网络地址转换实验报告

张翔雨 2018K8009929035

一、实验题目：网络地址转换实验

二、实验内容

- SNAT实验
  - 运行给定网络拓扑(`nat_topo.py`)
  - 在n1, h1, h2, h3上运行相应脚本
  - 在n1上运行nat程序： `n1# ./nat exp1.conf`
  - 在h3上运行HTTP服务： `h3# python ./http_server.py`
  - 在h1, h2上分别访问h3的HTTP服务
    - `h1# wget http://159.226.39.123:8000`
    - `h2# wget http://159.226.39.123:8000`
- DNAT实验
  - 运行给定网络拓扑(`nat_topo.py`)
  - 在n1, h1, h2, h3上运行相应脚本
  - 在n1上运行nat程序： `n1# ./nat exp2.conf`
  - 在h1, h2上分别运行HTTP Server： `h1/h2# python ./http_server.py`
  - 在h3上分别请求h1, h2页面
    - h3# `wget http://159.226.39.43:8000`
    - h3# `wget http://159.226.39.43:8001`
- 手动构建包含两个nat的拓扑
  - h1 <-> n1 <-> n2 <-> h2
  - 节点n1作为SNAT， n2作为DNAT，主机h2提供HTTP服务，主机h1穿过两个nat连接到h2并获取相应页面

三、实验过程

1.实现读入预设的config信息

实现`int parse_config(const char *filename)`

该函数主要功能是从conf文件中读入预置的配置信息。配置信息主要有三种：前两种 是external和internal的iface信息，要从中读出对应的iface并填入nat对应位置中，第三种是DNAT rules，需要读出两组ip和对应的port并填入rules中。

```
int parse_config(const char *filename)
{
    FILE *fd;

    if((fd = fopen(filename, "r")) == NULL)
        return -1;

    char line[100];
    while (fgets(line, 100, fd) != NULL)
    {
        if (strstr(line, "internal-iface"))
        {
            char *internal_iface;
            char *iface_name;
            internal_iface = strtok(line,": ");
            iface_name = strtok(NULL,"\n");
            iface_name ++;
            printf("%s\n",iface_name);
            iface_info_t * iface = if_name_to_iface(iface_name);
            nat.internal_iface = iface;
        }
        else if (strstr(line, "external-iface"))
        {
            char *external_iface;
            char *iface_name;
            external_iface = strtok(line,": ");
            iface_name = strtok(NULL,"\n");
            iface_name++;
            printf("%s\n",iface_name);
            iface_info_t * iface = if_name_to_iface(iface_name);
            nat.external_iface = iface;
        }
        else if (strstr(line, "dnat-rules"))
        {
            char *dnat = strtok(line, ": ");
            char *net1 = strtok(NULL, "-> ");
            char *net2 = strtok(NULL, "-> ");
            char *ip1 = strtok(net1, ":");
```

```
            char *port1 = strtok(NULL, ":");
            char *ip2 = strtok(net2, ":");
            char *port2 = strtok(NULL, ":");

            u16 external_port = atoi(port1);
            u16 internal_port = atoi(port2);
            u32 external_ip = ip_trans(ip1);
            u32 internal_ip = ip_trans(ip2);


            struct nat_mapping * entry = (struct nat_mapping *)malloc(sizeof(struct
nat_mapping));
            memset(entry,0,sizeof(struct nat_mapping));
            entry->external_ip = external_ip;
            entry->internal_ip = internal_ip;
            entry->external_port = external_port;
            entry->internal_port = internal_port;
            entry->update_time = time(NULL);

            struct dnat_rule * rule = (struct dnat_rule *)malloc(sizeof(struct
dnat_rule));
            rule->external_ip = external_ip;
            rule->internal_ip = internal_ip;
            rule->external_port = external_port;
            rule->internal_port = internal_port;
            list_add_tail(&rule->list ,&nat.rules);
        }
    }
    return 0;
}
```

2.实现对数据包的处理

（1）数据包的方向为DIR_IN

根据得到的rmt_ip和rmt_port生成哈希值，然后在对应的链表寻找是否存在对应关系，如果不存在，就要去查找是否存在对应的规则，若存在则生成一个新的表项，之后修改当前数据包头部中的信息，将其转发。

```
    struct nat_mapping *entry = NULL;
    if(dir == DIR_IN)
    {
        int find = 0;
        list_for_each_entry(entry,&nat.nat_mapping_list[index],list)
        {
            if(entry->external_ip == ntohl(iph->daddr) && entry->external_port ==
ntohs(tcph->dport))
            {
                find = 1;
```

```
                break;
            }
        }
        if(!find)
        {
            int rfind = 0;
            struct dnat_rule *rule_entry = NULL;
            list_for_each_entry(rule_entry, &nat.rules, list)
            {
                if(ntohl(iph->daddr) == rule_entry->external_ip && ntohs(tcph->dport) ==
rule_entry->external_port)
                {
                    rfind = 1;
                    break;
                }
            }
            if(rfind)
            {
                entry = (struct nat_mapping *)malloc(sizeof(struct nat_mapping));
                memset(entry,0, sizeof(struct nat_mapping));
                entry->internal_ip = rule_entry->internal_ip;
                entry->internal_port = rule_entry->internal_port;
                entry->external_ip = rule_entry->external_ip;
                entry->external_port = rule_entry->external_port;
                entry->remote_ip = rmt_ip;
                entry->remote_port = rmt_port;
                entry->update_time = time(NULL);
                list_add_tail(&entry->list, &nat.nat_mapping_list[index]);
            }
            else
            {
                pthread_mutex_unlock(&nat.lock);
                return NULL;
            }
        }
        tcph->dport = htons(entry->internal_port);
        iph->daddr = htonl(entry->internal_ip);
        entry->conn.external_fin = (tcph->flags == TCP_FIN) ? TCP_FIN : 0;
        if(ntohl(tcph->seq) > entry->conn.external_seq_end)
            entry->conn.external_seq_end = ntohl(tcph->seq);
        if(ntohl(tcph->ack) > entry->conn.external_ack && tcph->flags == TCP_ACK)
            entry->conn.external_ack = ntohl(tcph->ack);
    }
```

（2）数据包的方向为DIR_OUT

如果方向是`DIR_OUT`,则查找后如果没找到直接生成一个新的表项并为其分配一个没有使用过的端口，之后修改包头部的内容并转发。

```
    else if (dir == DIR_OUT)
```

```
    {
        int find = 0;
        list_for_each_entry(entry,&nat.nat_mapping_list[index],list)
        {
            if(entry->internal_ip == ntohl(iph->saddr) && entry->internal_port ==
ntohs(tcph->sport))
            {
                find = 1;
                break;
            }
        }
        if(!find)
        {
            entry = (struct nat_mapping*)malloc(sizeof(struct nat_mapping));
            memset(entry,0, sizeof(struct nat_mapping));
            entry->internal_ip = ntohl(iph->saddr);
            entry->internal_port = ntohs(tcph->sport);
            entry->external_ip = nat.external_iface->ip;
            entry->external_port = alloc_port();
            entry->remote_ip = rmt_ip;
            entry->remote_port = rmt_port;
            entry->update_time = time(NULL);
            list_add_tail(&entry->list, &nat.nat_mapping_list[index]);
        }
        tcph->sport = htons(entry->external_port);
        iph->saddr = htonl(entry->external_ip);
        if(tcph->flags == TCP_FIN)
            entry->conn.internal_fin = TCP_FIN;
        if(ntohl(tcph->seq) > entry->conn.internal_seq_end)
            entry->conn.internal_seq_end = ntohl(tcph->seq);
        if(ntohl(tcph->ack) > entry->conn.internal_ack && tcph->flags == TCP_ACK)
            entry->conn.internal_ack = ntohl(tcph->ack);
    }
```

3.实现NAT的老化过程

老化过程主要对两类数据包进行回收，一种是"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；第二种是"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。

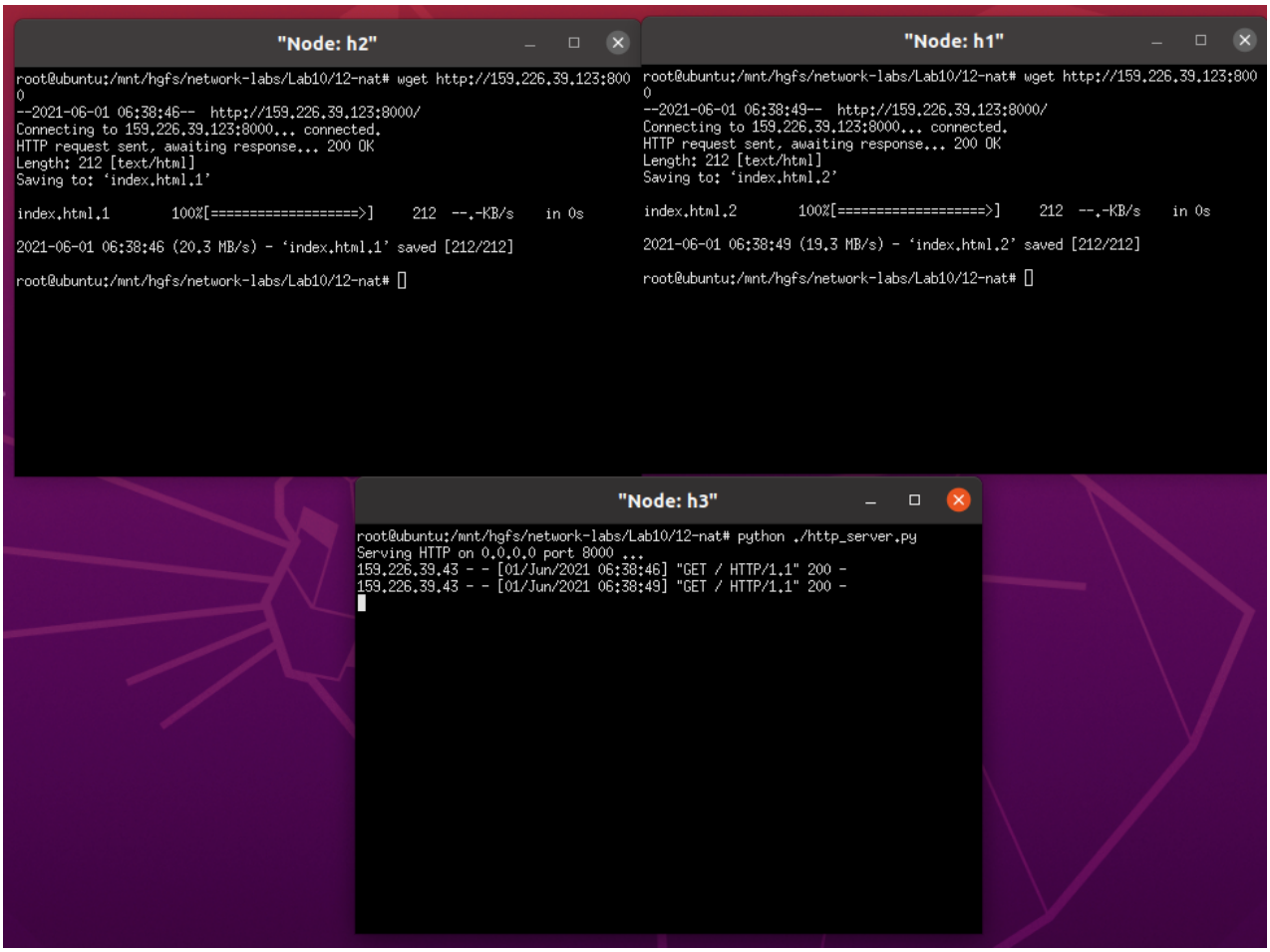
```
void *nat_timeout()
{
    while (1) {
        fprintf(stdout, "TODO: sweep finished flows periodically.\n");
        pthread_mutex_lock(&nat.lock);
        for (int i = 0; i < HASH_8BITS; i++)
        {
            struct nat_mapping * entry, *q;
            list_for_each_entry_safe (entry, q, &nat.nat_mapping_list[i], list)
            {
```

```
                if (time(NULL) - entry->update_time > TCP_ESTABLISHED_TIMEOUT)
                {
                    nat.assigned_ports[entry->external_port] = 0;
                    list_delete_entry(&entry->list);
                    free(entry);
                }
                else if (is_flow_finished(&entry->conn))
                {
                    nat.assigned_ports[entry->external_port] = 0;
                    list_delete_entry(&entry->list);
                    free(entry);
                }
            }
        }
        pthread_mutex_unlock(&nat.lock);
        sleep(1);
    }

    return NULL;
}
```

四、实验结果

1.实验内容1：SNAT实验

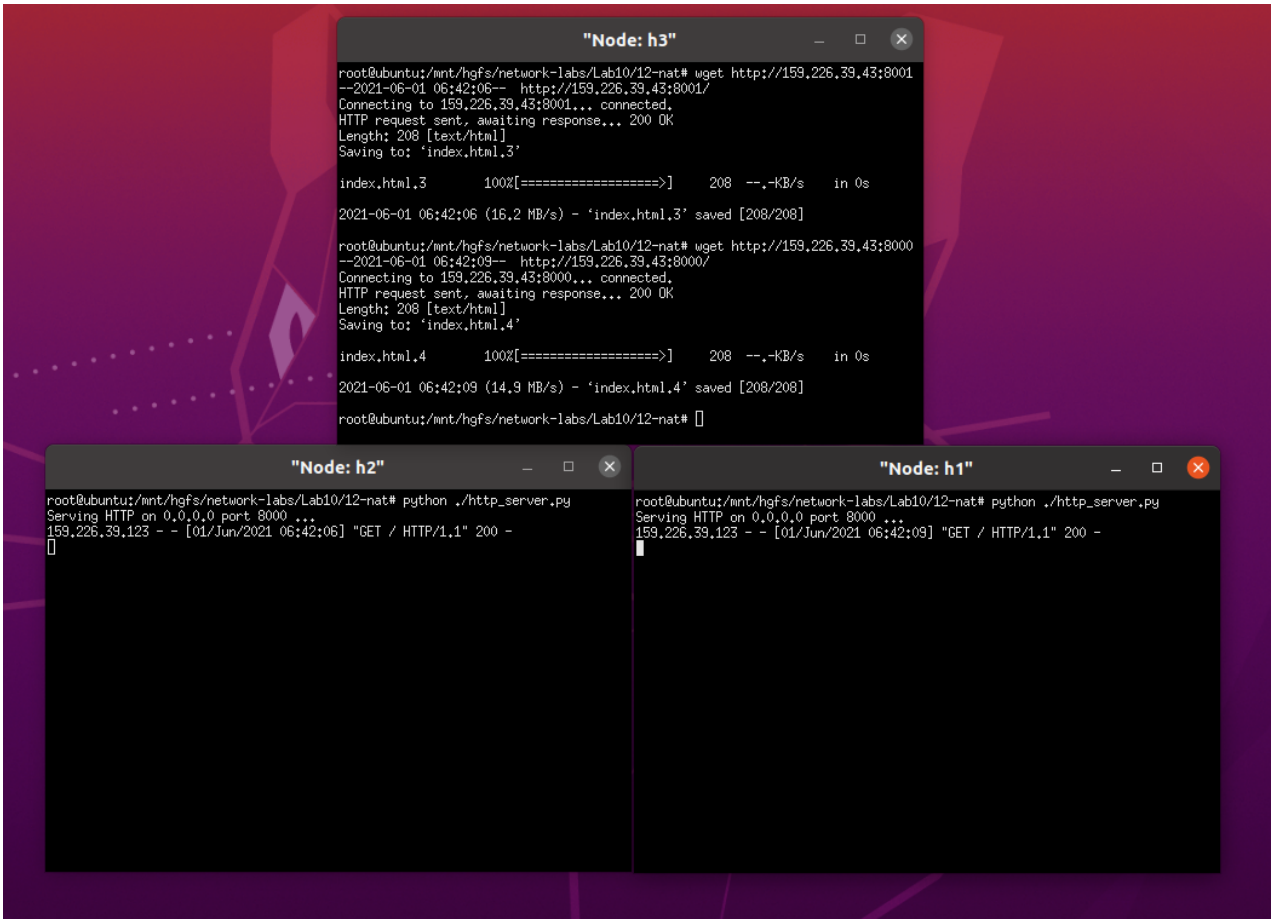


其中h3从h1和h2收到的html信息相同，如下：

```
<!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>
```

可以看到产生了正确的结果

2.实验内容2：DNAT实验



查看分别从h1，h2收到的数据

h1:

```
<!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>
```

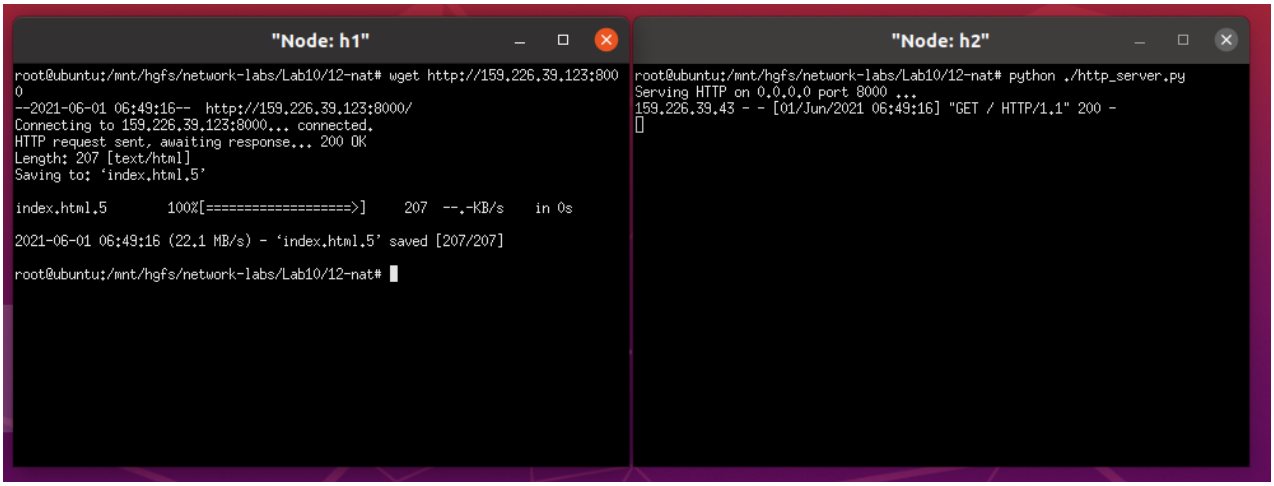
h2:

```
<!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>
```

可以看到收到了正确的结果

实验内容3：构建包含两个nat的拓扑

路径连接为：`h1 <-> n1 <-> n2 <-> h2`，n1和n2分别导入对应conf文件，h1的ip为`10.21.0.1`，h2的ip为`10.11.0.1`，在h2上运行服务器程序，h1向h2发送wget请求。结果如下



其中h1收到的内容如下：

```
<!doctype html>
<html>
    <head> <meta charset="utf-8">
        <title>Network IP Address</title>
    </head>
    <body>
            My IP is: 10.11.0.1
            Remote IP is: 159.226.39.43
        </body>
</html>
```

实验三成功。

五、思考题

1.实验中的NAT系统可以很容易实现支持UDP协议，现实网络中NAT还需要对ICMP进行地址翻译，请调研说明NAT系统如何支持ICMP协议。

这里首先回顾一下ICMP报文的格式：（节选自ICMP报文如何通过NAT来地址转换_sinat_33822516的博客-CSDN博客）

类型（Type）	代码（Code）	检验和（Checksum）
标识符（Identifier）		序列号（Sequence number）
选项（Option）		

在一个主机发送ICMP报文的时候，会根据（Type+Code）的值生成源端口号，根据Identifier的值生成目的端口号，即发送到路由器的报文如下：

Source IP	Source Port	Dest IP	Dest Port
192.168.0.2	（Type+Code）	200.10.2.1	Identifier

在路由器上进行SNAT，源IP更改后ICMP报文中的Identifier会改变，记作IDENTIFIER。这时候的报文如下：

Source IP	Source Port	Dest IP	Dest Port
188.10.1.2	IDENTIFIER	200.10.2.1	Identifier

NAT表：

Source IP	Source Port	Dest IP	Dest Port	协议
192.168.0.2	（Type+Code）	188.10.1.2	IDENTIFIER	ICMP

在web服务器收到ICMP请求后，生成ICMP响应报文，响应报文中的（Type+Code）会作为源端口，IDENTIFIER作为目的端口

源报文：

Source IP	Source Port	Dest IP	Dest Port
200.10.2.1	（Type+Code）	188.10.1.2	IDENTIFIER

报文到达路由器后，根据NAT表中，查询目的IP和目的端口为188.10.1.2和IDENTIFIER的信息。将目的IP和目的端口换为

192.168.0.2和（Type+Code）,这样报文就可以成功的到达了。

总的来说就是NAT根据ICMP的头部生成了虚拟的端口，按照TCP报文处理。

2.给定一个有公网地址的服务器和两个处于不同内网的主机，如何让两个内网主机建立TCP连接并进行数据传输。

1. 服务器启动两个网络侦听，一个做主连接，一个协助完成TCP打洞。
2. 两个主机分别与服务器的主连接保持联系。
3. 当两个主机建立直接的TCP连接时，首先连接服务器协助打洞的端口，并发送协助连接申请。同时在该端口号上启动侦听。
4. 服务器协助打洞的连接收到主机1的申请后通过主连接的侦听通知主机2，并将1经过NAT转换后的公网IP地址和端口等信息告诉2。
5. 2收到服务器的连接通知后首先与服务器协助端口连接，发送一些数据后立即断开，让服务器能知道2经过NAT转换后的公网IP和端口号。
6. 2尝试与1的经过NAT转换后的公网IP地址和端口进行connect，NAT会纪录此次连接的源地址和端口号，为接下来连接做好了准备，这就完成了打洞，即2向1打了一个洞，1就可以直接连接2刚刚使用的端口号
7. 2打洞的同时在相同的端口上启动侦听。2在一切准备就绪以后通过服务器告知1已经准备好，服务器将2经过NAT转换后的公网IP和端口号告诉给1。
8. 1收到2的公网IP和端口号等信息以后，开始连接到2的公网IP和端口号，由于在步骤6中2曾经尝试连接过1的公网IP地址和端口，NAT纪录了此次连接的信息，所以当1主动连接2时，NAT会认为是合法的数据，并允许通过，从而建立了直接的NAT连接

参考链接：TCP 打洞，内网穿透 - Guozht - 博客园 (cnblogs.com)

六、实验总结

本次实验相比较前几次实验难度有所下降，主体部分的代码结果很清晰，函数数量比较少，能够较快的完成，同时也串联了前后两个层次的实验，是个很好的过渡。