网络地址转换(NAT)实验

2020年12月8日

蔡润泽

本实验 `Github` 地址

实验内容

实验内容一

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
  - n1: disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh, disable_ipv6.sh
  - h1-h3: disable_offloading.sh, disable_ipv6.sh
- 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

实验内容二

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
  - n1: disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh, disable_ipv6.sh
  - h1-h3: disable_offloading.sh, disable_ipv6.sh
- 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

实验内容三

- 手动构造一个包含两个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

设计思路

配置config文件

`int parse_config(const char *filename)` 函数

负责根据config文件中读取的每一行字符串，分别配置`external-iface`，`internal-iface`以及`DNAT Rules`。 具体实现如下：

```
int parse_config(const char *filename) {
        FILE * fd = fopen(filename, "r");
        if (fd == NULL) {
                return 0;
        }
        char * line = (char *)malloc(MAX_LINE);
        while (fgets(line, MAX_LINE, fd) != NULL) {
                if (strstr(line, internal_iface_des)) {
                        parse_internal_iface(line);
                } else if (strstr(line, external_iface_des)) {
                        parse_external_iface(line);
                } else if (strstr(line, dnat_rules_des)) {
                        parse_dnat_rules(line);
                }
        }
        return 0;
}
```

其中调用了三个自己编写的parse函数，负责具体条目的配置。

区分数据包方向

`static int get_packet_direction(char *packet)` 函数

负责返回数据包方向：

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

具体设计时，可以根据源地址进行判断。具体实现如下：

```
static int get_packet_direction(char *packet)
{
        struct iphdr * ip = packet_to_ip_hdr(packet);
        rt_entry_t * rt = longest_prefix_match(ntohl(ip->saddr));
        if (rt->iface->index == nat.internal_iface->index) {
                return DIR_OUT;
        } else if (rt->iface->index == nat.external_iface->index) {
                return DIR_IN;
        }
        return DIR_INVALID;
}
```

合法数据包的处理

该数据包在NAT中有对应连接映射 (Existing)

根据Hash Mapping，若能找到存在的映射条目，则直接进行(internal_ip, internal_port) <-> (external_ip, external_port)之间的转换。

其中 Hash Mapping的设计如下：

根据（rmt_ip,rmt_port）生成Hash值，再根据数据包的方向匹配具体的条目。这部分涉及到的部分代码如下：

```
u32 addr = (dir == DIR_IN)? ntohl(ip_hdr->saddr) : ntohl(ip_hdr->daddr);
u16 port = (dir == DIR_IN)? ntohs(tcp_hdr->sport) : ntohs(tcp_hdr->dport);
rmt_set_t * rmt_set = (rmt_set_t *)malloc(sizeof(rmt_set_t));
bzero(rmt_set, sizeof(rmt_set_t));
rmt_set->ip = ntohl(addr);
rmt_set->port = (int)port;
u8 hash = hash8((char*)rmt_set, 8);
struct nat_mapping * mapping_entry = NULL;

if (dir == DIR_IN) {
    int isExisting = 0;
    list_for_each_entry(mapping_entry, &nat.nat_mapping_list[hash], list) {
        if (mapping_entry->external_ip == ntohl(ip_hdr->daddr) && mapping_entry->exterr
            isExisting = 1;
            break;
        }
    }
...
} else if (dir == DIR_OUT) {
    int isExisting = 0;
    list_for_each_entry(mapping_entry, &nat.nat_mapping_list[hash], list) {
        if (mapping_entry->internal_ip == ntohl(ip_hdr->saddr) && mapping_entry->interr
            isExisting = 1;
            break;
        }
}
...
```

该数据包的方向为DIR_OUT

NAT中没有对应连接映射(SNAT)，此时为该TCP连接的第一个数据包（请求连接数据包）。

此时需要分配一个新的端口，并建立映射表项，写入到对应的Hash链表当中。具体的实现如下：

```
if (!isExisting) {
    mapping_entry = (struct nat_mapping*)malloc(sizeof(struct nat_mapping));
    bzero(mapping_entry, sizeof(struct nat_mapping));
    mapping_entry->internal_ip = ntohl(ip_hdr->saddr);
    mapping_entry->external_ip = nat.external_iface->ip;
    mapping_entry->internal_port = ntohs(tcp_hdr->sport);
    mapping_entry->external_port = assign_port();
    list_add_tail(&mapping_entry->list, &nat.nat_mapping_list[hash]);
}

tcp_hdr->sport = htons(mapping_entry->external_port);
ip_hdr->saddr = htonl(mapping_entry->external_ip);
mapping_entry->conn.internal_fin = (tcp_hdr->flags == TCP_FIN)? TCP_FIN : 0;
mapping_entry->conn.internal_seq_end = tcp_hdr->seq;
if (tcp_hdr->flags == TCP_ACK) {
    mapping_entry->conn.internal_ack = tcp_hdr->ack;
}
```

其中，assign_port()函数负责通过遍历可用端口的方式，找到并分配一个新的端口。

该数据包的方向为DIR_IN

NAT中没有对应连接映射，但有对应处理规则 (DNAT)，此时为该TCP连接的第一个数据包。

此时根据遍历已有的DNAT Rules,建立映射表项，写入到对应的Hash链表当中。具体的实现如下：

```
if (!isExisting) {
    mapping_entry = (struct nat_mapping*)malloc(sizeof(struct nat_mapping));
    bzero(mapping_entry, sizeof(struct nat_mapping));
    mapping_entry->external_ip = ntohl(ip_hdr->daddr);
    mapping_entry->external_port = ntohs(tcp_hdr->dport);
    check_rules(mapping_entry);
    list_add_tail(&mapping_entry->list, &nat.nat_mapping_list[hash]);
}
tcp_hdr->dport = htons(mapping_entry->internal_port);
ip_hdr->daddr = htonl(mapping_entry->internal_ip);
mapping_entry->conn.external_fin = (tcp_hdr->flags == TCP_FIN)? TCP_FIN : 0;
mapping_entry->conn.external_seq_end = tcp_hdr->seq;
if (tcp_hdr->flags == TCP_ACK) {
    mapping_entry->conn.external_ack = tcp_hdr->ack;
}
```

其中，check_rules(mapping_entry)函数负责通过遍历DNAT Rules，找到并建立映射。

NAT老化（Timeout）操作

对认为已经结束的连接进行老化操作

- 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。

- 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。

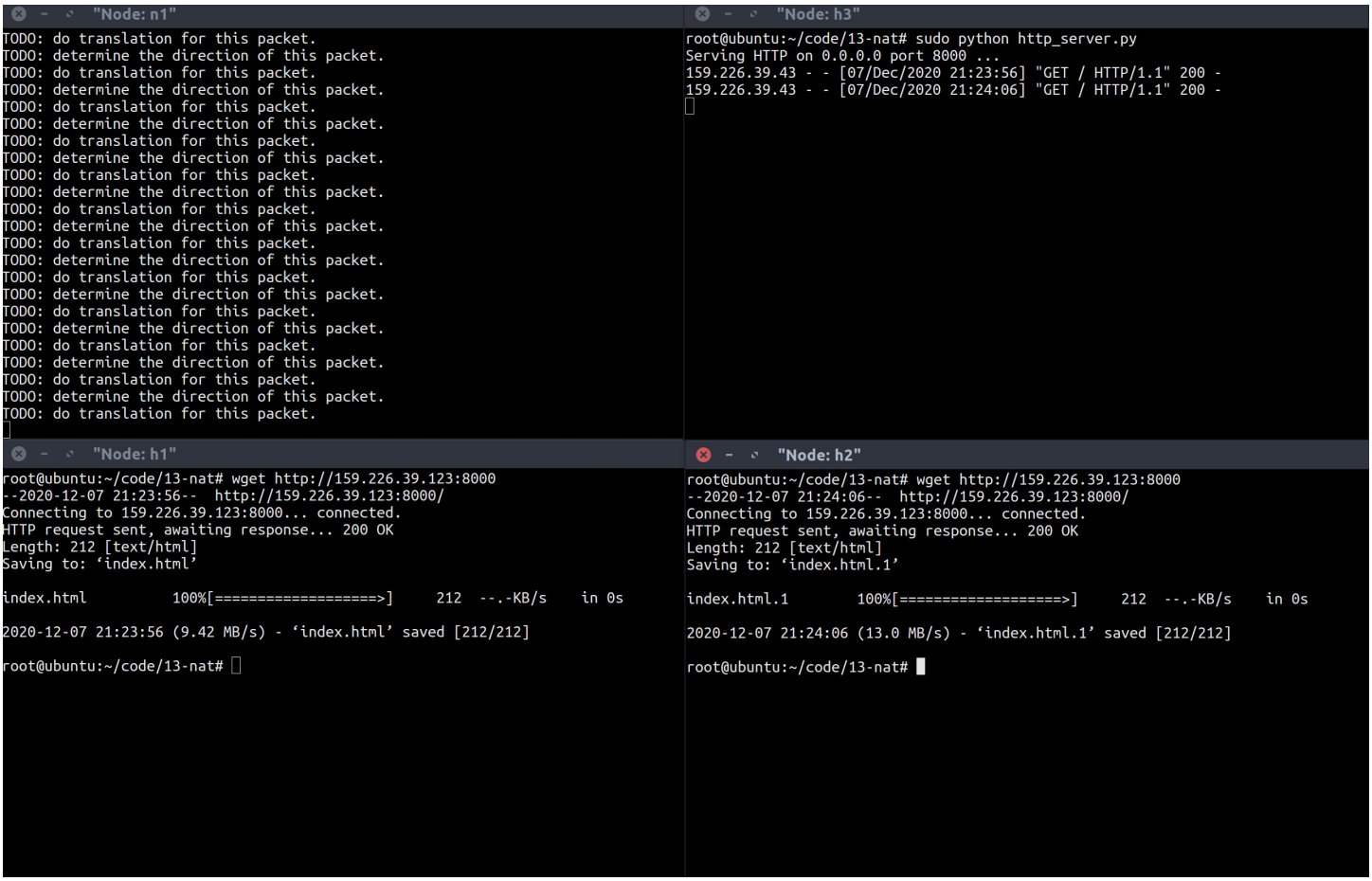
具体实现如下：

```
void *nat_timeout() {
        while (1) {
                sleep(1);
                pthread_mutex_lock(&nat.lock);
                for (int i = 0; i < HASH_8BITS; i++) {
                        struct nat_mapping * mapping_entry, *mapping_entry_q;
                        list_for_each_entry_safe (mapping_entry, mapping_entry_q, &nat.
                                if (time(NULL) - mapping_entry->update_time > TCP_ESTAE
                                        nat.assigned_ports[mapping_entry->external_port
                                        list_delete_entry(&mapping_entry->list);
                                        free(mapping_entry);
                                        continue;
                                }
                                if (is_flow_finished(&mapping_entry->conn)) {
                                        nat.assigned_ports[mapping_entry->external_port
                                        list_delete_entry(&mapping_entry->list);
                                        free(mapping_entry);
                                }
                        }
                }
                pthread_mutex_unlock(&nat.lock);
        }
        return NULL;
}
```

结果验证

实验一结果

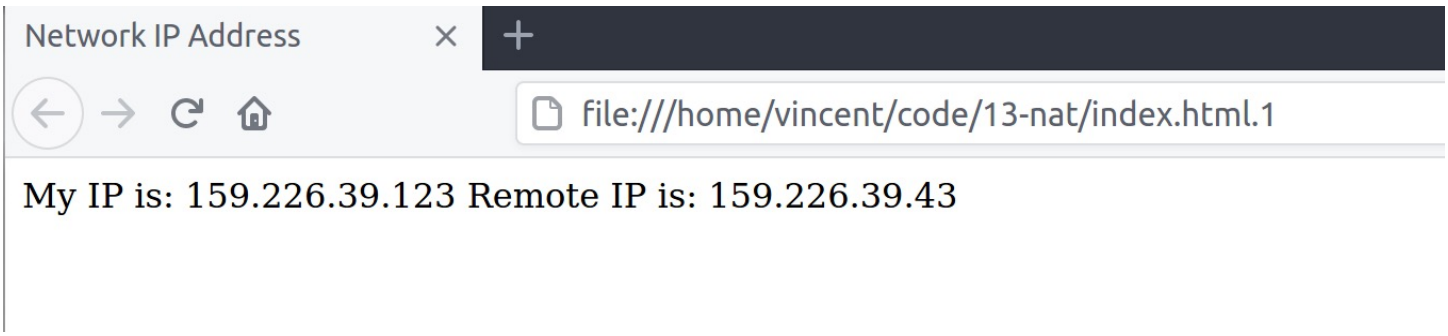
实验结果如下：



上图可知，H1和H2作为客户端向服务器端H3请求了两个网页。请求结果如下：

H3->H1:

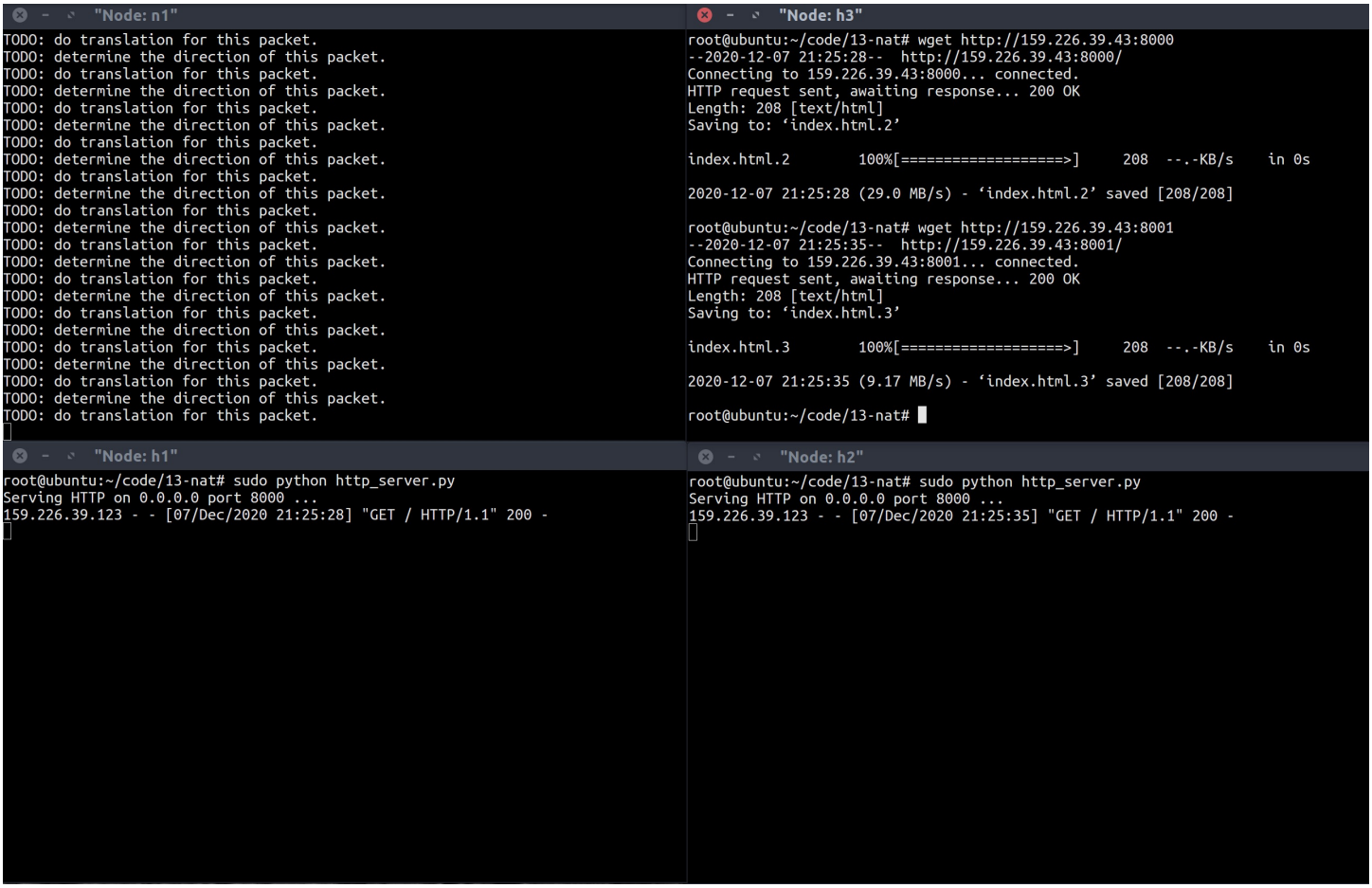
H3->H2:



由上述结果可知，SNAT转换实验成功。

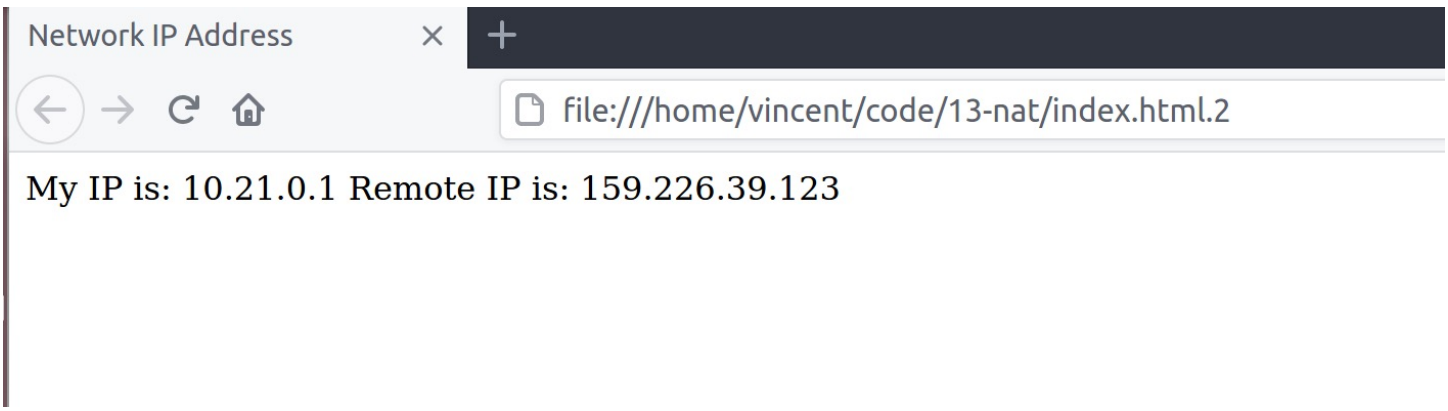
实验二结果

实验结果如下：



上图可知，H1和H2作为服务器端，H3作为客户端分别向H1和H2请求了两个网页。请求结果如下：

H1 -> H3:



H2 -> H3:

由上述结果可知，DNAT转换实验成功。

实验三结果

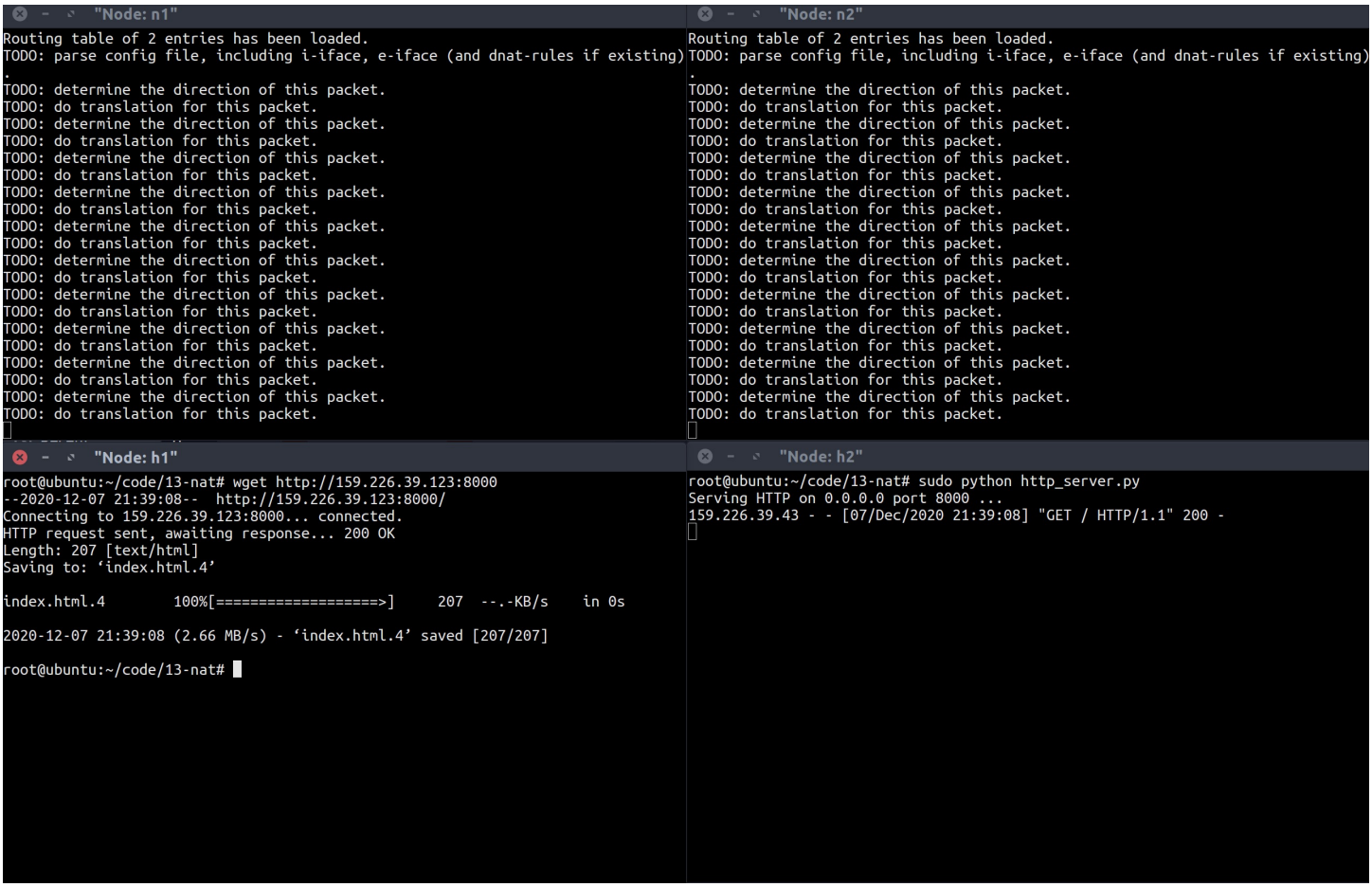
自建的网络拓扑文件如下：

```
    h1.cmd('ifconfig h1-eth0 10.21.0.1/16')
    h1.cmd('route add default gw 10.21.0.254')

    h2.cmd('ifconfig h2-eth0 10.22.0.1/16')
    h2.cmd('route add default gw 10.22.0.254')

    n1.cmd('ifconfig n1-eth0 10.21.0.254/16')
    n1.cmd('ifconfig n1-eth1 159.226.39.43/24')

    n2.cmd('ifconfig n2-eth0 10.22.0.254/16')
    n2.cmd('ifconfig n2-eth1 159.226.39.123/24')
```

即 `H1 <-> N1 <-> N2 <-> H2`

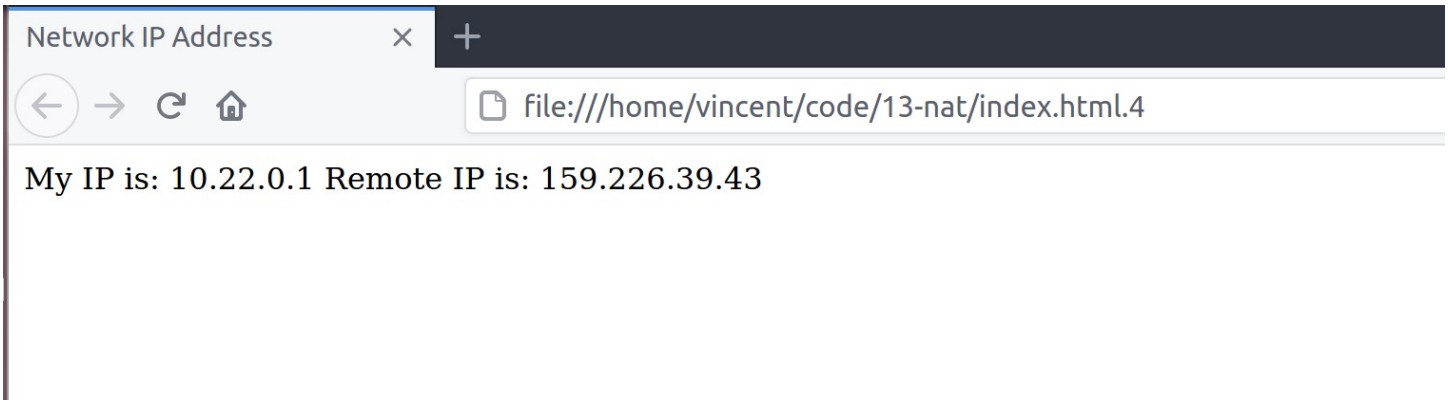
节点N1作为SNAT， N2作为DNAT，主机H2提供HTTP服务，主机H1穿过两个nat连接到h2并获取相应页面。

实验结果如下：



上图可知，H1和H2作为客户端向服务器端H3请求了两个网页。请求结果如下：

H2 -> H1:



由上述结果可知，实验三成功。

调研NAT系统如何支持ICMP协议

ICMP的报文格式如下：

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

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

源报文:

源IP	源端口	目的IP	目的端口
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。

此时报文如下：

源IP	源端口	目的IP	目的端口
188.10.1.2	IDENTIFIER	200.10.2.1	Identifier

对应的NAT表:

源IP	源端口	协议	目的IP	目的端口
192.168.0.2	(Type+Code)	ICMP	188.10.1.2	IDENTIFIER

在服务器收到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

源IP	源端口	目的IP	目的端口
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）,报文就可以成功的到达客户端了。

参考资料

ICMP报文如何通过NAT来地址转换