**1计算机网络大作业报告**

**学号： 20110031020 姓名： 许健 专业： 计算机科学与技术 年级： 20级**

1. 结合代码和LOG文件分析针对每个项目举例说明解决效果。（17分）

说明：目标版本号及得分对照

5.1 (Reno)：17

5.0 (Tahoe)：15

4.0 SR/GB/TCP：12

3.0：9

2.2: 6

2.0: 3

1. 未完全完成的项目，说明完成中遇到的关键困难，以及可能的解决方式。（2分）
2. 说明在实验过程中采用迭代开发的优点或问题。(优点或问题合理：1分)
3. 总结完成大作业过程中已经解决的主要问题和自己采取的相应解决方法(1分)
4. 对于实验系统提出问题或建议(1分)

实验内容：

一.I）RDT1.0

目的：熟悉程序

程序由四部分组成：

1. checkSum:

函数computeChkSum，根据发来的包进行校验，在RDT1.0中，不做任何处理，返回0.

1. TCP\_Sender:

1)rdt\_sender:

1]设置TCP数据报的seq,data和checkNum

2]调用udt\_send发送数据报

3]设置flag为0，直到flag不为0时才退出程序

2)udt\_send:

1]设置错误控制标志为0

2]发送数据报

3)recv:

1]将受到的ack包添加到队列中

2]调用waitACK处理ack包

4)waitACK:

检查队列中的确认信息，如果ack号和刚发出的seq号相 同，那么flag置1，可以接着发送下一个tcp包；否则再次发送此次的tcp包，flag置0。

1. TCP\_Receiver:
2. rdt\_recv:

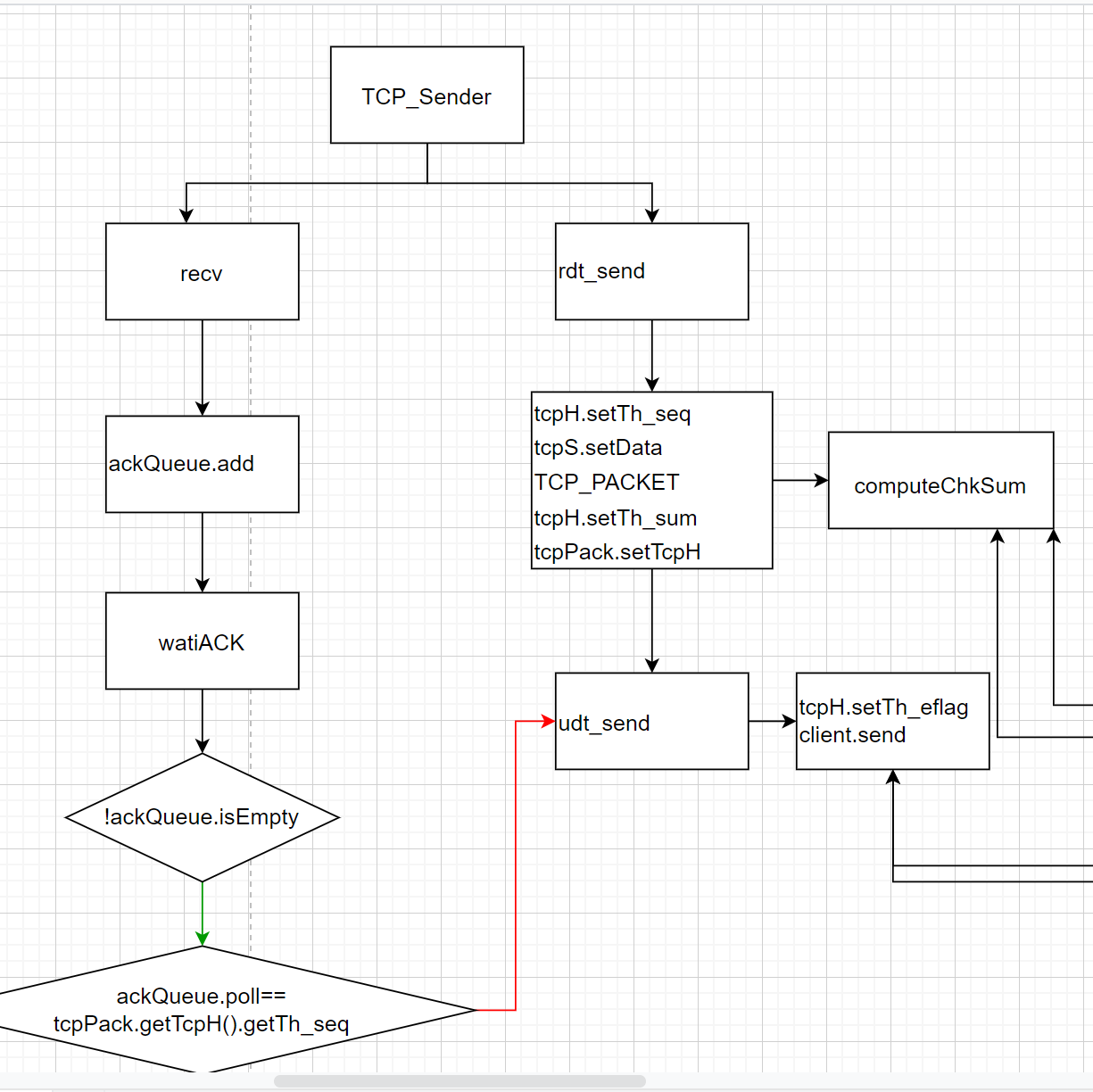
1]进行检验，若正确返回ack确认包，并将其加入到序列中；否则返回ack为-1的确认包

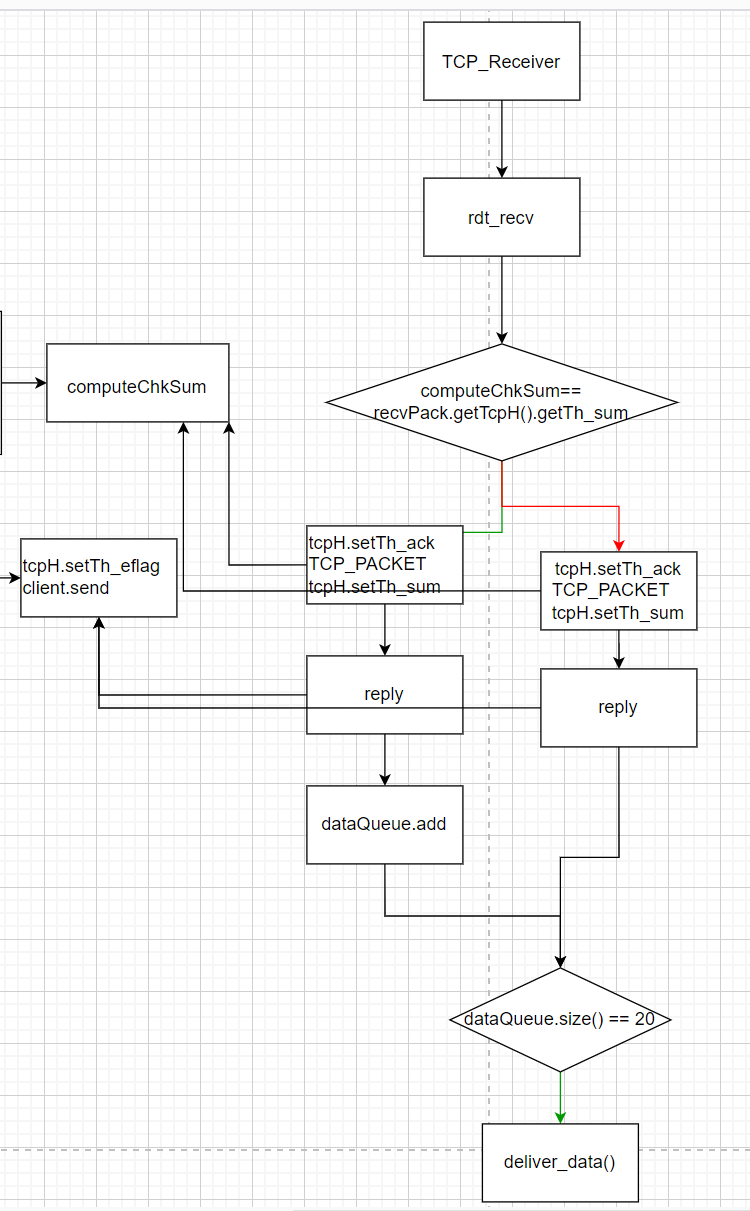
2]若队列满20，那么将数据传送到文件当中

2）deliver\_data:将队列中的数据写入文件中

3）reply:设置错误标志为0，发送确认包

4.调用与组成关系：

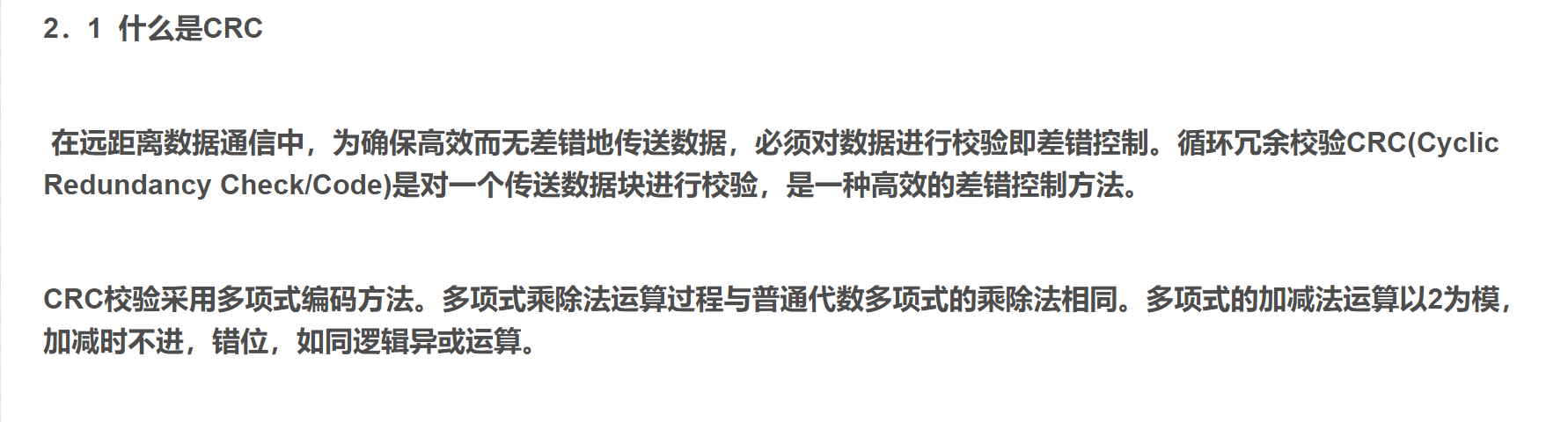




II)RDT2.0：

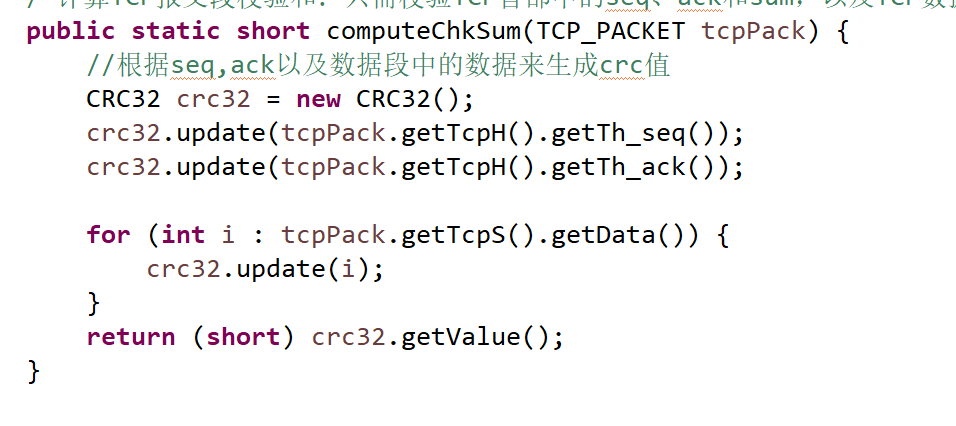
1.目标:完成校验函数。

2.方法:可以调用java中的CRC32类完成

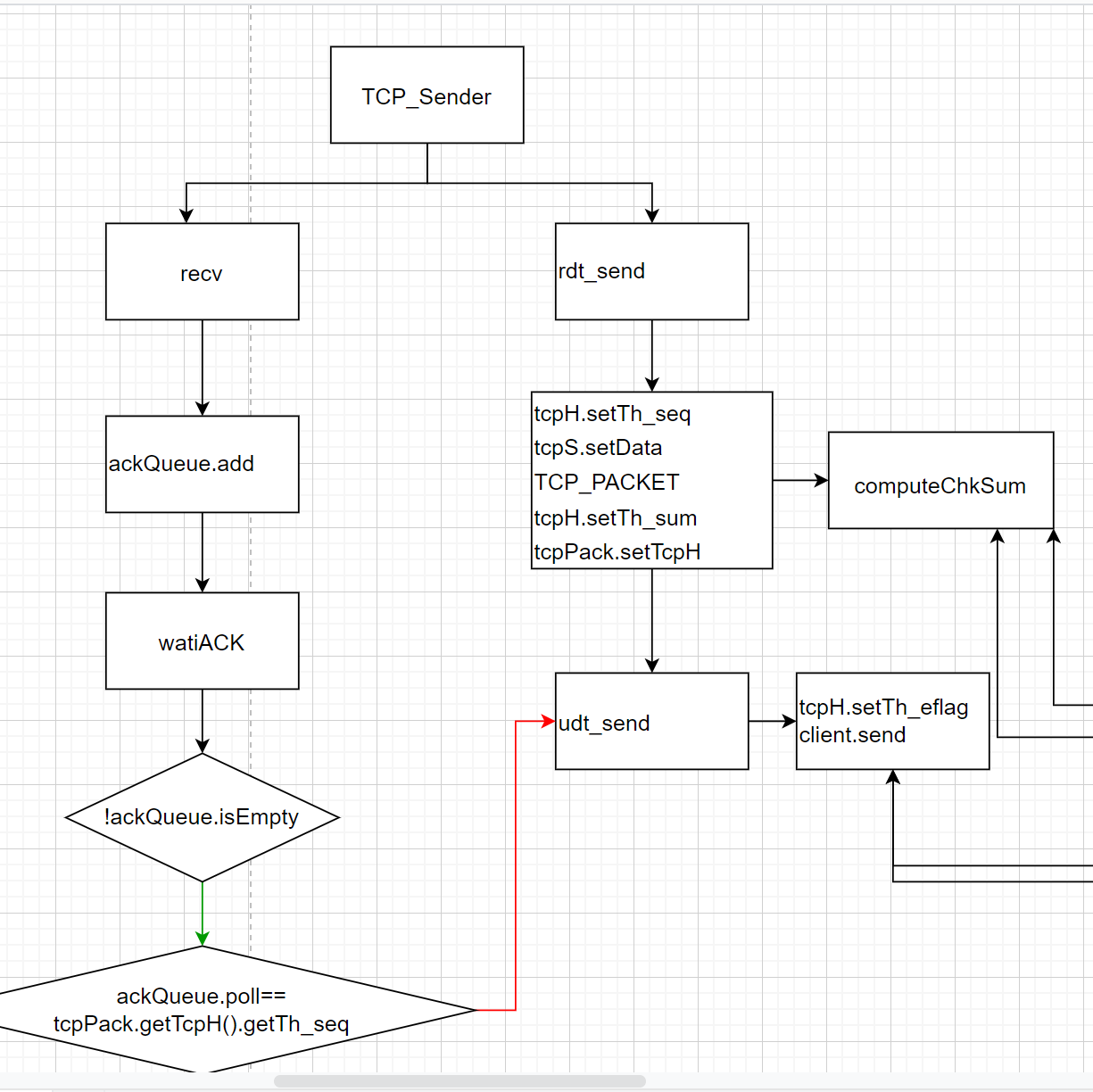


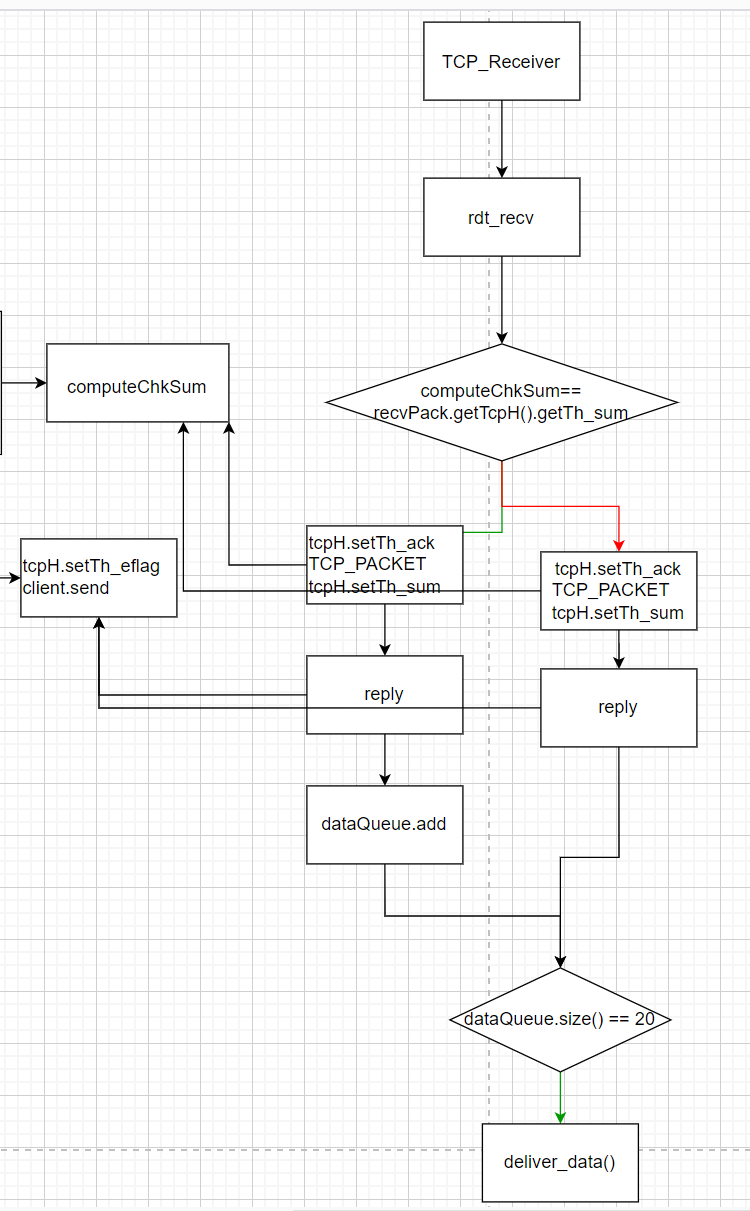
注意：不要忘记将发送端的信道标志设为1（允许出错）

3.函数补充：



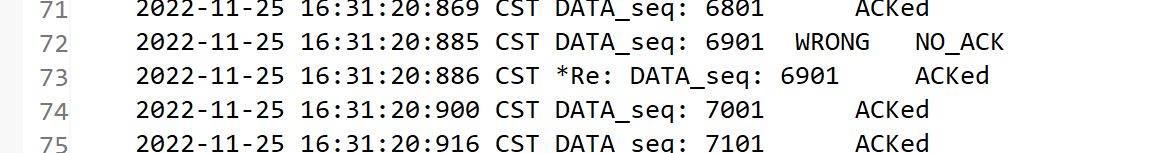
4.组成与调用关系：



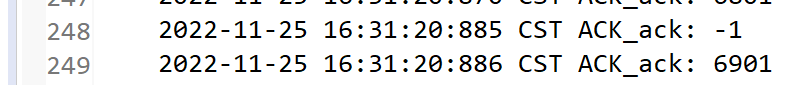


5.Log分析：

发送端在seq6901处出错，并重发



接收端在本是6901处发送了-1，说明校验函数有效



III)RDT2.1

1.要解决的问题：

发送方可能会接收一个错误的ACK，重发，导致数据包重复，所以接收方要具备丢弃相同分组的能力；

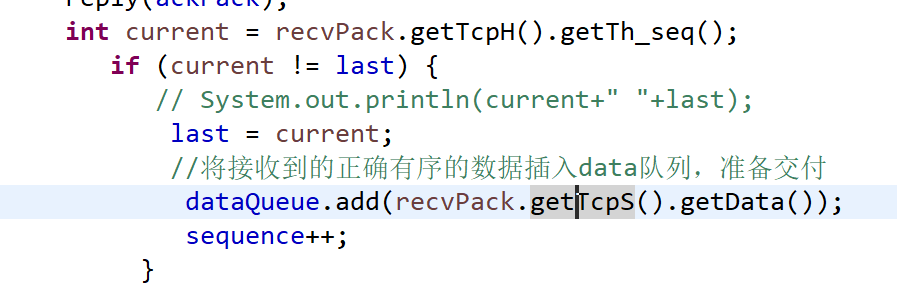
发送方所接收ack时还要进行校验码判断。

2.解决方法：

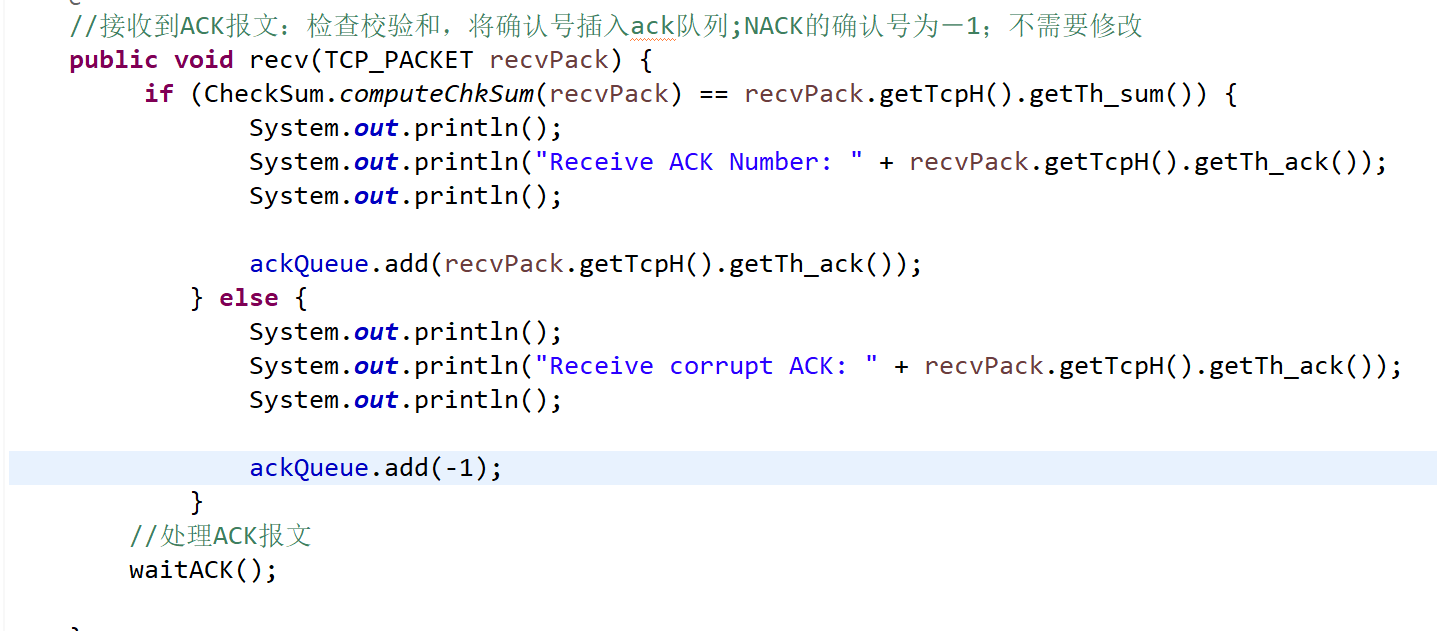
发送方重发肯定会重发上一次的包，所以让记录下上一次的seq和这一次的seq相对比就可以了，如果相同，那么就不进入队列。

3.程序代码：

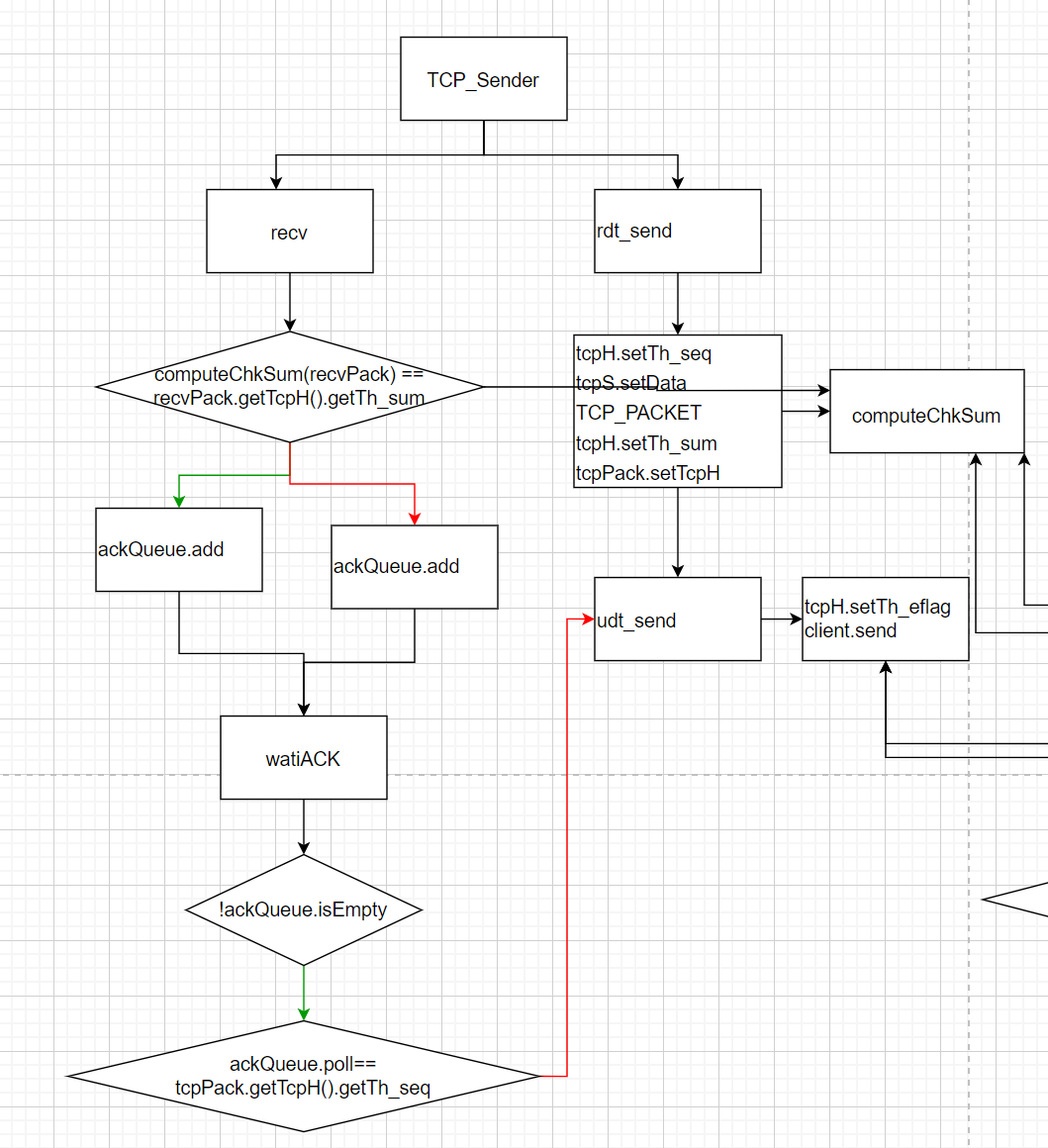
1）判断是否有重复的包

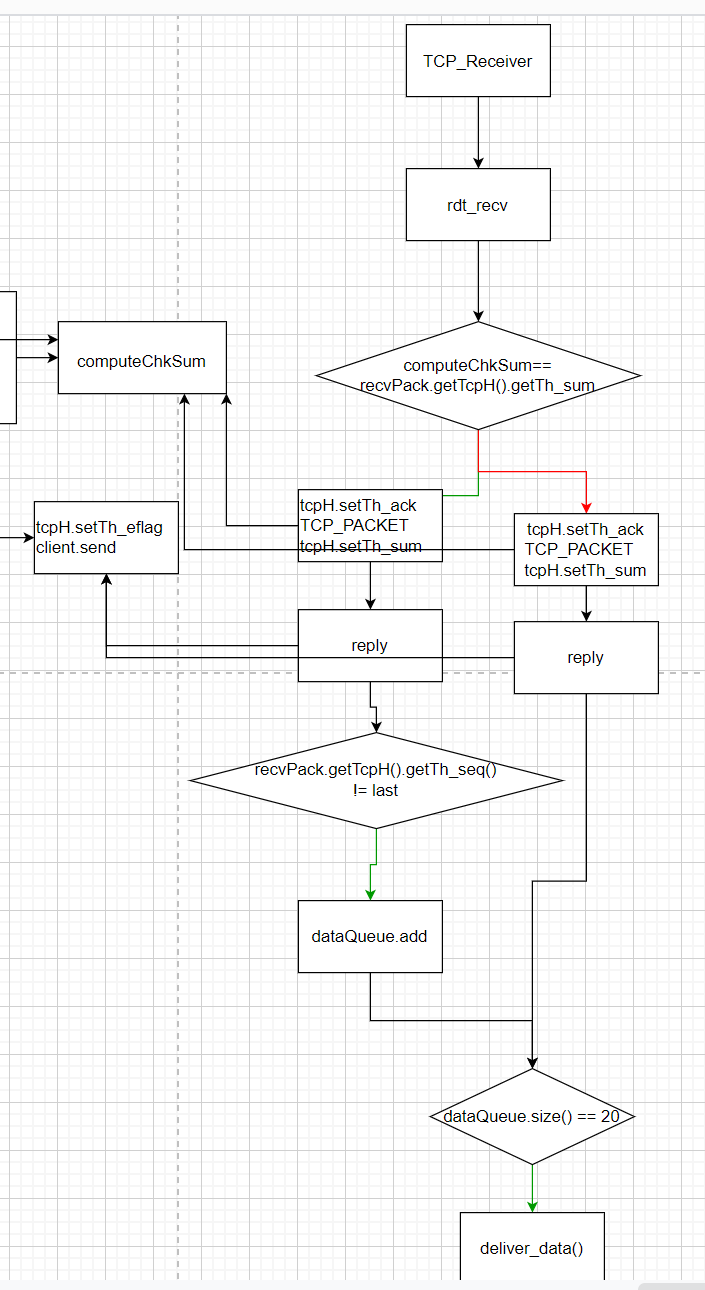


2）发送端对ack的校验码进行检验。



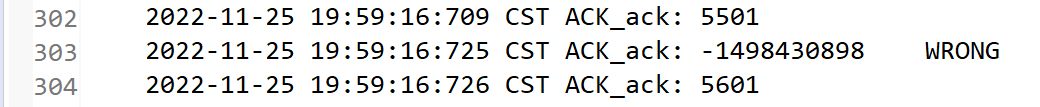
4.组成与调用关系：



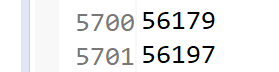


5.检验：

这是接收端第一次出错，那么发送端会发送两次5601，如果没有舍弃包，那么recvData中标号为5601和标号为5701两处的数据就会相同，也可以说标号为5700的数据要大于5701的数据。



经检查，标号5700和5701两处的数据大小顺序是从小到大的，说明重复的数据被抛弃掉，所以程序代码有效。

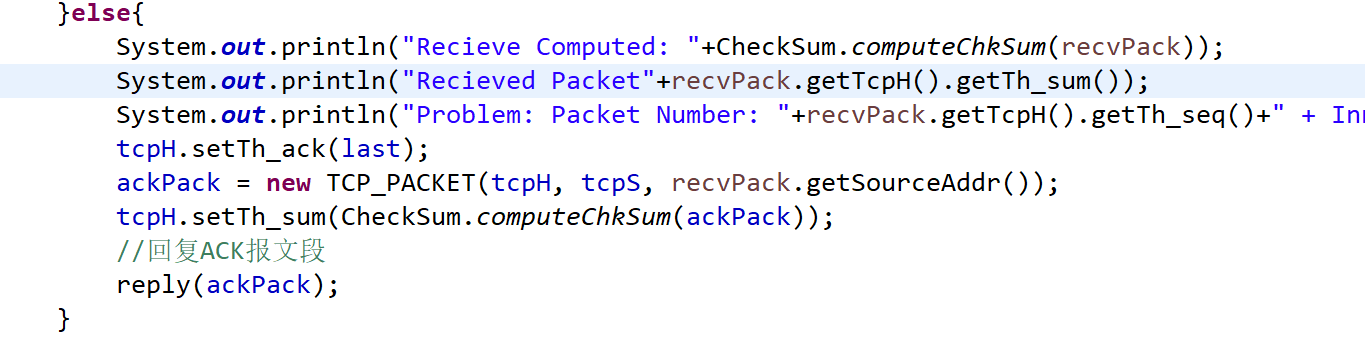


IV)RDT2.2

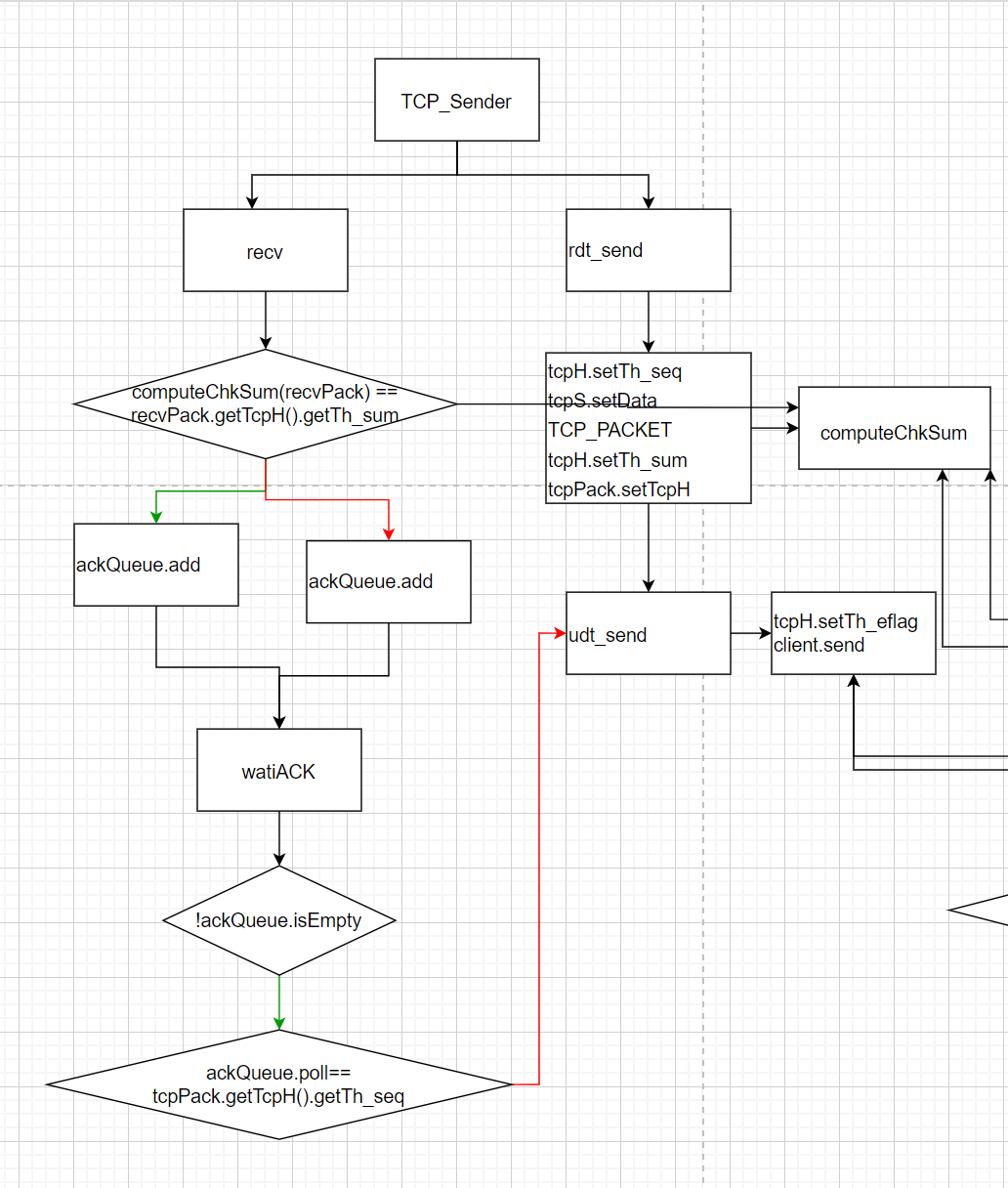
1.要解决的问题：接收端不再发送ack为-1的包，如果接收到的包校验码错误，那么接收端会发送上一次的ack，发送端没有接收到此次的ack，那么就会重发。

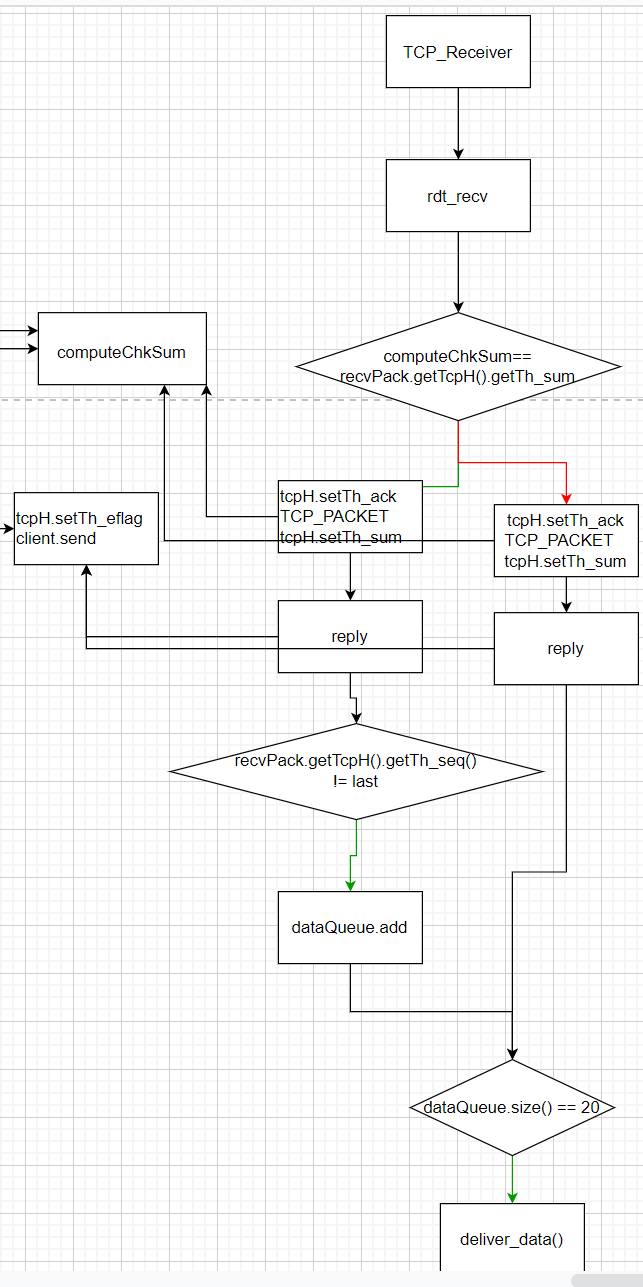
2.解决方法：如果接收端接收的包校验失败，那么回复上一次的ack即可。

3.代码：



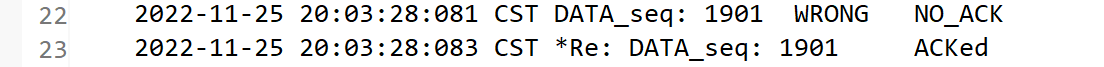
4.组成与调用关系：



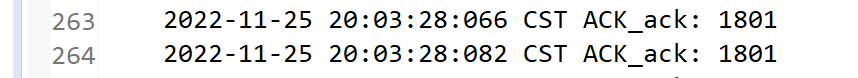


5.检验：

可知，在1901处发送端发送失败，那么接收端会出现两个1801



由下图可知代码正确



V)RDT3.0

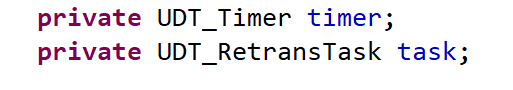
1.面临问题：在包和ack传输的过程中可能会面临丢包，延迟的情况。

2.解决方法：在发送端设置一个计时器，如果包发送完一段时间没有接收到ack，那么就会重发，直到正确的ack到达才会停止计时任务。

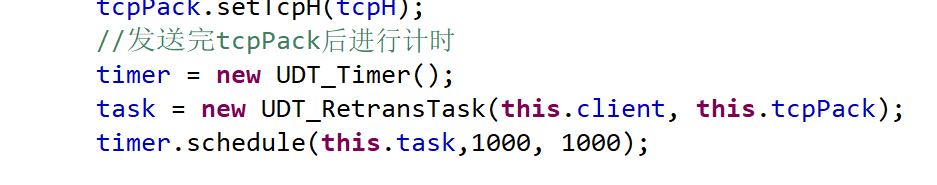
1. 假如发送端或者接收端丢包，这样都会导致包发送完一段时间后没有正确的ack到达，从而继续发送此包，方法可行；
2. 假如发送端或者接收端延迟，那么也会导致发送端在限定的时间内得不到想要的ack，那么就会重复发包；重复的包会被接收端抛弃掉，方法可行；
3. 假如发送端或者接收端出错，那么都会导致发送端拿不到正确的ack从而重新发包，如果让计时器单独去完成这项任务时间上会延长，也可以使用原有的逻辑判断，这样时间会缩短。

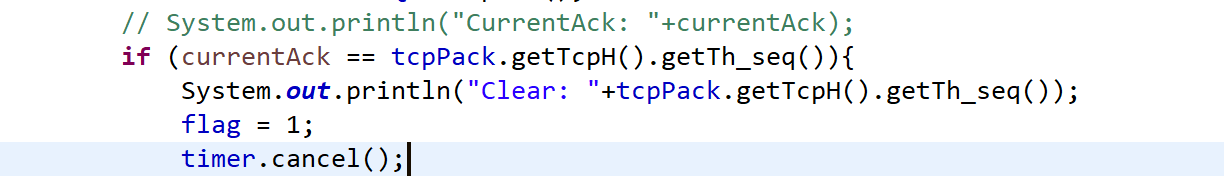
3.计时器的代码：

声明：

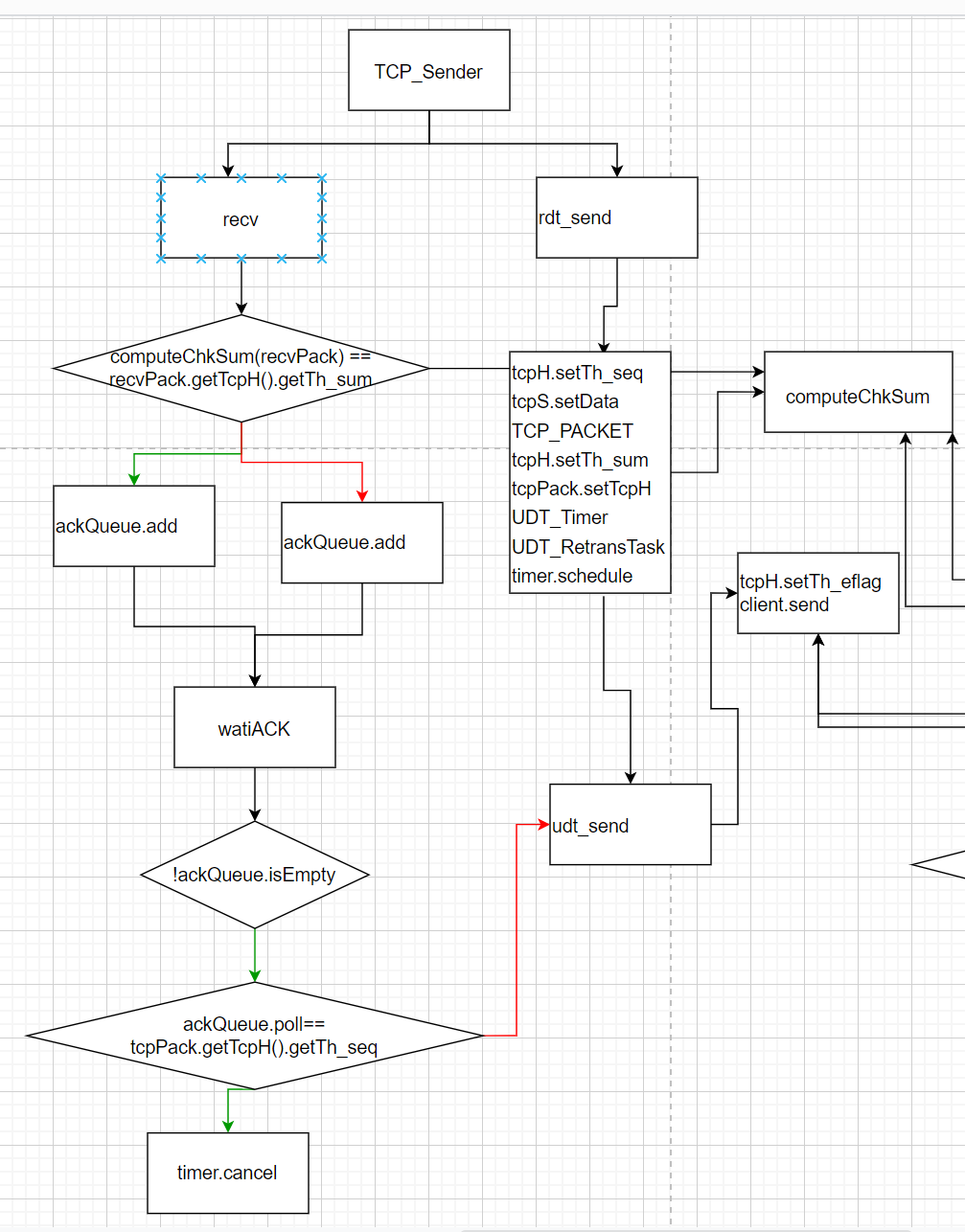


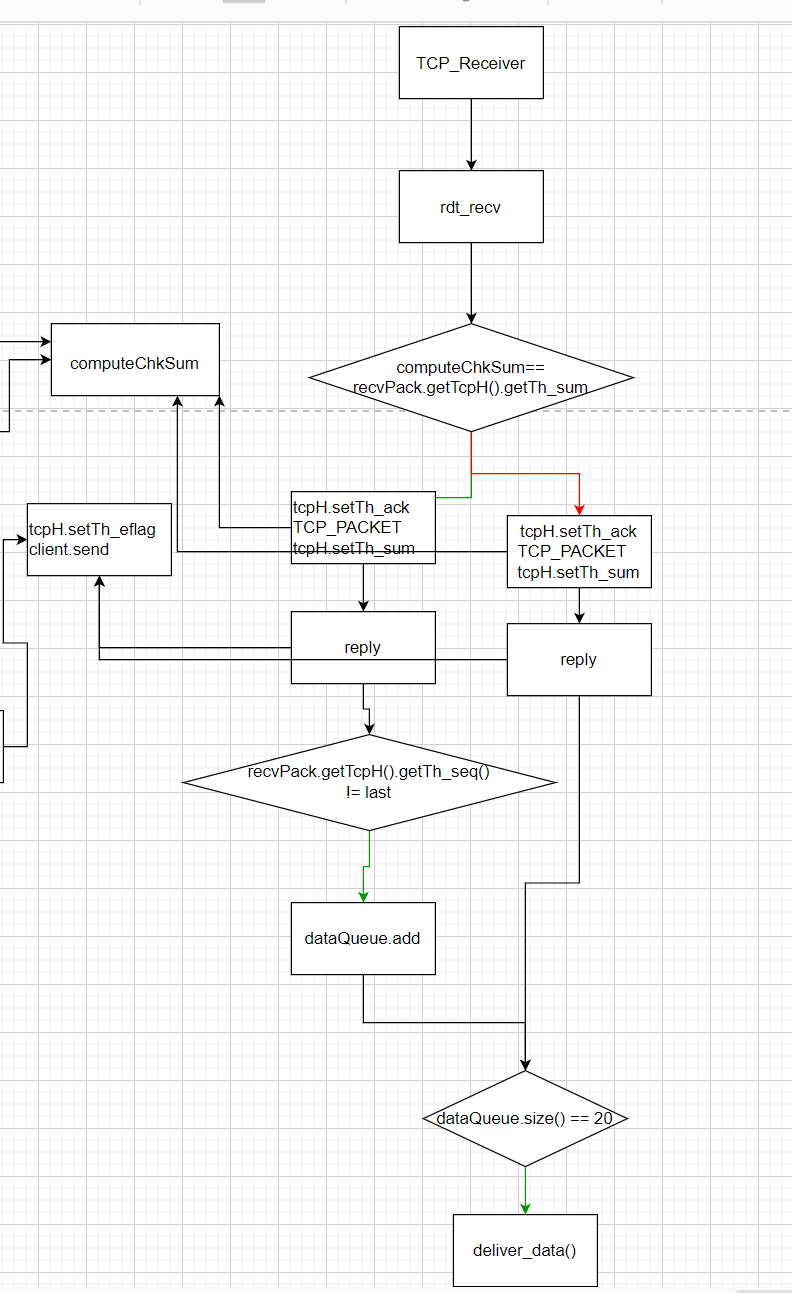
设置:



取消：  


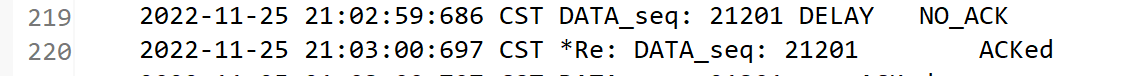
4.组成与调用关系：



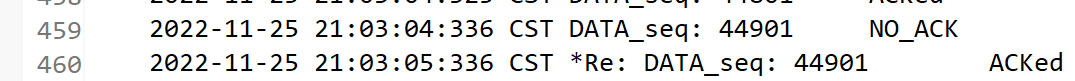


5.检验：

可以看到当发送端延迟时，隔1s后进行了重发



当接收端延迟或丢包后，发送端也会隔1s进行重发



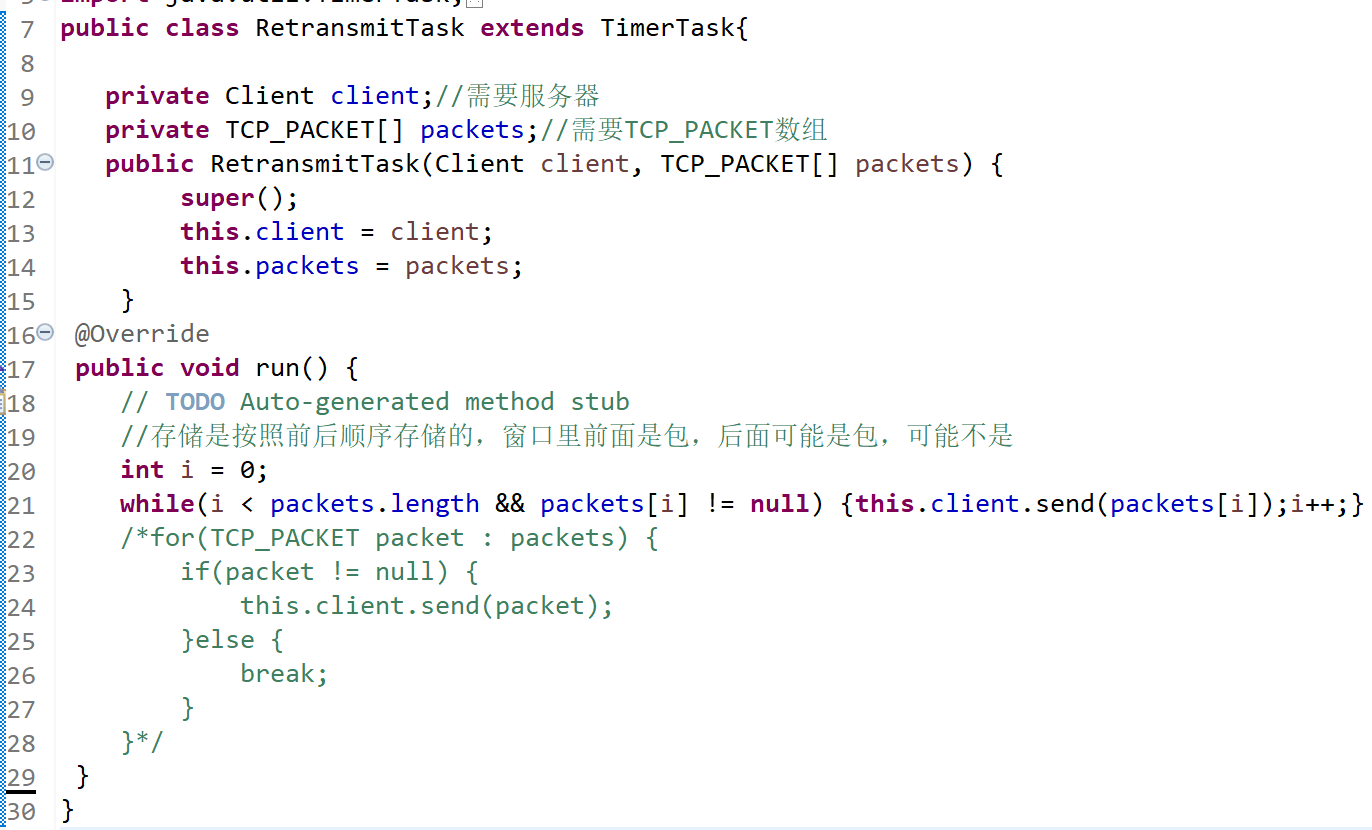
VI)RDT4.0 GBN

实现：

1. 继承重写TimerTask类

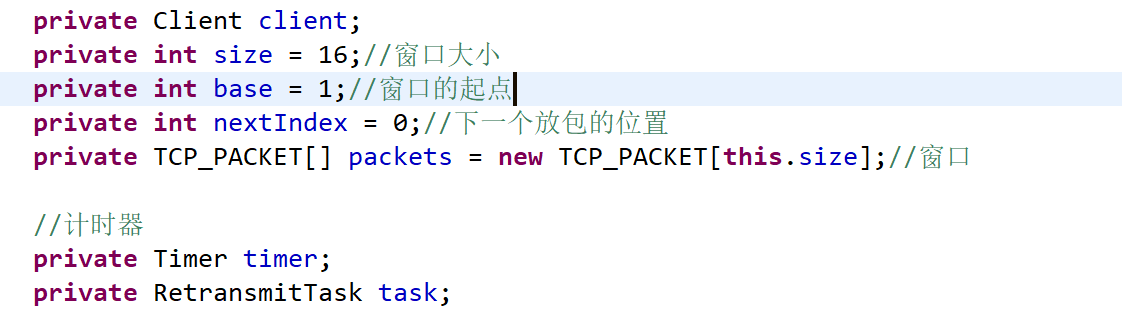
我们可以学习UDT\_RetransmitTask对TimerTask类的重写继承方法，来进行我们对TimerTask的继承与重写。

1. 增加Client和TCP\_PACKET[]成员
2. 重写run方法，实现TCP\_PACKET[]里TCP\_PACKET的重发。



1. 实现滑动窗口类

1)成员:

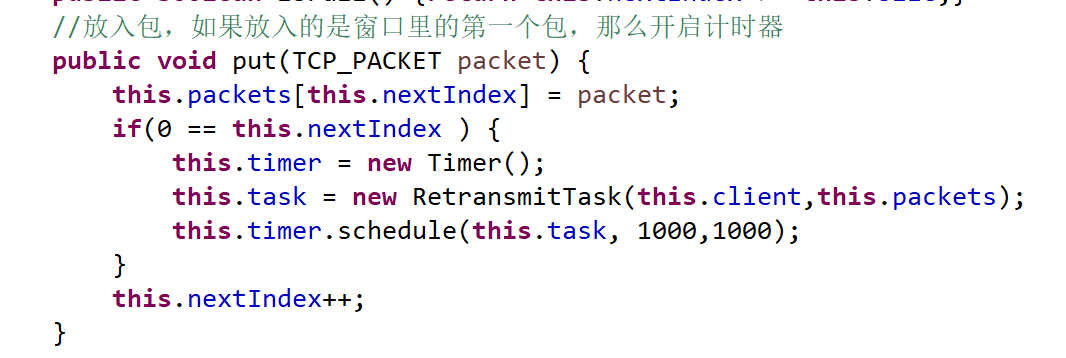


初始化时机:

Client通过构造函数在外部调用时进行初始化；

Timer和task在已发包存入窗口时进行初始化

2)函数：



将已发但未被确认的包放入滑动窗口内，为窗口内的第一个包设置计时器。



1]首先只会接收对滑动窗口内tcp包的ack确认

2]其次会将接收ack的包以及之前的包从窗口内清除掉，可以理解为窗口右向右滑动，窗口左边沿对应着接收到的ack右边的包，而nextIndex对应的位置则会向左移动相同的大小。

3]既然最左边的包已经不在窗口内，那么此包对应的计时任务就会被取消，并为新的左沿包重新设置一个计时任务。

3.发送端的改变:

1)

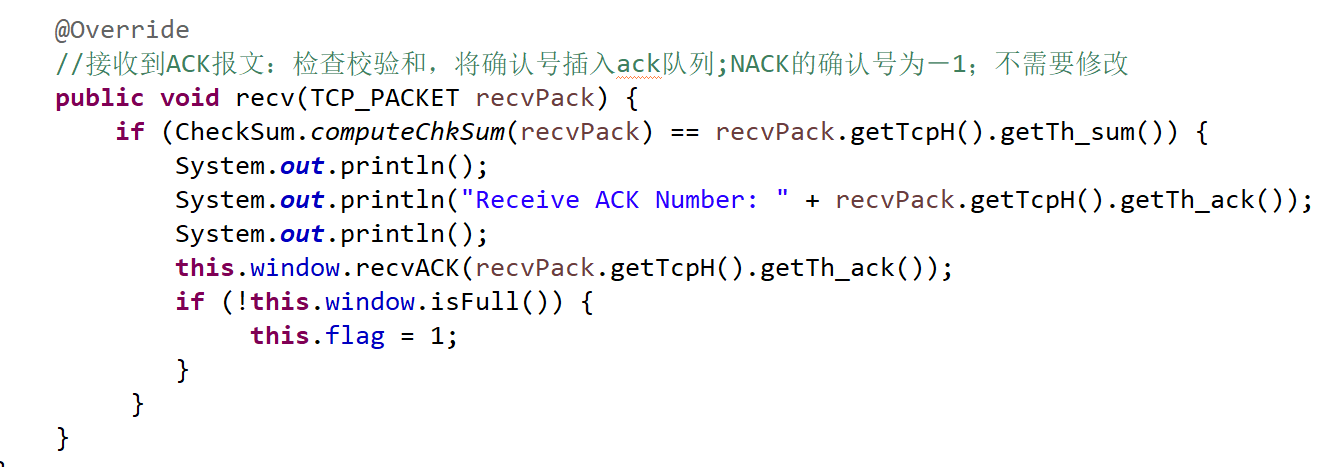


注意：

1]窗口已满自然不能够再发送，因为多余的tcp包将无处可存。

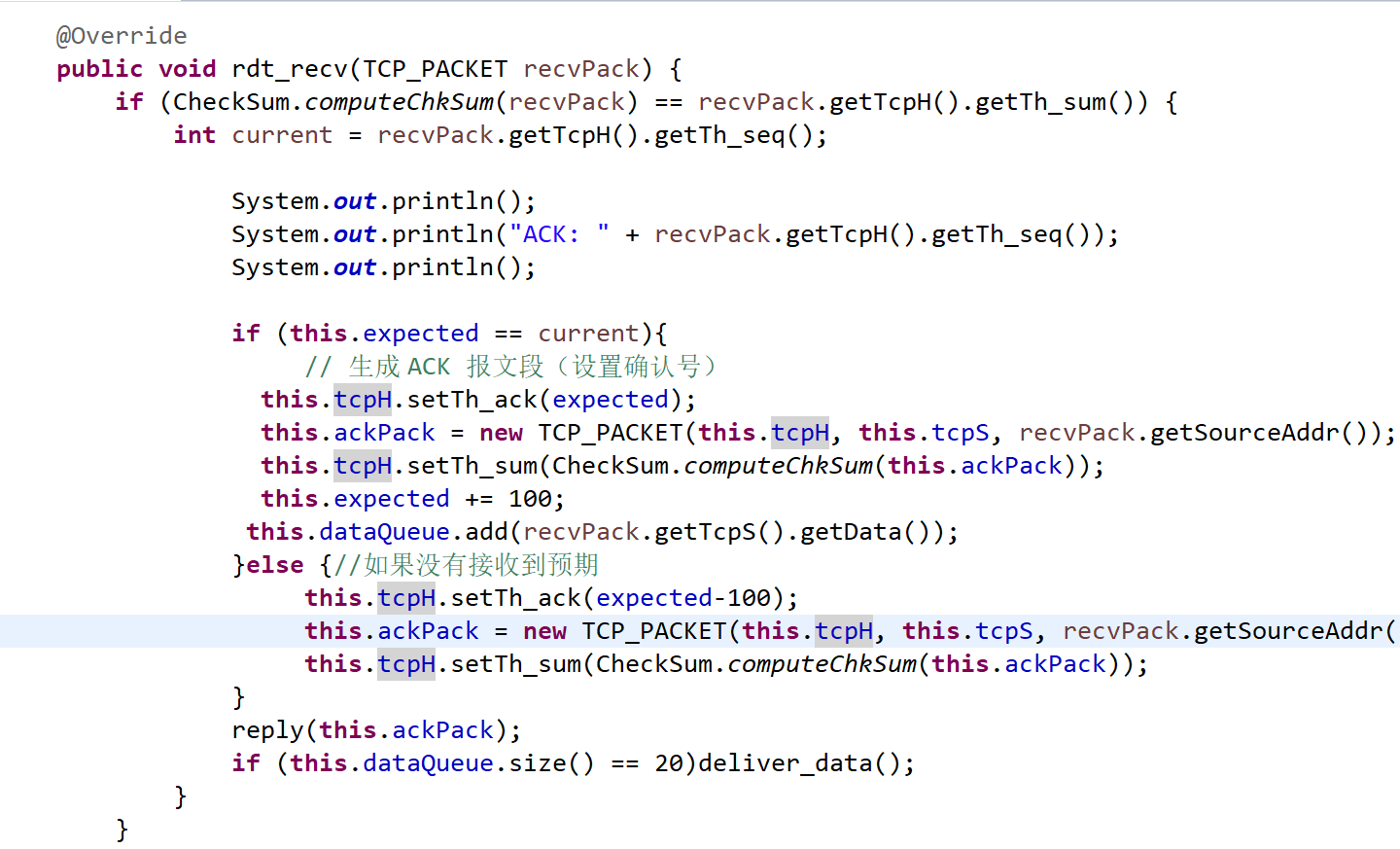
2]udt\_send():无论是否有位错，丢包与延迟，每个包都只会调用一次udt。

2)



接收到ack之后一般情况下窗口都会右移，从而为接下来要发送的tcp腾出空间。

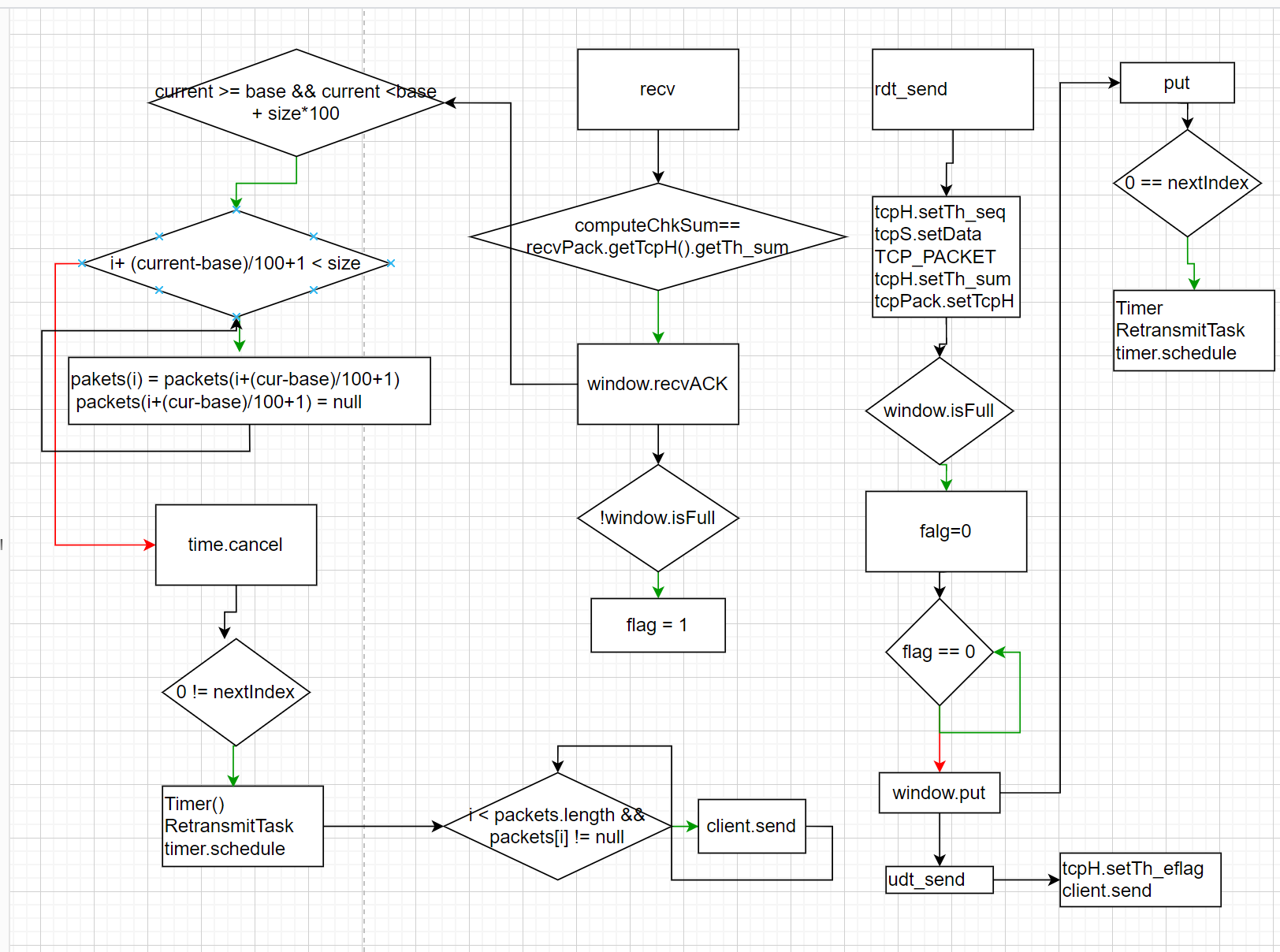
4.接收端的改变:

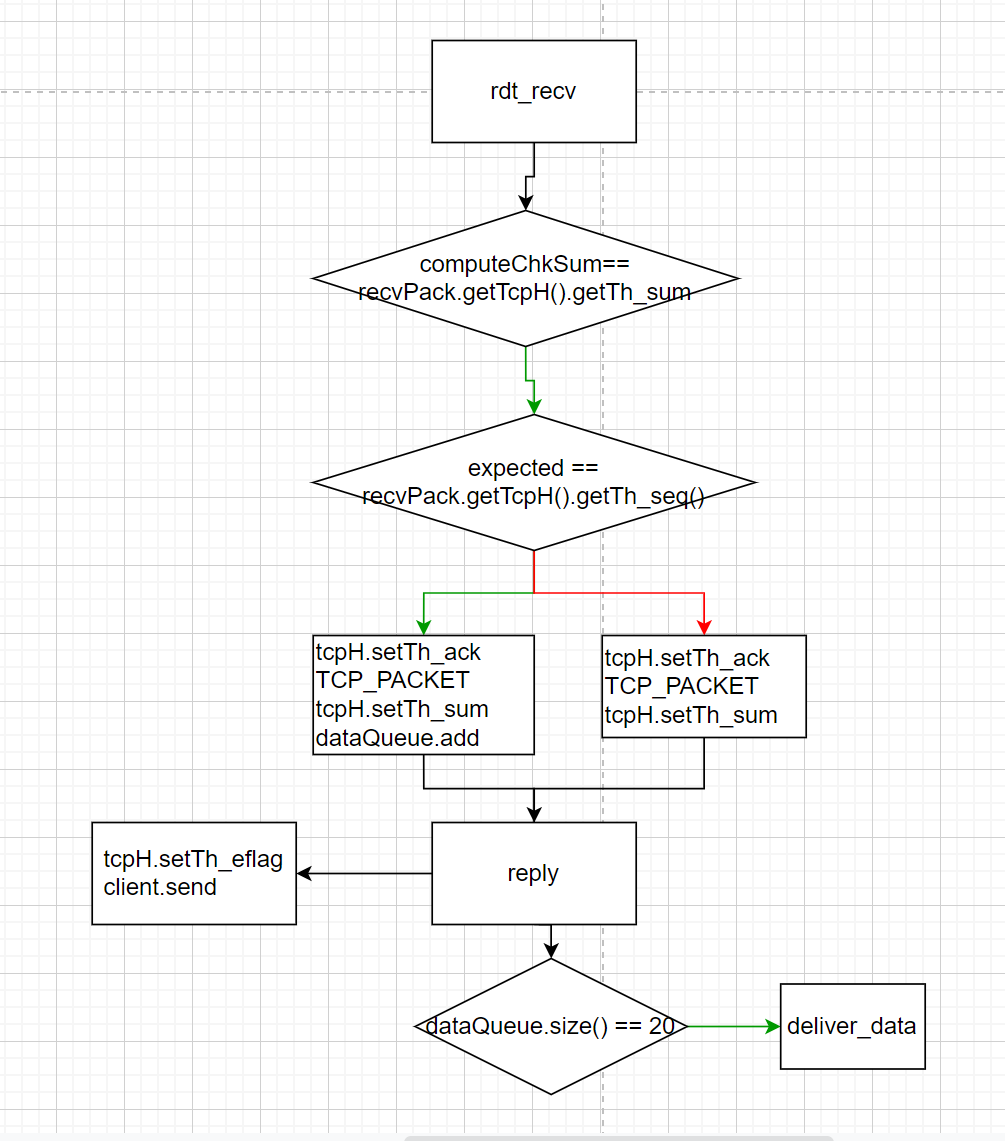


设置预期值，如果接收到了预期的包，那么预期值更新，返回带有预期值的ack；否则ack是上一次的预期值。

运行预测:当发送端的包位错，丢失，延迟时，接收端会返回上一次的ack，从而导致发送端重发一组包；但当接收端的单个包出现位错，丢失，延迟，而下一个包是正确的时候，由于累计确认，发送端不会重发。

调用与组成关系

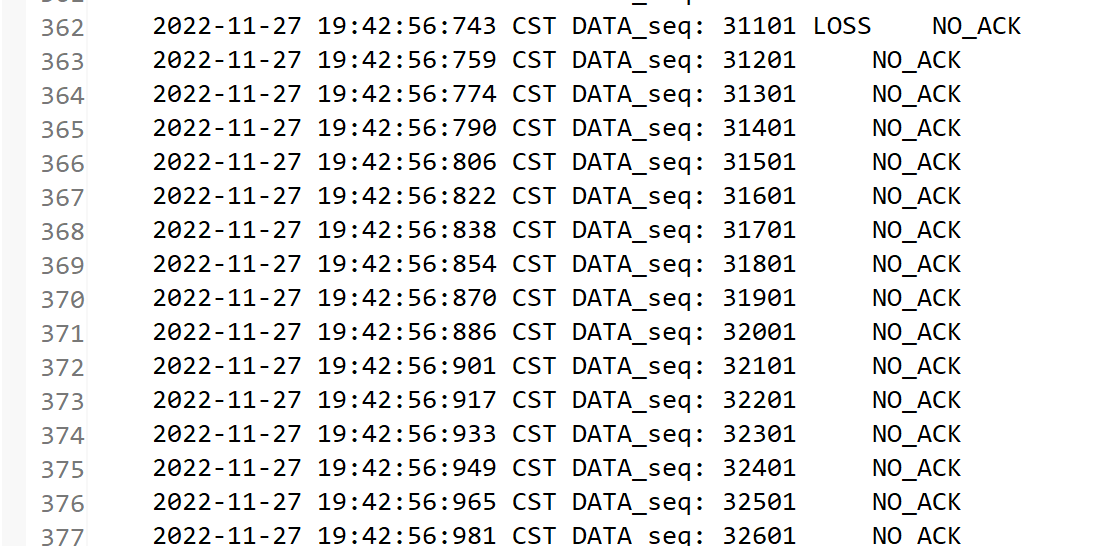


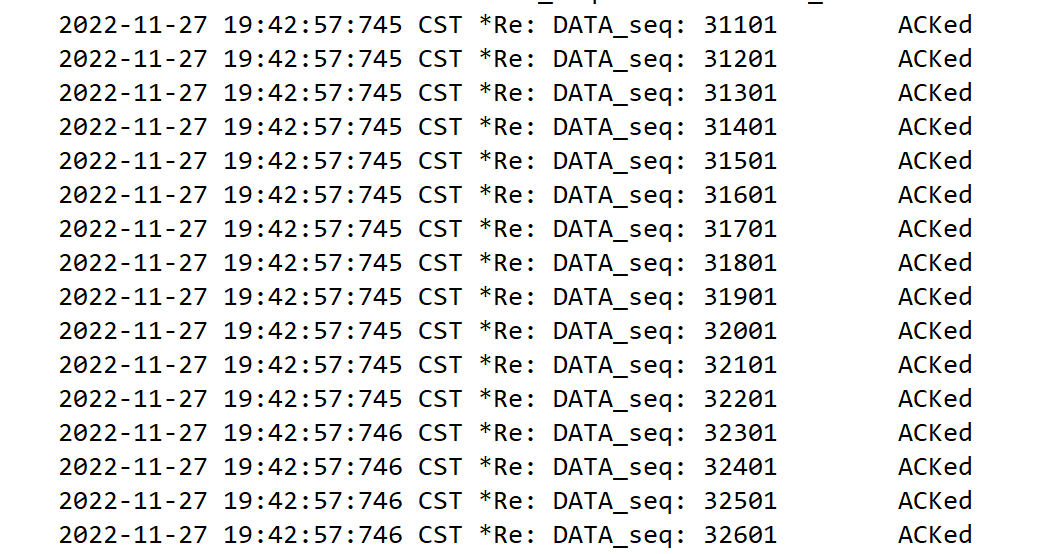


验证:

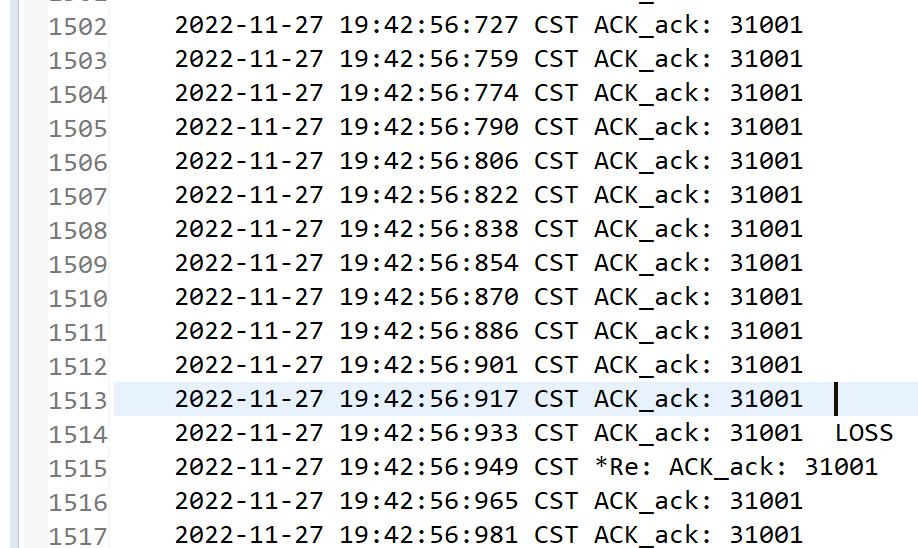
1.比如在31101处发送端包丢失，接收端会回应31001ack，从而导致这一组都不会ack掉，只能够重发

发送端：



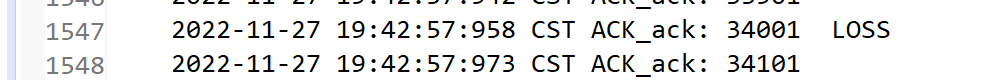


接收端：

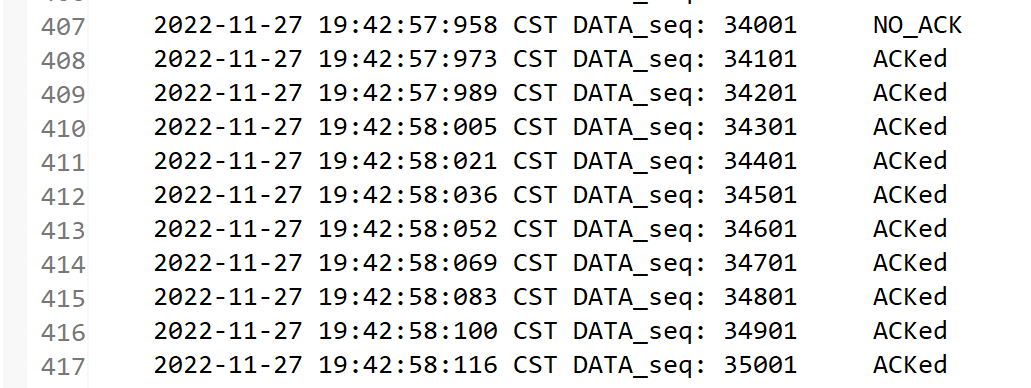


2.但如果是接收端遇到如下的错误，那么由于累计确认的机制发送端不会重发

接收端:



发送端:



VII)RDT4.0SR

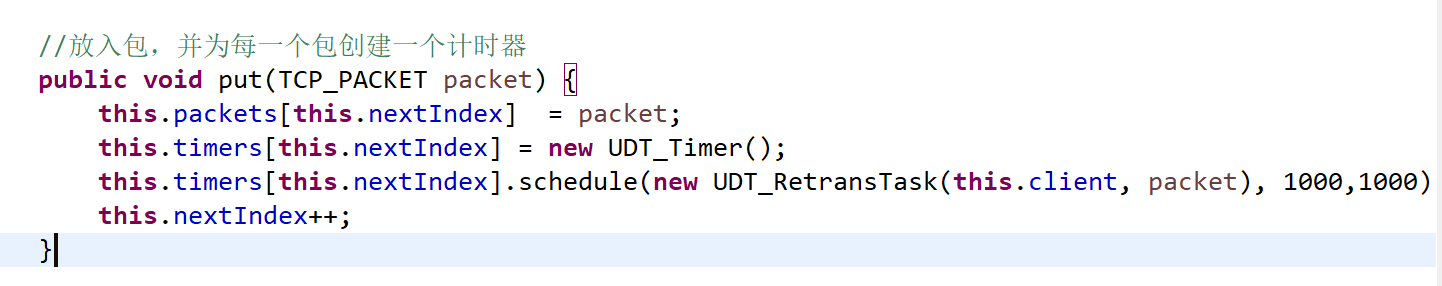
实现:

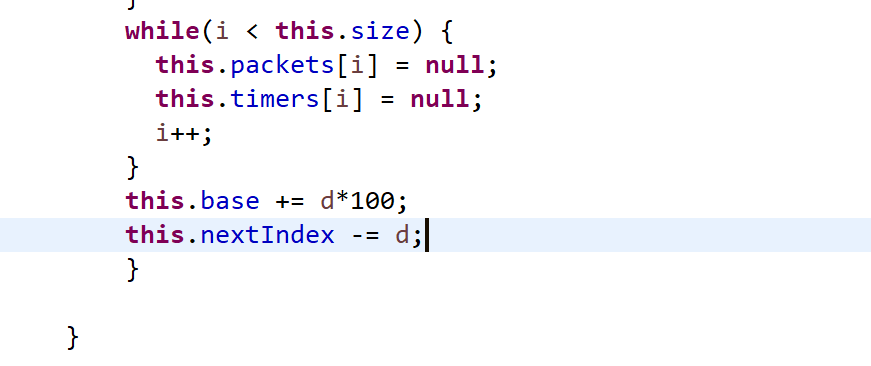
1. 发送端窗口的实现:

1)基本变量:

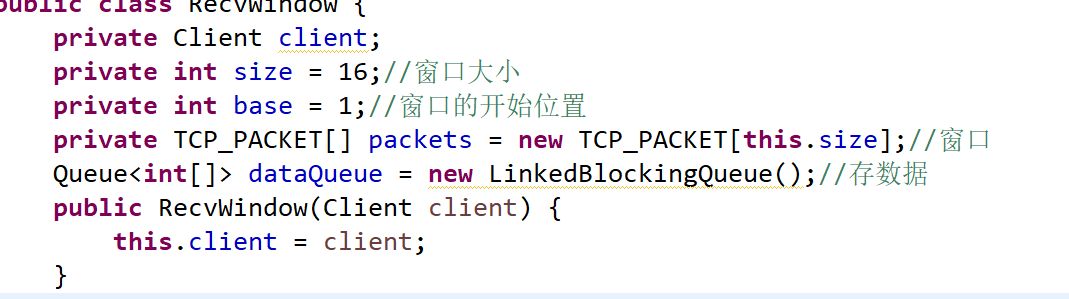


2)put函数

3)接收ack函数



1. 接收端窗口:
2. 基本变量

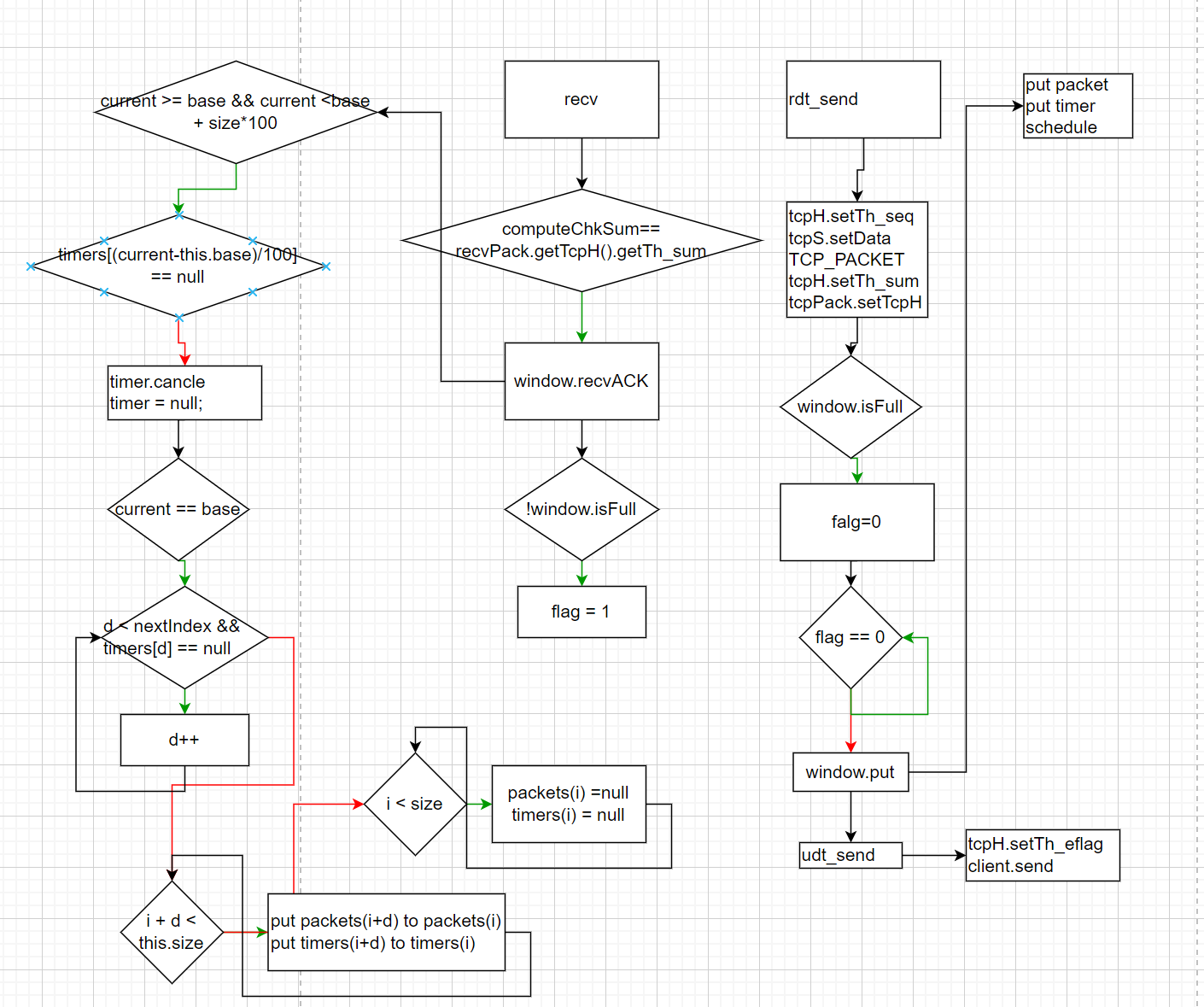
2）recvPacket函数

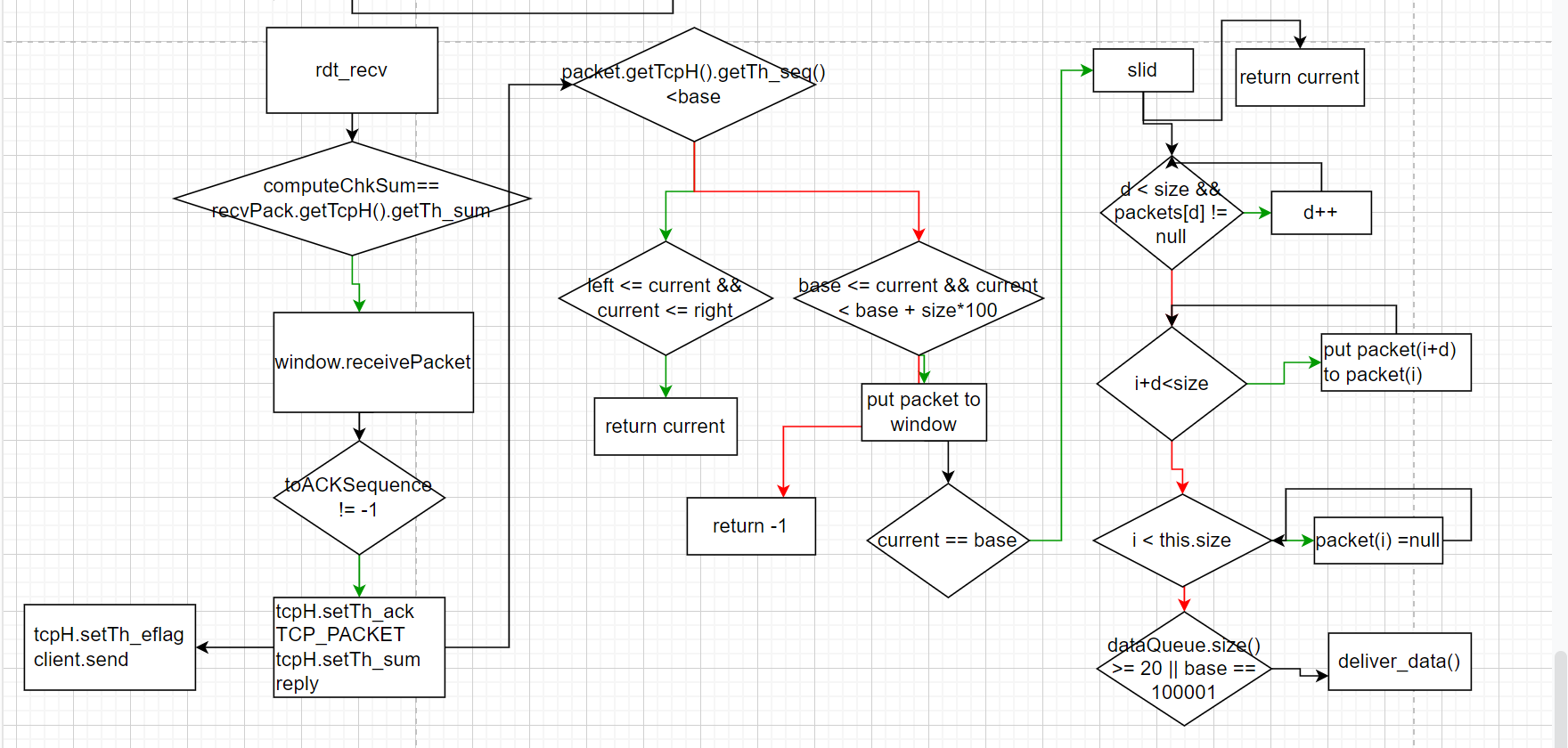
3）滑动窗口函数

4）将数据存入文件转到此类当中，便于交付数据



组成与调用关系：





验证:

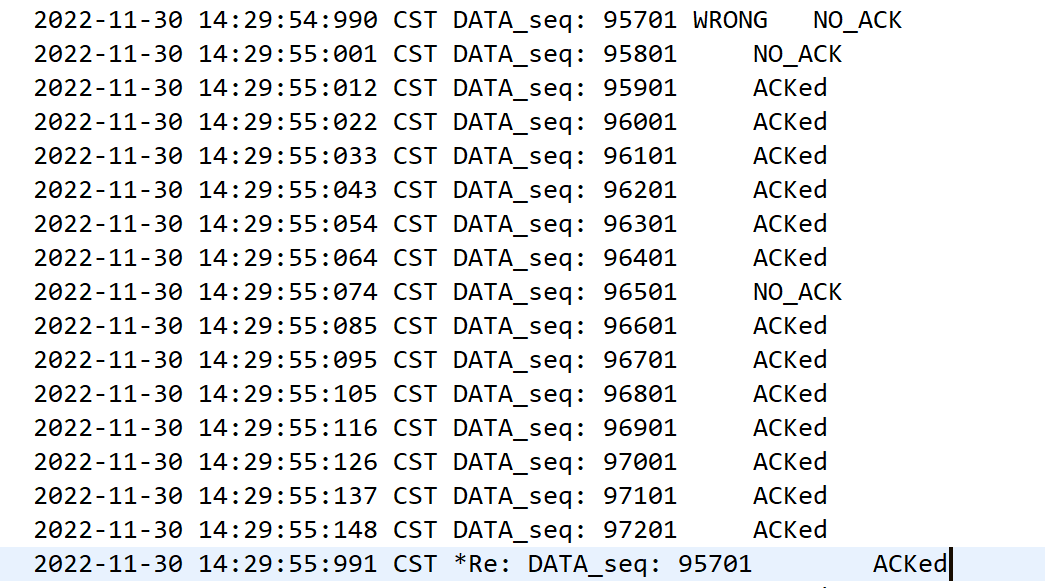
1）对于发送端来说:

无论是发送端发包位错，延迟，丢包，还是接收端发ack位错，延迟，丢包，这些都会导致发送端窗口内此包一直收不到ack，从而一直呆在窗口内，逐渐成为窗口左沿，阻止发送窗口移动，从而导致发送窗口处于full的状态，阻碍包的发送，破局之时便是计时器到时，重发此包，有得到ack回复的希望，若此次接收到ack那么系统才会正常运行。

1. 对于接收端来说：

发送端出错会导致接收端接收不到期望的包，从而卡住接收窗口，由发送端解决此方法；如果是接收端发ack出错，则不会卡住接收窗口；综上，仅发送端发包出错时才会卡住接收窗口

观察log文件我们可以发现规律，发送端若有一包没有接收到ack，那么会隔着15组之后再重发。出现这种现象的原因是设置的一组是16个，若一包没有接收到ack，那么会在1s后重发，而在这1s中，此包会逐渐成为窗口的左沿而后窗口不再移动，导致窗口处于full状态，因此窗口内的另外15个包可以发送但后来的其他包因为窗口已满从而无法发送。

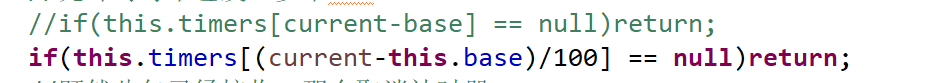


VIII)RDT5.0 Tahoe

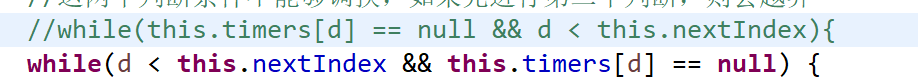
遇到的困难：

1. 在写go back n协议时，发现发送端只能接收顺序的ack，不能接收乱序的ack，各处排查之后发现发送端current的上限写成了this.base+this.size,但this.size只是针对于窗口值，不对应实际的seq，应该改成this.base+this.size\*100.
2. 在写go back n协议时，发现生成ack报文的顺序不能够乱改，否则会出现异常情况（比如会周期性地发送一些acked的分组或者直接运行错误），正确的顺序是设置确认号，初始化包，设置checksum
3. 在写sr协议当中，经常会出现数据越界的问题，比如:

1)数据忘记除100导致越界



2）条件判断的顺序错误导致d超过数组最大下标从而越界



4.循环时忘记下标自增导致死循环问题

发现系统的问题:

1. 如果运行程序过于频繁（相邻两次运行时间间隔短，那么log.txt不会更新）
2. 如果终端上刚显示交互完毕就结束运行，receive.txt里数据是全的，但是log.txt里数据显示不完整。
3. 终端里前半部分所运行的信息不会显示。
4. 工程问题:有时eclipse内的文件复制粘贴会显示不在此系统内，只能够重启eclipse