**计算机网络大作业报告**

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）
   1. **RDT2.0：信道上可能出现位错**
      1. 完成CheckSum类中的computeChkSum()方法



Java.util.zip.CRC32 类是一个可用于计算数据流的CRC-32的类，可以通过update() 函数用指定的字节数组（byte[]）或指定字节的低8位（int）更新CRC-32校验和。在该方法中，只需校验TCP首部中的seq、ack和sum，及TCP数据字段。对于数据字段，由于通过 getTcpS() 中的 getData() 方法得到的是一个int数组，不是 update 要求的参数类型，因此可以通过 Arrays.toString() 方法先转换为String 类型，再通过 getbytes() 转化为 byte[] 类型，从而作为可使用的参数供 update 使用。由于crc.getValue()方法的返回类型为long型，因此需要转换为short型来赋给checkSum并返回。

* + 1. 接收端：对于收到的每一个包，检验其校验和

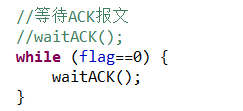


当接收端接收到一个包时，首先计算接收到的包的CRC校验和并和传过来的包带有的校验和进行比较，如果不相同，则检验失败，返回-1包；如果相同，则校验成功，插入到data队列。在插入队列前，需要注意重传可能导致重传正确的分组而发生数据重复，因此发送方需要给每个分组加上sequence number（序号），在接收方接收到正确分组后，需要对接收到的包的序号进行判断，如果和当前接受的包序号相同，表明接收到了重复的包，则丢弃该重复的分组（不向上递交）；当接收到正确补充度的包时，才会插入到data队列，并更新序号。

* + 1. 发送端：循环检查确认号队列中是否有新收到的ACK

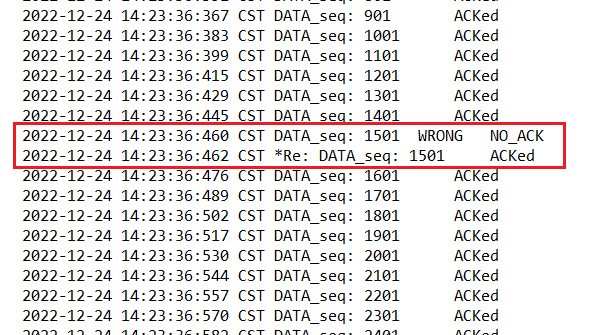


这个函数实现的是发送方检查确认号队列中是否有新收到的ACK，如果新收到的ACK等于刚刚发送包的seq，则令flag=1，如果接收到的ACK值不为刚刚发送的seq，则重发之前的包，并令flag=0。



发送方等待ACK报文是，当flag为0时循环等待，当flag为1时，停止等待。

* + 1. Log日志

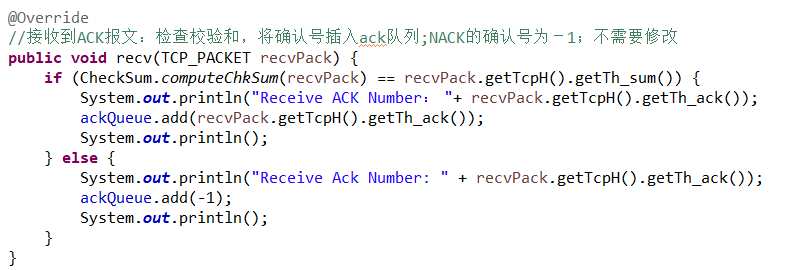


如图所示，对于出错的包，能够立即重传。

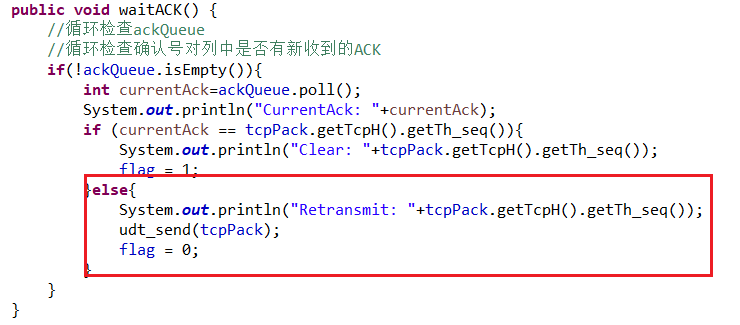
* 1. **RDT2.2**

一次使用两种确认信息ACK，NAK处理起来比较费力，因此RDT2.2中移除NAK的信息，在ACK中加入编号就可以达到确认与否的效果。发送方必须检查接收到的ACK中的报文中被确认的分组序号

* + 1. 发送端：对于接收到的每一个ACK包，检验其校验和。

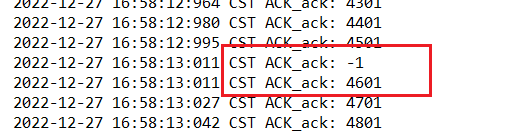


发送端接收到ACK报文后先检查校验和，如果正确，则插入ack队列，错误的话，插入-1。

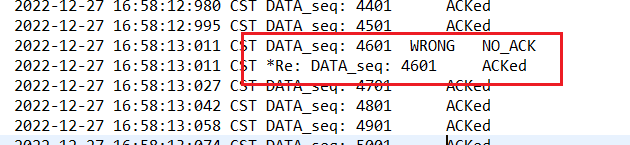


队列中为-1，认为接收方没有收到正确的包，则重传该数据包。

* + 1. Log日志



队列中为-1，表明ACK出错

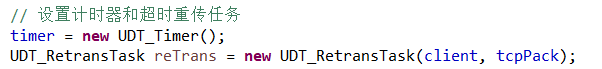


对应的包出错后，能够立即重传

* 1. **RDT3.0**

RDT3.0考虑到数据包丢失和数据传输错误的情况，除了使用ACK机制，在发送端设置了计时器，数据包发出后如果超过规定时间仍未收到ACK或者正确编号的ACK，则进行重传。

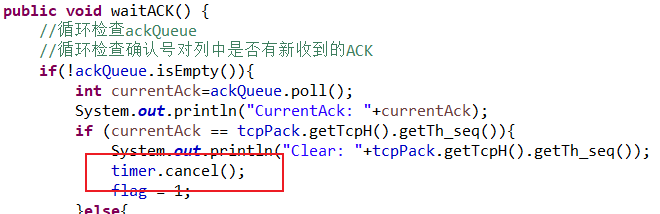
* + 1. 发送端：设置计时器，当数据包发出后三秒内未收到ACK，则进行数据包的重传



首先在TCP\_Sender里设置计时器和超时重传任务。



规定三秒内未收到ACK或者正确编号的ACK，则执行重传任务，直到收到正确的ACK。

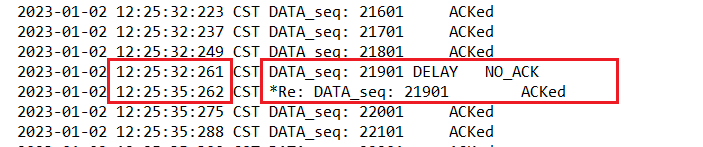


收到正确的ACK后将计时器关闭。



同时设置错误控制标志为只延迟。

* + 1. Log日志



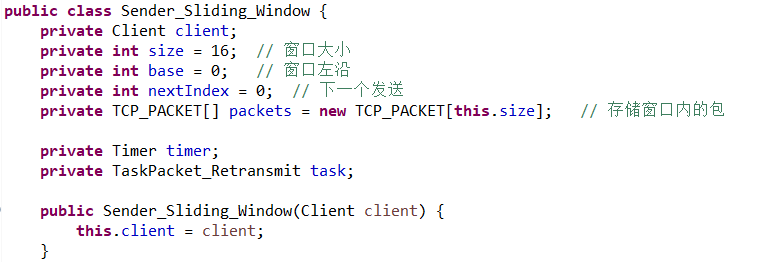
当通道出错，没有收到ACK，则过三秒后自动重发

* 1. **RDT4.0 回退N步协议(GBN)**

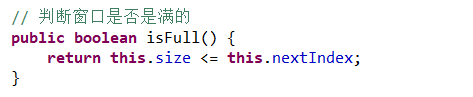
Go-back-N是发送窗口大小为N，发送方每次将窗口内等待发送并可以发送的包发送出去，接收方每次只发送一个ACK包，当发送方收到接收方的ACK不是期待的(即包发生出错、延迟或丢失)，则发送方发送当前窗口的所有的包。

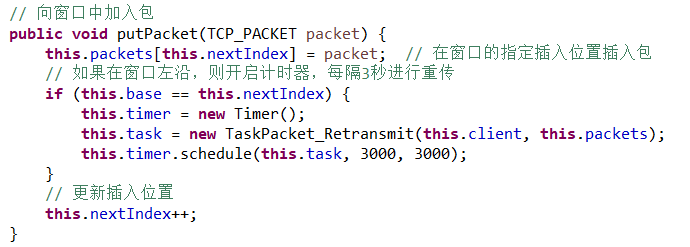
* + 1. 发送端：当接收到的不是期待的包，则发送方对当前窗口内的所有包进行重传。

首先需要自定义窗口类Sender\_Sliding\_Window.java

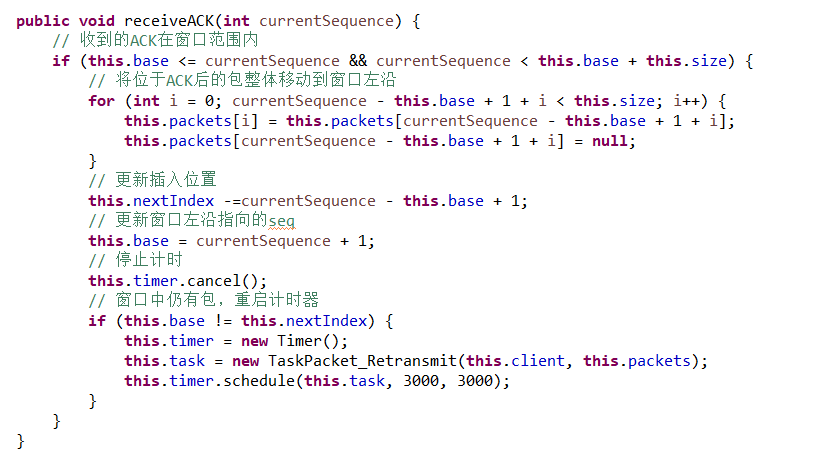


在窗口类中，要实现三个函数：判断窗口是否已满，向窗口内加入包和接收ACK。



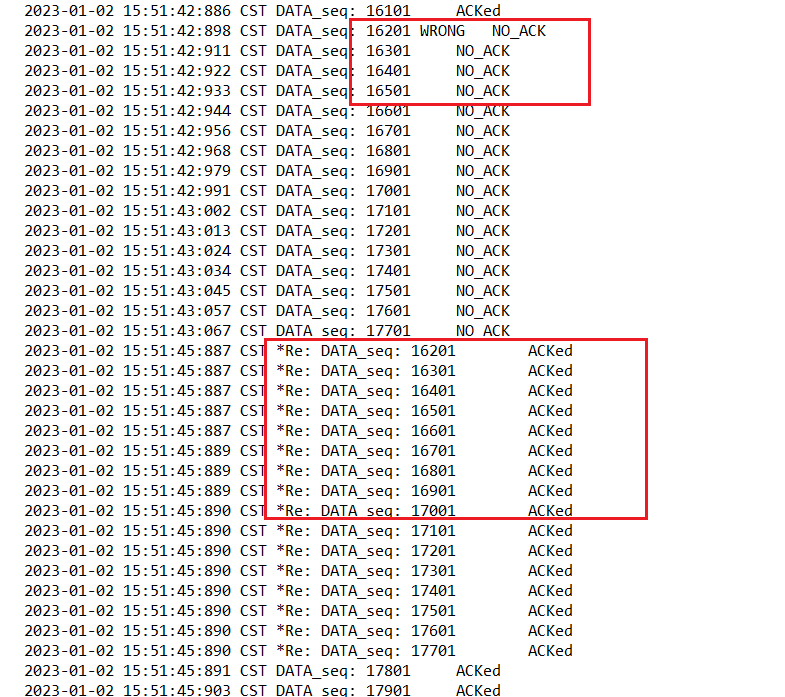


这个函数实现了向窗口中加入包的功能。



这个函数实现的功能是，当接收到ACK时，如果该ACK在窗口范围内，则将位于ACK之后的包整体移动到窗口左沿，更新下一个包的插入位置等并停止计时，如果当前窗口中仍有包，则重启计时器、重新设置超时重传任务。

* + 1. Log日志

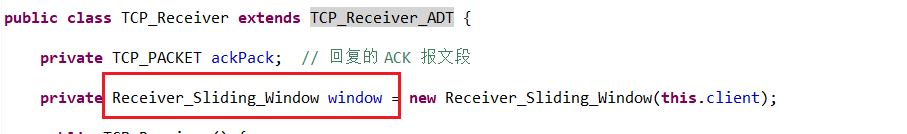


如图，16201包没有收到ACK，重发改帧及其之后窗口内的所有的帧

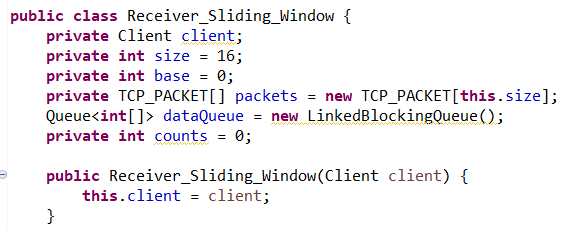
* 1. **RDT4.0 SR 选择重传**

Select Response则是发送方和接收的窗口都是N（可以不一样），接收方对收到无序的包进行缓存，并对每个包ACK，发送方只需重传出错、延迟、丢失的分组，提高了网络的吞吐量。

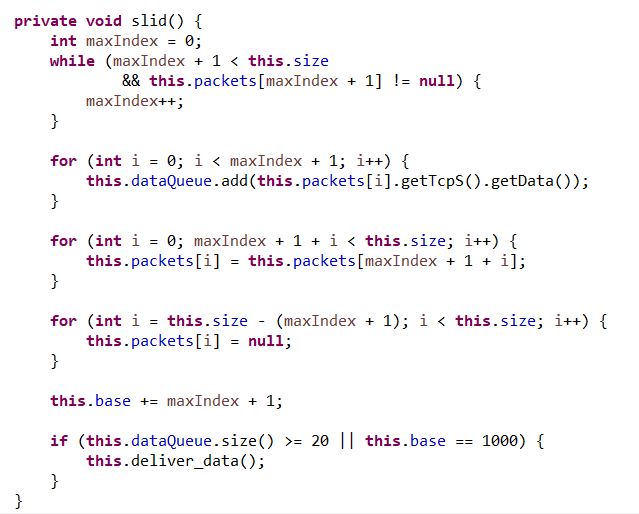
* + 1. 接收端将确认一个正确接收的帧而不管其是否按序。失序的帧将被缓存，并返回给发送方一个该帧的确认帧，直到所有帧都被收到为止，这时才可以将一批帧按序交付给上层，然后向前移动滑动窗口。



因此接收端同样需要自定义滑动窗口类Receiver\_Sliding\_Window.java



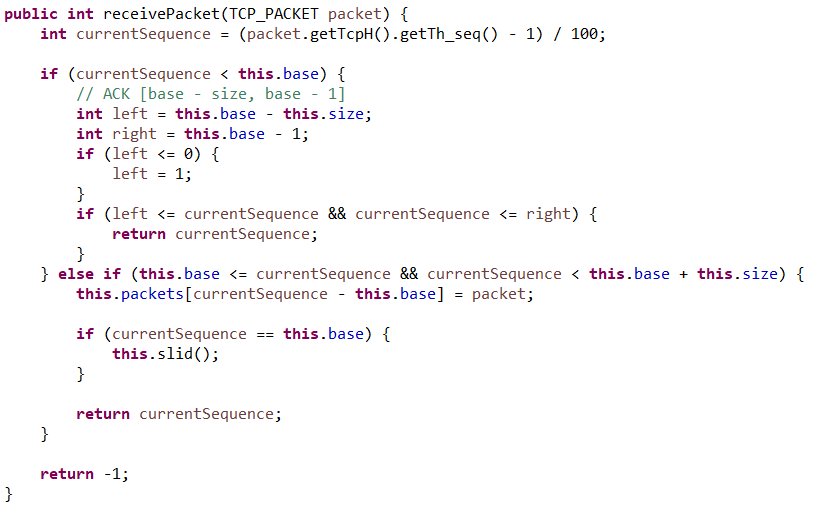
该窗口类共有三个功能函数



这个函数实现的是窗口的滑动，在这个函数中，需要记录窗口左移到的位置maxIndex为最小未收到数据包的位置，然后将已接收到的分组加入到交付队列，将剩余位置的包左移，将左移的包原来的位置置空，最后移动窗口左端。



这个函数实现的是交付数据，将数据写入文件。循环检查dataQueue中是否有新交付的数据，有的话将数据写入文件，最后清空输出缓存。



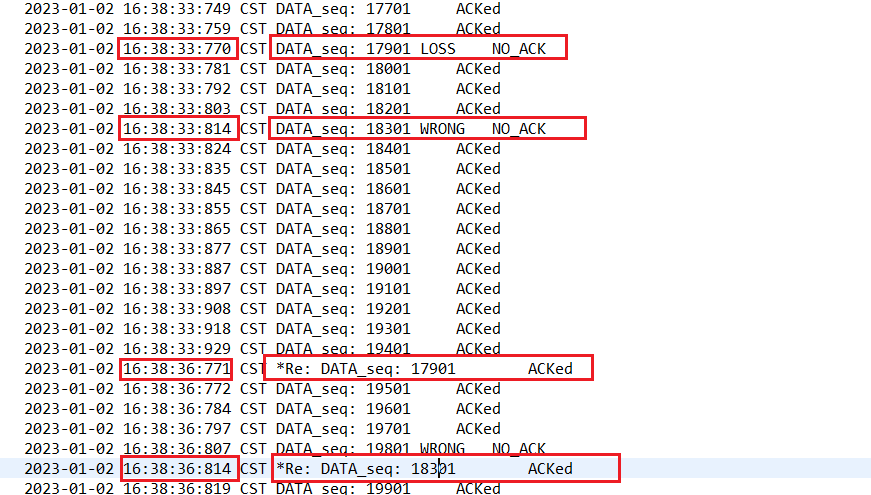
这个函数实现了发送端接收数据包的功能，如果当前收到的包不是期待的包，则将其缓存，返回ACK。对于正确分组，则加入窗口。



* + 1. 发送端

发送端收到ACK，加入该帧序号在窗口内，将改帧标记为已接受，如果改帧序号是窗口左沿，则窗口向前移动到具有最小序号的未确认帧处，每个帧都有自己的定时器，超时后只重传一个帧。

* + 1. Log文件



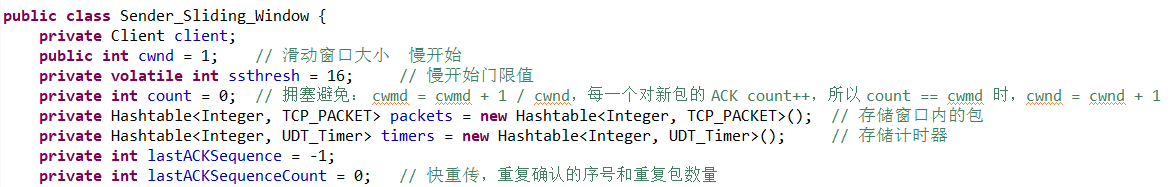
如图，当未收到正确的ACK，三秒后重传该帧。

* 1. **RDT5.0 Tahoe**

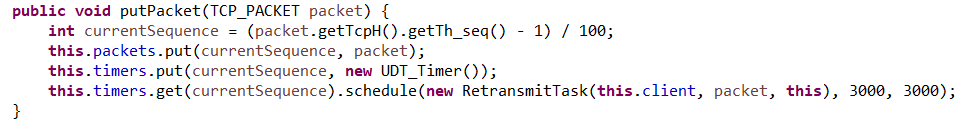
Tahoe主要有三个算法去控制数据流和拥塞窗口：慢启动、拥塞避免和快重传

* + 1. 发送端

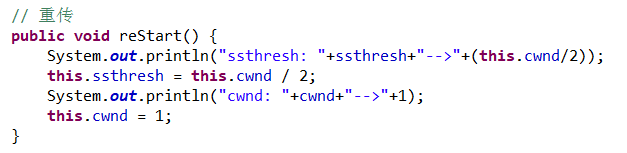
发送端需要更新窗口类



设置几个变量：cwnd为滑动窗口大小，ssthresh为慢开始门限值，count用来实现拥塞避免，packets用来存储窗口内的包，timers存储对应包的计时器，lastACKSequence和lastACKSequenceCount分别代表重复确认的包的序号和数量，用来实现快重传。



这个函数实现向窗口中加入包，并设置计时器和重传任务。

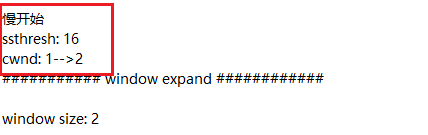


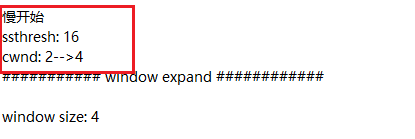
发送发收到了三个重复的ACK后，立马重传，这个函数实现了重传，将ssthresh的值设置为当前cwnd的一半，而cwnd减为1，重回慢开始阶段。

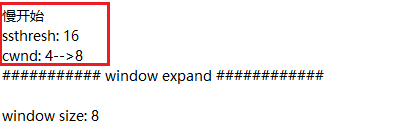


这个函数实现了接收ACK包的功能，如果收到的包的序号和前一个包的序号相同，即收到了重复的包，就需要对重复的包的个数进行累加。如果累加个数超过三，就认为发生了网络拥堵，就要实施快重传，并转入慢开始；如果是正常接收，即收到新的ACK，则清除前面的包，清空对应的计时器，记录当前收到的新的ACK，设置其重复次数为1，来供下一个ACK参照，之后需要比较cwnd和ssthresh的大小：如果cwnd<ssthresh，进入慢开始阶段，每收到一个ACK，cwnd的数量加一；否则(cwnd>ssthresh)进入拥塞避免阶段，在该阶段下，cwnd以线性方式增长，大约每经过一个RTT，cwnd的值就会加一。

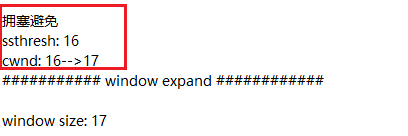
* + 1. Console文件

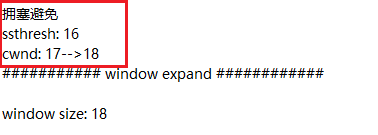


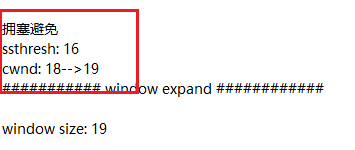




当cwnd<ssthresh时，执行慢开始算法，cwnd呈指数增长

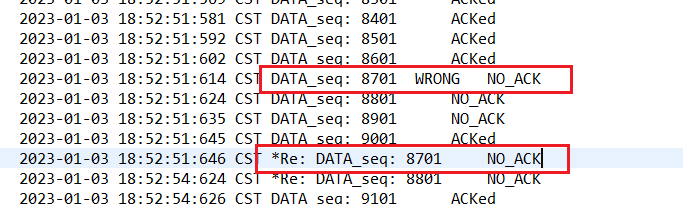




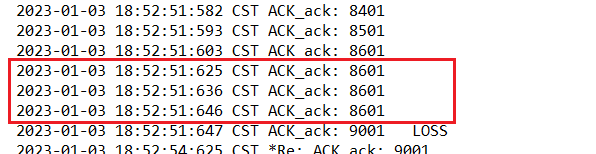


当cwnd>=ssthresh时，执行拥塞避免算法，cwnd呈加法增大

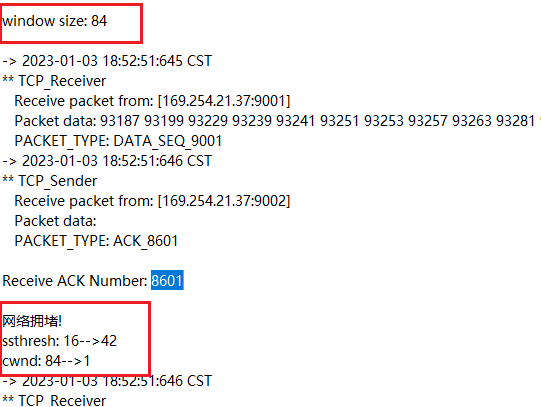
* + 1. Log文件



8701包发生错误，要快速重传应该重发三次8601的ACK包



收到重复的8601包



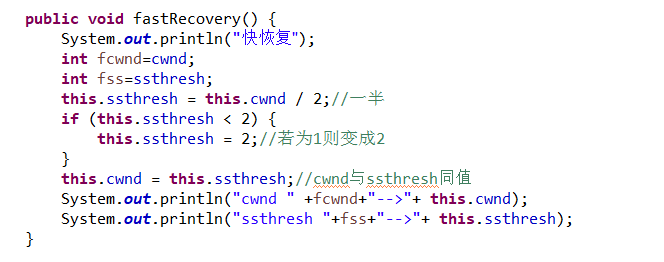
连续收到三个重复确认，认为发生网络拥堵，慢开始门限变为当前cwnd的一半，cwnd设置为1。

* 1. **RDT5.1 Reno**

Reno除了包含Tahoe的三个算法：慢开始、拥塞避免、快重传，多了一个快恢复算法。

* + 1. 发送端

在Tahoe的基础上，更新Sender\_Sliding\_Window，增加快恢复函数

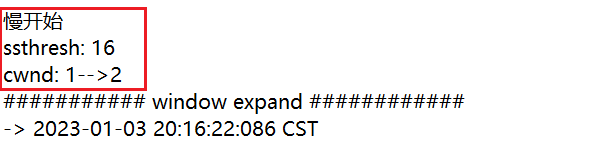


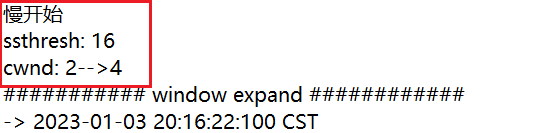
当收到三个重复的ACK或是超过了RTO时间且尚未收到某个数据包ACK，Reno就会认为丢包了，并认定网络中发生了拥塞。Reno把当前的ssthresh的值设置为当前cwnd的一半，cwnd设置为ssthresh的大小，然后执行拥塞避免算法。

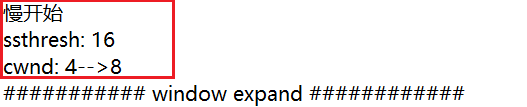


当收到三个重复的ACK，Reno认定网络中发生了拥塞，执行快恢复，其他的和Tahoe类似。

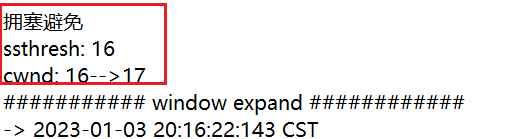
* + 1. Console 文件

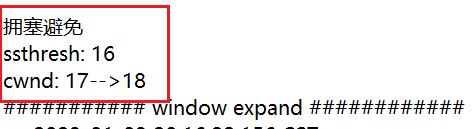




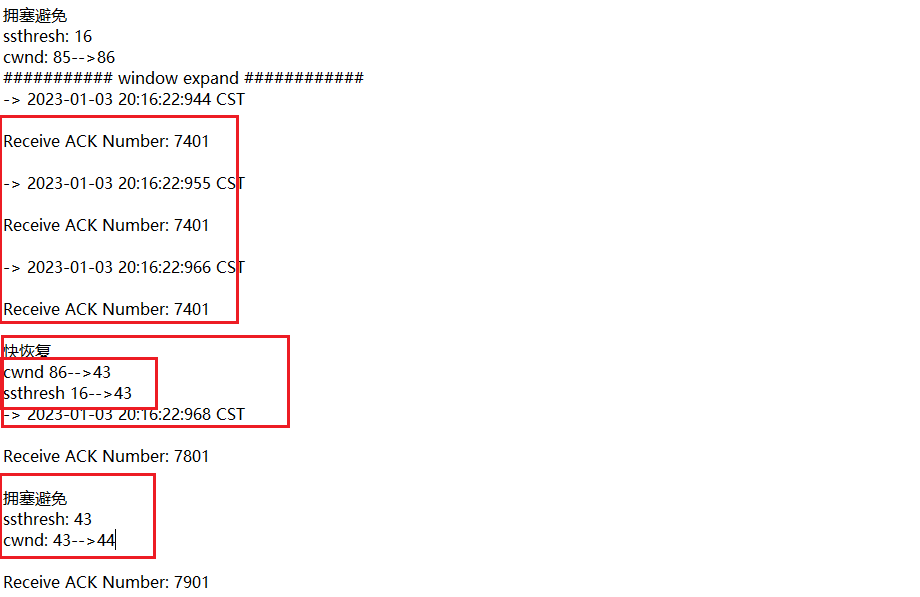


Cwnd<ssthresh时，执行慢开始，指数增加



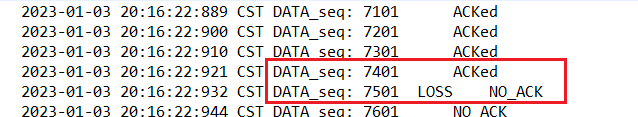


当cwnd>ssthresh时，执行拥塞避免算法，加法增加

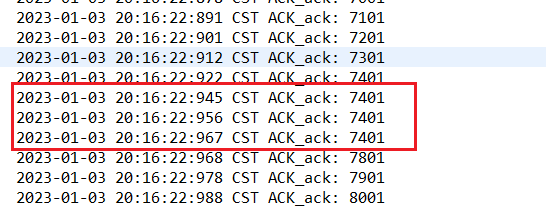


当收到三次重复ACK后，Reno认为丢包，认定网络中发生了拥塞，将ssthresh设置为当前cwnd的一半，在上图中，收到三个重复ACK时cwnd为86，收到三个重复ACK后，ssthresh的值变为cwnd的一半43，cwnd的值设置为更新后的ssthresh，即43，然后执行拥塞避免算法，cwnd呈加法增长。

* + 1. Log文件



7501丢失，要快速重传应该重发3次7401的ACK包



连续收到三个重复的ACK包。

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）

项目基本完成，遇到的最大困难是单纯看log日志文件很难看出慢开始、拥塞避免、快恢复等过程，以及在各个阶段cwnd和ssthresh的变化情况，解决方案是在各个关键阶段打印所需要的信息，然后将整个项目的输出保存到txt文件，对照着各个阶段的信息，就能很好地了解当前处于的阶段，cwnd以及ssthresh的动态变化。

1. 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)

首先迭代开发让我顺着RDT版本的更迭从头至尾复习了一遍基础知识，了解了RDT各个版本相较之前的版本发生了哪些变化；其次迭代开发使代码重用更加容易，比如这次实验中的TCP\_Sender和TCP\_Receiver，以及借助Tahoe实现Reno都体现了这一点，这让开发从易到难、循序渐进，让开发更加轻松。

1. 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)

问题1：单从log文件很难看出当前阶段以及cwnd、ssthresh的动态变化

解决方法：辅以系统输出文件，在每个阶段输出自己需要的信息

问题2：系统复习相关理论后，把滑动窗口的理论变为实际代码有点困难

解决方法：查阅资料，学习他人的实现方法。

1. 对于实验系统提出问题或建议(1分)

本次实验难度较大，耗时较长，通过实际编程，我对教材相关原理了解更深。