第八周实验报告

路由器转发实验

2015K8009922021

李一苇

一、实验内容

- 在主机上安装 arptables, iptables，用于禁止每个节点的相应功能
- 给定网络拓扑（router_topo.py）以及节点的路由表配置，实现路由器的转发功能，使得各节点之间能够连通并传送数据
- 构造一个包含多个路由器节点组成的网络（traceroute_topo.py）并进行连通性测试和路径测试

二、实验流程

本实验按照以下流程进行:

1. 安装 arptables 和 iptables，直接执行 sudo apt install arptables iptables
   测试效果为：ping 自己子网内的服务器不再由主机自己在局域网内进行 arp 查询和转发，而是发送给路由器，由路由器决定
2. 实现单路由器拓扑的正常工作:
   第一步，实现 ip 包转发
   第二步，实现 arp 包处理和 arpcache 维护
   第三步，实现 icmp 包发送
   实现某一步时，后面的步骤先空着，但在函数内发送 printf 消息，代表成功进入函数，到后面再实现。

完整的调用关系如下:

由 handle_packet(main.c)里对 ETHER 包头分类

如果收到非 IP、ARP 包，报错，不处理;

如果收到 IP 包，转入 handle_ip_packet(ip_forwarding.c)。函数对 ip 包的目的 ip 地址分析:

- 如果是发给当前网卡的 IP，说明是 ping 本网卡，在 ICMP 一节处理;
- 否则转入 ip_forwarding_packet(ip_forwarding.c)：进行 ip 包转发
  - 用最长前缀查找 FIB 表得到下一跳 ip 地址和发送端口，如果查找失败，则发送 ICMP 网络不可达
  - 否则，修改 ip 包 ttl 值，如果不大于 0，则发送 ICMP：TTL 耗尽
  - 否则，重新计算 ip 包 checksum，转入 iface_send_packet_by_arp(arp.c)

- 如果 ip 地址在 arpcache 中找到，则用 iface_send_packet 发往该 mac 地址
- 否则，转入 arpcache_append_packet(arpcache.c)
  - 查询在 cache 中是否有同一 ip 和发送端 iface 的 arp_req 请求，如果有，直接挂在其 cached_packets 后面
  - 否则，新开一个 arp_req，转入 arp_send_request(arp.c)广播发送 arp 请求

如果收到 ARP 包，转入 handle_arp_packet(arp.c)：：

- 如果收到广播的 arp 请求：
  - 如果本机是 arp 请求的目的主机 ip，则填好本机 mac 值，单独发送 arp 应答
  - 否则，用 arp 请求的源主机(ip, mac)更新本机 arpcache 中的 ip
- 如果收到 arp 应答，必定之前发送过 arp 请求，转入 arpcache_insert(arpcache.c)
  - 把对应的 arp_req 包中的所有 packet，以新收到的 mac 地址为目的地址，用 iface_send_packet 发送

与此同时，存在 arpcache_sweep 线程(arpcache.c)

每秒钟对所有 arp_req 重新发 arp 请求

如果某 arp_req 重试达到 5 次，发送 ICMP 主机不可达

3. 检验多路由器拓扑是否也能正常工作

三、debug 过程

1. 最长公共前缀匹配失败，匹配到没有的条目：

后来用 gdb 看了过程才发现，匹配的关系式(((dst&&entry->mask) == (entry->dest&&entry->mask)))写错了，应该是位运算&而非逻辑运算&&

2. 处理中的段错误：

- Arp 处理：仍用 gdb 调试，发现是内存访存问题，原来在 arp_send_request(arp.c)里单独 malloc 了 ether 包头和 arp 包，最后 packet 只指向 ether 包，造成内存空间不连续，访存异常。应该先分配连续地址给 packet，再指派各级指针
- 在 icmp_send_packet(icmp.c)内，组装 icmp 包的内存有问题，导致 iface_send_packet 读取时段错误：这一句 memcpy(icmp + 4 + 4, in_pkt, len)中涉及对 icmp 指针的偏移，但偏移量又是以 char 为单位，改成 memcpy((char *)icmp + 4 + 4, in_pkt, len)后正常

3. 大于 5 次 retry 后的 ICMP 信息收不到：

用 wireshark 抓包，比对 ref 程序和自己的程序，发现必须由收 packet 的端口及其 ip 发给主机的端口和 ip。但是 arp_req 里不含这些信息，因为已经到了发送端口了。此时重新用最长匹配查找路由表得到这些信息。

4. 多路由器拓扑，只能传一个路由器：
   用 wireshark 抓包发现，r1 发送的 arp 请求 ip 地址有误。不应该直接请求 dst_ip，除非是最后一跳，加了一个判定后正常：
   u32 next_ip = rt_entry_match->gw==0?ip_dst:rt_entry_match->gw;

5. 需要注意的点：
   1） 为什么要存储 arp_req->cached_pkt：防止收到的 arp_reply 不能对应每个 arp_req，造成部分包的死锁：
   详细了解
   https://books.google.ru/books?id=BvQUmsnVpQMC&pg=PA88&lpg=PA88&dq=arp+%E9%87%8D%E8%AF%95&source=bl&ots=f1lnryTLLT&sig=IOWnbFFQLAtam9rrOWkYPcM54rY&hl=en&sa=X&redir_esc=y#v=onepage&q=arp%20%E9%87%8D%E8%AF%95&f=false
   2） host 对 ip、icmp 包的处理由 Linux 网络部分写好，不用自己实现

四、实验结果和分析

1. 依次在 router_topo.py 拓扑环境下执行：

- ☐ 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

返回结果与理论相符：

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=0.199 ms

--- 10.0.1.1 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.199/0.199/0.199/0.000 ms

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=0.133 ms

--- 10.0.2.22 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.133/0.133/0.133/0.000 ms

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=0.092 ms

--- 10.0.3.33 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.092/0.092/0.092/0.000 ms

PING 10.0.3.22 (10.0.3.22) 56(84) bytes of data.

From 10.0.1.1 icmp_seq=1 Destination Host Unreachable

--- 10.0.3.22 ping statistics ---

1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

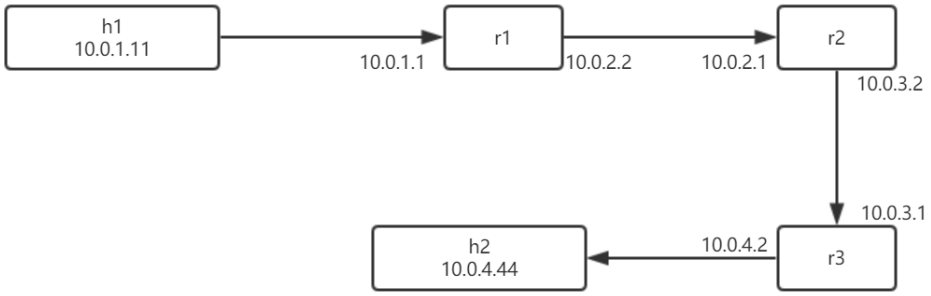
PING 10.0.4.1 (10.0.4.1) 56(84) bytes of data.

From 10.0.1.1 icmp_seq=1 Destination Net Unreachable

--- 10.0.4.1 ping statistics ---

1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

2. 自己写了一个三路由器的拓扑，如下图：



在 h1 上执行:

Ping 10.0.1.1

Ping 10.0.2.1

Ping 10.0.3.1

Ping 10.0.4.44

Traceroute 10.0.4.44

结果如下:

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=0.100 ms

--- 10.0.1.1 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.100/0.100/0.100/0.000 ms

PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.

64 bytes from 10.0.2.1: icmp_seq=1 ttl=63 time=0.623 ms

--- 10.0.2.1 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.623/0.623/0.623/0.000 ms

PING 10.0.3.1 (10.0.3.1) 56(84) bytes of data.

64 bytes from 10.0.3.1: icmp_seq=1 ttl=62 time=0.581 ms

--- 10.0.3.1 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.581/0.581/0.581/0.000 ms

PING 10.0.4.44 (10.0.4.44) 56(84) bytes of data.

64 bytes from 10.0.4.44: icmp_seq=1 ttl=61 time=0.632 ms

--- 10.0.4.44 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.632/0.632/0.632/0.000 ms

traceroute to 10.0.4.44 (10.0.4.44), 30 hops max, 60 byte packets

1 10.0.1.1 (10.0.1.1) 0.360 ms 0.345 ms 0.346 ms

2 10.0.2.1 (10.0.2.1) 3.225 ms 3.224 ms 3.219 ms

3 10.0.3.1 (10.0.3.1) 3.216 ms 3.213 ms 3.207 ms

4 10.0.4.44 (10.0.4.44) 3.198 ms 3.190 ms 3.172 ms