****

**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：**

**课 程：**

**指导老师：**

**学 号：**

**班 级：**

**姓 名：**

**2021年12月5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。

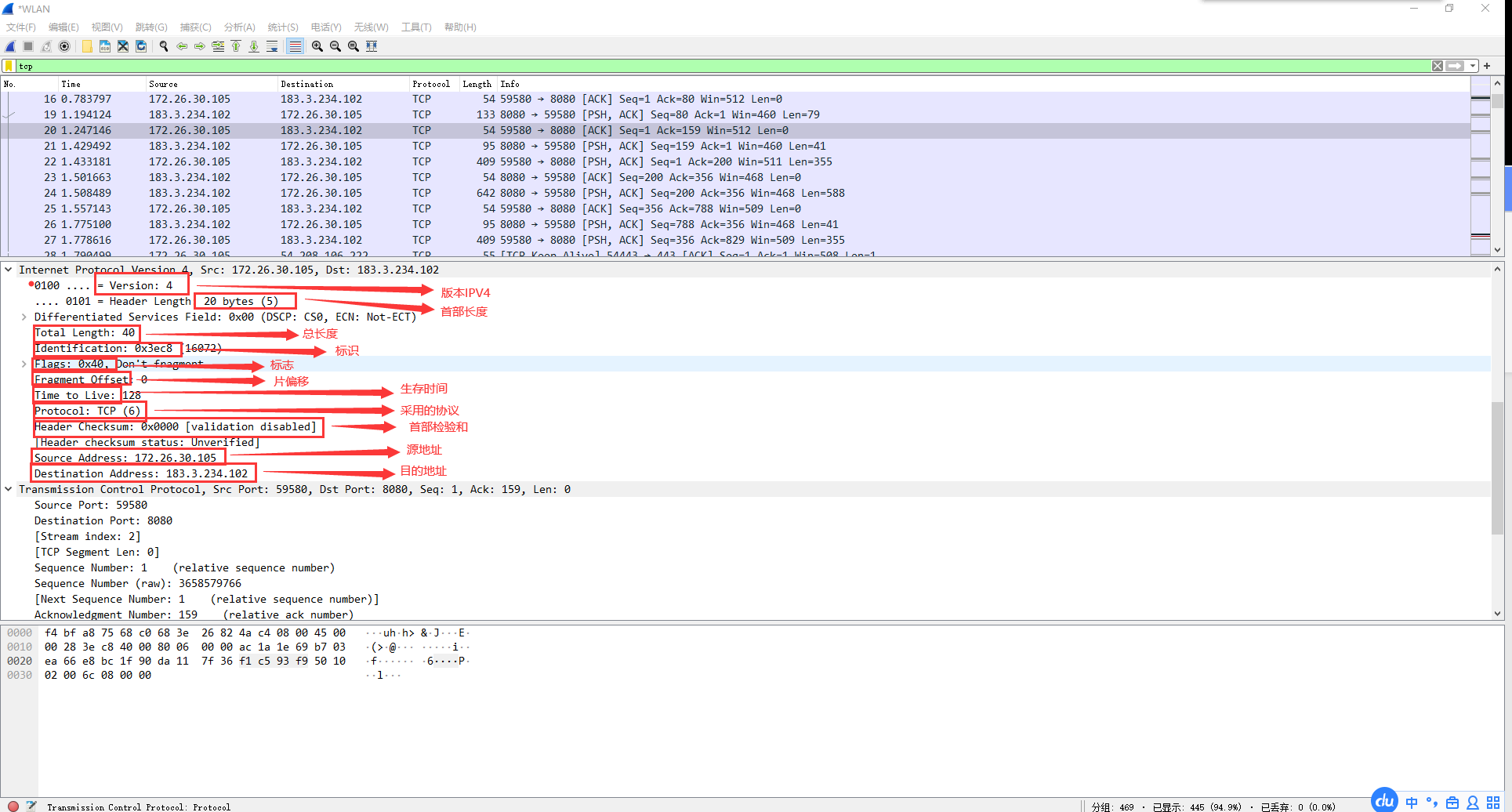


图 1 wireshark抓包结果及各项含义



图 2 IP数据报结构图

以上为抓取到的一个IP数据报，通过wireshark软件，我们可以得到与该IP数据报相关的信息（上图中红色标注）。版本号为IPV4、首部长度20bytes、总长度40bytes、标识为0x3ec8、标志为0x40、片偏移为0、生存时间128s、使用的协议为TCP、首部检验和0x0000（此处的首部检验和是由网卡完成的）、源地址172.26.30.105、目的地址183.3.234.102。

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：

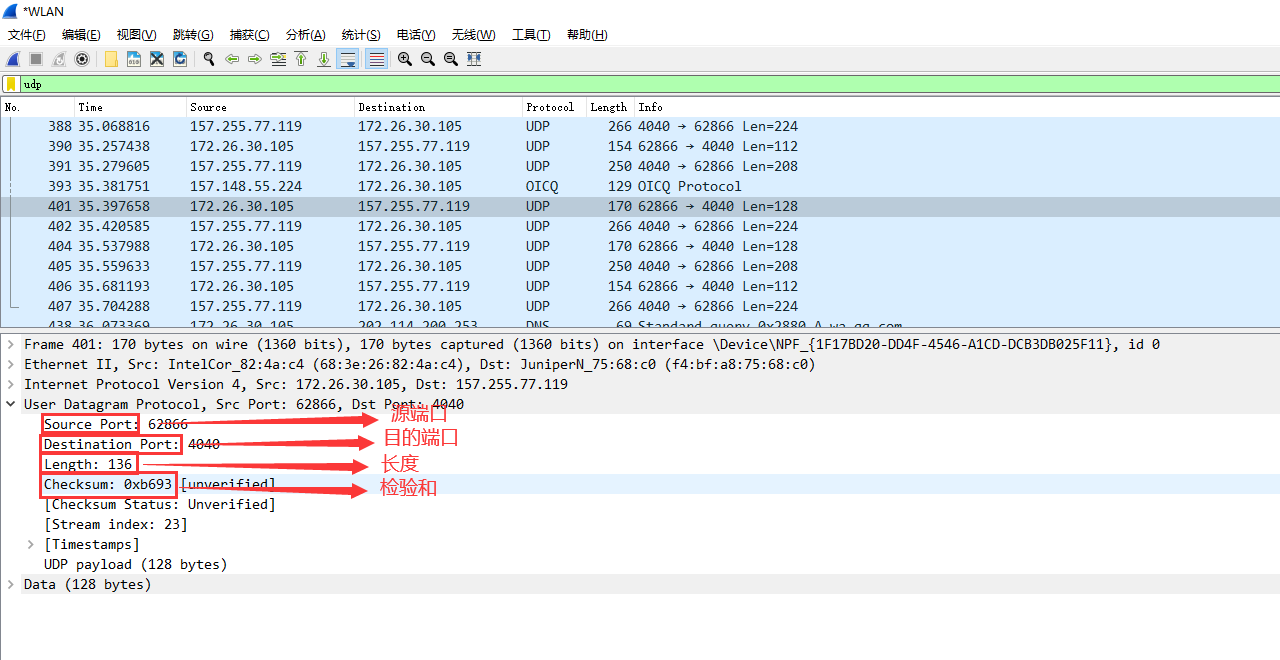


图 3wireshark抓包结果及各项含义

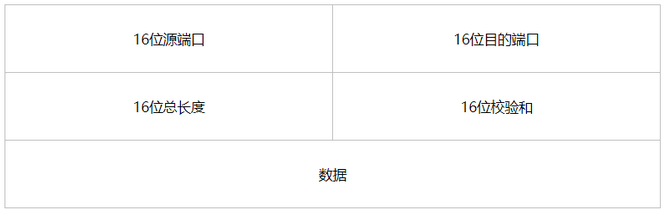


图 4UDP首部结构

由UDP首部结构图和抓包所得到的数据可知，该UDP数据报源端口为62866、目的端口为4040、首部长度为136bytes、检验和为0xb693。

TCP：

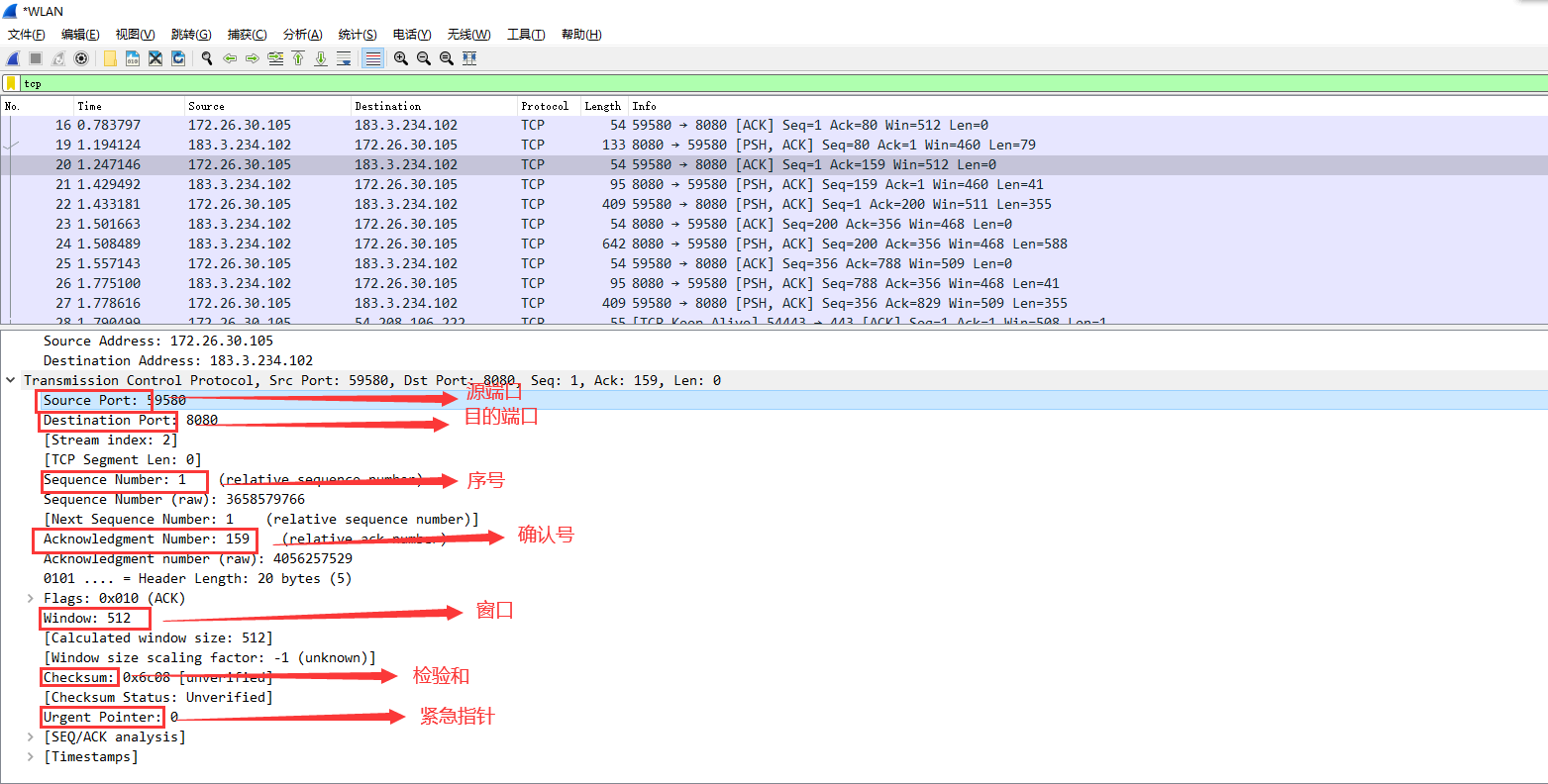


图 5wireshark抓包结果及各项含义

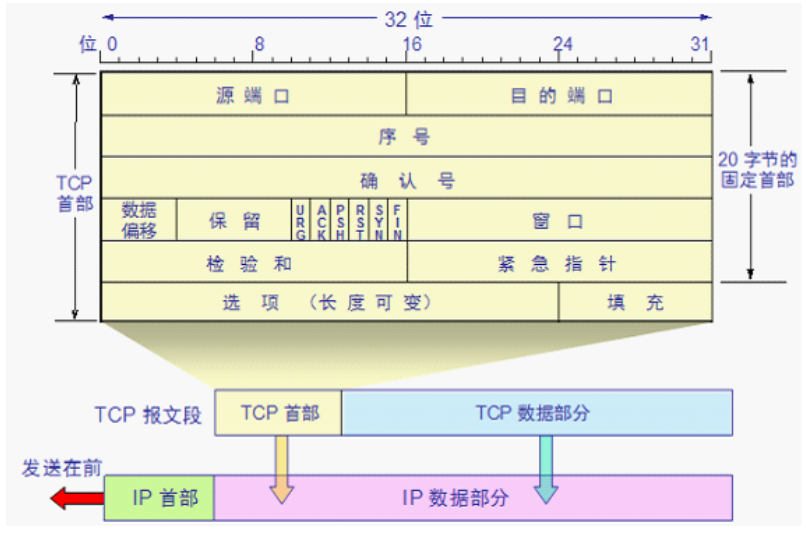


图 6TCP报文首部结构图

通过结构图对改TCP报文进行分析可知：源端口为59580、目的端口为8080、序号为1、确认号为159、窗口为大小512、检验和0x6c08、紧急指针为0。

1. 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。

**文本通信：**

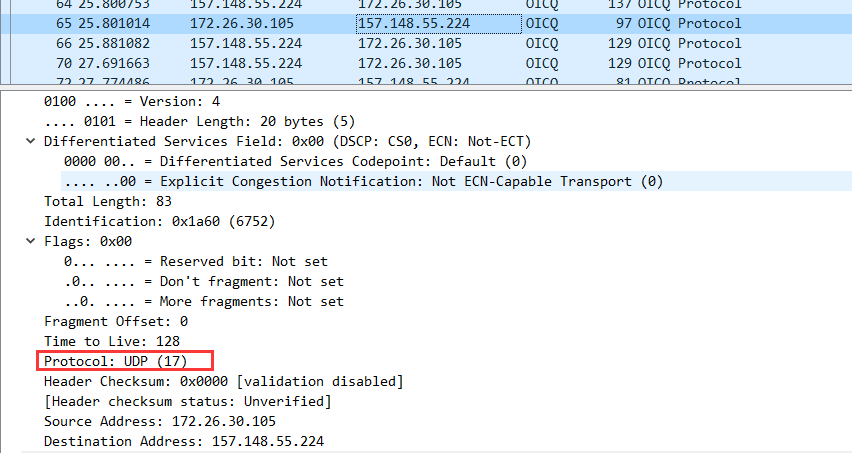


图 7wireshark抓包结果及各项含义

由图可见，文本通信使用的是UDP协议。

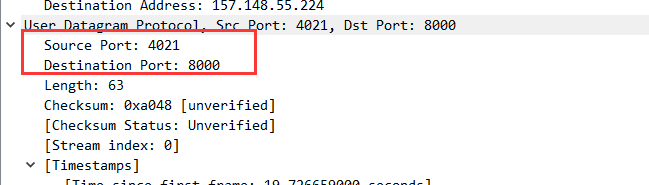


图 8wireshark抓包结果及各项含义

文本通信所使用的源端口为4021、目的端口为8000。

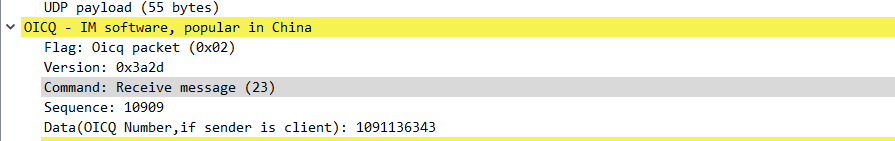


图 9wireshark抓包结果及各项含义

该栏中，Command代表报文的属性，此处为Receive message即为接收信息。

**视频通信：**

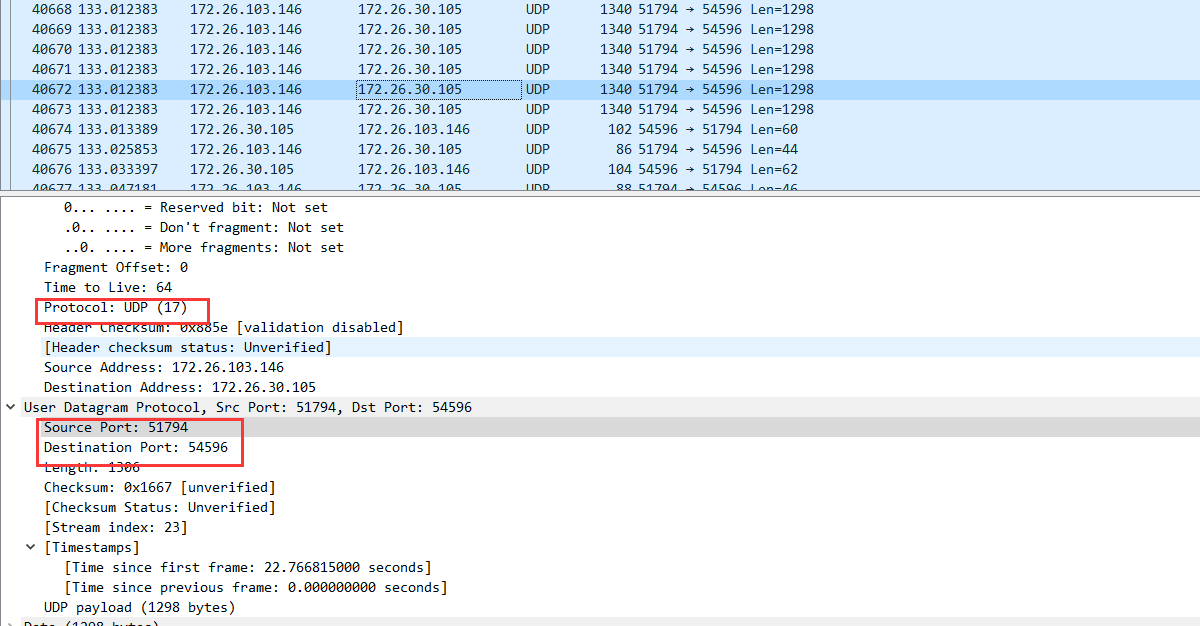


图 10 wireshark抓包结果

由图可见，视频通信使用的源端口为51794、目的端口为54596，使用的协议为UDP。

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。

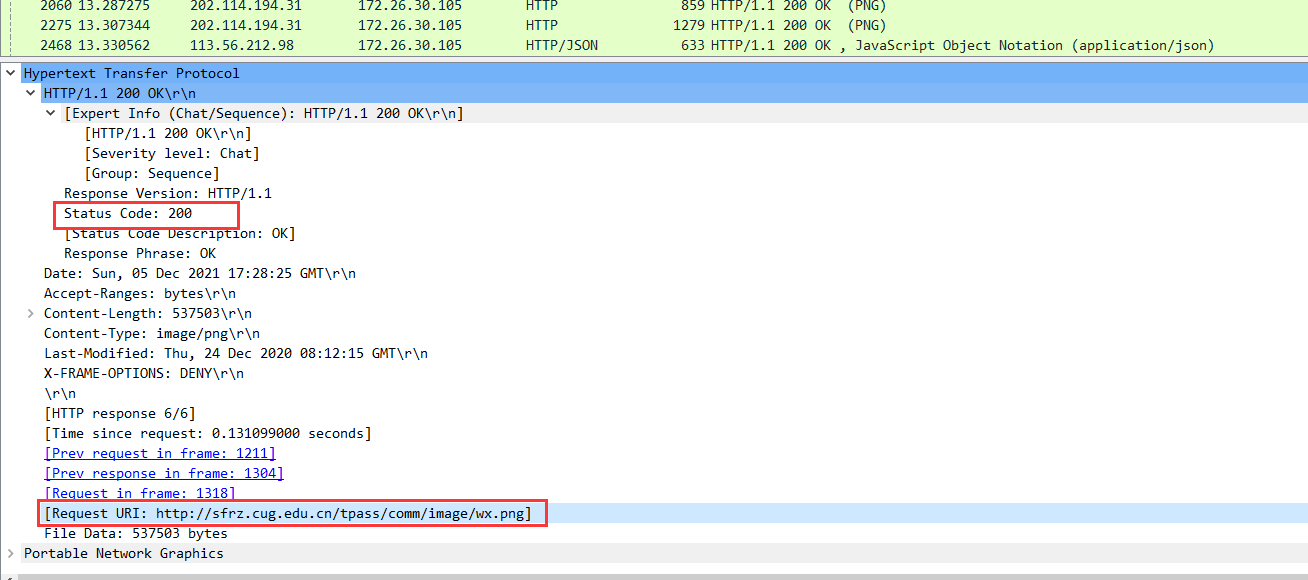


图 11 wireshark抓包结果

在此对学校官网进行抓包。使用的是HTTP协议，由上图可知，状态码为200，表明请求成功。

**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。

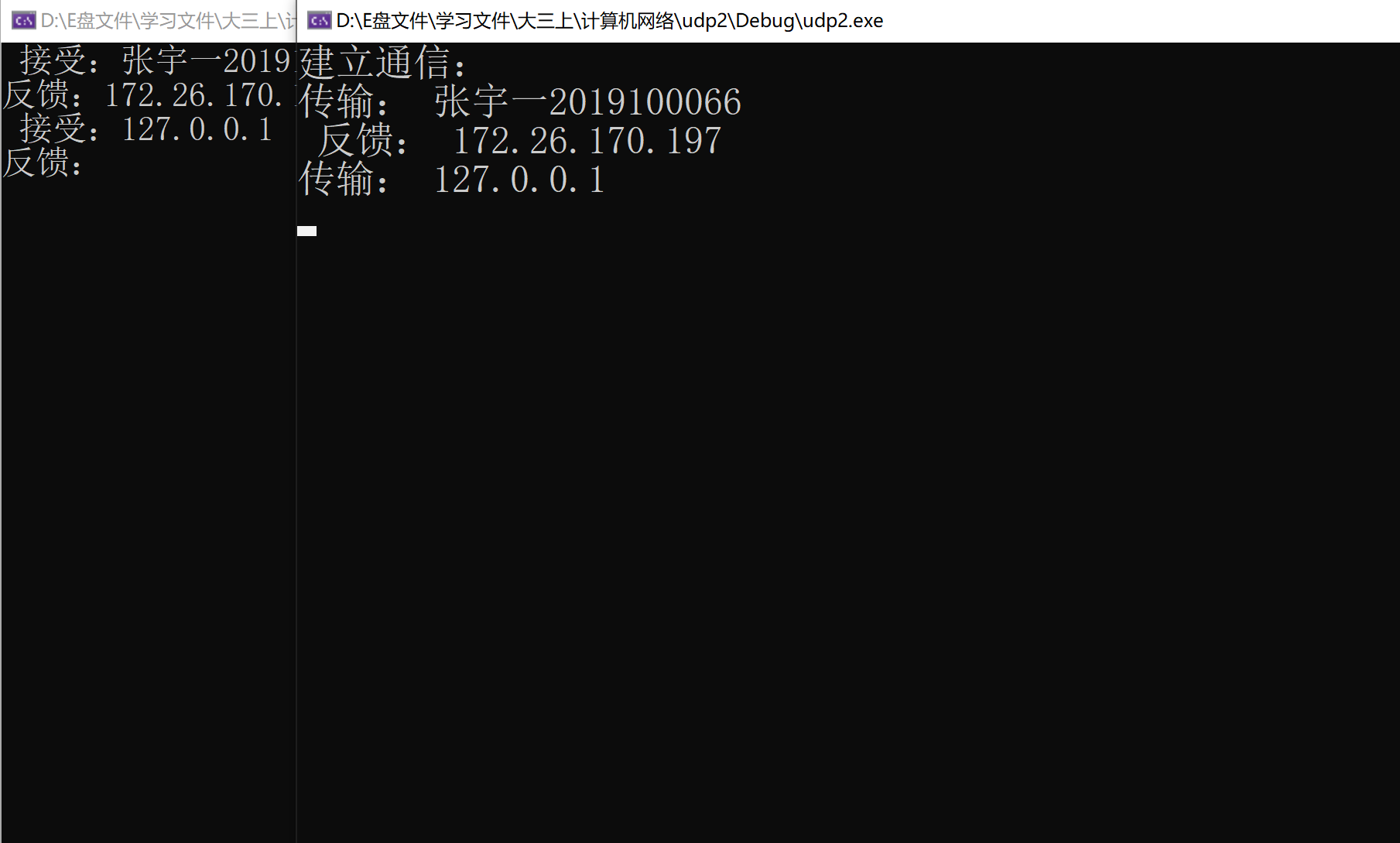


图 12UDP通信成功截图

UDP是面向无连接的，尽最大努力交付，先运行发送端或先运行接收端的程序均可。通过本次实验可知，UDP支持一对一的相互通信。

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。

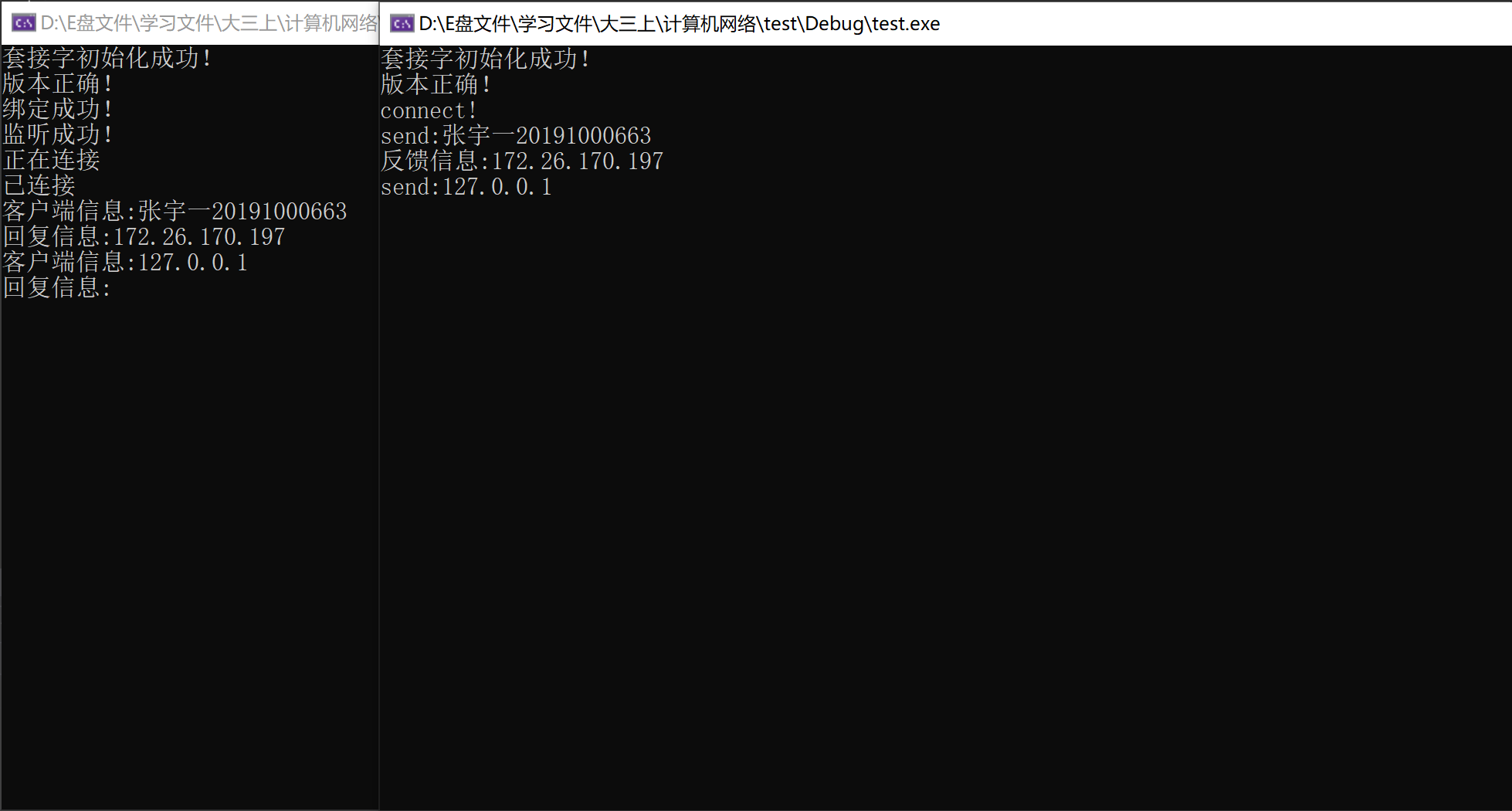


图 13TCP通信截图

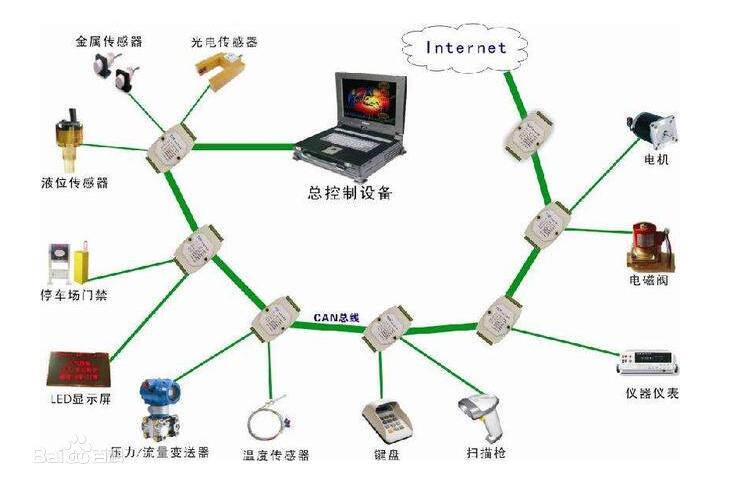
TCP是面向连接的，需要先运行发送方的程序，再运行接收方的程序，当两者成功连接后，便可以进行双向通信。TCP提供的是可靠传输，并且有拥塞控制。

**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

选择任一现场总线或者工业以太网（可以是 CAN 总线也可以是其他总线或者网络），描述其网络的分层结构，进行网络控制或管理所使用的相关软件及其主要功能。

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](https://baike.baidu.com/item/%E4%B8%B2%E8%A1%8C%E9%80%9A%E4%BF%A1%E5%8D%8F%E8%AE%AE" \t "https://baike.baidu.com/item/CAN%E6%80%BB%E7%BA%BF/_blank)。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](https://baike.baidu.com/item/%E6%8E%A7%E5%88%B6%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/CAN%E6%80%BB%E7%BA%BF/_blank)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](https://baike.baidu.com/item/%E5%8D%9A%E4%B8%96" \t "https://baike.baidu.com/item/CAN%E6%80%BB%E7%BA%BF/_blank)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遵从OSI模型，按照OSI基准模型，CAN结构划分为两层：数据链路层和物理层。按照IEEE 802.2和802.3 标准，数据链路层又划分为：逻辑链路控制（LLC-Logic Link Control）、媒体访问控制（MAC-Medium Access Control）。物理层又划分为：物理信令（PLS-Physical Signalling）、物理媒体附属装置（PMA-Physical Medium Attachment）、媒体相关接口（MDI-Medium Dependent Interface）。

MAC子层运行借助称之为“故障界定实体（FCE）”的管理实体进行监控。故障界定是使判别短暂干扰和永久性故障成为可能的一种自检机制。物理层可借助检测和管理物理媒体故障实体进行监控（例如总线短路或中断，总线故障管理）。

LLC和MAC两个同等的协议实体通过交换帧或协议数据单元（PDU-Protocol Data Unit）和（N）-用户数据组成，为传送一个NPDU,（N-1）层实体必须通过（N-1）服务访问点（SAP-Service Access Point）[（N-1）-SAP].NPDU借助于（N-1）层服务数据单元（SDU-Service Data U nit）[（N-1）-SDU]传至（N-1）层，其服务功能允许NPDU的传送。

SDU是接口数据，对其识别预先在（N）层实体间进行，亦即，它表示逻辑数据单元由服务进行传送。CAN协议的数据链层既不提供分配一个SDU至多个PDU,也不提供分配多个SDU至一个PDU的方法，亦即，NPDU直接由相应的NSDU和层指定控制信息N-PCI构成。

CAN具有十分优越的特点，使人们乐于选择。这些优越的特点包括：  
1、多主控制  
 当总线空闲时，连接到总线上的所有单元都可以启动发送信息，这就是所谓的多主控制的概念。  
 先占有总线的设备获得在总线上进行发送信息的资格。这就是所谓的CSMA/CR（Carrier Sense MultipleAccess/Collosion Avoidance）方法。如果多个设备同时开始发送信息，那么发送最高优先级ID消息的设备获得发送资格。  
2、信息的发送  
 在CAN协议中，所有发送的信息要满足预先定义的格式。当总线没有被占用的时候，连接在总线上的任何设备都能起动新信息的传输，如果两个或更多个设备在同时刻启动信息的传输，通过ID来决定优先级。ID并不是指明信息发送的目的地，而是指示信息的优先级。  
如果2个或者更多的设备在同一时刻启动信息的传输，在总线上按照信息所包含的ID的每一位来竞争，赢得竞争的设备（也就是具有最高优先级的信息）能够继续发送，而失败者则立刻停止发送并进入接收操作。因为总线上同一时刻只可能有一个发送者，而其它均处于接收状态，所以，并不需要在底层协议中定义地址的概念。  
3、系统的灵活性  
 连接到总线上的单元并没有类似地址这样的标识，所以，添加或去除一个设备，无需改变软件和硬件，或其它设备的应用层软件。  
4、通信速度  
 可以设置任何通讯速度，以适应网络规模。对一个网络，所有单元必须有相同的通讯速度，如果不同，就会产生错误，并妨碍网络通讯，然而，不同网络间可以有不同的通讯速度。  
5、远程数据请求  
 可以通过发送“遥控帧”，请求其他单元发送数据。  
6、错误检测、错误通知、错误恢复功能  
 所有单元均可以检测出错误（错误检测功能）。检测到错误的单元立刻同时通知其它所有的单元（错误通知功能）。如果一个单元发送信息时检测到一个错误，它会强制终止信息传输，并通知其它所有设备发生了错误，然后它会重传直到信息正常传输出去（错误恢复功能）。  
7、错误隔离  
 在CAN总线上有两种类型的错误：暂时性的错误（总线上的数据由于受到噪声的影响而暂时出错）；持续性的错误（由于设备内部出错（如驱动器坏了、连接有问题等）而导致的）。CAN能够区别这两种类型，一方面降低常出错单元的通讯优先级以阻止对其它正常设备的影响，另一方面，如果是一种持续性的错误，将这个设备从总线上隔离开。

8、连接  
 CAN总线允许多个设备同时连接到总线上且在逻辑上没有数目上的限制。然而由于延迟和负载能力的限制，实际可连接得设备还是有限制的，可以通过降低通讯速度来增加连接的设备个数。相反，如果连接的设备少，通讯的速度可以增加。

2、工业控制网络的难点问题

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。

（分析写在这里）

1. 存在实时性差，不确定性的问题
2. 工业可靠性问题
3. 以太网不提供电源，必须有额外的供电电缆
4. 以太网不是本质安全系统
5. 安全性问题
6. 现存的控制网络与新建以太控制网络的集成问题

**解决方法：**

1. 提高通信实时性，为满足用户对实时性要求，确定用哪种总线实现报文传输。

提高操作系统和交换技术以支持实时通信。操作系统基于优先级策略对非实时和实时传输提供多队列排队方式。交换技术支持高优先级的数据包接入到高优先级的端口, 以便高优先级的数据包能够快速进入到传输队列。此外, 可改善拓扑结构以提高实时性。其他研究方向还包括怎样提高在MAC层上的数据传输的调度方法等。

1. 提高通信可靠性，为满足用户对可靠性的要求，根据物理设备可靠性计算出总线系统是否满足要求，若不满足可加冗余的方法。

工业控制网络基于不同的网络交换技术, 需进行不同类型网络站点之间的通信, 因此通信的可靠性显得尤为重要。研究方向之一在于设计虚拟自动化网络, 以构筑深层防御系统。虚拟自动化网络中包含有不同的抽象层和可靠区域, 可靠区域包括远程接入区域、局部生产操作区域以及自动设备区域等, 重点在于可靠区域的设计。

1. 提高通信安全性，对信息加密可设置密码系统，还可以设置固定的操作指令，如果有某种权限才可以对设备进行读写。

安全性意味着能预防危险, 如系统故障、电磁干扰、高温辐射以及恶意攻击等因素所带来的威胁。IEC61508 针对安全通信提出了黑通道机制并制定了安全完整性等级SIL。提高工业通信的安全性, 以满足SI L 高级别的要求, 是工业控制网络安全性发展的趋势。目前一些总线研究机构基于黑通道原理针对数据破坏、丢失、时延以及非法访问等错误采用了数据编号、密码授权以及C RC 安全校验等安全保护措施, 如In-terbus Safety, Profisafe以及EtherCA TSafety 等, 这可作为工业控制网络安全性研究的参考。

1. 多现场总线集成，不同设备间需考虑集成、兼容问题。

多总线并存且相互竞争的局面由来已久, 在未来相当长的时间内这种局面还将继续。多总线集成协同完成工业控制任务, 是未来发展的趋势。研究方向之一是通过使用代理机制, 将单一总线系统中的设备映射到基于工业以太网的工业控制网络中。

1. 无线网络应用，采用屏蔽措施抗干扰，同时解决时延问题来提高效率。

无线通信进入工业控制领域的趋势无可置疑。通过有线网络与无线网络融合、广域网与局域网集成来构建实时异构网络, 是未来发展的趋势。

3、工业控制网络的发展趋势

通过调研，总结工业控制网络今后的发展趋势。

从目前趋势来看，工业以太网进入现场控制级毋庸置疑。但至少现在看来，它还难以完全取代现场总线，作为实时控制通信的单一标准。己有的现场总线仍将继续存在，最有可能的是发展一种混合式控制系统。

此外，并非每种现场总线协议都将被以太网TCP/ IP协议所替代，如AS-i、CAN。对于I/0传感器/执行器而言，应用这两种现场总线无疑是最佳的(AS-i传输4位数据，且可带电;CAN最多传输8个字节)，还有一些专用总线，如SERCOS(用于数控，控制运动轴，为IEC61491国际标准)、Instabus(用于楼宇)，都有其专门的应用领域，均不适宜于工业以太网。另外，易燃、易爆(如化工、制药)，以及环境条件恶劣、可靠性要求很高的应用场合，也不适宜于应用工业以太网。

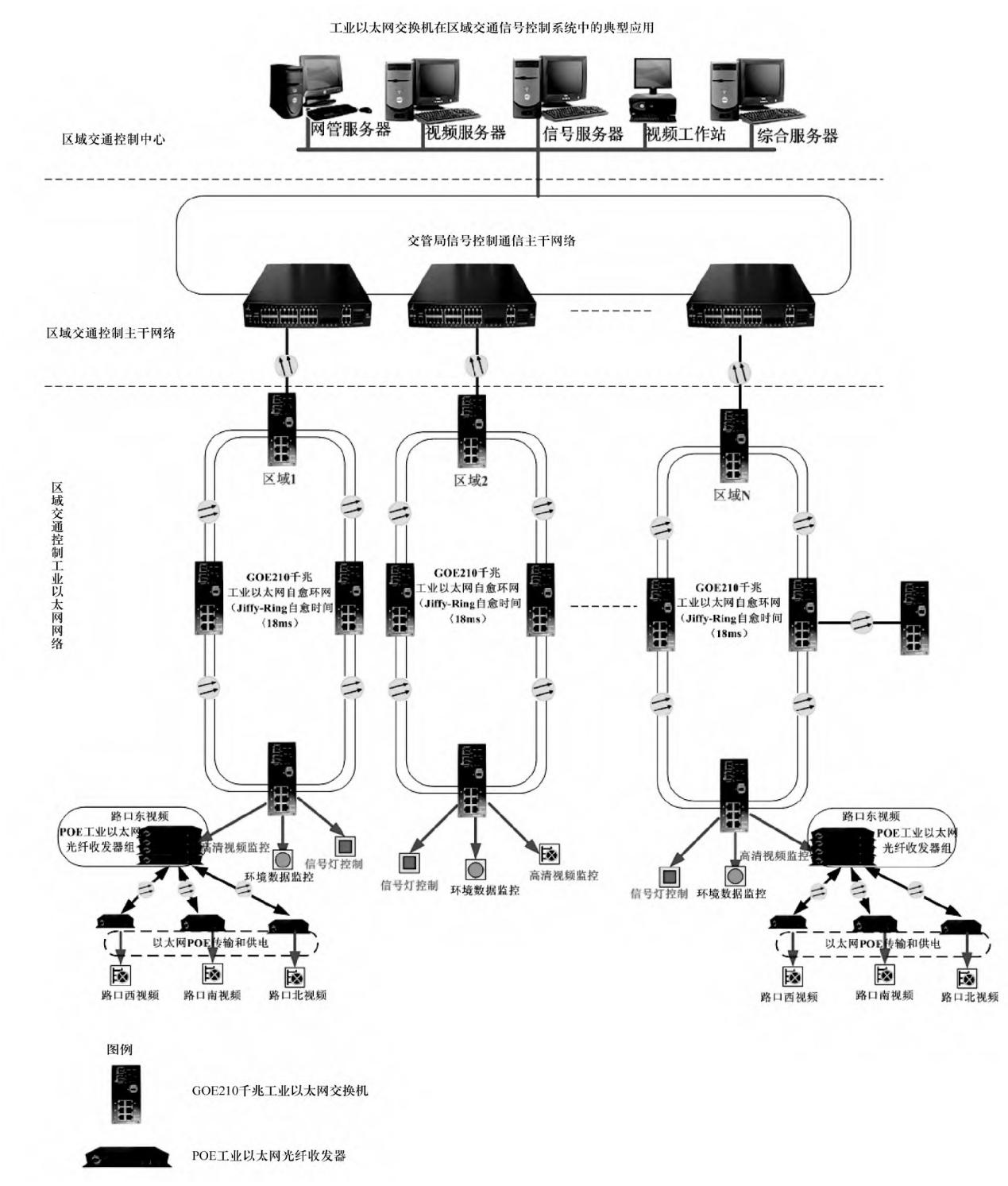
以太网进入工控领域，还有许多需要解决的问题。除了上述提到控制的实时性、协议的修订，信息传输的安全性与可靠性也是进一步的要求。此外，编址的唯一性，网络分流组件的多样化和切换技术的提高是对扩展通信网路和电缆网络的支持，也是待考虑的问题。

4、典型应用

以某一行业，例如汽车制造、钢铁生产等工业生产领域的应用为例子，说明该应用中采用的工业控制网络的结构、网络性能、特点等。

例如只能交通灯系统，它可以根据路口车辆多少，自动调节时间，可减少等候时间在75%以上，从而大大节省人们的出行时间，减少路口的无效等候，使出行更快捷。

网络结构如下图所示：



根据网络传输设备的需求及区域交通信号控制智能化，结合欧迈特公司工业以太网交换机性价比及网络可靠性的综合考虑，选用GOE210千兆工业以太网交换机构成区域冗余环网，保证系统稳定，可靠，安全传输；选用OMATE1100P工业以太网光纤收发器，实现以太网远程供电，并将路口的4个方向集中至现场工控机。

区域交通信号控制网络传输设备选用欧迈特千兆工业以太网交换机GOE210，安装于各区域路口信号机柜中并构建千兆冗余环网，启用Jiffy Ring瞬间环网冗余协议，保障了网络通信的无扰动切换，网络自愈时间小于18ms。千兆工业以太网交换机构建环网的同时，利用主节点的GOE210的上联千兆光口与各区域的三层以太网交换机通过光纤连接。采用GOE210的VLAN功能将信号控制、高清视频监控及综合环境监控等业务隔离，并将GOE210的各业务接入端口（交换机端口）采取相应带宽限制设置；同时禁用其他未接入业务的端口，以防止误接入导致交换网络的中断等安全问题。将欧迈特POE工业以太网光纤收发器OMATE1100P用于高清视频传输和以太网供电，可将路口的各方向集中传至现场工控机内作为本地存储，再将工控机中的抓拍图片、视频录像和少部分的实时视频上传监控中心。GOE210和OMATE1100P采用IP40防护等级设计，便于安装于路口机柜或机箱内，能适应如强电磁干扰、尘土、高温、低温、潮湿、振动、静电等恶劣的工作环境。

**二、附录**

**1、实验二程序**

**UDP发送端：**

#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("错误：");

return 0;

}

//创建客户端udp套接字

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

if (SOCKET\_ERROR == sClient) {

printf("连接错误：");

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 << "建立通信：" << endl;

while (true) {

//从控制台获取数据

cout << "传输： ";

//getline(cin, str) ;

cin >> buffer;

//将string型数据处理成char数组型

//const int len = sizeof(str);

// ar senddata[len];

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

//发送信息给客户端

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

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 << " 反馈： " << 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);

//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 << "接受： " << buffer << endl;

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

closesocket(hServer);

WSACleanup();

cout << "5s后关闭控制它。" << endl;

Sleep(5000);

return 0;

}

else {

cout << " 接受：" << 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 << "反馈：";

//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：**

#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 << "绑定失败！" << endl;

WSACleanup();

}

else {

cout << "绑定成功！" << endl;

}

//设置套接字为监听状态

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

cout << "监听失败！" << endl;

WSACleanup();

}

else {

cout << "监听成功！" << endl;

}

cout << "正在连接" << 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 << "已连接" << 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 << "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;

}

}

//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 << "套接字初始化失败！" << endl;

}

else {

cout << "套接字初始化成功！" << endl;

}

//检测版本号

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

cout << "版本错误！" << endl;

WSACleanup();

}

else {

cout << "版本正确！" << endl;

}

//填充服务端地址信息

}

**TCP接收端：**

#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 << "connect wroning!" << endl;

WSACleanup();

}

else {

cout << "connect!" << endl;

}

//发送,接收数据

while (1) {

cout << "send:";

cin >> send\_buf;

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

if (send\_len < 0) {

cout << "发送失败！" << endl;

break;

}

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

if (recv\_len < 0) {

cout << "接受失败！" << endl;

break;

}

else {

cout << "反馈信息:" << 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 << "套接字初始化失败！" << endl;

}

else {

cout << "套接字初始化成功！" << endl;

}

//检测版本号

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

cout << "版本错误！" << endl;

WSACleanup();

}

else {

cout << "版本正确！" << endl;

}

//填充服务端地址信息

}

**2、参考文献**

**[1]徐雷. (2013). 关于工业控制网络的发展现状及趋势的研究. 科技创新与应用(4), 1.**

**[2]胡毅, 于东, & 刘明烈. (2010). 工业控制网络的研究现状及发展趋势. 计算机科学(1), 6.**

**[3]邓波鑫.(2010).工业以太网交换机在区域交通信号控制系统中的典型应用. 自动化博览(S1),114-117. doi:CNKI:SUN:ZDBN.0.2010-S1-034.**

**[4]吴子龙,刘茜 & 轩东.(2006).以太网用于工业控制的主要难点和问题. 数字石油和化工(11),17-18. doi:CNKI:SUN:HGXI.0.2006-11-007.**

**三、心得与建议**

心得：在本次实验中，我通过wireshark软件抓包的数据，分析了IP数据报报文格式、UDP报文格式、TCP报文格式，并尝试使用Socket通信。通过本次实验，我初步将课本上的知识应用到实际中，这使我对这些知识点有了更进一步的了解。在实验中，我还尝试抓取了QQ在进行文本通信和视频通信的数据包，发现QQ在进行这两类通信时均使用的是UDP协议。除此之外，我还尝试抓取爱奇艺等视频播放软件的数据包，我发现爱奇艺在播放视频时使用的也是UDP协议，这是否说明UDP协议更为常用呢？这需要我在日后进一步的研究。本次实验涉及到的只是课本中一小部分的知识，对于拥塞控制、电子邮件收发等知识还需要我们进一步去探索。

建议：①根据现有的实验指导书，在刚上手做实验时感觉无从下手，可以进一步完善实验指导书，或者是给出一个使用wireshark的简易教程。

②可以适当增加实验内容，以加深对课本知识点点理解。