UDP可靠传输 -- Part 01

学号:2111252　姓名：李佳豪　时间: 11.17

> 发送端RDT3.0基础加入ST、END的状态，分别表示第一个数据包和最后一个包（挥手）；接受端RDT2.2基础加入对应的ST、END。握手则新增一个状态机；

默认参数：(这组参数基本可以遇见所有情况，丢包、延时时间超过Sender端超时时间、失序)

> Sender、Reciver端口分别 8998 8999
>
> ```
> const int MAX_DELAY_MS = 250; // 最大延迟时间（ms）
> const int MAX_DELAY_RATE = 0.01; //发送最大延迟的概率
> const double LOSS_RATE = 0.1; // 丢包率（0.1 表示10%的丢包率）
> ```
>
> #define MAX_TIME 0.2*CLOCKS_PER_SEC　//超时时间 即200ms

代码链接:

> https://github.com/FondH/cn/tree/master/StableUdp

实验要求分析（Rdt3.0 --2.2）

1、如何建立面向连接的可靠传输

> 目的面向连接的传输，虽然UDP本身是无连接的，但可以在应用层模拟出连接的概念，从而保证通信双方信息的**可靠**和**顺序**。发送端使用Rdt3.0（实际有修改）接受端rd2.2。

- **首先**保证信道的可靠，那么在正式通信前要保证连接的建立，可以模仿TCP的三次握手机制（但其实我们只涉及到Reciver和Sender，两次够用）；在结束通信后，确认不再有消息后，双方进行挥手关闭端口、释放socket资源。
- **其次**是数据的可靠和顺序，实验使用`rdt3.0`自动机的停等机制、使用0、1两个状态转移已经保证了数据包的顺序，而可靠性则通过自动机提到的**校验和**以及**超时重传**机制进行保证

2、连接的建立和消亡

> **握手**（三次）：在开始传输数据之前，客户端和服务器可以交换控制消息来建立这种“虚拟连接”。涉及到发送一个特殊的`SYN`（同步）数据包，然后由接收方回复一个`SYN-ACK`数据包，最后客户端发送一个`ACK`数据包来确认；**挥手通过**后，才进入Rdt3.0和2.2进行文件传输。

> **挥手**（两次）：当传输到文件的最后一个部分，发送端发送<END,ACK>数据包，表示发送完这个文件就结束，等待接受端回复<END,ACK>即表明接收端已经接受所有任务，可以断开。

3、差错检验和回复

- 使用序列号和确认机制检测丢包。本次实验序列号即状态机的0/1，但是为了后续动态窗口，这里提出序列号概念以供后续扩展。
- 使用校验计算计算数据在传输过程中是否损坏，使用上课中学习的累加的方式计算。

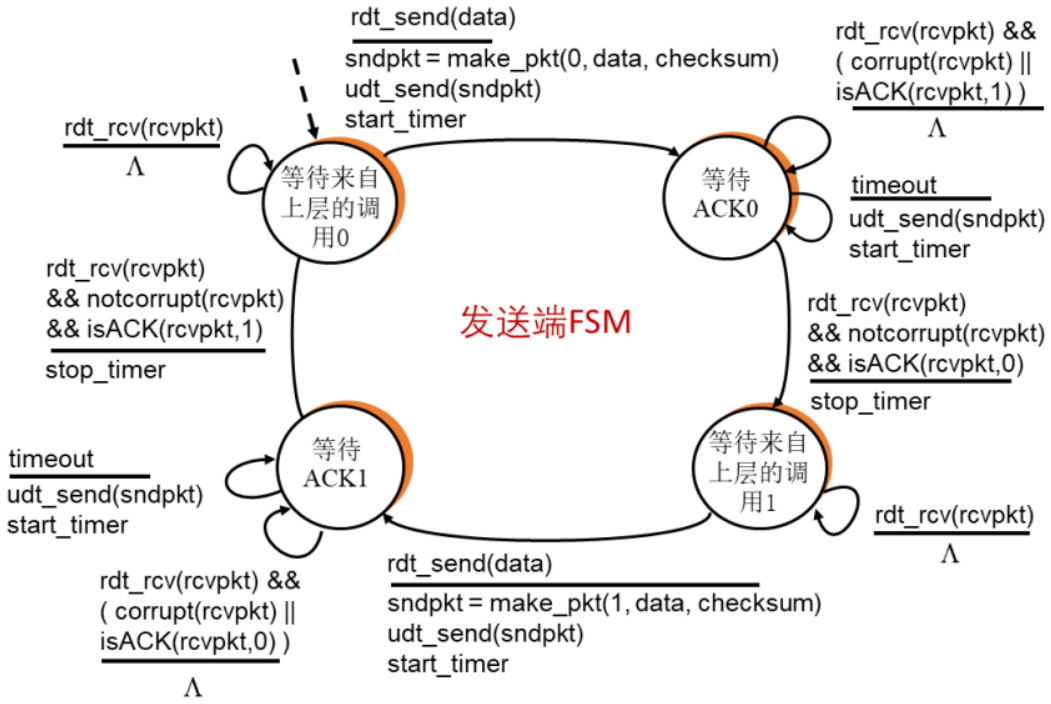
4、停等机制和接受确认

> 一问一答形式，Reciver每次发送一个数据包，等待接受端回复Ack。

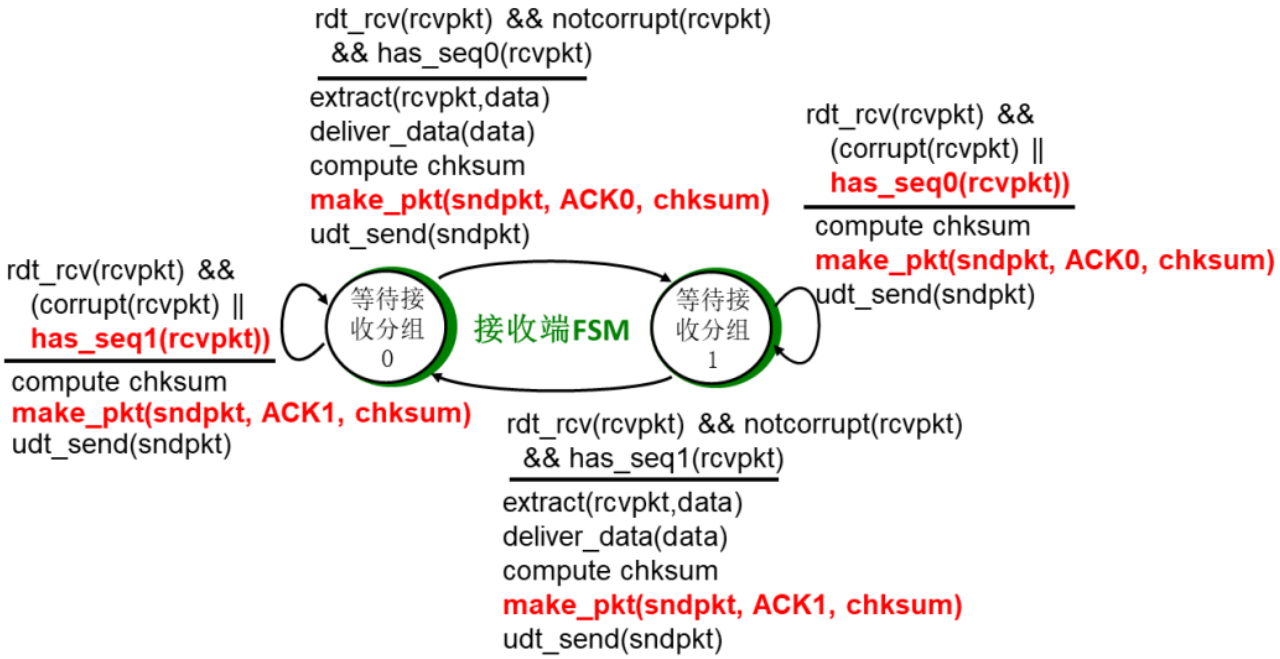
5、超时重传

> 发送端采取如下状态机方式，每次发送完一组数据开始计时，若超时重新发送；接受端不设计计时。

■ rdt3.0：发送端状态机

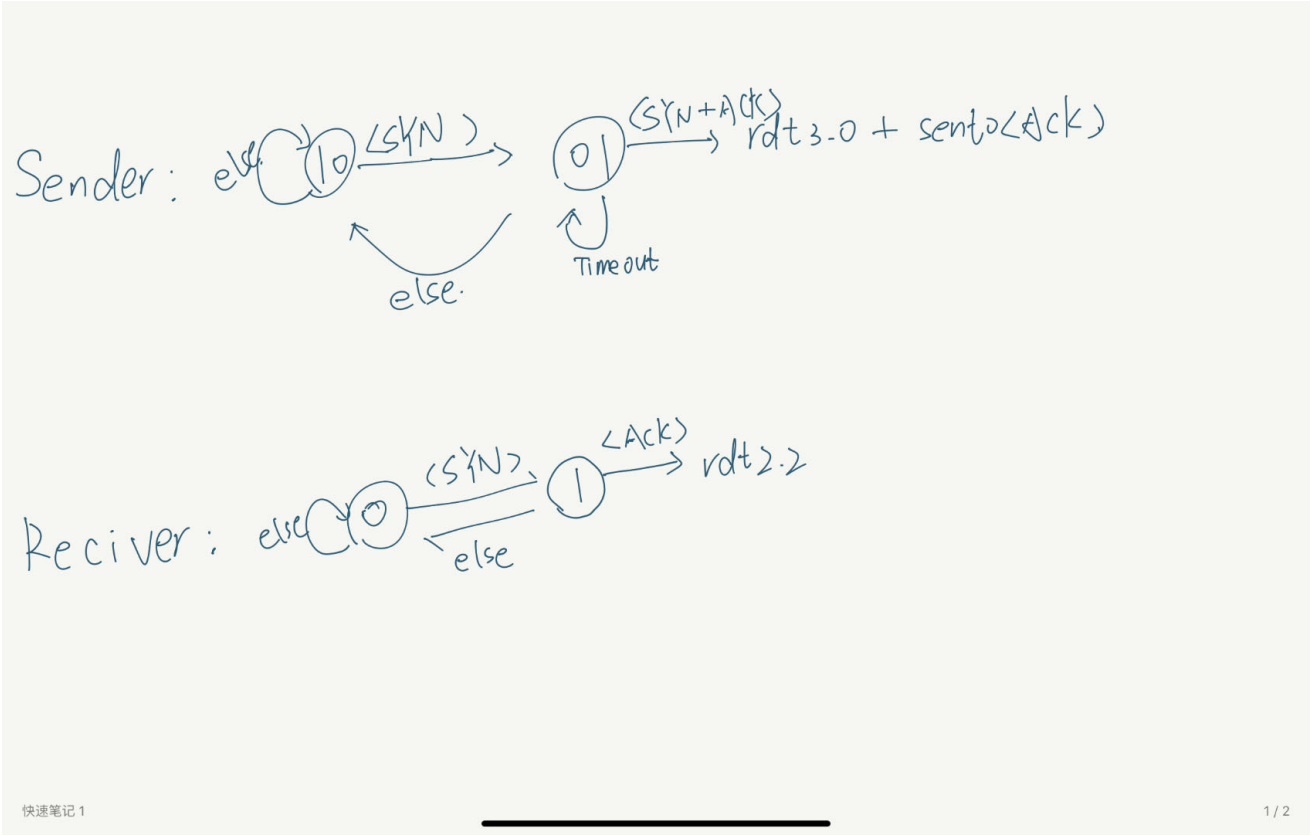


■ rdt2.2：接收端状态机



信道的可靠性进一步措施，采取三次握手的过渡?

根据上述分析，新增一个停等机制为基础的握手状态机，挥手成功后进入RDT3.0状态机。



程序设计

1、UDP报文格式

```
 0             7 0             7 0                           15
+-----------------------------------------------------------+
|            Flag               |          Checksum         |
+-----------------------------------------------------------+
|                         Sequence Number                   |
+-----------------------------------------------------------+
|                     Acknowledgment Number                 |
+-----------------------------------------------------------+
|           Payload Length      |       Header Length       |
+-----------------------------------------------------------+
|                             Data                          |
+-----------------------------------------------------------+
```

字段名称	大小 (位)	描述
Flag	16	0... St End Status SYN ACK FIN
Checksum	16	整个报文的校验和
Sequence Number	32	报文的序列号
Acknowledgment Number	32	确认序列号
Payload Length	16	Data的长度
Header Length	16	报文头部的长度

字段名称	大小 (位)	描述
Data	——	实际数据

```
using streamsize = long long;
struct Header {
    uint16_t flag;
    uint16_t checksum;
    uint32_t seq;
    uint32_t ack;
    uint32_t data_size;
    Header() : flag(0), checksum(0), seq(0), ack(0), data_size(0) {}
    void set_bit(int f, bool value) {
        value ? this->flag |= (uint16_t)1 << f : flag &= ~((uint16_t)1 << f);
    }
    void set_St(bool va) { set_bit(5, va); }
    ... ...
    int get_Ack() { return (flag & (0x1)) > 0 ? 1 : 0; }
};

class Udp {
public:
    Header  header;
    char payload[PayloadSize] = { '0' };
    void set(Header& h, char* c) {
        this->header = h;
        memcpy(payload, c, PayloadSize);
    }
    ... ...

    //这里关于校验和
    void set_cheksum() {this->header.checksum = ~calc_cheksum();}
    uint16_t calc_cheksum() {
        /*定一个16位的指针，开始遍历这个UDP*/
        int count = (PacketSize) / 2;  //16位，2字节除2
        uint16_t* buf = (uint16_t*)this;
        int res = 0;
        while (count--) {
            res += *buf++;
            if (res & 0x10000) { //溢出加到末尾
                res &= 0xffff;
                res +=1;
            }
        }
        return (res & 0xffff);

    }
    bool cmp_cheksum() { //通过校验和
        uint16_t ut = calc_cheksum();
        return (ut | header.checksum)==0xffff;
    }
};
```

2、Sender

1. Sender类

```
class Sender {
private:
    ...
public:
    Sender();
    int _send(int size);  //对sendto函数封装，size为数据段大小
    int get_connection(); //对握手、rdt3.0两个状态机结合
    ...
    friend DWORD WINAPI ConnectHandler(LPVOID param);//rdt3.0
    friend DWORD WINAPI SendHandler(LPVOID param);//三次握手
    ~Sender();
};
```

2.三次握手发送端状态机思路

同样的停等机制为基础。 发送端要做的是使用两个Bool变量(SYN、ACK)表示当前处于状态机的某个状态，根据每一个状态打包对应报文发送，然后进入等待，若超时即重传上一次，具体代码逻辑不赘述。

3.Rdt3.0状态机实现思路

在与接收端进行文件传输时，使用报文区的status（0和1）控制状态机的不同状态，st、end决定发送端发送的第一个报文还是最后一个报文。

- 若是第一个报文，st为1，发送端在这个报文的数据端发送fileName:fileSize的内容，然后进入状态<st=1,status=0>的等待恢复阶段；当收到回复为<st=1,status=0,ACK=1>的报文，且通过校验和，则让本地的st置0，status进入1；

```
    bool status = 0;
    bool st = 1;
    bool fin = 0;  //对应上述介绍的END位

    clock_t send_st = clock();
    while (sender->send_runner_keep && !fin) {
        //发送seq状态0/1的包
        if (st)
        {

            sender->package.set_status(status);
            sender->package.set_flag(st, fin);
            sender->package.packet_data(name_size.c_str(), payload_size);

            sender->_send(payload_size);
            //cout.write(sender->package.payload, sizeof(name_size));
            //cout << endl;
            st = false;
        }
```

- 后面则正式传输文件，发送的第二个报文实际是<st=0, status=1>的报文，他需要等待<status=1, st=0>的报文，才可以让自己status变为0；否则，无论是超时还是校验和未通过、报文flag段的状态和当前期待的状态对应不上，都会导致重传（这时候重传，只需要根据当前status、st、fin变量和实现分段传输文件的指针打包数据包即可）。

- 当发生到最后一个包时（遍历文件分段传输的指针超出文件大小时），使得END位至1，发送<END=1,STATUS=X>的包，等待回复<END=1,STATUS=X>，这里发送可以看作两次挥手的第一次挥手，等待回复的报文作为第二次挥手。

3、Reciver

相比Sender，Reciver并没有超时重传的概念。

1. Reciver类 和Sender相似。

```
class Reciver {
    SOCKET s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    ....
    int bytes = 0; //sum_of sended bytes
public:
    Reciver();
    void _send(char* tb, bool IsDelay);//同样sendto的封装
    void ThreadSend(); //这里和模拟包的延迟有关，在最后一节详细说
    int get_connection();
    void to_file();
    friend DWORD WINAPI ReciHandler(LPVOID param);
    friend DWORD WINAPI ConnectHandler(LPVOID param);
    void init();
};
```

2. 握手阶段状态机思路
    - 状态只需要SYN 一个变量控制。
    - 根据recifrom报文的SYN、ACK段发出对应报文即可。具体见状态机
3. Rdt2.2思路

    和发送端的Rdt3.0相比，这里简化了超时重传的检查，也简化了对于校验和、接受报文Status对应不上的要重传机制。（也就是忽略一切重传，这里我现场答辩时可能表达错了）；当收到END=1报文，且自己发送完<END,ACK>后，对存在Buffer的文件输出。

关于丢包和延迟？

现有的路由MFC程序对丢包的实现机制是每隔n个包进行一次丢失，可以模拟丢包产生的超时现象。但是它对于延迟的模拟是将每一个包进行延迟固定的秒数，显然这种模拟只可以测试是否程序可以超时重传的机制，但无法模拟现实中由于每一个包**延迟时间不同导致的失序问题**，失序问题会导致最后传输的文件丢失内容。

1. 丢包模拟：每一次调用sendto函数前都使用rand()函数生成概率，决定是否发送这个包。

```
const double LOSS_RATE = 0.01; // 丢包率（0.1 表示10%的丢包率）

void SimulateDelay() {
    double i = rand() / RAND_MAX;
    if(i < MAX_DELAY_RATE)
        Sleep(MAX_DELAY_MS);
}
int Sender::_send(int size){
    if (!SimulateDrop()) {
            SimulateDelay();
            rst_byte = sendto(this->s, this->SendBuffer, size + HeadSize, 0,
(sockaddr*)this->dst_addr, sizeof(*this->dst_addr));
    ... ...
}
```

2. 延迟模拟：在接受端模拟，在他调用sendto函数时，将sendto发送的内容memcopy放入线程，thread里使用rand()生成概率决定这次延迟的时间为X秒，（X最大设置为MAX_DELAY_MS），Sleep（X）后进行sendto。

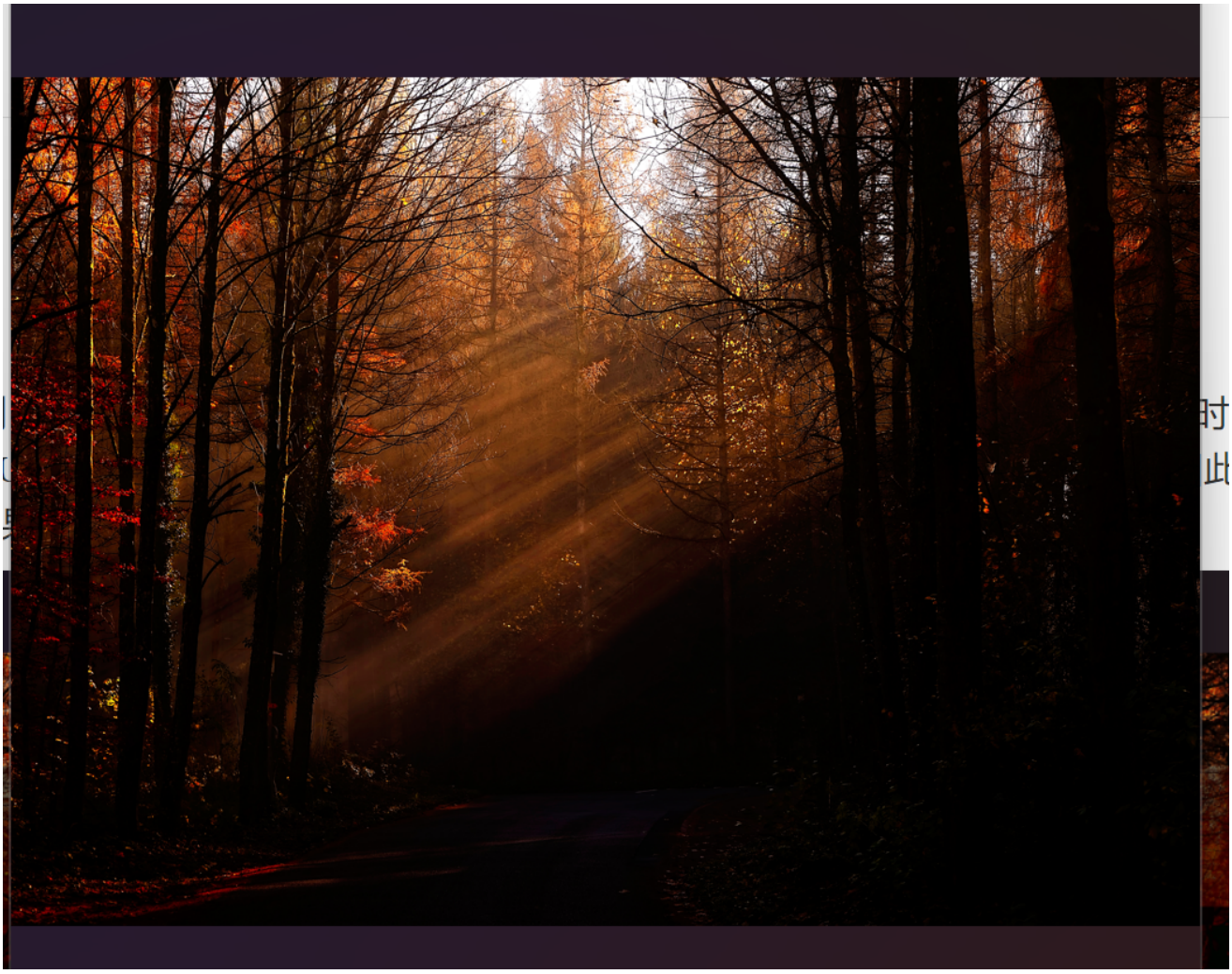
```
const int MAX_DELAY_MS = 100; // 最大延迟时间（毫秒）
const double MAX_DELAY_RATE = 0.1;

void SimulateDelay(bool IsDelay) {
    if (IsDelay)
        Sleep(MAX_DELAY_MS);
}

void Reciver::_send((char* tb, bool IsDelay){
    cout << "\n+-+-+-+-+-+- Reciver' Packetage  -+-+-+-+-+-+\n";
    ... ...

    SimulateDelay(IsDelay);
    int rst_byte = sendto(this->s, (const char*)tb, HeadSize, 0, (struct sockaddr*)this-
>send_addr, sizeof(*(this->send_addr)));
    return;
}
void Reciver::ThreadSend() {
        bool IsDelay = 0;
        double i = rand() / (double)RAND_MAX;
        if (i < MAX_DELAY_RATE)
            IsDelay = 1;
        //复制包
        this->package.set_cheksum();
        char* threadBuffer = new char[PacketSize];
        memcpy(threadBuffer, &this->package, HeadSize);

        thread send_thread(&Reciver::_send, this,threadBuffer, IsDelay);
        send_thread.detach();
}
```

延迟导致的失序

由于刚刚模拟了现实中产生的不同包延迟时间不同， 出现了发送端接受ACK时，接受的<STATUS=X, Seq=S>的包，可能是接受端上上个发送的<STATUS=X, Seq=S-2>，因此导致最后传输结果不对，比如下图

而导致这种的原因正如上述分析的，由于只验证Status，中间Sender接受的ACK可能是之前某个包的，除非延迟不能过高，否则就会导致数据失序、丢失导致的传输不可靠。这里修改验证RDT3.0 关于STATUS是否对应的验证修改为对ack的验证，即验证上一次发送报文的ACK是否等于这一次接受报文的SEQ。

这样问题解决。



最终解释程序输出的日志

发送端

1. 先输入文件名字（同一路径下） 然后挥手成功的信息，SEQ： Num表示当前发送的seq序号，底下的Flag: 1, Checksum: 56765, Sequence: 670, Acknowledgment: 770, Data Size: 16384 表示报文信息。

```
input filename
3.jpg
File: 3.jpg is Loading
Size: 2922.12 KB
Dst Prot: 8999
Try to connect



------------------------- SEQ: 0 --------------------------
Flag: 4, Checksum: 65483, Sequence: 0, Acknowledgment: 0, Data Size: 0

------------------- Dst Package ----------------
SYN: 1 ACK: 1
第二次挥手成功
Send Thread Ready

------------------------- SEQ: 0 --------------------------
Flag: 1, Checksum: 3, Sequence: 0, Acknowledgment: 0, Data Size: 0


start to send:
q to exit

------------------------- SEQ: 0 --------------------------
Flag: 21, Checksum: 65451, Sequence: 0, Acknowledgment: 0, Data Size: 40
----- Check correct -----
```

2. 输出--correct--表示收到的ACK验证通过。 ----check error---表示验证失败，重传

3. 输出---TIME OUT---则表示延迟重传，若带有----DROP----的，则是表明发送了丢包导致的超时，否则一般是单纯延迟导致的。

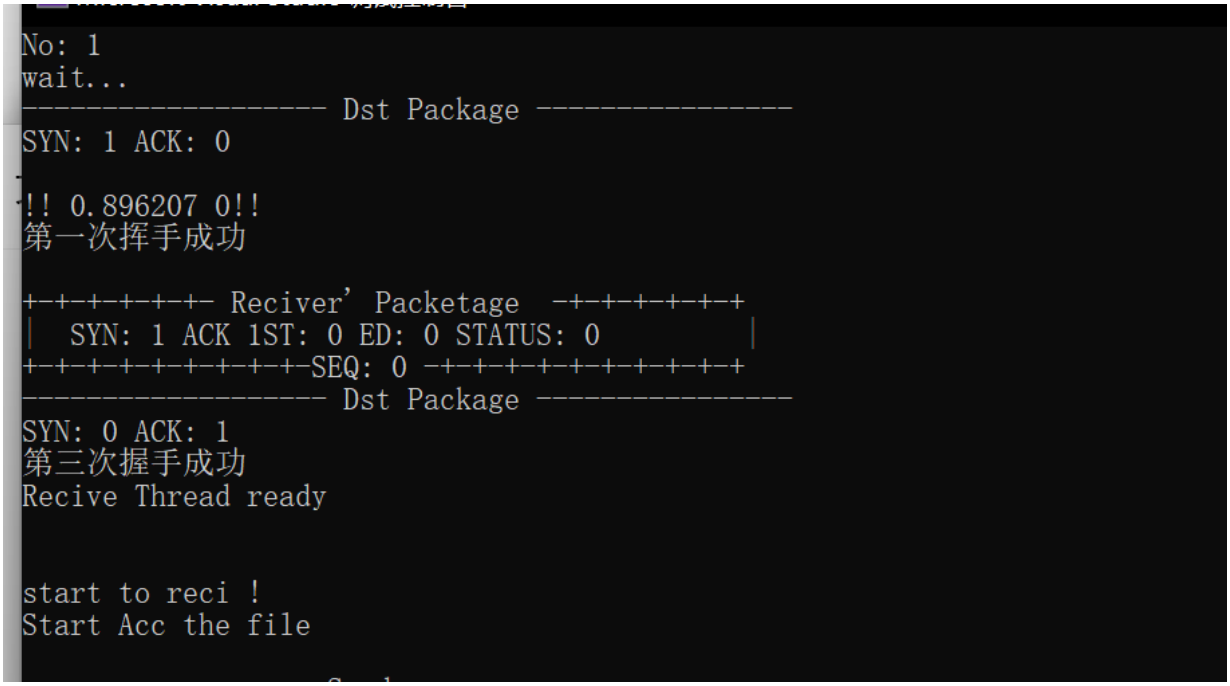
```
Flag: 1, Checksum: 36910, Sequence: 12, Acknowledgment: 112, Data Size: 16384
----- Check correct -----

---------------------------- DROP -----------------------------

------------------------- SEQ: 13 -----------------------------
Flag: 9, Checksum: 12945, Sequence: 13, Acknowledgment: 113, Data Size: 16384

--------------------- Time out  ------------------
Send again: send_status: 1 send_seq: 13

------------------------- SEQ: 13 -----------------------------
Flag: 9, Checksum: 12945, Sequence: 13, Acknowledgment: 113, Data Size: 16384
----- Check correct -----
```

4. 最后，发送最后输出吞吐量等数据，输入q停止传输，输入其他继续传输。

```
----- Check correct -----

------------------------- SEQ: 730 -----------------------------
Flag: 1, Checksum: 34064, Sequence: 730, Acknowledgment: 830, Data Size: 16384
----- Check correct -----

------------------------- SEQ: 731 -----------------------------
Flag: 19, Checksum: 5503, Sequence: 731, Acknowledgment: 831, Data Size: 8674
----- Check correct -----
Sending Over
Total Length: 12111978 bytes
Duration: 13 secs
Speed Rate: 931690 Bps

G:\大三上\计算机网络\code\StableUdp\x64\Debug\StableUdp.exe (进程 11888)已退出，代码为 -1。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制
```

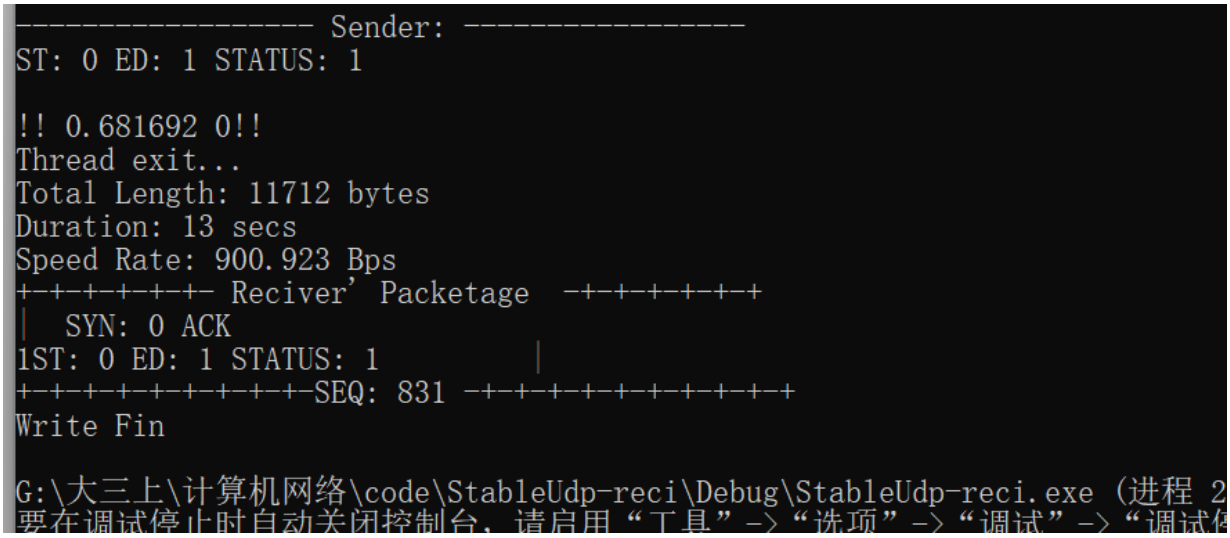
接受端

1. 同样输入wait时，程序等待握手，握手成功后输出Start Acc the file开始接受文件。



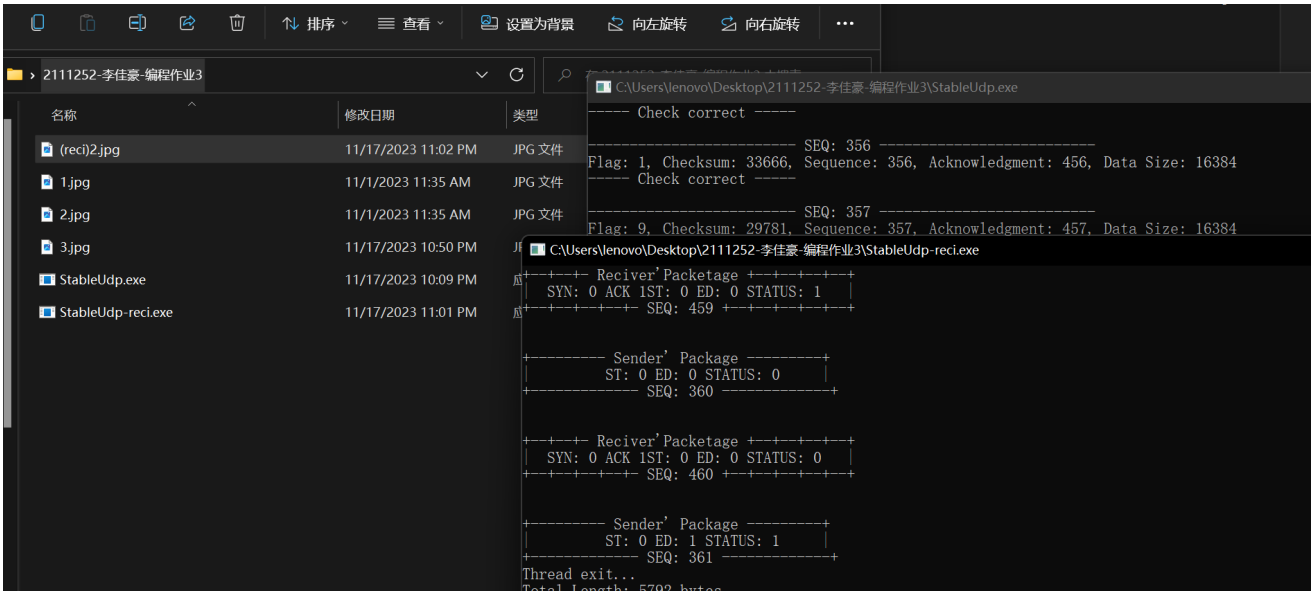
2. 中间Reciver' Packetage表示当前发送的包， Sender:表示接受的包

3. 最后输出吞吐率等信息。输入q关闭端口，否则继续等待握手。



#####

运行结果大概这样 （reci）开头的为结果


排序
查看
设置为背景
向左旋转
向右旋转
2111252-李佳豪-编程作业3
名称
修改日期
类型
(reci)2.jpg
11/17/2023 11:02 PM
JPG 文件
1.jpg
11/1/2023 11:35 AM
JPG 文件
2.jpg
11/1/2023 11:35 AM
JPG 文件
3.jpg
11/17/2023 10:50 PM
StableUdp.exe
11/17/2023 10:09 PM
StableUdp-reci.exe
11/17/2023 11:01 PM
C:\Users\lenovo\Desktop\2111252-李佳豪-编程作业3\StableUdp.exe
----- Check correct -----
---------- SEQ: 356 ----------
Flag: 1, Checksum: 33666, Sequence: 356, Acknowledgment: 456, Data Size: 16384
----- Check correct -----
---------- SEQ: 357 ----------
Flag: 9, Checksum: 29781, Sequence: 357, Acknowledgment: 457, Data Size: 16384
C:\Users\lenovo\Desktop\2111252-李佳豪-编程作业3\StableUdp-reci.exe
+--+--+- Reciver'Packetage +--+--+--+--+
| SYN: 0 ACK 1ST: 0 ED: 0 STATUS: 1 |
+--+--+--+--+- SEQ: 459 +--+--+--+--+--+
+--------- Sender' Package ---------+
| ST: 0 ED: 0 STATUS: 0 |
+---------- SEQ: 360 ----------+
+--+--+- Reciver'Packetage +--+--+--+--+
| SYN: 0 ACK 1ST: 0 ED: 0 STATUS: 0 |
+--+--+--+--+- SEQ: 460 +--+--+--+--+--+
+--------- Sender' Package ---------+
| ST: 0 ED: 1 STATUS: 1 |
+---------- SEQ: 361 ----------+
Thread exit...