100071011 Computer Networks 2022-2023-2

Project-1

Reliable File Transfer using Go-Back-N protocol

Specification

学号 (Student ID)	**********
姓名 (Name)	Yang Chang
班号 (Class No.)	*******
授课教师 (Instructor)	***

School of Computer
Beijing Institute of Technology
April 12, 2023

1. Requirement Analysis

分析实验要求，可以看到我们需要自己设计一个GBN协议的UDP发送算法，来应对UDP帧在传输过程中出错、丢失、失序等潜在的异常情况。GBN协议采用滑动窗口和超时重传机制，确保接收方正确接收到发送方发送的所有帧。当发送的帧出现错误时，接收方通过校验和检测出错误，发送方会重传窗口内的所有帧，直到所有帧被成功接收。当帧丢失时，接收方会向发送方发送一个带有最后一个成功接收帧编号的确认帧，发送方在接收到确认帧后，会重传窗口内的所有帧，以确保所有帧被成功接收。当帧失序时，发送方通过窗口大小机制等待接收到确认帧后才会发送下一个帧，并根据确认帧中的帧编号重新发送相应的帧。因此，GBN协议能够确保UDP帧的可靠传输，有效地处理了UDP帧在传输过程中出现的异常情况。

事实上，设计一个基于GBN协议的UDP发送算法并不复杂，但在本项目中我们还要使其支持全双工传输，并且实现传输配套服务，如信道基本参数设置、文件读取，帧校验，以及日志记录与生成等。关于这些，我将会在下文中详细阐述。

2. Design

2.1项目总体设计

考虑到对最基本的 UDPSocket 的库支持以及其他文件读取、字符串处理等底层函数的支持，在本次项目中我将基于 Qt 开发工具使用 C++构建一个具有图形界面的 UDP 文件传输工具，实现图形化的参数设置、参数载入、文件选择与发送/接收帧信息的实时显示。

2.2 UDP Socket 端口组合设计

在本项目中，全双工的文件传输体现在两个站点间可以同时互相发送文件。则各站点在发送数据的同时仍能够接收数据，因此在本项目中各站点应当维护两个 UDP Socket，两者分别处理数据的发送和接收。其中发送端口由一个定时器激活，使站点在发送状态下定时发送待发送数据或确认帧；接收端口则随时监听绑定端口，在收到数据后立即进行分析处理。我们可以将这两个 UDP Socket 组合起来看作一个支持全双工传输的抽象端口 Abstract Socket。

Fig 1 端口组合设计

2.3 UDP 帧结构设计与捎带确认

帧结构的设计是全双工同讯实现的关键。在两个站点同时互相发送文件时，每个站点不仅要发送数据帧，还需要发送确认帧。若设计两种不同的帧结构，不仅极大增加接收部分的复杂性，还会破坏帧序，降低发送效率。因此，在本项目中我采用一种帧结构包含数据和确

认信息的设计，用捎带确认思想返回确认信息，在帧中预留确认字段，需要返回确认时，将确认信息装载入帧，随下一次帧发送时发出。

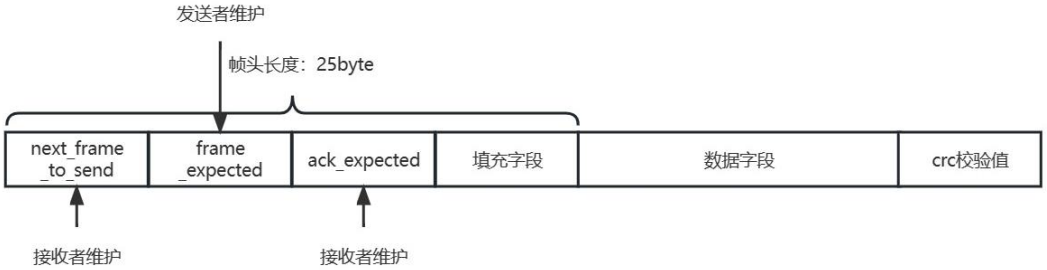


Fig 2 UDP帧结构

2.4 滑动窗口与窗口缓存设计

滑动窗口是 GBN 协议的核心。在教学中，窗口滑动时，各窗口内的内容都会向前移动一个窗口。为避免实际状况下额外的数据移动，我以“窗口固定，地址滑动”的思路，在项目中采用循环队列构建一个二维字节数组作为窗口缓存，维护一个字节数组指针指向逻辑头窗口的实际地址，在逻辑上发生“窗口滑动”时，“滑动”这一头指针。

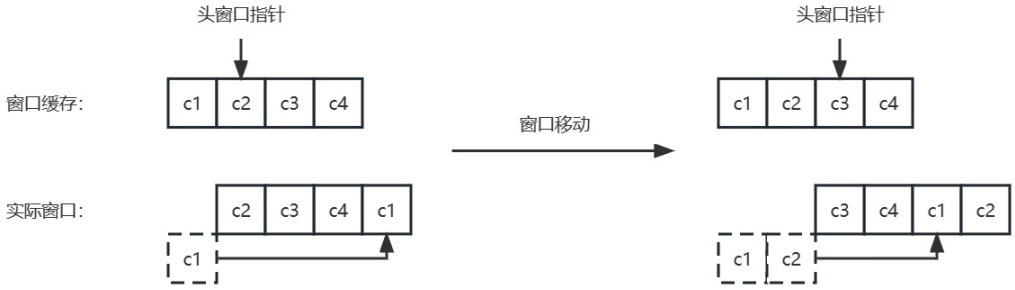


Fig 3 滑动窗口与窗口缓存

2.5 超时重传设计

超时重传是 GBN 协议处理发送异常的手段。对于每个被发送者发出的帧，发送者都会等待接收者返回确认帧。如果一定时间后发送者还未接收到该帧对应的确认帧，则发送者认为该帧未被正确接收，并重新发送该帧之后的所有帧。在本项目中，当发送任务建立后，发送者会激活一个超时重传专用的定时器，每经过固定时间便检测是否收到期望的确认帧，并根据结果重传或继续发送。

2.6 潜在错误与识别

由于 UDP 协议是无连接服务，在使用 UDP 协议进行传输的时候会存在多种潜在错误，如帧丢失、帧失序、帧错误等。对于帧丢失和帧失序，我们可以通过帧序号进行判断，对于帧错误，我们可以通过 CRC 校验实现，在发送时计算帧的校验值，接受后可通过校验值判断是否出错。同时，为模拟出错情况，本项目可根据预设参数主动产生错误情况。

2.7 CRC 校验

对于本项目中的帧校验，我使用 CRC-CCITT (Kermit)标准按字节计算校验值。发送前，在组帧时计算得到校验值并在帧末尾加入校验值；接收时，取出帧末尾的校验值，计算前部的校验值并与其进行比对，若校验正确，则对接收帧作进一步的处理，否则抛弃该帧。

2.8 UDP 与信道参数设置

本项目所构建的系统应当具有灵活性，应当能够自由调整参数。出于操作的简便，在本项目中我实现了图形化界面的参数设置方式和导入配置文件的参数设置方式。

3. Development and Implementation

本项目是在Windows 10平台使用Qt IDE开发的图形化C++程序，采用面向对象的设计方法，使用Git 2.38.1进行版本管理，项目已在Gitee开源。

项目大部分的代码都是围绕核心的GBN传输协议的外围功能支持和图形界面支持，这类代码基本只是业务和功能作用，没有太大需要解释的必要，该部分将略去，我将针对实现核心功能的类和函数详细展开说明。

3.1 使用的 Qt 官方库

3.1.1 <QMainWindow>

这是 Qt 官方的窗口界面类，本项目的图形界面依托于此建立。

3.1.2 <QUdpSocket>

这是 Qt 官方封装好的 UDP 端口操作类，其发送与监听函数是本项目发送与接收函数的基础。在本项目中使用此类实例化了发送与接收端口。

3.1.3 <QTimer>

这是 Qt 官方的定时器类，在本项目中使用此类实例化了发送定时器和超时定时器。

3.1.4 <QFile>

这是 Qt 官方的文件类，可将文件作为一个实例化的对象进行操作。本项目对文件的读写操作基于此类。

3.1.5 <QThread>

这是 Qt 官方的线程类，在本项目中使用此类进行基本的线程控制。

3.2 fileTransfer：文件传输主类

fileTransfer 类是本项目最核心的类，在其中实现了 GBN 协议与 UDP 端口的操作。由于其内部代码较为复杂，本部分将用伪代码的形式分析代码功能与结构。以下为该类中的关键函数。

3.2.1 开始传输文件：on_pushButton1Send_clicked()

该函数由“发送文件”按钮调用，首先发送一个包含文件信息的第一帧告知对方开始接收文件，然后激活发送定时器和超时定时器，为后续的数据发送做准备。

```
1. void on_pushButton1Send_clicked()
2. {
3.     组成第一帧
4.     发送第一帧
5.     激活发送定时器
6.     激活超时定时器
7.     更新状态信息
8. }
```

3.2.2 帧的发送：timerSend_triggered()

该函数由定时器 timerSend()调用，以实现帧的不断发送。定时器在发送按钮被按下时激活，每 20ms 触发一次 timerSend_triggered()函数。该函数结构如下

```
1. void timerSend_triggered()
2. {
3.     isReciveing? -> 设置帧头
4.     isSending? -> 设置帧头
5.                -> 从窗口缓存装载数据
6.     组帧
7.     sendSocket->send()
8.     更新状态数据
9. }
```

函数被调用时，将首先根据本地站点所担任的角色（发送者、接收者）来组成帧头，若担任发送者，首先根据现有的返回确认信息滑动窗口或重传数据，然后将数据从窗口缓存装入帧。通过调用 UDPFrame 类来组成最后发送的帧后，使用 sendSocket 将帧发出。在以上动作完成后更新状态数据

3.2.3 帧的接收：on_receiveSocket_readyRead()

该函数由接收端口调用，用于对原始数据进行正确性检验，为后续的分析做准备。

```
1. void on_receiveSocket_readyRead()
2. {
3.     读取接收数据
4.     check_crc()
5.     错误 -> 丢弃
6.     正确 -> 解析帧
7.     更新状态数据
8. }
```

3.2.4 帧的分析：analyseReceive()

该函数是程序最主要的数据分析与接收函数，被上述帧接收函数在校验通过后调用。

```
1. void analyseReceive()
2. {
3.     拆分字符串成 UDPFrame 类
4.
5.     是否是第一帧?
6.     是->激活发送定时器，开始发送确认帧
7.         ->建立本地文件并打开
8.         ->isReceiving = true
9.     isReceiving?
10.    是-> 数据写入本地文件
11.    isSending?
12.    是->更新 ack 信息
13.
14.    更新状态信息
15.}
```

3.3 UDPFrame：UDP 帧类

UDPFrame 类是本项目对构建与处理数据帧的方法的封装。其关键函数如下：

3.3.1 组帧：framing()

该函数根据设定参数计算 crc 校验值，并整合帧头、数据字段、crc 校验值，生成可被 UDP 端口直接发送的二进制串。

```
1. void UDPFrame::framing()
2. {
3.     arrSend.clear();
4.     arrSend.resize(buffer.size()+50);
5.     QString head = QString("%1##%2##%3##").arg(next_frame_to_send).arg(frame_e
   xpected).arg(ack_expected);
6.     arrSend = head.toLocal8Bit();
7.     for(int i = arrSend.size();i < 25;i++)   arrSend.append('$');
8.     arrSend.append(buffer);
9.     uint8_t temp;
10.    crc = crc16_CCITT(arrSend.data(),arrSend.size());
11.    temp = crc >> 8;
12.    arrSend.append(temp);
13.    temp = crc & 0xff;
14.    arrSend.append(temp);
15.    totalLen = arrSend.size();
16.}
```

3.3.2 分解帧：getReceive(QByteArray arr)

该函数是为接收者设计，将接收到的二进制串分解为数据字段和关键参数，以便于后续处理。

```
1. void UDPFrame::getReceive(QByteArray arr)
2. {
3.     buffer.clear();
4.     arrSend.clear();
5.     QByteArray head;
6.     for(int i = 0;i < 25;i++)   head[i] = arr[i];
7.     next_frame_to_send = QString(head).section("##", 0, 0).toInt();
8.     frame_expected = QString(head).section("##", 1, 1).toInt();
9.     ack_expected = QString(head).section("##", 2, 2).toInt();
10.    totalLen = arr.size() - 27;
11.    for(int i = 25;i < arr.size() - 2;i++)
12.    {
13.        buffer[i-25] = arr[i];
14.        arrSend[i - 25] = arr[i];
15.    }
16.}
```

3.4 其他

3.4.1 CRC 校验函数：

该函数基于 CRC-CCITT (Kermit)，按字节计算对应字符串的 CRC 校验值。由于其使用了预计算的 CCITT 表，因此该方法的 CRC 计算速度很快。

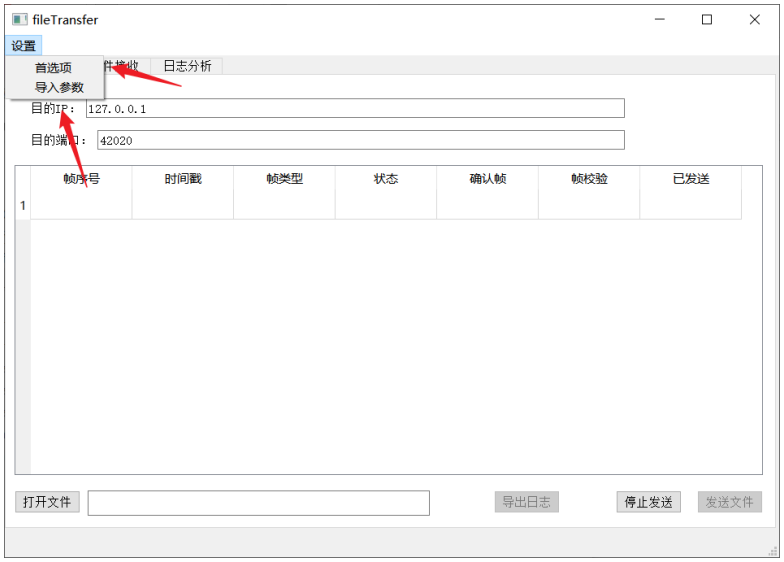
```
uint16_t crc16_CCITT(uint8_t* datas,uint16_t len){
    uint16_t crc_value = 0;
    while (len--){
        crc_value = (crc_value >> 8) ^ crc16_table_CCITT[(crc_value ^ *datas++) & 0xff];
    }
    return crc_value;
}
```

4. System Deployment, Startup, and Use

程序源码在Qt 5.12.9上开发，请在Qt 5.12及以上版本下编译。

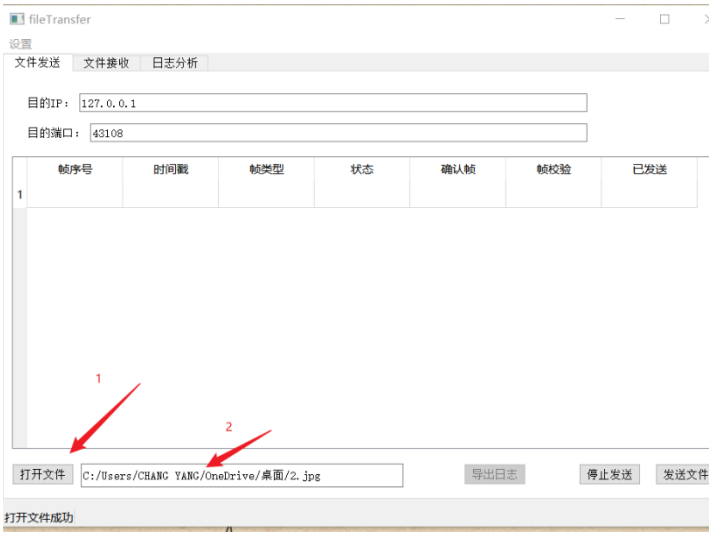
程序已编译完成，无需额外的环境依赖，将文件夹放在英文路径下，直接双击exe文件夹的.exe文件即可运行。请注意，双击后会出现两个重叠的窗口，请先移动窗口以看到第二个窗口。

4.1 设置参数



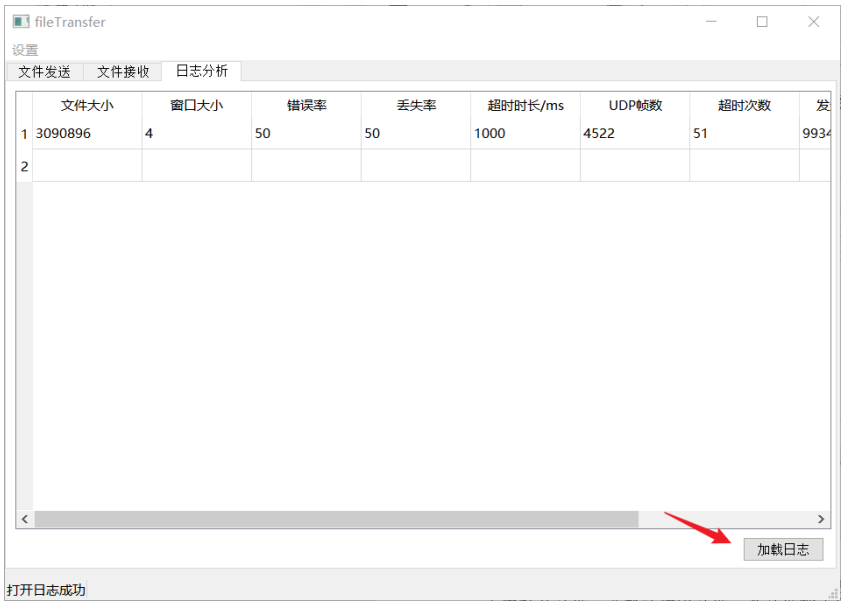
点击菜单栏的设置，可通过“首选项”用图形化的方式设置参数，也可使用导入参数选择配置文件载入。

4.2 传输文件



点击打开文件，选择待传输文件。若文件被正确读取，发送文件按钮将可被点击，方框将出现文件的绝对路径。在传输完成后，导出日志按钮将可被点击，日志会自动保存到./LOG 文件夹。

4.3 载入日志



进入日志分析页面，点击加载日志按钮，选择要载入的日志文件，将显示被处理过的日志信息。

5. System Test

针对本项目，我将就文件的全双工传输功能和日志加载功能进行测试。

5.1 3Mb 文件双向同时传输测试

在本测试中，我将从两个站点同时向对方传输 3Mb 左右的文件，观察软件能否正确传输文件、能否正确处理帧错误。

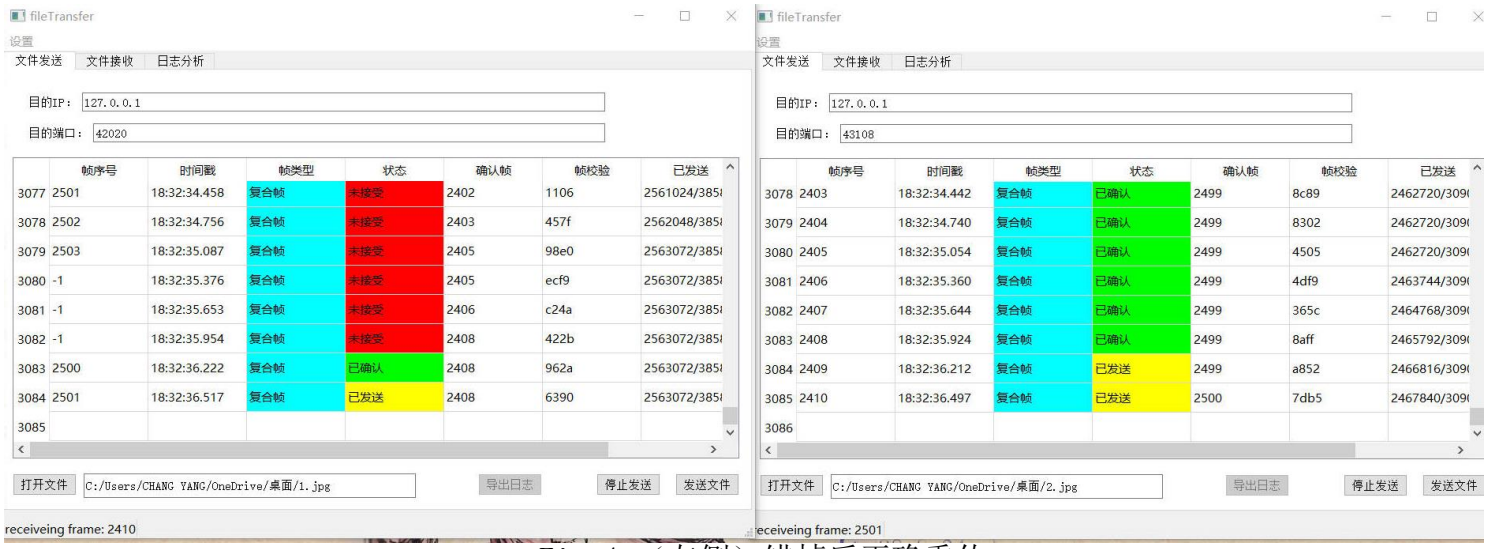


Fig 4 （左侧）错帧后正确重传

观察左侧窗口，可以看到当前时刻发生了重传事件，对比序号，可以看出 GBN 协议被正确实施。

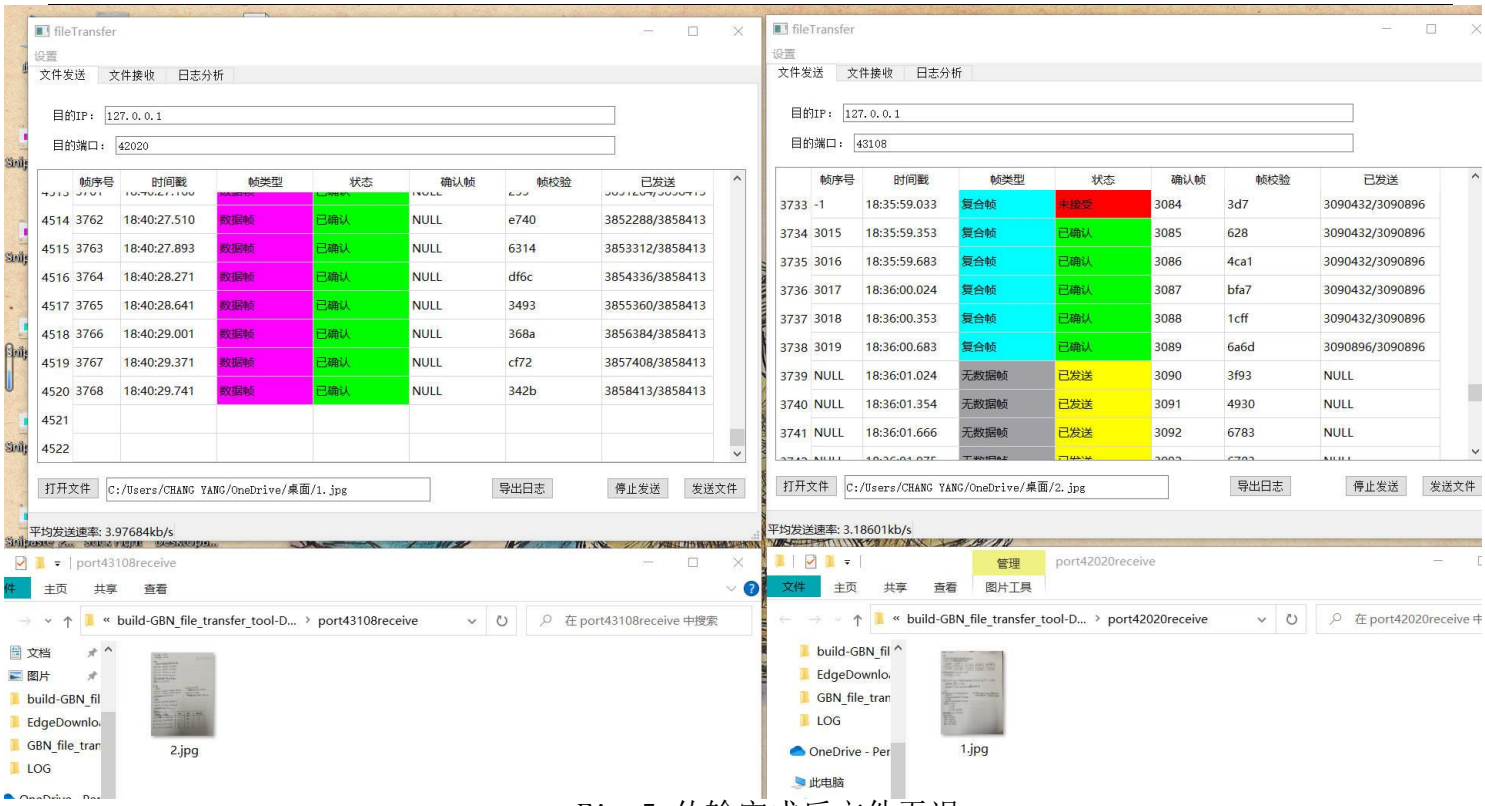


Fig 5 传输完成后文件无误

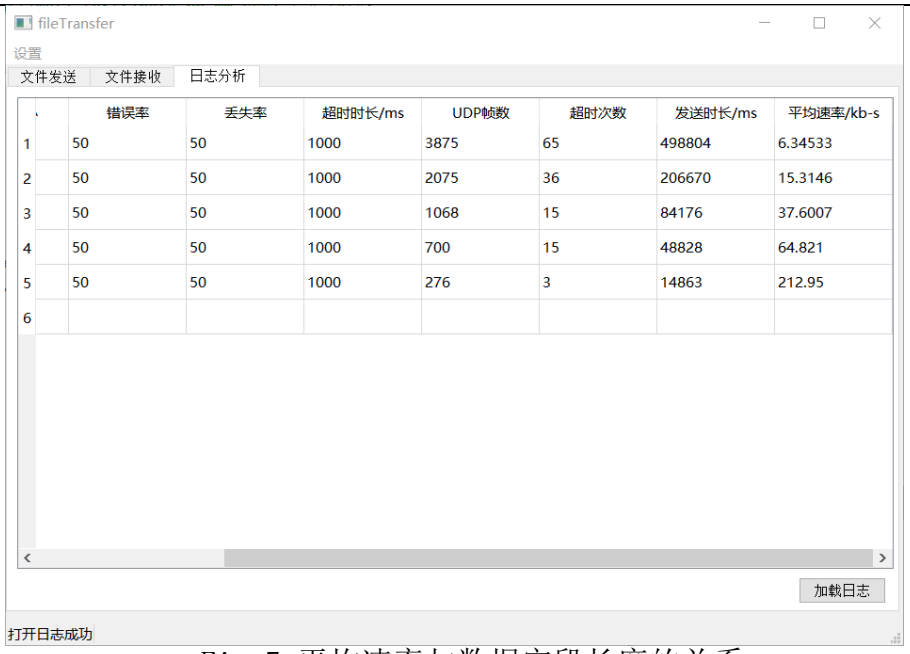
在文件传输完成后打开对应文件夹，可以看到文件完整无误。

5.2 日志加载测试

Fig 6 日志加载正常

6. Performance and Analysis

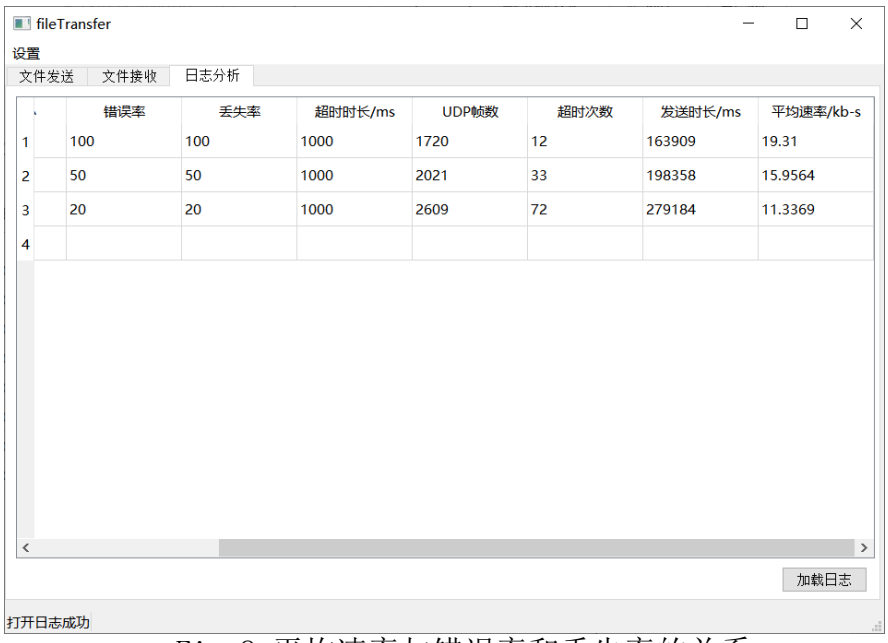
在本项目中，文件发送时可通过已发文件大小和发送时长得到平均速率，结合发送过程中的超时次数，可以较为完备地分析当前参数的发送效率。



	错误率	丢失率	超时时长/ms	UDP帧数	超时次数	发送时长/ms	平均速率/kb-s
1	50	50	1000	3875	65	498804	6.34533
2	50	50	1000	2075	36	206670	15.3146
3	50	50	1000	1068	15	84176	37.6007
4	50	50	1000	700	15	48828	64.821
5	50	50	1000	276	3	14863	212.95
6							

Fig 7 平均速率与数据字段长度的关系

经过多次尝试，我发现在固定错误率和丢失率的情况下，平均发送速率受帧数据大小的影响较大。如上文图6中的第四个日志发送速度是在帧数据大小设为4k的时候得到的，与之相比前三项的速率是在帧数据大小设为1k的时候得到的。而随着文件变大，平均传输速率逐渐降低。



	错误率	丢失率	超时时长/ms	UDP帧数	超时次数	发送时长/ms	平均速率/kb-s
1	100	100	1000	1720	12	163909	19.31
2	50	50	1000	2021	33	198358	15.9564
3	20	20	1000	2609	72	279184	11.3369
4							

Fig 8 平均速率与错误率和丢失率的关系

当帧数据大小固定时，错误率和丢失率会使超时重发数线性变化，继而影响传输速率。

以上结论不难理解。在发送过程中，帧尺寸基本固定，则发送速率取决于有效帧的发送频率。而随着文件的增大，丢帧和错帧情况越来越多，使得平均发送速率逐渐降低。

其次，在本项目中，为防止粘包，我设定的帧发送频率为20ms发送一次，但这个参数其实非常保守，可结合帧数据大小减小发送间隔，可有效提升发送速率。

7. Summary or Conclusions

在本项目中，我采用GBN协议构建了一个可靠的UDP传输算法，并在此基础上完成了一个图形化的文件传输工具的构建。在程序编写的过程中，我深入学习和实践了UDP端口类的使用和功能，反复学习并理解了滑动窗口的设计理念，自己设计数据结构实现了GBN协议。在不断的运行和测试中，可以发现这个程序的发送速率比起实际网络使用时的速率仍有不少差异，这也说明了不同参数对网络系统会产生重大的影响，理论到实际过程中仍需要大量的实践。

8. References

[1].本项目仓库地址：https://gitee.com/Yang__Chang/UDP-file-transfer-tool

9. Comments

虽然熬了几个大夜，但能看到一套完整的程序跑起来还是非常开心的。报告写得略微仓促，未尽之处还请老师谅解。我在项目程序的人机交互上花了很多心思构建，老师若有兴趣可以体验一下。