将两个终端的输出放在一张图中）：

SDC ping MIT

```
mininet> SDC ping MIT
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=71.4 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=130 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=128 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=130 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=130 ms
```

link s9 s8 down
SDC ping MIT

```
mininet> link s9 s8 down
mininet> SDC ping MIT
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=76.5 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=146 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=146 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=147 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=146 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=146 ms
path: 10.0.0.5 -> 10.0.0.3
10.0.0.5 -> 1:s6:3 -> 2:s7:3 -> 2:s8:1 -> 10.0.0.3
delay= 74.88799ms
time= 149.77598ms
```

link s9 s8 up
SDC ping MIT

```
mininet> link s9 s8 up
mininet> SDC ping MIT
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=208 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=129 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=129 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=129 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=128 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=128 ms
path: 10.0.0.5 -> 10.0.0.3
10.0.0.5 -> 1:s6:2 -> 4:s5:3 -> 3:s9:4 -> 3:s8:1 -> 10.0.0.3
delay= 69.24987ms
time= 138.49974ms
```

理论分析：SDC ping MIT 最短路径的时延$delay = 63\,ms, RTT = 126ms$。若 s9 和 s8 链路断开，此时最短路径的时延$delay = 72\,ms, RTT = 144ms$。若 s9 和 s8 链路重新恢复正常，则最短路径的时延 delay 以及 RTT 也恢复成最初的值。

在上面的示例中，开始时$RTT \approx 130\,ms$，在$s9$和$s8$链路断开后$RTT \approx 146\,ms$，

10.0.0.5−> 10.0.0.3的路径变为10.0.0.5−> $s6$−> $s7$−> $s8$−> 10.0.0.3。当链路恢复后，$RTT \approx 128\,ms$，路径重新变回10.0.0.5−> $s6$−> $s5$−> $s9$−> $s8$−> 10.0.0.3。实验结果与理论分析相近，可见实现了对链路故障的容忍。