

计算机网络研讨课实验报告

冯吕 2015K8009929049

2018 年 5 月 18 日

实验题目

网络地址转换实验

实验内容

本次实验需要实现 NAT 网络地址转换：

- NAT 映射表维护：维护 NAT 连接映射表，支持映射的添加、查找、更新和老化操作；
- 数据包的转换操作：
 - 对于到达的合法数据包，进行 IP 和 Port 转换操作，更新头部字段，并转发数据包
 - 对于到达的非法数据包，回复 ICMP Destination Host Unreachable

之后，根据给定网络拓扑，在 *n1* 上执行 *nat* 程序，在 *h2* 上运行 HTTP 服务，在 *h1* 上访问 *h2* 的 HTTP 服务。

实验流程

本次实验中，首先要实现 NAT 地址转换，主要需要实现三个函数：

*get_packet_direction(char *packet)*：该函数用于判断数据包的方向：进或者出去。*nat* 结构中存储有外部端口和内部端口的信息，因此，可根据数据包的源端口和目的端口来判断数据包的方向，如果数据包的源端口等于 *nat* 的内部端口并且目的端口等于 *nat* 的外部端口，那么则说明数据包是出去的；如果源端口等于 *nat* 的外部端口并且目的 *ip* 等于外部端口的 *ip*，则说明是进来；否则为无效。

```
1 static int get_packet_direction(char *packet)
2 {
3     struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
4     rt_entry_t *src_entry = longest_prefix_match(ntohl(ip_hdr->saddr));
5     rt_entry_t *dest_entry = longest_prefix_match(ntohl(ip_hdr->daddr));
6     if(src_entry->iface == nat.internal_iface
7        && dest_entry->iface == nat.external_iface){
8         return DIR_OUT;
9     }
10    else if(src_entry->iface == nat.external_iface
11            && ntohl(ip_hdr->daddr) == nat.external_iface->ip){
12        return DIR_IN;
```

```

13     }
14     return DIR_INVALID;
15 }

```

do_translation: 该函数完成数据包的转换。首先判断数据包的方向，如果是出去，那么查找是否已经存在映射，如果不存在，则建立映射，然后更新头部字段，将包转发出去；如果包是进来，那么直接查找映射，然后更新头部信息，转发数据包，如果查找失败，则回复 *ICMP* 目的主机不可达消息。

```

1 void do_translation(iface_info_t *iface, char *packet, int len, int dir)
2 {
3     pthread_mutex_lock(&nat.lock);
4     struct iphdr *ip = packet_to_ip_hdr(packet);
5     struct tcphdr *tcp = packet_to_tcp_hdr(packet);
6     int is_find = 0;
7     struct nat_mapping *nat_entry = NULL;
8     u32 dest = ntohl(ip->daddr);
9     u32 src = ntohl(ip->saddr);
10    u16 sport = ntohs(tcp->sport);
11    u16 dport = ntohs(tcp->dport);
12    if(dir == DIR_IN){
13        struct list_head *nat_entry_list =
14            &nat.nat_mapping_list[hash8((char*)&src,4)];
15        if(!list_empty(nat_entry_list)){
16            list_for_each_entry(nat_entry,
17                                nat_entry_list, list){
18                if(nat_entry->external_ip == dest
19                   && nat_entry->external_port == dport){
20                    is_find = 1;
21                    break;
22                }
23            }
24        }
25        if(!is_find){
26            icmp_send_packet(packet, len, ICMP_DEST_UNREACH,
27                              ICMP_HOST_UNREACH);
28            free(packet);
29        }
30        else{
31            ip->daddr = htonl(nat_entry->internal_ip);
32            ip->checksum = ip_checksum(ip);
33            tcp->dport = htons(nat_entry->internal_port);
34            tcp->checksum = tcp_checksum(ip, tcp);
35            nat_entry->update_time = time(NULL);
36        }
37    }
38    else{

```

```

39     struct list_head *nat_entry_list =
40     &nat.nat_mapping_list[hash8((char*)&dest,4)];
41     if(!list_empty(nat_entry_list)){
42         list_for_each_entry(nat_entry, nat_entry_list, list){
43             if(nat_entry->internal_ip == src
44                 && nat_entry->internal_port == sport){
45                 is_find = 1;
46                 break;
47             }
48         }
49     }
50     if(!is_find){
51         nat_entry = insert_nat_entry(src,
52             sport,dest,nat_entry_list);
53     }
54     ip->saddr = htonl(nat_entry->external_ip);
55     ip->checksum = ip_checksum(ip);
56     tcp->sport = htons(nat_entry->external_port);
57     tcp->checksum = tcp_checksum(ip,tcp);
58     nat_entry->update_time = time(NULL);
59 }
60 ip_send_packet(packet,len);
61 pthread_mutex_unlock(&nat.lock);
62 }

```

同时, *nat* 还要不断进行老化操作: *nat_timeout*。如果双方都已经发送 *FIN* 并回收 *ACK* 信息, 则进行回收; 如果已经超过 60s 没有发送数据, 那么也认为传输结束, 进行回收。

```

1 void *nat_timeout()
2 {
3     while (1) {
4         sleep(1);
5         struct nat_mapping *nat_entry = NULL;
6         struct nat_mapping *nat_entry_next = NULL;
7         pthread_mutex_lock(&nat.lock);
8         for(int i =0; i < HASH_8BITS; i++){
9             struct list_head *nat_entry_list =
10             &nat.nat_mapping_list[i];
11             if(list_empty(nat_entry_list))
12                 continue;
13             list_for_each_entry_safe(nat_entry, nat_entry_next,
14                 nat_entry_list, list){
15                 if(nat_entry->conn.external_fin == 1 &&
16                     nat_entry->conn.internal_fin == 1 &&
17                     nat_entry->conn.external_seq_end ==
18                     nat_entry->conn.internal_ack &&

```

```

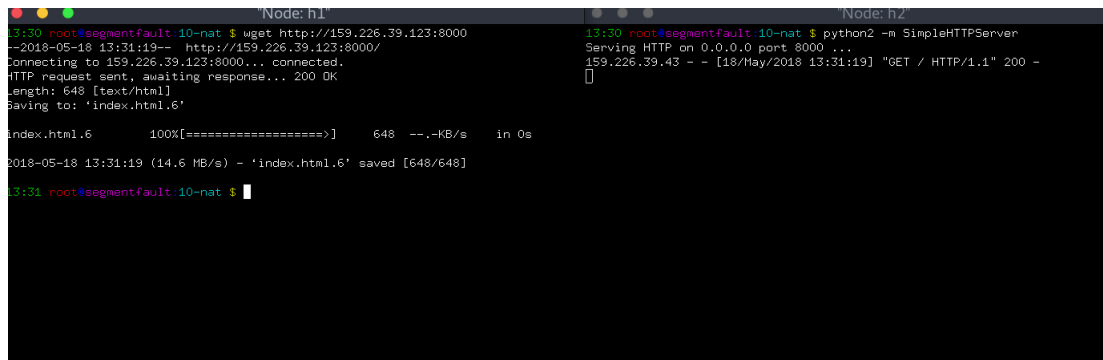
19         nat_entry->conn.internal_seq_end ==
20         nat_entry->conn.external_ack){
21             list_delete_entry(&nat_entry->list);
22             free(nat_entry);
23         }
24         else if(time(NULL) - nat_entry->update_time
25 > TCP_ESTABLISHED_TIMEOUT){
26             list_delete_entry(&nat_entry->list);
27             free(nat_entry);
28         }
29     }
30 }
31 pthread_mutex_unlock(&nat.lock);
32 }
33
34 return NULL;
35 }

```

实现 NAT 转换之后，根据给定拓扑进行网络服务实验。

实验结果

通过 h1 能够成功访问 h2 上的 HTTP 服务：



```

Node: h1
13:30 root@segmentfault:10-nat $ wget http://159.226.39.123:8000
--2018-05-18 13:31:19-- http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 648 [text/html]
Saving to: 'index.html.6'

index.html.6      100%[=====] 648  --.-KB/s  in 0s

2018-05-18 13:31:19 (14.6 MB/s) - 'index.html.6' saved [648/648]

13:31 root@segmentfault:10-nat $

Node: h2
13:30 root@segmentfault:10-nat $ python2 -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
159.226.39.43 - - [18/May/2018 13:31:19] "GET / HTTP/1.1" 200 -

```

结果分析

实验结果正确，NAT 能够正确实现映射表的管理和数据包的转换。