南开大学 Nankai University

南　开　大　学

网络空间安全学院

计算机网络课程实验报告

基于 UDP 服务设计可靠传输协议并编程实现 3-3

学号：2011897

姓名：任薏霖

年级：2020 级

专业：物联网工程

2022 年 12 月 30 日

一、实验内容

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、实验原理

（一）网络协议原理介绍

1. 本次实验所用 UDP 协议设计为拥塞控制机制，通过 Reno 算法，从而实现既不造成网络严重拥塞又能更快地传输数据的作用，其基本思想如下：

- 目标：既不造成网络严重拥塞，又能提高数据传输速率；
- 带宽监测：接收到 ACK 提高传输速率，发生丢失事件降低传输速率
  - ACK 返回：说明网络并未拥塞，可以继续提高发送速率；
  - 丢失事件：假设所有丢失是由拥塞造成的，则降低发送速率，具体思路如下图所示：

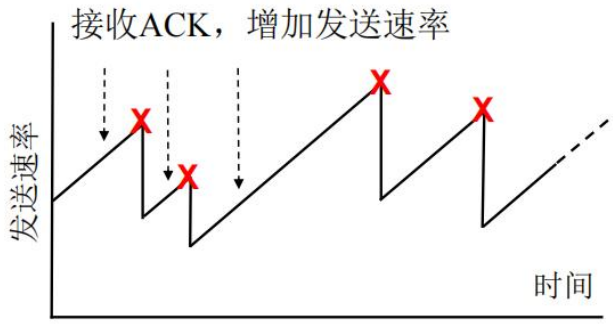

图 1 发送速率与接收 ACK 的关系

- 拥塞控制窗口：采用基于窗口的方法，通过拥塞窗口的增大或减小控制发送速率

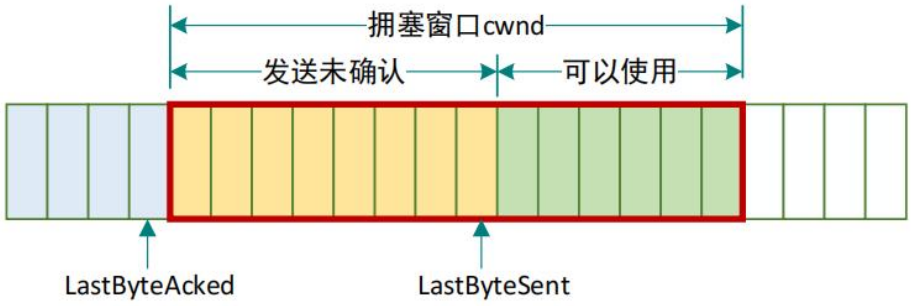

图 2 拥塞窗口

  - 慢启动阶段：当主机开始发送数据时，如果立即有大量数据字节注入到网络中则有可能导致网络拥塞，因此为避免出现网络负荷情况，可以由小到大逐渐增

大拥塞窗口数值；

- 每个 RTT, cwnd 自增 1；
- 每接收到一个 ACK，cwnd 自增 1；
- 当 cwnd 超过慢启动门限阈值时，进入拥塞避免阶段

- 拥塞避免阶段：拥塞窗口达到阈值 ssthresh 时，慢启动阶段结束，进入拥塞避免阶段；
  - 每接收到一个 ACK，$cwnd = cwnd + MSS * \frac{MSS}{cwnd}$ ；
- 快速恢复阶段：当收到三个重复的 ACK 或是超过了 RTT 时间且尚未收到某个数据包的 ACK，Reno 就会认为丢包了，并认定网络中发生了拥塞；
- 丢失检测：
  - 通过超时检测丢失：如果报文接收 ACK 超时，则阈值$ssthresh = cwnd/2$；*窗口大小* $cwnd = 1$，*进入慢启动阶段；*
  - 通过三次重复 ACK 检测丢失（Reno 算法）

无论在慢启动阶段还是拥塞避免阶段，只要发送方判断出现网络拥塞状态（根据未收到确认 ACK 判定），就要把慢启动时的阈值设置为出现拥塞时的发送方窗口大小的一半，并将拥塞窗口 cwnd 设置为 1，重新执行慢启动算法；

Reno 算法的有限状态机如下所示：

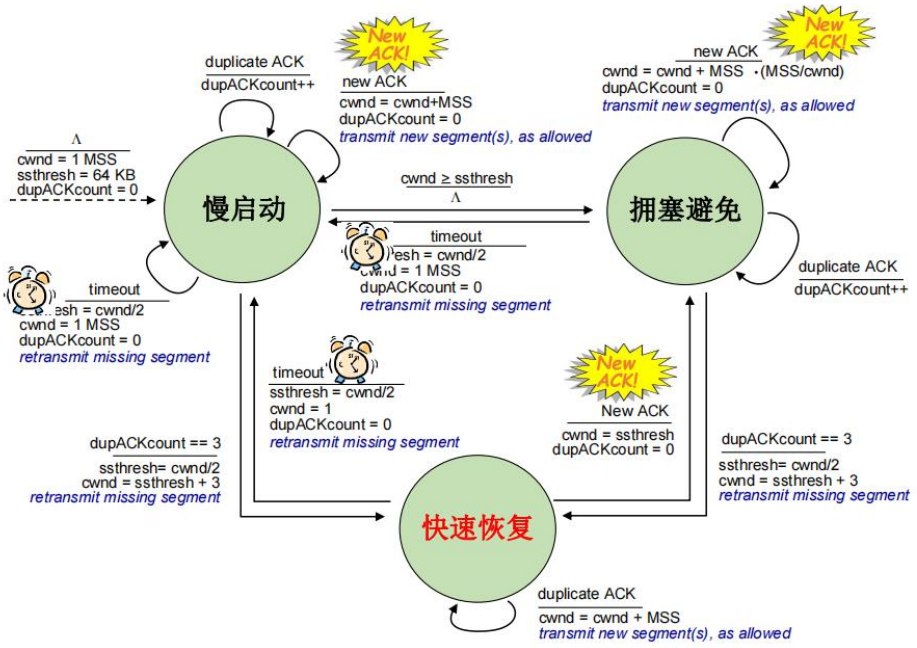

图 3 Reno 算法状态机

（二）协议设计

1. 报文格式，如下图所示：

图 4 报文格式

2. 三次握手 —— 发送端与接收端相连接

- 发送端：发送一条带有标识 SYN 的消息，表示希望建立连接，对方收到后回复对应的 ACK,连接建立成功。当收到 ACK 时，进入状态 1；
- 接收端：收到一条 SYN 消息，回复对应的 ACK,连接建立成功，进入状态 1；

3. 数据传输

- 发送端：选择 Reno 算法进行拥塞控制，具体如下：
  - 设定慢启动阈值 ssthresh，令其初始值为 10；最初窗口大小 cwnd 为 1；连接建立时，首先进入慢启动阶段；
  - 拥塞窗口达到阈值时，慢启动阶段结束，进入拥塞避免阶段；
  - 收到三次重复 ACK 时，进入快速恢复阶段；
  - 在快速恢复阶段，当收到新的 ACK 消息时，进入拥塞避免阶段；
  - 超时情况下，进入慢启动阶段；
- 接收端：采用 GBN 算法对数据包进行接收，其状态机如下所示：

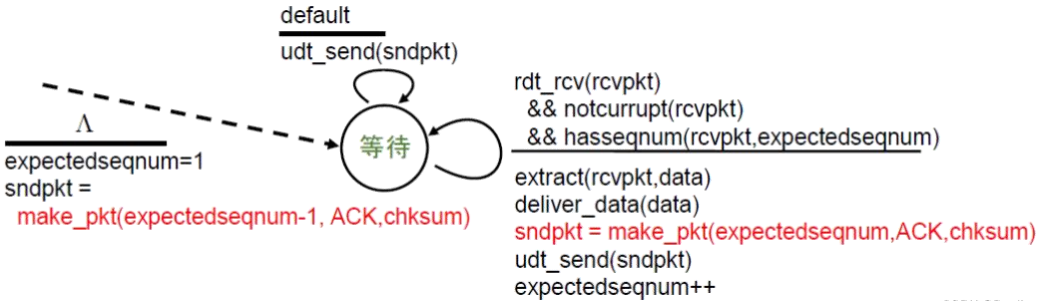

图 5 发送端 GBN 算法接收信息

初始阶段，发送端处于慢启动阶段，一段时间后当窗口大小超过阈值时，则进入拥塞控制阶段；在发送数据过程中，无需等待上一个发送包序号的 ACK=1，而可以继续发送直至窗口大小数量的数据包；当接收端接收到数据包后，进行校验，若校验和无误则向发送端返回

ACK，发送端在接收到 ACK 后调整窗口大小；

在一定时间内，若未收到某一数据包返回的 ACK，则将窗口中所有位于该数据包后已经传输但未得到确认的数据包丢弃，并从该数据包开始进行重传，发送端重新进入慢启动阶段，并调整阈值大小；若再次接收到已确认的报文 ACK，则将其忽略；若连续接收到相同 ACK 或窗口中未确认的报文多于一定数目，则调整窗口大小和阈值；

最后，发送端需要向接收端发送一个 FIN = 1/ACK = 1/SYN = 1 的数据包，代表数据传输结束；同时，接收端在收到包裹后，向发送端返回 ACK = 1，表示已接收到传输结束的信号；

4. 四次挥手 —— 发送端与接收端断开连接

- 发送端：发送一条表示为 FIN 的消息，对方收到后回复对应的 ACK，断开连接成功，退回状态 0；
- 接收端：收到包含标识为 FIN 的消息，返回 ACK，进入状态 0；

三、程序实现

（一）全局变量

在之前实验的基础上，为实现 Reno 算法需要设置新的全局变量，其核心代码如下所示：

```
int dupAck = 0;                 //重复ack个数
const int rwnd = 20;            //接收通告窗口大小，固定值
float cwnd = 1;                 //拥塞窗口大小
int ssthresh = 10;              //慢启动状态阈值
int renostate = 0;              //慢启动-0,拥塞避免-1,快速重传,快速恢复-2
```

（二）消息类型

为方便了解传输信息特征及内容，首先为传输的数据设计消息头从而存储重要信息。在本次实验中，为了保证成功完成文件的传输和下载，需要以二进制形式传输数据。因此，需要考虑消息头中的数据类型的位数。本文关于 DataSize 结构体，设计如下；

- datasize：表示传输的数据长度，共 16 位；
- Checksum：表示校验和，共 16 位，负责校验传输的消息和数据是否被损坏；
- type：表示消息类型，共 16 位，ack = 4 时代表最后一个数据包；
- ack：表示确认序号，共 16 位；
- SEQ：表示序列号，共 16 位，可以表示 0-255

其核心代码如下所示：

```
struct HeadMsg{
    u_short ack;          //确认序号              16位
    u_short SEQ;          //序号                  16位
    u_short datasize;     //数据长度              16位
    u_short checksum;     //校验和                16位
    u_short type;         //ACK标志位             16位    ack=4（最后一个数据包）
    u_short tag;          //标识：SYN=1;FIN=2     16位--------以上为12字节
};
```

（三）三次握手 --- 建立连接

实验 3-3 设计与实验 3-1 中的三次握手一致，其具体流程为：

- 第一次握手：Client 发送 SYN 消息；
- 第二次握手：Server 发送 SYN_ACK 消息；
- 第三次握手：Client 发送 ACK 消息；

（四）计时线程

在 Reno 算法中，计时线程需要加上超时后阈值和拥塞控制窗口大小的变化情况，以及更新 RENO 状态机的状态，其主要代码如下所示：

```
DWORD WINAPI Timer(LPVOID lpParameter){
    //确认消息接收
    while (1){
        //结束计时
        clock_t finish = clock();
        //超时
        if (finish - start > MAX_TIME){
            //超时未收到新的ACK，进入慢启动状态
            ssthresh = cwnd / 2;  //阈值设为窗口大小的一半
            cwnd = 1;             //窗口大小设为1
            dupAck = 0;           //重置重复ack个数
            if (renostate != 0)   //超时则进入慢启动状态
                renostate = 0;
            send_package(buffer);
        }
        //线程结束条件
        ……
    }
    return 0;
}
```

（五）发送消息

1. 以二进制读文件

本文需要以二进制方式读文件并将其保存到缓冲区，其主要代码如下所示：

```
ifstream fin(filename, ifstream::in | ios::binary);
    if (!fin) {
        ……}
    fin.seekg(0, fin.end);     // 指针定位在文件结束处
    length = fin.tellg();      // 获取文件大小（字节）
    fin.seekg(0, fin.beg);     // 指针定位在文件头
    buffer = new char[length];
    memset(buffer, 0, length);
    fin.read(buffer, length); // 读取文件数据到缓冲区buffer中
    filelen = length;
    fin.close();               //关闭文件
```

2. 文件发送

与实验 3-2 相比，本次实验中 window 应该代表实际发送窗口的大小，这取决于接收通告窗口和拥塞控制窗口中的较小值，本文将 rwnd 设置为固定窗口大小 20，其核心代码如下所示：

```
// 实际发送窗口取决于接收通告窗口和拥塞控制窗口中的较小值
int window = 0;
if (rwnd < cwnd)
    window = rwnd;
else  window = cwnd;
```

（六）接收线程

在接收到正确的 ACK 消息后，需要添加 Reno 状态机的三个状态处理，具体可参照图 ，其主要代码如下所示：

初始状态：慢启动阶段，初始化窗口大小 N 为 1，阈值 ssthresh = 10；

```
//接收到正确的消息
if (base <= recv_Msg.ack - 1) {
    if (recv_Msg.SEQ == packnum - 1) {
        flag = 0;
    }
//  慢启动
    if (renostate == 0) {
        cwnd++;       //窗口加1
        dupAck = 0;   //重置冗余ack个数
        base++;
//  窗口大小超过阈值，则进入拥塞避免状态
    `   if (cwnd >= ssthresh) {
            renostate = 1;//拥塞控制状态
        }
```

拥塞避免阶段：每收到一个 ACK，窗口大小 N 增加 1/窗口大小,此时窗口大小呈线性增加；

```
//  拥塞避免状态
    else if (renostate == 1) {
        base++;
        cwnd += 1.0 / cwnd;             //窗口大小N增加1 / 窗口大小
        dupAck = 0;                     //重置冗余ack个数
        ……
    }
```

快速恢复阶段：收到新的 ACK 进入拥塞避免阶段；当接收消息有误时且处于快速恢复阶段时，每收到一个冗余 ACK,窗口大小加 1，窗口增长到大于或者等于阈值 ssthresh 时，进入拥塞避免阶段；

```
//  快速恢复状态
    else if (renostate == 2){
        base++;
        dupAck = 0;         //重置冗余ack个数
        cwnd = ssthresh;
        renostate = 1;      //进入拥塞避免状态
        ……
        Sleep(2);
    }
        start = clock();
```

```
//  处于快速恢复状态 —— 收到冗余ACK
if (renostate == 2){
    cwnd += 1;    // 窗口大小加1
}
```

通过三次重复 ACK 检测丢失并记录失序位置，重传失序信息（Reno 算法）；

```
dis_Location = recv_Msg.SEQ;    //记录失序位置
dupAck++;                       //重复ACK数量++
//  当重复ACK超过3次，进入快速恢复阶段
if (dupAck == 3){
    ssthresh = cwnd / 2;        //阈值设为窗口大小的一半
    cwnd = ssthresh + 3;        //窗口大小设置为阈值加3
    renostate = 2;              //快速恢复阶段
    send_package(buffer);
    Sleep(20);
}
```

（七）异常检查：差错检测

差错检测分为校验和检测和序列号检测两部分；

1. 校验和检测

校验和是消息头中的冗余字段，用来检测数据报传输过程中出现的差错。其计算方法如下所示：

- 将消息头的校验和设置为 0；
- 将消息头和数据看成 16 位序列，不足 16 位补 0；
- 每 16 位相加，溢出部分加到最低位；
- 结果取反

接收端接收到数据时，采用上述方法计算校验和，若校验和全为 0，则证明文件数据包正确；反之，则证明文件数据包损坏。

其代码如下：

```
//校验和计算相关函数
u_short Checksum(u_short* message, int size) {
    int count = (size + 1) / 2;
    u_short* buf = (u_short*)malloc(size + 1);
    memset(buf, 0, size + 1);
    memcpy(buf, message, size);
    u_long sum = 0;
    while (count--) {
        sum += *buf++;
        if (sum & 0xFFFF0000) {
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

2. 序列号检测

接收端每次发送 ACK 消息时，序列号为其最后正确收到的消息的序列号。发送端发送一条序列号 seq=n 的消息，接收端收到后，会发送 ACK 消息确认序列号 seq=n；若发送端的消息损坏，接收端会发送 ACK 消息，此时序列号为 seq=n-1，以此通知发送端消息已损坏。

其代码如下所示：

```
//初始化序列号
int Seq = 0;
//每次成功发送消息后，序列号+1
Seq = (Seq + 1) % 256;
//序列号判断 传输文件是否正确 --- 否 重发 ACK
if (Seq != int(mesr.seq));
```

（八）丢包程序

```
//模拟丢包
BOOL lossInLossRatio(float lossRatio) {
    int lossBound = (int)(lossRatio * 100);
    int r = rand() % 101;
    if (r <= lossBound) {
        cout << "========================= LOSS LOSS LOSS ======================" <<
endl;
        return TRUE;
    }
    return FALSE;
}
```

（九）四次挥手 --- 断开连接

实验 3-3 四次挥手设计与实验 3-1 的四次挥手内容一致，其具体流程为：

- 第一次握手：Client 发送 FIN_ACK 消息；
- 第二次握手：Server 发送 ACK 消息；
- 第三次握手：Server 发送 FIN_ACK 消息；
- 第四次挥手：Client 发送 ACK 消息；

四、程序测试

（一）初始连接

运行程序，可以观察到三次握手的过程并成功建立连接；Client 端输出消息提示用户输入需要传输的文件名：

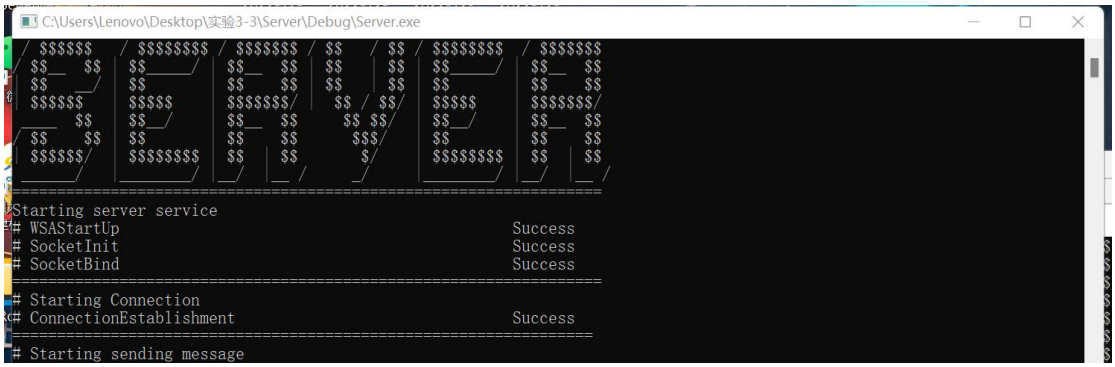

图 6 接收端建立连接

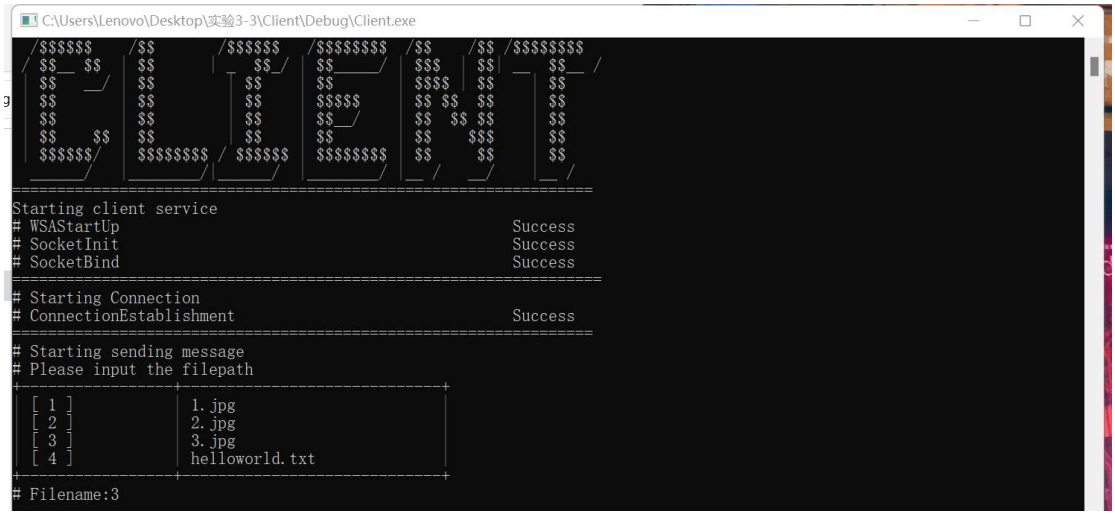

图 7 发送端建立连接

（二）传输数据

根据如下日志可以发现，拥塞控制首先处于慢启动阶段，收到 ACK 后消息窗口+1；

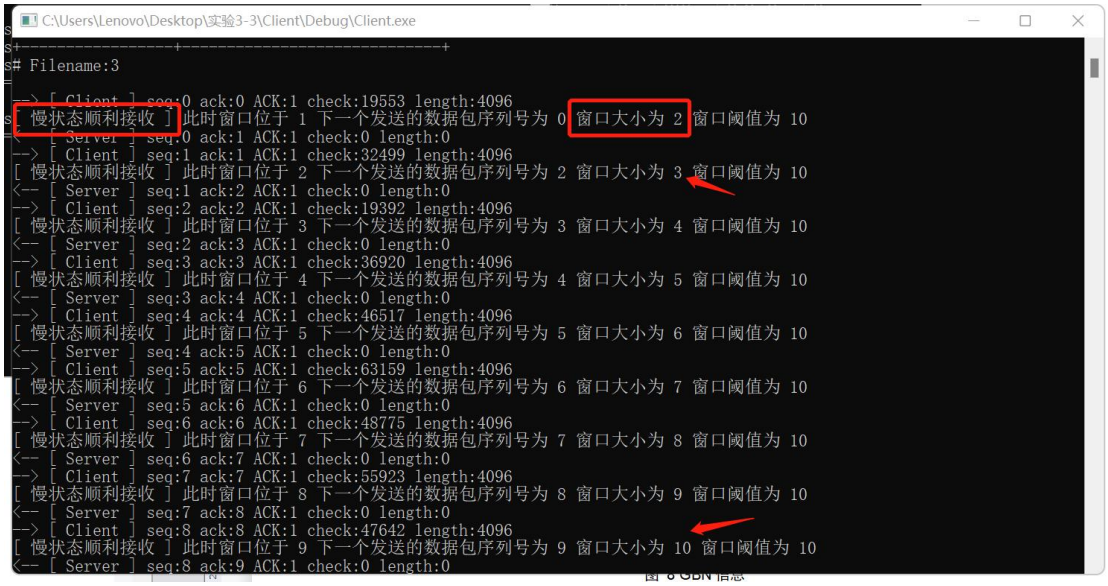

图 8 慢启动阶段

而当窗口大小 *cwnd* 超过阈值 *ssthresh* 时，程序进入拥塞避免阶段，窗口大小增长减

慢，如下：

图 9 慢启动阶段转入拥塞避免阶段

通过人为设置丢包程序，接收端接收到损坏的数据包，因此丢弃了序列号为 49 的数据包，并且一直在等待它，如下图所示：

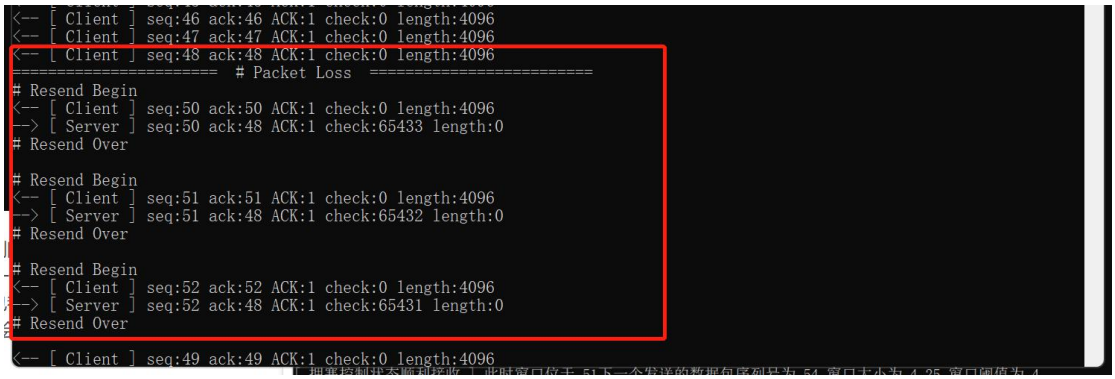

图 10 接收端发生丢包

而发送端还不知道，依然在流水线地发送后面的数据包，所以接收端将后面来的数据包都丢弃了，并返回 ACK=48 的消息。发送端一直收到重复的 ACK 消息，当重复 ACK 次数为 3 时，发送端会进入快速恢复阶段，此时 ssthresh = cwnd/2；cwnd = ssthresh+3，如下图所示：

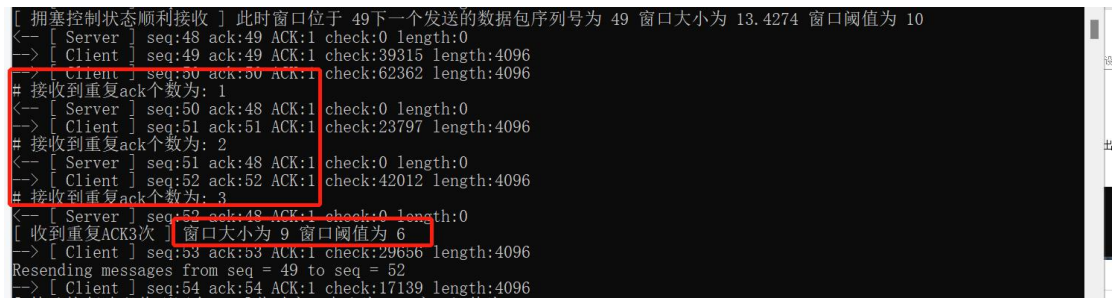

图 11 dupACK = 3 时转入快速恢复阶段

发送端重传所有未被确认的数据包，也就是从序列号 49 到序列号 52，此时拥塞控制会收到冗余的 ACK，因此 cwnd++，如下图所示：

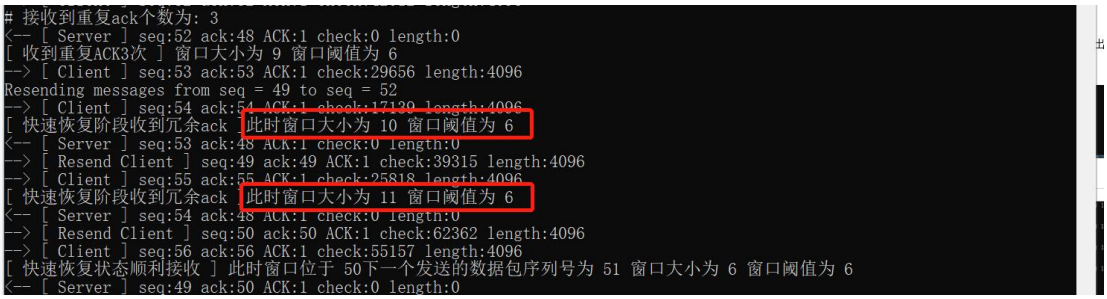

图 12 重传中收到冗余 ACK

当重传结束并收到新的 ACK 时，将会从快速恢复状态转入拥塞避免状态，而 cwnd 设置为阈值大小，如下图所示：

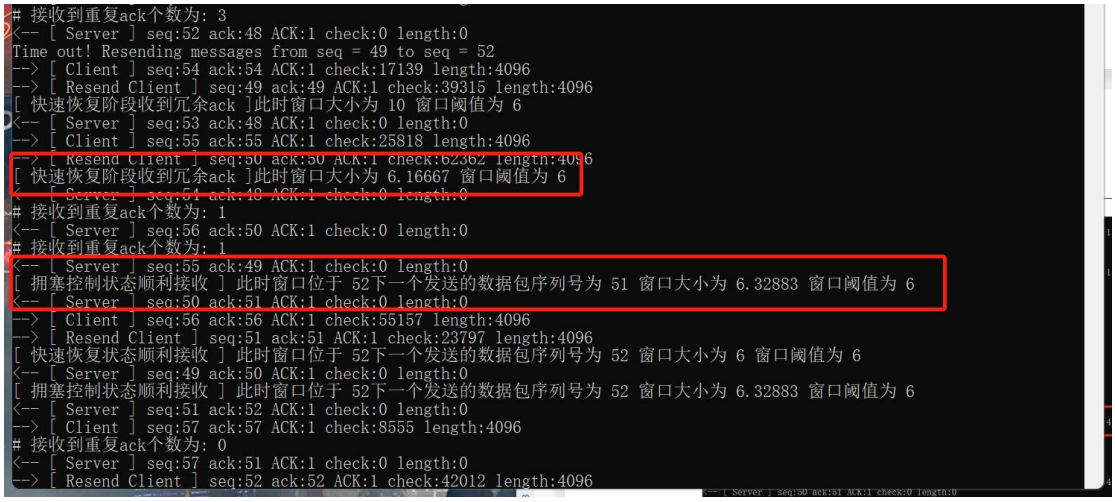

图 13 快速恢复阶段转入拥塞避免阶段

而接收端也已接收到所有重传的数据包，如下图所示：

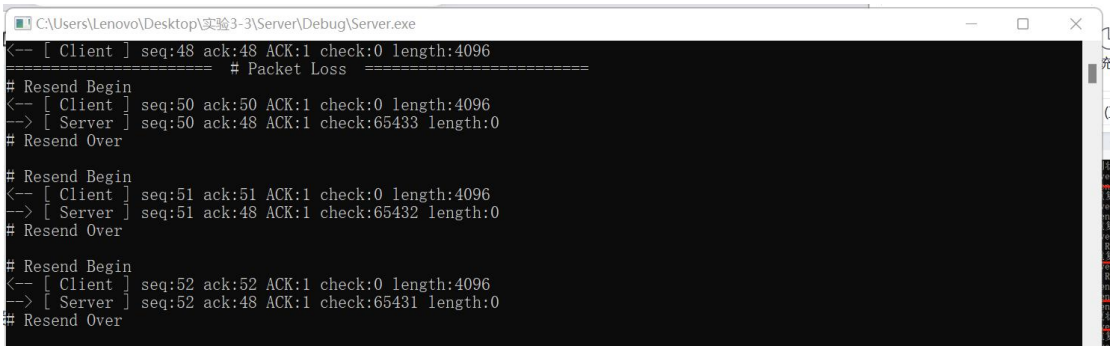

图 14 重传

（三）断开连接

最后，所有数据包都发送完毕并被成功确认后，发送端输出提示消息，并输出传输时间和吞吐率，并成功断开连接，如下图所示：

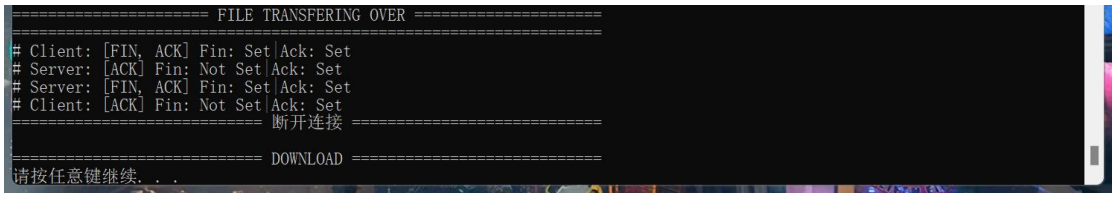

图 15 接收端断开连接

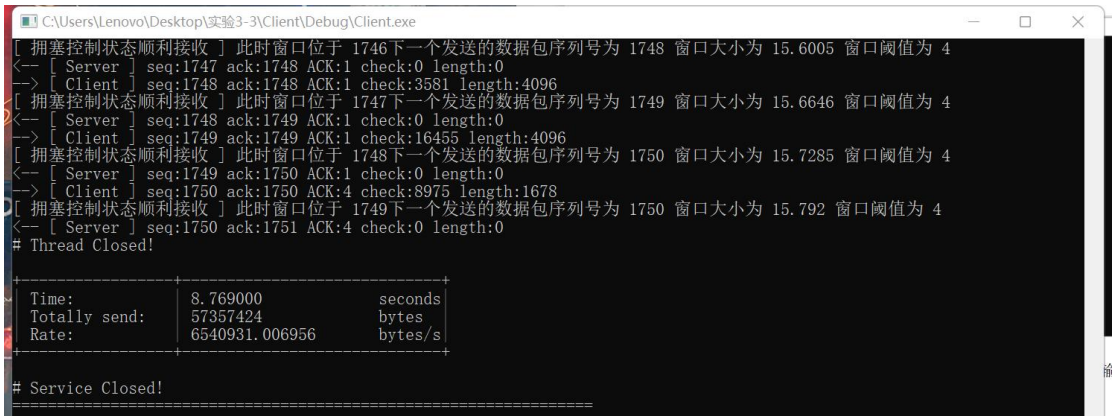

图 16 发送端断开连接

（四）传输结果

其传输结果如下图所示：

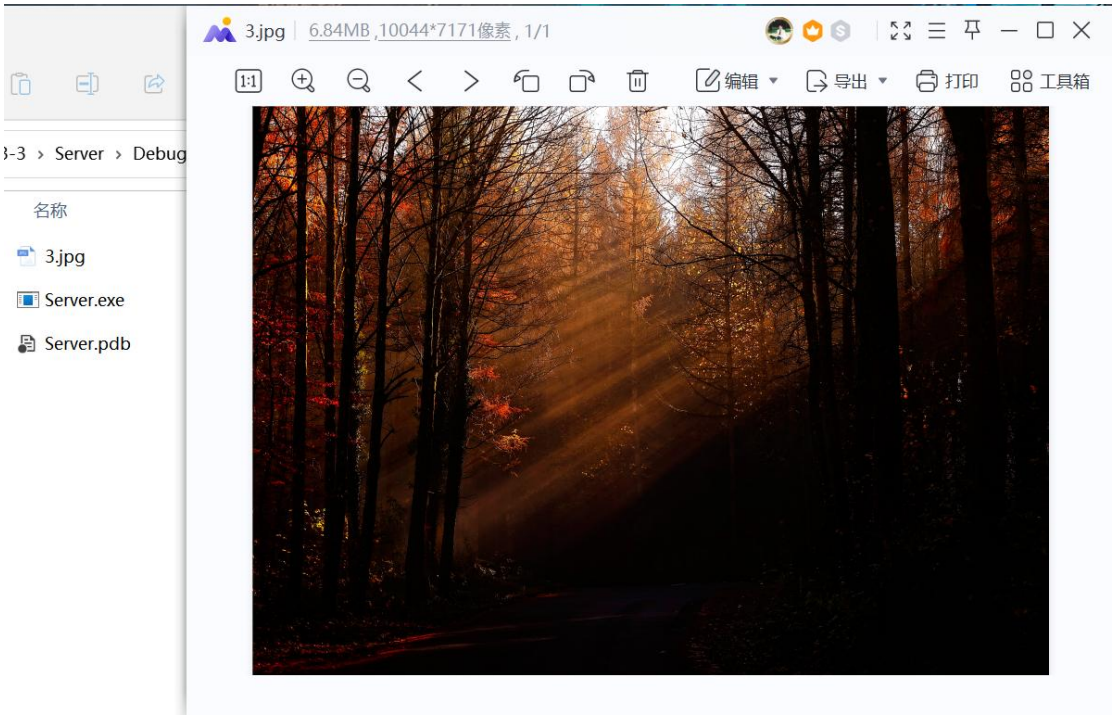

图 17 传输结果