# 实验3：基于UDP服务设计可靠传输协议并编程实现

## 协议设计

实现UDP协议下一个发送方对应一个接收方，每次进行一个文件单向传输，

发送方和接收方的主要功能：

发送方--发送文件-->接收方，接收方--确认号-->发送方

### 数据报格式

//每次传送的大小

#define nsize 6000

struct PDU {

uint16\_t source\_port;//源端口

uint16\_t dest\_port;//目的端口

uint32\_t ack = -1;//确认号

uint32\_t seq = 0;//序列号

uint8\_t flags = 0;//标志位 ACK SYN RST FIN

int B\_E = -1;//0开始，1结束

size\_t length = 0;//数据大小，保持文件大小不变

uint16\_t checksum = 0;//16位校验和

char buf[nsize] = { 0 };

};

其中，seq为发送的数据的首个字节序号，接收到ack=n表示对与字节序号为n-1之前的都进行正确的接收

### 建立连接

#### 双方

发送方和接受方采用一样的模式：

##### 初始化： 进行bind过程，绑定ip和端口号

##### 与路由器的连接：在三次握手后调用connect()函数

###### 三次握手时的传输是通过sendto()与recvfrom()函数进行，由于UDP是无连接的传输，需要每次指定ip与port，在不进行connect()的时候，每次传输后都需要连接和断开，

由于文件发送是向同一个端口连续发送，故在上述的三次握手后调用connect(),此后传输过程中使用send(),recv()；在文件传输结束前都不会断开连接

#### 三次握手建立连接

1. 发送方向接收方发送SYN+(seq=0)包，请求建立连接：

如果没有在超时时间内接收到接收端的SYN+ACK+(ack=1)，会进行重传

1. 接收方收到发送方的SYN包之后，向发送方发送SYN+ACK+(ack=1)+(seq=0)包,并期望接收到发送方的ACK+（ack=1）包；

如果没有在超时时间内接收到期望的ACK就会进行重传,如果超过最长等待时间20s说明期望的ACK包应该丢失了，不再等待进入建立连接

如果在限制时间内接收到了，接收方就进入了连接建立阶段

1. 发送方接收到接收方的SYN包之后，就会向接收方发送ACK+(ack=1)+(seq=1)包,进入连接建立阶段

### 数据传输

#### 超时重传

通过setsockopt中的SO\_RCVTIMEO设置超时,初始超时时间为5ms，在循环接收时判断超时(recvret == -1 && GetLastError() == 10060) ,超时就将超时时间更新为原来的2倍，如果收到了数据，就将超时时间更新为原来的1/2，在实验过程中，发现超时时间一般最大到1024ms，故当超时时间减少到1024ms时就不再继续减少。

//设置接收超时,初始化为5ms，设置成功后，recv（）非阻塞

int timeOut = 5;

setsockopt(Socket, SOL\_SOCKET, SO\_RCVTIMEO, (char\*)&timeOut, sizeof(timeOut)

#### 确认接收

##### 对接收方：

设置next\_want\_seq=0//希望接收到的序列号（为上次接收到的序列号+接收数据大小+1）

对于每次有效的recv()，

1. 检验16位校验和，如果相加不为0xffff,就丢弃该包，继续接收
2. 判断接收到的序列号是否等于next\_want\_seq,如果相等就将接收数据写入文件；

并设置要发送的确认包{ACK+(ack=recvpdu.seq+recvpdu.length+1)}；

并更新next\_want\_seq=ack;

如果收到的包中有B\_E=1（文件发送结束标志），跳出循环接收

继续循环接收

1. 判断接收到的序列号是否为上一次接收到的序列号(说明发生了延时或者ACK包丢失的情况)

重发上一次的ACK包，

继续循环接收

##### 对发送方：

对接收到的ACK包中的ack进行判断：

1. 若在限制时间内接收到的ack等于发送出去的seq+length+1,说明收到正确的确认包，设置下一次发送的seq=之前发送出去的seq+length+1; 继续后续数据的发送
2. 若超时没有收到ACK包（发生了延时或者丢包），就重发数据包

### 断开连接

#### 四次挥手断开连接

1. 发送方发送完数据并正确接收到最后的ACK包后，开始断开连接，向接收方发送ACK+FIN包，并期望接收到接收方的ACK包，若超时还未接收到就重发ACK+FIN
2. 接收判断文件接收完毕后，若收到ACK+FIN包，向发送方发送ACK确认包，之后继续发送ACK\_FIN包，并期望得到发送方的ACK包，若超时还未收到就会重发ACK+FIN,若超时时间超过20s,说明期望ACK丢失，不再等待，直接关闭连接，因为数据在开始关闭前已经传送完毕，最后的ACK确认包接收不影响
3. 发送方收到ACK后，判断确认号是否正确，发送方继续收到ACK\_FIN包后，向接收方发送ACK之后等待10秒后就断开连接
4. 接收方收到期望的ACK包后，断开连接

## 核心代码分析

功能函数

16位校验和 计算：

（将结构体中的source\_port,dest\_port,seq,ack,length,buf，flags按每十六位（其中seq，ack分为两个十六位，length，flags零扩展至十六位，buf每两个字节当成一个十六位，多出来的一个字节零扩展至十六位）进行相加，溢出加到结果最低位） 最后结果（取低16位后再取反）放入checksum中

uint16\_t setchecksum(PDU &sendpdu) {

sendpdu.checksum = 0;

//对结构体内进行每16位求和

uint32\_t tem =add( add(add(add(add(sendpdu.source\_port, sendpdu.dest\_port), sendpdu.ack), sendpdu.seq),sendpdu.flags),sendpdu.length);

int i = 0;

for (int i = 0; i < 6000; i+=2) {

uint16\_t t = 0 | sendpdu.buf[i] | sendpdu.buf[i + 1]<<8;

tem = add(tem, t);

}

while (i < 6000) {

tem = add(tem,sendpdu.buf[i]);

i++;

}

sendpdu.checksum =( ~tem)&0xffff;

return sendpdu.checksum;

}

检验：

//在计算的基础上加上checksum，溢出加到最低位，正确的检验结果应该为0xffff

int checkchecksum(PDU &recvpdu) {

uint32\_t tem = add(add(add(add(add(recvpdu.source\_port, recvpdu.dest\_port), recvpdu.ack), recvpdu.seq), recvpdu.flags),recvpdu.length);

int i = 0;

for (int i = 0; i < 6000; i += 2) {

uint16\_t t = 0 | recvpdu.buf[i] | recvpdu.buf[i + 1] << 8;

tem = add(tem, t);

}

while (i < 6000) {

tem = add(tem, recvpdu.buf[i]);

i++;

}

tem =tem+ recvpdu.checksum;

if (tem == 0xffff) {return 1; }

else { return 0; }

}

1. 超时重传 由于设置setsocketopt中接收超时，故recv()非阻塞，需要循环接收

int timeout\_retry() {

while (1) {

//发送错误

while (sendret < 0) {

cout << "send error" << GetLastError() << endl;

sendret = send();

cout << "======================resend=======================" << endl;

pdu\_to\_str(sendpdu);

}

recvret = recv();

cout << "======================recv=======================" << endl;

pdu\_to\_str(recvpdu);

//接收成功，减少接收时间

if (recvret > 0) {

cout << "================recv============================:" << endl;

pdu\_to\_str(recvpdu);

lessouttime();

cout << "success recv ,so less timeout" << timeOut << endl;

break;

}

//超时，增加接收时间

else if (recvret == -1 && GetLastError() == 10060) {

sendret = send();

cout << "======================resend=======================" << endl;

pdu\_to\_str(sendpdu);

updateouttime();

cout << "timeout happen,so more timeout" << timeOut << endl;

if (timeOut > 20000) {

//时间太长了,不继续等待了

return 0;

}

}

else {

cout << "other recv error" << GetLastError() << endl;

return 0;

}

}

return 1;

}

1. 述描述的三次握手建立连接一致

//发送方

bool establish\_conn() {

//第一次握手

sendpdu.flags = SYN;

sendpdu.seq = 0;

sendret = sendbefore();

if (sendret < 0) {

cout << "send error" << GetLastError() << endl;

return false;

}

//第二次握手

//超时重传

timeout\_retry\_before();

cout << "======================send=======================" << endl;

pdu\_to\_str(sendpdu);

while (!((recvpdu.ack == sendpdu.seq + 1))) {

sendret = sendbefore();

timeout\_retry\_before();

}

{

cout << "SECOND SHAKE HAND SUCCEED！！" << endl;

cout << "======================recv=======================" << endl;

pdu\_to\_str(recvpdu);

}

//第三次握手

sendpdu.flags = ACK;

sendpdu.seq++;

sendpdu.ack = recvpdu.seq + 1;

sendret = sendbefore();

cout << "======================send=======================" << endl;

pdu\_to\_str(sendpdu);

if (sendret > 0) {

cout << "client\_conn\_established";

return true;

}

else cout << "conn\_send\_error" << endl;

return false;

}

//接收方

bool establish\_conn() {

while (1) {

recvret = recvbefore();

if (recvbefore() > 0)break;

else cout << "waiting for client request......." << " "<<recvret<<endl;

}

sendpdu.dest\_port = ntohs(RemoteAddr.sin\_port);

cout << "======================recv=======================" << endl;

pdu\_to\_str(recvpdu);

if (recvpdu.flags == SYN) {

cout << "FIRST SHACK HANDS SUCCEED!!" << endl;

sendpdu.flags = ACK\_SYN;

sendpdu.seq = 0;

sendpdu.ack = recvpdu.seq + 1;

sendret = sendbefore();

//超时重传

timeout\_retry\_before();

//第三次握手

while (!(recvpdu.ack == sendpdu.seq + 1 && recvpdu.flags == ACK)) {

sendret = sendbefore();

if (timeout\_retry\_before() == 0) {

cout << "THRID SHACK HAND NOT ARRVE!!!no try anymore,establish conn" << endl;

return true;

}

}

{

cout << "THRID SHACK HAND SUCCEED!!!" << endl;

cout << "======================recv=======================" << endl;

pdu\_to\_str(recvpdu);

cout << "server\_conn\_established"; return true;

}

return false;

}

return false;

}

1. 释放连接,与上述描述的四次挥手释放连接一致

//发送方

void close\_conn() {

sendpdu.flags = ACK\_FIN;

sendpdu.seq = 0;

//超时重传

sendret = send();

cout << "======================send=========================" << endl;

pdu\_to\_str(sendpdu);

cout << "FIRST BYE，go state: FIN\_WAIT1" << endl;

timeout\_retry();

cout << "go state FIN\_WAIT2" << endl;

while (!(recvpdu.ack == sendpdu.seq + 1 && recvpdu.flags == ACK\_FIN)) {

Sleep(10000);

timeout\_retry();

}

sendpdu.flags = ACK;

sendpdu.ack = recvpdu.seq + 1;

//因为最后一次挥手可能丢包，故多发了几次

sendret = send();

sendret = send();

sendret = send();

cout << "======================send=========================" << endl;

pdu\_to\_str(sendpdu);

cout << "THIRD BYE ,go state:TIME\_WAIT" << endl;

Sleep(10000);//等待10秒

closesocket(Socket);

}

//接收方

bool close\_conn() {

//函数前收到了对端的ACK\_FIN

sendpdu.flags = ACK;

sendpdu.ack = recvpdu.seq + 1;

cout << "SECOND BYE,go to state CLOSE\_WAIT" << endl;

sendret = send();

//已经接收完毕,发送ACK+FIN

cout << "======================send=========================" << endl;

pdu\_to\_str(sendpdu);

sendpdu.flags = ACK\_FIN;

sendpdu.seq = 0;

sendret = send();

cout << "======================send=========================" << endl;

pdu\_to\_str(sendpdu);

cout << "THIRD BYE,go to state LAST\_ACK" << endl;

//超时重传

if (timeout\_retry() == 0) {

cout << "FOUR BYES NOT ARRIVED but not wait anymore,close connect!!" << endl;

closesocket(Socket);

return true;

}

if (check\_ack\_seq(sendpdu, recvpdu)) {

cout << "FOUR BYES SUCCEED!!" << endl;

closesocket(Socket);

return true;

}

return false;

}

1. Main函数

//发送方：循环发送

while (!fin.eof()) {

//接着读取文件

memset(sendpdu.buf, 0, nsize);

fin.read(sendpdu.buf, nsize);//设置发送文件到sendbuf

//判断是否是第一次读取（文件头）

sendpdu.length = fin.gcount();

st++;

sendpdu.flags = ACK;//设置发送ACK

//设置开始结束标志

if (st == 1) { sendpdu.B\_E = 0; sendpdu.seq = 0; }//第一次读取，设置开始标志和seq

else { sendpdu.B\_E = -1; }//不是第一次读取

if (fin.eof()) { sendpdu.B\_E = 1; }//最后一次读取，设置结束标志

sendpdu.checksum = setchecksum(sendpdu);

sendret = send();//发送

if (sendret < 0) {

cout << "send error" << GetLastError() << endl;

return 1;

}

else {

cout << "======================send=======================" << endl;

pdu\_to\_str(sendpdu);

}

//判断是否收到ACK包超时重发

timeout\_retry();

//收到srv的ACK进行检查

cout << "======================recv============================:" << endl;

//最后一个包已经发送完成，且得到最后的ACK包

if (sendpdu.B\_E == 1 && ((sendpdu.seq + sendpdu.length + 1) == recvpdu.ack) && recvpdu.flags == ACK) {

break;

}

//可能丢包重发

while (!((sendpdu.seq + sendpdu.length + 1) == recvpdu.ack && recvpdu.flags == ACK)) {

cout << "ack check failed,may loss or delay ,will resend" << endl;

if ((sendpdu.seq + sendpdu.length + 1) != recvpdu.ack) {

cout << "ack check failed" << sendpdu.seq << " " << sendpdu.length + 1 << " " << recvpdu.ack << endl;

}

if (recvpdu.flags != ACK) { cout << "not ack packet：" << recvpdu.flags << endl; }

sendret = send(); //重发

cout << "======================resend==========================:" << endl;

pdu\_to\_str(sendpdu);

timeout\_retry();

}

//接收到正确的确认包后才继续

sendpdu.seq += sendpdu.length + 1;

}

//接收方：循环接收

while (1) {

if (recvret=recv()> 0) {

cout << "=========recv==============================" << endl;

pdu\_to\_str(recvpdu);

//确认后将接收到的数据写入本地文件

if (checkchecksum(recvpdu) == 0) { cout << "check faild" << endl; }

if (recvpdu.seq == next\_want\_seq)

{

outf.write(recvpdu.buf, recvpdu.length);

}

else if (recvpdu.seq == next\_want\_seq - recvpdu.length - 1) {

//是ACK确认包丢失导致的重传

cout << "ACK packet loss or delay: will resend ACK packet" << endl;

sendret = send();//发送ACK包

cout << "======================resend======================" << endl;

pdu\_to\_str(sendpdu);

continue;

}

else {

//接收序列号不对，丢弃并继续接收

cout << "recv seq:" << recvpdu.seq << " but EXPECT recvseq：" << next\_want\_seq << endl;

continue;

}

//接收完成

sendpdu.ack = recvpdu.seq + recvpdu.length + 1;//设置发送ack.下一个希望接受的seq

next\_want\_seq = sendpdu.ack;//设置下一个期望的recv的seq

sendpdu.flags = ACK;//设置ACK

sendret = send();//发送ACK包

cout << "======================send========================="<< endl;

pdu\_to\_str(sendpdu);

if (recvpdu.B\_E == 1) { cout << "RECV finished!!!" << endl; recvpdu.length = 0; break; }

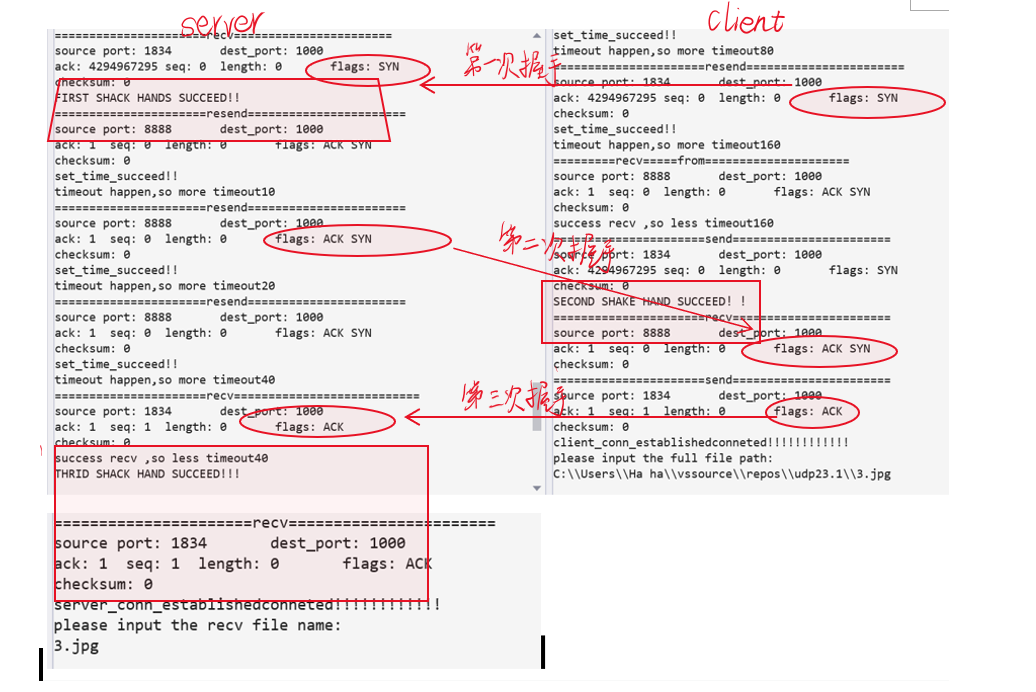
}

}

## 运行截图以及分析

1.设置时延30ms,丢包率20%,传输3.jpg,运行如下：

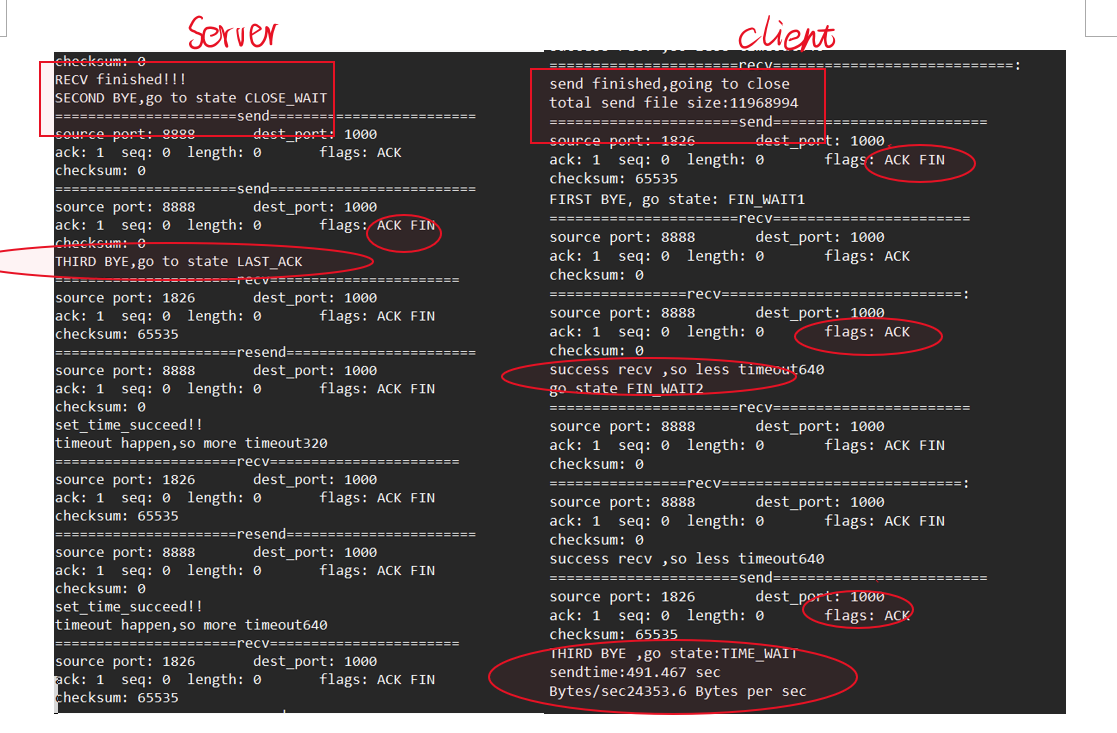
需要先输入绑定的端口号，之后进入建立连接阶段

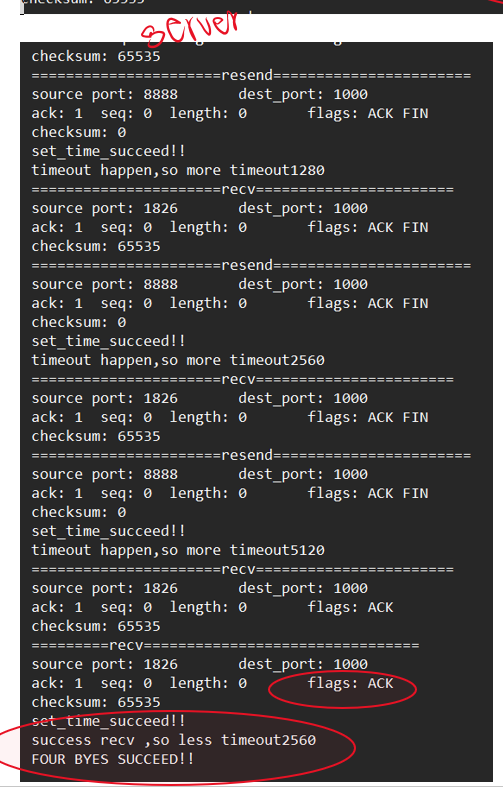


上图显示了连接建立过程，三次握手

通过重定向将文件传输的send，recv日志情况输出到了文件中

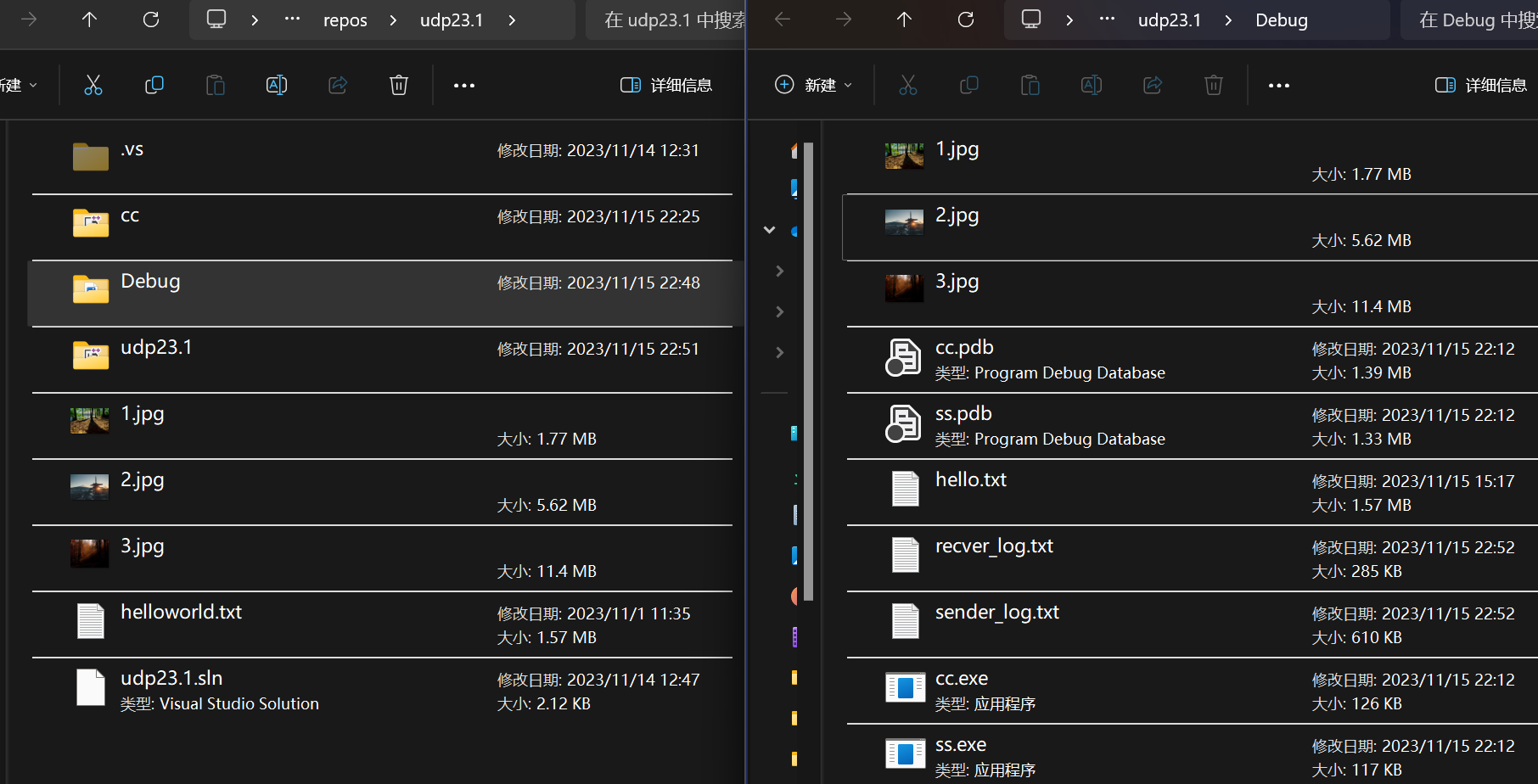
传输结果：



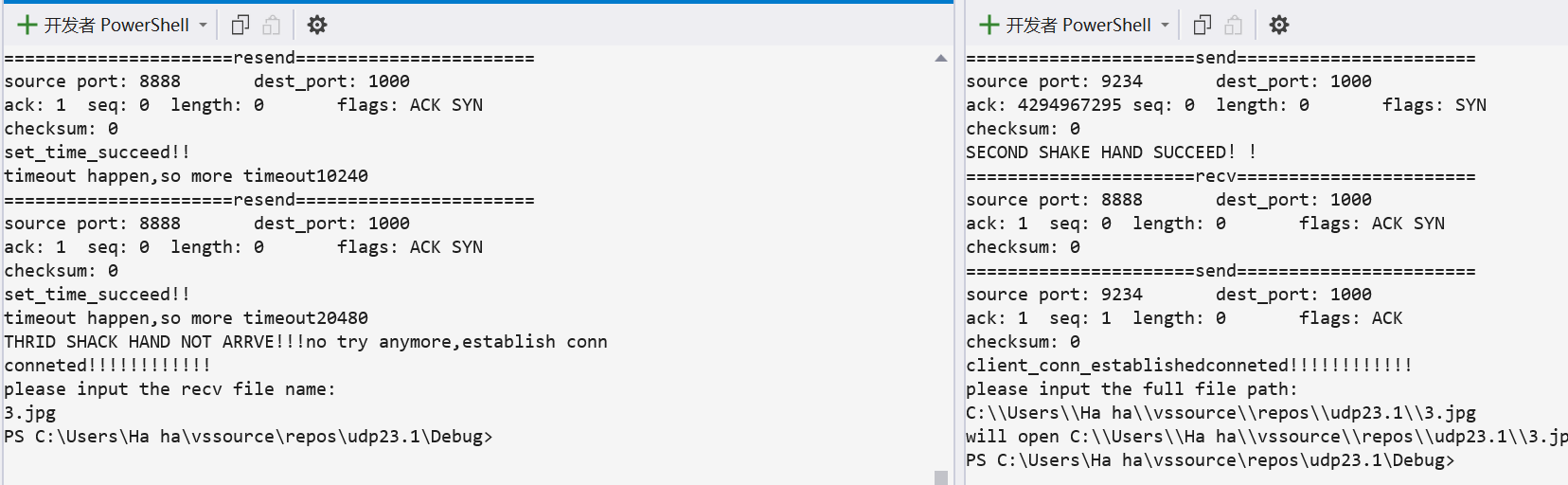


可以体现四次挥手的过程，也体现了超时重传的过程

三个测试文件都能传输成功，且文件大小不变

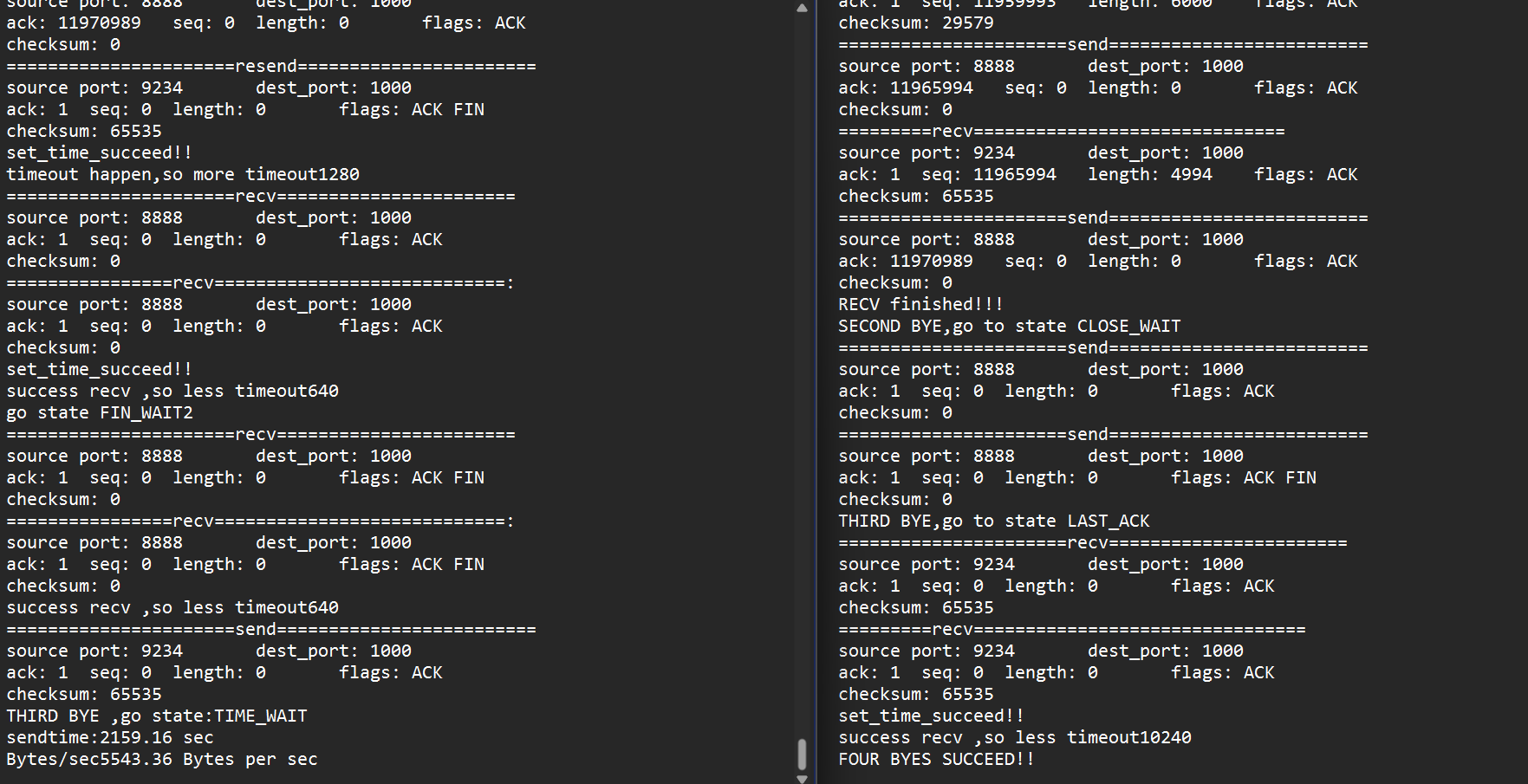


2.设置时延300ms,丢包率50%,传输3.jpg,最终传输成功：



上图即为三次握手最后一次ACK包丢失，超过最大延时时间不再等待，直接进入建立连接

最后四次挥手：



## 实验中的问题

当延时的时候，由于多个因为延时而再次发送的数据包在之后抵达，发送方接收到后就会重发ACK包，造成了一定的浪费

对于三次握手和四次挥手的最后一次ACK确认包的到达并不确认，如果超过设定的最大超时时间，就会直接建立连接或者关闭连接