

# 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 助教：方毓楚/于沛文/郑浩/陈伟/李国浩/李睿宸/杨溢...
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：201220096	姓名：钟亚晨
Email: <a href="mailto:2991908515@qq.com">2991908515@qq.com</a>	开始/完成日期：2022.04.26-2022.04.26

## 1.实验名称

Respond to ICMP

完成清单：

- ✧ 实现了对 ICMP echo request 的响应；
- ✧ 实现了在四种必要时刻生成 ICMP error message 并且填写正确的内容，通过正确的端口发送到正确的主机；
- ✧ 通过了框架测试中的所有测试用例；
- ✧ （可选）自己利用模板编写了测试用例，对所有实现功能进行了测试，并且通过了所有测试用例；
- ✧ 利用Mininet进行了仿真测试，一方面通过了实验手册上所给的测试方法，另一方面按要求更换源主机进行了重新测试，并且结果符合预期，利用Wireshark抓到并保存了所有数据包；
- ✧ 基本验证了Lab03-05当中实现的 IPv4 Router 的功能正确性，并且完成了本篇实验报告；

## 2.实验目的

- ✧ 最终实验一个 IPv4 路由器，并且结合 Switchyard 框架以及 Mininet 模拟仿真验证功能实现逻辑的正确性；

## 3.实验内容

- ✧ 完成对 ICMP echo request 的正确响应；
- ✧ 完成在必要时刻（共4种情况），路由器生成 ICMP error message 并且正确发送的功能；
- ✧ 通过框架测试的所有测试用例；
- ✧ 部署到 Mininet 中进行仿真环境测试，并用 Wireshark 抓包进一步验证逻辑实现的正确性；

## 4.实验结果

简单说明，以下对实验说明中的四个问题进行回答。

## 0x01 In the report, show how you implement the logic of responding to ICMP echo requests.

①基于先前的 Lab04，当路由器接收到一个 IPv4 报文，如果其目标IP指向路由器的某一个端口，那么将该报文丢弃。在本次实验中添加分支：如果接收到的 IPv4 报文目标IP指向路由器的端口，并且该报文含有一个 ICMP 标头，且类型指明为 EchoRequest，那么这是一个 ICMP echo request，转入对应的函数进行处理。

```
167         if packet[IPv4].dst == it.ipaddr:
168             icmp_header = packet.get_header(ICMP)
169             # If received a ICMP Request
170             if icmp_header and icmp_header.icmptype == ICMPType.EchoRequest:
171                 self.handle_icmp_request_packet(icmp_header, ifaceName, packet)
172         else:
```

②对 ICMP echo request 的处理：

```
127     ## Handle ICMP Packet##
128     def handle_icmp_request_packet(self, icmp_header, ifaceName, packet):
129         log_info(f'Receive ICMP Request::Src:{packet[IPv4].src} > Dst:{packet[IPv4].dst}')
130         # Build ICMP echo Reply packet
131         echo_reply = ICMP()
132         echo_reply.icmptype = ICMPType.EchoReply
133         echo_reply.icmpdata.sequence = icmp_header.icmpdata.sequence
134         echo_reply.icmpdata.identifier = icmp_header.icmpdata.identifier
135         echo_reply.icmpdata.data = icmp_header.icmpdata.data
136         reply_ip_header = IPv4()
137         reply_ip_header.dst = packet[IPv4].src
138         reply_ip_header.src = packet[IPv4].dst
139         reply_ip_header.ttl = 64
140         ethernet_header = Ethernet()
141         ethernet_header.ethertype = EtherType.IPv4
142         echo_reply_pkt = ethernet_header + reply_ip_header + echo_reply
143         # Lookup forwarding table and add to waiting queue
144         max_matched_prefixlen = 0
145         for it in self.forwarding_table:
146             if echo_reply_pkt[IPv4].dst in it['ipv4network'] and it['ipv4network'].prefixlen > max_matched_prefixlen:
147                 tmp = it['ipv4network']
148                 tmp1 = it['nextip']
149                 log_info(f'echo_reply_pkt[IPv4].dst > {tmp} > {tmp1}')
150                 max_matched_prefixlen = it['ipv4network'].prefixlen
151                 target_item = it
152         if max_matched_prefixlen == 0:
153             return None
154         for it in self.interfaces:
155             if it.name == target_item['interface']:
156                 target_interface = it
157         waitpkt = WaitPacket(target_item['nextip'], target_interface, echo_reply_pkt, ifaceName)
158         self.waiting_queue.append(waitpkt)
```

构造ICMP报文  
填充相关字段

构造IPv4标头

构造以太网头  
成帧

封装到封装体中  
加入到等待队列  
进行发送

主要处理步骤为：

Step1.构建 ICMP echo reply：

ICMP 报文依照实验说明进行构造即可：

- 1.设置 ICMP 类型为 EchoReply 以表征其为 ICMP echo reply；
- 2.将 EchoRequest 中的序列号复制到 EchoReply 当中；
- 3.将 EchoRequest 中的标识符复制到 EchoReply 当中；

IPv4 标头则设置好源IP地址、目标IP地址、TTL 即可：

- 1.源IP地址设置为路由器端口的地址，简单调换 ICMP echo request 的源地址和目的地址即可；
- 2.目标IP地址设置为 EchoRequest 的源主机的IP地址；
3. TTL 设置为通用最优值64；

以太网头则直接构建并指定类型即可，在之后的发送过程中会由 Lab04 中已实现的逻辑代码自动给填充相关字段并进行发送。

Step2.查询路由表，将其封装成等待包并加入到路由器的等待队列中进行发送：

该部分的逻辑实现说明已在 Lab04 中进行实现，此处不再赘述。

## 0x02 ✓ In the report, show how you implement the logic of generating ICMP error messages.

首先 ICMP error messages 一共有四种：

序号	错误类型	错误码	说明
1	/	network unreachable	无法在路由表中找到匹配表项
2	/	TTL expired	转发减少TTL过程中，TTL变为0
3	DestinationUnreachable	host unreachable	重复发送5次ARP未收到回复
4	DestinationUnreachable	port unreachable	目的IP指向一个路由器端口且不为 ICMP Request

而构建 ICMP error messages 报文并将其添加到等待队列进行发送的过程我们可以将其封装到一个函数当中，在对应的错误发生处调用该函数，填入不同参数即可构建并发出对应的 ICMP error messages。

下面是构建函数的实现：

```
86  """ ICMP Error """
87  def icmp_error(self, ori_pkt, err_type, err_code, ifaceName):
88      # Find interface which receive packet causes the ICMP error
89      for it in self.interfaces:
90          if it.name == ifaceName:
91              interface = it
92              break
93      # Build IP Header
94      ip_header = IPv4()
95      ip_header.dst = ori_pkt[IPv4].src
96      ip_header.src = interface.ipaddr
97      ip_header.ttl = 64
98      # Build ICMP error message
99      # Include original packet information(28 bytes)
100     oridata = ori_pkt
101     i = ori_pkt.get_header_index(Ethernet)
102     del ori_pkt[i]
103     icmp = ICMP()
104     icmp.icmptype = err_type
105     icmp.icmpcode = err_code
106     # Mark the length of original packet
107     icmp.icmpdata.origdgramlen = len(ori_pkt)
108     icmp.icmpdata.data = ori_pkt.to_bytes()[i:28]
109     # Build Packet
110     icmp_error_pkt = Ethernet() + ip_header + icmp
111     # Lookup forwarding table and add to waiting queue
112     max_matched_prefixlen = 0
113     for it in self.forwarding_table:
114         if icmp_error_pkt[IPv4].dst in it['ipv4network'] and it['ipv4network'].prefixlen > max_matched_prefixlen:
115             max_matched_prefixlen = it['ipv4network'].prefixlen
116             target_item = it
117     if max_matched_prefixlen == 0:
118         return None
119     for it in self.interfaces:
120         if it.name == target_item['interface']:
121             target_interface = it
122     waitpkt = WaitPacket(target_item['nextip'], target_interface, icmp_error_pkt, ifaceName)
123     tmp = target_item['nextip']
124     log_info(f'Wow!{tmp}')
125     self.waiting_queue.append(waitpkt)
```

依据传入端口名  
找到导致ICMP error的报文的传入端口  
作为ICMP error message的发送端口

构建IPv4标头

依据传入参数  
构建ICMP error message的相关内容  
组装成帧

封装并封装体  
加入等待队列中  
进行发送

①依照传入端口名，找到导致 ICMP error 的报文的传入端口，作为 ICMP error message 的发送端口（从哪来回哪去）。该参数的获取一方面是记录于 WaitPacket 的封装体中，另一方面是在路由器接收到数据包时会指明该包从哪个端口传入。

②构建 IPv4 标头，将源地址设置为发送端口的IP地址，而目标地址设置为导致 ICMP error 的报文的源IP地址，TTL设置为普遍最优值64。

③首先参照示例，将源数据包的去除以太网标头的前28字节（从IPv4标头开始）作为错误信息填入 ICMP error message 中的数据部分，并且指明原始包的长度，并根据传入参数指定 ICMP error message 的错误类型、错误码。最后组装成帧。

④在 Lab04 中已实现的逻辑，最长前缀匹配查询路由表，而后封装成 waitPacket 封装体加入到路由器等待队列中，利用已有逻辑自动将 ICMP error message 发送出去。

下面是函数的调用过程实现：

#### ①无法在路由表中找到匹配表项

```
177         # Search matched item in Forwarding Table
178         max_matched_prefixlen = 0
179         target_item = {}
180         for it in self.forwarding_table:
181             if packet[IPv4].dst in it['ipv4network'] and it['ipv4network'].prefixlen > max_matched_prefixlen:
182                 max_matched_prefixlen = it['ipv4network'].prefixlen
183                 target_item = it
184         # If no find matched item, do nothing
185         if max_matched_prefixlen == 0:
186             self.icmp_error(packet, ICMPType.DestinationUnreachable, 0, ifaceName)
187             return None
```

```
<DestinationUnreachable.NetworkUnreachable: 0>,
```

在对应分支处调用 icmp\_error 函数，并传入错误码 network unreachable 所对应的0。

#### ②转发减少TTL的过程中，TTL变为0

```
193         if max_matched_prefixlen == 0:
194             self.icmp_error(packet, ICMPType.DestinationUnreachable, 0, ifaceName)
195             return None
196         # Decrease TTL
197         # FAQ02 If DestinationUnreachable and TTL Zero at the same time, send Destination
198         packet[IPv4].ttl -= 1
199         if packet[IPv4].ttl == 0:
200             self.icmp_error(packet, ICMPType.TimeExceeded, 0, ifaceName)
201             return None
202         # Find interface
```

依据 Lab05 的FAQ02：当一个报文同时无法在路由表中无匹配项，且其TTL减少为0，此时生成的 ICMP error message 应当为 DestinationUnreachable。并且描述中有说TTL的自减应当是放在查询路由表后。

在对应分支处调用 icmp\_error 函数，并传入错误类型 TimeExceeded。

#### ③重复发送5次ARP Request未收到回复

```
265         # ARP Request time not less than 5
266         elif waitpkt.arp_cnt >= 5:
267             for it in self.waiting_queue:
268                 # FAQ03 only once
269                 if it.packet[IPv4].dst == self.waiting_queue[0].packet[IPv4].dst:
270                     self.icmp_error(it.packet, ICMPType.DestinationUnreachable, 1, it.ifaceName)
271                     self.delete_queue.append(it)
272         # ARP Request time less than 5
```

```
<DestinationUnreachable.HostUnreachable: 1>,
```

依据 Lab05 的FAQ03：如果在ARP解析发生了5次，此时要从等待缓冲中删除报文，并且对于每一个引发错误的报文都要向其源主机发送对应的错误报文。

在对应分支处调用 icmp\_error 函数，并传入错误码 host unreachable 所对应的1。

#### ④目标IP指向路由器的一个端口，并且其不为 ICMP echo request

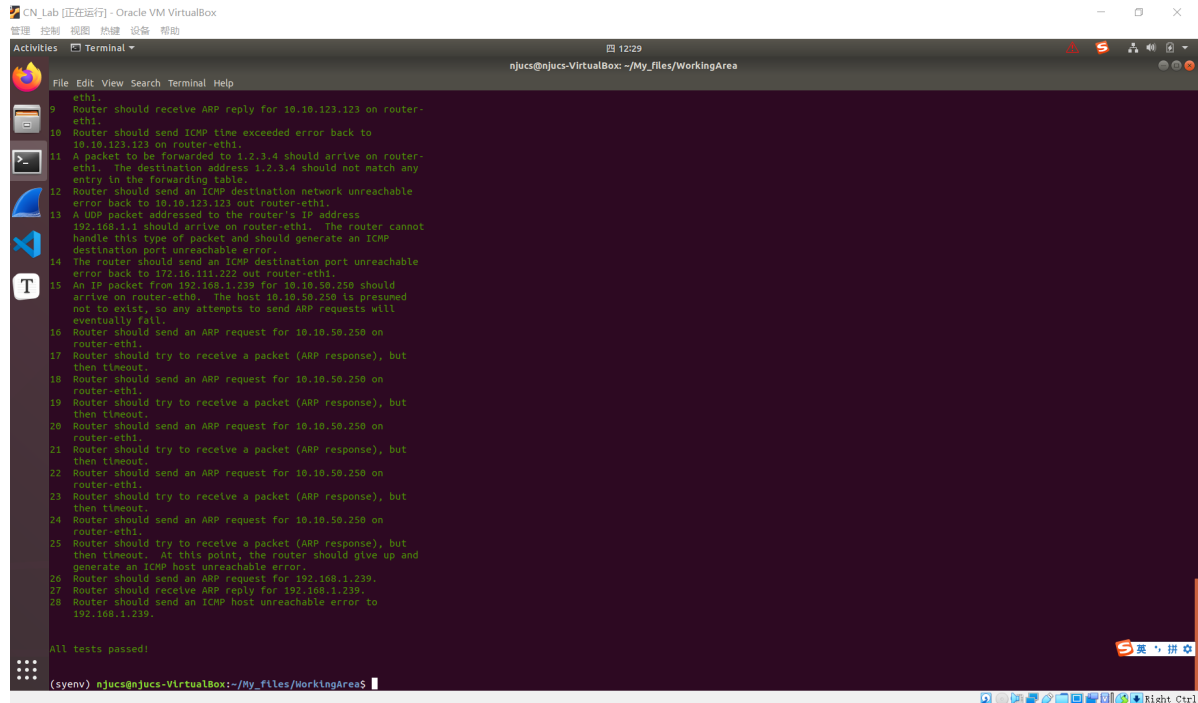
```
167         # If Targetip belongs to one of router interfaces, do nothing
168         for it in self.interfaces:
169             if packet[IPv4].dst == it.ipaddr:
170                 icmp_header = packet.get_header(ICMP)
171                 # If received a ICMP Request
172                 if icmp_header and icmp_header.icmptype == ICMPType.EchoRequest:
173                     self.handle_icmp_request_packet(icmp_header, ifaceName, packet)
174                 else:
175                     self.icmp_error(packet, ICMPType.DestinationUnreachable, 3, ifaceName)
176                 return None
```

```
<DestinationUnreachable.PortUnreachable: 3>,
```

在对应分支处调用 icmp\_error 函数，并传入错误码 port unreachable 所对应的3。

## 0x03 ✓ In the report, show the test result of your router. (Optional) If you have written the test files yourself, show how you test the forwarding packets.

执行框架代码自带的测试文件，可以顺利通过所有的测试用例，从而可以初步验证本次实验实现逻辑的正确性：



```
9 Router should receive ARP reply for 10.10.123.123 on router-eth1.
10 Router should send ICMP time exceeded error back to 10.10.123.123 on router-eth1.
11 A packet to be forwarded to 1.2.3.4 should arrive on router-eth1. The destination address 1.2.3.4 should not match any entry in the forwarding table.
12 Router should send an ICMP destination network unreachable error back to 10.10.123.123 out router-eth1.
13 A UDP packet addressed to the router's IP address 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!
```

### (可选)编写自己的测试用例，并且展示测试用例的设计逻辑：

共编写了五个测试用例，对本实验所实现的功能进行了较为全面的测试：

TestCase01：主要测试路由器对 ICMP echo request 的回复功能是否正确，路由器接收到一个 ICMP echo request 而后通过一对 ARP Request/ Reply 解析出下一跳MAC地址，之后从对应端口发出 ICMP echo reply，在该测试中测试了 ARP Request 和 ICMP echo reply 的内容是否符合预期；

TestCase02-05：测试路由器在四种情况下生成 ICMP error message 并且发出的功能。

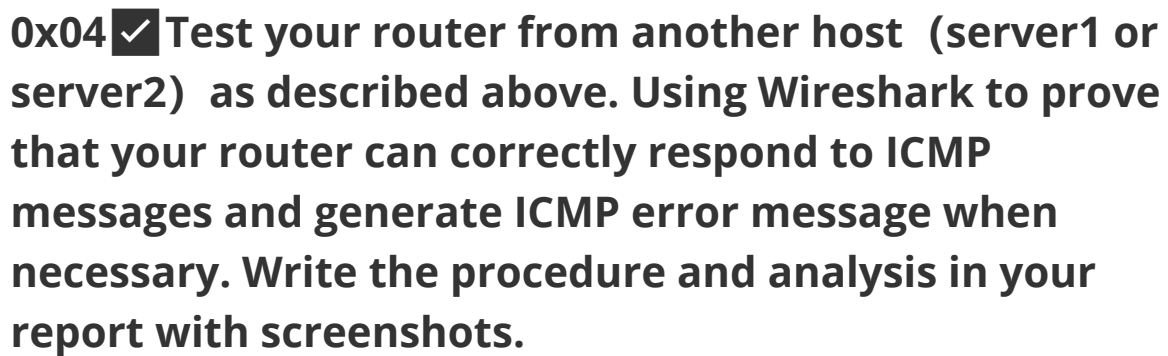
TestCase02：路由器接收到一个 ICMP 报文，但是无法通过路由表找到对应表项，需要发送 ICMP error message 到源IP指向的主机；

TestCase03：路由器接收到一个 UDP 报文，该报文不是 ICMP echo request 并且其目的IP指向路由器的一个端口，需要发送 ICMP error message 到源IP指向的主机；

TestCase04：路由器接收到一个 ICMP 报文，并且能够在路由表中找到对应表项（针对 FAQ02，如果一个包同时无法在路由表中找到表项，并且TTL减为0，那么 ICMP error message 应当为 DestinationUnreachable），且TTL自减后为0，需要发送 ICMP error message 到源IP指向的主机；

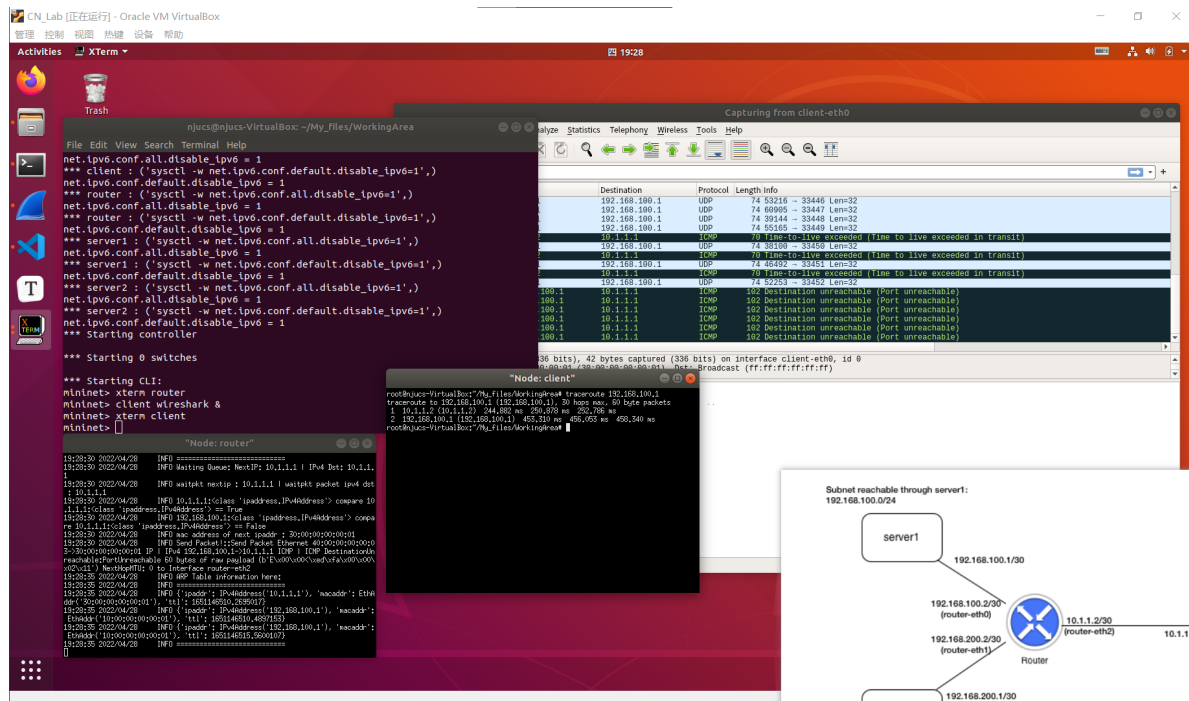
TestCase05：路由器接收到一个 ICMP 报文，并且其能够在路由表中找到对应的表项，但是通过5次 ARP 解析后都无法收到 ARP Reply，此时需要发送 ICMP error message 到源IP指向的主机；

执行自实现的测试，可以通过所有的测试用例，因而可以进一步验证实现逻辑的正确性：



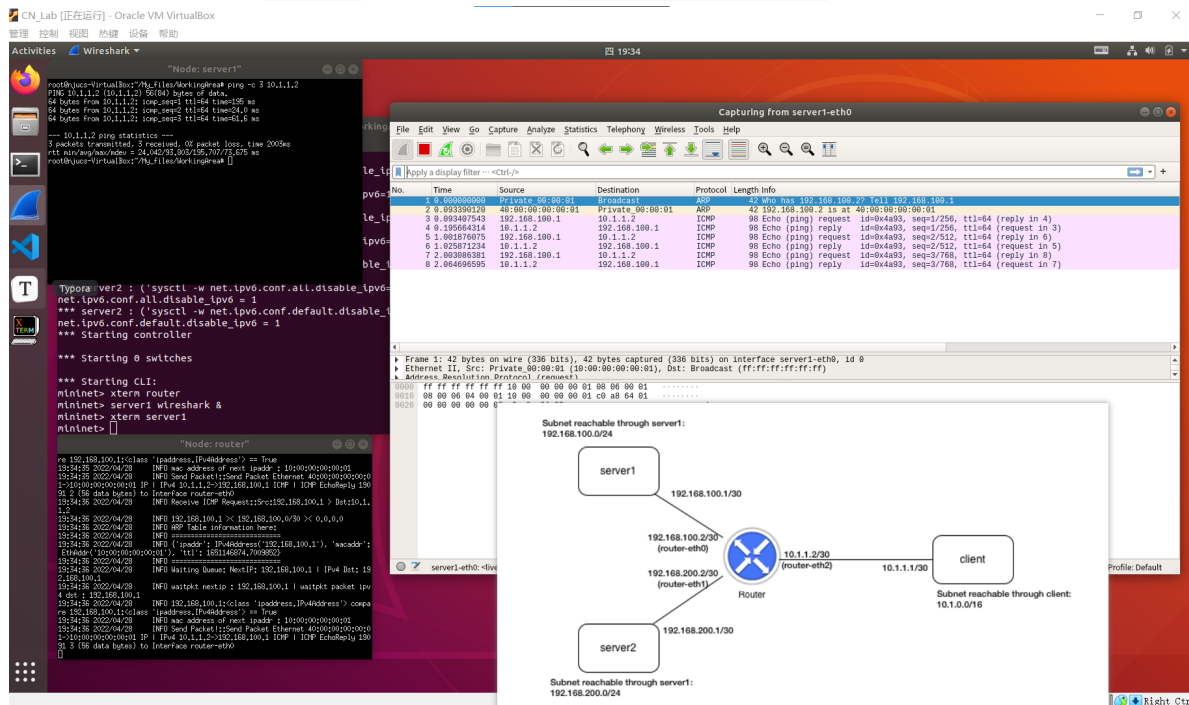


在 Mininet 当中以 client 运行实验说明所展示的四种操作，可以发现所展示的结果和实验说明中的一模一样，因而可以验证逻辑实现在模拟环境下的正确性：(./report/ 下可查看到所抓到的数据包)



而后以 server1 作为主机来在其上进行一些 ping 操作和 traceroute 操作，进一步验证模拟环境下逻辑的正确性，并且对过程进行一些分析：

①由 server1 直接 ping 到 router-eth2，一共进行3次。可以看到一共有3对 ICMP Echo Request/Reply，并且源IP地址的目的IP地址都正确，说明路由器对 ICMP Echo Request 进行了正确的回应，说明 Task01 中实现路由器对 ICMP Echo Request 的响应实现的正确：



②由 server1 直接 ping 到 client，一共1次，并且设置初始TTL为1。可以看到此时路由器发回一个 Time exceeded 的 ICMP error message，这是因为 server1 ping client 所发送的包 TTL为1，并且路由器能够在转发表中找到与目的IP对应的表项，而后TTL自减至0，此时就触发了 ICMP error，路由器此时便生成一个 ICMP error message 发送给源主机即 server1。

③ server1 直接乱 ping 一个不在路由表中的地址，此时由于没有匹配项，路由器传回一个 DestinationUnreachable 的 ICMP error message:

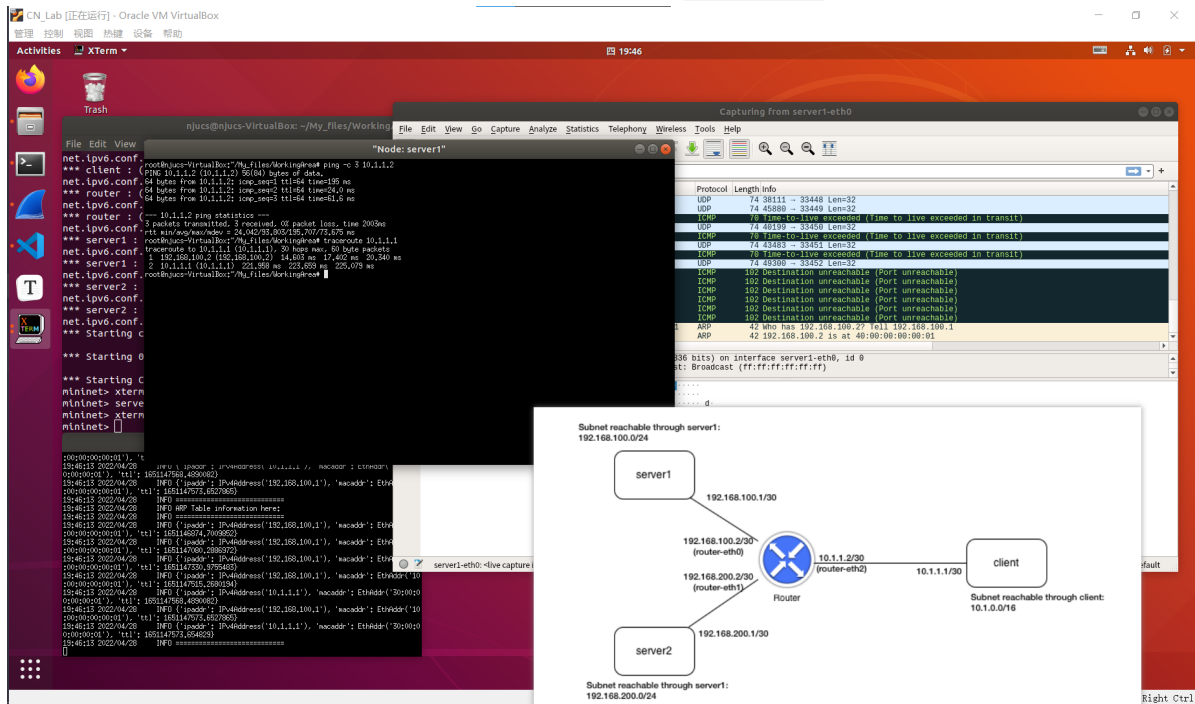
The screenshot displays a network simulation environment. On the left, a terminal window shows the configuration of a router (server1) and the execution of a ping command. The terminal output indicates that the ping command is successful, with 3 packets transmitted and 3 received. In the center, a Wireshark window shows a capture of the ping packets. The capture shows a sequence of ICMP Echo (ping) requests and replies. On the right, a network topology diagram illustrates the setup. The diagram shows a router (server1) connected to a client (10.1.1.30) via a router (10.1.1.2/30). The router is also connected to a server2 (192.168.200.2/30) via a router (192.168.200.2/30). The client is connected to a server1 (192.168.100.2/24) via a router (192.168.100.2/24). The diagram also shows a subnet reachable through the client (10.1.0.0/16) and a subnet reachable through the server1 (192.168.200.0/24).

③ server1 直接乱 ping 一个不在路由表中的地址，此时由于没有匹配项，路由器传回一个 DestinationUnreachable 的 ICMP error message:

The screenshot displays a network simulation environment. On the left, a terminal window shows the configuration of a router (server1) and the execution of a ping command. The terminal output indicates that the ping command is successful, with 3 packets transmitted and 3 received. In the center, a Wireshark window shows a capture of the ping packets. The capture shows a sequence of ICMP Echo (ping) requests and replies. On the right, a network topology diagram illustrates the setup. The diagram shows a router (server1) connected to a client (10.1.1.30) via a router (10.1.1.2/30). The router is also connected to a server2 (192.168.200.2/30) via a router (192.168.200.2/30). The client is connected to a server1 (192.168.100.2/24) via a router (192.168.100.2/24). The diagram also shows a subnet reachable through the client (10.1.0.0/16) and a subnet reachable through the server1 (192.168.200.0/24).



④ server1 直接对 client 进行 traceroute。在 server1 终端上显示的结果与实例结果对应，由此可以进一步验证在模拟环境下实现的正确性。（在 ./report/ 下可查看所抓到的数据包）：



## 5.核心代码

```
1  ### Handle ICMP Packet###
2  def handle_icmp_request_packet(self, icmp_header, ifaceName,
    packet):
3      log_info(f'Receive ICMP Request::Src:{packet[IPv4].src} > Dst:
        {packet[IPv4].dst}')
4      # Build ICMP echo Reply packet
5      echo_reply = ICMP()
6      echo_reply.icmptype = ICMPType.EchoReply
7      echo_reply.icmpdata.sequence = icmp_header.icmpdata.sequence
8      echo_reply.icmpdata.identifier =
        icmp_header.icmpdata.identifier
9      echo_reply.icmpdata.data = icmp_header.icmpdata.data
10     reply_ip_header = IPv4()
11     reply_ip_header.dst = packet[IPv4].src
12     reply_ip_header.src = packet[IPv4].dst
13     reply_ip_header.ttl = 64
14     ethernet_header = Ethernet()
15     ethernet_header.ethertype = EtherType.IPv4
16     echo_reply_pkt = ethernet_header + reply_ip_header + echo_reply
17     # Lookup forwarding table and add to waiting queue
18     max_matched_prefixlen = 0
19     for it in self.forwarding_table:
20         if echo_reply_pkt[IPv4].dst in it['ipv4network'] and
            it['ipv4network'].prefixlen > max_matched_prefixlen:
21             tmp = it['ipv4network']
22             tmp1 = it['nextip']
23             log_info(f'{echo_reply_pkt[IPv4].dst} >> {tmp} >>
                {tmp1}')
24             max_matched_prefixlen = it['ipv4network'].prefixlen
```

```

25         target_item = it
26         if max_matched_prefixlen == 0:
27             return None
28         for it in self.interfaces:
29             if it.name == target_item['interface']:
30                 target_interface = it
31                 waitpkt = waitPacket(target_item['nextip'], target_interface,
echo_reply_pkt, ifaceName)
32                 self.waiting_queue.append(waitpkt)
33
34     ### ICMP Error ###
35     def icmp_error(self, ori_pkt, err_type, err_code, ifaceName):
36         # Find interface which receive packet causes the ICMP error
37         for it in self.interfaces:
38             if it.name == ifaceName:
39                 interface = it
40                 break
41         # Build IP Header
42         ip_header = IPv4()
43         ip_header.dst = ori_pkt[IPv4].src
44         ip_header.src = interface.ipaddr
45         ip_header.protocol = IPPROTO.ICMP
46         ip_header.ttl = 64
47         ip_header.ipid = 0
48         # Build ICMP error message
49         # Include original packet information(28 bytes)
50         oridata = deepcopy(ori_pkt)
51         i = oridata.get_header_index(Ethernet)
52         del oridata[i]
53         icmp = ICMP()
54         icmp.icmptype = err_type
55         icmp.icmpcode = err_code
56         # Mark the length of original packet
57         icmp.icmpdata.origdgramlen = len(ori_pkt)
58         icmp.icmpdata.data = oridata.to_bytes()[:28]
59         # Build Packet
60         icmp_error_pkt = Ethernet() + ip_header + icmp
61         # Lookup forwarding table and add to waiting queue
62         max_matched_prefixlen = 0
63         for it in self.forwarding_table:
64             if icmp_error_pkt[IPv4].dst in it['ipv4network'] and
it['ipv4network'].prefixlen > max_matched_prefixlen:
65                 max_matched_prefixlen = it['ipv4network'].prefixlen
66                 target_item = it
67         if max_matched_prefixlen == 0:
68             return None
69         for it in self.interfaces:
70             if it.name == target_item['interface']:
71                 target_interface = it
72                 waitpkt = waitPacket(target_item['nextip'], target_interface,
icmp_error_pkt, ifaceName)
73                 tmp = target_item['nextip']

```

```
74 log_info(f'wow!{tmp}')
75 self.waiting_queue.append(waitpkt)
```

## 6.总结与感想

### 总结：

- 🧐清晰了解到了四种 ICMP error message 生成的典型状况；
- 🧐通过实践掌握了 ping 命令大致是怎样的一个工作原理，互联网控制协议的大致运行流程，相比以前黑盒一样更能够明白其中的细节，知其然并且知其所以然；
- 🧐阅读API文档以及适当的对代码进行封装、重构，都是在项目规模变得越来越大时掌握项目复杂度以及保证正确性的重要方法；
- 🧐利用 switchyard 框架写测试更加熟练了，使用 Mininet 进行仿真测试也是如此；

### 感想：

- 🧐自从上次实验开窍后，再说一句 switchyard 的错误信息非常有用，通过报错信息可以很快地找到出错点，并且把握其行为流程，实验速度加快了很多；
- 🧐阅读实验说明时分层级将需求和细节要求有层次地、清晰地提炼出来，这样做实验时对于所有信息都能够更加得心应手；
- 🧐下次实验时应当将写代码和写报告进行分离，等到最后所有代码确定下来后再开始报告的撰写，此前都是边写报告边做实验，一旦出现了一些需要修改的地方就会比较麻烦；