

基于UDP服务设计可靠传输协议并编程实现3-3

学号：2013622

姓名：罗昕珂

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。

三、协议设计

1、传输协议特点

- 发送方和接收方有握手和挥手过程，确认连接断开连接；
- 数据可以切包传输，保存序列号；
- 数据报有差错检测功能，通过序列号和标志位检查，正确接收返回确认；
- 数据报有出错重传，解决出错、超时、丢失等情况；
- 数据报使用滑动窗口传输机制和序列号，不会出现失序情况；
- 发送方和接收方有挥手过程，断开连接。

2、数据报格式

握手挥手时的传输内容

校验和	标志位
-----	-----

普通数据包

校验和	标志位	序列号	数据
-----	-----	-----	----

传输段最后一个数据包，则增加一个字节来保存该数据段长度。

校验和	标志位	序列号	长度	数据
-----	-----	-----	----	----

1) 校验和：

计算校验和时，累加的长度可变

2) 标志位：

- ACK,用于握手确认连接和发包确认接收以及确认挥手断开连接;
- SHAKE_1,SHAKE_2,SHAKE_3,三次握手确认连接
- WAVE_1,WAVE_2,两次挥手断开连接
- last_len: 最后一个包的数据段长度
- 结尾数据包标志位: 对于非结尾数据包, 标志位的倒数第 4 位为 1; 对于结尾数据包, 标志位的倒数 4, 5 位为 1;

3) 序列号:


8位, 用于握手确认连接和发送切片时保存序列号;

4) 长度限制:

对于一个数据包的长度是有限制的, 由于序列号只有8位, 对于不是最后一个传输的数据包来说最多数据长度为 253 个字节, 即数据报文为 256 个字节。对于传输段最后的不完整的数据来说则按具体长度发送。

3、握手确认连接

- 单向传输, 三次握手确认连接
- 步骤一: 客户端发送SHAKE_1段握手信号
- 步骤二: 服务端接收SHAKE_1段, 计算校验和, 若收到 SHAKE_1 且校验和等于 0, 则回送SHAKE_2段.
- 步骤三: 客户端接收SHAKE_2段, 并计算出校验和正确, 开始回送SHAKE_3段.
- 步骤四: 服务器端接收到SHAKE_3第三次握手信号, 且计算出校验和正确, 那么握手成功, 建立连接。

 Router

路由器IP:

端口:

丢包率: %

服务器IP:

服务器端口:

延时: ms

确定

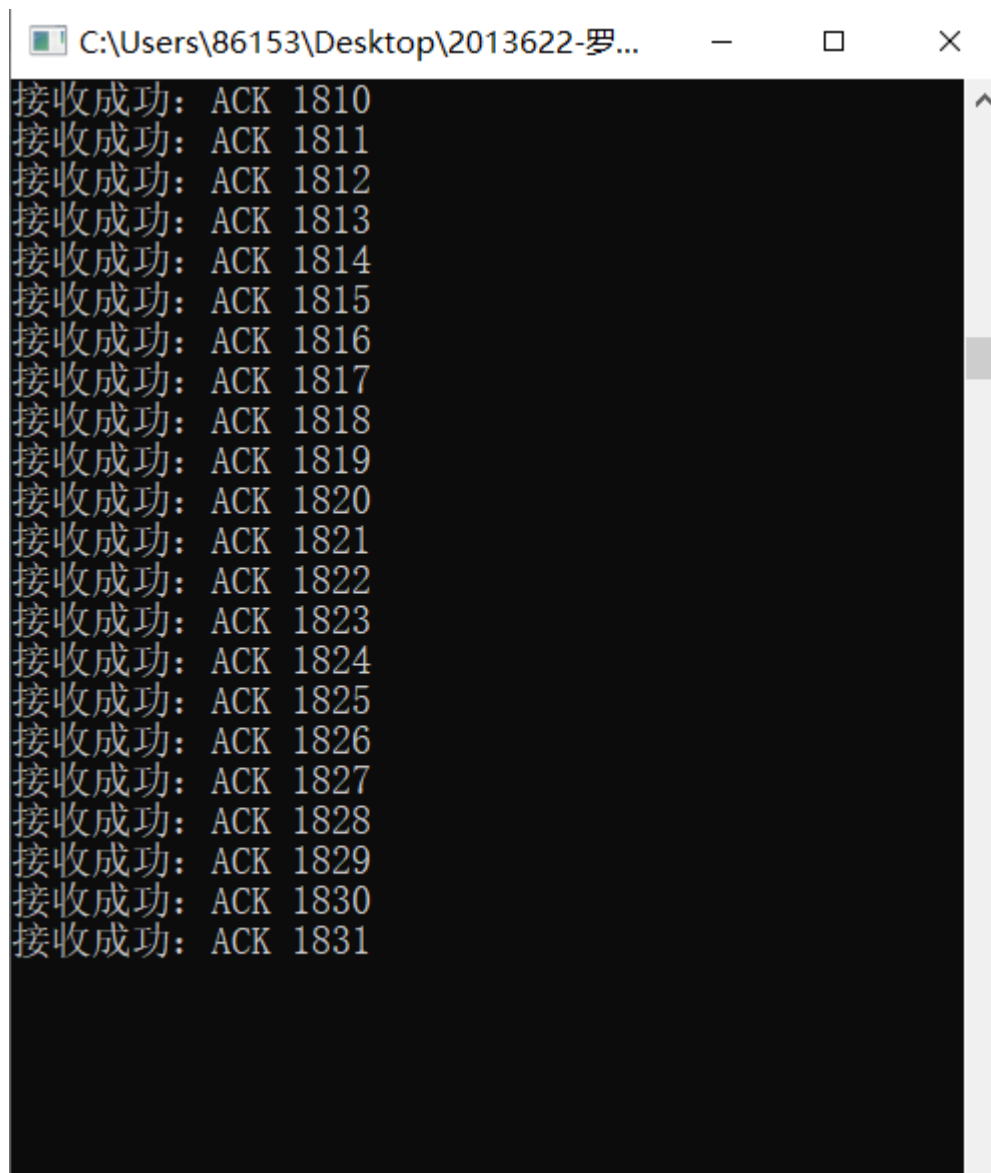
修改

日志

Router Ready!
Misscount :0 .
Delay :0 ms .

4、挥手断开连接

- 单向传输，二次握手断开连接
- 步骤一：客户端发送次挥手信号 wave_1。
- 步骤二：服务端接收后计算校验和，若收到 wave_1 且校验和等于 0，则开始 发送第二次挥手信号 wave_2。
- 步骤三：客户端收到服务端发来的 wave_2，并计算出校验和正确，则挥手成功，断开连接。



```
C:\Users\86153\Desktop\2013622-罗...
接收成功: ACK 1810
接收成功: ACK 1811
接收成功: ACK 1812
接收成功: ACK 1813
接收成功: ACK 1814
接收成功: ACK 1815
接收成功: ACK 1816
接收成功: ACK 1817
接收成功: ACK 1818
接收成功: ACK 1819
接收成功: ACK 1820
接收成功: ACK 1821
接收成功: ACK 1822
接收成功: ACK 1823
接收成功: ACK 1824
接收成功: ACK 1825
接收成功: ACK 1826
接收成功: ACK 1827
接收成功: ACK 1828
接收成功: ACK 1829
接收成功: ACK 1830
接收成功: ACK 1831
```

5、数据报的差错检测

在发送包（包括 ACK 和 NAK）还有握手挥手的时候都需要进行校验和的计算，来 确保发包是否出错

1) 发送端：

- 对要发送的UDP数据报，校验和域段清0
- 产生伪首部将数据报用0补齐为16位整数倍
- 将伪首部和数据报看成16位整数倍序列
- 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。

2) 接收端：

- 对接收到的UDP数据报，产生伪首部将数据报用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，没有检测到错误
- 否则，数据报存在差错。

6、数据报切包发送

单片长度过大时切包发送，单片长度不超过253

7、数据报确认重传：

1) 超时重传：

在发送一个数据包之后，就开启一个定时器，若是在这个时间内没有收到发送数据的 ACK 确认报文，则对该报文进行重传。并且将发送端的计数器减一，选择适当的序列号。

在文件传输过程中，在服务端 `recvfrom` 是进行阻塞接受的，使用库函数将其阻塞时间进行设置，在我们的 `TIMEOUT` 范围之内。

2) 出错重传：

- 接收端收到包之后，计算校验和，如果数据包检测无差错，将 ACK 标志置为 1，发送给发送端，确认收到；如果数据包校验和有差错，将 ACK 标志置为 0，发送给发送端，表示出现包错误，并丢弃错误的数据包；
- 发送端收到接收端发来的 ACK 之后，如果 ACK 标志位为 1,表示数据包发送成功，发送端继续发送下一个数据包；如果 ACK 标志位为 0，表示数据包有差错，发送端将重传数据包。

3) 挥手握手重传：

- 握手挥手过程中的丢包因为无法确认和检测校验和，只能进行重新进行握手挥手实现。
- 并且在挥手中进行超时次数的设置，超过次数认为对方完全断网，自己则进行资源回收后断网。

8、滑动窗口传输机制

该协议允许发送方在等待确认前可以连续发送多个分组。由于发送方不必每发一个分组就停下来等待确认，因此该协议可以加速数据的传输。但它也受限 于在流水线中未确认的分组不能超过最大窗口数。不同于 3-1 实现的停等协议的是，之前是一个一个连续地发送，相当于滑动窗口的窗口大小等于 1。

在发送端，窗口内的分组序号对应的数据分组是可以连续发送的。窗口内的数据 分组有：

- 已发送但尚未得到确认
- 未发送但可以连续发送
- 已发送且已得到确认，但窗口中本序号的前面还有未得到确认的数据分组

滑动窗口法要求各数据分组按顺序发送，但并不要求确认按序返回。一旦窗口前 面部分的数据分组得到确认，则窗口向前滑动相应的位置，落入窗口中的后续分 组又可以连续发送。

在本次程序中，对于发送方我们设定一个大小为 `WINDOW_SIZE` 的序号窗口，其 中的数据为已经发送确还未确认的。

- 如果窗口未满，则我们可以继续发送数据包；
- 如果窗口满了，则需要等待 ACK 确认后窗口的减小，之后才能发送数据包。

9、GBN 协议设计

采用 while 循环来回调换 recv 和 send 进行实现，避免了使用线程导致的繁琐。具体含义是对于每一个发送的数据包，记录了他发送出去的时间，存储在一个队列里 timer_list 中，之后根据当前 clock()-最初发送的时间来判断队首的包是否发送超时，若超时则需要进行回退重发，并且清空队列；如果未超时收到包，则需要弹出队列队首，即更新计时器。

10、累计确认

只有当接收端收到的序列号和自己累计确认的序列号相同，才会发送这个序号对应的 ACK 包，否则即使接受到了之后的包，也只会发送之前连续接受到的正确的包的 ACK。

3-3补充协议

11、拥塞控制reno算法

- Slow Start (慢启动)

当cwnd的值小于sssthresh时，则处于slow start阶段，每收到一个ACK，cwnd的值就会加1。

经过一个RTT的时间，cwnd的值就会变成原来的两倍，实为指数增长。

1. 如果期间有数据段包发生了超时重传

那么此时发送方应该进入，并且将下一轮的 sssthresh 设置为当前发送窗口的一半及 $cwnd/2$ ，之后将 cwnd 设置为 1，重新开始慢启动状态。

2. 如果当前发送窗口 cwnd 超过了 sssthresh

则进入下一个状态——拥塞避免状态。

3. 如果在当前出现了 3 个冗余 ACK 的情况

则计入另一个状态，快速回复状态。

- Congestion Avoidance (拥塞避免)

当cwnd的值超过sssthresh时，就会进入Congestion Avoidance阶段，在该阶段下，cwnd以线性方式增长，大约每经过一个RTT，cwnd的值就会加1

1. 出现某个包的超时事件

表明这个包已经丢失。并且后面的包也会丢失，因此进入慢启动阶段，将 sssthresh 设置为 $cwnd/2$ ，并且将 cwnd 设置为1，重新进入慢启动状态。

2. 如果出现了 3 次冗余 ACK

则证明只有一个包丢失，其余后面的包到达发送方，因此状态将走到下一个状态——快速恢复状态

- Fast Recovery (快速恢复)

按照拥塞避免算法中cwnd的增长趋势，迟早会造成拥塞（一般通过是否丢包来判断是否发生了拥塞）。

1. 当收到 3 个重复 ACK 时

把 ssthresh 设置为 cwnd 的一半，把 cwnd 设置为 ssthresh 的值加 3，然后重传丢失的报文段，加 3 的原因是因为收到 3 个重复的 ACK，表明有 3 个“老”的数据包离开了网络。

2. 若再收到重复的 ACK 时

拥塞窗口增加 1。

3. 当收到新的数据包的 ACK 时

把 cwnd 设置为第一步中的 ssthresh 的值。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。

四、3-3实验代码

定义数据包

```
struct MESSAGE {
    int server_port; // 端口号
    int seq_num; // 序号
    int length; // 报文段二进制长度
    int check_sum; // 校验和
    char* message; // 报文段
    MESSAGE() {}
    // 定义的数据包格式:
    MESSAGE(int server_port, int seq_num, int length, int check_sum, char*
message) :
        server_port(server_port),
        seq_num(seq_num),
        length(length),
        check_sum(check_sum),
        message(message) {}
    void print() {
        printf("端口:%d\n数据包序列号:%d\n数据段长度:%d\n校验和:%d\n数据:%s\n",
            server_port, seq_num, length, check_sum, message);
    }
}
```

整理收到的报文

```
// 真正的报文，二进制串，保存到msg中
void send_message(char* msg) {
    // 存的不是每一位，而是把8位作为一个字节，存到char的一个单位里
    msg[0] = High(server_port);
    msg[1] = Low(server_port); // 前两位存端口号
    msg[2] = High(seq_num);
    msg[3] = Low(seq_num); // 之后存序列号
    msg[4] = High(length);
    msg[5] = Low(length); // 之后是数据长度
    msg[6] = 0;
```

```

    msg[7] = 0; //校验和
    for (int i = 0; i < length; ++i) {
        msg[8 + i] = message[i]; //之后都存数据
    }

    int a = checksum(msg); //存入数据之后再计算校验和
    msg[6] = High(a);
    msg[7] = Low(a);
}

```

计算校验和

```

//计算校验和，每16位转为10进制，然后求和取反
int checksum(char* msg) {
    unsigned long sum = 0;
    for (int i = 0; i < 8 + length; i += 2) { //这里不能用strlen(msg)，因为如果中间有0，读到就不计算了
        sum += HighLow(msg[i], msg[i + 1]);
        sum = (sum >> 16) + (sum & 0xffff);
        //后16位加上自己的进位部分，相当于回卷
        //用二进制举例，因为和最大为111+111=1110=1111-1，所以加上进位后肯定不会再进位了
    }
    return (~sum) & 0xffff;
}
};

```

定义拥塞控制的一些变量

```

double cwnd = 1.0; //窗口大小
double ssthresh = 80.0; //阈值，一旦达到阈值，则指数->线性
int dup_ack_cnt = 0; //冗余ack计数器
int last_ack_seq = 0; //上一次的ack序号，用于更新ack_cnt

SOCKET localSocket;
struct sockaddr_in serverAddr, clientAddr; //接收端的ip和端口号信息

DWORD WINAPI handlerRequest(LPVOID lpParam); //负责接收的线程

bool begin_recv = false; //可以开始接收
bool waiting = false; //等待发送
string temp;

int nextseqnum = 0; //序号

```


接收和发送报文

```
void send_to(char* message) {
    MESSAGE u = MESSAGE(SERVER_PORT, nextseqnum, 1024 - 10, 0, message); //打包成
    udp
    //打包后的报文msg
    char msg[bufferSize];
    u.send_message(msg);
    //发送数据
    sendto(localSocket, msg, sizeof(msg), 0, (SOCKADDR*)&serverAddr,
    sizeof(SOCKADDR)); //大小很重要
}

//接收
void recv_from(char* message) {
    char msg[bufferSize];
    int size = sizeof(serverAddr);
    recvfrom(localSocket, msg, sizeof(msg), 0, (SOCKADDR*)&serverAddr, &size);
    handle(msg, message);
}
```

主函数

```
int main() {
    //初始化
    WSADATA wsaData;
    WORD wVersionRequested = MAKEWORD(2, 1);
    int error=WSAStartup(wVersionRequested, &wsaData);
    if (error) {
        printf("init error");
        return 0;
    }

    //去连接服务器的socket
    localSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    //服务器的ip和端口号
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(SERVER_PORT);
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    clientAddr.sin_family = AF_INET;
    clientAddr.sin_port = htons(CLIENT_PORT);
    clientAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    HANDLE hThread = CreateThread(NULL, 0, handlerRequest, LPVOID(), 0, NULL);

    //cout << "请输入目的端口号(路由器4001, 服务器6666): ";
```

```

//cin >> sendbuf;
string filename;
while (1) {
    printf("请输入要发送的文件名: ");
    cin >> filename;
    ifstream fin(filename.c_str(), ifstream::binary);
    if (!fin) {
        printf("文件未找到!\n");
        continue;
    }
    else {
        cout << "找到文件, 开始发送..." << endl;
        unsigned char t = fin.get();
        while (fin) {
            buffer[len++] = t;
            t = fin.get();
        }
        fin.close();
        send_file_2(filename);
    }
    break;
}

```

最开始的慢启动阶段

```

while (nextseqnum < n)
{
    int end = clock();
    dt = (double)(end - start);
    //如果超时, 未确认的全部重传
    if (dt >= TIME_OUT) {
        cout << "超时, 超时时间为: " << dt << endl;
        ssthresh = cwnd / 2;
        cwnd = 1;
        dup_ack_cnt = 0;
        nextseqnum = base; //重发, 回退到base
        cout << "超时, 窗口变化为: " << cwnd << endl;
        cout << "超时, 阈值变化为: " << ssthresh << endl;
    }
    //当现在有空间, 且不在停等状态时, 允许发送报文
    if (nextseqnum < base + cwnd)
    {
        message = v[nextseqnum];
        send_to(message);
        start = clock();
        begin_recv = true; //***可以开始接收服务器端消息啦***
        //cout<<nextseqnum<<' '<<strlen(message)<<endl;
        nextseqnum++; //新的报文, 序号变化
    }
}

```

```
}
```

用多线程去接收返回的ACK从而判断要变化的状态

- 1) 慢启动阶段: cwnd呈现指数增长趋势
- 2) 拥塞避免阶段: $cwnd > ssthresh$ 呈现线性增长趋势
- 3) 快速恢复阶段: 发送方只要一连接收到三个重复确认就应该立即重传对方尚未的报文段, 而不必等到重传计时器超时后发送。由3个重复应答判断有包丢失, 重新发送丢包的信息。

```
if (handle(msg, message)) {
    int new_ack_seq = HighLow(msg[2], msg[3]);
    cout << "ACK" << new_ack_seq << endl;
    base = new_ack_seq + 1;

    if (new_ack_seq != last_ack_seq) { //ack序号不冗余
        last_ack_seq = new_ack_seq;
        dup_ack_cnt = 0;
        //拥塞避免状态
        if (cwnd >= ssthresh) {
            cwnd += 1 / cwnd;
            cout << "发生拥塞避免, 窗口变化为: " << cwnd << endl;
            cout << "发生拥塞避免, 阈值为: " << ssthresh << endl;
        }
        //慢启动状态, 每收到一个ack, 窗口+1, 一轮过后就是*2
        else {
            cwnd++;
            cout << "慢启动状态, 窗口为: " << cwnd << endl;
            cout << "慢启动状态, 阈值为: " << ssthresh << endl;
        }
    }

    else { //冗余
        dup_ack_cnt++;
        if (dup_ack_cnt == 3) { //3个冗余ack, 则阈值=cwnd/2, cwnd=阈值+3
            ssthresh = cwnd / 2;
            cwnd = ssthresh + 3;
            nextseqnum = base; //重传, 回退到base
            cout << "3个重复ack, 窗口变化为: " << cwnd << endl;
            cout << "3个重复ack, 阈值为: " << ssthresh << endl;
        }
    }
}
```

接收端要判断是否是想要的序列号

```
//如果校验和正确, 并且收到的序号是想要的
if (check == true && expected_seqnum == recv_seqnum) {
```

```
    cout << "接收成功: ACK " << expected_seqnum << endl;
    fwrite(message, 1, HighLow(msg[4], msg[5]), fout);
    send_to(ss, expected_seqnum);
    expected_seqnum++; //更新想要的序号
}
else {
    //否则发送最近正确接收的序号
    cout << "接收失败: ACK " << expected_seqnum - 1 << endl;
    send_to(ss, expected_seqnum - 1);
}
memset(message, 0, sizeof(message));
```

打印吞吐率

```
Sleep(10000); //一定要等一会, 否则线程立即关掉, 没法打印结果
CloseHandle(hThread);
closesocket(localSocket);
WSACleanup();
cout << "传输时间为: " << all_pass << "s" << endl;
cout << "平均吞吐率为: " << len*8 / 1000 / all_pass << "kbps" << endl;
cout << "退出请输入0" << endl;
```

五、实验结果

为方便测试, 接收端的端口固定为7777, 路由器的端口为6666。

 Router ×

路由器IP:	<input type="text" value="127 . 0 . 0 . 1"/>	服务器IP:	<input type="text" value="127 . 0 . 0 . 1"/>
端口:	<input type="text" value="6666"/>	服务器端口:	<input type="text" value="7777"/>
丢包率:	<input type="text" value="0"/> %	延时:	<input type="text" value="0"/> ms
<input type="button" value="确定"/>		<input type="button" value="修改"/>	

日志

Router Ready!
Misscount :0 .
Delay :0 ms .

<

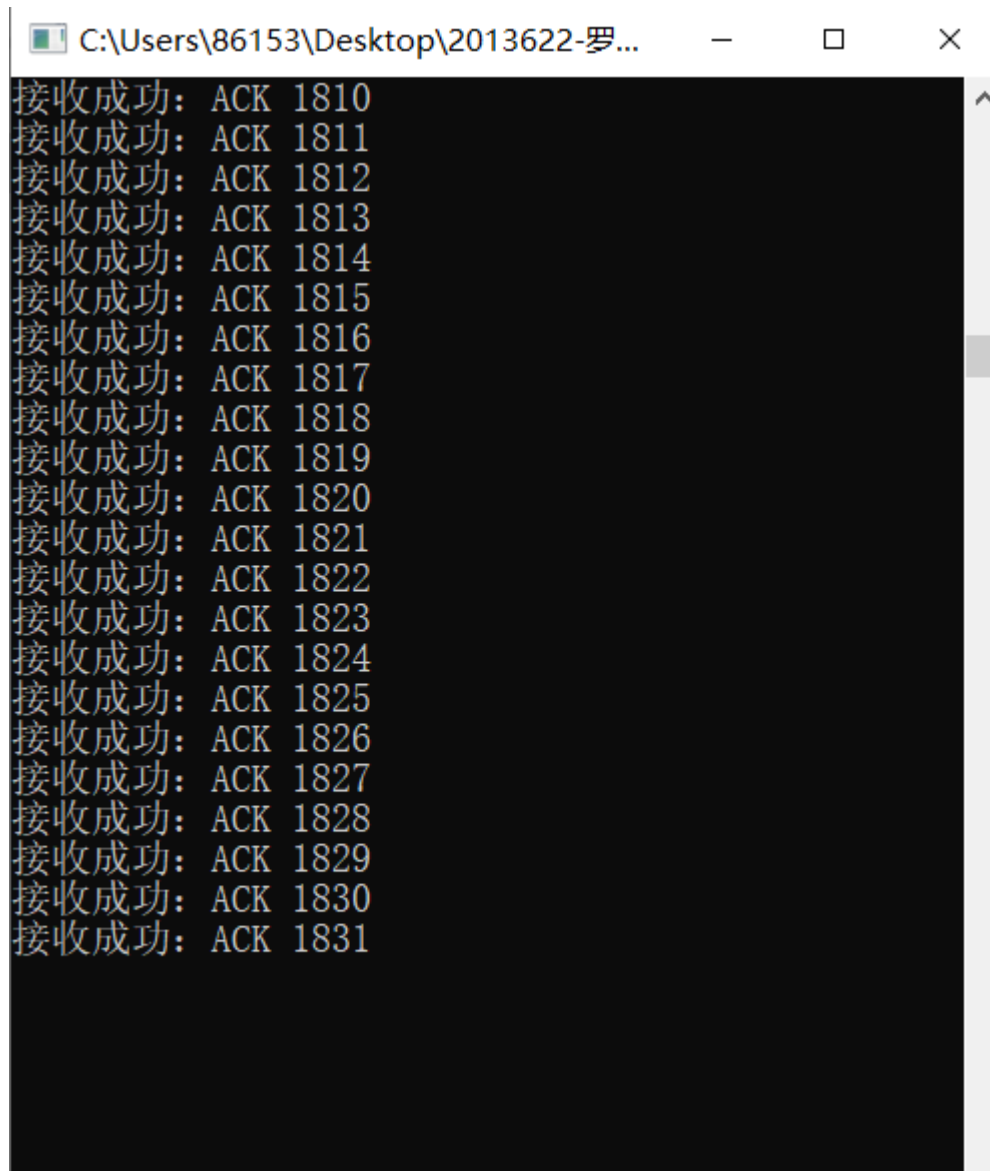
>

^

v

选择发送的文件名:1.jpg

由发送端发送给路由器，路由器转发到接收端，接收结果如下：



```
C:\Users\86153\Desktop\2013622-罗...  
接收成功: ACK 1810  
接收成功: ACK 1811  
接收成功: ACK 1812  
接收成功: ACK 1813  
接收成功: ACK 1814  
接收成功: ACK 1815  
接收成功: ACK 1816  
接收成功: ACK 1817  
接收成功: ACK 1818  
接收成功: ACK 1819  
接收成功: ACK 1820  
接收成功: ACK 1821  
接收成功: ACK 1822  
接收成功: ACK 1823  
接收成功: ACK 1824  
接收成功: ACK 1825  
接收成功: ACK 1826  
接收成功: ACK 1827  
接收成功: ACK 1828  
接收成功: ACK 1829  
接收成功: ACK 1830  
接收成功: ACK 1831
```

由发送端记录状态日志:

```
C:\Users\86153\Desktop\2013622-罗昕珂-lab3-3...  -  □  ×
发生拥塞避免, 窗口变化为: 83.7857
发生拥塞避免, 阈值为: 80
ACK1690
发生拥塞避免, 窗口变化为: 83.7977
发生拥塞避免, 阈值为: 80
ACK1695
发生拥塞避免, 窗口变化为: 83.8096
发生拥塞避免, 阈值为: 80
ACK1701
发生拥塞避免, 窗口变化为: 83.8215
发生拥塞避免, 阈值为: 80
ACK1706
发生拥塞避免, 窗口变化为: 83.8334
发生拥塞避免, 阈值为: 80
ACK1712
发生拥塞避免, 窗口变化为: 83.8454
发生拥塞避免, 阈值为: 80
ACK1717
发生拥塞避免, 窗口变化为: 83.8573
发生拥塞避免, 阈值为: 80
ACK1722
发生拥塞避免, 窗口变化为: 83.8692
发生拥塞避免, 阈值为: 80
ACK1727
发生拥塞避免, 窗口变化为: 83.8812
发生拥塞避免, 阈值为: 80
ACK1733
发生拥塞避免, 窗口变化为: 83.8931
发生拥塞避免, 阈值为: 80
ACK1738
发生拥塞避免, 窗口变化为: 83.905
发生拥塞避免, 阈值为: 80
ACK1743
发生拥塞避免, 窗口变化为: 83.9169
发生拥塞避免, 阈值为: 80
ACK1748
发生拥塞避免, 窗口变化为: 83.9288
发生拥塞避免, 阈值为: 80
传输时间为: 7.156s
平均吞吐率为: 2076.3kbps
退出请输入0
```

传输时间为: 7.156s

平均吞吐率为: 2076.3kbps

五、实验中遇到的问题

1. 得到报文后16位转8位取值

```
#define High(number) ((int)number&0xFF00)>>8//得到高八位
#define Low(number) ((int)number&0x00FF)//得到低八位
#define HighLow(h,l) (((int)h<<8)&0xff00)|((int)l&0xff)//得到全部
```

2. 设计标记，关闭线程，否则一直输出最后的ACK

```
DWORD WINAPI handlerRequest(LPVOID lpParam) {
    while (1) {
        Sleep(10); //改变begin_recv后，需要等一下再接收，不然会丢掉第一个ACK
        char msg[bufferSize], message[bufferSize];
        int size = sizeof(serverAddr);
        if (begin_recv == false) {
            continue;
        }
    }
}
```