



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

计算机网络实验报告

基于 UDP 服务设计可靠传输协议并编程实现

李潇逸 2111454

年级：2021 级

专业：信息安全、法学

指导教师：张建忠

2023 年 11 月 18 日

目录

一、 实验要求	1
二、 原理	1
(一) client 端	1
(二) server 端	1
(三) 停等机制	1
三、 代码分析	1
四、 展示	5
五、 总结	6

一、 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。

二、 原理

(一) client 端

建立连接：发送一条带有标识 SYN 的消息，表示希望建立连接，对方收到后回复对应的 ACK，连接建立成功。当收到 ACK 后进入状态 1

发送文件：

- 消息 1，文件开始标识位 SF=1，index= 发送文件需要的消息条数-1，filelength= 最后一条消息 msg 成员的有效位数。当收到对应的 ACK 后，进入状态 2。每条消息都是收到上一条对应的 ACK 后再发送。
- 发送文件，每条 msg 的有效长度固定为 1024，最后一条 EF=1，收到 ACK 后退回状态 1

断开连接：发送一条标识为 FIN 的消息，对方收到后回复对应的 ACK，断开连接成功，退回状态 0

client 端由于是主动发送的一端，一般情况下不会设计蓝色线条的状态跳转，但是可能会有超过重传次数情况下做出的断网处理

(二) server 端

建立连接：收到一条 SYN，回复对应的 ACK，连接建立成功，进入状态 1

发送文件：收到包含 SF、index、filelength 的消息，开始接收文件，进入状态 2。对于每条消息都返回一条 ACK。当收到包含标志位 EF 的消息，检查 index 与收到文件消息的条数，若一致，则返回 ACK，退回状态 1

断开连接：收到包含标识为 FIN 的消息，返回 ACK，进入状态 0

(三) 停等机制

简而言之就是发现一定时间不能收到返回的 ACK 就重发

三、 代码分析

包设计包括了各种包必须的属性，以及验证包是否损坏的函数

包设计

```
1 struct message
2 {
3     #pragma pack(1)
4     u_long  flag{};
5     u_short seq{}; // 序列号
6     u_short ack{}; // 确认号
7     u_long  len{}; // 数据部分长度
```

```
8     u_long num{}; //发送的消息包含几个包
9     u_short checksum{}; //校验和
10    char data[1024]{}; //数据长度
11    #pragma pack()
12    message() {
13        memset(this, 0, sizeof(message));
14    }
15    bool isSYN() {
16        return this->flag & 1;
17    }
18    bool isFIN() {
19        return this->flag & 2;
20    }
21    bool isSTART() {
22        return this->flag & 4;
23    }
24    bool isEND() {
25        return this->flag & 8;
26    }
27    bool isACK() {
28        return this->flag & 16;
29    }
30    bool isEXT() {
31        return this->flag & 32;
32    }
33    void setSYN() {
34        this->flag |= 1;
35    }
36    void setFIN() {
37        this->flag |= 2;
38    }
39    void setFIR() {
40        this->flag |= 4;
41    }
42    void setEND() {
43        this->flag |= 8;
44    }
45    void setACK() {
46        this->flag |= 16;
47    }
48    void setEXT() {
49        this->flag |= 32;
50    }
51    void setchecksum() {
52        int sum = 0;
53        u_char* temp = (u_char*)this;
54        for (int i = 0; i < 8; i++)
55            {
```

```

56         sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
57         while (sum >= 0x10000)
58         { // 溢出
59             int t = sum >> 16; // 将最高位回滚添加至最低位
60             sum += t;
61         }
62     }
63     this->checksum = ~(u_short)sum; // 按位取反, 方便校验计算
64 }
65 bool corrupt() {
66     // 包是否损坏
67     int sum = 0;
68     u_char* temp = (u_char*)this;
69     for (int i = 0; i < 8; i++)
70     {
71         sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
72         while (sum >= 0x10000)
73         { // 溢出
74             int t = sum >> 16; // 计算方法与设置校验和相同
75             sum += t;
76         }
77     }
78     // 把计算出来的校验和和报文中该字段的值相加, 如果等于0xffff, 则校验成功
79     if (checksum + (u_short)sum == 65535)
80         return false;
81     return true;
82 }
83 void output() {
84     cout << "checksum=" << this->checksum << ", len=" << this->len <<
85     endl;
86 }
};

```

三次握手

三次握手

```

1  int beginconnect()
2  {
3      int iMode = 1; // 1: 非阻塞, 0: 阻塞
4      ioctlsocket(Client, FIONBIO, (u_long FAR*) & iMode); // 非阻塞设置
5      SetColor(12, 0);
6      cout << "开始连接! 发送第一次握手!" << endl;
7      message recvMsg, sendMsg;
8      sendMsg.setSYN();
9      sendMsg.seq = 88;
10
11     sendmessage(sendMsg);
12

```

```

13     int start = clock();
14     int end;
15     while (true)
16     {
17         recvMsg = recvmessage();
18         if (!recvMsg.isEXT())
19         {
20             end = clock();
21             if (end - start > 50) {
22                 SetColor(12, 0);
23                 cout << "连接超时,请确认网络通畅和服务端启动无误后再运行本程
序!" << endl;
24                 sendmessage(sendMsg);
25                 break;
26             }
27             continue;
28         }
29         if (recvMsg.isACK() && recvMsg.isSYN() && recvMsg.ack == sendMsg.seq
+ 1) {
30             SetColor(14, 0);
31             cout << "收到第二次握手!" << endl;
32             break;
33         }
34     }
35     sendMsg.setACK();
36     sendMsg.seq = 89;
37     sendMsg.ack = recvMsg.seq + 1;
38     SetColor(14, 0);
39     cout << "发送第三次握手的数据包" << endl;
40     sendmessage(sendMsg);
41     return 0;
42 }
43

```

四次挥手

四次挥手

```

1  iint closeconnect() { // 断开连接
2      message recvMsg, sendMsg;
3      sendMsg.setFIN();
4      sendMsg.seq = 65534; // 此处是u_short的表示范围的最大值-1, 而我们收到的将会
再加一, 那么已经到了u_short的最大值了, 就自然结束了。
5      sendmessage(sendMsg);
6      cout << "发送出去第一次挥手!" << endl;
7      int count = 0;
8      while (true) {
9          Sleep(100);
10         if (count >= 50) {
11             SetColor(12, 0);

```

```
12         cout << "等待时间太长，退出连接" << endl;  
13         return closeconnect();  
14     }  
15     recvMsg = recvmessage();  
16     if (!recvMsg.isEXT()) {  
17         continue;  
18     }  
19     if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1) {  
20         break;  
21     }  
22     count++;  
23 }  
24 SetColor(12, 0);  
25 cout << "接收到确认连接，断开连接成功" << endl << endl;  
26 return 0;  
27 }
```

四、 展示



图 1: Caption

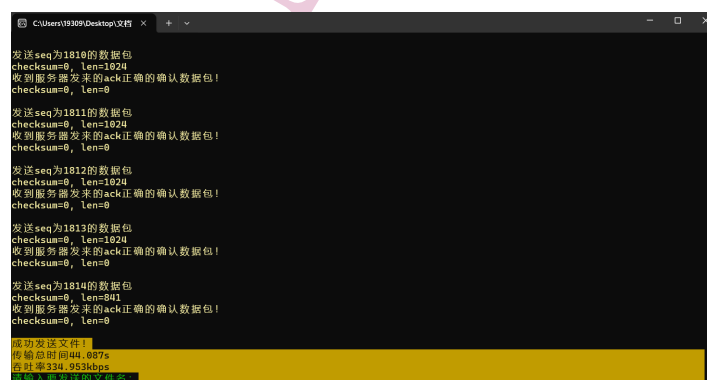


图 2: client 端

```
C:\Users\19309\Desktop\文档
收到seq为1809的数据包
checksum=0, len=1024
发送确认收到的数据包(对应的ack)

收到seq为1810的数据包
checksum=0, len=1024
发送确认收到的数据包(对应的ack)

收到seq为1811的数据包
checksum=0, len=1024
发送确认收到的数据包(对应的ack)

收到seq为1812的数据包
checksum=0, len=1024
发送确认收到的数据包(对应的ack)

收到seq为1813的数据包
checksum=0, len=1024
发送确认收到的数据包(对应的ack)

收到seq为1814的数据包
checksum=0, len=841
发送确认收到的数据包(对应的ack)

接收文件成功!!

传输总时间44.88s
吞吐量335.007kbps
服务器正在等待中.....
```

图 3: server 端



图 4: 路由器显示

1.jpg	2023/11/18 17:48	JPG 图片文件	1,814 KB
server.cpp	2023/11/16 17:35	C++ Source	10 KB
server.exe	2023/11/16 17:55	应用程序	35 KB

图 5: 结果

五、 总结

实现了 UDP 传输，将研究如何加快传输速度