# 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 | 2021.5.8-2021.5.19 |

## 1. 实验名称

Lab 5: Respond to ICMP

## 2. 实验目的

1. Respond to ICMP messages like echo requests ("pings").
2. Generate ICMP error messages

## 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`

`ftItem` is exactly the same as what I used in lab4, while there is a slight change in `queueItem`. To generate ICMP error messages, the Ifacename of the packet needs to be recorded. Therefore, add an `icmp_info` attribute to `queueItem` ,indicating the coming port of the packet.

```
class queueItem():
    def __init__(self,pkt,match,icmp):
        self.pkt=pkt
        self.rounds=0
        self.time=0
        self.match=match
        self.icmp_info=icmp # icmp_info records the coming port of the packet
```

## Step 2: Responding to ICMP echo requests

Define a function to respond to ICMP echo requests. The function takes in the original request packet and the IP address of the destination, and returns a reply packet.

The details of building each header can be found in the switchyard API references.

```
def ping_reply(packet,dstip):
    ether=Ethernet()
    ether.ethertype=EtherType.IP
    # construct ether header
    ip=IPv4()
    ip.src=IPAddr(dstip)# the reply ip src is ping dst
    ip.dst=IPAddr(packet[IPv4].src)# the reply ip dst is ping src
    ip.protocol=IPProtocol.ICMP
    ip.ttl=64
    ip.ipid=0
    #construct ip header
    icmp=ICMP()
    icmp.icmptype=ICMPType.EchoReply
    icmp.icmpcode=ICMPCodeEchoReply.EchoReply
    icmp.icmpdata.sequence=packet[ICMP].icmpdata.sequence
    icmp.icmpdata.identifier=packet[ICMP].icmpdata.identifier
    icmp.icmpdata.data=packet[ICMP].icmpdata.data
    #construct icmp header
    return ether+ip+icmp
```

Then, we need to call the function correctly when the router receives packets. There is only one case when the router needs to generate an ICMP echo reply, which is that the router receives a IPv4 packet heading to one of its interfaces and the packet happens to be an ICMP echo request. In other cases, the function above will not be called.

```
if ipv4:
  head=packet[IPv4]
  for i in self.ip_list:
    if packet[IPv4].dst==i:# IPv4 packet heading to the router interfaces
      if packet.has_header(ICMP) and
packet[ICMP].icmptype==ICMPType.EchoRequest:
        # the packet happens to be a ICMP echo request
        packet=ping_reply(packet,i)
        head=packet[IPv4]
        break
```

Above are the logic of responding the ICMP echo requests.

## Step 3: Generating ICMP error messages

Just like Step3, define a function to generate ICMP error messages.  The function needs the IP source and IP destination of the message, icmptype, icmpcode ,ttl and the packet that causes an ICMP error messages.

```
def construct_icmperror(ipsrc,ipdst,xtype,xcode=0,ttl=64,origpkt=None):
    icmp = ICMP()
    icmp.icmptype = xtype
    icmp.icmpcode=xcode
    if not origpkt is None:
        i = origpkt.get_header_index(Ethernet)
        del origpkt[i]
        icmp.icmpdata.data = origpkt.to_bytes()[:28]
        icmp.icmpdata.origdgramlen = len(origpkt)
    #set icmptype, icmpcode and fill icmpdata with the original packet
    ip = IPv4()
    ip.protocol = IPProtocol.ICMP
    ip.src=IPAddr(ipsrc)
    ip.dst=IPAddr(ipdst)
    ip.ttl=ttl
    # set IPv4 header
    ether=Ethernet()
    #set Ethernet header
    pkt = ether+ip + icmp
    #return the error message
    return pkt
```

According to the lab manual, there are 4 cases when we should generate ICMP error messages.

- An incoming packet is destined to an IP addresses assigned to one of the router's interfaces, but the packet is not an ICMP echo request

  This case is the `else` side of sending an ICMP echo reply.

```
if packet[IPv4].dst==i:
  if packet.has_header(ICMP) and packet[ICMP].icmptype==ICMPType.EchoRequest:
  '''
  ping reply
  '''
else:
  # receive a packet heading to the router itself, but not a ping request
  for i in self.interfaces:
    if i.name==ifaceName:
      p=i
      break
      #p is the dst interface of the router
      #ip address of interface p is the src, ip address of the packet is the dst
      # icmptype estination port unreachable

packet=construct_icmperror(p.ipaddr,head.src,ICMPType.DestinationUnreachable,3,64,packet)
```

- When attempting to match the destination address of an IP packet with entries in the forwarding table, no matching entries are found

```
# cannot match in the forwarding table
if pos ==-1:
    print("cannot match?")
    for i in self.interfaces:
        if i.name==ifaceName:
            p2=i
            break
packet=construct_icmperror(p2.ipaddr,head.src,ICMPType.DestinationUnreachable,0,64,packet)
    # generate a error message
    #look up the forwarding table for the error message packet
    head=packet[IPv4]
    pos=-1
    maxprifixlen=-1
    index=0
    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
    #find the match in the table and add it to the packet queue
    self.q.append(queueItem(packet,self.forward_table[pos],ifaceName))
```

- After decrementing an IP packet's TTL value as part of the forwarding process, the TTL becomes zero.

```
if head.ttl<=0:
  for i in self.interfaces:
    if i.name==ifaceName:
      p1=i
      break# find the coming port
  packet=construct_icmperror(p1.ipaddr,head.src,ICMPType.TimeExceeded,0,64,packet)
# then do maxprfix match for the error message packet
```

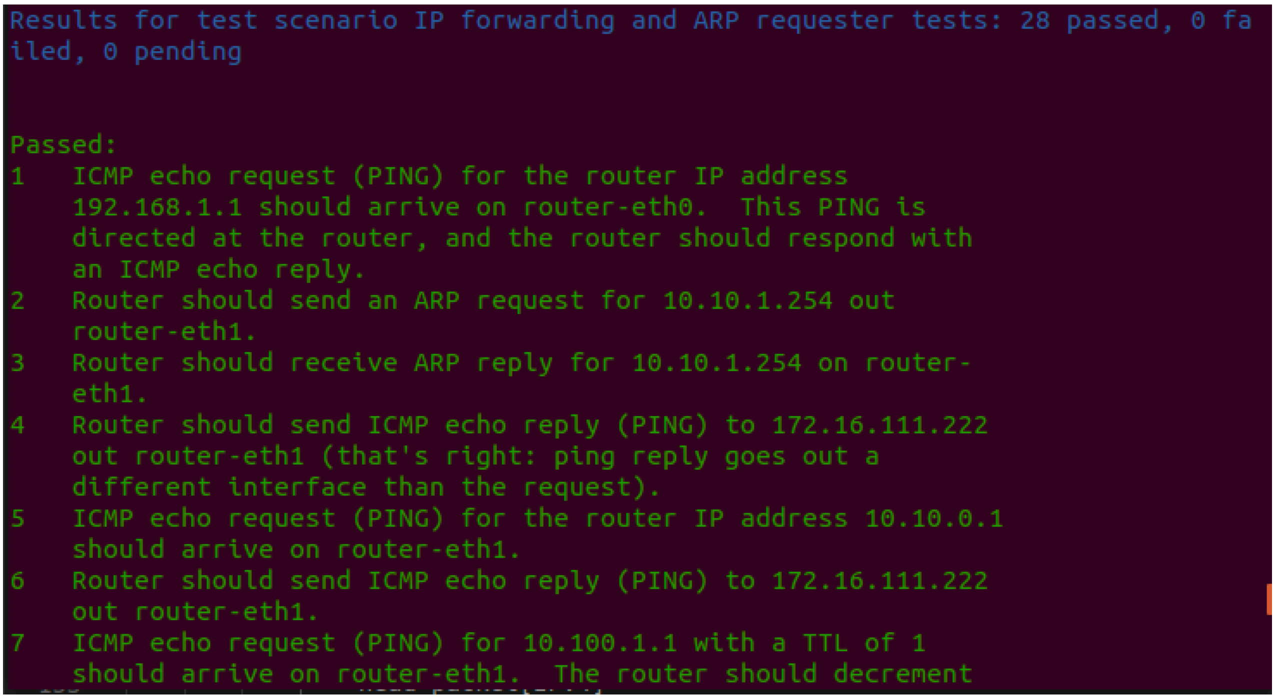
- ARP Failure.

```
if self.q[0].rounds>=5:
  for i in self.interfaces:
    if i.name == self.q[0].icmp_info:
    p4=i
    break
packet=construct_icmperror(p4.ipaddr,self.q[0].pkt[IPv4].src,ICMPType.DestinationUnreachable,1,64,self.q[0].pkt)
# arp failure icmp error message
  head=packet[IPv4]
'''
maxprix match for the error message
'''
  del(self.q[0])
  newq=queueItem(packet,self.forward_table[pos],p4.name)
  self.q.append(newq)
```

Above are the logic of Generating ICMP error messages.

## Step 4: Test

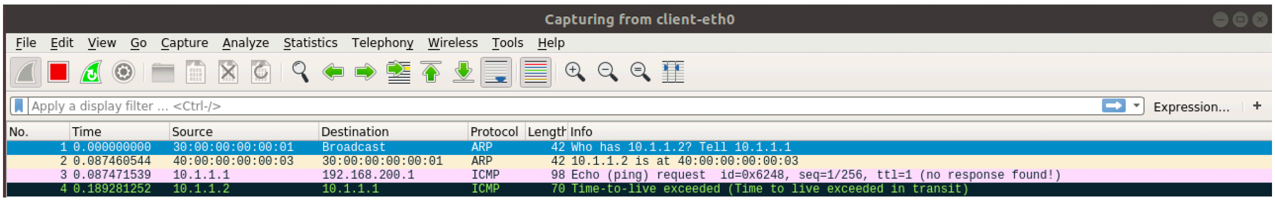
`$ swyard -t myrouter3_testscenario.srpy myrouter.py`



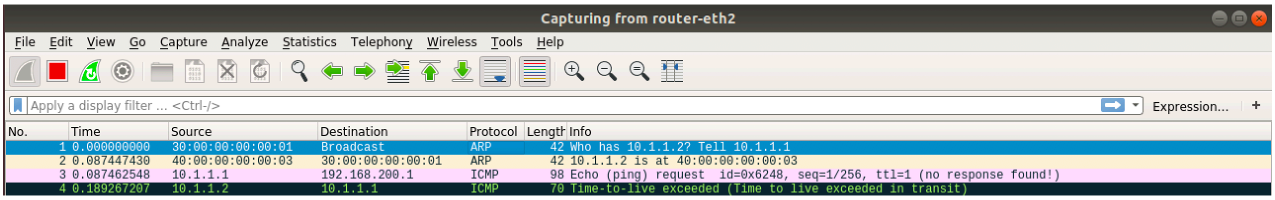
```
Results for test scenario IP forwarding and ARP requester tests: 28 passed, 0 fa
iled, 0 pending


Passed:
1   ICMP echo request (PING) for the router IP address
    192.168.1.1 should arrive on router-eth0.  This PING is
    directed at the router, and the router should respond with
    an ICMP echo reply.
2   Router should send an ARP request for 10.10.1.254 out
    router-eth1.
3   Router should receive ARP reply for 10.10.1.254 on router-
    eth1.
4   Router should send ICMP echo reply (PING) to 172.16.111.222
    out router-eth1 (that's right: ping reply goes out a
    different interface than the request).
5   ICMP echo request (PING) for the router IP address 10.10.0.1
    should arrive on router-eth1.
6   Router should send ICMP echo reply (PING) to 172.16.111.222
    out router-eth1.
7   ICMP echo request (PING) for 10.100.1.1 with a TTL of 1
    should arrive on router-eth1.  The router should decrement
```

## Step 5: Deploy

- `client# ping -c 1 -t 1 192.168.200.1`


Capturing from client-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

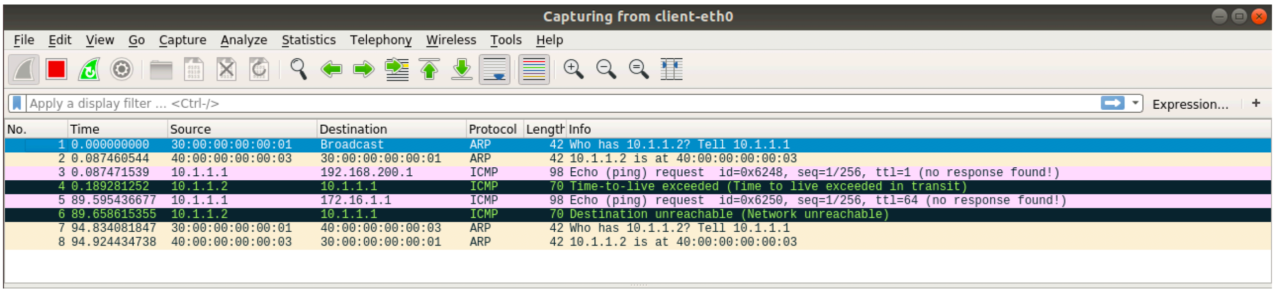
```
No.  Time         Source             Destination        Protocol Length Info
   1 0.000000000  30:00:00:00:00:01  Broadcast          ARP        42 Who has 10.1.1.2? Tell 10.1.1.1
   2 0.087460544  40:00:00:00:00:03  30:00:00:00:00:01  ARP        42 10.1.1.2 is at 40:00:00:00:00:03
   3 0.087471539  10.1.1.1           192.168.200.1      ICMP       98 Echo (ping) request  id=0x6248, seq=1/256, ttl=1 (no response found!)
   4 0.189281252  10.1.1.2           10.1.1.1           ICMP       70 Time-to-live exceeded (Time to live exceeded in transit)
```


Capturing from router-eth2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

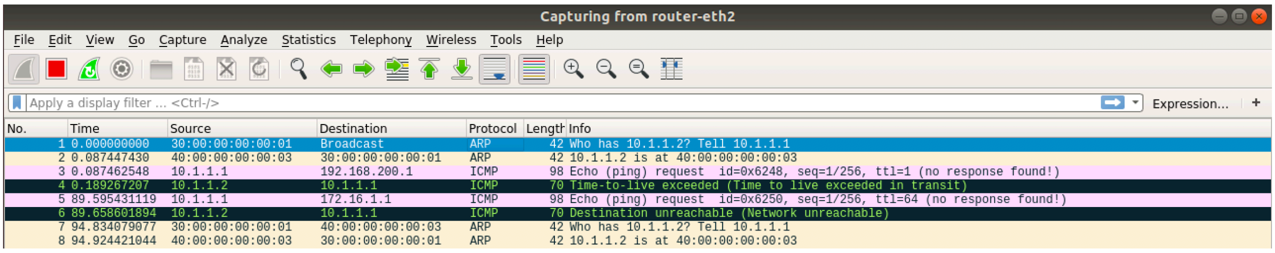
```
No.  Time         Source             Destination        Protocol Length Info
   1 0.000000000  30:00:00:00:00:01  Broadcast          ARP        42 Who has 10.1.1.2? Tell 10.1.1.1
   2 0.087447430  40:00:00:00:00:03  30:00:00:00:00:01  ARP        42 10.1.1.2 is at 40:00:00:00:00:03
   3 0.087462548  10.1.1.1           192.168.200.1      ICMP       98 Echo (ping) request  id=0x6248, seq=1/256, ttl=1 (no response found!)
   4 0.189267207  10.1.1.2           10.1.1.1           ICMP       70 Time-to-live exceeded (Time to live exceeded in transit)
```

The initial TTL in the ICMP packets is set to be 1, so an ICMP time exceeded error.

- `client# ping -c 1 172.16.1.1`


Capturing from client-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

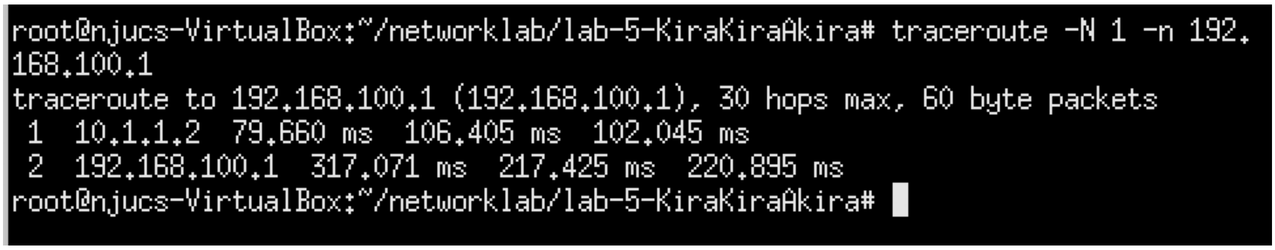
```
No.  Time          Source             Destination        Protocol Length Info
   1 0.000000000   30:00:00:00:00:01  Broadcast          ARP        42 Who has 10.1.1.2? Tell 10.1.1.1
   2 0.087460544   40:00:00:00:00:03  30:00:00:00:00:01  ARP        42 10.1.1.2 is at 40:00:00:00:00:03
   3 0.087471539   10.1.1.1           192.168.200.1      ICMP       98 Echo (ping) request  id=0x6248, seq=1/256, ttl=1 (no response found!)
   4 0.189281252   10.1.1.2           10.1.1.1           ICMP       70 Time-to-live exceeded (Time to live exceeded in transit)
   5 89.595436677  10.1.1.1           172.16.1.1         ICMP       98 Echo (ping) request  id=0x6250, seq=1/256, ttl=64 (no response found!)
   6 89.658615355  10.1.1.2           10.1.1.1           ICMP       70 Destination unreachable (Network unreachable)
   7 94.834081847  30:00:00:00:00:01  40:00:00:00:00:03  ARP        42 Who has 10.1.1.2? Tell 10.1.1.1
   8 94.924434738  40:00:00:00:00:03  30:00:00:00:00:01  ARP        42 10.1.1.2 is at 40:00:00:00:00:03
```


Capturing from router-eth2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

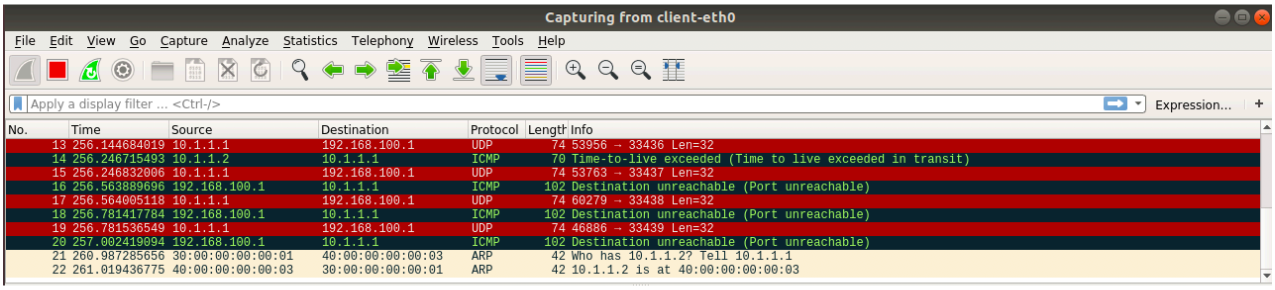
```
No.  Time          Source             Destination        Protocol Length Info
   1 0.000000000   30:00:00:00:00:01  Broadcast          ARP        42 Who has 10.1.1.2? Tell 10.1.1.1
   2 0.087447430   40:00:00:00:00:03  30:00:00:00:00:01  ARP        42 10.1.1.2 is at 40:00:00:00:00:03
   3 0.087462548   10.1.1.1           192.168.200.1      ICMP       98 Echo (ping) request  id=0x6248, seq=1/256, ttl=1 (no response found!)
   4 0.189267207   10.1.1.2           10.1.1.1           ICMP       70 Time-to-live exceeded (Time to live exceeded in transit)
   5 89.595431119  10.1.1.1           172.16.1.1         ICMP       98 Echo (ping) request  id=0x6250, seq=1/256, ttl=64 (no response found!)
   6 89.658601894  10.1.1.2           10.1.1.1           ICMP       70 Destination unreachable (Network unreachable)
   7 94.834079077  30:00:00:00:00:01  40:00:00:00:00:03  ARP        42 Who has 10.1.1.2? Tell 10.1.1.1
   8 94.924421044  40:00:00:00:00:03  30:00:00:00:00:01  ARP        42 10.1.1.2 is at 40:00:00:00:00:03
```

Send a ping from the client to an address that doesn't have a match in the router's forwarding table. So it results in an ICMP destination net unreachable message sent back to the client.

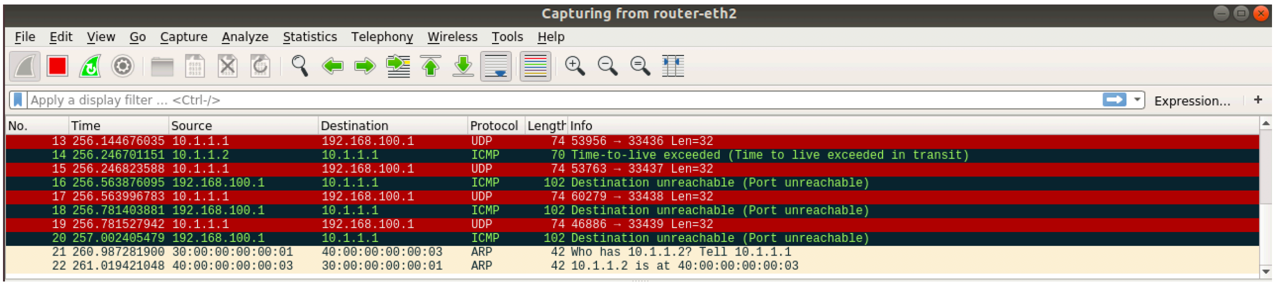
- `client# traceroute -N 1 -n 192.168.100.1`



```
root@njucs-VirtualBox:~/networklab/lab-5-KiraKiraAkira# traceroute -N 1 -n 192.
168.100.1
traceroute to 192.168.100.1 (192.168.100.1), 30 hops max, 60 byte packets
 1  10.1.1.2  79.660 ms  106.405 ms  102.045 ms
 2  192.168.100.1  317.071 ms  217.425 ms  220.895 ms
root@njucs-VirtualBox:~/networklab/lab-5-KiraKiraAkira# 
```


Capturing from client-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.  Time           Source             Destination        Protocol Length Info
  13 256.144684019  10.1.1.1           192.168.100.1      UDP        74 53956 → 33436 Len=32
  14 256.246715493  10.1.1.2           10.1.1.1           ICMP       70 Time-to-live exceeded (Time to live exceeded in transit)
  15 256.246832006  10.1.1.1           192.168.100.1      UDP        74 53763 → 33437 Len=32
  16 256.563889696  192.168.100.1      10.1.1.1           ICMP      102 Destination unreachable (Port unreachable)
  17 256.564005118  10.1.1.1           192.168.100.1      UDP        74 60279 → 33438 Len=32
  18 256.781417784  192.168.100.1      10.1.1.1           ICMP      102 Destination unreachable (Port unreachable)
  19 256.781536549  10.1.1.1           192.168.100.1      UDP        74 46886 → 33439 Len=32
  20 257.002419094  192.168.100.1      10.1.1.1           ICMP      102 Destination unreachable (Port unreachable)
  21 260.987285656  30:00:00:00:00:01  40:00:00:00:00:03  ARP        42 Who has 10.1.1.2? Tell 10.1.1.1
  22 261.019436775  40:00:00:00:00:03  30:00:00:00:00:01  ARP        42 10.1.1.2 is at 40:00:00:00:00:03
```



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 13 | 256.144676035 | 10.1.1.1 | 192.168.100.1 | UDP | 74 | 53956 → 33436 Len=32 |
| 14 | 256.246701151 | 10.1.1.2 | 10.1.1.1 | ICMP | 70 | Time-to-live exceeded (Time to live exceeded in transit) |
| 15 | 256.246823588 | 10.1.1.1 | 192.168.100.1 | UDP | 74 | 53763 → 33437 Len=32 |
| 16 | 256.563876095 | 192.168.100.1 | 10.1.1.1 | ICMP | 102 | Destination unreachable (Port unreachable) |
| 17 | 256.563996783 | 10.1.1.1 | 192.168.100.1 | UDP | 74 | 60279 → 33438 Len=32 |
| 18 | 256.781403881 | 192.168.100.1 | 10.1.1.1 | ICMP | 102 | Destination unreachable (Port unreachable) |
| 19 | 256.781527942 | 10.1.1.1 | 192.168.100.1 | UDP | 74 | 46886 → 33439 Len=32 |
| 20 | 257.002405479 | 192.168.100.1 | 10.1.1.1 | ICMP | 102 | Destination unreachable (Port unreachable) |
| 21 | 260.987281900 | 30:00:00:00:00:01 | 40:00:00:00:00:03 | ARP | 42 | Who has 10.1.1.2? Tell 10.1.1.1 |
| 22 | 261.019421048 | 40:00:00:00:00:03 | 30:00:00:00:00:01 | ARP | 42 | 10.1.1.2 is at 40:00:00:00:00:03 |

# 4. 总结与感想

Lab 5 is a sum up of lab3 and lab4. In lab3 and lab4, we have modules and in lab5 we need to combine them to deal with different kinds of coming packets. It is much harder than labs before, and I learnt a lot from building a router.

Luckily, I passed all tests, but my router is not that efficient. I simply use a queue to store all waiting packets and only process the head of queue. It is easy to implement, but I doubt its efficiency. Since time is limited, I didn't improve my version. Hope to see a much faster way to handle the waiting the packets.