

# 计算机网络实验3-2实验报告-可靠数据传输 (滑动窗口)

学号：2012039 姓名：冯朝芃

## 实验概要

本实验利用数据报套接字在用户空间实现面向连接的可靠数据传输，实现了类似TCP的**三次握手、四次挥手、校验和**的差错检测、实现了基于**GBN的滑动窗口**发送模式，发送速度较上个实验大幅提升。

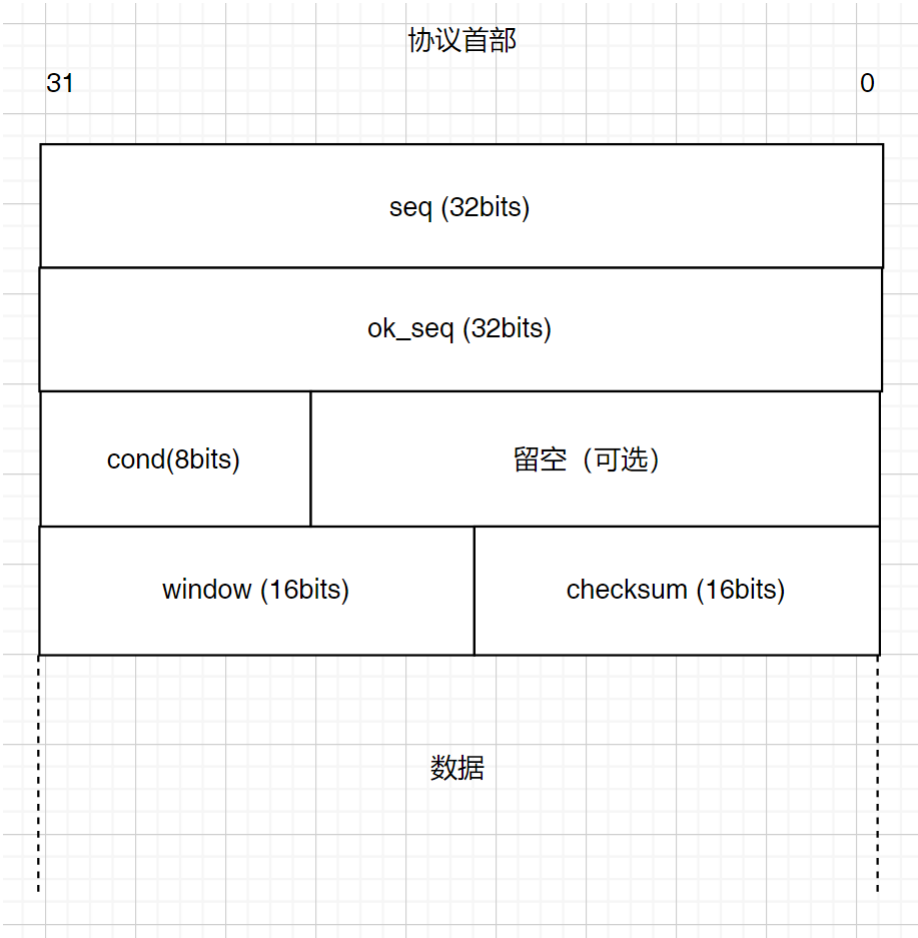
本实验程序能够完成**全部给定测试文件**的传输。

## 协议设计

协议设计部分与上一个实验保持一致。实验设计分为网络层和文件处理层，分层能够使我们的设计清晰简单。

### 网络层首部：

网络层首部模仿TCP首部设计，其中序列号长度为32bit，表示字节数位置。具体设计见下图。



seq和ok\_seq：序列号和确认号，支持32bit长度，在接收和发送时根据发送和确认的字节位置进行填写。

cond：状态值，目前实现有ack、syn、fin，分别使用此八位的第0、1、2位是否设置来表示。

window：发送窗口通告，这里用不上。

checksum：校验和

这个数据结构在代码中称transmission。

### 文件处理层首部：

文件处理层同样具有一个协议首部。在传输时，在数据最开始包含有文件处理层首部的数据包将率先被接收方收到，接收方解析其中的内容，将**知晓需要接收多少个数据包**。

设计如下：

```
struct fileInfo{
    char name[256]; //文件名
    char type[8];   //文件类型
    unsigned int size; //文件大小
};
```

## 协议时序

### 网络层：

与TCP协议不同，本实验seq将带有本次发送后，发送序列的结束字节位置，这样接收端就可以据此和保存的上一次的seq做减法来计算接收到的字节数量。ok\_seq是对对方发送字节位置的确认，与TCP协议一致，是期望接收到的下一个字节位置，接收方可以将对方发来的seq增加1来得到ok\_seq。

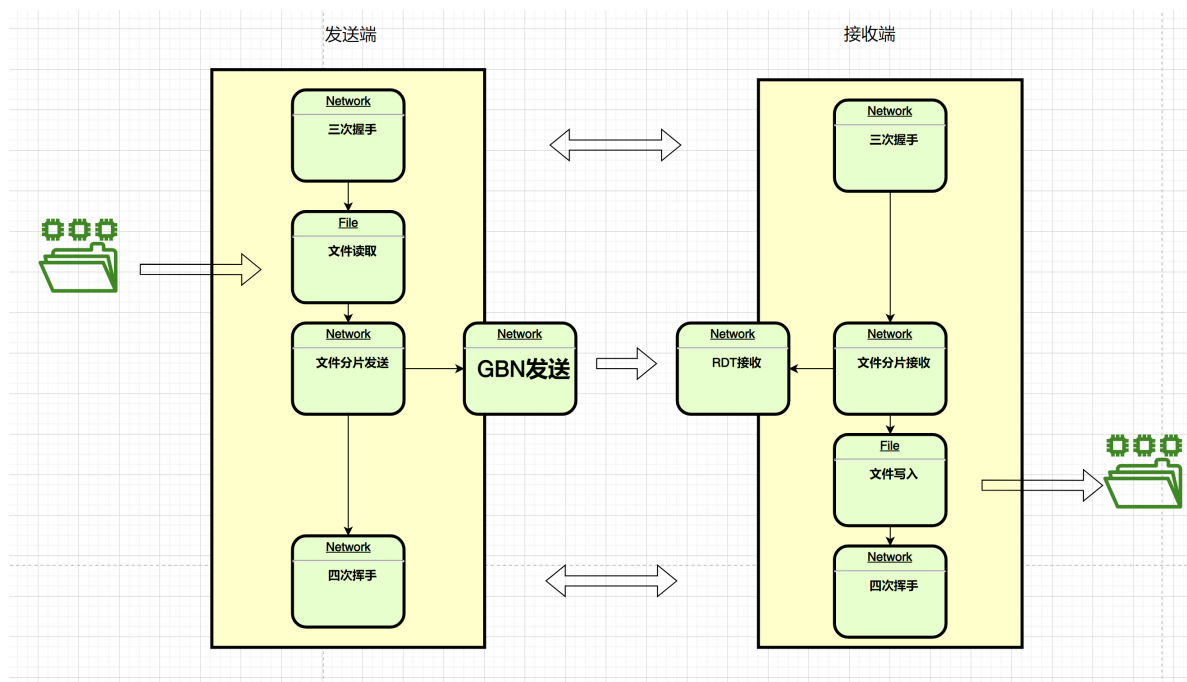
cond根据当前情况设置。checksum采用标准方法，发送端清零、计算、填写，接收端清零、计算、核对。

### 文件传输层：

fileInfo结构提在读取文件时被填写，接收端首先接收到该结构体，使用其中信息填写其他内部数据结构，并基于此决定接受次数和进行写盘。

## 功能模块划分

功能模块划分和工作基本流程见下图

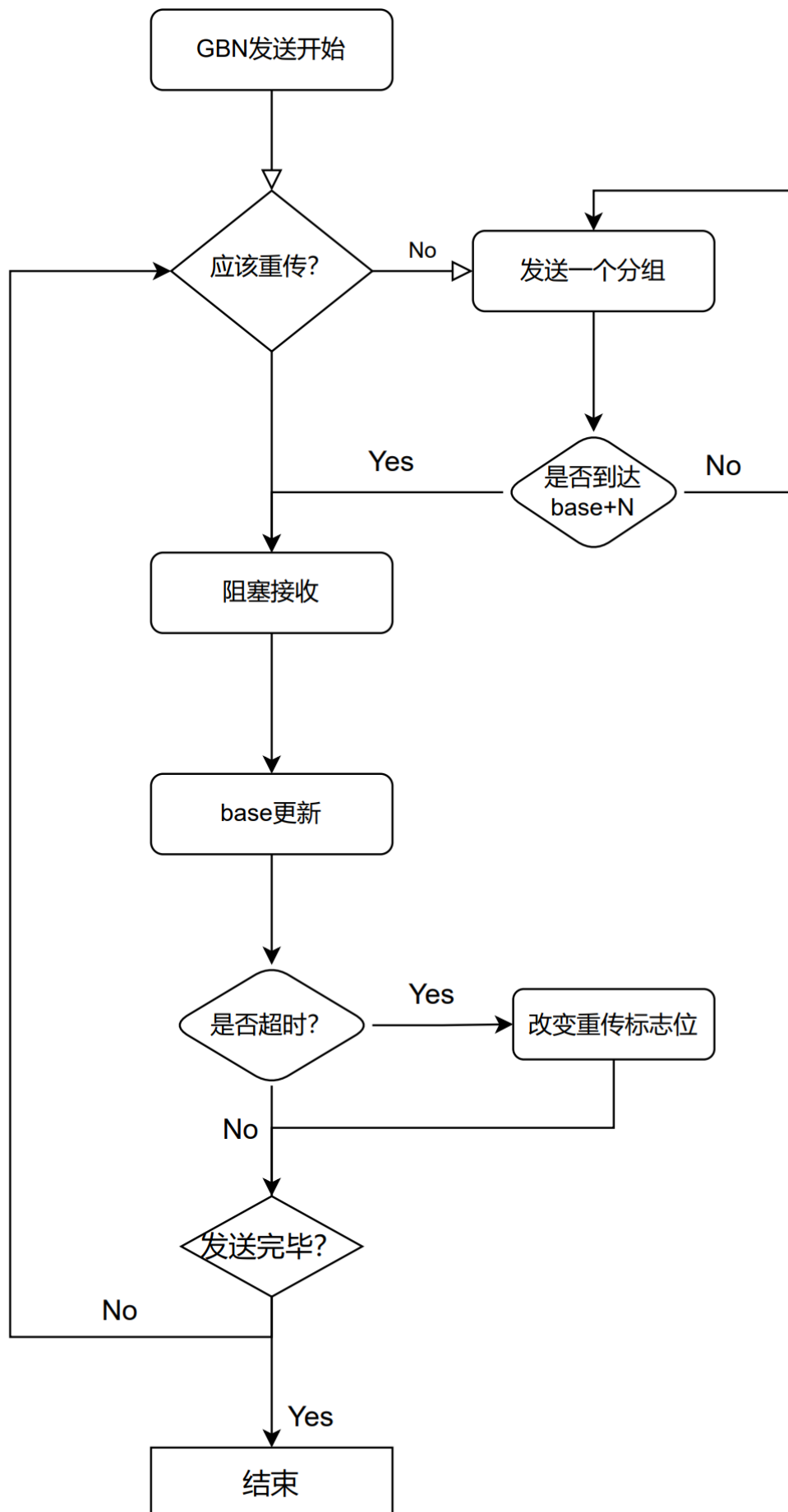


## 网络层模块：

GBN发送：本实验实现了**两种GBN发送方式**，分别是传统的单计时器模式和计时器序列模式。传统的单计时模式是在发送窗口头部更新时进行计时器重置，计时的精度相对低，注重的是窗口滑动的速度。计时器序列模式对每个不同的数据包各自设定计时器，在窗口滑动时进行计时器队列的更新和超时判断，计时更精确，注重的是每个数据包发送的时间，这种方法和TCP协议实现更加接近。

GBN发送基于累计确认和按字节数的序列号，以便于以后更灵活的调整窗口大小。

绘制GBN算法基本流程图如下：



三次握手：三次握手由接收端率先发起，流程和TCP协议一致。

四次挥手：服务端用户可通过输入命令来停止文件传送，这时将触发挥手操作，流程和TCP协议一致。

RDT接收：本实验接收和RDT3.0、RDT2.2的接收状态机一致，当收到非期望分组时将重发ACK。

文件分片发送：考虑UDP数据段所能携带的最大大小1475byte，大于我们的文件分片，在此函数中我们首先将分片拆成大小为NET\_BLOCK\_LIMIT（1024byte）的分片，仍然为链表结构。之后将每一个分片的二进制内容附在相应的transmission之后，调用RDT发送函数进行发送。

文件分片接收：在接收到分片后，首先根据第一个分片包含的fileInfo结构体初始化文件数据结构内容。之后接收分片形成链表。最后将其分片合并为大小不超过FILE\_BLOCK\_LIMIT大小的分片的链表。

### 文件层模块：

文件读取：为了能够读取较大的文件、更好的适应未来的流水线机制以及可能的对文件读取和发送并行化的实现，我们设计了一个文件分片链表的数据结构（blockHead）。链表上每个分片（unifiedBlock）将具有一个大小有上限的buffer并保存其大小。

文件读取模块将获取文件的基本信息：名称、类型和大小，使用其来填写fileInfo结构体。之后将这个结构体以及紧接的文件二进制内容存储进blockHead链表，这个过程中分片动态产生，每个分片大小不超过FILE\_BLOCK\_LIMIT（4096字节）。

文件写入：当网络层将第一个文件分片交给文件写入函数时，文件写入函数将首先解析fileInfo以获得文件信息。之后将遍历整个链表将文件写入。

## 关键代码展示

GBN发送1：定时器序列实现

```
void server::send_gbn(const unifiedBlock *c){
    sockaddr_in &client = clientVec.back();
    int dwSendSize = sizeof(client);
    //是否需要重发标志位
    bool needResend = false;
    //窗口：缓存buffer队列
    queue<char*> windowQueue;
    //计时器队列（记录一系列的开始发送时间）
    queue<long long> timeQueue;
    //窗口中的数据块
    queue<const unifiedBlock*> wait4ack;
    // 窗口大小、base和next都使用字节数，这是比较好适应网络发送需求也体现了解耦
    size_t N = 32768;
    //窗口base和nexttosend的初始值
    size_t base = curr_seq, next = curr_seq;
    char buffertemp[4096];
    //超时：3s
    long long timeout = 3000;

    while(1){
        //需要重传
        if(needResend){
            //清空计时器队列
            queue<long long>().swap(timeQueue);
            size_t queueLength = windowQueue.size();
            assert (windowQueue.size() == wait4ack.size());
            cout<<"进行超时重传"<<endl;
            //一次重传窗口内数据包
```

```

        for (int i = 0; i < queueLength; i++) {
            sendto(sListen, windowQueue.front(), sizeof(transmission) +
wait4ack.front()->getSize(), 0, (SOCKADDR *) &client, sizeof(SOCKADDR));
            //重新记录发送时间
            timeQueue.push(timeGetTime());
            windowQueue.push(windowQueue.front());
            windowQueue.pop();
            wait4ack.push(wait4ack.front());
            wait4ack.pop();
        }
        needResend = false;
    }else if(c){
        struct transmission tran = transmission_message::pure_ack;
        //更新curr_seq
        curr_seq += c->getSize();
        tran.seq = curr_seq;
        tran.ok_seq = client_seq++;
        auto s = new char[sizeof(transmission) + c->getSize()];
        memcpy(s, &tran, sizeof(transmission));
        memcpy(s + sizeof(transmission), c->getBuffer(), c->getSize());
        ((transmission *) s)->checksum = checksum(s, sizeof(transmission) +
c->getSize());
        //发送窗口新增1
        windowQueue.push(s);
        sendto(sListen, s, sizeof(transmission) + c->getSize(), 0, (SOCKADDR
*) &client, sizeof(SOCKADDR));
        //记录发送时间
        timeQueue.push(timeGetTime());
        wait4ack.push(c);
        //更新窗口大小, +发送数据包字节数
        next += c->getSize();
        c = c->getNext();
        //窗口是否还放得下
        if(c && next < base + N)
            continue;
    }

    size_t t, diff;
    transmission *p;
    //要等到窗口向前推进 (收到比窗口左端更靠后的数据包再退出)
    while(1) {
        recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *) &client,
&dwSendSize);
        p = (transmission*)buffertemp;
        t = p->ok_seq;
        diff = t - base;
        if (p->cond == ack && (diff > 0 || c == nullptr) &&
checksumchecker(buffertemp, 0)){
            break;
        }
    }
    base = t;
    //      cout<<"更新base diff: "<<diff<<endl;
    pretty->outputBaseNext(static_cast<condition>(p->cond), timeGetTime(),
p->seq, checksum(buffertemp, sizeof(transmission)), base, next);

```

```

    size_t acked = 0;
    if(diff > 0){
        //更新时间、窗口等队列
        //使用wait4ack判断应该pop多少
        while(acked < diff){
            acked += wait4ack.front()->getSize();
            wait4ack.pop();
            timeQueue.pop();
            assert(windowQueue.front() != nullptr);
            delete[] windowQueue.front();
            windowQueue.pop();
        }
    }
    //超时判断
    if(timeGetTime() - timeQueue.front() > timeout){
        needResend = true;
        cout<<"发现超时"<<endl;
    }
    //判断是否发完
    if (c == nullptr && wait4ack.empty()){
        assert(windowQueue.empty());
        assert(timeQueue.empty());
        break;
    }
}

cout<<"发送结束"<<endl;
}

```

GBN发送2：传统单计时器实现

```

//全局变量：是否需要重传
bool needRe;
//回调函数：设置重传控制变量
void CALLBACK TimerProc(HWND hwnd, UINT uMsg, UINT idEvent, DWORD dwTime)
{
    needRe = true;
}

void server::send_gbn_timer(const unifiedBlock *c){
    sockaddr_in &client = clientVec.back();
    int dwSendSize = sizeof(client);

    //重传控制变量清零
    needRe = false;
    //表示是否处于超时重传阶段
    bool isTimeout = false;
    queue<char*> windowQueue;
    queue<const unifiedBlock*> wait4ack;
    size_t N = 32768;
    size_t base = curr_seq, next = curr_seq;
    char buffertemp[4096];
    //超时: 3s
    long long timeout = 3000;

    while(1){

```

```

//需要重传
if(needRe){
    size_t queueLength = windowQueue.size();
    assert (windowQueue.size() == wait4ack.size());
    cout<<"进行超时重传"<<endl;
    //启动定时器
    SetTimer(NULL, 1, timeout, (TIMERPROC)TimerProc);
    //重传窗口内分组
    for (int i = 0; i < queueLength; i++) {
        sendto(sListen, windowQueue.front(), sizeof(transmission) +
wait4ack.front()->getSize(), 0, (SOCKADDR *) &client, sizeof(SOCKADDR));
        windowQueue.push(windowQueue.front());
        windowQueue.pop();
        wait4ack.push(wait4ack.front());
        wait4ack.pop();
    }
    needRe = false;
    //现在处于重传阶段
    isTimeout = true;
}else if(c && !isTimeout){
    //正常发送
    struct transmission tran = transmission_message::pure_ack;
    curr_seq += c->getSize();
    tran.seq = curr_seq;
    tran.ok_seq = client_seq++;

    auto s = new char[sizeof(transmission) + c->getSize()];
    memcpy(s, &tran, sizeof(transmission));
    memcpy(s + sizeof(transmission), c->getBuffer(), c->getSize());
    ((transmission *) s)->checksum = checksum(s, sizeof(transmission) +
c->getSize());
    windowQueue.push(s);
    sendto(sListen, s, sizeof(transmission) + c->getSize(), 0, (SOCKADDR
*) &client, sizeof(SOCKADDR));
    //启动或重传之后第一次正常: 启动定时器
    if(base == next){
        SetTimer(NULL, 1, timeout, (TIMERPROC)TimerProc);
    }
    wait4ack.push(c);
    next += c->getSize();
    c = c->getNext();
    if(c && next < base + N)
        continue;
}

size_t t, diff;
transmission *p;
while(1) {
    recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *) &client,
&dwsendSize);
    p = (transmission*)buffertemp;
    t = p->ok_seq;
    diff = t - base;
    if (p->cond == ack && (diff > 0 || c == nullptr) &&
checksumchecker(buffertemp, 0)){

```



```

        break;
    }
}
base = t;
pretty->outputBaseNext(static_cast<condition>(p->cond), timeGetTime(),
p->seq, checksum(buffertemp, sizeof(transmission)), base, next);
size_t acked = 0;
if(diff > 0){
    while(acked < diff){
        acked += wait4ack.front()->getSize();
        wait4ack.pop();
        assert(windowQueue.front() != nullptr);
        delete[] windowQueue.front();
        windowQueue.pop();
    }
}

if(base == next){
    //关闭定时器
    killTimer(NULL, 1);
    //重传完成
    isTimeout = false;
}else{
    //正常推动窗口
    //reset timer
    killTimer(NULL, 1);
    SetTimer(NULL, 1, timeout, (TIMERPROC)TimerProc);
}

if (c == nullptr && wait4ack.empty()){
    assert(windowQueue.empty());
    break;
}
}

cout<<"发送结束"<<endl;
}

```

文件读取fileLayer.cpp

```

fileBlockHead *fileLayer::loadFile(string filePath) {
    //用于程序内部的文件信息数据结构
    auto filehead = new fileBlockHead();
    //文件分片空间
    auto buff = new char[FILE_BLOCK_LIMIT];
    string fileName = filePath.substr(filePath.find_last_of('/') + 1, -1);
    //fileInfo数据结构
    struct fileInfo info;
    //获取文件大小
    HANDLE handle = CreateFile(filePath.c_str(), FILE_READ_EA,
                                FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
    if (handle != INVALID_HANDLE_VALUE) {
        info.size = GetFileSize(handle, NULL);
        CloseHandle(handle);
    }else{

```

```

        info.size = 0;
    }
    //完善fileInfo数据结构
    size_t pos = fileName.find(".");
    string name = fileName.substr(0, pos);
    string type = fileName.substr(pos + 1, -1);
    strcpy(info.name, name.c_str());
    strcpy(info.type, type.c_str());

    filehead->setFilename(name);
    filehead->setType(type);
    filehead->setSize(info.size);

    ifstream open_file(filePath, std::ios::binary);
    if (!open_file) {
        std::cout << "error" << std::endl;
        return 0;
    }
    //填充数据分片
    bool isFirst = true;
    memcpy(buff, &info, sizeof(fileInfo));
    while (!open_file.eof()) {
        if (isFirst) {
            //对第一个分片特殊处理, 加入fileInfo数据结构
            open_file.read(buff + sizeof(fileInfo), FILE_BLOCK_LIMIT -
sizeof(fileInfo));
            isFirst = false;
        } else {
            open_file.read(buff, FILE_BLOCK_LIMIT);
        }
        filehead->addBlock(buff, FILE_BLOCK_LIMIT);
        memset(buff, 0, FILE_BLOCK_LIMIT);
    }
    open_file.close();
    return filehead;
}

```

RDT发送server.cop

```

void server::sendrdt(char *b, unsigned int size) {
    int nRet = 0;
    sockaddr_in &client = clientVec.back();
    int dwSendSize = sizeof(client);
    auto m1 = (transmission *) b;
    //先发送一次, 然后进入循环计时等待
    nRet = sendto(sListen, b, size, 0, (SOCKADDR *) &client, sizeof(SOCKADDR));
    curr_seq++;
    if (nRet == SOCKET_ERROR) {
        cout << "sendto Error " << WSAGetLastError() << endl;
        return;
    }
    //设置接收超时
    DWORD timeout = 3000, t1, t2; //3s
    setsockopt(sListen, SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout,
sizeof(timeout));
}

```

```

//尝试接收数据
char buffertemp[4096];
auto p = (transmission *) buffertemp;
t1 = timeGetTime();
nRet = recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *) &client,
&dwSendSize);
// 等待ack
//此时要么是收到了东西、要么是超时了
while (1) {
    // 接收出错
    if (nRet == SOCKET_ERROR) {
        // 确实是接收超时
        if (WSAGetLastError() == 10060) {
            cout << "超时了" << endl;
            cout << "send2 seq ok_seq" << m1->seq << " " << m1->ok_seq <<
endl;

            //再次发送数据
            nRet = sendto(sListen, b, size, 0, (SOCKADDR *) &client,
sizeof(SOCKADDR));
            if (nRet == SOCKET_ERROR) {
                cout << "sendto Error " << WSAGetLastError() << endl;
                return;
            }
            cout << "超时返回初值" << endl;
            timeout = 3000;
            setsockopt(sListen, SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout,
sizeof(timeout));
            //再次等待
            t1 = timeGetTime();
            nRet = recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *)
&client, &dwSendSize);
            cout << "rcev2 seq ok_seq" << p->seq << " " << p->ok_seq <<
endl;

            // 返回等待ack状态
            continue;
        }
        else {
            // 超时以外的未知错误
            cout << "recv Error " << WSAGetLastError() << endl;
            return;
        }
    } else {
        //正常收到
        if (p->cond == ack && p->ok_seq == curr_seq &&
checksumchecker(buffertemp, p->seq - client_seq)) {
            //发送成功返回
            break;
        } else {
            cout << "损坏或不符合预期" << endl;
            //丢弃无用数据包
            memset(buffertemp, 0, 4096);
            t2 = timeGetTime();
            cout << "重置超时 " << 3000 - (double) (t2 - t1) << " ms" << endl;
            timeout = 3000 - (t2 - t1);

```

```

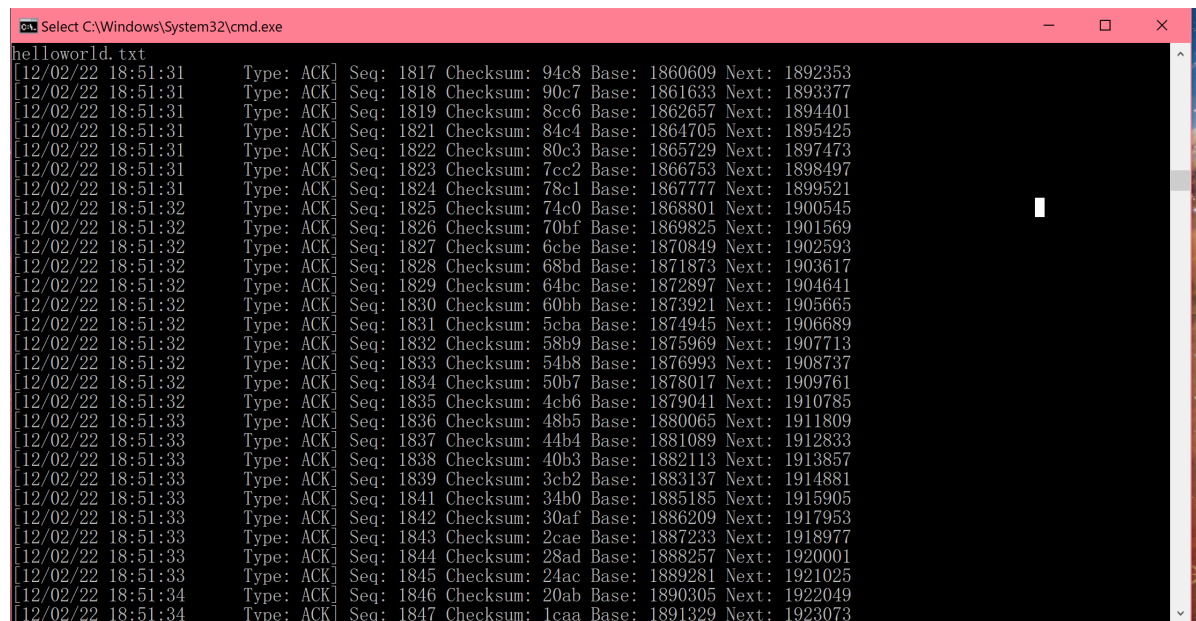
        setsockopt(sListen, SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout,
sizeof(timeout));
        //再次等待
        t1 = timeGetTime();
        //发送重复ACK
        nRet = recvfrom(sListen, buffertemp, 4096, 0, (SOCKADDR *)
&client, &dwsendSize);
        cout << "rcev3 seq ok_seq" << p->seq << " " << p->ok_seq <<
endl;

        continue;
    }
}
}
}
}

```

## 运行截图

1服务端开始发送：三次握手建立连接后，输入文件名称进行发送：



```

Select C:\Windows\System32\cmd.exe
helloworld.txt
[12/02/22 18:51:31 Type: ACK] Seq: 1817 Checksum: 94c8 Base: 1860609 Next: 1892353
[12/02/22 18:51:31 Type: ACK] Seq: 1818 Checksum: 90c7 Base: 1861633 Next: 1893377
[12/02/22 18:51:31 Type: ACK] Seq: 1819 Checksum: 8cc6 Base: 1862657 Next: 1894401
[12/02/22 18:51:31 Type: ACK] Seq: 1821 Checksum: 84c4 Base: 1864705 Next: 1895425
[12/02/22 18:51:31 Type: ACK] Seq: 1822 Checksum: 80c3 Base: 1865729 Next: 1897473
[12/02/22 18:51:31 Type: ACK] Seq: 1823 Checksum: 7cc2 Base: 1866753 Next: 1898497
[12/02/22 18:51:31 Type: ACK] Seq: 1824 Checksum: 78c1 Base: 1867777 Next: 1899521
[12/02/22 18:51:32 Type: ACK] Seq: 1825 Checksum: 74c0 Base: 1868801 Next: 1900545
[12/02/22 18:51:32 Type: ACK] Seq: 1826 Checksum: 70bf Base: 1869825 Next: 1901569
[12/02/22 18:51:32 Type: ACK] Seq: 1827 Checksum: 6cbe Base: 1870849 Next: 1902593
[12/02/22 18:51:32 Type: ACK] Seq: 1828 Checksum: 68bd Base: 1871873 Next: 1903617
[12/02/22 18:51:32 Type: ACK] Seq: 1829 Checksum: 64bc Base: 1872897 Next: 1904641
[12/02/22 18:51:32 Type: ACK] Seq: 1830 Checksum: 60bb Base: 1873921 Next: 1905665
[12/02/22 18:51:32 Type: ACK] Seq: 1831 Checksum: 5cba Base: 1874945 Next: 1906689
[12/02/22 18:51:32 Type: ACK] Seq: 1832 Checksum: 58b9 Base: 1875969 Next: 1907713
[12/02/22 18:51:32 Type: ACK] Seq: 1833 Checksum: 54b8 Base: 1876993 Next: 1908737
[12/02/22 18:51:32 Type: ACK] Seq: 1834 Checksum: 50b7 Base: 1878017 Next: 1909761
[12/02/22 18:51:32 Type: ACK] Seq: 1835 Checksum: 4cb6 Base: 1879041 Next: 1910785
[12/02/22 18:51:33 Type: ACK] Seq: 1836 Checksum: 48b5 Base: 1880065 Next: 1911809
[12/02/22 18:51:33 Type: ACK] Seq: 1837 Checksum: 44b4 Base: 1881089 Next: 1912833
[12/02/22 18:51:33 Type: ACK] Seq: 1838 Checksum: 40b3 Base: 1882113 Next: 1913857
[12/02/22 18:51:33 Type: ACK] Seq: 1839 Checksum: 3cb2 Base: 1883137 Next: 1914881
[12/02/22 18:51:33 Type: ACK] Seq: 1841 Checksum: 34b0 Base: 1885185 Next: 1915905
[12/02/22 18:51:33 Type: ACK] Seq: 1842 Checksum: 30af Base: 1886209 Next: 1917953
[12/02/22 18:51:33 Type: ACK] Seq: 1843 Checksum: 2cae Base: 1887233 Next: 1918977
[12/02/22 18:51:33 Type: ACK] Seq: 1844 Checksum: 28ad Base: 1888257 Next: 1920001
[12/02/22 18:51:33 Type: ACK] Seq: 1845 Checksum: 24ac Base: 1889281 Next: 1921025
[12/02/22 18:51:34 Type: ACK] Seq: 1846 Checksum: 20ab Base: 1890305 Next: 1922049
[12/02/22 18:51:34 Type: ACK] Seq: 1847 Checksum: 1caa Base: 1891329 Next: 1923073

```

2服务端正常发送过程中：显示收到的数据包类型、时间信息、序列号、校验和、窗口起始位置、窗口下一个发送位置

```
C:\Windows\System32\cmd.exe
[12/02/22 18:52:51 Type: ACK] Seq: 2961 Checksum: b03e Base: 3032065 Next: 3063809
[12/02/22 18:52:51 Type: ACK] Seq: 2962 Checksum: ac3d Base: 3033089 Next: 3064833
[12/02/22 18:52:51 Type: ACK] Seq: 2963 Checksum: a83c Base: 3034113 Next: 3065857
[12/02/22 18:52:52 Type: ACK] Seq: 2964 Checksum: a43b Base: 3035137 Next: 3066881
[12/02/22 18:52:52 Type: ACK] Seq: 2965 Checksum: a03a Base: 3036161 Next: 3067905
[12/02/22 18:52:52 Type: ACK] Seq: 2966 Checksum: 9c39 Base: 3037185 Next: 3068929
[12/02/22 18:52:52 Type: ACK] Seq: 2967 Checksum: 9838 Base: 3038209 Next: 3069953
[12/02/22 18:52:52 Type: ACK] Seq: 2968 Checksum: 9437 Base: 3039233 Next: 3070977
[12/02/22 18:52:52 Type: ACK] Seq: 2969 Checksum: 9036 Base: 3040257 Next: 3072001
[12/02/22 18:52:52 Type: ACK] Seq: 2970 Checksum: 8c35 Base: 3041281 Next: 3073025
[12/02/22 18:52:52 Type: ACK] Seq: 2972 Checksum: 8433 Base: 3043329 Next: 3074049
[12/02/22 18:52:52 Type: ACK] Seq: 2973 Checksum: 8032 Base: 3044353 Next: 3076097
[12/02/22 18:52:52 Type: ACK] Seq: 2974 Checksum: 7c31 Base: 3045377 Next: 3077121
[12/02/22 18:52:52 Type: ACK] Seq: 2975 Checksum: 7830 Base: 3046401 Next: 3078145
[12/02/22 18:52:52 Type: ACK] Seq: 2976 Checksum: 742f Base: 3047425 Next: 3079169
[12/02/22 18:52:52 Type: ACK] Seq: 2977 Checksum: 702e Base: 3048449 Next: 3080193
[12/02/22 18:52:52 Type: ACK] Seq: 2978 Checksum: 6c2d Base: 3049473 Next: 3081217
[12/02/22 18:52:52 Type: ACK] Seq: 2979 Checksum: 682c Base: 3050497 Next: 3082241
[12/02/22 18:52:52 Type: ACK] Seq: 2980 Checksum: 642b Base: 3051521 Next: 3083265
[12/02/22 18:52:52 Type: ACK] Seq: 2981 Checksum: 602a Base: 3052545 Next: 3084289
[12/02/22 18:52:52 Type: ACK] Seq: 2982 Checksum: 5c29 Base: 3053569 Next: 3085313
[12/02/22 18:52:52 Type: ACK] Seq: 2983 Checksum: 5828 Base: 3054593 Next: 3086337
[12/02/22 18:52:52 Type: ACK] Seq: 2984 Checksum: 5427 Base: 3055617 Next: 3087361
[12/02/22 18:52:52 Type: ACK] Seq: 2985 Checksum: 5026 Base: 3056641 Next: 3088385
[12/02/22 18:52:52 Type: ACK] Seq: 2986 Checksum: 4c25 Base: 3057665 Next: 3089409
[12/02/22 18:52:52 Type: ACK] Seq: 2987 Checksum: 4824 Base: 3058689 Next: 3090433
[12/02/22 18:52:53 Type: ACK] Seq: 2988 Checksum: 4423 Base: 3059713 Next: 3091457
[12/02/22 18:52:53 Type: ACK] Seq: 2989 Checksum: 4022 Base: 3060737 Next: 3092481
[12/02/22 18:52:53 Type: ACK] Seq: 2990 Checksum: 3c21 Base: 3061761 Next: 3093505
[12/02/22 18:52:53 Type: ACK] Seq: 2992 Checksum: 341f Base: 3063809 Next: 3094529
```

3客户端正常接收过程中：显示时间、数据包类型、序列号、校验和

```
C:\Windows\System32\cmd.exe
[12/02/22 18:52:26 Type: ACK] Seq: 2393089 Checksum: 57a0
[12/02/22 18:52:26 Type: ACK] Seq: 2394113 Checksum: 539f
[12/02/22 18:52:26 Type: ACK] Seq: 2395137 Checksum: 4f9e
[12/02/22 18:52:26 Type: ACK] Seq: 2396161 Checksum: 4b9d
[12/02/22 18:52:26 Type: ACK] Seq: 2397185 Checksum: 479c
[12/02/22 18:52:26 Type: ACK] Seq: 2398209 Checksum: 439b
[12/02/22 18:52:26 Type: ACK] Seq: 2399233 Checksum: 3f9a
[12/02/22 18:52:26 Type: ACK] Seq: 2400257 Checksum: 3b99
[12/02/22 18:52:26 Type: ACK] Seq: 2401281 Checksum: 3798
[12/02/22 18:52:26 Type: ACK] Seq: 2402305 Checksum: 3397
[12/02/22 18:52:26 Type: ACK] Seq: 2403329 Checksum: 2f96
[12/02/22 18:52:26 Type: ACK] Seq: 2404353 Checksum: 2b95
[12/02/22 18:52:26 Type: ACK] Seq: 2405377 Checksum: 2794
[12/02/22 18:52:26 Type: ACK] Seq: 2406401 Checksum: 2393
[12/02/22 18:52:26 Type: ACK] Seq: 2407425 Checksum: 1f92
[12/02/22 18:52:26 Type: ACK] Seq: 2408449 Checksum: 1b91
[12/02/22 18:52:26 Type: ACK] Seq: 2409473 Checksum: 1790
[12/02/22 18:52:26 Type: ACK] Seq: 2410497 Checksum: 138f
[12/02/22 18:52:26 Type: ACK] Seq: 2411521 Checksum: f8e
[12/02/22 18:52:26 Type: ACK] Seq: 2412545 Checksum: b8d
[12/02/22 18:52:26 Type: ACK] Seq: 2413569 Checksum: 78c
[12/02/22 18:52:26 Type: ACK] Seq: 2414593 Checksum: 38b
[12/02/22 18:52:26 Type: ACK] Seq: 2415617 Checksum: ff89
[12/02/22 18:52:26 Type: ACK] Seq: 2416641 Checksum: fb88
[12/02/22 18:52:26 Type: ACK] Seq: 2417665 Checksum: f787
[12/02/22 18:52:27 Type: ACK] Seq: 2418689 Checksum: f386
[12/02/22 18:52:27 Type: ACK] Seq: 2419713 Checksum: ef85
[12/02/22 18:52:27 Type: ACK] Seq: 2420737 Checksum: eb84
[12/02/22 18:52:27 Type: ACK] Seq: 2421761 Checksum: e783
[12/02/22 18:52:27 Type: ACK] Seq: 2422785 Checksum: e382
```

4服务端遇到超时重传：

```
C:\Windows\System32\cmd.exe
[12/02/22 18:51:11 Type: ACK] Seq: 1788 Checksum: 8a6 Base: 1830913 Next: 1859585
[12/02/22 18:51:11 Type: ACK] Seq: 1789 Checksum: 4e5 Base: 1831937 Next: 1859585
[12/02/22 18:51:11 Type: ACK] Seq: 1791 Checksum: fce2 Base: 1833985 Next: 1859585
[12/02/22 18:51:11 Type: ACK] Seq: 1792 Checksum: f8e1 Base: 1835009 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1793 Checksum: f4e0 Base: 1836033 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1794 Checksum: f0df Base: 1837057 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1795 Checksum: ecde Base: 1838081 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1796 Checksum: e8dd Base: 1839105 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1797 Checksum: e4dc Base: 1840129 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1798 Checksum: e0db Base: 1841153 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1799 Checksum: dcd4 Base: 1842177 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1800 Checksum: d8d9 Base: 1843201 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1801 Checksum: d4d8 Base: 1844225 Next: 1859585
[12/02/22 18:51:12 Type: ACK] Seq: 1802 Checksum: d0d7 Base: 1845249 Next: 1859585
[12/02/22 18:51:13 Type: ACK] Seq: 1803 Checksum: ccd6 Base: 1846273 Next: 1859585
[12/02/22 18:51:13 Type: ACK] Seq: 1804 Checksum: c8d5 Base: 1847297 Next: 1859585
[12/02/22 18:51:13 Type: ACK] Seq: 1805 Checksum: c4d4 Base: 1848321 Next: 1859585
[12/02/22 18:51:13 Type: ACK] Seq: 1806 Checksum: c0d3 Base: 1849345 Next: 1859585
[12/02/22 18:51:13 Type: ACK] Seq: 1807 Checksum: bcd2 Base: 1850369 Next: 1859585
[12/02/22 18:51:13 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
发现超时
进行超时重传
[12/02/22 18:51:13 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:13 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:14 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:14 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:14 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:14 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:14 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:14 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
[12/02/22 18:51:14 Type: ACK] Seq: 1808 Checksum: bcd1 Base: 1850369 Next: 1859585
```

5客户端发现重传的重复数据包：



## 遇到的问题

---

曾经在接受时重复扣去transmission的大小，导致文件中隔一段就有数个字节为全0，修改后传输正常。