Lab5_Report

课程名称：**计算机网络** 任课教师：田臣/李文中 助教：

学院	计算机	专业（方向）	计算机
学号	181860077	姓名	佘帅杰
Email	3121416933@qq.com	开始完成日期	2020.5.4

实验名称

计算机网络试验5

实验目的

加强路由器

实现对ICMP包以及异常的处理

实验内容

实现发给自己的ICMP包的处理和回复

以及实现异常的处理

核心代码

数据结构

在lab4的基础上进行了微调

就是在pac里加上了一个icmp info用来在超时5次之后的重发提供信息，默认是空，对前面的代码不影响

```
class Node():
    def __init__(self,p,m,n,i):
        self.prefix=p
        self.mask=m
        self.nexthop=n
        self.name=i
#swyard -t routertests2.srpy myrouter.py

class Pac():
    def __init__(self,pac,thing,i=None):
        self.pac=pac
        self.ci=0
        self.time=0
        self.th=thing
```

```
        self.icmp_info=i

class Router(object):
    def __init__(self, net):
        self.net = net
        # other initialization stuff here
        self.interfaces = net.interfaces()
        self.ip_list=[intf.ipaddr for intf in self.interfaces]
        self.mac_list=[intf.ethaddr for intf in self.interfaces]
        self.arp_table={}
        self.forward_table=[]
```

代码逻辑

全局函数

一个是提供的模板里的代码

主要就是

1. 构造以太网包头
2. 构造Ipv4包头，并指定源和目的地址，选定类型和ttl
3. 构造ICMP包头，这里是构造错误包，也就是选择对应的错误类型和错误码，然后取28位，按照手册上操作
4. 最后拼接即可

另一个是处理发送的ICMP ping request的函数

也是按照手册上的进行操作，拷贝数据和序列号，指定地址等

最后返回构造好的新的包

```
def mk_icmperr(hwsrc, hwdst, ipsrc, ipdst, xtype, xcode=0, origpkt=None,
ttl=64):
    ether = Ethernet()
    ether.src = EthAddr(hwsrc)
    ether.dst = EthAddr(hwdst)
    ether.ethertype = EtherType.IP
    ippkt = IPv4()
    ippkt.src = IPAddr(ipsrc)
    ippkt.dst = IPAddr(ipdst)
    ippkt.protocol = IPProtocol.ICMP
    ippkt.ttl = ttl
    ippkt.ipid = 0
    icmppkt = ICMP()
    icmppkt.icmptype = xtype
    icmppkt.icmpcode = xcodeyua
    if origpkt is not None:
        xpkt = deepcopy(origpkt)
        i = xpkt.get_header_index(Ethernet)
        if i >= 0:
            del xpkt[i]
        icmppkt.icmpdata.data = xpkt.to_bytes()[:28]
        icmppkt.icmpdata.origdgramlen = len(xpkt)

```

```
    return ether + ippkt + icmppkt

def deal_with_ping(packet,dstip):
    ether = Ethernet()
    ether.ethertype = EtherType.IP
    ippkt = IPv4()
    ippkt.src = IPAddr(dstip)
    ippkt.dst = IPAddr(packet[IPv4].src)
    ippkt.protocol = IPProtocol.ICMP
    ippkt.ttl = 64
    ippkt.ipid = 0

    icmppkt = ICMP()
    icmppkt.icmptype = ICMPType.EchoReply
    icmppkt.icmpcode = ICMPCodeEchoReply.EchoReply

    icmppkt.icmpdata.sequence = packet[ICMP].icmpdata.sequence
    icmppkt.icmpdata.data = packet[ICMP].icmpdata.data

    return ether + ippkt + icmppkt

```

队列处理

多数代码继承于Lab4，最大的不同就是在5次发送失败之后不只是输出信息并放弃了

这一块代码是用来处理队列的头元素，并且顺利匹配并发送的情况

```
if len(q)!=0:
    for i in self.interfaces:
        if i.name==q[0].th.name:      You, 6 hours ago • Working on Task3 (sry for
            port=i
    if q[0].th.nexthop is None:
        targetip=q[0].pac[IPv4].dst
    else:
        targetip=q[0].th.nexthop
    find_flag2=0
    for (k,v) in  self.arp_table.items():
        if targetip == k:
            print("common",targetip)
            q[0].pac[Ethernet].dst=v
            q[0].pac[Ethernet].src=port.ethaddr
            print("send pac (find) ",q[0].pac," through ",port)
            self.net.send_packet(port,q[0].pac)
            find_flag2=1
            del(q[0])
            break
```

如上可见，没有匹配的时候标志位find_flag2指示

那么对应的判断处理

判断发了5次了，那么找到当时接受的端口（这个时候相对于lab4修改的数据结构就有了意义）

然后构造出对应的错误信息ICMP包，源ip使用接受端口ip，目的ip用包里的源ip，错误码等通过查阅手册也可以正确的指定，那么就顺利的构造了

随后执行和之前一样的操作，构造出队列元素（lab4内容），不管顺利与否都删除队头元素

```
if find_flag2 ==0:
    if q[0].ci >= 5:
        print("Time out And fail to send arp request Final Give up")
        for i in self.interfaces:
            if i.name==q[0].icmp_info:
                receive_port4=i
                break

        pkt=mk_icmperr(q[0].pac[Ethernet].dst,q[0].pac[Ethernet].src,receive_port4.ipaddr,q[0].pac[IPv4].src, ICMPType.DestinationUnreachable, xcode=1, origpkt=pkt, ttl=64)
        head=(pkt[IPv4])
        pos=-1
        prefixlen=-1
        index=0
        for i in self.forward_table:
            if ((int(head.dst)&int(i.mask))==int(i.prefix)):
                netaddr = IPv4Network(str(i.prefix)+"/"+str(i.mask))
                if netaddr.prefixlen > prefixlen:
                    prefixlen=netaddr.prefixlen
                    pos=index
            index+=1
        del(q[0])
        print("send arp fail fix and reput in queue")
        immediate=Pac(pkt,self.forward_table[pos],receive_port4.name)
        print(immediate.th.name)
        for i in self.interfaces:
            if i.name==immediate.th.name:
                port=i
        if immediate.th.nexthop is None:
            targetip=immediate.pac[IPv4].dst
        else:
            targetip=immediate.th.nexthop
        print(targetip)
        find_flag3=0
```

上述的处理例程已经构造了所需要的包，并封装成了队列元素的形式，这个时候就可以统一到对队列元素的处理例程，也就是查表，发送arp request

总的来说就是查arp 表，有就发，没有就发request然后把当前的包放进队列等待处理

等到这个队列处理结束的时候开始收包，刚好就会收到arp reply，然后又进入了最初的循环，那么队列包就被处理了

最后的一个else是和上面的发送了5次的判断对立的，也就是没有发5次，那就看时间间隔后发送

这个内容和lab4完全一样

```
        find_flag3=0
        for (k,v) in  self.arp_table.items():
            print(k)
            if targetip == k:
                immediate.pac[Ethernet].dst=v
                immediate.pac[Ethernet].src=port.ethaddr
                find_flag3=1
                print("send pac (find) ",immediate.pac," through ",port)
                self.net.send_packet(port,immediate.pac)
                break
        if find_flag3==0:
            ether = Ethernet()
            ether.src = port.ethaddr
            ether.dst = 'ff:ff:ff:ff:ff:ff'
            ether.ethertype = EtherType.ARP
            arp = Arp(operation=ArpOperation.Request,senderhwaddr=port.ethaddr,senderprotoaddr=port.ipaddr,targethwaddr='ff:ff:ff:ff:ff:ff',targetprotoaddr=targetip)
            arppacket = ether + arp
            print("send requests",port)
            self.net.send_packet(port, arppacket)
            q.append(immediate)
    else:
        cur=time.time()
        if (q[0].ci==0) or (cur-q[0].time)>1:
            ether = Ethernet()
            ether.src = port.ethaddr
            ether.dst = 'ff:ff:ff:ff:ff:ff'
            ether.ethertype = EtherType.ARP
            arp = Arp(operation=ArpOperation.Request,senderhwaddr=port.ethaddr,senderprotoaddr=port.ipaddr,targethwaddr='ff:ff:ff:ff:ff:ff',targetprotoaddr=targetip)
            arppacket = ether + arp
            print("send requests",port)
            self.net.send_packet(port, arppacket)
            q[0].ci+=1
            q[0].time=time.time()
            print(q[0].time)
```

ICMP处理

收到了包先看看是不是ipv4

如果是就看看这个地址是不是路由器的地址，注意到这里的icmp_flag的设置

这个标志位专门指示非icmp且目标为路由器的包，先设置为1，如果ip是路由器的就设置为0，如果是ICMP就解除警戒，不然就会触发异常处理

对于icmp包的处理上述有一个专门的全局函数，不再赘述，全局函数返回了处理后的包，然后只需要把当前的包替换掉收到的包，就可以无缝套用lab4的代码（针对ipv4包的转发以及队列等操作）

如果触发了上述的异常，那就构造出异常信息包（调用另一个全局函数），同样的替换收到的包，得到和上述一样的结果

```
if gotpkt:
    log_info("Got a packet: {}".format(str(pkt)))
    if pkt.has_header(IPv4):
        head=(pkt[IPv4])
        if head is None:
            print("error")
        icmp_flag=1
        if pkt[IPv4].dst in self.ip_list:
            print("Match router ip")
            icmp_flag=0
            if pkt.has_header(ICMP) :
                if  pkt[ICMP].icmptype == ICMPType.EchoRequest:
                    print("ICMP ping request Packet")
                    icmp_flag=1
                    for i in self.ip_list:
                        if i==head.dst:
                            pkt=deal_with_ping(pkt,i)
                            print(pkt)
                            head=(pkt[IPv4])
                            break
        if icmp_flag==0:
            print("Not ICMP request pac Hit !")
            for i in self.interfaces:
                if i.name==dev:
                    receive_port=i
                    break
            print("send port",dev)
            pkt=mk_icmperr(pkt[Ethernet].dst, pkt[Ethernet].src,receive_port.ipaddr,pkt[IPv4].src, ICMPType.DestinationUnreachable, xcode=3, origpkt=pkt, ttl=64)
            head=(pkt[IPv4])
```

经过了上述的处理排除了所谓的非ICMP包的威胁，如果有则用处理包替换错误包

这个时候处理就融入了普通的ipv4包处理，进行ttl的减小，然后判断是否触发ttl异常

如果触发就如上的构造异常信息包并进行替换

然后就是进行最大前缀的匹配等工作了

```
head.ttl-=1
if head.ttl <= 0:
    for i in self.interfaces:
        if i.name==dev:
            receive_port2=i
            break
    pkt=mk_icmperr(pkt[Ethernet].dst, pkt[Ethernet].src,receive_port2.ipaddr,pkt[IPv4].src, 11, xcode=0, origpkt=pkt, ttl=64)
    head=(pkt[IPv4])
    print("TimeExceed",pkt)
print("ipv4",head)
pos=-1
prefixlen=-1
index=0
for i in self.forward_table:
    if ((int(head.dst)&int(i.mask))==int(i.prefix)):
        netaddr = IPv4Network(str(i.prefix)+"/"+str(i.mask))
        if netaddr.prefixlen > prefixlen:
            prefixlen=netaddr.prefixlen
            pos=index
    index+=1              You, 6 hours ago • Working on Task3 (sry for forget to commit)
```

如上进行了前缀匹配

可以看到index指示当前的下标，pos则指示的是选中的下标

如果pos=-1也就是失配了，这个时候触发异常，不存在的主机

同理构造信息包，最后替换，这个时候已经走到末尾了，直接进一步的封装到队列包的形式入队等候处理

反之就是一路顺风，不再构造，直接封装入队

```
if pos == -1:
    print("No Match Some Error occur!!!!!!!!!!!!!!!!")
    for i in self.interfaces:
        if i.name==dev:
            receive_port3=i
            break
    pkt=mk_icmperr(pkt[Ethernet].dst, pkt[Ethernet].src,receive_port3.ipaddr,pkt[IPv4].src, 3, xcode=0, origpkt=pkt, ttl=64)
    head=(pkt[IPv4])
    pos=-1
    prefixlen=-1
    index=0
    for i in self.forward_table:
        if ((int(head.dst)&int(i.mask))==int(i.prefix)):
            netaddr = IPv4Network(str(i.prefix)+"/"+str(i.mask))
            if netaddr.prefixlen > prefixlen:
                prefixlen=netaddr.prefixlen
                pos=index
        index+=1
    print("refix and add packet to queue")
    q.append(Pac(pkt,self.forward_table[pos],dev))
else:
    print("add packet to queue")
    q.append(Pac(pkt,self.forward_table[pos],dev))
```

最后的一块是arp处理，从lab3过来的，不再赘述

测试

测试样例

如下结合测试输出信息解释一下我的程序在测试中的逻辑表现

（中文注释）

```
(syenv) njucs@njucs-VirtualBox:~/switchyard/lab_5$ swyard -t
routertests3.srpy myrouter.py
21:46:24 2020/05/04     INFO Starting test scenario routertests3.srpy
192.168.1.0   255.255.255.0   None   router-eth0
10.10.0.0   255.255.0.0   None   router-eth1
172.16.42.0   255.255.255.252   None   router-eth2
172.16.0.0   255.255.0.0   192.168.1.2   router-eth0
172.16.128.0   255.255.192.0   10.10.0.254   router-eth1
172.16.64.0   255.255.192.0   10.10.1.254   router-eth1
10.100.0.0   255.255.0.0   172.16.42.2   router-eth2

# 发出了ICMP request
21:46:24 2020/05/04     INFO Got a packet: Ethernet ab:cd:00:00:00:47-
>10:00:00:00:00:01 IP | IPv4 172.16.111.222->192.168.1.1 ICMP | ICMP
EchoRequest 0 42 (12 data bytes)
# 匹配路由器ip
Match router ip
ICMP ping request Packet
Ethernet 00:00:00:00:00:00->00:00:00:00:00:00 IP | IPv4 192.168.1.1-
>172.16.111.222 ICMP | ICMP EchoReply 0 42 (12 data bytes)
ipv4 IPv4 192.168.1.1->172.16.111.222 ICMP
# 构造处理对应的回应包，入队
add packet to queue
21:46:24 2020/05/04     INFO Not arp Packet
21:46:24 2020/05/04     INFO Table Shown as follows
   # 要发的时候发现arp失配 发出arp请求
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
1588599984.7286534
# 收到arp reply进行处理
21:46:24 2020/05/04     INFO Got a packet: Ethernet ab:cd:00:00:00:01-
>10:00:00:00:00:02 ARP | Arp ab:cd:00:00:00:01:10.10.1.254
10:00:00:00:00:02:10.10.0.1
21:46:24 2020/05/04     INFO operation kind ArpOperation.Reply
21:46:24 2020/05/04     INFO recive arp reply
21:46:24 2020/05/04     INFO Table Shown as follows

        #可以看到这里的表项增加
10.10.1.254      ab:cd:00:00:00:01
10.10.0.1      10:00:00:00:00:02
common 10.10.1.254

#此时可以匹配了，发出
send pac (find)  Ethernet 10:00:00:00:00:02->ab:cd:00:00:00:01 IP | IPv4
192.168.1.1->172.16.111.222 ICMP | ICMP EchoReply 0 42 (12 data bytes)
 through  router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16

 # 又有一个ICMP request
```

```
21:46:24 2020/05/04      INFO Got a packet: Ethernet ab:cd:00:00:00:01->10:00:00:00:00:02 IP | IPv4 172.16.111.222->10.10.0.1 ICMP | ICMP EchoRequest 0 42 (0 data bytes)
                    # 命中路由器ip 随后的过程同上
Match router ip
ICMP ping request Packet
Ethernet 00:00:00:00:00:00->00:00:00:00:00:00 IP | IPv4 10.10.0.1->172.16.111.222 ICMP | ICMP EchoReply 0 42 (0 data bytes)
ipv4 IPv4 10.10.0.1->172.16.111.222 ICMP
add packet to queue
21:46:24 2020/05/04      INFO Not arp Packet
21:46:24 2020/05/04      INFO Table Shown as follows
10.10.1.254       ab:cd:00:00:00:01
10.10.0.1       10:00:00:00:00:02
common 10.10.1.254
send pac (find)  Ethernet 10:00:00:00:00:02->ab:cd:00:00:00:01 IP | IPv4 10.10.0.1->172.16.111.222 ICMP | ICMP EchoReply 0 42 (0 data bytes)
 through  router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16

 # 这里是故意发的一个ttl=1的包
21:46:24 2020/05/04      INFO Got a packet: Ethernet be:ef:00:00:00:01->10:00:00:00:00:02 IP | IPv4 10.10.123.123->10.100.1.1 ICMP | ICMP EchoRequest 0 42 (0 data bytes)

                # 构造错误包头
TimeExceed Ethernet 10:00:00:00:00:02->be:ef:00:00:00:01 IP | IPv4 10.10.0.1->10.10.123.123 ICMP | ICMP TimeExceeded:TTLExpired 28 bytes of raw payload (b'E\x00\x00\x1c\x00\x00\x00\x00\x00\x01') OrigDgramLen: 28
ipv4 IPv4 10.10.0.1->10.10.123.123 ICMP

#错误包替换原包入队
add packet to queue
21:46:24 2020/05/04      INFO Not arp Packet
21:46:24 2020/05/04      INFO Table Shown as follows
10.10.1.254       ab:cd:00:00:00:01
10.10.0.1       10:00:00:00:00:02

        # arp失配 发request
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
1588599984.7450554
21:46:24 2020/05/04      INFO Got a packet: Ethernet be:ef:00:00:00:01->10:00:00:00:00:02 ARP | Arp be:ef:00:00:00:01:10.10.123.123 10:00:00:00:00:02:10.10.0.1
21:46:24 2020/05/04      INFO operation kind ArpOperation.Reply
21:46:24 2020/05/04      INFO recive arp reply
21:46:24 2020/05/04      INFO Table Shown as follows
10.10.1.254       ab:cd:00:00:00:01
10.10.0.1       10:00:00:00:00:02
10.10.123.123       be:ef:00:00:00:01
common 10.10.123.123

# 错误包被发出
send pac (find)  Ethernet 10:00:00:00:00:02->be:ef:00:00:00:01 IP | IPv4 10.10.0.1->10.10.123.123 ICMP | ICMP TimeExceeded:TTLExpired 28 bytes of raw payload (b'E\x00\x00\x1c\x00\x00\x00\x00\x00\x01') OrigDgramLen: 28
 through  router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16

      # 这一次是ping了一个不存在的位置，直接失配
```

```
21:46:24 2020/05/04     INFO Got a packet: Ethernet ab:cd:00:00:00:01->10:00:00:00:00:02 IP | IPv4 10.10.123.123->1.2.3.4 ICMP | ICMP EchoRequest 0 42 (0 data bytes)
ipv4 IPv4 10.10.123.123->1.2.3.4 ICMP
No Match Some Error occur!!!!!!!!!!!!!!!!

# 发现失配，构造新包入队
refix and add packet to queue
21:46:24 2020/05/04     INFO Not arp Packet
21:46:24 2020/05/04     INFO Table Shown as follows
10.10.1.254      ab:cd:00:00:00:01
10.10.0.1      10:00:00:00:00:02
10.10.123.123      be:ef:00:00:00:01
common 10.10.123.123
send pac (find)  Ethernet 10:00:00:00:00:02->be:ef:00:00:00:01 IP | IPv4 10.10.0.1->10.10.123.123 ICMP | ICMP DestinationUnreachable:NetworkUnreachable 28 bytes of raw payload (b'E\x00\x00\x1c\x00\x00\x00\x00)\x01') NextHopMTU: 0  through  router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16



     # 这一次是UDP包
21:46:24 2020/05/04     INFO Got a packet: Ethernet ab:cd:00:00:00:01->10:00:00:00:00:02 IP | IPv4 172.16.111.222->192.168.1.1 UDP | UDP 10000->4444 | RawPacketContents (5 bytes) b'Help!'
Match router ip

#这里可见触发了不是ICMP包的警报 同理换包入队发出
Not ICMP request pac Hit !
send port router-eth1
ipv4 IPv4 10.10.0.1->172.16.111.222 ICMP
add packet to queue
21:46:24 2020/05/04     INFO Not arp Packet
21:46:24 2020/05/04     INFO Table Shown as follows
10.10.1.254      ab:cd:00:00:00:01
10.10.0.1      10:00:00:00:00:02
10.10.123.123      be:ef:00:00:00:01
common 10.10.1.254
send pac (find)  Ethernet 10:00:00:00:00:02->ab:cd:00:00:00:01 IP | IPv4 10.10.0.1->172.16.111.222 ICMP | ICMP DestinationUnreachable:PortUnreachable 28 bytes of raw payload (b'E\x00\x00!\x00\x00\x00\x00%\x11') NextHopMTU: 0  through  router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16


 #收到了icmp request 这一次是超时测试
21:46:24 2020/05/04     INFO Got a packet: Ethernet ab:cd:ef:ab:cd:ef->10:00:00:00:00:01 IP | IPv4 192.168.1.239->10.10.50.250 ICMP | ICMP EchoRequest 0 42 (0 data bytes)
ipv4 IPv4 192.168.1.239->10.10.50.250 ICMP
add packet to queue
21:46:24 2020/05/04     INFO Not arp Packet
21:46:24 2020/05/04     INFO Table Shown as follows
10.10.1.254      ab:cd:00:00:00:01
10.10.0.1      10:00:00:00:00:02
10.10.123.123      be:ef:00:00:00:01
```

```
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
1588599984.751784
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
1588599986.2535422
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
1588599987.7551665
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
1588599989.2574625
send requests router-eth1 mac:10:00:00:00:00:02 ip:10.10.0.1/16
1588599990.759004
Time out And fail to send arp request Final Give up

#如上的五次重发，触发了警报，构造新包入队
send arp fail fix and reput in queue
router-eth0
192.168.1.239
10.10.1.254
10.10.0.1
10.10.123.123

#往回发，发现mac也不知道，发送arp request
send requests router-eth0 mac:10:00:00:00:00:01 ip:192.168.1.1/24
21:46:32 2020/05/04     INFO Got a packet: Ethernet 00:01:02:03:04:05->10:00:00:00:00:01 ARP | Arp 00:01:02:03:04:05:192.168.1.239 10:00:00:00:00:01:192.168.1.1
21:46:32 2020/05/04     INFO operation kind ArpOperation.Reply
21:46:32 2020/05/04     INFO recive arp reply
21:46:32 2020/05/04     INFO Table Shown as follows
10.10.1.254       ab:cd:00:00:00:01
10.10.0.1       10:00:00:00:00:02
10.10.123.123       be:ef:00:00:00:01
192.168.1.239       00:01:02:03:04:05
192.168.1.1       10:00:00:00:00:01
common 192.168.1.239

#发回
send pac (find)  Ethernet 10:00:00:00:00:01->00:01:02:03:04:05 IP | IPv4 192.168.1.1->192.168.1.239 ICMP | ICMP DestinationUnreachable:HostUnreachable 28 bytes of raw payload (b'E\x00\x00\x1c\x00\x00\x00\x00?\x01') NextHopMTU: 0  through  router-eth0 mac:10:00:00:00:00:01 ip:192.168.1.1/24
```

```
Results for test scenario IP forwarding and ARP requester tests: 28 passed, 0 failed, 0 pending


Passed:
1   ICMP echo request (PING) for the router IP address
    192.168.1.1 should arrive on router-eth0.  This PING is
    directed at the router, and the router should respond with
    an ICMP echo reply.
2   Router should send an ARP request for 10.10.1.254 out
    router-eth1.
3   Router should receive ARP reply for 10.10.1.254 on router-
    eth1.
4   Router should send ICMP echo reply (PING) to 172.16.111.222
    out router-eth1 (that's right: ping reply goes out a
    different interface than the request).
5   ICMP echo request (PING) for the router IP address 10.10.0.1
    should arrive on router-eth1.
6   Router should send ICMP echo reply (PING) to 172.16.111.222
    out router-eth1.
7   ICMP echo request (PING) for 10.100.1.1 with a TTL of 1
    should arrive on router-eth1.  The router should decrement
    the TTL to 0 then see that the packet has "expired" and
    generate an ICMP time exceeded error.
8   Router should send ARP request for 10.10.123.123 out router-
    eth1.
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.
```

deploy mininet

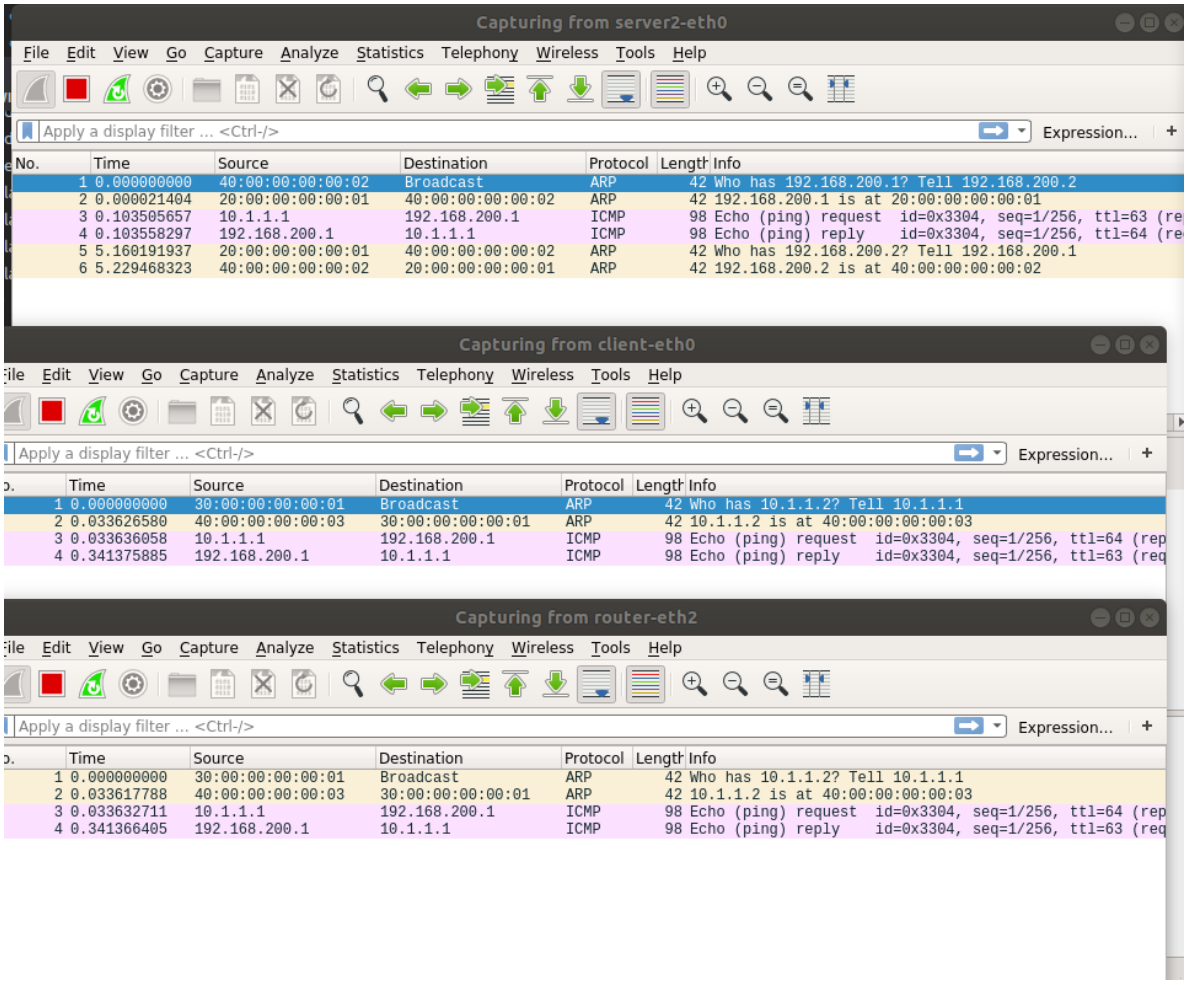
正常使用

正常的使用就是client召唤server2

整体流程：

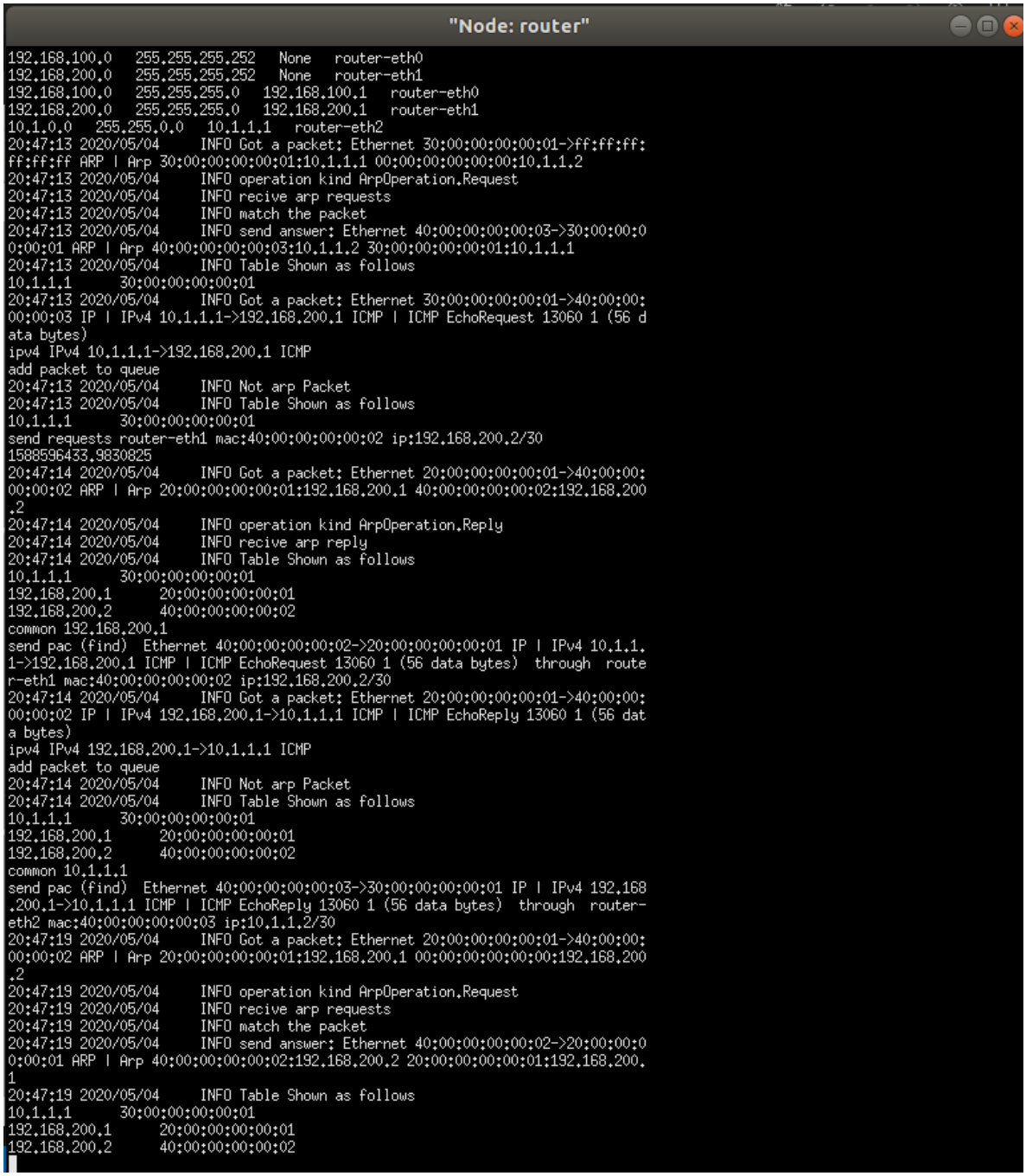
1. client广播问server2的位置
2. 路由器先和client建立连接，也就是client和router里的who is 10.1.1.2的原因
3. 随后路由器回答了client 10.1.1.2（也就是和client相接的端口的ip对应的mac）
4. client发出了ICMP request
5. 路由器收到了request
6. 路由器一看发现目的地自己不认识，发出了广播（这是为什么server2先收到了来自40开头的arp包）
7. server2回答，路由器收到了，发出来ICMP
8. server2回复reply，但是和client一样，先和路由器的端口连接，后面多出了两个arp包
9. 这个时候路由器转发了这个包，也正是抓包的最后一个包
10. client收到了，收工

```
client# ping -c 1 -t 1 192.168.200.1
```



对应router的心理活动

处理思路其实和上面的测试样例里的表现差不多。。

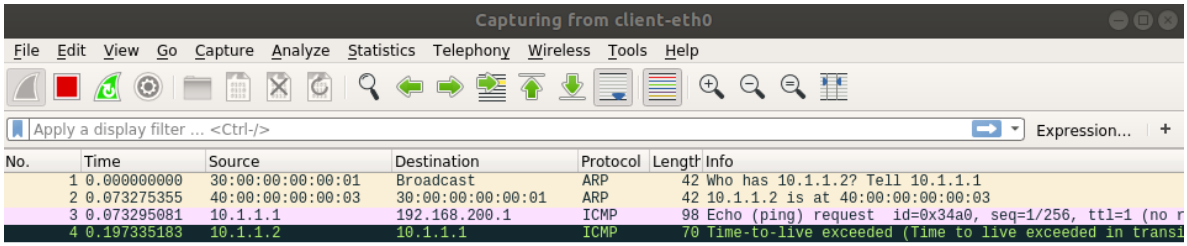


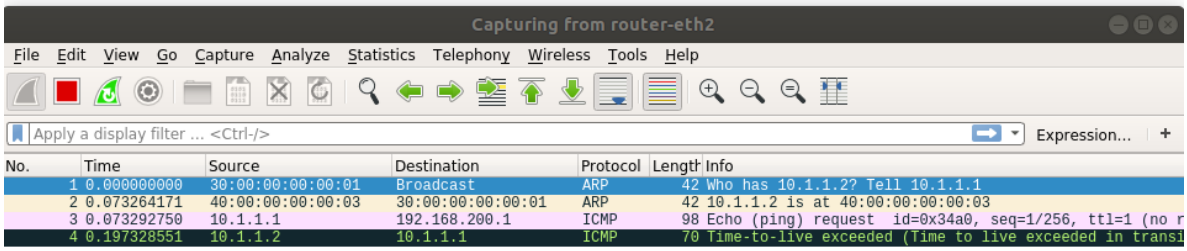
TTL超时

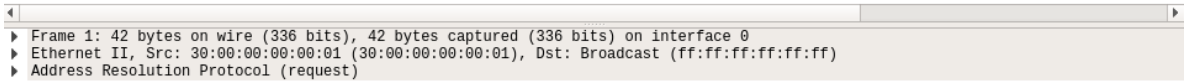
处理相似于上述

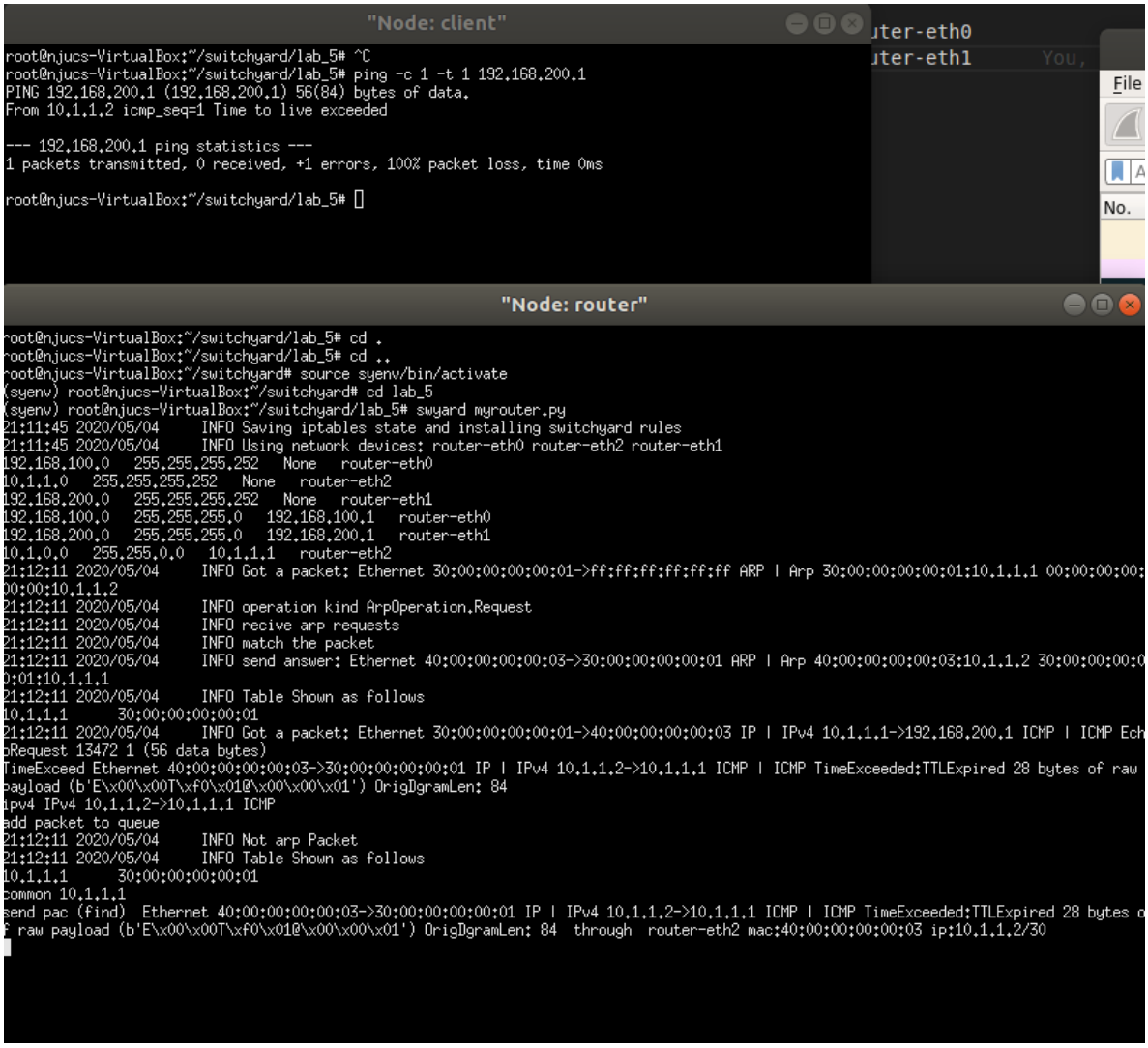
发出arp互相试探，然后发出ICMP

没想到被路由器发现问题，直接打回，双方收到4个包，随后一个是异常信息包


Capturing from client-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
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 0.073275355 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 0.073295081 10.1.1.1 192.168.200.1 ICMP 98 Echo (ping) request id=0x34a0, seq=1/256, ttl=1 (no r
4 0.197335183 10.1.1.2 10.1.1.1 ICMP 70 Time-to-live exceeded (Time to live exceeded in transi


Capturing from router-eth2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
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 0.073264171 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 0.073292750 10.1.1.1 192.168.200.1 ICMP 98 Echo (ping) request id=0x34a0, seq=1/256, ttl=1 (no r
4 0.197328551 10.1.1.2 10.1.1.1 ICMP 70 Time-to-live exceeded (Time to live exceeded in transi

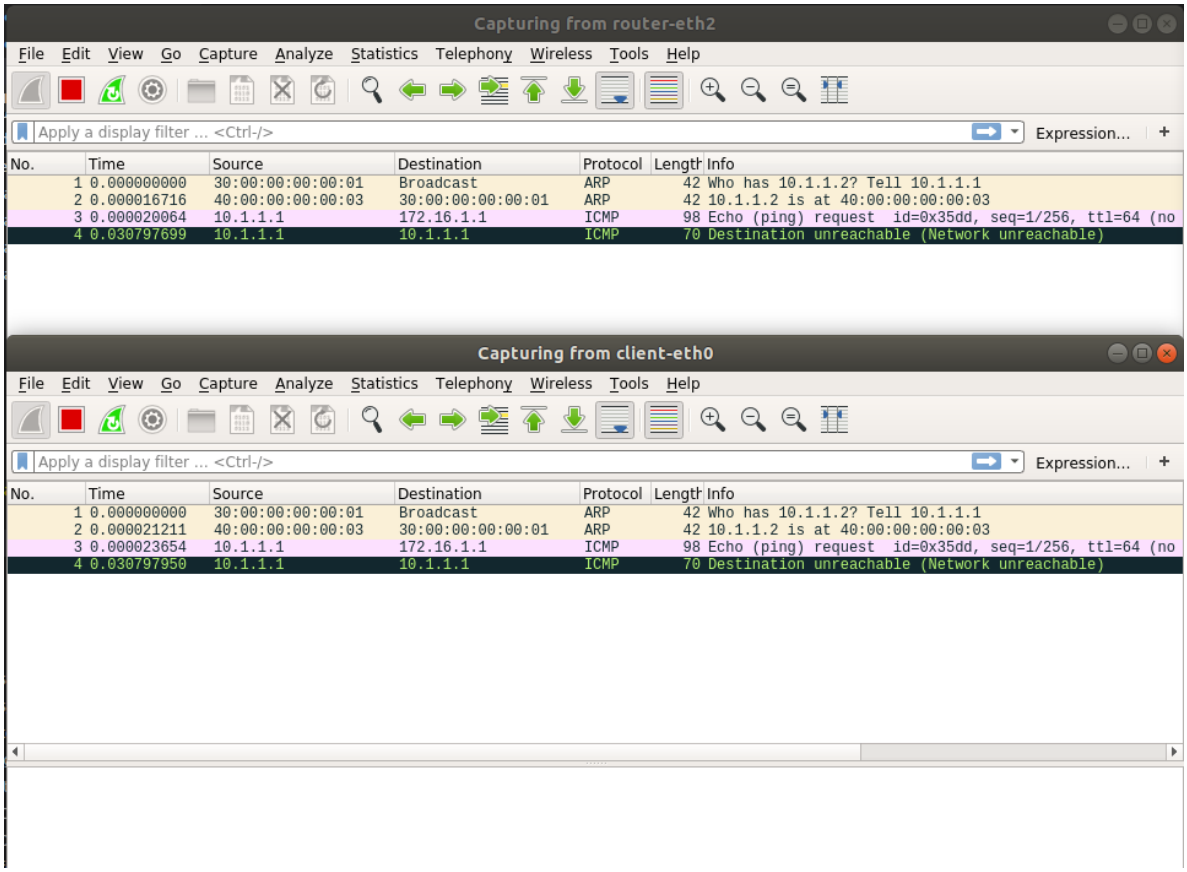

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)

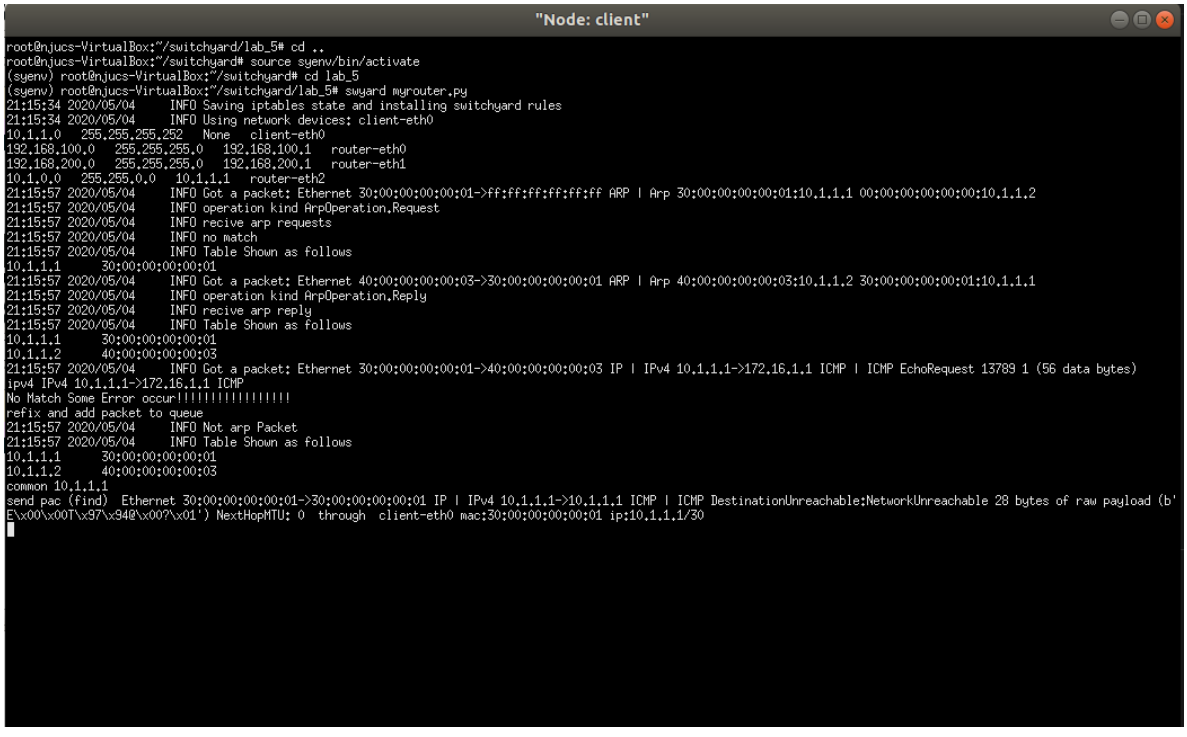

"Node: client"
root@njucs-VirtualBox:~/switchyard/lab_5# ^C
root@njucs-VirtualBox:~/switchyard/lab_5# ping -c 1 -t 1 192.168.200.1
PING 192.168.200.1 (192.168.200.1) 56(84) bytes of data.
From 10.1.1.2 icmp_seq=1 Time to live exceeded
--- 192.168.200.1 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
root@njucs-VirtualBox:~/switchyard/lab_5#
"Node: router"
root@njucs-VirtualBox:~/switchyard/lab_5# cd .
root@njucs-VirtualBox:~/switchyard/lab_5# cd ..
root@njucs-VirtualBox:~/switchyard# source syenv/bin/activate
(syenv) root@njucs-VirtualBox:~/switchyard# cd lab_5
(syenv) root@njucs-VirtualBox:~/switchyard/lab_5# swyard myrouter.py
21:11:45 2020/05/04 INFO Saving iptables state and installing switchyard rules
21:11:45 2020/05/04 INFO Using network devices: router-eth0 router-eth2 router-eth1
192.168.100.0 255.255.255.252 None router-eth0
10.1.1.0 255.255.255.252 None router-eth2
192.168.200.0 255.255.255.252 None router-eth1
192.168.100.0 255.255.255.0 192.168.100.1 router-eth0
192.168.200.0 255.255.255.0 192.168.200.1 router-eth1
10.1.0.0 255.255.0.0 10.1.1.1 router-eth2
21:12:11 2020/05/04 INFO Got a packet: Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00:00:00:00:01:10.1.1.1 00:00:00:00:
00:00:10.1.1.2
21:12:11 2020/05/04 INFO operation kind ArpOperation.Request
21:12:11 2020/05/04 INFO recive arp requests
21:12:11 2020/05/04 INFO match the packet
21:12:11 2020/05/04 INFO send answer: Ethernet 40:00:00:00:00:03->30:00:00:00:00:01 ARP | Arp 40:00:00:00:00:03:10.1.1.2 30:00:00:00:0
0:01:10.1.1.1
21:12:11 2020/05/04 INFO Table Shown as follows
10.1.1.1 30:00:00:00:00:01
21:12:11 2020/05/04 INFO Got a packet: Ethernet 30:00:00:00:00:01->40:00:00:00:00:03 IP | IPv4 10.1.1.1->192.168.200.1 ICMP | ICMP Ech
oRequest 13472 1 (56 data bytes)
TimeExceed Ethernet 40:00:00:00:00:03->30:00:00:00:00:01 IP | IPv4 10.1.1.2->10.1.1.1 ICMP | ICMP TimeExceeded:TTLExpired 28 bytes of raw
payload (b'E\x00\x00T\xf0\x01@\x00\x00\x01') OrigDgramLen: 84
ipv4 IPv4 10.1.1.2->10.1.1.1 ICMP
add packet to queue
21:12:11 2020/05/04 INFO Not arp Packet
21:12:11 2020/05/04 INFO Table Shown as follows
10.1.1.1 30:00:00:00:00:01
common 10.1.1.1
send pac (find) Ethernet 40:00:00:00:00:03->30:00:00:00:00:01 IP | IPv4 10.1.1.2->10.1.1.1 ICMP | ICMP TimeExceeded:TTLExpired 28 bytes o
f raw payload (b'E\x00\x00T\xf0\x01@\x00\x00\x01') OrigDgramLen: 84 through router-eth2 mac:40:00:00:00:00:03 ip:10.1.1.2/30

ping空主机

和上面一样，到路由器的时候被发现打回

对应的处理例程说明都在测试样例了里有说过



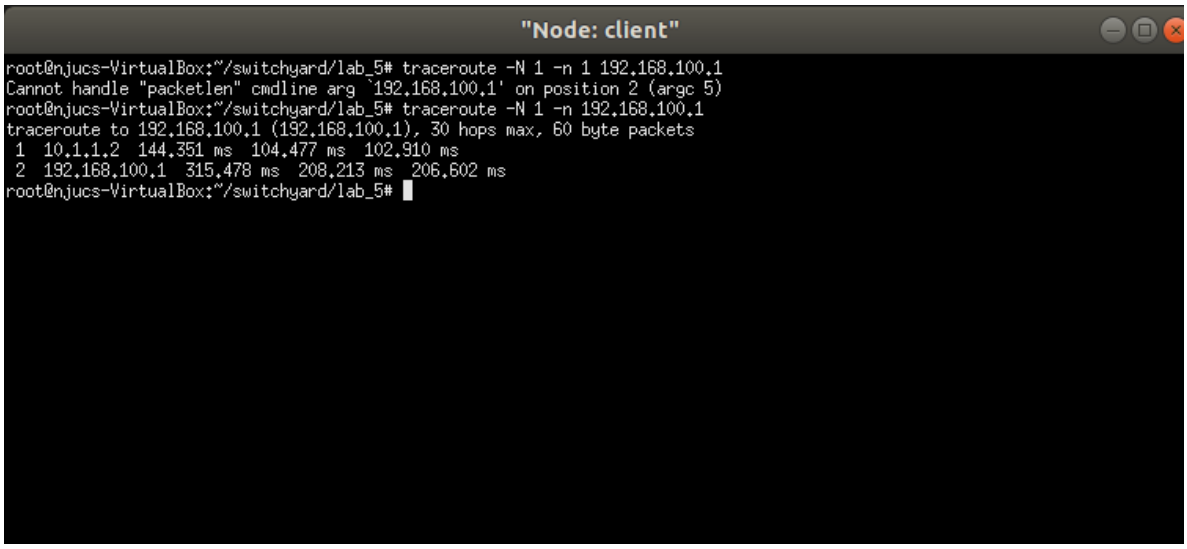


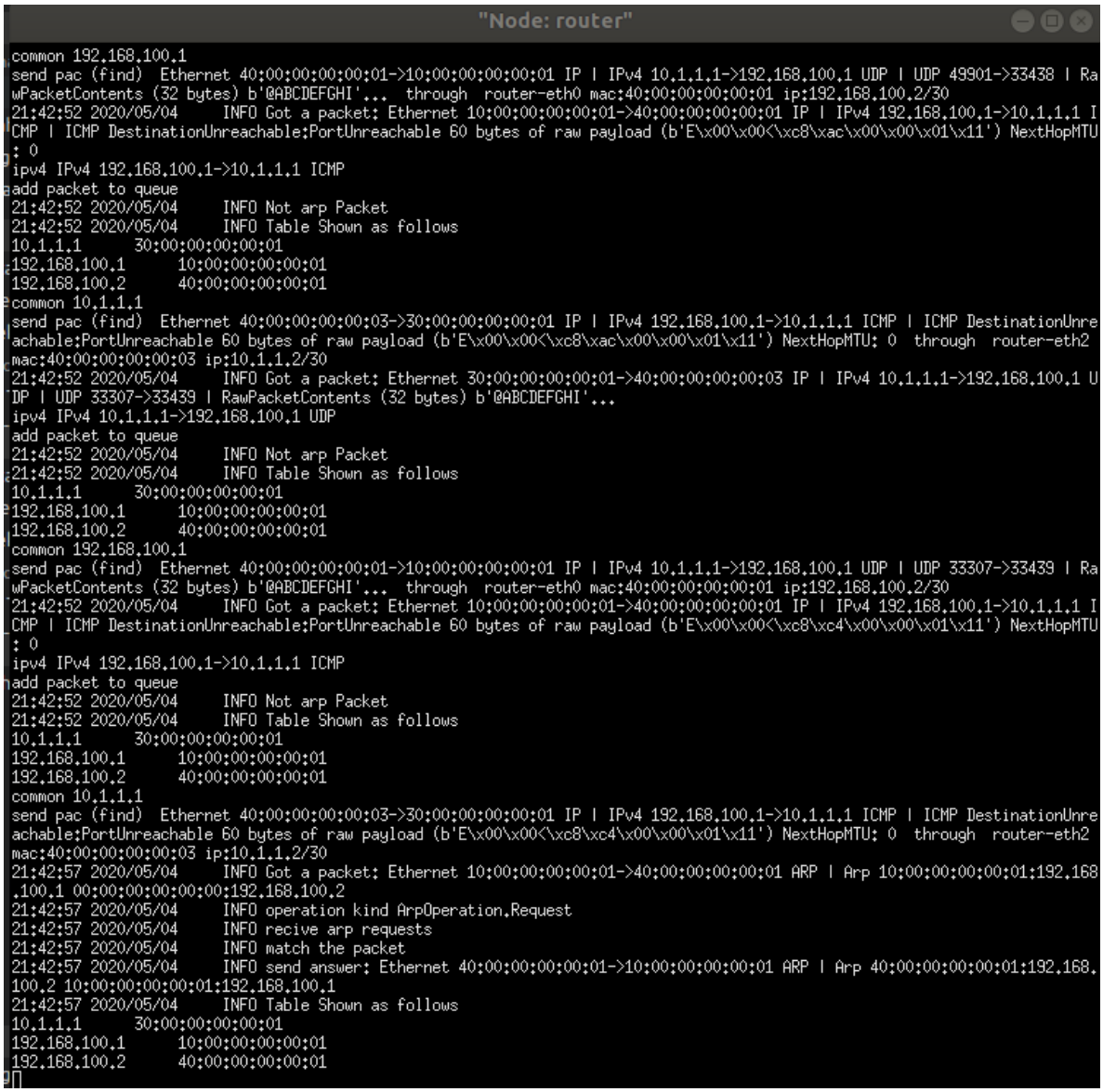
TraceRoute

一致，正确

处理例程应该就是client ping server1

先到了和client接壤的位置，然后到了server1





问题

本次实验我有一个小的问题

在处理5次重发的时候，测试样例要求5次重发检测到，随后发出arp request，然后触发收包，然后下一次循环发出错误信息包。（参考24-27样例）

这个就要求在最初处理队列的时候发现重发，然后在收包之前（也就是依旧在处理队列的逻辑里）发现新构造的包arp无匹配并发出对应的arp request，也就是说在重发包的处理逻辑里不得不套用在处理队列最初的代码（包括查表，发request等工作）

然后我有一个想法就是在重发处理里，只构造错误信息包，然后放入队列，就结束队列处理逻辑，转入收包工作。这个样子理论上收包为空，进入下一次循环也就是队列处理，这个时候就按照普通的包处理上次放置的错误信息包，就避免代码的复用。

这个思路有错误吗？（测试样例不可通过，因为测试样例一定要求先发request，然后触发调用收包函数）