



南开大学
Nankai University

南 开 大 学
计 算 机 学 院
计 算 机 网 络

3-3 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 15 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、 拥塞控制	5
(一) 状态机	5
(二) 算法原理	6
(三) 代码实现	6
五、 坑点与总结	8
六、 结果展示	9
(一) 传输时间和吞吐率	9
(二) 传输结果	9

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、拥塞控制、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、拥塞控制、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 4c19d16d9f79a05463803bc976cd02fbc2e37981
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

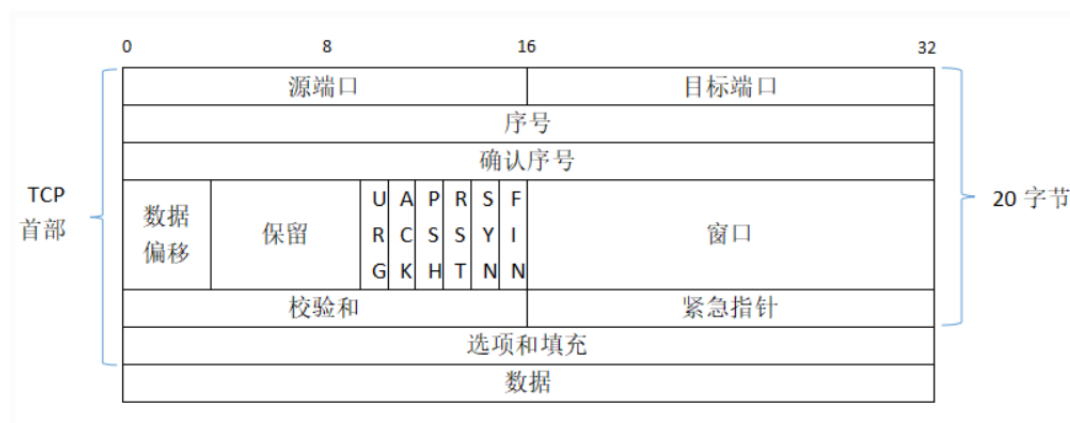
```

图 1: GITHUB 提交截图

三、 协议设计

(一) 数据包设计

数据包部分几乎完全按照标准 TCP 的报文头部设计，每个数据包的内容如下所示：



在具体实现过程中，我将数据包封装成了 `package` 类，类中属性对应上图中的报文格式：

```
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：

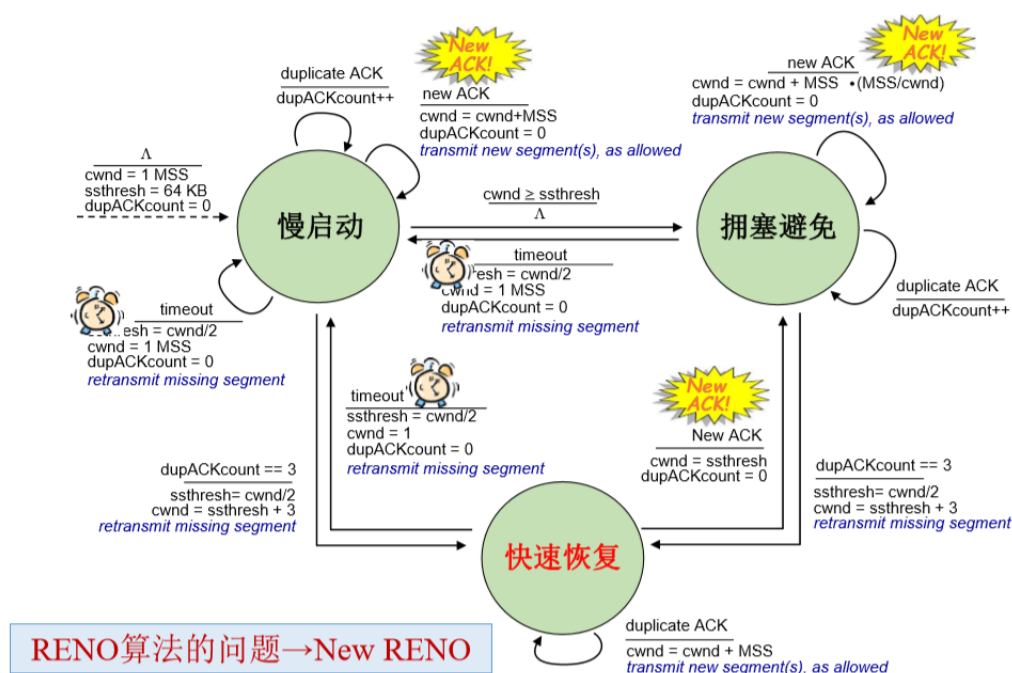
1. 开启 server 端
2. 在 client 端用户输入目的端口号，三次握手建立连接；
3. client 用户可以输入文件名称或任意文本。
4. 当系统识别到输入文件名在此目录下时，便利用二进制的格式打开相应文件，并进行传输；接收方在接收到此文件名称时，开启 pic 模式，将之后输入的文件保存到本地
5. client 在想退出的时机输入 q，程序四次挥手，断开连接

四、 拥塞控制

（一） 状态机

关于拥塞控制这一部分，我实现了讲义中的 reno 算法，算法状态机如下所示：

■ TCP拥塞控制：RENO算法状态机



(二) 算法原理

reno 算法主要包含三个状态，分别是：

1. 慢启动

慢启动阶段的特点如下：

初始拥塞窗口： $cwnd=1(MSS)$

每接收到一个 ACK， $cwnd$ 增 1(MSS)

当连接初始建立或报文段超时未得到确认时，TCP 拥塞控制进入慢启动阶段

注意：由于每次 RTT 之后， $cwnd$ 都会翻倍，因此“慢启动”实际上并不慢，而是呈现指数增长的状态。

2. 拥塞避免

阈值 $ssthresh$ ：拥塞窗口达到该阈值时，慢启动阶段结束，进入拥塞避免阶段

每个 RTT， $cwnd$ 增 1（线性增长）

3. 快速恢复

快速恢复主要决定于收到的重复应答数据的初始门限值，与慢启动不同，Reno 的发送方用额外到达的应答为后续包定时。

发送方窗口的上限值 $= \min$ （接收方窗口，拥塞窗口）

(三) 代码实现

首先，将 reno 状态机封装成了一个新的函数，代码如下所示，相关说明已经写在注释中。

```

1 //reno状态机函数
2 void reno_FSM(int nxt_packnum)
3 {
4     //维护全局acknum
5
6     if(acknum == nxt_packnum){//dup ACK
7         if(reno_state==0||reno_state==1)
8         {
9             dupACKcount++;
10            if (dupACKcount==3)
11            {
12                ssthresh= win_size/2;
13                win_size = ssthresh + 3;
14                reno_state=2;//状态机状态由 "慢启动"或"拥塞控制" 变 "快速恢复"
15            }
16        }
17        else if(reno_state==2)
18        {
19            win_size++;
20        }
21    }
22    else //new ack
23    {
24        acknum = nxt_packnum;
25        if(reno_state==0)
26        {
27            reno_state=1; //状态机状态由"慢启动"变"拥塞避免"
28            acknum = nxt_packnum;
29            dupACKcount=0;
30            if (win_size<ssthresh) win_size++;
31        }
32        else if(reno_state==1)
33        {
34            win_size = win_size + (MSS/win_size);//由于win_size本身就表示数据包的序号而不是字节的序号，因此不需要再乘以MSS
35            dupACKcount = 0 ;
36        }
37        else if(reno_state==2)
38        {
39            reno_state=1; //状态机状态由"慢启动"变"拥塞避免"
40            win_size = ssthresh;
41            dupACKcount = 0 ;
42        }
43    }
44 }

```

对于状态机中关于超时的处理，将在 timeout_handler 函数中完成

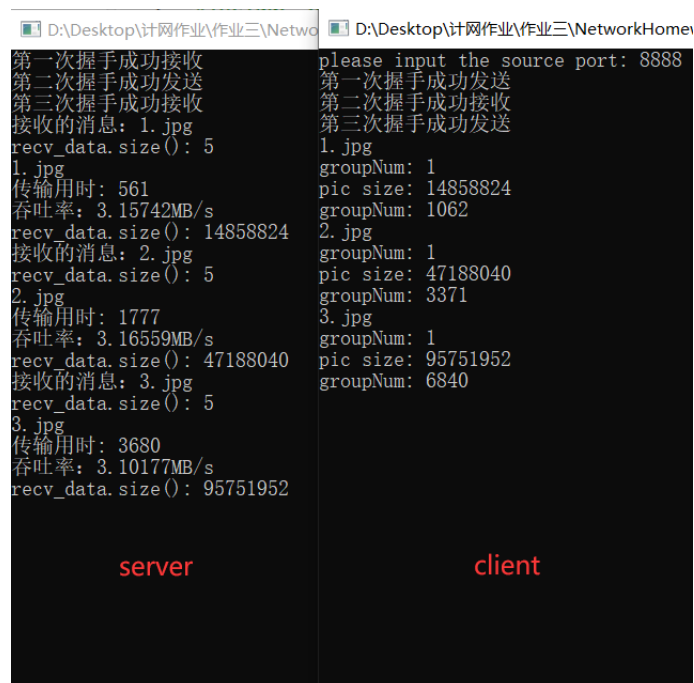

```
2 //处理超时的线程
3 //拥塞控制部分加入了相应状态转移函数
4 void* timeout_handler(void* args)
5 {
6     while(1)
7     {
8         Sleep(500);
9         for(int i=0;i<maxn;++i)
10        {
11            //没有这个package或者没有接受
12            Sleep(100);
13            if(!valid[i]||ack_state[i]) continue;
14
15            int cur_pckn=unrecv[i].packNum;
16            clock_t cur_time=clock();
17
18            //超时重传
19
20            if((cur_time-unrecv[i].start)>100)
21            {
22                cout<<"超时packnum: "<<unrecv[i].packNum<<"\n";
23                cout<<"当前时间: "<<cur_time<<"\n";
24                cout<<"pack start time: "<< unrecv[i].start<<"\n";
25                unrecv[i].start=clock();
26                rdt_send(unrecv[i].pack);
27
28                //超时状态转移
29                reno_state=0;
30                ssthresh = win_size/2;
31                win_size = 1;
32                dupACKcount = 0;
33            }
34        }
35    }
36
37 }
```

五、 坑点与总结

多线程的难点永远在于并发控制，本任务也不例外。在算法逻辑上进行巩固完善、适当加锁、在while(1)循环中加入Sleep等手段，都可以作为提高多线程并发稳定性的有效手段。

六、 结果展示

(一) 传输时间和吞吐率



```
D:\Desktop\计网作业\作业三\NetworkHomework>
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: 1.jpg
recv_data.size(): 5
1.jpg
传输用时: 561
吞吐率: 3.15742MB/s
recv_data.size(): 14858824
接收的消息: 2.jpg
recv_data.size(): 5
2.jpg
传输用时: 1777
吞吐率: 3.16559MB/s
recv_data.size(): 47188040
接收的消息: 3.jpg
recv_data.size(): 5
3.jpg
传输用时: 3680
吞吐率: 3.10177MB/s
recv_data.size(): 95751952

server

D:\Desktop\计网作业\作业三\NetworkHomework>
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
1.jpg
groupNum: 1
pic size: 14858824
groupNum: 1062
2.jpg
groupNum: 1
pic size: 47188040
groupNum: 3371
3.jpg
groupNum: 1
pic size: 95751952
groupNum: 6840

client
```

根据上图可以发现, 吞吐率大概在 3MB/s(注: 经过测试, 同一份文件在同学的游戏本上传输速率达到了 12MB/s, 猜测传输速率受笔记本性能影响较大); 传输时间根据文件大小的不同, 也会有差别。

(二) 传输结果

