南開大學
Nankai University

计算机网络 实验报告

基于 UDP 服务设计可靠传输协议并编程实现

实验 3-2

姓名： 辛浩然

学号： 2112514

年级： 2021 级

学院：网络空间安全学院

班级： 信息安全、法学

1 实验要求

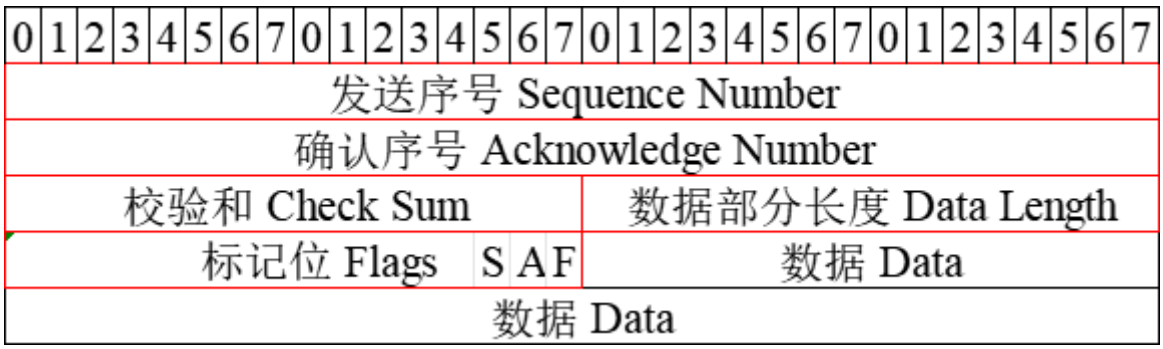
在实验3-1的基础上，将停等机制改成**基于滑动窗口的流量控制机制**，发送窗口和接收窗口不采用相同大小，接收窗口为1，发送窗口大于1。支持**累积确认**，传输给定测试文件。

- 单向传输：一端发数据，一端返回确认；
- 协议设计：数据包格式，发送端和接收端交互；
- **流水线协议**：多个序列号；
- 累积确认：**GBN** (Go Back N) 协议；
- 建立连接、断开连接：类似TCP的握手、挥手功能；
- 差错检验：校验和；
- 日志输出：收到/发送数据包的序号、ACK、校验和等，发送端和接收端的窗口大小等情况，传输时间与吞吐率。

2 协议设计

2.1 数据包格式

本次实验中，传输数据包包括首部和数据两部分，总体格式如下图所示：



具体而言，首部包括：

- 发送序号，32位：初始值随机产生，之后累积计算，表示已发送的数据字节总数。
- 确认序号，32位：接收方使用确认序号来告知发送方它期望接收的下一个字节的发送序号。确认序号是累积的，表示已成功接收的数据字节总数。
- 校验和，16位：仿照UDP校验和的计算方法，计算校验和，用于检测数据在传输过程中是否被篡改或损坏。
- 数据部分的长度：16位，用于记录数据的长度。
- 标记位，16位：目前使用的标记位为后三位，分别代表 `SYN` 、 `ACK` 、 `FIN` 。

数据部分 `data` 存放要发送的数据，限定可发送的最大长度为 `MSS` 。

2.2 建立连接

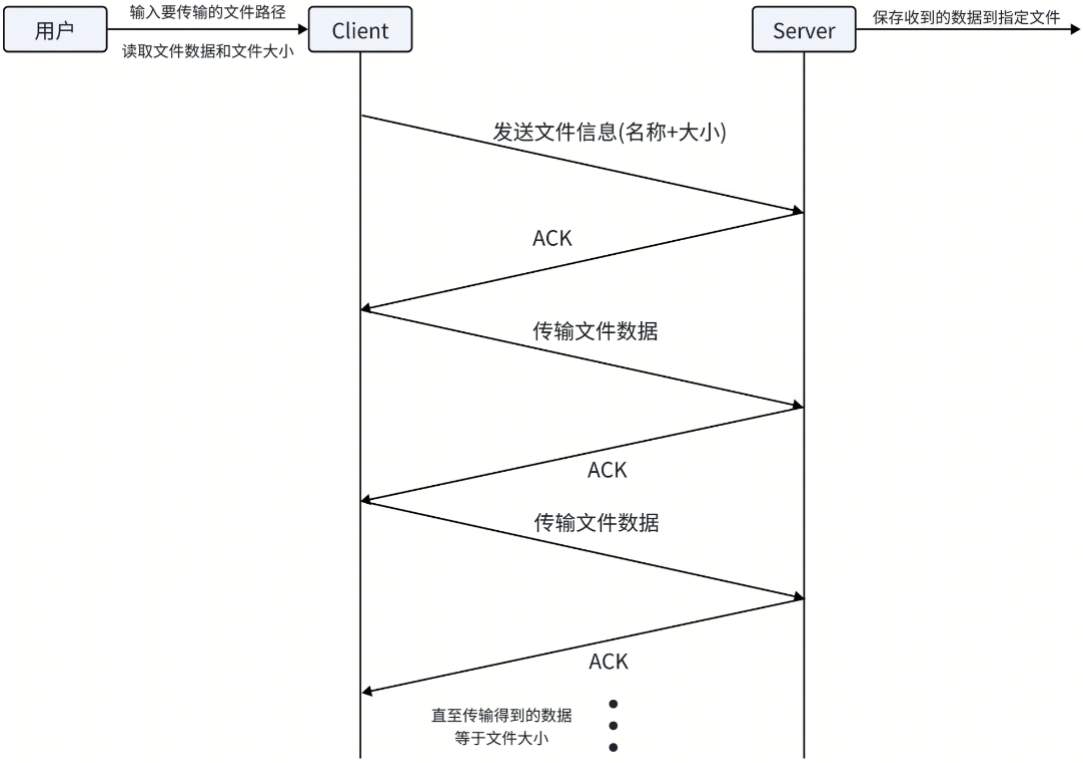
仿照TCP的三次握手，考虑到数据单向传输，因此建立连接只需要两次握手。

基本流程为：

- 第一次握手：客户端生成 `SYN` 报文，向服务器声明客户端的初始发送序号，发送至服务器，等待服务器的 `ACK` 报文；
- 第二次握手：服务器收到来自客户端的 `SYN` 报文后，判断标志位和校验和，回复 `ACK` 报文。
- 客户端超时未收到服务器的 `ACK` 报文，重传 `SYN` 报文。

2.3 文件传输的基本交互流程

本次实验中，建立连接后，客户端作为发送端，服务器作为接收端，发送文件数据的**基本交互流程**如下图所示：



如果用户在输入传输文件路径时输入 `exit` ，则客户端主动与服务器断开连接。

2.4 可靠数据传输

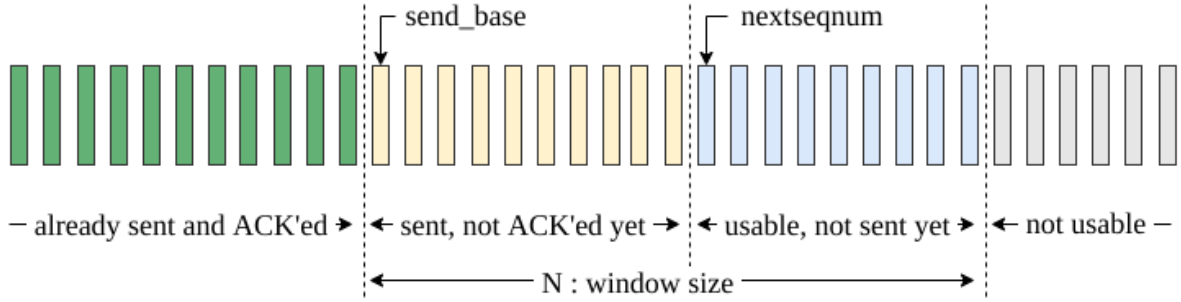
使用流水线可靠数据传输协议，采用基于滑动窗口的流量控制机制，即支持累积确认，回退N步进行差错恢复。

对于发送端的发送的数据包，每个数据包分配递增且不同的序号。

允许发送方发送多个分组(当有多个分组可用时)而不需等待确认，但在流水线中未确认的分组数不能超过某个最大允许数N，即窗口大小。

窗口的左边界代表最早发送但未确认数据包的序号，右边界代表最小的未使用序号(即下一个待发送数据包的序号)。

如下图，窗口内的数据包意味着已发送但未确认；窗口左边界左侧的数据包已发送且已确认；窗口右边界右侧的数据包未发送。

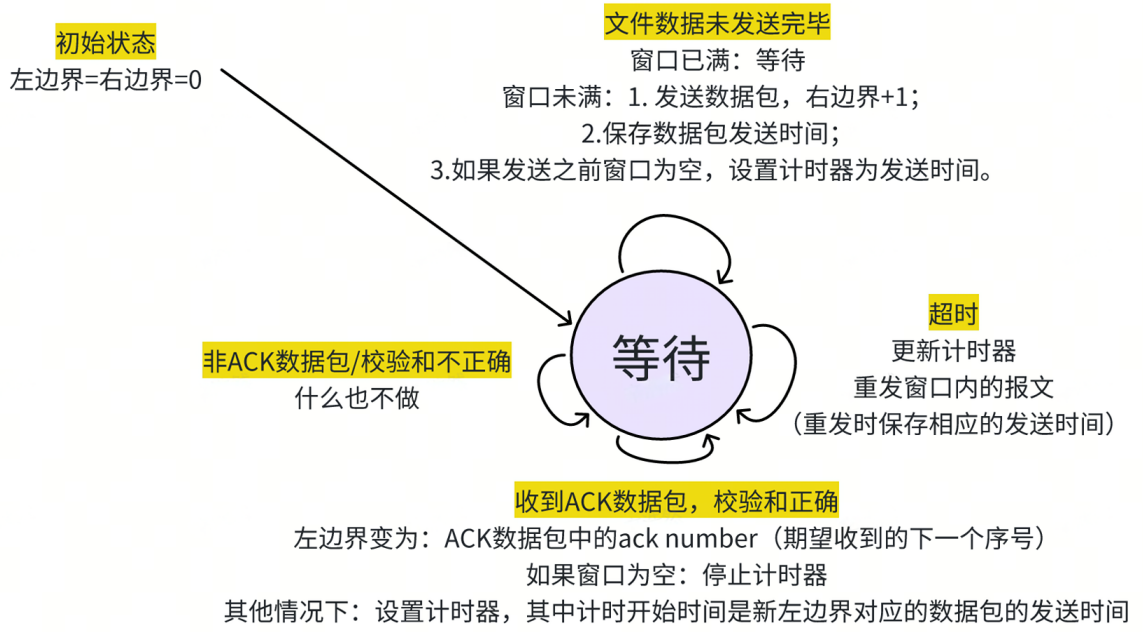


当发送端发送一个数据包后，右边界右移一个单位。

接收端回复的ACK报文中包含期待收到的下一个数据包的序号。当接收到一个数据包的ACK时，发送端所确认的数据包序号之前的所有数据包都被确认，左边界置于新的最小但未确认数据包的序号。

发送端状态机

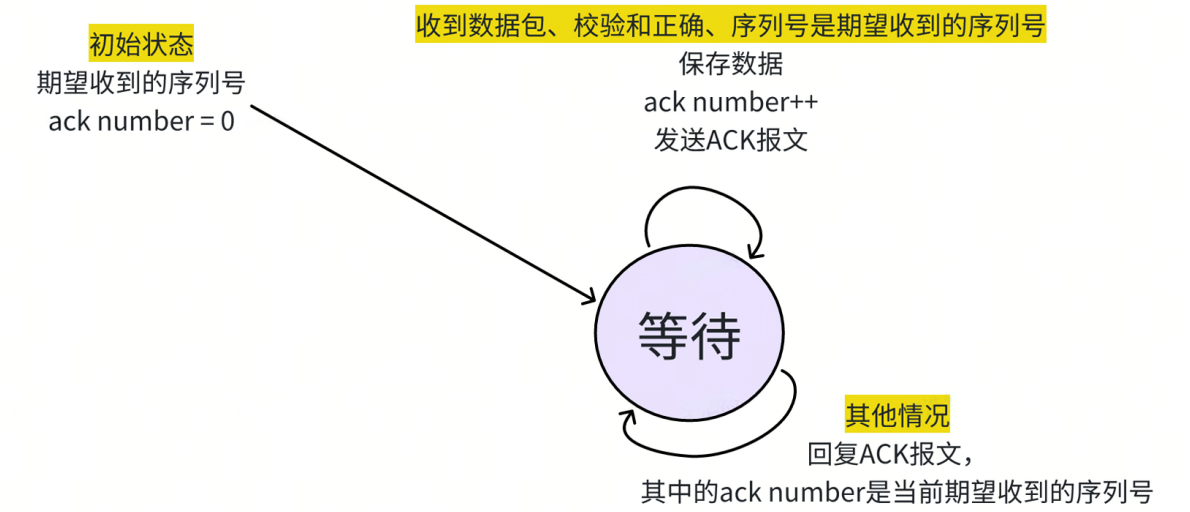
发送端状态机如图所示：



- 在文件数据未发送完毕的情况下：如果**窗口未满**，则发送序号为右边界的数据包，发送的数据包就是右边界对应的分组。
  - 保存该数据包的发送时间；
  - 如果发送前窗口为空，也就是当前数据包是第一个未确认数据包，将**定时器时间设置为该数据包的发送时间**；
  - **更新窗口右边界**，右边界加一。
- 如果接收到接收端发送的ACK报文且校验和正确，ACK报文中包含一个确认序列号，确认序列号是接收端期望收到的下一个数据包的序号，**因此该确认序号之前的所有序列号都已确认**。**将左边界置为发来的ACK报文中的确认序列号**。
  - 如果窗口为空，**停止计时器**；
  - 其他情况下，**启动计时器**，计时器的开始时间是最新左边界对应的数据包的发送时间。
- 如果超时，重发窗口内的报文，**更新计时器**，并更新重发的每条报文的发送时间。

通过在接收、发送报文时计时器的设置，**计时器的开始时间总为当前窗口最早发送但未确认的数据包的发送时间**，一旦超时，就需要重发当前窗口内的所有报文。

接收端状态机



接收端：接收端维护期望收到的序列号，为已确认序列号+1。如果一个序号为 `n` 的分组被正确接收到，并且按序(即上次交付给上层的数据是序号为 `n - 1` 的分组)，则接收方为分组 `n` 发送ACK，期望收到的序列号为 `n + 1` ，并将该分组中的数据部分交付到上层。在所有其他情况下，接收方丢弃该分组，并为最近按序接收的分组重新发送ACK。

2.5 断开连接

在获取传输文件路径时，如果用户输入 `exit` ，客户端主动发起断开连接。

- 客户端发送第一次挥手 `FIN + ACK` 报文。客户端发送后，等待服务器发来的第二次挥手，如果超时未收到则重发；
- 服务器收到第二次挥手后，回复 `ACK` 报文，进入等待期，等待一段时间再退出。在这段时间里，如果收到来自客户端的第一次挥手，就回复ACK报文。

3 代码实现

数据包定义、计算校验和、套接字初始化、建立与断开连接、读取文件数据和保存文件的代码实现与实验3-1基本一致，本次报告就不再赘述。接下来重点关注文件传输协议的实现以及相应的日志输出。

3.1 项目文件结构

```
|- Makefile : Windows 平台下使用 mingw32-make 执行
|- include
|    |- msg.h：数据包结构体及接口
|    |- socket.h：建立udp连接
|    |- router.h：模拟丢包与延时
|    |- log.h：写日志文件与输出
|    |- timer.h：计时器
|- client.cpp：发送端
|- server.cpp：接收端
```

3.2 模拟丢包与延时

路由器程序只能对来自发送端的数据包进行丢包或者延时处理后发给接收端，对接收端发来的包不做处理，直接转发给发送端。而累积确认的优势还体现在当接收端丢失一个ACK时，发送端不会因为单个确认的丢失而阻塞，它可以根据下一个ACK确认所有已成功接收的数据包。

因此，手动编写程序模拟丢包与延时，并且具有相应的日志输出。基本思想类似于Hooking，就是每次发送都调用 `sendWithRegularLoss` 函数，在该函数内先进行模拟丢包和延时的操作，只有不丢包的情况下去调用真正的 `sendto` 函数。

代码实现的基本思路是：每次调用时生成小于1的随机数，如果生成的随机数小于丢包率，则模拟丢包。如果设置了丢包，并且当前随机数满足丢包条件，则函数返回 `true` 表示数据包未发送成功；否则，需要发送数据包。为了避免阻塞主线程，通过异步任务引入延时，等待一定时间后使用 `sendto` 发送数据，最后记录成功发送的日志并返回 `false` 。具体完整代码可参见 `router.h` 。

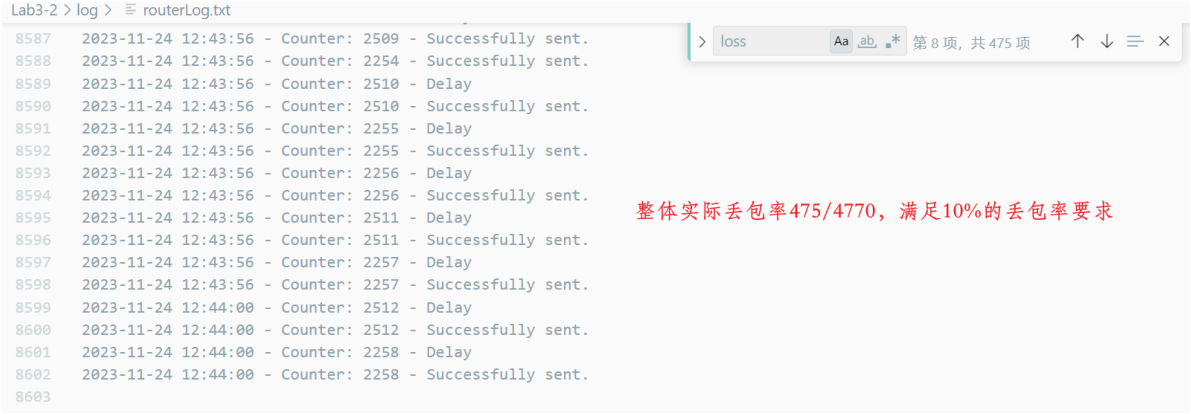
```
bool setMiss = true;    // 是否开启丢包
bool setDelay = true;   // 是否开启延迟
// 用户可以输入想要设置的丢包率和延时
double missRate = 0.05; // 默认丢包率为 5%
int delay = 5;          // 默认延时 5 毫秒

random_device rd;
mt19937 gen(rd());
uniform_real_distribution<> dis(0.0, 1.0); // 生成范围在 [0.0, 1.0) 之间的随
机数

// 调用 sendto 函数，在发送前模拟延时和丢包
bool sendWithRegularLoss(SOCKET sock, const char *buffer, size_t length,
int flags,
                         const struct sockaddr *destAddr, int addrlen)
```

```
{
    static int counter = 0;
    counter++;
    // 丢包
    if (setMiss && (dis(gen) < missRate))
    {
        logToFile("Simulating packet loss. Packet not sent.", counter); //
日志输出
        return true;
    }
    // 异步任务，延迟
    auto future = async(launch::async, [&sock, buffer, length, flags,
destAddr, addrlen]()
                        {
    if (setDelay) {
        this_thread::sleep_for(chrono::milliseconds(delay)); // 等待延时时
间
        logToFile("Delay", counter);//日志输出
    }
    sendto(sock, buffer, length, flags, destAddr, addrlen);
    logToFile("Successfully sent.", counter); }); // 日志输出
    return false;
}
```

这是进行一次文件传输时的模拟丢包延迟的日志输出：接收端和发送端分别计数。设置丢包率为10%，每次发送都有时延。



3.3 计时器

定义计时器 `Timer` 类，在构造函数中会启动一个计时线程，发送端定义一个计时器类对象，可以设置开始时间、启动计时线程，该线程在超时时会设置超时标志位，传递给发送端的超时处理线程。

计时器提供接口函数：

- `start()`：设置当前时间为起始时间；
- `start(clock_t now)`：设置给定的时间点为起始时间；
- `stop()`：停止计时器；
- `isTimeout()`：返回布尔值是否超时；
- `stopTimer`：终止计时线程，用于程序退出时；

具体完整代码可参见 `timer.h` 。

计时器的具体计时逻辑在 `timerFunction` 线程函数中实现。它通过计时器是否启动且检测时间差是否超过预定的超时时间来判断是否发生超时，并在超时时将相应的标志位设置为 `true` 。

```
while (!stopTimerThread)
{
    if (isTiming)
    {
        clock_t currentTime = clock();
        double duration = static_cast<double>(currentTime - startTime) /
CLOCKS_PER_SEC;

        // 超过预定的超时时间
        if (duration >= timeOut)
        {
            lock_guard<mutex> lock(mtx);
            timeout = true;
        }
```

```
        else
        {
            lock_guard<mutex> lock(mtx);
            timeout = false;
        }
    }
}
```

3.4 发送端

发送端定义窗口大小 `MAXWIN`，维护窗口的左右边界与发送序列号。发送序列号标识每个包，每按序发送一个文件数据包，发送序列号增一。左边界 `leftWin` 代表最早未确认数据包的序号，右边界 `rightWin` 代表最小的未使用序号(即下一个待发送数据包的序号)。

首先，定义一个函数，对给定的序号 `i`，传输文件中按 `MSS` 划分的第 `i` 个数据包。这样，后面发送数据包和重传时就可以直接调用该函数并传入需要发送的序号 `i`。

此外，定义一个大小与传输轮数(即文件大小/ `MSS` 向上取整)相同、`clock_t` 类型的数组 `times`，存储每个数据包的发送时间，以方便超时判定与处理。

发送数据包

发送方发送数据包时，开启循环，每次循环首先检查发送窗口是否已满和是否已经发送完毕。

- 如果**窗口未满且未发送完毕**，则发送序号为右边界 `rightWin` 的数据包，发送的数据包就是右边界对应的分组。
    - 记录该数据包的发送时间按序号存至数组中；
    - 如果发送前窗口为空，也就是当前数据包是第一个未确认数据包，将**定时器时间设置为该数据包的发送时间**；
    - **更新窗口右边界**。
- 若条件不满足，不做发送处理，等待至条件满足；
- 如果所有包都已发送且被确认，结束循环，终止计时器，结束其他线程。

下面是发送端发送数据包时的核心代码：

```
while (1)
{
    // 窗口未满，未发送完，可以发送
    if (rightWin - leftWin < MAXWIN && rightWin < rounds)
    {
        mtx.lock();
        sendFilePackage(rightWin);
```

```
            times[rightWin] = clock(); // 记录发送的包的发送时间
            if (leftWin == rightWin)
            {
                timer.setStart(times[rightWin]); // 如果窗口为空的时候发的包，设置计时器为相应时间
            }
            rightWin++; // 发送后，右窗口右移
            logger.log("[Window][AFTER SENT] LEFT: %d, RIGHT: %d", leftWin + beforeSendNum, rightWin + beforeSendNum);
            mtx.unlock();
        }
        if (leftWin == rounds) // 当所有包都发送结束后，结束循环
        {
            mtx.lock();
            timer.stop();
            mtx.unlock();
            break;
        }
    }
```

接收ACK线程

创建线程接收ACK报文。GBN协议中，对数据包的确认采取**累积确认**的方式。当收到确认号为 `n` 的ACK报文，在 `n` 小于等于当前右边界的情况下，则说明 `n` 之前(不包括 `n`)的数据包都被确认，将左边界修改为 `n` 。

- 如果窗口调整后，窗口为空，**停止计时器**；
- 其他情况下，启动计时器，**计时起始时间是左边界序号对应的报文的发送时间**，也就是最早已发送但未确认的报文的发送时间。

下面是接受ACK线程的核心代码：

```
while (1)
{
    int bytesRecv = recvfrom(clientSock, recvFileBuf, sizeof(recvFileBuf), 0, (SOCKADDR *)&serverAddr, &sockaddrSize);

    if (bytesRecv != SOCKET_ERROR)
    {
        recvMsg->reset();
        memcpy(recvMsg, recvFileBuf, bytesRecv);
        // 判断标志位 ACK 和校验和
        if (isValidACK(recvMsg))
        {
            mtx.lock();
```

```
            if (recvMsg->header.getAckNum() <= rightWin + beforeSendNum)
            {
                leftWin = recvMsg->header.getAckNum() - beforeSendNum;
                logMsg(recvMsg);
                logger.log("[Window][AFTER RECV] LEFT: %d, RIGHT: %d",
leftWin + beforeSendNum, rightWin + beforeSendNum);

                if (leftWin == rightWin)
                {
                    // 窗口空，停止计时器
                    timer.stop();
                }
                else
                {
                    // 否则，启动计时器
                    timer.setStart(times[leftWin]);
                }
            }
            mtx.unlock();
        }
    }
}
```

超时线程

创建线程，用于处理超时重传。在循环中检查定时器是否超时来确定是否需要进行重传操作。当超时发生时，**先暂停计时器**，然后**重传窗口内的所有包**，在重传第一个包的时候再**设置计时器**。在实现中，直接将右边界拉回至左边界，即将**窗口大小重置为0**，接下来发送线程会进行发送数据包的处理。

```
while (1)
{
    if (timer.isTimeout())
    {
        mtx.lock();
        timer.stop();
        logger.log("[RETRANSMIT] FROM %d TO %d", leftWin + beforeSendNum,
rightWin + beforeSendNum);
        rightWin = leftWin;
        mtx.unlock();
    }
}
```

线程锁

在前面发送、接收、超时线程中，都涉及到了窗口边界、日志对象、计时器等共享变量的读写。为了确保在任何时刻只有一个线程访问共享变量，因此定义互斥量 `mtx` 。在每次访问窗口边界、日志对象、计时器等共享变量前加锁 `lock` ，在访问结束后，释放锁 `unclock` 。具体加锁和解锁的时机可见前面线程的代码。

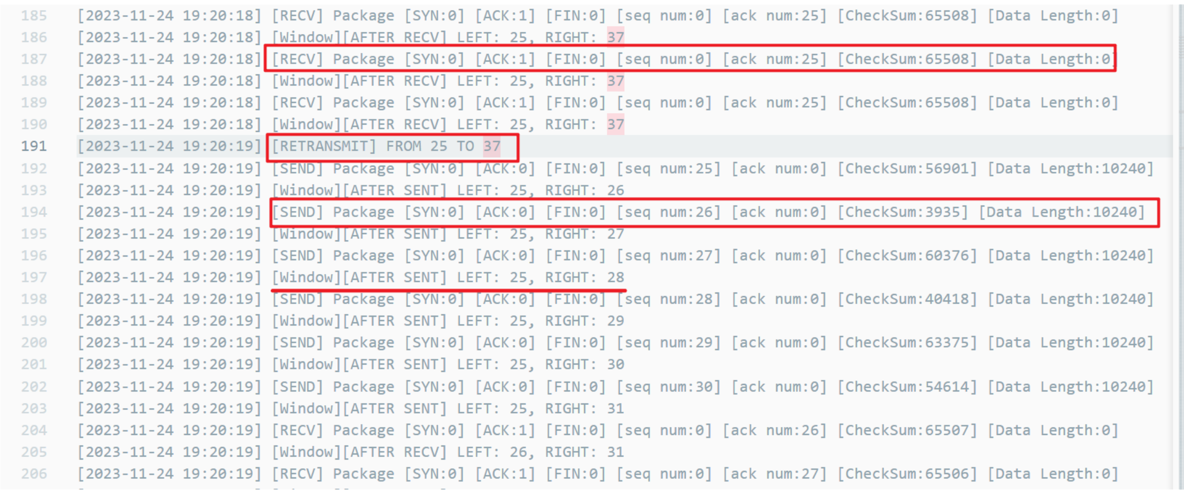
3.5 接收端

接收端对发来的数据包回复ACK报文。如果一个序号为 `n` 的数据包被正确接收到，并且按序(即是接收端的期望收到的序列号，上次收到的数据是序号为 `n-1` 的数据包)，则接收端为数据包 `n` 发送一个ACK，并将数据包中的数据部分保存至文件。在所有其他情况下，接收端丢弃数据包，并为最近按序接收的数据包重新发送ACK。

```
while (1)
{
    // 接收数据
    int bytesRecv = recvfrom(serverSock, recvBuf, sizeof(recvBuf), 0,
(SOCKADDR *)&clientAddr, &sockaddrSize);
    if (bytesRecv != SOCKET_ERROR)
    {
        recvMsg->reset();
        memcpy(recvMsg, recvBuf, bytesRecv);
        // 收到的是正确的文件数据包
        if (recvMsg->isValid())
        {
            logMsg(recvMsg);
            // 收到失序数据包，回复最近的有序的ACK序号
            if (recvMsg->header.getSeqNum() != initialAckNum)
            {
                sendflags(ACK, initialSeqNum, initialAckNum, true);
            }
            else if (recvMsg->header.getSeqNum() == initialAckNum &&
recvMsg->header.getLength())
            {
                fileStream.write(recvMsg->data, recvMsg-
>header.getLength());
                remainingSize -= recvMsg->header.getLength();
                initialAckNum++;
                sendflags(ACK, initialSeqNum, initialAckNum); // 发送ACK
                break;
            }
        }
    }
}
```

3.6 日志输出

整体输出内容包括：接收/发送的数据包格式信息、序列号、发送/接收时间、发送/接收后的窗口变化情况、超时重传的数据包序号等，如下图所示。



```
[2023-11-24 19:20:18] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:25] [CheckSum:65508] [Data Length:0]
[2023-11-24 19:20:18] [Window][AFTER RECV] LEFT: 25, RIGHT: 37
[2023-11-24 19:20:18] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:25] [CheckSum:65508] [Data Length:0]
[2023-11-24 19:20:18] [Window][AFTER RECV] LEFT: 25, RIGHT: 37
[2023-11-24 19:20:18] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:25] [CheckSum:65508] [Data Length:0]
[2023-11-24 19:20:18] [Window][AFTER RECV] LEFT: 25, RIGHT: 37
[2023-11-24 19:20:19] [RETRANSMIT] FROM 25 TO 37
[2023-11-24 19:20:19] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:25] [ack num:0] [CheckSum:56901] [Data Length:10240]
[2023-11-24 19:20:19] [Window][AFTER SENT] LEFT: 25, RIGHT: 26
[2023-11-24 19:20:19] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:26] [ack num:0] [CheckSum:3935] [Data Length:10240]
[2023-11-24 19:20:19] [Window][AFTER SENT] LEFT: 25, RIGHT: 27
[2023-11-24 19:20:19] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:27] [ack num:0] [CheckSum:60376] [Data Length:10240]
[2023-11-24 19:20:19] [Window][AFTER SENT] LEFT: 25, RIGHT: 28
[2023-11-24 19:20:19] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:28] [ack num:0] [CheckSum:40418] [Data Length:10240]
[2023-11-24 19:20:19] [Window][AFTER SENT] LEFT: 25, RIGHT: 29
[2023-11-24 19:20:19] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:29] [ack num:0] [CheckSum:63375] [Data Length:10240]
[2023-11-24 19:20:19] [Window][AFTER SENT] LEFT: 25, RIGHT: 30
[2023-11-24 19:20:19] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:30] [ack num:0] [CheckSum:54614] [Data Length:10240]
[2023-11-24 19:20:19] [Window][AFTER SENT] LEFT: 25, RIGHT: 31
[2023-11-24 19:20:19] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:26] [CheckSum:65507] [Data Length:0]
[2023-11-24 19:20:19] [Window][AFTER RECV] LEFT: 26, RIGHT: 31
[2023-11-24 19:20:19] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:27] [CheckSum:65506] [Data Length:0]
```

在多线程的情况下，由于 `cout` 语句不是原子操作，且终端输出不方便验证与检查，因此定义日志输出类 `Logger` 。日志类的核心代码参见 `log.h` 。

该类主要类成员包括日志线程、日志消息队列、条件变量、文件输出流等。日志类的构造函数接收日志路径，打开日志文件、创建日志线程。日志线程在后台异步运行，从消息队列中获取日志消息并写入日志文件，同时输出到控制台。析构函数负责停止日志线程并关闭日志文件。

提供接口 `log` 函数，将输出消息加入队列，并支持格式化字符串和可变参数列表。

在日志线程的异步写入函数 `async_write` 中，通过条件变量等待新的日志消息或者停止信号。一旦满足条件，如果有新的日志消息，线程会取出队列中的消息并写入日志文件和控制台。线程在收到停止信号时退出循环，结束线程函数。

```cpp
void Logger::async_write()
{
    while (true)
    {
        unique_lock<mutex> lock(mtx);

        // 执行条件：有数据或者需要停止线程；否则就一直等待
        condition.wait(lock, [this]
                       { return !logQueue.empty() || stopLog; });

        // 检查是否是停止日志的时候
        if (stopLog && logQueue.empty())
        {
            break;
        }
```

```
            // 从队列中取出最前面的消息
            string message = logQueue.front();
            logQueue.pop();
            lock.unlock();

            // 写入日志文件并输出
            logFile << message << endl;
            cout << message << endl;
        }
    }
```

主线程在调用日志记录函数时，只需将日志消息推送到队列中，而不必等待日志写入文件完成。这种异步写入的设计可以提高主线程的响应速度，因为主线程不会因为日志写入而被阻塞。

通过使用互斥锁和条件变量，确保了对日志队列的访问是线程安全的，防止多个线程同时修改日志队列。

4 文件传输测试

设置每个数据包大小为10240字节，发送端窗口大小为8，超时时间为500ms，丢包率为5%，延时为10ms，测试文件传输。以下是测试文件的实验结果。

1.jpg

设置截图：

```
PS D:\NKU\code\vscode\Lab3-2> ./client
[2023-11-24 00:16:50] Please input the window size.
8
[2023-11-24 00:16:52] Please input miss rate.
0.05
[2023-11-24 00:16:55] Please input delay time.
10
[2023-11-24 00:16:56] [SEND] Package [SYN:1] [ACK:0] [FIN
```

成功传输1.jpg，传输时间为6.53秒，吞吐率为每秒28.4705万字节。

```
[2023-11-24 00:17:15] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:183] [CheckSum:65350] [Data Length:0
]
[2023-11-24 00:17:15] [RECV] Package [SYN:0] [ACK:0] [FIN:0
] [seq num:183] [ack num:0] [CheckSum:7539] [Data Length:39
13]
[2023-11-24 00:17:15] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:184] [CheckSum:65349] [Data Length:0
]
[2023-11-24 00:17:17] [RECV] Package [SYN:0] [ACK:1] [FIN:1
] [seq num:184] [ack num:0] [CheckSum:65348] [Data Length:0
]
[2023-11-24 00:17:17] Successful first handwaving.
[2023-11-24 00:17:17] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:185] [CheckSum:65348] [Data Length:0
]
[2023-11-24 00:17:18] Bye.
PS D:\NKU\code\vscode\Lab3-2>
```

```
:0]
[2023-11-24 00:17:15] [Window][AFTER RECV] LEFT: 184, RIGH
T: 184
[2023-11-24 00:17:15] Successfully send file 1.jpg.
[2023-11-24 00:17:15] Transmit rounds: 182
[2023-11-24 00:17:15] Total time: 6.53 s
[2023-11-24 00:17:15] Throughput: 284705.39 byte/s
[2023-11-24 00:17:15] Please enter exit to exit.
exit
[2023-11-24 00:17:17] [SEND] Package [SYN:0] [ACK:1] [FIN:
1] [seq num:184] [ack num:0] [CheckSum:65348] [Data Length
:0]
[2023-11-24 00:17:17] [RECV] Package [SYN:0] [ACK:1] [FIN:
0] [seq num:0] [ack num:185] [CheckSum:65348] [Data Length
:0]
[2023-11-24 00:17:17] Successful second handwaving.
[2023-11-24 00:17:17] Bye
PS D:\NKU\code\vscode\Lab3-2>
```

传输后的文件：

2.jpg

设置截图：

```
PS D:\NKU\code\vscode\Lab3-2> ./client
[2023-11-24 00:18:42] Please input the window size.
8
[2023-11-24 00:18:45] Please input miss rate.
0.05
[2023-11-24 00:18:46] Please input delay time.
10
[2023-11-24 00:18:47] [SEND] Package [SYN:1] [ACK:0] [FIN:
0] [seq num:0] [ack num:0] [CheckSum:65531] [Data Length:0
```

成功传输2.jpg，传输时间为32.02秒，吞吐率为每秒18.448994万字节。

```
[2023-11-24 00:19:36] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:579] [CheckSum:64954] [Data Length:0
]
[2023-11-24 00:19:36] Simulating package loss. This package
 not sent.
[2023-11-24 00:19:36] [Disordered ACK]
[2023-11-24 00:19:36] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:579] [CheckSum:64954] [Data Length:0
]
[2023-11-24 00:19:40] [RECV] Package [SYN:0] [ACK:1] [FIN:1
] [seq num:579] [ack num:0] [CheckSum:64953] [Data Length:0
]
[2023-11-24 00:19:40] Successful first handwaving.
[2023-11-24 00:19:40] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:580] [CheckSum:64953] [Data Length:0
]
[2023-11-24 00:19:41] Bye.
PS D:\NKU\code\vscode\Lab3-2>
```

```
:0]
[2023-11-24 00:19:36] [Window][AFTER RECV] LEFT: 579, RIGH
T: 579
[2023-11-24 00:19:36] Successfully send file 2.jpg.
[2023-11-24 00:19:36] Transmit rounds: 577
[2023-11-24 00:19:36] Total time: 32.02 s
[2023-11-24 00:19:36] Throughput: 184489.94 byte/s
[2023-11-24 00:19:36] Please enter exit to exit.
exit
[2023-11-24 00:19:40] [SEND] Package [SYN:0] [ACK:1] [FIN:
1] [seq num:579] [ack num:0] [CheckSum:64953] [Data Length
:0]
[2023-11-24 00:19:40] [RECV] Package [SYN:0] [ACK:1] [FIN:
0] [seq num:0] [ack num:580] [CheckSum:64953] [Data Length
:0]
[2023-11-24 00:19:40] Successful second handwaving.
[2023-11-24 00:19:40] Bye
PS D:\NKU\code\vscode\Lab3-2>
```

传输后的文件：

3.jpg

```
PS D:\NKU\code\vscode\Lab3-2> ./client
[2023-11-24 00:20:44] Please input the window size.
8
[2023-11-24 00:20:45] Please input miss rate.
0.05
[2023-11-24 00:20:47] Please input delay time.
10
[2023-11-24 00:20:48] [SEND] Package [SYN:1] [ACK:0] [FIN:
```

成功传输3.jpg，传输时间为48.98秒，吞吐率为每秒24.475679万字节。

```
] [seq num:0] [ack num:1170] [CheckSum:64363] [Data Length:
0]
[2023-11-24 00:21:49] [RECV] Package [SYN:0] [ACK:0] [FIN:0
] [seq num:1170] [ack num:0] [CheckSum:55278] [Data Length:
8674]
[2023-11-24 00:21:49] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:1171] [CheckSum:64362] [Data Length:
0]
[2023-11-24 00:22:09] [RECV] Package [SYN:0] [ACK:1] [FIN:1
] [seq num:1171] [ack num:0] [CheckSum:64361] [Data Length:
0]
[2023-11-24 00:22:09] Successful first handwaving.
[2023-11-24 00:22:09] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:1172] [CheckSum:64361] [Data Length:
0]
[2023-11-24 00:22:10] Bye.
PS D:\NKU\code\vscode\Lab3-2>
```

```
[2023-11-24 00:21:49] [Window][AFTER RECV] LEFT: 1171, RIG
HT: 1171
[2023-11-24 00:21:49] Successfully send file 3.jpg.
[2023-11-24 00:21:49] Transmit rounds: 1169
[2023-11-24 00:21:49] Total time: 48.98 s
[2023-11-24 00:21:49] Throughput: 244756.79 byte/s
[2023-11-24 00:21:49] Please enter exit to exit.
exit
[2023-11-24 00:22:09] [SEND] Package [SYN:0] [ACK:1] [FIN:
1] [seq num:1171] [ack num:0] [CheckSum:64361] [Data Lengt
h:0]
[2023-11-24 00:22:09] [RECV] Package [SYN:0] [ACK:1] [FIN:
0] [seq num:0] [ack num:1172] [CheckSum:64361] [Data Lengt
h:0]
[2023-11-24 00:22:09] Successful second handwaving.
[2023-11-24 00:22:09] Bye
PS D:\NKU\code\vscode\Lab3-2>
```

传输后的文件：

helloworld.txt

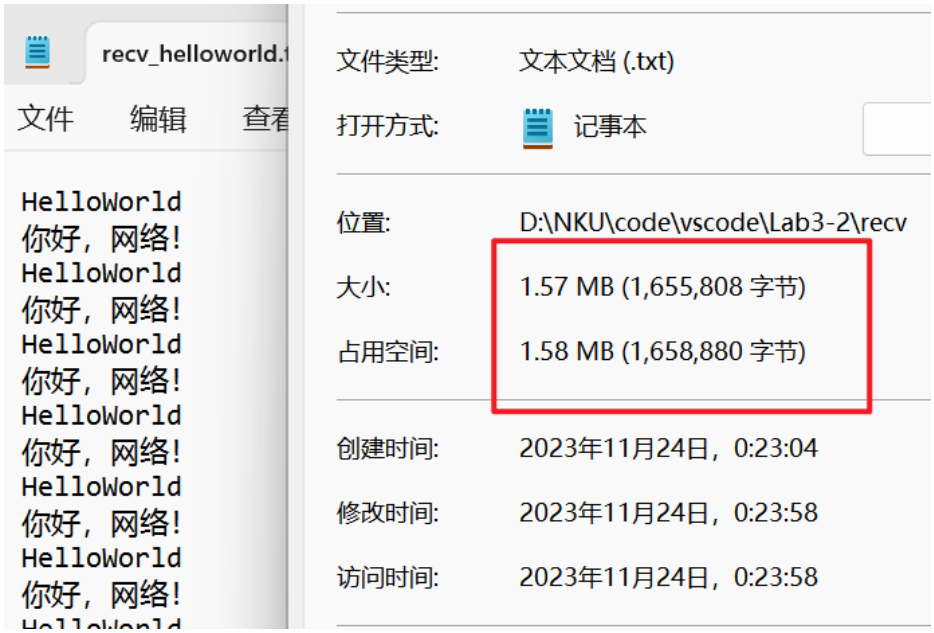
```
PS D:\NKU\code\vscode\Lab3-2> ./client
[2023-11-24 00:23:29] Please input the window size.
8
[2023-11-24 00:23:35] Please input miss rate.
0.05
[2023-11-24 00:23:37] Please input delay time.
10
[2023-11-24 00:23:39] [SEND] Package [SYN:1] [ACK:0] [
```

成功传输helloworld.txt，传输时间为5.71秒，吞吐率为每秒29.053959万字节。

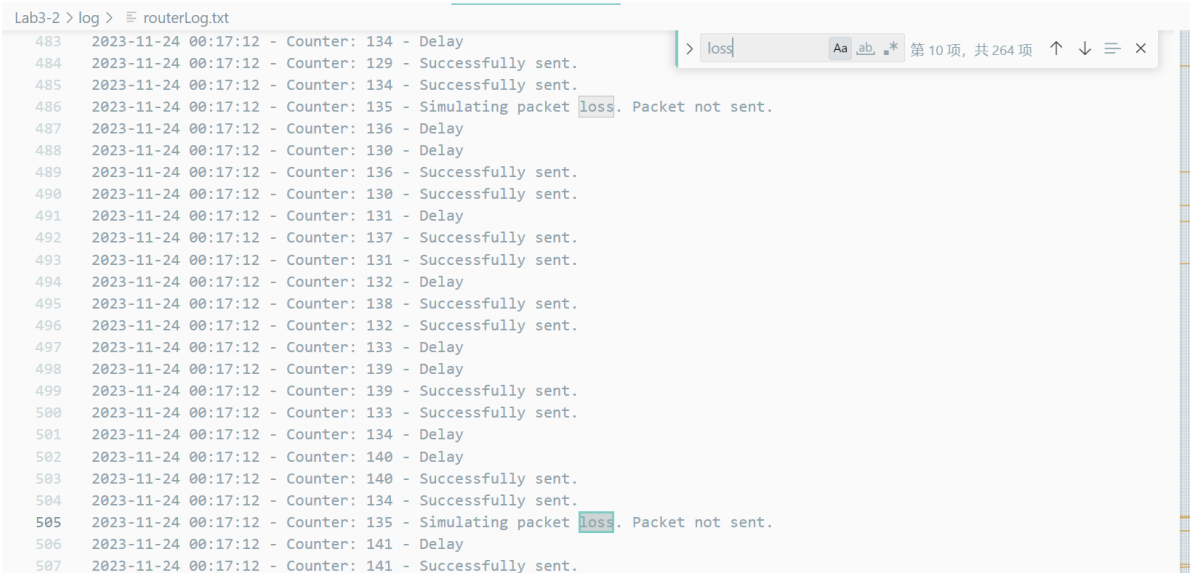
```
[2023-11-24 00:23:58] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:163] [CheckSum:65370] [Data Length:0
]
[2023-11-24 00:23:58] [RECV] Package [SYN:0] [ACK:0] [FIN:0
] [seq num:163] [ack num:0] [CheckSum:9652] [Data Length:71
68]
[2023-11-24 00:23:58] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:164] [CheckSum:65369] [Data Length:0
]
[2023-11-24 00:24:00] [RECV] Package [SYN:0] [ACK:1] [FIN:1
] [seq num:164] [ack num:0] [CheckSum:65368] [Data Length:0
]
[2023-11-24 00:24:00] Successful first handwaving.
[2023-11-24 00:24:00] [SEND] Package [SYN:0] [ACK:1] [FIN:0
] [seq num:0] [ack num:165] [CheckSum:65368] [Data Length:0
]
[2023-11-24 00:24:01] Bye.
PS D:\NKU\code\vscode\Lab3-2>
```

```
[2023-11-24 00:23:58] [Window][AFTER RECV] LEFT: 164, RIGH
T: 164
[2023-11-24 00:23:58] Successfully send file helloworld.tx
t.
[2023-11-24 00:23:58] Transmit rounds: 162
[2023-11-24 00:23:58] Total time: 5.71 s
[2023-11-24 00:23:58] Throughput: 290539.59 byte/s
[2023-11-24 00:23:58] Please enter exit to exit.
exit
[2023-11-24 00:24:00] [SEND] Package [SYN:0] [ACK:1] [FIN:
1] [seq num:164] [ack num:0] [CheckSum:65368] [Data Length
:0]
[2023-11-24 00:24:00] [RECV] Package [SYN:0] [ACK:1] [FIN:
0] [seq num:0] [ack num:165] [CheckSum:65368] [Data Length
:0]
[2023-11-24 00:24:00] Successful second handwaving.
[2023-11-24 00:24:00] Bye
PS D:\NKU\code\vscode\Lab3-2>
```

传输后的文件：



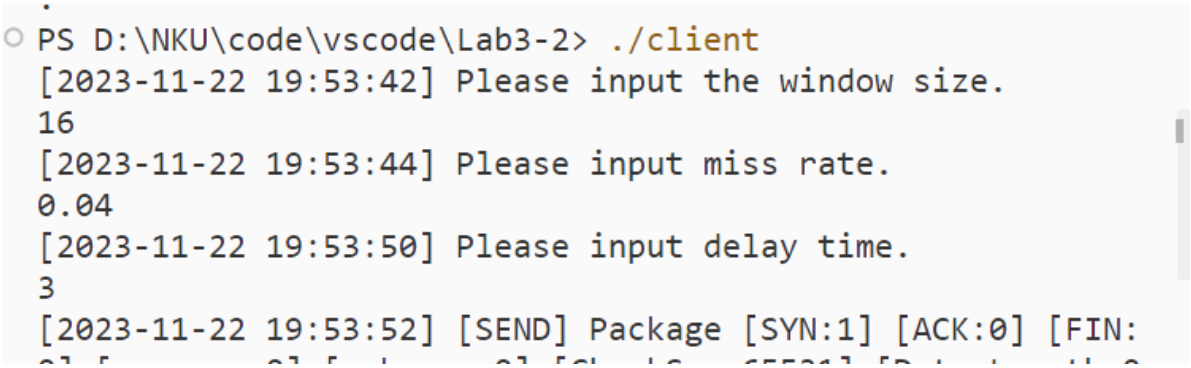
前面测试的丢包信息都在日志文件有所记录；在相应的客户端和服务器日志中也会有丢包记录。接下来进行一次传输，详细分析日志记录。



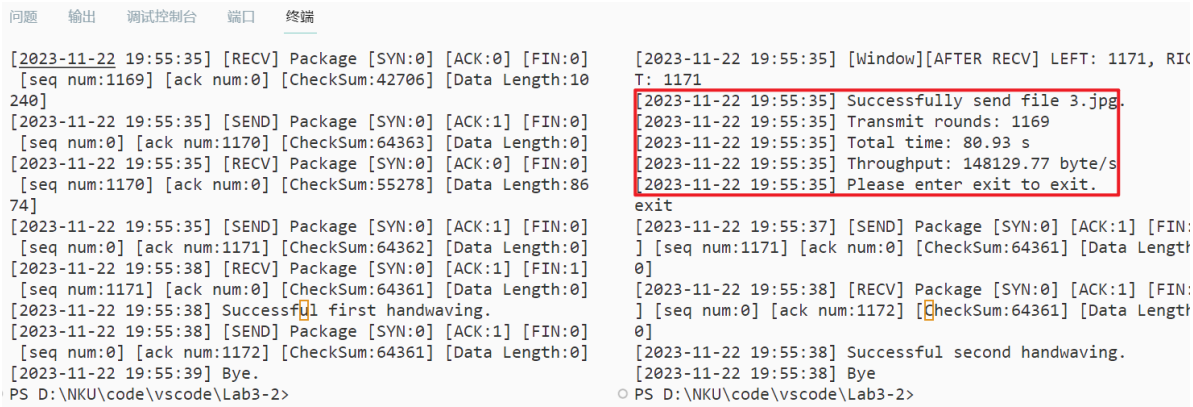
5 传输结果分析

下面给出一次传输结果的详细分析。设置每个数据包大小为10240字节，发送端窗口大小为16，超时时间为500ms，丢包率为4%，延时为3ms。

设置截图：



能够成功传输，传输时间为80.93秒，吞吐率为14.812977万字节每秒。



分析传输过程中的日志文件：

可以发现，在发送端发送数据包后，窗口右边界会增加。

Lab3-2 > log > clientLog.txt

```
[2023-11-22 19:53:42] Please input the window size.
[2023-11-22 19:53:44] Please input miss rate.
[2023-11-22 19:53:50] Please input delay time.
[2023-11-22 19:53:52] [SEND] Package [SYN:1] [ACK:0] [FIN:0] [seq num:0] [ack num:0] [CheckSum:65531] [Data Length:0]
[2023-11-22 19:53:52] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:1] [CheckSum:65532] [Data Length:0]
[2023-11-22 19:53:52] Successful second handshake.
[2023-11-22 19:53:52] Successfully shake hands with server.
[2023-11-22 19:53:52] Please input the filePath to send.
[2023-11-22 19:54:14] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:1] [ack num:0] [CheckSum:32477] [Data Length:32]
[2023-11-22 19:54:14] [Data]: File 3.jpg, Size: 11968994 bytes
[2023-11-22 19:54:14] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:2] [CheckSum:65531] [Data Length:0]
[2023-11-22 19:54:14] Successful send file information.
[2023-11-22 19:54:14] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:2] [ack num:0] [CheckSum:8000] [Data Length:1024
[2023-11-22 19:54:14] [Window][AFTER SENT] LEFT: 2, RIGHT: 3
[2023-11-22 19:54:14] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:3] [ack num:0] [CheckSum:54316] [Data Length:102
[2023-11-22 19:54:14] [Window][AFTER SENT] LEFT: 2, RIGHT: 4
[2023-11-22 19:54:14] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:4] [ack num:0] [CheckSum:53838] [Data Length:102
[2023-11-22 19:54:14] [Window][AFTER SENT] LEFT: 2, RIGHT: 5
[2023-11-22 19:54:14] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:5] [ack num:0] [CheckSum:44329] [Data Length:102
[2023-11-22 19:54:14] [Window][AFTER SENT] LEFT: 2, RIGHT: 6
```

而接收一个有效的数据包ACK后，窗口左边界会增加。

```
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:11] [ack num:0] [CheckSum:44826] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 2, RIGHT: 12
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:3] [CheckSum:65530] [Data Length:0]
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 3, RIGHT: 12
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:12] [ack num:0] [CheckSum:54753] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 3, RIGHT: 13
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:13] [ack num:0] [CheckSum:11458] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 3, RIGHT: 14
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:14] [ack num:0] [CheckSum:11369] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 3, RIGHT: 15
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:15] [ack num:0] [CheckSum:54367] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 3, RIGHT: 16
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:16] [ack num:0] [CheckSum:46367] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 3, RIGHT: 17
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:17] [ack num:0] [CheckSum:27420] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 3, RIGHT: 18
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:18] [ack num:0] [CheckSum:49690] [Data Length:1024
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 3, RIGHT: 19
```

接收

发送

在发送序号为24的数据包时丢包：

```
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:24] [ack num:0] [CheckSum:13895] [Data Length:10240]
[2023-11-22 19:54:15] Simulating package loss. This package not sent.
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 19, RIGHT: 25
```

丢包！

一直没有收到针对序号24的ACK，不断发送数据直到满窗口，期间收到的ACK都是针对于序号23的ACK。

```
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:39] [ack num:0] [CheckSum:42850] [Data Length:1
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
```

24丢包

24的之后发过去的数据包都会回复针对于23的ACK（也就是服务器期待收到24）

到达超时时间后，进行重传，重传窗口内的所有报文。直接将右窗口拉回左窗口，再重新发送。

```
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0]
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:24] [CheckSum:65509] [Data Length:0]
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 24, RIGHT: 40
[2023-11-22 19:54:15] [RETRANSMIT] FROM 24 TO 40        达到预定的超时时间，进行重传！重传窗口内所有报文
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:24] [ack num:0] [CheckSum:13895] [Data Length:10240]
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 24, RIGHT: 25      直接将窗口大小归0，然后重新进行发送
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:25] [ack num:0] [CheckSum:27791] [Data Length:10240]
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 24, RIGHT: 26
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:26] [ack num:0] [CheckSum:62751] [Data Length:10240]
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 24, RIGHT: 27
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:27] [ack num:0] [CheckSum:36937] [Data Length:10240]
```

查看接收端的日志。它在发送针对于序号19的ACK时丢包。

```
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:0] [FIN:0] [seq num:19] [ack num:0] [CheckSum:16983] [Data Length:10240]
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:20] [CheckSum:65513] [Data Length:0]
[2023-11-22 19:54:15] Simulating package loss. This package not sent.
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:0] [FIN:0] [seq num:20] [ack num:0] [CheckSum:29682] [Data Length:10240]
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:21] [CheckSum:65512] [Data Length:0]
```

可以看到，在客户端，收到针对于序号20的ACK时，左窗口直接从19变到了21，累积确认了19和20。

```
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:34] [ack num:0] [CheckSum:62293] [Data Length:10240]
[2023-11-22 19:54:15] [Window][AFTER SENT] LEFT: 19, RIGHT: 35
[2023-11-22 19:54:15] [RECV] Package [SYN:0] [ACK:1] [FIN:0] [seq num:0] [ack num:21] [CheckSum:65512] [Data Length:0]
[2023-11-22 19:54:15] [Window][AFTER RECV] LEFT: 21, RIGHT: 35
[2023-11-22 19:54:15] [SEND] Package [SYN:0] [ACK:0] [FIN:0] [seq num:35] [ack num:0] [CheckSum:60370] [Data Length:10240]
```