



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

姓名：杨冰雪
学号：2110508
专业：计算机科学与技术

2023 年 11 月 17 日

目录

1 实验要求	2
2 协议设计	2
2.1 报文格式	2
2.2 建立连接	2
2.3 断开连接	5
2.4 差错检测	8
2.5 确认重传	9
2.6 停等机制	10
2.6.1 客户端发送消息	10
2.6.2 服务端发送 ACK 消息	10
3 实验结果	11
3.1 性能测试	13
3.1.1 不同文件大小的传输时间图	13
3.1.2 不同文件大小的吞吐率图	13
4 实验总结	14

1 实验要求

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，功能包括：建立连接、差错检测、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输。

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。完成给定测试文件的传输，显示传输时间和平均吞吐率。
3. 性能测试指标：吞吐率、延时，给出图形结果并进行分析。

2 协议设计

2.1 报文格式

报文格式

```
1 struct message
2 {
3     bool SYN = false;
4     bool FIN = false;
5     bool START = false; //标志开始传输文件
6     bool END = false; //标志文件传输完成
7     bool ACK = false;
8     u_short seq; //序列号
9     u_short ack; //确认号
10    u_long len; //数据长度
11    u_long num;
12    u_short checksum; //校验和
13    char data[datasize]; //数据长度
14 };
```

- 设计了一个结构体类型的报文来包含我们所需要的基本信息，包括标志位、序列号、确认号、数据长度、数据包数量、校验和和要发送的数据。
- 标志位设计了 SYN、FIN、START、END、ACK。其中 START 用来标志开始传输文件，END 标志文件传输完成。
- data 为要传输的图片和文字部分，其长度设置的是 1024 个字节。

2.2 建立连接

建立连接就是仿造三次握手的过程。

第一次握手 TCP 客户端向服务端发送带有【SYN】标识的信息，同时随机初始化一个序列号 $seq=x$ 。

第二次握手 服务器收到客户端发来的消息后，会检查消息的 SYN 标识是否为 true，如果是则确认连接，向客户端发出带有【SYN，ACK】标识的确认消息。确认号 $ack=x+1$ ，同时也会随机初始化一个序列号 $seq=y$ 。

第三次握手 客户端收到确认消息后, 也会同样检查消息的 SYN 和 ACK 标识是否为 true, 并且确认号是否等于上一个发送消息的序列号加一, 如果是则向服务器发出带有【ACK】标识的消息。确认号 $ack=y+1$, 序列号 $seq=x+1$ 。此时客户端的连接建立, 但服务器端还要检查收到的消息的 ACK 和确认号, 如果满足要求则服务器端的连接建立。

其具体实现代码如下:

客户端三次握手

```

1 void sendconnect()
2 {
3     //第一次握手
4     int iMode = 1;
5     ioctlsocket(Client, FIONBIO, (u_long FAR*) & iMode); //非阻塞设置
6     cout << " 【消息】 开始连接! 发送第一次握手! " << endl;
7     message recvMsg, sendMsg;
8     sendMsg.SYN=true; // 【SYN】
9     sendMsg.seq = getrand(); //随机产生一个序列号
10    sendMsg.checksum=0; //将校验和置为0
11    sendMsg.checksum= cksum((u_short*)&sendMsg, sizeof(sendMsg)); //计算校验和
12    if (sendto(Client, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&serveraddr,
13        sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
14        cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
15    }
16    //第二次握手
17    clock_t start = clock();
18    clock_t end;
19    while (true)
20    {
21        message msg;
22        if (recvfrom(Client, (char*)&msg, BUFFER, 0, (SOCKADDR*)&serveraddr, &len) ==
23            -1 || cksum((u_short*)&msg, sizeof(msg)) != 0) {
24            recvMsg = message();
25        }
26        else {
27            recvMsg=msg;
28        }
29        //超时重传
30        end = clock();
31        if ((end - start) > MAXTIME) {
32            cout << " 【ERROR】 连接超时, 请保持网络通畅和服务端正常运行! " << endl;
33            if (sendto(Client, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&serveraddr,
34                sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
35                cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
36            }
37            start = clock();
38            continue;
39        }
40        //接收确认
41        if (recvMsg.ACK && recvMsg.SYN && recvMsg.ack == sendMsg.seq + 1) {

```

```

39         cout << " 【消息】 接收到第二次握手!" << endl;
40         break;
41     }
42
43 }
44 //第三次握手
45 sendMsg.ACK = true;
46 sendMsg.seq = recvMsg.ack; //序列号等于收到的确认号
47 sendMsg.ack = recvMsg.seq + 1; //确认号等于收到的序列号+1
48 cout << " 【消息】 发送第三次握手! " << endl;
49 sendMsg.checksum = 0; //将校验和置为0
50 sendMsg.checksum = cksum((u_short*)&sendMsg, sizeof(sendMsg)); //计算校验和
51 if (sendto(Client, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&serveraddr,
52         sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
53     cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
54 }

```

服务端三次握手

```

1 void recvconnect()
2 {
3     //第一次握手
4     int iMode = 1; //非阻塞
5     ioctlsocket(Server, FIONBIO, (u_long FAR*) & iMode); //非阻塞设置
6     cout << " 【消息】 服务器等待连接! " << endl;
7     message recvMsg, sendMsg;
8     while (true)
9     {
10         message msg;
11         if (recvfrom(Server, (char*)&msg, BUFFER, 0, (SOCKADDR*)&clientaddr, &len) ==
12             -1 || cksum((u_short*)&msg, sizeof(msg)) != 0) {
13             recvMsg= message();
14         }
15         else {
16             recvMsg = msg;
17         }
18         if (recvMsg.SYN==true)
19         {
20             cout << " 【消息】 收到第一次握手! " << endl;
21             break;
22         }
23     }
24     //第二次握手
25     sendMsg.SYN=true; // 【SYN, ACK】
26     sendMsg.ACK=true;
27     sendMsg.ack = recvMsg.seq + 1; // 确认号等于收到的序列号+1
28     sendMsg.seq = getrand(); //随机获取序列号
29     sendMsg.checksum = 0; //将校验和置为0

```

```

29     sendMsg.checksum = cksum((u_short*)&sendMsg, sizeof(sendMsg)); // 计算校验和
30     if (sendto(Server, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&clientaddr,
31             sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
32         cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
33     }
34     else {
35         cout << " 【消息】 发送第二次握手! " << endl;
36     }
37     // 第三次握手
38     clock_t start=clock();
39     clock_t end;
40     while (true) {
41         message msg;
42         if (recvfrom(Server, (char*)&msg, BUFFER, 0, (SOCKADDR*)&clientaddr, &len) ==
43             -1 || cksum((u_short*)&msg, sizeof(msg)) != 0) {
44             recvMsg= message();
45         }
46         else{
47             recvMsg = msg;
48         }
49         end = clock();
50         if ((end - start) > MAXTIME) {
51             cout << " 【ERROR】 等待时间太长, 取消连接! " << endl;
52             return recvconnect();
53         }
54         if (recvMsg.ACK==true && recvMsg.ack == sendMsg.seq + 1) {
55             break;
56         }
57     }
58     cout << " 【消息】 接收到第三次握手, 连接成功! " << endl;
59 }

```

2.3 断开连接

断开连接采用的是四次挥手，客户端首先向服务端发出第一次挥手，服务端收到挥手后向客户端发送第二次挥手，然后服务端向客户端发送第三次挥手，客户端收到后向服务端返回第四次挥手，然后断开连接，程序运行结束。

客户端四次挥手

```

1 void closeconnect() { // 断开连接
2     // 第一次挥手
3     message recvMsg, sendMsg1;
4     sendMsg1.FIN = true; // 【FIN】
5     sendMsg1.seq = 65533; // 序列号
6     sendMsg1.checksum = 0; // 将校验和置为0
7     sendMsg1.checksum = cksum((u_short*)&sendMsg1, sizeof(sendMsg1)); // 计算校验和
8     if (sendto(Client, (char*)&sendMsg1, BUFFER, 0, (SOCKADDR*)&serveraddr,
9             sizeof(SOCKADDR)) == (SOCKET_ERROR)) {

```

```

9      cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
10  }
11  else {
12      cout << " 【消息】 断开连接！ 发送第一次挥手！ " << endl;
13  }
14  //第二次挥手
15  clock_t start = clock();
16  clock_t end;
17  while (true) {
18      end = clock();
19      if ((start - end) > disconnecttime) {
20          cout << " 【ERROR】 等待时间太长， 退出连接" << endl;
21          return closeconnect();
22      }
23      message msg;
24      if (recvfrom(Client, (char*)&msg, BUFFER, 0, (SOCKADDR*)&serveraddr, &len) ==
25          -1 || cksum((u_short*)&msg, sizeof(msg)) != 0) {
26          recvMsg = message();
27      }
28      else {
29          recvMsg = msg;
30      }
31      if (recvMsg.ACK == true && recvMsg.ack == sendMsg1.seq + 1) {
32          cout << " 【消息】 客户端收到第二次挥手" << endl;
33          break;
34      }
35  }
36  //第三次挥手
37  start = clock();
38  while (true) {
39      end = clock();
40      if ((start - end) > disconnecttime) {
41          cout << " 【ERROR】 等待时间太长， 退出连接" << endl;
42          return closeconnect();
43      }
44      message msg;
45      if (recvfrom(Client, (char*)&msg, BUFFER, 0, (SOCKADDR*)&serveraddr, &len) ==
46          -1 || cksum((u_short*)&msg, sizeof(msg)) != 0) {
47          recvMsg = message();
48      }
49      else {
50          recvMsg = msg;
51      }
52      if (recvMsg.FIN==true) {
53          cout << " 【消息】 客户端收到第三次挥手" << endl;
54          break;
55      }
56  }
57  //第四次挥手

```

```

56     message sendMsg2;
57     sendMsg2.ACK = true; // 【ACK】
58     sendMsg2.ack = recvMsg.seq + 1; // 确认号等于序列号+1
59     sendMsg2.checksum = 0; // 将校验和置为0
60     sendMsg2.checksum = cksum((u_short*)&sendMsg2, sizeof(sendMsg2)); // 计算校验和
61     if (sendto(Client, (char*)&sendMsg2, BUFFER, 0, (SOCKADDR*)&serveraddr,
62             sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
63         cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
64     }
65     else {
66         cout << " 【消息】 断开连接成功! " << endl;
67     }
68 }

```

服务端四次挥手

```

1 void closeconnect(message msg) {
2     // 第二次挥手
3     message sendMsg1;
4     sendMsg1.ACK=true; // 【ACK】
5     sendMsg1.ack = msg.seq + 1; // 确认号等于序列号+1
6     sendMsg1.checksum = 0; // 将校验和置为0
7     sendMsg1.checksum = cksum((u_short*)&sendMsg1, sizeof(sendMsg1)); // 计算校验和
8     if (sendto(Server, (char*)&sendMsg1, BUFFER, 0, (SOCKADDR*)&clientaddr,
9             sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
10         cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
11     }
12     else {
13         cout << " 【消息】 发送第二次挥手! " << endl;
14     }
15     // 第三次挥手
16     message sendMsg2;
17     sendMsg2.FIN = true; // 【FIN】
18     sendMsg2.seq = 65534; // 序列号
19     sendMsg2.checksum = 0; // 将校验和置为0
20     sendMsg2.checksum = cksum((u_short*)&sendMsg2, sizeof(sendMsg2)); // 计算校验和
21     if (sendto(Server, (char*)&sendMsg2, BUFFER, 0, (SOCKADDR*)&clientaddr,
22             sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
23         cout << " 【ERROR】 send error:" << WSAGetLastError << endl;
24     }
25     else {
26         cout << " 【消息】 发送第三次挥手! " << endl;
27     }
28     // 第四次挥手
29     message recvMsg;
30     clock_t start = clock();
31     clock_t end;
32     while (true) {
33         end = clock();

```



```

32     if ((end - start) > MAXTIME) {
33         end = clock();
34         if ((start - end) > disconnecttime) {
35             cout << " 【ERROR】 等待时间太长，退出连接" << endl;
36             recFileName();
37             return;
38         }
39     }
40     message msg;
41     if (recvfrom(Server, (char*)&msg, BUFFER, 0, (SOCKADDR*)&clientaddr, &len) ==
42         -1 || cksum((u_short*)&msg, sizeof(msg)) != 0) {
43         recvMsg = message();
44     }
45     else {
46         recvMsg = msg;
47     }
48     if (recvMsg.ACK == true && recvMsg.ack == sendMsg2.seq + 1) {
49         cout << " 【消息】 断开连接成功！" << endl;
50         break;
51     }
52 }

```

2.4 差错检测

收到消息后，我们需要进行差错检测。首先，需要检查消息类型是否正确。其次，需要检查消息的序列号是否正确。最后，需要检查校验和是否正确。

校验和的计算方法如下：

- 将消息头的校验和设置为 0
- 将消息头和数据看成 16 位整数序列，不足 16 位的最后补 0
- 每 16 位相加，溢出的部分加到最低位上
- 最后的结果取反

当接收端接收到数据时，需要用同样的方法计算校验和，如果校验和结果全为 0，说明消息正确，否则，说明消息损坏。实现校验和的代码如下：

校验和

```

1  u_short cksum(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      u_long sum = 0;
7      while (count--) {
8          sum += *buf++;

```

```

9         if (sum & 0xffff0000) {
10             sum &= 0xffff;
11             sum++;
12         }
13     }
14     return ~(sum & 0xffff);
15 }

```

实现检验消息类型和序列号的代码如下：

检验消息类型和序列号

```

1 if (recvMsg.ACK==true && recvMsg.ack == seq) {
2     cout << "收到服务器的确认数据包！" << endl;
3     cout << "checksum=" << recvMsg.checksum << ", len=" << recvMsg.len << endl;
4     cout << endl;
5     return true;
6 }

```

2.5 确认重传

确认重传包括差错重传和超时重传。因为是类似于 rdt3.0 的设计，所以当遇到接收到的消息的校验和检验出错，或者接受到消息时出错和超过规定时间内没有接收到消息时，都会进行重传。

每次当接收到的消息出现检验和出错和接受错误时，都会将接受消息进行初始化，等待其重新接受，一旦等待超时，进行重传。这里重传的次数也进行了限制，最大重传次数设的 10 次，一旦超过 10 次，就退出循环，不再进行发送，这样能防止陷入死循环中。其代码如下：

确认重传

```

1 while (true) {
2     message msg;
3     if (recvfrom(Client, (char*)&msg, BUFFER, 0, (SOCKADDR*)&serveraddr, &len) == -1
4         || cksum((u_short*)&msg, sizeof(msg)) != 0) {
5         recvMsg = message();
6     }
7     else {
8         recvMsg = msg;
9     }
10    end = clock();
11    if ((end - start) > MAXTIME) { //超时重传
12        cout << "应答超时，重新发送数据包" << endl;
13        if (sendto(Client, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&serveraddr,
14            sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
15            cout << "【ERROR】 send failed:" << WSAGetLastError << endl;
16        }
17        count++;
18        cout << "尝试重新发送第" << count << "次数据包" << endl;
19        if (count >= 10) {
20            break;
21        }
22    }
23 }

```

```

19     }
20     start = clock();
21 }
22 if (recvMsg.ACK==true && recvMsg.ack == seq) {
23     cout << "收到服务器的确认数据包! " << endl;
24     cout << "checksum=" << recvMsg.checksum << ", len=" << recvMsg.len << endl;
25     cout << endl;
26     return true;
27 }
28 }

```

2.6 停等机制

该机制下客户端每发送一条消息后，都要等待服务端确认后发送 ACK 消息，当客户端接收到返回的 ACK 后才能发送下一条。

2.6.1 客户端发送消息

停等机制-客户端发送消息

```

1 message recvMsg;
2 sendMsg.seq = seq;
3 sendMsg.checksum = 0; //将校验和置为0
4 sendMsg.checksum = cksum((u_short*)&sendMsg, sizeof(sendMsg)); //计算校验和
5 if (sendto(Client, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&serveraddr,
6     sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
7     cout << "【ERROR】 send error:" << WSAGetLastError << endl;
8 }
9 cout << "checksum=" << sendMsg.checksum << ", len=" << sendMsg.len << endl;

```

2.6.2 服务端发送 ACK 消息

停等机制-服务端发送确认消息

```

1 if (recvMsg.seq == seq) {
2     cout << "收到seq为" << recvMsg.seq << "的数据包" << endl;
3     cout << "checksum=" << recvMsg.checksum << ", len=" << recvMsg.len << endl;
4     sendMsg.ACK = true;
5     sendMsg.ack = recvMsg.seq;
6     sendMsg.checksum = 0; //将校验和置为0
7     sendMsg.checksum = cksum((u_short*)&sendMsg, sizeof(sendMsg)); //计算校验和
8     if (sendto(Server, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&clientaddr,
9         sizeof(SOCKADDR)) == (SOCKET_ERROR)) {
10         cout << "【ERROR】 send error:" << WSAGetLastError << endl;
11     }
12     else {
13         cout << "向服务端发送确认数据包!" << endl;
14     }
15 }

```

```
13     }  
14     cout << endl;  
15     fout.write(recvMsg.data, recvMsg.len);  
16     break;  
17 }
```

3 实验结果

当运行程序，进行文件传输时，第一张照片传输结果如下：

```
发送seq为1812的数据包成功!!!  
checksum=10513, len=1024  
收到服务器的确认数据包!  
checksum=37712, len=0  
  
发送seq为1813的数据包成功!!!  
checksum=52952, len=1024  
收到服务器的确认数据包!  
checksum=37711, len=0  
  
发送seq为1814的数据包成功!!!  
checksum=19168, len=841  
收到服务器的确认数据包!  
checksum=37710, len=0  
  
【消息】成功发送文件!  
传输总时间1.112s  
吞吐率13356.2kbps
```

第二张照片传输结果如下：

```
发送seq为5759的数据包成功!!!  
checksum=58264, len=1024  
收到服务器的确认数据包!  
checksum=33765, len=0  
  
发送seq为5760的数据包成功!!!  
checksum=2386, len=1024  
收到服务器的确认数据包!  
checksum=33764, len=0  
  
发送seq为5761的数据包成功!!!  
checksum=11017, len=265  
收到服务器的确认数据包!  
checksum=33763, len=0  
  
【消息】成功发送文件!  
传输总时间3.508s  
吞吐率13445.9kbps
```

第三张照片传输结果如下：

```
发送seq为11687的数据包成功!!!  
checksum=34798, len=1024  
收到服务器的确认数据包!  
checksum=27837, len=0  
  
发送seq为11688的数据包成功!!!  
checksum=37855, len=1024  
收到服务器的确认数据包!  
checksum=27836, len=0  
  
发送seq为11689的数据包成功!!!  
checksum=3646, len=482  
收到服务器的确认数据包!  
checksum=27835, len=0  
  
【消息】成功发送文件!  
传输总时间7.153s  
吞吐率13379.5kbps
```

helloworld.txt 文件传输结果如下:

```
发送seq为1615的数据包成功!!!  
checksum=30715, len=1024  
收到服务器的确认数据包!  
checksum=37909, len=0  
  
发送seq为1616的数据包成功!!!  
checksum=30714, len=1024  
收到服务器的确认数据包!  
checksum=37908, len=0  
  
发送seq为1617的数据包成功!!!  
checksum=30457, len=1024  
收到服务器的确认数据包!  
checksum=37907, len=0  
  
【消息】成功发送文件!  
传输总时间0.965s  
吞吐率13719.4kbps
```

最后在文件夹下找到了传输的文件，可以看出传输的文件均正确，其接收文件的大小和传输文件的大小是一样的，打开后也是相同的，可以看出传输成功！

名称	修改日期	类型	大小
x64	2023/11/16 0:38	文件夹	
1.jpg	2023/11/17 15:26	JPG 文件	1,814 KB
2.jpg	2023/11/17 15:26	JPG 文件	5,761 KB
3.jpg	2023/11/17 15:27	JPG 文件	11,689 KB
helloworld.txt	2023/11/17 15:27	文本文档	1,617 KB
recv.vcxproj	2023/11/15 18:05	VC++ Project	7 KB
recv.vcxproj.filters	2023/11/15 18:05	VC++ Project Fil...	2 KB
recv.vcxproj.user	2023/11/13 22:36	Per-User Project...	1 KB
server.cpp	2023/11/16 22:02	C++ Source	9 KB

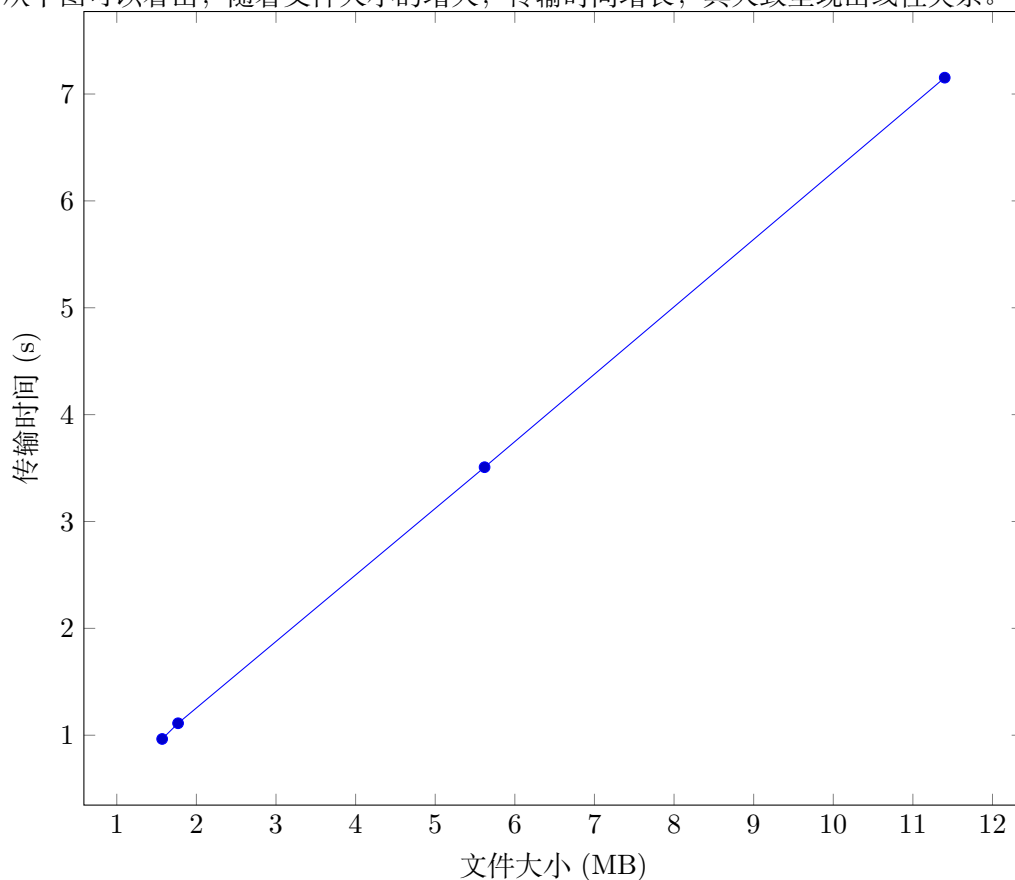
同时当我们运行 router.exe 程序进行丢包设置后，可以看出当超时后进行了重传。

```
发送seq为87的数据包成功!!!  
checksum=34752, len=1024  
应答超时, 重新发送数据包  
尝试重新发送第1次数据包  
收到服务器的确认数据包!  
checksum=39436, len=0  
  
发送seq为88的数据包成功!!!  
checksum=56757, len=1024  
收到服务器的确认数据包!  
checksum=39435, len=0  
  
发送seq为89的数据包成功!!!  
checksum=58034, len=1024  
应答超时, 重新发送数据包  
尝试重新发送第1次数据包  
收到服务器的确认数据包!  
checksum=39434, len=0
```

3.1 性能测试

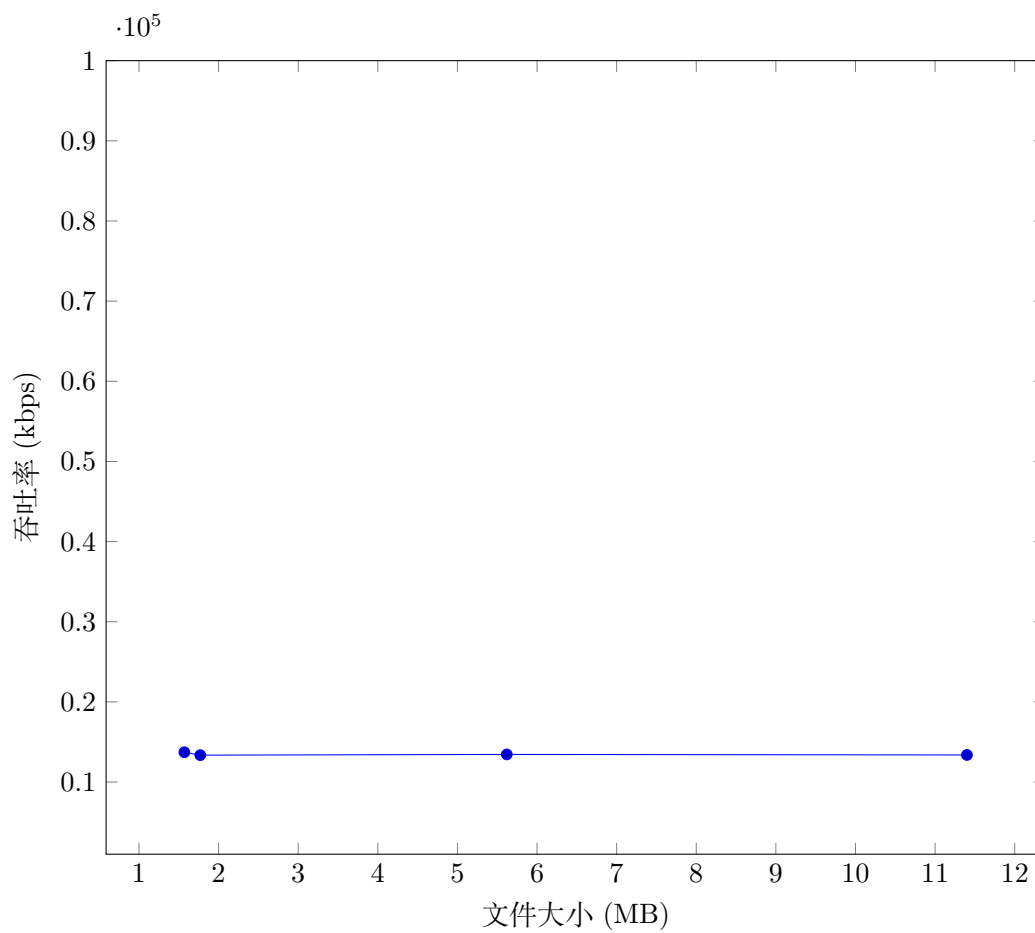
3.1.1 不同文件大小的传输时间图

从下图可以看出, 随着文件大小的增大, 传输时间增长, 其大致呈现出线性关系。



3.1.2 不同文件大小的吞吐率图

从下图可以看出, 随着文件大小的增大, 其吞吐率基本上不变。



4 实验总结

通过本次实验，我学会了对 UDP 实现可靠文件传输。让我对许多建立可靠连接的机制，如三次握手、差错检验、超时重传等有了更深刻的理解。但同时我在编写代码的时候也遇到了一些问题，比如协议设计好后，当我进行三次握手时，因为最开始在 main 函数里面定义的变量，当定义的函数想要使用时，还需要通过引用的方式进行传入，不仅增加了代码量，还容易出错，所以我定义为了全局的变量。以及最开始在编写接收到消息但消息检验码错误时，如何又进行重传这些最开始有点困难。