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	可靠数据传输协议-GBN 协议的设计与实现				
姓名	孙月晴		院系	计算机科学与技术	
班级	1603104		学号	1160300901	
任课教师	刘亚维		指导教师	刘亚维	
实验地点	格物 213		实验时间	2018 年 10 月 31 日	
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					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

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 GBN 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 GBN 协议，支持双向数据传输；（选作内容，加分项目，可以当堂完成或课下完成）

4）将所设计的 GBN 协议改进为 SR 协议。（选作内容，加分项目， 可以当堂完成或课下完成）

实验原理：

(1) GBN 协议数据分组格式，确认分组格式，各个域作用

服务器端：在以太网中，数据帧的 MTU 为 1500 字节，所以 UDP 数据报的数据部分应小于 1472 字节（除去 IP 头部 20 字节与 UDP 头的 8 字节），定义 UDP 数据报的数据部分格式为：

SEQ	DATA	0

Seq 为 1 个字节，取值为 1~20（因为设计的窗口大小为 20）；

Data=1024 个字节，为传输的数据；

客户端：ACK 数据帧定义：

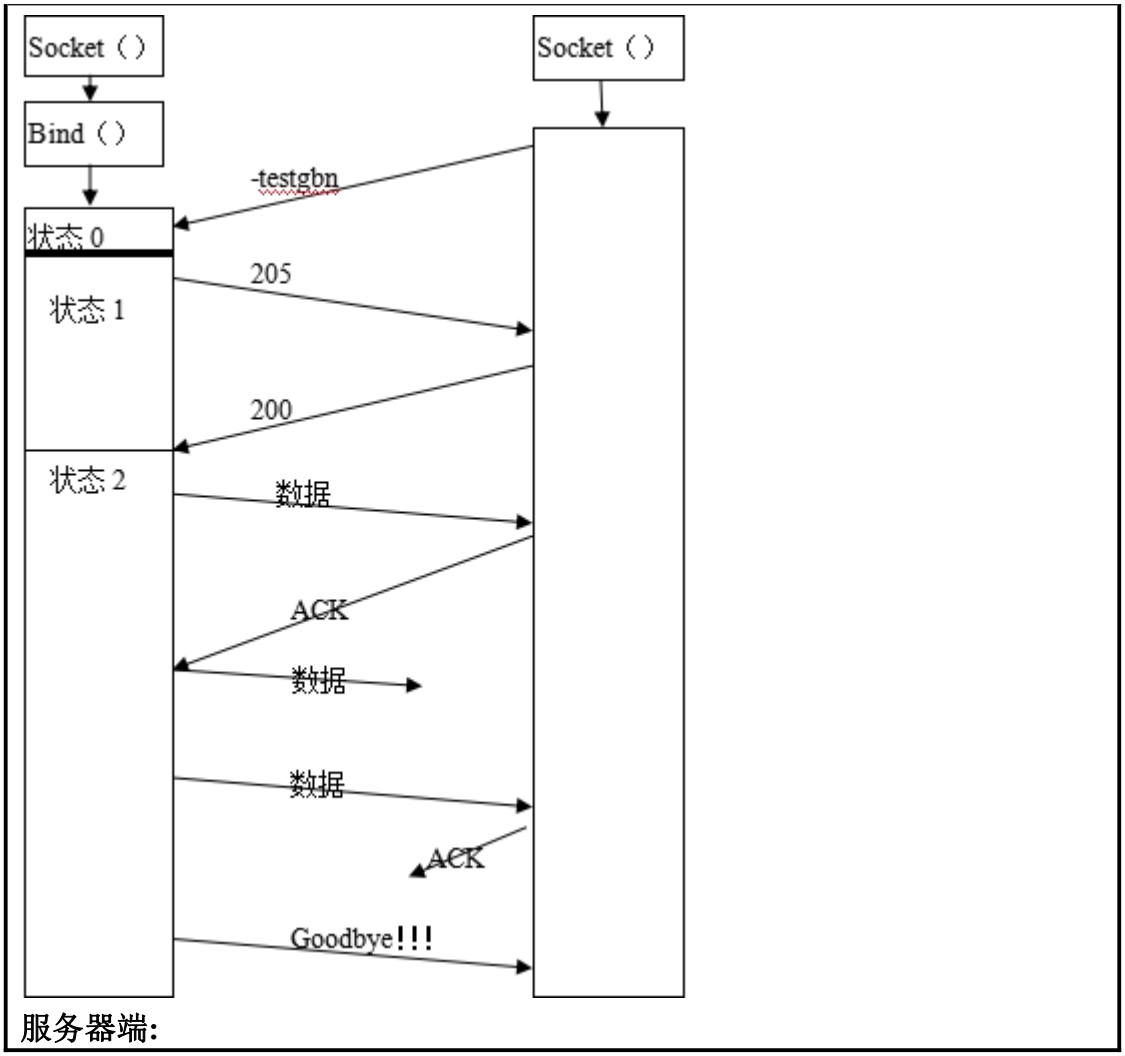
ACK	\0

由于是从服务器端到客户端的单向数据传输，因此 ACK 数据帧不包含任何数据，只需要将 ACK 发送给服务器端即可。

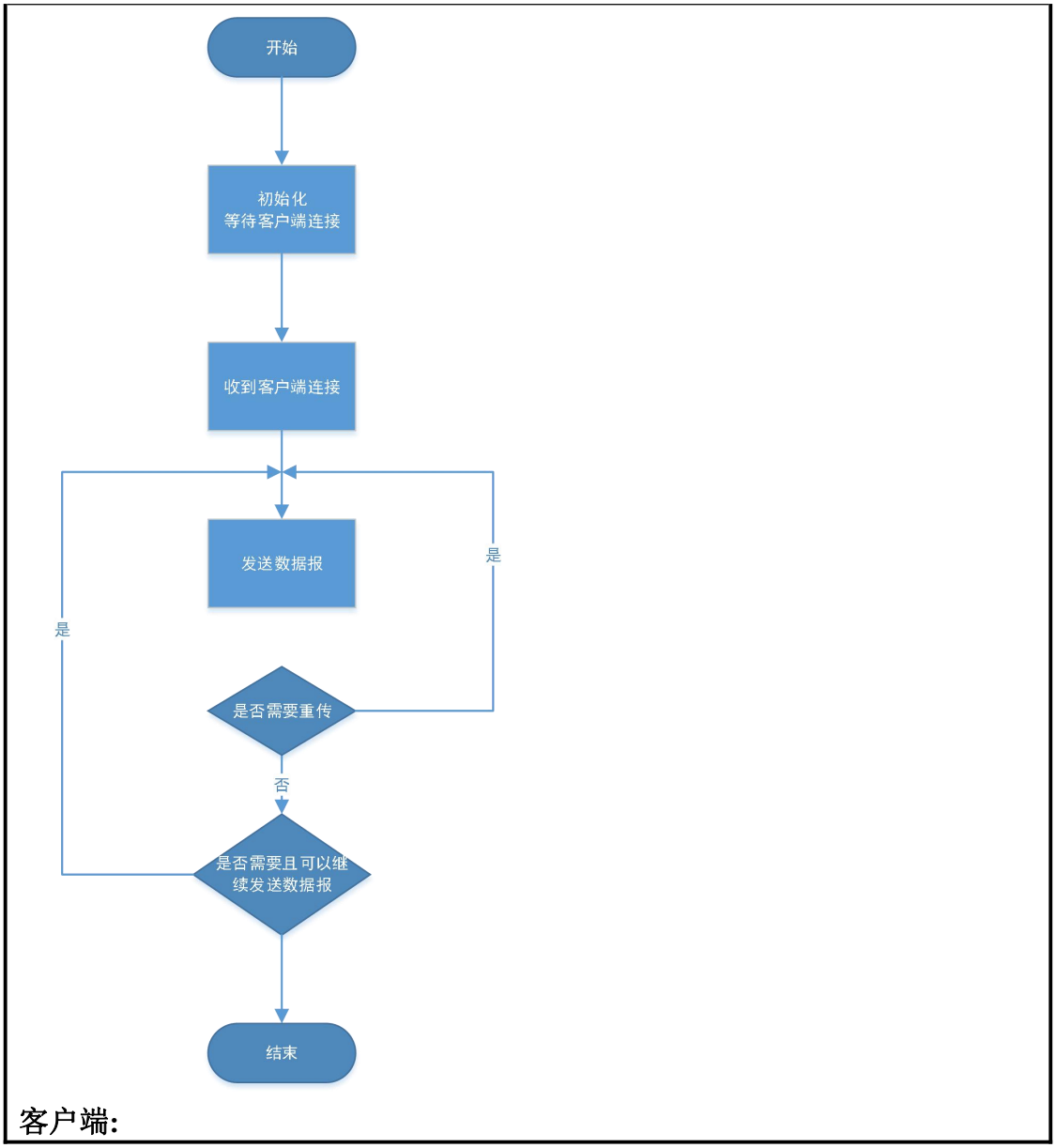
ACK 字段为 1 个字节，表示序列号数值；

末尾放入 0，表示数据结束。

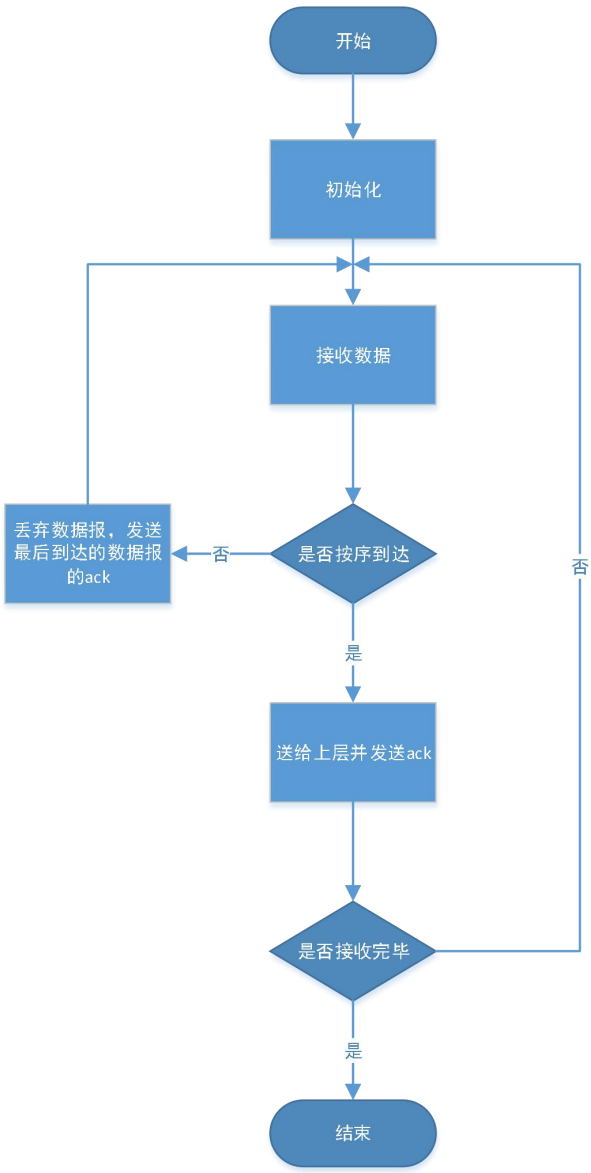
(2) 协议两端程序流程图


Socket（）
Socket（）
Bind（）
-testgbn
状态 0
205
状态 1
200
状态 2
数据
ACK
数据
数据
ACK
Goodbye!!!

服务器端:

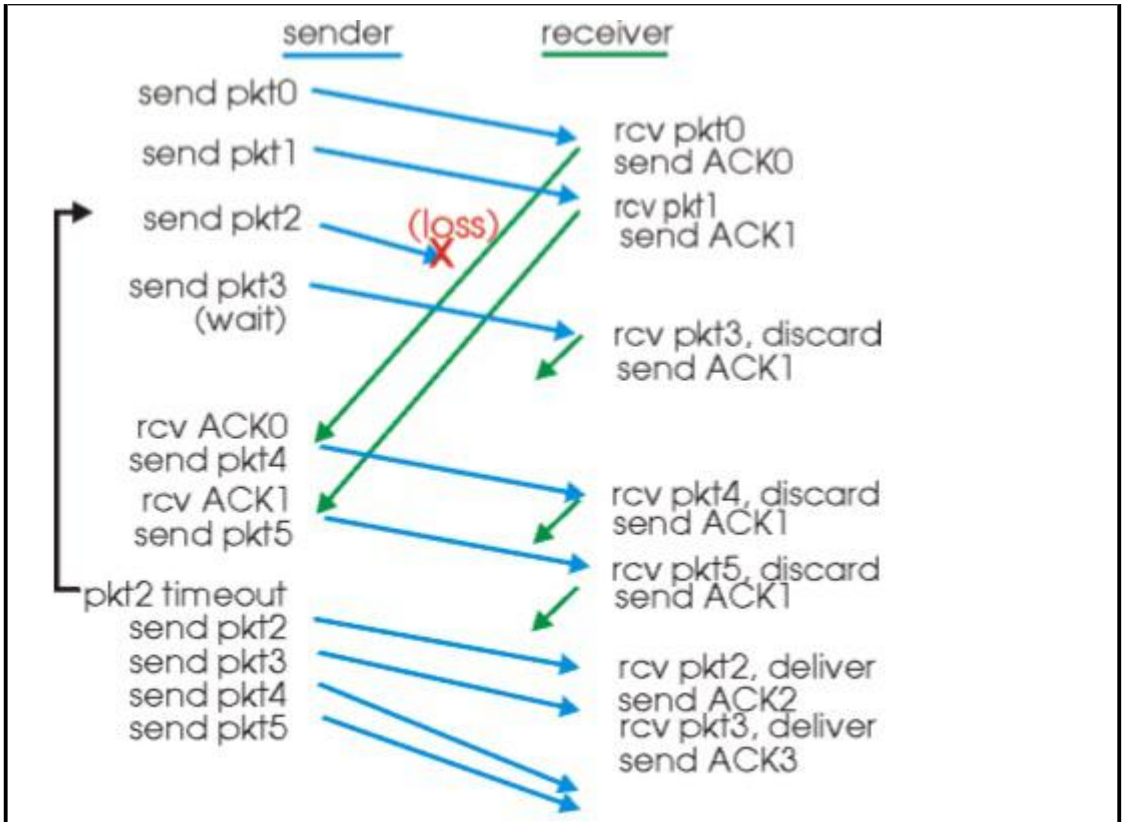


客户端:

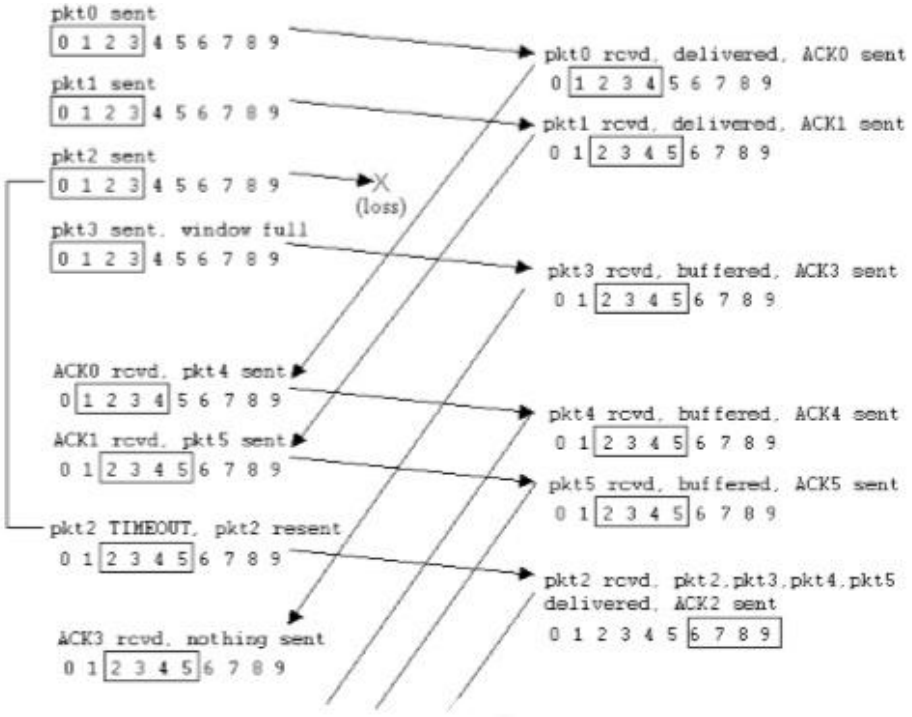


(3) 协议典型交互过程

GBN 协议中，发送方连续发送若干个数据帧，且在每发送完一个数据帧时都要设置超时定时器。只要在所设置的超时时间内仍收到确认帧，就要重发相应的数据帧。如：当发送方发送了 N 个帧后，若发现该 N 帧的前一个帧在计时器超时后仍未返回其确认信息，则该帧被判为出错或丢失，此时发送方就不得不重新发送出错帧及其后的 N 帧。



SR 协议是当接收方发现某帧出错后，其后继续送来的正确的帧虽然不能立即递交给接收方的高层，但接收方可收下来，存放在一个缓冲区中，同时要求发送方重新传送出错的那一帧。一旦收到重新传来的帧后，就可以原已存于缓冲区中的其余帧一并按正确的顺序递交高层。



实验过程：

1. 对于双向传输:

由于为双向通信，实际C和S的身份是互相切换的。服务器同时也作为客户端,所以增加了lossInLossRatio,根据丢失率随机丢失数据,在数据传输阶段,如果未接收的到客户端发送的ack回复,服务器重发相应的报文段。客户端也作为服务器,所以增加了客户端发送数据的端口号,增加了发送数据的逻辑,比如收到的ack,等待确认的ack以及发送的包的总数。

2. 数据分组丢失验证模拟方法

丢失模拟方式在客户端和服务器都有实现，以单向传输为例,如果数据包丢失，则客户端认为没有接受到；如果 ACK 丢失，则客户端不向服务器发送 ACK 数据包。在 lossInLossRatio 函数，将填写的 X 或者 Y 丢失率作为参数，在该函数中在 0-100 中随机生成一个数，如果该数小于丢失率*100 则认为丢失。

验证方式，在服务器端将 sever.txt 文件作为要传输的数据，如果数据传输完成后在客户端收到的数据是准确无误的，则认为是 GBN/SR 是可靠的。

3. 程序实现的主要函数及其主要作用

① void getCurTime(char *ptime)

// Qualifier: 获取当前系统时间，结果存入 ptime 中

② bool seqIsAvailable()

// Qualifier: 判断当前序列号curSeq是否可用，即看curseq是否在窗口内，是否已经发送过了

③ void timeoutHandler()

// Qualifier:超时重传处理函数，滑动窗口内的数据帧都要重传

④ void ackHandler(char c)

// Qualifier: 收到 ack，累积确认，取数据帧的第一个字节

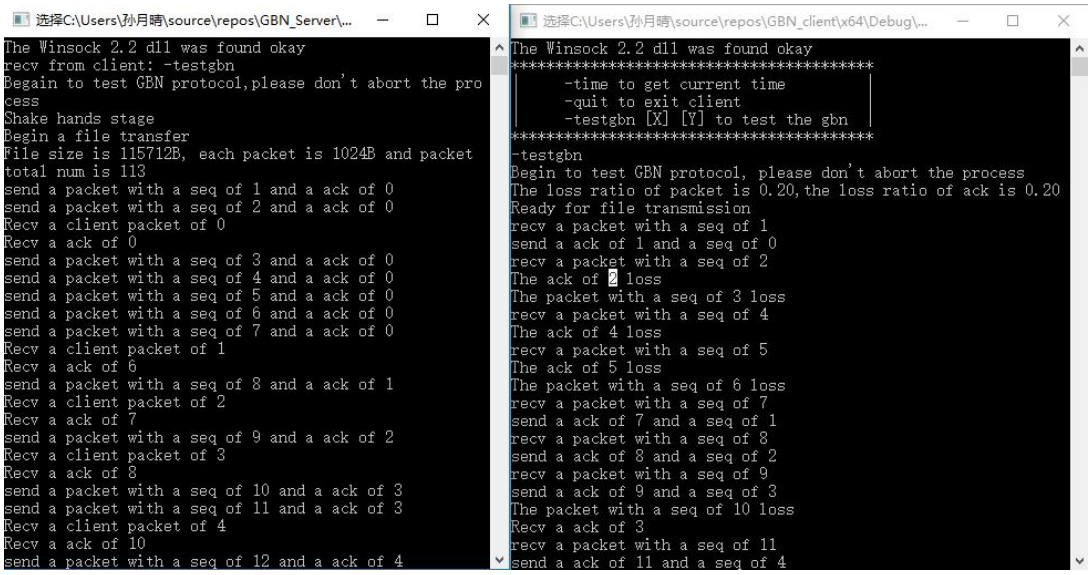
⑤ BOOL lossInLossRatio(float lossRatio)

// Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回 TRUE，否则返回 FALSE

实验结果：

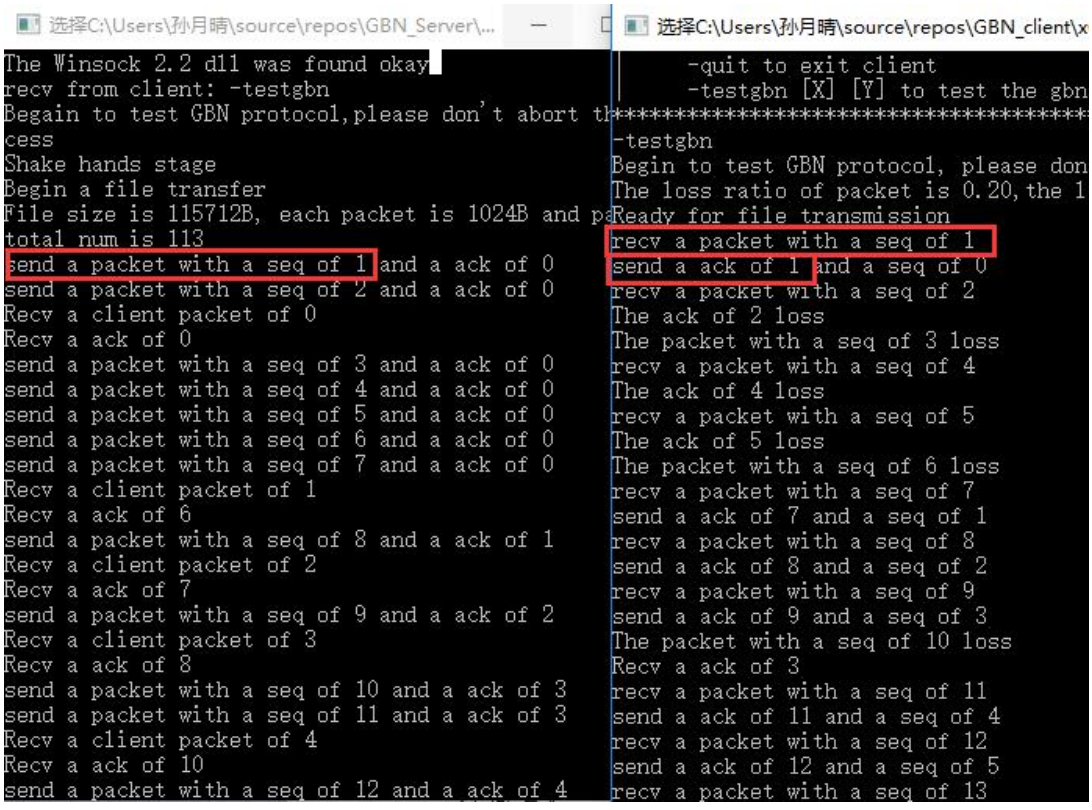
双向GBN:

1. 实现了服务器和客户端的双向数据传输,如图所示,客户端和服务器同时发送和接收数据。

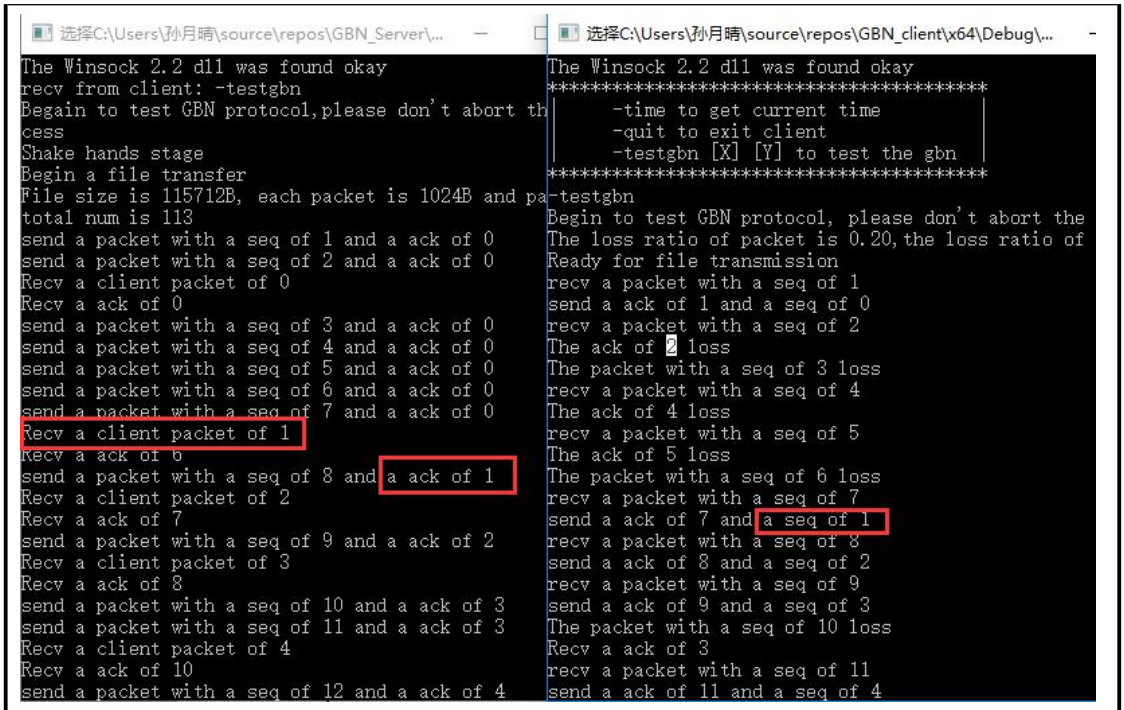


2. -testgbn [X] [Y]命令，[X],[Y]均为[0,1]的小数，其中：[X]表示客户端的丢包率，模拟网络中报文丢失；[Y]表示客户端的ACK的丢失率。（使用随机函数完成）。不输入，则默认丢失率均为0.2。

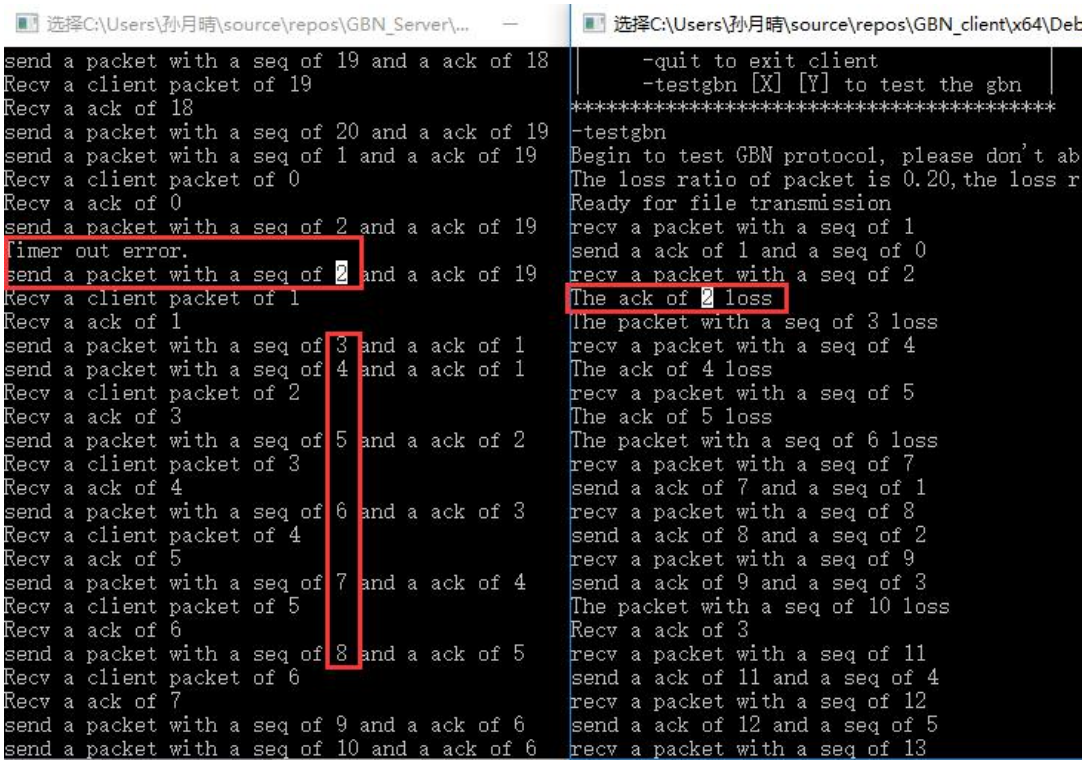
如图,服务器向客户端发送序列1,客户端收到序列1并回复ack1,实现了基本功能



3. 如图,客户端向服务器发送序列1,服务器收到序列1并给客户端发送ack1，可见改进设计的GBN协议支持双向数据传输

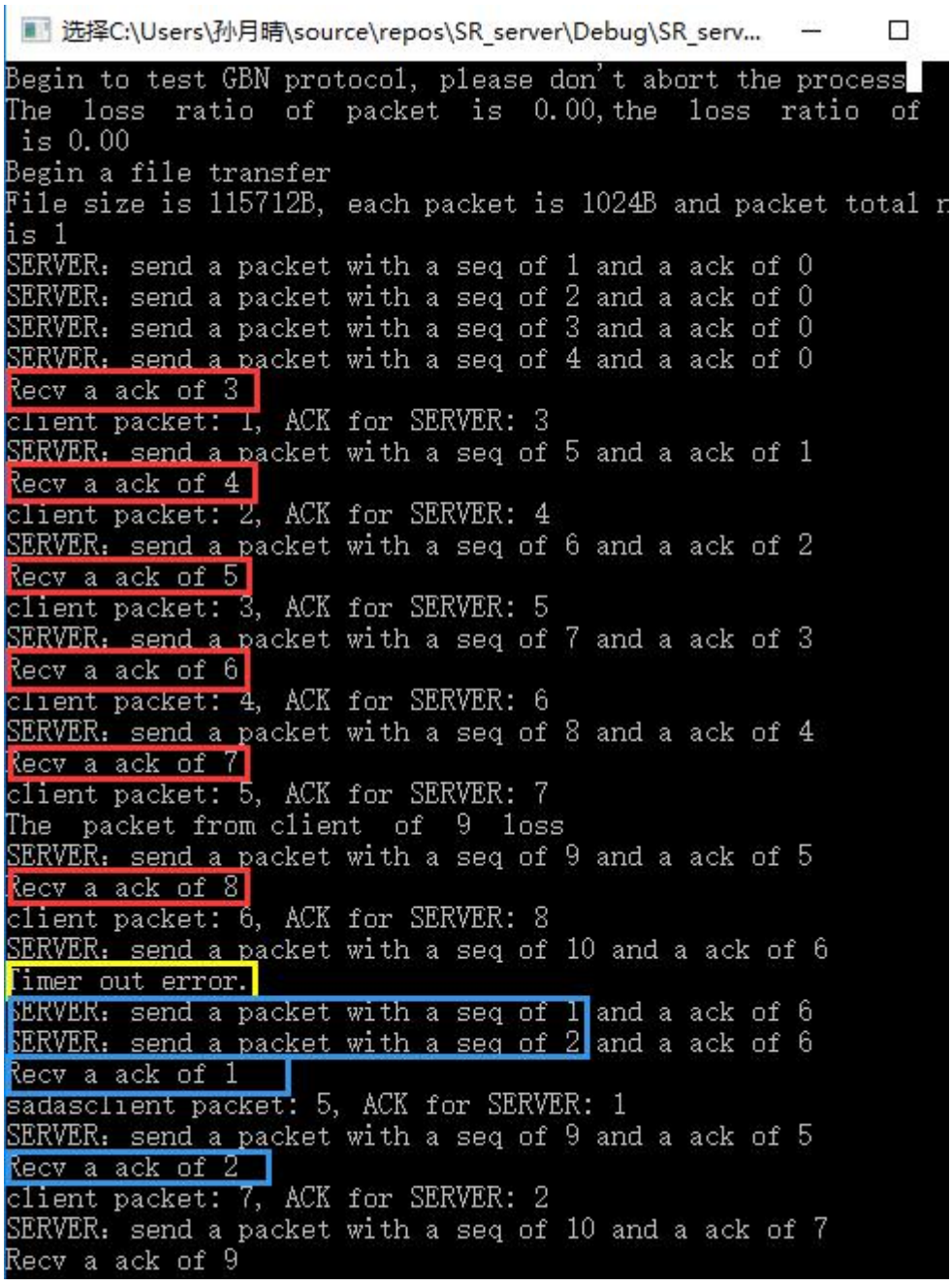


4. 如图,模拟数据包的丢失和超时,客户端向服务器发送的ack2丢失,超时之后服务器重传序列2以及序列2之后的报文

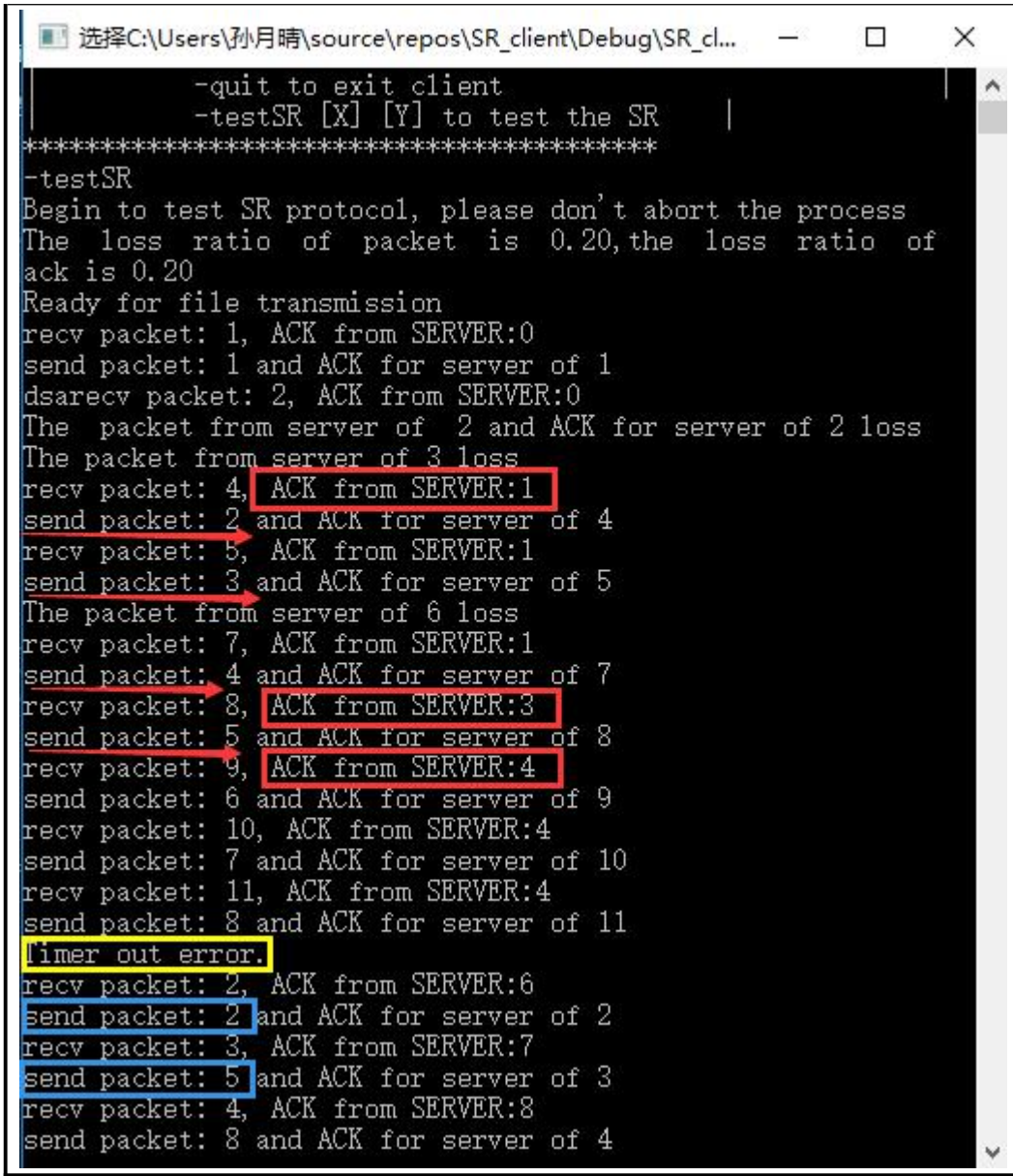


双向SR:

1. 服务器向客户端发送报文 1-8,收到 ack3-ack8,丢失了 ack1 和 ack2,超时之后重传序列 1 和序列 2,随后收到了 ack1 和 ack2,之后继续发送序列号 9 之后的报文



2. 客户端向服务器发送报文1-5,收到服务器的ack回复为ack1,ack3,ack4,未收到ack2和ack5,超时之后,重新向服务器发送报文2和5,并且继续发送之后未发送的报文



心得体会：

通过此次实验,我对 GBR 和 SR 协议的有了更加深刻的认识，SR 相比 GBR 在效率上有了很大的提高,也对滑动窗口这种机制有了更加切实的体会。

GBN 协议中的一个最大的问题就是，当窗口大小 N 非常大时（例如为 1000 的话），而最开始的分组 0 错误了，就需要重传之后的 999 个分组，因此造成了时间上的损失，延时增加。但对于 SR 协议而言，发送方和接收方并不是总能看到相同的结果，这也就说明两者的窗口不总是一致的。而 SR 协议会面临到的一个最重要的问题就是面临有限序号范围的现实时，发送方和接收方的窗口不同步会产生严重后果。这样的后果会导致接收方无法判断出该分组是一次重传还是下一个新的分组的，因而就 SR 的窗口不能够很大。

当某个分组序列错误较少，窗口较大的时候，应该是更适合用SR协议进行传输，因为这样可以减少很多不必要的重传，充分利用了信道；当某个分组序列错误较多，并且窗口较小的时候，应该更适合用GBN协议，因为这样接收方不需要缓存分组，节省了空间，并且该分组序列错误较多，相对SR来讲，也不会使得GBN的效率过于低下。

详细注释源程序:

双向GBN -- 服务器:

```
// GBN_Server.cpp: 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <stdlib.h>
#include <time.h>
#include <WinSock2.h>
#include <fstream>

#pragma comment(lib,"ws2_32.lib")
#define SERVER_PORT 12340 //端口号
#define SERVER_IP "0.0.0.0" //IP 地址
const int BUFFER_LENGTH = 1026; //缓冲区大小，（以太网中 UDP 的数据 帧中包长度应小于 1480 字节）

const int SEND_WIND_SIZE = 10;
//发送窗口大小为 10，GBN 中应满足 W + 1 <= N（W 为发送窗口大小，N 为序列号个数）
//本例取序列号 0...19 共 20 个
//如果将窗口大小设为 1，则为停-等协议
const int SEQ_SIZE = 20;
//序列号的个数，从 0~19 共计 20 个
//由于发送数据第一个字节如果值为 0，则数据会发送 失败
//因此接收端序列号为 1~20，与发送端一一对应
BOOL ack[SEQ_SIZE];//收到 ack 情况，对应 0~19 的 ack
int curSeq;//当前数据包的 seq
int curAck;//当前等待确认的 ack
int totalSeq;//收到的包的总数
int totalPacket;//需要发送的包总数

//*************************************
// Method: getCurTime
// FullName: getCurTime
// Access: public
// Returns: void
// Qualifier: 获取当前系统时间，结果存入 ptime 中
// Parameter: char * ptime
//*************************************
void getCurTime(char *ptime)
{
    char buffer[128];
    memset(buffer, 0, sizeof(buffer));
```

```
    time_t c_time;
    struct tm *p;
    time(&c_time);
    p = localtime(&c_time);
    sprintf_s(buffer, "%d/%d/%d %d:%d:%d", p->tm_year + 1900, p->tm_mon,
p->tm_mday, p->tm_hour, p->tm_min,
        p->tm_sec); strcpy_s(ptime, sizeof(buffer), buffer);
}
//************************************
// Method: seqIsAvailable
// FullName: seqIsAvailable
// Access: public
// Returns: bool
// Qualifier: 当前序列号 curSeq 是否可用
//************************************
bool seqIsAvailable()
{
    int step;
    step = curSeq - curAck;
    step = step >= 0 ? step : step + SEQ_SIZE; //序列号是否在当前发送窗口
之内
    if (step >= SEND_WIND_SIZE) {
        return false;
    }
    if (ack[curSeq]) {
        return true;
    }
    return false;
}

//************************************
// Method: timeoutHandler
// FullName: timeoutHandler
// Access: public
// Returns: void
// Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传
//************************************

void timeoutHandler()
{
    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 -= SEND_WIND_SIZE;
    curSeq = curAck;
}
//***********************************
// Method: ackHandler
// FullName: ackHandler
// Access: public
// Returns: void
// Qualifier: 收到 ack，累积确认，取数据帧的第一个字节
//由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一 了，此处需要减一还原
// Parameter: char c
//***********************************
void ackHandler(char c)
{
    unsigned char index = (unsigned char)c - 1; //序列号减一
    printf("Recv a ack of %d\n", index);//序号发送方和接收方应该统一，发送的是+1 过的，接收方 Ack+1 过的，在这里打印没必要还原
    if (curAck <= index) {
        for (int i = curAck; i <= index; ++i) {
            ack[i] = TRUE;
        }
        curAck = (index + 1) % SEQ_SIZE;
    }
    else { //ack 超过了最大值，回到了 curAck 的左边
        for (int i = curAck; i< SEQ_SIZE; ++i) {
            ack[i] = TRUE;
        }
        for (int i = 0; i <= index; ++i) {
            ack[i] = TRUE;
        }
        curAck = index + 1;
    }
}

int main(int argc, char* argv[]) { //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 sockServer = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP); //设
置套接字为非阻塞模式
    int iMode = 1; //1：非阻塞，0：阻塞
    ioctlsocket(sockServer, FIONBIO, (u_long FAR*) &iMode);//非阻塞设置
    SOCKADDR_IN addrServer; //服务器地址
                            //addrServer.sin_addr.S_un.S_addr =
inet_addr(SERVER_IP);
    addrServer.sin_addr.S_un.S_addr = htonl(INADDR_ANY);//两者均可
    addrServer.sin_family = AF_INET;
    addrServer.sin_port = htons(SERVER_PORT);
    unsigned short seq;//包的序列号
    unsigned short recvSeq;//接收窗口大小为 1，已确认的序列号
    unsigned short waitSeq;//等待的序列号
    err = bind(sockServer, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
    if (err) {
        err = GetLastError();
        printf("Could not bind the port %d for socket.Error code is %d\n",
SERVER_PORT, err);
        WSACleanup(); return -1;
    }
    SOCKADDR_IN addrClient; //客户端地址
    int length = sizeof(SOCKADDR);
    char buffer[BUFFER_LENGTH]; //数据发送接收缓冲区
    ZeroMemory(buffer, sizeof(buffer)); //将测试数据读入内存
    std::ifstream icin; icin.open("test.txt");
    char data[1024 * 113]; ZeroMemory(data, sizeof(data));
    icin.read(data, 1024 * 113);
    icin.close();
    totalPacket = sizeof(data) / 1024;
    int recvSize;
    for (int i = 0; i < SEQ_SIZE; ++i) {
        ack[i] = TRUE;
    }
```

```
    while (true) { //非阻塞接收，若没有收到数据，返回值为-1
        recvSize = recvfrom(sockServer, buffer, BUFFER_LENGTH, 0,
((SOCKADDR*)&addrClient), &length);
        if (recvSize < 0) {
            Sleep(200);
            continue;
        }
        printf("recv from client: %s\n", buffer);
        if (strcmp(buffer, "-time") == 0) {
            getCurTime(buffer);
        }
        else if (strcmp(buffer, "-quit") == 0) {
            strcpy_s(buffer, strlen("Good bye!") + 1, "Good bye!");
        }
        else if (strcmp(buffer, "-testgbn") == 0) {
            //进入 gbn 测试阶段
            //首先 server（server 处于 0 状态）向 client 发送 205 状态码
（server 进入 1 状态）
            //server 等待 client 回复 200 状态码，如果收到（server 进入 2
状态）， 则开始传输文件，否则延时等待直至超时\
            //在文件传输阶段，server 发送窗口大小设为
            ZeroMemory(buffer, sizeof(buffer));
            int recvSize;
            int waitCount = 0;
            printf("Begain to test GBN protocol,please don't abort the
process\n");
            //加入了一个握手阶段
            //首先服务器向客户端发送一个 205 大小的状态码（我自己定义的） 表
示服务器准备好了，可以发送数据
            //客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准 备
好了，可以接收数据了
            //服务器收到 200 状态码之后，就开始使用 GBN 发送数据了
            printf("Shake hands stage\n");
            int stage = 0;
            bool runFlag = true;
            while (runFlag) {
                switch (stage) {
                case 0://发送 205 阶段
                    buffer[0] = 205;
                    sendto(sockServer, buffer, strlen(buffer) + 1, 0,
(SOCKADDR*)&addrClient, sizeof(SOCKADDR));
                    Sleep(100);
                    stage = 1;
                    break;
```

```
                case 1://等待接收 200 阶段，没有收到则计数器+1，超时则 放弃此次“连接”，等待从第一步开始
                    recvSize = recvfrom(sockServer, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrClient), &length);
                    if (recvSize < 0) {
                        ++waitCount;
                        if (waitCount > 20) {
                            runFlag = false;
                            printf("Timeout error\n");
                            break;
                        }
                        Sleep(500);
                        continue;
                    }
                    else {
                        if ((unsigned char)buffer[0] == 200) {
                            printf("Begin a file transfer\n");
                            printf("File size is %dB, each packet is 1024B and packet total num is %d\n", sizeof(data), totalPacket);
                            curSeq = 0;
                            curAck = 0;
                            totalSeq = 0;
                            waitCount = 0;
                            stage = 2;
                            recvSeq = 0;
                            waitSeq = 1;
                        }
                    }
                    break;
                case 2://数据传输阶段
                    if (seqIsAvailable()) {
                        //发送给客户端的序列号从 1 开始
                        buffer[0] = curSeq + 1;//seq
                        ack[curSeq] = FALSE;
                        //数据发送的过程中应该判断是否传输完成
                        //为简化过程此处并未实现
                        buffer[1] = recvSeq;
                        memcpy(&buffer[2], data + 1024 * totalSeq, 1024);
                        printf("send a packet with a seq of %d and a ack of %d \n", curSeq + 1, buffer[1]);
                        sendto(sockServer, buffer, BUFFER_LENGTH, 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++; //20 次等待 ack 则超时重传
                        if (waitCount > 20) {
                            timeoutHandler();
                            waitCount = 0;
                        }
                    }
                    else {
                        //收到 客户端 ack 和客户端发送的数据；
                        seq = (unsigned short)buffer[1];
                        if (waitSeq - seq == 0)//收到的序列号正好是所等待
的序列号：接收 ACK，数据
                        {
                            recvSeq = seq;
                            ++waitSeq;
                            if (waitSeq == 20) {
                                waitSeq = 1;
                            }
                            printf("Recv a client packet of %d\n",
seq);//打印来自客户端数据的序号
                            ackHandler(buffer[0]);
                        }
                        else
                        {//没有接收到数据
                            printf("Recv a client packet of %d\n",
seq);//打印来自客户端数据的序号
                            ackHandler(buffer[0]);
                        }
                        Sleep(500);
                        break;
                    }
                }
            }
        }
        sendto(sockServer, buffer, strlen(buffer) + 1, 0,
(SOCKADDR*)&addrClient, sizeof(SOCKADDR));
        Sleep(500);
    }
    //关闭套接字，卸载库
```

```
    closesocket(sockServer);
    WSACleanup();
    return 0;
}
```

双向GBN -- 客户端:

```
// GBN_client.cpp: 定义控制台应用程序的入口点。
//

#include "stdafx.h"


#include <stdlib.h>
#include <WinSock2.h>
#include <time.h>
#include <cstdio>
#include <fstream>
#pragma comment(lib,"ws2_32.lib")
#define SERVER_PORT 12340 //接收数据的端口号
#define SERVER_IP "127.0.0.1" // 服务器的 IP 地址
const int BUFFER_LENGTH = 1026;
const int SEQ_SIZE = 20;//接收端序列号个数，为 1~20
BOOL cl_ack[SEQ_SIZE];//收到 ack 情况，对应 0~19 的 ack
int cl_curSeq;//当前数据包的 seq
int cl_curAck;//当前等待确认的 ack
int cl_totalSeq;//收到的包的总数
int cl_totalPacket;//需要发送的包总数
const int SEND_WIND_SIZE = 10;
//************************************
// Method: timeoutHandler
// FullName: timeoutHandler
// Access: public
// Returns: void
// Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传
//************************************

void timeoutHandler()//好像没什么用
{
    printf("Timer out error.\n");
    int index;
    for (int i = 0; i< SEND_WIND_SIZE; ++i) {
        index = (i + cl_curAck) % SEQ_SIZE;
        cl_ack[index] = TRUE;
    }
```

```
    cl_totalSeq -= SEND_WIND_SIZE;
    cl_curSeq = cl_curAck;
}
//***********************************
// Method: ackHandler
// FullName: ackHandler
// Access: public
// Returns: void
// Qualifier: 收到 ack，累积确认，取数据帧的第一个字节
//由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一 了，此处需要减一还原
// Parameter: char c
//***********************************
void ackHandler(char c)
{
    if (c == 0)
    {
        //printf("server's first package do not have ack!\n");
        return;
    }
    unsigned char index = (unsigned char)c - 1; //序列号减一
    printf("Recv a ack of %d\n", index + 1);//序号发送方和接收方应该统一，发送的是+1 过的，接收方 Ack+1 过的，在这里打印没必要还原
    if (cl_curAck <= index) {
        for (int i = cl_curAck; i <= index; ++i) {
            cl_ack[i] = TRUE;
        }
        cl_curAck = (index + 1) % SEQ_SIZE;
    }
    else { //ack 超过了最大值，回到了 curAck 的左边
        for (int i = cl_curAck; i< SEQ_SIZE; ++i) {
            cl_ack[i] = TRUE;
        }
        for (int i = 0; i <= index; ++i) {
            cl_ack[i] = TRUE;
        }
        cl_curAck = index + 1;
    }
}
bool seqIsAvailable()
{
    int step;
    step = cl_curSeq - cl_curAck;
    step = step >= 0 ? step : step + SEQ_SIZE; //序列号是否在当前发送窗口
```

```
之内
    if (step >= SEND_WIND_SIZE) {
        return false;
    }
    if (cl_ack[cl_curSeq]) {
        return true;
    }
    return false;
}

/****************************************************************/
/*
-time 从服务器端获取当前时间
-quit 退出客户端
-testgbn [X] 测试 GBN 协议实现可靠数据传输
[X] [0,1] 模拟数据包丢失的概率
[Y] [0,1] 模拟 ACK 丢失的概率 */
/****************************************************************/
void printTips() {
    printf("*************************************\n");
    printf("| -time to get current time |\n");
    printf("| -quit to exit client |\n");
    printf("| -testgbn [X] [Y] to test the gbn |\n");
    printf("*************************************\n");
}
//***********************************
// Method: lossInLossRatio
// FullName: lossInLossRatio
// Access: public
// Returns: BOOL
// Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回 TRUE，否则
返回 FALSE
// Parameter: float lossRatio [0,1]
//***********************************
BOOL lossInLossRatio(float lossRatio) {
    int lossBound = (int)(lossRatio * 100);
    int r = rand() % 101;
    if (r <= lossBound) {
        return TRUE;
    }
    return FALSE;
}
int main(int argc, char* argv[]) {
    //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 addrServer;
    addrServer.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
    addrServer.sin_family = AF_INET;
    addrServer.sin_port = htons(SERVER_PORT); //接收缓冲区
    char buffer[BUFFER_LENGTH];
    ZeroMemory(buffer, sizeof(buffer));
    int len = sizeof(SOCKADDR);
    //为了测试与服务器的连接，可以使用 -time 命令从服务器端获得当前 时间
    //使用 -testgbn [X] [Y] 测试 GBN 其中[X]表示数据包丢失概率
    // [Y]表示 ACK 丢包概率
    printTips(); int ret;
    int interval = 1;
    //收到数据包之后返回 ack 的间隔，默认为 1 表示每个都 返回 ack，0 或者负数
均表示所有的都不返回 ack
    char cmd[128];
    float packetLossRatio = 0.2;
    //默认包丢失率 0.2
    float ackLossRatio = 0.2;
    //默认 ACK 丢失率 0.2
    //用时间作为随机种子，放在循环的最外面

    std::ifstream icin; icin.open("test.txt");
    char data[1024 * 113]; ZeroMemory(data, sizeof(data));
```

```
    icin.read(data, 1024 * 113);
    icin.close();
    cl_totalPacket = sizeof(data) / 1024;
    for (int i = 0; i < SEQ_SIZE; ++i) {
        cl_ack[i] = TRUE;
    }

    srand((unsigned)time(NULL));

    while (true) {
        gets_s(buffer);
        ret = sscanf(buffer, "%s%f%f", &cmd, &packetLossRatio,
&ackLossRatio);
        //开始 GBN 测试，使用 GBN 协议实现 UDP 可靠文件传输
        if (!strcmp(cmd, "-testgbn")) {
            printf("%s\n", "Begin to test GBN protocol, please don't abort
the process");
            printf("The loss ratio of packet is %.2f,the loss ratio of ack
is %.2f\n", packetLossRatio, ackLossRatio);
            int waitCount = 0;
            int stage = 0;
            BOOL b;
            unsigned char u_code;//状态码
            unsigned short seq;//包的序列号
            unsigned short recvSeq;//接收窗口大小为 1，已确认的序列号
            unsigned short waitSeq;//等待的序列号
            sendto(socketClient, "-testgbn", strlen(" - testgbn") + 1, 0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
            while (true) {
                //等待 server 回复设置 UDP 为阻塞模式
                recvfrom(socketClient, buffer, BUFFER_LENGTH, 0,
(SOCKADDR*)&addrServer, &len);
                switch (stage) {
                case 0://等待握手阶段
                    u_code = (unsigned char)buffer[0];
                    if ((unsigned char)buffer[0] == 205) {
                        printf("Ready for file transmission\n");
                        buffer[0] = 200;
                        buffer[1] = '\0';
                        sendto(socketClient, buffer, 2, 0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
                        stage = 1;
                        recvSeq = 0;
                        waitSeq = 1;
```

```
                    cl_curSeq = 0;
                    cl_curAck = 0;
                    cl_totalSeq = 0;
                }
                break;
            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("The packet with a seq of %d loss\n", seq);
                    ackHandler((unsigned short)buffer[1]);
                    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 接收时同时发送构造数据并发送
                if (waitSeq - seq == 0) {
                    ++waitSeq;
                    if (waitSeq == 21) {
                        waitSeq = 1;
                    }
                    //输出数据
                    //printf("%s\n",&buffer[1]);

                    if (seqIsAvailable())
                    {
                        buffer[0] = seq;
                        buffer[1] = cl_curSeq;
                        cl_ack[cl_curSeq] = FALSE;
                        memcpy(&buffer[2], data + 1024 * cl_totalSeq, 1024);
                        recvSeq = seq;
                        ++cl_curSeq;
                        cl_curSeq %= SEQ_SIZE;
                        ++cl_totalSeq;
                        Sleep(500);
                    }
                }
                else {
                    //如果当前一个包都没有收到，则等待 Seq 为 1 的数 据包，跳出循环，不返回 ACK（因为并没有上一个正确的 ACK）
                    if (recvSeq == 0) {
                        continue;
```

```
                        }
                        if (seqIsAvailable())
                        {
                            buffer[0] = seq;
                            buffer[1] = cl_curSeq;
                            cl_ack[cl_curSeq] = FALSE;
                            memcpy(&buffer[2], data + 1024 * cl_totalSeq,
1024);
                            recvSeq = seq;
                            ++cl_curSeq;
                            cl_curSeq %= SEQ_SIZE;
                            ++cl_totalSeq;
                        }
                    }
                    b = lossInLossRatio(ackLossRatio);
                    if (b) {
                        printf("The ack of %d loss\n", (unsigned
char)buffer[0]);
                        if (cl_curSeq == 0) {
                            cl_curSeq = 19;
                            cl_ack[cl_curSeq] = TRUE;
                        }
                        else
                        {
                            cl_curSeq--;
                            cl_ack[cl_curSeq] = TRUE;
                        }
                        continue;
                    }
                    sendto(socketClient, buffer, 2, 0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
                    printf("send a ack of %d and a seq of %d\n", (unsigned
char)buffer[0], (unsigned char)buffer[1]);
                    break;
                }
                Sleep(500);
            }
        }
        sendto(socketClient, buffer, strlen(buffer) + 1, 0,
(SOCKADDR*)&addrServer, sizeof(SOCKADDR));
        ret = recvfrom(socketClient, buffer, BUFFER_LENGTH, 0,
(SOCKADDR*)&addrServer, &len);
        printf("%s\n", buffer);
        if (!strcmp(buffer, "Good bye!")) {
```

```
            break;
        }
        printTips();
    } //关闭套接字
    closesocket(socketClient);
    WSACleanup();
    return 0;
}
```

双向 SR -- 服务器:

```
// SR_server.cpp: 定义控制台应用程序的入口点。
//

#include "stdafx.h"


#define _CRT_SECURE_NO_WARNINGS
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <stdlib.h>
#include <time.h>
#include <WinSock2.h>
#include <fstream>
#include<io.h> //C 语言头文件
#include<iostream> //for system();

#pragma comment(lib,"ws2_32.lib")
using namespace std;
#define SERVER_PORT 12340 //端口号
#define SERVER_IP "127.0.0.1" //IP 地址
const int BUFFER_LENGTH = 1026; //缓冲区大小，（以太网中 UDP 的数据帧中包
长度应小于 1480 字节）
const int SEND_WIND_SIZE = 10;//发送窗口大小为 10，SR 中应满足 W + 1 <= N
（W 为发送窗口大小，N 为序列号个数）
                              //本例取序列号 0...19 共 20 个
                              //如果将窗口大小设为 1，则为停-等协议

const int SEQ_SIZE = 20; //序列号的个数，从 0~19 共计 20 个
                         //由于发送数据第一个字节如果值为 0，则数据会发送失
败
                         //因此接收端序列号为 1~20，与发送端一一对应

int ack[SEQ_SIZE];//收到 ack 情况，对应 0~19 的 ack
int curSeq;//当前数据包的 seq
int curAck;//当前等待确认的 ack
```

```
int totalSeq;//收到的包的总数
int totalPacket;//需要发送的包总数
//*************************************
// Method: lossInLossRatio
// FullName: lossInLossRatio
// Access: public
// Returns: BOOL
// Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回 TRUE，否则返回 FALSE
// Parameter: float lossRatio [0,1]
//*************************************
BOOL lossInLossRatio(float lossRatio) {
    int lossBound = (int)(lossRatio * 100);
    int r = rand() % 101;
    if (r <= lossBound) {
        return TRUE;
    }
    return FALSE;
}
//*************************************
// Method: getCurTime
// FullName: getCurTime
// Access: public
// Returns: void
// Qualifier: 获取当前系统时间，结果存入 ptime 中
// Parameter: char * ptime
//*************************************
void getCurTime(char *ptime) {
    char buffer[128];
    memset(buffer, 0, sizeof(buffer));
    time_t c_time;
    struct tm *p;
    time(&c_time);
    p = localtime(&c_time);
    sprintf_s(buffer, "%d/%d/%d %d:%d:%d",
        p->tm_year + 1900,
        p->tm_mon,
        p->tm_mday,
        p->tm_hour,
        p->tm_min,
        p->tm_sec);
    strcpy_s(ptime, sizeof(buffer), buffer);
}
```

```
//************************************
// Method: seqIsAvailable
// FullName: seqIsAvailable
// Access: public
// Returns: bool
// Qualifier: 当前序列号 curSeq 是否可用
//************************************
bool seqIsAvailable() {
    int step;

    step = curSeq - curAck;
    step = step >= 0 ? step : step + SEQ_SIZE;
    //序列号是否在当前发送窗口之内
    if (step >= SEND_WIND_SIZE) {
        return false;
    }
    if (ack[curSeq] == 1 || ack[curSeq] == 2) {
        return true;
    }
    return false;
}

//************************************
// Method: timeoutHandler
// FullName: timeoutHandler
// Access: public
// Returns: void
// Qualifier: 超时重传处理函数，只重传丢失的数据帧
//************************************
void timeoutHandler() {
    printf("Timer out error.\n");
    int index, number = 0;
    for (int i = 0; i< SEND_WIND_SIZE; ++i) {
        index = (i + curAck) % SEQ_SIZE;
        if (ack[index] == 0)
        {
            ack[index] = 2;
            number++;
        }
    }
    totalSeq = totalSeq - number;
    curSeq = curAck;
}
```

```
//**********************************
// Method: ackHandler
// FullName: ackHandler
// Access: public
// Returns: void
// Qualifier: 收到 ack，取数据帧的第一个字节
//由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一了，此处需要减一还原
// Parameter: char c
//**********************************
void ackHandler(char c) {
    unsigned char index = (unsigned char)c - 1; //序列号减一
    printf("Recv a ack of %d\n", index+1);
    if (index == 0) {
        printf("sadas");
    }
    if (curAck <= index && (curAck + SEND_WIND_SIZE >= SEQ_SIZE ? true : index<curAck + SEND_WIND_SIZE)) {

        ack[index] = 3;
        while (ack[curAck] == 3) {
            ack[curAck] = 1;
            curAck = (curAck + 1) % SEQ_SIZE;

        }
    }
}

//主函数
int main(int argc, char* argv[])
{
    //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 sockServer = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    //设置套接字为非阻塞模式
    int iMode = 1; //1: 非阻塞，0: 阻塞
    ioctlsocket(sockServer, FIONBIO, (u_long FAR*) &iMode);//非阻塞设置
    SOCKADDR_IN addrServer; //服务器地址
    addrServer.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
    //addrServer.sin_addr.S_un.S_addr = htonl(INADDR_ANY);//两者均可
    addrServer.sin_family = AF_INET;
    addrServer.sin_port = htons(SERVER_PORT);
    err = bind(sockServer, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
    if (err) {
        err = GetLastError();
        printf("Could not bind the port %d for socket.Error code is %d\n",
SERVER_PORT, err);
        WSACleanup();
        return -1;
    }

    SOCKADDR_IN addrClient; //客户端地址
    int length = sizeof(SOCKADDR);
    char buffer[BUFFER_LENGTH]; //数据发送接收缓冲区
    ZeroMemory(buffer, sizeof(buffer));
    //将测试数据读入内存
    std::ifstream icin;
    icin.open("../test.txt");

    char data[1024 * 113];
    ZeroMemory(data, sizeof(data));
    icin.read(data, 1024 * 113);
    icin.close();


    int handle;
    handle = _open("../test.txt", 0x0100); //open file for read
    long length_lvxiya = _filelength(handle); //get length of file
```

```
    totalPacket = length_lvxiya / 1024 + 1;
    int recvSize;
    for (int i = 0; i < SEQ_SIZE; ++i) {
        ack[i] = 1;
    }

    int ret;
    int interval = 1;
    float packetLossRatio = 0.2; //默认包丢失率 0.2
    float ackLossRatio = 0.2; //默认 ACK 丢失率 0.2
                              //用时间作为随机种子，放在循环的最外面
    srand((unsigned)time(NULL));

    while (true) {
        //非阻塞接收，若没有收到数据，返回值为-1
        recvSize =
            recvfrom(sockServer, buffer, BUFFER_LENGTH, 0,
((SOCKADDR*)&addrClient), &length);
        if (recvSize < 0) {
            Sleep(200);
            continue;
        }
        printf("recv from client: %s\n", buffer);
        if (strcmp(buffer, "-time") == 0) {
            getCurTime(buffer);
        }
        else if (strcmp(buffer, "-quit") == 0) {
            strcpy_s(buffer, strlen("Good bye!") + 1, "Good bye!");
        }
        else if (strcmp(buffer, "-testSR") == 0) {
            //进入 SR 测试阶段
            //首先 server（server 处于 0 状态）向 client 发送 205 状态码
（server 进入 1 状态）
            //server 等待 client 回复 200 状态码，如果收到（server 进入 2
状态），     则开始传输文件，否则延时等待直至超时\

      //在文件传输阶段，server 发送窗口大小设为
            ZeroMemory(buffer, sizeof(buffer));
            int recvSize;
            //加入了一个握手阶段
            //首先服务器向客户端发送一个 205 大小的状态码（我自己定义的）表
示服务器准备好了，可以发送数据
            //客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准备好
```

```
了，可以接收数据了
            //服务器收到 200 状态码之后，就开始使用 SR 发送数据了
            printf("Shake hands stage\n");
            int stage = 0;
            bool runFlag = true;
            printf("%s\n", "Begin to test SR protocol, please don't abort
the process");
            printf("The loss ratio of packet is %.2f,the loss ratio of ack
is %.2f\n", packetLossRatio, ackLossRatio);
            //-----------------------------------------------------
            int waitCount = 0;//---------------
            BOOL b;
            unsigned short seq = 0;//包的序列号
            unsigned short recvSeq = 0;//已确认的最大序列号
            unsigned short waitSeq = 1;//等待的序列号 ， 窗口大小为 10，这
个为最小的值
            char buffer_1[SEND_WIND_SIZE][BUFFER_LENGTH];//接收到的缓冲
区数据----------------add bylvxiya
            int i_state = 0;
            for (i_state = 0; i_state < SEND_WIND_SIZE; i_state++)
            {
                ZeroMemory(buffer_1[i_state],
sizeof(buffer_1[i_state]));
            }

            BOOL ack_send[SEND_WIND_SIZE];//ack 发送情况的记录，对应 1-20
的 ack,刚开始全为 false
            int success_number = 0;// 窗口内成功接收的个数
            for (i_state = 0; i_state < SEND_WIND_SIZE; i_state++) {//
记录哪一个成功接收了
                ack_send[i_state] = false;
            }
            std::ofstream out_result;
            out_result.open("result.txt", std::ios::out |
std::ios::trunc);
            if (!out_result.is_open()) {
                printf("文件打开失败！！！\n");
                continue;
            }

            //-----------------------------------------------------------
----------------
            while (runFlag) {
                int recv_lvxy = 0;
```

```
                switch (stage) {
                case 0://发送 205 阶段
                    buffer[0] = 205;
                    sendto(sockServer, buffer, strlen(buffer) + 1, 0,
                        (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
                    Sleep(100);
                    stage = 1;
                    break;
                case 1://等待接收 200 阶段，没有收到则计数器+1，超时则放弃此次“连接”，等待从第一步开始

                    recv_lvxy =
                        recvfrom(sockServer, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrClient), &length);
                    if (recv_lvxy < 0) {
                        ++waitCount;
                        if (waitCount > 20) {
                            runFlag = false;
                            printf("Timeout error\n");
                            break;
                        }
                        Sleep(500);
                        continue;
                    }
                    else {
                        if ((unsigned char)buffer[0] == 200) {
                            printf("Begin a file transfer\n");
                            printf("File size is %dB, each packet is 1024B and packet total num is %d\n", sizeof(data), totalPacket);
                            curSeq = 0;
                            curAck = 0;
                            totalSeq = 0;
                            waitCount = 0;
                            stage = 2;

                        }
                    }
                    break;
                case 2://数据传输阶段
                    if (seqIsAvailable() && totalSeq - SEND_WIND_SIZE <= totalPacket) {

                        b = lossInLossRatio(ackLossRatio);
                        //发送给客户端的序列号从 1 开始
```

```
                        buffer[0] = curSeq + 1;
                        if (b) {
                            printf("The packet from client of %d loss\n",
(unsigned char)buffer[0]);
                            break;
                        }

                        buffer[1] = seq;//-++++++++++++++++++++++待定，是
不是 0

                        ack[curSeq] = 0;

                        //数据发送的过程中应该判断是否传输完成
                        //为简化过程此处并未实现
                        memcpy(&buffer[2], data + 1024 * (curSeq +
(totalSeq / SEND_WIND_SIZE)*SEND_WIND_SIZE), 1024);
                        printf("SERVER: send a packet with a seq of %d and
a ack of %d\n", curSeq + 1, seq);
                        sendto(sockServer, buffer, BUFFER_LENGTH, 0,
                            (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
                        ++curSeq;
                        curSeq %= SEQ_SIZE;
                        ++totalSeq;
                        //Sleep(500);

                        recv_lvxy =
                            recvfrom(sockServer, buffer, BUFFER_LENGTH,
0, ((SOCKADDR*)&addrClient), &length);
                        //recv_lvxy =
                        //recvfrom(sockServer, buffer, BUFFER_LENGTH, 0,
((SOCKADDR*)&addrClient), &length);
                        if (recv_lvxy < 0) {
                            waitCount++;
                            //20 次等待 ack 则超时重传
                            if (waitCount > 20)
                            {
                                timeoutHandler();
                                waitCount = 0;
                            }
                        }
                        else {
                            b = lossInLossRatio(packetLossRatio);
                            if (b) {
                                printf("The packet from Client with a seq
```

```
of %d and ACK for SERVER of %d loss\n", buffer[0], buffer[1]);
                                break;
                            }
                            //注意这时 ack--(unsigned short)buffer[1];的
话要看看是不是 0，如果是 0 的话表示对方没有收过数据，是不能 ackHandel 的；
                            if ((unsigned short)buffer[1] != 0) {
                                ackHandler(buffer[1]);
                                waitCount = 0;
                            }
                            //--------------------------------————————
————————

                            seq = (unsigned short)buffer[0];
                            printf("client packet: %d, ACK for
SERVER: %d\n", seq, buffer[1]);
                            if (seq >= waitSeq && (waitSeq +
SEND_WIND_SIZE > SEQ_SIZE ? true : seq < (waitSeq + SEND_WIND_SIZE))) {//
在接收窗口范围内
                                memcpy(buffer_1[seq - waitSeq],
&buffer[2], sizeof(buffer));
                                ack_send[seq - waitSeq] = true;
                                int ack_s = 0;
                                while (ack_send[ack_s] && ack_s <
SEND_WIND_SIZE) {
                                    //向上层传输数
据
                                    out_result << buffer_1[ack_s];
                                    //printf("%s",buffer_1[ack_s - 1]);
                                    ZeroMemory(buffer_1[ack_s],
sizeof(buffer_1[ack_s]));
                                    waitSeq++;
                                    if (waitSeq == 21) {
                                        waitSeq = 1;
                                    }
                                    ack_s = ack_s + 1;
                                }
                                if (ack_s > 0) {
                                    for (int i = 0; i < SEND_WIND_SIZE;
i++) {
                                        if (ack_s + i < SEND_WIND_SIZE)
                                        {
                                            ack_send[i] = ack_send[i +
ack_s];
                                            memcpy(buffer_1[i],
buffer_1[i + ack_s], sizeof(buffer_1[i + ack_s]));
```

```
                                        ZeroMemory(buffer_1[i +
ack_s], sizeof(buffer_1[i + ack_s]));
                                    }
                                    else
                                    {
                                        ack_send[i] = false;
                                        ZeroMemory(buffer_1[i],
sizeof(buffer_1[i]));
                                    }

                                }
                            }
                            recvSeq = seq;
                        }
                        buffer[1] = seq;
                    }

                }
                else if (curSeq>0 && curSeq - curAck >= 0 ? curSeq -
curAck <= SEND_WIND_SIZE : curSeq - curAck + SEQ_SIZE <= SEND_WIND_SIZE
&& totalSeq - SEND_WIND_SIZE <= totalPacket) {

                    curSeq++;
                    curSeq %= SEQ_SIZE;
                }
                else {
                    waitCount++;

                    if (waitCount > 20)
                    {
                        timeoutHandler();
                        waitCount = 0;
                        //recvSeq = 0;
                    }

                }
                /*
                else if (totalSeq - SEND_WIND_SIZE > totalPacket){
                memcpy(buffer, "good bye\0", 9);
                runFlag = false;
                break;
                }*/
                Sleep(500);
                break;
```

```
                }
            }
        success:out_result.close();
        }
        sendto(sockServer, buffer, strlen(buffer) + 1, 0,
(SOCKADDR*)&addrClient,
            sizeof(SOCKADDR));
        Sleep(500);

    }
    //关闭套接字，卸载库
    closesocket(sockServer);
    WSACleanup();
    return 0;
}
```

双向 SR -- 客户端:

```
// SR_client.cpp: 定义控制台应用程序的入口点。
//

#include "stdafx.h"


#define _WINSOCK_DEPRECATED_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <WinSock2.h>
#include <time.h>
#include <fstream>
#include<io.h> //C 语言头文件
#include<iostream> //for system();
using namespace std;
#pragma comment(lib,"ws2_32.lib")

#define SERVER_PORT 12340 //接收数据的端口号
#define SERVER_PORT_recv 10240
#define SERVER_IP "127.0.0.1" // 服务器的 IP 地址

const int BUFFER_LENGTH = 1026;
const int SEQ_SIZE = 20;//接收端序列号个数，为 1~20
const int RECV_WIND_SIZE = 10;//接收窗口的大小，它要小于等于序号大小的一半
                              //由于发送数据第一个字节如果值为 0，则数据会发
送失败
                              //因此接收端序列号为 1~20，与发送端一一对应
```

```
int ack[SEQ_SIZE];//收到 ack 情况，对应 0~19 的 ack
int curSeq;//当前数据包的 seq
int curAck;//当前等待确认的 ack
int totalSeq;//收到的包的总数
int totalPacket;//需要发送的包总数

/****************************************************************/
/*
-time 从服务器端获取当前时间
-quit 退出客户端
-testSR [X] 测试 SR 协议实现可靠数据传输
[X] [0,1] 模拟数据包丢失的概率
[Y] [0,1] 模拟 ACK 丢失的概率
*/
/****************************************************************/
void printTips() {
    printf("*************************************\n");
    printf("| -time to get current time |\n");
    printf("| -quit to exit client |\n");
    printf("| -testSR [X] [Y] to test the SR |\n");
    printf("*************************************\n");
}
//***********************************
// Method: seqIsAvailable
// FullName: seqIsAvailable
// Access: public
// Returns: bool
// Qualifier: 当前序列号 curSeq 是否可用
//***********************************
bool seqIsAvailable() {
    int step;

    step = curSeq - curAck;
    step = step >= 0 ? step : step + SEQ_SIZE;
    //序列号是否在当前发送窗口之内
    if (step >= RECV_WIND_SIZE) {
        return false;
    }
    if (ack[curSeq] == 1 || ack[curSeq] == 2) {
        return true;
    }
    return false;
}
```

```
//************************************
// Method: timeoutHandler
// FullName: timeoutHandler
// Access: public
// Returns: void
// Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传
//************************************
void timeoutHandler() {
    printf("Timer out error.\n");
    int index, number = 0;
    for (int i = 0; i< RECV_WIND_SIZE; ++i) {
        index = (i + curAck) % SEQ_SIZE;
        if (ack[index] == 0)
        {
            ack[index] = 2;
            number++;
        }
    }
    totalSeq = totalSeq - number;
    curSeq = curAck;
}

//************************************
// Method: ackHandler
// FullName: ackHandler
// Access: public
// Returns: void
// Qualifier: 收到 ack，累积确认，取数据帧的第一个字节
//由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一了，此处需要减一还原
// Parameter: char c
//************************************
void ackHandler(char c) {
    unsigned char index = (unsigned char)c - 1; //序列号减一
    //printf("Recv a ack of %d\n", index+1);
    if (curAck <= index && (curAck + RECV_WIND_SIZE >= SEQ_SIZE ? true : index<curAck + RECV_WIND_SIZE)) {

        ack[index] = 3;
        while (ack[curAck] == 3) {
            ack[curAck] = 1;
            curAck = (curAck + 1) % SEQ_SIZE;

        }
```

```
    }
}
//************************************
// Method: lossInLossRatio
// FullName: lossInLossRatio
// Access: public
// Returns: BOOL
// Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回 TRUE，否则返回 FALSE
// Parameter: float lossRatio [0,1]
//************************************
BOOL lossInLossRatio(float lossRatio) {
    int lossBound = (int)(lossRatio * 100);
    int r = rand() % 101;
    if (r <= lossBound) {
        return TRUE;
    }
    return FALSE;
}

int main(int argc, char* argv[])
{
    //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);
    //设置套接字为非阻塞模式
    int iMode = 1; //1：非阻塞，0：阻塞
    ioctlsocket(socketClient, FIONBIO, (u_long FAR*) &iMode);//非阻塞设置



    SOCKADDR_IN addrServer;
    addrServer.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);
    addrServer.sin_family = AF_INET;
    addrServer.sin_port = htons(SERVER_PORT);


    //接收缓冲区
    char buffer[BUFFER_LENGTH];//接收数据的缓冲
                               //char buffer_send1[BUFFER_LENGTH];//发送数据的缓冲
    ZeroMemory(buffer, sizeof(buffer));
    int len = sizeof(SOCKADDR);
    //将测试数据读入内存
    std::ifstream icin;
    icin.open("../test.txt");

    char data[1024 * 113];
    ZeroMemory(data, sizeof(data));
    icin.read(data, 1024 * 113);
    icin.close();
    int handle;
    handle = _open("../test.txt", 0x0100); //open file for read
    long length_lvxiya = _filelength(handle); //get length of file
    for (int i = 0; i < SEQ_SIZE; ++i) {
        ack[i] = 1;
    }
    totalPacket = length_lvxiya / 1024 + 1;
    //为了测试与服务器的连接，可以使用 -time 命令从服务器端获得当前时间
    //使用 -testSR [X] [Y] 测试 SR 其中[X]表示数据包丢失概率
    // [Y]表示 ACK 丢包概率
    printTips();

    int ret;
    int recvSize = 0;
    int interval = 1;//收到数据包之后返回 ack 的间隔，默认为 1 表示每个都返回 ack，0 或者负数均表示所有的都不返回 ack
    char cmd[128];
```

```
    float packetLossRatio = 0.2; //默认包丢失率 0.2
    float ackLossRatio = 0.2; //默认 ACK 丢失率 0.2
                              //用时间作为随机种子，放在循环的最外面
    srand((unsigned)time(NULL));
    while (true) {
        gets_s(buffer);
        ret = sscanf(buffer, "%s%f%f", &cmd, &packetLossRatio,
&ackLossRatio);
        //开始 SR 测试，使用 SR 协议实现 UDP 可靠文件传输
        if (!strcmp(cmd, "-testSR")) {
            printf("%s\n", "Begin to test SR protocol, please don't abort
the process");
            printf("The loss ratio of packet is %.2f,the loss ratio of ack
is %.2f\n", packetLossRatio, ackLossRatio);
            int waitCount = 0;
            int stage = 0;
            BOOL b;
            unsigned char u_code;//状态码
            unsigned short seq;//包的序列号
            unsigned short recvSeq;//已确认的最大序列号
            unsigned short waitSeq;//等待的序列号 ，窗口大小为 10，这个为最
小的值

            char buffer_1[RECV_WIND_SIZE][BUFFER_LENGTH];//接收到的缓冲
区数据----------------add
            int i_state = 0;
            for (i_state = 0; i_state < RECV_WIND_SIZE; i_state++) {
                ZeroMemory(buffer_1[i_state],
sizeof(buffer_1[i_state]));
            }

            BOOL ack_send[RECV_WIND_SIZE];//ack 发送情况的记录，对应 1-20
的 ack,刚开始全为 false
            int success_number = 0;// 窗口内成功接收的个数
            for (i_state = 0; i_state < RECV_WIND_SIZE; i_state++) {//
记录哪一个成功接收了
                ack_send[i_state] = false;
            }
            std::ofstream out_result;
            out_result.open("result.txt", std::ios::out |
std::ios::trunc);
            if (!out_result.is_open()) {
                printf("文件打开失败！！！\n");
                continue;
```

```
            }
            //---------------------------------
            sendto(socketClient, "-testSR", strlen("-testSR") + 1, 0,
                (SOCKADDR*)&addrServer, sizeof(SOCKADDR));

            while (true)
            {
                //recvSize = recvfrom(socketClient, buffer,
BUFFER_LENGTH, 0, (SOCKADDR*)&addrServer, &len);
                switch (stage)
                {
                case 0://等待握手阶段
                    do
                    {
                        recvSize = recvfrom(socketClient, buffer,
BUFFER_LENGTH, 0, (SOCKADDR*)&addrServer, &len);
                    } while (recvSize < 0);
                    u_code = (unsigned char)buffer[0];
                    if ((unsigned char)buffer[0] == 205)
                    {
                        printf("Ready for file transmission\n");
                        buffer[0] = 200;
                        buffer[1] = '\0';
                        sendto(socketClient, buffer, 2, 0,
                            (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
                        stage = 1;
                        recvSeq = 0;
                        waitSeq = 1;
                        curAck = 0;
                        totalSeq = 0;
                        waitCount = 0;
                        curSeq = 0;
                    }
                    break;
                case 1://等待接收数据阶段
                    if (seqIsAvailable() && totalSeq - RECV_WIND_SIZE <=
totalPacket)
                    {
                        recvSize = recvfrom(socketClient, buffer,
BUFFER_LENGTH, 0, (SOCKADDR*)&addrServer, &len);
                        if (recvSize < 0) {
                            waitCount++;
                            //20 次等待 ack 则超时重传
                            if (waitCount > 20)
```

```
                            {
                                timeoutHandler();
                                waitCount = 0;

                            }
                        }
                        else {
                            b = lossInLossRatio(packetLossRatio);
                            if (b) {
                                printf("The packet from server of %d
loss\n", buffer[0]);
                                break;
                            }
                            seq = (unsigned short)buffer[0];

                            printf("recv packet: %d, ACK from
SERVER:%d\n", seq, (unsigned short)buffer[1]);
                            if (seq >= waitSeq && (waitSeq +
RECV_WIND_SIZE > SEQ_SIZE ? true : seq < (waitSeq + RECV_WIND_SIZE)))
                            {
                                memcpy(buffer_1[seq - waitSeq],
&buffer[2], sizeof(buffer));
                                ack_send[seq - waitSeq] = true;
                                int ack_s = 0;
                                while (ack_send[ack_s] && ack_s <
RECV_WIND_SIZE) {
                                    //向上层传输数
据
                                    out_result << buffer_1[ack_s];
                                    //printf("%s",buffer_1[ack_s - 1]);
                                    ZeroMemory(buffer_1[ack_s],
sizeof(buffer_1[ack_s]));
                                    waitSeq++;
                                    if (waitSeq == 21) {
                                        waitSeq = 1;
                                    }
                                    ack_s = ack_s + 1;
                                }
                                if (ack_s > 0) {
                                    for (int i = 0; i < RECV_WIND_SIZE;
i++) {
                                        if (ack_s + i < RECV_WIND_SIZE)
                                        {
                                            ack_send[i] = ack_send[i +
```

```
ack_s];
                                        memcpy(buffer_1[i],
buffer_1[i + ack_s], sizeof(buffer_1[i + ack_s]));
                                        ZeroMemory(buffer_1[i +
ack_s], sizeof(buffer_1[i + ack_s]));
                                    }
                                    else
                                    {
                                        ack_send[i] = false;
                                        ZeroMemory(buffer_1[i],
sizeof(buffer_1[i]));
                                    }

                                }
                            }
                            recvSeq = seq;
                        }
                        //注意这时 ack--(unsigned short)buffer[1];的
话要看看是不是 0，如果是 0 的话表示对方没有收过数据，是不能 ackHandel 的；
                        if ((unsigned short)buffer[1]) {
                            ackHandler(buffer[1]);
                            waitCount = 0;
                        }
                        buffer[1] = seq;

                        buffer[0] = curSeq + 1;
                        b = lossInLossRatio(ackLossRatio);
                        if (b) {
                            printf("The packet from server of %d and
ACK for server of %d loss\n", (unsigned char)buffer[0], seq);
                            break;
                        }
                        ack[curSeq] = 0;
                        //数据发送的过程中应该判断是否传输完成
                        //为简化过程此处并未实现
                        memcpy(&buffer[2], data + 1024 * (curSeq +
(totalSeq / RECV_WIND_SIZE)*RECV_WIND_SIZE), 1024);
                        printf("send packet: %d and ACK for server
of %d \n", curSeq + 1, seq);
                        sendto(socketClient, buffer,
sizeof(buffer), 0,
                            (SOCKADDR*)&addrServer,
sizeof(SOCKADDR));
                        if (buffer[1] == 1) {
```

```
                                printf("dsa");
                            }
                            ++curSeq;
                            curSeq %= SEQ_SIZE;
                            ++totalSeq;
                            //Sleep(500);
                            //break;
                        }
                    }
                    //memcpy(buffer_send1,buffer,sizeof(buffer));

                    else if (curSeq - curAck >= 0 ? curSeq - curAck <=
RECV_WIND_SIZE : curSeq - curAck + SEQ_SIZE <= RECV_WIND_SIZE && totalSeq
- RECV_WIND_SIZE <= totalPacket) {
                        curSeq++;
                        curSeq %= SEQ_SIZE;
                    }
                    else {
                        waitCount++;
                        if (waitCount > 18)
                        {
                            timeoutHandler();
                            waitCount = 0;
                        }
                    }
                    Sleep(500);
                    break;
                }
                //Sleep(500);
            }
            out_result.close();
        }
        sendto(socketClient, buffer, sizeof(buffer), 0,
            (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
        do
        {
            ret =
                recvfrom(socketClient, buffer, BUFFER_LENGTH, 0,
(SOCKADDR*)&addrServer, &len);
        } while (ret < 0);



        printf("%s\n", buffer);
```

```
        if (!strcmp(buffer, "Good bye!")) {
            break;
        }
        printTips();
    }
    //关闭套接字
    closesocket(socketClient);
    WSACleanup();
    return 0;
}
```