LAB3实验报告

专业	学号	姓名	开始/结束日期	Email
计算机科学与技术系	191220156	张桓	4.11~4.12	2659428453@qq.com

实验名称

Respond to ARP

实验目的

本实验作为实现 `IPV4` 路由器功能的第一个阶段，在本阶段中需要实现对分配给路由器接口地址的 `ARP` **请求**作出响应。

实验内容

Task2：Handle ARP Request

> 首先我们回顾一下 `ARP` 包的功能，本质上讲 `ARP` 是连接链路层和网络层的协议。当一个主机 `A` 想要向另一个主机 `B` 发送数据时，若主机 `A` 只知道 `B` 的 `IP` 地址，但是不知道 `B` 的 `MAC` 地址。为了使得数据包能够成功发送， `A` 就需要先发送一个 **`ARP` 请求包**到**路由器**，这个包的包头含有源 `A` 的 `IP` 地址和 `MAC` 地址以及目的 `B` 的 `IP` 地址，而没有被填写的目的 `B` 的 `MAC` 地址，就是**被请求**的地址。
>
> 因此我们更能明确本次实验的内容：就是实现路由器的功能，使其对 **`ARP` 请求包**做出响应，如果 `ARP` 的目的 `IP` 是分配给该路由器接口的地址，那么应**创建并发送**适当的 **`ARP` 答复包**。
>
> (注：本实验只关注路由器对 `ARP` 请求包的响应，其他包目前不做处理

实现逻辑

这次实验我们对于**需要处理的包**的要求是比较"苛刻"的：

- 需要是个 `ARP` 包，通过 `if packet.has_header(Arp):` 判断
- 更进一步是个 `ARP` 请求包，通过 `if arp.operation == ArpOperation.Request:` 判断
- 这个请求包的目的 `IP` 在路由器的端口中要存在，通过 `if arp.targetprotoaddr in self.myIPs:` 判断。

  其中 `self.myIPs` 是调用 `self.net` 类的 `interfaces()` 函数得到接口，再通过 `ipaddr` 得到路由器所有端口的 `IP` .

只有满足这几点条件，我们再进行后续的操作，即**构造一个 `ARP` 回应包**。这个包的源 `MAC` 和 `IP` 为原本的请求包所请求的目的主机的 `MAC` 和 `IP` ，目的 `MAC` 和 `IP` 则为原本的请求包的 `MAC` 和 `IP` 。再从**接收请求包的那个端口**发出即可(对应下文 `15 16` 行)。

核心代码

```
1  def __init__(self, net: switchyard.llnetbase.LLNetBase):
2          # all IPs for each port in myrouter
3          self.myIPs = [intf.ipaddr for intf in self.net.interfaces()]
4  def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
5          timestamp, ifaceName, packet = recv
6          # TODO: your logic here
7          log_info(f"In {self.net.name} received packet {packet} on {ifaceName}")
8          if(packet.has_header(Arp)): #must be a ARP packet
9              arp = packet.get_header(Arp) #get the arp header
10             if arp.operation == ArpOperation.Request: # must be a request packet
11                 if arp.targetprotoaddr in self.myIPs: #the dest's IP be in my
   ports
12                     for intf in my_interfaces:
13                         if intf.ipaddr == arp.targetprotoaddr:
14                             intf_dest = intf #find the intf of the dest's IP
15                             arp_reply = create_ip_arp_reply(intf_dest.ethaddr,
   arp.senderhwaddr, intf_dest.ipaddr, arp.senderprotoaddr)
16                             self.net.send_packet(ifaceName, arp_reply) #construct
   the reply arp and send it by get's port
```

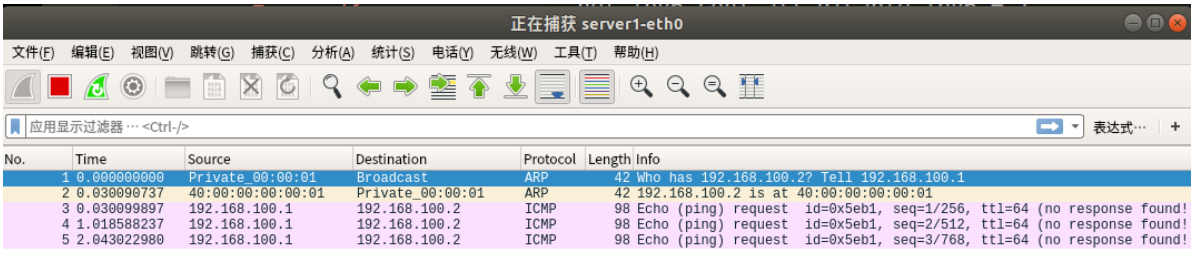
实验结果

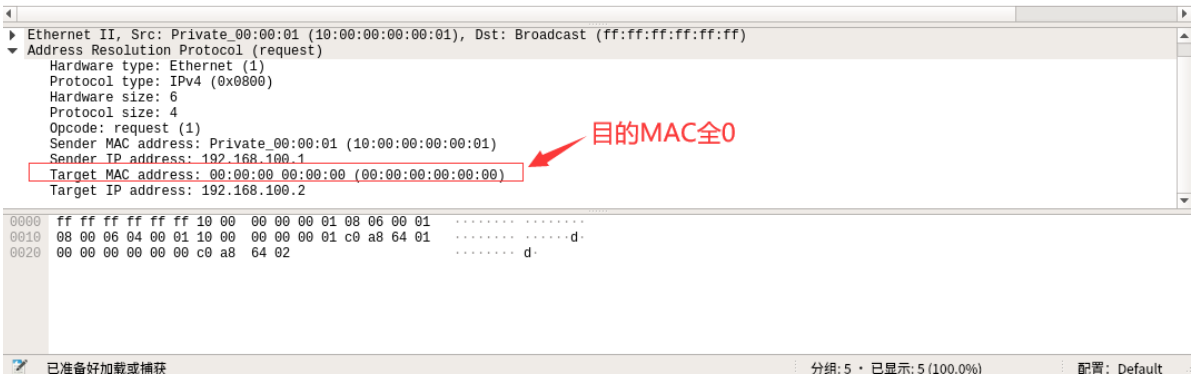
```
Results for test scenario ARP request: 6 passed, 0 failed, 0 pending


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


All tests passed!
```

在mininet上测试

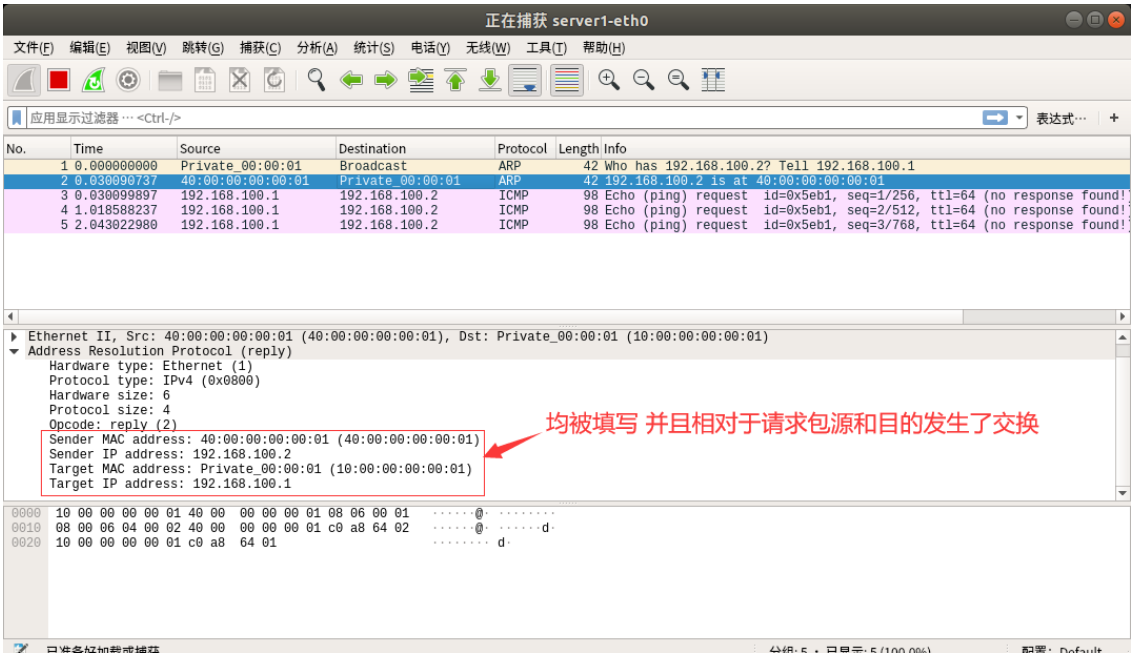
仿照教程里的例子来做，开启虚拟环境后在`router`的`xterm`上运行`myrouter.py`，在`server1`的`xterm`上输入命令`wireshark -k &`开始捕获，然后构造流量`server1 ping -c3 router`，捕获信息如下图：

- 路由器首先收到一个关于它自己的`IP`地址的`ARP`请求包，然后收到一个`ICMP echo`请求。在`Wireshark`中，单击捕获窗口第一行的`ARP`请求包，可以看到："Target MAC adderss"目前都是0，因为这是需要请求的地址。





- 同样在`Wireshark`中，单击`ARP`响应数据包（捕获窗口的第二行）时，可以看到：`ARP`标头中的所有地址都被填写（并且交换了源地址和目标地址）：



通过这些分析可以判断 `myrouter.py` 的 `ARP` 请求响应机制基本实现。

Task3：Cached ARP Table

> 实际的路由器中，目标 `IP` 地址和 `MAC` 地址之间会存储一个映射关系。 这是因为将 `IP` 数据包发送到另一台主机时需要与目标 `IP` 地址关联的 `MAC` 地址。 如果从路由器收到的 `ARP` 请求中**记录**所有的 <源 `IP` ,源 `MAC` >对，则可以减少 `ARP` 请求的开销。

实现逻辑

这个 `ARP` 表和交换机表的目的及其功能及其相似，只不过交换机表实现的是 `MAC` 地址和端口的映射，而 `ARP` 表实现的是 `IP` 地址和 `MAC` 地址的映射，都是为了做好记录以减少不必要的查询开销。

我们沿用交换机表的思路，同样用**字典**来实现 `ARP` 表，创建一个空字典 `arp_table` 作为 `ARP` 表，字典的表项为 `<IP，MAC>` 。当路由器接收到一个 `ARP` 请求包时，就用该 `ARP` 头部的**<源 `IP` ,源 `MAC` >**来更新表项。即 `self.arp_table[arp.senderprotoaddr] = arp.senderhwaddr` 。

核心代码

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        ...
        # TODO: your logic here
        log_info(f"received packet {packet} on {ifaceName}")
        if(packet.has_header(Arp)): #must be a ARP packet
            arp = packet.get_header(Arp) #get the arp header
            if arp.operation == ArpOperation.Request: # must be a request packet
                self.arp_table[arp.senderprotoaddr] = arp.senderhwaddr #add or update the arp_table
                ...
                self.print_table() #print the arp_table
```

打印ARP表并测试

当`ARP`表**发生更新或改变时**，就会将当前的`ARP`表**打印到终端**。可以看到上方代码第`10`行就调用了打印函数`self.print_table()`。

这个函数是用**面向对象**实现的类函数，该函数的实现如下：

```
1  def print_table(self):
2
   print('\033[1;35m=\033[0m'*18,'\033[1;32mARP_TABLE\033[0m','\033[1;35m=\033[0m'*1
   8)
3
    print('|','\033[1;36mIP\033[0m'.center(31),'|','\033[1;36mMAC\033[0m'.center(31)
   ,'|')
4      print("+" + "-" * 45 + "+")
5      for ip,mac in self.arp_table.items():
6          print('| {0} | {1} |'.format(str(ip).center(20), str(mac).center(20)))
7          print("+" + "-" * 45 + "+")
8      print('\033[1;35m=\033[0m'*47)
```

运行`testcase`中给好的测试用例得到输出如下：

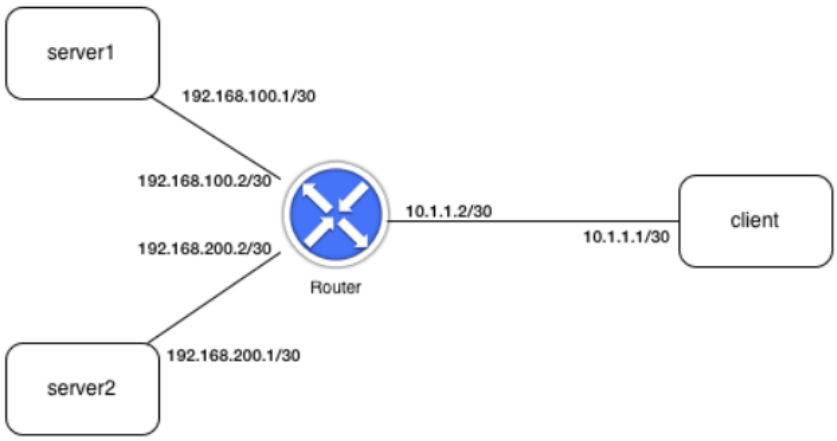
```
(syenv) njucs@njucs-VirtualBox:~/networkLab/lab-3-huanhuan6666$ swyard -t testcases/myrouter1_testsenario.
srpy myrouter.py
16:26:00 2021/04/12     INFO Starting test scenario testcases/myrouter1_testsenario.srpy
16:26:00 2021/04/12     INFO received packet Ethernet 30:00:00:00:00:01->ff:ff:ff:ff:ff:ff ARP | Arp 30:00
:00:00:00:01:192.168.1.100 ff:ff:ff:ff:ff:ff:192.168.1.1 on router-eth0
================== ARP_TABLE ==================
|          IP          |          MAC         |
+--------------------------------------------+
|    192.168.1.100     |  30:00:00:00:00:01   |
+--------------------------------------------+
=============================================
16:26:00 2021/04/12     INFO received packet Ethernet ab:cd:ef:00:00:01->10:00:00:00:00:01 IP | IPv4 192.1
68.1.242->10.10.12.34 ICMP | ICMP EchoRequest 0 42 (13 data bytes) on router-eth0
16:26:00 2021/04/12     INFO received packet Ethernet 60:00:de:ad:be:ef->ff:ff:ff:ff:ff:ff ARP | Arp 60:00
:de:ad:be:ef:10.10.1.1 ff:ff:ff:ff:ff:ff:10.10.1.2 on router-eth1
================== ARP_TABLE ==================
|          IP          |          MAC         |
+--------------------------------------------+
|    192.168.1.100     |  30:00:00:00:00:01   |
+--------------------------------------------+
|      10.10.1.1       |  60:00:de:ad:be:ef   |
+--------------------------------------------+
=============================================
16:26:00 2021/04/12     INFO received packet Ethernet 70:00:ca:fe:c0:de->ff:ff:ff:ff:ff:ff ARP | Arp 70:00
:ca:fe:c0:de:10.10.5.5 ff:ff:ff:ff:ff:ff:10.10.0.1 on router-eth1
================== ARP_TABLE ==================
|          IP          |          MAC         |
+--------------------------------------------+
|    192.168.1.100     |  30:00:00:00:00:01   |
+--------------------------------------------+
|      10.10.1.1       |  60:00:de:ad:be:ef   |
+--------------------------------------------+
|      10.10.5.5       |  70:00:ca:fe:c0:de   |
+--------------------------------------------+
=============================================

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

```
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
```

通过输出的信息可以知道测试用例一共有3个`ARP`请求包，结合打印的`INFO`信息可以看出`ARP`表的行为完全符合预期。

下面我自己构造了一些流量来演示`ARP`表的实时变化，在`mininet`上输入一系列命令。

> `mininet`的网络拓扑如下：



- 输入`client ping -c3 server1`后终端输出的`ARP`表如下：

```
15:47:15 2021/04/12     INFO Using network devices: router-eth2 router-eth0 rout
er-eth1
15:49:07 2021/04/12     INFO received packet Ethernet 30:00:00:00:00:01->ff:ff:f
f: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 on ro
uter-eth2
================= ARP_TABLE =================
|          IP          |          MAC         |
+--------------------------------------------+
|       10.1.1.1       |  30:00:00:00:00:01   |
+--------------------------------------------+
============================================
```

可以看到由于`client`会向路由器发送一个`ARP`请求包，这是一个新的表项，它的`IP`和`MAC`会被记录。

- 输入`server1 ping -c3 client`后终端输出的`ARP`表如下：

```
15:50:15 2021/04/12     INFO received packet Ethernet 10:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.1
00.2 on router-eth0
================= ARP_TABLE =================
|          IP          |          MAC         |
+--------------------------------------------+
|       10.1.1.1       |  30:00:00:00:00:01   |
+--------------------------------------------+
|    192.168.100.1     |  10:00:00:00:00:01   |
+--------------------------------------------+
============================================
```

由于 `server1` 也会向路由器发送一个 `ARP` 请求包， `ARP` 表中会增加其表项，内容为 `server1` 的 `IP` 和 `MAC` 。

- 输入 `server2 ping -c3 client` 后终端输出的 `ARP` 表如下：

```
15:50:47 2021/04/12     INFO received packet Ethernet 20:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 20:00:00:00:00:01:192.168.200.1 00:00:00:00:00:00:192.168.2
00.2 on router-eth1
================== ARP_TABLE ==================
|          IP          |          MAC         |
+--------------------------------------------+
|       10.1.1.1       |  30:00:00:00:00:01   |
+--------------------------------------------+
|    192.168.100.1     |  10:00:00:00:00:01   |
+--------------------------------------------+
|    192.168.200.1     |  20:00:00:00:00:01   |
+--------------------------------------------+
================================================
```

`server2` 也是第一次发包，作为一个新的表项，也被正确记录了。

实现超时机制

> 注意这一部分的代码在源码中我**注释**掉了，如果需要检查直接将注释去掉即可，关键代码在下面会有展示。

`ARP` 表的表项经过一段时间后没有更新的话会被删除，实现这个也很简单，类似 `lab2` 中交换机表的超时机制。我们在更新或者添加表项的时候，多保存该表项进入表的时间信息，即表项变为 `<IP : [MAC, time]>` ，其中 `time` 可以通过 `time.time()` 函数获取。然后在每次处理新的数据包之前遍历所有的表项，用当前时间减去该表项的进入时间，当**差值** `time.time()-self.arp_table[key][1]` 大于 `30s` 时就 `pop()` 掉这个表项即可。

代码只需在原有基础上修改几行即可：

- 删除超时表项

```
for key in self.arp_table:  #remove the out time items
    if time.time() - list(self.arp_table[key][1]) > 30:
        log_info(f"remove item {key}:{self.arp_table[key]}")
        self.arp_table.pop(key)
```

- 更新或添加表项

```
 self.arp_table[arp.senderprotoaddr] = [arp.senderhwaddr, time.time()]#add or
update the arp_table
```

下面简单验证一下超时机制

- 键入 `client ping -c3 router` ，结果如下：正常添加表项

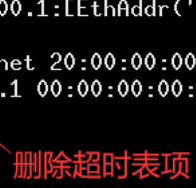
```
19:53:41 2021/04/20     INFO received packet Ethernet 30:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 30:00:00:00:00:01:10.1.1.1 00:00:00:00:00:00:10.1.1.2 on ro
uter-eth2
================== ARP_TABLE ==================
|          IP          |          MAC          |
+--------------------------------------------+
|       10.1.1.1       |  30:00:00:00:00:01    |
+--------------------------------------------+
==============================================
```

- 间隔**小于** `30s` 后再键入 `server1 ping -c3 router`，结果如下：由于没有表项超时，正常添加

```
19:53:42 2021/04/20     INFO received packet Ethernet 30:00:00:00:00:01->40:00:0
0:00:00:03 IP | IPv4 10.1.1.1->192.168.100.2 ICMP | ICMP EchoRequest 4680 2 (56
data bytes) on router-eth2
19:53:43 2021/04/20     INFO received packet Ethernet 30:00:00:00:00:01->40:00:0
0:00:00:03 IP | IPv4 10.1.1.1->192.168.100.2 ICMP | ICMP EchoRequest 4680 3 (56
data bytes) on router-eth2
19:53:58 2021/04/20     INFO received packet Ethernet 10:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.1
00.2 on router-eth0
================== ARP_TABLE ==================
|          IP          |          MAC          |
+--------------------------------------------+
|       10.1.1.1       |  30:00:00:00:00:01    |
+--------------------------------------------+
|    192.168.100.1     |  10:00:00:00:00:01    |
+--------------------------------------------+
==============================================
```

- 间隔**大于** `30s` 后再键入 `server2 ping -c3 router`，结果如下：之前的两个表项都超时了，应该删除。

```
19:54:29 2021/04/20     INFO remove item 10.1.1.1:[EthAddr('30:00:00:00:00:01'),
 1618919621.118634]
19:54:29 2021/04/20     INFO remove item 192.168.100.1:[EthAddr('10:00:00:00:00:
01'), 1618919638.148506]
19:54:29 2021/04/20     INFO received packet Ethernet 20:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 20:00:00:00:00:01:192.168.200.1 00:00:00:00:00:00:192.168.2
00.2 on router-eth1
================== ARP_TABLE ==================
|          IP          |          MAC          |
+--------------------------------------------+
|    192.168.200.1     |  20:00:00:00:00:01    |
+--------------------------------------------+
==============================================
```



可以判断超时机制正确实现了。

总结感想

这次实验总体来说难度不大，主要是理清了路由器的一些概念，完成代码并不是很难。实现完交换机表后，这个 `ARP` 表就很轻易了。

更重要的是学了写 `python` 的技巧，比如打印表格对齐，输出颜色等等。

参考文章：python输出对齐

最后感谢助教的耐心批改😁😁😁