实验报告

计算机网络-Lab3

专业：计算机科学与技术

任课教师：田臣

周迅 201220037

目录

一、 实验目的

模拟 IPv4 路由器，实现其基本功能的一部分，它仅能够对一个 ARP 请求作出回应，以及维护一个 ARP 缓存表。

二、 实验内容

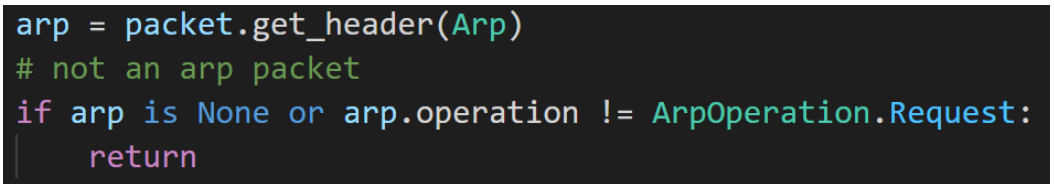
1. Preparation

重要的事情说三遍，熟读 Switchyard 手册，熟读 Switchyard 手册，熟读 Switchyard 手册……

2. Handle ARP Request

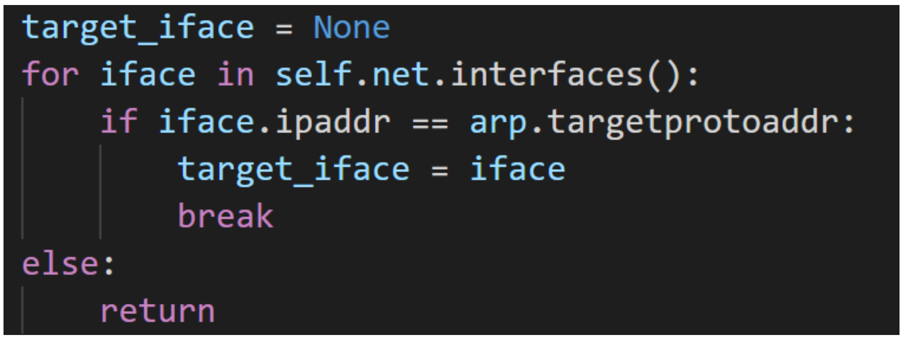
1) Coding

在经过第一阶段非常充分的准备后呢（并没有），我们开始构造我们的路由器。在本次实验中，我们只处理 ARP Request。凡是非 ARP 的（没有 ARP header），扔！凡是非 Request 的，扔！



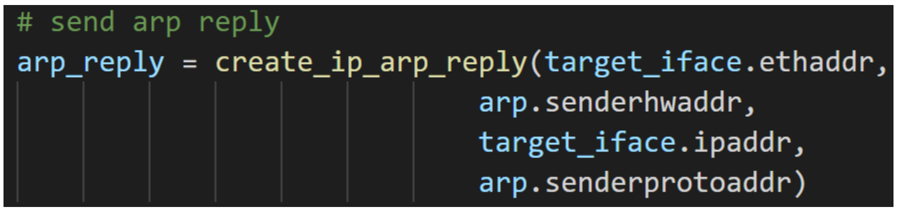
```python
arp = packet.get_header(Arp)
# not an arp packet
if arp is None or arp.operation != ArpOperation.Request:
    return
```

一个 ARP 请求中，目标的以太网地址是不知道的，我们就要根据在 header 中给出的目标的 IP，搜索路由器的端口——端口信息中，同时包含着 IP 和 MAC！找不到？扔！



```python
target_iface = None
for iface in self.net.interfaces():
    if iface.ipaddr == arp.targetprotoaddr:
        target_iface = iface
        break
else:
    return
```

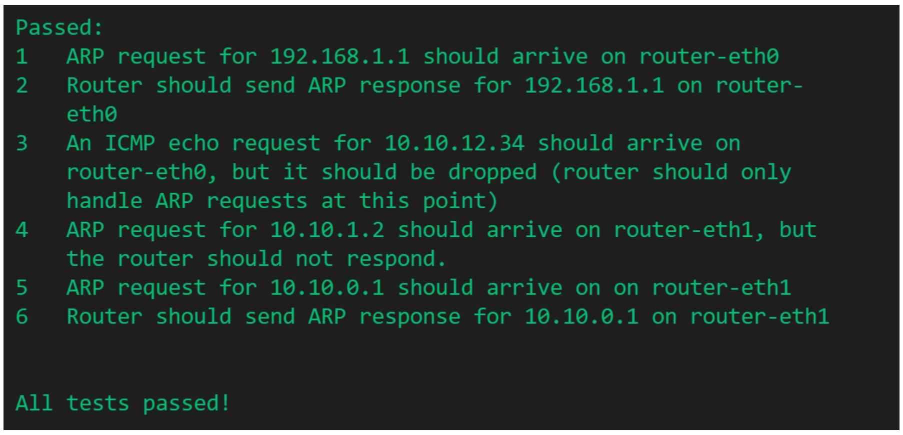
信息填充好之后，我们要原路返回告诉询问者：欸，你说的那哥们儿，就是那个 IP 是 xxx 的，它的 MAC 是 yyy。所以，在 `create_ip_arp_reply` 中，我们得把 src 和 dst 倒过来（相对于 request）。



```python
# send arp reply
arp_reply = create_ip_arp_reply(target_iface.ethaddr,
                            arp.senderhwaddr,
                            target_iface.ipaddr,
                            arp.senderprotoaddr)
```

2) Testing

嗯……好像这部分的代码就结束了。那我们来验证一下。首先当然是备受关注的官方测试集。



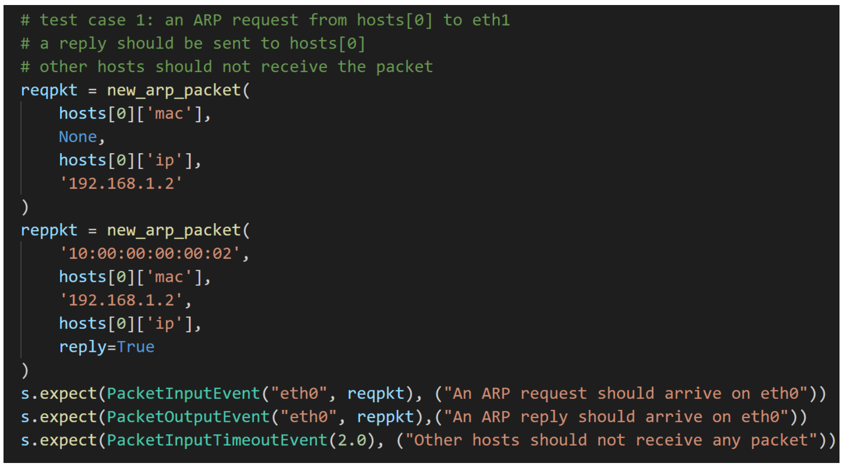
```
Passed:
1   ARP request for 192.168.1.1 should arrive on router-eth0
2   Router should send ARP response for 192.168.1.1 on router-
    eth0
3   An ICMP echo request for 10.10.12.34 should arrive on
    router-eth0, but it should be dropped (router should only
    handle ARP requests at this point)
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


All tests passed!
```

接着，我们自己来几个样例试试。我们创建了有三个样例，作用如下：

- 发送一个 ARP 请求，一个回应应当原路返回，其他端口无反应；
- 发送一个非 ARP，该 packet 应当被丢弃；

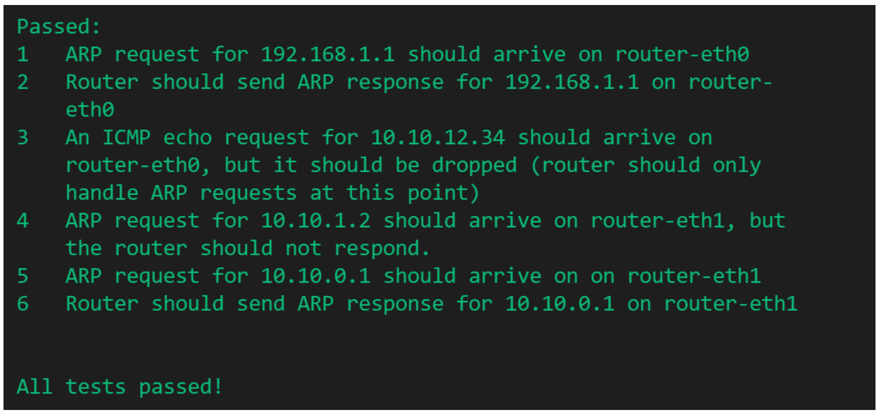
- 发送一个 ARP 回应，该 packet 应当被丢弃。

详细测试代码见 Github，下面为部分代码截图。



```
# test case 1: an ARP request from hosts[0] to eth1
# a reply should be sent to hosts[0]
# other hosts should not receive the packet
reqpkt = new_arp_packet(
    hosts[0]['mac'],
    None,
    hosts[0]['ip'],
    '192.168.1.2'
)
reppkt = new_arp_packet(
    '10:00:00:00:00:02',
    hosts[0]['mac'],
    '192.168.1.2',
    hosts[0]['ip'],
    reply=True
)
s.expect(PacketInputEvent("eth0", reqpkt), ("An ARP request should arrive on eth0"))
s.expect(PacketOutputEvent("eth0", reppkt),("An ARP reply should arrive on eth0"))
s.expect(PacketInputTimeoutEvent(2.0), ("Other hosts should not receive any packet"))
```

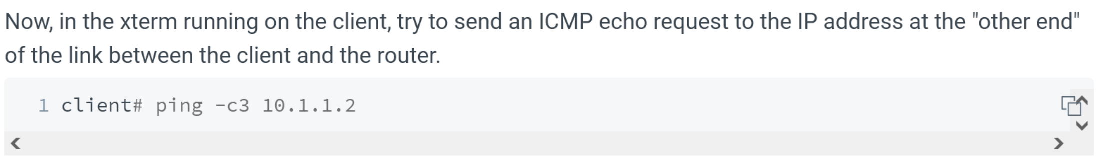
测试结果。



```
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


All tests passed!
```

3) Deploying

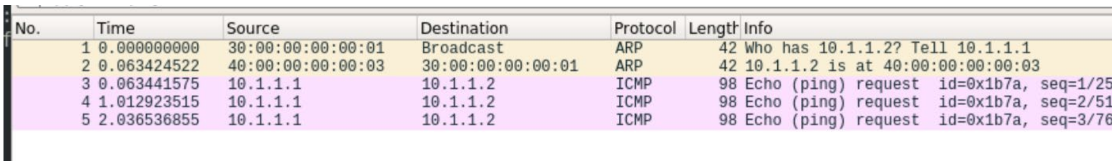
按照手册，我们 ping 一下 client 连接的路由器端口。



Now, in the xterm running on the client, try to send an ICMP echo request to the IP address at the "other end" of the link between the client and the router.

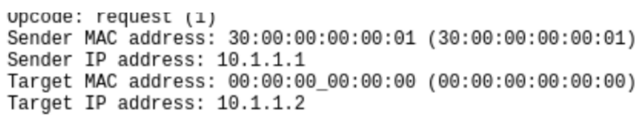
```
client# ping -c3 10.1.1.2
```

按照我们的逻辑，对于 client 的 ARP request，我们填上对应的 mac，然后发回去；而对于 ICMP echo request，则是毫无反应。

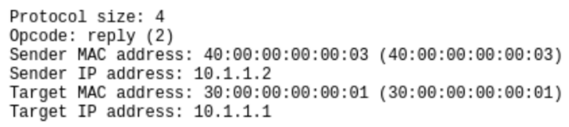


```
No.   Time          Source              Destination         Protocol Length Info
    1 0.000000000   30:00:00:00:00:01   Broadcast           ARP         42 Who has 10.1.1.2? Tell 10.1.1.1
    2 0.063424522   40:00:00:00:00:03   30:00:00:00:00:01   ARP         42 10.1.1.2 is at 40:00:00:00:00:03
    3 0.063441575   10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x1b7a, seq=1/25
    4 1.012923515   10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x1b7a, seq=2/51
    5 2.036536855   10.1.1.1            10.1.1.2            ICMP        98 Echo (ping) request  id=0x1b7a, seq=3/76
```

ARP request 的头中，目标的 mac 全 0（缺失）；而在回应的时候，已经全部补上了。

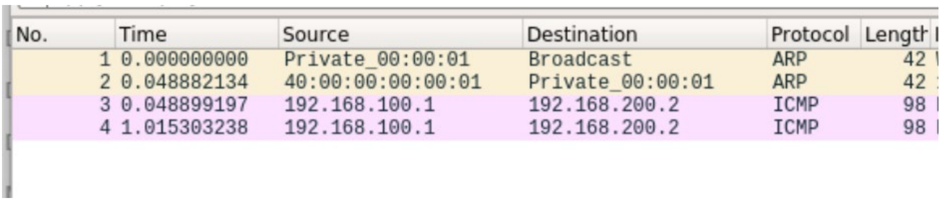


```
Opcode: request (1)
Sender MAC address: 30:00:00:00:00:01 (30:00:00:00:00:01)
Sender IP address: 10.1.1.1
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 10.1.1.2
```



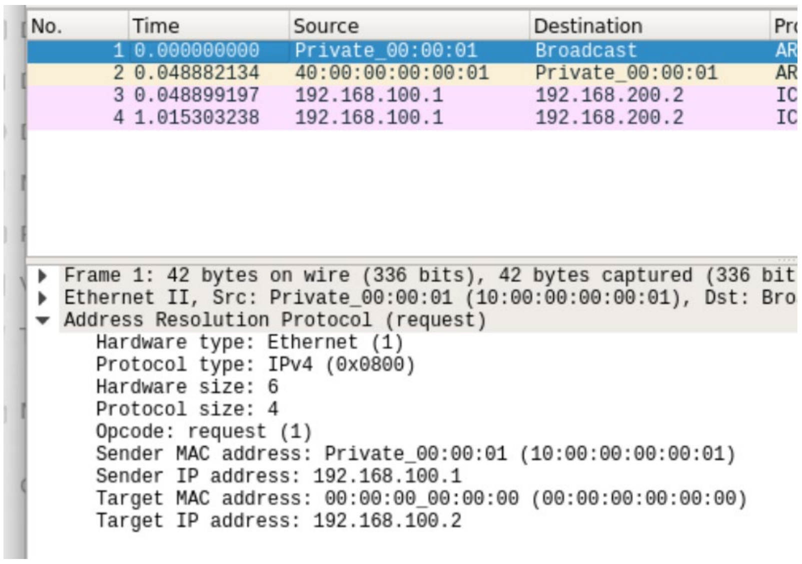
```
Protocol size: 4
Opcode: reply (2)
Sender MAC address: 40:00:00:00:00:03 (40:00:00:00:00:03)
Sender IP address: 10.1.1.2
Target MAC address: 30:00:00:00:00:01 (30:00:00:00:00:01)
Target IP address: 10.1.1.1
```

接下来，我们尝试使用 server1，做类似的操作，尝试获取 server2 连接路由器的那个端口。我们操作 `server1 ping -c2 192.168.200.2`，使用 wireshark 对 server1 抓包如下：

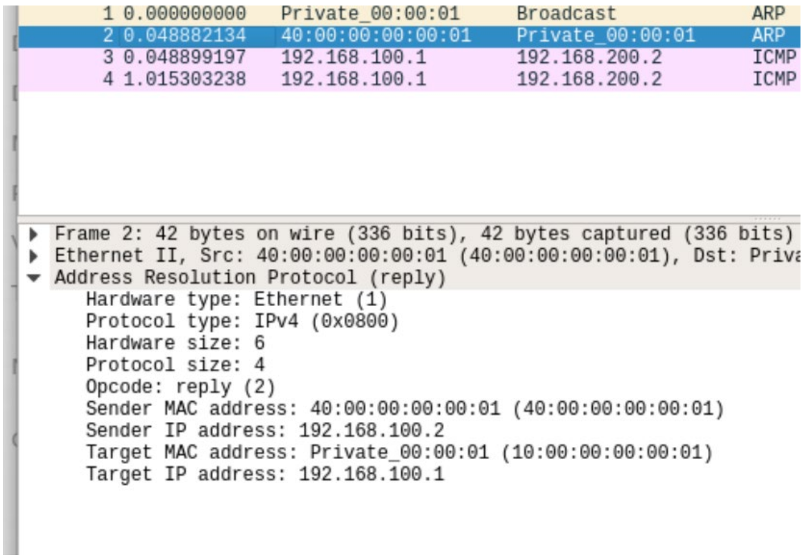


```
No.    Time           Source               Destination          Protocol  Length
     1 0.000000000    Private_00:00:01     Broadcast            ARP           42
     2 0.048882134    40:00:00:00:00:01    Private_00:00:01     ARP           42
     3 0.048899197    192.168.100.1        192.168.200.2        ICMP          98
     4 1.015303238    192.168.100.1        192.168.200.2        ICMP          98
```

非常好！我们的路由器机智地把正确的 mac 填入了相应的字段！



```
No.    Time           Source               Destination          Pro
     1 0.000000000    Private_00:00:01     Broadcast            AR
     2 0.048882134    40:00:00:00:00:01    Private_00:00:01     AR
     3 0.048899197    192.168.100.1        192.168.200.2        IC
     4 1.015303238    192.168.100.1        192.168.200.2        IC

▶ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bit
▶ Ethernet II, Src: Private_00:00:01 (10:00:00:00:00:01), Dst: Bro
▼ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Private_00:00:01 (10:00:00:00:00:01)
    Sender IP address: 192.168.100.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.100.2
```



```
1 0.000000000   Private_00:00:01     Broadcast          ARP
2 0.048882134   40:00:00:00:00:01    Private_00:00:01   ARP
3 0.048899197   192.168.100.1        192.168.200.2      ICMP
4 1.015303238   192.168.100.1        192.168.200.2      ICMP

▸ Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▸ Ethernet II, Src: 40:00:00:00:00:01 (40:00:00:00:00:01), Dst: Priva
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 40:00:00:00:00:01 (40:00:00:00:00:01)
    Sender IP address: 192.168.100.2
    Target MAC address: Private_00:00:01 (10:00:00:00:00:01)
    Target IP address: 192.168.100.1
```

3. Cached ARP Table

查询表，最直观的实现自然就是字典。

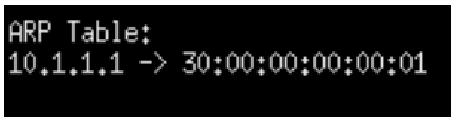
When the router receives a packet with ARP header, add or update an entry of the cached ARP table. For example, if there is an ARP request with the Ethernet source address `01:02:03:04:05:06` and the IP source address `10.1.2.3`, the router will *update* the entry whose key is `10.1.2.3` with the value `01:02:03:04:05:06`. You can also see here that the IP address is **unique** in the table.

对于每个到来的、含有 ARP header 的分组，针对其 src addr 更新 ARP Table。

```
self.arp_tab[arp.senderhwaddr] = arp.senderprotoaddr
```

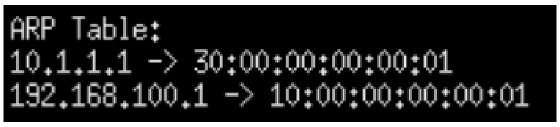
我们放在 Mininet，做个简单的验证。

首先 `client ping -c3 10.1.1.2`，如图，表中出现了 client 的 ip 与 mac 的映射关系。



```
ARP Table:
10.1.1.1 -> 30:00:00:00:00:01
```

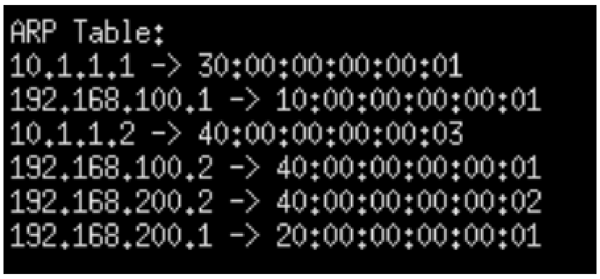
接着 `server1 ping -c2 192.168.200.2`，添加了一个表项。



```
ARP Table:
10.1.1.1 -> 30:00:00:00:00:01
192.168.100.1 -> 10:00:00:00:00:01
```

最后 `pingall`，表中无重复表项，且对应关系正确，3 个 host 加 router 的 3 个

口，共计 6 条记录。



```
ARP Table:
10.1.1.1 -> 30:00:00:00:00:01
192.168.100.1 -> 10:00:00:00:00:01
10.1.1.2 -> 40:00:00:00:00:03
192.168.100.2 -> 40:00:00:00:00:01
192.168.200.2 -> 40:00:00:00:00:02
192.168.200.1 -> 20:00:00:00:00:01
```

三、 核心代码

要不您往上边翻一翻，都有截图；或者，上 Github 看看代码？

四、 总结与感想

- 实验相对简单，重点在于对 Switchyard 的使用（具体来说，是对阅读手册能力的考察）；
- 对于自己功能的验证也是必不可少的一环（在不少情况下，这可能比实现功能还要难一些）；
- ARP 对于连接链路层和网络层的意义可以在此感受到。