

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

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

Task2: 缓存的ARP表

四、实验结果

五、总结与感想

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：

```
1 class Router(object):
2     def __init__(self, net: switchyard.llnetbase.LLNetBase):
3         self.net = net
4         self.arp_table = {}
5         self.interfaces=net.interfaces()
6         self.ip_list=[intf.ipaddr for intf in self.interfaces]
7         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。

```
1 def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
2     timestamp, ifaceName, packet = recv
3     arp=packet.get_header(Arp)
4     if not arp:
5         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：

```

1 #modify the arp_table
2 for ip_addr in list(self.arp_table.keys()):
3     if timestamp - self.arp_table[ip_addr][1] >=100.0:
4         del self.arp_table[ip_addr]
5 #modify the arp_table
6
7 if arp.targetprotoaddr in self.ip_list:
8     self.arp_table[arp.senderprotoaddr] = [arp.senderhwaddr,timestamp]
9     self.export_arp_table()
10 if arp.operation==ArpOperation.Request:
11     index=self.ip_list.index(arp.targetprotoaddr)
12     reply_pkt=create_ip_arp_reply(self.mac_list[index],arp.senderhwaddr,arp.targetprotoaddr,arp.senderprotoaddr)
13     self.net.send_packet(ifaceName,reply_pkt)
14     log_info("send a arp reply")
15 else:
16     log_info("this is not a arp request")
17 else:
18     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：

```

Fri 22:41
dk@ubuntu: ~/Desktop/workspace/Lab-3-Wangke/lab-3-atom-tracer
File Edit View Search Terminal Help

*** 22:40:43 2023/04/14 INFO no match interface
*** 22:40:43 2023/04/14 INFO receive a arp request
*** 22:40:43 2023/04/14 INFO send a arp reply

Results for test scenario ARP request: 6 passed, 0 failed, 0 pending

Passed:
1 ARP request for 192.168.1.1 should arrive on router-eth0
2 Router should send ARP response for 192.168.1.1 on router-eth0
3 An ICMP echo request for 10.10.12.34 should arrive on router-eth0, but it should be dropped (router should only handle ARP requests at this point)
4 ARP request for 10.10.1.2 should arrive on router-eth1, but the router should not respond.
5 ARP request for 10.10.0.1 should arrive on on router-eth1
6 Router should send ARP response for 10.10.0.1 on router-eth1

*** S1 All tests passed!
*** S1

mininet(syenv) root@ubuntu:~/Desktop/workspace/Lab-3-Wangke/lab-3-atom-tracer#
mininet> client wireshark &
mininet> client ping -c3 10.1.1.2

```

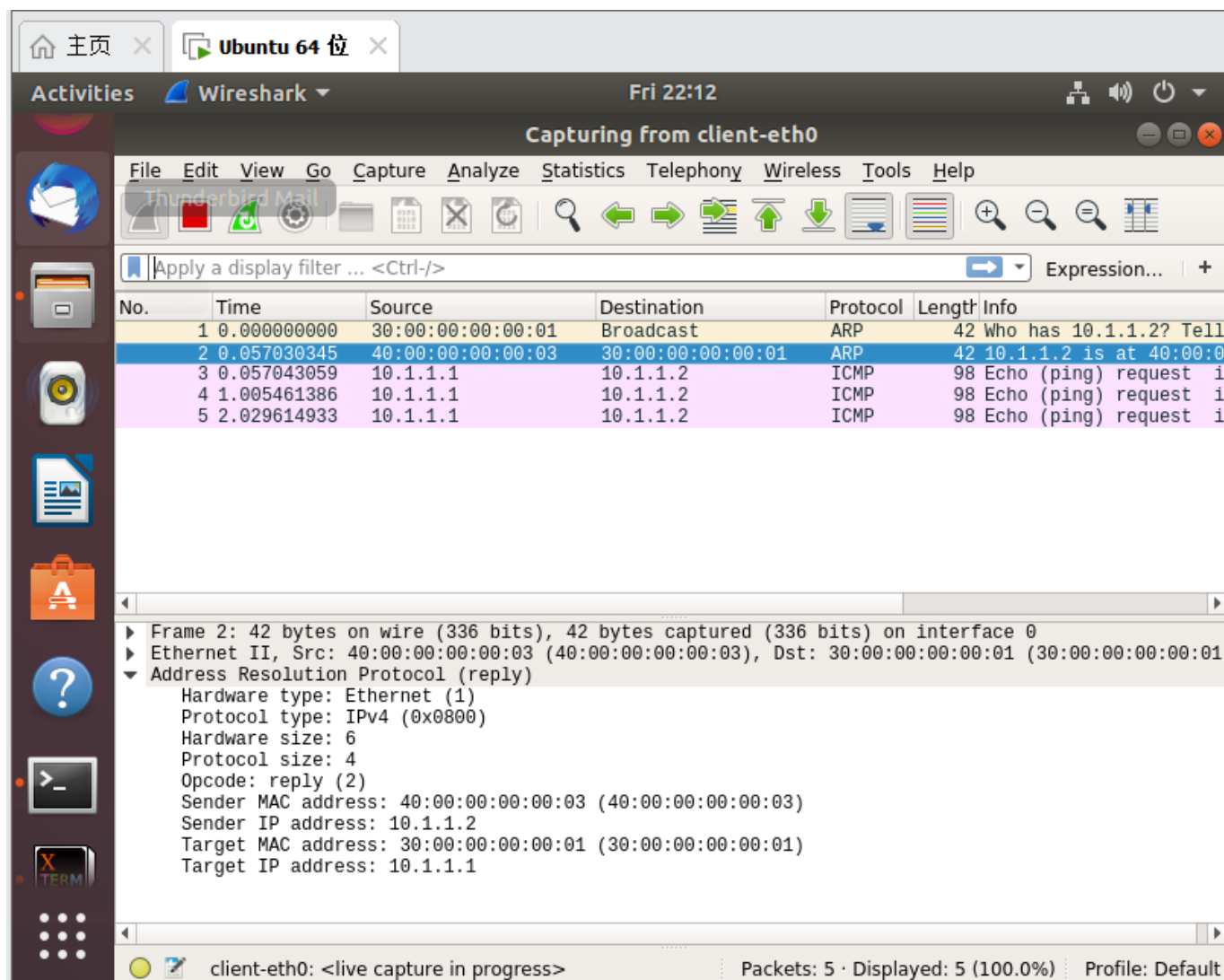
使用wireshark抓包，执行client ping -c3 10.1.1.2命令，在client上抓包结果如下：

The image shows the Wireshark network traffic capture interface. The top bar indicates the time is Fri 22:11. The main window title is "Capturing from client-eth0". The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for file operations, capture control, and analysis. A display filter bar shows "Apply a display filter ... <Ctrl-/>". The packet list table shows five captured packets:

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
2	0.057030345	40:00:00:00:00:03	30:00:00:00:00:01	ARP	42	10.1.1.2 is at 40:00:0
3	0.057043059	10.1.1.1	10.1.1.2	ICMP	98	Echo (ping) request i
4	1.005461386	10.1.1.1	10.1.1.2	ICMP	98	Echo (ping) request i
5	2.029614933	10.1.1.1	10.1.1.2	ICMP	98	Echo (ping) request i

The packet details pane for the selected packet (Frame 1) shows the following information:

- 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)
- Hardware type: Ethernet (1)
- Protocol type: IPv4 (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: request (1)
- Sender MAC address: 30:00:00:00:00:01 (30:00:00:00:00:01)
- Sender IP address: 10.1.1.1
- Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
- Target IP address: 10.1.1.2



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：



```
1 #nodify the arp_table
2 for ip_addr in list(self.arp_table.keys()):
3     if timestamp - self.arp_table[ip_addr][1] >=100.0:
4         del self.arp_table[ip_addr]
5 #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：



```
1 def export_arp_table(self):
2     with open('arp_table.txt','a') as f:
3         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

```
1 {IPv4Address('192.168.1.100'): [EthAddr('30:00:00:00:00:01'), 0.0]}
2 {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]}
3
```

执行client ping -c3 10.1.1.2后的打印结果：

```
1 {IPv4Address('10.1.1.1'): [EthAddr('30:00:00:00:00:01'), 1681538333.345572]}
2
```

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。