网络算法基础项目四

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；

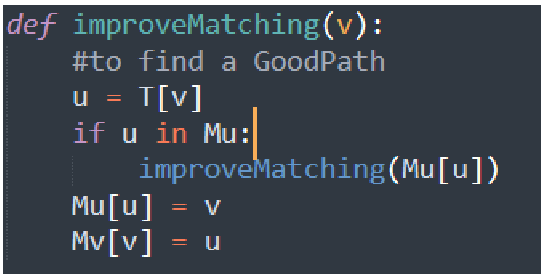
4. h1 收到 ARP 应答报文后，将 h2 的 MAC 地址加入到自己的 ARP 缓存表中，同时将 IP 数据包封装并发送出去。

二、项目实现

假设网络中有 1000 个流，每个流 f 都有一个权重值（代表重要性，随机生成），假定每个 SDN 交换机只有 20 个可用流表项，假如一个流经过了两个交换机 A 和 B，则只需要在 A 或者 B 处使用单独的流表项对流进行测量，即假如在 A 点测量则 A 点需要使用一个单独的流表项（比汇聚表项优先级更高），因此需要从 1000 个流中选择出权之和最大的流集合来进行测量，并得到每个流的测量点。

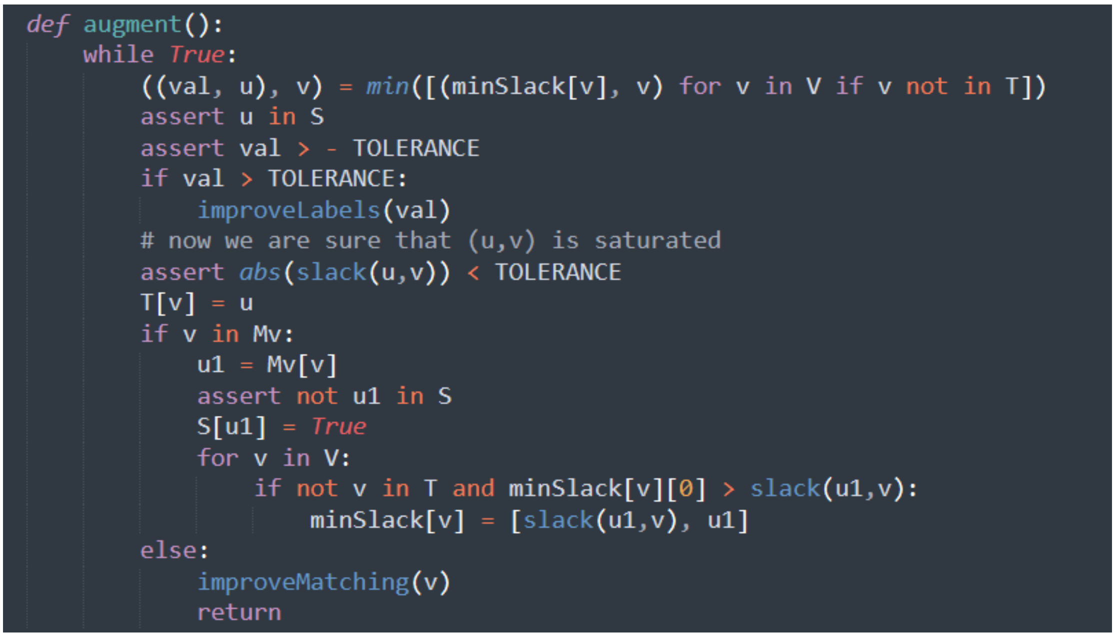
首先编写匈牙利算法代码，可分为 3 个部分：

1. 找到 GoodPath



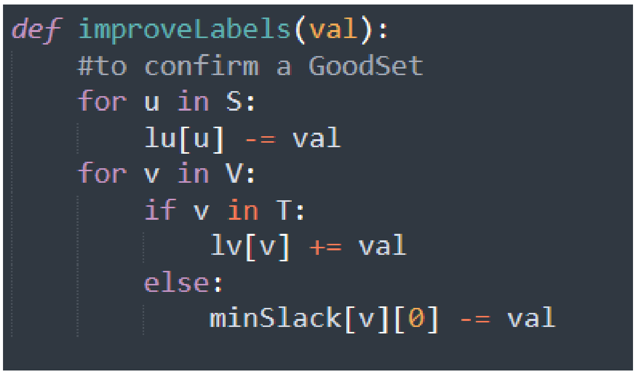
```
def improveMatching(v):
    #to find a GoodPath
    u = T[v]
    if u in Mu:
        improveMatching(Mu[u])
    Mu[u] = v
    Mv[v] = u
```

2. 增广匹配



```
def augment():
    while True:
        ((val, u), v) = min([(minSlack[v], v) for v in V if v not in T])
        assert u in S
        assert val > - TOLERANCE
        if val > TOLERANCE:
            improveLabels(val)
        # now we are sure that (u,v) is saturated
        assert abs(slack(u,v)) < TOLERANCE
        T[v] = u
        if v in Mv:
            u1 = Mv[v]
            assert not u1 in S
            S[u1] = True
            for v in V:
                if not v in T and minSlack[v][0] > slack(u1,v):
                    minSlack[v] = [slack(u1,v), u1]
        else:
            improveMatching(v)
            return
```

3. 确定 GoodSet



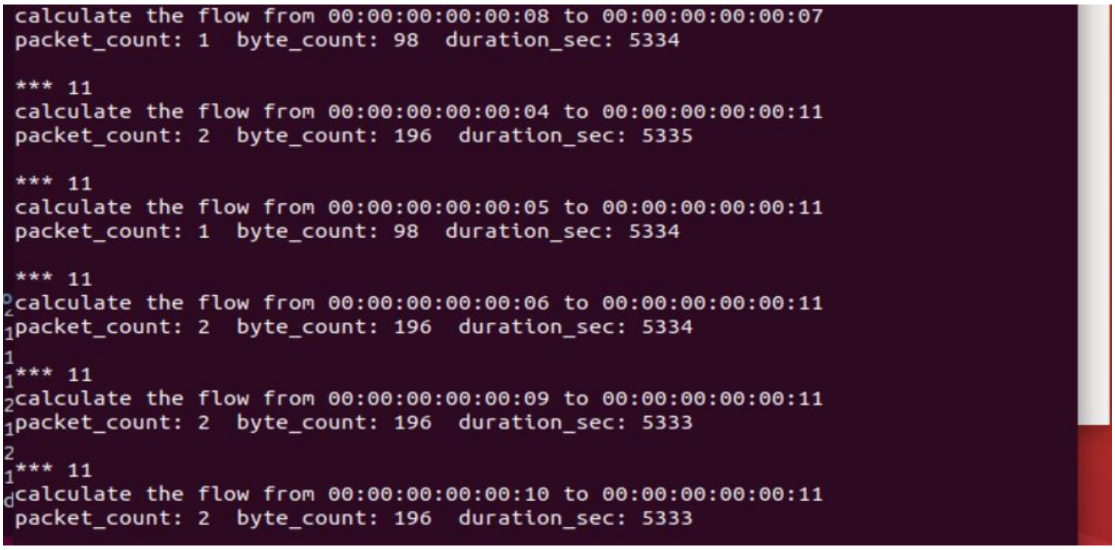
```
def improveLabels(val):
    #to confirm a GoodSet
    for u in S:
        lu[u] -= val
    for v in V:
        if v in T:
            lv[v] += val
        else:
            minSlack[v][0] -= val
```

仍使用原来的拓扑图，20 个交换机，每个交换机测 20 个流表，最多测量 400 个流表，无法满足需求，因此构造 1000*1000 的权重矩阵，将每个交换机都复制 20 次。代入匈牙利算法中，就等同于测量 20 个流表。若某个交换机无法测量某个流，则该边权重为 0。同时，为了保证完美匹配，还要在 400 个交换机的基础上补足到 1000 个交换机。补足的交换机的权重都为 0。（该部分代码见 match.py）

之后用 Iperf 生成流（流的数目为 100，容量为 5），在 mininet 中测量选择的流的大小，并给出在什么地方测量。为了方便计算，在一开始就把所需 100 个流的信息输入进去（如下图所示，其中字典 key 值为元组，存储流的起始主机与目的主机，value 值代表权重），仍用原来的拓扑图。在我们的 ryu 控制器中，使用 dijkstra 算法去配置任意两点之间的路径，即当输入 iperf 后，调用 dijkstra()函数，在配置路径的同时，记录路径，将路径作为输入再回到匈牙利算法。除此之外，需要用到测量流表。

（此部分代码见 ryu.py）

最后输出结果如下：



```
calculate the flow from 00:00:00:00:00:08 to 00:00:00:00:00:07
packet_count: 1  byte_count: 98  duration_sec: 5334

*** 11
calculate the flow from 00:00:00:00:00:04 to 00:00:00:00:00:11
packet_count: 2  byte_count: 196  duration_sec: 5335

*** 11
calculate the flow from 00:00:00:00:00:05 to 00:00:00:00:00:11
packet_count: 1  byte_count: 98  duration_sec: 5334

*** 11
calculate the flow from 00:00:00:00:00:06 to 00:00:00:00:00:11
packet_count: 2  byte_count: 196  duration_sec: 5334

*** 11
calculate the flow from 00:00:00:00:00:09 to 00:00:00:00:00:11
packet_count: 2  byte_count: 196  duration_sec: 5333

*** 11
calculate the flow from 00:00:00:00:00:10 to 00:00:00:00:00:11
packet_count: 2  byte_count: 196  duration_sec: 5333
```

打印出了：1.在哪个位置测量 2.所测的流起始节点和目的节点是什么

3.这个流经过了多少个包，经历了多少个字节数

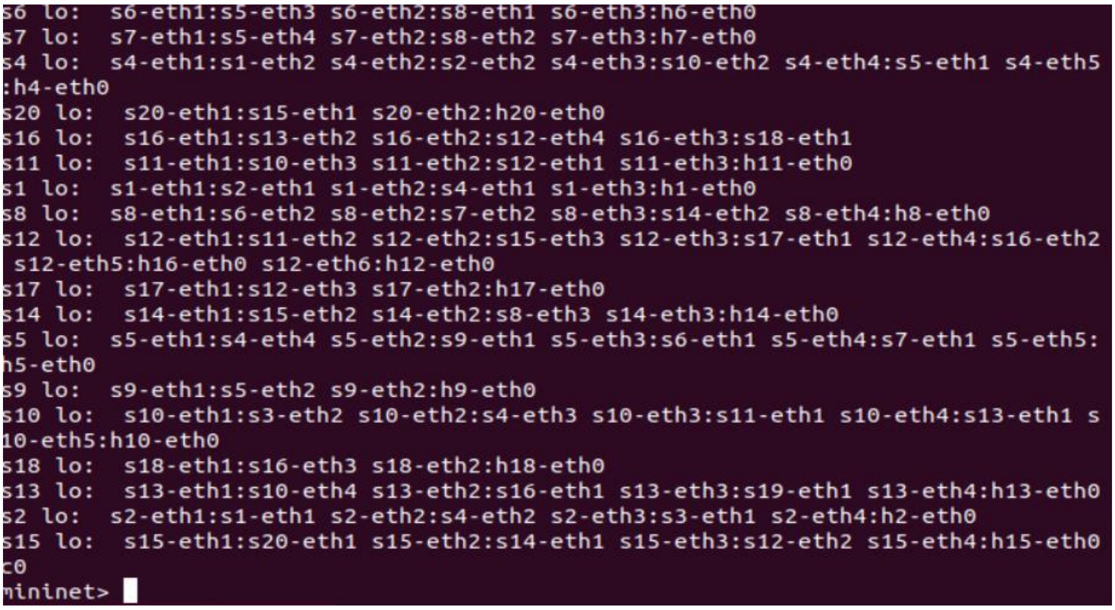
4.这个测量流表的存在时间（以 s 为单位）

（在这里每隔 1s 会统计并打印出结果一次）

三、结果展示

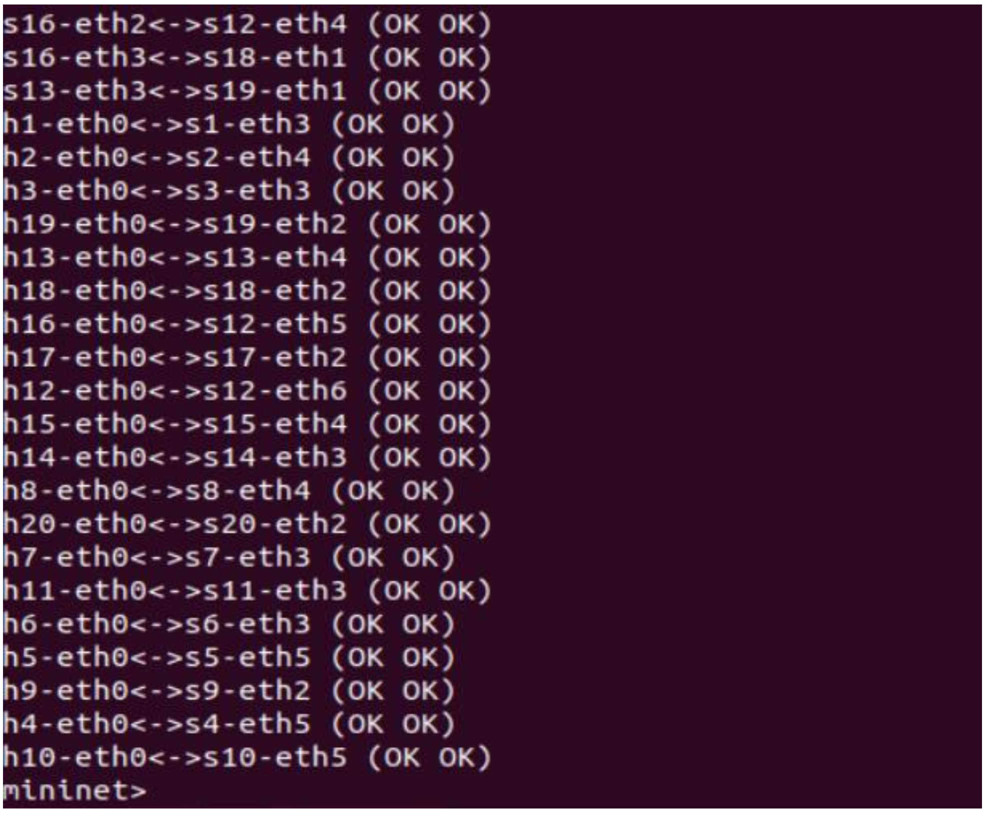
1. 测试拓扑信息

   a. 查看链路信息



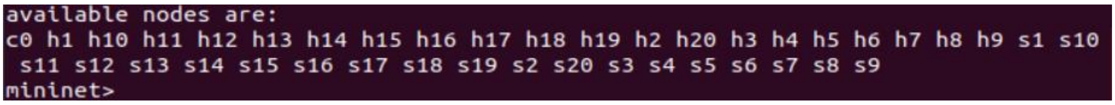
```
s6 lo:  s6-eth1:s5-eth3 s6-eth2:s8-eth1 s6-eth3:h6-eth0
s7 lo:  s7-eth1:s5-eth4 s7-eth2:s8-eth2 s7-eth3:h7-eth0
s4 lo:  s4-eth1:s1-eth2 s4-eth2:s2-eth2 s4-eth3:s10-eth2 s4-eth4:s5-eth1 s4-eth5
:h4-eth0
s20 lo:  s20-eth1:s15-eth1 s20-eth2:h20-eth0
s16 lo:  s16-eth1:s13-eth2 s16-eth2:s12-eth4 s16-eth3:s18-eth1
s11 lo:  s11-eth1:s10-eth3 s11-eth2:s12-eth1 s11-eth3:h11-eth0
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s4-eth1 s1-eth3:h1-eth0
s8 lo:  s8-eth1:s6-eth2 s8-eth2:s7-eth2 s8-eth3:s14-eth2 s8-eth4:h8-eth0
s12 lo:  s12-eth1:s11-eth2 s12-eth2:s15-eth3 s12-eth3:s17-eth1 s12-eth4:s16-eth2
 s12-eth5:h16-eth0 s12-eth6:h12-eth0
s17 lo:  s17-eth1:s12-eth3 s17-eth2:h17-eth0
s14 lo:  s14-eth1:s15-eth2 s14-eth2:s8-eth3 s14-eth3:h14-eth0
s5 lo:  s5-eth1:s4-eth4 s5-eth2:s9-eth1 s5-eth3:s6-eth1 s5-eth4:s7-eth1 s5-eth5:
h5-eth0
s9 lo:  s9-eth1:s5-eth2 s9-eth2:h9-eth0
s10 lo:  s10-eth1:s3-eth2 s10-eth2:s4-eth3 s10-eth3:s11-eth1 s10-eth4:s13-eth1 s
10-eth5:h10-eth0
s18 lo:  s18-eth1:s16-eth3 s18-eth2:h18-eth0
s13 lo:  s13-eth1:s10-eth4 s13-eth2:s16-eth1 s13-eth3:s19-eth1 s13-eth4:h13-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s4-eth2 s2-eth3:s3-eth1 s2-eth4:h2-eth0
s15 lo:  s15-eth1:s20-eth1 s15-eth2:s14-eth1 s15-eth3:s12-eth2 s15-eth4:h15-eth0
c0
mininet> 
```

b.　查看链路是否可用



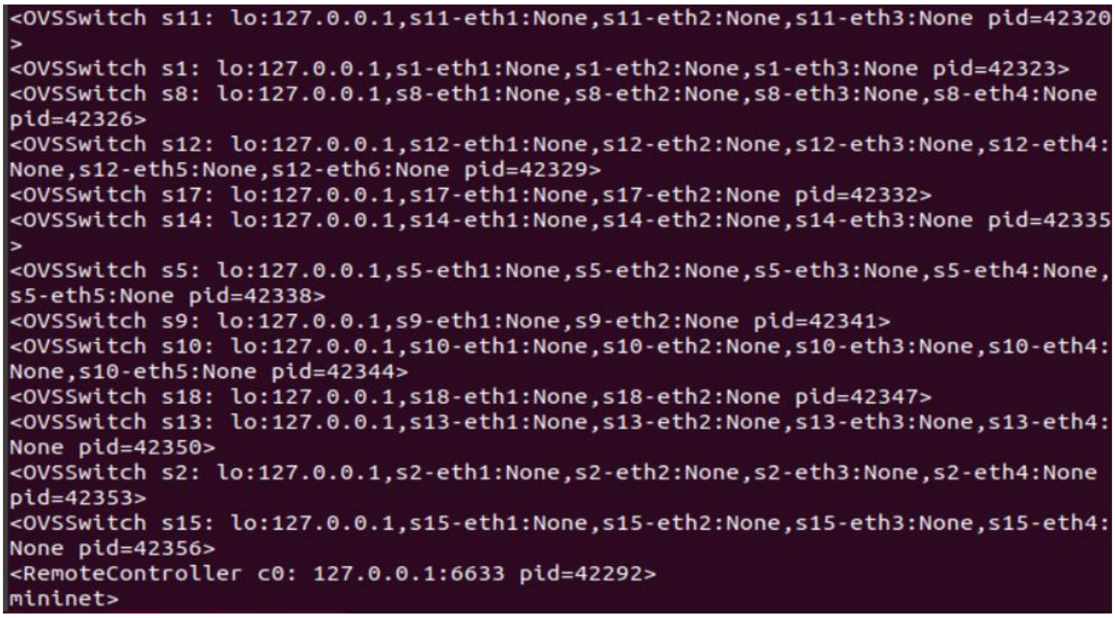
```
s16-eth2<->s12-eth4 (OK OK)
s16-eth3<->s18-eth1 (OK OK)
s13-eth3<->s19-eth1 (OK OK)
h1-eth0<->s1-eth3 (OK OK)
h2-eth0<->s2-eth4 (OK OK)
h3-eth0<->s3-eth3 (OK OK)
h19-eth0<->s19-eth2 (OK OK)
h13-eth0<->s13-eth4 (OK OK)
h18-eth0<->s18-eth2 (OK OK)
h16-eth0<->s12-eth5 (OK OK)
h17-eth0<->s17-eth2 (OK 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>
```

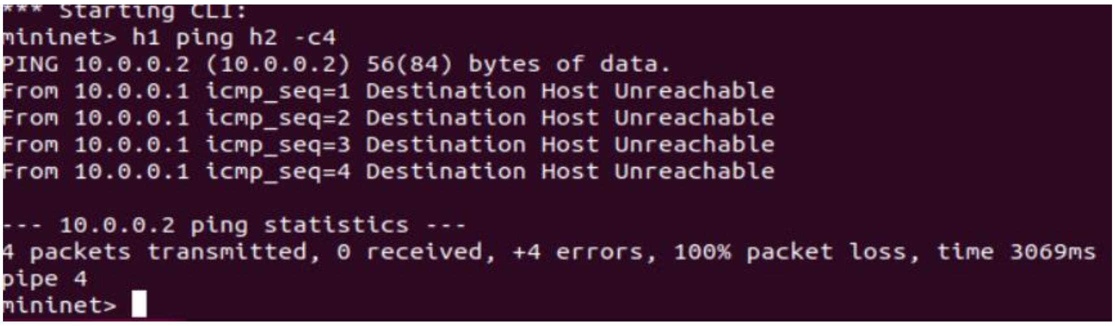
d.　查看节点信息



```
<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>
```

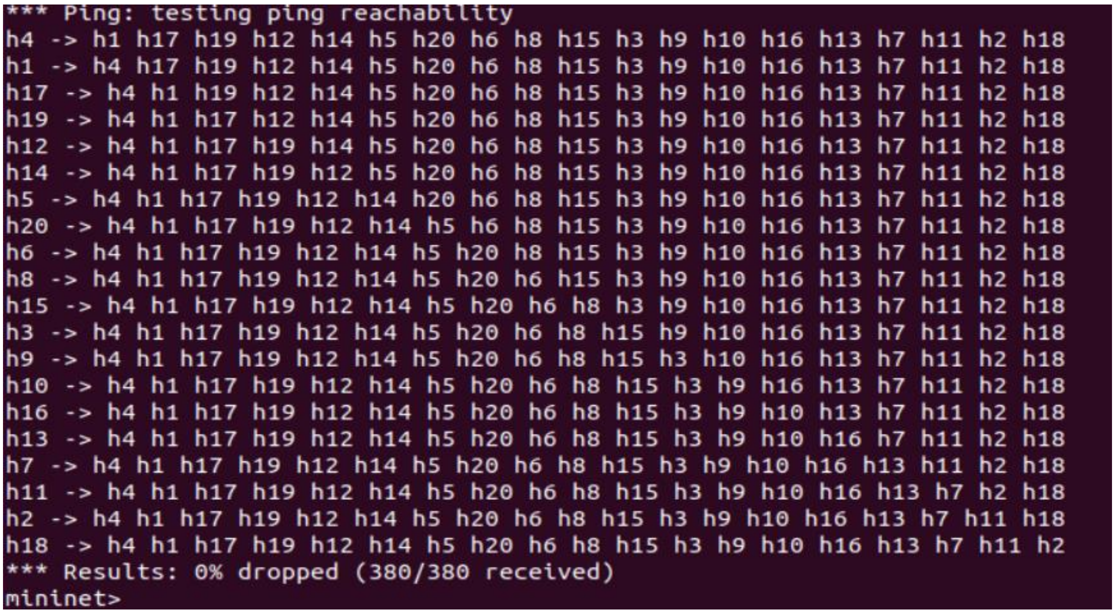
e. 查看连通性

连接 ryu 前



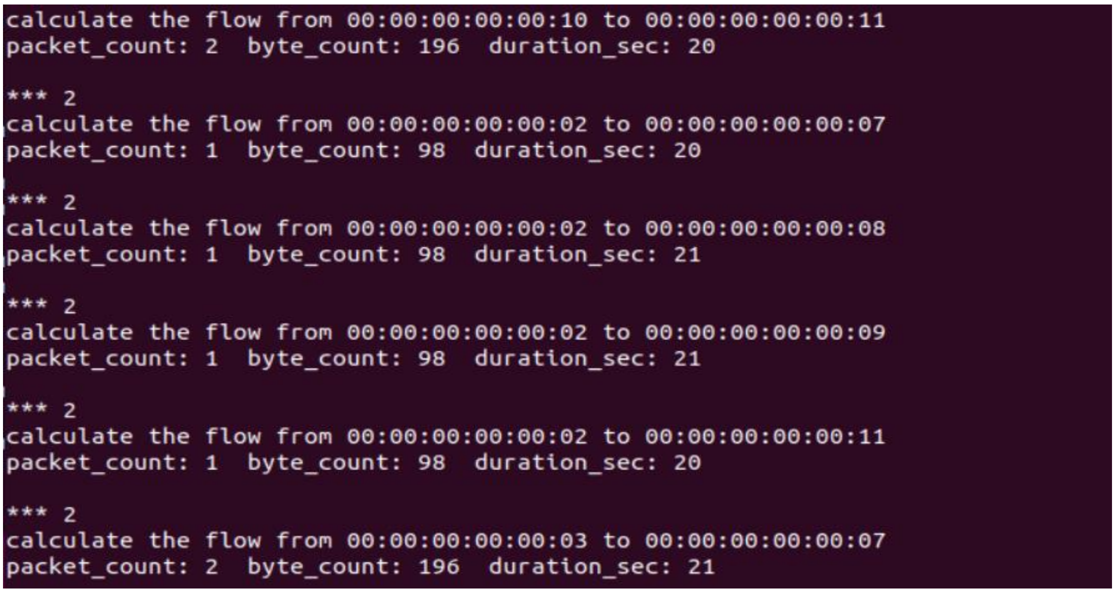
```
*** Starting CLI:
mininet> h1 ping h2 -c4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3069ms
pipe 4
mininet> 
```

使用命令 ryu-manager cal.py –observe-links 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 h16 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 h12 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 (380/380 received)
mininet>
```

2. 查看所测信息



```
calculate the flow from 00:00:00:00:00:10 to 00:00:00:00:00:11
packet_count: 2  byte_count: 196  duration_sec: 20

*** 2
calculate the flow from 00:00:00:00:00:02 to 00:00:00:00:00:07
packet_count: 1  byte_count: 98  duration_sec: 20

*** 2
calculate the flow from 00:00:00:00:00:02 to 00:00:00:00:00:08
packet_count: 1  byte_count: 98  duration_sec: 21

*** 2
calculate the flow from 00:00:00:00:00:02 to 00:00:00:00:00:09
packet_count: 1  byte_count: 98  duration_sec: 21

*** 2
calculate the flow from 00:00:00:00:00:02 to 00:00:00:00:00:11
packet_count: 1  byte_count: 98  duration_sec: 20

*** 2
calculate the flow from 00:00:00:00:00:03 to 00:00:00:00:00:07
packet_count: 2  byte_count: 196  duration_sec: 21
```

可以看到打印出了测量流表的信息

四、代码详情

cal.py

```
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 copy
import random
import sys
import queue
from ryu.lib.packet import arp
from ryu.lib.packet import ipv4
from ryu.lib import mac
from ryu.lib import hub


# this is topo implementing dijkstra algorithm
class Topo(object):
    def __init__(self,logger):


        self.adjacent=defaultdict(lambda s1s2:None)
        #datapathes
        self.switches=None

        self.host_mac_to={}
        self.logger=logger
        self.iperf_flows = {}
        self.iperf_flows = {(1, 2):1, (1, 3):2, (1, 4):3, (1, 5):4, (1, 6):5, (1, 7):6, (1, 8):7, (1, 9):8,
(1, 10):9, (1, 11):10,
                                      (2, 1):1, (2, 3):2, (2, 4):3, (2, 5):4, (2, 6):5, (2, 7):6,
(2, 8):7, (2, 9):8, (2, 10):9, (2, 11):10,
                                      (3, 2):1, (3, 1):2, (3, 4):3, (3, 5):4, (3, 6):5, (3, 7):6,
(3, 8):7, (3, 9):8, (3, 10):9, (3, 11):10,
                                      (4, 2):1, (4, 3):2, (4, 1):3, (4, 5):4, (4, 6):5, (4, 7):6,
(4, 8):7, (4, 9):8, (4, 10):9, (4, 11):10,
                                      (5, 2):1, (5, 3):2, (5, 4):3, (5, 1):4, (5, 6):5, (5, 7):6,
(5, 8):7, (5, 9):8, (5, 10):9, (5, 11):10,
                                      (6, 2):1, (6, 3):2, (6, 4):3, (6, 5):4, (6, 1):5, (6, 7):6,
(6, 8):7, (6, 9):8, (6, 10):9, (6, 11):10,
                                      (7, 2):1, (7, 3):2, (7, 4):3, (7, 5):4, (7, 6):5, (7, 1):6,
(7, 8):7, (7, 9):8, (7, 10):9, (7, 11):10,
                                      (8, 2):1, (8, 3):2, (8, 4):3, (8, 5):4, (8, 6):5, (8, 7):6,
(8, 1):7, (8, 9):8, (8, 10):9, (8, 11):10,
                                      (9, 2):1, (9, 3):2, (9, 4):3, (9, 5):4, (9, 6):5, (9, 7):6,
(9, 8):7, (9, 1):8, (9, 10):9, (9, 11):10,
                                      (10, 2):1, (10, 3):2, (10, 4):3, (10, 5):4, (10, 6):5, (10,
7):6, (10, 8):7,(10, 9):8,(10, 1):9, (10, 11):10}
        self.match_flag = 0
        self.cal_switches = {}
    # this is a TODO
    # not implemented
    def reset(self):
        self.adjacent=defaultdict(lambda s1s2:None)
        self.switches=None
        self.host_mac_to=None


    #helper method to fetch and modify the adjacent map
    def get_adjacent(self,s1,s2):
        return self.adjacent.get((s1,s2))

    def set_adjacent(self,s1,s2,port,weight):
        self.adjacent[(s1,s2)]=(port,weight)

    def __min_dist(self,distances, Q):
        mm=float('Inf')

        m_node=None
        for v in Q:
            if distances[v]<mm:
                mm=distances[v]
                m_node=v
        return m_node


    def shortest_path(self,src_sw,dst_sw,first_port,last_port):
        if(self.match_flag == 0):
            print(self.iperf_flows)
            print(self.best_weight_match())
            self.match_flag = 1
        distance={}
        previous = {}
        flag = 0
        assert self.switches is not None
```

```
for dpid in self.switches:
    distance[dpid]=float('Inf')
    previous[dpid]=None

distance[src_sw]=0
Q=set(self.switches)
while len(Q) > 0:
    u=self.__min_dist(distance,Q)
    if u is not None:
        Q.remove(u)
    else:
        return [dst_sw]

    for s in self.switches:


        if self.get_adjacent(u,s) is not None:
            _,weight=self.get_adjacent(u,s)
            if distance[u]+weight<distance[s]:
                distance[s]=distance[u]+weight
                previous[s] = u
                if (s == dst_sw):
                    flag=1
    # record path
    if (flag == 1):
        break
record=[]
record.append(dst_sw)
q=previous[dst_sw]

while q is not None:
    if q==src_sw:
            #we find it
        record.append(q)
        break
    p=q
    record.append(p)
```

```
            q=previous[p]



        record.reverse()

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

        record=[]
        inport=first_port

        # s1 s2; s2:s3, sn-1    sn
        for s1,s2 in zip(path[:-1],path[1:]):
            # s1--outport-->s2
            outport,_=self.get_adjacent(s1,s2)

            record.append((s1,inport,outport))
            inport,_=self.get_adjacent(s2,s1)

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



        print(record)
        return record



    def dijkstra(self, src_sw, dst_sw):
        distance={}
        previous = {}
        flag = 0
        assert self.switches is not None
        for dpid in self.switches:
            distance[dpid]=float('Inf')
    previous[dpid]=None

distance[src_sw]=0
Q=set(self.switches)
while len(Q) > 0:
    u=self.__min_dist(distance,Q)
    if u is not None:
        Q.remove(u)
    else:
        return [dst_sw]

    for s in self.switches:
        # get u->s port weight
        # for each neighbor s of u:
        if self.get_adjacent(u,s) is not None:
            _,weight=self.get_adjacent(u,s)
            if distance[u]+weight<distance[s]:
                distance[s]=distance[u]+weight
                previous[s] = u
                if (s == dst_sw):
                    flag=1
    # record path
    if (flag == 1):
        break
record=[]
record.append(dst_sw)
q=previous[dst_sw]

while q is not None:
    if q==src_sw:
            #we find it
        record.append(q)
        break
    p=q
    record.append(p)
    q=previous[p]
```

```
        record.reverse()

        if src_sw==dst_sw:
                path=[src_sw]
        else:
                path=record
        return path


    def best_weight_match(self):
        flowtable = 5
        fweightmax = 0
        flags = 0
        flagf = 0
        matchswitch = dict()
        matchflow = dict()
        matchedge = dict()
        dertaswitch = dict()
        dertaflow = dict()
        ftos = dict()
        stof = dict()
        fweight = dict()
        realflow = set()
        result = dict()
        for source, destination in self.iperf_flows:
            k = self.iperf_flows[(source, destination)]
            if (source != destination):
                if (len(self.dijkstra(source, destination)) == 1):
                    continue
                realflow.add((source, destination))
                for j in self.dijkstra(source, destination):
                    if (source, destination) not in ftos:
                        ftos[(source, destination)] = set()
                dertaflow[(source, destination)] = 0
                matchflow[(source, destination)] = 0
            if j not in stof:
                for m in range(flowtable):
                    stof[(j,m)] = set()
                    dertaswitch[(j,m)] = 0
                    matchswitch[(j, m)] = 0
            m = 0
            if (source, destination, j,m) not in fweight:
                for m in range(flowtable):
                    fweight[(source, destination, j, m)] = k
                    matchedge[(source, destination, j, m)] = 0
            for m in range(flowtable):
                ftos[(source, destination)].add((j,m))
                stof[(j,m)].add((source, destination))
if not realflow:
    return 0
i = len(self.switches)+1
m = 0
while (len(ftos) > len(stof)):#add not enough switches
    if (m == 20):
        m = 0
        i=i+1
    stof[(i, m)] = set()
    dertaswitch[(i, m)] = 0
    matchswitch[(i,m)] = 0
    m = m + 1
while (len(ftos) < len(stof)):
    if (m == 20):
        m = 0
        i=i+1
    ftos[(i, m)] = set()
    dertaflow[(i, m)] = 0
    matchflow[(i,m)] = 0
    m = m + 1
for (i, j) in stof:#match the goodpath
```

```
    for (src, dst) in ftos:
        if (i, j) not in ftos[(src,dst)]:
            ftos[(src, dst)].add((i, j))
        if (src, dst) not in stof[(i, j)]:
            stof[(i,j)].add((src,dst))
        if (src, dst, i, j) not in fweight:
            fweight[(src, dst, i, j)] = 0
            matchedge[(src, dst, i, j)] = 0
        if (fweightmax < fweight[(src, dst, i, j)]):
            fweightmax = fweight[(src, dst, i, j)]
for (i, j) in stof:
    for (src, dst) in ftos:
        fweight[(src, dst, i, j)] = fweightmax - fweight[(src, dst, i, j)]
while ((flags == 0) and (flagf == 0)):
    flags = 1
    flagf = 1
    m = self.modBFS(stof, ftos, fweight, dertaswitch, dertaflow, matchswitch, matchflow, matchedge)
    if (m == 2):
        print(realflow)
        print(result)
        return 2
    for (i, j) in stof:
        if (matchswitch[(i, j)] == 0):
            flags = 0
            break
    for (src, dst) in ftos:
        if (matchflow[(src, dst)] == 0):
            flagf = 0
            break
for (i, j) in realflow:
    for (m, n) in ftos[(i, j)]:
        if ((matchedge[(i, j, m, n)] == 1)and(m <= len(self.switches))):
            result[(i, j)] = (m, n)
print(realflow)
print(result)
```

```
        self.cal_switches = result #return the final record
        return 1

    def modBFS(self,stof,ftos,fweight,dertaswitch,dertaflow,matchswitch,matchflow,matchedge):
        q = queue.Queue()
        p = queue.Queue()
        start = set()
        S = set()
        NS = set()
        goodpath = 0
        previousswitch = dict()
        previousflow = dict()
        visitedswitch = dict()
        visitedflow = dict()
        rcfweight = dict()
        tsrc = 0
        tdst = 0
        for (src, dst, i, j) in fweight:#calculate the RC
            rcfweight[(src, dst, i, j)] = fweight[(src, dst, i, j)] - dertaflow[(src, dst)] - dertaswitch[(i, j)]
        for (i, j) in stof:
            if (matchswitch[(i, j)] == 0):
                start.add((i,j))
        for (a, b) in start:
            q.put((a, b))
            S.add((a, b))
            for (m, n) in stof:
                visitedswitch[(m,n)] = 0
            for (src, dst) in ftos:
                visitedflow[(src, dst)] = 0
            visitedswitch[(a, b)] = 1
            while not (q.empty() and p.empty()):
                while not q.empty():
                    i, j = q.get()
                    for (src, dst) in stof[(i, j)]:
                        if (visitedflow[(src,dst)] == 0):
```

```
                    if (rcfweight[(src, dst, i, j)] == 0):
                        if (matchedge[(src, dst, i, j)] == 0):
                            if (matchflow[(src, dst)] == 1):
                                p.put((src, dst))
                                NS.add((src, dst))
                                visitedflow[(src,dst)] = 1
                                previousflow[(src, dst)] = (i, j)
                            if (matchflow[(src, dst)] == 0):
                                visitedflow[(src,dst)] = 1
                                previousflow[(src, dst)] = (i, j)
                                tsrc = src
                                tdst = dst
                                goodpath = 1
                                q.queue.clear
                                p.queue.clear
                                break
        while not p.empty():
            src, dst = p.get()
            for (i, j) in ftos[(src, dst)]:
                if (visitedswitch[(i,j)] == 0):
                    if (matchedge[(src, dst, i, j)] == 1):
                        S.add((i, j))
                        q.put((i, j))
                        visitedswitch[(i, j)] = 1
                        previousswitch[(i, j)] = (src, dst)
    if (goodpath == 1):
        break
if (goodpath == 1):
    i = tsrc
    j = tdst
    m, n = previousflow[(i, j)]
    matchedge[(i, j, m, n)] = 1
    matchflow[(i, j)] = 1
    while (matchswitch[(m, n)] == 1):
        i, j = previousswitch[(m, n)]
        matchedge[(i, j, m, n)] = 0
```

```
                m, n = previousflow[(i, j)]
                matchedge[(i, j, m, n)] = 1
            matchswitch[(m, n)] = 1
        else:
            dertamin = float('Inf')
            for (src, dst, i, j) in rcfweight:
                if (rcfweight[(src, dst, i, j)] < 0):
                    return 2
            for (c, d) in S:
                for (i, j) in stof[(c, d)]:
                    if (((fweight[(i, j, c, d)] - dertaflow[(i, j)] - dertaswitch[(c, d)])<dertamin)and((fweight[(i, j, c, d)] - dertaflow[(i, j)] - dertaswitch[(c, d)])>0)):
                        dertamin = fweight[(i, j, c, d)] - dertaflow[(i, j)] - dertaswitch[(c, d)]
            for (c, d) in S:
                dertaswitch[(c, d)] = dertaswitch[(c, d)] + dertamin
            for (i, j) in NS:
                dertaflow[(i, j)] = dertaflow[(i, j)] - dertamin
            for (src, dst, i, j) in fweight:
                rcfweight[(src, dst, i, j)] = fweight[(src, dst, i, j)] - dertaflow[(src, dst)] - dertaswitch[(i, j)]
        return 1



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

    def __init__(self,*args,**kwargs):
        super(DijkstraController,self).__init__(*args,**kwargs)
        self.mac_to_port={}
        # logical switches
        self.datapaths=[]
        #ip ->mac
        self.arp_table={}

        # revser arp table
        # mac->ip
        # this is a TODO
        # not implemented
        self.rarp_table={}

        self.topo=Topo(self.logger)
        self.flood_history={}

        self.arp_history={}
        # self.is_learning={}
        self.check_thread = hub.spawn(self._send_request)

    def _send_request(self):
        while(True):
            for datapath in self.datapaths:
                if datapath is not None:
                    parser = datapath.ofproto_parser
                    req = parser.OFPFlowStatsRequest(datapath)
                    datapath.send_msg(req)
            hub.sleep(1)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):
        """
            Save flow stats reply info into self.flow_stats.
            Calculate flow speed and Save it.
        """
        body = ev.msg.body
        dpid = ev.msg.datapath.id
        for stat in sorted([flow for flow in body if flow.priority == 1000],
                           key=lambda flow: (flow.match.get('eth_src'),
                                             flow.match.get('eth_dst'))):
            key = (stat.match.get('eth_src'),    stat.match.get('eth_dst'),
                   stat.instructions[0].actions[0].port)
            value = (stat.packet_count, stat.byte_count,
                     stat.duration_sec, stat.duration_nsec)
        print("***", dpid)
        print("calculate the flow from",key[0],"to",key[1]) #TODO
        print("packet_count:", value[0], " byte_count:", value[1], " duration_sec:", value[2])
        print('')
        #print the flow table's information

def _find_dp(self,dpid):
    for dp in self.datapaths:
        if dp.id==dpid:
            return dp
    return None




#copy from example
@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)

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)

def add_best_weight_match_flow(self, dpid, eth_src, eth_dst, to_port, priority=1000):
    datapath = self._find_dp(dpid)
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    actions = [parser.OFPActionOutput(to_port)]
    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, eth_src=eth_src,
                            eth_dst=eth_dst)
    self.add_flow(datapath, priority, match, actions)

def configure_path(self,shortest_path:list,event,src_mac,dst_mac):
    #configure shortest path to switches
    msg=event.msg
    datapath=msg.datapath

    ofproto=datapath.ofproto

    parser=datapath.ofproto_parser

    # enumerate the calculated path
    # (s1,inport,outport)->(s2,inport,outport)->...->(dest_switch,inport,outport)
    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)]



        datapath=self._find_dp(int(switch))
        assert datapath is not None
        inst=[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]

        #idle and hardtimeout set to 0,making the entry permanent
        #reference openflow spec
        mod=datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath,
            match=match,
            idle_timeout=0,
            hard_timeout=0,
            priority=1,
            instructions=inst
        )
        datapath.send_msg(mod)


@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']

    #self.logger.info("From datapath {} port {} come in a packet".format(datapath.id,in_port))

    #get src_mac and dest mac
    pkt=packet.Packet(msg.data)
    eth=pkt.get_protocols(ethernet.ethernet)[0]

    # drop lldp
    if eth.ethertype==ether_types.ETH_TYPE_LLDP:
        #self.logger.info("LLDP")
        return
```

```
dst_mac=eth.dst

src_mac=eth.src

arp_pkt = pkt.get_protocol(arp.arp)

# a map recording arp table from arp request
# app_table={
#         ip:mac
# }
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 this is a ipv6 broadcast packet

if '33:33' in dst_mac[:5]:
        # the controller has not flooded this packet before
        if (src_mac,dst_mac) not in self.flood_history[dpid]:
                # we remember this packet
                self.flood_history[dpid].append((src_mac,dst_mac))
        else:
        # the controller have flooded this packet before,we do nothing and return
                return
    #self.logger.info("from dpid {} port {} packet in src_mac {} dst_mac{}".format(dpid,in_port,src_mac,dst_mac))



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

# if we have record the dest mac
# the dst mac has registered

# host_mac-> switch,inport
if dst_mac in self.topo.host_mac_to.keys():

        final_port=self.topo.host_mac_to[dst_mac][1]
        # the first switch
        src_switch=self.topo.host_mac_to[src_mac][0]
        # the final switch
        dst_switch=self.topo.host_mac_to[dst_mac][0]
        #calculate the shortest path
        shortest_path=self.topo.shortest_path(
                src_switch,
                dst_switch,
                1,
                1)
        print(shortest_path)

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

        assert len(shortest_path)>0

        #测量流表
        out_port = 0
        for key in self.topo.cal_switches:
                if key[0] == (shortest_path[0])[0] and key[1] == (shortest_path[-1])[0]:
                for s,ip,op in shortest_path:
                    if s == (self.topo.cal_switches[key])[0]:
                        out_port = op

                self.add_best_weight_match_flow((self.topo.cal_switches[key])[0],
src_mac, dst_mac, out_port, 1000)
                print((self.topo.cal_switches[key])[0], key[0], key[1], out_port)

        # log the shortest path
        path_str="

        # (s1,inport,outport)->(s2,inport,outport)->...->(dest_switch,inport,outport)
        for s,ip,op in shortest_path:
                path_str=path_str+"--{}-{}-{}--".format(ip,s,op)

        self.logger.info("The shortest path from {} to {} is
{}".format(src_mac,dst_mac,path_str))

        self.logger.info("Have calculated the shortest path from {} to
{}".format(src_mac,dst_mac))

        self.logger.info("Now configuring switches of interest")

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

        self.logger.info("Configure done")

        # current_switch=None
        out_port=None
        for s,_,op in shortest_path:
                #print(s,dpid)
                if s==dpid:
                out_port=op
        assert out_port is not None
else:
        # handle arp packet
        # in case we meet an arp packet
        if self.arp_handler(msg):    # 1:reply or drop;    0: flood
                return
        #the dst mac has not registered
        #self.logger.info("We have not learn the mac address {},flooding...".format(dst_mac))
        out_port=ofproto.OFPP_FLOOD


actions=[parser.OFPActionOutput(out_port)]

data=None

if msg.buffer_id==ofproto.OFP_NO_BUFFER:
        data=msg.data

# send the packet out to avoid packet loss
out=parser.OFPPacketOut(
        datapath=datapath,
        buffer_id=msg.buffer_id,
        in_port=in_port,
        actions=actions,
        data=data
)
datapath.send_msg(out)


@set_ev_cls(event.EventSwitchEnter)
def switch_enter_handler(self,event):
self.logger.info("A switch entered.Topology rediscovery...")
self.switch_status_handler(event)
self.logger.info('Topology rediscovery done')

@set_ev_cls(event.EventSwitchLeave)
def switch_leave_handler(self,event):
self.logger.info("A switch leaved.Topology rediscovery...")
self.switch_status_handler(event)
self.logger.info('Topology rediscovery done')



def switch_status_handler(self,event):

#api get_switch
#api app.send_request()
#api switch_request_handler
#return reply.switches
#switch.dp.id

# use copy to avoid unintended modification which is fatal to the network
all_switches=copy.copy(get_switch(self,None))



# get all datapathid
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]

# get link and get port
all_links=copy.copy(get_link(self,None))
#api link_request_handler
#api Link
# link port 1,port 2

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:
        # we would assign weight randomly
        # ignore the weight consistency
        # ie, in ryu,two links represent one physical link,
        # say s1======s2 ,in ryu we have
        # s1------>s2,s2----->s1
        # when enumerate all the links,the later one will overwrite the previous one.
        weight=random.randint(1,10)
        # weight=1
        self.topo.set_adjacent(s1,s2,p1,weight)
        self.topo.set_adjacent(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:
        # target ip
        arp_dst_ip = arp_pkt.dst_ip
```

```
            # arp_history={
            # (datapath.id,eth_src,dest_ip):inport
            # }

            # we have met this particular arp request before
            if (datapath.id, eth_src, arp_dst_ip) in self.arp_history:
                #(datapath.id,eth_src,target_ip)->inport
                # however, the new arp packet did not consist with the record, it comes from another port, so may be it's a broadcasted arp request
                # we just ignore it to break the broadcast loop
                if self.arp_history[(datapath.id, eth_src, arp_dst_ip)] != in_port:
                    #datapath.send_packet_out(in_port=in_port, actions=[])
                    return True
            else:
                # we didnt met this packet before, record
                self.arp_history[(datapath.id, eth_src, arp_dst_ip)] = in_port

        #construct arp packet
        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

            # arp_request
            if opcode == arp.ARP_REQUEST:
                self.logger.info("ARP Request src_ip: {}".format(arp_src_ip))
                # we have learned the target ip mac mapping
                if arp_dst_ip in self.arp_table:
                    # send arp reply from in port
                    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()
                    #arp reply
                    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
```

match.py

```
import random
TOLERANCE = 1e-6

def improveLabels(val):
    #to confirm a GoodSet
    for u in S:
        lu[u] -= val
    for v in V:
        if v in T:
            lv[v] += val
        else:
            minSlack[v][0] -= val

def improveMatching(v):
    #to find a GoodPath
    u = T[v]
    if u in Mu:
        improveMatching(Mu[u])
    Mu[u] = v
    Mv[v] = u

def slack(u,v): return lu[u]+lv[v]-w[u][v]
    #Reduced Cost

def augment():
    while True:
        ((val, u), v) = min([(minSlack[v], v) for v in V if v not in T])
        assert u in S
        assert val > - TOLERANCE
        if val > TOLERANCE:
            improveLabels(val)
        # now we are sure that (u,v) is saturated
        assert abs(slack(u,v)) < TOLERANCE
        T[v] = u
        if v in Mv:
            u1 = Mv[v]
            assert not u1 in S
            S[u1] = True
            for v in V:
                if not v in T and minSlack[v][0] > slack(u1,v):
                    minSlack[v] = [slack(u1,v), u1]
        else:
            improveMatching(v)
            return

def maxWeightMatching(weights):
    #input the weight Matrix
    global U,V,S,T,Mu,Mv,lu,lv, minSlack, w
    w    = weights
    n    = len(w)
    U    = V = range(n)
    lu = [ max([w[u][v] for v in V]) for u in U]   # start with trivial labels
    lv = [ 0                             for v in V]
    Mu = {}                                                  # start with empty matching
    Mv = {}
    while len(Mu)<n:
        free = [u for u in V if u not in Mu]          # choose free vertex u0
        u0 = free[0]
        S = {u0: True}                                # grow tree from u0 on
        T = {}
        minSlack = [[slack(u0,v), u0] for v in V]
        augment()
    val = sum(lu)+sum(lv)
    return (Mu, Mv, val)

if __name__=='__main__':
    #define the num
    switches_num = 20
    measure_num = 20
    flow_num = 1000

    #define randomly the weight matrix
    weight_matrix = [[0 for v in range(flow_num)] for u in range(flow_num)]

    for i in range(switches_num):
        for j in range(flow_num):
            w = random.randint(0,10)
            for k in range(measure_num):
                weight_matrix[j][k + i * measure_num] = w

    match = maxWeightMatching(weight_matrix)
    flows = match[0]
```

```
for key in flows:
        if weight_matrix[key][flows[key]] != 0:
                print(key, ': ', flows[key], end = '')

print('')
print("the final cost is: ", match[2])
```