



哈尔滨工业大学  
Harbin Institute of Technology

# 计算机网络 课程实验报告

实验名称	可靠数据传输协议-停等协议-GBN 协议的设计与实现					
姓名	冯开来		院系	计算学部		
班级	1903602		学号	119020125		
任课教师	李全龙		指导教师	李全龙		
实验地点	格物 207		实验时间	11.6		
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:

- 1) 理解可靠数据传输的基本原理
- 2) 理解滑动窗口协议的基本原理
- 3) 掌握停等协议的工作原理
- 4) 掌握 GBN 的工作原理
- 5) 掌握基于UDP 设计并实现一个停等协议的过程与技术
- 6) 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) 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）。
- 3) 模拟引入数据包的丢失，验证所设计协议的有效性。
- 4) 改进所设计的停等和GBN协议，支持双向数据传输；
- 5) 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。
- 6) 将所设计的GBN 协议改进为SR 协议

## 实验过程:

## 1) GBN客户端（停等协议只需将GBN协议窗口中大小改为1）

进入客户端功能之后，当匹配到输入的是“-time”或者“-quit”则作为数据包发送给服务端。如果匹配到输入是“-testgbn [X][Y]”，则进入GBN传输阶段。

```
// [Y]表示 ACK 丢包概率
printTips();
int ret;
int interval = 1; //收到数据包之后返回 ack 的间隔，默认为 1 表示每个都返回 ack，0 或者负数均表示所有的都不返回
char cmd[128];
float packetLossRatio = 0.2; //默认包丢失率 0.2
float ackLossRatio = 0.2; //默认 ACK 丢失率 0.2
//用时间作为随机种子，放在循环的最外面
srand((unsigned)time(NULL));
int recvNum = 0;
while (true) {
    gets_s(buffer);
    ret = sscanf(buffer, "%s%f", &cmd, &packetLossRatio, &ackLossRatio);
    //开始 GBN 测试，使用 GBN 协议实现 UDP 可靠文件传输
    if (!strcmp(cmd, "testgbn")) { ... }
    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();
```

而如果匹配到的是“testgbn [X][Y]”的话，则开始初始化数据包。将“testgbn [X][Y]”这段报文发到服务端（服务端也会有相应函数操作）。在等待服务端回复设置为UDP 为阻塞模式的时候，客户端发起握手连接给服务端，两者进入三次握手阶段（我也不知道为什么UDP传输要三次握手，只是参考代码这么写我就写了）。分别记为case 0和 case 1。

```

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 rcvSeq; // 接收窗口大小为 1, 已确认的序列号
    unsigned short waitSeq; // 等待的序列号
    sendto(socketClient, "-testgbn", strlen("-testgbn") + 1, 0,
        (SOCKADDR*)&addrServer, sizeof(SOCKADDR));
    while (true)
    {
        // 等待 server 回复设置 UDP 为阻塞模式
        recvfrom(socketClient, buffer, BUFFER_LENGTH, 0, (SOCKADDR*)&addrServer, &len);
        // 服务器状态传输完成状态码 255
        if ((unsigned char)buffer[0] == 255) {
            printf("| 数据接收成功 |\n");
            printf("| 接受的数据为 |\n");
            printf("%s\n", rcvPaper);
            break;
        }
        switch (stage) {
            case 0: // 等待握手阶段
                u_code = (unsigned char)buffer[0];
                if ((unsigned char)buffer[0] == 205) { ... }
                break;
            case 1: // 等待接收数据阶段

```

第一个case 0是等待握手阶段。如果服务端发回来的报文的seq匹配的话，就说明准备文件传输，并初始化rcvSeq（上一个收到的序列号）和waitSeq（我希望收到的序列号），和。并转到阶段1。

```

        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;
                rcvSeq = 0;
                waitSeq = 1;
            }

```

第二个case 1是等待接受数据阶段。这里面我们引入随机变量b，模拟数据包丢失的情况。如果没有丢失，那么判断是不是期望的数据包（即waitSeq - Seq是否为0），如果是期望收到的序列号，我们将缓存内容copy下来，并让制作ack（包括我确认收到的序列号、我一共收到了多少packet。

```

        case 1: // 等待接收数据阶段
            seq = (unsigned short)buffer[0];
            // 随机模拟包是否丢失
            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
                memcpy(&rcvPaper[rcvNum * 1024], &buffer[1], strlen(buffer) - 1);
                // printf("%s\n", &buffer[1]);
                rcvNum++;
                buffer[0] = seq;
                buffer[1] = rcvNum;
                buffer[2] = '\0';
                rcvSeq = seq;
            }

```

如果不是我期望收到的数据包，需要返回ack。不过这回返回的ack的数据包里面除了一共收到的packet数量不一样，其他和上面的ack一样。

最后发送ack给服务端，这一小段就完成了。

```

else {
    //如果当前一个包都没有收到，则等待 Seq 为 1 的数据包，不是则不返回 ACK（因为并没有上一个正确的 ACK）
    if (!recvSeq) {
        continue;
    }
    buffer[0] = recvSeq;
    buffer[1] = recvNum;
    buffer[2] = '\0';
}
b = lossInLossRatio(ackLossRatio);
if (b) {
    printf("The ack of %d loss\n", (unsigned char)buffer[0]);
    continue;
}
sendto(socketClient, buffer, 3, 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("recv from client: %s\n", buffer);
}

```

## 2) GBN服务端（停等协议只需将GBN协议窗口中大小改为1）

在服务端运行之后，首先会初始化套接字并绑定端口地址监听，监听到客户端发送的命令执行函数，如果是“-time”和“-quit”则执行返回时间并打包发给客户端和退出程序。

```

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!");
    }
}

```

如果收到的是“-testgbn [X][Y]”的话，就会进入三次握手阶段，阶段过后就是数据发送阶段。

```

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
    //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("Begin to test GBN protocol, please don't abort the process\n");
    //加入了一个握手阶段
    //首先服务器向客户端发送一个 205 大小的状态码（我自己定义的）表示服务器准备好了，可以发送数据
    //客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准备好了，可以接收数据了
    //服务器收到 200 状态码之后，就开始使用 GBN 发送数据了
    printf("Shake hands stage\n");
    int stage = 0;
    bool runFlag = true;
    while (runFlag) {
        switch (stage) {
            case 0: //发送 205 阶段
                buffer[0] = 205;
                sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
                Sleep(100);
                stage = 1;
                break;
            case 1: //等待接收 200 阶段，没有收到则计数器+1，超时则放弃此次“连接”，等待从第一步开始
                waitCount++;
                if (waitCount > 10) {
                    printf("Wait for 200 timeout, abort the process\n");
                    break;
                }
                continue;
            case 2: //传输文件阶段
                //这里应该写文件传输的代码
                break;
        }
    }
}

```

在case1也就是握手阶段，服务端首先调用recvfrom函数，接收到来自客户端的握手报文，如果recvSize小于0则说明该数据包丢失或者有错误，此时waitCount++（即等待时间加一）当waitCount超过3次（或者10次，这个可以改），则会提示超时。如果收到的是我们想要的握手的数据包，那就会初始化curSeq（现在的序列号），curAck（现在确认的ack），waitCount（重置计时器），并转到阶段2。

```

break;
case 1://等待接收 200 阶段，没有收到则计数器+1，超时则放弃此次“连接”，等待从第一步开始
recvSize = recvfrom(sockServer, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrClient), &length);
if (recvSize < 0) {
    ++waitCount;
    if (waitCount > 3) {
        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;
        waitCount = 0;
        stage = 2;
    }
}
break;

```

在case2中也就是数据传输阶段。服务端首先调用函数seqIsAvailable()函数查看是否有空的序列号，如果有空的序列号说明窗口还有剩余，然后将当前序列号+1并封装到buffer[0]，将相应长度的分组放到buffer[1]，开始将数据包发送给客户端。发完之后curSeq++并对SEQ\_SIZE取模得到新的curSeq。（因为取模会得到0，所以会在一开始会对curSeq+1）。

```

break;
case 2://数据传输阶段
if (seqIsAvailable()) {
    //发送给客户端的序列号从 1 开始
    buffer[0] = curSeq + 1;
    ack[curSeq] = FALSE;
    //数据发送的过程中应该判断是否传输完成
    //为简化过程此处并未实现
    int num = 1024;
    if (curSeq + 1 == totalPacket) num = tot - curSeq * 1024;
    memcpy(&buffer[1], data + 1024 * curSeq, 1024);
    printf("send a packet with a seq of %d\n", curSeq);
    sendto(sockServer, buffer, BUFFER_LENGTH, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
    ++curSeq;
    curSeq %= SEQ_SIZE;
    Sleep(500);
}

```

接下来是等待ack，如果没有收到ack，则会返回-1，并且计时器+1，直到waitCount超过10（即10次没有收到），则会重传刚刚的数据包。如果等到了想要的ack，则会重置计时器，并且如果buffer[1]（刚刚在客户端提到，会放入总共收到的packet组数）和packet总组数相等的话，就会打印数据传输完成。

```

400 //等待 ack，若没有收到，则返回-1，计数器+1
401 recvSize = recvfrom(sockServer, buffer, BUFFER_LENGTH, 0, ((SOCKADDR*)&addrClient), &length);
402 if (recvSize < 0) {
403     waitCount++;
404     //10 次等待 ack 则超时重传
405     if (waitCount > 10) {
406         {
407             timeoutHandler();
408             waitCount = 0;
409         }
410     }
411     else {
412         //收到 ack
413         ackHandler(buffer[0]);
414         waitCount = 0;
415         if (buffer[1] == totalPacket) {
416             runFlag = false;
417             printf(" | 数据传输完成 | \n");
418             buffer[0] = 255;
419             sendto(sockServer, buffer, BUFFER_LENGTH, 0, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
420         }
421     }
422     Sleep(500);
423     break;
424 }

```

### 3) 模拟引入丢包

在客户端等待数据接受阶段 (case1)，上文中提到了引入随机变量b，如果b的返回值大于0的话，则说明丢包。我们具体来看看lossInLossRatio()函数，根据丢失率随机生成一个数字，判断是否丢失，丢失在返回TRUE，否则返回FALSE。

```

46  BOOL lossInLossRatio(float lossRatio) {
47      int lossBound = (int)(lossRatio * 100);
48      int r = rand() % 101;
49      if (r <= lossBound) {
50          return TRUE;
51      }
52      return FALSE;
53  }
    
```

在返回后，会重新执行recvfrom()函数，而在服务端，因为收不到ack则会waitCount++ (到一定程度会重发数据包)。

```

        break;
    case 1://等待接收数据阶段
        seq = (unsigned short)buffer[0];
        //随机法模拟包是否丢失
        b = lossInLossRatio(packetLossRatio);
        if (b) {
            printf("The packet with a seq of %d loss\n", seq);
            continue;
        }
    }
    
```

### 4) 支持双向数据传输

本次实验实现双向数据传输的方法是，在运行程序后，由用户选择，该服务端（客户端）执行服务端功能还是客户端功能，所以整体来看，其实就是合并了客户端和服务端的代码。这样客户端能执行服务端代码，也能传输数据。

```

}
//输入 1 运营客户端功能 2 运行服务器功能
void init()
{
    printf("-----\n");
    printf("| 1 --客户端 |\n");
    printf("| 2 --服务器 |\n");
    printf("-----\n");
}
    
```

```

init();
int type;
scanf("%d", &type);
if (type == 1) client();
else {
    if (server(err) == -1) return -1;
}
    
```

## 5) 实现C/S结构的文件传输应用

实现文件传输，服务端首先要将文件内容拷贝到缓存。并计算出总的packet数，并且将每一个组的ack设为TURE（表明该分组还没发送成功即没有收到相应ack）

```
ZeroMemory(buffer, sizeof(buffer));
//将测试数据读入内存
std::ifstream icin;
icin.open("test.txt");
char data[1024 * 113];
ZeroMemory(data, sizeof(data));
icin.read(data, 1024 * 113);
icin.close();
int tot = strlen(data);
totalPacket = (int)ceil((double)strlen(data) / 1024);
int recvSize;
for (int i = 0; i < SEQ_SIZE; ++i) {
    ack[i] = TRUE;
}
```

客户端在收到服务端发送的数据包之后，如果是期望收到的，则将收到的数据缓存到recvPaper中。

```
printf("Recv a packet with a seq of %d\n", seq);
//如果是期望的包，正确接收，正常确认即可
if (!waitSeq - seq) {
    ++waitSeq;
    if (waitSeq == 21) {
        waitSeq = 1;
    }
    //输出数据
memcpy(&recvPaper[recvNum * 1024], &buffer[1], strlen(buffer) - 1);
//printf("%s\n", &buffer[1]);
recvNum++;
}
```

在服务端如果发送了所有的分组，那么将会在最后的分组的状态码设置为255，表明已经发送所有的分组。

```
else {
    //收到 ack
    ackHandler(buffer[0]);
    waitCount = 0;
    if (buffer[1] == totalPacket) {
        runFlag = false;
        printf(" | 数据传输完成 | \n");
        buffer[0] = 255;
        sendto(sockServer, buffer, BUFFER_LENGTH, 0,
            (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
    }
}
```

然后在客户端，如果收到了状态码为255的分组，则表明数据传输已经完成，客户端print所有的数据，即缓存在recvPaper的数据。

```
//等待 server 回复设置 UDP 为阻塞模式
recvfrom(sockClient, buffer, BUFFER_LENGTH, 0, (SOCKADDR*)&addrServer, &len);
//服务器状态传输完成状态码 255
if ((unsigned char)buffer[0] == 255) {
    printf(" | 数据接收成功 | \n");
    printf(" | 接受的数据为 | \n");
    printf("%s\n", recvPaper);
    break;
}
```

## 6) SR客户端

SR客户端和GBN客户端的区别在于GBN只能接受我期望收到的seq，否则就丢弃，而SR客户端可以接受任意的seq（只要在窗口范围内）

```
134 //等待接收数据阶段
135 seq = (unsigned short)buffer[0];
136 //随机法模拟包是否丢失
137 b = lossInLossRatio(packetLossRatio);
138 if (b) {
142 printf("Recv a packet with a seq of %d\n", seq);
143 //接受数据包
144 memcpy(&recvPaper[seq * 1024], &buffer[1], strlen(&buffer[1]));
145 }
```



当然如果收到的序列号已经达到我的recv\_base了那么就窗口基数++。在SR客户端我还模拟了ack丢失的情况，同样是调用lossInLossRatio函数，如果b返回TRUE则说明丢包并重新进行接受数据。

```

        if (recv_base == seq)
            recv_base++;
        /* ... */
        b = lossInLossRatio(ackLossRatio);
        if (b) {
            printf("The ack of %d loss\n", (unsigned char)buffer[0]);
            continue;
        }
    }

```

## 7) SR服务端

SR服务端和GBN服务端的区别在于，GBN期望收到的ack只有一个，而SR可以期望收到多个ack，这个ack范围在send\_base到send\_base + SEQ\_SIZE（当然要小于总的packet数）。在接受ack阶段，如果发生了丢包或者其他错误，程序会遍历期望收到的ack，查看哪些ack是还没有收到（即没有被确认），然后对应分组i的计时器++，超过三次则重新传送分组。

```

//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) {
    for (int i = send_base; i <= min(send_base + SEQ_SIZE, totalPacket - 1); i++)
    {
        if (!ack[i]) {
            waitCount[i]++;
            //3 次等待 ack 则超时重传
            if (waitCount[i] > 3) {
                printf("Packet %d ack Time out\n", i);
                sendPacket(i, sockServer);
                /*memcpy(&buffer[i], data + 1024 * i, 1024);
                printf("send a packet with a seq of %d\n", i);
                sendto(sockServer, buffer, BUFFER_LENGTH, 0,
                    (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
                */
                waitCount[i] = 0;
            }
        }
    }
}

```

当然如果收到了ack，则会对第i个分组的ack进行处理，将他设置为已经接受。

```

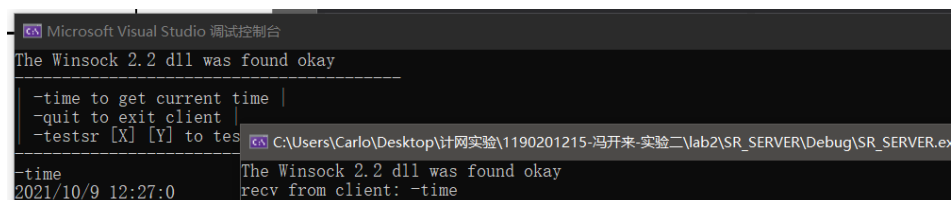
    else {
        //收到 ack
        ackHandler(buffer[0]);
        if (send_base == totalPacket) {

```

## 实验结果：

### 1. 基本功能

#### 1) 获取时间

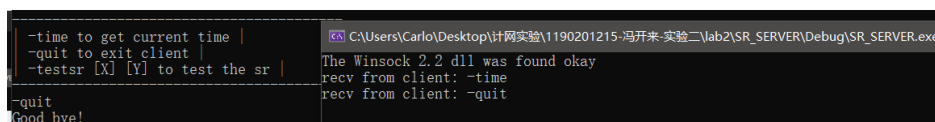


```

Microsoft Visual Studio 调试控制台
The Winsock 2.2 dll was found okay
-quit to exit client |
-testsr [X] [Y] to test the sr |
-quit |
Good bye!
C:\Users\Carlo\Desktop\计网实验\1190201215-冯开来-实验二\lab2\SR_SERVER\Debug\SR_SERVER.exe
The Winsock 2.2 dll was found okay
recv from client: -time

```

#### 2) 退出



```

C:\Users\Carlo\Desktop\计网实验\1190201215-冯开来-实验二\lab2\SR_SERVER\Debug\SR_SERVER.exe
The Winsock 2.2 dll was found okay
recv from client: -quit

```



### 3) 单向传递

The screenshot shows two terminal windows. The left window, titled 'GBN\_CLIENT.exe', displays the client's execution flow: it starts by checking for Winsock 2.2, then prompts the user to select '1' for client mode. After 'run client successfully', it enters a loop where it can quit, test the GBN protocol, or test the file transfer. The right window, titled 'GBN\_SERVER.exe', shows the server's execution flow: it also checks for Winsock 2.2, prompts the user to select '2' for server mode, and then enters a loop where it can quit, test the GBN protocol, or test the file transfer. Both windows show the successful completion of a file transfer process.

## 2. 附加功能

### 1) 模拟数据包丢失

The screenshot shows the 'GBN\_SERVER.exe' console window with a list of network events. A red arrow points to the line 'The packet with a seq of 8 loss', indicating a simulated packet loss. The log shows the server receiving packets, sending acknowledgments, and handling the loss of a packet, which triggers a 'timer out error' and a retransmission of the packet with sequence number 8.

### 2) 双向数据传递

服务端选择客户端功能，客户端选择服务端功能

The screenshot shows two terminal windows. The left window, 'GBN\_CLIENT.exe', shows the client selecting '2' for server mode. The right window, 'GBN\_SERVER.exe', shows the server selecting '1' for client mode. Both windows show the successful completion of a file transfer process, demonstrating bidirectional data transfer.

客户端给服务端传输数据

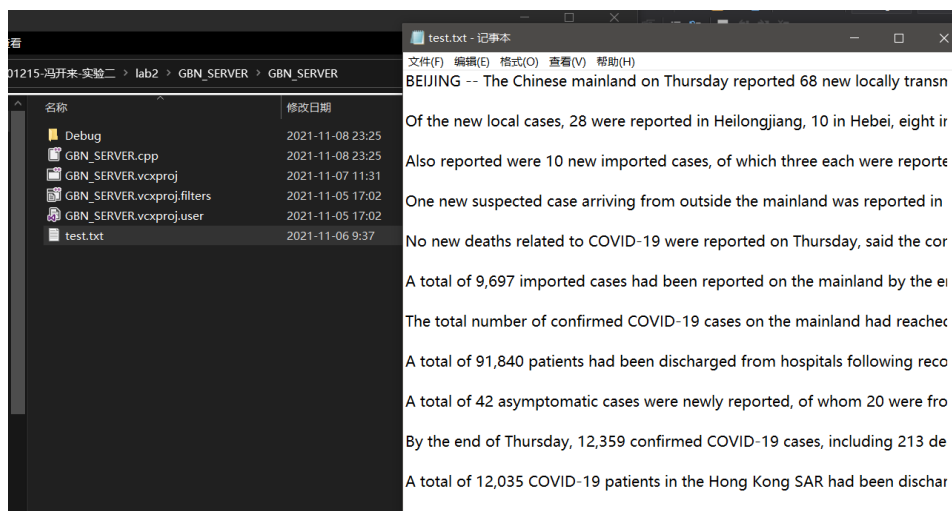
The screenshot shows two terminal windows. The left window, 'GBN\_CLIENT.exe', shows the client selecting '2' for server mode. The right window, 'GBN\_SERVER.exe', shows the server selecting '1' for client mode. Both windows show the successful completion of a file transfer process, demonstrating data transfer from the client to the server.

- 1) 停等协议数据分组格式
- 2) 停等协议两端程序流程图

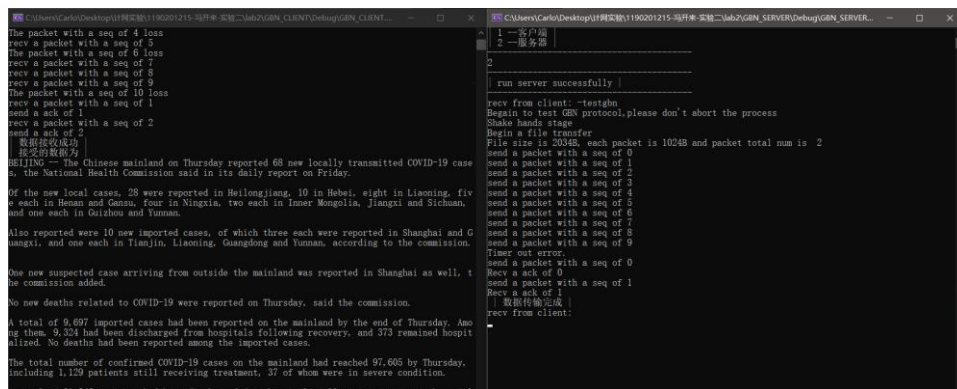
### 3) 文件传输

这里选择的是打开“test.txt”文件，在服务端传输成功后，客户端打印出来。

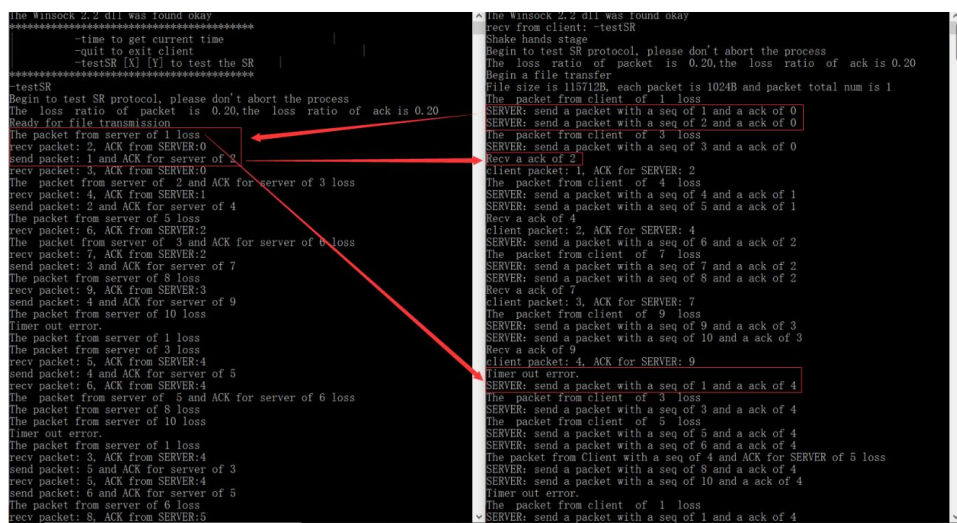
文件内容：



传输成功，客户端print结果：



### 4) SR协议



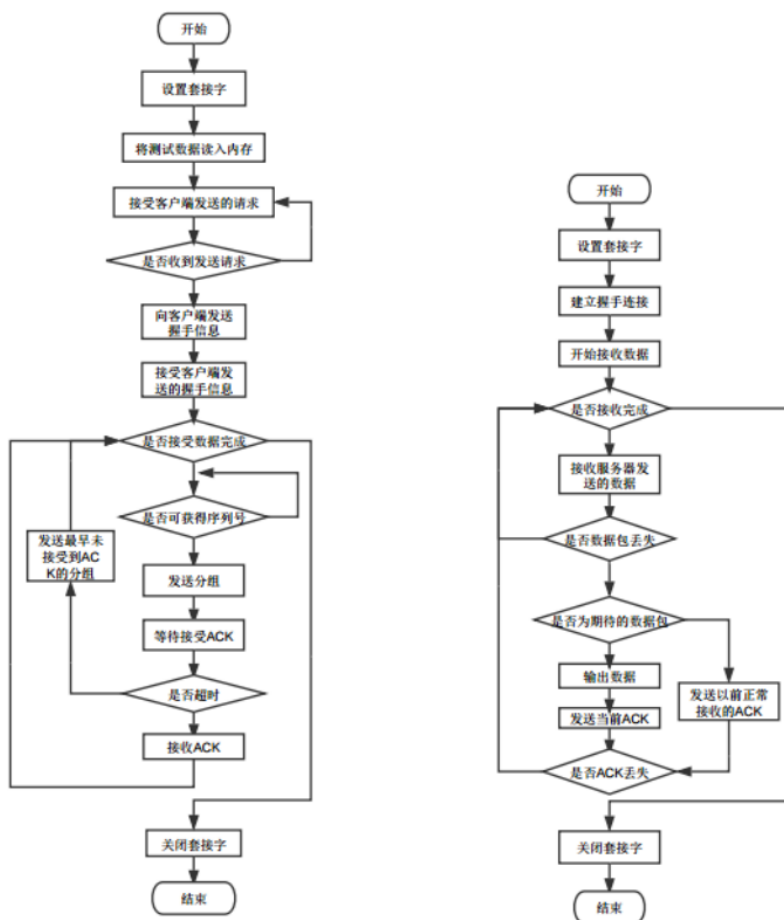
### 1) GBN (停等) 协议数据分组格式、确认分组格式、各个域的作用

Seq	Data	0
-----	------	---

最后一个字节放入EOF0，表示结尾。

ACK	0
-----	---

### 2) GBN (停等) 协议两端程序流程图



左图为服务端流程，右图为客户端流程

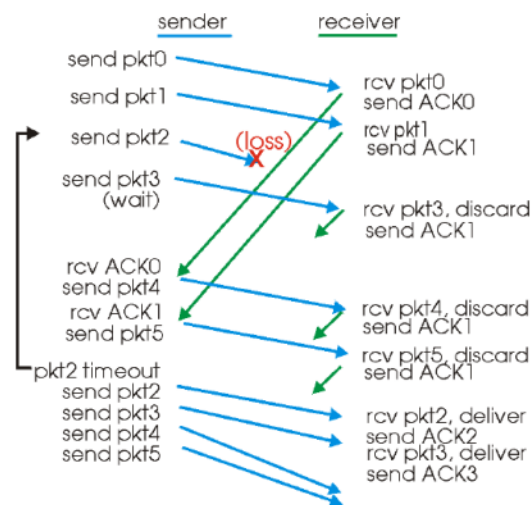
### 3) GBN（停等）协议典型交互过程

#### 服务端：

- 1.开启端口访问，等待客户端发起请求。
- 2.收到客户请求消息后，读取请求数据的内容，构造数据包并准备发送，开启新的发送数据报端口，在新端口上向客户端进行发送数据报同时开启计时器。
- 3.接受客户端返回的ACK确认数据报，根据序列号判断选择滑动窗口发送帧还是重传操作。

#### 客户端：

- 1.向服务端发送请求，然后等待服务端反馈；
- 2.接收到服务端数据报后进行判断是否为当前欲接收的数据报，是则接收，不是则丢弃，然后返回ACK确认。



典型交互中，出现的情况有发送的数据报丢失、超时或收到重复ACK导致重传；

### 4) 数据分组丢失验证模拟方法

实验中采用对某数取余，当余数为0时则视为该数据分组丢失，不返回确认ACK；客户端对收到的数据帧个数计数，当对某数取余后余数为0时则不返回确认ACK从而模拟丢包现象；

```

1 // *****
2
3
4
5
6 BOOL lossInLossRatio(float lossRatio) {
7     int lossBound = (int)(lossRatio * 100);
8     int r = rand() % 101;
9     if (r <= lossBound) {
10         return TRUE;
11     }
12     return FALSE;
13 }

```

### 5) 程序实现的主要类（或函数）及其主要作用

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

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

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

void printTips();//打印提示信息

BOOL lossInLossRatio(float lossRatio);//根据丢失率随机生成一个数字，判断是否丢失，丢失则返回 TRUE，否则返回 FALSE

```
void getCurTime(char* ptime);//获取当前系统时间，结果存入 ptime 中
```

6) UDP编程主要特点

- 1) udp发送和接收没有缓冲区，发送和接收都是整包，自动保持包的边界
- 2) udp包的发送和接收不保证一定成功，不保证按正确顺序抵达
- 3) 在接收udp包时，如果接收包时给定的buffer太小的话，会发生异常，要捕获异常，相应调整buffer的大小，和给出反馈信息。
- 4) 如果不允许丢包的情况出现的话，要有重发机制来保证，如：每发一条信息，只有收到正确的ack的时候，才证明成功

7) 实验验证结果

见上文

8) 详细注释的源程序：

见压缩包附件

心得体会：

通过本次实验，更加理解了 GBN 协议和 SR 协议的区别，以及通信过程中，从一端传送数据到另一端接受，返回 ACK，再到一端接受 ACK，必要时重传的逻辑过程，也更加了解到窗口设计的巧妙之处。