

**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 | 刘俊熙 | | 院系 | 信息安全 | | |
| 班级 | 2103201 | | 学号 | 2021113065 | | |
| 任课教师 | 詹东阳 | | 指导教师 | 詹东阳 | | |
| 实验地点 | 格物207 | | 实验时间 | 2023 | | |
| 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 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 实验目的： |
| （注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）  本次实验的主要目的。  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；  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： |
| 概述本次实验的主要内容，包含的实验项等。  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；  4）将所设计的 GBN 协议改进为 SR 协议。 |
| 实验过程： |
| 以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。  实验过程：  GBN 协议：  首先，实现 GBN 的单向传输，创建一个套接字，并绑定在指定的端口上。客户端（接收端）请求数据，读取控制台的请求信息，并解析该命令。根据不同的命令，请求不同的数据。  当执行单向传输的命令时，客户端首先发送请求信息，然后服务器端解析请求，进行一个握手阶段，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，服务器端读取本地文件，放到缓存中，发送给客户端。  在发送端设置分组丢失率和ACK丢失率（默认二者都为0.2），ACK采用累积确认（取数据帧的第一个字节，为 ACK 对应分组的序列号），当收到一个字段的序列号时，在其之前的所有分组全都确认被收到。当发生超时情况时，发送端重新发送整个窗户中的所有数据分组。  GBN 双向传输也是相同的原理。只不过发送端变成了客户端。  SR协议：  连接过程（握手阶段）与 GBN 相同，都是根据 205 状态码和 200 状态码来判断是否能够正常建立连接。不同的是，SR 协议，接收端也有一个窗口，当发送的数据在窗口中是，即使前面有还没有收到的分组，也会将该组缓存下来，并发送该分组的 ACK，采用超时重传技术，即哪个分组的 ack 超时了，就发送哪个分组；当收到在窗口外的分组时，则丢掉。 |
| 实验结果： |
| 采用演示截图、文字说明等方式，给出本次实验的实验结果。  实验结果见下面 |
| 问题讨论： |
| 对实验过程中的思考问题进行讨论或回答。  问题：  实验中，当发送数据到达窗口边界时，有的时候，会出现一些问题。  解决方案：  增加一些判断数据传输是否完成添加或修改的语句即可解决问题。 |
| 心得体会： |
| 结合实验过程和结果给出实验的体会和收获。  本次实验虽然做的很仓促，但是确实实现很多功能。并让我对socket编程有了进一步的了解，并且，对GBN和SR协议有了更深的认识。 |

所设计 GBN 协议数据分组格式、确认分组格式、各个域作用：

GBN 协议数据分组格式：

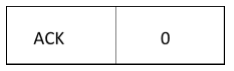


Seq ： 为 1 个字节，取值为 0~255，（故序列号最多为 256 个）；

Data ： 小于等于 1024 个字节，为传输的数据；

0 ： 最后一个字节放入 EOF0，表示结尾。

确认分组格式：

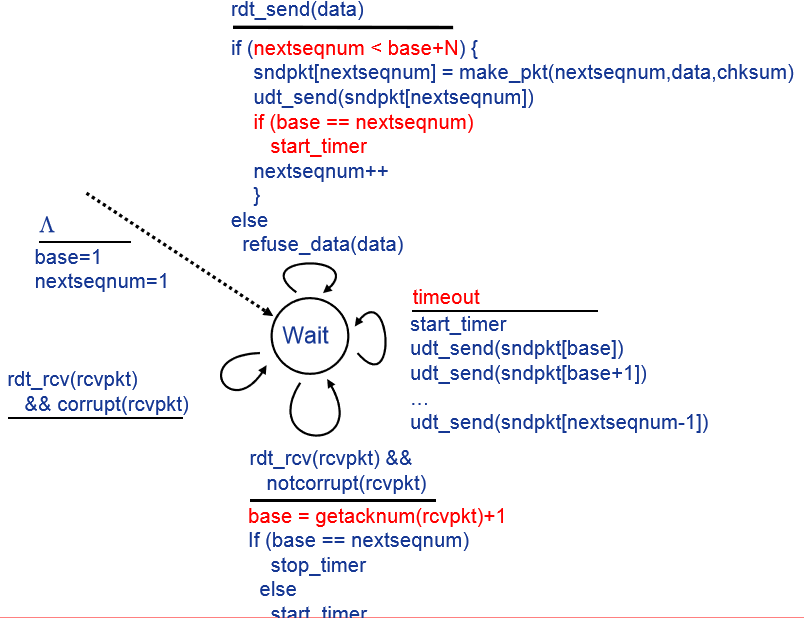


ACK 字段为一个字节，表示序列号数值；

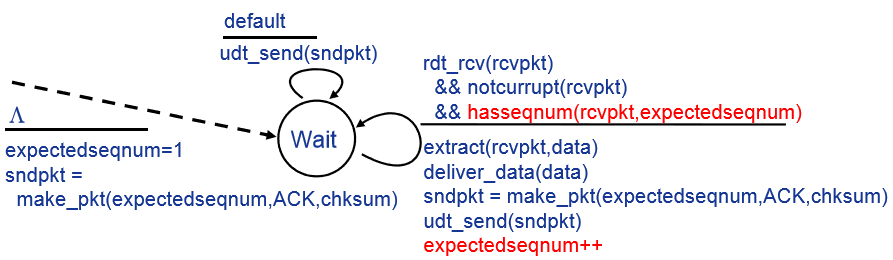
末尾放入 0，表示数据结束。

GBN 协议两端程序流程图：

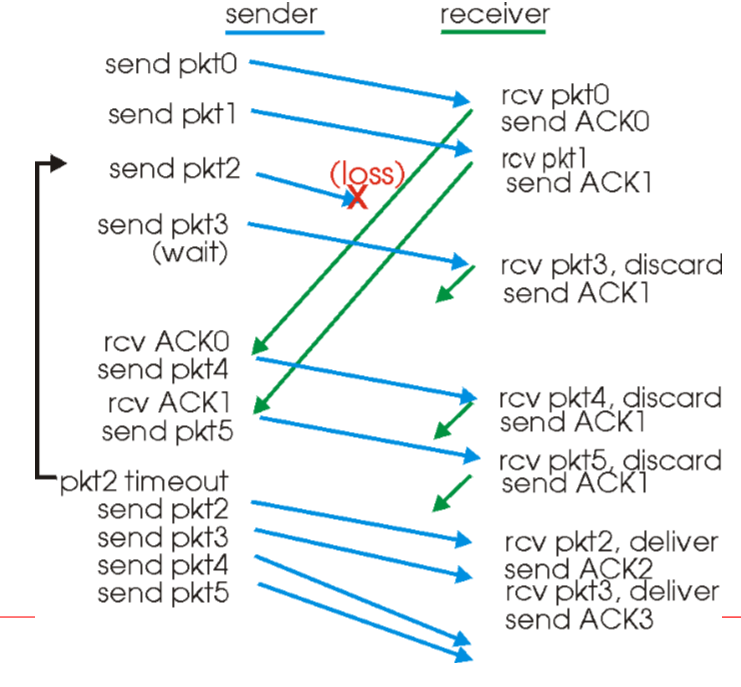
服务器端（发送端）：



客户端（接收端）：



GBN 协议典型交互过程：



所设计 SR 协议数据分组格式、确认分组格式、各个域作用：

SR 协议数据分组格式：

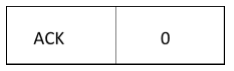


Seq ： 为 1 个字节，取值为 0~255，（故序列号最多为 256 个）；

Data ： 小于等于 1024 个字节，为传输的数据；

0 ： 最后一个字节放入 EOF0，表示结尾。

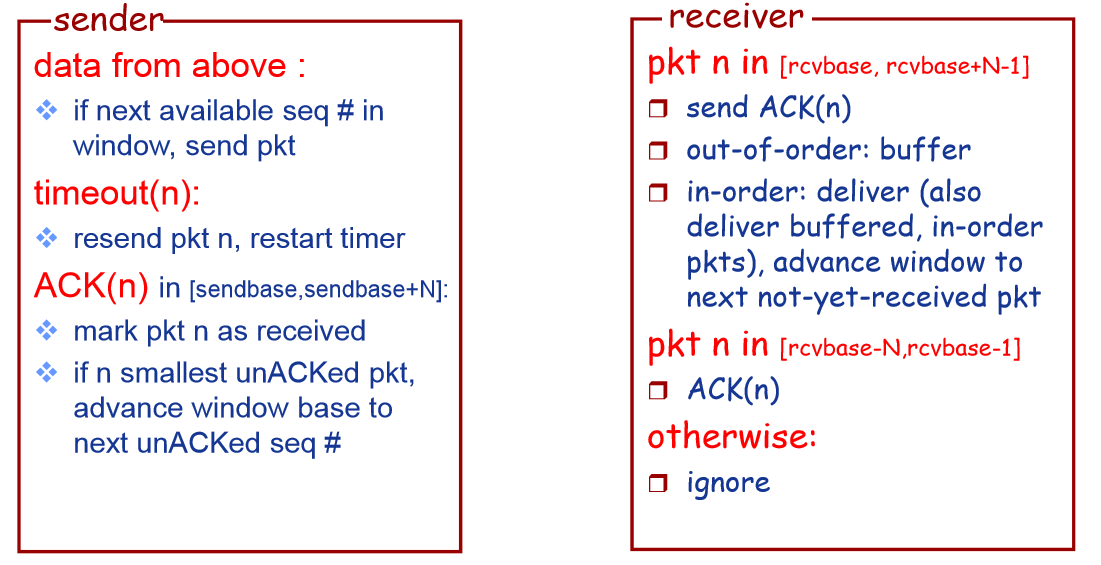
确认分组格式：



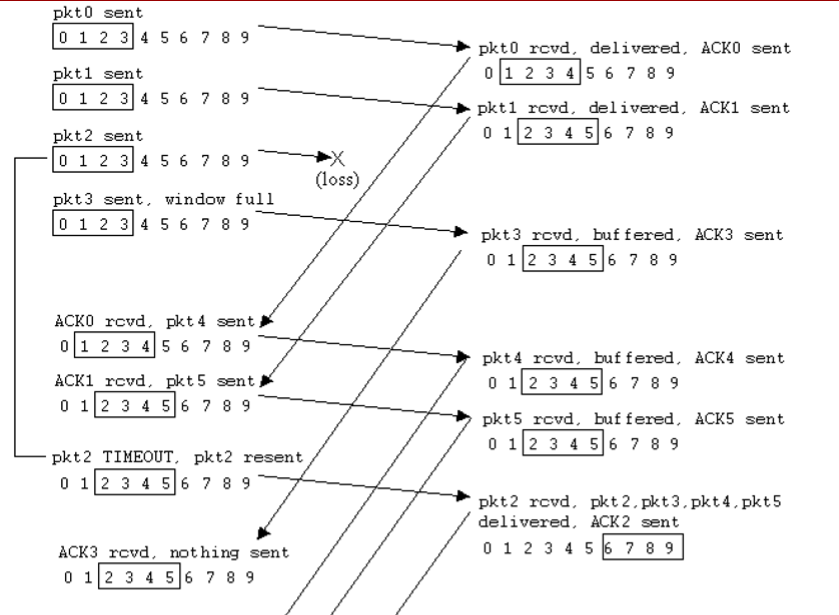
ACK 字段为一个字节，表示序列号数值；

末尾放入 0，表示数据结束。

SR 协议两端程序流程图：



SR 协议典型交互过程：



数据分组丢失验证模拟方法：

设置一个分组丢失率，将丢失率乘上100，随机生成一个0到100的数，检查该数是否在0到丢失率的范围内。如果在，则分组丢失；否则，没有丢失。

ACK 的丢失模拟也是同样的方法，二者调用相同的模拟丢失函数。

程序实现的主要类（或函数）及其主要作用：

主要有4个项目：

GBN：GBN\_server.cpp，GBN\_client.cpp

SR：SR\_client.cpp，SR\_server.cpp

主要函数及其作用：

void getCurTime(char \*ptime)：获取当前系统时间，结果存入 ptime 中

BOOL lossInLossRatio(float lossRatio)：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

void printTips()：打印提示信息：

-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

bool seqIsAvailable()：当前序列号 curSeq 是否可用

void timeoutHandler()：超时重传处理函数，滑动窗口内的数据帧都要重传

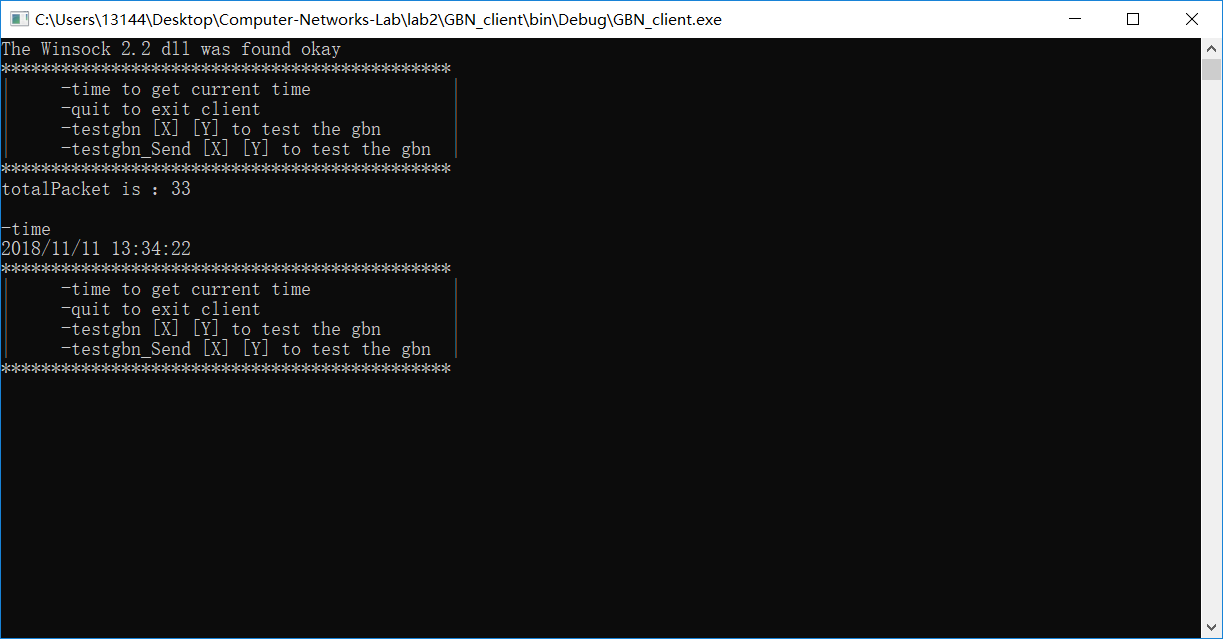
void ackHandler(char c)：收到 ack，累积确认，取数据帧的第一个字节

实验验证结果

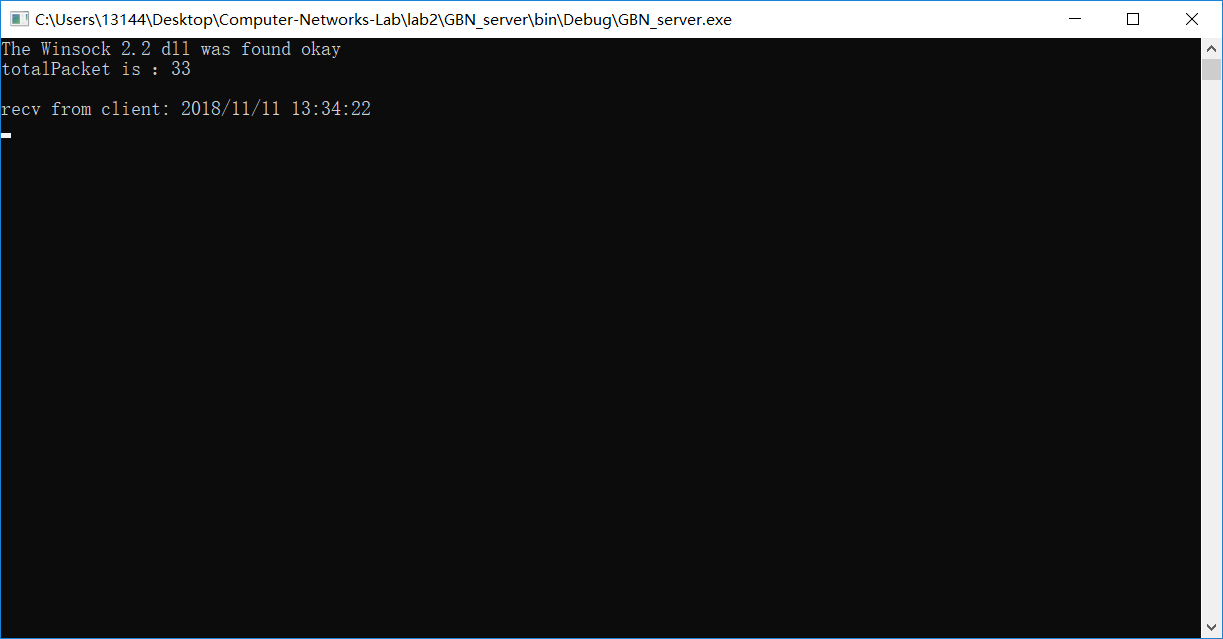
GBN 实验运行结果

-time获取时间测试：

客户端：

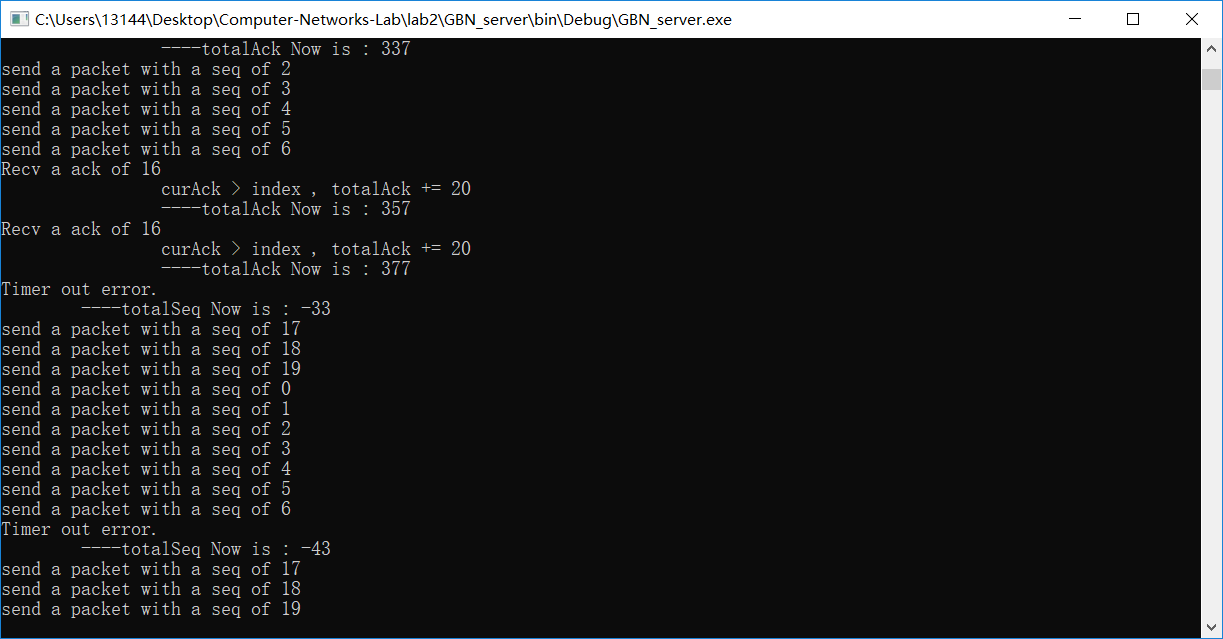


服务器端：

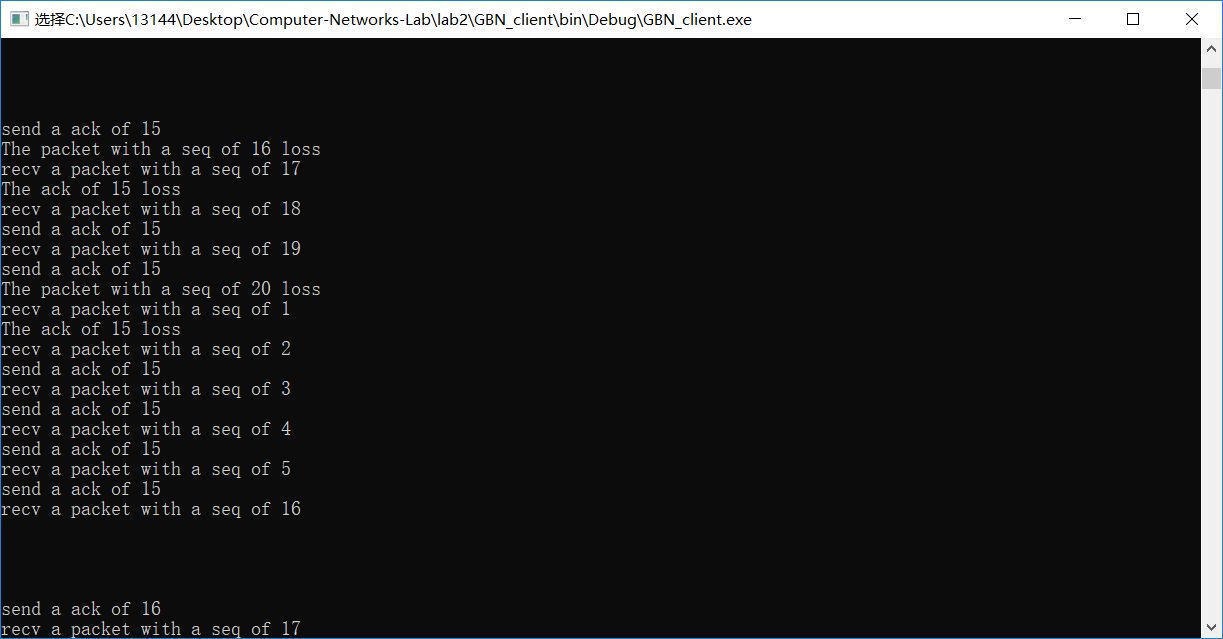


-testgbn 单向传输：

服务器端（发送端）：



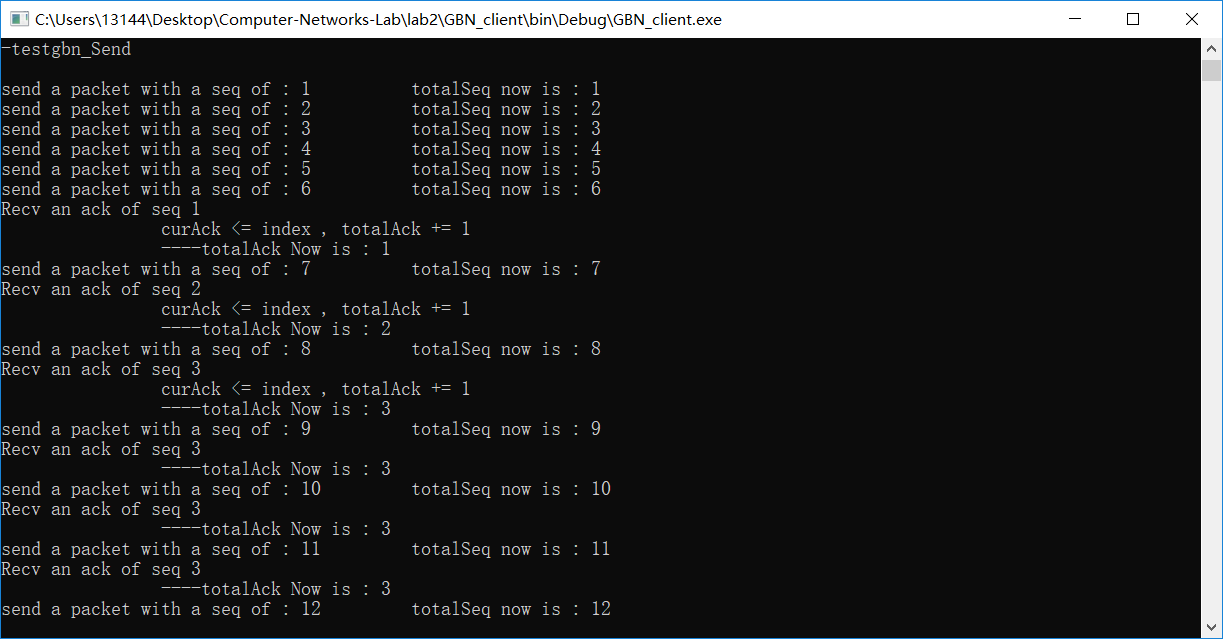
客户端（接收端）：



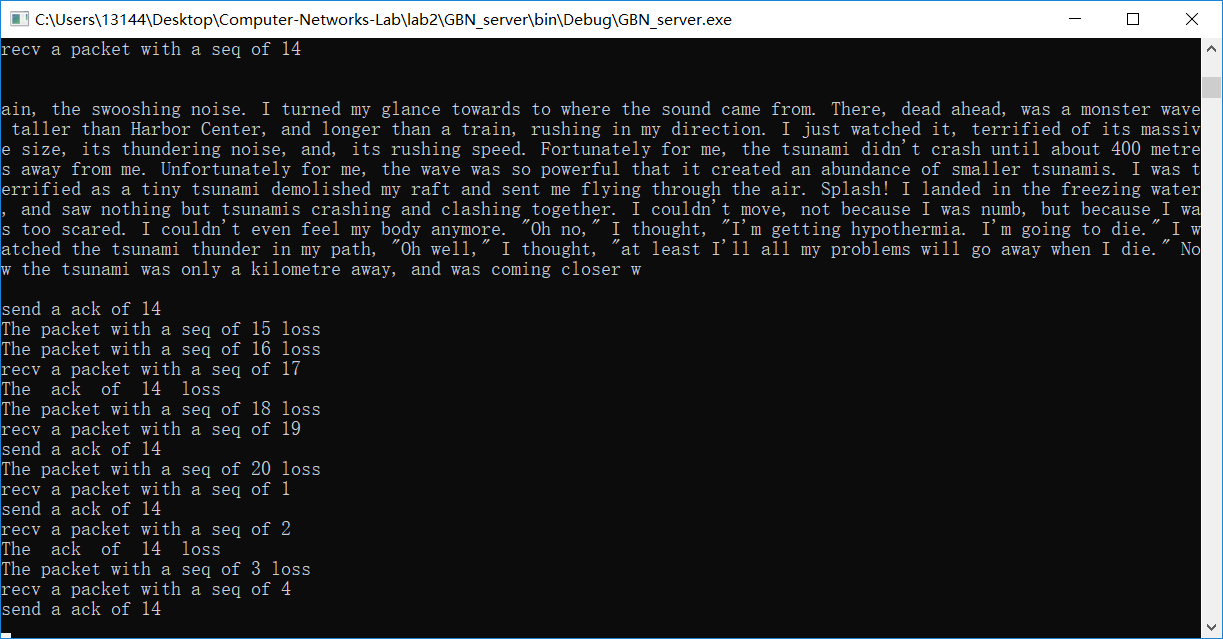
GBN 双向传输：

-testgbn\_Send：

客户端（发送端）：

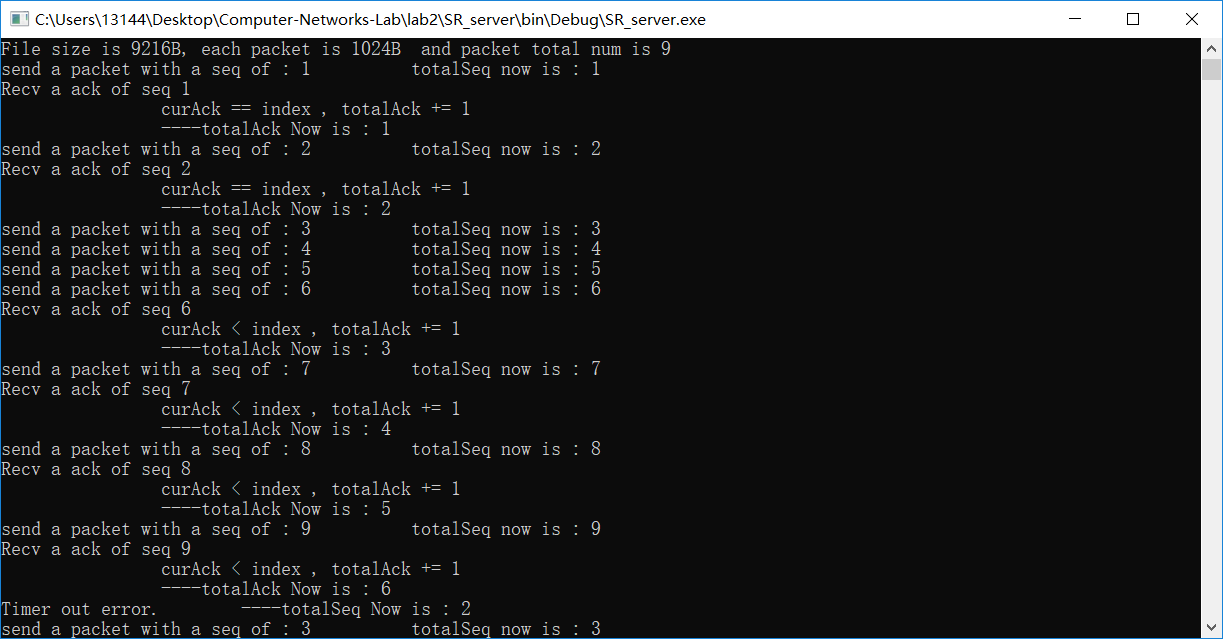


服务器端（接收端）：

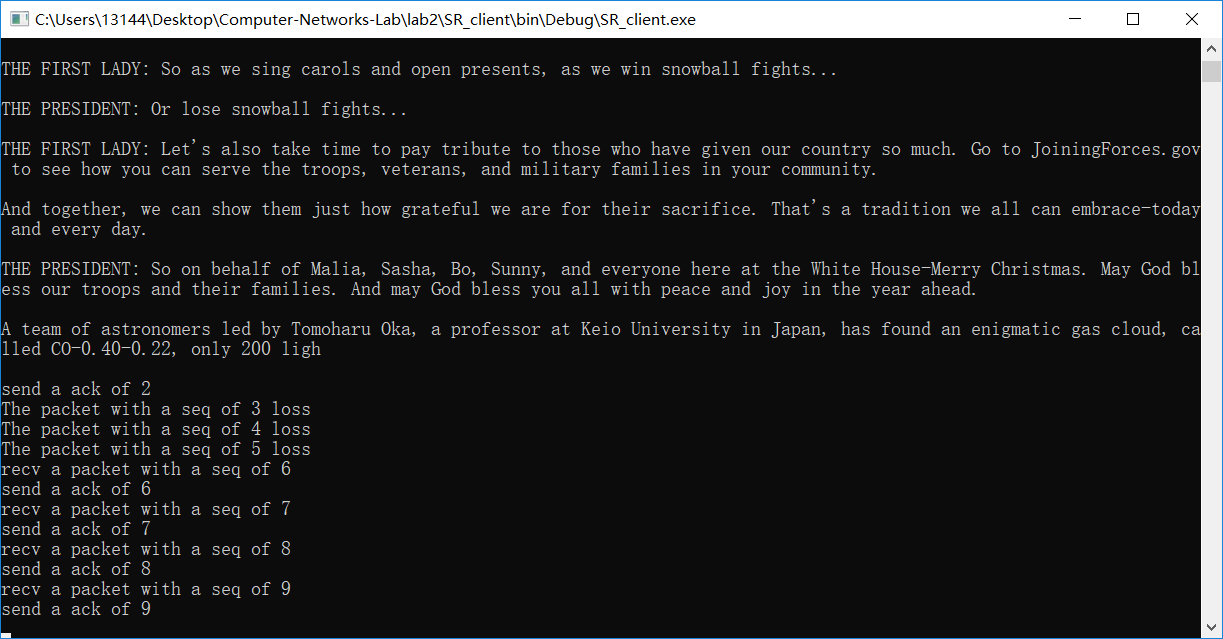


SR 运行结果：

服务器端（发送端）：



客户端（接收端）：



详细注释源程序：

GBN\_server:

//#include "stdafx.h" //创建 VS 项目包含的预编译头文件

#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

const int SEQ\_NUMBER = 33; // seq 的数目

//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

/\*为判断数据传输是否完成添加或修改的控制变量\*/

int totalAck; //确认收到（ack）的包的总数

int finish; //标志位：数据传输是否完成（finish=1->数据传输已完成）

int finish\_S;

/\*为判断数据传输是否完成添加的控制变量\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer,"%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime,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;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

if(totalSeq == totalPacket){//之前发送到了最后一个数据包

if(curSeq > curAck){

totalSeq -= (curSeq-curAck);

}

else if(curSeq < curAck){

totalSeq -= (curSeq - curAck + 20);

}

}

else{//之前没发送到最后一个数据包

totalSeq -= SEND\_WIND\_SIZE;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

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);

//如果收到的序列号大于 curAck 则在这之前的报文段全部被接收

if(curAck <= index){

for(int i= curAck; i <= index;++i){

ack[i] = TRUE;

}

printf("\t\tcurAck <= index , totalAck += %d\n",(index-curAck+1));

totalAck+=(index-curAck+1);

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;

}

totalAck += (SEQ\_SIZE-curAck+index+1);

printf("\t\tcurAck > index , totalAck += %d\n",(SEQ\_SIZE-curAck+index+1));

curAck = index + 1;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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;

}

//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;

}

/\*双向传输的丢包率\*/

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));

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 \* SEQ\_NUMBER];

ZeroMemory(data,sizeof(data));

icin.read(data,1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n",totalPacket);

int recvSize;

//设置标志位

finish=0;

finish\_S=0;

//初始化ack

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(buffer, "Good bye!");

} else if(strcmp(buffer,"-testgbn") == 0) {

finish = 0;

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = TRUE;

}

//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;

totalAck=0;

finish=0;

}

}

break;

case 2://数据传输阶段

/\*为判断数据传输是否完成添加或修改的语句\*/

if(seqIsAvailable() && totalSeq<=(totalPacket-1) ) {

//发送给客户端的序列号从 1 开始

buffer[0] = curSeq + 1;

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

memcpy(&buffer[1],data + 1024 \* totalSeq,1024);

printf("send a packet with a seq of %d\n",curSeq);

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();

printf("\t----totalSeq Now is : %d\n",totalSeq);

waitCount = 0;

}

} else {

//收到 ack

ackHandler(buffer[0]);

printf("\t\t----totalAck Now is : %d\n",totalAck);

waitCount = 0;

/\*为判断数据传输是否完成添加或修改的语句\*/

if(totalAck==totalPacket){//数据传输完成

finish=1;

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

Sleep(500);

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

if(finish==1){

printf("数据传输全部完成！！！\n");

strcpy(buffer,"数据传输全部完成！！！\n");

sendto(sockServer, buffer, strlen(buffer)+1, 0, (SOCKADDR\*)&addrClient,sizeof(SOCKADDR));

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

}

/\* 双向数据传输 \*/

else if(strcmp(buffer,"-testgbn\_Send") == 0){

iMode = 0; //1：非阻塞，0：阻塞

ioctlsocket(sockServer,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

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;

finish\_S=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(sockServer, "-testgbn\_Send", strlen("-testgbn\_Send")+1, 0,

(SOCKADDR\*)&addrClient, sizeof(SOCKADDR));

while (true)

{

//等待 server 回复设置 UDP 为阻塞模式

recvfrom(sockServer,buffer,BUFFER\_LENGTH,0,(SOCKADDR\*)&addrClient, &length);

/\*为判断数据传输是否完成添加或修改的语句\*/

if(!strcmp(buffer,"数据传输全部完成！！！\n")){

finish\_S=1;

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

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(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));

Sleep(5000);

stage = 1;

recvSeq = 0;

waitSeq = 1;

}

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);

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("\n\n%s\n\n",&buffer[1]);

buffer[0] = seq;

recvSeq = seq;

buffer[1] = '\0';

}else{

//如果当前一个包都没有收到，则等待 Seq 为 1 的数据包，不是则不返回 ACK（因为并没有上一个正确的 ACK）

if(!recvSeq){

continue;

}

buffer[0] = recvSeq;

buffer[1] = '\0';

}

b = lossInLossRatio(ackLossRatio);

if(b){

printf("The ack of %d loss\n",(unsigned char)buffer[0]);

continue;

}

sendto(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));

printf("send a ack of %d\n",(unsigned char)buffer[0]);

break;

}

Sleep(500);

}

iMode = 1; //1：非阻塞，0：阻塞

ioctlsocket(sockServer,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

}

if(finish\_S==1){

printf("数据传输全部完成！！！\n\n");

finish\_S=0;

continue;

}

if(finish==1){

finish=0;

continue;

}

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;

}

GBN\_client:

// GBN\_client.cpp : 定义控制台应用程序的入口点。

// #include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <time.h>

#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

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）

BOOL ack[SEQ\_SIZE]; //收到 ack 情况，对应 0~19 的 ack

const int SEQ\_NUMBER = 33; //设置seq的数量

int curSeq; //当前数据包的 seq

int curAck; //当前等待确认的 ack

int totalPacket; //需要发送的包总数

int totalSeq; //已发送的包的总数

int totalAck; //确认收到（ack）的包的总数

int finish; //标志位：数据传输是否完成（finish=1->数据传输已完成）

int finish\_S;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer,"%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime,buffer);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* -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("| -testgbn\_Send [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;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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.");

int index;

for(int i = 0;i< SEND\_WIND\_SIZE;++i){

index = (i + curAck) % SEQ\_SIZE;

ack[index] = TRUE;

}

/\*判断数据传输是否完成添加或修改\*/

if(totalSeq==totalPacket){//之前发送到了最后一个数据包

if(curSeq>curAck){

totalSeq -= (curSeq-curAck);

}

else if(curSeq<curAck){

totalSeq -= (curSeq-curAck+20);

}

}

else{//之前没发送到最后一个数据包

totalSeq -= SEND\_WIND\_SIZE;

}

/\*判断数据传输是否完成添加或修改\*/

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 an ack of seq %d \n",index+1);//从接收方收到的确认收到的序列号

/\*判断数据传输是否完成添加或修改的\*/

if(curAck <= index){

for(int i= curAck; i <= index;++i){

ack[i] = TRUE;

}

printf("\t\tcurAck <= index , totalAck += %d\n",(index-curAck+1));

totalAck+=(index-curAck+1);

curAck = (index + 1) % SEQ\_SIZE;

}else if(curAck != index+1 && curAck > index){

//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;

}

totalAck+=(SEQ\_SIZE-curAck+index+1);

printf("\t\tcurAck > index , totalAck += %d\n",(SEQ\_SIZE-curAck+index+1));

curAck = index + 1;

}

/\*判断数据传输是否完成添加或修改\*/

}

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];

int length = sizeof(SOCKADDR);

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));

ZeroMemory(buffer,sizeof(buffer));

//将测试数据读入内存

std::ifstream icin;

icin.open("test\_Client.txt");

char data[1024 \* SEQ\_NUMBER];

ZeroMemory(data,sizeof(data));

icin.read(data,1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n",totalPacket);

int recvSize ;

finish=0;

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = TRUE;

}

finish=0;

finish\_S=0;

while(true) {

gets(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;

}

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);

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("\n\n\t%s\n\n",&buffer[1]);

buffer[0] = seq;

recvSeq = seq;

buffer[1] = '\0';

}

else {

//如果当前一个包都没有收到，则等待 Seq 为 1 的数据包，不是则不返回 ACK（因为并没有上一个正确的 ACK）

if(!recvSeq) {

continue;

}

buffer[0] = recvSeq;

buffer[1] = '\0';

}

b = lossInLossRatio(ackLossRatio);

if(b) {

printf("The ack of %d loss\n",(unsigned char)buffer[0]);

continue;

}

sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));

printf("send a ack of %d\n",(unsigned char)buffer[0]);

break;

}

Sleep(500);

}

} else if(strcmp(cmd,"-time") == 0){

getCurTime(buffer);

} else if(!strcmp(cmd,"-testgbn\_Send")){

finish\_S=0;

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = TRUE;

}

//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;

sendto(socketClient, "-testgbn\_Send", strlen("-testgbn\_Send")+1, 0,

(SOCKADDR\*)&addrServer, sizeof(SOCKADDR));

Sleep(100);

recvfrom(socketClient,buffer,BUFFER\_LENGTH,0,((SOCKADDR\*)&addrServer),&length);

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

ZeroMemory(buffer,sizeof(buffer));

int iMode = 1; //1：非阻塞，0：阻塞

ioctlsocket(socketClient,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

while(runFlag){

switch(stage){

case 0://发送 205 阶段

buffer[0] = 205;

sendto(socketClient, buffer, strlen(buffer)+1, 0,

(SOCKADDR\*)&addrServer, sizeof(SOCKADDR));

Sleep(100);

stage = 2;

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(socketClient,buffer,BUFFER\_LENGTH,0,((SOCKADDR\*)&addrServer),&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;

totalAck=0;

finish=0;

stage = 2;

}

}

break;

case 2://数据传输阶段

/\*判断数据传输是否完成添加或修改\*/

if(seqIsAvailable() && totalSeq<=(totalPacket-1)){//totalSeq<=(totalPacket-1)：未传到最后一个数据包

/\*判断数据传输是否完成添加或修改\*/

//发送给客户端的序列号从 1 开始

buffer[0] = curSeq + 1;

ack[curSeq] = FALSE;

//数据发送的过程中应该判断是否传输完成->现在此代码已经实现了ok

//为简化过程此处并未实现->现在此代码已经实现了ok

memcpy(&buffer[1],data + 1024 \* totalSeq,1024);

printf("send a packet with a seq of : %d \t totalSeq now is : %d\n",curSeq+1,totalSeq+1);

sendto(socketClient, buffer, BUFFER\_LENGTH, 0,

(SOCKADDR\*)&addrServer, 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(socketClient,buffer,BUFFER\_LENGTH,0,((SOCKADDR\*)&addrServer),&length);

if(recvSize < 0){

waitCount++;

//20 次等待 ack 则超时重传

if (waitCount > 20) {

timeoutHandler();

printf("\t----totalSeq Now is : %d\n",totalSeq);

waitCount = 0;

}

} else{

//收到 ack

ackHandler(buffer[0]);

printf("\t\t----totalAck Now is : %d\n",totalAck);

waitCount = 0;

/\*判断数据传输是否完成添加或修改\*/

if(totalAck == totalPacket){//数据传输完成

finish\_S=1;

break;

}

/\*判断数据传输是否完成添加或修改\*/

}

Sleep(500);

break;

}

/\*判断数据传输是否完成添加或修改\*/

if(finish\_S==1){

printf("数据传输全部完成！！！\n");

strcpy(buffer,"数据传输全部完成！！！\n");

sendto(socketClient, buffer, strlen(buffer)+1, 0, (SOCKADDR\*)&addrServer,sizeof(SOCKADDR));

break;

}

/\*判断数据传输是否完成添加或修改\*/

}

iMode = 0; //1：非阻塞，0：阻塞

ioctlsocket(socketClient,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

}

/\*判断数据传输是否完成添加或修改\*/

if(finish==1){

printf("数据传输全部完成！！！\n\n");

printTips();

finish=0;

continue;

}

if(finish\_S==1){

printTips();

finish\_S=0;

continue;

}

/\*判断数据传输是否完成添加或修改\*/

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;

}

SR\_server:

//#include "stdafx.h" //创建 VS 项目包含的预编译头文件

#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

const int SEQ\_NUMBER = 9;

//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

char dataBuffer[SEQ\_SIZE][BUFFER\_LENGTH];

int curSeq; //当前数据包的 seq

int curAck; //当前等待确认的 ack

int totalPacket; //需要发送的包总数

int totalSeq; //已发送的包的总数

int totalAck; //确认收到（ack）的包的总数

int finish; //标志位：数据传输是否完成（finish=1->数据传输已完成）

int finish\_S;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer,"%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime,buffer);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: seqIsAvailable

// FullName: seqIsAvailable

// Access: public

// Returns: bool

// Qualifier: 当前序列号 curSeq 是否可用

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int 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 0;

}

if(!ack[curSeq]){//ack[curSeq]==FALSE

return 1;

}

return 2;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: timeoutHandler

// FullName: timeoutHandler

// Access: public

// Returns: void

// Qualifier: 超时重传处理函数，

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void timeoutHandler(){

printf("Timer out error.");

int index;

/\*为判断数据传输是否完成添加或修改的语句\*/

if(totalSeq==totalPacket){//之前发送到了最后一个数据包

if(curSeq>curAck){

totalSeq -= (curSeq-curAck);

}

else if(curSeq<curAck){

totalSeq -= (curSeq-curAck+20);

}

}

else{//之前没发送到最后一个数据包

totalSeq -= SEND\_WIND\_SIZE;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

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 seq %d \n",index + 1); //从接收方收到的确认收到的序列号

int all;

int next;

int add;

all = 1;

if(curAck == index){

add = 1;

totalAck += 1;

ack[index] = FALSE;

curAck = (index + 1) % SEQ\_SIZE;

for(int i = 1;i<SEQ\_SIZE;i++){

next=(i+index)%SEQ\_SIZE;

if(ack[next]==TRUE){

ack[next]=FALSE;

curAck = (next + 1) % SEQ\_SIZE;

totalSeq++;

++curSeq;

curSeq %= SEQ\_SIZE;

}

else{

break;

}

}

printf("\t\tcurAck == index , totalAck += %d\n",add);

}

else if(curAck < index && index-curAck + 1 <= SEND\_WIND\_SIZE){//要保证是要接受的消息（在滑动窗口内）

if(!ack[index]){

printf("\t\tcurAck < index , totalAck += %d\n",1);

totalAck += 1;

ack[index] = TRUE;

}

}else if(SEQ\_SIZE - curAck + index + 1 <= SEND\_WIND\_SIZE && curAck > index){ //要保证是要接受的消息（在滑动窗口内）

if(!ack[index]){

printf("\t\tcurAck > index , totalAck += %d\n",1);

totalAck += 1;

ack[index] = TRUE;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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: seqRecvAvailable

// FullName: seqRecvAvailable

// Access: public

// Returns: bool

// Qualifier: 当前收到的序列号 recvSeq 是否在可收范围内

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BOOL seqRecvAvailable(int recvSeq) {

int step;

int index;

index = recvSeq - 1;

step = index - 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;

}

return TRUE;

}

//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;

}

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));

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 \* SEQ\_NUMBER];

ZeroMemory(data,sizeof(data));

icin.read(data,1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n",totalPacket);

int recvSize ;

finish = 0;

finish\_S = 0;

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = FALSE;

}

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){

printf("收到 -time");

getCurTime(buffer);

}else if(strcmp(buffer,"-quit") == 0){

strcpy(buffer,"Good bye!");

}else if(strcmp(buffer,"-testsr") == 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

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = FALSE;

}

ZeroMemory(buffer,sizeof(buffer));

int recvSize;

int waitCount = 0;

finish=0;

printf("Begain to test SR 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;

totalAck=0;

finish=0;

stage = 2;

}

}

break;

case 2://数据传输阶段

/\*判断数据传输是否完成添加或修改\*/

if(seqIsAvailable()==1 && totalSeq<=(totalPacket-1)){//totalSeq<=(totalPacket-1)：未传到最后一个数据包

//发送给客户端的序列号从 1 开始

buffer[0] = curSeq + 1;

ack[curSeq] = FALSE;

//数据发送的过程中应该判断是否传输完成->现在此代码已经实现了ok

//为简化过程此处并未实现->现在此代码已经实现了ok

memcpy(&buffer[1],data + 1024 \* totalSeq,1024);

printf("send a packet with a seq of : %d \t totalSeq now is : %d\n",curSeq+1,totalSeq+1);

sendto(sockServer, buffer, BUFFER\_LENGTH, 0,

(SOCKADDR\*)&addrClient, sizeof(SOCKADDR));

++curSeq;

curSeq %= SEQ\_SIZE;

++totalSeq;

Sleep(500);

}

else if(seqIsAvailable()==2 && totalSeq<=(totalPacket-1)){

++curSeq;

curSeq %= SEQ\_SIZE;

++totalSeq;

//Sleep(500);

printf("\t\tBREAK1!\n");

break;

//printf("\t\tBREAK2!\n");

}

//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();

printf("\t----totalSeq Now is : %d\n",totalSeq);

waitCount = 0;

}

}else{

//收到 ack

ackHandler(buffer[0]);

printf("\t\t----totalAck Now is : %d\n",totalAck);

waitCount = 0;

/\*为判断数据传输是否完成添加或修改的语句\*/

if(totalAck==totalPacket){//数据传输完成

finish=1;

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

Sleep(500);

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

if(finish==1){

printf("数据传输全部完成！！！\n");

strcpy(buffer,"数据传输全部完成！！！\n");

sendto(sockServer, buffer, strlen(buffer)+1, 0, (SOCKADDR\*)&addrClient,sizeof(SOCKADDR));

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

}

else if(strcmp(buffer,"-testsr\_Send") == 0){

iMode = 0; //1：非阻塞，0：阻塞

ioctlsocket(sockServer,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

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;

finish\_S=0;

BOOL b;

curAck=0;

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = FALSE;

}

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

int next;

sendto(sockServer, "-testsr\_Send", strlen("-testsr\_Send")+1, 0,

(SOCKADDR\*)&addrClient, sizeof(SOCKADDR));

while (true)

{

//等待 server 回复设置 UDP 为阻塞模式

recvfrom(sockServer,buffer,BUFFER\_LENGTH,0,(SOCKADDR\*)&addrClient, &length);

/\*判断数据传输是否完成添加或修改\*/

if(!strcmp(buffer,"数据传输全部完成！！！\n")){

finish\_S=1;

break;

}

/\*判断数据传输是否完成添加或修改\*/

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(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));

stage = 1;

recvSeq = 0;

waitSeq = 1;

}

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);

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(seqRecvAvailable(seq)){

recvSeq = seq;

ack[seq-1]=TRUE;

ZeroMemory(dataBuffer[seq-1],sizeof(dataBuffer[seq-1]));

strcpy(dataBuffer[seq-1],&buffer[1]);

buffer[0] = recvSeq;

buffer[1] = '\0';

int tempt=curAck;

if(seq-1==curAck){

for(int i=0;i<SEQ\_SIZE;i++){

//printf("\t\ti=%d",i);

next=(tempt+i)%SEQ\_SIZE;

if(ack[next]){

//输出数据

//printf("\t\ti=%d",i);

printf("\n\n\t\tACK SEQ: %d\n%s\n\n",(next+1)%SEQ\_SIZE,dataBuffer[next]);

curAck=(next+1)%SEQ\_SIZE;

ack[next]=FALSE;

}

else{

break;

}

}

}

}

else{

recvSeq = seq;

buffer[0] = recvSeq;

buffer[1] = '\0';

}

b = lossInLossRatio(ackLossRatio);

if(b){

printf("The ack of %d loss\n",(unsigned char)buffer[0]);

continue;

}

sendto(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));

printf("send a ack of %d\n",(unsigned char)buffer[0]);

break;

}

Sleep(500);

}

iMode = 1; //1：非阻塞，0：阻塞

ioctlsocket(sockServer,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

}

if(finish\_S==1){

printf("数据传输全部完成！！！\n\n");

finish\_S=0;

continue;

}

if(finish==1){

finish=0;

continue;

}

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;

}

SR\_client:

#include <stdlib.h>

#include <WinSock2.h>

#include <time.h>

#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 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

const int SEQ\_NUMBER = 9;

//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

char dataBuffer[SEQ\_SIZE][BUFFER\_LENGTH];

int curSeq;//当前数据包的 seq

int curAck;//当前等待确认的 ack

int totalPacket;//需要发送的包总数

int totalSeq;//已发送的包的总数

int totalAck;//确认收到（ack）的包的总数

int finish;//标志位：数据传输是否完成（finish=1->数据传输已完成）

int finish\_S;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* -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

-testsr\_Send 双向数据传输

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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 (Receive message from the Server)|\n");

printf("| -testsr\_Send [X] [Y] to test the sr\_Send (Send message to the Server) |\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer,"%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime,buffer);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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: seqRecvAvailable

// FullName: seqRecvAvailable

// Access: public

// Returns: bool

// Qualifier: 当前收到的序列号 recvSeq 是否在可收范围内

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BOOL seqRecvAvailable(int recvSeq){

int step;

int index;

index=recvSeq-1;

step = index- 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;

}

return TRUE;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: seqIsAvailable

// FullName: seqIsAvailable

// Access: public

// Returns: bool

// Qualifier: 当前序列号 curSeq 是否可用

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int 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 0;

}

if(!ack[curSeq]){//ack[curSeq]==FALSE

return 1;

}

return 2;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: timeoutHandler

// FullName: timeoutHandler

// Access: public

// Returns: void

// Qualifier: 超时重传处理函数，哪个没收到 ack ，就要重传哪个

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void timeoutHandler(){

printf("Timer out error.");

int index;

/\*判断数据传输是否完成添加或修改\*/

if(totalSeq==totalPacket){//之前发送到了最后一个数据包

if(curSeq>curAck){

totalSeq -= (curSeq-curAck);

}

else if(curSeq<curAck){

totalSeq -= (curSeq-curAck+20);

}

}

else{//之前没发送到最后一个数据包

totalSeq -= SEND\_WIND\_SIZE;

}

/\*判断数据传输是否完成添加或修改\*/

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 seq %d \n",index+1);//从接收方收到的确认收到的序列号

int all;

int next;

int add;

all=1;

/\*SR 判断数据传输是否完成添加或修改\*/

if(curAck == index){

add = 1;

totalAck+=(1);

ack[index]=FALSE;

curAck = (index + 1) % SEQ\_SIZE;

for(int i=1;i<SEQ\_SIZE;i++){

next=(i+index)%SEQ\_SIZE;

if(ack[next]==TRUE){

ack[next]=FALSE;

curAck = (next + 1) % SEQ\_SIZE;

totalSeq++;

++curSeq;

curSeq %= SEQ\_SIZE;

}

else{

break;

}

}

printf("\t\tcurAck == index , totalAck += %d\n",add);

}

else if(curAck < index && index-curAck+1 <= SEND\_WIND\_SIZE){ //要保证是要接受的消息（在滑动窗口内）

if(!ack[index]){

printf("\t\tcurAck < index , totalAck += %d\n",1);

totalAck+=(1);

ack[index] = TRUE;

}

}else if(SEQ\_SIZE-curAck+index+1 <= SEND\_WIND\_SIZE && curAck > index){ //要保证是要接受的消息（在滑动窗口内）

if(!ack[index]){

printf("\t\tcurAck > index , totalAck += %d\n",1);

totalAck += 1;

ack[index] = TRUE;

}

}

/\*SR 判断数据传输是否完成添加或修改\*/

}

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 -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 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));

//将测试数据读入内存

std::ifstream icin;

icin.open("test\_Client.txt");

char data[1024 \* SEQ\_NUMBER];

ZeroMemory(data,sizeof(data));

//icin.read(data,1024 \* 113);

//icin.read(data,1024 \* 4);

icin.read(data,1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n",totalPacket);

int recvSize ;

finish=0;

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = FALSE;

}

finish=0;

finish\_S=0;

while(true){

gets(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")){

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = FALSE;

}

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;

finish=0;

BOOL b;

curAck=0;

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = FALSE;

}

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

int next;

sendto(socketClient, "-testsr", strlen("-testsr")+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);

/\*判断数据传输是否完成添加或修改\*/

if(!strcmp(buffer,"数据传输全部完成！！！\n")){

finish=1;

break;

}

/\*判断数据传输是否完成添加或修改\*/

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;

}

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);

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(seqRecvAvailable(seq)){

recvSeq = seq;

ack[seq-1]=TRUE;

ZeroMemory(dataBuffer[seq-1],sizeof(dataBuffer[seq-1]));

strcpy(dataBuffer[seq-1],&buffer[1]);

buffer[0] = recvSeq;

buffer[1] = '\0';

int tempt=curAck;

if(seq-1==curAck){

for(int i=0;i<SEQ\_SIZE;i++){

//printf("\t\ti=%d",i);

next=(tempt+i)%SEQ\_SIZE;

if(ack[next]){

//输出数据

//printf("\t\ti=%d",i);

printf("\n\n\t\tACK SEQ: %d\n%s\n\n",(next+1)%SEQ\_SIZE,dataBuffer[next]);

curAck=(next+1)%SEQ\_SIZE;

ack[next]=FALSE;

}

else{

break;

}

}

}

}

else{

recvSeq = seq;

buffer[0] = recvSeq;

buffer[1] = '\0';

}

b = lossInLossRatio(ackLossRatio);

if(b){

printf("The ack of %d loss\n",(unsigned char)buffer[0]);

continue;

}

sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));

printf("send a ack of %d\n",(unsigned char)buffer[0]);

break;

}

Sleep(500);

}

}

else if(strcmp(cmd,"-time") == 0){

getCurTime(buffer);

}

else if(!strcmp(cmd,"-testsr\_Send")){

//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

for(int i=0; i < SEQ\_SIZE; ++i){

ack[i] = FALSE;

}

ZeroMemory(buffer,sizeof(buffer));

int recvSize;

int waitCount = 0;

finish\_S=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;

sendto(socketClient, "-testsr\_Send", strlen("-testsr\_Send")+1, 0,

(SOCKADDR\*)&addrServer, sizeof(SOCKADDR));

Sleep(100);

recvfrom(socketClient,buffer,BUFFER\_LENGTH,0,((SOCKADDR\*)&addrServer),&len);

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

ZeroMemory(buffer,sizeof(buffer));

int iMode = 1; //1：非阻塞，0：阻塞

ioctlsocket(socketClient,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

while(runFlag){

switch(stage){

case 0://发送 205 阶段

buffer[0] = 205;

sendto(socketClient, buffer, strlen(buffer)+1, 0,

(SOCKADDR\*)&addrServer, 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(socketClient,buffer,BUFFER\_LENGTH,0,((SOCKADDR\*)&addrServer),&len);

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;

totalAck=0;

finish\_S=0;

stage = 2;

}

}

break;

case 2://数据传输阶段

/\*判断数据传输是否完成添加或修改\*/

if(seqIsAvailable()==1 && totalSeq<=(totalPacket-1)){//totalSeq<=(totalPacket-1)：未传到最后一个数据包

/\*判断数据传输是否完成添加或修改\*/

//发送给客户端的序列号从 1 开始

buffer[0] = curSeq + 1;

ack[curSeq] = FALSE;

//数据发送的过程中应该判断是否传输完成->现在此代码已经实现了ok

//为简化过程此处并未实现->现在此代码已经实现了ok

memcpy(&buffer[1],data + 1024 \* totalSeq,1024);

printf("send a packet with a seq of : %d \t totalSeq now is : %d\n",curSeq+1,totalSeq+1);

sendto(socketClient, buffer, BUFFER\_LENGTH, 0,

(SOCKADDR\*)&addrServer, sizeof(SOCKADDR));

++curSeq;

curSeq %= SEQ\_SIZE;

++totalSeq;

Sleep(500);

}

else if(seqIsAvailable()==2 && totalSeq <= totalPacket-1) {

++curSeq;

curSeq %= SEQ\_SIZE;

++totalSeq;

//Sleep(500);

printf("\t\tBREAK1!\n");

break;

}

//等待 Ack，若没有收到，则返回值为-1，计数器+1

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();

printf("\t----totalSeq Now is : %d\n",totalSeq);

waitCount = 0;

}

}else{

//收到 ack

ackHandler(buffer[0]);

printf("\t\t----totalAck Now is : %d\n",totalAck);

waitCount = 0;

/\*判断数据传输是否完成添加或修改\*/

if(totalAck==totalPacket){//数据传输完成

finish\_S = 1;

break;

}

/\*判断数据传输是否完成添加或修改\*/

}

Sleep(500);

break;

}

/\*判断数据传输是否完成添加或修改\*/

if(finish\_S==1){

printf("数据传输全部完成！！！\n");

strcpy(buffer,"数据传输全部完成！！！\n");

sendto(socketClient, buffer, strlen(buffer)+1, 0, (SOCKADDR\*)&addrServer,sizeof(SOCKADDR));

break;

}

/\*判断数据传输是否完成添加或修改\*/

}

iMode = 0; //1：非阻塞，0：阻塞

ioctlsocket(socketClient,FIONBIO, (u\_long FAR\*) &iMode);//非阻塞设置

}

/\*判断数据传输是否完成添加或修改\*/

if(finish==1){

printf("数据传输全部完成！！！\n\n");

printTips();

finish=0;

continue;

}

if(finish\_S==1){

printTips();

finish\_S=0;

continue;

}

/\*判断数据传输是否完成添加或修改\*/

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;

}