

计网实验五

1.实验名称

Respond to ICMP

2.实验目的

- 1.响应 icmp 的 Echo_Request
- 2.处理 ICMP 的错误信息

3.实验内容

TASK TWO.响应 ICMP echo_request 的逻辑：

首先判断他是 ipv4 类型的包，然后在所有的接口中遍历，查找是否有接口 ip 地址与该数据包的目标 IP 地址匹配，如果匹配查看他的类型是否为 ICMP 的请求包，如果是，则构建 ICMP 回复包，该 reply 包的以太网头的 src 和 dst 分别为请求包的以太网头的 dst 和 src，reply 包的 IPv4 头中的 src 和 dst 恰好为 request 包的 dst 和 src，然后构建 ICMP 头，其 icmp type 为 ICMPType.EchoReply 即为 0，而 code 只需要复制过来即刻，其中将请求中的序列号复制到您所做的回复中。将请求中的标识符复制到回复中，将回复中的数据字段设置为与请求中的数据相同。然后正常走流程发送即可。

```
#-----
index3=packet.get_header_index(ICMP)
if packet[1].protocol==IPProtocol.ICMP and packet[index3].icmpcode==8 and packet[index3].icmpcode==0:
    reply_icmp=ICMP()
    reply_icmp.icmpcode=packet[index3].icmpcode
    reply_icmp.icmpcode=0
    reply_icmp.icmpdata.data=packet[index3].icmpdata.data
    reply_icmp.icmpdata.identifier=packet[index3].icmpdata.identifier
    reply_icmp.icmpdata.sequence=packet[index3].icmpdata.sequence #响应icmp回显请求
    packet[2]=reply_icmp
    packet[0].ethertype=EtherType.IPv4
    packet[0].dst=packet[0].src
    packet[0].src=interface.ethaddr
    packet[1].dst=packet[1].src
    packet[1].src=interface.ipaddr
    packet[1].ttl = 64

    break
```

TASK THREE

1. 路由器不应该生成 ICMP 错误消息来响应任何 ICMP 错误消息的原因：

如果路由器能够生成 ICMP 错误来响应 ICMP 错误信息，则会有人通过主动构建一定不正确的包来使得路由器工作量加大，使得路由器无法正常的运作

2.生成 ICMP 错误信息的逻辑：在每次出现错误信息都要判断该包是否为 ICMP

错误包

首先构造函数 `error——icmp` 来构建 icmp 错误包，其构建逻辑如下表所示，其中 `ifaceName` 只是单纯为了补全信息，会在之后的数据包处理中获取到正确的 `eth.Src` 和 `interface.SRC`，其中以太网的 `dst` 和 IPv4 头的 `dst` 分别为所发来的包的对应的 `src`，然后通过传入的参数来构建相应的 ICMP 头，最后就可以得到相应的 ICMP 错误包了

```
def error_reply(self,origin_pkt,ittype,icode,ifaceName):
    port1=self.net.interface_by_name(ifaceName)
    etherheader = Ethernet(
        src = self.net.interface_by_name(ifaceName).ethaddr,
        dst=origin_pkt[0].src,
        ethertype = EtherType.IPV4)
    ipv4header=IPv4( )
    ipv4header.dst=origin_pkt[1].src
    ipv4header.src=port1.ipaddr
    ipv4header.protocol=IPProtocol.ICMP
    ipv4header.ttl=64
    ICMPheader=ICMP()
    ICMPheader.icmptype=ittype
    ICMPheader.icmpcode=icode
    del origin_pkt[0]
    ICMPheader.icmpdata.data = origin_pkt.to_bytes()[28]
    final_pkt=etherheader+ipv4header+ICMPheader

    return final_pkt
```

1.不支持的功能，即目标端口不可到达：同样的相当于在判断是否为 `echorequest` 时同时操作，当其不是 ICMP 请求包时，同时要判断它不是 ICMP 错误包，因为如果 ICMP 错误包则不做处理，则为不支持的功能：则构建的新 ICMP 错误包的 `icmptype` 应为 `ICMPType.DestinationUnreachable`，`icmpcode` 查 ICMP 头可知为 3

```
181 #
182
183     else :
184         if packet[1].src==self.net.interface_by_name(ifaceName).ipaddr:
185             flag=0
186             return
187         icmp=packet.get_header(ICMP)
188         if icmp is not None and (icmp.icmptype==3 or icmp.icmptype==11 or icmp.icmptype==12):
189             return
190         packet=self.error_reply(packet,ICMPType.DestinationUnreachable,3,interface.name) #Q4
191         self.erroricmplist.append(packet)
192     break
193
194
```

2.无匹配条目：

首先我们通过转发表进行最大 ip 匹配，如果没有匹配到，则就表明其为 ICMP 无匹配条目，表现为目的 ip 无法到达，当然也需要先判断其是否为 ICMP 错误包，若不是，则构建 ICMP 错误包，并且传入相应的参数，然后对 ICMP 错误包再次进行一系列数据包处理，判断器是否无匹配条目或者 TTL 过期或者 ARP 失败，在我的实现中直接将数据包更新为对应的 ICMP 错误包再往下接着处理的

```

196 #-----#
197
198     maxlen = 0
199     for context in self.forward_table.keys():
200         if packet[1].dst in context:
201             if context.prefixlen > maxlen:                #在转发表中进行最大匹配
202                 maxlen = context.prefixlen
203                 text = self.forward_table[context]
204 #-----#
205
206     if maxlen==0:
207         icmp=packet.get_header(ICMP)
208         if icmp is not None and (icmp.icmptype==3 or icmp.icmptype==11 or icmp.icmptype==12):
209             return
210         packet=self.error_reply(packet,ICMPType.DestinationUnreachable,0,ifaceName)
211         self.erroricmplist.append(packet)
212         for context in self.forward_table.keys():
213             if packet[1].dst in context:
214                 if context.prefixlen > maxlen:                #在转发表中进行最大匹配
215                     maxlen = context.prefixlen
216                     text = self.forward_table[context]
217         if maxlen==0:
218             return
219         if maxlen !=0:

```

3.TTL 过期:

当最大 ip 匹配到对应条目后，判断当前 ip 的 ttl 是否为大于 1，因为如果 ttl 为 1，在处理时要减一，则直接 ttl 超时，所以直接一起判断，如果不是，则表明此次操作过后 ttl 为 0 过期，需要构建 ICMP 错误包，传入相应的 icmptype 和 icmpcode 错误包中的 ttl 肯定自己构建的，肯定是大于 1 的，所以只需要对其进行 ip 匹配判断其生成的 ICMP 超时包是否出错即可

```

if maxlen !=0:

    if packet[1].ttl <=1:
        icmp=packet.get_header(ICMP)
        if icmp is not None and (icmp.icmptype==3 or icmp.icmptype==11 or icmp.icmptype==12):
            return
        packet=self.error_reply(packet,ICMPType.TimeExceeded,0,ifaceName)
        self.erroricmplist.append(packet)
        len1=0
        for context in self.forward_table.keys():

            if packet[1].dst in context:
                if context.prefixlen > len1:                #在转发表中进行最大匹配
                    len1 = context.prefixlen
                    text = self.forward_table[context]
            if len1 ==0:
                return
        packet[1].ttl-=1

```

4.arp 失败:

和 lab4 相同，我们对 LAB 四中的代码稍加修改，只需要判断对目标 IP 超过五次请求的包是否为 ICMP 错误包即刻，如果判断到它的 ICMP 头不为空且其是错误包，则不做处理，在我的实现中这类超过五次的包，标记为 1，而需要处理的包，标记为 -1，然后标记为 -1 则进行 ICMP 错误包构建，然后再对其进行数据包处理，因为其不在 handle 函数中，所以我直接调用 handle_packet 函数处理

```

else:
    tmp1=0
    while tmp1 < len(self.sendlist):
        if self.sendlist[tmp1]==self.waitinglist[index][2]:
            del self.sendlist[tmp1]
            tmp1+=1
    tmp2=index+1
    while tmp2 < len(self.waitinglist):
        if self.waitinglist[tmp2][2]==self.waitinglist[index][2]:
            self.waitinglist[tmp2][5]=True
            tmp2+=1
    icmp=self.waitinglist[index][0].get_header(ICMP)
    if icmp is not None and (icmp.icmptype==3 or icmp.icmptype==11 or icmp.icmptype==12):
        return 1
    return -1
else:

```

```

def handle_list(self,index):
    if self.waitinglist[index][5]==1:
        icmp=self.waitinglist[index][0].get_header(ICMP)
        if icmp is not None and (icmp.icmptype==3 or icmp.icmptype==11 or icmp.icmptype==12):
            return 1
    return -1

```

```

56
57
58 def clean_list(self):#根据时间更新缓存表对于缓存队列的表进行处理
59     i=0
60     while i < len(self.waitinglist):
61         flag = self.handle_list(i)
62         if flag==1:#已经成功发送,需要在队列中删除
63             self.Todelete.append(i)
64         elif flag== -1:
65             self.Todelete.append(i)
66             arp_error=self.error_reply(self.waitinglist[i][0],ICMPType.DestinationUnreachable,1,self.waitinglist[i][1].name)
67             self.erroricmplist.append(arp_error)
68             self.handle_packet((0,self.waitinglist[i][1].name,arp_error))
69             i+=1
70     n = len(self.Todelete)#对于处理完的和超过五次访问请求得要删除
71
72     while n > 0:
73

```

3.测试样例运行结果

```

njucs@njucs-VirtualBox: ~/workspace/lab-5-kayasa12321
File Edit View Search Terminal Help
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
03:54:43 2023/11/23 WARNING Tried to find non-existent header for output format
ting <class 'switchyard.lib.packet.tcp.TCP'> (test scenario probably needs fixin
g)
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!

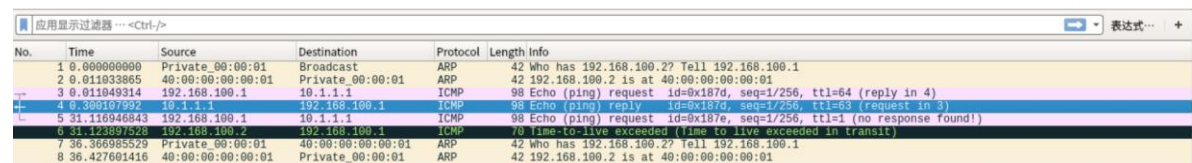
njucs@njucs-VirtualBox:~/workspace/lab-5-kayasa12321$

```

4.在 mininet 上执行并解释正确性：

A.对于 echo_request 的正确回应

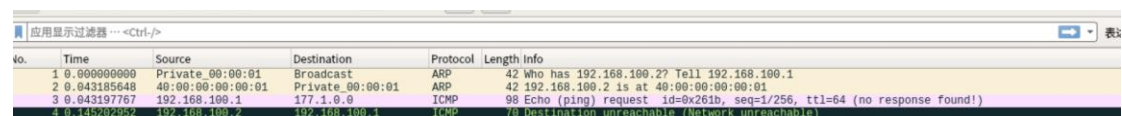
首先我们从 server1 向 client 10.1.1.1 发送一个 ICMP 的 echorequest，预期能够收到 echoreply，然后再发送一个 ttl 为一的 echorequest，预计收到一个 ttl 错误



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	0.011033865	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	0.011049314	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request id=0x187d, seq=1/256, ttl=64 (reply in 4)
4	0.300109392	10.1.1.1	192.168.100.1	ICMP	98	Echo (ping) reply id=0x187d, seq=1/256, ttl=63 (request in 3)
5	31.110946849	192.168.100.1	10.1.1.1	ICMP	98	Echo (ping) request id=0x187e, seq=1/256, ttl=1 (no response found!)
6	31.123897528	192.168.100.2	192.168.100.1	ICMP	78	Time-to-live exceeded (time to live exceeded in transit)
7	36.366985529	Private_00:00:01	40:00:00:00:00:01	ARP	42	Who has 192.168.100.2? Tell 192.168.100.1
8	36.427601416	40:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.2 is at 40:00:00:00:00:01

从结果上来看，我们在第一个包进行 echorequest 时，成功收到回复，第二个则是属于是超时，因为 ttl 减少为 0

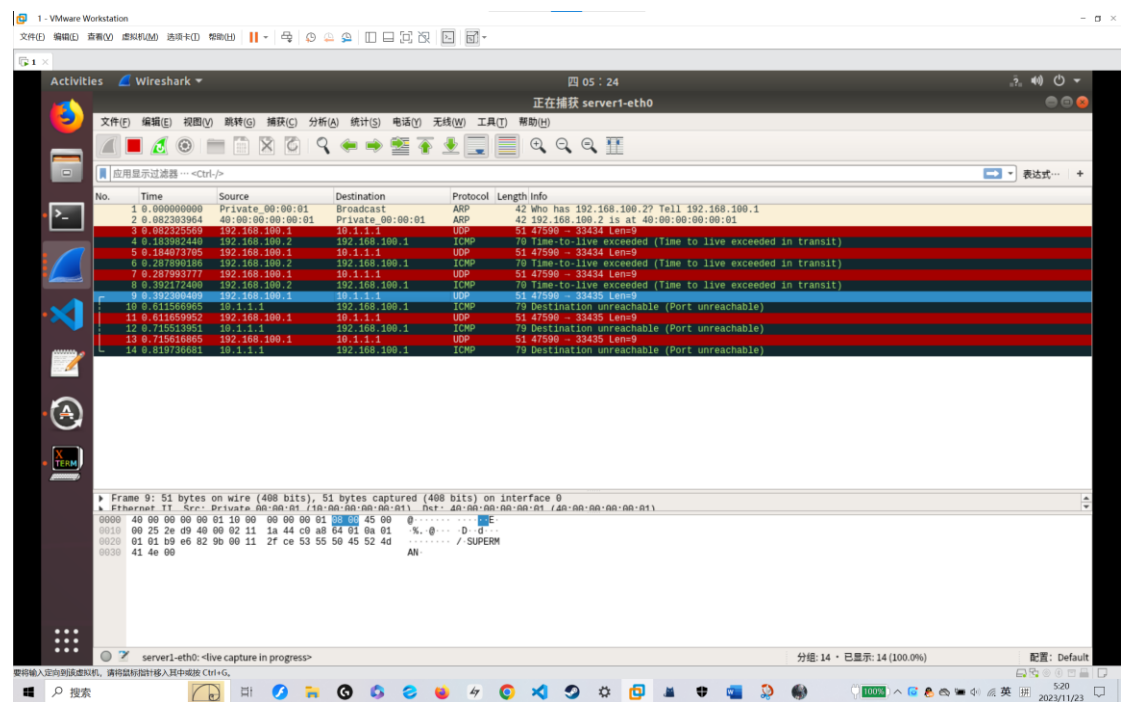
然后修改 start_mininet 增加一条 set_route(net,'server1','177.1.0.0','192.168.100.2') 然后启动 mininet，从 server1 向 177.1.0.0 ping 操作，应该收到接口不可到达



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	0.043185648	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	0.043197767	192.168.100.1	177.1.0.0	ICMP	98	Echo (ping) request id=0x261b, seq=1/256, ttl=64 (no response found!)
4	0.145202952	192.168.100.2	192.168.100.1	ICMP	78	Destination unreachable (Network unreachable)

B.TRACERoute

```
mininet> server1 traceroute 10.1.1.1
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
traceroute to 10.1.1.1 (10.1.1.1), 64 hops max
 1  192.168.100.2  184.020ms  103.874ms  104.253ms
 2  10.1.1.1  219.311ms  103.903ms  104.169ms
mininet>
```



在 arp 回复后，不断发送 ttl 为 1 的 udp 数据包，然后因为 ttl 为 1 则会发送 timeecced 错误，导致不断发送错误包，然后在 clien 发送了三个 ICMP 错误包