

# 12-NAT-report

作者：苗屹松

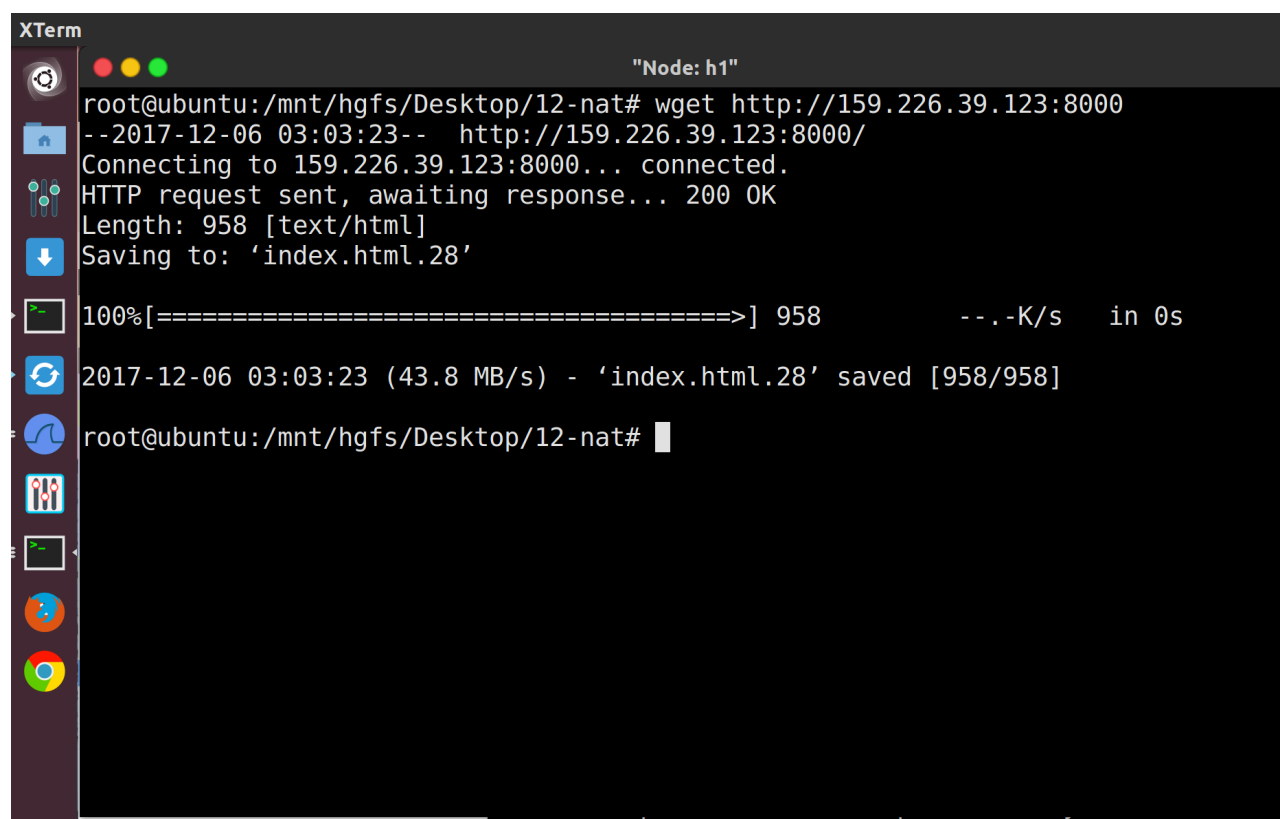
日期：2017/Dec/6

## 1. Overview

这次实验的主题非常清晰，完成一个网络地址转换.....器?(找不到合适的名词了😓)

这次实验调试起来也比上次做OSPF容易。

## 2. 实验结果展示

A screenshot of an XTerm terminal window titled "Node: h1". The terminal shows a root user at an Ubuntu machine in the directory /mnt/hgfs/Desktop/12-nat. The user runs the command 'wget http://159.226.39.123:8000'. The output shows the connection to 159.226.39.123:8000 is successful, the HTTP request is sent, and the response is 200 OK. The file 'index.html.28' is saved with a length of 958 bytes. The terminal also shows a progress bar for the download and the final status: '2017-12-06 03:03:23 (43.8 MB/s) - 'index.html.28' saved [958/958]'. The prompt returns to root@ubuntu:/mnt/hgfs/Desktop/12-nat#.

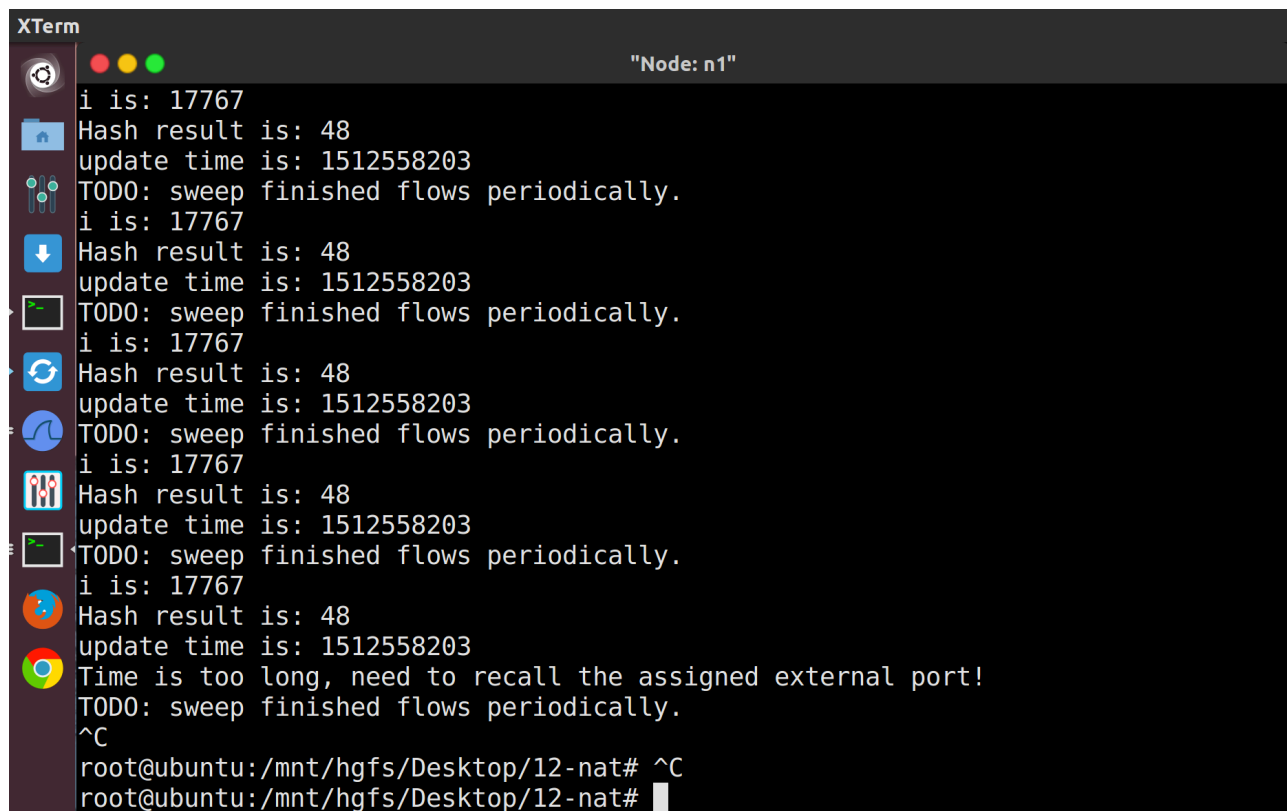
```
XTerm
"Node: h1"
root@ubuntu:/mnt/hgfs/Desktop/12-nat# wget http://159.226.39.123:8000
--2017-12-06 03:03:23--  http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 958 [text/html]
Saving to: 'index.html.28'

100%[=====] 958          --.-K/s   in 0s

2017-12-06 03:03:23 (43.8 MB/s) - 'index.html.28' saved [958/958]

root@ubuntu:/mnt/hgfs/Desktop/12-nat#
```

(成功完成wget，结果和nat-reference相同)



XTerm "Node: n1"

```
i is: 17767
Hash result is: 48
update time is: 1512558203
TODO: sweep finished flows periodically.
i is: 17767
Hash result is: 48
update time is: 1512558203
TODO: sweep finished flows periodically.
i is: 17767
Hash result is: 48
update time is: 1512558203
TODO: sweep finished flows periodically.
i is: 17767
Hash result is: 48
update time is: 1512558203
TODO: sweep finished flows periodically.
i is: 17767
Hash result is: 48
update time is: 1512558203
Time is too long, need to recall the assigned external port!
TODO: sweep finished flows periodically.
^C
root@ubuntu:/mnt/hgfs/Desktop/12-nat# ^C
root@ubuntu:/mnt/hgfs/Desktop/12-nat#
```

(成功完成对已经完成的连接的老化操作)

## 3. 实验原理 & 关键步骤的实现

### 3.1 Direction的获知

老师在PPT中给了很好的描述：

DIR\_OUT: 源地址为内部地址，目的地址为外部地址

DIR\_IN: 源地址为外部地址，目的地址为external\_iface

我翻译成C代码:

```

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 ip_saddr = ntohl(ip->saddr);
    u32 ip_daddr = ntohl(ip->daddr);
    if(longest_prefix_match(ip_saddr)->iface->ip == nat.internal_iface->ip){
        if(longest_prefix_match(ip_daddr)->iface->ip == nat.external_iface-
>ip){
            return DIR_OUT;
        }
    }
    if(longest_prefix_match(ip_saddr)->iface->ip == nat.external_iface->ip){
        if(ip_daddr == nat.external_iface->ip){
            return DIR_IN;
        }
    }
    return DIR_INVALID;
}

```

## 3.2 NAT地址翻译

做完实验后，我试图用 **说人话** 的方式来解释实验中最核心的这一块：

并不是主机交换数据，而是应用程序交换数据！

TCP协议的出发点，就是在软件层面上做了很多port。

对于每一个应用程序(或者我们讲，进程)，都有固定的port，来发送和接收数据。

内部主机和外部主机连接时，有内部port，内部ip，也有外部port和外部ip，NAT就是维护这一组映射关系。

所以最核心的部分，我们用伪代码的形式可以这样来描述：

```

if direction == DIR_OUT:
    packet->sport = mapping得到的外部端口
    packet->saddr = mapping得到的外部ip
if direction == DIR_IN:
    packet->dport = mapping得到的内部端口
    packet->daddr = mapping得到的内部ip

```

最核心的思想就是这样，非常简洁

我的C代码的片段是：

```

if(direction == DIR_OUT){
    //省略很多

    current_iphdr->saddr = htonl(nat.external_iface->ip);
    current_tcphdr->sport = htons(current_nat_mapping->external_port);

    current_tcphdr->checksum = tcp_checksum(current_iphdr,
current_tcphdr);

    current_iphdr->checksum = ip_checksum(current_iphdr);
    ip_send_packet(packet, len);
}

if(direction == DIR_IN){
    //省略很多

    current_iphdr->daddr = htonl(current_nat_mapping->internal_ip);

    current_tcphdr->dport = htons(current_nat_mapping->internal_port);

    current_tcphdr->checksum = tcp_checksum(current_iphdr,
current_tcphdr);

    current_iphdr->checksum = ip_checksum(current_iphdr);
    ip_send_packet(packet, len);
}

```

### 3.3 管理NAT table

NAT table的管理从high-level来讲是这样的：

```

根据hash结果，直接查找相应的nat映射
if found:
    发送相应的数据包
else:
    assign_external_port()
    插入新的nat映射
    根据刚刚插入的信息发送相应的数据包

```

下面要讲两个比较细节的东西：

#### 3.3.1 assign\_external\_port

```

int assign_external_port(){
    int r;
    int can = 0;
    while(can == 0){
        r = rand() % 65536;
        if(nat.assigned_ports[r] == 0){
            can = 1;
        }
    }
    return r;
}

```

在这个函数里，我们随机指派一个外部端口，但不能是已经指派过的！

### 3.3.2 hash

```

char sport_str[8];
sprintf(sport_str, "%d", sport);
int sport_hash_result = hash8(sport_str, (int)strlen(sport_str));

int assigning_port = assign_external_port();
char assign_str[8];
sprintf(assign_str, "%d", assigning_port);
int assign_hash_result = hash8(assign_str,
(int)strlen(assign_str));

```

上面这些代码给出了hash方法：对端口号 `hash()`

我一开始非常疑惑：当direction分别为DIR\_IN 和 DIR\_OUT时，端口号已经被NAT转换过了，什么自变量才能在两种情况下都能被 `hash()` 到相同的结果呢？

向林海涛同学请教了这个问题，他建议我：可以保存两个duplication的nat\_mapping，分别放到源端口的映射结果、分配的外部端口的映射结果。

所以当内部、外部端口查询 `hash()` 时得到的是duplication的nat\_mapping.

我暂时没有想到更好的方法，就采取了他的建议。

## 3.4 老化操作

老化操作有两种情况：

### 3.4.1 双方都FIN，且一方发送RST:

```

        if(current_tcphdr->flags == TCP_FIN + TCP_ACK){
            printf("It is a FIN + ACK!\n");
            current_nat_mapping->conn.external_fin = 1;
        }

        if(current_tcphdr->flags == TCP_RST){
            printf("It is a RST!\n");
            if(current_nat_mapping->conn.internal_fin +
current_nat_mapping->conn.external_fin == 2){
                nat.assigned_ports[current_nat_mapping->
>external_port] = 0;

                printf("both FIN, so recall the assigned
external port\n");

                list_delete_entry(&current_nat_mapping->list);
            }
        }
    }
}

```

### 3.4.2 超过60秒未传输数据的连接

```

        struct nat_mapping * current_nat_mapping;
        list_for_each_entry(current_nat_mapping,
&nat.nat_mapping_list[hash_result], list){
            printf("update time is: %ld\n", current_nat_mapping->
>update_time);

            if(now - current_nat_mapping->update_time > 5){
                printf("Time is too long, need to recall the assigned
external port!\n");

                nat.assigned_ports[current_nat_mapping->external_port]
= 0;

                list_delete_entry(&current_nat_mapping->list);
            }
        }
    }
}

```

## 4. 实验总结

实验中的一个bug很有趣：

我之前没有领会hash的意思，每次发送都错误地重新指定一个新的external\_port,

h1 发送 wget, h2确实也收到了，但当h1发送第二个TCP包时，h2就发送RST了，说明有严重的错误。

现在来看错误的原因非常明显：两次的external\_port(也就是source port)是不同的，这违反了TCP的协议。

我觉得传输层挺有趣的，期待后面的实验！

## 5. Reference & Give Credit to

---

谢谢林海涛同学建议我使用的hash的方法

我有时间会想想除了duplication之外还有哪些hash的方法