实验报告

姓名：王苑铮 学号：2015K8009922002

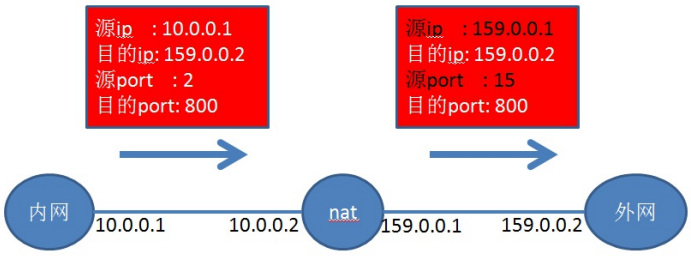
1.实验题目： nat地址转换

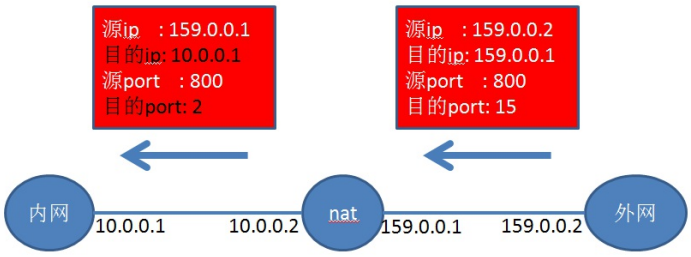
2.实验内容：

- 转换表的老化
- 地址转换
- 判断packet的发送方向

3.实验过程

nat转换过程示例：





packet方向的判断方法：

收到packet。查路由表项中packet的源ip、目的ip对应的iface

- 源iface == internal_iface && 目的iface == external_iface：内网向外网发

- 源iface == external_iface && 目的iface == external_iface：外网向内网发

代码：

```
static int get_packet_direction(char *packet)
{
        fprintf(stdout, "TODO: determine the direction of this packet.\n");
        struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
        u32 saddr = ntohl(ip_hdr->saddr),
                daddr = ntohl(ip_hdr->daddr);

        rt_entry_t *sentry = longest_prefix_match(saddr),
                           *dentry = longest_prefix_match(daddr);

        if(sentry && dentry){
                if( strcmp(sentry->iface->name,nat.internal_iface->name)==0 &&\
                        strcmp(dentry->iface->name,nat.external_iface->name)==0 )
                        return DIR_OUT;
                if( strcmp(sentry->iface->name,nat.external_iface->name)==0 &&\
                        strcmp(dentry->iface->name,nat.external_iface->name)==0 )
                        return DIR_IN;
        }
        return DIR_INVALID;
}
```

```
u16 assign_external_port(){
        static u16 port=1;
        int find=0;
        int i=0;
        for(i=0 ; i<65536 ; ++i){
                if((port+i)%65536 == 0)
                        continue;
                if( nat.assigned_ports[(port+i)%65536]==0 ){
                        find = 1;
                        break;
                }
        }
        if(find){
                port = (port+i)%65536;
                nat.assigned_ports[port] = 1;
                return port;
        }
        else
                return -1;
}

void do_translation(iface_info_t *iface, char *packet, int len, int dir)
{
        pthread_mutex_lock(&nat.lock);
        fprintf(stdout, "TODO: do translation for this packet.\n");
        struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
        struct tcphdr *tcp_hdr = packet_to_tcp_hdr(packet);
```

```
        int direction = get_packet_direction(packet);
        iface_info_t *iface_send=NULL;
        switch(direction){
                case DIR_OUT:{
                        //hash对应的链表
                        u32 daddr = ntohl(ip_hdr->daddr);
                        u8 key = hash8((char*)&daddr,sizeof(daddr));
                        struct list_head *head = &nat.nat_mapping_list[key];
                        struct nat_mapping *mapping_entry, *q;
                        int find=0;
                        //查找是否有对应的映射
                        list_for_each_entry_safe(mapping_entry, q, head, list) {
                                if(mapping_entry->external_ip == ntohl(ip_hdr->daddr))
                                        find = 1;
                                        break;
                                }
                        }
                        //如果没有对应映射，要新建映射表项
                        if(!find){
                                struct nat_mapping *new_mapping = (struct nat_mapping *
                                new_mapping->internal_ip = ntohl(ip_hdr->saddr);
                                new_mapping->external_ip = ntohl(ip_hdr->daddr);
                                new_mapping->internal_port = ntohs(tcp_hdr->sport);
                                new_mapping->external_port = assign_external_port();
                                new_mapping->update_time = 0;
                                memset(&new_mapping->conn,0,sizeof(struct nat_connectio
                                list_add_tail(&new_mapping->list, &mapping_entry->list)
                                mapping_entry = new_mapping;
                        }
                        //更新映射
                        mapping_entry->update_time = 0;
                        int fin = tcp_hdr->flags & 0x01; //TCP_FIN
                        int ack = tcp_hdr->flags & 0x10; //TCP_ACK
                        int rst = tcp_hdr->flags & 0x04; //TCP_RST
                        if(fin)
                                mapping_entry->conn.internal_fin = 1;
                        if(ack)
                                mapping_entry->conn.internal_ack = 1;
                        if(rst){
                                mapping_entry->conn.internal_fin = 1;
                                mapping_entry->conn.internal_ack = 1;
                                mapping_entry->conn.external_fin = 1;
                                mapping_entry->conn.external_ack = 1;
                        }
                        //修改packet
                        ip_hdr->saddr = htonl(nat.external_iface->ip);
                        tcp_hdr->sport = htons(mapping_entry->external_port);
                        tcp_hdr->checksum = tcp_checksum(ip_hdr,tcp_hdr);
                        ip_hdr->checksum = ip_checksum(ip_hdr);
                        ip_send_packet(packet,len);
                }
                break;

                case DIR_IN:{
                //hash对应的链表
```

```
                        u32 saddr = ntohl(ip_hdr->saddr);
                        u8 key = hash8((char*)&saddr,sizeof(saddr));
                        struct list_head *head = &nat.nat_mapping_list[key];
                        struct nat_mapping *mapping_entry, *q;
                        int find=0;
                        //查找是否有对应的映射
                        list_for_each_entry_safe(mapping_entry, q, head, list) {
                                if(mapping_entry->external_ip == ntohl(ip_hdr->saddr)){
                                        find = 1;
                                        break;
                                }
                        }

                        if(!find){
                                break;
                        }
                        //更新映射
                        mapping_entry->update_time = 0;
                        int fin = tcp_hdr->flags & 0x01; //TCP_FIN
                        int ack = tcp_hdr->flags & 0x10; //TCP_ACK
                        int rst = tcp_hdr->flags & 0x04; //TCP_RST
                        if(fin)
                                mapping_entry->conn.external_fin = 1;
                        if(ack)
                                mapping_entry->conn.external_ack = 1;
                        if(rst){
                                mapping_entry->conn.external_fin = 1;
                                mapping_entry->conn.external_ack = 1;
                                mapping_entry->conn.internal_fin = 1;
                                mapping_entry->conn.internal_ack = 1;
                        }
                        //修改packet
                        ip_hdr->daddr = htonl(mapping_entry->internal_ip);
                        tcp_hdr->dport = htons(mapping_entry->internal_port);
                        tcp_hdr->checksum = tcp_checksum(ip_hdr,tcp_hdr);
                        ip_hdr->checksum = ip_checksum(ip_hdr);
                        ip_send_packet(packet,len);
                }
                break;

                default:
                        break;
        }
        pthread_mutex_unlock(&nat.lock);
}
```

```
void *nat_timeout()
{
        while (1) {
                pthread_mutex_lock(&nat.lock);
                fprintf(stdout, "TODO: sweep finished flows periodically.\n");

                for (int i = 0; i < HASH_8BITS; i++) {
                        struct list_head *head = &nat.nat_mapping_list[i];
                        struct nat_mapping *mapping_entry, *q;
                        list_for_each_entry_safe(mapping_entry, q, head, list) {
                                mapping_entry->update_time += 1;
                                int conn_end =  mapping_entry->conn.external_fin &&\
                                                mapping_entry->conn.internal_fin &&\
                                                mapping_entry->conn.external_ack &&\
                                                mapping_entry->conn.internal_ack ;
                                if(mapping_entry->update_time >= 60 || conn_end){
                                        list_delete_entry(&mapping_entry->list);
                                }
                        }
                }

                pthread_mutex_unlock(&nat.lock);
                sleep(1);
        }

        return NULL;
}
```

4.实验结果

```
root@ubuntu:/home/wang/networking/exp-8_nat# python nat_topo.py
mininet> n1 ./nat &
mininet> h2 python -m SimpleHTTPServer &
mininet> h1 wget 159.226.39.123:8000
--2018-05-18 10:16:54--  http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 904 [text/html]
Saving to: ‘index.html.21’

index.html.21       100%[===================>]     904  --.-KB/s    in 0s

2018-05-18 10:16:54 (91.8 MB/s) - ‘index.html.21’ saved [904/904]

mininet>
```

5.结果分析

内网的h1经过n1的nat转换，get到了外网h2的http数据。
nat成功实现了在不让外部网络访问内部网络的情况下，让内部网络访问外部网络