

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 助教：方毓楚/于沛文/郑浩/陈伟/李国浩/李睿宸/杨溢...
学院：计算机科学与技术系	专业(方向)：计算机科学与技术
学号：201220096	姓名：钟亚晨
Email: 2991908515@qq.com	开始/完成日期：2022.04.18-2022.04.19

1.实验名称

IPv4 Router: Forwarding Packets

完成清单：

- ✧ 完成了IP转发表的构建与查询、生成ARP Request解析IP对应的MAC地址以及转发，并通过了框架测试中的所有测试用例；
- ✧ 自己编写了针对Lab4的自定义测试，并且通过了所有测试用例；
- ✧ 通过 `mininet` 模拟真实网络环境，结合 `wireshark` 抓包分析，进一步验证了功能实现逻辑的正确性；
- ✧ （选做）完成了多线程版本的逻辑实现，并且通过了框架中的多线程版本测试的所有测试用例；

2.实验目的

- ✧ 实现 `Forwarding Table` 的构建、查询逻辑；
- ✧ 实现路由器转发IP报文、构造ARP Request解析IP的逻辑，完成路由器最基本的转发功能；
- ✧ 熟悉 `switchyard` 相关API的查询、使用；
- ✧ 通过编码实践进一步掌握IPv4路由器的工作原理，并且理解链路层和网络层是怎样各司其职相互配合完成不同子网主机间的报文交换；
- ✧ 复习及使用ARP地址解析协议、报文结构；
- ✧ 完成多线程版本逻辑实现，并利用框架测试检验实现正确性；

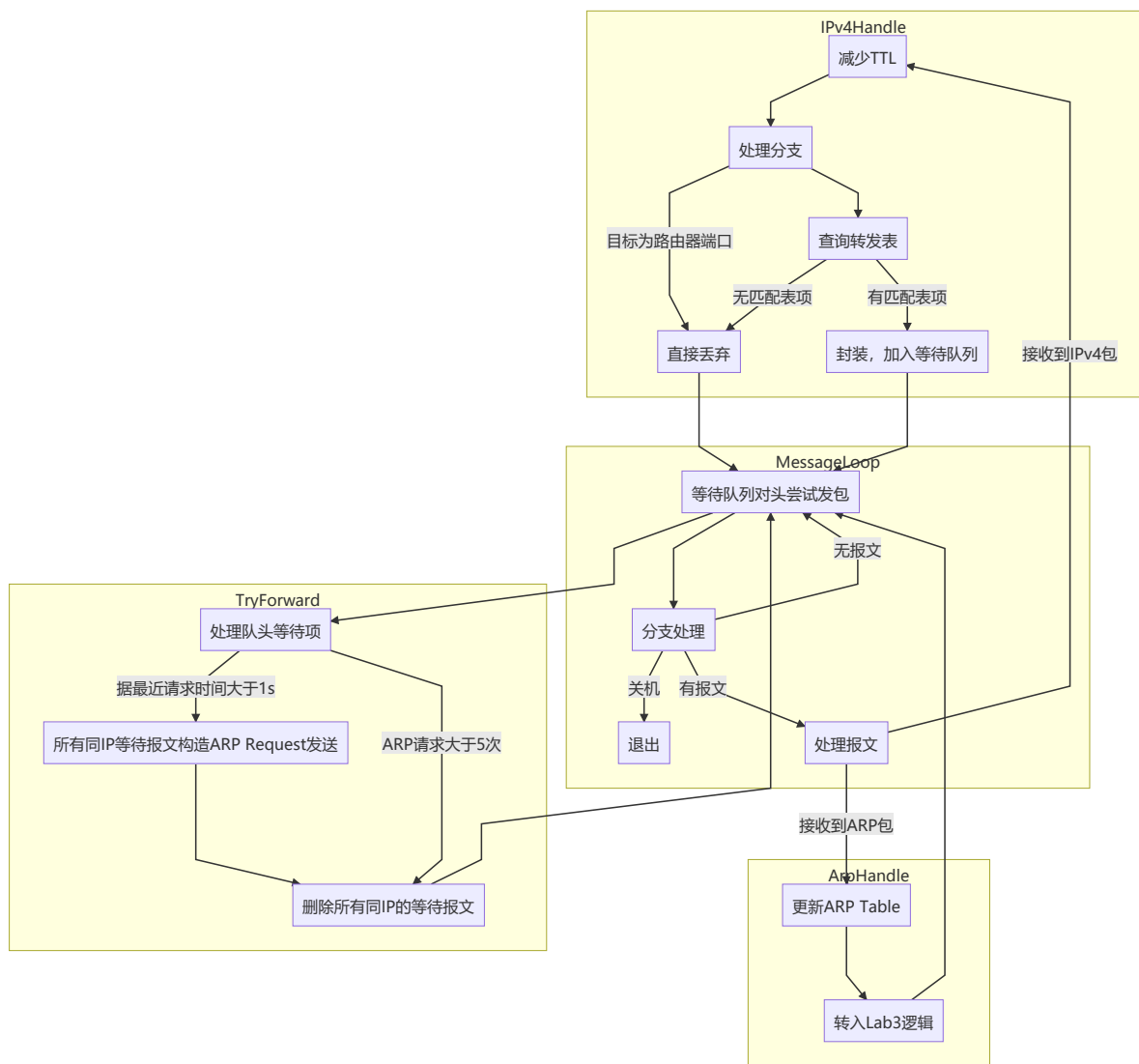
3.实验内容

- ✧ IPv4路由器使用静态路由，实现接收并转发从链路上到达、目标为其它主机的报文，完成路由器最基本的转发功能；
- ✧ 生成ARP Request以解析出IP对应的因特网MAC地址，配合先前实现的ARP Table，供转发使用；

4.实验结果

简单说明，以下对实验说明中的四个问题进行回答。

首先对实验框架代码进行了重构，并且依照实验说明，可知道**整体逻辑**如下：



0x01 In the report, show how you implement the logic of building IP forwarding table and matching the destination IP addresses.

一、转发表的构建

转发表的构建来源有两个

- ①.路由器的所有端口；
- ②.读取名为 forwarding_table.txt 文件中的内容；

转发表的结构为

一个列表类型作为转发表，转发表中的每一个表项为一个具有3个键值对的字典¹

转发表的具体构建细节

```
29 # Forwarding Table Initialization
30 # | network address / subnet address | next hop address | interface |
31 self.forwarding_table = []
32 # Initialize from interfaces
33 for it in self.interfaces:
34     self.forwarding_table.append({
35         'ipv4network': IPv4Network((str(IPv4Address(int(it.ipaddr) & int(it.netmask)))) + '/' + str(it.netmask)),
36         'nextip': IPv4Address('0.0.0.0'),
37         'interface': it.name
38     })
39 # Initialize from txt file
40 with open('forwarding_table.txt') as fp:
41     for line in fp.readlines():
42         buffer = line.split()
43         self.forwarding_table.append({
44             'ipv4network': IPv4Network(buffer[0] + '/' + buffer[1]),
45             'nextip': IPv4Address(buffer[2]),
46             'interface': buffer[3]
47         })
48 # Show Forwarding Table information--
```

- ①33-38行，遍历路由器端口追加表项，通过路由器端口的IP地址、子网掩码获取子网IP；
- ②40-47行，利用Python的文件IO相关函数，以及字符串split函数、类型转换逐行从文件中读入条目并作分割，追加表项；

二、利用转发表匹配目标地址

依照实验说明：

在构建转发表之后，在接收到IPv4报文当中的目标IP地址应当通过转发表进行匹配。

- 当至少有两个表项被匹配时，最长前缀匹配应当被使用
- 如果没有表项被匹配，则将该报文丢弃
- 如果报文目标是路由器的任意一个端口的IP地址，那么同样将该报文丢弃

实现逻辑如下：

```
84     ### Handle IPv4 Packet ###
85     def handle_ipv4_packet(self, ipv4_header, ifaceName, packet):
86         # Decrease TTL
87         packet[IPv4].ttl -= 1
88         # If Targetip belongs to one of router interfaces, do nothing
89         for it in self.interfaces:
90             if packet[IPv4].dst == it.ipaddr:
91                 return None
92         # Search matched item in Forwarding Table
93         max_matched_prefixlen = 0
94         target_item = {}
95         for it in self.forwarding_table:
96             if packet[IPv4].dst in it['ipv4network'] and it['ipv4network'].prefixlen > max_matched_prefixlen:
97                 max_matched_prefixlen = it['ipv4network'].prefixlen
98                 target_item = it
99         # If no find matched item, do nothing
100        if max_matched_prefixlen == 0:
101            return None
102        # Find nextip and interface
103        for it in self.interfaces:
104            if it.name == target_item['interface']:
105                target_interface = it
106        # Add to waiting queue
107        waitpacket = WaitPacket(target_item['nextip'], target_interface, packet)
108        self.waiting_queue.append(waitpacket)
```

①88-91行：遍历路由器的所有端口，如果IPv4报文的目标IP与其中某个端口匹配，则直接结束报文处理函数（即将该报文丢弃，什么也不做）；

②93-98行：遍历转发表，找到IPv4报文的目标IP所在的、拥有最长前缀的子网表项；

③100-101行：检验最长匹配前缀长度，如果为0则说明没有匹配表项，直接结束报文处理函数（即将该报文丢弃，什么也不做）；

④103-108行：如果通过了上述的所有检验执行到这一步，那么通过所得到的转发表表项，遍历路由器端口，找到对应端口，生成一个封装包加入到等待队列中：

1.传入下一跳IP地址以供进行ARP解析找到下一跳MAC地址；

2.传入发送接口，以提供构建ARP Request所需要的源MAC地址、源IP地址以及发送IP报文时构建以太网头所需要的源MAC地址，以及发送报文时指定端口；

3.传入报文，以等待转发；

0x02 In the report, show how you implement the logic of forwarding the packet and ARP.

前置工作

①创建一个类，封装等待发送的IP报文。每个对象包括：

下一跳IP地址、发送报文的端口、待发送的IP报文、最近发送ARP请求的时间、总共发送ARP请求的次数、ARP Request报文

```
13 class WaitPacket:
14     ### WaitPacket Initialization ###
15     def __init__(self, nextip, interface, packet):
16         self.nextip = packet[IPv4].dst if nextip == IPv4Address('0.0.0.0') else nextip
17         self.interface = interface
18         self.packet = packet
19         self.latest_arp_time = -2
20         self.arp_cnt = 0
21         self.arp_request = 0
```

②路由器初始化时追加一个等待队列和一个删除缓冲队列，用于存放等待转发的报文，以及该批次应当从等待队列中删除的所有报文（通过上一步的封装形成封装报文）：

```
57         # Some queues for scheduling
58         self.waiting_queue = []
59         self.delete_queue = []
```

③在路由器的消息循环中追加对等待队列队头数据报文的发送尝试：

```
188         while True:
189             # # ARP Table Timeout
190             # buffer = []
191             # for it in self.arp_table:
192                 # if time.time() < it['ttl']:
193                     # buffer.append(it)
194             # self.arp_table = buffer
195             # Try to forward packets
196             self.forward_packet()
197             # Some Cases
```

核心思路


代码量较大，以代码+注释的形式进行说明

```
1  ### Try To Forward ###
2  def forward_packet(self):
3      # Make delete queue empty
4      # 清空待删除缓冲
5      self.delete_queue = []
6      # If waiting queue is empty, do nothing
7      # 如果等待队列为空，当前没有报文需要转发，什么也不做
8      if len(self.waiting_queue) == 0:
9          return None
10     # Get the front of waiting queue
11     # 如果队列不为空，获取队列首部的封装报文，对其尝试转发
12     waitpkt = self.waiting_queue[0]
13     # 查询ARP Table以解析下一跳IP地址对应的MAC地址
14     nextmac = None
15     for arp_it in self.arp_table:
16         if arp_it['ipaddr'] == waitpkt.nextip:
17             nextmac = str(arp_it['macaddr'])
18     # Have matched item exists in ARP Table
19     # 如果可以通过ARP Table直接解析出IP地址对应的MAC地址，则对所有具有
    相同的目标IP的等待包修改报文的以太网头，将其从转发表表项指定的端口发出，并从
    等待队列中删除封装报文
20     if nextmac:
21         for it in self.waiting_queue:
22             if it.packet[IPv4].dst ==
self.waiting_queue[0].packet[IPv4].dst:
23                 it.packet[Ethernet].src = waitpkt.interface.ethaddr
24                 it.packet[Ethernet].dst = nextmac
25                 self.net.send_packet(it.interface.name, it.packet)
26                 self.delete_queue.append(it)
27     # No matched item exists in ARP Table!
28     # 如果无法通过ARP Table直接解析出IP地址对应的MAC地址，则进行如下操
    作：
29     # ARP Request time not less than 5
30     # 如果ARP Request已经发送达到或者大于5次，则放弃该报文的转发，并且
    将此时缓冲队列中所有具有相同目的IP的等待报文删除
31     elif waitpkt.arp_cnt >= 5:
```

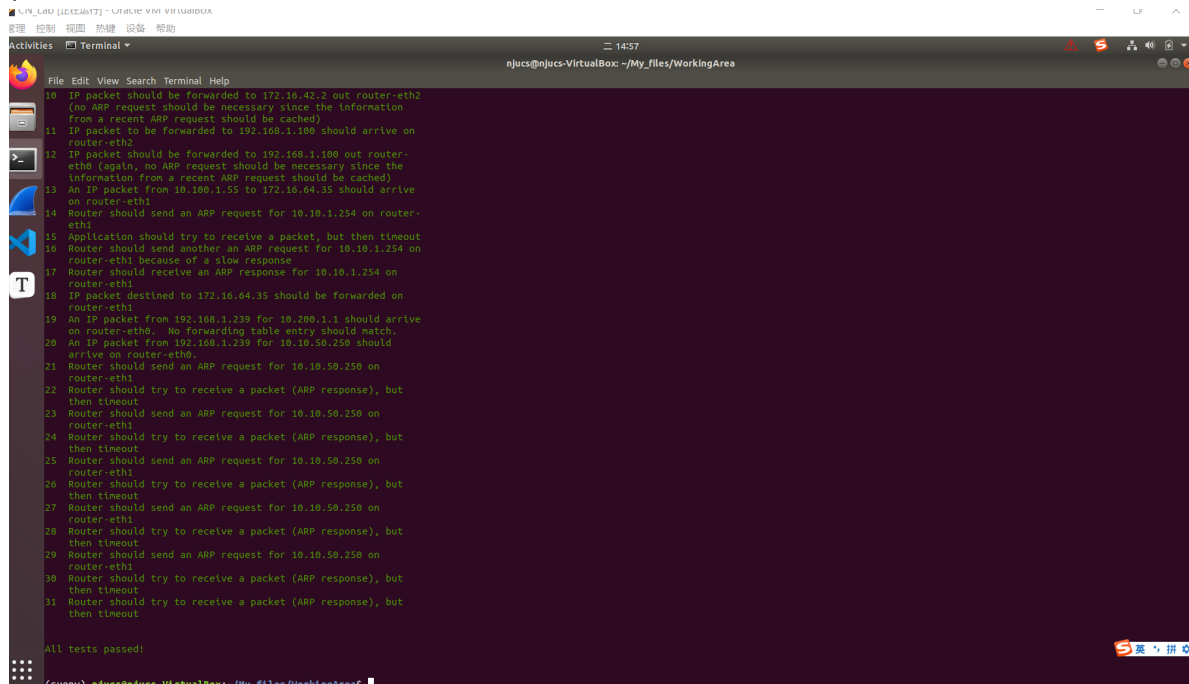
```

32         for it in self.waiting_queue:
33             if it.packet[IPv4].dst ==
self.waiting_queue[0].packet[IPv4].dst:
34                 self.delete_queue.append(it)
35             # ARP Request time less than 5
36             # 如果ARP Request发送次数小于五次，则继续尝试解析下一条IP
37             elif waitpkt.arp_cnt < 5:
38                 # 如果距离上一次解析不足1s，则什么也不做
39                 if time.time() - waitpkt.latest_arp_time < 1:
40                     return None
41                 # Build ARP Request
42                 # 如果是第一次发送ARP Request，则构建ARP Request并且将其存储在
封装对象当中
43                 if waitpkt.arp_cnt == 0:
44                     ethernet_header = Ethernet()
45                     ethernet_header.ethertype = EtherType.ARP
46                     ethernet_header.src = waitpkt.interface.ethaddr
47                     ethernet_header.dst = 'ff:ff:ff:ff:ff:ff'
48                     arp_data = Arp(
49                         operation=ArpOperation.Request,
50                         senderhwaddr=waitpkt.interface.ethaddr,
51                         senderprotoaddr=waitpkt.interface.ipaddr,
52                         targethwaddr='ff:ff:ff:ff:ff:ff',
53                         targetprotoaddr=waitpkt.nextip
54                     )
55                     arp_request_pkt = ethernet_header + arp_data
56                     waitpkt.arp_request = arp_request_pkt
57                 # Send ARP Request and update waitPacket
58                 # 从指定端口发出ARP Request尝试解析下一跳的IP地址，并且更新封装
对象的ARP请求次数、最近APR请求时间
59                 self.net.send_packet(waitpkt.interface.name,
waitpkt.arp_request)
60                 waitpkt.arp_cnt += 1
61                 waitpkt.latest_arp_time = time.time()
62                 # 从等待队列中删除所有处于删除缓冲中的等待发送报文
63                 for it in self.delete_queue:
64                     self.waiting_queue.remove(it)

```

0x03  In the report, show the test result of your router. (Optional) If you have written the test files yourself, show how you test forwarding packets.

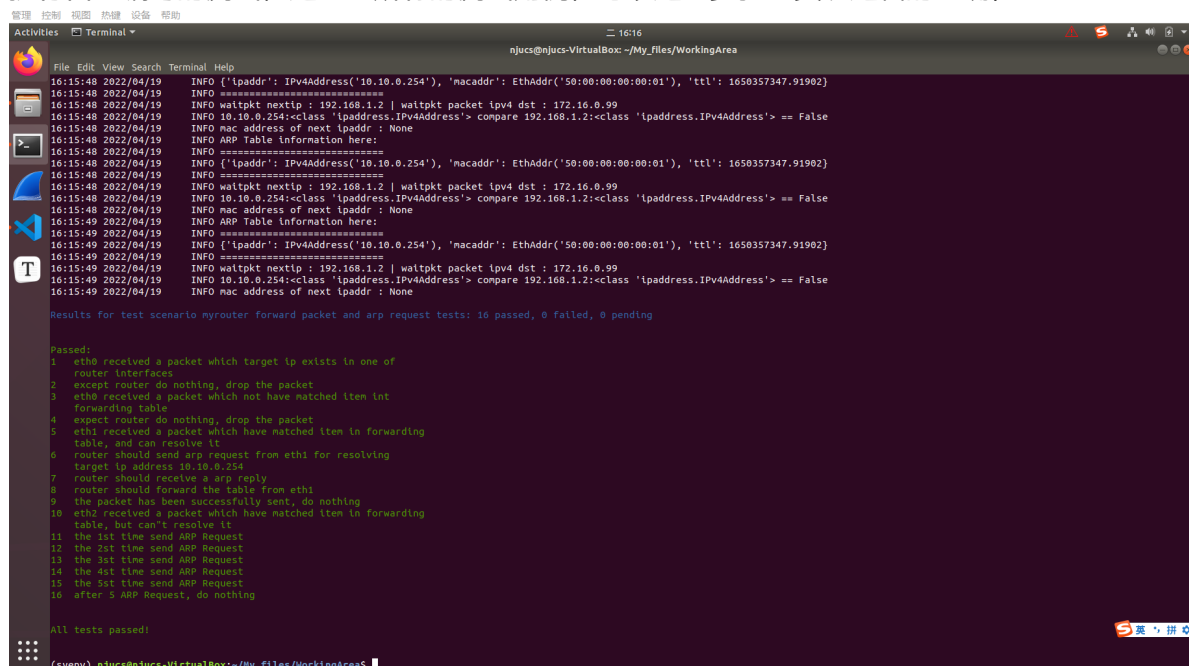
利用框架代码所给的测试用例进行测试，可以通过所有测试用例，可以初步验证实现逻辑的正确性：



也可见同目录下 demo00.gif 对框架测试用例执行的动态演示

（可选）自己编写测试用例，展示测试用例结果以及测试用例的测试逻辑。

执行自己编写的测试，通过了所有的测试用例，可以进一步验证实现逻辑的正确性：



另外，还使用了Lab3中自编写的测试用例进行测试，以保证在完成Lab4后，Lab3中实现的部分功能仍能测试通过。通过了Lab3中自编写测试的所有测试用例。

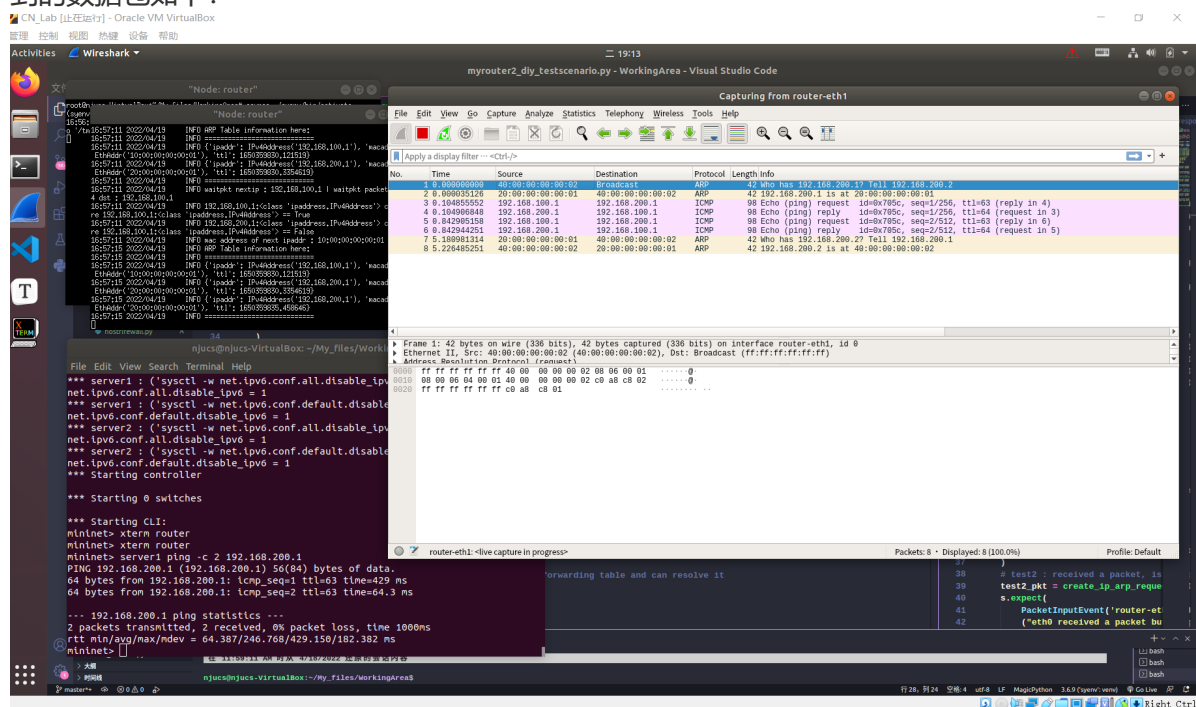
对于本次实验的自编写测试用例，一共有四个测试用例，各个用例的逻辑如下：

- ①接收到一个报文，其目标IP为路由器的某一个端口，期望路由器什么也不做，将该报文直接丢弃；
- ②接收到一个报文，其目标IP不为路由器的某一个端口，但是在转发表中无法找到匹配表项，期望路由器什么也不做，将该报文直接丢弃；
- ③接收到一个报文，其目标IP在转发表中有匹配项，并且下一跳的MAC地址可解析，期望路由器从特定端口发送一个ARP Request，并且接收到ARP Reply，然后再从该端口将修改后的报文发送出去，并且发送成功后不再有其它动作；

④接收到一个报文，其目标IP在转发表中有匹配项，但是下一跳的MAC地址不可解析，期望路由器从特定端口每间隔1s发送一个ARP Request，一共5个，当发送完第五个ARP Request之后，不再有其它动作；

0x04 ping another host(client or server2) from server1. Using Wireshark to prove that your router is correctly forwarding the packets. Write the procedure and analysis in your report with screenshots.

利用 wireshark 在路由器的 router-eth1 端口进行抓包，从 server1 去 ping server2，抓取到的数据包如下：



①第一个ARP报文：可以看到，一开始ARP表中并没有 server2 的IP-MAC表项，因此 server1 发送给路由器的报文无法解析，但是根据从转发表中最长前缀匹配得知应当由 router-eth1 端口发送ARP Request以解析出 server2 的MAC地址。

②第二个ARP报文：server2 接收到来自路由器的广播，并且比对自己的IP地址后，发送出一个ARP Reply被路由器接收到，从而路由器的ARP表中添加对应的表项，之后从 server1 到 server2 的报文便可被解析转发。

③第三到第六个ICMP报文：对应两对ICMP请求与应答，即我们执行的 ping -c 2 192.168.200.1 指令。

④第七、八个ARP报文：由于单播轮询机制所导致，定期向ARP缓存条目的主机发送单播的ARP请求报文，是一种ARP老化机制。可以参考英文文档：

Unicast Poll -- Actively poll the remote host by periodically sending a point-to-point ARP Request o it, and delete the entry if no ARP Reply is received from N successive polls. Again, the timeout should be on the order of a minute, and typically N is 2.

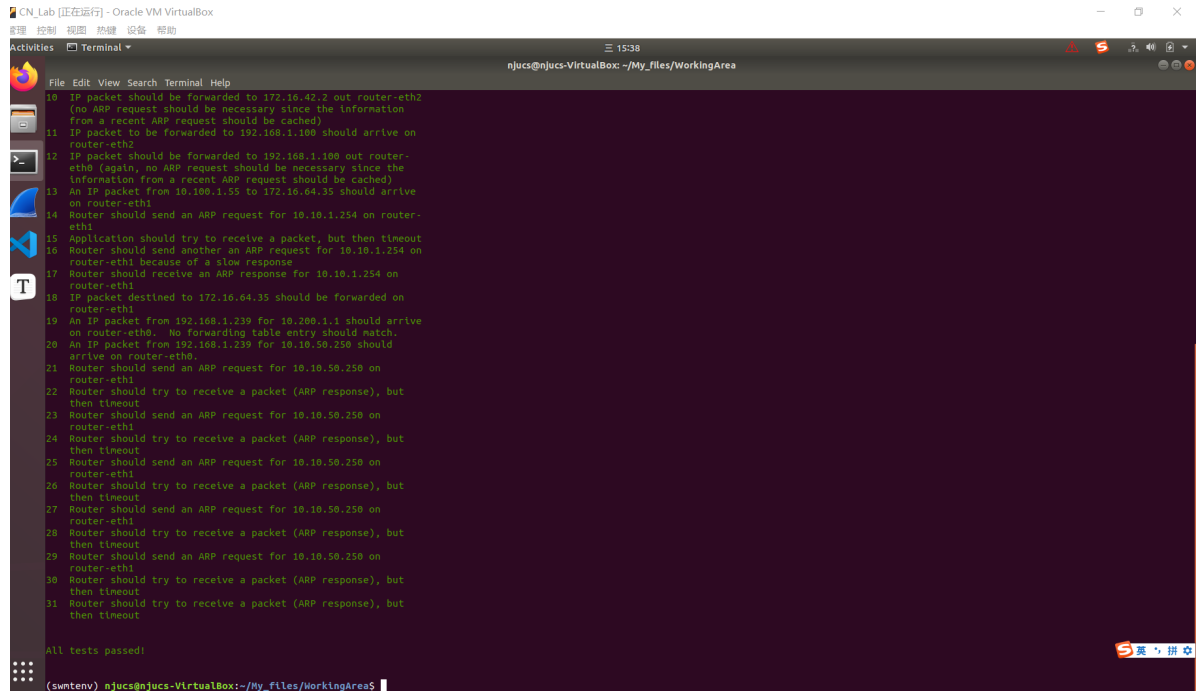
(选做) 多线程版本

具体实现代码存放在 `./report/myrouter_thread.py` 中，其实现思路为：

利用Python的threading模块进行多线程编程：

- ①接收报文占据一个线程，用以运行路由器收报文的逻辑；
 - ②与此同时，对路由器等待队列中等待报文的尝试发送逻辑占据另一个线程，用以运行路由器ARP请求以及报文转发的逻辑；
- 总而言之，路由器有两个线程并行工作，一个收报文，一个发报文，并发执行，从而将原本“收-发-收-发...”的交替逻辑变成同时进行。

可以看到，通过了多线程版本下的所有测试用例，可以初步验证实现的正确性：



```
CN_Lab [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities Terminal
File Edit View Search Terminal Help
njucs@njucs-VirtualBox: ~/My_files/WorkingArea
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)
11 IP packet to be forwarded to 192.168.1.100 should arrive on
   router-eth2
12 IP packet should be forwarded to 192.168.1.100 out router-
   eth2 (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!
```

5.核心代码

```
1 //Forwarding Table的建立
2 # Forwarding Table Initialization
3     # | network address / subnet address | next hop address |
   interface |
4     self.forwarding_table = []
5     # Initialize from interfaces
6     for it in self.interfaces:
7         self.forwarding_table.append({
8             'ipv4network' :
9             IPv4Network((str(IPv4Address(int(it.ipaddr) & int(it.netmask)))) + '/'
10             + str(it.netmask)),
11             'nextip' :
12             IPv4Address('0.0.0.0'),
13             'interface' : it.name
14         })
15     # Initialize from txt file
16     with open('forwarding_table.txt') as fp:
17         for line in fp.readlines():
18             buffer = line.split()
```

```

16         self.forwarding_table.append({
17             'ipv4network' :
18             IPv4Network(buffer[0] + '/' + buffer[1]),
19             'nextip' :
20             IPv4Address(buffer[2]),
21             'interface' :
22             buffer[3]
23         })
24
25 //利用 Forwarding Table匹配报文的目标IP
26 def handle_ipv4_packet(self, ipv4_header, ifaceName, packet):
27     # Decrease TTL
28     packet[IPv4].ttl -= 1
29     # If Targetip belongs to one of router interfaces, do nothing
30     for it in self.interfaces:
31         if packet[IPv4].dst == it.ipaddr:
32             return None
33     # Search matched item in Forwarding Table
34     max_matched_prefixlen = 0
35     target_item = {}
36     for it in self.forwarding_table:
37         if packet[IPv4].dst in it['ipv4network'] and
38         it['ipv4network'].prefixlen > max_matched_prefixlen:
39             max_matched_prefixlen = it['ipv4network'].prefixlen
40             target_item = it
41     # If no find matched item, do nothing
42     if max_matched_prefixlen == 0:
43         return None
44     # Find nextip and interface
45     for it in self.interfaces:
46         if it.name == target_item['interface']:
47             target_interface = it
48     # Add to waiting queue
49     waitpacket = WaitPacket(target_item['nextip'],
50                             target_interface, packet)
51     self.waiting_queue.append(waitpacket)
52
53 //对等待队列首部封装对象报文的尝试发送以及ARP Request请求解析下一跳IP
54 def forward_packet(self):
55     # If waiting queue is empty, do nothing
56     if len(self.waiting_queue) == 0:
57         return None
58     # Get the front of waiting queue
59     waitpkt = self.waiting_queue[0]
60     # For debugg--
61     # Show ARP Table information--
62     log_info(f'ARP Table information here:')
63     log_info('=====')
64     for it in self.arp_table:
65         log_info(f'{it}')
66     log_info('=====')
67     # Show ARP Table information--
68     log_info(f'waitpkt nextip : {waitpkt.nextip} | waitpkt packet
69     ipv4 dst : {waitpkt.packet[IPv4].dst}')

```

```

62         # For debug--
63         nextmac = None
64         for arp_it in self.arp_table:
65             # For debug--
66             tmp = arp_it['ipaddr']
67             log_info(f'{tmp}:{type(tmp)} compare {waitpkt.nexttip}:
{type(waitpkt.nexttip)} == {tmp == waitpkt.nexttip}')
68             # For debug--
69             if arp_it['ipaddr'] == waitpkt.nexttip:
70                 nextmac = str(arp_it['macaddr'])
71         # For debug--
72         log_info(f'mac address of next ipaddr : {nextmac}')
73         # For debug--
74         # Have matched item exists in ARP Table
75         if nextmac:
76             waitpkt.packet[Ethernet].src = waitpkt.interface.ethaddr
77             waitpkt.packet[Ethernet].dst = nextmac
78             self.net.send_packet(waitpkt.interface.name,
waitpkt.packet)
79             self.waiting_queue.remove(waitpkt)
80         # No matched item exists in ARP Table!
81         # ARP Request time not less than 5
82         elif waitpkt.arp_cnt >= 5:
83             self.waiting_queue.remove(waitpkt)
84         # ARP Request time less than 5
85         elif waitpkt.arp_cnt < 5:
86             if time.time() - waitpkt.latest_arp_time < 1:
87                 return None
88             # Build ARP Request
89             if waitpkt.arp_cnt == 0:
90                 ethernet_header = Ethernet()
91                 ethernet_header.ethertype = EtherType.ARP
92                 ethernet_header.src = waitpkt.interface.ethaddr
93                 ethernet_header.dst = 'ff:ff:ff:ff:ff:ff'
94                 arp_data = Arp(
95                     operation=ArpOperation.Request,
96
97                 senderhwaddr=waitpkt.interface.ethaddr,
98
99                 senderprotoaddr=waitpkt.interface.ipaddr,
100                     targethwaddr='ff:ff:ff:ff:ff:ff',
101                     targetprotoaddr=waitpkt.nexttip
102                 )
103                 arp_request_pkt = ethernet_header + arp_data
104                 waitpkt.arp_request = arp_request_pkt
105             # Send ARP Request and update WaitPacket
106             self.net.send_packet(waitpkt.interface.name,
waitpkt.arp_request)
107             waitpkt.arp_cnt += 1
108             waitpkt.latest_arp_time = time.time()

```

6.总结与感想

总结：

- 🧐对ARP地址解析协议的具体应用有了进一步理解与掌握；
- 🧐对链路层和网络层协同工作，顺利完成不同子网主机间信息交换有了实际体会。网络层指明大致方向，而链路层则更侧重于细节，根据网路层指明的大方向来决定下一跳该怎么走，不同层次间各司其职，相互配合，最终完成功能；
- 🧐对于报文结构有了进一步体会，报文结构实际上切实反应了计算机网络分层架构，通过一层一层向下，对最原始的数据包一层一层追加头部报文，不同的头部报文供不同层次的不同协议进行使用以完成功能；
- 🧐通过完成选做功能学习了python多线程编程，对于操作系统课程中线程的概念以及运行效果有了体会和理解，拓宽了知识面；

感想：

- 🧐本次实验的规模相较于先前突然大了起来，使用OOP当中的封装抽象思维可以较好的驾驭复杂度；
- 🧐当出现了报错时，阅读测试用例中所编写的行为信息以及错误信息实际上就可以很轻易的进行错误定位到代码段，而后使用 `log_info` 插桩便可以直接找到bug，先前并没有像这次那么注重报错信息，现在发现报错信息，尤其是 `switchyard` 框架所给的报错信息以及测试用例的行为描述实际上就是在直白地告诉自己哪里错了，本来期望是什么结果，但是实际代码却在做什么行为。**总之，跟着报错信息走就能一步一步完成debug。**（真的不是面向OJ编程x）
- 🧐多线程可以将无先后次序依赖的工作分发给不同线程进行并发执行，从而提高执行效率；

- 1 ①.后续可以直接通过`in`来方便判断给定IP地址是否处于表项所在的子网中，减少不必要运算；
- 2 ②.如果端口与主机直接相连，那么`next hop address`会为`0.0.0.0`，而首先会通过查询ARP表来确定下一条的MAC地址，即如果有主机为目标主机且直接与路由器的一个端口相连，那么会在ARP表中查询到对应表项，而目标IP地址本身就在IPv4报文当中，因而`network address`实际上并不是必要的，其子网地址更为重要。

1. 而实际上与实验说明中的示例不同，我将`network address`和`subnet address`通过说明中推荐的方法结合成了一个IPv4的子网地址以供后续方便查询（基于四个键值对字典作为表项的基础上重构获得）即每个表项中的第一个键值对对应了实验说明中的前两个键值对。之所以这样做是因为：[🔗](#)