**同济大学计算机系**

**系统安全评估实验报告**

****

**软件开发报告：网络扫描器的设计与开发**

**姓 名**

**同组成员**

**专 业**

**一、系统需求分析**

1. **功能需求**
   1. 主机扫描

本软件需实现简单的主机扫描功能，扫描范围针对某一网段进行，用户指定某一网段，软件进行扫描，最终输出扫描到的开放主机的IP地址。

* 1. 端口扫描

软件需实现端口扫描功能，即可更细致的扫描某一IP地址的开放端口，用户指定要扫描的IP地址以及端口号的范围（起始端口号和结束端口号），软件进行扫描，最终输出指定IP地址在端口范围内全部的开放端口号。

* 1. 数据包捕获

实现数据包的捕获，分析捕获的数据包并输出其中包含的信息，包括以太网头部、IP头部、UDP头部、TCP头部信息的分析及显示。

1. **非功能性需求**
   1. 易用性

本软件需要通过图形化界面来实现软件的易用性，要求有良好的输入输出提示，用户可通过图形化界面清晰简单的实现交互，完成自己想要的扫描。

* 1. 性能

网络扫描过程是相对时间较长，若扫描速度过慢十分影响使用。本软件要求网络扫描具有一定的效率，可通过多线程等方式来实现。

1. **环境声明**

运行与调试环境相同，均如下：

3.1 操作系统

Windows 10

3.2 软件框架

Qt Creator 4.11.1 Based on Qt 5.14.1 (MSVC 2017, 32bit)

Visual Studio 2019 (x86 Debug模式)

3.3 库

ws2\_32.lib

iphlpapi.lib

wpcap.lib

1. **初始状态**

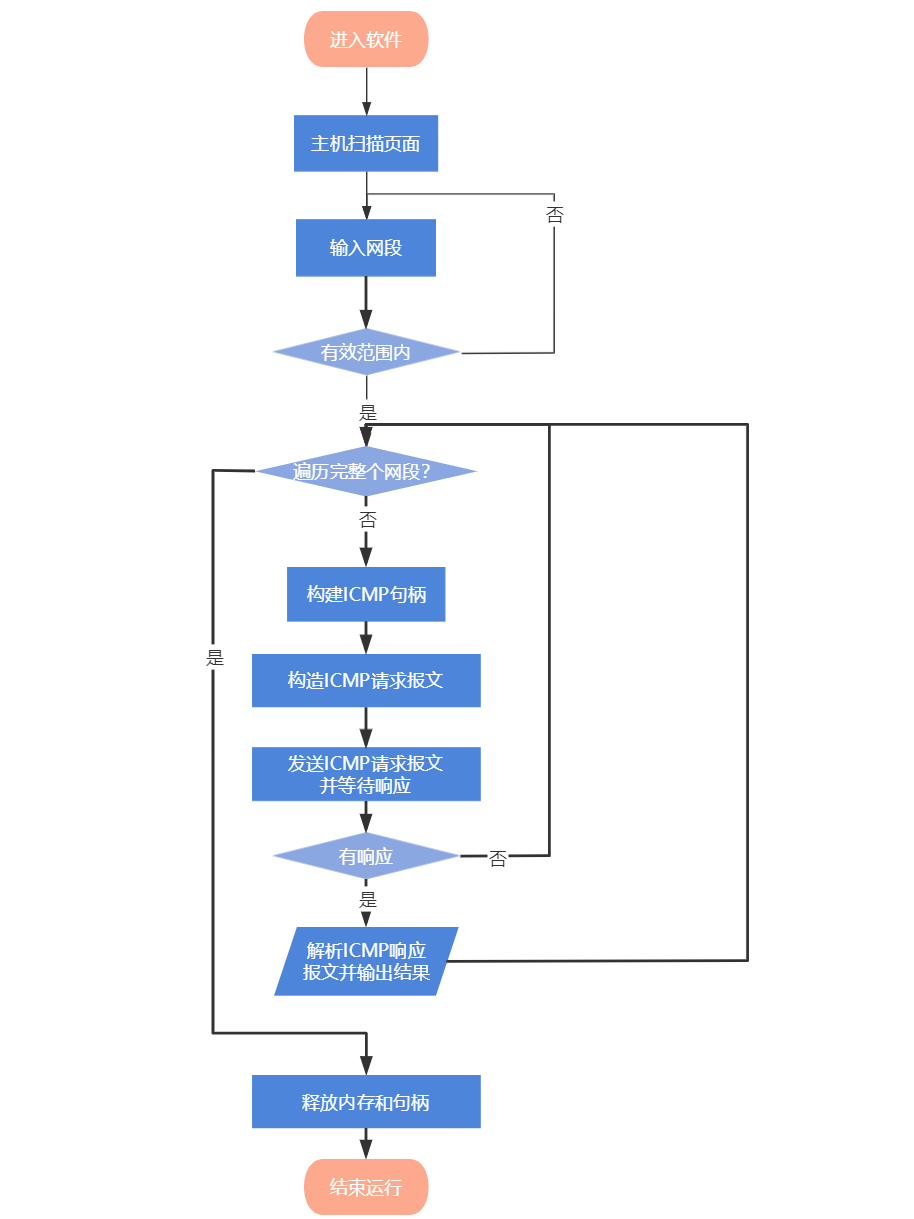
无特殊设置，在未指定状态下，主机扫描及端口扫描功能下所有需指定输入的部分均为0。也无特殊权限设置，本软件扫描暂不涉及管理员权限的功能。

1. **各组件逻辑调用关系**

本软件主要实现三个主要部件，用户界面、主机扫描和端口扫描。用户界面主要实现条件输入、交互和功能调用：在用户界面触发主机扫描条件时调用主机扫描部分；用户界面触发端口扫描条件时调用端口扫描部分。

**二、软件功能设计**

1. **主机扫描**
   1. 流程图



* 1. 功能详细说明

输入：输入xx.xx.xx.0/24网段前面xx.xx.xx.部分。

输出：依次输出开放主机的IP地址

错误处理：如输入网段不在合理范围内无法输入，只取到最后一个还在范围内的值。

主要实现方法：本软件主机扫描主要通过ICMP方式实现。

* 1. 相关原理

ICMP（Internet Control Message Protocol）是一种网络协议，常用于检测主机是否可达和测量网络性能等。主机扫描一般使用 ICMP Echo Request 信息包来实现。以下是 ICMP 方法实现主机扫描的原理：

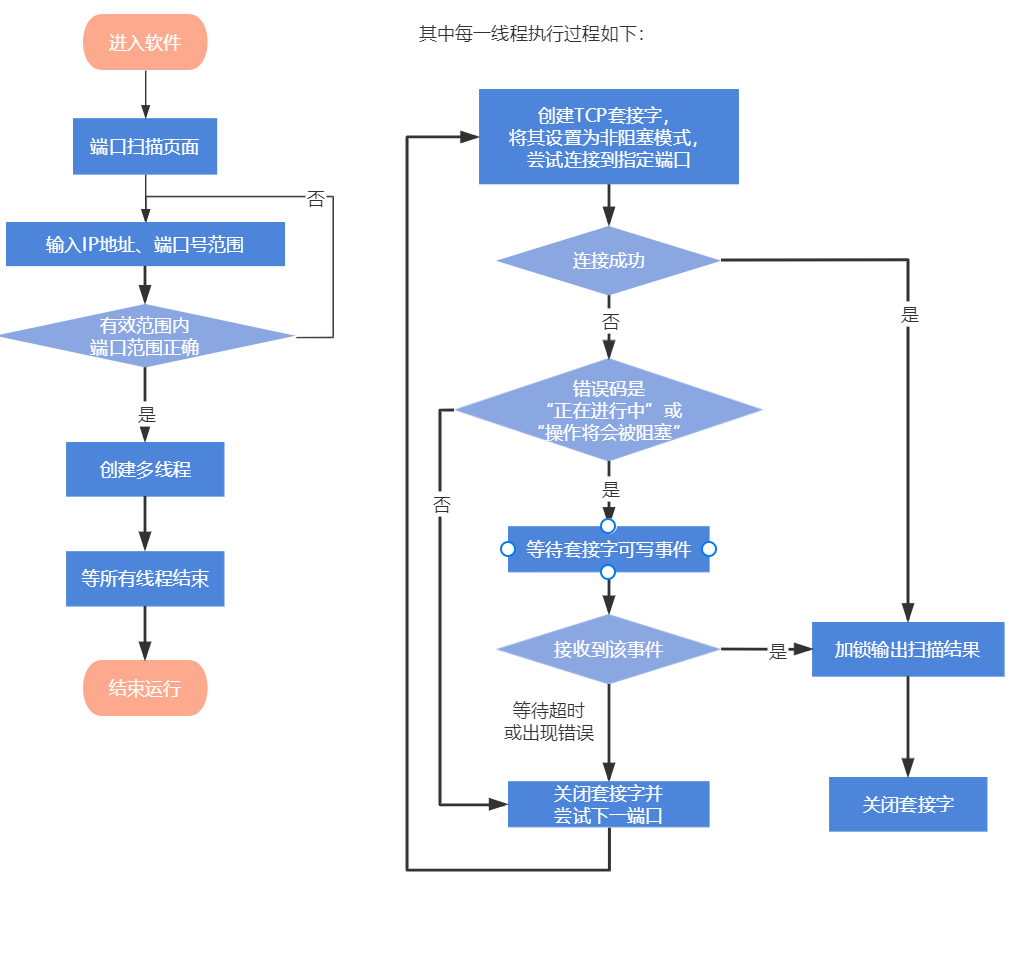
①发送 ICMP Echo Request 包：在主机扫描时，扫描器会向目标网络中的每个 IP 地址发送 ICMP Echo Request 包，请求远程主机返回一个 ICMP Echo Reply 包。

②接收 ICMP Echo Reply 包：如果远程主机处于活动状态并且网络连接正常，则它会响应 ICMP Echo Request 包并返回一个 ICMP Echo Reply 包。扫描器收到 ICMP Echo Reply 包后，就可以确认该 IP 地址是一个活动主机。

③未接收到 ICMP Echo Reply 包：如果扫描器在一定时间内未能收到 ICMP Echo Reply 包，则说明目标主机可能不可达或者网络连接不正常。扫描器会将该 IP 地址标记为不可用。

需要注意的是，某些防火墙和路由器可能会过滤 ICMP 包，从而导致主机扫描失败或不准确。此时，可以尝试使用其他扫描方法，如 TCP 扫描或 UDP 扫描。

1. **网络扫描**
   1. 流程图



2.2 功能详细说明

输入：要扫描的ip地址、起始端口号、结束端口号

输出：依次输出指定IP地址在端口范围内打开的端口

错误处理：不在合理范围内无法输入；若起始端口号大于结束端口号则报错。

主要实现方法：主要通过TCP套接字建立与目标主机的连接实现；通过互斥锁实现资源保护，多线程加快扫描速度。

2.3 相关原理

利用计算机网络通信中的TCP/IP协议，通过TCP套接字建立与目标主机的连接，向目标主机发送特定的数据包（例如SYN数据包），等待目标主机的响应。如果目标主机返回了相应的数据包（例如SYN/ACK数据包），则说明该端口是打开的，如果目标主机返回了RST数据包，则说明该端口是关闭的。基于这种原理，可以通过扫描目标主机的多个端口，来确定目标主机开放的端口和服务，从而判断目标主机的安全性和漏洞情况。

1. **数据包捕获**

3.1功能详细说明

输入：指定网卡编号

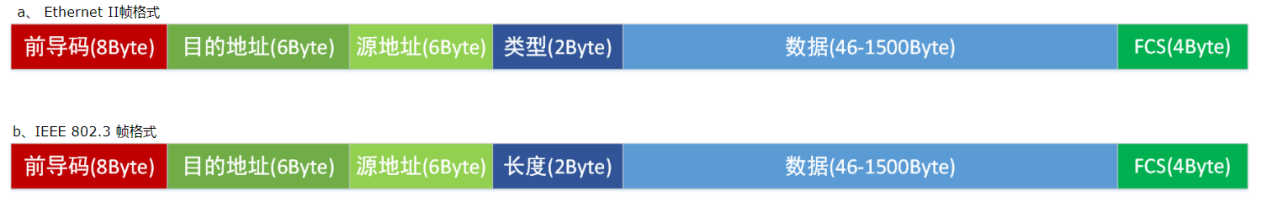
输出：以太网头部信息，IP头部信息，TCP头部信息，UDP头部信息等

具体限制：本功能暂时只能将IPv4协议下的数据包进行更详细的分解

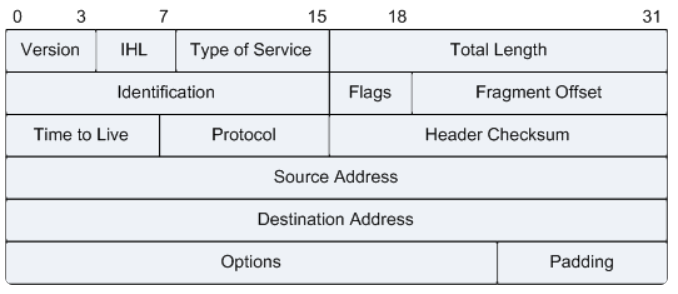
实现方法：通过winpcap进行数据包捕获

3.2 相关原理

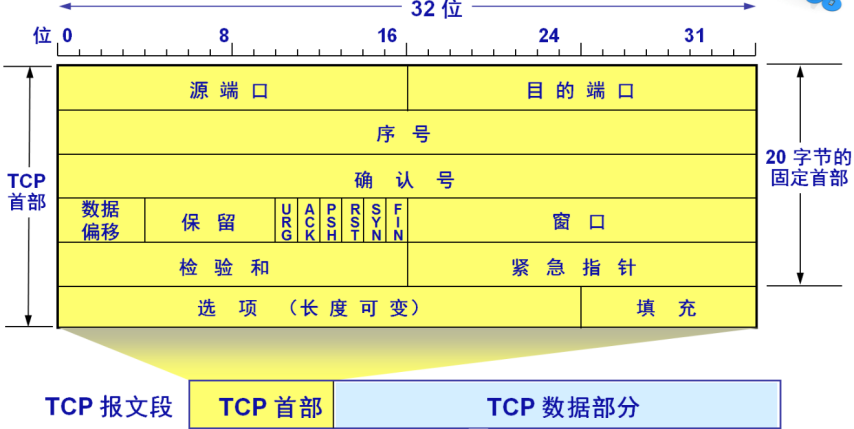
①以太网帧格式



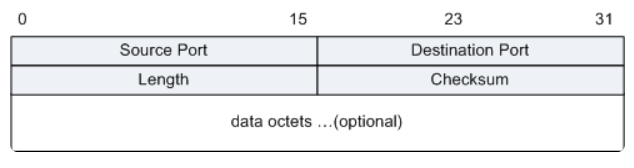
②IP头部



③TCP头部

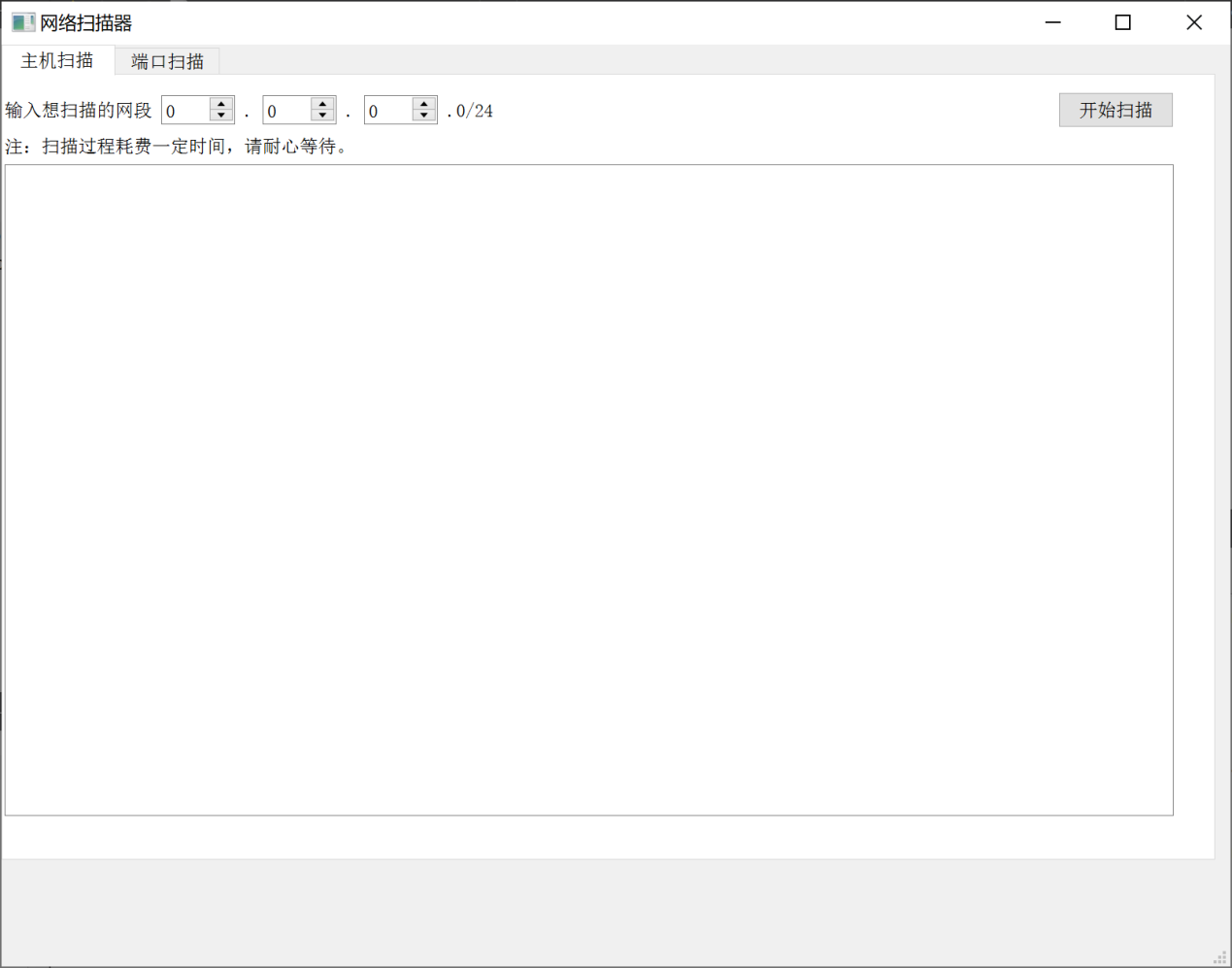


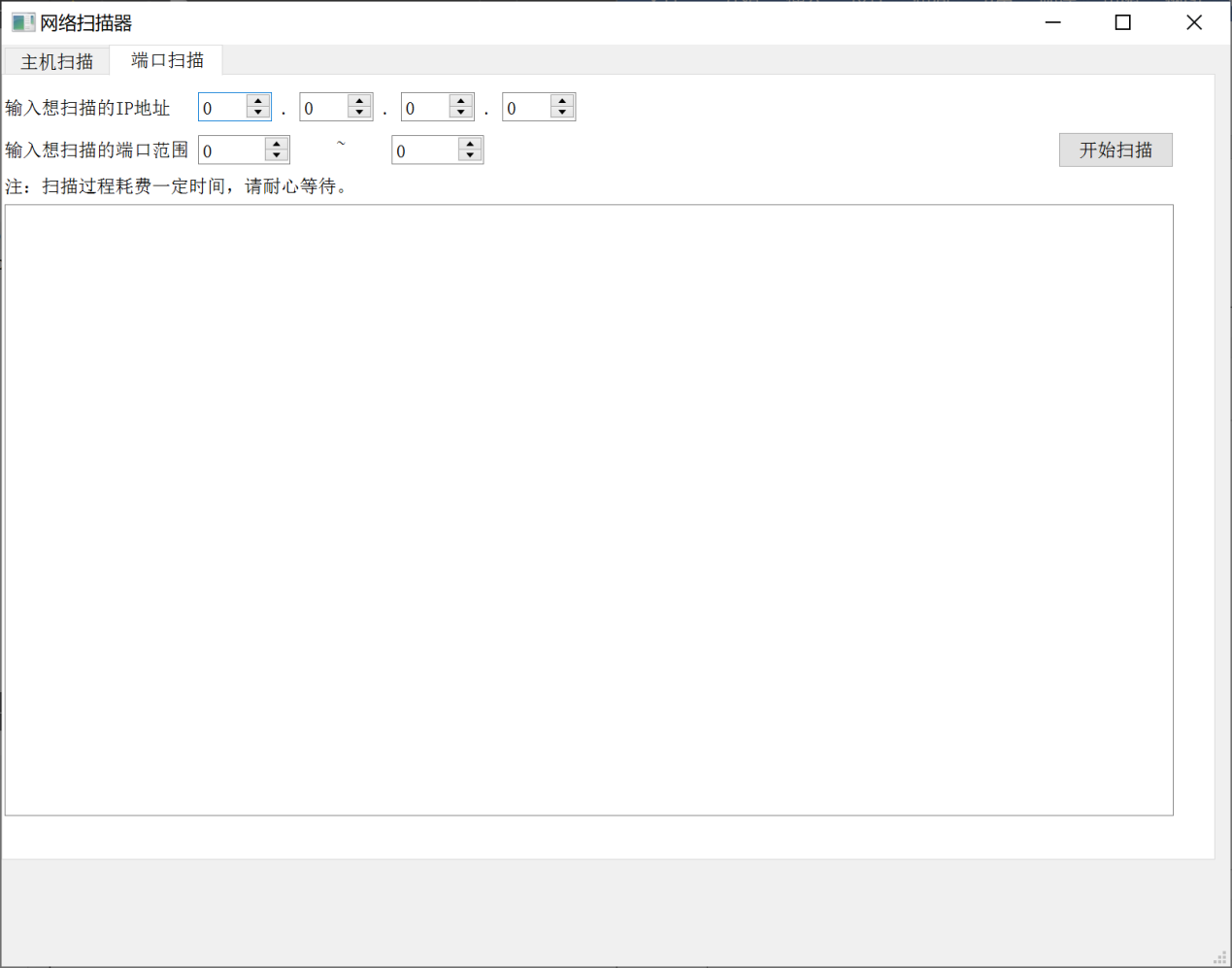
④UDP头部



**三、详细设计**

1. **界面设计**





输入通过spinbox实现，可有效控制范围；输出显示在text browser中。

1. **模块设计**

2.1主机扫描模块

调用函数int MainWindow::hostscan\_icmp(QString s)

S为输入网段x.x.x.形式，实现前端与后端很好的隔离

在函数中实现扫描并输出结果功能独立。

* 1. 端口扫描模块

调用函数int MainWindow::portscan\_multhr(QString s,int pstart,int pend)

S为输入ip地址，为x.x.x.x形式，pstart为起始端口号，pend为结束端口号。

在函数中实现扫描并输出结果功能独立。

* 1. 用户界面模块

将图形化界面的输入处理为其他模块需要的数据格式，并通过调用函数调用其他模块实现功能。输入部分的错误处理在用户界面模块实现。

1. **具体实现**

该部分以部分代码举例

3.1 ICMP方法实现主机扫描

// 遍历所有 IP 地址

string baseIp = "129.211.214.";

for (int i = 1; i <= 255; i++) {

string ip = baseIp + to\_string(i);

cout << "Pinging " << ip << "..." << endl;

// 创建 ICMP 文件句柄

HANDLE hIcmpFile = IcmpCreateFile();

if (hIcmpFile == INVALID\_HANDLE\_VALUE) {

cerr << "Unable to open handle." << endl;

continue;

}

// 构造 ICMP 请求报文

DWORD dwRetVal = 0;

char SendData[] = "Data Buffer";

LPVOID ReplyBuffer = NULL;

DWORD ReplySize = 0;

ReplySize = sizeof(ICMP\_ECHO\_REPLY) + sizeof(SendData);

ReplyBuffer = (VOID\*)malloc(ReplySize);

if (ReplyBuffer == NULL) {

cerr << "Unable to allocate memory." << endl;

continue;

}

// 发送 ICMP 请求报文并等待响应

dwRetVal = IcmpSendEcho(hIcmpFile, inet\_addr(ip.c\_str()), SendData, sizeof(SendData),

NULL, ReplyBuffer, ReplySize, 1000);

if (dwRetVal != 0) {

// 解析 ICMP 响应报文并显示结果

PICMP\_ECHO\_REPLY pEchoReply = (PICMP\_ECHO\_REPLY)ReplyBuffer;

struct in\_addr ReplyAddr;

ReplyAddr.S\_un.S\_addr = pEchoReply->Address;

cout << "Received reply from " << inet\_ntoa(ReplyAddr) << endl;

}

else {

cerr << "Call to IcmpSendEcho failed." << endl;

}

// 释放内存和文件句柄

free(ReplyBuffer);

IcmpCloseHandle(hIcmpFile);

}

3.2 TCP套接字实现端口扫描含互斥锁

void scan\_port(const std::string& ip)

{

while (true)

{

int port = s\_port++; // 获取要扫描的端口号，原子操作

if (port > PORT\_END) // 如果扫描的端口号已经超出了范围，结束循环

break;

sockaddr\_in addr{}; // 创建一个sockaddr\_in类型的变量，表示要连接的地址

addr.sin\_family = AF\_INET; // 设置地址类型为IPv4

addr.sin\_port = htons(port); // 设置端口号

addr.sin\_addr.s\_addr = inet\_addr(ip.c\_str()); // 设置IP地址

SOCKET s = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); // 创建TCP套接字

if (s == INVALID\_SOCKET) // 如果创建套接字失败，继续尝试下一个端口

continue;

u\_long mode = 1;

if (ioctlsocket(s, FIONBIO, &mode) == SOCKET\_ERROR) // 设置套接字为非阻塞模式

{

closesocket(s);

continue;

}

int result = connect(s, (sockaddr\*)&addr, sizeof(addr)); // 连接到指定地址

if (result == SOCKET\_ERROR) // 如果连接失败

{

int error = WSAGetLastError(); // 获取错误码

if (error != WSAEWOULDBLOCK && error != WSAEINPROGRESS) // 如果错误码不是"正在进行中"或"操作将会被阻塞"

{

closesocket(s); // 关闭套接字，继续尝试下一个端口

continue;

}

}

else if (result == 0) // 如果连接成功

{

std::lock\_guard<std::mutex> lock(s\_mutex); // 加锁，保护共享资源

std::cout << "Port " << port << " is open." << std::endl; // 输出扫描结果

}

fd\_set writefds{}; // 创建一个文件描述符集合，用于等待套接字可写事件

FD\_ZERO(&writefds);

FD\_SET(s, &writefds);

timeval timeout{}; // 设置超时时间

timeout.tv\_sec = 1; // 超时时间为1秒

timeout.tv\_usec = 0;

result = select(0, nullptr, &writefds, nullptr, &timeout); // 等待套接字可写事件

if (result == SOCKET\_ERROR) // 如果出现错误

{

closesocket(s); // 关闭套接字，继续尝试下一个端口

continue;

}

else if (result == 0) // 如果超时

{

closesocket(s); // 关闭套接字，继续尝试下一个端口

continue;

}

std::lock\_guard<std::mutex> lock(s\_mutex); // 加锁，保护共享资源

std::cout << "Port " << port << " is open." << std::endl; // 输出扫描结果

closesocket(s); // 关闭套接字

}

}

3.3 多线程

std::thread threads[THREAD\_NUM]; // 创建多个线程

for (int i = 0; i < THREAD\_NUM; ++i)

{

threads[i] = std::thread(scan\_port, ip); // 每个线程调用scan\_port函数进行扫描

}

for (int i = 0; i < THREAD\_NUM; ++i)

{

threads[i].join(); // 等待所有线程结束

}

3.4 接收指定网卡数据包，并输出以太网头部信息

void MonitorAdapter(int nChoose)

{

pcap\_if\_t\* adapters; // 网卡信息链表

char errbuf[PCAP\_ERRBUF\_SIZE]; // 错误信息缓冲区

// 查找系统中所有网卡的信息

if (pcap\_findalldevs\_ex((char\*)(PCAP\_SRC\_IF\_STRING), NULL, &adapters, errbuf) != -1)

{

// 找到指定的网卡

for (int x = 0; x < nChoose - 1; ++x)

adapters = adapters->next;

char errorBuf[PCAP\_ERRBUF\_SIZE];

// 打开指定网卡，设置为混杂模式（即接收所有的数据包），并设置超时时间为1000ms

pcap\_t\* handle = pcap\_open(adapters->name, 65534, 1, PCAP\_OPENFLAG\_PROMISCUOUS, 1000, errorBuf);

if (adapters == NULL)

return;

// 不断读取数据包，并解析其中的以太网头部信息

pcap\_pkthdr\* Packet\_Header; // 数据包头

const u\_char\* Packet\_Data; // 数据本身

int retValue;

while ((retValue = pcap\_next\_ex(handle, &Packet\_Header, &Packet\_Data)) >= 0)

{

if (retValue == 0)

continue;

PrintEtherHeader(Packet\_Data); // 解析以太网头部信息并打印

}

}

}

**四、软件测试（含测试方法，过程以及测试时发现的问题及解决办法）**

**1.测试方法及过程**

**1.1 主机扫描**

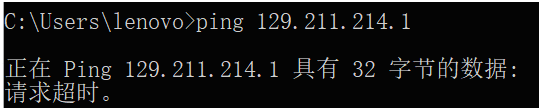
首先在控制台窗口进行某个主机的正确性验证：

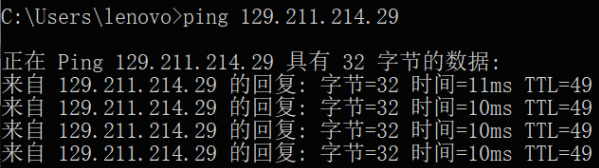
演示样例：扫描129.211.214.xxx网段下的活动主机

输入：修改“string subnet = "129.211.214.";”此行代码实现更改输入

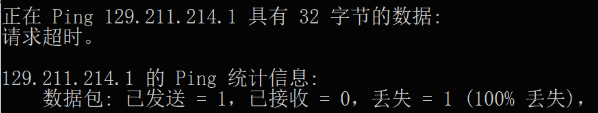
输出：控制台输出

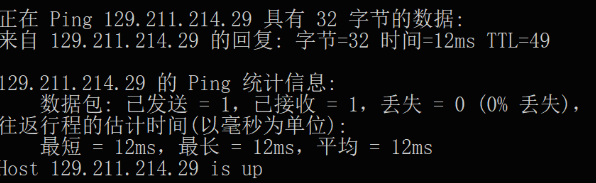
cmd标准结果：





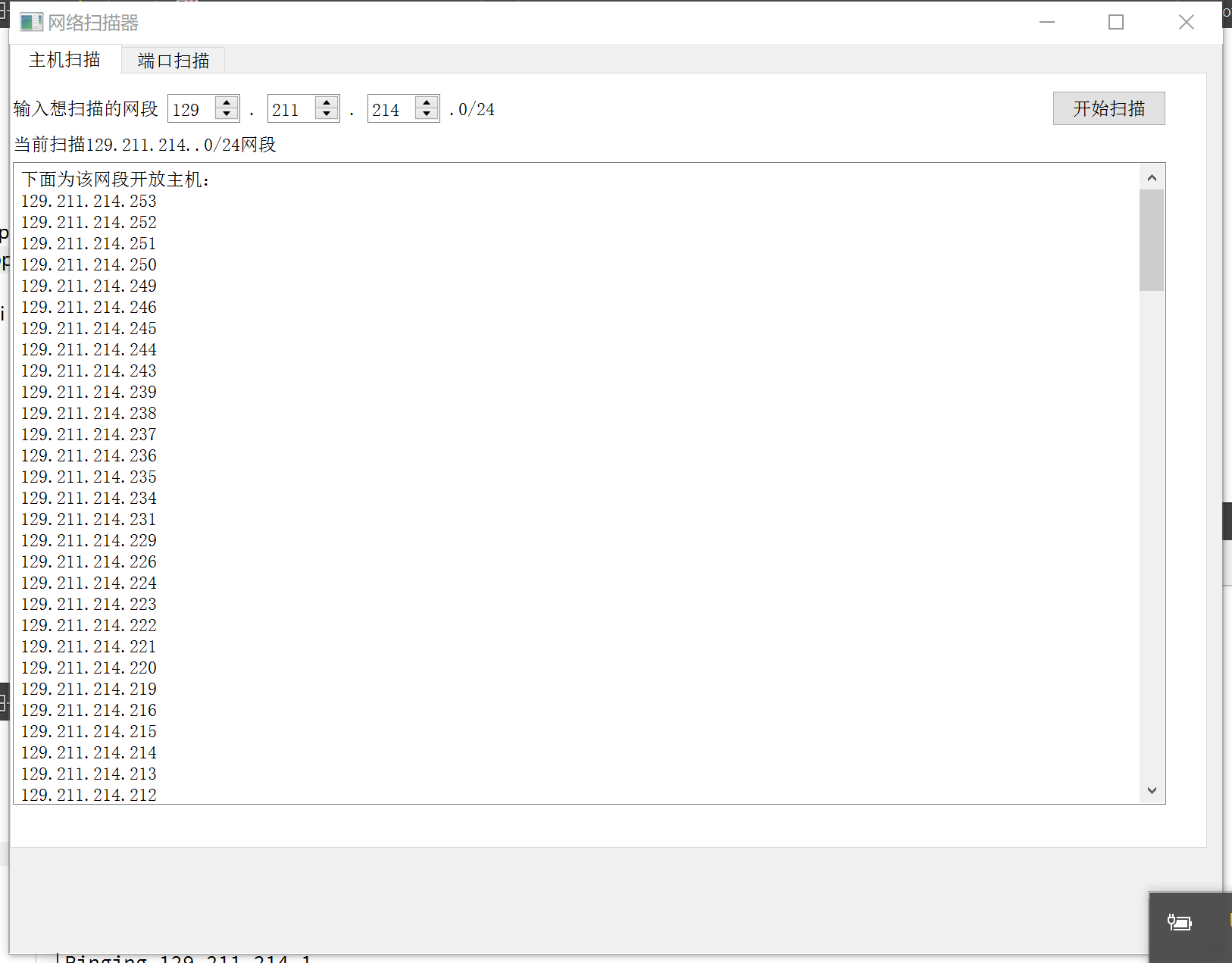
本程序运行结果：





经检验，完全一致。

经过正确性验证的检验之后，在Qt界面实现软件交互界面开发，最终呈现如下：



**1.2 端口扫描**

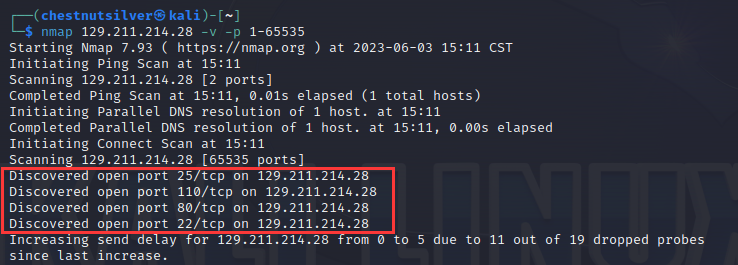
首先在控制台窗口进行端口扫描的正确性验证：

演示样例：扫描 129.211.214.28 活动IP下的端口

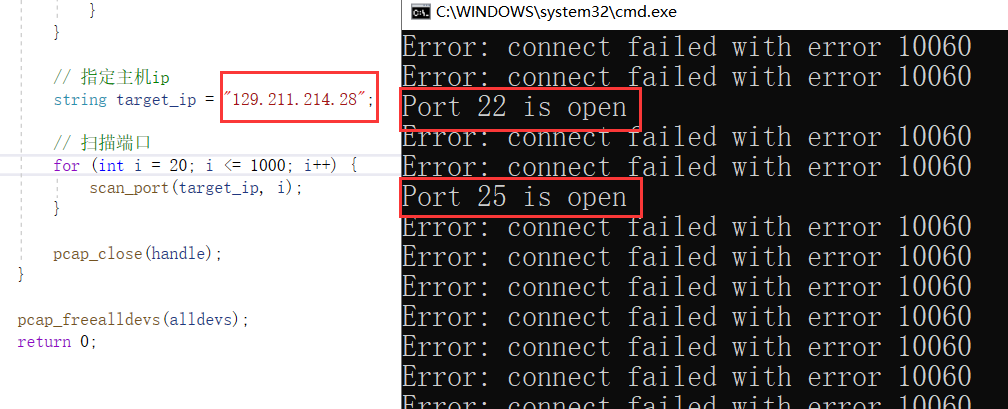
输入：修改代码“指定主机IP、扫描端口”以修改输入，当前代码为扫描129.211.214.28 IP下的20-1000端口

输出：控制台输出

kali标准结果：



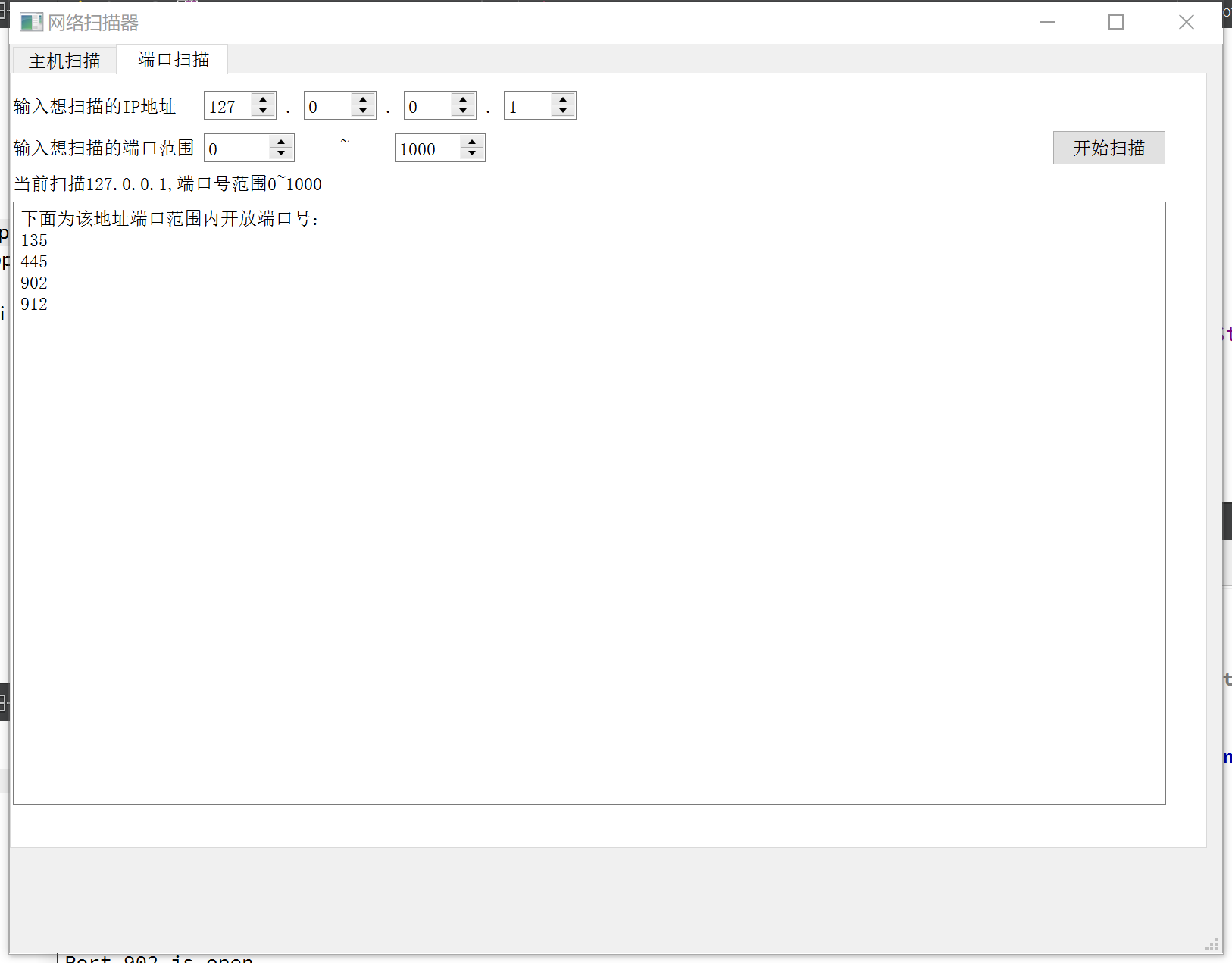
本程序运行结果：



完全一致。

经过正确性验证的检验之后，在Qt界面实现软件交互界面开发，最终呈现如下：

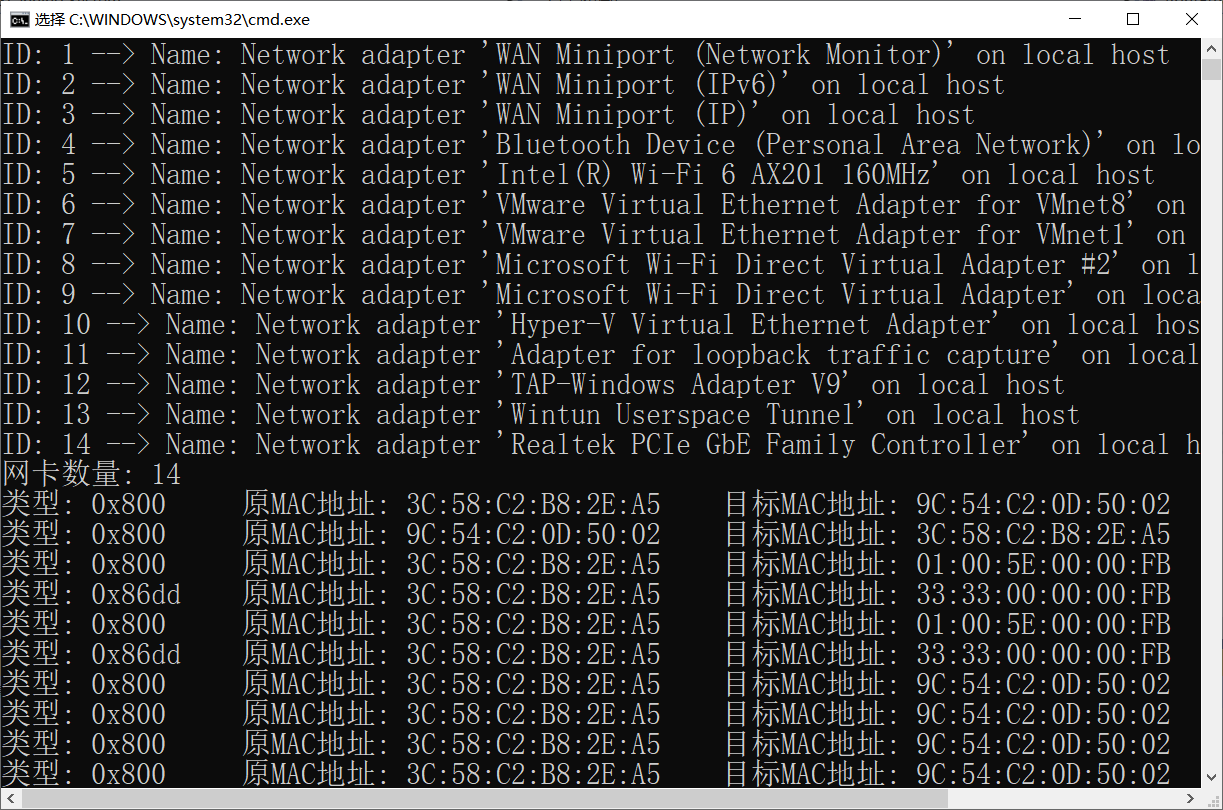
**（接下页）**



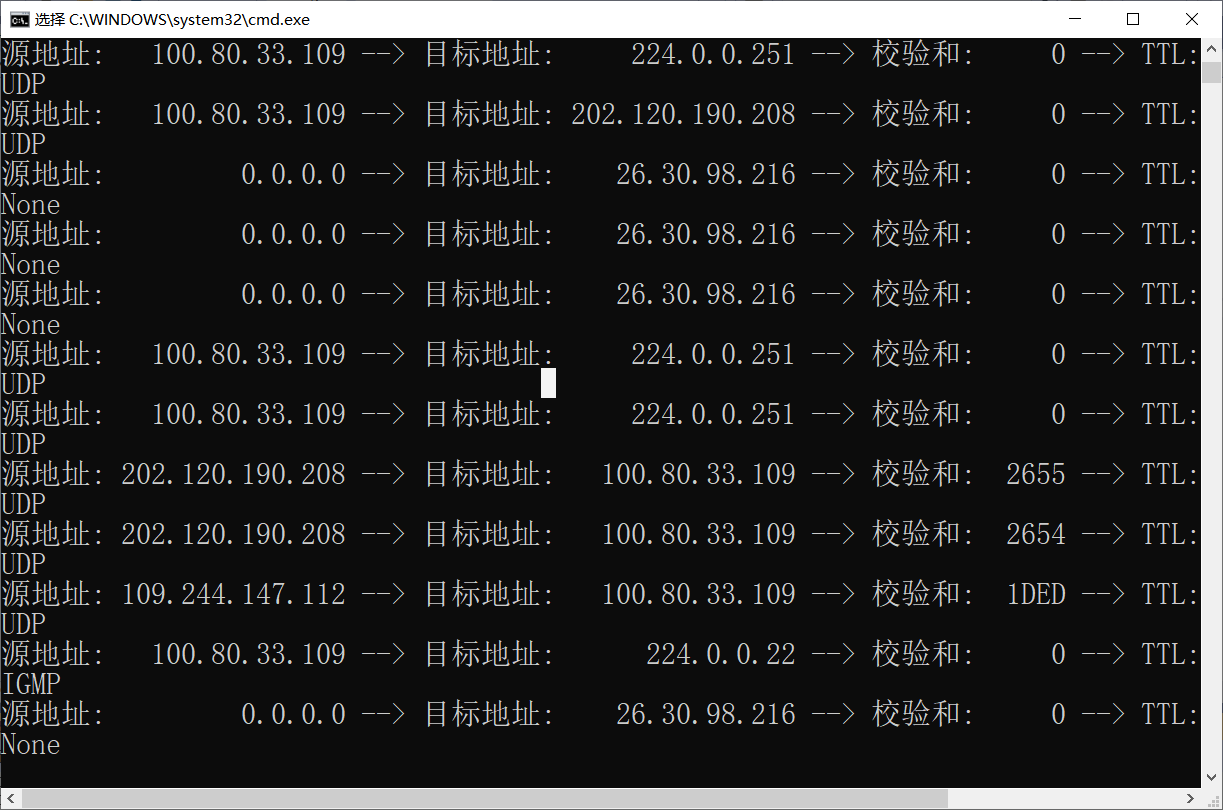
1.3 数据包捕获

在控制台窗口完成数据包捕获：

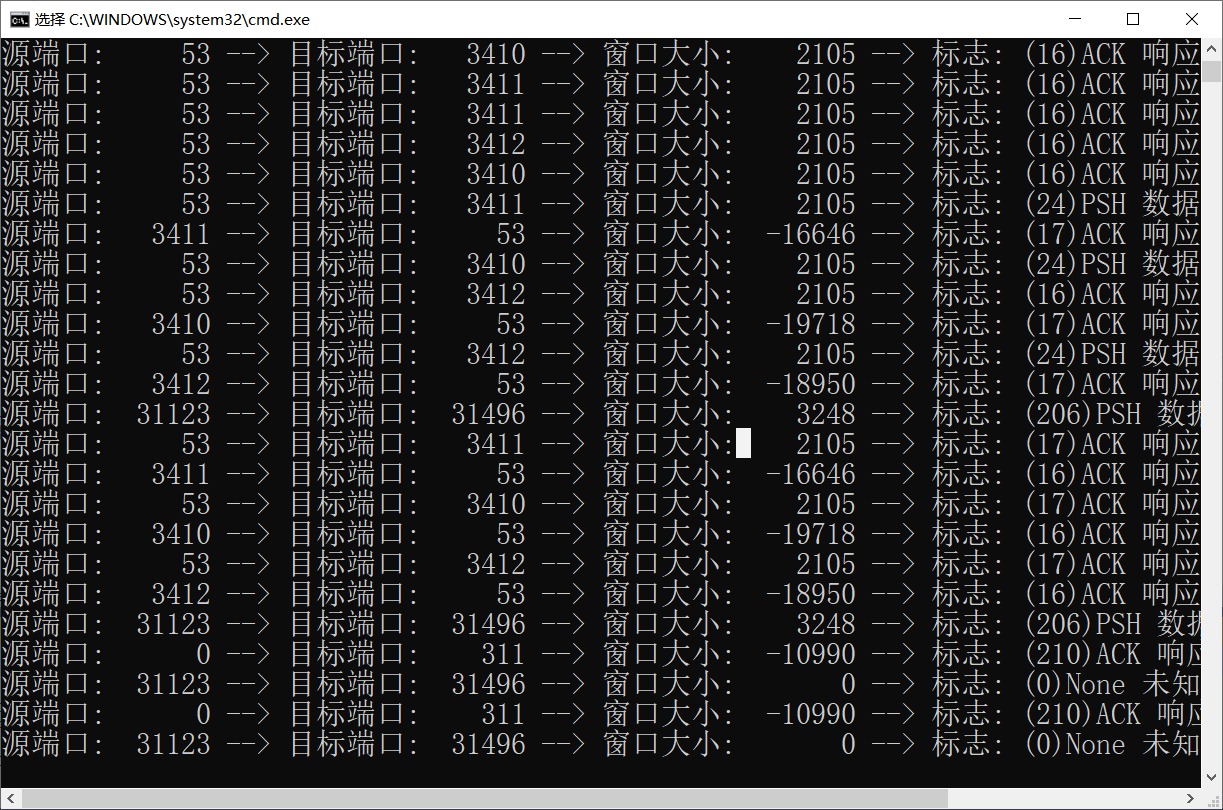
①PrintEtherHeader



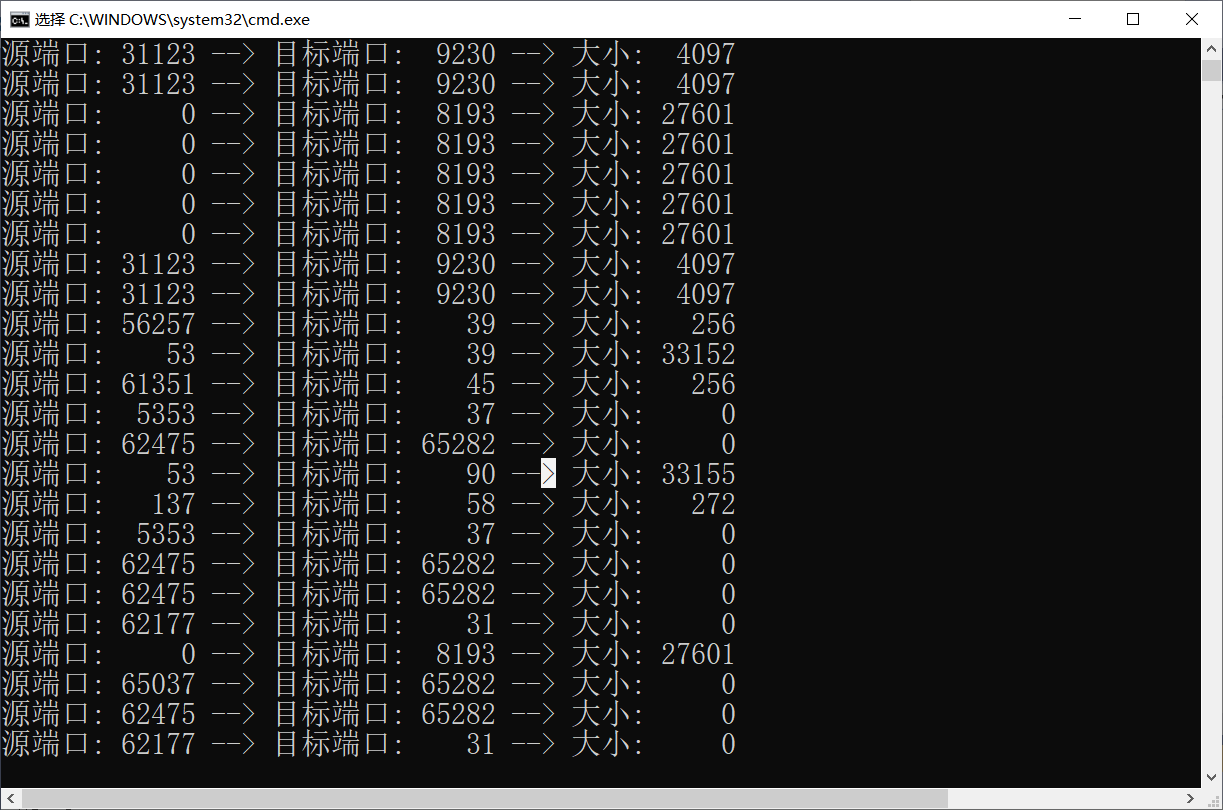
②PrintIPHeader



③PrintTCPHeader

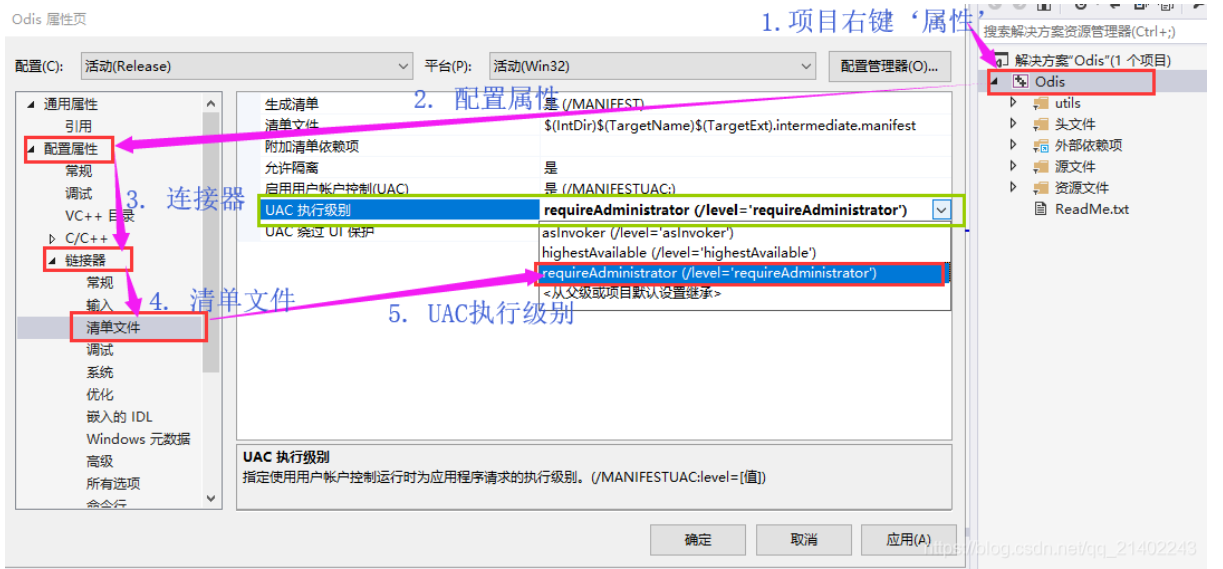


④PrintUDPHeader

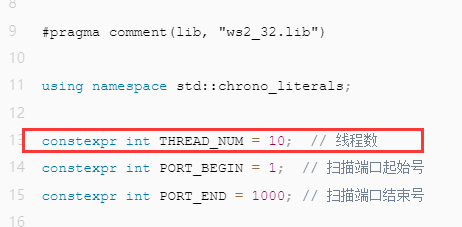


**2.测试时发现的问题及解决办法**

2.1 ICMP echo在测试时发现无法获得的情况，最终发现是程序需要使用管理员权限运行，解决方法如下：

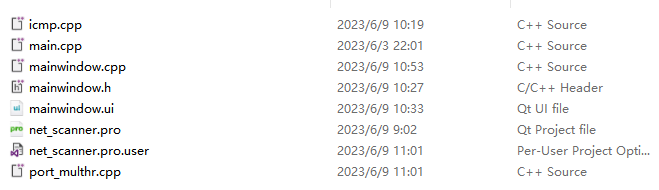


2.2 在测试时，发现端口扫描需要等待很久才出结果，当时是单线程端口扫描；我们后续研究了多线程端口扫描的方法，通过添加线程数量，大幅提升了扫描效率！



**五、软件安装方法**

我们的Qt源代码和.exe可执行文件均已经提交，可以使用Qt 5.12.0编译并运行本软件。



在电脑拥有所有需要的dll文件的前提下，也可以直接编译运行我们的.exe文件。