哈爾濱工業大學
Harbin Institute of Technology

计算机网络

课程实验报告

实验名称	HTTP 代理服务器的设计与实现				
姓名	王欣悦	院系	网络空间安全学院		
班级	2203201	学号	2022113365		
任课教师	詹东阳	指导教师	詹东阳		
实验地点	格物 207	实验时间	4.10		
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					

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

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)停等协议

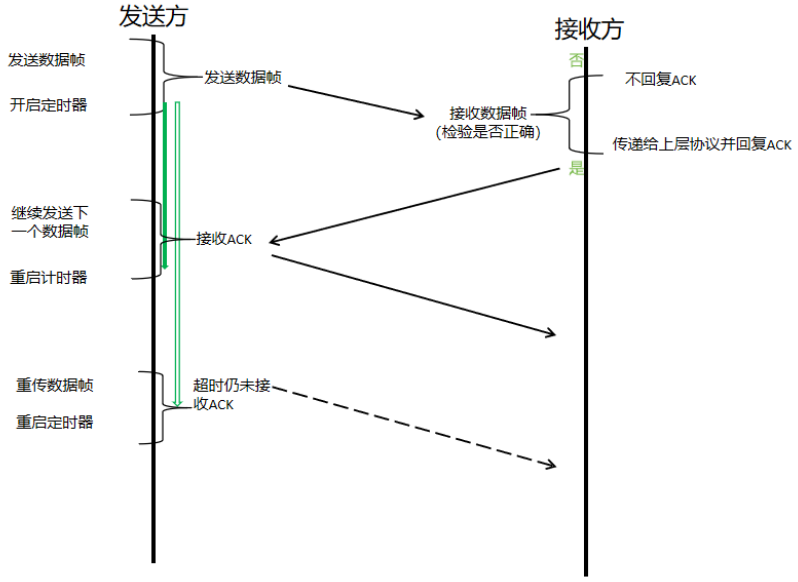
停等协议是一种用于确保数据可靠传输的协议，它主要用于数据传输过程中的流量控制。在停等协议中，发送方在每次发送数据帧后必须等待接收方的确认信息，才能继续发送下一帧数据。如果接收方没有收到数据或数据出错，发送方将重发当前的数据帧。

停等协议的主要步骤包括：

发送方发送数据->接收方接收数据并进行检验，如果正确则回复ACK->发送方接收ACK，继续发送下一帧数据

停等协议中发送方与接收方的窗口大小均为1，发送方将为发送窗口设置一个计时器。当发送过程中，发送方首先为发送的数据帧开启计时器，数据帧丢失或者接收方检验到错误导致接收方无法发送ACK，或者接收方检验正确但发送的ACK数据帧丢包，这两种情况下发送方均无法接受到ACK，当计时器超时时发送方仍未接受到ACK，则重新发送数据帧，否则继续发送下一个数据帧。

停等协议的交互过程如下图所示：



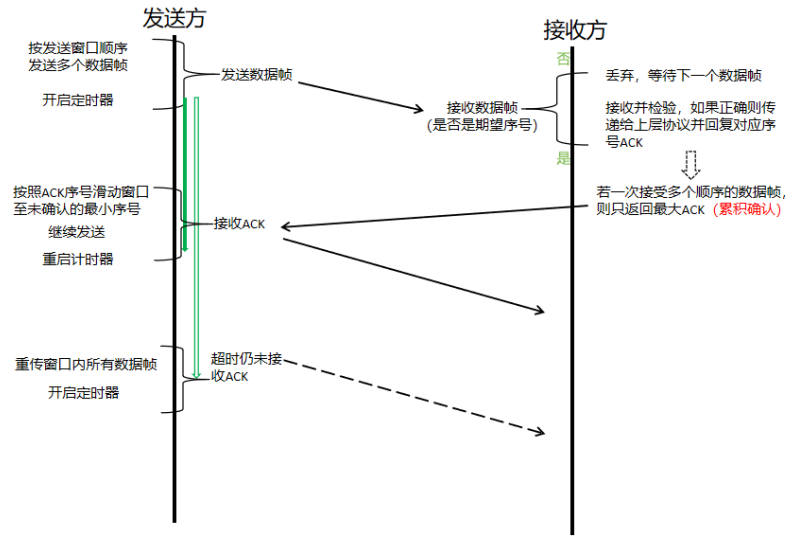
(2)GBN协议

与停等协议类似，GBN协议也用于在不可靠的信道上可靠地传输数据，是一种数据链路层的协议。与停等协议不同，GBN协议基于滑动窗口允许发送方连续发送多个数据帧，而接收方则按序接收，提高了数据发送的效率。

GBN协议的主要步骤包括：

发送方按发送窗口顺序发送多个数据帧并开启定时器->接收方接收数据帧，如果是期望序号的包则接收并检验，如果正确则传递给上层协议并回复对应序号ACK，若一次接受了多个顺序的数据帧，则返回的ACK为正确接收的最大序号（累积确认）。若接受的数据包序号并非期望，则丢弃，等待下一个数据帧，且不返回ACK->发送方接收ACK，并按照ACK中的序号将窗口向前滑动至尚未确认的最小序号并重启计时器，若直到计时器超时仍未接受到ACK则重传窗口内所有数据。

GBN协议的交互过程如下图所示：

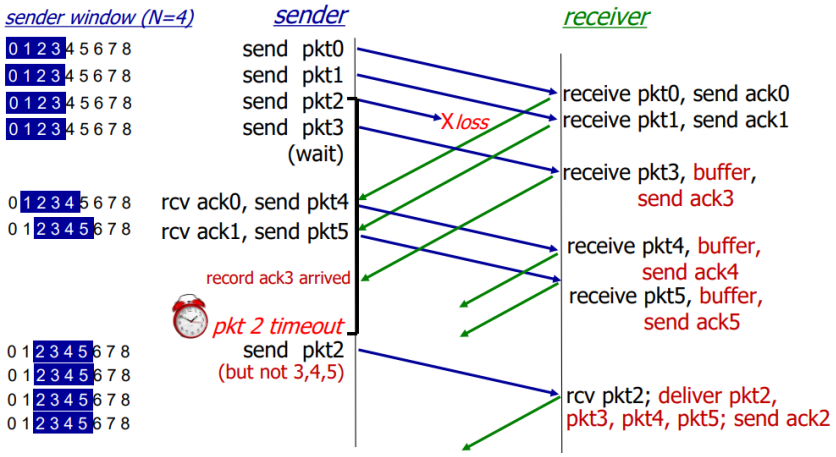


(3)SR协议

AR协议同样基于滑动窗口，在GBN的基础上，SR协议的发送窗口与接受窗口分别为m与n，并且为发送的数据帧单独设置计时器，而不是整个窗口共用一个计时器。因此，对于超时未

收到ACK的数据帧，只需要重传这一个数据帧即可，而不是重传整个窗口中的数据帧，已经正确到达接收端的乱序数据帧也能被有效的利用缓存到接收方，而不是直接丢弃重传，增大了链路的有效利用率，提升了数据传输的效率。

在GBN协议实现的基础上，需要增加对发送与接收双方窗口的维护，并且设置多个计时器。



(4)丢包实现

设置丢包率后，通过lossInLossRatio函数来实现对数据帧丢失的模拟。

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

首先生成一个100以内的随机数，如果该数小于lossBound，即丢包率的一百倍，则将该包设置为丢失。对于任意一个数据帧，执行该函数的过程中生成随机数的分布是相同的，即被设置为丢包的概率也相同，均为丢包率。因此可以实现对于不可靠链路中数据帧的丢失。

(5)计时器与超时重传的实现

本实验中，停等协议与GBN协议的计时器都是靠计数器实现的，与样例代码中的实现相同，通过每次等待sleep，然后累加sleep的时间，以次数来记录是否超时，如果超时则调用超时重传函数timeoutHandler来实现对窗口内全部数据帧的重传：

```
void timeoutHandler() {
 printf("Timer out error.\n");
 int index;
 for (int i = 0; i < SEND_WIND_SIZE; ++i) { //循环遍历发送窗口内的数据包
  index = (i + curAck) % SEQ_SIZE;  //计算序列号，从最后一个被确认的包开始
  ack[index] = TRUE;
 }
```

```
8.  totalSeq -= SEND_WIND_SIZE;
9.  curSeq = curAck;    //将当前序列号设置为最后一个被确认的包的序列，其后所有包都需要被重传
10. }
```

而SR协议的实现中，计时器是通过记录发送时间与当前时间now之间的差值来实现的：

```
1. if (now - sendWindow[0].start >= 1000L) {
2.  sendWindow[0].start = now;
3.  sendto(socketServer, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0,(SOCKADDR*)&addrClient, sizeof(SOCKADDR));
4. }
```

由于SR协议为发送的数据帧设置了单独的计时器，发送时启动计时器：

```
1. sendWindow[0].start = start - 1000L;
```

Start是sendwindow中类型为clock_t的数据，记录了发送时间。若当前时间与发送时间的差值超过预设的数据，则判定为超时，只需要重传该数据帧。

(6)双向传输的实现

在实现了单向传输的基础上，发送方也可以作为接收方接收来自另一方的数据包。在此基础上，发送方和接收方需要同时维护sendWin和recvWin，因为他们都有可能作为发送方与接收方。在此基础上，server首先需要判断来自client的请求，决定server是成为发送方还是接收方，随后控制自动机跳转到对应的状态中。client也需要在发送请求后跳转到对应的状态中。两者开始通信后扮演不同的角色进行数据的收发与确认，因此在server和client中都需要实现数据的发送功能与接收功能。

(7)UDP编程的主要特点

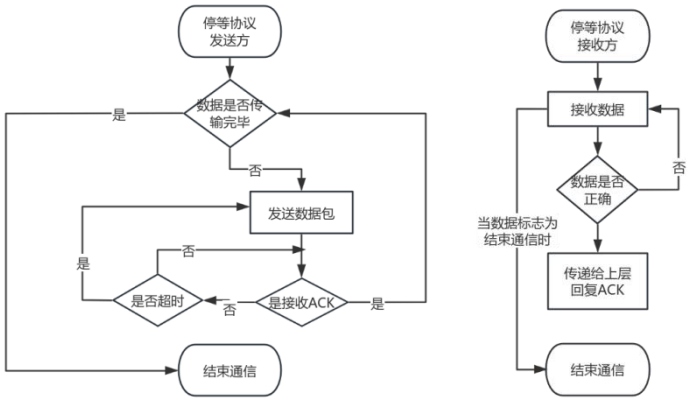
1）UDP是无连接的通信协议，通信双方在发送和接收数据前不需要建立连接。

2）UDP不提供数据包的可靠传输，因此数据包可能会在传输过程中丢失、重复或者无序到达。本实验中的丢包是通过控制丢包率实现的，但是除了丢包，实验网络环境也会造成数据包的乱序到达。UDP不提供重传机制，因此实验中需要手动处理重传的控制。

3）UDP的头部开销较小，试验过程中不需要维护状态信息。

(8)停等协议的实现

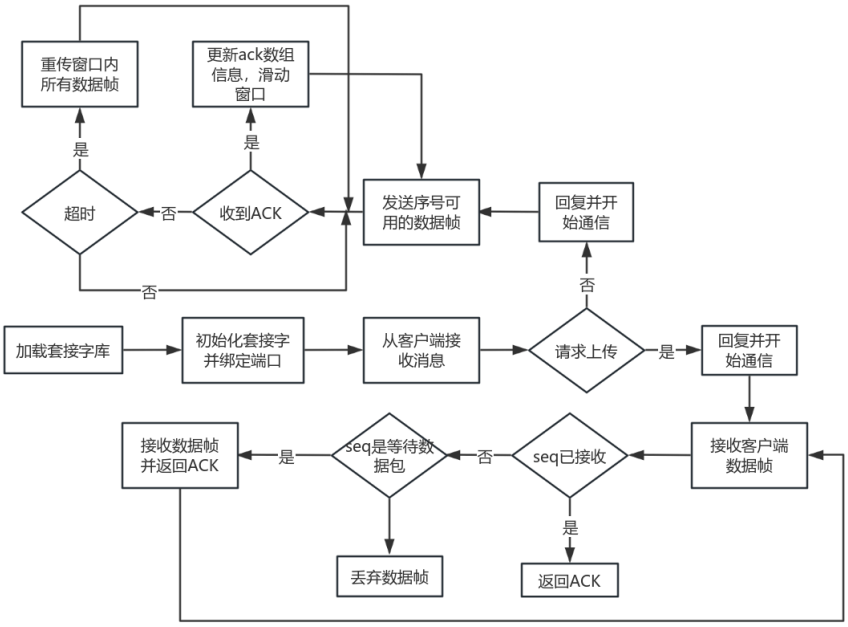
停等协议中，双方的流程图如下所示：

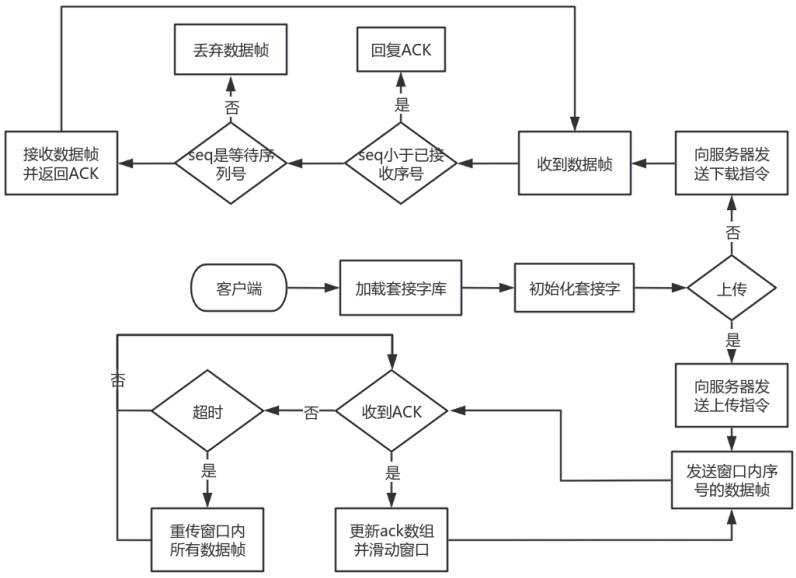


由于停等协议的实现与GBN相同，只是少了滑动窗口与窗口状态的维护，相当于发送窗口为1的GBN协议，对于代码的分析通GBN协议，源代码也不单独展示。

(9)GBN的实现

GBN协议中，双方的流程图如下所示：





以报告中的示例代码为基础，利用C语言实现GBN协议。

GBN协议数据分组格式如下：

数据分组格式	含义
curSeq	当前数据包的 seq
curAck	当前等待确认的 ack
ack[SEQ_SIZE]	收到 ack 情况，对应 0~19 的 ack
totalSeq	收到的包的总数

curSeq表示当前数据帧的编号，但是注意由于编号是1-20，而数组下标是0-19，因此发送数据帧时应当标记当前数据帧的序号为curSeq+1。发送数据帧的同时将ack[curSeq]赋为false，证明该数据帧发送出去仍未接受到ACK。当收到ACK数据帧时，读取buffer头部的序号信息，并将窗口内对应curSeq前所有数据帧的ack赋为true，表示累积确认已接收到ACK。

curAck标记了当前最近一个收到ACK的数据帧序号，用于在超时重传时计算整个窗口内数据帧的序号并进行重传。收到ACK数据帧后，修改curAck。

ack数组记录了发送窗口编号为0-19的20个分组接收ACK的情况。初始设置发送窗口与接收窗口窗口大小均为10，发送的数据帧按SEQ_SIZE=20顺序进行编号，不会产生错误。当编号为

totalSeq表示收到的包的总数，但是在代码实现中，发送完数据帧后首先将totalSeq++，如果发生超时重传，则totalSeq -= SEND_WIND_SIZE;表示一个窗口内的数据帧都未能成功发送，需要重传。

GBN协议确认分组格式如下：

确认分组格式	含义
seq	包的序列号
recvSeq	已确认的序列号
waitSeq	等待的期望序列号

Seq是接收方收到数据帧时，从buffer[0]中读出来的序列号，但当buffer[0]中的数字是某些特殊数字时，则表示控制信息而非数据传输。

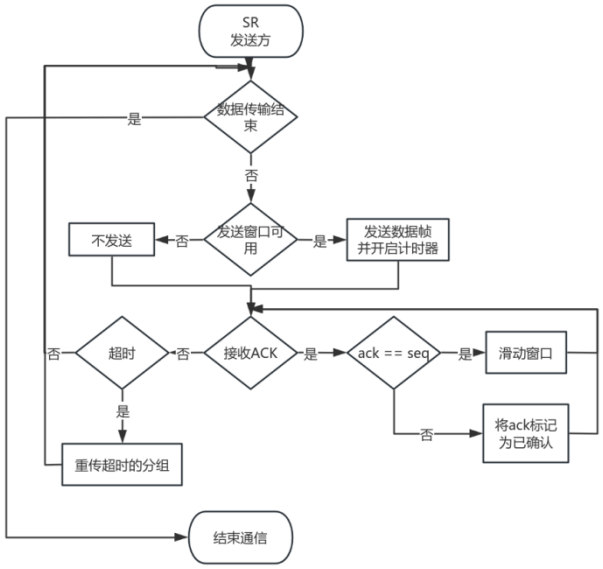
recvSeq表示当前接受到的序列号，并且将这个序列号插入buffer头部作为ACK的序列号返回发送方。

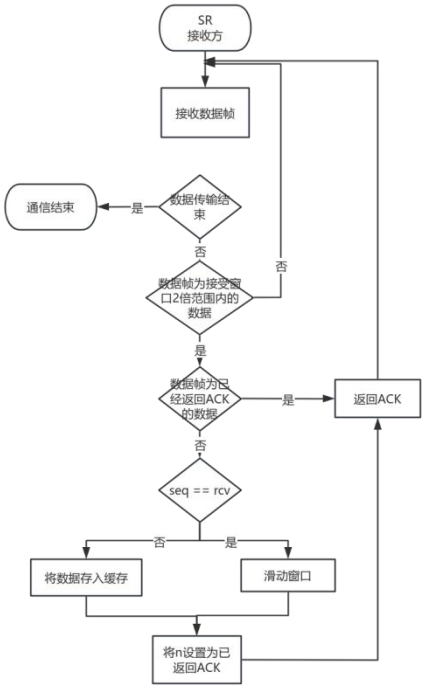
waitSeq表示当前等待的序列号，由于GBN协议的接受窗口大小只有1，因此对于乱序到达的seq != waitseq的数据帧直接丢弃，只有当seq == waitseq时才接收数据帧并将其传递给上层协议，同时更新ACK。当收到的seq小于recvSeq时，说明ACK丢包，此时不需要接收数据帧内容，只需要返回ACK即可。

GBN协议实现过程中的主要函数：

函数	作用
void getCurTime(char* ptime)	获取当前系统时间，结果存入 ptime 中
bool seqIsAvailable()	判断当前序列号curSeq是否可用
void timeoutHandler()	超时重传处理函数，将滑动窗口内数据帧全部重传
void ackHandler(char c)	收到 ack累积确认并进行处理

(10)SR的实现





实现 SR双向传输协议时，利用了与示例代码不同的结构，以服务器端为例：

```
struct recv {
    bool used;
    char buffer[BUFFER_SIZE];
    recv() {
        used = false;
        ZeroMemory(buffer, sizeof(buffer));
    }
}recvWindow[SEQ_SIZE];

struct send {
    clock_t start;//每个数据包有一个计时器
    char buffer[BUFFER_SIZE];
    send() {
        start = 0;
        ZeroMemory(buffer, sizeof(buffer));
    }
}sendWindow[SEQ_SIZE];
```

服务器端维护了接受窗口和发送窗口两个结构，既可以充当发送者也可以充当接收者。对于接收窗口，需要记录该数据帧是否到达，对于发送窗口，利用start做计时器，来记录发送时间并记录是否超时，如果超时则需重传并重启计时器。客户端也维护了相同的两个窗口。

SR协议数据分组格式：

数据分组格式	含义
sendWindow[SEQ_SIZE]	包含buffer[BUFFER_SIZE]和计时器start
ack	接收的ACK数据帧中的序号
seq	窗口最左端等待接收的数据帧序号

Sendwindow是一个结构体数组，每个元素中保存了该元素序号对应数据帧的缓冲区内容以及计时器。通过当前时间与计时器中保存时间的差值来判断是否超时，如果超时则向接收方重传缓冲区中的内容。

ack是发送方接受到的ACK数据帧中所保存的序号，收到后将对应序号的sendwindow元素中计时器重置为一个负值，表示该数据帧已被接收不会再超时。当ack == seq时，利用

MoveSendWindow函数将窗口向前滑动。

SR协议确认分组格式：

确认分组格式	含义
recvWindow	包含buffer[BUFFER_SIZE]和表示是否已经接收的标志used
temp	计算步长
seq	接受到数据帧的序号
ack	存储仍未接受到的最小序号

Recvwindow是一个结构体数组，每个元素保存了该元素序号对应数据帧的缓冲区内容以及标志是否已经被接收方接收的used。

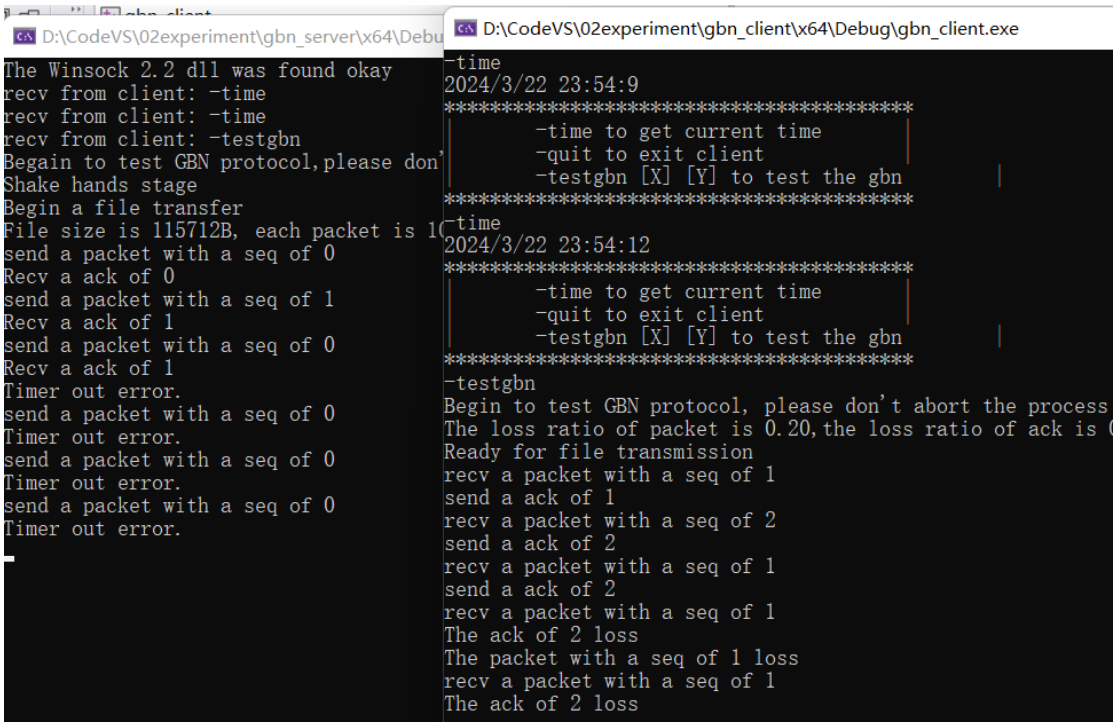
当接收方接收到数据帧时获得序号seq，首先计算步长temp，来判断该数据是否需要被缓存或者回复ACK。当ack == seq，即接受到的数据帧正好是尚未接受的数据帧的最小序号，则接受该数据帧并返回ACK，同时需要调用Deliver函数将一整块连续的数据包写入文件，并向前滑动接收窗口。

SR协议实现过程中的主要函数：

int Deliver(char* file, int ack)	将一整块连续的数据帧写入文件并向前滑动接收窗口
int Send(ifstream& infile, int seq, SOCKET socket, SOCKADDR* addr);	通过本方套接字向目的地址发送文件中读取的编号为seq的数据帧
int MoveSendWindow(int seq);	当发送方发送窗口最左端的数据发送完毕后滑动发送窗口
int Read(ifstream& infile, char* buffer)	从文件中读取需要发送的数据

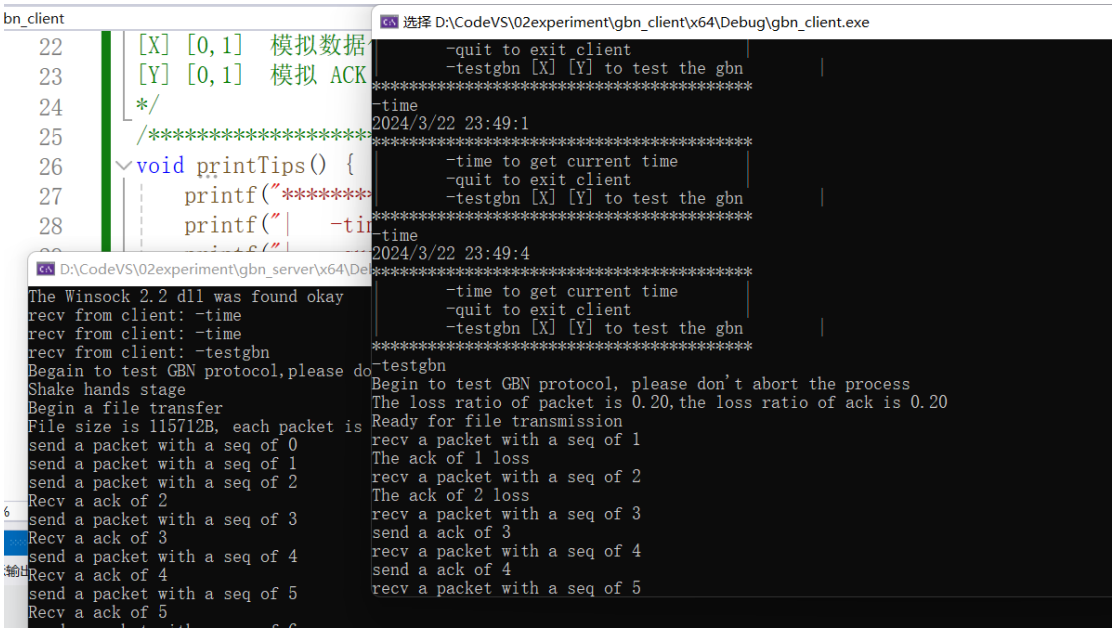
实验结果：

（1）停等协议



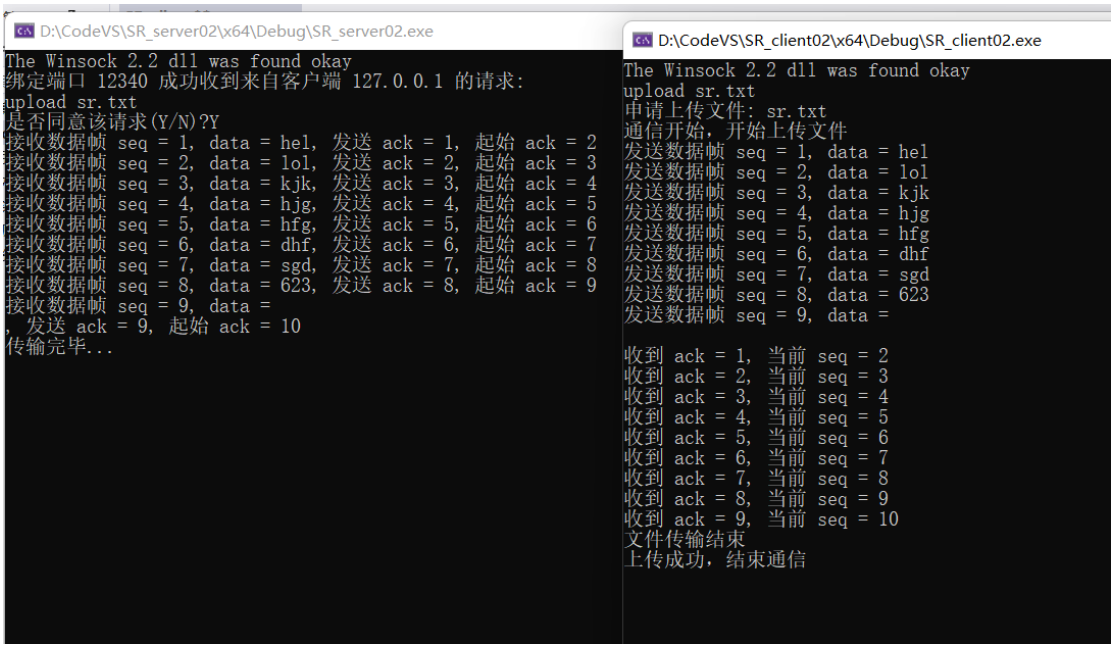
将GBN协议中的序号数改为2，发送窗口修改为1，即可实现基于停等协议的通信（两方的序号由于数组下标错位）。由结果可以看出，停等协议必须接受到本次发送的数据帧的ACK后才能继续发送下一个数据帧，否则发生超时重传，重新发送本数据帧直到收到正确的ACK。

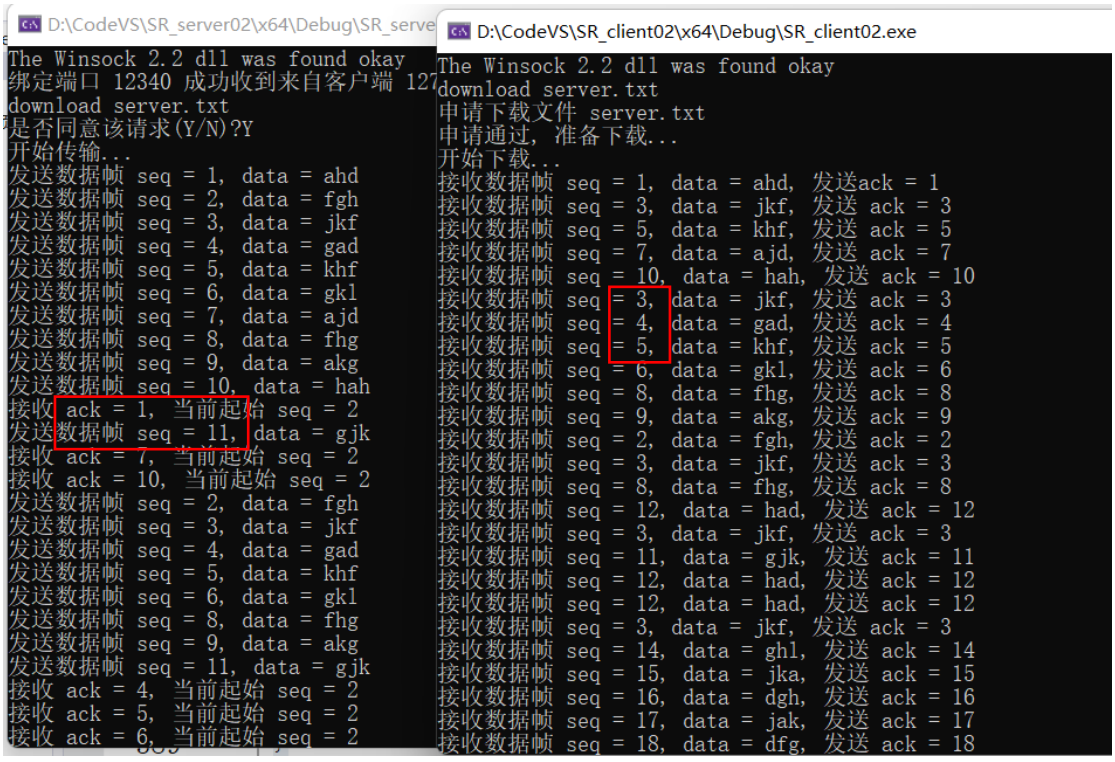
（2）GBN协议



首先通过-time来测试服务器与客户端之间的连接是否连通，随后进行gbn测试，发送文件中的内容。由于GBN是累积确认，虽然ack=1与ack=2均丢包，但是ack=3正确的发送后，发送方接收到了ack=2（序号与数组下标错位，实际上是3），并知道seq=1-3均被正确的接收，因此直接继续发送数据帧。

（3）SR协议





通过分析可知，发送方最开始发送了1-10数据帧，接收方收到了其中的5个，其余五个都丢包，将其缓存后回复ACK。发送方接受到ACK=1后向前滑动窗口，发送11数据帧。随后收到了ACK=7与ACK=11，但是由于seq=2尚未收到ACK，因此不能滑动窗口。随后尚未收到ACK的数据帧都超时，发送方只重传这些超时的数据帧，接收方也收到了重传的数据帧。当发送方接受到ACK＝2时立刻向前滑动窗口，此时seq=3也已经收到ACK，因此可以连续发送11、12数据帧，以此类推，

两者进行数据的传输。

问题讨论：

1. 当丢包率设置的较大时，终端有时一直卡住无法进行通信

经分析，原因应当是当丢包率较大时客户端申请通信的数据帧丢包，导致两者无法建立通信。因此在实现SR双向传输的过程中，增加了判断：如果发送申请通信的数据帧后超时，则重传申请通信的数据帧。

2. 当执行程序时，客户端终端总是一点开就异常退出

分析后发现是编程中，直接将server的初始化套接字部分粘贴过去。Client初始化套接字的过程中并不需要对server的地址绑定端口。

心得体会：

对于停等协议、GBN协议以及SR协议的思想与实现步骤有了更深入的了解，同时对于利用socket进行编程的步骤和方法有了更深的掌握。对于协议的学习很有帮助。

GBN协议源代码：

只展示server：

```
1. // GBN_client.cpp : 定义控制台应用程序的入口点。
2. // #include "stdafx.h"
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <WinSock2.h>
6. #include<Ws2tcpip.h>
7. #include <time.h>
8. #include <fstream>
9.
10. #pragma comment(lib,"ws2_32.lib")
11.
12. #define SERVER_PORT 12340 //接收数据的端口号
13. #define SERVER_IP  "127.0.0.1" // 服务器的 IP 地址
14.
15. const int BUFFER_LENGTH = 1026;
```

```
16. const int SEQ_SIZE = 20;//接收端序列号个数，为 1~20
17. 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）
18.
19. BOOL ack[SEQ_SIZE];          //收到 ack 情况，对
    应 0~19 的 ack
20. const int SEQ_NUMBER = 33;    //设置 seq 的数量
21. int curSeq;                  //当前数据包的 seq
22. int curAck;                  //当前等待确认的 ack
23. int totalPacket;             //需要发送的包总数
24.
25. int totalSeq;    //已发送的包的总数
26. int totalAck;    //确认收到（ack）的包的总数
27. int finish;      //标志位：数据传输是否完成（finish=1->数据
    传输已完成）
28. int finish_S;
29.
30. //************************************
31. // Method: getCurTime
32. // FullName: getCurTime
33. // Access: public
34. // Returns: void
35. // Qualifier: 获取当前系统时间，结果存入 ptime 中
36. // Parameter: char * ptime
37. //************************************
38. void getCurTime(char* ptime) {
39.  char buffer[128];
40.  memset(buffer, 0, sizeof(buffer));
41.  time_t c_time;
42.  struct tm p;
```

```
43.  time(&c_time);
44.  localtime_s(&p, &c_time);
45.  sprintf_s(buffer, sizeof(buffer), "%d/%d/%d %d:%d:%d",
46.   p.tm_year + 1900,
47.   p.tm_mon + 1,
48.   p.tm_mday,
49.   p.tm_hour,
50.   p.tm_min,
51.   p.tm_sec);
52.  strcpy_s(ptime, 128, buffer);
53. }
54.
55.
56.
57.
58. /*************************************************************
    *********/
59. /*  -time 从服务器端获取当前时间

60.  -quit 退出客户端

61.  -testgbn [X] 测试 GBN 协议实现可靠数据传输

62.    [X] [0,1] 模拟数据包丢失的概率

63.    [Y] [0,1] 模拟 ACK 丢失的概率
64. */
65. /*************************************************************
    *********/
66. void printTips() {
67.  printf("*****************************************\n
    ");
68.  printf("|     -
    time to get current time              |\n");
69.  printf("|     -
    quit to exit client                   |\n");
70.  printf("|     -
    testgbn [X] [Y] to test the gbn       |\n");
71.  printf("|     -
    testgbn_Send [X] [Y] to test the gbn  |\n");
72.  printf("*****************************************\n
    ");
73. }
```

```
74.
75. //************************************
76. // Method:    lossInLossRatio
77. // FullName:  lossInLossRatio
78. // Access:    public
79. // Returns:   BOOL
80. // 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
81. // Parameter: float lossRatio [0,1]
82. //************************************
83. BOOL lossInLossRatio(float lossRatio) {
84.  int lossBound = (int)(lossRatio * 100);
85.  int r = rand() % 101;
86.  if (r <= lossBound) {
87.   return TRUE;
88.  }
89.  return FALSE;
90. }
91.
92. //************************************
93. // Method: seqIsAvailable
94. // FullName: seqIsAvailable
95. // Access: public
96. // Returns: bool
97. // Qualifier: 当前序列号 curSeq 是否可用
98. //************************************
99. bool seqIsAvailable() {
100.  int step;
101.  step = curSeq - curAck;
102.  step = step >= 0 ? step : step + SEQ_SIZE;
103.  //序列号是否在当前发送窗口之内
104.  if (step >= SEND_WIND_SIZE) {
105.   return false;
106.  }
107.  if (ack[curSeq]) {
108.   return true;
109.  }
110.  return false;
111. }
112.
```

```
113. //************************************
114. // Method: timeoutHandler
115. // FullName: timeoutHandler
116. // Access: public
117. // Returns: void
118. // Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传
119. //************************************
120. void timeoutHandler() {
121.  printf("Timer out error.");
122.  int index;
123.  for (int i = 0; i < SEND_WIND_SIZE; ++i) {
124.   index = (i + curAck) % SEQ_SIZE;
125.   ack[index] = TRUE;
126.  }
127.
128.  /*判断数据传输是否完成添加或修改*/
129.  if (totalSeq == totalPacket) {//之前发送到了最后一个数据
     包
130.   if (curSeq > curAck) {
131.    totalSeq -= (curSeq - curAck);
132.   }
133.   else if (curSeq < curAck) {
134.    totalSeq -= (curSeq - curAck + 20);
135.   }
136.  }
137.  else {//之前没发送到最后一个数据包
138.   totalSeq -= SEND_WIND_SIZE;
139.  }
140.  /*判断数据传输是否完成添加或修改*/
141.
142.  curSeq = curAck;
143. }
144.
145. //************************************
146. // Method: ackHandler
147. // FullName: ackHandler
148. // Access: public
149. // Returns: void
```

```
150. // Qualifier: 收到 ack，累积确认，取数据帧的第一个字节
151. //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
152. // Parameter: char c
153. //*************************************
154. void ackHandler(char c) {
155.  unsigned char index = (unsigned char)c - 1; //序列号减一
156.  printf("Recv an ack of seq %d \n", index + 1);//从接收方收到的确认收到的序列号
157.
158.  /*判断数据传输是否完成添加或修改的*/
159.  if (curAck <= index) {
160.   for (int i = curAck; i <= index; ++i) {
161.    ack[i] = TRUE;
162.   }
163.   printf("\t\tcurAck <= index , totalAck += %d\n", (index - curAck + 1));
164.   totalAck += (index - curAck + 1);
165.   curAck = (index + 1) % SEQ_SIZE;
166.  }
167.  else if (curAck != index + 1 && curAck > index) {
168.   //ack 超过了最大值，回到了 curAck 的左边
169.   for (int i = curAck; i < SEQ_SIZE; ++i) {
170.    ack[i] = TRUE;
171.   }
172.   for (int i = 0; i <= index; ++i) {
173.    ack[i] = TRUE;
174.   }
175.   totalAck += (SEQ_SIZE - curAck + index + 1);
176.   printf("\t\tcurAck > index , totalAck += %d\n", (SEQ_SIZE - curAck + index + 1));
177.   curAck = index + 1;
178.  }
```

```
179.  /*判断数据传输是否完成添加或修改*/
180. }
181.
182. int main(int argc, char* argv[]) {
183.  //加载套接字库（必须）
184.  WORD wVersionRequested;
185.  WSADATA wsaData;
186.  //套接字加载时错误提示
187.  int err;
188.  //版本 2.2
189.  wVersionRequested = MAKEWORD(2, 2);
190.  //加载 dll 文件 Scoket 库
191.  err = WSAStartup(wVersionRequested, &wsaData);
192.  if (err != 0) {
193.   //找不到 winsock.dll
194.   printf("WSAStartup failed with error: %d\n", err);
195.   return 1;
196.  }
197.  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wV
   ersion) != 2) {
198.   printf("Could not find a usable version of Winsock.dl
   l\n");
199.   WSACleanup();
200.  }
201.  else {
202.   printf("The Winsock 2.2 dll was found okay\n");
203.  }
204.  SOCKET socketClient = socket(AF_INET, SOCK_DGRAM, 0);
205.  SOCKADDR_IN addrServer;
206.  inet_pton(AF_INET, SERVER_IP, &(addrServer.sin_addr));
207.  addrServer.sin_family = AF_INET;
208.  addrServer.sin_port = htons(SERVER_PORT);
209.  //接收缓冲区
210.  char buffer[BUFFER_LENGTH];
211.  ZeroMemory(buffer, sizeof(buffer));
212.  int len = sizeof(SOCKADDR);
```

```
213.  //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
214.  //使用 -testgbn [X] [Y] 测试 GBN 其中[X]表示数据包丢失概率
215.  //          [Y]表示 ACK 丢包概率
216.  printTips();
217.  int ret;
218.  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
219.  char cmd[128];
220.
221.  int length = sizeof(SOCKADDR);
222.
223.  float packetLossRatio = 0.2; //默认包丢失率 0.2
224.  float ackLossRatio = 0.2; //默认 ACK 丢失率 0.2
225.  //用时间作为随机种子，放在循环的最外面
226.  srand((unsigned)time(NULL));
227.
228.  ZeroMemory(buffer, sizeof(buffer));
229.  //将测试数据读入内存
230.  std::ifstream icin;
231.  icin.open("D:/computer_net/lab2/client_procedure/test.txt");
232.  char data[1024 * SEQ_NUMBER];
233.  ZeroMemory(data, sizeof(data));
234.  icin.read(data, 1024 * SEQ_NUMBER);
235.  icin.close();
236.  totalPacket = sizeof(data) / 1024;
237.  printf("totalPacket is : %d\n\n", totalPacket);
238.  int recvSize;
239.  finish = 0;
240.  for (int i = 0; i < SEQ_SIZE; ++i) {
```

```
241.    ack[i] = TRUE;
242.   }
243.   finish = 0;
244.   finish_S = 0;
245.
246.   while (true) {
247.    fgets(buffer, sizeof(buffer), stdin);
248.    ret = sscanf_s(buffer, "%s %f %f", cmd, sizeof(cmd),
    &packetLossRatio, &ackLossRatio);
249.    //开始 GBN 测试，使用 GBN 协议实现 UDP 可靠文件传输
250.    if (!strcmp(cmd, "-testgbn")) {
251.     printf("%s\n", "Begin to test GBN protocol, please d
    on't abort the process");
252.     printf("The loss ratio of packet is %.2f,the loss ra
    tio of ack is %.2f\n", packetLossRatio, ackLossRatio);
253.     int waitCount = 0;
254.     int stage = 0;
255.     BOOL b;
256.     unsigned char u_code;     //状态码
257.     unsigned short seq;       //包的序列号
258.     unsigned short recvSeq;   //接收窗口大小为 1，已确认的
    序列号
259.     unsigned short waitSeq;   //等待的序列号
260.     sendto(socketClient, "-testgbn", strlen("-
    testgbn") + 1, 0, (SOCKADDR*)&addrServer, sizeof(SOCKADD
    R));
261.     while (true) {
262.      //等待 server 回复设置 UDP 为阻塞模式
263.      recvfrom(socketClient, buffer, BUFFER_LENGTH, 0, (S
    OCKADDR*)&addrServer, &len);
264.      switch (stage) {
265.      case 0://等待握手阶段
266.       u_code = (unsigned char)buffer[0];
267.       if ((unsigned char)buffer[0] == 205) {
268.        printf("Ready for file transmission\n");
269.        buffer[0] = 200;
```

```
270.        buffer[1] = '\0';
271.        sendto(socketClient, buffer, 2, 0, (SOCKADDR*)&ad
   drServer, sizeof(SOCKADDR));
272.        stage = 1;
273.        recvSeq = 0;
274.        waitSeq = 1;
275.       }
276.       break;
277.      case 1://等待接收数据阶段
278.       seq = (unsigned short)buffer[0];
279.       //随机法模拟包是否丢失
280.       b = lossInLossRatio(packetLossRatio);
281.       if (b) {
282.        printf("The packet with a seq of %d loss\n", seq)
   ;
283.        continue;
284.       }
285.       printf("recv a packet with a seq of %d\n", seq);
286.       //如果是期待的包，正确接收，正常确认即可
287.       if (!(waitSeq - seq)) {
288.        ++waitSeq;
289.        if (waitSeq == 21) {
290.         waitSeq = 1;
291.        }
292.        //输出数据
293.        printf("\n\n\t%s\n\n", &buffer[1]);
294.        buffer[0] = seq;
295.        recvSeq = seq;
296.        buffer[1] = '\0';
297.       }
298.       else {
299.        //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）
300.        if (!recvSeq) {
301.         continue;
302.        }
303.        buffer[0] = recvSeq;
304.        buffer[1] = '\0';
```

```
305.       }
306.       b = lossInLossRatio(ackLossRatio);
307.       if (b) {
308.        printf("The ack of %d loss\n", (unsigned char)buf
   fer[0]);
309.        continue;
310.       }
311.       sendto(socketClient, buffer, 2, 0, (SOCKADDR*)&add
   rServer, sizeof(SOCKADDR));
312.       printf("send a ack of %d\n", (unsigned char)buffer
   [0]);
313.       break;
314.      }
315.      Sleep(500);
316.     }
317.    }
318.    else if (strcmp(cmd, "-time") == 0) {
319.     getCurTime(buffer);
320.    }
321.    else if (!strcmp(cmd, "-testgbn_Send")) {
322.     finish_S = 0;
323.     for (int i = 0; i < SEQ_SIZE; ++i) {
324.      ack[i] = TRUE;
325.     }
326.     //进入 gbn 测试阶段
327.     //首先 server（server 处于 0 状态）向 client 发
   送 205 状态码（server 进入 1 状态）
328.     //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\
329.     //在文件传输阶段，server 发送窗口大小设为
330.     ZeroMemory(buffer, sizeof(buffer));
331.     int recvSize;
332.     int waitCount = 0;
333.     printf("Begain to test GBN protocol,please don't abo
   rt the process\n");
```

```
334.     //加入了一个握手阶段
335.     //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
336.     //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
337.     //服务器收到 200 状态码之后，就开始使用 GBN 发送数据了
338.     printf("Shake hands stage\n");
339.     int stage = 0;
340.     bool runFlag = true;
341.     sendto(socketClient, "-testgbn_Send", strlen("-testgbn_Send") + 1, 0,
342.      (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
343.     Sleep(100);
344.     recvfrom(socketClient, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrServer), &length);
345.     printf("\n%s\n\n", buffer);
346.     ZeroMemory(buffer, sizeof(buffer));
347.     int iMode = 1; //1: 非阻塞，0: 阻塞
348.     ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & iMode);//非阻塞设置
349.     while (runFlag) {
350.      switch (stage) {
351.      case 0://发送 205 阶段
352.       buffer[0] = 205;
353.       sendto(socketClient, buffer, strlen(buffer) + 1, 0,
354.        (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
355.       Sleep(100);
356.       stage = 2;
357.       break;
358.      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
```

```
359.       recvSize = recvfrom(socketClient, buffer, BUFFER_L
   ENGTH, 0, ((SOCKADDR*)&addrServer), &length);
360.       if (recvSize < 0) {
361.        ++waitCount;
362.        if (waitCount > 20) {
363.         runFlag = false;
364.         printf("Timeout error\n");
365.         break;
366.        }
367.        Sleep(500);
368.        continue;
369.       }
370.       else {
371.
372.        if ((unsigned char)buffer[0] == 200) {
373.         printf("Begin a file transfer\n");
374.         printf("File size is %dB, each packet is 1024B
   and packet total num is %d\n", sizeof(data), totalPacket
   );
375.         curSeq = 0;
376.         curAck = 0;
377.         totalSeq = 0;
378.         waitCount = 0;
379.         totalAck = 0;
380.         finish = 0;
381.         stage = 2;
382.        }
383.       }
384.       break;
385.      case 2://数据传输阶段
386.
387.       /*判断数据传输是否完成添加或修改*/
388.       if (seqIsAvailable() && totalSeq <= (totalPacket -
    1)) {//totalSeq<=(totalPacket-1)：未传到最后一个数据包
389.        /*判断数据传输是否完成添加或修改*/
390.
391.         //发送给客户端的序列号从 1 开始
392.        buffer[0] = curSeq + 1;
393.        ack[curSeq] = FALSE;
```

```
394.        //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
395.        //为简化过程此处并未实现->现在此代码已经实现了ok
396.        memcpy(&buffer[1], data + 1024 * totalSeq, 1024);
397.        printf("send a packet with a seq of : %d \t total Seq now is : %d\n", curSeq + 1, totalSeq + 1);
398.        sendto(socketClient, buffer, BUFFER_LENGTH, 0,
399.         (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
400.        ++curSeq;
401.        curSeq %= SEQ_SIZE;
402.        ++totalSeq;
403.        Sleep(500);
404.       }
405.       //等待 Ack，若没有收到，则返回值为-1，计数器+1
406.       recvSize = recvfrom(socketClient, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrServer), &length);
407.       if (recvSize < 0) {
408.        waitCount++;
409.        //20 次等待 ack 则超时重传
410.        if (waitCount > 20) {
411.         timeoutHandler();
412.         printf("\t----
totalSeq Now is : %d\n", totalSeq);
413.         waitCount = 0;
414.        }
415.       }
416.       else {
417.        //收到 ack
418.        ackHandler(buffer[0]);
419.        printf("\t\t----
totalAck Now is : %d\n", totalAck);
420.        waitCount = 0;
421.
422.        /*判断数据传输是否完成添加或修改*/
423.        if (totalAck == totalPacket) {//数据传输完成
424.         finish_S = 1;
```

```
425.          break;
426.         }
427.         /*判断数据传输是否完成添加或修改*/
428.
429.
430.        }
431.        Sleep(500);
432.        break;
433.       }
434.
435.
436.       /*判断数据传输是否完成添加或修改*/
437.       if (finish_S == 1) {
438.        printf("数据传输全部完成！！！\n");
439.        strcpy_s(buffer, sizeof(buffer), "数据传输全部完
    成！！！\n");
440.        sendto(socketClient, buffer, strlen(buffer) + 1, 0
    , (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
441.        break;
442.       }
443.       /*判断数据传输是否完成添加或修改*/
444.
445.     }
446.     iMode = 0; //1：非阻塞，0：阻塞
447.     ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & i
    Mode);//非阻塞设置
448.    }
449.
450.    /*判断数据传输是否完成添加或修改*/
451.    if (finish == 1) {
452.     printf("数据传输全部完成！！！\n\n");
453.     printTips();
454.     finish = 0;
455.     continue;
```

```
456.    }
457.    if (finish_S == 1) {
458.     printTips();
459.     finish_S = 0;
460.     continue;
461.    }
462.    /*判断数据传输是否完成添加或修改*/
463.
464.    sendto(socketClient, buffer, strlen(buffer) + 1, 0, (
    SOCKADDR*)&addrServer, sizeof(SOCKADDR));
465.    ret = recvfrom(socketClient, buffer, BUFFER_LENGTH, 0
    , (SOCKADDR*)&addrServer, &len);
466.    printf("%s\n", buffer);
467.    if (!strcmp(buffer, "Good bye!")) {
468.     break;
469.    }
470.    printTips();
471.   }
472.   //关闭套接字
473.   closesocket(socketClient);
474.   WSACleanup();
475.   return 0;
476. }
```

SR协议源代码：

Server

```
1. #include <stdlib.h>
2. #include <time.h>
3. #include <WinSock2.h>
4. #include <WS2tcpip.h>
5. #include <fstream>
6. #pragma comment(lib,"ws2_32.lib")
7.
8. #define SERVER_PORT 12340 // 服务器端口号
9. #define SERVER_IP "127.0.0.1" // 服务器IP 地址
10.
```

```
#define SEQ_SIZE 20 // 序列号个数
#define SWIN_SIZE 10 // 发送窗口大小
#define RWIN_SIZE 10 // 接收窗口大小
#define BUFFER_SIZE 1024 // 缓冲区大小
#define LOSS_RATE 0.2 //丢包率

SOCKET socketServer;       //服务器 socket
SOCKADDR_IN addrServer; // 服务器网络地址

using namespace std;
struct recv {
 bool used;
 char buffer[BUFFER_SIZE];
 recv() {
  used = false;
  ZeroMemory(buffer, sizeof(buffer));
 }
}recvWindow[SEQ_SIZE];

struct send {
 clock_t start;//每个数据包有一个计时器
 char buffer[BUFFER_SIZE];
 send() {
  start = 0;
  ZeroMemory(buffer, sizeof(buffer));
 }
}sendWindow[SEQ_SIZE];

char cmdBuffer[50];    //命令缓冲区
char buffer[BUFFER_SIZE]; //大小为 1024 的缓冲区
char cmd[10];    //命令
char fileName[40];  //文件名
```

```
43. char filePath[50];  //文件路径
44. char file[1024 * 1024]; //文件内容指针
45. int len = sizeof(SOCKADDR);
46. int recvSize;
47.
48. int Deliver(char* file, int ack);
49. int Send(ifstream& infile, int seq, SOCKET socket, SOCKADDR* addr);
50. int MoveSendWindow(int seq);
51. int Read(ifstream& infile, char* buffer);
52.
53. //主函数
54. int main(int argc, char* argv[]) {
55.
56.  //加载套接字库（必须）
57.  WORD wVersionRequested;
58.  WSADATA wsaData;
59.  //套接字加载时错误提示
60.  int err;
61.  //版本 2.2
62.  wVersionRequested = MAKEWORD(2, 2);
63.  //加载 dll 文件 Socket 库
64.  err = WSAStartup(wVersionRequested, &wsaData);
65.  if (err != 0) {
66.   //找不到 winsock.dll
67.   printf("加载 winsock 失败， 错误代码为: %d\n", WSAGetLastError());
68.   return FALSE;
69.  }
70.  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
71.   printf("不能找到正确的 winsock 版本\n");
72.   WSACleanup();
```

```
73.   return FALSE;
74.  }
75.  else    printf("The Winsock 2.2 dll was found okay\n");
76.
77.  // 创建服务器套接字
78.  //AF_INET: IP7
79.  //SOCK_DGRAM: UDP 协议
80.  SOCKET socketServer = socket(AF_INET, SOCK_DGRAM, IPPRO
   TO_UDP);
81.  // 设置为非阻塞模式
82.  int iMode = 1;
83.  //设置服务器套接字的IO 模式
84.  ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & iMod
   e);
85.  //服务器网络地址
86.  SOCKADDR_IN addrServer;
87.  inet_pton(AF_INET, SERVER_IP, &addrServer.sin_addr);
88.  addrServer.sin_family = AF_INET;
89.  addrServer.sin_port = htons(SERVER_PORT);
90.  // 绑定端口@@@@@@@@客户端不需要绑定端口
91.  if (err = bind(socketServer, (SOCKADDR*)&addrServer, si
   zeof(SOCKADDR))) {
92.   err = GetLastError();
93.   printf("绑定端口 %d 失败，错误

   码: % d\n", SERVER_PORT, err);
94.   WSACleanup();
95.   return -1;
96.  }
97.  else
98.  {
99.   printf("绑定端口 %d 成功", SERVER_PORT);
100.   }
101.
```

```
102.  //客户端地址
103.  SOCKADDR_IN addrClient;
104.  int status = 0; //自动机状态——初始化为0
105.  clock_t start; //计时器
106.  clock_t now;
107.  int seq;
108.  int ack;
109.  ofstream outfile;
110.  ifstream infile;
111.  //进入接收状态，注意服务器主要处理的任务是接收客户机请求，共有上载和下载两种任务
112.  while (true) {
113.   //服务器套接字从客户端地址接收信息，传入 buffer
114.   //收到的 buffer 的结构：'10'+cmd[10]+filename[40]
115.   recvSize = recvfrom(socketServer, buffer, BUFFER_SIZE, 0, ((SOCKADDR*)&addrClient), &len);
116.   if ((float)rand() / RAND_MAX < LOSS_RATE) {  //设置丢包
117.    recvSize = 0;
118.    buffer[0] = 0;  //清空 buffer 首位的状态字节
119.   }
120.   switch (status) //进入自动机状态的循环
121.   {
122.   case 0://接收请求
123.    if (recvSize > 0 && buffer[0] == 10) {
124.     char addr[100];
125.     ZeroMemory(addr, sizeof(addr)); //全置零
126.     inet_ntop(AF_INET, &addrClient.sin_addr, addr, sizeof(addr));
127.
```

```
128.      //buffer 中将命令与文件名用数组分隔开
129.      sscanf_s(buffer + 1, "%s%s", cmd, sizeof(cmd) - 1,
   fileName, sizeof(fileName) - 1);
130.
131.      if (strcmp(cmd, "upload") && strcmp(cmd, "download"
   )) continue;  //判断命令是否合法
132.      strcpy_s(filePath, "./");
133.      strcat_s(filePath, fileName); //获取文件路径
134.
135.      printf("收到来自客户端 %s 的请
   求: %s\n", addr, buffer); //去除请求判断
136.      printf("是否同意该请求(Y/N)?");
137.      gets_s(cmdBuffer, 50); //命令缓冲区
138.      if (!strcmp(cmdBuffer, "Y")) {
139.       buffer[0] = 100; //缓冲区首字节的状态字
140.       strcpy_s(buffer + 1, 3, "OK"); //将指令写入缓冲区
141.       if (!strcmp(cmd, "upload")) {
142.        file[0] = 0; //如果是要上传
143.        start = clock();
144.        ack = 0;
145.        status = 1;
146.        outfile.open(filePath); //打开/创建对应文件
147.       }
148.       else if (!strcmp(cmd, "download")) {
149.        start = clock();
150.        seq = 0;
151.        status = -1;
152.        infile.open(filePath);//如果要下载就打开对应文件
153.       }
154.      }
155.      else {
156.       buffer[0] = 100;
```

```
157.       strcpy_s(buffer + 1, 3, "NO");
158.      }
159.      //把首字节为100 的buffer 发回去，开始通信
160.      sendto(socketServer, buffer, strlen(buffer) + 1, 0,
    (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
161.     }
162.     break;
163.    case 1://客户机请求上传，也就是服务器端是接收方
164.     if (recvSize > 0) {
165.      if (buffer[0] == 10) {  //状态字节
166.       if (!strcmp(buffer + 1, "Finish")) {
167.        printf("传输完毕...\n");
168.        start = clock();
169.        sendWindow[0].start = start - 1000L;
170.        sendWindow[0].buffer[0] = 100;
171.        strcpy_s(sendWindow[0].buffer + 1, 3, "OK");
172.        outfile.write(file, strlen(file));
173.        status = 2;
174.       }
175.       buffer[0] = 100;
176.       strcpy_s(buffer + 1, 3, "OK");
177.       sendto(socketServer, buffer, strlen(buffer) + 1, 0
    , (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
178.      }
179.      else if (buffer[0] == 20) { //接收到一个数据包
180.       seq = buffer[1];
181.       int temp = seq - 1 - ack; //步长
182.       if (temp < 0) {
183.        temp += SEQ_SIZE;
184.       }
185.       start = clock();
186.       seq--;
187.       if (temp < RWIN_SIZE) {  //步长小于窗口大小，可以继
    续发送
```

```
188.        if (!recvWindow[seq].used) { //如果接受窗口当前待发送可用
189.         recvWindow[seq].used = true;
190.         strcpy_s(recvWindow[seq].buffer, strlen(buffer + 2) + 1, buffer + 2);
191.        }
192.        if (ack == seq) {    //如果当前接收到的数据帧是最小未接受序号
193.         ack = Deliver(file, ack); //这一块数据帧可以写入文件，并且向前滑动窗口
194.        }
195.       }
196.       printf("接收数据帧 seq = %d, data = %s, 发送 ack = %d, 起始 ack = %d\n", seq + 1, buffer + 2, seq + 1, ack + 1);
197.       buffer[0] = 101;  //返回ack 信息
198.       buffer[1] = seq + 1;
199.       buffer[2] = 0;
200.       sendto(socketServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
201.      }
202.     }
203.     break;
204.    case 2://接收完成
205.     if (recvSize > 0 && buffer[0] == 10 && !strcmp(buffer + 1, "OK")) { //状态码为0，且不是开始通信
206.      printf("传输成功，结束通信\n");
207.      status = 0;
208.      outfile.close();
209.     }
210.     now = clock();
```

```
211.    if (now - sendWindow[0].start >= 1000L) {
212.     sendWindow[0].start = now;
213.     sendto(socketServer, sendWindow[0].buffer, strlen(s
   endWindow[0].buffer) + 1, 0, (SOCKADDR*)&addrClient, siz
   eof(SOCKADDR));
214.    }
215.    break;
216.   case -1://客户机请求下载，也就是服务器端充当发送方
217.    if (recvSize > 0) {
218.     if (buffer[0] == 10) {  //返回开始通信的状态码
219.      if (!strcmp(buffer + 1, "OK")) {
220.       printf("开始传输...\n");
221.       start = clock();
222.       status = -2;
223.      }
224.      buffer[0] = 100;
225.      strcpy_s(buffer + 1, 3, "OK");
226.      sendto(socketServer, buffer, strlen(buffer) + 1, 0
   , (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
227.     }
228.    }
229.    //break;
230.    continue;
231.   case -2://服务器端发送数据
232.    if (recvSize > 0 && buffer[0] == 11) { //接受到ack
233.     start = clock();
234.     ack = buffer[1];
235.     ack--;
236.     sendWindow[ack].start = -1L; //收到了ack 则重置，不
   超时
237.     if (ack == seq) {
238.      seq = MoveSendWindow(seq);
239.     }
240.     printf("接收 ack = %d, 当前起
   始 seq = %d\n", ack + 1, seq + 1);
```

```
241.     }
242.     if (!Send(infile, seq, socketServer, (SOCKADDR*)&add
   rClient)) { //到文件结尾
243.      printf("传输完毕...\n");
244.      status = -3;
245.      start = clock();
246.      sendWindow[0].buffer[0] = 100;
247.      strcpy_s(sendWindow[0].buffer + 1, 7, "Finish"); //
   传输结束信息
248.      sendWindow[0].start = start - 1000L;
249.     }
250.     break;
251.    case -3://请求完成
252.     if (recvSize > 0 && buffer[0] == 10) {
253.      if (!strcmp(buffer + 1, "OK")) {
254.       printf("传输成功，结束通信\n");
255.       infile.close();
256.       status = 0;
257.       break;
258.      }
259.     }
260.     now = clock();
261.     if (now - sendWindow[0].start >= 1000L) {
262.      sendWindow[0].start = now;
263.      sendto(socketServer, sendWindow[0].buffer, strlen(s
   endWindow[0].buffer) + 1, 0, (SOCKADDR*)&addrClient, siz
   eof(SOCKADDR));
264.     }
265.    default:
266.     break;
267.    }
268.    if (status != 0 && clock() - start > 20000L) {
269.     printf("通信超时，结束通信\n");
270.     status = 0;
271.     outfile.close();
272.     continue;
273.    }
```

```
274.    if (recvSize <= 0) { //没有接收到信息时需要休眠
275.     Sleep(20);
276.    }
277.   }
278.   //关闭套接字，卸载库
279.   closesocket(socketServer);
280.   WSACleanup();
281.   return 0;
282. }
283. int Read(ifstream& infile, char* buffer) {
284.   //从文件中读取需要发送的数据
285.   if (infile.eof()) {
286.    return 0;
287.   }
288.   infile.read(buffer, 3);
289.   int cnt = infile.gcount();
290.   buffer[cnt] = 0;
291.   return cnt;
292. }
293. int Deliver(char* file, int ack) {
294.   while (recvWindow[ack].used) {
295.    recvWindow[ack].used = false;
296.    strcat_s(file, strlen(file) + strlen(recvWindow[ack].
    buffer) + 1, recvWindow[ack].buffer);
297.    ack++;
298.    ack %= SEQ_SIZE;
299.   }
300.   return ack;
301. }
302. int Send(ifstream& infile, int seq, SOCKET socket, SOCK
    ADDR* addr) {
303.   //发送数据
304.   clock_t now = clock();
305.   for (int i = 0; i < SWIN_SIZE; i++) {
306.    int j = (seq + i) % SEQ_SIZE;
307.    if (sendWindow[j].start == -1L) { //还未发送
308.     continue;
309.    }
```

```
310.    if (sendWindow[j].start == 0L) { //超时，重新加载
    buffer
311.     if (Read(infile, sendWindow[j].buffer + 2)) {//从文
    件中读取该数据帧存储的内容
312.      sendWindow[j].start = now;
313.      sendWindow[j].buffer[0] = 200;  //表示发送数据帧
314.      sendWindow[j].buffer[1] = j + 1; //序号
315.     }
316.     else if (i == 0) {
317.      return 0;
318.     }
319.     else {
320.      break;
321.     }
322.    }
323.    else if (now - sendWindow[j].start >= 1000L) {//更新
    时间
324.     sendWindow[j].start = now;
325.    }
326.    else {
327.     continue;
328.    }
329.    printf("发送数据
    帧 seq = %d, data = %s\n", j + 1, sendWindow[j].buffer +
     2);
330.    sendto(socket, sendWindow[j].buffer, strlen(sendWindo
    w[j].buffer) + 1, 0, addr, sizeof(SOCKADDR));
331.   }
332.   return 1;
333. }
334.
335. int MoveSendWindow(int seq) { //当发送序号与 ack 序号相等
    时移动，因此只需要判断是否超时而不需要判断发送未 ack 的情况
```

```
336.  //移动窗口
337.  while (sendWindow[seq].start == -1L) {
338.   sendWindow[seq].start = 0L;
339.   seq++;
340.   seq %= SEQ_SIZE;
341.  }
342.  return seq;
343. }
```

Client：

```
1.  #include <stdlib.h>
2.  #include <time.h>
3.  #include <WinSock2.h>
4.  #include <WS2tcpip.h>
5.  #include <fstream>
6.  #pragma comment(lib,"ws2_32.lib")
7.
8.  #define SERVER_PORT 12340 // 接收数据的端口号
9.  #define SERVER_IP "127.0.0.1" // 服务器的 IP 地址
10. #define BUFFER_SIZE 1024 // 缓冲区大小
11. #define SEQ_SIZE 20 // 序列号个数
12. #define SWIN_SIZE 10 // 发送窗口大小
13. #define RWIN_SIZE 10 // 接收窗口大小
14. #define LOSS_RATE 0.2 // 丢包率
15.
16. SOCKET socketServer;       //服务器 socket
17. SOCKADDR_IN addrServer; // 服务器网络地址
18. SOCKET socketClient;
19. using namespace std;
20.
21. char cmdBuffer[50];
22. char buffer[BUFFER_SIZE];
23. char cmd[10];
```

```
char fileName[40];
char filePath[50];
char file[1024 * 1024];
int len = sizeof(SOCKADDR);
int recvSize;
int Deliver(char* file, int ack);
int Send(ifstream& infile, int seq, SOCKET socket, SOCKADDR* addr);
int MoveSendWindow(int seq);
int Read(ifstream& infile, char* buffer);
struct Cache {
 bool used;
 char buffer[BUFFER_SIZE];
 Cache() {
  used = false;
  ZeroMemory(buffer, sizeof(buffer));
 }
}recvWindow[SEQ_SIZE];
struct DataFrame {
 clock_t start;
 char buffer[BUFFER_SIZE];
 DataFrame() {
  start = 0;
  ZeroMemory(buffer, sizeof(buffer));
 }
}sendWindow[SEQ_SIZE];

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 Socket 库
 err = WSAStartup(wVersionRequested, &wsaData);
 if (err != 0) {
```

```
62.   //找不到 winsock.dll
63.   printf("加载 winsock 失败， 错误代码为: %d\n", WSAGetLastError());
64.   return FALSE;
65.  }
66.  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
67.   printf("不能找到正确的 winsock 版本\n");
68.   WSACleanup();
69.   return FALSE;
70.  }
71.  else    printf("The Winsock 2.2 dll was found okay\n");
72.
73.  // 创建服务器套接字
74.  socketClient = socket(AF_INET, SOCK_DGRAM, 0);
75.
76.  // 设置为非阻塞模式
77.  int iMode = 1;
78.  //设置客户端服务器套接字的IO模式
79.  ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & iMode);
80.  SOCKADDR_IN addrServer;
81.  inet_pton(AF_INET, SERVER_IP, &addrServer.sin_addr);
82.  addrServer.sin_family = AF_INET;
83.  addrServer.sin_port = htons(SERVER_PORT);
84.
85.  int status = 0;
86.  clock_t start;
87.  clock_t now;
88.  int seq;
89.  int ack;
90.  srand((unsigned)time(NULL));
91.  while (true) {
92.   gets_s(cmdBuffer, 50); //先从键盘输入缓冲区，再从缓冲区中读取
```

```
93.   sscanf_s(cmdBuffer, "%s%s", cmd, sizeof(cmd) - 1, fileName, sizeof(fileName) - 1);
94.
95.   if (!strcmp(cmd, "upload")) {
96.    printf("申请上传文件: %s\n", fileName);
97.    strcpy_s(filePath, "./");
98.    strcat_s(filePath, fileName);
99.    ifstream infile(filePath);
100.    start = clock(); //开始计时
101.    seq = 0;
102.    status = 0;    //初始化自动机的状态为0
103.
104.    sendWindow[0].buffer[0] = 10;
105.    strcpy_s(sendWindow[0].buffer + 1, strlen(cmdBuffer) + 1, cmdBuffer);
106.    sendWindow[0].start = start - 1000L;
107.    //进入通信循环
108.    while (true) {
109.     //等待从服务器接收的信息
110.     //从服务器接收信息写入客户端的buffer
111.     recvSize = recvfrom(socketClient, buffer, BUFFER_SIZE, 0, (SOCKADDR*)&addrServer, &len);
112.     switch (status)
113.     {
114.     case 0:
115.      if (recvSize > 0 && buffer[0] == 100) {  //状态字节，表示控制
116.       if (!strcmp(buffer + 1, "OK")) {
117.        printf("通信开始，开始上传文件\n");
118.        start = clock();
119.        status = 1;
120.        sendWindow[0].start = 0L; //重新开始计时
121.        continue;
122.       }
```

```
123.        else if (!strcmp(buffer + 1, "NO")) {
124.         status = -1;
125.         break;
126.        }
127.       }
128.       now = clock();
129.       if (now - sendWindow[0].start >= 1000L) { //如果超时
130.        sendWindow[0].start = now;
131.        //
132.        sendto(socketClient, sendWindow[0].buffer, strlen(sendWindow[0].buffer) + 1, 0, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
133.       }
134.       break;
135.      case 1://开始上传
136.       if (recvSize > 0 && buffer[0] == 101) {  //表示ack
137.        start = clock();
138.        ack = buffer[1]; //接收缓冲区的位置
139.        ack--;    //错位
140.        sendWindow[ack].start = -1L; //重置
141.        if (ack == seq) {
142.         seq = MoveSendWindow(seq);  //移动窗口
143.        }
144.        printf("收到 ack = %d, 当前 seq = %d\n", ack + 1, seq + 1);
145.       }
146.       if (!Send(infile, seq, socketClient, (SOCKADDR*)&addrServer)) {
147.        printf("文件传输结束\n");
148.        status = 2;  //进入状态2
149.        start = clock();
```

```
150.        sendWindow[0].buffer[0] = 10;
151.        strcpy_s(sendWindow[0].buffer + 1, 7, "Finish");
152.        sendWindow[0].start = start - 1000L;
153.        continue;
154.       }
155.       break;
156.      case 2:
157.       if (recvSize > 0 && buffer[0] == 100) {
158.        if (!strcmp(buffer + 1, "OK")) {
159.         buffer[0] = 10;
160.         strcpy_s(buffer + 1, 3, "OK");
161.         sendto(socketClient, buffer, strlen(buffer) + 1,
    0, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
162.         status = 3;
163.         break;
164.        }
165.       }
166.       now = clock();
167.       if (now - sendWindow[0].start >= 1000L) {
168.        sendWindow[0].start = now;
169.        sendto(socketClient, sendWindow[0].buffer, strlen
   (sendWindow[0].buffer) + 1, 0, (SOCKADDR*)&addrServer, s
   izeof(SOCKADDR));
170.       }
171.      default:
172.       break;
173.      }
174.      if (status == -1) {
175.       printf("服务器拒绝请求\n");
176.       infile.close();
177.       break;
178.      }
179.      if (status == 3) {
180.       printf("上传成功，结束通信\n");
181.       infile.close();
182.       break;
183.      }
184.      if (clock() - start >= 10000L) {
185.       printf("通信超时，结束通信\n");
186.       infile.close();
187.       break;
```

```
188.      }
189.      if (recvSize <= 0) {
190.       Sleep(200);
191.      }
192.     }
193.    }
194.    else if (!strcmp(cmd, "download")) {
195.     printf("申请下载文件 %s\n", fileName);
196.     strcpy_s(filePath, "./");
197.     strcat_s(filePath, fileName);
198.     ofstream outfile(filePath);
199.     start = clock();
200.     ack = 0;
201.     status = 0;
202.     sendWindow[0].buffer[0] = 10; //状态字节为0
203.     strcpy_s(sendWindow[0].buffer + 1, strlen(cmdBuffer)
    + 1, cmdBuffer);
204.     sendWindow[0].start = start - 1000L; //重置
205.     while (true) {
206.      //从服务器读到
207.      recvSize = recvfrom(socketClient, buffer, BUFFER_SI
   ZE, 0, (SOCKADDR*)&addrServer, &len);
208.      if ((float)rand() / RAND_MAX < LOSS_RATE) { //设置
    丢包
209.       recvSize = 0;
210.       buffer[0] = 0;
211.      }
212.      switch (status)
213.      {
214.      case 0:
215.       if (recvSize > 0 && buffer[0] == 100) {
216.        if (!strcmp(buffer + 1, "OK")) {
217.         printf("申请通过，准备下载...\n");
218.         start = clock();
219.         status = 1;
220.         sendWindow[0].buffer[0] = 10;
221.         strcpy_s(sendWindow[0].buffer + 1, 3, "OK");
```

```
222.          sendWindow[0].start = start - 1000L;
223.          continue;
224.         }
225.         else if (!strcmp(buffer + 1, "NO")) {
226.          status = -1;
227.          break;
228.         }
229.        }
230.        now = clock();
231.        if (now - sendWindow[0].start >= 1000L) {
232.         sendWindow[0].start = now;
233.         sendto(socketClient, sendWindow[0].buffer, strlen
    (sendWindow[0].buffer) + 1, 0, (SOCKADDR*)&addrServer, s
    izeof(SOCKADDR));
234.        }
235.        break;
236.       case 1:
237.        if (recvSize > 0 && (unsigned char)buffer[0] == 20
    0) {
238.         printf("开始下载...\n");
239.         start = clock();
240.         seq = buffer[1];
241.         printf("接收数据帧 seq = %d, data = %s, 发送
    ack = %d\n", seq, buffer + 2, seq);
242.         seq--;
243.         recvWindow[seq].used = true;
244.         strcpy_s(recvWindow[seq].buffer, strlen(buffer +
    2) + 1, buffer + 2);
245.         if (ack == seq) {
246.          ack = Deliver(file, ack);
247.         }
248.         status = 2;
249.         buffer[0] = 11;
250.         buffer[1] = seq + 1;
251.         buffer[2] = 0;
252.         sendto(socketClient, buffer, strlen(buffer) + 1,
    0, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
253.         continue;
254.        }
255.        now = clock();
256.        if (now - sendWindow[0].start >= 1000L) {
257.         sendWindow[0].start = now;
```

```
258.         sendto(socketClient, sendWindow[0].buffer, strlen
   (sendWindow[0].buffer) + 1, 0, (SOCKADDR*)&addrServer, s
   izeof(SOCKADDR));
259.       }
260.       break;
261.     case 2:
262.       if (recvSize > 0) {
263.         if ((unsigned char)buffer[0] == 200) {
264.           seq = buffer[1];
265.           int temp = seq - 1 - ack;
266.           if (temp < 0) {
267.             temp += SEQ_SIZE;
268.           }
269.           start = clock();
270.           seq--;
271.           if (temp < RWIN_SIZE) {
272.             if (!recvWindow[seq].used) {
273.               recvWindow[seq].used = true;
274.               strcpy_s(recvWindow[seq].buffer, strlen(buffer
    + 2) + 1, buffer + 2);
275.             }
276.             if (ack == seq) {
277.               ack = Deliver(file, ack);
278.             }
279.           }
280.           printf("接收数据帧 seq = %d, data = %s, 发
   送 ack = %d\n", seq + 1, buffer + 2, seq + 1, ack + 1);
281.           buffer[0] = 11;
282.           buffer[1] = seq + 1;
283.           buffer[2] = 0;
284.           sendto(socketClient, buffer, strlen(buffer) + 1,
    0, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
285.         }
286.         else if (buffer[0] == 100 && !strcmp(buffer + 1,
   "Finish")) {
287.           status = 3;
288.           outfile.write(file, strlen(file));
289.           buffer[0] = 10;
290.           strcpy_s(buffer + 1, 3, "OK");
291.           sendto(socketClient, buffer, strlen(buffer) + 1,
    0, (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
292.           continue;
```

```
        }
       }
       break;
      default:
       break;
      }
      if (status == -1) {
       printf("服务器拒绝请求\n");
       outfile.close();
       break;
      }
      if (status == 3) {
       printf("下载成功, 结束通信\n");
       outfile.close();
       break;
      }
      if (clock() - start >= 20000L) {
       printf("通信超时, 结束通信\n");
       outfile.close();
       break;
      }
      if (recvSize <= 0) {
       Sleep(20);
      }
     }
    }
    else if (!strcmp(cmd, "quit")) {
     break;
    }
   }
   closesocket(socketClient);
   printf("关闭套接字\n");
   WSACleanup();
   return 0;
}

int Read(ifstream& infile, char* buffer) {
 if (infile.eof()) {
  return 0;
 }
```

```
333.  infile.read(buffer, 3);
334.  int cnt = infile.gcount();
335.  buffer[cnt] = 0;
336.  return cnt;
337. }
338.
339. int Deliver(char* file, int ack) {
340.  while (recvWindow[ack].used) {
341.   recvWindow[ack].used = false;
342.   strcat_s(file, strlen(file) + strlen(recvWindow[ack].
   buffer) + 1, recvWindow[ack].buffer);
343.   ack++;
344.   ack %= SEQ_SIZE;
345.  }
346.  return ack;
347. }
348.
349. int Send(ifstream& infile, int seq, SOCKET socket, SOCK
   ADDR* addr)
350. {
351.  clock_t now = clock();
352.  for (int i = 0; i < SWIN_SIZE; i++) {
353.   int j = (seq + i) % SEQ_SIZE;
354.   if (sendWindow[j].start == -1L) {
355.    continue;
356.   }
357.   if (sendWindow[j].start == 0L) {
358.    if (Read(infile, sendWindow[j].buffer + 2)) {
359.     sendWindow[j].start = now;
360.     sendWindow[j].buffer[0] = 20;
361.     sendWindow[j].buffer[1] = j + 1;
362.    }
363.    else if (i == 0) {
364.     return 0;
365.    }
366.    else {
367.     break;
368.    }
369.   }
370.   else if (now - sendWindow[j].start >= 1000L) {
371.    sendWindow[j].start = now;
372.   }
373.   else {
```

```
374.     continue;
375.    }
376.    printf("发送数据帧 seq = %d, data = %s\n", j + 1, sendWindow[j].buffer + 2);
377.    sendto(socket, sendWindow[j].buffer, strlen(sendWindow[j].buffer) + 1, 0, addr, sizeof(SOCKADDR));
378.   }
379.   return 1;
380. }
381.
382. int MoveSendWindow(int seq) {
383.   while (sendWindow[seq].start == -1L) {
384.    sendWindow[seq].start = 0L;
385.    seq++;
386.    seq %= SEQ_SIZE;
387.   }
388.   return seq;
389. }
```