网络地址转换(NAT)实验报告

姓名：钟赟

学号：2016K8009915009

实验内容

NAT映射表管理

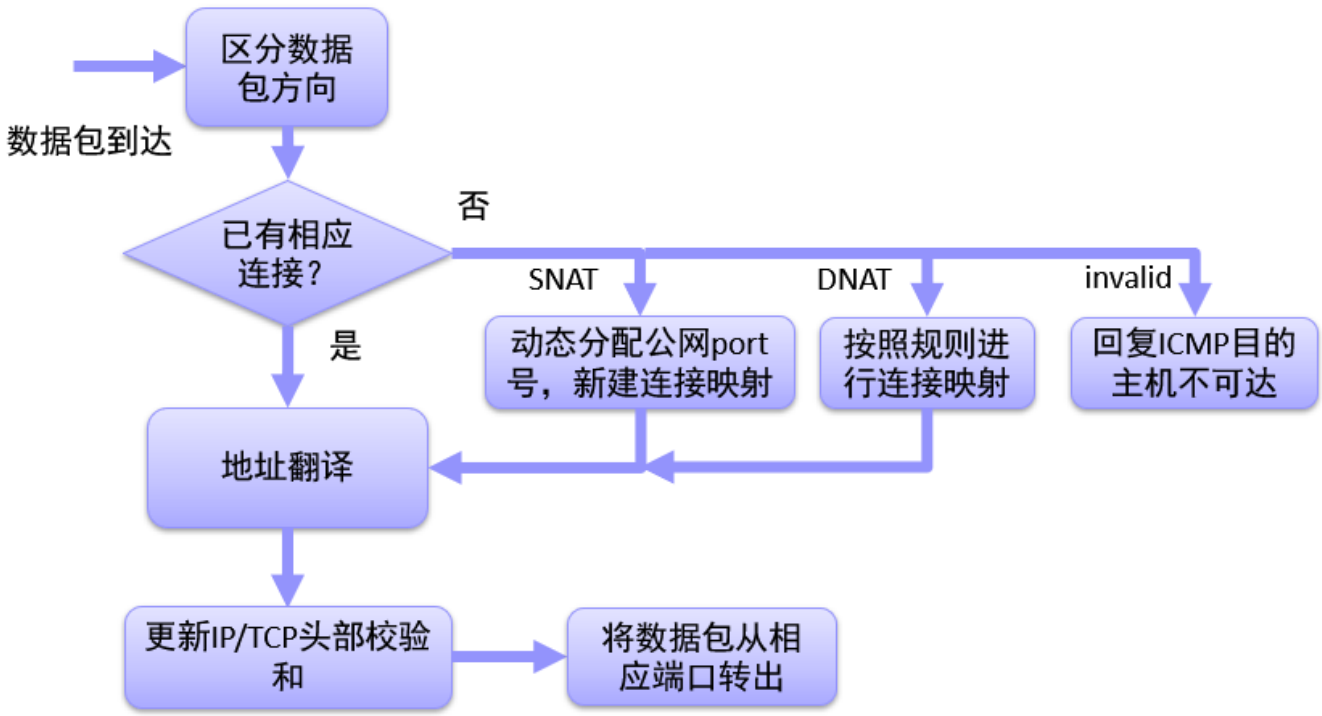
- 维护NAT连接映射表，支持映射的添加、查找、更新和老化操作

数据包的翻译操作

- 对到达的合法数据包，进行`IP`和`Port`转换操作，更新头部字段，并转发数据包
- 对于到达的非法数据包，回复`ICMP Destination Host Unreachable`

实验步骤

本次实验按照如下流程图进行：



1. 区分数据包方向

- 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**
- 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**
- **判断依据**：查询路由表，根据目的地址相应转发条目对应的iface判断地址类别

- 代码如下：

```
static int get_packet_direction(char *packet)
{
  fprintf(stdout, "Determine packet direction.\n");
  struct iphdr *ip = packet_to_ip_hdr(packet);
  u32 src = ntohl(ip->saddr);
  u32 dst = ntohl(ip->daddr);
  rt_entry_t *s_entry = longest_prefix_match(src);
  rt_entry_t *d_entry = longest_prefix_match(dst);

  if (!strcmp(s_entry->iface->name, nat.internal_iface->name) && \
      !strcmp(d_entry->iface->name, nat.external_iface->name))
      return DIR_OUT;

  if (!strcmp(s_entry->iface->name, nat.external_iface->name) && \
      !strcmp(d_entry->iface->name, nat.external_iface->name))
      return DIR_IN;

  return DIR_INVALID;
}
```

2. NAT地址翻译

2.1 NAT中有对应连接 (Existing)

- 查找映射关系，进行`(internal_ip, internal_port) <-> (external_ip, external_port)`之间的转换
- 更新IP/TCP数据包头部字段(包括校验和)

2.2 SNAT

- `saddr = external_iface->ip;` `sport = assign_external_port();`
- 建立连接映射关系：`(internal_ip, internal_port) <-> (external_ip, external_port)`
- 更新IP/TCP数据包头部字段(包括校验和)
- 代码如下：

```
u16 assign_external_port()
{
  u16 i = NAT_PORT_MIN;
  for (; i < NAT_PORT_MAX; i ++) {
      if (!nat.assigned_ports[i]) {
          nat.assigned_ports[i] = 1;
          return i;
      }
  }
  return 0;
}
```

```
if (dir == DIR_OUT) {
      list_for_each_entry_safe(mapping_entry, entry, head, list)
      {
```

```
            if (mapping_entry->internal_ip == saddr && \
                mapping_entry->internal_port == sport) {
                found = 1;
                break;
            }
        }
        // build a new mapping
        if (!found) {
            mapping_entry = (struct nat_mapping *)malloc(sizeof(struct nat_mapping));
            init_list_head(&mapping_entry->list);
            mapping_entry->internal_ip = saddr;
            mapping_entry->internal_port = sport;
            mapping_entry->external_ip = nat.external_iface->ip;
            mapping_entry->external_port = assign_external_port();
            mapping_entry->remote_ip = daddr;
            mapping_entry->remote_port = dport;
            mapping_entry->update_time = time(NULL);
            bzero(&(entry->conn), sizeof(struct nat_connection));
            list_add_tail(&(mapping_entry->list), &(nat.nat_mapping_list[key]));
        }

        // update mapping
        mapping_entry->conn.internal_fin = fin | rst;
        mapping_entry->conn.internal_ack = ack | rst;
        mapping_entry->conn.external_fin = rst;
        mapping_entry->conn.external_ack = rst;

        ip->saddr = htonl(mapping_entry->external_ip);
        tcp->sport = htons(mapping_entry->external_port);
        ip->checksum = ip_checksum(ip);
        tcp->checksum = tcp_checksum(ip, tcp);
    }
```

2.3 DNAT

- `daddr = rule->daddr;` `dport = rule->dport;`
- 建立连接映射关系：`(internal_ip, internal_port) <-> (external_ip, external_port)`
- 更新IP/TCP数据包头部字段(包括校验和)
- 代码如下：

```
    else if (dir == DIR_IN){
        found = 0;
        mapping_entry = NULL;
        entry = NULL;
        list_for_each_entry_safe(mapping_entry, entry, head, list)
        {
            if (mapping_entry->external_ip == daddr && \
                mapping_entry->external_port == dport) {
                found = 1;
                break;
            }
        }
```

```
        if (!found) {
            struct dnat_rule *rule, *q;
            printf("DIR_IN: non-recorded src.\n");
            int found_rule = 0;
            list_for_each_entry_safe(rule, q, &(nat.rules), list) {
                if (rule->external_ip == daddr && rule->external_port == dport)
                {
                    printf("DIR_IN: FOUND.\n");
                    found_rule = 1;
                    break;
                }
            }
            if (found_rule) {
                mapping_entry = (struct nat_mapping *)malloc(sizeof(struct nat_mapping));
                init_list_head(&mapping_entry->list);
                mapping_entry->internal_ip = rule->internal_ip;
                mapping_entry->internal_port = rule->internal_port;
                mapping_entry->external_ip = rule->external_ip;
                mapping_entry->external_port = rule->external_port;
                mapping_entry->remote_ip = saddr;
                mapping_entry->remote_port = sport;
                list_add_tail(&(mapping_entry->list), &(nat.nat_mapping_list[key]));
            }
        }

        // update mapping
        mapping_entry->conn.internal_fin = fin;
        mapping_entry->conn.internal_ack = ack;
        mapping_entry->conn.external_fin = fin | rst;
        mapping_entry->conn.external_ack = ack | rst;

        ip->daddr = htonl(mapping_entry->internal_ip);
        tcp->dport = htons(mapping_entry->internal_port);
        ip->checksum = ip_checksum(ip);
        tcp->checksum = tcp_checksum(ip, tcp);
    }
    ip_send_packet(packet, len);

    pthread_mutex_unlock(&nat.lock);
}
```

2.4 IVALID

- 回复ICMP目的主机不可达

3. 转发数据包

- `ip_send_packet(packet, len);`

4. NAT老化操作

- 对于已经结束的连接，收回已分配的端口号，释放连接映射资源
    - 双方都已发送FIN且回复相应ACK的连接，可以直接回收

    - 一方发送RST包的连接，可以直接回收
    - 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
- 代码如下;

```
void *nat_timeout()
{
  while (1) {
      pthread_mutex_lock(&nat.lock);
      for (int i = 0; i < HASH_8BITS; i ++) {
          struct list_head *head = &nat.nat_mapping_list[i];
          struct nat_mapping *mapping_entry, *entry;
          list_for_each_entry_safe(mapping_entry, entry, head, list)
          {
              mapping_entry->update_time += 1;
              if (time(NULL) - mapping_entry->update_time > TCP_ESTABLISHED_TIMEOUT ||\
                  is_flow_finished(&mapping_entry->conn))
              {
                  nat.assigned_ports[mapping_entry->external_port] = 0;
                  list_delete_entry(&mapping_entry->list);
                  free(mapping_entry);
                  fprintf(stdout, "Sweep aged connection.\n");
              }
          }
      }
      pthread_mutex_unlock(&nat.lock);
      sleep(1);
  }
  return NULL;
}
```

5. NAT退出操作

- 退出NAT时的操作：删除释放mapping，结束nat_timeout进程
- 代码如下：

```
void nat_exit()
{
  pthread_mutex_lock(&nat.lock);
  for (int i = 0; i < HASH_8BITS; i ++) {
      struct list_head *head = &nat.nat_mapping_list[i];
      struct nat_mapping *mapping_entry, *entry;
      list_for_each_entry_safe(mapping_entry, entry, head, list)
      {
          list_delete_entry(&mapping_entry->list);
          free(mapping_entry);
      }
  }
  pthread_kill(nat.thread, SIGTERM);

  pthread_mutex_unlock(&nat.lock);
}
```

实验结果

1. SNAT实验

验证方法

- 运行给定网络拓扑(nat_topo.py)
- 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
```

- 在n1上运行nat程序： `n1# ./nat`
- 在h3上运行HTTP服务：`h3# python ./http_server.py`
- 在h1, h2上分别访问h3的HTTP服务

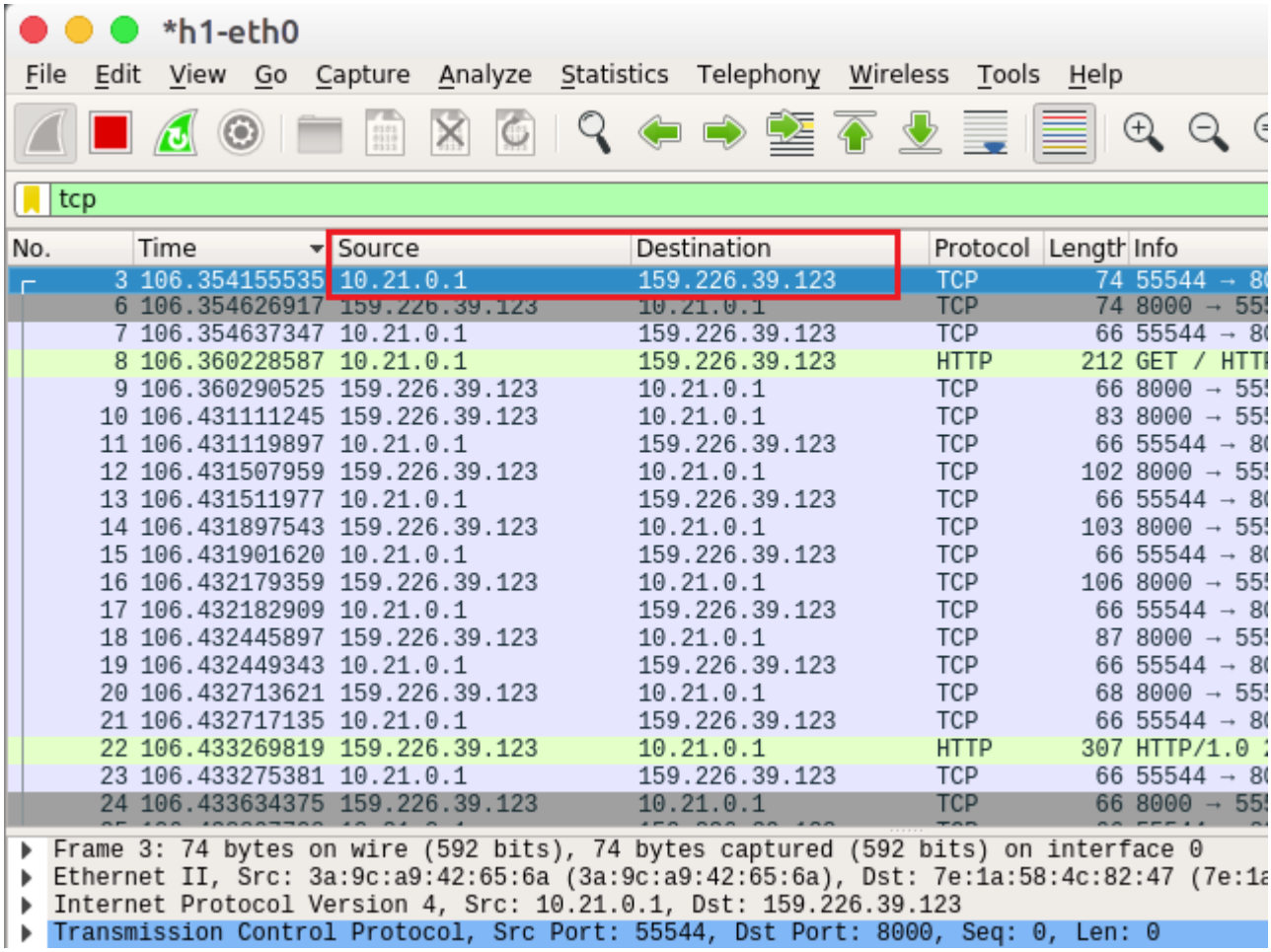
```
h1# wget http://159.226.39.123:8000 (公网)

h2# wget http://159.226.39.123:8000
```

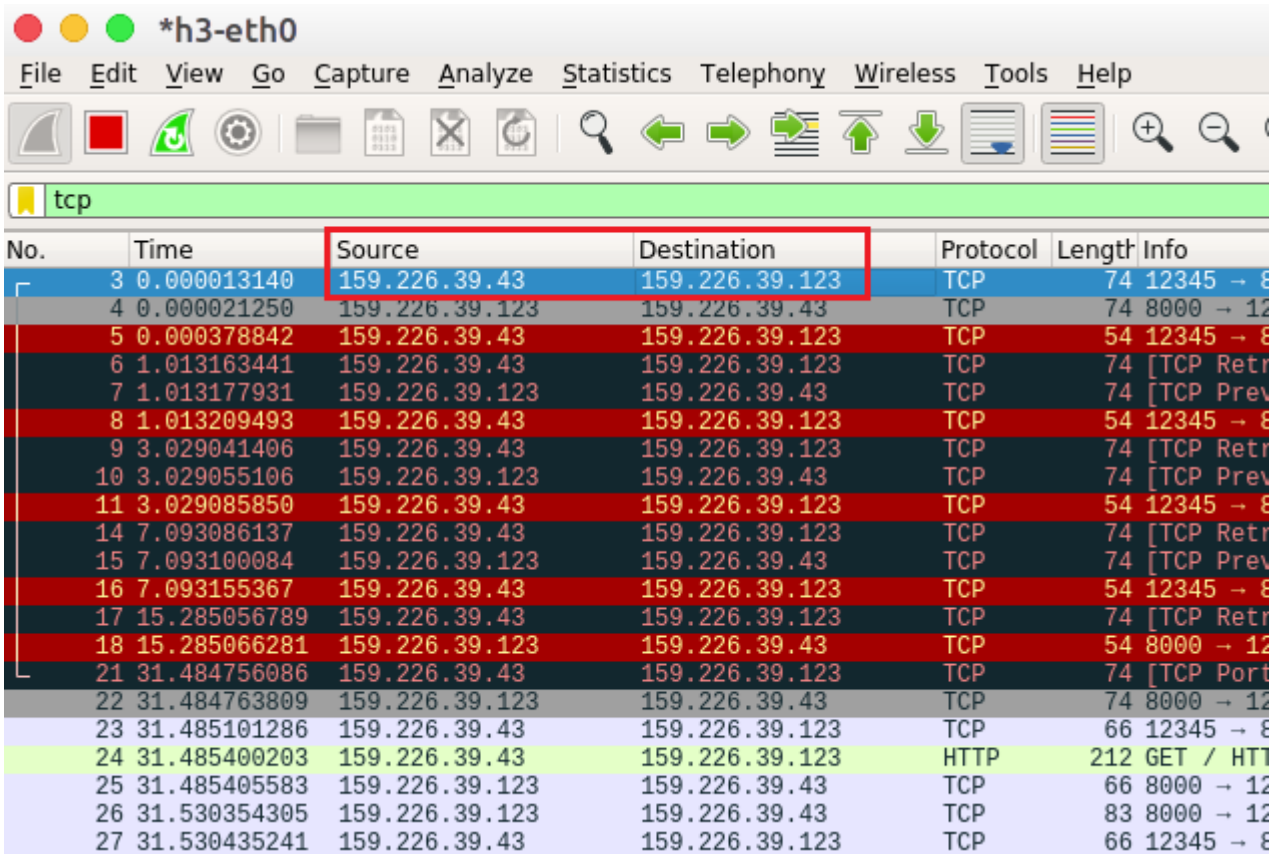
获取网页

"Node: h1"

```
root@zy-VB:09-nat# wget http://159.226.39.123:8000
--2019-04-30 22:12:22--  http://159.226.39.123:8000/
Connecting to 159.226.39.123:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 241 [text/html]
Saving to: 'index.html'

index.html          100%[===============>]     241  --.-KB/s    in 0s

2019-04-30 22:12:22 (29.3 MB/s) - 'index.html' saved [241/241]

root@zy-VB:09-nat# cat index.html

<!doctype html>
<html>
        <head> <meta charset="utf-8">
                <title>Network IP Address</title>
        </head>
        <body>
            My network IP is: addr:159.226.39.123
addr:127.0.0.1

            Remote IP is: 159.226.39.43
        </body>
</html>
root@zy-VB:09-nat#
```

抓包结果


*h1-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp
No. Time Source Destination Protocol Length Info
3 106.354155535 10.21.0.1 159.226.39.123 TCP 74 55544 → 8
6 106.354626917 159.226.39.123 10.21.0.1 TCP 74 8000 → 55
7 106.354637347 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
8 106.360228587 10.21.0.1 159.226.39.123 HTTP 212 GET / HTT
9 106.360290525 159.226.39.123 10.21.0.1 TCP 66 8000 → 55
10 106.431111245 159.226.39.123 10.21.0.1 TCP 83 8000 → 55
11 106.431119897 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
12 106.431507959 159.226.39.123 10.21.0.1 TCP 102 8000 → 55
13 106.431511977 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
14 106.431897543 159.226.39.123 10.21.0.1 TCP 103 8000 → 55
15 106.431901620 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
16 106.432179359 159.226.39.123 10.21.0.1 TCP 106 8000 → 55
17 106.432182909 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
18 106.432445897 159.226.39.123 10.21.0.1 TCP 87 8000 → 55
19 106.432449343 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
20 106.432713621 159.226.39.123 10.21.0.1 TCP 68 8000 → 55
21 106.432717135 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
22 106.433269819 159.226.39.123 10.21.0.1 HTTP 307 HTTP/1.0
23 106.433275381 10.21.0.1 159.226.39.123 TCP 66 55544 → 8
24 106.433634375 159.226.39.123 10.21.0.1 TCP 66 8000 → 55
Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: 3a:9c:a9:42:65:6a (3a:9c:a9:42:65:6a), Dst: 7e:1a:58:4c:82:47 (7e:1a
Internet Protocol Version 4, Src: 10.21.0.1, Dst: 159.226.39.123
Transmission Control Protocol, Src Port: 55544, Dst Port: 8000, Seq: 0, Len: 0


*h3-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp
No. Time Source Destination Protocol Length Info
3 0.000013140 159.226.39.43 159.226.39.123 TCP 74 12345 → 8
4 0.000021250 159.226.39.123 159.226.39.43 TCP 74 8000 → 12
5 0.000378842 159.226.39.43 159.226.39.123 TCP 54 12345 → 8
6 1.013163441 159.226.39.43 159.226.39.123 TCP 74 [TCP Retr
7 1.013177931 159.226.39.123 159.226.39.43 TCP 74 [TCP Prev
8 1.013209493 159.226.39.43 159.226.39.123 TCP 54 12345 → 8
9 3.029041406 159.226.39.43 159.226.39.123 TCP 74 [TCP Retr
10 3.029055106 159.226.39.123 159.226.39.43 TCP 74 [TCP Prev
11 3.029085850 159.226.39.43 159.226.39.123 TCP 54 12345 → 8
14 7.093086137 159.226.39.43 159.226.39.123 TCP 74 [TCP Retr
15 7.093100084 159.226.39.123 159.226.39.43 TCP 74 [TCP Prev
16 7.093155367 159.226.39.43 159.226.39.123 TCP 54 12345 → 8
17 15.285056789 159.226.39.43 159.226.39.123 TCP 74 [TCP Retr
18 15.285066281 159.226.39.123 159.226.39.43 TCP 54 8000 → 12
21 31.484756086 159.226.39.43 159.226.39.123 TCP 74 [TCP Port
22 31.484763809 159.226.39.123 159.226.39.43 TCP 74 8000 → 12
23 31.485101286 159.226.39.43 159.226.39.123 TCP 66 12345 → 8
24 31.485400203 159.226.39.43 159.226.39.123 HTTP 212 GET / HTT
25 31.485405583 159.226.39.123 159.226.39.43 TCP 66 8000 → 12
26 31.530354305 159.226.39.123 159.226.39.43 TCP 83 8000 → 12
27 31.530435241 159.226.39.43 159.226.39.123 TCP 66 12345 → 8

结果分析

内网h1和h2经过n1的正确转换，成功访问了外网h3的HTTP服务。

2. DNAT实验

验证方法

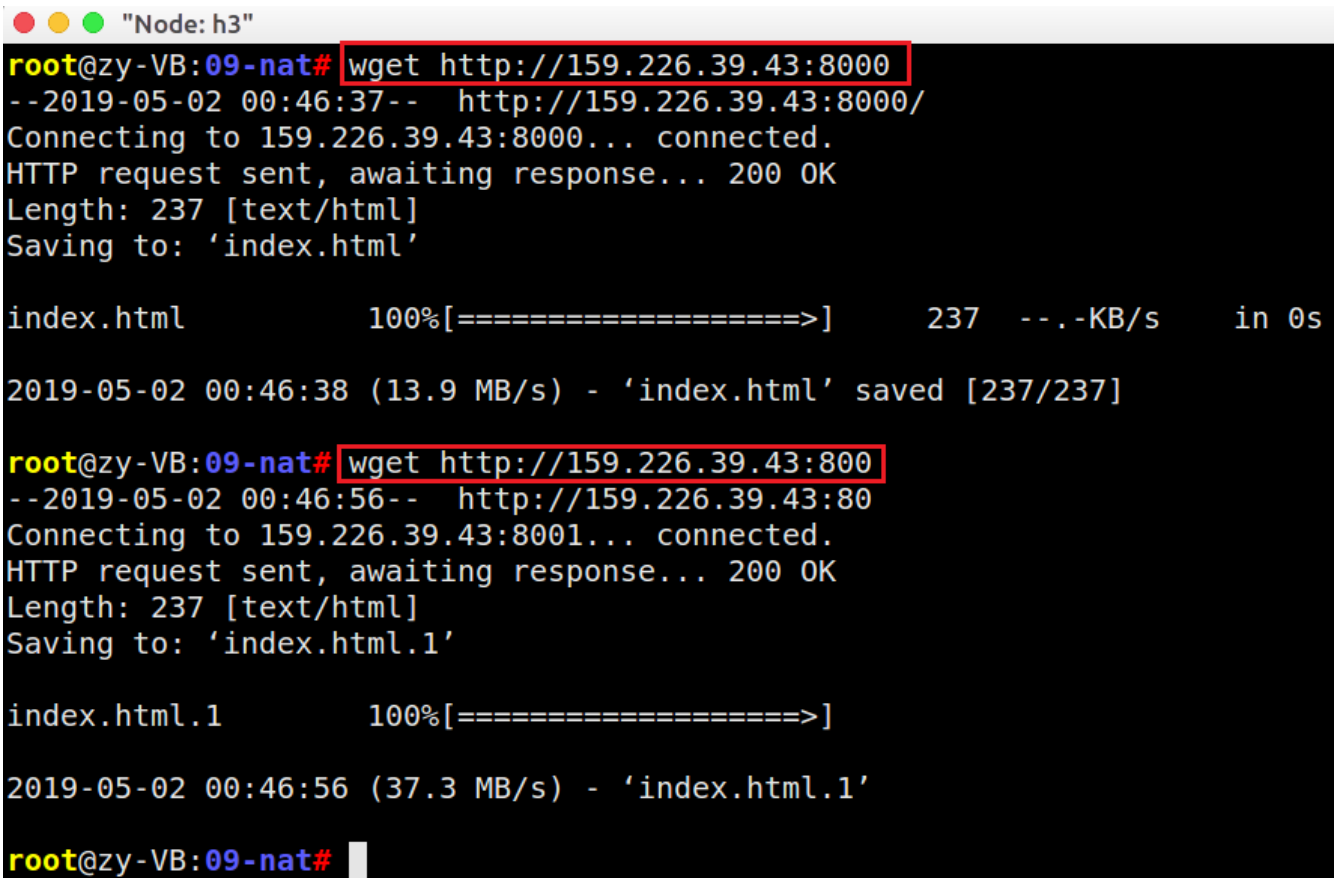
- 运行给定网络拓扑`nat_topo.py`
- 在n1, h1, h2, h3上运行相应脚本

```
n1   # disable_arp.sh, disable_icmp.sh, disable_ip_forward.sh, disable_ipv6.sh

h1-h3# disable_offloading.sh, disable_ipv6.sh
```

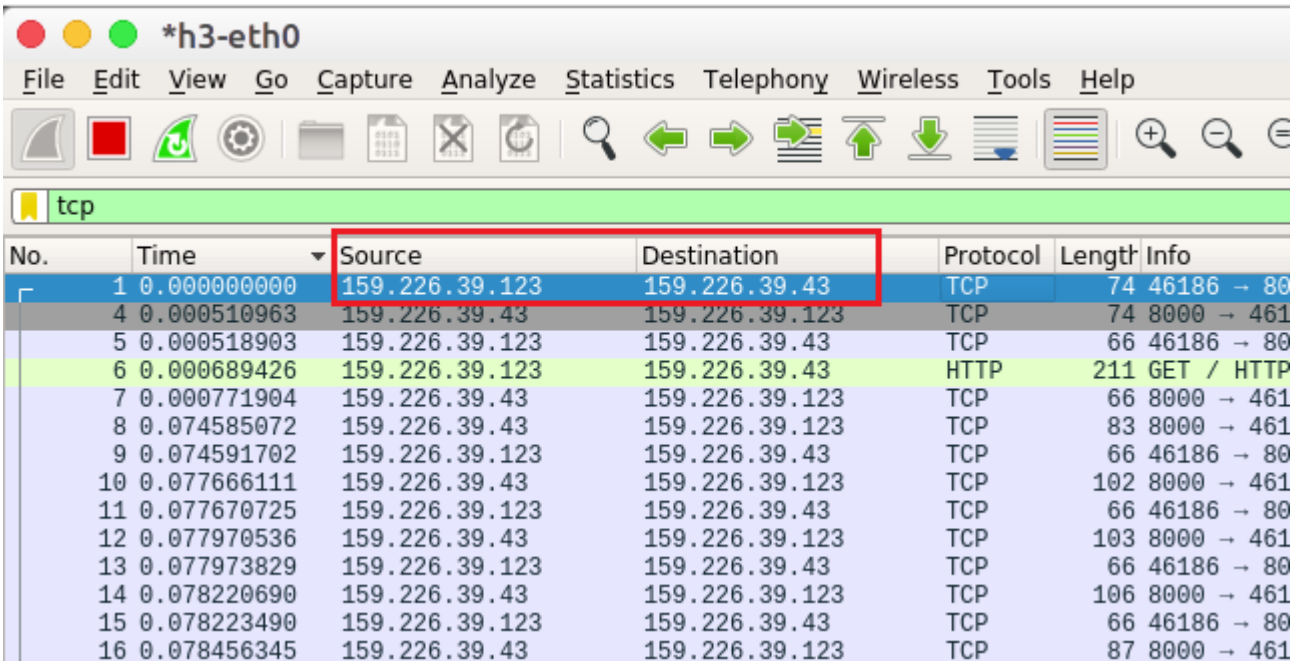
- 在n1上运行nat程序： `n1# ./nat`
- 在h1, h2上分别运行HTTP Server： `h1/h2# python ./http_server.py`
- 在h3上分别请求h1, h2页面

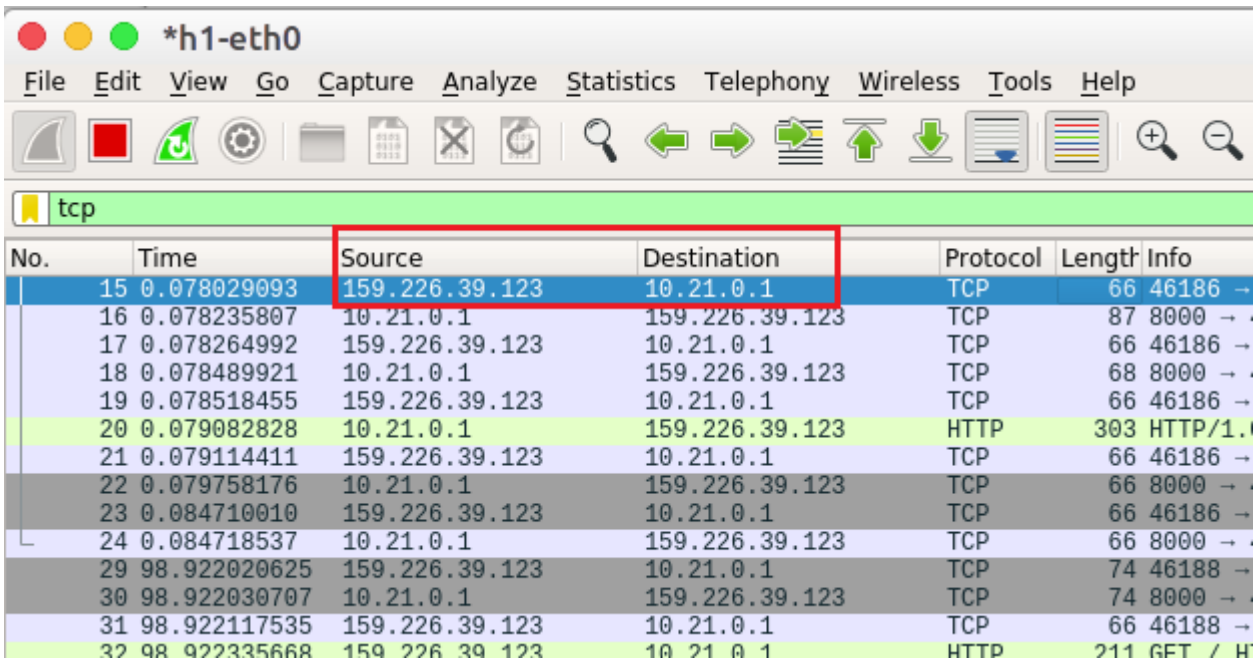
```
h3# wget http://159.226.39.43:8000

h3# wget http://159.226.39.43:8001
```

获取网页



抓包结果





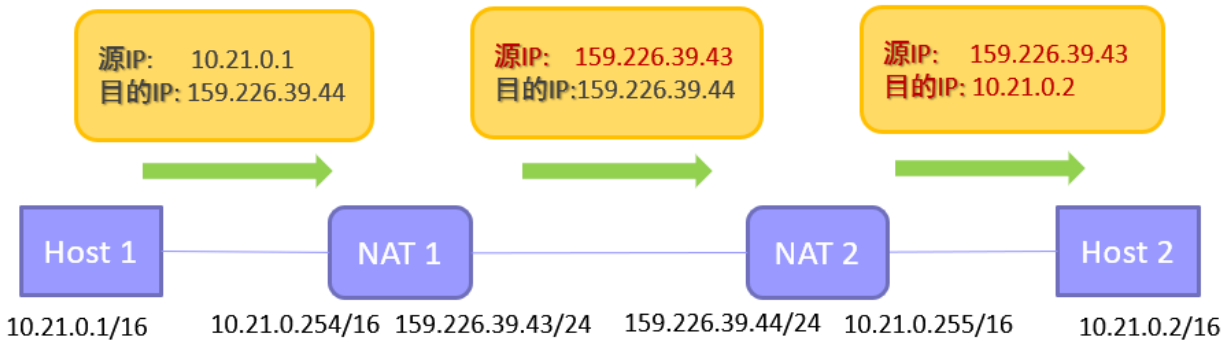
结果分析

外网h3经过n1的正确转换，成功访问了内网h1和h2的HTTP服务。

3. 构造一个包含两个nat的拓扑

验证方法

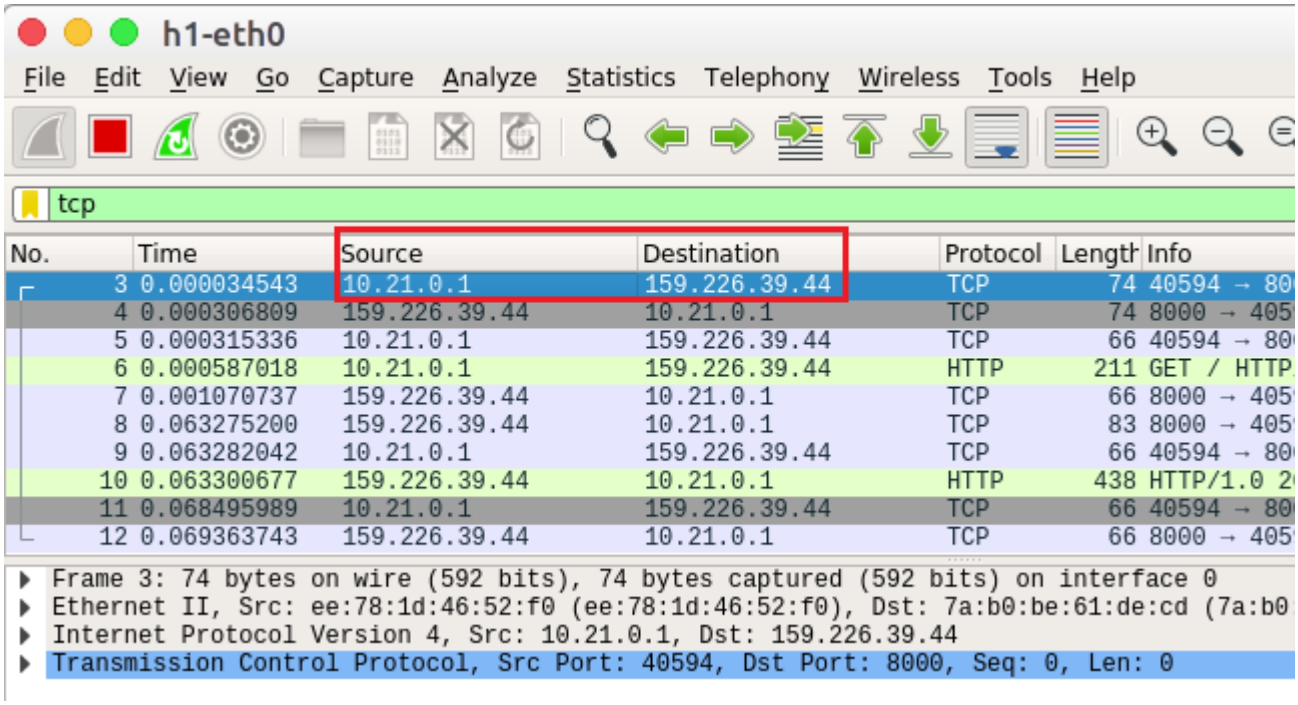
- 结点连接方式：`h1 <-> n1 <-> n2 <-> h2`
- 节点n1作为SNAT， n2作为DNAT，主机h2提供HTTP服务，主机h1穿过两个nat连接到h2并获取相应页面
- 代码见文件 `two_nat_topo.py`，H1和H2的配置文件分别为`nat1_config.txt`和`nat2_config.txt`。
- 构造的拓扑原理图如下图所示：

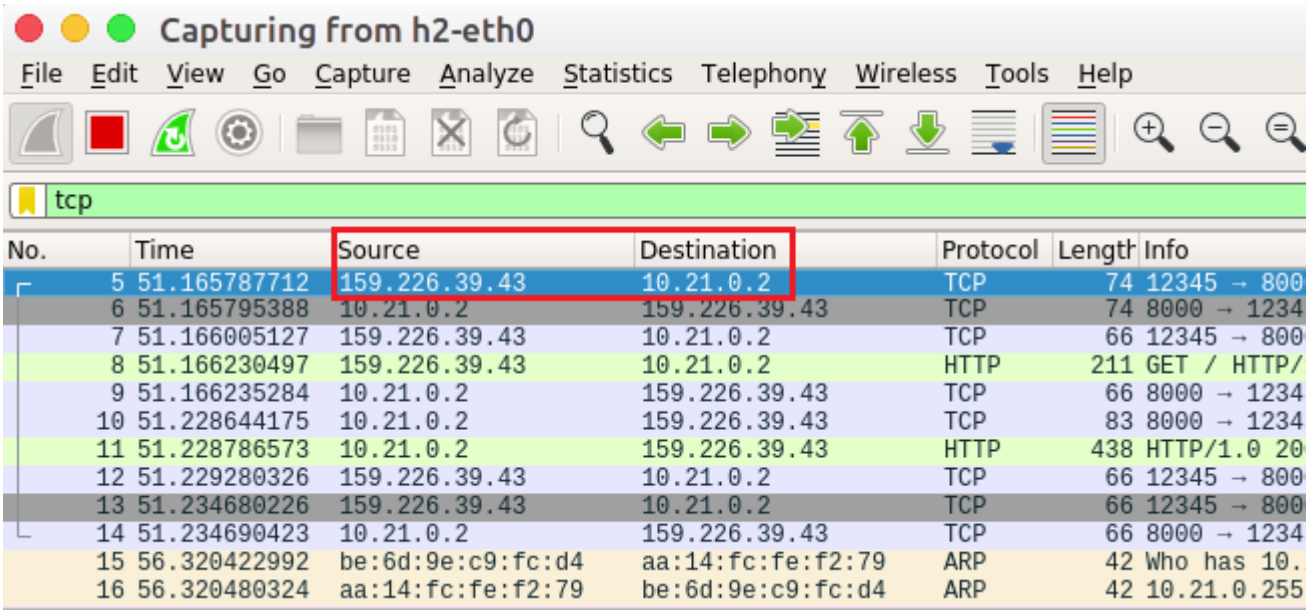


获取网页

```
"Node: h1"
root@zy-VB:09-nat# wget http://159.226.39.44:8000
--2019-05-02 20:09:16--  http://159.226.39.44:8000/
Connecting to 159.226.39.44:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 236 [text/html]
Saving to: 'index.html'

index.html          100%[===================>]     236  --.-KB/s    in 0s

2019-05-02 20:09:16 (32.2 MB/s) - 'index.html' saved [236/236]

root@zy-VB:09-nat# cat index.html

<!doctype html>
<html>
        <head> <meta charset="utf-8">
                <title>Network IP Address</title>
        </head>
        <body>
            My network IP is: addr:10.21.0.2
addr:127.0.0.1

            Remote IP is: 159.226.39.43
        </body>
</html>
```

抓包结果





结果分析

由以上结果知，NAT1和NAT2正确完成了地址转换，h1成功访问了h2的HTTP服务。

实验记录

注意在运行自己构造的NAT拓扑之前，需要修改`nat.c`中`parse_config`函数中读取`internal_iface`和`external_iface`的部分代码，以正确读取NAT2的配置文件。