计算机网络上机作业

作业 1：编写简单的客户-服务器程序

实验报告

October 26, 2019

学号：1711425

姓名：曹元议

专业：计算机科学与技术

Contents

1 实验要求

实现一个 UDP 客户端和服务器，客户和服务器之间使用数据报方式传送信息，服务器收到来自客户端的请求之后获取本地当前时间日期返回给客户端。

2 开发环境

1. Windows 10 64 位专业版
2. Visual Studio 2015

3 界面设计

控件的排列方式主要是为了看起来协调，没有其他目的。因为一般都习惯把输入文本框放在窗口上方，按钮放在下方。无论是服务器还是客户端都有一个工作日志框，下方都有一个“清除日志”的按钮和一个“命令列表”的按钮。点击“清除日志”以后工作日志框的内容将会被清空。点击“命令列表”可以看到服务器端所支持的命令。

Figure 1: 命令列表框

上图是服务器对命令的处理，下文不再赘述。

3.1 服务器

以下是服务器的界面：

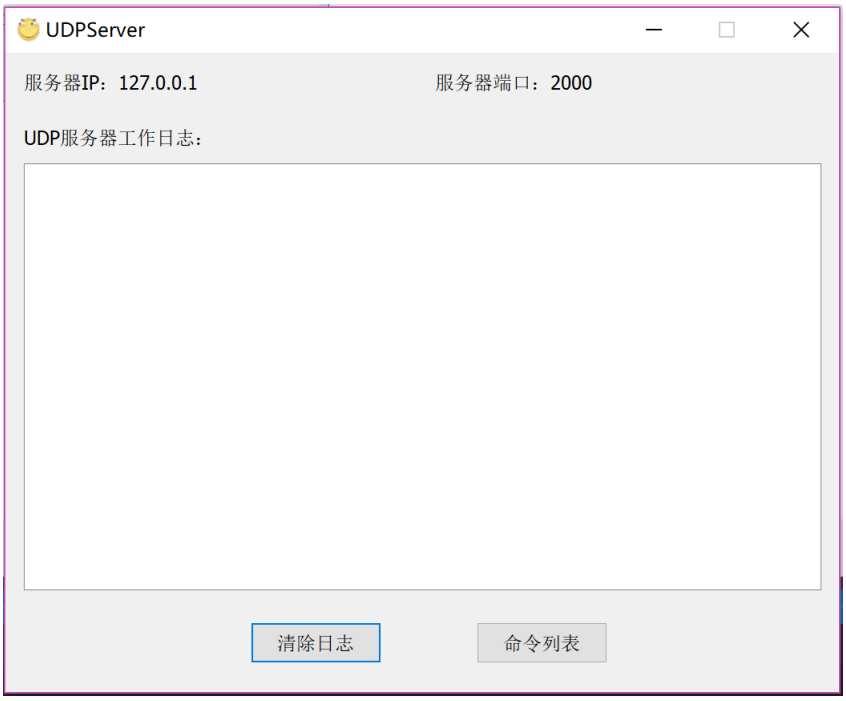

Figure 2: 服务器界面

服务器的界面设计比较简单，除了上面列出了服务器 IP 和服务器端口号，下面有一个服务器工作日志框，最底下有一个“清除日志”和“命令列表”的按钮之外就没有其他控件了。如果服务器收到客户端的请求，则会作出响应并在工作日志框打印一条日志。

服务器的工作日志格式为：

< 系统日期和时间 >，收到 IP=< 客户端 IP>，Port=< 客户端所在端口 >，请求：< 客户端请求文本 >，响应：< 服务器响应内容 >

3.2 客户端

以下是客户端的界面：

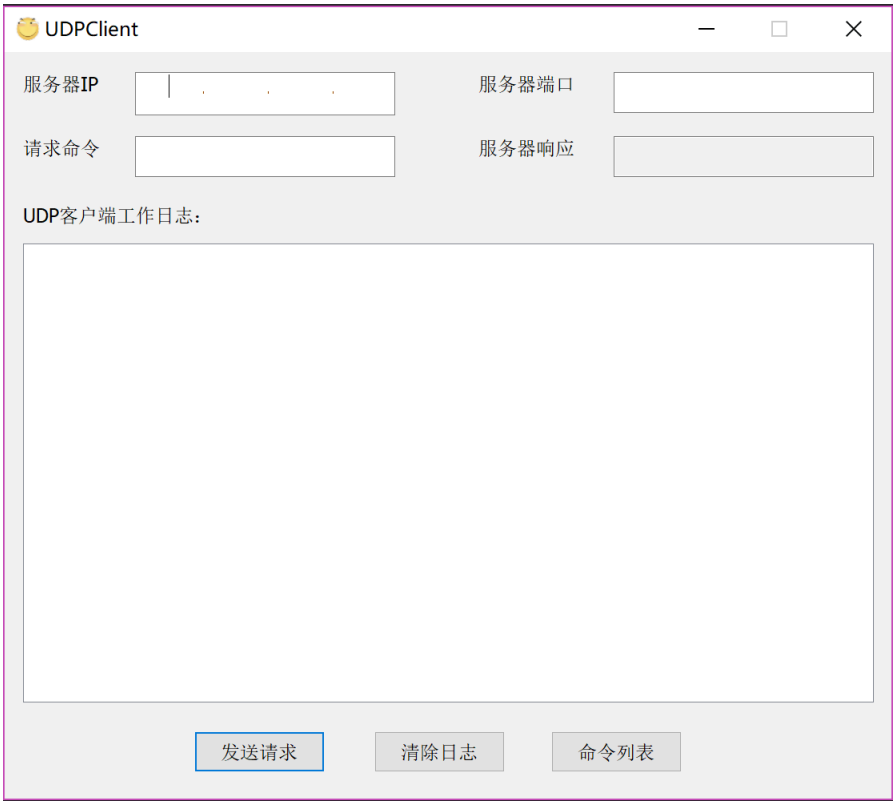

Figure 3: 客户端界面

在以上界面有四个输入文本框，前三个分别用来输入服务器 IP、服务器端口和客户端请求命令，最后一个不可编辑的文本框用来显示最近一个来自服务器的响应内容（如果对应的服务器端不响应则该框内容设为空）。点击“发送请求”即可向服务器端发送请求（前提是前三个文本框的内容都不为空）。发送请求以后会在工作日志框打印一条日志，表明是否发送成功或接收成功。

客户端的工作日志格式为：

- 成功收到服务器响应的格式为：< 系统日期和时间 >，向服务器 IP=< 服务器 IP>，服务器 Port=< 服务器端口 >，发送请求：< 客户端请求文本 >，服务器响应：< 服务器响应文本 >
- 在超时以内未收到服务器响应：< 系统日期和时间 >，向服务器 IP=< 服务器 IP>，服务器 Port=< 服务器端口 >，发送请求：< 客户端请求文本 >，发送失败，请求超时
- 客户端发送或接收错误：< 系统日期和时间 >，向服务器 IP=< 服务器 IP>，服务器 Port=< 服务器端口 >，发送请求：< 客户端请求文本 >，发送失败，错误码：< 该错误所对应的错误码 >

4 实验原理

UDP 和 TCP 都是传输层的协议。和 TCP 不同，UDP 是无连接的（传输数据前双方都不需建立任何连接）、不可靠的（数据包可能出现丢失、重复、乱序），并且也不提供流量控制、拥塞控制、时延和带宽保证等服务。并且，UDP 是以数据报的方式传输数据的，而 TCP 是以字节流的方式。

C/C++ 语言的实现有很多种。其中，`CAsyncSocket`类是 MFC 的异步非阻塞套接字类，由`CObject`类派生而来，对 Windows Sockets API 封装程度较低。由于封装程度低，因此在编程时可以看到更多的底层细节。

`CAsyncSocket`类主要的函数及其功能列举如下：

函数	功能
`Create()`	创建和初始化一个套接字
`Send()` 或`SendTo()`	发送数据
`Receive()` 或`ReceiveFrom()`	接收数据
`Connect()`	请求连接服务器
`Listen()`	侦听连接请求
`Accept()`	接收连接请求
`Bind()`	将 IP 地址、端口号与套接字绑定
`Close()`	关闭套接字

Table 1: CAsyncSocket 类的主要的函数及其功能

如上所述，由于 UDP 无需建立连接，所以上述的函数只需使用`Create()`、`SendTo()`、`ReceiveFrom()`、`Bind()`、`Close()`。使用`SendTo()`和`ReceiveFrom()`而不使用`Send()`和`Receive()`是因为 UDP 不建立连接，发送和接收数据都是点对点的，必须要指定对方的 IP 地址和端口号，所以使用指定对方 IP 地址和端口号的`SendTo()`和`ReceiveFrom()`；而 TCP 需要建立连接，对方的 IP 地址和端口号已经在建立连接（`Connect()`）的时候就已经指定好了，所以使用不指定对方 IP 地址和端口号的`Send()`和`Receive()`。只有服务器端才需要使用`Bind()`，因为服务器的 IP 和端口号一般是固定的。

5 实现思路和代码解释

本节仅展示主要的涉及实验原理的代码，其余代码在项目文件中。

使用`CAsyncSocket`类实现 UDP 客户端和服务器通信的流程图如下，下述的代码实现也是基于这个流程：

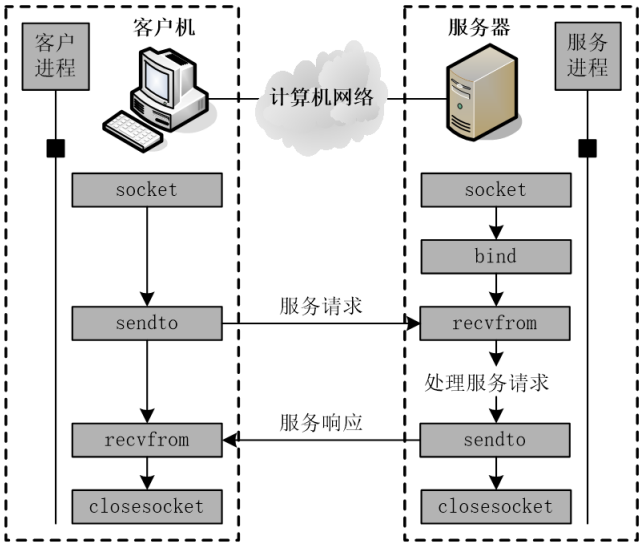

Figure 4: 流程图

上图可概括为：客户端进程在向服务器请求之前先要创建套接字（Create()），然后使用SendTo()函数把客户端请求文本发送到指定的服务器进程（IP 和端口号指定），然后使用ReceiveFrom()函数接收服务器端的响应文本（也会得到对应的服务器 IP 和端口号），接收完之后应使用Close()函数关闭套接字。由于服务器端进程是总在一个固定的 IP 地址和端口号在线的，所以必须是服务器进程创建之后不久服务器端就要创建套接字，另外还应使用Bind()函数把服务器端进程的 IP、端口号与套接字绑定，然后使用ReceiveFrom()函数接收客户端的请求文本（也会得到对应的客户端 IP 和端口号），然后服务器还应对客户端的请求文本进行处理（什么样的请求对应什么样的响应），然后再使用SendTo()函数向刚刚发送请求的客户端发送响应文本，最后服务器进程在退出的时候应关闭套接字。

5.1　服务器端的实现

服务器的源代码和项目文件在文件夹UDPServer中。

首先从CAsyncSocket类派生出一个子类CUDPSocket专门用于通信，并重写子类的虚函数OnReceive()。然后在服务器对话框类CUDPServerDlg中增加一个新的CUDPSocket的指针成员变量m_pSocket，表明CUDPSocket类的对象将在堆上创建。这里选择在CUDPSocket的构造函数创建新的对象，在析构函数中销毁该对象。

由于服务器是总在线的，并且其 IP 和端口号一般是固定的，要保证服务器进程刚创建不久就要创建套接字，在进程退出时关闭套接字，因此选择在对话框初始化的时候就创建套接字并将服务器 IP 地址和端口号与套接字绑定，对话框对象被销毁时再关闭套接字，以营造服务器总是在线并在同一个地方守候的样子。

以下是对CUDPServerDlg::OnInitDialog()函数的修改：

```
BOOL CUDPServerDlg::OnInitDialog()
{
  CDialogEx::OnInitDialog();

  // 将“关于...”菜单项添加到系统菜单中。

......

  // 设置此对话框的图标。  当应用程序主窗口不是对话框时，框架将自动
  //  执行此操作
  SetIcon(m_hIcon, TRUE);      // 设置大图标
  SetIcon(m_hIcon, FALSE);     // 设置小图标

  // TODO: 在此添加额外的初始化代码
  //m_pSocket->m_pDlg = this;
  //m_loglist.SetHorizontalExtent(1000);
  //CRect rectTemp;
  //m_loglist.GetWindowRect(rectTemp);
  CString ip = L"127.0.0.1";
  int port = 2000;
  CString temp1, temp2, temp3;
  temp3.Format(L"%d", port);
  //这四句的操作是为了在修改服务器IP和端口号的同时，在对话框上也能得到更新
  ((CStatic*)GetDlgItem(IDC_IP))->GetWindowTextW(temp1);
  ((CStatic*)GetDlgItem(IDC_PORT))->GetWindowTextW(temp2);
  ((CStatic*)GetDlgItem(IDC_IP))->SetWindowTextW(temp1 + ip);
  ((CStatic*)GetDlgItem(IDC_PORT))->SetWindowTextW(temp2 + temp3);
  //创建套接字
  if (!m_pSocket->Create(port, SOCK_DGRAM, FD_READ, ip))
  {
    MessageBox(L"创建套接字错误！", L"UDPServer", MB_OK | MB_ICONERROR);
   //错误消息框
    return FALSE;
  }
  m_pSocket->Bind(port, ip); //将IP地址、端口号与套接字绑定
  //服务器应该总在线并在固定的ip和端口等候

  return TRUE;  // 除非将焦点设置到控件，否则返回 TRUE
}
```

其中，函数调用Create(port, SOCK_DGRAM, FD_READ, ip)的第一个参数是服

务器的端口号。第二个参数SOCK_DGRAM表示创建数据报套接字，因为 UDP 是以数据报的方式传送数据的。第三个参数FD_READ表示一个事件，会触发对CUDPSocket类的虚函数OnReceive()的调用，这个函数表示服务器可以接收客户端请求并在收到请求之后应该做的动作（响应客户端请求，打印日志等），MFC 大量使用消息机制。第四个参数是服务器的 IP 地址。

CUDPSocket::OnReceive()函数如下：

```
void CUDPSocket::OnReceive(int nErrorCode)
{
  // TODO: 在此添加专用代码和/或调用基类
  TCHAR buff[4096];  //存放接收到的消息的缓冲区
  CString ip; //发送请求的客户端的IP
  UINT port;  //发送请求的客户端的端口号
  //接收消息
  int nRead = ReceiveFrom(buff, sizeof(buff), ip, port);

  if (!nRead)
  {
    //Close(); //什么都没收到则什么都不做
  }
  else if (nRead == SOCKET_ERROR)
  {
    //WSAEWOULDBLOCK：非阻塞模式下，请求的操作被阻塞，需等待再次调用
    if (GetLastError() != WSAEWOULDBLOCK) //错误处理
    {
      CString error;
      int errorcode = GetLastError();  //GetLastError()函数为获取错误码
      error.Format(L"Socket failed to receive: %d", errorcode);
      LPVOID lpMsgBuf;
      //获取具体错误信息
      FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        errorcode,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
        (LPTSTR)&lpMsgBuf,
        0,
        NULL
        );
      error = error + L"\n" + (LPCTSTR)lpMsgBuf;
```

```
36        AfxMessageBox(error); //错误提示框
37      }
38    }
39    else
40    {
41      //实际的字符数应是发送缓冲区空间的一半（一个字符占2个字节），nRead只表
      示收到消息的字节数，除以2才是真正的字符数
42      buff[nRead / 2] = _T('\0'); //terminate the string
43      //生成日志
44      CString receiveText;
45      receiveText.Format(L"%s", buff);
46      CString strTime;
47      CTime tm;
48      tm = CTime::GetCurrentTime(); //获取当前日期和时间
49      strTime = tm.Format("%Y-%m-%d %H:%M:%S"); //日志时间
50      CString temp;
51      temp.Format(strTime + L", 收到IP=" + ip + L", Port=%d, 请求：%s, 响应：
      ", port, receiveText);
52      CString response;
53      //CompareNoCase为不区分大小写的字符串比较
54      if (receiveText.CompareNoCase(L"date") == 0)
55      {
56        response = tm.Format("%Y-%m-%d"); //日期
57      }
58      else if (receiveText.CompareNoCase(L"time") == 0)
59      {
60        response = tm.Format("%H:%M:%S"); //时间
61      }
62      else
63      {
64        response = L"Bad Request"; //非法请求
65      }
66      temp += response;
67      CUDPServerDlg *pDlg = (CUDPServerDlg*)AfxGetApp()->GetMainWnd(); //获取
      当前对话框
68      pDlg->m_loglist.InsertString(pDlg->m_loglist.GetCount(), temp);  //在对
      话框中的工作日志框中插入此条日志
69      TCHAR *resbuf = response.GetBuffer(response.GetLength()); //产生长度为
      响应消息字符数的发送消息缓冲区
70      //向客户端发送服务器响应
71      if (SendTo(resbuf, response.GetLength() * 2, port, ip) == SOCKET_ERROR)
72      {
```

```
73        //发送的错误处理
74        if (GetLastError() != WSAEWOULDBLOCK)
75        {
76          CString error;
77          int errorcode = GetLastError();
78          error.Format(L"Socket failed to send: %d", errorcode);
79          LPVOID lpMsgBuf;
80          FormatMessage(
81            FORMAT_MESSAGE_ALLOCATE_BUFFER |
82            FORMAT_MESSAGE_FROM_SYSTEM |
83            FORMAT_MESSAGE_IGNORE_INSERTS,
84            NULL,
85            errorcode,
86            MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
87            (LPTSTR)&lpMsgBuf,
88            0,
89            NULL
90            );
91          error = error + L"\n" + (LPCTSTR)lpMsgBuf;
92          AfxMessageBox(error);  //错误提示框
93        }
94      }
95    }
96    CAsyncSocket::OnReceive(nErrorCode);
97  }
```

其中，ReceiveFrom(buff,sizeof(buff), ip, port)的第一个参数是接收消息缓冲区，第二个参数是缓冲区的大小。后两个参数是引用，该函数返回以后会得到客户端的 IP 地址和端口号。该函数的返回值nRead为收到消息的字节数，若为-1则接收错误，需要作相应的错误处理（打印错误日志，弹出错误提示框）。

SendTo(resbuf, response.GetLength()*2, port, ip)的第一个参数是发送消息缓冲区，第二个参数是缓冲区的大小。后两个参数则是刚刚发送请求的客户端的端口号和 IP 地址。

这里需要解释的是TCHAR和CString中的字符都是 Unicode 字符，一个字符占 2 个字节，因此缓冲区实际的大小sizeof(buff)是缓冲区长度的 2 倍。nRead是接收到的消息所占用的字节数，除以 2 才是真正的有效字符（不包括'\0'）数。

其余部分函数展示如下：

```
1  //CUDPServerDlg类构造函数
2  CUDPServerDlg::CUDPServerDlg(CWnd* pParent /*=NULL*/)
3    : CDialogEx(IDD_UDPSERVER_DIALOG, pParent)
```

```
4  {
5    m_hIcon = AfxGetApp()->LoadIcon(IDI_SMILE);
6    m_pSocket = new CUDPSocket; //在堆上建立套接字对象
7  }
8
9  //CUDPServerDlg类析构函数
10 CUDPServerDlg::~CUDPServerDlg()
11 {
12   delete m_pSocket;  //销毁套接字对象
13 }
14
15 //点击对话框顶部的关闭按钮关闭对话框时被调用的函数
16 void CUDPServerDlg::OnClose()
17 {
18   // TODO: 在此添加消息处理程序代码和/或调用默认值
19   m_pSocket->Close(); //此时应关闭套接字
20   CDialogEx::OnClose();
21 }
22
23 //服务器的CUDPSocket类的析构函数
24 CUDPSocket::~CUDPSocket()
25 {
26     //如果在销毁套接字对象前未关闭套接字，先关闭套接字
27   if (m_hSocket != INVALID_SOCKET)
28   {
29     Close();
30   }
31 }
```

5.2　客户端的实现

服务器的源代码和项目文件在文件夹UDPClient中。

比服务器稍复杂，下面部分代码由于与服务器端的类似，故不重复解释。当然，这里也要从CAsyncSocket类派生出一个专门用于通信的子类CUDPSocket，但是这个类由于在不同的工程文件下因此跟服务器的不一样。也要重写虚函数OnReceive()。然后在客户端对话框类CUDPClientDlg中也需要增加一个新的CUDPSocket的指针成员变量m_pSocket,表明CUDPSocket类的对象将在堆上创建。同样,这里也选择在CUDPSocket的构造函数创建新的对象，在析构函数中销毁该对象。

由于客户端不用总在线，而且 IP 或端口号可以不固定。因此选择在单击“发送请求”按钮之后创建套接字，也不需要做绑定的操作，收到服务器响应或者遇到错误之后

就关闭套接字。

以下是对CUDPClientDlg::OnBnClickedOk()函数的定义（在单击“发送请求”之后触发对此函数的调用）：

```
void CUDPClientDlg::OnBnClickedOk()
{
  // TODO: 在此添加控件通知处理程序代码
  CString portstr;
  m_request.GetWindowTextW(msg);
  m_port.GetWindowTextW(portstr);
  int msglength = msg.GetLength();
  //判断输入框是否为空
  if (!msglength)
  {
    MessageBox(L"命令不能为空！", L"UDPClient", MB_OK | MB_ICONWARNING);
    return;
  }
  if (m_ip.IsBlank())
  {
    MessageBox(L"服务器IP不能为空！", L"UDPClient", MB_OK | MB_ICONWARNING)
   ;
    return;
  }
  if (!portstr.GetLength())
  {
    MessageBox(L"服务器端口号不能为空！", L"UDPClient", MB_OK |
   MB_ICONWARNING);
    return;
  }
  BYTE nf1, nf2, nf3, nf4;
  m_ip.GetAddress(nf1, nf2, nf3, nf4);
  ip.Format(L"%d.%d.%d.%d", nf1, nf2, nf3, nf4);
  //UpdateData(true);
  //创建套接字
  //由操作系统随机分配空闲端口
  //IP为NULL表示本地地址
  if (!m_pSocket->Create(0, SOCK_DGRAM, FD_READ))
  {
    MessageBox(L"创建套接字错误！", L"UDPClient", MB_OK | MB_ICONERROR);
    return;
  }
  UpdateData(true);
  //输入的请求转成TCHAR缓冲区
```

```
38  TCHAR *msgbuf = msg.GetBuffer(msg.GetLength());
39  //Unicode模式下一个字符占两个字节
40  //向服务器发送请求
41  if (m_pSocket->SendTo(msgbuf, msglength * 2, m_portnum, ip) ==
      SOCKET_ERROR)
42  {
43    //WSAEWOULDBLOCK：非阻塞模式下，请求的操作被阻塞，需等待再次调用
44    if (GetLastError() != WSAEWOULDBLOCK)
45    {
46      //生成发送错误日志
47      CString error;
48      int errorcode = GetLastError(); //获取错误码
49      error.Format(L"Socket failed to send: %d", errorcode);
50      LPVOID lpMsgBuf;
51      //错误详细信息
52      FormatMessage(
53        FORMAT_MESSAGE_ALLOCATE_BUFFER |
54        FORMAT_MESSAGE_FROM_SYSTEM |
55        FORMAT_MESSAGE_IGNORE_INSERTS,
56        NULL,
57        errorcode,
58        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
59        (LPTSTR)&lpMsgBuf,
60        0,
61        NULL
62        );
63      error = error + L"\n" + (LPCTSTR)lpMsgBuf;
64      CString strTime;
65      CTime tm;
66      tm = CTime::GetCurrentTime();  //获取当前系统时间
67      strTime = tm.Format("%Y-%m-%d %H:%M:%S"); //日志项的日期和时间
68      CString temp;
69      temp.Format(strTime + L"，向服务器IP=" + ip + L"，服务器Port=%d，发送
      请求：%s，发送失败，错误码：%d", m_portnum, msg, errorcode);
70      m_loglist.InsertString(m_loglist.GetCount(), temp);  //往工作日志框插
      入错误日志项
71      m_response.SetWindowTextW(L""); //未收到响应应清空服务器响应框的内容
72      AfxMessageBox(error); //错误提示框
73      m_pSocket->Close();  //遇到错误就关闭套接字
74    }
75  }
76  else
```

```
77    {
78      SetTimer(1, 6000, NULL);  //发送成功就设置定时器，定时6s
79    }
80    //m_pSocket->Close(); //这里还不能关闭套接字，因为要等待服务器响应
81    //CDialogEx::OnOK();
82  }
```

其中Create()函数的第一个参数为 0 表示由操作系统随机分配一个空闲端口。IP 地址参数为空表示 IP 地址为本地地址，可以是127.0.0.1，也可以是本机的其他 IP。

以下是对CUDPSocket::OnReceive()的定义：

```
1   void CUDPSocket::OnReceive(int nErrorCode)
2   {
3     // TODO: 在此添加专用代码和/或调用基类
4     //把主窗口指针放在前面是为了避免这个操作改写了GetLastError()的值
5     CUDPClientDlg *pDlg = (CUDPClientDlg*)AfxGetApp()->GetMainWnd();
6     TCHAR buff[4096];
7     CString ip;
8     UINT port;
9     int nRead = ReceiveFrom(buff, sizeof(buff), ip, port);
10
11    if (!nRead)
12    {
13      Close();
14    }
15    else if (nRead == SOCKET_ERROR)
16    {
17      //WSAEWOULDBLOCK: 非阻塞模式下，请求的操作被阻塞，需等待再次调用
18      if (GetLastError() != WSAEWOULDBLOCK)
19      {
20          //WSAECONNRESET: 表示连接被reset，UDP是不需要建立连接的，这里直接忽略这个错误
21        //事实上这是Windows socket的一个bug，当UDP套接字在某次发送后收到一个不可到达的ICMP包时
22        //这个错误将在下一次调用OnReceive()后返回，会造成假读的现象，忽略它就可以避免对一个网络中不可达的主机报这个错
23        if (GetLastError() == WSAECONNRESET)
24        {
25          return;
26        }
27        pDlg->KillTimer(1);   //关闭定时器，定时器号为1
28        CString error;
29        int errorcode = GetLastError();
```

```
30        error.Format(L"Socket failed to receive: %d", errorcode);
31        LPVOID lpMsgBuf;
32        //错误详细信息
33        FormatMessage(
34          FORMAT_MESSAGE_ALLOCATE_BUFFER |
35          FORMAT_MESSAGE_FROM_SYSTEM |
36          FORMAT_MESSAGE_IGNORE_INSERTS,
37          NULL,
38          errorcode,
39          MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
40          (LPTSTR)&lpMsgBuf,
41          0,
42          NULL
43          );
44        error = error + L"\n" + (LPCTSTR)lpMsgBuf;
45        //生成接收错误日志
46        CString request;
47        pDlg->m_request.GetWindowTextW(request);
48        CString strTime;
49        CTime tm;
50        tm = CTime::GetCurrentTime();
51        strTime = tm.Format("%Y-%m-%d %H:%M:%S"); //日志日期时间
52        CString temp, errorip;
53        pDlg->m_ip.GetWindowTextW(errorip);
54        int errorport = pDlg->m_portnum;
55        temp.Format(strTime + L", 向服务器IP=" + errorip + L", 服务器Port=%d
      , 发送请求：%s，发送失败，错误码：%d", errorport, request, errorcode);
56        pDlg->m_loglist.InsertString(pDlg->m_loglist.GetCount(), temp);  //往
      工作日志框加入日志项
57        pDlg->m_response.SetWindowTextW(L"");  //未收到响应应清空服务器响应框
      的内容
58        AfxMessageBox(error);  //错误提示
59        Close(); //遇到错误就关闭套接字
60      }
61    }
62    else
63    {
64      pDlg->KillTimer(1);  //关闭定时器
65      CString request;
66      pDlg->m_request.GetWindowTextW(request);
67      //实际的字符数应是发送缓冲区空间的一半（一个字符占2个字节）
68      buff[nRead / 2] = _T('\0'); //terminate the string
```

```
69      //生成成功响应的日志
70      CString receiveText;
71      receiveText.Format(L"%s", buff);
72      pDlg->m_response.SetWindowTextW(receiveText); //更新服务器响应框
73      CString strTime;
74      CTime tm;
75      tm = CTime::GetCurrentTime();
76      strTime = tm.Format("%Y-%m-%d %H:%M:%S"); //日志日期时间
77      CString temp;
78      temp.Format(strTime + L", 向服务器IP=" + ip + L", 服务器Port=%d, 发送请
        求：%s, 服务器响应：%s", port, request, receiveText);
79      pDlg->m_loglist.InsertString(pDlg->m_loglist.GetCount(), temp);  //往工
        作日志框加入日志项
80      MessageBox(NULL, receiveText, L"服务器响应", MB_OK | MB_ICONINFORMATION
        ); //服务器响应提示
81      Close();  //服务器响应之后关闭套接字
82    }
83    CAsyncSocket::OnReceive(nErrorCode);
84  }
```

上述在错误处理中忽略了WSAECONNRESET错误，它表示连接被重置，由于 UDP 是不需要建立连接的，就直接忽略这个错误。否则当在客户端输入的 IP 是客户端 IP 所可达的 IP 但是没有相应的进程在这个输入端口上（例如把本文的服务器程序关闭了），则会收到一个表示对方主机不可达的 ICMP 包，在下次调用OnReceive()时会误认为这个 ICMP 包就是收到的对方的数据包从而读取这个数据包，返回这个错误。

SetTimer()函数是是设置定时器的函数。第一个参数是定时器的 ID，第二个参数是定时器的时间间隔（单位为毫秒），第三个函数是回调函数。这里为NULL，但是定时器的时间到了会发出WM_TIMER消息，然后会调用函数OnTimer()作为对定时器时间到了的处理（打印超时日志，显示错误提示框）。

需要解释一下定时器的作用。因为输入某些不在线的 IP 或端口之后，遇到的错误会是WSAEWOULDBLOCK，这严格来说不算是错误，直白的说，意思就是当前的操作无法立即完成，需要一直等到消息到来时再重新调用函数，然后再看看有没有这个错误，有了就重复刚才的等候。这样的话，程序就会看起来没有任何反应，就是因为要一直等待服务器的响应。由于CAsyncSocket是异步的不是同步的，因此窗口并没有因为没等到服务器响应而被冻结。

因此就要设计定时器，相当于是设计超时，到了定时器的时间还没有收到响应或者遇到错误就停止计时并提示请求超时，并打印请求超时的日志。如果在定时器的时间之前就收到了响应或错误则提前停止计时。

已到定时器的时间会发出消息WM_TIMER，所调用的函数CUDPClientDlg::OnTime

r()如下：

```
void CUDPClientDlg::OnTimer(UINT_PTR nIDEvent)
{
  // TODO: 在此添加消息处理程序代码和/或调用默认值
  KillTimer(1);  //关闭定时器
  CString strTime;
  CTime tm;
  tm = CTime::GetCurrentTime();
  strTime = tm.Format("%Y-%m-%d %H:%M:%S"); //日志项日期时间
  CString temp;
  temp.Format(strTime + L", 向服务器IP=" + ip + L", 服务器Port=%d, 发送请
    求：%s, 发送失败, 请求超时", m_portnum, msg); //如果在这6s内改变了输入框
    的值，应该取旧值，而不是取刚修改过的新值
  m_loglist.InsertString(m_loglist.GetCount(), temp); //往工作日志框插入日
    志项
  m_response.SetWindowTextW(L"");
  AfxMessageBox(L"请求超时！");
  m_pSocket->Close();  //关闭套接字
  CDialogEx::OnTimer(nIDEvent);
}
```

其他部分函数由于和服务器的一样因此不再展示。

5.2.1 多客户端的实现

未显式实现多客户端的设计。但是可以同时开启多个客户端进程。当然，由于建立客户端套接字时的端口号是随机分配的，因此无论在客户端的哪个进程的对话框，每次单击“发送请求”之后服务器接收到的客户端的端口号都是不同的。

6 程序演示

6.1 服务器正常响应的情况

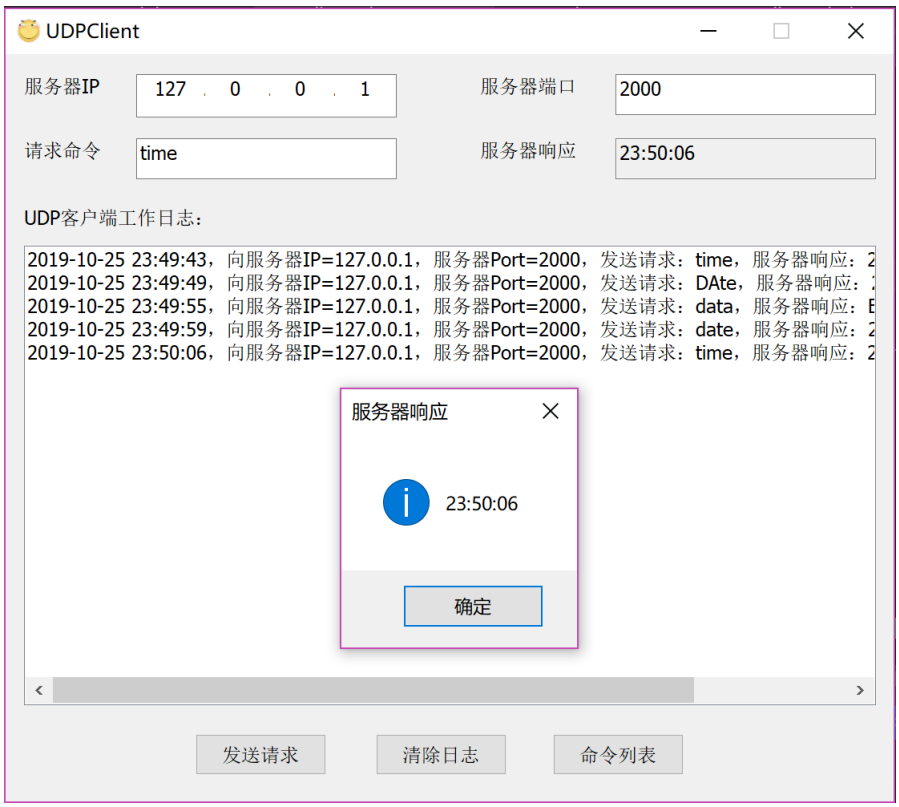

Figure 5: 客户端

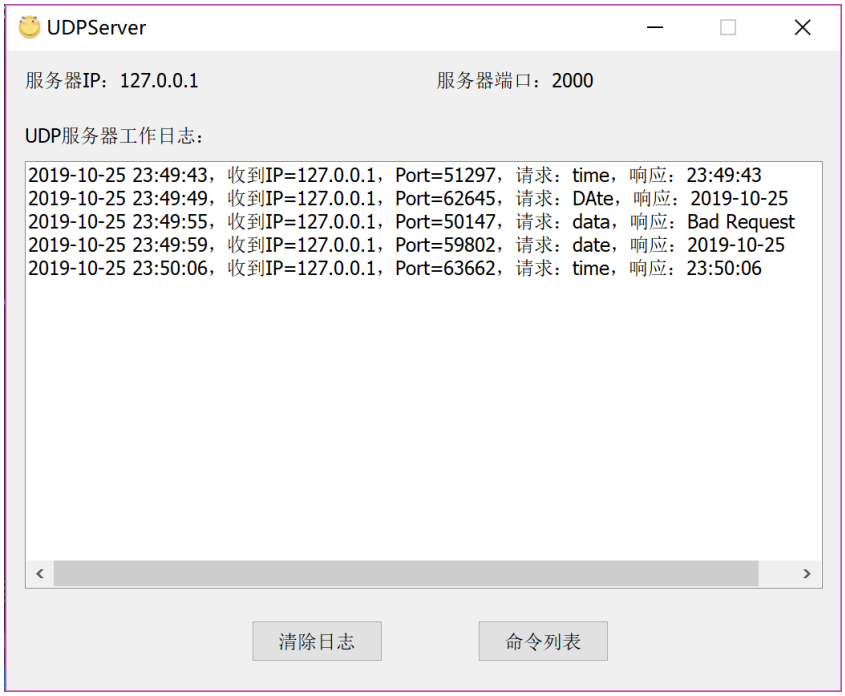

Figure 6: 服务器

利用 Wireshark + Npcap 可以捕捉本地回环接口的数据包。如果服务器成功响应则会看到发送和接收的数据包信息。

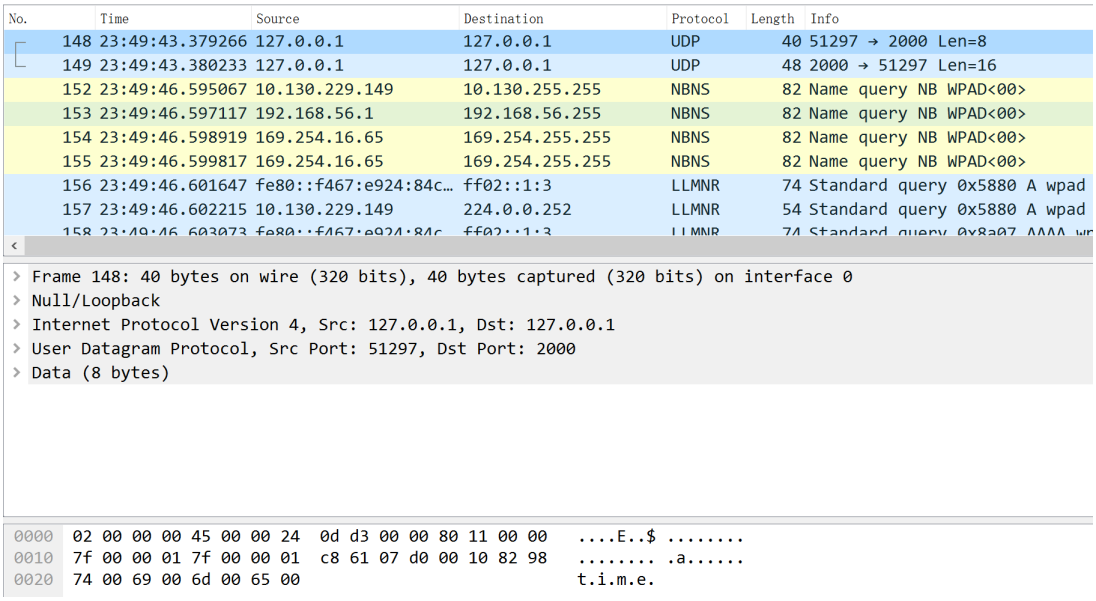

Figure 7: 客户端发送出去的数据包

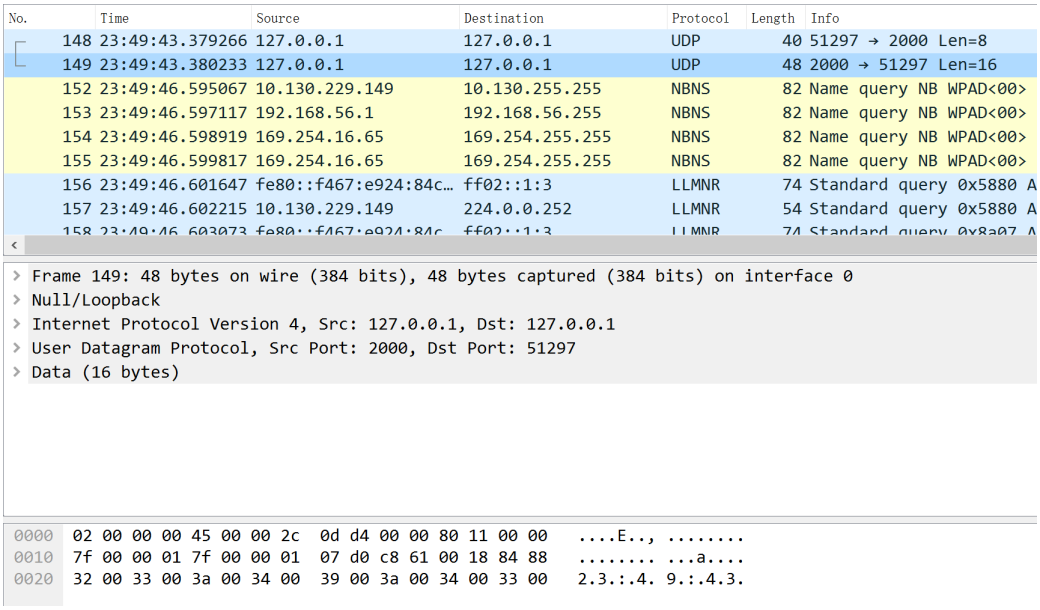

Figure 8: 客户端接收到的数据包

由上图也可以看出，传送的数据包中，一个字符占两个字节。

6.2 其他情况

本程序默认的 IP 为本地 IP`127.0.0.1`、端口为 2000，如果把服务器的对话框关闭，或者输入了不是 2000 的端口，或者在客户端输入了其他不是服务器的 IP（包括在 cmd 能 ping 通的 IP，或 ping 了之后请求超时的 IP），在这里定时器的时间为 6s，因此过了 6s 后会出现请求超时的错误：

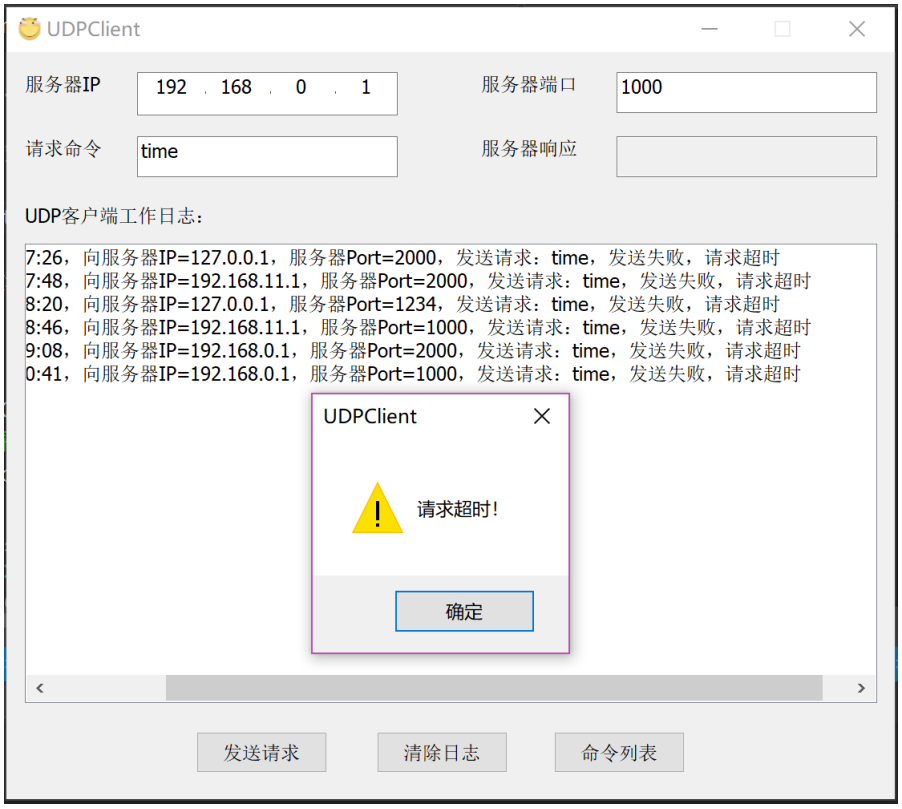

Figure 9: 超时错误

在 Wireshark 中查看捕获到的数据包，发现如果是向源 IP 地址可达的目的 IP 地址可达但是端口号没有相应进程的主机发送数据，则不仅会看到客户端发送的 UDP 数据包，还会看到客户端收到的在 5.2 节中提到的表示对方主机不可达的 ICMP 数据包：

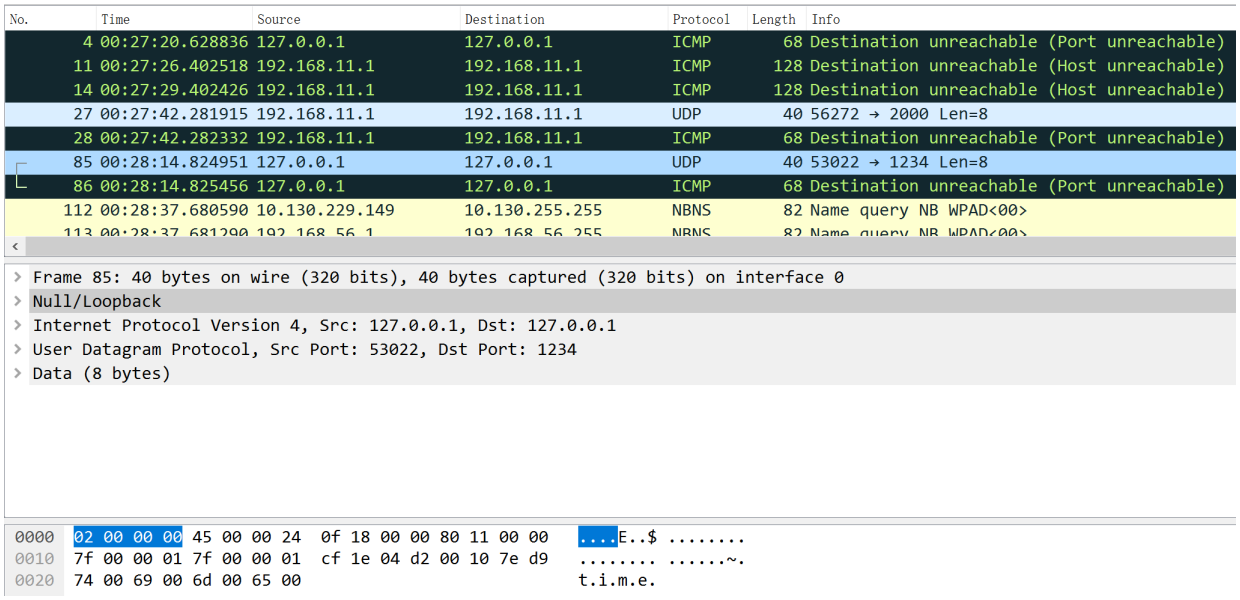

Figure 10: 客户端发送的 UDP 数据包

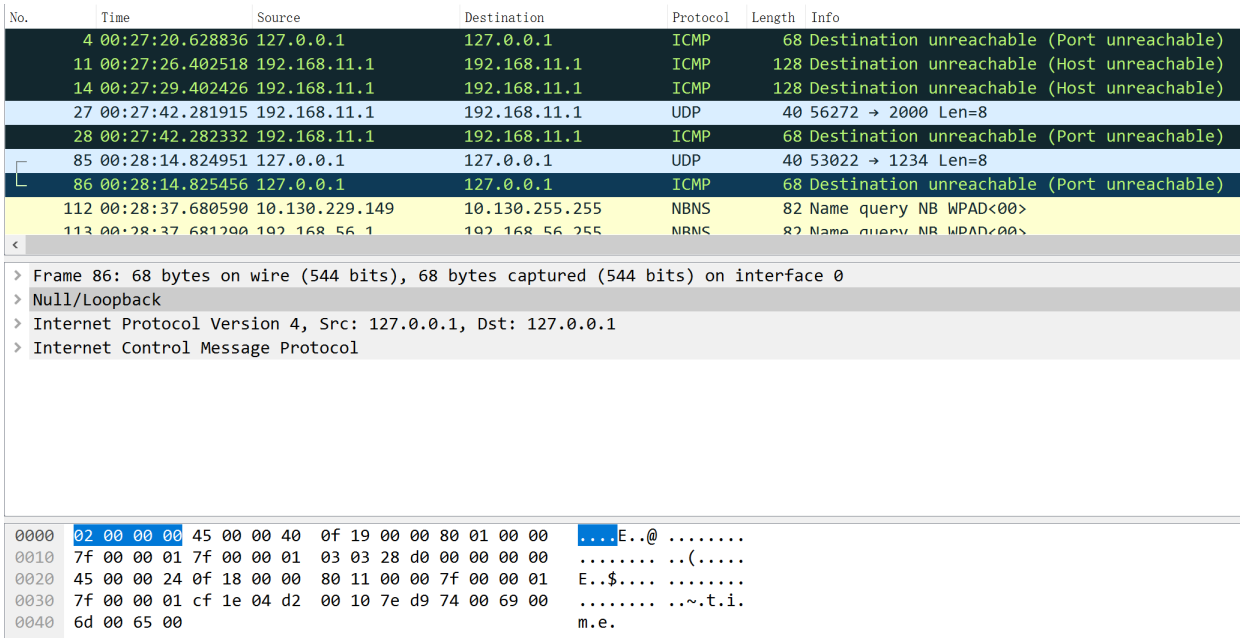

Figure 11: 客户端收到的不可达的 ICMP 数据包

程序中设定的日志日期和时间是定时器时间到了的时间，并非真正的发送时间。所以程序中看到的比 Wireshark 的晚了 6s。

由于现在使用的这台电脑的 IP 是`192.168.11.1`，也是本机 IP 可达的 IP，因此也可以看到`192.168.11.1`的 UDP 和 ICMP 数据包。

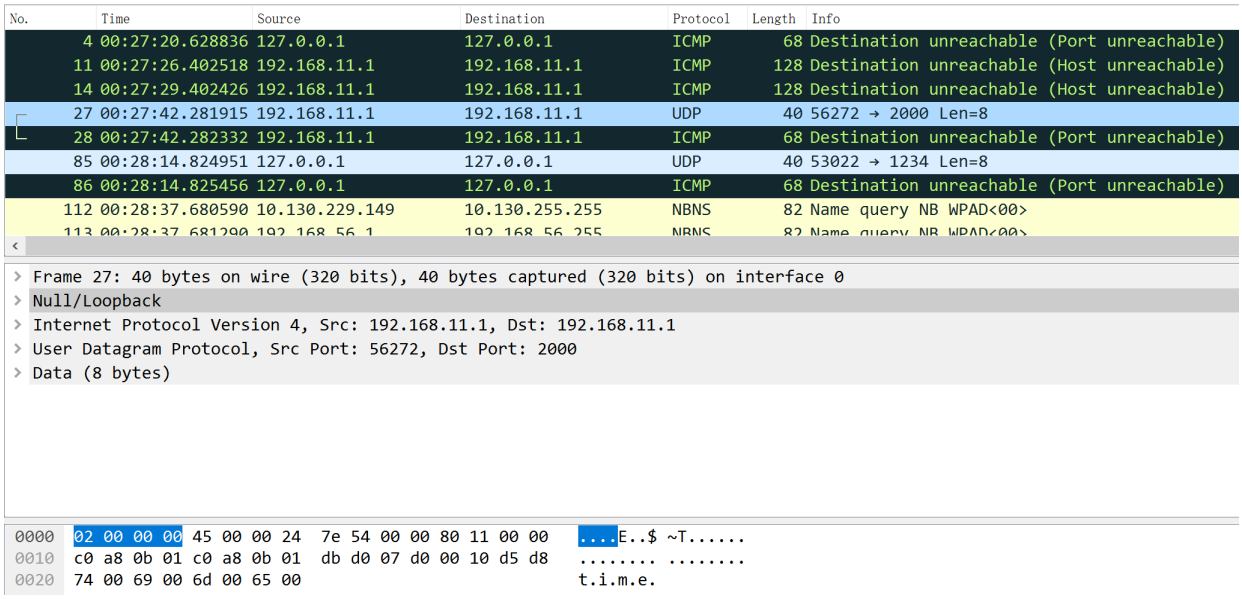

Figure 12: 客户端发送的 UDP 数据包

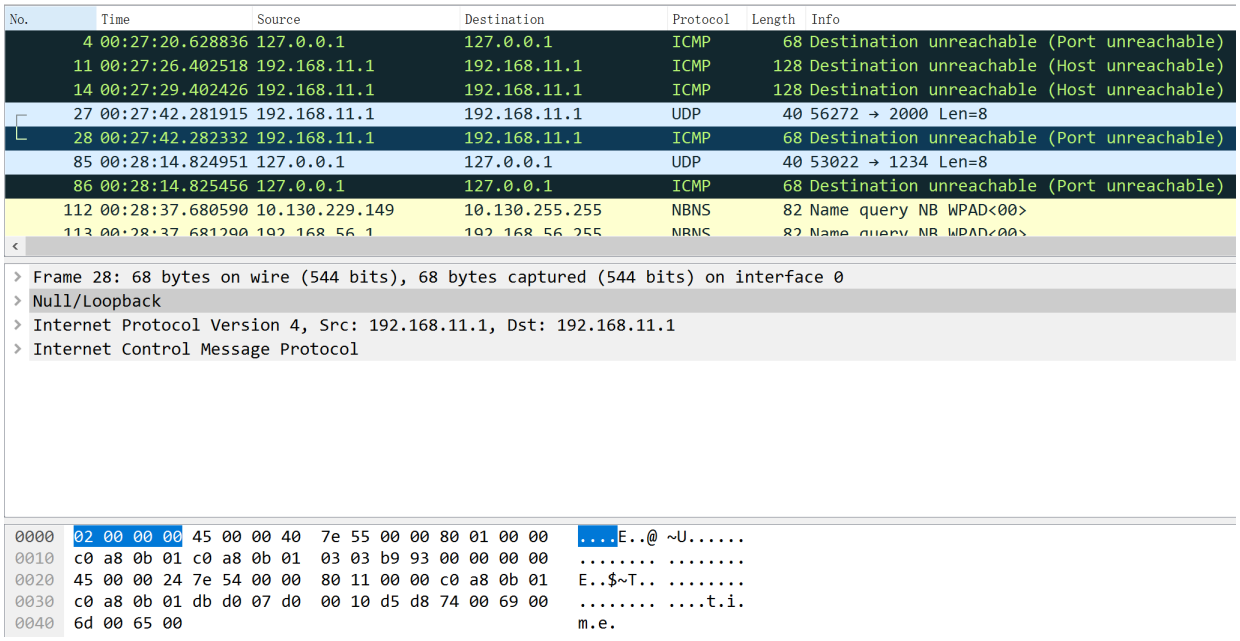

Figure 13: 客户端收到的不可达的 ICMP 数据包

输入某些 IP 还会报其他错误（在 cmd 中无法 ping 的 IP，而不是 ping 了之后请求超时的 IP），会显示错误码和详细错误信息：

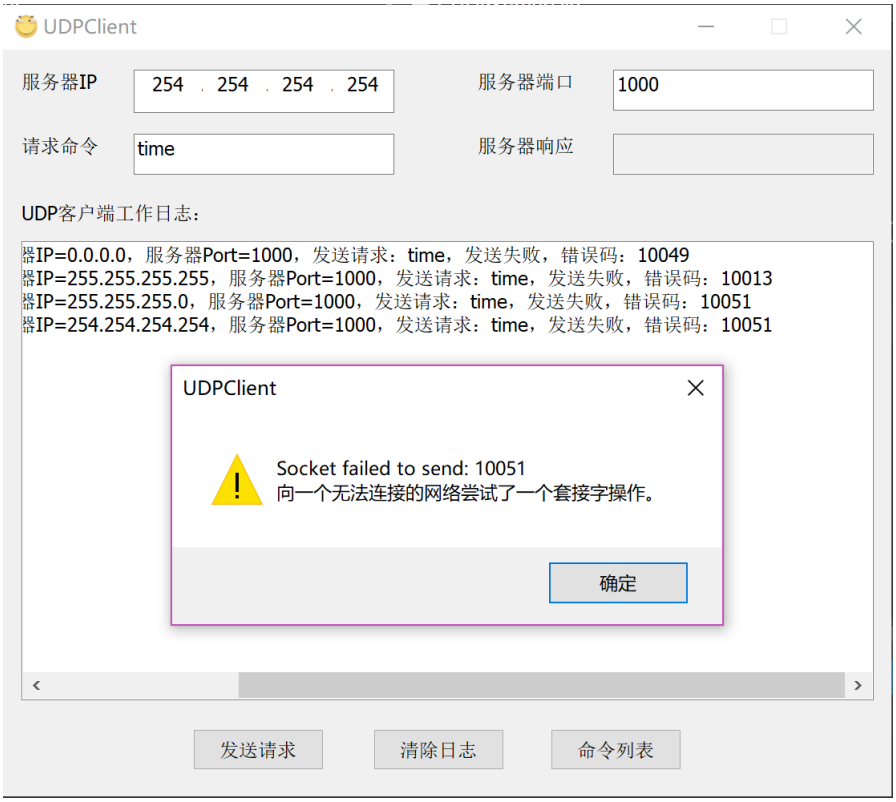

Figure 14: 无法连接的 IP

当然，在 Wireshark 上也看不到相关的包。

7 其他建议

建议及时在学院网站上发布作业要求，以让大家在做作业前心里有个数，什么时候提交，知道作业应该实现到什么程度，什么是加分项。谢谢配合！