

计算机网络实验报告

Lab3-1 基于UDP服务设计可靠传输协议并编程实现

网络空间安全学院 物联网工程

2110951 梁晓储

代码已发布到github: https://github.com/WangshuXC/Computer_network

一、实验要求

1. 实现单向数据传输（一端发数据，一端返回确认）。
2. 对于每个任务要求给出详细的协议设计。
3. 完成给定测试文件的传输，显示传输时间和平均吞吐率。
4. 性能测试指标：吞吐率、延时，给出图形结果并进行分析。
5. 完成详细的实验报告（每个任务完成一份，主要包含自己的协议设计、实现方法、遇到的问题、实验结果，不要抄写太多的背景知识）。
6. 编写的程序应该结构清晰，具有较好的可读性。
7. 提交程序源码、可执行文件和实验报告。

二、协议设计和实验流程

Header协议设计

在send.cpp和receive.cpp中定义一个结构体 `HEADER`，其中包含如下信息。

```

1  struct HEADER
2  {
3      u_short sum = 0;
4      u_short datasize = 0;
5      unsigned char flag = 0;
6      unsigned char SEQ = 0;
7      HEADER() {
8          sum = 0;
9          datasize = 0;
10         flag = 0;
11         SEQ = 0;
12     }
13 };

```

每次发送packet需要修改header中的信息时修改该全局数组，再将其加入sendBuf。

- `sum`:16位的校验和
- `datasize`:所包含数据长度 16位
- `flag`:8位，使用后三位，排列是 FIN ACK SYN
- `SEQ`:8位，传输的序列号，0~255，超过后 mod

flag:

```

1  const unsigned char SYN = 0x1;
2  // 001— FIN = 0 ACK = 0 SYN = 1
3
4  const unsigned char ACK = 0x2;
5  // 010— FIN = 0 ACK = 1 SYN = 0
6
7  const unsigned char ACK_SYN = 0x3;
8  // 011— FIN = 0 ACK = 1 SYN = 1
9
10 const unsigned char FIN = 0x4;

```

```
11 // 100— FIN = 1 ACK = 0 SYN = 0
12
13 const unsigned char FIN_ACK = 0x5;
14 // 101— FIN = 1 ACK = 0 SYN = 1
15
16 const unsigned char OVER = 0x7;
17 // 结束标志 111— FIN = 1 ACK = 1 SYN = 1
```

实验流程

1. 三次握手建立连接：

- 发送端发送第一次握手（SYN）。
- 接收端接收第一次握手，并发送第二次握手（ACK）。
- 发送端接收第二次握手，并发送第三次握手（ACK_SYN）。

2. 文件数据传输：

- 发送端发送文件数据，每个数据包都有一个 **HEADER** 头部，包含校验和、数据长度、标志位等信息。
- 接收端接收文件数据，对每个数据包进行校验和验证，如果校验和正确，发送确认 ACK 给发送端，表示成功接收数据。

3. 四次挥手断开连接：

- 接收端发送第一次挥手（FIN）。
- 发送端接收第一次挥手，并发送第二次挥手（ACK）。
- 接收端接收第二次挥手，并发送第三次挥手（ACK）。
- 发送端接收第三次挥手，并发送第四次挥手（FIN_ACK）。

三、功能实现和代码分析

差错检测实现

差错校验是通过计算校验和来实现的。具体来说，校验和是在每个数据包的 **HEADER** 结构中计算得出的一个值，用于检测数据在传输过程中是否发生了错误或丢失。

在发送数据包之前，通过 **send_package** 函数将数据按照指定长度和顺序号组织成 **HEADER** 结构。然后，在计算校验和之前，先将校验和字段置为0。接下来，对 **HEADER** 结构中的所有成员变量（包括数据、数据长度、标志位和序列号）进行逐位异或（XOR）运算，最终得到校验和的值。将这个计算得到的校验和写入 **HEADER** 结构的校验和字段中。

当接收端收到数据包时，它会重新计算接收到的数据包的校验和。如果计算得到的校验和与接收到的数据包中的校验和相等，说明数据在传输过程中没有发生错误或丢失。如果两者不相等，则表示数据包可能存在差错，需要进行处理。

```
1  u_short checksum(u_short* mes, int size) {
2      int count = (size + 1) / 2;
3      u_short* buf = (u_short*)malloc(size + 1);
4      memset(buf, 0, size + 1);
5      memcpy(buf, mes, size);
6
7      u_long sum = 0;
8      while (count--) {
9          sum += *buf++;
10         if (sum & 0xffff0000) {
11             sum &= 0xffff;
12             sum++;
13         }
14     }
15     return ~(sum & 0xffff);
16 }
```

三次握手实现

发送端

```
1  int Connect(SOCKET& socketClient, SOCKADDR_IN& servAddr,
2             int& servAddrLen) // 三次握手建立连接
3  {
4      HEADER header;
5      char* Buffer = new char[sizeof(header)];
6
7      // 第一次握手
8      header.flag = SYN;
9      header.sum = 0; // 校验和置0
10     // 计算校验和
11     header.sum = checksum((u_short*)&header,
12                           sizeof(header));
13     // 将数据头放入buffer
14     memcpy(Buffer, &header, sizeof(header));
15     if (sendto(socketClient, Buffer, sizeof(header), 0,
16               (sockaddr*)&servAddr, servAddrLen) == -1)
17     {
18         return -1;
19     }
20     else
21     {
22         cout << "[\033[1;31mSend\033[0m] 成功发送第一次握手数
23         据" << endl;
24     }
25     clock_t start = clock(); // 记录发送第一次握手时间
26
27     // 设置socket为非阻塞状态
28     u_long mode = 1;
29     ioctlsocket(socketClient, FIONBIO, &mode);
30
31     // 第二次握手
32     while (recvfrom(socketClient, Buffer, sizeof(header), 0,
33                     (sockaddr*)&servAddr, &servAddrLen) ≤ 0)
```

```
29     {
30         // 超时需要重传
31         if (clock() - start > MAX_TIME) // 超时, 重新传输第一次
握手
32     {
33         cout << "[\033[1;33mInfo\033[0m] 第一次握手超时"
<< endl;
34         header.flag = SYN;
35         header.sum = 0;
36         // 校验和置0
37         header.sum = checksum((u_short*)&header,
sizeof(header)); // 计算校验和
38         memcpy(Buffer, &header, sizeof(header));
39         // 将数据头放入Buffer
40         sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrLen);
41         start = clock();
42         cout << "[\033[1;33mInfo\033[0m] 已经重传" <<
endl;
43     }
44 }
45
46 // 第二次握手, 收到来自接收端的ACK
47 // 进行校验和检验
48 memcpy(&header, Buffer, sizeof(header));
49 if (header.flag == ACK && checksum((u_short*)&header,
sizeof(header)) == 0)
50 {
51     cout << "[\033[1;32mReceive\033[0m] 接收到第二次握手数
据" << endl;
52 }
53 else
54 {
55     cout << "[\033[1;33mInfo\033[0m] 错误数据, 请重试" <<
endl;
56     return -1;
57 }
```

```

57     // 进行第三次握手
58     header.flag = ACK_SYN;
59     header.sum = 0;
60     header.sum = checksum((u_short*)&header,
sizeof(header)); // 计算校验和
61     if (sendto(socketClient, (char*)&header, sizeof(header),
0, (sockaddr*)&servAddr, servAddrlen) == -1)
62     {
63         return -1;
64     }
65     else
66     {
67         cout << "[\033[1;31mSend\033[0m] 成功发送第三次握手数
据" << endl;
68     }
69     cout << "[\033[1;33mInfo\033[0m] 服务器成功连接! 可以发送数
据" << endl;
70     return 1;
71 }

```

函数的作用是发送第一次握手数据，等待接收端的第二次握手数据，并发送第三次握手数据。

函数首先创建一个包含握手信息的数据头 `HEADER`。然后将数据头放入缓冲区 `Buffer` 中，并通过 `sendto` 函数发送给服务器端。如果发送失败，返回-1；否则打印成功发送的消息。

接下来，将发送端的套接字 `socketClient` 设置为非阻塞模式。然后进入一个循环，不断尝试接收接收端的第二次握手数据。如果接收超时（超过最大时间限制），则重新发送第一次握手数据。如果接收成功，将接收到的数据拷贝到数据头 `header` 中，检查其合法性。如果数据正确且校验和正确，打印成功接收的消息；否则返回-1。

最后，发送第三次握手数据给接收端，并打印成功发送的消息。函数返回1表示成功建立连接。

接收端

```
1  int Connect(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int&
   ClientAddrLen)
2  {
3      HEADER header;
4      char* Buffer = new char[sizeof(header)];
5
6      // 接收第一次握手信息
7      while (1)
8      {
9          // 通过绑定的socket传递、接收数据
10         if (recvfrom(sockServ, Buffer, sizeof(header), 0,
11         (sockaddr*)&ClientAddr, &ClientAddrLen) == -1)
12         {
13             return -1;
14         }
15         memcpy(&header, Buffer, sizeof(header));
16         if (header.flag == SYN &&
17         checkSum((u_short*)&header, sizeof(header)) == 0)
18         {
19             cout << "[\033[1;32mReceive\033[0m] 接收到第一次握
20             手数据 " << endl;
21             break;
22         }
23     }
24     // 发送第二次握手信息
25     header.flag = ACK;
26     header.sum = 0;
27     header.sum = checkSum((u_short*)&header,
28     sizeof(header));
29     memcpy(Buffer, &header, sizeof(header));
30
31     if (sendto(sockServ, Buffer, sizeof(header), 0,
32     (sockaddr*)&ClientAddr, ClientAddrLen) == -1)
33     {
34         return -1;
35     }
```



```

30     }
31     else
32     {
33         cout << "[\033[1;31mSend\033[0m] 成功发送第二次握手数据
" << endl;
34     }
35     clock_t start = clock(); // 记录第二次握手发送时间
36
37     // 接收第三次握手
38     while (recvfrom(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, &ClientAddrLen) ≤ 0)
39     {
40         // 超时重传
41         if (clock() - start > MAX_TIME)
42         {
43             cout << "[\033[1;33mInfo\033[0m] 第二次握手超时 "
<< endl;
44             header.flag = ACK;
45             header.sum = 0;
46             header.flag = checksum((u_short*)&header,
sizeof(header));
47             memcpy(Buffer, &header, sizeof(header));
48             if (sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, ClientAddrLen) = -1)
49             {
50                 return -1;
51             }
52             cout << "[\033[1;33mInfo\033[0m] 已经重传 " <<
endl;
53         }
54     }
55
56     // 解析收到的第三次握手的数据包
57     HEADER temp1;
58     memcpy(&temp1, Buffer, sizeof(header));
59
60     if (temp1.flag = ACK_SYN && checksum((u_short*)&temp1,
sizeof(temp1)) = 0)

```

```

61     {
62         cout << "[\033[1;32mReceive\033[0m] 接收到第三次握手数
        据" << endl;
63         cout << "[\033[1;33mInfo\033[0m] 成功连接" << endl;
64     }
65     else
66     {
67         cout << "[\033[1;33mInfo\033[0m] 错误数据, 请重试" <<
        endl;
68     }
69     return 1;
70 }

```

函数的作用是接收客户端的第一次握手数据，发送第二次握手数据，并等待接收发送端的第三次握手数据。

函数首先创建一个包含握手信息的数据头 `HEADER`。然后进入一个循环，通过 `recvfrom` 函数不断尝试接收发送端的第一次握手数据。如果接收成功，将接收到的数据拷贝到数据头 `header` 中，检查其合法性。如果数据正确且校验和正确，打印成功接收的消息；否则返回-1。

接下来，发送第二次握手数据给发送端。将数据头放入缓冲区 `Buffer` 中，并通过 `sendto` 函数发送给发送端。如果发送失败，返回-1；否则打印成功发送的消息。

然后，进入一个循环，不断尝试接收发送端的第三次握手数据。如果接收超时（超过最大时间限制），则重新发送第二次握手数据。如果接收成功，将接收到的数据拷贝到临时数据头 `temp1` 中，检查其合法性。如果数据正确且校验和正确，打印成功接收的消息，并打印成功连接的消息；否则打印错误数据的消息。

最后，函数返回1表示成功建立连接。

文件传输

在连接（握手）完毕后，会进入到文件传输的过程。

发送端

发送端有关文件传输的函数是send()和send_package(), 大致步骤如下：

1. 创建HEADER实例并填充相应信息，同时分配一个长度为 `MAXSIZE + sizeof(header)` 字节的缓冲区 `buffer`，用于存储整个数据包（包括头部和实际数据）。

```
1  HEADER header;  
2  char* buffer = new char[MAXSIZE + sizeof(header)];  
3  header.datasize = len;  
4  header.SEQ = unsigned char(order); // 序号  
5  memcpy(buffer, &header, sizeof(header)); // 将计算好校验和的  
   header重新赋值给buffer, 此时的buffer可以发送了  
6  memcpy(buffer + sizeof(header), message, len); // buffer为  
   header+message  
7  header.sum = checksum((u_short*)buffer, sizeof(header) +  
   len); // 计算校验和
```

2. 发送文件名和文件大小给接收端

```
1  // 发送  
2  sendto(socketClient, buffer, len + sizeof(header), 0,  
   (sockaddr*)&servAddr, servAddrLen);  
3  cout << "[\033[1;31mSend\033[0m] 发送了" << len << " 字节, "  
4       << " flag:" << int(header.flag)  
5       << " SEQ:" << int(header.SEQ) << " SUM:" <<  
   int(header.sum) << endl;  
6  clock_t start = clock(); // 记录发送时间
```

3. 发送文件数据（包含超时重传处理）

- 将套接口状态改为非阻塞模式，通过 `ioctlsocket` 函数将 `FIONBIO` 参数设为 1，这样 `recvfrom` 函数在没有数据可读时不会阻塞程序的执行，而是立即返回。
- 进入一个无限循环，尝试通过 `recvfrom` 函数接收对方发送的数据包。如果接收失败（返回值小于等于0），说明可能是由于超时导致的，此时会进行超时重传。
- 如果超时，则重新设置要发送的数据包（设置数据长度、序列号、标志位等），计算新的校验和，并通过 `sendto` 函数重新发送数据包。
- 如果接收成功，则将接收到的数据包解析到 `header` 结构体中，并检查确认信息的合法性。条件判断中检查了校验和是否为零、序列号是否匹配、标志位是否为 ACK。
- 如果确认信息合法，输出一些信息表示成功接收到确认信息，并退出循环，将套接口状态改回阻塞模式，否则继续下一轮循环。

```
1 //接收ack等信息
2 while (1)
3 {
4     u_long mode = 1;
5     ioctlsocket(socketClient, FIONBIO, &mode); //将套接口状
    态改为非阻塞
6
7
8     while (recvfrom(socketClient, buffer, MAXSIZE, 0,
        (sockaddr*)&servAddr, &servAddrLen) ≤ 0) {
9         //超时重传
10        if (clock() - start > MAX_TIME)
11        {
12            cout << "[\033[1;33mInfo\033[0m] 发送数据超时"
13            << endl;
14            header.datasize = len;
15            header.SEQ = u_char(order); //序列号
16            header.flag = u_char(0x0);
17            memcpy(buffer, &header, sizeof(header));
18            memcpy(buffer + sizeof(header), message,
19                sizeof(header) + len);
```

```

18         u_short check = checksum((u_short*)buffer,
sizeof(header) + len); //计算校验和
19         header.sum = check;
20         memcpy(buffer, &header, sizeof(header));
21         sendto(socketClient, buffer, len +
sizeof(header), 0, (sockaddr*)&servAddr, servAddrLen); //发
送
22         cout << "[\033[1;33mInfo\033[0m] 重新发送数据"
<< endl;
23         start = clock(); //记录发送时间
24     }
25     else break;
26 }
27     memcpy(&header, buffer, sizeof(header)); //缓冲区接收到信
息, 放入header, 此时header中是收到的数据
28     //u_short check = cksum((u_short*)&header,
sizeof(header));
29     if (checksum((u_short*)&header, sizeof(header)) == 0
&& header.SEQ == u_short(order) && header.flag == ACK)
30     {
31         cout << "[\033[1;32mReceive\033[0m] 已确认收到 -
Flag:" << int(header.flag)
32             << " SEQ:" << int(header.SEQ) << " SUM:"
<< int(header.sum) << endl;
33         break;
34     }
35     else
36     {
37         continue;
38     }
39 }
40 u_long mode = 0;
41 ioctlsocket(socketClient, FIONBIO, &mode); //改回阻塞模式

```

4. 发送结束信息

```

1 //发送结束信息
2 HEADER header;

```

```

3   char* Buffer = new char[sizeof(header)];
4   header.flag = OVER;
5   header.sum = checksum((u_short*)&header, sizeof(header));
6   memcpy(Buffer, &header, sizeof(header));
7   sendto(socketClient, Buffer, sizeof(header), 0,
          (sockaddr*)&servAddr, servAddrLen);
8   cout << "[\033[1;31mSend\033[0m] 发送OVER信号" << endl;
9   clock_t start = clock();
10  while (1)
11  {
12      //超时重传, 同上
13      // ...
14  }
15  u_long mode = 0;
16  ioctlsocket(socketClient, FIONBIO, &mode); //改回阻塞模式

```

接收端

接收端接受文件的函数是RecvMessage(), 大致步骤如下:

1. 接受文件大小, 分配缓冲区。

```

1   long int fileLength = 0; // 文件长度
2   HEADER header;
3   char* Buffer = new char[MAXSIZE + sizeof(header)];
4   int seq = 0;
5   int index = 0;

```

2. 接收文件数据

- 使用 `recvfrom` 函数接收数据包, 其中 `sockServ` 是套接字, `Buffer` 是接收数据的缓冲区, `sizeof(header) + MAXSIZE` 是期望接收的数据包的最大长度, `0` 是接收数据的标志, `ClientAddr` 和 `ClientAddrLen` 是保存发送方地址信息的参数。

- 判断接收到的数据包的标志位和校验和是否满足预期条件。如果接收到的标志位是结束标志 `OVER` 且校验和为零，则表示文件传输结束，退出循环。
- 如果接收到的标志位为零（可能是数据包），并且校验和满足预期条件，继续处理。
- 检查序列号是否正确。如果序列号不正确，说明接收到的数据包可能出错，需要重新返回确认信息（ACK）给发送方。在此处，它会重新发送上一个回复包的确认信息。
- 如果序列号正确，将接收到的数据包的信息存储到缓冲区 `message` 中，同时更新文件长度和序列号。
- 返回确认信息（ACK）给发送方，表示成功接收到数据包。
- 继续接收下一个数据包。

```
1     while (1)
2     {
3         int length = recvfrom(sockServ, Buffer,
4                                sizeof(header) + MAXSIZE, 0, (sockaddr *)&ClientAddr,
5                                &ClientAddrLen); // 接收报文长度
6         // cout << length << endl;
7         memcpy(&header, Buffer, sizeof(header));
8         // 判断是否是结束
9         if (header.flag == OVER && checkSum((u_short
10            *)&header, sizeof(header)) == 0)
11         {
12             cout << "[\033[1;33mInfo\033[0m] 文件传输结束"
13             << endl;
14             break;
15         }
16         if (header.flag == static_cast<unsigned char>(0)
17             && checkSum((u_short *)Buffer, length - sizeof(header)))
18         {
19             // 如果收到的序列号出错，则重新返回ACK，此时seq没有变
20             化
21             if (checkSum((u_short *)Buffer, length -
22                           sizeof(header)) == 0 && seq != int(header.SEQ))
```

```

16         {
17             //说明出了问题, 返回ACK
18             header.flag = ACK;
19             header.datasize = 0;
20             header.SEQ = (unsigned char)seq;
21             header.sum = checkSum((u_short*)&header,
sizeof(header));
22             memcpy(Buffer, &header, sizeof(header));
23             //重发该包的ACK
24             sendto(sockServ, Buffer, sizeof(header),
0, (sockaddr*)&ClientAddr, ClientAddrLen);
25             cout << "[\033[1;33mInfo\033[0m]重发上一个
回复包 - ACK:" << (int)header.SEQ << " SEQ:" <<
(int)header.SEQ << endl;
26             continue; // 丢弃该数据包
27         }
28         seq = int(header.SEQ);
29         if (seq > 255)
30         {
31             seq = seq - 256;
32         }
33
34         cout << "[\033[1;32mReceive\033[0m] 收到了 "
<< length - sizeof(header) << " 字节 - Flag:" <<
int(header.flag) << " SEQ : " << int(header.SEQ) << "
SUM:" << int(header.sum) << endl;
35         char *temp = new char[length -
sizeof(header)];
36         memcpy(temp, Buffer + sizeof(header), length
- sizeof(header));
37         // cout << "size" << sizeof(message) << endl;
38         memcpy(message + fileLength, temp, length -
sizeof(header));
39         fileLength = fileLength +
int(header.datasize);
40
41         // 返回ACK
42         header.flag = ACK;

```



```

43         header.datasize = 0;
44         header.SEQ = (unsigned char)seq;
45         header.sum = 0;
46         header.sum = checksum((u_short *)&header,
sizeof(header));
47         memcpy(Buffer, &header, sizeof(header));
48         // 返回客户端ACK, 代表成功收到了
49         sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr *)&ClientAddr, ClientAddrLen);
50         cout << "[\033[1;31mSend\033[0m] 回复客户端 -
flag:" << (int)header.flag << " SEQ:" << (int)header.SEQ
<< " SUM:" << int(header.sum) << endl;
51         seq++;
52         if (seq > 255)
53         {
54             seq = seq - 256;
55         }
56     }
57 }

```

3. 发送OVER信息并返回文件长度

```

1 //发送OVER信息
2 header.flag = OVER;
3 header.sum = 0;
4 header.sum = checksum((u_short*)&header, sizeof(header));
5 memcpy(Buffer, &header, sizeof(header));
6 if (sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr*)&ClientAddr, ClientAddrLen) == -1)
7 {
8     return -1;
9 }
10 return fileLength;

```

实现四次挥手

发送端

```
1  int disconnect(SOCKET &socketClient, SOCKADDR_IN &servAddr,
2  int &servAddrLen)
3  {
4      HEADER header;
5      char *Buffer = new char[sizeof(header)];
6
7      u_short sum;
8
9      // 进行第一次挥手
10     header.flag = FIN;
11     header.sum = 0;
12     // 校验和置0
13     header.sum = cksum((u_short *)&header, sizeof(header));
14     // 计算校验和
15     memcpy(Buffer, &header, sizeof(header));
16     // 将首部放入缓冲区
17     if (sendto(socketClient, Buffer, sizeof(header), 0,
18 (sockaddr *)&servAddr, servAddrLen) == -1)
19     {
20         return -1;
21     }
22     else
23     {
24         cout << "[\033[1;31mSend\033[0m] 成功发送第一次挥手数
25 据" << endl;
26     }
27     clock_t start = clock(); // 记录发送第一次挥手时间
28
29     u_long mode = 1;
30     ioctlsocket(socketClient, FIONBIO, &mode); // FIONBIO为
31 命令, 允许1/禁止0套接口s的非阻塞1/阻塞0模式。
32
33     // 接收第二次挥手
```

```

27     while (recvfrom(socketClient, Buffer, sizeof(header), 0,
28         (sockaddr *)&servAddr, &servAddrLen) ≤ 0)
29     {
30         // 超时, 重新传输第一次挥手
31         if (clock() - start > MAX_TIME)
32         {
33             cout << "[\033[1;33mInfo\033[0m] 第一次挥手超时"
34             << endl;
35             header.flag = FIN;
36             header.sum = 0;
37             // 校验和置0
38             header.sum = cksum((u_short *)&header,
39                 sizeof(header)); // 计算校验和
40             memcpy(Buffer, &header, sizeof(header));
41             // 将首部放入缓冲区
42             sendto(socketClient, Buffer, sizeof(header), 0,
43                 (sockaddr *)&servAddr, servAddrLen);
44             start = clock();
45             cout << "[\033[1;33mInfo\033[0m] 已重传第一次挥手数
46             据" << endl;
47         }
48     }
49
50     // 进行校验和检验
51     memcpy(&header, Buffer, sizeof(header));
52     if (header.flag == ACK && cksum((u_short *)&header,
53         sizeof(header)) == 0)
54     {
55         cout << "[\033[1;32mReceive\033[0m] 接收到第二次挥手数
56         据" << endl;
57     }
58     else
59     {
60         cout << "[\033[1;33mInfo\033[0m] 错误数据, 请重试" <<
61         endl;
62         return -1;
63     }
64

```

```

55     // 进行第三次挥手
56     header.flag = FIN_ACK;
57     header.sum = 0;
58     header.sum = cksum((u_short *)&header, sizeof(header));
    // 计算校验和
59     if (sendto(socketClient, (char *)&header,
sizeof(header), 0, (sockaddr *)&servAddr, servAddrLen) ==
-1)
60     {
61         return -1;
62     }
63     else
64     {
65         cout << "[\033[1;31mSend\033[0m] 成功发送第三次挥手数
据" << endl;
66     }
67     start = clock();
68     // 接收第四次挥手
69     while (recvfrom(socketClient, Buffer, sizeof(header), 0,
(sockaddr *)&servAddr, &servAddrLen) ≤ 0)
70     {
71         if (clock() - start > MAX_TIME) // 超时, 重新传输第三次
挥手
72         {
73             cout << "[\033[1;33mInfo\033[0m] 第三次挥手超时"
<< endl;
74             header.flag = FIN;
75             header.sum = 0;
            // 校验和置0
76             header.sum = cksum((u_short *)&header,
sizeof(header)); // 计算校验和
77             memcpy(Buffer, &header, sizeof(header));
            // 将首部放入缓冲区
78             sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr *)&servAddr, servAddrLen);
79             start = clock();
80             cout << "[\033[1;33mInfo\033[0m] 已重传第三次挥手数
据" << endl;

```

```

81         }
82     }
83     cout << "[\033[1;33mInfo\033[0m] 四次挥手结束, 连接断开! "
    << endl;
84     return 1;
85 }

```

函数首先构造并发送第一次挥手数据，然后设置socket为非阻塞模式，并在超时情况下重新传输第一次挥手数据。

接着进行校验和检验，如果通过则发送第三次挥手数据，并等待接收服务端的第四次挥手数据，同样在超时情况下重新传输第三次挥手数据。

最后输出四次挥手结束的信息，表示连接已经成功断开。

接收端

```

1  int disconnect(SOCKET &sockServ, SOCKADDR_IN &ClientAddr,
    int &ClientAddrLen)
2  {
3      HEADER header;
4      char *Buffer = new char[sizeof(header)];
5      while (1)
6      {
7          int length = recvfrom(sockServ, Buffer,
            sizeof(header) + MAXSIZE, 0, (sockaddr *)&ClientAddr,
            &ClientAddrLen); // 接收报文长度
8          memcpy(&header, Buffer, sizeof(header));
9          if (header.flag == FIN && checksum((u_short
            *)&header, sizeof(header)) == 0)
10         {
11             cout << "[\033[1;32mReceive\033[0m] 接收到第一次挥
                手数据 " << endl;
12             break;
13         }
14         else

```

```
15         {
16             return -1;
17         }
18     }
19     // 发送第二次挥手信息
20     header.flag = ACK;
21     header.sum = 0;
22     header.sum = checkSum((u_short *)&header,
sizeof(header));
23     memcpy(Buffer, &header, sizeof(header));
24     if (sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr *)&ClientAddr, ClientAddrLen) == -1)
25     {
26         cout << "[\033[1;33mInfo\033[0m] 发送第二次挥手失败" <<
endl;
27         return -1;
28     }
29     else
30     {
31         cout << "[\033[1;33mInfo\033[0m] 成功发送第二次挥手数
据" << endl;
32     }
33     clock_t start = clock(); // 记录第二次挥手发送时间
34
35     // 接收第三次挥手
36     while (recvfrom(sockServ, Buffer, sizeof(header), 0,
(sockaddr *)&ClientAddr, &ClientAddrLen) ≤ 0)
37     {
38         // 发送第二次挥手等待第三次挥手过程中超时, 重传第二次挥手
39         if (clock() - start > MAX_TIME)
40         {
41             cout << "[\033[1;33mInfo\033[0m] 第二次挥手超时 "
<< endl;
42             header.flag = ACK;
43             header.sum = 0;
44             header.sum = checkSum((u_short *)&header,
sizeof(header));
45             memcpy(Buffer, &header, sizeof(header));
```

```
46         if (sendto(sockServ, Buffer, sizeof(header), 0,
47         (sockaddr *)&ClientAddr, ClientAddrLen) == -1)
48             {
49                 return -1;
50             }
51             cout << "[\033[1;33mInfo\033[0m] 已重传第二次挥手数
据 " << endl;
52         }
53         // 解析收到的第三次挥手
54         HEADER temp1;
55         memcpy(&temp1, Buffer, sizeof(header));
56         if (temp1.flag == FIN_ACK && checksum((u_short *)&temp1,
sizeof(temp1) == 0))
57         {
58             cout << "[\033[1;33mInfo\033[0m] 接收到第三次挥手数据 "
<< endl;
59         }
60         else
61         {
62             cout << "[\033[1;33mInfo\033[0m] 错误数据, 请重试" <<
endl;
63             return -1;
64         }
65
66         // 发送第四次挥手信息
67         header.flag = FIN_ACK;
68         header.sum = 0;
69         header.sum = checksum((u_short *)&header,
sizeof(header));
70         memcpy(Buffer, &header, sizeof(header));
71         if (sendto(sockServ, Buffer, sizeof(header), 0,
(sockaddr *)&ClientAddr, ClientAddrLen) == -1)
72         {
73             cout << "[\033[1;33mInfo\033[0m] 第四次挥手发送失败 "
<< endl;
74             return -1;
75         }
```

```

76     else
77     {
78         cout << "[\033[1;31mSend\033[0m] 成功发送第四次挥手数据
" << endl;
79     }
80     cout << "[\033[1;33mInfo\033[0m] 四次挥手结束, 连接断开! "
<< endl;
81     return 1;
82 }

```

函数首先接收客户端的第一次挥手数据，然后发送第二次挥手数据，并等待接收客户端的第三次挥手数据，同样在超时情况下重新传输第二次挥手数据。

接着解析客户端的第三次挥手数据，如果通过校验则发送第四次挥手数据，最后输出四次挥手结束的信息，表示连接已经成功断开。

计算传输时间和吞吐率

该过程是在发送端中完成的

```

1  send(server, severAddr, len, (char *)(inputFile.c_str()),
    inputFile.length());
2  clock_t start1 = clock();
3  send(server, severAddr, len, buffer, i);
4  clock_t end1 = clock();
5
6  cout << "[\033[1;36mOut\033[0m] 传输总时间为:" << (end1 -
    start1) / CLOCKS_PER_SEC << "s" << endl;
7  cout << "[\033[1;36mOut\033[0m] 吞吐率为:" << fixed <<
    setprecision(2) << (((double)i) / ((end1 - start1) /
    CLOCKS_PER_SEC)) << "byte/s" << endl;

```

首先分别在发送文件内容前后运用clock函数记录两次时间。

然后用第二个clock函数调用返回的时间减去第一个clock函数调用返回的时间，并除以CLOCKS_PER_SEC（每秒钟的时钟周期数），得到传输时间。

最后，通过将文件大小除以传输时间来得到吞吐率，同时我为了数据的精准，使用了fixed和setprecision函数来设置输出的小数位数。

实验结果展示

路由器设置



Router

路由器IP: 127 . 0 . 0 . 1 服务器IP: 127 . 0 . 0 . 1

端口: 8888 服务器端口: 8889

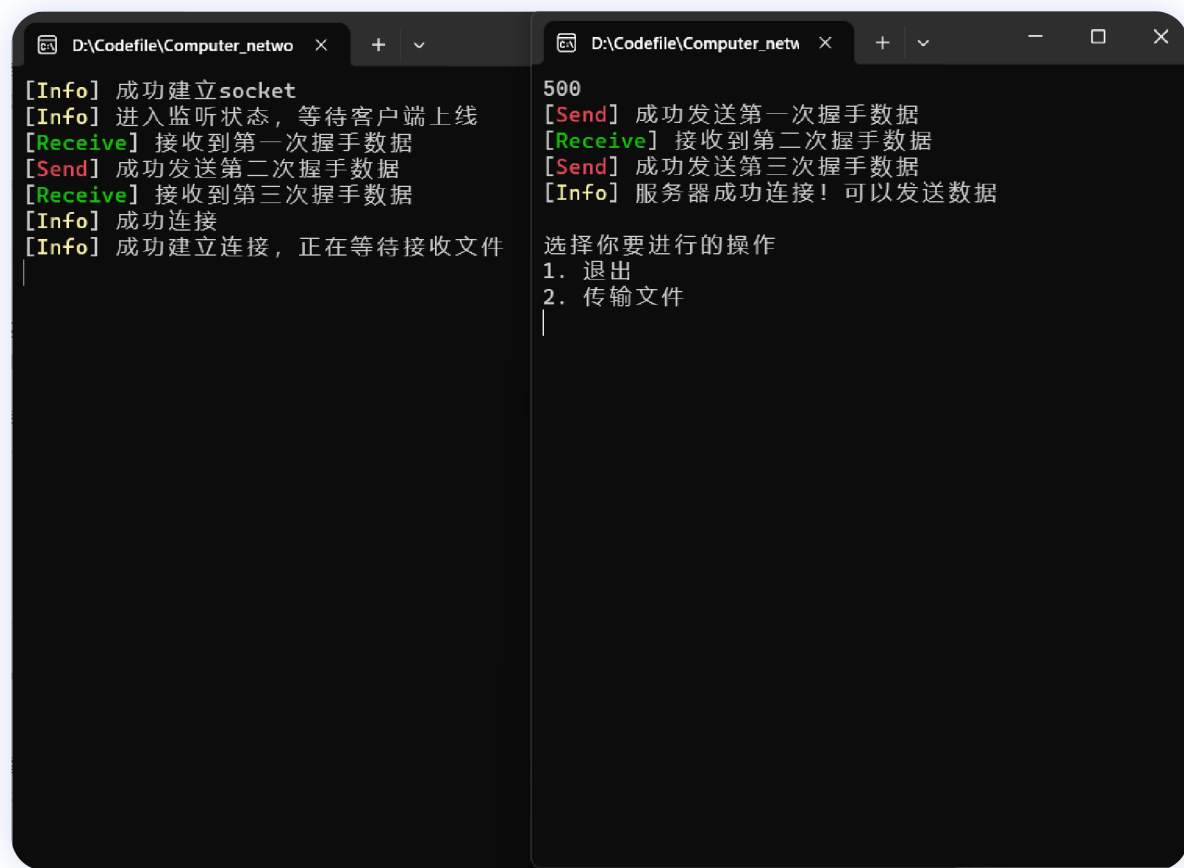
丢包率: 1 % 延时: 2 ms

确定 修改

日志

```
Router Ready!
Misscount :100 .
Delay :2 ms .
Delay 2 ms.
count:1.
Delay 2 ms.
count:2.
```

三次握手建立连接



The image shows two terminal windows side-by-side, both titled "D:\Codefile\Computer_netwo". The left window displays the server's perspective of the three-way handshake:

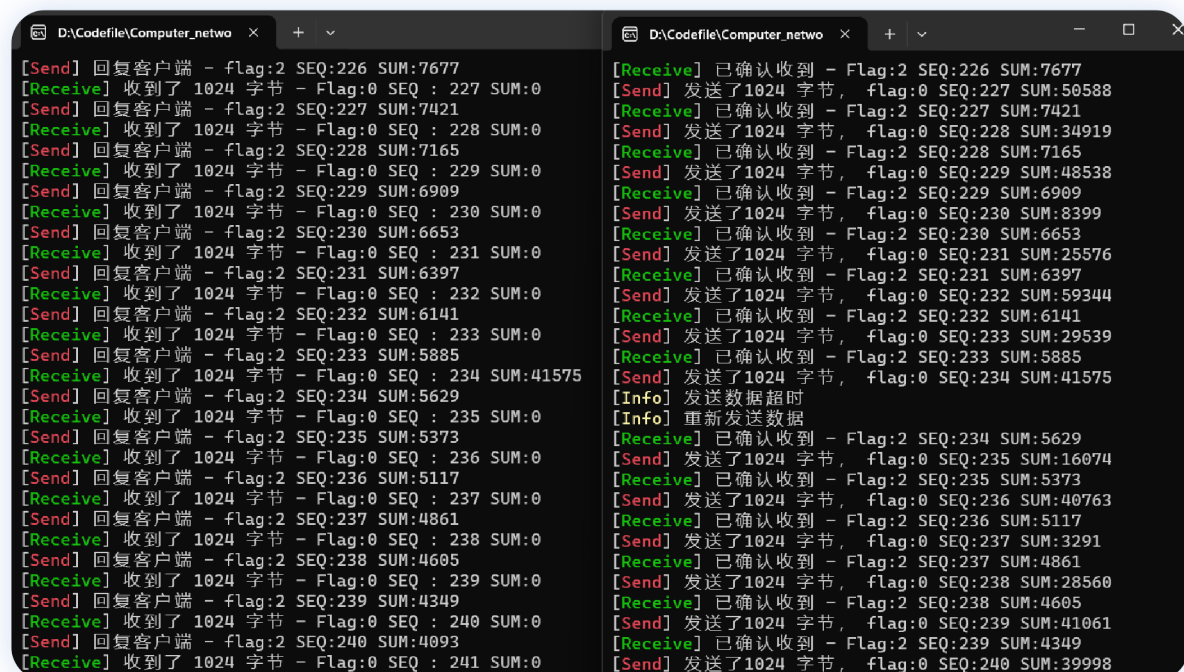
```
[Info] 成功建立socket
[Info] 进入监听状态, 等待客户端上线
[Receive] 接收到第一次握手数据
[Send] 成功发送第二次握手数据
[Receive] 接收到第三次握手数据
[Info] 成功连接
[Info] 成功建立连接, 正在等待接收文件
```

The right window displays the client's perspective:

```
500
[Send] 成功发送第一次握手数据
[Receive] 接收到第二次握手数据
[Send] 成功发送第三次握手数据
[Info] 服务器成功连接! 可以发送数据

选择你要进行的操作
1. 退出
2. 传输文件
```

丢包后实现超时重传



The image shows two terminal windows side-by-side, both titled "D:\Codefile\Computer_netwo". The left window displays the server's perspective of a data transmission with a timeout and retransmission:

```
[Send] 回复客户端 - flag:2 SEQ:226 SUM:7677
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 227 SUM:0
[Send] 回复客户端 - flag:2 SEQ:227 SUM:7421
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 228 SUM:0
[Send] 回复客户端 - flag:2 SEQ:228 SUM:7165
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 229 SUM:0
[Send] 回复客户端 - flag:2 SEQ:229 SUM:6909
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 230 SUM:0
[Send] 回复客户端 - flag:2 SEQ:230 SUM:6653
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 231 SUM:0
[Send] 回复客户端 - flag:2 SEQ:231 SUM:6397
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 232 SUM:0
[Send] 回复客户端 - flag:2 SEQ:232 SUM:6141
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 233 SUM:0
[Send] 回复客户端 - flag:2 SEQ:233 SUM:5885
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 234 SUM:41575
[Send] 回复客户端 - flag:2 SEQ:234 SUM:5629
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 235 SUM:0
[Send] 回复客户端 - flag:2 SEQ:235 SUM:5373
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 236 SUM:0
[Send] 回复客户端 - flag:2 SEQ:236 SUM:5117
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 237 SUM:0
[Send] 回复客户端 - flag:2 SEQ:237 SUM:4861
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 238 SUM:0
[Send] 回复客户端 - flag:2 SEQ:238 SUM:4605
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 239 SUM:0
[Send] 回复客户端 - flag:2 SEQ:239 SUM:4349
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 240 SUM:0
[Send] 回复客户端 - flag:2 SEQ:240 SUM:4093
[Receive] 收到了 1024 字节 - Flag:0 SEQ : 241 SUM:0
```

The right window displays the client's perspective of the same process:

```
[Receive] 已确认收到 - Flag:2 SEQ:226 SUM:7677
[Send] 发送了1024 字节, flag:0 SEQ:227 SUM:50588
[Receive] 已确认收到 - Flag:2 SEQ:227 SUM:7421
[Send] 发送了1024 字节, flag:0 SEQ:228 SUM:34919
[Receive] 已确认收到 - Flag:2 SEQ:228 SUM:7165
[Send] 发送了1024 字节, flag:0 SEQ:229 SUM:48538
[Receive] 已确认收到 - Flag:2 SEQ:229 SUM:6909
[Send] 发送了1024 字节, flag:0 SEQ:230 SUM:8399
[Receive] 已确认收到 - Flag:2 SEQ:230 SUM:6653
[Send] 发送了1024 字节, flag:0 SEQ:231 SUM:25576
[Receive] 已确认收到 - Flag:2 SEQ:231 SUM:6397
[Send] 发送了1024 字节, flag:0 SEQ:232 SUM:59344
[Receive] 已确认收到 - Flag:2 SEQ:232 SUM:6141
[Send] 发送了1024 字节, flag:0 SEQ:233 SUM:29539
[Receive] 已确认收到 - Flag:2 SEQ:233 SUM:5885
[Send] 发送了1024 字节, flag:0 SEQ:234 SUM:41575
[Info] 发送数据超时
[Info] 重新发送数据
[Receive] 已确认收到 - Flag:2 SEQ:234 SUM:5629
[Send] 发送了1024 字节, flag:0 SEQ:235 SUM:16074
[Receive] 已确认收到 - Flag:2 SEQ:235 SUM:5373
[Send] 发送了1024 字节, flag:0 SEQ:236 SUM:40763
[Receive] 已确认收到 - Flag:2 SEQ:236 SUM:5117
[Send] 发送了1024 字节, flag:0 SEQ:237 SUM:3291
[Receive] 已确认收到 - Flag:2 SEQ:237 SUM:4861
[Send] 发送了1024 字节, flag:0 SEQ:238 SUM:28560
[Receive] 已确认收到 - Flag:2 SEQ:238 SUM:4605
[Send] 发送了1024 字节, flag:0 SEQ:239 SUM:41061
[Receive] 已确认收到 - Flag:2 SEQ:239 SUM:4349
[Send] 发送了1024 字节, flag:0 SEQ:240 SUM:39998
```

输出传输时间和吞吐率，四次挥手断开连接

D:\Codefile\Computer_netwo

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 120 SUM:0

[Send] 回复客户端 - flag:2 SEQ:120 SUM:34813

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 121 SUM:0

[Send] 回复客户端 - flag:2 SEQ:121 SUM:34557

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 122 SUM:0

[Send] 回复客户端 - flag:2 SEQ:122 SUM:34301

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 123 SUM:0

[Send] 回复客户端 - flag:2 SEQ:123 SUM:34045

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 124 SUM:0

[Send] 回复客户端 - flag:2 SEQ:124 SUM:33789

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 125 SUM:0

[Send] 回复客户端 - flag:2 SEQ:125 SUM:33533

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 126 SUM:0

[Send] 回复客户端 - flag:2 SEQ:126 SUM:33277

[Receive] 收到了 1024 字节 - Flag:0 SEQ : 127 SUM:0

[Send] 回复客户端 - flag:2 SEQ:127 SUM:33021

[Receive] 收到了 265 字节 - Flag:0 SEQ : 128 SUM:0

[Send] 回复客户端 - flag:2 SEQ:128 SUM:32765

[Info] 文件传输结束

[Out] 接收的文件名:2.jpg

[Out] 接收的文件长度:5898505

[Out] 文件已成功下载到本地

[Receive] 接收到第一次挥手数据

[Send] 成功发送第二次挥手数据

[Receive] 接收到第三次挥手数据

[Send] 成功发送第四次挥手数据

[Info] 四次挥手结束，连接断开！

请按任意键继续...

D:\Codefile\Computer_netwo

[Receive] 已确认收到 - Flag:2 SEQ:121 SUM:34557

[Send] 发送了1024 字节, flag:0 SEQ:122 SUM:11354

[Receive] 已确认收到 - Flag:2 SEQ:122 SUM:34301

[Send] 发送了1024 字节, flag:0 SEQ:123 SUM:5474

[Receive] 已确认收到 - Flag:2 SEQ:123 SUM:34045

[Send] 发送了1024 字节, flag:0 SEQ:124 SUM:38097

[Receive] 已确认收到 - Flag:2 SEQ:124 SUM:33789

[Send] 发送了1024 字节, flag:0 SEQ:125 SUM:32134

[Receive] 已确认收到 - Flag:2 SEQ:125 SUM:33533

[Send] 发送了1024 字节, flag:0 SEQ:126 SUM:57777

[Receive] 已确认收到 - Flag:2 SEQ:126 SUM:33277

[Send] 发送了1024 字节, flag:0 SEQ:127 SUM:1644

[Receive] 已确认收到 - Flag:2 SEQ:127 SUM:33021

[Send] 发送了265 字节, flag:0 SEQ:128 SUM:2783

[Receive] 已确认收到 - Flag:2 SEQ:128 SUM:32765

[Send] 发送OVER信号

[Info] 对方已成功接收文件

[Out] 传输总时间为:6s

[Out] 吞吐率为:983084.17byte/s

选择你要进行的操作

1. 退出

2. 继续传输文件

1

[Send] 成功发送第一次挥手数据

[Receive] 接收到第二次挥手数据

[Send] 成功发送第三次挥手数据

[Info] 四次挥手结束，连接断开！

请按任意键继续...

性能测试指标

对三个文件进行传输测试（未经过路由）

文件名	文件大小	传输时间	吞吐率
helloworld.txt	1655808byte	1.08s	1655808.00byte/s
1.jpg	1857353byte	1.25s	1857353.00byte/s
2.jpg	5898505byte	4.18s	1474626.25byte/s
3.jpg	11968994byte	8.36s	1496124.25byte/s

图表结果如下

