



南开大学  
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

---

### 实验三：基于 UDP 服务设计可靠传输协议

---

沙璇 1911562

年级：2019 级

专业：信息安全

指导教师：徐敬东

2022 年 9 月 19 日

## 摘要

基于 UDP 服务设计可靠传输协议并编程实现

关键字: UDP , C++

## 目录

一、 实验内容	1
(一) 内容概述 . . . . .	1
(二) 实验要求 . . . . .	1
二、 协议设计	1
(一) 慢启动 . . . . .	1
(二) 拥塞避免 . . . . .	2
(三) 快速重传 . . . . .	3
(四) 快速恢复 . . . . .	3
(五) 日志打印 . . . . .	4
三、 编程赋现	4
(一) sever.cpp . . . . .	4
(二) client.cpp . . . . .	7
四、 实验结果	7
五、 实验收获	11

## 一、 实验内容

### (一) 内容概述

实验 3-3: 在实验 3-2 的基础上, 选择实现一种拥塞控制算法, 也可以是改进的算法, 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。

## 二、 协议设计

拥塞即对资源的需求超过了可用的资源。若网络中许多资源同时供应不足, 网络的性能就要明显变坏, 整个网络的吞吐量随之负荷的增大而下降。

为了提高网络利用率, 降低丢包率, 并保证网络资源对每条数据流的公平性, 因此采取拥塞控制机制。

拥塞控制的方法:

- (1) 慢启动
- (2) 拥塞避免
- (3) 快速重传
- (4) 快速恢复

### (一) 慢启动

发送方维持一个拥塞窗口  $cwnd$  (congestion window) 的状态变量。拥塞窗口的大小取决于网络的拥塞程度, 并且动态地在变化。发送方让自己的发送窗口等于拥塞窗口。

发送方控制拥塞窗口的原则是: 只要网络没有出现拥塞, 拥塞窗口就再增大一些, 以便把更多的分组发送出去。但只要网络出现拥塞, 拥塞窗口就减小一些, 以减少注入到网络中的分组数。

慢启动算法:

- (1) 拥塞窗口和接收窗口共同决定了发送者的发送窗口。
- (2) 当主机开始发送数据时, 如果立即所大量数据字节注入到网络, 那么就有可能引起网络拥塞, 因为现在并不清楚网络的负荷情况。
- (3) 较好的方法是先探测一下, 即由小到大逐渐增大发送窗口, 也就是说, 由小到大逐渐增大拥塞窗口数值。
- (4) 通常在刚刚开始发送报文段时, 先把拥塞窗口  $cwnd$  设置为一个最大报文段  $MSS$  的数值。而在每收到一个对新的报文段的确认后, 把拥塞窗口增加至多一个  $MSS$  的数值。用这样的方法逐步增大发送方的拥塞窗口  $cwnd$ , 可以使分组注入到网络的速率更加合理。

(5) 如果不施加手段进行控制, 慢启动必然使得 CWBD 很快膨胀, 为防止拥塞窗口  $cwnd$  的增长引起网络拥塞, 还需要另外一个变量, 慢开始门限  $ssthresh$

$cwnd < ssthresh$  时, 进行慢开始算法。

$cwnd > ssthresh$  时, 进行拥塞避免算。

$cwnd = ssthresh$  时, 两者皆可。

如图1所示



图 1: 拥塞控制

## (二) 拥塞避免

让拥塞窗口  $cwnd$  缓慢地增大, 即每经过一个往返时间 RTT 就把发送方的  $cwnd$  拥塞窗口  $cwnd$  加  $1cwnd$ , 而不是加倍  $cwnd$ 。这样拥塞窗口  $cwnd$  按线性规律缓慢增长, 比慢开始算法的拥塞窗口增长速率缓慢得多。

不论是慢开始还是拥塞避免只要网络出现拥塞 (没有按时到达) 时, 就把  $ssthresh$  的值置为出现拥塞时的拥塞窗口的一半 (但不能小于 2), 以及  $cwnd$  置为 1, 进行慢开始。目的是迅速减少主机发送到网络中的分组数, 使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

注意:

由指数增长拉低到线性增长, 降低出现拥塞的可能。“拥塞避免”并非指完全能够避免拥塞, 利用以上的措施要完全避免网络拥塞还是不可能的。

如图2所示

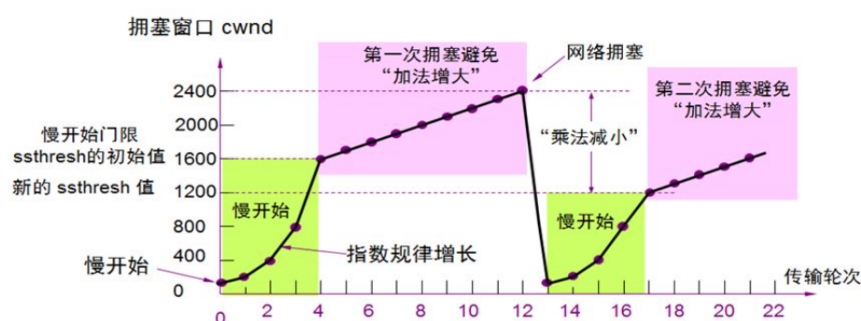


图 2: 拥塞避免

### (三) 快速重传

当发送方收到三个或以上的冗余 ACK (duplicate ACK), 就意识到之前发的包存在丢失, 于是快速重传。

如图3所示

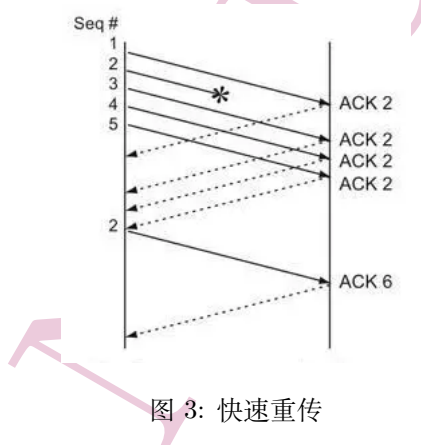


图 3: 快速重传

报文段 1 成功接收并被确认 ACK 2, 接收端的期待序号为 2, 当报文段 2 丢失, 报文段 3 失序到来, 与接收端的期望不匹配, 接收端重复发送冗余 ACK 2。

### (四) 快速恢复

快速恢复 (Fast retransmit) 具体过程:

(1) 当发送方连续收到三个重复确认, 就执行“乘法减小”算法, 把慢开始门限 ssthresh 减半。这是为了预防网络发生拥塞。然后立即重传丢失的报文段, 并将 CWND 设置为新的 ssthresh (减半后的 ssthresh) 请注意: 接下去不执行慢开始算法

有些快重传实现是把开始时的拥塞窗口 cwnd 值再增大一点, 即等于  $ssthresh + 3 * MSS$ 。这样做的理由是: 既然发送方收到三个重复的确认, 就表明有三个分组已经离开了网络。这三个分组不再消耗网络的资源而是停留在接收方的缓存中。可见现在网络中并不是堆积了分组而是减少了三个分组。因此可以适当把拥塞窗口扩大了些。

(2) 由于发送方现在认为网络很可能没有发生拥塞, 因此与慢开始不同之处是现在不执行慢开始算法 (即拥塞窗口 cwnd 现在不设置为 1), 而是把 cwnd 值设置为慢开始门限 ssthresh 减

半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

(3) 每次收到一个重复的确认时，设置  $CWND = CWND + SMSS$ （拥塞窗口加 1）。此时发送端可以发送新的 TCP 报文段

(4) 当收到新数据的确认时，设置  $CWND = ssthresh$ （ $ssthresh$  是新的慢启动门限值，由第一步计算得到）原因是因为该 ACK 确认了新的数据，说明从重复 ACK 时的数据都已收到，该恢复过程已经结束，可以回到恢复之前的状态了，也即再次进入拥塞避免状态。

快速重传和快速恢复完成之后，拥塞控制将恢复到拥塞避免阶段。

如图6所示

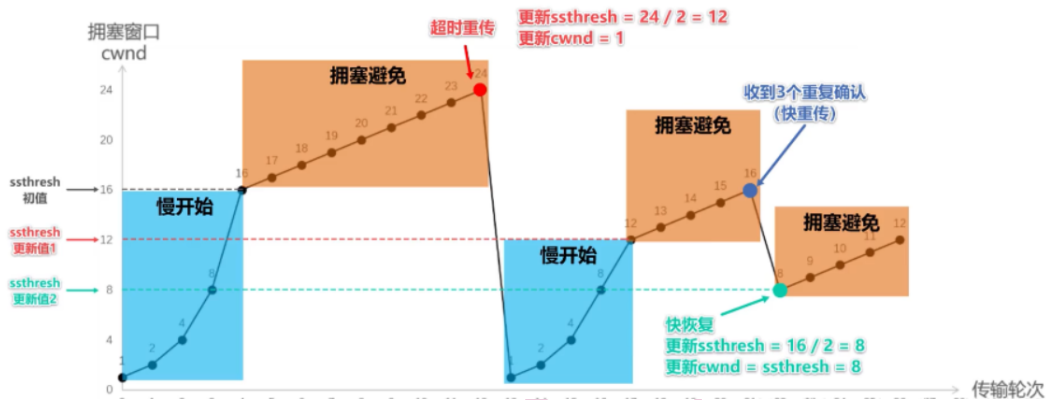


图 4: 快速恢复

## (五) 日志打印

测试文件传输时间、接收和发送窗口大小、平均吞吐率、建立连接、序列号、ack、校验和。

## 三、编程赋现

代码分为两部分，包括 client.cpp 和 sever.cpp。具体代码见附件 sever.cpp、client.cpp。

### (一) sever.cpp

由于基本功能在实验 3-2 已经赋现，因此在本次实验中只对新增内容进行讲解。

ackhandler

```

1 void ackhandler(unsigned int a)
2 {
3     long long index = a;
4     switch (STATE) // 状态
5     {
6     case SLOWSTART: // 慢启动状态
7         if ((index + seqnumber - curack) % seqnumber < minwindow(cwnd,
8             WINDOWSIZE)) // 收到的数据包 < 拥塞避免上限
9         {
10            cout << " <<<<<<<<收到了" << index << "号数据包的ack<<<<<<<<" << endl
11                << endl;
12            ack[index % WINDOWSIZE] = 3; // 已确认

```

```

11     if (cwnd <= ssthresh)//拥塞窗口小于阈值，使用慢启动
12     {
13         cwnd++;//每收到一个ack，cwnd+1；指数增加
14         cout << "=====慢启动阶段
15         =====" << endl;
16         cout << "cwnd= " << cwnd << "      ssthresh= " << ssthresh << endl <<
17         endl;
18     }
19     else
20     {
21         STATE = AVOID;//拥塞窗口大于等于阈值，进入拥塞避免阶段
22         //累积确认
23         for (int j = curack; j != (index + 1) % seqnumber; j = (++j) %
24             seqnumber)//滑动窗口最左端到当前序列号
25         {
26             ack[j % WINDOWSIZE] = 1;//ack初始化
27             ++totalack;//确认个数++
28             curack = (curack + 1) % seqnumber;//滑动窗口最左端++
29         }
30         else if (index == curack - 1)
31         //收到的ack是当前等待被确认的数据包的最小序列号-1的话，就证明收到了冗
32         余ack
33         {
34             dupack++;
35             //冗余ack++
36             //快速重传是指如果连续收到3个重复的确认报文(DUPACK)则认为该报文很可能
37             丢失了
38             //此时即使重传定时器没有超时，也重传
39             if (dupack == 3)//进入快速恢复 状态跳转到拥塞避免
40             {
41                 fasthandler();//快速重传
42                 ssthresh = cwnd / 2;//阈值设置为拥塞窗口的一半
43                 /*
44                 将CWND设置为新的ssthresh（减半后的ssthresh）
45                 拥塞窗口cwnd值增大，即等于 ssthresh + 3 * MSS 。
46                 理由：既然发送方收到三个重复的确认，就表明有三个分组已经离开
47                 了网络。
48                 这三个分组不再消耗网络 的资源而是停留在接收方的缓存中。
49                 可见现在网络中并不是堆积了分组而是减少了三个分组。
50                 因此可以适当把拥塞窗口扩大了些。
51                 */
52                 cwnd = ssthresh + 3;//扩大拥塞窗口
53                 STATE = AVOID;//进入拥塞避免阶段
54                 dupack = 0;//冗余ack归零
55             }
56         }

```

```

53         break;
54     case AVOID:
55         if ((index + seqnumber - curack) % seqnumber < minwindow(cwnd,
56             WINDOWSIZE))
57         {
58             cout << "收到" << index << "号数据包的ack" << endl
59                 << endl;
60             ack[index % WINDOWSIZE] = 3; //已确认
61             cwnd = cwnd + 1 / cwnd; //每接收一个ACK增长1/cwnd
62             cout << "达到阈值，进入拥塞避免阶段"
63                 << endl;
64             cout << "cwnd= " << int(cwnd) << " ssthresh=" << ssthresh << endl
65                 << endl;
66             //累积确认
67             for (int j = curack; j != (index + 1) % seqnumber; j = (++j) %
68                 seqnumber)
69             {
70                 ack[j % WINDOWSIZE] = 1;
71                 ++totalack;
72                 curack = (curack + 1) % seqnumber;
73             }
74             else if (index == curack - 1)
75             {
76                 dupack++;
77                 if (dupack == 3)
78                 {
79                     fasthandler(); //快速重传
80                     STATE = AVOID; //拥塞避免
81                     dupack = 0;
82                 }
83             }
84             break;
85         }
86     }

```

快速重传函数如下所示

#### fasthandler

```

1 void fasthandler()
2 {
3     packet* pkt1 = new packet;
4     pkt1->init_packet(); //pkt初始化
5     //把当前滑动窗口最左端的包（当前等待被确认的数据包的最小序列号）一直到当前
6     //发送的序列号的包都重传一遍
7     for (int i = curack; i != curseq; i = (i++) % seqnumber)
8     {
9         memcpy(pkt1, &buffer[i % WINDOWSIZE], BUFFER); //buffer传入

```



```

10         sendto(sockServer, (char*)pkt1, BUFFER, 0, (SOCKADDR*)&
11             addrClient, sizeof(SOCKADDR)); //socket 发送
12         cout << "***重传第 " << i << " 号数据包***" << endl;
13     }
14 }

```

## (二) client.cpp

在本次实验中只对 client.cpp 的新增内容进行讲解。

### 模拟丢包

```

1 //模拟丢包
2 BOOL lossInLossRatio(float lossRatio) {
3     int lossBound = (int)(lossRatio * 100); //丢包率的百分数
4     int r = rand() % 101; //产生0~100之间的随机整数
5     if (r <= lossBound) { //小于丢包率
6         return TRUE; //丢包
7     }
8     return FALSE; //没丢
9 }

```

## 四、实验结果

测试文件 1.png，传输文件如图5所示


 1.jpg	2020/4/2 21:24	JPG 图片文件	1,814 KB
---	----------------	----------	----------

图 5: 传输文件

注意此时日期为 2020/4/2。传输开始前 sever 端显示如图6所示

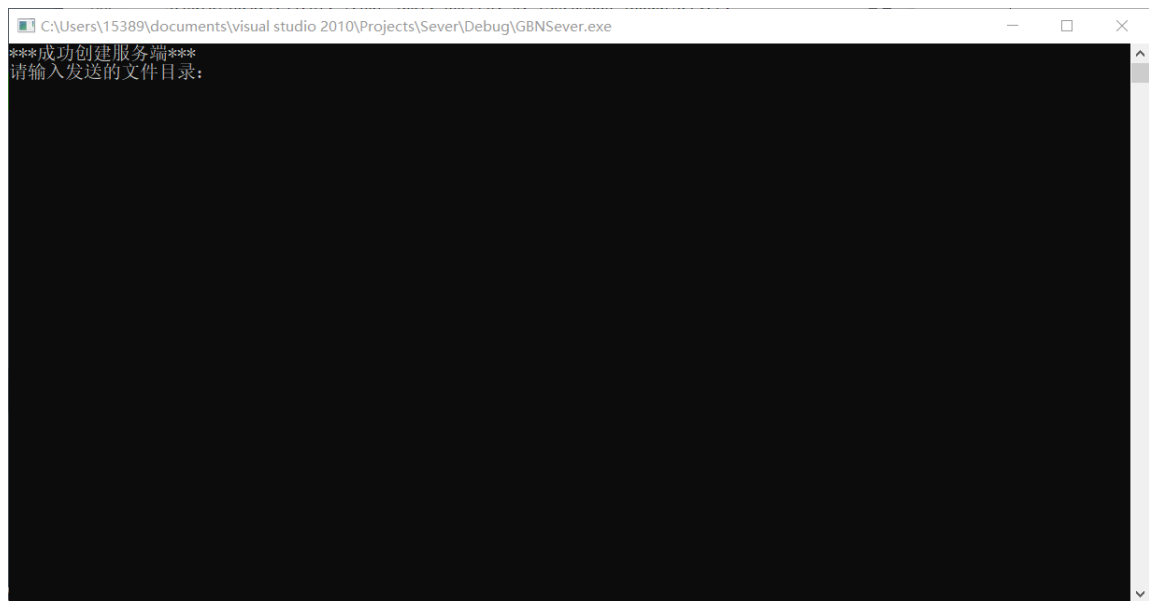


图 6: sever 开始前

传输开始前 client 端显示如图7所示

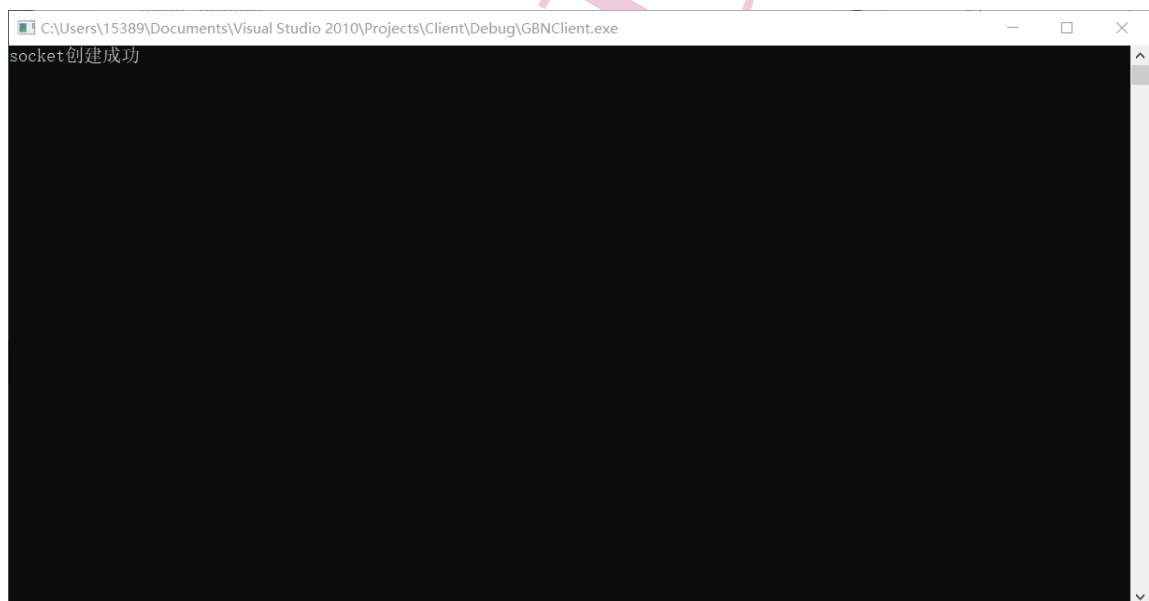


图 7: sever 开始前

传输过程中 sever 端显示如图8所示

```

选择C:\WINDOWS\system32\cmd.exe
当前发送窗口大小: 21440
当前发送窗口大小: 21440
***重传第 325 号数据包***
***重传第 326 号数据包***
***重传第 327 号数据包***
***重传第 328 号数据包***
***重传第 329 号数据包***
***重传第 330 号数据包***
***重传第 331 号数据包***
***重传第 332 号数据包***
***重传第 333 号数据包***
***重传第 334 号数据包***
***重传第 335 号数据包***
***重传第 336 号数据包***
***重传第 337 号数据包***
***重传第 338 号数据包***
***重传第 339 号数据包***
***重传第 340 号数据包***
***重传第 341 号数据包***
***重传第 342 号数据包***
***重传第 343 号数据包***
***重传第 344 号数据包***
当前发送窗口大小: 21440
<<<<<<<<收到了325号数据包的ack<<<<<<<<

=====达到阈值, 进入拥塞避免阶段=====
cwnd= 24      ssthresh=1
当前发送窗口大小: 21440
发送了序列号为 345 的数据包

```

图 8: sever 传输中

传输过程中 client 端显示如图9所示

```

C:\WINDOWS\system32\cmd.exe
***收到第325号数据包***
当前接收窗口大小: 20368
写入后, 当前接收窗口大小: 21440
-----发送对第325号数据包的确认-----
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----325

```

图 9: client 传输中

传输结束后 sever 端显示如图10所示

```

C:\WINDOWS\system32\cmd.exe
***重传第 1815 号数据包***
***重传第 1816 号数据包***
***重传第 1817 号数据包***
***重传第 1818 号数据包***
***重传第 1819 号数据包***
***重传第 1820 号数据包***
***重传第 1821 号数据包***
***重传第 1822 号数据包***
***重传第 1823 号数据包***
***重传第 1824 号数据包***
***重传第 1825 号数据包***
***重传第 1826 号数据包***
***重传第 1827 号数据包***
***重传第 1828 号数据包***
***重传第 1829 号数据包***
***重传第 1830 号数据包***
***重传第 1831 号数据包***
***重传第 1832 号数据包***
当前发送窗口大小: 21440
<<<<<<<收到了1813号数据包的ack<<<<<<<

=====达到阈值, 进入拥塞避免阶段=====
cwnd= 59      ssthresh=1
当前发送窗口大小: 21440
~~~~~
***文件传输完毕***
文件传输时间: 357808ms
吞吐率: 0.0415274Mbps
请按任意键继续. . .

```

图 10: sever 传输后

传输结束后 client 端显示如图11所示

```

C:\WINDOWS\system32\cmd.exe
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
-----不是期待的数据包, 发送了一个重复ack-----1813
当前接收窗口大小: 21440
*****
***文件传输完毕***请按任意键继续. . .

```

图 11: client 传输后

测试文件 1.png 传输后更名为 11.png, 接收文件如图12所示

 11.jpg	2021/12/15 20:18	JPG 图片文件	1,814 KB
--	------------------	----------	----------

图 12: 接收文件

注意此时日期更新为 2021/12/15。

## 五、 实验收获

本次实验深化了我对拥塞控制的理解，进一步熟悉了 socket 套接字的使用，对如何在用户空间实现面向连接的可靠数据传输、进行流量控制、实现拥塞控制有了更深入的理解，对于快速重传、快速恢复、慢启动、拥塞避免都有了具体代码的实现。

NIKU