Lab 5: Respond to ICMP

181180050 孔孟荀

Task 1: Preparation

略

Task 2: Responding to ICMP echo requests

> you should first check whether the IP destination address is the same as one of the addresses assigned to one of the router's interfaces.

lab4中已经完成了对于ipv4包，如果目的地不是router的interface的转发，对于是interface的则丢弃，这里只要稍作修改,对于目的地是router的interface的icmp request包，构造一个icmp应答包，以request包的src为dst进行前缀匹配并且转发。

```
#构造reply包
def icmpreply(self, packet, ttl=64):
    eo = packet.get_header(Ethernet)
    e = copy.deepcopy(eo)

    io = packet.get_header(IPv4)
    ip = copy.deepcopy(io)
    ip.src = io.dst
    ip.dst = io.src
    ip.ttl=ttl

    icmp = packet.get_header(ICMP)
    ic = ICMP()
    ic.icmptype = 0  #reply
    ic.icmpdata.data = icmp.icmpdata.data
    ic.icmpdata.sequence = icmp.icmpdata.sequence

    #ip.protocol = IPProtocol.ICMP
    sendpkt = e + ip + ic
    return sendpkt
```

遇到的问题：

1. 根据icmp request 包生成 icmp reply 包的时候发现如果先复制icmpdata里的data 和 sequence内容再修改 icmp type，会导致前面复制的icmpdata的内容丢失
2. 上次做lab4的时候没有把前缀匹配、forward等内容做成函数，导致这次一开始也没想改成函数，后来发现重复代码太多读起来很不方便才修改：

```
#前缀匹配
def longest_prefix(self, ipv4, flag=1):
    pos = 0
    maxnum = 0
    i = 0
    find = False
    for entry in self.forwarding_table:
        prefixnet = IPv4Network(str(entry[0]) + "/" + str(entry[1]), False)
        if (flag == 0):
            destaddr = IPv4Address(ipv4.src)
        else:
            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
    return pos, find
```

```
def forwardpacket(self, packet, pos):
    ipv4 = packet.get_header(IPv4)
    nexthopip = self.forwarding_table[pos][2]
    if nexthopip == "0.0.0.0":
        nexthopip = ipv4.dst
    if ip_address(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[ip_address(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
```

Task 3: Generating ICMP error messages

主要思路就是在前两个实验许多pass了的判断语句的分支里，把icmp error的情况给加进去，这里遇到的**主要问题**是情况很多，搞得逻辑有点混乱，比如说faq里的那个同时遇到no match 和ttl=0的情况，一开始我想的很混乱，后来自己在纸上写了写各种情况的关系，发现其实并不难。在myrouter.py的代码中，我已经注释了四种error所在的位置，还是比较清楚的。对于每个error也要像icmp reply那样做forward
写了一个专门用来根据ictype和iccode返回icmp error包的函数：

```
def icmperr(self, ifacename, origpkt, ictype, iccode, ttl=64):

    eo = origpkt.get_header(Ethernet)

    io = origpkt.get_header(IPv4)
    ip = copy.deepcopy(io)
    ip.dst = io.src

    e = copy.deepcopy(eo)
    e.src = self.net.interface_by_name(ifacename).ethaddr
    if ip_address(io.src) not in self.arp_table:
        e.dst = eo.src
    else:
        e.dst = self.arp_table[ip_address(io.src)][0]

    i = origpkt.get_header_index(Ethernet)
    del origpkt[i]

    ip.protocol = IPProtocol.ICMP
    ip.src = self.net.interface_by_name(ifacename).ipaddr
    ip.ttl = ttl

    ic = ICMP()
    ic.icmptype = ictype
    ic.icmpcode = iccode
    ic.icmpdata.data = origpkt.to_bytes()[:28]
    sendpkt = e + ip + ic
    return sendpkt
```

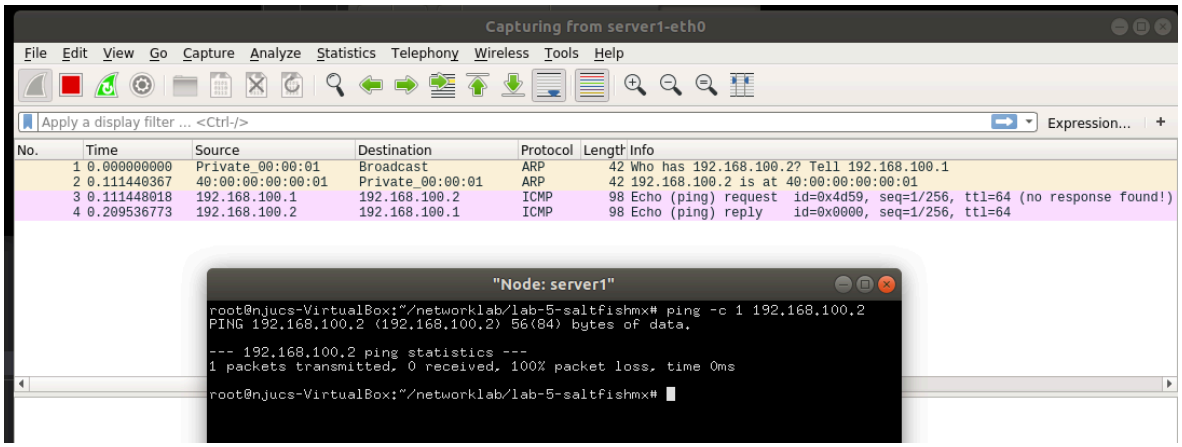
testing：

```
OUTPUT   TERMINAL   DEBUG CONSOLE   PROBLEMS

    192.168.1.1 should arrive on router-eth1.  The router cannot
    handle this type of packet and should generate an ICMP
    destination port unreachable error.
14  The router should send an ICMP destination port unreachable
    error back to 172.16.111.222 out router-eth1.
15  An IP packet from 192.168.1.239 for 10.10.50.250 should
    arrive on router-eth0.  The host 10.10.50.250 is presumed
    not to exist, so any attempts to send ARP requests will
    eventually fail.
16  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
17  Router should try to receive a packet (ARP response), but
    then timeout.
18  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
19  Router should try to receive a packet (ARP response), but
    then timeout.
20  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
21  Router should try to receive a packet (ARP response), but
    then timeout.
22  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
23  Router should try to receive a packet (ARP response), but
    then timeout.
24  Router should send an ARP request for 10.10.50.250 on
    router-eth1.
25  Router should try to receive a packet (ARP response), but
    then timeout.  At this point, the router should give up and
    generate an ICMP host unreachable error.
26  Router should send an ARP request for 192.168.1.239.
27  Router should receive ARP reply for 192.168.1.239.
28  Router should send an ICMP host unreachable error to
    192.168.1.239.


All tests passed!
```

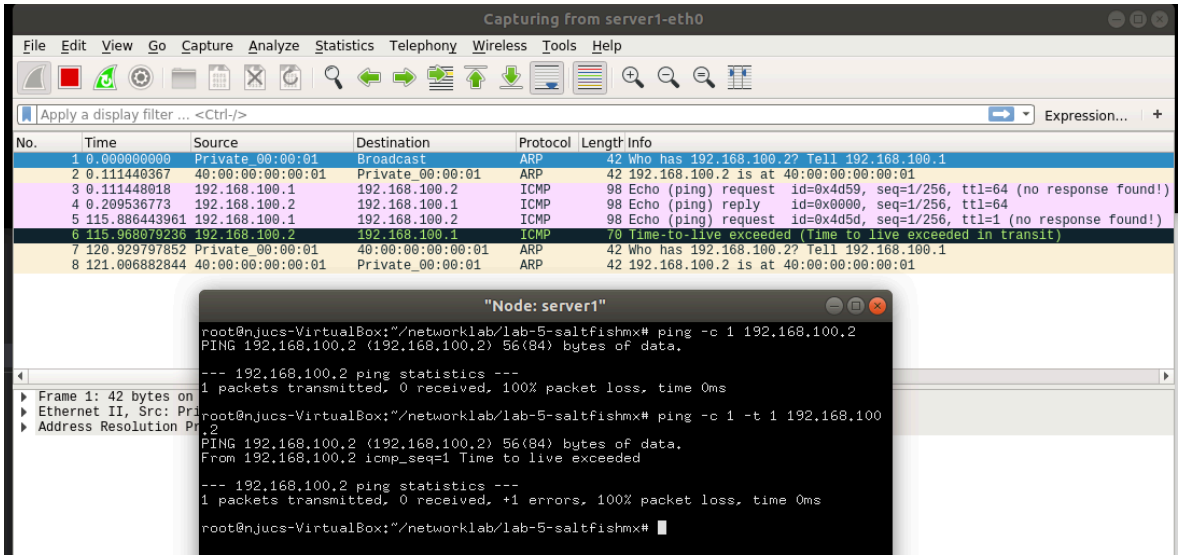
deploying：

选择server1

1. 从server1 ping router的interface 192.168.100.2



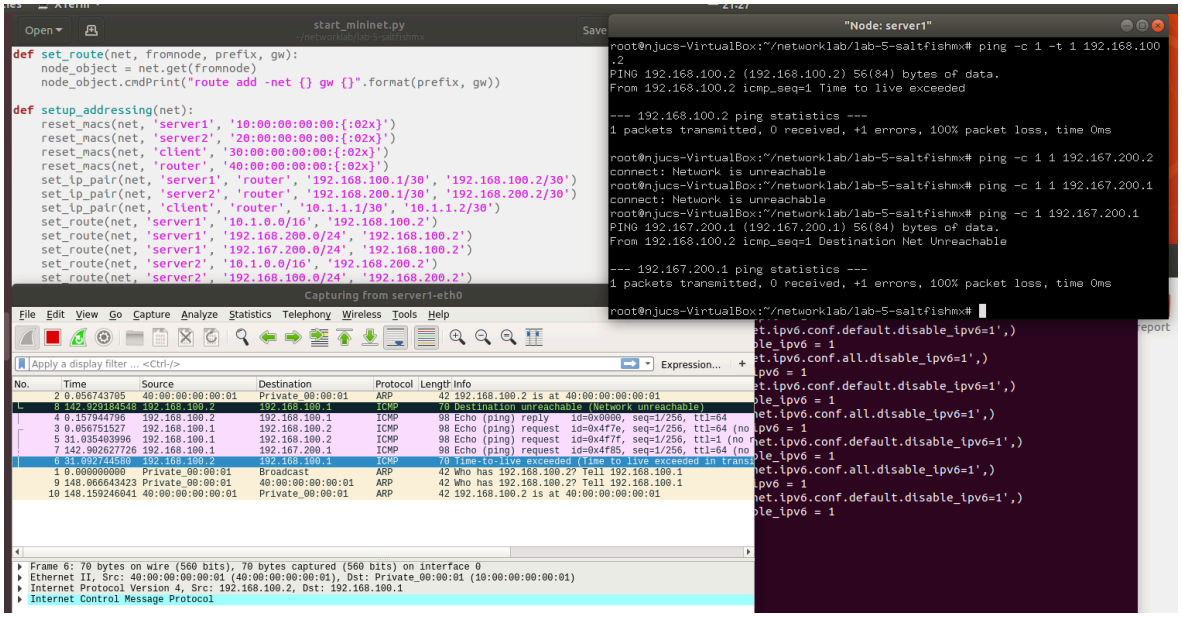
2. 从server1 ping router的interface 192.168.100.2,ttl为1



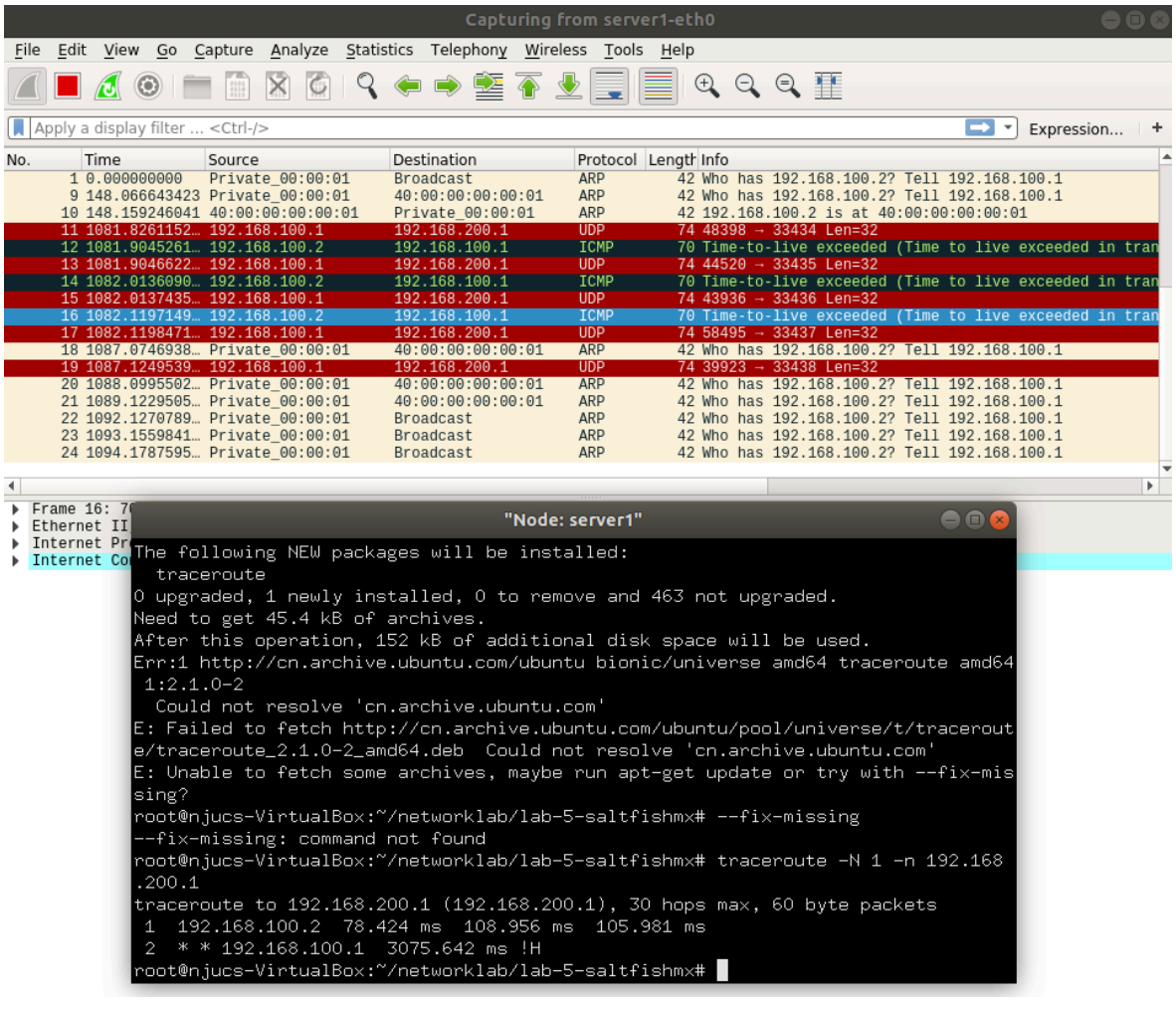
可见router发了icmp time exceeded error包，它的颜色在抓包结果里是黑的

3. send a ping from the server1 to an address that doesn't have a match in the router's forwarding table.

这里发现start_mininet.py里server1可以走的所有路在forwarding table里都包含了，所以我自己给他加了一个route，以完成destination unreachable



4. do a "traceroute" across the toy network in Mininet.



可见traceroute中从server1到192.168.200.1经过了两跳，与实际情况相符，实验成功

实验总结

感觉这次实验还是挺难的，比lab4还难，做了大概十个小时，基本上是面向测试用例编程，在过程中感觉自己代码写得太丑了，上次lab4构造的命名古怪的结构要不是这次读上次报告自己都不知道是干嘛的，给自己增加了很多麻烦。中间很多的时间在把上次的功能重构成函数让逻辑清晰点，不然自己真是debug不下去了，幸好下次lab好像不用继承这次的代码了，再也不想看到自己写的这团东西了（笑）

完