

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议-停等协议、GBN协议的设计与实现 | | | | | |
| 姓名 | 郑旭然 | | 院系 | 软件工程 | | |
| 班级 | 2037102 | | 学号 | 120L020719 | | |
| 任课教师 | 李全龙 | | 指导教师 | 李全龙 | | |
| 实验地点 | 格物楼207 | | 实验时间 | 2022.10.10 | | |
| 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： |
| 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： |
| 1) 基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。  2) 模拟引入数据包的丢失，验证所设计协议的有效性。  3) 改进所设计的停等协议，支持双向数据传输；  4) 基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。  5) 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。  6) 模拟引入数据包的丢失，验证所设计协议的有效性。  7) 改进所设计的 GBN 协议，支持双向数据传输；  8) 将所设计的 GBN 协议改进为 SR 协议。 |
| 实验过程： |
| 1. **协议的数据分组格式、确认分组格式、各个域作用**  |  |  |  | | --- | --- | --- | | Seq | Data | \0 |   GBN与SR协议的数据分组格式：  Seq 为 1 个字节，取值为 0~255；Data 不超过1024 个字节，为传输的数据；末尾放入 EOF0，表示结尾。   |  |  | | --- | --- | | ACK | \0 |   GBN与SR协议的确认分组格式：  ACK 字段为一个字节，表示序列号数值；末尾放入EOF0，表示数据结束。   1. **协议两端程序流程图**     图 1 GBN发送方扩展FSM    图 2 GBN接收方扩展FSM    图 3 SR发送与接收方流程   1. **协议典型交互过程**     图 4 GBN协议典型交互过程    图 5 SR协议典型交互过程   1. **数据分组丢失验证模拟方法**   为了模拟数据丢失的情况，接收端接收数据时以某一个概率“将数据丢弃”，也就是表现为最终不能发回ACK。这个概率将在下面的函数中进行设置。   1. **程序实现的主要类（或函数）及其主要作用**   **GBN\_client**   |  |  | | --- | --- | | void getCurTime(char \*ptime) | 获取当前系统时间，结果存入 ptime 中 | | void printTips() | 打印提示信息 | | 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 | | 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 |   **GBN\_server**   |  |  | | --- | --- | | 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 | | 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 |   **SR\_client**   |  |  | | --- | --- | | void getCurTime(char \*ptime) | 获取当前系统时间，结果存入 ptime 中 | | void printTips() | 打印提示信息 | | 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 | | BOOL seqRecvAvailable(int recvSeq) | 当前收到的序列号 recvSeq 是否在可收范围内 | | int seqIsAvailable(){ | 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 | | void timeoutHandler() | 超时重传处理函数，哪个没收到 ack ，就要重传哪个 |   **SR\_server**   |  |  | | --- | --- | | 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 | | BOOL seqRecvAvailable(int recvSeq) | 当前收到的序列号 recvSeq 是否在可收范围内 | | int seqIsAvailable(){ | 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 | | void timeoutHandler() | 超时重传处理函数 |  1. **UDP 编程的主要特点** 2. UDP 使用尽最大努力交付，即不保证可靠交付，主机不需要维持复杂的连接状态表。支持一对一、一对多、多对一和多对多的交互通信。 3. UDP 是面向报文的，发送方的 UDP 对应用程序交下来的报文，在添加首部后就向下交付 IP 层。UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。 4. UDP是数据报协议，无连接、不可靠，追求传输效率的一种通信协议，数据的发送和接收是同步的。在进行通信之前，不需要建立连接。其传输效率比TCP高。对其服务器而言，并没有三次握手的过程。因此和TCP相比，少了被动监听(listen)和(accept)。只需要创建通信设备，绑定IP地址和端口号，然后进行数据的收发。 |
| 实验结果： |
| **在实验中有以下几个验收要点：**   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）。**      1. **模拟引入数据包的丢失，验证所设计协议的有效性。**   为了模拟数据丢失的情况，接收端接收数据时以某一个概率“将数据丢弃”，也就是表现为最终不能发回ACK。           1. **改进所设计的 GBN 协议，支持双向数据传输；**      1. **将所设计的 GBN 协议改进为 SR 协议。**      1. **基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。**   可以看到运行过后result.txt中内容和原txt内容一致。 |
| 问题讨论： |
| 1.基于 UDP 实现的 GBN 协议，可以不进行差错检测，可以利用 UDP 协议差错检测。  2.自行设计数据帧的格式，应至少包含序列号Seq和数据两部分。  3.自行定义发送端序列号Seq比特数L以及发送窗口大小W，应满足条件 W+1≤2L。  4.为了模拟ACK丢失，可以利用模N运算，每N次模拟丢包，或者每N次模拟接收。因为只是模拟，这个操作既可以在发送端也可以在接收端，如果在发送端，则少发送数据包，在接收端则不发回ACK。  5.当设置服务器端发送窗口的大小为1时，GBN协议就是停-等协议。  6.分组的规定：发送端末尾加上"Seq = %d"的字符串，接收端返回ACK时，在返回的packet的数据末尾加入字符串"ACK: %d"。 |
| 心得体会： |
| 1.在接收 UDP 包时，如果接收包时给定的buffer太小的话，会发生异常，要捕获异常，相应调整buffer的大小，和给出反馈信息。  2.如果不允许丢包的情况出现的话，要有重发机制来保证，如：每发一条信息，只有收到正确的反馈信息的时候，才证明成功，不然就重试一定次数后才证明真正失败。  3.可以用SetSockOption等来设定接收等待时间，以免傻等。 |

附录：程序源代码

GBN：client.cpp

// 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>

#include <iostream>

#include <cmath>

using *namespace* std;

// #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

*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

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

// 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("|     -testgbn2 [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("\*\*\*\*\*Time out\n");

*int* index;

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

    {

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

        ack[index] = TRUE;

    }

*int* temp = curSeq - curAck;

    totalSeq -= temp > 0 ? temp : temp + SEQ\_SIZE;

    curSeq = curAck;

    printf("\*\*\*\*\*Rensend from Packet %d\n\n", totalSeq);

}

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

// 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 %d \n", index);

    // 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;

        }

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

    }

    else

    {

        if (ack[index] == FALSE)

        {

            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* 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;

*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("client\_file.txt");

    icin.seekg(0, *ios*::end);

*int* fileSize = (*int*)icin.tellg();

    icin.seekg(0, *ios*::beg);

*char* data[fileSize + 1];

    icin.read(data, fileSize);

    data[fileSize] = 0;

    icin.close();

    totalPacket = ceil(sizeof(data) / 1024.0);

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

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

    {

        ack[i] = TRUE;

    }

    while (true)

    {

        gets(buffer);

        // printf("%s\n", 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* stage = 0;

*BOOL* b;

*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);

                //////////////////////TODO

                if (strcmp(buffer, "Data Transfer Is Complete") == 0)

                {

                    break;

                }

                switch (stage)

                {

                case 0: //等待握手阶段

                    if ((*unsigned* *char*)buffer[0] == 205)

                    {

                        printf("Ready for file transmission\n");

                        buffer[0] = 200;

                        buffer[1] = '\0';

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

                    printf("\nThe packet wished: %d\n", waitSeq - 1);

                    // 包丢失

                    if (b)

                    {

                        printf("The packet with a seq of %d loss\n", seq - 1);

                        break;

                    }

                    // 包没有丢失

                    printf("recv a packet with a seq of %d\n", seq - 1);

                    //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';

                    }

                    // ACK 是否丢失

                    b = lossInLossRatio(ackLossRatio);

                    if (b)

                    {

                        printf("The ack of %d loss\n", (*unsigned* *char*)buffer[0] - 1);

                        continue;

                    }

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

                    printf("send a ack of %d\n", (*unsigned* *char*)buffer[0] - 1);

                    break;

                }

                Sleep(500);

            }

        }

        else if (!strcmp(cmd, "-testgbn2"))

        {

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

            // sendto(socketClient, "-testgbn2", strlen("-testgbn2")+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;

                    buffer[1] = '\0';

                    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), &length);

                    if (recvSize < 0)

                    {

                        ++waitCount;

                        printf("recv: %d, waitCount: %d\n", recvSize, waitCount);

                        if (waitCount > 20)

                        {

                            runFlag = false;

                            printf("200 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\n", sizeof(data), totalPacket);

                            curSeq = 0;

                            curAck = 0;

                            totalSeq = 0;

                            waitCount = 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)

                    { // 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\n", curSeq);

                        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)

                        {

                            waitCount = 0;

                            timeoutHandler();

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

                        }

                    }

                    else

                    {

                        //收到 ack

                        ackHandler(buffer[0]);

                        waitCount = 0;

                    }

                    Sleep(500);

                    break;

                }

                // 发送完，验证是否接收到全部ACK

                if (totalSeq == totalPacket)

                {

*BOOL* isFinish = TRUE;

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

                    {

                        if (ack[i] == FALSE)

                        {

                            isFinish = FALSE;

                        }

                    }

                    // 收到了全部ACK，发送完成信号，退出运行（runFlag = false）

                    if (isFinish == TRUE)

                    {

                        // printf("Data Transfer Is Complete\n");

                        strcpy(buffer, "Data Transfer Is Complete");

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

                        // break;

                        totalSeq = 0;

                        runFlag = false;

                    }

                }

            }

            iMode = 0;                                                // 1：非阻塞，0：阻塞

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

        }

        // -time -quit 命令发送

        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;

}

GBN：server.cpp

//#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>

#include <iostream>

#include <cmath>

using *namespace* std;

// #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(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: 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("\*\*\*\*\*Time out\n");

*int* index;

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

    {

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

        ack[index] = TRUE;

    }

*int* temp = curSeq - curAck;

    totalSeq -= temp > 0 ? temp : temp + SEQ\_SIZE;

    curSeq = curAck;

    printf("\*\*\*\*\*Rensend from Packet %d\n\n", totalSeq);

}

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

// 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;

        }

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

    }

    else

    {

        if (ack[index] == FALSE)

        {

            for (*int* i = curAck; i < SEQ\_SIZE; ++i)

            {

                ack[i] = TRUE;

            }

            for (*int* i = 0; i <= index; ++i)

            {

                ack[i] = TRUE;

            }

            curAck = index + 1;

        }

    }

}

//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("server\_file.txt");

    icin.seekg(0, *ios*::end);

*int* fileSize = (*int*)icin.tellg();

    icin.seekg(0, *ios*::beg);

*char* data[fileSize + 1];

    icin.read(data, fileSize);

    data[fileSize] = 0;

    icin.close();

    totalPacket = ceil(sizeof(data) / 1024.0);

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

*int* recvSize;

    //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;

        }

        else if (strcmp(buffer, "Data Transfer Is Complete") == 0)

        {

            printf("Data Transfer Is Complete\n");

            continue;

        }

        printf("recv from client: %s\n", buffer);

*char* cmd[128];

        sscanf(buffer, "%s%f%f", &cmd, &packetLossRatio, &ackLossRatio);

        if (strcmp(buffer, "-time") == 0)

        {

            getCurTime(buffer);

        }

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

        {

            strcpy(buffer, "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;

                    buffer[1] = '\0';

                    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("200 Timeout error\n");

                            break;

                        }

                        Sleep(500);

                        continue;

                    }

                    else

                    {

                        // waitCount = 0;

                        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\n", sizeof(data), totalPacket);

                            curSeq = 0;

                            curAck = 0;

                            totalSeq = 0;

                            waitCount = 0;

                            stage = 2;

                        }

                    }

                    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)

                    {

                        //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

                    // 无论有没有收到ACK，都会继续传数据

                    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)

                        {

                            waitCount = 0;

                            timeoutHandler();

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

                        }

                    }

                    else

                    {

                        //收到 ack

                        ackHandler(buffer[0]);

                        waitCount = 0;

                    }

                    Sleep(500);

                    break;

                }

                // 发送完，验证是否接收到全部ACK

                if (totalSeq == totalPacket)

                {

*BOOL* isFinish = TRUE;

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

                    {

                        if (ack[i] == FALSE)

                        {

                            isFinish = FALSE;

                        }

                    }

                    // 收到了全部ACK，发送完成信号，退出运行（runFlag = false）

                    if (isFinish == TRUE)

                    {

                        // printf("Data Transfer Is Complete\n");

                        strcpy(buffer, "Data Transfer Is Complete");

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

                        // break;

                        totalSeq = 0;

                        runFlag = false;

                    }

                }

            }

        }

        /\* 双向数据传输 \*/

        else if (strcmp(cmd, "-testgbn2") == 0)

        {

            iMode = 0;                                              // 1：非阻塞，0：阻塞

            ioctlsocket(sockServer, FIONBIO, (*u\_long* FAR \*)&iMode); // 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* stage = 0;

*BOOL* b;

*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, "-testgbn2", strlen("-testgbn2") + 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);

                if (strcmp(buffer, "Data Transfer Is Complete") == 0)

                {

                    break;

                }

                switch (stage)

                {

                case 0: //等待握手阶段

                    if ((*unsigned* *char*)buffer[0] == 205)

                    {

                        printf("Ready for file transmission\n");

                        buffer[0] = 200;

                        buffer[1] = '\0';

                        sendto(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);

                    printf("\nThe packet wished: %d\n", waitSeq - 1);

                    if (b)

                    {

                        printf("The packet with a seq of %d loss\n", seq - 1);

                        continue;

                    }

                    printf("recv a packet with a seq of %d\n", seq - 1);

                    //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] - 1);

                        continue;

                    }

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

                    printf("send a ack of %d\n", (*unsigned* *char*)buffer[0] - 1);

                    break;

                }

                Sleep(500);

            }

            iMode = 1;                                              // 1：非阻塞，0：阻塞

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

        }

        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.cpp

#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

//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

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

/\*  -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("| -testsr [X] [Y] to test the SR        |\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 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;

}

*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;

*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));

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

    {

        ack[i] = FALSE;

    }

    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("Begin to test SR protocol, please don't abort the process\n");

            printf("The loss ratio of packet is %.2f, the loss ratio of ack is %.2f\n", packetLossRatio, ackLossRatio);

*int* stage = 0;

            BOOL b;

            curAck = 0;

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

            {

                ack[i] = FALSE;

            }

*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

*int* next;

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

            // 保存到文件

            std::ofstream out\_result;

            out\_result.open("result.txt", std::ios::out | std::ios::trunc);

            if (!out\_result.is\_open())

            {

                printf("File Open Error.\n");

                continue;

            }

            while (true)

            {

                //等待 server 回复设置 UDP 为阻塞模式

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

                if (!strcmp(buffer, "Data Transfer Is Complete\n"))

                {

                    break;

                }

                switch (stage)

                {

                case 0: //等待握手阶段

                    if ((*unsigned* *char*)buffer[0] == 205)

                    {

                        printf("Ready for file transmission\n");

                        buffer[0] = 200;

                        buffer[1] = '\0';

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

                        stage = 1;

                        recvSeq = 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);

                        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++)

                            {

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

                                if (ack[next])

                                {

                                    //输出数据

                                    // printf("\n%s\n", dataBuffer[next]);

                                    out\_result << 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);

            }

            out\_result.close();

        }

        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.cpp

//#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>

#include <iostream>

#include <cmath>

using *namespace* std;

// #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

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

// 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，滑动窗口内的数据帧都要重传

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

*void* timeoutHandler()

{

    printf("\*\*\*\*\*\*Time out\n");

    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;

    }

    curSeq = curAck;

    printf("\*\*\*\*\*Rensend from Packet %d\n\n", totalSeq + 1);

}

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

// 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* next;

    if (curAck == index)

    {

        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;

            }

        }

    }

    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])

        {

            totalAck += 1;

            ack[index] = TRUE;

        }

    }

    else if (SEQ\_SIZE + index - curAck + 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])

        {

            totalAck += 1;

            ack[index] = TRUE;

        }

    }

}

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

// 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;

    }

    //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("server\_file.txt");

    icin.seekg(0, *ios*::end);

*int* fileSize = (*int*)icin.tellg();

    icin.seekg(0, *ios*::beg);

*char* data[fileSize + 1];

    icin.read(data, fileSize);

    data[fileSize] = 0;

    icin.close();

    totalPacket = ceil(sizeof(data) / 1024.0);

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

*int* recvSize;

    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)

        {

            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;

            printf("Begin 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;

                            stage = 2;

                        }

                    }

                    break;

                case 2: //数据传输阶段

                    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

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

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

                        // printf("totalSeq now is: %d\n", 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++;

                        break;

                    }

                    //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 ack

                        ackHandler(buffer[0]);

                        waitCount = 0;

                        if (totalAck == totalPacket)

                        { //数据传输完成

                            printf("Data Transfer Complete\n");

                            strcpy(buffer, "Data Transfer Complete\n");

                            runFlag = false;

                            break;

                        }

                    }

                    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;

}