****

**中国地质大学（武汉）**

**计算机网络与工业互联网课程报告**

**学 院：\_\_\_\_\_\_\_\_\_自动化\_\_\_\_\_\_\_**

**课 程：计算机网络与工业互联网**

**指导老师：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**学 号：\_\_\_\_\_\_\_\_\_\_\_\_**

**班 级：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

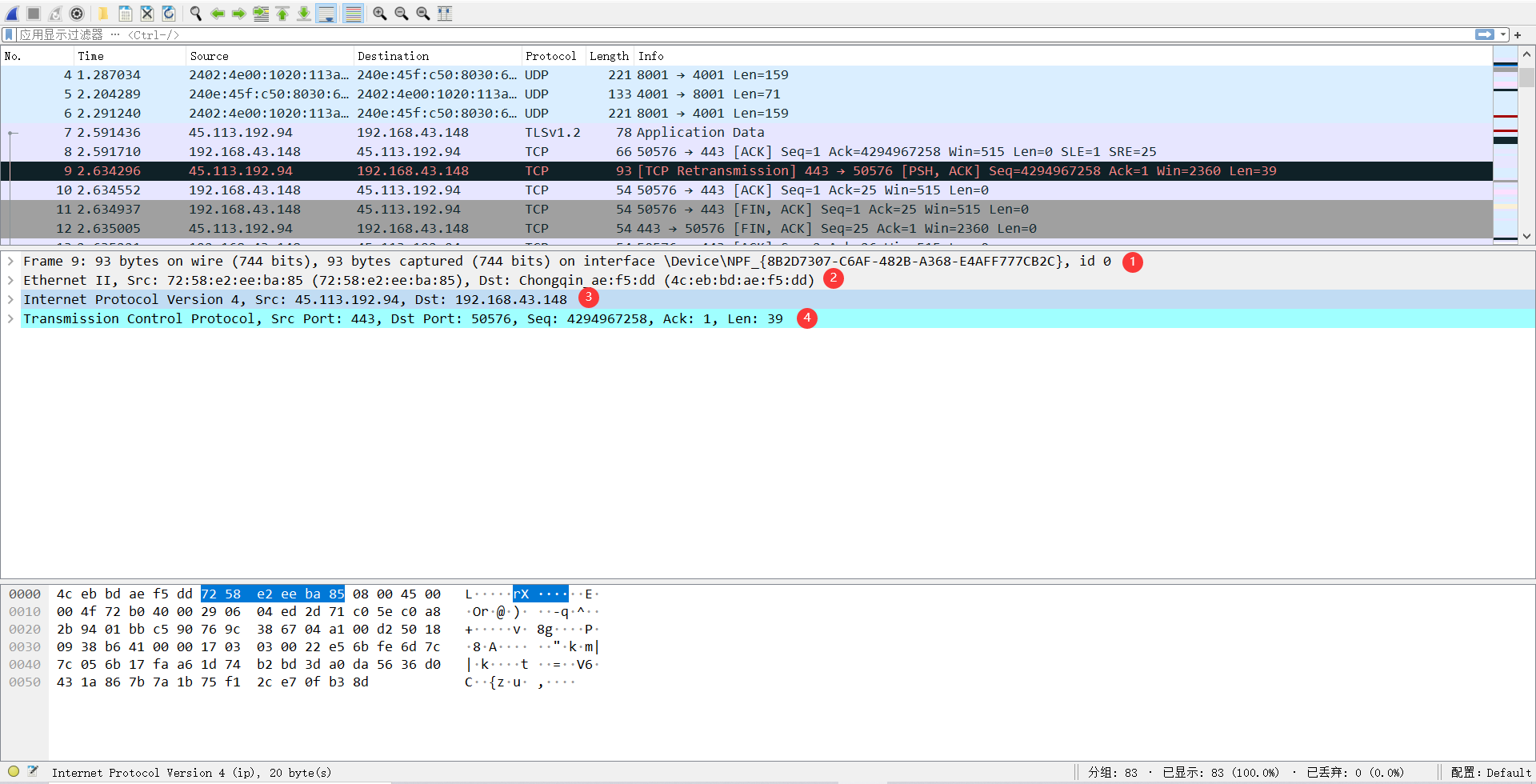
**姓 名：\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_**

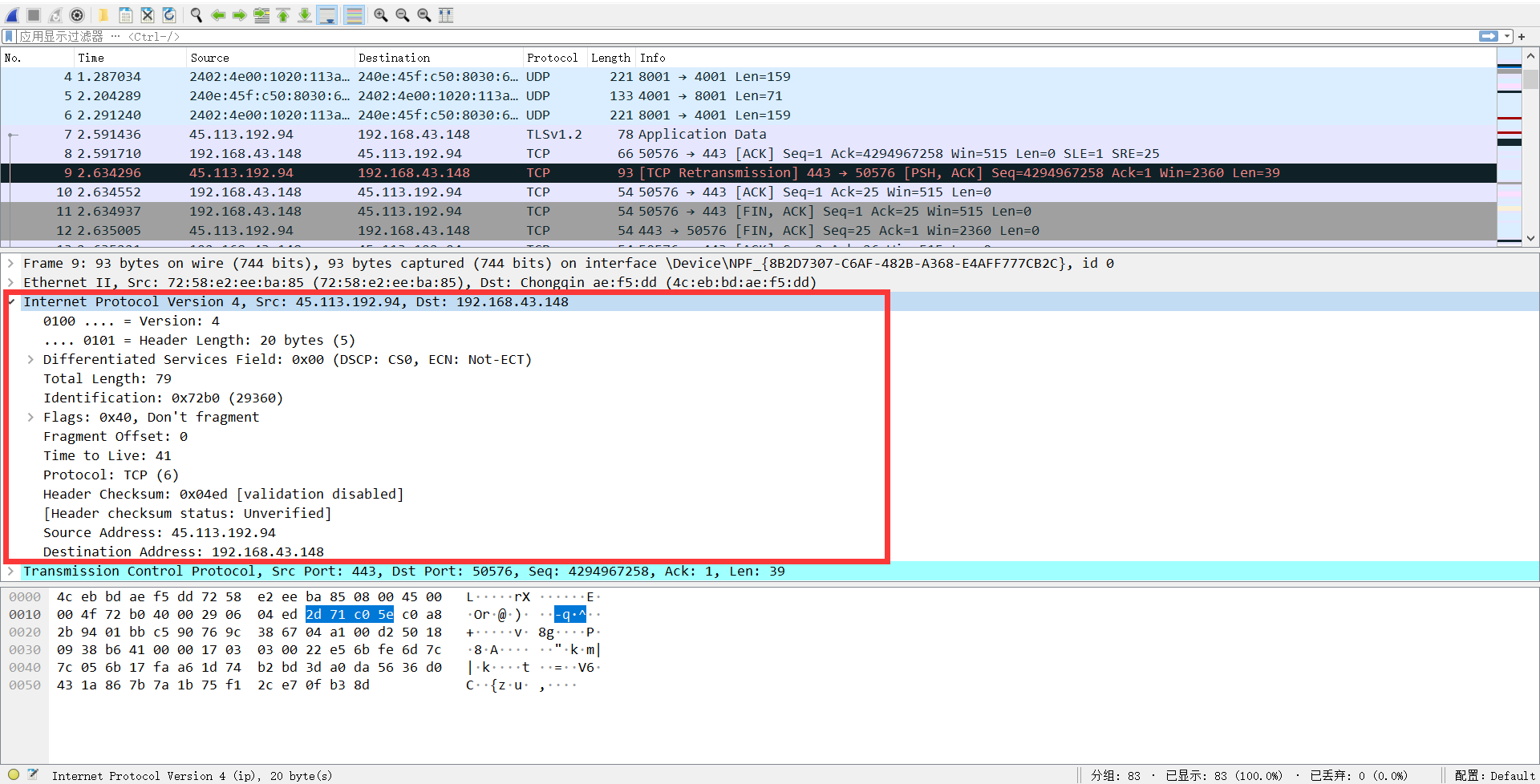
**二〇二一年十二月五日**

**一、实验内容**

**实验一 协议与数据包分析实验**

**1、IP数据报的报文结构如何？请打印截图贴在作业本上，并简要分析。**





打开wireshark，选择WLAN，开始捕获分组。

**屏幕从上到下分为三个区域：**

最上面是Packet List面板，用于显示每个数据帧的摘要。采用表格的形式列出了当前捕获文件中的所有数据包，包括了数据包序号、捕获的相对时间、源地址和目标地址、协议以及在数据包中找到的概况信息等。

中间是Packet Details面板，这里分层次地显示了一个数据包中的内容，并且可以通过展开或是收缩来显示这个数据包中所捕获的全部内容。

最下面是Packet Bytes面板，以十六进制和ASCII码的形式显示数据包的内容。

在Packet List面板中单击选择一条使用TCP协议的报文。

在Packet Details面板显示的4行内容从上到下依次是：

①Frame（物理层）: 物理层的数据帧概况

②Ethernet II（链路层）: 数据链路层以太网帧首部信息

③Internet Protocol Version 4（网络层）: 互联网层IP数据报首部信息

④Transmission Control Protocol（传输层）: 传输层TCP报文的首部信息。

点击展开“Internet Protocol Version 4”栏，可以看到IP数据报首部的详细信息：

版本：0100(version: 4，表示是IPV4)；

首部长度：0101(5\*4 byte，表示首部共20字节)；

区分服务：00000000；

总长度：00000000 01001111 (IP数据报总长度为79 bytes，包括首部和数据部分)；

标识：01001010 01111010（使分片后的各数据报片最后能正确地重装成为原来的数据报）；

标志：010(最低位MF=0表示后面再没有分片了，中间位DF=1表示不能分片)；

片偏移：0（以8个字节为偏移单位，表示分片在原数据报中的相对位置，这里没有分片，这个包就是第一个，故片偏移=0）；

生存时间：0010 1001（表示该数据报能在互联网中经过的路由器最大数量是41）；

协议：00000110(TCP)；

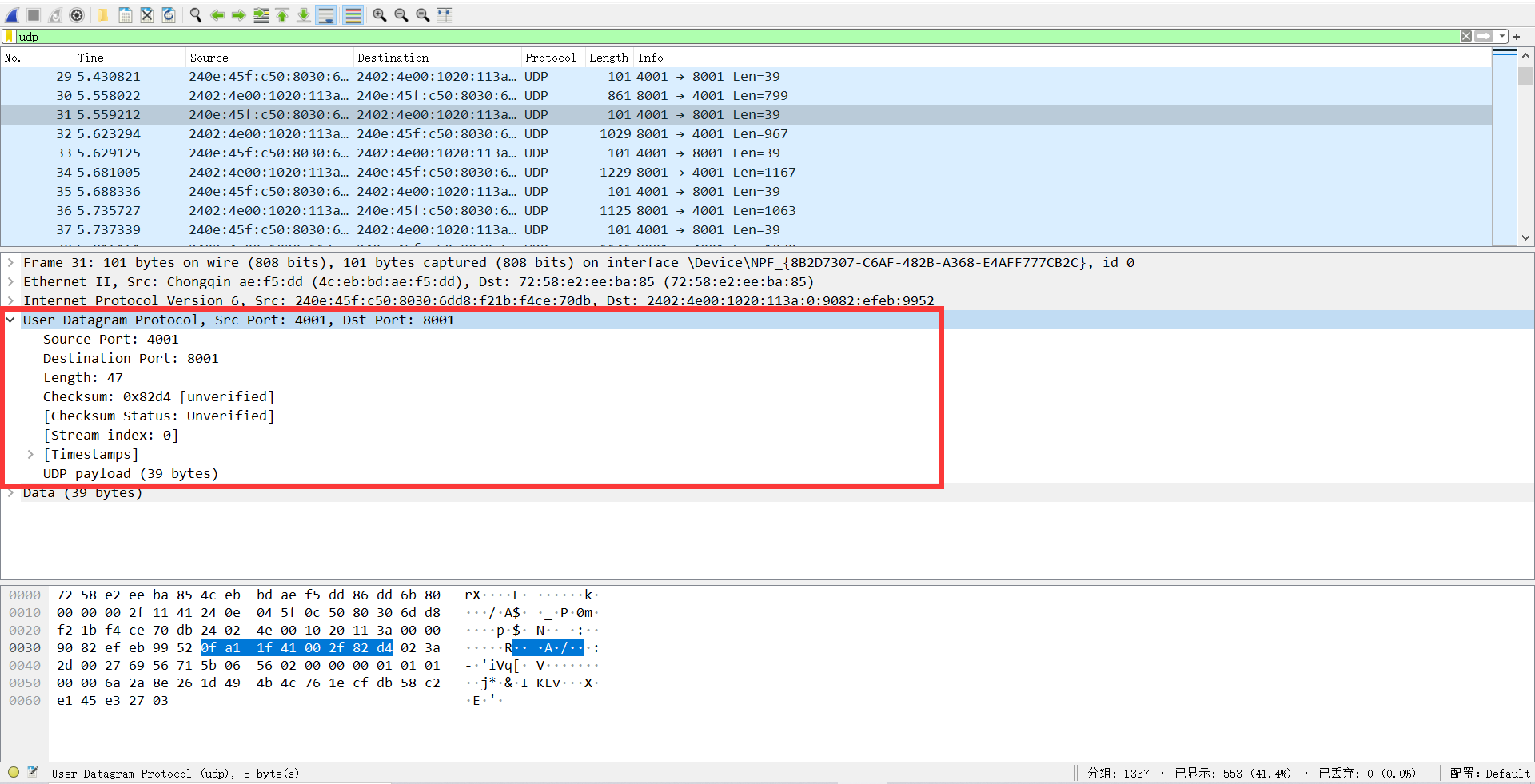
首部检验和：11100011 10001101（IP协议只检验IP数据报首部，不检验数据部分）；

源地址：45.133.192.94；

目的地址：192.168.43.148（本机的IP地址，显然是个专用IP）

**2、UPD和TCP协议数据包中，UDP和TCP的首部如何？请打印截图贴在作业本上，并简要分析。**

UDP：



使用过滤器筛选出UDP协议的报文，点击展开查看UDP首部：

源端口：4001

目的端口：8001

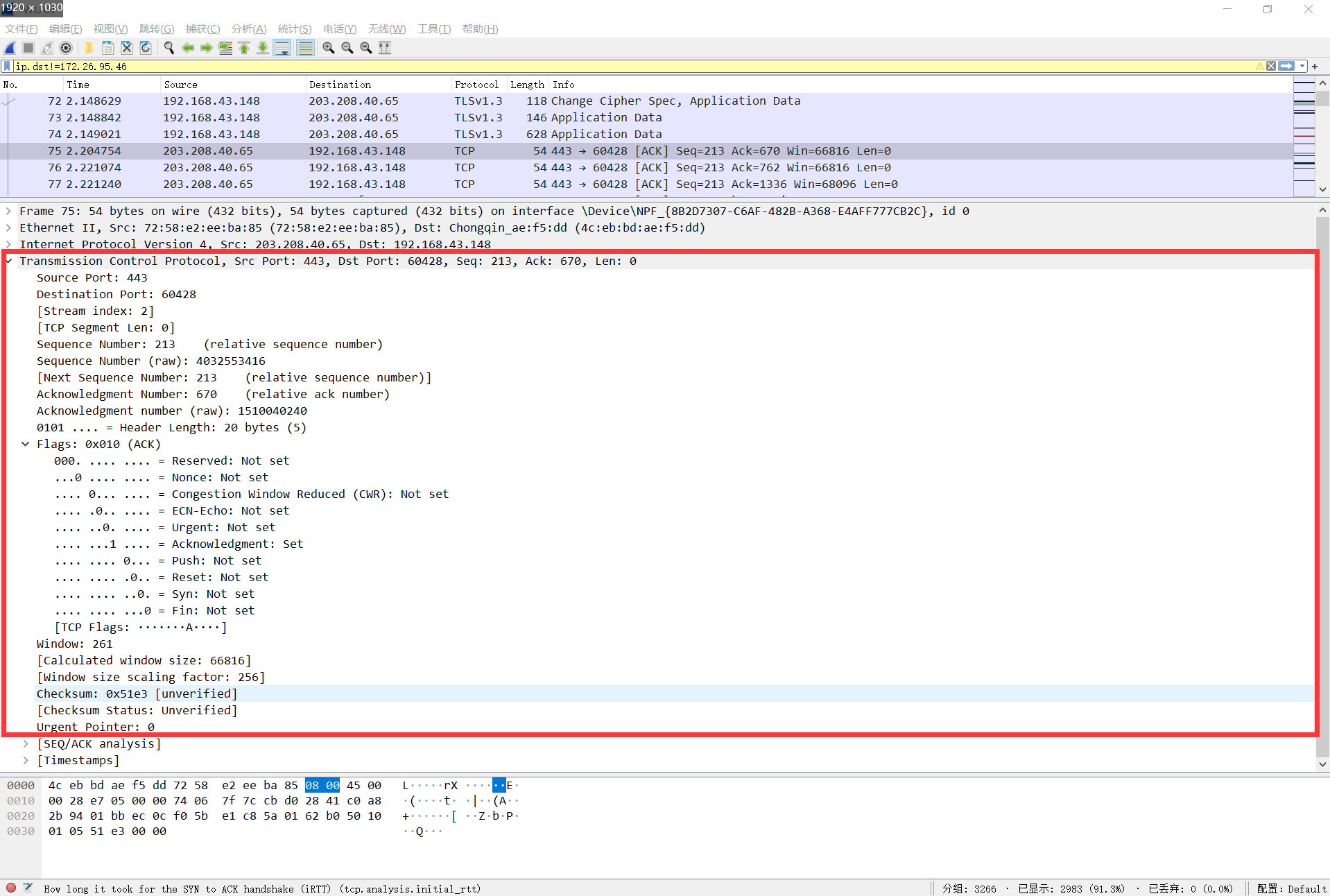
长度：47（UDP报文总长47 bytes）

校验号：0x82d4（把首部和数据部分一起检验，如果有错就会丢弃）

UDP报文由首部和后面的数据部分组成，首部长度只有8字节，开销小。UDP协议相比IP协议增加了复用、分用和差错检测的功能，它在传送数据之前不需要建立连接，也不进行可靠传输，对方在收到 UDP 报文后，不需要给出任何确认。它对应用层交下来的报文既不拆分也不合并，而是保留这些报文的原边界。

虽然UDP不提供可靠交付，但对于像IP电话、在线视频等一些对低时延要求高而对少量数据丢失要求不严苛的场景，UDP比TCP更合适。

TCP：



TCP传送的数据单元是报文段，一个TCP报文段分为首部和数据两部分。TCP首部的前20字节是固定的，还有可选的“选项”部分（0~40bytes），故TCP首部的长度为20~40bytes。选择一个TCP报文，其首部具体内容如下：

源端口：443；

目的端口：60428；

序号：213（本报文段的第一个字节的数据在原字节流中的序号）；

确认号：670（告诉对方自己希望收到的下一个报文段的第一个数据字节的序号是670，序号670之前的数据都已正确收到）；

数据偏移：0101（指出TCP报文段的首部长度为4\*5bytes=20bytes）；

保留：000000（占6位）；

URGent：0（紧急指针无效）；

ACK：1（使确认号有效）；

PSH：0

RST：0

SYN：0（这既不是连接请求报文也不是连接接受报文）；

FIN：0（不释放连接）；

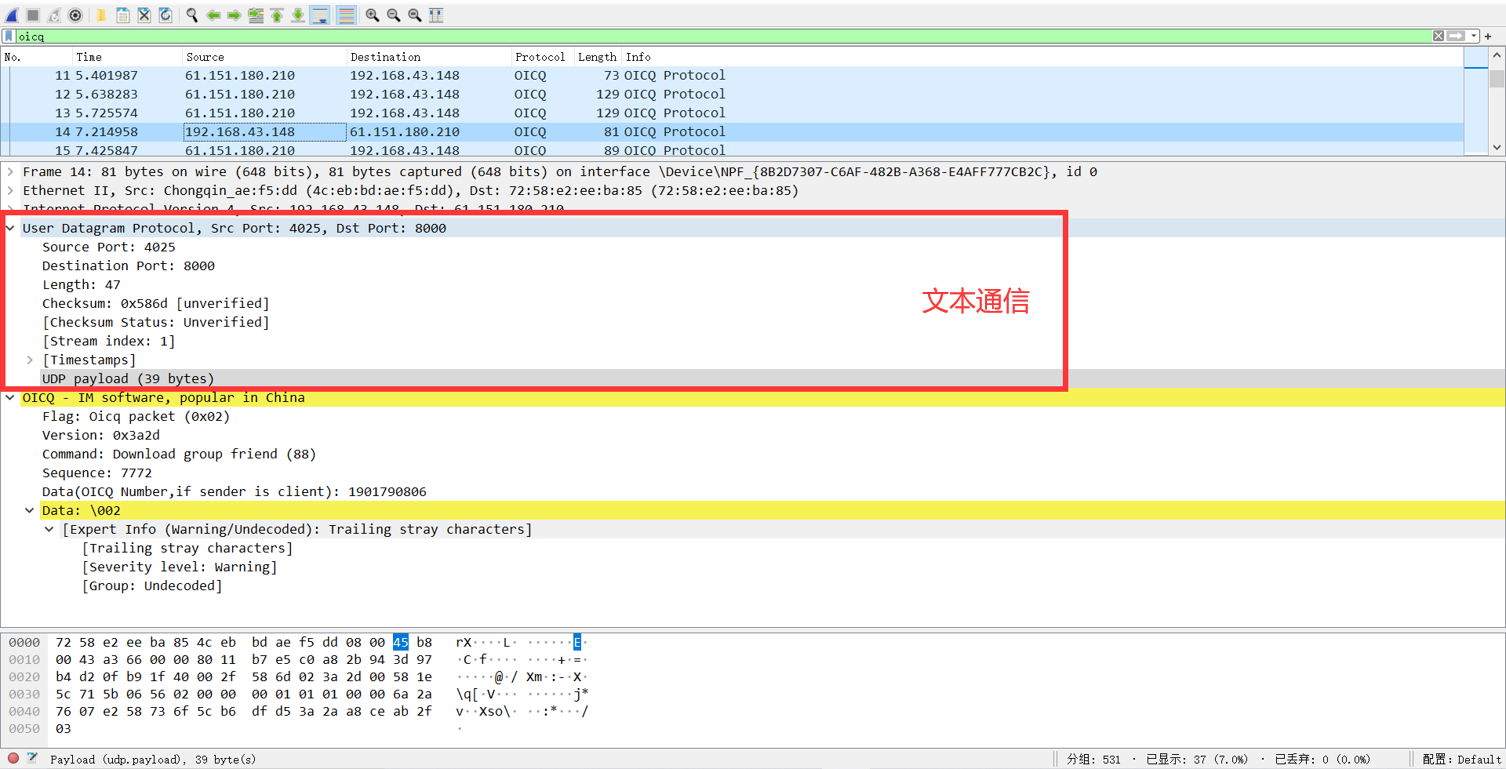
窗口：261（告诉对方，自己的缓存还可接收261字节的数据）；

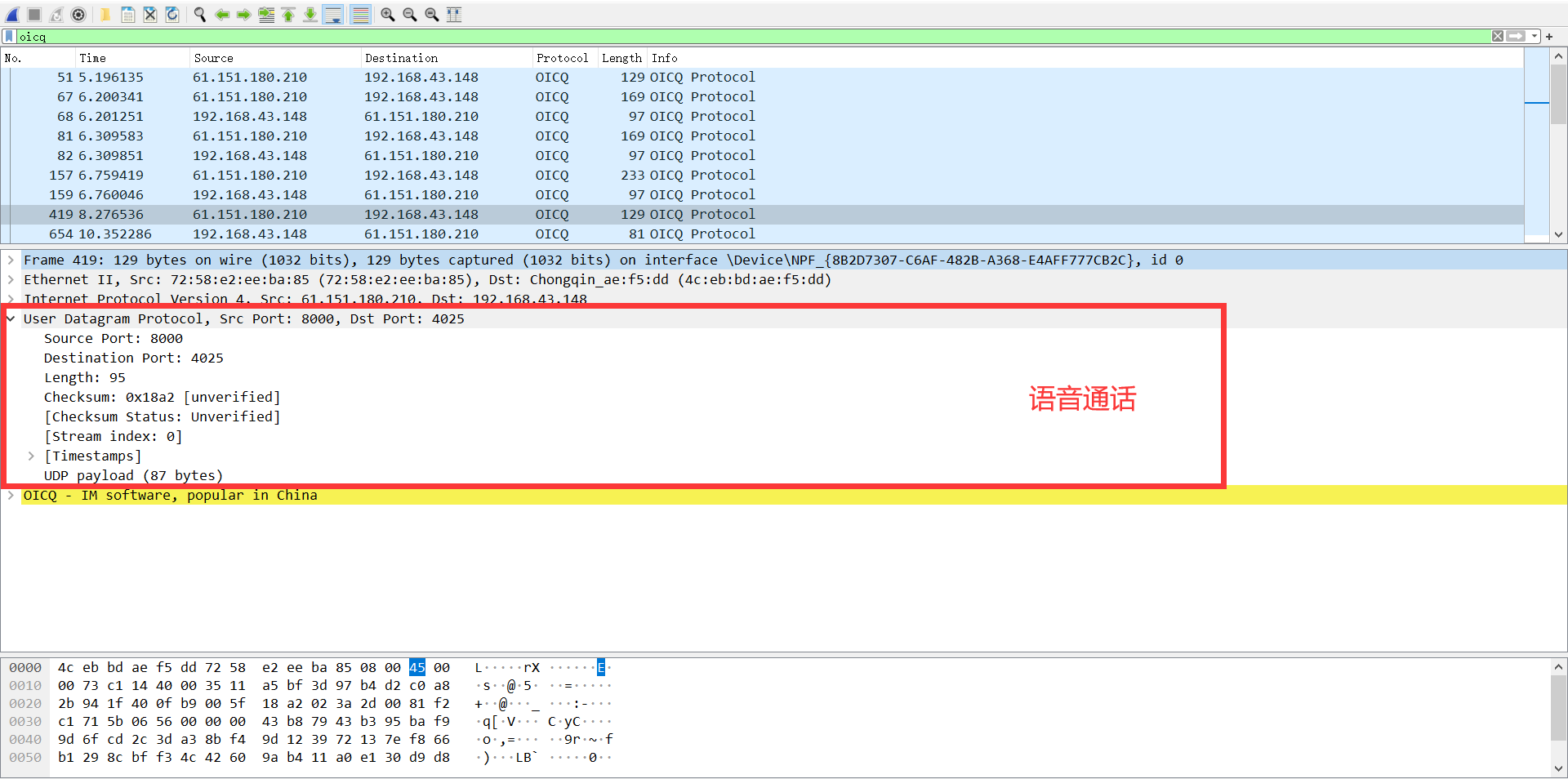
检验和：0x51e3（TCP协议也是把数据和首部一起检验）；

紧急指针：0



**3、使用QQ进行通信时，进行文本和视频通信所使用的端口号分别是多少？使用的传输层协议分别是什么？请打印截图贴在作业本上，并简要分析。**





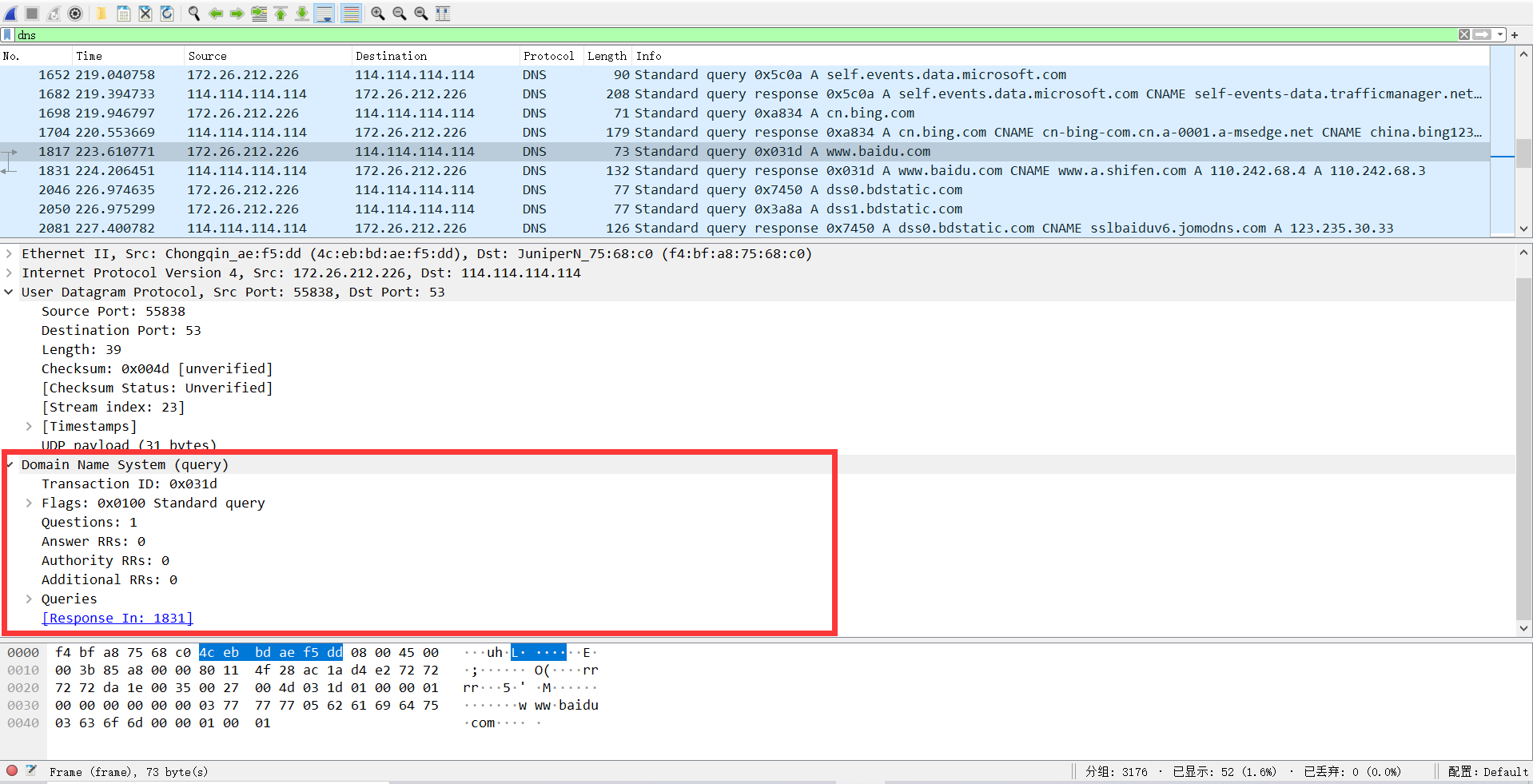
可以看出，无论是文本通信还是语音通话，QQ都是用的UDP协议，端口号都是8000。

QQ使用的服务器分为三类：UDP 8000端口类18个：速度最快，服务器最多。QQ上线会向这些服务器发送UDP数据包，选择回复速度最快的一个作为连接服务器。TCP HTTP连接服务器5个：使用HTTP 80 和443端口连接。

查阅资料得知，其实TCP和UDP协议QQ都有使用，但是以UDP为主，因为UDP传输速度快，简单的说，QQ登录，发消息聊天都是UDP的，只有对方不在线或者隐身时，你所发送的消息是TCP的。双方在线时，直接对话，用UDP，UDP协议头中包含对方的IP地址，这也是有些功能版QQ能显示对方IP的原因。而TCP协议，如果对方隐身，你通过TCP建立和腾讯服务器的连接，把消息存在服务器上，再由服务器选择合适时机传送给对方。

TCP协议要保证可靠传输、要建立和释放连接，还要进行流量控制、拥塞控制，因此不可避免变得复杂、所需开销大；而UDP简单开销小、速度快，两者优势互补，能更好的满足多种通信需求。

4、（选做）分析一下其他软件所使用的应用层协议（如HTTP等，数量不限）的数据包的结构，请打印截图贴在作业本上，并简要分析。



以DNS协议为例进行分析。

在浏览器的地址栏输入www.baidu.com按下回车，发生以下过程：

①运行ARP协议解析网关的MAC地址

②访问DNS服务器，解析得到域名baidu.com对应的IP地址

③访问这个IP地址的网站，建立TCP连接（三报文握手）

④正式传送数据，我们在浏览器上看到网页内容被加载了出来

DNS 报文分为查询请求和查询响应，二者的结构基本相同。上图所选的是一条我的主机发给DNS服务器的查询请求报文，在运输层使用UDP协议，源端口号55838，目的端口号53。该DNS报文包括以下内容：

事务ID：0x031d（DNS 报文的ID标识。对于请求报文和其对应的应答报文，该字段的值是相同的。通过它可以区分 DNS 应答报文是对哪个请求进行响应的）

标志：0x0100（QR字段值为0, 因为是一个请求包；Opcode字段值为0, 因为是标准查询；TC字段值为0，表示没有被截断。RD字段值为1，表示执行递归查询）

问题计数：1（DNS 查询请求的数目为1个）

回答资源记录数：0

权威名称服务器计数：0

附加资源记录数：0

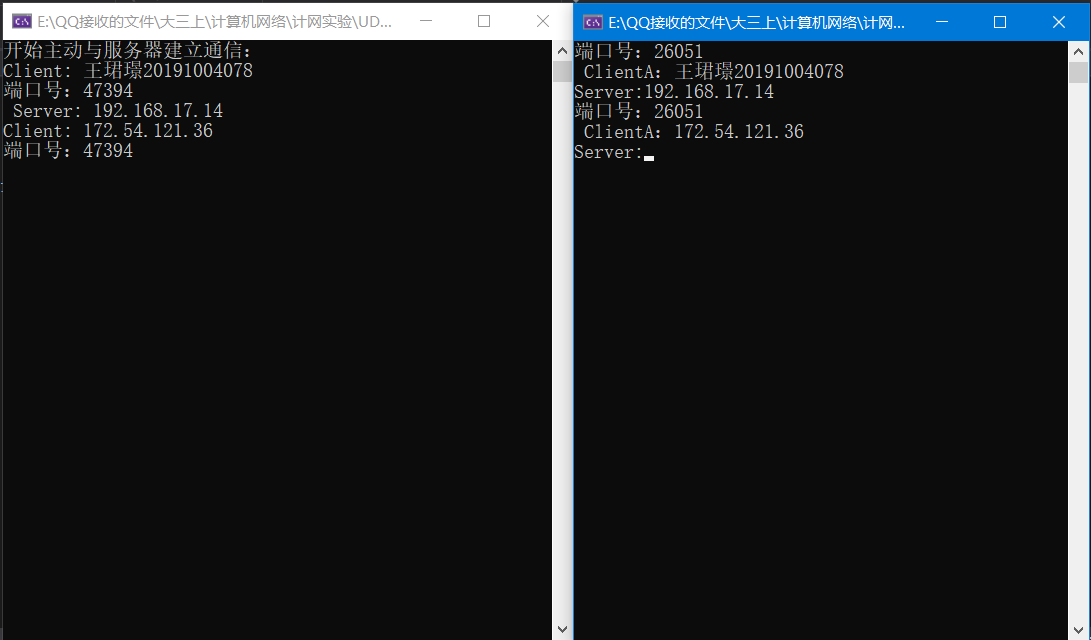
因为在请求中还没有响应的查询结果信息，所以Answer RRs，Authority RRs，Additional RRs 的值都为 0。



图 1 DNS报文结构

**实验二 Socket 通信实验**

**1、运行基于UDP的通信程序，使用程序A发送自己的学号、姓名、IP地址和端口号到程序B，并收到程序B回复的它的IP地址和端口号；请打印相关截图贴在作业本上，并简要分析。**



UDP 无连接，不提供可靠传输，没有拥塞控制，简单开销小。

Socket套接字概述：

网络上具有唯一标识的IP地址和端口号组合在一起才能构成唯一能识别的标识符套接字。

通信的两端都有Socket。

网络通信其实就是Socket间的通信。

数据在两个Socket间通过IO流传输。

Socket在应用程序中创建，通过一种绑定机制与驱动程序建立关系，告诉自己所对应的IP和port。

UDP传输

1.发送Send

创建DatagramSocket, 随机端口号

创建DatagramPacket, 指定数据, 长度, 地址, 端口

使用DatagramSocket发送DatagramPacket

关闭DatagramSocket

2.接收Receive

创建DatagramSocket, 指定端口号

创建DatagramPacket, 指定数组, 长度

使用DatagramSocket接收DatagramPacket

关闭DatagramSocket

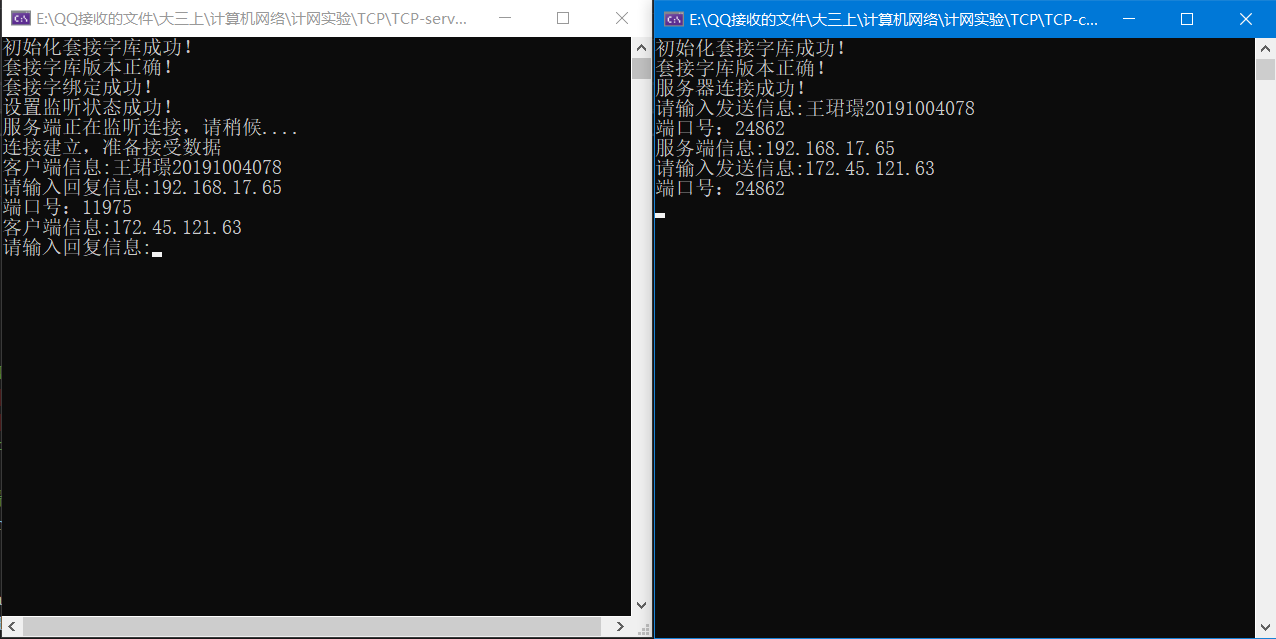
从DatagramPacket中获取数据

3.接收方获取ip和端口号

String ip = packet.getAddress().getHostAddress();

int port = packet.getPort();

**2、运行基于TCP的通信程序，使用客户端A发送自己的学号、姓名、IP地址和端口号到程序B，并收到程序B回复的它的IP地址和端口号；请打印相关截图贴在作业本上，并简要分析。**



TCP协议

1.客户端

创建Socket连接服务端(指定ip地址,端口号)通过ip地址找对应的服务器

调用Socket的getInputStream()和getOutputStream()方法获取和服务端相连的IO流

输入流可以读取服务端输出流写出的数据

输出流可以写出数据到服务端的输入流

2.服务端

创建ServerSocket(需要指定端口号)

调用ServerSocket的accept()方法接收一个客户端请求，得到一个Socket

调用Socket的getInputStream()和getOutputStream()方法获取和客户端相连的IO流

输入流可以读取客户端输出流写出的数据

输出流可以写出数据到客户端的输入流

**实践一 现代工业控制网络应用调研**

**1、工业控制网络的基本架构和组网原理**

CAN 是Controller Area Network 的缩写（以下称为CAN），是ISO国际标准化的串行通信协议。在汽车产业中，出于对安全性、舒适性、方便性、低功耗、低成本的要求，各种各样的电子控制系统被开发了出来。由于这些系统之间通信所用的数据类型及对可靠性的要求不尽相同，由多条总线构成的情况很多，线束的数量也随之增加。为适应“减少线束的数量”、“通过多个LAN，进行大量数据的高速通信”的需要，1986 年德国电气商博世公司开发出面向汽车的CAN 通信协议。此后，CAN 通过ISO11898 及ISO11519 进行了标准化，在欧洲已是汽车网络的标准协议。CAN 的高性能和可靠性已被认同，并被广泛地应用于工业自动化、船舶、医疗设备、工业设备等方面。现场总线是当今自动化领域技术发展的热点之一，被誉为自动化领域的计算机局域网。它的出现为分布式控制系统实现各节点之间实时、可靠的数据通信提供了强有力的技术支持。

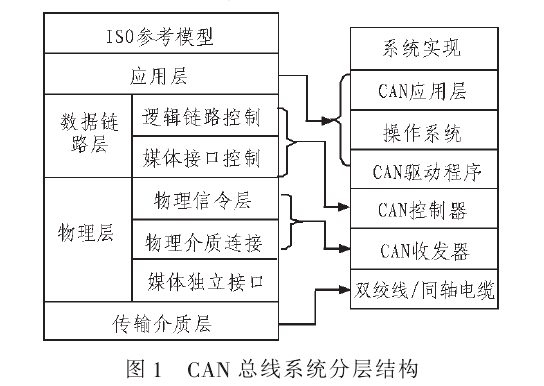


图 CAN总线系统分层结构

逻辑链路控制子层(LLC)的作用:为远程数据请求以及数据传输提供服务,确定由实际要使用的LLC子层接收哪一个报文,为恢复管理和过载通知提供手段。

MAC子层的作用主要是传送规则，也就是控制帧结构、执行仲裁、错误检测、出错标定、故障界定。位定时的一些普通功能也可以看作是MAC子层的一部分。

物理层的作用是在不同节点之间根据所有的电气属性进行位的实际传输。

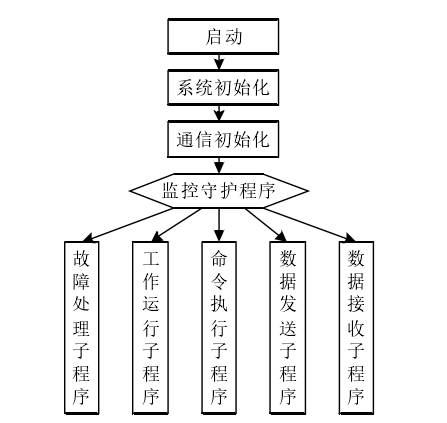


图 CAN节点软件流程图

**2、总结工业控制网络的难点是什么，当前有哪些解决措施或者方法。**

得益于新技术的迅猛发展，目前工业控制网络拥有良好发展前景，但同时也面临如下的问题和挑战：

①行业壁垒需要打破

某些行业采用行业私有标准的信息采集终端和应用管理平台，行业封闭性强，与外部网络的互连互通性差；某些行业由于信息保密等原因，不愿开放内部资源，也不愿采用第三方信息系统，无法纳入到工业互联网框架。工业互联网需要各行业遵守统一的标准规范，在不涉及行业机密和信息安全的范围内实现有效的互连互通。

② 产业链有待完善

工业互联网的一些重要的环节尚未发展起来，比如工业互联网的网络设计、工业互联网统一信息平台、工业大数据服务等方面的能力还有急需建设。

③安全问题亟待关注

从工业互联网本身来看，工业互联网的发展意味着工业控制系统将更加复杂化、IT化和通用化，不同工业控制系统互连互通，内部将越来越多采用通用软件、通用硬件和通用协议，这将增加信息安全隐患。

④我国急需攻破核心技术

从产业角度看，虽然我国工业控制系统已经形成较大的产业规模，但是芯片、嵌入式操作系统、嵌入式软件、总线协议和工控软件等核心技术仍受制于国外公司，高端市场拥有自主知识产权的技术和产品较少，这意味着如果不能自主可控，工业互联网的发展将受制于人。

**解决办法：**

完善支持鼓励政策，促进制造业数字化改造

推动工业数据标准制定与应用，促进数据的开放共享

加强数据安全保护体系建设。强化工业数据和个人信息保护

加强核心技术攻关，夯实技术基础。

**3、工业控制网络的发展趋势**

工业控制网络的发展历程是分步骤的，从传统的控制网络到较为先进的现场总线，再后来随着科技文明的进步，发展为现在研究热点工业以太网以及到无线网络控制。未来工业网络的发展需要从通信的实时性，安全性和可靠性来努力，想要达到这个层度也不是很简单的，实现多总线路集成，实时异构网络也是将来发展的一个重要方向。预计未来工业控制网络将向着更高的带宽、开放性的工业控制网络体系的方向发展。

（一）网络一体化方向发展

网络一体化是指自动化控制与系统网络有效结合在一起，在未来发展应用中有着良好的技术优势

（二）全局系统化方向发展

全局系统化是指从长效发展、全局的角度进行考虑，让工业自动化控制未来的发展目标更加明确、思路更加清晰。在计算机网络、信息技术的大力支持下，工业自动化控制正朝着全局系统化方向发展。

（三）人工智能化方向发展

基于工业自动化控制发展趋势的探讨，也需要重视其在人工智能方面的科学发展。人工智能化方向发展的具体表现为：一是现阶段随着我国科技力量的不断壮大，人工智能应用范围正在加大，工业生产领域亦不例外。二是基于人工智能的工业自动化控制，也可通过对分布型智能化系统设计与开发方面的考虑，增强自动化控制系统的应用中的性能可靠性，并在智能化模块的支持下，加快该控制系统发展速度，增加我国工业生产领域控制中的技术优势。

随着我国工业生产模式的转型升级，工业控制网络将在创新工业要素配置、生产制造和产业组织方式中带来更多的创新，从大规模生产过渡到定制化生产的模式，将加快工业生产向网络化、智能化、柔性化和服务化转变，从而推进我国制造业和信息业的融合、制造业和服务业的融合。

**4、典型应用**

钢铁工业的生产过程是一个综合性的复杂过程，以往的技术并不能较好的对其生产进行全方位的控制管理，但是自动化 网络控制技术却可以实现这一目的。钢铁企业在运用自动化网 络控制技术的时候，能够从整体的角度上对钢铁工业的生产展 开全面的监控管理，不仅如此还可以将环境参数进行一定的优 化处理，科学探测出生产过程中出现的故障问题，然后再根据 实际的故障问题采取有针对性的修复工作。所以自动化网络控制技术不仅可以提升钢铁工业的生产效率，同时还可以全面提升钢铁工业企业的经济效益以及信息管理水平。

（一）自动化网络控制技术的基础自动化系统应用分析

根据钢铁基础自动化系统分析，目前为众多企业所常用的技术是 PLC 和 DCS 控制技术。但是这两种技术却有着不同的应用目的，对于 PLC 控制技术来说，PLC 控制技术主要是应用在生产装备上的，对生产装备进行控制。而 DCS 控制技术针 对的却是整个钢铁工业的生产流程。这两种技术中，PLC 控制技术在钢铁基础自动化系统中承担的是一种核心作用，它通过对具体的生产设备自动化的控制，以此来实现自动化控制系统中的众多功能。而 DCS 控制技术的地位就并不是前者那么重要，它主要起到的作用是一种辅助的效用，它能够将整个生产流程进行一定的优化，从而钢铁工业企业的经济效益也就得到了提升。

（二）自动化网络控制技术应用于生产管理系统的分析

钢铁工业的生产一定要以经济效益作为重要指标，因为这样才能使得钢铁企业在纵横两个方向上的集成得以实现。正是因为如此，也就对钢铁企业的生产控制提出了一定的要求，这个要求是钢铁企业在生产过程中，应当及时的对生产的数据进行交换和管理，这样钢铁企业的管理层才能拥有切实的数据依据。自动化网络控制技术应用到钢铁生产中是具有一定的意义的，它能够将整个生产过程加以协调处理，从而质量检测力度才能得到有效的提升，也就全面提升了钢铁企业生产过程的流畅度以及高效化。

**二、心得与建议**

经过本次实验学习wireshark软件的使用并亲手使用WireShark软件进行抓包，我对IP数据报、TCP报文和UDP数据报的结构、什么是套接字、怎样用套接字通信都有了更进一步的理解，进一步体会了几种主要协议是如何工作的，对课堂所学的理论知识起到了很好的巩固和补充。本次实验拓展了眼界，锻炼了自己的动手能力和学习能力，收获了更多网络技术方面的知识。

同时我也感受到了Wireshark这个软件的便捷与实用，它会自动整理数据包中的主要信息如源地址、目的地址、端口号、确认号、协议名称等，中间的Packet Details面板分层次地显示了一个数据包中的内容，可以点击展开来查看各个数据包中的所有内容，并且还会把各种首部中的不同功能区的值按顺序分别列出来，而且在Packet List面板，它还会根据一定的颜色规则对不同性质的数据包进行着色，以区分正常和有问题的，非常清晰、直观。

纸上得来终觉浅，绝知此事要躬行。课堂理论总是难免抽象和理解不深，实践，也只有实践才能弥补这一缺陷，很多东西都是这样，自己动手做一遍，不懂的问题就懂了，已经懂的问题则往往会有不一样的理解。遇事要提前准备，开始实验之前就要下载好软件并自己看视频学习它的入门使用，这样才不会在实验课到来时慌慌张张。

**附录**

**1、实验二程序**

**UDP发送端：**

//udp-client

#include<iostream>

#include<WinSock2.h>

#include <string>

#pragma comment(lib,"ws2\_32.lib")

#pragma warning(disable : 4996)

using namespace std;

int main() {

//套接字信息结构

WSADATA wsadata;

//设置版本号

WORD sockVersion = MAKEWORD(2, 2);

//建立一个客户端套接字；

SOCKET sClient;

//启动构建，将“为ws2\_32.lib”加载到内存中，做一些初始化工作

if (WSAStartup(sockVersion, &wsadata) != 0) {

//判断是否构建成功，若失败，则客户端打印一句提示话。

printf("WSAStartup failed \n");

return 0;

}

//创建客户端udp套接字

sClient = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (SOCKET\_ERROR == sClient) {

printf("socket failed !\n");

return 0;

}

//创建服务器端地址

sockaddr\_in serverAddr;

//创建服务器端地址

sockaddr\_in clientAddr;

//设置服务器端地址，端口号，协议族

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(8889);

serverAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

//获取服务器地址和客户端地址构造体的长度

int slen = sizeof(serverAddr);

int clen = sizeof(clientAddr);

//设置接受数据缓冲区大小

char buffer[1024];

memset(buffer, 0, sizeof(buffer));

//用于记录发送函数和接受函数的返回值

int iSend = 0;

int iRcv = 0;

//ring str;

cout << "开始主动与服务器建立通信：" << endl;

while (true) {

//从控制台获取数据

cout << "Client: ";

//getline(cin, str) ;

cin >> buffer;

//发送信息给客户端

iSend = sendto(sClient, buffer, strlen(buffer), 0, (SOCKADDR\*)& serverAddr, slen);

cout << "端口号："<<serverAddr.sin\_port << endl;

if (iSend == SOCKET\_ERROR) {

cout << "sendto failed " << endl;

closesocket(sClient);

WSACleanup();

cout << "5s后关闭控制台。" << endl;

Sleep(5000);

return 0;

}

//若数据为byebye，断开连接

if (!(strcmp(buffer, "byebye"))) {

cout << "close connection " << endl;

closesocket(sClient);

WSACleanup();

cout << "5s后关闭控制它。" << endl;

Sleep(5000);

return 0;

}

memset(buffer, 0, sizeof(buffer));

//接受服务器端数据

iRcv = recvfrom(sClient, buffer, sizeof(buffer), 0, (SOCKADDR\*)& clientAddr, &clen);

if (iRcv == SOCKET\_ERROR) {

cout << "recvFrom failed " << endl;

closesocket(sClient);

WSACleanup();

cout << "5s后关闭控制台。" << endl;

Sleep(5000);

return 0;

}

//判断服务器是否关闭

if (iRcv <= 0) {

cout << "server disconnected··· " << endl;

closesocket(sClient);

WSACleanup();

cout << "5s后关闭控制台。" << endl;

Sleep(5000);

return 0;

}

else {

cout << " Server: " << buffer << endl;

}

memset(buffer, 0, sizeof(buffer));

}

closesocket(sClient);

WSACleanup();

return 0;

}

**UDP接收端：**

#include <iostream>

#include <WinSock2.h>

#include<WS2tcpip.h>

#include<string>

#pragma comment(lib,"ws2\_32.lib")

using namespace std;

int main() {

//设置版本号

WORD sockVersion = MAKEWORD(2, 2);

//定义一个WSADATA类型的结构体，存储被WSAStartup函数调用后返回的Windows Sockets数据

WSADATA wsadata;

//初始化套接字，启动构建，将“ws2\_32.lib”加载到内存中

if (WSAStartup(sockVersion, &wsadata)) {

printf("WSAStartup failed \n");

return 0;

}

//创建一个套接字，即创建一个内核对象

SOCKET hServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (hServer == INVALID\_SOCKET) {

printf("socket failed \n");

return 0;

}

//创建服务器端地址并绑定端口号的IP地址

sockaddr\_in addrServer;

addrServer.sin\_family = AF\_INET;

addrServer.sin\_port = htons(8889);

addrServer.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

// 初始化内核对象，传参给内核对象，此时数据可能都处于未就绪链表

int nRet = bind(hServer, (sockaddr\*)& addrServer, sizeof(addrServer));

if (nRet == SOCKET\_ERROR) {

printf("socket bind failed\n");

closesocket(hServer);

WSACleanup();

return 0;

}

//创建一个客服端地址

sockaddr\_in addrClient;

int nlen = sizeof(addrClient);

//创建一个中间变量，用于存放用户输入的信息

// ring str;

//用于接受数据的缓冲区。

char buffer[1024];

//初始化缓冲区

memset(buffer, 0, sizeof(buffer));

int irecv;

int isend;

//可以循环接受数据

while (true) {

//接收数据：

//接受客户端的消息

irecv = recvfrom(hServer, buffer, sizeof(buffer), 0, (SOCKADDR\*)& addrClient, &nlen);

cout << "端口号："<< addrClient.sin\_port << endl;

//缓冲区有数据，开始读取数据

if (irecv > 0) {

//判断数据是否为结束标志，若是则关闭服务器

if (!(strcmp(buffer, "byebye"))) {

//关闭服务器套接字

cout << "ClientA: " << buffer << endl;

cout << "close connection··· " << endl;

closesocket(hServer);

WSACleanup();

cout << "5s后关闭控制它。" << endl;

Sleep(5000);

return 0;

}

else {

cout << " ClientA：" << buffer << endl;

}

}

//接受数据失败

else {

cout << "recvFrom failed " << endl;

closesocket(hServer);

WSACleanup();

cout << "5s后关闭控制台。" << endl;

Sleep(5000);

return 0;

}

//初始化缓冲区，用于下一次数据的接收

memset(buffer, 0, sizeof(buffer));

//发送数据：

cout << "Server:";

//从键盘获取数据

cin >> buffer;

//getline(cin, str);

//建立发送数据缓冲区

//const int len = sizeof(str);

// ar senddata[len];

// rcpy\_s(senddata, str.c\_str());

//发送数据

isend = sendto(hServer, buffer, strlen(buffer), 0, (SOCKADDR\*)& addrClient, nlen);

if (isend == SOCKET\_ERROR) {

cout << "sendto failed " << endl;

closesocket(hServer);

WSACleanup();

cout << "5s后关闭控制台。" << endl;

Sleep(5000);

return 0;

}

// r = "";

//初始化缓冲区，用于下一次数据的接收

memset(buffer, 0, sizeof(buffer));

}

//关闭服务器套接字

closesocket(hServer);

WSACleanup();

return 0;

}

**TCP发送端：**

//tcp-clinet

#include<iostream>

#include<winsock.h>

#pragma comment(lib,"ws2\_32.lib")

using namespace std;

void initialization();

int main() {

//定义长度变量

int send\_len = 0;

int recv\_len = 0;

//定义发送缓冲区和接受缓冲区

char send\_buf[100];

char recv\_buf[100];

//定义服务端套接字，接受请求套接字

SOCKET s\_server;

//服务端地址客户端地址

SOCKADDR\_IN server\_addr;

initialization();

//填充服务端信息

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

server\_addr.sin\_port = htons(7777);

//创建套接字

s\_server = socket(AF\_INET, SOCK\_STREAM, 0);

if (connect(s\_server, (SOCKADDR\*)& server\_addr, sizeof(SOCKADDR)) == SOCKET\_ERROR) {

cout << "服务器连接失败！" << endl;

WSACleanup();

}

else {

cout << "服务器连接成功！" << endl;

}

//发送,接收数据

while (1) {

cout << "请输入发送信息:";

cin >> send\_buf;

send\_len = send(s\_server, send\_buf, 100, 0);

if (send\_len < 0) {

cout << "发送失败！" << endl;

break;

}

cout << "端口号："<<server\_addr.sin\_port << endl;//发送端口号

recv\_len = recv(s\_server, recv\_buf, 100, 0);

if (recv\_len < 0) {

cout << "接受失败！" << endl;

break;

}

else {

cout << "服务端信息:" << recv\_buf <<endl;

}

}

//关闭套接字

closesocket(s\_server);

//释放DLL资源

WSACleanup();

return 0;

}

void initialization() {

//初始化套接字库

WORD w\_req = MAKEWORD(2, 2);//版本号

WSADATA wsadata;

int err;

err = WSAStartup(w\_req, &wsadata);

if (err != 0) {

cout << "初始化套接字库失败！" << endl;

}

else {

cout << "初始化套接字库成功！" << endl;

}

//检测版本号

if (LOBYTE(wsadata.wVersion) != 2 || HIBYTE(wsadata.wHighVersion) != 2) {

cout << "套接字库版本号不符！" << endl;

WSACleanup();

}

else {

cout << "套接字库版本正确！" << endl;

}

//填充服务端地址信息

}

**TCP接收端：**

//tcp-serve

#include<iostream>

#include<winsock.h>

#pragma comment(lib,"ws2\_32.lib")

using namespace std;

void initialization();

int main() {

//定义长度变量

int send\_len = 0;

int recv\_len = 0;

int len = 0;

//定义发送缓冲区和接受缓冲区

char send\_buf[100];

char recv\_buf[100];

//定义服务端套接字，接受请求套接字

SOCKET s\_server;

SOCKET s\_accept;

//服务端地址客户端地址

SOCKADDR\_IN server\_addr;

SOCKADDR\_IN accept\_addr;

initialization();

//填充服务端信息

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);

server\_addr.sin\_port = htons(7777);

//创建套接字

s\_server = socket(AF\_INET, SOCK\_STREAM, 0);

if (bind(s\_server, (SOCKADDR\*)& server\_addr, sizeof(SOCKADDR)) == SOCKET\_ERROR) {

cout << "套接字绑定失败！" << endl;

WSACleanup();

}

else {

cout << "套接字绑定成功！" << endl;

}

//设置套接字为监听状态

if (listen(s\_server, SOMAXCONN) < 0) {

cout << "设置监听状态失败！" << endl;

WSACleanup();

}

else {

cout << "设置监听状态成功！" << endl;

}

cout << "服务端正在监听连接，请稍候...." << endl;

//接受连接请求

len = sizeof(SOCKADDR);

s\_accept = accept(s\_server, (SOCKADDR\*)& accept\_addr, &len);

if (s\_accept == SOCKET\_ERROR) {

cout << "连接失败！" << endl;

WSACleanup();

return 0;

}

cout << "连接建立，准备接受数据" << endl;

//接收数据

while (1) {

recv\_len = recv(s\_accept, recv\_buf, 100, 0);

if (recv\_len < 0) {

cout << "接受失败！" << endl;

break;

}

else {

cout << "客户端信息:" << recv\_buf << endl;

}

cout << "请输入回复信息:";

cin >> send\_buf;

send\_len = send(s\_accept, send\_buf, 100, 0);

if (send\_len < 0) {

cout << "发送失败！" << endl;

break;

}

cout << "端口号：" << accept\_addr.sin\_port << endl;

}

//关闭套接字

closesocket(s\_server);

closesocket(s\_accept);

//释放DLL资源

WSACleanup();

return 0;

}

void initialization() {

//初始化套接字库

WORD w\_req = MAKEWORD(2, 2);//版本号

WSADATA wsadata;

int err;

err = WSAStartup(w\_req, &wsadata);

if (err != 0) {

cout << "初始化套接字库失败！" << endl;

}

else {

cout << "初始化套接字库成功！" << endl;

}

//检测版本号

if (LOBYTE(wsadata.wVersion) != 2 || HIBYTE(wsadata.wHighVersion) != 2) {

cout << "套接字库版本号不符！" << endl;

WSACleanup();

}

else {

cout << "套接字库版本正确！" << endl;

}

//填充服务端地址信息

}

**2、参考文献**

[1]韩成浩,高晓红.CAN总线技术及其应用[J].制造业自动化,2010,32(02):146-149.

[2]徐雷.关于工业控制网络的发展现状及趋势的研究[J].科技创新与应用,2013(04):56.

[3]李文杰.网络安全未来的发展趋势[J].电子技术与软件工程,2020(06):238-239.

[4]何戡,曲超,宗学军,郑洪宇,纪胜龙.机器学习在工业网络入侵检测中的研究应用[J].小型微型计算机系统,2021,42(02):437-442.