南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络实验报告

Lab3-1：基于 UDP 服务设计可靠传输协议并编程实现

冯思程 2112213

年级：2021 级

专业：计算机科学与技术

指导教师：吴英、文静静

2023 年 11 月 17 日

摘要

本次实验，根据要求在 UDP 不可靠信道上设计了可靠传输协议，基于 TCP 协议设计了类似的三次握手连接、四次挥手断开连接和可靠的数据传输协议（基于 rdt3.0 协议），并设计了合理的消息格式，并利用到提供的 router 进行包括超时重传机制在内的多种测试，最后综合分析编写报告。

关键字：UDP、可靠数据传输协议、rdt3.0 协议、router、超时重传

目录

一、 预备工作及实验环境

(一) 实验要求与功能

实验要求的基础功能与要求:

1. 实现单向数据传输（一端发数据，一端返回确认）。
2. 对于每个任务要求给出详细的协议设计。
3. 完成给定测试文件的传输，显示传输时间和平均吞吐率。
4. 性能测试指标：吞吐率、延时，给出图形结果并进行分析。
5. 完成详细的实验报告（每个任务完成一份，主要包含自己的协议设计、实现方法、遇到的问题、实验结果，不要抄写太多的背景知识）。
6. 编写的程序应该结构清晰，具有较好的可读性。
7. 提交程序源码、可执行文件和实验报告。
8. 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。

合理的自行扩展功能:

1. 使用非阻塞实现本次实验，在非阻塞模式下，套接字操作（sendto 和 recvfrom）不会使程序挂起等待操作完成。这表示程序可以继续执行其他任务，而不是闲置等待，从而提高了整体效率和响应能力。
2. 在断开连接的设计中，考虑到了最后一个 ACK 可能丢失的情况，让 client 等待 2MSL 来确保 client 和 server 的正确关闭。
3. 在设计协议的时候额外的考虑到了失序等特殊情况，在分析的时候也进行了丰富的测试对比分析。
4. 不只局限于单向，我也考虑到了双向的丢包，即 server 的 ACK 丢包的处理方法，具体见后文。

(二) 实验环境与说明

具体的实验环境配置如下:

Windows 版本	vs code 版本	docker 版本	g++ 版本
windows11	1.82.0	24.0.6	8.1.0

表 1: 实验环境说明表

代码文件使用 GBK 编码，以支持中文字符。makefile 文件使用正常的 UTF-8 编码。
感谢老师与助教的审查批阅与指正，辛苦!

二、 实验过程

(一) 协议设计

先展示一下整体的时许协议流程，如图：

图 1: 三次握手示意图

下面分点进行说明：

1. 报文格式

这里报文的格式是由我自行设计以满足这次实验的要求，同时报文格式部分参考到了 TCP 协议的报文格式，报文分为两个部分：报文头和报文段。

报文头 报文头一共包括源 IP（4 字节）、目的 IP（4 字节）、源 PORT（2 字节）、目的 PORT（2 字节）、序列号（4 字节）、确认号（4 字节）、发送文件大小（4 字节）、标志（2 字节）、校验和（2 字节）。

一共 28 字节，其中由于本次实验测试使用的是本地回环地址，所以源 IP 和目的 IP 这两个字段一直没用上。其中发送文件大小是记录要发送的文件的大小的，后文会进一步讲到这个字

段。

报文段 这里我设计的报文段就是装载数据的数据段。这里最大字节数我设置为 10000 字节，因为 router 转发的数据包最大是 15000 字节。

源 IP	
目的 IP	
源 PORT	目的 PORT
序列号	
确认号	
发送文件大小	
标志	校验和
报文段	

表 2: 报文格式

标志位 报文头的标志字段可以从下面设计的四种选择若干种进行加到 flag 上从而实现标志设置，其中包括 TCP 协议中用到的 SYN、ACK、FIN，还有一个我自己设计的 SFileName 标志，这个是用来表示传输文件的具体数据前发送文件名和文件大小的那条报文的，代码如下：

标志位

```
//设置不同的标志位，用到的包括SYN、ACK、FIN，SFileName表示传输了文件名字
const unsigned short SYN = 0x1;//0001
const unsigned short ACK = 0x2;//0010
const unsigned short FIN = 0x4;//0100
const unsigned short SFileName = 0x8;//1000
```

2. 建立连接：三次握手（包含超时重传）

这个三次握手的我是基于 TCP 协议进行设计的，实现的示意图如下：

图 2: 三次握手示意图

这里在 UDP 的不可靠信道上进行三次握手来建立一个连接，需要 client 和 server 总共发送 3 个包。三次握手的目的是连接服务器指定端口，建立连接，并同步连接双方的序列号和确认号。这里的序列号我设计的是双端（client 和 server）各自维护各自序列号，然后确认号的维护机制是根据情况相应地设置。

第一次握手 client 将报文标志位 SYN 置为 1，随机产生一个序号值 seq=x。

第二次握手 server 接收到了 client 发起的连接请求后进行回应，这条报文 ACK、SYN 置 1，确认号是 x+1（client 的发来消息的序列号 +1）同时，自身的序列号也会随机出一个 y。

第三次握手 client 发给 server 的一个确认报文，序列号是 x+1（这也是唯一和 TCP 协议不同的地方，ACK 报文在 TCP 协议中是不消耗序列号的，这里我将其设置也会消耗序列号），确认号是 server 发来的报文的序列号 +1。

由于这里存储的是相对序列号，所以图中的 x 和 y 可以都看作 0。

三次握手的超时重传机制 这里我设计了基于 rdt3.0 协议的超时重传机制，下面分别是 client 和 server 的重传机制实现流程：

client：

1. 发送第一次握手，在发送后接着就开始计时
2. 接收 server 发来的第二次握手，在这个不断等待接收的过程中，如果第一次握手超时，则会重传第一次握手并重新开始计时。这里注意，我设置最大重传次数为 10 次，如果 10 次重传都失败则会退出。
3. 发送第三次握手，这是一个 ACK 报文，不需要进行重传。发送后 client 成功建立连接。

server：

1. 接收第一次握手
2. 接收到第一次握手后，发送第二次握手消息并在发送后接着进行计时。
3. 接收 client 的第三次握手，在这个不断等待接收的过程中，如果第二次握手超时，则会重传第二次握手并重新开始计时。这里注意，我设置最大重传次数为 10 次，如果 10 次重传都失败则会退出。接收第三次握手成功后 server 成功建立连接，现在双方可以开始进行数据传输了。

3. 不可靠信道上的可靠数据传输：差错检验、接收确认、超时重传（基于 rdt3.0 协议）、流量控制（停等机制）

本次实验中从 client 向 server 发送文件，文件较大，需要拆分成多个数据报文进行传输，一次完整的文件传输的数据报文一共分为三类，如下：

1. 第一类数据报文是装载文件名字和文件大小的数据报文。
2. 第二类数据报文是满载数据报文，这类报文传输的是具体的文件数据，满载就是这个报文的报文段可以装满。
3. 第三类数据报文是未满载数据报文，这类报文只能是 0 个或者 1 个，装载的数据大小是文件大小除以报文段最大装载大小的余数。

数据传输中我首先实现了差错检验、接受确认机制。数据传输协议设计我基于 **rdt3.0 协议**进行改进，rdt3.0 协议的序列号是 0/1 交替的，但是我基于此进行了改进，依旧沿用上文中三次握手连接中由 client 和 server 双端各自维护序列号进行递增，在正常状态下，client 发送一个序列号为 k 的数据报文，接收端会回复一个确认号为 k 的 ACK 报文（client 会对来自 server 的

ACK 报文进行校验和检查确保数据报文传输无误），同时这里依旧沿用三次握手中 ACK 报文也会占用序列号的设计，server 发送的 ACK 报文的序列号是在 server 维护的序列号上每次自动递增 1 进行计算的。示意图如下：

图 3: 可靠数据传输（左为 client，右为 server）

接下来面对丢包、延时、失序等异常状态，需要扩展实现超时重传机制，超时重传机制也是基于 rdt3.0 协议进行设计。下面根据不同的情况具体说明超时重传协议的设计：

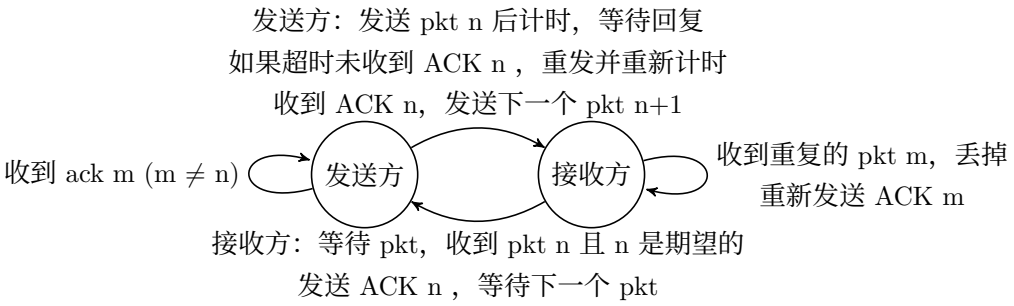
1. 第一种情况就是在**理想情况**下，没有丢包、没有延时、没有失序问题。client 按顺序依次发送装载文件名和文件大小的数据包、满载数据包、未满载数据包（可能没有），然后 server 会按顺序接收 client 发来的数据包，每接收到一个回复一个 ACK 包，确认号是发来数据包的序列号，ACK 包的序列号是 server 维护的序列号自动递增。另外基于**停等协议**，这里只有 client 确认了 server 回复的 ACK 包正确后才会继续发送数据包。

2. 第二种情况是存在数据包丢失的情况，在数据包丢失的时候，client 为每个数据包设置了计时器，当超时没有收到 server 发来的 ACK 包的时候就会进行重传并重新计时，这里设置的最大重传次数为 10 次，如果重传次数超限会报错退出。server 仍是按顺序接收，收到数据包就会返回一个 ACK 包，其他同上。

3. 第三种情况是存在 server 回复的 ACK 包丢失的情况（在本次实验中由于 router 的设置并不会发生这种情况）client 的机制同上，但是 server 发送的 ACK 包如果丢失了，会发生 server 会收到重复的数据包（这是由于 client 以为数据包没发过去进行了超时重传），当收到重复的数据包的时候，会直接丢掉这个重复的数据包（因为已经收到过一个了）并回复对应的 ACK 包。

4. 第四种情况是失序问题，可能会发生数据包/ACK 包迟到（由于 router 的设置本次实验也不会发生这个情况）client 重传同上，但是如果收到 ACK 报文不是期待正确的 ACK 报文的话会继续等待，不做出处理。server 可能会收到重复的数据包，处理同第三种情况。

最后我画出数据传输过程中的 client（发送端）和 server（接收端）的状态机，具体如下：



4. 断开连接：四次挥手（包含超时重传）

这个四次挥手的断开连接，我是基于 TCP 协议进行设计的，实现的示意图如下：

图 4: 四次挥手

第一次挥手 数据传输完成后，先由 client 向 server 发送一个将 FIN、ACK 置 1 的报文，代表想进行断开连接，这个报文的序列号是从前面的数据报文序列号进行递增的，注意我的设计种这个条报文不设置确认号。

第二次挥手 server 收到 client 发来的第一次挥手报文后，需要回复一个 ACK 报文，这个报文我这里也会占据一个序列号（这里我设计的协议 ACK 报文都是占据序列号的，这一点与 TCP 有所区别）序列号就是在 server 维护的序列号自动向下递增 1。确认号是 client 发来报文的序列号 +1，这一点与 TCP 相同。

第三次挥手 server 完成所有的数据传输后（本次实验中在挥手前就已经完全了全部的数据传输，因为这里使用的是停等机制），server 会向 client 发送一个报文表示也可以断开连接，这条报文将 FIN、ACK 置 1，序列号还是 server 维护的序列号自动向下递增 1，这条报文不设置确认号。

第四次挥手 client 收到 server 发来的第三次挥手后，会回复一个 ACK 报文，这个报文的序列号是 client 维护的序列号自动向下递增 1（这里我设计的协议 ACK 报文都是占据序列号的，这一点与 TCP 有所区别）。确认号是 server 发来的第三次挥手报文的序列号 +1。

四次挥手的超时重传机制 这里我设计了基于 rdt3.0 协议的超时重传机制，下面分别是 client 和 server 的重传机制实现流程：

client：

1. 发送第一次挥手，在发送后接着就开始计时
2. 接收 server 发来的第二次挥手，在这个不断等待接收的过程中，如果第一次挥手超时，则会重传第一次挥手并重新开始计时。这里注意，我设置最大重传次数为 10 次，如果 10 次重传都失败则会退出。
3. 接收第三次挥手
4. 发送第四次挥手，这是一个 ACK 报文

5. 等待 2MSL，由于 client 端并不知道 server 端是否正确收到了最后一次 ACK 报文，为了防止最后一个 ACK 报文丢失，要等待 2MSL，如果再次收到了消息，则说明最后一个 ACK 丢失，会重传第四次挥手的 ACK 包，注意这里是只重传了一次，没有设置最大重传次数。等待结束后则会退出。

server：

1. 接收第一次挥手
2. 接收到第一次挥手后，发送第二次挥手，这是一个 ACK 报文。
3. 发送第三次挥手，并开始计时。
4. 接收 client 的第四次握手，在这个不断等待接收的过程中，如果第三次挥手超时，则会重传第三次挥手并重新计时。这里注意，我设置最大重传次数为 10 次，如果 10 次重传都失败则会退出。接收到第四次挥手就会退出。

5. 日志输出

输出了三次握手和四次挥手的完整过程，其中输出了发送的报文的几乎所有字段，包括但不限于序列号、确认号、标志等。

在数据传输过程中，首先在 client 端将发送的数据报文的所有字段进行输出，而且在 server 端会输出要传输的文件名字和大小，并会记录收到的数据报文。

对于重传机制的实现，在全过程，我也编写了完整的输出，可以在后文的测试中观察到。

对于可能出现的报错情况，我也都进行了完整的报错处理，

在 client 端文件传输结束后输出了总体传输时间、平均吞吐率。

(二) 核心代码实现

1. 报文格式核心代码与校验和

报文的实现是通过结构体进行实现的，用对应的字节长度定义上文协议设计中说明的不同字段，具体代码如下：

报文格式

```
struct Message
{
        //头部（一共28字节）
        //源IP、目的IP
        unsigned int SrcIP, DestIP;//4字节、4字节
        //源端口号、目的端口号
        unsigned short SrcPort, DestPort;//2字节、2字节
        //序号，这个表示相对序列号
        unsigned int SeqNum;//4字节
        //确认号
        unsigned int AckNum;//4字节
        //数据段大小
        unsigned int size;//4字节
        //标志位
        unsigned short flag;//2字节
```

```
        //校验和
        unsigned short checkNum;//2字节
        //报文数据段，实验要求一共不能超过15000字节，这里我设定不超过10000字节
        BYTE data[MaxMsgSize];
        //下面是实现的方法
        Message();
        bool check();
        void setCheck();
};
```

同时这里实现了设置校验和和检验校验和的函数，校验和计算遵循互联网校验和的标准计算方法。

设置校验和这里采用的粒度是 2 字节（16 位）。首先将校验和字段清零，然后对结构体执行端进位加法（端进位加法就是高 16 位不为 0 的话，就将其加到低 16 位，不断重复直到高 16 位为 0）。最后将计算得到的 32 位和的低 16 位取反，存储为校验和。这确保了在传输或存储过程中数据包的完整性可以被接收方验证。具体代码如下：

设置校验和

```
/*
计算并设置Message结构体的校验和。
这里采用的粒度是2字节（16位）
校验和计算遵循互联网校验和的标准计算方法，通过对结构体中每16位进行端进位加法。
首先将校验和字段清零，然后对结构体的其余部分执行端进位加法。
最后将计算得到的32位和的低16位取反，存储为校验和。
这确保了在传输或存储过程中数据包的完整性可以被接收方验证。
 */
void Message::setCheck()
{
        this->checkNum = 0;
        int sum = 0;
        unsigned short* msgStream = (unsigned short*)this;


        for (int i = 0; i < sizeof(*this) / 2; i++)
        {
                sum += *msgStream++;
                if (sum & 0xFFFF0000)
                {
                        sum &= 0xFFFF;
                        sum++;
                }
        }
        this->checkNum = ~(sum & 0xFFFF);

}
```

检验校验和这里采用的粒度也是 2 字节（16 位），对整个结构体进行端进位加法，来验证数据的完整性。如果在将所有 16 位加起来之后，计算得到的 32 位和的低 16 位全为 1，则说明校验和是正确的，数据在传输或存储过程中未发生变化，返回 true。如果低 16 位不全为 1，则校验和验证失败，返回 false。

检验校验和

```
/*
验证Message结构体的校验和是否正确。
这里采用的粒度同上是2字节（16位）
该函数通过对整个结构体包括校验和字段进行端进位加法，来验证数据的完整性。
如果在将所有16位加起来之后，计算得到的32位和的低16位全为1，
则说明校验和是正确的，数据在传输或存储过程中未发生变化，返回true。
如果低16位不全为1，则校验和验证失败，返回false。
*/
bool Message::check()
{

        unsigned int sum = 0;
        unsigned short* msgStream = (unsigned short*)this;

        for (int i = 0; i < sizeof(*this) / 2; i++)
        {
                sum += *msgStream++;
                if (sum & 0xFFFF0000)
                {
                        sum &= 0xFFFF;
                        sum++;
                }
        }
        if ((sum & 0xFFFF) == 0xFFFF)
        {
                return true;
        }
        return false;

}
```


2. 三次握手核心代码

这里具体的协议设计，包括连接协议和超时重传协议已经在上文的协议设计中进行了具体的说明，下面讲一下实现：这里的实现流程与协议设计中说明的一样，在连接的时候必须是一端等待另一端的消息，也就是流量控制使用停等协议，收发报文利用的函数主要是 UDP 中常用的 **sendto 函数**和 **recvform 函数**。具体实现代码如下：

client 端 ：

client 三次握手

```
//实现client的三次握手
bool threewayhandshake(SOCKET clientSocket, SOCKADDR_IN serverAddr)
{
        int AddrLen = sizeof(serverAddr);
        Message buffer1,buffer2,buffer3;


        //发送第一次握手的消息（SYN有效，seq=x（相对seq就是0））
        buffer1.SrcPort = ClientPORT;
        buffer1.DestPort = RouterPORT;
        buffer1.flag += SYN;
        buffer1.SeqNum = initrelseq;
        buffer1.setCheck();


        int sendByte = sendto(clientSocket, (char*)&buffer1, sizeof(buffer1),
            0, (sockaddr*)&serverAddr, AddrLen);
        clock_t buffer1start = clock();
        if (sendByte > 0)
        {
                cout << "client发送第一次握手: "
                << "源端口: " << buffer1.SrcPort << ", "
                << "目的端口: " << buffer1.DestPort << ", "
                << "序列号: " << buffer1.SeqNum << ", "
                << "标志位: " << "[SYN: " << (buffer1.flag & SYN ? "SET" : "
                    NOT SET")
                << "] [ACK: " << (buffer1.flag & ACK ? "SET" : "NOT SET")
                << "] [FIN: " << (buffer1.flag & FIN ? "SET" : "NOT SET") <<
                    "], "
                << "校验和: " << buffer1.checkNum << endl;
        }

        int resendtimes = 0;
        //接收第二次握手的消息
        while (true)
        {
                int recvByte = recvfrom(clientSocket, (char*)&buffer2, sizeof
                    (buffer2), 0, (sockaddr*)&serverAddr, &AddrLen);
                if (recvByte > 0)
                {
                        //成功收到消息，检查校验和、ACK、SYN、ack
                        if ((buffer2.flag & ACK) && (buffer2.flag & SYN) &&
                            buffer2.check() && (buffer2.AckNum == buffer1.
                            SeqNum+1))
                        {
                                cout << "client接收第二次握手成功" << endl;
                                break;
                        }
```

```
43                      else{
44                              cout<<"接收第二次握手消息检查失败"<<endl;
45                      }
46              }
47
48              //client发送第一次挥手超时, 重新发送并重新计时
49              if (clock() - buffer1start > MAX_WAIT_TIME)
50              {
51          if (++resendtimes > MAX_SEND_TIMES) {
52              cout << "client发送第一次握手超时重传已到最大次数, 发送失败"
                    << endl;
53              return false;
54          }
55                      cout << "client发送第一次握手, 第"<< resendtimes <<"
                        次超时, 正在重传....." << endl;
56                      int sendBtye = sendto(clientSocket, (char*)&buffer1,
                        sizeof(buffer1), 0, (sockaddr*)&serverAddr,
                        AddrLen);
57                      buffer1start = clock();
58          if (sendByte <= 0) {
59              cout << "client第一次握手重传失败" << endl;
60                              return false;
61          }
62              }
63      }
64
65      //发送第三次握手的消息（ACK有效, seq=x+1（相对seq就是1） ack=y+1（也就
            是1））
66      buffer3.SrcPort = ClientPORT;
67      buffer3.DestPort = RouterPORT;
68      buffer3.flag += ACK;
69      buffer3.SeqNum = ++initrelseq;
70      buffer3.AckNum = buffer2.SeqNum+1;
71      buffer3.setCheck();
72
73      sendByte = sendto(clientSocket, (char*)&buffer3, sizeof(buffer3), 0,
            (sockaddr*)&serverAddr, AddrLen);
74      clock_t buffer3start = clock();
75      if (sendByte == 0)
76      {
77              cout << "client发送第三次握手失败" << endl;
78              return false;
79      }
80      cout << "client发送第三次握手:"
81      << "源端口:␣" << buffer3.SrcPort << ",␣"
82      << "目的端口:␣" << buffer3.DestPort << ",␣"
83      << "序列号:␣" << buffer3.SeqNum << ",␣"
84      << "确认号:␣" << (buffer3.flag & ACK ? to_string(buffer3.AckNum) : "0
```

```
            ") << ",␣"
85          << "标志位:␣" << "[SYN:␣" << (buffer3.flag & SYN ? "SET" : "NOT␣SET")
86          << "]␣[ACK:␣" << (buffer3.flag & ACK ? "SET" : "NOT␣SET")
87          << "]␣[FIN:␣" << (buffer3.flag & FIN ? "SET" : "NOT␣SET") << "],␣"
88          << "校验和:␣" << buffer3.checkNum << endl;
89          cout << "client连接成功! " << endl;
90          return true;
91  }
```

server 端 ：

server 三次握手

```
1   //实现server的三次握手
2   bool threewayhandshake(SOCKET serverSocket, SOCKADDR_IN clientAddr)
3   {
4           int AddrLen = sizeof(clientAddr);
5           Message buffer1,buffer2,buffer3;
6           int resendtimes = 0;
7
8           while (true)
9           {
10                  //接收第一次握手的消息
11                  int recvByte = recvfrom(serverSocket, (char*)&buffer1, sizeof
                        (buffer1), 0, (sockaddr*)&clientAddr, &AddrLen);
12
13                  if (recvByte > 0)
14                  {
15                          //成功收到消息，检查SYN、检验和
16                          if (!(buffer1.flag & SYN) || !buffer1.check())
17                          {
18                                  cout << "server接收第一次握手成功，检查失败"
                                        << endl;
19                                  return false;
20                          }
21                          cout << "server接收第一次握手成功"<< endl;
22
23                          //发送第二次握手的消息（SYN、ACK有效，seq=y（相对seq
                                也是0）ack=x+1（也就是1））
24                          //我的协议设计双端的seq是各自维护的，并不会相互影响，
                                所以这里是y，但是相对seq还是0。
25                          buffer2.SrcPort = ServerPORT;
26                          buffer2.DestPort = RouterPORT;
27                          buffer2.SeqNum = initrelseq;
28                          buffer2.AckNum = buffer1.SeqNum+1;
29                          buffer2.flag += SYN;
30                          buffer2.flag += ACK;
31                          buffer2.setCheck();
32
```

```
33                  int sendByte = sendto(serverSocket, (char*)&buffer2,
                        sizeof(buffer2), 0, (sockaddr*)&clientAddr,
                        AddrLen);
34                  clock_t buffer2start = clock();
35
36                  if (sendByte == 0)
37                  {
38                          cout << "server发送第二次握手失败" << endl;
39                          return false;
40                  }
41                  cout << "server发送第二次握手: "
42                  << "源端口:␣" << buffer2.SrcPort << ",␣"
43                  << "目的端口:␣" << buffer2.DestPort << ",␣"
44                  << "序列号:␣" << buffer2.SeqNum << ",␣"
45                  << "确认号:␣" << (buffer2.flag & ACK ? to_string(
                        buffer2.AckNum) : "N/A") << ",␣"
46                  << "标志位:␣" << "[SYN:␣" << (buffer2.flag & SYN ? "
                        SET" : "NOT␣SET")
47                  << "]␣[ACK:␣" << (buffer2.flag & ACK ? "SET" : "NOT␣
                        SET")
48                  << "]␣[FIN:␣" << (buffer2.flag & FIN ? "SET" : "NOT␣
                        SET") << "],␣"
49                  << "校验和:␣" << buffer2.checkNum << endl;
50
51
52                  //接收第三次握手的消息
53                  while (true)
54                  {
55                          int recvByte = recvfrom(serverSocket, (char*)
                                &buffer3, sizeof(buffer3), 0, (sockaddr*)
                                &clientAddr, &AddrLen);
56                          if (recvByte > 0)
57                          {
58                                  //成功收到消息, 检查ACK、校验和、ack
59                                  if ((buffer3.flag & ACK) && buffer3.
                                        check() && (buffer3.AckNum ==
                                        buffer2.SeqNum+1))
60                                  {
61                                          initrelseq++;
62                                          cout << "server接收第三次握手
                                                成功"<< endl;
63                                          cout << "server连接成功! " <<
                                                endl;
64                                          return true;
65                                  }
66                                  else
67                                  {
68                                          cout << "server接收第三次握手
```

```
                                                成功，检查失败" << endl;
69                                          return false;
70                                      }
71                                  }
72
73                                  //server发送第二次握手超时，重新发送并重新计
                                        时
74                                  if (clock() - buffer2start > MAX_WAIT_TIME)
75                                  {
76                                      if (++resendtimes > MAX_SEND_TIMES) {
77                                          cout << "server发送第二次握手
                                                超时重传已到最大次数，发
                                                送失败" << endl;
78                                          return false;
79                                      }
80                                      cout << "server发送第二次握手，第"<<
                                            resendtimes <<"次超时，正在重传
                                            ....." << endl;
81                                      int sendByte = sendto(serverSocket, (
                                            char*)&buffer2, sizeof(buffer2),
                                            0, (sockaddr*)&clientAddr,
                                            AddrLen);
82                                      buffer2start = clock();
83                                      if(sendByte>0){
84                                          cout<<"server发送第二次握手重
                                                传成功"<<endl;
85                                      }
86                                      else{
87                                          cout<<"server第二次握手重传失
                                                败"<<endl;
88                                      }
89                                  }
90                              }
91                          }
92                  }
93                  return false;
94  }
```

3. 可靠数据传输核心代码

这里的实现流程与协议设计中的一致，这个数据传输也是停等机制，即 client 收到 ACK 包后才会继续按顺序发送后面的数据包，具体代码如下:

client 端 :

首先展示一下进行文件发送的主体函数,首先会将要发送的文件读成字节流,然后发送装载文件名和文件大小的数据报文, 然后开始遍历接收来自 client 的数据报文, 会去调用 **sendMessage 函数来发送数据报文并检查回复的 ACK 报文**这里注意基于停等机制，都是需要等待确认了回

复的 ACK 报文是正确的后才能继续发送后面的数据报文。

client 数据传输

```
//实现文件传输
void sendFileToServer(string filename, SOCKADDR_IN serverAddr, SOCKET clientSocket)
{

        int starttime = clock();
        string realname = filename;
        filename="测试文件\\"+filename;
        //将文件读成字节流
        ifstream fin(filename.c_str(), ifstream::binary);
        if (!fin) {
                printf("无法打开文件! \n");
                return;
        }

        //文件读取到fileBuffer
        BYTE* fileBuffer = new BYTE[MaxFileSize];
        unsigned int fileSize = 0;
        BYTE byte = fin.get();
        while (fin) {
                fileBuffer[fileSize++] = byte;
                byte = fin.get();
        }
        fin.close();

        //发送文件名和文件大小，文件大小放在头部的size字段了，设置了SFileName标志位
        Message nameMessage;
        nameMessage.SrcPort = ClientPORT;
        nameMessage.DestPort = RouterPORT;
        nameMessage.size = fileSize;
        nameMessage.flag += SFileName;
        nameMessage.SeqNum = ++initrelseq;
        //将文件名放在前面并加一个结束符
        for (int i = 0; i < realname.size(); i++)
                nameMessage.data[i] = realname[i];
        nameMessage.data[realname.size()] = '\0';
        nameMessage.setCheck();
        if (!sendMessage(nameMessage, clientSocket, serverAddr))
        {
                cout << "装载文件名和文件大小的报文发送失败" << endl;
                return;
        }
        cout << "装载文件名和文件大小的报文发送成功" << endl<<endl;

```

```
44          int batchNum = fileSize / MaxMsgSize;//可以装满的报文数量
45          int leftNum = fileSize % MaxMsgSize;//剩余数据量大小
46          //满载的数据报文段发送，类似批量发送，这里为了简便并没有设置标志位
47          //设计的协议：发送一个报文就需要server回复一个ACK，短小精悍。
48          //发送的数据报文并不需要这是标志位，这里是为了方便快捷
49          for (int i = 0; i < batchNum; i++)
50          {
51                  Message dataMsg;
52                  dataMsg.SrcPort = ClientPORT;
53                  dataMsg.DestPort = RouterPORT;
54                  dataMsg.SeqNum = ++initrelseq;
55                  for (int j = 0; j < MaxMsgSize; j++)
56                  {
57                          dataMsg.data[j] = fileBuffer[i * MaxMsgSize + j];
58                  }
59                  dataMsg.setCheck();
60                  if (!sendMessage(dataMsg, clientSocket, serverAddr))
61                  {
62                          cout << "满载数据报文发送失败" << endl;
63                          return;
64                  }
65                  cout << "第" << i+1 << "个满载数据报文发送成功" << endl<<endl
                        ;
66          }
67          //未满载的数据报文段发送，如果是整除就不发送这个报文了
68          if (leftNum > 0)
69          {
70                  Message dataMsg;
71                  dataMsg.SrcPort = ClientPORT;
72                  dataMsg.DestPort = RouterPORT;
73                  dataMsg.SeqNum = ++initrelseq;
74                  for (int j = 0; j < leftNum; j++)
75                  {
76                          dataMsg.data[j] = fileBuffer[batchNum * MaxMsgSize +
                                j];
77                  }
78                  dataMsg.setCheck();
79                  if (!sendMessage(dataMsg, clientSocket, serverAddr))
80                  {
81                          cout << "未满载的数据报文发送失败" << endl;
82                          return;
83                  }
84                  cout << "未满载的数据报文发送成功" << endl<<endl;
85          }
86
87          //计算传输时间和吞吐率
88          int endtime = clock();
89          cout << "\n总传输时间为:" << (endtime - starttime) << "ms" << endl;
```

```
90          cout << "平均吞吐率:" << ((float)fileSize) / (endtime - starttime)
                << "bytes/ms" << endl << endl;
91          delete[] fileBuffer;//释放内存
92          return;
93  }
```

下面展示一下 **sendMessage 函数**，具体代码如下：

sendMessage 函数

```
1   //实现单个报文发送
2   bool sendMessage(Message& sendMsg, SOCKET clientSocket, SOCKADDR_IN
        serverAddr)
3   {
4           sendto(clientSocket, (char*)&sendMsg, sizeof(sendMsg), 0, (sockaddr*)
                &serverAddr, sizeof(SOCKADDR_IN));
5           cout << "client发送: "
6           << "源端口:␣" << sendMsg.SrcPort << ",␣"
7           << "目的端口:␣" << sendMsg.DestPort << ",␣"
8           << "序列号:␣" << sendMsg.SeqNum << ",␣"
9           << "标志位:␣" << "[SYN:␣" << (sendMsg.flag & SYN ? "SET" : "NOT␣SET")
10          << "]␣[ACK:␣" << (sendMsg.flag & ACK ? "SET" : "NOT␣SET")
11          << "]␣[FIN:␣" << (sendMsg.flag & FIN ? "SET" : "NOT␣SET")
12          << "]␣[SFileName:␣" << (sendMsg.flag & SFileName ? "SET" : "NOT␣SET")
                <<"],␣"
13          << "校验和:␣" << sendMsg.checkNum << endl;
14          int msgStart = clock();
15          Message recvMsg;
16          int AddrLen = sizeof(serverAddr);
17          int resendtimes = 0;
18
19          while (1)
20          {
21                  int recvByte = recvfrom(clientSocket, (char*)&recvMsg, sizeof
                        (recvMsg), 0, (sockaddr*)&serverAddr, &AddrLen);
22                  if (recvByte > 0)
23                  {
24                          //成功收到消息，检查校验和、ack，如果不正确，则会继续
                                等待，直到发过来对的或者等到超时了重传一个过去，
                                确保一个发送必须有一个ACK回来
25                          //这一点是基于rdt3.0协议实现的，同时对ACK和ack值进行
                                检验，几号发送包就要对应几号ACK包
26                          if ((recvMsg.flag & ACK) && (recvMsg.AckNum ==
                                sendMsg.SeqNum))
27                          {
28                                  cout << "client收到: ack␣=␣" << recvMsg.
                                        AckNum << "的ACK, ";
29                                  return true;
30                          }
31                  }
```

```
32                  //基于rdt3.0协议，超时重传机制，重新发送并重新计时
33                  if (clock() - msgStart > MAX_WAIT_TIME)
34                  {
35                          cout << "seq=" <<sendMsg.SeqNum << "的报文，第" <<
                                ++resendtimes << "次超时，正在重传....." << endl
                                ;
36                          int sendByte = sendto(clientSocket, (char*)&sendMsg,
                                sizeof(sendMsg), 0, (sockaddr*)&serverAddr,
                                sizeof(SOCKADDR_IN));
37                          msgStart = clock();
38                          if(sendByte>0){
39                                  cout<<"报文重传成功"<<endl;
40                                  break;
41                          }
42                          else{
43                                  cout<<"报文重传失败"<<endl;
44                          }
45                  }
46                  if (resendtimes == MAX_SEND_TIMES)
47                  {
48                          cout << "超时重传已到最大次数，发送失败" << endl;
49                          return false;
50                  }
51          }
52          return true;
53  }
```

server 端 ：

首先展示一下进行文件接收的主体函数，首先会接收装载文件名和文件大小的数据报文，并存下来，然后开始遍历接收来自 client 的数据报文，会去调用 **recvMessage 函数**接收并回复一个 ACK 报文，最后将收到的文件进行写入，得到的文件与原来一致。

server 数据传输

```
1   //实现文件接收
2   void recvFileFormClient(SOCKET serverSocket, SOCKADDR_IN clientAddr)
3   {
4           int AddrLen = sizeof(clientAddr);
5           //接收文件名和文件大小
6           Message nameMessage;
7           unsigned int fileSize;
8           char fileName[50] = { 0 };
9           while (1)
10          {
11                  int recvByte = recvfrom(serverSocket, (char*)&nameMessage,
                        sizeof(nameMessage), 0, (sockaddr*)&clientAddr, &AddrLen)
                        ;
12                  if (recvByte > 0)
```

```
13          {
14              //如果成功收到装载文件名和文件大小的消息，则进行一下输出并回复对应的ACK报文
15              //这里我设计的是基于rdt3.0协议，将ack值设为发来报文的seq值
16              if (nameMessage.check() && (nameMessage.SeqNum == initrelseq + 1) && (nameMessage.flag & SFileName))
17              {
18                  fileSize = nameMessage.size;
19                  for (int i = 0; nameMessage.data[i]; i++)
20                      fileName[i] = nameMessage.data[i];
21                  cout << "\n接收文件名: " << fileName << ", 文件大小: " << fileSize << endl<<endl;
22
23                  Message replyMessage;
24                  replyMessage.SrcPort = ServerPORT;
25                  replyMessage.DestPort = RouterPORT;
26                  replyMessage.flag += ACK;
27                  replyMessage.SeqNum=initrelseq++;
28                  replyMessage.AckNum = nameMessage.SeqNum;
29                  replyMessage.setCheck();
30                  sendto(serverSocket, (char*)&replyMessage, sizeof(replyMessage), 0, (sockaddr*)&clientAddr, sizeof(SOCKADDR_IN));
31                  cout << "server收到: seq␣=␣" << nameMessage.SeqNum << "的数据报文"<<endl;
32                  cout << "server发送: seq␣=␣"<<replyMessage.SeqNum<<", ack␣=␣" << replyMessage.AckNum << "␣的ACK报文" << endl<<"装载文件名和文件大小的报文接收成功，下面开始正式传送数据段"<< endl<< endl;
33                  break;
34              }
35
36              //如果seq值不正确，说明接收到重复的报文，处理如下:
37              //丢弃重复报文，重传该报文的回复ACK，实现就是只回复一个ack值对应接收消息的seq值的ACK报文
38              //这种情况是因为上一个ACK报文丢失，从而使client超时重传
39              else if (nameMessage.check() && (nameMessage.SeqNum != initrelseq + 1) && (nameMessage.flag & SFileName))
40              {
41                  Message replyMessage;
42                  replyMessage.SrcPort = ServerPORT;
43                  replyMessage.DestPort = RouterPORT;
44                  replyMessage.flag += ACK;
```

```
45                                  replyMessage.SeqNum=initrelseq++;
46                                  replyMessage.AckNum = nameMessage.SeqNum;
47                                  replyMessage.setCheck();
48                                  sendto(serverSocket, (char*)&replyMessage,
                                        sizeof(replyMessage), 0, (sockaddr*)&
                                        clientAddr, sizeof(SOCKADDR_IN));
49                                  cout << "[重复接收报文段]server收到 seq =" 
                                        << nameMessage.SeqNum << "的数据报文, 并
                                        发送 ack =" << replyMessage.AckNum << " 
                                        的ACK报文" << endl;
50                          }
51                  }
52          }
53
54
55          int batchNum = fileSize / MaxMsgSize;//可以装满的报文数量
56          int leftNum = fileSize % MaxMsgSize;//剩余数据量大小
57          BYTE* fileBuffer = new BYTE[fileSize];
58
59          //满载的数据报文段接收
60          for (int i = 0; i < batchNum; i++)
61          {
62                  Message dataMsg;
63                  if (recvMessage(dataMsg, serverSocket, clientAddr))
64                  {
65                          cout << "第" << i+1 << "个满载数据报文接收成功" <<
                                endl<< endl;
66                  }
67                  else
68                  {
69                          cout << "第" << i+1 << "个满载数据报文接收失败" <<
                                endl<< endl;
70                          return;
71                  }
72                  //读取数据
73                  for (int j = 0; j < MaxMsgSize; j++)
74                  {
75                          fileBuffer[i * MaxMsgSize + j] = dataMsg.data[j];
76                  }
77          }
78
79          //未满载的数据报文段接收, 如果是整除就不接收这个报文了
80          if (leftNum > 0)
81          {
82                  Message dataMsg;
83                  if (recvMessage(dataMsg, serverSocket, clientAddr))
84                  {
85                          cout << "未满载的数据报文接收成功" << endl<< endl;
```

```
86                  }
87                  else
88                  {
89                          cout << "未满载的数据报文接收失败" << endl<< endl;
90                          return;
91                  }
92                  //读取数据
93                  for (int j = 0; j < leftNum; j++)
94                  {
95                          fileBuffer[batchNum * MaxMsgSize + j] = dataMsg.data[
                                j];
96                  }
97          }
98
99          //写入文件
100         cout << "\n文件传输成功, 开始写入文件" << endl;
101         FILE* outputfile;
102         outputfile = fopen(fileName, "wb");
103         if (fileBuffer != 0)
104         {
105                 fwrite(fileBuffer, fileSize, 1, outputfile);
106                 fclose(outputfile);
107         }
108         cout << "文件写入成功" << endl<<endl;
109         delete[] fileBuffer;//释放内存
110 }
```

下面展示一下 **recvMessage 函数**，具体代码如下：

recvMessage 函数

```
1  //实现单个报文接收
2  bool recvMessage(Message& recvMsg, SOCKET serverSocket, SOCKADDR_IN
       clientAddr)
3  {
4          int AddrLen = sizeof(clientAddr);
5          while (1)
6          {
7                  int recvByte = recvfrom(serverSocket, (char*)&recvMsg, sizeof
                       (recvMsg), 0, (sockaddr*)&clientAddr, &AddrLen);
8                  if (recvByte > 0)
9                  {
10                         //成功收到消息, 回复ACK报文
11                         if (recvMsg.check() && (recvMsg.SeqNum == initrelseq
                               + 1))
12                         {
13                                 Message replyMessage;
14                                 replyMessage.SrcPort = ServerPORT;
15                                 replyMessage.DestPort = RouterPORT;
16                                 replyMessage.flag += ACK;
```

```
17                                  replyMessage.SeqNum=initrelseq++;
18                                  replyMessage.AckNum = recvMsg.SeqNum;
19                                  replyMessage.setCheck();
20                                  sendto(serverSocket, (char*)&replyMessage,
                                        sizeof(replyMessage), 0, (sockaddr*)&
                                        clientAddr, sizeof(SOCKADDR_IN));
21                                  cout << "server收到: seq=" << recvMsg.
                                        SeqNum << "的数据报文"<< endl;
22                                  cout << "server发送: seq="<<replyMessage.
                                        SeqNum<<", ack=" << replyMessage.AckNum
                                         << "的ACK报文" << endl;
23                                  return true;
24                              }
25
26                              //如果seq值不正确, 说明接收到重复的报文, 处理如下:
27                              //丢弃重复报文, 重传该报文的回复ACK, 实现就是只回复一
                                    个ack值对应接收消息的seq值的ACK报文
28                              //这种情况是因为上一个ACK报文丢失, 从而使client超时重
                                    传
29                              else if (recvMsg.check() && (recvMsg.SeqNum !=
                                    initrelseq + 1))
30                              {
31                                  Message replyMessage;
32                                  replyMessage.SrcPort = ServerPORT;
33                                  replyMessage.DestPort = RouterPORT;
34                                  replyMessage.flag += ACK;
35                                  replyMessage.SeqNum = initrelseq;
36                                  replyMessage.AckNum = recvMsg.SeqNum;
37                                  replyMessage.setCheck();
38                                  sendto(serverSocket, (char*)&replyMessage,
                                        sizeof(replyMessage), 0, (sockaddr*)&
                                        clientAddr, sizeof(SOCKADDR_IN));
39                                  cout << "[重复接收报文]server收到seq=" <<
                                         recvMsg.SeqNum << "的数据报文, 并发送
                                        ack=" << replyMessage.AckNum << "的ACK
                                        报文" << endl;
40                              }
41                          }
42                          else if (recvByte == 0)
43                          {
44                              return false;
45                          }
46                      }
47                      return true;
48  }
```

4. 四次挥手核心代码

这里具体的协议设计，包括断开连接协议和超时重传协议已经在上文的协议设计中进行了具体的说明，下面讲一下实现：这里的实现流程与协议设计中说明的一样，在连接的时候必须是一端等待另一端的消息，也就是流量控制使用停等协议，收发报文利用的函数主要是 UDP 中常用的 **sendto 函数**和 **recvform 函数**。具体实现代码如下：

client 端 ：

client 四次挥手

```
//实现client的四次挥手
bool fourwayhandwave(SOCKET clientSocket, SOCKADDR_IN serverAddr)
{
        int AddrLen = sizeof(serverAddr);
        Message buffer1;
        Message buffer2;
        Message buffer3;
        Message buffer4;

        //发送第一次挥手（FIN、ACK有效，seq是之前的发送完数据包后的序列号，之前的序列号每次+1
        //（对于0延时和0丢包的1.jpg文件的发送来说，seq这里已经到了189）
        buffer1.SrcPort = ClientPORT;
        buffer1.DestPort = RouterPORT;
        buffer1.flag += FIN;
        buffer1.flag += ACK;
        buffer1.SeqNum = ++initrelseq;
        buffer1.setCheck();
        int sendByte = sendto(clientSocket, (char*)&buffer1, sizeof(buffer1), 0, (sockaddr*)&serverAddr, AddrLen);
        clock_t buffer1start = clock();
        if (sendByte == 0)
        {
                cout << "client发送第一次挥手失败，退出" << endl;
                return false;
        }
        cout << "client发送第一次挥手: "
        << "源端口:␣" << buffer1.SrcPort << ",␣"
        << "目的端口:␣" << buffer1.DestPort << ",␣"
        << "序列号:␣" << buffer1.SeqNum << ",␣"
        << "标志位:␣" << "[SYN:␣" << (buffer1.flag & SYN ? "SET" : "NOT␣SET")
        << "]␣[ACK:␣" << (buffer1.flag & ACK ? "SET" : "NOT␣SET")
        << "]␣[FIN:␣" << (buffer1.flag & FIN ? "SET" : "NOT␣SET")
        << "]␣[SFileName:␣" << (buffer1.flag & SFileName ? "SET" : "NOT␣SET")
        <<"],␣"
        << "校验和:␣" << buffer1.checkNum << endl;
        int resendtimes=0;
        //接收第二次挥手的消息
```

```
36          while (1)
37          {
38                  int recvByte = recvfrom(clientSocket, (char*)&buffer2, sizeof
                        (buffer2), 0, (sockaddr*)&serverAddr, &AddrLen);
39                  if (recvByte == 0)
40                  {
41                          cout << "client第二次挥手接收失败" << endl;
42                          return false;
43                  }
44                  else if (recvByte > 0)
45                  {
46                          //成功收到消息，检查校验和、ACK、ack
47                          if ((buffer2.flag & ACK) && buffer2.check() && (
                                buffer2.AckNum == buffer1.SeqNum+1))
48                          {
49                                  cout << "client接收第二次挥手成功" << endl;
50                                  break;
51                          }
52                          else
53                          {
54                                  //cout << "client第二次挥手接收成功，检查失败
                                        " << endl;
55                                  continue;
56                          }
57                  }
58                  //client发送第一次挥手超时，重新发送并重新计时
59                  if (clock() - buffer1start > MAX_WAIT_TIME)
60                  {
61                          cout << "client发送第一次挥手，第"<< ++resendtimes <<
                                "次超时，正在重传....." << endl;
62                          int sendByte = sendto(clientSocket, (char*)&buffer1,
                                sizeof(buffer1), 0, (sockaddr*)&serverAddr,
                                AddrLen);
63                          buffer1start = clock();
64                          if(sendByte>0){
65                                  cout<<"client发送第一次挥手重传成功"<<endl;
66                                  break;
67                          }
68                          else{
69                                  cout<<"client发送第一次挥手重传失败"<<endl;
70                          }
71                  }
72                  if (resendtimes == MAX_SEND_TIMES)
73          {
74                          cout << "client发送第一次挥手超时重传已到最大次数，发
                                送失败" << endl;
75                          return false;
76          }
```

```
77          }
78
79          //接收第三次挥手的消息
80          while (1)
81          {
82                  int recvByte = recvfrom(clientSocket, (char*)&buffer3, sizeof
                        (buffer3), 0, (sockaddr*)&serverAddr, &AddrLen);
83                  if (recvByte == 0)
84                  {
85                          cout << "client接收第三次挥手失败" << endl;
86                          return false;
87                  }
88                  else if (recvByte > 0)
89                  {
90                          //收到消息, 检查校验和、FIN、ACK
91                          if ((buffer3.flag & ACK)&& (buffer3.flag & FIN) &&
                                buffer3.check())
92                          {
93                                  cout << "client接收第三次挥手成功" << endl;
94                                  break;
95                          }
96                          else
97                          {
98                                  //cout << "client接收第三次挥手失败" << endl;
99                                  continue;
100                         }
101                 }
102         }
103
104
105
106         //发送第四次挥手的消息（ACK有效, ack等于第三次挥手消息的seq+1, seq自
                动向下递增）
107         buffer4.SrcPort = ClientPORT;
108         buffer4.DestPort = RouterPORT;
109         buffer4.flag += ACK;
110         buffer4.SeqNum=++initrelseq;
111         buffer4.AckNum = buffer3.SeqNum+1;
112         buffer4.setCheck();
113         sendByte = sendto(clientSocket, (char*)&buffer4, sizeof(buffer4), 0,
                (sockaddr*)&serverAddr, AddrLen);
114         if (sendByte == 0)
115         {
116                 cout << "client发送第四次挥手失败" << endl;
117                 return false;
118         }
119         cout << "client发送第四次挥手: "
120         << "源端口:␣" << buffer4.SrcPort << ",␣"
```

```
        << "目的端口:␣" << buffer4.DestPort << ",␣"
        << "序列号:␣" << buffer4.SeqNum << ",␣"
        << "确认号:␣" << (buffer4.flag & ACK ? to_string(buffer4.AckNum) : "N/A") << ",␣"
        << "标志位:␣" << "[SYN:␣" << (buffer4.flag & SYN ? "SET" : "NOT␣SET")
        << "]␣[ACK:␣" << (buffer4.flag & ACK ? "SET" : "NOT␣SET")
        << "]␣[FIN:␣" << (buffer4.flag & FIN ? "SET" : "NOT␣SET")
        << "]␣[SFileName:␣" << (buffer4.flag & SFileName ? "SET" : "NOT␣SET") <<"],␣"
        << "校验和:␣" << buffer4.checkNum << endl;



        //第四次挥手之后还需等待2MSL，防止最后一个ACK丢失
        //此时client处于TIME_WAIT状态
        int tempclock = clock();
        cout << "client正处于2MSL的等待时间" << endl;
        Message tmp;
        while (clock() - tempclock < 2 * MAX_WAIT_TIME)
        {
                int recvByte = recvfrom(clientSocket, (char*)&tmp, sizeof(tmp), 0, (sockaddr*)&serverAddr, &AddrLen);
                if (recvByte == 0)
                {
                        cout << "TIME_WAIT状态时收到错误消息，退出" << endl;
                        return false;
                }
                else if (recvByte > 0)
                {
                        sendByte = sendto(clientSocket, (char*)&buffer4, sizeof(buffer4), 0, (sockaddr*)&serverAddr, AddrLen);
                        cout << "TIME_WAIT状态时发现最后一个ACK丢失，重发" << endl;
                }
        }
        cout << "\nclient关闭连接成功! " << endl;
        return true;
}
```

server 端 ：

server 四次挥手

```
//实现server的四次挥手
bool fourwayhandwave(SOCKET serverSocket, SOCKADDR_IN clientAddr)
{
        int AddrLen = sizeof(clientAddr);
```

```
        Message buffer1;
        Message buffer2;
        Message buffer3;
        Message buffer4;
        while (1)
        {
                //接收第一次挥手的消息
                int recvByte = recvfrom(serverSocket, (char*)&buffer1, sizeof
                    (buffer1), 0, (sockaddr*)&clientAddr, &AddrLen);
                if (recvByte == 0)
                {
                        cout << "第一次挥手接收失败, 退出" << endl;
                        return false;
                }

                else if (recvByte > 0)
                {
                        //检查FIN、ACK、检验和
                        if (!(buffer1.flag & FIN) || !(buffer1.flag & ACK) ||
                             !buffer1.check())
                        {
                                cout << "第一次挥手接收成功, 检查失败" <<
                                    endl;
                                return false;
                        }
                        cout << "server接收第一次挥手成功" << endl;

                        //发送第二次挥手的消息（ACK有效, ack是第一次挥手消息
                            的seq+1,seq值自动向下递增）
                        buffer2.SrcPort = ServerPORT;
                        buffer2.DestPort = RouterPORT;
                        buffer2.SeqNum = initrelseq++;
                        buffer2.AckNum = buffer1.SeqNum+1;
                        buffer2.flag += ACK;
                        buffer2.setCheck();//设置校验和
                        int sendByte = sendto(serverSocket, (char*)&buffer2,
                            sizeof(buffer2), 0, (sockaddr*)&clientAddr,
                            AddrLen);
                        clock_t buffer2start = clock();
                        if (sendByte == 0)
                        {
                                cout << "server发送第二次挥手失败" << endl;
                                return false;
                        }
                        cout << "server发送第二次挥手: "
                        << "源端口:␣" << buffer2.SrcPort << ",␣"
                        << "目的端口:␣" << buffer2.DestPort << ",␣"
                        << "序列号:␣" << buffer2.SeqNum << ",␣"
```

```
47                      << "确认号:␣" << (buffer2.flag & ACK ? to_string(
                            buffer2.AckNum) : "N/A") << ",␣"
48                      << "标志位:␣" << "[SYN:␣" << (buffer2.flag & SYN ? "
                            SET" : "NOT␣SET")
49                      << "]␣[ACK:␣" << (buffer2.flag & ACK ? "SET" : "NOT␣
                            SET")
50                      << "]␣[FIN:␣" << (buffer2.flag & FIN ? "SET" : "NOT␣
                            SET")
51                      << "]␣[SFileName:␣" << (buffer2.flag & SFileName ? "
                            SET" : "NOT␣SET") <<"],␣"
52                      << "校验和:␣" << buffer2.checkNum << endl;
53                      break;
54              }
55      }
56
57      //server发送第三次挥手的消息（FIN、ACK有效，seq自动向下递增）
58      buffer3.SrcPort = ServerPORT;
59      buffer3.DestPort = RouterPORT;
60      buffer3.flag += FIN;//设置FIN
61      buffer3.flag += ACK;//设置ACK
62      buffer3.SeqNum = initrelseq++;//设置序号seq
63      buffer3.setCheck();//设置校验和
64      int sendByte = sendto(serverSocket, (char*)&buffer3, sizeof(buffer3),
            0, (sockaddr*)&clientAddr, AddrLen);
65      clock_t buffer3start = clock();
66      if (sendByte == 0)
67      {
68              cout << "server发送第三次挥手失败" << endl;
69              return false;
70      }
71      cout << "server发送第三次挥手: "
72      << "源端口:␣" << buffer3.SrcPort << ",␣"
73      << "目的端口:␣" << buffer3.DestPort << ",␣"
74      << "序列号:␣" << buffer3.SeqNum << ",␣"
75      << "标志位:␣" << "[SYN:␣" << (buffer3.flag & SYN ? "SET" : "NOT␣SET")
76      << "]␣[ACK:␣" << (buffer3.flag & ACK ? "SET" : "NOT␣SET")
77      << "]␣[FIN:␣" << (buffer3.flag & FIN ? "SET" : "NOT␣SET")
78      << "]␣[SFileName:␣" << (buffer3.flag & SFileName ? "SET" : "NOT␣SET")
            <<"],␣"
79      << "校验和:␣" << buffer3.checkNum << endl;
80      int resendtimes=0;
81
82      //接收第四次挥手的消息
83      while (1)
84      {
85              int recvByte = recvfrom(serverSocket, (char*)&buffer4, sizeof
                    (buffer4), 0, (sockaddr*)&clientAddr, &AddrLen);
86              if (recvByte == 0)
```

```
87                  {
88                          cout << "server接收第四次挥手失败" << endl;
89                          return false;
90                  }
91                  else if (recvByte > 0)
92                  {
93                          //成功收到消息，检查校验和、ACK、ack
94                          if ((buffer4.flag & ACK) && buffer4.check() && (
                                buffer4.AckNum == buffer3.SeqNum+1))
95                          {
96                                  cout << "server接收第四次挥手成功" << endl;
97                                  break;
98                          }
99                          else
100                         {
101                                 cout << "server接收第四次挥手成功，检查失败"
                                        << endl;
102                                 return false;
103                         }
104                 }
105                 //server发送第三次挥手超时，重新发送并重新计时
106                 if (clock() - buffer3start > MAX_WAIT_TIME)
107                 {
108                         cout << "server发送第三次挥手，第"<< ++resendtimes <<
                            "次超时，正在重传....." << endl;
109                         int sendByte = sendto(serverSocket, (char*)&buffer3,
                            sizeof(buffer3), 0, (sockaddr*)&clientAddr,
                            AddrLen);
110                         buffer3start = clock();
111                         if(sendByte>0){
112                                 cout<<"server发送第三次挥手重传成功"<<endl;
113                                 //break;
114                                 continue;
115                         }
116                         else{
117                                 cout<<"server发送第三次挥手重传失败"<<endl;
118                         }
119                 }
120                 if (resendtimes == MAX_SEND_TIMES)
121         {
122                         cout << "server发送第三次挥手超时重传已到最大次数，发
                            送失败" << endl;
123                         return false;
124         }
125         }
126         cout << "\nserver关闭连接成功！" << endl;
127         return true;
128 }
```

(三) 测试

1. makefile 编写

这里简单的编写了一个 makefile 来方便编译和运行，具体代码为了节省篇幅就不复制到报告中了，我已经放在了提交作业的压缩包中。

2. 开启测试

首先按照 router 使用手册设置好参数，地址使用的是本地回环地址，router 端口号是 30000，服务器端口号是 40460，丢包率设置为 5%，延时设置为 10ms。

然后同时开启两个命令行，前后分别输入 **make server** 和 **make client** 命令，结果如下：

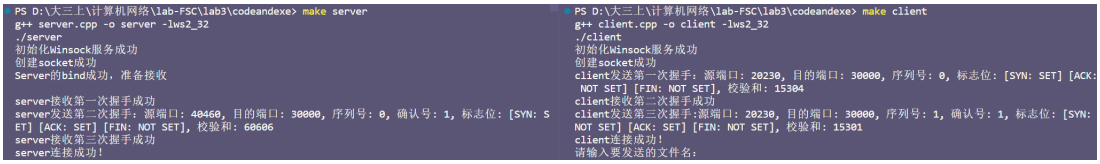


图 5: 三次握手建立连接

图中展示了三次握手建立连接的过程，可以看到连接是正确无误的，经过我的检查输出中的序列号和确认号等与协议设计中一致。然后这里我选择发送 1.jpg 文件，然后结果如下：

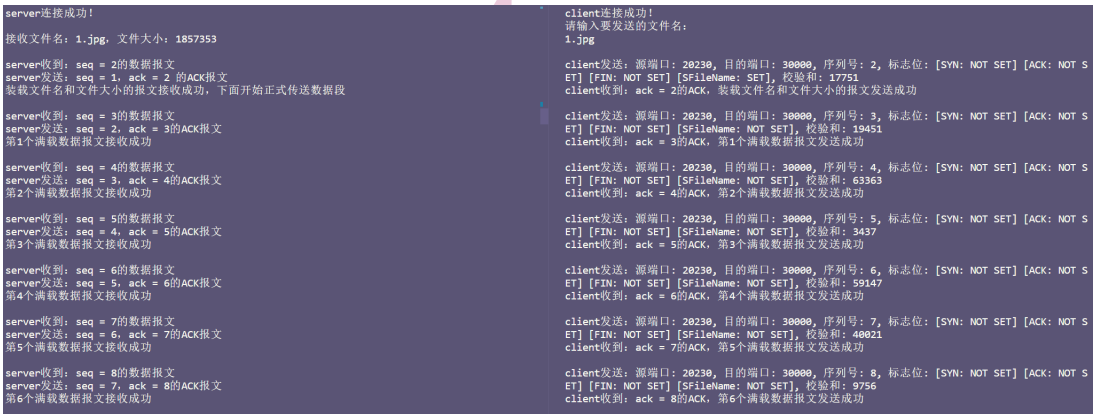


图 6: 数据传输输出

经过检查，输出与协议设计中的一致，正确无误，接下来，我们来看一下如果产生了丢包，重传机制是否可以正确执行，如图：

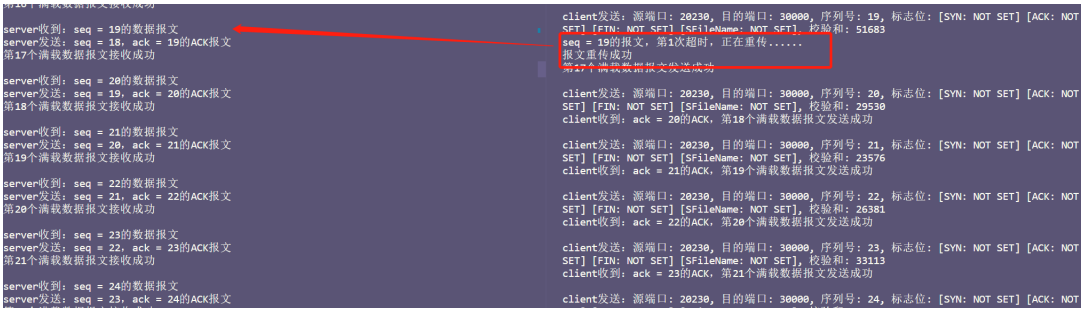


图 7: 重传输出日志

图 8: router 日志

可以看到发生了丢包也可以正确重传，这里设置丢包率为 5%，所以这里会每 count 到 19 就发生一次丢包，这里 router 日志也输出正确。延时输出也均正确。

接下来发送文件流程如上，然后到了关闭连接的流程，结果如下：

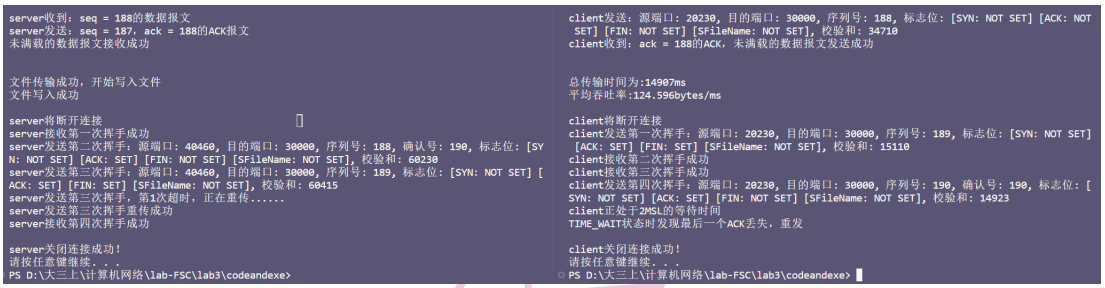


图 9: 四次挥手输出

这里发生了一种意外情况，就是 client 发送的最后一个 ACK 报文丢失了，这是一种比较特殊的情况，但是可以看到我的输出正确的处理的这种情况，由于最后一个 ACK 丢失，所以会导致 server 没收到这个 ACK 报文，所以会进行超时重传，然后这次重传 client 回复的 ACK 报文正确地被 server 接收到了，从而实现了正确的断开连接的流程。说明我的协议设计的健壮性。其余信息经过检查均与协议中的设计完全相符，正确无误。另外可以看到 client 端输出了总传输时间和平均吞吐率。

接下来我依次测试了 2.jpg、3.jpg 和 helloworld.txt 测试文件，均可以正确实现传输，同时这里的输出的文件名字和文件大小与原来完全一致，下面由于篇幅限制，仅展示 1.jpg 的对比图，如下：（红框表示传输后的文件，绿框表示源文件）

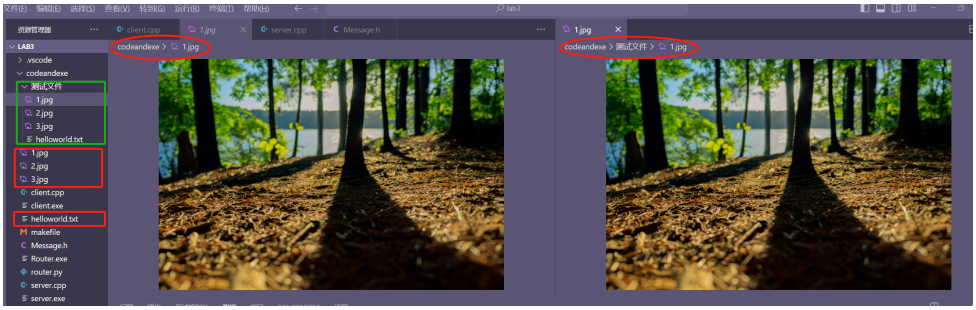


图 10: 1.jpg 图片传输前后对比

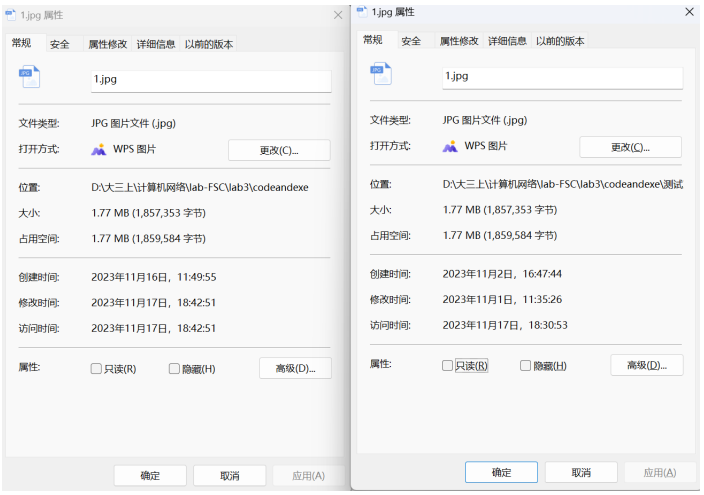


图 11: 1.jpg 图片传输前后属性对比

3. 性能测试分析

首先展示一下在 5% 丢包率和 10ms 延时，以及我将超时重传时间设为 1000ms 下 4 个测试文件对应的总传输时间和平均吞吐率，这是一种纵向对比，结果如下:

Time (ms)	Rate (bytes/ms)	Name
14907	124.596	1.jpg
51849	113.763	2.jpg
106726	112.147	3.jpg
15479	106.971	helloworld.txt

表 3: 测试文件的总传输时间和平均吞吐率

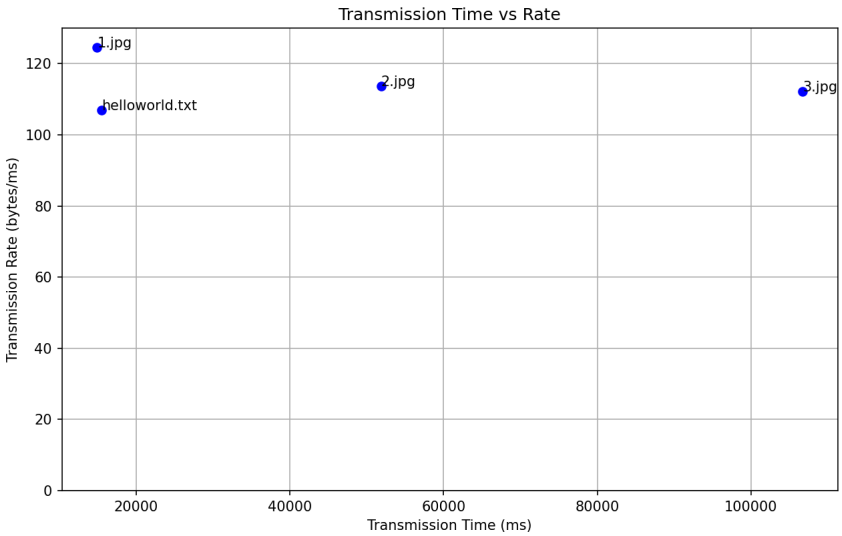


图 12: 性能对比测试结果 1

可以看到这里参数设置的一致的情况下，可以看到平均吞吐率是相近的，但是总传输时间会

和文件大小成正相关。

然后我又测试了对于同一个文件仅调整延时的结果，由于篇幅限制我不进行图的展示，只是说明一下结果，我发现如果只是对延时较小调整，只看不出来效果的，这是由于每次发送的时间都是有一些波动的，所以这个值测量的是不准的。后来发现其实这个延时还和你设置的最大超时时间有关，每次丢包都会让传输时间增加一个你的最大超时时间。所以从这一点可以看出来如果你丢包率设置的很高的话，也会让传输速度变慢。

另一方面，由于本次实验利用的 router 程序是一个 MFC 程序，会有一定的延时，所以其实测试的结果也会有较大的误差。

4. 额外测试

这里我要进行一些验证程序高健壮性的测试，这里我将丢包率设置为 25%，延时设置为 50ms，测试 1.jpg 的传输，结果如下：

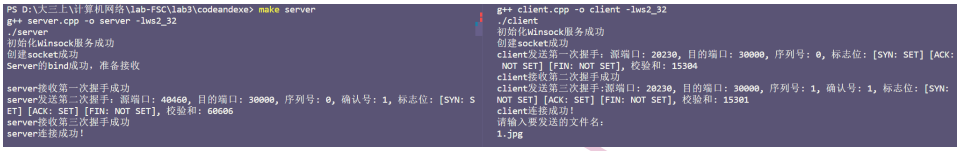


图 13: 三次握手

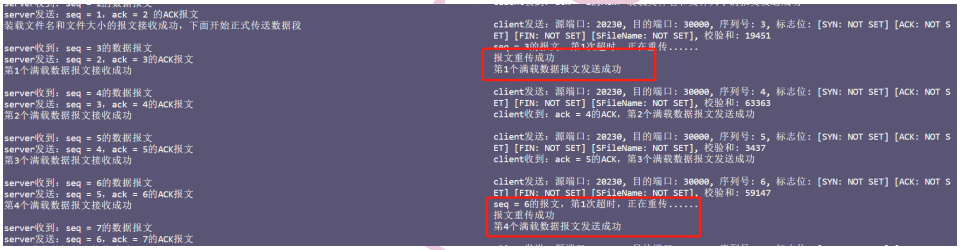


图 14: 重传机制

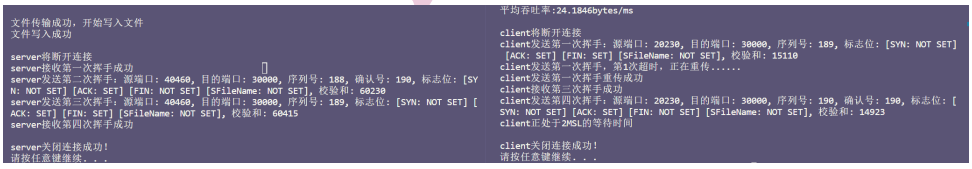


图 15: 四次挥手

图 16: router 的输出

经过检查发现，实现是完全正确的。可以应对高丢包率和高延时的情况，而且也体现了一些意外情况的正确处理。

三、 总结与问题分析

这次实验在编码方面难度较高，主要遇到的困难是考虑到各种的意外情况并进行逻辑上的debug，这一点耗费了我很长时间，其他的问题最后都解决了，唯一我觉得还存在疑惑的点就是三次握手中如果发现 client 发送的第三次握手丢失了这种意外情况。这时候 client 会认为自己已经连接上了，但是 server 一直没收到 ACK 报文，会不断的重传，直到重传到最大重传次数后退出程序，这是一个 bug，而且是理论上的 bug，但是我后来一想由于本次实验是单向传输，为了方便我完全可以直接设计一个两次握手的协议，client 发送一个 SYN，然后 server 返回一个 ACK，这样就避免了这个 bug，但是这样也导致只能单向传输而不能双向，后来考虑到后面的 3-x 实验要实现的更加完善，我没有采用这种简便的两次握手，而是沿用基于 TCP 协议的三次握手。

还有一个问题就是设置非阻塞的问题，需要在初始化 socket 的时候将其设置为非阻塞，由于 recvfrom 函数是阻塞的，设置非阻塞后就可以在 while 循环里等候消息的同时判断是否延时，而非一直被阻塞到接收消息的地方。

另一方面，我还测试了**非常高时延**的情况，例如将时延设置为 1000+，程序可能会出现问题，这是由于一些旧的或者重传的包在不正确的时候发送到对方，会导致状态机的混乱。但是实际上这也是正常现象，数据包传输受到多个变量的影响。因素如路由器的处理能力、网络的拥挤程度和物理距离都会影响数据包的传输时间。由于这些因素都有不确定性，网络中的延迟也因此变得难以预测且频繁变动。这种不确定性可能导致数据包在到达目标时出现延迟，进而可能对接收端的缓冲区造成损害，引发一些意外状况。所以现在我们也在计算机网络领域做进一步的设计，增强鲁棒性。

在这个实验过程，我对 socket 编程更加的熟悉，使用起来也更加的得心应手。同时对于协议设计有了更加深刻的认识和了解。

整个过程锻炼了我的分析能力也磨练了我的耐心，可以说是非常不错的一次练习在不可靠信道上设计可靠数据传输协议设计的实验。