



南开大学
Nankai University

南 开 大 学
计 算 机 学 院
计 算 机 网 络

3-2 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
(一) 数据包设计	3
(二) 程序流程设计	5
四、 功能展示	5
(一) 建立连接	5
(二) 差错检测	6
(三) 滑动窗口与状态机	7
(四) 超时重传	8
(五) 累计确认	9
五、 结果展示	10
(一) 吞吐率	11
(二) 传输结果	12
六、 坑点与总结	12
(一) 文件的打开与保存	12
(二) 多线程的设计与并发维护	13
(三) 输入的管理	13

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/
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/
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

(一) 数据包设计

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; //序列号和确认序列号主要在3-1中使用,对于3-2用处不大
12
13  string packNum; //数据包序号
14 }
15
16 ...
17
18 string encode(package& p)
19 {
20     string ans = "";
21     ans += match(p.srcPort);
22     ans += match(p.desPort);
23
24     //计算长度
25     get_len(p);
26     ans += match(p.len);
27
28     //计算校验和
29     get_sum(f, p);
30     ans += match(p.check_sum);
31
32
33     ans += p.flag;
34     ans += p.ackNum;
35     ans += p.seq;
36     ans += p.packNum;
37     ans += p.data;
38     return ans;
39 }
40
41 void decode(string s, package& p)
42 {
43     p.srcPort = s.substr(0, 16);
44     p.desPort = s.substr(16, 16);
45     p.len = s.substr(32, 16);
46     p.check_sum = s.substr(48, 16);
47     p.flag = s.substr(64, 16);
48     p.ackNum = s.substr(80, 32);
49     p.seq = s.substr(112, 32);
50     p.packNum = s.substr(144, 32);
51     p.data = s.substr(176);
52 }

```

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

```

(二) 程序流程设计

程序运行流程：

1. 开启 server 端
2. 在 client 端用户输入目的端口号，三次握手建立连接；
3. client 用户可以输入文件名称或任意文本。
4. 当系统识别到输入文件名在此目录下时，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
5. client 在想退出的时机输入 q，程序四次挥手，断开连接

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_que.empty());
33     package p = file_que.front(); file_que.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_que.empty());
41     cout << "qsize: " << file_que.size() << "\n";
42     p = file_que.front(); file_que.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;

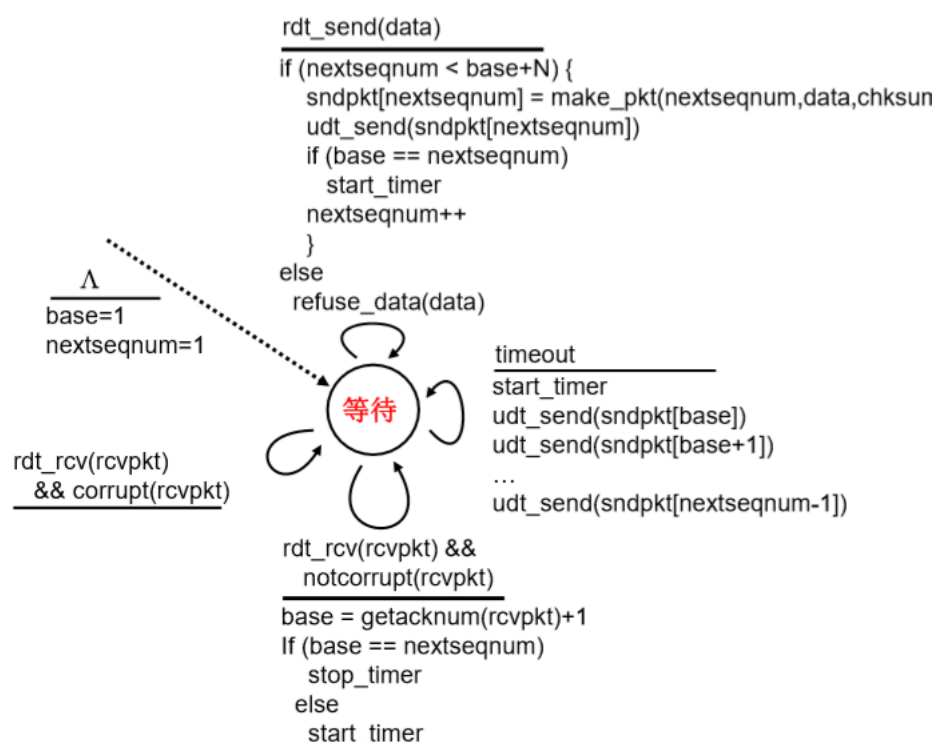
```

14 }

(三) 滑动窗口与状态机

第二次作业运用的状态机如下所示：

■ GBN发送端扩展FSM



对于发送方，主要的状态机函数如下所示：

```

1
2 void maintain_sb()
3 {
4     while(ack_state[sendbase]) sendbase++;
5 }
6
7 void send()
8 {
9     //先分组
10    groupNum = (sendData.size()+max_len-1)/max_len;
11    vector<string> groupData;
12    for (int i = 0; i < groupNum; ++i)
13    {
14        if (i < (groupNum - 1)) groupData.push_back(sendData.substr(i * max_len, max_len));
15        else groupData.push_back(sendData.substr(i * max_len));
16    }
17
18    //cout<<"groupNum: "<<groupNum<<"\n";

```



```

19
20 //下面开始发送,根据sendbase和win_size确定可以发送的数据包的下标范围
21 int cur_packnum=sendbase,cnt=0;
22 int old_sendbase=sendbase;
23 while(cur_packnum<old_sendbase+groupNum)
24 {
25     while(cur_packnum<(sendbase+win_size)&&cur_packnum<(old_sendbase+groupNum))
26     {
27         int cur_seqnum = seqnum + cnt * max_len;
28
29         string seq = match(to_bin(to_string(cur_seqnum)), 32);
30         string curpackNum= match(to_bin(to_string(cur_packnum)), 32);
31         bool end=(cnt==(groupNum-1));
32
33         simple_packet sp(groupData[cnt],seq,curpackNum,end);
34
35         node no(cur_packnum,clock(),sp);
36         ack_state[cur_packnum]=0;
37         valid[cur_packnum]=1;
38
39         unrecv[cur_packnum]=no;
40
41         rdt_send(sp);
42
43         cnt++;cur_packnum++;
44     }
45 }
46
47 }

```

发送方通过在接收线程调用 maintain_sb 函数,来动态维护 sendbase; send 线程感知到 sendbase 的变化,便会不断移动窗口继续发送。

(四) 超时重传

```

1 //处理超时的线程
2 void* timeout_handler(void* args)
3 {
4     while(1)
5     {
6         Sleep(500);
7         for(int i=0;i<maxn;++i)
8         {
9             //没有这个package或者没有接受
10            Sleep(100);
11            if(!valid[i]||ack_state[i])continue;
12
13            int cur_pckn=unrecv[i].packNum;
14            clock_t cur_time=clock();

```

```

15
16 //超时重传
17 if ((cur_time-unrecv[i].start)>1000)
18 {
19     cout<<"序号: "<<i<<"\n";
20     cout<<"超时packnum: "<<unrecv[i].packNum<<"\n";
21     cout<<"当前时间: "<<cur_time<<"\n";
22     cout<<"pack start time: "<< unrecv[i].start<<"\n";
23     unrecv[i].start=clock();
24     rdt_send(unrecv[i].pack);
25 }
26 }
27 }
28
29 }

```

(五) 累计确认

对于接收方，维护 `recvbase`，表示当前已经收到的连续数据包的最大序列号，利用 `recvbase` 返回期待的下一个序列号，主要的状态机函数如下所示：

```

1 void recv_manager()
2 {
3     while(1)
4     {
5         if (state==2){continue;} //进入挥手状态
6
7         while (state==1&&!file_queue.empty())
8         {
9             package p=file_queue.front();
10            //接收到挥手信号,进入disconnect函数
11            if (p.flag[FIN] == '1')
12            {
13                state=2;
14                disconnect();
15                break;
16            }
17            file_queue.pop();
18            if (check_lose(p))
19            {
20                maintain_rb();
21                string nxt_seq=match(to_bin(to_string(recvbase+1)),32);
22                rdt_send("",nxt_seq,p.packNum);
23
24                recv_data+=p.data;
25                if (pic_set.count(p.data)){
26                    pic=1;
27                    recv_data=file_name=p.data;
28                    start=clock();

```

```
29     }
30     if (p.flag[END]== '1')
31     {
32         if (pic&&!pic_set.count(p.data))
33         {
34             save_pic();
35             finish=clock();
36             cout<<" 传输用时: "<<finish-start<<endl;
37             double sec=(finish-start)/1000.0;
38             double mb=recv_data.size()/(8.0*1024*1024);
39             double v=mb/sec;
40             cout<<" 吞吐率: "<< v<<"MB/s\n";
41             pic=0;
42         }
43         else cout<<"接收的消息: "<<recv_data<<"\n";
44         cout<<"recv_data.size(): "<<recv_data.size()<<"\n";
45         recv_data="";
46         break;
47     }
48
49     }
50     else
51     {
52         rdt_send("",p.ackNum,p.packNum);
53     }
54 }
55 }
56
57 }
```

五、 结果展示

程序运行结果如下所示:

```

D:\Desktop\计网作业\作业三\NetworkHomework - 副本\
D:\Desktop\计网作业\作业三\NetworkHomework - 副本\3-2\Client.exe

server
第一次握手成功接收
第二次握手成功发送
第三次握手成功接收
接收的消息: 3.jpg
recv_data.size(): 5
3.jpg
传输用时: 4269
吞吐率: 2.67382MB/s
recv_data.size(): 95751952
接收的消息: 1.jpg
recv_data.size(): 5
1.jpg
传输用时: 650
吞吐率: 2.72509MB/s
recv_data.size(): 14858824
接收的消息: 2.jpg
recv_data.size(): 5
2.jpg
传输用时: 2131
吞吐率: 2.63972MB/s
recv_data.size(): 47188040
接收的消息: 1.txt
recv_data.size(): 5
1.txt
传输用时: 457
吞吐率: 3.45536MB/s
recv_data.size(): 13246464
接收的消息: 再见啦!!
recv_data.size(): 10
SrcPort: 0000000000008888
desPort: 0000000000008888
len: 0000000000000000
check_sum: 0000000000000000
data:
flag: 0000010000000000
ackNum: 00000000000000000000000000000000
packNum: 00000000000000000000000000000000
seq: 000000010100011000110001111000111000
第一次挥手成功接收
第二次挥手发送成功

client
please input the source port: 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
3.jpg
groupNum: 1
pic size: 95751952
groupNum: 10640
1.jpg
groupNum: 1
pic size: 14858824
groupNum: 1651
2.jpg
groupNum: 1
pic size: 47188040
groupNum: 5244
1.txt
groupNum: 1
pic size: 13246464
groupNum: 1472
再见啦!!
groupNum: 1
q
第一次挥手发送成功
SrcPort: 0000000000008888
desPort: 0000000000008888
len: 0000000000000000
check_sum: 0000000000000000
data:
flag: 0100000000000000
ackNum: 00000000000000000000000000000000
packNum: 00000000000000000000000000000000
seq: 000000000000000000000000000000000010
第二次挥手成功接收
SrcPort: 0000000000008888
desPort: 0000000000008888
len: 0000000000000000
check_sum: 0000000000000000
data:
flag: 0100010000000000

```

图 2: 3-2 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 2.6MB/s（如果数据包长度更长，则传输速度会更快）。

对比 3-1（1MB/s），可以明显看出窗口机制要优于停等机制。

(二) 传输结果

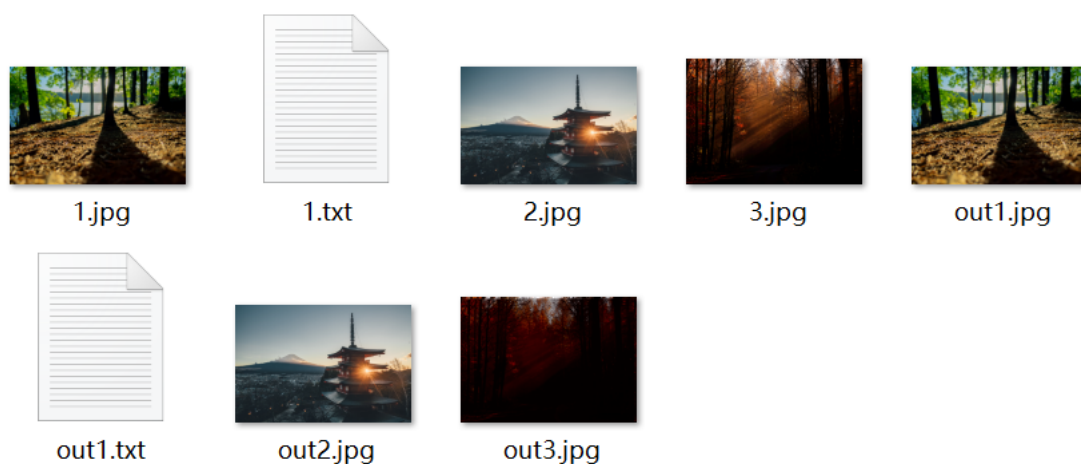


图 3: 3-2 结果 (out 开头为接收方保存的文件)

六、 坑点与总结

与 3-1 类似，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

本次实验涉及到多个线程，如下所示：

```
1 //接受信息新开一个线程
2 pthread_t* thread = new pthread_t;
3 pthread_create(thread, NULL, receive, NULL);
4
5 //处理信息的线程
6 pthread_t* thread2 = new pthread_t;
7 pthread_create(thread2, NULL, recv_manager, NULL);
8
9 //超时重传的线程
10 pthread_t* thread3 = new pthread_t;
11 pthread_create(thread3, NULL, timeout_handler, NULL);
```

因此，本次实验的并发控制更加困难。总的来讲，超时重传以及其他线程的终止条件都是全局进入挥手状态。

(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 }
```