

实验 10 • 网络地址转换实验

吴嘉皓

2015K8009915007

一、实验内容

修改 nat.c 文件，实现以下功能：

1. NAT 映射表管理：
 - 1) 维护 NAT 连接映射表，支持映射的**添加、查找、更新和老化**操作；
2. 数据包的转换操作
 - 1) 对到达的合法数据包，进行 **IP 和 Port 转换**操作，**更新**头部字段，并**转发**数据包；
 - 2) 对于到达的非法数据包，回复 ICMP Destination Host Unreachable

二、实验流程

(一) 代码目录

```
2015K8009915007_吴嘉皓_10.tar.gz
├── Makefile
├── Makefile.bak
├── include
│   ├── arp.h
│   ├── arpcache.h
│   ├── base.h
│   ├── checksum.h
│   ├── ether.h
│   ├── hash.h
│   ├── icmp.h
│   ├── ip.h
│   ├── list.h
│   ├── log.h
│   ├── nat.h
│   ├── packet.h
│   ├── rtable.h
│   ├── tcp.h
│   └── types.h
├── libipstack.a
├── main.c
└── nat.c
```



```
├── nat_topo.py
├── scripts
│   ├── disable_arp.sh
│   ├── disable_icmp.sh
│   ├── disable_ip_forward.sh
│   ├── disable_offloading.sh
│   └── disable_tcp_rst.sh
├── test.c
└── 实验 10-网络地址转换-实验报告.pdf
```

(二) 实验流程

1. 在 10-nat 目录下输入如下命令:

```
1 make
2 sudo python nat_topo.py
3 mininet> xterm n1 h1 h2
4 n1> ./nat
5 h2> python -m SimpleHTTPServer
6 h1> wget http://159.226.39.123:8000
```

三、实验结果

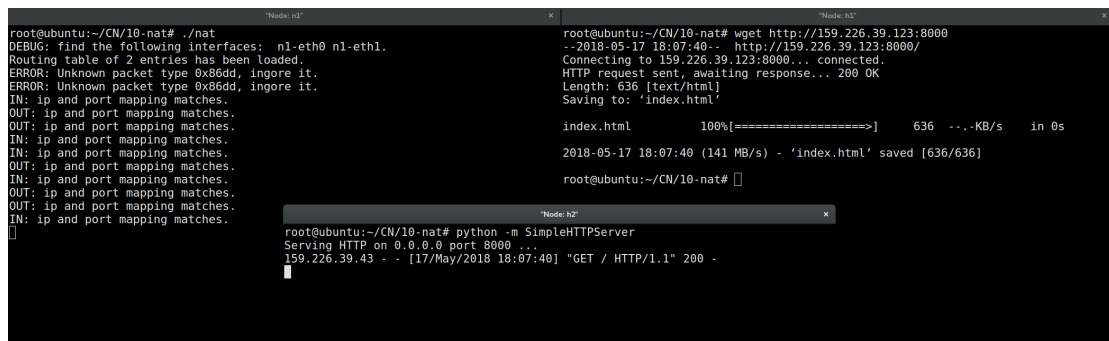


图 1 获取网页结果图

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::8010:dff:fe41:26f5	ff02::2	ICMPv6	70	Router Solicitation from 82:10:0d:41:26:f5
2	0.237415163	10.21.0.1	159.226.39...	TCP	74	48880 → 8000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1869338022 TSecr=0 WS=512
3	0.237412214	82:10:0d:41:26:f5	Broadcast	ARP	42	Who has 10.21.0.1? Tell 10.21.0.254
4	0.287416059	a6:0f:e7:fa:cb:53	82:10:0d:41...	ARP	42	10.21.0.1 is at a6:0f:e7:fa:cb:53
5	0.287423286	159.226.39.123	10.21.0.1	TCP	74	8000 → 48880 [SYN, ACK] Seq=0 Ack=1 Win=29696 Len=0 MSS=1460 SACK_PERM=1 TSval=3910470035 TSecr=1869338022
6	0.287435166	10.21.0.1	159.226.39...	TCP	66	48880 → 8000 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=1869338022 TSecr=3910470035
7	0.287599014	10.21.0.1	159.226.39...	HTTP	212	GET / HTTP/1.1
8	0.288194985	159.226.39.123	10.21.0.1	TCP	66	8000 → 48880 [ACK] Seq=1 Ack=147 Win=30208 Len=0 TSval=3910470035 TSecr=1869338022
9	0.288220268	159.226.39.123	10.21.0.1	TCP	83	[TCP segment of a reassembled PDU]
10	0.288228364	10.21.0.1	159.226.39...	TCP	66	48880 → 8000 [ACK] Seq=147 Ack=18 Win=29696 Len=0 TSval=1869338023 TSecr=3910470035
11	0.288265561	159.226.39.123	10.21.0.1	TCP	235	[TCP segment of a reassembled PDU]
12	0.288269982	10.21.0.1	159.226.39...	TCP	66	48880 → 8000 [ACK] Seq=147 Ack=187 Win=30720 Len=0 TSval=1869338023 TSecr=3910470036
13	0.299740048	159.226.39.123	10.21.0.1	HTTP	702	HTTP/1.0 200 OK (text/html)
14	0.299767675	10.21.0.1	159.226.39...	TCP	66	48880 → 8000 [ACK] Seq=147 Ack=823 Win=31744 Len=0 TSval=1869338034 TSecr=3910470047
15	0.299827138	159.226.39.123	10.21.0.1	TCP	66	8000 → 48880 [FIN, ACK] Seq=823 Ack=147 Win=30208 Len=0 TSval=3910470047 TSecr=1869338023
16	0.300545462	10.21.0.1	159.226.39...	TCP	66	48880 → 8000 [FIN, ACK] Seq=824 Ack=824 Win=31744 Len=0 TSval=1869338035 TSecr=3910470047
17	0.300945876	159.226.39.123	10.21.0.1	TCP	66	8000 → 48880 [ACK] Seq=824 Ack=148 Win=30208 Len=0 TSval=3910470048 TSecr=1869338035
18	5.376432729	a6:0f:e7:fa:cb:53	82:10:0d:41...	ARP	42	Who has 10.21.0.254? Tell 10.21.0.1
19	5.37651092	82:10:0d:41:26:f5	a6:0f:e7:fa...	ARP	42	10.21.0.254 is at 82:10:0d:41:26:f5

图 2 h1 节点的抓包结果

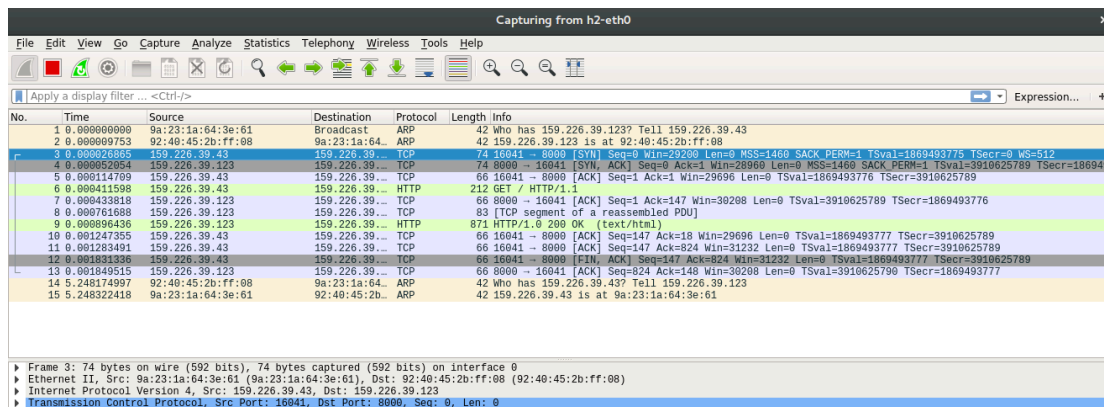


图 3 h2 节点的抓包结果

四、结果分析

(一) 结果分析

由图 1 可知, h1 成功将 h2 上生成的 html 网页获取到了本机 h1 上。

通过 wireshark 抓包, 我们得到了图 2 和图 3 的结果, 并且我们能够从这一次抓包中看出 h1 获取 h2 上网页的整个过程。

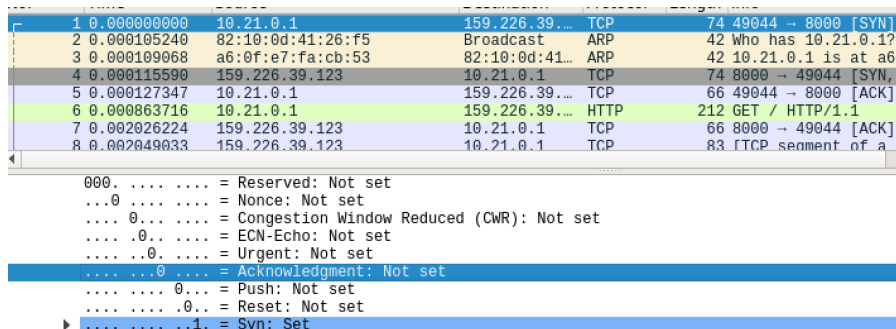


图 5 h2 向 h1 发送建立连接信号

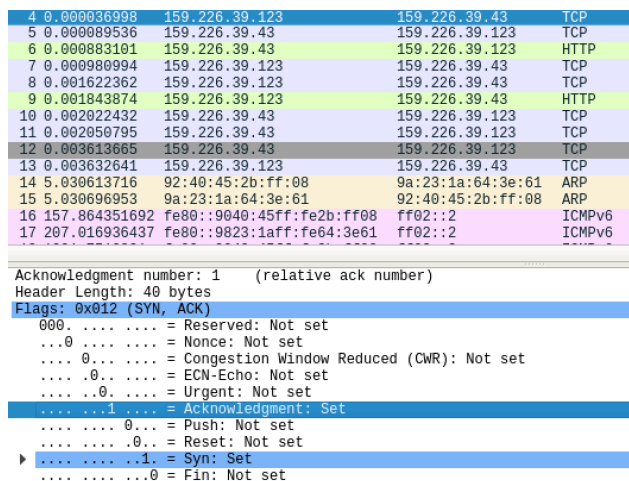


图 4 h2 收到 h1 的连接接收信号

图 2 中, h2 首先广播 ARP 请求, 获取到了 h1 的地址; 然后 (图 5) 发送 TCP

请求 (SYN=1, ACK=0) 请求与 h1 建立连接; 在收到 h1 的 ACK 信号 (图 4) 后, 向 h1 发送 ACK 确认, 之后通过 http 协议来获取 html 的信息; 接着通过 TCP 协议进行数据的传输。

图 3 中, h1 收到来自 h2 的 ARP 请求后, 向 h2 单播了自己的地址; 来自 h2 的 tcp 请求经过 NAT 转换后, 被 h1 接收; h1 向 h2 发送 ACK 消息, 经 NAT 转换后, 到达 h2; 之后, h1 再次接收到了 h2 经 NAT 转换后的 ACK 确认, 这时 tcp 连接成功建立。然后开始传输数据。

(二) 代码实现分析

⇒ `get_packet_direction(char *packet)`

该函数通过数据包中的**目的地址**和**源地址**来判断包的发送方向。

```
1 static int get_packet_direction(char *packet)
2 {
3     struct iphdr *ip = packet_to_ip_hdr(packet);
4     u32 saddr = ntohl(ip->saddr);
5     u32 daddr = ntohl(ip->daddr);
6
7     if (IS_DIR_OUT(saddr, daddr))
8         return DIR_OUT;
9     else if (IS_DIR_IN(saddr, daddr))
10        return DIR_IN;
11    else
12        return DIR_INVALID;
13 }
```

判断逻辑如下:

```
1 #define IS_INTERNAL_IP(addr) \
2     (longest_prefix_match((addr))->iface->ip == nat.internal_iface->ip)
3 #define IS_EXTERNAL_IP(addr) \
4     (longest_prefix_match((addr))->iface->ip == nat.external_iface->ip)
5
6 #define IS_DIR_IN(saddr, daddr) \
7     (daddr == nat.external_iface->ip) && IS_EXTERNAL_IP(saddr)
8 #define IS_DIR_OUT(saddr, daddr) \
9     IS_INTERNAL_IP(saddr) && IS_EXTERNAL_IP(daddr)
```

如果路由表中地址和 NAT 的内网端口地址一致, 则判定为内部 IP, 反之为外部 IP; 如果目的地址是外部 IP, 源地址是内部 IP, 则判定方向为 IN; 如果目的地址为 NAT 的外网端口地址, 源地址是外部 IP, 则判定方向为 OUT。



⇒ do_translation(iface_info_t *iface, char *packet, int len, int dir)

```
1 void do_translation(iface_info_t *iface, char *packet, int len, int dir){
2     struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
3     struct tcphdr *tcp_hdr = packet_to_tcp_hdr(packet);
4     u16 sport = ntohs(tcp_hdr->sport);
5     u32 saddr = ntohl(ip_hdr->saddr);
6     u16 dport = ntohs(tcp_hdr->dport);
7     u32 daddr = ntohl(ip_hdr->daddr);
8     // find out whether there exists nat mapping
9     u32 serv_ip = (dir == DIR_IN) ? saddr : daddr;
10    u16 serv_port = (dir == DIR_IN) ? sport : dport;
11    u8 hash_value = caculate_hash8(serv_port, serv_ip);
12    struct list_head * mapping_entry = &(nat.nat_mapping_list[hash_value]);
13    pthread_mutex_lock(&nat.lock);
14    struct nat_mapping * pos = NULL, *q = NULL;
15    if(!list_empty(mapping_entry)){
16        list_for_each_entry_safe(pos, q, mapping_entry, list){
17            if(dir == DIR_OUT && NAT_MAPPING_MATCH_IN(pos,saddr,sport)){
18                printf("OUT: ip and port mapping matches.\n");
19                update_ip_and_tcp_header(packet,
20                                        nat.external_iface->ip,
21                                        pos->external_port,
22                                        DIR_OUT);
23                pos->update_time = time(NULL);
24                ip_send_packet(packet, len);
25                recover_unused_conn(pos, tcp_hdr);
26                pthread_mutex_unlock(&nat.lock);
27                return ;
28            }
29            else if(dir == DIR_IN && NAT_MAPPING_MATCH_EX(pos,daddr,dport)){
30                printf("IN: ip and port mapping matches.\n");
31                update_ip_and_tcp_header(packet,
32                                        pos->internal_ip,
33                                        pos->internal_port,
34                                        DIR_IN);
35                pos->update_time = time(NULL);
36                ip_send_packet(packet, len);
37                recover_unused_conn(pos, tcp_hdr);
38                pthread_mutex_unlock(&nat.lock);
39                return ;
40            }
41        }
42    }
43    // nat mapping does not find (OUT)
44    // assign a new port and build a new connection
45    u16 new_port = assign_external_port();
46    struct nat_mapping *new_mapping = init_new_mapping(saddr,sport,new_port);
47    list_add_tail(&(new_mapping->list), mapping_entry);
48    // update saddr, sport and checksum of tcp header and ip header
49    update_ip_and_tcp_header(packet,
50                            (nat.external_iface->ip,
51                            new_port,
52                            DIR_OUT);
53    pthread_mutex_unlock(&nat.lock);
54    ip_send_packet(packet, len);
55    return ;
56 }
```

这是本次实验的核心代码,用于 NAT 的网络地址转换,包含了 NAT 表的添加、查找、更新操作。



先根据 (serv_ip, serv_port) 二元组的值计算查找哈希表的哈希值;

```
1 u8 caculate_hash8(u16 serv_port, u32 serv_ip){
2     char buf[6];
3     memcpy(buf, &serv_ip, 4);
4     memcpy(buf+4, &serv_port, 2);
5     return hash8(buf, 6);
6 }
```

然后得到对应 nat mapping 的链表;

如果链表为空, 显然需要插入新的 nat mapping;

```
1     u16 new_port = assign_external_port();
2     struct nat_mapping *new_mapping = init_new_mapping(saddr, sport, new_port);
3     list_add_tail(&(new_mapping->list), mapping_entry);
4     // update saddr, sport and checksum of tcp header and ip header
5     update_ip_and_tcp_header(packet,
6                               (nat.external_iface->ip,
7                               new_port,
8                               DIR_OUT);
```

否则, 根据包的方向和 (ip, port) 二元组来查找对应的 nat mapping。

查找到对应的 nat mapping 后, 更新 tcp 和 ip 头部, 并刷新该 mapping 的更新时间, 然后将包通过 ip_send_packet() 发送出去。

```
1 void update_ip_and_tcp_header(char *packet, u32 addr, u16 port, int dir){
2     struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
3     struct tcphdr *tcp_hdr = packet_to_tcp_hdr(packet);
4     if(dir == DIR_IN){
5         ip_hdr->daddr = htonl(addr);
6         tcp_hdr->dport = htons(port);
7     }
8     else{
9         ip_hdr->saddr = htonl(addr);
10        tcp_hdr->sport = htons(port);
11    }
12    tcp_hdr->checksum = tcp_checksum(ip_hdr, tcp_hdr);
13    ip_hdr->checksum = ip_checksum(ip_hdr);
14 }
```

更新头部, 即根据方向修改相应的部分:

1. 方向为 IN, 更新 ip->daddr 和 tcp->dport;
2. 方向为 OUT, 更新 ip->saddr 和 tcp->sport;