南京大学本科生实验报告

课程名称：计算机网络　　　　任课教师：黄程远　　　　助教：刘松岳

学院	**数学学院**	专业（方向）	**信息与计算科学**
学号	**221840199**	姓名	**郑鸿鑫**
Email	**221840199@smail.nju.edu.cn**	开始/完成日期	**2024.11.3/2024.11.6**

1. 实验名称

Forwarding Packets

2. 实验目的

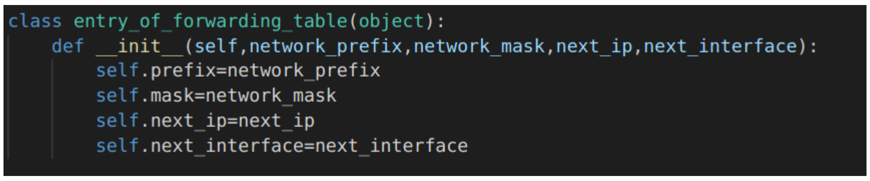
本次实验的目的为实现静态路由和通过 ARP_Request 获得目标 MAC 地址。

3. 实验内容

实现上述功能并通过测试，并在 Mininet 中部署。

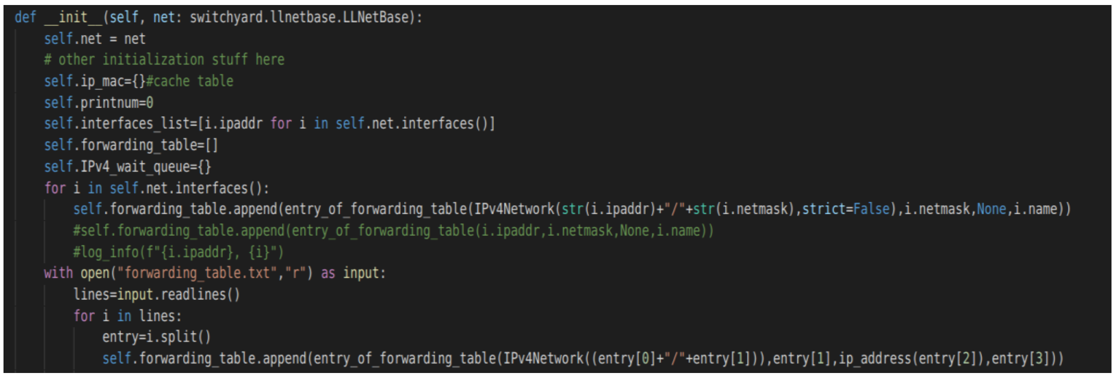
4. 实验结果

（1）实现转发表：



```python
class entry_of_forwarding_table(object):
    def __init__(self,network_prefix,network_mask,next_ip,next_interface):
        self.prefix=network_prefix
        self.mask=network_mask
        self.next_ip=next_ip
        self.next_interface=next_interface
```

如上图，创建转发表的表项对应的类，包括四个属性，即网络前缀，子网掩码，下一跳 IP 地址和对应的接口。

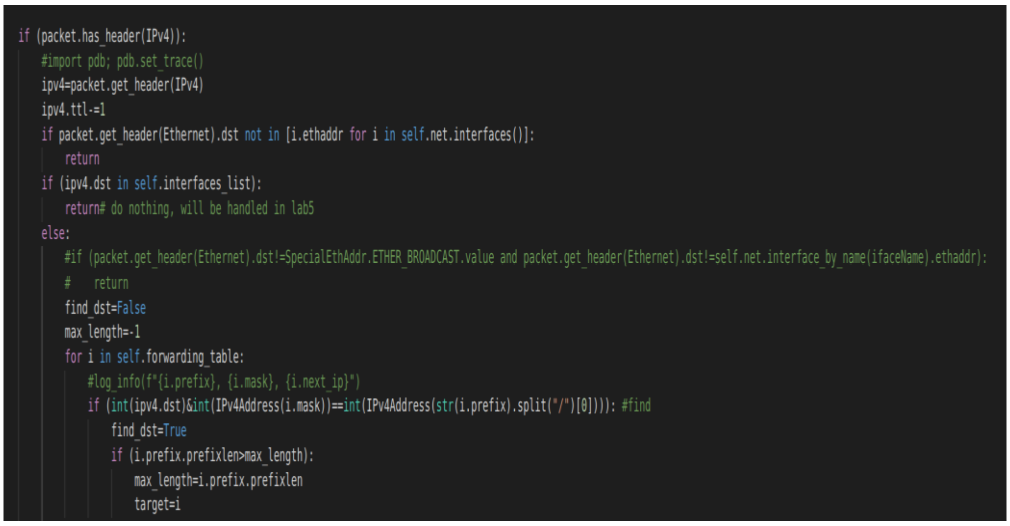


```python
def __init__(self, net: switchyard.llnetbase.LLNetBase):
    self.net = net
    # other initialization stuff here
    self.ip_mac={}#cache table
    self.printnum=0
    self.interfaces_list=[i.ipaddr for i in self.net.interfaces()]
    self.forwarding_table=[]
    self.IPv4_wait_queue={}
    for i in self.net.interfaces():
        self.forwarding_table.append(entry_of_forwarding_table(IPv4Network(str(i.ipaddr)+"/"+str(i.netmask),strict=False),i.netmask,None,i.name))
        #self.forwarding_table.append(entry_of_forwarding_table(i.ipaddr,i.netmask,None,i.name))
        #log_info(f"{i.ipaddr}, {i}")
    with open("forwarding_table.txt","r") as input:
        lines=input.readlines()
        for i in lines:
            entry=i.split()
            self.forwarding_table.append(entry_of_forwarding_table(IPv4Network((entry[0]+"/"+entry[1])),entry[1],ip_address(entry[2]),entry[3]))
```

如图，在初始化 Router 时创建转发表 forwarding_table，并遍历所有接口，加入转发表中，其中网络前缀通过实验手册中的操作拼接得到，对应 ip 地址设为 None，然后再根据 forwarding_table.txt 文件中的内容写入对应表项，**注意要将其中部分数据由 str 类型转为 IPv4Address 类型。**转发表创建完成。

（2）匹配 IP 地址：

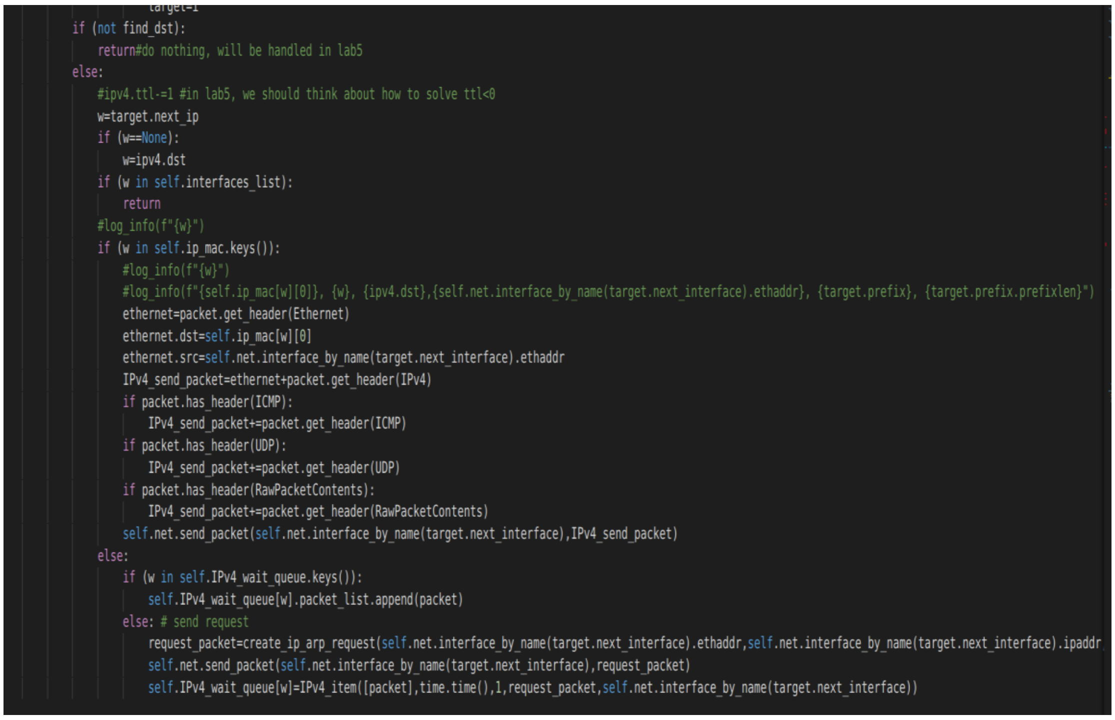
当接收到带有 IPv4 包头的数据包时，将 ttl 减 1，同时判断是否需要丢弃（比如目标地址就是当前 router），若需要转发，则遍历转发表，查看目标 IP 地址是否与表中的表项有匹配（若有多个匹配项，选择前缀最长的表项），匹配的实现通过对目标 IP 地址和子网掩码作与运算并和网络前缀比较完成。匹配成功则准备发送对应报文或缓存该报文。



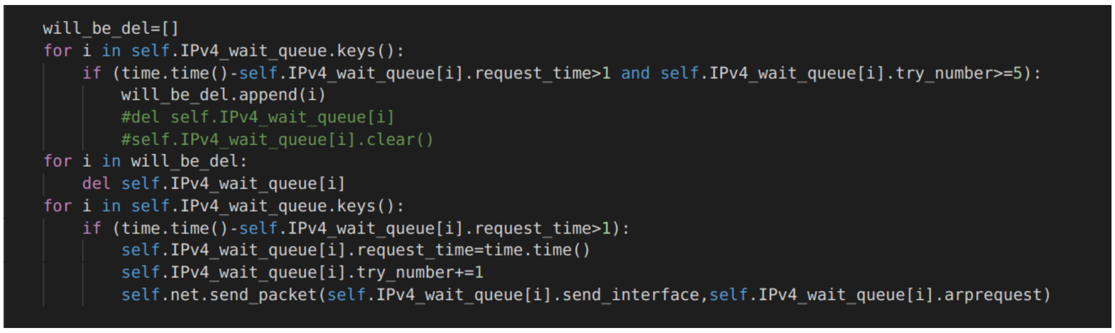
```python
if (packet.has_header(IPv4)):
    #import pdb; pdb.set_trace()
    ipv4=packet.get_header(IPv4)
    ipv4.ttl-=1
    if packet.get_header(Ethernet).dst not in [i.ethaddr for i in self.net.interfaces()]:
        return
    if (ipv4.dst in self.interfaces_list):
        return# do nothing, will be handled in lab5
    else:
        #if (packet.get_header(Ethernet).dst!=SpecialEthAddr.ETHER_BROADCAST.value and packet.get_header(Ethernet).dst!=self.net.interface_by_name(ifaceName).ethaddr):
        #    return
        find_dst=False
        max_length=-1
        for i in self.forwarding_table:
            #log_info(f"{i.prefix}, {i.mask}, {i.next_ip}")
            if (int(ipv4.dst)&int(IPv4Address(i.mask))==int(IPv4Address(str(i.prefix).split("/")[0]))): #find
                find_dst=True
                if (i.prefix.prefixlen>max_length):
                    max_length=i.prefix.prefixlen
                    target=i
```

（3）转发数据包：

在（2）匹配成功的基础上，根据是否能从 ARP 表中得知目标 IP 对应的 MAC 地址来决定是直接转发还是缓存到 IPv4_wait_queue 中，若直接转发，先填好对应的以太网报头，组合后发送；若先缓存，则根据缓存表中是否已有目标 IP，若有，则将数据包直接放入缓存表的对应位置中，若没有，则根据目标 IP 新创建对应表项，并尝试发送 ARP_Request。然后在收发数据包时更新 IPv4_wait_queue，根据超时机制丢弃数据包或尝试重传 ARP_Request。



```python
                    target=i
        if (not find_dst):
            return#do nothing, will be handled in lab5
        else:
            #ipv4.ttl-=1 #in lab5, we should think about how to solve ttl<0
            w=target.next_ip
            if (w==None):
                w=ipv4.dst
            if (w in self.interfaces_list):
                return
            #log_info(f"{w}")
            if (w in self.ip_mac.keys()):
                #log_info(f"{w}")
                #log_info(f"{self.ip_mac[w][0]}, {w}, {ipv4.dst},{self.net.interface_by_name(target.next_interface).ethaddr}, {target.prefix}, {target.prefix.prefixlen}")
                ethernet=packet.get_header(Ethernet)
                ethernet.dst=self.ip_mac[w][0]
                ethernet.src=self.net.interface_by_name(target.next_interface).ethaddr
                IPv4_send_packet=ethernet+packet.get_header(IPv4)
                if packet.has_header(ICMP):
                    IPv4_send_packet+=packet.get_header(ICMP)
                if packet.has_header(UDP):
                    IPv4_send_packet+=packet.get_header(UDP)
                if packet.has_header(RawPacketContents):
                    IPv4_send_packet+=packet.get_header(RawPacketContents)
                self.net.send_packet(self.net.interface_by_name(target.next_interface),IPv4_send_packet)
            else:
                if (w in self.IPv4_wait_queue.keys()):
                    self.IPv4_wait_queue[w].packet_list.append(packet)
                else: # send request
                    request_packet=create_ip_arp_request(self.net.interface_by_name(target.next_interface).ethaddr,self.net.interface_by_name(target.next_interface).ipaddr
                    self.net.send_packet(self.net.interface_by_name(target.next_interface),request_packet)
                    self.IPv4_wait_queue[w]=IPv4_item([packet],time.time(),1,request_packet,self.net.interface_by_name(target.next_interface))
```

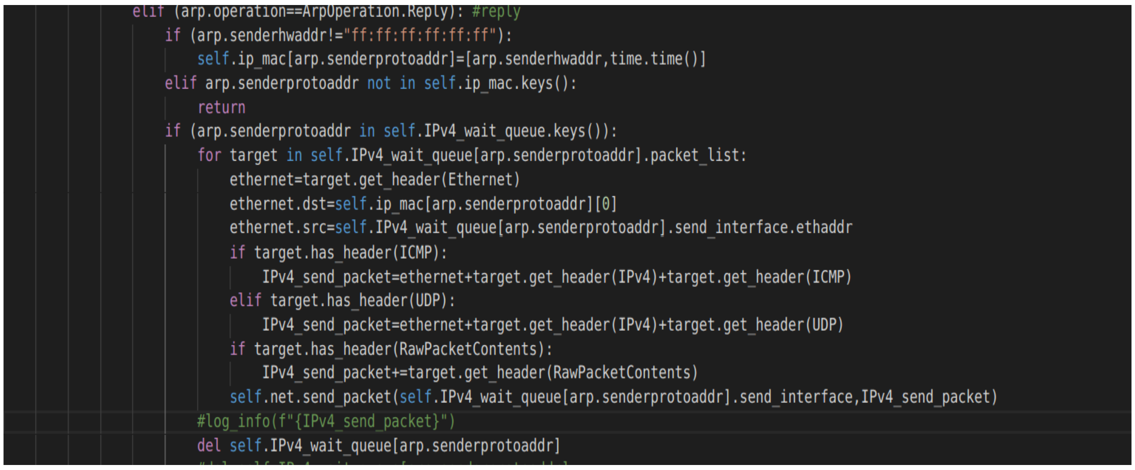


```
will_be_del=[]
for i in self.IPv4_wait_queue.keys():
    if (time.time()-self.IPv4_wait_queue[i].request_time>1 and self.IPv4_wait_queue[i].try_number>=5):
        will_be_del.append(i)
        #del self.IPv4_wait_queue[i]
        #self.IPv4_wait_queue[i].clear()
for i in will_be_del:
    del self.IPv4_wait_queue[i]
for i in self.IPv4_wait_queue.keys():
    if (time.time()-self.IPv4_wait_queue[i].request_time>1):
        self.IPv4_wait_queue[i].request_time=time.time()
        self.IPv4_wait_queue[i].try_number+=1
        self.net.send_packet(self.IPv4_wait_queue[i].send_interface,self.IPv4_wait_queue[i].arprequest)
```

（更新缓存表）

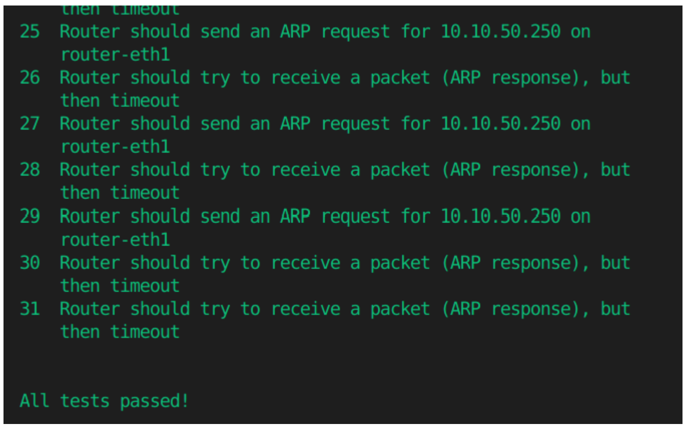
（4）实现 ARP：

在 lab3 中已经实现了对 ARP_Request 的应答（即根据对应情况回复 ARP_Reply 或丢弃），而当收到 ARP_Reply 时，根据报文的内容更新 ARP 缓存表并遍历缓存的 IPv4 报文，寻找目的地址为 ARP_Reply 的发出地址的数据包，并根据 ARP 缓存表发送对应的报文。同时删除缓存表中的表项。



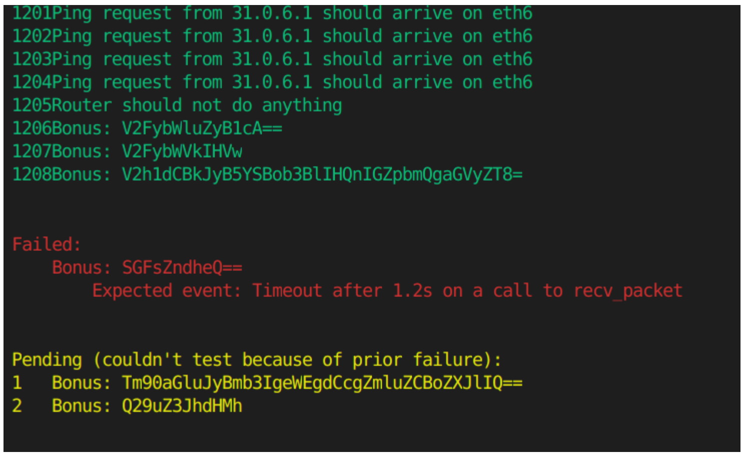
```
elif (arp.operation==ArpOperation.Reply): #reply
    if (arp.senderhwaddr!="ff:ff:ff:ff:ff:ff"):
        self.ip_mac[arp.senderprotoaddr]=[arp.senderhwaddr,time.time()]
    elif arp.senderprotoaddr not in self.ip_mac.keys():
        return
    if (arp.senderprotoaddr in self.IPv4_wait_queue.keys()):
        for target in self.IPv4_wait_queue[arp.senderprotoaddr].packet_list:
            ethernet=target.get_header(Ethernet)
            ethernet.dst=self.ip_mac[arp.senderprotoaddr][0]
            ethernet.src=self.IPv4_wait_queue[arp.senderprotoaddr].send_interface.ethaddr
            if target.has_header(ICMP):
                IPv4_send_packet=ethernet+target.get_header(IPv4)+target.get_header(ICMP)
            elif target.has_header(UDP):
                IPv4_send_packet=ethernet+target.get_header(IPv4)+target.get_header(UDP)
            if target.has_header(RawPacketContents):
                IPv4_send_packet+=target.get_header(RawPacketContents)
            self.net.send_packet(self.IPv4_wait_queue[arp.senderprotoaddr].send_interface,IPv4_send_packet)
        #log_info(f"{IPv4_send_packet}")
        del self.IPv4_wait_queue[arp.senderprotoaddr]
```

（5）测试结果：通过了两个测试文件中大多数测试点，仅 bonus 内容未实现。



```
    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!
```

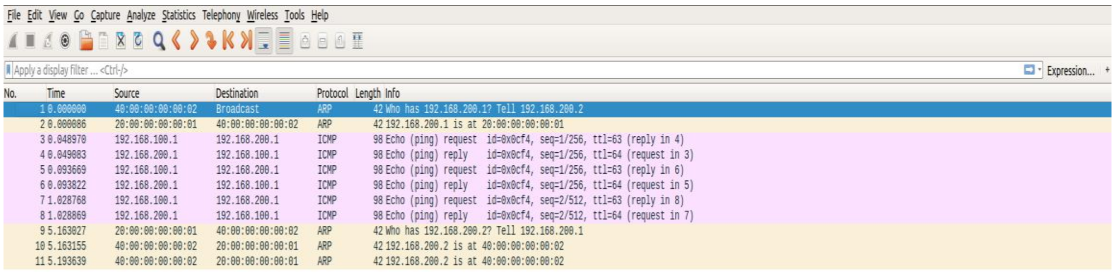


```
1201Ping request from 31.0.6.1 should arrive on eth6
1202Ping request from 31.0.6.1 should arrive on eth6
1203Ping request from 31.0.6.1 should arrive on eth6
1204Ping request from 31.0.6.1 should arrive on eth6
1205Router should not do anything
1206Bonus: V2FybWluZyB1cA==
1207Bonus: V2FybWVkIHVw
1208Bonus: V2h1dCBkJyB5YSBob3BlIHQnIGZpbmQgaGVyZT8=


Failed:
    Bonus: SGFsZndheQ==
        Expected event: Timeout after 1.2s on a call to recv_packet


Pending (couldn't test because of prior failure):
1   Bonus: Tm90aGluJyBmb3IgeWEgdCcgZmluZCBoZXJlIQ==
2   Bonus: Q29uZ3JhdHMh
```

（6）mininet 测试：


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 0.000000   40:00:00:00:00:02  Broadcast          ARP      42 Who has 192.168.200.1? Tell 192.168.200.2
    2 0.000086   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 0.048970   192.168.100.1      192.168.200.1      ICMP     98 Echo (ping) request  id=0x0cf4, seq=1/256, ttl=63 (reply in 4)
    4 0.049083   192.168.200.1      192.168.100.1      ICMP     98 Echo (ping) reply    id=0x0cf4, seq=1/256, ttl=64 (request in 3)
    5 0.093669   192.168.100.1      192.168.200.1      ICMP     98 Echo (ping) request  id=0x0cf4, seq=1/256, ttl=63 (reply in 6)
    6 0.093822   192.168.200.1      192.168.100.1      ICMP     98 Echo (ping) reply    id=0x0cf4, seq=1/256, ttl=64 (request in 5)
    7 1.028768   192.168.100.1      192.168.200.1      ICMP     98 Echo (ping) request  id=0x0cf4, seq=2/512, ttl=63 (reply in 8)
    8 1.028869   192.168.200.1      192.168.100.1      ICMP     98 Echo (ping) reply    id=0x0cf4, seq=2/512, ttl=64 (request in 7)
    9 5.163027   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
   10 5.163155   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
   11 5.193639   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
```

如图，从 server1 向 server2 ping 信号 3 次，先发送 ARP_Request 获取对应的 MAC 地址，接收到 Reply 后发送 ICMP 报文，同时 server2 也会向 server1 发送对应的 ICMP_Reply 报文，因此会看到上图中的结果。

总结与感想

Lab4 远比前几个 Lab 困难，通过测试需要经过大量的调试，还需要对 Lab3 中实现的 ARP 协议理解的更清楚，才能修正之前 Lab 中遗留下的错误。