# 基于UDP服务设计可靠传输协议并编程实现3-2

**学号：2013622**

**姓名：罗昕珂**

## 一、实验内容

在实验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。

## 二、实验要求

(1) 实现单向传输。 (2) 对于每一个任务要求给出详细的协议设计。 (3) 给出实现的拥塞控制算法的原理说明。 (4) 完成给定测试文件的传输，显示传输时间和平均吞吐率。 (5) 性能测试指标：吞吐率、时延，给出图形结果并进行分析。 (6) 完成详细的实验报告（每个任务完成一份）。 (7) 编写的程序应结构清晰，具有较好的可读性。 (8) 提交程序源码和实验报告。

## 三、协议设计

### 1、传输协议特点

- 发送方和接收方有握手和挥手过程，确认连接断开连接；
- 数据可以切包传输，保存序列号；
- 数据报有差错检测功能，通过序列号和标志位检查，正确接收返回确认；
- 数据报有出错重传，解决出错、超时、丢失等情况；
- 数据报使用滑动窗口传输机制和序列号，不会出现失序情况；
- 发送方和接收方有挥手过程，断开连接。

### 2、数据报格式

握手挥手时的传输内容

| 校验和 | 标志位 |
| --- | --- |

普通数据包

| 校验和 | 标志位 | 序列号 | 数据 |
| --- | --- | --- | --- |

传输段最后一个数据包，则增加一个字节来保存该数据段长度。

| 校验和 | 标志位 | 序列号 | 长度 | 数据 |
| --- | --- | --- | --- | --- |

**1）校验和：**

计算校验和时，累加的长度可变

**2）标志位：**

- ACK,用于握手确认连接和发包确认接收以及确认挥手断开连接；
- SHAKE_1,SHAKE_2,SHAKE_3,三次握手确认连接
- WAVE_1,WAVE_2,两次挥手断开连接
- last_len：最后一个包的数据段长度
- 结尾数据包标志位： 对于非结尾数据包，标志位的倒数第 4 位为 1；对于结尾数据包，标志位的倒数 4，5 位为 1；

**3）序列号：**

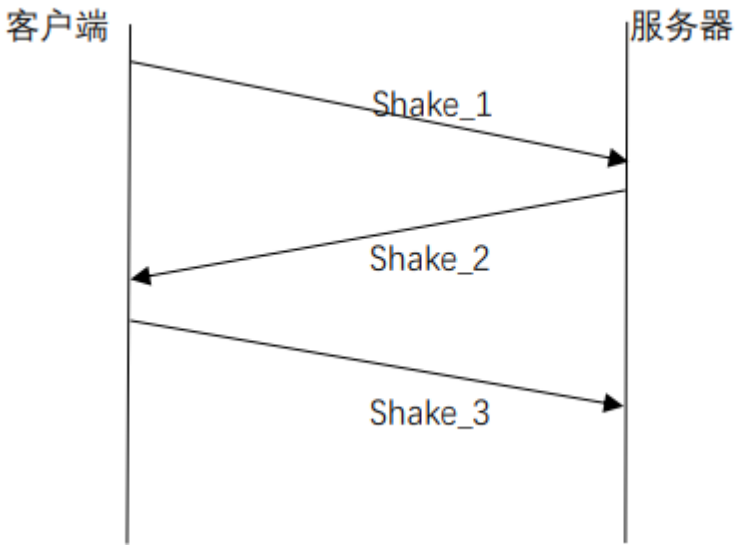
8位，用于握手确认连接和发送切片时保存序列号；

**4）长度限制：**

对于一个数据包的长度是有限制的，由于序列号只有8位，对于不是最后一个传输的数据包来说最多数据长度为 253 个字节，即数据报文为 256 个字节。对于传输段最后的不完整的数据来说则按具体长度发送。

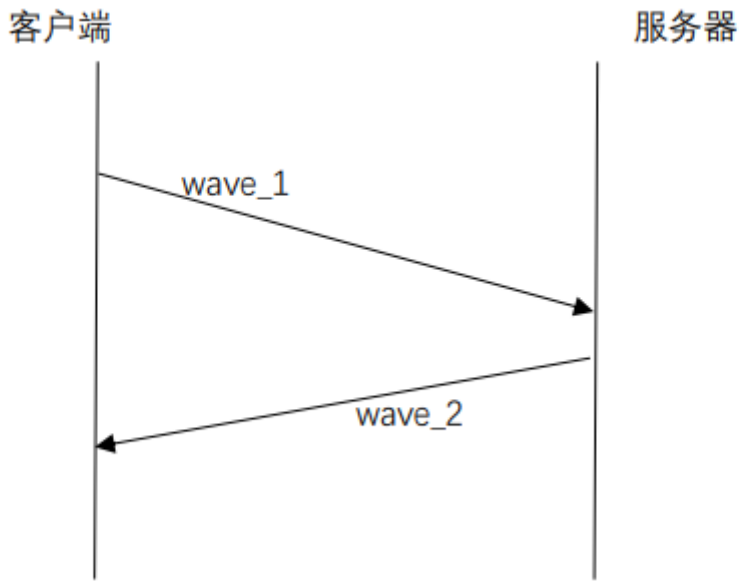
## 3、握手确认连接

- 单向传输，三次握手确认连接
- 步骤一：客户端发送SHAKE_1段握手信号
- 步骤二：服务端接收SHAKE_1段，计算校验和，若收到 SHAKE_1 且校验和等于 0，则回送SHAKE_2段.
- 步骤三：客户端接收SHAKE_2段，并计算出校验和正确，开始回送SHAKE_3段.
- 步骤四：服务器端接收到SHAKE_3第三次握手信号，且计算出校验和正确，那么握手成功，建立连接。




## 4、挥手断开连接

- 单向传输，二次握手断开连接
- 步骤一：客户端发送次挥手信号 wave_1。
- 步骤二：服务端接收后计算校验和，若收到 wave_1 且校验和等于 0，则开始 发送第二次挥手信号 wave_2。
- 步骤三：客户端收到服务端发来的 wave_2，并计算出校验和正确，则挥手成功，断开连接。




## 5、数据报的差错检测

在发送包（包括 ACK 和 NAK）还有握手挥手的时候都需要进行校验和的计算，来 确保发包是否出错

**1）发送端：**

- 对要发送的UDP数据报，校验和域段清0
- 产生伪首部将数据报用0补齐为16位整数倍
- 将伪首部和数据报看成16位整数倍序列
- 进行16位二进制反码求和运算计算校验和，并将校验和结果取反写入校验和域段。

**2）接收端：**

- 对接收到的UDP数据报，产生伪首部将数据报用0补齐为16位整数倍
- 将数据报看成16位整数序列
- 进行16位二进制反码求和运算计算校验和
- 并将校验和结果取反
- 如果取反结果为0，没有检测到错误
- 否则，数据报存在差错。

## 6、数据报切包发送

单片长度过大时切包发送，单片长度不超过253

## 7、数据报确认重传：

**1）超时重传：**

在发送一个数据包之后，就开启一个定时器，若是在这个时间内没有收到发送数据的 ACK 确认报文，则对该报文进行重传。并且将发送端的计数器减一，选择适当的序列号。

在文件传输过程中，在服务端 recvfrom 是进行阻塞接受的，使用库函数将其阻塞时间进行设置，在我们的 TIMEOUT 范围之内。

**2）出错重传：**

- 接收端收到包之后，计算校验和，如果数据包检测无差错，将 ACK 标志置为 1，发送给发送端，确认收到；如果数据包校验和有差错，将 ACK 标志置为 0，发送给发送端，表示出现包错误，并丢弃错误的数据包；
- 发送端收到接收端发来的 ACK 之后，如果 ACK 标志位为 1,表示数据包发送成功，发送端继续发送下一个数据包；如果 ACK 标志位为 0，表示数据包有差错，发送端将重传数据包。

**3）挥手握手重传：**

- 握手挥手过程中的丢包因为无法确认和检测校验和，只能进行重新进行握手挥手实现。
- 并且在挥手中进行超时次数的设置，超过次数认为对方完全断网，自己则进行资 源回收后断网。

### 8、滑动窗口传输机制

该协议允许发送方在等待确认前可以连续发送多个分组。由于发送方不必 每发一个分组就停下来等待确认，因此该协议可以加速数据的传输。但它也受限 于在流水线中未确认的分组不能超过最大窗口数。 不同于 3-1 实现的停等协议的是，之前是一个一个连续地发送，相当于滑动窗口的窗口大小等于 1。

在发送端，窗口内的分组序号对应的数据分组是可以连续发送的。窗口内的数据 分组有：

- 已发送但尚未得到确认
- 未发送但可以连续发送
- 已发送且已得到确认，但窗口中本序号的前面还有未得到确认的数据分组

滑动窗口法要求各数据分组按顺序发送，但并不要求确认按序返回。一旦窗口前 面部分的数据分组得到确认，则窗口向前滑动相应的位置，落入窗口中的后续分 组又可以连续发送。

在本次程序中，对于发送方我们设定一个大小为 WINDOW_SIZE 的序号窗口，其 中的数据为已经发送确还未确认的。

- 如果窗口未满，则我们可以继续发送数据包；
- 如果窗口满了，则需要等待 ACK 确认后窗口的减小，之后才能发送数据包。

### 9、GBN 协议设计

采用 while 循环来回调换 recv 和 send 进行实现，避免了 使用线程导致的繁琐。具体含义是对于每一个发送的数据包，记录了他发送出去 的时间，存储在一个队列里 timer_list 中，之后根据当前 clock()-最初发送的时间来判断队首的包是否发送超时，若超时则需要进行回退重发，并且清空队列；如 果未超时收到包，则需要弹出队列队首，即更新计时器。

### 10、累计确认

只有当接收端收到的序列号和自己累计确认的序列号相同，才会发送这个序号对应的 ACK 包，否则即使接受到了之后的包，也只会发送之前连续接受到的正确的包的 ACK。

## 四、实验代码

### 三次握手，两次挥手，分片发送，校验和计算，差错检测，超时重传，确认重传等同3-1代码一致

## 1、滑动窗口

**先判断是否已全部发送成功**

```
        if (send_succ_count == package_sum)
            break;
```

**在发送窗口未填满时，边发送边进行接收检测，查看接收方是否有累计确认状态码发送过来。发送后放入窗口，开始计时，并记录在窗口里面的序列号，进入窗口的记为1。**

发送窗口是否已满的判断：

因为设定的序列号只有一个 char 型大小(8 比特)，因此存在序号重复的可能，
使得 ACK 序号和发送方窗口是否已满的判断很难通过 base 和 nextpacknum 判断，
因此我们可以利用前面提到的 send_time_list 的长度和变换进行判断

```
    if (send_time_list.size() < WINDOW_SIZE && send_count != package_sum) {
            send_package(message + send_count * piece_len,
                send_count == package_sum - 1 ? length - (package_sum - 1) *
piece_len : piece_len,//是最后一个包长度取前面那个
                nextseqnum % ((int)UCHAR_MAX + 1),//序列号
                send_count == package_sum - 1);//是否是最后一个包

            //发送后放入窗口，开始计时
            send_time_list.push(make_pair(clock(), nextseqnum % ((int)UCHAR_MAX +
1)));
            //并记录在窗口里面的序列号，进入窗口的记为1
            in_list[nextseqnum % ((int)UCHAR_MAX + 1)] = 1;
            nextseqnum++;
            send_count++;
        }
```

**成功接收到数据包，则进行发送窗口的滑动，并且更新定时器，继续数据包发送。**

```
        if (recvfrom(client, recv, 3, 0, (sockaddr*)&serverAddr, &lengthmp) !=
SOCKET_ERROR && checksum(recv, 3) == 0 &&
            recv[1] == ACK && in_list[(unsigned char)recv[2]]) {//recv2是序列号，接
收ACK

            //一个累计确认状态的序号高于当前的windowbase序号的包，则进行发送窗口的滑
动，
            while (send_time_list.front().second < (unsigned char)recv[2]) {
                send_succ_count++;//已确认
                windowbase++;//窗口移动
```

```
                in_list[send_time_list.front().second] = 0;//将该包的序列号移出窗口，记为0
                send_time_list.pop();//删除窗口时间序列第一个元素
            }
            //未超时收到包，则需要弹出队列队首，即更新计时器。
            in_list[send_time_list.front().second] = 0;//对应的序列号，不在窗口内，记为0
            send_succ_count++;
            windowbase++;
            send_time_list.pop();//删除第一个元素
        }
```

**在对方 ACK 包丢失的情况下，我们需要处理队列 send_time_list 的出队个数，我们通过创建一个 bool 型的数组来实现在队列中的序号这样可以快速地进行查找。**

```
queue<pair<int, int>> send_time_list;
in_list[nextseqnum % ((int)UCHAR_MAX + 1)] = 1;
```

**GBN 协议设计根据当前 clock()-最初发送的时间来判断队首的包是否发送超时，若超时则需要进行回退重发，并且清空队列；如果未超时收到包，则需要弹出队列队首，即更新计时器。**

```
            if (clock() - send_time_list.front().first > TIMEOUT) {//当前clock()判断队首的包是否发送超时
                nextseqnum = windowbase;//回退到windowbase
                send_count -= send_time_list.size();//减去现在队列中的元素个数，缩小窗口等待的数据包
                //清空计时器。
                while (!send_time_list.empty())
                    send_time_list.pop();
            }
```

**打印传输日志**

```
        if (windowbase % 100 == 0)
            printf("已经发送第%d个数据包，此文件总共已经发送%.2f%%\n",windowbase,(float)windowbase / package_sum * 1000);
```

## 2、main函数

**可设置固定窗口的大小，但由于序列号的限制，不能大于序号域长度**

```
    printf("请输入发送窗口大小：\n");
    WINDOW_SIZE=1;
```

```
    WINDOW_SIZE %= UCHAR_MAX; //防止窗口大小大于序号域长度
```

3、累计确认

**只有当接收端收到的序列号和自己累计确认的序列号相同，才会发送这个序号对应的 ACK 包，否则即使接受到了之后的包，也只会发送之前连续接受到的正确的包的 ACK。**

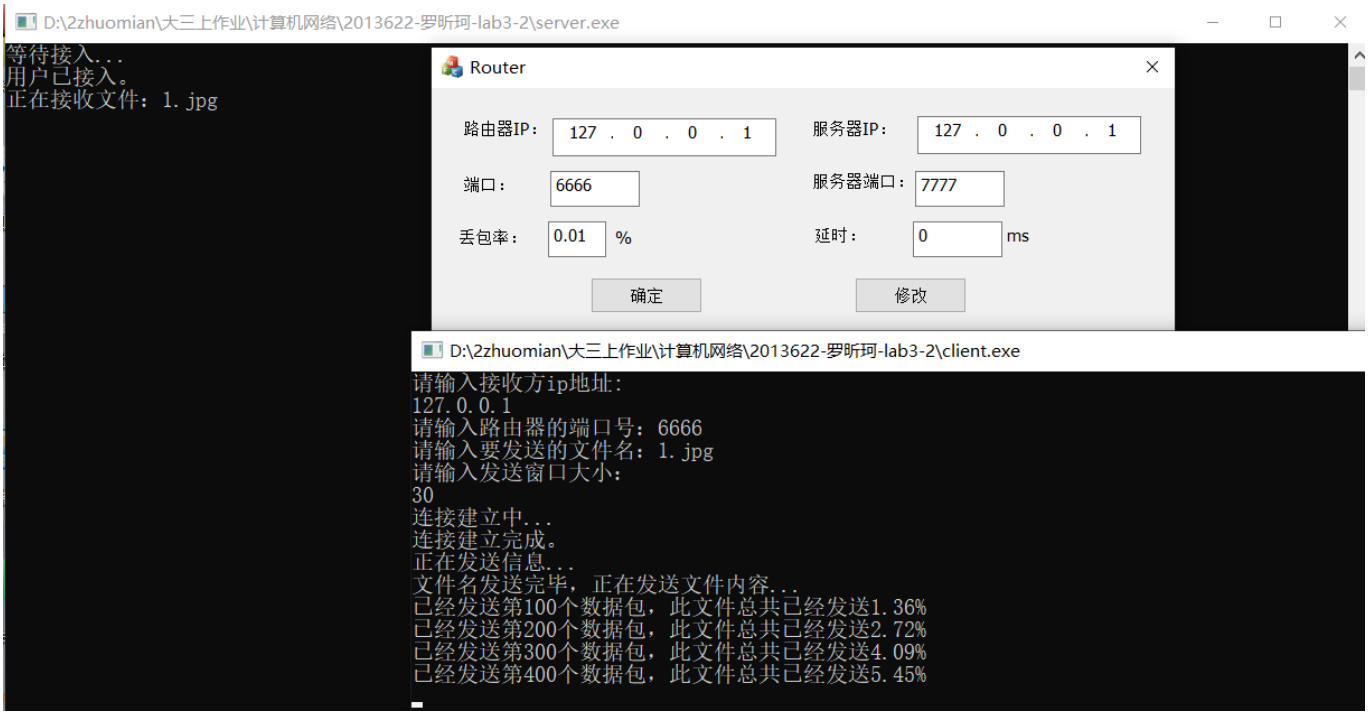
```
            memset(recv, 0, sizeof(recv));
            while (recvfrom(server, recv, piece_len + 4, 0,
(sockaddr*)&clientAddr, &lentmp) == SOCKET_ERROR);
            char send[3];
            int flag = 0;
            if (checksum(recv, piece_len + 4) == 0 && (unsigned char)recv[2] ==
last_order) {
                last_order++;
                flag = 1;
            }

            //返回ack时发送正确接收的序列号
            send[1] = ACK;//标志位
            send[2] = last_order - 1;//序列号
            send[0] = checksum(send + 1, 2);//校验和
            sendto(server, send, 3, 0, (sockaddr*)&clientAddr,
sizeof(clientAddr));//发送ACK回去
            if (flag)
                break;
```

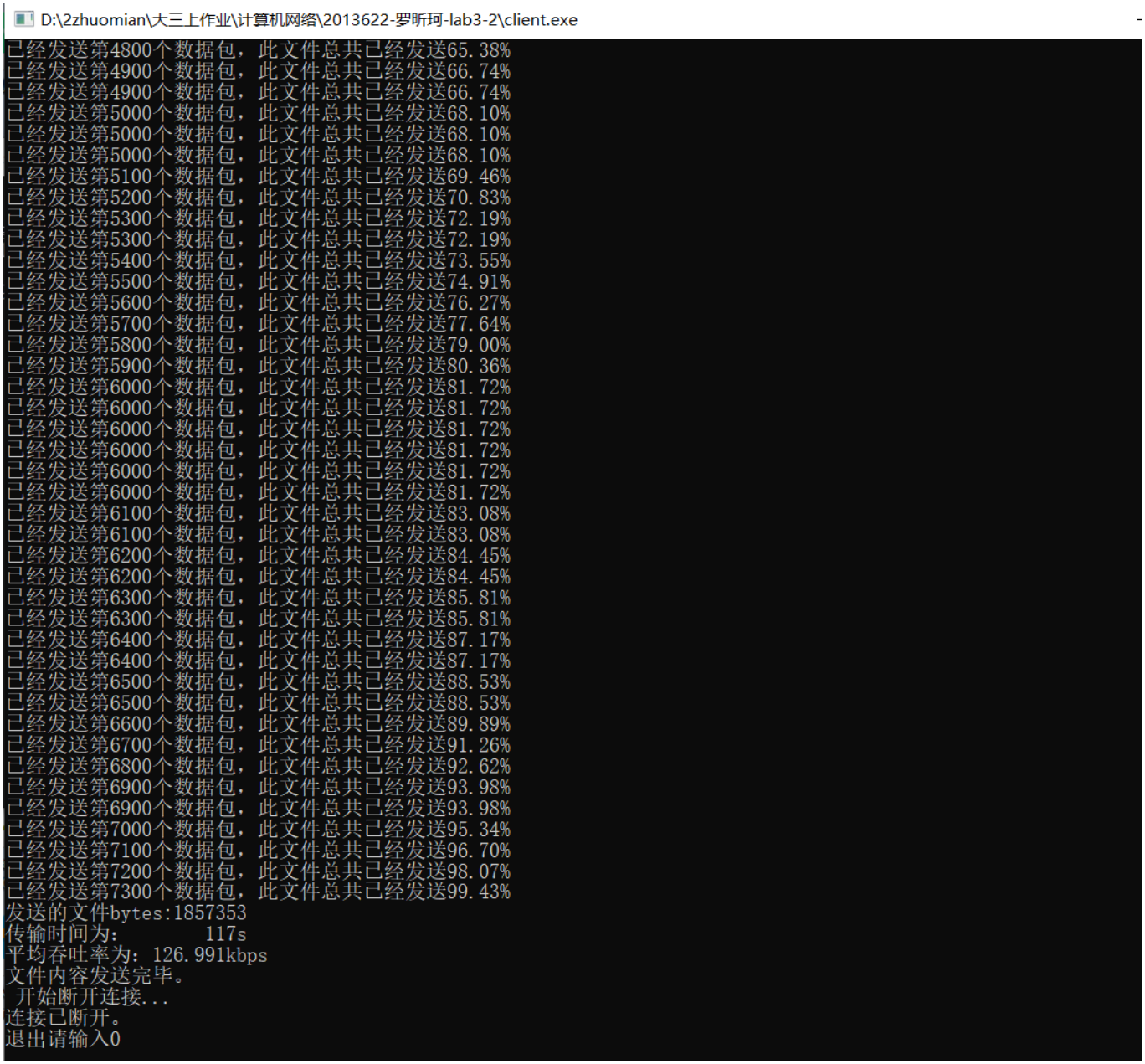
# 五、实验结果

接收端的端口固定为7777，路由器的端口可变。

由发送端选择路由器的端口6666，窗口设定为30，选择发送的文件名:1.jpg




由发送端发送给路由器，路由器转发到接收端，接收结果如下：




发送的文件bytes:1857353

传输时间为： 117s

平均吞吐率为：126.991kbps