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

1. 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

1. 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。

1. 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;

1. 需要注意的点：
2. 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>

1. 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

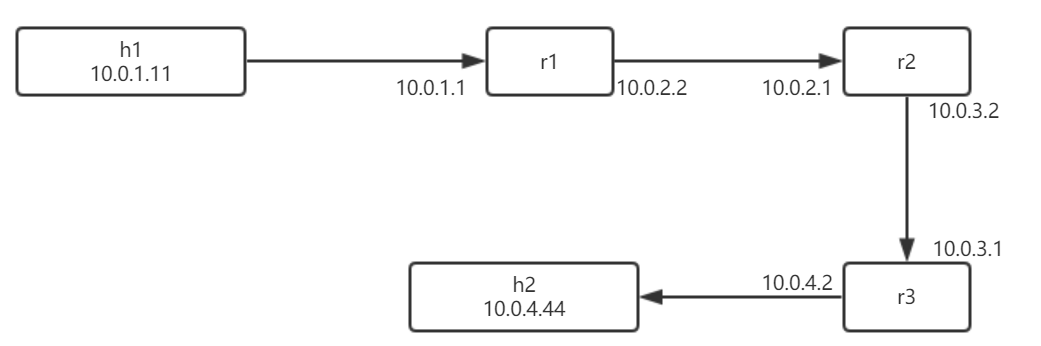
PING 10.0.4.1 (10.0.4.1) 56(84) bytes of data.

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