计算机网络课程实验报告

郭裕彬 2114052 物联网工程

实验3-1：基于UDP服务设计可靠传输协议并编程实现

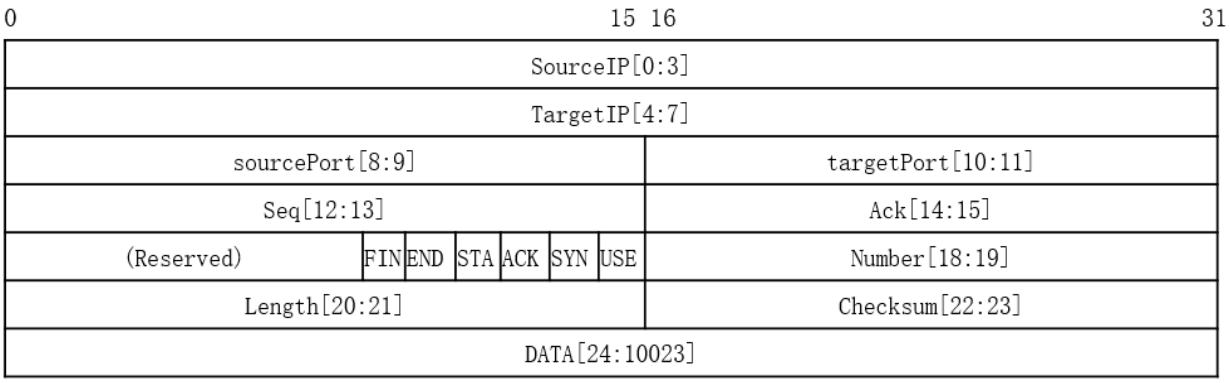
发送端使用地址：127.0.0.1:9995
接收端使用地址：127.0.0.1:9997
路由器使用地址：127.0.0.1:9996

实验要求：

- 利用数据报套接字在用户空间实现面向连接的可靠数 据传输，功能包括：建立连接、差错检测、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输。
- 数据报套接字：UDP
- 协议设计：数据包格式，发送端和接收端交互，详细完整
- 建立连接、断开连接：类似TCP的握手、挥手功能
- 差错检验：校验和
- 接收确认、超时重传：rdt2.0、rdt2.1、rdt2.2、rtd3.0等，亦可自行设计协议
- 单向传输：发送端、接收端
- 日志输出：收到/发送数据包的序号、ACK、校验和等，传输时间与吞吐率
- 测试文件：必须使用助教发的测试文件（1.jpg、2.jpg、3.jpg、helloworld.txt）

协议设计

整体结构



- 协议报文头部共24字节：
    - SourceIP：32位源IP地址
    - TargetIP：32位目的IP地址
    - SourcePort：16位源端口号
    - TargetPort：16位目的端口号
    - Seq：16位序列号，取值为0或1
    - Ack：16位确认序列号，取值为0或1。（本实验握手部分的ACK包的Ack值设置为对应包的Seq+1；其他部分的ACK包的Ack值都为对应包的Seq）
    - FLAGS
        - USE：表示该报文有效
        - SYN：表示为连接请求建立报文
        - ACK：表示为确认报文
        - STA：表示为文件传输开始报文
        - END：表示为文件传输结束报文
        - FIN：表示为连接请求断开报文
    - Number：16位数据，用于在STA报文中记录文件分包总数
    - Length：16位数据，用于标识该包的数据段长度
    - Checksum：16位校验和
- 该协议在具体实现中使用自定义类结构`UDP_packet`装载

```
class UDP_packet{
public:
    u_int sourceIP;
```

```
    u_int targetIP;
    u_short sourcePort;
    u_short targetPort;
    u_short Seq;
    u_short Ack;
    u_short flags;
    u_short number;
    u_short length;
    u_short checksum;
    char data[MAX_SIZE];
    ···some function···
}
```

双端交互

- 发送端发起连接请求，与接收端实现类似TCP的三次握手，发送端进入等待发送状态，接收端进入等待接收状态
- 发送端发送STA报文，接收端收到该报文，进入接收状态，返回ACK并记录报文中的Number字段用于计包数；
- 发送端收到ACK，进入发送状态，发送一个数据包后进入停等状态
- 接收端收到数据包后返回ACK，继续处于接收状态；接收端收到ACK后进入发送阶段，重复上一步直至发送完最后一个文件数据包（END）
- 接收端收到END包后返回ACK，进入等待接收状态，等待继续接收文件或断开连接
- 发送端收到END包的ACK后，进入等待发送状态，等待用户选择继续发送文件或断开连接
- 用户选择断开连接，发送端发起断开连接请求，与接收端实现类似TCP的四次挥手，完成后双端关闭

建立、断开连接

- 类似TCP的三次握手：发送端发起一次握手，接收二次握手，发送三次握手；接收端接收一次握手，发送二次握手，接收三次握手建立连接，使用超时重传机制。

```
[Info]发送请求包，开始第一次握手
[Send] ACK=0 SEQ=0 CHECKSUM=e5b0 LEN=0 [SYN]
[Recv] ACK=1 SEQ=0 CHECKSUM=e3ab LEN=0 [SYN][ACK]
[Info]收到确认包，开始第三次握手
[Send] ACK=1 SEQ=1 CHECKSUM=e5b0 LEN=0 [ACK]
[Info]--------三次握手完成--------
```

- 第一次握手：SYN=1，Seq=0，Ack=0
- 第二次握手：SYN=1，ACK=1，Seq=0，Ack=0
- 第三次握手：ACK=1，Seq=1，Ack=1

```
//发送端
bool makeConnection(SOCKADDR_IN s,SOCKADDR_IN t,SOCKET
client){
    //握手建立连接
    //第一次握手，Seq恒为0
    UDP_packet send,receive;

 send.setSourceIP(s);send.setTargetIP(t);send.setFlags(FLAG_
SYN);
    send.setSeq(clientseq);
    flushSeq();
    send.setFlags(FLAG_USE);
    send.calChecksum();
    sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
    cout<<"[Info]发送请求包，开始第一次握手"<<endl;
    printf("[Send] ");send.printInfo();
    int len=sizeof(t);
    //超时重传
    time_t start=clock();
    time_t end;
    while(1)
    {
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&t,&len);
        //第二次握手包Ack=1,Seq=0
        if(receive.flags&
(FLAG_USE)&&receive.Ack==send.Seq+1)
        {
```

```
            printf("[Recv] ");receive.printInfo();
            cout<<"[Info]收到确认包，开始第三次握手"<<endl;
            //第三次握手包Ack=1,Seq=1
            send.setAck(receive.Seq+1);
            send.setSeq(clientseq);
            flushSeq();
            send.setFlags(0);
            send.setFlags(FLAG_USE);
            send.setFlags(FLAG_ACK);
            sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
            printf("[Send] ");send.printInfo();
            cout<<"[Info]--------三次握手完成--------"<<endl;
            return 1;
        }
        //超时重传
        end=clock();
        if(((end-start)/CLOCKS_PER_SEC)>=2){
            start=clock();
            sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
            cout<<"[Info]未收到第二次握手包，尝试重发......"
<<endl;
        }
    }
}

//接收端
bool makeConnection(SOCKET client,UDP_packet
receive,SOCKADDR_IN s,SOCKADDR_IN t){
    //握手建立连接
    //第二次握手Seq为0,Ack为0+1=1
    UDP_packet send;
    send.setSourceIP(s);send.setTargetIP(t);
    send.setFlags(FLAG_SYN);
    send.setSeq(0);
    send.setFlags(FLAG_USE);
    send.setFlags(FLAG_ACK);
    send.setAck(receive.Seq+1);
```

```
    send.calChecksum();
    sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
    cout<<"[Info]--------开始握手--------"<<endl;
    printf("[Recv] ");receive.printInfo();
    cout<<"[Info]收到请求包，开始第二次握手"<<endl;
    printf("[Send] ");send.printInfo();
    int len=sizeof(t);
    //超时重传
    time_t start=clock();
    time_t end;
    while(1)
    {
        //接收第三次握手包，Ack=0+1=1,Seq=1
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&t,&len);
        if(receive.flags&
(FLAG_ACK)&&receive.Ack==send.Seq+1)
        {
            cout<<"[Info]收到确认包"<<endl;
            printf("[Recv] ");receive.printInfo();
            cout<<"[Info]--------三次握手完成--------"<<endl;
            return 1;
        }
        //超时重传
        end=clock();
        if(((end-start)/CLOCKS_PER_SEC)>=2){
            start=clock();
            sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
            cout<<"[Info]未收到第三次握手包，尝试重发......"
<<endl;
        }
    }
}
```

- 类似TCP的四次挥手：发送端发起一次挥手，接收二次挥手，接收三次挥手，发送四次挥手；接收端接收一次挥手，发送二次、三次挥手，接收四次挥手，使用超时重传机制。

```
[Info]发送端尝试断开连接
[Info]--------开始挥手--------
[Info]收到第一次挥手包
[Recv] ACK=0 SEQ=1 CHECKSUM=e591 LEN=0 [FIN]
[Info]发送第二次挥手包
[Send] ACK=1 SEQ=1 CHECKSUM=e3ac LEN=0 [ACK]
[Info]发送第三次挥手包
[Send] ACK=0 SEQ=0 CHECKSUM=e392 LEN=0 [FIN]
[Info]收到第四次挥手包
[Recv] ACK=0 SEQ=1 CHECKSUM=e591 LEN=0 [FIN]
[Info]---------四次挥手完成---------
```

  - 第一次挥手：FIN=1，Seq=x，Ack=0
  - 第二次挥手：ACK=1，Seq=x，Ack=x
  - 第三次挥手：FIN=1，Seq=~x，ACK=0
  - 第四次挥手：ACK=1，Seq=~x，ACK=~x

```
//发送端
void endConnection(SOCKADDR_IN s,SOCKADDR_IN t,SOCKET
client){
    //四次挥手断开连接
    UDP_packet send,receive;
    send.setSourceIP(s);send.setTargetIP(t);
    send.setFlags(FLAG_USE);
    send.setFlags(FLAG_FIN);
    flushSeq();
    cout<<"[Info]发送第一次挥手包"<<endl;
    //发送第一次挥手，并停等第二次挥手ACK包
    stopWaitSend(send,receive,client,s,t);
    UDP_packet send2,receive2;
    int len=sizeof(t);
    //接收第三次挥手的FIN包
    recvfrom(client,(char*)&receive2,sizeof(receive2),0,
(SOCKADDR*)&t,&len);
    if(receive2.flags&FLAG_FIN){
        cout<<"[Info]收到第三次挥手包"<<endl;
        printf("[Recv] ");receive2.printInfo();
```

```
        cout<<"[Info]发送第四次挥手包"<<endl;
        //发送第四次挥手ACK包
        send2.setFlags(FLAG_USE);
        send2.setFlags(FLAG_ACK);
        send2.setAck(receive2.Seq);
        send2.setSeq(clientseq);
        flushSeq();
        sendto(client,(char*)&send2,sizeof(send2),0,
(SOCKADDR*)&t,sizeof(t));
        printf("[Send] ");send2.printInfo();
        cout<<"[Info]---------断开链接--------"<<endl;
        return;
    }
}

//接收端
void endConnection(SOCKET client,UDP_packet
receive,SOCKADDR_IN s,SOCKADDR_IN t){
    //四次挥手断开连接
    UDP_packet send,send2,receive2;
    //发送第二次挥手ACK包
    send.setSourceIP(s);send.setTargetIP(t);
    send.setFlags(FLAG_ACK);send.setFlags(FLAG_USE);
    send.setAck(receive.Seq);clientack=flushSeq();
    send.setSeq(clientack);
    send.calChecksum();
    cout<<"[Info]--------开始挥手--------"<<endl;
    cout<<"[Info]收到第一次挥手包"<<endl;
    printf("[Recv] ");receive.printInfo();
    cout<<"[Info]发送第二次挥手包"<<endl;
    printf("[Send] ");send.printInfo();
    sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
    int len=sizeof(t);
    time_t start=clock();
    time_t end;
    //发送第三次挥手FIN包
    send2.setSourceIP(s);send2.setTargetIP(t);
```

```
 send2.setFlags(FLAG_USE);send2.setFlags(FLAG_FIN);clientack
=flushSeq();
    send2.setSeq(clientack);
    send2.calChecksum();
    while(1)
    {
        sendto(client,(char*)&send2,sizeof(send2),0,
(SOCKADDR*)&t,sizeof(t));
        cout<<"[Info]发送第三次挥手包"<<endl;
        printf("[Send] ");send2.printInfo();
        recvfrom(client,(char*)&receive2,sizeof(receive2),0,
(SOCKADDR*)&t,&len);
        //收到第四次挥手ACK包
        if(receive.flags&(FLAG_USE)&&receive2.Ack!=send.Seq)
        {
            cout<<"[Info]收到第四次挥手包"<<endl;
            printf("[Recv] ");receive2.printInfo();
            cout<<"[Info]---------四次挥手完成---------"
<<endl;
            return;
        }
        //收到的包不是对应的ACK包
        else if(receive.flags&
(FLAG_USE)&&receive2.Ack==send.Seq)
        {
            cout<<"[Info]发送端未收到第二次挥手报文，重发......"
<<endl;
            sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
        }
        //超时重传
        end=clock();
        if(((end-start)/CLOCKS_PER_SEC)>=2){
            cout<<"[Info]未收到ACK确认报文，尝试重发......"
<<endl;
            start=clock();
            sendto(client,(char*)&send2,sizeof(send2),0,
(SOCKADDR*)&t,sizeof(t));
```

```
        }
    }
}
```

差错检验

- 校验和计算`UDP_packet::calChecksum()`
    - 以16bit为单位对首部数据进行累加求和
    - 如果累加和超过16bit，产生了进位，需将高16bit和低16bit累加求和
    - 循环上一步，直至未产生进位为止
    - 累加和取反得到校验和
- 校验和校验`UDP_packet::checkChecksum()`
    - 以16bit为单位进行累加求和
    - 如果累加和超过16bit，产生了进位，需将高16bit和低16bit累加求和
    - 循环步骤2，直至未产生进位为止
    - 累加和和校验和相加得到0xffff，校验成功，否则失败

```
    void calChecksum(){
        //计算校验和
        u_int sum=0;
        u_short *pointer=(u_short*)this;
        //对首部字段按16位求和
        for(int i=0;i<10;i++)
        {
            sum+=pointer[i];
        }
        //超出16位则循环相加
        while(sum&0xffff0000)
        sum=sum>>16+sum&0xffff;
        //取反
        checksum=(u_short)~sum;
    }
    bool checkChecksum(){
```

```
        //检验校验和
        u_int sum=0;
        u_short *pointer=(u_short*)this;
        //对首部字段按16位求和
        for(int i=0;i<10;i++)
        {
            sum+=pointer[i];
        }
        //超出16位则循环相加
        while(sum&0xffff0000)
        sum=sum>>16+sum&0xffff;
        //累加和和校验和相加为全1则校验通过
        if((u_short)sum+checksum==0xffff)
            return true;
        else return false;
    }
```

接收确认，超时重传

发送端每发出一个数据包就需要等待接收方传回ACK确认包后才能继续行为；设置一个定时器，在设定的时间内未收到ACK确认包则需要重新传送原本的数据包。

- 发送端的`stopWaitSend`函数负责实现停等机制中的发送部分。

  发送数据包的同时开始计时等待，若在等待时间内收到对应的ACK确认包，则正常返回；若等待时间内未收到包，则尝试在规定的重发次数内重发该包，重置计时器，循环直至收到ACK或超过规定的次数，若是后者则返回错误值

```
bool stopWaitSend(UDP_packet& send,UDP_packet&
receive,SOCKET client,SOCKADDR_IN s,SOCKADDR_IN t){
    //停等机制发送端的部分，实现接受确认、超时重传
    send.setSeq(clientseq);
    flushSeq();
    send.setFlags(1);
    send.calChecksum();
    sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
    printf("[Send] ");send.printInfo();
    int len=sizeof(t);
    //设置计时器以超时重传
```

```
    time_t start=clock();
    time_t end;
    int count=1;
    while(1)
    {
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&t,&len);
        //接收确认部分
        if(receive.flags&(FLAG_USE)&&receive.Ack==send.Seq)
        {
            printf("[Recv] ");receive.printInfo();
            return 1;
        }
        end=clock();
        //超时重传部分
        if(((end-start)/CLOCKS_PER_SEC)>=2){
            if(count == 6){
                cout<<"[Error]重发超过次数"<<endl;
                system("pause");
                return 0;
            }
            cout<<"[Info]未收到ACK确认报文，尝试第"<<count<<"次重
发......"<<endl;
            count++;
            start=clock();
            sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
        }
    }
}
```

- 接收端的`stopWaitReceive`函数负责实现停等机制中的接收部分。

  接收到发送端发来的数据包时，首先根据SEQ判断该包是上一次接收到的重复的包还是新的包，若是重复的包则重发之前的ACK报文，以纠正发送端没有收到该ACK的异常情况；若是新的包，进一步判断该包的校验和校验结果，若包数据异常则不做任何处理，继续循环接收发送端发送的数据包；若校验和校验结果通过，则发送对该数据包的ACK报文，正常返回。

```
```

```
bool stopWaitReceive(UDP_packet &send,UDP_packet
&receive,SOCKADDR_IN s,SOCKADDR_IN t,SOCKET client){
    //停等机制接收端的部分，实现接受确认、超时重传
    while(1){
        int len=sizeof(SOCKADDR_IN);
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&t,&len);
        if(receive.flags&FLAG_USE){
            //收到的包不是新的包，重发之前的ACK报文
            if(receive.Seq==clientack){
                cout<<"[Info]收到重复的包"<<endl;
                printf("[Error]" );receive.printInfo();
                send.setAck(flushSeq());
                send.calChecksum();
                sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
            }
            else if(receive.checkChecksum()){
                //收到新的包，且通过了校验和校验
                send.calChecksum();
                clientack=receive.Seq;
                send.setAck(clientack);
                send.calChecksum();
                sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
                return 1;
            }
            //收到新的包数据不完整，继续接收
            else cout<<"[Info]校验和错误报文"<<endl;
        }
    }
}
```

单向传输

- 发送端维护一个全局二维数组用作缓冲区，其中第二维容量的最大值不超过路由器单次能转发的包大小减去设计的包首部后的大小，本次设计通过`#define MAX_SIZE 10000`规定数据包的数据部分`UDP_packet::data`最大为10000字节。

- 接收端同样维护一个全局二维数组用作接收数据包后的缓冲区，原型设计与发送端一致。
- 发送端要发送文件时，首先发送一个标志位STA置位的数据包，首部的`UDP_packet::Number`字段的值为接下来要发送的带有数据的包总数-1，接收端收到后接收接下来的有效的`Number`个包；首部的`UDP_packet::data`字段存储了发送端上传的文件的原始路径，接收端可通过`UDP_packet::Length`字段计数完整读取。
- 接收端收到每一个数据包，都需要检验校验和、返回ACK，接收到标志位END置位的数据包时，认为收到最后一个数据包，按照最后的数据包进行处理，只接受该包中`Length`字段个数据，之后结束接收循环。询问用户输入要保存的文件名，从缓冲区中依次读出数据保存到对应文件中。

```
//part of sendFile()
UDP_packet send,receive;
    //发送STA包，传递文件路径、包总数
    send.setNumber(amount);
    send.setSourceIP(s);
    send.setTargetIP(t);
    send.setFlags(FLAG_STA);
    send.setLength(strlen(filePath));
    memcpy(send.data,filePath,strlen(filePath));
    time_t startsend=clock();
    time_t endsend;
    cout<<"[Info]---------开始传输文件---------"<<endl;
    if(stopWaitSend(send,receive,client,s,t)==0){
        cout<<"[Info]开始文件传输失败"<<endl;
        return;
    }
    cout<<"[Info]起始传输报文发送成功"<<endl;
    //发送带有数据的数据包，共amount+1个
    for(int i=0;i<=amount;i++){
        UDP_packet pk;
        pk.setSourceIP(s);
        pk.setTargetIP(t);
        pk.setNumber(amount);
        pk.setFlags(FLAG_USE);
        if(i==amount){
            //最后一个数据包
```

```
            pk.setFlags(FLAG_END);
            pk.setLength(p);
            for(int j=0;j<p;j++)
            {
                pk.data[j]=sendbuf[i][j];
            }
        }
        else{
            pk.setLength(MAX_SIZE);
            for(int j=0;j<MAX_SIZE;j++)
            {
                pk.data[j]=sendbuf[i][j];
            }
        }
        stopWaitSend(pk,receive,client,s,t);
        if(pk.flags&FLAG_END){
            cout<<"[Info]结束报文发送成功"<<endl;
        }
    }
    endsend=clock();

//part of recvFile()
void recvFile(SOCKET client,UDP_packet& pk,SOCKADDR_IN
s,SOCKADDR_IN t){
    //接收并保存文件的部分
    for(int i=0;i<MAX_SIZE;i++){
        memset(receivebuf[i],0,MAX_SIZE);
    }
    //收到STA包后返回ACK应答
    UDP_packet answer;
    u_short amount,len;
    answer.setAck(pk.Seq);answer.setSeq(clientack);
    answer.setSourceIP(s);answer.setTargetIP(t);
    answer.setFlags(FLAG_USE);answer.setFlags(FLAG_ACK);
    sendto(client,(char*)&answer,sizeof(answer),0,
(SOCKADDR*)&t,sizeof(SOCKADDR));
    amount=pk.number;
    len=pk.length;
    cout<<"[Info]收到开始传输报文"<<endl;
```

```
    //解析STA包
    printf("[Recv] ");pk.printInfo();
    cout<<"上传文件的路径是："<<endl;
    for(int i=0;i<len;i++){
        char a=(char)pk.data[i];
        cout<<a;
    }
    cout<<endl;
    //接收接下来的amount+1个不同的有效数据包到缓冲区，并返回对应ACK应答
    for(int i=0;i<=amount;i++){
        UDP_packet send,receive;
        send.setFlags(FLAG_USE);send.setFlags(FLAG_ACK);
        stopWaitReceive(send,receive,s,t,client);
        if(i==amount){
            len=receive.length;
            for(int j=0;j<len;j++)receivebuf[i][j]=receive.data[j];
        }
        else{
            for(int j=0;j<MAX_SIZE;j++)receivebuf[i]
[j]=receive.data[j];
        }
        printf("[Info]收到第%d个数据包\n",i);
        printf("[Recv] ");receive.printInfo();
        if(receive.flags&FLAG_END){
            cout<<"[Info]该数据包为最后一个包"<<endl;
            cout<<"[Info]--------文件接受完成--------"<<endl;
        }
    }
    //获取用户输入的文件名，并从缓冲区保存到磁盘中
    ····some code····
    return;
}
```

问题及解决

针对检查时提出的NACK报文反馈校验和错误的情况， 增加了NACK标志位，接收端收到未通过校验和验证的报文后，会发送NACK报文；发送端在收到NACK后会直接重发之前的报文，而不是等待超时后再重传。

```
#define FLAG_NACK 1<<6
//part of stopWatiSend()
while(1)
{
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&t,&len);
        //接收确认部分
        if((receive.flags&FLAG_NACK)&&(receive.flags&FLAG_USE)){
            sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
            start=clock();
            continue;
        }
        else if(receive.flags&(FLAG_USE)&&receive.Ack==send.Seq)
        ...
        ...
}

//part of stopWaitReceive
while(1)
{
    ...
    ...
    //收到新的包数据不完整，继续接收
            else {
                cout<<"[Info]校验和错误报文"<<endl;
                send.setFlags(0);
                send.setFlags(FLAG_USE);
                send.setFlags(FLAG_NACK);
                send.calChecksum();
                sendto(client,(char*)&send,sizeof(send),0,
(SOCKADDR*)&t,sizeof(t));
        }
}
```

运行结果

建立连接与开始传输文件：

- 发送端

```
输入文件路径或退出(q)
1.jpg
[Info]发送请求包，开始第一次握手
[Send] ACK=0 SEQ=0 CHECKSUM=e5b0 LEN=0 [SYN]
[Recv] ACK=1 SEQ=0 CHECKSUM=e3ab LEN=0 [SYN][ACK]
[Info]收到确认包，开始第三次握手
[Send] ACK=1 SEQ=1 CHECKSUM=e5b0 LEN=0 [ACK]
[Info]--------三次握手完成--------
[Info]---------开始传输文件---------
[Send] ACK=0 SEQ=0 CHECKSUM=e4f1 LEN=5 [STA]
[Recv] ACK=0 SEQ=0 CHECKSUM=e3ae LEN=0 [ACK]
[Info]起始传输报文发送成功
[Send] ACK=0 SEQ=1 CHECKSUM=e4f8 LEN=10000
[Recv] ACK=1 SEQ=0 CHECKSUM=fff9 LEN=0 [ACK]
[Send] ACK=0 SEQ=0 CHECKSUM=e4f9 LEN=10000
[Recv] ACK=0 SEQ=0 CHECKSUM=fffa LEN=0 [ACK]
```

- 接收端

```
[Info]--------开始握手--------
[Recv] ACK=0 SEQ=0 CHECKSUM=e5b0 LEN=0 [SYN]
[Info]收到请求包，开始第二次握手
[Send] ACK=1 SEQ=0 CHECKSUM=e3ab LEN=0 [SYN][ACK]
[Info]收到确认包
[Recv] ACK=1 SEQ=1 CHECKSUM=e5b0 LEN=0 [ACK]
[Info]--------三次握手完成--------
[Info]--------开始接收文件--------
[Info]收到开始传输报文
[Recv] ACK=0 SEQ=0 CHECKSUM=e4f1 LEN=5 [STA]
上传文件的路径是:
1.jpg
[Info]收到第0个数据包
[Recv] ACK=0 SEQ=1 CHECKSUM=e4f8 LEN=10000
[Info]收到第1个数据包
[Recv] ACK=0 SEQ=0 CHECKSUM=e4f9 LEN=10000
[Info]收到第2个数据包
[Recv] ACK=0 SEQ=1 CHECKSUM=e4f8 LEN=10000
[Info]收到第3个数据包
[Recv] ACK=0 SEQ=0 CHECKSUM=e4f9 LEN=10000
```

超时重传

```
[Recv] ACK=0 SEQ=0 CHECKSUM=fffa LEN=0 [ACK]
[Send] ACK=0 SEQ=1 CHECKSUM=e4f8 LEN=10000
[Info]未收到ACK确认报文，尝试第1次重发......
[Recv] ACK=1 SEQ=0 CHECKSUM=fff9 LEN=0 [ACK]
[Send] ACK=0 SEQ=0 CHECKSUM=e4f9 LEN=10000
[Recv] ACK=0 SEQ=0 CHECKSUM=fffa LEN=0 [ACK]
[Send] ACK=0 SEQ=1 CHECKSUM=e4f8 LEN=10000
[Recv] ACK=1 SEQ=0 CHECKSUM=fff9 LEN=0 [ACK]
```

传输结束与断开连接

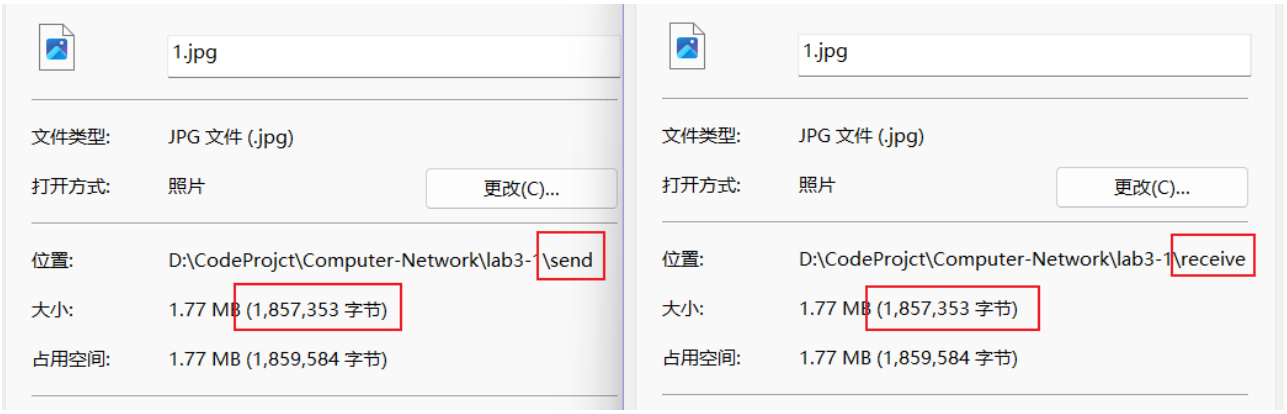
- 发送端

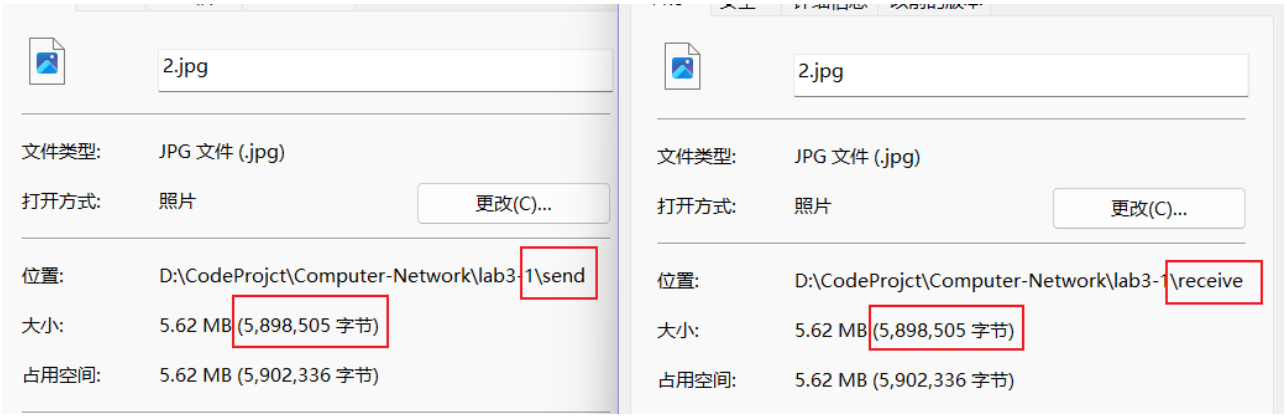
```
[Send] ACK=0 SEQ=0 CHECKSUM=e106 LEN=10000
[Recv] ACK=0 SEQ=0 CHECKSUM=fffa LEN=0 [ACK]
[Send] ACK=0 SEQ=1 CHECKSUM=e0f5 LEN=8994 [END]
[Recv] ACK=1 SEQ=0 CHECKSUM=fff9 LEN=0 [ACK]
[Info]结束报文发送成功
[Info]发送时间：150.362000s
总大小：11968994Bytes
吞吐率：636809.513042Kbps
[Info]---------文件发送成功---------
输入文件路径或退出(q)
q
[Info]发送第一次挥手包
[Send] ACK=0 SEQ=1 CHECKSUM=e591 LEN=0 [FIN]
[Recv] ACK=1 SEQ=0 CHECKSUM=e3ad LEN=0 [ACK]
[Info]收到第三次挥手包
[Recv] ACK=0 SEQ=1 CHECKSUM=e391 LEN=0 [FIN]
[Info]发送第四次挥手包
[Send] ACK=1 SEQ=0 CHECKSUM=0 LEN=0 [ACK]
[Info]---------断开链接--------
请按任意键继续. . .
```

- 接收端

```
[Recv] ACK=0 SEQ=1 CHECKSUM=e105 LEN=10000
[Info]收到第1195个数据包
[Recv] ACK=0 SEQ=0 CHECKSUM=e106 LEN=10000
[Info]收到第1196个数据包
[Recv] ACK=0 SEQ=1 CHECKSUM=e0f5 LEN=8994 [END]
[Info]该数据包为最后一个包
[Info]--------文件接受完成--------
输入文件名:
3.jpg
[Info]---------文件保存成功---------
[Info]发送端尝试断开连接
[Info]--------开始挥手--------
[Info]收到第一次挥手包
[Recv] ACK=0 SEQ=1 CHECKSUM=e591 LEN=0 [FIN]
[Info]发送第二次挥手包
[Send] ACK=1 SEQ=0 CHECKSUM=e3ad LEN=0 [ACK]
[Info]发送第三次挥手包
[Send] ACK=0 SEQ=1 CHECKSUM=e391 LEN=0 [FIN]
[Info]发送端未收到第二次挥手报文，重发.....
[Info]发送第三次挥手包
[Send] ACK=0 SEQ=1 CHECKSUM=e391 LEN=0 [FIN]
[Info]收到第四次挥手包
[Recv] ACK=1 SEQ=0 CHECKSUM=0 LEN=0 [ACK]
[Info]---------四次挥手完成---------
请按任意键继续. . .
```

传输结果





可以看出，传输后文件的大小没有发生改变，且文件打开后正常显示为图片或文本文档。