



南京大學

## LAB 5: IPv4 Router - Respond to ICMP

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](mailto:201830210@smail.nju.edu.cn)

任课教师: 李文中

实验时间: 2022. 4. 21 - 2022. 5. 5

## 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。总的来说，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。

在实验三和实验四中，我们已经实现了前三个任务，创建了一个可以响应处理 ARP 请求和转发报文的路由器。在本次实验五中我们将进一步改进路由器，实现响应并发送 ICMP 请求和发送 ICMP 错误消息的功能，对应上面的#4，#5 两个任务。

## 二、实验内容

### 1. 响应 ICMP 回显请求

首先，我们的路由器要能响应 ICMP 回显请求，即对于一个发送给路由器接口之一（IP 目标地址与分配给路由器接口之一的地址相同）的 ICMP 回显请求，我们的路由器应该能够构造应该 ICMP 会先回复并发送回原 ping 的主机，具体方法为：

- ✧ 构造 ICMP 标头+回显回复，根据 API 手册内容正确填充标头中的字段（icmptype, icmdata.sequence, icmdata.identifier 等）；
- ✧ 构造应该 IP 头，目标 IP 地址设置为传入 ICMP 回显请求的源地址，IP 源地址设置为路由器的接口地址；
- ✧ 将 ICMP 回显回复从对应的路由器接口中转发出去。

在 myrouter.py 文件中，模拟对应功能的代码如下：

```
# 如果数据包是发给路由器某个接口的ICMP回显请求，那么构造一个ICMP回显回复给发送ping的原始主机
for intf in my_interfaces:
    if pkt[1].dst == intf.ipaddr:
        if pkt.has_header(ICMP) and pkt.get_header(ICMP).icmptype == 8:
            # 响应ping路由器的包
            icmp_index = pkt.get_header_index(ICMP)
            icmp = pkt.get_header(ICMP)
            icmp_reply = ICMP()
            icmp_reply.icmptype = ICMPType.EchoReply
            icmp_reply.icmpdata.sequence = icmp.icmpdata.sequence
            icmp_reply.icmpdata.identifier = icmp.icmpdata.identifier
            icmp_reply.icmpdata.data = icmp.icmpdata.data
            pkt[1].dst = pkt[1].src
            pkt[1].src = intf.ipaddr
            pkt[icmp_index] = icmp_reply
            break
```

### 2. 生成 ICMP 错误消息

接下来我们的路由器需要在一些特殊情况发生时生成一系列 ICMP 错误消息，具体而言考虑如下四种情况：

- 1) 当尝试将 IP 数据包的目标地址与转发表中的条目匹配时，找不到匹配的条目（即路由器不知道将这个数据包转发到哪里了）。在这种情况下，路由器需要将 **ICMP 目标网络不可达** 错误消息发送回 IP 数据包中源地址所指的主机。此时 ICMP 类型应该是 `destination unreachable`，ICMP code 应该是 `network unreachable`。
- 2) 在转发过程中减少 IP 数据包的 TTL 值后，TTL 变为零。在这种情况下，路由器需要将 **ICMP 超时** 错误消息发送回 IP 数据包中源地址所指的主机。注意：ICMP code 应该是 `TTL expired`。
- 3) ARP 失败。在转发过程中，路由器经常要发出 ARP 请求来获取下一跳或目的主机的以太网地址。如果不存在特定 IP 地址对应的主机，那么路由器将永远不会收到 ARP 回复。在这种情况下，如果 ARP 请求重传 5 次后路由器没有收到 ARP 回复，则路由器应将 **ICMP 目标主机不可达** 错误消息发送回 IP 数据包中源地址所指的主机。注意：ICMP 类型应该是 `destination unreachable`，ICMP code 应该是 `host unreachable`。
- 4) 传入数据包的目的地是分配给路由器接口之一的 IP 地址，但该数据包不是 ICMP 回显请求。因为路由器唯一需要处理的发往路由器本身的数据包是 ICMP 回显请求，收到任何其他数据包时，路由器都需要将 **ICMP 目标端口不可达** 错误消

息发送回 IP 数据包中的源地址。注意：ICMP 类型应该是 destination unreachable, ICMP code 应该是 port unreachable。

为便于实现以上四种情况中的生成 ICMP 错误消息，我们在实验四中封装的 Router 类中添加一个 icmperr 方法，用于生成对应的 ICMP 数据包，注意：要创建任何 ICMP 错误数据包，必须从 IPv4 标头开始，将原始数据包的前 28 个字节作为 ICMP 标头的“数据”有效负载包括在内。参考 Switchyard 文档及实验手册，我们实现的函数代码如下：

```
# 生成ICMP错误消息
def icmperr(self, inter, pkt, xtype, code):
    packet = Ethernet() + IPv4() + ICMP()
    # packet[0].src = inter.ethaddr
    packet[1].dst = pkt[1].src
    packet[1].src = inter.ipaddr
    packet[1].ttl = 10
    packet[2].icmpptype = xtype
    packet[2].icmpcode = code
    xpkt = deepcopy(pkt)
    i = xpkt.get_header_index(Ethernet)
    if i >= 0:
        del xpkt[i]
    packet[2].icmpdata.data = xpkt.to_bytes()[28:]
    packet[2].icmpdata.origdgramlen = len(xpkt)
    return packet
```

它可以生成一个将要在 inter 端口上发出的，类型为 xtype，代码为 code 的 ICMP 错误消息。

接下来我们在主体代码部分添加以上四种情况的判断，对于满足条件的情况则在相应的端口上发送 ICMP 错误消息。注意：这里生成的 ICMP 错误消息与实验四中正常转发的数据包一样，也需要通过实

验四中实现的队列排队等待发送。主体代码中对应的四种情况如下：

➤ ICMP 目标网络不可达：

```
# 最长前缀匹配
net, prefix = self.prefixmatch(forwarding_table, pkt[1].dst)
# 匹配失败/目标网络不可达
if prefix == 0:
    for intf in my_interfaces:
        if intf.name == dev:
            inter = intf
            break
    # 调用生成ICMP方法，发送ICMP目标网络不可达的错误消息
    pkt = self.icmperr(inter, pkt, ICMPType.DestinationUnreachable, 0)
    net, prefix = self.prefixmatch(forwarding_table, pkt[1].dst)
```

➤ ICMP 超时：

```
# TTL超时
if pkt[1].ttl <= 0:
    for intf in my_interfaces:
        if intf.name == dev:
            inter = intf
            break
    # 调用生成ICMP方法，发送ICMP超时的错误消息
    pkt = self.icmperr(inter, pkt, ICMPType.TimeExceeded, 0)
    prefix = 0
    net, prefix = self.prefixmatch(forwarding_table, pkt[1].dst)
```

➤ ICMP 目标主机不可达：

```
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:
        # ARP失败
        prefix = 0
        tempip = item.dstipaddr
        for temp in waiting_queue:
            if temp.dstipaddr == tempip:
                # 调用生成ICMP方法，发送五方访问ICMP目标主机的错误消息
                packet = self.icmperr(temp.interface, temp.packet,
                    ICMPType.DestinationUnreachable, 1)
                prefix = 0
                net, prefix = self.prefixmatch(forwarding_table, packet[1].dst)
```

➤ ICMP 目标端口不可达:

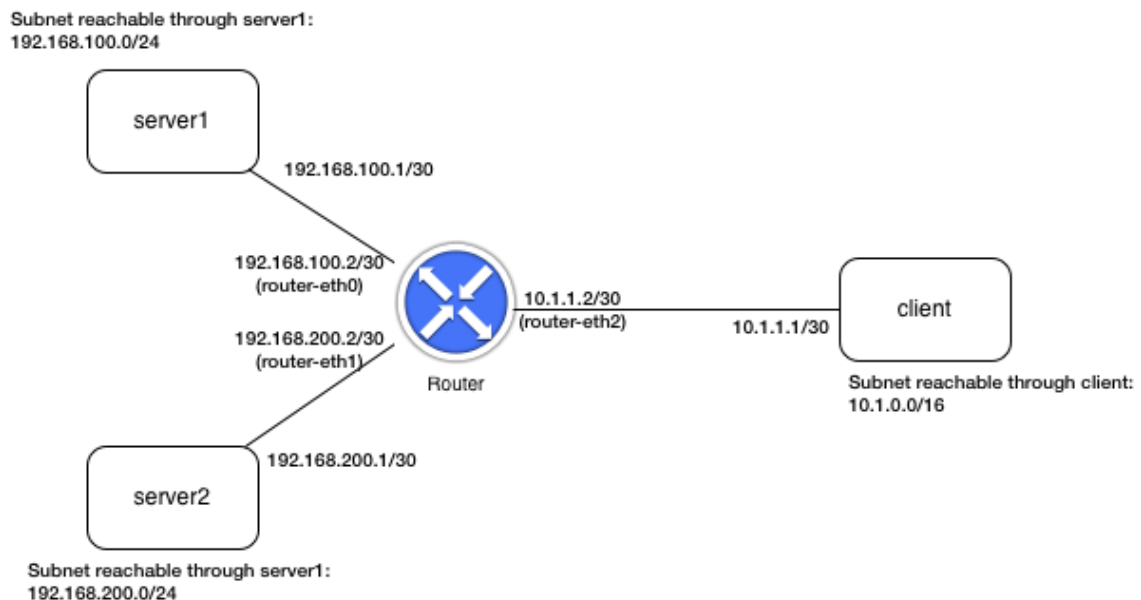
```
else:
    # 传入数据包的目的地是路由器接口之一的IP地址, 但该数据包不是ICMP回显请求
    for intf in my_interfaces:
        if intf.name == dev:
            inter = intf
            break
    # 调用生成ICMP方法, 发送ICMP目标端口不可达的错误消息
    pkt = self.icmperr(inter, pkt, ICMPType.DestinationUnreachable, 3)
    break
```

Myroute.py 其余部分代码与实验四大致相同, 这里不再展示。

### 3. 通过测试用例检测并使用 switchyard 进行模拟

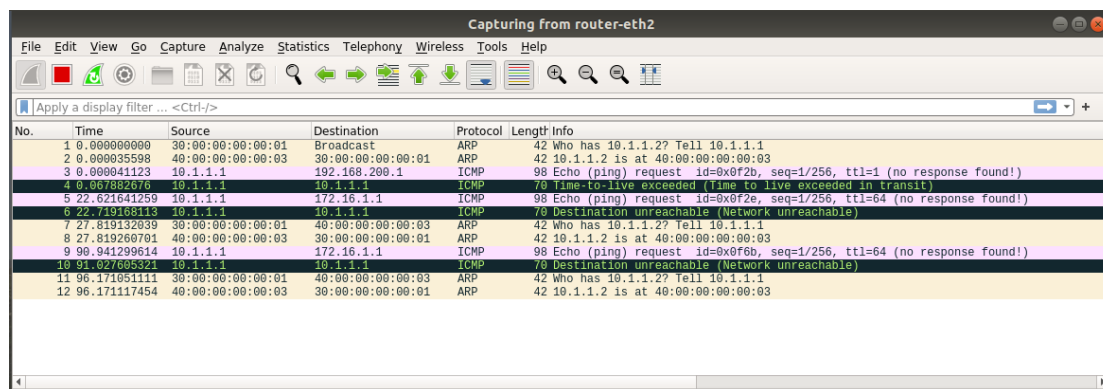
接下来通过测试样例对我们实现的路由器功能进行检测, 结果如下:

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, 从客户端 ping server2 终端的 IP 地址 192.168.200.1, 运行 `mininet>client ping -c 1 t 1 192.168.200.1`, 再从客户端 ping 一个不存在的 IP 地址 172.16.1.1, 通过 wireshark

捕获到的路由器 Eth2 端的抓包结果如下：



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	0.000035598	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	0.000041123	10.1.1.1	192.168.200.1	ICMP	98	Echo (ping) request id=0x0f2b, seq=1/256, ttl=1 (no response found!)
4	0.000022076	10.1.1.1	10.1.1.1	ICMP	76	Time to live exceeded (time to live exceeded in transit)
5	22.621641259	10.1.1.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x0f2b, seq=1/256, ttl=64 (no response found!)
6	22.719168113	10.1.1.1	10.1.1.1	ICMP	76	Destination unreachable (Network unreachable)
7	27.819132839	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	Who has 10.1.1.2? Tell 10.1.1.1
8	27.819260701	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
9	99.941299614	10.1.1.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x0f0b, seq=1/256, ttl=64 (no response found!)
10	99.942100832	10.1.1.1	10.1.1.1	ICMP	76	Destination unreachable (Network unreachable)
11	96.171051111	30:00:00:00:00:01	40:00:00:00:00:03	ARP	42	Who has 10.1.1.2? Tell 10.1.1.1
12	96.17117454	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

其中 eth2 端先发送了一个 ARP 请求，这是由于路由器收到 client 发给 server2 的数据包时，不知道 server2 的 MAC 地址导致的；接着路由器发现这个数据包的 TTL 减为 0，因此要从 eth2 端口向源 IP 地址 client 发送一个 ICMP 错误消息；最后路由器发现 172.16.1.1 这个 IP 地址不在转发表内，因此要再向 client 发送一个 ICMP 错误消息。

### 三、核心代码

本次实验在实验三和实验四中的 `my_router.py` 文件基础上进一步改进并增加功能，其中主要部分代码已在实验过程中截图展示并解释说明，具体代码提交到 Github Classroom 中。

### 四、总结与反思

本次实验在实验三和实验四的基础上，进一步改进并增加了路由器的响应 ICMP 和发送错误信息等功能，至此我们的路由器已经完成！回顾三个阶段以来（越写越长）越来越充实的代码，成就感还是满满滴！通过这个阶段对路由器的模拟，抽象死板的课本知识变得更加形



象和具体化，也进一步加深了我对这部分知识的理解与思考。希望本学期最后的两个实验也可以顺利完成！👉