

# 路由器转发实验报告

李昊宸

2017K8009929044

## (一) 路由器转发机制实现

### 一、实验内容

1. 运行给定网络拓扑(router\_topo.py), 在 h1 上进行 ping 实验

Ping 10.0.1.1 (r1), 能够 ping 通

Ping 10.0.2.22 (h2), 能够 ping 通

Ping 10.0.3.33 (h3), 能够 ping 通

Ping 10.0.3.11, 返回 ICMP Destination Host Unreachable

Ping 10.0.4.1, 返回 ICMP Destination Net Unreachable

2.

1) 构造一个包含多个路由器节点组成的网络

手动配置每个路由器节点的路由表

有两个终端节点, 通过路由器节点相连, 两节点之间的跳数不少于 3 跳, 手动配置其默认路由表

2) 连通性测试

终端节点 ping 每个路由器节点的入端口 IP 地址, 能够 ping 通

3) 路径测试

在一个终端节点上 traceroute 另一节点, 能够正确输出路径上每个节点的 IP 信息

### 二、实验流程

#### 1. 搭建实验环境

include: 相关头文件

scripts: 禁止协议栈的数据包处理

main.c: 路由器的代码实现, 编译后在路由器结点上运行

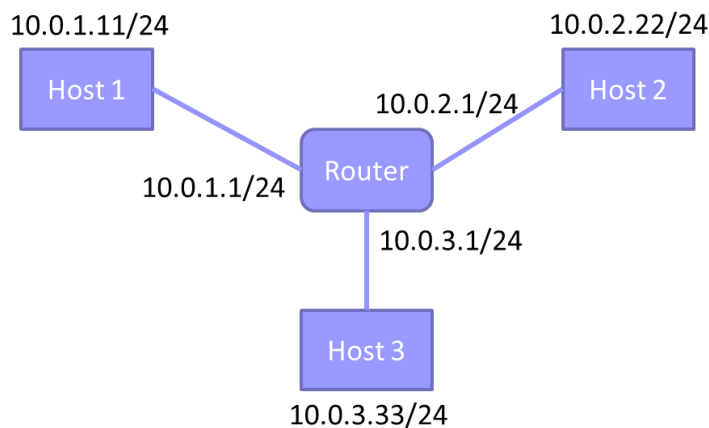
arp.c: 发送 ARP 请求和应答

arpcache.c: ARP 缓存相关操作

icmp.c: 发送 ICMP 数据包

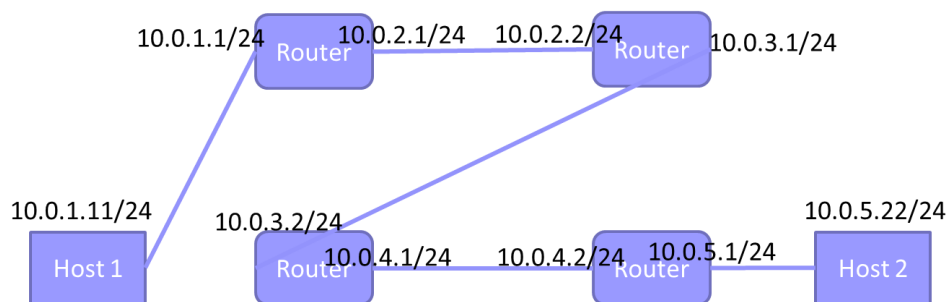
ip\_base.c: 前缀查找和发送 IP 数据包

packet.c: 发送数据包函数  
 rtable.c: 路由表相关  
 rtable\_internal.c: 从协议栈中读取路由条目  
 ip.c: 处理 IP 数据包, 包括转发  
 router\_topo.py: 实现如下图的四节点拓扑



图一 四节点网络拓扑

my\_topo.py: 实现如下图的六节点拓扑



图二 六节点网络拓扑

### 路由器需要维护两个表：路由表和 ARP 缓存

**路由表:** routing table, 或称路由择域信息库 (RIB, Routing Information Base), 是一个存储在路由器或者联网计算机中的电子表格 (文件) 或类数据库。路由表存储着指向特定网络地址的路径 (在有些情况下, 还记录有路径的路由度量值)。路由表中含有网络周边的拓扑信息。

**ARP 高速缓存:** ARPcache, 由最近的 ARP 项组成的内在中的一个临时表每个主机或者路由器都有一个 ARP 高速缓存表。它用来存放最近 Internet 地址到硬件地址之间的映射记录。高速缓存表中每一项的生存时间都是有限的, 起始时间从被创建时开始计算的。

### 路由器路由流程:

1. 路由器某端口收到一个包后, 解析该包以太网首部的协议类型, 如果是 IP, 就跳转到 2; 如果是 ARP, 就跳转到 14。
2. 收到的包是 IP, 调用对 IP 包的解析过程, 转到 3。
3. 首先解析 IP 首部的目的 IP 地址, 如果等于本端口 IP, 转到 4; 否则, 转到 5。
4. 目的 IP 等于本端口 IP, 说明该包要与本端口进行交互。解析该包的 ICMP 首部类型。如果类型是 ping, 就调用 ICMP 响应函数对源主机进行响应; 否则将该包丢弃。
5. 目的 IP 不等于本端口 IP, 说明要转发该包, 转到 6。

6. 在路由表中使用最长前缀匹配查找目的 IP。如果找到对应表项，转到 7；否则调用 ICMP 响应函数回复源主机网络不可达。

7. 找到对应表项，先将 IP 首部的 ttl 减 1。如果减 1 后等于 0，说明生存时间耗尽，调用 ICMP 响应函数回复源主机生存期耗尽；否则转到 8。

8. 重新计算校验和并写入 IP 首部，转到 9。

9. 查看表项中记录的下一跳网关 IP。如果为全 0，说明该表项记录的端口与目的 IP 在同一个网段内，转 10；否则说明该路由器任何一个端口都不与目的 IP 在同一网段，该包需要被转发到下一跳网关，转到 12。

10. 修改以太网首部中源 mac 地址为端口 mac 地址。在 ARP 缓存中查找目的 IP 对应的表项，如果找到，就将表项记录的映射 mac 地址填写到以太网首部中目的 mac 地址，然后从该端口将包发送出去；否则，转 11。

11. 该目的 IP 到 mac 地址的映射未被保存在 ARP 缓存中，但该 IP 与端口处于同一个网段，就将该包挂起，并从该端口发送 ARP 请求。每隔 1s，如果没有收到 ARP 应答，就重发 1 次。如果超过 5 次仍没有收到应答，就将该包丢弃，并调用 ICMP 响应函数，从收到该包的端口回应源主机，目的主机不可达。如果收到 ARP 应答，就准备好所有因等待该 ARP 应答而挂起的包，将应答中的映射 mac 地址填写到以太网首部中目的 mac 地址，然后从该端口将包发送出去。

12. 修改以太网首部中源 mac 地址为端口 mac 地址。在 ARP 缓存中查找下一跳网关 IP 对应的表项，如果找到，就将表项记录的映射 mac 地址填写到以太网首部中目的 mac 地址，然后从该端口将包发送出去；否则，转 13。

13. 下一跳网关 IP 到 mac 地址的映射未被保存在 ARP 缓存中，但该 IP 与端口处于同一个网段，就将该包挂起，并从该端口发送 ARP 请求。每隔 1s，如果没有收到 ARP 应答，就重发 1 次。如果超过 5 次仍没有收到应答，就将该包丢弃，并调用 ICMP 响应函数，从收到该包的端口回应源主机，目的主机不可达。如果收到 ARP 应答，就准备好所有因等待该 ARP 应答而挂起的包，将应答中的映射 mac 地址填写到以太网首部中目的 mac 地址，然后从该端口将包发送出去。

14. 收到的包是 ARP，解析 ARP 首部。如果目的 IP 不是端口 IP，说明源主机要交互的对象不是该端口，将该包丢弃；否则，转到 15。

15. 解析 ARP 首部，如果代码是 ARP 请求，就从该端口发送 ARP 应答；如果是 ARP 应答，就将应答中 IP→mac 的映射关系添加到 ARP 缓存中。

主机请求交互流程：

1. 源主机在发起通信之前，将自己的 IP 与目的主机的 IP 进行比较，如果两者位于同一网段（用子网掩码计算后具有相同的网络号），那么源主机直接向目的主机发送 ARP 请求，在接收到目的主机的 ARP 应答后获取对方 MAC 地址，然后用对方的 MAC 地址作为目标 MAC 地址进行报文发送，位于同一网段的主机互访时属于这种情况，这是互联的交换机做二层转发。

2. 当源主机判断目的主机与自己位于不同网段时，它会通过网关来提交报文，即发送 ARP 请求来获取网关 IP 地址对应的 MAC。通常来讲，每个主机也有自己的路由表，查询路由表一般情况下都会选择默认网关。在得到网关的 ARP 应答后，用网关 MAC 作为报文目的 MAC 进行报文发送，但是，报文的源 IP 是源主机 IP，目的 IP 依然是目的主机 IP。

## 2. 启动脚本

### 1) 四节点拓扑 ROUTER 测试


```
make all
sudo python router_topo.py
mininet> xterm h1 h2 h3 r1
r1# ./router
h1# ping 10.0.1.1 -c 6
h1# ping 10.0.2.22 -c 6
h1# ping 10.0.3.33 -c 6
h1# ping 10.0.3.11 -c 6
h1# ping 10.0.4.1 -c 6
mininet> quit
```

### 2) 六节点拓扑 ROUTER 测试

```
make all
sudo python my_topo.py
mininet> xterm h1 h2 r1 r2 r3 r4
r1# ./router
r2# ./router
r3# ./router
r4# ./router
h1# ping 10.0.1.1 -c 6
h1# ping 10.0.2.2 -c 6
h1# ping 10.0.3.2 -c 6
h1# ping 10.0.4.2 -c 6
h1# ping 10.0.5.22 -c 6
h1# traceroute 10.0.5.22 -m 6
mininet> quit
```

### 三、实验结果及分析

#### 1. 实验结果



```
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.1.1 -c 4
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.131 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.638 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.650 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.076 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3075ms
rtt min/avg/max/mdev = 0.076/0.373/0.650/0.272 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.2.22 -c 4
PING 10.0.2.22 (10.0.2.22) 56(84) bytes of data.
64 bytes from 10.0.2.22: icmp_seq=1 ttl=63 time=1.13 ms
64 bytes from 10.0.2.22: icmp_seq=2 ttl=63 time=0.167 ms
64 bytes from 10.0.2.22: icmp_seq=3 ttl=63 time=0.218 ms
64 bytes from 10.0.2.22: icmp_seq=4 ttl=63 time=0.170 ms

--- 10.0.2.22 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3045ms
rtt min/avg/max/mdev = 0.167/0.422/1.136/0.413 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.3.33
PING 10.0.3.33 (10.0.3.33) 56(84) bytes of data.
64 bytes from 10.0.3.33: icmp_seq=1 ttl=63 time=1.05 ms
64 bytes from 10.0.3.33: icmp_seq=2 ttl=63 time=0.158 ms
64 bytes from 10.0.3.33: icmp_seq=3 ttl=63 time=0.212 ms
64 bytes from 10.0.3.33: icmp_seq=4 ttl=63 time=0.154 ms
64 bytes from 10.0.3.33: icmp_seq=5 ttl=63 time=0.155 ms
64 bytes from 10.0.3.33: icmp_seq=6 ttl=63 time=0.155 ms
64 bytes from 10.0.3.33: icmp_seq=7 ttl=63 time=0.152 ms
^C
--- 10.0.3.33 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6117ms
rtt min/avg/max/mdev = 0.152/0.292/1.059/0.313 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.3.11 -c 4
PING 10.0.3.11 (10.0.3.11) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Host Unreachable
From 10.0.1.1 icmp_seq=2 Destination Host Unreachable
From 10.0.1.1 icmp_seq=3 Destination Host Unreachable
From 10.0.1.1 icmp_seq=4 Destination Host Unreachable

--- 10.0.3.11 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3061ms
pipe 4
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.4.1 -c 4
PING 10.0.4.1 (10.0.4.1) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Net Unreachable
From 10.0.1.1 icmp_seq=2 Destination Net Unreachable
From 10.0.1.1 icmp_seq=3 Destination Net Unreachable
From 10.0.1.1 icmp_seq=4 Destination Net Unreachable

--- 10.0.4.1 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3055ms

root@CN-VirtualBox:/mnt/shared/07-router#
```

图三 四节点拓扑路由结果

可以看到，h1 节点 ping 网关以及其他两个节点都能 ping 通，如果 ping 不存在的节点会报 Host Unreachable，ping 不存在的网段会报 Net Unreachable。

```

"Node: h1"
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.1.1 -c 3
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=1.00 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.124 ms

--- 10.0.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2011ms
rtt min/avg/max/mdev = 0.122/0.415/1.001/0.414 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.2.2 -c 3
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=63 time=0.814 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=63 time=0.168 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=63 time=0.644 ms

--- 10.0.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2048ms
rtt min/avg/max/mdev = 0.168/0.542/0.814/0.273 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.3.2 -c 3
PING 10.0.3.2 (10.0.3.2) 56(84) bytes of data.
64 bytes from 10.0.3.2: icmp_seq=1 ttl=62 time=2.87 ms
64 bytes from 10.0.3.2: icmp_seq=2 ttl=62 time=0.812 ms
64 bytes from 10.0.3.2: icmp_seq=3 ttl=62 time=0.631 ms

--- 10.0.3.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2037ms
rtt min/avg/max/mdev = 0.631/1.437/2.870/1.016 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.4.2 -c 3
PING 10.0.4.2 (10.0.4.2) 56(84) bytes of data.
64 bytes from 10.0.4.2: icmp_seq=1 ttl=61 time=1.29 ms
64 bytes from 10.0.4.2: icmp_seq=2 ttl=61 time=0.520 ms
64 bytes from 10.0.4.2: icmp_seq=3 ttl=61 time=0.439 ms

--- 10.0.4.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2021ms
rtt min/avg/max/mdev = 0.439/0.752/1.298/0.387 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.5.22 -c 3
PING 10.0.5.22 (10.0.5.22) 56(84) bytes of data.
64 bytes from 10.0.5.22: icmp_seq=1 ttl=60 time=0.865 ms
64 bytes from 10.0.5.22: icmp_seq=2 ttl=60 time=0.977 ms
64 bytes from 10.0.5.22: icmp_seq=3 ttl=60 time=0.562 ms

--- 10.0.5.22 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2029ms
rtt min/avg/max/mdev = 0.562/0.801/0.977/0.176 ms
root@CN-VirtualBox:/mnt/shared/07-router# ping 10.0.5.12 -c 6
PING 10.0.5.12 (10.0.5.12) 56(84) bytes of data.
From 10.0.4.2 icmp_seq=1 Destination Host Unreachable
From 10.0.4.2 icmp_seq=2 Destination Host Unreachable
From 10.0.4.2 icmp_seq=3 Destination Host Unreachable
From 10.0.4.2 icmp_seq=4 Destination Host Unreachable
From 10.0.4.2 icmp_seq=5 Destination Host Unreachable
From 10.0.4.2 icmp_seq=6 Destination Host Unreachable

--- 10.0.5.12 ping statistics ---
6 packets transmitted, 0 received, +6 errors, 100% packet loss, time 5115ms
pipe 6

root@CN-VirtualBox:/mnt/shared/07-router# traceroute 10.0.5.22 -m 10
traceroute to 10.0.5.22 (10.0.5.22), 10 hops max, 60 byte packets
 1 10.0.1.1 (10.0.1.1) 0.120 ms 0.024 ms 0.017 ms
 2 10.0.2.2 (10.0.2.2) 0.187 ms 0.191 ms 0.193 ms
 3 10.0.3.2 (10.0.3.2) 1.248 ms 1.240 ms 1.235 ms
 4 10.0.4.2 (10.0.4.2) 1.229 ms 1.218 ms 1.210 ms
 5 10.0.5.22 (10.0.5.22) 1.201 ms 1.190 ms 1.182 ms

```

图四 六节点拓扑路由结果

可以看到，h1 节点 ping 网关以及其他三个路由器以及 h2 节点都能 ping 通，如果 ping 不存在的节点会报 Host Unreachable，traceroute h2 会返回正确的路由途径。

## (二) 实验代码详解

本次实验代码过长，不再赘述代码的内容。

### 一、arp.c: 发送 ARP 请求和应答

(1) void handle\_arp\_packet(iface\_info\_t \*iface, char \*packet, int len): 处理一个收到的 ARP 包

首先从收到的 packet 中提取出属于 arp 报文的部分。

如果 arp 报文中 target protocol address 经过 ntohs 转换后与本地端口中保存的 IP 不相等的话，说明该 arp 报文并不是发给自己的，丢弃。

否则，进行如下的操作：

检查操作码，经过 ntohs 转换后如果为 Request 报文，说明是请求获得自己 mac 地址的服务，就调用 arp\_send\_reply 将自己的信息回复给源主机。如果为 Reply 报文，说明是自己的 arp 请求报文被回应，需要将收到 packet 中的 sender protocol address 和 sender hardware address 调用 arpcache\_insert 保存到 ARP 缓存中。

(2) void arp\_send\_reply(iface\_info\_t \*iface, struct ether\_arp \*req\_hdr): 发送 ARP 应答

首先分配一块内存 packet，大小为一个以太网报头加一个 ARP 报文的大小。

对该 packet 做预处理，装填好：ether\_shost 为端口的 mac 地址，ether\_type 为 ARP，arp\_header 为 htons (0x0001)，arp\_protocol 为 htons (0x0800)，hardware address length 为 6，protocol address length 为 4，arp\_sender hardware address 为端口的 mac 地址，arp\_sender protocol address 为端口的 IP。

然后装填：ether\_dhost 为收到 arp 报文中的 arp\_sender hardware address，arp\_op 为 htons (ARPOP\_REPLY)，target hardware address 为收到 arp 报文中的 arp\_sender hardware address，target protocol address 为收到 arp 报文中的 arp\_sender protocol address。

最后将 packet 从该端口发送出去。

(3) void arp\_send\_request(iface\_info\_t \*iface, u32 dst\_ip):

### 发送 ARP 请求

首先分配一块内存 packet，大小为一个以太网报头加一个 ARP 报文的大小。

对该 packet 做预处理，装填好：ether\_shost 为端口的 mac 地址，ether\_type 为 ARP，arp\_header 为 htons (0x0001)，arp\_protocol 为 htons (0x0800)，hardware address length 为 6，protocol address length 为 4，arp\_sender hardware address 为端口的 mac 地址，arp\_sender protocol address 为端口的 IP。

然后装填：ether\_dhost 为 0xFFFFFFFFFFFFFFFF，arp\_op 为 htons (ARPOP\_REQUEST)，target hardware address 为 0，target protocol address 为 dst\_ip。

最后将 packet 从该端口发送出去。



## 二、arpcache.c:ARP 缓存相关操作

### (1) int arpcache\_lookup(u32 ip4, u8 mac[ETH\_ALEN]): 查询 ARP 缓存

首先获取 ARP 缓存的互斥锁。

然后遍历查找 ARP 缓存的所有表项, 如果某一表项的 valid 值为 1, 且表项中的 IP 地址与要查找的 IP 地址相同, 就将该表项中缓存的 mac 地址保存到 mac[ETH\_ALEN] 中, 释放互斥锁, 返回 1。

如果没找到, 就释放互斥锁, 返回 0。

### (2) void arpcache\_append\_packet(iface\_info\_t \*iface, u32 ip4, char \*packet, int len): 查询 ARP 缓存失败, 将要发送的包挂起, 发送 ARP 请求

首先分配一个 cached\_pkt 类型的 cached\_packet 节点, 缓存 packet 和 len。

获取互斥锁。

随后调用 list\_for\_each\_entry 宏查找链表 arpcache.req\_list, 链表中的节点类型为 arp\_req。

如果找到链表中某个节点的等待 ARP 回复 IP 地址与 ip4 相同, 且转发数据包的端口等于 iface, 那么就将 cached\_packet 调用 list\_add\_tail 挂到该 arp\_req 节点中的 cached\_packets 末端。

如果没找到, 就新建这样一个 arp\_req 节点, 设置其发送 ARP 请求的端口为 iface, 等待 ARP 回复的 IP 地址为 ip4, 发送 ARP 请求时间为当前时间, 重试次数为 0, 初始化 cached\_packets 链表头, 调用 list\_add\_tail 将 cached\_packet 挂载到 cached\_packets, 调用 list\_add\_tail 将该 arp\_req 节点挂到 arpcache.req\_list 上, 然后调用 arp\_send\_request 从 iface 端口发送 ARP 请求 ip4。

最后释放互斥锁。

### (3) void arpcache\_insert(u32 ip4, u8 mac[ETH\_ALEN]): 收到 ARP 应答后将 IP->MAC 映射写到缓存中, 并将等待该映射的数据包发送出去

首先获取互斥锁。

随后对 ARP 缓存表项进行遍历, 查找是否有 valid 项为 0 的表项 (即超时)。如果查找到 1 个, 就将该表项记录的 IP 替换为 ip4, MAC 地址替换为 mac, 添加时间修改为当前时间, valid 修改为 1, 停止遍历。

如果没有找到, 就要随机挑选一个表项删除: 随机生成一个 0 到 31 之间的整数 index, 修改 ARP 缓存表中 index 号表项: IP 替换为 ip4, MAC 地址替换为 mac, 添加时间修改为当前时间, valid 修改为 1。

之后调用 `list_for_each_entry_safe` 宏对 `arpcache.req_list` 进行遍历：

查找链表中是否有 `arp_req` 节点的 IP 与 `ip4` 相等。如果有，就继续调用 `list_for_each_entry_safe` 遍历其下链表 `cached_packets`：

将链表下每一个 `cached_pkt` 节点中 `packet` 项的首部 `ether_dhost` 填充为 `mac`，将该包从 `arp_req` 节点中记录的端口 `iface` 发送出去，然后释放该 `cached_pkt` 节点。

处理完 `cached_packets` 链表后，调用宏 `list_delete_entry` 将该 `arp_req` 节点从 `arpcache.req_list` 链表中删除，释放节点空间。

最后释放互斥锁。

(4) `void *arpcache_sweep(void *arg)`：如果 ARP 缓存表中某表项存在 15s 以上，就标记为失效；对于等待 ARP 应答而挂起的待发包，如果 ARP 请求发送超过 1s 并且还没收到应答，就重发请求。如果发送 5 次请求后仍未收到应答，就向这些包的源主机发送 ICMP `DEST_HOST_UNREACHABLE`，并丢弃这些包。

该程序通过 `sleep(1)` 控制，每 1 秒执行一次：

首先获取互斥锁，获取当前时间 `now`。

遍历 ARP 缓存表，如果某个表项的 `valid` 为 1，且当前时间与表项记录的入表时间之差大于 15s，就将其 `valid` 修改为 0。

随后调用宏 `list_for_each_entry_safe` 遍历 `arpcache.req_list`：

首先判断当前遍历到的 `arp_req` 节点的发送次数是否超过 5 次：

如果超过 5 次，就要调用宏 `list_for_each_entry_safe` 对该节点下的 `cached_packets` 链表进行遍历：

首先释放互斥锁。随后调用 `icmp_send_packet` 向各个包的源主机发送 ICMP `HOST` 不可达报文 (`type: 3 code: 1`, ARP 查询失败)。然后获取互斥锁，释放该节点。注意，在此处对锁的释放和获取是有意义的，因为在 `icmp_send_packet` 中控制流可能会通过调用到达 `arpcache_append_packet` 函数，从而发起对锁的另一次获取，导致死锁。

对 `cached_packets` 的遍历结束后，从 `arpcache.req_list` 中删除当前 `arp_req` 节点，并释放节点空间。使用 `continue` 语句跳到外层遍历的下一个对象。

如果没超过 5 次，并且 `now` 减 `arp_req` 节点记录的发送时间大于 1s，就调用 `arp_send_request` 重新发送请求，将发送时间修改为 `now`，发送次数加 1。

对 `arpcache.req_list` 的遍历结束后，释放互斥锁。

### 三、icmp.c:发送 ICMP 数据包

(1) void icmp\_send\_packet(const char \*in\_pkt, int len, u8 type, u8 code): 发送 ICMP 数据包

首先用 in\_pkt\_etherhead 指向 in\_pkt 的以太网首部, 调用 packet\_to\_ip 将 in\_pkt\_IPhead 指向 in\_pkt 的 IP 首部。

随后计算要发送 packet 的空间:

如果是要回复 ping 本端口的数据包, 那么大小为 len;

如果是其他, 大小为 ETHER\_HDR\_SIZE + IP\_BASE\_HDR\_SIZE + ICMP\_HDR\_SIZE + IP\_HDR\_SIZE(in\_pkt\_IPhead) + 8: 要发送的 packet 有自己的以太网头部, 自己的标准 IP 首部 (20 字节), 标准 ICMP 首部 (type+code+Checksum+4, 其中后 4 字节要设置为 0), 以及 Rest of ICMP header (该部分填充 in\_pkt 的 IP 首部和随后的 8 字节)

按照计算出的空间大小分配 packet。

设置 packet 的以太网头: ether\_dhost 设置为 in\_pkt\_etherhead 的 ether\_shost, ether\_shost 设置为 in\_pkt\_etherhead 的 ether\_dhost, ether\_type 设置为 htons(ETH\_P\_IP)。

调用 packet\_to\_ip 找到 packet 的 IP 首部, 然后设置 IP 首部:

调用 longest\_prefix\_match 在 ARP 缓存中查找经过 ntohl() 转换的 in\_ihdr 的 sender IP address, 查找到表项后, 调用 ip\_init\_hdr 对 packet 的 IP 首部进行设置:

设置 version 为 4, ihl 为 5 (对应首部大小 20 字节), tos 为 0, tot\_len 为 packet 的大小减以太网首部的大小, id 为 rand(), frag\_off 为 htons(IP\_DF), ttl 为 64 (DEFAULT), protocol 为 1 表示为 ICMP, saddr 为 htonl() 处理后的 ARP 表项中的端口 IP (表示该条 ICMP 由该端口发出), daddr 为 htonl() 处理后的 [ntohl() 处理后的从 in\_pkt\_IPhead 提取出的 sender IP address], 校验和为 ip\_checksum(ip)。

之后设置 ICMP 首部: 类型为 type, 代码为 code。

随后查找到 packet 的 Rest of ICMP header 的起始处, 对内容进行修改:

如果是响应 ping 的应答, 就将 in\_pkt 相同位置起始处的剩余内容都 copy 过来。

否则, 就先设置开头 4 个字节为 0, 剩余部分将 in\_pkt 的 IP 首部以及之后的 8 个字节 (ICMP 首部) copy 过来。

最后设置 ICMP 首部的校验和为 icmp\_checksum(packet\_IChead, pkt\_len - ETHER\_HDR\_SIZE - IP\_BASE\_HDR\_SIZE)。之所以这样计算, 是因为当要回复的 in\_pkt 是 PING 时, PING 发送的报文的 IP Head 大小为 20; 要回复的如果不是 PING, 那么 IP Head 会初始化为 20。也就是说, 无论应答谁, 该报文的 IP Head 都是 20 (IP\_BASE\_HDR\_SIZE)。

最后调用 ip\_send\_packet 将 packet 发送出去。

#### 四、icmp.c:最长前缀查找和发送 IP 数据包

##### (1) `rt_entry_t *longest_prefix_match(u32 dst)`: 最长前缀查找

调用 `list_for_each_entry` 遍历转发表 `rtable`, 如果 `(rt->dest & rt->mask) == (dst & rt->mask)`, 并且 `rt->mask` 比当前找到的最长前缀掩码更大, 就更新最长前缀掩码的值和最长前缀掩码对应的表项。遍历结束后, 返回最长前缀掩码对应的表项。

##### (2) `void ip_send_packet(char *packet, int len)`: 发送 IP 数据

##### 包 (只用于自己发送 ICMP 报文, 而不是转发 IP 数据包)

首先找到 `packet` 中 IP 首部中的 `dest IP address`, 用 `ntohl()` 处理后记为 `dst_ip`。调用 `longest_prefix_match` 函数查找 `dst_ip` 的表项。

返回的表项的下一跳网关地址 `gw` 若不为 0, 说明该路由器任何端口的 IP 都与目的 IP 不在同一网段, 就调用 `iface_send_packet_by_arp(rt->iface, rt->gw, packet, len)`, 含义为从该表项记录的端口将 ICMP 报文发出, 以太网报头的目的 IP 为 `rt->gw`, 源 IP 为发送端口的 IP。

如果 `gw` 为 0, 说明该表项记录的下一跳网关为空, 即目标 IP 与该表项记录的端口在同一个网段, 就调用 `iface_send_packet_by_arp(rt->iface, dst_ip, packet, len)`, 含义为从该表项记录的端口将 ICMP 报文发出, 以太网报头的目的 IP 为 `dst_ip`, 源 IP 为发送端口的 IP。

## 五、ip.c:处理 IP 数据包，包括转发

(1) void handle\_ip\_packet(iface\_info\_t \*iface, char \*packet, int len): 如果收到的包目标是本路由器端口，并且 ICMP 首部 type 为 8, 就回应 ping 的 ICMP 报文, 否则丢弃; 如果目标不是本路由器, 就转发。

首先从 packet 的 IP 首部中找到 dest IP address, 用 ntohl() 处理后记为 IP\_dest\_addr。将 IP\_dest\_addr 与当前端口 IP 比较:

如果相同, 说明是发给本路由器端口的 ICMP 包。解析该包的 ICMP type 字段:

如果为 8, 说明为 PING 本端口的数据包, 就调用 icmp\_send\_packet(packet, len, ICMP\_ECHOREPLY, 0), type: 0 code: 0 应答。

如果不为 8 的话就将该包丢弃。

如果不相同, 说明是需要转发的包。首先在路由表中最长前缀匹配查找路由表项:

如果没找到 (路由表查找失败, 没有该网段), 就调用 icmp\_send\_packet(packet, len, ICMP\_DEST\_UNREACH, ICMP\_NET\_UNREACH), 发送 type: 3 code: 1 的 ICMP 报文。

如果找到了, 首先将 IP 首部的 ttl 减 1。

如果 ttl 变为 0 (生存期耗尽), 就调用 icmp\_send\_packet(packet, len, ICMP\_TIME\_EXCEEDED, ICMP\_EXC\_TTL), 发送 type: 11 code: 0 的 ICMP 报文, 返回。

否则就重新计算校验和, 填写到 checksum 中。返回的表项的下一跳网关地址 gw 若不为 0, 说明该路由器任何端口的 IP 都与目的 IP 不在同一网段, 就调用 iface\_send\_packet\_by\_arp(rt->iface, rt->gw, packet, len), 含义为从该表项记录的端口将 ICMP 报文发出, 以太网报头的目的 mac 地址为 rt->gw 对应的 mac 地址, 源 mac 地址为发送端口的 mac 地址。

如果 gw 为 0, 说明该表项记录的下一跳网关为空, 即目标 IP 与该表项记录的端口在同一个网段, 就调用 iface\_send\_packet\_by\_arp(rt->iface, IP\_dest\_addr, packet, len), 含义为从该表项记录的端口将 ICMP 报文发出, 以太网报头的目的 mac 地址为 IP\_dest\_addr 对应的 IP 地址, 源 mac 地址为发送端口的 mac 地址。