



南 开 大 学
Nankai University

南 开 大 学
计 算 机 学 院
计 算 机 网 络

3-1 报告

学号：1810780

姓名：苑华莹

年级：2018 级










专业：计算机科学与技术

2020 年 12 月 11 日

一、 代码架构	2
二、 版本管理	2
三、 协议设计	3
四、 功能展示	5
(一) 建立连接	5
(二) 差错检测	6
(三) 确认重传与累计确认	7
五、 结果展示	9
(一) 吞吐率	9
(二) 传输结果	10
六、 坑点与总结	10
(一) 文件的打开与保存	10
(二) 多线程的设计与并发维护	11
(三) 输入的管理	11

一、 代码架构

代码整体架构如下图所示：

 test	2020/12/5 22:57	文件夹	
 bit_cal.h	2020/12/2 16:57	C Header File	1 KB
 Client.cpp	2020/12/9 18:46	C++ Source File	6 KB
 Client.exe	2020/12/9 18:39	应用程序	3,208 KB
 common.h	2020/12/3 18:04	C Header File	1 KB
 package.h	2020/12/5 22:20	C Header File	5 KB
 README.md	2020/12/2 15:26	MD 文件	1 KB
 Server.cpp	2020/12/9 18:39	C++ Source File	6 KB
 Server.exe	2020/12/9 18:39	应用程序	3,205 KB

每个文件的功能如下所示：

1. bit_cal: 用于处理有关二进制的计算
2. common.h: 用于规定 server 端和 client 端共同的 flag 对应的位置约定
3. package.h: 包含传输文件的打包解包、校验和计算、校验和检验等功能
4. client.cpp: 客户端（发送端），包含三次握手四次挥手，并处理发送方输入的发送任务（含超时重传、差错检验、确认重传）
5. server.cpp: 服务端（接收端），包含三次握手四次挥手；处理发送方的发送内容并保存（含超时重传、差错检验、确认重传）

二、 版本管理

为了更好的进行版本管理,本次作业利用率 github 工具,网址为:<https://github.com/yhy-2000/NetworkHomework>提交的部分记录如下所示:

```

:...skipping...
commit d382bcc61aa59aaa96b801cca510f0633d4a113d (HEAD -> master, origin/master, origin/HEAD)
Author: yhy-2000 <1792885489@qq.com>
Date: Thu Dec 10 00:39:56 2020 +0800

    挥手修改成功

commit 0ff11fcc4afa718487a86a85b07b0deccfb0e93b
Author: yhy-2000 <1792885489@qq.com>
Date: Wed Dec 9 22:37:31 2020 +0800

    更新累计确认

commit 295586864a3cf8e104be152b6b535418fc0292b3
Author: yhy-2000 <1792885489@qq.com>
Date: Wed Dec 9 20:26:09 2020 +0800

    累计确认完毕

commit 5a89068002e016dae7c7494929c693901901075a
Author: yhy-2000 <1792885489@qq.com>
Date: Wed Dec 9 19:54:37 2020 +0800

    更新3-2

commit 4c19d16d9f79a05463803bc976cd02fbe2e37981
Author: yhy-2000 <1792885489@qq.com>
Date: Tue Dec 8 16:13:34 2020 +0800

    3-2 还差累计确认

:...skipping...
commit d382bcc61aa59aaa96b801cca510f0633d4a113d (HEAD -> master, origin/master, origin/HEAD)
Author: yhy-2000 <1792885489@qq.com>
Date: Thu Dec 10 00:39:56 2020 +0800

    挥手修改成功

```

图 1: GITHUB 提交截图

三、 协议设计

协议头部分完全仿照 tcp 协议设计，tcp 报文字段这里不再赘述。每个 package 的内容如下所示：

```

1  \\package.h
2  struct package
3  {
4      //首先是UDP的真实首部
5      string srcPort, desPort; //源端口 目的端口
6      string len, check_sum; //数据包长度, 校验和
7      string data; //数据部分
8
9      //下面是TCP新增内容
10     string flag; //开了16位, 具体功能在common.h中
11     string ackNum, seq;
12
13     string packNum; //数据包序号
14
15
16 }
17

```

```
18 //伪首部只是用来计算校验和，并不打包进package
19 string encode(package& p)
20 {
21     string ans = "";
22     ans += match(p.srcPort);
23     ans += match(p.desPort);
24
25     //计算长度
26     get_len(p);
27     ans += match(p.len);
28
29     //计算校验和
30     get_sum(f, p);
31     ans += match(p.check_sum);
32
33
34     ans += p.flag;
35     ans += p.ackNum;
36     ans += p.seq;
37     ans += p.packNum;
38     ans += p.data;
39     return ans;
40 }
41
42 void decode(string s, package& p)
43 {
44     p.srcPort = s.substr(0, 16);
45     p.desPort = s.substr(16, 16);
46     p.len = s.substr(32, 16);
47     p.check_sum = s.substr(48, 16);
48     p.flag = s.substr(64, 16);
49     p.ackNum = s.substr(80, 32);
50     p.seq = s.substr(112, 32);
51     p.packNum = s.substr(144, 32);
52     p.data = s.substr(176);
53 }
54
55 //通过校验和判断是否有数据丢失
56 bool check_lose(package p)
57 {
58     //保存旧的校验和
59     f.len=p.len;
60     string olds = p.check_sum;
61     p.check_sum = match("");
62
63     get_sum(f, p);
64     string news = p.check_sum;
65
66     return news == olds;
```

67 }

其中，flag 各个位的功能如下所示：

```

1 //common.h
2 //规定 flag 各个位
3 #define SYN 0
4 #define ACK 1
5 #define PSH 2
6 #define RST 3
7 #define URG 4
8 #define FIN 5
9 #define ACK_GROUP 6 //主要用于3-1的ack确认号（0或1）
10 #define END 7 //当前数据包是否是整个完整数据包的结尾

```

四、 功能展示

(一) 建立连接

建立连接与断开连接采用 TCP 标准协议的三次握手四次挥手模式，握手与挥手通过设置 SYN FIN 等标志位，来建立一个稳定连接

```

1 //三次握手
2 void connect()
3 {
4     //第一次握手(SYN=1, seq=x)
5     string flag = match(""); flag[SYN] = '1';
6     _rdt_send(flag); flag[SYN] = '0';
7     cout << "第一次握手成功发送\n";
8
9     //第二次握手 SYN=1, ACK=1, seq=y, ACKnum=x+1
10    while (file_queue.empty());
11    package p = file_queue.front(); file_queue.pop();
12    assert(p.flag[SYN] == '1' && p.flag[ACK] == '1');
13    cout << "第二次握手成功接收\n";
14
15    //第三次握手 ACK=1, ACKnum=y+1
16    flag[ACK] = '1';
17    _rdt_send(flag);
18    cout << "第三次握手成功发送\n";
19    state = 1;
20 }
21
22 //四次挥手
23 void disconnect()
24 {
25     state = 2;
26     //第一次挥手(FIN=1, seq=x) c->s
27     string flag = match(""); flag[FIN] = '1';

```

```

28 _rdt_send(flag); flag[FIN] = '0';
29 cout << "第一次挥手发送成功\n";
30
31 //第二次挥手(ACK=1, ACKnum=x+1)          s->c
32 while (file_queue.empty());
33 package p = file_queue.front(); file_queue.pop();
34
35
36 assert(p.flag[ACK] == '1');
37 cout << "第二次挥手成功接收\n";
38
39 //第三次挥手(FIN=1, seq=y)                s->c
40 while (file_queue.empty());
41 cout << "qsize: " << file_queue.size() << "\n";
42 p = file_queue.front(); file_queue.pop();
43
44
45 assert(p.flag[FIN] == '1');
46 cout << "第三次挥手成功接收\n";
47
48 //第四次挥手(ACK=1, ACKnum=y+1)          c->s
49 flag[ACK] = '1'; _rdt_send(flag);
50 cout << "第四次挥手发送成功\n";
51 }

```

(二) 差错检测

差错检测通过 package.h 中的 check_lose 函数来计算，计算流程如下所示：

1. 发送方生成一个伪首部，综合利用伪首部以及真正的数据包，共同计算出一个校验和
2. 接收方利用接收到的数据以及生成的伪首部计算出一个校验和
3. 比较这两个校验和观察是否差错
4. 如果有差错将会返回 ACK=0，发送端重新发送

```

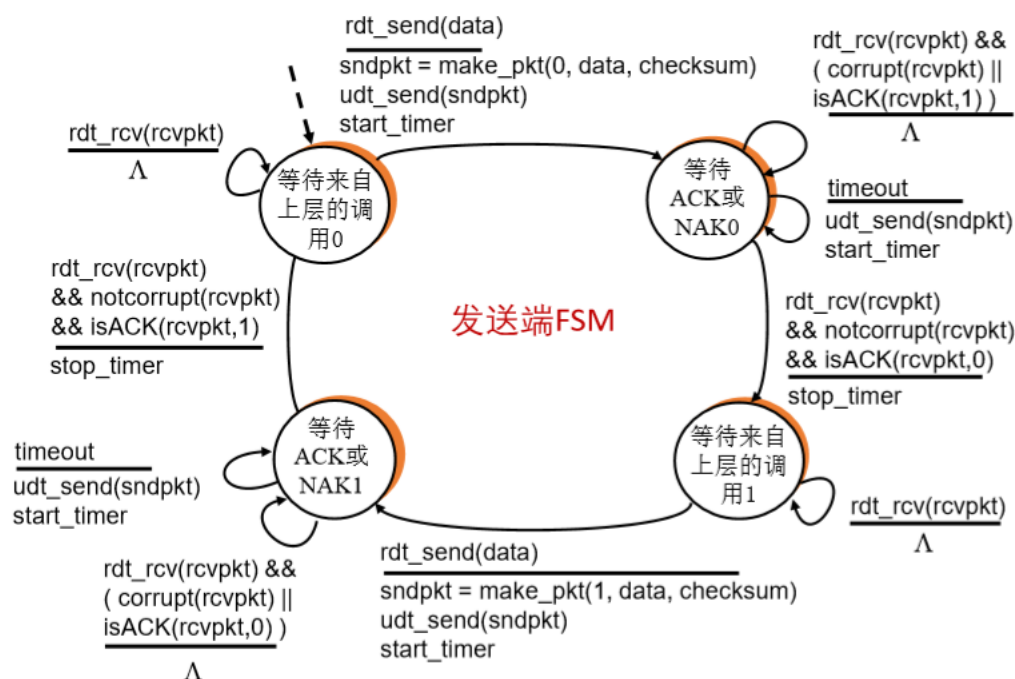
1 //package.h
2 //通过校验和判断是否有数据丢失
3 bool check_lose(package p)
4 {
5     //保存旧的校验和
6     f.len=p.len;
7     string olds = p.check_sum;
8     p.check_sum = match("");
9
10    get_sum(f, p);
11    string news = p.check_sum;
12
13    return news == olds;
14 }

```

(三) 确认重传与累计确认

第二次作业运用的状态机如下所示：

■ rdt3.0: 发送端状态机



对于发送方，主要的状态机函数如下所示：

```

1 void send()
2 {
3     //分组发送
4     int groupNum = (sendData.size()+max_len-1)/max_len;
5     string groupData;
6     for (int i = 0; i < groupNum; ++i)
7     {
8         if (i < (groupNum - 1)) groupData = sendData.substr(i * max_len, max_len);
9         else groupData = sendData.substr(i * max_len);
10        bool end=(i==groupNum-1);
11
12
13        //下面开始分奇偶发送
14        rdt_send(groupData,t,end);
15        start=clock();
16        while(1)
17        {
18            while(file_queue.empty());
19            package p=file_queue.front(); file_queue.pop();
20            finish=clock();
21            if (check_lose(p)&&p.isACK(t))
22            {

```



```

23         t^=1;
24         break;
25     }
26     else if((finish-start) > 10 || !p.isACK(t) || !check_lose(p))
27     {
28         rdt_send(sendData, t, end);
29     }
30 }
31 }
32 }

```

对于接收方，主要的状态机函数如下所示：

```

1 void recv_manager()
2 {
3     int t=0;
4     while(1)
5     {
6         if(state==2){disconnect();break;} //进入挥手状态
7
8         while(!file_que.empty())
9         {
10             package p=file_que.front();file_que.pop();
11             if(check_lose(p)&& p.flag[ACK_GROUP]=='0'+t)
12             {
13                 recv_data+=p.data;
14                 rdt_send("",t);
15                 t^=1;
16                 if(pic_set.count(p.data)){
17                     pic=1;file_name=p.data;start=clock();
18                 }
19                 if(p.flag[END]=='1')
20                 {
21                     if(pic&&!pic_set.count(p.data))
22                     {
23                         save_pic();
24                         finish=clock();
25                         cout<<"传输用时: "<<finish-start<<endl;
26                         double sec=(finish-start)/1000.0;
27                         double mb=recv_data.size()/(8.0*1024*1024);
28                         double v=mb/sec;
29                         cout<<"吞吐率: "<< v<<"MB/s\n";
30                         pic=0;
31                     }
32                     else cout<<"接收的消息: "<<recv_data<<endl;
33                     cout<<"recv_data.size(): "<<recv_data.size()<<"\n";
34                     recv_data="";
35                     break;
36                 }
37             }

```

```

38     }
39     else
40     {
41         rdt_send("",p.ackgroup());
42     }
43 }
44 }
45
46 }

```

五、 结果展示

程序运行结果如下所示：

D:\Desktop\计网作业\作业三\3-1\Server.exe	D:\Desktop\计网作业\作业三\3-1\Client.exe
<pre> 第二次握手成功发送 第三次握手成功接收 接收的消息: 1. jpg recv_data.size(): 5 1. jpg 传输用时: 1880 吞吐率: 0.942186MB/s recv_data.size(): 14858824 接收的消息: 2. jpg recv_data.size(): 5 2. jpg 传输用时: 5623 吞吐率: 1.0004MB/s recv_data.size(): 47188040 接收的消息: 3. jpg recv_data.size(): 5 3. jpg 传输用时: 10550 吞吐率: 1.08195MB/s recv_data.size(): 95751952 接收的消息: 1. txt recv_data.size(): 5 1. txt 传输用时: 1369 吞吐率: 1.15347MB/s recv_data.size(): 13246464 第一次挥手成功接收 第二次挥手发送成功 第三次挥手发送成功 </pre>	<pre> 请输入目的端口号: 8888 第一次握手成功发送 第二次握手成功接收 第三次握手成功发送 1. jpg pic size: 14858824 2. jpg pic size: 47188040 3. jpg pic size: 95751952 1. txt pic size: 13246464 q 第一次挥手发送成功 第二次挥手成功接收 qsize: 1 第三次挥手成功接收 第四次挥手发送成功 ----- Process exited after 51.65 seconds with return value 0 请按任意键继续. . . </pre>

图 2: 3-1 结果

首先开启 server 端，在 client 端用户输入目的端口号，便可三次握手建立连接；

client 用户可以输入文件名称，当系统识别到此文件在此目录下时，便利用二进制的格式打开相应文件，并进行传输。接收方在接收到此文件名称时，开启 pic 模式，将之后输入的文件保存到本地。

(一) 吞吐率

本次实验设定数据包的最大长度为 1500，根据结果图可知，吞吐率维持在 1MB/s（如果数据包长度更长，则传输速度会更快）。

(二) 传输结果

D:\Desktop\计网作业\作业三\3-1\Server.exe	D:\Desktop\计网作业\作业三\3-1\Client.exe
<pre> 第二次握手成功发送 第三次握手成功接收 接收的消息: 1.jpg recv_data.size(): 5 1.jpg 传输用时: 1880 吞吐率: 0.942186MB/s recv_data.size(): 14858824 接收的消息: 2.jpg recv_data.size(): 5 2.jpg 传输用时: 5623 吞吐率: 1.0004MB/s recv_data.size(): 47188040 接收的消息: 3.jpg recv_data.size(): 5 3.jpg 传输用时: 10550 吞吐率: 1.08195MB/s recv_data.size(): 95751952 接收的消息: 1.txt recv_data.size(): 5 1.txt 传输用时: 1369 吞吐率: 1.15347MB/s recv_data.size(): 13246464 第一次挥手成功接收 第二次挥手发送成功 第三次挥手发送成功 </pre>	<pre> server client 请输入目的端口号: 8888 第一次握手成功发送 第二次握手成功接收 第三次握手成功发送 1.jpg pic size: 14858824 2.jpg pic size: 47188040 3.jpg pic size: 95751952 1.txt pic size: 13246464 q 第一次挥手发送成功 第二次挥手成功接收 qsize: 1 第三次挥手成功接收 第四次挥手发送成功 ----- Process exited after 51.65 seconds with return value 0 请按任意键继续. . . </pre>

图 3: 3-1 结果 (out 开头为接收方保存的文件)

六、 坑点与总结

本次实验的难点也包括文件的打开与保存/多线程的并发控制/用户任意输入机制下的程序逻辑维护等等。

(一) 文件的打开与保存

如果直接打开二进制文件，并将二进制文件保存成 char* 的话，由于 jpg 文件很可能出现连续的零，这就对应了 char 类型的”，文件到此便会提前结束。

因此，我采用手动将二进制文件转成 char* 的方式，完美的解决了这个问题，函数如下所示：

```

1 // client.cpp
2 void get_pic()
3 {
4     send();
5
6     // 读入的文件地址
7     string file_addr="test\\"+sendData;
8
9     ifstream in(file_addr, ios::binary);
10
11     sendData="";
12
13     char buf;
```

```
14 while (in.read(&buf, sizeof(buf)))
15 {
16     for (int i=0; i<sizeof(buf)<<3; ++i)
17     {
18         if (buf&(1<<i)) sendData+='1';
19         else sendData+='0';
20     }
21 }
22 cout<<"pic size: "<<sendData.size()<<endl;
23 }
```

(二) 多线程的设计与并发维护

尽管本次实验只涉及了两个线程，但线程的并行控制以及何时 kill 等问题仍然值得仔细思考。

(三) 输入的管理

由于本次实验旨在实现一个更智能的程序，因此将由用户决定何时挥手、传输文本或文件、传输什么文件等。因此，主函数设计了 send_manager() 对这些输入进行统一处理。

一旦检测到退出，便会将全局状态置为 2（挥手状态），并顺次进入 disconnect 函数；如果检测到文件名，将调用 get_pic 函数完成指定文件的读入。

send_manager() 函数如下所示：

```
1 void send_manager()
2 {
3
4     while (cin >> sendData)
5     {
6         //检查断开连接
7         if (sendData == "q")
8         {
9             state=2;
10            break;
11        }
12
13        if(pic_set.count(sendData))
14        {
15            get_pic();
16        }
17
18        send();
19
20    }
21 }
```