

# 计算机网络 3-2

## 个人信息

- 学号：2013750
- 姓名：管昀孜
- 专业：计算机科学与技术

## 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，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。

## 协议设计

本次实验为实现单向传输，并且顺序较为固定。

- 对于客户端：建立链接（主动）-> 选择文件发送-> 关闭连接（主动）
- 对于服务器：建立连接（被动）-> 接收文件-> 关闭连接（被动）

本次基于的协议是**滑动窗口 + SR（选择重传）**，并根据个人用法稍微做出了一点改变：为了减少数据的传输量与方便编程，本次协议设计时在建立/关闭连接和发送文件内容时使用了两套协议（即两套不同的结构）

### 1. 建立与关闭连接

#### 格式设计

在建立和关闭连接时，最主要的标志位是SYN和FIN，这两个标志位就能告诉对方本次的数据包的作用。

- SYN同步标志：该标志仅在三次握手建立TCP连接时有效。在三次握手期间，SYN被置为1。
- FIN断开标志：带有该标志置位的数据包用来结束一个TCP回话，在挥手期间被置为1。

TCP协议除了SYN和FIN标志位以外还有另外4个标志位；但本次实验并不会用到这么多功能，因此进行简化处理，去掉另外4个标志位，而增加了另外一个标志位FLAG。

- FLAG标志位：为了避免因为翻转导致歧义，除去SYN、FIN和校验和之外，数据包中还增加了一个FLAG位用来标志本次数据包的作用。此标志位标明是握手还是挥手；且标明是第几次握手/挥手。

简单起见，建立连接是保持三次握手，但关闭连接时只进行两次挥手。

FLAG标志位设计如下：

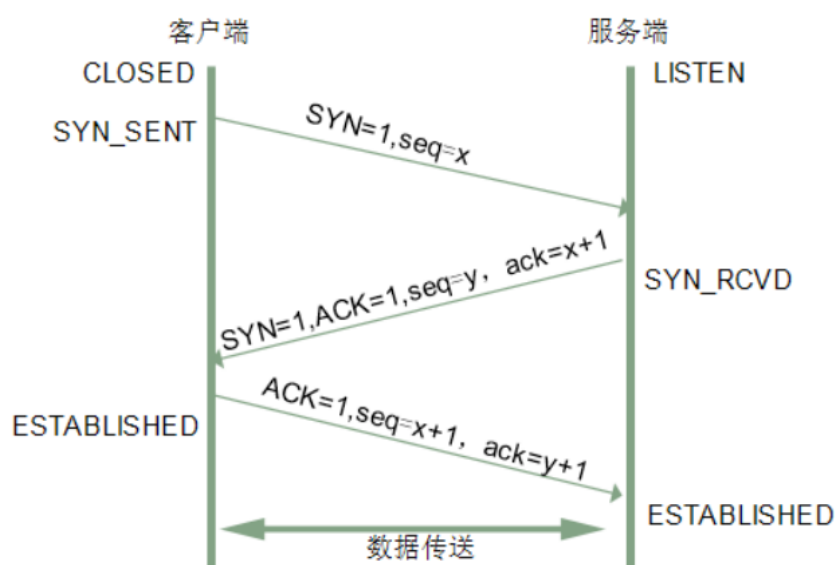
```
// 三次握手的时候对应的flag
const unsigned char SHAKE1 = 0x01;
const unsigned char SHAKE2 = 0x02;
const unsigned char SHAKE3 = 0x03;
// 四次挥手的时候的flag
const unsigned char WAVE1 = 0x04;
const unsigned char WAVE2 = 0x05;
```

数据包格式：`char package[4]`，具体为：

```
package[0]-----校验和
package[1]-----SYN
package[2]-----FIN
package[3]-----FLAG
```

## 建立连接 - 三次握手

1. 由client发起建立连接请求，发送时需要将**SYN置1**，**FIN置0**；第一次握手时**FLAG**需要设置为 **SHAKE1**。
2. server接收到第一次握手内容后，若不正确则不做任何相应，让client自行超时重传；若数据包正确，则发送**SYN为1**，**FIN为0**，**FLAG为SHAKE2**的数据包。
3. client收到来自server的第二次握手内容后，检查数据包，如果不正确，不做回复，等待server的超时重传；如果正确，向server发送**SYN为1**，**FIN为0**，**FLAG为SHAKE3**的数据包。
4. server收到来自client的第三次握手内容，检查数据包，如果不正确，回到最开始的状态重新等待连接；如果正确，回显连接建立。



如图，模仿的TCP三次握手，但没有设计ACK/seq/ack位

## 关闭连接 - 两次挥手

1. 由client发起关闭连接请求，发送时需要将**SYN置0**，**FIN置1**；第一次挥手时FLAG需要设置为 **WAVE1**。
2. server接收到第一次挥手内容后，若不正确则不做任何相应，让client自行超时重传；若数据包正确，则发送**SYN为0**，**FIN为1**，FLAG为**WAVE2**的数据包，并从服务器这边关闭连接。
3. client收到来自server的第二次挥手内容后，检查数据包，如果不正确，重新发送挥手数据包（重发的原因是猜测可能服务器那边没有收到挥手数据包，而在这里因为只有两次挥手，在这里可以简单当作收到第二次挥手的时候无论是否正确直接关闭连接），等待server的超时重传；如果正确，关闭连接。

## 2. 数据传输

### 格式设计

由于本次是单项传输：

- 对于client端来说，要发送校验和、seq位、数据长度、判断是否为最后一个包的标志位以及数据内容，因此比较复杂；
- 对于server端来说，只需要回复是ACK0还是ACK1即可（client作为发送端，server作为接收端），另外需要一个校验和，因此数据包格式比较简单

由于发送端和接收端复杂程度不同，为了简化，设计成两种不同的数据格式。

发送端：每次传送的数据包大小均为1024个字节，具体内容为：

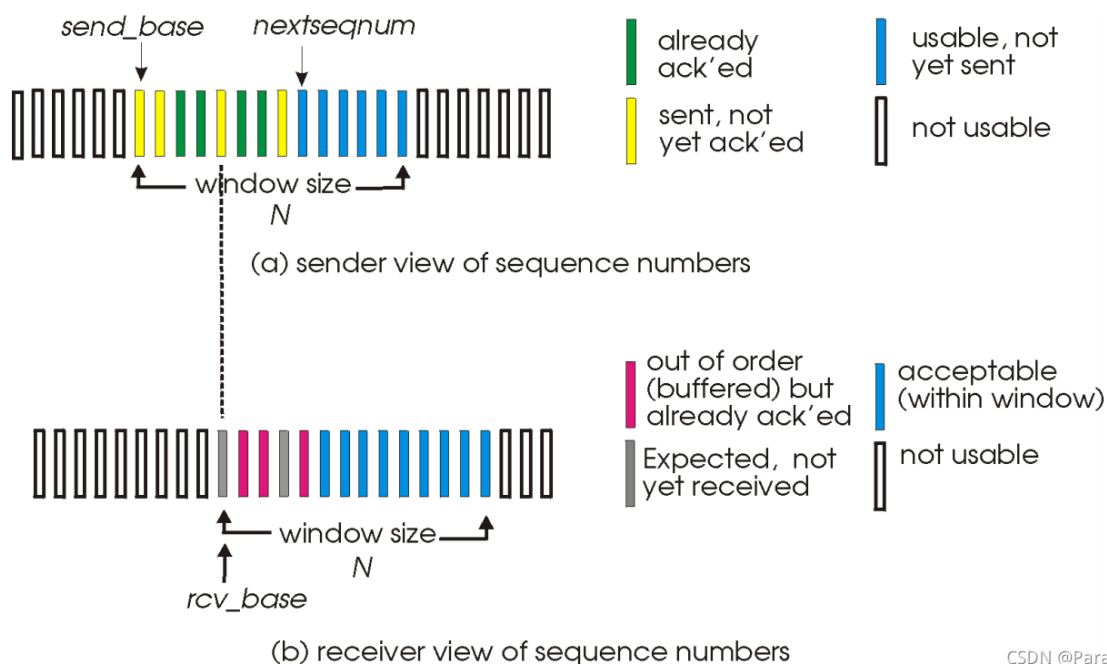
```
data[1024];
data[0] ---- 校验和
data[1] ---- 序号seq最高8位
data[2] ---- 序号seq中间8位
data[3] ---- 序号seq最低8位
data[4] ---- 数据长度高8位
data[5] ---- 数据长度低8位
data[6] ---- isLast位，用于当前数据包是否为最后一个数据包，是为1，反之为0
data[7~1023] ---- 要传输的数据内容，长度与data[3]、data[4]中算出来的数相同
```

接收端：传输内容仅为2个字节，具体内容为：

```
answer[4];
answer[0] ---- 校验和
answer[1] ---- ACK最高8位
answer[2] ---- ACK中间8位
answer[3] ---- ACK最低8位
```

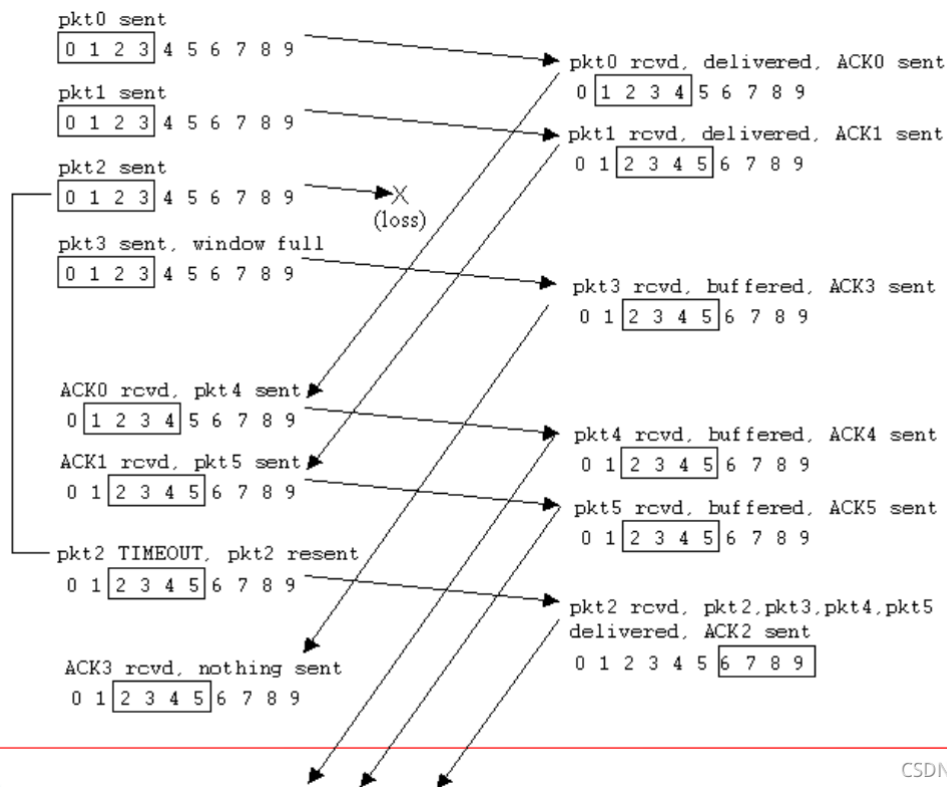
## 传输流程

SR（选择重传协议）简述：



CSDN @Paranoid☆

1. 接收方对每个分组单独进行确认，设置缓存机制，为了缓存乱序到达的分组，发送方就不会再次发送，限制已发送且未确认的分组
2. 如果计时器到点，仅重传该个未确认的数据报
3. 发送方窗口， $N$ 个连续的序列号，发送者在流水线中最多有  $N$  个未确认的数据报



如图中：滑动窗口的大小为4，发送数据包0，1，2，3，窗口满了，停止发送，等待确认，其中数据包2丢失，依次确认数据包0，1，同时窗口向后移，依次发送数据包4，5，当数据包2超时，发送方会再次发送数据包2，同时缓存乱序到达的3，4，5，当接收完数据包2后，滑动窗口后移，发送方继续发送数据包。

下面进行SR的构建：

本次实验实现的并不是完整的SR，而是在SR的基础上做了一些方便编程的修改，但不影响文件传输的正确性。

以下数据包在封装时默认都计算过校验和，并默认在收到数据包时都会检查校验和，校验和出错和下面其他位的出错处理方式相同。

## 发送端

对于发送端来说，需要维护的数据结构有：

- 滑动窗口的左端 `leftwindow`
- 标志是否已经被ACK的bool类型数组 `srBuf`。若 `srBuf[i]` 为 `true`，则表示该数据包确认收到，且该数据包之前的数据报都已收到。
- 将要发送的数据包指针 `curr`

滑动窗口+SR协议相关动作为：

1. 发送当前窗口内所有的数据包（从 `leftwindow` 到 `leftwindow + window_Size`），进入等待阶段，发送完成后开始计时。
2. 收到来自接收端的回复，如果 `ack >= leftwindow` 并且 `ack < curr`，则 `leftwindow = ack + 1`，并且将该 `seq` 的 `srBuf` 数组对应项更新为 `true`，表示已收到；如果 `ack == curr` 则窗口直接向后推进，发送移动之后的内容，并更新 `srBuf` 数组。这么做是因为在接收端处理时能保证：若第 `i` 号数据包收到，则之前所有的数据包都已收到。
3. 如果计时器的时间超过最大等待时间，重发窗口内的所有**未经确认**（即 `srBuf` 数组对应项为 `false`）的数据包。
4. 如果收到的包有误（损坏），需要重传所有**未经确认**的数据包。

## 接收端

接收端需要维护的数据结构有：

- 一个用来存储**超前到达**的序号的包的vector `srUnsure`
- 表示当前需要接收到的数据 `currentNum`

接收端首先使用停等协议等待写着文件名的数据包，然后再执行SR协议。

滑动窗口+SR协议相关动作为：

1. 在一轮接收数据包后，逐个计算收到的 `seq`
  - 如果 `seq > currentNum`，则将当前的包压入 `srUnsure` 中，此时不回ack；
  - 如果 `seq < currentNum`，则将当前数据包丢弃（由于累积确认，会认为该数据包已收到）；
  - 如果 `seq == currentNum`，则将当前的包解析后传递给上层，且 `currentNum ++`；接下来开始遍历 `srUnsure`，查看是否有所需要的 `currentNum` 号数据包，如果有则将其按照序号顺序传递给上层，并将其从 `srUnsure` 中删除，`currentNum ++`，直到遍历到不连续序号结束为止（即当前栈中 `seq != currentNum`）；
  - 如果没有则继续等待
2. 每次返回的ack为**最后一个交付上层的数据包**，即 `ack = currentNum`。这样就能保证发送端只要收到第 `i` 个ack就能保证之前所有的数据包已收到。
3. 如果数据包损坏，则需要返回最后一个已确认（已交付上层）的数据包，即 `currentNum`

## 计时器

双方各自都会维护一个计时器，当时间为-1的时候表示停止计时，其他均为正常计时。

且发送端的计时器是从当前轮中发送的最后一个包开始计时；如果中间有出现重传，则以当前重传的最后一个包为基准开始计时；接收端则从回复ack开始计时，每次回复都会刷新。

## 程序设计

本次程序流程和上述协议的流程相同，这里主要介绍一些功能函数。

### 1. 计算校验和

将除去第一位的剩下每一位都进行求和，如果求和结果超过8位，则将进位轮到末尾相加。最后将求和的结果取反得到校验和。相关代码如下：

```
/// <summary>
/// 计算校验和，从第1位开始算，一直算到最后一位
/// </summary>
/// <param name="pac">计算的包</param>
/// <param name="len">包的长度</param>
/// <returns>返回校验和</returns>
unsigned char newGetChecksum(char* package, int len) {
    if (len == 0) {
        return ~(0);
    }
    unsigned int sum = 0;
    int i = 0;
    while (len-- > 0) {
        sum += (unsigned char)package[i++];
        if (sum > 0xFF) {
            sum &= 0xFF;
            sum++;
        }
    }
    return ~(sum & 0xFF);
}
```

```

    }
}
return ~(sum & 0x00FF);
}

```

## 2. 获取文件列表

```

oid getFileList() {
    string path = "./test/*";

    // 建立句柄
    long handle;
    struct _finddata_t fileinfo;

    // 第一个文件
    handle = _findfirst(path.c_str(), &fileinfo);
    // 判断句柄状态
    if (handle == -1) {
        cout << "文件名称出现错误\n";
        return;
    }
    else {
        cout << "测试文件夹下目录列表为: \n";
        int tempNumOfFile = 0;
        do
        {
            //找到的文件的文件名
            if (fileinfo.name[0] == '.')
                continue;
            cout << ++tempNumOfFile << ": " << fileinfo.name << endl;

        } while (!_findnext(handle, &fileinfo));
        _findclose(handle);
        return;
    }
}

```

## 3. 获取IP

```

void getIP(char* p) {
    cout << "Please use 'ipconfig' to get your ip and input\n";
    char ip[16];
    cin.getline(ip, sizeof(ip));
    int index = 0;
    while (ip[index] != '\0') {
        p[index] = ip[index];
        index++;
    }
}

```

## Client(发送端)相关函数

### 4. 三次握手 - 发送端

```
// 三次握手，建立连接
void connet2Server() {
    // 创建一个要发送的数据包，并初始化
    // 数据包: check, SYN, FIN, SHAKE/WAVE
    char shakeChar[4];
    shakeChar[1] = 0x01;
    shakeChar[2] = 0x00;
    shakeChar[3] = SHAKE1;
    shakeChar[0] = newGetChecksum(shakeChar+1, 3);

    // 发送shake1
    sendto(sendSocket, shakeChar, 4, 0, (SOCKADDR*)&serverAddr, socketAddrLen);
    cout<< "开始建立连接，请求建立连接的数据包为（第一次握手）：\n";
    cout << "SYN: " << int(shakeChar[1]) << "\tFIN: " << int(shakeChar[2]) <<
    "\tSHAKE: " << int(shakeChar[3]) << "\tchecksum: " << int(shakeChar[0]) << endl;

    // 开始计时
    int beginTime = clock();

    // 接收缓冲区
    char recvShakeBuf[4];
    memset(recvShakeBuf, 0, 4);

    //超时重传
    while((recvfrom(sendSocket, recvShakeBuf, 4, 0, (SOCKADDR*)&serverAddr,
    &socketAddrLen))==SOCKET_ERROR){
        if (clock() - beginTime > MAX_WAIT_TIME) {
            cout << "超过最长等待时间，重传:";
            sendto(sendSocket, shakeChar, 4, 0, (SOCKADDR*)&serverAddr,
            socketAddrLen);
            beginTime = clock();
            cout << "SYN: " << int(shakeChar[1]) << "\tFIN: " <<
            int(shakeChar[2]) << "\tSHAKE: " << int(shakeChar[3]) << "\tchecksum: " <<
            int(shakeChar[0]) << endl;
        }
    }

    // 收到数据包
    cout << "收到数据包为: \n";
    cout << "SYN: " << int(recvShakeBuf[1]) << "\tFIN: " << int(recvShakeBuf[2])
    << "\tSHAKE: " << int(recvShakeBuf[3]) << "\tchecksum: " << int(recvShakeBuf[0])
    << endl;

    // 收到shake2正确，则发送shake3
    if (newGetChecksum(recvShakeBuf, 4)==0 && recvShakeBuf[1] == 0x01 &&
    recvShakeBuf[2] == 0x00 && recvShakeBuf[3] == SHAKE2) {
        memset(shakeChar, 0, 4);
        shakeChar[1] = 0x01;
        shakeChar[2] = 0x00;
        shakeChar[3] = SHAKE3;
        shakeChar[0] = newGetChecksum(shakeChar+1, 3);
        sendto(sendSocket, shakeChar, 4, 0, (SOCKADDR*)&serverAddr,
        socketAddrLen);
        cout << "发送第三次握手数据包: \n";
    }
}
```

```

        cout << "SYN: " << int(shakeChar[1]) << "\tFIN: " << int(shakeChar[2])
        << "\tSHAKE: " << int(shakeChar[3]) << "\tchecksum: " << int(shakeChar[0]) <<
        endl;

        cout << "\n连接建立成功\n\n";
    }
}

```

## 5. 滑动窗口与选择重传(SR)

```

/// <summary>
/// 向服务器传输文件使用的函数，滑动窗口
/// </summary>
/// <param name="fileName">要传输的文件名</param>
/// <param name="fileNameLen">文件名的长度</param>
static void sendFile2Server(char* fileName, unsigned short int fileNameLen) {
    // 对输入的名字进行处理
    char filePath[] = "./test/";
    char* fileNameN = new char[7 + fileNameLen + 1];
    for (int i = 0; i < 7 + fileNameLen + 1; i++) {
        if (i < 7)
            fileNameN[i] = filePath[i];
        else
            fileNameN[i] = fileName[i - 7];
    }
    fileNameN[7 + fileNameLen + 1] = '\0';

    // 读取文件（以二进制形式读入）
    ifstream toSendFile(fileNameN, ifstream::in | ios::binary);

    // 计算文件的长度
    toSendFile.seekg(0, toSendFile.end);
    unsigned int fileLen = toSendFile.tellg();
    toSendFile.seekg(0, toSendFile.beg);
    cout << "计算得到要发送的文件长度为: " << fileLen << endl;

    // 计算需要封装成多少个包
    unsigned int totalNum = fileLen / dataLen;
    if (fileLen % dataLen > 0)
        totalNum += 1;
    // 再加一个表示文件名称的包
    totalNum += 1;
    cout << "一共需要封装成" << totalNum << "个包\n";

    // 计时器，当为-1的时候表示计时器停止。
    int timer = -1;

    // 创建临时缓冲区
    char* tempBuf = new char[fileLen];
    // 将文件装载到缓冲区
    toSendFile.read(tempBuf, fileLen);

    // 窗口的最左边
    unsigned int leftwindow = 0;
    // 下一个要发送的
    unsigned int curr = 0;

    // 开始封装数据包

```



```

// 将文件全部封装，以后就直接发，而不是每次发之前封装
char** sendPackage = new char*[totalNum];
for (int tmp = 0; tmp < totalNum; tmp++) {
    *(sendPackage + tmp) = new char[1024];
    memset(sendPackage[tmp], 0, 1024);
}

// 设置文件名包体的内容
sendPackage[0][3] = 0x00;
for (unsigned short int i = 0; i < fileNameLen; i++) {
    sendPackage[0][i + 7] = fileName[i];
}
sendPackage[0][5] = unsigned char(fileNameLen & 0x00FF); //len高
sendPackage[0][4] = unsigned char((fileNameLen >> 8) & 0x00FF); //len低
sendPackage[0][0] = newGetChecksum(sendPackage[0]+1, 1024-1); //check

// 开始对文件整体进行打包
for (int tmp = 1; tmp < totalNum; tmp++) {
    // 设置标志位 seq
    sendPackage[tmp][3] = tmp & 0xFF;
    sendPackage[tmp][2] = (tmp >> 8) & 0xFF;
    sendPackage[tmp][1] = (tmp >> 16) & 0xFF;

    // 设置isLast位
    if (tmp == totalNum - 1)
        sendPackage[tmp][6] = 0x01;
    else
        sendPackage[tmp][6] = 0x00;

    // 装载下一部分文件
    unsigned short int actualLen = 0;
    for (int tempPoint = 0; (tempPoint < dataLen) && ((tmp - 1) * dataLen +
tempPoint) < fileLen; tempPoint++) {
        sendPackage[tmp][7 + tempPoint] = tempBuf[(tmp - 1) * dataLen +
tempPoint];
        actualLen += 1;
    }

    // 设置长度
    sendPackage[tmp][5] = unsigned char(actualLen & 0x00FF);
    sendPackage[tmp][4] = unsigned char((actualLen >> 8) & 0x00FF);

    // 计算校验和
    sendPackage[tmp][0] = newGetChecksum(sendPackage[tmp] + 1, 1024 - 1);
}
// -----至此文件全部打包结束

// 创建一个sr缓冲区，并初始化。该数组标识是否acked
bool* srBuf = new bool[totalNum] ;
for (int i = 0; i < totalNum; i++)
    srBuf[i] = false;

// 从开始发送第一个数据包这里开始计算需要统计的时间
sendFileTime = clock();

while (leftwindow != totalNum) {
    // for循环发送窗口那么多个数据包，窗口大小为WINDOW_SIZE
    // 对于当前需要发送的数据包的计算公式：curr = leftwindow + willSendInwindow

```

```

        for (; (curr < leftwindow + WINDOW_SIZE) && (curr < totalNum); curr++) {
            // 发送数据包
            sendto(sendSocket, (char*)sendPackage[curr], 1024, 0,
(SOCKADDR*)&serverAddr, socketAddrLen);
            if (curr == leftwindow)
                // 计时器开始计时
                timer = clock();
            // 输出相关信息
            cout << "当前发送数据包为: \nseq: " << curr << "\nlen高: " <<
int(sendPackage[curr][4]) << "\tlen低: " << int(sendPackage[curr][5])
                << "\tchecksum: " << int(sendPackage[curr][0]) << endl << "剩余窗
口大小: " << (leftwindow + WINDOW_SIZE - curr) << endl;
        }

        // 创建一个用来存放回复的缓冲区, 长度为4
        char recvBuf[4];
        // 循环接收数据包, 并更新窗口左端
        while (recvfrom(sendSocket, recvBuf, 4, 0, (SOCKADDR*)&serverAddr,
&socketAddrLen) == SOCKET_ERROR) {
            // 超时重传
            if ((timer != -1) && (clock() - timer > MAX_WAIT_TIME)) {
                cout << "=====超过最长等待时间, 重传:\n";
                int tempBase = leftwindow;
                for (; tempBase < curr; tempBase++) {
                    if (!srBuf[tempBase]) { // 仅重传未确认的数据报
                        // 发送数据包
                        sendto(sendSocket, (char*)sendPackage[tempBase], 1024,
0, (SOCKADDR*)&serverAddr, socketAddrLen);
                        // 输出相关信息
                        cout << "当前发送数据包为: \nseq: " << tempBase << "\nlen高:
" << int(sendPackage[tempBase][4]) << "\tlen低: " << int(sendPackage[tempBase]
[5])
                            << "\tchecksum: " << int(sendPackage[tempBase][0])
<< endl << "剩余需要重传的包个数为: " << (curr - tempBase - 1) << endl << endl;
                    }
                    // 重新开始计时
                    timer = clock();
                }
            }
        }
        // 表示收到了有效的数据包, 开始检查数据包的正确性

        // 计算回来的序号
        unsigned int answerNum;
        answerNum = unsigned short int(recvBuf[1]) & 0xFF;
        answerNum = (answerNum << 8) + (unsigned short int(recvBuf[2]) & 0xFF);
        answerNum = (answerNum << 8) + (unsigned short int(recvBuf[3]) & 0xFF);
        cout << "FROM S:\nchecksum: " << int(recvBuf[0]) << "\tack: " <<
answerNum << endl;

        // 检查checksum
        if ((newGetChecksum(recvBuf, 4) == 0) && (answerNum >= leftwindow)) {
            if (answerNum == curr-1) {
                // bool数组更新, 设置为true, 表示已经传到了
                for (int i = leftwindow; i < answerNum; i++)
                    srBuf[i] = true;
                // 窗口的左边移动
                leftwindow = answerNum + 1;
            }
        }
    }
}

```

```

        // 判断是否需要更新timer
        if (leftwindow == curr)
            timer = -1;
        else
            timer = clock();
    }
    else if (answerNum < curr-1) {
        // bool数组更新, 设置为true, 表示已经传到了
        for (int i = leftwindow; i < answerNum; i++)
            srBuf[i] = true;
        // 窗口的左边移动
        leftwindow = answerNum + 1;
    }
}
else {
    // 要么是收到的包错了, 要么是ack的序号是之前的, 此时就需要把现在这些全都重传
    cout << "=====收到的数据包有误, 重传:\n";
    //int tempBase = leftwindow;
    int tempBase = answerNum + 1;
    for (; tempBase < curr; tempBase++) {
        if (!srBuf[tempBase]) {
            // 发送数据包
            sendto(sendSocket, (char*)sendPackage[tempBase], 1024, 0,
                (SOCKADDR*)&serverAddr, socketAddrLen);
            // 输出相关信息
            cout << "当前发送数据包为: \nseq: " << tempBase << "\nlen高: "
                << int(sendPackage[tempBase][4]) << "\tlen低: " << int(sendPackage[tempBase][5])
                << "\tchecksum: " << int(sendPackage[tempBase][0]) << endl;
            endl << "\n剩余需要重传的包个数为: " << (curr - tempBase - 1) << endl << endl;
        }
    }
    //重新开始计时
    timer = clock();
}
}

sendFileTime = clock() - sendFileTime;
// 计算成秒
sendFileTime = sendFileTime / CLOCKS_PER_SEC;
cout << "本次传输所用时间为: " << sendFileTime << " s." << endl;
// 计算吞吐量
Throughput = fileLen / sendFileTime;
}

```

## 6. 两次挥手 - 发送端

```

static void sayGoodBye2Server() {
    // 两次挥手, 断开连接

    // 数据包: check, SYN, FIN, SHAKE/WAVE
    char shakeChar[4];
    shakeChar[1] = 0x00;
    shakeChar[2] = 0x01;
    shakeChar[3] = WAVE1;
    shakeChar[0] = newGetChecksum(shakeChar+1, 4-1);
    // 发送WAVE1
}

```

```

sendto(sendSocket, shakeChar, 4, 0, (SOCKADDR*)&serverAddr, socketAddrLen);
cout << "断开连接，发送数据包为（第一次挥手）：\n";
cout << "SYN: " << int(shakeChar[1]) << "\tFIN: " << int(shakeChar[2]) <<
"\tSHAKE: " << int(shakeChar[3]) << "\tchecksum: " << int(shakeChar[0]) << endl;

// 开始计时
int beginTime = clock();

// 停等，等服务器的挥手
char recvWaveBuf[4];
while (true) {
    while ((recvfrom(sendSocket, recvWaveBuf, 4, 0, (SOCKADDR*)&serverAddr,
&socketAddrLen))==SOCKET_ERROR) {
        if (clock() - beginTime > MAX_WAIT_TIME) {
            cout << "超过最长等待时间，重传:";
            sendto(sendSocket, shakeChar, 4, 0, (SOCKADDR*)&serverAddr,
socketAddrLen);
            beginTime = clock();
            cout << "SYN: " << int(shakeChar[1]) << "\tFIN: " <<
int(shakeChar[2]) << "\tSHAKE: " << int(shakeChar[3]) << "\tchecksum: " <<
int(shakeChar[0]) << endl;
        }
    }
    if (newGetChecksum(recvWaveBuf, 4)==0 && recvWaveBuf[1] == 0x00 &&
recvWaveBuf[2] == 0x01 && recvWaveBuf[3] == WAVE2) {
        // 收到握手包正确
        cout << "收到来自服务器的第二次挥手：\n";
        cout << "SYN: " << int(recvWaveBuf[1]) << "\tFIN: " <<
int(recvWaveBuf[2]) << "\tSHAKE: " << int(recvWaveBuf[3]) << "\tchecksum: " <<
int(recvWaveBuf[0]) << endl;
        memset(recvWaveBuf, 0, 4);
        memset(shakeChar, 0, 4);
        break;
    }
    else {
        cout << "服务器挥手失败，重传挥手数据包\n";
        sendto(sendSocket, shakeChar, 4, 0, (SOCKADDR*)&serverAddr,
socketAddrLen);
        cout << "断开连接，发送数据包为（第一次挥手）：\n";
        cout << "SYN: " << int(shakeChar[1]) << "\tFIN: " <<
int(shakeChar[2]) << "\tSHAKE: " << int(shakeChar[3]) << "\tchecksum: " <<
int(shakeChar[0]) << endl;
    }
}
}
}

```

## Server(接收端)相关函数

### 7. 三次握手 - 接收端

```

static bool shakeHand() {
    // 握手
    //Package p;
    //Package sendP;
    char recvShakeBuf[4];
    while (1) {
        memset(recvShakeBuf, 0, 4);
    }
}

```

```

// 停等机制接收数据包
while (recvfrom(serverSocket, recvShakeBuf, 4, 0,
(sockaddr*)&clientAddr, &sockaddrLen) == SOCKET_ERROR) {

}

// 检查接收到的数据包校验和是否正确
// 先对收到的数据进行转换
//p = *((Package*)recvBuf);
// 输出信息
cout << "收到建立连接的请求，数据包为：\n";
// packageOutput(p);
cout << "SYN: " << int(recvShakeBuf[1]) << "\tFIN: " <<
int(recvShakeBuf[2]) << "\tSHAKE: " << int(recvShakeBuf[3]) << "\tchecksum: " <<
int(recvShakeBuf[0]) << endl;
// 第一次
if (newGetChecksum(recvShakeBuf,4)==0 && recvShakeBuf[1] == 0x01 &&
recvShakeBuf[2] == 0x00 && recvShakeBuf[3] == SHAKE1) {
// 收到握手包正确
cout << "握手包正确\n";
// 创建第二次握手包
char secondShake[4];
secondShake[1] = 0x01;
secondShake[2] = 0x00;
secondShake[3] = SHAKE2;
secondShake[0] = newGetChecksum(secondShake+1, 4-1);
// 发送第二次握手包
sendto(serverSocket, secondShake, 4, 0, (SOCKADDR*)&clientAddr,
sockaddrLen);
// 输出内容
cout << "SYN: " << int(secondShake[1]) << "\tFIN: " <<
int(secondShake[2]) << "\tSHAKE: " << int(secondShake[3]) << "\tchecksum: " <<
int(secondShake[0]) << endl;

int beginTime = clock();

// 清空刚刚接收的包，等待第三次握手
memset(recvShakeBuf, 0, 4);

while (recvfrom(serverSocket, recvShakeBuf, 4, 0,
(sockaddr*)&clientAddr, &sockaddrLen) == SOCKET_ERROR) {
if (clock() - beginTime > MAX_WAIT_TIME) {
cout << "超过最长等待时间，重传:";
sendto(serverSocket, secondShake, 4, 0,
(SOCKADDR*)&clientAddr, sockaddrLen);
beginTime = clock();
cout << "SYN: " << int(secondShake[1]) << "\tFIN: " <<
int(secondShake[2]) << "\tSHAKE: " << int(secondShake[3]) << "\tchecksum: " <<
int(secondShake[0]) << endl;
}
}

cout << "收到数据包为：\n";
cout << "SYN: " << int(recvShakeBuf[1]) << "\tFIN: " <<
int(recvShakeBuf[2]) << "\tSHAKE: " << int(recvShakeBuf[3]) << "\tchecksum: " <<
int(recvShakeBuf[0]) << endl;

```

```

        if (newGetChecksum(recvShakeBuf, 4)==0 && recvShakeBuf[1] == 0x01 &&
recvShakeBuf[2] == 0x00 && recvShakeBuf[3] == SHAKE3) {
            // 收到握手包正确
            cout << "\n连接建立成功\n\n";
            return true;
        }
    }
    else {
        cout << "收到的数据包有误，重新等待连接\n";
        continue;
    }
}
}

```

## 8. 滑动窗口与选择重传(SR)

```

static void recvFile() {
    // 接收文件
    // 创建缓冲区
    char recvBuf[1024];
    memset(recvBuf, 0, 1024);

    // 创建存储选择部分的vec
    vector<char*> srUnsure;

    // 创建发送使用的缓冲区
    char sendBuf[4];
    memset(sendBuf, 0, 4);

    // 记录名字和名字长度
    unsigned short int nameLen = 0;
    char* fileName;

    // 记录收到的seq
    unsigned int seq;

    // 停等，等名字
    while (true) {
        // 接收到的第一个包是名字
        while (recvfrom(serverSocket, recvBuf, 1024, 0, (sockaddr*)&clientAddr,
&sockaddrLen) == SOCKET_ERROR) {

        }

        // 计算seq
        seq = 0;
        seq = unsigned int(recvBuf[1]);
        seq = (seq << 8) + unsigned int(recvBuf[2]);
        seq = (seq << 8) + unsigned int(recvBuf[3]);

        cout << "From C\t收到的数据包为;\n";
        cout << "seq: " << seq << "\nlen高: " << int(recvBuf[4]) << "\tlen低: "
<< int(recvBuf[5])
        << "\tchecksum: " << int(recvBuf[0]) << endl;

        if (newGetChecksum(recvBuf, 1024)==0) {
            if (seq == currentNum) {

```

```

        // 记录名称
        // 计算名字的长度
        unsigned short int temp = unsigned short int(recvBuf[4]);
        nameLen = (temp << 8) + unsigned short int(recvBuf[5]);
        // 解析名字
        fileName = new char[nameLen];
        for (int i = 0; i < nameLen; i++)
            fileName[i] = recvBuf[i + 7];

        // 查看是否需要回传
        if (currentNum != 0 && currentNum % divenum == 0) {
            // 构建要返回的数据包
            sendBuf[1] = (currentNum >> 16) & 0xFF;
            sendBuf[2] = (currentNum >> 8) & 0xFF;
            sendBuf[3] = currentNum & 0xFF;
            sendBuf[0] = newGetChecksum(sendBuf + 1, 3);

            // 发送数据包
            sendto(serverSocket, sendBuf, 4, 0, (SOCKADDR*)&clientAddr,
sockaddrLen);

            cout << "TO C\t返回ack数据包\n";
            cout << "checksum: " << int(sendBuf[0]) << "\tack: " <<
currentNum << endl;
        }
        // 期待的数据包+1
        currentNum += 1;

        break;
    }
}

// 对名字进行处理
char filePath[] = "C:\\Users\\11955\\Desktop\\test\\";
char* fileNameN = new char[28 + nameLen + 1];
for (int i = 0; i < 28 + nameLen; i++) {
    if (i < 28)
        fileNameN[i] = filePath[i];
    else {
        fileNameN[i] = fileName[i - 28];
    }
}
fileNameN[28 + nameLen] = '\0';
cout << "收到的文件保存路径为: " << fileNameN << endl;

ofstream towriteFile;
towriteFile.open(fileNameN, ios::binary);
if (!towriteFile) {
    cout << "文件创建失败\n";
    return;
}

// 开始计时
int beginTime = clock();

while (true) {

```

```

        while (recvfrom(serverSocket, recvBuf, 1024, 0, (sockaddr*)&clientAddr,
&sockaddrLen) == SOCKET_ERROR) {
            if (clock() - beginTime > MAX_WAIT_TIME) {
                cout << "超过最大等待时间，重传：";
                sendBuf[3] = currentNum-1 & 0xFF;
                sendBuf[2] = (currentNum-1 >> 8) & 0xFF;
                sendBuf[1] = (currentNum-1 >> 16) & 0xFF;
                sendBuf[0] = newGetChecksum(sendBuf + 1, 3);

                sendto(serverSocket, sendBuf, 4, 0, (SOCKADDR*)&clientAddr,
sockaddrLen);

                cout << "checksum: " << int(sendBuf[0]) << "\tack: " <<
(currentNum-1) << endl;
                beginTime = clock();
            }
        }
        // 计算seq
        seq = 0;
        seq = unsigned int(recvBuf[1]);
        seq = (seq << 8) + (unsigned int(recvBuf[2]) & 0xff);
        seq = (seq << 8) + (unsigned int(recvBuf[3]) & 0xff);

        cout << "From C\t收到的数据包为;\n";
        cout << "seq: " << seq << "\tlen高: " << int(recvBuf[4]) << "\tlen低: "
<< int(recvBuf[5]) << "\tisLast: " << int(recvBuf[6]) << "\tchecksum: " <<
int(recvBuf[0]) << endl;

        if (newGetChecksum(recvBuf, 1024) == 0) { // 校验正确
            if (currentNum == seq) {
                // 获取要读取的字符串的长度
                unsigned short int temp = unsigned short int(recvBuf[4]);
                temp = (temp << 8) + (unsigned short int(recvBuf[5]) & 0x00FF);

                // 写入文件
                for (int i = 0; i < temp; i++) {
                    towriteFile << recvBuf[7 + i];
                }
                cout << "delivered:\t" << currentNum << endl;
                currentNum++;

                bool hasLargerThen = true;
                // 看vec里有没有比这个大的，如果有，按照循序写

                while (hasLargerThen && srUnsure.size()) {
                    bool tempFlag = false;
                    for (int tempPoint = 0; tempPoint < srUnsure.size();
tempPoint++) {
                        // 获取seq
                        unsigned int vecSeq = unsigned int(srUnsure[tempPoint]
[1] & 0xff);
                        vecSeq = (vecSeq << 8) + unsigned
int(srUnsure[tempPoint][2] & 0xff);
                        vecSeq = (vecSeq << 8) + unsigned
int(srUnsure[tempPoint][3] & 0xff);
                        if (vecSeq == currentNum) {
                            // 获取要读取的字符串的长度
                            unsigned short int temp = unsigned short
int(srUnsure[tempPoint][4]);

```



```

        temp = (temp << 8) + (unsigned short
int(srUnsure[tempPoint][5]) & 0x00FF);
        //cout << "temp长度: " << temp << endl << endl;

        // 写入文件
        for (int i = 0; i < temp; i++) {
            towriteFile << srUnsure[tempPoint][7 + i];
        }
        cout << "delivered:\t" << currentNum << endl;
        // 指针后移
        currentNum += 1;

        // vec中弹出
        srUnsure.erase(srUnsure.begin() + tempPoint);
        tempPoint -= 1;
    }
    else if (vecSeq > currentNum) {
        hasLargerThen = true;
        tempFlag = true;
        break;
    }
    else if (vecSeq < currentNum) {
        // vec中弹出
        srUnsure.erase(srUnsure.begin() + tempPoint);
        tempPoint -= 1;
    }
}
if (tempFlag)
    continue;
else
    hasLargerThen = false;
}

// 查看是否需要回传ack
if (currentNum % divenum == 0 || int(recvBuf[6]) == 1) {
    // 没有损坏, 返回对应ack, 然后ack翻转
    sendBuf[1] = (currentNum-1 >> 16) & 0xFF;
    sendBuf[2] = (currentNum-1 >> 8) & 0xFF;
    sendBuf[3] = (currentNum-1) & 0xFF;
    sendBuf[0] = newGetChecksum(sendBuf + 1, 3);

    // 发送文件
    sendto(serverSocket, sendBuf, 4, 0, (SOCKADDR*)&clientAddr,
sockaddrLen);
}

beginTime = clock();

cout << "TO C\t正确返回数据包\n";
cout << "checksum: " << int(sendBuf[0]) << "\tack: " <<
currentNum-1 << endl;

// 确认当前收到的是最后一个
if (int(recvBuf[6]) == 1)
    break;
}
else {

```

```

        if (seq > currentNum) {
            // 加入到vec中
            srUnsure.push_back(recvBuf);
        }
        else {
            // 有损坏, 返回已确认过的最后一个ack
            //sendBuf[1] = (currentNum-1 >> 16) & 0xFF;
            //sendBuf[2] = (currentNum-1 >> 8) & 0xFF;
            //sendBuf[3] = (currentNum-1) & 0xFF;
            //sendBuf[0] = newGetChecksum(sendBuf + 1, 3);

            //sendto(serverSocket, sendBuf, 4, 0,
(SOCKADDR*)&clientAddr, sockaddrLen);

            //beginTime = clock();
            //cout << "TO C\t接收到的数据包有损坏, 返回需要的ack\n";
            //cout << "check: " << int(sendBuf[0]) << "\tack: " <<
(currentNum-1) << endl;
            //continue;
        }
    }
}
else {
    // 有损坏, 返回之前已经确认过的ack
    sendBuf[1] = (currentNum-1 >> 16) & 0xFF;
    sendBuf[2] = (currentNum-1 >> 8) & 0xFF;
    sendBuf[3] = (currentNum-1) & 0xFF;
    sendBuf[0] = newGetChecksum(sendBuf + 1, 3);

    sendto(serverSocket, sendBuf, 4, 0, (SOCKADDR*)&clientAddr,
sockaddrLen);

    beginTime = clock();
    cout << "TO C\t接收到的数据包有损坏, 返回需要的ack\n";
    cout << "check: " << int(sendBuf[0]) << "\tack: " << (currentNum-1)
<< endl;
    continue;
}
}

// 文件接收完毕, 关闭文件
towriteFile.close();

cout << "文件保存成功\n";
}

```

## 9. 两次挥手 - 接收端

```

static void waveHand() {
    // 挥手, 挥两次
    char recvWaveBuf[4];
    while (true) {
        while (recvfrom(serverSocket, recvWaveBuf, 4, 0, (sockaddr*)&clientAddr,
&sockaddrLen) == SOCKET_ERROR) {

        }
    }
}

```

```

        cout << "收到断开连接的请求，数据包为：\n";
        cout << "SYN: " << int(recvWaveBuf[1]) << "\tFIN: " <<
int(recvWaveBuf[2]) << "\tSHAKE: " << int(recvWaveBuf[3]) << "\tchecksum: " <<
int(recvWaveBuf[0]) << endl;

        // 第一次
        if (newGetChecksum(recvWaveBuf, 4)==0 && recvWaveBuf[1] == 0x00 &&
recvWaveBuf[2] == 0x01 && recvWaveBuf[3] == WAVE1) {
            // 收到挥手包正确
            cout << "挥手包正确\n";
            // 创建第二次挥手包
            char secondShake[4];
            secondShake[1] = 0x00;
            secondShake[2] = 0x01;
            secondShake[3] = WAVE2;
            secondShake[0] = newGetChecksum(secondShake+1, 4-1);
            // 发送第二次挥手包
            sendto(serverSocket, secondShake, 4, 0, (SOCKADDR*)&clientAddr,
sockaddrLen);
            // 输出内容
            cout << "发送给客户端的第二次挥手: \n";
            cout << "SYN: " << int(secondShake[1]) << "\tFIN: " <<
int(secondShake[2]) << "\tSHAKE: " << int(secondShake[3]) << "\tchecksum: " <<
int(secondShake[0]) << endl;

            memset(secondShake, 0, 4);
            memset(recvWaveBuf, 0, 4);
            break;
        }
    }
}

```

## 吞吐率计算

使用C++中的clock函数，从开始传输文件时计时，文件传输完成结束。

吞吐率计算公式为：传输的文件大小/总时长；其中文件大小按字节计算，总时长按秒计算

## 使用方法

### Client - 发送端

1. 运行程序
2. 输入要发送的IP地址
3. 输入本地的IP地址
4. 选择文件发送（需要输入文件的全名）
5. 文件发送完成自动结束程序

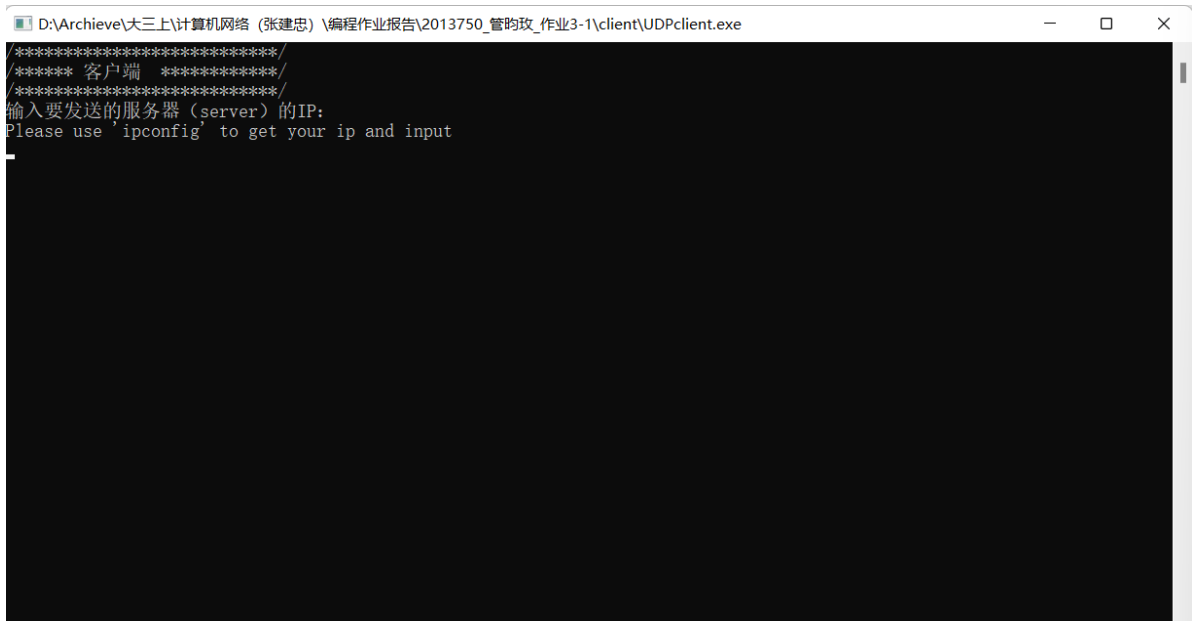
### Server - 接收端

1. 运行程序
2. 输入要接收的发送端的IP
3. 等待连接
4. 连接成功后等待发送端发送文件
5. 文件接收完成后自动结束程序

# 实验成果展示

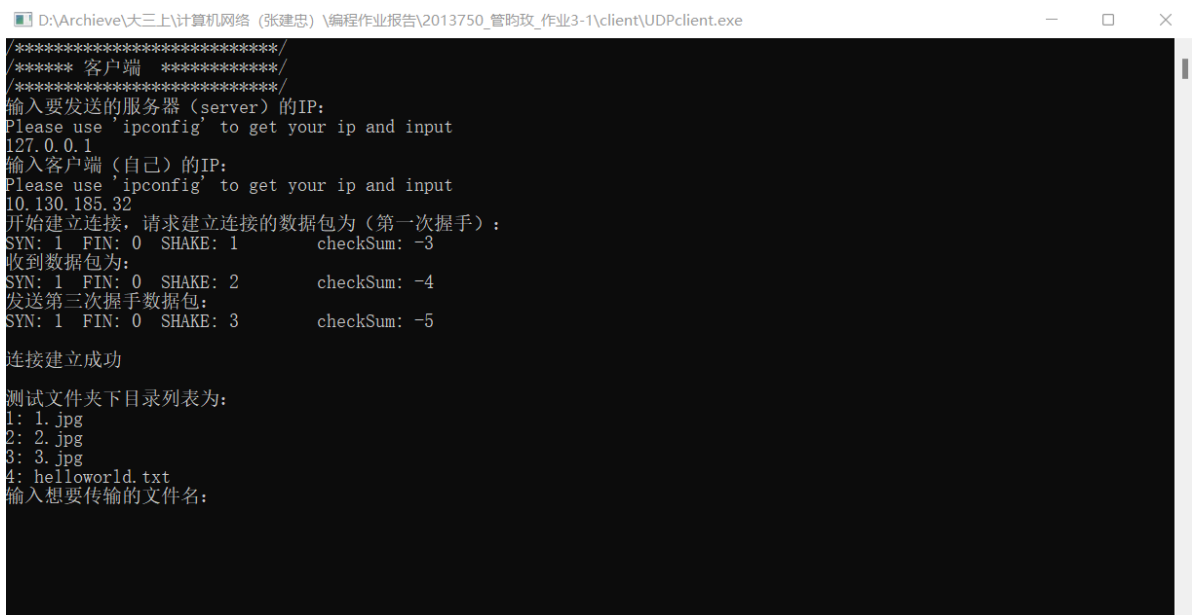
## Client发送端

等待连接页面



```
D:\Archive\大三上\计算机网络 (张建忠) \编程作业报告\2013750_管昀孜_作业3-1\client\UDPclient.exe
***** 客户端 *****
***** 客户端 *****
***** 客户端 *****
输入要发送的服务器 (server) 的IP:
Please use 'ipconfig' to get your ip and input
```

输入服务器IP和客户端IP，三次握手成功，自动弹出文件列表并等待输入



```
D:\Archive\大三上\计算机网络 (张建忠) \编程作业报告\2013750_管昀孜_作业3-1\client\UDPclient.exe
***** 客户端 *****
***** 客户端 *****
***** 客户端 *****
输入要发送的服务器 (server) 的IP:
Please use 'ipconfig' to get your ip and input
127.0.0.1
输入客户端 (自己) 的IP:
Please use 'ipconfig' to get your ip and input
10.130.185.32
开始建立连接, 请求建立连接的数据包为 (第一次握手):
SYN: 1 FIN: 0 SHAKE: 1      checkSum: -3
收到数据包为:
SYN: 1 FIN: 0 SHAKE: 2      checkSum: -4
发送第三次握手数据包:
SYN: 1 FIN: 0 SHAKE: 3      checkSum: -5

连接建立成功

测试文件夹下目录列表为:
1: 1.jpg
2: 2.jpg
3: 3.jpg
4: helloworld.txt
输入想要传输的文件名:
```

输入要发送的文件名，进行发送



```
输入想要传输的文件名: helloworld.txt
计算得到要发送的文件长度为: 1655808
一共需要封装成1626个包
```

发送过程中

```
剩余窗口大小: 8
当前发送数据包为:
seq: 650
len高: 3      len低: -7      checksum: 109
剩余窗口大小: 7
当前发送数据包为:
seq: 651
len高: 3      len低: -7      checksum: 102
剩余窗口大小: 6
当前发送数据包为:
seq: 652
len高: 3      len低: -7      checksum: -67
剩余窗口大小: 5
当前发送数据包为:
seq: 653
len高: 3      len低: -7      checksum: 97
剩余窗口大小: 4
当前发送数据包为:
seq: 654
len高: 3      len低: -7      checksum: 76
剩余窗口大小: 3
当前发送数据包为:
seq: 655
len高: 3      len低: -7      checksum: 62
剩余窗口大小: 2
当前发送数据包为:
seq: 656
len高: 3      len低: -7      checksum: 127
剩余窗口大小: 1
```

## 重传

```
剩余需要重传的包个数为: 6

当前发送数据包为:
seq: 1047
len高: 3      len低: -7      checksum: 12

剩余需要重传的包个数为: 5

当前发送数据包为:
seq: 1048
len高: 3      len低: -7      checksum: 47

剩余需要重传的包个数为: 4

当前发送数据包为:
seq: 1049
len高: 3      len低: -7      checksum: -32

剩余需要重传的包个数为: 3

当前发送数据包为:
seq: 1050
len高: 3      len低: -7      checksum: -105

剩余需要重传的包个数为: 2

当前发送数据包为:
seq: 1051
len高: 3      len低: -7      checksum: -106
```

完成发送后，告知用户文件传输完成，并输出传输时间和吞吐率

最后进行挥手，结束连接

```
本次传输所用时间为: 206.18 s.
断开连接，发送数据包为（第一次挥手）：
SYN: 0 FIN: 1 SHAKE: 4      checkSum: -6
收到来自服务器的第二次挥手：
SYN: 0 FIN: 1 SHAKE: 5      checkSum: -7
本次传输吞吐率为: 8030.89 Kb/s
请按任意键继续 .
```

## Server接收端

输入接收端IP并等待连接(本该输入本机IP，但此处为了方便直接固定为回环地址127.0.0.1)

```
D:\Archive\大三上\计算机网络 (张建忠) \编程作业报告\2013750_管昀孜_作业3-1\server\UDPserver.exe
*****
***** 服务器 *****
*****
获取client的IP
Please use 'ipconfig' to get your ip and input
10.130.185.32
服务器启动成功，等待连接.....
```

### 三次握手成功页面

```
D:\Archive\大三上\计算机网络 (张建忠) \编程作业报告\2013750_管昀孜_作业3-1\server\UDPserver.exe
*****
***** 服务器 *****
*****
获取client的IP
Please use 'ipconfig' to get your ip and input
10.130.185.32
服务器启动成功，等待连接.....
收到建立连接的请求，数据包为：
SYN: 1 FIN: 0 SHAKE: 1      checksum: -3
握手包正确
SYN: 1 FIN: 0 SHAKE: 2      checksum: -4
收到数据包为：
SYN: 1 FIN: 0 SHAKE: 3      checksum: -5
连接建立成功
```

### 接收文件页面

```
From C 收到的数据包为：
seq: 1147      len高: 3      len低: -7      isLast: 0      checksum: 54
From C 收到的数据包为：
seq: 1148      len高: 3      len低: -7      isLast: 0      checksum: -66
From C 收到的数据包为：
seq: 1149      len高: 3      len低: -7      isLast: 0      checksum: -75
From C 收到的数据包为：
seq: 1150      len高: 3      len低: -7      isLast: 0      checksum: 112
From C 收到的数据包为：
seq: 1151      len高: 3      len低: -7      isLast: 0      checksum: 73
超过最大等待时间，重传: checksum: -77  ack: 1096
From C 收到的数据包为：
seq: 1152      len高: 3      len低: -7      isLast: 0      checksum: 8
From C 收到的数据包为：
seq: 1153      len高: 3      len低: -7      isLast: 0      checksum: -25
From C 收到的数据包为：
seq: 1154      len高: 3      len低: -7      isLast: 0      checksum: -95
From C 收到的数据包为：
seq: 1155      len高: 3      len低: -7      isLast: 0      checksum: -34
From C 收到的数据包为：
seq: 1156      len高: 3      len低: -7      isLast: 0      checksum: -51
From C 收到的数据包为：
seq: 1157      len高: 3      len低: -7      isLast: 0      checksum: 101
From C 收到的数据包为：
seq: 1158      len高: 3      len低: -7      isLast: 0      checksum: -127
From C 收到的数据包为：
seq: 1159      len高: 3      len低: -7      isLast: 0      checksum: -82
From C 收到的数据包为：
seq: 1160      len高: 3      len低: -7      isLast: 0      checksum: 110
```

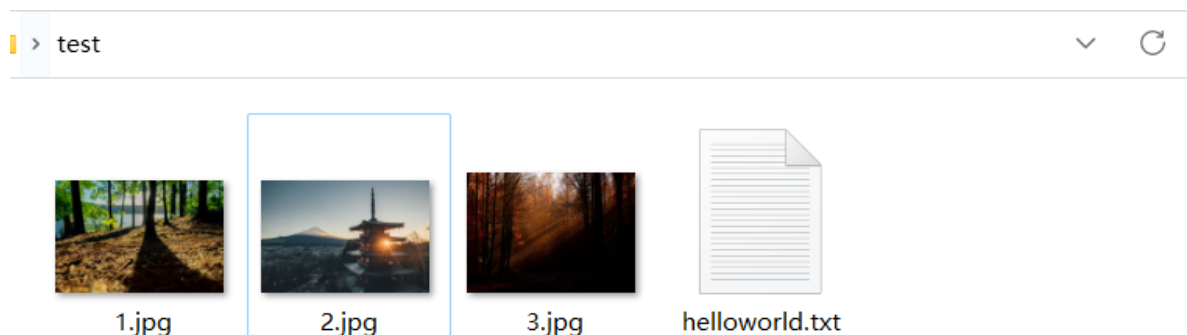
### 正确接受并交付上层

```
checksum: -71 ack: 1624
From C 收到的数据包为:
seq: 1625 len高: 3 len低: -7 isLast: 0 checksum: -98
delivered: 1625
TO C 正确返回数据包
checksum: -71 ack: 1625
From C 收到的数据包为:
seq: 1626 len高: 3 len低: -7 isLast: 0 checksum: 85
delivered: 1626
TO C 正确返回数据包
checksum: -71 ack: 1626
From C 收到的数据包为:
seq: 1627 len高: 3 len低: -7 isLast: 0 checksum: 84
delivered: 1627
TO C 正确返回数据包
checksum: -71 ack: 1627
From C 收到的数据包为:
seq: 1628 len高: 3 len低: -7 isLast: 0 checksum: -36
delivered: 1628
TO C 正确返回数据包
checksum: -71 ack: 1628
From C 收到的数据包为:
seq: 1629 len高: 0 len低: -124 isLast: 1 checksum: 91
delivered: 1629
TO C 正确返回数据包
checksum: -100 ack: 1629
文件保存成功
数据传输完成, 等待客户端退出.....
收到断开连接的请求, 数据包为:
SYN: 0 FIN: 1 SHAKE: 4 checkSum: -6
```


文件传输完成, 并自动进行挥手, 结束连接

```
TO C 正确返回数据包
checksum: -1 ack: 0
From C 收到的数据包为:
seq: 1 len高: 3 len低: -5 isLast: 0 checksum: 116
temp长度: 1019
TO C 正确返回数据包
checksum: -2 ack: 1
From C 收到的数据包为:
seq: 0 len高: 3 len低: -5 isLast: 0 checksum: -102
temp长度: 1019
TO C 正确返回数据包
checksum: -1 ack: 0
From C 收到的数据包为:
seq: 1 len高: 3 len低: -72 isLast: 1 checksum: 113
temp长度: 952
TO C 正确返回数据包
checksum: -2 ack: 1
文件保存成功
数据传输完成, 等待客户端退出.....
收到断开连接的请求, 数据包为:
SYN: 0 FIN: 1 SHAKE: 4 checkSum: -6
挥手包正确
发送给客户端的第二次挥手:
SYN: 0 FIN: 1 SHAKE: 5 checkSum: -7
挥手结束, BYE
服务正常结束
请按任意键继续. . .
```

## 接收文件夹



## 路由器配置

 Router ×

路由器IP:	<input type="text" value="127 . 0 . 0 . 1"/>	服务器IP:	<input type="text" value="127 . 0 . 0 . 1"/>
端口:	<input type="text" value="12770"/>	服务器端口:	<input type="text" value="12660"/>
丢包率:	<input type="text" value="0"/> %	延时:	<input type="text" value="0"/> ms

日志

Router Ready!  
Misscount :0 .  
Delay :0 ms .

注:

client IP: 10.130.187.185 Port: 12880

Router IP: 127.0.0.1 Port: 12770

Server IP:127.0.0.1 Port: 12660