实验 3：基于 UDP 服务设计可靠传输协议并编程实现

实验 3-2：改进实验 3-1

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，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。

(1)实现单向传输。

(2)对于每一个任务要求给出详细的协议设计。

(3)给出实现的拥塞控制算法的原理说明。

(4)完成给定测试文件的传输，显示传输时间和平均吞吐率。

(5)性能测试指标：吞吐率、时延，给出图形结果并进行分析。

(6)完成详细的实验报告（每个任务完成一份）。

(7)编写的程序应结构清晰，具有较好的可读性。

(8)提交程序源码和实验报告。

协议设计

面向连接

建立连接和断开连接的设计参考了 TCP 协议的三次握手和四次挥手。在 TCP 协议中，三次握手过程如下：

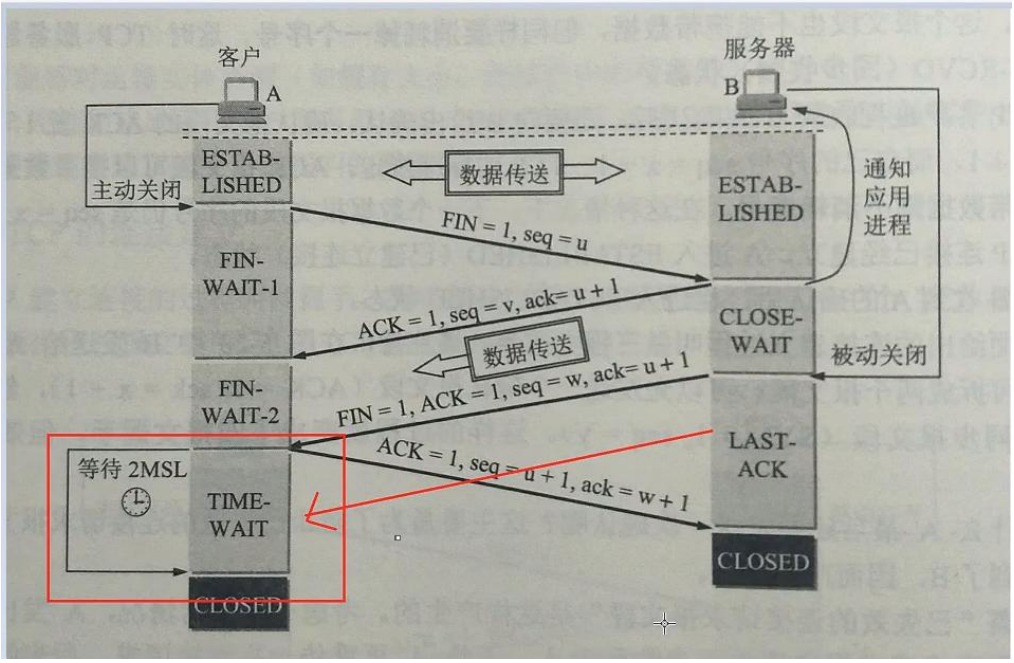
基于此，设计建立连接过程：

① 首先客户端向服务器端发送一个报文，其 SYN 标志位置 1，标志请求建立连接

② 服务器收到请求后，向客户端回复一个报文，SYN 和 ACK 标志位置 1，标志允许建立连接

③ 客户端收到服务器反馈后，向服务器发送一个报文，ACK 置 1，标志可以开始传输

TCP 中四次挥手过程如下：



基于此，设计断开连接过程：

①客户端向服务器端发送一个报文，将 FIN 标志位置 1，标识文件传输完毕请求断开连接

②服务器端收到断开请求后，回应一个报文，将 ACK 标志位置 1，标识接到断开请求

③服务器端向客户端发送一个报文，将 A=FIN 标志位置 1，标识请求断开连接

④客户端收到断开请求后，回应一个报文，将 ACK 标志位置 1，标识接到断开请求。之后客户端在等待两个 MSL 时间后，确保不再接收到服务端的数据包（即服务端已经收到客户端的回应，不会再进行重传操作）后，再关闭。

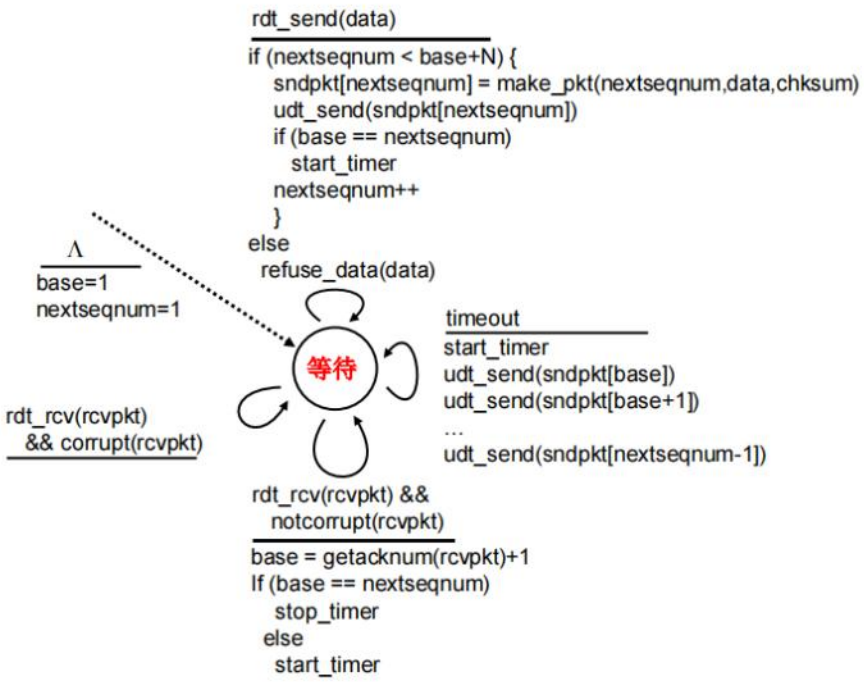
拥塞设置：阻塞模式与非阻塞模式

由于重传机制的实现需要对数据报的发送和接收进行计时，但是初始

化 socket 时默认是阻塞态的 socket，调用 recvfrom 函数后线程被阻塞，计时函数也不能正常运行。如果我们在阻塞态调用 recvfrom 那么计时函数就需要新开一个线程，为了避免这种麻烦，我们需要计时重传的阶段调用以下代码，将 socket 切换为非阻塞态。

可靠数据传输：使用基于滑动窗口的流量控制机制的 rdt3.0

发送端有限状态状态机如图：



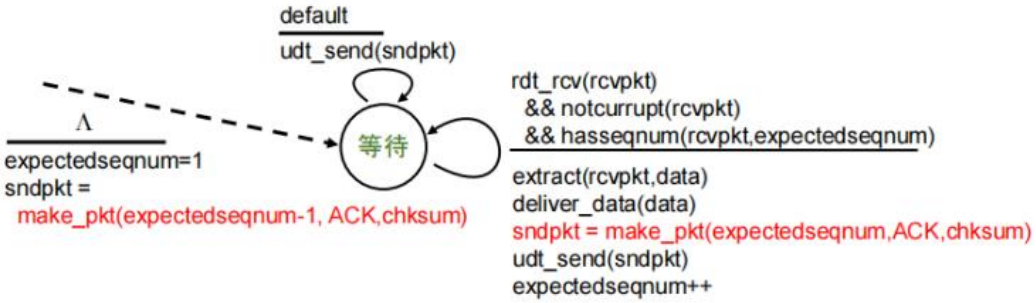
重传策略采用 GBN（滑动窗口）：

1. 当滑动窗口有空闲时，持续发送数据包；

2. 发送一个数据包后，看是否收到了确认包，收到了则进行判断是否需要确认重传或者滑动窗口；

3. 判断是否有超时的情况出现，有则进行重传

接收端有限状态机如图：



采取累计确认:

1. 当收到正确 seq 的数据包，将其存储到缓存空间表示，并等待下一个数据包。

2. 当收到错误 seq 的数据包，则丢弃，并发送对最新 seq 数据包的确认包

超时重传

服务器端对于窗口中第一个数据包进行计时，当超时时，重发窗口中的所有数据报，并重新开始计时。

差错检测

模仿 tcp，发送方发送报文前先计算 checksum 并封装到包内，接收方收到包进行校验，如果正确则正确接收，如果错误则忽略。发送方与接收方相似，但当发送方接收到错误的校验和时，会重发窗口中的所有数据包。

功能实现

报文结构定义和一些宏定义

报文结构

```
struct PacketHead {
    u_int seq;
    u_int ack;
    u_short checkSum;
    u_short bufSize;
    char flag;
    u_char windows;

    PacketHead() {
        seq = ack = 0;
        checkSum = bufSize = 0;
        flag = 0;
        windows = 0;
    }
};

struct Packet {
    PacketHead head;
    char data[MAX_DATA_SIZE];
};
```

宏定义

```
#define SYN 0x1
#define ACK 0x2
#define FIN 0x4
#define END 0x8
#define MAX_DATA_SIZE 8192
#define MAX_SEQ 0xffff
#define OUTPUT_LOG
#define min(a, b) a>b?b:a
#define max(a, b) a>b?a:b
```

发送端：

```
static SOCKADDR_IN addrSrv;
static int addrLen = sizeof(addrSrv);
double MAX_TIME = CLOCKS_PER_SEC;
static int windowSize = 16;
static unsigned long int base = 0;//握手阶段确定的初始序列号
static unsigned long int nextSeqNum = 0;
static Packet *sendPkt;
static unsigned long int sendIndex = 0, recvIndex = 0;
static bool stopTimer = false;
static clock_t start;
static int packetNum;
```

接收端：

```
#define PORT 7879
#define ADDRSRV "127.0.0.1"
#define MAX_FILE_SIZE 100000000
double MAX_TIME = CLOCKS_PER_SEC;
static u_long base_stage = 0;
static int windowSize = 16;
char fileBuffer[MAX_FILE_SIZE];
```

计算校验和

```
u_short CheckPacketSum(u_short *packet, int packetLen) {

    u_long sum = 0;
    int count = (packetLen + 1) / 2;
    //u_short *temp = new u_short[count];
    u_short temp1[count];
    u_short *temp = temp1;
    memset( Dst: temp,  Val: 0,  Size: 2 * count);
    memcpy( Dst: temp,  Src: packet,  Size: packetLen);

    while (count--) {
        sum += *temp++;
        if (sum & 0xFFFF0000) {
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

发送端判断是否在窗口内

```
u_int waitingNum(u_int nextSeq) {
    if (nextSeq >= base)
        return nextSeq - base;
    return nextSeq + MAX_SEQ - base;
}

bool connectToServer(SOCKET &socket, SOCKADDR_IN &addr) {...}

bool disConnect(SOCKET &socket, SOCKADDR_IN &addr) {...}

Packet makePacket(u_int seq, char *data, int len) {...}

bool inWindows(u_int seq) {
    if (seq >= base && seq < base + windowSize)
        return true;
    if (seq < base && seq < ((base + windowSize) % MAX_SEQ))
        return true;

    return false;
}
```

主函数

发送端：

```
int main() {
    WSAData wsaData{};
    if (WSAStartup( wVersionRequested: MAKEWORD(2, 2), lpWSAData: &wsaData) != 0) {
        //加载失败
        cout << "加载DLL失败" << endl;
        return -1;
    }
    SOCKET sockClient = socket( af: AF_INET, type: SOCK_DGRAM, protocol: 0);

    u_long imode = 1;
    ioctlsocket( s: sockClient, cmd: FIONBIO, argp: &imode);//非阻塞

    addrSrv.sin_family = AF_INET;
    addrSrv.sin_port = htons( hostshort: PORT);
    addrSrv.sin_addr.S_un.S_addr = inet_addr( cp: ADDRSRV);

    if (!connectToServer( &: sockClient, &: addrSrv)) {
        cout << "连接失败" << endl;
        return 0;
    }
    sendPkt=new Packet[windowSize];
    string filename = R"(D:\wtx\computer-network\computer-network\Lab\lab3\3-2\code\test\in\1.jpg)";
```

```
    ifstream infile( s: filename,  mode: ifstream::binary);

    if (!infile.is_open()) {
        cout << "无法打开文件" << endl;
        return 0;
    }

    infile.seekg(0, infile.end);
    u_long fileLen = infile.tellg();
    infile.seekg(0, infile.beg);
    cout << fileLen << endl;

    char *fileBuffer = new char[fileLen];
    infile.read( s: fileBuffer,  n: fileLen);
    infile.close();
    cout << "开始传输" << endl;

    clock_t start_time = clock();
    sendFSM( len: fileLen, fileBuffer,  &: sockClient,  &: addrSrv);
    clock_t end_time = clock();
    cout << "传输总时间为:" << (end_time - start_time) / CLOCKS_PER_SEC << "s" << endl;
    cout << "吞吐率为:" << ((float)fileLen) / ((float)(end_time - start_time) / CLOCKS_PER_SEC) << "byte/s" << endl;

    if (!disConnect( &: sockClient,  &: addrSrv)) {
        cout << "断开失败" << endl;
        return 0;
    }
    cout << "文件传输完成" << endl;
    return 1;
}
```

接收端：

```
int main() {
    WSAData wsaData{};
    if (WSAStartup( wVersionRequested: MAKEWORD(2, 2),  lpWSAData: &wsaData) != 0) {
        //加载失败
        cout << "加载DLL失败" << endl;
        return -1;
    }
    SOCKET sockSrv = socket( af: AF_INET,  type: SOCK_DGRAM,  protocol: 0);

    SOCKADDR_IN addrSrv;
    addrSrv.sin_family = AF_INET;
    addrSrv.sin_port = htons( hostshort: PORT);
    addrSrv.sin_addr.S_un.S_addr = inet_addr( cp: ADDRSRV);
    bind( s: sockSrv,  name: (SOCKADDR *) &addrSrv,  namelen: sizeof(SOCKADDR));

    SOCKADDR_IN addrClient;

    //三次握手建立连接
    if (!acceptClient( &: sockSrv,  &: addrClient)) {
        cout << "连接失败" << endl;
        return 0;
    }
```

```
    //可靠数据传输过程
    u_long fileLen = recvFSM( filebuffer: fileBuffer,  &: sockSrv,  &: addrClient);

    //写入复制文件
    string filename = R"(D:\wtx\computer-network\computer-network\Lab\lab3\3-2\code\test\out\1_recv.jpg)";
    ofstream outfile( s: filename,  mode: ios::binary);
    if (!outfile.is_open()) {
        cout << "打开文件出错" << endl;
        return 0;
    }
    cout << fileLen << endl;
    outfile.write( s: fileBuffer,  n: fileLen);
    outfile.close();

    cout << "文件复制完毕" << endl;

    //四次挥手断开连接
    disConnect( &: sockSrv,  &: addrClient);
    return 1;
}
```

客户端（发送端）有限状态机

```
void sendFSM(u_long len, char *fileBuffer, SOCKET &socket, SOCKADDR_IN &addr) {

    packetNum = int(len / MAX_DATA_SIZE) + (len % MAX_DATA_SIZE ? 1 : 0);
    //sendIndex==packetNum时，不再发送；recvIndex==packetNum，收到全部ACK，结束传输
    sendIndex = 0, recvIndex = 0;
    int packetDataLen;
    addrLen = sizeof(addr);

    stopTimer = false;//是否停止计时
    char *data_buffer = new char[packetDataLen], *pkt_buffer = new char[sizeof(Packet)];
    Packet recvPkt;
    nextSeqNum = base;

    sendPkt[windowSize];
    cout << "本次文件数据长度为" << len << "Bytes,需要传输" << packetNum << "个数据包" << endl;
```

```
    while (true) {
        if (recvIndex == packetNum) {
            //recv全部ACK，结束传输，发送END报文
            PacketHead endPacket;
            endPacket.flag |= END;
            endPacket.checkSum = CheckPacketSum( packet: (u_short *) &endPacket,  packetLen: sizeof(PacketHead));
            memcpy( Dst: pkt_buffer,  Src: &endPacket,  Size: sizeof(PacketHead));
            sendto( s: socket,  buf: pkt_buffer,  len: sizeof(PacketHead),  flags: 0,  to: (SOCKADDR *) &addr,  tolen: addrLen);

            while (recvfrom( s: socket,  buf: pkt_buffer,  len: sizeof(PacketHead),  flags: 0,  from: (SOCKADDR *) &addr,  fromlen: &addrLen) <= 0) {
                if (clock() - start >= MAX_TIME) {
                    start = clock();
                    goto resend;
                }
            }

            if ((((PacketHead *) (pkt_buffer))->flag & ACK) &&
                    (CheckPacketSum( packet: (u_short *) pkt_buffer,  packetLen: sizeof(PacketHead)) == 0)) {
                cout << "文件传输完成" << endl;
                return;
            }

            resend:
            continue;
        }
```

```
        packetDataLen = min(MAX_DATA_SIZE, len - sendIndex * MAX_DATA_SIZE);

        //如果下一个序列号在滑动窗口中
        if (inWindows( seq: nextSeqNum) && sendIndex < packetNum) {

            memcpy( Dst: data_buffer,  Src: fileBuffer + sendIndex * MAX_DATA_SIZE,  Size: packetDataLen);
            //缓存进入数组
            sendPkt[(int) waitingNum( nextSeq: nextSeqNum)] = makePacket( seq: nextSeqNum,  data: data_buffer,  len: packetDataLen);
            memcpy( Dst: pkt_buffer,  Src: &sendPkt[(int) waitingNum( nextSeq: nextSeqNum)],  Size: sizeof(Packet));
            //发送给接收端
            sendto( s: socket,  buf: pkt_buffer,  len: sizeof(Packet),  flags: 0,  to: (SOCKADDR *) &addr,  tolen: addrLen);

            //如果目前窗口中只有一个数据报，开始计时（整个窗口共用一个计时器）
            if (base == nextSeqNum) {
                start = clock();
                stopTimer = false;
            }
            nextSeqNum = (nextSeqNum + 1) % MAX_SEQ;
            sendIndex++;
            //cout << sendIndex << "号数据包已经发送" << endl;
        }
```

```
        //判断是否有ACK到来
        while (recvfrom( s: socket,  buf: pkt_buffer,  len: sizeof(Packet),  flags: 0,  from: (SOCKADDR *) &addr,  fromlen: &addrLen) > 0) {
            memcpy( Dst: &recvPkt,  Src: pkt_buffer,  Size: sizeof(Packet));
            //corrupt
            if (CheckPacketSum( packet: (u_short *) &recvPkt,  packetLen: sizeof(Packet)) != 0 || !(recvPkt.head.flag & ACK))
                goto time_out;
            //not corrupt
            if (base < (recvPkt.head.ack + 1)) {
                //不是窗口外的ACK
                int d = recvPkt.head.ack + 1 - base;
                for (int i = 0; i < (int) waitingNum( nextSeq: nextSeqNum) - d; i++) {
                    sendPkt[i] = sendPkt[i + d];
                }
                recvIndex+=d;
                cout << "[window move]base:" << base << "\tnextSeq:" << nextSeqNum << "\tendWindow:" << base + windowSize << endl;
            }
            base = (max((recvPkt.head.ack + 1), base)) % MAX_SEQ;
            //当窗口为空，停止计时
            if (base == nextSeqNum)
                stopTimer = true;
            else {
                start = clock();
                stopTimer = false;
            }

        }
```

```
        //超时发生，将数组中缓存的数据报全部重传一次，这就是Go Back N
        time_out:
        if (!stopTimer && clock() - start >= MAX_TIME) {
            //cout << "resend" << endl;
            for (int i = 0; i < (int) waitingNum( nextSeq: nextSeqNum); i++) {
                memcpy( Dst: pkt_buffer,  Src: &sendPkt[i],  Size: sizeof(Packet));
                sendto( s: socket,  buf: pkt_buffer,  len: sizeof(Packet),  flags: 0,  to: (SOCKADDR *) &addr,  tolen: addrLen);
            }
            cout << "第" << base << "号数据包超时重传" << endl;
            start = clock();
            stopTimer = false;
        }
    }
}
```

服务端（接收端）有限状态机

```
u_long recvFSM(char *filebuffer, SOCKET &socket, SOCKADDR_IN &addr) {
    u_long fileLen = 0;
    int addrLen = sizeof(addr);
    u_int expectedSeq = base_stage;
    int dataLen;

    char *pkt_buffer = new char[sizeof(Packet)];
    Packet recvPkt, sendPkt= makePacket( ack: base_stage - 1);//缓存的ACK报文

    while (true) {
        memset( Dst: pkt_buffer, Val: 0, Size: sizeof(Packet));
        recvfrom( s: socket, buf: pkt_buffer, len: sizeof(Packet), flags: 0, from: (SOCKADDR *) &addr, fromlen: &addrLen);
        memcpy( Dst: &recvPkt, Src: pkt_buffer, Size: sizeof(Packet));
        //ShowPacket(&recvPkt);

        if ((recvPkt.head.flag & END) && CheckPacketSum( packet: (u_short *) &recvPkt, packetLen: sizeof(PacketHead)) == 0) {
            cout << "传输完毕" << endl;
            PacketHead endPacket;
            endPacket.flag |= ACK;
            endPacket.checkSum = CheckPacketSum( packet: (u_short *) &endPacket, packetLen: sizeof(PacketHead));
            memcpy( Dst: pkt_buffer, Src: &endPacket, Size: sizeof(PacketHead));
            sendto( s: socket, buf: pkt_buffer, len: sizeof(PacketHead), flags: 0, to: (SOCKADDR *) &addr, tolen: addrLen);
            return fileLen;
        }
```

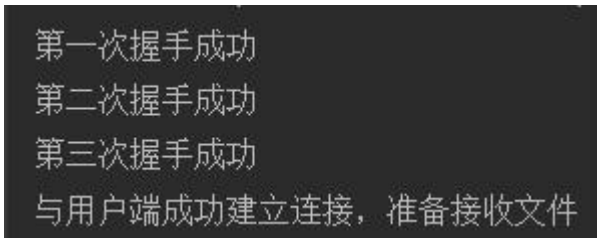
```
        if((recvPkt.head.seq==expectedSeq) && (CheckPacketSum( packet: (u_short *) &recvPkt, packetLen: sizeof(Packet)) == 0)){
            //correctly receive the expected seq
            dataLen = recvPkt.head.bufSize;
            memcpy( Dst: filebuffer + fileLen, Src: recvPkt.data, Size: dataLen);
            fileLen += dataLen;

            //give back ack=seq
            sendPkt = makePacket( ack: expectedSeq);
            memcpy( Dst: pkt_buffer, Src: &sendPkt, Size: sizeof(Packet));
            expectedSeq=(expectedSeq+1)%MAX_SEQ;
            continue;
        }
        //不是期望的Seq或传输过程出错，重传ACK
        memcpy( Dst: pkt_buffer, Src: &sendPkt, Size: sizeof(Packet));
        sendto( s: socket, buf: pkt_buffer, len: sizeof(Packet), flags: 0, to: (SOCKADDR *) &addr, tolen: addrLen);
    }
}
```

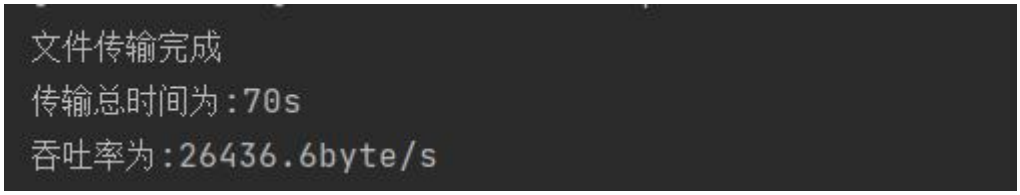
输出结果

建立连接

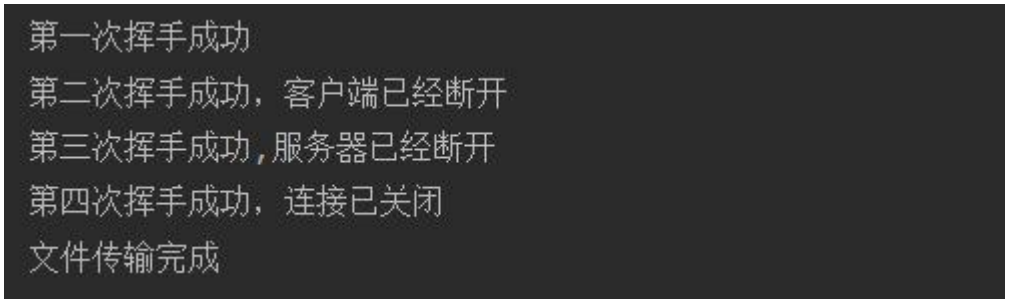
```
第一次握手成功
[SYN:1  ACK:1   FIN:0   END:0]SEQ:0 ACK:0   LEN:0
第二次握手成功
[SYN:0  ACK:1   FIN:0   END:0]SEQ:0 ACK:0   LEN:0
三次握手成功
成功与服务器建立连接，准备发送数据
```

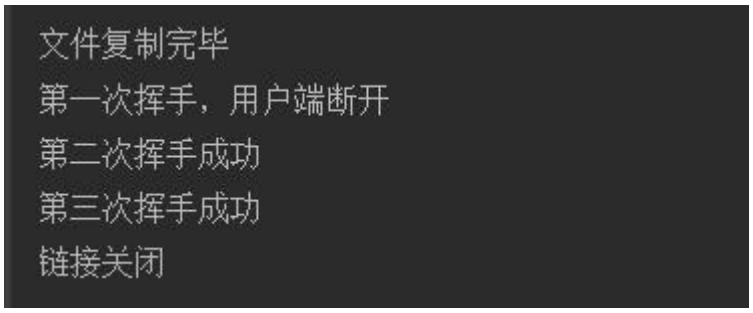


传输时间、吞吐率

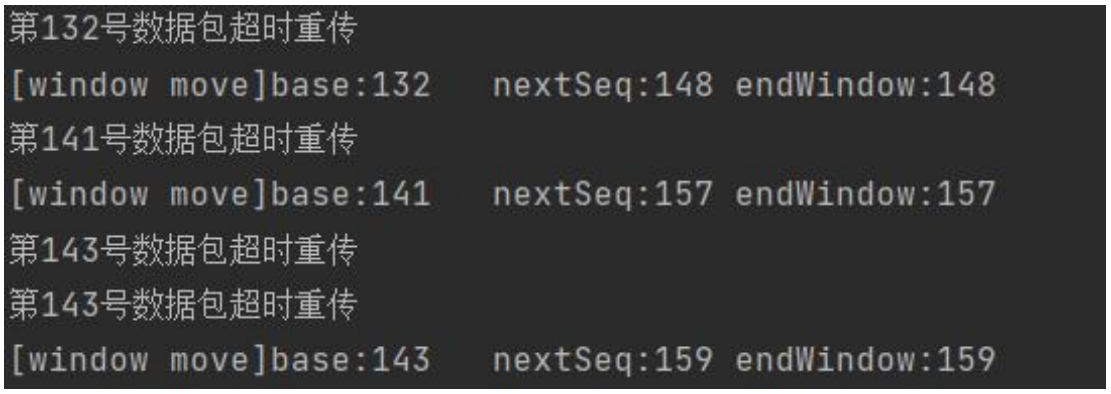


断开连接





文件传输过程部分截图（包括超时重传与窗口滑动）



遇到问题

Q：在 clion 运行项目时，选择运行时无法进行吞吐率的输出以及断开连接，但是选择调试时可以正确执行。

A：暂未解决。