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	**191220154**	姓名	**张涵之**
Email	**1683762615@qq.com**	开始/完成日期	**2021/5/1-2021/5/2**

1. 实验名称：Lab 4: Forwarding Packets

2. 实验目的：

Continue to improve the router to implement the function of packets forwarding.

3. 实验内容

a) Task 2: IP Forwarding Table Lookup
Build Forwarding Table
Match Destination IP Addresses against Forwarding Table

b) Task 3: Forwarding the Packet and ARP
Send ARP Request and Forward Packet

4. 实验结果

a) Task 2 & Task 3 in test scenario:
Building forwarding table:

```
-------------------------------------------------------------------------------
network address         subnet Address          next hop address        interface
192.168.1.1             255.255.255.0           None                    router-eth0
10.10.0.1               255.255.0.0             None                    router-eth1
172.16.42.1             255.255.255.252         None                    router-eth2
172.16.0.0              255.255.0.0             192.168.1.2             router-eth0
172.16.128.0            255.255.192.0           10.10.0.254             router-eth1
172.16.64.0             255.255.192.0           10.10.1.254             router-eth1
10.100.0.0              255.255.0.0             172.16.42.2             router-eth2
-------------------------------------------------------------------------------
```

Passing test cases:

```
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!
```

Some log info on handling packets and queue entries:

```
11:41:01 2021/05/02     INFO Receive an IPV4 packet.
11:41:01 2021/05/02     INFO Matching dest IP against torwaring table...
11:41:01 2021/05/02     INFO Matching successful.
11:41:01 2021/05/02     INFO Sending ARP request from queue...
11:41:01 2021/05/02     INFO Sending ARP request......
11:41:02 2021/05/02     INFO Sending ARP request from queue...
11:41:02 2021/05/02     INFO Sending ARP request......
11:41:02 2021/05/02     INFO Receive an ARP reply.
11:41:02 2021/05/02     INFO Deleting ARP entry from queue...
---------------------------------
IP               Mac Address
172.16.42.2      30:00:00:00:00:01
192.168.1.100    20:00:00:00:00:01
10.10.1.254      11:22:33:44:55:66
---------------------------------
11:41:02 2021/05/02     INFO Fowarding packet from queue...
11:41:02 2021/05/02     INFO Deleting wait entry from queue...
```

b) Task 2 & Task 3 in Mininet:

Ping server2 from server1:

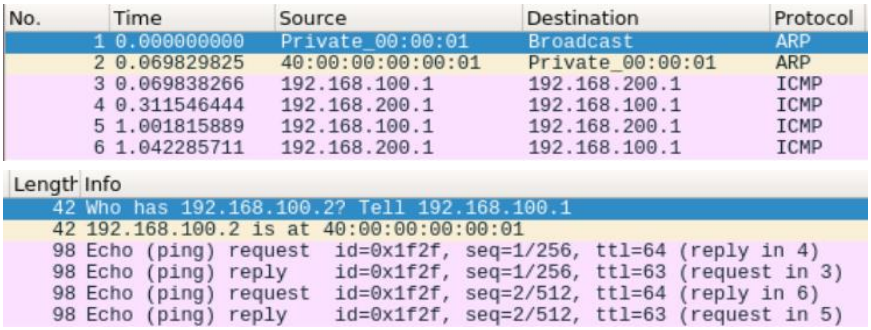
```
root@njucs-VirtualBox:~/switchyard/lab-4-RainTreeCrow# ping -c2 192.168.200.1
PING 192.168.200.1 (192.168.200.1) 56(84) bytes of data.
64 bytes from 192.168.200.1: icmp_seq=1 ttl=63 time=311 ms
64 bytes from 192.168.200.1: icmp_seq=2 ttl=63 time=40.4 ms

--- 192.168.200.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 40.497/176.053/311.609/135.556 ms
root@njucs-VirtualBox:~/switchyard/lab-4-RainTreeCrow#
```

Router log info:

```
11:42:12 2021/05/02     INFO Receive an ARP request.
11:42:12 2021/05/02     INFO Sending ARP reply...
---------------------------------
IP              Mac Address
192.168.100.1   10:00:00:00:00:01
---------------------------------
11:42:12 2021/05/02     INFO Receive an IPV4 packet.
11:42:12 2021/05/02     INFO Matching dest IP against torwaring table...
11:42:12 2021/05/02     INFO Matching successful.
11:42:12 2021/05/02     INFO Sending ARP request from queue...
11:42:12 2021/05/02     INFO Sending ARP request......
11:42:12 2021/05/02     INFO Receive an ARP reply.
11:42:12 2021/05/02     INFO Deleting ARP entry from queue...
---------------------------------
IP              Mac Address
192.168.100.1   10:00:00:00:00:01
192.168.200.1   20:00:00:00:00:01
---------------------------------
11:42:12 2021/05/02     INFO Fowarding packet from queue...
11:42:12 2021/05/02     INFO Deleting wait entry from queue...
11:42:12 2021/05/02     INFO Receive an IPV4 packet.
11:42:12 2021/05/02     INFO Matching dest IP against torwaring table...
11:42:12 2021/05/02     INFO Matching successful.
11:42:12 2021/05/02     INFO Fowarding packet from queue...
11:42:12 2021/05/02     INFO Deleting wait entry from queue...
11:42:13 2021/05/02     INFO Receive an IPV4 packet.
11:42:13 2021/05/02     INFO Matching dest IP against torwaring table...
11:42:13 2021/05/02     INFO Matching successful.
11:42:13 2021/05/02     INFO Fowarding packet from queue...
11:42:13 2021/05/02     INFO Deleting wait entry from queue...
11:42:13 2021/05/02     INFO Receive an IPV4 packet.
11:42:13 2021/05/02     INFO Matching dest IP against torwaring table...
11:42:13 2021/05/02     INFO Matching successful.
11:42:13 2021/05/02     INFO Fowarding packet from queue...
11:42:13 2021/05/02     INFO Deleting wait entry from queue...
11:42:18 2021/05/02     INFO Receive an ARP request.
11:42:18 2021/05/02     INFO Sending ARP reply...
---------------------------------
IP              Mac Address
192.168.100.1   10:00:00:00:00:01
192.168.200.1   20:00:00:00:00:01
---------------------------------
```

From the log info I can infer that the interface received four IPV4 packets, two ARP requests and one ARP reply, all of which are dealt with accordingly.
Captured file on server1-eth0:



No.	Time	Source	Destination	Protocol
1	0.000000000	Private_00:00:01	Broadcast	ARP
2	0.069829825	40:00:00:00:00:01	Private_00:00:01	ARP
3	0.069838266	192.168.100.1	192.168.200.1	ICMP
4	0.311546444	192.168.200.1	192.168.100.1	ICMP
5	1.001815889	192.168.100.1	192.168.200.1	ICMP
6	1.042285711	192.168.200.1	192.168.100.1	ICMP

Length	Info
42	Who has 192.168.100.2? Tell 192.168.100.1
42	192.168.100.2 is at 40:00:00:00:00:01
98	Echo (ping) request id=0x1f2f, seq=1/256, ttl=64 (reply in 4)
98	Echo (ping) reply id=0x1f2f, seq=1/256, ttl=63 (request in 3)
98	Echo (ping) request id=0x1f2f, seq=2/512, ttl=64 (reply in 6)
98	Echo (ping) reply id=0x1f2f, seq=2/512, ttl=63 (request in 5)

5. 核心代码

a) Task 2: IP Forwarding Table Lookup
Building Forwarding Table:
The forwarding table entry class is defined as below:

```
class FwEntry():
    def __init__(self, nwaddr, snaddr, nhaddr, iface):
        self.networkAddr = nwaddr
        self.subnetAddr = snaddr
        self.nextHopAddr = nhaddr
        self.interface = iface

    def print_entry(self):
        print(self.networkAddr, "\t\t", self.subnetAddr, "   \t",
            self.nextHopAddr, "          \t", self.interface
        )
```

The router's forwarding table is initialized together with the router itself. The entries are added from the router's interfaces and the txt file.

```
class Router(object):
    def __init__(self, net: switchyard.llnetbase.LLNetBase):
        self.net = net
        # other initialization stuff here
        self.interfaces = net.interfaces()
        self.arpTable = {}
        self.forwardingTable = []
        self.waitQueue = WaitQueue(net)
        for iface in self.interfaces:
            self.forwardingTable.append(
                FwEntry(iface.ipaddr, iface.netmask, None, iface.name)
            )
        with open("forwarding_table.txt") as fwFile:
            for line in fwFile:
                entry = line.split()
                if entry:
                    self.forwardingTable.append(
                        FwEntry(entry[0], entry[1], entry[2], entry[3])
                    )
        self.print_fwTable()
```

Match Destination IP Addresses against Forwarding Table:
this part of logic is defined in the handle_packet function:

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

we skip the part that deals with ARP packets and move on to IPV4:

```
    elif ipv4 is not None:
        log_info("Receive an IPV4 packet.")
        ipv4.ttl -= 1
```

Decrement the TTL field in the IP header by 1

```
        if ipv4.dst in [iface.ipaddr for iface in self.interfaces]:
            log_info("Packet is for the router itself.")
            #To be handled at a larger stage
        else:
            log_info("Matching dest IP against torwaring table...")
            maxlen = 0
            match = None
            for fwEntry in self.forwardingTable:
                netAddr = IPv4Network(str(fwEntry.networkAddr) +
                    "/" + str(fwEntry.subnetAddr), False
                )
                if ipv4.dst in netAddr:
                    pflen = netAddr.prefixlen
                    if pflen > maxlen:
                        match = fwEntry
                        maxlen = pflen
```

If packet is for the router itself (i.e., destination address is an address of one of the router's interfaces), drop/ignore it. (to be handled at a later stage.)
While matching destination IP address against the forwarding table, longest prefix match is used, the entry with the max prefix length is picked out
If there is no match in the table, drop. (to be handled in a later stage.)

```
            if match is not None:
```

```
            else:
                log_info("Matching failed. Packet dropped.")
                #To be handled in a later stage
```

b) Task 3: Forwarding the Packet and ARP

Here is how the router behaves after a successful match:

```
            if match is not None:
                log_info("Matching successful.")
                iface = match.interface
                for interface in self.interfaces:
                    if interface.name == iface:
                        routerIface = interface
                        break
                packet[Ethernet].src = routerIface.ethaddr
                if match.nextHopAddr is not None:
                    self.waitQueue.add_entry(WEntry(packet,
                            ip_address(match.nextHopAddr), routerIface
                        ), self.arpTable
                    )
                else:
                    self.waitQueue.add_entry(WEntry(
                            packet, ipv4.dst, routerIface
                        ), self.arpTable
                    )
```

Create a new Ethernet header for the IP packet to be forwarded. The next hop host is either the destination host if the destination address is directly reachable through one of the router interfaces (i.e., the subnet that the destination address belongs to is directly connected to a router interface), or IP address on a router through which the destination is reachable. In the initialization strategy, those entries we add from the router's interfaces do not have a next hop address, they are the ones "directly reachable through one of the interfaces", through this we can clearly and easily tell the two separate cases apart

The new Ethernet header should also include new source address, that is the ethernet address of the matched interface, the header is modified accordingly

At this point, we do not know yet whether an ARP request is needed before we send the packet, so we add an entry to the wait queue defined as below:
Here is how the wait entry is defined, when the router finds an entry's next hop address in the ARP table, it informs the entry of the matching destination MAC address, and it forwards the Ethernet packet accordingly:

```
class WEntry():
    def __init__(self, pkt, nhaddr, iface):
        self.packet = pkt
        self.nextHopAddr = nhaddr
        self.interface = iface

    def forward(self, dstmac, net):
        self.packet[Ethernet].dst = dstmac
        net.send_packet(self.interface.name, self.packet)
```

Another class defined is the ARP entry, for several different entries in the wait queue may "want" to send ARP requests with the same target address, the ARP entry has the variable lastArpRequest, which records the last time a request is send inquiring about the address, and countArpRequest that records how many ARP requests are sent (so far) inquiring about this address.

```
class ArpEntry:
    def __init__(self, nhaddr, iface):
        self.nextHopAddr = nhaddr
        self.interface = iface
        self.lastArpRequest = 0
        self.countArpRequest = 0

    def arp_request(self, net):
        log_info("Sending ARP request......")
        arpRequest = create_ip_arp_request(
            self.interface.ethaddr,
            self.interface.ipaddr,
            self.nextHopAddr
        )
        self.lastArpRequest = time.time()
        self.countArpRequest += 1
        net.send_packet(self.interface.name, arpRequest)
```

Each time an ARP request is sent, the two variables are updated
Here is the class that helps maintain the two entries above, waitQueue records the wait entries, arpQueue records the ARP entries, and the list arpAddr is used to remember which address are already in the ARP queue, so that we only need to traverse a list to determine whether the ARP entry is present.

```
class WaitQueue():
    def __init__(self, net):
        self.waitQueue = []
        self.net = net
        self.arpQueue = []
        self.arpAddr = []

    def add_entry(self, entry, arp):
        self.waitQueue.append(entry)
        arpReqAddr = entry.nextHopAddr
        if arpReqAddr not in arp.keys() and arpReqAddr not in self.arpAddr:
            self.arpQueue.append(ArpEntry(arpReqAddr, entry.interface))
            self.arpAddr.append(arpReqAddr)
```

The function add_entry adds to the wait queue every time it is called, but the ARP queue is appended only if the target address is NOT already in the router's ARP table or the wait queue's ARP address list

```
def remove_entry(self, arpRepAddr):
    for arpReq in self.arpQueue[:]:
        if arpReq.nextHopAddr == arpRepAddr:
            self.arpQueue.remove(arpReq)
            log_info("Deleting answered ARP entry from queue...")
    if arpReq in self.arpAddr:
        self.arpAddr.remove(arpRepAddr)
```

The function remove_entry deletes the corresponding entry in the ARP queue and removes the address from the ARP address list, the function is called every time the router receives an ARP reply:

```
elif arp.operation == 2: #Reply
    log_info("Receive an ARP reply.")
    self.waitQueue.remove_entry(arp.senderprotoaddr)
```

Here is the update_queue function, called in the while loop of the router's start function before the try statement that captures packets:

```
while True:
    self.waitQueue.update_queue(self.arpTable)
```

```
def update_queue(self, arp):
    for entry in self.waitQueue[:]:
        if entry.nextHopAddr in arp.keys():
            log_info("Fowarding packet from queue...")
            entry.forward(arp[entry.nextHopAddr], self.net)
            self.waitQueue.remove(entry)
            log_info("Deleting forwarded wait entry from queue...")
    for arpReq in self.arpQueue[:]:
        if time.time() - arpReq.lastArpRequest >= 1:
            if arpReq.countArpRequest < 5:
                log_info("Sending ARP request from queue...")
                arpReq.arp_request(self.net)
            else:
                self.arpQueue.remove(arpReq)
                if arpReq in self.arpAddr:
                    self.arpAddr.remove(arpReq)
                log_info("Deleting timeout ARP entry from queue...")
                for entry in self.waitQueue[:]:
                    if entry.nextHopAddr == arpReq.nextHopAddr:
                        self.waitQueue.remove(entry)
                        log_info("Deleting timeout wait entry from que
```

Each time the function is called, it goes through each entry in the wait queue finding whether the next hop address is in the ARP table, if there happen to be a new item (after receiving an ARP packet) that matches the address, the entry calls its forward function, and the entry is removed from the queue. The queue is designed using a list, so the entries in it are arranged according to the order in which they are added. Thus, the router buffers multiple packets sharing the same ARP request, and upon receiving the corresponding ARP reply these packets are forwarded in the order they arrived to the router

For each ARP entry in the ARP queue, if the time elapsed since the last time a request is sent is more then one second, and the same request have been sent for less than five times, it calls arp_request to resend. If the same request has been sent for more than five times, the ARP entry and each wait entry with the same target next hop address are removed from the queues they are in

6. 总结与感想

a) Next time I should read through the entire manual (including the FAQ) before I start coding. This time I did not take into account the case where several wait entries might share the same next hop and send redundant ARP requests, so I had to write another class to store and maintain ARP information. If I had taken it into consideration, I should have written a different data structure, perhaps I would put ARP requests into a wait queue and store the packets to be forwarded inside the ARP entry. The structure seems simpler and more efficient to me.

b) At first, I did not realize the differences between the addresses I read from the txt file (strings) and the IP addresses in the net. Python is a language in which the data types can be very obscure and confusing, in a class we can use _str_ function, so even though an object can be printed, it does not necessarily mean it is a string, and even though the two objects appear identical when you print them out, it does not necessarily mean they are equal. I found a simple (stupid) way to debug, that is to print (x == y) or (Type(x)) all over the place. A better idea may be reading the definition of each class more patiently.

c) Pdb is a good tool. Before this lab I did not even know how to use “where” to find out where my code had stop executing. Perhaps it was because in the first three labs I did not come across some bugs that are really confusing and hard to locate, but writing code blindly without and test methods is definitely not a good habit. I shall learn to use the tools myself instead of waiting for someone to ask questions in the QQ group and some TA come to their rescue.