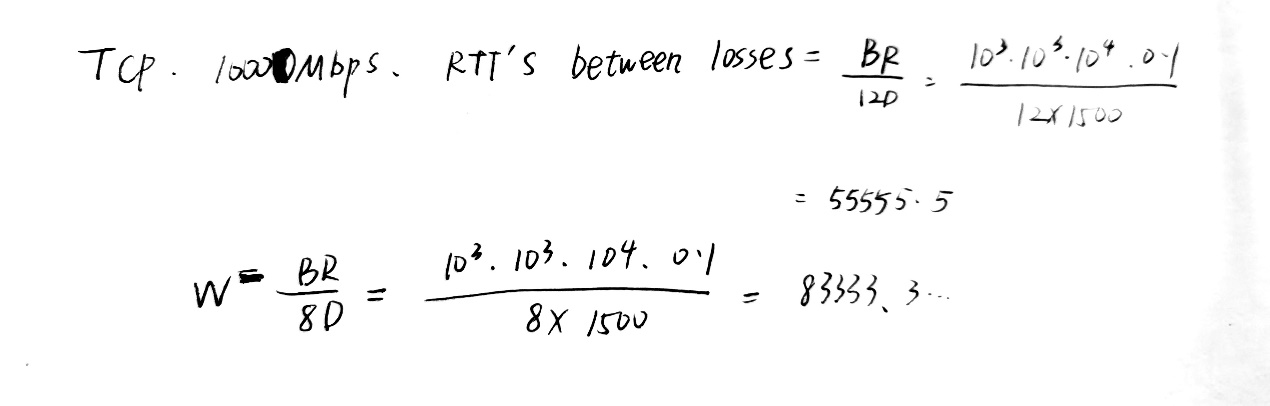
1. Kurose&Ross教材第七版，第三章。第182页，倒数第7行的“83333”这个数值，是怎么计算出来的？ （提示，这是RFC3649中的原文）

原文：For example, for a Standard TCP connection with 1500- byte packets and a 100 ms round-trip time, achieving a steady-state throughput of 10 Gbps would require an average congestion window of 83,333 segments

……

From Appendix A, achieving an average TCP throughput of B bps requires a loss event at most every BR/(12D) round-trip times. This is illustrated in Table 1, for R = 0.1 seconds and D = 1500 bytes. The table also gives the average congestion window W of BR/(8D), and the steady-state packet drop rate P of 1.5/W^2.



2. 给出解释：第179页，图3-51中“快速恢复/Fast recovery”状态中的

cwnd=cwnd+MSS”，与“拥塞避免”状态中的cwnd=cwnd+MSS\*（MSS/cwnd）”。二者在cwnd窗口增长中的差异是什么？为什么起了个“【快】恢复”的名字？有多快？

快速恢复中cwnd=cwnd+MSS是在收到三个重复的ACK后，对拥塞窗口的值做调整（减半加3）之后，再次收到重复的ACK时，大小只增加1；

拥塞避免中cwnd=cwnd+MSS\*（MSS/cwnd）是cwnd >= ssthresh后，接收到新的不重复的ACK之后，以这种方式增加拥塞窗口的大小。

因为这个算法旨在对收到三次重复的ACK之后的状态进行快速调整，达到从重复ACK时的数据都已收到，回到恢复之前的状态，也即再次进入拥塞避免状态，这个过程依靠的是发送方接收到三个重复的ACK之后，主动把ssthresh设置为cwnd的一半，把cwnd设置为ssthresh+3，然后重传丢失的报文段（加3的原因是因为收到3个重复的ACK，表明有3个“老”的数据包离开了网络）所以说“快”。

有多快体现在cwnd初始大小上。还有3个Duplicated Acks说明网络也不那么糟糕，所以并不需要重新回到慢启动从1开始发送。

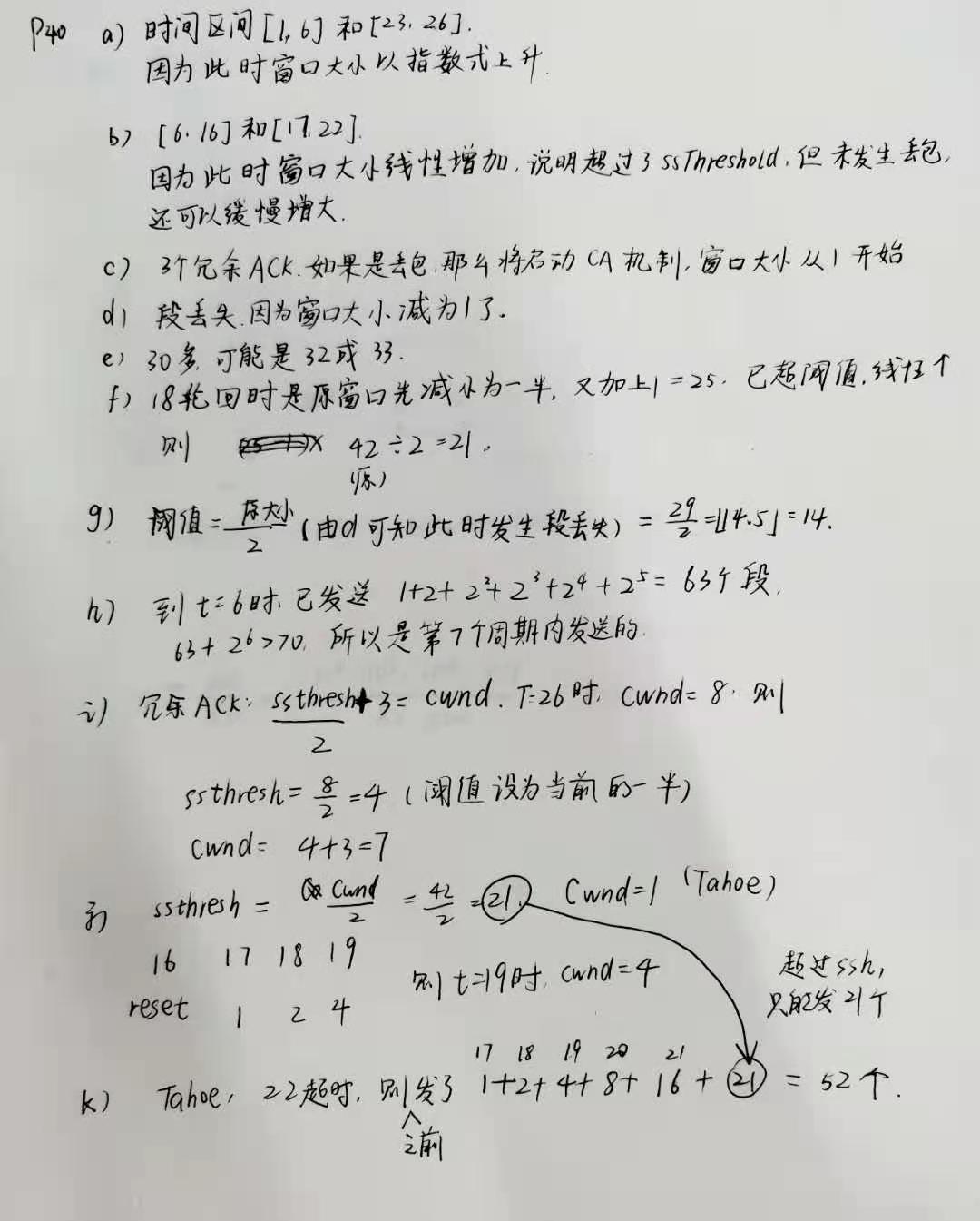
3. 不管是哪种congestion control algorithm，本质上都是对可用带宽/资源的探测，

以决定能向网络中投放多少数据才合适。教材中的Tahoe、reno算法，是如何做探测的？请阅读教材P176页起的内容。

Tahoe：在Congestion Avoidance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1；快重传：只要发送方收到了三个重复的ACK就马上重传，并且将ssthresh的值设置为当前cwnd的一半，而cwnd减为1，重回slow start阶段。

Reno: 除了包含Tahoe的三个算法，Reno多了快速恢复算法。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的ACK，Reno就会认为丢包，并认定网络中发生了拥塞。Reno会把当前的ssthresh的值设置为当前cwnd的一半，但是并不会回到slow start阶段，而是将cwnd设置为（更新后的）ssthresh+3MSS，之后cwnd呈线性增长。

4. Kurose&Ross教材第七版，第193页，P40。注：不仅仅是给出答案，还要给出简要的解释说明）



论文阅读

1.due day： 20221028 （周五）22点。

2.论文：讲义/Literature/!1988Congestion Avoidance and Control.pdf，阅读笔记

模板在同一目录下。在这篇文献中，提出了拥塞控制算法的原始思想。

3.上传：作业/20221021文献阅读-1988Congestion Avoidance and Control

4. 文件名：学号姓名.doc