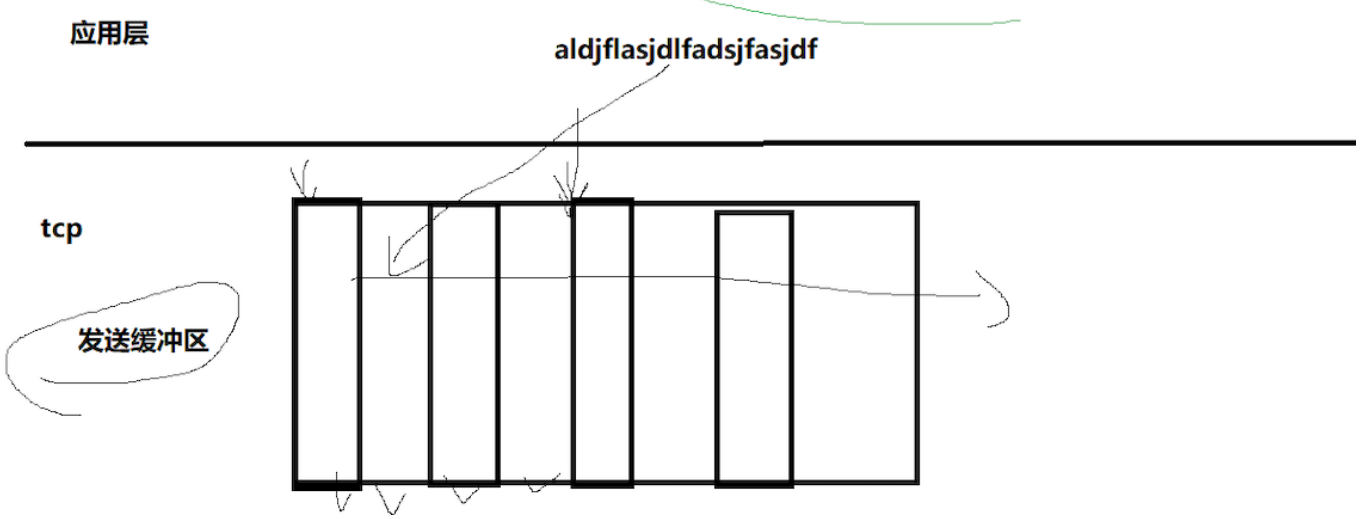
0314_Tcp

确认应答机制（ACK）

我们理解tcp的缓冲区时
可以把tcp的发送缓冲区看作一个 char sendbuffer[NUM] 数组

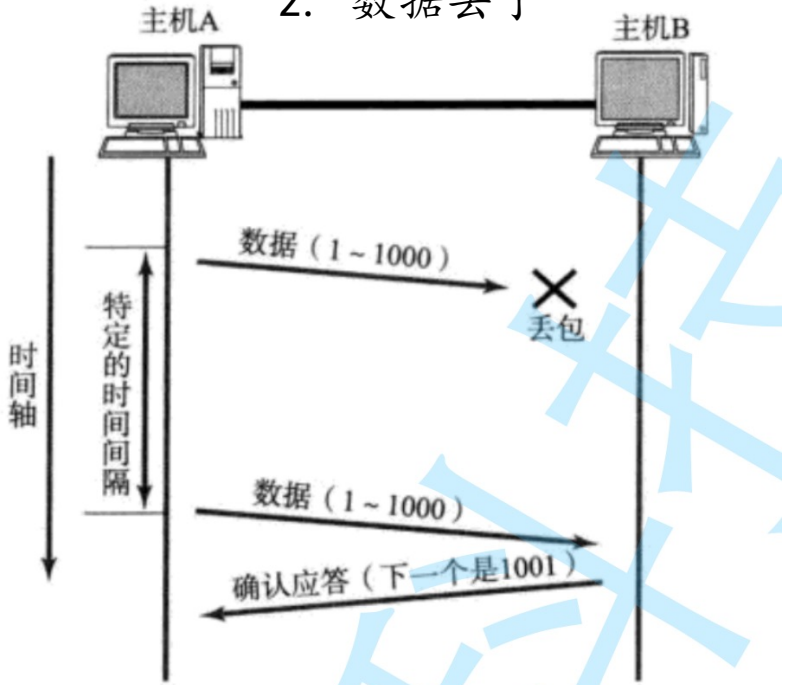
所以报文编号的下标，可以用数组的下标！

1. 我们理解TCP的发送缓冲区，我们可以将tcp的发送缓冲区看做一个 char sendbuffer[NUM]样的数组。



超时重传机制

1. ACK丢了
2. 数据丢了



- 主机A发送数据给B之后，可能因为网络拥堵等原因，数据无法到达主机B；
- 如果主机A在一个特定时间间隔内没有收到B发来的确认应答，就会进行重发；

收到了重复报文，可以通过需要进行去重

超时重传的时间间隔如何设置?

那么, 如果超时的时间如何确定?

> - 最理想的情况下, 找到一个最小的时间, 保证 "确认应答一定能在这个时间内返回".
> - 但是这个时间的长短, 随着网络环境的不同, 是有差异的.
> - 如果超时时间设的太长, 会影响整体的重传效率;
> - 如果超时时间设的太短, 有可能会频繁发送重复的包;

TCP为了保证无论在任何环境下都能比较高性能的通信, 因此会动态计算这个最大超时时间.

> - Linux中(BSD Unix和Windows也是如此), 超时以500ms为一个单位进行控制, 每次判定超时重发的超时时间都是500ms的整数倍.
> - 如果重发一次之后, 仍然得不到应答, 等待 2*500ms 后再进行重传.
> - 如果仍然得不到应答, 等待 4*500ms 进行重传. 依次类推, 以指数形式递增.
> - 累计到一定的重传次数, TCP认为网络或者对端主机出现异常, 强制关闭连接.

流量控制

因此TCP支持根据接收端的处理能力, 来决定发送端的发送速度. 这个机制就叫做**流量控制(Flow Control)**;

- 接收端将自己可以接收的缓冲区大小放入 TCP 首部中的 "窗口大小" 字段, 通过ACK端通知发送端;
- 窗口大小字段越大, 说明网络的吞吐量越高;
- 接收端一旦发现自己的缓冲区快满了, 就会将窗口大小设置成一个更小的值通知给发送端;
- 发送端接受到这个窗口之后, 就会减慢自己的发送速度;
- 如果接收端缓冲区满了, 就会将窗口置为0; 这时发送方不再发送数据, 但是需要定期发送一个窗口探测数据段, 使接收端把窗口大小告诉发送端.

TCP这种根据接收端的处理能力，来决定发送端端发送速度，这个机制叫做流量控制！两个方向上都能做流量控制。

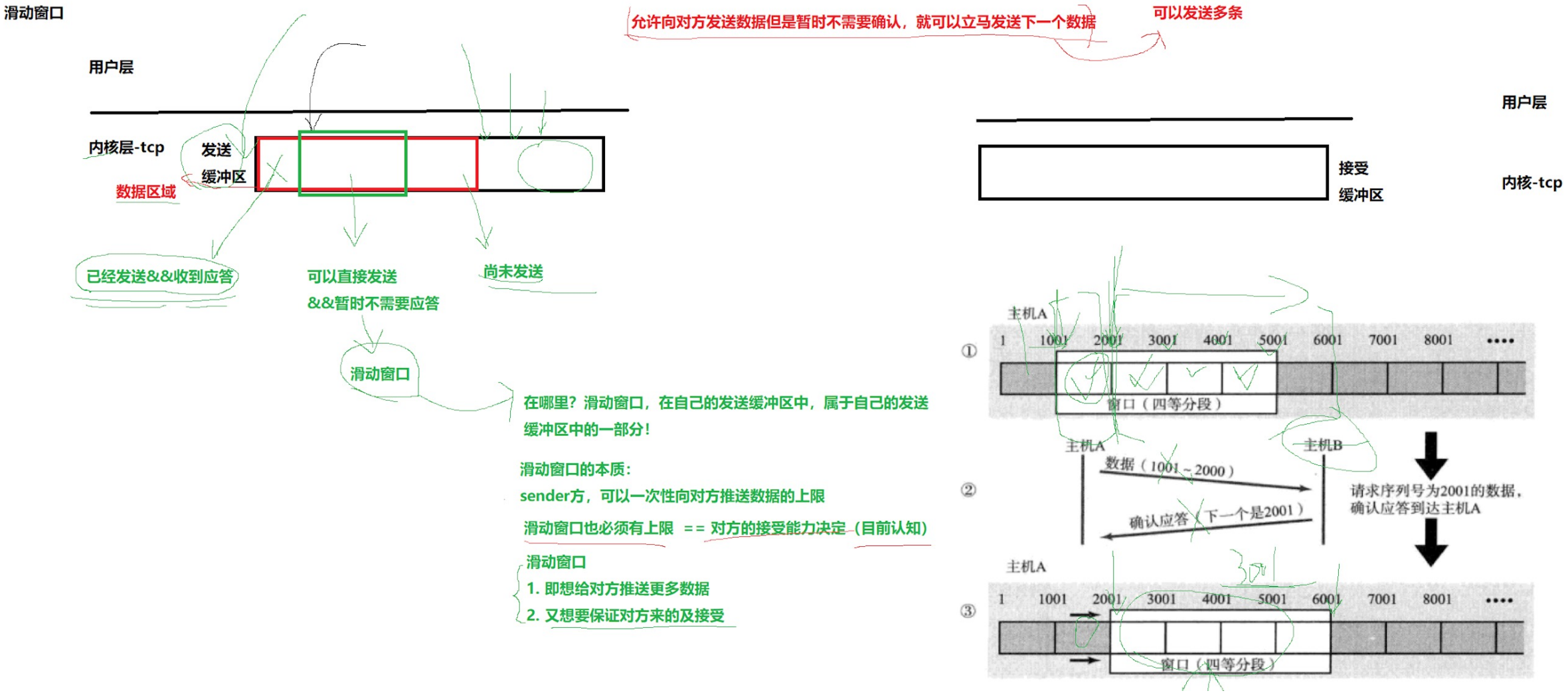
接收端如何把窗口大小告诉发送端呢? 回忆我们的TCP首部中, 有一个16位窗口字段, 就是存放了窗口大小信息;
那么问题来了, 16位数字最大表示65535, 那么TCP窗口最大就是65535字节么?
实际上, TCP首部40字节选项中还包含了一个窗口扩大因子M, 实际窗口大小是 窗口字段的值左移 M 位;

在进行流量控制的时候，第一次我们如何得知对方的接受能力？
我们所要的是对方的接受能力（对方缓冲区中的剩余大小）如何得知？

注意！第一次发数据不等于第一次交换报文。
什么时候第一次交换报文？
三次握手的时候
而且此时，三次握手没有完成是不能带数据的，此时不就交换了报文了吗？

滑动窗口

允许向对方发送数据，但是暂时不需要确认，就可以发送下一个数据
即，一次发送一堆数据



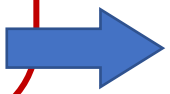
所以要发送的数据被分成了三个部分

1. 已经发送 && 已经收到ACK
2. 可以直接发送 && 暂时不需要ACK
3. 尚未发送

滑动窗口，也必须有上限！

滑动窗口只向右滑动呢？
会不会向左呢？

1. 滑动窗口必须向右移动吗？不一定！
2. 滑动窗口可以为0吗？可以！
3. 如果没有收到开始的报文的应答，而是收到中间的？
   可能！但不影响
4. 超时重传，背后的含义：就是没有收到应答的时候，
   数据必须背暂时保存起来！
5. 滑动窗口一只向右移动，会有越界问题吗？不会！

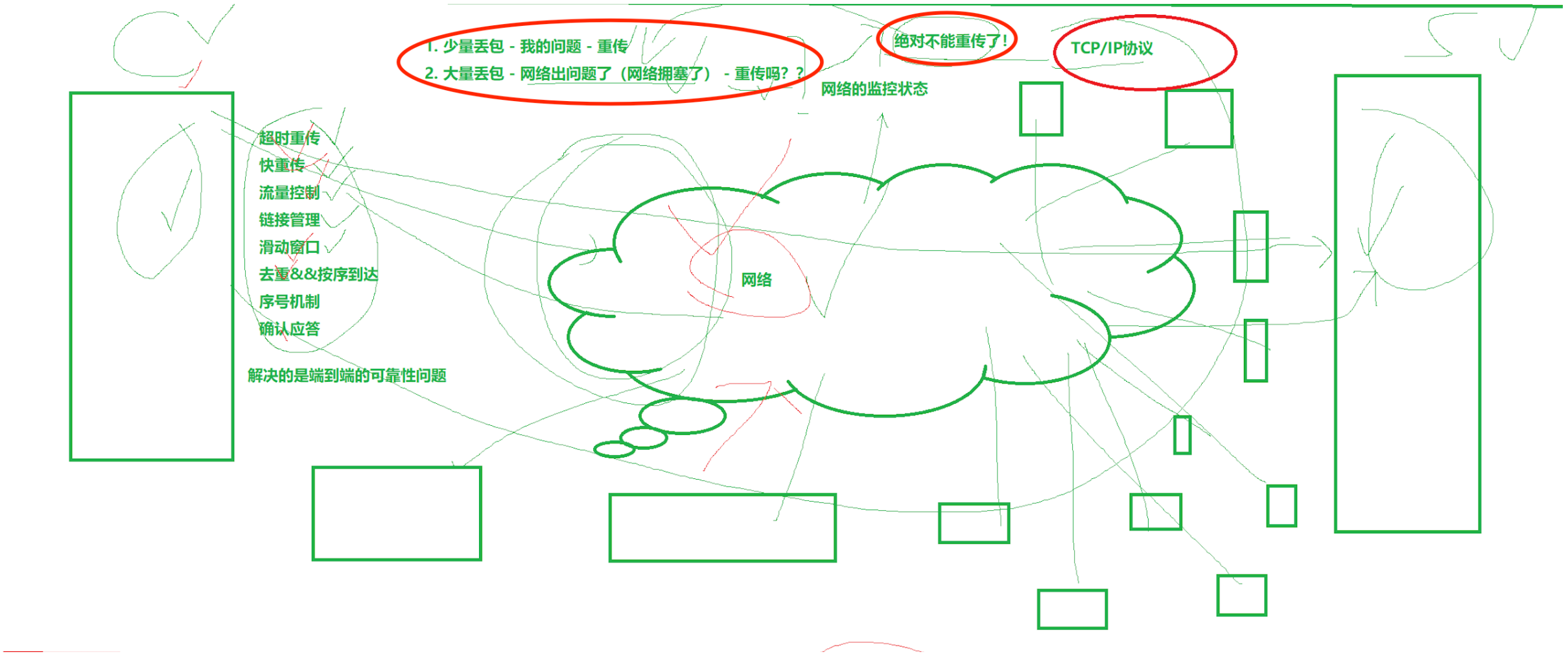


实际上tcp的缓冲区是环状的

收到谁的ACK，就直接把窗口的start右到ACK对序号就行了

滑动窗口是解决效率问题，还是可靠性问题？

拥塞控制


1. 少量丢包 - 我的问题 - 重传
2. 大量丢包 - 网络出问题了（网络拥塞了） - 重传吗？？
绝对不能重传了！
TCP/IP协议
网络的监控状态
超时重传
快重传
流量控制
链接管理
滑动窗口
去重&&按序到达
序号机制
确认应答
解决的是端到端的可靠性问题
网络

TCP引入**慢启动**机制，先发少量的数据，探探路，摸清当前的网络拥堵状态，再决定按照多大的速度传输数据

- 此处引入一个概念程为**拥塞窗口**
- 发送开始的时候, 定义拥塞窗口大小为1;
- 每次收到一个ACK应答, 拥塞窗口加1;
- 每次发送数据包的时候, 将拥塞窗口和接收端主机反馈的窗口大小做比较, 取较小的值作为实际发送的窗口;

拥塞窗口：单台主机一次向网络中发送大量数据时，可能会引发网络拥塞的上限值。

所以本质上滑动窗口大小 = min(拥塞窗口，对方接受能力)

像上面这样的拥塞窗口增长速度, 是指数级别的. "慢启动" 只是指初使时慢, 但是增长速度非常快.

- 为了不增长的那么快, 因此不能使拥塞窗口单纯的加倍.
- 此处引入一个叫做慢启动的阈值
- 当拥塞窗口超过这个阈值的时候, 不再按照指数方式增长, 而是按照线性方式增长

为什么拥塞之后，前期是指数增长？

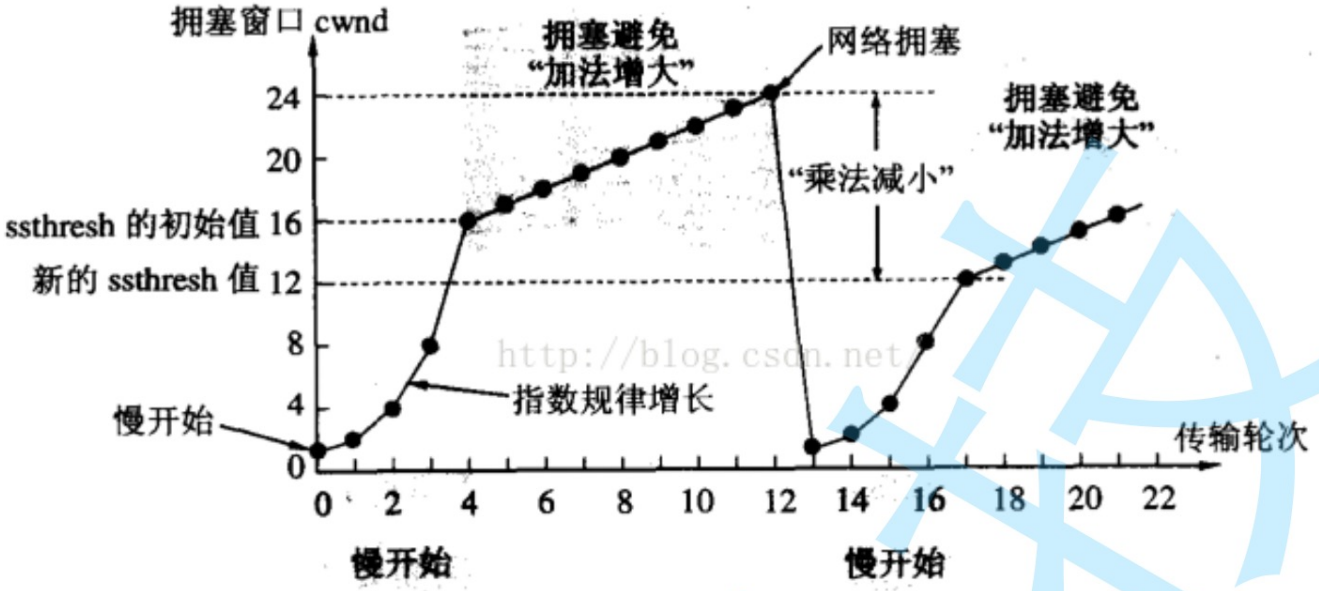
为何拥塞之后，前期是指数增长?

指数 前期慢，后期非常快

一旦拥塞

1. 前期要让网络有一个缓一缓的机会 - 少，慢
2. 中后期，网络恢复了之后，尽快恢复通信的过程 - 可能会影响通信效率

1. 向解决网络拥塞问题
2. 尽快恢复双方通信的效率



- 当TCP开始启动的时候, 慢启动阈值等于窗口最大值;
- 在每次超时重发的时候, 慢启动阈值会变成原来的一半, 同时拥塞窗口置回1;

少量的丢包, 我们仅仅是触发超时重传; 大量的丢包, 我们就认为网络拥塞;
当TCP通信开始后, 网络吞吐量会逐渐上升; 随着网络发生拥堵, 吞吐量会立刻下降;
拥塞控制, 归根结底是TCP协议想尽可能快的把数据传输给对方, 但是又要避免给网络造成太大压力的折中方案.

TCP拥塞控制这样的过程, 就好像 **热恋的感觉**