

# 实验 3-2 实现基于滑动窗口的流量控制机制

1813800 沈哲

## 目录

<b>1 实验要求</b>	<b>2</b>
<b>2 功能实现</b>	<b>2</b>
2.1 协议设计 . . . . .	2
2.2 建立与断开连接 . . . . .	3
2.2.1 建立连接 . . . . .	3
2.2.2 断开连接 . . . . .	3
2.3 差错检测 . . . . .	4
2.4 确认重传 . . . . .	4
2.5 基于滑动窗口的流量控制机制 . . . . .	4
2.5.1 发送端 . . . . .	5
2.5.2 接收端 . . . . .	6
2.6 其他设计 . . . . .	6
2.6.1 定时器的设计 . . . . .	6
2.6.2 随机丢包和延时的设计 . . . . .	6
2.6.3 快速重传和窗口加速滑动 . . . . .	6
2.6.4 测试方法 . . . . .	6
<b>3 代码分析 (仅展示部分核心代码)</b>	<b>7</b>
3.1 公共头文件定义 . . . . .	7
3.2 基本函数 (不传入参数, 直接执行完成相应功能) . . . . .	8
3.2.1 makePackage . . . . .	8
3.2.2 doCheckSum . . . . .	8
3.2.3 Check_expected_ACK(发送端使用) . . . . .	8
3.2.4 CheckSEQ_EXPECTED(接收端使用) . . . . .	9
3.2.5 IsACK,IsFIN,IsSYN . . . . .	9
3.3 发送端代码 . . . . .	9
3.3.1 窗口滑动 . . . . .	10

3.3.2 接收 ACK . . . . .	11
3.3.3 超时重传 . . . . .	12
3.4 接收端代码 . . . . .	13
3.4.1 接收按序的分组并写入文件 . . . . .	13
3.4.2 接收到失序的分组, 直接丢弃 . . . . .	13
3.5 断开连接的其他辅助处理 . . . . .	14
3.5.1 发送端可能收不到接收端回复的 FIN . . . . .	14
<b>4 结果展示</b>	<b>15</b>
<b>5 总结反思</b>	<b>16</b>
5.1 编程实现 . . . . .	16
5.2 协议理解 . . . . .	16

## 1 实验要求

利用数据报套接字在用户空间实现面向连接的可靠数据传输, 功能包括: 建立连接、差错检测、确认重传。在任务 3-1 的基础上, 将停等机制改成基于滑动窗口的流量控制机制, 采用固定窗口大小, 支持累积确认, 完成给定测试文件的传输。

要求实现单向传输。对于每一个任务要求给出详细的协议设计。完成给定测试文件的传输, 显示传输时间和平均吞吐率。

## 2 功能实现

### 2.1 协议设计

UDP 是传输层中面向无连接的协议, 在编程上服务端和客户端是没有区别的, 本实验实现从客户端(发送端)到服务端(接收端)的传输。本实验参考了 TCP 协议来设计数据报相应字段, 数据报分为两部分——头部和数据部分, 相较于上次实验作了一些改动, 使得编程更方便。

实验 3-2 进行了较大改动, 协议更加完善清晰(接近 TCP), 数据报的结构更容易处理(使用结构体而不是字符数组, 虽然字符数组更简单更易于传送, 但编程操作上可能更加困难)。实际上就是把原来的字符数组(头部 + 数据部分)改写成结构体 PACKAGE(仍是 10 字节的头部 + 数据部分, 占用字节数不变, 但操作变得简单, 可以直接对结构体属性赋值)。属性类型全部改用 unsigned short, 表示 0-65535 范围的无符号整数。

TCP 是字节流传送(序列号每次增加的值为字节数), 本协议简单地使用序列号递增模式。即 SEQ 每次递增 1(等于接收到的 ACKnum, 可以对 65535 取模), ACKnum 是对方序列号加 1(表示

期待对方下次发送的序列号): send\_SEQ=recv\_ACKnum,send\_ACKnum=recv\_SEQ。本实验中 SEQ 主要在发送端使用, ACKnum 主要在接收端使用。

数据部分占 65300 字节。头部包括: 序列号、确认序号、检验和字段、标志位字段、长度字段。每一部分占 2 字节, 共占 10 字节。

Header : 10 bytes
2 bytes   2 bytes   2 bytes   2 bytes   2 bytes
SEQ   ACKnum   CheckSum   ID   Length

SEQ 是序列号, 范围是 0 到 65535, 主要在发送端使用。

ACKnum 是确认序列号, 主要在接收端使用, 接收端回复的 ACKnum 等于发送端的 SEQ+1, (代表下次期望收到的发送端序列号)。

CheckSum 是检验和字段, 发送端制作分组时计算检验和填入, 接收端对收到的分组进行差错检验。

ID 是多位标志位, 包括 SYN (连接建立标志位)、ACK (确认标志位)、FIN (断开连接标志位)。

ID : 2 bytes
FIN   ACK   SYN

## 2.2 建立与断开连接

假设 a 为客户端 (发送) b 为服务端 (接收), 下面是交互过程的简单描述。

SYN 和 FIN 只在开始和结束时使用。

### 2.2.1 建立连接

假设初始 X=0,Y=0, 这里的序列号从 0 开始, 也可使用随机数取模产生随机序列号。

a->b: SYN=1 ACK=0 SEQ=X=0 ACKnum=0
b->a: SYN=1 ACK=1 SEQ=Y=0 ACKnum=X+1=1
a->b: SYN=0 ACK=1 SEQ=1 ACKnum=1 第三次握手的同时发送数据

### 2.2.2 断开连接

假设初始 X=1,Y=1, 模仿 TCP 四次挥手 (<https://www.jianshu.com/p/cd801d1b3147>) 但这里三次交互就够, a 不需要第二次等待, b 收到 FIN 后立即发送 FIN。

a->b: FIN=1 ACK=1 SEQ=X=1 ACKnum=1
b->a: FIN=1 ACK=1 SEQ=Y=1 ACKnum=2
a->b: FIN=0 ACK=1 SEQ=2 ACKnum=2

断开连接时, a 先发送 FIN 信号。b 接收到 FIN 之后, 也发送 FIN (让对方知道自己知道了要断开连接)。a 发送 FIN, 再收到 FIN 之后, 发送 ACK (这里对方收不到可以重传), 然后退出。b 发送 FIN 并且收到 ACK 之后退出。

## 2.3 差错检测

采用 UDP 校验和计算方法。发送端在发送分组时进行 UDP 校验和计算，结果写入 CheckSum 位。接收端接收到分组后也进行校验和计算，结果为 0xFFFF 则无错误。

计算校验和时，本实验没有添加伪首部，只是对头部和数据部分计算。首先需要把 char 型的字符数组（char 类型占 1 字节）转为 unsigned short 型的数组（unsigned short 类型占 2 字节），采用反码求和方法（加法的溢出位需要回卷），需要注意头部占 10 字节，数据部分的字节数若为奇数，需要对最后一个字节进行单独处理（可以直接拷贝到 unnsiged short 类型的数组，这与 C++ 在 x86 下属于小端编址有关）。

## 2.4 确认重传

在上次实验的停等协议中，发送端收到确认序号 ACKnum 后再次发送下一个分组， $SEQ = ACKnum$ 。接收端进行累积确认。如果发送端发送的分组丢失，接收端一直等待接收不发送 ACK，或者接收端发送的 ACK 丢失，都会导致发送端收不到 ACKnum，这时发送端就重传窗口内还未确认的所有分组。

## 2.5 基于滑动窗口的流量控制机制

本实验采用 GBN 方法，大小窗口固定为 20，注意 GBN 方法的窗口大小需要小于序列号数量即  $GBN \leq 2^n - 1$ ，SR 方法窗口大小需要满足  $SR \leq 2^{n-1}$ 。

滑动窗口的思想可以简单理解为，发送端一次发送多个分组。接收端依旧使用累积确认，按序接收，当收到失序的分组时直接丢弃，并发送原来的 ACKnum（期待按序收到的分组序号）。发送端收到 ACK 后，滑动窗口。若发生超时事件或收到重复的与之前相同的 ACK，代表接收端没有收到具有 ACKnum 序号的分组，需要进行重传。

### 2.5.1 发送端

发送端需要响应三种类型的事件：

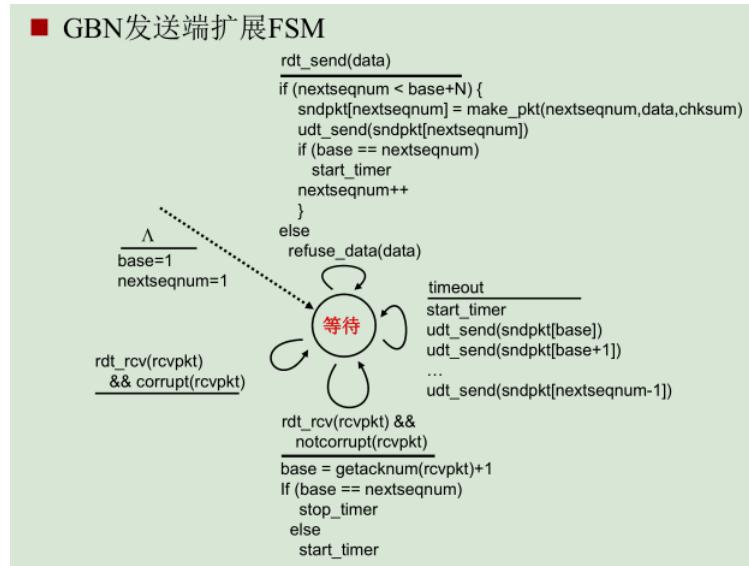
1. 上层调用。本实验使用的是同步机制，仅当窗口不满时才读取文件进行打包传输。
2. 收到一个 ACK (ACKnum 为 n)。代表接收端已经接收分组 n 以及之前的分组，这就是累积确认，这一点很重要，方便之后发送端窗口的加速滑动。
3. 超时事件。本实验使用的是非阻塞方式，通过设置发送和接收的超时时间，也能很好实现“定时器”的功能，并且似乎能产生加速的效果。按照 GBN 的思想，例如发送端发送多个分组，序号从 1 到 10，分组 10 发送最晚，接收端最晚接收，假设只有分组 10 损坏，只有等接收端发送完毕 ACK9 之后发送端才能知道第 10 个分组损坏，需要重传。但在编程中发现，在本实验中不使用定时器（一个原因包括可能需要涉及线程较为复杂）的非阻塞方式中，发送端可能在发送完这 10 个分

组后，先收到分组 9 的 ACK 而不是分组 1 的 ACK（注意因为累积确认的原则，接收端肯定发送了 ACK2 到 ACK10），但发送端先收到 ACK10，这窗口就可以加速滑动，之前的 ACK 就可以忽略了。

发送端要负责维护一个发送分组的缓存便于之后可能的重传。

可以使用队列，队列中的元素代表着当前窗口——已发送但还未确认的分组。发送的分组都加入队列，发送后经确认的分组可以从队列移出。重传时只需重传队列中的所有分组。

下图是发送端的扩展状态机：



## 2.5.2 接收端

接收端的处理很简单。如果收到按序的分组 n（即上次收到的分组序号是 n-1），则接收并回复 ACKnum=n+1，接收到失序的分组，直接丢弃，并发送原来的 ACKnum（期待按序接收的分组序号）。

## 2.6 其他设计

### 2.6.1 定时器的设计

采用非阻塞模式，设置一个超时时间（设置为 50ms），超过此时间无响应就返回一个值 (-1)，这样编程更方便，也实现了“定时器”的功能。经过测试，超时未响应 recvfrom 函数的返回值为 -1，错误类型 WSAGetLastError() 的返回值为 10060。

```
// 设置发送超时
setsockopt(socket_client,SOL_SOCKET,SO_SNDTIMEO, (char *)&nNetTimeout,sizeof(
    int));
// 设置接收超时
setsockopt(socket_client,SOL_SOCKET,SO_RCVTIMEO, (char *)&nNetTimeout,sizeof(
    int));
```

## 2.6.2 随机丢包和延时的设计

使用随机数取模的方法，随机产生丢包，包括发送端丢弃数据包、丢弃重传的数据包，接收端丢弃 ACK 等等。

## 2.6.3 快速重传和窗口加速滑动

实现了快速重传，发送端维护一个变量，代表期望接收的 ACK 序号，如果发送端收到了三次相同的冗余 ACK，就执行快速重传，立即重传当前滑动窗口内的所有分组。如果收到了超前的 ACKnum，比如发送了分组 1、2、3，期待收到 ACKnum=2、3、4，如果先收到的是 4，就可以加速滑动窗口，而不是每次滑动一位。这种情况在实际运行中是会发生的。

## 2.6.4 测试方法

将输入输出重定向到文件，实现运行日志的保存。在 windows 环境下使用批处理文件实现快捷测试。

```
批处理命令：  
test_server < input_server >> output_server & exit  
test_client < input_client >> output_client & exit  
python 文件作用是将输入参数（窗口大小、文件序号）写入文件并运行批处理命令。  
运行以下命令或将此命令写入bat文件运行，即可设置滑动窗口大小为20，传输文件1：  
python s.py 20 1
```

## 3 代码分析(仅展示部分核心代码)

### 3.1 公共头文件定义

```
//设置IP和端口号等  
#define server_Port 1001  
#define server_IP "127.0.0.1"  
#define client_Port 1002  
#define client_IP "127.0.0.1"  
//rand()%RAND_MOD_NUM 对随机数取模来决定发送还是丢弃分组  
int RAND_MOD_NUM=15;  
//非阻塞模式的超时时间  
int nNetTimeout=50;//毫秒  
//缓冲区大小  
#define BUF_LEN 65310//比2**16小一点  
#define HEADER_LEN 10//头部长度  
#define DATA_LEN 65300  
typedef unsigned short ushort;
```

```

/*
头部的设计
Header:10 bytes
| 2 bytes | 2 bytes | 2 bytes | 2 bytes | 2 bytes |
|   SEQ    | ACKnum  | CheckSum |      ID   | Length   |

ID:2 bytes
| FIN     | ACK     | SYN     |
*/
struct PACKAGE
{
    ushort SEQ;//序列号 2bytes,16bits 0-65535
    ushort ACKnum;//确认序号
    ushort CheckSum;//检验和
    ushort ID;//多个标志位
    ushort Length;//数据长度，这是基本固定的
    char data[DATA_LEN];
};

//各个标志位
#define SYN 0x1//建立连接
#define ACK 0x2//确认
#define FIN 0x4//断开连接

```

## 3.2 基本函数（不传入参数，直接执行完成相应功能）

### 3.2.1 makePackage

用于制作分组的头部，发送端需要设置校验和、发送序列号、数据长度。接收端需要设置确认序号。

```

sendbuf.Length = length;
sendbuf.SEQ =(sendbuf.SEQ+1)%65536;
sendbuf.CheckSum = 0;
memcpy(&sendbuf, &header, HEADER_LEN);

```

注意建立连接时填充 SYN 字段，结束连接时填充 FIN 字段。发送端在读取数据后，需要再次填充 CheckSum 字段。

### 3.2.2 doCheckSum

计算 UDP 校验和的函数。发送端计算校验和填入 CheckSum 字段。接收端计算校验和判断是否等于 0xFFFF，不等于 0xFFFF 则数据出错。

```
//先将char型发送缓冲区转换成为16位的unsigned short型数据
```

```

//需要分情况，考虑缓冲区长度
array = new unsigned short[count];
for (int i = 0; i < count; i++)
{
    memcpy(&array[i], &sendbuf[i * 2], 2);
}
//计算校验和
while (count--)
{
    sum += array[i++];
    sum = (sum >> 16) + (sum & 0xFFFF);
}
sendbuf.CheckSum = ~sum;

```

### 3.2.3 Check\_expected\_ACK(发送端使用)

发送端检查接收的 ACKnum 是否是期望且按序的（即等于最早未确认的分组序号加 1）。如果等于期望的 ACKnum，窗口滑动一位。如果小于期望的 ACKnum，直接忽略即可。如果大于期望的 ACKnum，则窗口可以加速滑动。每次收到一个按序的 ACKnum 后，expected\_ack 进行更新： $expected\_ack = recvbuf.ACKnum + 1;$ 。

### 3.2.4 CheckSEQ\_EXPECTED(接收端使用)

接收端检查接收到的分组是否按序，比较接收到的分组序号是否等于刚刚发送的 ACKnum 即可。

### 3.2.5 IsACK,IsFIN,IsSYN

检查这些标志位是否置位。

```

bool IsFIN()
{
    if ((recvbuf.ID & FIN) != 0)
        return true;
    else
        return false;
}

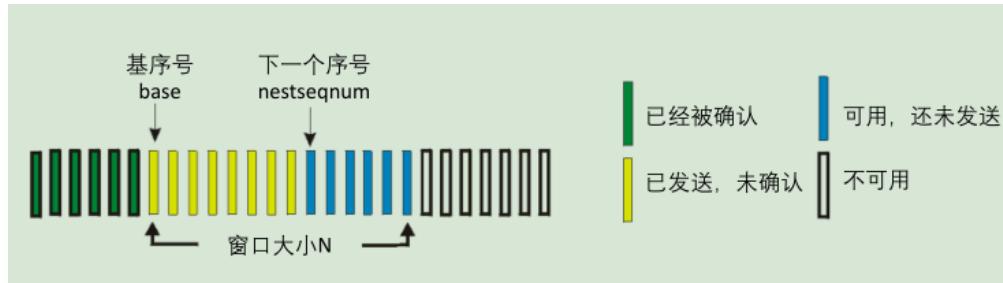
```

## 3.3 发送端代码

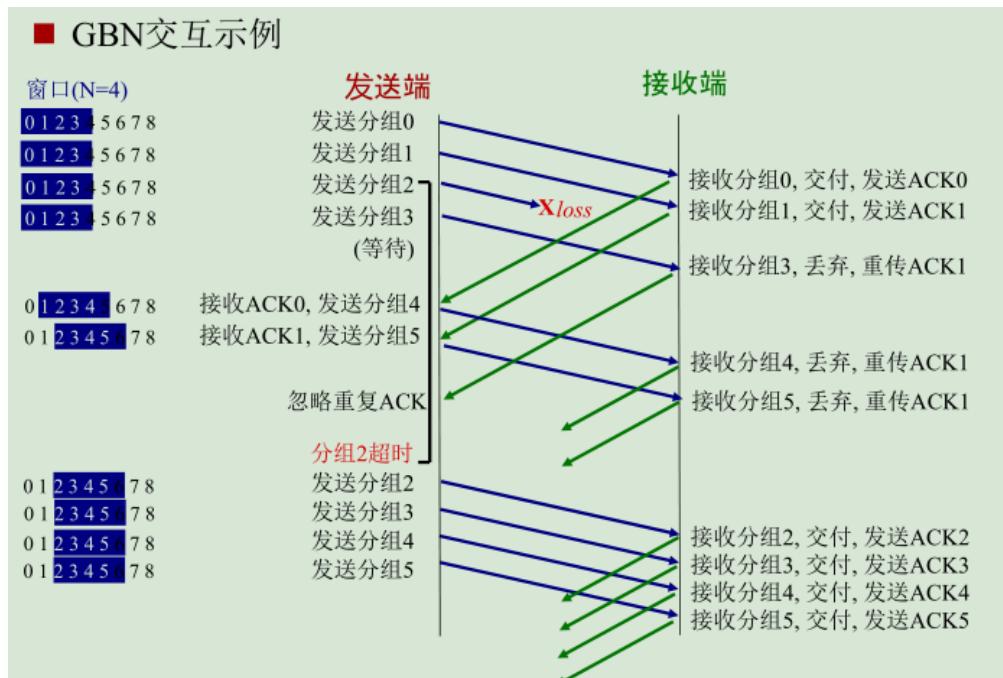
先是 WinSocket 的初始化，创建 socket，设置超时时间等操作。

第一步发送包含 SYN 的分组，准备建立连接。连接建立后，先发送文件名。之后开始读文件（每次读取数据缓冲区大小的数据量）。制作分组（包括序列号每次取模递增，数据长度，待装入数据后计算填充校验和）。发送分组，可能会随机丢弃。

发送端维护一个队列 `Send_Buf_Queue` 用于缓存发送的分组。滑动窗口具有 `p_Base`、`p_NextSeqNum` 两个指针，如下图所示，分别指向最早的发送未确认分组和即将发送的分组。在发送分组时，将分组加入发送缓冲队列，在接收到 ACK 后，将队头元素（已确认的分组）出队列。



GBN 的交互主要包括窗口滑动、接收 ACK、超时重传三种情况。



### 3.3.1 窗口滑动

初始时，`base` 和 `nextseqnum` 都为 1，先发送多个分组。收到期待的 `ACKnum` 后，`base` 加 1，收到超前的 `ACKnum` 后，`base` 增加值大于 1。`base` 变化后，`p_NextSeqNum < p_Base + WINDOW_WIDTH` 条件就满足了，进入 while 循环，窗口实现滑动。

```
/***** 读取、发送 *****
* 0 1 2 3 4 5 6 7 8 9
```

```

*      |          |      n=5(1 2 3 4 5)
*      base      next
*  */
while((p_NextSeqNum<p_Base+WINDOW_WIDTH)&&read_flag)//这里是while!!!! 不是if
    ! ! !
{
    if(p_NextSeqNum==SEQ_SIZE)
        {//实现循环，这样也可以很简单
            p_Base-=p_Base;p_NextSeqNum-=p_Base;
        }
    file.read((char*)&sendbuf.data, DATA_LEN);
    if(file.gcount() < DATA_LEN)
    {
        //读取结束了
        sendbuf.ID |= FIN;//或操作使得FIN位置为1
        read_flag=0;//不再读取了
    }
    makePackage(file.gcount());
    //设置校验和并填充
    doChecksum(file.gcount());
    //加入发送缓冲队列
    PACKAGE t1=sendbuf;
    Send_Buf_Queue.push(t1);
    //发送数据包，随机丢弃
    if(rand()%RAND_MOD_NUM!=0)//即遇到rand()%RAND_MOD_NUM==0丢弃
    {
        //宏替换语句，包括发送和延时sendto(socket_client, (char*)&sendbuf, BUF
        _LEN, 0, (SOCKADDR*)&server_addr, sizeof(SOCKADDR));Sleep(SEND_TIME
        _DELAY)
        CLIENT_SEND;
        cout<<"send seq: "<<sendbuf.SEQ<<endl;//cout<<"发送第 "<<sendbuf.SEQ
        <<" 个数据包!!!"<<endl;
    }
    else
    {
        cout<<"*****随机丢弃数据包 "<<sendbuf.SEQ<<"*****"<<endl;
    }

    p_NextSeqNum++;
}

```

### 3.3.2 接收 ACK

这里根据收到的 ACKnum 和期待的 ACKnum 不同，又分为三种情况。

1.recvbuf.ACKnum==expected\_ack

直接移动 base，发送端缓存队列队头元素出队。

```
cout << "successfully trans " << recvbuf.ACKnum-1 << endl;
trans_bytes+=recvbuf.Length;
p_Base=recvbuf.ACKnum;
Send_Buf_Queue.pop();
```

2.recvbuf.ACKnum<expected\_ack

记录一个当前收到的 ACKnum，当连续三次收到冗余的 ACK 时，就执行快速重传。

```
if(recvbuf.ACKnum==cur_ack)
{
    dup_ack_num++;
    cout << "收到之前的ACKnum且重复！！！：" << recvbuf.ACKnum << ""<< endl;
}
else
{
    dup_ack_num=0;
    cout << "收到之前的ACKnum：" << recvbuf.ACKnum << "！不执行动作即可"<<
        endl;
    cur_ack=recvbuf.ACKnum;
}
if(dup_ack_num==2)
{
    //快速重传
    dup_ack_num=0;
    cout << "*****执行快速重传！！！*****" << endl;
    goto LABEL_TRANS AGAIN;
}
```

3.recvbuf.ACKnum>expected\_ack

收到超前的 ACKnum，base 多移动几位，窗口能加速滑动。

```
cout << "收到超前的ACKnum，可以加速滑动！！！ ACKnum：" << recvbuf.ACKnum << endl;
p_Base=recvbuf.ACKnum;
//p_Base 移动
for (int i = 0; i < recvbuf.ACKnum-expected_ack+1; i++)
{
    Send_Buf_Queue.pop();
}
expected_ack=recvbuf.ACKnum+1;
trans_bytes+=(recvbuf.ACKnum-expected_ack+1)*DATA_LEN;
```

### 3.3.3 超时重传

超时未收到 ACK，重传窗口内的所有分组，只需要把发送缓冲队列里的分组依次出队列、发送、再加入队列。

```
/* base--next-1
* 0 1 2 3 4 5 6 7 8
*           |       | 已发送 1 2 3 4 5
* 收到 ack2，收不到 ack3，可能情况：1. 对方未收到 seq3，期待收到 seq3 2. 对方收到了
* seq3，但是 ack3 丢失
* 都执行重传，重传 2 3 4 5 6
*/
for (int i = p_Base; i < p_NextSeqNum; i++)
{
    sendbuf=Send_Buf_Queue.front();
    if(rand()%RAND_MOD_NUM!=0)
    {
        CLIENT_SEND;
    }
    else
    {
        cout<<"*****随机丢弃重传的数据包！！！！！
*****"<<endl;
    }
    PACKAGE t1=Send_Buf_Queue.front();
    Send_Buf_Queue.push(t1);
    Send_Buf_Queue.pop();
}
cout << "超时未接收到 ACK，重传 "<<p_NextSeqNum<<"-"<<p_Base<<" == "<<p_
NextSeqNum-p_Base<<" 个数据包！！！" << endl;
```

## 3.4 接收端代码

和发送端相同，建立连接后接收文件名。之后开始接收分组，写入文件，发送 ACK 确认分组（也进行随机丢弃）。

CheckSum 函数进行校验和计算，CheckSEQ\_EXPECTED 确保接收到的是按序期望的不重复的分组，否则丢弃即可，IsFIN 判断是否传输完成。

### 3.4.1 接收按序的分组并写入文件

```
cout << "successfully recv " << recvbuf.SEQ << endl;
trans_bytes += recvbuf.Length;
file.write(recvbuf.data, recvbuf.Length);
```

```

makePackage();
if(rand()%RAND_MOD_NUM!=0)
{
    SERVER_SEND;
}
else
{
    cout<<"*****随机丢弃ACK包！！！！！
*****"<<endl;
}

```

### 3.4.2 接收到失序的分组, 直接丢弃

```

/*****************收到失序(超前)的包
*****
* 比如收到了seq5, 回复acknum==6, 期望的是seq6
* 结果收到seq8, 直接丢弃
* package.ACKnum不变
*/
cout<<"接收到失序的数据包 "<<recvbuf.SEQ<<" ,丢弃, 发送原ACKnum: "<<sendbuf.
    ACKnum<<"!!!!"<<endl;
if(rand()%RAND_MOD_NUM!=0)//即遇到rand()%RAND_MOD_NUM==0丢弃
{
    SERVER_SEND;
}
else
{
    cout<<"*****随机丢弃ACK包(收到失序数据包时的ACK
)！！！！！*****"<<endl;
}

```

## 3.5 断开连接的其他辅助处理

### 3.5.1 发送端可能收不到接收端回复的 FIN

发送端在发送最后一个分组的同时发送 FIN, 但有可能收不到接收端的 FIN, 这时 base 和 nextseqnum 已经相等, 可以根据此来决定终止连接。

```

if((p_Base==p_NextSeqNum&&p_Base!=1))
{
    cout<<"已经结束了！！！"<<endl;
    goto LABEL_Complete;
}

```

另外也可能遇到接收端退出过快，发送端甚至收不到 ACKnum。注意接收端只有接收完毕并且收到 FIN 和 ACK 才会退出。可以判断发送端重传次数，如果多次重传相同的分组，则有可能是接收端已经退出了，发送端退出即可。

```
if(p_Base==T_a&&p_NextSeqNum==T_b)
{
    dup_trans_num++;
}
else
{
    dup_trans_num=0;
    T_a=p_Base;T_b=p_NextSeqNum;
}
if(dup_trans_num==20)
{
    cout<<"对方无应答，应该是退出了，结束！！"<<endl;
    goto LABEL_Complete;
}
```

对于接收端，必须收到发送端的 FIN 和 ACK 才能退出。

```
/*发送端发来了FIN，接收端再发一次FIN，等到发送端知道了，发送ACK（发送端只有这时候发送ACK），之后再退出*/
while(recvbuf.ID&ACK==0)
{
    cout<<"send FIN!"<<" ";
    SERVER_SEND;//含有FIN
    do
    {
        recv_Ret = recvfrom(socket_server, (char*)&recvbuf, BUF_LEN, 0,
                            SOCKADDR *)&client_addr, &recv_para_len);
    } while (recv_Ret < 0);
}
```

## 4 结果展示

程序可以连续发送任意类型的文件，也可以选择输入窗口大小、文件名等等，能够实现文件的无损正确传输。超时时间越短（这里是 10ms），缓冲区越大（这里是比 2 的 16 次方小一些，不能超过，取 65310），文件发送速度越快。

以下是窗口大小为 20 的测试结果。

从下图可以看出，发生过发送端丢弃分组和接收端丢弃 ACK 的情况，也发生了窗口加速滑动。

```

3.2源代码 > 》 output_client
1 connect!
2 请输入要传输的文件序号 (1 2 3 4 5) : 1
3 *****start transport*****
4 send seq: 1
5 send seq: 2
6 send seq: 3
7 send seq: 4
8 send seq: 5
9 send seq: 6
10 send seq: 7
11 send seq: 8
12 send seq: 9
13 send seq: 10
14 send seq: 11
15 send seq: 12
16 send seq: 13
17 send seq: 14
18 *****随机丢弃数据包 15*****
19 send seq: 16
20 send seq: 17
21 send seq: 18
22 send seq: 19
23 send seq: 20
24 successfully trans 1
25 send seq: 21
26 -----收到超前的ACKnum,可以加速滑动,开心!!!! ACKnum: 15
27 send seq: 22
28 send seq: 23
29 send seq: 24
30 send seq: 25
31 send seq: 26
32 send seq: 27
33 send seq: 28
34 send seq: 29

3.2源代码 > 》 output_server
1 connect!!!
2 接收到文件名: 1.jpg
3 successfully recv 1
4 successfully recv 2
5 successfully recv 3
6 successfully recv 4
7 successfully recv 5
8 successfully recv 6
9 successfully recv 7
10 successfully recv 8
11 successfully recv 9
12 successfully recv 10
13 successfully recv 11
14 successfully recv 12
15 successfully recv 13
16 successfully recv 14
17 接收到失序的数据包 16 ,丢弃, 发送原ACKnum: 15!!!
18 *****随机丢弃ACK包(收到失序数据包时的ACK)!!!!
19 接收到失序的数据包 17 ,丢弃, 发送原ACKnum: 15!!!
20 接收到失序的数据包 18 ,丢弃, 发送原ACKnum: 15!!!
21 接收到失序的数据包 19 ,丢弃, 发送原ACKnum: 15!!!
22 接收到失序的数据包 20 ,丢弃, 发送原ACKnum: 15!!!
23 接收到失序的数据包 21 ,丢弃, 发送原ACKnum: 15!!!
24 接收到失序的数据包 22 ,丢弃, 发送原ACKnum: 15!!!
25 接收到失序的数据包 23 ,丢弃, 发送原ACKnum: 15!!!
26 接收到失序的数据包 24 ,丢弃, 发送原ACKnum: 15!!!
27 接收到失序的数据包 25 ,丢弃, 发送原ACKnum: 15!!!
28 接收到失序的数据包 26 ,丢弃, 发送原ACKnum: 15!!!
29 接收到失序的数据包 27 ,丢弃, 发送原ACKnum: 15!!!
30 接收到失序的数据包 28 ,丢弃, 发送原ACKnum: 15!!!
31 接收到失序的数据包 29 ,丢弃, 发送原ACKnum: 15!!!
32 no data successfully recv 15
33 successfully recv 16
34 successfully recv 17

```

下图是超时重传，可以看到第一次重传从分组 15 开始，之后分组 15 得到确认。但发送端随即丢弃了分组 16，导致第二次重传。这时也发生了接收端退出过早的情况，可以通过检查发送端是否多次重传相同分组来确认。

```

send seq: 25
send seq: 26
send seq: 27
send seq: 28
send seq: 29
收到之前的ACKnum: 15! 不执行动作即回
超时未接收到ACK, 重传 30-15 == 15个数据包!!!
successfully trans 15
*****随机丢弃重传的数据包!!!!!
超时未接收到ACK, 重传 30-16 == 14个数据包!!!
*****随机丢弃重传的数据包!!!!!
超时未接收到ACK, 重传 30-16 == 14个数据包!!!
*****随机丢弃重传的数据包!!!!!
*****随机丢弃重传的数据包!!!!!
超时未接收到ACK, 重传 30-16 == 14个数据包!!!
超时未接收到ACK, 重传 30-16 == 14个数据包!!!
*****随机丢弃重传的数据包!!!!!
超时未接收到ACK, 重传 30-16 == 14个数据包!!!
对方无应答, 应该是退出了, 结束!!!
文件 1.jpg(1857353 bytes) 传输完成!

窗口大小20
传输用时:2.739秒
传输数据量(不包含重传数据): 1857353 bytes ( 1813KB )
吞吐率: 184683 bytes/s ( 180.272KB/s )

*****完成一次测试!!!!

```

下图也包含了超时重传，发送端的三次重传分别从分组 13、14、25 开始。

```

33 send seq: 25
34 send seq: 26
35 收到之前的ACKnum: 13! 不执行动作即可
36 *****随机丢弃重传的数据包! ! ! ! *****
37 超时未接收到ACK，重传 27-13 == 14个数据包! ! !
38 successfully trans 13
39 *****随机丢弃重传的数据包! ! ! ! *****
40 超时未接收到ACK，重传 27-14 == 13个数据包! ! !
41 -----收到超前的ACKnum,可以加速启动，开心! ! ! ACKnum: 25
42 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
43 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
44 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
45 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
46 *****随机丢弃重传的数据包! ! ! ! *****
47 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
48 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
49 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
50 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
51 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
52 *****随机丢弃重传的数据包! ! ! ! *****
53 超时未接收到ACK，重传 27-25 == 2个数据包! ! !
54 对方无应答，应该是退出了，结束! !
55 文件helloWorld.txt(1655808 bytes)传输完成!
56
57 窗口大小20
58 传输用时:6.407秒
59 传输数据量(不包含重传数据): 1655808 bytes ( 1617KB )
60 \
61 \
62 \
63 \
64 \
65 \
66 \
67 \
68 \
69 \
70 \
71 \
72 \
73 \
74 \
75 \
76 \
77 \
78 \
79 \
80 \
81 \
82 \
83 ****完成一次测试! ! ! ****

```

## 5 总结反思

### 5.1 编程实现

在实验过程中因代码不严谨出现许多错误，通过改进数据包的类型（改为结构体）简化了编程操作。通过使用超时时间间接完成“定时器”功能，避免了线程可能引发的问题，也达到了很好的效果。

### 5.2 协议理解

UDP 是传输层中面向无连接的协议，本实验以 UDP 为平台，进行可靠传输协议的设计，参考了 TCP 的很多思想，包括序列号的设计、快速重传方法等等，对两种协议的了解也更深入。在编程过程中，复现巩固了 GBN 和 SR 方法，了解了 TCP 中的具体实现，TCP 的设计可以说是 GBN 和 SR 的混合体。

熟悉了流量控制和拥塞控制的区别。流量控制是作用于接收者的，它是控制发送者的发送速度从而使接收者来得及接收，根本目的是防止分组丢失，它是构成 TCP 可靠性的一方面。拥塞控制是作用于网络的，它是防止过多的数据注入到网络中，避免出现网络负载过大的情况。