网络算法基础项目三

一、基本原理

- **Flow-Mod 信息**

Flow-Mod 消息是 OpenFlow 控制器对 OpenFlow 交换机设置流表项的消息。可对流表项进行添加、删除、变更设置等操作。整个消息可以分为三部分：openflow 主体部分、match 部分、instruction 部分。match 部分是匹配条件，instruction 部分是指令，当一个数据包满足匹配条件就会执行 instruction 中的指令。

- **ARP**

ARP 协议，即地址解析协议，可以通过解析 IP 地址得到 MAC 地址。主要通过报文工作，ARP 报文分为 ARP 请求和 ARP 应答报文两种。

ARP 请求报文：

当一个主机想要找出另一个主机的 MAC 地址时，首先会查看自己的 ARP 缓存表，若在 ARP 缓存表中找不到对应的 MAC 地址，则将缓存该数据报文，然后以广播方式发送一个 ARP 请求报文。ARP 请求报文中的发送端 IP 地址和发送端 MAC 地址为 h1 的 IP 地址和 MAC 地址，目标 IP 地址和目标 MAC 地址为 h2 的 IP 地址和全 0 的 MAC 地址。

ARP 应答报文：

受到请求报文的主机比较自己的IP地址和ARP请求报文中的目标IP地址，当两者相同时将ARP请求报文中的发送端的IP地址和MAC地址存入自己的ARP表中。之后以单播方式发送ARP应答报文给发送端，其中包含了自己的MAC地址（只有验证成功的主机才会发送ARP应答报文。

- **h1 ping h2 的过程**

1.h1 查看自己的 ARP 缓存表，若其中有 h2 对应的表项，将直接利用 ARP 表中的 MAC 地址，对 IP 数据包帧封装，并将数据包发送给 h2；

2.若 h1 的 ARP 缓存表中没有 h2 对应的表项，将缓存该数据报文，然后以广播方式发送一个 ARP 请求报文；

3.h2 比较自己的 IP 地址和 ARP 请求报文中的目标 IP 地址，当两者相同时将 ARP 请求报文中的发送端（即 h1）的 IP 地址和 MAC 地址存入自己的 ARP 表中。之后将 ARP 应答报文单独发送给 h1；

4. h1 收到 ARP 应答报文后，将 h2 的 MAC 地址加入到自己的 ARP 缓存表中，同时将 IP 数据包封装并发送出去。

二、Dial 实现的 Dijkstra 算法思路

首先创建 bucket 列表，由于该图中设定边最大权重为 5，因此在列表中存入 6 个空列表作为桶，创建字典 d 存储节点与起点之间最短路径，并将起点的最短路置为 0，其他节点置为无穷，创建字典 pre 存储最短路径中节点的父节点，起点的父节点置为本身，创建字典 point 记录节点存入的桶的编号，对 flag 编号的桶，桶不为空时不断取出桶中的元素，存入 min_node 中，若该元素为目标节点，则停止取出，对于取出的元素遍历其临界节点，当元素节点的最短路径与其邻接节点的路径权重之和小于在字典 d 中存储的最短距离时，更新字典 d，pre，point，并将邻接节点存入相应桶中，flag++，若 flag>5,则

flag=flag%6，桶全为空时停止循环，通过字典 pre 得到最短路径，存入 current_path 中。

三、代码详情

```
from collections import defaultdict
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.topology import event
from ryu.controller.handler import MAIN_DISPATCHER,CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.topology.api import get_switch,get_all_link,get_link
import os
import copy
import random
from ryu.lib.packet import arp
from ryu.lib import mac
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(graph,path,weight):
    nodes = set([n1 for n1, n2 in graph] + [n2 for n1, n2 in graph])
    G=nx.Graph()
    for node in nodes:
        G.add_node(node)

    G.add_edges_from(graph,color='purple')
    G.add_edges_from(path,color='orange')
```

```
        pos=nx.spring_layout(G)
        edges = G.edges()
        colors = [G[u][v]['color'] for u,v in edges]

        nx.draw_networkx_nodes(G,pos,node_size=400)
        nx.draw_networkx_edges(G,pos,width=2,edge_color=colors)
        nx.draw_networkx_labels(G,pos,font_size=10)

        nx.draw_networkx_edge_labels(G,pos,weight,font_size=7)

        if not os.path.exists("./show_photo/"):
            os.mkdir("./show_photo/")
        if path != []:
            plt.savefig('./show_photo/' + str(path[0][0]) + "---" + str(path[-1][0]) + ".png")
        else:
            plt.savefig('./show_photo/Topo.png')
            plt.savefig('Topo.png')
        plt.savefig('now_photo.png')
        plt.close()

class Topo(object):
    def __init__(self,logger):

        self.switches=None
        self.graph = None

        self.host_mac_to={}
        self.logger=logger
        self.flag=[]
        self.weight = None
        self.edges={}

    # 使用循环桶实现 Dijkstra 算法求最短路
    def compute_path(self,src_sw,dst_sw,first_port,last_port):
        bucket = []
        for i in range(6):
```

```
        bucket.append([])
    node_s = (0, src_sw)                # 源节点二元组（距离标记，节点）
    d = {} #距离
    for u in self.switches:
        d[u] = 9999999
    d[src_sw] = 0
    pre = {}                            # 父节点

    bucket[0].append(node_s)
    flag = 0
    point={} #记录在桶中的相应位置
    point[src_sw]=0
    while bucket != [[]]*6 :    # 所有的桶未空
        min_list = bucket[flag] #抽取一个桶中的元素
        while not len(min_list) == 0:
            min_node = min_list.pop()   #这个桶出一个节点
            del point[min_node[1]]   #删去这个节点在桶中的指向
            if (min_node[1] == dst_sw):
                break

            for u in self.switches:
                if(min_node[1]==u):
                    continue
                if (min_node[1],u) in self.edges:
                    if (d[min_node[1]] + (self.edges[(min_node[1],u)])[1] < d[u]):    #
更短就更新桶中的节点位置
                        pre[u] = min_node[1]
                        if u in point:
                            bucket[point[u]].remove((d[u], u))
                        d[u] = d[min_node[1]] + self.edges[(min_node[1],u)][1]   #更
新距离
                        bucket[d[u] % 6].append((d[u], u))   #重新加入桶
                        point[u] = d[u] % 6 #循环利用桶所以取模
        flag += 1
        if flag > 5:
            flag %= 6 #大于桶的长度取模循环
```

```
current_path=[]
s = dst_sw
while s != src_sw:
        current_path.append(s)
        s = pre[s]
current_path.append(src_sw)

current_path.reverse()

graph=[]
weight={}
for a in self.switches:
        for b in self.switches:
                if a==b:
                        continue;
                if (a,b) in self.edges:
                        graph.append((a,b))
                        weight[(a,b)] = self.edges[(a,b)][1]
graph_path=[]
for i in range(len(current_path)-1):
        graph_path.append((current_path[i],current_path[i+1]))
        graph_path.append((current_path[i+1],current_path[i]))

self.weight = weight
self.graph = graph

print("the shortest path is: ")
print(current_path)

if src_sw==dst_sw:
                path=[src_sw]
else:
                path=current_path

record=[]
```

```
            inport=first_port

            for s1,s2 in zip(path[:-1],path[1:]):
                outport,_=self.edges[(s1,s2)]

                record.append((s1,inport,outport))
                inport,_=self.edges[(s2,s1)]

            record.append((dst_sw,inport,last_port))

            return record,graph_path

# Ryu 控制器
class DijkstraController(app_manager.RyuApp):
    # 指明 OpenFlow 版本
    OFP_VERSIONS=[ofproto_v1_3.OFP_VERSION]

    def __init__(self,*args,**kwargs):
        super(DijkstraController,self).__init__(*args,**kwargs)
        self.mac_to_port={} # 全局的 mac 表,{{datapath:mac->port},...,{datapath:mac->port}}
        self.datapaths=[]
        self.initshow = 0
        self.flag = []
        self.arp_table={}

        self.topo=Topo(self.logger)
        self.flood_history={}    # 泛洪历史表

        self.arp_history={} # arp 历史表

    # 向控制器传输交换机特征
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```

```
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    # 添加流表
    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    # 配置路径
    def configure_path(self,shortest_path,event,src_mac,dst_mac):
        msg=event.msg
        datapath=msg.datapath
        flag=0
        ofproto=datapath.ofproto

        parser=datapath.ofproto_parser
        for switch,inport,outport in shortest_path:
            match=parser.OFPMatch(in_port=inport,eth_src=src_mac,eth_dst=dst_mac)

            actions=[parser.OFPActionOutput(outport)]

            # 发现该交换机的 dpid，并将 dpid 赋值给 datapath
```

```
            for dp in self.datapaths:
                if dp.id==int(switch):
                    datapath=dp
                    flag=1
                    break
            if flag==0:
                datapath=None
            assert datapath is not None

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

            mod=datapath.ofproto_parser.OFPFlowMod(
                datapath=datapath,
                match=match,
                idle_timeout=0,
                hard_timeout=0,
                priority=1,
                instructions=inst
            )

            # 下发流表
            datapath.send_msg(mod)

    # 监听 Packet_in 事件
    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_DISPATCHER)
    def packet_in_handler(self,event):
        msg=event.msg
        datapath=msg.datapath
        ofproto=datapath.ofproto
        parser=datapath.ofproto_parser
        in_port=msg.match['in_port']

        # 获取数据
        pkt=packet.Packet(msg.data)

        # 假设为以太帧，获取帧头
```

```
eth=pkt.get_protocols(ethernet.ethernet)[0]

# 丢弃 LLDP 帧
if eth.ethertype==ether_types.ETH_TYPE_LLDP:
    return

dst_mac=eth.dst
src_mac=eth.src
arp_pkt = pkt.get_protocol(arp.arp)

if arp_pkt:
    self.arp_table[arp_pkt.src_ip] = src_mac

dpid=datapath.id
self.mac_to_port.setdefault(dpid,{})
self.mac_to_port[dpid][src_mac]=in_port
self.flood_history.setdefault(dpid,[])

if '33:33' in dst_mac[:5]:
    if (src_mac,dst_mac) not in self.flood_history[dpid]:
        self.flood_history[dpid].append((src_mac,dst_mac))
    else:
        return

if src_mac not in self.topo.host_mac_to.keys():
    self.topo.host_mac_to[src_mac]=(dpid,in_port)

if dst_mac in self.topo.host_mac_to.keys():
    final_port=self.topo.host_mac_to[dst_mac][1]
    src_switch=self.topo.host_mac_to[src_mac][0]
    dst_switch=self.topo.host_mac_to[dst_mac][0]

    shortest_path,path=self.topo.compute_path(
        src_switch,
        dst_switch,
        in_port,
```

```
                final_port)

            if set([path[0][0], path[-1][0]]) not in self.flag:
                draw_graph(self.topo.graph, path, self.topo.weight)
                self.flag.append(set([path[0][0], path[-1][0]]))

            self.logger.info("The shortest path from {} to {} contains {} switches".format(src_mac,dst_mac,len(shortest_path)))

            assert len(shortest_path)>0

            path_str=''
            for s,ip,op in shortest_path:
                path_str=path_str+"--{}-{}-{}--".format(ip,s,op)

            self.configure_path(shortest_path,event,src_mac,dst_mac)

            self.logger.info("----------------done----------------")

            out_port=None
            for s,_,op in shortest_path:
                 if s==dpid:
                     out_port=op

        else:
            if self.arp_handler(msg):
                return
            out_port=ofproto.OFPP_FLOOD
        actions=[parser.OFPActionOutput(out_port)]

        data=None
        if msg.buffer_id==ofproto.OFP_NO_BUFFER:
            data=msg.data

        out=parser.OFPPacketOut(
            datapath=datapath,
```

```
            buffer_id=msg.buffer_id,
            in_port=in_port,
            actions=actions,
            data=data
        )
        datapath.send_msg(out)

    # 交换机进入时触发
    @set_ev_cls(event.EventSwitchEnter)
    def switch_enter_handler(self,event):
        self.logger.info("一个交换机进入，重新发现拓扑")
        self.switch_status_handler(event)
        self.logger.info('拓扑发现完毕')

        weight = {}
        self.initshow += 1
        if self.initshow == len(self.topo.switches):
            graph = []
            for a in self.topo.switches:
                for b in self.topo.switches:
                    if (a,b) in self.topo.edges:
                        weight[(a, b)] = self.topo.edges[(a, b)][1]
                        graph.append((a, b))
            graph_path = []
            draw_graph(graph, graph_path, weight)

    # 交换机离开时触发
    @set_ev_cls(event.EventSwitchLeave)
    def switch_leave_handler(self,event):
        self.logger.info("一个交换机退出，重新发现拓扑")
        self.switch_status_handler(event)
        self.logger.info('拓扑发现完毕')

    # 配置交换机状态与打印连通信息
    def switch_status_handler(self,event):
        all_switches=copy.copy(get_switch(self,None))
```

```
        # 获取交换机的 ID 值
        self.topo.switches=[s.dp.id for s in all_switches]

        self.logger.info("switches {}".format(self.topo.switches))

        self.datapaths=[s.dp for s in all_switches]

        all_links=copy.copy(get_link(self,None))

        all_link_stats=[(l.src.dpid,l.dst.dpid,l.src.port_no,l.dst.port_no) for l in all_links]

        self.logger.info("Number of links {}".format(len(all_link_stats)))

        all_link_repr=''

        for s1,s2,p1,p2 in all_link_stats:

            weight=random.randint(1,5)
            self.topo.edges[(s1,s2)]=(p1,weight)
            self.topo.edges[(s2,s1)]=(p2,weight)

            all_link_repr+='s{}p{}--s{}p{}\n'.format(s1,p1,s2,p2)
        self.logger.info("All links:\n "+all_link_repr)

    def arp_handler(self, msg):
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        arp_pkt = pkt.get_protocol(arp.arp)

        if eth:
            eth_dst = eth.dst
```

```
        eth_src = eth.src

    if eth_dst == mac.BROADCAST_STR and arp_pkt:
        arp_dst_ip = arp_pkt.dst_ip
        if (datapath.id, eth_src, arp_dst_ip) in self.arp_history:
            if self.arp_history[(datapath.id, eth_src, arp_dst_ip)] != in_port:
                return True
        else:
            self.arp_history[(datapath.id, eth_src, arp_dst_ip)] = in_port

    if arp_pkt:
        hwtype = arp_pkt.hwtype
        proto = arp_pkt.proto
        hlen = arp_pkt.hlen
        plen = arp_pkt.plen

        opcode = arp_pkt.opcode
        arp_src_ip = arp_pkt.src_ip
        arp_dst_ip = arp_pkt.dst_ip

        if opcode == arp.ARP_REQUEST:
            if arp_dst_ip in self.arp_table:
                actions = [parser.OFPActionOutput(in_port)]
                arp_reply = packet.Packet()
                arp_reply.add_protocol(ethernet.ethernet(
                    ethertype=eth.ethertype,
                    dst=eth_src,
                    src=self.arp_table[arp_dst_ip]))
                arp_reply.add_protocol(arp.arp(
                    opcode=arp.ARP_REPLY,
                    src_mac=self.arp_table[arp_dst_ip],
                    src_ip=arp_dst_ip,
                    dst_mac=eth_src,
                    dst_ip=arp_src_ip))
                arp_reply.serialize()
                out = parser.OFPPacketOut(
```

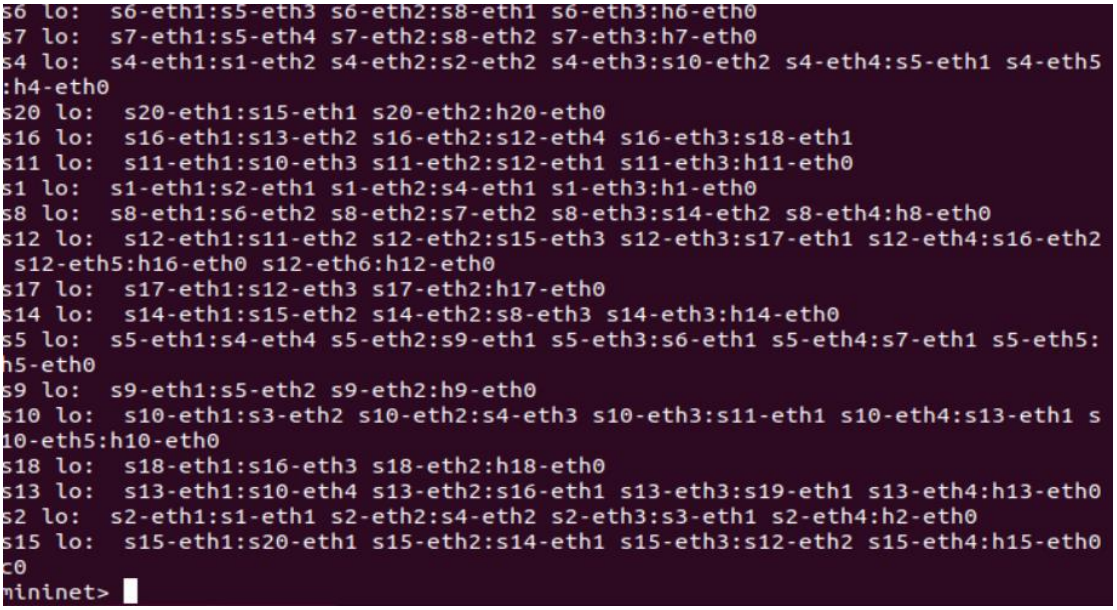
```
                datapath=datapath,
                buffer_id=ofproto.OFP_NO_BUFFER,
                in_port=ofproto.OFPP_CONTROLLER,
                actions=actions, data=arp_reply.data)
            datapath.send_msg(out)

            return True

        return False
```

四、结果展示

1. 测试拓扑信息
   a. 查看链路信息



   b. 查看链路是否可用

```
s16-eth2<->s12-eth4 (OK OK)
s16-eth3<->s18-eth1 (OK OK)
s13-eth3<->s19-eth1 (OK OK)
h1-eth0<->s1-eth3 (OK OK)
h2-eth0<->s2-eth4 (OK OK)
h3-eth0<->s3-eth3 (OK OK)
h19-eth0<->s19-eth2 (OK OK)
h13-eth0<->s13-eth4 (OK OK)
h18-eth0<->s18-eth2 (OK OK)
h16-eth0<->s12-eth5 (OK OK)
h17-eth0<->s17-eth2 (OK OK)
h12-eth0<->s12-eth6 (OK OK)
h15-eth0<->s15-eth4 (OK OK)
h14-eth0<->s14-eth3 (OK OK)
h8-eth0<->s8-eth4 (OK OK)
h20-eth0<->s20-eth2 (OK OK)
h7-eth0<->s7-eth3 (OK OK)
h11-eth0<->s11-eth3 (OK OK)
h6-eth0<->s6-eth3 (OK OK)
h5-eth0<->s5-eth5 (OK OK)
h9-eth0<->s9-eth2 (OK OK)
h4-eth0<->s4-eth5 (OK OK)
h10-eth0<->s10-eth5 (OK OK)
mininet>
```

c. 查看可用节点

```
available nodes are:
c0 h1 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h2 h20 h3 h4 h5 h6 h7 h8 h9 s1 s10
 s11 s12 s13 s14 s15 s16 s17 s18 s19 s2 s20 s3 s4 s5 s6 s7 s8 s9
mininet>
```

d. 查看节点信息

```
<OVSSwitch s11: lo:127.0.0.1,s11-eth1:None,s11-eth2:None,s11-eth3:None pid=42320
>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=42323>
<OVSSwitch s8: lo:127.0.0.1,s8-eth1:None,s8-eth2:None,s8-eth3:None,s8-eth4:None
pid=42326>
<OVSSwitch s12: lo:127.0.0.1,s12-eth1:None,s12-eth2:None,s12-eth3:None,s12-eth4:
None,s12-eth5:None,s12-eth6:None pid=42329>
<OVSSwitch s17: lo:127.0.0.1,s17-eth1:None,s17-eth2:None pid=42332>
<OVSSwitch s14: lo:127.0.0.1,s14-eth1:None,s14-eth2:None,s14-eth3:None pid=42335
>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None,s5-eth4:None,
s5-eth5:None pid=42338>
<OVSSwitch s9: lo:127.0.0.1,s9-eth1:None,s9-eth2:None pid=42341>
<OVSSwitch s10: lo:127.0.0.1,s10-eth1:None,s10-eth2:None,s10-eth3:None,s10-eth4:
None,s10-eth5:None pid=42344>
<OVSSwitch s18: lo:127.0.0.1,s18-eth1:None,s18-eth2:None pid=42347>
<OVSSwitch s13: lo:127.0.0.1,s13-eth1:None,s13-eth2:None,s13-eth3:None,s13-eth4:
None pid=42350>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None
pid=42353>
<OVSSwitch s15: lo:127.0.0.1,s15-eth1:None,s15-eth2:None,s15-eth3:None,s15-eth4:
None pid=42356>
<RemoteController c0: 127.0.0.1:6633 pid=42292>
mininet>
```

e. 查看连通性

连接 ryu 前

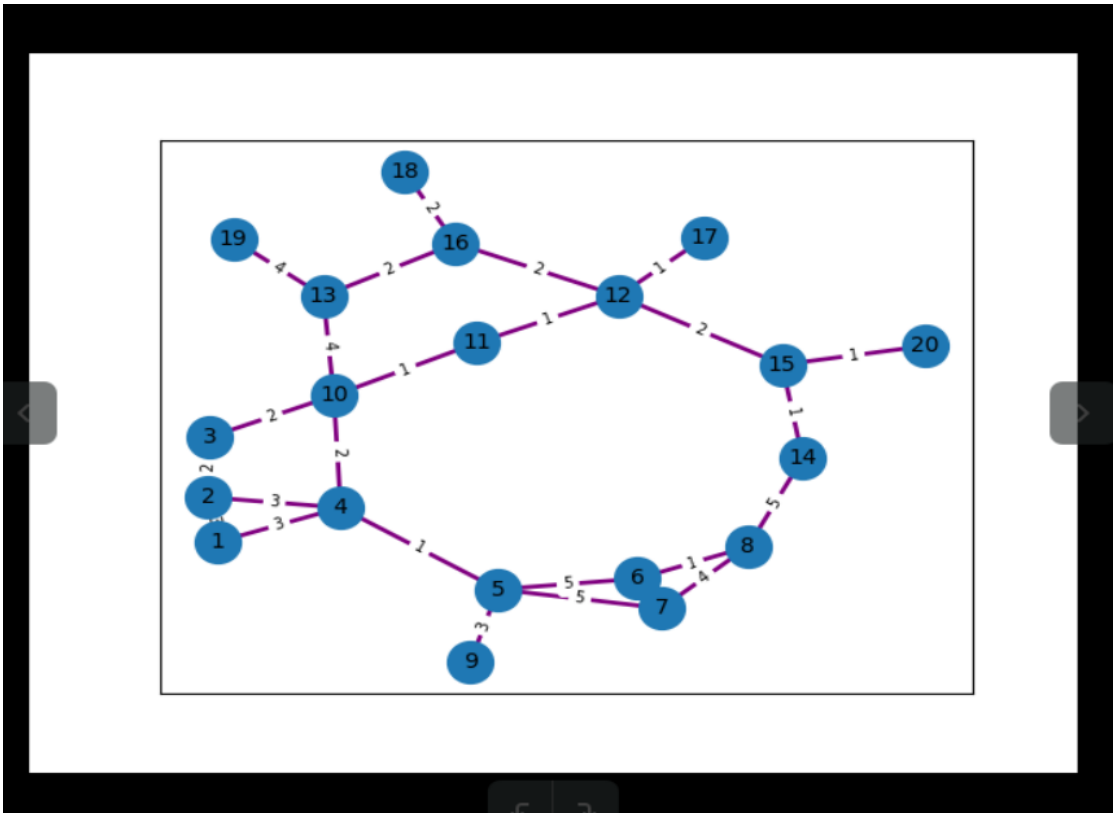
```
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3079ms
pipe 4
mininet>
```

连接 ryu 后

```
*** Ping: testing ping reachability
h4 -> h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h1 -> h4 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h17 -> h4 h1 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h19 -> h4 h1 h17 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h12 -> h4 h1 h17 h19 h14 h5 h20 h6 h8 h15 h3 h9 h10 X h13 h7 h11 h2 h18
h14 -> h4 h1 h17 h19 h12 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h5 -> h4 h1 h17 h19 h12 h14 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h20 -> h4 h1 h17 h19 h12 h14 h5 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h6 -> h4 h1 h17 h19 h12 h14 h5 h20 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h8 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h15 h3 h9 h10 h16 h13 h7 h11 h2 h18
h15 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h3 h9 h10 h16 h13 h7 h11 h2 h18
h3 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h9 h10 h16 h13 h7 h11 h2 h18
h9 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h10 h16 h13 h7 h11 h2 h18
h10 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h16 h13 h7 h11 h2 h18
h16 -> h4 h1 h17 h19 X h14 h5 h20 h6 h8 h15 h3 h9 h10 h13 h7 h11 h2 h18
h13 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h7 h11 h2 h18
h7 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h11 h2 h18
h11 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h2 h18
h2 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h18
h18 -> h4 h1 h17 h19 h12 h14 h5 h20 h6 h8 h15 h3 h9 h10 h16 h13 h7 h11 h2
*** Results: 0% dropped (378/380 received)
mininet>
```

2. 查看生成的 topo



3. 查看最短路