网络实验报告

雷正宇 2016K8009909005

实验内容

安装arptables, iptables

用于禁止每个节点的相应功能

运行给定网络拓扑(router_topo.py)

路由器节点r1上执行脚本(disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh)禁止协议栈的相应功能

终端节点h1-h3上执行脚本disable_offloading.sh

执行路由程序

在r1上执行路由器程序

在r1中运行./router，进行数据包的处理

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

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

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

实验流程

本次试验代码量较大，仅截取重要部分进行说明。

arp部分

arp 部分有几个重要的处理过程。首先，收到 arp 包时，利用 handle_arp_packet 进行处理：

```
void handle_arp_packet(iface_info_t *iface, char *packet, int len)
{
  struct arp_with_hdr *awh = (struct arp_with_hdr *)packet;
  if (ntohs(awh->arp.arp_op) == ARPOP_REQUEST) {
    if (ntohl(awh->arp.arp_tpa) == iface->ip) {
      arpcache_insert(ntohl(awh->arp.arp_spa), awh->arp.arp_sha);
      arp_send_reply(iface, &awh->arp);
    }
  } else if (ntohs(awh->arp.arp_op) == ARPOP_REPLY) {
    arpcache_insert(ntohl(awh->arp.arp_spa), awh->arp.arp_sha);
  }
}
```

整体过程很简单，如果收到的是 arp 请求，就回应对应的 arp 包，并将新的 mac 条目插入缓存；如果是回应，就执行插入缓存操作，插入过程中会取出对应等待的 packet 进行发送。

这里我用到了自己定义的 struct arp_with_hdr 结构体。利用以太头的固定性，可以使用该结构体来对包含以太头的 arp 包进行更安全的操作，定义如下：

```
struct arp_with_hdr {
    struct ether_header hdr;
    struct ether_arp arp;
} __attribute__((packed));
```

要注意的细节是不能忘了限制对齐，否则发送出去的包会变混乱。

arp 请求和 arp 回应的发送大同小异，过程比较简单：

```
void arp_send_request(iface_info_t *iface, u32 dst_ip)
{
  struct arp_with_hdr awh = {
    .hdr = {
      .ether_dhost = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff},
      .ether_type = htons(ETH_P_ARP)
    },
    .arp = {
      .arp_hrd = htons(ARPHRD_ETHER),
      .arp_pro = htons(ETH_P_IP),
      .arp_hln = 6,
      .arp_pln = 4,
      .arp_spa = htonl(iface->ip),
      .arp_tpa = htonl(dst_ip),
      .arp_op = htons(ARPOP_REQUEST)
    }
  };
```

```
  memcpy(awh.hdr.ether_shost, iface->mac, ETH_ALEN);
  memcpy(awh.arp.arp_sha, iface->mac, ETH_ALEN);
  memset(awh.arp.arp_tha, 0, ETH_ALEN);

  iface_send_packet(iface, (char *)&awh, sizeof(awh));
}

void arp_send_reply(iface_info_t *iface, struct ether_arp *req_hdr)
{
  struct arp_with_hdr awh = {
    .hdr = {
      .ether_type = htons(ETH_P_ARP)
    },
    .arp = {
      .arp_hrd = htons(ARPHRD_ETHER),
      .arp_pro = htons(ETH_P_IP),
      .arp_hln = 6,
      .arp_pln = 4,
      .arp_spa = htonl(iface->ip),
      .arp_tpa = req_hdr->arp_spa,
      .arp_op = htons(ARPOP_REPLY)
    }
  };

  memcpy(&awh.hdr.ether_dhost, req_hdr->arp_sha, ETH_ALEN);
  memcpy(&awh.hdr.ether_shost, iface->mac, ETH_ALEN);

  memcpy(&awh.arp.arp_tha, req_hdr->arp_sha, ETH_ALEN);
  memcpy(&awh.arp.arp_sha, iface->mac, ETH_ALEN);

  iface_send_packet(iface, (char *)&awh, sizeof(awh));
}
```

但像这样编写代码会造成一个难以预知的问题：awh 结构体对象在栈上，而 iface_send_packet 函数一般都在处理完包后把包所在的内存空间释放掉。所以，在不改动 iface_send_packet 的情况下，最好还是用 malloc 分配一段供发送包的内存。

arp 缓存插入的操作稍微繁琐一点，原因在于 arp 缓存的链表结构有两层：

```
void arpcache_insert(u32 ip4, u8 mac[ETH_ALEN])
{
  pthread_mutex_lock(&arpcache.lock);
  struct arp_cache_entry *target = NULL;
  for (int i = 0; i < MAX_ARP_SIZE; i++) {
    if (arpcache.entries[i].valid == 0) {
      target = &arpcache.entries[i];
      break;
    }
  }
```

```
  if (!target) {
    target = &arpcache.entries[rand() % MAX_ARP_SIZE];
  }

  target->ip4 = ntohl(ip4);
  target->added = time(0);
  target->valid = 1;
  memcpy(target->mac, mac, ETH_ALEN);

  struct arp_req *req = NULL, *req_q;
  list_for_each_entry_safe(req, req_q, &arpcache.req_list, list) {
    if (req->ip4 == ntohl(ip4)) {
      struct cached_pkt *pkt = NULL, *pkt_q;
      list_for_each_entry_safe(pkt, pkt_q, &req->cached_packets, list) {
        struct ether_header *eh = (struct ether_header *)(pkt->packet);
        memcpy(eh->ether_dhost, mac, ETH_ALEN);
        iface_send_packet(req->iface, pkt->packet, pkt->len);
        list_delete_entry(&pkt->list);
      }
      list_delete_entry(&req->list);
      free(req);
    }
  }
  pthread_mutex_unlock(&arpcache.lock);
}

void arpcache_append_packet(iface_info_t *iface, u32 ip4, char *packet,
int len)
{
  struct cached_pkt *new_pkt = malloc(sizeof(struct cached_pkt));
  new_pkt->len = len;
  new_pkt->packet = packet;

  struct arp_req *req = NULL;
  pthread_mutex_lock(&arpcache.lock);
  list_for_each_entry(req, &arpcache.req_list, list) {
    if (req->ip4 == ip4 && req->iface == iface) {
      list_add_tail(&new_pkt->list, &req->cached_packets);
      pthread_mutex_unlock(&arpcache.lock);
      return;
    }
  }
  req = malloc(sizeof(struct arp_req));
  req->iface = iface;
  req->ip4 = ip4;
  req->sent = time(0);
  req->retries = 0;
  init_list_head(&req->cached_packets);
  list_add_tail(&new_pkt->list, &req->cached_packets);
```

```
  list_add_tail(&req->list, &arpcache.req_list);
  arp_send_request(iface, ip4);
  pthread_mutex_unlock(&arpcache.lock);
}
```

在 insert 过程中，如果查找到无效的缓存，可以将其替换；如果缓存满了，就随机替换一条。然后将在缓存中等待该映射的数据包，依次填写目的MAC地址，转发出去，并删除掉相应缓存数据包。

更加繁琐的是 sweep 过程，但逻辑上是简单的，这里就不粘贴代码了。

ip部分

收到 ip 包后提取目的 ip 地址：

```
struct iphdr *ip = packet_to_ip_hdr(packet);
struct icmphdr *icmp = (struct icmphdr *)(packet + ETHER_HDR_SIZE + IP_HDR_SIZE(ip));
u32 dst = ntohl(ip->daddr);
```

检测转发表，如果没有对应条目，则 ICMP 网络不可达：

```
rt_entry_t *entry = longest_prefix_match(ip_dst);
if (entry) {
  ip_send_packet(packet, len);
  return;
} else {
  icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_NET_UNREACH);
  return;
}
```

最长匹配过程比较简单，唯一要注意的是这里的 mask 是掩码本身，不是掩码位数：

```
rt_entry_t *longest_prefix_match(u32 dst)
{
  rt_entry_t *entry = NULL;
  u32 longest_mask = 0;
  rt_entry_t *longest_entry = NULL;
  list_for_each_entry(entry, &rtable, list) {
    if (((dst & entry->mask) == (entry->dest & entry->mask)) && entry->mask > longest_mask) {
      longest_mask = entry->mask;
      longest_entry = entry;
    }
  }
  return longest_entry;
}
```

实验结果及分析

路由程序执行效果

```
root@ubuntu:/mnt/hgfs/[illegible]/08-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=0.069 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.092 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.098 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3060ms
rtt min/avg/max/mdev = 0.069/0.082/0.098/0.015 ms
root@ubuntu:/mnt/hgfs/[illegible]/08-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=0.103 ms
64 bytes from 10.0.2.22: icmp_seq=2 ttl=63 time=0.248 ms
64 bytes from 10.0.2.22: icmp_seq=3 ttl=63 time=0.069 ms
64 bytes from 10.0.2.22: icmp_seq=4 ttl=63 time=0.124 ms

--- 10.0.2.22 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3057ms
rtt min/avg/max/mdev = 0.069/0.136/0.248/0.067 ms
root@ubuntu:/mnt/hgfs/[illegible]/08-router# ping 10.0.3.33 -c 4
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=0.208 ms
64 bytes from 10.0.3.33: icmp_seq=2 ttl=63 time=0.107 ms
64 bytes from 10.0.3.33: icmp_seq=3 ttl=63 time=0.076 ms
64 bytes from 10.0.3.33: icmp_seq=4 ttl=63 time=0.119 ms

--- 10.0.3.33 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3053ms
rtt min/avg/max/mdev = 0.076/0.127/0.208/0.050 ms
root@ubuntu:/mnt/hgfs/[illegible]/08-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 3064ms
pipe 4
root@ubuntu:/mnt/hgfs/[illegible]/08-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 3051ms
```

新的拓扑

定义新拓扑的脚本如下：

```python
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class RouterTopo(Topo):
    def build(self):
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        r1 = self.addHost('r1')
        r2 = self.addHost('r2')
        r3 = self.addHost('r3')

        self.addLink(h1, r1)
        self.addLink(r1, r2)
        self.addLink(r2, r3)
        self.addLink(r3, h2)

if __name__ == '__main__':
    topo = RouterTopo()
    net = Mininet(topo = topo, controller = None)

    h1, h2, r1, r2, r3 = net.get('h1', 'h2', 'r1', 'r2', 'r3')
    h1.cmd('ifconfig h1-eth0 10.0.1.11/24')
    h2.cmd('ifconfig h2-eth0 10.0.4.44/24')

    h1.cmd('route add default gw 10.0.1.1')
    h2.cmd('route add default gw 10.0.4.1')

    for h in (h1, h2):
        h.cmd('./scripts/disable_offloading.sh')
        h.cmd('./scripts/disable_ipv6.sh')

    r1.cmd('ifconfig r1-eth0 10.0.1.1/24')
    r1.cmd('ifconfig r1-eth1 10.0.2.1/24')
    r2.cmd('ifconfig r2-eth0 10.0.2.2/24')
    r2.cmd('ifconfig r2-eth1 10.0.3.1/24')
    r3.cmd('ifconfig r3-eth0 10.0.3.2/24')
    r3.cmd('ifconfig r3-eth1 10.0.4.1/24')

    r1.cmd('route add -net 10.0.3.0/24 gw 10.0.2.2')
    r1.cmd('route add -net 10.0.4.0/24 gw 10.0.2.2')
    r2.cmd('route add -net 10.0.1.0/24 gw 10.0.2.1')
    r2.cmd('route add -net 10.0.4.0/24 gw 10.0.3.2')
    r3.cmd('route add -net 10.0.1.0/24 gw 10.0.3.1')
    r3.cmd('route add -net 10.0.2.0/24 gw 10.0.3.1')

    for r in (r1, r2, r3):
```

```
        r.cmd('./scripts/disable_arp.sh')
        r.cmd('./scripts/disable_icmp.sh')
        r.cmd('./scripts/disable_ip_forward.sh')
        r.cmd('./router &')

    net.start()
    CLI(net)
    net.stop()
```

形象表示：

```
(10.0.1.11)h1--r1--r2--r3--h2(10.0.4.44)
```

连通性测试

```
root@ubuntu:/mnt/hgfs/[illegible]/08-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=0.162 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.107 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.169 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.129 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.107/0.141/0.169/0.029 ms
root@ubuntu:/mnt/hgfs/[illegible]/08-router# ping 10.0.1.11 -c 4
PING 10.0.1.11 (10.0.1.11) 56(84) bytes of data.
64 bytes from 10.0.1.11: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 10.0.1.11: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 10.0.1.11: icmp_seq=3 ttl=64 time=0.054 ms
64 bytes from 10.0.1.11: icmp_seq=4 ttl=64 time=0.054 ms

--- 10.0.1.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
rtt min/avg/max/mdev = 0.039/0.050/0.054/0.006 ms
root@ubuntu:/mnt/hgfs/[illegible]/08-router# ping 10.0.2.1 -c 4
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.116 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.112 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.113 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.125 ms

--- 10.0.2.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3055ms
rtt min/avg/max/mdev = 0.112/0.116/0.125/0.011 ms
root@ubuntu:/mnt/hgfs/[illegible]/08-router# ping 10.0.2.2 -c 4
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.100 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.104 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.117 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.128 ms

--- 10.0.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3052ms
rtt min/avg/max/mdev = 0.100/0.112/0.128/0.013 ms
root@ubuntu:/mnt/hgfs/[illegible]/08-router# ping 10.0.4.1 -c 4
```

```
PING 10.0.4.1 (10.0.4.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.113 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.140 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.136 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.120 ms

--- 10.0.4.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.113/0.127/0.140/0.013 ms
root@ubuntu:/mnt/hgfs/□□/□□□□□/□□8/08-router# ping 10.0.4.44 -c 4
PING 10.0.4.44 (10.0.4.44) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.357 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.109 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.117 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.101 ms

--- 10.0.4.44 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3065ms
rtt min/avg/max/mdev = 0.101/0.171/0.357/0.107 ms
root@ubuntu:/mnt/hgfs/□□/□□□□□/□□8/08-router#
```

路径测试

```
root@ubuntu:/mnt/hgfs/□□/□□□□□/□□8/08-router# traceroute 10.0.4.44
traceroute to 10.0.4.44 (10.0.4.44), 30 hops max, 60 byte packets
 1  10.0.2.2 (10.0.2.2)  0.693 ms  0.672 ms  0.668 ms
 2  10.0.3.2 (10.0.3.2)  0.667 ms  0.667 ms  0.664 ms
 3  10.0.4.44 (10.0.4.44)  0.655 ms  0.647 ms  0.637 ms
```