南京大学本科生实验报告

181860097 计算机科学与技术系 王明骁
Email：1164151483@qq.com
开始日期：2020年5月3日 结束日期：2020年5月4日
lab5

实验名称

Lab 5: Respond to ICMP

实验目的

- 为构建一个完整的路由器编写其最后的部分
- 掌握路由器回复icmp的原理和逻辑
- 掌握路由器针对几种网络中的典型错误进行发送错误信息的原理和逻辑

task1

1. **实验内容**

   task1为准备阶段，只需要按照实验手册上的说明，将相关文件copy到lab_5文件夹并做好命名即可

task2

1. 实验内容

   task2的内容为针对icmp request进行回复，回复的包的源地址为发送icmp request的目的地址，而目的地址为发送icmp request的源地址，需要构造icmp部分，ip包头和以太网头。其中，icmp部分的icmptype为echoreply，序列号，标识符，和data都是直接拷贝icmp request包的对应部分过来，最后将这个包按照lab4中的查找转发表等一系列逻辑发送即可

2. 实验结果

   等待task3完成之后一起测试

3. 核心代码

```
        if ipv4 is not None:
            #debugger()
            pkt[IPv4].ttl -= 1
            pretime = time.time()
            #debugger()
            flag = 1
            error = 0
            maxlength = 0
            for intf in intftable:
                if intf.ipaddr == ipv4.dst:      You, 17 hours ago • finish icmp reply
                    flag = 0
                    srcmac = intf.ethaddr
                    break
            if flag == 0:
                icmp = pkt.get_header(ICMP)
                if icmp is not None:
                    if icmp.icmptype == ICMPType.EchoRequest:
                        i = ICMP(icmptype = ICMPType.EchoReply, data = icmp.icmpdata.data,sequence = icmp.icmpdat
                        ipheader = IPv4(src= ipv4.dst,dst = ipv4.src,ttl = 32)
                        e = Ethernet(src = srcmac)
                        sendpkt = e + ipheader + i
                        pkt = sendpkt
                        ipv4 = ipheader
                        flag = 1
```

在收到一个ipv4的包时，首先判断他的目的ip是不是路由器的某个端口的ip，如果是，则检查其类型是不是icmp request，如果是icmp request，则按照1.实验内容中所述的内容构建一个icmp回复的包，构建好之后将其赋值给pkt，同时将ip包头赋值给ipv4，伪装成一个需要forwarding的包，这样可以完全复用lab4的代码来发送icmpreply而不用做修改

task3

1. 实验内容

   task要求针对四种典型的网络错误进行构建icmp错误信息并发送，几种错误类型和需要构建的icmp信息如下：

   - 转发表中没有匹配的项：

     icmptype = destination unreachable

     icmpcode = icmptypecodemap[icmp.icmptype].network unreachable
   - 数据包的ttl减到0：

     icmptype = time exceeded

     icmpcode = icmptypecodemap[icmp.icmptype].TTL expired
   - 发送5次arp之后仍未得到回复：

     icmptype = destination unreachable

     icmpcode = icmptypecodemap[icmp.icmptype].host unreachable
   - 目的ip是路由器的某个端口ip，但是数据包的类型不是icmprequest：

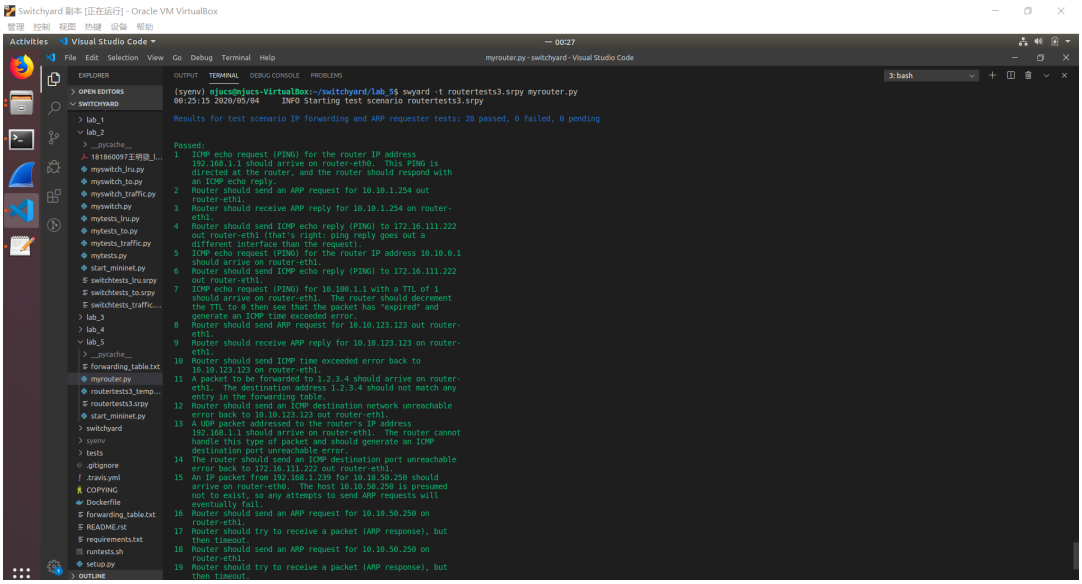
     icmptype = destination unreachable

     icmpcode = icmptypecodemap[icmp.icmptype].port unreachable

   而构建icmp其他部分的代码在实验手册上已经提供，值得一提的是，实验手册上的代码只构建了ip包头部分和icmp部分，而最后要发出去的icmp错误信息还需要以太网头，因此在构建时我加上了以太网头，以便后续代码填入源mac地址和目的mac地址，构建好包后的发送逻辑也是完全复用lab4

2. 实验结果

   运行提供的测试用例，得到的结果如下：

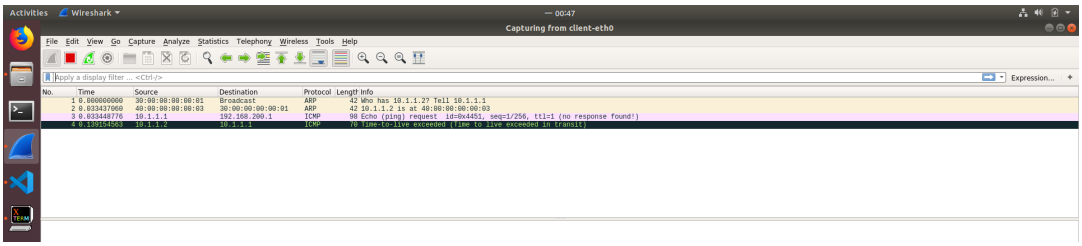


可以看到所有的测试均正确通过

在mininet上运行该拓扑，并使用wireshark进行抓包：

运行 client# ping -c 1 -t 1 192.168.200.1测试ttl为0是否正确发送错误信息：

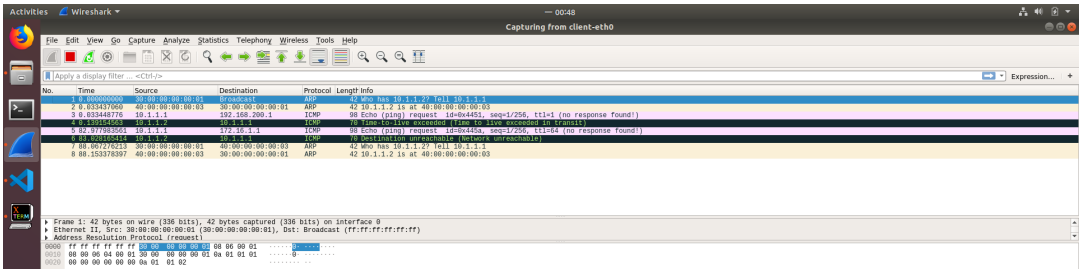
在wireshark上的结果如下：



可以看到client正确收到一个time-to-live exceed的icmp错误信息，因此路由器正确发送该错误信息

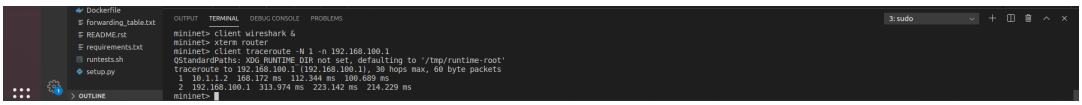
运行client# ping -c 1 172.16.1.1测试匹配不到转发表中的每一项时是否正确发送错误信息：

在wireshark上的结果如下：



由于是两次运行测试并未重启路由器，因此截图中包含了之前测试ttl的部分，可以看到在client ping 172.16.1.1之后收到了一个icmptype为 destination unreachable，而icmpcode为network unreachable的错误信息，因此路由器正确发送错误信息

使用tranceroute进行测试，输入client# traceroute -N 1 -n 192.168.100.1得到下面结果：



与实验手册上提供的截图基本一致，因此可以说明路由器的基本功能基本上已实现

3. 核心代码

构建icmp错误信息：

```
def createicmperror(self,origpkt,errortype,srcip,dstip):
    i = origpkt.get_header_index(Ethernet)
    del origpkt[i]
    icmp = ICMP()
    if errortype == 1:
        icmp.icmptype = ICMPType.DestinationUnreachable
        icmp.icmpcode = ICMPTypeCodeMap[icmp.icmptype].NetworkUnreachable
    elif errortype == 2:
        icmp.icmptype = ICMPType.TimeExceeded
        icmp.icmpcode = ICMPTypeCodeMap[icmp.icmptype].TTLExpired
    elif errortype == 3:
        icmp.icmptype = ICMPType.DestinationUnreachable
        icmp.icmpcode = ICMPTypeCodeMap[icmp.icmptype].HostUnreachable
    elif errortype == 4:
        icmp.icmptype = ICMPType.DestinationUnreachable
        icmp.icmpcode = ICMPTypeCodeMap[icmp.icmptype].PortUnreachable
    icmp.icmpdata.data = origpkt.to_bytes()[:28]
    icmp.icmpdata.origdgramlen = len(origpkt)
    str(icmp)
    ip = IPv4()
    ip.protocol = IPProtocol.ICMP
    ip.ttl = 32
    ip.src = srcip
    ip.dst = dstip
    e = Ethernet()
    pkt = e + ip + icmp
    return pkt
```

将构建icmp错误的信息的代码封装为一个函数，该函数接收四个参数，错误的包origpkt，错误类型errortype，源ip，目的ip，其中，错误类型是为了指定icmptype和icmpcode，errortype为1-4分别是手册上写出的1-4的错误，即分别为：转发表中没有匹配的项、数据包的ttl减到0、发送5次arp之后仍未得到回复、目的ip是路由器的某个端口ip，但是数据包的类型不是icmprequest，而在指定好错误类型之后，又为该数据包加上了以太网头，以便后续的代码直接可以取以太网头进行操作

下面针对每种错误详细进行说明：

1. **转发表中没有匹配的项：**

```
# not match
else:
    flag = 1
    maxlength = 0
    for index in forwardingtable:
        #debugger()
        matches = ipv4.src in index.prefixnet
        if matches == True:
            if index.prefixnet.prefixlen > maxlength:
                maxlength = index.prefixnet.prefixlen
                dest = index
                flag = 0
    if flag == 0:
        self.port = self.net.interface_by_name(dest.intfname)
        pkt = self.createicmperror(pkt,1,port.ipaddr,ipv4.src)
        ipv4 = pkt.get_header(IPv4)
```

首先使用发送的错误信息的src到转发表中去查找是否有匹配的项，如果找不到，则不管该错误的包直接丢弃，找到之后调用createicmperror函数构造icmp错误信息的包，其中错误类型为1，对应手册上的第一种错误，并将其赋值给pkt，同时修改ipv4为pkt所对应的ip包头，这样将错误信息伪装成一个要forwarding的包，即可复用lab4的转发逻辑为其进行发送

2. **ttl为0：**

```
# matched
if flag == 0:
    if pkt[IPv4].ttl == 0:
        pkt[IPv4].ttl = 1
        flag = 1
        maxlength = 0
        #debugger()
        for index in forwardingtable:
            #debugger()
            matches = ipv4.src in index.prefixnet
            if matches == True:
                if index.prefixnet.prefixlen > maxlength:
                    maxlength = index.prefixnet.prefixlen
                    dest = index
                    flag = 0
        #debugger()
        if flag == 0:
            port = self.net.interface_by_name(dest.intfname)
            pkt= self.createicmperror(pkt,2,port.ipaddr,ipv4.src)
            ipv4 = pkt.get_header(IPv4)
            #debugger()
```

由于错误之间具有优先顺序，因此ttl为0错误只有在匹配到转发表中的某一项时才会去检查，否则就会按照未匹配到转发表来生成错误信息，由于在刚收到数据包时就将其ttl减1，因此这里直接判断ttl是否为0，构造之前首先将错误包的ttl改为1，即还原之前的错误的包，然后按照与之前的错误类似的逻辑调用createicmperror函数，这里传的错误类型为2，对应ttl为0错误，并将其赋值给pkt，同时修改ipv4为pkt所对应的ip包头，这样将错误信息伪装成一个要forwarding的包，即可复用lab4的转发逻辑为其进行发送

3. **目的ip是路由器的某个端口ip，但是数据包的类型不是icmprequest：**

```
#not icmp request
if flag == 0:
    flag = 1
    maxlength = 0
    for index in forwardingtable:
        #debugger()
        matches = ipv4.src in index.prefixnet
        if matches == True:
            if index.prefixnet.prefixlen > maxlength:
                maxlength = index.prefixnet.prefixlen
                dest = index
                flag = 0
    if flag == 0:
        self.port = self.net.interface_by_name(dest.intfname)
        pkt = self.createicmperror(pkt,4,port.ipaddr,ipv4.src)
        ipv4 = pkt.get_header(IPv4)
        error = 1
```

检查到目的ip是路由器的某个端口ip，但是数据包的类型不是icmp request时，构建错误信息，整体逻辑与之前相似，只不过传给createicmperror函数的错误类型为4，对应数据包的类型不是icmprequest错误，由于检查其是不是icmp request在收到普通数据包时查找转发表的逻辑之前，因此这里额外设置标志位error，用来标识前面已经构造过错误信息了，因此接下来就不必再去查找转发表，故收到普通数据包去查找转发表的代码只有在error为0时才会去执行

以上三种错误都是在将数据包加入待发送队列之前即可检测到的，因此都将其伪装成一个待转发的数据包，复用lab4的转发逻辑为其进行发送

4. **发送5次arp request仍未收到arp reply：**

由于之前的代码中是将发送5次arp request仍未收到reply的数据包直接丢弃，因此只需更改原本丢弃的代码即可：

```
if index.cnt > 5:
    #debugger()
    pkt = index.pkt
    ipv4 = pkt.get_header(IPv4)
    flag = 1
    maxlength = 0
    for index2 in forwardingtable:
        #debugger()      You, 10 hours ago • pass all tests
        matches = ipv4.src in index2.prefixnet
        if matches == True:
            if index2.prefixnet.prefixlen > maxlength:
                maxlength = index2.prefixnet.prefixlen
                dest = index2
                flag = 0
    #debugger()
    if flag == 0:
        port = self.net.interface_by_name(dest.intfname)
        pkt = self.createicmperror(pkt,3,port.ipaddr,ipv4.src)
        ipv4 = pkt.get_header(IPv4)
    #debugger()

    port = self.net.interface_by_name(dest.intfname)
    if dest.nextip is None:
        nextip = ipv4.dst
    else:
        nextip = dest.nextip
    pkt[Ethernet].src = port.ethaddr
    #debugger()

    flag = 1
    for searchindex in arptable:
        if str(searchindex.ip) == str(nextip):
```

首先将pkt和ipv4指定为当前发送5次还未收到回复的数据包，然后按照与其他错误类型相似的逻辑构建icmp错误信息数据包，这里传给createicmperror的错误类型为3，对应于发送5次arp还未收到回复错误。由于检测到该错误只能是在处理待发送队列时，因此是在将一个数据包加入到待发送队列的逻辑之后，故并不能直接复用之前的代码，而是将在将这个数据包加入到待发送队列之前所做的一些必要工作的代码复制过来，即修改源mac地址，检查其是否与arptable中的某一项匹配

```
    flag = 1
    for searchindex in arptable:
        if str(searchindex.ip) == str(nextip):
            pkt[0].dst = searchindex.mac
            self.net.send_packet(dest.intfname,pkt)
            flag = 0
            break

    if flag == 1:
        #debugger()
        sendtemp = queueitem(pkt,nowtime - 1,nextip,1,dest.intfname)
        sendqueue.append(sendtemp)
    sendqueue.remove(index)
    index = sendtemp
```

而当其不与arptable中的任何一项匹配时才将其加入到待发送队列中，同时删除待发送队列中这个错误的包，并将index改为指向新加入的包，这样可以在这一次循环时就处理这个错误信息的包

总结与感想

- 整个lab3-5都是在完成一个路由器，且是使用上一个lab的代码来作为基础完成这一个lab的内容，但是我在完成lab3和lab4时，并不知道lab5的内容，因此导致当时写的代码仅针对lab3或者lab4，有的代码例如在转发表中查找相关的逻辑，本身并不复杂，且在lab4中只有一个地方使用，因此我个人的编程习惯是不会将其封装为一个函数来调用的，但是这部分相关的代码在lab5中其实是在多个地方被使用，因此在lab5中我是将其复制粘贴了好几次，最后导致在lab5中的代码冗余的部分比较多，最后代码的结构也比较难看，给debug带来一定的困扰，因此整体上而言编程习惯还有待改进
- 由于我并没有仔细的阅读api相关的内容，导致实验时出现了玄学bug，比如如果先把origpkt的内容去掉以太网头之后放到icmpdata.data里，然后再指定icmptype，最后测试时会出现icmpdata.data 不匹配的情况；再比如如果先把IPv4()和ICMP()加在一起，然后再用Ethernet()去加，会出现负载不对的报错