

计算机网络第三次实验报告

- 姓名：刘沿辰
- 学号：2012543
- 年级：2020级

实验要求

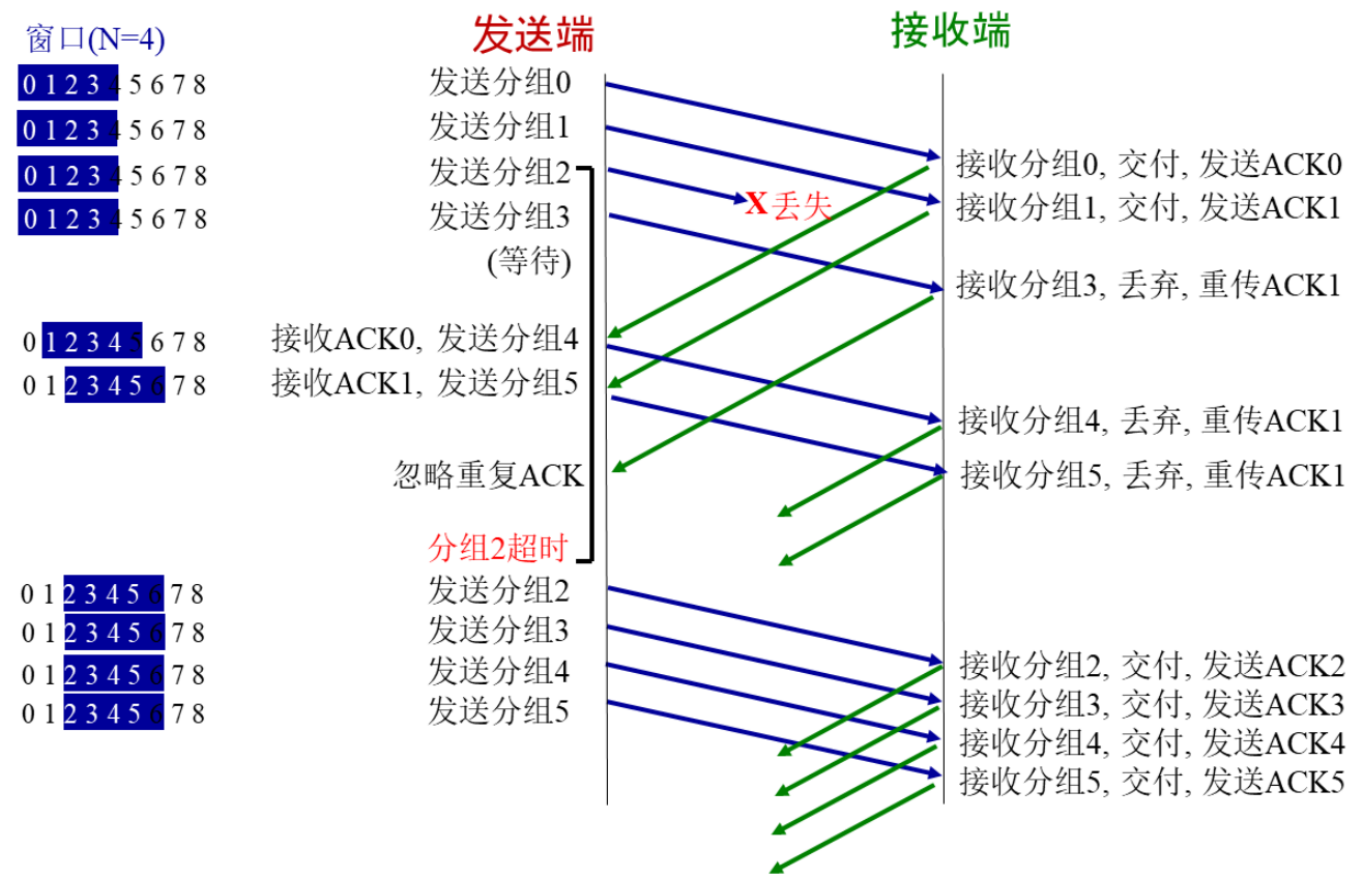
1. 将停等机制改成基于滑动窗口的流量控制机制
2. 给出详细的协议设计
3. 完成详细的实验报告
4. 编写的程序应结构清晰，具有较好的可读性
5. 提交程序源码和实验报告

实现平台

Windows11系统 Microsoft VisualStudio2022

协议设计

本次实验的滑动窗口实现采用gbrn方式示意图如下：



该方式仅需要ack作为确认码，由此，修改报文头结构如下：

```
struct MsgHead {  
    u_short len;           // 数据长度，16位  
    u_short checkSum;      // 校验和，16位  
    unsigned char type;    // 消息类型，8位  
    u_short ack;          // 累计ack，16位  
};
```

len字段表示消息内容长度（不包括文件头），由于u_short类型能表示的范围是0-65535，即8192个字节以内，所以每个包的最大长度maxSize在本次实验中设置为8000。

checkSum字段为校验和，计算方法较3.1中没有变化。

type字段有8位，目前使用了前5位，分别如下：

```
#define SYN 1  
#define ACK 2  
#define FIN 4  
#define PSH 8  
#define OVE 16
```

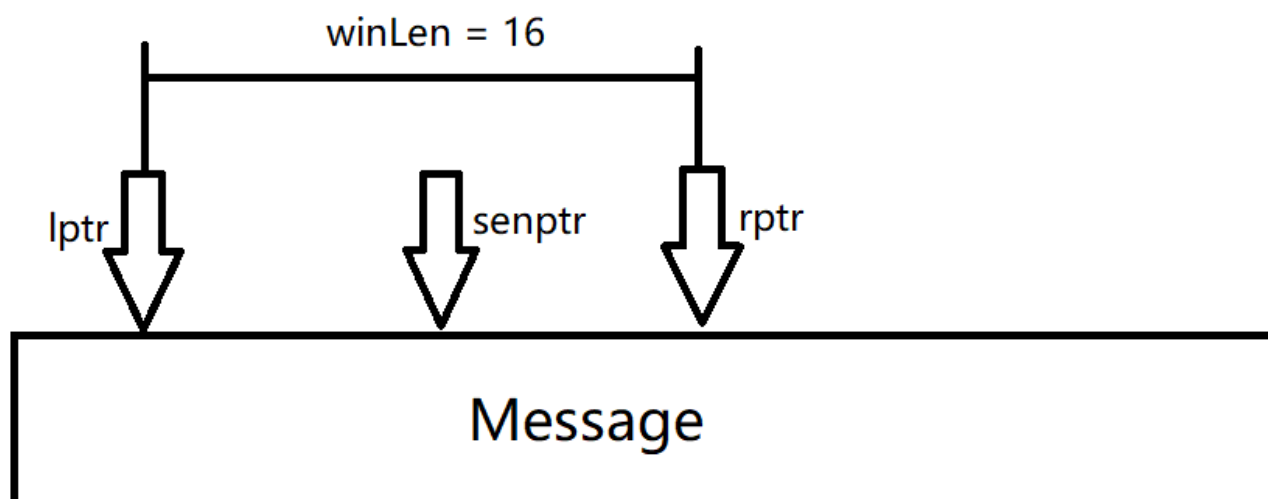
ack段是新修改加入的。由于gbn实现只需要ack来确认，为了保证精简性，在协议中去除了原有的seq计数，修改为ack。而ack需要较大的计数空间，所以采用了u_short类型，支持65535以内的ack计数。

滑动窗口

建立协议和简单的传输已经在3.1报告中有详细说明，在此不做赘述。本次主要说明滑动窗口的设计及实现方式。

滑动窗口指针

本程序的滑动窗口的原子单元为数据包，窗口长度winLen暂定为30，并使用三个指针实现，分别是标示滑动窗口左边界的lptr，标示滑动窗口右边界的rptr和标示当前正在传输的数据包的指针senptr。窗口采用左闭右开形式，即实际窗口范围为： $[[lptr, rptr))$ 即满足 $[lptr \leq senptr < rptr]$ 示意图如下：



文件发送端将文件分包后，发送模块会发送senptr指向的数据包，然后将senptr指向下一个数据包，直到senptr被rptr限制住或文件发送完毕。

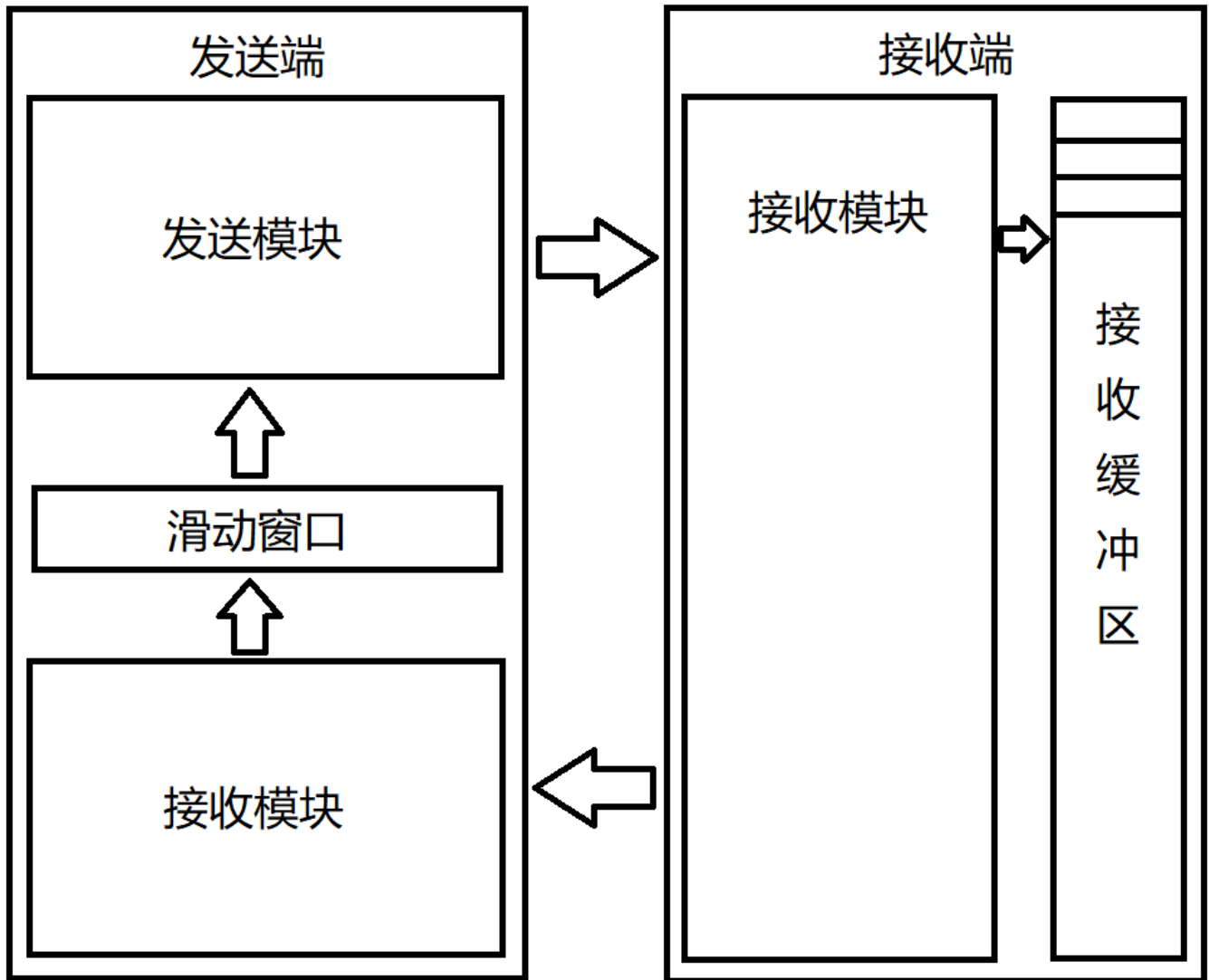
模块设计

本程序的发送端有两个模块组成，接收端只有一个模块。

发送端：由于发送端需要发送数据的同时接收数据，且需要将两个工作分开，所以设计了发送模块和接收模块两个线程来并行处理；而接收端只需要接收数据然后发出返回包，所以有一个接收模块即可。

控制关系

滑动窗口的整体传输控制关系如下：



- 发送端发送模块：发送模块只可以控制senptr，即当前发送包的指针。其主要功能是发送senptr指向的数据包，然后将senptr指向下一个包，然后循环发送数据包.....依此类推，直到数据包发完或遇到滑动窗口右边界，则卡住不动，等待接收模块调整滑动窗口。
- 接收端接收模块：接收端的接收模块不停的接收发送端发来的数据包，在收到数据包后返回一个ACK确认数据包，ack的值为上一个连续且正确收到的数据包的ack。并将正确接收到的ack包放到接收缓冲区中。
- 发送端接收模块：发送端的接收模块用于接受接收端发来的确认数据包并据此调整滑动窗口。若收到的确认包ack和滑动窗口左边界相同，则代表左窗口第一个数据包已经被正确收到，窗口整体右移一格；发送模块检测到窗口右移之后，就可以继续发出新的数据包。若收到的ack比左边界大，则可以直接将窗口移动到ack+1的位置，因为在此之前的数据包都已经被正确接收。

发送端：发送模块

发送模块的主要实现基于一个不停发消息的while循环，判断条件是`lptr < maxPackage`，maxPackage是提前计算出的整个文件发完需要多少数据包。

```

while (lptr < maxPackge) {
    if (senptr < rptr && senptr < maxPackge) {
        mtx.lock();
        // 设置信息头
        h.ack = senptr;
        cout << "senptr: " << (int)h.ack << endl;
        if (maxSize < length - senptr * 8000) {
            h.len = maxSize;
            h.type = PSH;
        }
        else {
            h.len = length - senptr * 8000;
            h.type = PSH + OVE;
        }
        h.checkSum = 0;
        char* sendBuf = new char[h.len + sizeof(h)];
        memset(sendBuf, 0, sizeof(sendBuf));
        memcpy(sendBuf, &h, sizeof(h));
        // data存放的是读入的二进制数据, sentLen是已发送的长度
        memcpy(sendBuf + sizeof(h), data + senptr * 8000, h.len);
        // 计算校验和
        h.checkSum = checkSumVerify((u_short*)sendBuf, sizeof(h) + h.len);
        memcpy(sendBuf, &h, sizeof(h));
        // 发送消息
        if (7 * clock() % 100 >= lost_rate) {
            if (send(clientSocket, sendBuf, h.len + sizeof(h), 0) == -1) {
                cout << "Error: fail to send messages!" << endl;
                mtx.unlock();
                return -1;
            }
        }
        senptr++;
    }
}

```

而在循环中首先对senptr、rptr和lptr上锁（防止发送端的接收模块在过程中修改窗口），然后设置消息头并封包，最后发出数据包，然后将senptr指向下一个数据包。整个过程不等待接收端的回馈，会一直发送直到滑动窗口内的数据包被发完。

当然，如果滑动窗口内的数据包已经被发完，则每次循环后线程都会睡眠100ms，然后再检查滑动窗口是否移动。

发送端：接收模块

发送端的接收模块同样是一个不停接收消息的while循环，作为另一个独立线程与发送模块并行执行。其在收到接收端发来的数据包并检查后，会根据数据包的ack来移动滑动窗口。

```

while (send_not_over) {
    if (recv(clientSocket, recvBuf, 1024, 0) > 0) {
        // 收到消息需要验证消息类型、序列号和校验和
        MsgHead h1;
        memcpy(&h1, recvBuf, sizeof(h1));
        if (7 * clock() % 100 < rerror_rate) {
            error_num++;
            cout << "-----" << endl;
            cout << "Receiver packet error!" << endl;
            cout << "-----" << endl;
            h1.checkSum = 0;
            memcpy(recvBuf, &h1, sizeof(h1));
        }
        if (h1.type == ACK && !checkSumVerify((u_short*)recvBuf, sizeof(h1) + h1.len)) {
            cout << "Send" << (int)h1.ack << " successfully" << endl;
            mtx.lock();
            if (lp_ptr - (int)h1.ack == 0) {
                lp_ptr++;
                rp_ptr++;
                cout << "窗口左侧移动至" << lp_ptr << ", 窗口右侧移动至" << rp_ptr << endl;
                if (lp_ptr == maxPackage) {
                    mtx.unlock();
                    return;
                }
            }
        }
    }
}

```

```

        else if (lp_ptr - (int)h1.ack < 0) {
            cout << "senptr从 " << senptr;
            senptr = (int)h1.ack + 1;
            cout << " 移动至" << senptr << endl;
            lp_ptr = (int)h1.ack;
            rp_ptr = lp_ptr + 10;
            cout << "窗口左侧移动至" << lp_ptr << ", 窗口右侧移动至" << rp_ptr << endl;
        }
        mtx.unlock();
    }
    else if (h1.type == OVE && !checkSumVerify((u_short*)recvBuf, sizeof(h1) + h1.len)) {
        cout << "-----Send-over-----" << endl;
        break;
    }
}

```

接收端：接收模块

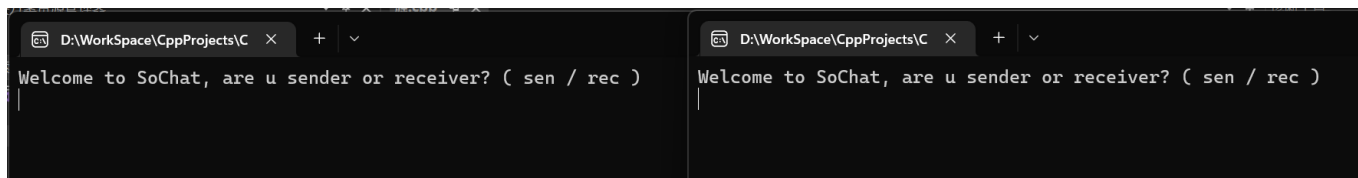
接收端的接收模块虽然既负责接收又负责发送，但是这两个过程可以顺序执行，所以只有一个模块。接收端在接收到数据包后会立即处理接收，然后返回一个ACK包，其中包含lastAck，代表接收端连续接收到的最后一个数据包。例如：接收端正确接收到了数据包1，数据包2.....数据包8，没有接收到数据包9，然后接收到了数据包10，数据包11.....那么此时，系统会丢弃数据包10和11及之后的数据包，并在确认包中一直返回“ack=8”，告诉发送端自己正确且连续收到的最后一个数据包是8号数据包。这样，当发送端发现自己连续很久没收到数据包，或者窗口已经在右边界，发送端就会重新发送窗口内的所有数据包。这样就可以保证接收端一定可以收到数据包。

发送数据：结束标记

本程序的type字段设计新添加了一个OVE标识，该标识标识“当前数据包为完整文件的最后一个包”，当接收端收到这个包后，就会准备关闭循环，并将缓冲区中的文件写到对应地址中，最后返回一个带OVE标识的确认包。同样的，发送端在接收到这个包后，也会准备结束传输，并断开连接。（如果确认包丢了没收到也没关系，长时间没有响应代表接收端已经接收到并退出了文件传输，发送端也会自动关闭传输）。

运行结果

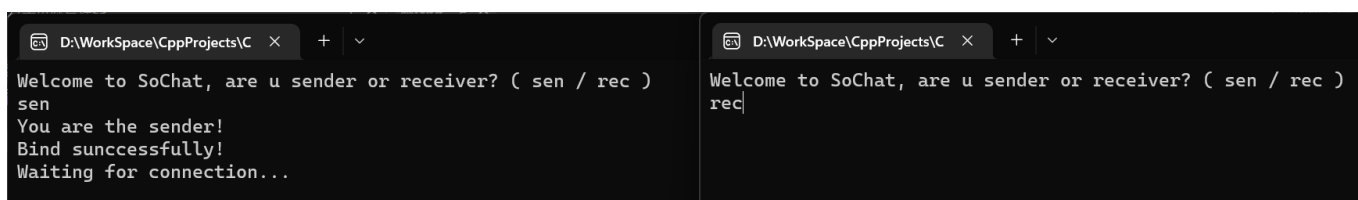
双开程序，端对端传输，首先建立连接：



```
D:\Workspace\CppProjects\C x + v
Welcome to SoChat, are u sender or receiver? ( sen / rec )

D:\Workspace\CppProjects\C x + v
Welcome to SoChat, are u sender or receiver? ( sen / rec )
```

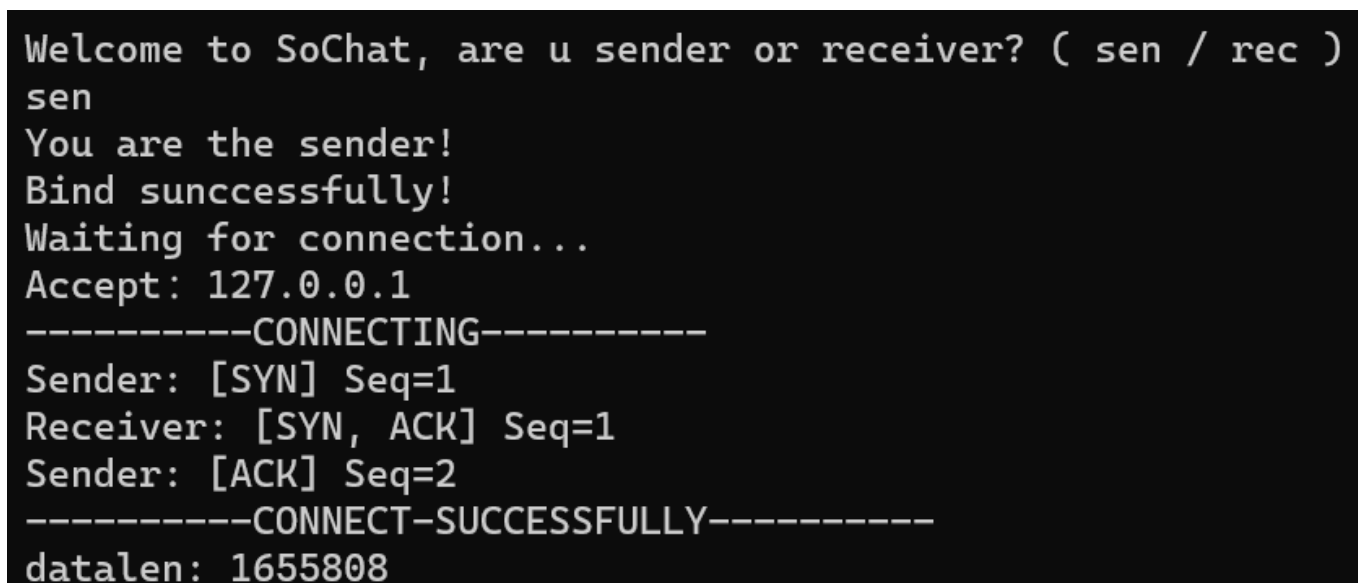
输入sender和receiver



```
D:\Workspace\CppProjects\C x + v
Welcome to SoChat, are u sender or receiver? ( sen / rec )
sen
You are the sender!
Bind sunccessfully!
Waiting for connection...

D:\Workspace\CppProjects\C x + v
Welcome to SoChat, are u sender or receiver? ( sen / rec )
rec
```

三次握手



```
Welcome to SoChat, are u sender or receiver? ( sen / rec )
sen
You are the sender!
Bind sunccessfully!
Waiting for connection...
Accept: 127.0.0.1
-----CONNECTING-----
Sender: [SYN] Seq=1
Receiver: [SYN, ACK] Seq=1
Sender: [ACK] Seq=2
-----CONNECT-SUCCESSFULLY-----
datalen: 1655808
```

建立连接后传输文件

```
senptr: 0
Send length: 8000, senptr: 0, rate: 0%
senptr: 1
Send length: 8000, senptr: 1, rate: 0%
senptr: 2
Send length: 8000, senptr: 2, rate: 1%
senptr: 3
Send length: 8000, senptr: 3, rate: 1%
senptr: 4
Send length: 8000, senptr: 4, rate: 2%
senptr: 5
Send length: 8000, senptr: 5, rate: 2%
senptr: 6
Send length: 8000, senptr: 6, rate: 3%
senptr: 7
Send length: 8000, senptr: 7, rate: 3%
senptr: 8
Send length: 8000, senptr: 8, rate: 3%
senptr: 9
Send length: 8000, senptr: 9, rate: 4%
senptr: 10
Send length: 8000, senptr: 10, rate: 4%
senptr: 11
Send length: 8000, senptr: 11, rate: 5%
senptr: 12
Send length: 8000, senptr: 12, rate: 5%
senptr: 13
Send length: 8000, senptr: 13, rate: 6%
senptr: 14
```

可以看到发送端在没有收到反馈包的时候不停的发出文件


```
窗口左侧移动至1，窗口右侧移动至31  
Send1 successfully  
窗口左侧移动至2，窗口右侧移动至32  
Send2 successfully  
窗口左侧移动至3，窗口右侧移动至33  
Send3 successfully  
窗口左侧移动至4，窗口右侧移动至34  
Send4 successfully  
窗口左侧移动至5，窗口右侧移动至35  
Send5 successfully  
窗口左侧移动至6，窗口右侧移动至36  
Send6 successfully  
窗口左侧移动至7，窗口右侧移动至37  
Send7 successfully  
窗口左侧移动至8，窗口右侧移动至38  
Send8 successfully  
窗口左侧移动至9，窗口右侧移动至39  
Send9 successfully  
窗口左侧移动至10，窗口右侧移动至40  
Send10 successfully  
窗口左侧移动至11，窗口右侧移动至41  
Send11 successfully  
窗口左侧移动至12，窗口右侧移动至42  
Send12 successfully  
窗口左侧移动至13，窗口右侧移动至43  
Send13 successfully  
窗口左侧移动至14，窗口右侧移动至44
```

然后接收到确认包后，调整窗口位置

由此，便可不停调整窗口，直到文件传输完毕。