



# Lab 4: Forwarding Packets

181180050 孔孟荀

---

## Task 1: Preparation

略

---

## Task 2: IP Forwarding Table Lookup

### Build Forwarding Table

由guide, 知道forwarding table的内容来自两方面, 一方面是router自己的net.interfaces的内容, 另一方面来自forwarding\_table.txt, 由于对python文件读写不是很熟, 后者花费了我一定的时间。

*show how you implement the logic of building IP forwarding table and matching the destination IP addresses.*

- implement the logic of building ip forwarding table:  
在router类的init初始化方法中, 进行如下定义

```
self.forwarding_table = []  
self.set_forwardingtable()
```

可见我把forwarding table设为了一个列表, 并且为他写了一个初始化的函数

```
def set_forwardingtable(self):
    for itface in self.net.interfaces():
        self.forwarding_table.append([
            itface.ipaddr, itface.netmask, "0.0.0.0", itface.name])
    with open("forwarding_table.txt", "r") as f:
        for line in f:
            substr = line.split()
            self.forwarding_table.append([
                substr[0], substr[1], substr[2], substr[3]])
```

在这个set函数里，从router的接口、forwarding\_table.txt文件两个来源，完成了对forwarding\_table的初始化，注意对于router的接口，它的下一跳按照教程说的，设定为了0.0.0.0

- matching the destination ip addresses :

```
if ipv4.dst not in [itface.ipaddr for itface in self.net.interfaces()]:
    pos = 0
    maxnum = 0
    i = 0
    find = False
    for entry in self.forwarding_table:
        prefixnet = IPv4Network(str(entry[0])+"/"+str(entry[1]),False)
        destaddr= IPv4Address(ipv4.dst)
        matches = destaddr in prefixnet
        if matches == True :
            if prefixnet.prefixlen > maxnum:
                maxnum = prefixnet.prefixlen
                pos = i
                find = True # pos indicates the entry which has longest match ,can get ipaddress via it
        i += 1
```

如上代码段

- 首先判断ip包的目的地地址不是router的接口，按照要求，如果是则丢弃这个包

If packet is for the router itself (i.e., destination address is an address of one of the router's interfaces), also drop/ignore the packet. (We'll also handle this better at a later stage.)

- 接着遍历整个表，设定一个最长匹配值maxnum，如果当前表项和dst匹配，而且匹配值大于maxnum，说明当前这个匹配更长，记住该表项的位置
- 通过一个find 的flag，来判断有没有匹配项，如果没有，按照要求，依然丢弃

If there is no match in the table, just drop the packet. (We'll handle this better in a later stage of creating the router.)

## Task 3: Forwarding the Packet and ARP

logic :

在task2里，已经完成了：实现forwarding table，并且找到最长匹配

接下来按照顺序说说还要做什么，以及我是怎么做的

- Decrement the TTL field in the IP header by 1
- 判断当前包的目的地地址的mac地址知不知道，这依赖于lab3的arp表，
  - 1.如果知道，那么直接换一下ethernet 头地址，并且转发一下即可，
  - 2.如果不知道，则需要向前面匹配找到的端口发一个arp request，请求这个mac 地址。并且把当前的这个包，按照ip地址索引（因为faq里说以后可能来相同ip地址的包），加入到一个类对象ippacketinqueue的字典ipv4packetqueue中。这个类对象ippacketinqueue的第一项是一个列表，记录了当前所有等待arp request 并且没发送的ip地址相同（都为字典索引）的包，剩下的项是时间戳，retrynum（发送的arp request的数量），以及要发的arp request 包。
  - 3.当然，如果虽然不知道目的的mac 地址，但是当前的这个目的ip地址经过字典ipv4packetqueue检索，发现已经有对应的类对象了，说明之前已经来过相同目的地地址的包了，只要加入到对应类对象的列表中，继续等待即可

```

if find == True: # find == False means no match ,will be dropped
    nexthopip = self.forwarding_table[pos][2]
    if nexthopip == "0.0.0.0":
        nexthopip=ipv4.dst
    if nexthopip in self.arp_table:
        e = packet.get_header(Ethernet)
        intf = self.forwarding_table[pos][3]
        e.src = self.net.interface_by_name(intf).ethaddr
        e.dst = self.arp_table[nexthopip][0]
        sendpkt = e + \
            packet.get_header(IPv4) + packet.get_header(ICMP)
        self.net.send_packet(intf, sendpkt)
    else: # nexthopip not in arptable,need to be put into a queue and send arp request
        if nexthopip in self.ipv4packetqueue:
            self.ipv4packetqueue[ip_address(nexthopip)].ippacket.append(packet)
        else:
            intf = self.forwarding_table[pos][3]
            arprequest = create_ip_arp_request(self.net.interface_by_name(
                intf).ethaddr, self.net.interface_by_name(intf).ipaddr, nexthopip)
            self.net.send_packet(intf, arprequest)
            ippacketqueue = ippacketinqueue(
                [packet], time.time(), 1, arprequest, intf)
            self.ipv4packetqueue[ip_address(nexthopip)] = ippacketqueue

```

- 发送了arp request以后，就要做接受arp reply的任务，lab3里已经实现了arp table以及接受arp request，这里做进一步的细化

```

if packet.has_header(Arp) == True:
    arp = packet.get_header(Arp)
    self.arp_table[arp.senderprotoaddr] = [
        arp.senderhwaddr, time.time()]
    self.update_table(10)
    if arp.operation == 1: # arp request
        for itface in self.net.interfaces():
            if arp.targetprotoaddr == itface.ipaddr:
                pkt = create_ip_arp_reply(
                    itface.ethaddr, arp.senderhwaddr, arp.targetprotoaddr, arp.senderprotoaddr)
                self.net.send_packet(itfaceName, pkt)
    elif arp.operation == 2:
        ip = arp.senderprotoaddr
        if ip in self.ipv4packetqueue:
            for pcket in self.ipv4packetqueue[ip].ippacket:
                e = pcket.get_header(Ethernet)
                e.dst = arp.senderhwaddr
                e.src = arp.targethwaddr
                sendpkt = e + \
                    pcket.get_header(IPv4) + pcket.get_header(ICMP)
                self.net.send_packet(itfaceName, sendpkt)
            self.ipv4packetqueue.pop(ip)

```

当收到arp reply, 记录下arp包的源地址的mac地址, 作为目的mac地址, 完善我们的以太网头。由目的地址的ip地址为索引, 找到词典ipv4packetqueue中对应的项里的列表里的所有包, 按照顺序添加上述以太网头以后逐一发送, 并且清除词典里的这一项即可。

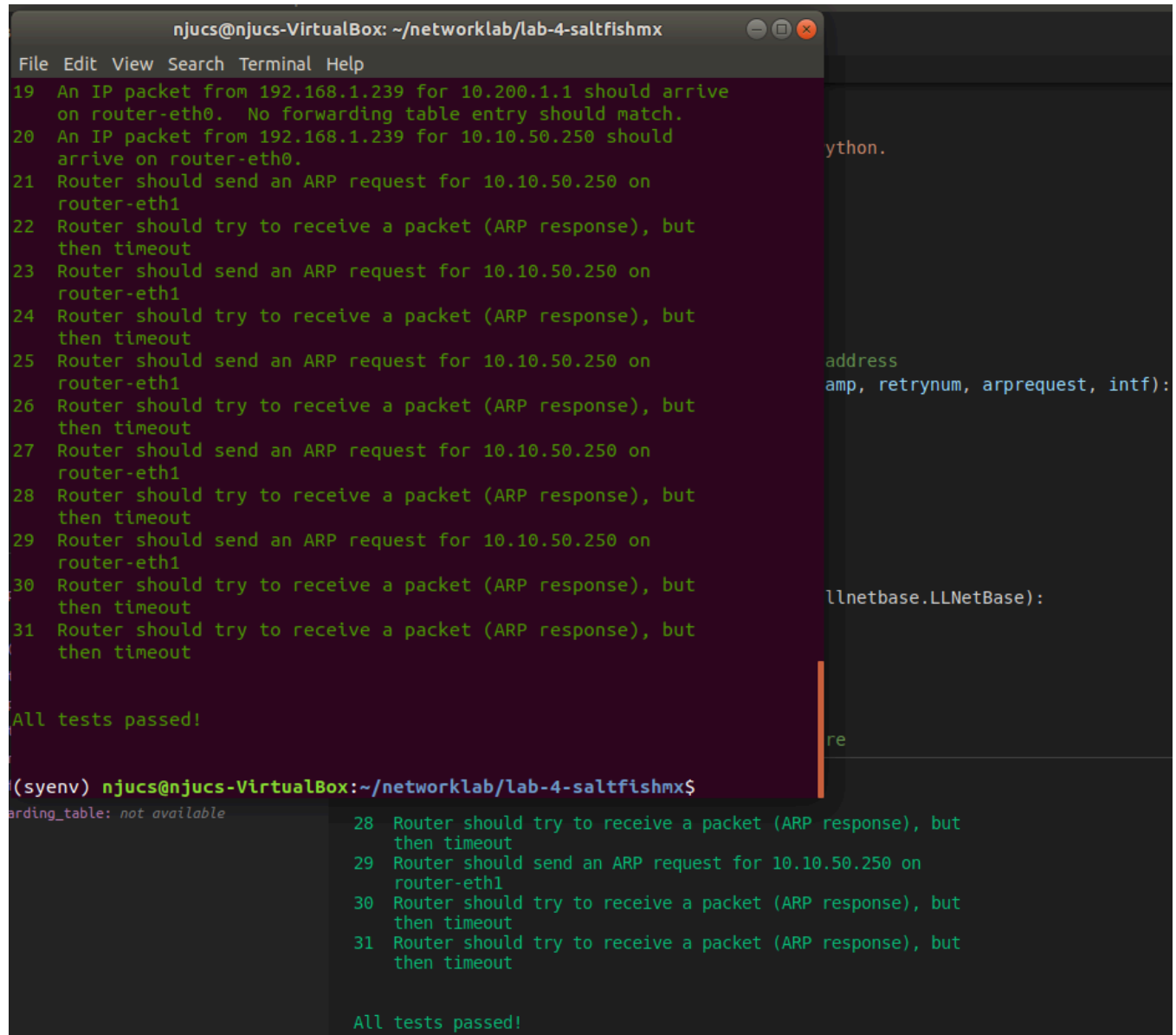
- 最后还要实现一个在主循环中时刻依照时间戳遍历词典ipv4packetqueue里的每一项, 并且把超过限时1s的项里的arp request包重新发送, 亦或者达到retry num即删除的功能, 为此专门实现了一个函数

```

def checkqueue(self, duration=1):
    for key in list(self.ipv4packetqueue):
        flag = time.time() - self.ipv4packetqueue[key].timestamp > duration
        if flag == True:
            if(self.ipv4packetqueue[key].retrynum >= 5):
                self.ipv4packetqueue.pop(key)
            else:
                arprequest = self.ipv4packetqueue[key].arprequest
                intf = self.ipv4packetqueue[key].intf
                self.net.send_packet(intf, arprequest)
                self.ipv4packetqueue[key].timestamp = time.time()
                self.ipv4packetqueue[key].retrynum += 1

```

testing:



```
njucs@njucs-VirtualBox: ~/networklab/lab-4-saltfishmx
File Edit View Search Terminal Help
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!

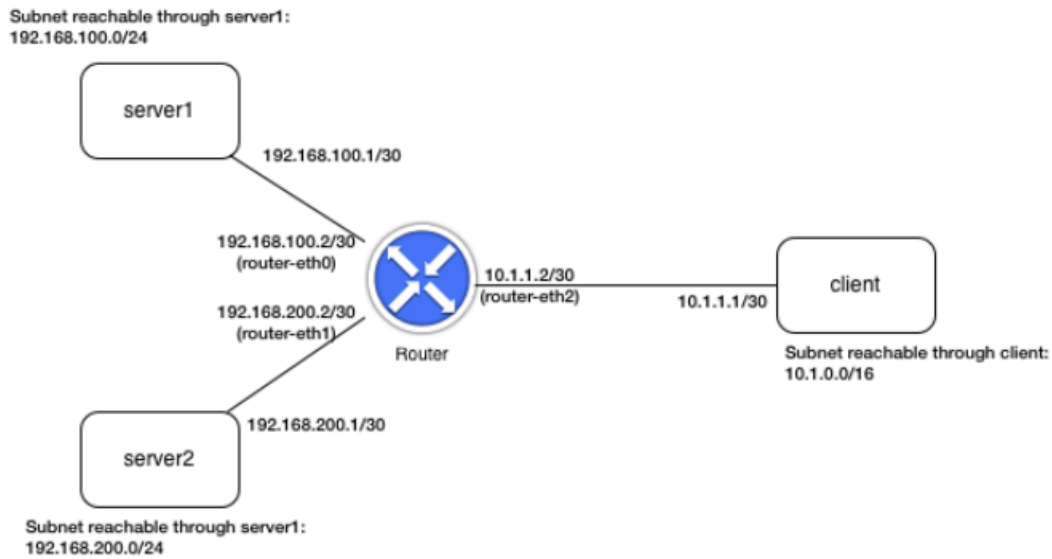
(syenv) njucs@njucs-VirtualBox:~/networklab/lab-4-saltfishmx$
arding_table: not available
28 Router should try to receive a packet (ARP response), but
   then timeout
29 Router should send an ARP request for 10.10.50.250 on
   router-eth1
30 Router should try to receive a packet (ARP response), but
   then timeout
31 Router should try to receive a packet (ARP response), but
   then timeout

All tests passed!
```

deploying

决定ping client from server1

由示意图



要从server1 ping client, 显然可以观察router-eth0端口

```
wireshark -i router-eth0
```

抓包结果如下

| No. | Time        | Source            | Destination      | Protocol | Length | Info  |
|-----|-------------|-------------------|------------------|----------|--------|---|
| 1   | 0.000000000 | Private_00:00:01  | Broadcast        | ARP      | 42     | Who has 192.168.100.2? Tell 192.168.100.1                     |
| 2   | 0.067422791 | 40:00:00:00:00:01 | Private_00:00:01 | ARP      | 42     | 192.168.100.2 is at 40:00:00:00:00:01                         |
| 3   | 0.067432790 | 192.168.100.1     | 10.1.1.1         | ICMP     | 98     | Echo (ping) request id=0x0e67, seq=1/256, ttl=64 (reply in 4) |
| 4   | 0.375176603 | 10.1.1.1          | 192.168.100.1    | ICMP     | 98     | Echo (ping) reply id=0x0e67, seq=1/256, ttl=63 (request in 3) |
| 5   | 1.001327059 | 192.168.100.1     | 10.1.1.1         | ICMP     | 98     | Echo (ping) request id=0x0e67, seq=2/512, ttl=64 (reply in 6) |
| 6   | 1.212507124 | 10.1.1.1          | 192.168.100.1    | ICMP     | 98     | Echo (ping) reply id=0x0e67, seq=2/512, ttl=63 (request in 5) |

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)  
 Destination: Broadcast (ff:ff:ff:ff:ff:ff)  
 Address: Broadcast (ff:ff:ff:ff:ff:ff)  
 ... = LG bit: Locally administered address (this is NOT the factory default)

```

"Node: server1"
root@njucs-VirtualBox:~/networklab/lab-4-saltfishmx# ping -c2 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data:
64 bytes from 10.1.1.1: icmp_seq=1 ttl=63 time=375 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=63 time=211 ms
--- 10.1.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 211.252/293.225/375.199/81.975 ms
root@njucs-VirtualBox:~/networklab/lab-4-saltfishmx#
  
```

可见router成功的实现了arp request 的发送 和arp reply 的接收功能, 并在此基础上将icmp包修

改以太网头进行了转发，使得192.168.100.1的server1能够ping到10.1.1.1的client，从而达到了目的

**说明实现成功**

---

## 4.实验思考和感想

写这次实验过程中，遇到的问题比以前多，后来发现可以用vscode配置以后进行调试，一下子方便了很多。

- 遇到的主要问题是在一处判断时，“192.168.1.10”这样的地址 `in IPv4Address(192.168.1.10)` 会判断false，源头在于forwarding table的两个来源router接口 以及 txt文件得到的表项类型并不相同，最后翻文档找到一个方法 `ip_address()` 可以把字符串转化成 `IPv4Address` 格式，成功解决。
  - 还有checkqueue 函数在每次主循环里都要调用，一开始只写在了主循环的try里 `recv packet`函数后面，结果发现如果没收到包会直接进入except的判断，而不会调用这个函数，后来进行了一些修改。
- 

完