南開大學
Nankai University

计算机学院

计算机网络实验报告

基于 UDP 服务设计可靠传输协议 (3-2)

姓名：杨馨仪

学号：2011440

专业：计算机科学与技术

2022 年 12 月 3 日

目录

1 实验要求

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。

- 多个序列号
- 发送缓冲区、接受缓冲区
- 滑动窗口：Go Back N
- 有必要日志输出（须显示传输过程中发送端、接收端的窗口具体情况）

2 协议设计

上次实验中我使用数据报套接字实现了面向连接的可靠数据传输。在建立连接的过程中实现了类似于 TCP 的三次握手，实现了从客户端到服务器端以二进制单向传输文件，实现了对消息类型、校验和、序列号等成员的差错检验，实现了 rdt3.0 协议的确认重传功能，采用了停等机制的流量控制方法，断开连接实现了类似于 TCP 的四次挥手功能。

本次实验在上次实验的基础上，实现了基于滑动窗口的流量控制机制，采用累计确认和超时重传，差错重传，实现了文件的传输。

详细设计方式将在下一部分进行具体介绍。

3 模块功能介绍

3.1 UDP 协议的基本框架

在这里我们以客户端为例，介绍 UDP 协议的基本框架。在此次客户端的设计中我才用了双线程的模式，其中一个线程负责接收报文，而主线程负责发送报文，大致结构如下代码所示：

```
DWORD WINAPI recvMsgThread(LPVOID Iparam)//接收消息的线程
{
  SOCKET sockClient = *(SOCKET*)Iparam;
  Package p1;
  int addrlen = sizeof(SOCKADDR);
  while (1)
  {
    if (recvfrom(sockClient, (char*)&p1, sizeof(p1), 0, (SOCKADDR*)&addrSrv,
        &addrlen) > 0)
    {
      ...
    }
  }
  return 0L;
}

void main()
{
```

```
18    WSADATA wsaData;
19    WSAStartup(MAKEWORD(2, 2), &wsaData);
20
21    SOCKET sockClient = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
22
23    SOCKADDR_IN addrSrv = { 0 };
24    addrSrv.sin_family = AF_INET;//用AF_INET表示TCP/IP协议。
25    addrSrv.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");//设置为本地回环地址
26    addrSrv.sin_port = htons(4001);
27
28    SOCKADDR_IN addrClient;
29    int len = sizeof(SOCKADDR);
30
31    HandShake(sockClient, addrSrv);
32
33    hMutex = CreateMutex(NULL, FALSE, L"screen");
34    CloseHandle(CreateThread(NULL, 0, recvMsgThread, (LPVOID)&sockClient, 0, 0));
35
36    while (1)
37    {
38        ...// 以二进制方式打开文件
39      SendMsg(data, sockClient, addrSrv, dataLen, Buf[0]);
40    }
41    WaveHand(sockClient, addrSrv) == true
42    closesocket(sockClient);
43    WSACleanup();
44  }
```

由上述代码我们可以分析出：首先在主线程中，客户端（服务器端）开始工作，初始化 Socket DLL，协商使用的 Socket 版本，创建一个 socket，并绑定到 UDP 传输层服务。然后先对服务器端的地址信息（包括 IP 地址与端口号）进行初始化，服务器端使用 bind 函数将本地地址绑定到服务器 socket 上。接着进行三次握手操作并创建线程，之后进入循环，对于用户输入的想要传输给服务器的文件进行传输。然后在用户的指示下，结束数据传输，主动断开连接，进行四次挥手操作。最后关闭服务器 socket，并且释放 socket DLL 资源。

在第二线程中，主要负责对从服务器端接收到的消息进行处理。

服务器端的结构与此类似，只不过没有第二线程，不是读取文件而是存入文件上有所区别。

3.2 数据报格式介绍

```
1  struct HeadMsg {
2    u_long dataLen;
3    u_short len;
4    u_short checkSum;
5    unsigned char type;
6    unsigned char seq;
7    unsigned char fileNum;
8    unsigned char fileTyp;
```

```
};

struct Package {
  HeadMsg hm;
  char data[8000];
};
```

本次实验中我定义了两个结构体，分别为 HeadMsg 用于存放首部，Package 用于存放整个数据报。

HeadMsg 中有七个成员：

- dataLen，4 个字节，用于存放整个文件的长度
- len，2 个字节，用于存放本数据报的数据部分长度（数据不能超过 8191 字节）
- checkSum，2 个字节，用于存放校验和
- type，1 个字节，用于存放标志位（包括 SYN，SYN_ACK，ACK，FIN_ACK，PSH，NAK 等）
- seq，1 个字节，用来存放序列号（0-255）
- fileNum，1 个字节，用来存放此次发送的文件编号
- fileTyp，1 个字节，用于存放文件类型

Package 有两个成员：

- hm，16 个字节，用于存放数据报头部
- data，8000 个字节，用于存放数据部分

3.3 数据报发送

本次实验选择实现 GBN，即实现重点为累计确认，超时重传，滑动窗口等。

在客户端与服务器端实现三次握手建立连接之后，客户端开始出传输文件，用序列号 seq 来标记传输的报文，server 回复 ACK，如果客户端收到服务器端发来的 ACK 的序列号 seq 与自己待确认的 seq 相等，则表示已经收到，窗口根据 ack 按照相应的大小向前滑动。使用队列记录每个已经发出并且没有收到回复报文 ack 的发送，判断队列中第一个记录的时间是否超时，超时则清空队列，并将窗口中的内容进行重传操作，具体实现如以下几个部分所示。

3.3.1 读取文件

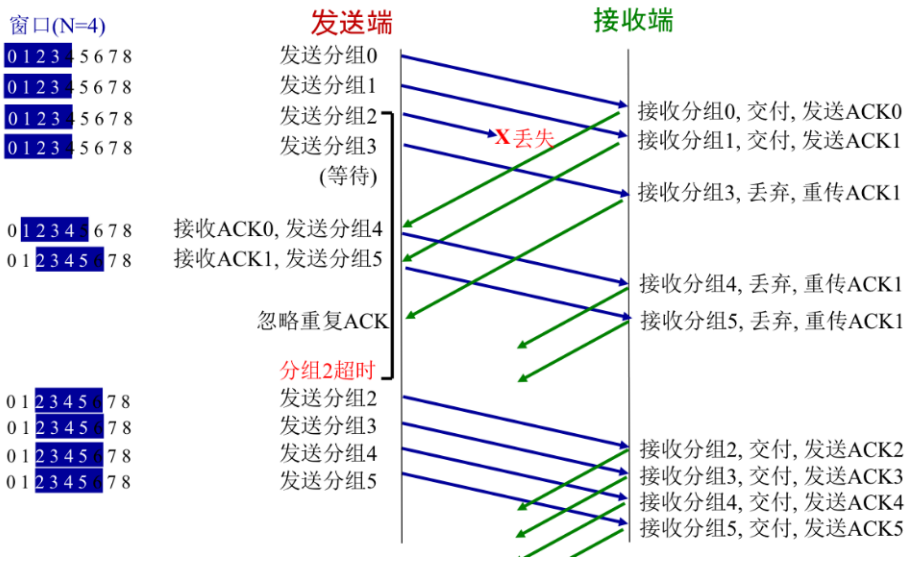
```
char Buf[BUFFER_SIZE] = {};
cin.getline(Buf, BUFFER_SIZE);
char* data;
if (strcmp(Buf, "1") == 0 || strcmp(Buf, "2") == 0 || strcmp(Buf, "3") == 0 ||
    strcmp(Buf, "4") == 0)
{
  char file[100] = "..\\test\\";
  if (Buf[0] == '1' || Buf[0] == '2' || Buf[0] == '3')
    sprintf(file, "%s%c.jpg", file, Buf[0]);
  else
    sprintf(file, "%s%s", file, "helloworld.txt");
  ifstream in(file, ifstream::in | ios::binary);
  int dataLen = 0;
  if (!in)
  {
    printf("%s [ INFO ] Client: can't open the file! Please retry\n", timei());
    continue;
  }
  // 文件读取到data
  BYTE t = in.get();
  char* data = new char[100000000];
  memset(data, 0, sizeof(data));
  while (in)
  {
    data[dataLen++] = t;
    t = in.get();
  }
  in.close();
  printf("read over\n");
```

首先根据交互获得用户想要发送的文件，然后将文件打开，将数据读取到 data，并存入文件数据的总长度，再将文件关闭。如此完成文件读取的过程。

3.3.2 发送信息与超时重传

对于发送消息的过程：我们首先计算该文件应当被切分为多少个数据报，然后初始化 head 和 tail 两个变量用来标记当前窗口大小，head 用来指向下一个待确认的数据报，tail 指向已经发送的数据报。然后我们循环将这些切分好的数据报发送给服务器端，注意在这个过程中，我们需要判断当前发送出去的数据报个数是否超过了窗口的大小 WINDOW_SIZE 以及是否已经发送结束。如果满足发送的条件，我们为每一个数据报设置头部信息，其中包括序列号，校验和，标志位等等。然后发送信息，把 tail+1 代表又发出了一个数据报，并且将刚刚发出的数据报的序列号与发送时间记录到队列之中。

如果我们发现当前的时间减去队列中第一个数据报的发送时间已经大于等待时间 WAIT_TIME 了，那么我们重新将 tail 赋值为 head-1，代表需要将窗口内的内容进行重传。



接下来来看客户端具体的实现。

```
bool SendMsg(char* data, SOCKET sockClient, SOCKADDR_IN addrSrv, int dataLen, char
    fileNum)
{
  int pcknum = ceil(dataLen / 8000.0);
  head = 0, tail = -1; //head为等待确认的pkg tail为已经发送的最后一个pkg
  while (head <= pcknum - 1)
  {
    WaitForSingleObject(hMutex, INFINITE);
    if (tail - head +1 < WINDOW_SIZE && tail != pcknum-1)
        //如果没超过窗口大小，且没发到结尾
    {
      // 设置信息头
      Package p;
      p.hm.dataLen = dataLen;
      p.hm.seq = (tail + 1) % 256;
      p.hm.type = PSH;
      p.hm.checkSum = 0;
      p.hm.fileNum = fileNum;
      if (fileNum == '1' || fileNum == '2' || fileNum == '3')
        p.hm.fileTyp = JPG;
      else
        p.hm.fileTyp = TXT;
      if (tail != pcknum - 2)
        p.hm.len = 8000;
      else
        p.hm.len = dataLen % 8000;
      // data存放的是读入的二进制数据，sentLen是已发送的长度，作为分批次发送的偏移量
      memcpy(p.data, data + (tail+1) * 8000, p.hm.len); //把本个包的数据存进去
      // 计算校验和
      p.hm.checkSum = checkSumVerify((u_short*)&p, sizeof(p));
      SendPkg(p, sockClient, addrSrv);
```

```
30        tail++;
31        printf("%s [ INFO ] Client: WINDOW HEAD: %d, WINDOW TAIL: %d, TOTAL NUM: %d\n",
              timei(), head, tail, pcknum);
32        timer_list.push(make_pair(tail+1, clock()));
33      }
34      ReleaseMutex(hMutex);
35
36
37      //判断发送时间
38      WaitForSingleObject(hMutex, INFINITE);
39      if (timer_list.size() != 0)
40      {
41        //cout << "目前延迟时间" << clock() - timer_list.front().second << endl;
42        if ((clock() - timer_list.front().second) > WAIT_TIME)
43        {
44          tail = head - 1;
45          printf("%s [ INFO ] Client: WINDOW HEAD: %d, WINDOW TAIL: %d, TOTAL NUM:
                %d\n", timei(), head, tail, pcknum);
46          while (timer_list.size()) timer_list.pop();
47        }
48      }
49      ReleaseMutex(hMutex);
50    }
51    return true;
52  }
```

在上面 SendMsg 函数中调用了 SendPkg 函数，接下来是对这个函数的具体介绍：

其实对于发送数据这个步骤来说，这个函数只是完成了 sendto 函数的工作。但该函数在三次握手与四次挥手中也会用到，里面大部分内容与握手挥手相关，在此统一做出讲解，后续不再赘述。

客户端 SendPkg 函数

```
1   bool SendPkg(Package p, SOCKET sockClient, SOCKADDR_IN addrSrv)
2   {
3     char Type[10];
4     switch (p.hm.type) {
5     case 1: strcpy(Type, "SYN"); break;
6     case 4: strcpy(Type, "ACK"); break;
7     case 8: strcpy(Type, "FIN_ACK"); break;
8     case 16:strcpy(Type, "PSH"); break;
9     }
10    // 发送消息
11    while (sendto(sockClient, (char*)&p, sizeof(p), 0, (SOCKADDR*)&addrSrv,
          sizeof(SOCKADDR)) == -1)
12    {
13      printf("%s [ ERR  ] Client: send [%s] ERROR:%s Seq=%d\n", timei(), Type,
            strerror(errno), p.hm.seq);
14    }
15    printf("%s [ INFO ] Client: [%s] Seq=%d\n", timei(), Type, p.hm.seq);
```

```
16
17    if (!strcmp(Type, "ACK") || !strcmp(Type, "PSH"))
18      return true;
19    // 开始计时
20    clock_t start = clock();
21    // 等待接收消息
22    Package p1;
23    int addrlen = sizeof(SOCKADDR);
24    while (true) {
25      if (recvfrom(sockClient, (char*)&p1, sizeof(p1), 0, (SOCKADDR*)&addrSrv,
          &addrlen) > 0 && clock() - start <= WAIT_TIME) {
26        // 收到消息需要验证消息类型、序列号和校验和
27        u_short ckSum = checkSumVerify((u_short*)&p1, sizeof(p1));
28        if ((p1.hm.type == SYN_ACK && !strcmp(Type, "SYN")) && p1.hm.seq == seq &&
            ckSum == 0)
29        {
30          printf("%s [ GET  ] Client: receive [SYN, ACK] from Server\n", timei());
31          return true;
32        }
33        else if ((p1.hm.type == ACK && (!strcmp(Type, "FIN_ACK") || !strcmp(Type,
            "PSH"))) && p1.hm.seq == seq && ckSum == 0)
34        {
35          printf("%s [ GET  ] Client: receive [ACK] from Server\n", timei());
36          return true;
37        }
38        else {
39          SendPkg(p, sockClient, addrSrv);
40          return true;
41        }
42      }
43      else {
44        SendPkg(p, sockClient, addrSrv);
45        return true;
46      }
47    }
48  }
```

这里以客户端的 SendPkg 为例。首先这个函数的参数包括打包好的 Package p, 客户端的 socket 以及服务器端的地址。首先我们获取数据报中首部的标志位信息，并将其以字符串的形式存放在 Type 中。然后发送这个数据报给服务器端。如果这个数据报的标志位只有 ACK 置 1 或只有 PSH 置一，那么直接返回；否则开始计时，使用 recvfrom 函数接收消息。如果当前数据报标志位为 SYN，并且接收到了 SYN_ACK 的数据报，在确认序列号与校验和无误后，直接返回；如果当前数据报标志位为 FIN_ACK，并且接收到了 ACK 的数据报，在确认序列号与校验和无误后，直接返回；如果不是以上三种情况，若序列号或校验和错误则触发差错重传；如果超时没有接收到数据报，则超时重传。

以上是它的整个功能，部分还要在三次握手和四次挥手中用到的，在此介绍过之后，之后不再赘述。

3.3.3 服务器接收消息

对于服务器，他在完成一系列初始化工作后，便等待由客户端发来的数据报，我们采用一个死循环来接受被切分的数据报，接收到数据报之后，我们首先确认他的类型是否是 PSH，并且确认其校验和无误，然后在判断传递过来的这个数据报的序号是否是我们期待的想要收到的那个序列号 lastAck，如果是的话，我们新建一个数据报，将类型设置为 ACK，将序列号设置为已经确认收到的序列号即 lastAck, 并把 lastAck+1(不要忘记要模 256)，设置它的校验和，并把它发送给客户端；如果不是我们期待收到的数据报，那么我们也创建一个数据报，但把他的序列号设置为 lastAck-1，然后发送给客户端。如果接收到的数据长度等于文件总长度了，那么代表文件接收成功。

```
RecvData RecvMsg(SOCKET sockSrv, SOCKADDR_IN addrClient)
{
  Package p1;
  int addrlen = sizeof(SOCKADDR);
  int totalLen = 0;
  RecvData rd;
  int lastAck = 0;
  rd.data = new char[100000000];
  // 等待接收消息
  while (true) {
    // 收到消息需要验证校验和及序列号
    if (recvfrom(sockSrv, (char*)&p1, sizeof(p1), 0, (SOCKADDR*)&addrClient,
        &addrlen) > 0)
    {
      if (p1.hm.type == FIN_ACK)
      {
        cout <<
            "----------------------------DISCONNECTION--------------------------" <<
            endl;
        rd.fileNum = '0';
        WaveHand(sockSrv, addrClient, p1);
        break;
      }

      Package p2;
      p2.hm.fileTyp = p2.hm.fileNum = p2.hm.checkSum = p2.hm.dataLen = p2.hm.len = 0;

      int ck = !checkSumVerify((u_short*)&p1, sizeof(p1));
      if (p1.hm.type == PSH && ck == 1)
      {
        if (p1.hm.seq != lastAck)
        {
          p2.hm.type = ACK;
          p2.hm.seq = lastAck -1;
          p2.hm.checkSum = checkSumVerify((u_short*)&p2, sizeof(p2));
          while (sendto(sockSrv, (char*)&p2, sizeof(p2), 0, (SOCKADDR*)&addrClient,
              sizeof(SOCKADDR)) == -1)
            printf("%s [ ERR  ] Server: send [ACK] ERROR:%s Seq=%d\n", timei(),
                strerror(errno), p2.hm.seq);
```

```
35              printf("%s [ INFO ] Server: [ACK] Seq=%d\n", timei(), p2.hm.seq);
36              continue;
37          }
38          p2.hm.type = ACK;
39          p2.hm.seq = lastAck;
40          lastAck = (lastAck + 1) % 256;
41          p2.hm.checkSum = checkSumVerify((u_short*)&p2, sizeof(p2));
42          while (sendto(sockSrv, (char*)&p2, sizeof(p2), 0, (SOCKADDR*)&addrClient,
                sizeof(SOCKADDR)) == -1)
43              printf("%s [ ERR  ] Server: send [ACK] ERROR:%s Seq=%d\n", timei(),
                    strerror(errno), p2.hm.seq);
44          printf("%s [ INFO ] Server: [ACK] Seq=%d\n", timei(), p2.hm.seq);
45          memcpy(rd.data + totalLen, (char*)&p1 + sizeof(p1.hm), p1.hm.len);
46          totalLen += (int)p1.hm.len;
47          rd.dataLen = p1.hm.dataLen;
48          rd.fileNum = p1.hm.fileNum;
49          if (totalLen == p1.hm.dataLen)
50          {
51              Sleep(3000);
52              break;
53          }
54      }
55    }
56  }
57  return rd;
58 }
```

3.3.4 客户端接收消息

那么客户端对于由服务器端发来的消息需要进行确认，同样在接收到数据报后要先对数据报类型与校验和进行检验，确认无误后，再检查它的序列号。这时有两种情况，可以证明无误。

- 接收到的序列号比 head 模 256 要大，这是代表，接收到的序列号之前的那些数据报都已经被确认。
- 由于模运算的关系，接收到的序列号比 head 模 256 要小，但在整个文件中的位置是在 head 之后的，这种情况我们要特别处理 head 将要指向的内容。

对于两种无误的情况，我们要将队列中的已经被确认的数据报的发送时间 pop 出来，并且将 head 指向接下来待确认的数据报处。

如果我们发现接收到的类型或校验和有误，这时我们将 tail 赋为 head-1，将窗口内容进行重传处理。

客户端接收线程

```
1 DWORD WINAPI recvMsgThread(LPVOID Iparam)//接收消息的线程
2 {
3   SOCKET sockClient = *(SOCKET*)Iparam;//获取客户端的SOCKET参数
4   Package p1;
```

```
5   int addrlen = sizeof(SOCKADDR);
6   while (1)
7   {
8     if (recvfrom(sockClient, (char*)&p1, sizeof(p1), 0, (SOCKADDR*)&addrSrv,
        &addrlen) > 0)
9     {
10      int ck = !checkSumVerify((u_short*)&p1, sizeof(p1));
11      WaitForSingleObject(hMutex, INFINITE);
12      if (p1.hm.type != ACK && ck != 1)  //类型错误或校验和有误
13      {
14        tail = head - 1;
15        printf("%s [ ERR  ] Client: receive wrong PKG from Server   Seq = %d\n",
            timei(), p1.hm.seq);
16        printf("%s [ INFO ] Client: WINDOW HEAD: %d, WINDOW TAIL: %d\n", timei(),
            head, tail);
17        continue;
18      }
19      else
20      {
21        int acknum = 0;
22        if (p1.hm.seq >= head % 256)
23        {
24          acknum = p1.hm.seq - head % 256 + 1;
25          head = head + acknum;
26          while (timer_list.size() != 0 && acknum != 0)
27          {
28            //到达最后一个时base=115
29            timer_list.pop();
30            acknum--;
31            //cout << "base=" << base << endl;
32          }
33          printf("%s [ GET  ] Client: receive [ACK] from Server   Seq = %d\n",
              timei(), p1.hm.seq);
34          printf("%s [ INFO ] Client: WINDOW HEAD: %d, WINDOW TAIL: %d\n", timei(),
              head, tail);
35        }
36        else if (head % 256 > 256 - WINDOW_SIZE && p1.hm.seq < WINDOW_SIZE - (256 -
            head % 256))
37        {
38          acknum = 256 - head % 256 + p1.hm.seq + 1;
39          head = head + acknum;
40          while (timer_list.size() != 0 && acknum != 0)
41          {
42            //到达最后一个时base=115
43            timer_list.pop();
44            acknum--;
45            //cout << "base=" << base << endl;
46          }
47          printf("%s [ GET  ] Client: receive [ACK] from Server   Seq = %d\n",
```

```
                timei(), p1.hm.seq);
48          printf("%s [ INFO ] Client: WINDOW HEAD: %d, WINDOW TAIL: %d\n", timei(),
                head, tail);
49        }
50      }
51      ReleaseMutex(hMutex);
52    }//释放互斥量锁
53    //cout << "收到报文的错误: " << GetLastError() << endl;
54  }
55  return 0L;
56 }
```

3.4 建立连接——三次握手

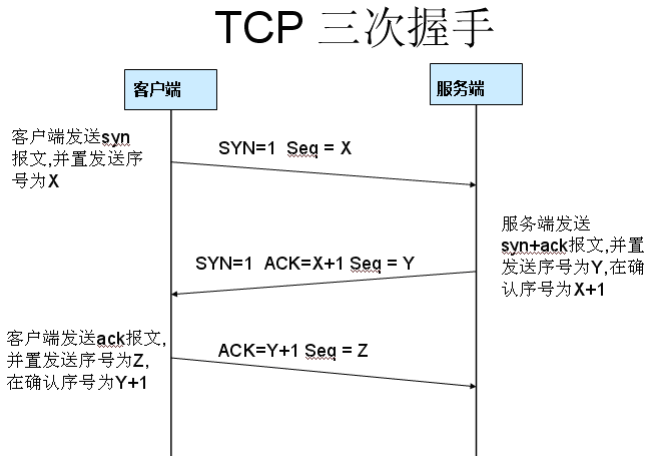
三次握手连接建立过程:

Step1: Client 将标志位 SYN 置为 1，产生 seq 值为 J，并将该数据包发送给 Server，Client 进入 SYN_SENT 状态，等待 Server 确认, 这是第一次握手。

Step2: Server 收到数据包后由标志位 SYN=1 知道 Client 请求建立连接，Server 将标志位 SYN 和 ACK 都置为 1，产生序列号 seq 为 J，代表确认收到客户端的建立连接请求。并将该数据包发送给 Client 以确认连接请求，Server 进入 SYN_RCVD 状态, 这是第二次握手。

Step3: Client 收到确认后，检查 seq 是否为 J，ACK,SYN 标志位是否为 1，如果正确则将标志位 ACK 置为 1，并将该数据包发送给 Server，Server 检查 ACK 是否为 1，如果正确则连接建立成功，Client 和 Server 进入 ESTABLISHED 状态，完成三次握手，随后 Client 与 Server 之间可以开始传输数据了。

TCP 连接建立，开始通讯。



然后我们来看一看客户端与服务器端的握手函数:

客户端 HandShake 函数

```
1  bool HandShake(SOCKET sockClient, SOCKADDR_IN addrSrv)
2  {
3    char sendBuf[BUFFER_SIZE] = {};
4    cin.getline(sendBuf, BUFFER_SIZE);
```

```
  if (strcmp(sendBuf, "connect") != 0)
    return false;
  Package p1;
  p1.hm.type = SYN;
  p1.hm.seq = seq;
  p1.hm.checkSum = 0;
  p1.hm.checkSum = checkSumVerify((u_short*)&p1, sizeof(p1));
  int len = sizeof(SOCKADDR);
  SendPkg(p1, sockClient, addrSrv);
  seq = (seq + 1) % 256;
  p1.hm.type = ACK;
  p1.hm.seq = seq;
  p1.hm.checkSum = 0;
  p1.hm.checkSum = checkSumVerify((u_short*)&p1, sizeof(p1));

  if (sendto(sockClient, (char*)&p1, sizeof(p1), 0, (SOCKADDR*)&addrSrv,
      sizeof(SOCKADDR)) != -1)
  {
    printf("%s [ INFO ] Client: [ACK] Seq=%d\n", timei(), seq);
    seq = (seq + 1) % 256;
    return true;
  }
  else
  {
    printf("%s [ ERR  ] Client: send [ACK] ERROR\n", timei());
    return false;
  }
}
```

服务器端 HandShake 函数

```
bool HandShake(SOCKET sockSrv, SOCKADDR_IN addrClient)
{
  Package p2;
  int len = sizeof(SOCKADDR);
  while (true)
  {
    if (recvfrom(sockSrv, (char*)&p2, sizeof(p2), 0, (SOCKADDR*)&addrClient, &len) >
        0)
    {
      int ck = checkSumVerify((u_short*)&p2, sizeof(p2));
      if (p2.hm.type == SYN && ck == 0)
      {
        printf("%s [ GET  ] Server: receive [SYN] from Client\n", timei());
        Package p3;
        p3.hm.type = SYN_ACK;
        p3.hm.seq = (lastAck + 1) % 256;
        lastAck = (lastAck + 2) % 256;
        p3.hm.checkSum = 0;
```

```
            p3.hm.checkSum = checkSumVerify((u_short*)&p3, sizeof(p3));
            SendPkg(p3, sockSrv, addrClient);
            break;
        }
        else
        {
            printf("%s [ ERR  ] Server: receive [SYN] ERROR\n", timei());
            return false;
        }
    }
  }
  return true;
}
```

通过服务器端与客户端的多次交互，我们可以发现实现了开始所说的三次握手功能，并且其中也包含有差错检验与确认重传，超时重传的相关功能。

3.5 断开连接——四次挥手

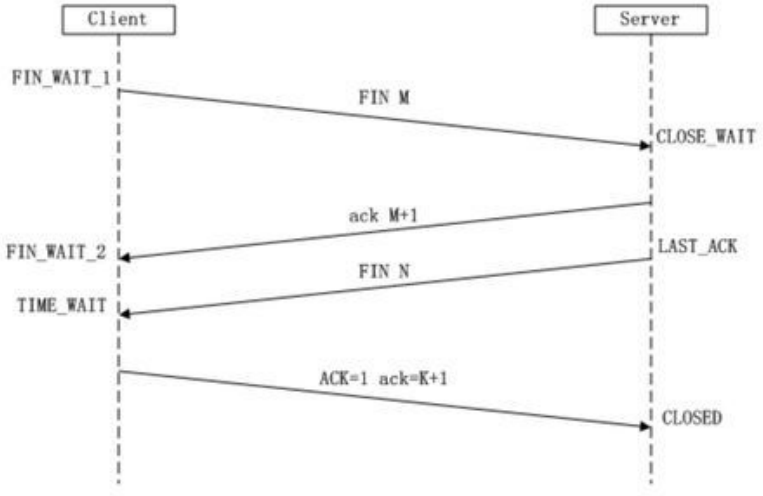
四次挥手即终止连接，就是指断开一个 TCP 连接时，需要客户端和服务端总共发送 4 个包以确认连接的断开。

Step1: Client 发送一个 FIN，用来关闭 Client 到 Server 的数据传送，Client 进入 FIN_WAIT_1 状态。

Step2: Server 收到 FIN 后，发送一个 ACK 给 Client，序号为收到序号，Server 进入 CLOSE_WAIT 状态。

Step3: Server 发送一个 FIN，用来关闭 Server 到 Client 的数据传送，Server 进入 LAST_ACK 状态。

Step4: Client 收到 FIN 后，Client 进入 TIME_WAIT 状态，接着发送一个 ACK 给 Server，序号为收到序号，Server 进入 CLOSED 状态，完成四次挥手。



实验中我们通过 WaveHand 函数来实现四次挥手功能，在其中依旧调用了 SendMsg 函数。

客户端 WaveHand 函数

```
bool WaveHand(SOCKET sockClient, SOCKADDR_IN addrSrv)
{
  Package p1;
  p1.hm.type = FIN_ACK;
  p1.hm.seq = seq;
  p1.hm.checkSum = 0;
  p1.hm.checkSum = checkSumVerify((u_short*)&p1, sizeof(p1));
  int len = sizeof(SOCKADDR);

  SendPkg(p1, sockClient, addrSrv);
  Package p4;

  while (true)
  {
    if (recvfrom(sockClient, (char*)&p4, sizeof(p4), 0, (SOCKADDR*)&addrSrv, &len) >
        0)
    {
      if (p4.hm.type == FIN_ACK)
      {
        printf("%s [ GET  ] Client: receive [FIN, ACK] from Server\n", timei());
        p4.hm.type = ACK;
        p4.hm.checkSum = 0;
        p4.hm.checkSum = checkSumVerify((u_short*)&p4, sizeof(p4));
        SendPkg(p4, sockClient, addrSrv);
        break;
      }
      else
      {
        printf("%s [ ERR  ] Server: receive [FIN, ACK] ERROR\n", timei());
        return false;
      }
    }
  }
  return true;
}
```

服务器端 WaveHand 函数

```
bool WaveHand(SOCKET sockSrv, SOCKADDR_IN addrClient, Package p2)
{
  int len = sizeof(SOCKADDR);
  u_short ckSum = checkSumVerify((u_short*)&p2, sizeof(p2));
  if (p2.hm.type == FIN_ACK && ckSum == 0)
  {
    printf("%s [ GET  ] Server: receive [FIN, ACK] from Client\n", timei());
    p2.hm.type = ACK;
    p2.hm.seq = (lastAck + 1) % 256;
    lastAck = (lastAck + 2) % 256;
```

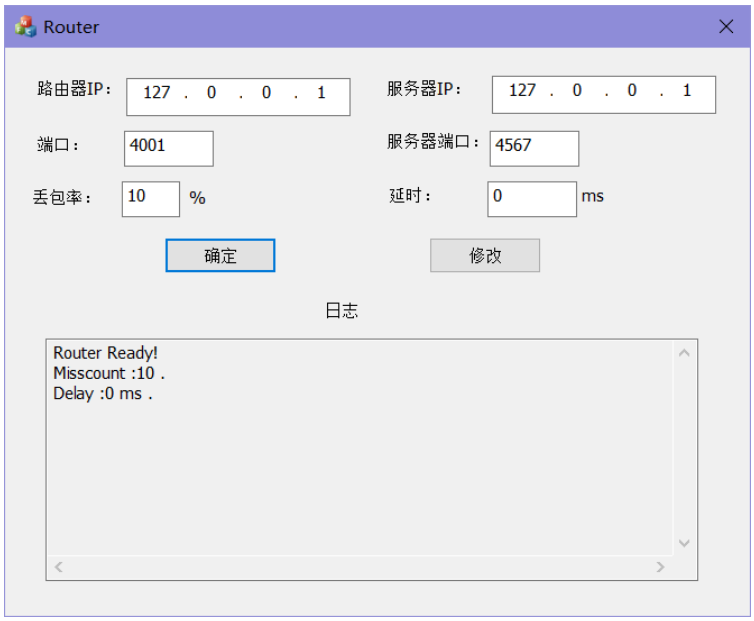
```
    p2.hm.checkSum = 0;
    p2.hm.checkSum = checkSumVerify((u_short*)&p2, sizeof(p2));
    SendPkg(p2, sockSrv, addrClient);
  }
  else
  {
    printf("%s [ ERR  ] Server: receive [FIN, ACK] ERROR\n", timei());
    return false;
  }

  Package p3;
  p3.hm.type = FIN_ACK;
  p3.hm.checkSum = 0;
  p3.hm.checkSum = checkSumVerify((u_short*)&p2, sizeof(p2));
  SendPkg(p3, sockSrv, addrClient);
  return true;
}
```

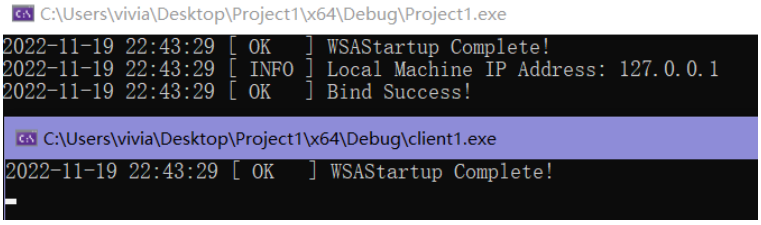
局部代码不能体现完整流程，但大致的框架如上所示，具体实现详见源码。

4 程序界面展示及运行说明

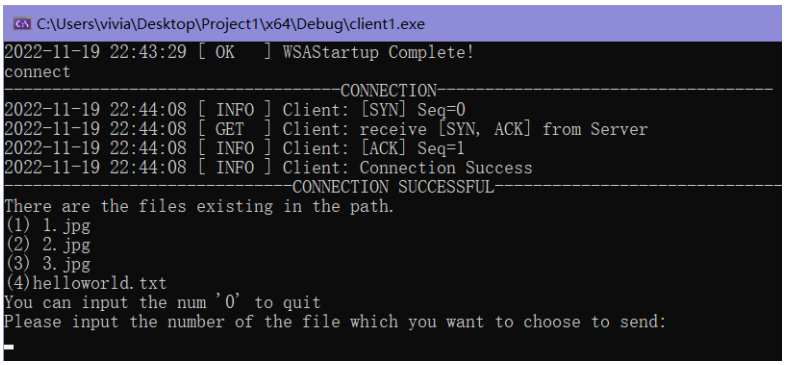
step1: 首先设置丢包率如下所示:



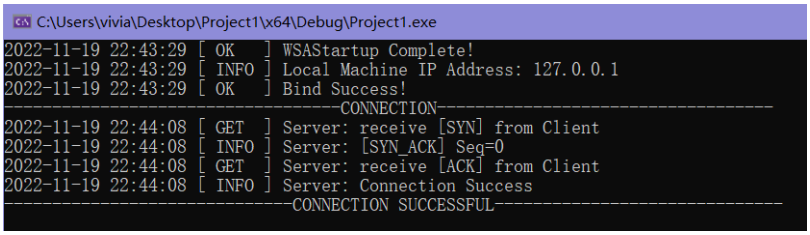
step2: 通过 visual studio 运行程序，如下是客户端与服务器端的一些初始打印信息



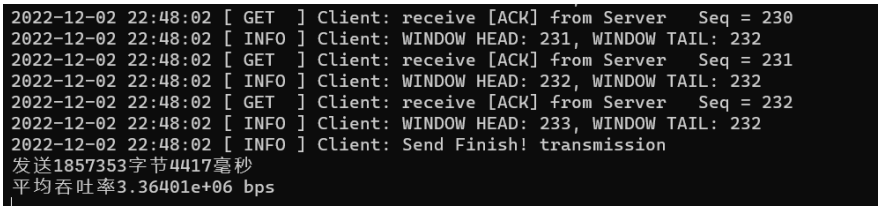
step3: 在客户端输入 connect，进行三次握手，建立连接，如下为客户端的截图



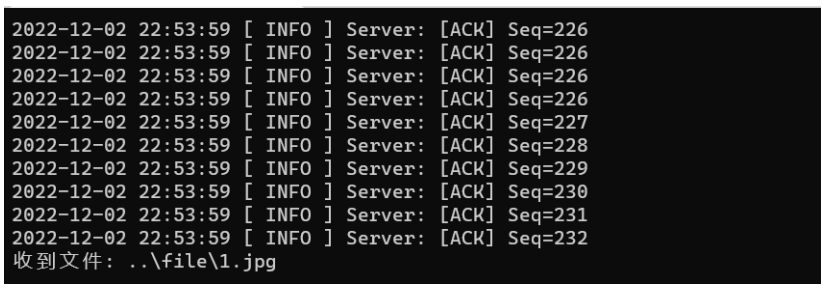
如下为服务器端的截图



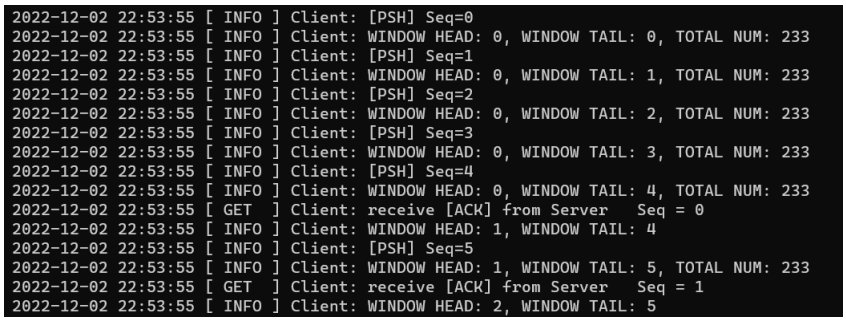
step4: 在客户端输入 1，开始对第一张图片进行可靠性传输；传输完毕显示发送时延，发送字节数以及平均吞吐量等信息



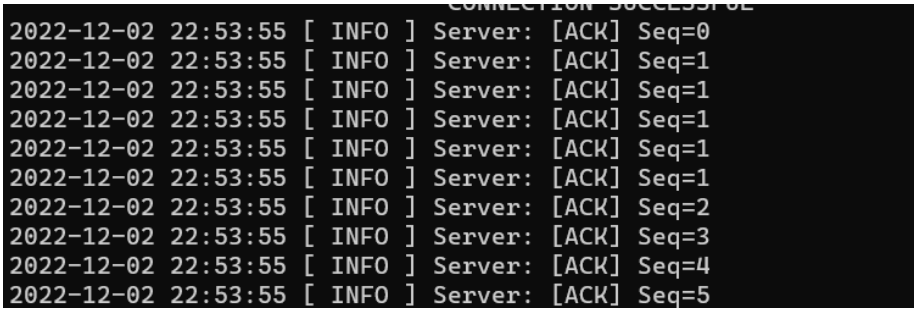
服务器端显示收到文件



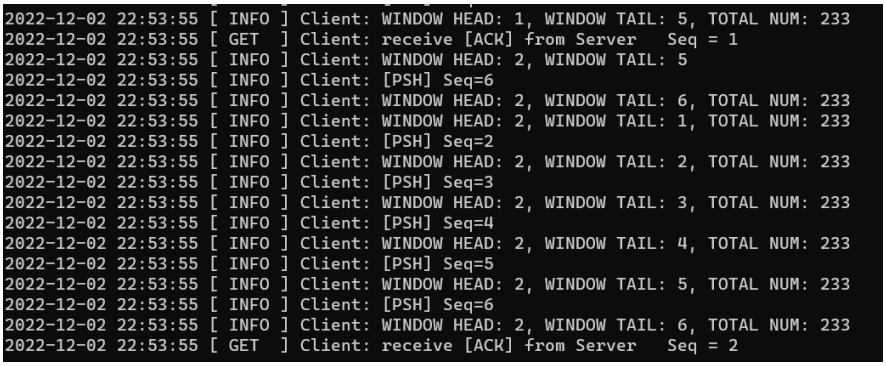
我们可以看到滑动窗口的变化，首先 tail 不断后移知道达到窗口大小 WINDOW_SIZE，然后知道接收到来自服务器端的 ACK 报文，我们将 head 后移一位，并发送下一个数据报，以此类推



直到发生了丢包事件，我们可以看到服务器端一直在重复发生序列号为 1 的报文，表明这个时候序列号为 2 的报文一直没有收到。

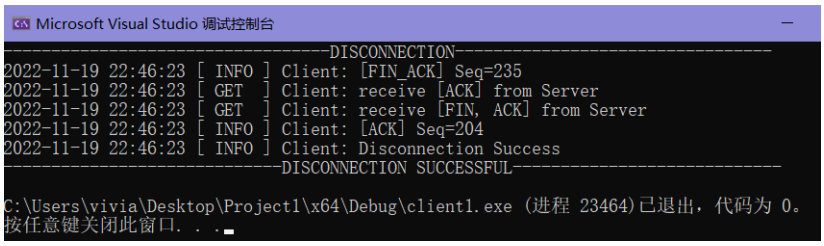


这时，我们看客户端当达到 WAIT_TIME 之后，将 tail 改为了 head-1，并且重新发送了从第二个包往后的包，然后终于得到了来自服务器端的 ACK 报文，处理丢包事件完毕。

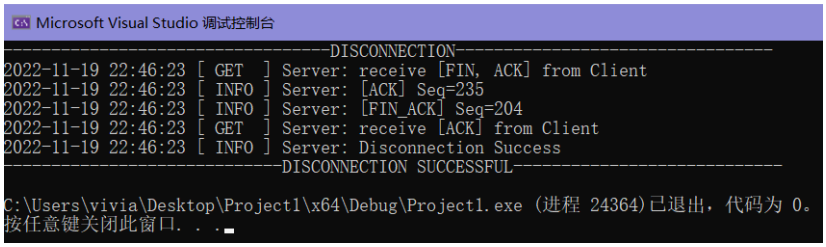


step5: 输入 5，提示没有第五个文件

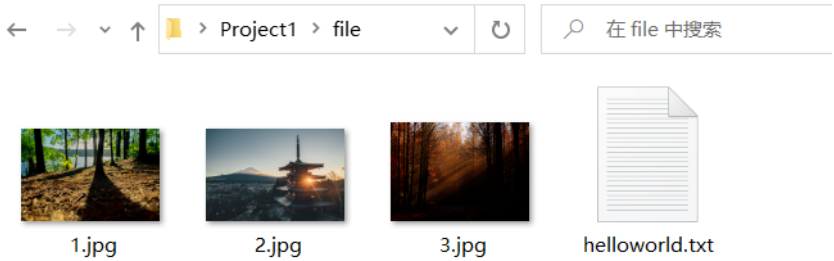
step6: 输入 0，进行四次挥手，断开连接，如下是客户端截图



如下是服务器端截图



step7: 打开文件夹，可以看到成功传输的图片与文本文件



5 实验反思

在本次实验中，我认为有以下几点发现问题并得到了解决：

- 由于 recvfrom 默认为堵塞传输，导致 recvfrom 在没有接收到数据报时一直不返回，无法计算 clock 的时间，使超时重传部分失效。后来我才用了队列的方式，记录每一个数据报的序列号与发送时间，如果接收到了队列首部的 ACK 报文则将它对应的发送时间 pop 出队列，然后再主线程中定时检查队列首部的发送时间是否超时，成功解决了 recvfrom 的堵塞问题。
- 由于加入时延后，可能客户端发送完最后一个数据报之后，以为丢包了会再超时重发，而服务器端其实已经接收到并且退出了接收函数，重置了 lastAck 的大小，但是这时候又接收到了来自客户端的上一个文件的冗余的数据报，这导致该程序加入时延后会再收到文件后还会在服务器端显示又收到了文件，但不影响传输文件的过程。