南京大学

LAB 4：IPv4 Router Forwarding Packets

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：2022.4.7 - 2022.4.21

一、 实验目的

在实验三到实验五中，我们将创建一个功能齐全的 IPv4 路由器。总地来说，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，互联网路由器的基本功能是：

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，我们已经实现了任务一，创建了一个可以响应处理 ARP 请求的路由器。本次实验我们将在实验三的基础上继续改进路由器，实现转发数据包的功能，对应上面的#2，#3 两个任务。

二、 实验内容

1. 为路由器创建一个 IP 转发查找表

与实验二中的交换机 Switch 一样，路由器也有自己的转发表。当接受到数据包时，路由器将根据目标地址与转发表中进行匹配（最大长度后缀原则），并将数据包转发到正确的接口上。根据手册要求，转发表中每个条目都包含四项内容：网络前缀（例如 149.43.0.0）；网络“掩码”（例如 255.255.0.0）；“下一跳”IP 地址，如果目标网络前缀不是直接连接的网络；转发数据包的接口。其格式如下：

网络地址	子网地址	下一跳地址	界面
192.168.1.0	255.255.255.0	0.0.0.0	路由器-eth0
172.16.1.0	255.255.255.0	0.0.0.0	路由器-eth1

我们同样使用一个字典 forwarding_table 来存储转发表，其中字典的键是网络地址和子网掩码共同组成的字符串，其格式为“网络地址/子网掩码”，值是下一跳地址和路由器端口构成的列表，实现该部分内容的代码如下：

```
def start(self):
    '''
    Main method for router; we stay in a loop in this method, receiving
    packets until the end of time.
    '''

    arp_cache = {} # ARP缓存表
    forwarding_table = {} # 路由器转发表
    waiting_queue = [] #  创建队列轮流处理，包含等待ARP解析的IP数据包信息

    # 转发表内容的来源有两个，一个来自于路由器本身的端口，另一个来自于转发表文件
    my_interfaces = self.net.interfaces()
    for intf in my_interfaces:
        # print(intf)
        ipaddr = IPv4Address(int(intf.ipaddr) & int(intf.netmask))  # 通过掩码取出端口的IP地址所在的子网
        key = IPv4Network(str(ipaddr) + '/' + str(intf.netmask))    # "子网地址/掩码"的组合形成键
        forwarding_table[key] = ['', intf.name]                     # 下一跳地址与路由器端口作为值
```

```
# 打开转发表文件并存入转发表字典中
with open("forwarding_table.txt") as file:
    for line in file:
        info = line.rsplit()        # 去掉行尾回车
        if info:                    # 确保不是空行
            key = IPv4Network(info[0] + '/' + info[1])  # 读取子网地址与掩码组合
            forwarding_table[key] = info[2:] # 读取下一跳地址与路由器端口

# 打印转发表中的表项
log_info("***Now print forwarding table.***")
for item in forwarding_table.items():
    print(item)
```

需要注意的是，转发表内容的来源有两个，一个来自于路由器本身的端口，另一个来自于转发表文件 forwarding_table.txt，此文件由 Switchyard 测试场景或 Mininet 启动脚本生成，格式与我们的转发表类似，每行包含 4 个由空格分隔的项目：网络地址、子网掩码、下一跳地址和转发数据包接口。

2. 将目标 IP 地址与转发表匹配

建立转发表后，路由器接收到的 IP 数据包中的目标地址应该与转发表匹配。如果表中有两个项目都匹配，则使用“最长前缀原则”进行匹配，即选择地址前缀匹配长度最大的那个项目。

实现代码如下：

```
if my_packet:
    # 最长前缀匹配
    for key in forwarding_table.keys():
        if pkt[1].dst in key:
            if key.prefixlen > prefix:
                net = key
                prefix = key.prefixlen
```

需要注意的是，若表中没有匹配项或者数据包时发给路由器自身的，那么丢弃这个数据包：

```
# 抛弃目标地址为路由器的包
my_packet = True
for intf in my_interfaces:
    if pkt[1].dst == intf.ipaddr:
        my_packet = False
        break
```

3. 接受数据包等待转发

有了以上准备工作，我们的路由器可以开始接受并转发数据包了！

总体代码框架如下：

```
while True:
    gotpkt = True
    try:
        timestamp, dev, pkt = self.net.recv_packet(timeout=1.0)
    except NoPackets:
        log_debug("No packets available in recv_packet")
        gotpkt = False
    except Shutdown:
        log_debug("Got shutdown signal")
        break

    if gotpkt:
        log_debug("Got a packet: {}".format(str(pkt)))

        # 将非ARP和IPv4的包抛弃 （Lab4不用考虑）
        arp = pkt.get_header(Arp)
        ipv4 = pkt.get_header(IPv4)

        if arp:
            # 更新ARP缓存表
            arp_cache[arp.senderprotoaddr] = arp.senderhwaddr
            # 打印ARP缓存表表项
            log_info("***Now print ARP cache table.***")
            for key, value in arp_cache.items():
                print(key, "\t", value)
            print()
            # 如果收到的是一个ARP请求
            if arp.operation == ArpOperation.Request:
                for intf in my_interfaces:
                    if arp.targetprotoaddr == intf.ipaddr: # 若存在对应表项，则发送ARP回复到对应的接口上
                        packet = create_ip_arp_reply(intf.ethaddr, arp.senderhwaddr,
                                                     arp.targetprotoaddr, arp.senderprotoaddr)
                        self.net.send_packet(intf.name, packet)
                        log_debug("send packet {} to {}".format(packet, intf.name))
```

```
elif ipv4:
    # 准备转发数据包的预处理
    pkt[1].ttl = pkt[1].ttl-1 # TTL字段减一，将在下个项目中处理过期的TTL
    # print(pkt)
    prefix = 0 # 记录当前最大前缀的长度
    net = IPv4Network('0.0.0.0/0')

    # 抛弃目标地址为路由器的包
    my_packet = True
    for intf in my_interfaces:
        if pkt[1].dst == intf.ipaddr:
            my_packet = False
            break

    if my_packet:
        # 最长前缀匹配
        for key in forwarding_table.keys():
            if pkt[1].dst in key:
                if key.prefixlen > prefix:
                    net = key
                    prefix = key.prefixlen
        # 抛弃无法匹配的包
        if prefix != 0:
            # 确定下一跳地址
            if forwarding_table[net][0]: # 转发表中已有下一跳地址
                dstipaddr = IPv4Address(forwarding_table[net][0])
            else:
                dstipaddr = pkt[1].dst
            # 找出转发端口
            interface = forwarding_table[net][1]
            for intf in my_interfaces:
                if intf.name == interface:
                    router_intf = intf
                    break
            pkt[0].src = router_intf.ethaddr
            # 将数据包和转发端口、目标地址打包成一个类对象加入队列
            waiting_queue.append(Waiting_packet(pkt, router_intf, dstipaddr))
```

首先，目前阶段我们只处理 ARP 和 ipv4 标头的数据包，对于 ARP 数据包，如果收到的是一个 ARP 请求，则执行我们实验三中完成的部分，即查找 ARP 缓存表并且在已知请求地址的 MAC 地址情况下发送一个 ARP 回复给对应的接口，同时增加 ARP 缓存表内容。

对于 ipv4 表头的 IP 数据包，首先将数据包的 TTL 字段减一，以在下个阶段处理过期的 TTL，接着对于符合条件的数据包，在转发表中进行最长前缀匹配确定下一跳地址和转发端口，最后我们将此时的数据包和它将被转发到的目标地址和转发端口打包成一个类对象放入队列 waiting_queue 中，等待下一步处理。

4. 发送 ARP 请求并转发数据包

接下来我们要为待转发的 IP 数据包创建一个以太网标头，因此我们需要知道与应将数据包转发到的主机相对应的目标以太网 MAC 地址。为此我们需要发送 ARP 查询以获取与下一跳 IP 地址对应的以太网地址。我们的步骤如下：

首先，为需要解析的 IP 地址（即相应以太网地址的 IP 地址）创建并发送 ARP 请求。接着，在收到 ARP 回复后，为对应的 IP 数据包添加以太网标头，完成转发工作，同时将回复中的 IP-MAC 键值对加入到 ARP 缓存表中。但是要注意，如果一秒内没有收到相应请求的 ARP 回复，则再次发送 ARP 请求，直到同一个 IP 地址对应的发送 ARP 请求个数已经达到五个，此时放弃转发并丢弃这个数据包。

为了实现以上逻辑，我们需要创建一个队列，其中包含有关等待 ARP 解析的 IP 数据包的信息。通过代码中每次 while 循环，处理队列中的项目，查看是否需要发送 ARP 请求重传。如果收到一个 ARP 回复数据包，则从队列中删除一个项目，更新 ARP 表，构建以太网头，然后转发数据包。如果一个 ARP 请求重复发送五次都没有回复，则丢弃这个数据包。我们用一个单独的类 waiting_packet 来表示队列中等待 ARP 响应的数据包，该类中包含了最近发送的 ARP 请求的时间和重复发送到次数等变量。

这个类的实现代码如下：

```
class Waiting_packet:
    def __init__(self, pkt, intf, dstip):
        self.packet = pkt
        self.interface = intf
        self.dstipaddr = dstip
        self.lasttime = 0    # 上次发送ARP请求的时间
        self.count = 0       # 累计发送ARP请求的次数

    def forwarding(self, arp_cache, net, repeat):
        if self.dstipaddr in arp_cache.keys():
            # ARP缓存表中有匹配则替换相应地址后发送
            dstmac = arp_cache[self.dstipaddr]
            self.packet[0].dst = str(dstmac)
            net.send_packet(self.interface.name, self.packet)
            return 0

        # 需要距离上次发送超过1秒且相同的目的地址还没发送过请求
        elif time.time() - self.lasttime >= 1 and self.dstipaddr not in repeat:
            if self.count < 5:          # 发送次数不超过5次
                repeat.append(self.dstipaddr)
                # 构建ARP请求包，发送并更新状态
                ether = Ethernet()
                ether.src = self.interface.ethaddr
                ether.dst = 'ff:ff:ff:ff:ff:ff'
                ether.ethertype = EtherType.ARP
                arp = Arp(operation=ArpOperation.Request,
                          senderhwaddr=self.interface.ethaddr,
                          senderprotoaddr=self.interface.ipaddr,
                          targethwaddr='ff:ff:ff:ff:ff:ff',
                          targetprotoaddr=self.dstipaddr)
                arppacket = ether + arp
                self.lasttime = time.time() # 更新发送时间
                net.send_packet(self.interface.name, arppacket)
                self.count += 1 # 更新发送次数
                return 1
            else:
                return -1
        else:
            return 1
```

同时我们在主函数中加入对于该队列的处理，删除成功转发的包以及请求次数超过五次的数据包。实现代码如下：

```
# 对队列中所有包尝试转发并删除某些包（成功转发和ARP请求将超过5次）
repeat = []       # 存储之前已经发送过ARP请求的包的目的地址
delete = []       # 存储将要被删除的包
for item in waiting_queue:
    flag = item.forwarding(arp_cache, self.net, repeat)
    if flag == 0: # ARP表项中已有目标地址
        delete.append(item)
    elif flag == -1:
        tempip = item.dstipaddr
        for temp in waiting_queue:
            if temp.dstipaddr == tempip:
                delete.append(temp) # 删除转发成功的表项
for i in delete:
    waiting_queue.remove(i)
```

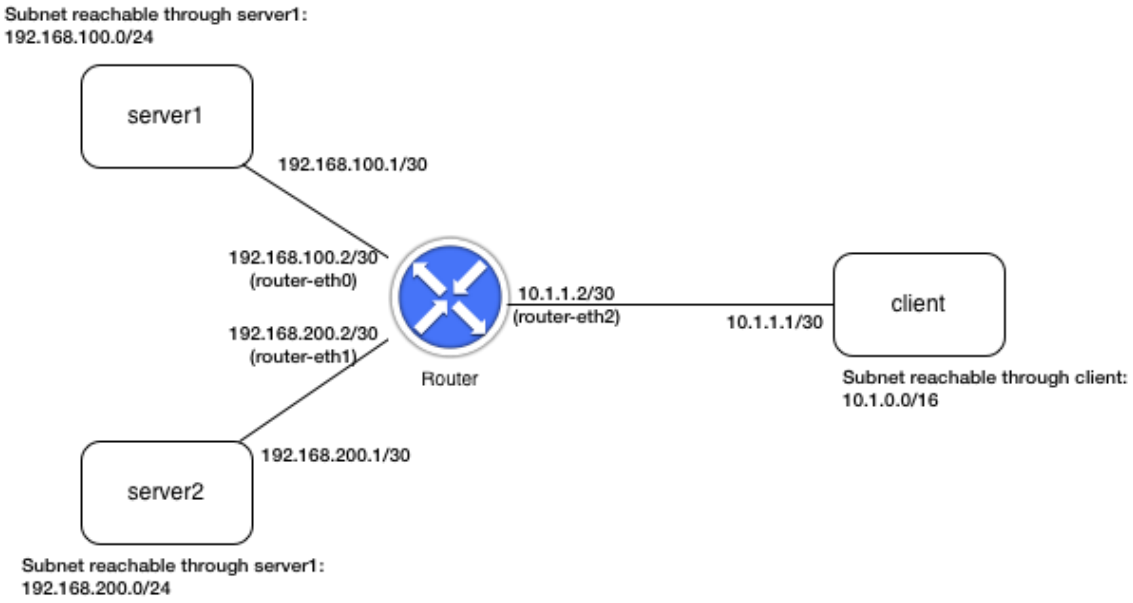
至此，本阶段对于 myrouter.py 的修改步骤已完成！

5. 通过测试用例并使用 switchyard 模拟

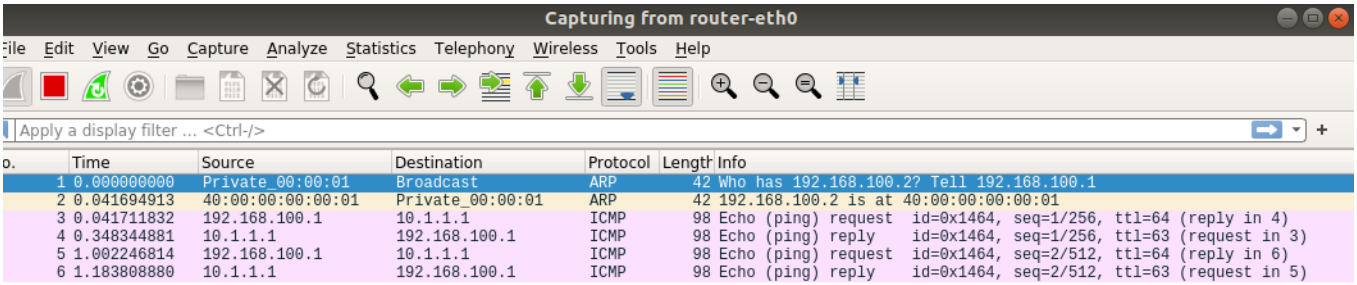
接下来通过测试样例对我们的路由器进行检测，结果如下：

```
18  IP packet destined to 172.16.64.35 should be forwarded on
    router-eth1
19  An IP packet from 192.168.1.239 for 10.200.1.1 should arrive
    on router-eth0.  No forwarding table entry should match.
20  An IP packet from 192.168.1.239 for 10.10.50.250 should
    arrive on router-eth0.
21  Router should send an ARP request for 10.10.50.250 on
    router-eth1
22  Router should try to receive a packet (ARP response), but
    then timeout
23  Router should send an ARP request for 10.10.50.250 on
    router-eth1
24  Router should try to receive a packet (ARP response), but
    then timeout
25  Router should send an ARP request for 10.10.50.250 on
    router-eth1
26  Router should try to receive a packet (ARP response), but
    then timeout
27  Router should send an ARP request for 10.10.50.250 on
    router-eth1
28  Router should try to receive a packet (ARP response), but
    then timeout
29  Router should send an ARP request for 10.10.50.250 on
    router-eth1
30  Router should try to receive a packet (ARP response), but
    then timeout
31  Router should try to receive a packet (ARP response), but
    then timeout


All tests passed!


(testsyenv) njucs@njucs-VirtualBox:~/switchyard/lab-04-Florentino-73$
```

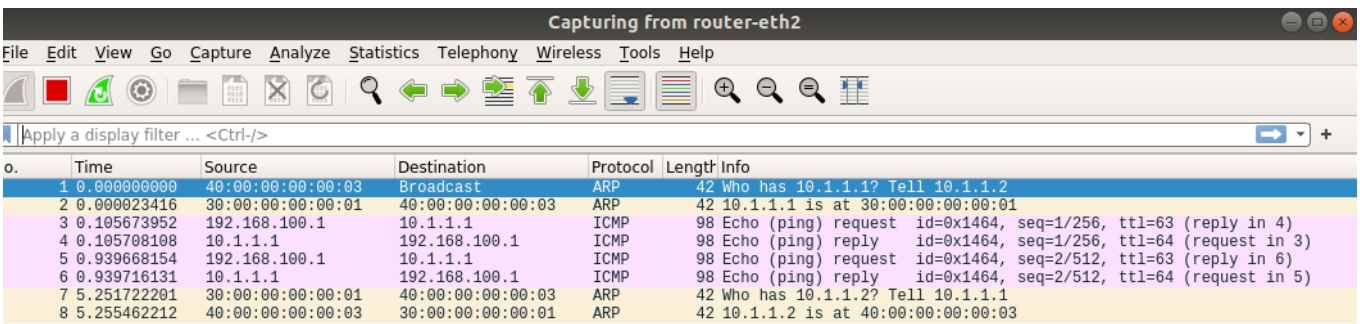
在 mininet 中搭建我们自己的网络拓扑如下图：



在我们的 Mininet 仿真中，从 server1 服务器 ping 客户终端的 IP 地址 10.1.1.1，运行 mininet>server1 ping -c2 10.1.1.1，通过 wireshark 捕获到的路由器 Eth0 端的抓包结果如下：



Eth2 端的抓包结果如下：



其中 eth2 端发送了一个 ARP 请求，这是由于路由器收到 server1 发

给 client 的数据包时，不知道 client 的 MAC 地址导致的，当它收到来自 client 的 ARP 回复时，会将这个数据包正确的转发给 client。

三、 核心代码

本次实验在实验三中的 my_router.py 文件基础上进一步增加功能，其中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

本次实验在实验三的基础上，增加了路由器的通过查询转发表实现转发数据包以及发送 ARP 请求等功能，我们的路由器已经基本完善！期待在接下来的实验中，实现一个完全体路由器！💪