计算机网络实验报告3-3

姓名：高祎珂　　学号：2011743

实验要求

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。

实验原理

1、数据报套接字UDP:

UDP是User Datagram Protocol的简称,中文名是用户数据报协议,是OSI参考模型中的传输层协议,它是一种无连接的传输层协议,提供面向事务的简单不可靠信息传送服务.

2、建立连接:

根据TCP的三次握手和四次挥手原则，进行发送端与接收端的连接和断连。

3、流水线协议

在确认未返回之前允许发送多个分组。

4、滑动窗口

滑动窗口(Sliding window)是一种流量控制技术。如果网络通信中，通信双方不会考虑网络的拥挤情况直接发送数据，由于大家不知道网络拥塞状况，同时发送数据，则会导致中间节点阻塞掉包，谁也发不了数据， 所以就有了滑动窗又机制来解决此问题。 TCP中采用滑动窗口来进行传输控制，滑动窗又的大小意味着接收方还有多大的缓冲区可以用于接收数据。发送方可以通过滑动窗口的大小来确定应该发送多少字节的数据。当滑动窗口余量为0时，发送方一般不能再发送数据报，但有两种情况除外，一种情况是可以发送紧急数据，例如，允许用户终止在远端机上的运行进程。另一种情况是发送方可以发送一个1字节的数据报来通知接收方重新声明，新声明它希望接收的下一字节及发送方的滑动窗口大小。

5、拥塞控制

拥塞控制是一种用来调整传输控制协议(TCP)连接单次发送的分组数量的算法。它通过增减单次发送量逐步调整，使之近当前风络的承载量，如果单次发送量为1，此协议就退化为停等协议，单次发送量是以字节来做单位的，但是如果假没TCP每次传输都是按照最大报文段来发送数据的，那么也可以把数据包个数当作单次发送量的单位，所以有时我们说单次发送量增加1也就是增加相当于1个最大报文段的字节教。
网络拥建现象是指到达通信子网中某一部分的分组数量过多 使得该分网络来不及处理,以致引 起这部分乃至整个网络性能下的现象 严重时其至会导致网络通信业务陷入停顿,即出现死锁现象。拥塞控制是处理网络拥塞现象的一种机制。

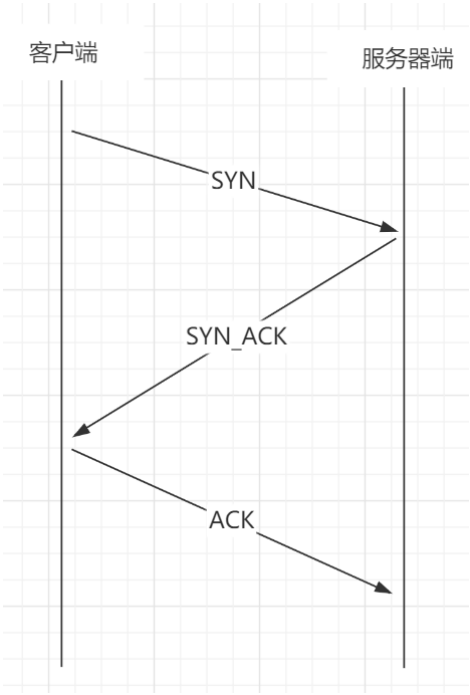
协议设计

- **报文格式**

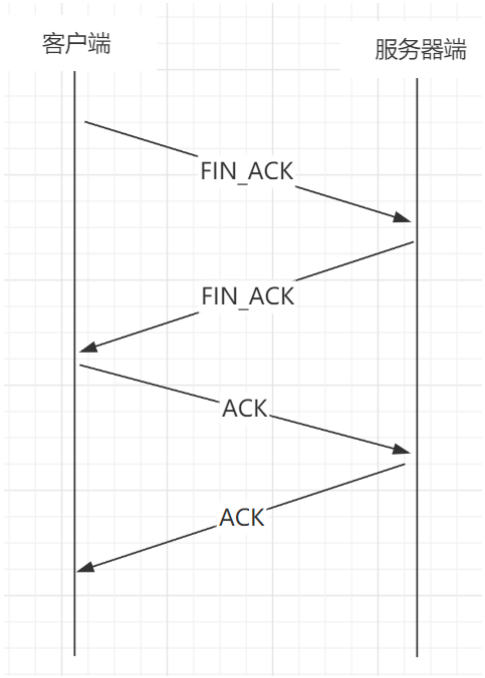
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
序列号															
数据长度															
					NAK							EOF	FIN	ACK	SYN
校验和															
设定大小的数据															

前16位为数据长度，用于记录数据区大小，17-32位位校验和，用于检验传输的正确性，33-40为标志位，40-48位为传输的数据包的序列号。

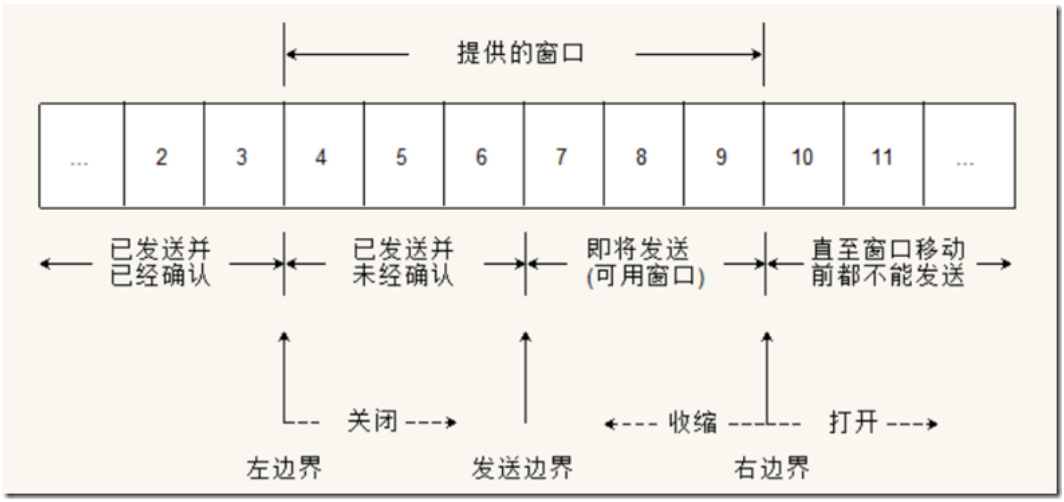
- **连接与断开**
  - **三次握手**



  - **四次挥手**



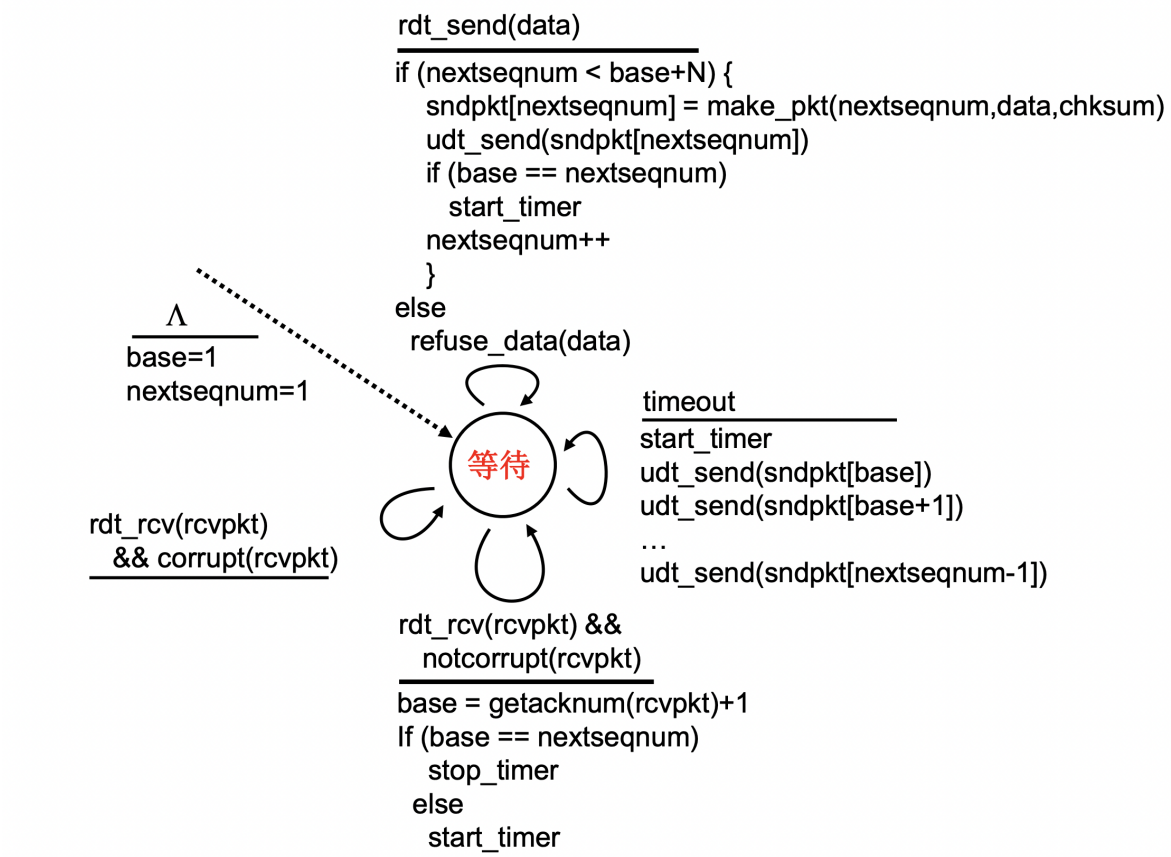
- **滑动窗口**



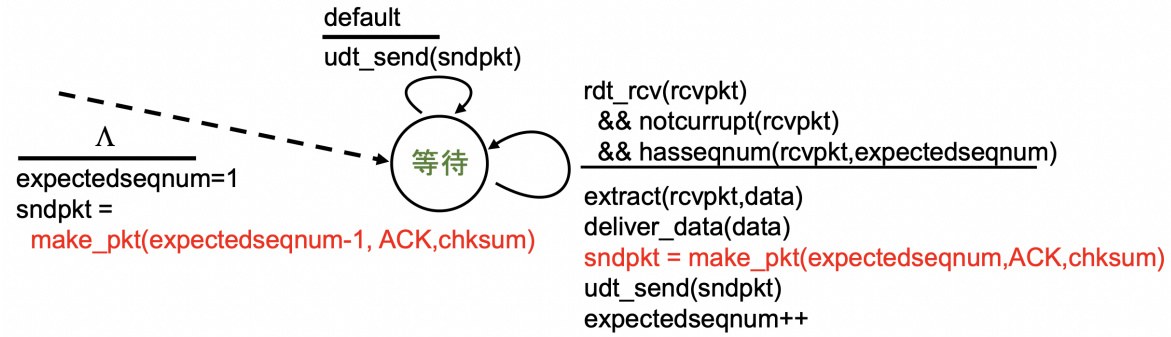
窗口分为左边界、发送边界和右边界，窗口大小**动态调整**，随着接收状态而改变。窗口左边界左侧为已经发送并得到确认的数据，左边界到发送边界的数据为已发送但未得到确认的数据，发送边界到右边界为等待发送的数据，右边界右侧为不可发送的数据。

发送端状态机

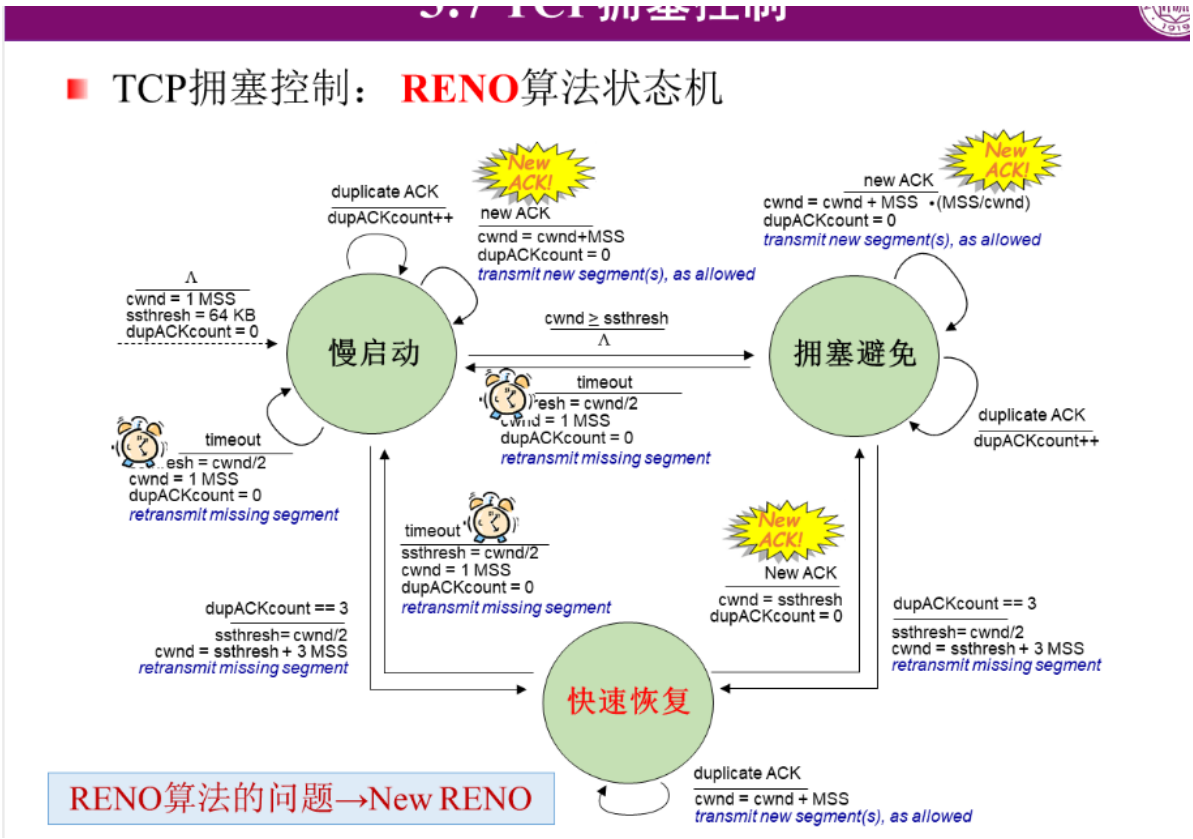
使用rdt3.0实现可靠数据传输



接收端状态机



拥塞控制状态机



这里的cwnd即为发送端接收端状态机中的N。

代码实现(握手，挥手，接收端与上一个实验基本相同)

一些相同的代码不再展示，上次使用的非阻塞，这次改变了策略，使用多线程，因此在发送端发送接收数据包的过程，代码改动较大，主要设计思路是使用**3个线程**，分别执行发送数据包，接收数据包，超时重传数据包。使用一个数据记录每个数据包的发送时间进行超时判断。

数据包

```
struct HEADER {
    unsigned int Seq;
    unsigned short datasize;
    unsigned short Flags;
    unsigned short Checksum;
    char content[MAXSIZE];
    HEADER() {
        this->Seq = 0;
        this->datasize = 0;
        this->Flags = 0;
        this->Checksum = 0;
        for (int i = 0; i < MAXSIZE; i++)
            this->content[i] = 0;
    }
}
```

发送端发送数据包线程

```
void SEND()
{
    //时间列表初始化
    for (int i = 0; i < 500; i++)
```

```
        time_list[i] = 0;

    HEADER msg;

    cout << "请输入文件名：" << endl;
    string filename = "";
    cin >> filename;
    // 发送文件名
    sendto(ClientSocket,(char*)(filename.c_str()), filename.length(),0,
(SOCKADDR*)&ServerAddr,l);

    //读文件
    sendbuf = new char* [80000];
    for (int i = 0; i < 80000; i++)
        sendbuf[i] = new char[MAXSIZE];

    ifstream filein;
    filein.open(filename, ifstream::binary);
    cout << "文件是否打开：" << filein.is_open() << endl;
    while (!filein.is_open())
    {
        cout << "找不到该文件，请重新输入，或确认文件是否存在" << endl;
        cin >> filename;
        filein.open(filename, ifstream::binary);
        if (filein.is_open())
            cout << "文件已打开" << endl;
    }

    filein.seekg(0, filein.end);  //将文件流指针定位到流的末尾
    length = filein.tellg();
    cout << "文件长度：" << length << endl;

    filein.seekg(0, filein.beg);  //将文件流指针重新定位到流的开始


    int len = length;
    while (len > 0)
    {
        filein.read(sendbuf[pagenum], min(len, MAXSIZE));
        len -= MAXSIZE;
        pagenum++;
    }
    cout <<"pagenum: "<< pagenum << endl;

    filein.close();
    save_pagenum = pagenum;
    int sendnum = pagenum;
    last_length = length % MAXSIZE;
    cout <<"last_length"<< last_length << endl;

    clock_t start = clock();
    recver = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)Recvpac, NULL,
NULL, NULL);
    //开始循环发送数据
    while (tail< save_pagenum)
```

```
    {
        //cout << "here..." << endl;

        //超时发送，这里先等一下
        while (!istimeout&& tail < save_pagenum)

        {
            //cout << "tail " << tail << endl;
            if (tail - base < NUM_WINDOW&& tail < save_pagenum)
            {
                SetPriorityClass(GetCurrentThread(), HIGH_PRIORITY_CLASS);

                if (sendnum == 1)
                {
                    //msg.set_EOF();
                    cout << "最后一个包的长度" << length << endl;
                }


                msg.Seq = tail;
                msg.datasize = min(MAXSIZE, length);

                msg.clearcontent();
                msg.setcontent(sendbuf[tail], min(MAXSIZE, length));

                msg.Checksum = 0;
                msg.set_Checksum();
                //cout << msg.Checksum << endl;
                //setsockopt(ClientSocket, SOL_SOCKET, SO_SNDBUF, (const
char*)&sendbuf, sizeof(char) * MAXSIZE);
                sendto(ClientSocket, (char*)&msg, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));

                time_list[tail] = clock();
                cout << "**************SEND time_list[" << tail << "]=" <<
time_list[tail] << endl;
                //time_list.push_back(make_pair(clock(), msg.Seq));//开始计时，在末
尾存入一个新元素
                cout << "[SEND] 发送包的长度" << msg.datasize << " 发送包的缓存区数："
<< tail << " SEQ:" << int(msg.Seq) << " 剩余长度 ： " << length << endl;

                //cout << clock() << endl;
                tail++;
                sendnum--;
                length -= MAXSIZE;

                //cout << "[Send]  " << msg.datasize << " bytes!" << " SEQ:" <<
int(msg.Seq) << " SUM:" << int(msg.Checksum) << endl;
                cout << "当前窗口：  " << base << " to " << tail << endl;
                SetPriorityClass(GetCurrentThread(), NORMAL_PRIORITY_CLASS);
            }
        }
```

```
    }


    while (!isend)
    {

    }
    msg.set_EOF();
    sendto(ClientSocket, (char*)&msg, sizeof(HEADER), 0, (SOCKADDR*)&ServerAddr,
sizeof(SOCKADDR));
    cout << "发送最后一个数据包" << endl;
    clock_t end = clock();

    stime = (end - start) / CLOCKS_PER_SEC;
    cout << "文件发送完毕" << endl;
    cout << "共计用时" << stime << "s" << endl;
    double throupt_rate = save_pagenum * sizeof(HEADER) / 1024 / stime;
    cout << "吞吐率为" << throupt_rate << "Mb/s" << endl;
    cout << "传输完毕" << endl;
    for (int i = 0; i < 80000; i++)
        delete[]sendbuf[i];
    delete[]sendbuf;
}
```

发送端接收数据包线程

```
void Recvpac() {
    clocker = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)TimeCheck, NULL,
NULL, NULL);

    while (true) {
        HEADER result;

        int res = recvfrom(ClientSocket, (char*)&result, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, &l);
        if (res > 0)
            cout << "收到了[RECV]  SEQ: " << result.Seq - 1 << endl;

        //cout << "here1 time_list:"<< time_list.size() << endl;

            //接收到的包没错，且是想要的
        if (result.get_ACK() && !result.get_NAK() && result.Seq == base + 1)
        {

            time_list[base] = MAXTIME;
            //cout << "改变time_list["<<base<<"]为 **************" <<
time_list[base] << endl;
            pagenum--;

            base++;
            if (base == save_pagenum)
            {
                isend = true;
                //time_list.clear();
                cout << "here" << endl;
```

```
                return;
            }

            if (NUM_WINDOW <= sshstrech)
                NUM_WINDOW *= 2;
            else
                NUM_WINDOW += 1;
            cout << "[RECV]  SEQ: " << result.Seq - 1 << "   FLAG: ACK WANT:" <<
base - 1 << "当前窗口" << base << " to " << tail << "窗口大小" << NUM_WINDOW <<
endl;
        }
        //丢包路由器中会截服务器端传回的包
        else if (result.Seq > base + 1) {
            base = result.Seq - 1;
            time_list[base] = MAXTIME;
        }

        else {
            cout << "[RECV]  不对的包 SEQ: " << result.Seq - 1 << "   FLAG: ACK
WANT:" << base << endl;

        }

        if (lastack != result.Seq) {
            repulicateack = 0;
            lastack = result.Seq;
            isrepluciteack = false;

        }
        else {
            repulicateack++;
            if(repulicateack>2)
                isrepluciteack = true;
        }



    }

}
```

发送端超时重传线程

```
void TimeCheck() {
    while (true)
    {
        if (base == save_pagenum) {
            return;
        }
        //cout << "time_list[base]*****************" << time_list[base] << endl;
        clock_t nowclock = clock();
        if (time_list[base]!=0&&nowclock-time_list[base]>MAX_TIME)
        {
            //cout << "now clock" << nowclock << " time_list["<<base<<"] " <<
time_list[base] << "  时间差   " << int(nowclock - time_list[base]) << endl;
```

```
                //重复ack
                istimeout = true;
                if (isrepluciteack) {
                    sshstrech /= 2;
                    NUM_WINDOW = sshstrech + 3;
                }
                else {
                    sshstrech /= 2;
                    NUM_WINDOW = 1;

                }

                SetPriorityClass(clocker, HIGH_PRIORITY_CLASS);
                //istimeout = true;

                cout << "超时重传 Seq= from " << base << " to " << tail << endl;
                cout << "-------------------------------------------------------
--" << endl;
                for (int i = base; i < tail; i++)   //即发出：SendBase ~ Seq-1 的报
文
                {
                    HEADER msg;
                    msg.Seq = i;


                    msg.clearcontent();
                    if (i != save_pagenum - 1)
                    {
                        msg.setcontent(sendbuf[i], MAXSIZE);
                        msg.datasize = MAXSIZE;
                    }
                    else
                    {
                        //return;
                        //msg.set_EOF();
                        msg.setcontent(sendbuf[i], last_length);
                        //cout << "last_length: " << last_length << endl;
                        msg.datasize = last_length;
                    }


                    msg.Checksum = 0;
                    msg.set_Checksum();
                    //cout << msg.Checksum << endl;
                    //setsockopt(ClientSocket, SOL_SOCKET, SO_SNDBUF, (const
char*)&sendbuf, sizeof(char) * MAXSIZE);
                    sendto(ClientSocket, (char*)&msg, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
                    cout << "重发包的大小" << msg.datasize << " 发送包的缓存区数：" <<
i << " 剩余长度 ： " << length << endl;

                    time_list[i] = clock();
                    //time_list[i-base]=make_pair(clock(), msg.Seq);
                    //time_list.pop();
                }
```

```
                // 恢复线程并行
                istimeout = false;
                SetPriorityClass(clocker, NORMAL_PRIORITY_CLASS);

        }
    }
}
```

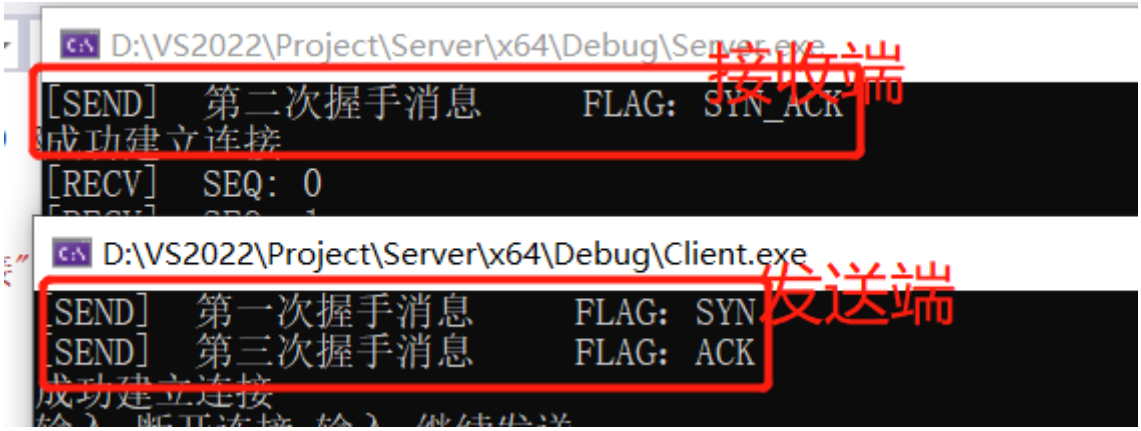
接收端收包

```
void RECV_FILE()
{
    //想要得到的包的序列号
    ack = 0;
    HEADER recv_msg, answer;
    //收文件名
    char* mes = new char[20];
    int length=recvfrom(ListenSocket, mes, 20, 0, (SOCKADDR*)&ServerAddr, &l);

    string filename;
    for (int i = 0; i < length; i++)
    {
        filename = filename + mes[i];
    }
    ofstream fileout;
    fileout.open(filename, ofstream::binary);

    //cvfrom(ListenSocket, mes, 4, 0, (SOCKADDR*)&ServerAddr, &l);
    //cout << mes << length << endl;
    int resize = recvfrom(ListenSocket, (char*)&recv_msg, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, &l);


    if (recv_msg.get_FIN() && recv_msg.get_ACK())
    {
        DisConnect();
        return;
    }
    if (recv_msg.check_Checksum())
    {
        fileout.write(recv_msg.content, recv_msg.datasize);
        ack = recv_msg.Seq + 1;
        cout << "[RECV]  SEQ: " << recv_msg.Seq << endl;

    }
    else
        answer.set_NAK();
    answer.set_ACK();
    answer.Seq = ack;

    sendto(ListenSocket, (char*)&answer, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
```

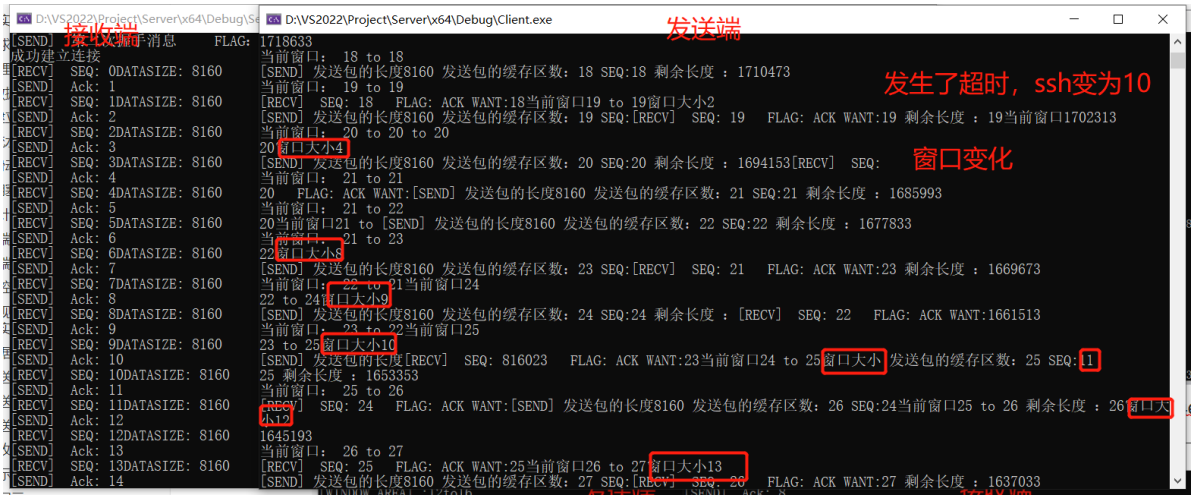
```
    while (!recv_msg.get_EOF())
    {

        recv_msg.clearcontent();

        int res = recvfrom(ListenSocket, (char*)&recv_msg, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, &l);
        while (res < 0)
        {
            sendto(ListenSocket, (char*)&answer, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
            res = recvfrom(ListenSocket, (char*)&recv_msg, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, &l);
        }
        if (recv_msg.Seq != ack)
        {
            sendto(ListenSocket, (char*)&answer, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
            cout << "[RECV] not want ,need to resend" << endl;
            continue;
        }

        if (recv_msg.get_FIN() && recv_msg.get_ACK())
        {
            DisConnect();
            return;
        }

        if (recv_msg.check_Checksum())
        {

            fileout.write(recv_msg.content, recv_msg.datasize);
            cout << "[RECV]  SEQ: " << recv_msg.Seq << endl;
            ack = recv_msg.Seq + 1;
        }
        else
        {
            answer.set_NAK();
            cout << "收到的包数据有误，WRONG CHECKNUM: " << recv_msg.Checksum <<
endl;
        }

        answer.set_ACK();
        answer.Seq = ack;
        sendto(ListenSocket, (char*)&answer, sizeof(HEADER), 0,
(SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
        cout << "[SEND]  Ack: " << answer.Seq << endl;
    }


    cout << "接收完毕" << endl;
    fileout.close();
}
```

结果展示

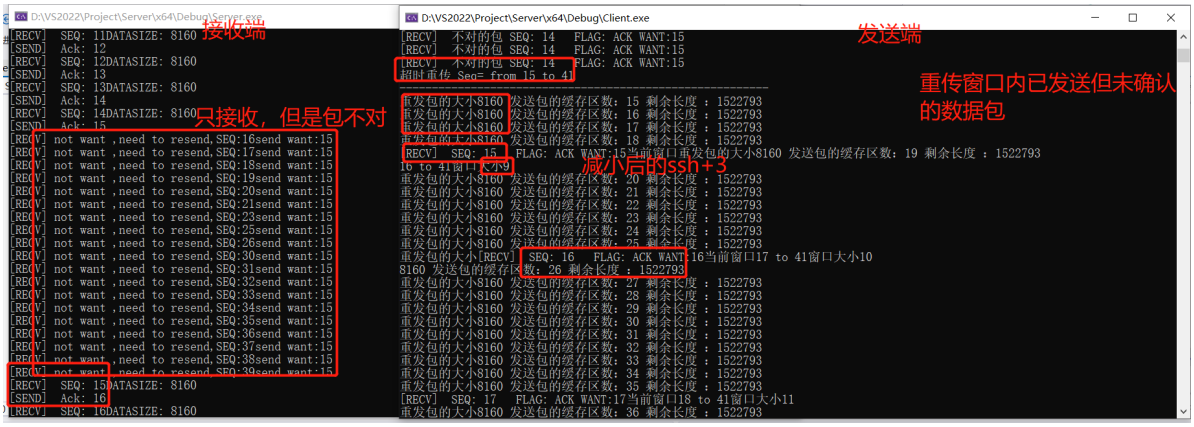
握手展示

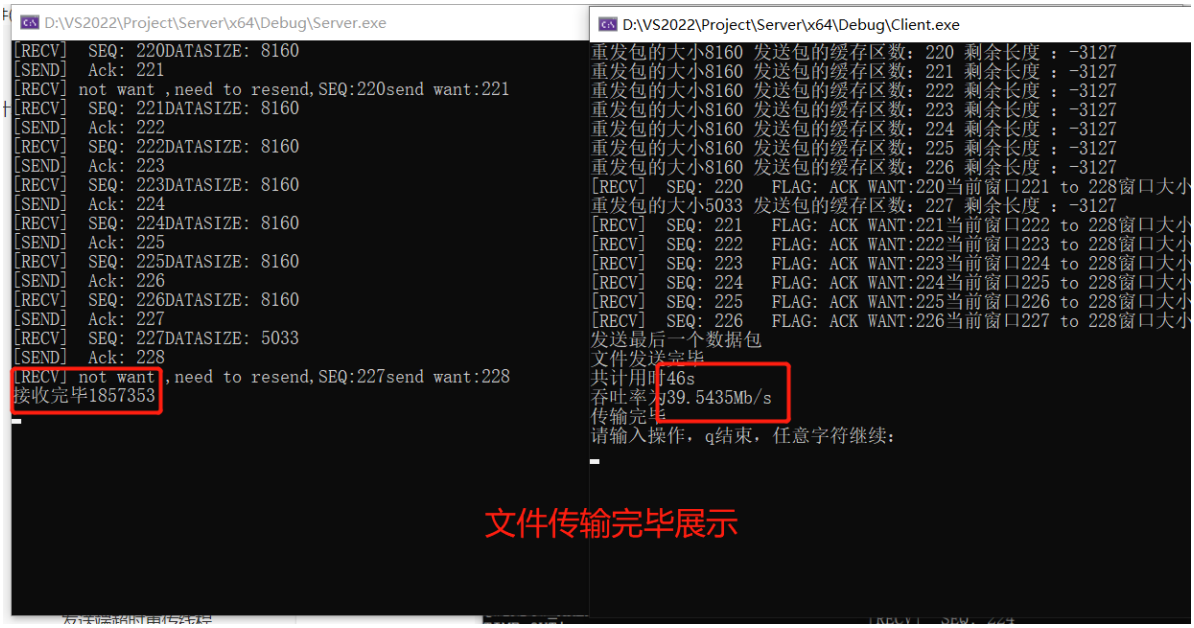


数据传输展示

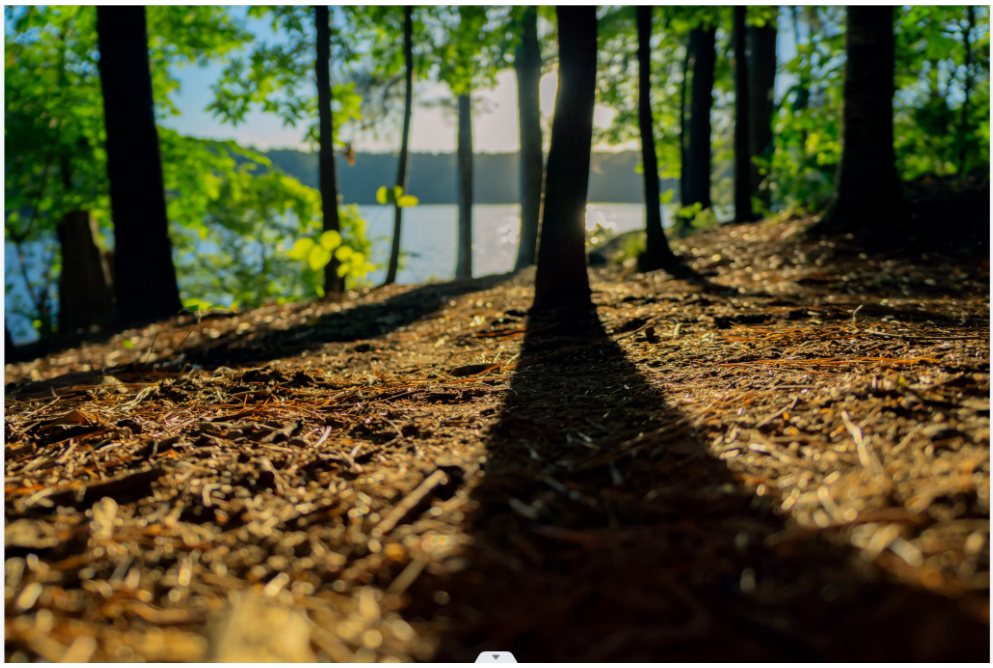


丢包展示



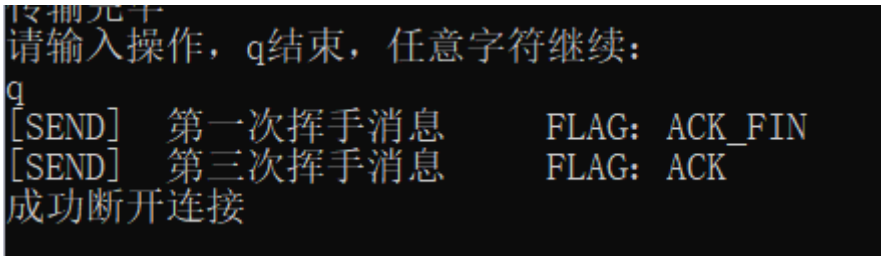


名称	修改日期	类型	大小
.vs	2022/11/16 21:44	文件夹	
x64	2022/11/18 0:20	文件夹	
1.jpg	2022/12/29 13:25	JPG 图片文件	1,814 KB
2.jpg	2022/12/25 23:34	JPG 图片文件	5,761 KB
3.jpg	2022/11/18 18:32	JPG 图片文件	11,689 KB
helloworld.txt	2022/12/25 22:40	文本文档	0 KB
server.cpp	2022/12/29 13:24	C++ source file	7 KB
Server.sln	2022/11/17 0:10	Microsoft Visual ...	3 KB
Server.vcxproj	2022/11/17 0:10	VC++ Project	7 KB
Server.vcxproj.filters	2022/11/17 0:10	VC++ Project Fil...	1 KB
Server.vcxproj.user	2022/11/16 21:44	USER 文件	1 KB



最终可以得到正常数据，文件大小和原来的都一样。

挥手展示



可以看到在GBN滑动窗口的传输的基础上，实现了Reno算法的拥塞控制。