该协议为自定义协议，模仿了 YModem-1K 帧结构，每包传输 1K 数据，最后一包数据根据文件大小决定，为总字节数对 1024 取整。

该协议用 N 字节信息块传输，N 可以自定义，数据的发送会使用 CRC16 校验，保证数据传输的正确性。它每传输一个信息块数据时，就会等待接收端回应 ACK 信号，接收到回应后，才会继续传输下一个信息块，保证数据已经全部接收。且支持了下位机的序列包定位，当漏发了一包数据，或者需要跳转，当前仅支持按照 1024 字节整数倍进行文件跳转，该协议不包含结束帧，可根据序列号进行判断。文件大小应小于 67108864 Bytes，即 64Mbyte。

文件发送篇

1、起始帧的数据格式

起始帧并不直接传输文件的数据，而是将文件名与文件的大小放在数据帧中传输，它的帧长=4 字节帧头+4 字节文件大小+2 字节包大小+2 字节 CRC16 校验码+文件名(不定长字符串)。

它的数据结构如下：

AA BB CC DD FileSize[4] PacketSize[2] CRCH CRCL filename[…]

其中 AA BB CC DD，表示这个数据帧为起始帧；在帧头后面的 FileSize [4]表示文件大小，4 个字节高位在前低位在后；PacketSize[2] 表示每包文件数据大小，文件将拆分成多个 PacketSize 进行传输；CRCH CRCL 分别表示 16 位 CRC 校验码的高 8 位与低 8 位，校验的数据为 4 字节文件长度+2 字节包大小；filename[…]就是文件名，如文件名 foo.c，它在数据帧中存放格式为：66 6F 6F 2E 63 00，一定要在文件名最后跟上一个 00，表示文件名结束 。

2、数据帧的数据格式

数据帧中会预留 PacketSize 字节空间用来传输文件数据，它跟起始帧接收差不多，如下：

00 00 data[PacketSize] CRCH CRCL

其中 00 00 表示第一帧数据帧，当然如果是第二帧数据的话就是：00 01；data[PacketSize]表示存放着 PacketSize 字节的文件数据；CRCH 与 CRCL 是 CRC16 校验码的高 8 位与低 8 位，校验的数据为 data 中的数据。

如果文件数据的最后剩余的数据小于 PacketSize，假设最后一包序列号为 num 的数据，剩余 n 字节数据，且 n< PacketSize，则如下结构：

[num] data[n] CRCH CRCL

3、文件传输过程

文件的传输过程，以具体的例子说明。把 foo.c，大小为 4196Byte（16 进制为 0x1064）的文件作为传输的对象，拆分为 1024byte(PacketSize=1024, 16 进制为 0x0400)一包，则它的传输过程如下：

发送端 接收端

AA BB CC DD 00 00 64 10 **04 00** CRCH CRCL "foo.c" >>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK

00 00 data[1024] CRCH CRCL >>>>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK

00 01 data[1024] CRCH CRCL >>>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK

00 02 data[1024] CRCH CRCL >>>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK

00 03 data[1024] CRCH CRCL >>>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK

00 04 data[100] CRCH CRCL >>>>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< ACK

传输过程就是上面所示。但是上面传输过程中存在许多通信信号，它们的数值与意义如下表所示：

符号	数值	含义
ACK	0x06	正常回应
ERR	0X07	校验错误
ERR1	0x08	请求重传序列号包
CA	0XFFFF	传输中止

还是有几点需要说明下：

1）CA 中止传输信号都可以发送，可在需要传输新文件时发送

2）ERR 信号，当校验失败的时候，由接收方发送，发送方收到后重传当前序列包。

3）ERR1 信号，基本上不可能出现。当出现该信号的时候说明帧错乱，需要重新定位序列帧数据，请求重传，格式为 0x08 numH numL ，num 为请求的序列包，高字节在前，低字节在后。

4、CRC 的计算

采用的是 CRC-16-IBM(A001)的 CRC 校验，它的生成多项式为 x16+x12+x5+1。

下面列出两种 c 语言的计算方法查表和计算。

计算方式：

```
/**************************************************
*函数名称:CRC16RTU
*输    入:pszBuf   要校验的数据
          unLength  校验数据的长
*输    出:校验值
*功    能:循环冗余校验-16
           （RTU 标准-0xA001）
**************************************************/
u16 CRC16RTU( u8 * pszBuf, u16 unLength)
{
    u16 CRCx=0XFFFF;
    u32 CRC_count;
    for(CRC_count=0;CRC_count<unLength;CRC_count++)
    {
        int i;
        CRCx=CRCx^*(pszBuf+CRC_count);

        for(i=0;i<8;i++)
        {
            if(CRCx&1)
            {
                CRCx>>=1;
                CRCx^=0xA001;
            }
            else
            {
               CRCx>>=1;
            }
        }
    }

    return CRCx;
}
```

查表方式：

```
//CRC 高位字节值表
const u8 auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
```

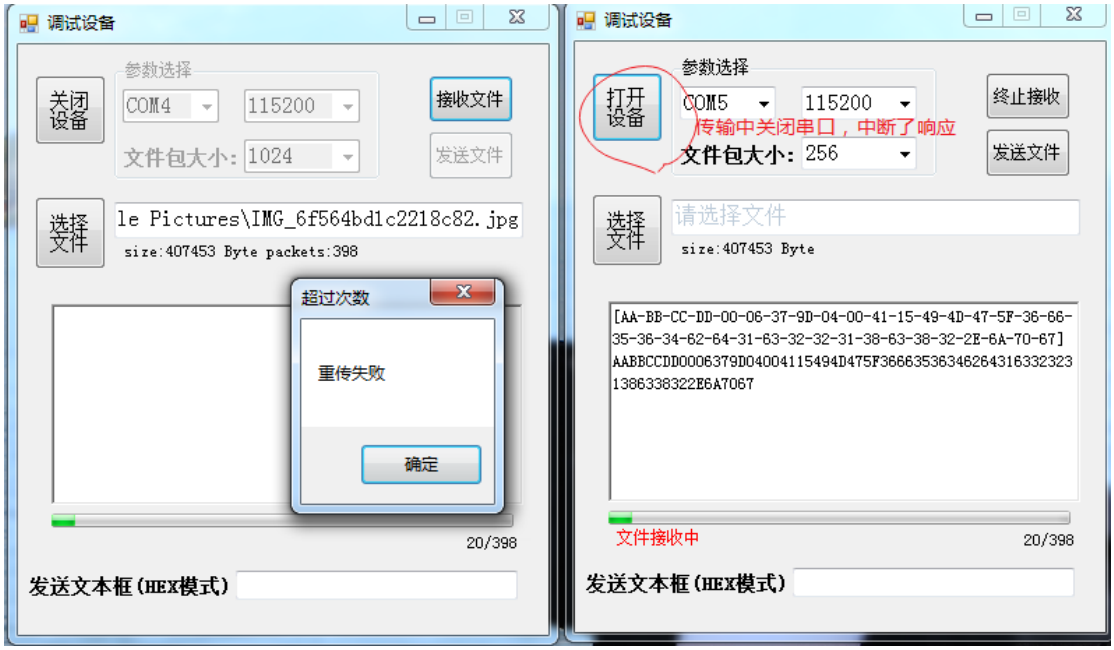
```
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
};
//CRC 低位字节值表
const u8 auchCRCLo[]={
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40
};
//获得 CRC16 值
//puchMsg:要校验的数组
//usDataLen:数组长度
u16 Get_Crc16(u8 *puchMsg,u16 usDataLen)
{
    u8 uchCRCHi=0xFF;       //高 CRC 字节初始化
    u8 uchCRCLo=0xFF;       //低 CRC 字节初始化
    u32 uIndex;             //CRC 循环中的索引
    while(usDataLen--)  //传输消息缓冲区
    {
        uIndex=uchCRCHi^*puchMsg++; //计算 CRC
        uchCRCHi=uchCRCLo^auchCRCHi[uIndex];
        uchCRCLo=auchCRCLo[uIndex];
    }
    return (uchCRCHi<<8|uchCRCLo);
}
```

文件接收篇

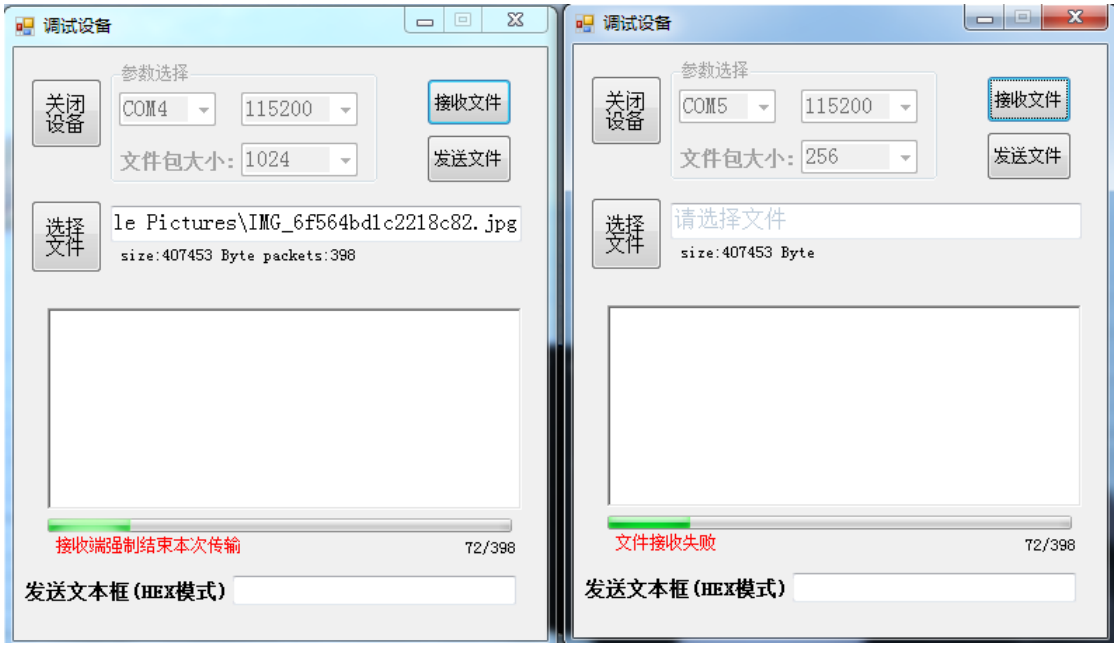
准备进入接收模式的时候，发送一帧数据告诉对方已准备好接收(当然也是可以不发该帧数据,让发送方主动发送就可以了)。该协议默认通知发送方已准备好接收数据。

帧格式(3 字节)： 0XAA 0XBB 0XDD

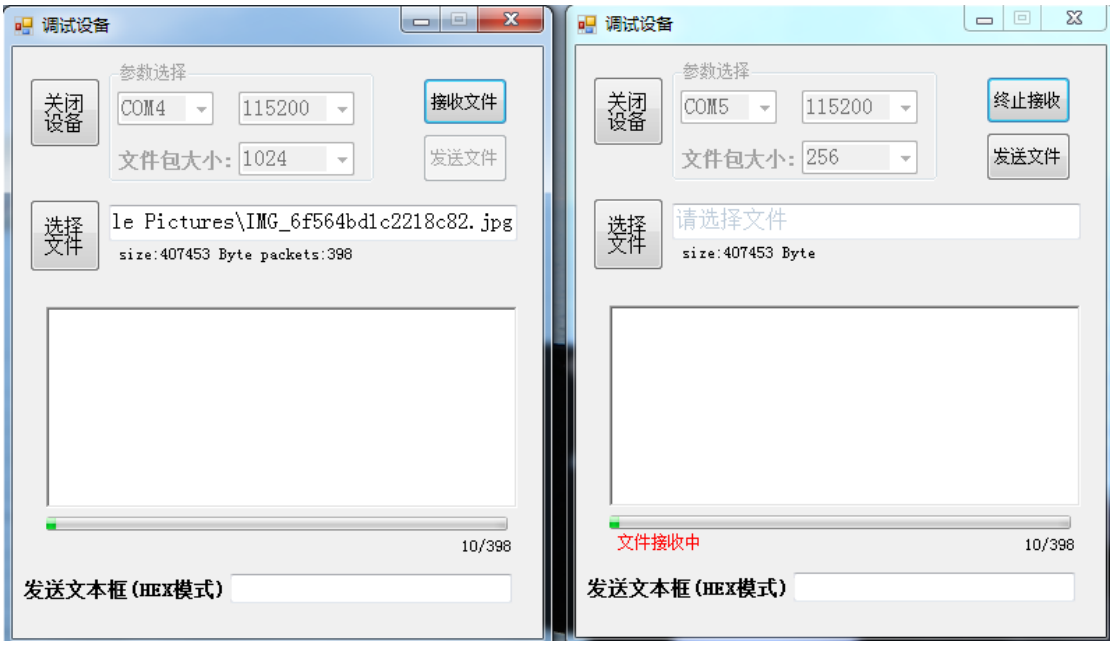
文件传输无响应状态：



文件传输被中断状态：



文件传输中：



文件传输完成：