



南開大學  
Nankai University

计算机学院  
计算机网络实验报告

# UDP 可靠数据传输

姓名：赵一名

学号：2013922

专业：计算机科学与技术

2022 年 11 月 26 日

## 目录

<b>1 概述</b>	<b>2</b>
1.1 程序使用说明 . . . . .	3
<b>2 原理介绍</b>	<b>3</b>
2.1 滑动窗口机制 . . . . .	4
2.2 <i>GBN</i> 协议 . . . . .	4
2.3 滑动窗口算法 . . . . .	4
<b>3 具体实现</b>	<b>5</b>
3.1 基于滑动窗口的流量控制机制 . . . . .	5
3.2 累计确认机制实现 . . . . .	5
<b>4 传输速率测量</b>	<b>6</b>
<b>5 实验中遇到的问题</b>	<b>8</b>

## 1 概述

本次实验中我在实验 3.1 的基础上进一步 *UDP* 实现了可靠文件传输，本程序支持固定窗口大小的流量控制机制，并支持累计确认。

程序运行的最终效果图如下图所示：

```

D:\ppt3-\computer_network\3.1\test\send.exe
please enter file path
test\\1.jpg
file name:1.jpg
start send filelen=1857353
datanum:0 ack:1 syn:0 fin:0 seqnum:1 acknum:4092 checksum:61441 datalen:0
datanum:1 ack:1 syn:0 fin:0 seqnum:1 acknum:8178 checksum:57354 datalen:0
datanum:2 ack:1 syn:0 fin:0 seqnum:1 acknum:12264 checksum:53267 datalen:0
datanum:3 ack:1 syn:0 fin:0 seqnum:1 acknum:16350 checksum:49180 datalen:0
datanum:4 ack:1 syn:0 fin:0 seqnum:1 acknum:20436 checksum:45093 datalen:0
datanum:5 ack:1 syn:0 fin:0 seqnum:1 acknum:24522 checksum:41006 datalen:0
datanum:6 ack:1 syn:0 fin:0 seqnum:1 acknum:28608 checksum:36919 datalen:0
datanum:7 ack:1 syn:0 fin:0 seqnum:1 acknum:32694 checksum:32832 datalen:0
datanum:8 ack:1 syn:0 fin:0 seqnum:1 acknum:36780 checksum:28745 datalen:0
datanum:9 ack:1 syn:0 fin:0 seqnum:1 acknum:40866 checksum:24658 datalen:0
datanum:10 ack:1 syn:0 fin:0 seqnum:1 acknum:44952 checksum:20571 datalen:0
datanum:11 ack:1 syn:0 fin:0 seqnum:1 acknum:49038 checksum:16484 datalen:0
timeout!
resend datanum:12 ack:0 syn:0 fin:0 seqnum:49038 acknum:0 checksum:0 datalen:4086
datanum:12 ack:1 syn:0 fin:0 seqnum:1 acknum:53124 checksum:12397 datalen:0
datanum:13 ack:1 syn:0 fin:0 seqnum:1 acknum:57210 checksum:8310 datalen:0
datanum:14 ack:1 syn:0 fin:0 seqnum:1 acknum:61296 checksum:4223 datalen:0
datanum:15 ack:1 syn:0 fin:0 seqnum:1 acknum:65382 checksum:136 datalen:0
datanum:16 ack:1 syn:0 fin:0 seqnum:1 acknum:69468 checksum:61584 datalen:0
datanum:17 ack:1 syn:0 fin:0 seqnum:1 acknum:73554 checksum:57497 datalen:0
datanum:18 ack:1 syn:0 fin:0 seqnum:1 acknum:77640 checksum:53410 datalen:0

```

图 1.1: 发送端

未发生丢包时，程序可正常发送，当丢包事件触发计时器时，重发丢失的数据包。

```

D:\ppt3-\computer_network\3.1\test\receive.exe
duplicate data datanum:40 ack:0 syn:0 fin:0 seqnum:163446 acknum:0 checksum:4842 datalen:4086
duplicate data datanum:39 ack:0 syn:0 fin:0 seqnum:159360 acknum:0 checksum:46091 datalen:4086
duplicate data datanum:41 ack:0 syn:0 fin:0 seqnum:167532 acknum:0 checksum:33503 datalen:4086
unexpected data datanum:44 ack:0 syn:0 fin:0 seqnum:179790 acknum:0 checksum:23795 datalen:4086
unexpected data datanum:45 ack:0 syn:0 fin:0 seqnum:183876 acknum:0 checksum:14963 datalen:4086
duplicate data datanum:40 ack:0 syn:0 fin:0 seqnum:163446 acknum:0 checksum:4842 datalen:4086
duplicate data datanum:42 ack:0 syn:0 fin:0 seqnum:171618 acknum:0 checksum:36624 datalen:4086
duplicate data datanum:39 ack:0 syn:0 fin:0 seqnum:159360 acknum:0 checksum:46091 datalen:4086
duplicate data datanum:40 ack:0 syn:0 fin:0 seqnum:175704 acknum:0 checksum:27939 datalen:4086
duplicate data datanum:40 ack:0 syn:0 fin:0 seqnum:163446 acknum:0 checksum:4842 datalen:4086
duplicate data datanum:41 ack:0 syn:0 fin:0 seqnum:167532 acknum:0 checksum:33503 datalen:4086
duplicate data datanum:41 ack:0 syn:0 fin:0 seqnum:167532 acknum:0 checksum:33503 datalen:4086
duplicate data datanum:42 ack:0 syn:0 fin:0 seqnum:171618 acknum:0 checksum:36624 datalen:4086
duplicate data datanum:42 ack:0 syn:0 fin:0 seqnum:171618 acknum:0 checksum:36624 datalen:4086
duplicate data datanum:43 ack:0 syn:0 fin:0 seqnum:175704 acknum:0 checksum:27939 datalen:4086
duplicate data datanum:43 ack:0 syn:0 fin:0 seqnum:175704 acknum:0 checksum:27939 datalen:4086
datanum:44 ack:0 syn:0 fin:0 seqnum:179790 acknum:0 checksum:23795 datalen:4086
unexpected data datanum:46 ack:0 syn:0 fin:0 seqnum:187962 acknum:0 checksum:57495 datalen:4086
unexpected data datanum:47 ack:0 syn:0 fin:0 seqnum:192048 acknum:0 checksum:28215 datalen:4086
unexpected data datanum:48 ack:0 syn:0 fin:0 seqnum:196134 acknum:0 checksum:51893 datalen:4086
unexpected data datanum:49 ack:0 syn:0 fin:0 seqnum:200220 acknum:0 checksum:1053 datalen:4086
datanum:45 ack:0 syn:0 fin:0 seqnum:183876 acknum:0 checksum:14963 datalen:4086
duplicate data datanum:44 ack:0 syn:0 fin:0 seqnum:179790 acknum:0 checksum:23795 datalen:4086
duplicate data datanum:45 ack:0 syn:0 fin:0 seqnum:183876 acknum:0 checksum:14963 datalen:4086
datanum:46 ack:0 syn:0 fin:0 seqnum:187962 acknum:0 checksum:57495 datalen:4086
datanum:47 ack:0 syn:0 fin:0 seqnum:192048 acknum:0 checksum:28215 datalen:4086
duplicate data datanum:45 ack:0 syn:0 fin:0 seqnum:183876 acknum:0 checksum:14963 datalen:4086
datanum:48 ack:0 syn:0 fin:0 seqnum:196134 acknum:0 checksum:51893 datalen:4086
duplicate data datanum:46 ack:0 syn:0 fin:0 seqnum:187962 acknum:0 checksum:57495 datalen:4086

```

图 1.2: 接收端

本实验中我采用的是 *GBN* 协议，所以仅会接受当前预期的包。

一次文件发送完毕后会输出所传输的文件的大小，传输所需的时间与吞吐量等信息，并询问发送端是否继续发送下一个文件，如下图所示：

```

D:\ppt3-computer_network\3.1\test\send.exe
resend datanum:441 ack:0 syn:0 fin:0 seqnum:1801932 acknum:0 checksum:0 datalen:4086
datanum:438 ack:1 syn:0 fin:0 seqnum:1 acknum:1793760 checksum:40780 datalen:0
resend datanum:442 ack:0 syn:0 fin:0 seqnum:1806018 acknum:0 checksum:0 datalen:4086
resend datanum:443 ack:0 syn:0 fin:0 seqnum:1810104 acknum:0 checksum:0 datalen:4086
resend datanum:444 ack:0 syn:0 fin:0 seqnum:1814190 acknum:0 checksum:0 datalen:4086
datanum:439 ack:1 syn:0 fin:0 seqnum:1 acknum:1797846 checksum:36693 datalen:0
datanum:440 ack:1 syn:0 fin:0 seqnum:1 acknum:1801932 checksum:32606 datalen:0
datanum:441 ack:1 syn:0 fin:0 seqnum:1 acknum:1806018 checksum:28519 datalen:0
datanum:442 ack:1 syn:0 fin:0 seqnum:1 acknum:1810104 checksum:24432 datalen:0
datanum:443 ack:1 syn:0 fin:0 seqnum:1 acknum:1814190 checksum:20345 datalen:0
datanum:444 ack:1 syn:0 fin:0 seqnum:1 acknum:1818276 checksum:16258 datalen:0
datanum:445 ack:1 syn:0 fin:0 seqnum:1 acknum:1822362 checksum:12171 datalen:0
datanum:446 ack:1 syn:0 fin:0 seqnum:1 acknum:1826448 checksum:8084 datalen:0
datanum:447 ack:1 syn:0 fin:0 seqnum:1 acknum:1830534 checksum:3997 datalen:0
timeout!
resend datanum:448 ack:0 syn:0 fin:0 seqnum:1830534 acknum:0 checksum:0 datalen:4086
resend datanum:449 ack:0 syn:0 fin:0 seqnum:1834620 acknum:0 checksum:0 datalen:4086
resend datanum:450 ack:0 syn:0 fin:0 seqnum:1838706 acknum:0 checksum:0 datalen:4086
resend datanum:451 ack:0 syn:0 fin:0 seqnum:1842792 acknum:0 checksum:0 datalen:4086
datanum:448 ack:1 syn:0 fin:0 seqnum:1 acknum:1834620 checksum:65445 datalen:0
resend datanum:452 ack:0 syn:0 fin:0 seqnum:1846878 acknum:0 checksum:0 datalen:4086
datanum:449 ack:1 syn:0 fin:0 seqnum:1 acknum:1838706 checksum:61358 datalen:0
timeout!
resend datanum:450 ack:0 syn:0 fin:0 seqnum:1838706 acknum:0 checksum:0 datalen:4086
datanum:450 ack:1 syn:0 fin:0 seqnum:1 acknum:1842792 checksum:57271 datalen:0
file size:1857353 bytes
file transmission time:33922.000000 ms
throughput:0.417737 Mbit/s
send next picture?
1:yes 2:no

```

图 1.3: 发送端

## 1.1 程序使用说明

使用路由器小程序的各个参数如下图所示：

The image shows a 'Router' configuration window. It contains the following fields and values:

- 路由器IP: 127 . 0 . 0 . 1
- 服务器IP: 127 . 0 . 0 . 1
- 端口: 5000
- 服务器端口: 6000
- 丢包率: 5 %
- 延时: 0 ms

At the bottom, there are two buttons: '确定' (OK) and '修改' (Modify).

图 1.4: 路由器小程序

打开 *receive.exe* 与 *send.exe* 两个可执行程序后，在发送端中输入待发送的文件的路径，文件将会自动传输到接收端同级目录下的 *receive* 文件夹中。

## 2 原理介绍

本程序中我采用的 *GBN*(回退 *N* 步) 协议来实现滑动窗口的流量控制机制。

## 2.1 滑动窗口机制

滑动窗口的基本原理就是在任意时刻，发送方都维持了一个连续的允许发送的帧的序号，称为发送窗口；同时，接收方也维持了一个连续的允许接收的帧的序号，称为接收窗口。发送窗口和接收窗口的序号的上下界不一定要一样，甚至大小也可以不同。不同的滑动窗口协议窗口大小一般不同。发送方窗口内的序列号代表了那些已经被发送，但是还没有被确认的帧，或者是那些可以被发送的帧。

若从滑动窗口的观点来统一看待 1 比特滑动窗口、GBN((回退  $N$  步) 及 SR(选择重传) 三种协议，它们的差别仅在于各自窗口尺寸的大小不同而已。

1 比特滑动窗口协议就是在实验 3.1 中实现的协议，其发送窗口 =1，接收窗口 =1。

GBN 协议，也即本次实验中我实现的协议，其发送窗口 >1，接收窗口 =1。

SR 协议，其发送窗口 >1，接收窗口 >1。

## 2.2 GBN 协议

GBN 协议指的是当接收方发现某数据包出错后，其后继续送来的正确 (但失序) 的包直接抛弃掉，并重发 ACK 以期待发送端超时重传，直至接收端收到预期的包。这样做的好处是接受端设计简单，不必设计缓冲区，为实现流量控制仅需在发送端维护一个发送窗口即可。

## 2.3 滑动窗口算法

为实现 GBN 协议的滑动窗口算法，其维护 3 个变量，发送窗口大小  $windowSize$ ，最早未确认的分组序号  $sendBase$ ，下一个待发送的序号  $nextSeqNum$ ，并始终维护如下不等式成立：

$$nextSeqNum < sendBase + windowSize$$

在本次实验中，由于采用固定窗口大小，所以上式中  $windowSize$  为一常量。当一个确认到达时，发送方向右移动  $sendBase$ ，从而允许发送方发送另一帧。同时，发送方为始终对  $sendBase$  处的数据包维护一个定时器，如果定时器在 ACK 到达之前超时，则重发此数据包。同时需要注意的是，发送方必须存储  $sendBase$  到  $nextSeqNum$  之间的所有数据包，因为在它们得到确认之前必须准备重发。

对于接收端，GBN 协议要求其必须按序接受发送端数据包，当遇到一个正确 (但失序) 的包后，接受端直接丢弃该包，并发送上一个已经按序接受的包的 ACK，以发送端再次发送丢失的包，当重复接收到一个已经收到的包时，接收端不做任何动作，即直接抛弃该包。

在本次实验中我采用的 GBN 协议的滑动窗口算法的一个示意图如下：

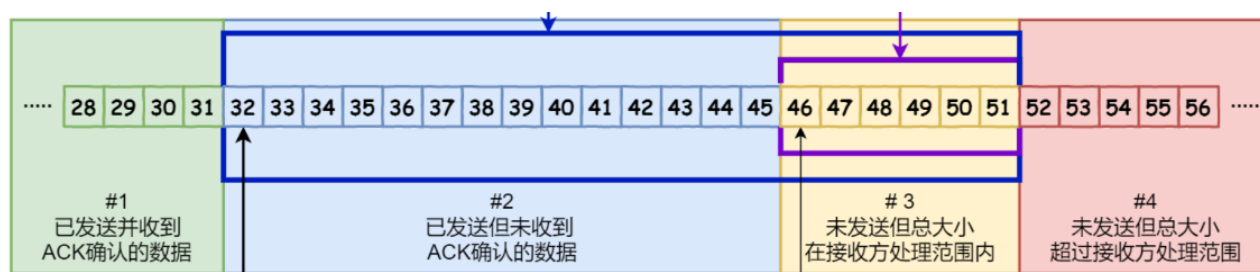


图 2.5: 滑动窗口示意图

### 3 具体实现

#### 3.1 基于滑动窗口的流量控制机制

发送端负责发送的线程的核心代码如下图所示：

```
i=0;
while(i<dataNum)
{
    if(buffer[i].seqnum+maxDataBufferSize>=sendBase+windowSize)
    {
        sleep(200);
        continue;
    }
    sd(i);
    if(noTimer)
    {
        noTimer=0;
        timeoutData=i;
        CreateThread(NULL,NULL,&handleTimer,NULL,0,NULL);
    }
    i++;
}
```

图 3.6: 滑动窗口机制实现

第一个 *if* 语句用于实现固定窗口大小的流量控制机制，每隔 0.2 秒检测一次当前是否满足 2.3 节中的不等式，若满足不等式，则可继续发送，若不满足不等式，则阻塞直到可以发送为止。这里使用睡眠的方式来等待不等式满足条件，但实际中理想实现应该是通过信号量机制，即当接受端由于接受到符合预期的数据包导致 *sendBase* 右移时向发送端发送一个表示可以继续发送的信号量，但在实现过程中，我发现使用信号量也是相当消耗系统资源的，所以我采用了消耗系统资源小，且实现容易的睡眠等待方式来实现这个机制。

#### 3.2 累计确认机制实现

实际中主要由接收端来实现累计确认机制，其实现代码如下图所示：

```

while(1)
{
    memset(&receiveData,0,datagramLen);
    rd();
    checksum(receiveData);
    if(receiveData.seqnum==nextAckNum)
    {
        nextAckNum+=receiveData.dataLen;
        fwrite(receiveData.data,receiveData.dataLen,1,p);
        showDataInfo(receiveData);
        memset(&sendData,0,datagramHeaderLen);
        sendData.seqnum=1;
        sendData.ack=1;sendData.acknum=nextAckNum;
        sendData.rwnd=receiveData.rwnd;
        checksum(sendData);
        if(receiveData.fin==1){
            sendData.fin=1;
            sd();
            break;
        }
        sd();
    }
    else if(receiveData.seqnum>nextAckNum)
    {
        printf("unexpected data ");
        sd();
        showDataInfo(receiveData);
    }
    else if(receiveData.seqnum<nextAckNum)
    {
        printf("duplicate data ");
        showDataInfo(receiveData);
    }
}

```

图 3.7: 累计确认机制实现

当一个具有顺序列号 *SeqNum* 的包到达时,接收方采取如下行动: 如果 *receiveData.seqnum == nextAckNum* 表示当前接收的包在滑动窗口内,即这就是我们想要接收的包;如果 *receiveData.seqnum < nextAckNum*,表明这是重复发送的包,接收端直接抛弃掉即可;如果 *receiveData.seqnum > nextAckNum*,表明当前接受的是一个失序的包,此时重发 *ACK*,也就是说即使已经收到更高序号的分组,接收方仍仅确认当前已经按序收到的 *SeqNum* 的最大值,这种确认被称为是累积的 (*cumulative*)。

## 4 传输速率测量

本次实验中,我对传输速率随丢包率与路由器延时的变化情况进行了测量,并与实验 3.1 中设计的程序进行了对比,接下来的测试结果都是重复测量五次并取平均值的结果。

首先比较传输速率随丢包率的变化,在测量过程中路由器时延设为 0ms



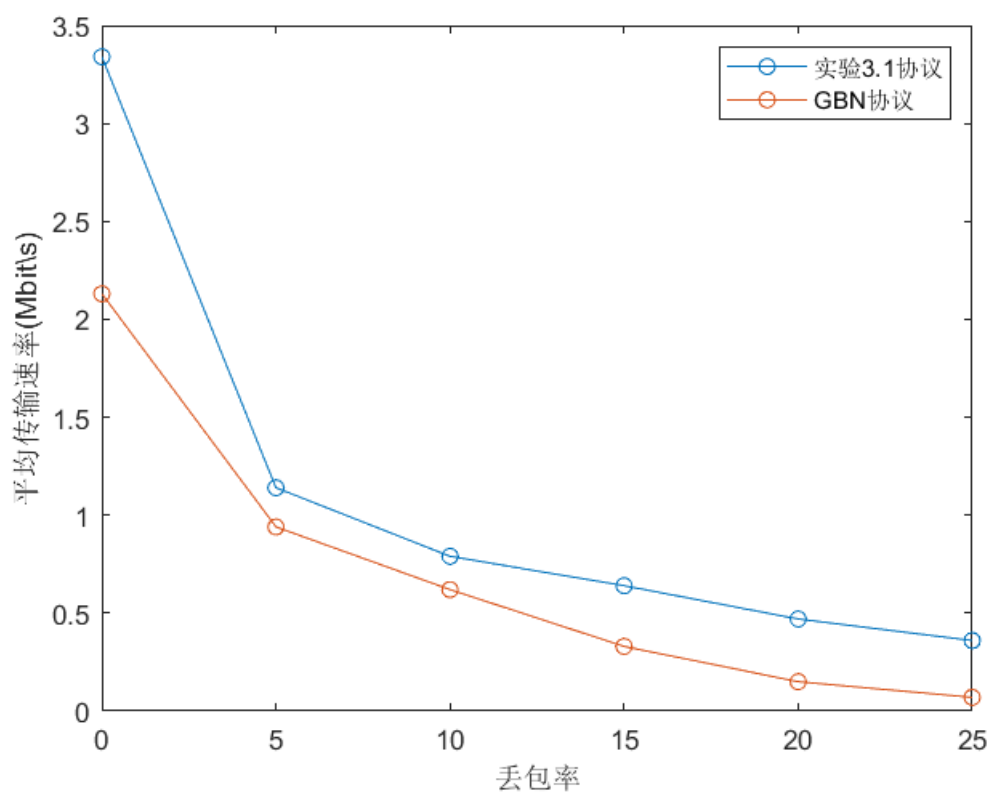


图 4.8: 传输速率随丢包率的变化

可以看到，当初始丢包率较小时，由于有流量控制的功能，*GBN* 协议的传输速率明显要慢一些，当丢包率为 5% 左右时，两协议传输速率相差不大，但当丢包率进一步增大，由于 *GBN* 协议丢弃失序分组的特性，其传输速率几乎已经变得很小。

接下来比较传输速率随路由器时延的变化，在测量过程中丢包率设为 0%



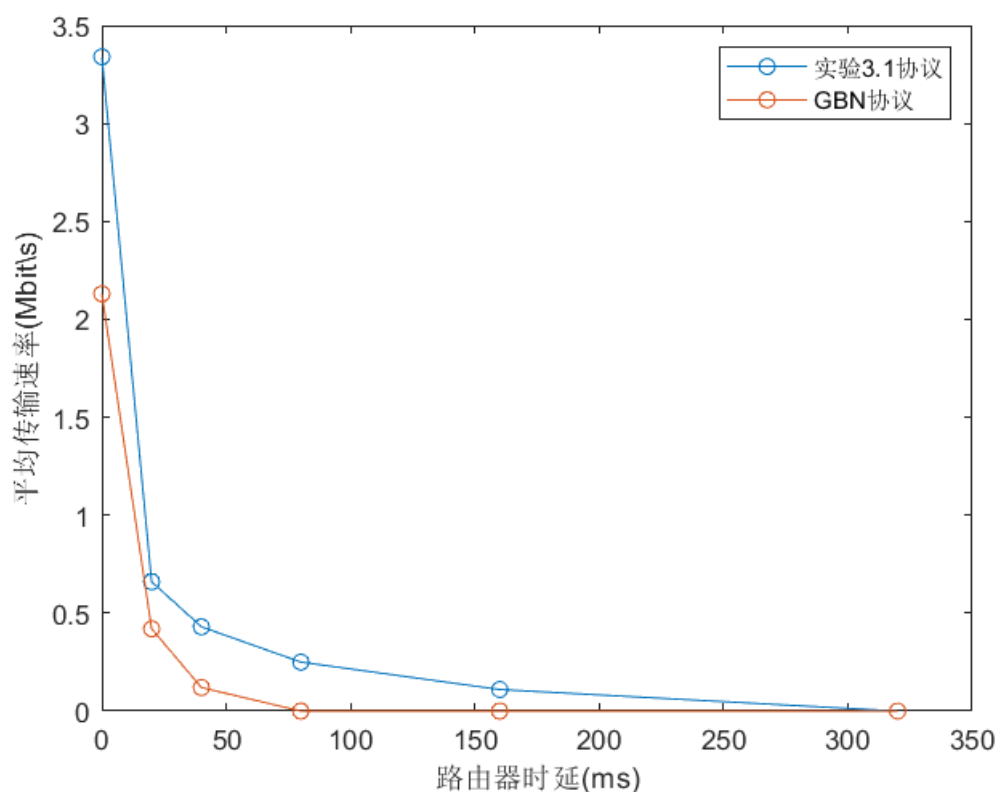


图 4.9: 传输速率随路由器时延的变化

可以看到路由器时延对两协议的影响都非常大, 实际测试中我发现当时延大于  $120ms$  时 *GBN* 协议传输速率已经让人无法接受, 所以我直接将上图中对应位置的传输速率设为 0.

## 5 实验中遇到的问题

由于我采用的机制是 *GBN* (退回  $N$  步), 所以当出现单个分组的差错时就会引起大量的重传发送, 而且这些重传都是有计时器超时来发现的, 这会大大拖垮程序的性能。最初我在超时重传的函数中是这样设计的:

```
void CALLBACK TimerProc(HWND hwnd, UINT uMsg, UINT idEvent, DWORD dwTime)
{
    if(SOCKET_ERROR==sendto(clientSocket, (char*)&(*timeoutData), datagramLen, 0, (SOCKADDR*)&destinationAddrInfo, addrLen))
    {
        printf("send error\n");
    }
}
```

图 5.10: 超时重传 0

我仅仅重传了设置计时器时 *sendBase* 处的单个数据包, 但是由于接收方丢弃了所有失序分组, 所以当超时事件发送时意味着从 *sendBase* 到下一个要发送的 *nextSeqNum* 之间的所有包接收方都未收到, 在我实际测试中, 采用这样的方式发送一个文件所需要的时间几乎是不可接受的, 我对给出的 *1.jpg* 这张图片进行测试, 得到的数据如下图所示:

```
file size:1857353 bytes
file transmission time:97094.000000 ms
throughput:0.145946 Mbit\s
```

图 5.11: 测试 0

为了改善这一情况,我采用了类似 *reno* 算法的快速恢复机制来解决这一问题,即当遇到超时事件时不仅重发 *sendBase* 处的数据包,而且同时发送 *sendBase* 到 *nextSeqNum* 处的所有数据包,这样就可以有效解决上述问题了。

改正后的超时重传函数如下:

```
void CALLBACK TimerProc(HWND hwnd, UINT uMsg, UINT idEvent, DWORD dwTime)
{
    int t=i;
    for(int j=timeoutData;j<=t;j++)
        sd(j);
}
```

图 5.12: 超时重传 1

再次测试的结果如下: 可以看到,性能提升了  $0.762/0.146 = 5.21$  倍左右,基本达到了预期的要求。

```
file size:1857353 bytes
file transmission time:18596.000000 ms
throughput:0.762018 Mbit\s
```

图 5.13: 测试 1