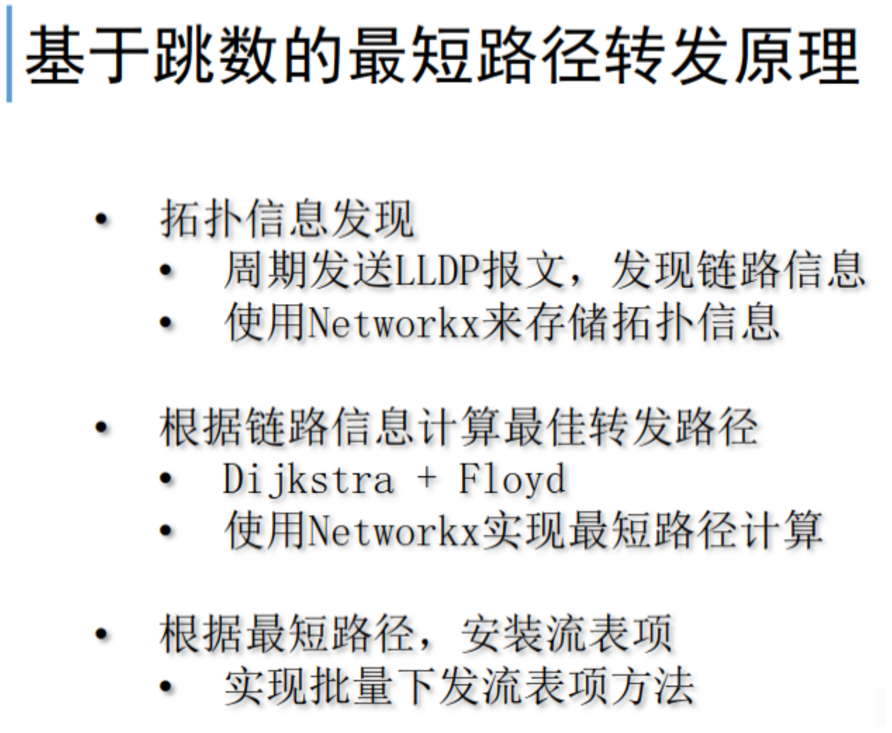
Ryu的应用开发（四）基于跳数的最短路径转发原理

一：实现最短跳数转发
　（一）原理
推文：迪杰斯特拉算法和弗洛伊德算法
二：代码实现
　（一）全部代码
　（二）注意：由于各种版本的不同，导致我们使用函数的时候可能有所不同，需要我们自己去调试
　　1.安装networkx模块，若是下载太慢或者出错，换国内源：
　　2.出现self.network.add_edge(dpid, src, {'port':in_port})使用时，会出现参数太多
　　3.学会代码调试和思考
三：代码讲解
　（一）不变代码：实现初始连接处理和公共函数--下发流表
　（二）实现获取网络拓扑结构
　（三）实现下一跳端口获取（根据图获取最短路径，从中获取信息）
　（四）实现包接收，计算最短路径，按照最短路径进行动作下发《重点》
三：实验演示：注意开启顺序，否则可能导致其他错误
　（一）启动Ryu控制器 observe-links命令会导致控制器在运行期间会不间断地发送LLDP数据包进行链路探测
　（二）启动Mininet
　（三）Ryu查看信息

一：实现最短跳数转发

掌握基于Ryu开发最短路径转发应用：基于网络信息进行复杂网络算法实现

（一）原理



基于跳数的最短路径转发原理

- 拓扑信息发现
  - 周期发送LLDP报文，发现链路信息
  - 使用Networkx来存储拓扑信息
- 根据链路信息计算最佳转发路径
  - Dijkstra + Floyd
  - 使用Networkx实现最短路径计算
- 根据最短路径，安装流表项
  - 实现批量下发流表项方法

推文：迪杰斯特拉算法和弗洛伊德算法

二：代码实现

（一）全部代码

Python 复制代码

```python
from ryu.base import app_manager
from ryu.controller.handler import set_ev_cls
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER,CONFIG_DISPATCHER
from ryu.lib.packet import packet,ethernet
from ryu.topology import event
from ryu.topology.api import get_switch,get_link
from ryu.ofproto import ofproto_v1_3

import networkx as nx

class MyShortestForwarding(app_manager.RyuApp):
    '''
    class to achive shortest path to forward, based on minimum hop count
    '''
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(MyShortestForwarding,self).__init__(*args,**kwargs)

        #set data structor for topo construction
        self.network = nx.DiGraph()        #store the dj graph
        self.paths = {}        #store the shortest path
        self.topology_api_app = self

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,CONFIG_DISPATCHER)
    def switch_features_handler(self,ev):
        '''
        manage the initial link between switch and controller
        '''
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        match = ofp_parser.OFPMatch()    #for all packet first arrive, match it successful, send it to controller
        actions  = [ofp_parser.OFPActionOutput(
                            ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER
                            )]

        self.add_flow(datapath, 0, match, actions)

    def add_flow(self,datapath,priority,match,actions):
        '''
        fulfil the function to add flow entry to switch
        '''
        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,match=match,instructions=inst)

        datapath.send_msg(mod)


    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev):
        '''
        manage the packet which comes from switch
        '''
        #first get event infomation
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

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

        #second get ethernet protocol message
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)

        eth_src = eth_pkt.src     #note: mac info willn`t  change in network
        eth_dst = eth_pkt.dst

        out_port = self.get_out_port(datapath,eth_src,eth_dst,in_port)
        actions = [ofp_parser.OFPActionOutput(out_port)]

        if out_port != ofproto.OFPP_FLOOD:
            match = ofp_parser.OFPMatch(in_port=in_port,eth_dst=eth_dst)
            self.add_flow(datapath,1,match,actions)

        out = ofp_parser.OFPPacketOut(
                datapath=datapath,buffer_id=msg.buffer_id,in_port=in_port,
                actions=actions,data=msg.data
            )

        datapath.send_msg(out)

    @set_ev_cls(event.EventSwitchEnter,[CONFIG_DISPATCHER,MAIN_DISPATCHER])    #event is not from openflow protocol, is come from switchs` state changed, just like: link to controller at the first time or send packet to controller
    def get_topology(self,ev):
        '''
        get network topo construction, save info in the dict
        '''

        #store nodes info into the Graph
        switch_list = get_switch(self.topology_api_app,None)    #------------need to get info,by debug
        switches = [switch.dp.id for switch in switch_list]
        self.network.add_nodes_from(switches)

        #store links info into the Graph
        link_list = get_link(self.topology_api_app,None)
        #port_no, in_port    ---------------need to debug, get diffirent from  both
        links = [(link.src.dpid,link.dst.dpid,{'attr_dict':{'port':link.dst.port_no}}) for link in link_list]    #add edge, need src,dst,weigtht
        self.network.add_edges_from(links)

        links  = [(link.dst.dpid,link.src.dpid,{'attr_dict':{'port':link.dst.port_no}}) for link in link_list]
        self.network.add_edges_from(links)

    def get_out_port(self,datapath,src,dst,in_port):
        '''
        datapath: is current datapath info
        src,dst: both are the host info
        in_port: is current datapath in_port
        '''
        dpid = datapath.id

        #the first :Doesn`t find src host at graph
        if src not in self.network:
            self.network.add_node(src)
            self.network.add_edge(dpid, src, attr_dict={'port':in_port})
            self.network.add_edge(src, dpid)
            self.paths.setdefault(src, {})

        #second: search the shortest path, from src to dst host
        if dst in self.network:
            if dst not in self.paths[src]:    #if not cache src to dst path,then to find it
                path = nx.shortest_path(self.network,src,dst)
                self.paths[src][dst]=path

            path = self.paths[src][dst]
            next_hop = path[path.index(dpid)+1]
            #print("1oooooooooooooooooooo")
            #print(self.network[dpid][next_hop])
            out_port = self.network[dpid][next_hop]['attr_dict']['port']
            #print("2oooooooooooooooooooo")
            #print(out_port)

            #get path info
            #print("6666666666 find dst")
            print(path)
        else:
            out_port = datapath.ofproto.OFPP_FLOOD    #By flood, to find dst, when dst get packet, dst will send a new back,the graph will record dst info
            #print("8888888888 not find dst")
        return out_port
```

（二）注意：由于各种版本的不同，导致我们使用函数的时候可能有所不同，需要我们自己去调试

1.安装networkx模块，若是下载太慢或者出错，换国内源：

在末尾加上-i https://pypi.tuna.tsinghua.edu.cn/simple

```shell
pip3 install networkx
pip3 install multiqc
```

2.出现self.network.add_edge(dpid, src, {'port':in_port})使用时，会出现参数太多

```python
self.network.add_edge(dpid, src, attr_dict={'port':in_port})    attr_dict是对我们提供的扩展成熟
```

3.学会代码调试和思考

三：代码讲解

（一）不变代码：实现初始连接处理和公共函数--下发流表

Python | 复制代码

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures,CONFIG_DISPATCHER)
    def switch_features_handler(self,ev):
        '''
        manage the initial link between switch and controller
        '''
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        match = ofp_parser.OFPMatch()    #for all packet first arrive, match it successful, send it to controller
        actions  = [ofp_parser.OFPActionOutput(
                            ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER
                            )]

        self.add_flow(datapath, 0, match, actions)

    def add_flow(self,datapath,priority,match,actions):
        '''
        fulfil the function to add flow entry to switch
        '''
        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,match=match,instructions=inst)

        datapath.send_msg(mod)
```

（二）实现获取网络拓扑结构

Python | 复制代码

```python
class MyShortestForwarding(app_manager.RyuApp):
    '''
    class to achive shortest path to forward, based on minimum hop count
    '''
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(MyShortestForwarding,self).__init__(*args,**kwargs)

        #set data structor for topo construction
        self.network = nx.DiGraph()        #store the dj graph    设置图结构存储信息
        self.paths = {}        #store the shortest path
        self.topology_api_app = self

    @set_ev_cls(event.EventSwitchEnter,[CONFIG_DISPATCHER,MAIN_DISPATCHER])    #event is not from openflow protocol, is come from switchs` state changed, just like: link to controller at the first time or send packet to controller
    def get_topology(self,ev):
        '''
        get network topo construction, save info in the dict    由于监听交换机信息进入，触发，注意事件不在flow协议类，在from ryu.topology import event中
        '''

        #store nodes info into the Graph
        switch_list = get_switch(self.topology_api_app,None)    #------------need to get info,by debug
        switches = [switch.dp.id for switch in switch_list]
        self.network.add_nodes_from(switches)

        #store links info into the Graph
        link_list = get_link(self.topology_api_app,None)
        #port_no, in_port    ---------------need to debug, get diffirent from  both
        links = [(link.src.dpid,link.dst.dpid,{'attr_dict':{'port':link.dst.port_no}}) for link in link_list]    #add edge, need src,dst,weigtht
        self.network.add_edges_from(links)

        links  = [(link.dst.dpid,link.src.dpid,{'attr_dict':{'port':link.dst.port_no}}) for link in link_list]
        self.network.add_edges_from(links)
```

补充：event.EventSwitchEnter---由于监听交换机信息进入，触发，注意事件不在flow协议类，在from ryu.topology import event中

（三）实现下一跳端口获取（根据图获取最短路径，从中获取信息）

Python 复制代码

```python
def get_out_port(self,datapath,src,dst,in_port):
        '''
        datapath: is current datapath info
        src,dst: both are the host info
        in_port: is current datapath in_port
        '''
        dpid = datapath.id

        #the first :Doesn`t find src host at graph
        if src not in self.network:       #根据src主机是否在网络中，决定是否新添加进入
            self.network.add_node(src)
            self.network.add_edge(dpid, src, attr_dict={'port':in_port})
            self.network.add_edge(src, dpid)
            self.paths.setdefault(src, {})    #设置数据结构：用于保存每个源主机到各个目的主机的最短路径{src1:{dst1:[],dst2:[],dst3:[]....},src2:{dst1:[],dst2:[],dst3:[]....},}

        #second: search the shortest path, from src to dst host
        if dst in self.network:
            if dst not in self.paths[src]:    #if not cache src to dst path,then to find it
                path = nx.shortest_path(self.network,src,dst)
                self.paths[src][dst]=path

            path = self.paths[src][dst]
            next_hop = path[path.index(dpid)+1]    #根据数据结构获取下一跳datapath信息
            #print("1ooooooooooooooooooo")
            #print(self.network[dpid][next_hop])
            out_port = self.network[dpid][next_hop]['attr_dict']['port']  #根据该id和下一跳id去获取出端口，进行数据转发
            #print("2ooooooooooooooooooo")
            #print(out_port)

            #get path info
            #print("6666666666 find dst")
            print(path)
        else:    #否则是泛洪处理
            out_port = datapath.ofproto.OFPP_FLOOD    #By flood, to find dst, when dst get packet, dst will send a new back,the graph will record dst info
            #print("8888888888 not find dst")
        return out_port
```

（四）实现包接收，计算最短路径，按照最短路径进行动作下发《重点》

Python | 复制代码

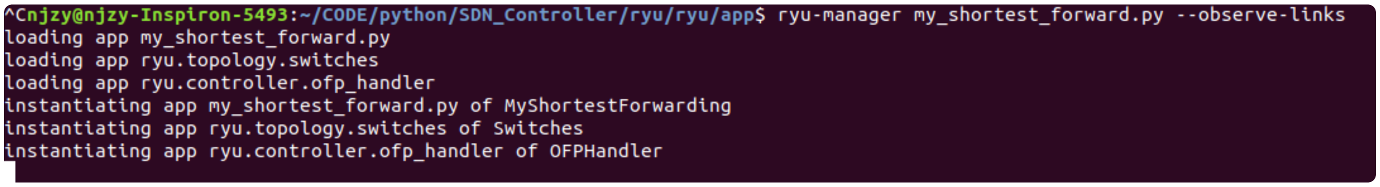
```python
@set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,ev):
        '''
        manage the packet which comes from switch
        '''
        #first get event infomation
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

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

        #second get ethernet protocol message
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)

        eth_src = eth_pkt.src     #note: mac info willn`t  change in network
        eth_dst = eth_pkt.dst

        out_port = self.get_out_port(datapath,eth_src,eth_dst,in_port)    #这里进行获取下一跳端口
        actions = [ofp_parser.OFPActionOutput(out_port)]

        if out_port != ofproto.OFPP_FLOOD:
            match = ofp_parser.OFPMatch(in_port=in_port,eth_dst=eth_dst)
            self.add_flow(datapath,1,match,actions)

        out = ofp_parser.OFPPacketOut(
                datapath=datapath,buffer_id=msg.buffer_id,in_port=in_port,
                actions=actions,data=msg.data
            )

        datapath.send_msg(out)
```

三：实验演示：注意开启顺序，否则可能导致其他错误

（一）启动Ryu控制器 observe-links命令会导致控制器在运行期间会不间断地发送LLDP数据包进行链路探测

Shell 复制代码

```shell
ryu-manager my_shortest_forward.py --observe-links --verbose
```

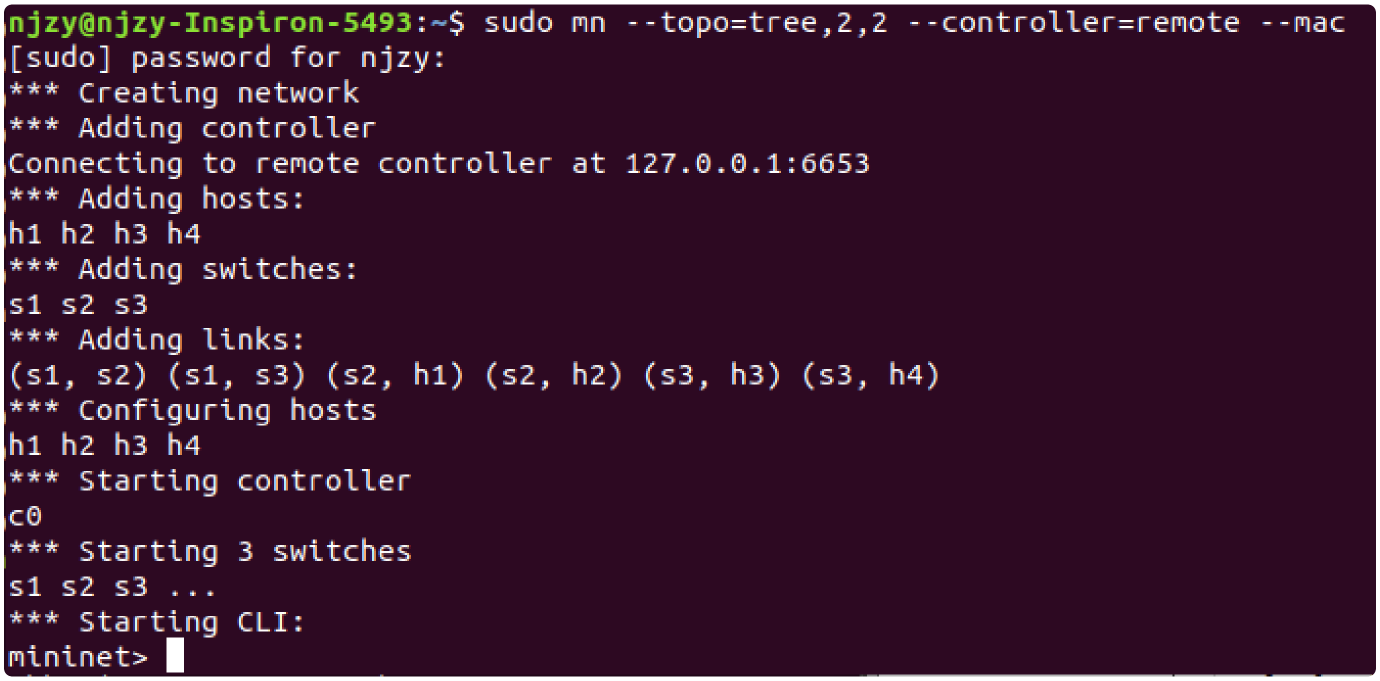


```
^Cnjzy@njzy-Inspiron-5493:~/CODE/python/SDN_Controller/ryu/ryu/app$ ryu-manager my_shortest_forward.py --observe-links
loading app my_shortest_forward.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app my_shortest_forward.py of MyShortestForwarding
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
```

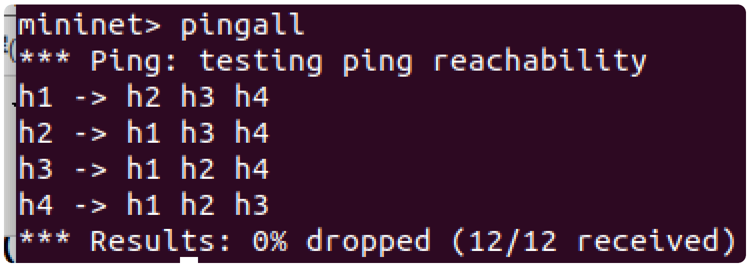
（二）启动Mininet

Shell 复制代码

```shell
sudo mn --topo=tree,2,2 --controller=remote --mac
```

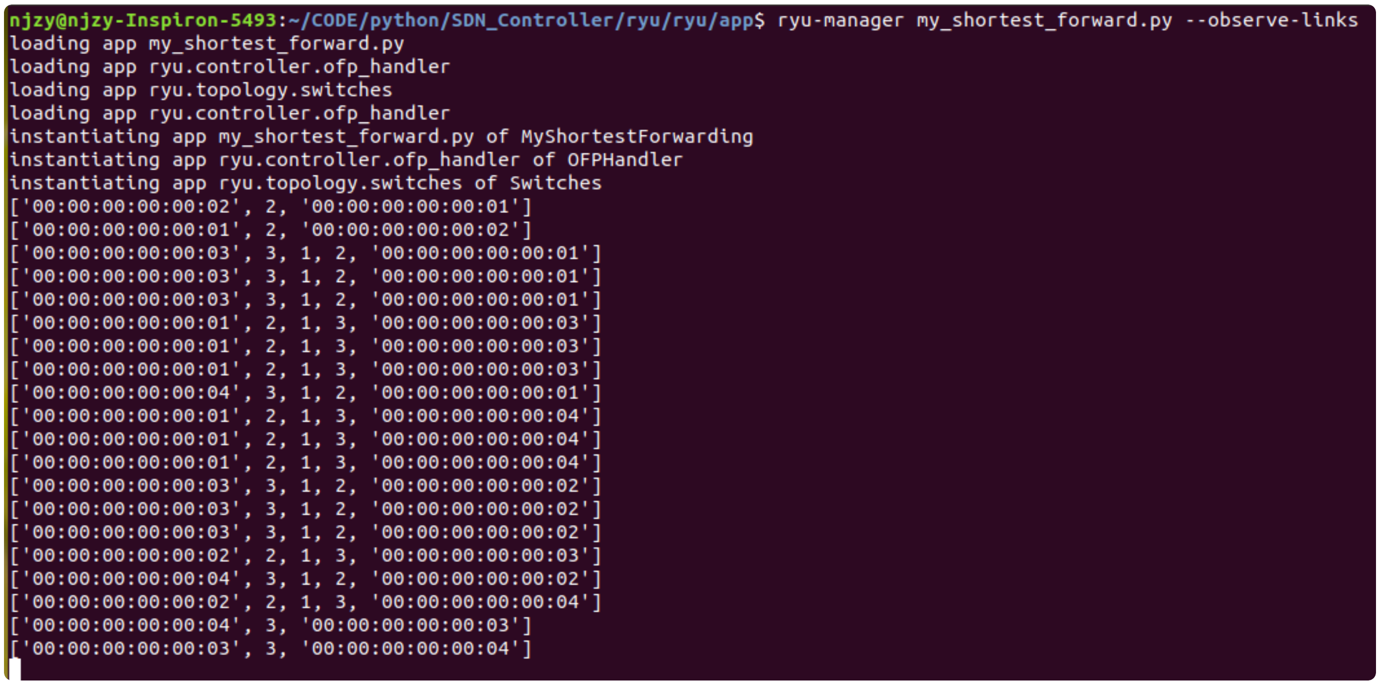


```
njzy@njzy-Inspiron-5493:~$ sudo mn --topo=tree,2,2 --controller=remote --mac
[sudo] password for njzy:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```



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

（三）Ryu查看信息



```
njzy@njzy-Inspiron-5493:~/CODE/python/SDN_Controller/ryu/ryu/app$ ryu-manager my_shortest_forward.py --observe-links
loading app my_shortest_forward.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app my_shortest_forward.py of MyShortestForwarding
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
['00:00:00:00:00:02', 2, '00:00:00:00:00:01']
['00:00:00:00:00:01', 2, '00:00:00:00:00:02']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:04', 3, 1, 2, '00:00:00:00:00:01']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:01', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:03', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:02', 2, 1, 3, '00:00:00:00:00:03']
['00:00:00:00:00:04', 3, 1, 2, '00:00:00:00:00:02']
['00:00:00:00:00:02', 2, 1, 3, '00:00:00:00:00:04']
['00:00:00:00:00:04', 3, '00:00:00:00:00:03']
['00:00:00:00:00:03', 3, '00:00:00:00:00:04']
```