

Computer Network* Lab 5

- 徐思源 191220133
- Department of Computer Science and Technology
- Nanjing University
- 1357307497@qq.com

实验目的

分成两个部分，主要是实现对ICMPRequest的回复和对几种出错情况的处理，完善我们的路由器功能

Step1 核心代码和实现

- 对于回复ICMPRequest的情况，首先检查包是否是发给自己的，对比包的地址和路由器接口即可，如果是，我们向收到的接口发送一个回复的包，否则，执行转发命令（lab3中已经实现）

◦ 框架如下：

```
■ if ipv4 is not None:                # recv a ipv4 packet
    icmp=None
    flag=0
    for i in self.net.interfaces():    # whether the ICMP
is for me
        if i.ipaddr==ipv4.dst:
            flag=1
    if flag==1:
        if icmp is not None:
            if icmp.icmptype is ICMPType.EchoRequest: #
request
                # need to reply
                echoReply=
self.create_ICMP_EchoReply_packet(packet) # create reply packet
                self.deal_send(echoReply,64,ifaceName)
            else:

                self.destination_port_unreachable(packet,ifaceName)
            else:

                self.destination_port_unreachable(packet,ifaceName)
            else:
                # do forward
```

- 分析总体的router功能，重新对结构进行分割，因为lab3里面对模块的划分不是很好，在lab4中再次做出调整，根据forwardingtable和cache发送包的过程封装为函数deal_send(packet,ttl,ifaceName)，所以现在需要解决的问题就是如何构造一个ICMPReply包

- 根据文件中的指导，新建一个包，进行复制和相关的处理，组装成一个新的包，然后直接发送这个包就可以

```

■      # create the icmp header
      src_icmp_header = request_packet.get_header(ICMP)
      icmp = ICMP()
      icmp.icmptype = ICMPType.EchoReply
      icmp.icmpdata.sequence = src_icmp_header.icmpdata.sequence
      icmp.icmpdata.identifier =
src_icmp_header.icmpdata.identifier
      icmp.icmpdata.data = src_icmp_header.icmpdata.data

      #src_icmp_header.icmptype=ICMPType.EchoReply

      #print(icmp)

      # create ipve header
      src_ipv4_header = request_packet.get_header(IPv4)
      temp=src_ipv4_header.src
      src_ipv4_header.src=src_ipv4_header.dst
      src_ipv4_header.dst=temp
      src_ipv4_header.ttl = 64

      eth = Ethernet()
      eth.ethertype = EtherType.IP
      pkt= eth+src_ipv4_header+icmp

```

• 错误以及解决方案

- 一开始考虑到ICMPheader中除了type之外的内容都没有改变，考虑直接改变type然后转发，但是会发现不能通过测试，输出包的结果，可以看到我们构造的ICMPReply包是不符合要求的，如下

```

■ ICMP EchoReply 0 0 (0 data bytes)

```

- 查阅手册，我们会发现，手册的这一段说明了当我们改变包的type时，其他部分会随之改变，变成默认的初始值，而不是保留原来的数据：

```

■ >>> i = ICMP()
>>> print(i)
ICMP EchoRequest 0 0 (0 data bytes)
>>> i.icmptype = ICMPType.TimeExceeded
>>> print(i)
ICMP TimeExceeded:TTLExpired 0 bytes of raw payload (b'') 0x0gramLen: 0
>>> i.icmpcode
<ICMPCodeTimeExceeded.TTLExpired: 0>
>>> i.icmpdata
<switchyard.lib.packet.icmp.ICMPTimeExceeded object at 0x10d3a3308>

```

Notice above that when the icmptype changes, other contents in the ICMP header object change appropriately.

- 改成复制的形式再输出，可以发现正确的包的数据如下：

```

■ ICMP EchoReply 0 42 (0 data bytes)

```

Step2 核心代码和实现

- 用create_error_packet函数来构造处理错误的包，用四个小函数来分别处理文件中所说的四种错误：

```
def destination_network_unreachable(self, packet, ifaceName):
def destination_host_unreachable(self, q_packet, ifaceName):
def time_exceeded(self, packet, ifaceName): ...
def create_error_packet(self, packet, kind, code, interface):
def destination_port_unreachable(self, packet, ifaceName):
```

- create_error_packet函数的实现，主要就是根据指导文件中的要求设置ICMPType和code等部分，传入kind和code参数并处理即可

```
# delete eth
i = packet.get_header_index(Ethernet)
del packet[i]

# create icmp header
icmp = ICMP()
icmp.icmptype = kind
if code >= 0:
    icmp.icmpcode = code
icmp.icmpdata.data = packet.to_bytes()[28:]
icmp.icmpdata.origdgramlen = len(packet)

# create ipv4 header
ipv4 = IPv4()
ipv4.protocol = IPProtocol.ICMP
ipv4.ttl = 64
src_ipv4 = packet.get_header(IPv4)
ipv4.dst = src_ipv4.src
ipv4.src = interface.ipaddr

# create eth header
eth = Ethernet()
eth.src = interface.ethaddr
eth.ethertype = EtherType.IP

pkt = eth + ipv4 + icmp
```

- 接下来只需要针对几种情况处理就可以了，我们以destination_network_unreachable为例

```
def destination_network_unreachable(self, packet, ifaceName):
    #if kind == "destination network unreachable":
    interface = self.net.interface_by_name(ifaceName)

    pkt = self.create_error_packet(packet, ICMPType.DestinationUnreachable, 0, interface)

    ipv4 = pkt.get_header(IPv4)
    target = self.ip_mac_table.get(ipv4.dst)
```

```

        if target is None:
            arp_request =
create_ip_arp_request(interface.ethaddr, interface.ipaddr, ipv4.dst)

            self.net.send_packet(ifaceName, arp_request)
            #pkt[0].src=interface.ethaddr

self.q.append(q_item(arp_request, pkt, ipv4.dst, ifaceName, ifaceName))
        else:
            pkt[0].dst = target
            self.net.send_packet(ifaceName, pkt)

```

- 其他几种情况的处理大同小异
 - destination_host_unreachable由于是从队列中提取出来的，需要考虑从队列元素中提取出包
 - destination_port_unreachable由于时直接把包送还回去，所以我们不需要重新获取目的的mac，直接从原来的包里获取就可以
 - time_exceeded改变type和code就可以
- 最后再对应的位置调用对应的出错处理函数即可

结果和分析

- 测试样例通过

```

njucs@njucs-VirtualBox: ~/Desktop/lab-5-X-March
File Edit View Search Terminal Help
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!

(syenv) njucs@njucs-VirtualBox:~/Desktop/lab-5-X-March$

```

- mininet运行和抓包
 - 首先构造一个ttl为1的包，这样就会触发time exceeded的错误，导致不能成功的ping，client会收到一个time exceeded错误提示的回复包
 - 抓包结果

Capturing from client-eth0					
No.	Source	Time	Destination	Protocol	Length Info
1	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
2	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
3	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
4	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)

- destination unreachable 的情况，client 同样会收到一个显示 destination unreachable 错误情况的包

抓包结果

Capturing from client-eth0					
No.	Source	Time	Destination	Protocol	Length Info
1	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
2	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
3	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
4	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)

- traceroute 结果如下：

```

"Node: client"
root@njucs-VirtualBox:~/Desktop/lab-5-X-March# traceroute -n 192.168.100.1
traceroute: invalid option -- 'n'
Try 'traceroute --help' or 'traceroute --usage' for more information.
root@njucs-VirtualBox:~/Desktop/lab-5-X-March# traceroute 192.168.100.1
traceroute to 192.168.100.1 (192.168.100.1), 64 hops max
 1  10.1.1.2  140.618ms 105.108ms 102.899ms
 2  192.168.100.1 270.988ms 104.589ms 103.579ms
root@njucs-VirtualBox:~/Desktop/lab-5-X-March#

```

抓包结果如下：

Capturing from client-eth0					
No.	Source	Time	Destination	Protocol	Length Info
1	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
2	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
3	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
4	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
5	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
6	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
7	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
8	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
9	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
10	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
11	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
12	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
13	10.1.1.1	0.0000000	192.168.100.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)
14	192.168.100.1	0.0000000	10.1.1.1	ICMP	60 Echo (ping) request 140.1.1.1 (11154)