计算机网络上机作业

作业 3：使用 UDP 实现可靠的文件传输

实验报告

December 20, 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

1. 下层使用 UDP 协议（即使用数据报套接字完成本次程序）；
2. 完成客户端和服务器端程序；
3. 实现可靠的文件传输：能可靠下载文件，能同时下载文件。

2 开发环境

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

3 界面设计

控件的排列方式也是为了看起来协调，没有其他目的。

3.1 客户端

左上角显示客户端当前的 IP 地址和端口号，右上角显示与服务器的连接状态（已连接、未连接）。中间是输入要连接的服务器 IP 地址和端口号，以及要从本地上传到服务器的文件、要从服务器下载的文件。左下角是客户端的日志，右下角是客户端要做的操作：与服务器建立连接、断开连接、上传文件、下载文件。

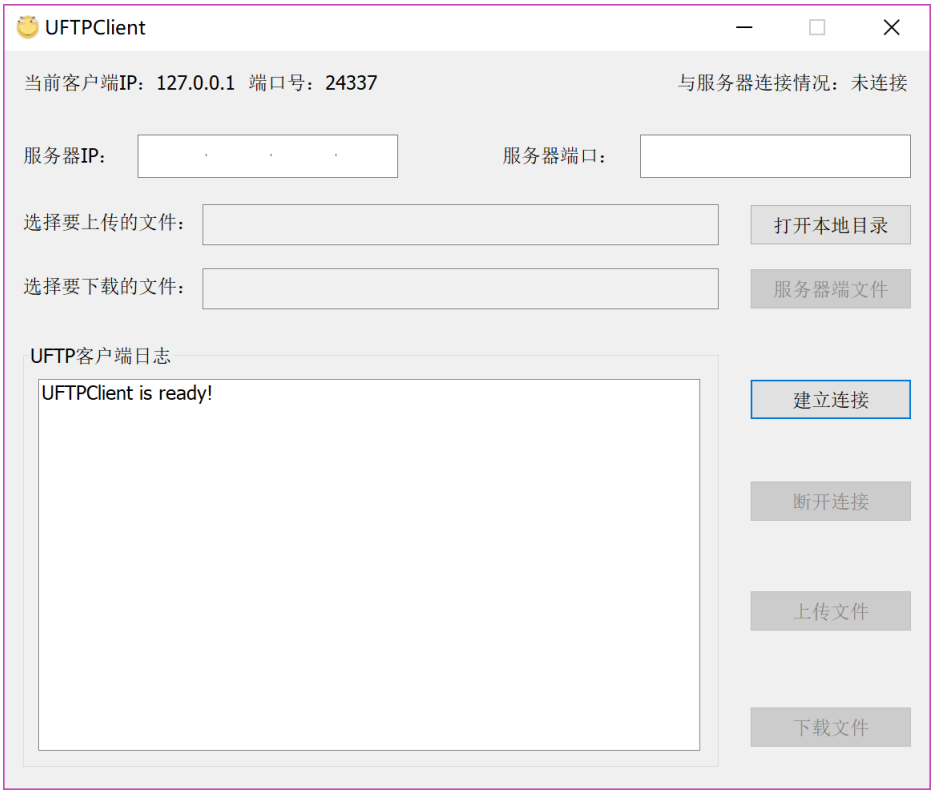

Figure 1: 客户端

3.2 服务器

服务器的界面设计较简单。左上角显示客户端当前的 IP 地址和端口号，右上角显示已经与服务器的建立连接的客户端数。中间是一个测试功能，是对服务器行为的选择，主要是可能造成不可靠的行为：丢包、延时，然后是输入丢包和延时行为的发生率和延迟时间。下方是服务器的日志。

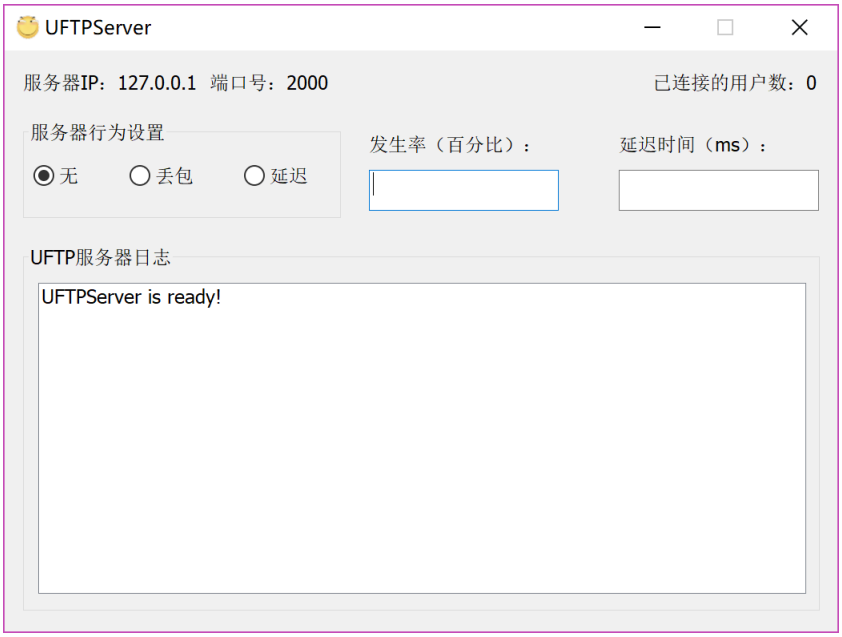

Figure 2: 客户端

4 设计思路

4.1 UFTP 协议

提到文件传输协议，不由得想起 FTP 协议，但是这是基于 TCP 的，可靠性由 TCP 来保证。在这里需要在不可靠的 UDP 上实现可靠文件传输，因此为下面要设计的协议取名为 UFTP。

协议的主要功能有：建立连接、断开连接、获取服务器文件列表、下载文件、上传文件，每一个功能都有其对应的报文格式。

4.2 与服务器建立连接、断开连接

这里默认服务器的 IP 地址为`127.0.0.1`，端口号为`2000`，在客户端必须要输入这样的 IP 地址和端口号并点击“建立连接”按钮才能成功和服务器端建立连接，成功建立连接之后点击“断开连接”即可关闭与服务器的连接。

与服务器建立连接、断开连接的报文格式如下：

opcode	ack
1 Byte	1 Byte

Figure 3: 与服务器建立连接、断开连接的报文格式

其中opcode为操作码，占 1 个字节，值为 1 表示建立连接，值为 2 表示断开连接；ack为请求/应答确认标记，占 1 个字节，值为 0 表示建立连接/断开连接的请求，值为 1 表示建立连接/断开连接的确认。

由于 TCP 协议的建立连接需要 3 次握手，关闭连接需要 4 次挥手，比较繁琐。为简化实现，这里的建立连接是 2 次握手，关闭连接是 2 次挥手。

建立连接的过程如下：客户端向服务器端按照上述报文格式发送建立连接请求（opcode=1,ack=0),服务器端收到该连接请求之后做出上述报文格式的响应（opcode=1，ack=1），客户端收到该响应之后进入已建立连接的状态，此时就可以请求获取服务器文件列表、请求下载文件、请求上传文件了，也可以请求断开连接。

关闭（断开）连接的过程如下：一方（可以是客户端或服务器端）向另一方按照上述报文格式发送断开连接请求（opcode=2，ack=0），另一方收到该断开连接请求之后做出上述报文格式的响应（opcode=2，ack=1），这一方收到该响应之后进入断开连接的状态，除了请求建立连接以外，请求获取服务器文件列表、请求下载文件、请求上传文件这些功能都被禁用。触发关闭连接的操作可以是客户端的“断开连接”的按钮被按下，关闭客户端或服务器的窗口。其中关闭服务器的窗口为服务器向客户端发起关闭连接的请求。

4.3 请求服务器端文件列表

请求服务器端文件列表的报文格式如下：

opcode	ack	len	filelistdata
1 Byte	1 Byte	4 Bytes	

Figure 4: 请求服务器端文件列表的报文格式

其中opcode字段值为 3；ack为 0 表示客户端向服务器端请求文件列表；ack为 1 表示服务器的响应（夹带客户端要请求的文件列表），此时len和filelistdata字段才变为可用。filelistdata是服务器端文件列表的数据，由客户端负责解析并显示，其长度是可变的，由服务器端文件数目和文件名长度决定。len记录filelistdata字段的长度，方便客户端解析filelistdata的数据，共占 4 个字节。在filelistdata中，每个文件名后都有一个换行符（包括结尾文件名），用来与下一个或上一个文件名分隔开来。

4.4 上传/下载文件请求和响应

上传/下载文件请求和响应的报文格式如下：

opcode	ack	filenamelen	filename
1 Byte	1 Byte	4 Bytes	

Figure 5: 上传/下载文件请求和响应的报文格式

其中opcode字段值为 4、ack字段值为 0 表示客户端请求上传文件，ack为 1 表示服务器端对客户端上传文件请求的响应。一旦客户端收到上传文件的响应就将要发送的文件打包成一个个小块，服务器端根据filename（4 字节）和filenamelen（长度可变）字段来创建文件将客户端发来的文件数据写入。opcode字段值为 5、ack字段值为 0 表示客户端请求下载文件，ack为 1 表示服务器端对客户端下载文件请求的响应。一旦客户端收到下载文件的响应就根据filename和filenamelen字段来创建文件将服务器发来的文件数据写入，服务器端将要发送的文件打包成一个个小块。只有当ack值为 0 时filename和filenamelen字段才可用（不可用的字段值统一为 0）。

4.5 传输文件及确认

传输文件及确认的报文格式如下：

opcode	ack	block	datalen	data
1 Byte	1 Byte	4 Bytes	4 Bytes	

Figure 6: 传输文件及确认的报文格式

其中opcode字段值为 6 表示客户端正在上传文件，ack为 0 客户端向服务器发送文件块，ack为 1 表示服务器对客户端发来的文件块的确认（表示已经成功收到文件块）；opcode字段值为 7 表示客户端正在下载文件，ack为 0 表示服务器向客户端发送文件块，ack为 1 表示客户端对服务器发来的文件块的确认（表示已经成功收到文件块）。block字段表示文件块号，占 4 个字节。datalen表示文件块内数据的字节数，占 4 个字节（在这里设置为最大 1024 字节，其实该字段只需要 2 个字节就够了，设置 4 个字节是方便对文件块大小的扩充）。data字段大小是可变的（最大 1024 字节），表示文件块内的数据。只有当ack值为 0 时datalen和data字段才可用。

4.6 文件的可靠传输

为简化实现，假定建立连接、断开连接、请求服务器端文件列表的操作过程是可靠的，而文件传输可能会出现不可靠因素。

这里实现的可靠传输机制也非常简单，假定会发生丢包和延迟的现象，是类似于 Rdt 3.0 的停等协议。即发一个数据，等一个 ack。这里和 Rdt 3.0 的不同点是序列号的范围。Rdt 3.0 的序列号范围只有 0 和 1；这里的序列号数目等于文件块数目，序列号范围为 0~ 文件块数目-1。

这里将文件块的分成若干个连续的 1024 字节的小块（最后一块可能小于 1024 字节）。除此之外，还设定了一个数据大小为 0 的文件终止块（`ack=0`，`datalen=0`），表示该文件已经发完了。当一方收到这个块并发送这个块的确认，或接收到这个块的确认即可进入本次文件传输结束的状态。

由于是发一个数据，等一个 ack。因此不用考虑乱序的问题，只需要在发送方设置一个定时器，这里的定时器的超时统一设置成 5s。

发送方和接收方的工作流程如下：

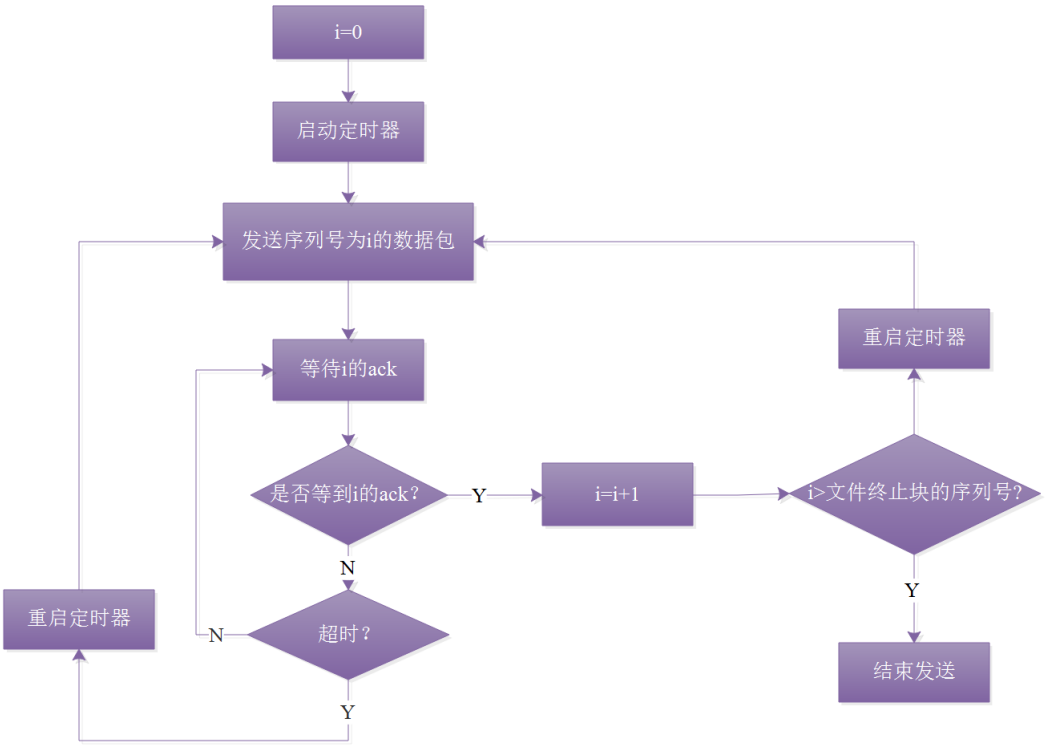

Figure 7: 发送方的工作流程

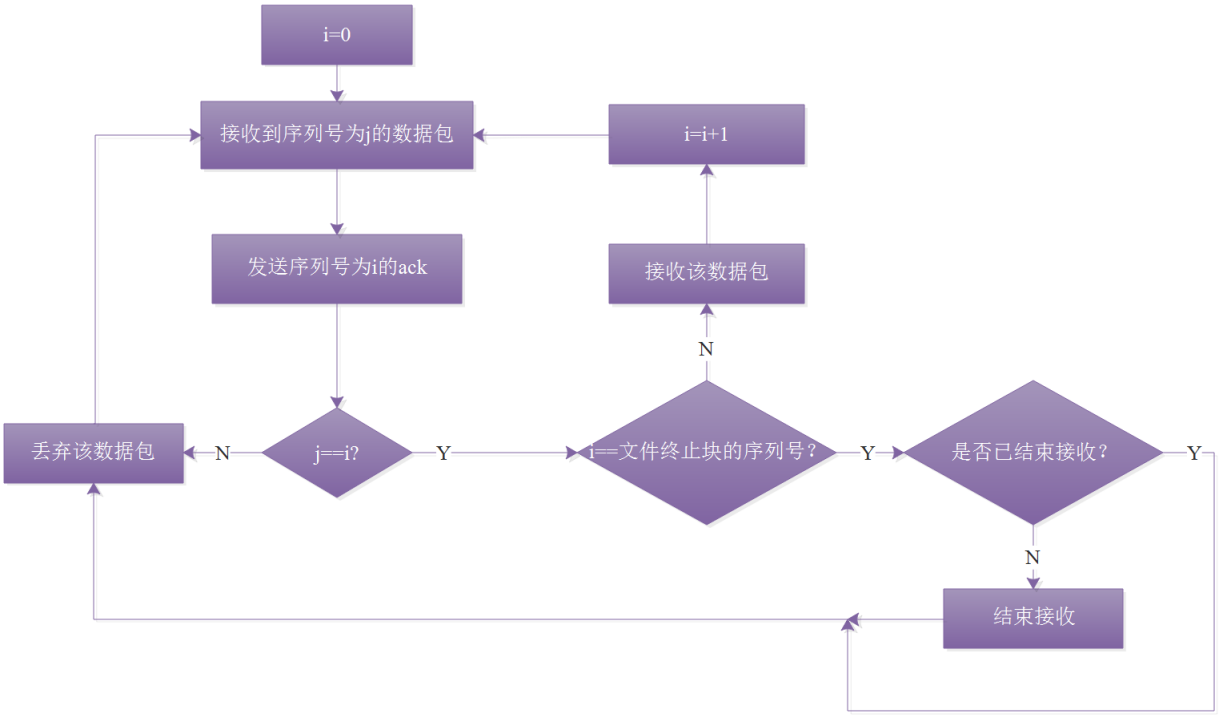

Figure 8: 接收方的工作流程

4.7 程序实现

在程序实现上，这次依然使用了`CAsyncSocket`类实现。

`CAsyncSocket`类主要的函数及其功能重复列举如下：

函数	功能
`Create()`	创建和初始化一个套接字
`SendTo()`	发送数据（数据报套接字）
`ReceiveFrom()`	接收数据（数据报套接字）
`Bind()`	将 IP 地址、端口号与套接字绑定
`Close()`	关闭套接字
`AsyncSelect()`	选择接收消息的模式

Table 1: CAsyncSocket 类的主要的函数及其功能

5 部分代码解释

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。

5.1　报文的结构体定义

这里的做法采取了类似网络技术与应用课上的编程实验的做法：

```
#pragma pack(1) //设置1字节对齐，强制让结构体的成员变量在内存中连续存放
 struct Pkt //opcode=1||2 or others
 {
     BYTE opcode;
     BYTE ack;
 };
 ......
 struct TransDataPkt  //opcode=6||7
 {
     BYTE opcode;
     BYTE ack;
     DWORD block;
     DWORD datalen;
     BYTE data[1]; //数组大小为1表示可变大小
 };
 #pragma pack() //恢复默认的字节对齐方式
```

创建数据包的部分函数如下（在CUFTPClientDlg.cpp或CUFTPServerDlg.cpp中）：

```
1  Pkt *makeSynFinPkt(BYTE opcode, BYTE ack) //建立连接/断开连接
2  {
3    Pkt *pkt = new Pkt;
4    memset(pkt, 0, sizeof(Pkt));
5    pkt->opcode = opcode;
6    pkt->ack = ack;
7    return pkt;
8  }
9
10 TransDataPkt *makeTransDataPkt(BYTE opcode, const u_char *data, int datalen
       , int block) //传输数据
11 {
12   TransDataPkt *pkt = (TransDataPkt*)malloc(sizeof(TransDataPkt) + datalen
       - 1);
13   memset(pkt, 0, sizeof(TransDataPkt) + datalen - 1);
14   pkt->opcode = opcode;
```

```
15    pkt->block = block;
16    pkt->datalen = datalen;
17    memcpy((void*)(pkt->data), data, datalen);
18    return pkt;
19  }
20
21  TransDataPkt *makeTransDataEnd(BYTE opcode, int block) //文件终止块
22  {
23    TransDataPkt *pkt = new TransDataPkt;
24    memset(pkt, 0, sizeof(TransDataPkt));
25    pkt->opcode = opcode;
26    pkt->block = block;
27    pkt->datalen = 0;
28    return pkt;
29  }
30
31  TransDataPkt *makeTransDataAck(BYTE opcode, int block) //对文件块的确认
32  {
33    TransDataPkt *pkt = new TransDataPkt;
34    memset(pkt, 0, sizeof(TransDataPkt));
35    pkt->opcode = opcode;
36    pkt->ack = 1;
37    pkt->block = block;
38    return pkt;
39  }
```

5.2 客户端与服务器之间的交互

由于服务器端的实现与客户端的实现相差不大，因此这里主要以客户端为例。

5.2.1 发送文件或 ack

```
1  void CUFTPClientDlg::sendPkt()
2  {
3    if (uploaddata) //如果处在上传数据状态，那么客户端就是发送方，就向服务器
       发送文件块
4    {
5      if (!isconnect) //由于UDP是无连接的，不能在未建立连接的时候做任何操作
6        return;
7      if (endsenddata) //数据传输完毕
```

```
8       return;
9     if (currsendpos+1<sendFileblock.size()) //如果当前文件块还没有发完，
    currsendpos为当前文件传输的进度（序列号），序列号等于文件块号减1
10    {
11      //send
12      if (sendFileblock[currsendpos + 1] == NULL&&currsendpos + 1 != 0) //
    如果文件块已经发完，准备发送文件终止块
13      {
14        lastsendpkt = currsendpos + 1; //标记最后一个块
15        TransDataPkt *pkt = makeTransDataEnd(6, currsendpos + 1); //制作并
    发送文件
16        if (m_pSocket->SendTo(pkt, sizeof(TransDataPkt), servport, servip)
    == SOCKET_ERROR)
17        {
18            ......
19            //错误处理
20        }
21        else
22        {
23          CString temp;
24          temp.Format(L"Send end block #%d", currsendpos + 1);
25          m_loglist.InsertString(m_loglist.GetCount(), temp);
26        }
27      }
28      else //继续发送文件块
29      {
30        TransDataPkt *pkt = makeTransDataPkt(6, sendFileblock[currsendpos +
     1]->buf, sendFileblock[currsendpos + 1]->len, currsendpos + 1); //制作
    并发送文件块
31        if (m_pSocket->SendTo(pkt, sizeof(TransDataPkt) + pkt->datalen - 1,
     servport, servip) == SOCKET_ERROR)
32        {
33            ......
34            //错误处理
35        }
36        else
37        {
38        //打印发出文件块的日志
39          CString temp;
40          temp.Format(L"Send block #%d", currsendpos + 1);
41          m_loglist.InsertString(m_loglist.GetCount(), temp);
42        }
```

```
43        }
44
45
46        cursendseq++; //当前序列号+1
47        currsendpos++; //发送进度+1
48        cursendseq %= maxsendseq; //不能让序列号超过最大范围
49      }
50      m_pSocket->AsyncSelect(FD_READ); //数据发送完了准备接收ack
51    }
52    else if (downloaddata) //如果处在下载数据状态，那么客户端就是接收方，就向
       服务器发送对文件块的确认
53    {
54      if (!isconnect)
55        return;
56      if (endrecvdata) //数据传输完毕
57        return;
58      //send
59      TransDataPkt *pkt = makeTransDataAck(7, currecvpos);
60      if (m_pSocket->SendTo(pkt, sizeof(TransDataPkt), servport, servip) ==
       SOCKET_ERROR)
61      {
62          ......
63        //错误处理
64      }
65      else
66      {
67      //打印收到ack的日志
68        CString temp;
69        temp.Format(L"Block #%d has been successfully received!", currecvpos)
       ;
70        m_loglist.InsertString(m_loglist.GetCount(), temp);
71        if (currecvpos == lastrecvpkt) //如果收到对最后一个块的确认，则本次传
       输结束，打印结束日志，关闭文件句柄，重置状态，启用按钮
72        {
73          endrecvdata = true;
74          downloaddata = false;
75          m_download.EnableWindow(TRUE);
76          m_upload.EnableWindow(TRUE);
77          m_servdir.EnableWindow(TRUE);
78          m_loglist.InsertString(m_loglist.GetCount(), L"Download file " +
       sefilename + " succeeded!");
79          currrecvfile.Close();
```

```
80        }
81        m_pSocket->AsyncSelect(FD_READ);//继续接收文件块
82      }
83    }
84  }
```

5.2.2 接收文件或 ack

CUFTPClientDlg::recvPkt()由从CAsyncSocket类派生出的套接字CClientSocket发出消息FD_READ的回调函数OnReceive()调用。

```
1  void CUFTPClientDlg::recvPkt()
2  {
3    u_char receiveBuf[4096];
4    CString servip1;
5    UINT servport1;
6    int ret = m_pSocket->ReceiveFrom(receiveBuf, sizeof(receiveBuf) - 1,
       servip1, servport1); //接收数据包
7    if (ret == SOCKET_ERROR)
8    {
9      ......
10     //错误处理
11   }
12   Pkt *pkt = (Pkt*)receiveBuf; //先判断收到的报文的操作码和ack字段，这里复
       用了建立连接/关闭连接的数据包来判断
13   if (pkt->opcode == 1 && pkt->ack == 1)  //收到服务器建立连接的响应
14   {
15     if (isconnect) //若已连接，则不做任何操作
16       return;
17     isconnect = true; //进入已连接状态
18     KillTimer(pkt->opcode); //关闭连接请求的定时器
19     m_loglist.InsertString(m_loglist.GetCount(), L"Connection to server
       succeeded!");
20     ((CStatic*)GetDlgItem(IDC_CONNSTATE))->SetWindowTextW(L"与服务器连接情
       况：已连接"); //输出右上角状态
21     m_conn.EnableWindow(FALSE);
22     m_disconn.EnableWindow(TRUE);
23     m_servdir.EnableWindow(TRUE);
24     m_servip.EnableWindow(FALSE);
25     m_servport.EnableWindow(FALSE);
26     m_upload.EnableWindow(TRUE);
27     m_download.EnableWindow(TRUE);
```

```
28    }
29    else if (pkt->opcode == 2 && pkt->ack == 1)  //收到服务器关闭连接的响应
30    {
31      if (!isconnect) //若未连接，则不做任何操作
32        return;
33      isconnect = false;
34      KillTimer(pkt->opcode);
35      m_loglist.InsertString(m_loglist.GetCount(), L"Connection to server is
        closed!");
36      ((CStatic*)GetDlgItem(IDC_CONNSTATE))->SetWindowTextW(L"与服务器连接情
        况：未连接"); //输出右上角状态
37      m_conn.EnableWindow(TRUE); //设置除了“建立连接”按钮以外其他按钮不可点
        击
38      m_disconn.EnableWindow(FALSE);
39      m_servdir.EnableWindow(FALSE);
40      m_servip.EnableWindow(TRUE);
41      m_servport.EnableWindow(TRUE);
42      m_upload.EnableWindow(FALSE);
43      m_download.EnableWindow(FALSE);
44    }
45    else if (pkt->opcode == 2 && pkt->ack == 0)  //收到服务器关闭连接的请求
46    {
47      if (!isconnect) //若未连接，则不做任何操作
48        return;
49      isconnect = false;
50      m_loglist.InsertString(m_loglist.GetCount(), L"Server request to close
        connection!");
51      ((CStatic*)GetDlgItem(IDC_CONNSTATE))->SetWindowTextW(L"与服务器连接情
        况：未连接");
52      //关闭所有定时器，以免可能发生继续重传数据包的情况
53      KillTimer(1);
54      KillTimer(2);
55      KillTimer(3);
56      KillTimer(4);
57      KillTimer(5);
58      KillTimer(6);
59      m_conn.EnableWindow(TRUE);
60      m_disconn.EnableWindow(FALSE);
61      m_servdir.EnableWindow(FALSE);
62      m_servip.EnableWindow(TRUE);
63      m_servport.EnableWindow(TRUE);
64      m_upload.EnableWindow(FALSE);
```

```
65      m_download.EnableWindow(FALSE);
66    }
67    else if (pkt->opcode == 3 && pkt->ack == 1)  //收到服务器返回的文件列表
68    {
69      if (!isconnect)
70        return;
71      KillTimer(pkt->opcode);
72      m_loglist.InsertString(m_loglist.GetCount(), L"Request for file list of
        server succeeded!");
73      FileListPkt *list = (FileListPkt *)pkt;
74      CFileDlg dlg;
75      dlg.filelist = list->filelistdata;
76      if (dlg.DoModal() == IDOK) //打开服务器文件列表的对话框
77      {
78        sefilename = dlg.curselstr; //选择要下载的文件
79        m_downloadfn.SetWindowTextW(sefilename);
80      }
81      else
82      {
83        sefilename = dlg.curselstr;
84        m_downloadfn.SetWindowTextW(L"");
85      }
86      m_download.EnableWindow(TRUE);
87      m_upload.EnableWindow(TRUE);
88      m_servdir.EnableWindow(TRUE);
89    }
90    else if (pkt->opcode == 4 && pkt->ack == 1)  //收到服务器允许上传文件的响
       应，接下来就要发文件了
91    {
92      if (!isconnect)
93        return;
94      KillTimer(pkt->opcode);
95      if (!currsendfile.Open(filepath, CFile::modeRead | CFile::typeBinary))
       //文件读取失败
96      {
97        m_loglist.InsertString(m_loglist.GetCount(), L"Open file "+refilename
       +L" failed! No such file or directory!");
98        m_upload.EnableWindow(TRUE);
99      }
100     else
101     {
102       uploaddata = true;
```

```
103       endsenddata = false;
104       downloaddata = false;
105       endrecvdata = false;
106       int dwRead = 0;
107       sendFileblock.clear();
108       sendFileblock.push_back(NULL); //不用0号，文件块号从1开始
109       //将文件分割成很多1024字节的小块，最后一块可能小于1024字节
110       do
111       {
112         Fileblock *block = new Fileblock;
113         //每次读1024个字节，可能小于1024字节
114         dwRead = currsendfile.Read(block->buf, 1024);
115         if (dwRead <= 0)
116           break;
117         block->len = dwRead;
118         sendFileblock.push_back(block); //将文件块加入到发送文件块列表中
119       }
120       while (dwRead > 0);
121       sendFileblock.push_back(NULL);  //告知对方文件已经发完的块（文件终止
      块）
122       SenderInit(); //初始化发送方状态
123       SetTimer(6, 5000, NULL); //设置定时器并发送文件块，超时为5s，如果超时
      则进入超时处理程序，准备重传
124       sendPkt(); //发送文件块
125     }
126   }
127   else if (pkt->opcode == 5 && pkt->ack == 1)  //收到服务器允许下载文件的响
      应，接下来就要准备接收文件了
128   {
129     if (!isconnect)
130       return;
131     KillTimer(pkt->opcode);
132     if (!currrecvfile.Open(L"shared\\" + sefilename, CFile::modeCreate |
      CFile::modeWrite | CFile::typeBinary)) //文件读取失败
133     {
134       m_loglist.InsertString(m_loglist.GetCount(), L"Open file " +
      sefilename + L" failed! Create file error!");
135       m_download.EnableWindow(TRUE);
136     }
137     else
138     {
139       downloaddata = true;
```

```
140       endrecvdata = false;
141       uploaddata = false;
142       endsenddata = false;
143       recvFileblock.clear();
144       recvFileblock.push_back(NULL);
145       ReceiverInit(); //初始化接收方状态
146       m_pSocket->AsyncSelect(FD_READ); //准备接收文件
147     }
148   }
149   else if (pkt->opcode == 6 && pkt->ack == 1)  //收到服务器对发送的某个文件
      块的确认
150   {
151     if (!isconnect)
152       return;
153     if (!(uploaddata&&!endsenddata)) //如果传输已经结束，则不做任何操作
154       return;
155     TransDataPkt *ack = (TransDataPkt*)pkt;
156     if (!sendackHandler((int)(ack->block) - 1)) //如果收到的ack不是当前已发
      送文件块的ack，则什么也不做，等待超时重传
157       return;
158     KillTimer(6); //关闭定时器
159     if (sendFileblock[ack->block]==NULL && ack->block != 0) //收到了对最后
      一个包的确认，则是传输完成
160     {
161       endsenddata = true;
162       uploaddata = false;
163       m_download.EnableWindow(TRUE);
164       m_upload.EnableWindow(TRUE);
165       m_servdir.EnableWindow(TRUE);
166       currsendfile.Close();
167       m_loglist.InsertString(m_loglist.GetCount(), L"Upload file "+
      refilename+" succeeded!");
168       //m_pSocket->AsyncSelect(FD_READ);
169       return;
170     }
171     sendPkt(); //继续发送文件
172     SetTimer(6, 5000, NULL); //重启定时器
173   }
174   else if (pkt->opcode == 7 && pkt->ack == 0)  //收到服务器传来的文件
175   {
176     if (!isconnect)
177       return;
```

```
178     if (!(downloaddata&&!endrecvdata))
179       return;
180     TransDataPkt *dat = (TransDataPkt*)pkt;
181     if (!(downloaddata&&!endrecvdata))//已传输结束，但还是收到了包
182     {
183       sendPkt();
184       return;
185     }
186     if (dat->block == currecvpos + 1) //如果当前收到的文件块号是刚刚的的文
        件块号，就接收此数据包并回复当前文件块号的ack
187     {
188       if (dat->datalen == 0)
189       {
190         lastrecvpkt = dat->block;
191       }
192       waitseq++;
193       currecvpos++;
194       recvseq = waitseq - 1;
195       currrecvfile.Write(dat->data, dat->datalen); //成功接收文件块，写文件
196       Fileblock *block = new Fileblock;
197       block->len = dat->datalen;
198       memcpy((void*)(block->buf), dat->data, dat->datalen);
199       recvFileblock.push_back(block);
200       sendPkt();
201     }
202     else //如果不是，就丢弃该数据包并回复最后一个已确认文件块号的ack
203     {
204       sendPkt();
205     }
206   }
207 }
```

5.2.3 超时处理

这是定时器超时消息WM_TIMER的回调函数OnTimer()。

```
1 void CUFTPClientDlg::OnTimer(UINT_PTR nIDEvent) //参数nIDEvent和opcode一致
2 {
3   // TODO: 在此添加消息处理程序代码和/或调用默认值
4   //除了发送的文件块超时以外，其他超时不做重传
5   if (nIDEvent == 1) //连接请求超时
6   {
```

```
7     KillTimer(1);
8     m_loglist.InsertString(m_loglist.GetCount(), L"Request for connection
      timed out!");
9     m_conn.EnableWindow(TRUE);
10  }
11  else if (nIDEvent == 2) //关闭连接请求超时
12  {
13    KillTimer(2);
14    m_loglist.InsertString(m_loglist.GetCount(), L"Request for
      disconnection timed out!");
15    m_disconn.EnableWindow(TRUE);
16  }
17  else if (nIDEvent == 3) //请求服务器文件列表超时
18  {
19    KillTimer(3);
20    m_loglist.InsertString(m_loglist.GetCount(), L"Request for file list of
      server timed out!");
21    m_download.EnableWindow(TRUE);
22    m_upload.EnableWindow(TRUE);
23    m_servdir.EnableWindow(TRUE);
24  }
25  else if (nIDEvent == 4) //上传请求超时
26  {
27    KillTimer(4);
28    m_loglist.InsertString(m_loglist.GetCount(), L"Request for uploading
      file timed out!");
29    m_download.EnableWindow(TRUE);
30    m_upload.EnableWindow(TRUE);
31    m_servdir.EnableWindow(TRUE);
32  }
33  else if(nIDEvent == 5) //下载请求超时
34  {
35    KillTimer(5);
36    m_loglist.InsertString(m_loglist.GetCount(), L"Request for downloading
      file timed out!");
37    m_download.EnableWindow(TRUE);
38    m_upload.EnableWindow(TRUE);
39    m_servdir.EnableWindow(TRUE);
40  }
41  else if (nIDEvent == 6) //发送的文件块超时
42  {
43    timeoutHandler(); //将发送文件的进度减1，以实现重传这个包的效果
```

```
44      m_loglist.InsertString(m_loglist.GetCount(), L"Timed out! Waiting for
        retransmission......");
45      KillTimer(6); //关闭并重启定时器
46      SetTimer(6, 5000, NULL);
47      m_pSocket->AsyncSelect(FD_WRITE); //重传
48    }
49    CDialogEx::OnTimer(nIDEvent);
50  }
```

5.2.4 发送和接收状态的维护

```
1   void CUFTPClientDlg::SenderInit() //初始化发送者状态
2   {
3     maxsendseq = sendFileblock.size() - 1; //序列号的数目为要发送的文件数
4     cursendseq = 0;//当前已发送出去的序列号
5     cursendack = 0;//当前已得到确认的最后一个序列号
6     currsendpos = 0;//当前发送进度
7     lastsendpkt = 0;//收到的文件终止块的序号标记
8     sendack = new bool[maxsendseq];
9     for (int i = 0; i < maxsendseq; i++)
10    {
11      sendack[i] = true;
12    }
13  }
14
15  bool CUFTPClientDlg::sendackHandler(int index)
16  {
17    if (index == cursendack) //如果收到的ack是刚发出去的包对应的ack
18    {
19      sendack[index] = true;
20      cursendack = (index + 1) % maxsendseq;
21      return true;
22    }
23    return false;
24  }
25
26  void CUFTPClientDlg::timeoutHandler()
27  {
28    int index;
29    index = cursendack % maxsendseq; //将发送进度减1（其实是设成当前确认序列
```

```
    号，因为发送完一个包以后，如果该包没得到确认，则当前序列号值比当前确认序
    列号大1)
30    sendack[index] = true;
31    currsendpos = cursendack;
32    cursendseq = cursendack;
33  }
34
35  void CUFTPClientDlg::ReceiverInit() //初始化接收者状态
36  {
37    recvseq = 0;//已确认的最后一个序列号
38    waitseq = 0;//期望的序列号
39    currecvpos = 0;//当前接收进度
40    lastrecvpkt = 0;//收到的文件终止块的序号标记
41  }
```

5.3　多用户的实现

在客户端中，这次选择直接随机生成端口号并用这个端口号去创建套接字（而不是由操作系统直接生成），如果没创建成功（可能这个端口号被占用）就重复刚才的过程直到成功为止。这样非但可以使用随机端口号，也可以知道自己的端口号是什么（由操作系统直接分配的端口不能直接看到，得通过收发数据包才能看到）。

服务器端维护了一个多用户列表，用于记录所有已连接客户端的状态信息（如果客户端请求建立连接，则将该客户端加入到列表中；如果请求断开连接，则将该客户端从列表中删除）。

由于CAsyncSocket是异步非阻塞套接字，并且默认和主界面线程绑定，因此在CAsyncSocket上实现多线程很困难（同步套接字相对比较容易实现多线程，但是异步非阻塞套接字可以提供一个类似于多线程的效果）。因此在这里可以充分利用 MFC 的消息机制，根据每次FD_READ消息到来时收到的数据包的 IP 和端口号可以确定是哪一个客户端发来的数据包，因此就可以由 IP 和端口号信息使用CUFTPServerDlg::sendPkt()函数定点回复数据包，这样就可以在单线程中实现多用户同时下载/上传文件了。除此之外，在有客户端请求下载文件时还要为每一个客户端维护一个定时器。

用户状态结构体和用户列表定义如下，结构体中有该客户端的 IP 地址、端口号和一系列状态：

```
struct Client
{
    CString ip;
    int port;
```

```
        bool uploaddata;
        bool downloaddata;
        bool endsenddata;//数据发送完成的标志
        CString sefilename;//客户端要上传文件的文件名
        CString refilename;//客户端要下载文件的文件名
        bool endrecvdata;//数据接收完成的标志
        CFile currsendfile;//当前发送文件的句柄
        CFile currrecvfile;//当前接收文件的句柄
        vector<Fileblock*> recvFileblock;//当前发送文件的文件块列表
        vector<Fileblock*> sendFileblock;//当前发送文件的文件块列表

        //sender
        int maxsendseq;
        int cursendseq;
        int cursendack;
        bool *sendack;
        int currsendpos; //当前发送文件的进度
        int lastsendpkt;
        void SenderInit();
        bool sendackHandler(int index);
        void timeoutHandler();

        //receiver
        int waitseq;
        int recvseq;
        int currecvpos;//当前接收文件的进度
        int lastrecvpkt;
        void ReceiverInit();
    };
    vector<Client*> clients; //已连接的用户信息列表
```

随机生成端口的逻辑如下：

```
1 BOOL CUFTPClientDlg::OnInitDialog()
2 {
3   CDialogEx::OnInitDialog();
4
```

```
  // 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
  srand((unsigned)time(NULL));//设置随机数种子为time
  do
  {
    port = rand() % 65536 + 10000;
  } while (!m_pSocket->Create(port, SOCK_DGRAM, FD_READ, ip));
  ......

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

5.4　测试功能

这里向服务器端添加了测试功能，通过指定丢包、延迟行为的发生概率和延迟时间来控制服务器实际的行为，来测试协议是否真的可靠。

```
int CUFTPServerDlg::udtHandler(int c)
{
  srand((unsigned)time(NULL) * 5); //使用time*5作为随机数种子
  CString drop, delay;
  double dropnum;
  int delaynum;
  m_setdrop.GetWindowTextW(drop);
  m_setdelay.GetWindowTextW(delay);
  if (drop.GetLength() == 0)
  {
    dropnum = 0.5; //如果没有输入发生率，默认发生率为0.5
  }
  else
  {
    dropnum = _ttof(drop);
  }
  if (delay.GetLength() == 0)
```

```
  {
    delaynum = 4000; //如果没有输入延时，则默认延时为4000ms
  }
  else
  {
    delaynum = _ttoi(delay);
  }
  int bound = (int)(dropnum * 100);
  int r = rand() % 101; //随机一个值，由均匀分布确认是否发生
  if (r <= bound) //如果随机的结果小于等于发生率则发生
  {
    if (((CButton *)GetDlgItem(IDC_DROP))->GetCheck() == TRUE) //检查是否选
    择了丢包按钮
    {
      return 2;
    }
    else if (((CButton *)GetDlgItem(IDC_DELAY))->GetCheck() == TRUE) //检查
    是否选择了延时按钮
    {
    //打印延时日志
      CString temp;
      temp.Format(L"Client ip: %s port: %d A delay event happened!",
    clients[c]->ip, clients[c]->port);
      m_loglist.InsertString(m_loglist.GetCount(), temp);
      XSleep(delaynum); //实现延迟并不阻塞主线程
      return 3;
    }
  }
  return 1; //未发生
}
```

测试功能的添加的具体位置为服务器的`CUFTPServerDlg::recvPkt()`中在服务器收到文件块或对文件块的 ack 时。如果`udtHandler()`函数的返回值为 2，则是丢包，直接从`recvPkt()`返回，不对这个收到的包作任何处理。

以下是服务器端的`recvPkt()`的代码，只展示与客户端代码不同的地方：

```
void CUFTPServerDlg::recvPkt()
{
  u_char receiveBuf[4096];
  CString clientip1;
  UINT clientport1;
  int ret = m_pSocket->ReceiveFrom(receiveBuf, sizeof(receiveBuf) - 1,
    clientip1, clientport1);
```

```
7   if (ret == SOCKET_ERROR)
8   {
9     ......
10    //错误处理
11  }
12  Pkt *pkt = (Pkt*)receiveBuf;
13  if (pkt->opcode == 1 && pkt->ack == 0) //收到客户端连接请求
14  {
15    if (findClient(clientip1, clientport1) < 0)  //不存在此用户，就新建一个
      用户
16    {
17      Client *client = newClient(clientip1, clientport1);
18      clients.push_back(client);
19    }
20    else
21      return;
22    Pkt *pkt1 = makeSynFinPkt(1, 1);
23    //发送连接响应
24    if (m_pSocket->SendTo(pkt1, sizeof(Pkt), clientport1, clientip1) ==
      SOCKET_ERROR)
25    {
26      ......
27      //错误处理
28    }
29    else
30    {
31      CString temp1, temp2;
32      temp1.Format(L"Client ip: %s port: %d request for connection",
      clientip1, clientport1);
33      m_loglist.InsertString(m_loglist.GetCount(), temp1);
34      temp2.Format(L"已连接的用户数: %d", clients.size());
35      ((CStatic*)GetDlgItem(IDC_USERINFO))->SetWindowTextW(temp2);
36    }
37  }
38  else if (pkt->opcode == 2 && pkt->ack == 0) //收到客户端关闭连接请求
39  {
40    int c;
41    if ((c = findClient(clientip1, clientport1)) >= 0)  //删除此用户
42    {
43      clients.erase(clients.begin() + c);
44    }
45    else
```

```
46        return;
47      Pkt *pkt1 = makeSynFinPkt(2, 1);
48      //发送关闭连接响应
49      if (m_pSocket->SendTo(pkt1, sizeof(Pkt), clientport1, clientip1) ==
      SOCKET_ERROR)
50      {
51        ......
52        //错误处理
53      }
54      else
55      {
56        CString temp1, temp2;
57        temp1.Format(L"Client ip: %s port: %d request for disconnection",
      clientip1, clientport1);
58        m_loglist.InsertString(m_loglist.GetCount(), temp1);
59        temp2.Format(L"已连接的用户数: %d", clients.size());
60        ((CStatic*)GetDlgItem(IDC_USERINFO))->SetWindowTextW(temp2);
61      }
62    }
63    else if (pkt->opcode == 3 && pkt->ack == 0)   //向客户端返回文件列表
64    {
65      int c = findClient(clientip1, clientport1);
66      if (c < 0)
67        return;
68      findAllFile();//找出服务器端文件夹的所有文件
69      //制作服务器端文件列表
70      CString filelist;
71      for (int i = 0; i < allFile.size(); i++)
72      {
73        filelist += (allFile[i] + L"\n");
74      }
75      FileListPkt *filelistpkt = makeFileListReply(CString2char(filelist));
76
77      if (m_pSocket->SendTo(filelistpkt, sizeof(FileListPkt)+filelistpkt->len
      -1, clientport1, clientip1) == SOCKET_ERROR)
78      {
79        ......
80        //错误处理
81      }
82      else
83      {
84        CString temp1;
```

```
85        temp1.Format(L"Client ip: %s port: %d request for file list",
    clientip1, clientport1);
86        m_loglist.InsertString(m_loglist.GetCount(), temp1);
87      }
88    }
89    else if (pkt->opcode == 4 && pkt->ack == 0)  //上传文件请求
90    {
91      ......
92    }
93    else if (pkt->opcode == 5 && pkt->ack == 0) //下载文件请求
94    {
95      ......
96    }
97    else if (pkt->opcode == 6 && pkt->ack == 0)  //收到客户端发来的文件块
98    {
99          TransDataPkt *dat = (TransDataPkt*)pkt;
100     int c = findClient(clientip1, clientport1);
101     if (c < 0)
102       return;
103     int udt = udtHandler(c);
104     //如果丢包事件发生，就直接打印丢包日志，不对此包作任何处理
105     if (udt == 2)
106     {
107       //这里丢弃的是来自客户端的文件块
108       CString temp;
109       temp.Format(L"Client ip: %s port: %d The block #%d is dropped!",
    clientip1, clientport1, dat->block);
110       m_loglist.InsertString(m_loglist.GetCount(), temp);
111       return;
112     }
113     ......
114   }
115   else if (pkt->opcode == 7 && pkt->ack == 1)  //下载文件块确认
116   {
117     TransDataPkt *ack = (TransDataPkt*)pkt;
118     int c = findClient(clientip1, clientport1);
119     if (c < 0)
120       return;
121     if (!(clients[c]->downloaddata&&!clients[c]->endsenddata))
122       return;
123     int udt = udtHandler(c);
124     if (udt == 2) //如果丢包事件发生，就直接打印丢包日志，不对此包作任何处
```

```
    理
125     {
126         //这里要丢弃的是来自客户端的ack
127       CString temp;
128       temp.Format(L"Client ip: %s port: %d The ack of block #%d is dropped!
    ", clientip1, clientport1, ack->block);
129       m_loglist.InsertString(m_loglist.GetCount(), temp);
130       return;
131     }
132     ......
133     SetTimer(10000 + c, 5000, NULL);//10000+客户端在列表中的序号为该客户端
    定时器的序号
134   }
135 }
```

6 程序演示

6.1 上传文件

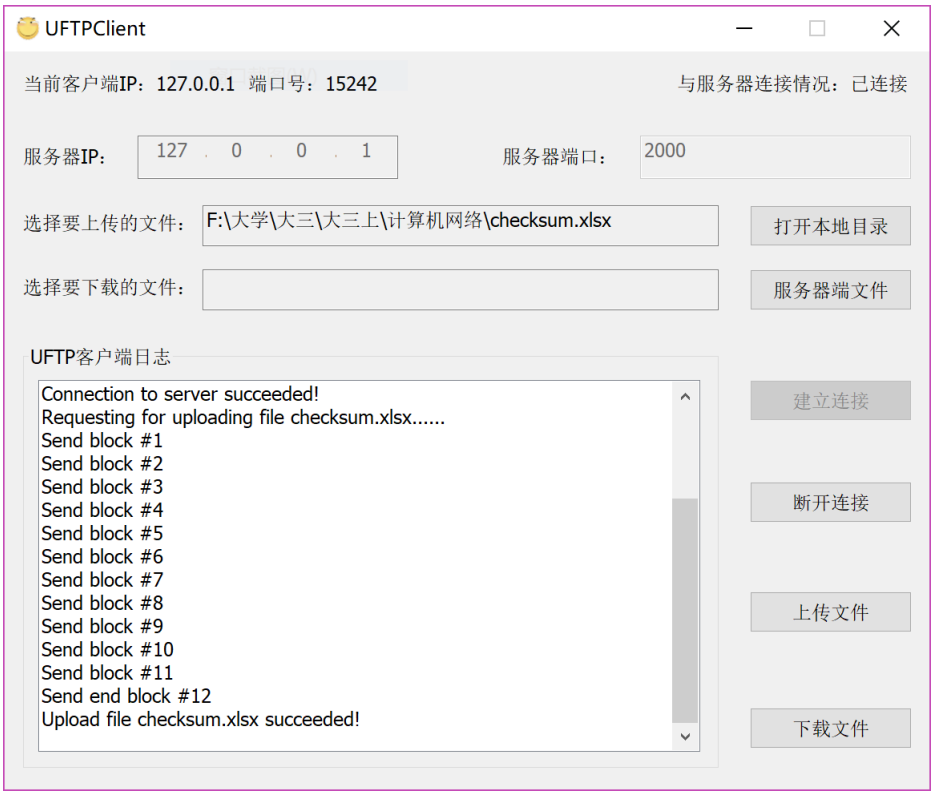

Figure 9: 客户端

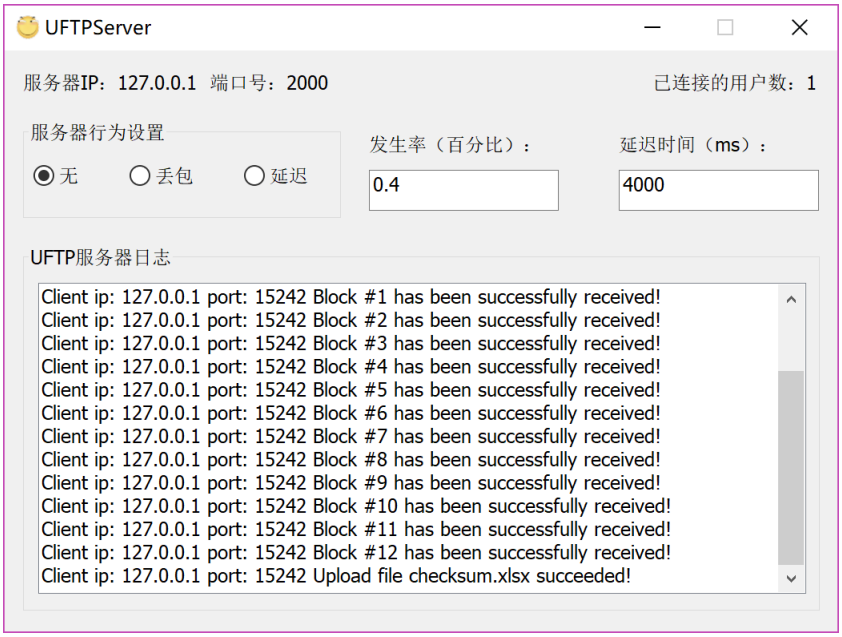

Figure 10: 服务器端

6.2 获取服务器文件列表

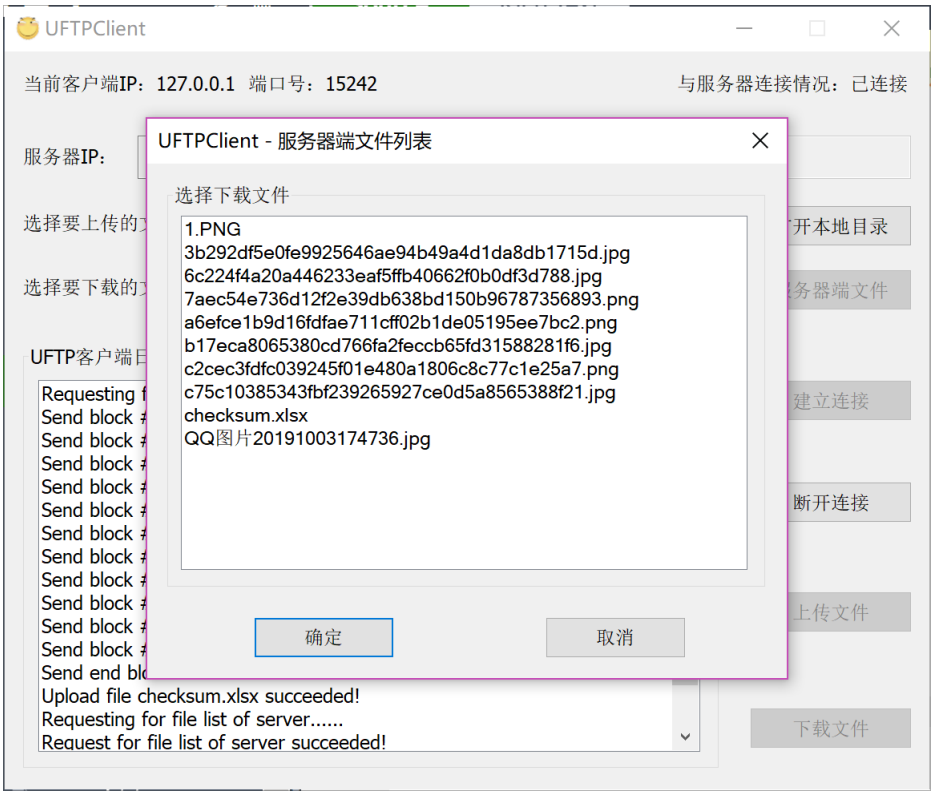

Figure 11: 服务器端文件列表

6.3 下载文件

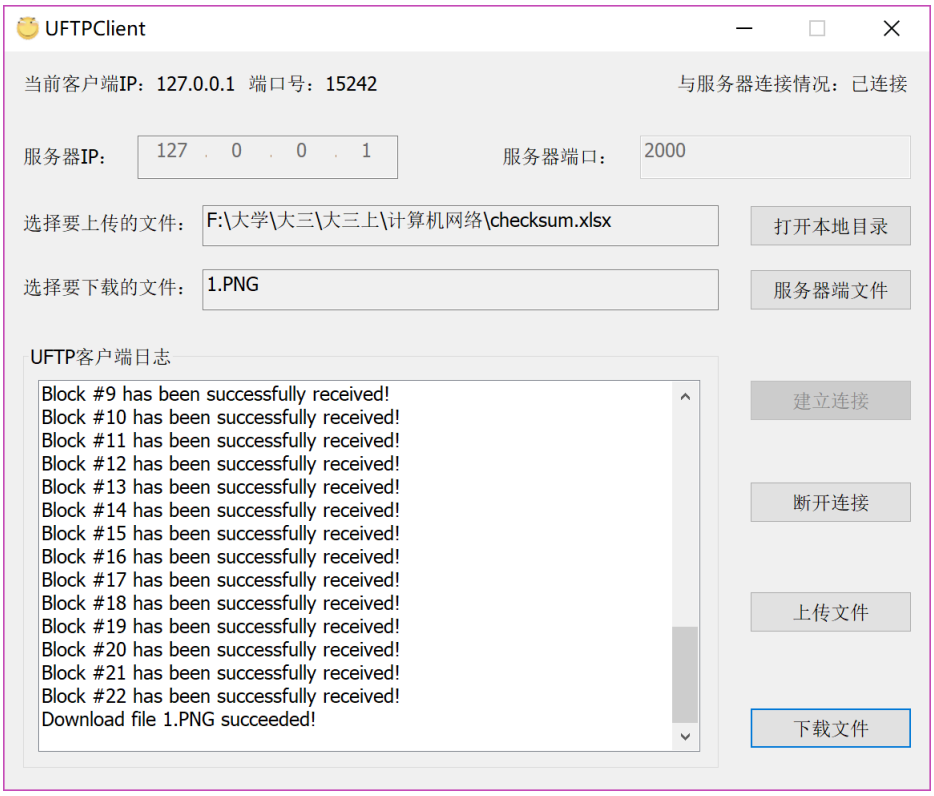

Figure 12: 客户端

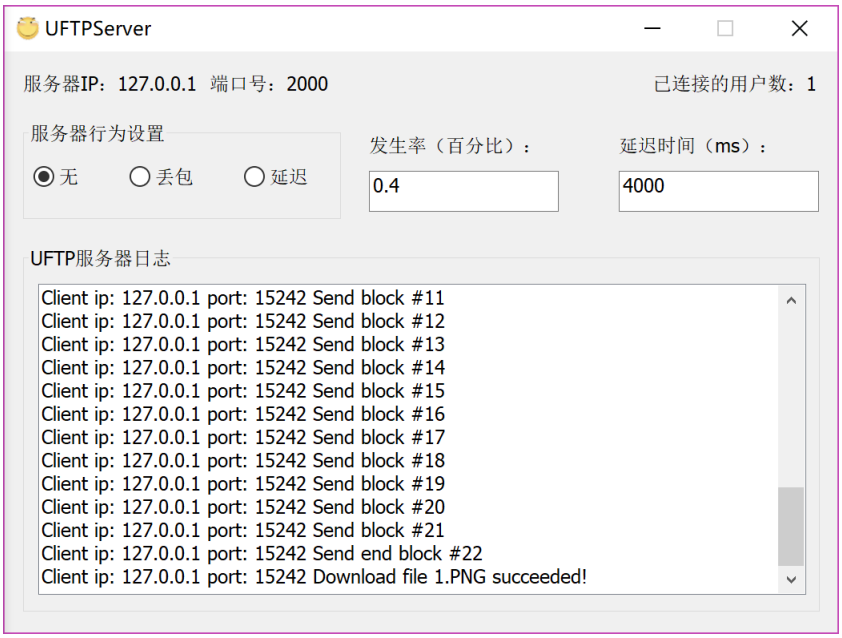

Figure 13: 服务器端

6.4 多用户同时下载和上传文件

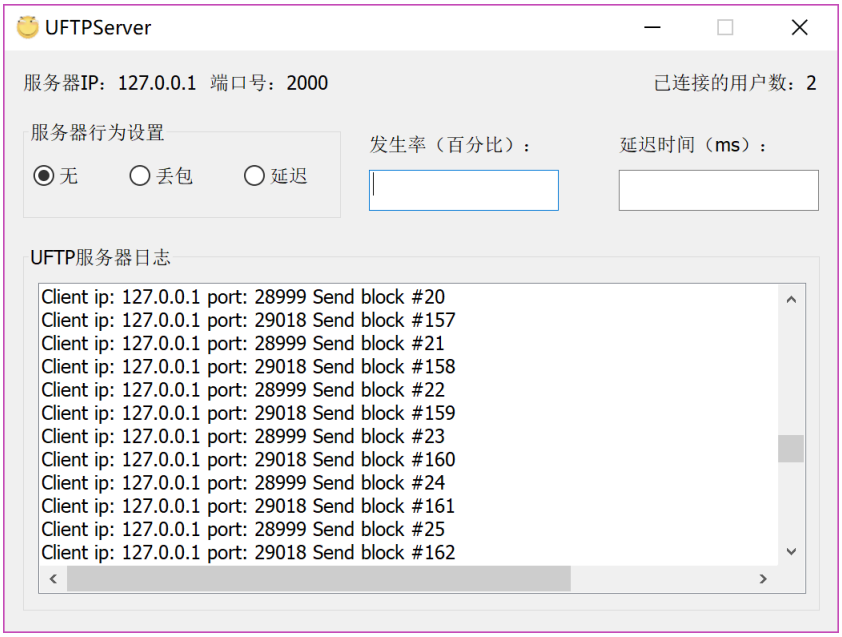

Figure 14: 同时下载文件

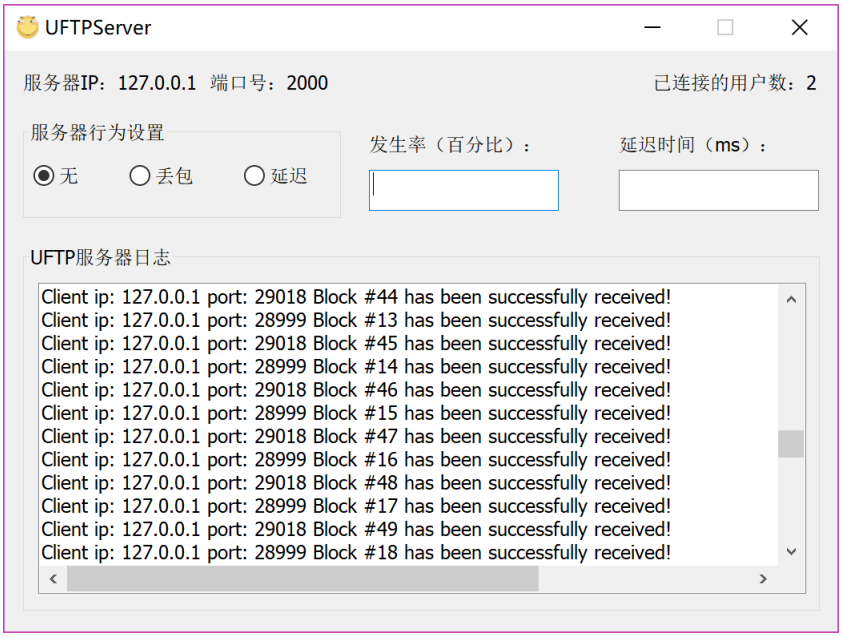

Figure 15: 同时上传文件

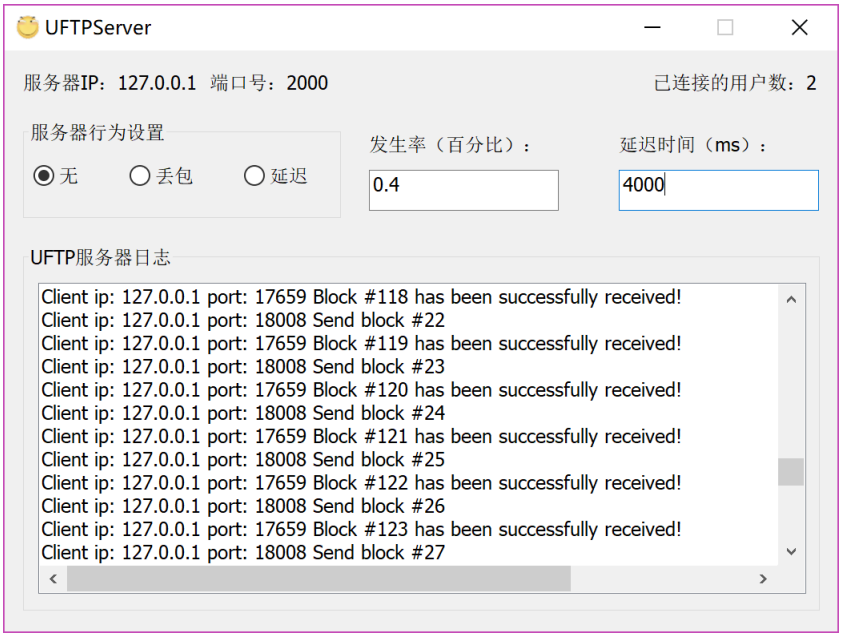

Figure 16: 同时下载和上传文件

6.5 丢包

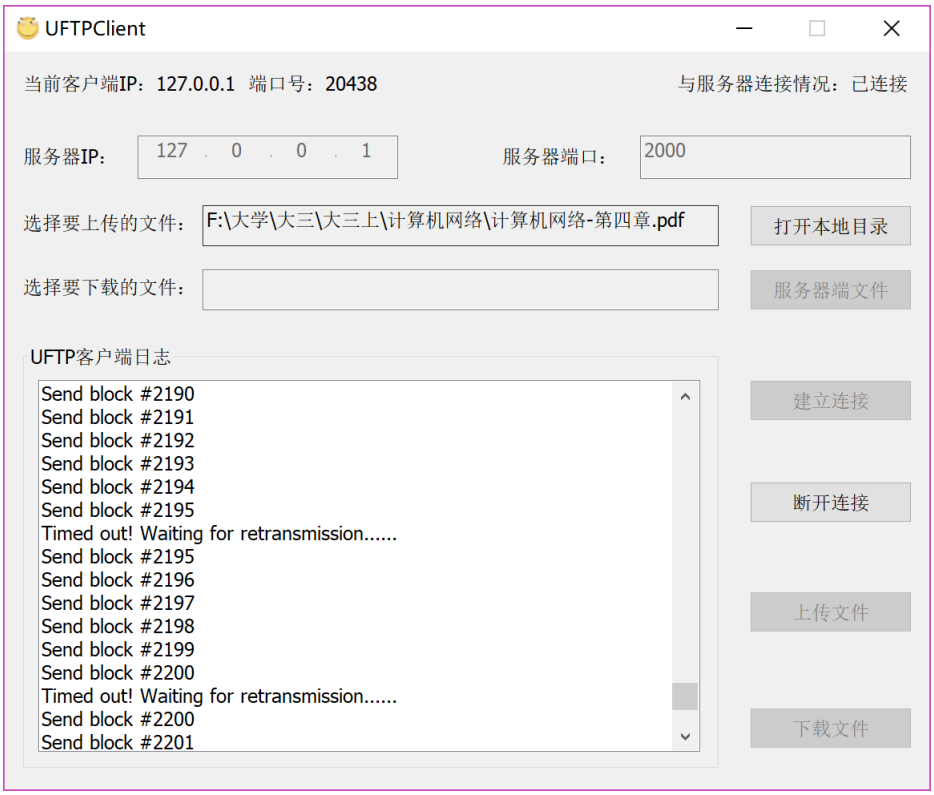

Figure 17: 客户端上传文件

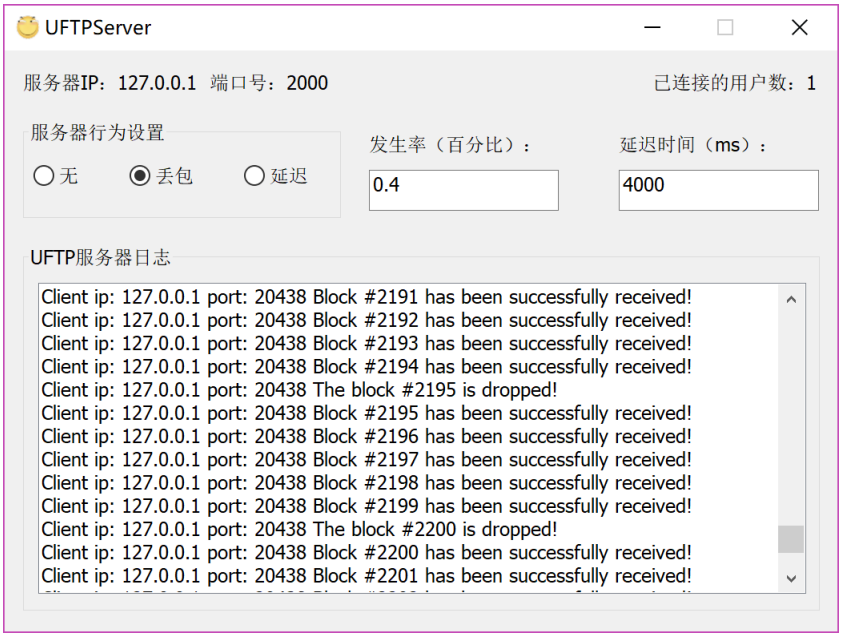

Figure 18: 服务器端接收上传的文件

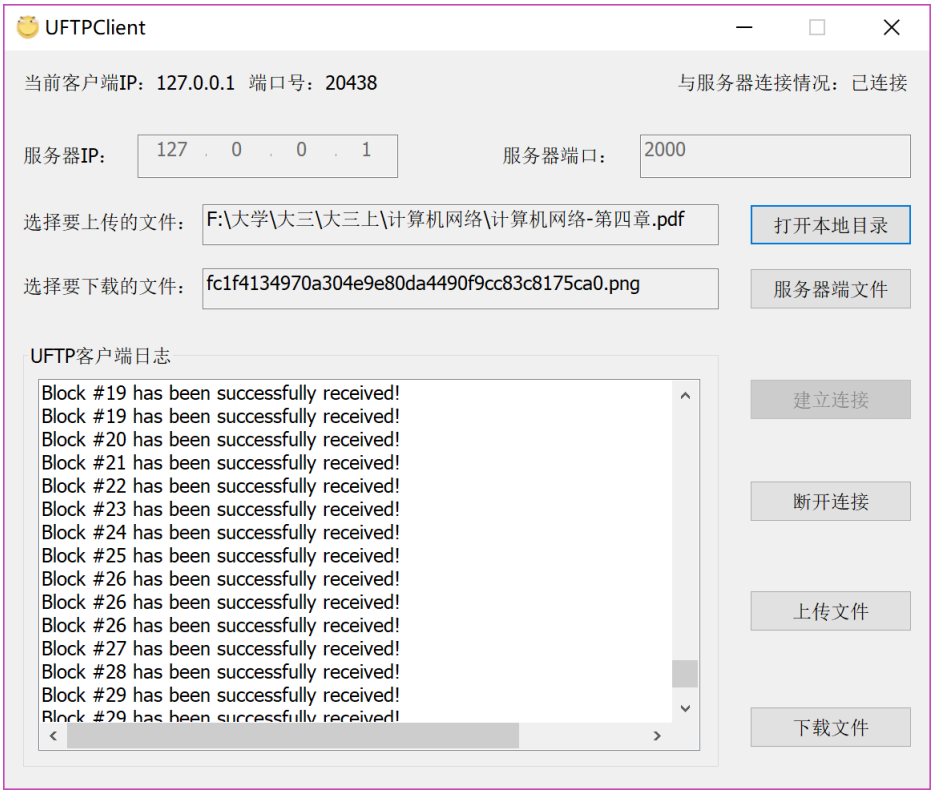

Figure 19: 客户端接收下载的文件

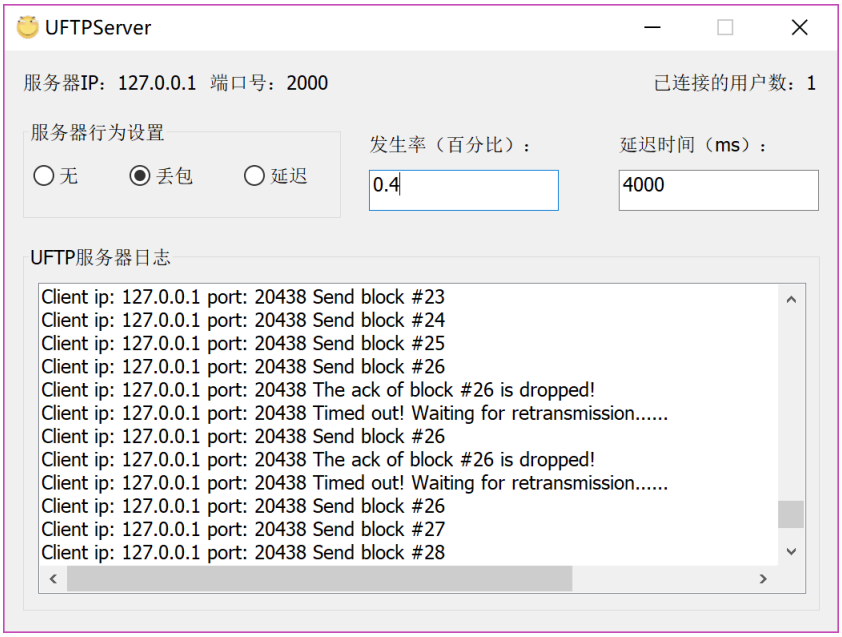

Figure 20: 服务器端发送下载的文件

6.6 延迟

如果在下载文件的时候引入比定时器时间短（5s）的延迟，则不会发生丢包重传现象：

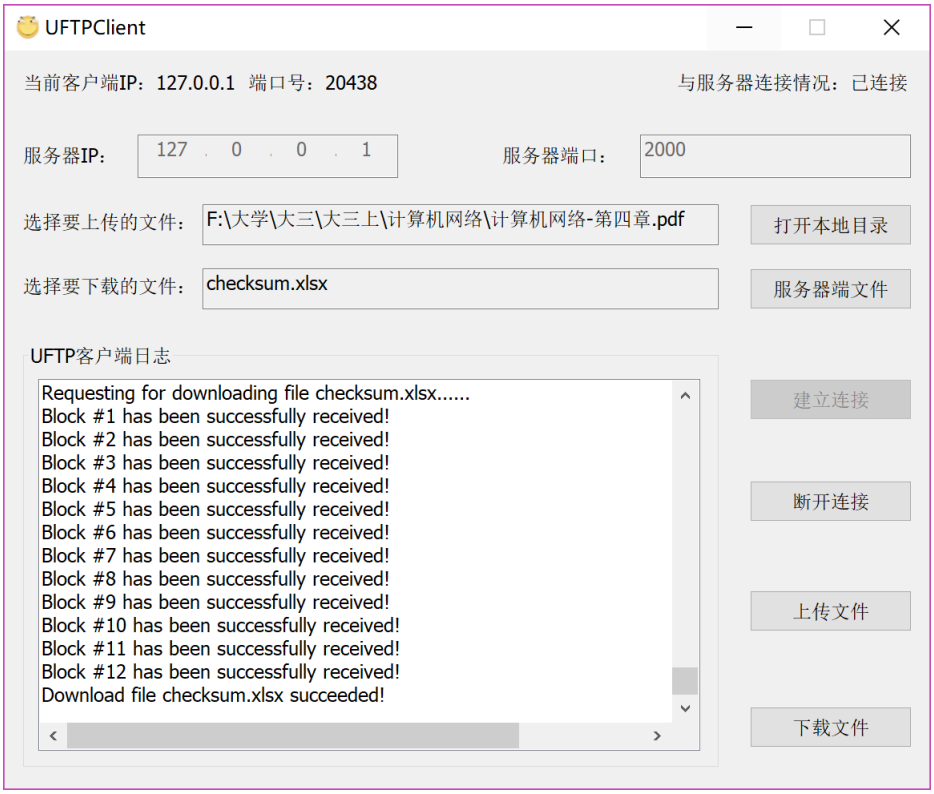

Figure 21: 客户端接收下载的文件

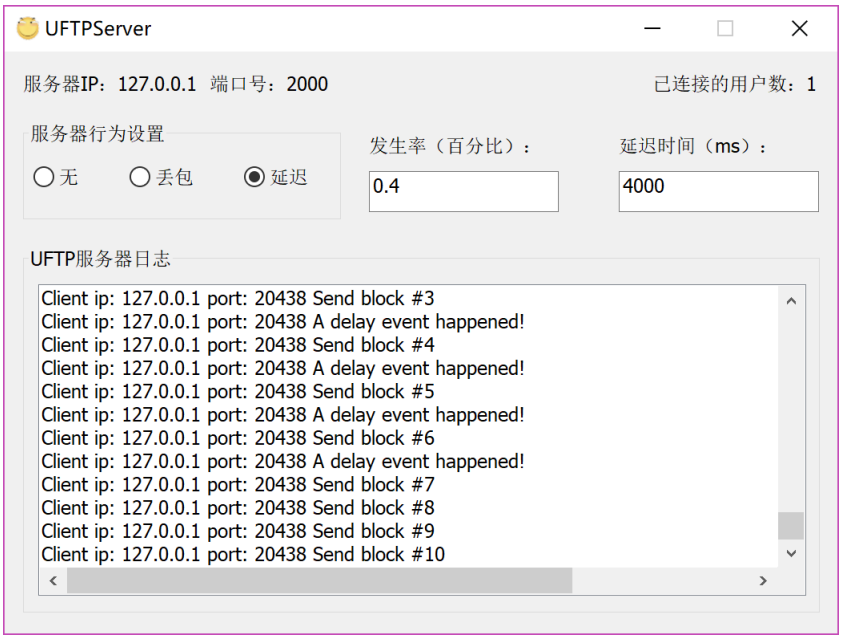

Figure 22: 服务器端发送下载的文件

如果引入比定时器时间长（5s）的延迟，就会发生丢包重传现象，但是在这里服务

器端日志输出的 bug 还没有解决：

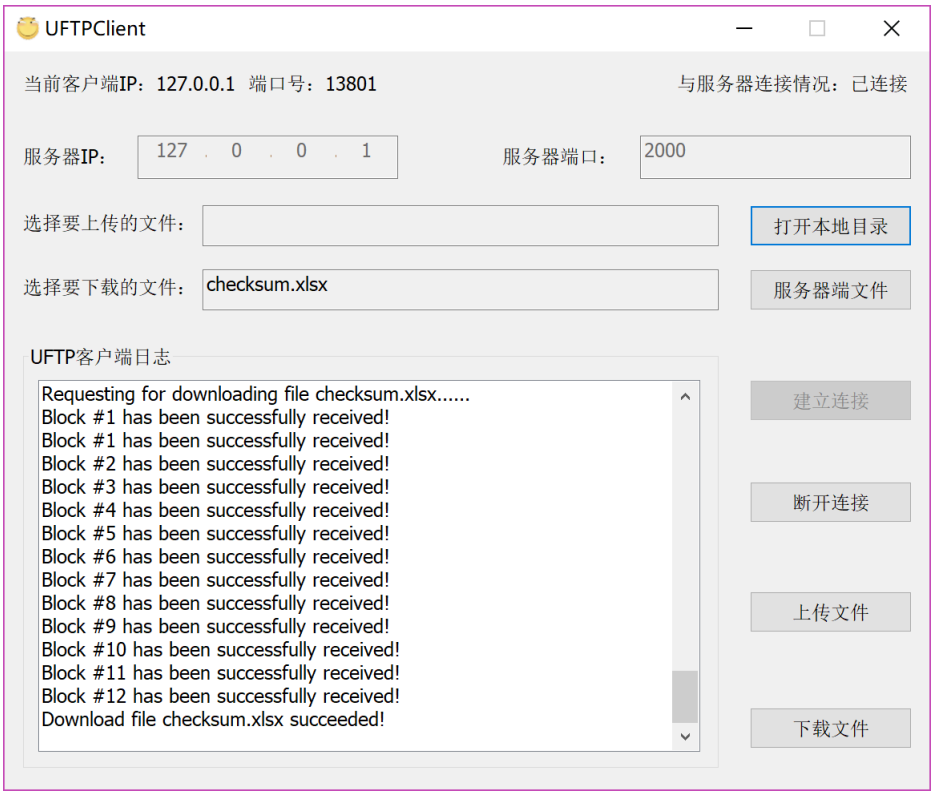

Figure 23: 客户端接收下载的文件

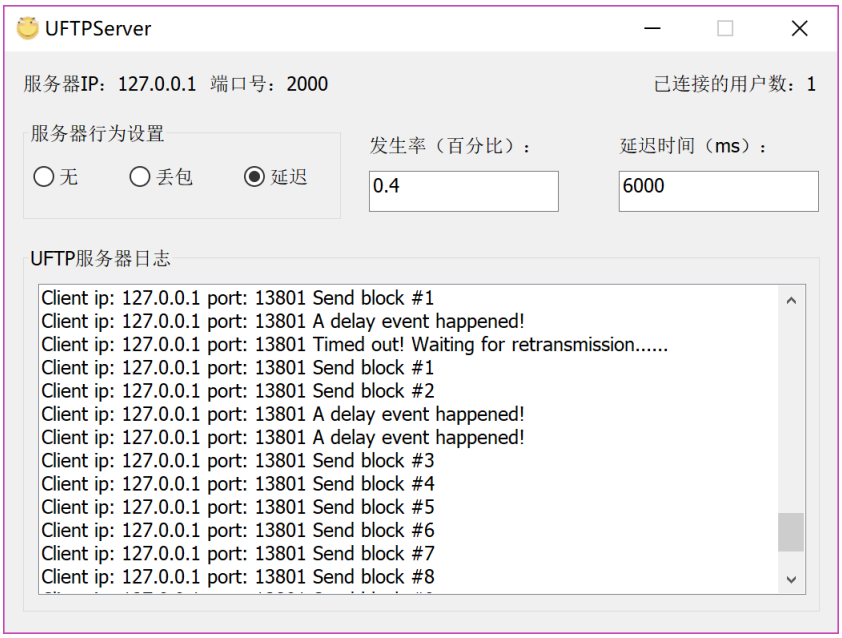

Figure 24: 服务器端发送下载的文件

如果在上传文件的时候引入比定时器时间短（5s）的延迟，则不会发生丢包重传现象：

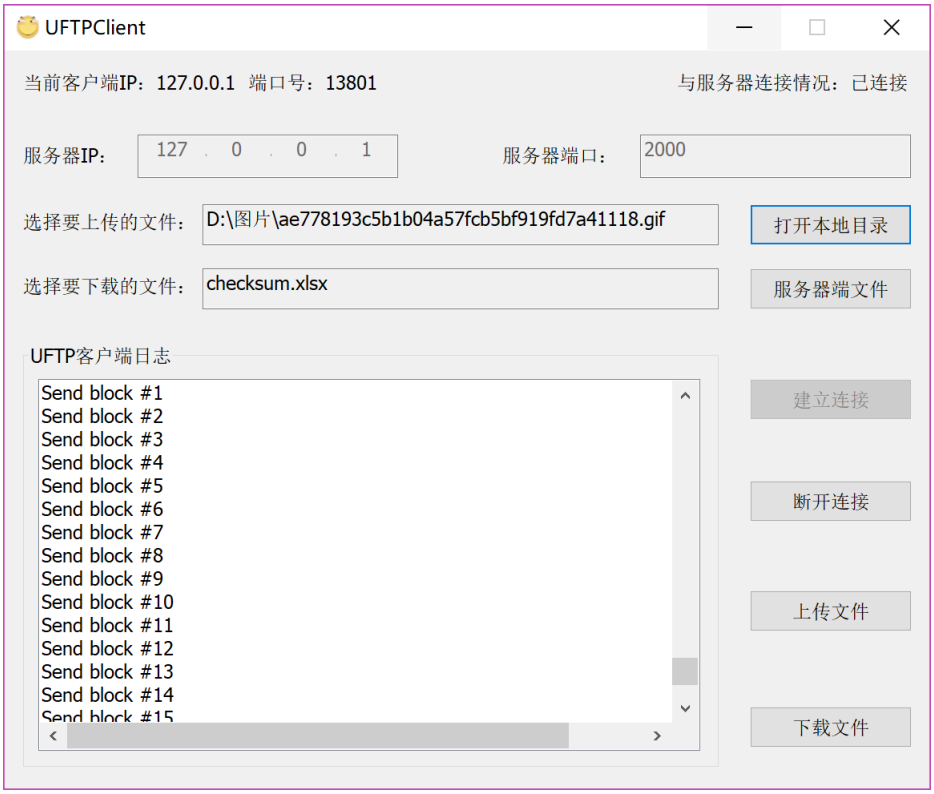

Figure 25: 客户端上传文件

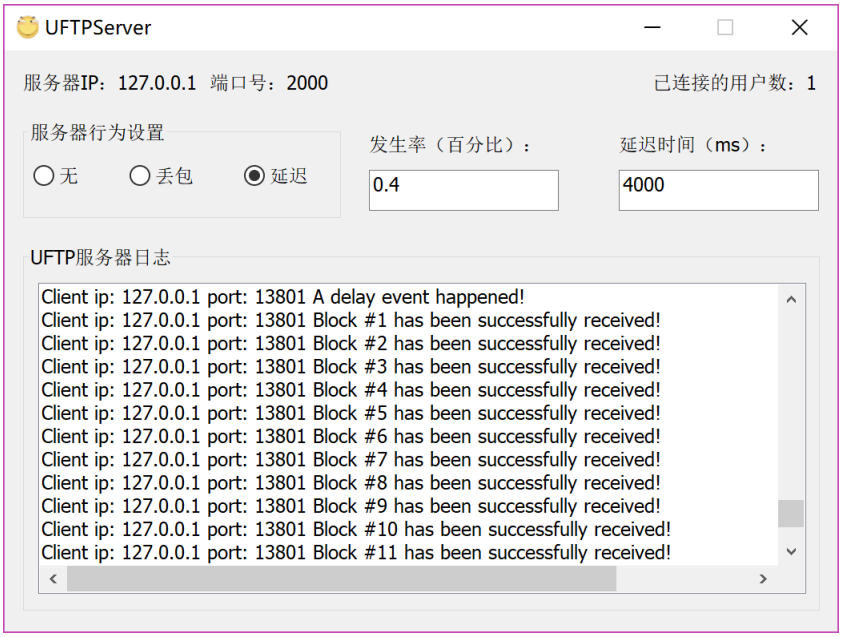

Figure 26: 服务器端接收上传的文件

如果引入比定时器时间长（5s）的延迟，就会发生丢包重传现象：

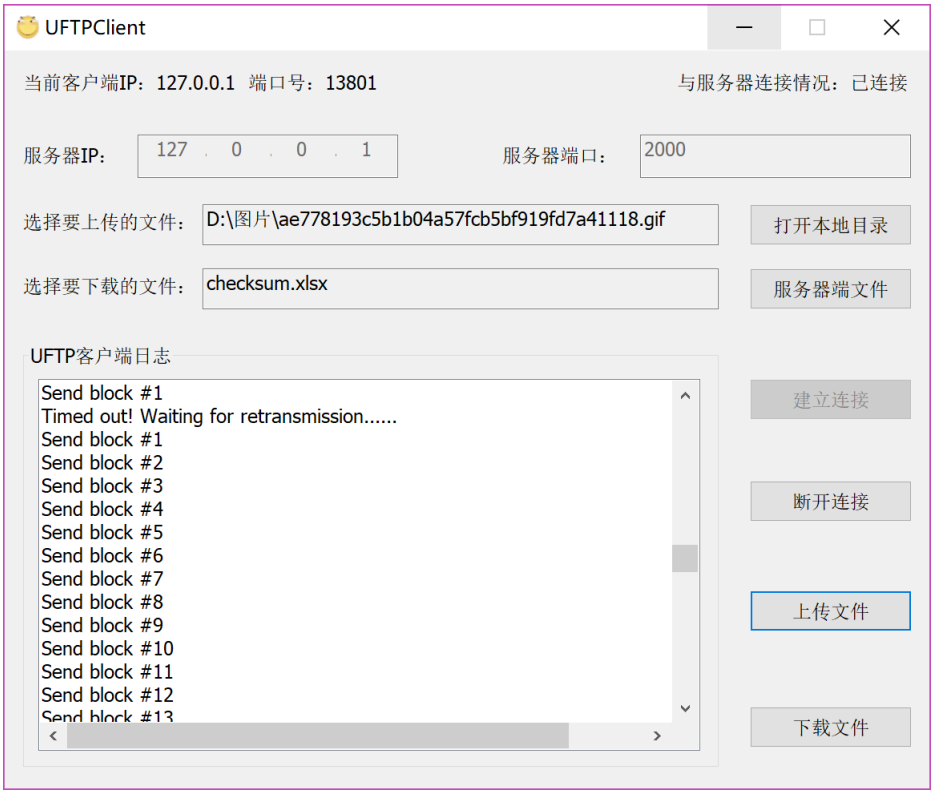

Figure 27: 客户端上传文件

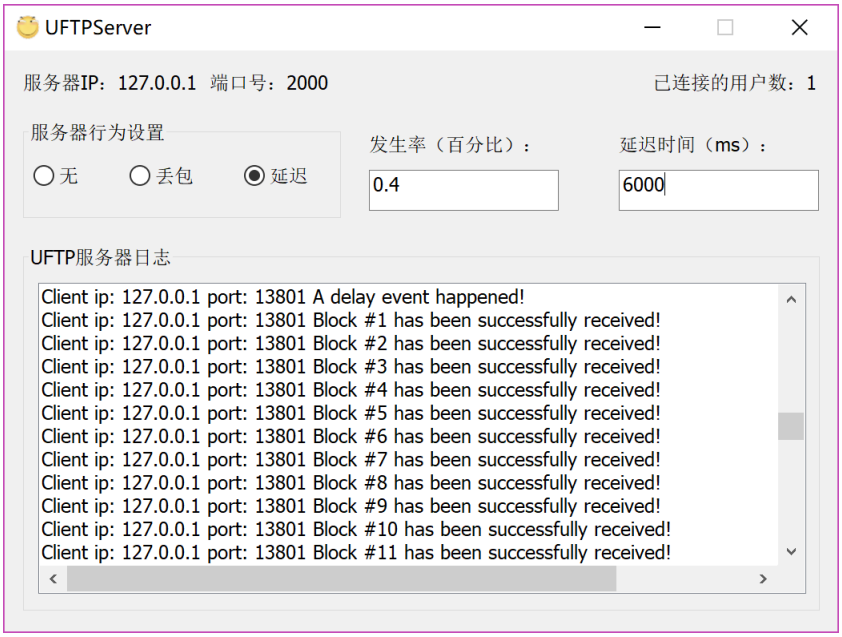

Figure 28: 服务器端接收上传的文件