

LAB5实验报告

专业	学号	姓名	开始/结束日期	Email
计算机科学与技术系	191220156	张桓	5.15~5.16	2659428453@qq.com

LAB5实验报告

实验名称

实验目的

实验内容

Task 2: Responding to ICMP echo requests

实现逻辑

核心代码

Task 3: Generating ICMP error messages

实现逻辑

核心代码

实验结果

在mininet上部署测试

测试方法

分析过程

总结感想

实验名称

Respond to ICMP

实验目的

本实验作为实现 IPv4 路由器的最后一个阶段，在此之前我们实现的路由器可以对 ARP request 包做出回应；能够将收到的 IP 数据包正确转发，为了实现这一点我们的路由器可以对于未知 MAC 地址的目的 IP 做出 ARP request。

本阶段我们将实现路由器的最后两个功能：

- 能够响应 ICMP 请求消息，比如 echo request (pings)
- 并且能够在必要时生成 ICMP 错误消息，例如当 IP 包的 TTL (生存时间)值减为零时。

实验内容

Task 2: Responding to ICMP echo requests

实现逻辑

- 路由器在收到包之后，如果想要做出对于 ICMP echo request 的回应，必须满足以下条件：
 - 首先判断是否为 IP 包，即 `packet[Ethernet].ethertype == EtherType.IPv4`
 - 再检查包的**目的地址**是否属于路由器的某个接口的地址，即必须 `ipv4.dst in myIPs`
 - 且该数据包必须是一个 ICMP echo request 包
- 如果收到的包满足上面三点，那么路由器就必须做出 ICMP echo reply 给源主机，按照以下步骤：
 - 复制 EchoRequest 的 `sequence` 和 `data` 还有 `identifier` 到自己构造的 Echoreply 包中。
 - 构造 IPv4 头，根据 rfc 792，地址只需要将原来 request 的源和目的**简单交换**即可。

Addresses

The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

- **转发**构造好的数据包，调用 `send_mypacket(packet, normally = 1)`。

为了简化代码我将 lab4 中的转发功能封装成了一个成员函数 `send_mypacket()`，该函数的定义在[这里](#)。该函数中包含转发数据包时**查询转发表**，**查询ARP表**，**维护等待队列**的完整过程。

核心代码

```
1 elif ipv4.dst in self.myIPs: #the dest is in router
2     #print(type(ICMP_header))
3     if packet.has_header(ICMP):
4         ICMP_header = packet.get_header(ICMP)
5         if ICMP_header.icmp_type == ICMPType.EchoRequest: #construct a ICMP echo
reply
6             print(f"\033[1;36mreceive a echo_request for port's IP:
{ipv4.dst}\033[0m")
7             ICMPpkt = ICMP()
8             ICMPpkt.icmp_type = ICMPType.EchoReply
9             ICMPpkt.icmp_code = ICMPCodeEchoReply.EchoReply
10            ICMPpkt.icmpdata.data = ICMP_header.icmpdata.data
11            ICMPpkt.icmpdata.sequence = ICMP_header.icmpdata.sequence
12            ICMPpkt.icmpdata.identifier = ICMP_header.icmpdata.identifier
13
14            IP_pkt = IPv4()
15            IP_pkt.protocol = IPProtocol.ICMP
16            IP_pkt.ttl = 64
17            IP_pkt.dst = ipv4.src
18            IP_pkt.src = ipv4.dst
```

```

19
20     eth_pkt = Ethernet()
21     eth_pkt.ethertype = EtherType.IPv4
22     eth_pkt.dst = eth_head.src
23     eth_pkt.src = eth_head.dst
24     my_packet = eth_pkt + IP_pkt + ICMPpkt
25     print(f"send a ICMP_reply for the src:{IP_pkt.dst}")
26     self.send_mypacket(my_packet, from_intf, normally = 1)

```

Task 3: Generating ICMP error messages

之前的实验中都没有考虑对于路由器收到的特殊情况的数据包怎么处理，比如目的 IP 在转发表中找不到匹配项，或者 IP 包的 TTL 已经为0，对于这些特殊情况的数据包，路由器需要发出对应的 ICMP error 包。

实现逻辑

- 路由器收到一个数据包后，先判断这个数据包是否属于**某些特殊情况**，并且根据不同的特殊包构造出对应的 ICMP 回应包，特殊情况及其对应的回应包如下：
 - 数据包的目的 IP 地址在**转发表中找不到匹配**，路由器需要给该包的源主机发送一个 ICMPType 为 destination unreachable，ICMPCode 为 host unreachable 的包。
 - 数据包的 TTL 为零，且如果该包的**目的地址不是路由器端口**，那么包的寿命已尽。则路由器需要给该包的源主机发送一个 ICMPType 为 TimeExceeded，ICMPCode 为 TTL expired 的包。
 - 路由器对于某个目的 IP 的 ARP request 的5次请求后还没有收到 ARP reply，那么需要对等待队列中每个数据包的源主机发送一个 ICMPType 为 destination unreachable，ICMPCode 为 host unreachable 的包。
 - 数据包的目的 IP 地址属于路由器的接口地址，但是不是 ICMP echo request 包，这时路由器需要为源主机发送一个 ICMPType 为 destination unreachable，ICMPCode 为 port unreachable 的包。
- 构造这些 ICMP error 包工作比较重复，因此可以将其封装成一个函数 make_ICMP()。

核心代码

- 定义构造 ICMP error 包的函数 make_ICMP(self, hwsrc, hwdst, ipsrc, ipdst, icmp_type, icmp_code, origpkt)，其中 hwsrc, hwdst, ipsrc, ipdst 是包的 MAC 及其 IP 源和目的地址，icmp_type, icmp_code 是包指示的错误类型及其编码，origpkt 是出错的原包，根据手册我们知道构造包的 icmpdata 字段必须包含原始包以 IPv4 开头的28个字节，函数完整定义如下：

```

1  def make_ICMP(self, hwsrc, hwdst, ipsrc, ipdst, icmp_type, icmp_code, origpkt):
2      ether = Ethernet()
3      ether.src = EthAddr(hwsrc)
4      ether.dst = EthAddr(hwdst)
5      ether.ethertype = EtherType.IP
6
7      ippkt = IPv4()
8      ippkt.src = IPAddr(ipsrc)

```

```

9     ippkt.dst = IPAddr(ipdst)
10    ippkt.protocol = IPPROTO_ICMP
11    ippkt.ttl = 64
12
13    icmppkt = ICMP()
14    icmppkt.icmptype = icmp_type
15    icmppkt.icmpcode = icmp_code
16
17    i = origpkt.get_header_index(Ethernet)
18    del origpkt[i]
19    icmppkt.icmpdata.data = origpkt.to_bytes()[28:]
20
21    print("construct a ICMP error packet")
22    return ether + ippkt + icmppkt

```

特别注意!!!:

在构造 ICMP error 包时的设置源 MAC，目的 MAC 的意义不大，因为构造完之后就调用 `sendmypacket()` 函数，通过查询转发表来确定这个包到底从哪个端口出去，会重新修改包的源 MAC 为出端口的 MAC，目的 MAC 为下一跳的 MAC。并且源 IP 也有可能被修改，当包为路由器构造的时候，源 IP 也为出端口的 IP。只有目的 IP 不会被修改，一直都是原始数据包的源 IP。

在 RFC 手册中我只找到了对于 ICMP reply 包的源 IP 规定为原来包的目的 IP，其他的 ICMP error 包源 IP 并没有说明)

因此函数中的参数 `hwsrc`，`hwdst` 的设置其实可有可无，在构造 ICMP error 包的时候调用 `make_ICMP` 函数时，我之前的想法是从入端口发出，源地址设置成入端口，但是由于路由并不可逆，详见总结感想部分，因此这两个参数的意义实际上就相当于一个临时填写的变量，在之后查表转发时立刻就会被更改。

下面定义的 `sendmypacket(packet, from_intf)` 中的 `from_intf` 参数只是为了迎合上面所说的逻辑，在将数据包加入等待队列时同时记录该包的入端口，当然根据上面的分析这样做的意义并不大。

- 定义转发函数 `sendmypacket(packet, from_intf, normally = 0)` :
 - 其中 `packet` 参数是需要发出去的包。这个函数包含将 `packet` 包转发出去的完整过程：查询转发表，查询 ARP 表，若 ARP 表查不到发出 ARP request，然后在转发表对应的出端口调用 `send_packet()` 函数转发出去。
 - 其中 `normally = 0` 参数表示这个转发的这个包是自己构造的 ICMP error 包，源 IP 需要设置成出端口，而转发正常包的时候 `normally = 1`，无需改动源 IP，还有发出 ICMP reply 时也不需要改动源 IP。在调用该函数时需要设置好，显然转发 ICMP error 时不需要设置，因为默认 `normally` 为 0，转发正常包的时候需要显式设置值为 1，我们将在紧接着下一部分看到调用该函数的代码细节。

```

1  def sendmypacket(self, packet, from_intf, normally = 0):
2      ipv4 = packet.get_header(IPv4)
3      for item in self.fw_table:

```

```

4         if ipv4.dst in item[0]: # the dst addr is belong item's netaddr
5             nextHop = item[2] if item[2] is not None else ipv4.dst
6             out_intf = self.net.interface_by_name(item[3])
7             if nextHop in self.arp_table.keys(): #nextHop is in arp_table, send
directly
8                 print(f"the nextHop's IP:{nextHop} has been in arp_table")
9                 packet.get_header(Ethernet).src = out_intf.ethaddr
10                packet.get_header(Ethernet).dst = self.arp_table[nextHop]
11                if normally == 0:
12                    packet.get_header(IPv4).src = out_intf.ipaddr
13                    print(f"send the packet {packet} in {out_intf}")
14                    self.net.send_packet(out_intf, packet) #send the new pkt in out
intf
15            else: #nextHop is not in arp_table, make ARP request for nextHop's IP
16                if nextHop not in self.wait_Queue:
17                    print(f"the nextHop's IP:{nextHop} is not in waitQueue")
18                    arp_request = create_ip_arp_request(out_intf.ethaddr,
out_intf.ipaddr, nextHop)
19                    self.net.send_packet(out_intf, arp_request)
20                    self.wait_Queue[nextHop] = [time.time(), 1, out_intf,
arp_request, [[packet, from_intf]]]
21                    print(f"add a nextHop's IP: {nextHop} in waitQueue")
22                else: #nextHop is in wait queue, add the packet in the pkt queue
23                    print(f"the nextHop's IP:{nextHop} has been in waitQueue")
24                    self.wait_Queue[nextHop][4].append([packet, from_intf])
25                break #FUCK!!!

```

- 对于不同 ICMP error 包类型的判断逻辑即包的参数设置如下：下面这部分代码包含 TTLExceeded, host unreachable 和 port unreachable 的情况。ARP failure 的情况放在下文
 - 注意判断时先看目标 IP 是否在路由器中，如果不在路由器：
 - 目标 IP 是否在 fw_table 中找到匹配项，如果找不到就发 ICMP error，并设置 normally = False
 - 目标 IP 在 fw_table 中找到匹配项，这时将 ttl -= 1，然后判断 ttl == 0，如果为0则发 ICMP error，并设置 normally = False
 - 如果最后发现 normally == True，则说明是个可以正常转发的包，正常转发即可。
 - 如果目标 IP 不在路由器中：
 - 若是个 ICMP echo request 包，则回应 echo reply，具体操作见上文，（下面的代码用...省略了
 - 如果不是，那么就需要回应 ICMP error 包。
 - 转发包时通过自己封装好的 send_mypacket() 函数实现，该函数内部封装了查询转发表，查询 ARP 表维护等待队列的完整流程，详情在这里

```

1     if ipv4.dst not in self.myIPs: # the dest is not in router
2         normally = True
3         in_fw_table = False
4         for item in self.fw_table:
5             if ipv4.dst in item[0]:
6                 in_fw_table = True

```

```

7         if in_fw_table == False: #if dst's IP can't find in fw_table
8             normally = False
9             print(f"the dst's IP: {ipv4.dst} can't find in fw_table")
10            my_packet = self.make_ICMP(eth_head.dst, eth_head.src,
11            from_intf.ipaddr, ipv4.src, ICMPType.DestinationUnreachable, 0, packet)
12            print(f"send a ICMP_error for the src:{ipv4.src}")
13            self.send_mypacket(my_packet, from_intf)
14        elif in_fw_table = True:
15            packet.get_header(IPv4).ttl -= 1
16            if packet.get_header(IPv4).ttl == 0:
17                normally = False #the dst's IP is not in router and the TTL is 0
18                print(f"the packet's TTL is 0 and dst isnot route")
19                my_packet = self.make_ICMP(eth_head.dst, eth_head.src,
20                from_intf.ipaddr, ipv4.src, ICMPType.TimeExceeded, 0, packet)
21                print(f"send a ICMP_error for the src:{ipv4.src}")
22                self.send_mypacket(my_packet, from_intf)
23            if normally == True:
24                print(f"receive a normal packet and foward it normally")
25                self.send_mypacket(packet, from_intf, normally = 1) #or foward the
26            packet
27        elif ipv4.dst in self.myIPs: #the dest is in router
28            if packet.has_header(ICMP):
29                ICMP_header = packet.get_header(ICMP)
30                if ICMP_header.icmptype == ICMPType.EchoRequest:
31                    ...#construct a ICMP echo reply
32                else: #is not ICMP request
33                    print(f"the packet for router isnot ICMP_request")
34                    my_packet = self.make_ICMP(eth_head.dst, eth_head.src,
35                    from_intf.ipaddr, ipv4.src, ICMPType.DestinationUnreachable, 3, packet)
36                    print(f"send a ICMP_error for the src:{ipv4.src}")
37                    self.send_mypacket(my_packet, from_intf)

```

- ARP failure 的情况：显然判断这种情况应该放在等待队列更新的那部分，当对某个 nextHop 的请求超过5次后，for 循环等待队列的每个包的源主机回复一个 ICMP error 包，再将这一项从队列中丢弃。需要注意的是，如果等待队列中有多个包的源主机是一样的，那么只需要向这个源主机发送一个 ICMP error 就够了。在下面的代码中通过遍历每个包，利用集合 have_done 来保证路由器向每个源主机只发一个 ICMP error 包。

```

1  if(send_counts <= 4):#repeat again
2      self.net.send_packet(self.wait_Queue[nextHop][2], self.wait_Queue[nextHop]
3      [3])
4      self.wait_Queue[nextHop][0] = time.time()
5      self.wait_Queue[nextHop][1] = send_counts + 1
6  else: #ARP failure, construct a ICMP error
7      print(f"a ARP failure for nextHop's IP:{nextHop}")
8      have_done = []
9      for wait_pkt, from_intf in self.wait_Queue[nextHop][4]:
10          eth_head = wait_pkt.get_header(Ethernet)
11          ipv4 = wait_pkt.get_header(IPv4)
12          if ipv4.src not in have_done:

```

```

12         my_packet = self.make_ICMP(from_intf.ethaddr, eth_head.src,
    from_intf.ipaddr, ipv4.src, ICMPType.DestinationUnreachable, 1, wait_pkt)
13         self.send_mypacket(my_packet, from_intf)
14         have_done.append(ipv4.src)
15         self.wait_Queue.pop(nextHop) #delete this ip

```

调用 `make_ICMP()` 函数参数中的 `icmp_code` 字段是通过查询手册得到的, 比如 `ICMPType == DestinationUnreachable` 时, 若要表示 `hostunreachable` 则 `icmp_code` 应设置为1.

实验结果

在虚拟环境下通过 `testcase/router3_testscenario.srpy` 所有测试用例, 结果如下:

```

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.

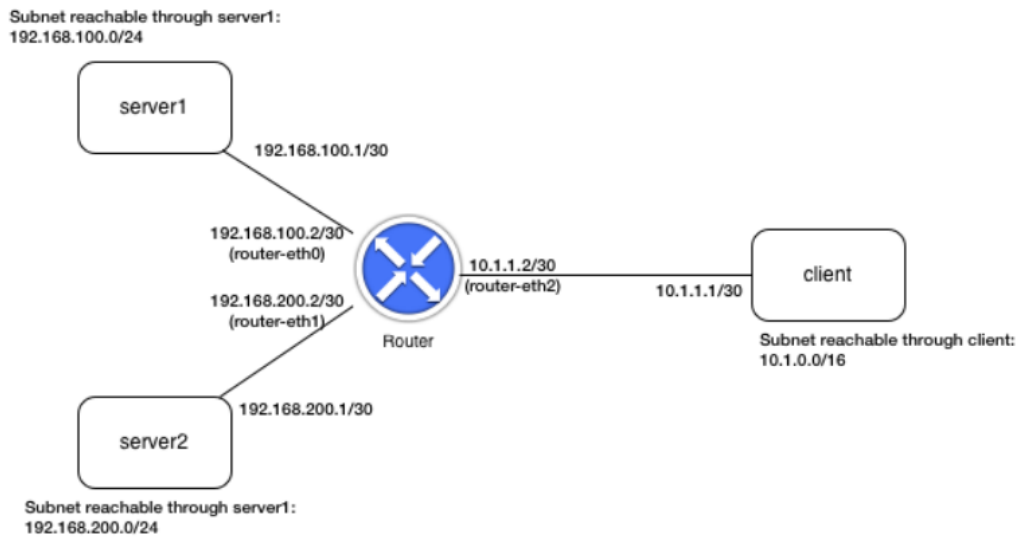
All tests passed!

```

在mininet上部署测试

测试方法

`mininet` 上部署的网络拓扑如下:



在 router 上的 xterm 上运行 myrouter3.py , 然后 router wireshark & 监控 router-eth0 和 riuter-eth2 两个端口, 构造流量 server1 traceroute 10.1.1.1 , 在终端上得到如下信息:

```
mininet> server1 traceroute 10.1.1.1
traceroute to 10.1.1.1 (10.1.1.1), 30 hops max, 60 byte packets
 1 192.168.100.2 (192.168.100.2) 195.846 ms 197.309 ms 198.697 ms
 2 10.1.1.1 (10.1.1.1) 406.387 ms 407.083 ms 407.765 ms
```

分析过程

- 首先路由器在 eth0 收到 ARP request 并且发出回应, 然后紧接着收到 TTL = 1 的包, 显然这个包到达路由器寿命会耗尽, 可以看到路由器随后在 eth0 上也发出了响应的 ICMP error 包, 类型为 TimeExceeded 。这个包的源 IP 地址就是出端口 eth0 的 IP 地址 192.168.100.2 。

正在捕获 router-eth0

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	0.006981480	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	0.006995610	192.168.100.1	10.1.1.1	UDP	74	54803 → 33434 Len=32
4	0.006996683	192.168.100.1	10.1.1.1	UDP	74	44182 → 33435 Len=32
5	0.006998011	192.168.100.1	10.1.1.1	UDP	74	52673 → 33436 Len=32
6	0.006998937	192.168.100.1	10.1.1.1	UDP	74	59003 → 33437 Len=32
7	0.006999818	192.168.100.1	10.1.1.1	UDP	74	45696 → 33438 Len=32
8	0.007000584	192.168.100.1	10.1.1.1	UDP	74	34366 → 33439 Len=32
9	0.007001470	192.168.100.1	10.1.1.1	UDP	74	36702 → 33440 Len=32
10	0.007002281	192.168.100.1	10.1.1.1	UDP	74	41849 → 33441 Len=32
11	0.007003108	192.168.100.1	10.1.1.1	UDP	74	43852 → 33442 Len=32
12	0.007003869	192.168.100.1	10.1.1.1	UDP	74	56659 → 33443 Len=32
13	0.007004692	192.168.100.1	10.1.1.1	UDP	74	45388 → 33444 Len=32
14	0.007005986	192.168.100.1	10.1.1.1	UDP	74	43650 → 33445 Len=32
15	0.007006808	192.168.100.1	10.1.1.1	UDP	74	35808 → 33446 Len=32
16	0.007007658	192.168.100.1	10.1.1.1	UDP	74	46218 → 33447 Len=32
17	0.007008673	192.168.100.1	10.1.1.1	UDP	74	52904 → 33448 Len=32
18	0.007009461	192.168.100.1	10.1.1.1	UDP	74	48190 → 33449 Len=32
19	0.195821816	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
20	0.197228107	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)

应用显示过滤器: <Ctrl-/> 表达式: +

0 0000 0000 0000 = Fragment offset: 0

Time to live: 1

Protocol: UDP (17)

Header checksum: 0xf844 [validation disabled]

[Header checksum status: Unverified]

0000 40 00 00 00 00 01 10 00 00 00 00 01 08 00 45 00 @.....E.

0010 00 3c 91 c1 00 00 01 11 f8 44 c0 a8 64 01 0a 01 <.....D..d..

0020 01 01 d6 13 82 9a 00 28 2f e5 40 41 42 43 44 45(./@ABCDE

- 由于 traceroute 是连续发三个一样的包, 之后的两个 TTL = 1 的包路由器也会在 eth0 上发出 ICMP error 包, 如下: 包 3 4 5 对应的 TimeExceeded 包为 19 20 21 。

正在捕获 router-eth0

应用显示过滤器: <Ctrl-/>

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	0.000000000	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	0.006995610	192.168.100.1	10.1.1.1	UDP	74	54803 -> 33434 Len=32
4	0.006996683	192.168.100.1	10.1.1.1	UDP	74	44182 -> 33435 Len=32
5	0.006998011	192.168.100.1	10.1.1.1	UDP	74	52673 -> 33436 Len=32
6	0.006998937	192.168.100.1	10.1.1.1	UDP	74	59003 -> 33437 Len=32
7	0.006999818	192.168.100.1	10.1.1.1	UDP	74	45696 -> 33438 Len=32
8	0.007000584	192.168.100.1	10.1.1.1	UDP	74	34366 -> 33439 Len=32
9	0.007001470	192.168.100.1	10.1.1.1	UDP	74	36702 -> 33440 Len=32
10	0.007002281	192.168.100.1	10.1.1.1	UDP	74	41849 -> 33441 Len=32
11	0.007003108	192.168.100.1	10.1.1.1	UDP	74	43852 -> 33442 Len=32
12	0.007003869	192.168.100.1	10.1.1.1	UDP	74	56659 -> 33443 Len=32
13	0.007004692	192.168.100.1	10.1.1.1	UDP	74	45388 -> 33444 Len=32
14	0.007005596	192.168.100.1	10.1.1.1	UDP	74	43650 -> 33445 Len=32
15	0.007006808	192.168.100.1	10.1.1.1	UDP	74	35808 -> 33446 Len=32
16	0.007007658	192.168.100.1	10.1.1.1	UDP	74	46218 -> 33447 Len=32
17	0.007008673	192.168.100.1	10.1.1.1	UDP	74	52904 -> 33448 Len=32
18	0.007009461	192.168.100.1	10.1.1.1	UDP	74	48190 -> 33449 Len=32
19	0.195821816	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
20	0.197328197	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	0.198728835	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)

...0 0000 0000 0000 = Fragment offset: 0

Time to live: 1

Protocol: UDP (17)

Header checksum: 0xf842 [validation disabled]

[Header checksum status: Unverified]

- 然后路由器会在 eth0 收到 server1 发出的 TTL = 2 的数据包，可以看段这个包的信息为 59003 -> 33437，这个包需要正常转发，

正在捕获 router-eth0

应用显示过滤器: <Ctrl-/>

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	0.000000000	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	0.006995610	192.168.100.1	10.1.1.1	UDP	74	54803 -> 33434 Len=32
4	0.006996683	192.168.100.1	10.1.1.1	UDP	74	44182 -> 33435 Len=32
5	0.006998011	192.168.100.1	10.1.1.1	UDP	74	52673 -> 33436 Len=32
6	0.006998937	192.168.100.1	10.1.1.1	UDP	74	59003 -> 33437 Len=32
7	0.006999818	192.168.100.1	10.1.1.1	UDP	74	45696 -> 33438 Len=32
8	0.007000584	192.168.100.1	10.1.1.1	UDP	74	34366 -> 33439 Len=32
9	0.007001470	192.168.100.1	10.1.1.1	UDP	74	36702 -> 33440 Len=32
10	0.007002281	192.168.100.1	10.1.1.1	UDP	74	41849 -> 33441 Len=32
11	0.007003108	192.168.100.1	10.1.1.1	UDP	74	43852 -> 33442 Len=32
12	0.007003869	192.168.100.1	10.1.1.1	UDP	74	56659 -> 33443 Len=32
13	0.007004692	192.168.100.1	10.1.1.1	UDP	74	45388 -> 33444 Len=32
14	0.007005596	192.168.100.1	10.1.1.1	UDP	74	43650 -> 33445 Len=32
15	0.007006808	192.168.100.1	10.1.1.1	UDP	74	35808 -> 33446 Len=32
16	0.007007658	192.168.100.1	10.1.1.1	UDP	74	46218 -> 33447 Len=32
17	0.007008673	192.168.100.1	10.1.1.1	UDP	74	52904 -> 33448 Len=32
18	0.007009461	192.168.100.1	10.1.1.1	UDP	74	48190 -> 33449 Len=32
19	0.195821816	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
20	0.197328197	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	0.198728835	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)

...0 0000 0000 0000 = Fragment offset: 0

Time to live: 2

Protocol: UDP (17)

- 在 eth2 端口转发这个包，转发之前路由器并不知道 10.1.1.1 的 MAC 地址，因此先在 eth2 上发出 ARP request 并且收到了回应。然后正式转发这个包，在下图红框部分可以看到信息同样为 59003 -> 33437，说明确实是这个包。由于端口不可达，路由器会收到来自 client 的 ICMP error 包，类型为 Destination unreachable，见包4。

正在捕获 router-eth2

应用显示过滤器: <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	40:00:00:00:00:03	Broadcast	ARP	42	Who has 10.1.1.1? Tell 10.1.1.2
2	0.000015447	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	10.1.1.1 is at 30:00:00:00:00:01
3	0.101064201	192.168.100.1	10.1.1.1	UDP	74	59003 -> 33437 Len=32
4	0.101094900	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
5	0.101335589	192.168.100.1	10.1.1.1	UDP	74	45696 -> 33438 Len=32
6	0.101347665	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
7	0.101545227	192.168.100.1	10.1.1.1	UDP	74	34366 -> 33439 Len=32
8	0.101554554	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
9	0.101744129	192.168.100.1	10.1.1.1	UDP	74	36702 -> 33440 Len=32
10	0.101750270	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
11	0.101934201	192.168.100.1	10.1.1.1	UDP	74	41849 -> 33441 Len=32
12	0.101942819	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
13	0.102125529	192.168.100.1	10.1.1.1	UDP	74	43852 -> 33442 Len=32
14	0.102134023	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
15	0.102357043	192.168.100.1	10.1.1.1	UDP	74	56659 -> 33443 Len=32
16	0.102539915	192.168.100.1	10.1.1.1	UDP	74	45388 -> 33444 Len=32
17	0.102726603	192.168.100.1	10.1.1.1	UDP	74	43650 -> 33445 Len=32
18	0.102911954	192.168.100.1	10.1.1.1	UDP	74	35808 -> 33446 Len=32
19	0.103097404	192.168.100.1	10.1.1.1	UDP	74	46218 -> 33447 Len=32

Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0

Ethernet II, Src: 40:00:00:00:00:03 (40:00:00:00:00:03), Dst: 30:00:00:00:00:01 (30:00:00:00:00:01)

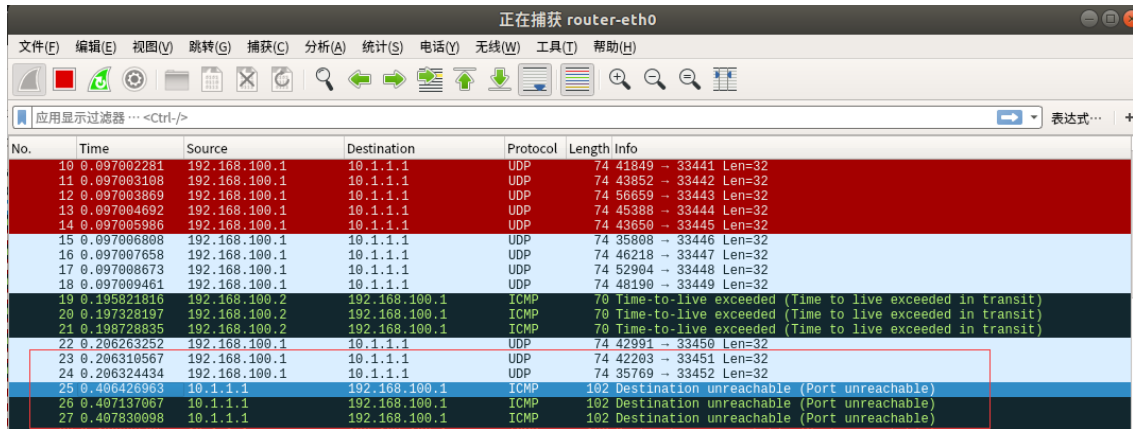
Internet Protocol Version 4, Src: 192.168.100.1, Dst: 10.1.1.1

0000 30 00 00 00 01 40 00 00 00 03 08 00 45 00 0...0...E

0010 00 3c 91 c4 00 01 11 f8 41 c0 a8 64 01 0a 01 <.....A..d...

0020 04 04 04 7b 03 04 00 20 74 44 04 44 43 43 45 f...f...B...P...

- 收到这个由 client 发出的 ICMP error 后，路由器会转发这个包，从 eth0 发出，由于 traceroute 总共发三个一样的包，所以转发了对应的三次 Destination unreachable 包。这个包的源 IP 地址是 client 的 IP 即 10.1.1.1。



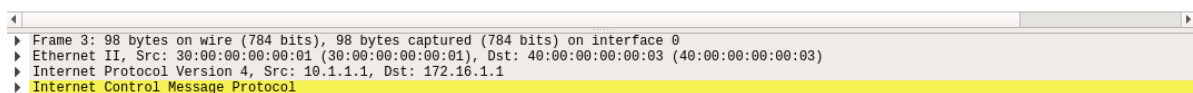
No.	Time	Source	Destination	Protocol	Length	Info
10	0.097002281	192.168.100.1	10.1.1.1	UDP	74	41849 - 33441 Len=32
11	0.097003108	192.168.100.1	10.1.1.1	UDP	74	43852 - 33442 Len=32
12	0.097003860	192.168.100.1	10.1.1.1	UDP	74	56659 - 33443 Len=32
13	0.097004692	192.168.100.1	10.1.1.1	UDP	74	45388 - 33444 Len=32
14	0.097005986	192.168.100.1	10.1.1.1	UDP	74	43650 - 33445 Len=32
15	0.097006808	192.168.100.1	10.1.1.1	UDP	74	35808 - 33446 Len=32
16	0.097007658	192.168.100.1	10.1.1.1	UDP	74	46218 - 33447 Len=32
17	0.097008673	192.168.100.1	10.1.1.1	UDP	74	52904 - 33448 Len=32
18	0.097009461	192.168.100.1	10.1.1.1	UDP	74	48190 - 33449 Len=32
19	0.195821816	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
20	0.197328197	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	0.198728835	192.168.100.2	192.168.100.1	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
22	0.206263252	192.168.100.1	10.1.1.1	UDP	74	42991 - 33450 Len=32
23	0.206310567	192.168.100.1	10.1.1.1	UDP	74	42203 - 33451 Len=32
24	0.206324434	192.168.100.1	10.1.1.1	UDP	74	35769 - 33452 Len=32
25	0.400420953	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
26	0.407137067	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)
27	0.407830098	10.1.1.1	192.168.100.1	ICMP	102	Destination unreachable (Port unreachable)

上面分析过程中的抓包文件已经放在 /report 文件中，命名为 lab_5_1eth0 和 lab_5_1eth2

下面再测试一下目的 IP 转发无匹配项的情况：用 client ping -c 1 172.16.1.1，监控路由器的 eth_2 端口。得到如下信息：



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	0.103579263	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	0.103587396	10.1.1.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x19c4, seq=1/256, ttl=64 (no response found!)
4	0.207576721	10.1.1.2	10.1.1.1	ICMP	70	Destination unreachable (Network unreachable)



Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: 40:00:00:00:00:03 (40:00:00:00:00:03)
Internet Protocol Version 4, Src: 10.1.1.1, Dst: 172.16.1.1
Internet Control Message Protocol

可以看到路由器首先在 eth2 收到 client 的 ARP request 并且做出回应，随后就收到 echo request，但是由于转发无匹配项，又在 eth2 发出 ICMP error 包，类型为 Destination unreachable。

上面分析过程中的抓包文件已经放在 /report 文件中，命名为 lab_5_2eth2

综上可以推断路由器正常工作了。

总结感想

本次实验在上一次实验的基础上，难度并不大，主要是概念问题。重复工作比较多，比如转发包封装成了独立函数，包括构造 ICMP error 包的时候，也是由于重复性比较大独立成了单独函数，因此整体来说工作量不算多。

- 由于写的时候没有留意 FAQ 的内容/(ToT)/~~，开始写 ICMP error 转发的时候我天真的认为转发的出端口就是这个问题包的入端口，更大的问题在于我这么写还过了测试用例☹，毕竟这个测试用例比较简单。。后来通过助教gg在群里的答疑才让我惊坐起，才看到 FAQ 原来有这个问题的描述：

Q: When sending ICMP echo replies or error messages, does the router need to do a forwarding table lookup and send ARP requests if needed? Can't the router just send the ICMP messages back on the interface through which the IP packet was received?

A: The router will still need to do an ARP query as it normally does for forwarding an IP packet. It doesn't matter that an echo request arrives on, say port eth0. The echo reply may end up going out on a different port depending on the forwarding table lookup. The entire lookup and ARP query process should be the same as forwarding an IP packet, and will always behave exactly this way.

也就是说我原来记录的入端口根本没什么实际意义，在转发 ICMP error 包时还是需要走流程，源 MAC 在随后的查询转发表时会被立刻覆盖掉，在查表之前源 MAC 不管写成什么都可以。

助教gg的话还是一针见血：说过了，路由不可逆

总之就是印象十分深刻👏👏👏

- 还有在于理解了 traceroute 的工作原理，以前一直奇怪 traceroute 怎么知道包到达了目的主机，原来它发出的包本身 port 就很大，根本没法匹配，导致目的主机接收到之后得知这是个 traceroute 发出的包，回应一个 port unreachable 的包，这样源主机就知道包已经到达目的地了。
- 再有就是帮助我理清了路由器的完整逻辑，对各种包：ARP 请求包、ARP 回应包、能正常转发的 IP 包，有问题的 IP 包的不同处理方式。
- 参考文章：[RFC792](#)

最后感谢助教的耐心批改😊😊😊