Lab4_Report

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	2020.4.22

实验名称

计算机网络试验4

实验目的

升级路由器，完善功能

实验内容

router进阶功能：对子网的识别以及发送arp request以及转发包

实验结果与核心代码解读（合并了原模板的两个模块）

注：直接合在一起分析了

代码解读

数据结构

用的到的数据结构都在下面

Node是表示的一个Forward_Table表项

具有prefix（前缀），mask（掩码），nexthop（下一跳地址），name（端口名）

Pac表示的待处理队列的表项

具有pac（待发的包），ci（次数，本来想用time的，怕和时间弄混，用了拼音。。。），time（时间，收到的时间，准备做超时处理），th（匹配到的表项）

Router里用的数据需要说明的就是arp_table是lab3里就有的arp-mac对应表。forwardtable是新来的

```
class Node():
    def __init__(self,p,m,n,i):
        self.prefix=p
        self.mask=m
        self.nexthop=n
        self.name=i
#swyard -t routertests2.srpy myrouter.py

class Pac():
    def __init__(self,pac,thing):
        self.pac=pac
        self.ci=0
        self.time=0
        self.th=thing

class Router(object):
    def __init__(self, net):
        self.net = net
        # other initialization stuff here
        self.interfaces = net.interfaces()
        self.ip_list=[intf.ipaddr for intf in self.interfaces]
        self.mac_list=[intf.ethaddr for intf in self.interfaces]
        self.arp_table={}
        self.forward_table=[]
```

处理逻辑

初始化逻辑

变量的声明略

第一个循环就是基于路由器做forward建立

第二个是读文件建立

其他的说明见注释（为了简洁，我删除了源代码里的调试用的一些注释等信息，只留代码）

```
def __init__(self, net):
        self.net = net
        # other initialization stuff here
        self.interfaces = net.interfaces()
        self.ip_list=[intf.ipaddr for intf in self.interfaces]
        self.mac_list=[intf.ethaddr for intf in self.interfaces]
        self.arp_table={}
        self.forward_table=[]
        for i in self.interfaces:
            prefix=ipaddress.ip_network(str(i.ipaddr)+"/"+str(i.netmask), strict=False)
            #利用这个方法会返回前缀和掩码
            #然后利用字符串处理就可以丢掉后面的掩码就只有前缀了
            prefix=str(prefix)
            if '/' in  prefix:
                prefix=prefix.split("/")
                prefix=prefix[0]


            #利用处理好的前缀字符串重新构造前缀地址
            prefix=IPv4Address(prefix)
            #然后建立表项，加入列表
            a=Node(prefix,i.netmask,None,i.name)
            self.forward_table.append(a)

        file = open("forwarding_table.txt")
        while 1:
            line = file.readline()
            if not line:
                break
            else:
                #打开文件的操作，去除末尾的换行符，用空格分割数据
                line=line.strip('\n')
                d=line.split(" ")
                #同理构造表项
                a=Node(IPv4Address(d[0]),IPv4Address(d[1]),IPv4Address(d[2]),d[3])
                self.forward_table.append(a)
        for a in self.forward_table:
            print(a.prefix," ",a.mask," ",a.nexthop," ",a.name)

```

核心处理逻辑

队列处理逻辑

见注释

```
q = []
        while True:
            if len(q)!=0:
                #判断队列不空
                for i in self.interfaces:
                    if i.name==q[0].th.name:
                        port=i
                #用端口名找到端口
                #若下一跳是空：也就是基于路由器构造的表项则用目的地址作为目的
                if q[0].th.nexthop is None:
                    targetip=q[0].pac[IPv4].dst
                else:
                    targetip=q[0].th.nexthop
                find_flag2=0
                #遍历本地mac-arp是否有匹配
                for (k,v) in  self.arp_table.items():
                    if targetip == k:
                        #找到了，修改以太网头部发送
                        q[0].pac[Ethernet].dst=v
                        q[0].pac[Ethernet].src=port.ethaddr
                        print("send pac (find) ",port)
                        self.net.send_packet(port,q[0].pac)
                        find_flag2=1
                        del(q[0])
                        break
                #很不幸的没有找到
                if find_flag2 ==0:
                    #是不是发5次了？
                    if q[0].ci >= 5:
                        del(q[0])
                    else:
                        #是不是刚刚进来的？或者是已经1s了
                        cur=time.time()
                        if (q[0].ci==0) or (cur-q[0].time)>1:
                            ether = Ethernet()
                            ether.src = port.ethaddr
                            ether.dst = 'ff:ff:ff:ff:ff:ff'
                            #构造查询包
                            ether.ethertype = EtherType.ARP
                            arp = Arp(operation=ArpOperation.Request,senderhwaddr=port.ethaddr,senderprotoaddr=port.ipaddr,targethwaddr='ff:ff:ff:ff:ff:ff',targetprotoaddr=targetip)
                            arppacket = ether + arp
                            print("send requests",port)
                            self.net.send_packet(port, arppacket)
                            #次数修改，刷新发送时间
                            q[0].ci+=1
                            q[0].time=time.time()
                            print(q[0].time)
```

收包处理逻辑

见注释

```
if pkt.has_header(IPv4):
    #确认是否是ipv4包
    head=(pkt[IPv4])
    if head is None:
        #之前不知道TTL的报错，以为是自己错了，所以设置了这个
        print("error")
    head.ttl-=1
    #ttl处理
    print("ipv4",head)
    pos=-1
    prefixlen=-1
    index=0
    for i in self.forward_table:
        #判断前缀匹配
        if ((int(head.dst)&int(i.mask))==int(i.prefix)):
             netaddr = IPv4Network(str(i.prefix)+"/"+str(i.mask))
             #构造地址求前缀长，“最长前缀匹配”
             if netaddr.prefixlen > prefixlen:
                 prefixlen=netaddr.prefixlen
                 pos=index
                 #print("Match")
         index+=1
     print("add packet to queue")
     if pos == -1:
          #发现没有匹配，报错。测试样例有这个
          print("No Match Some Error occur!!!!!!!!!!!!!!!!")
     else:
        #匹配了，那就放进处理队列
          q.append(Pac(pkt,self.forward_table[pos]))
```

最后一块是关于arp处理的，之前lab3写的没啥问题，没有修改，就不再单独展示了

测试

测试样例

分析

简单的基于print的信息进行分析处理例程（以下非粗体内容复制于terminal）

```
15:31:05 2020/04/22 INFO Got a packet: Ethernet 20:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4
192.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 42 (0 data bytes) ipv4 IPv4
192.168.1.100->172.16.42.2 ICMP add packet to queue
```

收到了ipv4包，加入待处理队列

```
15:31:05 2020/04/22 INFO Not arp Packet 15:31:05 2020/04/22 INFO Table Shown as follows

send requests router-eth2 mac:10:00:00:00:00:03 ip:172.16.42.1/30

1587540665.9747787
```

发现没有对应的mac和arp表，于是发送request进行查询

```
15:31:05 2020/04/22 INFO Got a packet: Ethernet 30:00:00:00:00:01->10:00:00:00:00:03 ARP | Arp
30:00:00:00:00:01:172.16.42.2 10:00:00:00:00:03:172.16.42.1 15:31:05 2020/04/22 INFO operation
kind ArpOperation.Reply 15:31:05 2020/04/22 INFO recive arp reply 15:31:05 2020/04/22 INFO
Table Shown as follows 172.16.42.2 30:00:00:00:00:01 172.16.42.1 10:00:00:00:00:03 send pac
(find) router-eth2 mac:10:00:00:00:00:03 ip:172.16.42.1/30
```

收到了reply，如lab3处理，之后处理例程就可以发送原ipv4包了

```
15:31:05 2020/04/22 INFO Got a packet: Ethernet 30:00:00:00:00:01->10:00:00:00:00:03 IP | IPv4
172.16.42.2->192.168.1.100 ICMP | ICMP EchoReply 0 42 (0 data bytes) ipv4 IPv4 172.16.42.2-
>192.168.1.100 ICMP add packet to queue
```

收到了ipv4包，加入待处理队列

```
15:31:05 2020/04/22 INFO Not arp Packet 15:31:05 2020/04/22 INFO Table Shown as follows
172.16.42.2 30:00:00:00:00:01 172.16.42.1 10:00:00:00:00:03 send requests router-eth0
mac:10:00:00:00:00:01 ip:192.168.1.1/24 1587540665.9774284
```

发现没有对应的mac和arp表，于是发送request进行查询

```
15:31:05 2020/04/22 INFO Got a packet: Ethernet 20:00:00:00:00:01->10:00:00:00:00:01 ARP | Arp
20:00:00:00:00:01:192.168.1.100 10:00:00:00:00:01:192.168.1.1 15:31:05 2020/04/22 INFO
operation kind ArpOperation.Reply 15:31:05 2020/04/22 INFO recive arp reply 15:31:05
2020/04/22 INFO Table Shown as follows 172.16.42.2 30:00:00:00:00:01 172.16.42.1
10:00:00:00:00:03 192.168.1.100 20:00:00:00:00:01 192.168.1.1 10:00:00:00:00:01 send pac (find)
router-eth0 mac:10:00:00:00:00:01 ip:192.168.1.1/24
```

收到了reply，如lab3处理，之后处理例程就可以发送原ipv4包了

```
15:31:05 2020/04/22 INFO Got a packet: Ethernet 20:00:00:00:00:01->10:00:00:00:00:01 IP | IPv4
192.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 42 (0 data bytes) ipv4 IPv4
192.168.1.100->172.16.42.2 ICMP add packet to queue
```

收到了ipv4包，加入待处理队列

```
15:31:05 2020/04/22 INFO Not arp Packet 15:31:05 2020/04/22 INFO Table Shown as follows
172.16.42.2 30:00:00:00:00:01 172.16.42.1 10:00:00:00:00:03 192.168.1.100 20:00:00:00:00:01
192.168.1.1 10:00:00:00:00:01 send pac (find) router-eth2 mac:10:00:00:00:00:03
ip:172.16.42.1/30
```

这里收到的包已经有缓存了，直接发送

```
15:31:05 2020/04/22 INFO Got a packet: Ethernet 30:00:00:00:00:01->10:00:00:00:00:03 IP | IPv4
172.16.42.2->192.168.1.100 ICMP | ICMP EchoReply 0 42 (0 data bytes) ipv4 IPv4 172.16.42.2-
>192.168.1.100 ICMP add packet to queue
```

收到了ipv4包，加入待处理队列

```
15:31:05 2020/04/22 INFO Not arp Packet 15:31:05 2020/04/22 INFO Table Shown as follows
172.16.42.2 30:00:00:00:00:01 172.16.42.1 10:00:00:00:00:03 192.168.1.100 20:00:00:00:00:01
192.168.1.1 10:00:00:00:00:01 send pac (find) router-eth0 mac:10:00:00:00:00:01
ip:192.168.1.1/24
```

这里收到的包已经有缓存了，直接发送

```
15:31:05 2020/04/22 INFO Got a packet: Ethernet 40:00:00:00:00:11->10:00:00:00:00:03 IP | IPv4
10.100.1.55->172.16.64.35 ICMP | ICMP EchoRequest 0 42 (0 data bytes) ipv4 IPv4 10.100.1.55-
>172.16.64.35 ICMP add packet to queue 15:31:05 2020/04/22 INFO Not arp Packet 15:31:05
2020/04/22 INFO Table Shown as follows 172.16.42.2 30:00:00:00:00:01 172.16.42.1
10:00:00:00:00:03 192.168.1.100 20:00:00:00:00:01 192.168.1.1 10:00:00:00:00:01
```

发送request

```
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16 1587540665.9818509s
```

超时了再发，下面的是时间，可以看到确实超过了1s

```
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16 1587540667.4834826 15:31:07
2020/04/22 INFO Got a packet: Ethernet 11:22:33:44:55:66->10:00:00:00:00:02 ARP | Arp
11:22:33:44:55:66:10.10.1.254 10:00:00:00:00:02:10.10.0.1 15:31:07 2020/04/22 INFO operation
kind ArpOperation.Reply 15:31:07 2020/04/22 INFO recive arp reply 15:31:07 2020/04/22 INFO
Table Shown as follows 172.16.42.2 30:00:00:00:00:01 172.16.42.1 10:00:00:00:00:03
192.168.1.100 20:00:00:00:00:01 192.168.1.1 10:00:00:00:00:01 10.10.1.254 11:22:33:44:55:66
10.10.0.1 10:00:00:00:00:02 send pac (find) router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
```

reply终于拿到手了，进行处理

```
15:31:07 2020/04/22 INFO Got a packet: Ethernet ab⊙ef🆎cd:ef->10:00:00:00:00:01 IP | IPv4
192.168.1.239->10.200.1.1 ICMP | ICMP EchoRequest 0 42 (0 data bytes) ipv4 IPv4 192.168.1.239-
>10.200.1.1 ICMP add packet to queue No Match Some Error occur!!!!!!!!!!!!!!!!!
```

发现了一个没有匹配的Entry

```
15:31:07 2020/04/22 INFO Not arp Packet 15:31:07 2020/04/22 INFO Table Shown as follows
172.16.42.2 30:00:00:00:00:01 172.16.42.1 10:00:00:00:00:03 192.168.1.100 20:00:00:00:00:01
192.168.1.1 10:00:00:00:00:01 10.10.1.254 11:22:33:44:55:66 10.10.0.1 10:00:00:00:00:02 15:31:07
2020/04/22 INFO Got a packet: Ethernet ab⊙ef🆎cd:ef->10:00:00:00:00:01 IP | IPv4
192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequest 0 42 (0 data bytes) ipv4 IPv4
192.168.1.239->10.10.50.250 ICMP add packet to queue
```

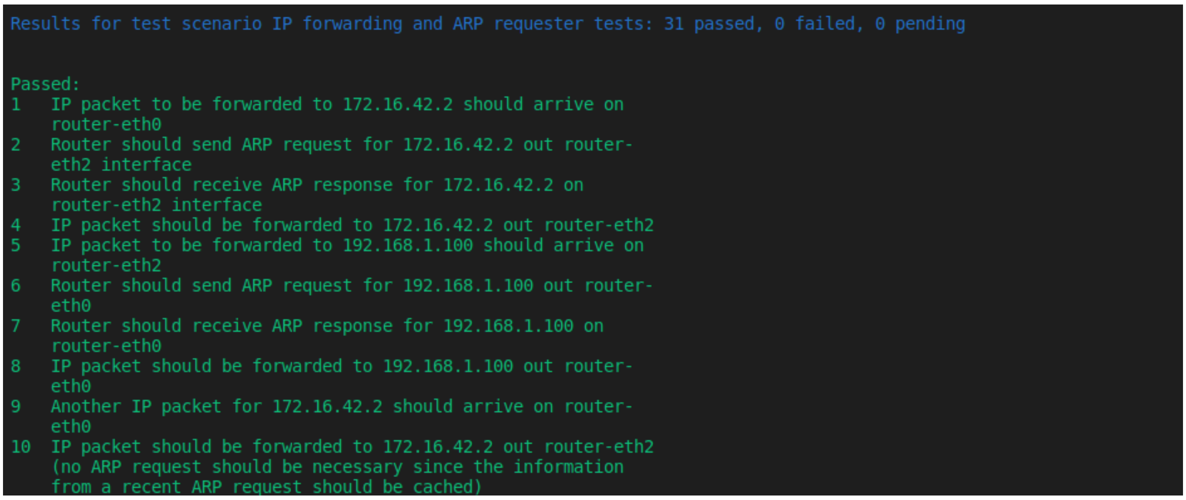
同样的加入队列待处理

```
15:31:07 2020/04/22 INFO Not arp Packet 15:31:07 2020/04/22 INFO Table Shown as follows
172.16.42.2 30:00:00:00:00:01 172.16.42.1 10:00:00:00:00:03 192.168.1.100 20:00:00:00:00:01
192.168.1.1 10:00:00:00:00:01 10.10.1.254 11:22:33:44:55:66 10.10.0.1 10:00:00:00:00:02
```

```
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16 1587540667.486576 send
requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16 1587540668.9877965 send requests
router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16 1587540670.4934397 send requests router-
eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16 1587540671.996658 send requests router-eth1
mac:10:00:00:00:00:02 ip:10.10.0.1/16 1587540673.4989097
```

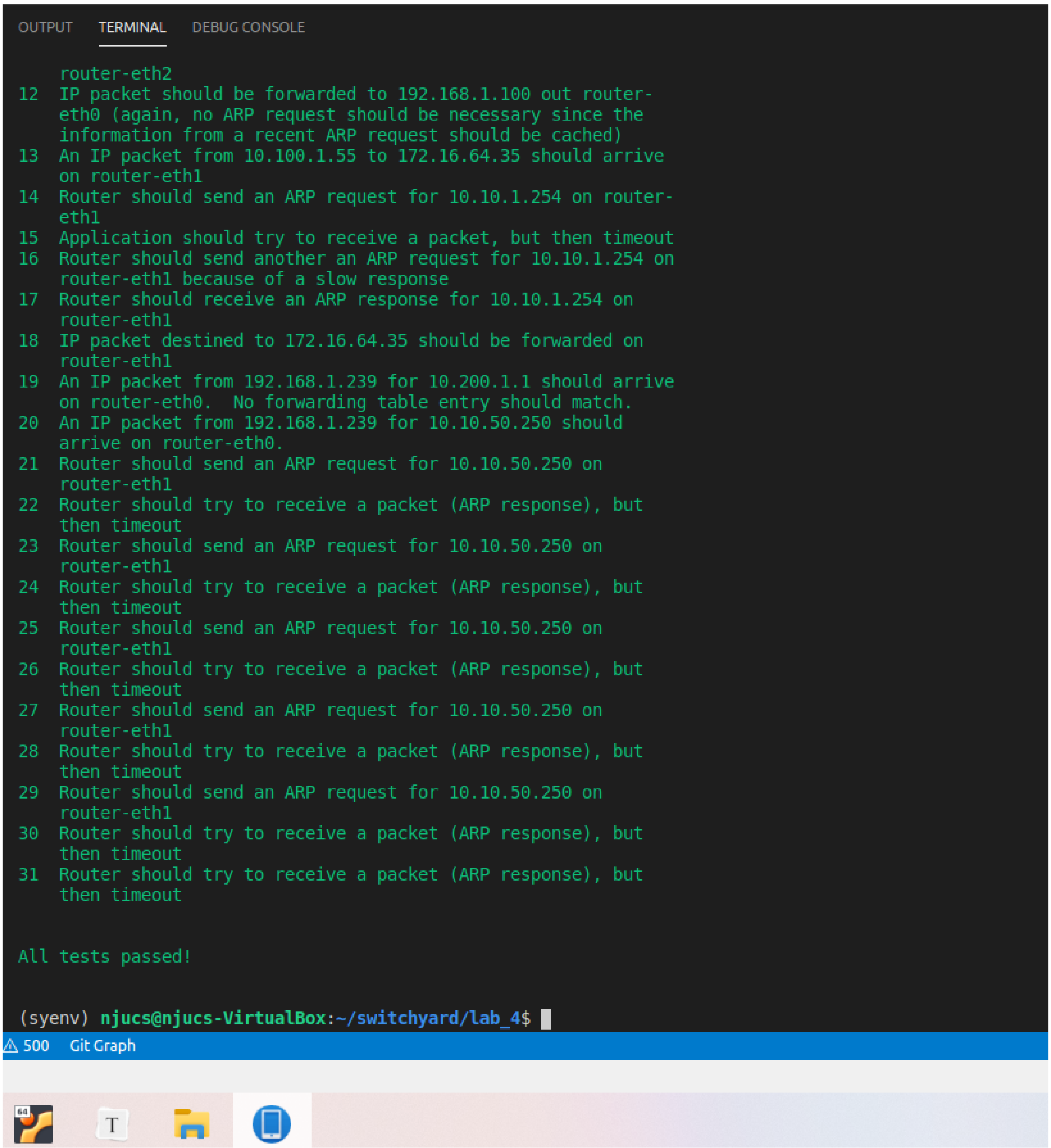
反复超时直到被丢弃

结果



```
Results for test scenario IP forwarding and ARP requester tests: 31 passed, 0 failed, 0 pending


Passed:
1   IP packet to be forwarded to 172.16.42.2 should arrive on
    router-eth0
2   Router should send ARP request for 172.16.42.2 out router-
    eth2 interface
3   Router should receive ARP response for 172.16.42.2 on
    router-eth2 interface
4   IP packet should be forwarded to 172.16.42.2 out router-eth2
5   IP packet to be forwarded to 192.168.1.100 should arrive on
    router-eth2
6   Router should send ARP request for 192.168.1.100 out router-
    eth0
7   Router should receive ARP response for 192.168.1.100 on
    router-eth0
8   IP packet should be forwarded to 192.168.1.100 out router-
    eth0
9   Another IP packet for 172.16.42.2 should arrive on router-
    eth0
10  IP packet should be forwarded to 172.16.42.2 out router-eth2
    (no ARP request should be necessary since the information
    from a recent ARP request should be cached)
```



```
    router-eth2
12  IP packet should be forwarded to 192.168.1.100 out router-
    eth0 (again, no ARP request should be necessary since the
    information from a recent ARP request should be cached)
13  An IP packet from 10.100.1.55 to 172.16.64.35 should arrive
    on router-eth1
14  Router should send an ARP request for 10.10.1.254 on router-
    eth1
15  Application should try to receive a packet, but then timeout
16  Router should send another an ARP request for 10.10.1.254 on
    router-eth1 because of a slow response
17  Router should receive an ARP response for 10.10.1.254 on
    router-eth1
18  IP packet destined to 172.16.64.35 should be forwarded on
    router-eth1
19  An IP packet from 192.168.1.239 for 10.200.1.1 should arrive
    on router-eth0.  No forwarding table entry should match.
20  An IP packet from 192.168.1.239 for 10.10.50.250 should
    arrive on router-eth0.
21  Router should send an ARP request for 10.10.50.250 on
    router-eth1
22  Router should try to receive a packet (ARP response), but
    then timeout
23  Router should send an ARP request for 10.10.50.250 on
    router-eth1
24  Router should try to receive a packet (ARP response), but
    then timeout
25  Router should send an ARP request for 10.10.50.250 on
    router-eth1
26  Router should try to receive a packet (ARP response), but
    then timeout
27  Router should send an ARP request for 10.10.50.250 on
    router-eth1
28  Router should try to receive a packet (ARP response), but
    then timeout
29  Router should send an ARP request for 10.10.50.250 on
    router-eth1
30  Router should try to receive a packet (ARP response), but
    then timeout
31  Router should try to receive a packet (ARP response), but
    then timeout


All tests passed!

(syenv) njucs@njucs-VirtualBox:~/switchyard/lab_4$ 
```

部署至mininet

my_example

使用server1 ping -c2 192.168.200.2构造流量

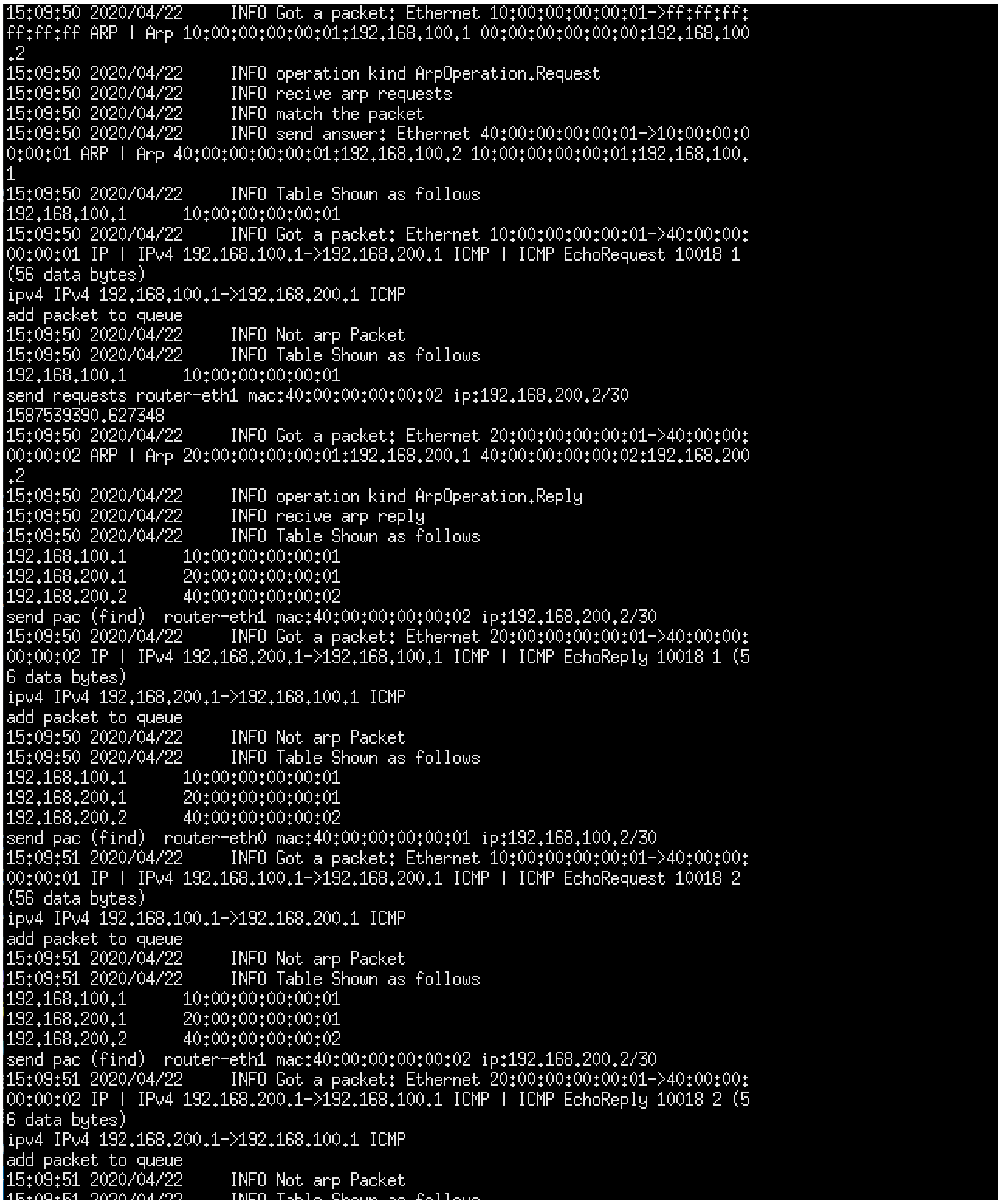
本质就是server1 ping server2

测试手段就是在server1和server2的端口设置wireshark进行抓包

（按照群里的说法在发送端（server1）抓包会有2+4的结果。在接收端server2会有2+4+2的结果）

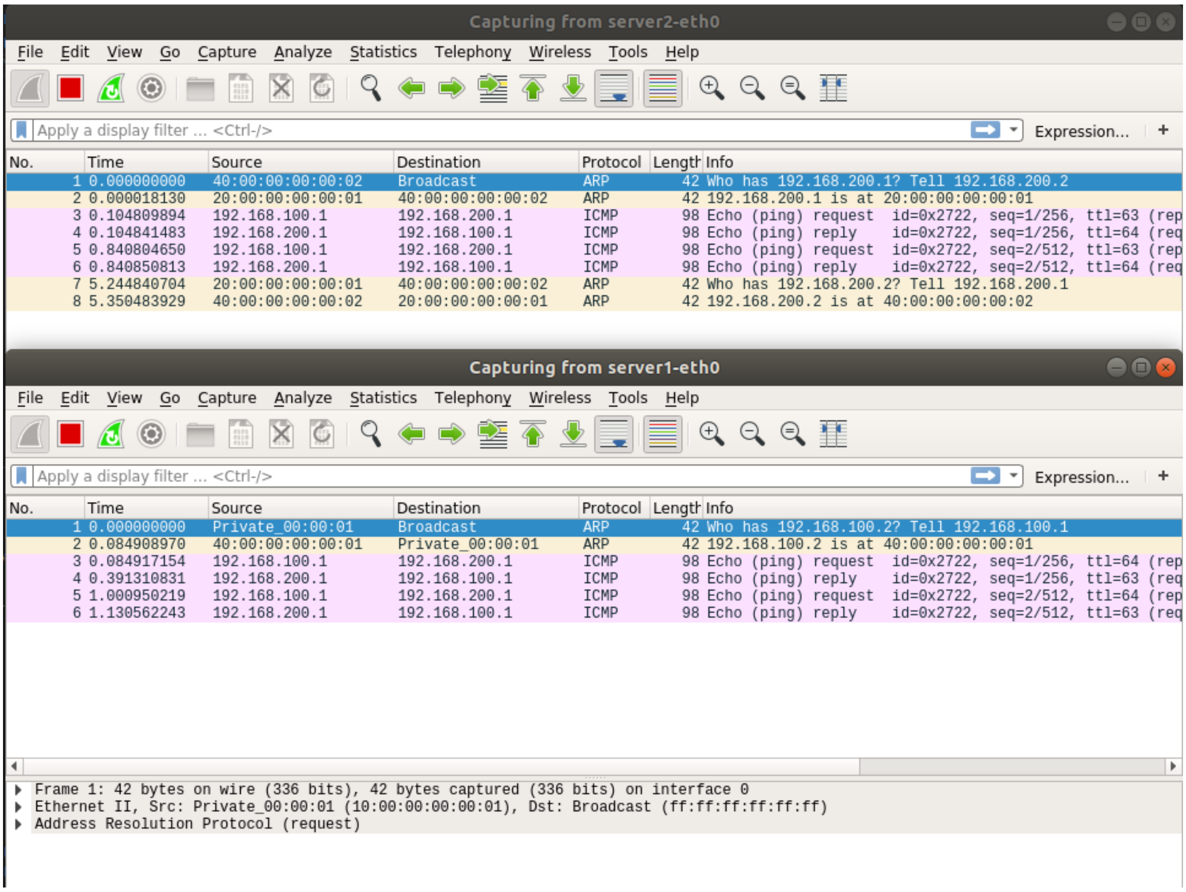
如下可见结果的正确性

最早的是server1的端口和路由器的端口先发送了arp request，路由器进行回复，server1得到了路由器和server1相连的端口的mac。随后server1发出了ICMP包，路由器一拿到就加入了处理队列，随后处理的时候发现本地的arp-mac表里没有，从eth1向server2发出了arp request。随后收到了server2的reply，这个时候队列就可以开始处理了，发出了ICMP包。然后路由器收到了ICMP的reply包，加入处理队列，由于之前有过arp的通信，于是有缓存，直接处理ICMP reply包。随后同理又是ICMP的request。



```
15:09:50 2020/04/22     INFO Got a packet: Ethernet 10:00:00:00:00:01->ff:ff:ff:
ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.100
.2
15:09:50 2020/04/22     INFO operation kind ArpOperation.Request
15:09:50 2020/04/22     INFO recive arp requests
15:09:50 2020/04/22     INFO match the packet
15:09:50 2020/04/22     INFO send answer: Ethernet 40:00:00:00:00:01->10:00:00:0
0:00:01 ARP | Arp 40:00:00:00:00:01:192.168.100.2 10:00:00:00:00:01:192.168.100.
1
15:09:50 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
15:09:50 2020/04/22     INFO Got a packet: Ethernet 10:00:00:00:00:01->40:00:00:
00:00:01 IP | IPv4 192.168.100.1->192.168.200.1 ICMP | ICMP EchoRequest 10018 1
(56 data bytes)
ipv4 IPv4 192.168.100.1->192.168.200.1 ICMP
add packet to queue
15:09:50 2020/04/22     INFO Not arp Packet
15:09:50 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
send requests router-eth1 mac:40:00:00:00:00:02 ip:192.168.200.2/30
1587539390.627348
15:09:50 2020/04/22     INFO Got a packet: Ethernet 20:00:00:00:00:01->40:00:00:
00:00:02 ARP | Arp 20:00:00:00:00:01:192.168.200.1 40:00:00:00:00:02:192.168.200
.2
15:09:50 2020/04/22     INFO operation kind ArpOperation.Reply
15:09:50 2020/04/22     INFO recive arp reply
15:09:50 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
192.168.200.1      20:00:00:00:00:01
192.168.200.2      40:00:00:00:00:02
send pac (find)  router-eth1 mac:40:00:00:00:00:02 ip:192.168.200.2/30
15:09:50 2020/04/22     INFO Got a packet: Ethernet 20:00:00:00:00:01->40:00:00:
00:00:02 IP | IPv4 192.168.200.1->192.168.100.1 ICMP | ICMP EchoReply 10018 1 (5
6 data bytes)
ipv4 IPv4 192.168.200.1->192.168.100.1 ICMP
add packet to queue
15:09:50 2020/04/22     INFO Not arp Packet
15:09:50 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
192.168.200.1      20:00:00:00:00:01
192.168.200.2      40:00:00:00:00:02
send pac (find)  router-eth0 mac:40:00:00:00:00:01 ip:192.168.100.2/30
15:09:51 2020/04/22     INFO Got a packet: Ethernet 10:00:00:00:00:01->40:00:00:
00:00:01 IP | IPv4 192.168.100.1->192.168.200.1 ICMP | ICMP EchoRequest 10018 2
(56 data bytes)
ipv4 IPv4 192.168.100.1->192.168.200.1 ICMP
add packet to queue
15:09:51 2020/04/22     INFO Not arp Packet
15:09:51 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
192.168.200.1      20:00:00:00:00:01
192.168.200.2      40:00:00:00:00:02
send pac (find)  router-eth1 mac:40:00:00:00:00:02 ip:192.168.200.2/30
15:09:51 2020/04/22     INFO Got a packet: Ethernet 20:00:00:00:00:01->40:00:00:
00:00:02 IP | IPv4 192.168.200.1->192.168.100.1 ICMP | ICMP EchoReply 10018 2 (5
6 data bytes)
ipv4 IPv4 192.168.200.1->192.168.100.1 ICMP
add packet to queue
15:09:51 2020/04/22     INFO Not arp Packet
15:09:51 2020/04/22     INFO Table Shown as follows
```

wireshark结果


Capturing from server2-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression… +

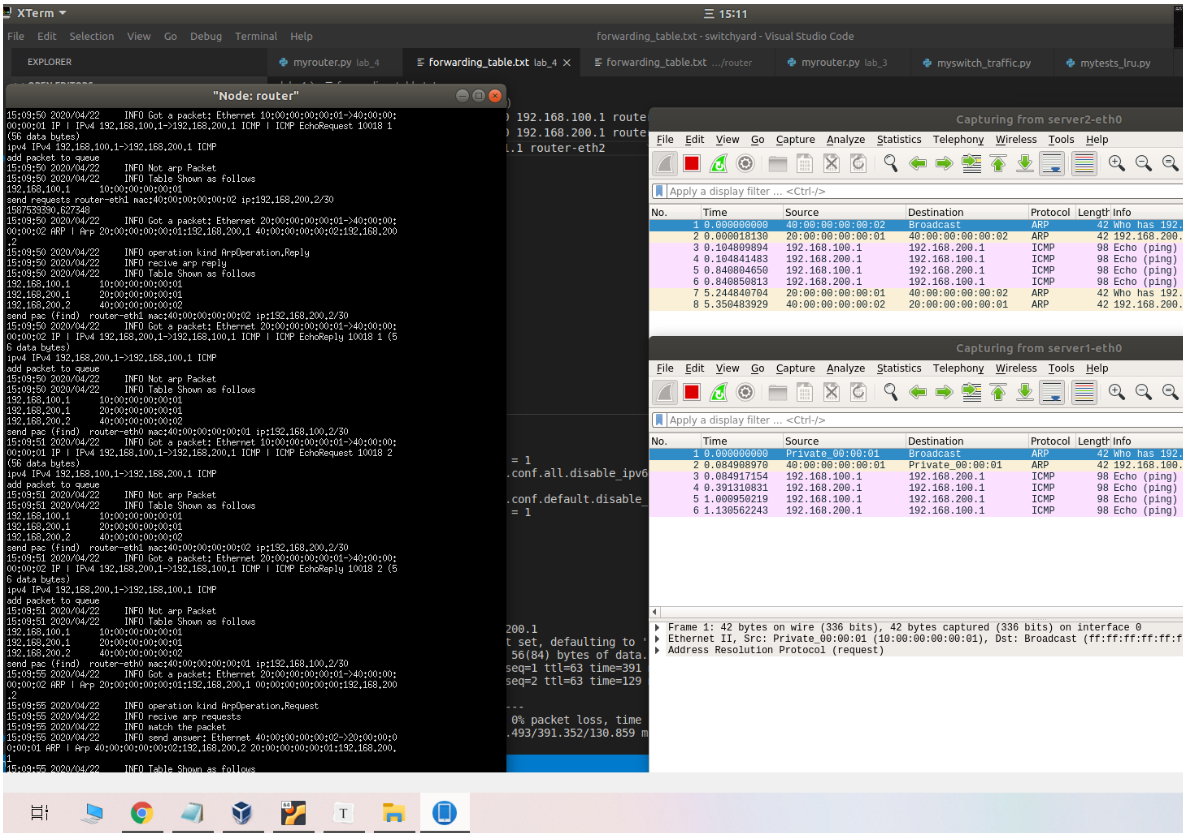
```
No.  Time         Source             Destination        Protocol Length Info
  1 0.000000000 40:00:00:00:00:02  Broadcast          ARP        42 Who has 192.168.200.1? Tell 192.168.200.2
  2 0.000018130 20:00:00:00:00:01  40:00:00:00:00:02  ARP        42 192.168.200.1 is at 20:00:00:00:00:01
  3 0.104809894 192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2722, seq=1/256, ttl=63 (rep
  4 0.104841483 192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2722, seq=1/256, ttl=64 (req
  5 0.840804650 192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2722, seq=2/512, ttl=63 (rep
  6 0.840850813 192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2722, seq=2/512, ttl=64 (req
  7 5.244840704 20:00:00:00:00:01  40:00:00:00:00:02  ARP        42 Who has 192.168.200.2? Tell 192.168.200.1
  8 5.350483929 40:00:00:00:00:02  20:00:00:00:00:01  ARP        42 192.168.200.2 is at 40:00:00:00:00:02
```

Capturing from server1-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression… +

```
No.  Time         Source             Destination        Protocol Length Info
  1 0.000000000 Private_00:00:01   Broadcast          ARP        42 Who has 192.168.100.2? Tell 192.168.100.1
  2 0.084908970 40:00:00:00:00:01  Private_00:00:01   ARP        42 192.168.100.2 is at 40:00:00:00:00:01
  3 0.084917154 192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2722, seq=1/256, ttl=64 (rep
  4 0.391310831 192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2722, seq=1/256, ttl=63 (req
  5 1.000950219 192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2722, seq=2/512, ttl=64 (rep
  6 1.130562243 192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2722, seq=2/512, ttl=63 (req

▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: Private_00:00:01 (10:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)
```

完全截图


XTerm
15:11
File Edit Selection View Go Debug Terminal Help
forwarding_table.txt - switchyard - Visual Studio Code
EXPLORER
myrouter.py lab_4
forwarding_table.txt lab_4
forwarding_table.txt …/router
myrouter.py lab_3
myswitch_traffic.py
mytests_lru.py
"Node: router"

```
15:09:50 2020/04/22     INFO Got a packet: Ethernet 10:00:00:00:00:01->40:00:00:
00:00:01 IP | IPv4 192.168.100.1->192.168.200.1 ICMP | ICMP EchoRequest 10018 1
(56 data bytes)
ipv4 IPv4 192.168.100.1->192.168.200.1 ICMP
add packet to queue
15:09:50 2020/04/22     INFO Not arp Packet
15:09:50 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
send requests router-eth1 mac:40:00:00:00:00:02 ip:192.168.200.2/30
1587539390.627348
15:09:50 2020/04/22     INFO Got a packet: Ethernet 20:00:00:00:00:01->40:00:00:
00:00:02 ARP | Arp 20:00:00:00:00:01:192.168.200.1 40:00:00:00:00:02:192.168.200
.2
15:09:50 2020/04/22     INFO operation kind ArpOperation.Reply
15:09:50 2020/04/22     INFO recive arp reply
15:09:50 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
192.168.200.1      20:00:00:00:00:01
192.168.200.2      40:00:00:00:00:02
send pac (find)  router-eth1 mac:40:00:00:00:00:02 ip:192.168.200.2/30
15:09:50 2020/04/22     INFO Got a packet: Ethernet 20:00:00:00:00:01->40:00:00:
00:00:02 IP | IPv4 192.168.200.1->192.168.100.1 ICMP | ICMP EchoReply 10018 1 (5
6 data bytes)
ipv4 IPv4 192.168.200.1->192.168.100.1 ICMP
add packet to queue
15:09:50 2020/04/22     INFO Not arp Packet
15:09:50 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
192.168.200.1      20:00:00:00:00:01
192.168.200.2      40:00:00:00:00:02
send pac (find)  router-eth0 mac:40:00:00:00:00:01 ip:192.168.100.2/30
15:09:51 2020/04/22     INFO Got a packet: Ethernet 10:00:00:00:00:01->40:00:00:
00:00:01 IP | IPv4 192.168.100.1->192.168.200.1 ICMP | ICMP EchoRequest 10018 2
(56 data bytes)
ipv4 IPv4 192.168.100.1->192.168.200.1 ICMP
add packet to queue
15:09:51 2020/04/22     INFO Not arp Packet
15:09:51 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
192.168.200.1      20:00:00:00:00:01
192.168.200.2      40:00:00:00:00:02
send pac (find)  router-eth1 mac:40:00:00:00:00:02 ip:192.168.200.2/30
15:09:51 2020/04/22     INFO Got a packet: Ethernet 20:00:00:00:00:01->40:00:00:
00:00:02 IP | IPv4 192.168.200.1->192.168.100.1 ICMP | ICMP EchoReply 10018 2 (5
6 data bytes)
ipv4 IPv4 192.168.200.1->192.168.100.1 ICMP
add packet to queue
15:09:51 2020/04/22     INFO Not arp Packet
15:09:51 2020/04/22     INFO Table Shown as follows
192.168.100.1      10:00:00:00:00:01
192.168.200.1      20:00:00:00:00:01
192.168.200.2      40:00:00:00:00:02
send pac (find)  router-eth0 mac:40:00:00:00:00:01 ip:192.168.100.2/30
15:09:55 2020/04/22     INFO Got a packet: Ethernet 20:00:00:00:00:01->40:00:00:
00:00:02 ARP | Arp 20:00:00:00:00:01:192.168.200.1 00:00:00:00:00:00:192.168.200
.2
15:09:55 2020/04/22     INFO operation kind ArpOperation.Request
15:09:55 2020/04/22     INFO recive arp requests
15:09:55 2020/04/22     INFO match the packet
15:09:55 2020/04/22     INFO send answer: Ethernet 40:00:00:00:00:02->20:00:00:0
0:00:01 ARP | Arp 40:00:00:00:00:02:192.168.200.2 20:00:00:00:00:01:192.168.200.
1
15:09:55 2020/04/22     INFO Table Shown as follows
```

在最后有接收端的arp请求和回复过程（应该是为什么接收端会多出2个arp的原因，既有一开始发送arp的时候的请求和回复，还有最后的一次请求和回复。对比起来发送端只有一开始的一次）

总结与感想

1. 一个是start_mininet竟然会对txt进行修改，还改乱了。确实没有想到，花了写时间才解决

2. 另一个就是写代码出错的时候多看看测试样例，测试样例写的很清楚，关于每一步做了啥，应该得到啥，结果得到了啥都很清楚，多读读样例有利于快速找到错误，同时写报告的时候也轻松一些。
3. 调一些自己不熟的包很容易出现完全想不到的错误（报错会稀奇古怪的落到一些库文件里，看起来很恐怖而且一点调错的思路都没有）但是其实错的可能是一些比较基础的错误，可以多看看一看Q&A