课程项目汇报

——TCP_sliding_window_tool用户使用手册

一、项目名称及成员分工

项目名称：

模拟TCP滑动窗口协议的工作原理，集合成工具：**TCP_sliding_window_tool**

成员分工：

吴雨娟：交互动画编程、PPT制作、课程项目汇报撰写

黄斐桢：发送方代码撰写、调试

黄蓉：发送方代码撰写、调试

黄蕴怡：接收方代码撰写、调试

李涵：接收方代码撰写、调试

二、项目背景、动机及目标

项目背景：

TCP是一个面向连接的可靠的传输协议，既然是可靠的就需要对传输的数据进行确认。TCP窗口机制有两种，一种是固定窗口大小，另一种是滑动窗口。数据在传输时，TCP会对所有数据进行编号，发送方在发送过程中始终保持着一个窗口，只有落在发送窗口内的数据帧才允许被发送；同时接收方也始终保持着一个接收窗口，只有落在窗口内的数据才会被接收。这样通过改变发送窗口和接收窗口的大小就可以实现流量控制。

滑动窗口是TCP使用的一种流量控制方法。该协议允许发送方在停止并等待接收确认报文前可以连续发送多个分组。由于发送方不必每发一个分组就停下来等待确认，因此该协议可以加速数据的传输。只有在接收窗口向前滑动时，发送窗口才有可能向前滑动。收发两端的窗口按照以上规律不断地向前滑动，因此这种协议又称为滑动窗口协议。

项目动机：

在课程中学习了TCP滑动窗口的知识后，对于这种精妙的设计十分感兴趣，想要用自己所学的知识编程模拟出TCP滑动窗口的工作过程。

项目目标：

使用C++进行编程，代码主要分为发送方和接收方两部分。在命令行输入相关参数传输文件，可以显示窗口的滑动以及数据传输过程中的丢包信息等。同时对TCP滑动窗口的原理进行可视化，做成交互动画的形式，更直观地展现代码中的原理和机制。实现上述功能后，集成一个工具**TCP_sliding_window_tool**，便于用户使用，观察TCP滑动窗口情况。

三、项目内容、任务及方案

项目内容：

使用C++进行编程，模拟TCP滑动窗口协议的工作原理，代码主要分为发送方和接收方两部分。并用交互动画直观地展现代码中的原理和机制。为了方便观看具体的TCP窗口滑动，我们设定了接收方收到一个数据包，就对该数据包发送确认ACK的机制。

项目任务：

1.实现TCP滑动窗口机制。

2.展示TCP窗口滑动过程及数据包传输信息。

3.交互动画演示TCP滑动窗口原理。

4.实现上述功能，集成一个工具**TCP_sliding_window_tool**。

项目方案：

1.在课程实验三代码的基础上进行修改，加上模拟网络丢包、模拟窗口滑动等控制机制。

2.在命令行输出中实现TCP滑动窗口过程的可视化，将不同信息用不同颜色标出。

3.用scratch编程制作交互动画。

四、结果和分析

（1）使用C++编写的模拟TCP滑动窗口程序在Linux系统下的运行结果展示

这个部分只对运行结果进行分析，代码部分的分析详见第五部分代码分析。

为了方便用户使用该工具，此处附上操作说明：

注：

用户在Linux系统下进入TCP_sliding_window_tool，输入下述命令将源代码编译成可执行文件：

```
make
```

发送方命令行输入格式：

```
./sendfile <filename> <window_len> <buffer_size> <destination_ip>
<destination_port> <loss_rate>
```

发送方字体颜色意义：

绿色：收到的ack确认分组。

白色：窗口大小及控制信息、未发送过的分组正常发送。

蓝色：已发送但未收到ack或者发送后丢失的分组，进行超时重传操作。

红色：发送方发送到接收方的分组丢失。

发送方文件为：data/send.txt

"window:"后面的值表示当前发送方滑动窗口的位置，窗口从0开始滑动

"buffer NO:"表示当前缓冲区编号，初始值为0

接收方命令行输入格式：

```
./recvfile <filename> <window_size> <buffer_size> <port> <loss_rate>
```

并且要先运行接收方程序，再运行发送方程序。

接收方字体颜色意义：

绿色：收到数据包，正常发送ack。

红色：收到数据包，模拟丢失ack。

白色：窗口大小及控制信息。

接收方文件为：data/recv.txt

"window:"后面的值表示当前接收方滑动窗口的位置

"The received data will be written to buffer (数值)":表示当前缓冲区编号，初始值为0

注：要先打开服务器（接收方），再打开客户机（发送方）进行数据传输。

以下展示几种正常情况下的运行结果：

1.发送方和接收方的丢包率都为0

接收方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./recvfile data/recv.txt 7 32 5556 0
```

发送方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./sendfile data/send.txt 7 32 127.0.0.1 5556 0
```

接收方结果（开头部分）：

```
Bind socket to port: 5556 successfully.
listening...
connected: 127.0.0.1
---The received data will be written to buffer 0---
window:     0 ---------- 6
Receive seq num: 0. No error, ack replied.
window:     1 ---------- 7
Receive seq num: 1. No error, ack replied.
window:     2 ---------- 8
Receive seq num: 2. No error, ack replied.
window:     3 ---------- 9
Receive seq num: 3. No error, ack replied.
window:     4 ---------- 10
Receive seq num: 4. No error, ack replied.
window:     5 ---------- 11
Receive seq num: 5. No error, ack replied.
window:     6 ---------- 12
Receive seq num: 6. No error, ack replied.
window:     7 ---------- 13
Receive seq num: 7. No error, ack replied.
window:     8 ---------- 14
Receive seq num: 8. No error, ack replied.
window:     9 ---------- 15
Receive seq num: 9. No error, ack replied.
window:     10 ---------- 16
Receive seq num: 10. No error, ack replied.
window:     11 ---------- 17
Receive seq num: 11. No error, ack replied.
window:     12 ---------- 18
Receive seq num: 12. No error, ack replied.
window:     13 ---------- 19
Receive seq num: 13. No error, ack replied.
```

发送方结果（开头部分）：

```
window: 0-----6
-----buffer NO: 0-----
        send_num: 0
        send_num: 1
        send_num: 2
        send_num: 3
        send_num: 4
        send_num: 5
        send_num: 6
        0:ack
        1:ack
        2:ack
        3:ack
        4:ack
        5:ack
window: 6-----12
        send_num: 7
        send_num: 8
        send_num: 9
        send_num: 10
        send_num: 11
        send_num: 12
        6:ack
        7:ack
        8:ack
        9:ack
window: 10-----16
        send_num: 13
        send_num: 14
        send_num: 15
        send_num: 16
        10:ack
        11:ack
        12:ack
```

从上述运行结果中可以看到：发送方和接收方的丢包率都为0，所以都没有出现红色的丢包提示。在双方都没有丢包的情况下，由于是在本机的两个终端中运行，不用经过路由器，所以数据包和ACK包都是按序到达的。在发送端，由于线程执行先后的问题，窗口没有一个个滑动。但是在接收方可以看到，窗口在平滑地滑动。

2.发送方和接收方的丢包率均不为0

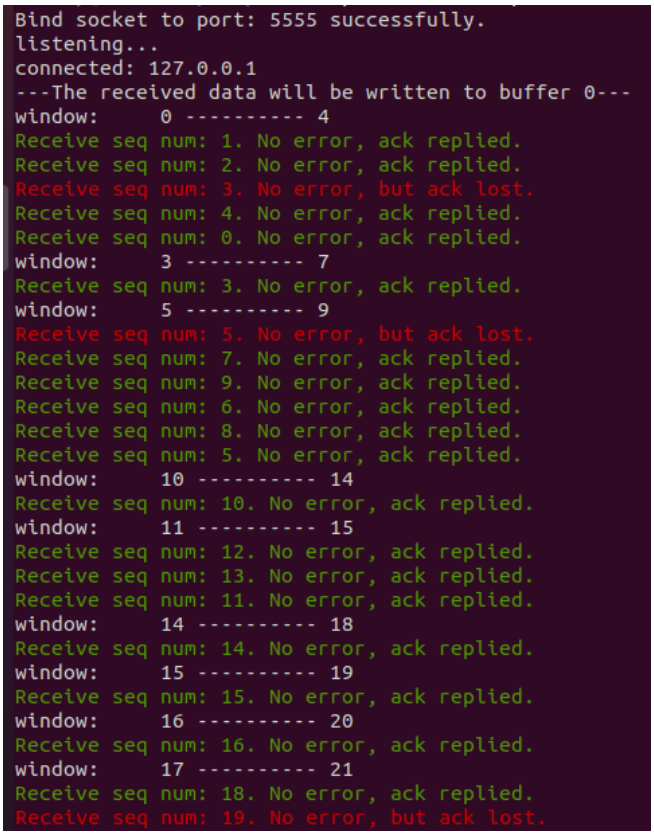
接收方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./recvfile data/recv.txt 5 64 5555 0.2
```

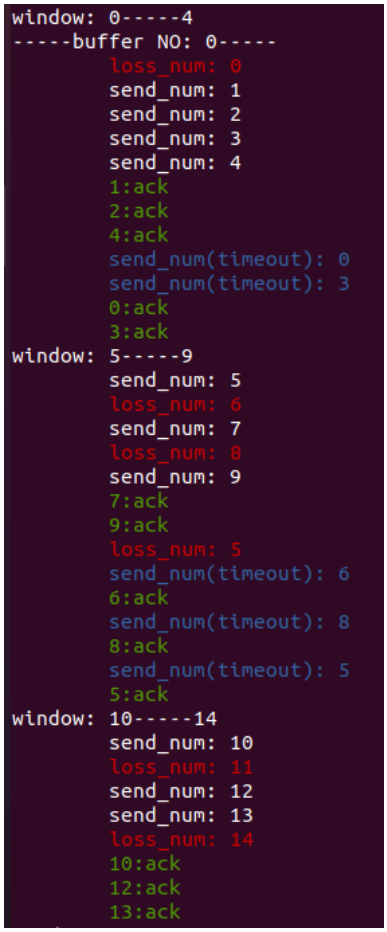
发送方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./sendfile data/send.txt 5 64 127.0.0.1 5555 0.1
```

接收方结果（开头部分）：



发送方结果（开头部分）：



从上述运行结果中可以看到：由于接收方有20%的概率丢包，所以接收方发送的ACK有20%的概率丢失，发送方未收到ACK，就重传报文。同样，发送方有10%的概率丢包，所以发送方发送的数据包有10%的概率丢失，需要重传。接收方红色字体部分提示ACK包丢失，如上图中的3号、5号报文的ACK丢失，在发送方可以看到相应的蓝色字体部分标出的重传。发送方红色字体提示数据包丢失，如上图中的0号、6号、8号，在截图中也能看到相应的蓝色字体部分标出的重传。接收方收到重传后，会正常发送ACK，显示的是绿色字体，如上图中3号报文ACK丢失，发送方重传，接收方收到重传报文后正常发送对3号报文的ACK。

错误输入示例说明：

1.发送方和接收方的缓存区大小，即buffer_size值需一致，否则会出错。

例如：

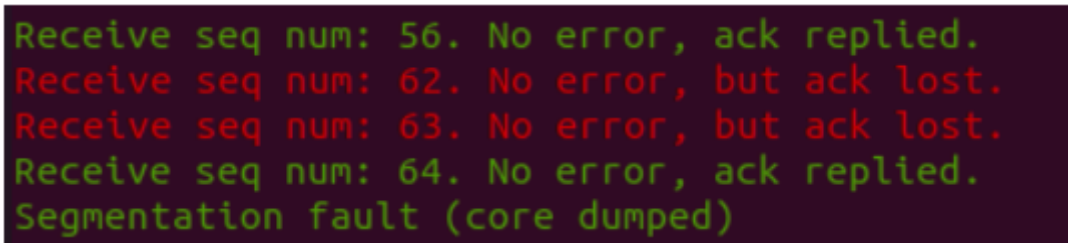
接收方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./recvfile data/recv.txt 6 64 6669 0.2
```

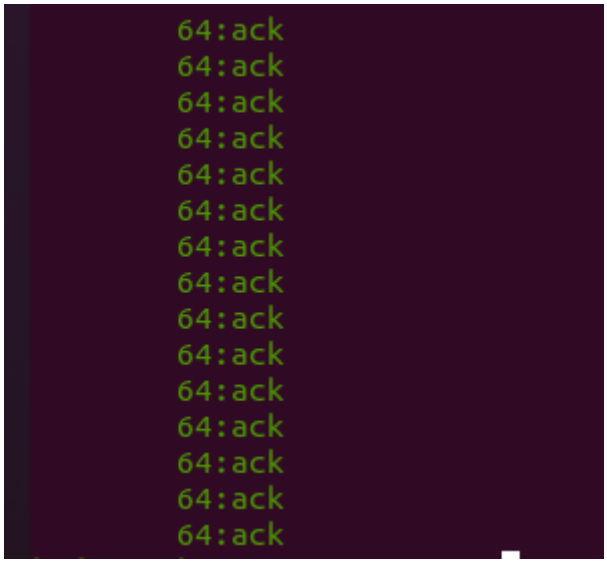
发送方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./sendfile data/send.txt 6 128 127.0.0.1 6669 0.1
```

接收方结果（最后部分）：



发送方结果（最后部分）：



如果发送方和接收方的缓存区大小不一致，如上图所示，就可能会产生内核错误，分片失败，导致数据传输失败。运行后接收方的文件recv.txt是空白的。所以用户在使用过程中需注意，尽量使发送方和接收方的缓存区大小一致。

2.发送方的窗口不能大于接收方的窗口，否则会出错。

例如：

接收方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./recvfile data/recv.txt 6 1024 6669 0.2
```

发送方命令行输入：

```
hadoop@ubuntu:~/net/TCP_sliding_window_tool$ ./sendfile data/send.txt 60 1024 127.0.0.1 6669 0.1
```

接收方结果（最后部分）：

```
Receive seq num: 47. Frame error, ack replied.
Segmentation fault (core dumped)
hadoop@ubuntu:~/net/TCP_sliding_window_tool$
```

发送方结果（最后部分）：

```
        send_num(timeout): 17
        send_num(timeout): 18
        send_num(timeout): 20
hadoop@ubuntu:~/net/TCP_sliding_window_tool$
```

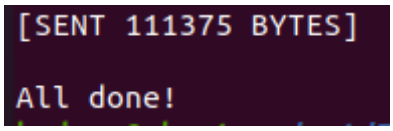
从上图的结果中可以看到，如果输入的发送窗口大于接收窗口，会产生帧错误。发送方没有显示"All done!"。打开接收方文件recv.txt可以看到，文件是空白的，数据传输失败。

成功运行结果说明：

1.接收方的运行结果末尾部分显示：

```
Received EOT, wait to end.
[RECEIVED 111375 BYTES]
[STANDBY TO SEND ACK FOR 3 SECONDS \ ]
All done!
```

2.发送方的运行结果末尾部分显示：



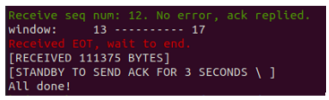
如果用户运行后在终端看到如上结果，表示运行成功，在接收方的recv.txt文件中可以看到接收的数据，与发送方的send.txt文件内容一致。

相关计算说明：

1.接收方的相关计算说明：

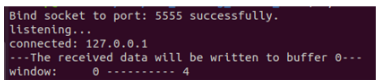
命令行输入：

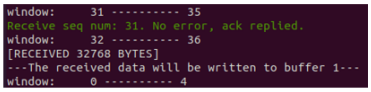
recv.txt收到了111375字节：

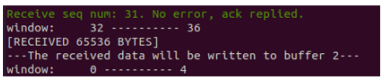


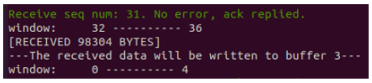
每个缓冲区大小：32×1024=32768字节

111375/32768=3……13071，所以需要4个缓冲区，编号0-3：







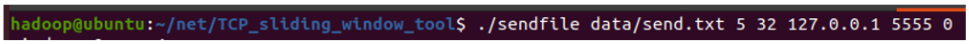


每个缓冲区中报文序号：0-31

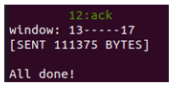
最后一个缓冲区：13071/1024=12……783，所以还需13个报文来发送，报文序号为0-12

2.发送方的相关计算说明：

命令行输入：

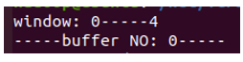


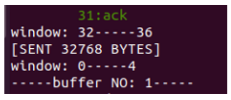
send.txt共有111375字节：

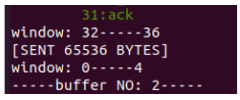


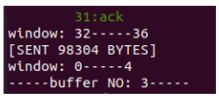
每个缓冲区大小：32×1024=32768字节

111375/32768=3……13071，所以需要4个缓冲区，编号0-3：









每个缓冲区中报文序号：0-31

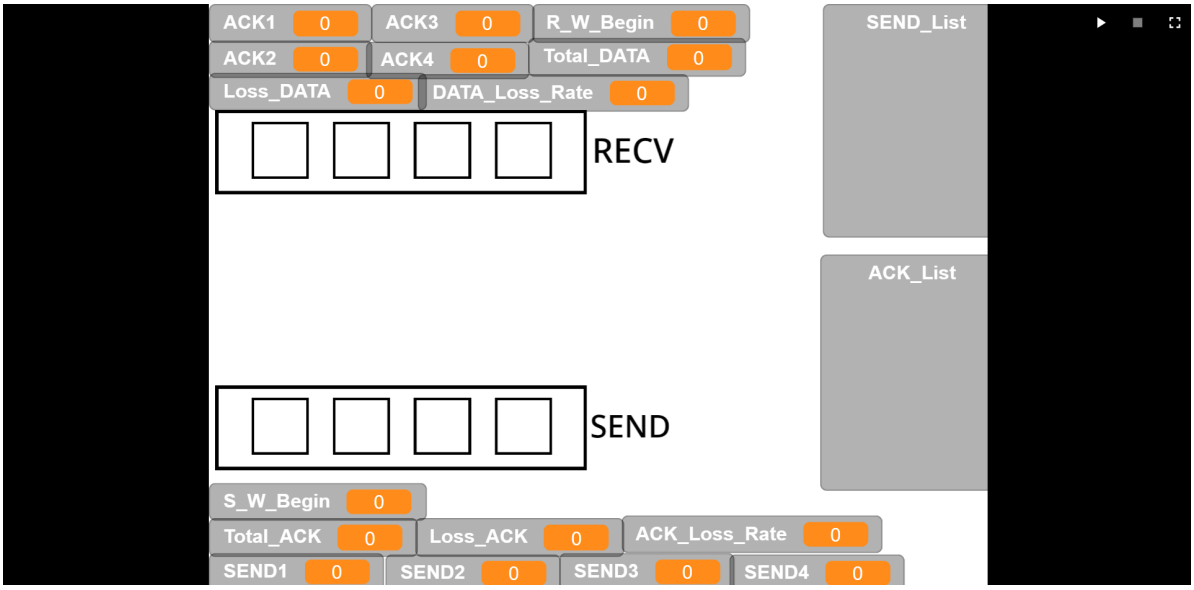
最后一个缓冲区：13071/1024=12……783，所以还需13个报文来发送，报文序号为0-12

（2）使用scratch编写的TCP滑动窗口交互动画展示

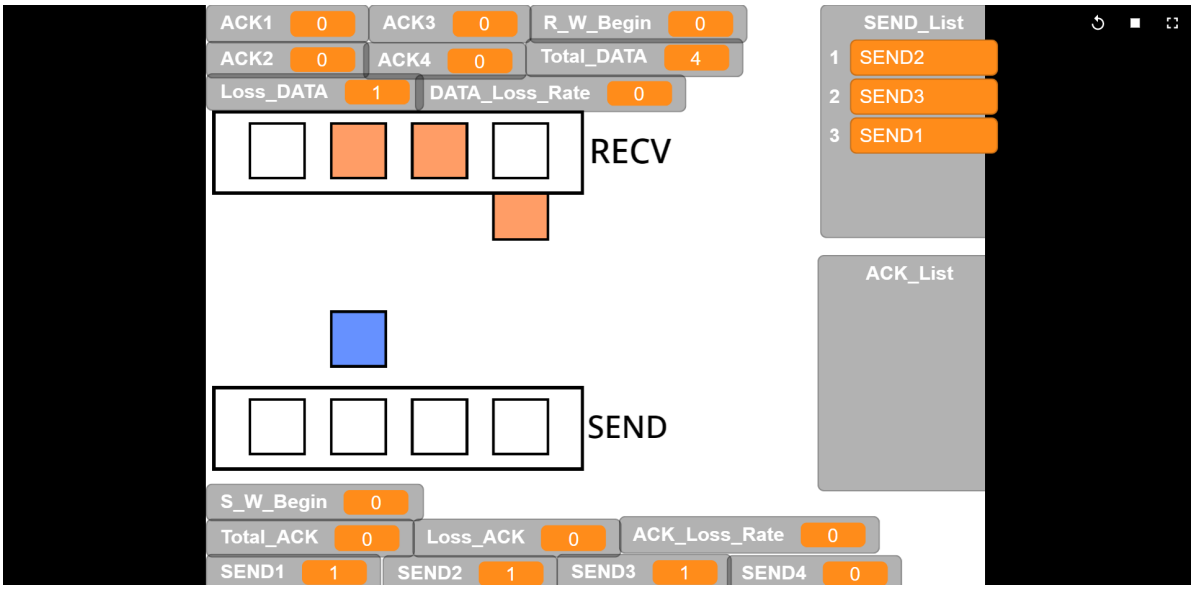
1.在命令行输入下述命令，打开show.html，运行交互动画

```
firefox show.html
```

打开后页面如下图所示：



打开后动画会自动开始运行，用户可以开始交互。



每个运动中的方块表示一个数据包。运行过程中，鼠标触碰到运动中的方块，方块就会消失，表示这个数据包丢失。橙红色的方块表示发送方发送的数据包，蓝色方块表示接收方发送的ACK确认包。

运行完毕后，如果要重新该运行交互动画，点击两下右上角的第一个按键，完成初始化后可以重新运行。

2.交互动画相关说明

①关于动画界面

动画界面展示了发送方和接收方的滑动窗口，由于界面大小限制，发送方和接收方的滑动窗口大小都为4，用来简单演示原理。窗口右侧标记SEND的是发送方的窗口，标记为RECV的是接收方的窗口。橙红色的方块表示发送方发送的数据包，蓝色方块表示接收方发送的ACK确认包。发送方和接收方都有一个包括了4个数据包大小的长方形，那是滑动窗口，随着数据包的发送和接收，它们会进行滑动。发送方发送的数据包编号：从左至右分别为SEND1、SEND2、SEND3、SEND4。接收方发送的ACK包编号：从左至右分别为ACK1、ACK2、ACK3、ACK4。

②关于变量

动画界面中标出了用到的所有变量以及变量列表。

最右侧有两个列表：SEND_List和ACK_List，分别记录成功接收到的数据包和ACK包。

最下方的变量中，SEND1、SEND2、SEND3、SEND4表示发送的1、2、3、4号数据包的状态，值为0表示接收方还未接收到该数据包，值为1表示接收方收到了该数据包。Total_ACK值表示接收方发送的ACK确认包总数，Loss_ACK值表示丢失的ACK确认包数量，ACK_Loss_Rate表示ACK确认包的丢失率。S_W_Begin表示发送方滑动窗口的起始位置，初始值为0。

最上方的变量中，ACK1、ACK2、ACK3、ACK4表示发送的1、2、3、4号ACK确认包的状态，值为0表示发送方还未接收到该ACK确认包，值为1表示发送方收到了该ACK确认包。Total_DATA值表示发送方发送的数据包总数，Loss_DATA值表示丢失的数据包数量，DATA_Loss_Rate表示数据包的丢失率。R_W_Begin表示接收方滑动窗口的起始位置，初始值为0。

③关于运行结果

运行动画后，用鼠标触碰方块，可以造成丢包。观察动画运行结果可以发现：在发送方，只有当当前窗口最左边的数据包收到ACK确认后，窗口才会开始滑动，滑动到窗口内最左侧数据包未收到ACK处为止。在接收方，只要当前窗口最左边的位置收到了相应数据包，窗口就开始滑动，滑动到窗口内最左侧位置未收到数据包处为止。而且，一定是接收方的接收窗口先开始滑动，因为只有接收方收到了数据包，才能发送ACK包，发送方收到了ACK包才能开始滑动发送窗口。

④关于代码

由于scratch编程的特殊性，源代码不便在此处展示，详细源代码可见文件TCP sliding window.sb3。

五、C++源代码说明分析

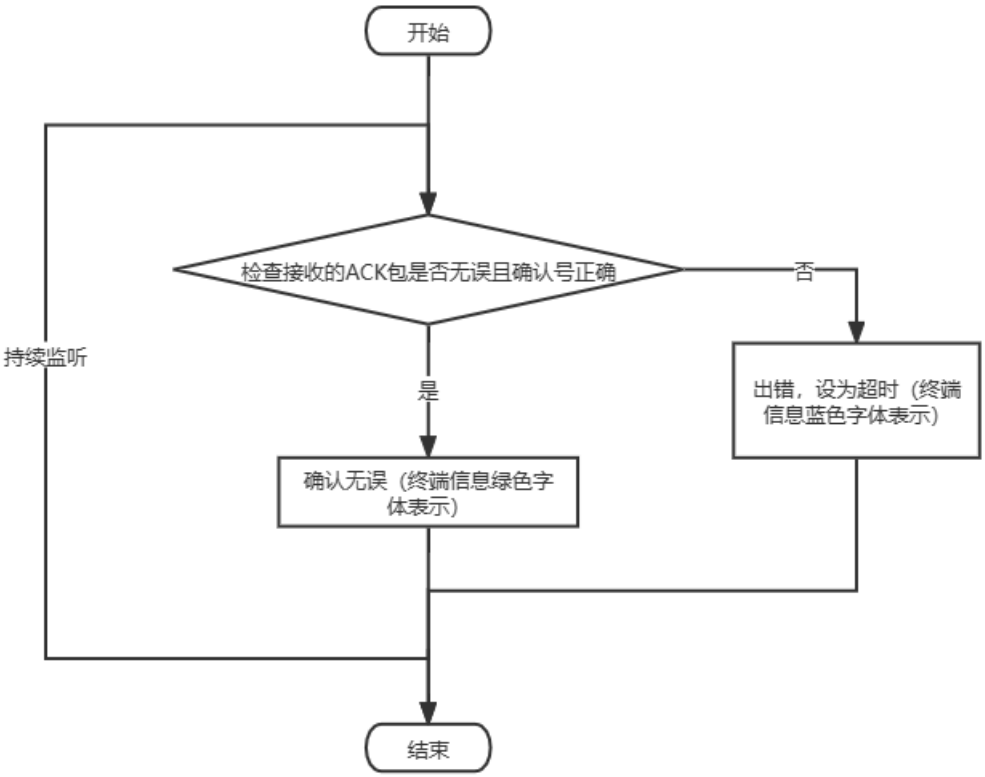
注：这个部分进行主要功能性源码的说明分析，如socket建立TCP连接相关的内容就不在此赘述了，详细源代码分析见源代码文件中的注释。

名词说明：发送方即客户端，接收方即服务器端。帧、数据包、报文是一个概念。

1.发送方代码sendfile.cpp

（1）void listen_ack()函数

函数功能流程图如下：

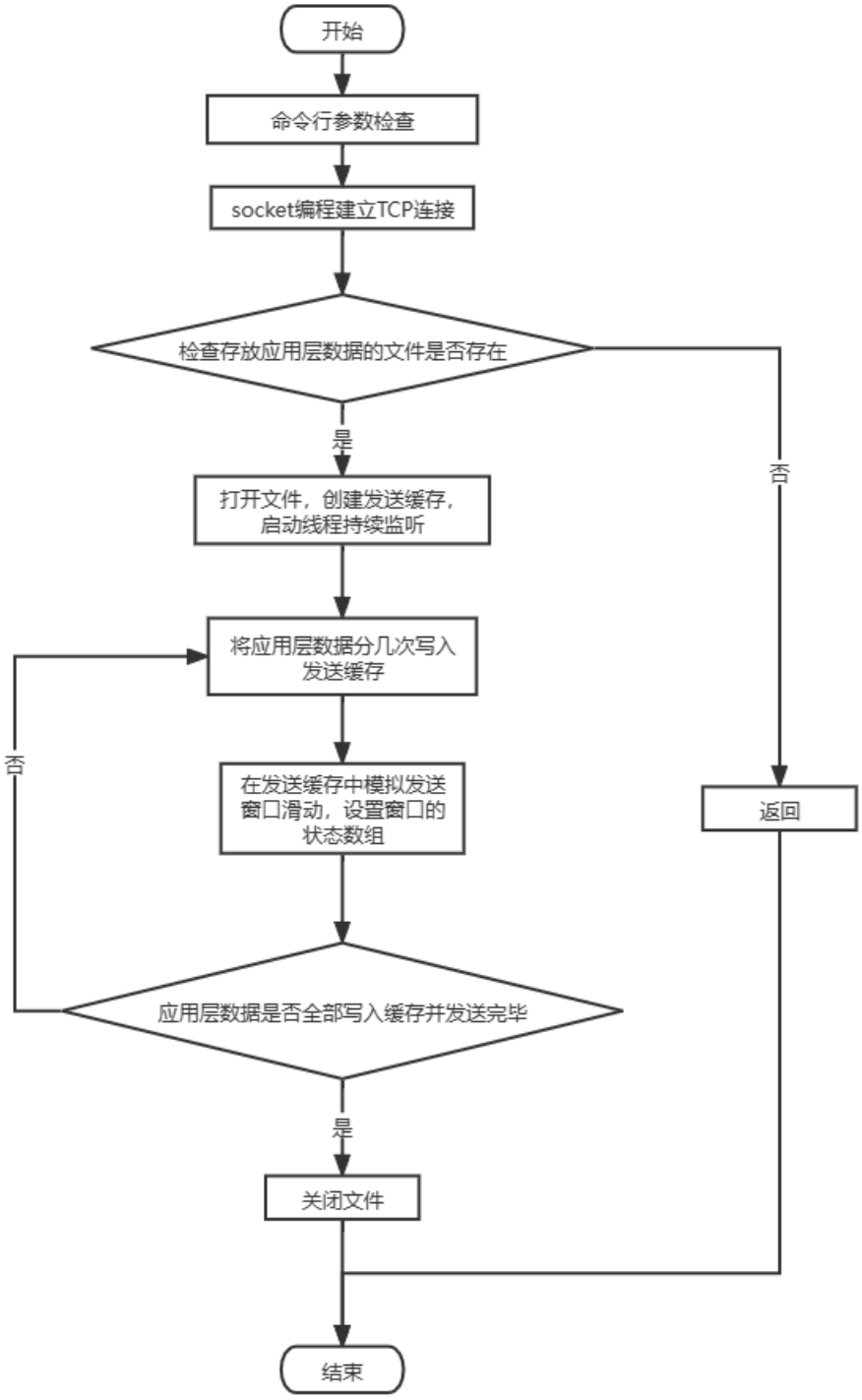


该函数用于接收服务器端发来的确认分组并进行差错检测，设置滑动窗口状态。

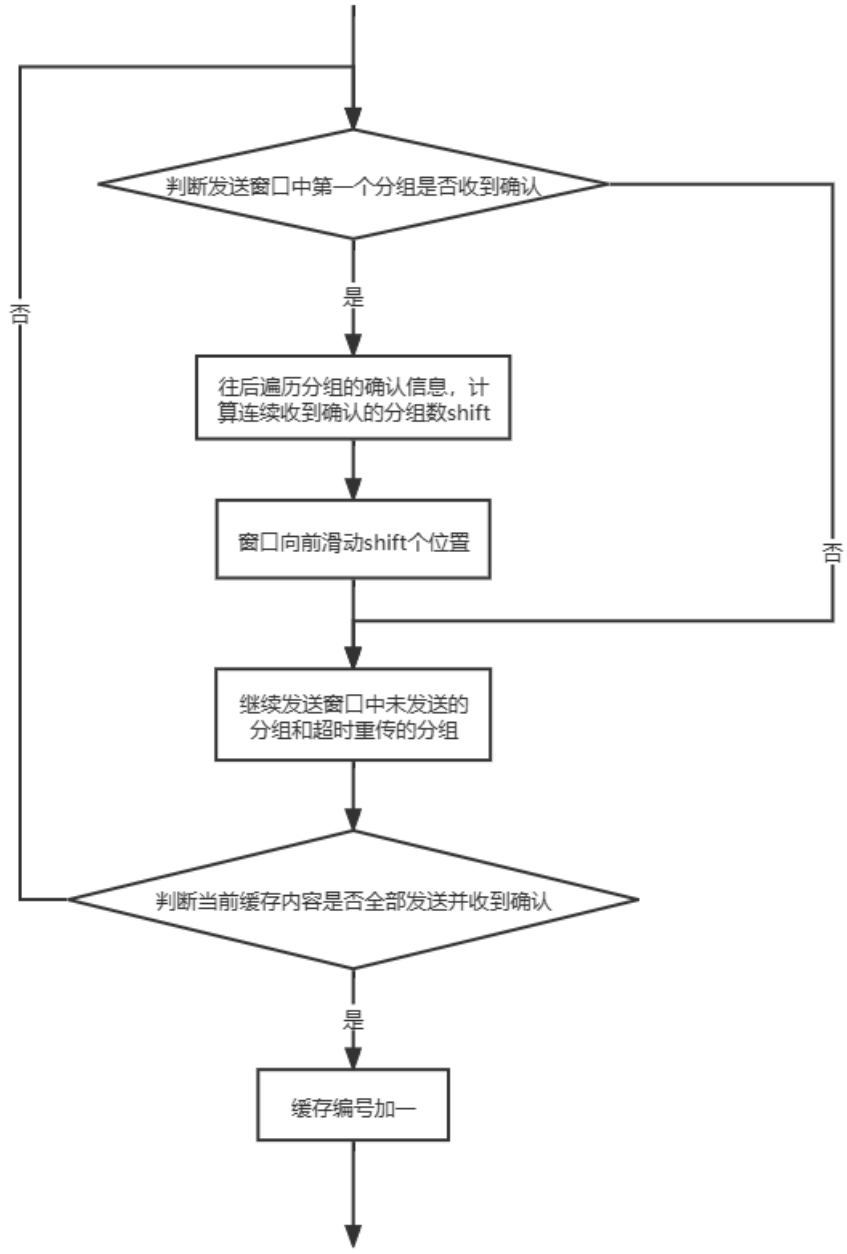
先读取数据检查确认分组是否超时，使用函数read_ack()函数（见addition.cpp）。若分组出错，直接丢弃，不作处理；若该确认分组无误，且确认号ack值正确（在发送窗口中），如果该确认分组无误，设置对应状态为收到确认；如果该确认分组出错，需重新发送，通过设置发送时间为TMIN，分组进行超时重传实现。

（2）主函数

主函数功能流程图如下：



其中在发送缓存中模拟发送窗口滑动并设置窗口的状态数组部分，放大细节后画出流程图如下：



以下是对流程图中各个部分功能的说明：

①命令行参数检查

进行命令行参数检查，若正确将命令行参数存入对应变量。

②socket编程

建立TCP连接。

③检查存放应用层数据的文件是否存在

若存在，打开文件，准备往发送缓存中写入数据；若不存在，程序结束。创建缓冲区buffer作为发送缓存。启动线程运行listen_ack()函数，持续监听，接收服务器端发来的确认分组。

④将应用层数据分几次写入发送缓存

设置read_done作为应用层数据是否已全部写入发送缓存的标志，初始为false。当read_done=false，就读取部分应用层的数据，写满新的发送缓存，计算发送缓存中总共需要发送的分组个数（seq_count）。

⑤在发送缓存中模拟发送窗口滑动，设置窗口的状态数组

首先初始化发送窗口的状态和指针位置。当前窗口最左侧是位置0，当前缓冲区序号是0。

设置send_done作为发送缓存中的数据全部发送并收到确认分组的标志，初始为false。当send_done=false。

首先判断第一个分组是否收到确认，若未收到确认，跳过if语句，发送分组。若第一个分组收到确认，则计算按需收到确认分组的个数，并将窗口进行滑动，设置对应状态数组和指针（模拟窗口滑动）。

窗口滑动后，根据窗口中的最新状态，发送窗口中未发送的分组依据超时需要重传的分组。通过socket编程发送到服务器端，需要向下交付给网络层，使用creat_frame()函数（见addition.cpp）;并更新滑动窗口的状态数组，重新设置分组发送的时间。

⑥循环退出

当前发送缓存中的数据全部收到确认分组，移向下一个发送缓存（应用层往发送缓存中继续写入后续数据）。

当应用层数据分批次全部写入发送缓存并发送完毕，退出循环，应用层数据发送完毕。

（3）部分机制说明

①多线程中的互斥锁mutex

```
#include <mutex>
mutex window_info_mutex;
window_info_mutex.lock();
window_info_mutex.lock();
```

多线程中的提供了多种互斥操作，可以显式避免数据竞争。mutex是可锁定的类型，用来保护临界区代码，mutex在某一个时刻只能被一个线程锁定，解锁后才能被其他线程上锁。

lock()：给mutex上锁。如果mutex此时被其他线程占用，线程阻塞直到mutex被释放；线程为mutex上锁后直到调用unlock函数，期间mutex一直被当前线程占有；如果mutex此时已经被当前线程锁住，则产生死锁。

unlock()：解锁mutex。调用线程释放对该mutex的所有权。

②线程

```
thread recv_thread(listen_ack);
recv_thread.detach();  // 线程退出，释放所有分配的资源
```

实例化一个线程对象recv_thread，使用函数listen_ack构造，该线程就开始执行了。

thread::detach()：从 thread 对象分离执行的线程，允许执行独立地持续。一旦线程退出，则释放所有分配的资源。调用 detach 后， *this 不再占有任何线程。

③通过域名获取IP地址

```
struct hostent *dest_hnet;
/* 通过域名转换成IP地址 */
dest_hnet = gethostbyname(dest_ip);  //desi_ip为域名
```

gethostbyname()函数可以完成通过域名获取IP地址的这种转换，它的原型为：

struct hostent *gethostbyname(const char *hostname);

hostname为主机名，也就是域名。使用该函数时，只要传递域名字符串，就会返回域名对应的 IP 地址，返回的地址信息会装入 hostent 结构体。

④时间戳

```
time_stamp *window_sent_time;  // 发送窗口中，每个分组发送后的超时计时器
time_stamp TMIN = current_time();
window_sent_time[ack_seq_num - (lar + 1)] = TMIN;
elapsed_time(current_time(), window_sent_time[i]) > TIMEOUT
```

（4）代码主要功能说明

①模拟TCP发送端send_file.cpp的发送数据原理

应用层数据写入TXT文件（放在data文件夹中），以二进制读取文件的方式（rb），将应用层数据分批次写入发送缓存（设置一个缓冲区buffer作为发送缓存），发送窗口在发送缓存中滑动，发送缓存中所有分组全部发送并收到确认后，重新往发送缓存中写入应用层数据（发送缓存的个数buffer_num加一）。

发送窗口共有2个指针lp1、lp2（P1、P3，与课本上略有不同，在课本P1、P3前一个位置）和3个状态数组。3个状态数组大小均为窗口大小，下标为窗口中的分组编号，分别记录发送窗口中每个分组的以下状态：是否已发送window_sent_mask（bool型）、是否收到确认分组window_ack_mask（bool型）、发送分组的时间window_sent_time（time_stamp型）。

②模拟网络丢包

客户端通过发送端不发送数据，实现模拟发送端发往接收端的过程中丢包；服务器端通过接收端不发送数据，实现模拟接收端发往发送端的过程中丢包。从命令行输入丢包率loss_rate（发送端发往接收端的丢包率），输入为小数。在发送数据前，产生一个随机数，若该随机数对100取余小于loss_rate×100，则不发送数据；若该随机数对100取余大于等于loss_rate×100，则发送数据。

③终端输出信息及颜色含义

发送窗口：输出窗口的前后沿，如0-----5，表示窗口大小为6。第一个数为lp1指针下一个位置，第二个数为lp2指针位置。

绿色：收到的ack确认分组。

白色：窗口大小及控制信息、未发送过的分组正常发送。

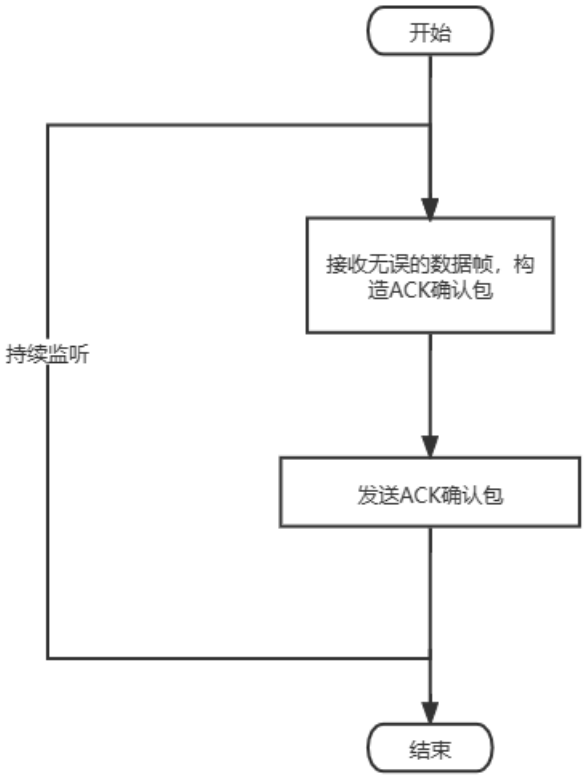
蓝色：已发送但未收到ack，或者发送后丢失的分组进行超时重传。

红色：发送方发送到接收方的分组丢失。

2.接收方代码recvfile.cpp

（1）void send_ack()函数

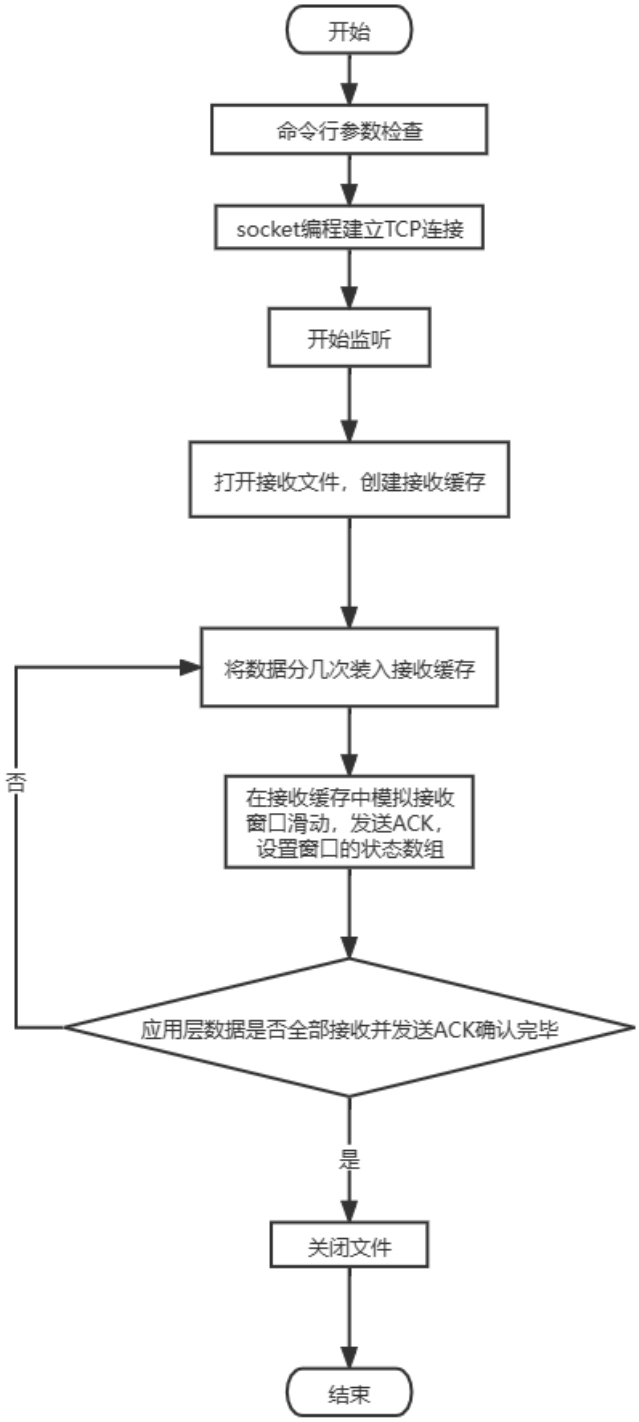
函数功能流程图如下：



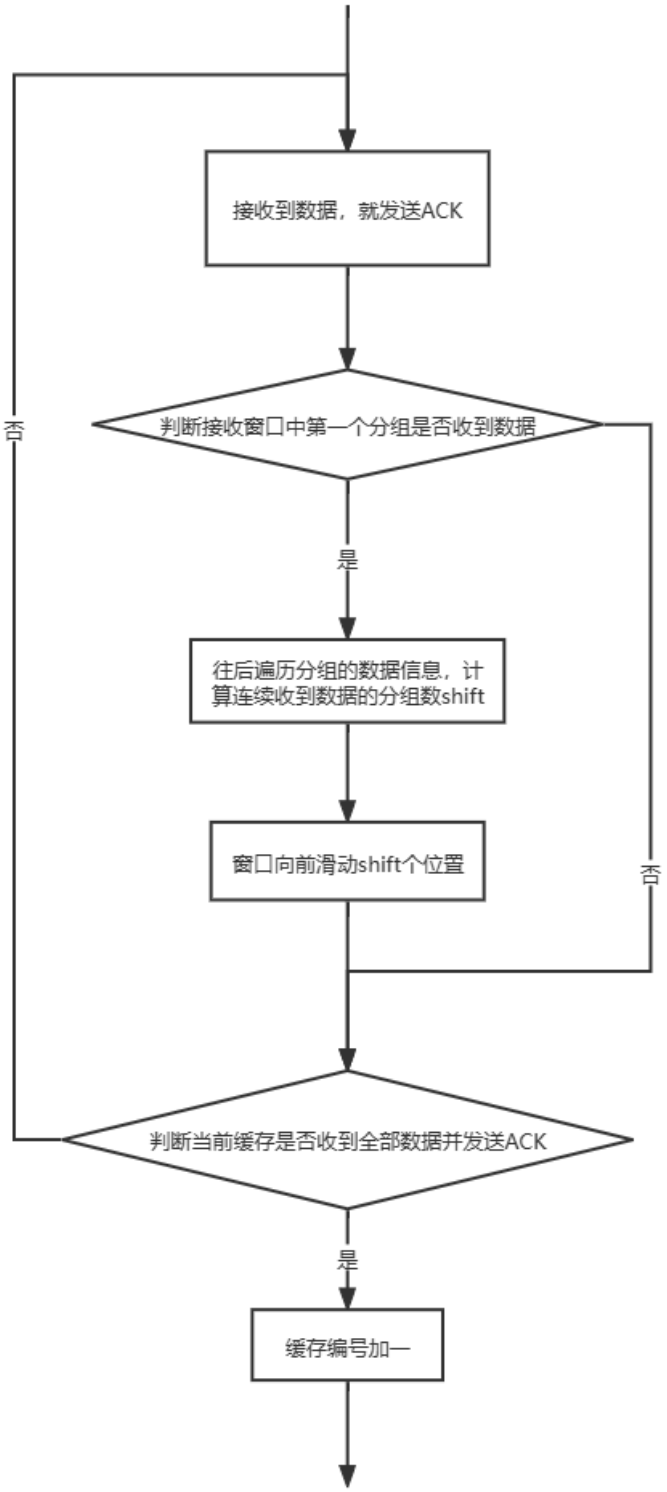
该函数用于关闭TCP连接时给所有接收到的分组回复ACK。因为关闭的时候接收方已经收到全部信息，但不知道发送方有没有收到全部ACK，如果发送方有再重传，就要回复ACK。

（2）主函数

主函数功能流程图如下：



其中在接受缓存中模拟发送窗口滑动、发送ACK并设置窗口的状态数组部分，放大细节后画出流程图如下：



以下是对流程图中涉及的各个部分以及一些未画出的功能的补充说明：

①命令行参数检查

判断输入的参数是否包含文件名， 窗口大小，缓冲区大小，端口号，丢包率这几项，如若满足，则根据参数的输入设定fname，window_len，max_buffer_size，port，loss_rate这几项的数值。

②缓冲区信息处理

若信息没有接受完（即收到的帧中没有EOT）则要继续接受信息。此时的缓冲区长度设置为最大缓冲区长度，并对缓冲区进行清空，打印出收到的信息将要写入buffer的信息。

③初始化

bool数组中存放的是true和false，true即代表数组中相应位置已经接收到发送给服务器的序列号，初始化所有位置均为false。

左右边界的设置不代表数组下标，代表接收窗口所能接受的序列范围。左边界代表此序列号已经收到，下一个序列号是第一个未收到的序列号，右边界则是指接收窗口可以接收的最后一个序列号，左右边界相差的长度为window_len。

④接收客户端发来的帧

将接受到帧的帧长返回，并将帧的内容写到frame中；再从frame读取帧，将帧的数据部分写入data，序列号填入recv_seq_num，实际数据长赋给data_size。

frame_error = read_frame(&recv_seq_num, data, &data_size, &eot, frame);

eot为true时，代表客户端发来的帧中有结束符，frame_error为true时则代表帧的传输出错。

⑤产生随机数，发送ACK包，并滑动接收窗口

若接收到的帧的序列号小于右边界的设定，则代表其在接受的范围内，并进行接下来的操作。

若接受到的帧并没有出错，则计算该帧存放在缓冲区的相对位置，准备接收新的数据帧。

若接收到的序列号就是左边界旁的第一个，则滑动窗口就需要移动。设定shift的值为1，则代表至少会收到一个，在接收窗口中进行查找，每找到一个值为true的项，则窗口左边界便移动一位，直到找到值为false的项，此时的shift就代表未收到的序列号。

此时再查找窗口中在值为false的这一项后面的所有项的值，采用的方法就是将值为false的这一项后面的项挪到窗口的前方，并将剩余位置填充为false，换言之就是使窗口发生偏移，偏移量即为shift。

若第一个收到的序号不是第一个未收到的序号，而是先收到了后面的序号，则存储这个数据包，但是接收窗口不滑动。

若收到的帧中含有EOT，则标记此次缓冲区全部收到后可以结束接收。

⑥丢包时输出丢失信息

```
    else //丢包时输出丢失信息
    {
      cout << RED << "Receive seq num: " << recv_seq_num;
      if (!frame_error)
        cout << RED << ". No error, but ack lost." << endl;
      else
        cout << RED << ". Frame error and ack lost." << endl;
    }
```

设置为红色字体的丢失信息，在终端可以看到。

（3）代码中的机制说明

相关机制与sendfile.cpp中基本相同，这里就不再赘述。此处说明一下终端输出颜色的含义：

绿色：收到数据包，正常发送ack。

红色：收到数据包，模拟丢失ack。

白色：窗口大小及控制信息。

3.辅助功能头文件addition.h

定义了我们用到的字体颜色：

```
#define RESET   "\033[0m"
#define BLACK   "\033[30m"      /* Black */
#define RED     "\033[31m"      /* Red */
#define GREEN   "\033[32m"      /* Green */
#define YELLOW  "\033[33m"      /* Yellow */
#define BLUE    "\033[34m"      /* Blue */
#define MAGENTA "\033[35m"      /* Magenta */
#define CYAN    "\033[36m"      /* Cyan */
#define WHITE   "\033[37m"      /* White */
```

定义了sendfile.cpp和recvfile.cpp中用到的一些函数：

```
char checksum(char *frame, int count);//计算校验和
int create_frame(int seq_num, char *frame, char *data, int data_size, bool
eot);//构造帧
void create_ack(int seq_num, char *ack, bool error);//构造ACK回复包
bool read_frame(int *seq_num, char *data, int *data_size, bool *eot, char
*frame);//读帧
bool read_ack(int *seq_num, bool *error, char *ack);//读ACK
```

4.辅助功能代码addition.cpp

为了使sendfile.cpp和recvfile.cpp中的执行过程更加简洁，将部分处理函数放到addition.cpp中实现。

addition.cpp中的部分函数功能代码借鉴了网上的资料，我们基本没有改动，只是使用这些函数接口实现我们想要的功能，所以此处就不分析具体代码，详细代码注释可见源代码addition.cpp。

六、实验小结

以下是小组内五个成员的实验心得：

1.吴雨娟：

在确定小组选题时，我们比较了不同题目的可操作性，最后确定了TCP滑动窗口模拟这个选题。我对于TCP滑动窗口机制的设计挺感兴趣，也一直想通过动画演示TCP滑动窗口机制，于是我在这次工作中就负责制作交互动画，以及课程项目汇报用户手册的撰写、展示PPT制作。

TCP滑动窗口原理其实挺好理解的，比如我们可以直观地看到一定是接收窗口先开始滑动。通过一步步解析它的工作步骤，我用scratch编程软件制作了交互动画，反复调试后实现了TCP滑动窗口的基本功能，而且可以显示出数据包和ACK包的发送数、丢失数、丢包率，可以帮助用户更好地复盘窗口滑动过程。但是动画也有一些不足的地方：由于受到动画窗口大小限制，窗口大小设置为4，再大的话画不下。而且也没有画出缓存，同样也是因为画不下。麻雀虽小五脏俱全，这个交互动画虽然简单，但是能完整地展现出TCP滑动窗口的原理，可以满足用户需求。

我们小组的其他四个组员分别负责发送方代码sendfile.cpp和接收方代码recvfile.cpp的编写。我们各自的工作完成后，进行了两次项目代码联调，解决了一些小问题，完善了项目功能。最后我将我们小组的项目工作集合成一个工具**TCP_sliding_window_tool**，这是一个TCP滑动窗口模拟工具，也是我们这个课程项目的最终产出结果。

在本次课程项目的工作中，小组成员们都能在我定下的deadline之前完成该阶段的工作，大家各司其职完成各自的工作，在最后的联调中互相提出改进意见，让我们的最终成果更加完善。感谢组员们对我的支持，也感谢谢怡老师和助教覃翔老师在工作中对我们提出的改进意见，让我们的工作更具展示性。我在这次课程项目工作中收获了许多宝贵的经验。

2.黄斐桢：

本次课程项目是编程实现TCP滑动窗口的机制以及方法，了解其在流量控制中的作用。本次实验中，我主要负责发送方代码的编写与调试。

我们组使用C++的socket编程建立TCP连接，用文件读写的方式来模拟应用层数据交付给传输层的过程，在程序中建立临时缓冲区来模拟发送缓存和接收缓存，分批次将文件中的内容写入缓存区，并把缓冲区的数据按数据块的大小分成数据报，并发送。通过几个窗口状态数组来模拟TCP滑动窗口过程中数据报的发送和确认，包括窗口的指针、未发送状态、已发送未收到确认状态、记录发送的时间判断是否超时等，实现了TCP的窗口滑动机制，也对数据报进行了差错检测，实现了可靠传输。TCP窗口滑动过程中的参数如，窗口大小、缓存大小、端口号等均可通过命令行输入。

为了观察由于网络层的因素造成的丢包、失序等造成的未收到确认报文段的超时重传，我们还模拟了网络中的丢包，丢包率也可从命令行参数输入。手动模拟了丢包后，更能清楚得观察到滑动窗口机制在面对不可靠的网络层提供的服务，如何进行窗口的滑动，做到可靠传输。实验中还在终端中通过不同的颜色来区分收发数据和是否丢包，能更直观的观察窗口如何滑动。

实验中还存在着许多可以改进的地方。如报文段的序号编号是按缓存大小编号，当一个缓存写满并全部发送完，序号会重新从0开始编号；发送方的窗口大小是根据接收方提供的接收窗口大小来决定，但实际实现中为了方便设置为固定值；还可以引入对RTO的计算来精确超时重传的时间。

实验增进了我对滑动窗口机制的理解，并在实践中对协议进行了简单的复现，也提高了我的编程能力。在实际编程过程中，还需要一定的进程、线程的知识，合理使用互斥锁，以免发生错误。复现过程中会发现许多意想不到的情况，需要考虑周全，特别是条件判断的语句，实验中就因为条件判断不全导致结果不理想，所有对实现协议时对各种可能发生的情况都需要充分考虑有了深刻的体会。

综上所述，本次课程项目提高了我的编程能力，加深了对课本知识的理解，积累了一些经验，增加了团队合作意识。

3.黄蓉：

TCP协议是运输层很重要的一个协议，这是因为它比较“简洁”却能实现可靠传输，而这种实现依赖于以字节为单位的滑动窗口，并且每一个字节都有相应序号，我们要做的是将该滑动协议用soket编程实现，在课程实验三的基础上，我们了解了各发送、接收、监听函数等，这对实验有很大帮助，编程语言采用c++，将确认分组是否有误、检验校验和等步骤用函数模块化，也增强了程序的可读性和简洁性。在这个过程中通过向队友的学习，对TCP滑动窗口有了更深一步的认识，能区别窗口和缓存的关系。网络情况多变，我们更实现了手动模拟网络丢包，由用户输入丢包率控制，更能在终端用颜色直观表示。

无论是实现发送端还是接收端都会涉及到一定编程，尽管原理实现已经在书上明了，但用代码具体实现仍会有一定难度，队友在其中做出了突出贡献，并给自己仔细讲解，让我在这个过程中学到很多忽略的知识。

调试是一个更重要的过程，丢包率太大？发送数据太少？用了已经绑定过的端口导致连接错误？收不到ack？发送方接收方缓冲区大小必须一样？每晚上99+的消息是不断改正但不断成功的见证。这也让我们对传送数据过程中发生的各种情况有更深刻的了解与体会。

4.黄蕴怡：

本次课程项目的主要目的在于深入了解TCP滑动窗口的机制以及该方法在流量控制方面的作用。我们使用C++编程模拟TCP滑动窗口在收发消息过程中的情况。本次实验中，我主要负责接收方代码撰写与调试。

TCP协议要求在接收方维护一个接收窗口，用于暂时存储发送方发送的消息，保证收到的报文不出错、不失序，同时要对收到的报文进行ACK回复。接收到的有序信息被送入缓冲区，此时窗口需要进行滑动，以便接收接下来的信息。我们还根据实际网络，模拟了ACK报文丢失的情况。

在实验过程中，我不仅加深了课本上学到的有关于TCP滑动窗口的知识，也对实际过程中可能产生的更多情况有了更全面的了解，比如丢包了怎么办，缓冲区大小对数据传输的影响，报文中序列号的作用等等。同时，也掌握了使用C++的socket模块编程的方法，比如UDP与TCP在socket中的不同使用等，也学习了线程、互斥锁的使用方法。

本次实验也有一定缺陷及可以改进的地方，比如我们实际使用的序列号最大值较小，局限于一个缓冲区之内，所以接收方和发送方的缓冲区大小需要一致。以及可以增加对重传时间和最大重传次数的限制。

总的来说，本次课程项目巩固了我的课本知识，加深了我对TCP协议的理解，增进了我的C++编程水平，加强了我的团队合作的意识，是一次很有意义的实验。

5.李涵：

在本次的课程项目中，我主要负责的部分是接收方的代码撰写和调试。本次的课程项目是基于TCP滑动窗口协议的工作原理展开的，所以在项目开始之前，我们一起认真学习了有关于TCP滑动窗口协议的内容并进行了讨论。在代码撰写和修改的过程中，我依照我们已经完成的接收方的代码书写了应用文档，并对每一个部分要完成的内容进行了解释与归纳，对代码内容后续的几次调整与更改也进行了记录。在本次的课程项目的完成过程中，我意识到了团队协作的重要性，小组成员有关项目的讨论让我更深入地理解了项目的内容，从而更好地参与其中。同时我也意识到了自己在代码撰写上的不足，以后需要着重提高自己在这方面的能力，可以为之后的团队项目做出更多贡献。

这次的课程项目让我收获了很多，对于我之后的学习与团队协作会有很大帮助。

七、附件说明

1.C++源代码

在**TCP_sliding_window_tool/src**中

2.数据文件

发送方数据在**TCP_sliding_window_tool/data/send.txt**中

接收方数据在**TCP_sliding_window_tool/data/recv.txt**中

当然用户也可以自己创建发送文件和接收文件，但是注意一定要创建发送文件后才能在命令行中输入该文件名作为参数，否则程序会报错，而接受文件可由程序创建，用户可以不用创建接受文件。

3.Makefile文件

TCP_sliding_window_tool/Makefile文件用于将源代码编译成可执行代码。

4.scratch动画源代码

源代码文件为：**TCP_sliding_window_tool/TCP sliding window.sb3**

5.scratch交互动画网页

交互动画网页：**TCP_sliding_window_tool/show.html**