实验报告

一、实验名称

Lab 5: Respond to ICMP

二、实验目的

完成路由器的以下功能：

1. 响应 ICMP 消息，如回显请求（“pings”）。
2. 必要时生成 ICMP 错误消息，例如当 IP数据包的 TTL（生存时间）值已减少到零。

三、实验内容

响应 ICMP 回显请求

如果收到一个发给自己的icmp EchoRequest，那么使用make_icmp_reply函数创建一个icmp EchoReply包

```
if dst_ip in self.ip_list:
        log_info("there is a packet for me")
        if icmp:
            if icmp.icmptype == ICMPType.EchoRequest:
                log_info("Received ICMP Echo Request")
                reply = self.make_icmp_reply(packet)
                self.forwarding(reply, ifaceName)
                return
```

分别创建Ethernet头，ipv4头和icmp头，并合并成为新的包

把请求的序列号、标识符和数据段复制到icmp回复中，

并构造ipv4头，使得目标IP地址为为传入 ICMP 回显请求的源地址，IP 源地址为路由器的接口地址

```
def make_icmp_reply(self, request_packet):
    ipv4 = request_packet.get_header(IPv4)
    icmp = request_packet.get_header(ICMP)
    icmp_header = ICMP()
    icmp_header.icmptype = ICMPType.EchoReply
    icmp_header.icmpdata.sequence = icmp.icmpdata.sequence
    icmp_header.icmpdata.identifier = icmp.icmpdata.identifier
    icmp_header.icmpdata.data = icmp.icmpdata.data
    ipv4_header = IPv4()
    ipv4_header.src = ipv4.dst
    ipv4_header.dst = ipv4.src
    ipv4_header.protocol = IPProtocol.ICMP
    ipv4_header.ttl = 64
    eth_header = Ethernet()
    reply_packet = eth_header + ipv4_header + icmp_header
    return reply_packet
```

生成 ICMP 错误消息

#####

1. **没有匹配的条目：尝试将 IP 数据包的目标地址与转发表中的条目进行匹配时，找不到匹配的条目**

   将 ICMP 目标网络无法访问错误发送回 IP 数据包中源地址引用的主机。ICMP类型为“目标不可访问”，ICMP代码为“网络不可访问”。

   ```
   if entry is None:
       log_info("Sending ICMP Destination Unreachable")
       self.send_icmp_error(packet, fromintf, ICMPType.DestinationUnreachable,
   0)
       return
   ```

2. **TTL 已过期：在转发过程中递减 IP 数据包的 TTL 值后，TTL 变为 0**

   将 ICMP 超出时间错误消息发送回 IP 数据包中源地址引用的主机。ICMP类型为超时，ICMP代码为TTL过期。

   ```
   elif ipv4.ttl <= 1:
       log_info("Sending ICMP Time Exceeded")
       self.send_icmp_error(packet, fromintf, ICMPType.TimeExceeded, 0)
       return
   ```

3. **ARP故障：在 ARP 请求重传 5 次后，路由器没有收到 ARP 应答**

   将无法访问的 ICMP 目标主机发送回 IP 数据包中源地址所引用的主机。ICMP类型为“目标不可访问”，ICMP代码为“主机不可访问”。

   ```
   if retries >= 5:
       entry = self.get_forwarding_entry(ip)
       for packet, fromintf in self.waiting_packet[ip]:
           self.send_icmp_error(packet, fromintf,
   ICMPType.DestinationUnreachable, 1, True)
       del self.waiting_ip[ip]
       del self.waiting_packet[ip]
   ```

4. **不支持的功能：传入的数据包发往分配给路由器接口之一的 IP 地址，但该数据包不是 ICMP 回显请求**

```
if dst_ip in self.ip_list:
    log_info("there is a packet for me")
    if icmp:
        ...
    else:
        log_info("Sending ICMP Destination Unreachable")
        errpacket = self.make_icmp_error(packet,
ICMPType.DestinationUnreachable,
self.net.interface_by_name(ifaceName).ipaddr, ipv4.src, icmpcode=3)
        self.forwarding(errpacket, None)
        return
```

发送icmp错误消息

路由器不应生成ICMP错误消息来响应任何ICMP错误消息，即使它们符合错误条件。

生成icmp错误包并发送出去

```
icmp_error_types = {ICMPType.DestinationUnreachable, ICMPType.SourceQuench,
                ICMPType.Redirect, ICMPType.TimeExceeded,
ICMPType.ParameterProblem}

def send_icmp_error(self, packet, fromintf, icmptype, icmpcode, intfchoice =
False):
    icmp = packet.get_header(ICMP)
    ipv4 = packet.get_header(IPv4)
    if icmp and icmp.icmptype in icmp_error_types:
        log_info("icmp.icmptype: {}, return".format(icmp.icmptype))
        return
    entry = self.get_forwarding_entry(ipv4.src)
    if entry is None: return
    if not intfchoice:
        errpacket = self.make_icmp_error(packet, icmptype,
            self.net.interface_by_name(entry.interface).ipaddr, ipv4.src,
icmpcode)
    else:
        errpacket = self.make_icmp_error(packet, icmptype,
            self.net.interface_by_name(fromintf).ipaddr, ipv4.src, icmpcode)
    self.send_out(ipv4.src, entry.interface, errpacket, fromintf)
```

ICMP错误消息（如目标不可达、源抑制、重定向、生存时间过期、参数问题等）是为了通知发送端有关传输问题的。然而，如果路由器对这些ICMP错误消息也生成并发送ICMP错误消息，可能会导致以下问题：

1. **错误消息风暴**：如果一个ICMP错误消息引发另一个ICMP错误消息，可能会形成一个无限循环，导致网络上充斥大量的ICMP错误消息，这被称为“错误消息风暴”。这不仅会浪费网络带宽，还可能导致网络设备过载。
2. **安全问题**：对所有接收到的ICMP错误消息都回应，可能会被恶意用户利用，进行如拒绝服务攻击（DoS）等恶意行为。

构造icmp错误包

先删除原始数据包的以太网包头，

然后分别创建Ethernet头，ipv4头和icmp头，合并成为新的包，

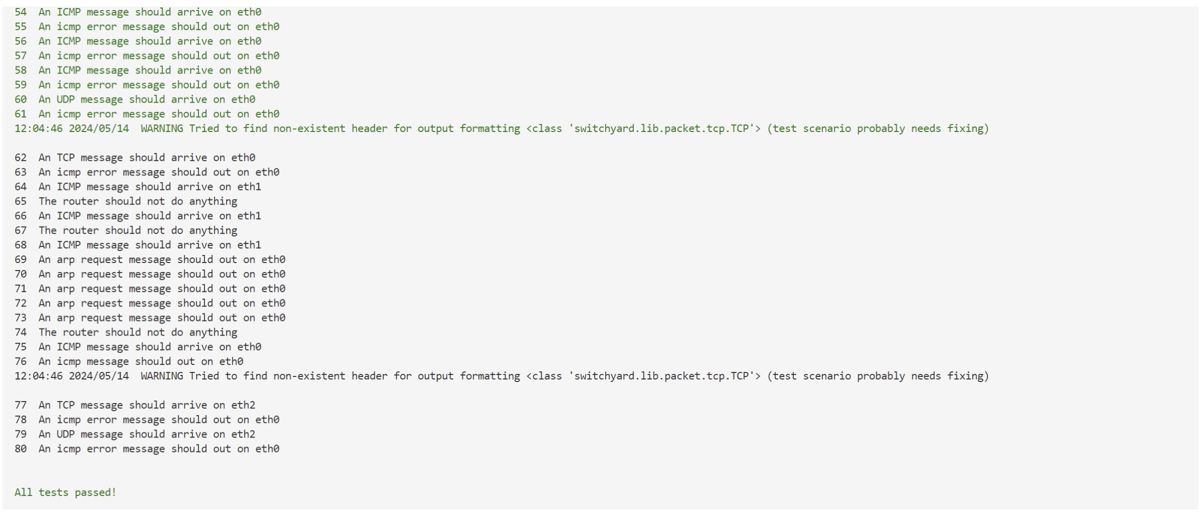
给icmp填入类型，编码，和数据（包含原始数据包的前28个字节）

给ipv4填好源IP和目标IP，并设置ttl为64

```python
def make_icmp_error(self, origpkt, icmptype, src_ip, dst_ip, icmpcode = 0):
    log_info("Making ICMP error")
    del origpkt[Ethernet]
    eth = Ethernet()
    icmp = ICMP()
    icmp.icmptype = icmptype
    icmp.icmpcode = icmpcode
    icmp.icmpdata.data = origpkt.to_bytes()[:28]
    ipv4 = IPv4()
    ipv4.protocol = IPProtocol.ICMP
    ipv4.src = src_ip
    ipv4.dst = dst_ip
    ipv4.ttl = 64
    pkt = eth + ipv4 + icmp
    return pkt
```

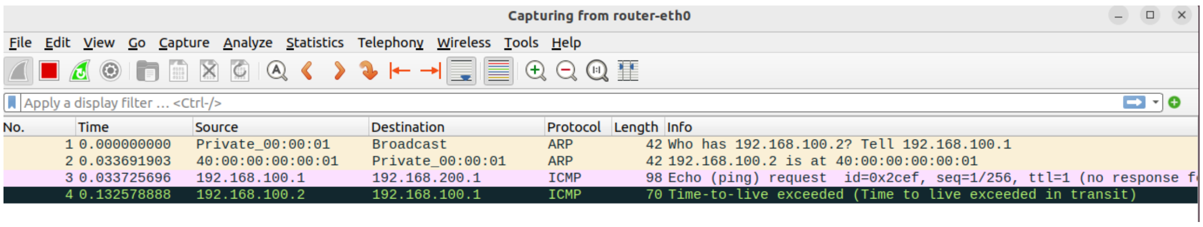
四、实验结果

成功通过测试样例



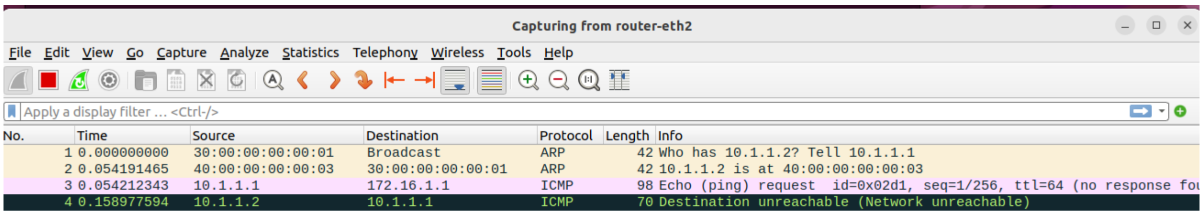
```
54  An ICMP message should arrive on eth0
55  An icmp error message should out on eth0
56  An ICMP message should arrive on eth0
57  An icmp error message should out on eth0
58  An ICMP message should arrive on eth0
59  An icmp error message should out on eth0
60  An UDP message should arrive on eth0
61  An icmp error message should out on eth0
12:04:46 2024/05/14  WARNING Tried to find non-existent header for output formatting <class 'switchyard.lib.packet.tcp.TCP'> (test scenario probably needs fixing)

62  An TCP message should arrive on eth0
63  An icmp error message should out on eth0
64  An ICMP message should arrive on eth1
65  The router should not do anything
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
12:04:46 2024/05/14  WARNING Tried to find non-existent header for output formatting <class 'switchyard.lib.packet.tcp.TCP'> (test scenario probably needs fixing)

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!
```

server1 ping -c 1 -t 1 server2


Capturing from router-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

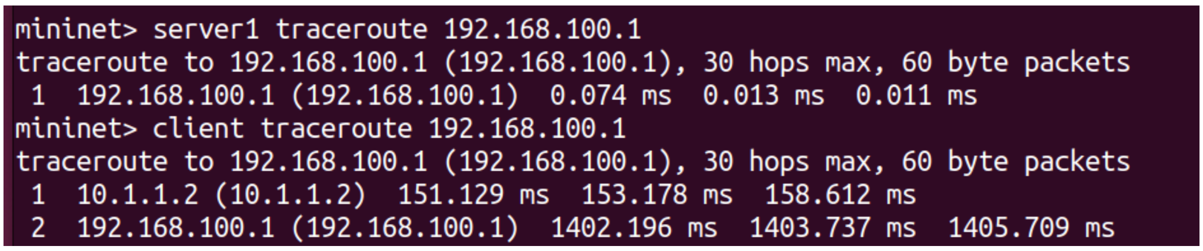
```
No.   Time         Source              Destination         Protocol Length Info
    1 0.000000000  Private_00:00:01    Broadcast           ARP        42 Who has 192.168.100.2? Tell 192.168.100.1
    2 0.033691903  40:00:00:00:00:01   Private_00:00:01    ARP        42 192.168.100.2 is at 40:00:00:00:00:01
    3 0.033725696  192.168.100.1       192.168.200.1       ICMP       98 Echo (ping) request  id=0x2cef, seq=1/256, ttl=1 (no response f
    4 0.132578888  192.168.100.2       192.168.100.1       ICMP       70 Time-to-live exceeded (Time to live exceeded in transit)
```

client ping -c 1 172.16.1.1


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

```
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 0.054191465  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 0.054212343  10.1.1.1            172.16.1.1          ICMP       98 Echo (ping) request  id=0x02d1, seq=1/256, ttl=64 (no response fou
    4 0.158977594  10.1.1.2            10.1.1.1            ICMP       70 Destination unreachable (Network unreachable)
```

traceroute



```
mininet> server1 traceroute 192.168.100.1
traceroute to 192.168.100.1 (192.168.100.1), 30 hops max, 60 byte packets
 1  192.168.100.1 (192.168.100.1)  0.074 ms  0.013 ms  0.011 ms
mininet> client traceroute 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 (10.1.1.2)  151.129 ms  153.178 ms  158.612 ms
 2  192.168.100.1 (192.168.100.1)  1402.196 ms  1403.737 ms  1405.709 ms
```