****

**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： 自动化学院**

**课 程：计算机网络与工业互联网**

**指导老师： 熊永华**

**学 号： 20201000128**

**班 级： 231202**

**姓 名： 刘瑾瑾**

**2022年 10 月 30 日**

**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;

[TCP/IP协议](https://baike.baidu.com/item/TCP/IP%E5%8D%8F%E8%AE%AE?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/IP%E6%95%B0%E6%8D%AE%E6%8A%A5/_blank)定义了一个在[因特网](https://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91/114119?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/IP%E6%95%B0%E6%8D%AE%E6%8A%A5/_blank)上传输的包，称为IP数据报，由首部和数据两部分[组成](https://baike.baidu.com/item/%E7%BB%84%E6%88%90/10881662?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/IP%E6%95%B0%E6%8D%AE%E6%8A%A5/_blank)。首部的前一部分是固定长度，共20字节，是所有IP数据报必须具有的。在首部的固定部分的后面是一些可选字段，其长度是可变的。首部中的源地址和目的地址都是IP协议地址。

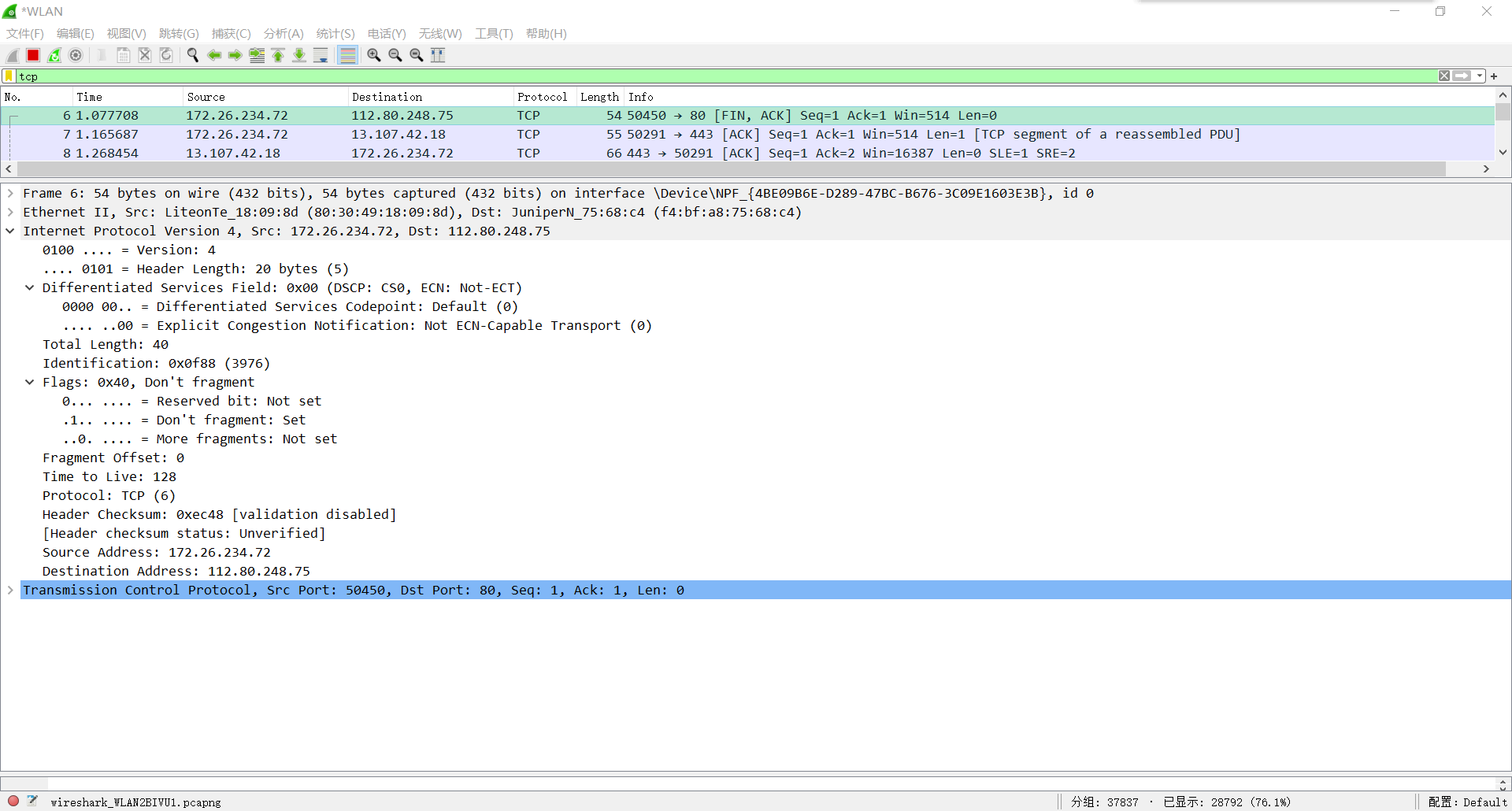
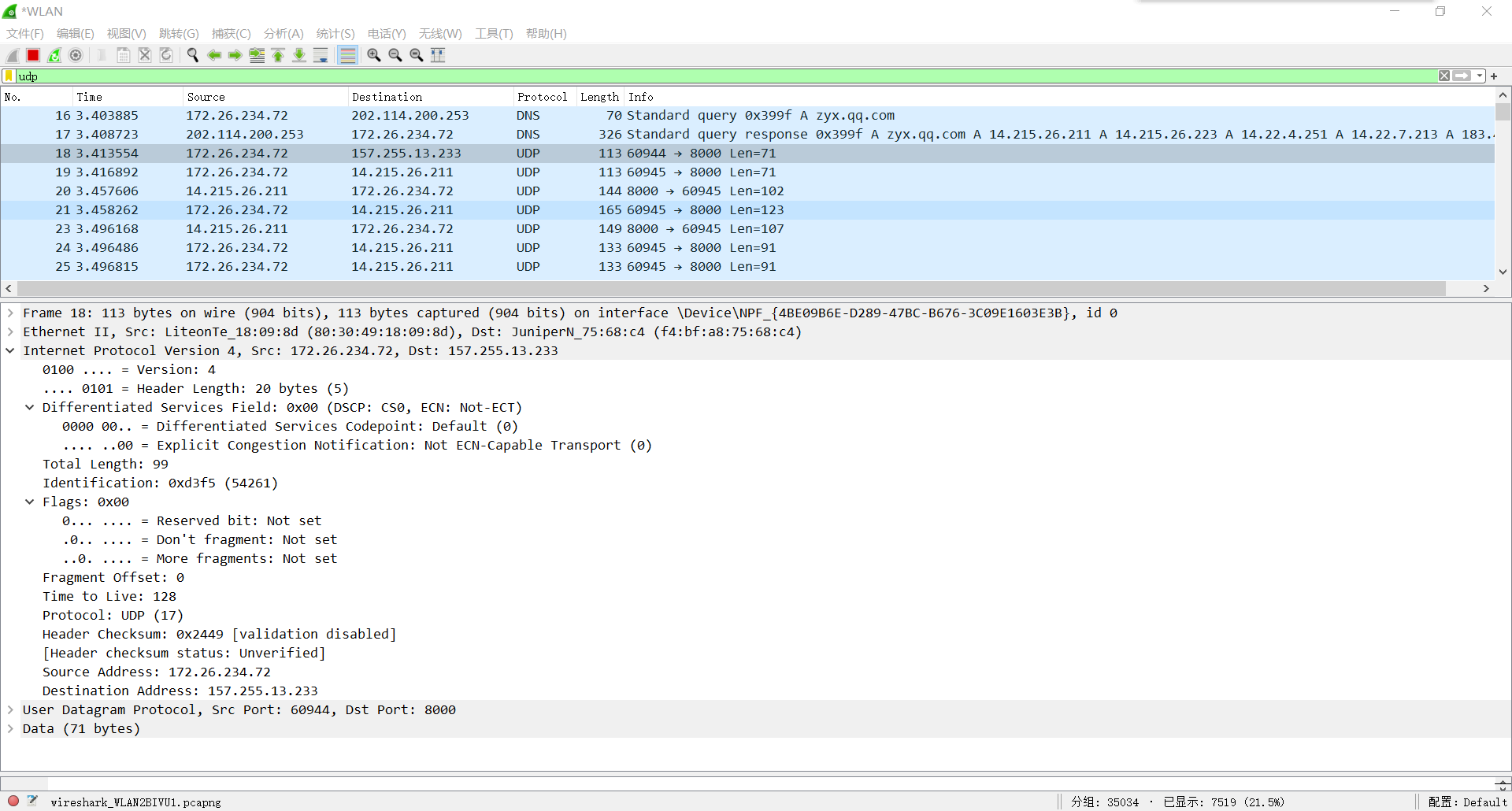
无论应用层使用TCP协议或UDP协议，或者其他的协议，运输层为其添加IP首部，成为IP数据报，其IP数据报格式都是相同的，如图1和图2所示：

图 1应用层使用TCP协议的IP数据报

图 2 应用层使用UDP协议的IP数据包

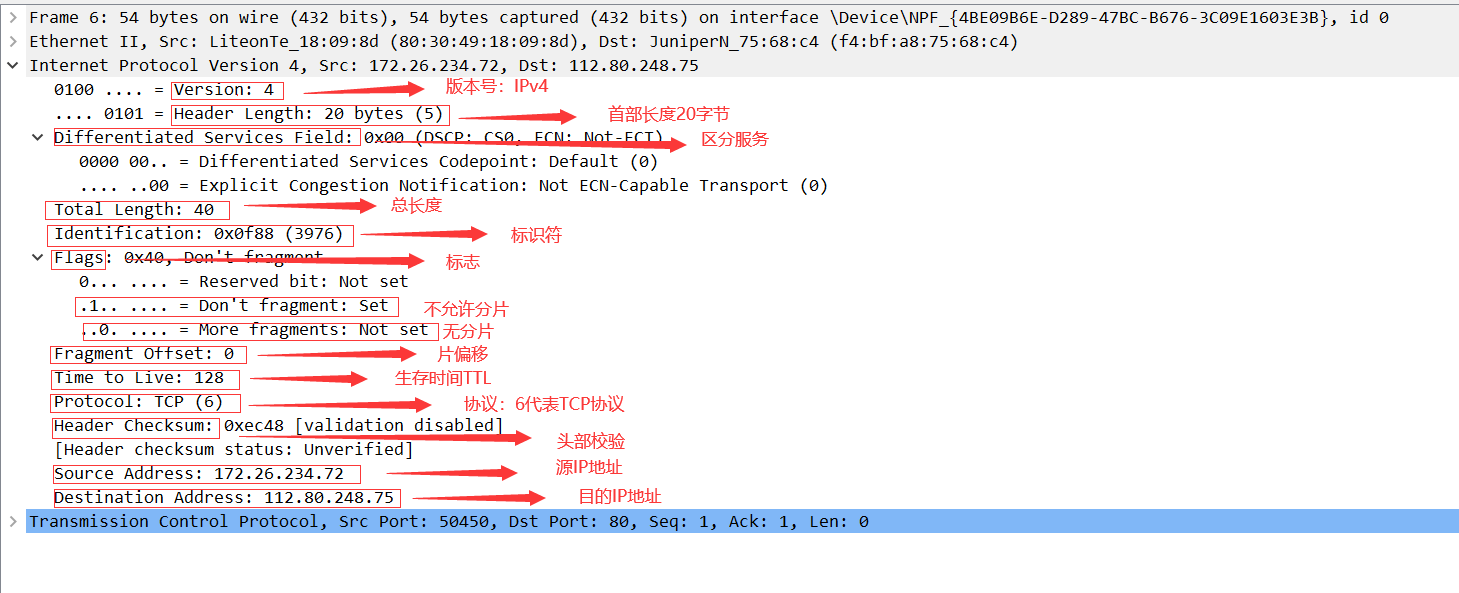


图 3 IP数据报报文结构分析

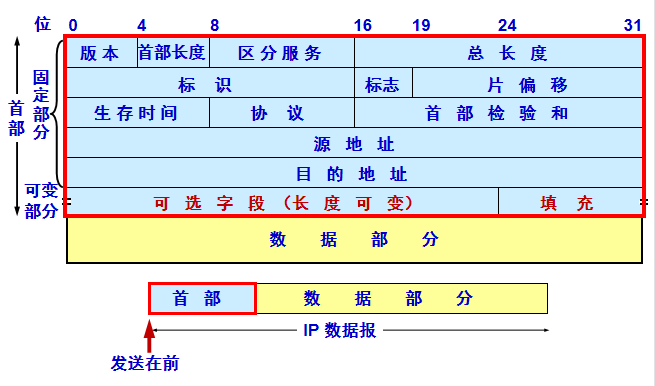


图 4 IP数据报首部格式

**版本**：占 4位，指 IP 协议的版本，目前的 IP 协议版本号为 4 (IPv4)。

**首部长度：**占 4 位，可表示的最大数值是 15 个单位(一个单位为 4 字节)，IP的首部长度的最大值是 60 字节。

**区分服务**：占 8 位，用来获得更好的服务，在一般的情况下都不使用这个字段

**总长度：**占 16 位，指首部和数据之和的长度，单位为字节，因此数据报的最大长度为 65535 字节。

**标识**：占 16 位，是一个计数器，用来产生 IP 数据报的标识。

**标志：**占 3 位，目前只有两位有意义。标志字段的最低位是 MF (More Fragment)。MF =1 表示后面“还有分片”。MF =0 表示最后一个分片。标志字段中间的一位是 DF (Don't Fragment) 。只有当 DF =0 时才允许分片。

**片偏移：**占13 位，指出：较长的分组在分片后，某片在原分组中的相对位置。片偏移以 8 个字节为偏移单位。

**生存时间：**占8 位，记为 TTL (Time To Live)，指示数据报在网络中可通过的路由器数的最大值。

**协议：**占8 位，指出此数据报携带的数据使用何种协议，以便目的主机的 IP 层将数据部分上交给那个处理过程

**首部检验和：**占16 位，只检验数据报的首部，不检验数据部分。这里不采用 CRC 检验码而采用16位二进制反码求和算法。

**源地址：**源端口的IP地址

**目的地址：**目的端口的IP地址

1. 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;
2. UDP协议数据包分析：

用户数据报 UDP 有两个字段：数据字段和首部字段。首部字段有 8 个字节，由 4 个字段组成，每个字段都是 2 个字节，包括源端口、目的端口、长度和检验和。在计算检验和时，临时把“伪首部”和 UDP 用户数据报连接在一起。伪首部仅仅是为了计算检验和。

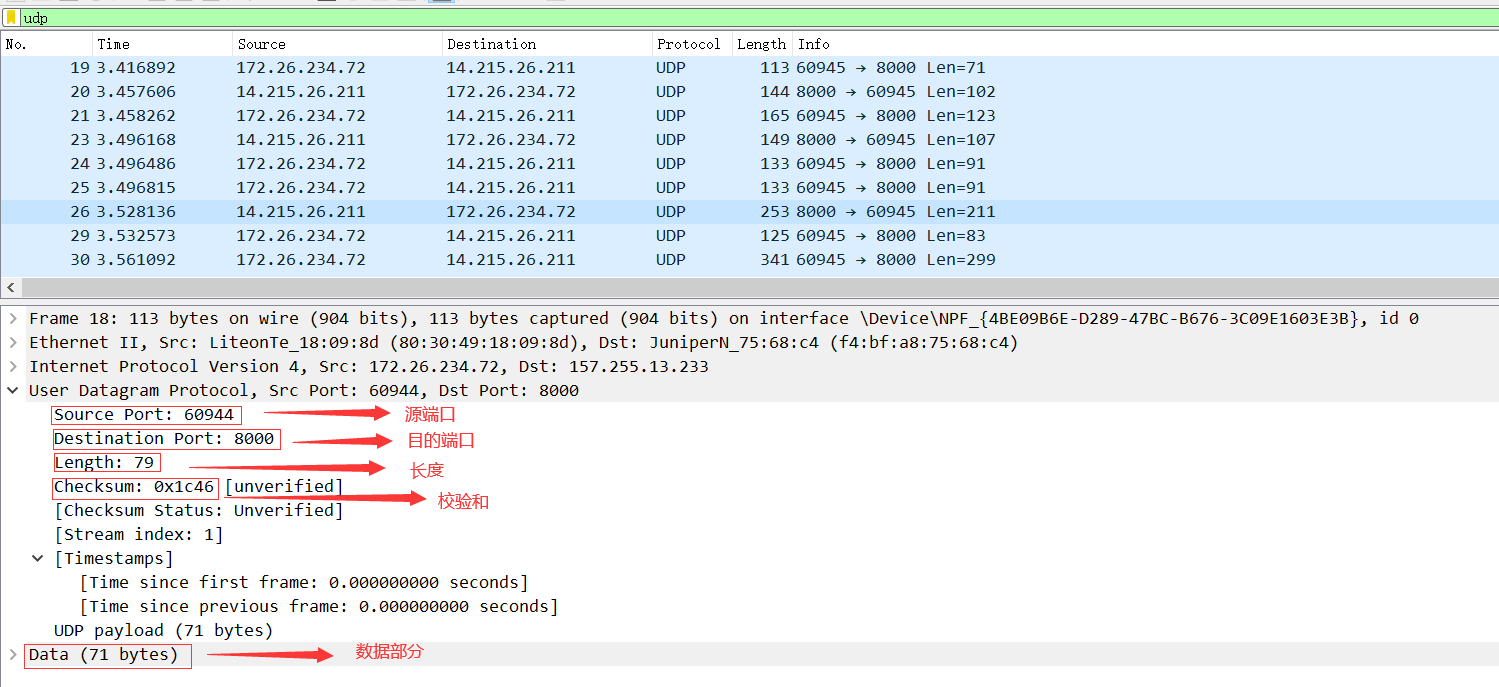


图 5 UDP数据报报文结构分析

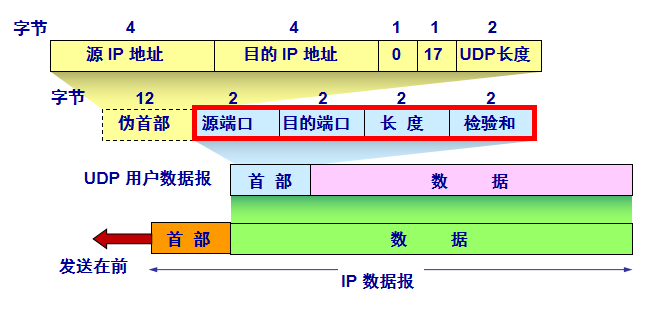


图 6 UDP数据报首部格式

**源端口：**该字段占据 UDP 报文头的前 16 位，通常包含发送数据报的应用程序所使用的 UDP 端口。接收端的应用程序利用这个字段的值作为发送响应的目的地址。这个字段是可选的，所以发送端的应用程序不一定会把自己的端口号写入该字段中。如果不写入端口号，则把这个字段设置为 0。这样，接收端的应用程序就不能发送响应了。

**目的端口：**该字段占据 16 位，表示接收端计算机上 UDP 软件使用的端口。

**长度：**该字段占据 16 位，表示 UDP 数据报长度，包含 UDP 报文头和 UDP 数据长度。因为 UDP 报文头长度是 8 个字节，所以这个值最小为8。

**检验和：**该字段占据 16 位，可以检验数据在传输过程中是否被损坏。

1. TCP协议数据包分析：

TCP 虽然是面向字节流的，但 TCP 传送的数据单元却是报文段。一个 TCP 报文段分为首部和数据两部分，而 TCP 的全部功能都体现在它首部中各字段的作用。TCP 报文段首部的前 20 个字节是固定的，后面有 4n 字节是根据需要而增加的选项 (n 是整数)。因此 TCP 首部的最小长度是 20 字节。

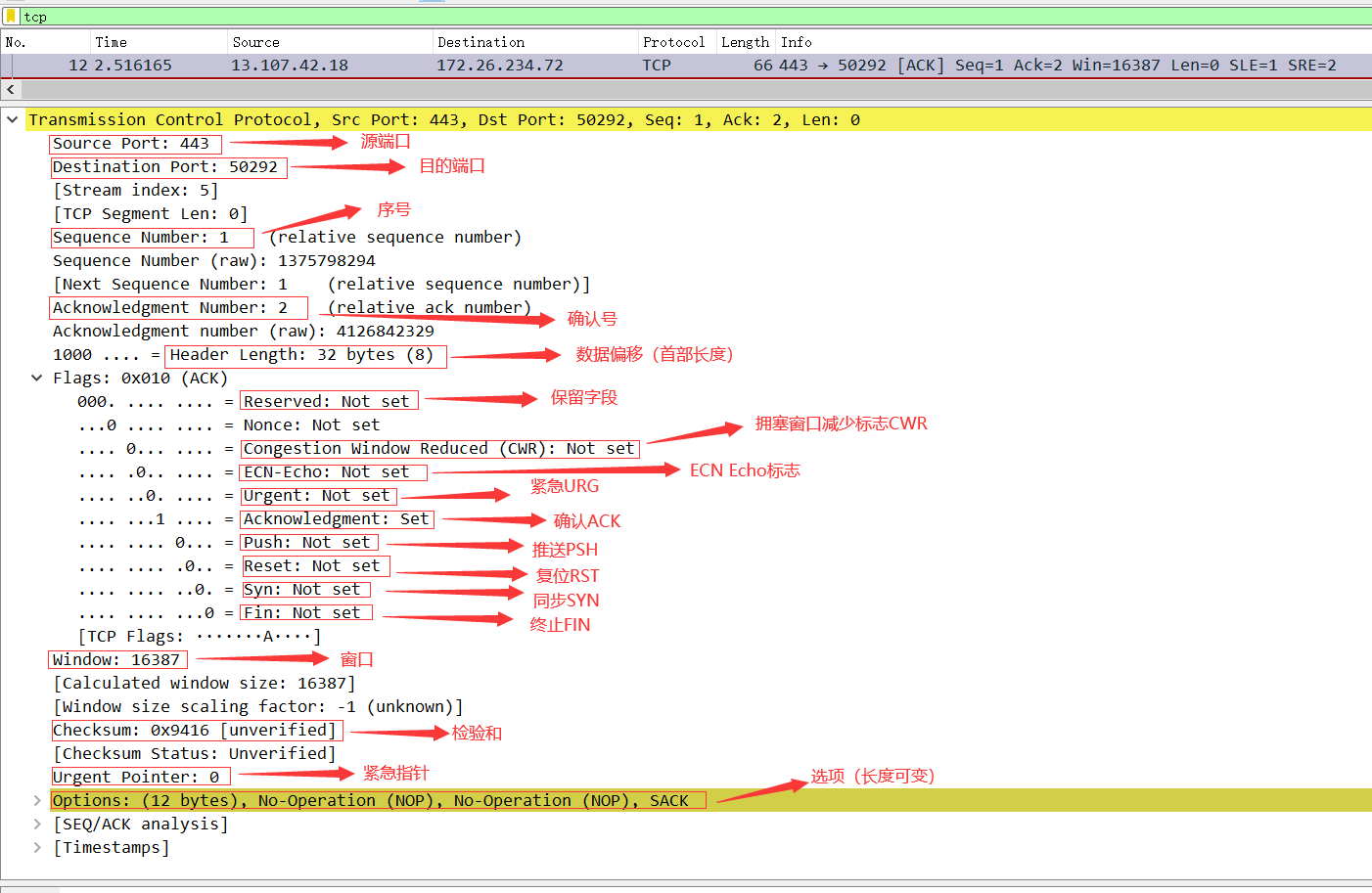


图 7 TCP数据报报文结构分析

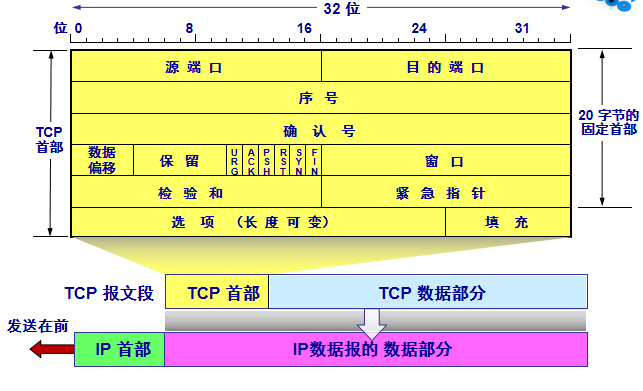


图 8 TCP数据报首部格式

**源端口**：源计算机上的应用程序的端口号，占 16 位，即2个字节。

**目的端口：**目标计算机的应用程序端口号，占 16 位，即2个字节。

**序号：**占 4 字节。TCP 连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。

**确认号：**占 4 字节，是期望收到对方的下一个报文段的数据的第一个字节的序号。

**数据偏移：**占 4 位，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。“数据偏移”的单位是 32 位字（以 4 字节为计算单位）。

**保留字段：**占 6 位，保留为今后使用，但目前应置为 0。

**标志位：**

* CWR：拥塞窗口减少标志，用来表明它接收到了设置 ECE 标志的 TCP 包。并且，发送方收到消息之后，通过减小发送窗口的大小来降低发送速率。
* ECE：用来在 TCP 三次握手时表明一个 TCP 端是具备 ECN 功能的。在数据传输过程中，它也用来表明接收到的 TCP 包的 IP 头部的 ECN 被设置为 11，即网络线路拥堵。
* URG：当 URG=1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。
* ACK：只有当 ACK =1 时确认号字段才有效。当 ACK =0 时，确认号无效。
* PSH：接收 TCP 收到 PSH = 1 的报文段，就尽快地交付接收应用进程，而不再等到整个缓存都填满了后再向上交付。
* RST： 当 RST=1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。
* SYN：同步 SYN = 1 表示这是一个连接请求或连接接受报文。
* FIN：用来释放一个连接。FIN=1 表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。

**窗口：**占 2 字节，用来让对方设置发送窗口的依据，单位为字节。

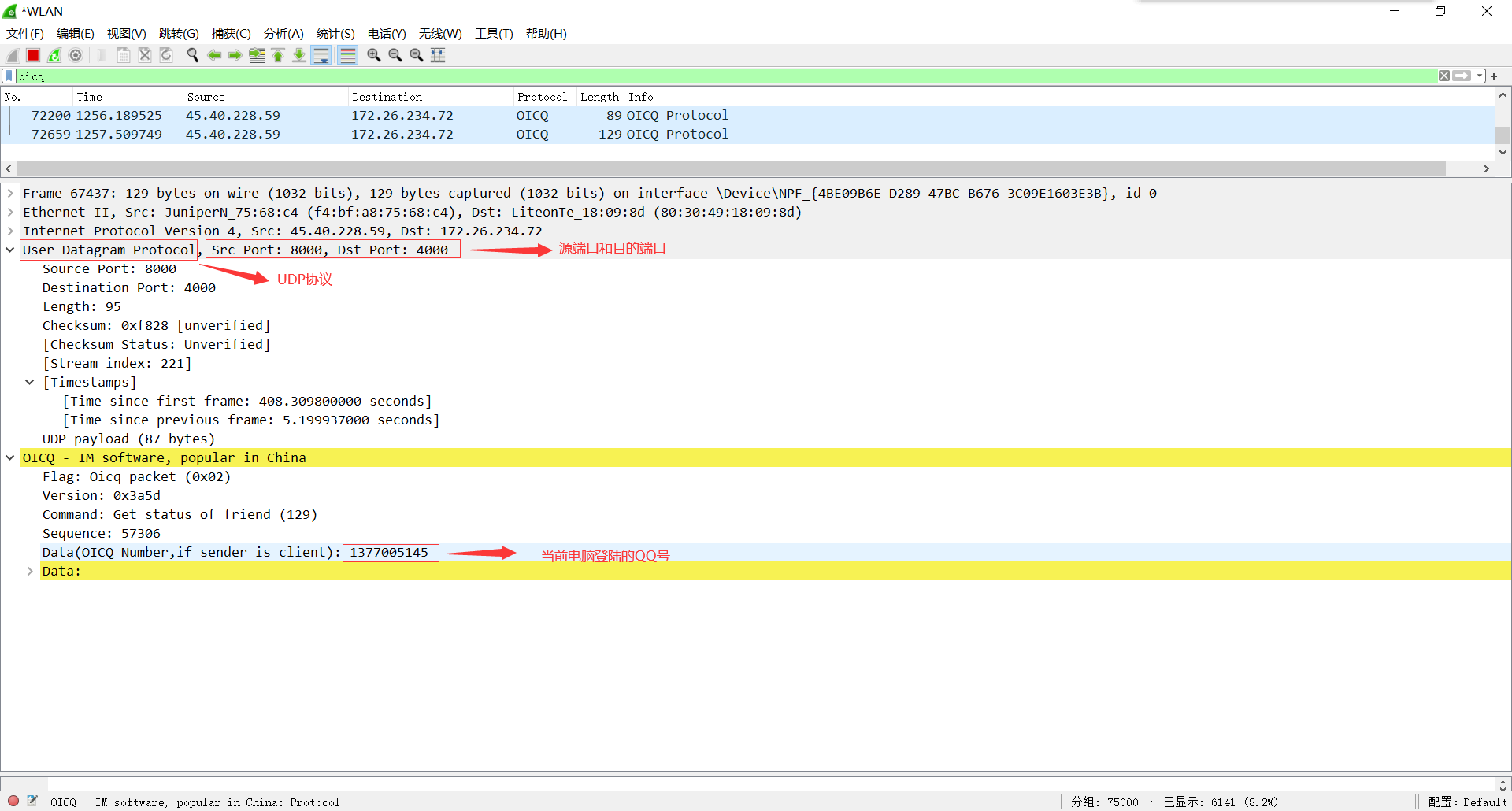
**检验和：**占 2 字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部。

**紧急指针：**占 16 位，指出在本报文段中紧急数据共有多少个字节（紧急数据放在本报文段数据的最前面）。

**选项（长度可变）：**TCP 最初只规定了一种选项，即最大报文段长度 MSS。MSS是TCP报文段中的数据字段的最大长度。数据字段加上 TCP 首部才等于整个的 TCP 报文段。所以，MSS是“TCP 报文段长度减去 TCP 首部长度”。

**填充字段：**使整个首部长度是 4 字节的整数倍。

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;
2. 文本通信：使用QQ进行文本通信时，端口号为8000和4000，使用的传输层协议为UDP协议。

图 9 使用QQ进行文本通信

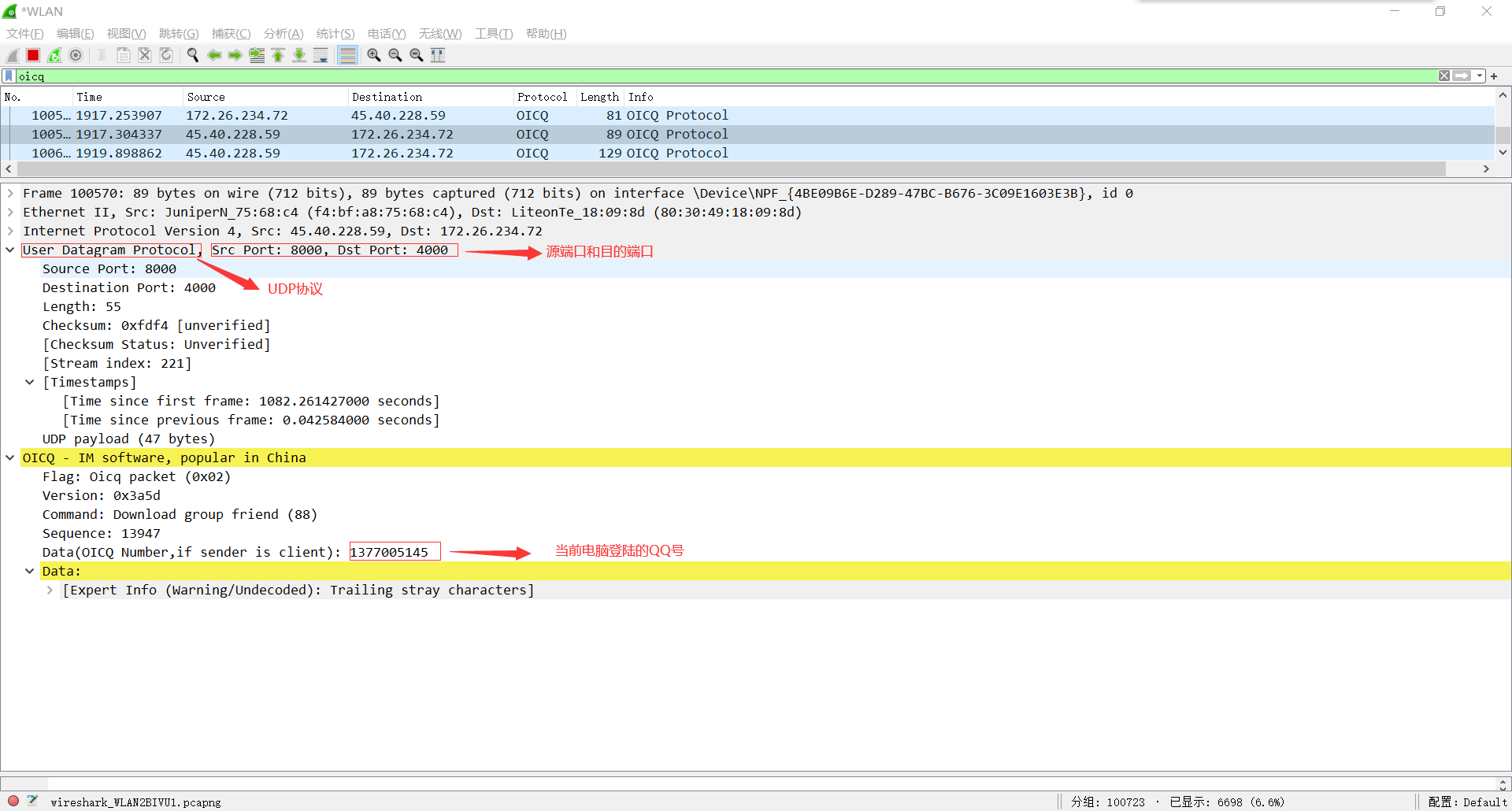
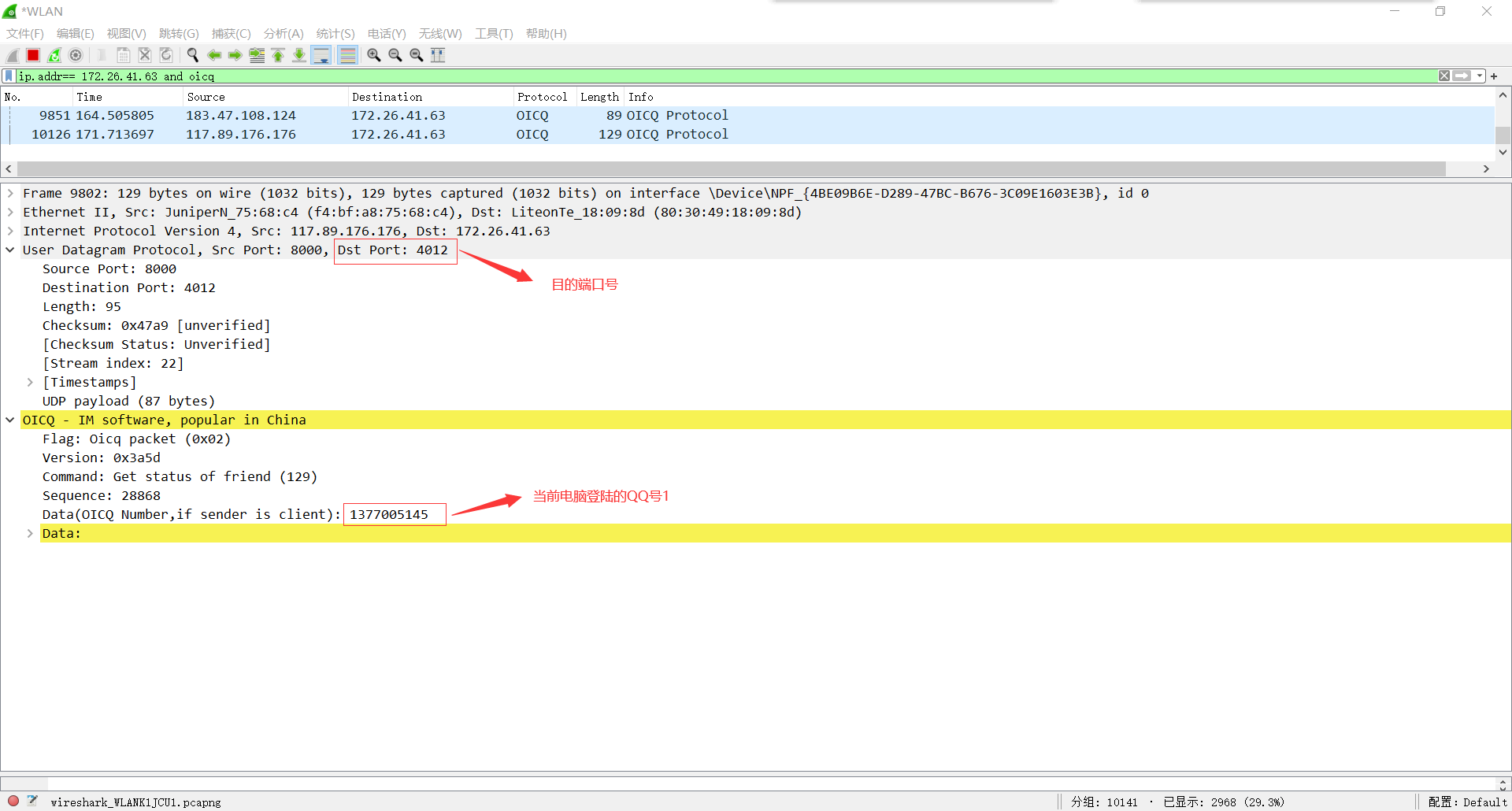
1. 视频通信：使用QQ进行视频通信时，端口号为8000和4000，使用的运输层协议为UDP协议。

图 10 使用QQ进行视频通信

1. 当电脑同时登陆两个QQ号时，每个QQ号对应的端口号不同

如图11所示，用户1377005145对应的端口号是4012。

图 11 QQ号1通信

如图12所示，用户1664073240对应的端口号是49507。

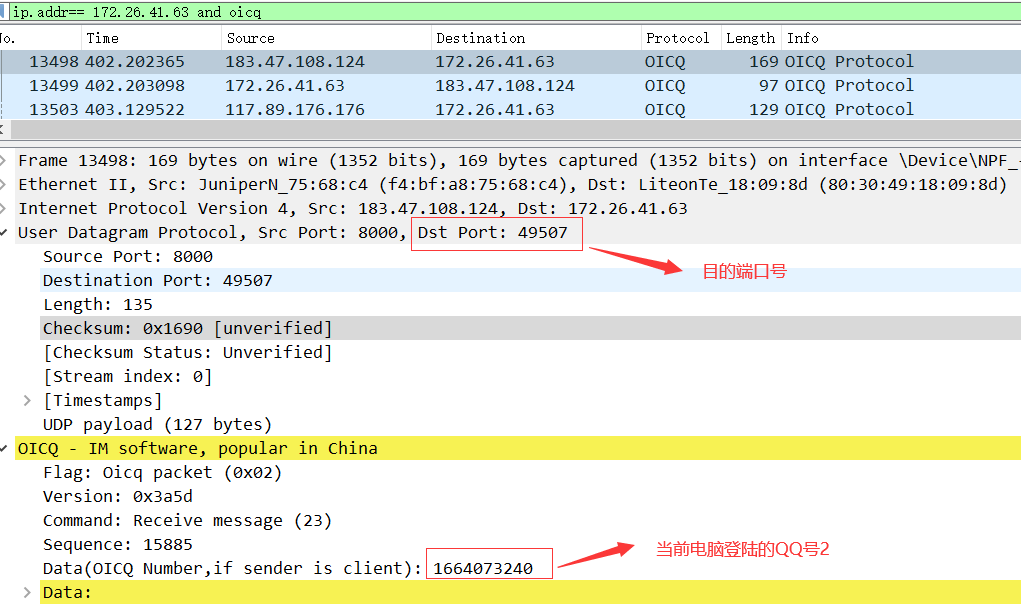
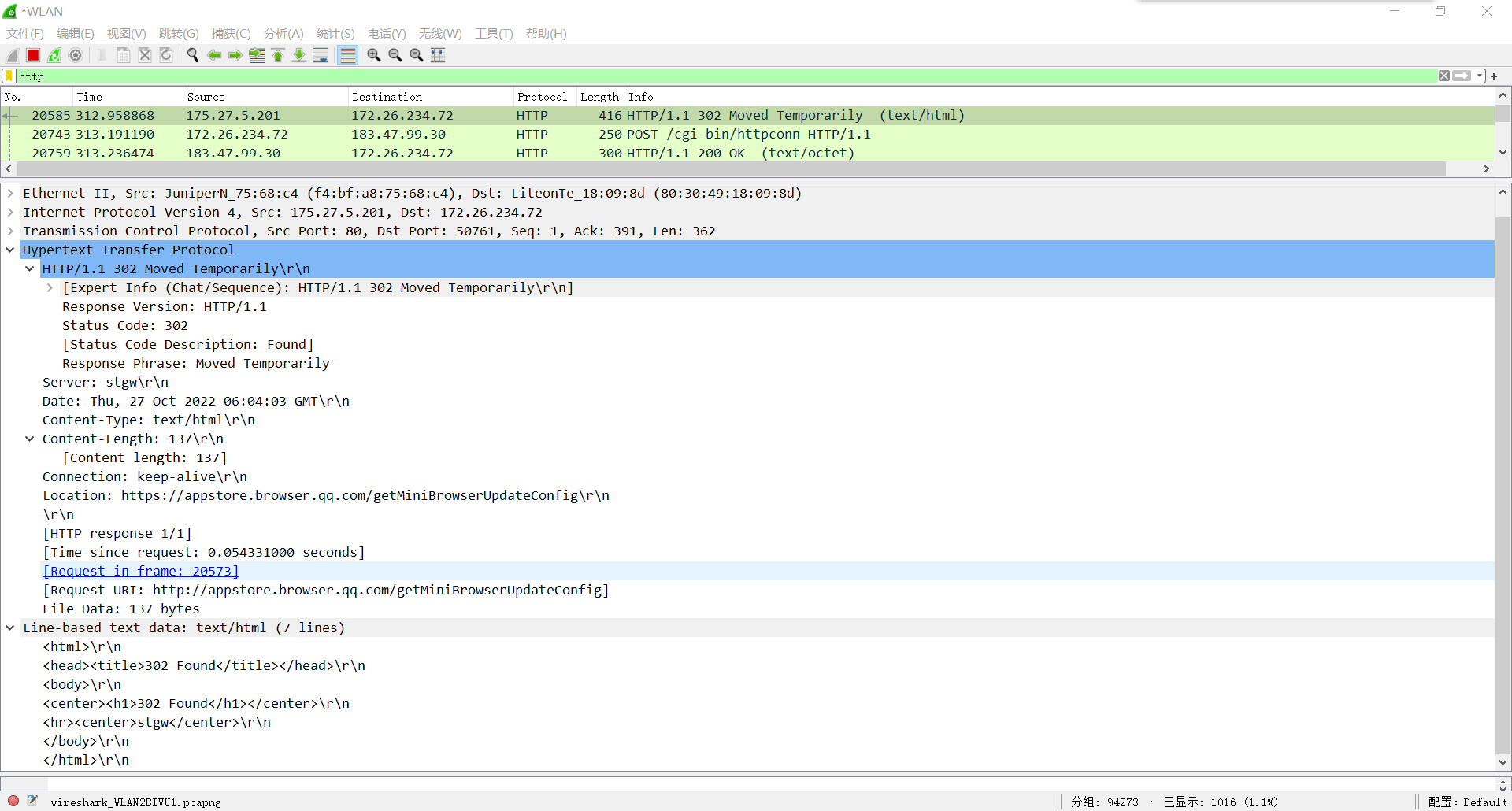


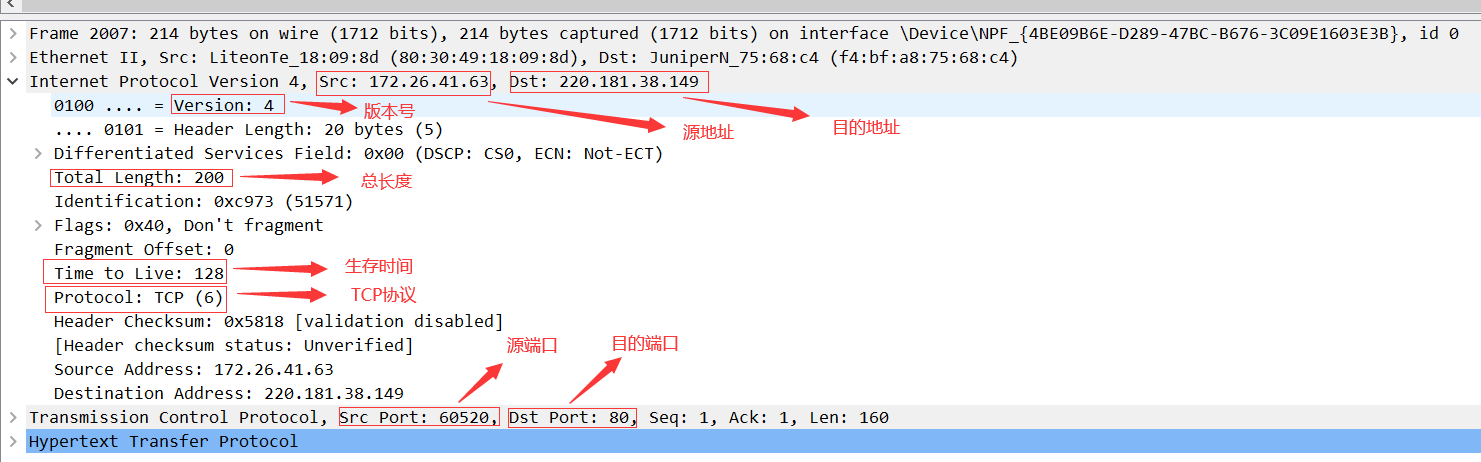
图 12 QQ号2通信

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。

（1）打开浏览器，捕获使用HTTP协议的数据包；

图 13 HTTP协议数据报

（2）运输层协议分析：

图 14 HTTP协议分析

由图14容易知，浏览器使用的应用层协议为HTTP协议，使用的运输层协议是TCP协议，原因在于：HTTP协议中的数据是利用TCP协议传输的,特点是客户端发送的每次请求都需要服务器回送响应,它是TCP协议族中的一种,默认使用 TCP 80端口。

（3）HTTP协议分析：

* HTTP的请求包括：请求行、请求头部、空行和请求数据四个部分组成。

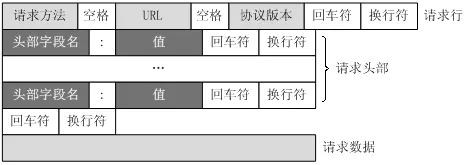


图 15 HTTP请求报文

**请求行：**可以指出请求类型，要访问的资源和协议版本

**请求头部：**从第二行起为请求头部，能够指出请求的目的地（主机域名），客户端的信息，它是检测浏览器类型的重要信息，由浏览器定义，并且在每个请求中自动发送。

**空行：**请求头后面必须有一个空行

**请求数据：**请求的数据也叫请求体，可以添加任意的其它数据。

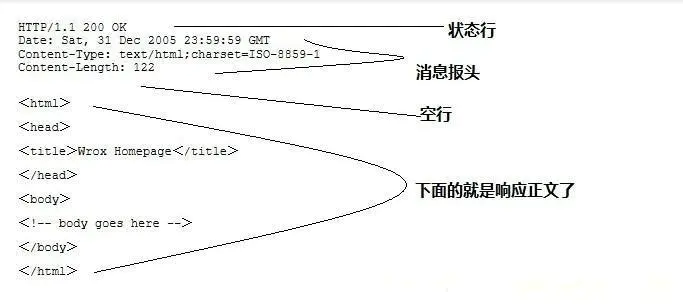
* HTTP的响应消息，HTTP响应也由4部分组成，分别是：状态行、响应头、空行和响应体。

图 16 HTTP响应报文

**状态行：**状态行由协议版本号、状态码、状态消息组成

**响应头：**响应头是客户端可以使用的一些信息，如：Date（生成响应的日期）、Content-Type（MIME类型及编码格式）、Connection（默认是长连接）等等

**空行：**响应头和响应体之间必须有一个空行

**响应体：**响应正文，本例中是键值对信息

**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;

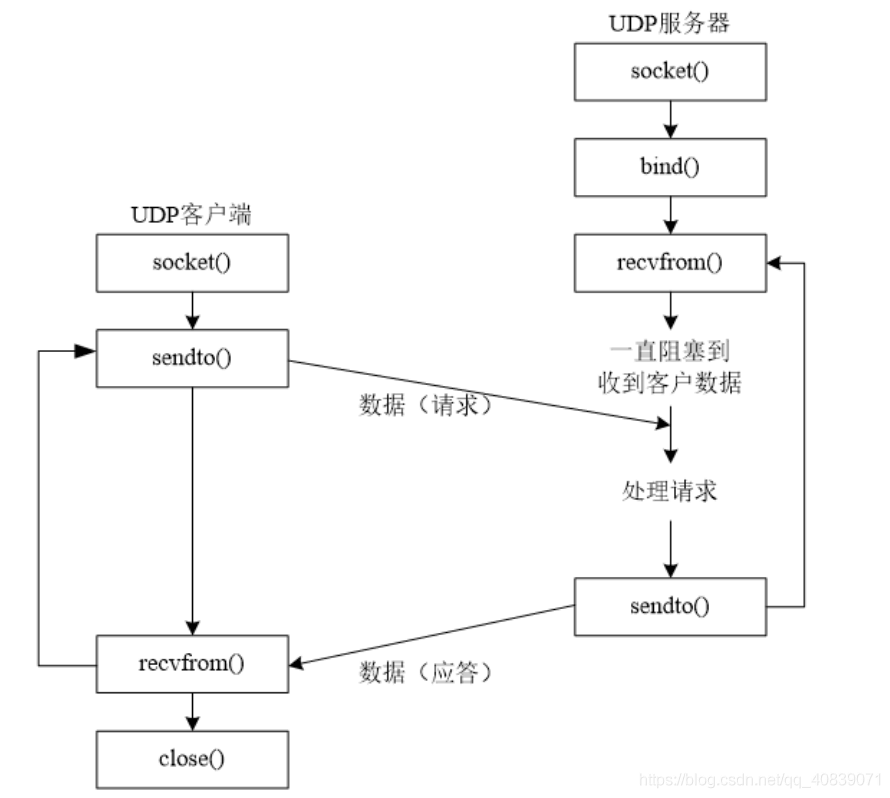


图 17 UDP通信流程图

图17中函数的作用：

socket():创建socket

bind():绑定socket到本地地址和端口，通常由服务端调用

sendto():UDP专用，发送数据到指定的IP地址和端口

recvfrom():UDP专用，返回数据到指定的IP地址和端口

close():关闭socket

网络应用程序都是基于C/S(客户端/服务器)模式的，因此在进行网络应用程序开发时，不仅要开发服务器应用程序也要开发客户端应用程序。开发服务器应用程序和客户端应用程序在步骤上略有不同。UDP的网络应用程序开发的步骤：

UDP客户端，如图17所示：

1. 创建套接字socket；
2. 使用sendto()函数向服务器发送数据（请求）；
3. 使用recvfrom()函数接收来自服务器的数据（应答）；
4. 重复（2）和（3）
5. 关闭套接字socket。

UDP服务器端，如图17所示：

（1）创建套接字socket；

（2）使用bind()函数将套接字绑定到指定的IP地址和端口上；

（3）检测到数据使用recvfrom()函数接收数据；

（4）处理请求后，使用sendto()函数发送数据；

（5）循环等待，重复（3）和（4）；

（6）关闭套接字socket。

UDP通信实验截图如图18所示：

IP地址为：127.0.0.1，客户机端口号为4567

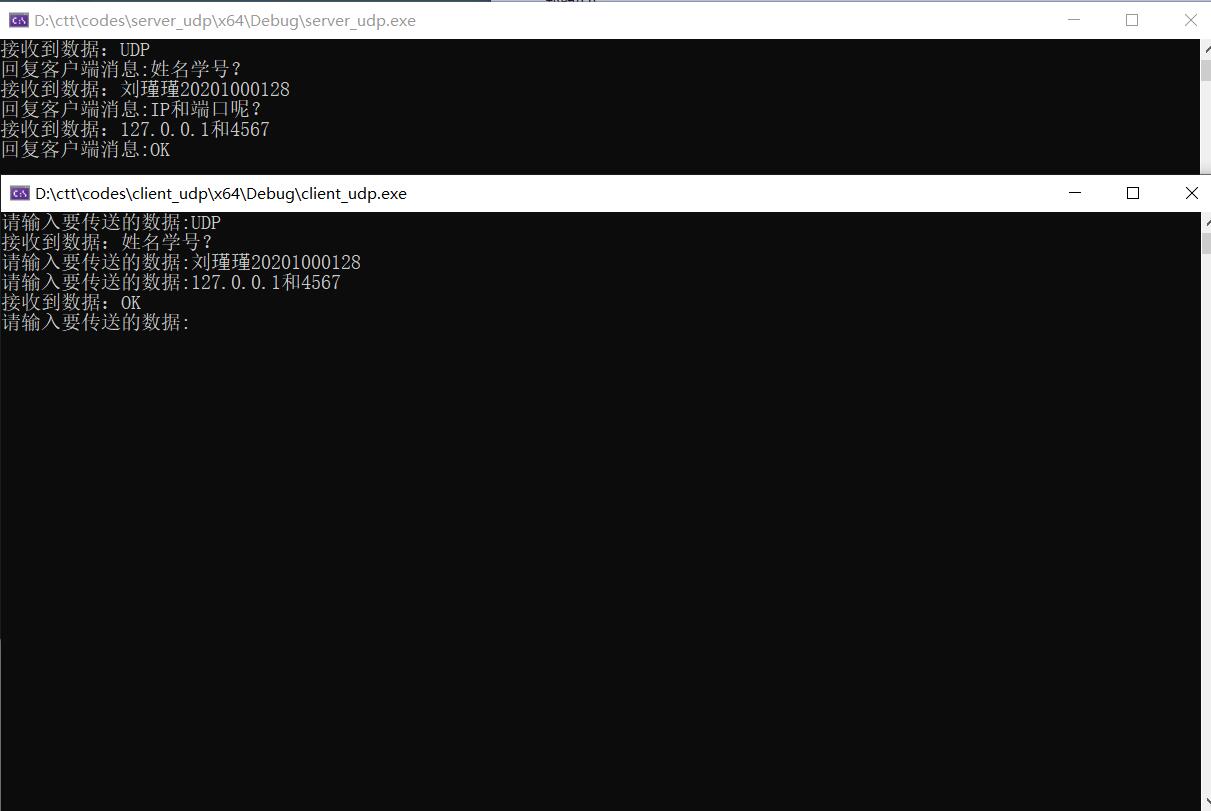


图 18 UDP通信实验

1. 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P地址和端口号;请打印相关截图贴在作业本上，并简要分析。

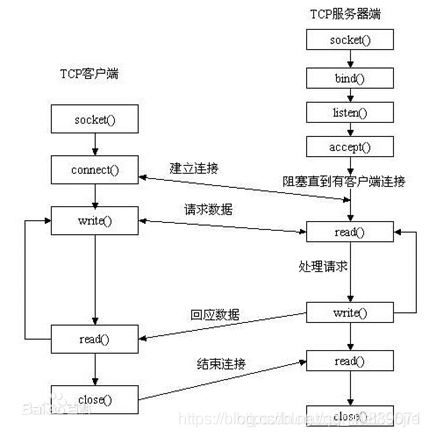


图 19 TCP通信流程图

图19中各个函数的作用：

socket():创建socket

bind():绑定socket到本地地址和端口，通常由服务端调用

listen():TCP专用，开启监听模式

accept():TCP专用，服务器等待客户端连接，一般是阻塞态

connect():TCP专用，客户端主动连接服务器

send()/write():TCP专用，发送数据

recv()/read():TCP专用，接收数据

close():关闭socket

网络应用程序都是基于C/S(客户端/服务器)模式的，因此在进行网络应用程序开发时，不仅要开发服务器应用程序也要开发客户端应用程序。开发服务器应用程序和客户端应用程序在步骤上略有不同。TCP的网络应用程序开发的步骤：

TCP客户端，如图19所示：

1. 创建套接字socket；
2. 客户机使用connect()函数连接服务器；

（3）使用send()函数向服务器发送数据（请求）；

（4）使用recv()函数接收来自服务器的数据（应答）；

（5）重复（3）和（4）；

（6）关闭套接字socket。

TCP服务器端，如图19所示：

（1）创建套接字socket；

（2）使用bind()函数将套接字绑定到指定的IP地址和端口上；

（3）将套接字设置为监听模式（listen），准备接受客户端的请求；

（4）等待客户端请求的到来(accept),阻塞直到有客户端建立连接；

（5）检测到数据使用recv()函数接收数据；

（6）处理请求后，使用send()函数发送数据；

（7）循环等待，重复（5）、（6）；

（8）检测到客户端套接字socket关闭，结束连接；

（9）关闭套接字socket。

TCP通信实验截图如图20所示：

IP地址为：127.0.0.1，端口号为4999。

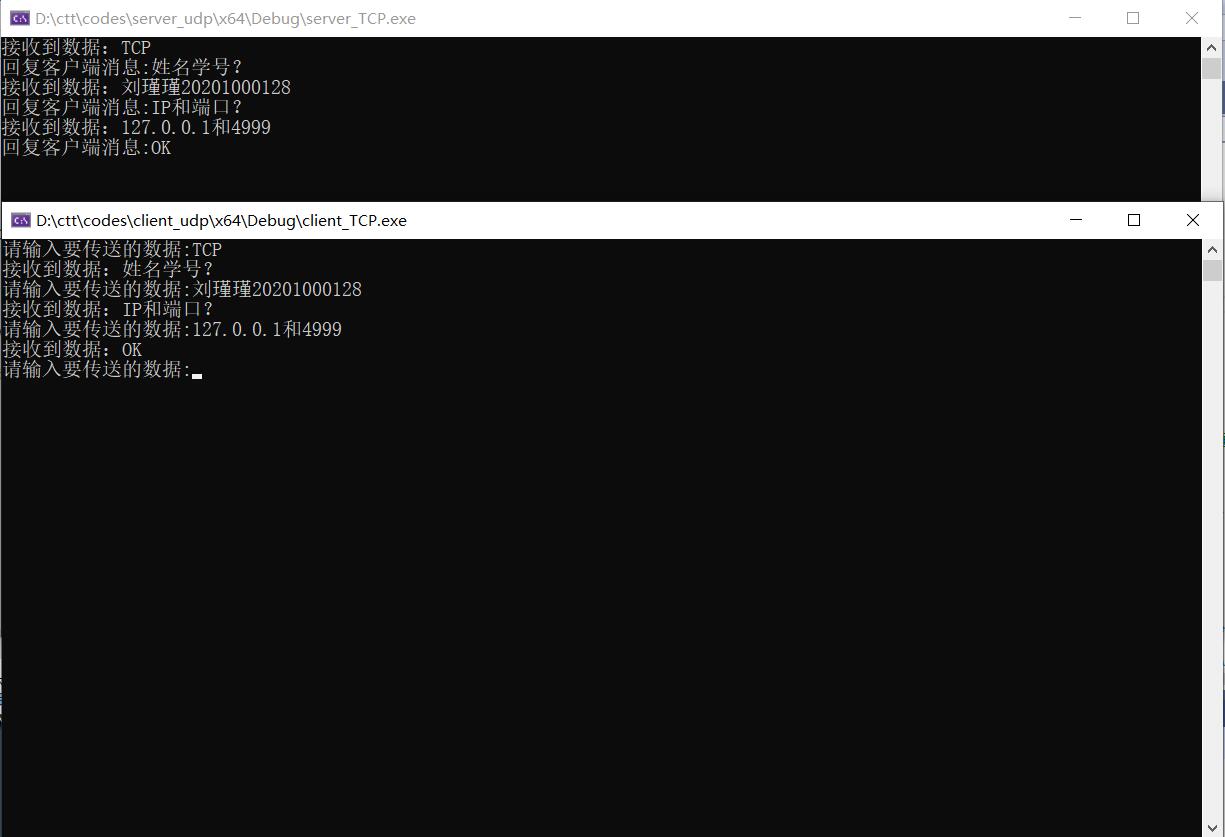


图 20 TCP通信实验

**附录**

UDP客户端：

//#include "stdafx.h"

#include <stdlib.h>

#include <Winsock2.h>

#include <stdio.h>

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

void main()

{

WORD wVersionRequested;

WSADATA wsaData;

int err;

wVersionRequested = MAKEWORD(1, 1);

err = WSAStartup(wVersionRequested, &wsaData);

if (err != 0)

{

return;

}

if (LOBYTE(wsaData.wVersion) != 1 ||

HIBYTE(wsaData.wVersion) != 1)

{

WSACleanup();

return;

}

SOCKET sockClient = socket(AF\_INET, SOCK\_DGRAM, 0);

SOCKADDR\_IN addrClient;

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

addrClient.sin\_family = AF\_INET;

addrClient.sin\_port = htons(6000);

char buff[100];

sprintf(buff, "This is UDP client! Name:刘瑾瑾 Student\_Number:20201000128 IP\_Address:%s Port:%s", inet\_ntoa(addrClient.sin\_addr), "6000");

sendto(sockClient, buff, strlen(buff) + 1, 0,(const sockaddr\*)&addrClient, sizeof(SOCKADDR));

char recvBuf[100];

int len = sizeof(SOCKADDR);

recvfrom(sockClient, recvBuf, 100, 0, (sockaddr\*)&addrClient, &len);

printf("%s", recvBuf);

closesocket(sockClient);

WSACleanup();

system("pause");

}

UDP服务器端：

//#include "stdafx.h"

#include <stdlib.h>

#include <Winsock2.h>

#include <stdio.h>

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

void main()

{

WORD wVersionRequested;

WSADATA wsaData;

int err;

wVersionRequested = MAKEWORD(1, 1);

err = WSAStartup(wVersionRequested, &wsaData);

if (err != 0)

{

return;

}

if (LOBYTE(wsaData.wVersion) != 1 ||

HIBYTE(wsaData.wVersion) != 1)

{

WSACleanup();

return;

}

SOCKET sockSrv = socket(AF\_INET, SOCK\_DGRAM, 0);

SOCKADDR\_IN addrSrv;

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

addrSrv.sin\_family = AF\_INET;

addrSrv.sin\_port = htons(6000);

bind(sockSrv, (const sockaddr\*)&addrSrv, sizeof(SOCKADDR));

SOCKADDR\_IN addrClient;

char recvBuf[100];

int len = sizeof(SOCKADDR);

recvfrom(sockSrv, recvBuf, 100, 0, (sockaddr\*)&addrClient, &len);

printf("%s", recvBuf);

char sendBuf[100];

sprintf(sendBuf, "This is UDP server! Welcome IP:%s,Port:%s", inet\_ntoa(addrClient.sin\_addr),"6000");

sendto(sockSrv, sendBuf, strlen(sendBuf) + 1, 0,(const sockaddr\*)&addrClient, sizeof(SOCKADDR));

closesocket(sockSrv);

WSACleanup();

system("pause");

}

TCP客户端：

//#include "stdafx.h"

#include <stdlib.h>

#include <Winsock2.h>

#include <stdio.h>

#include <iostream>

using namespace std;

#pragma comment(lib, "WS2\_32")

void main()

{

WORD wVersionRequested;//双字节型

WSADATA wsaData;

int err;

wVersionRequested = MAKEWORD(1, 1);

err = WSAStartup(wVersionRequested, &wsaData);//调用WSAStartup函数 为0：初始化成功

if (err != 0)

{

cout << "发生错误！" << endl;

return;

}

if (LOBYTE(wsaData.wVersion) != 1 ||

HIBYTE(wsaData.wVersion) != 1)

{

WSACleanup();//调用 WSACleanup 时，将取消此过程中任何线程发出的挂起阻止或异步 Windows 套接字调用

cout << "已关闭！" << endl;

return;

}

SOCKET sockClient = socket(AF\_INET, SOCK\_STREAM, 0);//TCP

SOCKADDR\_IN addrSrv;

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

addrSrv.sin\_family = AF\_INET;

addrSrv.sin\_port = htons(6000);

connect(sockClient, (const sockaddr\*)&addrSrv, sizeof(SOCKADDR));

char recvbuf[100];

recv(sockClient, recvbuf, strlen(recvbuf) + 1, 0);

printf("%s", recvbuf);

/\*string s;

char buff[100];

cin >> s;

for (int i = 0; i < sizeof(s); i++) {

buff[0] = s[0];

}\*/

char buff[100] = "This is TCP client! Name:刘瑾瑾 Student\_Number:20201000128 IP\_Address:127.0.0.1 Port:6000";

send(sockClient,buff, strlen(buff) + 1, 0);

closesocket(sockClient);

WSACleanup();

system("PAUSE");

}

TCP服务器端：

#include <stdlib.h>

#include <Winsock2.h>

#include <stdio.h>

#pragma comment(lib, "WS2\_32")

#define PORT 6000

void main()

{

WORD wVersionRequested;

WSADATA wsaData;

int err;

wVersionRequested = MAKEWORD(1, 1);

err = WSAStartup(wVersionRequested, &wsaData);

if (err != 0)

{

return;

}

if (LOBYTE(wsaData.wVersion) != 1 ||

HIBYTE(wsaData.wVersion) != 1)

{

WSACleanup();

return;

}

SOCKET sockSrv = socket(AF\_INET, SOCK\_STREAM, 0);

SOCKADDR\_IN addrSrv;

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

addrSrv.sin\_family = AF\_INET;

addrSrv.sin\_port = htons(PORT);

bind(sockSrv, (const sockaddr\*)&addrSrv, sizeof(SOCKADDR));

listen(sockSrv, 5);

SOCKADDR\_IN addrClient;

int len = sizeof(SOCKADDR);

while (1)

{

SOCKET sockCon = accept(sockSrv, (SOCKADDR\*)&addrClient, &len);

char sendBuf[100];

sprintf(sendBuf, "This is TCP server,Welcome IP:%s,Port:%s", inet\_ntoa(addrClient.sin\_addr),"6000");

send(sockCon, sendBuf, strlen(sendBuf) + 1, 0);

char recvbuf[100];

recv(sockCon, recvbuf, strlen(recvbuf) + 1, 0);

printf("%s\n", recvbuf);

closesocket(sockCon);

}

system("PAUSE");

}