计算机网络实验 Lab4实验报告

工程管理学院 计算机金融实验班 211275032 汪科

Email：211275032@smail.nju.edu.cn

一、实验名称：Forwarding Packets

二、实验目的

接收和转发到达链路并发往其他主机的数据包。转发过程的一部分是在转发表中执行地址查找（“最长前缀匹配”查找）。
对没有已知以太网 MAC 地址的 IP 地址发出 ARP 请求。

三、实验内容

本实验采用了**多线程实现。且（伪）通过了bonus，详见PartVI**

然而，github上的autograding系统采用单线程检测方式，因此提交上去的代码是0分。

PartI.准备工作

我们初始化Router类：

```python
class Router(object):
    def __init__(self, net: switchyard.llnetbase.LLNetBase):
        self.net = net
        self.arp_table = {}
        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.port_list = [intf.name for intf in self.interfaces]
        self.export_interfaces()
        self.forward_table = []  # 每一项也是一个列表，包含匹配IP，子网掩码，下一跳ip，端口号
        self.ArpWaitingList = {}  # 正在发送arp请求对应的IP地址：[时间戳,发送次数]
        self.end_of_test = False

        self.forward_init()
        self.ArpRequestQueue = queue.Queue()
        self.ArpReplyQueue = queue.Queue()
        self.lock = threading.Lock()
        self.t2 = threading.Thread(target=self.arp_handler)
        self.t2.start()
```

其中定义了forward_table，ArpwaitingList等存放必要数据的变量。

在此描述一下路由器转发表的构建，以及下一跳IP地址为0.0.0.0的含义。

路由器转发表构建函数如下：

```python
def forward_init(self):
    for interface in self.interfaces:
        TmpIp = interface.ipaddr
        TmpMask = interface.netmask
        TmpNextHop = '0.0.0.0'
        TmpPort = interface.name
        self.forward_table.append([IPv4Address(int2ipv4(int(IPv4Address(TmpIp)) & int(
            IPv4Address(TmpMask)))), TmpMask, TmpNextHop, TmpPort])
    with open('forwarding_table.txt', 'r') as f:
        lines = f.readlines()
    for line in lines:
        line = line.strip()
        line = line.split(' ')
        self.forward_table.append(
            [IPv4Address(line[0]), IPv4Address(line[1]), IPv4Address(line[2]), line[3]])
    self.forward_table.sort(key=lambda x: IPv4Network(
        str(x[0])+'/'+str(x[1])).prefixlen, reverse=True)
```

首先获取路由器端口的IP地址、子网掩码和端口名，以下一跳IP为0.0.0.0的形式，用列表self.forward_table储存。然后打开文件forwarding_table.txt抄。

这里IP地址都是以IPv4Address格式储存的，**并且按照子网掩码的长度排过序了。这样从头到尾匹配转发表项时就可以达成最长前缀匹配。**

self.forward_query函数是用来进行转发表项匹配的，但是不重要，这里不赘述。

PartII.线程交互

我设置了两个线程，这两个线程在Router类初始化时创建。

```python
self.ArpRequestQueue = queue.Queue()
self.ArpReplyQueue = queue.Queue()
self.lock = threading.Lock()
self.t2 = threading.Thread(target=self.arp_handler)
self.t2.start()
```

这里ArpRequestQueue用于存放需要经过ARP请求才能转发的数据包，ArpReplyQueue用于存放接收到的相应ARP回复包。

主线程和副线程通过这两个Queue和一个实例变量ArpWaitingList（字典，用于存放需要进行ARP请求的IP地址及请求状态）进行通信。

主线程主要做如下事：

- 解包，获取信息。
- 判断包的类型、合法性以及需要执行的操作。
- 如果是需要发送ARP请求的数据包，则将其经过处理后放入等待队列。
- 转发/回复相应的包。

副线程主要做如下事：

- 对需要进行ARP请求的IP地址进行请求。
- 控制上一点中请求超时及重发机制。
- 控制清理请求IP地址超时相应数据包的机制。

具体来说，它们的交互部分如下：

在主线程中，如果一个数据包的下一跳IP地址不在路由器的ARP表中，路由器将为这个IP地址发送一个ARP请求以确定其mac地址。此时，主线程会将这个IP地址写入ArpWaitingList中告诉副线程去请求它，并把这个数据包放入等待队列（改组成等待发送的状态，比如改写IP和mac地址，改动ttl）：

```
if next_hop_ip in self.ArpWaitingList.keys():
    log_info(f'{next_hop_ip}已经在等待队列中了，不用再发送了')
else:
    self.lock.acquire()
    self.ArpWaitingList[next_hop_ip] = [
        time.time()-10, 0]
    self.lock.release()
tmp_packet = WaitingPacket(
    src_ip, dst_ip, src_mac, next_hop_ip, forward_info[3], pkt_ttl-1, packet)
self.ArpRequestQueue.put(
    tmp_packet)  # 待转发数据包放入等待队列
```

而如果主线程收到了一个Arp响应，它就直接放入ArpReplyQueue，让副线程自己处理。

```
forward_info = self.forward_query(
    arp_header.targetprotoaddr)
if forward_info != None and forward_info[3] == ifaceName:
    self.arp_table[arp_header.senderprotoaddr] = [
        arp_header.senderhwaddr, time.time()]
    self.export_arp_table()
self.ArpReplyQueue.put(packet)
```

所以总体来说，是主线程一直在将部分任务交给副线程去处理。关于副线程的机制将在第四部分进行描述。

线程结束的机制如下：

设定一个变量end_of_test，当其值为False时副线程会一直运转，当主线程结束时将这个值设为True，副线程结束，整个程序随之结束。

```
def start(self):
    '''A running daemon of the router.
    Receive packets until the end of time.
    '''
    while True:
        try:
            recv = self.net.recv_packet(timeout=1)
        except NoPackets:
            continue
        except Shutdown:
            break
        self.handle_packet(recv)
    self.end_of_test = True
    self.stop()
```

PartIII.主线程逻辑

着重描述handle_pakcet函数内部的逻辑。

首先尝试获取各部分包头以判断包的类型。

```
timestamp, ifaceName, packet = recv
arp_header = packet.get_header(Arp)
icmp_header = packet.get_header(ICMP)
eth_header = packet.get_header(Ethernet)
udp_header = packet.get_header(UDP)
```

然后根据一些简单的逻辑排除不合法的包：

```
# 路由器只处理具有合法以太网目的地址的包
if eth_header.dst not in self.mac_list and eth_header.dst != 'ff:ff:ff:ff:ff:ff':
    log_info(f'路由器收到了一个以太网目标地址不合法的包，丢弃')
    return

# 路由器不处理带有VLAN标记的包
if packet[Ethernet].ethertype == EtherType.VLAN:
    log_info(f'路由器收到了一个带有VLAN包，丢弃')
    return
```

如果是ARP包：

如果目标IP不属于路由器的端口，则直接扔掉。

接下来根据是请求还是响应分类。

- 如果是请求，则判断该是否正在请求这个包到达端口的mac，如果不是就丢弃。接着更新ARP表，并直接构造ARP reply。

```python
if arp_header.operation == ArpOperation.Request:
    log_info(f'收到了一个ARP请求')
    self.arp_table[arp_header.senderprotoaddr] = [
            arp_header.senderhwaddr, time.time()]
    self.export_arp_table()
    index = self.ip_list.index(arp_header.targetprotoaddr)
    reply_pkt = create_ip_arp_reply(
        self.mac_list[index], arp_header.senderhwaddr, arp_header.targetprotoaddr, arp_header.senderprotoaddr)
    self.net.send_packet(ifaceName, reply_pkt)
    log_info(f'路由器收到了针对自己的一个arp请求，回复了一个ARP包，包内容为{reply_pkt}')
```

- 如果是响应，**首先扔掉目标地址/源地址为广播地址的包**，然后进行同样的判断。接着更新ARP表，并将相应包交给副线程处理。

```python
elif arp_header.operation == ArpOperation.Reply:
    log_info(f'收到了一个ARP回复')
    if (eth_header.dst == 'ff:ff:ff:ff:ff:ff') or (eth_header.src=='ff:ff:ff:ff:ff:ff'):
        log_info(f'ARP reply不应出现广播的以太网地址')
        # 这是一个根据lab3提示的非法包

    else:
        forward_info = self.forward_query(arp_header.targetprotoaddr)
        if forward_info != None and forward_info[3]==ifaceName:
            self.arp_table[arp_header.senderprotoaddr] = [
            arp_header.senderhwaddr, time.time()]
            self.export_arp_table()
        self.ArpReplyQueue.put(packet)

```

如果是IP包：

首先排除发给路由器端口的ICMP包。

在转发表中查找目标IP对应的下一跳IP（如果没找到直接扔），并在ARP表中查找下一跳IP。

关于下一跳IP：

如果转发表中下一跳IP为0.0.0.0，那么设定下一跳地址为目标IP地址。

如果有，直接转发。

```python
log_info('ARP表中有匹配，直接转发')
next_hop_mac = self.arp_table[next_hop_ip][0]
packet[Ethernet].src = self.mac_list[self.port_list.index(
    forward_info[3])]
packet[Ethernet].dst = next_hop_mac
packet[IPv4].ttl -= 1
log_info(f'转发了一个包：{packet}')
self.net.send_packet(forward_info[3], packet)
```

如果没有，则将这个包改装成这样的形式：

```python
class WaitingPacket(object):
    def __init__(self, src_ip, dst_ip, src_mac, next_hop_ip, port, pkt_ttl, my_packet):
        self.src_ip = src_ip
        self.dst_ip = dst_ip
        self.src_mac = src_mac
        self.next_hop_ip = next_hop_ip
        self.port = port
        self.pkt_ttl = pkt_ttl
        self.my_packet = my_packet

    def __str__(self):
        return str(self.my_packet)
```

```python
if self.arp_table.get(next_hop_ip) == None:
    log_info(f'ARP表中没有匹配，发送地址为{next_hop_ip}的arp请求')
    if next_hop_ip in self.ArpWaitingList.keys():
        log_info(f'{next_hop_ip}已经在等待队列中了，不用再发送了')
    else:
        self.lock.acquire()
        self.ArpWaitingList[next_hop_ip] = [
            time.time()-10, 0]
        self.lock.release()
    tmp_packet = WaitingPacket(
        src_ip, dst_ip, src_mac, next_hop_ip, forward_info[3], pkt_ttl-1, packet)
    self.ArpRequestQueue.put(tmp_packet)  # 待转发数据包放入等待队列
```

其中ttl已经减过了，所以副线程如果收到了合适的ARP reply，就可以将这个包略微修改一下直接发过去，这一步在副线程中完成。

PartIV.副线程逻辑

副线程有两个主要函数。

```python
def arp_handler(self):
    # 用于处理发送ARP请求和收到ARP回复的线程
```

这里先从第一个函数开始描述：

```python
def arp_handler(self):
    # 用于处理发送ARP请求和收到ARP回复的线程
    while self.end_of_test==False:
        try:
            ReplyPacket = self.ArpReplyQueue.get(block=False)
        except queue.Empty:
            self.arp_forward_handler()
            continue

        self.lock.acquire()
        arp_header = ReplyPacket.get_header(Arp)
        src_ip = arp_header.senderprotoaddr
        src_mac = arp_header.senderhwaddr
        log_info(f'收到了来自ip地址为{src_ip}的ARP reply，现在开始清理数据包')

        # 这一部分是为了防止收到重复的arp reply，在删除字典时发生报错
        try:
            del self.ArpWaitingList[src_ip]
        except KeyError:
            log_info('收到了重复的ARP reply')
            pass

        for i in range(self.ArpRequestQueue.qsize()):
            tmp_packet = self.ArpRequestQueue.get()
            if tmp_packet.next_hop_ip == src_ip:
                tmp_packet.my_packet[Ethernet].dst = src_mac
                tmp_packet.my_packet[Ethernet].src = self.mac_list[self.port_list.index(
                    tmp_packet.port)]
                tmp_packet.my_packet[IPv4].ttl -= 1
                self.net.send_packet(tmp_packet.port, tmp_packet.my_packet)
                log_info(f'收到ARP reply后路由器发送了一个包，包的内容为{tmp_packet.my_packet},端口为{tmp_packet.port}')

            else:
                self.ArpRequestQueue.put(tmp_packet) # 如果不是目标IP的包，就放回队列

        self.lock.release()
        self.arp_forward_handler()
```

副线程的函数就是这个函数。它由一个while循环控制。

通过高频率的执行arp_forward_handler()，来近似地实现对需要进行ARP请求的IP地址的精确管理。

如果没有通过ArpReplyQueue没有获取到Arp回复包：

在异常中，执行arp_forward_handler()。进行新一轮的循环。

如果获得了一个包：

首先获取这个包的基本信息。然后从ArpWaitingList从删除这个IP地址。最后遍历ArpRequestQueue，把这个IP地址对应的Request数据包都发出去。

在这个过程中，由于涉及到线程不安全的实例变量，所以用锁进行控制，确保安全性。

arp_forward_handler()

```python
def arp_forward_handler(self):
    # 遍历ArpWaitingList，对过期条目进行清算
    AddrToBeDeleted = []
    if self.ArpWaitingList == {}:
        return
    # 获取锁，防止另一线程在遍历过程中修改字典
    self.lock.acquire()
    for key, value in self.ArpWaitingList.items():
        if time.time()-value[0] >= 1.0:
            if value[1] >= 5:
                log_info(f'请求IP为{key}的arp超时，丢弃该IP对应所有数据包')
                for i in range(self.ArpRequestQueue.qsize()):
                    tmp_packet = self.ArpRequestQueue.get(block=False)
                    if tmp_packet.next_hop_ip == key:
                        pass
                    else:
                        self.ArpRequestQueue.put(tmp_packet)
                AddrToBeDeleted.append(key)
                continue
            else:
                self.ArpWaitingList[key][0] = time.time()
                self.ArpWaitingList[key][1] += 1
                forward_info = self.forward_query(key)
                #再次构造ARP请求包
                arp_request_packet = Ethernet(src=self.mac_list[self.port_list.index(forward_info[3])],\
                                              dst='ff:ff:ff:ff:ff:ff', ethertype=EtherType.ARP)+\
                    Arp(
                    operation=ArpOperation.Request, senderhwaddr=self.mac_list[self.port_list.index(forward_info[3])],\
                          senderprotoaddr=self.ip_list[self.port_list.index(forward_info[3])], \
                            targethwaddr='ff:ff:ff:ff:ff:ff', targetprotoaddr=key)
                #发包
                self.net.send_packet(forward_info[3], arp_request_packet)
                log_info(f'刚才发了一个arp请求，目的地址为{key}，端口为{forward_info[3]},次数为{value[1]}内容为{arp_request_packet}')
        else:
            pass

    # 为了防止在遍历过程中删除字典元素，所以最后统一删除
    for addr in AddrToBeDeleted:
        del self.ArpWaitingList[addr]
        log_info(f'因为arp超时，删除了地址{addr}，此时为{time.time()}')
    self.lock.release()
```

ArpWaitingList的结构是：{IPaddr：[timestamp，number]}

遍历ArpWaitingList，当时间超过一秒且已重发次数小于5次时，重置时间戳为当前系统时间，并重发包，发送次数自增1；当时间超过一秒且已重发次数等于5次时，遍历ArpRequestQueue，把相应IP的数据包全部扔掉。

PartV.测试用例的错误

在advanced_srpy中，在这组数据包附近：

```
36  ARP request for 172.16.40.2 should leave from eth3
37  UDP packet to 12.1.1.23 should arrive on eth1
38  Router should not do anything
39  ARP request for 172.16.40.2 should leave from eth3
40  Ping request to 12.1.1.23 should arrive on eth4
41  Router should not do anything
42  ARP request for 172.16.40.2 should leave from eth3
43  UDP packet to 12.1.1.23 should arrive on eth5
44  Router should not do anything
45  ARP request for 172.16.40.2 should leave from eth3
46  Ping request to 12.1.1.23 should arrive on eth4
47  Router should not do anything
48  Ping request to 12.1.1.23 should arrive on eth4
49  ARP request for 172.16.40.2 should leave from eth3
50  Router should not do anything
51  ARP request for 172.16.40.2 should leave from eth3
52  UDP packet to 12.1.1.23 should arrive on eth4
53  UDP packet to 12.1.1.23 should arrive on eth1
```

存在用例的错误。

错误具体为：路由器在34.36.39.42事件中为了转发前面收到的包，向172.16.40.2发送了4次ARP请求。事件45要求路由器发送第五次请求。

当路由器发送第五次请求（事件45）后，1秒内并未受到ARP回复，本应删除地址请求并清理相应数据包。但事件46.48是两个需要路由器请求相同地址的ICMP包，它们与事件45两两之间的间隔在$10^{-3}s$数量级。

然而后续的测试用例期望路由器在接收到数据包46后删除该IP对应的所有数据包，但保留数据包47，并为了它在事件49中发送新一轮ARP包。这是不符合路由器的ARP重发逻辑的。

但是为了通过测试，在主线程ICMP包处理中对其进行了特判：

```python
if next_hop_ip==IPv4Address('172.16.40.2') and self.ArpWaitingList[next_hop_ip][1]==5:
    self.ArpWaitingList[next_hop_ip][0]-=1
    log_info('这是一个特判，测试用例有问题')
```

作用是让路由器在收到数据包46后改动时间戳，立即删除对应等待表项和所有数据包。

PartVI.Bonus

主要针对最后两个bonus进行阐述。

1208：

不该转发包的原因是IPv4包头中的total_length参数不符合实际。

```
dscp:0
dst:172.32.0.187
ecn:0
flags:0
fragment_offset:0
hl:5
ipid:0
options:IPOptionList ()
protocol:17
src:15.16.0.199
tos:0
total_length:34
ttl:17
10:43:23 2023/05/04     INFO ARP表中有匹配，直接转发
10:43:23 2023/05/04     INFO 转发了一个包：Ethernet 20:00:00:00:00:05->30:00:00:00:05:02 IP | I
Pv4 15.16.0.199->172.32.0.187 UDP | UDP 10000->10000 | RawPacketContents (6 bytes) b'test90'
10:43:23 2023/05/04     INFO 收到了一个包，系统当前时间戳为1683222203.8968644
10:43:23 2023/05/04     INFO 收到一个IP包，内容为Ethernet 30:00:00:00:05:02->20:00:00:00:00:05
IP | IPv4 15.0.0.92->192.168.129.2 UDP | UDP 0->0 | RawPacketContents (36 bytes) b'\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00'..., 到达端口为eth5
10:43:23 2023/05/04     INFO 收到包头长度错误包
```

这里IPv4包头长度为20，UDP包头长度8，载荷长度36，加起来显然不等于34。因此这个数据包的包头存在错误，不转发。

因此，在这个包为ICMP/UDP包时加入判断：

```
IPv4length=packet[IPv4].total_length
if 14 + IPv4length != packet.size():
    log_info("收到包头长度错误包\n")
    return
```

即可通过。

1209：

```
dscp:0
dst:192.168.129.2
ecn:1
flags:0
fragment_offset:0
hl:5
ipid:0
options:IPOptionList ()
protocol:17
src:15.0.0.92
tos:1
total_length:63
ttl:64
10:43:23 2023/05/04     INFO ARP表中有匹配，直接转发
10:43:23 2023/05/04     INFO 转发了一个包：Ethernet 20:00:00:00:00:06->30:00:00:00:06:02 IP | I
Pv4 15.0.0.92->192.168.129.2 UDP | UDP 0->0 | RawPacketContents (35 bytes) b'\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00'...
```

这个包的特殊在于其ECN值为1，是显式拥塞通知的标志位。

实际上是要转发的。但是助教gg说忘弄了，所以这里摘一下DSCP和ECN位的描述。

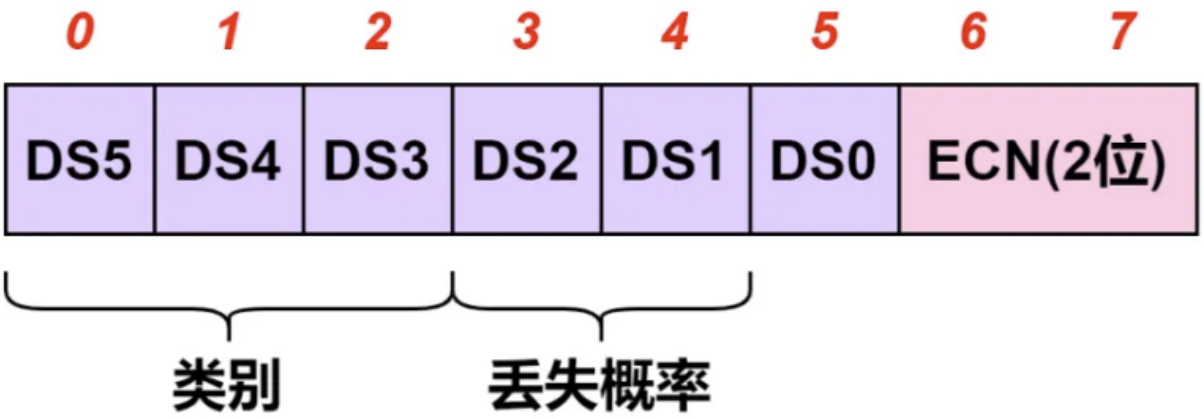
这是IPv4包头结构：

IPv4 header format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
⋮	⋮																																
60	480																																

其中DSCP和ECN位的结构：

DSCP（Differential Services Codepoint，差分服务代码点）是TOS（Type Of Service）的一部分。现在统称为DiffServ，用来进行质量控制。



如果3~5位的值为0，0~2位则被称作类别选择代码点。这样就可以像TOS的优先度那样提供8种类型的质量控级别。对于每一种级别所采用的措施则由提供DiffServ的运营管理者制定。为了与TOS保持一致，值越大优先度越高。

名称	值	参考文献	描述
CS0	000000	[RFC2474]	类别选择(尽力而为/常规)
CSl	001000	[RFC2474]	类别选择(优先)
CS2	010000	[RFC2474]	类别选择(立即)
CS3	011000	[RFC2474]	类别选择(瞬间)
CS4	100000	[RFC2474]	类别选择(瞬间覆盖)
CS5	101000	[RFC2474]	类别选择(CRITIC/ECP)
CS6	110000	[RFC2474]	类别选择(网间控制)
CS7	111000	[RFC2474]	类别选择(控制)
AF11	001010	[RFC2597]	保证转发(1,1)
AF12	001100	[RFC2597]	保证转发(1,2)
AF13	001110	[RFC2597]	保证转发(1,3)
AF21	010010	[RFC2597]	保证转发(2,1)
AF22	010100	[RFC2597]	保证转发(2,2)
AF23	010110	[RFC2597]	保证转发(2,3)
AF31	011010	[RFC2597]	保证转发(3,1)
AF32	011100	[RFC2597]	保证转发(3,2)
AF33	011110	[RFC2597]	保证转发(3,3)
AF41	100010	[RFC2597]	保证转发(4,1)
AF42	100100	[RFC2597]	保证转发(4,2)
AF43	100110	[RFC2597]	保证转发(4,3)
EF PHB	101110	[RFC3246]	加速转发
VOICE-ADMIT	101100	[RFC5865]	容量许可的流量

ECN（Explicit Congestion Notification，显式拥塞通告）用来报告网络拥堵情况，由两个比特构成。

比特	简称	含义
6	ECT	ECN-Capable Transport
7	CE	Congenstion Experienced

第6位的ECT用以**通告上层TCP层协议**是否处理ECN。当路由器在转发ECN为1的包过程中，如果出现网络拥堵的情况，就将第7位CE位设置为1。

四、实验结果

测试用例

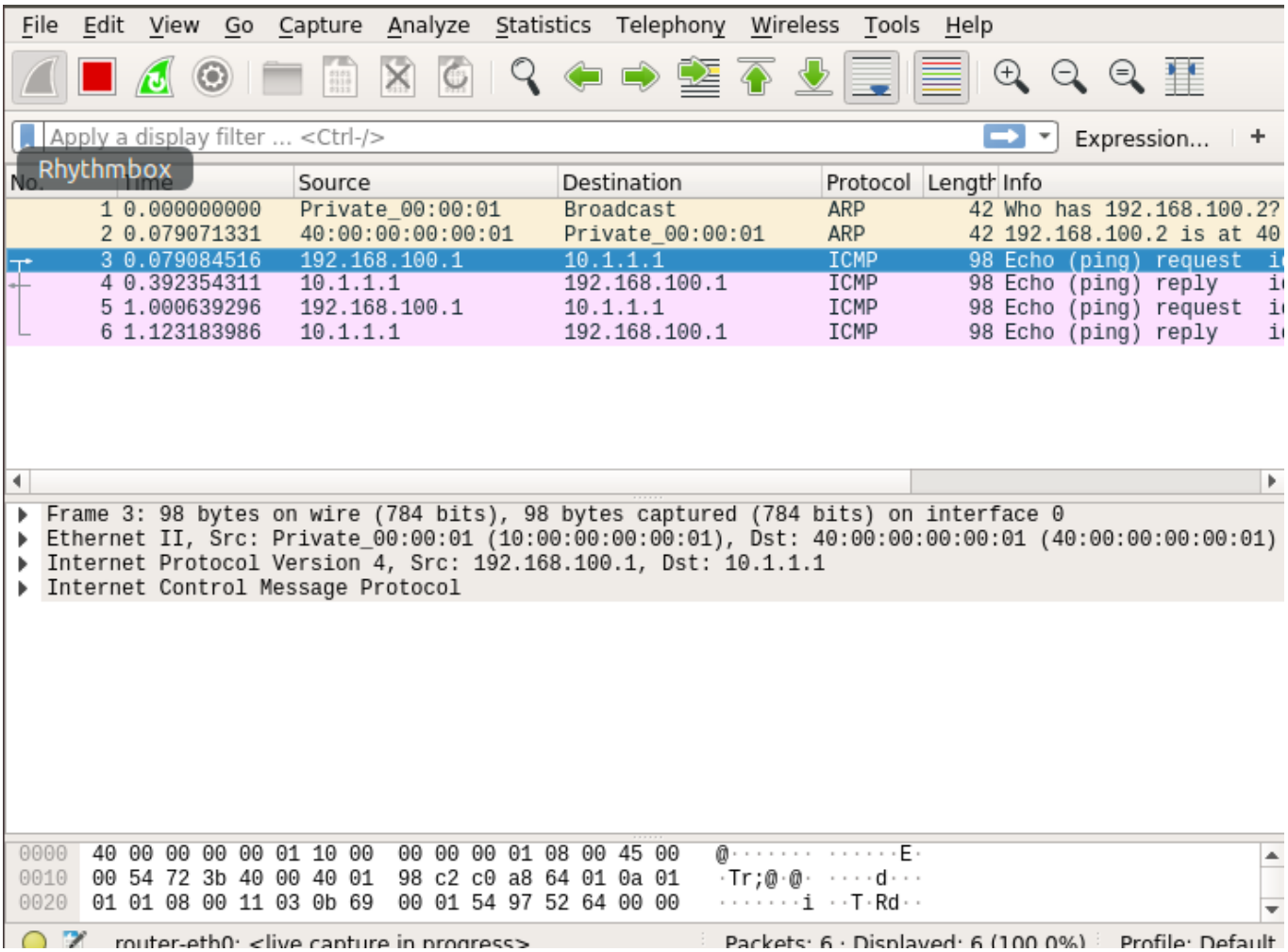
```
26  Router should try to receive a packet (ARP response), but
    then timeout
27  Router should send an ARP request for 10.10.50.250 on
    router-eth1
28  Router should try to receive a packet (ARP response), but
    then timeout
29  Router should send an ARP request for 10.10.50.250 on
    router-eth1
30  Router should try to receive a packet (ARP response), but
    then timeout
31  Router should try to receive a packet (ARP response), but
    then timeout


All tests passed!


(threadsyenv) dk@ubuntu:~/Desktop/workspace/Lab-4-Wangke/lab-4-atom-tracer$
```

```
1208Bonus: V2h1dCBkJyB5YSBob3BlIHQnIGZpbmQgaGVyZT8=
1209Bonus: Tm90aGluJyBmb3IgeWEgdCcgZmluZCBoZXJlIQ==


Failed:
    Q29uZ3JhdHMh
        Expected event: Timeout after 1.2s on a call to recv_packet


**********************************************************
Your code didn't crash, but a test failed.
**********************************************************

This is the Switchyard equivalent of the blue screen of death.
As far as I can tell, here's what happened:

    Expected event:
        Q29uZ3JhdHMh

    Failure observed:
        send_packet was called but no sending is expected.
```

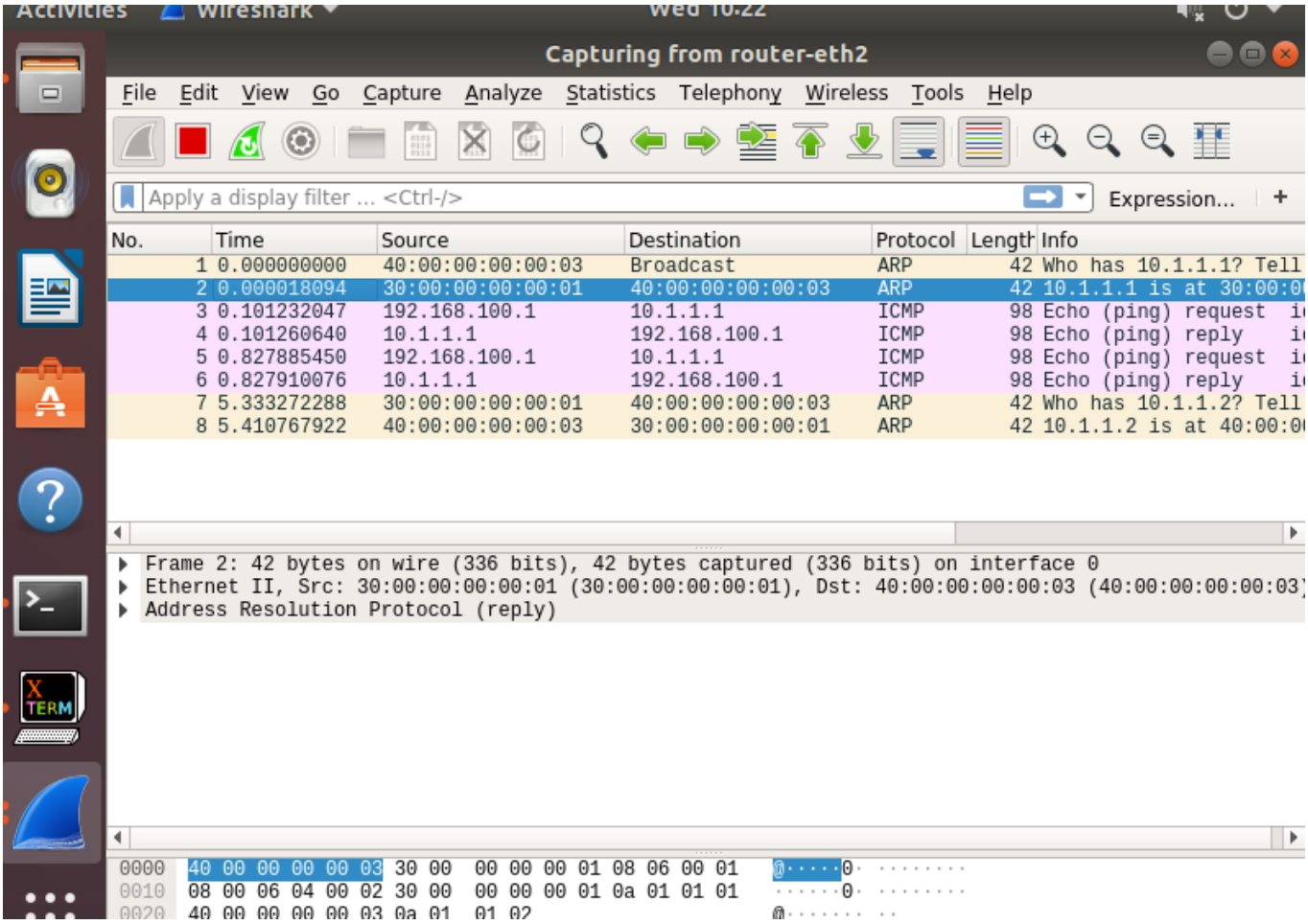
部署

命令：`server1 ping -c2 client`

监听端口router-eth0（与server1相连）：



监听端口router-eth2（与client相连）：



解释：在eth0中可以看到，server1先问了router-eth的mac地址，然后发送了ICMP包。在它发送第一个ICMP包时，从eth2中看到，路由器询问了client的mac地址，随后转发了这个ICMP包。第二个ICMP包因为有ARP表记录的存在，直接转发即可。

五、总结与感想

- 实验非常难，一开始转发逻辑都没理清，后来重新梳理了一遍，总共用时接近30h。
- 多线程设计资源访问问题，需要控制。在这个项目中我学会了使用基本的线程安全数据结构-Queue和锁来控制资源访问。
- 对数据包和路由器的转发逻辑的认识更上一层楼（如VLAN的含义与作用等）。