

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议- 停等协议与GBN协议的设计与实现 | | | | | |
| 姓名 | 刘天瑞 | | 院系 | 未来技术学院 | | |
| 班级 | 20W0361 | | 学号 | 7203610121 | | |
| 任课教师 | 刘亚维 | | 指导教师 | 刘亚维 | | |
| 实验地点 | G001 | | 实验时间 | 2022.10.12 | | |
| 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。  答：   1. 理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于UDP设计并实现一个停等协议的过程与技术。 2. 理解滑动窗口协议的基本原理；掌握GBN的工作原理；掌握基于UDP设计并实现一个GBN协议的过程与技术。 |
| 实验内容： |
| 概述本次实验的主要内容，包含的实验项等。  答：  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；（选作内容，加分项目，可以当堂完成或课下完成）  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；（选作内容，加分项目，可以当堂完成或课下完成）  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；（选作内容，加分项目，可以当堂完成或课下完成）  8) 将所设计的GBN协议改进为SR协议。（选作内容，加分项目，可以当堂完成或课下完成） |
| 实验过程： |
| 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。  将GBN协议的滑动窗口大小设为1，则转换为停-等协议，故只需先写出GBN协议的代码，便可以很容易得到停-等协议，然后在GBN的基础上添加缓存和更多计时器就可将其修改为SR协议。     1. **设计GBN协议：**   GBN协议，即回退N步协议(go back N),如果某个报文段没有被正确的接收，那么从这个报文段到后面的报文段都要重新发送，在GBN协议中，返回某个ACK，则代表该ACK序号及其之前的报文段均已经收到。  单向通信的GBN协议由客户端发送请求，服务器端收到确认后向客户端发送数据。  对于服务器端，在本实验中，使用UDP协议传输数据（在具体实现时，读取我们准备的测试文件进行传输），首先，服务器端等待客户端的请求，接收来自客户端的消息当服务器端接收到客户端发来的数据传输请求时，服务器端将数据分割成一个个数据报进行发送，并在服务器端的命令行界面中打印发送出的数据报相关信息。在发送完成后，客户端即启动计时器，开始等待客户端的ACK信息。当收到客户端回复的ACK时，服务器端确认ACK及其之前的数据报均已收到，服务器端的发送窗口可以滑动，正常发送下一个数据报，计时器重新计时。若在计时器超时前没有收到ACK，则全部重传窗口内最大ACK之后的所有已发送的数据报。  对于客户端，客户端使用UDP协议向服务器端请求数据，接收服务器端发送的数据报并返回确认信息ACK。在实际的程序中，必须模拟ACK丢失，然后由服务器端检测到超时重传的情况。  在实际的程序中，首先创建一个套接字，并将其绑定在指定的端口（12340）上。客户端解析命令行中的请求信息。根据不同的字符串，采取不同的应对策略。  当执行数据传输指令（-testgbn）时，客户端首先向服务器端发送请求信息，然后服务器端解析请求，进行握手，由服务器向客户端发送一个205状态码（示例程序中的定义），在客户端收到服务器端发来的205状态码后，回复一个200大小的状态码，表示客户端可以接收数据；在完成握手（服务器收到200状态码）之后，服务器端便可以读取本地文件，将其切分，封装到数据报中，通过GBN协议的规则（ACK采取累积确认机制，若丢失则重传之后的所有数据报等等）发送数据，服务器端读取本地文件，放到缓存中，发送给客户端。  使用UDP协议传输数据，收发数据使用非阻塞方式，为了保证数据可靠传输，我们需要设计数据传输步骤，在实验指导书里的参考代码基础上进行改进。  1.服务器端(server)：  (i)命令解析  为了测试客户端与服务器端的通信交互，方便操作，设置了此过程。  首先，服务器接收客户端发来的请求数据：  “-time”表示客户端请求获取当前时间，服务器回复当前时间；  “-quit”表示客户端退出，服务器回复“Good bye!”；  “-testgbn”表示客户端请求开始测试GBN协议，服务器开始进入GBN传输状态；  其他数据：则服务器直接回复原数据。  (ii)数据传输数据帧格式定义  在以太网中，数据帧的MTU为1500字节，所以UDP数据报的数据部分应小于1472字节（除去IP头部20字节与UDP头的8字节），为此，定义UDP数据报的数据部分格式为：    图1 UDP数据报的数据部分格式  Seq为1个字节，取值为0~255，（故序列号最多为256个）；  Data≤1024 个字节，为传输的数据；  最后一个字节放入EOF0，表示结尾。  (iii)传输开始与结束  模拟TCP三次握手与四次挥手，这里只进行一次握手与挥手。  传输开始时，向客户端发送序列号为200的分组，表示服务器准备好传输数据，等待客户端。  回复205分组，表示客户端准备好传输数据，开始传输数据。  totalSeq > totalPacket时，表示已经输出分组数量大于总要传输的分组数量，但这时可能有的ACK还没有收到，我们需要检查分组ACK是否收到，若有没收到的ACK继续向客户端发送空分组，否则当客户端的ACK丢失，客户端不会向服务器端重传ACK；若都收到，则向客户端发送序列号为204的分组，表示传输完成，结束分组传输。    图2 UDP协议服务器端传输数据流程图  2.客户端(client)：  (i) ACK数据帧定义    图3 ACK数据帧部分格式  因为是从服务器端到客户端的单向数据传输，因此ACK数据帧不包含任何数据，只需要将ACK发送给服务器端即可。  ACK字段为一个字节，表示序列号数值；末尾放入0，表示数据结束。    (ii)命令设置  客户端的命令和服务器端的解析命令向对应，获取当前用户输入并 发送给服务器并等待服务器返回数据，例如输入  “-time”可以得到服务器的当前时间。  此处重点介绍“-testgbn [X] [Y]”命令，[X],[Y]均为[0,1]的小数，其中：  [X]表示客户端的数据包丢失率，模拟网络中报文丢失；  [Y]表示客户端的ACK的丢失率（使用随机函数完成）。  如果用户不输入，则默认丢失率均为0.2。  (iii)丢包  使用LossInLossRatio()函数，输入丢失率（分组或客户端），在函数内首先获得1到100的随机数，根据丢失率获得随机数限制，当随机数在限制范围内表示超时，输出true；否则表示非超时，输出false。  (iv)传输开始与结束  收到服务器端的序列号为200的分组，回复205分组，表示准备好传输，等待服务器端的数据。  收到服务器端的序列号为204的分组后，结束分组接收。    图4 UDP协议客户端传输数据流程图    3.扩展(expand)：  (i)双向数据传输：  在上一步编写的程序基础上，将服务端与客户端通过多线程整合到一起，实现一个端口能够同时做到收发。  具体方法如下所示：  在主线程中使用客户端程序，在进入while前创建子线程，令其handle函数服务器端函数，将此代码复制两份，然后将其中一份的发送和接收的IP地址（SERVER\_IP,CLIENT\_IP）与端口号（SERVER\_PORT,CLIENT\_PORT）调换（若接收地址为0.0.0.0，变为发送地址时需要使用本机具体地址或环回地址进行验证），使通信的两个端口能够发送和接收IP和端口对应。  (ii)停-等协议：  将GBN协议中的窗口大小（SEND\_WIND\_SIZE）改为1，GBN协议变为停-等协议，且能实现双向数据传输。  (iii)滑动窗口协议(C/S)：  将输入输出的数据分别通过文件读写，在传输与接收之前分别通过read和write，即读和写一个数据报中数据大小的缓存，实现文件有序可靠的发送和接收。    **二、设计SR协议：**  在GBN协议的基础上通过为每一个发送窗口增加缓存和计时器，为接收窗口增加缓存，从而实现SR协议。  在SR协议中，接收方设置缓冲区，且为每个报文段单独设置计时器，如果某个报文段没有被正确接收但是后面的报文段被正确接收了，那么就只需要重发这一个报文段，之前收到的乱序报文段存在缓存中，在接收方确认整理排序之后就收到了之前的所有报文段。在SR协议中，返回的ACK仅为当前接收成功的报文段序号。  SR协议与GBN协议在握手阶段相同，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。它们的不同之处主要是在数据重传方面。对于SR协议，当超时事件发生时，服务器端并不重传窗口内所有已发送的数据报，而是只发送窗口内那些没有收到ACK的数据报；对于客户端，其也有一个窗口。它接受服务端发来的数据报并返回确认信息ACK，但并未采用累积确认机制。此时的客户端收到哪个序号的数据包就返回哪个序号的ACK。即使前面有还没有收到的分组，也会将该组缓存下来，待全部收到按顺序的一组数据报时，将其交付给上层协议。  以下所阐述的都是在以上GBN协议的基础上进行修改的内容：  1.服务器端(server)：  (i)缓存和重传：  创建全局变量，为每个发送窗口创建缓存（cache[SEND\_WIND\_SIZE + 1][DATA\_SIZE + 1]）与计时器(counter[SEND\_WIND\_SIZE])。  每个计时器在关闭时为-1，在开启时表示超时时间。每次循环，使用click()函数将所有开启且没有收到ACK的计时器+1。因为在发送与接收两端有足够的sleep()函数，故可以忽略传输时延对sleep的总时间带来的影响，可以使用循环次数来表示超时计数。  发送每个分组，在cache对应序号中进行缓存，并开启对应序号计时器（设置为0）。当接收到ACK时，关闭对应的计时器；若超时，则将计时器重置，并重新发送cache中缓存的分组。  (ii)传输结束：  因为每个窗口都设置有自身的计时器，所以可以不用在发送完毕后继续发送空报文。  2.客户端(client)：  创建全局变量，为每个接收窗口创建缓存（cache[SEND\_WIND\_SIZE + 1][DATA\_SIZE + 1]）。  当分组失序到达时，首先在ack函数中进行标记，然后在cache中缓存分组。  当后到达的分组非失序时，进行累计确认，从最开始失序的分组开始（如果有），将所有缓存的分组进行接收（写入文件），保证数据有序。  其他运行部分与GBN协议基本相同，这里便不再赘述。 |
| 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。  检测用传输数据：美国之音（VOA）听力原文的txt文件（分别命名为server\_in.txt , client\_in.txt），我重复地复制粘贴了七遍，便于比较。    图5 server\_in.txt文本文件内容   1. **GBN协议实现单向传输（UDP协议的基础功能）**   如下图所示，在传输双方文件目录内分别有client\_in.txt和server\_in.txt作为检测用传输数据，内部数据和以上图示一样。    图6 文档目录下的client\_in.txt和server\_in.txt文本文件  首先输入-time进行请求获取对方当前时间的测试：    图5 GBN\_client与GBN\_server的请求获取对方当前时间测试运行结果  然后在客户端输入-testgbn，进行单向传输的实验：  客户端(GBN\_client)：          图6 GBN协议客户端发送数据文件的运行结果  服务器端(GBN\_server)：    图7 GBN协议服务器端接收数据文件的运行结果  可以看出，第二个包在传输过程中丢失，由于服务器端迟迟没有收到ACK2，在超时后，服务器便对第二个包进行了重传（对第三、四、五个包的分析情况相同），在完成全部数据报的传输后，打印提示字符，结束传输。   1. **GBN双向数据传输功能**   在服务器端命令行中输入“-testgbn”，测试反向传输（由客户端向服务器发送数据）功能。运行结果如下图所示：  客户端（此时发送数据，作“服务器”）：          图8 GBN协议“服务器端”发送数据文件的运行结果    服务器端（此时接收数据，作“客户端”）：    图9 GBN协议“客户端”接收数据文件的运行结果    在此次传输过程中，无数据报丢失，于是按照GBN协议的规则无需重传。   1. **C/S架构文件传输功能**   **(i)GBN协议的文件传输**  在GBN协议中，设计出较简单的文件传输机制：在传输数据时，读取文件，并将其封装到数据报中，在命令行中为方便查看GBN传输的过程，仅在传输开始时显示数据报的数量以及总大小。综上所述，GBN协议传输成功，即代表着实现文件传输功能。  由服务器端向客户端传输的文件如下：    图10 由服务器端向客户端传输的文件server\_in.txt  其大小为8664B，对应着传输时的8个数据报：    图11 传输数据文件大小所对应的数据报数量  由客户端向服务器端反向传输的文件如下：    图12 由客户端向服务器端反向传输的文件client\_in.txt  其大小为8663B，对应着传输时的8个数据报：    图13 传输数据文件大小所对应的数据报数量  **(ii)SR协议的文件传输**  在SR协议中，可以采用另一种方式来验证文件传输的正确性：即由服务器端向客户端发送数据，客户端接收到数据后，将其保存在server\_out.txt文件中。在传输完成后，查看生成的server\_out.txt文件，发现内容与server\_in.txt相同，者从另一种角度验证了文件传输的正确性。      图14 server\_in.txt与server\_out.txt文件进行比较  其相关的部分代码如下所示：    图15 相关部分代码   1. **GBN改进为SR协议**   改进GBN协议为SR协议，结果如下：  客户端(client)：          图16 SR协议客户端发送数据文件的运行结果  服务器端(server)：    图17 SR协议服务器端接收数据文件的运行结果  可以看出，在数据报7丢失的情况下，服务器端超时仍未收到ACK7，便将数据报7重发一次。但是与GBN协议不同的是，SR协议中只重发了数据报7，并未重发序号7之后的所有数据报。这也直接证明所设计的SR协议其合理性。 |
| 问题讨论： |
| 对实验过程中的思考问题进行讨论或回答。  我在实验过程中发现检测数据是否传输完成时，对于GBN协议会出现数据乱序或重复。  答：GBN协议当服务器端发送完所有分组后等待ACK时不能停止传输数据，即使继续传输空数据也不要停止传输。因为在客户端后续发送的ACK都丢失时，后续服务器端停止发送分组，客户端不会回复ACK，所以服务器端不会知道客户端已接收的分组序号，默认分组全部丢失，就会在计时器超时全部重传，所以ACK丢失的部分（客户端已经接收过）就会重复，若代码逻辑有问题或鲁棒性不足还有可能出现乱序或死机现象。 |
| 心得体会： |
| 结合实验过程和结果给出实验的体会和收获。  答：   1. 对UDP协议的相关特性有了更为深刻的了解：由于UDP协议属于无连接传输协议，无法保证可靠的数据传输，因此若在上层使用则需上层来实现可靠数据传输部分。与TCP协议不同，UDP在传输数据前不需要三次挥手建立连接，本实验只是逻辑层面的连接。而SR协议的出现便主要解决在使用GBN过程中一定量的资源浪费情况，并且SR、GBN和停-等协议三者本质上的区别主要只是接受和发送窗口大小不同。； 2. 掌握了停等协议、滑动窗口协议的协议内容，学习了全双工与单向传输模式的实现技巧。值得一提的是，SR协议为每一个发送窗口的数据都设置了一个计时器来重传超时的部分，实际的实现过程中每次只需比较发送窗口最低位判断是否超时，若超时则重传，否则表示发送窗口中没有超时数据； 3. 更加熟悉Socket编程的相关方法，设计基于UDP的停-等协议、GBN协议与SR协议的服务器端与客户端，基本实现简单的通信双方的有序可靠的数据传输，理解UDP协议的传输方法和特点，并且了解双向传输的原理，深入理解停-等协议、GBN协议与SR协议的实现过程与技术，理解和掌握可靠数据传输和滑动窗口协议的基本原理。 |
| 实验源代码：  GBN协议的服务器端代码如下：  #include <stdlib.h>  #include <math.h>  #include <time.h>  #include <WinSock2.h>  #include <stdio.h>  #include <fstream>  #include <process.h>  #include <iostream>  using namespace std;  std::ifstream icin;  #pragma warning(disable:4996)  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 12340 //端口号  #define CLIENT\_PORT 12341 //端口号  #define SERVER\_IP "0.0.0.0" //IP 地址  #define CLIENT\_IP "127.0.0.1" //客户端IP  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 = 5;//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  unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter);  /\*---------------------------------------------------发送端相关参数及函数------------------------------------------------------\*/  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 totalSeq;//收到的包的总数  int totalPacket;//需要发送的包总数  int a = 0;//已经发送的包  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: getCurTime  // FullName: getCurTime  // Access: public  // Returns: void  // Qualifier: 获取当前系统时间，结果存入 ptime 中  // Parameter: char \* ptime  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void getCurTime(char\* ptime) {  char buffer[128];  memset(buffer, 0, sizeof(buffer));  time\_t c\_time;  struct tm\* p;  time(&c\_time);  p = localtime(&c\_time);  sprintf\_s(buffer, "%d/%d/%d %d:%d:%d",  p->tm\_year + 1900,  p->tm\_mon + 1,//获得的系统时间慢一个月  p->tm\_mday,  p->tm\_hour,  p->tm\_min,  p->tm\_sec);  strcpy\_s(ptime, sizeof(buffer), buffer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: seqIsAvailable  // FullName: seqIsAvailable  // Access: public  // Returns: bool  // Qualifier: 当前序列号 curSeq 是否可用  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;  //序列号是否在当前发送窗口之内  if (step >= SEND\_WIND\_SIZE) {  return false;  }  if (ack[curSeq]) {  return true;  }  return false;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: timeoutHandler  // FullName: timeoutHandler  // Access: public  // Returns: void  // Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void timeoutHandler() {  printf("Timer out error.\n");  a = curAck;  int index;  for (int i = 0; i < SEND\_WIND\_SIZE; ++i) {  index = (i + curAck) % SEQ\_SIZE;  ack[index] = TRUE;  }  totalSeq = curAck;  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  // Parameter: char c  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1; //序列号减一  printf("Recv a ack of %d\n", index + 1);  if (curAck <= index) {  for (int i = curAck; i <= index; ++i) {  ack[i] = TRUE;  }  curAck = (index + 1) % SEQ\_SIZE;  }  else if (curAck != index + 1) {  //ack 超过了最大值，回到了 curAck 的左边  for (int i = curAck; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  for (int i = 0; i <= index; ++i) {  ack[i] = TRUE;  }  curAck = index + 1;  }  }  /\*---------------------------------------------------接收端相关参数及函数------------------------------------------------------\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* -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(" -time to get current time \n");  printf(" -quit to exit client \n");  printf(" -testgbn [X] [Y] to test the gbn \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;  }  struct ProxyParam {  };  //主函数  int main(int argc, char\* argv[])  {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return -1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  SOCKET sockServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  //设置套接字为非阻塞模式  int iMode = 1; //1：非阻塞，0：阻塞  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  SOCKADDR\_IN addrServer; //服务器地址  //addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);  addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//两者均可  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  err = bind(sockServer, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  if (err) {  err = GetLastError();  printf("Could not bind the port %d for socket.Error code is % d\n", SERVER\_PORT, err);  WSACleanup();  return -1;  }  SOCKADDR\_IN addrClient; //客户端地址  int length = sizeof(SOCKADDR);  char buffer[BUFFER\_LENGTH]; //数据发送接收缓冲区  ZeroMemory(buffer, sizeof(buffer));  //将测试数据读入内存  icin.open("server\_in.txt");  char data[1024 \* 113];  ZeroMemory(data, sizeof(data));  icin.read(data, 1024 \* 113);  icin.close();  totalPacket = ceil(strlen(data) / 1024);  int recvSize;  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  //创建子线程负责传输分组  ProxyParam\* lpProxyParam = new ProxyParam;  HANDLE hThread = (HANDLE)\_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, NULL, 0);  while (true) {  //非阻塞接收，若没有收到数据，返回值为-1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  Sleep(200);  continue;  }  printf("recv from client: %s\n", buffer);  if (strcmp(buffer, "-time") == 0) {  getCurTime(buffer);  }  else if (strcmp(buffer, "-quit") == 0) {  strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!");  }  else if (strcmp(buffer, "-testgbn") == 0) {  //进入 gbn 测试阶段  //首先 server（server 处于 0 状态）向 client 发送 205 状态码（server进入 1 状态）  //server 等待 client 回复 200 状态码，如果收到（server 进入 2 状态），则开始传输文件，否则延时等待直至超时\  //在文件传输阶段，server 发送窗口大小设为  ZeroMemory(buffer, sizeof(buffer));  int recvSize;  int waitCount = 0;  printf("Begain to test GBN protocol,please don't abort the process\n");  //加入了一个握手阶段  //首先服务器向客户端发送一个 205 大小的状态码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", strlen(data), totalPacket);  curSeq = 0;  curAck = 0;  totalSeq = 0;  waitCount = 0;  stage = 2;  }  }  break;  case 2://数据传输阶段  if (seqIsAvailable()) {  //数据发送的过程中应该判断是否传输完成  int z = totalPacket >= SEQ\_SIZE ? SEQ\_SIZE : totalPacket;  if (totalSeq >= totalPacket) {//传输完成，若都收到Ack则传输结束，否则不发送数据等待超时重传  bool finish = true;  for (int i = 0; i < z; i++) {  if (!ack[i]) {  finish = false;  break;  }  }  if (finish) {  printf("\nServer send finish!\n");  buffer[0] = 204;  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(100);  runFlag = false;  break;  }  }  //发送给客户端的序列号从 1 开始  if (a < totalPacket) {  buffer[0] = curSeq + 1;  ack[curSeq] = FALSE;  memcpy(&buffer[1], data + 1024 \* totalSeq, 1024);  printf("send a packet with a seq of %d\n", curSeq + 1);  sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  ++curSeq;  curSeq %= SEQ\_SIZE;  ++totalSeq;  ++a;  }  Sleep(500);  }  //等待 Ack，若没有收到，则返回值为-1，计数器+1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  waitCount++;  //20 次等待 ack 则超时重传  if (waitCount > 20)  {  timeoutHandler();  waitCount = 0;  }  }  else {  //收到 ack  ackHandler(buffer[0]);  waitCount = 0;  }  Sleep(500);  break;  }  }  }  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(500);  }  //关闭套接字，卸载库  CloseHandle(hThread);  closesocket(sockServer);  WSACleanup();  return 0;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: ProxyThread  // FullName: ProxyThread  // Access: public  // Returns: unsigned int \_\_stdcall  // Qualifier: 线程执行函数  // Parameter: LPVOID lpParameter  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter) {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return 1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0);  SOCKADDR\_IN recvClient;  recvClient.sin\_addr.S\_un.S\_addr = inet\_addr(CLIENT\_IP);  recvClient.sin\_family = AF\_INET;  recvClient.sin\_port = htons(CLIENT\_PORT);  //接收缓冲区  char buffer[BUFFER\_LENGTH];  ZeroMemory(buffer, sizeof(buffer));  int len = sizeof(SOCKADDR);  //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;//受到数据大小  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 = 0.2f; //默认包丢失率 0.2  float ackLossRatio = 0.2f; //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));  std::ofstream out;  out.open("server\_out.txt");  while (true) {  gets\_s(buffer);  ret = sscanf\_s(buffer, "%s%f%f", &cmd, sizeof(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\*)&recvClient, sizeof(SOCKADDR));  while (true)  {  //等待 server 回复设置 UDP 为阻塞模式  recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&recvClient, &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\*)&recvClient, sizeof(SOCKADDR));  stage = 1;  recvSeq = 0;  waitSeq = 1;  }  break;  case 1://等待接收数据阶段  if ((unsigned char)buffer[0] == 204) {  printf("\nReceive finished\n");  break;  }  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("%s\n", &buffer[1]);  buffer[0] = seq;  recvSeq = seq;  out.write(&buffer[1], 1024);  }  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;  }  b = lossInLossRatio(ackLossRatio);  if (b) {  printf("The ack of %d loss\n", (unsigned char)buffer[0]);  continue;  }  sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&recvClient, sizeof(SOCKADDR));  printf("send a ack of %d\n", (unsigned char)buffer[0]);  break;  }  Sleep(500);  }  }  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&recvClient, sizeof(SOCKADDR));  ret = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&recvClient, &len);  printf("%s\n", buffer);  if (!strcmp(buffer, "Good bye!")) {  break;  }  printTips();  }  out.close();  //关闭套接字  closesocket(socketClient);  WSACleanup();  return 0;  }  GBN协议的客户端代码如下：  #include <stdlib.h>  #include <math.h>  #include <time.h>  #include <WinSock2.h>  #include <stdio.h>  #include <fstream>  #include <process.h>  #include <iostream>  using namespace std;  #pragma warning(disable:4996)  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 12341 //端口号  #define CLIENT\_PORT 12340 //端口号  #define SERVER\_IP "127.0.0.1" //IP 地址  #define CLIENT\_IP "127.0.0.1" //客户端IP  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 = 5;//发送窗口大小为 5，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  unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter);  /\*---------------------------------------------------发送端相关参数及函数------------------------------------------------------\*/  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 totalSeq;//收到的包的总数  int totalPacket;//需要发送的包总数  int a = 0;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: getCurTime  // FullName: getCurTime  // Access: public  // Returns: void  // Qualifier: 获取当前系统时间，结果存入 ptime 中  // Parameter: char \* ptime  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void getCurTime(char\* ptime) {  char buffer[128];  memset(buffer, 0, sizeof(buffer));  time\_t c\_time;  struct tm\* p;  time(&c\_time);  p = localtime(&c\_time);  sprintf\_s(buffer, "%d/%d/%d %d:%d:%d",  p->tm\_year + 1900,  p->tm\_mon + 1,  p->tm\_mday,  p->tm\_hour,  p->tm\_min,  p->tm\_sec);  strcpy\_s(ptime, sizeof(buffer), buffer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: seqIsAvailable  // FullName: seqIsAvailable  // Access: public  // Returns: bool  // Qualifier: 当前序列号 curSeq 是否可用  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;  //序列号是否在当前发送窗口之内  if (step >= SEND\_WIND\_SIZE) {  return false;  }  if (ack[curSeq]) {  return true;  }  return false;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: timeoutHandler  // FullName: timeoutHandler  // Access: public  // Returns: void  // Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void timeoutHandler() {  printf("Timer out error.\n");  a = curAck;  int index;  for (int i = 0; i < SEND\_WIND\_SIZE; ++i) {  index = (i + curAck) % SEQ\_SIZE;  ack[index] = TRUE;  }  totalSeq = curAck;  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  // Parameter: char c  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1; //序列号减一  printf("Recv a ack of %d\n", index + 1);  if (curAck <= index) {  for (int i = curAck; i <= index; ++i) {  ack[i] = TRUE;  }  curAck = (index + 1) % SEQ\_SIZE;  }  else if (curAck != index + 1) {  //ack 超过了最大值，回到了 curAck 的左边  for (int i = curAck; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  for (int i = 0; i <= index; ++i) {  ack[i] = TRUE;  }  curAck = index + 1;  }  }  /\*---------------------------------------------------接收端相关参数及函数------------------------------------------------------\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* -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(" -time to get current time \n");  printf(" -quit to exit client \n");  printf(" -testgbn [X] [Y] to test the gbn \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;  }  struct ProxyParam {  };  //主函数  int main(int argc, char\* argv[])  {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return -1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  SOCKET sockServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  //设置套接字为非阻塞模式  int iMode = 1; //1：非阻塞，0：阻塞  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  SOCKADDR\_IN addrServer; //服务器地址  //addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);  addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//两者均可  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  err = bind(sockServer, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  if (err) {  err = GetLastError();  printf("Could not bind the port %d for socket.Error code is % d\n", SERVER\_PORT, err);  WSACleanup();  return -1;  }  SOCKADDR\_IN addrClient; //客户端地址  int length = sizeof(SOCKADDR);  char buffer[BUFFER\_LENGTH]; //数据发送接收缓冲区  ZeroMemory(buffer, sizeof(buffer));  //将测试数据读入内存  std::ifstream icin;  icin.open("client\_in.txt");  char data[1024 \* 113];  ZeroMemory(data, sizeof(data));  icin.read(data, 1024 \* 113);  icin.close();  totalPacket = ceil(strlen(data) / 1024);  int recvSize;  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  //创建子线程负责传输分组  ProxyParam\* lpProxyParam = new ProxyParam;  HANDLE hThread = (HANDLE)\_beginthreadex(NULL, 0, &ProxyThread, (LPVOID)lpProxyParam, NULL, 0);  while (true) {  //非阻塞接收，若没有收到数据，返回值为-1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  Sleep(200);  continue;  }  printf("recv from client: %s\n", buffer);  if (strcmp(buffer, "-time") == 0) {  getCurTime(buffer);  }  else if (strcmp(buffer, "-quit") == 0) {  strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!");  }  else if (strcmp(buffer, "-testgbn") == 0) {  //进入 gbn 测试阶段  //首先 server（server 处于 0 状态）向 client 发送 205 状态码（server进入 1 状态）  //server 等待 client 回复 200 状态码，如果收到（server 进入 2 状态），则开始传输文件，否则延时等待直至超时\  //在文件传输阶段，server 发送窗口大小设为  ZeroMemory(buffer, sizeof(buffer));  int recvSize;  int waitCount = 0;  printf("Begain to test GBN protocol,please don't abort the process\n");  //加入了一个握手阶段  //首先服务器向客户端发送一个 205 大小的状态码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", strlen(data), totalPacket);  curSeq = 0;  curAck = 0;  totalSeq = 0;  waitCount = 0;  stage = 2;  }  }  break;  case 2://数据传输阶段  if (seqIsAvailable()) {  int z = totalPacket >= SEQ\_SIZE ? SEQ\_SIZE : totalPacket;  //数据发送的过程中应该判断是否传输完成  if (totalSeq >= totalPacket) {//传输完成，若都收到Ack则传输结束，否则不发送数据等待超时重传  bool finish = true;  for (int i = 0; i < z; i++) {  if (!ack[i]) {  finish = false;  break;  }  }  if (finish) {  printf("\nClient send finish!\n");  buffer[0] = 204;  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(100);  runFlag = false;  break;  }  }  //发送给客户端的序列号从 1 开始  if (a < totalPacket) {  buffer[0] = curSeq + 1;  ack[curSeq] = FALSE;  memcpy(&buffer[1], data + 1024 \* totalSeq, 1024);  printf("send a packet with a seq of %d\n", curSeq + 1);  sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  ++curSeq;  curSeq %= SEQ\_SIZE;  ++totalSeq;  ++a;  }  Sleep(500);  }  //等待 Ack，若没有收到，则返回值为-1，计数器+1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  waitCount++;  //20 次等待 ack 则超时重传  if (waitCount > 20)  {  timeoutHandler();  waitCount = 0;  }  }  else {  //收到 ack  ackHandler(buffer[0]);  waitCount = 0;  }  Sleep(500);  break;  }  }  }  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(500);  }  //关闭套接字，卸载库  CloseHandle(hThread);  closesocket(sockServer);  WSACleanup();  return 0;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: ProxyThread  // FullName: ProxyThread  // Access: public  // Returns: unsigned int \_\_stdcall  // Qualifier: 线程执行函数  // Parameter: LPVOID lpParameter  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  unsigned int \_\_stdcall ProxyThread(LPVOID lpParameter) {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return 1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0);  SOCKADDR\_IN recvClient;  recvClient.sin\_addr.S\_un.S\_addr = inet\_addr(CLIENT\_IP);  recvClient.sin\_family = AF\_INET;  recvClient.sin\_port = htons(CLIENT\_PORT);  //接收缓冲区  char buffer[BUFFER\_LENGTH];  ZeroMemory(buffer, sizeof(buffer));  int len = sizeof(SOCKADDR);  //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;//受到数据大小  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 = 0.2f; //默认包丢失率 0.2  float ackLossRatio = 0.2f; //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));  std::ofstream out;  out.open("client\_out.txt");  while (true) {  gets\_s(buffer);  ret = sscanf\_s(buffer, "%s%f%f", &cmd, sizeof(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\*)&recvClient, sizeof(SOCKADDR));  while (true)  {  //等待 server 回复设置 UDP 为阻塞模式  recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&recvClient, &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\*)&recvClient, sizeof(SOCKADDR));  stage = 1;  recvSeq = 0;  waitSeq = 1;  }  break;  case 1://等待接收数据阶段  if ((unsigned char)buffer[0] == 204) {  printf("\nReceive finished\n");  break;  }  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("%s\n", &buffer[1]);  buffer[0] = seq;  recvSeq = seq;  out.write(&buffer[1], 1024);  }  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;  }  b = lossInLossRatio(ackLossRatio);  if (b) {  printf("The ack of %d loss\n", (unsigned char)buffer[0]);  continue;  }  sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&recvClient, sizeof(SOCKADDR));  printf("send a ack of %d\n", (unsigned char)buffer[0]);  break;  }  Sleep(500);  }  }  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&recvClient, sizeof(SOCKADDR));  ret = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&recvClient, &len);  printf("%s\n", buffer);  if (!strcmp(buffer, "Good bye!")) {  break;  }  printTips();  }  out.close();  //关闭套接字  closesocket(socketClient);  WSACleanup();  return 0;  }  SR协议的服务器端代码如下：  #include <stdlib.h>  #include <time.h>  #include <WinSock2.h>  #include <fstream>  #include <iostream>  using namespace std;  #pragma comment(lib,"ws2\_32.lib")  #pragma warning(disable:4996)  #define SERVER\_PORT 12340 //端口号  #define SERVER\_IP "0.0.0.0" //IP 地址  #define DATA\_SIZE 1024  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 = 5;//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 counter[SEQ\_SIZE];//计时器，当为负数时表示未启动，为正数时表示启动；其他同GBN  int curSeq;//当前数据包的 seq  int curAck;//当前等待确认的 ack  int totalSeq;//收到的包的总数  int totalPacket;//需要发送的包总数  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: getCurTime  // FullName: getCurTime  // Access: public  // Returns: void  // Qualifier: 获取当前系统时间，结果存入 ptime 中  // Parameter: char \* ptime  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void getCurTime(char\* ptime) {  char buffer[128];  memset(buffer, 0, sizeof(buffer));  time\_t c\_time;  struct tm\* p;  time(&c\_time);  p = localtime(&c\_time);  sprintf\_s(buffer, "%d/%d/%d %d:%d:%d",  p->tm\_year + 1900,  p->tm\_mon,  p->tm\_mday,  p->tm\_hour,  p->tm\_min,  p->tm\_sec);  strcpy\_s(ptime, sizeof(buffer), buffer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: seqIsAvailable  // FullName: seqIsAvailable  // Access: public  // Returns: bool  // Qualifier: 当前序列号 curSeq 是否可用  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;  //序列号是否在当前发送窗口之内  if (step >= SEND\_WIND\_SIZE) {  return false;  }  if (ack[curSeq]) {  return true;  }  return false;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: ackHandler  // FullName: ackHandler  // Access: public  // Returns: void  // Qualifier: 收到 ack，累积确认，取数据帧的第一个字节  //由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一了，此处需要减一还原  // Parameter: char c  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1; //序列号减一  printf("Recv a ack of %d\n", index + 1);  if (curAck != index) {//分组失序，暂时缓存  ack[index] = TRUE;  }  else {  //一次分组到达，窗口向前移动（可能不止一次移动）  ack[index] = TRUE;  for (int i = index; i < index + SEQ\_SIZE; i++) {  i %= SEQ\_SIZE;  if (ack[i]) {  counter[i] = -1;//计时器关闭  }  else {  curAck = i + 1;//修改curAck  break;  }  }  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: click  // FullName: click  // Access: public  // Returns: void  // Qualifier: 所有开始的计时器加一  // Parameter: void  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void click() {  for (int i = 0; i < SEQ\_SIZE; i++) {  if (counter[i] >= 0) {  counter[i] += 1;  }  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: checkTimeout  // FullName: checkTimeout  // Access: public  // Returns: BOOL  // Qualifier: 检查所有计时器是否有超时的，若有返回超时的计时器的下标；否则返回-1  // 若有同时出现两个计时器同时超时，则逐个处理  // Parameter: void  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  int checkTimeout() {  bool finish;  for (int i = 0; i < SEQ\_SIZE; i++) {  if (counter[i] >= 20) {  return i;  }  }  return -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;  }  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\_in.txt");  char data[DATA\_SIZE \* 113];//需要发送的数据  ZeroMemory(data, sizeof(data));  icin.read(data, DATA\_SIZE \* 113);  icin.close();  totalPacket = strlen(data) / DATA\_SIZE;  int recvSize;  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  counter[i] = -1;  }  char cache[SEND\_WIND\_SIZE + 1][DATA\_SIZE + 1];//缓存，暂时保存发送但未受到ack的分组  while (true) {  //非阻塞接收，若没有收到数据，返回值为-1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  Sleep(200);  continue;  }  printf("recv from client: %s\n", buffer);  if (strcmp(buffer, "-time") == 0) {  getCurTime(buffer);  }  else if (strcmp(buffer, "-quit") == 0) {  strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!");  }  else if (strcmp(buffer, "-testsr") == 0) {  //进入 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 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 大小的状态码表示服务器准备好了，可以发送数据  //客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准备好了，可以接收数据了  //服务器收到 200 状态码之后，就开始使用 SR 发送数据了  printf("Shake hands stage\n");  int stage = 0;  bool runFlag = true;  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", strlen(data), totalPacket);  curSeq = 0;  curAck = 0;  totalSeq = 0;  //waitCount = 0;  stage = 2;  }  }  break;  case 2://数据传输阶段  if (seqIsAvailable()) {  if (totalSeq <= totalPacket) {  //发送给客户端的序列号从 1 开始  buffer[0] = curSeq + 1;  ack[curSeq] = FALSE;  memcpy(&buffer[1], data + DATA\_SIZE \* totalSeq, DATA\_SIZE);  memcpy(cache[curSeq], data + DATA\_SIZE \* totalSeq, DATA\_SIZE);//缓存分组  printf("send a packet with a seq of %d\n", curSeq + 1);  sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  counter[curSeq] = 0;//计时器开启  ++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) {  click();  //20 次等待 ack 则超时重传  if (checkTimeout() != -1) {  int index = checkTimeout();  printf("Seq %d time out.\n", index + 1);  buffer[0] = index + 1;  memcpy(&buffer[1], cache[index], DATA\_SIZE);  printf("Re : send a packet with a seq of %d\n", index + 1);  sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  counter[index] = 0;//重置计时器  }  }  else {  //收到 ack  ackHandler(buffer[0]);  counter[buffer[0] - 1] = -1;//计时器关闭  //数判断是否传输完成  if (totalSeq >= totalPacket) {//传输完成，若都收到Ack则传输结束，否则不发送数据等待超时重传  bool finish = true;  for (int i = 0; i < SEQ\_SIZE; i++) {  if (!ack[i]) {  finish = false;  break;  }  }  if (finish) {  printf("\nServer send finish!\n");  buffer[0] = 204;  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(100);  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;  }  SR协议的客户端代码如下：  #include <stdlib.h>  #include <WinSock2.h>  #include <time.h>  #include <stdio.h>  #include <fstream>  #include <iostream>  using namespace std;  #pragma warning(disable:4996)  #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 = 5;//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）  const int SEQ\_SIZE = 20;//接收端序列号个数，为 1~20  #define DATA\_SIZE 1024  bool ack[SEQ\_SIZE];  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* -time 从服务器端获取当前时间  -quit 退出客户端  -testsr [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(" -time to get current time \n");  printf(" -quit to exit client \n");  printf(" -testsr [X] [Y] to test the sr \n");  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: lossInLossRatio  // FullName: lossInLossRatio  // Access: public  // Returns: BOOL  // Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回TRUE，否则返回 FALSE  // Parameter: float lossRatio [0,1]  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  BOOL lossInLossRatio(float lossRatio) {  int lossBound = (int)(lossRatio \* 100);  int r = rand() % 101;  if (r <= lossBound) {  return TRUE;  }  return FALSE;  }  int main(int argc, char\* argv[])  {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return 1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0);  SOCKADDR\_IN addrServer;  addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  //接收缓冲区  char buffer[BUFFER\_LENGTH];  ZeroMemory(buffer, sizeof(buffer));  int len = sizeof(SOCKADDR);  //为了测试与服务器的连接，可以使用 -time 命令从服务器端获得当前时间  //使用 -testsr [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;//受到数据大小  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 = 0.2f; //默认包丢失率 0.2  float ackLossRatio = 0.2f; //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));  std::ofstream out;  out.open("client\_out.txt");  char cache[SEND\_WIND\_SIZE][DATA\_SIZE];//缓存，暂时保存失序但未确认的分组  while (true) {  gets\_s(buffer);  //printf("buffer:%s\n", buffer);  ret = sscanf\_s(buffer, "%s%f%f", &cmd, sizeof(cmd), &packetLossRatio, &ackLossRatio);  printf("buffer:%s\n", cmd);  printf("packet:%f2\n", packetLossRatio);  printf("ack:%2f\n", ackLossRatio);  if (!strcmp(cmd, "-testsr")) {  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, "-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);  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;  for (int i = 0; i < SEQ\_SIZE; i++)  {  ack[i] = false;  }  }  break;  case 1://等待接收数据阶段  /\*for (int i = 0; i < SEND\_WIND\_SIZE; i++) {  cout << i << ":" << ack[i] << endl;  }\*/  if ((unsigned char)buffer[0] == 204) {  printf("\nReceive finished\n");  break;  }  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  //cout << "waitSeq：" << ":" << waitSeq << endl;  if (waitSeq == seq) {  waitSeq++;  if (waitSeq == 21) {  waitSeq = 1;  }  //输出数据  printf\_s("%s\n", &buffer[1]);  //当前接收分组直接写入文件  out.write(&buffer[1], DATA\_SIZE);  //查看是否有失序分组需要写入文件  for (int i = waitSeq - 1; i < waitSeq - 1 + SEND\_WIND\_SIZE; i++) {  i %= SEQ\_SIZE;  if (ack[i]) {  ack[i] = false;  //cout <<"修改："<< i << ":" << ack[i] << endl;  waitSeq++;  if (waitSeq == 21) {  waitSeq = 1;  }  //从缓存写入  out.write(cache[i], DATA\_SIZE);  }  else {  break;  }  }  buffer[0] = seq;//回复下一个需要分组的序列号  recvSeq = seq;//当前已确认的序列号为下一个需要的序列号前一个  buffer[1] = '\0';  }  else if (seq > waitSeq) {//分组失序到达  memcpy(cache[seq - 1], &buffer[1], DATA\_SIZE);//缓存收到的数据，不修改下一个需要的分组序列号  char copy[DATA\_SIZE];  memcpy(copy, cache[seq - 1], DATA\_SIZE);  printf("缓存：%s\n", copy);  buffer[0] = seq;  buffer[1] = '\0';  ack[seq - 1] = true;  }  else {//ack丢失重传，直接返回ack，不缓存  buffer[0] = seq;  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);  }  }  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;  } |