南 开 大 学

计算机网络

1811507 文静静

2020 年 12 月 22 日

基于 UDP 服务设计可靠传输协议并编程实现 3-3

一、实验内容

在任务 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) 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) 完成详细的实验报告。

(6) 编写的程序应结构清晰，具有较好的可读性。

(7) 提交程序源码和实验报告。

三、协议设计

协议特点：

- 发送方和接收方有握手过程，确认连接；
- 数据可以切包传输，保存序列号；
- 数据报有差错检测功能，正确接收返回确认；
- 数据报使用拥塞控制窗口传输机制；
- 数据报使用 ACK 累计确认机制，传输速率增长；

- 发送端有超时重传，解决数据包丢失的情况；
- 发送方和接收方有挥手过程，断开连接。

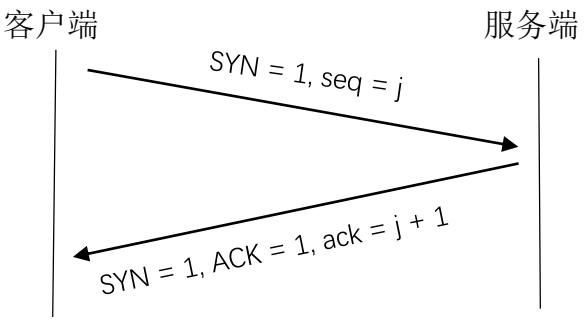
数据包格式：

- seq 和 ack：8 位，用于握手确认连接和发送切片时保存序列号；
- 标志位：ACK，SYN/FIN，用于握手确认连接和发包确认接收以及挥手断开连接；
- 长度：不包括头部，只保存数据长度。

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
序号(seq)								确认序号(ack)							
ACK								SYN/FIN							
长度(length)															
校验和(checksum)															
数据(data)															

握手确认连接：

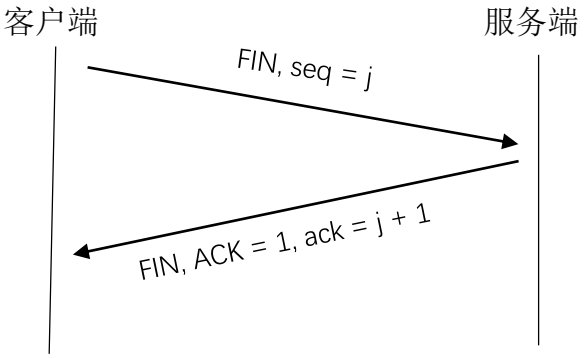
单向传输，二次握手确认连接



- 步骤一：客户端发送 SYN 段
- 步骤二：服务端接收 SYN 段，回送 SYN+ACK 段

挥手断开连接：

单向传输，二次挥手断开连接



- 步骤一：客户端发送 FIN 段
- 步骤二：服务端接收 FIN 段，回送 FIN+ACK 段

数据报的差错检测：

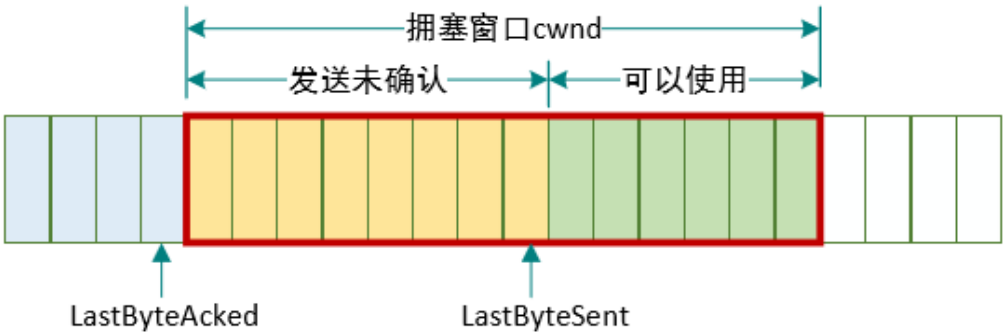
- 发送端：对要发送的 UDP 数据报，校验和域段清 0，将数据报用 0 补齐为 16 位整数倍，将数据报看成 16 位整数序列，进行 16 位二进制反码求和运算计算校验和，并将校验和结果取反写入校验和域段；
- 接收端：对接收到的 UDP 数据报，将数据报用 0 补齐为 16 位整数倍，将数据报看成 16 位整数序列，进行 16 位二进制反码求和运算计算校验和，并将校验和结果取反，如果取反结果为 0，则没有检测到错误，否则，数据报存在差错。

数据报切包发送：

- 数据报大小上限为 packetsize(测试时为 1500)字节，数据报头部为固定 10 个字节，数据域段上限为(packetsize-10)字节，当要传输的数据长度大于(packetsize-10)字节时，需要进行切包发送。
- 发送端使用 seq 表示当前数据包的序列号，使用 ack 表示该数据包之后是否还有数据包，如果还有，ack 为 1，如果现在发送的数据包为传输文件的最后一个数据包，那么 ack 为 0。
- 接收端接收数据包时检测收到的包的序列号是否为上一次收到包的序列号加一，避免重复接收，如果收到重复接受的数据包将直接丢弃。
- 接收端接收到 ack 为 0 的数据包，表示为传输文件的最后一个包。

拥塞窗口发送：

- 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



- 拥塞控制算法：New RENO 算法
  - 慢启动阶段：初始拥塞窗口：cwnd=1

    每收到一个 ACK，cwnd 增 1

    当连接初始建立或者超时时，进入慢启动阶段
  - 拥塞避免阶段：阈值 ssthresh：拥塞窗口达到该阈值时，进入拥塞避免

阶段；

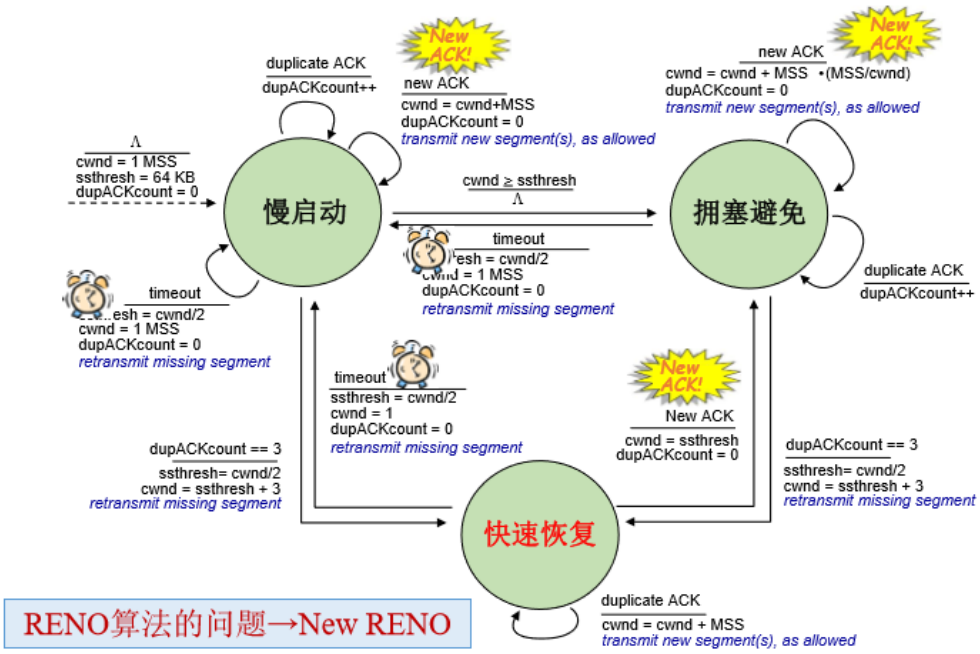
每个 RTT，cwnd 增 1；

- 快速恢复阶段：当收到三次重复 ACK 时，进入快速恢复阶段；

  每收到一个重复 ACK，cwnd 增 1；

  出现超时之后进入慢启动阶段；

  收到新的 ACK 进入拥塞避免阶段。



四、代码思路

- **发送端和接收端数据报头：**8 个字节，保存数据报序列号 seq、下一个数据报序列号 ack 和标志位 ACK, SYN/FIN、数据长度、校验和。

```
void setPacketHead() {
    sendBuf[0] = 0;//seq
    sendBuf[1] = 0;//ack
    sendBuf[2] = 0;//ACK
    sendBuf[3] = 0;//1 for SYN; 2 for FIN
    sendBuf[4] = 0;//长度高8位
    sendBuf[5] = 0;//长度低8位
    sendBuf[6] = 0;//校验和高8位
    sendBuf[7] = 0;//校验和低8位
}
```

发送端握手请求：

```
char sendBuf[8] = { 0 };
char receiveBuf[8] = { 0 };
seq = rand() % 100;
sendBuf[0] = seq;
sendBuf[3] = 2;//FIN
u_short buf[5] = { 0 };
int i;
for (i = 0;i < 8;i += 2) {
    buf[i / 2] = ((u_char)sendBuf[i] << 8);
    if ((i + 1) < 8) {
        buf[i / 2] += (u_char)sendBuf[i + 1];
    }
}
u_short checks = checksum(buf, i / 2);
sendBuf[6] = (u_char)(checks >> 8);//校验和
sendBuf[7] = (u_char)(checks & 0xFF);//校验和
//发送握手报文
sendto(sendSocket, sendBuf, 8, 0, (SOCKADDR*)&receiveaddr, len);
```

接收端握手确认：

循环接收连接请求报文，通过判断数据报的 SYN 标志位是否为 1 来判断连接请求报文，收到请求报文之后，检验校验和，如果校验和为 0，则回送握手确认报文，将 ack 设置为 seq+1，并将标志位 SYN 和 ACK 都置 1。如果校验和不对，或者捕获到不是 SYN 请求连接报文，将显示握手失败，退出。

```
while (true) {
    res = recvfrom(receiveSocket, receiveBuf, 8, 0, (SOCKADDR*)&sendaddr, &len);
    if (res == SOCKET_ERROR) {
        Sleep(2000);
        continue;
    }
    if (receiveBuf[3] == 1) {//即数据报的SYN标志位为1，标志握手报文
        //设置数据包发送报头
        setPacketHead(sendBuf);
        u_short buf[5] = { 0 };
        int i;
        for (i = 0;i < 8;i += 2) {
            buf[i / 2] = (u_char)receiveBuf[i] << 8;
            if ((i + 1) < 8) {
                buf[i / 2] += (u_char)receiveBuf[i + 1];
            }
        }
        u_short checks = checksum(buf, i / 2);
        if (checks == 0) {//校验和为0，数据包传输正确
            //获取数据报中的seq
            seq = (u_char)receiveBuf[0];
            sendBuf[0] = seq;
            //确认序号ack=seq+1
            ack = seq + 1;
            sendBuf[1] = ack;
            //标志位SYN和ACK置1
            sendBuf[2] = 1;//ACK
            sendBuf[3] = 1;//SYN
            sendBuf[6] = checks >> 8;//校验和
            sendBuf[7] = checks & 0xFF;//校验和
            //发送握手确认报文
            sendto(receiveSocket, sendBuf, 8, 0, (SOCKADDR*)&sendaddr, len);
            cout << "shake hand successfully" << endl;
            return 1;
        }
        else {
            cout << "shake hand failed" << endl;
            break;
```

发送端收到连接确认报文：

循环接收握手确认报文，如果报文的 SYN 和 ACK 位均为 1，表示握手确认报文，获取报文的 ack，如果 ack=seq+1，则握手成功，否则失败。

```
while (true) {
    res = recvfrom(sendSocket, receiveBuf, 8, 0, (SOCKADDR*)&receiveaddr, &len);
    if (res == SOCKET_ERROR) {
        Sleep(2000);
        continue;
    }
    if (receiveBuf[3] == 1 && receiveBuf[2] == 1) {
        //获取确认报文的ack
        ack = receiveBuf[1];
        //如果ack=seq+1，则握手成功
        if (seq + 1 == ack) {
            cout << "shake hand successfully" << endl;
            return 1;
        }
        else {
            cout << "shake hand failed" << endl;
            break;
        }
    }
    else {
        cout << "shake hand failed" << endl;
        break;
    }
}
return 0;
```

发送端发送数据：

如果建立连接成功，那么发送端就可以开始发送数据报文。

➢ 读取文件

根据传入的文件路径，打开文件，获取大小并读取文件内容。

```
ifstream in(filename, ifstream::in | ios::binary);
in.seekg(0, in.end);//将指针定位在文件结束处
length = in.tellg();//获取文件大小
in.seekg(0, in.beg);//将指针定位在文件头
buffer = new char[length];
memset(buffer, 0, sizeof(char) * length);
in.read(buffer, length);//读取文件数据到缓冲区buffer中
in.close();//关闭文件
totallength += length;//将该文件长度加入总传输数据长度
//计算需要传输的包数
pnum = length / (packetsize - 8) + ((length % (packetsize - 8)) > 0 ? 1 : 0);
```

➢ 发送数据包

✧ 当 nextseqnum < base + cwnd && nextseqnum < pnum 的时候，可以继续发送数据报。获取数据，设置序列号，校验和计算并且发送数据包。base 指向已确认收到数据包的下一个，nextseqnum 指向已发送未确认数据包的下一个；

```
setPacketHead();
//计算数据包长度
int ll = min((packetsize - 8), length - nextseqnum * (packetsize - 8));
sendBuf[4] = ll >> 8;//长度
sendBuf[5] = ll & 0xFF;//长度
//从缓冲区获取数据
for (int i = 0;i < ll;i++) {
    sendBuf[i + 8] = buffer[nextseqnum * (packetsize - 8) + i];
}
//如果是最后一个包，将ack置为0
if (nextseqnum == pnum - 1) {
    ack = 0;
}
else {
    ack = 1;
}
//将nextseqnum与255按位与得到的seq保存到数据报头中
sendBuf[0] = nextseqnum & 0xFF;
sendBuf[1] = ack;
//进行校验和计算
u_short* buf = new u_short[packetsize / 2 + 1];
memset(buf, 0, packetsize / 2 + 1);
int j;
for (j = 0;j < 8 + ll;j += 2) {
    buf[j / 2] = ((u_char)sendBuf[j] << 8);
    if ((j + 1) < 8 + ll) {
        buf[j / 2] += (u_char)sendBuf[j + 1];
    }
}
u_short checks = checksum(buf, j / 2);
sendBuf[6] = checks >> 8;//校验和
sendBuf[7] = checks & 0xFF;//校验和
//发送数据报
sendto(sendSocket, sendBuf, packetsize, 0, (SOCKADDR*)&receiveaddr, len);
if (base == nextseqnum) {//打开定时器
    start = clock();
}
nextseqnum++;
if (nextseqnum == 1) {// 第一个包之后打开接收和定时器线程
    CreateThread(NULL, 0, &TimerThread, NULL, 0, 0);
    CreateThread(NULL, 0, &RevThread, NULL, 0, 0);
}
```

✧ 设置发送结束情况：base == nextseqnum && nextseqnum == pnum

接收端接收数据：

➢ 接收数据

循环接收数据报，如果收到的数据报的第 4 个字节为 2 表示为 FIN 报文，即挥手申请断开连接，如果收到的数据报的第 4 个字节不为 1(SYN)，2(FIN)，并且数据报序列号连续，将对数据报进行处理，获取长度序列号，计算校验和。

```
if (receiveBuf[3] != 2 && receiveBuf[3] != 1 && seq % 256 == (u_char)receiveBuf[0]) {
    setPacketHead(sendBuf);
    u_short* buf = new u_short[packetsize / 2 + 1];
    memset(buf, 0, packetsize / 2 + 1);
    int length = ((u_char)receiveBuf[4] << 8) + (u_char)receiveBuf[5];
    int i;
    for (i = 0;i < 8 + length;i += 2) {
        buf[i / 2] = ((u_char)receiveBuf[i] << 8);
        if ((i + 1) < 8 + length) {
            buf[i / 2] += (u_char)receiveBuf[i + 1];
        }
    }
    u_short checks = checksum(buf, i / 2);
    if (checks == 0) {
        seq = (u_char)receiveBuf[0] + 1;//期望序号
        ack = (u_char)receiveBuf[1];
        char* buffer = new char[length];
        for (int j = 0;j < length;j++) {
            buffer[j] = receiveBuf[8 + j];
        }
        ofstream out(&filename, ifstream::out | ios::binary | ios::app);
        out.write(buffer, length);
        out.close();
        sendBuf[0] = (u_char)seq % 256;
        sendBuf[1] = (u_char)ack;
        sendBuf[2] = 1;//ACK
        sendto(receiveSocket, sendBuf, 8, 0, (SOCKADDR*)&sendaddr, len);
        if (ack == 0) {
            cout << "receive the file" << filename << " successfully" << endl;
            seq = 0;
            ack = 1;
            k++;
            break;
        }
    }
}
```

如果收到的是期望序号的数据包，就接收，如果检验和为 0，表示数据报传输正确，并将数据报中的数据写入接收文件中，并设置 ACK 为 1，期望序号加 1，发送 ACK 确认报文；如果检验和不为 0，那么直接丢弃。如果发送的包为最后一个包(ack=0)，显示传输成功，将序列 seq 设为 0，关闭打开的文件，标识文件个数

的 k 自增，然后跳出接收循环。如果收到的不是期望序号，就会发送重复序号 ACK。

发送端超时线程：

➢ 超时检测：

不断循环检测是否超时，如果超时，进入慢启动阶段，重新设置阈值和窗口并重传所有已发送未确认的包，即从 base 到 nextseqnum-1 的包；

如果窗口超过了阈值，进入拥塞避免阶段；

如果触发了结束条件，即 base == nextseqnum && nextseqnum == pnum，将结束线程。

```
DWORD WINAPI TimerThread(LPVOID lpParameter) {
    while (1) {
        last = clock();
        if (last - start >= 500) {//超时进入慢启动状态
            ssthresh = cwnd / 2;// 设置阈值为窗口一半
            cwnd = 1;
            flag = 0;//慢启动
            dupACKcount = 0;
            resend(base, nextseqnum);// 重传未确认数据包
            start = clock();
        }
        if (cwnd >= ssthresh) {// 当窗口超过阈值，进入拥塞避免阶段
            flag = 1;
        }
        if (base == nextseqnum && nextseqnum == pnum) {
            break;
        }
    }
    return 0;
}
```

发送端接收线程：

如果收到期望的 ACK，判断现在所处的状态，如果处于拥塞控制状态，并判断 segCount 的大小（segCount 表示在当前拥塞阶段一个 RTT 应该收到的 ACK 数量，即当前窗口大小，只有当整个 RTT 的数据包都确认收到，窗口才能增 1）。如

果 segCount 未达到窗口大小，更新 segCount 和 base 的值，将重复 ACK 数量清 0，并重启定时器；如果 segCount 达到窗口大小或者处于慢启动状态或者快速恢复状态，更新 base，清零 segCount 和重复 ACK 数量，窗口增一，如果是在快速恢复状态，收到 ACK 将进入拥塞避免状态。

```
if (receiveBuf[2] == 1) {//当收到ACK
    //如果收到的ACK为期待ACK
    if (((base + 1) % 256 <= (u_char)receiveBuf[0] && (u_char)receiveBuf[0] <= (nextseqnum % 256))
        || ((base + 1) % 256 <= (u_char)receiveBuf[0] && ((base + 1) % 256) >= (nextseqnum % 256))) {
        if (flag == 1 && segcount < cwnd) {
            segcount = segcount + (int)((u_char)receiveBuf[0]) - (base + 1) % 256 + 1;
            if (segcount >= cwnd) {
                segcount = segcount - cwnd;
                cwnd = cwnd + 1;
            }
            base = base + (int)((u_char)receiveBuf[0]) - (base + 1) % 256 + 1;
            dupACKcount = 0;
            start = clock();
        }
        else {
            base = base + (int)((u_char)receiveBuf[0]) - (base + 1) % 256 + 1;
            segcount = 0;
            dupACKcount = 0;
            cwnd = cwnd + 1;
            if (flag == 2) {
                cwnd = ssthresh;
                flag = 1;
            }
            start = clock();
        }
    }
```

如果收到的是重复 ACK，如果在快速恢复阶段，窗口增一，否则重复 ACK 数量增一，当收到三个重复 ACK 时，设置窗口和阈值，进入快速恢复阶段。

```
    else if (base % 256 == (u_char)receiveBuf[0]) {//如果是重复ACK
        if (flag == 2) {
            cwnd = cwnd + 1;
        }
        else {
            dupACKcount++;
        }
    }
    if (dupACKcount == 3) {//三次重复ACK进入快速恢复阶段
        ssthresh = cwnd / 2;
        cwnd = ssthresh + 3;
        flag = 2;//进入快速恢复阶段
        //cout << "快速恢复" << endl;
        resend(base, nextseqnum);
    }
```

发送端挥手请求：

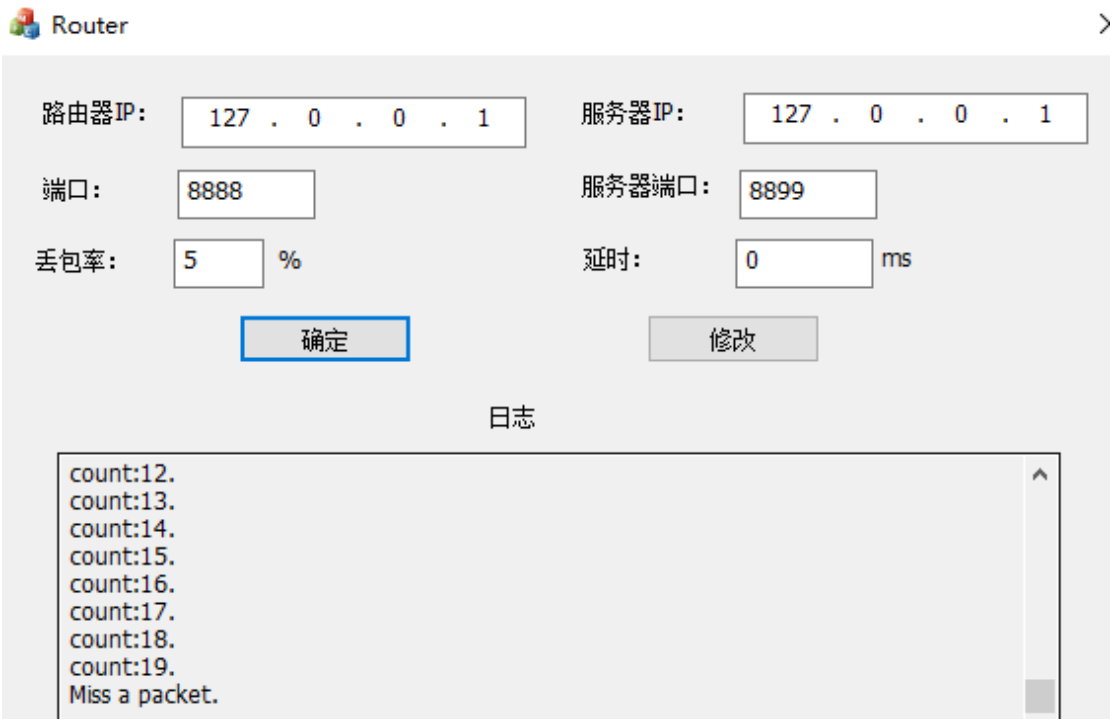
过程和握手相似，只将 SYN/FIN 标志位置 2，表示 FIN，即可发送请求。

接收端挥手确认：

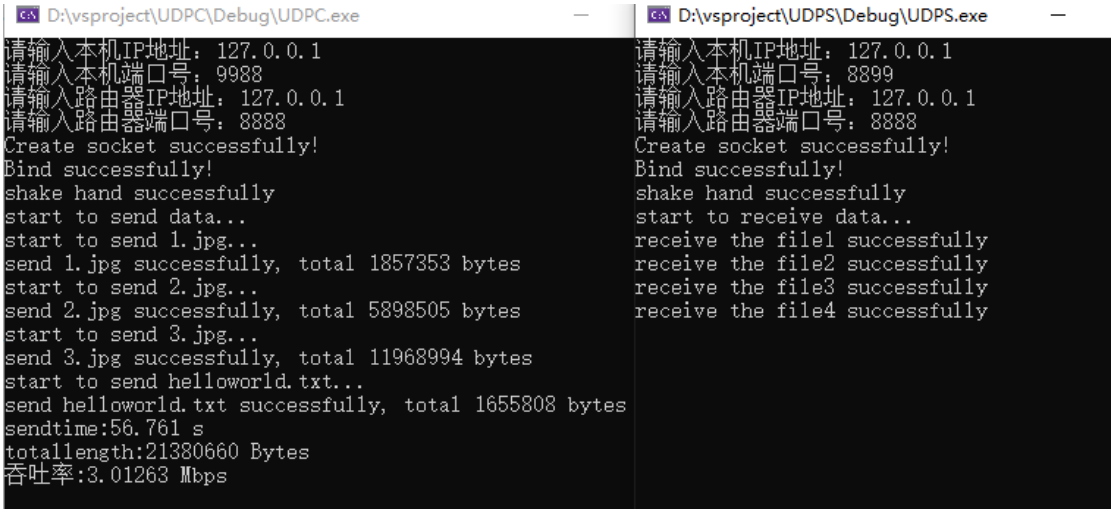
接收端挥手确认和握手确认相识，只将 SYN/FIN 标志位置 2，表示 FIN，即可发送确认报文。

五、实验结果

打开路由器：设置路由器 IP 和端口，并设置丢包率为 5%



发送端和接收端收发结果：

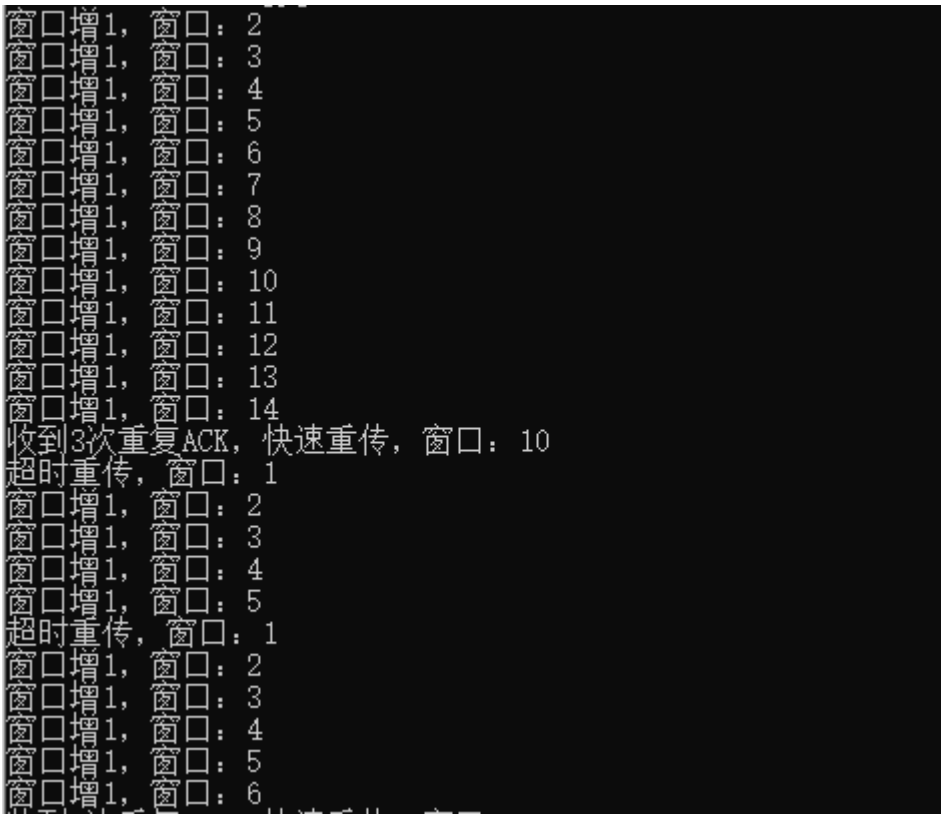


如图，传输四个测试文件的时间为 56.761 s，

总传输数据大小为 21380660 Bytes，

算得的平均吞吐率为 3.01263 Mbps。

监测窗口大小（阈值为 16）



如图，窗口随不同状态发送变化（收到 3 次重复 ACK 时，阈值=窗口/2，窗

口=阈值+3，超时重传时，窗口=1，阈值=窗口/2）。