# 计算机网络实验 Lab5实验报告


## 工程管理学院 计算机金融实验班 211275032 汪科

## Email：211275032@smail.nju.edu.cn




# 一、实验名称：Respond to ICMP

# 二、实验目的

响应 ICMP 消息，如回显请求（“ping”）。

必要时生成 ICMP 错误消息，例如当 IP数据包的 TTL（生存时间）值已递减为零。

# 三、实验内容

沿用lab4的代码，本实验采用了**多线程实现。**

## PartⅠ.响应ICMP回显请求

在将数据包分类，在ICMP包中，判断：

```python
if dst_ip in self.ip_list:
    if icmp_header.icmptype == ICMPType.EchoRequest:
        self.ICMPTORouter(src_ip, dst_ip, pkt_ttl,src_mac, dst_mac, icmp_header,packet)
```

将数据包的信息传入函数ICMPTORouter中，其实现如下：

```python
def ICMPTORouter(self,src_ip,dst_ip,pkt_ttl,src_mac,dst_mac,icmp_header,packet):# Responding to ICMP echo requests
    i=packet.get_header_index(Ethernet)
    del packet[i]

    log_info(f'收到一个ICMP echo request包，回复一个ICMP echo reply包')
    snd_icmp_header = ICMP()
    snd_icmp_header.icmptype = ICMPType.EchoReply
    log_info(f'snd_icmp_header_icmptype:{snd_icmp_header.icmptype}')

    snd_icmp_header.icmpdata.data = icmp_header.icmpdata.data
    snd_icmp_header.icmpdata.identifier = icmp_header.icmpdata.identifier
    snd_icmp_header.icmpdata.sequence = icmp_header.icmpdata.sequence

    snd_ip_header=IPv4()
    snd_ip_header.src=dst_ip
    snd_ip_header.dst=src_ip
    snd_ip_header.protocol=IPProtocol.ICMP
    snd_ip_header.ttl=64
    snd_eth_header=Ethernet()
    snd_eth_header.src=dst_mac
    snd_eth_header.dst=src_mac
    snd_eth_header.ethertype=EtherType.IPv4
    snd_packet=snd_eth_header+snd_ip_header+snd_icmp_header


    forward_info = self.forward_query(src_ip)

    if forward_info == None:
        log_info("转发表没有匹配，丢弃")

    else:
        next_hop_ip = forward_info[2]

        if next_hop_ip != '0.0.0.0':
            pass
        else:
            next_hop_ip = src_ip
        if self.arp_table.get(next_hop_ip) == None:
            log_info(f'ARP表中没有匹配，发送地址为{next_hop_ip}的arp请求')
            if next_hop_ip in self.ArpWaitingList.keys():
                log_info(f'{next_hop_ip}已经在等待队列中了，不用再发送了')
            else:
                log_info(f'{next_hop_ip}不在等待队列中，加入队列')
                self.lock.acquire()
                self.ArpWaitingList[next_hop_ip] = [
                    time.time()-10, 0]
                self.lock.release()
            tmp_packet = WaitingPacket(
                dst_ip, src_ip, dst_mac, next_hop_ip, forward_info[3], snd_ip_header.ttl, snd_packet)
            log_info(f'待转发数据包：{tmp_packet}')
            self.ArpRequestQueue.put(tmp_packet)  # 待转发数据包放入等待队列

        else:
            log_info('ARP表中有匹配，直接转发')
            next_hop_mac = self.arp_table[next_hop_ip][0]
            snd_packet[Ethernet].src = self.mac_list[self.port_list.index(
                forward_info[3])]
            snd_packet[Ethernet].dst = next_hop_mac
            snd_packet[IPv4].ttl -= 1
            log_info(f'转发了一个包：{snd_packet}')
            self.net.send_packet(forward_info[3], snd_packet)
```

这里前半部分是构造回显请求数据包：

- 请求中的数据字段复制
- 请求中的标识符复制
- 请求中的序列号复制
- ICMP包头类型设置
- 目标IP设置为原包源地址，源IP设置为路由器接口地址

后半部分与lab4基本一致，判断这个数据包下一跳IP是否存在于ARP表中，如果不存在的话就放入等待队列并发送ARP请求，与普通转发数据包无异。这一部分在后面也会略去。

## PartⅡ.ICMP错误消息——Unsupported function

这部分判断所处的位置在开头：

```
if dst_ip in self.ip_list:
    if icmp_header.icmptype == ICMPType.EchoRequest:
        self.ICMPTORouter(src_ip, dst_ip, pkt_ttl,src_mac, dst_mac, icmp_header,packet)

    else:
        #TODO: 不支持的功能
        if icmp_header.icmptype in ErrorMessageList:
            log_info(f'Unsupported function，但是路由器不应主动发送错误消息的回复')
            return
        self.DstPortUnreachable(src_ip, dst_ip, pkt_ttl,src_mac, dst_mac, icmp_header,packet)
    return
```

由DstPortUnreachable函数控制完成。

由于路由器不应主动回复任何ICMP错误消息，所以构造了ErrorMessageList来判断发来的包是否为ICMP错误消息：

```
ErrorMessageList=[
    ICMPType.DestinationUnreachable,
    ICMPType.SourceQuench,
    ICMPType.Redirect,
    ICMPType.TimeExceeded,
    ICMPType.ParameterProblem
]
```

如果是，则不构造任何回复包。

如果不是，调用self.DstPortUnreachable函数，并传入包的信息。这个函数的实现如下：

```python
def DstPortUnreachable(self,src_ip,dst_ip,pkt_ttl,src_mac,dst_mac,icmp_header,packet):#Unsupported function
    i=packet.get_header_index(Ethernet)
    del packet[i]

    log_info(f'ICMP Destination Unreachable:Unsupported function,回复错误')
    snd_icmp_header = ICMP()
    snd_icmp_header.icmptype = ICMPType.DestinationUnreachable
    snd_icmp_header.icmpcode = ICMPTypeCodeMap[ICMPType.DestinationUnreachable].PortUnreachable
    snd_icmp_header.icmpdata.data=packet.to_bytes()[:28]
    snd_icmp_header.icmpdata.nexthopmtu=1500

    snd_ip_header=IPv4()
    snd_ip_header.src=dst_ip
    snd_ip_header.dst=src_ip
    snd_ip_header.protocol=IPProtocol.ICMP
    snd_ip_header.ttl=64
    snd_eth_header=Ethernet()
    snd_eth_header.src=dst_mac
    snd_eth_header.dst=src_mac
    snd_eth_header.ethertype=EtherType.IPv4
    snd_packet=snd_eth_header+snd_ip_header+snd_icmp_header
    forward_info = self.forward_query(src_ip)

    if forward_info == None:
        log_info("转发表没有匹配，丢弃")

    else:
        next_hop_ip = forward_info[2]

        if next_hop_ip != '0.0.0.0':
            pass
        else:
            next_hop_ip = src_ip
        if self.arp_table.get(next_hop_ip) == None:
            log_info(f'ARP表中没有匹配，发送地址为{next_hop_ip}的arp请求')
            if next_hop_ip in self.ArpWaitingList.keys():
                log_info(f'{next_hop_ip}已经在等待队列中了，不用再发送了')
            else:
                log_info(f'{next_hop_ip}不在等待队列中，加入队列')
                self.lock.acquire()
                self.ArpWaitingList[next_hop_ip] = [
                    time.time()-10, 0]
                self.lock.release()
            tmp_packet = WaitingPacket(
                self.ip_list[self.port_list.index(forward_info[3])], src_ip, dst_mac, next_hop_ip, forward_info[3], snd_ip_header.ttl, snd_packet)
            self.ArpRequestQueue.put(tmp_packet)  # 待转发数据包放入等待队列

        else:
            log_info('ARP表中有匹配，直接转发')
            next_hop_mac = self.arp_table[next_hop_ip][0]
            snd_packet[Ethernet].src = self.mac_list[self.port_list.index(
                forward_info[3])]
            snd_packet[Ethernet].dst = next_hop_mac
            snd_packet[IPv4].ttl -= 1
            snd_packet[IPv4].src=self.ip_list[self.port_list.index(forward_info[3])]
            log_info(f'转发了一个包：{snd_packet}')
            self.net.send_packet(forward_info[3], snd_packet)
```

作为第一个构造ICMP错误回复的函数，这里没有略去下面的实现，因为有一点不一样：**这个回复包的源IP地址不再是接收到原数据包对应的端口的IP地址，而是将要发送这个回复包的端口的IP地址。**

重要部分在于上面对ICMP包头各类字段的处理。

## PartⅢ.ICMP错误消息——No matching entries

在排除这个ICMP包是发送给自身的情况之后，在转发表里查找目的IP。作以下判断来确定是否要构造目的地不可达的ICMP错误回复：

```
forward_info = self.forward_query(dst_ip)
if forward_info == None:
    #TODO:错误：转发表没有匹配
    if icmp_header.icmptype in ErrorMessageList:
        log_info(f'No matching entries，但是路由器不应主动发送错误消息的回复')
        return
    self.DstNetUnreachable(src_ip, dst_ip, pkt_ttl,src_mac, dst_mac, icmp_header,packet)
```

DstNetUnreachable函数实现如下：

```
def DstNetUnreachable(self,src_ip,dst_ip,pkt_ttl,src_mac,dst_mac,icmp_header,packet):#No matching entries
    i=packet.get_header_index(Ethernet)
    del packet[i]

    log_info(f'ICMP Destination Net Unreachable:No matching entries,回复错误')
    snd_icmp_header = ICMP()
    snd_icmp_header.icmptype = ICMPType.DestinationUnreachable
    snd_icmp_header.icmpcode = ICMPTypeCodeMap[ICMPType.DestinationUnreachable].NetworkUnreachable
    snd_icmp_header.icmpdata.data=packet.to_bytes()[:28]


    snd_ip_header=IPv4()
    snd_ip_header.src=dst_ip
    snd_ip_header.dst=src_ip
    snd_ip_header.protocol=IPProtocol.ICMP
    snd_ip_header.ttl=64
    snd_eth_header=Ethernet()
    snd_eth_header.src=dst_mac
    snd_eth_header.dst=src_mac
    snd_eth_header.ethertype=EtherType.IPv4
    snd_packet=snd_eth_header+snd_ip_header+snd_icmp_header
```

处理方式基本差不多。

## PartIV.ICMP错误消息——TTL expired

在转发表有查找结果之后，判断其TTL是否合法：

```
else:
    #TODO：检测TTL
    if pkt_ttl <= 1:
        if icmp_header.icmptype in ErrorMessageList:
            log_info(f'TTL expired，但是路由器不应主动发送错误消息的回复')
            return
        self.TimeExceeded(src_ip, dst_ip, pkt_ttl,src_mac, dst_mac, icmp_header,packet)
        return
```

如果TTL小于等于1，执行函数TimeExceeded：

```python
def TimeExceeded(self,src_ip,dst_ip,pkt_ttl,src_mac,dst_mac,icmp_header,packet):#TTL expired
    i=packet.get_header_index(Ethernet)
    del packet[i]

    log_info(f'ICMP Time Exceeded,回复错误')
    snd_icmp_header = ICMP()
    snd_icmp_header.icmptype = ICMPType.TimeExceeded
    snd_icmp_header.icmpdata.data=packet.to_bytes()[:28]

    snd_ip_header=IPv4()
    snd_ip_header.src=dst_ip
    snd_ip_header.dst=src_ip
    snd_ip_header.protocol=IPProtocol.ICMP
    snd_ip_header.ttl=64
    snd_eth_header=Ethernet()
    snd_eth_header.src=dst_mac
    snd_eth_header.dst=src_mac
    snd_eth_header.ethertype=EtherType.IPv4
    snd_packet=snd_eth_header+snd_ip_header+snd_icmp_header
    forward_info = self.forward_query(src_ip)
    log_info(f'forward_info:{forward_info}')
```

## Part Ⅴ.ICMP错误消息——ARP failure

由于是多线程，这部分在副线程中完成。

同lab4，副线程的主要函数是arp_handler，它会不停地调用一个函数arp_forward_handler来管理ARP请求的再发送与过期数据包的清理，其实现如下：

```python
def arp_forward_handler(self):
    # 遍历ArpWaitingList，对过期条目进行清算
    AddrToBeDeleted = []
    UnreachablePacketList = []
    if self.ArpWaitingList == {}:
        return
    # 获取锁，防止另一线程在遍历过程中修改字典
    self.lock.acquire()
    for key, value in self.ArpWaitingList.items():
        if time.time()-value[0] >= 1.0:
            if value[1] >= 5:
                #log_info(f'请求IP为{key}的arp超时，发送ICMP错误信息，时间为{time.time()}')
                #TODO：ARP故障
                for i in range(self.ArpRequestQueue.qsize()):
                    tmp_packet = self.ArpRequestQueue.get(block=False)
                    if tmp_packet.next_hop_ip == key:
                        #这里融合了ICMP/UDP/TCP的情况
                        if tmp_packet.my_packet[IPv4].protocol!=IPProtocol.ICMP:
                            pass
                        elif tmp_packet.my_packet[ICMP].icmptype in ErrorMessageList:
                            log_info(f'ARP faliure，但是路由器不应主动发送错误消息的回复')
                            continue
                        else:
                            pass
                        UnreachablePacketList.append(tmp_packet)
                        #不放回=丢弃数据包
                        pass
                    else:
                        self.ArpRequestQueue.put(tmp_packet)
                AddrToBeDeleted.append(key)
                continue
            else:
                #log_info(f'请求IP为{key}的arp超时，重新发送arp请求')
                self.ArpWaitingList[key][0] = time.time()
                self.ArpWaitingList[key][1] += 1

                if key==IPv4Address('192.168.1.233') and self.ArpWaitingList[key][1]==5:
                    self.ArpWaitingList[key][0]-=2
                    log_info(f'这是一个特判，测试样例有点问题,时间为{time.time()}')

                forward_info = self.forward_query(key)
                #再次构造ARP请求包
                arp_request_packet = Ethernet(src=self.mac_list[self.port_list.index(forward_info[3])],\
                                              dst='ff:ff:ff:ff:ff:ff', ethertype=EtherType.ARP)+\
                    Arp(
                    operation=ArpOperation.Request, senderhwaddr=self.mac_list[self.port_list.index(forward_info[3])],\
                          senderprotoaddr=self.ip_list[self.port_list.index(forward_info[3])], \
                            targethwaddr='ff:ff:ff:ff:ff:ff', targetprotoaddr=key)
                #发包
                log_info(f'要发了，目的地址为{key}，端口为{forward_info[3]},次数为{value[1]}内容为{arp_request_packet}')
                self.net.send_packet(forward_info[3], arp_request_packet)
                log_info(f'刚才发了一个arp请求，目的地址为{key}，端口为{forward_info[3]},次数为{value[1]}内容为{arp_request_packet}')

        else:
            pass
```

在lab4中，我们把超过五次还未收到ARP回复的IP地址对应的数据包都丢掉，但是这里需要针对每个数据包进行回复。所以在过程中，将过期IP对应的数据包记录下来（同时移出等待队列），在结尾统一对每个要移除的数据包发送ICMP错误回复：

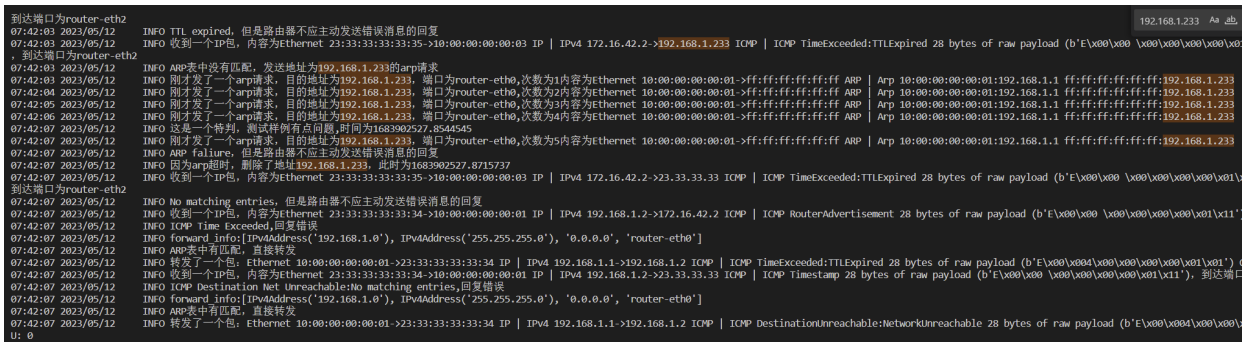
```python
# 针对每个要删除的数据包发送ICMP错误回复
for packet in UnreachablePacketList:
    self.DstHostUnreachable(packet.src_ip, packet.dst_ip,packet.pkt_ttl,packet.my_packet)
```

DstHostUnreachable的实现：

```python
def DstHostUnreachable(self,src_ip,dst_ip,pkt_ttl,packet):#ARP Failure
    i=packet.get_header_index(Ethernet)
    del packet[i]

    log_info(f'ICMP Destination Unreachable:ARP Failure,回复错误')
    snd_icmp_header = ICMP()
    snd_icmp_header.icmptype = ICMPType.DestinationUnreachable
    snd_icmp_header.icmpcode = ICMPTypeCodeMap[ICMPType.DestinationUnreachable].HostUnreachable
    snd_icmp_header.icmpdata.data=packet.to_bytes()[:28]
    snd_icmp_header.icmpdata.nexthopmtu=1500

    snd_ip_header=IPv4()
    snd_ip_header.src=dst_ip
    snd_ip_header.dst=src_ip
    snd_ip_header.protocol=IPProtocol.ICMP
    snd_ip_header.ttl=64
    snd_eth_header=Ethernet()
    snd_eth_header.ethertype=EtherType.IPv4
    snd_packet=snd_eth_header+snd_ip_header+snd_icmp_header
```

## PartVI.测试用例问题

在用例46-50处，路由器需要发送ARP请求，请求IP地址未192.168.1.233的mac地址，并且收不到回复，处理目的地址为它的数据包。



```
38  An arp request message should out on eth0
39  An arp response message should arrive on eth0
40  The packet should be forwarded out port eth0, w
    appropriately decremented TTL.
41  An ICMP error message should arrive on eth2
42  The router should not do anything
43  An ICMP error message should arrive on eth2
44  The router should not do anything
45  An ICMP error message should arrive on eth2
46  An arp request message should out on eth0
47  An arp request message should out on eth0
48  An arp request message should out on eth0
49  An arp request message should out on eth0
50  An arp request message should out on eth0
51  The router should not do anything
52  An ICMP error message should arrive on eth2
53  The router should not do anything
54  An ICMP message should arrive on eth0
55  An icmp error message should out on eth0
56  An ICMP message should arrive on eth0
57  An icmp error message should out on eth0
58  An ICMP message should arrive on eth0
59  An icmp error message should out on eth0
60  An UDP message should arrive on eth0
61  An icmp error message should out on eth0
07:42:12 2023/05/12  WARNING Tried to find non-exis
```

然而，在发送第五次ARP请求的一秒之内，发生了很多事，其中包括收到了另一个目的地址为192.168.1.233的ICMP包：

```
07:42:07 2023/05/12    INFO ICMP Destination Net Unreachable:No matching entries,回复错误
07:42:07 2023/05/12    INFO forward info:None
07:42:07 2023/05/12    INFO 转发表没有匹配，丢弃
07:42:07 2023/05/12    INFO 收到一个IP包，内容为Ethernet 23:33:33:33:33:37->10:00:00:00:00:02 IP | IPv4 23.33.33.33->192.168.1.233 ICMP | ICMP EchoRequest 0 42 (5 data bytes)，到达端口为router-eth1
07:42:07 2023/05/12    INFO ARP表中没有匹配，发送地址为192.168.1.233的arp请求
07:42:08 2023/05/12    INFO 刚才发了一个arp请求，目的地址为192.168.1.233，端口为router-eth0,次数为1内容为Ethernet 10:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.1.1 ff:ff:ff:ff:ff:ff:192.168.1.233
07:42:09 2023/05/12    INFO 刚才发了一个arp请求，目的地址为192.168.1.233，端口为router-eth0,次数为2内容为Ethernet 10:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.1.1 ff:ff:ff:ff:ff:ff:192.168.1.233
07:42:10 2023/05/12    INFO 刚才发了一个arp请求，目的地址为192.168.1.233，端口为router-eth0,次数为3内容为Ethernet 10:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.1.1 ff:ff:ff:ff:ff:ff:192.168.1.233
07:42:11 2023/05/12    INFO 刚才发了一个arp请求，目的地址为192.168.1.233，端口为router-eth0,次数为4内容为Ethernet 10:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.1.1 ff:ff:ff:ff:ff:ff:192.168.1.233
07:42:12 2023/05/12    INFO 这是一个特判，测试样例有点问题,时间为1683902532.0339835
07:42:12 2023/05/12    INFO 刚才发了一个arp请求，目的地址为192.168.1.233，端口为router-eth0,次数为5内容为Ethernet 10:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.1.1 ff:ff:ff:ff:ff:ff:192.168.1.233
07:42:12 2023/05/12    INFO 因为arp超时，删除了地址192.168.1.233，此时为1683902532.0360048
07:42:12 2023/05/12    INFO ICMP Destination Unreachable:ARP Failure,回复错误
07:42:12 2023/05/12    INFO 转发表没有匹配，丢弃
07:42:12 2023/05/12    INFO 收到一个IP包，内容为Ethernet 23:33:33:33:33:34->10:00:00:00:00:01 IP | IPv4 192.168.1.2->10.10.0.1 ICMP | ICMP EchoRequest 0 42 (5 data bytes)，到达端口为router-eth0
```

注意时间戳：从第五次ARP到下一个目的地址为192.168.1.233的ICMP包，都在07:42:07内发生，**但是测试用例实际上希望后面那个ICMP包是在原来数据包被清理之后收到，并重新发起一轮ARP请求的**，因此此处只能作特判，**在第五次ARP之后直接令其过期。**

```python
if key==IPv4Address('192.168.1.233') and self.ArpWaitingList[key][1]==5:
    self.ArpWaitingList[key][0]-=2
    log_info(f'这是一个特判，测试样例有点问题,时间为{time.time()}')
```

## 为什么路由器不应回复ICMP错误消息?

如果路由器对错误消息进行回复，很容易造成这样的情况：当一个路由器收到错误消息并回复，在这个包回去的路上，另一个路由器又对这个包进行回复，就会造成两个路由器之间来回地发包，无意义地抢占网络资源，造成网络风暴（虽然不是广播，但是这么做的路由器多了，就会造成资源抢占）。

所以路由器不应回复ICMP错误消息。

# 四、实验结果

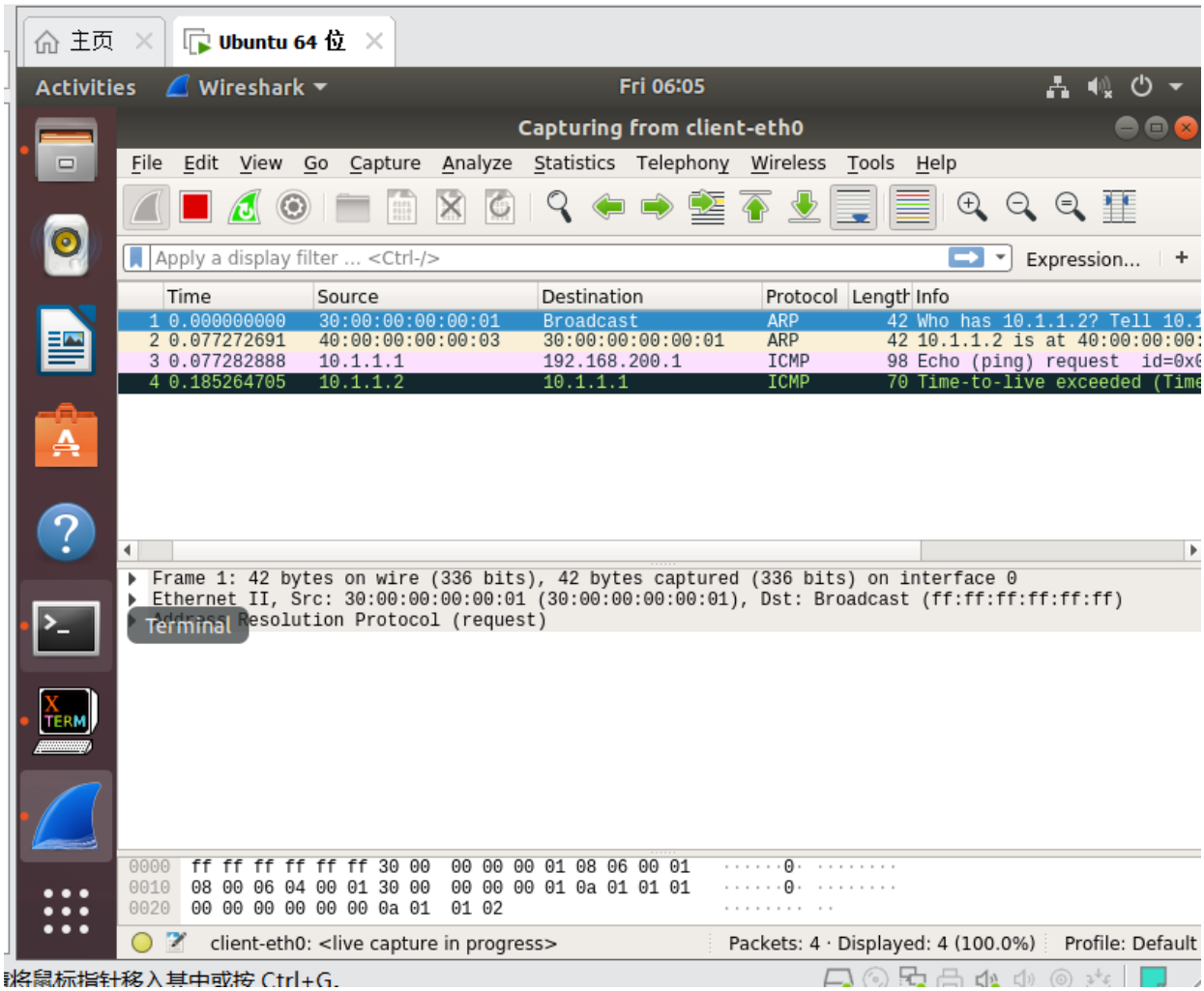
## 测试用例

```
66  An ICMP message should arrive on eth1
67  The router should not do anything
68  An ICMP message should arrive on eth1
69  An arp request message should out on eth0
70  An arp request message should out on eth0
71  An arp request message should out on eth0
72  An arp request message should out on eth0
73  An arp request message should out on eth0
74  The router should not do anything
75  An ICMP message should arrive on eth0
76  An icmp message should out on eth0
07:42:12 2023/05/12  WARNING Tried to find non-existent header f

77  An TCP message should arrive on eth2
78  An icmp error message should out on eth0
79  An UDP message should arrive on eth2
80  An icmp error message should out on eth0


All tests passed!
```

## Wireshark抓包

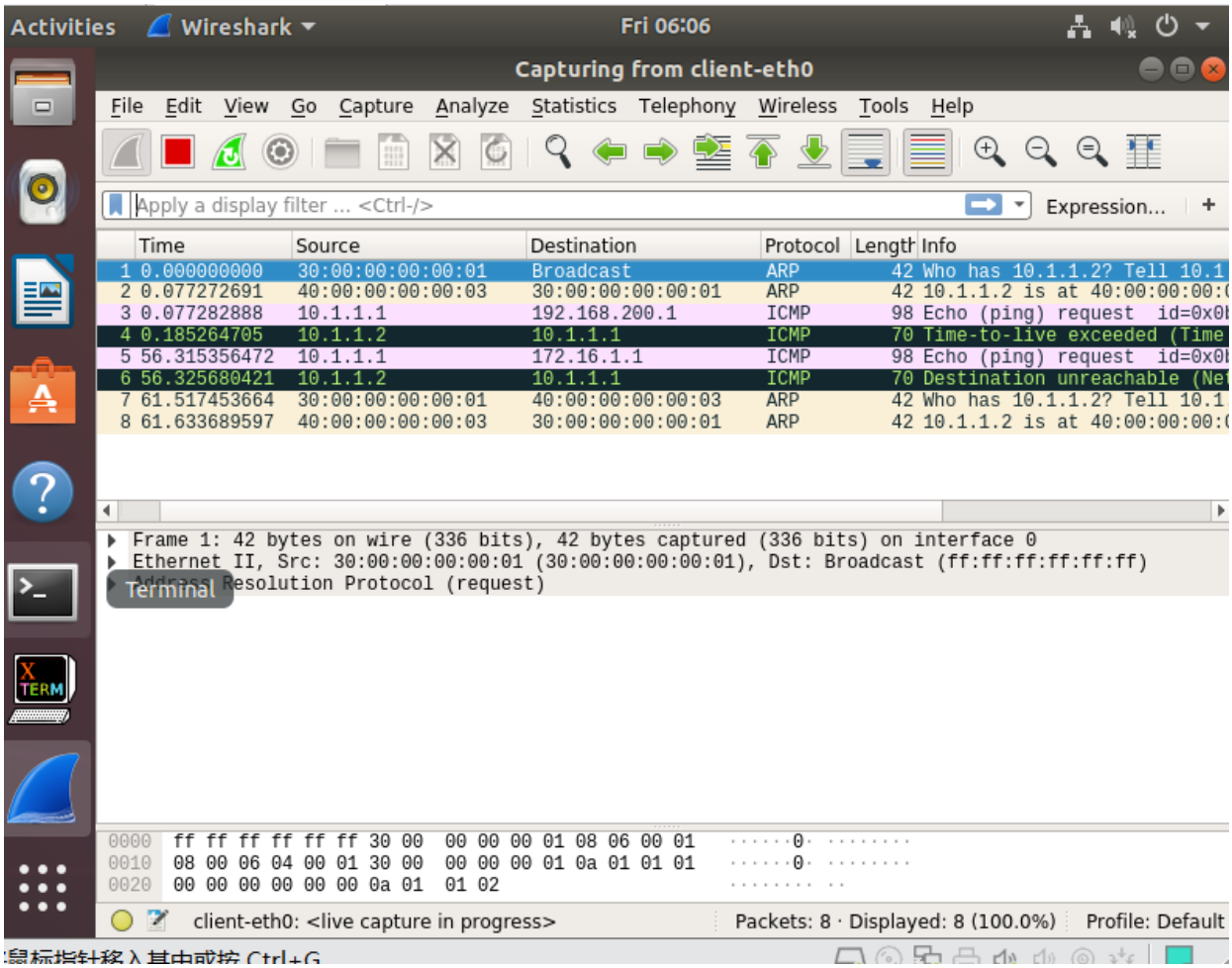
命令：client ping -c 1 -t 1 192.168.200.1

结果：（ttl expired）



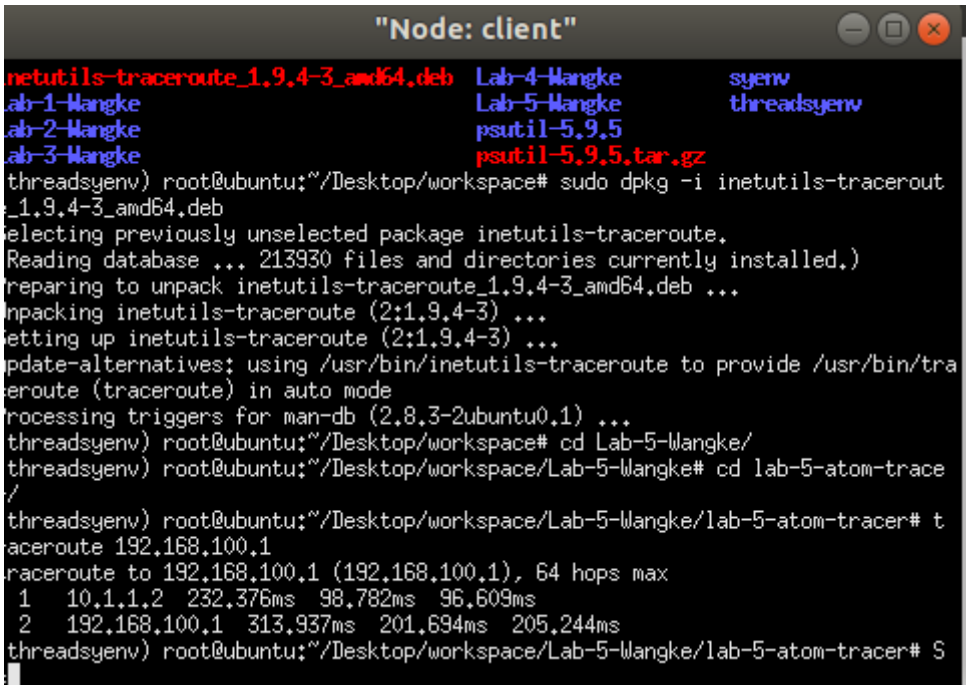

命令：client ping -c 1 172.16.1.1

结果：（destination unreachable）



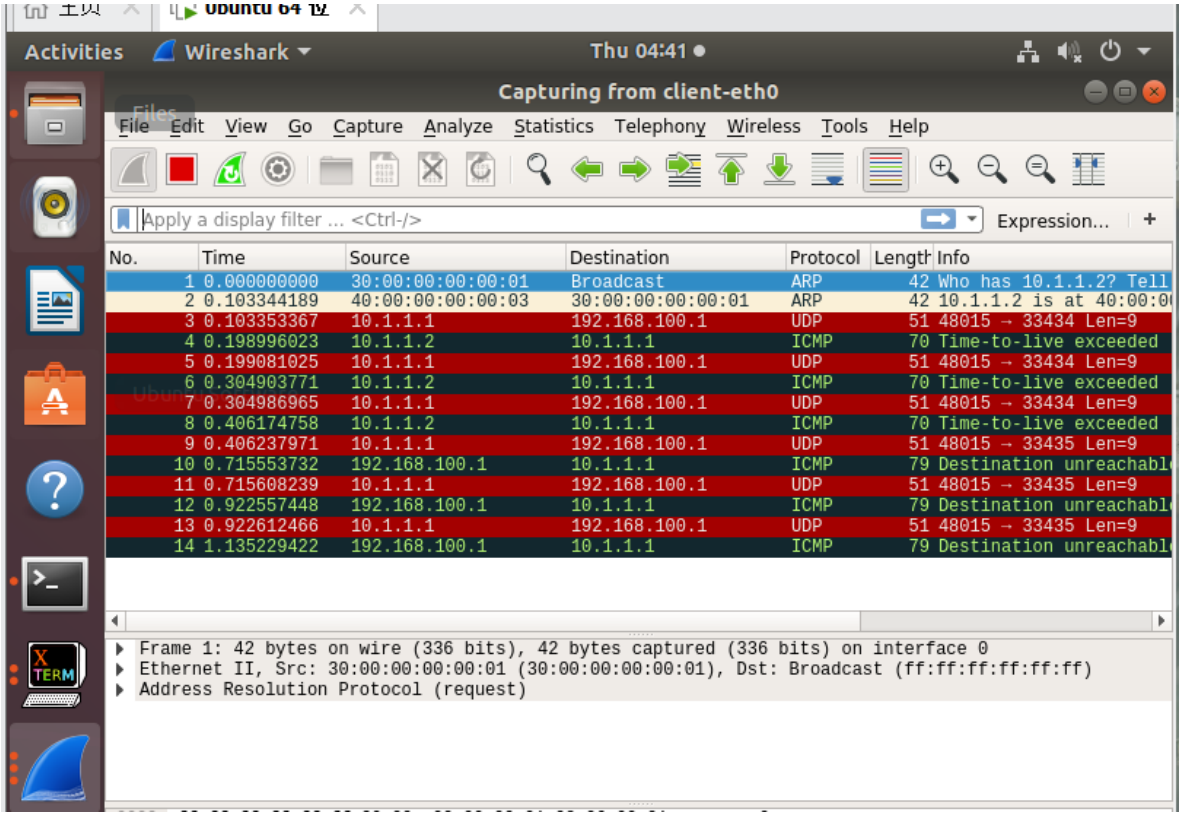

命令：client# traceroute 192.168.100.1

结果：




wireshark抓包：

- client-eth0：




从client-eth0来看，client先请求了路由器上与其相连的端口的mac，随后连发三个ttl为1的UDP包，目的是探测第一跳地址及时延。然后连发三个ttl为2个UDP包，目的是探测第二跳地址及时延。

当收到的ICMP不再是TTL expired，说明已经TTL已经达到连通目的地的最小值，此时client不再发包。

# 五、总结与感想

- ICMP回复错误信息的情况还是不少的，需要比较细致的分析
- 其实代码有相当大的优化空间，比如四种ICMP的包的构造有80%相同的部分（但是我已经优化了一部分了，剩下的优化不动了）