计算机网络实验 Lab3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：Respond to ARP

二、实验目的：实现对ARP请求的响应

三、实验内容

Task1：处理ARP请求

程序开始时运行Router类的run函数，进入接收包循环。如果包正常，我们运行`self.handle_packet()`。我们主要修改这个函数。

我们需要小小地修改一下`__init__`函数：

```
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]
```

这里我们构建了task2中用到的ARP表，以及获取并存储了Router的各端口ip地址和mac地址。

然后进行ARP包的基本处理。

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    timestamp, ifaceName, packet = recv
    arp=packet.get_header(Arp)
    if not arp:
        return
```

这里我们用`arp=packet.get_header(Arp)`解包。我们暂且只处理arp包，所以对非arp包我们直接return。

接下来执行如下代码：

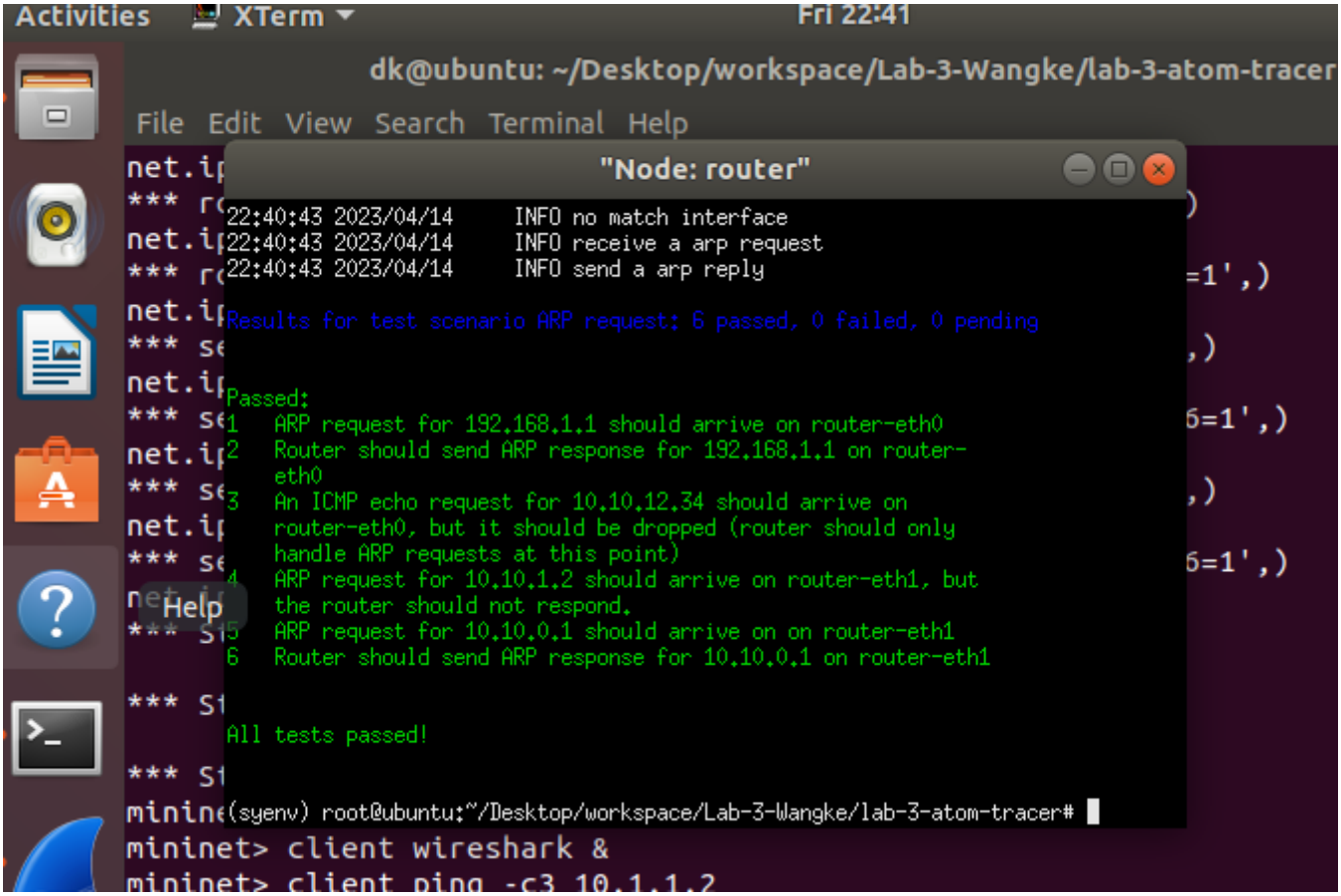
```python
#nodify the arp_table
for ip_addr in list(self.arp_table.keys()):
    if timestamp - self.arp_table[ip_addr][1] >=100.0:
        del self.arp_table[ip_addr]
#modify the arp_table

if arp.targetprotoaddr in self.ip_list:
    self.arp_table[arp.senderprotoaddr] = [arp.senderhwaddr,timestamp]
    self.export_arp_table()
    if arp.operation==ArpOperation.Request:
        index=self.ip_list.index(arp.targetprotoaddr)
        reply_pkt=create_ip_arp_reply(self.mac_list[index],arp.senderhwaddr,arp.targetprotoaddr,arp.senderprotoaddr)
        self.net.send_packet(ifaceName,reply_pkt)
        log_info("send a arp reply")
    else:
        log_info("this is not a arp request")
else:
    log_info("no match interface")
```

arp.operation代表了这个ARP包的操作状态，也就是分辨请求ARP和响应ARP包。

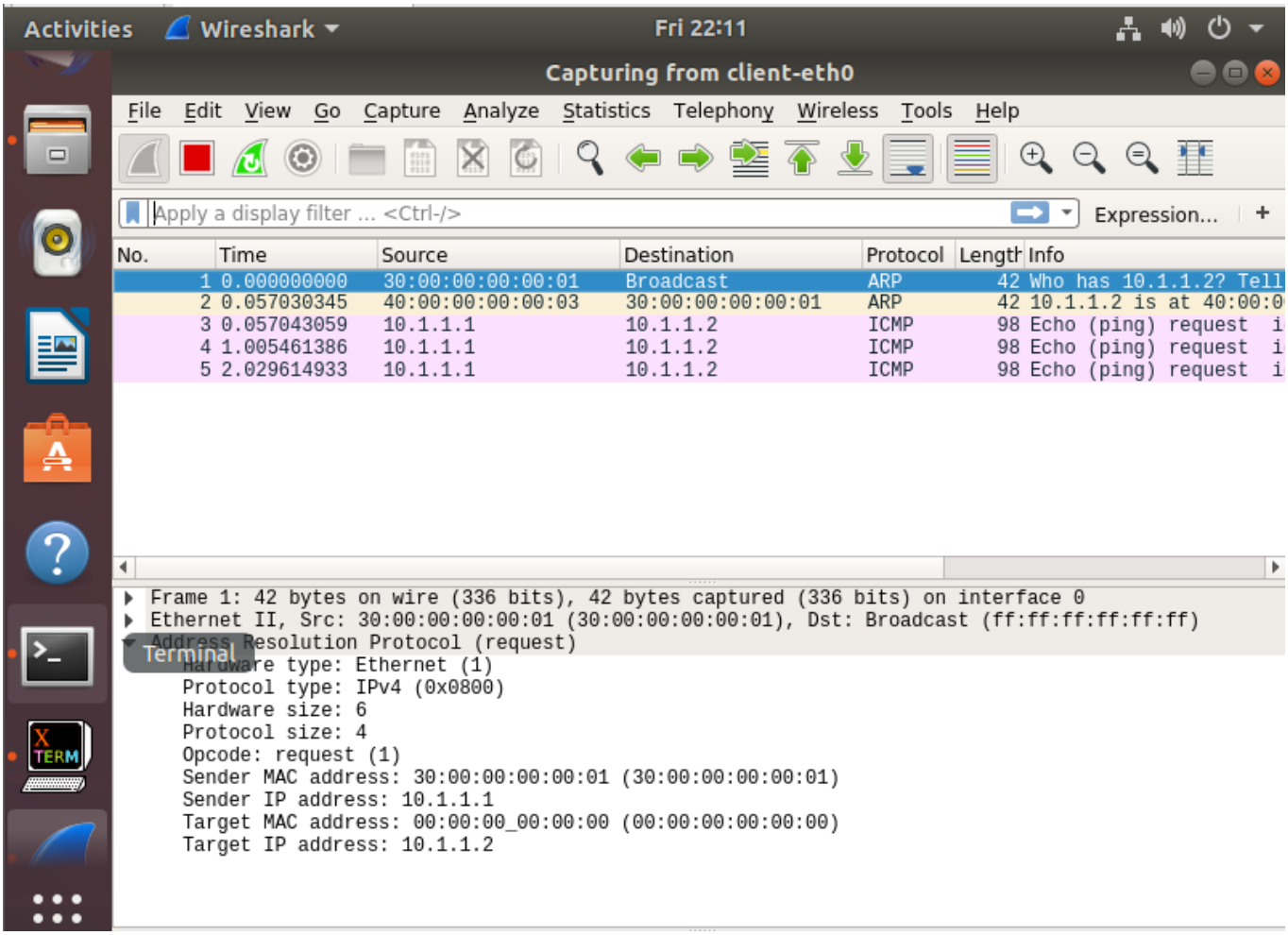
我们检测目标ip是否是Router端口的ip。

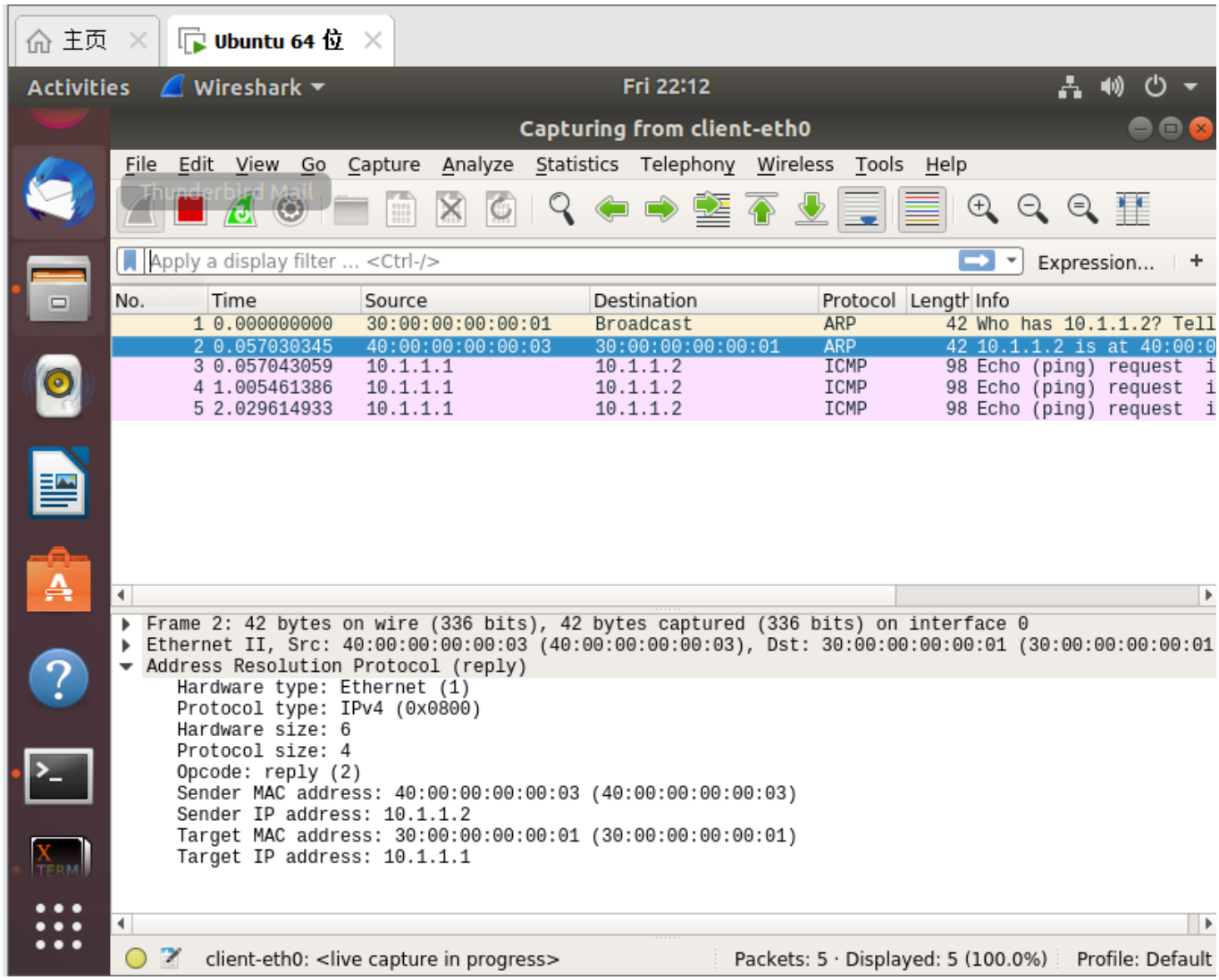
- 如果是，更新ARP表。同时，如果是ARP请求包，则构造一个ARP响应包并填充请求ip对应的端口的mac地址，然后发回原端口；如果不是ARP请求包，那么打印错误信息。
- 如果不是，那么直接打印一个未匹配信息。

testcase测试结果如下：



使用wireshark抓包，执行`client ping -c3 10.1.1.2`命令，在client上抓包结果如下：





可以看出，第一个ARP的target mac addr是空的，而Router构造的ARP响应包已经填充了请求的ip对应端口的mac地址。

Task2：缓存的ARP表

我们构造实例变量`self.arp_table{}`来存储ARP表，它的key是ip_addr，value是列表[mac_addr, timestamp]。每接收到一个ARP包，执行：

```
#nodify the arp_table
for ip_addr in list(self.arp_table.keys()):
    if timestamp - self.arp_table[ip_addr][1] >=100.0:
        del self.arp_table[ip_addr]
#modify the arp_table
```

这里用之前获取的timestamp当作当前时间，检测ARP表中所有的表项，将存在时间超过100s的表项删除。

最后一行是执行了自行定义的打印ARP表的函数：

```
def export_arp_table(self):
    with open('arp_table.txt','a') as f:
        f.write(str(self.arp_table)+'\n')
```

ARP表的更新已经在task1中阐明。

执行testcases后，arp_table的打印结果是：

arp_table.txt

```
{IPv4Address('192.168.1.100'): [EthAddr('30:00:00:00:00:01'), 0.0]}
{IPv4Address('192.168.1.100'): [EthAddr('30:00:00:00:00:01'), 0.0], IPv4Address('10.10.5.5'): [EthAddr('70:00:ca:fe:c0:de'), 1.0]}

```

执行`client ping -c3 10.1.1.2`后的打印结果：

```
{IPv4Address('10.1.1.1'): [EthAddr('30:00:00:00:00:01'), 1681538333.345572]}

```

四、实验结果

- 实现了Router处理ARP请求的功能
- 在Router上实现了ARP表缓存机制

五、总结与感想

- 实验简单易操作，注意可复用前面的代码。