实验 3-2 基于UDP设计可靠传输协议并实现

一、协议设计

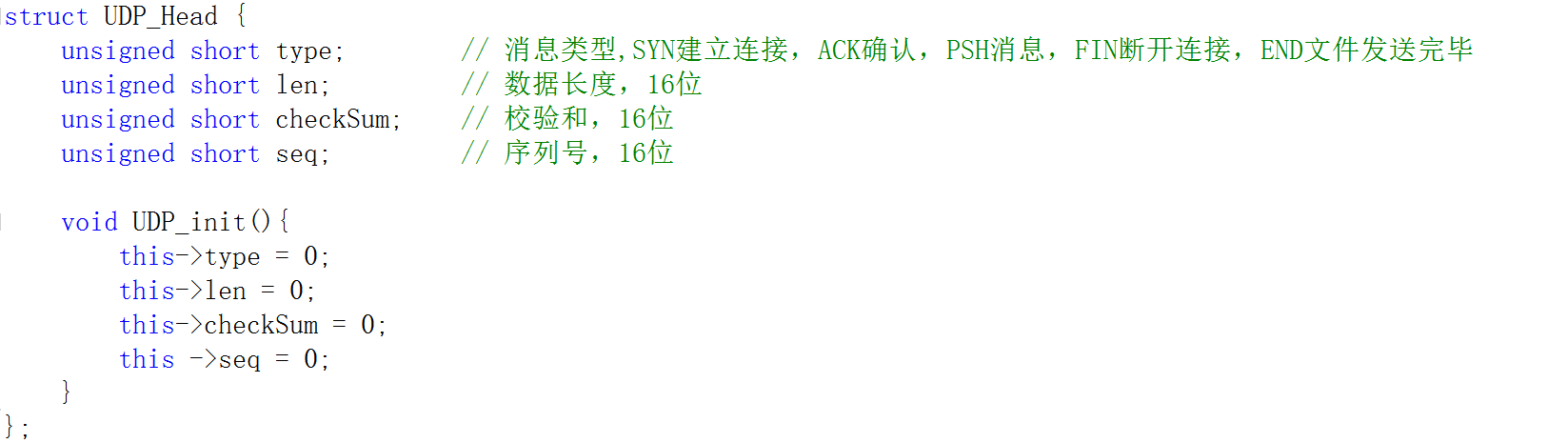
1. 规定客户端和服务器端的报文格式

服务器和客户端为了便于交流，我们首先规定传输消息的报文格式。报文主要包括消息头和消息主体两部分。最终，发送消息缓冲区sendBuf和接收消息缓冲区recvBuf，它们的类型都为char 类型的字符数组。

1. UDP传输消息的消息头：设置消息头为一个struct结构体，每个成员变量都是16位。 type表示消息类型，用define宏定义了SYN建立连接，ACK确认，PSH消息，FIN断开连接，END文件发送完毕，SYN\_ACK表示服务器发来的连接确认，FIN\_ACK表示服务器发来的断开连接确认，END\_ACK表示服务器发来的文件传输完毕的确认。

len表示消息主体的字节数量，我们规定传输消息主体最大字节数为10240，也就是10KB。

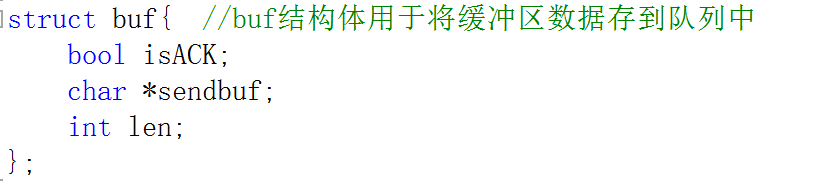
chechSum是校验和，发送时先计算一遍校验和并取反，并存入消息头的这个成员变量中，接收方收到后，再计算一遍校验和，取反后如果为全0，则说明消息无误。之后在程序设计中详细介绍校验和函数。seq表示发送端的序列号，在接收端也要进行校验。



1. UDP传输消息的消息主体：文件传输的消息为char类型，我们规定如果消息字节数超过10240，那么分段发送，最大发送10240字节数，没有超过则发送现有字节数。

2、滑动窗口中的队列中存储元素的格式

滑动窗口设置最大值为5，在发送缓冲区中滑动，存的是已发送但滑动窗口后沿还未被确认的消息，这里利用队列进行实现，队列的元素中存的是一种struct类型，定义如下：



其中，sendbuf用来存储已经发送的消息的内容，是char类型的字符数组，len存的是消息的长度（单位：字节），isACK用来看滑动窗口中的该元素是否被确认。

3、客户端（发送端）和服务器（接收端）

使用了课程中讲授的GO-BACK-N(GBN)协议进行流水线的设计，在之前rdt3.0设计的建立连接、发送消息时的序列号检验、校验和检验、以及丢包时的超时重传的功能的基础上进行改进：在客户端（发送端）有一个可变的滑动窗口，在最大窗口限制的范围内可以流水线的发送消息，并且在滑动窗口范围内都可以接收来自服务器（接收端）的ACK消息；服务器（接收端）采用累积确认的方式接收并发送ACK消息，且接收端的滑动窗口大小为1。

（1）建立连接：客户端（发送端）先发送SYN信号的消息给服务器（接收端），服务器再发送SYN\_ACK信号给客户端，如果两边都接收到消息并且序列号和校验和都正确，则连接成功。

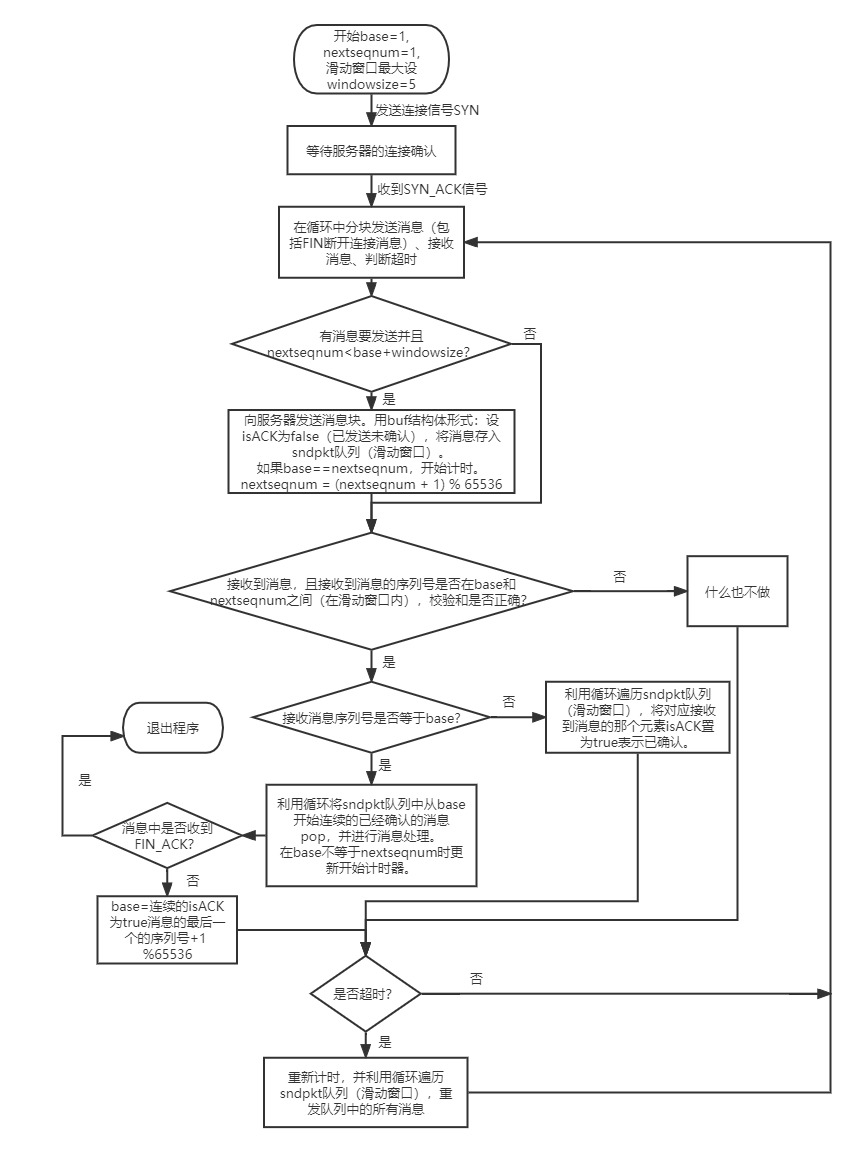
（2）客户端发送消息：一个大文件分成多个块循环发送消息，在一个循环中利用判断条件区分发送消息、接收消息、超时重传消息。

发送消息：要先if判断一下在不在要发送的序列号在不在最大滑动窗口的范围内中，如果在就发送消息，并将消息压入队列（队列表示当前滑动窗口），该消息的isACK设false，表示这里保存的是已发送但未被确认的消息。如果滑动窗口已经最大了就不能发送消息了。如果滑动窗口为0，则开始计时。之后滑动窗口前沿nextseqnum+1。

接收消息：要先if判断接收到消息的长度是否大于零，大于零则进入处理接收消息。进入后将消息头取出，检查校验和是否正确、接收到的序列号是否在滑动窗口中，如果都符合条件，又要分成两种情况，一种是序列号不等于base（滑动窗口后沿，等待接收消息后移动窗口），利用一个循环遍历并重插一遍队列中的元素，同时将收到那个序列号的消息的isACK标记为true，表示已经接收确认消息。一种是序列号等于base，则循环将队列中从base开始连续的已经确认的消息全部pop出来，更新base=连续已经确认的最后一个消息的序列号+1。

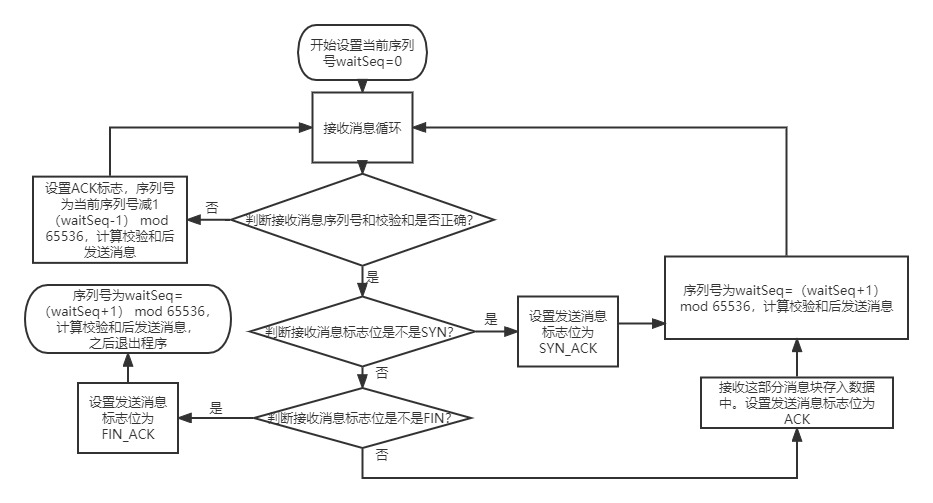
如果遇到超时情况，则重新计时，并将队列（滑动窗口）中的所有元素全部重传一遍。

我们看到发送端流程图如图所示：



（3）服务器接收消息：和之前停等协议基本一致。接收消息首先检验序列号是不是和等待的序列号相等，并且检验校验和是否为0。如果序列号或者校验和有误，那么发送ACK消息，并且消息的序列号为等待的序列号-1(mod 65536)。如果没有错误，进入类型判断。如果收到SYN，说明要建立连接，回SYN\_ACK标志消息，如果收到FIN，说明要断开连接，回FIN\_ACK标志消息，并退出程序，如果都不是以上两个，则说明接收到文件信息，将消息主体部分存入要写入文件的数据中，并且发送ACK标志消息表示确认收到消息。

接收端流程图如下：

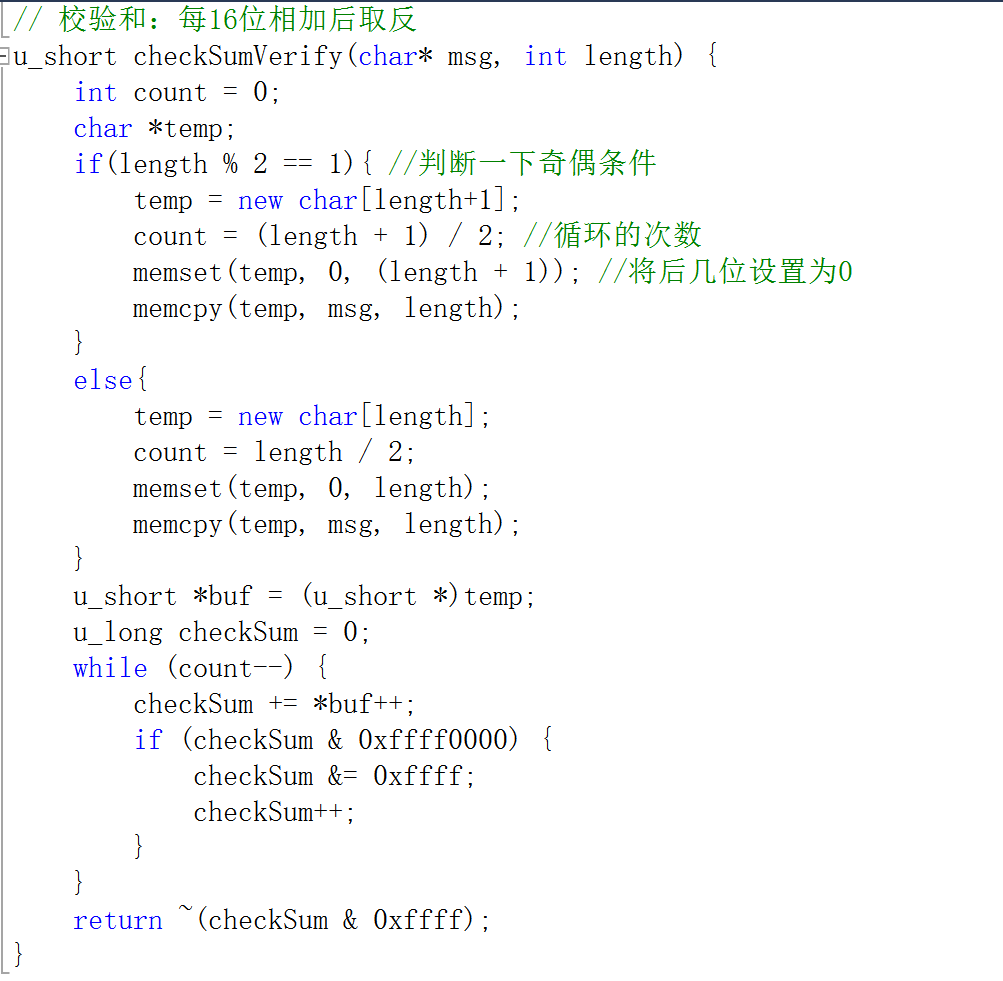


（4）断开连接：断开连接只需要两次挥手过程。客户端先向服务端发送FIN标志的消息，服务端收到后，发送FIN\_ACK标志的消息，则断开连接。

二、程序设计与实现

在本小节主要介绍程序中主要实现的功能。

1. 校验和函数

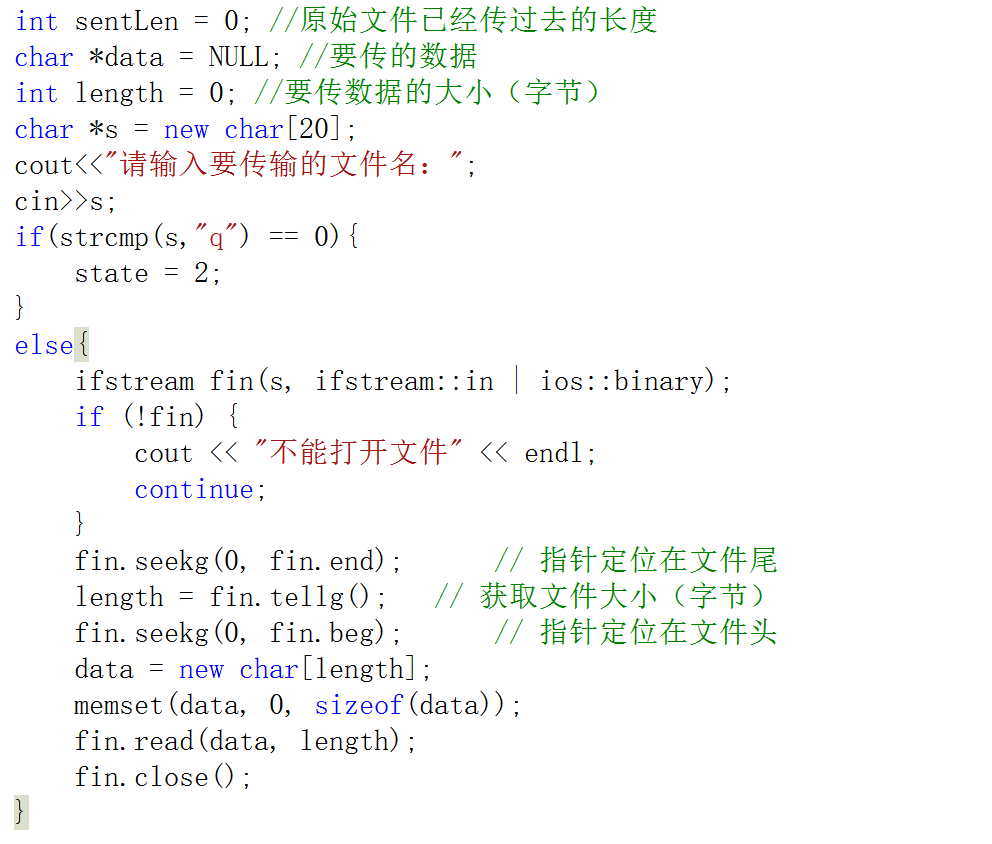


传入的参数是char 类型的消息，length是消息的字节长度。由于校验时我们是消息的每个16位数据进位相加后取反，char 类型是1个字节8位，我们要将消息分成16位的段。因此，首先要判断一下length的奇偶性，如果是奇数需要给结尾补0来保证16为格式对齐，需要先扩展一个length+1的空间，count是要循环的次数，count = (length+1)/2；如果是偶数则count直接除以2即可。用usigned long类型checkSum承接每个16位进位相加后的和，在循环中，每加16位的数据，判断一下是否超出16位，如果是则取后16位并末位+1。最后返回总和checkSum后16位并取反。

在发送端，计算校验和后存入消息头，并进行传输。在接收端，计算校验和，如果消息传输正确，那么校验和16位应该全是0。

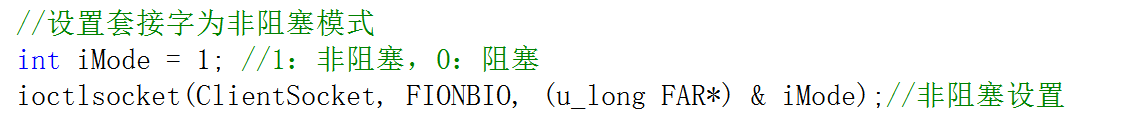
1. 客户端程序设计

客户端程序部分主要是两层嵌套的循环。第一层循环中用来输入文件名，并用二进制形式打开该文件读取数据。输入文件名之后用ifstream流fin打开并读取文件，将文件的所有数据存入data中，文件长度存入length中（单位：字节）。如果输入的文件名是’q’，表示断开连接，之后发送消息时客户端就会发送FIN标志位的消息给服务端。



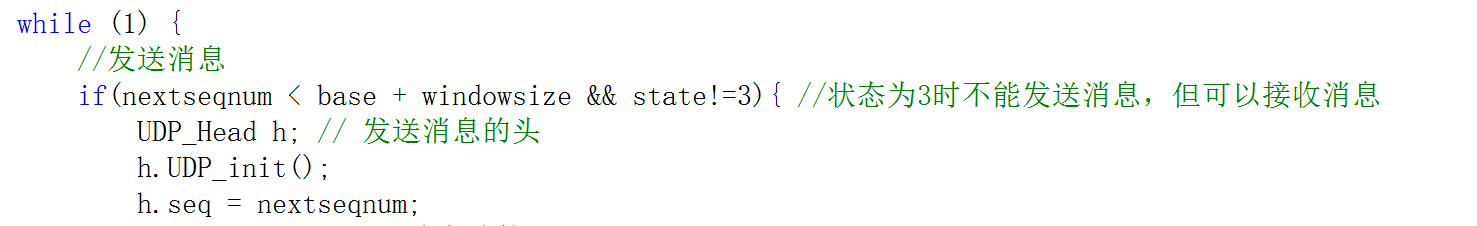
第二层循环，将文件中的数据分块进行发送，并接收ACK消息，以及超时重传。

这里由于发送消息、接收消息、超时重传在一个循环中，因此需要设置socket为非阻塞模式：



1. 发送消息部分：

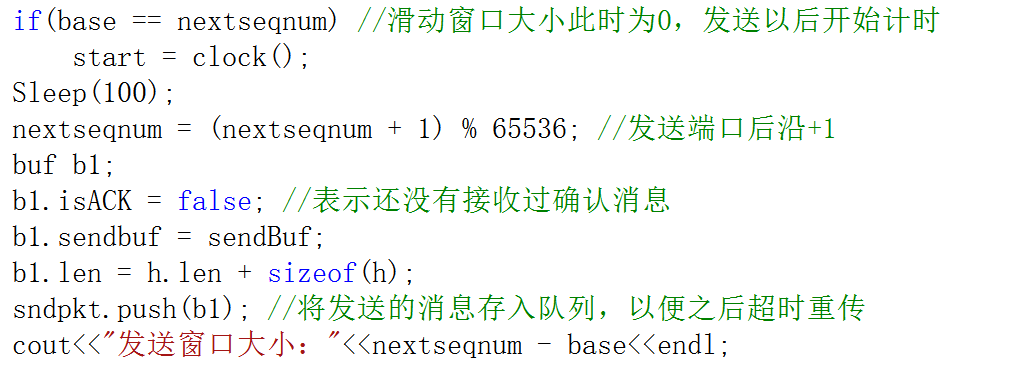
发送消息时要先if判断一下在不在要发送的序列号在不在最大滑动窗口的范围内中，如果在就发送消息。



进入发送消息部分后，如果是第一个文件会先进行自动连接，这里涉及状态变化，状态为0表示未连接，为1表示已经连接，可以分块发送消息，为2表示断开连接，为3表示一个文件已经发送完毕，此时只能进行接收ACK消息，不能再发送消息。

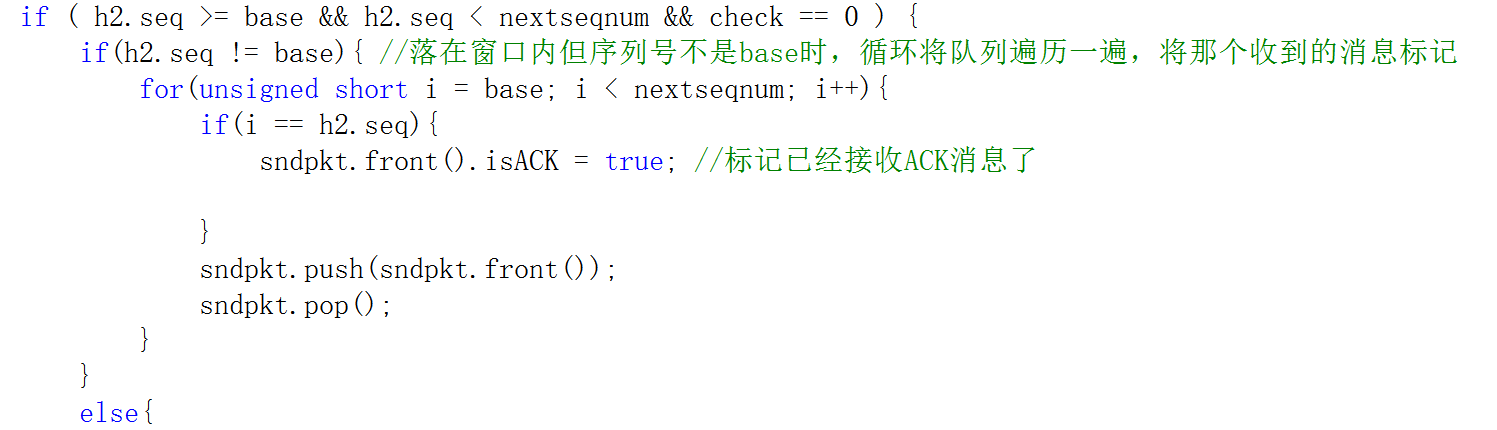


发送消息完毕后，需要将消息压入队列（队列表示当前滑动窗口），该消息的isACK设false，表示这里保存的是已发送但未被确认的消息。如果滑动窗口已经最大了就不能发送消息了。如果滑动窗口为0，则开始计时。之后滑动窗口前沿nextseqnum+1。



1. 接收消息部分：

首先检查校验和是否正确、接收到的序列号是否在滑动窗口中，如果都符合条件，又要分成两种情况：第一种情况是序列号不等于base（滑动窗口后沿，等待接收消息后移动窗口），利用一个循环遍历并重插一遍队列中的元素，同时将收到那个序列号的消息的isACK标记为true，表示已经接收确认消息。



另一种情况是序列号等于base，则循环将队列中从base开始连续的已经确认的消息全部pop出来，处理消息，并且在每次循环中base+1，并且在滑动窗口不是0的时候更新一下开始计时器。最终的base=连续已经确认的最后一个消息的序列号+1。



（3）超时重传：

超时重传的主要部分是遇到超时情况，重新计时，并将队列中所有的元素用一个while循环全部重传。

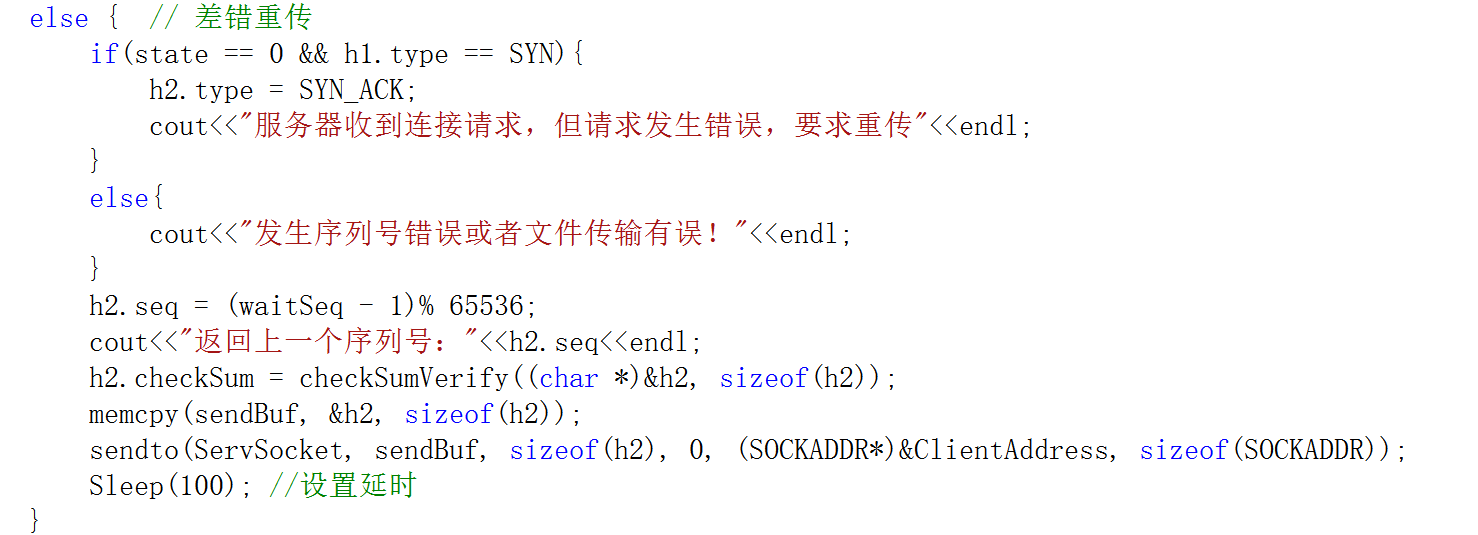


1. 服务端程序设计

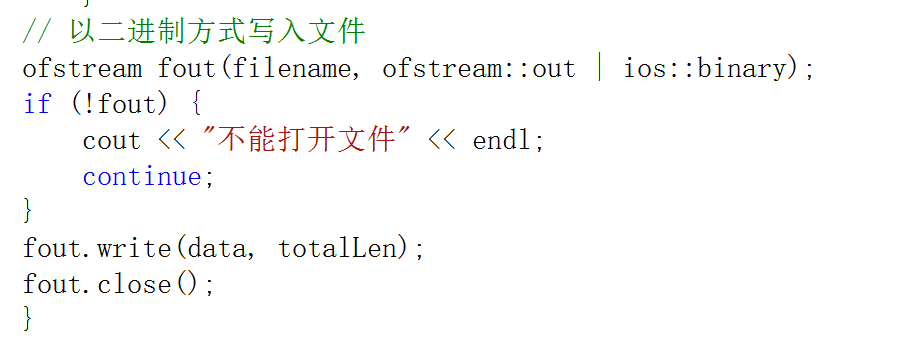
服务端主要包括两层循环，第一层循环用来接收一整个文件，具体分块接收消息需要进入第二层循环。在第二层循环中，当接收到消息时，判断一下序列号和校验和，如果都正确，则发送对应的ACK标志和当前序列号，表示这之前的序列号的消息都已经确认接收，否则发送ACK标志和前一个序列号，表示让客户端进行重传。下图是当校验和和序列号都正确时的程序：



如果校验和或者序列号不正确，则传正在等待接收序列号的前一个，表示在这个序列号以及之前的所有序列号的消息都已经收到（累积确认）。

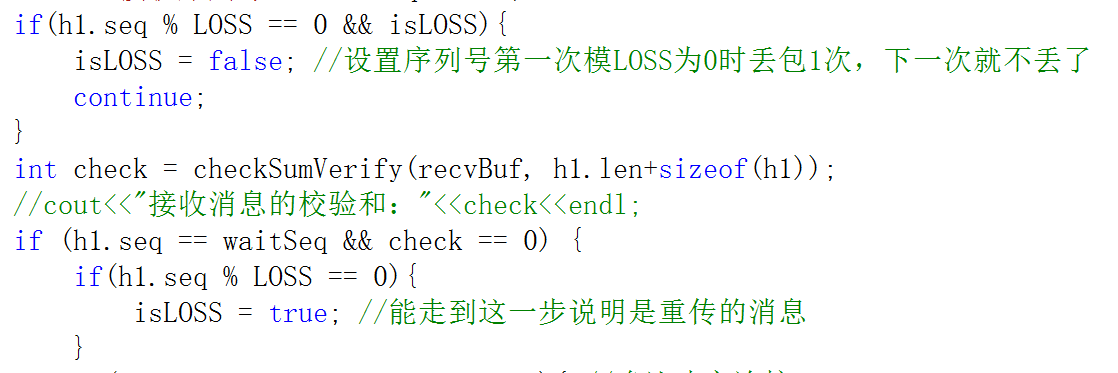


最终当收到END标志位的消息时，表示一个文件传输结束，发送END\_ACK标志位的消息给客户端后，跳出第二层循环。随着END标志位消息传来的还有文件名，接收到文件名之后，用ofstream打开并写入文件。注意是以二进制形式写入文件，也和之前的读入文件类似：

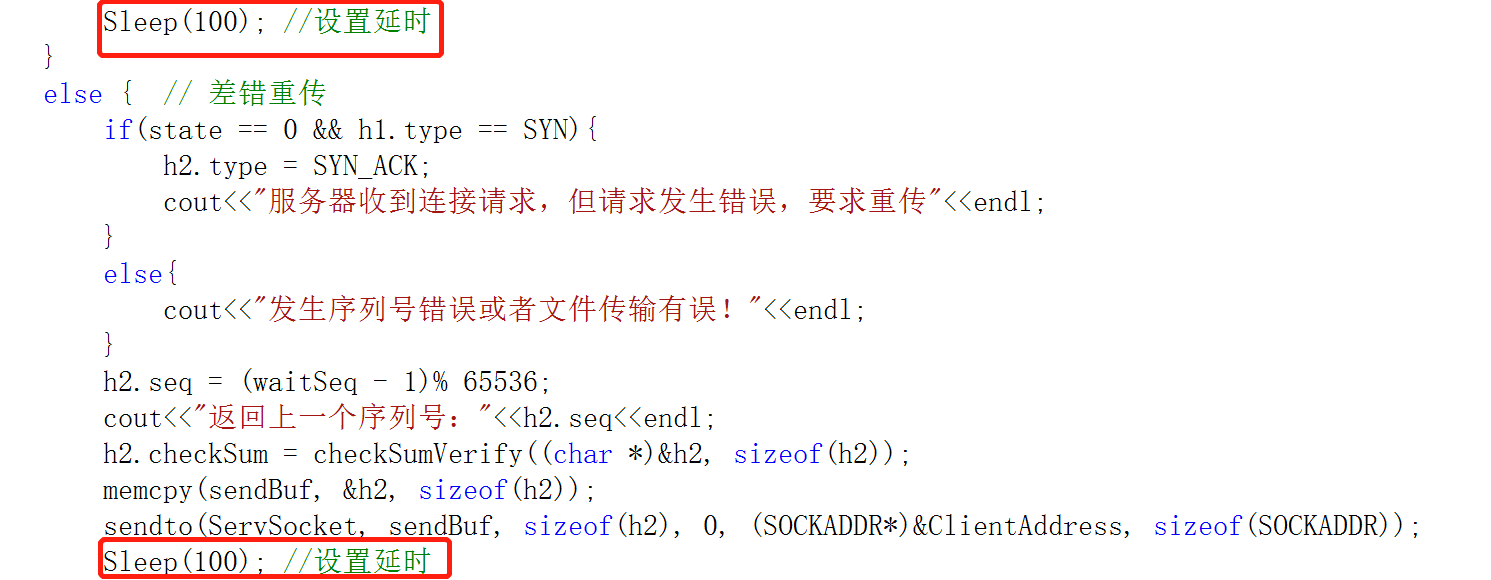


4、丢包、延时设计：

丢包在接收端设置：可以在序列号这边设置一个宏定义LOSS，这里我们先设置LOSS为100，序列号模LOSS为0时，循环comtinue，则服务器没有发送ACK消息，表示丢包。这里设置isLOSS开始为true，当丢包一次后设置为false，那么重传的这次就不会continue，也就不丢包，当检查重传的这次校验和和序列号都正确后，设置isLOSS为true，为下一次丢包做准备。



延时也在接收端：即设置在发送端发送完ACK消息后，sleep函数中的数字，为延时多少ms。

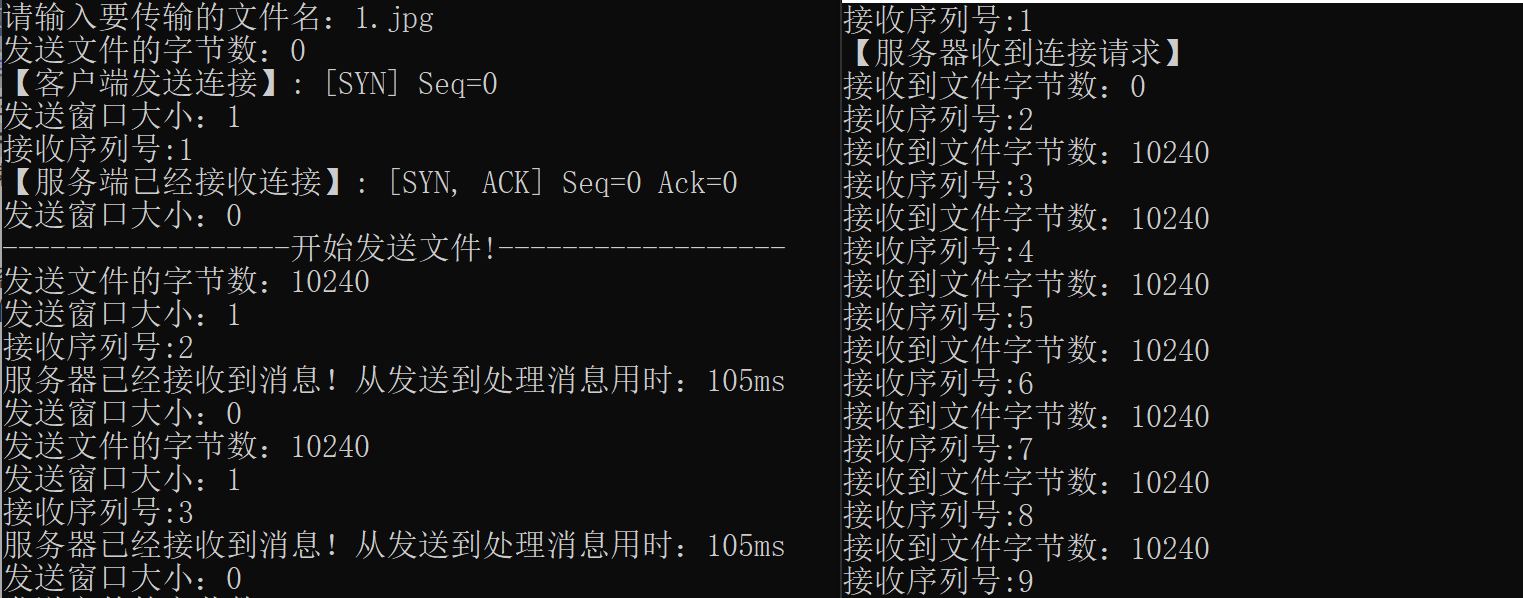


1. 使用方法和实验结果

使用方法：将要发送的文件放在客户端cpp文件同级目录下，在客户端输入要发送的文件名，之后客户端就和服务器先进行自动连接，并将文件分块的消息进行发送，在客户端这边可以看到每次发送、接收消息后的滑动窗口大小。在服务器这边进行接收客户端发送的消息，在一个文件发送完毕后，输出文件名、文件总用时、和平均吞吐率。如果输入文件名时，输入q，那么将断开连接，并退出程序。

1. 发送1.jpg

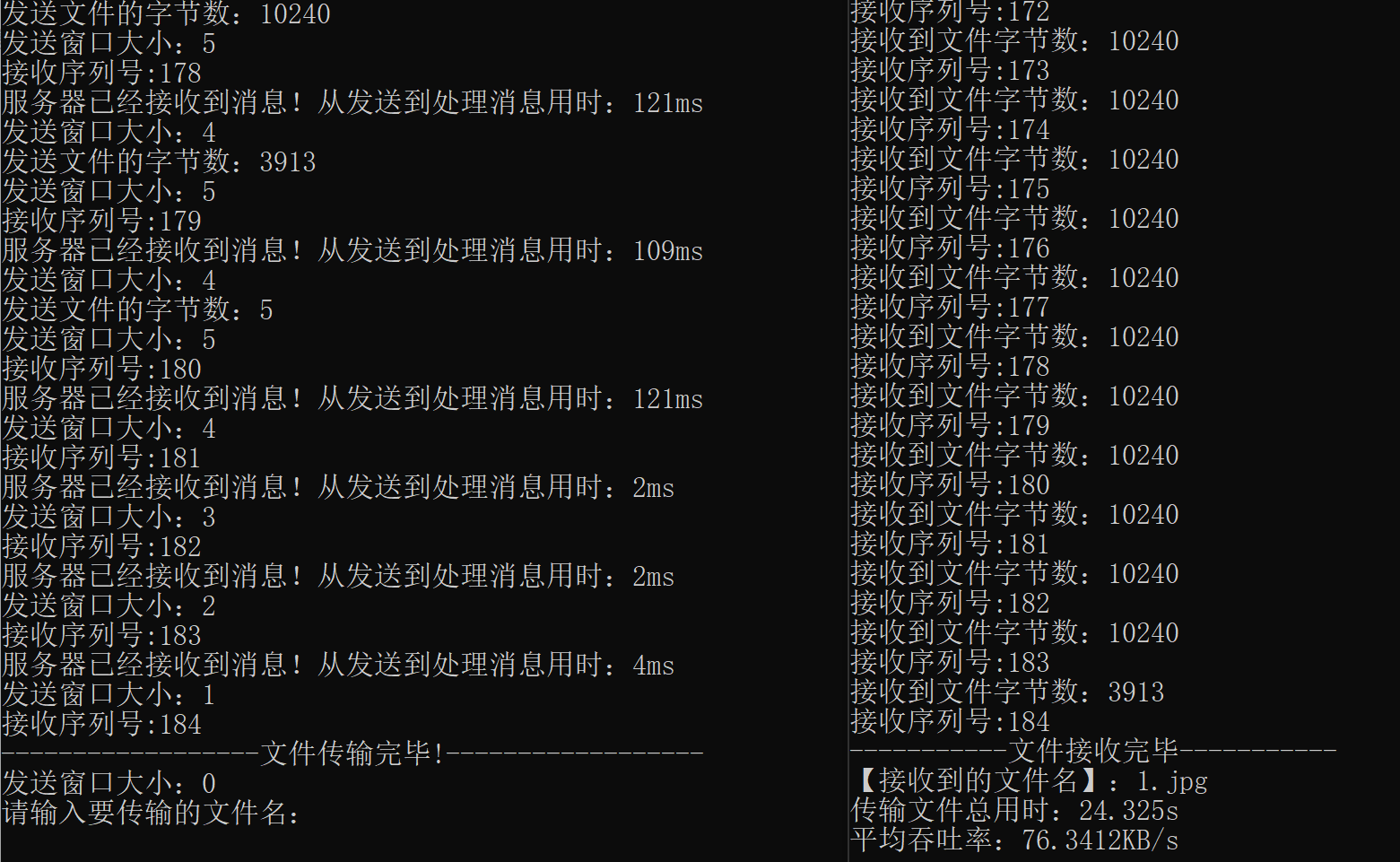
左边是客户端，右边是服务器，下图是初始自动连接的日志输出：



由于设置了LOSS为100，即接收到的消息序列号整除100时，服务器会丢包。因此看下图，左边客户端在接收到99号消息的ACK时，滑动窗口为0，发送100号消息后，服务器丢包，但客户端还在发送消息，于是滑动窗口开始增长，滑动窗口达到最大值5后，客户端不再发送消息。而右边服务器由于100号消息丢包了，之后接收到的101-104号消息都返回的ACK99，在客户端也能看到一直接收的序列号都是99。直到左边客户端超时之后，重传100-104号消息，而右边图中看到服务器之后也能够正确接收到消息。

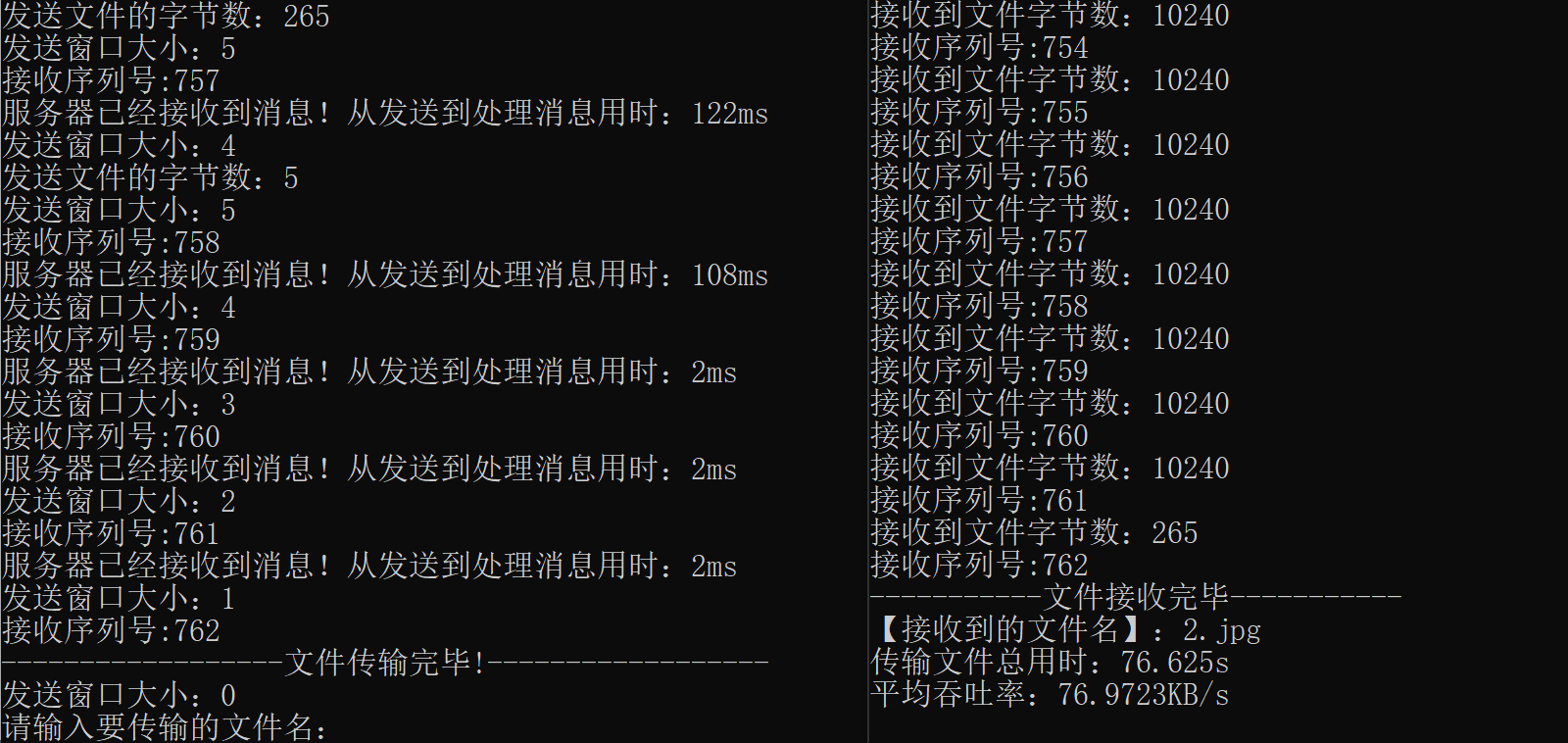


之后由于滑动窗口大小的限制，滑动窗口发送、接收消息时，窗口大小一直是4或5，直到最后文件全部发送完毕后，一直在接收消息，滑动窗口开始减小，直到最后接收ACK消息全部完毕后，滑动窗口大小为0。



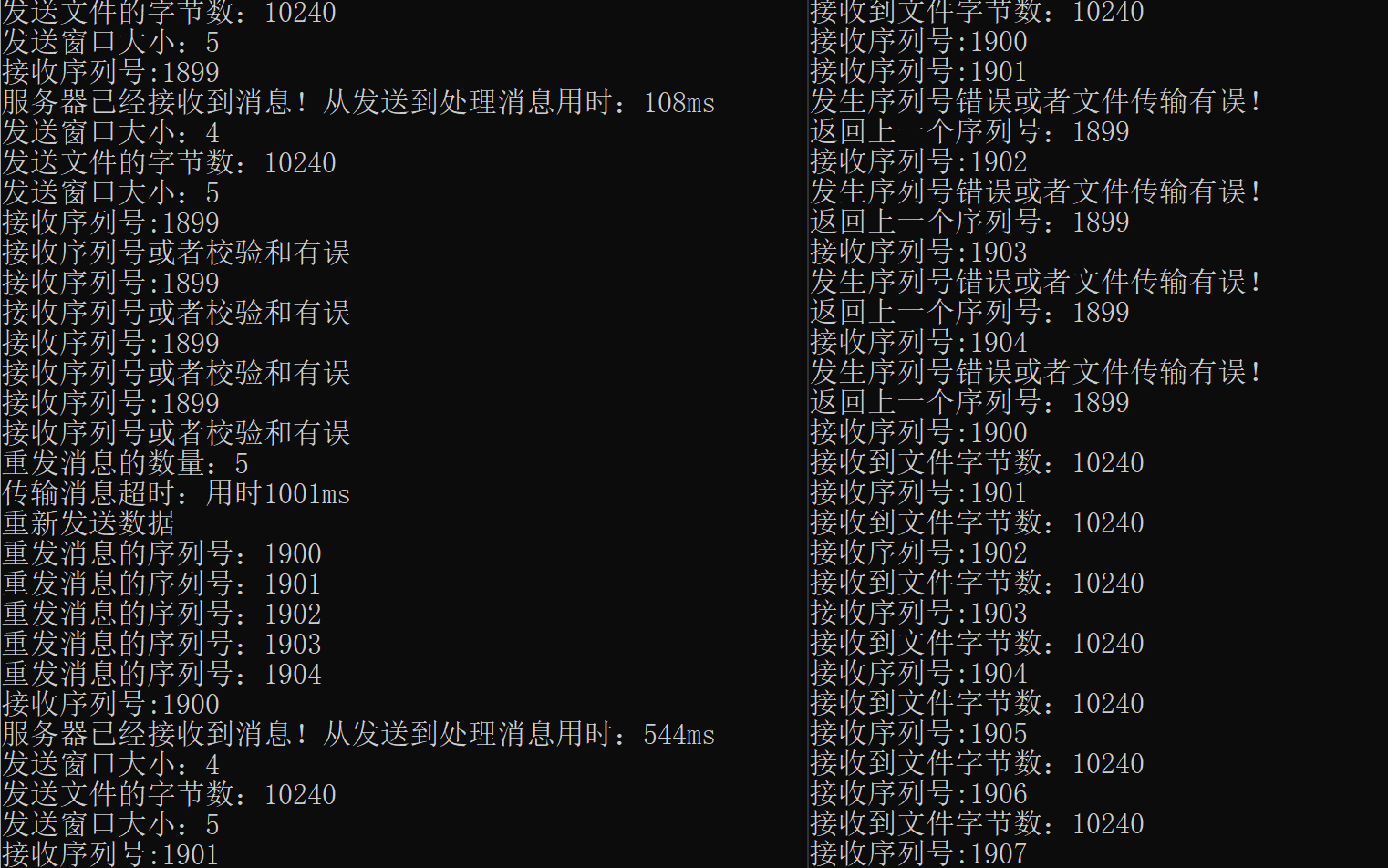
1. 发送2.jpg

发送2.jpg文件也是类似的，这里看一下最后几条消息的日志输出：

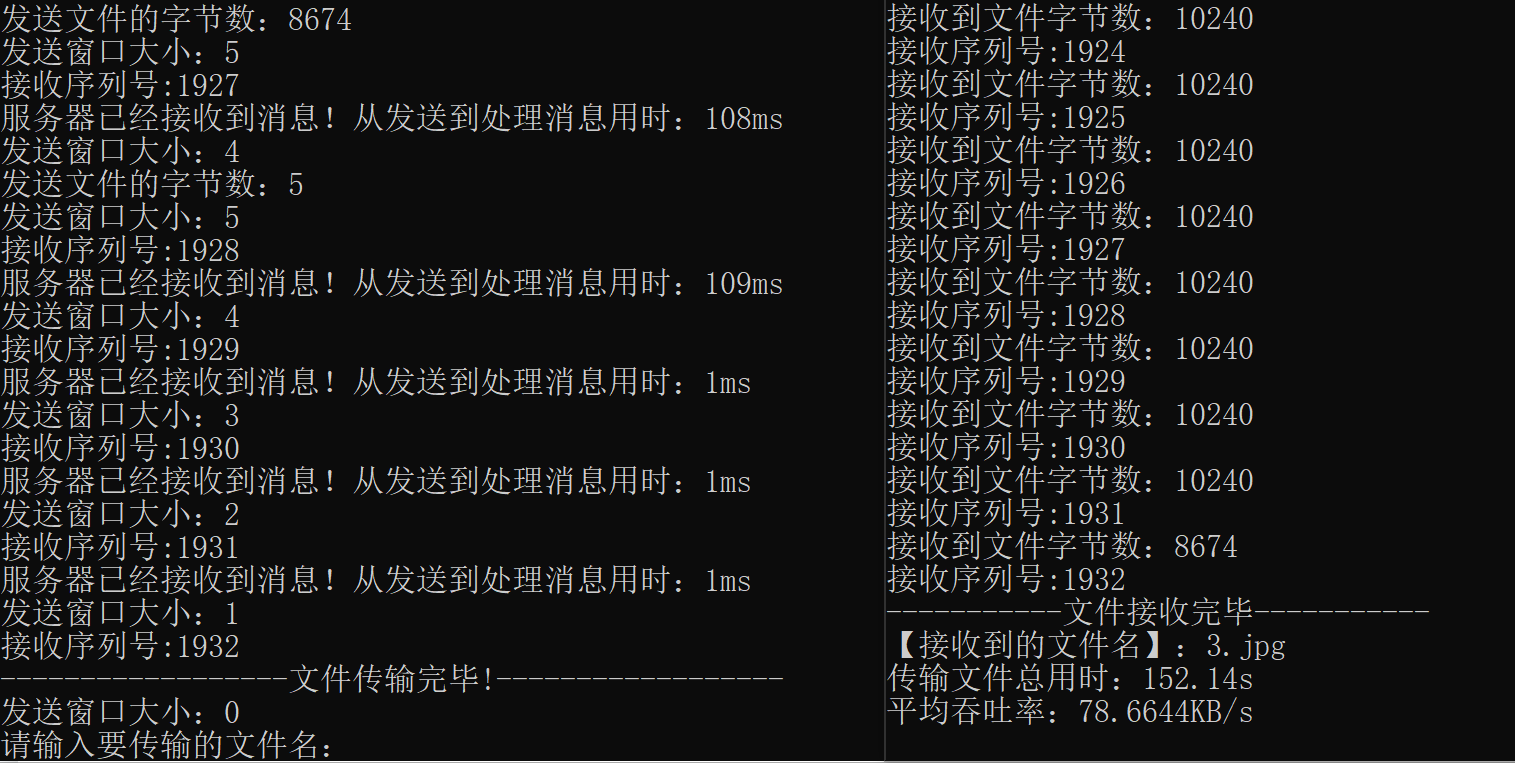


1. 发送3.jpg

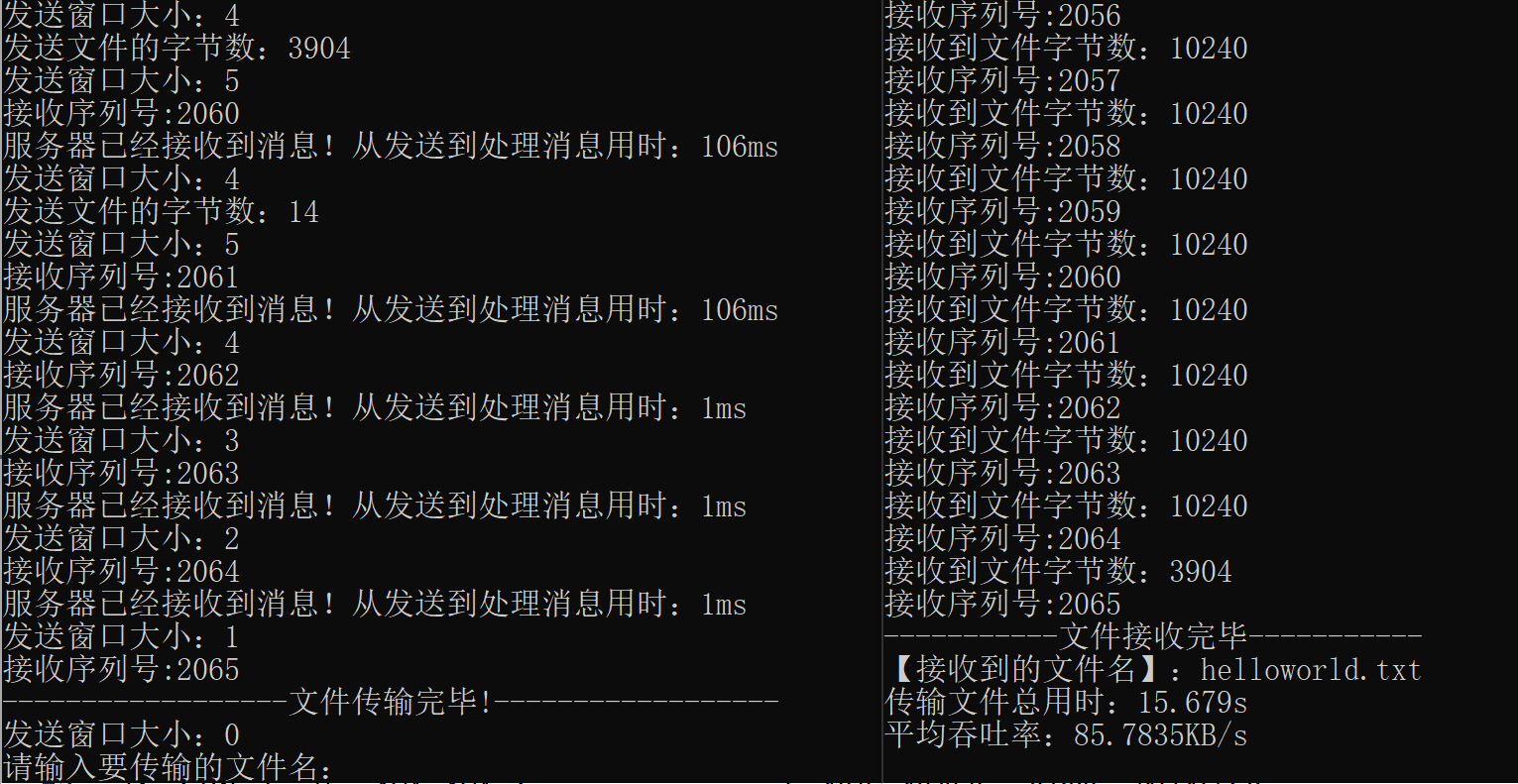
发送2.jpg文件也是类似的，这里可以看看里面的超时重传：



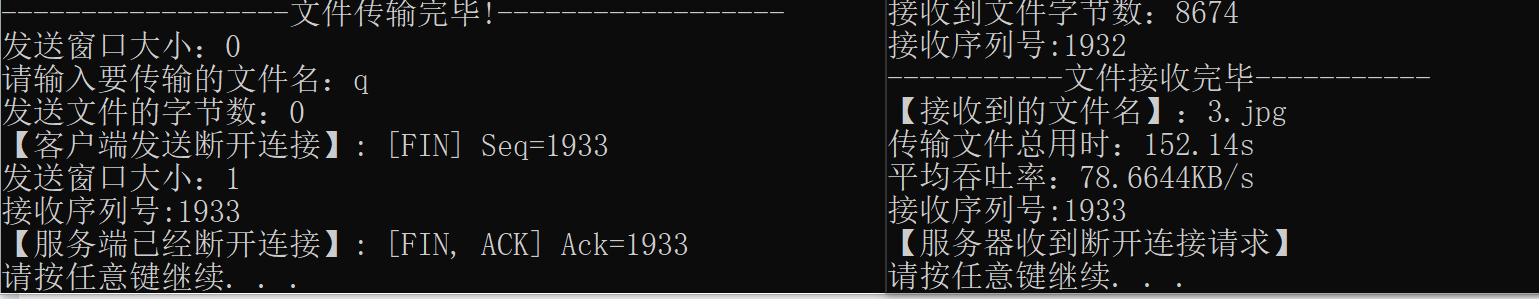
看看最后的日志输出：



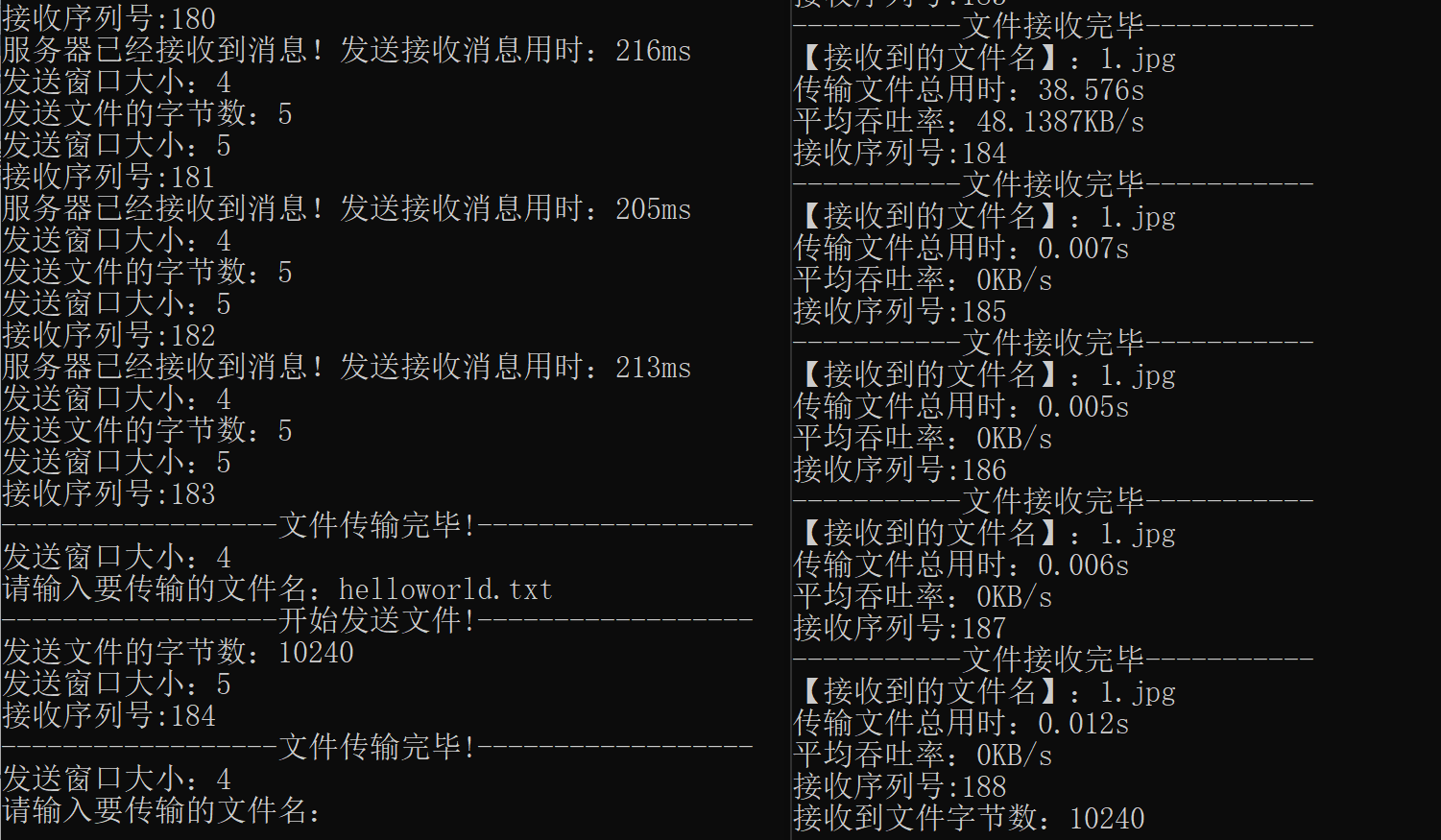
1. 发送helloworld.txt



5、如果输入q，可以看到，断开连接，退出了程序：



1. 出现的bug以及解决方法
2. 在实现过程中，发现当增加服务器端的延迟后，会出现最后传文件名的时候重复传名字，由于接收端程序设置的是判断消息是文件名时，创建一个文件并写入数据，因此在重复传文件名却没穿数据时，出现创建一个空文件，覆盖掉之前的文件。如下所示：



因此，在发送端新增加一个状态state = 3，用来表示一个文件已经发送完毕，但还没有接收所有的ACK，这时程序只能接收ACK消息，不能发送消息，实验证明程序可以恢复正常。