**实验名称：**计算机网络课程设计

**实验目的：**

将书本上抽象的概念与具体实现技术结合，通过网络软件编程的实践，深入理解理论课上学习到的计算机网络基本原理和重要协议，通过自己动手编程封装与发送数据分组，加深对网络协议的理解，掌握协议数据单元的结构和协议工作原理及其对协议栈的贡献。

**实验仪器：**

实验硬件设备：计算机、手机

用到的计算机：计算机

网络设备：笔记本电脑、手机

实验软件要求：使用Wireshark进行抓包

使用的操作系统：Windows 10专业版

开发工具：Visio Studio2019、WinPcap4.1、Wireshark

编程语言：C++

**实验原理：**

**题目1实验原理：**

数据包的封装发送和解析（ARP/ICMP/TCP），网络协议栈的多种协议都有自己的功能，协议包括语义、语法和同步三个要素，不同的网络协议其分组的首部格式不同，必须按照协议规定的格式封装（发送）和理解（接收）数据分组首部，才能使得不同站点的计算机按照规定的方式相互通信。ARP协议是IP地址和MAC地址解析协议；ICMP是控制IP数据包传递的协议；TCP是面向连接的可靠的传输层协议，它们均有自己固定的分组首部格式。

本实验中使用WinPcap技术或Socket技术，根据ARP/ICMP/TCP协议数据单元的结构和封装规则，封装数据帧发送到局域网中。另外要捕获网络中的ARP/ICMP/TCP数据包，解析数据包的内容，并显示结果，同时写入日志文件。

**题目2实验原理：**

基于IP的路由跟踪小程序，如tracert（在Windows中）或traceroute（在Linux中），基于一种称为“时间生存”（TTL）的Internet协议机制实现。这个程序利用ICMP消息来发现和记录从源主机到目的主机之间的路径。

时间生存（TTL）机制:

TTL是IP头中的一个字段，表示一个数据包在网络中可以通过的最大路由器数。每当数据包经过一个路由器，其TTL值减1。当TTL值减至0时，路由器将丢弃该数据包，并向源地址发送一个ICMP“时间超过”（Time Exceeded）消息。

ICMP消息:

ICMP是用于IP主机或路由器之间传递控制消息的协议，如目的不可达、时间超过等。路由跟踪程序主要使用了两种ICMP消息：ICMP回显请求（用于发起跟踪）和ICMP时间超过（用于响应TTL过期）。

递增TTL值:

路由跟踪程序通过逐步增加发往目标地址的ICMP回显请求消息的TTL值来工作。首先，程序将TTL设置为1，并发送第一个ICMP请求。当该请求的TTL值耗尽时，路径上的第一个路由器将丢弃该请求并发送一个ICMP时间超过消息回到源地址。程序记录下这个路由器的IP地址，并使用更高的TTL值发送下一个请求。这个过程重复进行，直到到达目标或达到预定的最大跳数。

往返时间（RTT）测量:

RTT是指发送ICMP回显请求从源主机到目的主机，再返回源主机的总时间。通过记录发送请求和接收响应的时间差，程序可以计算出每一跳的RTT。

**实验内容与步骤：**

**题目1：数据分组的发送和解析（ARP/ICMP/TCP）**

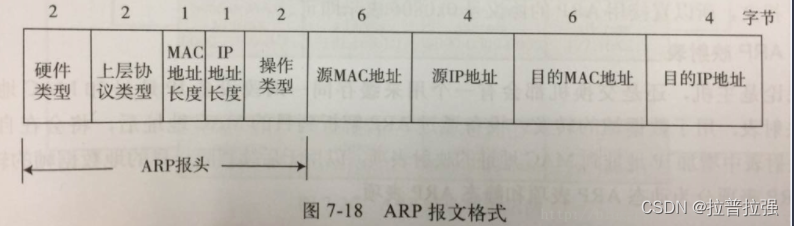
（1）需求分析

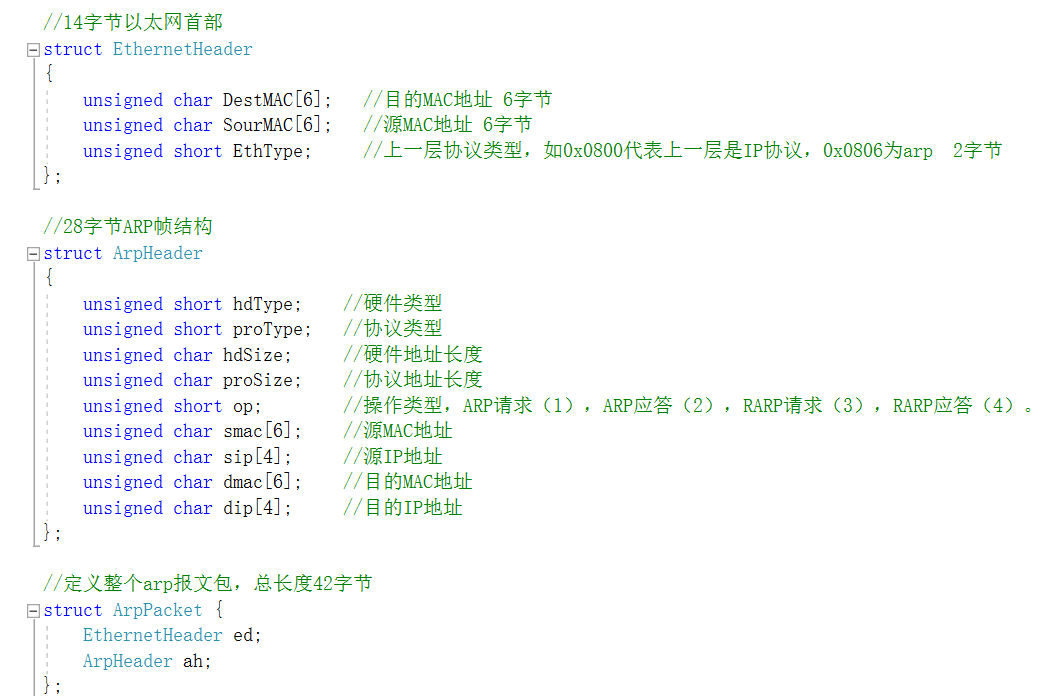
选择ARP、ICMP或TCP协议之一，在一台计算机上编写、编译和运行程序，使其能封装好相应的数据包并发送到另一个网络设备（电脑或手机）上接收并解析该数据包，将该数据包的首部字段的信息输出显示出来。本程序主要使用“WinPcap”库和“C++”编程语言实现，编程过程中要着重定义好ARP\ICMP\TCP协议首部的数据结构，自定义填充要发送的数据包，以命令行形式运行程序并且发送和接受的数据包能在Wireshark抓包软件中捕获显示出来。

（2）设计方案

该程序主要分为五大模块：变量定义模块、获取设备列表模块、发送ARP数据包模块、ARP数据包监听模块和接收ARP数据包模块。

首先在变量定义模块，我们严格对照ARP协议首部字段格式定义了“EthernetHeader”（14字节的以太网首部）和“ArpHeader”（28字节的ARP首部）这两个主要数据结构，并定义了“ArpPacket”(42字节的ARP报文包)为后面发送解析ARP报文做准备。其中“ARP报文格式”和定义的上述三个结构体代码截图如下。此外，还定义了“alldevs”（适配器列表）等全局变量为后面的具体实现做准备。





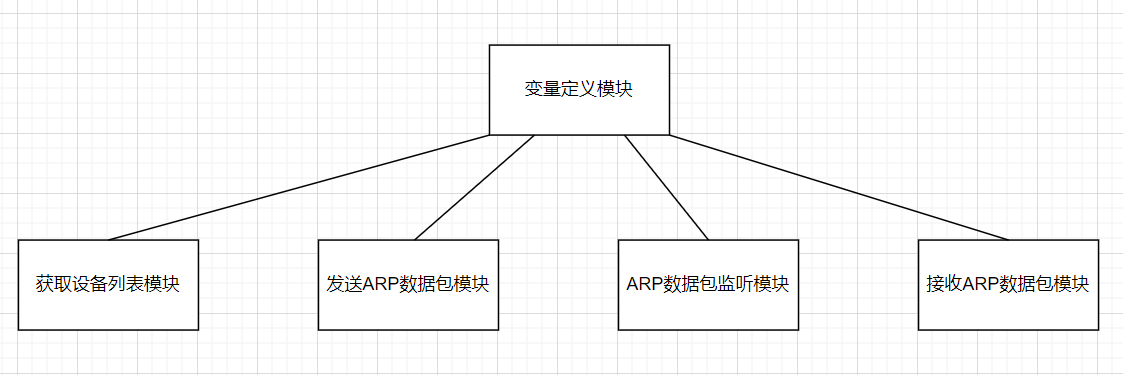
后面四个模块分别与我们设计的四个具体实现函数device\_init()、sent\_arp\_packet()、listen\_init()、recv\_arp\_packet()一一对应，在主函数main（）（如下图）中我们按顺序依次调用了四个函数来实现发送和接收分析ARP数据包的一整套流程。



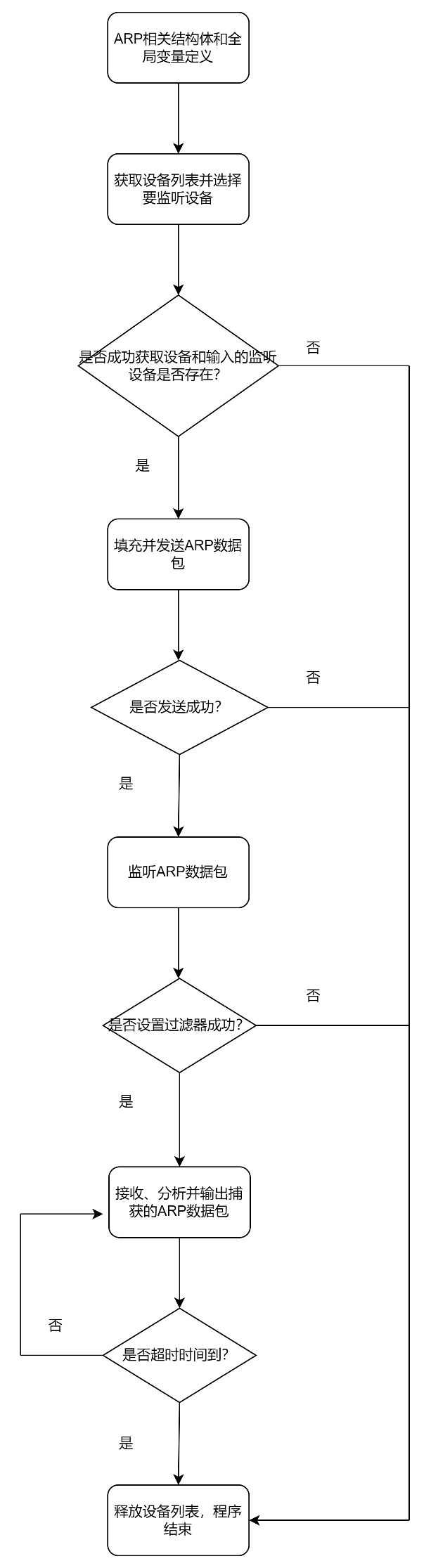
该程序的这五个模块密不可分，共同构建了一个完整的ARP包发送接收分析处理流程，并且通过共享部分数据（全局变量）和状态信息来协同工作。具体来说就是，变量定义模块为其他四个模块提供了必要的数据结构定义和常量；获取设备列表模块初始化网络设备列表，为后续的发送和监听等操作提供了基础；发送ARP数据包模块又依赖于设备列表初始化模块选定的网络接口和全局状态信息；ARP数据包监听模块要根据定义的ARP协议数据结构来配置过滤规则只捕获ARP包；接收ARP数据包模块则根据定义的ARP协议数据结构来分析输出捕获到的ARP数据包。总之，每个模块都扮演着特定的角色，并且依赖于其他模块提供的信息和功能，缺一不可。

（3）程序结构和流程图、

本程序结构图如下：

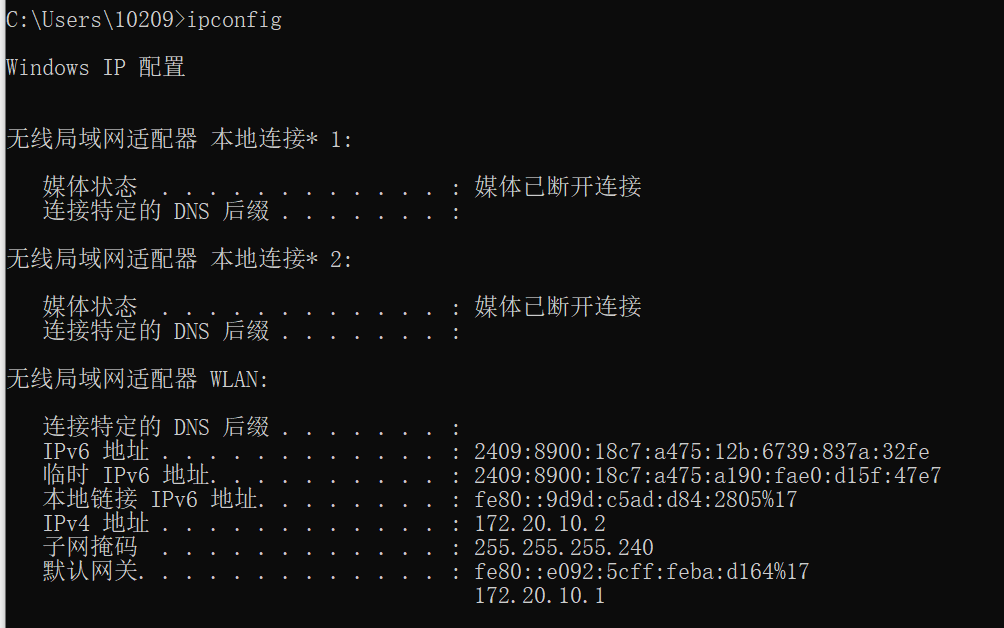


本程序流程图如下：

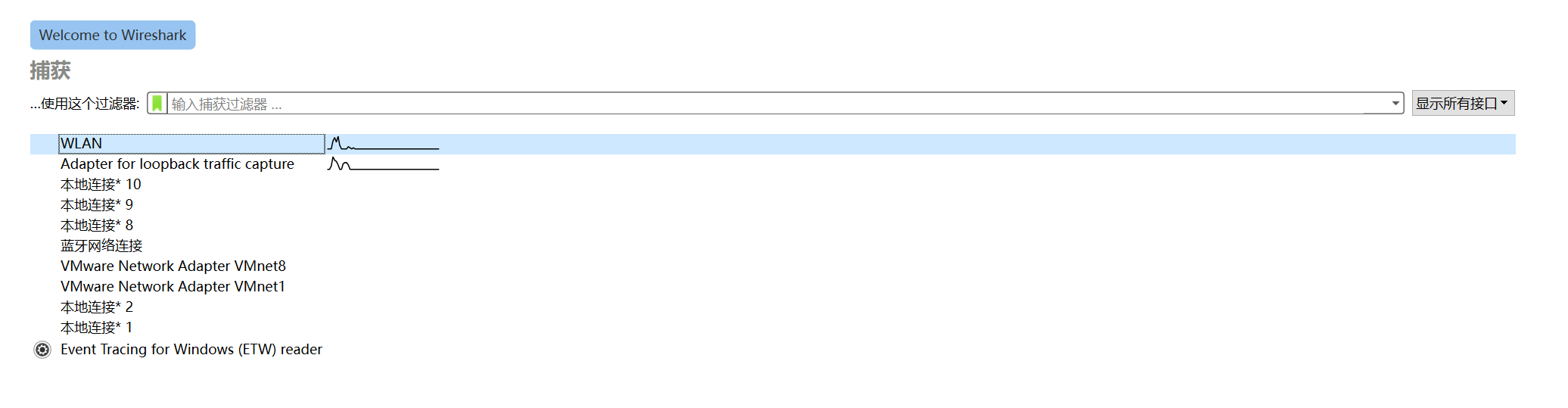


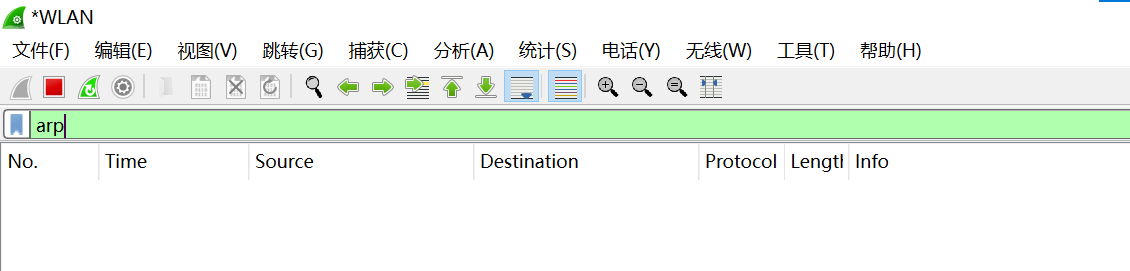
（4）测试方案

打开手机热点，将一台电脑连接至该热点中使得电脑与手机在同一个网络中。在电脑CMD命令行中输入“ipconfig/all”命令（如下图）查看本电脑的硬件地址（即源MAC地址）和源以及目的IPv4地址填入程序的“发送ARP数据包模块”以便确保在程序填充ARP数据包发送时是从电脑发往手机的。

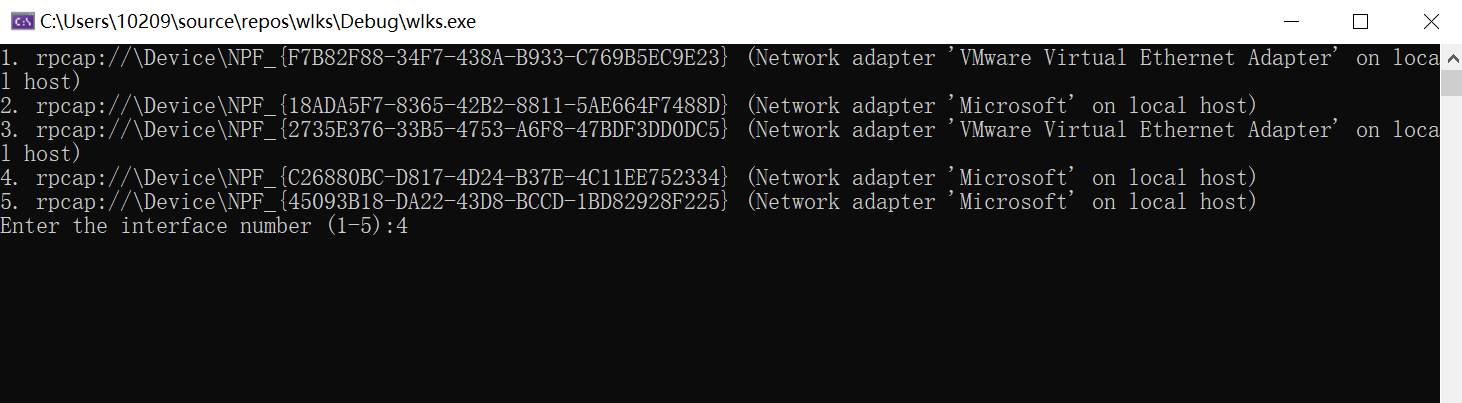


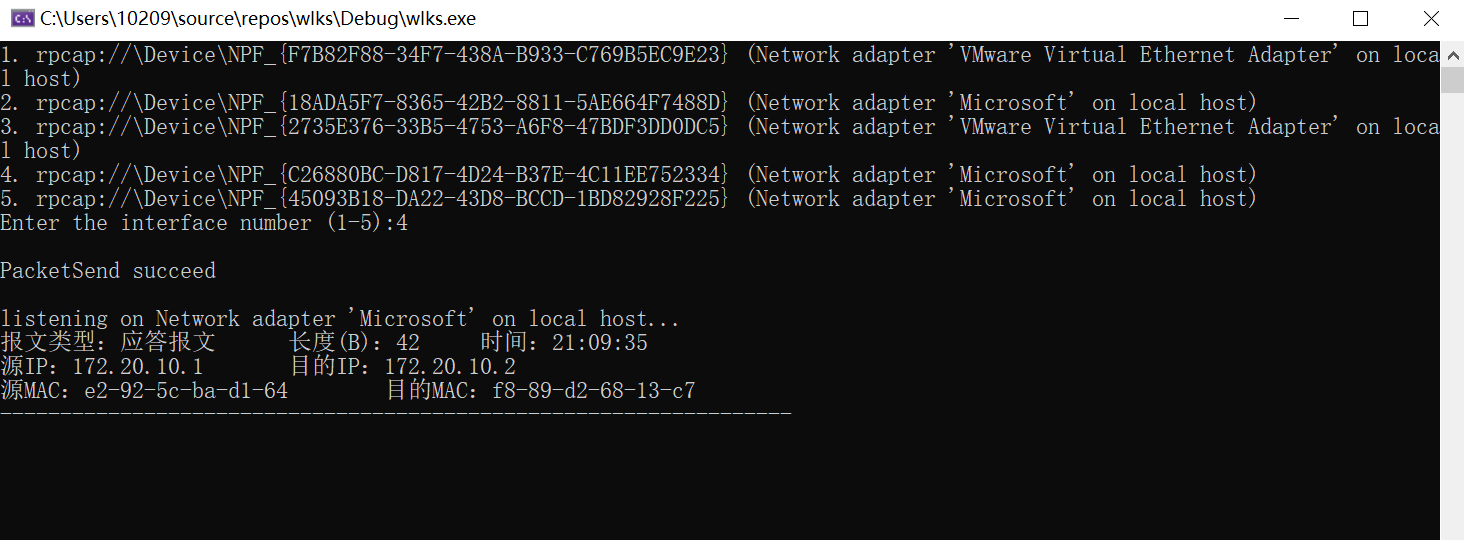
做完上述准备工作后，即可打开Wireshark抓包软件选择对应的无线网络WLAN并开启“ARP”过滤器准备抓包，如下图：



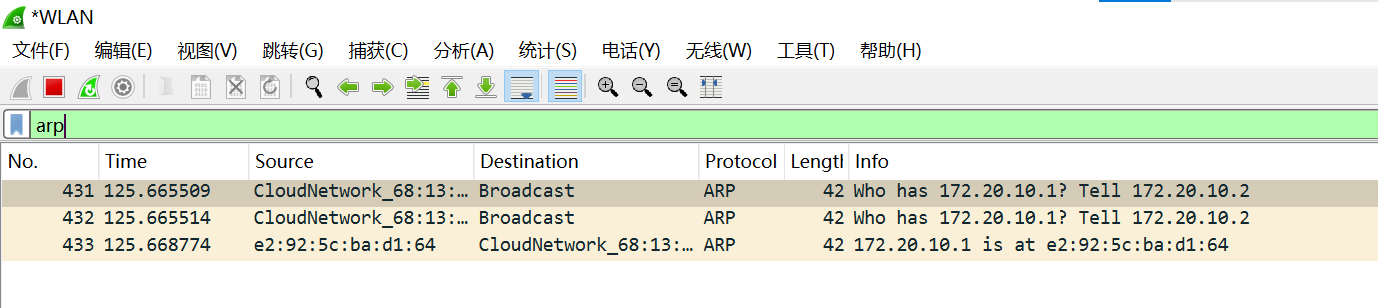


程序运行前Wireshark软件中未捕获到ARP数据包。此时，运行该程序，选择对应的设备号“4”运行截图如下：

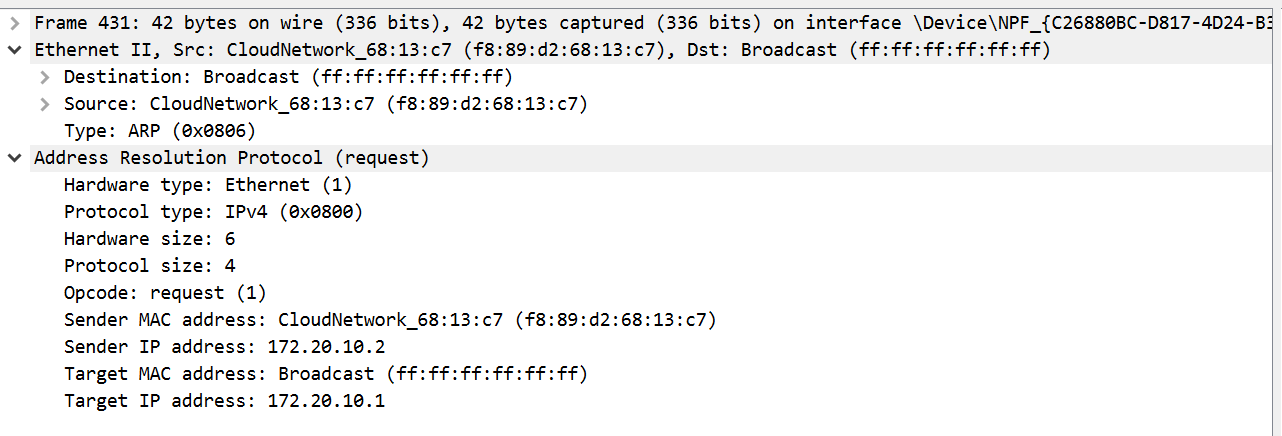




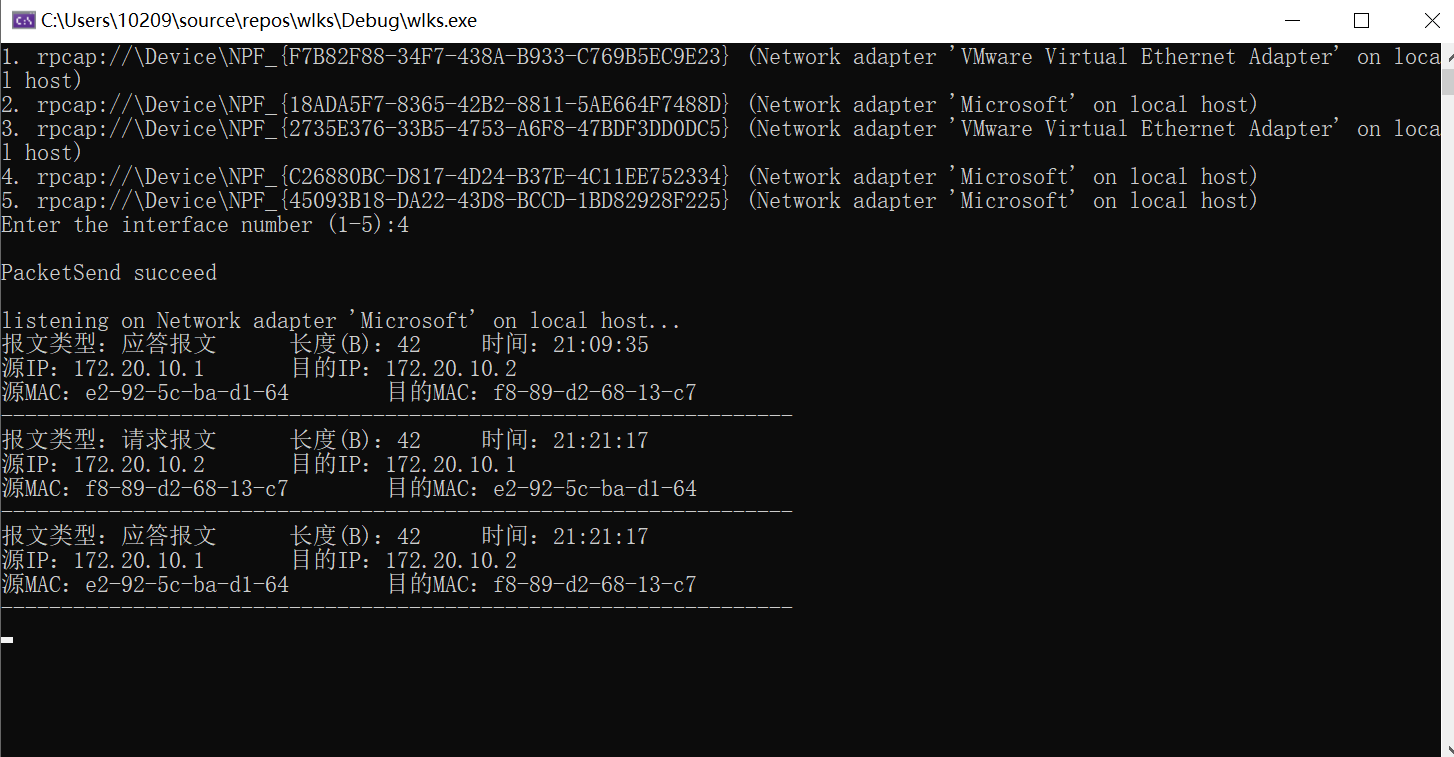
可以看到，选择正确的设备号“4”后立即显示发送数据包成功（PacketSend succeed）以及正在监听中（listening on Network adapter 'Microsoft' on local host...）的提示，并且显示了一条ARP应答报文以及该报文的类型、长度、时间、IP地址、MAC地址等首部信息。

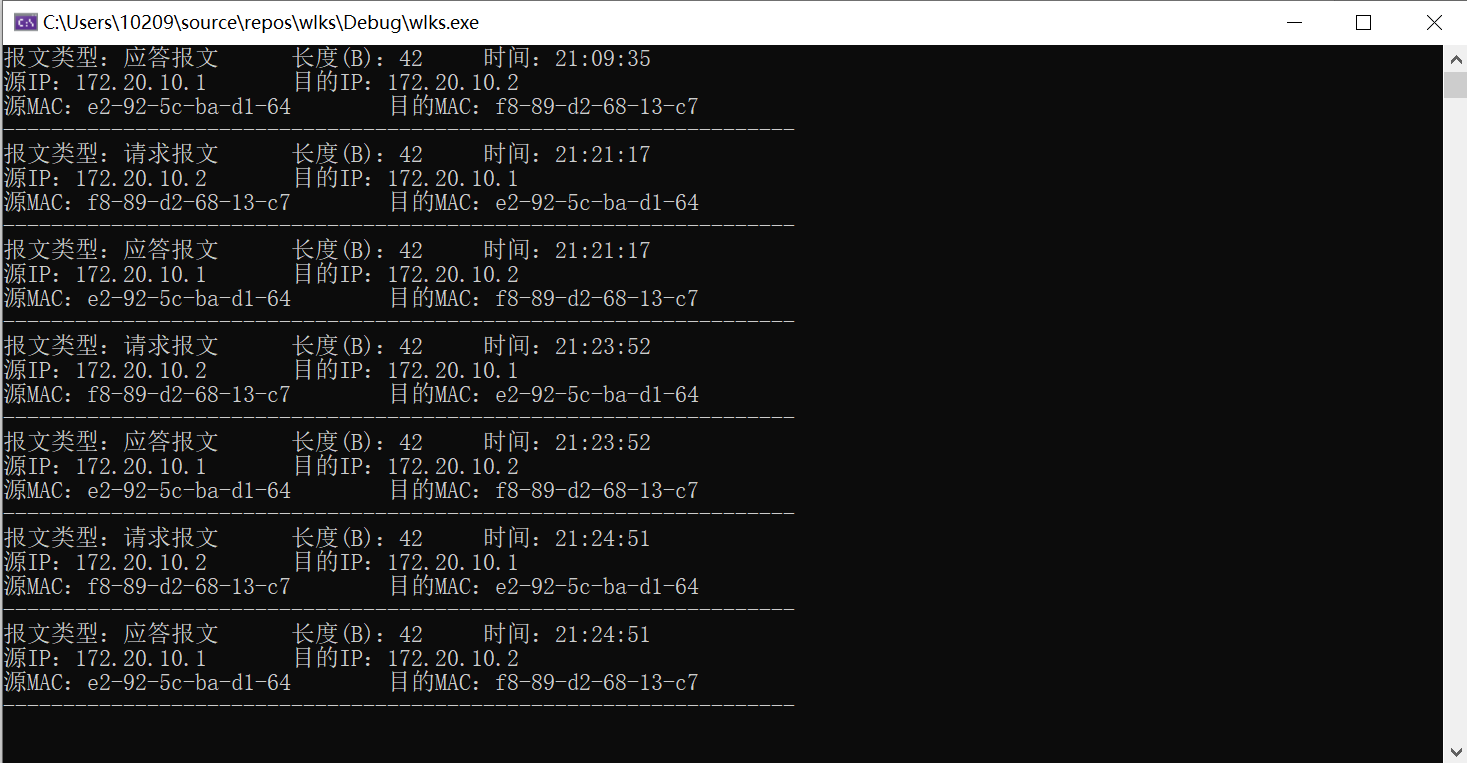


此时再打开Wireshark软件可以看到其成功捕获到了三条ARP报文。并且这些报文的首部信息字段与实际情况一致，如下图：

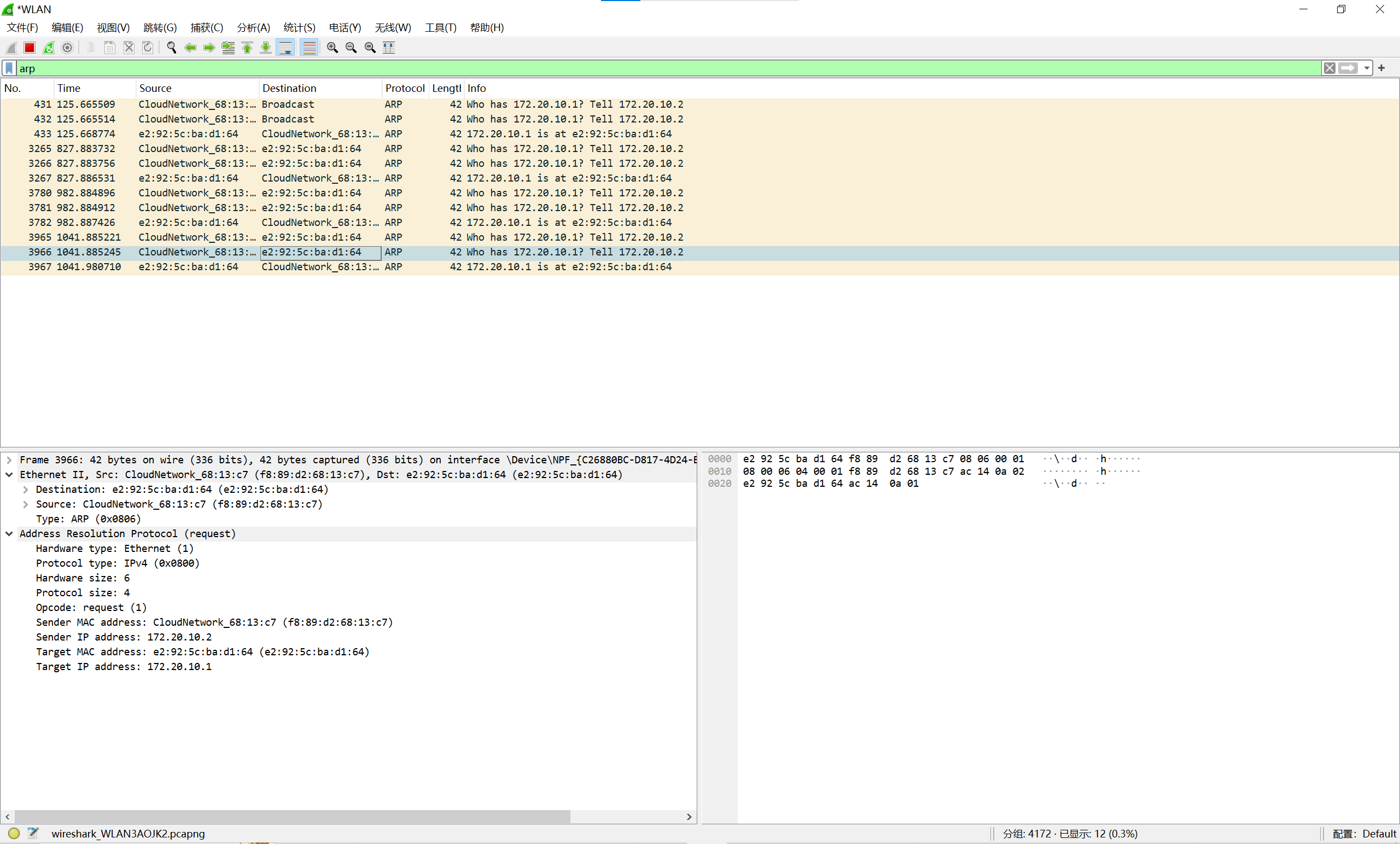


此外，过了一段时间后，本程序还监听输出到了多条其他ARP数据包如下图：





可以看出在程序运行过程中电脑会向手机发送不止一条“ARP请求报文”，当然也会收到多条“ARP应答报文”。再次打开Wireshark软件截图图下：



从上述测试过程可以看出本程序能实现发送ARP数据包并接收、解析ARP数据包的一整套完整流程。

**题目2：路由跟踪小程序**

（1）需求分析

该小程序需要能够解析用户输入的域名或 IP 地址，通过发送一系列具有递增 TTL 值的网络数据包，并监听返回的 ICMP 超时响应，从而追踪并显示到达目标的路径上经过的每个路由器的 IP 地址及其 RTT。同时，需要有良好的异常和错误处理机制，以及用户友好的界面和输出格式。

（2）设计方案

**1、程序功能模块划分：**

**主程序:**

初始化网络环境

解析用户输入的 IP 地址或域名

初始化套接字

执行路由跟踪操作

**解析 IP 地址:**

将用户输入的 IP 地址或域名转换成网络字节序的 IP 地址

**初始化套接字:**

创建原始套接字

设置套接字选项，如超时

**计算校验和:**

计算 ICMP 消息的校验和

**解析 ICMP 响应**

分析接收到的 ICMP 响应消息

计算并输出每一跳的往返时间

**路由跟踪:**

发送 ICMP 回显请求

接收 ICMP 回显应答或超时信息

输出跟踪路径和每一跳的 RTT

**2、模块间的联系：**

主程序负责调用解析IP 地址来处理用户输入，随后通过初始化套接字 准备网络通讯环境，最后调用路由跟踪来执行主要的跟踪任务。

路由跟踪模块在执行过程中，会多次调用计算校验和和解析ICMP响应 来处理网络数据包。

解析ICMP响应需要计算校验和来验证接收到的数据包的完整性。

**3、数据结构：**

IP\_HEADER和ICMP\_HEADER结构体用于封装和解析IP和ICMP协议的头部信息。

DECODE\_RESULT结构体用于存储解析ICMP响应后得到的数据，包括序列号、往返时间、以及发送者的IP地址。

**4、关键问题及其解决方法：**

**校验和计算:**

问题：正确计算 ICMP 消息的校验和，确保数据完整性。

解决方法：通过累加消息中每 16 位的和（同时处理进位），最后取反，得到校验和。

**TTL 设置和增加:**

问题：如何实现让 ICMP 数据包在网络中逐跳前进。

解决方法：通过 setsockopt 函数设置套接字的 IP\_TTL 选项，然后逐步增加 TTL 的值。

**接收超时和错误处理:**

问题：如何处理网络超时和异常情况。

解决方法：设置套接字的接收和发送超时时间，使用 WSAGetLastError 函数识别和处理特定错误代码。

**域名解析：**

问题：将用户输入的域名转换为 IP 地址。

解决方法：使用 gethostbyname 函数进行 DNS 查询，将域名解析为 IP 地址。

（3）程序结构和流程图

**程序结构：**

**1、主函数:**

负责程序的初始化（包括网络环境的设置），用户输入的接收，以及启动路由追踪过程。

**2、IP地址解析:**

将用户输入的IP地址或域名转换为可用于网络通信的32位IP地址。

**3、套接字初始化:**

基于目标IP地址，初始化一个原始套接字用于发送和接收ICMP数据包。

**4、校验和计算:**

计算ICMP数据包的校验和，确保数据在传输过程中的完整性。

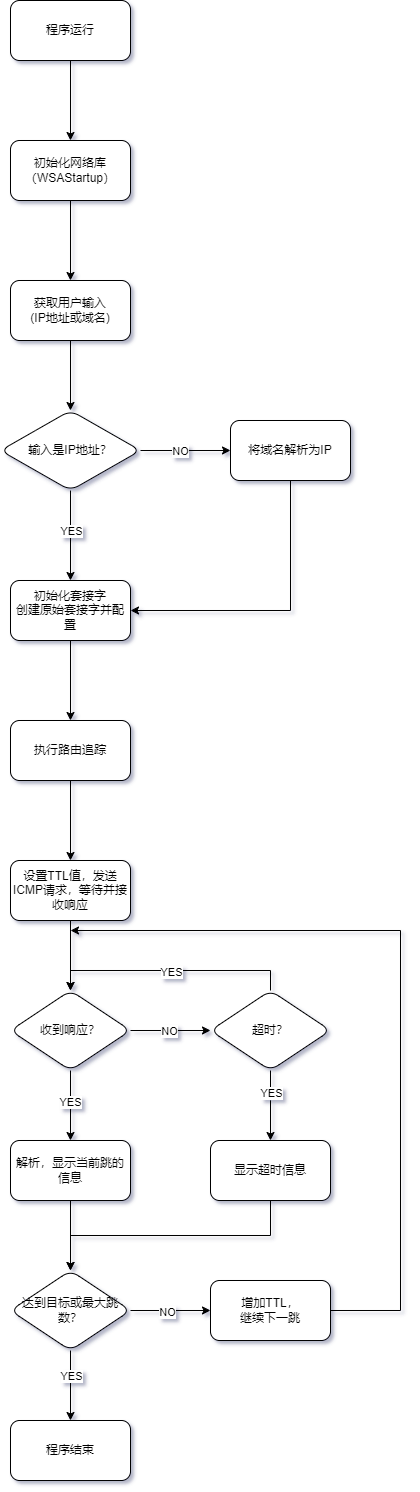
**5、ICMP响应解析：**

解析收到的ICMP响应包，提取相关信息（如序列号、IP地址、RTT等）。

**6、路由追踪:**

实现路由追踪逻辑，包括发送ICMP请求、接收响应、显示路径信息。

**程序流程图：**

****

（4）测试方案

**1、功能测试：**

确保程序能够正确执行其核心功能：追踪并显示从本机到目标地址的路由路径。

测试用例1：追踪到公网IP地址

目的：验证程序能够追踪到达一个公网IP地址的路径。

步骤：选择一个公网IP地址（如8.8.8.8，Google的DNS服务器）作为目标，运行程序。

预期结果：程序输出从本机到该公网IP地址的路由器列表，包括每一跳的IP地址和RTT。

测试用例2：追踪到域名

目的：验证程序能够解析域名并追踪到该域名对应IP地址的路径。

步骤：选择一个已知域名（如www.google.com）作为目标，运行程序。

预期结果：程序首先解析域名为IP地址，然后输出从本机到该IP地址的路由器列表。

**2、边界条件测试**

测试程序在某些极端条件下的表现，确保其稳定性。

测试用例3：追踪到本机地址

目的：验证程序处理目标地址为本机地址时的行为。

步骤：使用本机IP地址或localhost作为目标，运行程序。

预期结果：程序应识别目标地址为本机，直接返回而不进行追踪或显示相应提示。

测试用例4：最大跳数限制

目的：验证程序能正确处理达到最大跳数限制的情况。

步骤：设置一个目标地址，确保到达该地址的路径超过程序设置的最大跳数限制，运行程序。

预期结果：程序在达到最大跳数后停止追踪，并提示用户路径可能更长。

3. 异常和错误处理测试

确保程序能妥善处理各种异常情况，给出清晰的错误提示。

测试用例5：无效的IP地址或域名

目的：验证程序对无效输入的处理。

步骤：输入一个格式错误的IP地址或不存在的域名，运行程序。

预期结果：程序提示输入错误，并要求用户重新输入。

测试用例6：网络不可达

目的：验证当目标网络不可达时程序的行为。

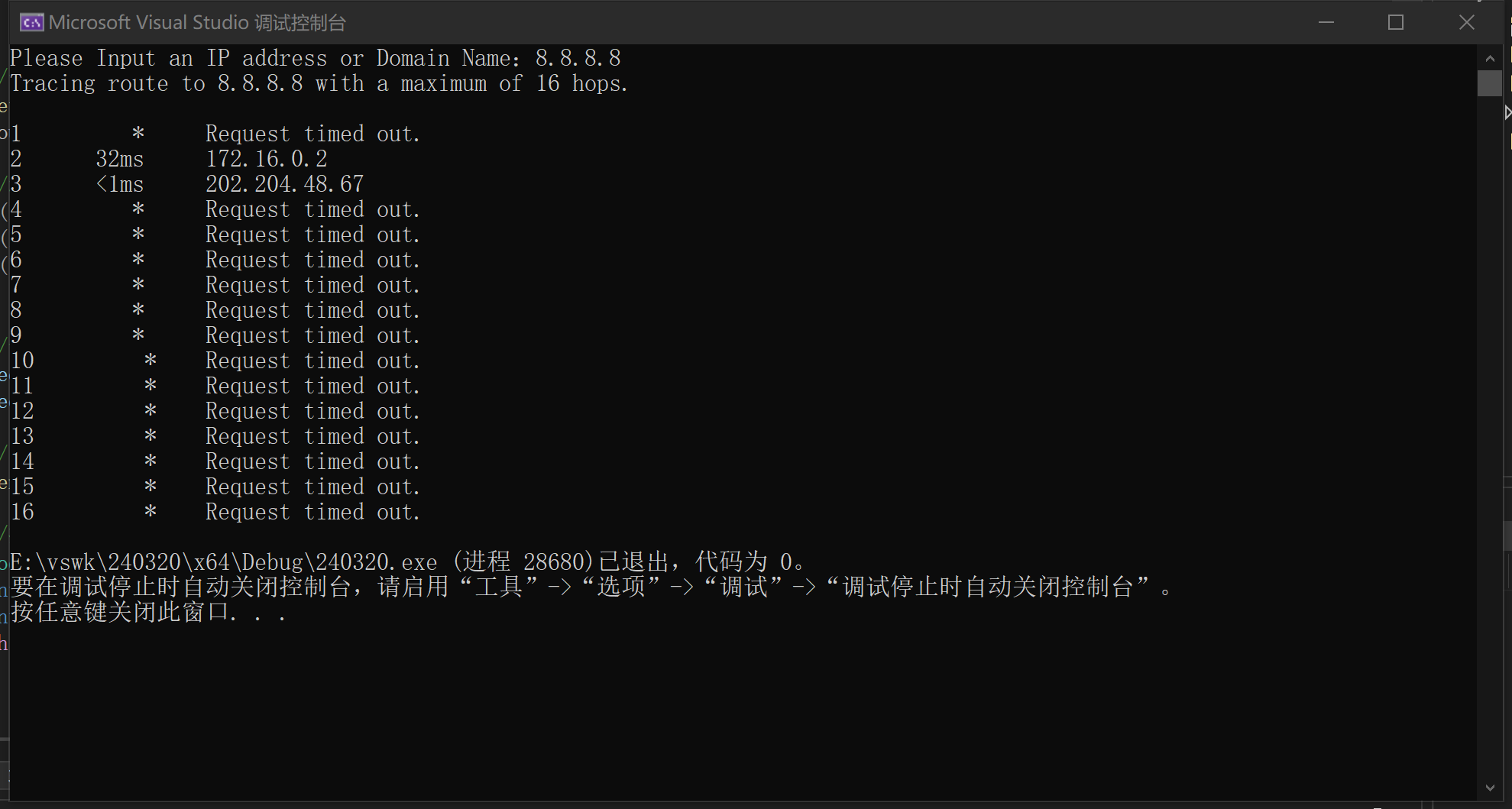
步骤：断开本机网络连接，选择一个公网IP地址作为目标，运行程序。

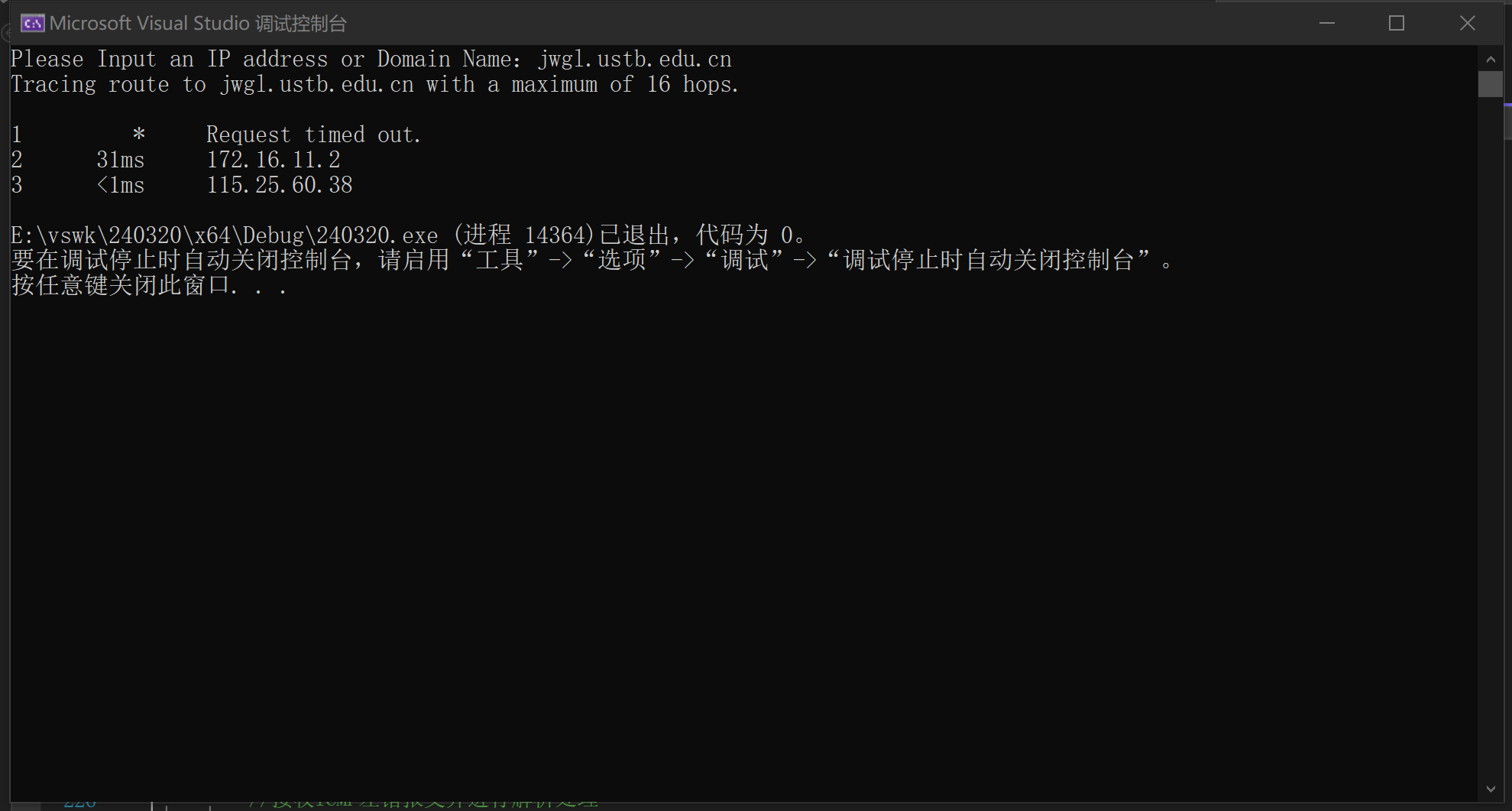
预期结果：程序提示网络不可达或超时。

**实验数据：**

题目1：见其“测试方案”。

题目2：

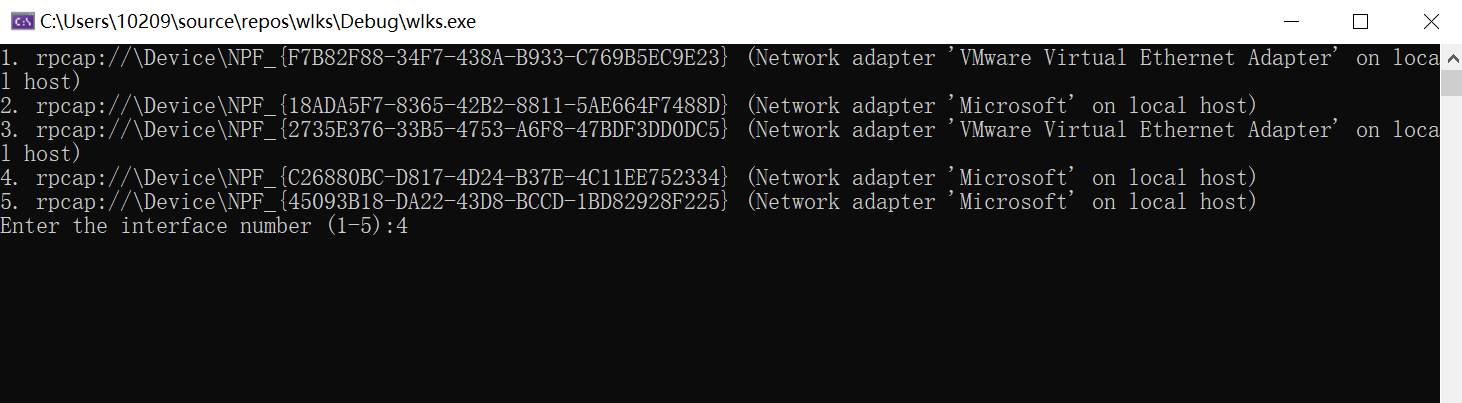


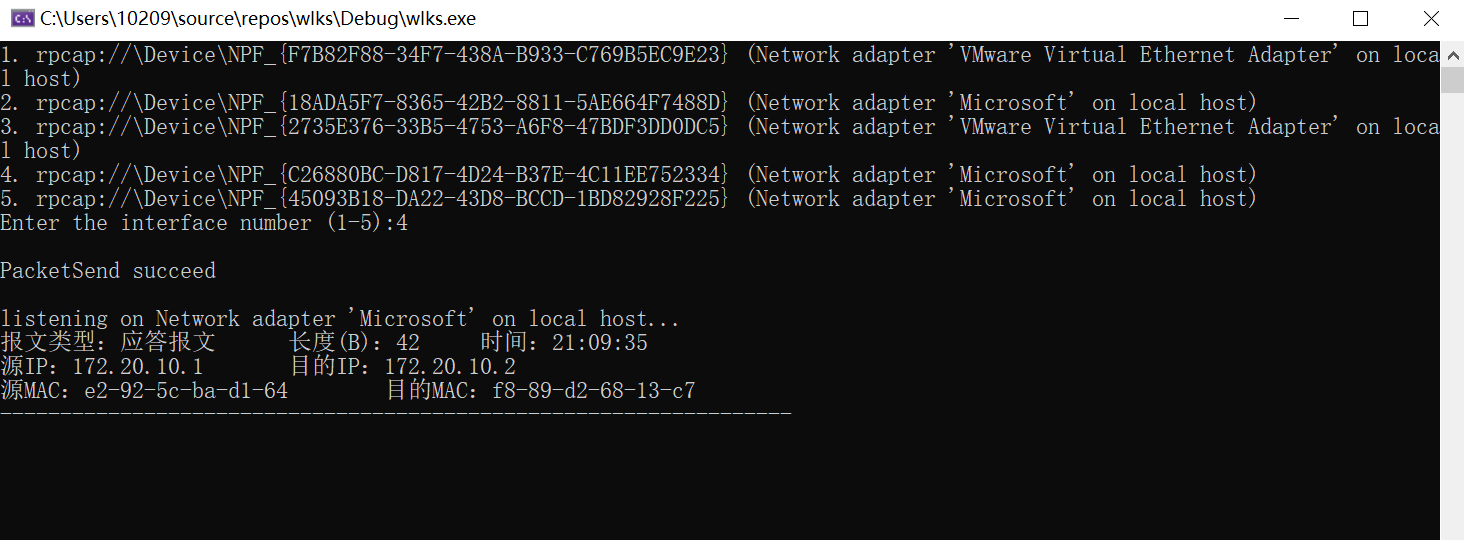


**实验结果与分析：**

**题目1：**

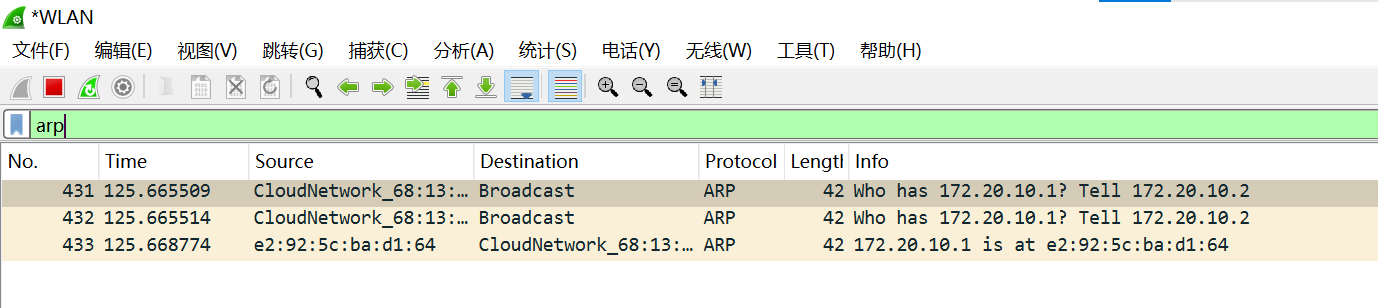
1. 程序运行结果



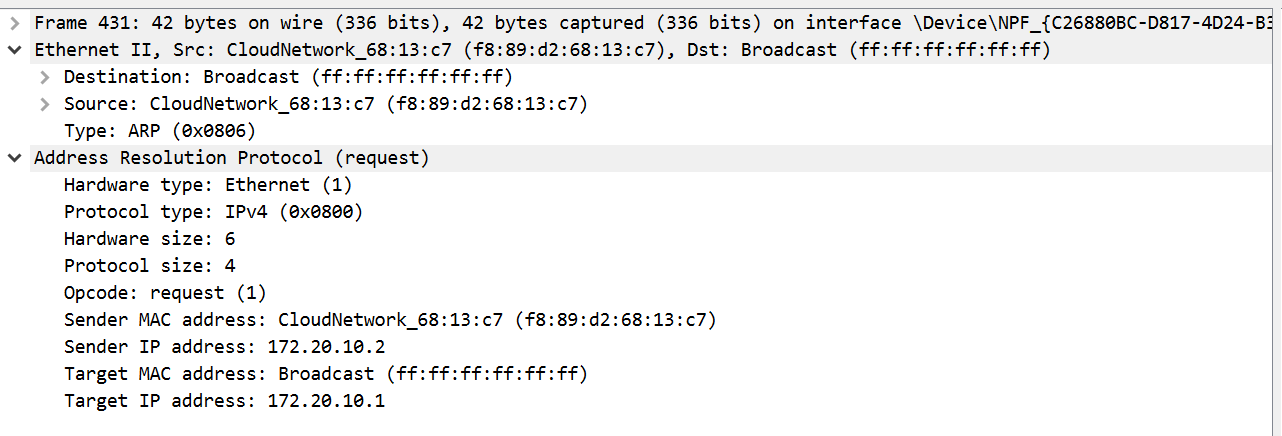


1. 运行结果分析

在选择正确的设备号“4”后立即显示发送数据包成功（PacketSend succeed）以及正在监听中（listening on Network adapter 'Microsoft' on local host...）的提示，并且显示了一条ARP应答报文以及该报文的类型、长度、时间、IP地址、MAC地址等首部信息。



此时再打开Wireshark软件可以看到其成功捕获到了三条ARP报文。并且这些报文的首部信息字段与实际情况一致，如下图：



1. 实验结论

通过本次实验，我们深刻理解和掌握了数据包的封装发送和解析过程，尤其是对ARP、ICMP和TCP协议的实际应用有了直观的认识。实验过程中，使用WinPcap技术或Socket技术根据ARP/ICMP/TCP协议数据单元的结构和封装规则，封装数据帧发送到局域网中，并捕获网络中的ARP/ICMP/TCP数据包，解析数据包的内容，显示结果，并写入日志文件，这一过程增强了我们对网络协议栈工作机制的理解。

通过编写和执行代码，我们学会了如何利用程序直接与网络协议交互，实现低级别的网络通信，这不仅仅是理论知识的学习，更重要的是提高了动手能力和问题解决能力。同时，本实验也加深了我们对网络安全的认识，理解了网络攻击和防御的一些基本原理。

1. 讨论

尽管实验取得了成功，但我们在实验过程中也遇到了一些问题，以下是一些存在的问题及可能的改进方向：

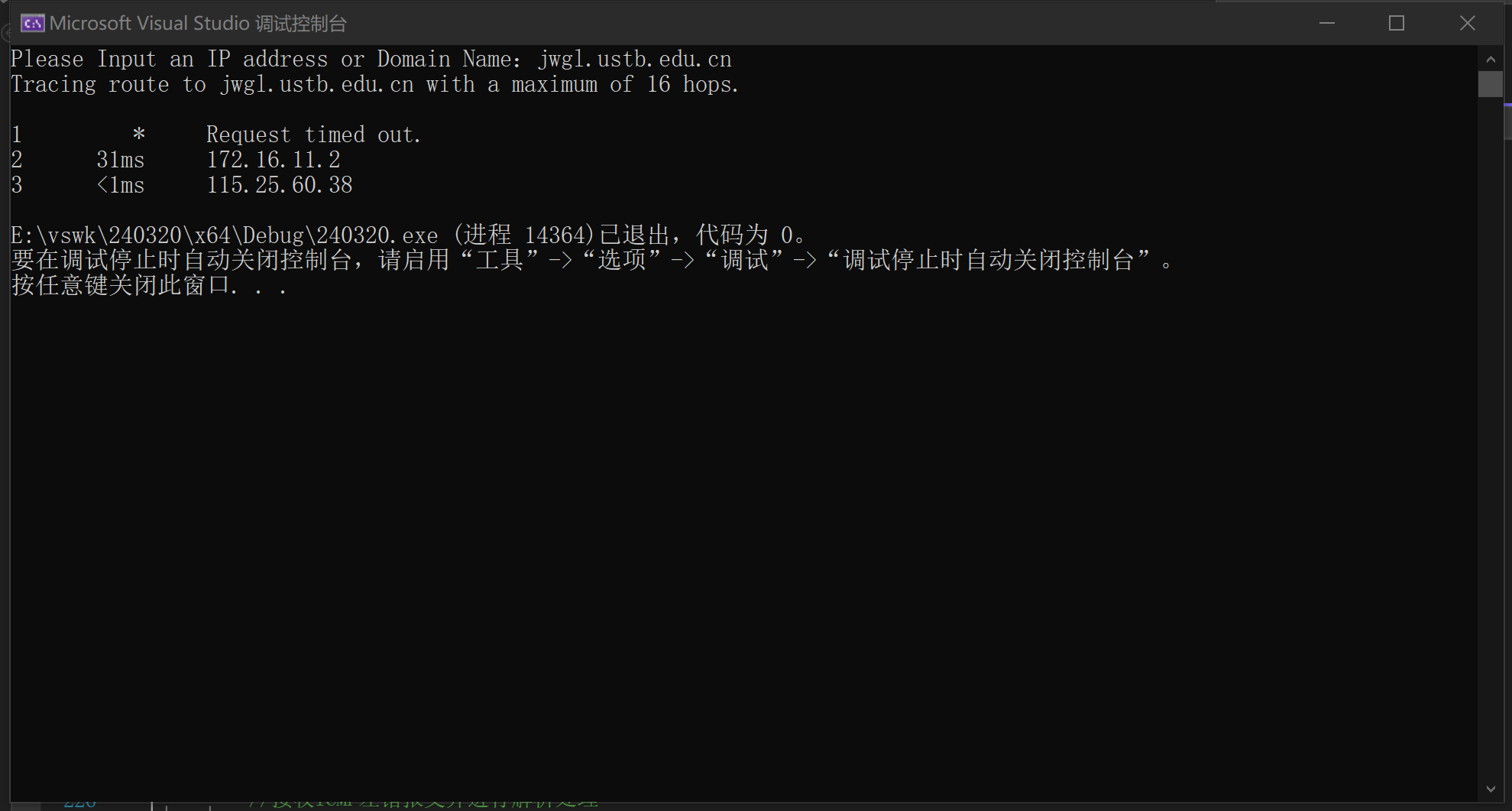
跨平台兼容性问题：由于WinPcap主要针对Windows操作系统，对于使用Linux或MacOS的用户来说，需要寻找相应的替代技术如libpcap或其他。改进方向：提供跨平台的网络编程指导，如使用更加通用的libpcap库，以及如何在不同操作系统上进行相同实验的详细说明。

理论与实践脱节：虽然实验强调了协议的实际应用，但学生可能仍然难以理解协议设计背后的理论基础和原理。改进方向：在实验指导书中加入更多关于网络协议设计原理的讲解，或者通过案例研究的方式，引导学生思考协议设计的理论依据和实际应用之间的联系。

实验结果验证：在发送和接收数据包的过程中，验证实验结果的准确性有时较为困难，特别是在复杂的网络环境下。改进方向：开发或引入更多的网络分析工具和技术，帮助学生更容易地捕获、分析和验证网络通信过程中的数据包。

**题目2：**

1. 程序运行结果



我们使用该路由跟踪小程序来追踪从本机到jwgl.ustb.edu.cn的路径。

（2）运行结果分析

跳数：从上述输出中，我们可以看到到达 jwgl.ustb.edu.cn 总共需要3个跳。每一行表示一跳，显示了通过该路由器的一次测量的往返时间（RTT）。

路由路径：第一跳通常是本地网络的网关，而最后一跳是目标服务器的IP地址。中间的跳数显示了通过ISP和互联网核心网络的路径。

第一跳超时：

在第一跳时，请求超时，这可能意味着起始点（通常是本地网络的网关）未响应ICMP请求。这种情况可能是由于网络配置（如防火墙规则）导致ICMP包被丢弃或不被回应。

后续跳数响应时间：

从第二跳开始，响应时间极低（<1ms），这表明跟踪的后续路由器非常接近源主机，或者网络条件良好，数据包传输效率高。

IP地址分析：

第二跳的IP地址（172.16.11.2）属于私有IP地址范围，这表明该跳仍在本地网络内或连接到本地网络的近距离节点。

第三跳的IP地址（115.25.60.38）是一个公网IP地址，这意味着数据包已经进入到更广阔的互联网环境中。该IP可能属于ISP的路由器或网络接入点。

（3）实验结论

本次实验设计并实现了一个基于 IP 的路由跟踪小程序，它模仿了 tracert 工具的功能，根据输入的 IP 地址或域名，展示了数据包从本机到目标地址所经过的路由器路径，包括各路由器的 IP 地址和往返时间（RTT）。通过实践，我们深入理解了网络基础中的路由跟踪原理，以及如何通过编程实现网络诊断工具的基本功能。这个过程强化了我们对网络协议栈的理解，特别是 IP 和 ICMP 协议的使用和实现。同时，我们也掌握了原始套接字的创建和使用，以及在 Windows 平台下基于 Winsock API 进行网络编程的方法。

（4）讨论

存在的问题：

环境限制：本程序基于 Windows 平台和 Winsock API 实现，其跨平台性较差。在其他操作系统上运行需要相应的适配和修改。

TTL 设置与实际路径：在实际互联网环境中，由于路由器的负载均衡等因素，相同的 TTL 值可能会通过不同的路径，这可能导致程序输出的路径与实际路径有所不同。

超时处理：当前实现中，如果一个跳点无响应，程序将输出“\* Request timed out.”，但这可能不足以反映所有网络异常情况。

可能的改进方向：

增强跨平台性：可以考虑使用跨平台的网络编程库，如 Boost.Asio 或 Qt Network，来提高程序的可移植性。

路径稳定性分析：引入机制分析多次执行时路径的变化，提供路径稳定性的评估。

增强错误处理和诊断能力：除了简单的超时响应，还可以通过分析 ICMP 错误消息，提供更多网络故障的诊断信息，如链路断开、网络不可达等。

图形用户界面（GUI）：可以开发一个图形用户界面，使得非技术用户也能方便地使用这个工具，同时在 GUI 中展示更为直观的路由路径图。

**重要程序代码：**

**问题1：**

1. ARP数据包的封装与发送

void sent\_arp\_packet() {

unsigned char sendbuf[42]; // ARP包的固定大小为42字节

unsigned char mac[6] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55 }; // 示例MAC地址

unsigned char ip[4] = { 0xC0, 0xA8, 0x01, 0x01 }; // 示例IP地址，192.168.1.1

EthernetHeader eh;

ArpHeader ah;

// 设置以太网头部

memset(eh.DestMAC, 0xff, 6); // 目标MAC地址设置为广播地址

memcpy(eh.SourMAC, mac, 6); // 源MAC地址

eh.EthType = htons(ETH\_ARP); // 设置帧类型为ARP

// 设置ARP头部

ah.hdType = htons(ARP\_HARDWARE);

ah.proType = htons(ETH\_IP);

ah.hdSize = 6;

ah.proSize = 4;

ah.op = htons(ARP\_REQUEST); // ARP请求操作

memcpy(ah.smac, mac, 6); // 源MAC地址

memcpy(ah.sip, ip, 4); // 源IP地址

memset(ah.dmac, 0x00, 6); // 目标MAC地址，ARP请求中通常未知

memset(ah.dip, 0x00, 4); // 目标IP地址，需要查询的IP地址

// 构建ARP请求包

memcpy(sendbuf, &eh, sizeof(EthernetHeader));

memcpy(sendbuf + sizeof(EthernetHeader), &ah, sizeof(ArpHeader));

// 发送ARP请求

if (pcap\_sendpacket(adhandle, sendbuf, 42) != 0) {

fprintf(stderr, "\nError sending the packet: %s\n", pcap\_geterr(adhandle));

}

}

2. 设备初始化与网络监听设置

void device\_init() {

// 获取并列出所有网络设备

if (pcap\_findalldevs\_ex(PCAP\_SRC\_IF\_STRING, NULL, &alldevs, errbuf) == -1) {

fprintf(stderr, "Error in pcap\_findalldevs: %s\n", errbuf);

exit(1);

}

// 遍历设备列表，让用户选择一个网络设备进行监听

// 此部分省略具体选择设备的代码，假设用户已选择适配器d

// 打开选中的网络设备进行监听

adhandle = pcap\_open(d->name, 65536, PCAP\_OPENFLAG\_PROMISCUOUS, 1000, NULL, errbuf);

if (adhandle == NULL) {

fprintf(stderr, "Unable to open the adapter. %s is not supported by WinPcap\n", d->name);

// 释放设备列表

pcap\_freealldevs(alldevs);

exit(1);

}

}

**问题2：**

1. 计算校验和

//计算校验和函数

USHORT checksum(USHORT\* pBuf, int iSize)

{

unsigned long cksum = 0;

while (iSize > 1)

{

cksum += \*pBuf++;

iSize -= sizeof(USHORT);

}

if (iSize)

{

cksum += \*(UCHAR\*)pBuf;

}

cksum = (cksum >> 16) + (cksum & 0xffff);

cksum += (cksum >> 16);

return (USHORT)(~cksum);

}

此函数用于计算 ICMP 报文的校验和，这是确保数据在传输过程中完整性的一种方式。

2. ICMP 报文解析

//对ICMP数据包进行解析

BOOL DecodeIcmpResponse(char\* pBuf, int iPacketSize, DECODE\_RESULT& DecodeResult, BYTE ICMP\_ECHO\_REPLY, BYTE ICMP\_TIMEOUT)

{

// 这里的代码省略了一部分初始化和检查逻辑...

if (pIcmpHdr->type == ICMP\_ECHO\_REPLY || pIcmpHdr->type == ICMP\_TIMEOUT)

{

//立刻输出往返时间信息

if (DecodeResult.dwRoundTripTime)

cout << " " << DecodeResult.dwRoundTripTime << "ms" << flush;

else

cout << " " << "<1ms" << flush;

}

return true;

}

这部分代码负责解析接收到的 ICMP 数据包，提取重要信息，例如 ICMP 类型、序列号等，并计算往返时间。

3. IP 地址解析

u\_long resolveIPAddress(char\* ipAddress) {

// 将点分十进制的 IP 地址转换为 32 位整数

u\_long DestIP = inet\_addr(ipAddress);

if (DestIP == INADDR\_NONE)

{

// 如果转换失败，则尝试将其作为域名进行解析

hostent\* pHostent = gethostbyname(ipAddress);

if (pHostent) {

DestIP = (\*(in\_addr\*)pHostent->h\_addr).s\_addr;

}

else {

cout << "Invalid IP address or Domain Name!" << endl;

WSACleanup();

exit(1);

}

}

return DestIP;

}

这段代码尝试将用户输入的字符串（IP 地址或域名）转换为 32 位的 IP 地址，如果输入的是域名，则通过 DNS 解析获取对应的 IP 地址。

4. 主循环：路由跟踪逻辑

void Traceroute() {

// 初始化和设置 ICMP 报文...

USHORT usSeqNo = 0; // ICMP报文序列号

int iTTL = 1; // 初始 TTL 值

BOOL bReachDestHost = FALSE; // 到达目的主机的标志

while (!bReachDestHost && iTTL <= DEF\_MAX\_HOP)

{

// 设置 IP 报头的 TTL 字段

setsockopt(sockRaw, IPPROTO\_IP, IP\_TTL, (char\*)&iTTL, sizeof(iTTL));

// 发送 ICMP 回显请求报文...

// 接收 ICMP 回显回复或超时信息...

iTTL++; // 递增 TTL 值以探测下一个跳跃点

}

}

这是 Traceroute 程序的核心循环，负责逐步增加 TTL 值，发送 ICMP 请求，并接收回复或超时信息，直到到达目的地或达到最大跳数。