Nat 实验设计报告

中国科学院大学

页

2022年5月13日

一、网络地址转换实验

# 实验内容

⽹络中的 IP 地址分为公⽹地址和私⽹地址两类，当在专用网内部的一些主机本来已经分配到了本地IP地址，但又想和因特网上的主机通信（并不需要加密）时，可使用NAT。。NAT不仅能解决IP地址不足的问题，而且还能够有效地避免来自网络外部的攻击，隐藏并保护网络内部的计算机 nat设备的主要⼯作包括：

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。

NAT的⼯作场景可以分为私网主机连接到公网服务器（ SNAT）和 私网主机作为服务器（DNAT）两种。在本次实验中，我们需要实现 nat在这两种情况下不同的功能。我们的测试包括三个部分：

1. SNAT实验：单nat下私网主机访问公网服务器；
2. DNAT实验： 公网主机访问私网服务器；
3. 构造⼀个包含两个 nat的拓扑，测试主机能否穿过两个 nat通信。

# 实验流程

（1）编写实验代码，完成nat.相关代码实现，本次实验中需要完成其中get\_packet\_direction、）do\_translation、nat\_timeout 、nat\_exit和parse\_config等函数，make生成nat可执行程序。

（2）完成SAT实验：在 n1 上运⾏ nat程序，在 h3 上运⾏ HTTP Server ，在 h1 和 h2 上分别访问 h3。

给定拓扑如图1：

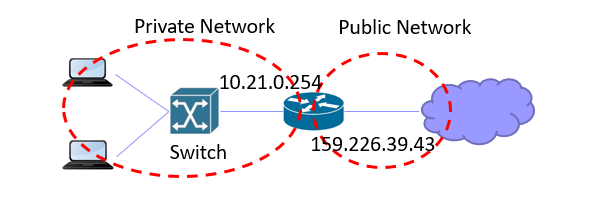


图1

① 运行给定网络拓扑(nat\_topo.py)；

② 在n1, h1, h2, h3上运行相应脚本；

③ 在n1上运行nat程序： n1# ./nat exp1.conf；

④ 在h3上运行HTTP服务：h3# python ./http\_server.py

⑤ 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>

补充测试代码(nat\_topo.py)：

    n1.cmd('./nat exp1.conf > n1-output.txt 2>&1 &' )

    h3.cmd('python http\_server.py > h3-output.txt 2>&1 &')

*for* h *in* (h1, h2):

        h.cmd('wget http://159.226.39.123:8000 > %s-output.txt 2>&1' % h)

    n1.cmd('pkill -SIGTERM nat')

（3）完成DNAT实验：在 n1 上运⾏ nat程序，在 h1 和 h2 上分别运⾏ HTTP Server ，在 h3 上访问 h1 和 h2 ；

① 运行给定网络拓扑(nat\_topo.py)；

② 在n1, h1, h2, h3上运行相应脚本；

③ 在n1上运行nat程序： n1# ./nat exp2.conf；

④ 在h1、h2上运行HTTP服务：h1/h2# python ./http\_server.py

⑤ 在h3上分别访问h1、h2的HTTP服务

h3# wget http://159.226.39.43:8000

h3# wget <http://159.226.39.43:8001>

补充测试代码(nat\_topo.py)：

    n1.cmd('./nat exp2.conf > n1-output.txt 2>&1 &' )

    h1.cmd('python http\_server.py > h1-output.txt 2>&1 &')

    h2.cmd('python http\_server.py > h2-output.txt 2>&1 &')

    h3.cmd('wget http://159.226.39.43:8000 > h3-output.txt 2>&1')

    h3.cmd('wget http://159.226.39.43:8001 > h3-output.txt 2>&1')

    n1.cmd('pkill -SIGTERM nat')

（3）完成SNAT实验构造⼀个包含两个 nat的拓扑，测试主机能否穿过两个 nat通信。

手动构造一个包含两个nat的拓扑h1 <-> n1 <-> n2 <-> h2，类比exp1.conf完成SNAT和DNAT的规则转换exp3\_1.config和exp3\_2c=.config。

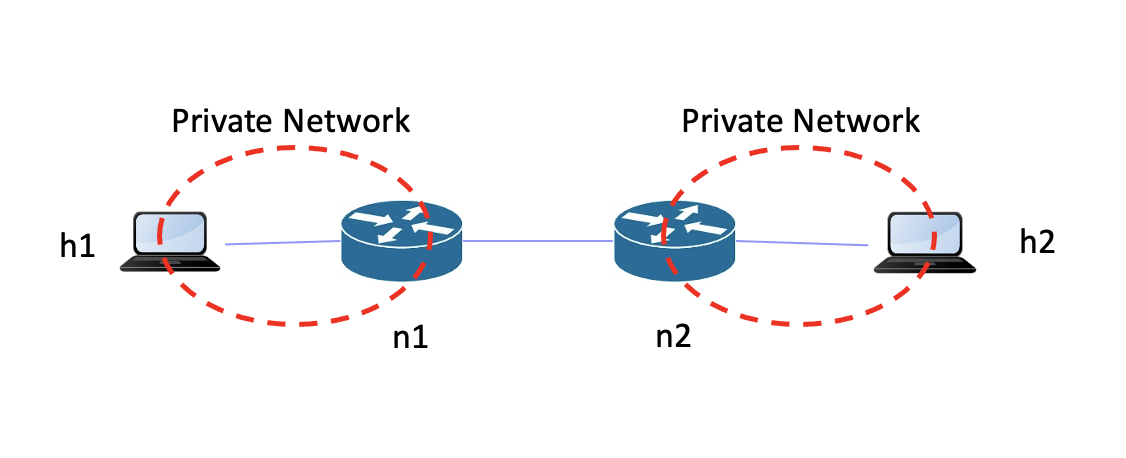


图2

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

拓扑：

class NATTopo(Topo):

    def build(self):

        h1 = *self*.addHost('h1')

        h2 = *self*.addHost('h2')

        n1 = *self*.addHost('n1')

        n2 = *self*.addHost('n2')

*self*.addLink(h1, n1)

*self*.addLink(n2, h2)

*self*.addLink(n1, n2)

测试代码：

    n1.cmd('./nat exp3\_1.conf &' )

    n2.cmd('./nat exp3\_2.conf &' )

    h2.cmd('python http\_server.py > h2-output.txt 2>&1 &')

    h1.cmd('wget http://159.226.39.123:8000')

    n1.cmd('pkill -SIGTERM nat')

    n2.cmd('pkill -SIGTERM nat')

# 实验设计

（1）确定 pkt 方向，比较 packet 的地址端口和匹配的路由表项对应的端口 即可判定其方向. 1. 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；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\_I。

static int get\_packet\_direction(char \*packet)

{

*//fprintf(stdout, "TODO: determine the direction of this packet.\n");*

    struct iphdr \*ihr = packet\_to\_ip\_hdr(packet);

    rt\_entry\_t \*sentry = longest\_prefix\_match(ntohl(ihr->saddr));

    rt\_entry\_t \*dentry = longest\_prefix\_match(ntohl(ihr->daddr));

*if*(sentry->iface==nat.internal\_iface && dentry->iface==nat.external\_iface)

*return* DIR\_OUT;

*if*(sentry->iface==nat.external\_iface && ntohl(ihr->daddr)==nat.external\_iface->ip)

*return* DIR\_IN;

*return* DIR\_INVALID;

}

（2）pkt 的重写，完成对合法数据包的处理，进行(internal\_ip, internal\_port) <-> (external\_ip,

external\_port)之间的转换，填充映射表。具体说来，若该数据包在NAT中有对应连接映射，根据方向重写, 发包。若该数据包暂无连接，尝试添加连接。

若该数据包的方向为DIR\_OUT，为该TCP连接的第一个数据包（请求连接数据包），NAT中没有对应连接映射 (SNAT)；该数据包的方向为DIR\_IN，为该TCP连接的第一个数据包，NAT中没有对应连接映射，但有对应处理规则 (DNAT)。确定映射关系时可以先用(rmt\_ip, rmt\_port) 定位到一组映射结构（链表），再根据数据包方向，决定用(rmt\_ip, rmt\_port) + (int\_ip, int\_port) 还是(rmt\_ip, rmt\_port) +(ext\_ip, ext\_port) 来确定唯一的映射结构，然后添加映射后更新检验和将数据包从相应端口发出。如果无法添加，则丢弃当前数据包，发送 ICMP 主机不可达。处理过程如下图：

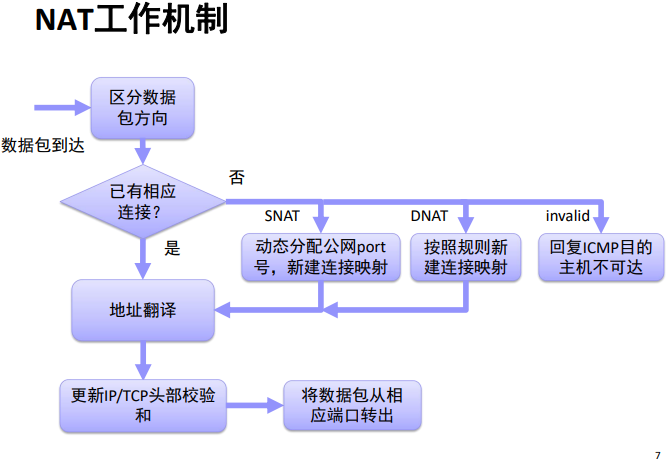


图3

void do\_translation(iface\_info\_t \*iface, char \*packet, int len, int dir)

{

*// fprintf(stdout, "TODO: do translation for this packet.\n");*

    pthread\_mutex\_lock(&nat.lock);

    struct iphdr \*ihr = packet\_to\_ip\_hdr(packet);

    struct tcphdr \*thr = packet\_to\_tcp\_hdr(packet);

    u32 rm\_ip = dir == DIR\_IN ? ntohl(ihr->saddr) : ntohl(ihr->daddr);

    u16 rm\_port = dir == DIR\_IN ? ntohs(thr->sport) : ntohs(thr->dport);

    u32 temp\_ip = dir == DIR\_IN ? ntohl(ihr->daddr) : ntohl(ihr->saddr);

    u16 temp\_port = dir == DIR\_IN ? ntohs(thr->dport) : ntohs(thr->sport);

    u8 id = hash8((char \*)&rm\_ip, 4);

    int exist = 0, found = 0;

    struct nat\_mapping \*entry = NULL;

*if* (dir == DIR\_OUT)

    {

        list\_for\_each\_entry(entry, &nat.nat\_mapping\_list[id], list)

        {

*if* (entry->remote\_ip == rm\_ip && entry->remote\_port == rm\_port && entry->internal\_ip == temp\_ip && entry->internal\_port == temp\_port)

            {

                exist = 1;

*break*;

            }

        }

*if* (!exist)

        {

            entry = malloc(sizeof(struct nat\_mapping));

            entry->remote\_ip = rm\_ip;

            entry->remote\_port = rm\_port;

            entry->internal\_ip = temp\_ip;

            entry->internal\_port = temp\_port;

            entry->external\_ip = nat.external\_iface->ip;

*for* (int i = NAT\_PORT\_MIN; i < NAT\_PORT\_MAX; i++)

            {

*if* (!nat.assigned\_ports[i])

                {

                    nat.assigned\_ports[i] = 1;

                    entry->external\_port = i;

                    found = 1;

*break*;

                }

            }

*if* (!found)

            {

                free(packet);

                free(entry);

*return*;

            }

            list\_add\_tail(&entry->list, &nat.nat\_mapping\_list[id]);

        }

        ihr->saddr = htonl(entry->external\_ip);

        thr->sport = htons(entry->external\_port);

        entry->conn.internal\_fin = thr->flags & TCP\_FIN;

        entry->conn.internal\_seq\_end = tcp\_seq\_end(ihr, thr);

        entry->conn.internal\_ack = thr->flags & TCP\_ACK;

    }

*else*

    { *// DIR\_IN*

        list\_for\_each\_entry(entry, &nat.nat\_mapping\_list[id], list)

        {

*if* (entry->remote\_ip == rm\_ip && entry->remote\_port == rm\_port && entry->external\_ip == temp\_ip && entry->external\_port == temp\_port)

            {

                exist = 1;

*break*;

            }

        }

*if* (!exist)

        {

            struct dnat\_rule \*rule = NULL;

            list\_for\_each\_entry(rule, &nat.rules, list)

            {

*if* (rule->external\_ip == temp\_ip && rule->external\_port == temp\_port)

                {

                    entry = malloc(sizeof(struct nat\_mapping));

                    entry->remote\_ip = rm\_ip;

                    entry->remote\_port = rm\_port;

                    entry->external\_ip = temp\_ip;

                    entry->external\_port = temp\_port;

                    entry->internal\_ip = rule->internal\_ip;

                    entry->internal\_port = rule->internal\_port;

                    list\_add\_tail(&entry->list, &nat.nat\_mapping\_list[id]);

                    found = 1;

*break*;

                }

            }

*if* (!found)

            {

                free(packet);

                free(entry);

*return*;

            }

        }

        ihr->daddr = htonl(entry->internal\_ip);

        thr->dport = htons(entry->internal\_port);

        entry->conn.external\_fin = thr->flags & TCP\_FIN;

        entry->conn.external\_seq\_end = tcp\_seq\_end(ihr, thr);

        entry->conn.external\_ack = thr->flags & TCP\_ACK;

    }

    entry->update\_time = time(NULL);

    thr->checksum = tcp\_checksum(ihr, thr);

    ihr->checksum = ip\_checksum(ihr);

    pthread\_mutex\_unlock(&nat.lock);

    ip\_send\_packet(packet, len);

}

（3）NAT老化：对认为已经结束的连接进⾏⽼化操作：

①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。

②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。

void \*nat\_timeout()

{

*while* (1) {

*// fprintf(stdout, "TODO: sweep finished flows periodically.\n");*

        sleep(1);

        pthread\_mutex\_lock(&nat.lock);

        struct nat\_mapping \*entry, \*q;

*for* (int i = 0; i < HASH\_8BITS; i++)

        {

            list\_for\_each\_entry\_safe(entry, q, &nat.nat\_mapping\_list[i], list)

            {

*if* (is\_flow\_finished(&entry->conn) || time(NULL) - entry->update\_time >= TCP\_ESTABLISHED\_TIMEOUT)

                {

                    list\_delete\_entry(&entry->list);

                    nat.assigned\_ports[entry->external\_port] = 0;

                    free(entry);

                }

            }

        }

        pthread\_mutex\_unlock(&nat.lock);

    }

*return* NULL;

}

（4）根据config⽂件中读取的每⼀⾏字符串，分别配置 external-iface , internal-iface 和 DNAT Rules。

int parse\_config(const char \*filename)

{

*// fprintf(stdout, "TODO: parse config file, including i-iface, e-iface (and dnat-rules if existing).\n");*

    FILE \*fd = fopen(filename, "r");

*if* (!fd)

*return* -1;

    char buf[MAX\_LINE];

*for* (int i = 0; i < 2; i++)

    {

        fgets(buf, MAX\_LINE, fd);

        char \*face = strtok(buf, " ");

        face = strtok(NULL, "\n");

        iface\_info\_t \*iface = if\_name\_to\_iface(face);

*if* (!iface)

        {

            log(ERROR, "get iface error");

            exit(0);

        }

*if* (i == 0)

            nat.internal\_iface = iface;

*else*

            nat.external\_iface = iface;

    }

    int index[10], c;

*while* (1)

    {

*while* ((c = fgetc(fd)) != ':' && c != EOF)

            ;

*if* (c == EOF)

*return* 0;

*if* (fscanf(fd, " %d.%d.%d.%d:%d -> %d.%d.%d.%d:%d", index, index + 1, index + 2, index + 3, index + 4, index + 5, index + 6, index + 7, index + 8, index + 9) == 10)

        {

            unsigned ip1 = (((((index[0] << 8) + index[1]) << 8) + index[2]) << 8) + index[3];

            unsigned port1 = index[4];

            unsigned ip2 = (((((index[5] << 8) + index[6]) << 8) + index[7]) << 8) + index[8];

            unsigned port2 = index[9];

            struct dnat\_rule \*rule = malloc(sizeof(struct dnat\_rule));

            rule->external\_ip = ip1;

            rule->internal\_ip = ip2;

            rule->external\_port = (u16)port1;

            rule->internal\_port = (u16)port2;

            list\_add\_tail(&rule->list, &nat.rules);

        }

    }

*return* 0;

}

# 实验结果

(1)SNAT实验：

h1->h3:

<!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>

h2->h3:

<!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>

结果显示，从h3角度来看，h1和h2的ip一致为159.226.39.43说明NAT转换成功。

(2)DNAT实验

h3->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>

h3->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>

h1发回IP为10.21.0.1，h1发回IP为10.21.0.2，它们发回到h3自己没有经过nat转换的私网ip，结果符合预期。

（3）双NAT穿透实验

<!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: 159.226.39.43

        </body>

</html>

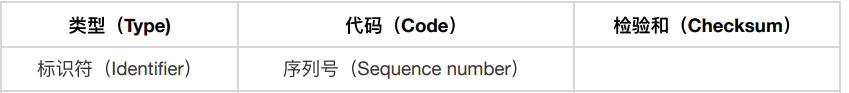
h2正常返回页面，h2ip为私网ip10.21.0.2，h2认为的h1地址为公网ip159.226.39.43符合实验预期结果。

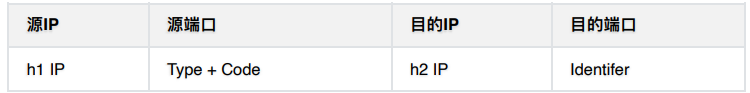
# 思考题

**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**：

ICMP 协议的报文如下：





首先 ICMP 报文可以分为两类，询问报文、差错报告报文。

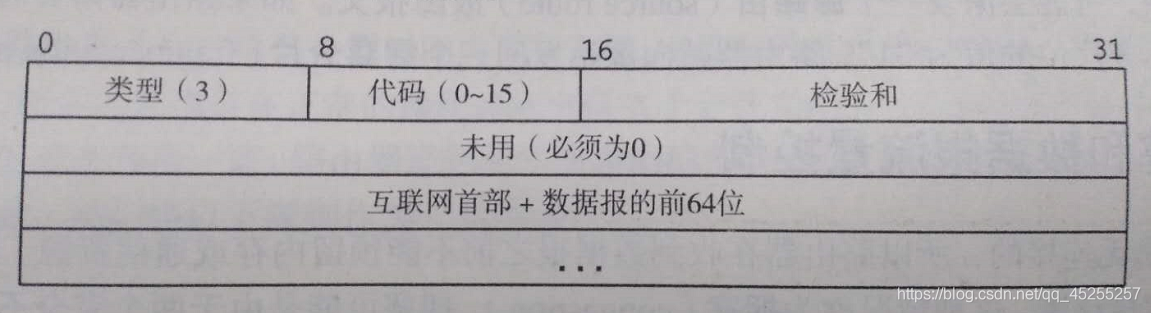
（1）

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。假设主机结点 h1 要 ping 结点 h2，h1 发送 ICMP报⽂， h1会根据报⽂头部中包含的类型信息 Type 和代码信息 Code ⽣成源端⼝号，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，添加相应的 nat表项：



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。

但根据报文格式它将 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。一种解决方案是 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ＩＣＭＰ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。

**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。（提示：不需要DNAT机制）。**

**答：**

这种情况是要两个处于不同 nat⽹络下的结点建⽴直接连接，可以称为 nat穿透。NAT穿透，简单地讲就是要让处于不同NAT网络下的两个节点(Peer)建立直接连接，只知道自己的内网地址是肯定不行的，他们需要知道对方的公网IP和端口，然后双方互相向对方发送数据包，从而建立起连接。整个流程可以看做两个关键步骤：

１.发现自己的公网IP和Port；

２.将自己的IP和Port共享给对方

其中，第二步，我们可以简单地通过一个第三方服务器来交换双方的IP和Port，但是第一步就比较困难，我们不妨根据不同类型的NAT的特点，分别看看在不同的NAT类型下，怎样才能拿到一个可供通讯的公网IP和Port。

　　基本思路如下：

１.给定具有公网IP和Port的服务(Server 1)

２.(Client)发送一个数据包给这个公网服务(Server1)

３.(Server1)通过解析IP协议包，就能得知(Client)的公网地(eAddr:ePort) ，服务器收到主机1和主机2的连接后，知道1与2的公网地址和NAT分配给它们的端口号.

４.(Server1)将该公网地址(eAddr:ePort)回传给Client1和2

５.这样两个不同的节点Client1和Client2通过第三方服务器交换公网地址(eAddr1:ePort1)(eAddr2:ePort2)以及对方的NAT分配的端口号从而避开了DNAT的解析。

６.自由地进行通讯

参考资料：

【１】[ICMP报文如何通过NAT来地址转换](https://blog.csdn.net/sinat_33822516/article/details/81088724?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromBaidu-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromBaidu-1.control)。

【２】[差错报文](https://blog.csdn.net/qq_45255257/article/details/108834673)ICMP。