南京大学本科生实验报告

课程名称：**计算机网络** 任课教师：田臣/李文中 助教：

学院	计算机	专业（方向）	计算机
学号	181860077	姓名	佘帅杰
Email	3121416933@qq.com	开始完成日期	2020.3.18

实验名称

计算机网络试验2

实验目的

实现交换机以及各种机制

实验内容

Python基本语法使用

交换机的工作机制

实验结果与核心代码解读（合并了原模板的两个模块）

PS：下面结合代码进行实现思路的介绍（代码是摘录重点为了展示，在缩进代码块上和源代码不一定一样）

Task2

实现思维简述

这个任务是打算实现一个基本的交换机。

考虑到交换机的功能逻辑，有两个重点问题。1：对于广播的地址特殊处理，2：对已知地址和端口对应存储，为发送做准备，以下结合代码说明实现思路

初始定义所需的变量

```
boardcast_addr="ff:ff:ff:ff:ff:ff"
list_port=[]
list_address=[]
```

对广播地址处理，先对来源的数据对（端口和地址）进行处理，看看是否在列表里，然后决定增加与否，最后洪泛

```
if packet[0].dst == boardcast_addr:
   if input_port in list_port:
       print("boardcast src port in list")
   else:
        print("boardcast src port not in list")
        list_port.append(input_port)
        list_address.append(packet[0].src)
   for intf in my_interfaces:
        if input_port != intf.name:
            log_debug ("Flooding packet {} to {}".format(packet,
intf.name))
            net.send_packet(intf.name, packet)
            log_info ("boardcast packet from {} to {}".format(input_port,
intf.name))
```

对非广播地址处理，首先看看来源的数据对（端口和地址)是否存在

```
if input_port in list_port:
   print("src port in list")
else:
   print("src port not in list")
   list_port.append(input_port)
   list_address.append(packet[0].src)
```

接着就是找一找有没有地址的保存记录，有就发，没有就算了，有标志位send_flag指示

```
index=0
send_flag=0
for address in list_address:
    if address==packet[0].dst:
        net.send_packet(list_port[index], packet)
        log_info ("find port {} of address
{}".format(list_port[index],packet[0].dst))
        send_flag=1
        break
    index+=1
```

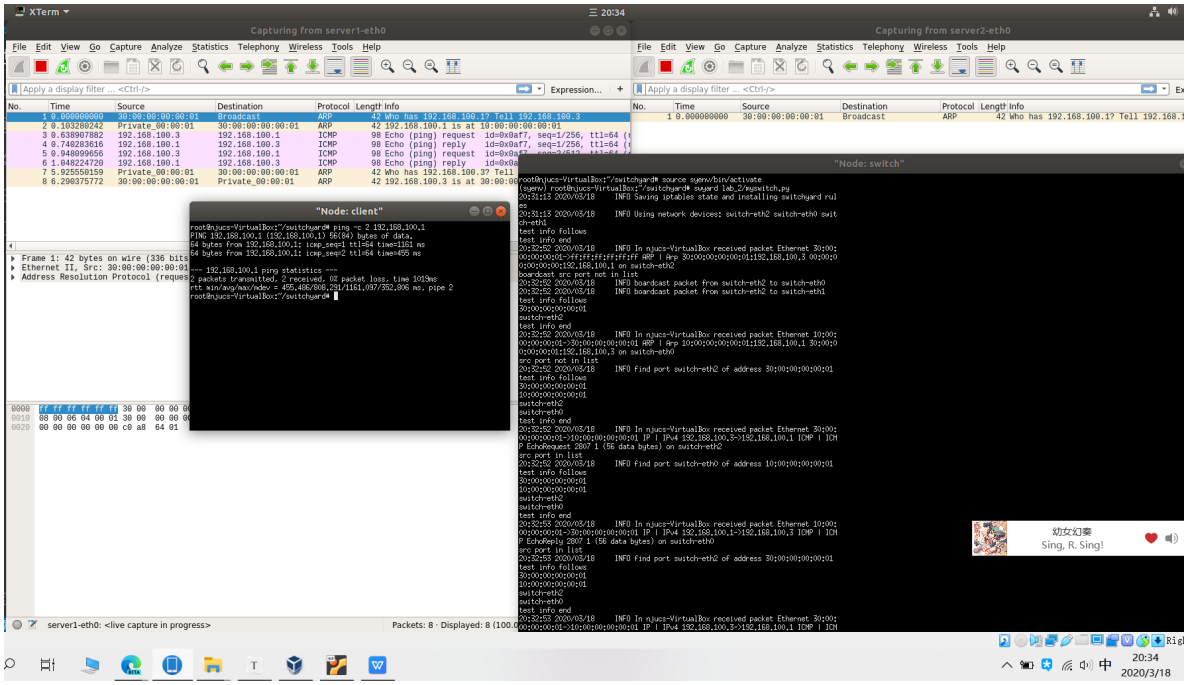
对于上面没有的情况进行处理

```
if send_flag!=1:
    print("didn't find port")
    for intf in my_interfaces:
        if input_port != intf.name:
            log_debug ("Flooding packet {} to {}".format(packet, intf.name))

            net.send_packet(intf.name, packet)
            log_info ("didnt find packet from {} to {}".format(input_port,
intf.name))
```

测试结果展示

如试验手册中介绍的方法进行测试，实现之后利用client去ping server1发送两个request

理论而言只有server1收到request以及回送两个reply，server2只有Arp包，如下证明了正确性



Task3

实现思维简述

Task3就是引入了超时，所以实现有所不同，根据流程图实现如下

定义一些所需的类和变量

```
import time
class info:
    def __init__(self,create_times,ports,addresses):
        self.create_time=create_times
        self.port=ports
        self.address=addresses
boardcast_addr="ff:ff:ff:ff:ff:ff"
list_info=[]
```

在最开始检查并移出超时项（每一项有创建时间，只要获取当前的时间戳进行计算即可）

这里是在一收到包就进行检查

```
index=0
while index<len(list_info):
    temp_time=time.time()
    if temp_time-list_info[index].create_time>=10:
        del(list_info[index])
    else:
        index+=1
```

同样的先对广播的地址进行处理

先检查源地址和端口的对应情况，如果不对就刷新端口重置事件

如果是没有对应那就创建对应的数据，然后放在列表里

最后就是一个洪泛的模块了

```
if packet[0].dst == boardcast_addr:
```

```
    flag=0
    for i in list_info:
        if i.address== packet[0].src:
            flag=1
            if i.port != input_port:
                i.port=input_port
                i.create_time=time.time()
            break
    if flag==0:
        cur_time=time.time()
        temp=info(cur_time,input_port,packet[0].src)
        list_info.append(temp)

    for intf in my_interfaces:
        if input_port != intf.name:
            log_debug ("Flooding packet {} to {}".format(packet,
intf.name))
            net.send_packet(intf.name, packet)
            log_info ("boardcast packet from {} to {}".format(input_port,
intf.name))
```

然后就是非广播模式了

同样的是对源数据和端口的情况进行分析，原理和实现和上面一模一样

```
flag=0
for i in list_info:
    if i.address== packet[0].src:
        flag=1
        if i.port != input_port:
             i.port=input_port
             i.create_time=time.time()
             break
if flag==0:
    cur_time=time.time()
    temp=info(cur_time,input_port,packet[0].src)
    list_info.append(temp)
```

然后就是对发出的数据端口进行分析了

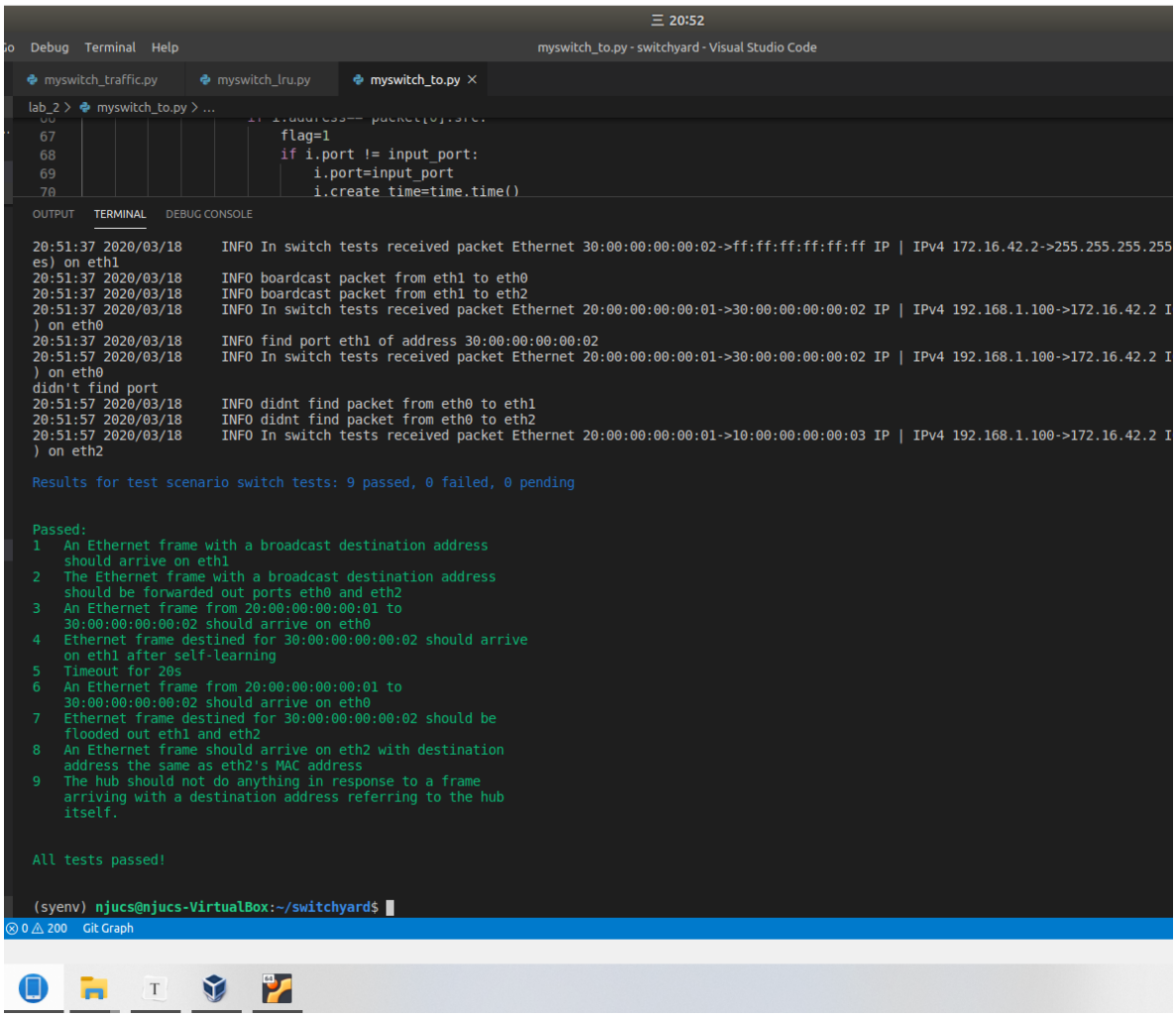
其实就是有匹配就发送，一个都不匹配就和广播一样了。

```
send_flag=0
for i in list_info:
    if i.address==packet[0].dst:
        net.send_packet(i.port, packet)
        log_info ("find port {} of address
{}".format(i.port,packet[0].dst))
        send_flag=1
        break

if send_flag!=1:
    print("didn't find port")
    for intf in my_interfaces:
        if input_port != intf.name:
           log_debug ("Flooding packet {} to {}".format(packet, intf.name))
```

```
14              net.send_packet(intf.name, packet)
15              log_info ("didnt find packet from {} to {}".format(input_port,
intf.name))
```

运行结果

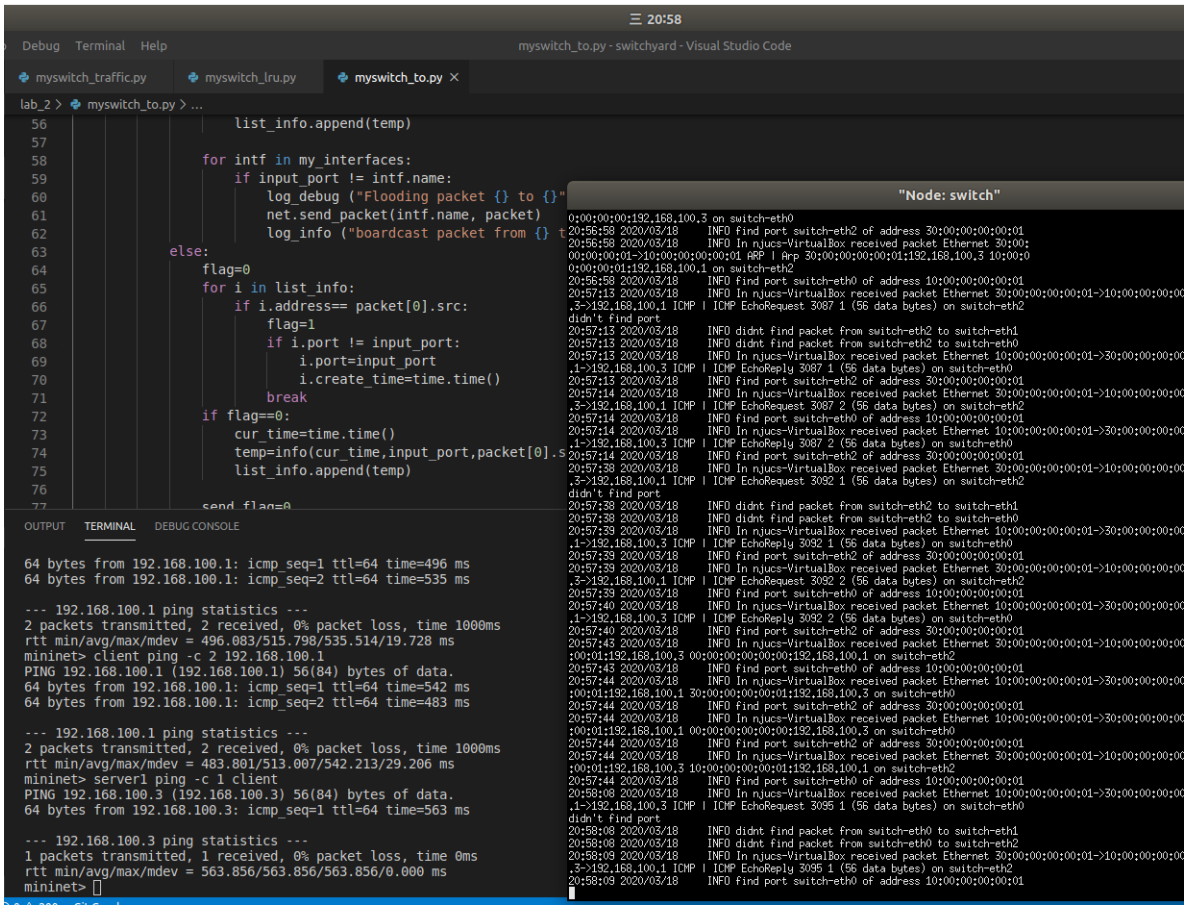
VirtualBox



Deploy测试

在mininet里用client和server1互相ping，并根据输出语句查看功能是否可用

server1 ping client 后，理论上已经保存了server1的端口和地址数据，隔上一段时间后再client ping server1显示didn`t find port 的提示语句。



Task4

实现思路

定义所需的变量，和上面比起来不需要时间了，但是有一个列表大小的限制，然后这个列表类似于队列，下标小的是队头，出队的位置。队尾则相反。

```
class info:
    def __init__(self,ports,addresses):
        self.port=ports
        self.address=addresses
boardcast_addr="ff:ff:ff:ff:ff:ff"
max_len=5
list_info=[]
```

然后同理是对广播地址的处理

一样是先看看源地址，是不是在列表里，如果是检查端口是否匹配，如果不匹配就更新端口

然后如果没有匹配上，那么就要加入列表，按照定义加在最后面，如果已经满了那就把前面全都移动一格，否则就直接加在后面

最后一步就是洪泛，没什么好说的，用了很多次了

```
if packet[0].dst == boardcast_addr:
    flag=0
    for i in range(len(list_info)):
        if list_info[i].address== packet[0].src:
            flag=1
            if list_info[i].port != input_port:
                list_info[i].port = input_port
```

```
            break
    if flag==0:
        if len(list_info) >=max_len:
            for i in range(1,len(list_info)):
                list_info[i-1]=list_info[i]
            temp=info(input_port,packet[0].src)
            list_info[len(list_info)-1]=temp
        else:
            temp=info(input_port,packet[0].src)
            list_info.append(temp)
    for intf in my_interfaces:
        if input_port != intf.name:
             log_debug ("Flooding packet {} to {}".format(packet,
intf.name))
             net.send_packet(intf.name, packet)
             log_info ("boardcast packet from {} to {}".format(input_port,
intf.name))
```

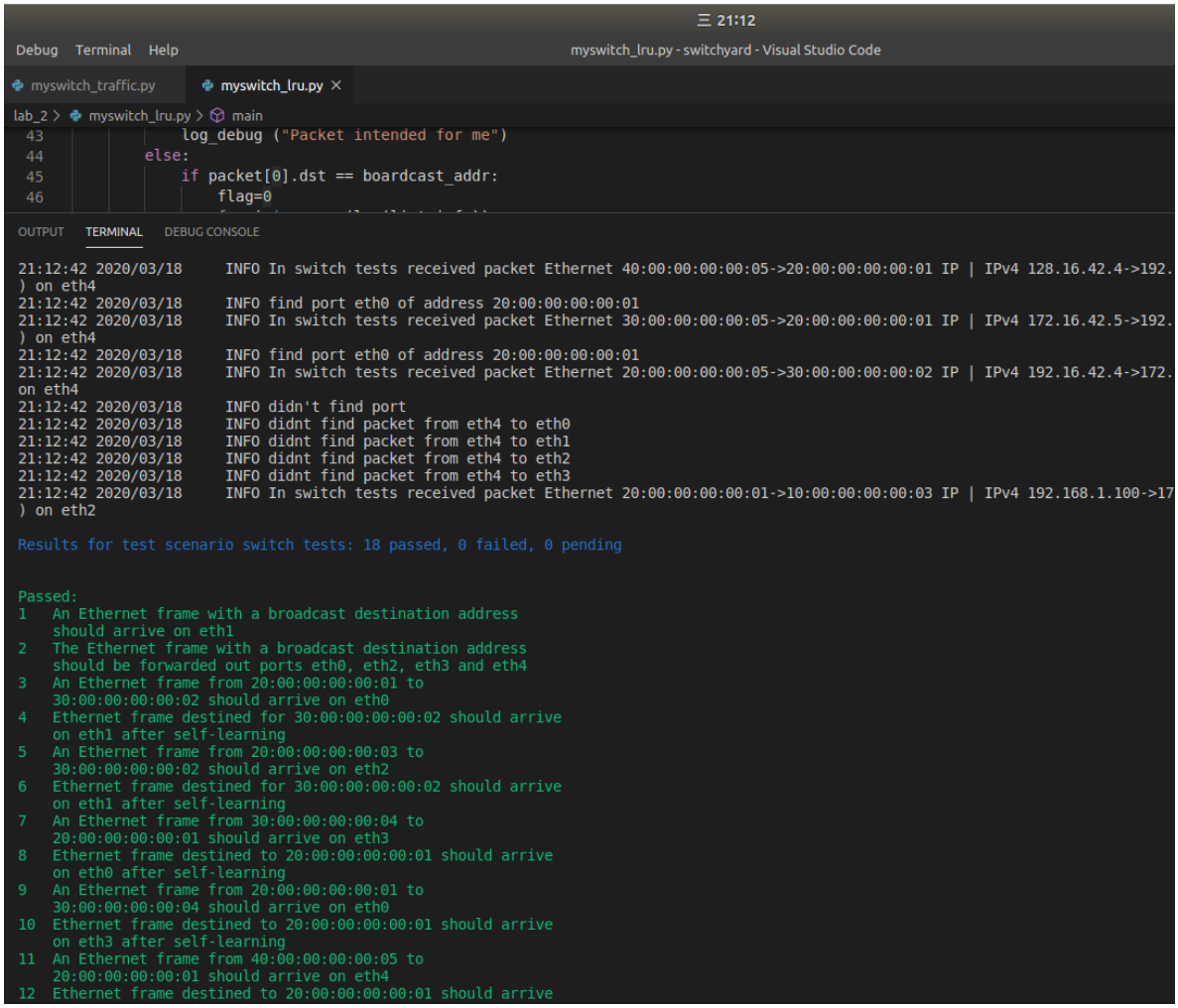
然后就是非广播的处理了，对于源数据的处理和上面是一样的，然后就是对发出的数据的处理

其实差不多，匹配成功就放到最后面，实现和源数据处理差不多，不成功就广播

```
flag=0
for i in range(len(list_info)):
     if list_info[i].address== packet[0].src:
          flag=1
          if list_info[i].port != input_port:
              list_info[i].port = input_port
           break
if flag==0:
    if len(list_info) >= max_len:
        for i in range(1,len(list_info)):
             list_info[i-1]=list_info[i]
             temp=info(input_port,packet[0].src)
             list_info[len(list_info)-1]=temp
    else:
         temp=info(input_port,packet[0].src)
         list_info.append(temp)


send_flag=0
for i in range(len(list_info)):
    if list_info[i].address == packet[0].dst:
         net.send_packet(list_info[i].port, packet)
         log_info ("find port {} of address
{}".format(list_info[i].port,packet[0].dst))
         send_flag=1
         temp_info = list_info[i]
         del(list_info[i])
         list_info.append(temp_info)
         break


if send_flag!=1:
     log_info("didn't find port")
     for intf in my_interfaces:
         if input_port != intf.name:
```

```
35                log_debug ("Flooding packet {} to {}".format(packet,
intf.name))
36                net.send_packet(intf.name, packet)
37                log_info ("didnt find packet from {} to
{}".format(input_port, intf.name))
```

测试结果



Deploy测试

测试代码

```
print("test info start")
for i in list_info:
    print("add",i.address)
    print("por",i.port)
print("test info end")
```

测试思路：由于每一次ping都是双向的（request和reply），就先client ping server1这个时候已经有了两个端口数据了（由于测试的需求把maxsize改为2了）如下

可以看到有10和30两个地址和端口了（30是client，10是server1）

```
21:25:36 2020/03/18     INFO find port switch-eth2 of address 30:00:00:00:00:01
test info start
add 10:00:00:00:00:01
por switch-eth0
add 30:00:00:00:00:01
por switch-eth2
test info end
21:25:42 2020/03/18     INFO In njucs-VirtualBox received packet Ethernet 10:00:00:00:00:01->30:0
0:00:00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.100.3 on switch
-eth0
21:25:42 2020/03/18     INFO find port switch-eth2 of address 30:00:00:00:00:01
test info start
add 10:00:00:00:00:01
por switch-eth0
add 30:00:00:00:00:01
por switch-eth2
test info end
21:25:42 2020/03/18     INFO In njucs-VirtualBox received packet Ethernet 30:00:00:00:00:01->10:0
0:00:00:00:00:01 ARP | Arp 30:00:00:00:00:01:192.168.100.3 10:00:00:00:00:01:192.168.100.1 on switch
-eth2
21:25:42 2020/03/18     INFO find port switch-eth0 of address 10:00:00:00:00:01
test info start
add 30:00:00:00:00:01
por switch-eth2
add 10:00:00:00:00:01
por switch-eth0
test info end
```

然后server1 ping -c 1 server2，如下

可以看到端口换了是10和20了，原因就是30被挤出去了（根据LRU）

```
21:26:09 2020/03/18     INFO find port switch-eth0 of address 10:00:00:00:00:01
test info start
add 20:00:00:00:00:01
por switch-eth1
add 10:00:00:00:00:01
por switch-eth0
test info end
21:26:09 2020/03/18     INFO In njucs-VirtualBox received packet Ethernet 20:00:00:00:00:01->10:0
0:00:00:00:00:01 ARP | Arp 20:00:00:00:00:01:192.168.100.2 00:00:00:00:00:00:192.168.100.1 on switch
-eth1
21:26:09 2020/03/18     INFO find port switch-eth0 of address 10:00:00:00:00:01
test info start
add 20:00:00:00:00:01
por switch-eth1
add 10:00:00:00:00:01
por switch-eth0
test info end
21:26:09 2020/03/18     INFO In njucs-VirtualBox received packet Ethernet 10:00:00:00:00:01->20:0
0:00:00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 20:00:00:00:00:01:192.168.100.2 on switch
-eth0
21:26:09 2020/03/18     INFO find port switch-eth1 of address 20:00:00:00:00:01
```

Task5

基本实现思路

基本思路和上一个任务差不多，代码也是抄过来改一改

同样定义需要的变量

```
class info:
    def __init__(self,ports,addresses,vals):
        self.port=ports
        self.address=addresses
        self.val=vals

boardcast_addr="ff:ff:ff:ff:ff:ff"
max_len=5
list_info=[]
```

对广播模式进行分析，同理先分析源数据的地址和端口，发现不对的就校对，发现没有就加入。

如果原来的满了就循环找到最小进行删除，反之就直接加入

最后进行洪泛

```
if packet[0].dst == boardcast_addr:
    flag=0
    for i in range(len(list_info)):
        if list_info[i].address== packet[0].src:
             flag=1
             if list_info[i].port != input_port:
                  list_info[i].port = input_port
             break
     if flag==0:
         if len(list_info) >=max_len:
              min_pos=0
              min_data=10000
              for i in range(len(list_info)):
                  if list_info[i].val < min_data:
                       min_pos=i
                       min_data=list_info[i].val
              del(list_info[min_pos])
              temp=info(input_port,packet[0].src,0)
              list_info.append(temp)
          else:
              temp=info(input_port,packet[0].src,0)
              list_info.append(temp)
     for intf in my_interfaces:
        if input_port != intf.name:
           log_debug ("Flooding packet {} to {}".format(packet, intf.name))
            net.send_packet(intf.name, packet)

```

然后对非广播模式进行分析

对于源数据的分析和上面完全一致

```
if packet[0].dst == boardcast_addr:
    flag=0
    for i in range(len(list_info)):
        if list_info[i].address== packet[0].src:
             flag=1
             if list_info[i].port != input_port:
                  list_info[i].port = input_port
             break
     if flag==0:
```

```
        if len(list_info) >=max_len:
             min_pos=0
             min_data=10000
             for i in range(len(list_info)):
                 if list_info[i].val < min_data:
                      min_pos=i
                      min_data=list_info[i].val
             del(list_info[min_pos])
             temp=info(input_port,packet[0].src,0)
             list_info.append(temp)
         else:
             temp=info(input_port,packet[0].src,0)
             list_info.append(temp)
```

然后是发送逻辑的实现

就是找端口匹配，如果有那就匹配发送，流量+=1，反之就洪泛就完事了

```
send_flag=0
for i in range(len(list_info)):
    if list_info[i].address == packet[0].dst:
        net.send_packet(list_info[i].port, packet)
        log_info ("find port {} of address
{}".format(list_info[i].port,packet[0].dst))
        send_flag=1
        list_info[i].val+=1
        break

if send_flag!=1:
    log_info ("didn't find port")
    for intf in my_interfaces:
         if input_port != intf.name:
            log_debug ("Flooding packet {} to {}".format(packet,
intf.name))
            net.send_packet(intf.name, packet)
            log_info ("didnt find packet from {} to {}".format(input_port,
intf.name))


```

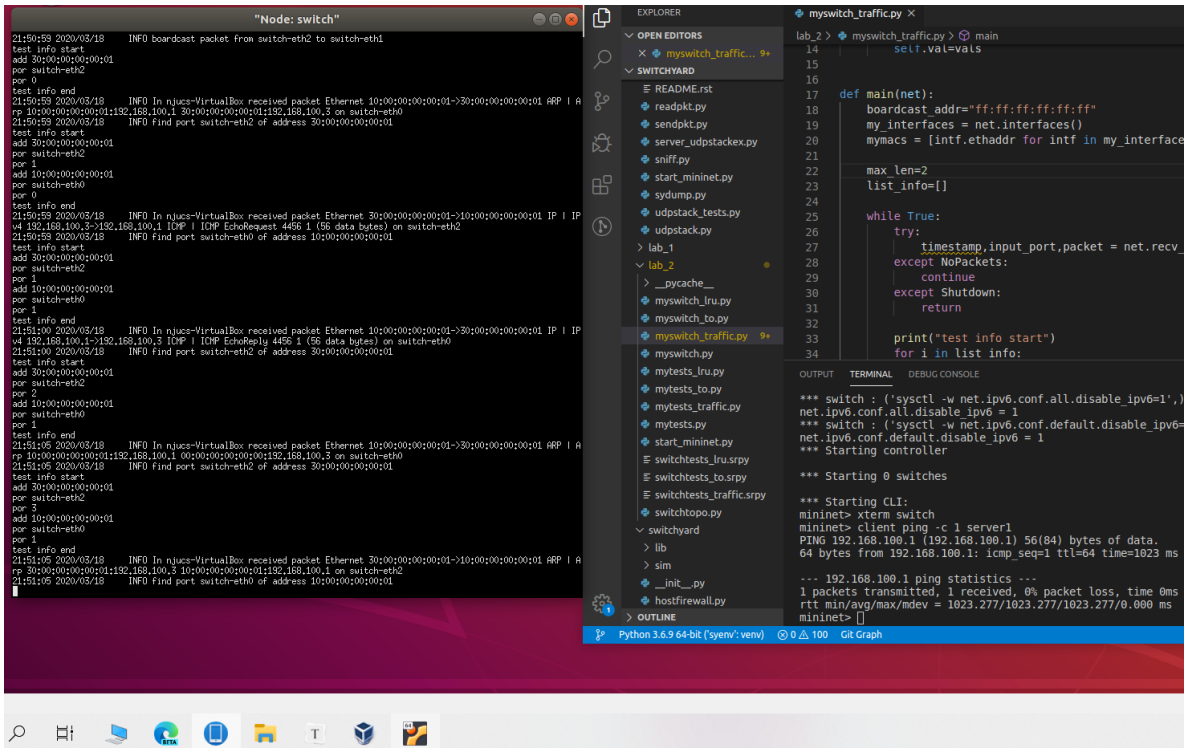
测试结果

```
15
16
17  def main(net):
18      boardcast_addr="ff:ff:ff:ff:ff:ff"
19      my_interfaces = net.interfaces()

OUTPUT  TERMINAL  DEBUG CONSOLE

21:54:57 2020/03/18     INFO didn't find port
21:54:57 2020/03/18     INFO didnt find packet from eth2 to eth0
21:54:57 2020/03/18     INFO didnt find packet from eth2 to eth1
test info start
add 30:00:00:00:00:02
por eth1
por 1
add 20:00:00:00:00:03
por eth2
por 0
test info end
21:54:57 2020/03/18     INFO In switch tests received packet Ethernet 20:00:00:00:00:01->10:00:00:00:00:03 IP | IPv4 192.168.1.100->172
) on eth2

Results for test scenario switch tests: 8 passed, 0 failed, 0 pending


Passed:
1   An Ethernet frame with a broadcast destination address
    should arrive on eth1
2   The Ethernet frame with a broadcast destination address
    should be forwarded out ports eth0 and eth2
3   An Ethernet frame from 20:00:00:00:00:01 to
    30:00:00:00:00:02 should arrive on eth0
4   Ethernet frame destined for 30:00:00:00:00:02 should arrive
    on eth1 after self-learning
5   An Ethernet frame from 20:00:00:00:00:03 to
    30:00:00:00:00:03 should arrive on eth2
6   Ethernet frame destined for 30:00:00:00:00:03 should be
    flooded on eth0 and eth1
7   An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
8   The switch should not do anything in response to a frame
    arriving with a destination address referring to the switch
    itself.


All tests passed!


(syenv) njucs@njucs-VirtualBox:~/switchyard$
```

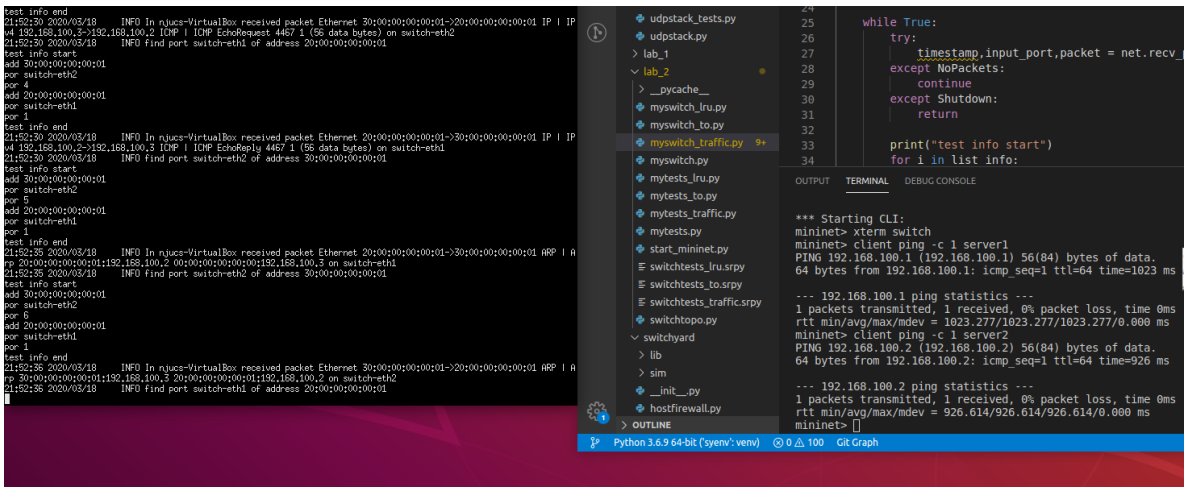
Deploy 测试

测试思路：在代码的前面加上和上一个任务差不多的测试代码，然后改小了maxsize，然后client ping -c 1 server1。这个时候应该是有client和server1的端口了，然后再client ping -c 1 server2 查看结果如下

第一个ping



第二个ping



可以看到client端口的数据量累加了，server1的端口在列表里的位置被server2的端口替代，且数据包的数目一致

总结与感想

总结：结合上课的知识，按照试验手册的流程图完成交换机的策略实现。很重要的一点就是多使用log_info进行快速的debug。同时注意出错信息即可

感想：用markdown再转pdf方便不少