计算机网络实验报告

实验名称：Lab3-2 基于UDP服务设计可靠传输协议

学号：2012682　　姓名：韩佳迅

一、实验内容

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**，采用固定窗口大小，支持**累积确认**，完成给定测试文件的传输。

二、协议设计

（一）报文格式

报文格式如下所示：

报文格式分为首部和数据部分：

首部：

- 1—4字节：源IP
- 5—8字节：目的IP
- 9—10字节：源端口号
- 11—12字节：目的端口号
- 13—16字节：序号
- 17—20字节：确认号
- 21—24字节：数据大小
- 25—26字节：标志
- 27—28字节：校验和

数据：

- 剩余字节流是数据部分

报文段的校验：

在报文段里设置校验和字段，发送时设置校验和，收到时检验校验和是否正确。若不正确，说明报文损坏。

（二）数据传输：滑动窗口、累计确认、快速重传

本次实验基于 ***Go-Back-N（GBN）***协议，添加了**三次快速重传**，实现了**滑动窗口**、**累计确认**。

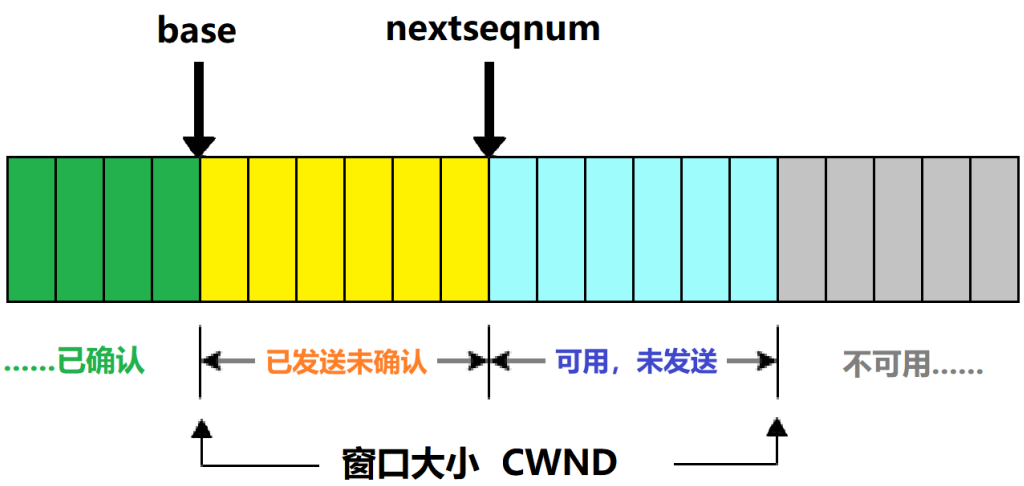
流水线：由于停等机制在每一次发送报文时，都需要等待上个报文的确认报文被接收到，才可以继续发送，因此会产生很长的等待时延，效率低下。而流水线协议则指的是，在确认未返回之前允许发送多个分组，从而提高传输效率。

为了实现流水线协议，在停等机制的基础上需要实现：

- **序列号的扩展：**从0、1二值，需要扩展到更大的范围才够表示
- **发送端、接收端的缓冲区：**由于每次发送不止一个报文，且不确定一次发送的多个报文是否被确认，因此需要在发送端、接收端设置缓冲区。

1. 滑动窗口

发送窗口如下：



实验中的流量控制采用发送端（固定）滑动窗口的方法。

- 设置窗口大小固定为 ***CWND***。
- 设置两个指针，来控制窗口的滑动和下一个序号的发送：
  - ***base：***基序号，是滑动窗口的开始位置，指向已发送为确认的第一个序号，或窗口的第一个序号
  - ***nextseqnum：***指向下一个要发送的序号
- **理想情况：**

  1. 初始时，***base*** 和 ***nextseqnum*** 都指向第一个序号。
  2. 由于此时窗口中还有报文可以继续发送，因此发送端继续发送报文，每发送一个，***nextseqnum*** 右移，直至窗口中可发送的报文都已发送完。

3. 当发送端收到确认报文，整个窗口右移，***base*** 右移（移动到当前已经累计确认的最后一个报文）。
4. 当窗口移动后，新窗口内有了可以发送的报文，则继续发送新报文。
5. 持续上述步骤，直到所有报文发送完成。

- **超时重传：**

  滑动窗口每发送报文时，会设置定时器。当超时未收到报文时，会重传当前窗口内所有已经发送的报文，即 ***base ~ nextseqnum*** 的所有报文，而 base 、nextseqnum 位置不变。（除此之外，还实现了快速重传，见下文）

- **失序：**

  见累计确认部分。

2. 累计确认

接收端：

理想情况：接收端每接收到发来的报文，若该报文的序号 ***等于*** 接收端期待接收的报文序号，则回复一个确认报文，确认号 ack = seq，并接收该报文段，将其交付给上层应用。

失序：当接收端收到了 ***不等于*** 期待序号值的报文，则回复一个确认报文，确认号 ack = 接收端累计确认的最后一个报文号（也就是 期待接收的报文序号 的前一个值），并丢弃该报文段。

发送端：

当发送端收到确认报文时：

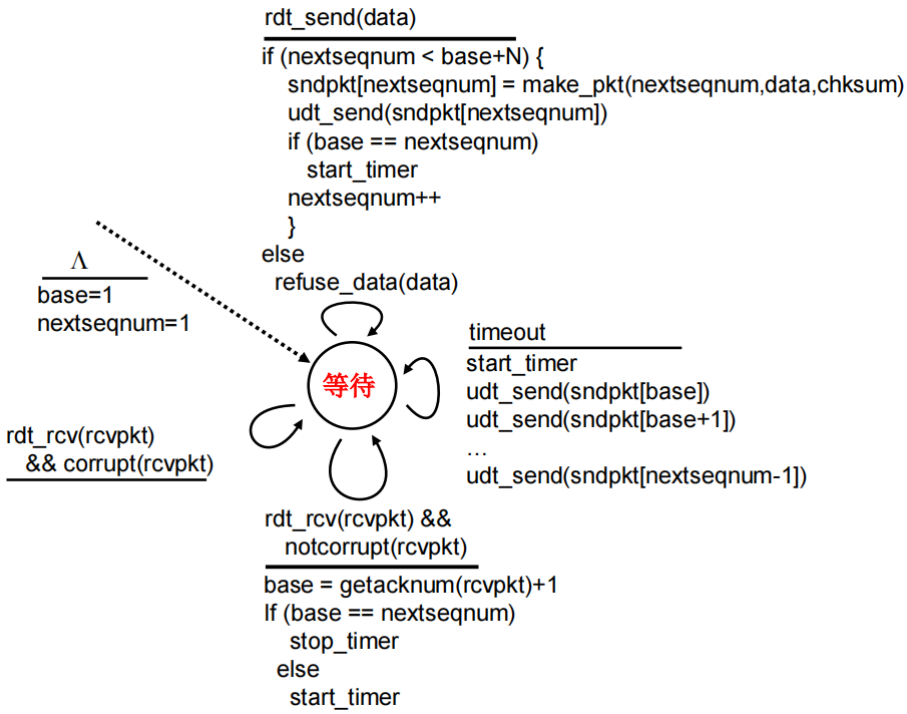
- 首先检查确认报文是否 ***ack < base***，若小于，说明这是上一次发送的失序确认报文，不移动窗口
- 若确认报文 ***ack >= base***，说明这是对此窗口内的确认，***base*** 移动到 ***ack+1*** 的位置，窗口右移

3. 快速重传

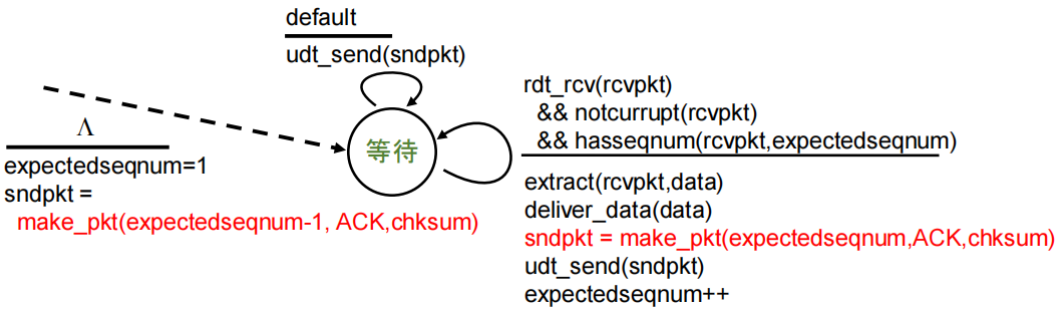
由于接收端每次收到失序的报文时，会回复 ack = 期待值的确认报文，因此当报文丢失或失序时，发送端会连续收到多个重复且冗余的 ACK 报文端。因此，实验中规定，当连续收到三次冗余的 ACK 报文时，可认为报文丢失，需要重传当前窗口内 ***base ~ nextseqnum*** 的所有报文。

GBN状态机

- 发送端：

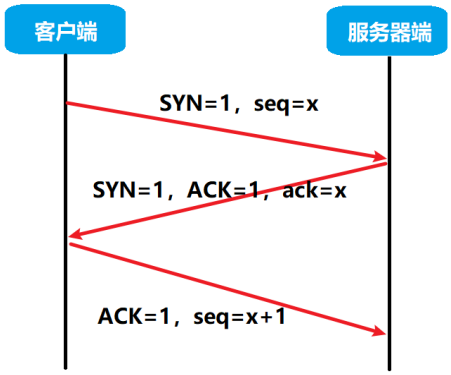


- 接收端：



（三）建立连接：三次握手

本程序在标准三次握手的基础上，进行了部分改进，最终实现的结构示意图如下：

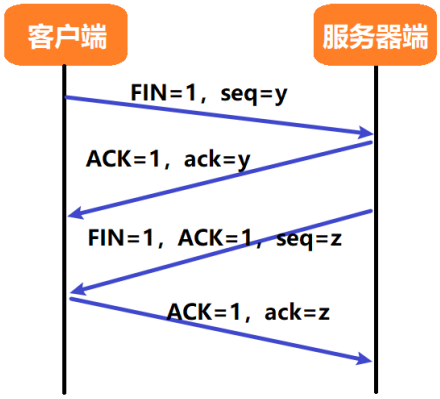


1. 客户端向服务器端发送数据包：
    - SYN=1，seq=x

2. 服务器端向客户端回复数据包：
    - SYN=1，ACK=1
    - ack=x【这里对标准三次握手进行了更改：回复的ack = 上一个包发来的seq】
3. 客户端向服务器端发送数据包：
    - ACK=1
    - seq=x+1【序列号+1】

（四）关闭连接：四次挥手

本程序在标准四次挥手的基础上，进行了部分改进，最终实现的结构示意图如下：



1. 客户端向服务器端发送数据包：
    - FIN=1，seq=y
2. 服务器端向客户端回复数据包：
    - ACK=1，ack=y【回复的ack = 上一个包发来的seq】
3. 服务器端向客户端发送数据包：
    - FIN=1，ACK=1
    - seq=z
4. 客户端向服务器端发送数据包：
    - ACK=1，ack=z【回复的ack = 上一个包发来的seq】

三、各模块功能与具体实现

（一）报文段

结构体实现报文段

- 首部和数据部分由结构体的属性实现
    - 注意：标志位字段使用一个unsigned short变量 ***flag*** 实现
    - 设置SYN、ACK、FIN字段分别为0x1、0x2、0x4，置位时直接加到 ***flag*** 上即可

- 功能实现：
  - 设置校验和
  - 检验校验和

UDP校验和

- **设置校验和：**

1. 校验和全部填充为0
2. 剩余数据部分填充为0
3. 按16bit为单位反码求和：
   - 【首部+数据】每16位求和得到一个32位数
   - 如果这个32位的数，高16位不为0，则高16位加低16位再得到一个32位的数
   - 重复直到高16位为0
   - 将低16位取反，得到校验和
4. 得到的16位结果填充到结构体的校验和字段

- **检验校验和：**

1. 按照设置校验和的方式，将接收到的结构体按16bit为单位反码求和
2. 如果得出的结果低16位全1，则校验成功

```
//（该部分使用3-1代码，未做修改）
```

（二）数据传输：滑动窗口、累计确认、快速重传

文件传输总体流程：

- 客户端：
  - 先发送文件名和文件大小，文件名通过报文数据段传递，文件大小通过报文的size字段传递
  - 然后依次发送文件内的数据部分，分为最大装载报文和剩余部分的报文
- 服务器端：
  - 收到文件名和文件大小，根据文件大小计算要收到多少个报文
  - 依次按顺序等待接收，当收到了所有报文即结束接收

客户端：多线程实现

- 读取文件内容：
  - 使用 ifstream 以二进制方式打开文件，将文件内容读到BYTE类型数组中
- 设置两个线程，一个用来发送，一个用于接收：

- **两线程共享变量：**
  - `int base = 0`：基序号
  - `int nextseqnum = 0`：下一个发送的序号
  - `int msgStart`：计时器
  - `bool over = 0`： 表示数据传输是否结束。

    当接收线程收到最后一个报文的ack，表示传输结束。此时设置全局变量为true，则发送线程停止发送。
  - `bool sendAgain = 0`：表示是否需要三次快速重传。

    当接收线程收到三次冗余ACK，则设置变量为true，从而发送线程即可开始重新发送报文段
- **主线程：发送**

  while循环持续准备发送报文：
  - nextseqnum < base + N 时：窗口中有就绪报文，可以发送

    make_pkt 创建要发送的报文并 send 发送

    nextseqnum++ 发送一个报文，可发送的序列号++
  - timeout 时：超时重传

    重传从 base 到 nextseqnum - 1 的所有报文

    重新计时
  - 三次冗余ACK：快速重传

    重传从 base 到 nextseqnum - 1 的所有报文

    重新计时

```
void clientSendFunction_GBN(string filename, SOCKADDR_IN
serverAddr, SOCKET clientSocket)
{
    int startTime = clock();
    //=================先截取文件名（删除无用路径）
    string realname = "";
    for (int i = filename.size() - 1; i >= 0; i--)
    {
        if (filename[i] == '/' || filename[i] == '\\')
            break;
        realname += filename[i];
    }
    realname = string(realname.rbegin(), realname.rend());
    //=================打开文件，读成字节流=================
    ifstream fin(filename.c_str(), ifstream::binary);
    if (!fin) {
        printf("无法打开文件！\n");
```

```
        return;
    }
    //文件读取到fileBuffer
    BYTE* fileBuffer = new BYTE[MaxFileSize];
    unsigned int fileSize = 0;
    BYTE byte = fin.get();
    while (fin) {
        fileBuffer[fileSize++] = byte;
        byte = fin.get();
    }
    fin.close();
    int batchNum = fileSize / MaxMsgSize;//全装满的报文个数
    int leftSize = fileSize % MaxMsgSize;//不能装满的剩余报文大小
    //=================== 创建接受消息线程 ====================
    int msgSum = leftSize > 0 ? batchNum + 2 : batchNum + 1;
    parameters param;
    param.serverAddr = serverAddr;
    param.clientSocket = clientSocket;
    param.msgSum = msgSum;
    HANDLE hThread = CreateThread(NULL, 0,
                    (LPTHREAD_START_ROUTINE)recvThread,
                    &param, 0, 0);
    while (1)
    {
        //rdt_send(data)
        if (nextseqnum < base + N && nextseqnum < msgSum)
        {
            //make_pkt
            Message sendMsg;
            if (nextseqnum == 0)
            {
                sendMsg.SrcPort = ClientPORT;
                sendMsg.DestPort = RouterPORT;
                sendMsg.size = fileSize;
                sendMsg.flag += isName;
                sendMsg.SeqNum = nextseqnum;
                for (int i = 0; i < realname.size(); i++)
                    //填充报文数据段
                    sendMsg.msgData[i] = realname[i];
                //字符串结尾补\0
                sendMsg.msgData[realname.size()] = '\0';
                sendMsg.setCheck();
            }
            else if (nextseqnum == batchNum + 1 && leftSize > 0)
            {
                sendMsg.SrcPort = ClientPORT;
                sendMsg.DestPort = RouterPORT;
                sendMsg.SeqNum = nextseqnum;
```

```
                for (int j = 0; j < leftSize; j++)
                {
                    sendMsg.msgData[j] = fileBuffer[batchNum *
MaxMsgSize + j];
                }
                sendMsg.setCheck();
            }
            else
            {
                sendMsg.SrcPort = ClientPORT;
                sendMsg.DestPort = RouterPORT;
                sendMsg.SeqNum = nextseqnum;
                for (int j = 0; j < MaxMsgSize; j++)
                {
                    sendMsg.msgData[j] = fileBuffer[(nextseqnum -
1) * MaxMsgSize + j];
                }
                sendMsg.setCheck();
            }

            //send_pkt
            sendto(clientSocket,
                   (char*)&sendMsg,
                   sizeof(sendMsg),
                   0,
                   (sockaddr*)&serverAddr,
                   sizeof(SOCKADDR_IN));
            cout << "----client已发送【Seq = "
                 << sendMsg.SeqNum << "】的报文段! " << endl;

            if (base == nextseqnum)
            {
                msgStart = clock();
            }
            nextseqnum++;
            //打印窗口情况
            cout << "【当前窗口情况】 窗口总大小: " << N
                 << ", 已发送但未收到ACK: " << nextseqnum - base
                 << ", 尚未发送: " << N - (nextseqnum - base)
                 << "\n";
        }

        //timeout
        if (clock() - msgStart> MAX_WAIT_TIME || sendAgain)
        {
            if (sendAgain)
                cout << "连续收到三次冗余ACK，快速重传......" << endl;
            //重发当前缓冲区的message
```

```
            Message sendMsg;
            for (int i = 0; i < nextseqnum - base; i++)
            {
                int sendnum = base + i;
                if (sendnum == 0)
                {
                    sendMsg.SrcPort = ClientPORT;
                    sendMsg.DestPort = RouterPORT;
                    sendMsg.size = fileSize;
                    sendMsg.flag += isName;
                    sendMsg.SeqNum = sendnum;
                    for (int i = 0; i < realname.size(); i++)
                        //填充报文数据段
                        sendMsg.msgData[i] = realname[i];
                    //字符串结尾补\0
                    sendMsg.msgData[realname.size()] = '\0';
                    sendMsg.setCheck();
                }
                else if (sendnum == batchNum + 1 && leftSize > 0)
                {
                    sendMsg.SrcPort = ClientPORT;
                    sendMsg.DestPort = RouterPORT;
                    sendMsg.SeqNum = sendnum;
                    for (int j = 0; j < leftSize; j++)
                    {
                        sendMsg.msgData[j] = fileBuffer[batchNum *
MaxMsgSize + j];
                    }
                    sendMsg.setCheck();
                }
                else
                {
                    sendMsg.SrcPort = ClientPORT;
                    sendMsg.DestPort = RouterPORT;
                    sendMsg.SeqNum = sendnum;
                    for (int j = 0; j < MaxMsgSize; j++)
                    {
                        sendMsg.msgData[j] = fileBuffer[(sendnum -
1) * MaxMsgSize + j];
                    }
                    sendMsg.setCheck();
                }


                sendto(clientSocket,
                       (char*)&sendMsg,
                       sizeof(sendMsg),
                       0,
```

```
                        (sockaddr*)&serverAddr,
                        sizeof(SOCKADDR_IN));
                    cout << "Seq = " << sendMsg.SeqNum
                         << "的报文段已超时，正在重传......" << endl;
                }
                msgStart = clock();
                sendAgain = 0;
            }
            if (over == 1)//已收到所有ack
                break;
        }
        CloseHandle(hThread);
        cout << "\n\n已发送并确认所有报文，文件传输成功！\n\n";

        //计算传输时间和吞吐率
        int endTime = clock();
        cout << "\n\n总体传输时间为:"
            << (endTime - startTime) / CLOCKS_PER_SEC
            << "s" << endl;
        cout << "吞吐率:"
            << ((float)fileSize) / ((endTime - startTime) /
CLOCKS_PER_SEC)
            << "byte/s" << endl << endl;
}
```

- **新线程：接收**

  while循环持续准备发送报文：

  - 接收到未损坏报文：

    recvMsg.AckNum >= base：更新base，新base = recvMsg.AckNum + 1

    recvMsg.AckNum < base：不更新base

  - 接收到最后一个确认报文：

    设置全局变量 `over` 标识为 true，传给主线程

  - 三次冗余：

    设置 `wrongACK` 记录上一次接收到的ACK，`wrongCount` 记录重复ACK的数量，当 `wrongCount` 为 3 时，设置全局变量 `sendAgain` 为 true，传给主线程。

  - 接收到损坏报文：

    丢弃

```
//接收ack的线程
DWORD WINAPI recvThread(PVOID pParam)
```

```
{
    parameters* para = (parameters*)pParam;
    SOCKADDR_IN serverAddr = para->serverAddr;
    SOCKET clientSocket = para->clientSocket;
    int msgSum = para->msgSum;
    int AddrLen = sizeof(serverAddr);
    int wrongACK = -1;
    int wrongCount = 0;
    while (1)
    {
        //rdt_rcv
        Message recvMsg;
        int recvByte = recvfrom(clientSocket, (char*)&recvMsg,
                        sizeof(recvMsg), 0,
                        (sockaddr*)&serverAddr, &AddrLen);
        if (recvByte > 0)
        {
            //成功收到消息，且notcorrupt
            if (recvMsg.check())
            {
                if (recvMsg.AckNum >= base)
                    base = recvMsg.AckNum + 1;
                if (base != nextseqnum)
                    msgStart = clock();
                cout << "----client已收到【Ack = "
                     << recvMsg.AckNum << "】的确认报文" << endl;

                //打印窗口情况
                cout << "【当前窗口情况】 窗口总大小：" << N
                     << "，已发送但未收到ACK：" << nextseqnum - base
                     << "，尚未发送：" << N - (nextseqnum - base)
                     << "\n";

                //判断结束的情况
                if (recvMsg.AckNum == msgSum - 1)
                {
                    cout << "\nover------------------" << endl;
                    over = 1;
                    return 0;
                }

                //快速重传
                if (wrongACK != recvMsg.AckNum)
                {
                    wrongCount = 0;
                    wrongACK = recvMsg.AckNum;
                }
                else
```

```
                    wrongCount++;
                if (wrongCount == 3)
                    sendAgain = 1;//重发
            }
            //若校验失败，则忽略，继续等待
        }
    }
    return 0;
}
```

服务器端

- 收到**文件名**和**文件大小**
- 根据文件大小计算一共将要收到几个报文，并依次接收这些报文
  - 收到 recvMsg.SeqNum == expectedseqnum：

    回复确认报文 ack = recvMsg.SeqNum
  - 收到 recvMsg.SeqNum ！ = expectedseqnum：

    回复确认报文 ack = expectedseqnum - 1

```
bool recvMessage(Message& recvMsg, SOCKET serverSocket,
                 SOCKADDR_IN clientAddr, int& expectedseqnum)
{

    int AddrLen = sizeof(clientAddr);
    while (1)
    {
        int recvByte = recvfrom(serverSocket, (char*)&recvMsg,
                        sizeof(recvMsg), 0,
                        (sockaddr*)&clientAddr, &AddrLen);
        if (recvByte > 0)
        {
            //成功收到消息
            if (recvMsg.check() && (recvMsg.SeqNum == expectedseqnum))
            {
                //回复ACK
                Message replyMessage;
                replyMessage.SrcPort = ServerPORT;
                replyMessage.DestPort = RouterPORT;
                replyMessage.flag += ACK;
                replyMessage.AckNum = recvMsg.SeqNum;
                replyMessage.setCheck();
                sendto(serverSocket,
                       (char*)&replyMessage,
                       sizeof(replyMessage),
```

```
                        0,
                        (sockaddr*)&clientAddr,
                        sizeof(SOCKADDR_IN));
                cout << "server收到 Seq = " << recvMsg.SeqNum
                    << "的报文段，并发送 Ack = " << replyMessage.AckNum
                    << " 的回复报文段" << endl;
                expectedseqnum++;
                return true;
            }
            //如果seq! = 期待值，则返回累计确认的ack（expectedseqnum-1）
            else if (recvMsg.check() && (recvMsg.SeqNum !=
expectedseqnum))
            {
                //回复ACK
                Message replyMessage;
                replyMessage.SrcPort = ServerPORT;
                replyMessage.DestPort = RouterPORT;
                replyMessage.flag += ACK;
                replyMessage.AckNum = expectedseqnum - 1;
                replyMessage.setCheck();
                sendto(serverSocket,
                        (char*)&replyMessage,
                        sizeof(replyMessage),
                        0,
                        (sockaddr*)&clientAddr,
                        sizeof(SOCKADDR_IN));
                cout << "【累计确认（失序）】server收到 Seq = "
                     << recvMsg.SeqNum << "的报文段，并发送 Ack = "
                     << replyMessage.AckNum << " 的回复报文段" << endl;
            }
        }
        else if (recvByte == 0)
        {
            return false;
        }
    }
}
```

（三）建立连接

客户端

- 发送第一次握手的消息
    - 计时器开始计时
- 接收第二次握手的消息

    - 若超时没收到，则重传第一次的消息
- 发送第三次握手的消息

服务器端

- 接收第一次握手的消息并检验
- 发送第二次握手的消息
- 接收第三次握手的消息并检验
    - 若超时没收到，则重传第二次的消息

```
//（该部分使用3-1代码，未做修改）
```

（四）关闭连接

客户端

- 发送第一次挥手的消息
    - 计时器开始计时
- 接收第二次挥手的消息
    - 若超时没收到，则重传第一次的消息并重新计时
- 接收第三次挥手的消息
- 发送第四次挥手的消息
- 等待2MSL
    - 防止最后一个ACK丢失，处于半关闭
    - 若再次收到了消息，则回复第四次挥手的数据包

服务器端

- 接收第一次挥手的消息并检验
- 发送第二次挥手的消息
- 发送第三次挥手的消息
- 接收第四次挥手的消息并检验
    - 若超时没收到，则重传第三次的消息

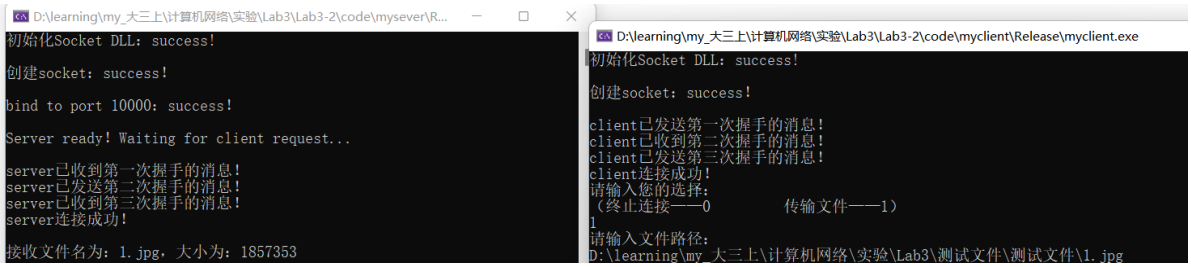
```
// （该部分使用3-1代码，未做修改）
```

三、程序界面与运行

- 设置路由器如下：

  5%的丢包率和5ms延时

建立连接



传输数据

- **流水线式发送：**

  只要窗口未满，发送端就可以持续发送：（见窗口状态）

- **快速重传**

三次冗余确认重传：

```
server收到 Seq = 14的报文段，并发送 Ack = 14 的回复报文段
数据报14接收成功
server收到 Seq = 15的报文段，并发送 Ack = 15 的回复报文段
数据报15接收成功
【累计确认（失序）】server收到 Seq = 17的报文段，并发送 Ack = 15 的回复报文段
【累计确认（失序）】server收到 Seq = 18的报文段，并发送 Ack = 15 的回复报文段
【累计确认（失序）】server收到 Seq = 19的报文段，并发送 Ack = 15 的回复报文段
【累计确认（失序）】server收到 Seq = 20的报文段，并发送 Ack = 15 的回复报文段
server收到 Seq = 16的报文段，并发送 Ack = 16 的回复报文段
数据报16接收成功
```

```
------client已发送【Seq = 18】的报文段!
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
------client已收到【Ack = 14】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
------client已发送【Seq = 19】的报文段!
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：5，尚未发送：0
------client已收到【Ack = 15】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
------client已收到【Ack = 15】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
------client已发送【Seq = 20】的报文段!
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：5，尚未发送：0
------client已收到【Ack = 15】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：5，尚未发送：0
------client已收到【Ack = 15】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：5，尚未发送：0
连续收到三次冗余ACK，快速重传......
Seq = 16的报文段已超时，正在重传......
Seq = 17的报文段已超时，正在重传......
Seq = 18的报文段已超时，正在重传......
Seq = 19的报文段已超时，正在重传......
Seq = 20的报文段已超时，正在重传......
------client已收到【Ack = 15】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：5，尚未发送：0
------client已收到【Ack = 16】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
------client已发送【Seq = 21】的报文段!
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
```

重传当前窗口内未确认报文

- **累计确认：**

  当收到接收端的ack，表示在此之前的报文段都已成功接收

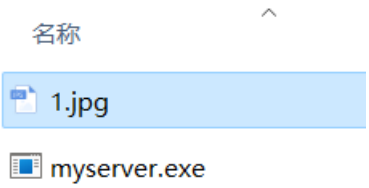
```
------client已收到【Ack = 150】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：5，尚未发送：0
Seq = 151的报文段已超时，正在重传......
Seq = 152的报文段已超时，正在重传......
Seq = 153的报文段已超时，正在重传......
Seq = 154的报文段已超时，正在重传......
Seq = 155的报文段已超时，正在重传......
------client已收到【Ack = 151】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
------client已发送【Seq = 156】的报文段!
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
------client已收到【Ack = 152】的确认报文
【当前窗口情况】 窗口总大小：5，已发送但未收到ACK：4，尚未发送：1
```

收到ack=151的确认报文，
证明此前的报文全部接收

```
server收到 Seq = 149的报文段，并发送 Ack = 149 的回复报文段
数据报149接收成功
server收到 Seq = 150的报文段，并发送 Ack = 150 的回复报文段
数据报150接收成功
【累计确认（失序）】server收到 Seq = 152的报文段，并发送 Ack = 150 的回复报文段
【累计确认（失序）】server收到 Seq = 153的报文段，并发送 Ack = 150 的回复报文段
【累计确认（失序）】server收到 Seq = 154的报文段，并发送 Ack = 150 的回复报文段
【累计确认（失序）】server收到 Seq = 155的报文段，并发送 Ack = 150 的回复报文段
server收到 Seq = 151的报文段，并发送 Ack = 151 的回复报文段
数据报151接收成功
server收到 Seq = 152的报文段，并发送 Ack = 152 的回复报文段
数据报152接收成功
server收到 Seq = 153的报文段，并发送 Ack = 153 的回复报文段
数据报153接收成功
```

- **关闭连接**

- 最终在sever端程序的目录下，成功得到了传输的文件，且文件大小和内容完全一致。



四、实验过程遇到的问题及分析

累计重传中，发送端收到确认报文段时，要更新base值（右移），但是此时需要增加判断条件，不可直接修改：

当失序的确认报文传来时（ack 小于 base的值），若此时不加判断，直接让base = ack+1，则会导致窗口“ 左移”，导致之前的包重发。因此，在更新 base 前，需要添加 ack >= base 的判断条件。