****

**《计算机网络》**

**期中项目报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 数据科学与计算机学院 | | | | | |
| **学号姓名** | **：** | 16340273杨芮、16340316朱逸渠 | | | | | |
| **专业方向** | **：** | 数字媒体方向 | | | | | |
| **时间** | **：** | 2018 | 年 | 12 | 月 | 13 | 日 |

1. **项目要求**

Please write a network application, LFTP, to support large file transfer between two computers in the Internet.   
1. A student can finish the project by himself or herself. Two students may form a group to work on the project too. If two students work together, the tasks to be finished should be assigned to each one by almost 50%;  
2. The technical requirements are:  
(1) Please choose one of following programing languages: C, C++， Java, Python;  
(2) LFTP should use a client-server service model;  
(3) LFTP must include a client side program and a server side program; Client side program can not only send a large file to the server but also download a file from the server.   
Sending file should use the following format：  
LFTP lsend myserver mylargefileGetting file should use the following format  
LFTP lget myserver mylargefileThe parameter myserver can be a url address or an IP address.   
(4) LFTP should use UDP as the transport layer protocol.   
(5) LFTP must realize 100% reliability as TCP;  
(6) LFTP must implement flow control function similar as TCP;  
(7) LFTP must implement congestion control function similar as TCP;  
(8) LFTP server side must be able to support multiple clients at the same time;  
(9) LFTP should provide meaningful debug information when programs are executed.

1. **项目开发环境**

Java JRE1.8.181

Windows 10

1. **项目源码**

（1）Github网址：

（2）执行命令：

a.开启服务端

java LFTPSever

b.客户端上传文件到服务器

java LFTPClient –m lsend –s 服务器地址 –f 上传的文件路径 –o 输出到服务器的文件路径

c.客户端从服务器下载文件

java LFTPClient –m lget –s 服务器地址 –f 从服务器下载的文件路径 –o 输出到本地的文件路径

1. **项目架构**

（1）文件目录：

（2）客户端-服务端服务模型：

本次项目使用了客户端-服务端服务模型，将客户端与服务器区分开来。每一个客户端软件的实例都可以向一个服务器或应用程序服务器发出请求。本次项目开发时候拥有两个主程序，一个是LFTPSever运行服务端程序，LFTPClient运行客户端程序。服务端程序既可以接收客户端上传的文件，又可以向客户端发送文件。

1. **项目设计**

5.1 报文设计

我们使用一个报文类来处理段的拼接和获取信息。根据模仿TCP协议，我们把报文分成了(依次) ：ACK（4byte） , SYN（4byte），FIN（4byte），SEQ（4byte），RECW（4byte），blanket1（4byte），DataLength（4byte）, data（1000 byte）

对于数据段data，如果其不满1000byte，则补空字节（00000000） 一般来说，只有最后一个包可能出现字节数不满1000byte的情况

5.2 本地文件读取与文件接收后的拼接

这一部分需要考虑两个问题：分成两个模块：读包和写包

5.2.1读包

5.2.1.1 读入操作的内存效率问题

我们不能把整个文件读入再分包，这样对发送端的内存消耗是惊人的，即需要实现动态读入。

5.2.1.2 重传问题

在TCP的发送过程中，时常会出现重传，而只使用动态读入机制会导致读头后移，如果你要重传一个包，则需要传输读头之前的包只能重新读入包并把读头移动到对应位置，这样会导致大量的读写开销（浪费）。

为了解决这个问题，我们选择维护一个读入类，类内有读入缓存。读入类是TCP发送端的一个成员。当TCP发送端想发送一个包时，他可以输入seq从读入类中获取动态读出的包。需要注意的是，这个seq必须大于目前收到的ack，且小于已经发送过的seq最大的包的包号+1。当TCP接收端想要发送一个从未发送过的新包时，读入类读入下一段内容、将其存入缓存并且返回这个内容。当TCP接收端收到了一个ack时，他会操控读入类释放掉seq小于此ack的所有包的缓存。

5.2.2写包

5.2.2.1 写包的缓存机制

这里主要是要考虑后包先到的问题。假如包以3 2 1的顺序到达，如果直接舍弃掉前面两个包就比较浪费性能，所以写入类的缓存会先把3 和 2 存起来，1到达时开始写入，可以次性写完三个包。这样就不会浪费性能了。通过对写入方法的封装，TCP接收端可以简单地将包传输给给写入类，写入类自己思考什么时候应该写入，什么时候不该写入。

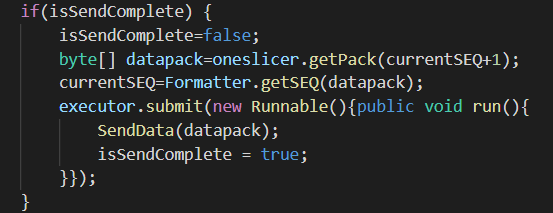
5.3 实现类似TCP的100%可靠传输

这个部分可以根据书上的实现TCP可靠传输的伪代码进行构思，分成为发送方和接收方。

5.3.1 发送数据方

5.3.1.1 从上层应用接收数据

从slicer类中取得要传的序号的报文，slicer类已经封装好了将数据加上报文头变成报文的函数，所以只需传入要传包的序号就可以获得报文。获取数据后，将下一个要发送的序号变成当前发送序号加1，这时如果发送方计时器没有启动的话，则需要启动计时器。下面是发送方执行发送包的指令：



上面使用了线程，是为了让发送数据和接收ACK独立执行，并且用了一个锁，让发送数据能够顺序执行。

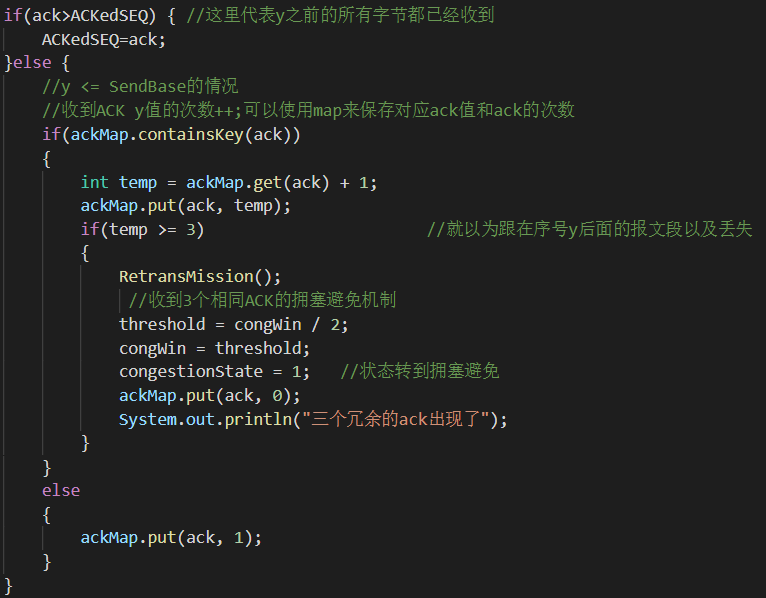
5.3.1.2 超时

在发送数据后开启的计时器，计时的时间到了的话会重新传送最早未被确认的报文。倒计时的时间和RTT有关，在多次测试后决定使用100ms作为初始的超时时间。使用了加倍超时时间机制，如果上一次重传的包还是这一次重传包的序号则让超时时间变成之前的两倍。下面是发送方的超时执行的指令：



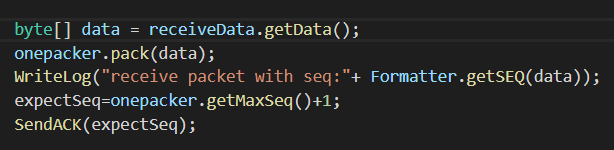
5.3.1.3 接收ACK

作为发送方需要接收从发送方发送回的ACK值，假设ACK的值大于最早确认收到的序号，代表该值之前的所有字节都收到了，并且设置最早确认收到的序号为该ACK值，如果之后还有未确认的报文段，需要重启定时器。如果接收到3个重复的ACK值则需要重新发送该ACK值的包，这是快速重传机制，接收到3个重复的ACK则默认表示超时了。下面是接收到ACK值的部分代码：



5.3.2 接收数据方

接收数据方ACK的值是希望下一次收到的包的序号，如果接收数据方收到一个报文，并且该报文的序号等于所期望收到的序号则期望序号增加，并且ACK增加后的期望序号，如果不等于则期望序号值不变，依旧ACK该值。这里是使用pakcer类封装了这个功能，因为接收数据方也需要一个缓存去存储接收到的包，它会返回按照顺序拼接包，所拼接到的最大序号，下面是接收数据方的代码：

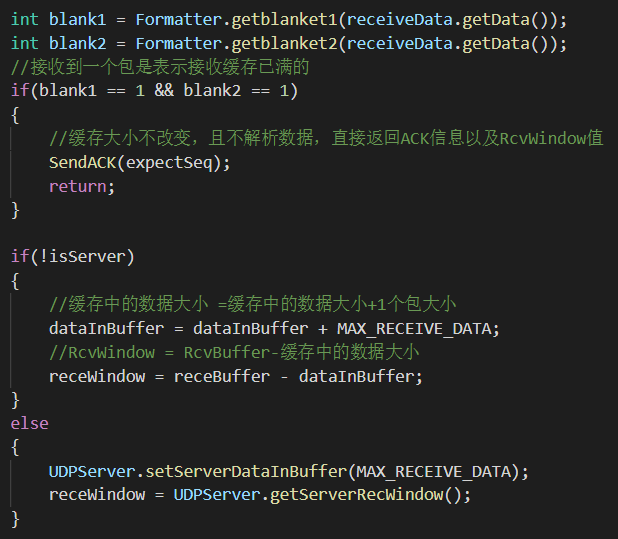


5.4 流量控制设计

流量控制主要用来消除发送方使接收方缓存溢出的可能性，流量控制也分为发送数据方和接收数据方

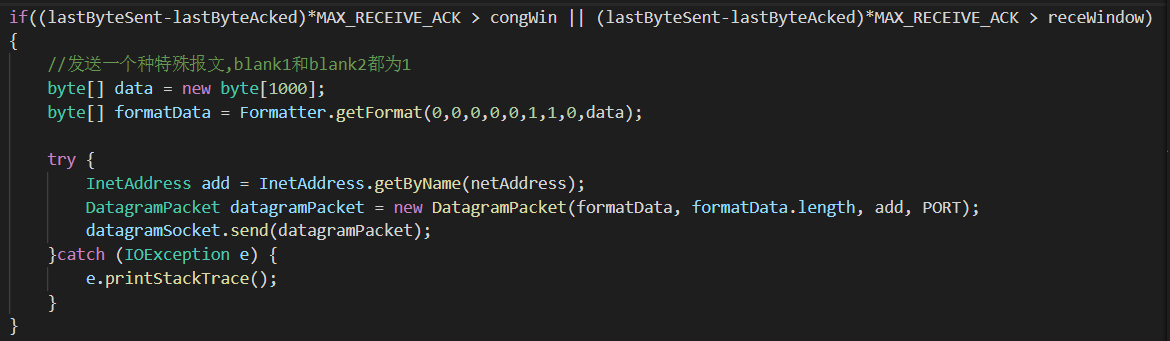
5.4.1 接收数据方

给接收数据方安排一个接收缓存，接收缓存的大小初始的时候是十个报文的大小，然后接收窗口的大小是接收缓存的初始大小。当接收方接收到一个包的时候，如果这个包不是因为缓存满了发送方发出询问接收窗口是否空的报文，则缓存中的大小就增加一个报文的大小。如果是询问接收窗口的报文直接就返回ACK信息和带有接收窗口的值。在缓存中数据被读取后，接收窗口的值会增大。作为服务器的话，服务器只有一个，所以它的接收缓存是公用的所以要单独设置服务器的接收缓存，下面是部分代码：



5.4.2 发送数据方

数据发送方接收到接收方发送的ACK报文，将里面的接收窗口的值提出，如果接收窗口为0，则就不发送之后的报文，但需要发送询问报文去询问数据接收方的接收窗口大小，这样避免了接收方有接收缓存的时候发送数据方不知道的情况。下面是发送方的代码：

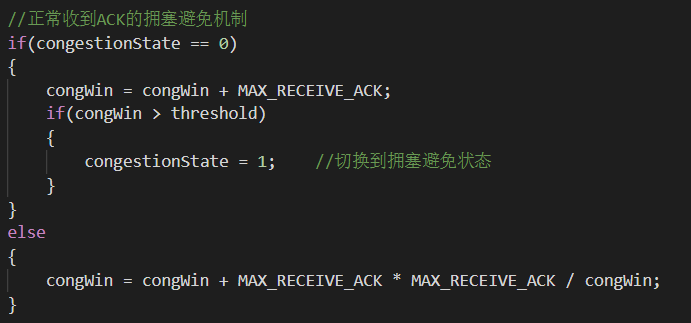


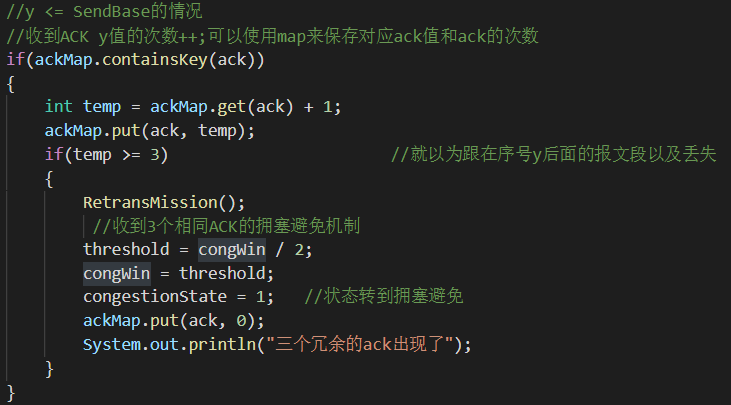
5.5 拥塞控制设计

发送方可能因为IP网络的拥塞而限制了速度，拥塞控制让发送方可以感受到网络拥塞。让连接的每一端都记录一个额外的变量拥塞窗口。这一机制主要分为两种状态，慢启动状态和拥塞避免状态。

5.5.1 慢启动状态

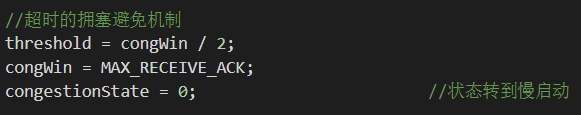
发送方维护一个拥塞窗口，初始的时候值为1个报文的最大长度，还有一个阈值初始化的时候比较大。在慢启动阶段，发送方的拥塞窗口大小是指数增长的，如果增长到大于阈值的时候，就会切换到拥塞避免状态。或者受到3个冗余的ACK的时候，阈值会变成拥塞窗口的一半，拥塞窗口变成当前的阈值，并且状态转到拥塞避免状态。如果遇到发送的包超时的话，阈值也会变成拥塞窗口大小的一半，但是拥塞窗口会变成1个报文的长度，状态又从慢启动开始。慢启动的部分代码如下：





5.5.2 拥塞避免状态

在拥塞避免状态下如果收到前面未确认数据的ACK是加性增，不是指数型增长，如果收到3个冗余的ACK或者是发送的包超时了和慢启动状态处理一样。拥塞窗口的值在发送数据的时候判断，判断最后一个发送的报文和最后一个确认收到的报文的差值是否大于拥塞窗口。如果大于的话发送的报文和流量控制的报文一样。



5.6 发送数据和接收数据的异步执行

因为JAVA是一个顺序执行的语言，所以要实现发送数据和接收数据的异步执行，需要加入新的线程。而且要保持线程一直在执行。对于发送数据方来说，发送数据和接收ACK数据要独立，所以我在一个while循环中让他们独立线程执行，在发送完数据之后跳出while循环。并且它们在独立线程中的执行时顺序的，所以需要使用锁让其顺序执行。

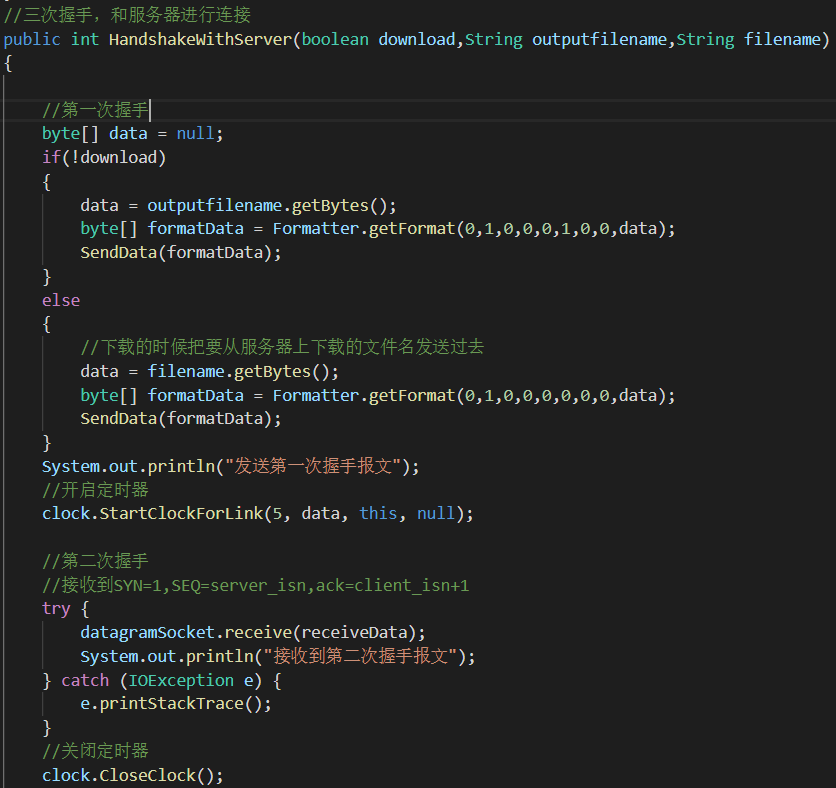


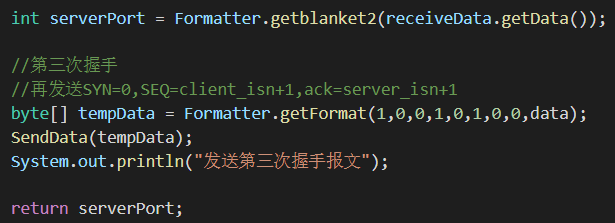
5.7 上传与下载

服务器如何知道是上传文件还是下载文件，并且如何知道上传或下载的文件是的路径呢，这些信息我在三次握手的时候发送给了服务端。那又是怎样知道发送完或接收完数据了，这就需要四次挥手机制了。

5.7.1 三次握手

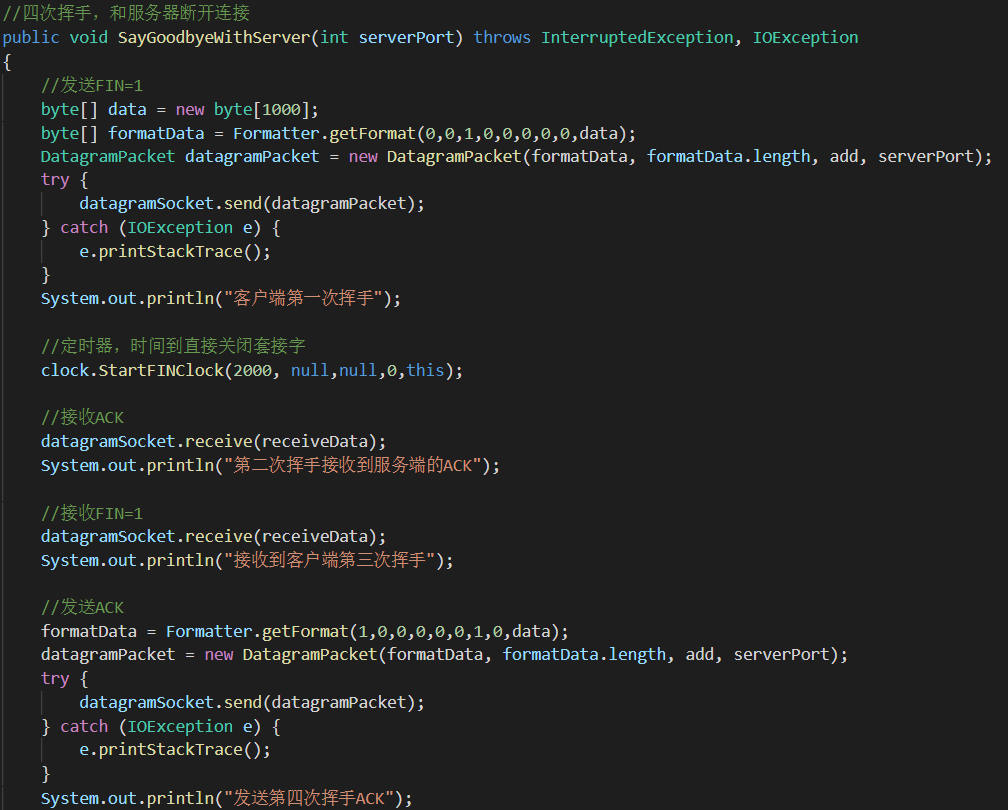
这里客户端无论是上传还是下载文件都需要和服务器三次握手，在第一次握手的时候我将文件名作为数据发送给服务器，并且用一个未被使用的报头位来标识是上传还是下载文件，这样服务器能根据请求执行响应操作。在握手的时候也可能会发生丢包事件，所以在第一次握手的时候也需要启动定时器。在握手过程是顺序执行的，意味着客户端只有握手完成才能上传或者下载数据。





5.7.2 四次挥手

在发送数据方发送完数据之后会主动向接收数据方进行断开连接操作，因为我是要确保读取完文件最后一个传输的序号被接收方确认后才会挥手，所以整个过程只用发送挥手消息，然后开启计时器，如果计时器结束则直接就断开连接。



5.8 一个服务器支持多个客户端

一个服务器需要支持多个客户端的请求，我设置了一个总的服务器套接字端口，专门接收握手报文，有握手报文就意味着有新的客户端请求，于是在握手过程中，创建一个新的套接字端口，并在第二次握手的时候将端口号发送给客户端，客户端接收到该端口号之后的上传或者下载工作都是由该端口号负责。在客户端的任务完成之后，负责该客户端的服务器端口号关闭，我使用了一个map去存储对应端口与其对应的套接字，只要知道哪个端口号的任务完成，就去关闭哪个套接字。端口号的分配是按顺序分配的。部分代码如下：



可以看到接收数据或者发送数据都是开了一个新线程去进行，所以不同客户端之间的传输工作是异步执行的。

1. **测试样例**

6.1 一个服务端和一个客户端

6.1.1 上传1G数据到服务端

6.1.1.1 数据完整性

6.1.1.2 Wireshark分析

6.1.1.3 程序运行截图

6.1.2 从服务端下载1G数据

6.2 一个服务端和多个客户端

6.2.1 同时上传数据

6.2.2 同时下载数据

6.2.3 既有上传又有下载

6.3 流量控制测试

6.4 拥塞控制测试