

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 described as annotations.

```
class fitItem():
    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 enqueue")
            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.nextthop is None:

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

```
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)
- 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-eth0 (again, no ARP request should be necessary since the information from a recent ARP request should be cached)

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.

The image shows a Wireshark packet capture window titled "Capturing from router-eth0". The packet list shows 18 packets. The packet details pane shows the selected packet (No. 18) as an Echo (ping) reply from 192.168.200.1 to 192.168.100.1. The packet bytes pane shows the raw data of the packet.

Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
9	45.580607420	Private_00:00:01	40:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.1
10	45.601531251	40:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:01
11	187.060622634	192.168.100.1	192.168.200.2	ICMP	98	Echo (ping) request id=0x13bc, seq=1/256, ttl=64 (no response found!)
12	188.077965982	192.168.100.1	192.168.200.2	ICMP	98	Echo (ping) request id=0x13bc, seq=2/512, ttl=64 (no response found!)
13	192.269823900	Private_00:00:01	40:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.1
14	202.310115437	40:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:01
15	202.450579916	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x13be, seq=1/256, ttl=64 (reply in 16)
16	202.594375811	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x13be, seq=1/256, ttl=63 (request in 15)
17	203.452295846	192.168.100.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x13be, seq=2/512, ttl=64 (reply in 18)
18	203.655294323	192.168.200.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x13be, seq=2/512, ttl=63 (request in 17)

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)

Inset Terminal Window "Node: server1":

```
root@juces-VirtualBox:/networklab/lab-4-KiraKiraKira# ping -c2 192.168.200.1
PING 192.168.200.1 (192.168.200.1) 56(84) bytes of data:
64 bytes from 192.168.200.1: icmp_seq=1 ttl=63 time=269 ms
64 bytes from 192.168.200.1: icmp_seq=2 ttl=63 time=125 ms

--- 192.168.200.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1013ms
rtt min/avg/max/mdev = 125.261/197.432/269.815/72.181 ms
root@juces-VirtualBox:/networklab/lab-4-KiraKiraKira# ping -c2 192.168.200.2
PING 192.168.200.2 (192.168.200.2) 56(84) bytes of data:
2 packets transmitted, 0 received, 100% packet loss, time 1009ms

--- 192.168.200.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1009ms
root@juces-VirtualBox:/networklab/lab-4-KiraKiraKira# ping -c2 192.168.200.1
PING 192.168.200.1 (192.168.200.1) 56(84) bytes of data:
64 bytes from 192.168.200.1: icmp_seq=1 ttl=63 time=143 ms
64 bytes from 192.168.200.1: icmp_seq=2 ttl=63 time=203 ms

--- 192.168.200.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 143.822/173.427/203.032/29.605 ms
root@juces-VirtualBox:/networklab/lab-4-KiraKiraKira#
```

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.