目录

实验一 协议与数据包分析实验

一、目的:

1. 熟悉 Wireshark 抓包软件的使用方法；
2. 了解 IP 数据报的结构；
3. 了解传输层 UDP 和 TCP 协议报文的结构；
4. 了解常用应用层协议的特点；

二、实验环境

1. 设备：具有上网功能的电脑一台；
2. 软件：Windows 操作系统；Wireshark 软件；

三、实验步骤

1. 安装 Wireshark 软件，并了解各个菜单的功能；
2. 关闭所有具有通信功能的软件，比如浏览器、QQ、微信电脑版等；
3. 运行 Wireshark 软件；
4. 依次单独运行浏览器、QQ(进行文本、视频通信)、在线视频播放软件、迅雷下载等常用软件，并使用 Wireshark 进行抓包操作；
5. 分析 IP 数据报、UDP 和 TCP 报文结构、HTTP 等应用层协议的数据包结构。

四、实验内容

实验一中将对 IP 数据报的报文结构进行相关分析，以及 UDP 和 TCP 首部分析，再使用 QQ 进行通信，查看文本和视频通信所使用的端口号以及传输层协议，最后再对 HTTP 报文进行相关分析。具体内容如下：

IP 数据报

常规 IP 数据报结构如图 1 所示：

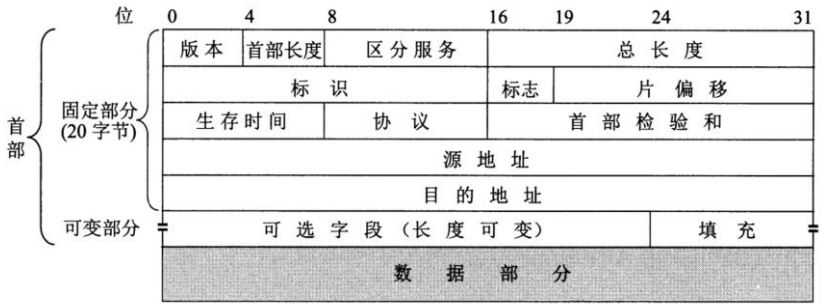

图 1 IP 数据报结构示意图

利用 Wireshark 进行抓包，选取其中一条 TCP 协议数据报，其 IP 数据报结构如图 2 所示：

```
Internet Protocol Version 4, Src: 172.26.36.43, Dst: 58.49.138.222
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0xfe9f (65183)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.36.43
    Destination Address: 58.49.138.222
```

图 2 Wireshark IP 数据报抓包结果

观察 IP 数据报抓包对比 IP 数据报结构可以获取以下信息如表 1 所示：

表 1 IP 数据报实例表

字段	内容	含义
版本	0b0100	IP 协议为 IPv4
首部长度	0b0101	首部长度为 20 字节
区分服务	0	不使用区分服务
总长度	0x0028	数据报总长度 40 字节
标识	0x40	将 0x40 作为数据报标识
标志	0	是若干数据报中最后一个
片偏移	0	片偏移为 0
生存时间	0x80	生存时间为 128 秒
协议	0x06	采用协议为 TCP
首部检验和	0	首部未发生变化，保留
源地址	0xac1a242b	源地址：172.26.36.43
目的地址	0x3a318ade	目的地址：58.49.138.222

TCP 数据报首部

常规 TCP 数据报结构如图 3 所示：

图 3 TCP 数据报结构示意图

利用 Wireshark 进行抓包，选取其中一条 TCP 协议数据报，TCP 数据报首部内容如图 4 所示：

```
Transmission Control Protocol, Src Port: 55113, Dst Port: 80, Seq: 618, Ack: 38518, Len: 0
    Source Port: 55113
    Destination Port: 80
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence Number: 618    (relative sequence number)
    Sequence Number (raw): 274024134
    [Next Sequence Number: 618    (relative sequence number)]
    Acknowledgment Number: 38518    (relative ack number)
    Acknowledgment number (raw): 1333828518
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 32768
    [Calculated window size: 65536]
    [Window size scaling factor: 2]
    Checksum: 0x956f [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
```

图 4 Wireshark TCP 数据报抓包结果

观察 TCP 数据报抓包对比 TCP 数据报结构可以获取以下信息如表 2 所示：

表 2 TCP 数据报首部实例表

字段	内容	含义
源端口	0xd749	源端口号为 55113
目的端口	0x0050	目的端口号为 80
序号	0x105546c6	报文序号为 618
确认号	0x4f809ba6	报文确认号为 38518
数据偏移	0x50	TCP 首部长度为 20 字节
保留	0	/
URG	0	无紧急数据
ACK	1	确认号字段有效
PSH	0	不进行推送操作
RST	0	TCP 连接无严重错误
SYN	0	不是连接请求
FIN	0	不释放运输连接
窗口	0x8000	窗口大小为 32768
检验和	0x956f	数据报检验和为
紧急指针	0	URG=0 无意义
选项	/	/
填充	/	/

UDP 数据报首部

常规 UDP 数据报结构如图 5 所示：

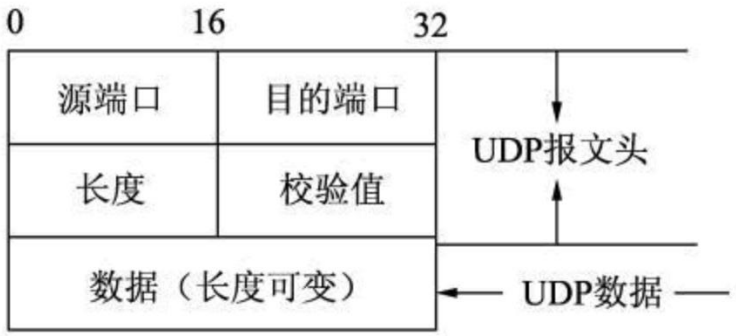

图 5 UDP 数据报结构示意图

利用 Wireshark 进行抓包，选取其中一条 UDP 协议数据报，UDP 数据报首部内容如图 6 所示：

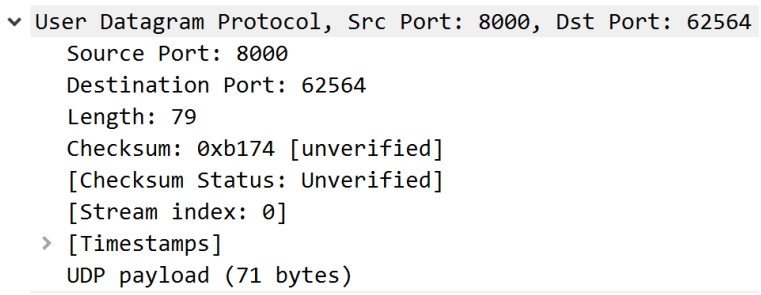

图 6 Wireshark UDP 数据报抓包结果

观察 UDP 数据报抓包对比 UDP 数据报结构可以获取以下信息如表 3 所示：

表 3 UDP 数据报首部实例表

字段	内容	含义
源端口	0x1f40	源端口号为 8000
目的端口	0xf464	目的端口号为 62564
长度	0x004f	UDP 用户数据报长度为 79 字节
检验和	0xb174	检验和为 0xb174

QQ 抓包

文本发送

利用 QQ 向好友发送消息，用 Wireshark 抓包如图 7 所示：

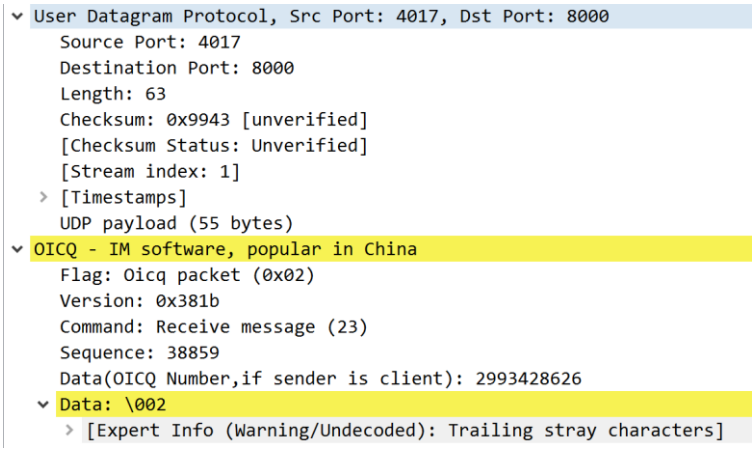

图 7 Wireshark QQ 文本发送数据报抓包结果

通过观察抓包结果，可以得出以下信息：

1. 端口：源端口为 4017 即本机端口，目的端口为 8000 为国内 QQ 主要使用端口。
2. 协议：采用的传输层协议为 UDP 协议，其中数据报的目的 IP 地址为：223.166.151.94，并非我发送信息的目的 IP，经查找可以发现该 IP 隶属于上海市，初步猜测是腾讯位于上海的服务器 IP，本机将数据传输给该服务器后由服务器转发至最终的目的 IP。
3. 数据内容：从抓包结果可以看出，Data 显示了使用的 QQ 账号为本人 QQ 账号，且对数据进行 follow UDP stream 操作，可以看出发送的数据被进行了加密，显示为乱码。

视频通话

通过电脑 QQ 向 IP 地址为 113.57.246.219 的进行视频通话，用 Wireshark 进行抓包如图所示：

```
User Datagram Protocol, Src Port: 52744, Dst Port: 33015
    Source Port: 52744
    Destination Port: 33015
    Length: 80
    Checksum: 0x8a64 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 6]
  > [Timestamps]
    UDP payload (72 bytes)
Data (72 bytes)
    Data: 020048000100000e0600002643e3f700000000b26c18920000563c000000005bd48f376d…
    [Length: 72]
```

图 8 Wireshark QQ 视频通话数据报抓包结果

通过观察抓包结果，可以得出以下信息：

1. 端口：源端口为 52744 即本机端口，目的端口为 33015 即对方手机端口。
2. 协议：视频通话所采用的协议为 UDP，并相比于文本通信，视频通话可以直接与目的 IP 进行通讯，而文本往往需要经过腾讯内部的服务器进行。
3. 数据：相比文本通讯，视频通话可以直接看到每一次传输所发送的数据，并在 Wireshark 中可以直接查看。

HTTP 报文结构

使用百度浏览器访问网站，可以对 HTTP 协议进行相关分析。

HTTP 请求报文

HTTP 协议通常由两种报文结构，其中请求报文的结构如图 9 所示：

图 9 HTTP 请求报文结构示意图

通过 Wireshark 软件对访问网站过程进行抓包可以得出抓包结果如下所示：

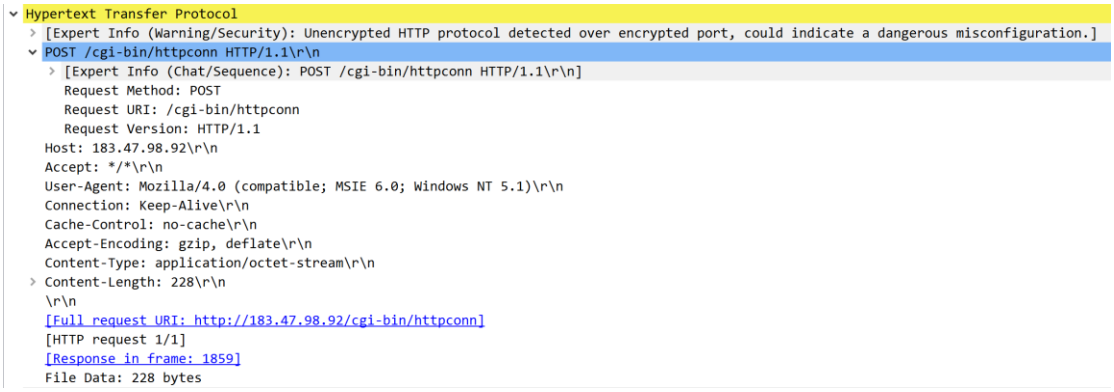

图 10 Wireshark HTTP 请求报文抓包结果

从 Wireshark 抓包结果可以看出，HTTP 数据报请求方法为 POST，URL 为 /cgi-bin/httpconn，所采用的 HTTP 协议版本为 HTTP/1.1，这三者共同构成了请求报文的请求行。

请求头部由头部字段号和值构成通过 Wireshark 界面可以明显看出报文的请求头部，包括 Host，Accept，User-Agent 等头部字段名。

在空白行后即为报文的请求正文部分。

HTTP 响应报文

HTTP 协议响应报文的结构如图 11 所示：

图 11 HTTP 请求报文结构示意图

通过 Wireshark 软件对访问网站过程进行抓包可以得出抓包结果如下所示：

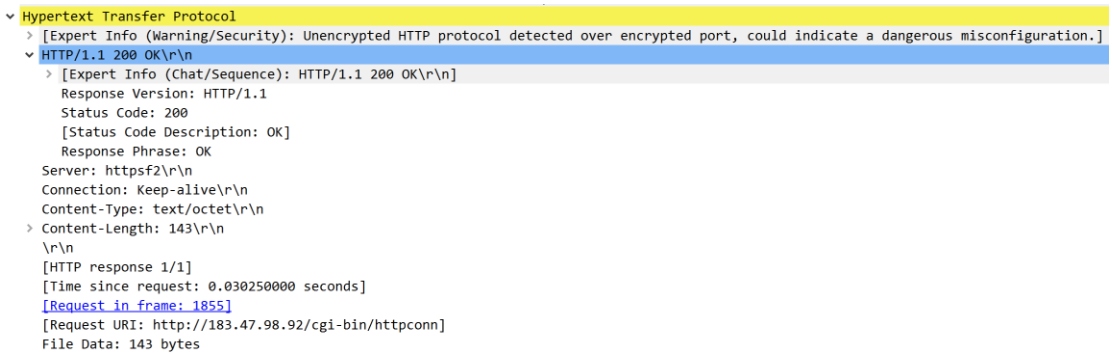

图 12 Wireshark HTTP 请求报文抓包结果

从结果可以看出 HTTP 请求报文版本协议为 HTTP/1.1，状态码为 200，状态码描述为 0x323230 表示 OK。

响应头部由头部字段号和值构成通过 Wireshark 界面可以明显看出报文的请求头部，包括 Server，Connection，Content-Type，Content-Length 等头部字段名。

在空白行后即为报文的请求正文部分。

实验二 Socket 通信实验

一、目的:

熟悉传输层 UDP 和 TCP 协议的报文结构;

了解 Socket 的结构和编程实现，网络通信程序的基本原理和编写方法。

二、实验环境

设备：具有上网功能电脑、局域网环境;

软件：Windows 操作系统；C++编程环境;

三、实验步骤

使用 Socket 类库，开发基于 UDP、TCP 的两个通信程序，并实现文本通信;

四、实验内容

TCP Socket 实现

原理

TCP Socket 实现流程如图 13 所示，其具体函数介绍如下：

socket()

socket()函数的作用在于生成一个用于通信的套接字文件描述符，这个套接字描述符可以用作 bind()函数的绑定对象，在客户端则用于 connect()函数。

```
ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
    ptr->ai_protocol);
if (ConnectSocket == INVALID_SOCKET) {
    printf("socket failed with error: %ld\n", WSAGetLastError());
    WSACleanup();
    return 1;
}
```

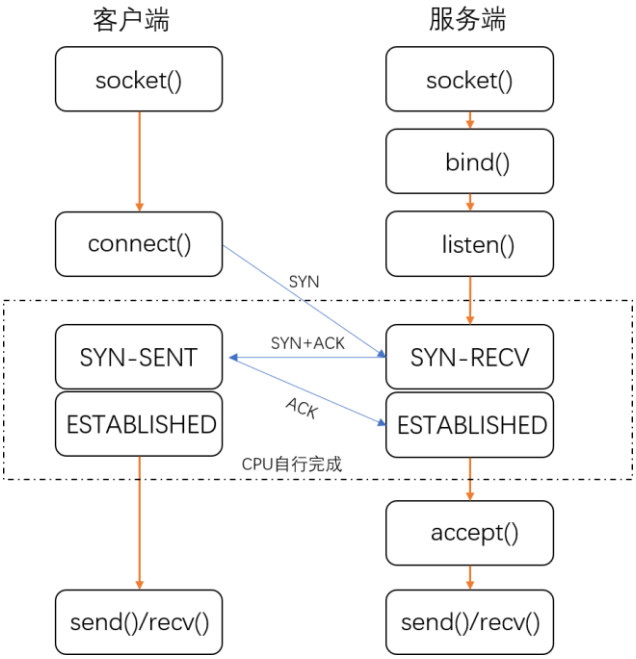

图 13 TCP Socket 流程示意图

bind()

服务器通过分析配置文件，从中解析出想要监听的地址和端口，再加上可以通过 socket()函数生成的套接字，可以使用 bind()函数将套接字绑定到想要连接的地址和端口组合上，绑定了地址和端口的套接字可以作为 listen()函数的监听对象。

```
1.  iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen
    );
2.  if (iResult == SOCKET_ERROR) {
3.      printf("bind failed with error: %d\n", WSAGetLastError());
4.      freeaddrinfo(result);
5.      closesocket(ListenSocket);
6.      WSACleanup();
7.      return 1;
8.  }
```

listen()

listen()用来监听已经绑定了的套接字。

```
1.  iResult = listen(ListenSocket, SOMAXCONN);
2.  if (iResult == SOCKET_ERROR) {
3.      printf("listen failed with error: %d\n", WSAGetLastError());
4.      closesocket(ListenSocket);
5.      WSACleanup();
6.      return 1;
7.  }
```

Connect()

用于向某个已经监听的套接字发起连接请求，也就是发起 TCP 的三次握手过程。

```
iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

if (iResult == SOCKET_ERROR) {
    closesocket(ConnectSocket);
    ConnectSocket = INVALID_SOCKET;
    continue;
}
```

accept()

用于读取已完成连接队伍中的第一项，并对此项生成一个用于后续连接的套接字描述符，工作进程通过新的套接字和客户端进行数据传输。

```
ClientSocket = accept(ListenSocket, NULL, NULL);
if (ClientSocket == INVALID_SOCKET) {
    printf("accept failed with error: %d\n", WSAGetLastError());
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
}
```

send() recv()

send()将数据复制到 send buffer 中，recv()函数用于将数据从 recv buffer 中读出。

```
iSendResult = send(ClientSocket, sendbuf, (int)strlen(sendbuf), 0);
if (iSendResult == SOCKET_ERROR) {
    printf("send failed with error: %d\n", WSAGetLastError());
    closesocket(ClientSocket);
    WSACleanup();
    return 1;
}

ult = recv(ClientSocket, recvbuf, recvbuflen, 0);
```

结果

将 C++代码移植到 Qt 中进行开发，按照题意进行测试可得 TCP 协议 Socket 通信结果如图 14、15 所示。

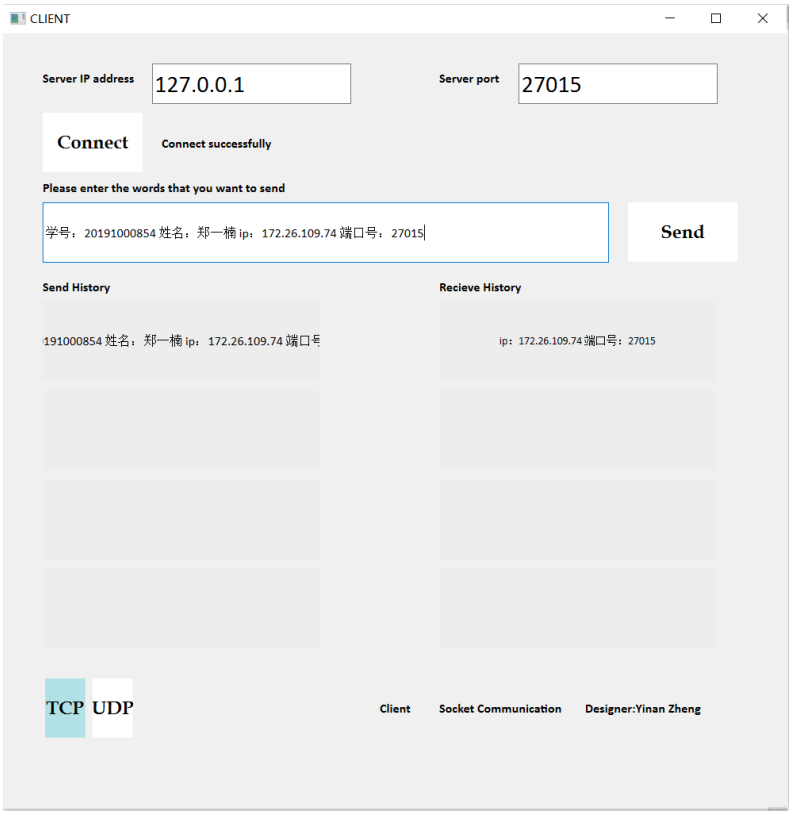

图 14 TCP 通信客户端界面

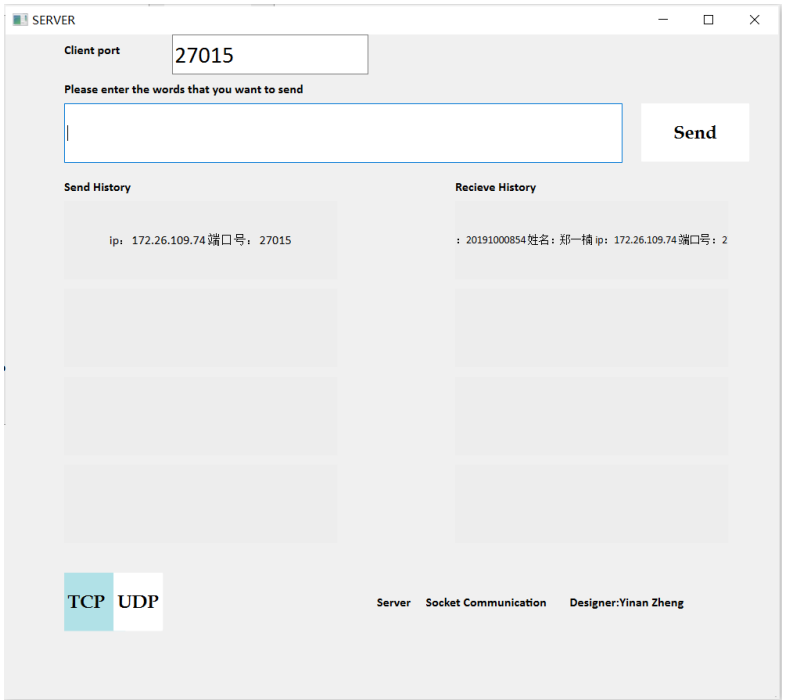

图 15 TCP 通信服务器端界面

UDP Socket 实现

原理

UDP Socket 实现流程如图 16 所示，其部分具体函数介绍如下：

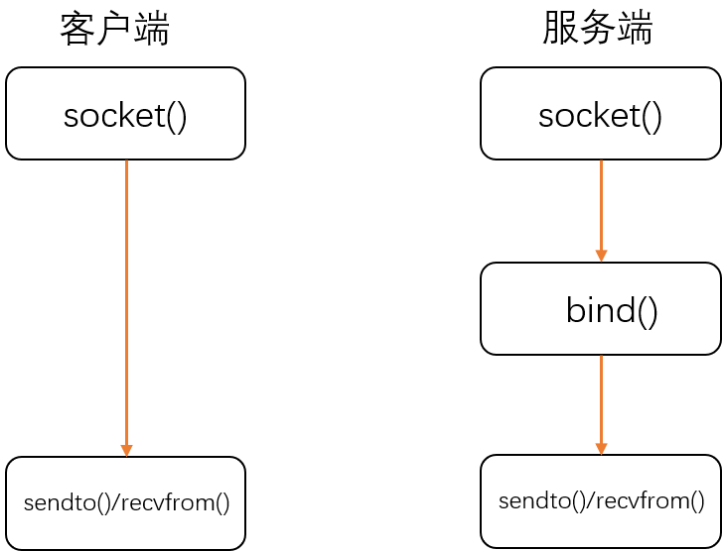

图 16 TCP Socket 流程示意图

sendto() recvfrom

函数 sendto()和 recvfrom()与 send()和 recv()类似，都用于数据的发送和接收，但是由于前者用于 UDP 通讯，需要每次提供 IP 地址和端口号，而 TCP 下则不需要。

```
iResult = sendto(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0, (SOCKADDR*)result->ai_addr, result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    printf("send failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}
iResult = recvfrom(ConnectSocket, recvbuf, recvbuflen, 0, (SOCKADDR*)&RecverAddr, &RecverAddrSize);
if (iResult > 0)
{
    cout << recvbuf << endl;
    printf("Bytes received: %d\n", iResult);
}
```

结果

按照题意进行测试可得 TCP 协议 Socket 通信结果如图 17、18 所示。

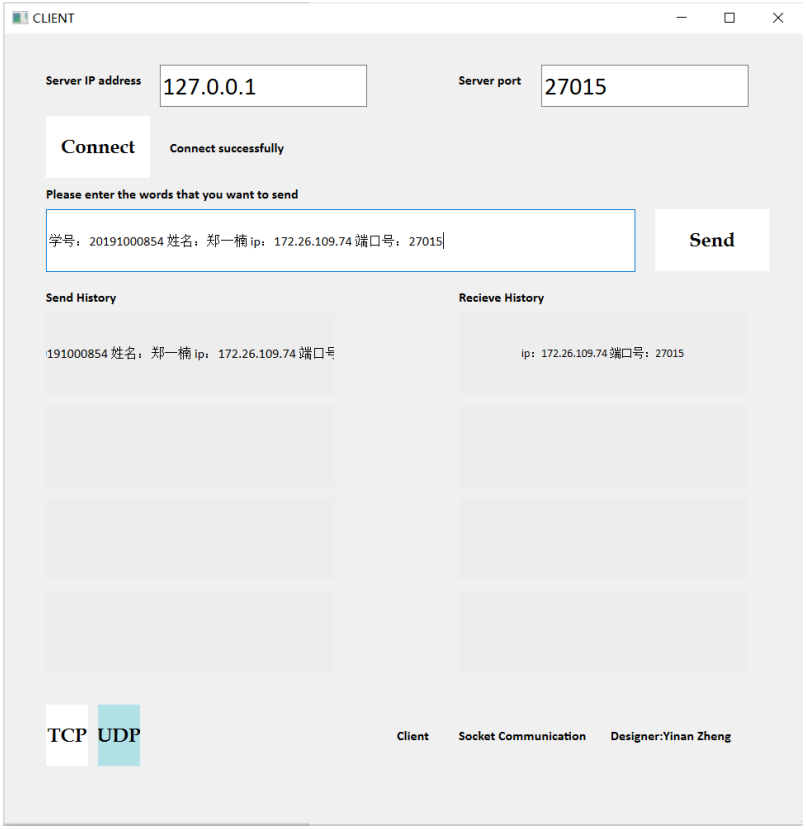

图 17 UDP 通信客户端界面

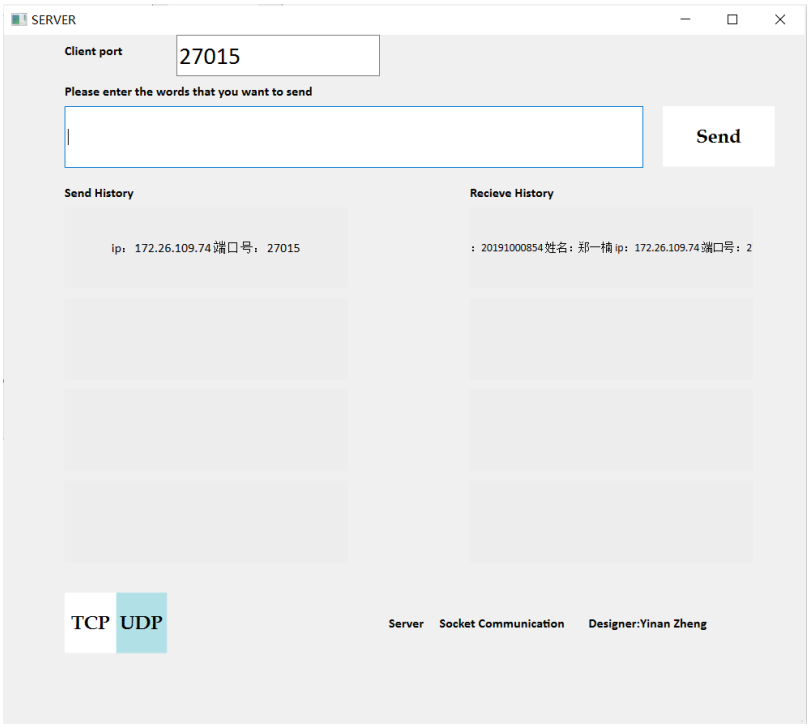

图 18 UDP 通信服务器端界面

附录

sockettcpserver.cpp

```
1.  #include "sockettcpserver.h"
2.  #include "ui_sockettcpserver.h"
3.
4.  #include <QMessageBox>
5.
6.  SocketTCPServer::SocketTCPServer(QWidget *parent) :
7.      QDialog(parent),
8.      ui(new Ui::SocketTCPServer)
9.  {
10.     ui->setupUi(this);
11.
12.     ui->m_portLineEdit->setText("5550");
13.
14.     connect(ui->m_initSocketBtn, SIGNAL(clicked()), this, SLOT(OnBtnI
    nitSocket()));
15.     connect(ui->m_sendData, SIGNAL(clicked()), this, SLOT(OnBtnSendDa
    ta()));
16. }
17.
18. SocketTCPServer::~SocketTCPServer()
19. {
20.     delete ui;
21. }
22.
23. void SocketTCPServer::ServerNewConnection()
24. {
25.     //获取客户端连接
26.     mp_TCPSocket = mp_TCPServer->nextPendingConnection();
27.
28.     if(!mp_TCPSocket)
29.     {
30.         QMessageBox::information(this, "QT 网络通信", "未正确获取客户端
    连接！");
31.         return;
32.     }
33.     else
34.     {
```

```
35.         QMessageBox::information(this, "QT 网络通信", "成功接受客户端的
    连接");
36.
37.         //当有数据接收时，会触发信号 SIGNAL:readyRead()，此时执行槽函数
    ServerReadData()
38.         connect(mp_TCPSocket, SIGNAL(readyRead()), this, SLOT(ServerR
    eadData()));
39.
40.         //当断开连接时
41.         connect(mp_TCPSocket, SIGNAL(disconnected()), this, SLOT(sSer
    verDisConnection()));
42.     }
43. }
44.
45. void SocketTCPServer::ServerReadData()
46. {
47.     QByteArray buffer;
48.
49.     buffer = mp_TCPSocket->read(1024);
50.
51.     if(buffer.isEmpty())
52.     {
53.         QMessageBox::information(this, "QT 网络通信", "未正确接收数据
    ");
54.         return;
55.     }
56.     else
57.     {
58.         QString showMsg = buffer;
59.         ui->m_recvDataTextEdit->append(showMsg);
60.     }
61.     /*
62.     char buffer[1024] = {0};
63.
64.     mp_TCPSocket->read(buffer, 1024);
65.
66.     if( strlen(buffer) > 0)
67.     {
68.         QString showNsg = buffer;
69.         ui->m_recvDataTextEdit->append(showNsg);
70.     }
71.     else
72.     {
```

```
73.         QMessageBox::information(this, "QT 网络通信", "未正确接收数据
    ");
74.         return;
75.     }
76.     */
77.
78. }
79.
80. void SocketTCPServer::OnBtnInitSocket()
81. {
82.     mp_TCPServer = new QTcpServer();
83.
84.     int port = ui->m_portLineEdit->text().toInt();
85.     if(!mp_TCPServer->listen(QHostAddress::Any, port))
86.     {
87.         QMessageBox::information(this, "QT 网络通信", "服务器端监听失
    败！");
88.         return;
89.     }
90.     else
91.     {
92.         QMessageBox::information(this, "QT 网络通信", "服务器监听成功！
    ");
93.     }
94.
95.     //当检测到有新连接时，会触发信号 SIGNAL:newConnection()，此时执行槽函
    数 ServerNewConnection()
96.     connect(mp_TCPServer, SIGNAL(newConnection()), this, SLOT(ServerN
    ewConnection()));
97. }
98.
99. void SocketTCPServer::OnBtnSendData()
100.   {
101.       char sendMsgChar[1024] = {0};
102.
103.       QString sendMsg = ui->m_inputTextEdit->toPlainText();
104.
105.       if(sendMsg.isEmpty())
106.       {
107.           QMessageBox::information(this, "QT 网络通信", "发送数据为空，
    请输入数据");
108.           return;
109.       }
110.
```

```
111.        strcpy_s(sendMsgChar, sendMsg.toStdString().c_str());
112.
113.        if(mp_TCPSocket->isValid())
114.        {
115.            int sendRe = mp_TCPSocket->write(sendMsgChar, strlen(sendM
    sgChar));
116.
117.            if( -1 == sendRe)
118.            {
119.                QMessageBox::information(this, "QT 网络通信", "服务端发送
    数据失败！");
120.            }
121.        }
122.        else
123.        {
124.            QMessageBox::information(this, "QT 网络通信", "套接字无效！
    ");
125.        }
126.    }
127.
128.    void SocketTCPServer::sServerDisConnection()
129.    {
130.        QMessageBox::information(this, "QT 网络通信", "与客户端的连接断开
    ");
131.
132.        return;
133.    }
```

sockettcpclient.cpp

```
1.  #include "sockettcpclient.h"
2.  #include "ui_sockettcpclient.h"
3.
4.  #include <QMessageBox>
5.
6.  SocketTCPClient::SocketTCPClient(QWidget *parent) :
7.      QDialog(parent),
8.      ui(new Ui::SocketTCPClient)
9.  {
10.     ui->setupUi(this);
11.
12.     ui->m_serverIPLineEdit->setText("127.0.0.1");
13.     ui->m_serverPortLineEdit_2->setText("5550");
14. }
15.
16. SocketTCPClient::~SocketTCPClient()
17. {
18.     delete ui;
19. }
20.
21. void SocketTCPClient::on_m_connectServerBtn_clicked()
22. {
23.     mp_clientSocket = new QTcpSocket();
24.
25.     QString ip = ui->m_serverIPLineEdit->text();\
26.     int port = ui->m_serverPortLineEdit_2->text().toInt();
27.
28.     mp_clientSocket->connectToHost(ip, port);
29.
30.     if(!mp_clientSocket->waitForConnected(30000))
31.     {
32.         QMessageBox::information(this, "QT 网络通信", "连接服务端失败!
    ");
33.         return;
34.     }
35.
36.     //当有消息到达时，会触发信号 SIGNAL:readyRead(), 此时就会调用槽函数
    ClientRecvData()
37.      connect(mp_clientSocket, SIGNAL(readyRead()), this, SLOT(ClientR
    ecvData()));
38. }
39.
```

```
40. void SocketTCPClient::on_pushButton_2_clicked()
41. {
42.     //获取 TextEdit 控件中的内容
43.     QString sendMsg = ui->m_sendTextEdit->toPlainText();
44.
45.     //转换成字符串发送
46.     char sendMsgChar[1024] = {0};
47.     strcpy_s(sendMsgChar, sendMsg.toStdString().c_str());
48.
49.     int sendRe = mp_clientSocket->write(sendMsgChar, strlen(sendMsgCh
    ar));
50.
51.     if(sendRe == -1)
52.     {
53.          QMessageBox::information(this, "QT 网络通信", "向服务端发送数据
    失败！");
54.          return;
55.     }
56. }
57.
58. void SocketTCPClient::ClientRecvData()
59. {
60.     //将接收内容存储到字符串中
61.     char recvMsg[1024] = {0};
62.     int recvRe = mp_clientSocket->read(recvMsg, 1024);
63.
64.     if(recvRe == -1)
65.     {
66.         QMessageBox::information(this, "QT 网络通信", "接收服务端数据失
    败！");
67.         return;
68.     }
69.
70.     QString showQstr = recvMsg;
71.     ui->m_recvTextEdit_2->setText(showQstr);
72. }
```

socketudpserver.cpp

```
1.  #include "SocketudpServer.h"
2.  #include "ui_SocketudpServer.h"
3.
4.  SocketudpServer::SocketudpServer(QWidget *parent) :
5.      QDialog(parent),
6.      ui(new Ui::SocketudpServer)
7.  {
8.      ui->setupUi(this);
9.
10.     setWindowTitle("QT UDP Sender");
11.
12.     //创建 UDP Socket
13.     udpSocket = new QUdpSocket(this);
14.
15.     //发送数据的端口
16.     QString portStr = portLineEdit->text();
17.     port = portStr.toInt();
18.
19.     //接收数据的端口
20.     portStr = portRecvLineEdit->text();
21.     recvPort = portStr.toInt();
22.     bool bindResult = udpSocket->bind(recvPort);    //接收数据时，需要
    将 SOCKET 与接收端口绑定在一起
23.     if(!bindResult)
24.     {
25.         QMessageBox::information(this, "消息提示", "绑定失败！");
26.         return;
27.     }
28.
29.     //connect slot
30.     connect(sendButton, SIGNAL(clicked()), this, SLOT(sendData()));
       //绑定发送
31.     connect(udpSocket, SIGNAL(readyRead()), this, SLOT(recvData()));
       //绑定接收
32.
33.
34. }
35.
36. SocketudpServer::~SocketudpServer()
37. {
38.     delete ui;
39. }
```

```
40.
41. void SocketudpServer::sendData()
42. {
43.     QString sendStr = sendLineEdit->text();
44.     if(sendStr.isEmpty())
45.     {
46.         QMessageBox::information(this, "消息提示", "输入数据为空，请重新输入...");
47.         return;
48.     }
49.
50.     //QHostAddress::Broadcast,指定向广播地址发送
51.     int length = udpSocket->writeDatagram(sendStr.toStdString().c_str(), QHostAddress::Broadcast, port);        //向指定端口发送数据
52.     if(length != sendStr.length())
53.     {
54.         QMessageBox::information(this, "消息提示", "发送失败！");
55.         return;
56.     }
57. }
58.
59. void SocketudpServer::recvData()
60. {
61.     while(udpSocket->hasPendingDatagrams())
62.     {
63.         QByteArray datagram;
64.         datagram.resize(udpSocket->pendingDatagramSize());
65.
66.         udpSocket->readDatagram(datagram.data(), datagram.size());
67.         QString msg = datagram.data();
68.         showRecvTextEdit->insertPlainText(msg + "\n");
69.         //showRecvTextEdit->setText(msg);
70.     }
71. }
```

socketudpclient.cpp

```
#include "Socketudpclient.h"

Socketudpclient::Socketudpclient(QWidget *parent)
    : QDialog(parent)
{
    setWindowTitle("QT UDP Receiver");

    //创建 UDP Socket
    udpSocket = new QUdpSocket(this);

    //发送数据的端口
    QString portStr = portLineEdit->text();
    port = portStr.toInt();

    //接收数据的端口
    portStr = portRecvLineEdit->text();
    recvPort = portStr.toInt();
    bool bindResult = udpSocket->bind(recvPort);    //接收数据时，需要将 SOCKET 与接收端口绑定在一起
    if(!bindResult)
    {
        QMessageBox::information(this, "消息提示", "绑定失败！");
        return;
    }

    //connect slot
    connect(sendButton, SIGNAL(clicked()), this, SLOT(sendData()));//绑定发送
    connect(udpSocket, SIGNAL(readyRead()), this, SLOT(recvData()));//绑定接收
}

Socketudpclient::~Socketudpclient()
{

}

void Socketudpclient::sendData()
{
    QString sendStr = sendLineEdit->text();
    if(sendStr.isEmpty())
    {
```

```
40.         QMessageBox::information(this, "消息提示", "输入数据为空，请重新输入...");
41.         return;
42.     }
43.
44.     //QHostAddress::Broadcast,指定向广播地址发送
45.     int length = udpSocket->writeDatagram(sendStr.toStdString().c_str(), QHostAddress::Broadcast, port);        //向指定端口发送数据
46.     if(length != sendStr.length())
47.     {
48.         QMessageBox::information(this, "消息提示", "发送失败！");
49.         return;
50.     }
51. }
52.
53. void Socketudpclient::recvData()
54. {
55.     while(udpSocket->hasPendingDatagrams())
56.     {
57.         QByteArray datagram;
58.         datagram.resize(udpSocket->pendingDatagramSize());
59.
60.         udpSocket->readDatagram(datagram.data(), datagram.size());
61.         QString msg = datagram.data();
62.         showRecvTextEdit->insertPlainText(msg+"\n");
63.     }
64. }
```