



南開大學
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

实验 3-2 GBN

学号：1911437

姓名：刘浩通

年级：2019 级

专业：计算机科学与技术

2021 年 12 月 11 日

目录

一、 实验要求	1
二、 实验设计	1
(一) 设计协议功能	1
(二) UDP 报文设计	1
(三) 两次握手设计	2
(四) 两次挥手设计	3
(五) 差错检验	3
(六) 发送端设计 (GBN)	4
(七) 接收端设计 (GBN)	4
(八) 超时重传	5
三、 具体代码实现	5
(一) 校验和的计算	5
(二) 两次握手建立连接	6
(三) 两次挥手实现	9
(四) 滑动窗口实现 (重点)	11
(五) 预处理数据 (提前将数据转换成数据报)	11
(六) 发送端发送数据报线程 (主线程)	12
(七) 发送端接收 ACK 回复线程 (子线程)	13
(八) 接收端收到数据报并且回复	15
四、 实验结果	18

一、 实验要求

实验 3-2: 在实验 3-1 的基础上, 将停等机制改成基于滑动窗口的流量控制机制, 采用固定窗口大小, 支持累积确认, 完成给定测试文件的传输。

二、 实验设计

本次实验使用的 GBN(回退 N) 滑动窗口协议。

1. 允许发送端发出 N 个未得到确认的分组
2. 需要增加序列号范围
 - 分组首部中增加 k 位的序列号, 序列号空间为 $[0, 2^k-1]$
3. 采用累积确认, 只确认连续正确接收分组的最大序列号
 - 可能接收到重复的 ACK
4. 发送端设置定时器, 定时器超时时, 重传所有未确认的分组

(一) 设计协议功能

1. 发送方和接收方有握手过程, 确认连接
2. 数据可以切包传输, 保存序列号
3. 数据报有差错检测功能, 正确接收返回确认
4. 数据报使用 GBN 滑动窗口传输机制和序列号, 不会出现失序情况
5. 发送端有超时重传, 解决数据包丢失的情况
6. 发送方和接收方有挥手过程, 断开连接

(二) UDP 报文设计

本次实验的报文结构是在上次实验的报文基础上加以改动。

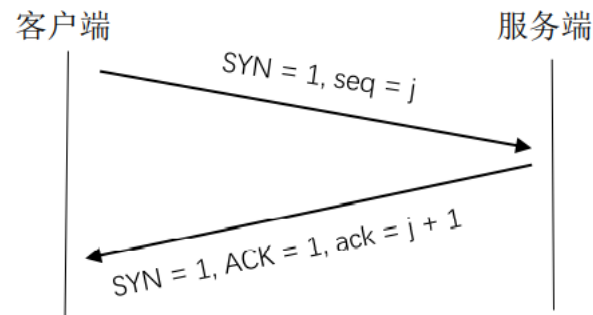
8	8
校验和(checksum)	
标志位(flag)	填充0(暂时不用)
报文长度(length)	
序号(seq)	
数据(Data)	

1. 校验和 (checksum): 16 位, 用于判断报文是否出错
2. 标志位 (Flag): 8 位, ACK 标志字 (接收消息用于回复确认)、SYN 标志字段 (握手阶段使用)、SF 标志字段 (信息, 标明未结束)、EF 标志字段 (传输的最后一条信息)、FIN 标志字段 (挥手阶段使用)。
3. 报文长度 (Length): 16 位, 整个报文的长度, 数据字段长度就是减去前面定长的头部信息。
4. 序号 (seq): 16 位, 本次实验使用的是 GBN 滑动窗口协议, 给与足够长度的序号
5. data: 应用层的数据

(三) 两次握手设计

单向传输, 二次握手确认连接

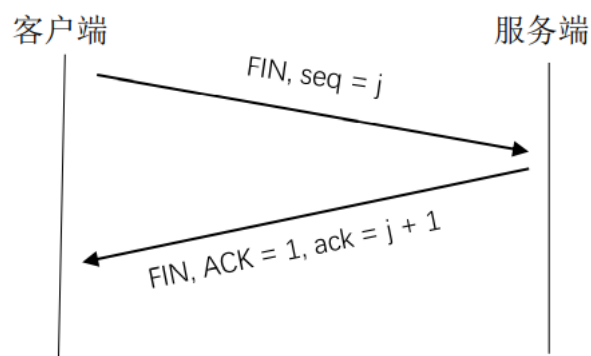
1. 发送端向接收端发送具有 SYN 的请求握手报文信息
2. 接收端收到信息后, 回复给发送端具有 SYN+ACK 的报文信息



(四) 两次挥手设计

单向传输，二次挥手断开连接

1. 发送端向接收端发送具有 FIN 的请求断开连接报文信息
2. 接收端收到信息后，回复给发送端具有 FIN+ACK 的报文信息



(五) 差错检验

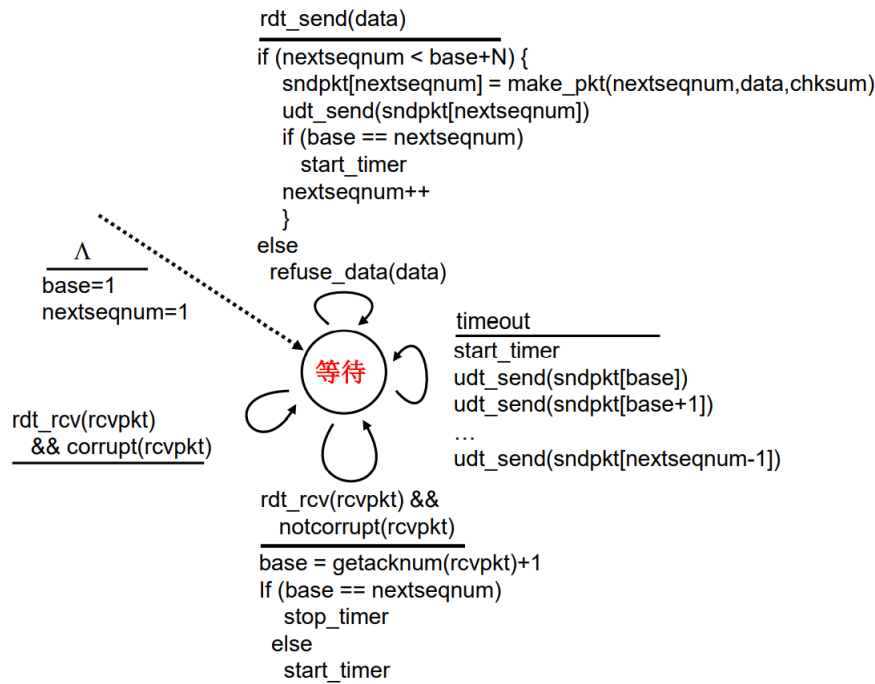
1. 发送端：

对要发送的 UDP 数据报，校验和域清零，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。

2. 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。

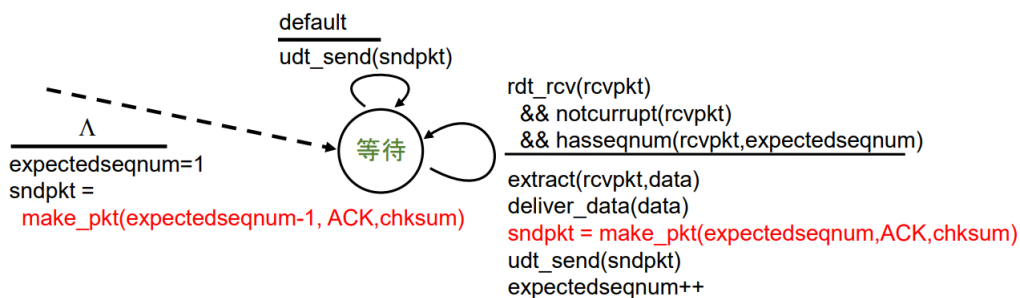
(六) 发送端设计 (GBN)



GBN 发送端 FSM 如上图所示，有以下 4 个事件：

1. 应用层发送数据：如果 $\text{nextseqnum} < \text{base} + N$ ，即滑动窗口中还有空余位置，则将数据封装后发送至接收端。如果发送前缓冲区为空则开启定时器；如果滑动窗口已满，则拒绝应用层发送数据。
2. 收到的报文损坏：将报文丢弃
3. 收到的 ack 报文未损坏： $\text{base} = \text{acknum} + 1$ ，如果滑动窗口变为空，则停止定时器，否则重启定时器。
4. 超时：将 $[\text{base}, \text{nextseqnum})$ 区间内的报文重传给接收端

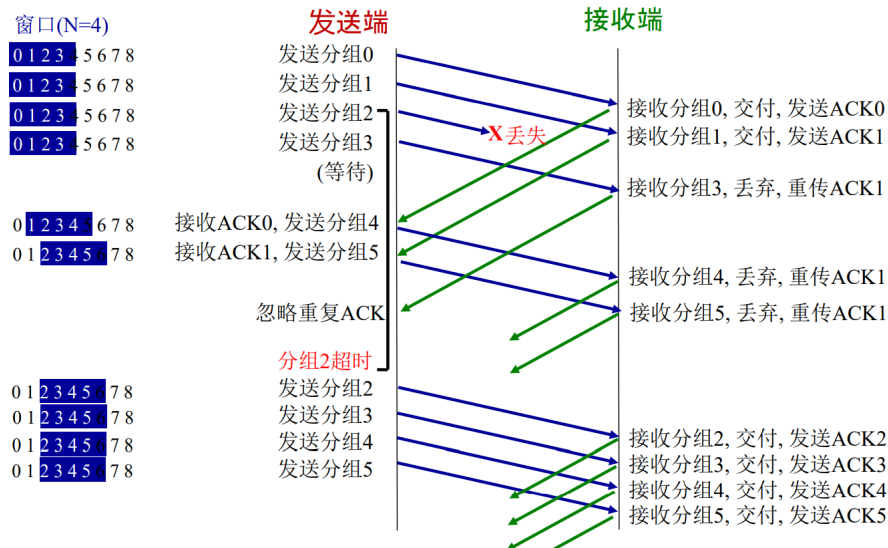
(七) 接收端设计 (GBN)



GBN 接收端 FSM 如上图所示，有以下两个事件：

1. 接收的报文未损坏并且按序: 向发送端发送 ack, 确认号为正确接收的最高序号, 然后将数据交付给应用层。
2. 接受的报文损坏或失序: 向发送端发送 ack, 确认号为正确接收的最高序号。

(八) 超时重传



再上图显示的示例中，如果在发送端发送的过程中发生了丢包。

接收端没有收到预期要收到序号的数据报，则会则会重传上一次发送的数据包。

接受端会忽略重复 ACK，并且等待分组超时会重传 [base, nextseqnum) 之间的报文

三、具体代码实现

(一) 校验和的计算

```

1  USHORT checksum(char* buf, int len) { // 计算校验和
2      unsigned char* ubuf = (unsigned char*)buf;
3      register ULONG sum = 0;
4      while (len) {
5          USHORT temp = USHORT(*ubuf << 8) + USHORT(*(ubuf + 1));
6          sum += temp;
7          if (sum & 0xFFFF0000){
8              sum &= 0xFFFF;
9              sum++;
10         }
11         ubuf += 2;
12         len -= 2;
13     }
14     return ~(sum & 0xFFFF);
15 }

```

会传入一个 char 类型数组和数组长度 len，len 是偶数。因为是 16 进制进行校验计算，而 char 是 8 位，所以是将两个 char 组成 16 位进行校验和计算。在调用此函数时，会把数组补全成偶数长度，补全用 0。

(二) 两次握手建立连接

发送端：

```

1 void shake_hand() {
2     while (1) {
3         char tmp[8];
4         tmp[2] = SYN;
5         tmp[3] = 0;
6         tmp[4] = 6 >> 8;
7         tmp[5] = 6 & 0xff;
8         tmp[6] = curSeq << 8;
9         tmp[7] = curSeq & 0xff;
10
11         USHORT ck = checksum(tmp + 2, 6);
12         tmp[0] = ck >> 8;
13         tmp[1] = ck & 0xff;
14
15         //cout << checksum(tmp, 6) << endl;
16
17         sendto(client, tmp, 8, 0, (sockaddr*)&serverAddr, sizeof(
            serverAddr));
18
19         clock_t begintime = clock();
20         char recv[8];
21         int lentmp = sizeof(clientAddr);
22         int fail_send = 0;
23         while (recvfrom(client, recv, 8, 0, (sockaddr*)&serverAddr, &
            lentmp) == SOCKET_ERROR)
24             if (clock() - begintime > timeout) {
25                 fail_send = 1;
26                 break;
27             }
28         double sampleRTT = clock() - begintime;
29         SHORT seq = (USHORT)((unsigned char)recv[6] << 8) + (USHORT)(
            unsigned char)recv[7];
30         if (fail_send == 0 && checksum(recv, 8) == 0 && recv[2] == (
            SYN + ACK) && seq == curSeq) {
31             addSampleRTT(sampleRTT);
32             curSeq++;
33             return;
34         }
35     }
36 }

```


接收端:

```

1 void wait_shake_hand() {
2     while (1) {
3         char recv[Mlenx+8], send[8];
4         memset(recv, 0, sizeof(recv));
5         int connect = 0;
6         int lentmp = sizeof(clientAddr);
7         while (recvfrom(server, recv, Mlenx + 8, 0, (sockaddr*)&
8             clientAddr, &lentmp) == SOCKET_ERROR);
9         SHORT seq = (USHORT)((unsigned char)recv[6] << 8) + (USHORT)(
10             unsigned char)recv[7];
11         if (checksum(recv, Mlenx + 8) == 0 ) {
12             if (recv[2] == SYN && seq != curSeq) {
13                 cout << "ACK 丢失" << endl;
14
15                 send[2] = (SYN + ACK);
16                 send[3] = 0;
17                 send[4] = 6 >> 8;
18                 send[5] = 6 & 0xff;
19
20                 send[6] = seq << 8;
21                 send[7] = seq & 0xff;
22
23                 USHORT ck = checksum(send + 2, 6);
24                 send[0] = ck >> 8;
25                 send[1] = ck & 0xff;
26                 sendto(server, send, 8, 0, (sockaddr*)&
27                     clientAddr, sizeof(clientAddr));
28
29                 cout << "收到:";
30                 output(recv[2]);
31                 cout << " Length:" << (USHORT)((unsigned char)
32                     recv[4] << 8) + (USHORT)(unsigned char)
33                     recv[5];
34                 cout << " Checksum:" << (USHORT)((unsigned
35                     char)recv[0] << 8) + (USHORT)(unsigned
36                     char)recv[1];
37                 cout << " Seq" << seq;
38                 cout << endl;
39                 cout << "发送:";
40                 output(send[2]);
41                 cout << " Length:" << (USHORT)((unsigned char)
42                     send[4] << 8) + (USHORT)(unsigned char)
43                     send[5];
44                 cout << " Checksum:" << ck;
45                 cout << " Seq:" << seq;
46                 cout << endl;

```

```
39         continue;
40     }
41     else if (seq != curSeq && (recv[2] == SF || recv[2]
42         == EF)) {
43     }
44 }
45
46 if (checksum(recv, Mlenx + 8) != 0 || recv[2] != SYN || seq
47     != curSeq)
48     continue;
49
50 //收到了SYN,若校验成功, 回复
51 send[2] = (SYN + ACK);
52 send[3] = 0;
53 send[4] = 6 >> 8;
54 send[5] = 6 & 0xff;
55 send[6] = curSeq << 8;
56 send[7] = curSeq & 0xff;
57
58 USHORT ck = checksum(send + 2, 6);
59 send[0] = ck >> 8;
60 send[1] = ck & 0xff;
61 sendto(server, send, 8, 0, (sockaddr*)&clientAddr, sizeof(
62     clientAddr));
63
64 cout << "收到:";
65 output(recv[2]);
66 cout << " Length:" << (USHORT)((unsigned char)recv[4] << 8) +
67     (USHORT)(unsigned char)recv[5];
68 cout << " Checksum:" << (USHORT)((unsigned char)recv[0] << 8)
69     + (USHORT)(unsigned char)recv[1];
70 cout << " Seq" << seq;
71 cout << endl;
72 cout << "发送:";
73 output(send[2]);
74 cout << " Length:" << (USHORT)((unsigned char)send[4] << 8) +
75     (USHORT)(unsigned char)send[5];
76 cout << " Checksum:" << ck;
77 cout << " Seq:" << curSeq;
78 cout << endl;
79 curSeq++;
80 break;
81 }
82 }
```

(三) 两次挥手实现

发送端:

```
1 // 两次握手和两次挥手，就不用滑动窗口了，就一个数据报。。。
2 void wave_hand() {
3     int tot_fail = 0;
4     while (1) {
5
6         char tmp[8];
7         tmp[2] = FIN;
8         tmp[3] = 0;
9         tmp[4] = 8 >> 8;
10        tmp[5] = 8 & 0xff;
11        tmp[6] = curSeq >> 8;
12        tmp[7] = curSeq & 0xff;
13
14        USHORT ck = checksum(tmp + 2, 6);
15        tmp[0] = ck >> 8;
16        tmp[1] = ck & 0xff;
17        sendto(client, tmp, 8, 0, (sockaddr*)&serverAddr, sizeof(
            serverAddr));
18
19        double begintime = clock();
20        char recv[8];
21        int lentmp = sizeof(serverAddr);
22        int fail_send = 0;
23        while (recvfrom(client, recv, 8, 0, (sockaddr*)&serverAddr, &
            lentmp) == SOCKET_ERROR) {
24            if (clock() - begintime > timeout) {
25                fail_send = 1;
26                tot_fail++;
27                break;
28            }
29        }
30        double sampleRTT = clock() - begintime;
31        SHORT seq = (USHORT)((unsigned char)recv[6] << 8) + (USHORT)(
            unsigned char)recv[7];
32        if (fail_send == 0 && checksum(recv, 8) == 0 && recv[2] == (
            FIN + ACK) && seq == curSeq) {
33            addSampleRTT(sampleRTT);
34            curSeq++;
35            break;
36        }
37        else {
38            if (tot_fail == 10) {
39                printf("断开失败，释放资源");
40                break;
41            }

```

```

42         continue;
43     }
44 }
45 }

```

接收端:

```

1 void wait_wave_hand() {
2     while (1) {
3         char recv[8], send[8];
4         int lentmp = sizeof(clientAddr);
5         while (recvfrom(server, recv, 8, 0, (sockaddr*)&clientAddr, &
6             lentmp) == SOCKET_ERROR);
7
8         SHORT seq = (USHORT)((unsigned char)recv[6] << 8) + (USHORT)(
9             unsigned char)recv[7];
10        if (checksum(recv, 8) != 0 || recv[2] != (char)FIN || seq !=
11            curSeq)
12            continue;
13
14        send[2] = (ACK + FIN);
15        send[3] = 0;
16        send[4] = 6 >> 8;
17        send[5] = 6 & 0xff;
18        send[6] = curSeq >> 8;
19        send[7] = curSeq & 0xff;
20
21        USHORT ck = checksum(send + 2, 6);
22        send[0] = ck >> 8;
23        send[1] = ck & 0xff;
24        sendto(server, send, 8, 0, (sockaddr*)&clientAddr, sizeof(
25            clientAddr));
26        cout << "收到:";
27        output(recv[2]);
28        cout << " Length:" << (USHORT)((unsigned char)recv[4] << 8) +
29            (USHORT)(unsigned char)recv[5];
30        cout << " Checksum:" << (USHORT)((unsigned char)recv[0] << 8)
31            + (USHORT)(unsigned char)recv[1];
32        cout << " Seq" << seq;
33        cout << endl;
34        cout << "发送:";
35        output(send[2]);
36        cout << " Length:" << (USHORT)((unsigned char)send[4] << 8) +
37            (USHORT)(unsigned char)send[5];
38        cout << " Checksum:" << ck;
39        cout << " Seq:" << curSeq;
40        cout << endl;
41        curSeq++;
42        break;

```

```

36     }
37 }

```

(四) 滑动窗口实现 (重点)

重点是发送端实现

一些重要的参数说明

```

1 char buffer[200000000]; //读取的文件数据
2 int len; //文件数据的长度
3 vector<char*>messages; //存放文件数据包
4 queue<double> timesaver; //保存发送时间的, 顺序存入
5 // 发一个数据包, 存一个
6 // 如果发生超时, 清空队列, 重新根据sendbase发送数据
7
8 int lastSeq = 0; // 发送文件最后一个数据报的序号
9 int startSeq = 0; // 发送文件第一个数据报的序号
10 int curSeq = 0; // 已经发送的最后一个数据报的编号
11 int SendBase = 0; // 等待确认的第一个数据报的编号
12 // 其中 curSeq - SendBase <= windows_size
13 bool closeall = false; // 用于关闭接收数据报的线程
14 int window_size = 1; // 窗口大小

```

(五) 预处理数据 (提前将数据转换成数据报)

考虑到发送端会一次性连续发送多个连续序号的数据报, 为了防止写入要发送数据报拖延发送时间, 会提前把所有包处理好, 存入队列中。

```

1 // 造文件数据报的函数
2 char* makeDatagram(char* message, int lent, bool last, int seq) { //length是数
    据的长度
3     char* tmp;
4     int length = lent % 2 == 0 ? lent : lent + 1;
5     tmp = new char[length + 8];
6     tmp[length + 7] = 0;
7     if (last) {
8         tmp[2] = EF;
9         lastSeq = seq;
10    }
11    else {
12        tmp[2] = SF;
13    }
14    tmp[3] = 0;
15    tmp[4] = (lent + 8) >> 8;
16    tmp[5] = (lent + 8) & 0xff;
17    tmp[6] = seq >> 8;
18    tmp[7] = seq & 0xff;
19    for (int i = 8; i < lent + 8; i++)

```

```

20     tmp[i] = message[i - 8];
21     USHORT ck = checksum(tmp + 2, length + 6);
22     tmp[0] = ck >> 8;
23     tmp[1] = ck & 0xff;
24     cout << seq << " " << (USHORT)((unsigned char)tmp[6] << 8) + (USHORT)
        (unsigned char)tmp[7] << endl;
25     return tmp;
26 }
27 int main(){
28     ....
29
30     //现在buffer中有所有文件内容，len是长度
31     //然后开始造包
32     int package_num = len / Mlenx + (len % Mlenx != 0); //看要发送几个包
33     int seq = curSeq;
34     startSeq = curSeq;
35     for (int i = 0; i < package_num; i++) {
36
37         bool last = (i == (package_num - 1)); //是否是最后一个包
38         int length = last ? len - (package_num - 1) * Mlenx : Mlenx
            ; //数据的长度
39
40         char* package = makeDatagram(buffer + i * Mlenx, length, last
            , seq);
41         seq++;
42         messages.push_back(package);
43     }
44     //到这一步，数据报全部准备完毕了
45     ....
46 }

```

(六) 发送端发送数据报线程 (主线程)

```

1 void GBN_Send() {
2     // 滑动窗口发送数据包
3     recver = CreateThread(NULL, NULL, RecvThread, (LPVOID)client, NULL,
        NULL);
4     //创建接收线程
5     while (true)
6     {
7         while (curSeq - SendBase < (window_size))
8         {
9             int last = curSeq + window_size ;
10            if (last > lastSeq)
11                last = lastSeq;
12            cout << curSeq << endl;
13            for (int i = curSeq + 1; i <= last; i++) {

```

```

14         char* tmp = messages[i - startSeq];
15
16         int tmp_len = ((unsigned char)tmp[4] << 8) +
17             (unsigned char)tmp[5];
18
19         tmp_len = tmp_len % 2 == 0 ? tmp_len :
20             tmp_len + 1;
21
22         USHORT seq = (USHORT)((unsigned char)tmp[6]
23             << 8) + (USHORT)(unsigned char)tmp[7];
24         cout << "发送 " << seq << "号数据报" << endl;
25         sendto(client, tmp, tmp_len, 0, (sockaddr*)&
26             serverAddr, sizeof(serverAddr));
27         timesaver.push(clock());
28     }
29     curSeq = last;
30     if (closeall)
31         return;
32 }
33 if (closeall)
34     return;
35 }

```

其中 curSeq、SendBase、timesaver 和 closeall 是发送线程和接收线程都会修改的数据，我是将接收线程优先级提高，来避免冲突。

发送线程在满足 $\text{curSeq} - \text{SendBase} < (\text{window_size})$ 的条件下会按序发送数据报，直至长度等于窗口大小。每发送一个数据报，则会记录发送数据报的时间存入到队列结构 timesaver 中去。到了接收线程，会读取队列头部的时间来判断是否超时，如果在超时时间内收到队列的 ACK，则会将 timesaver 头部的时间弹出。

(七) 发送端接收 ACK 回复线程 (子线程)

大致思路看代码中注释

1. 在时间队列 timesaver 为空时，接收线程空转
2. 收到一个数据报，先进行差错检验，若检验通过判断收到的 ACK 是否是要确认的 baseSeq 的 ack，若是，弹出 timesaver 头部时间，移动窗口。若不是，继续等待直到超时。
3. 若发生超时情况，清空 timesaver，重新等待数据报发送存入时间。调整 curSeq 到 SendBase，即从最近一个未确认序号开始窗口重新发送。
4. 累计确认：若当前收到的 $\text{ACKseq} > \text{SendBase}$ 并且 $\text{ACK} \leq \text{CurSeq}$ ，也就是说收到的 ack 序号在发送窗口范围内，说明 Ackseq 之前的所有序号的数据报全部被接收，所以重新调整 Sendbase 到 AckSeq。
5. 如果最后一个数据报的 ACK 也收到了，跳转 closeall 告知发送线程可以退出。接收线程关闭。

```

1  DWORD WINAPI RecvThread(LPVOID lparam) {
2      // 这玩意是个接收线程，设计出来
3      // 1.接收来及接收方的ACK回复，来推动窗口移动
4      // 2.检查最早发的报文是否超时
5      // 感觉会出现数据冲突，事件队列的问题
6      cout << "线程启动！！！！！！" << endl;
7      int lastAckSeq = 1;
8      while (true) {
9          while (!timesaver.empty()) //时间队列有东西
10         {
11             SetPriorityClass(recver, HIGH_PRIORITY_CLASS);
12             double start = timesaver.front(); //获取第一个时间也
13             //是最早的时间
14             char recv[8]; // 接收数组
15             int lentmp = sizeof(clientAddr);
16             bool flag = false;
17             while (recvfrom(client, recv, 8, 0, (sockaddr*)&
18                 serverAddr, &lentmp) == SOCKET_ERROR) {
19                 if (clock() - start > timeout) {
20                     //现在出现了时间超时
21                     timesaver = queue<double>(); //队列清
22                     //空
23                     curSeq = SendBase;
24                     flag = true;
25                     cout << "超时啦" << curSeq << " " <<
26                         SendBase << endl;
27                     break;
28                 }
29             }
30             //若超时，时间队列被清空，跳出这层循环，直至等待时间
31             //队列有东西
32             if (flag)
33                 break;
34             double sampleRTT = clock() - start;
35             SHORT seq = (USHORT)((unsigned char)recv[6] << 8) + (
36                 USHORT)(unsigned char)recv[7];
37             if (checksum(recv, 8) == 0 && recv[2] == ACK) {
38                 if (seq == SendBase+1) {
39                     // 收到了正确的报文，窗口向后面移动一
40                     // 下
41                     addSampleRTT(sampleRTT);
42                     timesaver.pop();
43                     lastAckSeq = seq;
44                     SendBase++;
45                     cout << "收到了 " << seq << "号报文的
46                         ACK" << endl;
47                     if (lastSeq == seq) {

```



```

40         closeall = true;
41         cout << "都收到了" << endl;
42         cout << curSeq << endl;
43         return 0;
44     }
45 }
46 else if (seq > SendBase + 1 && seq <= curSeq)
47 {
48     addSampleRTT(sampleRTT);
49
50     int sub = seq - SendBase;
51     for (int i = 0; i < sub; i++)
52         timesaver.pop();
53     SendBase = seq;
54     cout << "收到了 " << seq << "号报文的
55         ACK" << endl;
56     if (lastSeq == seq) {
57         closeall = true;
58         cout << "都收到了" << endl;
59         cout << curSeq << endl;
60         return 0;
61     }
62 }
63 else {
64     cout << "非正确回复, 编号: " << seq
65         << endl;
66
67     if (seq != lastAckSeq) {
68
69         lastAckSeq = seq;
70
71         SendBase = seq;
72         curSeq = seq;
73
74         timesaver = queue<double>();
75         break;
76     }
77 }
78 }
79 SetPriorityClass(recver, NORMAL_PRIORITY_CLASS);
80 }
81 }
82 }

```

(八) 接收端收到数据报并且回复

在这里接收端是使用一个线程完成数据的接收和回复 ACK

1. 如果收到了正在等待的 seq 数据报，则回复 ACK，将 data 写入到数组中去。
2. 如果收到的 seq 与等待的 seq 不同，回复上一个 ACK 信息，继续等待。

```

1 void recv_message(char* message, int& len_rcv) {
2     char rcv[Mlenx + 8];
3     int lentmp = sizeof(clientAddr);
4     len_rcv = 0;
5     while (1) {
6         while (1) {
7             memset(rcv, 0, sizeof(rcv));
8             while (recvfrom(server, rcv, Mlenx + 8, 0, (sockaddr
9                 *)&clientAddr, &lentmp) == SOCKET_ERROR);
10            char send[8];
11
12            SHORT seq = (USHORT)((unsigned char)rcv[6] << 8) + (
13                USHORT)(unsigned char)rcv[7];
14            cout << seq << " ";
15            if (checksum(rcv, Mlenx + 8) == 0 && seq == curSeq)
16            {
17                // 正常收到的情况下啊，返回ACK
18
19                send[2] = ACK;
20                send[3] = 0;
21                send[4] = 6 >> 8;
22                send[5] = 6 & 0xff;
23                send[6] = curSeq >> 8;
24                send[7] = curSeq & 0xff;
25                USHORT ck = checksum(send + 2, 6);
26
27                send[0] = ck >> 8;
28                send[1] = ck & 0xff;
29                sendto(server, send, 8, 0, (sockaddr*)&
30                    clientAddr, sizeof(clientAddr));
31
32                cout << "收到:";
33                output(rcv[2]);
34                cout << " Length:" << (USHORT)((unsigned char
35                    )rcv[4] << 8) + (USHORT)(unsigned char)
36                    rcv[5];
37                cout << " Checksum:" << (USHORT)((unsigned
38                    char)rcv[0] << 8) + (USHORT)(unsigned
39                    char)rcv[1];
40                cout << " Seq:" << seq;
41                cout << endl;
42                cout << "发送:";
43                output(send[2]);
44                cout << " Length:" << 6;
45                cout << " Checksum:" << ck;

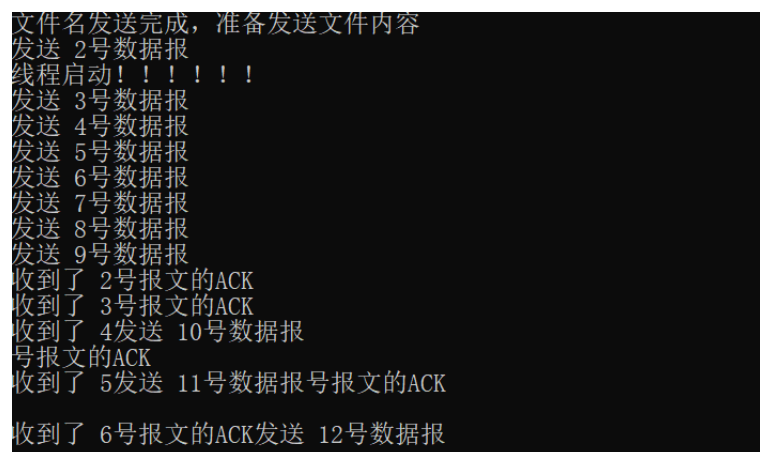
```

```
38         cout << " Seq:" << curSeq ;
39         cout << endl;
40         curSeq++;
41         break;
42     }
43     else if (checksum(recv, Mlenx + 8) == 0 && seq !=
44             curSeq) {
45         //校验通过了，序号对不上
46         //这个是接收方ACK丢失，发送方重传的情况
47         //亦或者是，有一条丢了，后续继续发，反正我们
48         //只要我们需要的那一条
49         // 回复的序号是上一条ACK回复
50         cout << "?" << endl;
51         send[2] = ACK;
52         send[3] = 0;
53         send[4] = 6 >> 8;
54         send[5] = 6 & 0xff;
55         send[6] = (curSeq - 1) >> 8;
56         send[7] = (curSeq - 1) & 0xff;
57
58         USHORT ck = checksum(send + 2, 6);
59
60         send[0] = ck >> 8;
61         send[1] = ck & 0xff;
62         sendto(server, send, 8, 0, (sockaddr*)&
63             clientAddr, sizeof(clientAddr));
64
65         cout << "收到:";
66         output(recv[2]);
67         cout << " Length:" << (USHORT)((unsigned char
68             )recv[4] << 8) + (USHORT)(unsigned char)
69             recv[5];
70         cout << " Checksum:" << (USHORT)((unsigned
71             char)recv[0] << 8) + (USHORT)(unsigned
72             char)recv[1];
73         cout << " Seq:" << seq;
74         cout << endl;
75
76         cout << "发送:";
77         output(send[2]);
78         cout << " Length:" << 6;
79         cout << " Checksum:" << ck;
80         cout << " Seq:" << curSeq - 1;
81         cout << endl;
82     }
83     cout << checksum(recv, Mlenx + 8) << endl;
84 }
```

```
79         USHORT length = (USHORT)((unsigned char)recv[4] << 8) + (  
80             USHORT)(unsigned char)recv[5];  
81  
82         for (int i = 8; i < length; i++) {  
83             message[len_recv] = recv[i];  
84             len_recv++;  
85         }  
86  
87         if (EF == recv[2]) {  
88             cout << curSeq << endl;  
89             break;  
90         }  
91     }
```

四、 实验结果

选定测试文件为 2.png，设置时延为 5ms，设置窗口大小为 8。
传输时间为 10.767s，吞吐率为 4382.65Kbps。



```
文件名发送完成, 准备发送文件内容  
发送 2号数据报  
线程启动!!!!  
发送 3号数据报  
发送 4号数据报  
发送 5号数据报  
发送 6号数据报  
发送 7号数据报  
发送 8号数据报  
发送 9号数据报  
收到了 2号报文的ACK  
收到了 3号报文的ACK  
收到了 4号报文的ACK  
发送 10号数据报  
收到了 5号报文的ACK  
发送 11号数据报  
收到了 6号报文的ACK  
发送 12号数据报
```

如上图所示，当传输文件数据时，首先发送端会连续发送 2-9 号共 8 个数据报，也就是窗口大小，随后等待接收端传来 ack 信息，实现窗口滑动。

当收到 2 号数据报的 ack 时，窗口向后滑动，此时发送 10 号数据报。以此类推就实现了滑动窗口。

然后再来观察一下丢包时的情况：

```
发送 97号数据报
收到了 90号报文的ACK发送 98号数据报
收到了 91号报文的ACK发送 99号数据报
收到了 92号报文的ACK
收到了 93号报文的ACK
发送 100收到了 94号报文的ACK号数据报
收到了 95号报文的ACK
发送 101号数据报
收到了 96号报文的ACK发送 102号数据报
发送 103号数据报
发送 104号数据报
发送 105号数据报
收到了 97号报文的ACK发送 106号数据报
非正确回复, 编号: 98
非正确回复, 编号: 发送 107号数据报
98
发送 108号数据报非正确回复, 编号: 98
非正确回复, 编号: 98
发送 109号数据报非正确回复, 编号: 98
发送 110号数据报
发送 111号数据报
发送 112号数据报
发送 113号数据报
非正确回复, 编号: 98
非正确回复, 编号: 98
非正确回复, 编号: 98
非正确回复, 编号: 98
非正确回复, 编号: 98
非正确回复, 编号: 98
非正确回复, 编号: 98
超时啦98 98
发送 99号数据报
发送 100号数据报
发送 101号数据报
发送 102号数据报
发送 103号数据报
发送 104号数据报
发送 105号数据报
发送 106号数据报
收到了 99号报文的ACK
收到了 100号报文的ACK
```

如上图所示, 在发送 99 号数据报时出现了丢包现象, 接收端没有收到 99 号数据报, 所以回复的是上一条 ACK 报文。

接收端收到抛弃重复 ACK, 99 号数据报超时后, 立刻重传窗口内的已发送未确认数据报。