计算机网络课程实验报告

Lab 3-3 基于UDP服务设计可靠传输协议并编程实现（3-3）

姓名：彭钰钊　学号：2110756　专业：计算机科学与技术

一、前期准备

上一次实验中我们实现了**GBN协议**，但是存在一些会影响性能的情况——单个分组的差错就会使得我们重传整个窗口内的已发送分组，事实上并不需要这样做。因此，此次实验中我们将实现**选择重传（SR）协议**。

选择重传协议的核心是个别的、按需重传要求接收方逐个地确认正确接收的分组。

二、协议设计

（一）报文格式

我们根据UDP数据报报文格式设计我们的协议数据报报文格式如下：

首部

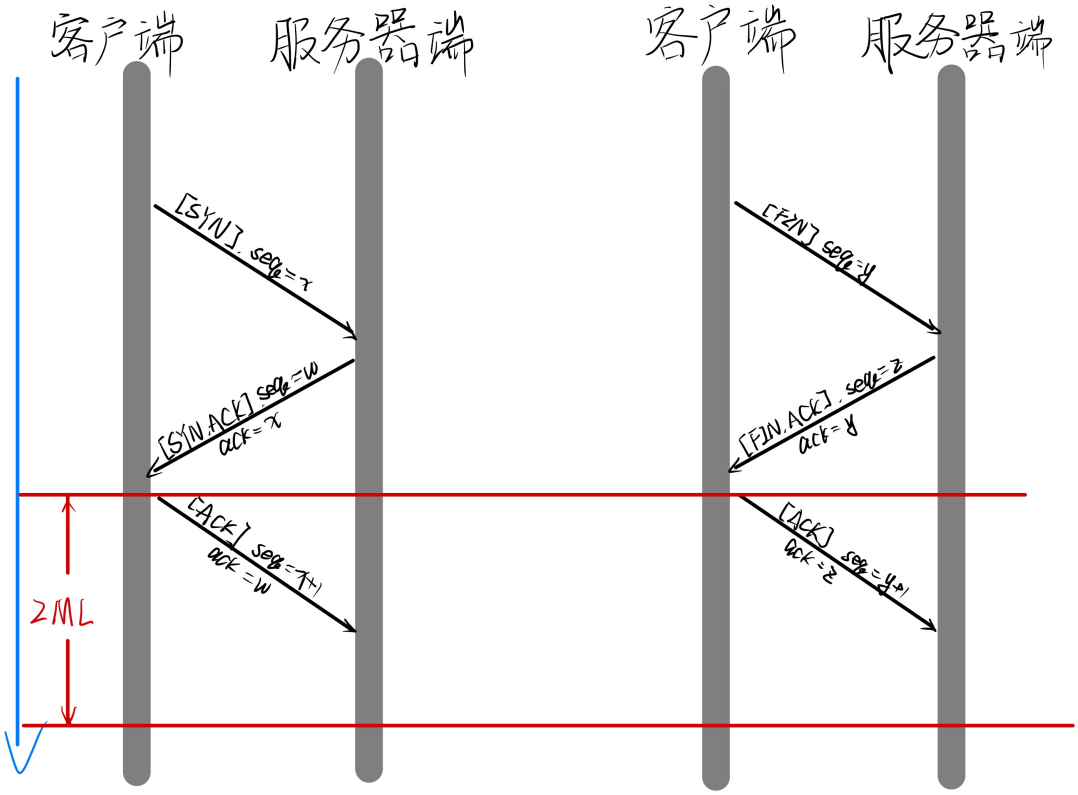
- 源ip地址：4字节
- 目的ip地址：4字节（由于我们的实验中是在本地回环中进行数据传输，所以以上两个字段实际上是不需要的）
- 源端口号：2字节
- 目的端口号：2字节
- 发送序列号：4字节
- 确认序列号：4字节
- 数据大小：4字节

- 校验和：2字节
- 标志位：2字节

数据

其余MAX_MSG_SIZE大小的字节是数据部分

（二）建立连接&断开连接



本次实验与3-1采用类似的建立连接和断开连接方式——三次握手&三次挥手，因此不再详细赘述，值得注意的是我们在建立连接的时候采用了等待2MSL的方式防止第三次ACK丢失。

（三）数据传输

滑动窗口

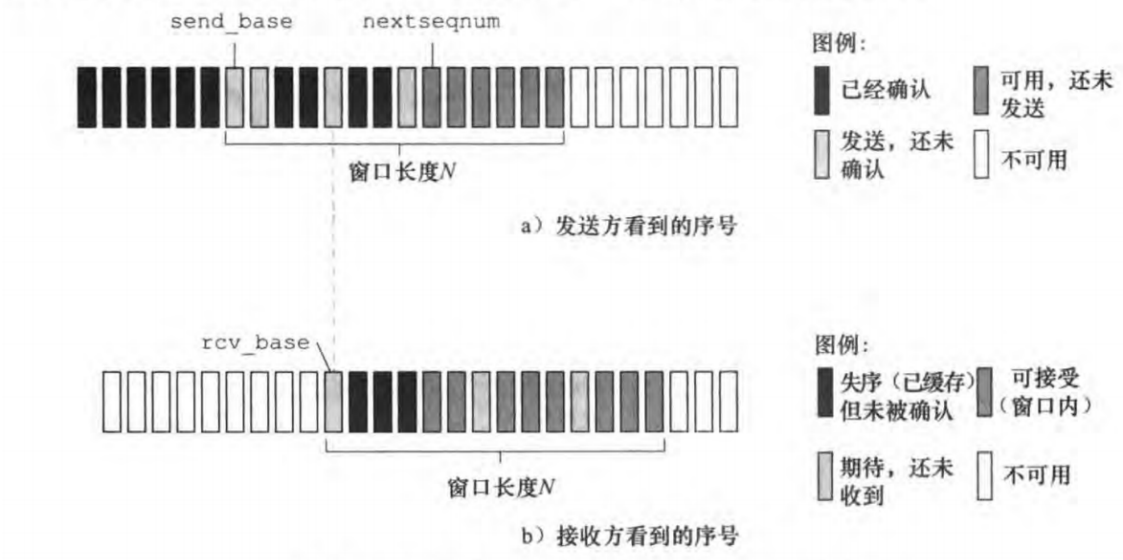


图 3-23　选择重传（SR）发送方与接收方的序号空间

本次实验中发送窗口和接收窗口大小均为**N**。

在**发送窗口**中设置两个标记**base**和**nextseqnum**，分别代表滑动窗口基序号（滑动窗口开始的位置，指向窗口初始位置，随着发送过程指向已发送但还未确认的第一个序号）和下一个序号。

同时，在**接收窗口**中设置标记**base**，其含义和发送窗口中的一致。

选择重传

发送方

- **超时重传**

  为每一个数据报设置计时器，当超时未收到报文确认信息时，会重传当前报文，此时窗口基序号 ***base*** 和 下一个序号 ***nextseqnum*** 均不发生变化。

- **收到ACK**
  - 如果收到ACK且该数据报在窗口范围内，则标记该数据报已接收。
  - 如果收到ACK且该数据报正好是窗口起始位置，即***ack = base***，则将窗口基序号 ***base*** 移动至具有最小序号的未确认数据报处。
  - 当窗口内存在可发送的数据报时，发送这些数据报。

接收方

- 接收数据报序号在 ***[base, base + N - 1]***，即**窗口内的数据报被正确接收**
  - 接收数据报序号恰好是窗口起始位置，则将窗口基序号 ***base*** 移动至窗口内具有最小序号的未接收数据报处；同时缓存该数据报并回复ACK数据报。
  - 当该数据报**未被接收过**时，缓存该数据报并回复ACK数据报。
- 接收数据报序号在 ***[base -N, base - 1]***，即**上一个窗口内的数据报被正确接收**

  回复ACK数据报（此前已确认并缓存）。

- **其他情况**

  忽略该数据报。

（四）程序使用

客户端控制是否连接：

- 当客户端接收符号为**q/Q**时断开连接
- 当客户端接受符号**r/R**时建立连接准备传输文件

三、功能模块实现与分析

（一）报文格式、建立连接&断开连接

对于标志位为了方便程序编写，我们在头文件`MessageFormat.h`中使用全局常量的方式设置；并对于一些关于文件传输的常量采取宏定义

```
// 本部分代码未作修改，与3-2一致
```

在头文件`MessageFormat.h`中设计实现我们的报文格式，报文格式主体使用结构体`MsgFormat`实现。

校验和

我们在结构体中定义了校验和的设置函数以及校验函数，其实现逻辑为：

- **发送端　设置校验和**

- 校验和域段清零【同时填充0】
- 以16位（2字节）为单位对数据报进行求和运算，注意溢出部分回加
- 将最终结果（变量低16位）取反填充校验和域段

- **接收端　校验和确认**
  - 以16位（2字节）为单位对数据报进行求和运算，注意溢出部分回加
  - 最终结果（变量低16位）若是全1，则表示未检测到错误，否则说明数据报存在差错
  - 机制解释：由于此时校验和已存在，求和运算实际上是原码+反码，那么正常情况下得到的应该是全1的数据

```
// 本部分代码未作修改，与3-1一致
```

客户端

```
// 本部分代码未作修改，与3-2一致
```

服务器端

```
// 本部分代码未作修改，与3-2一致
```

（二）数据传输

客户端

多线程处理

- 两个线程共享数据
  - `int base`：基序号
  - `int nextseqnum`：下一个序号
  - `clock_t* timers`：计时器
  - `bool over`：结束标记
  - `bool* acked`：确认标记
- 主线程：发送数据

  持续循环发送数据报：
  - 当 ***nextseqnum < base + N*** 时，表示窗口内有可发送数据报，那么此时发送数据报，设置该报文的计时器，并且右移 ***nextseqnum***。
  - **超时重传**：

    重传超时报文，并重新计时。
- 辅助函数线程：接收ACK

  持续循环接收数据报：
  - 接收到ACK：

    当***recvMsg.ack >= base && recvMsg.ack <= base + N - 1***时：`acked[recvMsg.ack] = true;`
    - ***recvMsg.ack == base***：更新窗口
  - 接收到最后一个报文，标记 **over=true**

```
// 报文传输辅助函数--接收线程
struct parameters {
    SOCKADDR_IN serverAddr;
```

```
    SOCKET clientSocket;
    int pkt_amount;  // 报文总数：相对序列号不会比这个大
};

int sended_notacked(int base, int nextseqnum1) {
    int i = 0;
    for (int j = base; j < nextseqnum1; j++) {
        if (!acked[j])
            i++;
    }
    return i;
}

DWORD WINAPI ACKRecvThread(PVOID pParam) {
    parameters* para = (parameters*)pParam;
    SOCKADDR_IN serverAddr = para->serverAddr;
    SOCKET clientSocket = para->clientSocket;
    int pkt_amount = para->pkt_amount;
    int addrlen = sizeof(serverAddr);

    while (1) {
        MsgFormat recvMsg;
        int recvByte = recvfrom(clientSocket, (char*)&recvMsg, sizeof(recvMsg), 0, (sockaddr*)&serverAddr, &addrlen);
        if (recvByte > 0) {  // 成功收到消息【不管是啥，收到东西了】
            if ((recvMsg.Flag && ACK) && recvMsg.check()) {  // 检查校验和
                if (recvMsg.ack >= base && recvMsg.ack <= base + N - 1) {  // 在窗口内
                    // 更新窗口状态
                    if (recvMsg.ack == base) {
                        acked[base] = true;
                        while (acked[base]) {
                            base++;
                        }
                        outputMutex.lock();  // 手动加锁
                        cout << "[传输日志] " << recvMsg.SrcPort << " -> " << recvMsg.DestPort << " ack = " << recvMsg.ack << " Flag = " << recvMsg.Flag << endl;
                        if (base + N < pkt_amount) {  // 展示窗口情况
                            cout << "[传输日志] [接收ACK]当前窗口情况——窗口总大小：" << N << "，已发送但未收到ACK的数据报数目：" << sended_notacked(base, nextseqnum) << "，窗口内尚未发送数据报数目：" << N - (nextseqnum - base) << endl;
                        }
                        else {  // 展示窗口情况
                            cout << "[传输日志] [接收ACK]当前窗口情况——窗口总大小：" << N << "，已发送但未收到ACK的数据报数目：" << sended_notacked(base, nextseqnum) << "，窗口内尚未发送数据报数目：" << pkt_amount - base - (nextseqnum - base) << endl;
                        }
                        outputMutex.unlock();  // 手动解锁
                    }
                    else {
                        acked[recvMsg.ack] = true;
                        outputMutex.lock();  // 手动加锁
                        cout << "[传输日志] " << recvMsg.SrcPort << " -> " << recvMsg.DestPort << " ack = " << recvMsg.ack << " Flag = " << recvMsg.Flag << endl;
```

```
                        outputMutex.unlock();  // 手动解锁
                    }
                }
                // 判断传输结束
                if (base >= pkt_amount) {
                    cout << "[传输日志] **********传输结束! **********" << endl;
                    over = true;
                    return 0;
                }
            }
        }
    }
    return 0;
}
// 报文传输函数
void SR_SendFunc(string path, SOCKET clientSocket, SOCKADDR_IN serverAddr)
{
    int addrlen = sizeof(serverAddr);
    clock_t startTime = clock();
    string filename = "";
    for (int i = path.size() - 1; i >= 0; i--) {  // 逆序获取逆序文件名
        if (path[i] == '/' || path[i] == '\\')
            break;
        filename += path[i];
    }
    filename = string(filename.rbegin(), filename.rend());  // 逆序获取正序文件名
    ifstream f(path.c_str(), ifstream::binary);  // 以二进制方式读取文件
    if (!f) {
        cout << "[传输日志] 无法打开文件..." << endl;
        return;
    }
    BYTE* fileBuffer = new BYTE[MAX_FILE_SIZE];
    unsigned int fileSize = 0;
    BYTE byte = f.get();
    while (f) {  // 将文件读取到缓冲区
        fileBuffer[fileSize++] = byte;
        byte = f.get();
    }
    f.close();

    int batchNum = fileSize / MAX_MSG_SIZE;  // 满载报文数
    int leftSize = fileSize % MAX_MSG_SIZE;  // 剩余报文大小

    //=========================创建接受消息线程=========================
    int pkt_amount = leftSize > 0 ? batchNum + 2 : batchNum + 1;  // +1是由于第一个文件名字及大小的包占一个
    parameters param;
    param.serverAddr = serverAddr;
    param.clientSocket = clientSocket;
    param.pkt_amount = pkt_amount;
    acked = new bool[pkt_amount];
    memset(acked, false, pkt_amount);
    timers = new clock_t[pkt_amount];
    HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ACKRecvThread, &param, 0, 0);
    //=========================主线程：发送=========================
```

```
    MsgFormat sendMsg;
    while (1) {
        if (nextseqnum < base + N && nextseqnum < pkt_amount) {
            if (nextseqnum == 0) {  // 文件名字及大小
                sendMsg.SrcPort = ClientPORT;
                sendMsg.DestPort = RouterPORT;
                sendMsg.size = fileSize;
                sendMsg.Flag += FILEFLAG;
                sendMsg.seq = nextseqnum;
                for (int i = 0; i < filename.size(); i++)  // 填充报文数据段
                    sendMsg.MSGDataBuf[i] = filename[i];
                sendMsg.MSGDataBuf[filename.size()] = '\0';//字符串结尾补\0
                sendMsg.setCheckNum();
            }
            else if (nextseqnum == batchNum + 1 && leftSize > 0) {  // 剩余数据
                sendMsg.SrcPort = ClientPORT;
                sendMsg.DestPort = RouterPORT;
                sendMsg.size = leftSize;
                sendMsg.Flag = FILEDATA;
                sendMsg.seq = nextseqnum;
                for (int j = 0; j < leftSize; j++) {
                    sendMsg.MSGDataBuf[j] = fileBuffer[batchNum * MAX_MSG_SIZE + j];
                }
                sendMsg.setCheckNum();
            }
            else {
                sendMsg.SrcPort = ClientPORT;
                sendMsg.DestPort = RouterPORT;
                sendMsg.size = MAX_MSG_SIZE;
                sendMsg.Flag = FILEDATA;
                sendMsg.seq = nextseqnum;
                for (int j = 0; j < MAX_MSG_SIZE; j++) {
                    sendMsg.MSGDataBuf[j] = fileBuffer[(nextseqnum - 1) * MAX_MSG_SIZE + j];
                }
                sendMsg.setCheckNum();
            }
            // 发送
            sendto(clientSocket, (char*)&sendMsg, sizeof(sendMsg), 0, (sockaddr*)&serverAddr, addrlen);
            outputMutex.lock();  // 手动加锁
            cout << "[传输日志] " << sendMsg.SrcPort << " -> " << sendMsg.DestPort << " size = " << sendMsg.size << "B seq = " << sendMsg.seq << " Flag = " << sendMsg.Flag << endl;
            outputMutex.unlock();  // 手动解锁

            timers[nextseqnum] = clock();  // 给每一个包设置计时器
            nextseqnum++;
            outputMutex.lock();  // 手动加锁
            if (base + N < pkt_amount) {  // 展示窗口情况
                cout << "[传输日志] 当前窗口情况--窗口总大小：" << N << "，已发送但未收到ACK的数据报数目：" << sended_notacked(base, nextseqnum) << "，窗口内尚未发送数据报数目：" << N - (nextseqnum - base) << endl;
            }
            else {  // 展示窗口情况
```

```
                cout << "[传输日志] 当前窗口情况--窗口总大小：" << N << "，已发送但未收到ACK的数据报数目：" << sended_notacked(base, nextseqnum) << "，窗口内尚未发送数据报数目：" << pkt_amount - base - (nextseqnum - base) << endl;
            }
            outputMutex.unlock();  // 手动解锁
        }
        // 超时重传
        for (int i = 0; i < nextseqnum - base; i++) {
            int send_seq = base + i;
            if (!acked[send_seq] && ((clock() - timers[send_seq]) > MAX_WAIT_TIME)) {  // SR--超时包重传
                if (send_seq == 0) {
                    sendMsg.SrcPort = ClientPORT;
                    sendMsg.DestPort = RouterPORT;
                    sendMsg.size = fileSize;
                    sendMsg.Flag += FILEFLAG;
                    sendMsg.seq = send_seq;
                    for (int i = 0; i < filename.size(); i++)  // 填充报文数据段
                        sendMsg.MSGDataBuf[i] = filename[i];
                    sendMsg.MSGDataBuf[filename.size()] = '\0';//字符串结尾补\0
                    sendMsg.setCheckNum();
                    outputMutex.lock();  // 手动加锁
                    cout << "[传输日志] 传输文件相关信息--文件名：" << filename << " 文件大小：" << fileSize << "B" << endl;
                    outputMutex.unlock();  // 手动解锁
                }
                else if (send_seq == batchNum + 1 && leftSize > 0) {
                    sendMsg.SrcPort = ClientPORT;
                    sendMsg.DestPort = RouterPORT;
                    sendMsg.size = leftSize;
                    sendMsg.Flag = FILEDATA;
                    sendMsg.seq = send_seq;
                    for (int j = 0; j < leftSize; j++) {
                        sendMsg.MSGDataBuf[j] = fileBuffer[batchNum * MAX_MSG_SIZE + j];
                    }
                    sendMsg.setCheckNum();
                }
                else {
                    sendMsg.SrcPort = ClientPORT;
                    sendMsg.DestPort = RouterPORT;
                    sendMsg.size = MAX_MSG_SIZE;
                    sendMsg.Flag = FILEDATA;
                    sendMsg.seq = send_seq;
                    for (int j = 0; j < MAX_MSG_SIZE; j++) {
                        sendMsg.MSGDataBuf[j] = fileBuffer[(send_seq - 1) * MAX_MSG_SIZE + j];
                    }
                    sendMsg.setCheckNum();
                }
                // 发送
                sendto(clientSocket, (char*)&sendMsg, sizeof(sendMsg), 0, (sockaddr*)&serverAddr, addrlen);
                timers[send_seq] = clock();
                outputMutex.lock();  // 手动加锁
```

```
                cout << "[传输日志] seq = " << sendMsg.seq << "的报文段超时，正在重传..." << endl;
                outputMutex.unlock();  // 手动解锁
            }
        }
        if (over == true) break;  // 已收到所有ACK
    }
    CloseHandle(hThread);
    cout << "[传输日志] 已发送并确认所有报文，文件传输成功！" << endl;
    clock_t endTime = clock();
    cout << "-------------------------------------------------" << endl;
    cout << "[传输日志] 总体传输时间为:" << (endTime - startTime) / CLOCKS_PER_SEC << "s" << endl;
    cout << "[传输日志] 吞吐率:" << ((float)fileSize) / ((endTime - startTime) / CLOCKS_PER_SEC) << "byte/s" << endl;
    cout << "=================================================" << endl;
    return;
}
```

服务器端

引入如下变量：

- `int base`：基序号
- `bool* acked`：确认标记

重点在于辅助函数`bool msgRecv`，引入映射`unordered_map<int, MsgFormat>& receivedPackets`，与客户端具有相似的窗口更新逻辑，具体实现如下

```
// 报文接收辅助函数
bool msgRecv(MsgFormat& recvMsg, SOCKET serverSocket, SOCKADDR_IN clientAddr, unordered_map<int, MsgFormat>& receivedPackets) {
    int addrlen = sizeof(clientAddr);
    while (1) {
        int recvByte = recvfrom(serverSocket, (char*)&recvMsg, sizeof(recvMsg), 0, (sockaddr*)&clientAddr, &addrlen);
        if (recvByte > 0) {
            if (recvMsg.check()) {
                if (recvMsg.seq >= base && recvMsg.seq <= base + N - 1) { // 在窗口内
                    if (recvMsg.seq == base) {  // 接收左边界
                        receivedPackets[recvMsg.seq] = recvMsg;
                        MsgFormat ackMsg;
                        ackMsg.SrcPort = ServerPORT;
                        ackMsg.DestPort = RouterPORT;
                        ackMsg.Flag += ACK;
                        ackMsg.ack = recvMsg.seq;
                        ackMsg.setCheckNum();
                        sendto(serverSocket, (char*)&ackMsg, sizeof(ackMsg), 0, (sockaddr*)&clientAddr, addrlen);
                        cout << "[传输日志] " << recvMsg.SrcPort << " -> " << recvMsg.DestPort << " size = " << recvMsg.size << "B seq = " << recvMsg.seq << " Flag = " << recvMsg.Flag << endl;
                        cout << "[传输日志] " << ackMsg.SrcPort << " -> " << ackMsg.DestPort << " ack = " << ackMsg.ack << " Flag = " << ackMsg.Flag << endl;
                        acked[base] = true;
```

```
                        while (acked[base]) {
                            base++;
                        }
                        return true;
                    }
                    else {  // 窗口内其他包
                        if (!acked[recvMsg.seq]) {
                            acked[recvMsg.seq] = true;
                            receivedPackets[recvMsg.seq] = recvMsg;
                            MsgFormat ackMsg;
                            ackMsg.SrcPort = ServerPORT;
                            ackMsg.DestPort = RouterPORT;
                            ackMsg.Flag += ACK;
                            ackMsg.ack = recvMsg.seq;
                            ackMsg.setCheckNum();
                            sendto(serverSocket, (char*)&ackMsg, sizeof(ackMsg), 0, (sockaddr*)&clientAddr, addrlen);
                            cout << "[传输日志] [失序缓存]" << recvMsg.SrcPort << " -> " << recvMsg.DestPort << " size = " << recvMsg.size << "B seq = " << recvMsg.seq << " Flag = " << recvMsg.Flag << endl;
                            cout << "[传输日志]  " << ackMsg.SrcPort << " -> " << ackMsg.DestPort << " ack = " << ackMsg.ack << " Flag = " << ackMsg.Flag << endl;
                            return true;
                        }
                    }
                }
                else if (recvMsg.seq >= base - N && recvMsg.seq <= base - 1) {  // 前一个窗口
                    MsgFormat ackMsg;
                    ackMsg.SrcPort = ServerPORT;
                    ackMsg.DestPort = RouterPORT;
                    ackMsg.Flag += ACK;
                    ackMsg.ack = recvMsg.seq;
                    ackMsg.setCheckNum();
                    sendto(serverSocket, (char*)&ackMsg, sizeof(ackMsg), 0, (sockaddr*)&clientAddr, addrlen);
                    cout << "[传输日志] [已缓存] " << recvMsg.SrcPort << " -> " << recvMsg.DestPort << " size = " << recvMsg.size << "B seq = " << recvMsg.seq << " Flag = " << recvMsg.Flag << endl;
                    cout << "[传输日志] [回复ACK]" << ackMsg.SrcPort << " -> " << ackMsg.DestPort << " ack = " << ackMsg.ack << " Flag = " << ackMsg.Flag << endl;
                }
            }
        }
        else if (recvByte == 0) {
            return false;
        }
    }
}
// 报文接收函数
void RecvFunc(SOCKET serverSocket, SOCKADDR_IN clientAddr) {
    int addrlen = sizeof(clientAddr);
    // 接收端缓冲区
    unordered_map<int, MsgFormat> receivedPackets;
    //============================文件名和大小=========================
    MsgFormat rMsg;
```

```
68      unsigned int filesize;
69      char filename[50] = { 0 };
70      while (1) {
71          int recvByte = recvfrom(serverSocket, (char*)&rMsg, sizeof(rMsg),
    0, (sockaddr*)&clientAddr, &addrlen);
72          if (recvByte > 0) {
73              if (rMsg.check() && ((rMsg.seq == base))) {
74                  filesize = rMsg.size;  // 获取文件大小
75                  for (int i = 0; rMsg.MSGDataBuf[i]; i++)  // 获取文件名
76                      filename[i] = rMsg.MSGDataBuf[i];
77                  cout << "-----------------------------------------------"
    << endl;
78                  cout << "[传输日志] 接收文件：" << filename << "，文件大小：" <<
    filesize << "B" << endl;
79                  cout << "-----------------------------------------------"
    << endl;
80                  MsgFormat ackMsg;
81                  ackMsg.SrcPort = ServerPORT;
82                  ackMsg.DestPort = RouterPORT;
83                  ackMsg.Flag += ACK;
84                  ackMsg.ack = rMsg.seq;
85                  ackMsg.setCheckNum();
86                  sendto(serverSocket, (char*)&ackMsg, sizeof(ackMsg), 0,
    (sockaddr*)&clientAddr, addrlen);
87                  cout << "[传输日志] " << rMsg.SrcPort << " -> " <<
    rMsg.DestPort << " size = " << rMsg.size << "B seq = " << rMsg.seq << "
    Flag = " << rMsg.Flag << endl;
88                  cout << "[传输日志] " << ackMsg.SrcPort << " -> " <<
    ackMsg.DestPort << " ack = " << ackMsg.ack << " Flag = " << ackMsg.Flag <<
    endl;
89                  base++;
90                  break;
91              }
92          }
93      }
94      //============================文件数据部分============================
95      int batchNum = filesize / MAX_MSG_SIZE;  // 满载报文数
96      int leftSize = filesize % MAX_MSG_SIZE;  // 剩余报文大小
97      BYTE* fileBuffer = new BYTE[filesize];
98      cout << "[传输日志] 开始接收数据，共 " << batchNum << " 个满载报文段，非满载报
    文大小：" << leftSize << endl;
99      int pkt_amount = leftSize > 0 ? batchNum + 2 : batchNum + 1;
100     acked = new bool[pkt_amount];
101     memset(acked, false, pkt_amount);
102     int i = 0;
103     while (i < pkt_amount - 1) {
104         MsgFormat dMsg;
105         if (msgRecv(dMsg, serverSocket, clientAddr, receivedPackets))
106             i++;
107         else {
108             cout << "[传输日志] 数据接收失败..." << endl;
109             cout << "================================================" <<
    endl;
110             return;
111         }
112         if (dMsg.seq == batchNum + 1 && leftSize > 0) {
113             for (int j = 0; j < leftSize; j++) {
```

```
                fileBuffer[batchNum * MAX_MSG_SIZE + j] = receivedPackets[dMsg.seq].MSGDataBuf[j];
            }
        }
        else {
            for (int j = 0; j < MAX_MSG_SIZE; j++) {  // 读取数据部分
                fileBuffer[(dMsg.seq - 1) * MAX_MSG_SIZE + j] = receivedPackets[dMsg.seq].MSGDataBuf[j];
            }
        }
    }
    cout << "[传输日志] 数据接收成功！正在写入本地..." << endl;
    ofstream f(filename, ofstream::binary);  // 以二进制方式读取文件
    if (f.is_open()) {
        f.write(reinterpret_cast<const char*>(fileBuffer), filesize);
        f.close();
        cout << "[传输日志] 文件写入成功！" << endl;
        cout << "================================================" << endl;
    }
}
```

（三）程序使用

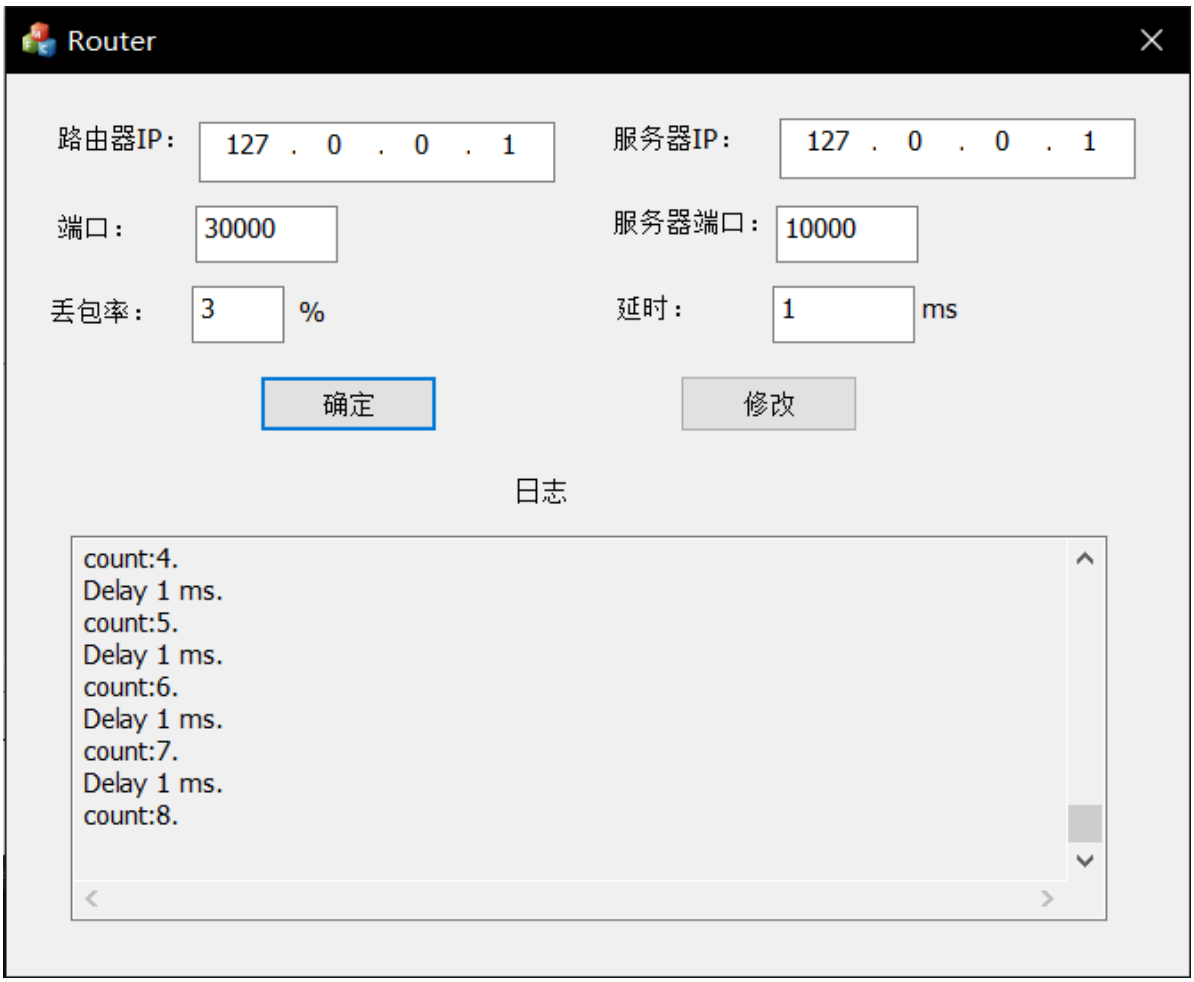
客户端

```
while (res) {  // res是建连返回值
        char c;
        if (run_flag) {
            run_flag = 0;
            cout << "[系统提示] 请输入选择您要使用的功能：";
        }
        else {
            cout << "[系统提示] 请输入 q/Q 关闭连接：";
        }
        cin >> c;
        if (c == 'r' || c == 'R')
        {
            string filepath;
            cout << "[系统提示] 请输入文件绝对路径：" << endl;
            cin >> filepath;
            //=============================文件传输============================
            GBN_SendFunc(filepath, clientSocket, serverAddr);
        }
        else if (c == 'q' || c == 'Q')
        {
            res = false;  // 退出循环
        }
    }
```

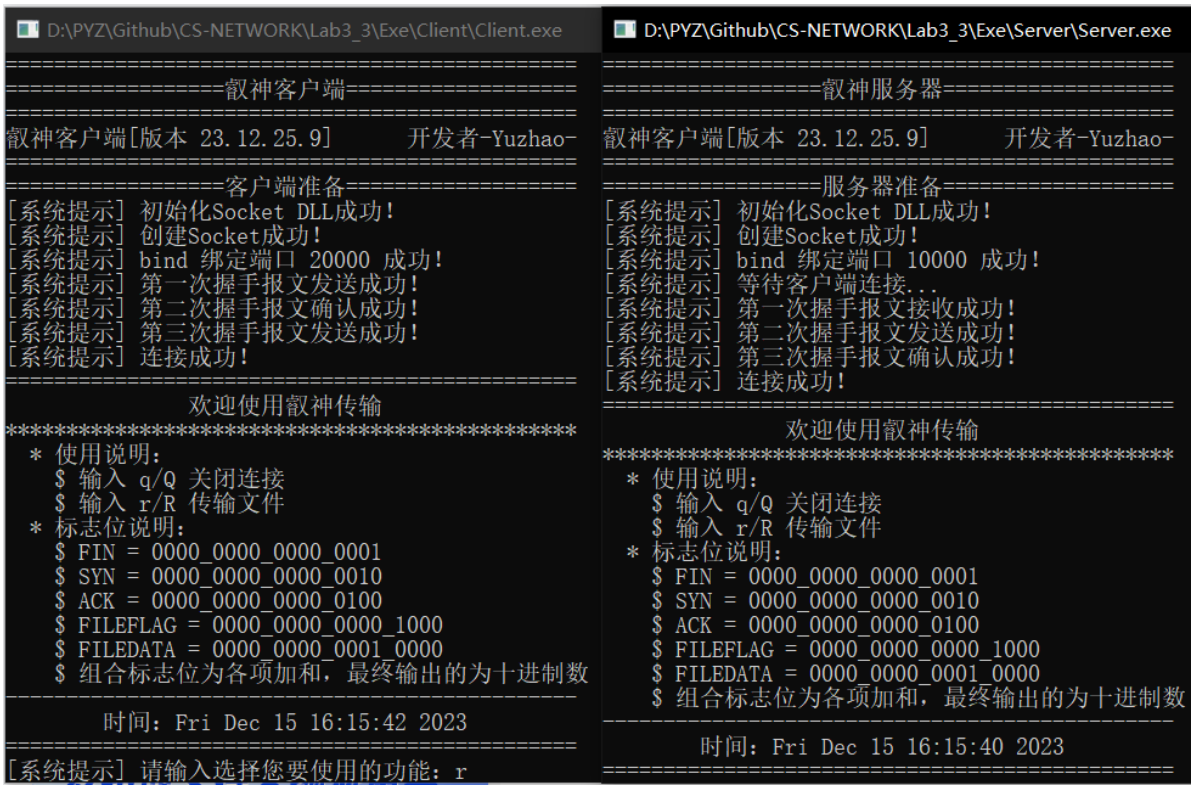
四、程序运行演示

路由器设置

- 丢包率：3%
- 延时：1ms


Router
路由器IP：
127 . 0 . 0 . 1
服务器IP：
127 . 0 . 0 . 1
端口：
30000
服务器端口：
10000
丢包率：
3
%
延时：
1
ms
确定
修改
日志
count:4.
Delay 1 ms.
count:5.
Delay 1 ms.
count:6.
Delay 1 ms.
count:7.
Delay 1 ms.
count:8.

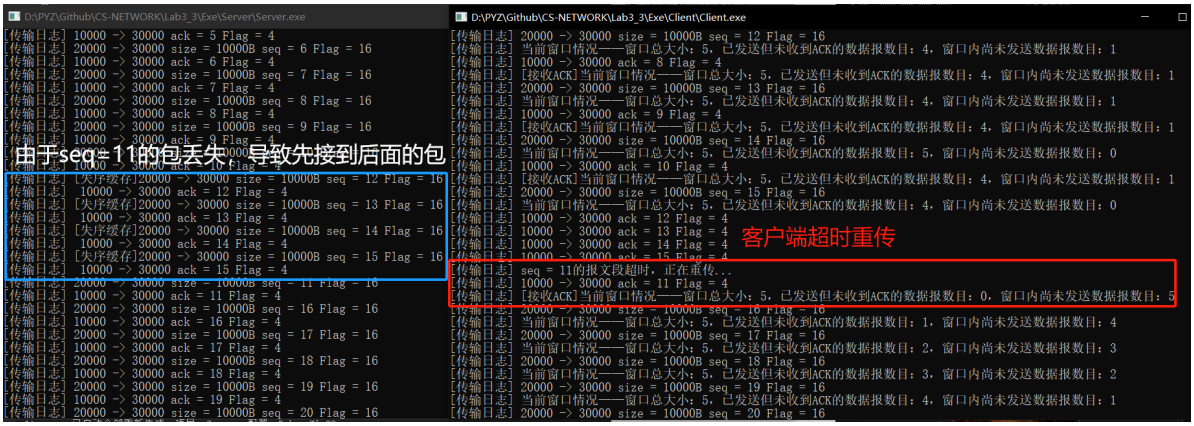
建立连接


D:\PYZ\Github\CS-NETWORK\Lab3_3\Exe\Client\Client.exe
==========叡神客户端==========
叡神客户端[版本 23.12.25.9] 开发者-Yuzhao-
==========客户端准备==========
[系统提示] 初始化Socket DLL成功!
[系统提示] 创建Socket成功!
[系统提示] bind 绑定端口 20000 成功!
[系统提示] 第一次握手报文发送成功!
[系统提示] 第二次握手报文确认成功!
[系统提示] 第三次握手报文发送成功!
[系统提示] 连接成功!
欢迎使用叡神传输
* 使用说明:
$ 输入 q/Q 关闭连接
$ 输入 r/R 传输文件
* 标志位说明:
$ FIN = 0000_0000_0000_0001
$ SYN = 0000_0000_0000_0010
$ ACK = 0000_0000_0000_0100
$ FILEFLAG = 0000_0000_0000_1000
$ FILEDATA = 0000_0000_0001_0000
$ 组合标志位为各项加和，最终输出的为十进制数
时间：Fri Dec 15 16:15:42 2023
[系统提示] 请输入选择您要使用的功能：r
D:\PYZ\Github\CS-NETWORK\Lab3_3\Exe\Server\Server.exe
==========叡神服务器==========
叡神客户端[版本 23.12.25.9] 开发者-Yuzhao-
==========服务器准备==========
[系统提示] 初始化Socket DLL成功!
[系统提示] 创建Socket成功!
[系统提示] bind 绑定端口 10000 成功!
[系统提示] 等待客户端连接...
[系统提示] 第一次握手报文接收成功!
[系统提示] 第二次握手报文发送成功!
[系统提示] 第三次握手报文确认成功!
[系统提示] 连接成功!
欢迎使用叡神传输
* 使用说明:
$ 输入 q/Q 关闭连接
$ 输入 r/R 传输文件
* 标志位说明:
$ FIN = 0000_0000_0000_0001
$ SYN = 0000_0000_0000_0010
$ ACK = 0000_0000_0000_0100
$ FILEFLAG = 0000_0000_0000_1000
$ FILEDATA = 0000_0000_0001_0000
$ 组合标志位为各项加和，最终输出的为十进制数
时间：Fri Dec 15 16:15:40 2023

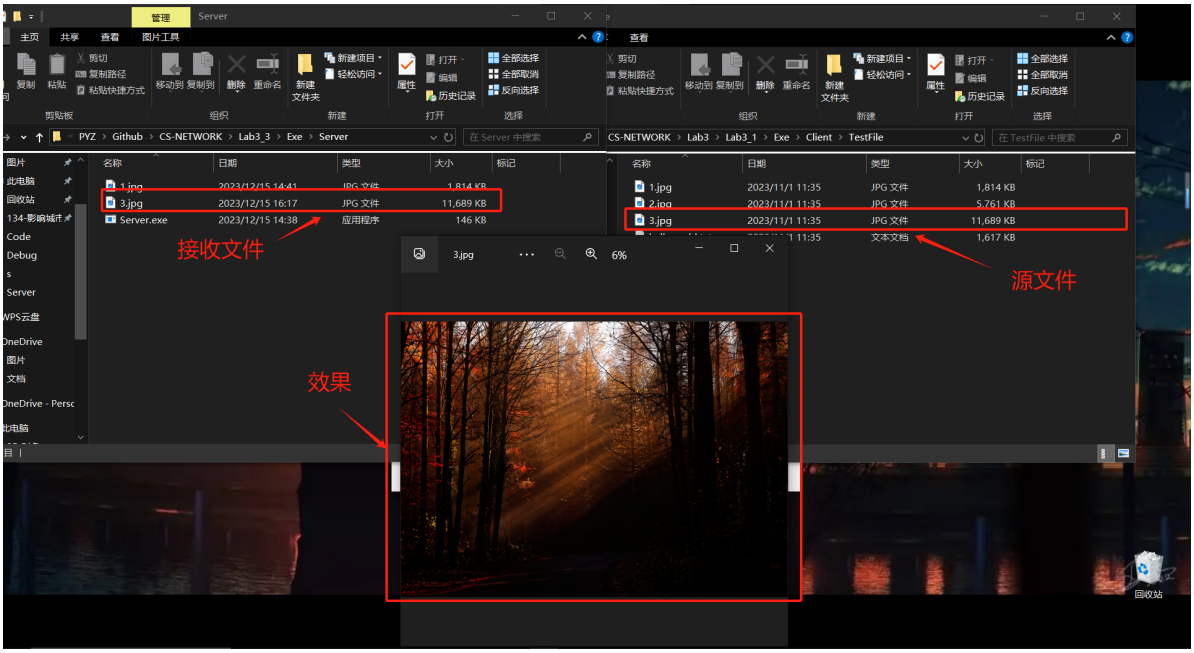
关闭连接

文件传输

超时重传&选择确认



传输结果



数据分析

窗口大小——发送窗口和接收窗口均为5

测试文件	传输总时间	吞吐率
1.jpg	11s	168850Bps
2.jpg	30s	196617Bps
3.jpg	65s	184138Bps
helloworld.txt	9s	183979Bps

五、总结

本次实验基于上一次实验，重点在于SR协议的实现，原理已在上文中有所叙述。在整个实验过程中，被一个小小的地方卡住了很久——服务器端接收辅助函数的返回值。这里的问题是，***当我们接到窗口内的报文时都需要缓存***这样粗放的感觉是否正确？很显然是不对的，当我重复收到一个报文时（注意这不是没有可能的），我将会认为我需要写入缓冲区，事实上我已经写过这一块了，那么就会导致我们服务器端提前结束收取报文，并造成数据丢失。因此，我需要先判断我是否已经收过该包，在没收过时才会接收并将数据写入缓冲区。