哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	实验 2：可靠数据传输协议-停等协议、**GBN** 协议的设计与实现				
姓名			院系		
班级			学号		
任课教师	刘亚维		指导教师	刘亚维	
实验地点	G002		实验时间	2024.04.23	
实验课表现	出勤、表现得分(10)		实验报告得分(40)	实验总分	
	操作结果得分(50)				
教师评语					

哈尔滨工业大学计算学部
FACULTY OF COMPUTING, HIT

实验目的：

1. 理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。
2. 理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

实验内容：

1. 基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）
2. 模拟引入数据包的丢失，验证所设计协议的有效性；
3. 改进所设计的停等协议，支持双向数据传输
4. 基于所设计的停等协议，实现一个 C/S 结构的文件传输应用
5. 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）
6. 模拟引入数据包的丢失，验证所设计协议的有效性
7. 改进所设计的 GBN 协议，支持双向数据传输
8. 将所设计的 GBN 协议改进为 SR 协议

实验过程：

首先设计 GBN 协议，将 GBN 协议的滑动窗口设置为 1 就可以得到停-等协议。之后在 GBN 协议的基础上，为每个分组都设置一个计时器，并修改超时重传函数，同时添加缓存，便可以得到 SR 协议。在实现了文件单向传输后，将服务器的发送代码添加至客户代码中，将客户的接收代码添加至服务器代码中，便可以支持双向传输了。

GBN 协议概览

GBN 协议又叫回退 N 步协议，如果某个报文段没有被正确的接收，那么从这个报文段到后面的报文段都要重新发送，在 GBN 协议中，返回某个 ACK，则代表该 ACK 序号及其之前的报文段均已经收到。

对于服务器端：首先，服务器端等待客户端的请求，接收来自客户端的消息。当服务器端接收到客户端发来的数据传输请求-testgbn 时，服务器端将数据分割成一个个数据报进行发送。发送数据报之后，服务端会开启计时器（只有一个计时器），并等待客户端的 ACK 信息。当收到客户端回复的 ACK 时，服务器端就能确认 ACK 及其之前的数据报均已收到，服务器端的发送窗口可以滑动，正常发送下一个数据报，计时器重新计时。若在计时器超时前没有收到 ACK，则全部重传窗口内最大 ACK 之后的所有数据报。

其中计时器代码如下

```
//等待 Ack，若没有收到，则返回值为-1，计数器+1
recvSize = recvfrom(sockServer, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrClient), &length);
if (recvSize < 0) {
    waitCount++;
    //10 次等待 ack 则超时重传
    if (waitCount > 10)
    {
        waitCount = 0;
        timeoutHandler();
    }
}
else {
    //收到 ack
    ackHandler(buffer[0]);
    waitCount = 0;
}
```

超时处理函数如下

```
void timeoutHandler() { // modify
    printf("Timer out error.\n");
    int index;
    for (int i = 0; i < SEND_WIND_SIZE; ++i) {
        index = (i + curAck) % SEQ_SIZE;
        ack[index] = TRUE;
    }
    // totalSeq = curAck;
    if (curSeq>=curAck)
        totalSeq = totalSeq - (curSeq - curAck);
    else
        totalSeq = totalSeq - (SEQ_SIZE - curAck + curSeq);
    curSeq = curAck;
}
```

对于客户端：接收服务器端发送的数据报并返回确认信息 ACK。通过随机数的方式可以模拟 ACK 丢失和数据报丢失，然后由服务器端检测超时重传的情况。

模拟丢失的函数如下

```
BOOL lossInLossRatio(float lossRatio) {
    int lossBound = (int)(lossRatio * 100);
    int r = rand() % 101;
    if (r <= lossBound) {
        return TRUE;
    }
    return FALSE;
}
```

GBN 协议具体实现

进入**客户端**后，当匹配到输入的是“-time”或者“-quit”则作为数据包发送给服务器端。如果匹配到输入是“-testgbn [X] [Y]”，则进入 GBN 传输阶段。

以下重点介绍“-testgbn [X] [Y]”输入时的过程

首先初始化数据包，把“testgbn [X][Y]”这段报文发到服务器端，并与服务器端进行握手，握手后服务器开始向客户传输数据

如图所示，case 0 为等待握手阶段

```
case 0://等待握手阶段
    if ((unsigned char)buffer[0] == 205)
    {
        printf("Ready for file transmission\n");
        buffer[0] = 200;
        buffer[1] = '\0';
        sendto(socketClient, buffer, 2, 0,
            (SOCKADDR*)&recvClient, sizeof(SOCKADDR));
        stage = 1;
        recvSeq = 0;
        waitSeq = 1;
    }
    break;
```

case 1 为等待数据接收阶段，在这使用刚刚提到的 lossInLossRatio 函数模拟数据丢失的情况。如果数据没有丢失且是期望接收的数据，则会制作 ack 并发送给服务器端。如果不是期望接收的数据，则发回上一个 ack。同样的，使用 lossInLossRatio 函数模拟 ack 丢失的情况。

```
case 1://等待接收数据阶段
    seq = (unsigned short)buffer[0];
    //随机法模拟包是否丢失
    b = lossInLossRatio(packetLossRatio);
    if (!b) printf("\nThe packet wished to receive is %d\n", waitSeq);  // modify*3
    if (b) {
        printf("The packet with a seq of %d loss\n", seq);
        continue;
    }
    printf("recv a packet with a seq of %d\n", seq);
    //如果是期待的包，正确接收，正常确认即可
    if (!(waitSeq - seq)) {
        ++waitSeq;
        if (waitSeq == 21) {
            waitSeq = 1;
        }
        //输出数据
        // printf("buffer=%s\n", &buffer[1]);
        buffer[0] = seq;
        recvSeq = seq;
        if (out.is_open()) {
            try {
                out.write(&buffer[1], strlen(&buffer[1]));
            }
            catch (const std::ios_base::failure& e) {
                std::cerr << "Caught an I/O exception: " << e.what()
                    << ", error code: " << e.code() << std::endl;
            }
        }
        else {
            std::cerr << "File not opened!" << std::endl;
        }
    }
```

在**服务端**运行之后，首先会绑定监听端口，监听客户端的命令执行函数，如果是“time”会返回时间，如果是“-quit”则退出程序

如果是“testgbn [X][Y]”则会开始与客户端进行握手，握手后向客户端发送数据

```
case 2://数据传输阶段
    if (seqIsAvailable() && totalSeq < totalPacket) {
        buffer[0] = curSeq + 1;
        ack[curSeq] = FALSE;
        memcpy(&buffer[1], data + 1024 * totalSeq, 1024);
        printf("send a packet with a seq of %d\n", curSeq + 1);
        sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
        ++curSeq;
        curSeq %= SEQ_SIZE;
        ++totalSeq;
        Sleep(500);
    }
    //等待 Ack，若没有收到，则返回值为-1，计数器+1
    recvSize = recvfrom(sockServer, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrClient), &length);
    if (recvSize < 0) {
        waitCount++;
        //10 次等待 ack 则超时重传
        if (waitCount > 10)
        {
            waitCount = 0;
            timeoutHandler();
        }
    }
    else {
        //收到 ack
        ackHandler(buffer[0]);
        waitCount = 0;
    }
    Sleep(500);
    break;
}
```

如代码所示，case2 就是数据传输阶段。服务端首先调用函数 seqIsAvailable()函数查看是否有空的序列号，如果有空的序列号说明窗口还有剩余，然后将当前序列号+1 并封装到 buffer[0]，将相应长度的分组放到 buffer[1]，开始将数据包发送给客户端。发完之后 curSeq++并对 SEQ_SIZE 取模得到新的 curSeq。接下来会等待 ack，如果没有收到则会返回-1 并计时器+1，知道超时进入超时重传函数。如果等到了期待的 ack 则会重置计时器

为了支持**双向数据传输**，代码将数据接收封装成了一个子线程，同时在服务端添加数据接收子线程。在服务器和客户启动时便会创建数据接收子线程，以检测发送来的数据。与此同时在客户端添加发送数据相关代码，至此便实现了双向数据传输功能。

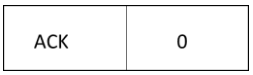
```
unsigned int __stdcall ProxyThread(LPVOID lpParameter) {
    //加载套接字库（必须）
    WORD wVersionRequested;
    WSADATA wsaData;
    //套接字加载时错误提示
    int err;
    //版本 2.2
    wVersionRequested = MAKEWORD(2, 2);
    //加载 dll 文件 Scoket 库
    err = WSAStartup(wVersionRequested, &wsaData);
    if (err != 0) {
        //找不到 winsock.dll
        printf("WSAStartup failed with error: %d\n", err);
        return 1;
    }
    if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
    {
        printf("Could not find a usable version of Winsock.dll\n");
        WSACleanup();
    }
    else {
        printf("The Winsock 2.2 dll was found okay\n");
    }
    SOCKET socketClient = socket(AF_INET, SOCK_DGRAM, 0);
    SOCKADDR_IN recvClient;
    recvClient.sin_addr.S_un.S_addr = inet_addr(CLIENT_IP);
    recvClient.sin_family = AF_INET;
    recvClient.sin_port = htons(CLIENT_PORT);
    //接收缓冲区
    char buffer[BUFFER_LENGTH];
    ZeroMemory(buffer, sizeof(buffer));
    int len = sizeof(SOCKADDR);
```

GBN 协议理论补充（停-等协议同理）

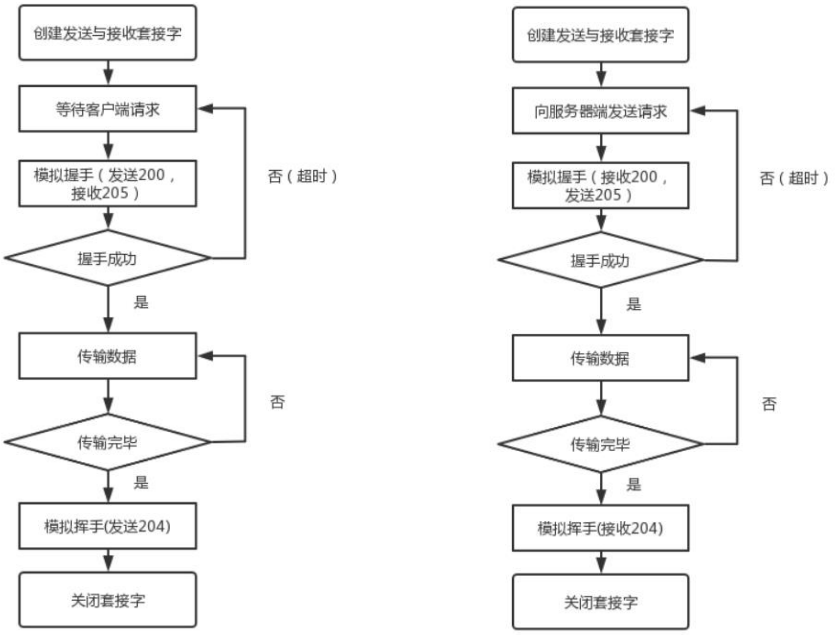
GBN 协议数据分组格式

1. 序列号：Seq 为一个字节，取值 0-255
2. 数据：Data 为实际要传输的数据
3. 结尾：最后一个字节放入 0 表示结尾

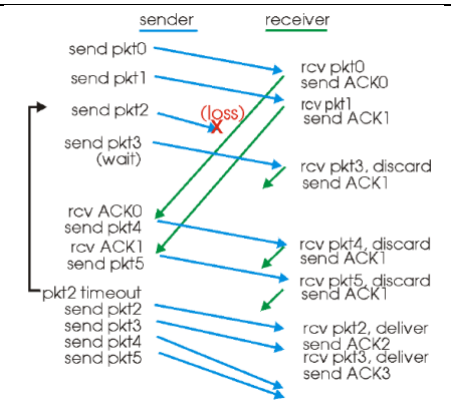
确认分组格式

1. 代码中使用 ack 数组维护了确认分组，其中 curAck 表示当前等待确认的 ack，curSeq 表示当前数据包的 seq
2. 

协议两端程序流程图（左为服务端，右为客户端）



协议典型交互过程：即上文所述的握手过程和文件传输过程



数据分组丢失模拟验证方法：即上文所述 lossInLossRatio 函数

程序实现的主要类（或函数）及其主要作用

1. getCurTime：获取并封装当前时间
2. seqIsAvailable：检查窗口是否满了
3. timeoutHandler：超时重传
4. ackHandler：ack 确认函数，处理收到的 ack
5. lossInLossRatio：模拟数据丢失

UDP 编程的主要特点：无连接、简单、快速、不可靠

停-等协议具体实现

将 GBN 协议中的窗口大小（SEND_WIND_SIZE）改为 1，GBN 协议变为停-等协议，并且能实现双向数据传输。

SR 协议的具体实现

SR 协议只需要在 GBN 协议的基础上通过为每一个数据分组都增加计时器，同时增添缓存区以缓存那些乱序到达的数据

通过在接收方设置缓冲区，为每个报文段单独设置计时器的方式，实现如果某个报文段没有被正确接收但是后面的报文段被正确接收了，那么就只需要重发这一个报文段，而之前收到的乱序报文段存在缓存中。SR 协议中返回的 ACK 为当前接收成功的报文段序号。

SR 与 GBN 最大的区别是：对于 SR 协议，超时事件发生时，服务器端并不重传窗口内所有已发送的数据报，而是只发送窗口内那些没有收到 ACK 的数据报；对于客户端，收到哪个序号的数据包就返回哪个序号的 ACK。即使前面有还没有收到的分组，也会将该组缓存下来，待收到按顺序的一组数据报时，将其交付给上层协议。

对于**服务端**，通过一维组 counter 为每个分组计时，通过二维数组 cache 缓存已发送的分组。当发送某个分组时，首先将它缓存至 cache 中，它的计时器启动，在接收到该分组的 ack 后关闭它的计时器并标记确认，如果窗口左侧的分组已确认收到，则移动窗口；若

超时，则会重新发送 cache 中的该分组，并将计时器重置

```
case 2://数据传输阶段
    if (seqIsAvailable()) {
        if (totalSeq <= totalPacket - 1) {
            //发送给客户端的序列号从 1 开始
            //printf("curSeq = %d\n", curSeq + 1);
            buffer[0] = curSeq + 1;
            ack[curSeq] = 1;
            memcpy(&buffer[1], data + DATA_SIZE * totalSeq, DATA_SIZE);
            memcpy(cache[curSeq], data + DATA_SIZE * totalSeq, DATA_SIZE);//缓存分组
            printf("send a packet with a seq of %d\n", curSeq + 1);
            //printf("buffer = %s\n", buffer);
            sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
            counter[curSeq] = 0;//计时器开启
            ++curSeq;
            curSeq %= SEQ_SIZE;
            ++totalSeq;
            Sleep(500);
        }
    }
```

对于**客户端**，同样使用二维数组 cache 缓存收到的分组。当分组失序到达时会在 cache 中进行缓存，缓存后发送 ack，当 cache 中的分组有序时，从窗口最左开始将缓存中连续有序的分组写入文件，随后移动窗口

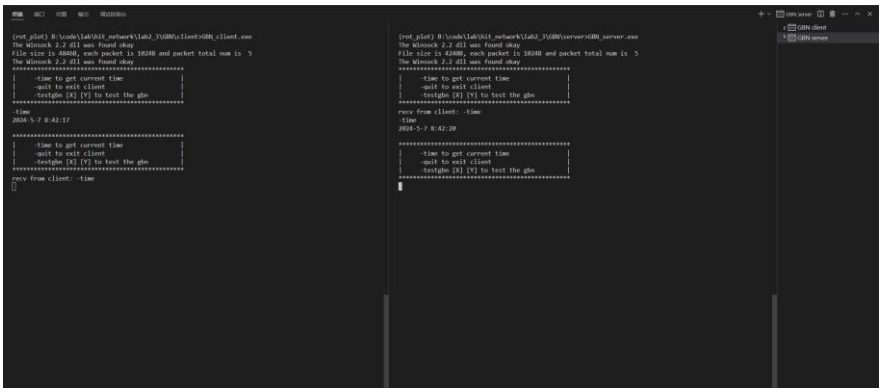
```
printf("recv a packet with a seq of %d\n", seq);
if (waitSeq == seq) {
    //当前接收分组直接写入文件
    out.write(&buffer[1], strlen(&buffer[1]));
    //printf("buffer %d: %s\n", seq, &buffer[1]);
    Ack[seq] = 0;
    waitSeq += 1;
    waitSeq = (waitSeq % SEQ_SIZE == 0) ? SEQ_SIZE : waitSeq % SEQ_SIZE;
    int k = waitSeq + SEND_WIND_SIZE - 1;
    k = (k % SEQ_SIZE == 0) ? SEQ_SIZE : k % SEQ_SIZE;
    Ack[k] = 1;
    //查看是否有失序分组需要写入文件
    for (int i = waitSeq; i < waitSeq + SEND_WIND_SIZE; i++) {
        int t = (i % SEQ_SIZE == 0) ? SEQ_SIZE : i % SEQ_SIZE;
        waitSeq = t;
        //printf("i : %d, Ack[i] : %d\n", i, Ack[i]);
        if (Ack[t] == 2) {
            Ack[t] = 0;
            //printf("I'm here!!%d\n", t);
            k = t + SEND_WIND_SIZE;
            k = (k % SEQ_SIZE == 0) ? SEQ_SIZE : k % SEQ_SIZE;
            Ack[k] = 1;
            printf("seq %d will be Cached out now\n", t);
            //printf("Cache %d write in len %d: %s\n", t, cache_lengths[t], cache[t]);
            out.write(cache[t], cache_lengths[t]);
        }
        else break;
    }
```

其它部分的实现与 GBN 协议相同，这里不做赘述

实验结果：

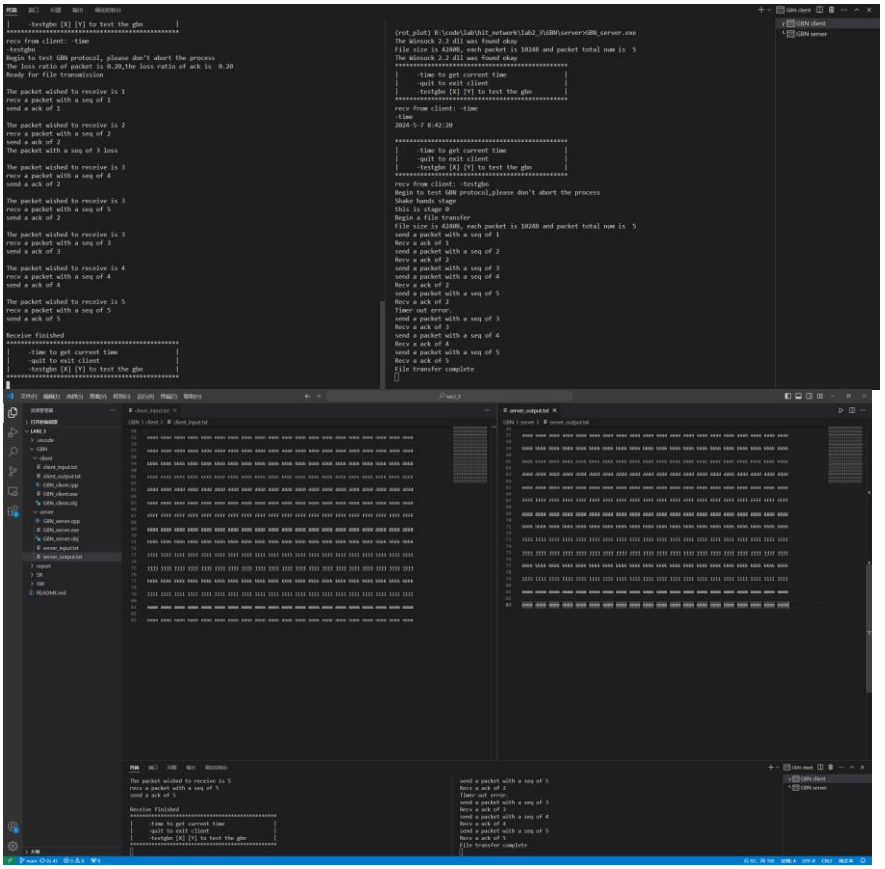
GBN协议结果展示

1. 获取时间（双向）

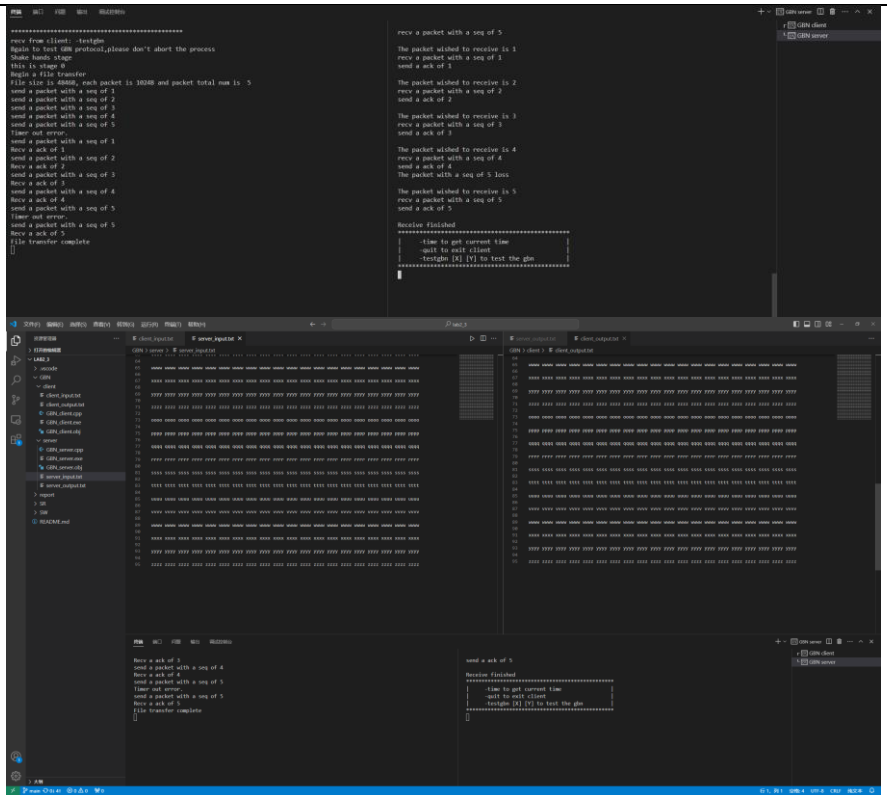


2. 双向传递

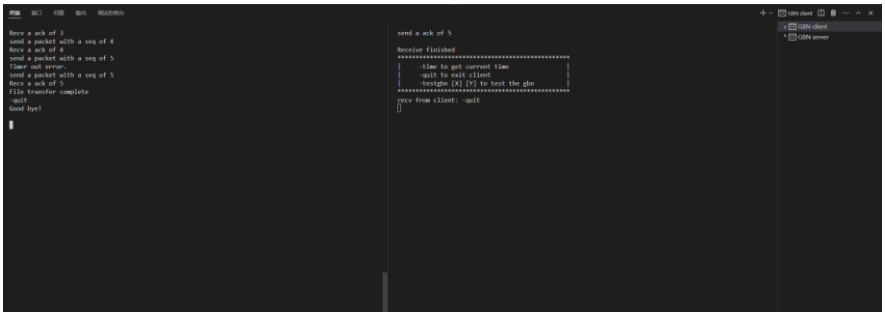
服务端 -> 客户端，可以发现接收的文件和发送的文件内容一致，同时命令行输出表明成功模拟了数据丢失和 ack 丢失



客户端 -> 服务端，可以发现接收的文件和发送的文件内容一致，同时命令行输出表明成功模拟了数据丢失和 ack 丢失



3. 退出功能

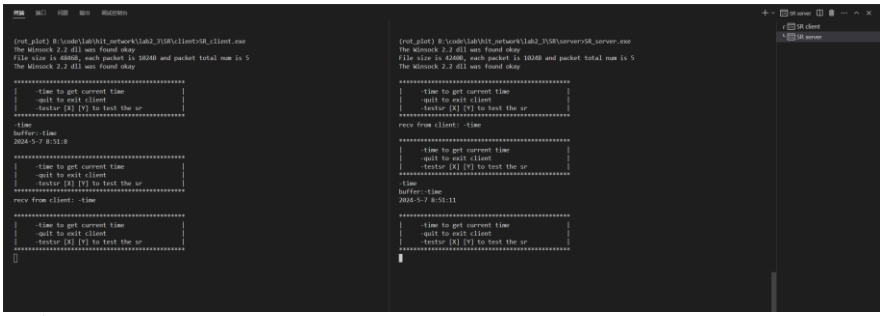


停-等协议结果展示

1. 由于停-等协议与GBN协议共用一个代码，结果一致，再次不重复展示结果

SR协议结果展示

1. 获取时间（双向）

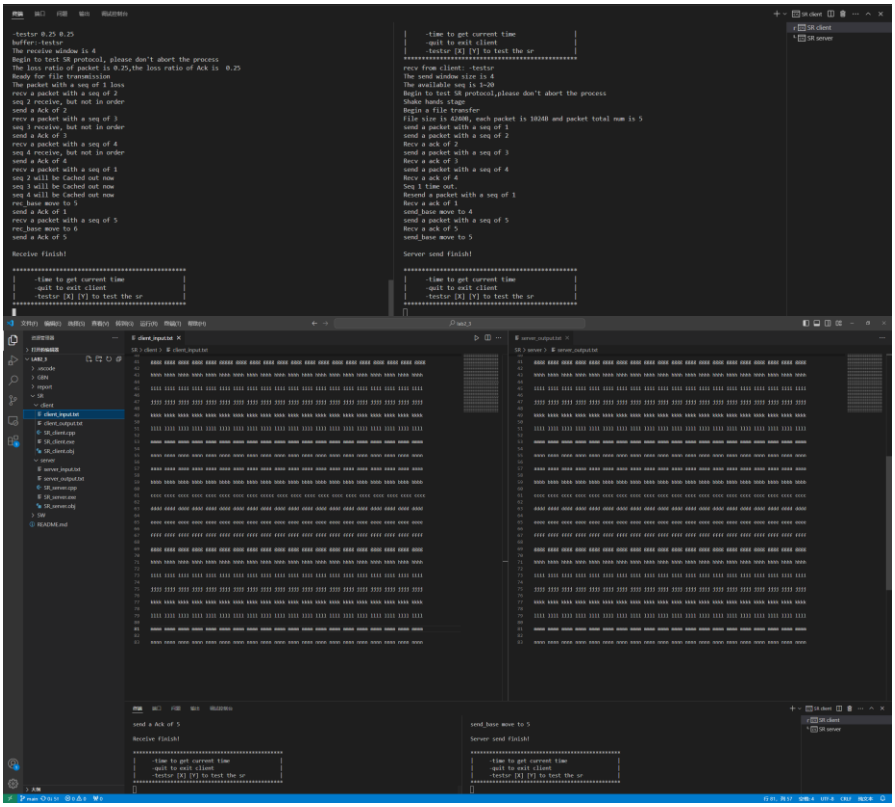


2. 双向传递

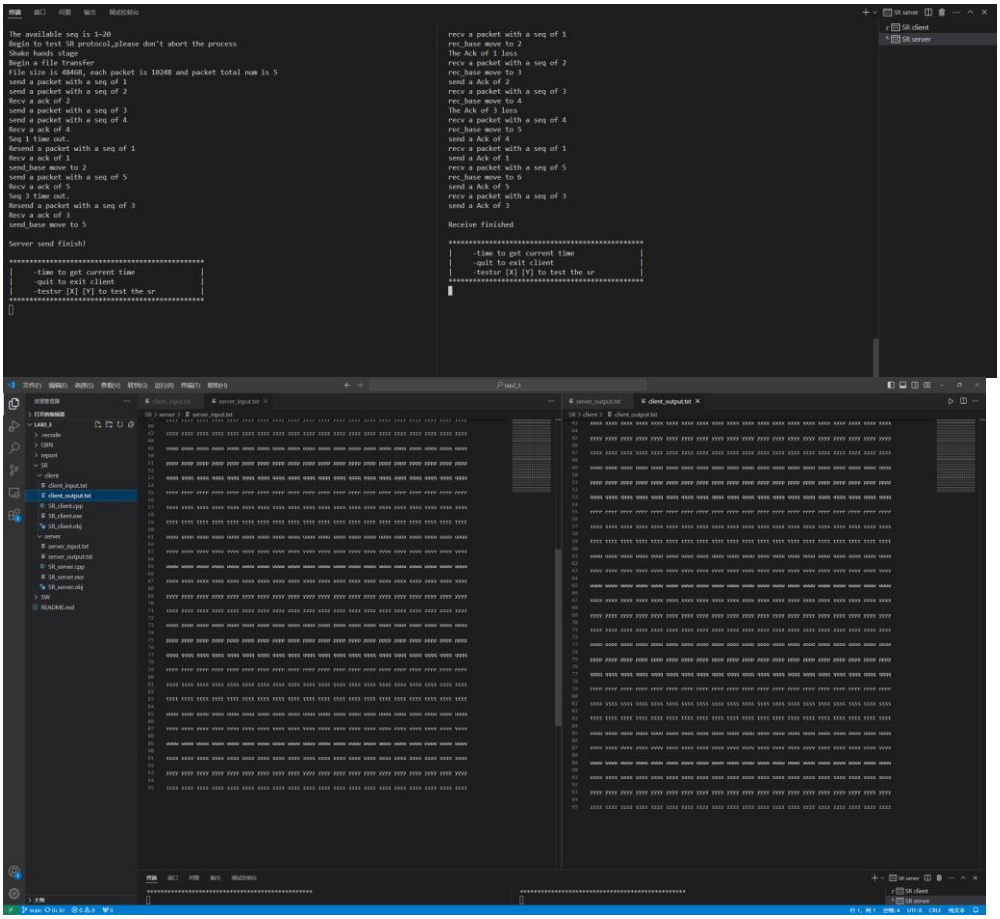
服务端 -> 客户端，可以发现接收的文件和发送的文件内容一致

同时命令行输出分析如下：首先窗口在1，接着接收到数据2并存入缓存中，接着接收到数据3并存入缓存中，接着接收到数据4并存入到缓存中，接着接收到数据1并存入到缓存中，此时缓存中的分组有序，便将缓存中的1、2、3、4写入到文

件中，同时将窗口移动至5。随后接收到数据5，由于有序所以存入缓存后便马上写入到了文件中，同时窗口移动到6，这个时候文件传输完毕



客户端 -> 服务端，可以发现接收的文件和发送的文件内容一致



3. 退出

问题讨论：

问题：在实验过程中，经常会出现文件传输完毕，但接收到的文件内容和发送的内容不一致的问题，有时甚至会出现死循环的进行传输的问题。通常文件内容不一致表现为有部分内容重复。

解决：经过长时间的debug我发现，出现这样的原因主要是代码逻辑的问题，特别是超时重传函数的编写和ack处理函数的编写。由于用于分组的序号是首尾相接的（即最后一个序号与第一个序号相接），而当窗口较长（跨越了头尾）时，需要单独考虑，以正确的处理这类特殊问题

心得体会：

1. 加深了我对UDP传输协议的理解：UDP协议是无连接的传输协议
2. 掌握了停等协议、GBN协议、SR协议的实现方式
3. 更加熟悉了Socket编程的相关方法
4. 了解到了一些编程技巧，例如：使用随机数的方式模拟数据丢失、使用计数器模拟计时器