实验报告

网络地址转换(NAT)实验

一、实验内容

了解 NAT 地址转换原理，实现 NAT 设备。通过实验验证 SNAT、DNAT、多 NAT 功能正确性。最后，调研 NAT 如何支持 ICMP 协议。

二、实验流程

1. 了解 NAT 工作机制，实现 NAT 地址转换。
2. 完成三个实验，验证 SNAT、DNAT、多 NAT 功能正确性。
3. 调研 NAT 如何支持 ICMP 协议。
4. 实现通过公网服务器中转不同内网主机建立 TCP 连接。

三、实验结果及分析

（一）NAT 实现思路

1、处理配置信息

```
int parse_config(const char *filename)
```

函数，从 filename 文件中提取配置信息。

主要是字符串匹配，调用库函数完成。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。

2、处理 NAT 地址转换

```
void nat_translate_packet(iface_info_t *iface, char *packet, int len)
```

函数，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。

`static int get_packet_direction(char *packet)`

函数，负责判断数据包方向。

通过查询路由表，得到转出端口，如果转出端口是 NAT 指定的内部端口，那么其 IP 地址为内部地址，否则为外部地址。根据源 IP 地址和目的 IP 地址的类型判断数据包方向。

```
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);

int src_is_internal = (src_entry->iface == nat.internal_iface);
int dst_is_internal = (dst_entry->iface == nat.internal_iface);
int dst_is_external = (daddr == nat.external_iface->ip);

if (src_is_internal && !dst_is_internal) {
    return DIR_OUT;
}

if (!src_is_internal && dst_is_external) {
    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，看是否已经建立连接。

如果没有建立连接，根据数据包方向看是否能新建立连接。

如果无法新建连接，丢弃数据包。

```
struct nat_mapping *map_entry = nat_table_lookup(ip, tcp, dir);

if (map_entry == NULL) {
    if (dir == DIR_OUT && tcp->flags == TCP_SYN) {
        u16 ext_port = assign_external_port();
        map_entry = new_map_entry(ntohl(ip->daddr), ntohs(tcp->dport), ntohl(ip->saddr),
        ntohs(tcp->sport), nat.external_iface->ip, ext_port);
    }
    if (dir == DIR_IN && tcp->flags == TCP_SYN) {
        struct dnat_rule *rule_entry = NULL;
        list_for_each_entry(rule_entry, &nat.rules, list) {
            if (rule_entry->external_ip == ntohl(ip->daddr) && rule_entry->external_port == ntohs(tcp->dport)) {
                map_entry = new_map_entry(ntohl(ip->saddr), ntohs(tcp->sport), rule_entry->internal_ip,
                rule_entry->internal_port, rule_entry->external_ip, rule_entry->external_port);
            }
        }
    }
}
```

查找到或新建连接后，根据数据包方向进行地址转换。

以 IN 方向为例，首先根据 TCP 报头内容，更新连接控制数据结构、最近连接时间。

然后将目的 IP 地址、目的端口替换为内网对应主机的 IP 地址、端口，重新计算 TCP、IP 的校验和。

最后查找路由表，通过 ARP 协议发送到内网的下一跳节点或直接交付。

```
if (dir == DIR_IN) {
    log(DEBUG, "handle in tcp packet\n");
    int clear = (tcp->flags & TCP_RST) ? 1 : 0;
    map_entry->conn.external_fin = (tcp->flags & TCP_FIN) ? 1 : 0;
    map_entry->conn.external_seq_end = tcp_seq_end(ip, tcp);
    map_entry->conn.external_ack = ntohl(tcp->ack);
    map_entry->update_time = time(NULL);

    tcp->dport = htons(map_entry->internal_port);
    ip->daddr = htonl(map_entry->internal_ip);
    tcp->checksum = tcp_checksum(ip, tcp);
    ip->checksum = ip_checksum(ip);

    rt_entry_t *rt_dest = longest_prefix_match(map_entry->internal_ip);
    if (!rt_dest) {
        log(ERROR, "can not find the route to dest ip\n");
        free(packet);
        pthread_mutex_unlock(&nat.lock);
        return;
    }

    if (rt_dest->gw == 0) {
        iface_send_packet_by_arp(nat.internal_iface, map_entry->internal_ip, packet, len);
    }
    else {
        iface_send_packet_by_arp(nat.internal_iface, rt_dest->gw, packet, len);
    }

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

注意一个直接断开连接的条件，当一方发送 RST 包时，可以直接断开连接，释放资源。

```
void *nat_timeout(void *arg)
```

线程函数，负责检查连接存活情况，将超时没有传输数据、已经握手完毕断开连接的条目删除。

设定为每秒执行一次，将映射关系表中的在 60s 时间内未更新，或者已经完成四次挥手断开连接的条目删除，释放端口资源。

（二）实验验证 NAT 功能

1、SNAT 实验

在 n1 设备上配置 SNAT 配置信息。

在外部节点 h3 上执行服务器程序。

在内部节点 h1、h2 分别请求 h3 的页面。

h1 节点请求结果：

"Node: h1"

```
root@joker-linux:/mnt/hgfs/share/12-nat# wget http://159.226.39.123:8000
--2021-05-27 18:04:01--  http://159.226.39.123:8000/
正在连接 159.226.39.123:8000... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度:  212 [text/html]
正在保存至: “index.html.4”

index.html.4        100%[===================>]     212  --.-KB/s    用时 0s

2021-05-27 18:04:01 (18.6 MB/s) - 已保存 “index.html.4” [212/212])
```

h2 节点请求结果：

"Node: h2"

```
root@joker-linux:/mnt/hgfs/share/12-nat# wget http://159.226.39.123:8000
--2021-05-27 18:04:07--  http://159.226.39.123:8000/
正在连接 159.226.39.123:8000... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度:  212 [text/html]
正在保存至: “index.html.5”

index.html.5        100%[===================>]     212  --.-KB/s    用时 0s

2021-05-27 18:04:07 (21.0 MB/s) - 已保存 “index.html.5” [212/212])
```

可以看到两者都成功得到 h3 的页面。

得到的页面信息为：

```
root@joker-linux:/mnt/hgfs/share/12-nat# cat 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>
```

可以看到其源 IP 地址为 h3 地址，而目的 IP 地址为 NAT 设备的 IP 地址。

由上可知，SNAT 功能正确。

2、DNAT 实验

在 n1 设备上配置 DNAT 配置信息。

在内部节点 h1、h2 上执行服务器程序。

在外部节点 h3 上请求 NAT 设备两个端口的页面。

请求页面结果如下，都请求成功。

"Node: h3"

```
root@joker-linux:/mnt/hgfs/share/12-nat# wget http://159.226.39.43:8000
--2021-05-27 18:16:27--  http://159.226.39.43:8000/
正在连接 159.226.39.43:8000... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度:  208 [text/html]
正在保存至: “index.html”

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

2021-05-27 18:16:27 (25.4 MB/s) - 已保存 “index.html” [208/208])

root@joker-linux:/mnt/hgfs/share/12-nat# wget http://159.226.39.43:8001
--2021-05-27 18:16:30--  http://159.226.39.43:8001/
正在连接 159.226.39.43:8001... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度:  208 [text/html]
正在保存至: “index.html.1”

index.html.1        100%[===================>]     208  --.-KB/s    用时 0s

2021-05-27 18:16:30 (22.5 MB/s) - 已保存 “index.html.1” [208/208])
```

得到的页面信息为：

```
root@joker-linux:/mnt/hgfs/share/12-nat# cat index.html

<!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>
```

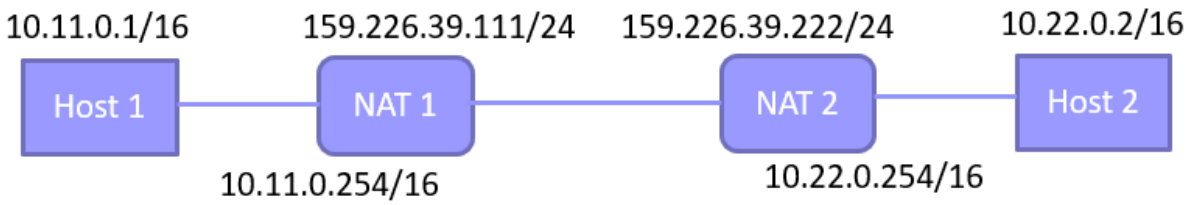
```
root@joker-linux:/mnt/hgfs/share/12-nat# cat index.html.1

<!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>
```

可以看到其源 IP 地址为内部节点私有 IP 地址，目的地址为 h3 公网地址。

由上可知，DNAT 功能正确。

3、多 NAT 实验

本实验网络拓扑结构为：



其中 n1 作为 SNAT，n2 作为 DNAT，主机 h2 执行服务器程序，主机 h1 请求 n2 NAT 设备的端口。

n1 建立 SNAT 连接：

"Node: n1"

```
root@joker-linux:/mnt/hgfs/share/12-nat# ./nat my_exp1.conf
DEBUG: find the following interfaces:  n1-eth0 n1-eth1.
Routing table of 2 entries has been loaded.
DEBUG: internal_iface: 10.11.0.254

DEBUG: external_iface: 159.226.39.111

DEBUG: snat new mapping: 10.11.0.1 43392 -> 159.226.39.111 12345
```

n2 建立 DNAT 连接：

"Node: n2"

```
root@joker-linux:/mnt/hgfs/share/12-nat# ./nat my_exp2.conf
DEBUG: find the following interfaces:  n2-eth0 n2-eth1.
Routing table of 2 entries has been loaded.
DEBUG: internal_iface: 10.22.0.254

DEBUG: external_iface: 159.226.39.222

DEBUG: dnat_rule: 159.226.39.222 8000 10.22.0.2 8000

DEBUG: dnat new mapping: 10.22.0.2 8000 -> 159.226.39.222 8000
```

h1 请求页面结果为：

"Node: h1"

```
root@joker-linux:/mnt/hgfs/share/12-nat# wget http://159.226.39.222:8000
--2021-05-27 19:45:08--  http://159.226.39.222:8000/
正在连接 159.226.39.222:8000... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度:  208 [text/html]
正在保存至: “index.html”

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

2021-05-27 19:45:08 (18.4 MB/s) - 已保存 “index.html” [208/208])

root@joker-linux:/mnt/hgfs/share/12-nat# cat index.html

<!doctype html>
<html>
        <head> <meta charset="utf-8">
                <title>Network IP Address</title>
        </head>
        <body>
            My IP is: 10.22.0.2
            Remote IP is: 159.226.39.111
        </body>
</html>
```

可以看到 h1 请求页面成功，得到的页面显示源 IP 地址为 h2 内部 IP 地址，目的 IP 为 h1 的 NAT 设备地址。

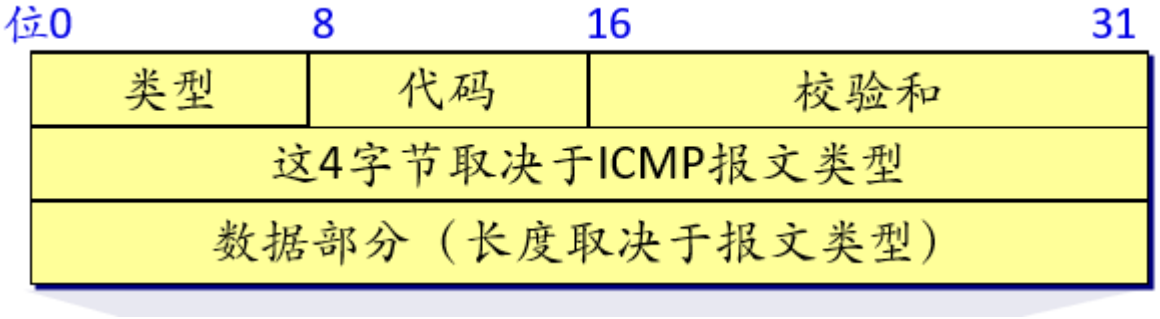
由上可知，实验三各 NAT 设备功能正确。

四、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。

TCP、UDP 协议都有端口号，我们都可以通过<IP, 端口号>来建立唯一映射。但是 ICMP 协议的报文如下：显

然它是没有端口号的，那 ICMP 如何进行地址转换。



首先 ICMP 报文可以分为两类，询问报文、差错报告报文。

（1）对于询问报文，其头部的最后 4 字节，前 2 字节为 identifier，后 2 字节为 sequence。

Identifier 为主机标识，每台主机有唯一固定的标识，且各主机的标识不同。Sequence 为序列号，每个 ICMP 包都有不同的序列号。

因此可以使用 Identifier 或者 Identifier+Sequence 作为标识符，代替端口号。使用<IP, Identifier>来建立唯一映射。

当内部主机发出 ICMP 询问报文时，NAT 以其 Identifier 作为标识，并分配 NAT 的一个端口与之建立映射。

当外部回信时，根据 ICMP 协议规则，Identifier 与 Sequence 字段原封不动保留。NAT 收到回复报文时，根据 Identifier 找到 NAT 的对应端口，然后查找到映射关系，将目的 IP 地址转换后发给内网主机。

（2）对于差错报告报文问题就没那么简单了。

我们先回顾一下 ICMP 差错报告报文的产生情况，比如内部主机向外部发送一个 PING 报文，NAT 根据 Identifier 建立了映射，到这都没有问题。但是报文从 NAT 发出后，在某一级路由器发生了差错，这时该路由器向回发送 ICMP 差错报文，根据报文格式它将 ICMP 头部的 Identifier 与 Sequence 全部抹为 0。当差错报文回到 NAT 时，它发现 Identifier 字段全 0，无法根据 Identifier 查找映射表。

那么此时应该如何处理呢，一种方案是 NAT 此时需要检查 ICMP 报文数据段。根据 ICMP 协议规则，当差错报文产生时，出错 IP 数据包的 IP 头部以及接下来的 8 字节会填入数据部分。

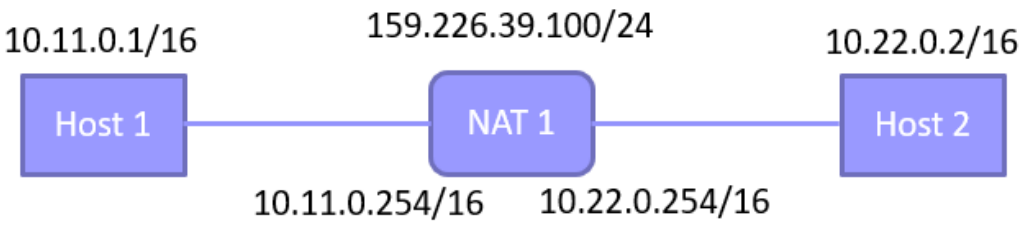
如果出错数据包是 ICMP 询问报文，IP 头部接下来的 8 字节正好是其头部，可以从其中找到原本的 Identifier 与 Sequence，根据该字段查找映射表就能像上面那样找到内网主机。

如果出错数据包是 TCP、UDP 数据包，那么可以从那 8 字节中提取到 NAT 发出时的源端口，根据端口号查找映射表找到内网主机。

综上，通过利用头部的 Identifier 与 Sequence，以及从数据部分提取原信息，就可以完成对 ICMP 协议的地址转换支持。

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。（最好有概念验证代码）

（1）理论设计



网络拓扑结构如上，服务器连接 2 个不同内网主机。显然不同内网的主机不能直接相互通信，我们希望借助服务器的公网进行中转。

基本思路为，把对一个内网主机的访问映射到对服务器<公网 IP, 端口>的访问，由服务器完成公网到内网地址的转换。此时的一个问题是，没有 NAT 那样直观上的内部和外部网络，其连接的每个主机都是内网。此时一个连接映射为：<内网主机 IP1，端口 1> ---- <服务器公网 IP，端口 x> ---- <另一内网主机 IP2，端口 2>。在 NAT 中我们采用外网（远端）IP 端口作为连接的不变量，但此时连接的两方都是内网，没有可用的不变量。在多个主机同时访问时，可能出现这样的连接，<内网主机 IP1，端口 1> ---- <服务器公网 IP，端口 x> ---- <另一内网主机 IP3，端口 3>，那从主机 1 发送数据包到服务器时，服务器就不知道该转发给主机 2 还是主机 3。因此这样 3 元组的连接映射方式在这里就行不通了。

那么该如何实现连接的映射呢？我们先考虑面向主机和端口的映射，即让每个内网主机在服务器上以唯一<服务器公网 IP，端口>来注册。当另一内网主机访问服务器该端口，就将其数据包转发给对应内网主机。

服务器需要同时执行 SNAT 和 DNAT 的工作，其基本流程为：

收到某一内网主机 1 的访问，将其目标地址 <服务器公网 IP，端口 x> 映射到对应的另一内网主机 2，更改目标 IP 地址、端口。如果没有查找到映射关系，查看 dnat-rules，尝试新建映射关系。

然后根据源地址 <IP1，端口 1> 查找主机 1 在服务器上的映射，如果没有映射就给它分配一个端口 y。用<服务器公网 IP，端口 y> 替换源 IP 地址、端口，然后转发给目的主机 2。

那么如何描述一个 TCP 连接呢，使用主机 1 到服务器端口 x 映射和主机 2 到服务器端口 y 映射，这两个映射的组合即可完整描述一个 TCP 连接。

综上，公网服务器需要的功能为：不再区分内部外部网络，同时执行 SNAT 和 DNAT 功能，使用内网主机 IP 端口到服务器端口的映射对建立 TCP 连接，转发数据包时同时更改源、目的 IP 地址和端口。

（2）具体实现

下面来具体实现这样一个服务器。实现的代码位于 12-nat-Server 文件夹下。

首先是 NAT 数据结构，此时连接的都是内网，没必要区分内网外网接口，去掉这两个字段。另外，增加一个服务器的公网 IP 字段。

在配置文件中也不必再提供 internal iface 和 external iface，不过 dnat-rules 仍需要提供，现在所有主机都是内网主机，必须提供 dnat 配置才能建立连接。

```
struct nat_table {
    struct list_head nat_mapping_list[HASH_8BITS];

    u8 assigned_ports[65536];

    u32 public_ip;

    struct list_head rules;

    pthread_mutex_t lock;
    pthread_t thread;
};
```

TCP 连接映射数据结构，现在是使用两个主机的 <主机 IP，端口> 到 <服务器公网 IP，端口> 的映射对来描述一个 TCP 连接，因此远端（remote）项就不需要了，替换为 2 个 internal 和 2 个 external。2 个 internal 表示 TCP 连接的两个内网主机，2 个 external 是它们在服务器上对应的端口。

```
struct nat_mapping {
    struct list_head list;

    u32 internal_src_ip;        // ip address of source host in private network
    u16 internal_src_port;      // port of source host in private network
    u32 external_src_ip;        // ip address of source host in public network
    u16 external_src_port;      // port of source host in public network

    u32 external_dst_ip;        // ip address of destination host in public network
    u16 external_dst_port;      // port of destination host in public network
    u32 internal_dst_ip;        // ip address of destination host in private network
    u16 internal_dst_port;      // port of destination host in private network

    time_t update_time;         // when receiving the latest packet
    struct nat_connection conn; // statistics of the tcp connection
};
```

现在没有直观上的数据包方向，但我们仍然可以定义方向。我们规定最先发起 TCP 连接的一方为源主机，目的方为目的主机，从源主机发往目的主机的数据包方向为 IN，反之为 OUT。

现在由于没有内外网络之分，判断数据包方向的工作无法在处理地址转换的一开始就完成，将其放到与查找映射表一起完成。

服务器需要同时处理 SNAT 与 DNAT 的工作，同时对源、目的地址进行转换。

内网主机发出的数据包，其目的地址、端口为服务器的公网地址、端口，要将其转换到对应的内网主机。首先查找映射表，看是否已经有连接。如果目的 IP 端口与 nat_mapping 中 external_dst 的 IP 端口匹配，且源 IP 端口与 nat_mapping 中 internal_src 的 IP 端口匹配，说明已经有连接，且方向是 IN。而如果目的 IP 端口与 nat_mapping 中 external_src 的 IP 端口匹配，且源 IP 端口与 nat_mapping 中 internal_dst 的 IP 端口匹配，说明已经有连接，且方向是 OUT。

```
u8 index = hash_nat(dst_ip, dst_port);

list_for_each_entry(map_entry, &(nat.nat_mapping_list[index]), list) {
    if (map_entry->external_dst_ip == dst_ip && map_entry->external_dst_port == dst_port &&
    map_entry->internal_src_ip == src_ip && map_entry->internal_src_port == src_port) {
        dir = DIR_IN;
        find = 1;
        break;
    }
}

if (find == 0) {
    for (int i=0; i<HASH_8BITS; i++) {
        list_for_each_entry(map_entry, &(nat.nat_mapping_list[i]), list) {
            if (map_entry->external_src_ip == dst_ip && map_entry->external_src_port == dst_port &&
            map_entry->internal_dst_ip == src_ip && map_entry->internal_dst_port == src_port) {
                dir = DIR_OUT;
                find = 1;
                break;
            }
        }
        if (find) break;
    }
}
```

如果查找不到连接，查看 dnat-rule，看看能否新建一个连接。如果无法新建，返回 ICMP 报文。在新建连接时，我们还需要给源主机分配一个服务器端口，方式与 SNAT 中相似。

```
if (find == 0) {
    if (tcp->flags == TCP_SYN) {
        struct dnat_rule *rule_entry = NULL;
        list_for_each_entry(rule_entry, &nat.rules, list) {
            if (rule_entry->external_ip == dst_ip && rule_entry->external_port == dst_port) {
                u16 ext_port = assign_external_port();
                map_entry = new_map_entry(src_ip, src_port, ext_port, rule_entry->external_ip,
                rule_entry->external_port, rule_entry->internal_ip, rule_entry->internal_port);

                dir = DIR_IN;
                find = 1;
                break;
            }
        }
    }
}

if (find == 0) {
    log(ERROR, "can not find or build mapping\n");
    icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_HOST_UNREACH);
    free(packet);
    pthread_mutex_unlock(&nat.lock);
    return;
}
```

当找到或者建立连接后，就进行地址转换。根据数据包方向，将目的地址端口替换为目的内网主机地址端口；将源地址端口替换为服务器与之对应的公网地址端口。

以方向为 IN 为例，作如下地址转换：

```
if (dir == DIR_IN) {
    log(DEBUG, "handle in tcp packet\n");
    int clear = (tcp->flags & TCP_RST) ? 1 : 0;
    map_entry->conn.external_fin = (tcp->flags & TCP_FIN) ? 1 : 0;
    map_entry->conn.external_seq_end = tcp_seq_end(ip, tcp);
    map_entry->conn.external_ack = ntohl(tcp->ack);
    map_entry->update_time = time(NULL);

    tcp->dport = htons(map_entry->internal_dst_port);
    tcp->sport = htons(map_entry->external_src_port);
    ip->daddr = htonl(map_entry->internal_dst_ip);
    ip->saddr = htonl(map_entry->external_src_ip);

    tcp->checksum = tcp_checksum(ip, tcp);
    ip->checksum = ip_checksum(ip);
```

之后借助路由表和 ARP 将数据包转发出去即可。

（3）实验结果

在上述实现下，由 h1 主机作为 TCP 服务方，h2 主机向其发起页面请求，在公网服务器 n1 上配置 h1 的 dnat-rules。结果如下：

"Node: h2"

```
root@joker-linux:/mnt/hgfs/share/12-nat-Server# wget http://159.226.39.100:8000
--2021-05-28 21:27:58--  http://159.226.39.100:8000/
正在连接 159.226.39.100:8000... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度:  208 [text/html]
正在保存至: “index.html”

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

2021-05-28 21:27:58 (19.1 MB/s) - 已保存 “index.html” [208/208])

root@joker-linux:/mnt/hgfs/share/12-nat-Server# cat index.html

<!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: 159.226.39.100
        </body>
</html>
```

可以看到 h1、h2 主机成功建立了 TCP 连接，并传输数据。发送的文件显示，源 IP 为主机 h1 内网 IP，目的地址为服务器公网 IP。端口在文件中没有体现，但我们查看 n1 的信息可以看到，h2 主机被映射到 12345 端口，h1 映射到 dnat-rule 中预先设定的 8000 端口。

由上可知，成功实现了利用一个公网服务器转发，在两个不同内网的主机间建立 TCP 连接进行数据传输。

五、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。