

网络地址转换实验报告

李昊宸

2017K8009929044

(一) nat 机制实现

一、实验内容

1. SNAT 源地址转换实验:

1) 运行网络拓扑(nat_topo.py)

2) 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

3) 在 n1 上运行 nat 程序

4) 在 h3 上运行 HTTP 服务

5) 在 h1, h2 上分别访问 h3 的 HTTP 服务

2. DNAT 目的地址转换实验:

1) 运行网络拓扑(topo.py)

2) 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

3) 在 n1 上运行 nat 程序

4) 在 h1, h2 上分别运行 HTTP Server

5) 在 h3 上分别请求 h1, h2 页面

3. 两个 nat 穿越实验:

1) 手动构造一个包含两个 nat 的拓扑: h1 <-> n1 <-> n2 <-> h2

2) 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

二、实验流程

1. 搭建实验环境

include: 相关头文件

scripts: 禁止协议栈的数据包处理

main.c: NAT 的代码实现, 编译后在路由器结点上运行

ip.c: 处理 IP 数据包, 包括转发

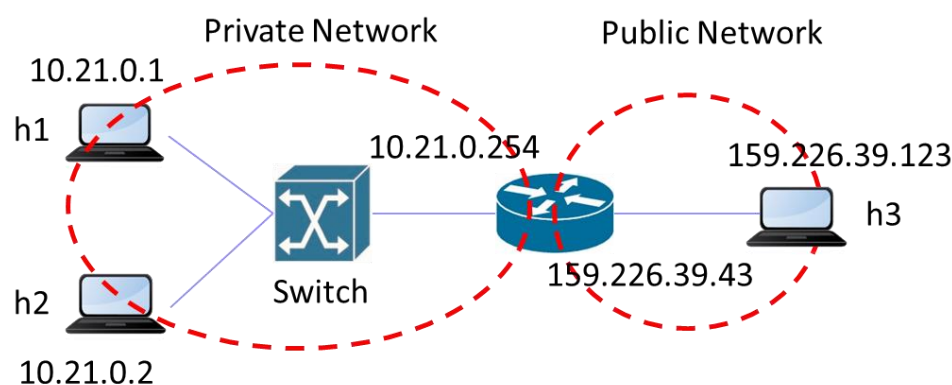
libipstack.a: 路由实验实现的 ARP 和路由表相关函数编译出的库

http_server.py: 简单 HTTP Server 实现

exp[1-41].conf: NAT 配置文件

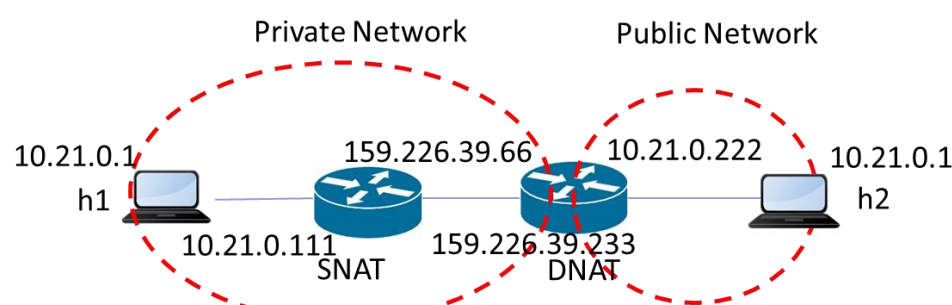
nat.c: NAT 相关代码实现

nat_topo.py: 实现如下图的公-私网络拓扑



图一 公-私网络拓扑

exp3_nat-topo.py: 实现如下图的私-公-私网络拓扑



图二 私-公-私网络拓扑

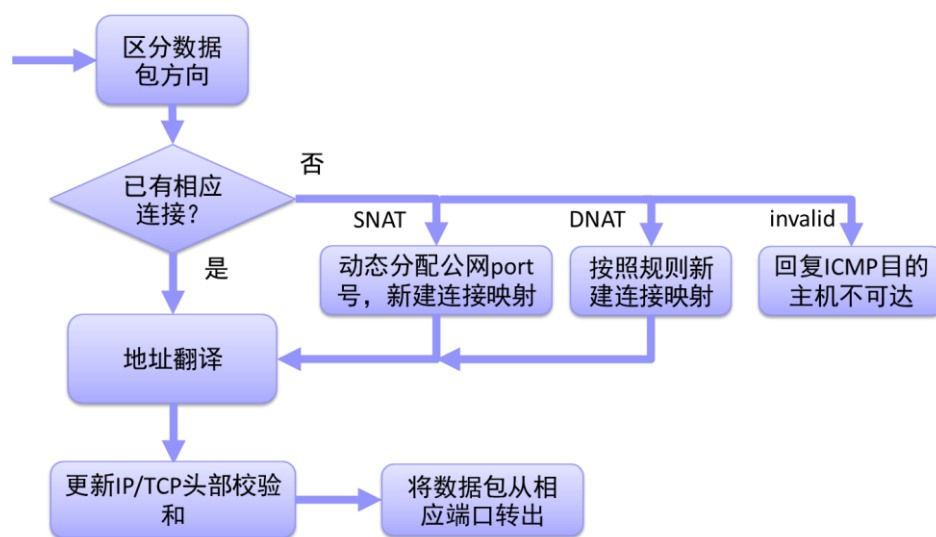
NAT: Network Address Translation, 网络地址转换, 也叫做网络掩蔽或者 IP 掩蔽 (IP masquerading), 是一种在 IP 数据包通过路由器或防火墙时重写来源 IP 地址或目的 IP 地址的技术, 被普遍使用在有多台主机但只通过一个公有 IP 地址访问因特网的私有网络中。但是, NAT 也让主机之间的通信变得复杂, 导致了通信效率的降低。

NAT 设备主要工作: 1) 维护私网地址/端口 与 公网地址/端口的映射关系 2) 对数据包内容进行重写 (Translation), 修改 IP 地址、端口等字段, 使得数据包在相应网络中有意义。

SNAT: Source Network Address Translation, 源地址转换, 将 IP 数据包的源地址转换成另外一个地址。内网主机 A 和外网主机 B 通信, A 可以向 B 发送 IP 数据包, 但 B 向 A 回复时, 如果 A 使用的是内网地址, 那么数据包无法到达, 此时需要 SNAT 设备记录内网主机的 A 的 IP, 与之建立并记录连接, 更换 A 发来的数据包 IP 为路由器公网 IP, 为该连接保存一个端口。

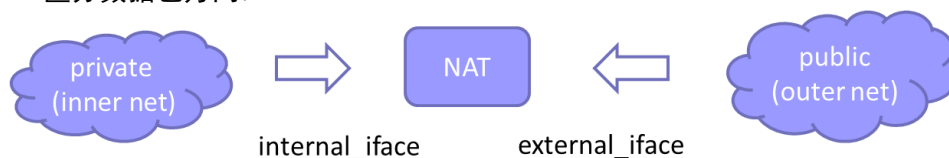
DNAT: Destination Network Address Translation, 目的地址转换, 将一组本地内部的地址映射到一组全球地址。继续上面的例子, 此后主机 B 与主机 A 交互就使用该 IP 和端口, DNAT 设备收到来自主机 B 的数据包后解析, 如果目的地址为路由器公网 IP, 端口为记录的端口, 就将该数据包转发至 A。

NAT 工作机制:



图三 NAT 工作机制

区分数据包方向:



图四 NAT 工作示例

NAT 设备收到一个数据包, 进行数据包方向的区分:

1. 源地址为内部地址, 且目的地址为外部地址时, 方向为 DIR_OUT
2. 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

可以通过查询源地址和目的地址的转发表项, 将转发表项的端口取出, 如果源地址侧端口是内网端口, 目的地址侧是外网端口, 方向为 DIR_OUT; 如果源地址侧是外网端口, 目的地址侧是内网端口, 方向为 DIR_IN。在该实验中, 因为不涉及其他的路由, 不满足这两项的 IP 数据包设置为 invalid 即可。

建立连接:

如果连接已经建立，跳过此步。

如果连接没有建立，进行如下操作：

SNAT 的场合：如果该数据包的方向为 DIR_OUT，说明该包为内网主机到外网主机 TCP 连接的第一个数据包（请求连接数据包），在 NAT 缓存表中加入该表项，记录内网主机的内网 IP、内网端口号和映射的外网 IP、随机分配的外网端口号（不能为 0）。

DNAT 的场合：如果该数据包的方向为 DIR_IN，说明该包为外网主机到内网主机 TCP 连接的第一个数据包（请求连接数据包），查看 DNAT 是否有记录处理来自该外网主机的规则，如果有，将该规则添加到 NAT 缓存表中。

INVALID 的场合：回复 ICMP Destination Host Unreachable。

NAT 地址翻译：对于已经存在的表项，进行(internal_ip, internal_port) <-> (external_ip, external_port)之间的转换

SNAT：修改 IP 首部的源主机地址为表项中记录的外网 IP，TCP 首部的源主机端口为表项中记录的外网端口号，按 TCP 内容修改表项是否有效（如果 TCP 连接结束就将表项设置为无效）和内网侧最新序列号和内网侧最新确认号，更新使用时间，更新 IP 和 TCP 的校验和。最后将数据包从表项中记录的端口转发出去。

DNAT：修改 IP 首部的目的主机地址为表项中记录的内网 IP，TCP 首部的目的主机端口为表项中记录的内网端口号，按 TCP 内容修改表项是否有效（如果 TCP 连接结束就将表项设置为无效）和外网侧最新序列号和外网侧最新确认号，更新使用时间，更新 IP 和 TCP 的校验和。最后将数据包从表项中记录的端口转发出去。

连接的维护：可以考虑使用 Hash 表的方式进行存储。对外网主机 IP 进行哈希，相同哈希值的外网主机与内网主机的映射关系保存在同一个表单下，这样一来，每当有新的数据包要翻译地址，第一层查找只需要直接拉取对应哈希值的表单，在该表单下的链表进行第二层查找即可。

连接的老化:

1. TCP 连接结束时的四次握手：双方都发送 FIN 并回复相应 ACK，说明该连接已经结束，可以将该表项删除。
2. 一方发送了 RST 包，说明该连接意外终止，可以将该表项删除。
3. 双方超过 60 秒没有进行过数据传输，认为传输已经结束，可以将该表项删除。

2. 启动脚本

1) SNAT 实验

```
make all
sudo python nat_topo.py
mininet> xterm h1 h2 h3 n1
n1# ./nat_exp1.conf
h3# python ./http_server.py
h1# wget http://159.226.39.123:8000
h1# cat index.html
h2# wget http://159.226.39.123:8000
h2# cat index.html.1
mininet> quit
```

2) DNAT 实验

```
make all
sudo python nat_topo.py
mininet> xterm h1 h2 h3 n1
n1# ./nat_exp1.conf
h1# python ./http_server.py
h2# python ./http_server.py
h3# wget http://159.226.39.43:8000
h3# cat index.html.2
h3# wget http://159.226.39.43:8001
h3# cat index.html.3
mininet> quit
```

3) 双 nat 穿透实验

```
make all
sudo python exp3_nat-topo.py
mininet> xterm h1 h2 h3 n1
n1# ./nat_exp1.conf
h1# python ./http_server.py
h2# python ./http_server.py
h3# wget http://159.226.39.43:8000
h3# cat index.html.2
h3# wget http://159.226.39.43:8001
h3# cat index.html.3
mininet> quit
```

三、实验结果及分析

1. 实验结果

```
root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# python ./http_server.py
Serving HTTP on 0.0.0.0 port 8000 ...
159.226.39.43 - - [04/Jun/2020 04:00:54] "GET / HTTP/1.1" 200 -
159.226.39.43 - - [04/Jun/2020 04:01:10] "GET / HTTP/1.1" 200 -

root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# wget http://159.226.39.123:8000/
--2020-06-04 04:00:54-- 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"

index.html 100%[=====] 212 --.-KB/s in 0s
2020-06-04 04:00:54 (17.7 MB/s) - 已保存 "index.html" [212/212]

root@CN-VirtualBox:/mnt/shared/10-nat/10-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>

root@CN-VirtualBox:/mnt/shared/10-nat/10-nat#

gcc -c -g -Wall -Wno-unused-variable -Wno-...
.o
gcc -L. ip.o main.o nat.o -o nat -lipstack
moto@CN-VirtualBox:/mnt/shared/10-nat/10-nat#
mininet> xterm h1 h2 h3 n1
mininet> quit
moto@CN-VirtualBox:/mnt/shared/10-nat/10-nat#
mininet> xterm h1 h2 h3 n1
mininet>
```

图五 SNAT 实验结果

可以看到，h1 作为服务器的 IP 为公网 159.226.39.123，h2 和 h3 处于同一内网，共用同一个公网 IP 159.226.39.43

```
root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# wget http://159.226.39.43:8000
--2020-06-04 03:50:38-- 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.15"

index.html.15 100%[=====] 208 --.-KB/s in 0s
2020-06-04 03:50:38 (22.9 MB/s) - 已保存 "index.html.15" [208/208]

root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# cat index.html.15
<!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@CN-VirtualBox:/mnt/shared/10-nat/10-nat# wget http://159.226.39.43:8001
--2020-06-04 03:50:54-- 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.16"

index.html.16 100%[=====] 208 --.-KB/s in 0s
2020-06-04 03:50:54 (18.6 MB/s) - 已保存 "index.html.16" [208/208]

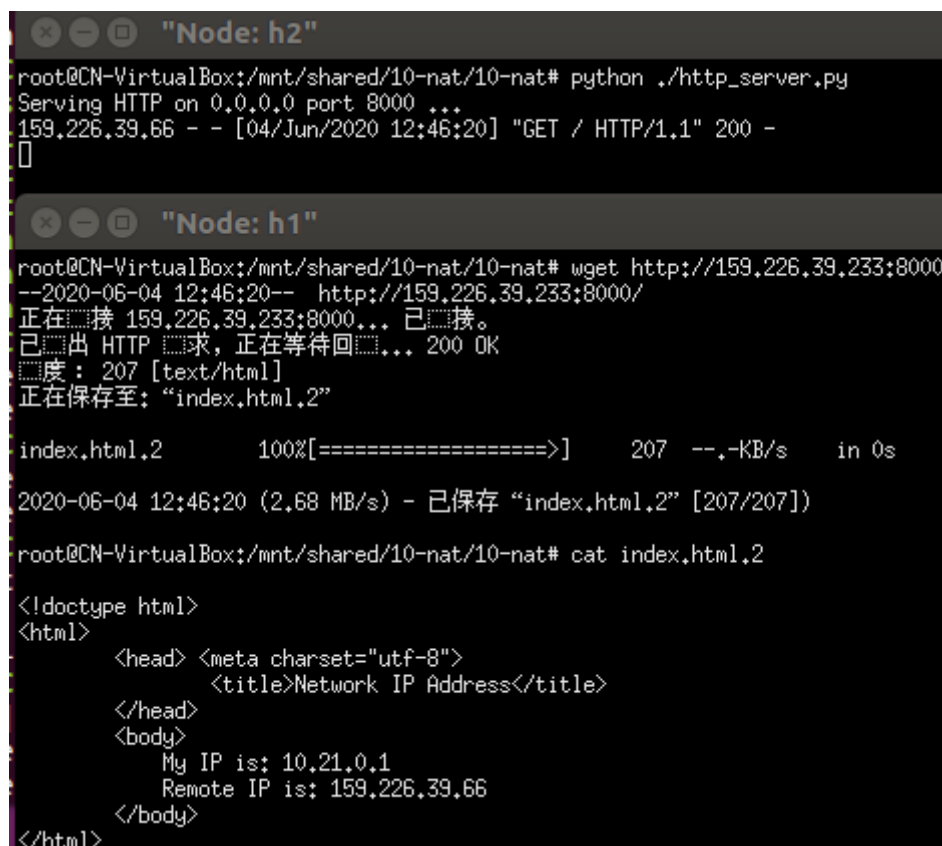
root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# cat index.html.16
<!doctype html>
<html>
  <head> <meta charset="utf-8">
    <title>Network IP Address</title>
  </head>
  <body>
    My IP is: 10.21.0.2
    Remote IP is: 159.226.39.123
  </body>
</html>

root@CN-VirtualBox:/mnt/shared/10-nat/10-nat#

root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# python ./http_server.py
Serving HTTP on 0.0.0.0 port 8000 ...
159.226.39.123 - - [04/Jun/2020 03:50:38] "GET / HTTP/1.1" 200 -
159.226.39.123 - - [04/Jun/2020 03:50:54] "GET / HTTP/1.1" 200 -
```

图六 DNAT 实验结果

可以看到，作为服务器的 h1 的内网 IP 为 10.21.0.1，作为服务器的 h2 的内网 IP 为 10.21.0.2，作为客户的 h3 的公网 IP 为 159.226.39.123



```
root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# python ./http_server.py
Serving HTTP on 0.0.0.0 port 8000 ...
159.226.39.66 - - [04/Jun/2020 12:46:20] "GET / HTTP/1.1" 200 -
[]

root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# wget http://159.226.39.233:8000/
--2020-06-04 12:46:20-- http://159.226.39.233:8000/
正在连接 159.226.39.233:8000... 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.2"

index.html.2      100%[=====>]      207  --.-KB/s   in 0s
2020-06-04 12:46:20 (2.68 MB/s) - 已保存 "index.html.2" [207/207]

root@CN-VirtualBox:/mnt/shared/10-nat/10-nat# cat 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: 159.226.39.66
  </body>
</html>
```

图七 双 nat 穿透实验结果

可以看到，作为服务器的 h2 的内网 IP 为 10.21.0.1，虽然作为客户的 h1 的内网 IP 为 10.21.0.1，但是其映射的公网 IP 为 159.226.39.66，所以 h2 回应的报文中 RemoteIP 设置为 159.226.39.66。

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

答：

首先来看 ICMP 的报文格式：

类型 Type	代码 Code	校验和 Checksum
标识符 Identifier		序列号 Seq_num
Option		

假设 A 和 B 在局域网内，C 在公网上

A: IP: 192.168.0.2/24

B: IP: 192.168.0.3/24

Router: IP1: 192.168.0.1/24

IP2: 188.10.1.2 (公网地址)

C: IP: 200.10.2.1

如果 A 想要 Ping C，首先 A 在发送 ICMP 报文的时候，会根据（类型+代码）的值生成源端口号，根据标识符的值生成目的端口号，也就是说，路由器会收到 A 发来的如下 ICMP 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

路由器进行 SNAT，生成如下 NAT 表：

内网 IP	内网端口	协议	外网 IP	外网端口
192.168.0.2	Type + Code	ICMP	188.10.1.2	IDENTIFIER

其中 IDENTIFIER 是随机分配的端口号。

路由器修改 ICMP 报文为以下：

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

C 收到 ICMP 报文后，生成 ICMP 响应报文，路由器收到的 C 的响应报文如下：

源 IP	源端口	目的 IP	目的端口
200.10.2.1	Type + Code	188.10.1.2	IDENTIFIER

路由器查询 NAT 表，找到 NAT 表项，将目的 IP 和目的端口替换后转发 ICMP 响应报文即可。

2. 假设 SNAT 设备只有一个公网 IP（外部端口），请问如何修改设计，使得其可以支持多于 65535 个并发连接

答：可以设计每一个端口号绑定多个 IP 地址。一方面，当 SNAT 时，从内访问外网使用哪个端口由外网 IP 决定，所以即使多个外网 IP 绑定同一端口也不会引起冲突；另一方面，当 DNAT 时，从外访问内网使用哪个端口由目的 IP 和端口决定，如果绑定到目的端口上的内网 IP 多于一个，就同时向多个 IP 进行多播。不过这样一来可能会出现多个主机响应的问题。

(二) 实验代码详解

本次实验代码过长，不再赘述代码的内容。

一、nat.c:NAT 功能具体实现代码

(1) static int get_packet_direction(char *packet): 返回收到数据包的发送方向

首先解析数据包的 IP 首部，获取网络到本地字节序转换的源 IP 地址与目的 IP 地址。调用最长前缀匹配查找两个 IP 的转发表项，获取两个转发表项的端口。如果目的 IP 端口为外网端口，源 IP 端口为内网端口，那么返回 DIR_OUT；如果源 IP 端口为外网端口，目的 IP 端口为内网端口，那么返回 DIR_IN；否则返回 DIR_INVALID。

(2) void do_translation(iface_info_t *iface, char *packet, int len, int dir): 更换数据包的 IP 地址和端口号，重新计算校验和，更新 TCP 连接状态

首先获取 NAT 的互斥锁。

然后解析 IP 报头，如果方向为 DIR_IN 就计算经网络到本地字节序转换的源 IP 地址的哈希值，并找到该哈希值对应的表单；否则就计算经网络到本地字节序转换的目的 IP 地址的哈希值，并找到该哈希值对应的表单。

如果方向为 DIR_IN，遍历找到的表单，首先看有没有外网 IP、外网端口号与目的 IP、目的端口号相同的表项：

如果有，就按照表项中记录的信息，修改 TCP 中目的端口号为经本地到网络字节序转换的内网端口号，修改 IP 中目的 IP 地址为经本地到网络字节序转换的内网 IP。

如果没有，就查看 DNAT 规则表单。遍历表单，首先看有没有外网 IP、外网端口号与目的 IP、目的端口号相同的表项。如果有，就新建一个表项，将对应信息填入，调用宏 list_add_tail 添加到哈希表单中。

随后设置表项的 connection 状态中的外网连接结束符，外网最后序列号，外网确认序列号。

如果方向为 DIR_OUT，遍历找到的表单，首先看有没有内网 IP、内网端口号与源 IP、源端口号相同的表项：

如果没有，就新建一个表项，设置表项的内网 IP 为经网络到本地字节序转换的源 IP 地址，内网端口号为经网络到本地字节序转换的 TCP 中的源端口号，外网 IP 为 NAT 记录的外网端口的 IP，然后选取一个 NAT 设备未使用的端口号作为表项的外网端口号。最后将该表项添加到哈希表单中。

如果有（就算没有，经过上面的步骤之后也应该有了），就将 TCP 中的源端口号修改为经本地到网络字节序转换的表项记录的外网端口号，IP 首部中的源 IP 地址修改为经本地到网络字节序转换的外网 IP。

随后设置表项的 `connection` 状态中的内网连接结束符，内网最后序列号，内网确认序列号。

最后，重新计算 TCP 和 IP 校验和，更新表项访问时间，释放 NAT 锁，调用 `IP_send_packet` 将数据包转发。

(3) `void *nat_timeout()`：NAT 老化操作

和之前的老化操作类似，程序位于 `while` 循环中，每 1 秒执行一次。

首先获取 NAT 互斥锁，获取当前时间。

遍历所有的哈希表单：

对于一个哈希表单，如果非空，就遍历其下所有表项，如果当前时间大于表项访问时间 60 秒，就删除该表项；如果表项记录的 TCP 连接已经结束，也删除该表项。

最后释放互斥锁，`sleep(1)`。

(4) `int parse_config(const char *filename)`：配置 NAT 的初始信息

首先要调用 `if_name_to_iface` 从配置文件中获取端口的配置，存放到 NAT 结构体中，然后配置 DNAT 的 `rule`，如外网 IP、外网端口号到内网 IP、内网端口号的映射。

(5) `void nat_exit()`：释放 NAT 表

首先获取 NAT 互斥锁，然后遍历所有的哈希表单，删除所有哈希表项，调用 `kill` 杀死 NAT 老化进程，释放互斥锁。