计算机网络课程实验报告

郭裕彬 2114052 物联网工程

实验3-3：基于UDP服务设计可靠传输协议并编程实现-SR

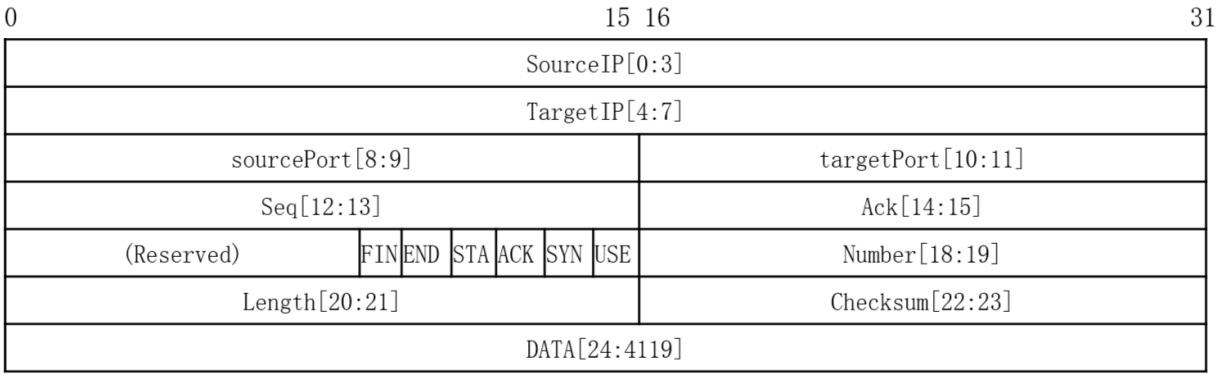
发送端使用地址：127.0.0.1:9995
接收端使用地址：127.0.0.1:9997
路由器使用地址：127.0.0.1:9996

实验要求

- 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，发送窗口和接收窗口采用相同大小，支持选择确认，完成给定测试文件的传输。
- 协议设计：数据包格式，发送端和接收端交互，详细完整
- 流水线协议：多个序列号
- 发送缓冲区、接收缓冲区
- 累积确认：SR（Selective Repeat）
- 日志输出：收到/发送数据包的序号、ACK、校验和等，发送端和接收端的窗口大小等情况，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

具体代码实现同实验3-1，此处不再叙述。

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
- 发送端对文件进行处理，封装STA包放入缓冲区包队列第一个位置，之后依次放入读取并封装好的数据包
- 使用两个标记位数组来标记每一个包是否已经发送（haveSend）过以及是否收到ACK(recvAck)。初始化时，发送端的发送线程从队列中依次取出包并发送直至发送窗口满。
- 接收端一直监听，收到第一个STA数据包时进入接收状态，循环接收数据包：
    - 使用一个标记位数组(recvFlag)来标记每一个包的接收情况：收到有效且大于接收窗口左侧base的包时放入缓存，更新该包对应的标志位；只要收到的包是有效的，都返回对应包序号的ACK。
    - 从窗口最左侧base+1开始向右遍历，只要对应包接收标记位为收到，就将其从缓存区读取数据放入数据缓冲区，并将base右移一位，即接收窗口右滑一个单位；只要遇到标记为为未收到的情况，就跳出这个遍历，重新开始一次循环。
- 发送端的接收线程不断监听接收端返回的ACK包，每收到一个ACK包就将对应的recvAck位置标记为收到，当收到的ACK是最左侧的base+1时，base也即发送窗口向右滑动直至下一个未收到ACK的包位置。
- 发送端的发送线程在未发送完时不断检测发送窗口范围内的数据包：
    - 该包未收到ACK，若对应的haveSend位置也为0，说明之前没有发送过，启动定时器并发送，更改标志位；
    - 该包未收到ACK，对应的haveSend位置为1，判断定时器是否超时，超时则重发、重置定时器
    - 该包收到ACK，不做任何动作
- 循环上述步骤直至发送端发送队列中的最后一个数据包END包
- 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

握手挥手

- 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。
- 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。
- 具体代码实现除了将序列号从0/1更新为逐步增大的唯一序列号外与实验3-1的代码一致，此处不再叙述。

差错校验

与实验3-1的实现一致，不再叙述

滑动窗口&超时重传

发送端

定义四个全局变量`windowSize`、`base`、`lastWritten`、`nextSeqNum`分别记录窗口大小、发送窗口左侧左边的位置（不在窗口中）、缓冲区队列中最后一个包的位置和下一个待发送的包的位置。

发送线程负责初始化发送满窗口的数据包、在窗口滑动时发送新的数据包以及检测超时重发对应的数据包。

```
int windowSize = 1;//窗口大小
int base = -1;//窗口左侧最大已确认的包位置
int lastWritten = 0;//最后的包位置
int nextSeqNum = 0;//下一个待发送的包位置

UDP_packet sendList[10000];//发送窗口包队列
bool recvAck[10000]={0};//确认收到标记位数组
bool haveSend[10000]={0};//已经发送标记位数组
time_t startTime[10000];//发送时间数组
//发送线程
DWORD WINAPI sendHandle(LPVOID lparam){
    time_t end;
```

```
    while(base<lastWritten){
        //从窗口最左侧开始向右移动遍历
        for(nextSeqNum =
base+1;nextSeqNum<=base+WindowSize&&nextSeqNum<=lastWritten;nextSeq
Num++){
            if(recvAck[nextSeqNum]==0){
                //如果遍历到的位置未收到过ACK
                if(haveSend[nextSeqNum]==0){
                    //之前没有发送过这个位置对应的包
                    //发送，并启动定时器，标记该位置已发送过
                    startTime[nextSeqNum] = clock();
                    sendto(client,
(char*)&sendList[nextSeqNum],sizeof(sendList[nextSeqNum]),0,
(SOCKADDR*)&targetAd,sizeof(targetAd));
                    haveSend[nextSeqNum]=1;
                    cout<<"[Send]发送第"<<nextSeqNum<<"个数据包 ";
                    sendList[nextSeqNum].printInfo();
                    printWindowInfo();
                }
                else{
                    //之前发送过，检测是否超时，超时则重发并重启该位置定时器
                    time_t end = clock();
                    if((double)(end-
startTime[nextSeqNum])/CLOCKS_PER_SEC >= 1){
                        startTime[nextSeqNum] = clock();
                        sendto(client,
(char*)&sendList[nextSeqNum],sizeof(sendList[nextSeqNum]),0,
(SOCKADDR*)&targetAd,sizeof(targetAd));
                        cout<<"[Resend]重发第"<<nextSeqNum<<"个数据
包";
                        sendList[nextSeqNum].printInfo();
                        printWindowInfo();
                    }
                }
            }
        }
    }
    return 1;
}
```

具体实现为，每一次循环开始时线程都遍历[nextSeqNum=base+1,(base+windowSize)||(lastWritten)]范围内的所有数据包，每一个数据包有如下几种可能：

- recvAck数组中对应位置为1，说明该包发出且收到了接收端发回的ACK，什么都不做；
- recvAck数组中对应位置为0，haveSend数组中对应位置为0，说明之前还未发送过这个包，将其发送，并在startTime数组中设置该包的定时器启动，开始重发倒计时，标记该包已发送；
- recvAck数组中对应位置为0，haveSend数组中对应位置为1，说明之前已经发送过这个包但没有收到ACK，此时需要检查定时器，如果超时就需要对该包进行重发，同时重新设置定时器。

这个循环范围中的base收到接收线程的影响，在接收线程收到窗口最左侧的ACK时（也就是base+1的ACK），窗口开始右滑，体现在发送线程就是发送窗口的滑动。

接收端

定义两个全局变量`windowSize`、`base`、分别记录窗口大小、接收窗口左侧左边的位置（不在窗口中）。

接收端循环收包，在收到有序包后更新接收窗口，进行右滑。

```
while(1){
        int len = sizeof(SOCKADDR);
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&t,&len);
        if(receive.checkChecksum()&&receive.flags&FLAG_USE){
            if(receive.Seq > base){
                //大于窗口左侧，放入缓冲区及更新flag
                RecvList[receive.Seq]=receive;
                recvFlag[receive.Seq]=1;
            }
            //返回ACK包，打印信息
            ···print and send ACK···
        }
        //从窗口最左侧开始，交付所有有序的包，遇到失序时跳出
        for(int i=base+1,right=base+WindowSize;i<=right;i++){
            if(recvFlag[i]){
                //遇到最后一个包时也跳出
```

```
                if(RecvList[i].flags&FLAG_END){
                        length = RecvList[i].length;
                        cout<<"[Info]该数据包为最后一个包"<<endl;
                        cout<<"[Info]--------文件接受完成--------"
<<endl;

 memcpy(&receivebuf[i],&RecvList[i].data,length);
                        goto FINISH;
                    }
                else{
                    //将缓存的包交付到缓冲区中

 memcpy(&receivebuf[i],&RecvList[i].data,MAX_SIZE);
                    base++;
                }
            }
            else{
                break;
            }
        }
    }
```

收到包后，只要seq是大于窗口左侧base的包，就更新recvFlag对应位置为1，同时将包放入缓存区；之后遍历整个窗口[base+1,right=base+Windowsize]的范围，只要recvFlag对应位置为1，说明该位置是有序的包，交付给缓冲区，同时将base加1，标识窗口右滑一个单位；遇到传输文件的最后一个包时，交付后直接跳出到结束部分；遇到第一个recvFlag对应位置为0的情况，说明这个位置和之后的包要么是没有收到，要么是乱序的包，不进行处理，跳出遍历过程。

选择确认

- 接收端

  接收端循环接收数据包，只要是收到了发送端发来的有意义通过校验和的包，都返回对应seq的ACK包。具体代码在上一点中。

- 发送端

发送端使用接收线程循环接收从接收端发回的ACK包，只要收到，就更新recvAck数组对应位置为1。如果收到的ACK是窗口最左侧的，即ack=base+1，就开始更新base，即开始滑动窗口：只要recvAck对应位置为1，就将base向右移动一个单位，直至遇见第一个非1位置为止，base回退一个单位并退出滑动过程。

```
//接收线程
DWORD WINAPI receiveHandle(LPVOID lparam){
    while(base < lastWritten){
        UDP_packet receive;
        int len=sizeof(targetAd);
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&targetAd,&len);

 if(receive.checkChecksum()&&receive.flags&FLAG_USE&&receive
.flags&FLAG_ACK){
            if(receive.Ack>=base){
                //收到确认ACK，窗口右移
                base = receive.Ack;
                cout<<"[Recv] ";
                receive.printInfo();
                cout<<"[Info] ";
                printWindowInfo();
            }
        }
    }
    return 1;
}
```

单向传输

在实验3-1`sendFile`的基础上进行修改，使得读取文件内容到缓冲区之后将其打包封装成能够发出的数据包放入缓冲区包队列，而不是直接发送；之后再创建发送和接收线程进行数据包传输，线程返回后判断是否传输完成，同时打印相关信息。

```
void sendFile(SOCKADDR_IN s,SOCKADDR_IN t,char* filePath,SOCKET
&client){
    //读取并发送文件的部分
    //···some code···
    //读取文件到缓冲区并记录总共的包数和最后一个包的数据长度
```

```
//···some code···
lastWritten = amount;
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_USE);send.setFlags(FLAG_STA);
send.setLength(strlen(filePath));
memcpy(send.data,filePath,strlen(filePath));
send.calChecksum();
sendList[0]=send;
//把带有数据的包存入发送缓冲队列
for(int i=1;i<=amount+1;i++){
    UDP_packet pk;
    pk.setSourceIP(s);pk.setTargetIP(t);
    pk.setNumber(amount);
    pk.setFlags(FLAG_USE);
    pk.setSeq(i);
    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];
        }
    }
    pk.calChecksum();
    sendList[i]=pk;
}

time_t startsend=clock();
```

```
    time_t endsend;
    cout<<"[Info]---------开始传输文件---------"<<endl;
    if(base<lastWritten){
        HANDLE recvThread =
CreateThread(NULL,NULL,receiveHandle,LPVOID(),0,NULL);
        HANDLE sendThread =
CreateThread(NULL,NULL,sendHandle,LPVOID(),0,NULL);
        WaitForSingleObject(recvThread,INFINITE);
        WaitForSingleObject(sendThread,INFINITE);
        if(base == lastWritten){
            cout<<"[Info]---------传输完成---------"<<endl;
            endsend = clock();
        }
    }
    //输出本次传输的相关数据
    long fileSize=(long)(amount*MAX_SIZE+p);
    double time=(double)(endsend-startsend)/CLOCKS_PER_SEC;
    double rate=(double)(fileSize*8.0)/(time*1000);
    printf("[Info]发送时间：%lfs\n总大小：%dBytes\n吞吐率：
%lfKbps\n",time,fileSize,rate);
    cout<<"[Info]---------文件发送成功---------"<<endl;
    return ;
}
```

接收端额外实现了一个缓存和交付的设计，因为SR协议设计中缓存失序的包是相对于包处理程序来说的，交付是相对于上层应用程序来说的，本次实验中对传输文件的处理在同一个程序中实现，所以设计了一个UDP_packet数组来作为缓存区，对应一个二维字符数组保存包中的数据部分来假设为包处理程序交付给上层程序的接口，在之前的代码可以看到，recvList作为缓存区，通过memcpy拷贝到receivebuf缓冲区中作为交付的实现。

实验结果

运行截图

- 握手

```
输入窗口大小
8
[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=e3e4 LEN=5 [STA]
[Sent] ACK=0 SEQ=0 CHECKSUM=e3ae LEN=0 [ACK]
上传文件的路径是：
1.jpg
[Recv] ACK=0 SEQ=1 CHECKSUM=e3eb LEN=4096
[Info] Base=0 Win=8 Ack: 1
[Send] ACK=1 SEQ=0 CHECKSUM=e3ad LEN=0 [ACK]
[Recv] ACK=0 SEQ=2 CHECKSUM=e3ea LEN=4096
[Info] Base=1 Win=8 Ack: 2
[Send] ACK=2 SEQ=0 CHECKSUM=e3ac LEN=0 [ACK]
[Recv] ACK=0 SEQ=3 CHECKSUM=e3e9 LEN=4096
[Info] Base=2 Win=8 Ack: 3
```

```
输入窗口大小：
8
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]发送第0个数据包 ACK=0 SEQ=0 CHECKSUM=e3e4 LEN=5 [STA]
Win=8 Base=-1 NextSeqNum=0
[Send]发送第1个数据包 ACK=0 SEQ=1 CHECKSUM=e3eb LEN=4096
Win=8 Base=-1 NextSeqNum=1
[Send]发送第2个数据包 ACK=0 SEQ=2 CHECKSUM=e3ea LEN=4096
Win=8 Base=-1 NextSeqNum=2
[Send]发送第3个数据包 ACK=0 SEQ=3 CHECKSUM=e3e9 LEN=4096
Win=8 Base=-1 NextSeqNum=3
[Send]发送第4个数据包 ACK=0 SEQ=4 CHECKSUM=e3e8 LEN=4096
Win=8 Base=-1 NextSeqNum=4
[Send]发送第5个数据包 ACK=0 SEQ=5 CHECKSUM=e3e7 LEN=4096
Win=8 Base=-1 NextSeqNum=5
```

- 选择重传、重发

```
[Recv] ACK=0 SEQ=57 CHECKSUM=e3b3 LEN=4096
[Info] Base=56 Win=8 Ack: 57
[Send] ACK=57 SEQ=0 CHECKSUM=e375 LEN=0 [ACK]
[Recv] ACK=0 SEQ=59 CHECKSUM=e3b1 LEN=4096
[Info] Base=57 Win=8 Ack: 59
[Send] ACK=59 SEQ=0 CHECKSUM=e373 LEN=0 [ACK]
[Recv] ACK=0 SEQ=60 CHECKSUM=e3b0 LEN=4096
[Info] Base=57 Win=8 Ack: 59 60
[Send] ACK=60 SEQ=0 CHECKSUM=e372 LEN=0 [ACK]
[Recv] ACK=0 SEQ=61 CHECKSUM=e3af LEN=4096
[Info] Base=57 Win=8 Ack: 59 60 61
[Send] ACK=61 SEQ=0 CHECKSUM=e371 LEN=0 [ACK]
[Recv] ACK=0 SEQ=62 CHECKSUM=e3ae LEN=4096
[Info] Base=57 Win=8 Ack: 59 60 61 62
[Send] ACK=62 SEQ=0 CHECKSUM=e370 LEN=0 [ACK]
[Recv] ACK=0 SEQ=63 CHECKSUM=e3ad LEN=4096
[Info] Base=57 Win=8 Ack: 59 60 61 62 63
[Send] ACK=63 SEQ=0 CHECKSUM=e36f LEN=0 [ACK]
[Recv] ACK=0 SEQ=64 CHECKSUM=e3ac LEN=4096
[Info] Base=57 Win=8 Ack: 59 60 61 62 63 64
[Send] ACK=64 SEQ=0 CHECKSUM=e36e LEN=0 [ACK]
[Recv] ACK=0 SEQ=65 CHECKSUM=e3ab LEN=4096
[Info] Base=57 Win=8 Ack: 59 60 61 62 63 64 65
[Send] ACK=65 SEQ=0 CHECKSUM=e36d LEN=0 [ACK]
[Recv] ACK=0 SEQ=58 CHECKSUM=e3b2 LEN=4096
[Info] Base=57 Win=8 Ack: 58 59 60 61 62 63 64 65
[Send] ACK=58 SEQ=0 CHECKSUM=e374 LEN=0 [ACK]
[Recv] ACK=0 SEQ=66 CHECKSUM=e3aa LEN=4096
[Info] Base=65 Win=8 Ack: 66
[Send] ACK=66 SEQ=0 CHECKSUM=e36c LEN=0 [ACK]
```

```
[Recv] ACK=56 SEQ=0 CHECKSUM=e376 LEN=0 [ACK]
[Info] Win=8 Base=55 NextSeqNum=57
[Send]发送第64个数据包 ACK=0 SEQ=64 CHECKSUM=e3ac LEN=4096
Win=8 Base=56 NextSeqNum=64
[Recv] ACK=57 SEQ=0 CHECKSUM=e375 LEN=0 [ACK]
[Info] Win=8 Base=56 NextSeqNum=58
[Send]发送第65个数据包 ACK=0 SEQ=65 CHECKSUM=e3ab LEN=4096
Win=8 Base=57 NextSeqNum=65
[Recv] ACK=59 SEQ=0 CHECKSUM=e373 LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=62
[Recv] ACK=60 SEQ=0 CHECKSUM=e372 LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=62
[Recv] ACK=61 SEQ=0 CHECKSUM=e371 LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=64
[Recv] ACK=62 SEQ=0 CHECKSUM=e370 LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=63
[Recv] ACK=63 SEQ=0 CHECKSUM=e36f LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=64
[Recv] ACK=64 SEQ=0 CHECKSUM=e36e LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=65
[Recv] ACK=65 SEQ=0 CHECKSUM=e36d LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=64
[Resend]重发第58个数据包ACK=0 SEQ=58 CHECKSUM=e3b2 LEN=4096
Win=8 Base=57 NextSeqNum=58
[Recv] ACK=58 SEQ=0 CHECKSUM=e374 LEN=0 [ACK]
[Info] Win=8 Base=57 NextSeqNum=65
[Send]发送第66个数据包 ACK=0 SEQ=66 CHECKSUM=e3aa LEN=4096
Win=8 Base=65 NextSeqNum=66
[Send]发送第67个数据包 ACK=0 SEQ=67 CHECKSUM=e3a9 LEN=4096
Win=8 Base=65 NextSeqNum=67
```

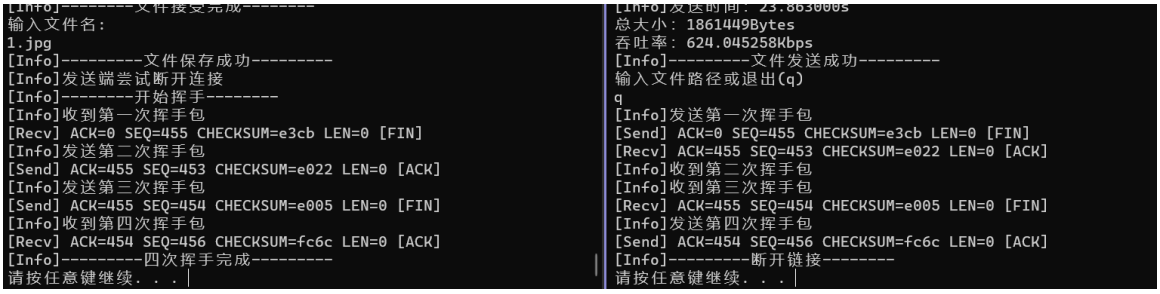
接收端对收到的包返回对应的ACK，其中58号包缺失，发送端不断收到ACK，但缺少58的ACK，base不会移动，直至超时，重发58号包

- 传输结束

```
[Recv] ACK=0 SEQ=448 CHECKSUM=e22c LEN=4096
[Info] Base=441 Win=8 Ack: 443 444 445 446 447 448
[Send] ACK=448 SEQ=0 CHECKSUM=e1ee LEN=0 [ACK]
[Recv] ACK=0 SEQ=449 CHECKSUM=e22b LEN=4096
[Info] Base=441 Win=8 Ack: 443 444 445 446 447 448 449
[Send] ACK=449 SEQ=0 CHECKSUM=e1ed LEN=0 [ACK]
[Recv] ACK=0 SEQ=442 CHECKSUM=e232 LEN=4096
[Info] Base=441 Win=8 Ack: 442 443 444 445 446 447 448 449
[Send] ACK=442 SEQ=0 CHECKSUM=e1f4 LEN=0 [ACK]
[Recv] ACK=0 SEQ=450 CHECKSUM=e22a LEN=4096
[Info] Base=449 Win=8 Ack: 450
[Send] ACK=450 SEQ=0 CHECKSUM=e1ec LEN=0 [ACK]
[Recv] ACK=0 SEQ=451 CHECKSUM=e229 LEN=4096
[Info] Base=450 Win=8 Ack: 451
[Send] ACK=451 SEQ=0 CHECKSUM=e1eb LEN=0 [ACK]
[Recv] ACK=0 SEQ=452 CHECKSUM=e228 LEN=4096
[Info] Base=451 Win=8 Ack: 452
[Send] ACK=452 SEQ=0 CHECKSUM=e1ea LEN=0 [ACK]
[Recv] ACK=0 SEQ=453 CHECKSUM=e227 LEN=4096
[Info] Base=452 Win=8 Ack: 453
[Send] ACK=453 SEQ=0 CHECKSUM=e1e9 LEN=0 [ACK]
[Recv] ACK=0 SEQ=454 CHECKSUM=e216 LEN=1865 [END]
[Info] Base=453 Win=8 Ack: 454
[Send] ACK=454 SEQ=0 CHECKSUM=e1e8 LEN=0 [ACK]
[Info]该数据包为最后一个包
[Info]--------文件接受完成--------
输入文件名：
1.jpg
[Info]---------文件保存成功---------
```

```
[Recv] ACK=442 SEQ=0 CHECKSUM=e1f4 LEN=0 [ACK]
[Info] Win=8 Base=441 NextSeqNum=447
[Send]发送第450个数据包 ACK=0 SEQ=450 CHECKSUM=e22a LEN=4096
Win=8 Base=449 NextSeqNum=450
[Send]发送第451个数据包 ACK=0 SEQ=451 CHECKSUM=e229 LEN=4096
Win=8 Base=449 NextSeqNum=451
[Send]发送第452个数据包 ACK=0 SEQ=452 CHECKSUM=e228 LEN=4096
Win=8 Base=449 NextSeqNum=452
[Send]发送第453个数据包 ACK=0 SEQ=453 CHECKSUM=e227 LEN=4096
Win=8 Base=449 NextSeqNum=453
[Send]发送第454个数据包 ACK=0 SEQ=454 CHECKSUM=e216 LEN=1865 [END]
Win=8 Base=449 NextSeqNum=454
[Recv] ACK=450 SEQ=0 CHECKSUM=e1ec LEN=0 [ACK]
[Info] Win=8 Base=449 NextSeqNum=452
[Recv] ACK=451 SEQ=0 CHECKSUM=e1eb LEN=0 [ACK]
[Info] Win=8 Base=450 NextSeqNum=452
[Recv] ACK=452 SEQ=0 CHECKSUM=e1ea LEN=0 [ACK]
[Info] Win=8 Base=451 NextSeqNum=453
[Recv] ACK=453 SEQ=0 CHECKSUM=e1e9 LEN=0 [ACK]
[Info] Win=8 Base=452 NextSeqNum=454
[Recv] ACK=454 SEQ=0 CHECKSUM=e1e8 LEN=0 [ACK]
[Info] Win=8 Base=453 NextSeqNum=454
[Info]---------传输完成---------
[Info]发送时间：23.863000s
总大小：1861449Bytes
吞吐率：624.045258Kbps
[Info]---------文件发送成功---------
输入文件路径或退出(q)
```

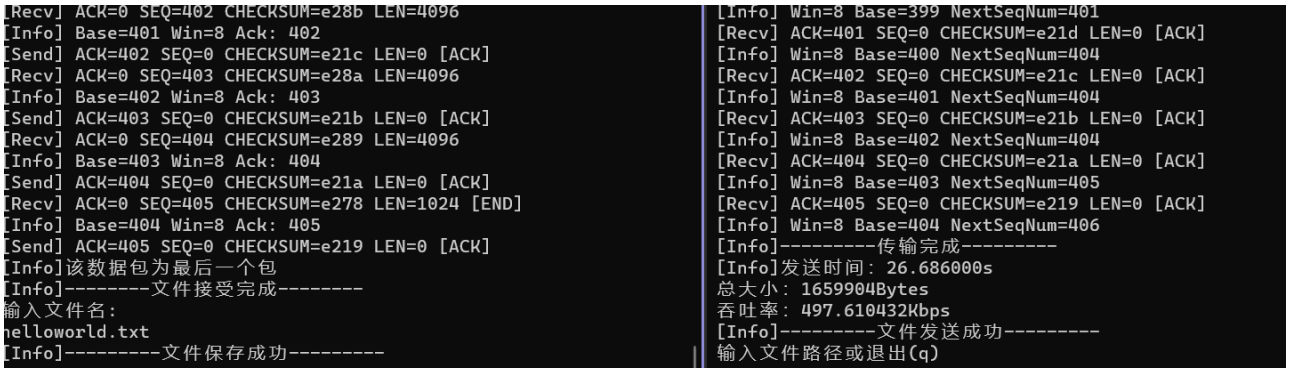
- 挥手

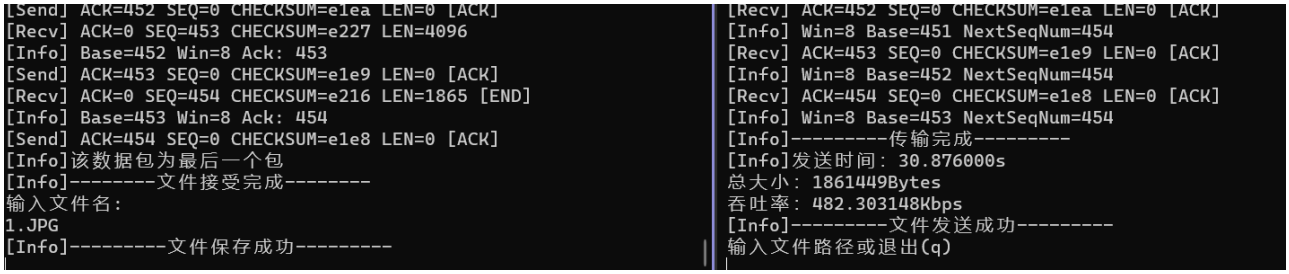


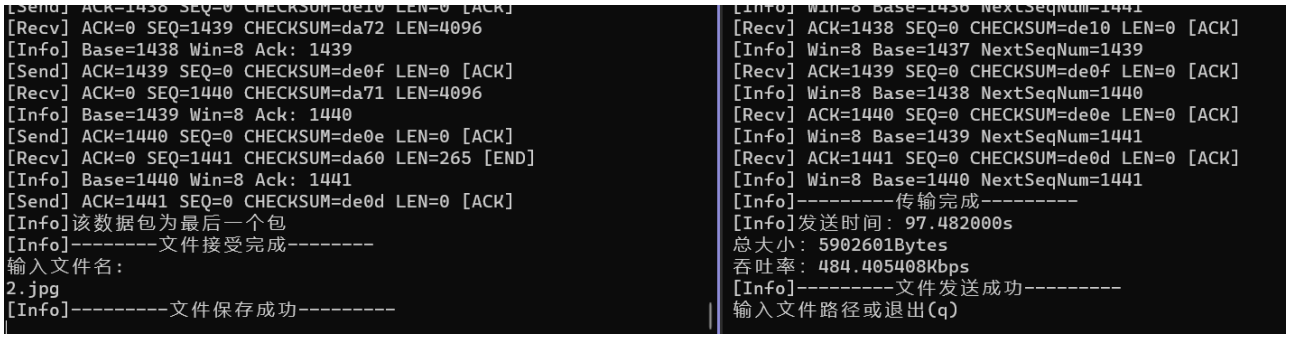
测试结果

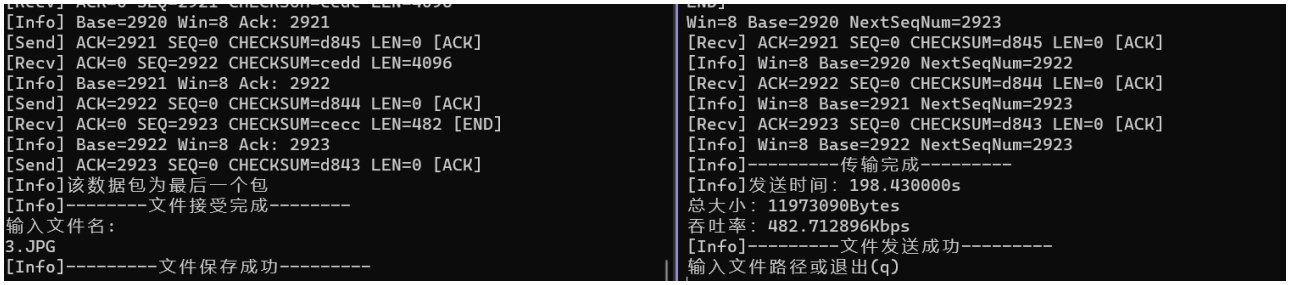
与讲解实验时设置一致，路由器设置为丢包率3%，延迟3ms，窗口大小设置为8：

	文件大小/BYTES	发送时间/S	吞吐率
helloworld.txt	1659904	26.686	497.610kbps
1.jpg	1861449	23.863	482.303kbps
2.jpg	5902601	97.482	484.405kbps
3.jpg	11973090	198.43	482.713kbps









思考

在最初实验设计的时候，将缓存区、标记位数组等设置成了一个很大的全局变量，这样一来在实现窗口滑动的过程中就相当于是利用两个指针指定了窗口的范围，在这个大的包队列上向右移动，这种设计实现比较简单，并且在实验环境下是能够完成实验要求的，但实际应用中，因为文件大小实际可以远大于程序能够拥有的内存空间，更为合理的做法还是设计一个较小的缓存区和对应的标记位数组，通过指针记录base对应的在这个缓冲区或数组中的位置，并通过移动指针来实现窗口的滑动以及空间的循环重复利用。

实验讲解后对接收端进行了相应的修改，经过测试，能够正常进行文件收发。改动代码如下，在.cpp文件中以注释方式呈现。

```
UDP_packet RecvList[33];
bool recvFlag[33]={false};
...
void recvFile(){
......
......
while(1){
        int len = sizeof(SOCKADDR);
        recvfrom(client,(char*)&receive,sizeof(receive),0,
(SOCKADDR*)&t,&len);
        if(receive.checkChecksum()&&receive.flags&FLAG_USE){
            if(receive.Seq > base){
```

```
                //大于窗口左侧，放入缓存区及更新flag
                RecvList[receive.Seq%WindowSize]=receive;
                recvFlag[receive.Seq%WindowSize]=1;
            }
            //返回ACK包，打印信息
            ......
        }
        //从窗口最左侧开始，交付所有有序的包，遇到失序时跳出
        for(int i=base+1,right=base+WindowSize;i<=right;i++){
            int posi = i%WindowSize;
            if(recvFlag[posi]){
                //遇到最后一个包时也跳出
                 if(RecvList[posi].flags&FLAG_END){
                        length = RecvList[posi].length;
                        cout<<"[Info]该数据包为最后一个包"<<endl;
                        cout<<"[Info]--------文件接受完成--------"
<<endl;

memcpy(&receivebuf[i],&RecvList[posi].data,length);
                        recvFlag[posi]=0;
                        goto FINISH;
                    }
                else{
                    //将缓存的包交付到缓冲区中

memcpy(&receivebuf[i],&RecvList[ii%WindowSize].data,MAX_SIZE);
                    recvFlag[posi]=0;
                    base++;
                }
            }
            else{
                break;
            }
        }
    }
......
......
}
```