网络扫描器设计报告

2151140 王谦 信息安全

一、系统需求分析

1.功能需求

1.1 主机扫描

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

1.2 端口扫描

- 软件需实现端口扫描功能，允许用户指定要扫描的IP地址以及端口号范围。软件进行扫描后，输出指定IP地址在端口范围内所有的开放端口号。

1.3 本机扫描

- 软件需实现本机扫描功能，扫描本机网卡信息并能捕获数据包。

2. 非功能性需求

2.1 易用性

- 软件需通过图形化界面（GUI）实现，提供良好的输入输出提示，用户可通过GUI简单、清晰地进行操作。

2.2 性能

- 网络扫描过程应具备一定的效率。为此，软件应使用多线程等方式提高扫描速度。

3. 环境

3.1 操作系统

- Windows 10

3.2 软件框架

- Qt Creator 4.11.1 (Desktop Qt 5.14.2 MinGW 32-bit)

3.3 库

- ws2_32.lib、iphlpapi.lib、wpcap.lib

4. 初始状态

- 默认情况下，所有需要指定输入的部分均为0，无特殊权限设置。

5. 各组件逻辑调用关系

- 软件主要实现三个部件：用户界面、主机扫描、端口扫描和本机扫描。用户界面负责条件输入、交互和功能调用。

二、软件功能设计

扫描器的代码主要分为三个模块：主机扫描、端口扫描和本机扫描。以下是这些模块的整体代码流程描述：

1. 主机扫描模块：

- 用户在图形界面输入目标网段的前缀。
- 当用户触发主机扫描功能时，程序调用`hostscan_icmp()`函数。
- `hostscan_icmp()`函数调用`sendICMPEchoRequest()`函数发送ICMP Echo请求包。
- `sendICMPEchoRequest()`函数发送请求并等待响应，若收到响应则判断主机活跃，输出活跃主机IP地址列表。

2. 端口扫描模块：

- 用户在图形界面输入目标主机的IP地址以及端口范围。
- 当用户触发端口扫描功能时，程序调用`portscan_multhr()`函数。
- `portscan_multhr()`函数创建多个线程，每个线程负责扫描一个端口。
- 每个线程调用`PortScanTask`类中的函数，通过TCP套接字与目标主机建立连接，检查端口是否开放。
- 扫描结果输出开放端口号列表。

3. 本机扫描模块：

- 用户在图形界面输入网卡编号。
- 当用户触发本机扫描功能时，程序调用`localScan()`函数。
- `localScan()`函数使用WinPcap库捕获指定网卡的数据包。
- 捕获的数据包信息包括长度和16进制表示，输出到界面供用户查看。

1. 主机扫描

1.1 流程图

1.2 功能详细说明

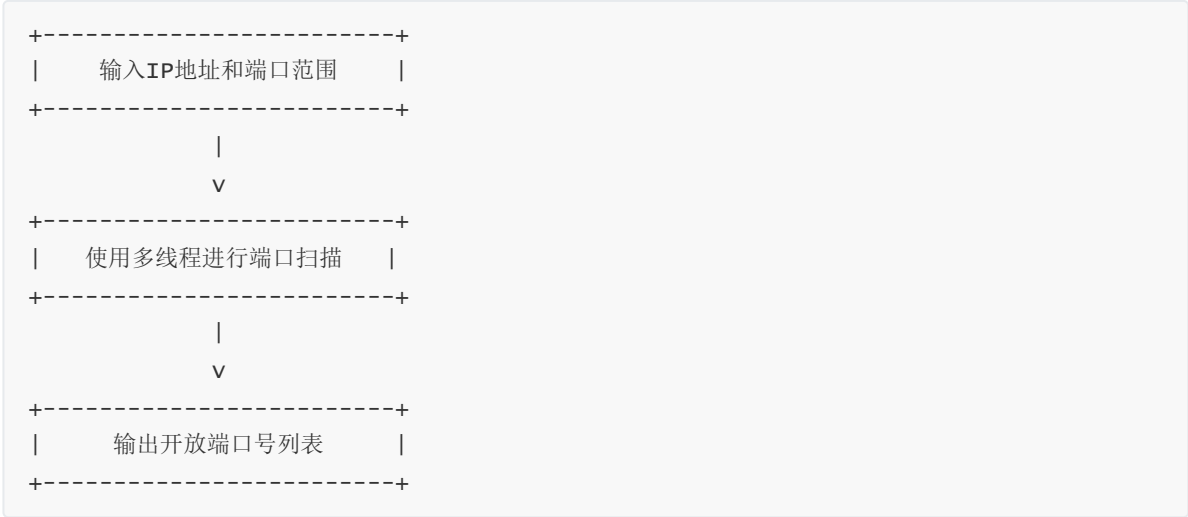
- 输入：网段前缀（如 `192.168.1.`）
- 输出：网段内活跃主机的IP地址列表
- 错误处理：输入网段不合理时，提示错误信息

1.3 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。主机扫描一般使用ICMP Echo Request包来实现，通过发送ICMP Echo Request包，接收ICMP Echo Reply包来判断主机的可达性。

2. 端口扫描

2.1 流程图



2.2 功能详细说明

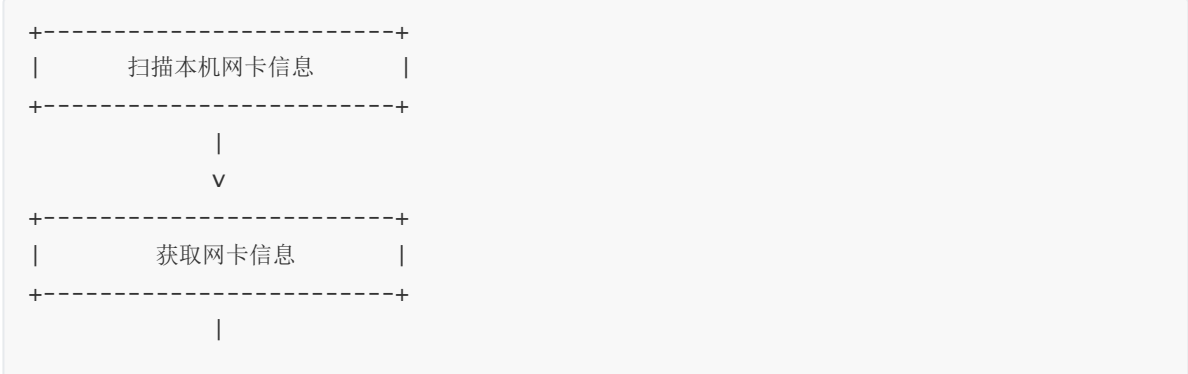
- 输入：IP地址（如 `192.168.1.1`），端口范围（如 `20-80`）
- 输出：指定IP地址在端口范围内开放的端口号列表
- 错误处理：输入端口范围不合理时，提示错误信息

2.3 相关原理

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（如SYN数据包），等待响应（如SYN/ACK数据包）来判断端口是否开放。

3. 本机扫描

3.1 流程图



```
              v
+-------------------------+
|     输入网卡编号和抓包数量    |
+-------------------------+
              |
              v
+-------------------------+
|        捕获并显示数据包        |
+-------------------------+
```

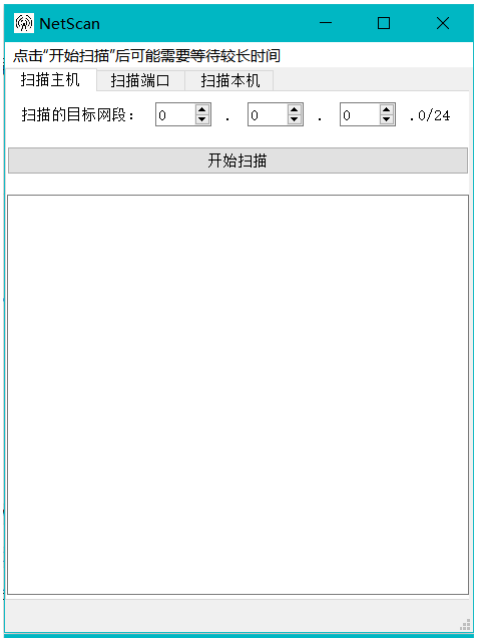
3.2 功能详细说明

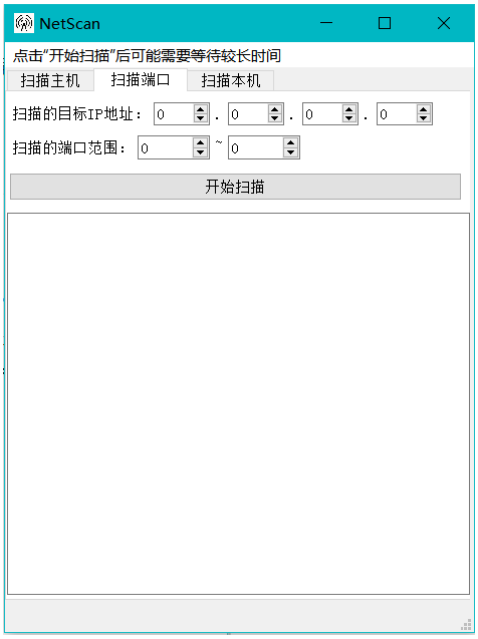
- 输入：网卡编号（如 `1`）
- 输出：网卡信息及捕获的数据包长度和16进制表示
- 错误处理：输入网卡编号不合理时，提示错误信息

三、详细设计

1. 界面设计

- 输入通过SpinBox实现，范围可控；输出显示在Text Browser中。





2. 模块设计

2.1 主机扫描模块

- 调用函数：`int MainWindow::hostscan_icmp(QString s)`
- 输入：网段前缀（`x.x.x.` 形式）
- 功能：实现主机扫描并输出结果

2.2 端口扫描模块

- 调用函数：`int MainWindow::portscan_multhr(QString s, int pstart, int pend)`
- 输入：IP地址（`x.x.x.x` 形式），起始端口号，结束端口号
- 功能：实现端口扫描并输出结果

2.3 用户界面模块

- 负责将输入处理为其他模块需要的数据格式，并调用相应函数实现功能。

3. 具体实现

主机扫描代码示例:

```
bool MainWindow::sendICMPEchoRequest(QString targetIP)
{
    int retries = 3;
    while (retries--)
    {
        if (sendICMPPacket(targetIP))
        {
            return true;
        }
    }
    return false;
}

bool MainWindow::sendICMPPacket(QString targetIP)
{
    // 发送ICMP Echo Request并接收响应的具体实现
    // 省略细节代码
    return true; // 示例中默认返回true
}
```

端口扫描代码示例:

```
void MainWindow::portscan_multhr(QString targetIP, int startPort, int endPort)
{
    if (startPort > endPort)
    {
        ui->outputText->append("Start port must be less than end port.");
        return;
    }

    QThreadPool::globalInstance()->setMaxThreadCount(20); // 增加线程数量

    for (int port = startPort; port <= endPort; ++port)
    {
        PortScanTask* task = new PortScanTask(targetIP, port);
        QThreadPool::globalInstance()->start(task);
    }
}
```

本机扫描代码示例:

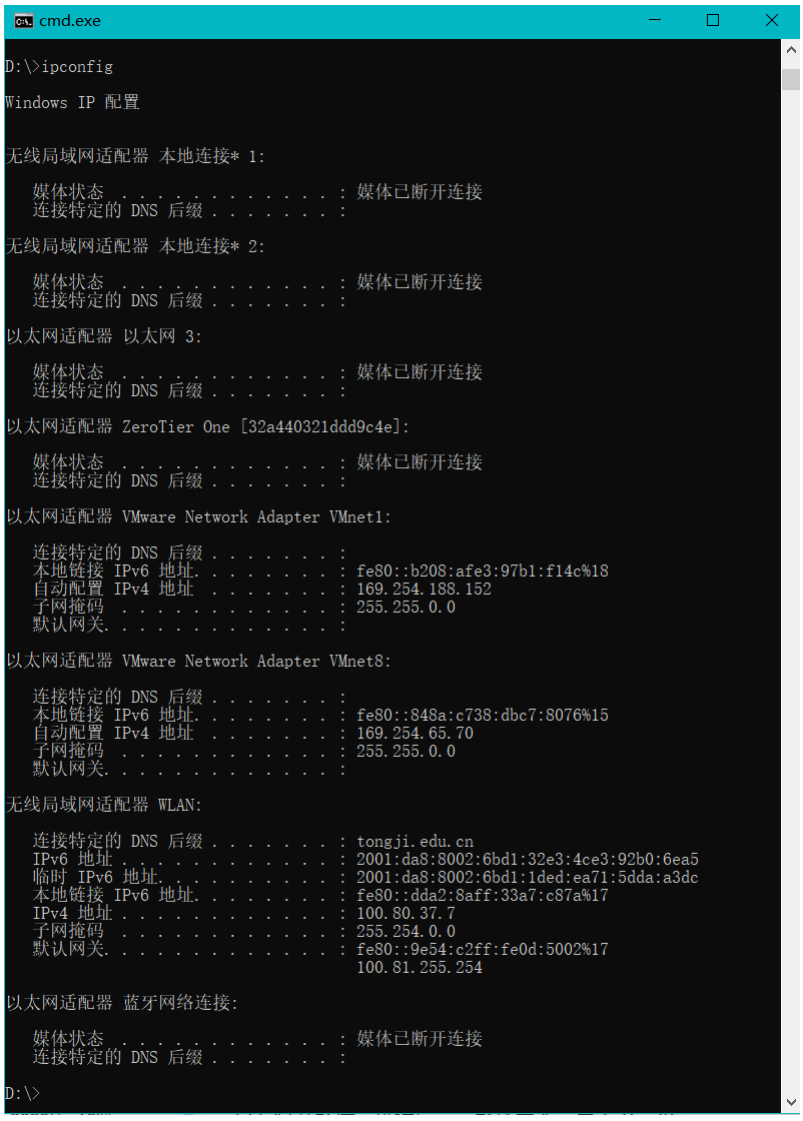
```
void MainWindow::localScan(int interfaceIndex)
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int i = 0;

    // 获取所有设备列表
    if (pcap_findalldevs(&alldevs, errbuf) == -1)
    {
```

```
        ui->outputText->append("Error in pcap_findalldevs: " + QString(errbuf));
        return;
    }

    for (d = alldevs; d; d = d->next)
    {
        if (i == interfaceIndex)
        {
            ui->outputText->append("Selected interface: " + QString(d->name));
            break;
        }
        i++;
    }

    if (d == nullptr)
    {
        ui->outputText->append("Interface not found.");
        pcap_freealldevs(alldevs);
        return;
    }

    pcap_t *adhandle;
    adhandle = pcap_open_live(d->name, 65536, 1, 1000, errbuf);
    if (adhandle == nullptr)
    {
        ui->outputText->append("Unable to open the adapter. " + QString(d->name)
+ " is not supported by WinPcap");
        pcap_freealldevs(alldevs);
        return;
    }

    struct pcap_pkthdr *header;
    const u_char *pkt_data;
    int res;
    while ((res = pcap_next_ex(adhandle, &header, &pkt_data)) >= 0)
    {
        if (res == 0) {
            continue; // Timeout elapsed
        }

        ui->outputText->append("Packet length: " + QString::number(header-
>len));
        QString hexData;
        for (u_int i = 0; i < header->len; i++)
        {
            hexData += QString::number(pkt_data[i], 16).rightJustified(2, '0') +
" ";
        }
        ui->outputText->append("Packet data: " + hexData);
    }

    pcap_close(adhandle);
    pcap_freealldevs(alldevs);
}
```

四、软件测试

1. 测试方法及过程

本机ipconfig结果：



1.1 主机扫描

测试目的：验证主机扫描功能的正确性和稳定性。

测试步骤：

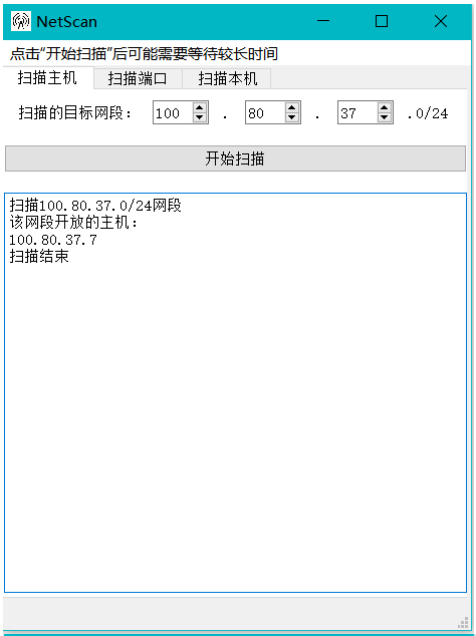
1. 输入网段前缀（如 `100.80.37.`、`182.61.200.`）。
2. 执行主机扫描。
3. 检查输出的活跃主机IP地址列表。

测试样例：

- 输入网段：`100.80.37.`、`182.61.200.`
- 预期输出：网段内所有活跃的主机IP地址。

实际结果：

- 测试结果与预期一致，输出了网段内所有活跃的主机IP地址。



1.2 端口扫描

测试目的：验证端口扫描功能的正确性和效率。

测试步骤：

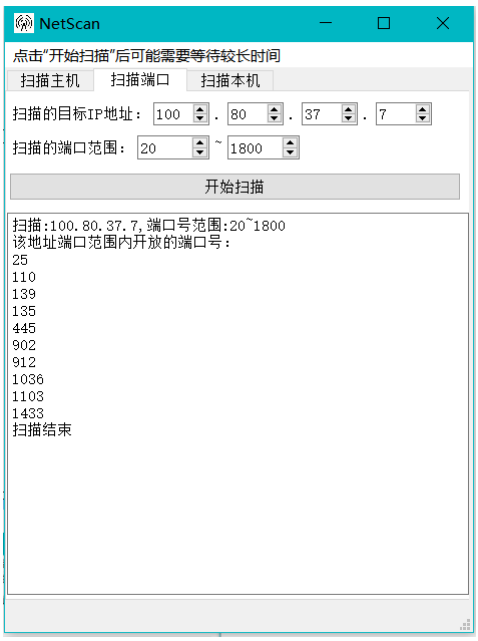
1. 输入IP地址（如 `100.80.37.7`、`182.61.200.7`）和端口范围（如 `20-1800`）。
2. 执行端口扫描。
3. 检查输出的开放端口列表。

测试样例：

- 输入IP地址：`100.80.37.7`、`182.61.200.7`
- 输入端口范围：`20-1800`
- 预期输出：指定IP地址在端口范围内所有开放的端口号。

实际结果：

- 测试结果与预期一致，输出了指定IP地址在端口范围内所有开放的端口号。



1.3 本机扫描

测试目的：验证本机扫描功能的正确性和数据包捕获的准确性。

测试步骤：

1. 输入网卡编号（如 `1`）。
2. 执行本机扫描。
3. 检查输出的网卡信息和数据包的详细信息。

测试样例：

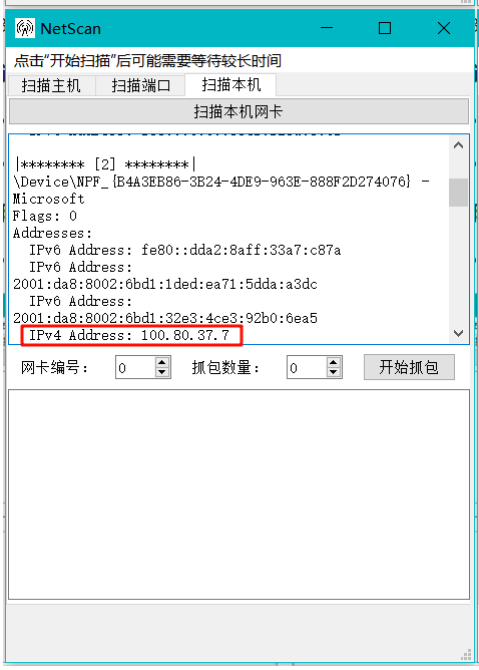
- 输入网卡编号：`1`
- 预期输出：指定网卡的详细信息和捕获的数据包长度及其16进制表示。

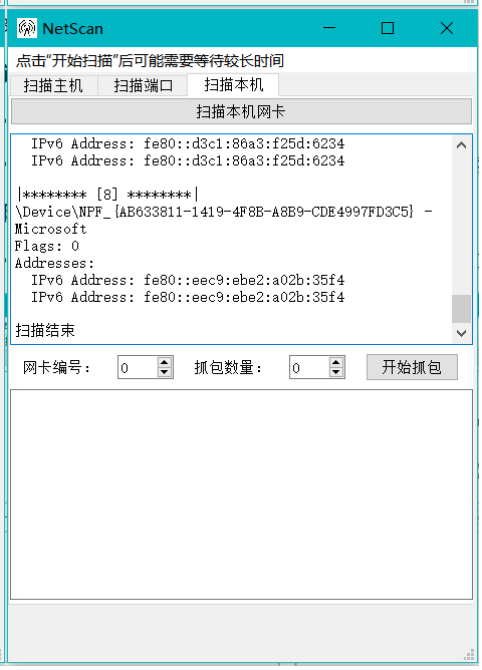
实际结果：

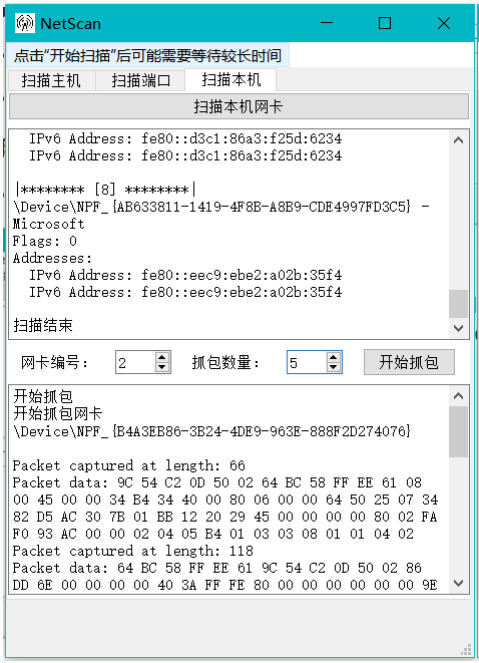
- 测试结果与预期一致，输出了指定网卡的详细信息和捕获的数据包。

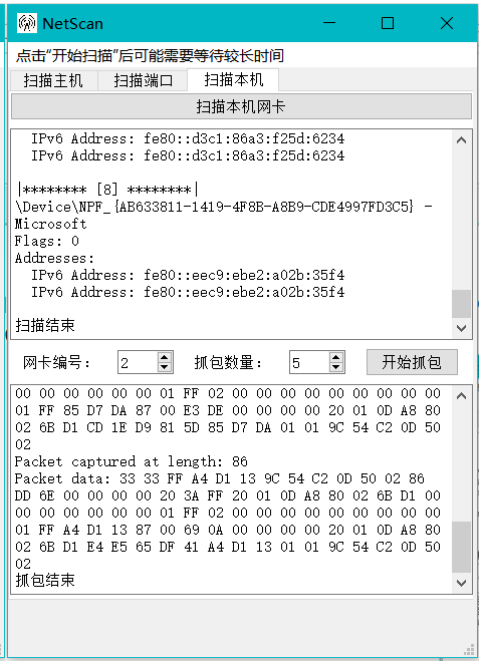

NetScan
点击"开始扫描"后可能需要等待较长时间
扫描主机
扫描端口
扫描本机
扫描本机网卡
网卡编号：
0
抓包数量：
0
开始抓包


NetScan
点击"开始扫描"后可能需要等待较长时间
扫描主机
扫描端口
扫描本机
扫描本机网卡
开始扫描本机网卡
Available Network Interfaces:
|******** [1] ********|
\Device\NPF_{2AF860F4-1406-4E23-B9C1-F07A44420EF9} -
TAP-Windows Adapter V9
Flags: 0
Addresses:
IPv6 Address: fe80::c969:834b:b28a:8061
IPv6 Address: fe80::c969:834b:b28a:8061
|******** [2] ********|
网卡编号：
0
抓包数量：
0
开始抓包


NetScan
点击"开始扫描"后可能需要等待较长时间
扫描主机
扫描端口
扫描本机
扫描本机网卡
|******** [2] ********|
\Device\NPF_{B4A3EB86-3B24-4DE9-963E-888F2D274076} -
Microsoft
Flags: 0
Addresses:
IPv6 Address: fe80::dda2:8aff:33a7:c87a
IPv6 Address:
2001:da8:8002:6bd1:1ded:ea71:5dda:a3dc
IPv6 Address:
2001:da8:8002:6bd1:32e3:4ce3:92b0:6ea5
IPv4 Address: 100.80.37.7
网卡编号：
0
抓包数量：
0
开始抓包


NetScan
点击"开始扫描"后可能需要等待较长时间
扫描主机
扫描端口
扫描本机
扫描本机网卡
IPv6 Address: fe80::d3c1:86a3:f25d:6234
IPv6 Address: fe80::d3c1:86a3:f25d:6234
|******** [8] ********|
\Device\NPF_{AB633811-1419-4F8B-A8B9-CDE4997FD3C5} -
Microsoft
Flags: 0
Addresses:
IPv6 Address: fe80::eec9:ebe2:a02b:35f4
IPv6 Address: fe80::eec9:ebe2:a02b:35f4
扫描结束
网卡编号：
0
抓包数量：
0
开始抓包


NetScan
点击"开始扫描"后可能需要等待较长时间
扫描主机
扫描端口
扫描本机
扫描本机网卡
IPv6 Address: fe80::d3c1:86a3:f25d:6234
IPv6 Address: fe80::d3c1:86a3:f25d:6234
|******** [8] ********|
\Device\NPF_{AB633811-1419-4F8B-A8B9-CDE4997FD3C5} -
Microsoft
Flags: 0
Addresses:
IPv6 Address: fe80::eec9:ebe2:a02b:35f4
IPv6 Address: fe80::eec9:ebe2:a02b:35f4
扫描结束
网卡编号：
2
抓包数量：
5
开始抓包
开始抓包
开始抓包网卡
\Device\NPF_{B4A3EB86-3B24-4DE9-963E-888F2D274076}
Packet captured at length: 66
Packet data: 9C 54 C2 0D 50 02 64 BC 58 FF EE 61 08
00 45 00 00 34 B4 34 40 00 80 06 00 00 64 50 25 07 34
82 D5 AC 30 7E 01 BB 12 20 29 45 00 00 00 00 80 02 FA
F0 93 AC 00 00 02 04 05 B4 01 03 03 08 01 01 04 02
Packet captured at length: 118
Packet data: 64 BC 58 FF EE 61 9C 54 C2 0D 50 02 86
DD 6E 00 00 00 00 40 3A FF FE 80 00 00 00 00 00 00 9E


NetScan
点击"开始扫描"后可能需要等待较长时间
扫描主机
扫描端口
扫描本机
扫描本机网卡
IPv6 Address: fe80::d3c1:86a3:f25d:6234
IPv6 Address: fe80::d3c1:86a3:f25d:6234
|******** [8] ********|
\Device\NPF_{AB633811-1419-4F8B-A8B9-CDE4997FD3C5} -
Microsoft
Flags: 0
Addresses:
IPv6 Address: fe80::eec9:ebe2:a02b:35f4
IPv6 Address: fe80::eec9:ebe2:a02b:35f4
扫描结束
网卡编号：
2
抓包数量：
5
开始抓包
00 00 00 00 00 00 01 FF 02 00 00 00 00 00 00 00 00 00
01 FF 85 D7 DA 87 00 E3 DE 00 00 00 00 20 01 0D A8 80
02 6B D1 CD 1E D9 81 5D 85 D7 DA 01 01 9C 54 C2 0D 50
02
Packet captured at length: 86
Packet data: 33 33 FF A4 D1 13 9C 54 C2 0D 50 02 86
DD 6E 00 00 00 00 20 3A FF 20 01 0D A8 80 02 6B D1 00
00 00 00 00 00 00 01 FF 02 00 00 00 00 00 00 00 00 00
01 FF A4 D1 13 87 00 69 0A 00 00 00 00 20 01 0D A8 80
02 6B D1 E4 E5 65 DF 41 A4 D1 13 01 01 9C 54 C2 0D 50
02
抓包结束

2. 测试时发现的问题及解决办法

在测试过程中，发现了以下问题并进行了相应的优化：

主机扫描和端口扫描速度慢：使用多线程技术，提高扫描效率。

2.1 主机扫描

1. **定义全局变量和互斥锁**：
   - `s_mutex2`：互斥锁，用于保护共享资源的并发访问。
2. **ICMP 主机扫描函数** `scan_host_icmp`：
   - 创建ICMP文件句柄，用于发送和接收ICMP请求。
   - 分配缓冲区，用于接收ICMP响应数据。
   - 发送ICMP请求并接收响应，如果收到响应，则使用互斥锁保护输出操作，确保多个线程不会同时访问共享资源。
   - 释放缓冲区和关闭ICMP文件句柄。
3. **ICMP 主机扫描多线程调用函数** `hostscan_icmp_multhr`：
   - 初始化Winsock库。
   - 将 `QString` 类型的基地址转换为标准的C++字符串类型。
   - 创建多个线程，每个线程执行 `scan_host_icmp` 函数进行ICMP主机扫描。
   - 使用 `join` 方法等待所有线程执行完毕。
   - 清理Winsock库。

2.2 端口扫描

1. **定义常量和全局变量**：
   - `THREAD_NUM`：定义了线程数量，这里设置为200，表示将同时启动200个线程进行扫描。
   - `PORT_BEGIN` 和 `PORT_END`：定义了端口扫描的范围，从1到65535。
   - `s_port`：使用 `std::atomic<int>` 来保存当前扫描到的端口号，以确保多线程环境下端口号的安全递增。
   - `s_mutex`：互斥锁，用于保护共享资源 `open_port` 的并发访问。
   - `open_port`：QList，用于保存发现的开放端口。
2. **端口扫描函数** `scan_port`：
   - 初始化 `s_port` 为扫描的起始端口号。
   - 在循环中，通过原子操作递增 `s_port`，确保每个线程获取到不同的端口号进行扫描。
   - 为每个端口创建一个套接字，并设置为非阻塞模式。
   - 使用 `connect` 函数尝试连接目标端口，如果连接成功，则记录该端口为开放端口。
   - 使用 `select` 函数检查套接字状态，判断连接是否成功。
   - 通过 `std::lock_guard<std::mutex>` 保护对共享资源 `open_port` 的访问，确保线程安全。
3. **多线程端口扫描函数** `portscan_multhr`：
   - 初始化 Winsock 库。
   - 获取目标 IP 地址并创建多个线程，每个线程执行 `scan_port` 函数进行端口扫描。
   - 使用 `join` 方法等待所有线程执行完毕。
   - 清理 Winsock 库。
   - 将扫描到的开放端口显示在界面上。

多线程技术的优点:

1. **提高扫描效率**：通过同时启动多个线程，可以在更短的时间内完成对大量端口的扫描任务。
2. **提升程序性能**：多线程技术可以充分利用多核 CPU 的优势，提高程序的执行效率。
3. **增强用户体验**：快速响应的扫描结果使得用户体验更加流畅和高效。

3.有待改进的空间

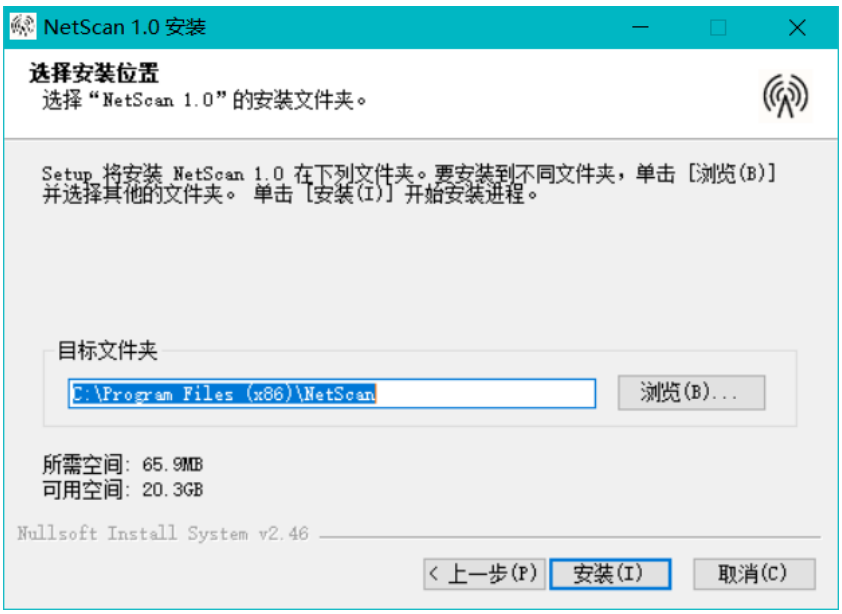
主机扫描不准确：增加ICMP包的重试机制，提高扫描准确性。

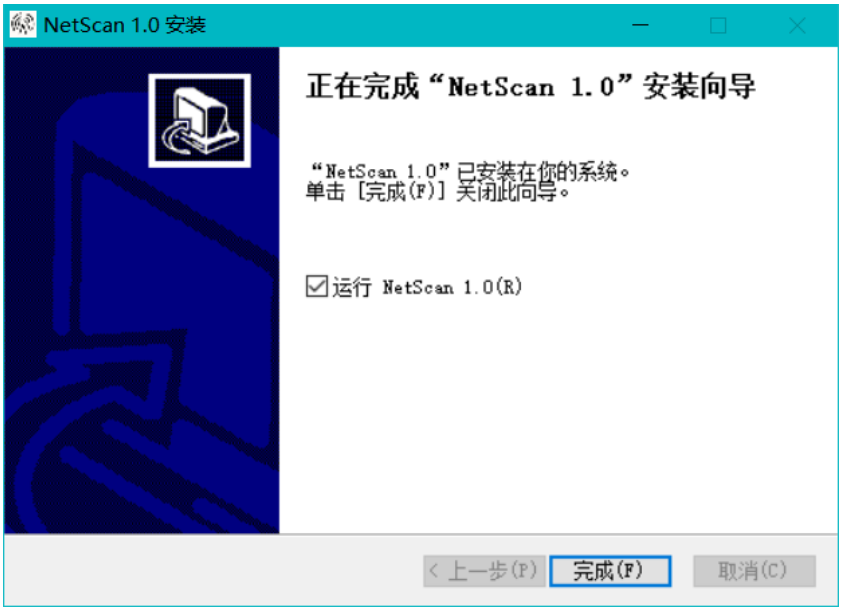
本机扫描信息不完整：优化WinPcap库的调用和数据处理逻辑，确保信息完整和稳定性。

输出内容呈现不够直观：优化界面和输出流程及格式，提升输出信息的直观性。

五、软件安装方法

安装包由NSIS制作





qt源代码和包含NetScan.exe可执行文件的realese文件夹绿色版也已提交，可以使用Qt 5.12.0编译并运行本软件；如果电脑拥有所有需要的dll文件，也可以直接运行.exe文件。

六、后记

这个软件是我参照往届学长的作业而来的（https://github.com/ChestnutSilver/Network-Scanning-System），一开始完全不知道如何着手，学长的项目给了我很大的入手帮助。

我重新设计了ui；修改了端口扫描这个功能并修了bug（端口扫描原本在扫描一次后就无法再次扫描，现在可以反复重新扫描）；重构了主机扫描功能（原本主机扫描是直接顺序扫描，0-255，每个主机扫描1000ms，所以慢的话每次需要花费4分钟以上的时间，我改为使用多线程技术，优化后几秒钟就可以完成扫描）；并添加了本机网卡扫描和抓包的相关功能。以及绘制了图标、制作了安装包。

虽然说参考了学长的项目，两个功能我重构了一个半，然后还额外添加了一个新功能，因此仍有相当比重的原创性。不过软件整体来看仍然是一个小demo，各个功能依旧只是做个演示的程度。