
南京大学

# LAB 3：IPv4

# Router_Respond to RAP

课程名称：计算机网络

姓名：孙文博

学号：201830210

学院：计算机科学与技术系

Email：201830210@smail.nju.edu.cn

任课教师：李文中

实验时间：2022.3.24 - 2022.4.7

# 一、 实验目的

在前面两次实验中，我们分别实现了集线器 hub 和链路层交换机 Switch 的模拟。在接下来的实验 3 到实验 5 中，我们将创建一个功能齐全的 IPv4 路由器。概括地说，我们的路由器将具有以下功能：

- 响应/发出 ARP 请求
- 使用查找表接收数据包并将其转发到目的地
- 响应/生成 ICMP 消息

具体而言，我们需要实现以下五个任务：

1.响应分配给路由器接口的地址的 ARP(地址解析协议)请求。

2.对没有已知以太网 MAC 地址的 IP 地址发出 ARP 请求。路由器通常必须向其他主机发送数据包，并且需要以太网 MAC 地址才能这样做。

3.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信息库中执行地址查找(“最长前缀匹配”查找)。我们最终将只在路由器中使用“静态”路由，而不是实现像 RIP 或 OSPF 那样的动态路由协议。

4.响应 Internet 控制消息协议(ICMP)消息，例如回显请求("ping")。

5.必要时生成 ICMP 错误消息,例如当 IP 数据包的 TTL(生存时间)值已减为零时。

在实验三中，我们只需要完成第一项任务，即创建一个可以响应处理 ARP 请求的路由器。

# 二、 实验内容

## 1. 接受并分析收到的 ARP 请求数据包

通过查阅 API 中相关部分得知，对于 ARP 数据包头，有两种类型的地址，每种地址类型包括源地址和目标地址，总共给出四个地址。其中源 IP 地址和目标 IP 地址分别被称为 senderprotoaddr 和 targetprotoaddr，源 MAC 地址和目标 MAC 地址分别被称为 senderhwaddr 和 targethwaddr。对于 ARP 请求，将给出源以太网和 IP 地址以及目标 IP 地址，而目的以太网地址未知：这是正在请求的地址。首先我们利用 packet 类的 get_header()方法尝试获得 ARP 表头，并判断其是否是一个 ARP 请求：

```
# TODO: your logic here
log_info(f"Received a packet {packet} on {ifaceName} at {timestamp}.")
arp = packet.get_header(Arp)

# receive an ARP request
if arp is not None and arp.operation == 1:
```

其中 arp 的 operation 属性对应 ARP 数据报类别，分为：

*class* switchyard.lib.packet.common.**ArpOperation**

An enumeration.

**Request = 1**

**Reply = 2**

## 2. 创建并发送合适的 ARP 回复

接下来，对于我们收到的每个 ARP 请求，需要判断 ARP 标头中

的字段 targetprotoaddr（目标 IP 地址）是否是分配给路由器上的接口之一的 IP 地址。如果目标 IP 地址是分配给路由器接口的地址，则应创建并发送适当的 ARP 回复。（如果目标 IP 地址不是分配给路由器的某个接口的 IP 地址，则即使有足够的信息，也不需要使用 ARP 回复进行响应）。ARP 回复应发送到 ARP 请求所在的同一接口处。具体的实现代码如下：

```
# search the ip/mac
for iface in self.net.interfaces():
    if iface.ipaddr == arp.targetprotoaddr:
        # create an Arp reply
        ether = Ethernet()
        ether.src = iface.ethaddr
        ether.dst = arp.senderhwaddr
        ether.ethertype = EtherType.ARP
        reply_arp = Arp(operation = ArpOperation.Reply,
                    senderhwaddr = iface.ethaddr,
                    senderprotoaddr = iface.ipaddr,
                    targethwaddr = arp.senderhwaddr,
                    targetprotoaddr = arp.senderprotoaddr)
        reply_packet = ether + reply_arp
        log_info(f"Send a packet {reply_packet} out {ifaceName} at {timestamp}.")
        self.net.send_packet(ifaceName,reply_packet)
```

其中 self.net 的 interfaces()方法是得到当前路由器接口列表，通过对每个接口 iface 的 ipaddr 属性的调用判断当前接口的 IP 地址是否是需要的地址。若找到需要地址则打包一个以太网帧并加入 ARP 回复报头，其中源地址是路由器对应接口的地址，目标地址是 ARP 请求发送方的地址。最后通过 net 的 send_packet()方法将数据包发送到来时的接口上。任务完成！

如果在路由器中收到的数据包不是 ARP 请求，则我们现在忽略它（丢弃它）。在以后的任务中，我们将在路由器中处理更多传入的数据包类型。

## 3. 为路由器构造 ARP 缓存表

我们还需要在路由器中存储目标 IP 地址和以太网 MAC 地址之间的映射（假设存在一对一的映射）。因为当路由器向另一台主机发送 IP 数据包时，它还需要与目标 IP 地址关联的以太网地址。如果我们让路由器“记住”收到的 ARP 请求中源 IP/MAC 地址对，则它可以避免未来必须构造和发送 ARP 请求来获取相同的信息。

缓存表类似于以太网学习交换机中使用的表。对于每个条目，有两个或多个字段需要 IP 和 MAC 地址。缓存的 ARP 表的结构如下所示：

| IP | MAC Address |
| --- | --- |
| 10.1.2.3 | 01:02:03:04:05:06 |
| ... | ... |

当路由器接收到带有 ARP 请求的数据包时，它添加或更新缓存 ARP 表的条目。例如，如果有一个 ARP 请求的源以太网地址为 01:02:03:04:05:06，IP 源地址 10.1.2.3，则路由器将增加键为 10.1.2.3 与值为 01:02:03:04:05:06 的键值对。整个数据结构可以用 python 中的 dict(字典)类型实现。具体代码如下：

```
# update ARP table
for item in self.start_time:
    if time.time()-self.start_time[item] > 20:
        del self.table[item]
        log_info("Delete an entry.")
```

```
# receive an ARP request
if arp is not None and arp.operation == 1:
    # add an entry in ARP table
    self.table[arp.senderprotoaddr] = arp.senderhwaddr
    self.start_time[arp.senderprotoaddr] = time.time()
    log_info("Add an entry.")

    # print the table
    log_info("***Now print the table.***")
    print(self.table)
```

类似于 Lab 2 中的数据结构，我们使用一个名为 table 的字典来存储 ARP 缓存表，并在每次收到包的时候进行更新，淘汰超时条目。每增加一条新条目，我们打印一次缓存表以验证正确性。

在后续 testcase 检测中，打印出的缓存表如下：

```
(testsyenv) njucs@njucs-VirtualBox:~/switchyard/lab-03-Florentino-73$ swyard -t testcases/myrouter1_testscenario.srpy myrouter.py
12:10:38 2022/04/07     INFO Starting test scenario testcases/myrouter1_testscenario.srpy
12:10:38 2022/04/07     INFO Received a 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 at 0.0.
12:10:38 2022/04/07     INFO Add an entry.
12:10:38 2022/04/07     INFO ***Now print the table.***
{IPv4Address('192.168.1.100'): EthAddr('30:00:00:00:00:01')}
12:10:38 2022/04/07     INFO Send a packet Ethernet 10:00:00:00:00:01->30:00:00:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.1.1 30:00:00:00:00:01:192.168.1.100 out router-eth0 at 0.0.
12:10:38 2022/04/07     INFO Received a packet Ethernet ab:cd:ef:00:00:01->10:00:00:00:00:01 IP | IPv4 192.168.1.242->10.10.12.34 ICMP | ICMP EchoRequest 0 42 (13 data bytes) on router-eth0 at 1.0.
12:10:38 2022/04/07     INFO Received a 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 at 1.0.
12:10:38 2022/04/07     INFO Add an entry.
12:10:38 2022/04/07     INFO ***Now print the table.***
{IPv4Address('192.168.1.100'): EthAddr('30:00:00:00:00:01'), IPv4Address('10.10.1.1'): EthAddr('60:00:de:ad:be:ef')}
12:10:38 2022/04/07     INFO Received a 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 at 1.0.
12:10:38 2022/04/07     INFO Add an entry.
12:10:38 2022/04/07     INFO ***Now print the table.***
{IPv4Address('192.168.1.100'): EthAddr('30:00:00:00:00:01'), IPv4Address('10.10.1.1'): EthAddr('60:00:de:ad:be:ef'), IPv4Address('10.10.5.5'): EthAddr('70:00:ca:fe:c0:de')}
12:10:38 2022/04/07     INFO Send a packet Ethernet 10:00:00:00:00:02->70:00:ca:fe:c0:de ARP | Arp 10:00:00:00:00:02:10.10.0.1 70:00:ca:fe:c0:de:10.10.5.5 out router-eth1 at 1.0.
```

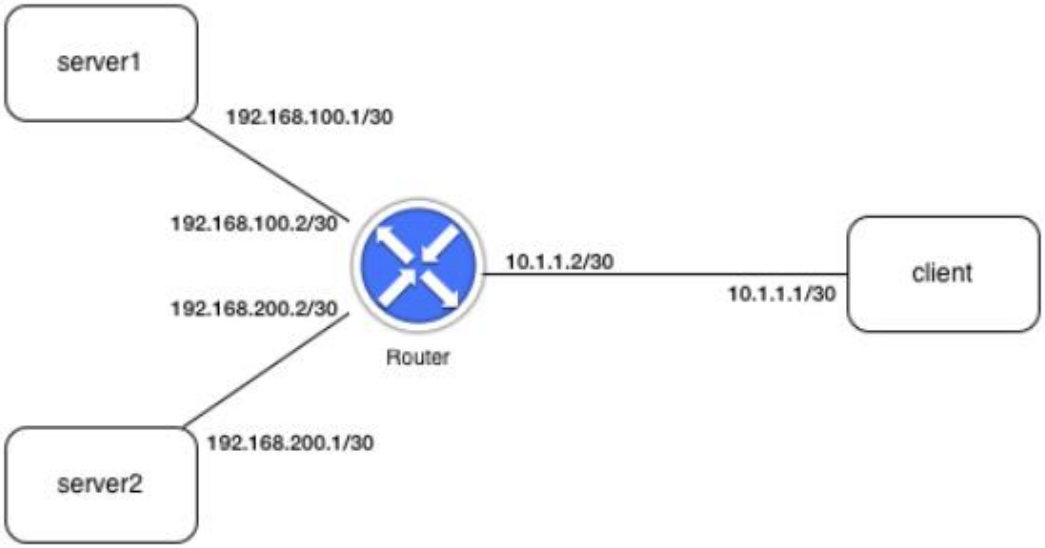
可以看到，当路由器接受第一条 ARP 请求 Arp30:00:00:00:00:01:192.168.1.100|ff:ff:ff:ff:ff:ff:192.168.1.1 时，缓存表中也添加了一条键值对 IPv4Address('192.168.1.100'): EthAddr('30:00:00:00:00:01')，符合我们的要求。后续变化同理。

## 4. 测试用例与 mininet 模拟

修改 my_router.py 文件后进行 Switchyard 测试，结果如下：

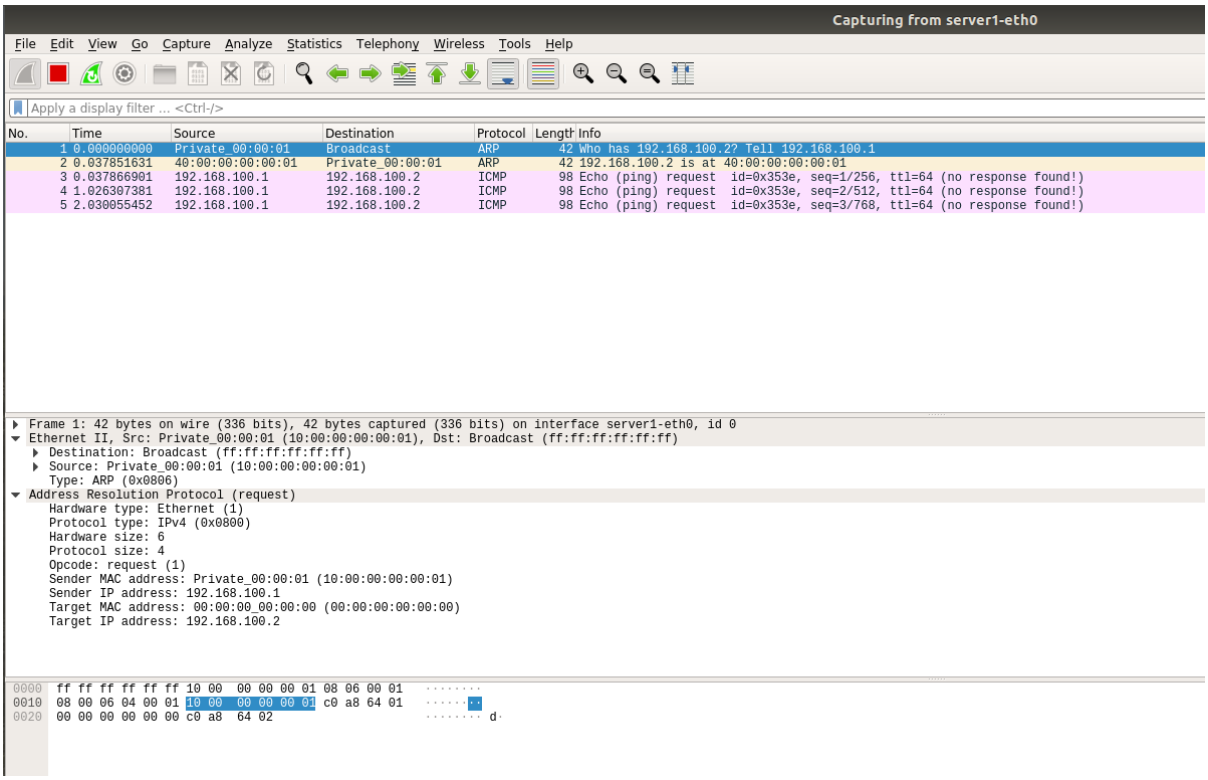
```
Results for test scenario ARP request: 6 passed, 0 failed, 0 pending


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 中搭建我们自己的网络拓扑如下图：




我们从 server1 向对应的路由器接口（192.168.199.2/30）发送 ICMP 回显请求并进行 wireshark 抓包检测，得到结果如下：

```
*** Starting CLI:
mininet> xterm router
mininet> server1 wireshark -k &
mininet> server1 ping -c3 192.168.100.2
17:35:16.963     Main Warn QStandardPaths: XDG_RUNTIME_DIR
oot'
PING 192.168.100.2 (192.168.100.2) 56(84) bytes of data.
```




其中路由器最初收到一个针对它自己的 IP 地址的 ARP 请求并对此进行正确响应，可以看到第一行的 ARP 请求数据包中，目标 MAC 地址全为 1（即广播 MAC 地址），因为这是请求的地址；第二行是我们的路由器的 ARP 回复时，现在 ARP 标头中的所有地址都已填写（并且源地址和目标地址已有效交换）。此时在 ARP 的缓存表中还会添加一项条目（如我们前面所验证的那样）。

后三行是路由器收到的三个 ICMP 回显请求。由于我们的路由器尚未编程以响应 ping 请求，因此不会发生其他任何事情（接下来的实验中会陆续实现）。

# 三、 核心代码

本次实验需要修改的 my_router.py 文件的主要部分代码截图已在实验过程中展示并解释说明，具体代码已提交到 github classroom 中。

# 四、 总结与反思

本次实验相较上一次实验更加顺利，因为有了之前的基础，对 switchyard 以及 python 语言都有了更深的理解，同时这次实验也是路由器模拟的开始，希望接下来的几个实验中顺利进行！💪