

# 网络地址转换实验

学号： 2021K8009929010

姓名： 贾城昊

## 一、 实验题目：网络地址转换实验

## 二、 实验任务：

了解 NAT 地址转换原理，自己实现 NAT 设备。通过实验验证 SNAT、DNAT、多 NAT 功能正确性。

## 三、 实验流程

### 实验内容一：SNAT 实验

1. 运行给定网络拓扑(nat\_topo.py)在 n1, h1, h2, h3 上运行相应脚本
2. 在 n1 上运行 nat 程序
3. 在 h3 上运行 HTTP 服务
4. 在 h1, h2 上分别访问 h3 的 HTTP 服务

### 实验内容二：DNAT 实验

1. 运行给定网络拓扑(nat\_topo.py)
2. 在 n1 上运行 nat 程序
3. 在 h1, h2 上分别运行 HTTP Serve
4. 在 h3 上分别请求 h1, h2 页面

### 实验内容三：多 NAT 实验

#### 1. 手动构造一个包含两个 nat 的拓扑：

- $h1 \leftrightarrow n1 \leftrightarrow n2 \leftrightarrow h2$
- 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

## 四、 实验过程

### （一）配置信息的读取

NAT 的配置由 `parse_config` 函数处理，从配置文件中提取配置信息并进行配置。主要流程是字符串匹配。首先完成 `internal` 和 `external` 端口的配置。然后查看有无 `dnat-rules` 信息，有的话将其添加到 `rules` 列表。

具体实现方式如下：。

```

while (fgets(line, MAX_LINE_LEN, fp)) {
    char *name_end = line;

    if (line[0] == 'i') {
        char* internal = line + 16;
        nat.internal_iface = if_name_to_iface(internal);
        log(DEBUG, "internal_iface: "IP_FMT"\n", HOST_IP_FMT_STR(nat.internal_iface->ip));
        continue;
    }
    else if (line[0] == 'e') {
        char* external = line + 16;
        nat.external_iface = if_name_to_iface(external);
        log(DEBUG, "external_iface: "IP_FMT"\n", HOST_IP_FMT_STR(nat.external_iface->ip));
        continue;
    }
    else if (line[0] == 'd') {
        struct dnat_rule *new_rule = (struct dnat_rule *)malloc(sizeof(struct dnat_rule));
        memset(new_rule, 0, sizeof(struct dnat_rule));

        char* drule = line + 12;
        new_rule->external_ip = ip_to_u32(drule);

        while(*drule != ':')
            drule++;
        drule++;
        new_rule->external_port = atoi(drule);

        while(*drule != '-')
            drule++;
        drule += 3;
        new_rule->internal_ip = ip_to_u32(drule);

        while(*drule != ':')
            drule++;
        drule++;
        new_rule->internal_port = atoi(drule);

        init_list_head(&new_rule->list);
        list_add_tail(&new_rule->list, &nat.rules);

        nat.assigned_ports[new_rule->external_port] = 1;

        log(DEBUG, "dnat_rule: "IP_FMT" %d "IP_FMT" %d\n", HOST_IP_FMT_STR(new_rule->external_ip),
            new_rule->external_port, HOST_IP_FMT_STR(new_rule->internal_ip), new_rule->internal_port);
        continue;
    }
}

```

## (二) NAT 地址转换

### 1. 总体逻辑

首先调用 `get_packet_direction` 函数判断数据包方向。将不可达数据包或非 TCP 数据包丢弃，并发送 ICMP 报文。最后调用 `do_translation` 函数完成实际地址转换和数据包发送。

```

void nat_translate_packet(iface_info_t *iface, char *packet, int len)
{
    int dir = get_packet_direction(packet);
    if (dir == DIR_INVALID) {
        log(ERROR, "invalid packet direction, drop it.");
        icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_HOST_UNREACH);
        free(packet);
        return ;
    }

    struct iphdr *ip = packet_to_ip_hdr(packet);
    if (ip->protocol != IPPROTO_TCP) {
        log(ERROR, "received non-TCP packet (0x%0hhx), drop it", ip->protocol);
        free(packet);
        return ;
    }

    do_translation(iface, packet, len, dir);
}

```

## 2. 区分数据包方向

get\_packet\_direction 函数用于返回数据包方向：当源地址为内部地址，且目的地地址为外部地址时，方向为 DIR\_OUT；当源地址为外部地址，且目的地地址为 external\_iface 地址时，方向为 DIR\_IN；否则，返回 DIR\_INVALID。

具体实现的代码如下：

```

// determine the direction of the packet, DIR_IN / DIR_OUT / DIR_INVALID
static int get_packet_direction(char *packet)
{
    //fprintf(stdout, "TODO: determine the direction of this packet.\n");

    struct iphdr *ip = packet_to_ip_hdr(packet);
    u32 saddr = ntohl(ip->saddr);
    u32 daddr = ntohl(ip->daddr);
    rt_entry_t *src_entry = longest_prefix_match(saddr);
    rt_entry_t *dst_entry = longest_prefix_match(daddr);

    if ((src_entry->iface == nat.internal_iface) && (dst_entry->iface == nat.external_iface)) {
        return DIR_OUT;
    }
    else if ((src_entry->iface == nat.external_iface) && (daddr == nat.external_iface->ip)) {
        return DIR_IN;
    }
    else{
        return DIR_INVALID;
    }
}

```

### 3. 数据包翻译更新

只有合法数据包会进入翻译阶段。翻译阶段大致可以分为查找已有连接和创建新连接两种策略，后者针对无法查找到已有连接的 TCP。不论采用何种处理方式，都首先需要根据数据包方向确定远端地址和端口号，依此计算已有或新建连接在连接映射表中的哈希索引值。

```
struct iphdr *iphdr = packet_to_ip_hdr(packet);
struct tcphdr *tcphdr = packet_to_tcp_hdr(packet);

u32 daddr = ntohl(iphdr->daddr);
u32 saddr = ntohl(iphdr->saddr);
u32 raddr = (dir == DIR_IN) ? saddr : daddr;
u16 sport = ntohs(tcphdr->sport);
u16 dport = ntohs(tcphdr->dport);
u16 rport = (dir == DIR_IN) ? sport : dport;

u8 idx = rmt_hash(raddr, rport);
struct list_head *head = &nat.nat_mapping_list[idx];
struct nat_mapping *entry;
```

Hash 表存储映射关系是(rmt\_ip, rmt\_port)到一个 nat\_mapping 的链表，其映射函数如下：

```
static u8 rmt_hash(u32 addr, u16 port) {
    char str[6];
    memset(str, 0, 6 * sizeof(char));
    memcpy(str, &addr, sizeof(u32));
    memcpy(str + 4, &port, sizeof(u16));

    u8 res = hash8(str, 6);
    return res;
}
```

之后需要先遍历已建立连接表。查找连接时的匹配条件包括 4 项记录的匹配：远端 IP 和远端端口、内部 IP 和端口号(对于来自内网的数据包)或外部 IP 和端口号(对于来自外网的数据包)。一旦匹配，就将 IP 头部和 TCP 头部的源地址或目的地址修改为映射记录中的对应的两项条目。同时还需要根据 TCP 报头内容，更新连接控制数据结构、最近连接时间，供老化线程检查。处理完毕后重新计算 TCP 和 IP 部分的校验和，转发数据包即

可。

具体这部分代码如下所示：

```
pthread_mutex_lock(&nat.lock);
list_for_each_entry(entry, head, list) {
    if (raddr != entry->remote_ip || rport != entry->remote_port){
        continue;
    }

    int clear = (tcphdr->flags & TCP_RST) ? 1 : 0;

    if (dir == DIR_IN) {
        if (daddr != entry->external_ip || dport != entry->external_port){
            continue;
        }
        iphdr->daddr = htonl(entry->internal_ip);
        tcphdr->dport = htons(entry->internal_port);

        entry->conn.external_fin = (tcphdr->flags & TCP_FIN) ? 1 : 0;
        entry->conn.external_seq_end = tcp_seq_end(iphdr, tcphdr);
        if (tcphdr->flags & TCP_ACK){
            entry->conn.external_ack = tcphdr->ack;
        }
    }
    else {
        if (saddr != entry->internal_ip || sport != entry->internal_port){
            continue;
        }
        iphdr->saddr = htonl(entry->external_ip);
        tcphdr->sport = htons(entry->external_port);
        entry->conn.internal_fin = (tcphdr->flags & TCP_FIN) ? 1 : 0;
        entry->conn.internal_seq_end = tcp_seq_end(iphdr, tcphdr);
        if (tcphdr->flags & TCP_ACK){
            entry->conn.internal_ack = tcphdr->ack;
        }
    }

    pthread_mutex_unlock(&nat.lock);

    entry->update_time = time(NULL);
    tcphdr->checksum = tcp_checksum(iphdr, tcphdr);
    iphdr->checksum = ip_checksum(iphdr);
    ip_send_packet(packet, len);

    if (clear) {
        nat.assigned_ports[entry->external_port] = 0;
        list_delete_entry(&(entry->list));
        free(entry);
    }
}
return;
```

若查找失败，首先检查该数据包是否为 SYN 包，即是否为建立连接的 TCP 包。若不是，则作为无效数据包处理。

```

if ((tcphdr->flags & TCP_SYN) == 0) {
    fprintf(stderr, "Invalid packet!\n");
    icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_HOST_UNREACH);
    free(packet);
    pthread_mutex_unlock(&nat.lock);
    return;
}

```

如果没有找到映射表项，则需要建立新的映射关系。对于公网发起连接的情况，只需要直接遍历检索初始化时导入的映射规则。若找到了匹配的条目，则按该条目的映射关系填写连接记录并加入索引，生成连接映射条目，修改 IP 头及 TCP 头的源地址及端口信息，更新校验和后转发数据包，并进行转发。

```

if (dir == DIR_OUT) {
    ul6 pid;
    for (pid = NAT_PORT_MIN; pid <= NAT_PORT_MAX; ++pid) {
        if (!nat.assigned_ports[pid]) {
            struct nat_mapping *new_entry = (struct nat_mapping *) malloc(sizeof(struct nat_mapping));
            list_add_tail(&new_entry->list, head);

            new_entry->remote_ip = raddr;
            new_entry->remote_port = rport;
            new_entry->external_ip = nat.external_iface->ip;
            new_entry->external_port = pid;
            new_entry->internal_ip = saddr;
            new_entry->internal_port = sport;

            new_entry->conn.internal_fin = ((tcphdr->flags & TCP_FIN) != 0);
            new_entry->conn.internal_seq_end = tcp_seq_end(iphdr, tcphdr);
            if (tcphdr->flags & TCP_ACK){
                new_entry->conn.internal_ack = tcphdr->ack;
            }

            new_entry->update_time = time(NULL);
            pthread_mutex_unlock(&nat.lock);

            iphdr->saddr = htonl(new_entry->external_ip);
            tcphdr->sport = htons(new_entry->external_port);
            tcphdr->checksum = tcp_checksum(iphdr, tcphdr);
            iphdr->checksum = ip_checksum(iphdr);
            ip_send_packet(packet, len);
            return;
        }
    }
}

```

对于私网发起连接的情况，大致相同。但是此时改为了遍历所有可分配给 SNAT 连接的端口(本实验中设置为 12345 到 23456 号端口)，找到一个未使用的端口，然后与公网发起连接的情况的处理一样，填写映射信息并记录连接状态后加入哈希索引得到的表项中，生成连接映射条目，修改 IP 头及 TCP 头的源地址及端口信息，更新校验和后转发数据包即可。

```

    else {
        u16 pid;
        for (pid = NAT_PORT_MIN; pid <= NAT_PORT_MAX; ++pid) {
            if (!nat.assigned_ports[pid]) {
                struct nat_mapping *new_entry = (struct nat_mapping *) malloc(sizeof(struct nat_mapping));
                list_add_tail(&new_entry->list, head);

                new_entry->remote_ip = raddr;
                new_entry->remote_port = rport;
                new_entry->external_ip = nat.external_iface->ip;
                new_entry->external_port = pid;
                new_entry->internal_ip = saddr;
                new_entry->internal_port = sport;

                new_entry->conn.internal_fin = ((tcphdr->flags & TCP_FIN) != 0);
                new_entry->conn.internal_seq_end = tcp_seq_end(iphdr, tcphdr);
                if (tcphdr->flags & TCP_ACK){
                    new_entry->conn.internal_ack = tcphdr->ack;
                }

                new_entry->update_time = time(NULL);
                pthread_mutex_unlock(&nat.lock);

                iphdr->saddr = htonl(new_entry->external_ip);
                tcphdr->sport = htons(new_entry->external_port);
                tcphdr->checksum = tcp_checksum(iphdr, tcphdr);
                iphdr->checksum = ip_checksum(iphdr);
                ip_send_packet(packet, len);
                return;
            }
        }
    }
}

```

最后，如果 DNAT 包无法查询到 DNAT 规则或者 SNAT 包无法分配新端口，将作为无效数据包处理，发送 ICMP 消息。

### （三）端口映射记录老化及清除

老化线程每隔 1 秒唤醒一次，遍历检查 NAT 的端口映射记录。将超时没有传输数据、已经握手完毕断开连接的条目删除，并释放端口占用，使之可以被重新使用。对于前者，只需计算连接状态最后一次被更新的时间与当前时间的差值即可。对于后者，直接调用实验提供的 is-flow-finished 函数来检查最近一次更新的连接状态是否显示连接已经结束



具体代码如下所示：

```
void *nat_timeout()
{
    while (1) {
        //fprintf(stdout, "TODO: sweep finished flows periodically.\n");
        sleep(1);
        pthread_mutex_lock(&nat.lock);
        time_t now = time(NULL);

        for (int i = 0; i < HASH_8BITS; i++) {
            struct nat_mapping *map_entry = NULL, *map_q = NULL;
            list_for_each_entry_safe(map_entry, map_q, &(nat.nat_mapping_list[i]), list) {
                if ((now - map_entry->update_time > TCP_ESTABLISHED_TIMEOUT) || is_flow_finished(&(map_entry->conn))) {
                    log(DEBUG, "remove map entry, port: %d\n", map_entry->external_port);
                    nat.assigned_ports[map_entry->external_port] = 0;
                    list_delete_entry(&(map_entry->list));
                    free(map_entry);
                }
            }
        }

        pthread_mutex_unlock(&nat.lock);
    }

    return NULL;
}
```

## 五、实验结果与分析

### （一）SNAT 测试

执行 nat\_topo.py 脚本后，在 n1 节点上启动 nat 程序，读入配置文件 exp1.conf。然后在 h3 节点运行 http\_server.py 脚本，建立起服务后，并分别在 h1 和 h2 节点执行 wget http: //159.226.39.123: 8000 命令访问该服务。运行结果如下图所示：

```
"Node: h1"
root@Computer:~/workspace/Network/lab11/11-nat# wget http://159.226.39.123:8000
--2023-11-18 14:07:26-- http://159.226.39.123:8000/
正在连接 159.226.39.123:8000... 已连接。
已发出 HTTP 请求，正在等待回... 200 OK
长度：212 [text/html]
正在保存至: "index.html.7"

index.html.7      100%[=====>]      212  --.-KB/s   in 0s

2023-11-18 14:07:27 (63.5 MB/s) - 已保存 "index.html.7" [212/212])

root@Computer:~/workspace/Network/lab11/11-nat#
```

```
"Node: h2"
root@Computer:~/workspace/Network/lab11/11-nat# wget http://159.226.39.123:8000
--2023-11-18 14:08:39-- http://159.226.39.123:8000/
正在连接 159.226.39.123:8000... 已连接。
已发出 HTTP 请求，正在等待回... 200 OK
长度：212 [text/html]
正在保存至: "index.html.8"

index.html.8      100%[=====>]      212  --.-KB/s   in 0s

2023-11-18 14:08:39 (63.3 MB/s) - 已保存 "index.html.8" [212/212])
```

从上面可知，h1 和 h2 都可以正常访问 h3 的服务。通过 wireshark 查看节点 h2 和 h3 的收发包情况，可以看到 TCP 连接正常建立到关闭的过程，并且 NAT 完成了私有 IP 地址到公有 IP 地址的转换

正在捕获 h2-eth0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.21.0.2	159.226.39.123	TCP	74	41246 → 8000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK
2	0.000396222	36:6e:b3:6e:82:ce	Broadcast	ARP	42	Who has 10.21.0.2? Tell 10.21.0.254
3	0.000404421	ee:55:bf:e3:94:d0	36:6e:b3:6e:82:ce	ARP	42	10.21.0.2 is at ee:55:bf:e3:94:d0
4	0.000609766	159.226.39.123	10.21.0.2	TCP	74	8000 → 41246 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MS
5	0.000622443	10.21.0.2	159.226.39.123	TCP	66	41246 → 8000 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=5
6	0.000710405	10.21.0.2	159.226.39.123	HTTP	212	GET / HTTP/1.1
7	0.001152536	159.226.39.123	10.21.0.2	TCP	66	8000 → 41246 [ACK] Seq=1 Ack=147 Win=30208 Len=0 TSval
8	0.001158671	159.226.39.123	10.21.0.2	TCP	83	8000 → 41246 [PSH, ACK] Seq=1 Ack=147 Win=30208 Len=17
9	0.001160727	10.21.0.2	159.226.39.123	TCP	66	41246 → 8000 [ACK] Seq=147 Ack=18 Win=29696 Len=0 TSva
10	0.001169747	159.226.39.123	10.21.0.2	HTTP	414	HTTP/1.0 200 OK (text/html)
11	0.001399430	10.21.0.2	159.226.39.123	TCP	66	41246 → 8000 [FIN, ACK] Seq=147 Ack=367 Win=30720 Len=
12	0.001437360	159.226.39.123	10.21.0.2	TCP	66	8000 → 41246 [ACK] Seq=367 Ack=148 Win=30208 Len=0 TSv
13	5.016396959	ee:55:bf:e3:94:d0	36:6e:b3:6e:82:ce	ARP	42	Who has 10.21.0.254? Tell 10.21.0.2
14	5.016516188	36:6e:b3:6e:82:ce	ee:55:bf:e3:94:d0	ARP	42	10.21.0.254 is at 36:6e:b3:6e:82:ce

正在捕获 h3-eth0

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)

应用显示过滤器... <Ctrl-/> 表达式...

No.	Time	Source	Destination	Protocol	Length	Info
12	0.000722669	159.226.39.123	159.226.39.43	TCP	66	8000 → 12345 [ACK] Seq=367 Ack=148 Win=30208 Len=0 T
13	5.009364043	ca:c2:8d:ad:29:62	4e:0b:6a:1c:03:ec	ARP	42	Who has 159.226.39.43? Tell 159.226.39.123
14	5.009402722	4e:0b:6a:1c:03:ec	ca:c2:8d:ad:29:62	ARP	42	159.226.39.43 is at 4e:0b:6a:1c:03:ec
15	207.533290896	4e:0b:6a:1c:03:ec	Broadcast	ARP	42	Who has 159.226.39.123? Tell 159.226.39.43
16	207.533298161	ca:c2:8d:ad:29:62	4e:0b:6a:1c:03:ec	ARP	42	159.226.39.123 is at ca:c2:8d:ad:29:62
17	207.533308001	159.226.39.43	159.226.39.123	TCP	74	[TCP Port numbers reused] 12345 → 8000 [SYN] Seq=0 W
18	207.533317085	159.226.39.123	159.226.39.43	TCP	74	8000 → 12345 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 W
19	207.533773704	159.226.39.43	159.226.39.123	TCP	66	12345 → 8000 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval
20	207.533885431	159.226.39.43	159.226.39.123	HTTP	212	GET / HTTP/1.1
21	207.533891775	159.226.39.123	159.226.39.43	TCP	66	8000 → 12345 [ACK] Seq=1 Ack=147 Win=30208 Len=0 TSv
22	207.534065522	159.226.39.123	159.226.39.43	TCP	83	8000 → 12345 [PSH, ACK] Seq=1 Ack=147 Win=30208 Len=
23	207.534103789	159.226.39.123	159.226.39.43	HTTP	414	HTTP/1.0 200 OK (text/html)
24	207.534246828	159.226.39.43	159.226.39.123	TCP	66	12345 → 8000 [ACK] Seq=147 Ack=18 Win=29696 Len=0 TS
25	207.534496757	159.226.39.43	159.226.39.123	TCP	66	12345 → 8000 [FIN, ACK] Seq=147 Ack=367 Win=30720 Le
26	207.534502300	159.226.39.123	159.226.39.43	TCP	66	8000 → 12345 [ACK] Seq=367 Ack=148 Win=30208 Len=0 T
27	212.549381846	ca:c2:8d:ad:29:62	4e:0b:6a:1c:03:ec	ARP	42	Who has 159.226.39.43? Tell 159.226.39.123
28	212.549420156	4e:0b:6a:1c:03:ec	ca:c2:8d:ad:29:62	ARP	42	159.226.39.43 is at 4e:0b:6a:1c:03:ec

## (二) DNAT 测试

执行 nat\_topo.py 脚本后，在 n1 节点上启动 nat 程序，并读入配置文件 exp2.conf。

在 h1 和 h2 节点运行 http\_server.py 脚本，建立 http 服务后，从 h3 节点执行 wget 命令访问 h1 的服务，将端口号改为 8001 以访问 h2 的服务。结果如下图所示。可见连接正常建立和处理。。

```

"Node: h3"
root@Computer:~/workspace/Network/lab11/11-nat# wget http://159.226.39.43:8000
--2023-11-18 14:26:51-- http://159.226.39.43:8000/
正在连接 159.226.39.43:8000... 已连接。
已发出 HTTP 请求，正在等待回... 200 OK
长度：208 [text/html]
正在保存至：“index.html.11”

index.html.11 100%[=====>] 208 --.-KB/s in 0s

2023-11-18 14:26:51 (58.3 MB/s) - 已保存 “index.html.11” [208/208]

root@Computer:~/workspace/Network/lab11/11-nat# wget http://159.226.39.43:8001
--2023-11-18 14:28:08-- http://159.226.39.43:8001/
正在连接 159.226.39.43:8001... 已连接。
已发出 HTTP 请求，正在等待回... 200 OK
长度：208 [text/html]
正在保存至：“index.html.12”

index.html.12 100%[=====>] 208 --.-KB/s in 0s

2023-11-18 14:28:08 (69.1 MB/s) - 已保存 “index.html.12” [208/208]

```

通过 wireshark 查看节点 h2 和 h3 的收发包情况，可以看到 TCP 连接正常建立到关闭的过程，并且 NAT 完成了公有 IP 地址到私有 IP 地址的转换

正在捕获 h1-eth0

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)

应用显示过滤器... <Ctrl-/> 表达式... +

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0a:8b:17:3f:15:cf	Broadcast	ARP	42	Who has 10.21.0.1? Tell 10.21.0.254
2	0.000009511	72:25:15:f3:68:c9	0a:8b:17:3f:15:cf	ARP	42	10.21.0.1 is at 72:25:15:f3:68:c9
3	0.000223141	159.226.39.123	10.21.0.1	TCP	74	41430 → 8000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK
4	0.000234549	10.21.0.1	159.226.39.123	TCP	74	8000 → 41430 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MS
5	0.000351775	159.226.39.123	10.21.0.1	TCP	66	41430 → 8000 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=5
6	0.000534411	159.226.39.123	10.21.0.1	HTTP	211	GET / HTTP/1.1
7	0.000539337	10.21.0.1	159.226.39.123	TCP	66	8000 → 41430 [ACK] Seq=1 Ack=146 Win=30208 Len=0 TSval
8	0.000697029	10.21.0.1	159.226.39.123	TCP	83	8000 → 41430 [PSH, ACK] Seq=1 Ack=146 Win=30208 Len=17
9	0.000742270	10.21.0.1	159.226.39.123	HTTP	410	HTTP/1.0 200 OK (text/html)
10	0.000767859	159.226.39.123	10.21.0.1	TCP	66	41430 → 8000 [ACK] Seq=146 Ack=18 Win=29696 Len=0 TSva
11	0.001078314	159.226.39.123	10.21.0.1	TCP	66	41430 → 8000 [FIN, ACK] Seq=146 Ack=363 Win=30720 Len=
12	0.001083768	10.21.0.1	159.226.39.123	TCP	66	8000 → 41430 [ACK] Seq=363 Ack=147 Win=30208 Len=0 TSv
13	5.007836221	72:25:15:f3:68:c9	0a:8b:17:3f:15:cf	ARP	42	Who has 10.21.0.254? Tell 10.21.0.1
14	5.008147320	0a:8b:17:3f:15:cf	72:25:15:f3:68:c9	ARP	42	10.21.0.254 is at 0a:8b:17:3f:15:cf

正在捕获 h3-eth0

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)

应用显示过滤器... <Ctrl-/> 表达式... +

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	ee:37:c1:b0:7d:80	Broadcast	ARP	42	Who has 159.226.39.43? Tell 159.226.39.123
2	0.000110060	0e:32:d3:47:c0:0f	ee:37:c1:b0:7d:80	ARP	42	159.226.39.43 is at 0e:32:d3:47:c0:0f
3	0.000114007	159.226.39.123	159.226.39.43	TCP	74	41430 → 8000 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK
4	0.000619133	159.226.39.43	159.226.39.123	TCP	74	8000 → 41430 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MS
5	0.000631459	159.226.39.123	159.226.39.43	TCP	66	41430 → 8000 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=5
6	0.000732235	159.226.39.123	159.226.39.43	HTTP	211	GET / HTTP/1.1
7	0.000831120	159.226.39.43	159.226.39.123	TCP	66	8000 → 41430 [ACK] Seq=1 Ack=146 Win=30208 Len=0 TSval
8	0.001037983	159.226.39.43	159.226.39.123	TCP	83	8000 → 41430 [PSH, ACK] Seq=1 Ack=146 Win=30208 Len=17
9	0.001042870	159.226.39.123	159.226.39.43	TCP	66	41430 → 8000 [ACK] Seq=146 Ack=18 Win=29696 Len=0 TSva
10	0.001060868	159.226.39.43	159.226.39.123	HTTP	410	HTTP/1.0 200 OK (text/html)
11	0.001313239	159.226.39.123	159.226.39.43	TCP	66	41430 → 8000 [FIN, ACK] Seq=146 Ack=363 Win=30720 Len=
12	0.001379440	159.226.39.43	159.226.39.123	TCP	66	8000 → 41430 [ACK] Seq=363 Ack=147 Win=30208 Len=0 TSv

### (三) 双 NAT 拓扑测试

构造包含两个主机节点和两个 NAT 节点的网络“h1-n1-n2-h2”，并设置各端口的 IP 地址如下所示：

```
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 10.21.0.2/16')
h2.cmd('route add default gw 10.21.0.254')

n1.cmd('ifconfig n1-eth0 10.21.0.254/16')
n1.cmd('ifconfig n1-eth1 159.226.39.23/24')

n2.cmd('ifconfig n2-eth0 10.21.0.254/16')
n2.cmd('ifconfig n2-eth1 159.226.39.43/24')
```

编写两个 NAT 节点的配置文件，以 n2 为例，设置 eth0 为内部端口，eth1 为外部端口，并设置从 8001 号外部端口到 10.21.0.2: 8000 的内部节点(即 h2 的 eth0 及其默认端

口)的映射, n1 的配置也类似, 如下图所示:

```
internal-iface: n1-eth0
external-iface: n1-eth1

dnat-rules: 159.226.39.23:8002 -> 10.21.0.1:8000
```

```
internal-iface: n2-eth0
external-iface: n2-eth1

dnat-rules: 159.226.39.43:8001 -> 10.21.0.2:8000
```

执行 nat\_topo2.py 脚本后, 在 n1 和 n2 节点上启动 nat 程序, 并分别读入配置文件 exp3\_1.conf 和 exp3\_2.conf。在 h2 节点运行 http\_server.py 脚本, 建立 http 服务后, 从 h1 节点执行 wget http://159.226.39.43:8001 命令访问该服务。然后在 h1 节点运行 http\_server.py 脚本, 建立 http 服务后, 从 h2 节点执行 wget http://159.226.39.23:8002 命令访问该服务。运行如下图所示, 可以看到连接正常建立和处理:

```
"Node: h1"
root@Computer:~/workspace/Network/lab11/11-nat# wget http://159.226.39.43:8001
--2023-11-18 15:21:02-- http://159.226.39.43:8001/
正在连接 159.226.39.43:8001... 已连接。
已发出 HTTP 请求, 正在等待回... 200 OK
长度: 207 [text/html]
正在保存至: "index.html.13"

index.html.13      100%[=====>]      207 --.-KB/s   in 0s
2023-11-18 15:21:02 (62.1 MB/s) - 已保存 "index.html.13" [207/207]
root@Computer:~/workspace/Network/lab11/11-nat#
```

```
"Node: h2"
root@Computer:~/workspace/Network/lab11/11-nat# wget http://159.226.39.23:8002
--2023-11-18 15:21:55-- http://159.226.39.23:8002/
正在连接 159.226.39.23:8002... 已连接。
已发出 HTTP 请求, 正在等待回... 200 OK
长度: 207 [text/html]
正在保存至: "index.html.14"

index.html.14      100%[=====>]      207 --.-KB/s   in 0s
2023-11-18 15:21:55 (67.6 MB/s) - 已保存 "index.html.14" [207/207]
```

## 六、 实验总结

通过本次实验,我了解了 NAT 地址转换的原理,掌握了 SNAT 和 DNAT 的具体实现方法,这  
让我对理论课上讲过的 NAT 地址转换有了更深的理解。