学号: 2111252　　姓名：李佳豪

https://github.com/FondH/NetTech/

NPCAP获取IP与MAC地址对应

0、关于检查时候的问题

1. ARP报文是根据目标IP，在本网段内寻找对应的网卡地址的，但是我怎么知道本网段有这个ip呢

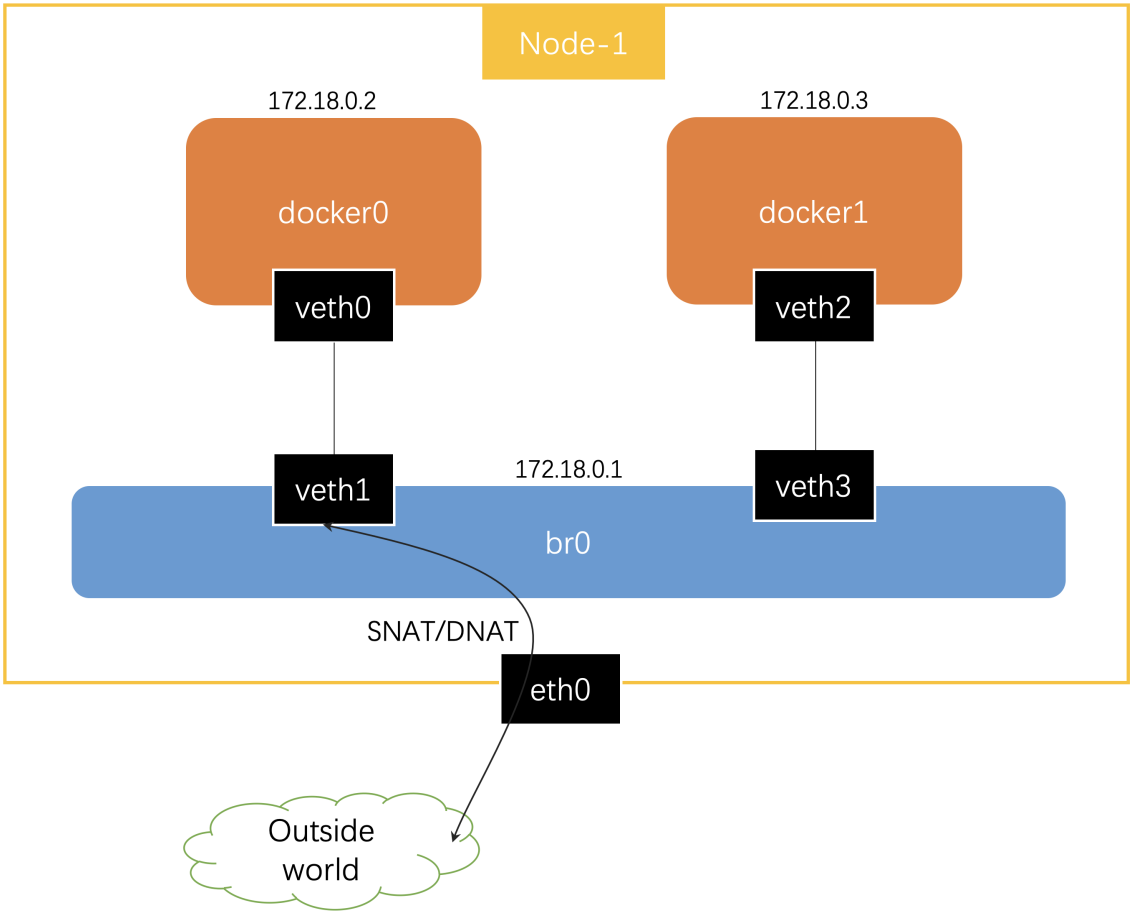
> 首先声明ARP工作的上下文，这是我之前没有搞懂的，ARP是在路由层决定下一跳IP后，在链路层通过ARP搜索下一跳IP地址对应的MAC地址，而这次实验我已知一个虚拟机或者别人电脑的ip去寻找它的MAC地址本身就和它实际的作用有出入；
>
> 因此我不需要知道本网段有哪些IP，（如果需要知道，我现在学习到的是Nmap的主机探测技术，若在同一子网内ARP也确实可以确定哪些IP地址被分配给了网络上的设备；用更广泛更经典的是利用ICMP，发送ICMP Echo请求到目标IP地址，如果目标活跃，它通常会回应一个ICMP Echo响应。）；而为什么路由层、路由表能得知有这个下一跳ip，和动态路由协议开始讲...

2. 我编程中给虚拟机发送的ARP包，经过的是几层网络

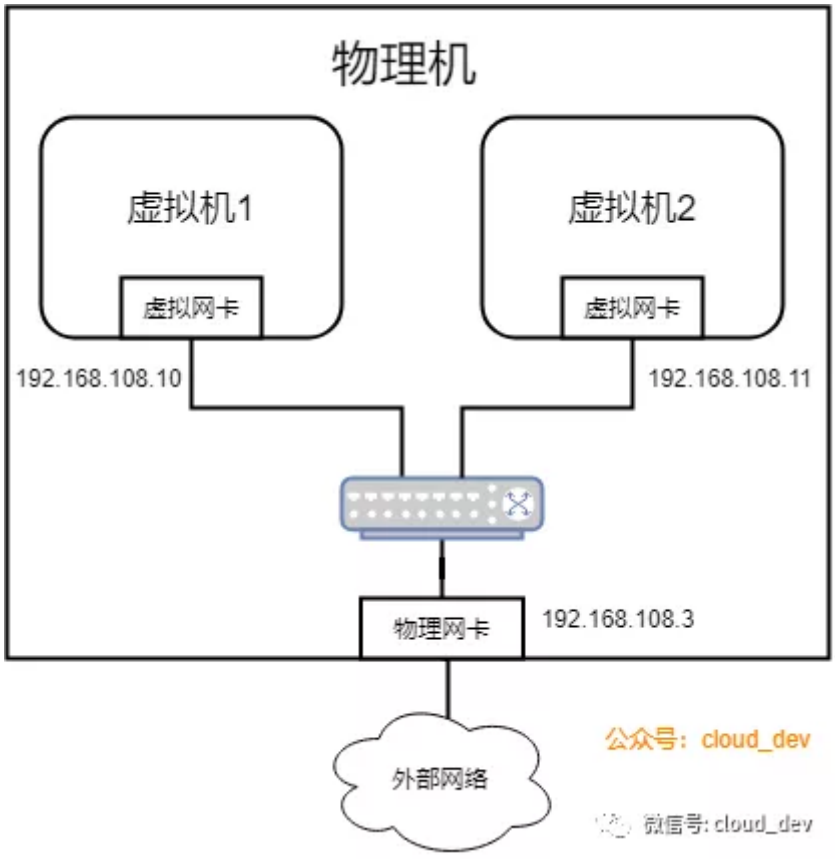
> ~~检查的时候我对ARP的上下文理解有误，不怎么理解2层、3层网络编程层面的意义，蒙了个3层，因为我知道虚拟机的nat模式下，和主机不在一个子网...~~但是ARP如上所说它工作在数据链路层，是一定不穿越网络层的，至于为什么我给我的虚拟机发ARP能得到正确的回复，就是vmware自己的路由的原因

3. 虚拟机 docker的网络拓扑差异

> 注:docker只使用过它编程，没有实际网络经验,**理解应该不太到位。**而且我实验也没有用docker实验...
>
> 主要关注了docker的网桥和虚拟机的网桥
>
> **docker单机的桥接下**，不同容器都有自己的虚拟网络接口（veth pair），连接到Docker虚拟网络桥，成对的veth对隔离了不同的网络空间，数据从一端--容器发送，会在另一端--另一个网络命名空间接收，而这个虚拟网桥，则作为二层网络的交换机，将上面的数据进行广播或者单播等操作，使得不同网络空间之间通信。



vmware的网桥，模拟了真实交换机(和docker的网桥类似)和真实网卡驱动和网卡接口（vNIC）。虚拟机的vNIC连接到虚拟桥接器，虚拟桥接器再连接到物理主机的网络接口卡（NIC）。



桥接模式下拓扑结构的差别，主要在他们和网桥的连接方面，另外虚拟机模拟了完整的传统的网络模型，包括网络协议栈到NIC，Docker桥接利用了容器的网络隔离手段，Docker虚拟网络桥出来的流量进入宿主机的网络协议栈直接利用主机的网络栈，进行后面的操作...

参考文章:

https://chenxiaoweii.github.io/2017/04/21/trace-linux-network-stack/

https://www.cnblogs.com/bakari/p/10592421.html#03-nat

https://www.cnblogs.com/heian99/p/12585724.html

关于docker实验

https://blog.crazytaxii.com/posts/docker_network_bridge_lab/

https://github.com/mz1999/blog/blob/master/docs/docker-network-bridge.md

1、关于ARP背景

ARP报文用于在网络上将IP地址解析为MAC地址，报文内容如下

0x0001	**硬件类型 (以太网)**
0x0800	协议类型 (IPv4)
6	硬件地址长度
4	协议地址长度
0x0001	操作码 (请求)
01:23:45:67:89:AB	发送方硬件地址
192.168.1.2	发送方协议地址
00:00:00:00:00:00	目标硬件地址
192.168.1.1	目标协议地址

1. 发送

   以太网帧方面：`DestMac` 设置为全1，意思在**本网段广播**；Source则是本网卡物理地址。

   Type 则是ARP(0x806)，意思网络层协议按照ARP对应结构解析。

   ARP报文方面：注意`Opcode`为1 表示发送；`Target Hardware Address`为0即可

```
Ethernet II, Src: IETF-VRRP-VRID_0d (00:00:5e:00:01:0d), Dst: Broadcast (ff:ff:ff:ff:ff:ff
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: IETF-VRRP-VRID_0d (00:00:5e:00:01:0d)
    Type: ARP (0x0806)
    Trailer: 00000000000000000000000000000000
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: IETF-VRRP-VRID_0d (00:00:5e:00:01:0d)
    Sender IP address: 0.0.0.0
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 255.255.255.255
```

2. 回复

   以太网帧方面：DestMac 不再时全网段广播；Destination则是本网卡物理地址。

   Type 则是ARP(0x806)

   ARP报文方面：注意Opcode为2 表示回复;

```
Ethernet II, Src: VMware_bd:d8:4e (00:0c:29:bd:d8:4e), Dst: VMware_c0:00:08 (00:50:56:c0:0
  > Destination: VMware_c0:00:08 (00:50:56:c0:00:08)
  > Source: VMware_bd:d8:4e (00:0c:29:bd:d8:4e)
    Type: ARP (0x0806)
    Padding: 00000000000000000000000000000000000000
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: VMware_bd:d8:4e (00:0c:29:bd:d8:4e)
    Sender IP address: 192.168.137.130
    Target MAC address: VMware_c0:00:08 (00:50:56:c0:00:08)
    Target IP address: 192.168.137.1
[Duplicate IP address detected for 192.168.137.130 (00:0c:29:bd:d8:4e) - also in use by 00
```

2、NPCAP编程设计

首先NPCAP的发包是需要完成由应用层、传输层、网络层和数据链路层以太网帧的封装。再交付物理层传输，而ARP协议的发送可以视为网络层的协议。因此需要我们手动完成ARP协议内容的封装以及链路层帧的封装。

1. 数据结构定义 -- 需要定义三个struct，以太网帧、ARP报文以及两者打包的结构。
   1. eth_header.InitArp()函数，初始化ARP报文的以太网帧内容，type是固定值、DestMac是广播域、Source是本网卡的MAC地址。
   2. arp_hdr.set_srd_dst()函数则初始化ARP报文内的mac、ip等内容，注意大小端转换。

```
struct eth_header {
```

```
    uint8_t  dst_mac[ETH_HW_ADDR_LEN];
    uint8_t  src_mac[ETH_HW_ADDR_LEN];
    uint16_t eth_type;

    void InitArp(const u_char* src_mac_addr){
        memcpy(src_mac, src_mac_addr,ETH_HW_ADDR_LEN);
        memset(dst_mac, 0xff, ETH_HW_ADDR_LEN);
        eth_type = htons(ETH_TYPE_ARP);
    }
};

struct arp_hdr {
    uint16_t hw_type;
    uint16_t proto_type;
    uint8_t hw_addr_len;
    uint8_t proto_addr_len;
    uint16_t opcode;
    uint8_t sender_hw_addr[ETH_HW_ADDR_LEN];
    uint8_t sender_proto_addr[IP_ADDR_LEN];
    uint8_t target_hw_addr[ETH_HW_ADDR_LEN];
    uint8_t target_proto_addr[IP_ADDR_LEN];
    arp_hdr() {
        hw_type = htons(HW_TYPE);
        proto_type = htons(PROTO_IP);
        hw_addr_len = ETH_HW_ADDR_LEN;
        proto_addr_len = IP_ADDR_LEN;
        opcode = htons(OP_REQ);
    }
    void set_srd_dst(const u_char* src_hw_addr, const u_char* src_pro_addr, const u_char*
dst_hw_addr, const u_char* dst_pro_addr)
    {
        memcpy(sender_hw_addr, src_hw_addr, ETH_HW_ADDR_LEN);
        memcpy(sender_proto_addr, src_pro_addr, IP_ADDR_LEN);
        memcpy(target_hw_addr, dst_hw_addr, ETH_HW_ADDR_LEN);
        memcpy(target_proto_addr, dst_pro_addr, IP_ADDR_LEN);
    }
};
struct arp_package {
    eth_header eth_head;
    arp_hdr arp_head;
};
```

2. ARP包的发送

1、获得本机MAC地址、网卡Device的handle：

Mac查看电脑网络适配器设置，硬编码到程序内;

```
#define DEFAULT_PC_MAC "F0-77-C3-16-85-5F"
#define VM_PC_MAC  "00-50-56-C0-00-08"
#define VM_PC_IP   "192.168.137.1"
#define DEFAULT_VM_MAC  "00-0C-29-BD-D8-4E"
#define DEFAULT_VM_IP   "192.168.137.130"
```

2、获得对应device的handle，这里使用上次npcap封装的函数即可

```
 // 获取网络接口
pcap_if_t* alldevs = NULL;
pcap_t* adhandle = NULL;
get_device_list(&alldevs, errbuf, 0);
open_device(&adhandle, VMnet8_NUM, alldevs, errbuf);
pcap_freealldevs(alldevs);
```

3、打包Packet

Local_MAC、Local_IP是我们本地属性，为了初始化定义的的结构体，传入对应大小的u_char数组；target_ip即为欲解析的ip字符串，同样存入u_char内初始化结构体

```
u_char PCMac[ETH_HW_ADDR_LEN];
u_char PCIP[IP_ADDR_LEN];
u_char DstMac[ETH_HW_ADDR_LEN] = { 0 };
u_char DstIP[IP_ADDR_LEN];

sscanf_s(Local_MAC.c_str(), "%hhx-%hhx-%hhx-%hhx-%hhx-%hhx",
             &PCMac[0], &PCMac[1], &PCMac[2],
             &PCMac[3], &PCMac[4], &PCMac[5]);
sscanf_s(Local_IP.c_str(), "%hhu.%hhu.%hhu.%hhu", &PCIP[0], &PCIP[1], &PCIP[2],
&PCIP[3]);

memset(DstMac, 0, ETH_HW_ADDR_LEN);
sscanf_s(target_ip.c_str(), "%hhu.%hhu.%hhu.%hhu", &DstIP[0], &DstIP[1], &DstIP[2],
&DstIP[3]);
```

下一步封装包、使用pcap_sendpacket发送

```
/*Init 报文*/
arp_package arp_req;
arp_req.eth_head.InitArp(PCMac);
arp_req.arp_head.set_srd_dst(PCMac, PCIP, DstMac, DstIP);
pcap_sendpacket(adhandle, (unsigned char*)&arp_req, sizeof(arp_req));
```

3. 捕获、解析ARP包

过滤

这里考虑超时失败、首先**验证**以太帧的`eth_type`是ARP对应的值，**验证**ARP报文是 `Opcode` 是2表示回复，**验证**ARP报文内的`Target_MAC`,`Target_IP`是我们发送时报文内的`Source_MAC`,`Source_IP`，通过验证则得到对应`MAC`和`IP`的匹配关系

解析

根据ARP报文解析即可

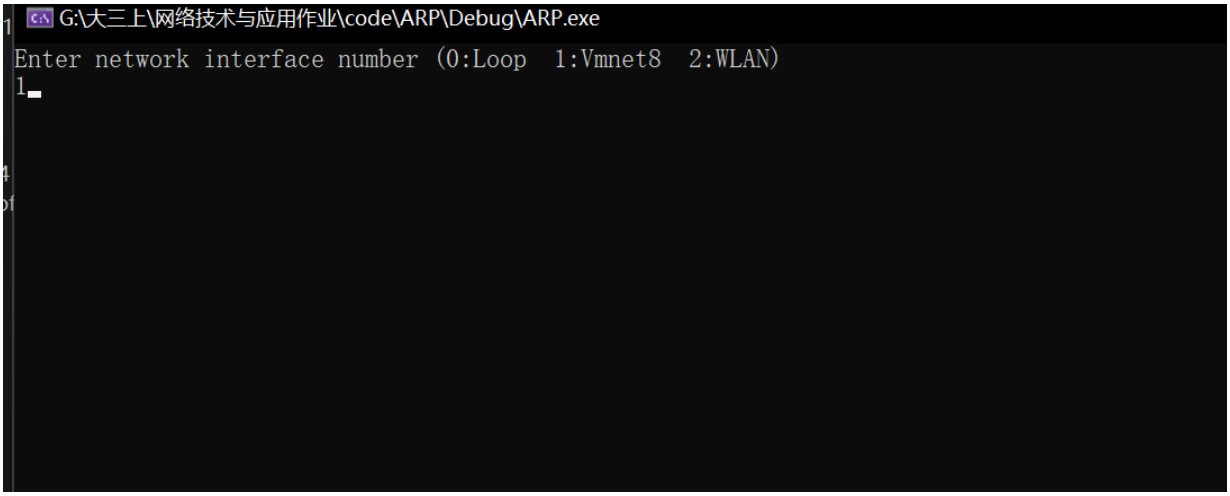
```
clock_t start = clock();
cout << "\n---------------------- wait ------------------------ \n";
while ((res = pcap_next_ex(adhandle, &header, &pkt_data)) >= 0) {
        if (res < 1)
            continue;
    // 首先设置一个超时时间5s，超时退出
    clock_t end = clock();
    if (end - start > MaxRtt) {
```

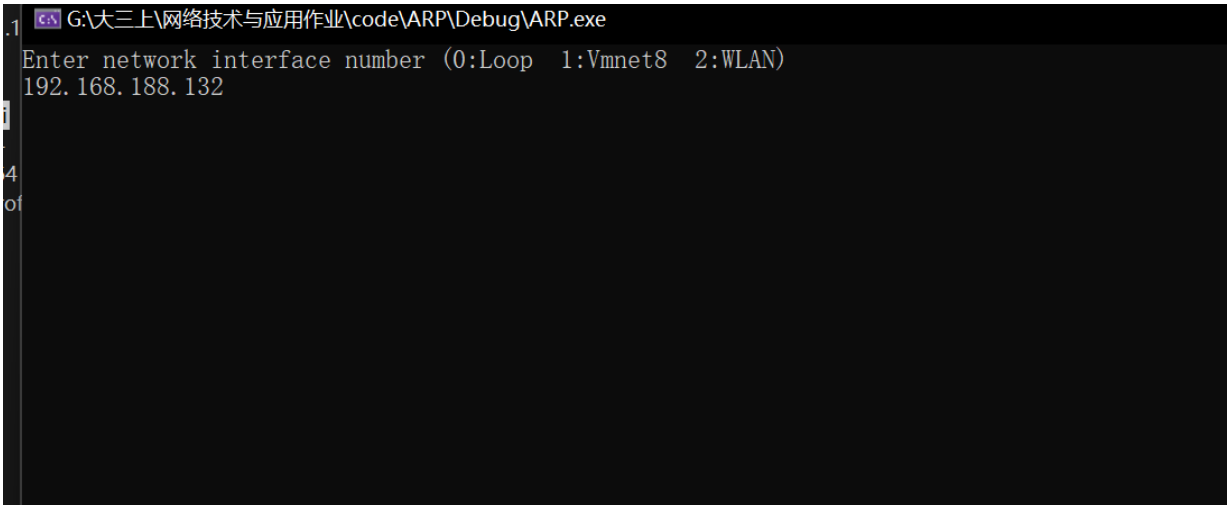
```
            cout << "Time out" << endl;
            break;
        }
        eth_header* eth = (eth_header*)pkt_data;
        arp_hdr* arp = (arp_hdr*)(pkt_data + sizeof(eth_header));

        //验证是ARP 且是ARP报文是 类型 -- 2
        if (!(ntohs(eth->eth_type) == ETH_TYPE_ARP) || !(ntohs(arp->opcode) == OP_REP))
            continue;

        //验证报文段的Target_MAC,Target_IP是我们发送方的
        if (!memcmp(arp->target_hw_addr, PCMac, ETH_HW_ADDR_LEN) && !memcmp(arp-
>target_proto_addr, PCIP, IP_ADDR_LEN)) {
            cout << "Dst MAC: ";
            for (int i = 0; i < ETH_HW_ADDR_LEN; ++i) {
               cout << setfill('0') << setw(2) << hex << static_cast<int>(arp-
>sender_hw_addr[i]);
            if (i < ETH_HW_ADDR_LEN - 1) cout << "-";
                   else cout << "\n";
               }
            cout << dec;
            cout << "Dst IP ADDR: ";
            for (int i = 0; i < IP_ADDR_LEN; ++i) {
                cout << static_cast<int>(arp->sender_proto_addr[i]);
                if (i < IP_ADDR_LEN - 1) cout << ".";
                else cout << "\n";
            }

            end = clock();
            double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
            printf("RTT: %f seconds\n", time_taken);
            cout << "\n--------- Arp Response captured ---------" << endl;
            break;
         }


}
```

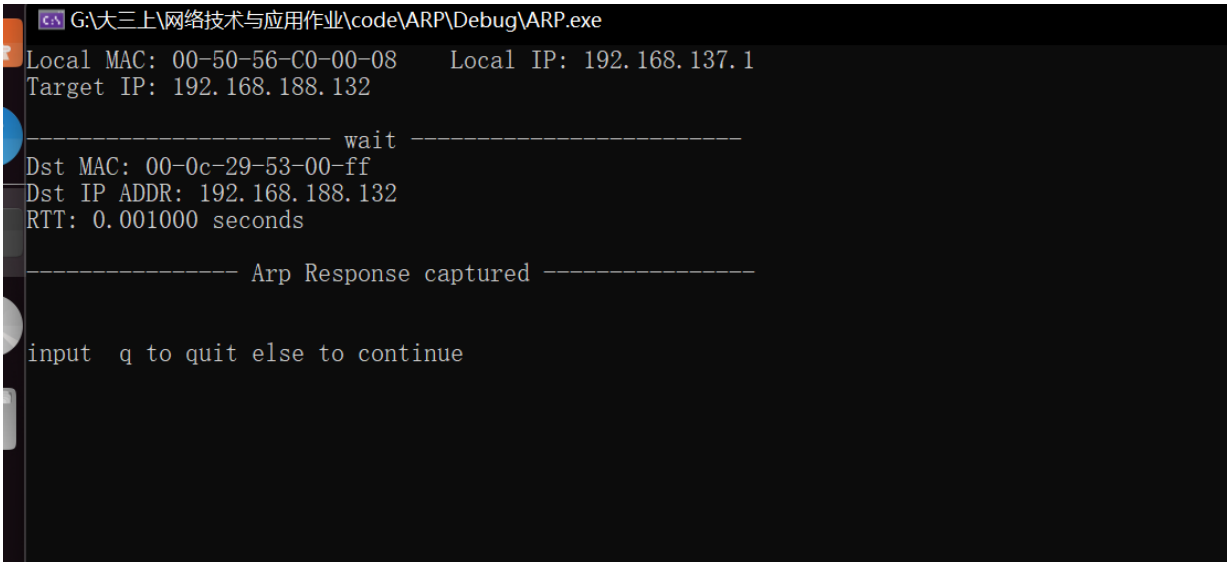
4.程序流程

输入捕获的网卡、然后输入目的IP地址，之后等待返回即可；程序可能返回超时或得到结果，输入q结束程序，否则继续输入目的IP地址。

结果样例如下：







github链接

https://github.com/FondH/NetTech/