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	计算机科学与技术系	专业（方向）	计算机科学与技术
学号	202220013	姓名	徐简
Email	161200063@smail.nju.edu.cn	开始/完成日期	2021.4.8-2021.4.21

1. 实验名称

Lab 4: Forwarding Packets

2. 实验目的

1. IP Forwarding Table Lookup
2. Forwarding the Packet and ARP

3. 实验内容、代码与结果

Step 1: Initialize required tables

- Define required classes.
- Initialize a `forward_table` and a `packet_queue`

`fitItem` is what we will store in the `forward_table` containing prefix, net mask, next hop address, and the name of the port through which packets destined to the given network should be forwarded.

`queueItem` is what we will store in the `packet_queue` containing the packet itself and its match in the `forward_table`, i.e. a `fitItem`.

Details of initializing the `forward_queue` are descibed as annotations.

```
class ftItem():
    def __init__(self,p,m,nh,i):
      '''type of p,m,nh are all IPv4Adress'''
        self.prefix=p
        self.mask=m
        self.nexthop=nh
        self.name=i
class queueItem():
```

```
    def __init__(self,pkt,match):
        self.pkt=pkt
        self.rounds=0
        self.time=0
        self.match=match


class Router(object):
    def __init__(self, net: switchyard.llnetbase.LLNetBase):
        self.net = net
        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=[]
        '''packet_queue:pkt waiting to be sent out'''
        self.q=[]
        for i in self.interfaces:
          '''get the prefix using & operation'''
            prefix=IPv4Address(int(i.ipaddr)&int(i.netmask))
            tempNetmask=IPv4Address(i.netmask)
            '''convert to IPv4Adress type'''
            temp=ftItem(prefix,tempNetmask,None,i.name)
            self.forward_table.append(temp)
        file=open("forwarding_table.txt")
        while True:
            l=file.readline()
            if not l:
                break
            else:
                l=l.strip('\n')
                d=l.split(" ")
                prefix=IPv4Address(d[0])
                netmask=IPv4Address(d[1])
                nh=IPv4Address(d[2])
                name=d[3]
                temp=ftItem(prefix,netmask,nh,name)
                self.forward_table.append(temp)

        for a in self.forward_table:
            print(a.prefix," ",a.mask," ",a.nexthop," ",a.name)
```

Step 2: Handle ipv4 packets via forward_table

Iterate the forward_table, find the longest prefix match and add it to the queue defined above.

```
        ipv4=packet.get_header(IPv4)
        if ipv4:
```

```
            head=packet[IPv4]
        ''' ttl decrease'''
            head.ttl-=1
            print("ipv4",head)
            pos=-1
            maxprifixlen=-1
            index=0
            '''iterate find the longest match'''
            for i in self.forward_table:
                if((int(head.dst)&int(i.mask))==int(i.prefix)):
                    netaddr=IPv4Network(str(i.prefix)+"/"+str(i.mask))
                    if netaddr.prefixlen>maxprifixlen:
                        maxprifixlen=netaddr.prefixlen
                        pos=index
                index+=1
            '''pos indicate the result of match'''
            if pos ==-1:
                print("cannot match?")
            else:
                print("paclet enque")
                self.q.append(queueItem(packet,self.forward_table[pos]))
            '''if successfully match, add the packet and its match to the
queue'''
```

Step 3: Handle the packet_queue

Iterate through the `arp_table` to find the mac address of the destination. If find one match, send the packet out and delete it from the queue. If not , send an arp request for the mac address of the destination.

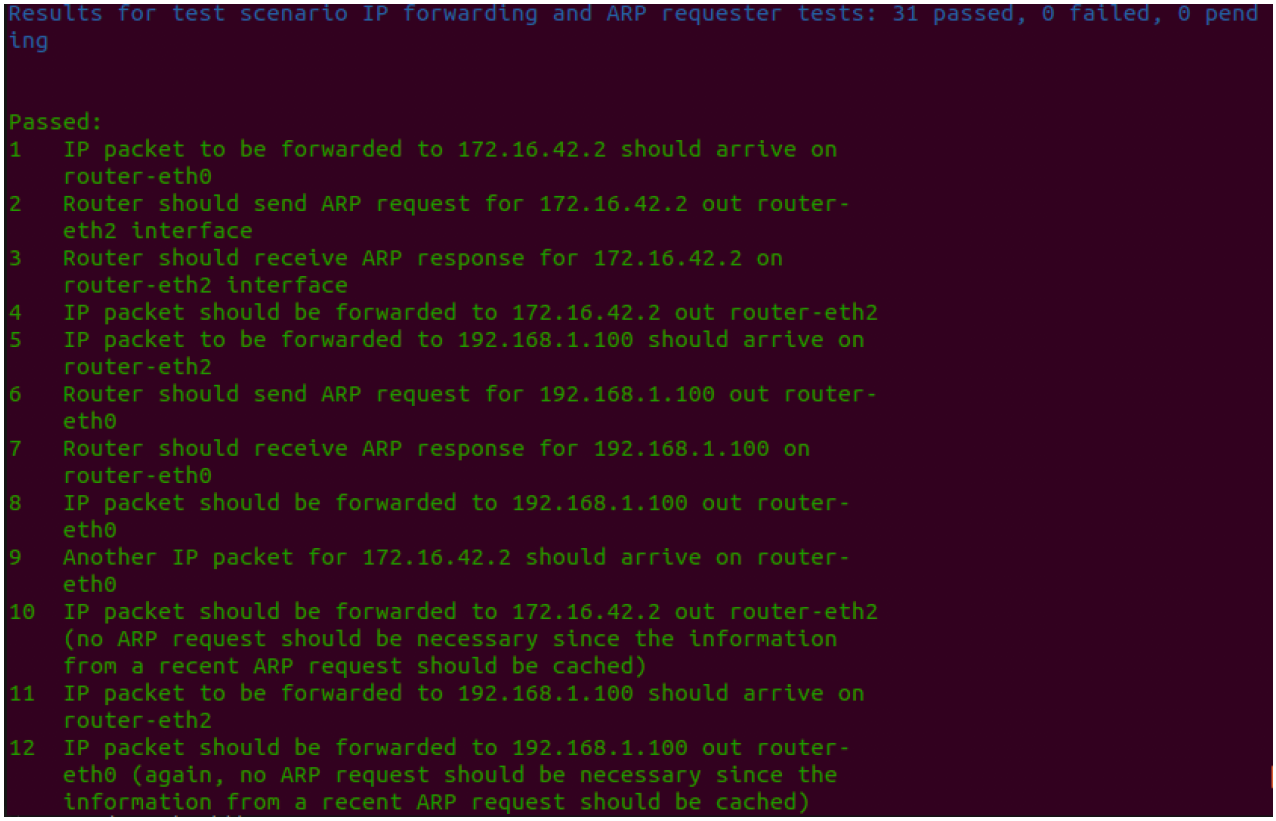
Besides, if the packet cannot find a match after 1 seconds timeout since last request, send an arp request again. After 5 rounds of request and the packet still cannot find a match, drop it.

```
            if len(self.q)!=0:
                for i in self.interfaces:
                    if i.name==self.q[0].match.name:
                        port=i
                if self.q[0].match.nexthop is None:

                    targetip=self.q[0].pkt[IPv4].dst
                    print("None case")
                    print(targetip)
                else:
                    print("not none")
                    targetip=self.q[0].match.nexthop
                flag=0
```

```
                for (k,v) in self.arp_table.items():
                    if targetip==k:
                        self.q[0].pkt[Ethernet].dst=v
                        self.q[0].pkt[Ethernet].src=port.ethaddr
                        print("send pkt found in arptable",port)
                        self.net.send_packet(port,self.q[0].pkt)
                        flag=1
                        del(self.q[0])
                        break
                if flag==0:
                    if self.q[0].rounds>=5:
                        del(self.q[0])
                    else:
                        cur=time.time()
                        if(self.q[0].rounds==0) or (cur-self.q[0].time>1):
                            ether=Ethernet()
                            ether.src=port.ethaddr
                            ether.dst='ff:ff:ff:ff:ff:ff'
                            ether.ethertype=EtherType.ARP

 arp=Arp(operation=ArpOperation.Request,senderhwaddr=port.ethaddr,senderprotoad
dr=port.ipaddr,targethwaddr='ff:ff:ff:ff:ff:ff',targetprotoaddr=targetip)
                            arppkt=ether+arp
                            print("send arp request",port)
                            self.net.send_packet(port,arppkt)
                            self.q[0].rounds+=1
                            self.q[0].time=time.time()
```

Step 4: Test

```
$ swyard -t myrouter2_testscenario.srpy myrouter.py
```

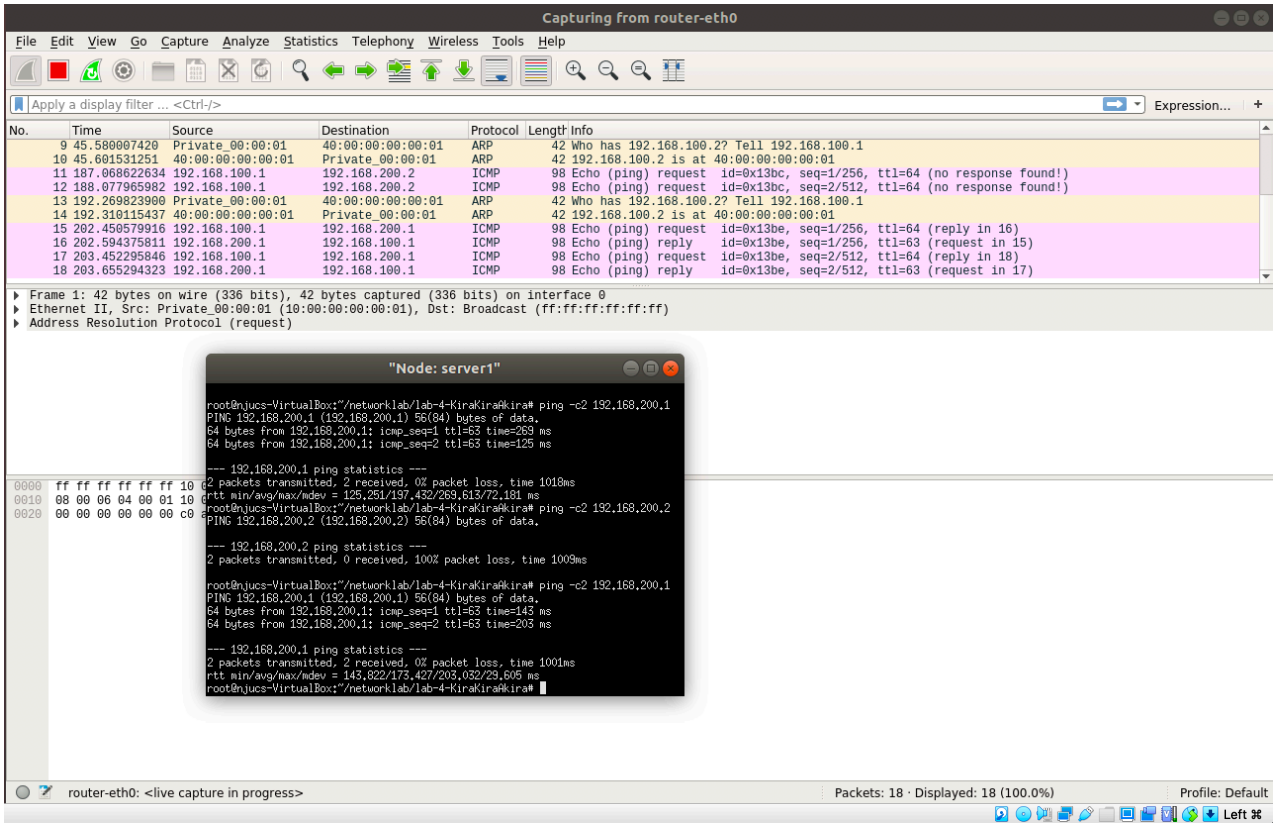


Step 5: Deploy

`server1 ping -c2 192.168.200.2`

`server1 ping -c2 192.168.200.1`

Server1 send out a ICMP packet. The router receives it and add it to the queue. However, the Mac of dst is not in the arp_table. Therefore, port eth1 send an arp request to server2. After getting the arp reply, the ICMP in the queue can be processed. The router receives the ICMP reply packet, having the dst mac address already in the arp_table, then send it out directly.



4. 总结与感想

Lab 4 is not easy to implement because the process logic is much more complex than labs before. I am not familiar with multithreading programming, so I just implement the router logic using one single thread. Multithreading is powerful, so next time I will try.