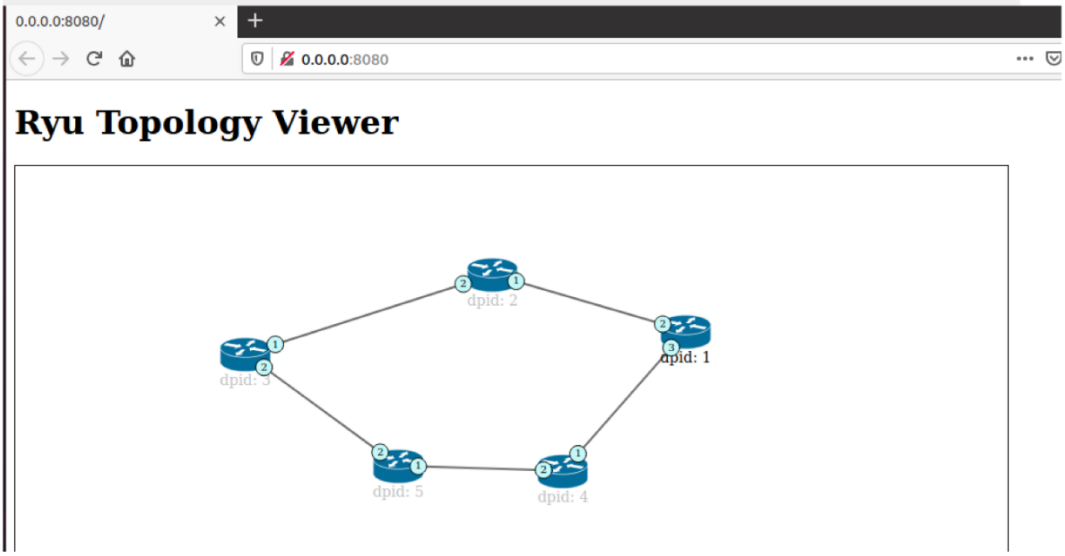
SDN 第二次实验报告

计算机86常傲2182212573

```python
from mininet.topo import Topo

class MyTopo( Topo ):
    def __init__( self ):
        "Create custom topo."
        # Initialize topology
        Topo.__init__( self )

        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')
        self.addLink(s1,s2)
        self.addLink(s2,s3)
        self.addLink(s3,s5)
        self.addLink(s1,s4)
        self.addLink(s4,s5)
        self.addLink(h1,s1)
        self.addLink(s5,h2)

topos = { 'mytopo': ( lambda: MyTopo() ) }
```



一、 动态改变转发规则

思路以及代码：

1. 由于包含两条路径： s1-s4-s5 与 s1-s2-s3-s5，可以视为最短路径和最长路径。因此，利用给出 的最短路径函数 short_path() ，修改得到最长路径函数 long_path()

```
def long_path(self, src, dst, bw=0):
        if src == dst:
            return []
        result = defaultdict(lambda: defaultdict(lambda: None))
        distance = defaultdict(lambda: None)
        # the node is checked
        seen = [src]

        # the distance to src
        distance[src] = 0

        w = 1  # weight
        while len(seen) < len(self.src_links):
            node = seen[-1]
            if node == dst:
                break
            for (temp_src, temp_dst) in self.src_links[node]:
                if temp_dst not in seen:
                    temp_src_port = self.src_links[node][(temp_src,
temp_dst)][0]
                    temp_dst_port = self.src_links[node][(temp_src,
temp_dst)][1]
                    if (distance[temp_dst] is None) or (distance[temp_dst] <
distance[temp_src] + w):
                        distance[temp_dst] = distance[temp_src] + w
                        # result = {"dpid":(link_src, src_port, link_dst,
dst_port)}
                        result[temp_dst] = (temp_src, temp_src_port,
temp_dst, temp_dst_port)
            max_node = None
            max_path = -999
            # get the max_path node
            for temp_node in distance:
                if (temp_node not in seen) and (distance[temp_node] is not
None):
                    if distance[temp_node] > max_path:
                        max_node = temp_node
                        max_path = distance[temp_node]
            if max_node is None:
                break
            seen.append(max_node)

        path = []

        if dst not in result:
            return None

        while (dst in result) and (result[dst] is not None):
            path = [result[dst][2:4]] + path
            path = [result[dst][0:2]] + path
            dst = result[dst][0]
        #self.logger.info("path : %s", str(path))
        return path
```

2. 利用 hard_timeout 参数，设置每 5 秒硬删除相关流表项

```
self.add_flow(datapath_path, 100, match, actions, idle_timeout=5,
hard_timeout=5)
```

3.同时利用全局变量 self.pathmod ，初始化为 0，在每次下发流表时，判断：

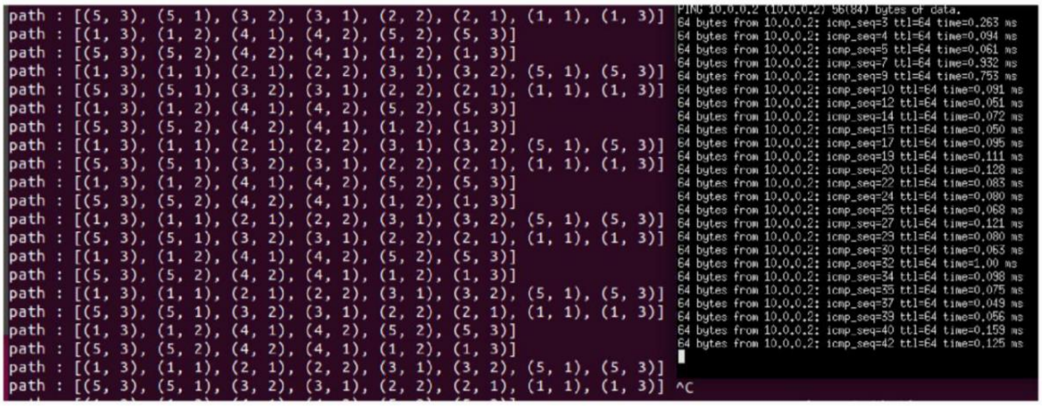
（1） 若 pathmod 为偶数，下发最短路径， pathmod +1

（2） 若 pathmod 为奇数，下发最长路径， pathmod +1

```
    def install_path(self, src_dpid, dst_dpid, src_port, dst_port, ev, src,
dst, pkt_ipv4, pkt_tcp):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        mid_path = None
        if(self.pathmod % 2 == 0) :
            mid_path = self.short_path(src=src_dpid, dst=dst_dpid)
            self.pathmod = self.pathmod + 1
        else:
            mid_path = self.long_path(src=src_dpid, dst=dst_dpid)
            self.pathmod = self.pathmod + 1

        if mid_path is None:
            return
        self.path = None
        self.path = [(src_dpid, src_port)] + mid_path + [(dst_dpid,
dst_port)]

        self.logger.info("path : %s", str(self.path))
```

```
        for i in xrange(len(self.path) - 2, -1, -2):
            datapath_path = self.datapaths[self.path[i][0]]
            match = parser.OFPMatch(in_port=self.path[i][1], eth_src=src,
eth_dst=dst, eth_type=0x0800,
                                    ipv4_src=pkt_ipv4.src,
ipv4_dst=pkt_ipv4.dst)

            if i < (len(self.path) - 2):
                actions = [parser.OFPActionOutput(self.path[i + 1][1])]
            else:
                actions =
[parser.OFPActionSetField(eth_dst=self.ip_to_mac.get(pkt_ipv4.dst)),
                           parser.OFPActionOutput(self.path[i + 1][1])]

            self.add_flow(datapath_path, 100, match, actions, idle_timeout=5,
hard_timeout=5)
```

实验结果：



二、 链路故障恢复功能

思路以及代码：

1.分析了添加流表项的函数 add_flow()，在查询、阅读 RYU 文档后得知， OFPFlowMod 的构造参数中，有一项 command，默认是 OFPFC_ADD，因此在 add_flow() 中并没有写出，而是如果要构造删除函数，要对 command 参数进行设置。

Value	Explanation
OFPFC_ADD	Adds new flow entries.
OFPFC_MODIFY	Updates flow entries.
OFPFC_MODIFY_STRICT	Update strictly matched flow entries
OFPFC_DELETE	Deletes flow entries.
OFPFC_DELETE_STRICT	Deletes strictly matched flow entries.

2.经过修改，将 command 参数改为 OFPFC_DELETE，可以得到删除流表项的函数 delete_flow()

```
    def delete_flow(self, datapath, priority, match, actions, buffer_id=None,
idle_timeout=0, hard_timeout=0):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = []
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    command=ofproto.OFPFC_DELETE,
```

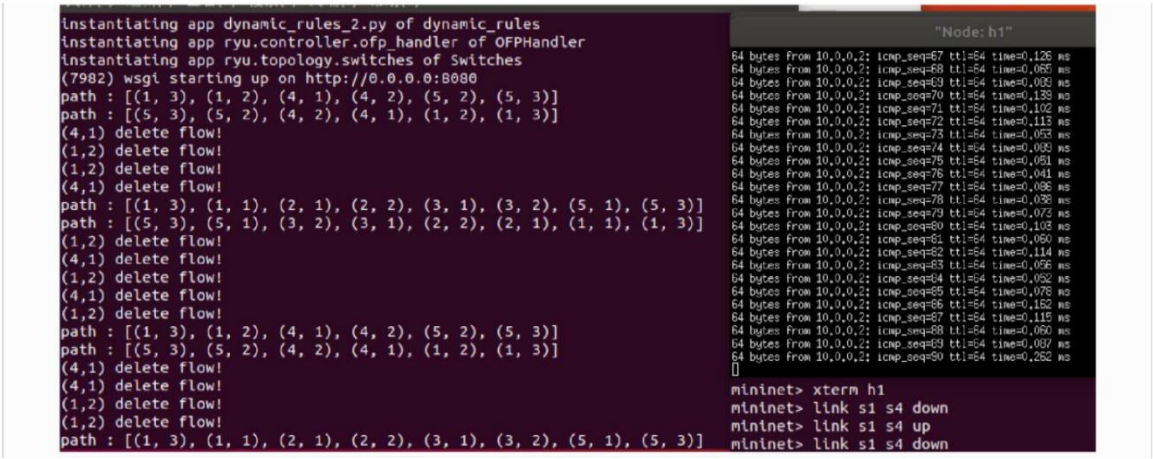
```
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    idle_timeout=idle_timeout,
                                    hard_timeout=hard_timeout,
                                    command=ofproto.OFPFC_DELETE,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)
```

3.构造 get_OFPPortStatus_msg() 函数。利用事件包 EventOFPPortStatus 中 reason 参数可以进行判断。对链路的 up 和 down 操作，会使得端口信息变化，一旦检测到 reason==ofproto.OFPPR_MODIFY，即删除流表项

```
    @set_ev_cls(ofp_event.EventOFPPortStatus, [CONFIG_DISPATCHER,
MAIN_DISPATCHER, DEAD_DISPATCHER, HANDSHAKE_DISPATCHER])
    def get_OFPPortStatus_msg(self, ev):
        msg=ev.msg
        datapath=ev.msg.datapath
        dpid = msg.datapath.id
        ofproto=datapath.ofproto
        parser=datapath.ofproto_parser
        port=msg.desc.port_no
        reason=msg.reason
        if (reason==ofproto.OFPPR_MODIFY):
            self.delete_flow(datapath=datapath,dst_port=port)
```

实验结果：

反复执行 link s1 s4 down 与 link s1 s4 up，如图：



三、 ping 的界面不停顿地执行，但会有延迟，为什么？

流表被删除后需要向控制器发送信息重新下发流表，因此会有延迟