03-nat

My name: 蒋鹏宇

My Student ID: 211502021

This lab took me about 6 hours to do.

Implementation Explanation:

处理配置信息

`int parse_config(const char *filename)` 负责根据config文件中读取的字符串，配置external-iface,internal-iface和DNAT Rules

实现代码过长，在此不表

处理NAT地址转换

`static int get_packet_direction(char *packet)` 函数，负责判断分组方向，当源地址为内部地址，且目的地址为外部地址时，方向为DIR_OUT，当源地址为外部地址，且目的地址为external_iface地址时，方向为DIR_IN，实现如下：

```
static int get_packet_direction(char *packet)
{
        struct iphdr *ih=packet_to_ip_hdr(packet);
        rt_entry_t *match=longest_prefix_match(ntohl(ih->saddr));

        if(match->iface->index==nat.internal_iface->index)
                return DIR_OUT;
        else if(match->iface->index==nat.external_iface->index)
                return DIR_IN;

        return DIR_INVALID;
}
```

`void do_translation(iface_info_t *iface, char *packet, int len, int dir)` 函数负责实际处理地址转换，首先查询映射关系表，看能否已经建立连接，如果没有建立连接，根据分组方向看能否新建连接，如果无法新建连接则丢弃该分组；如果能新建的话，根据分组方向进行地址转换，IN方向需要将目的IP地址、目的端口更新为内网对应主机的IP地址、目的端口，重新计算检验和，将分组发送到对

应主机；OUT方向时需要NAT分配一个新的端口，建立映射表项，将分组的源IP地址、端口更新为NAT WAN端口的IP地址与分配的端口号，重新计算检验和，并将分组发送出去

以IN方向为例，实现如下：

```
if(dir==DIR_IN)
        {
                int found=0;
                struct list_head *head=&(nat.nat_mapping_list[hash]);
                struct nat_mapping *map;
                struct nat_mapping *new_mapping=(struct nat_mapping*)malloc(sizeof(struc

                list_for_each_entry(map,head,list)
                {
                        if(map->external_ip==ntohl(ih->daddr)&&map->external_port==ntohs
                        {
                                found=1;
                                break;
                        }
                }

                if(!found)
                {
                        struct dnat_rule *rule;
                        list_for_each_entry(rule,&nat.rules,list)
                        {
                                if(nat.assigned_ports[rule->external_port]==0&&rule->ext
                                        &&rule->external_port==ntohs(th->dport))
                                        {
                                                nat.assigned_ports[rule->external_port]=
                                                new_mapping->external_ip=rule->external_
                                                new_mapping->external_port=rule->externa
                                                new_mapping->internal_ip=rule->internal_
                                                new_mapping->internal_port=rule->interna
                                                list_add_tail(&(new_mapping->list),head)
                                                map=new_mapping;
                                                break;
                                        }
                        }
                }

                th->dport=htons(map->internal_port);
                ih->daddr=htonl(map->internal_ip);

                map->conn.external_seq_end= th->seq;
                if (th->flags==TCP_ACK)
                        map->conn.external_ack=th->ack;
                map->conn.external_fin=(th->flags==TCP_FIN)?TCP_FIN:0;
```

```
            map->update_time=time(NULL);
        }
```

NAT老化操作

对认为已经结束的连接进行老化操作，思路如下：

(1)双方都已发送FIN且回复相应ACK的连接，一方发送RST包的连接，可以直接回收端口号以及相关内存空间。

(2)双方已经超过60秒未传输数据的连接，认为其已经传输结束，可以回收端口号以及相关内存空间。

实现如下：

```
void *nat_timeout()
{
        while(1)
        {
                // fprintf(stdout, "TODO: sweep finished flows periodically.\n");
                pthread_mutex_lock(&nat.lock);
                time_t now=time(NULL);
                for (int i=0;i<HASH_8BITS;i++)
                {
                        struct list_head *head=&(nat.nat_mapping_list[i]);
                        if (!list_empty(head))
                        {
                                struct nat_mapping *map=NULL,*q;
                                list_for_each_entry_safe(map,q,head,list)
                                {
                                        if(now-map->update_time>TCP_ESTABLISHED_TIMEOUT
                                        {
                                                nat.assigned_ports[map->external_port]=0
                                                list_delete_entry(&(map->list));
                                                free(map);
                                        }
                                }
                        }
                }
                pthread_mutex_unlock(&nat.lock);
                sleep(1);
        }

        return NULL;
}
```

Screenshots:

SNAT

h1访问h3：

```
mininet> h1 wget http://159.226.39.123:8000
--2023-12-21 15:31:20--  http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 212 [text/html]
Saving to: 'index.html'

index.html          100%[===================>]     212  --.-KB/s    in 0s

2023-12-21 15:31:20 (53.3 MB/s) - 'index.html' saved [212/212]
```

生成的html文件：

index.html

```
<!doctype html>
<html>
    <head> <meta charset="utf-8">
        <title>Network IP Address</title>
    </head>
    <body>
            My IP is: 159.226.39.123
            Remote IP is: 159.226.39.43
        </body>
</html>
```

h2访问h3:

```
mininet> h2 wget http://159.226.39.123:8000
--2023-12-21 15:32:25--  http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 212 [text/html]
Saving to: 'index.html.1'

index.html.1        100%[===================>]     212  --.-KB/s    in 0s

2023-12-21 15:32:25 (56.0 MB/s) - 'index.html.1' saved [212/212]
```

生成的html文件：

index.html.1

```
<!doctype html>
<html>
    <head> <meta charset="utf-8">
        <title>Network IP Address</title>
    </head>
    <body>
            My IP is: 159.226.39.123
            Remote IP is: 159.226.39.43
        </body>
</html>
```

DNAT

h3访问h1:

```
mininet> h3 wget http://159.226.39.43:8000
--2023-12-21 15:40:25--  http://159.226.39.43:8000/
Connecting to 159.226.39.43:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 208 [text/html]
Saving to: 'index.html.2'

index.html.2        100%[===================>]     208  --.-KB/s    in 0s

2023-12-21 15:40:25 (51.3 MB/s) - 'index.html.2' saved [208/208]
```

生成的html文件:

index.html.2

```
<!doctype html>
<html>
    <head> <meta charset="utf-8">
        <title>Network IP Address</title>
    </head>
    <body>
            My IP is: 10.21.0.1
            Remote IP is: 159.226.39.123
        </body>
</html>
```

h3访问h2:

```
mininet> h3 wget 159.226.39.43:8001
--2023-12-21 15:41:10--  http://159.226.39.43:8001/
Connecting to 159.226.39.43:8001... connected.
HTTP request sent, awaiting response... 200 OK
Length: 208 [text/html]
Saving to: 'index.html.3'

index.html.3        100%[===================>]     208  --.-KB/s    in 0s

2023-12-21 15:41:10 (42.2 MB/s) - 'index.html.3' saved [208/208]
```

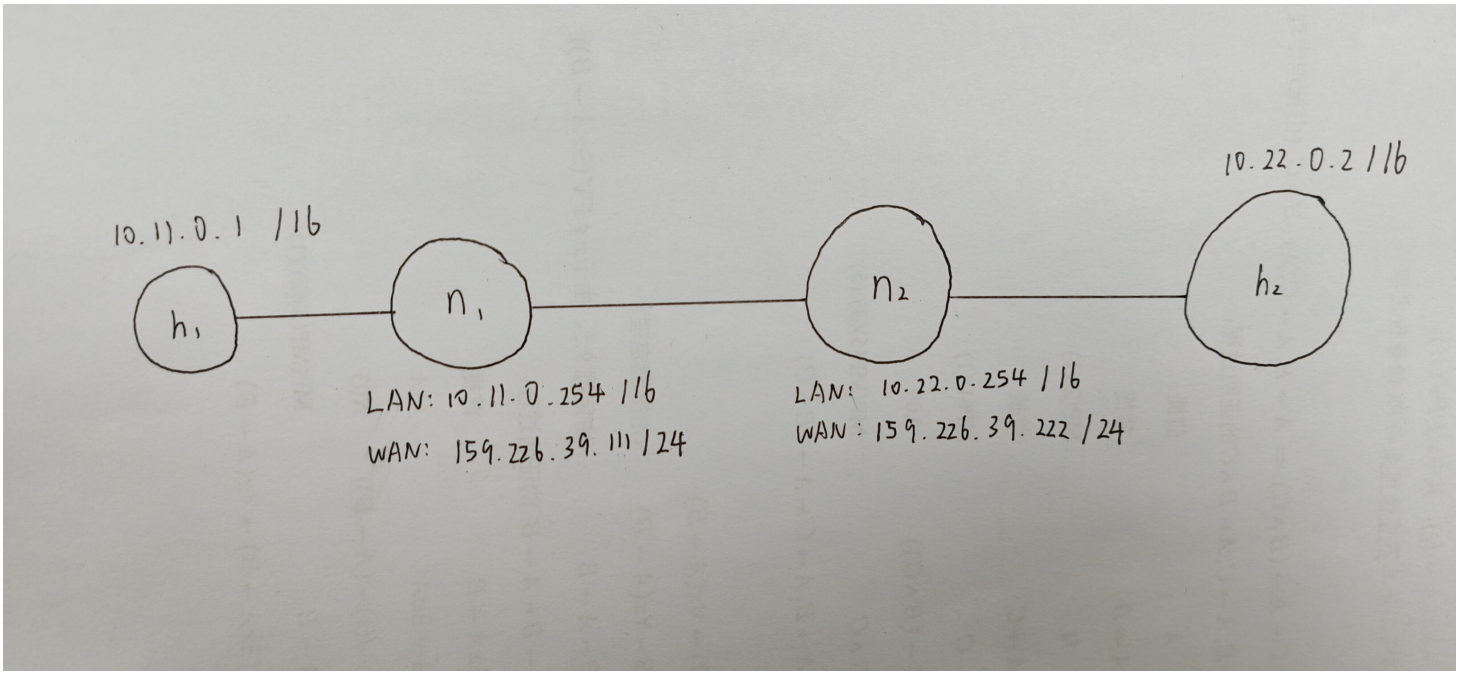
生成的html文件:

index.html.3

```
<!doctype html>
<html>
    <head> <meta charset="utf-8">
        <title>Network IP Address</title>
    </head>
    <body>
            My IP is: 10.21.0.2
            Remote IP is: 159.226.39.123
        </body>
</html>
```

多NAT实验

自建网络拓扑结构如下:



其中n1作为SNAT,n2作为DNAT,主机h2执行服务器程序，主机h1向h2请求页面

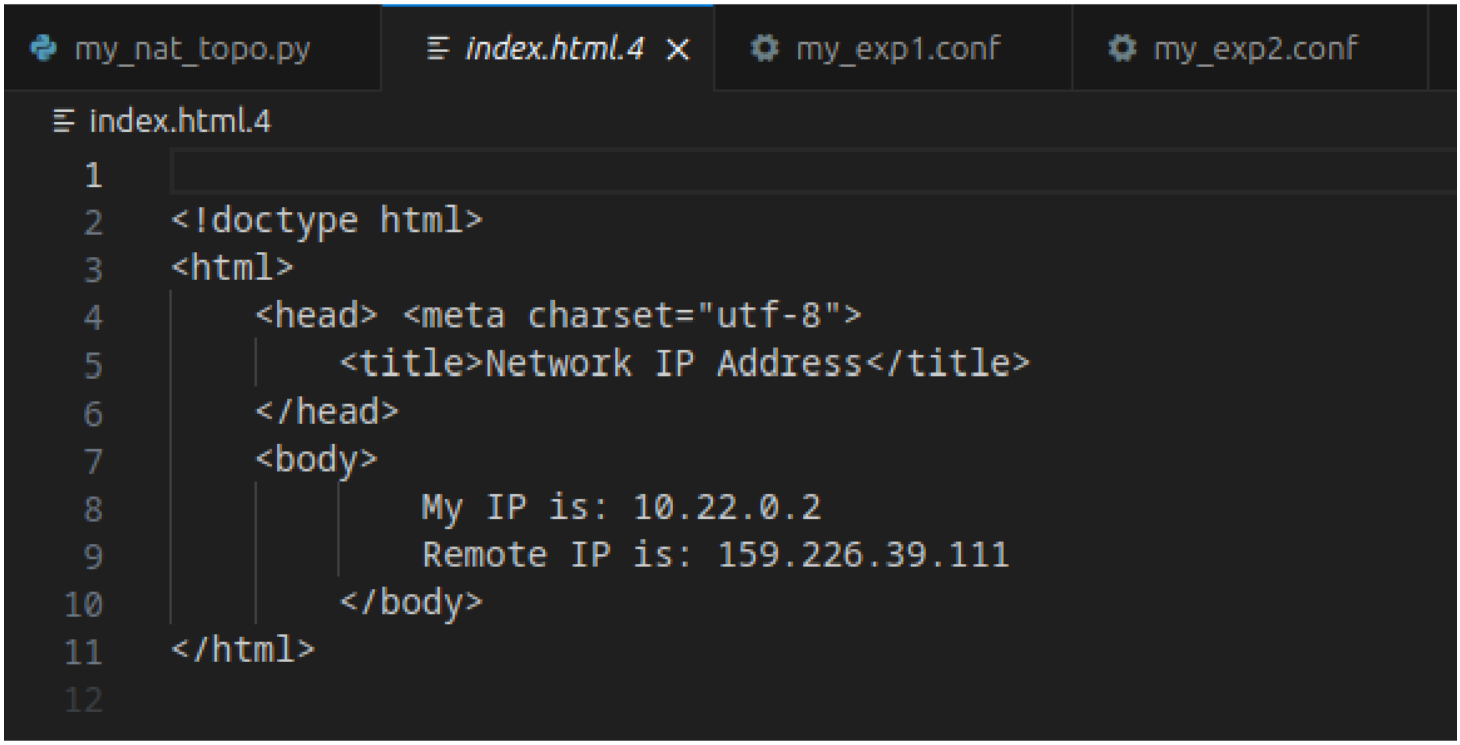
h1请求h2:

```
mininet> n1 ./nat my_exp1.conf &
DEBUG: find the following interfaces:  n1-eth0 n1-eth1.
mininet> n2 ./nat my_exp2.conf
DEBUG: find the following interfaces:  n2-eth0 n2-eth1.
Routing table of 2 entries has been loaded.
1: 9fe227de:8000,a160002:8000
^C
mininet> n2 ./nat my_exp2.conf &
mininet> h2 python3 ./http_server.py &
mininet> h1 wget  http://159.226.39.222:8000
--2023-12-21 16:38:57--  http://159.226.39.222:8000/
Connecting to 159.226.39.222:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 208 [text/html]
Saving to: 'index.html.4'

index.html.4        100%[===================>]     208  --.-KB/s    in 0s

2023-12-21 16:38:57 (51.8 MB/s) - 'index.html.4' saved [208/208]
```

生成的index文件:



Remaining Bugs:
受制于自身能力，暂未发现