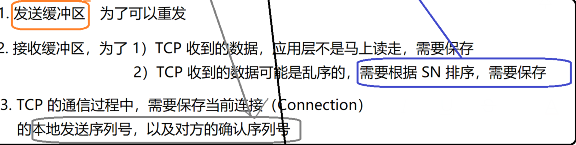
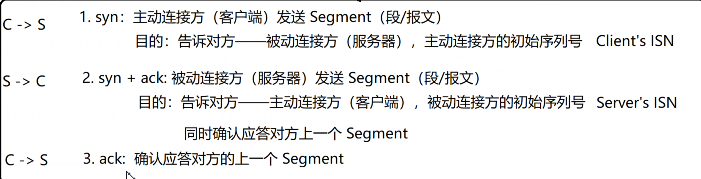
TCP具有发送缓冲区和接收缓冲区的原因：



所以TCP协议栈需要进行连接管理

连接管理：TCP需要知道当前状态

因为两个连接端初始序列号是随机的——所以需要双方事先进行沟通——所以需要进行三次握手



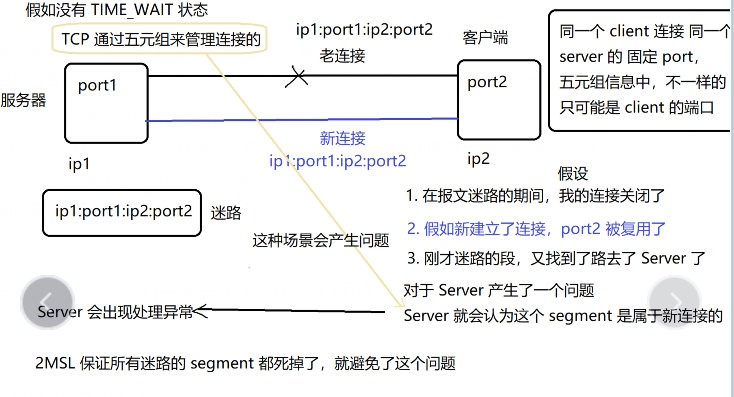
因为TCP是全双工的——（任何一方都可以关闭连接）需要四次握手

**TIME\_WAIT:**

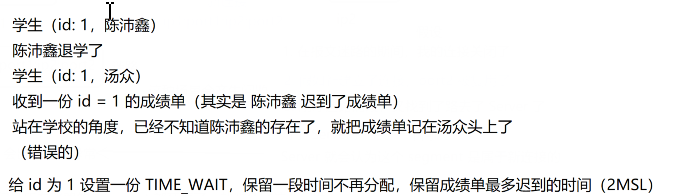
TIME\_WAIT为什么是2MSL?

MSL是TCP的最长生存期，一定大于超时重传的超时时间

保证在两个传输方向上的尚未被接收或迟到的报文段都已经消失(防止数据错误）



在理论上保证最后一个报文可靠到达



**CLOSE\_WAIT**

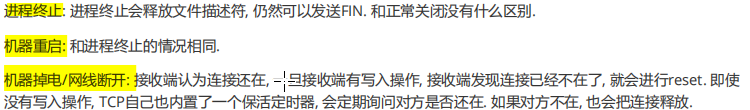
为什么出现CLOSE\_WAIT状态？

收到对方同意关闭的FIN，而自己迟迟不关闭

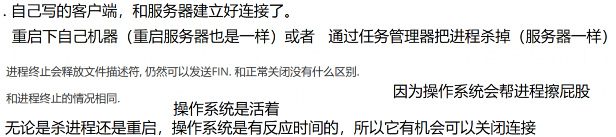
错误的原因

忘记调用socket.close()，对方调用了socket.close()，而且进程常年不关闭（进程关闭时，OS会自动释放一部分资源，其中就包括TCP连接）导致很多处于半连接状态（只有一方关闭连接）

TCP异常情况：



进程终止/机器重启



拔插销（电脑猝死）操作系统没有反应时间

只有写数据才能发现异常，读数据不会发现

a.B的应用层写数据

A重启好了，收到一份B发来的数据，但是却不认识，会给B发送一个RST异常.B进行关闭连接

A没有重启好，不会回复B,B进行超时重传，超时重传到达一定程度后，给A发送RST

b.B的应用层读数据

如果B没有设置超时时间，则一直阻塞

解决办法：

TCP层：定期（2h)给对方发送数据（TCP内部的KeepAlive机制）

应用层：

设置超时时间

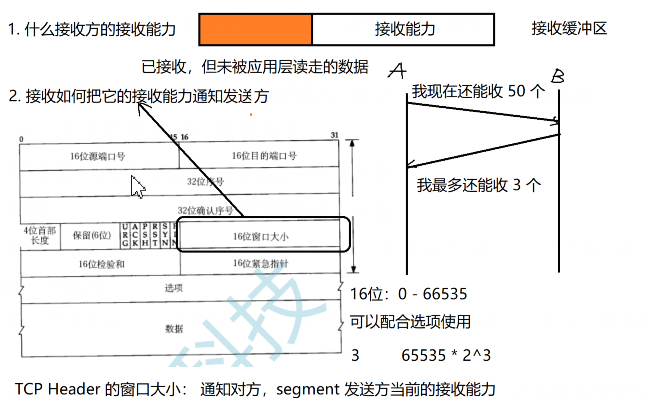
定期给连接写数据（定期时间可控）

**流量控制**：TCP支持根据接收端的处理能力, 来决定发送端的发送速度.

问题：一次发送的量与那些因素有关？

与接收方的接收能力 流量控制（先有）

网络 拥塞控制（后有）



窗口大小：与选项中的偏移值有关，是动态变化的

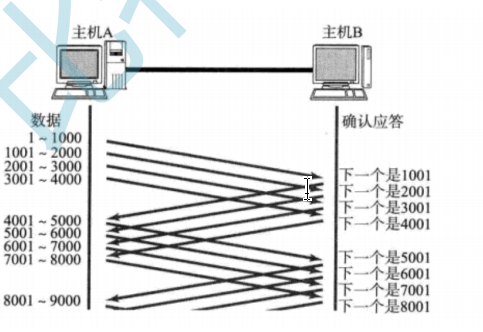
接收端将自己可以接收的缓冲区大小放入 TCP 首部中的 "窗口大小" 字段, 通过ACK端通知发送端;

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;

如果缓冲区满了, 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.

**滑动窗口**：描述发送方的发送缓冲区

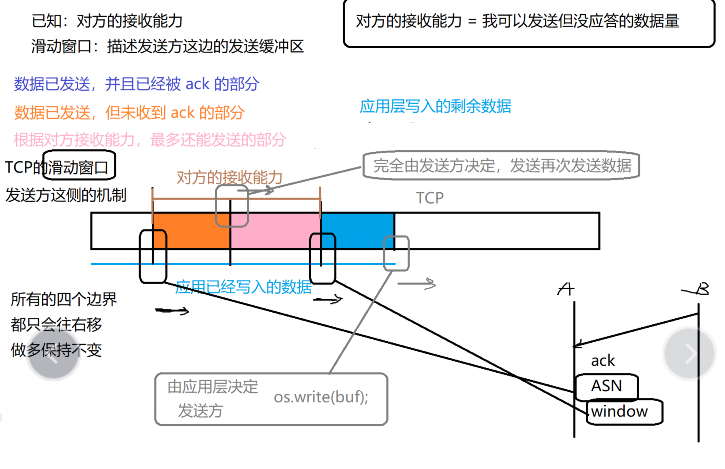
窗口大小：指的是无需等待确认应答而可以继续发送数据的最大值



上图所示 窗口大小为4000

发送前四个段的时候, 不需要等待任何ACK, 直接发送;

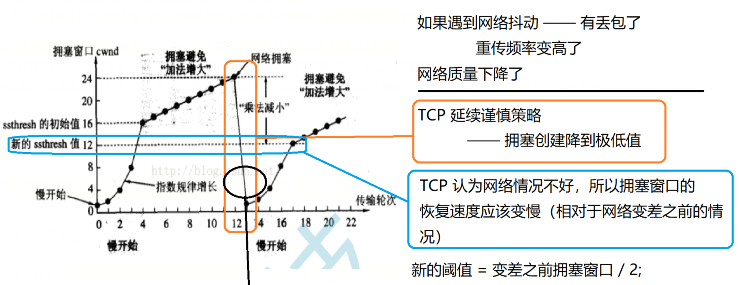
收到第一个ACK后, 滑动窗口向后移动, 继续发送第五个段的数据;



**拥塞控制**：由于网络上有很多计算机，所以TCP采取慢启动机制

发送方能发送的数据=min(接收能力，拥塞窗口）

为了不增长的那么快,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



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;

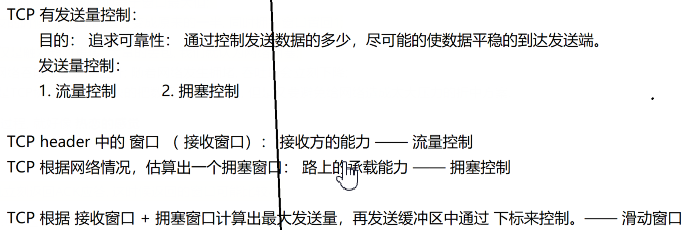
每次发送数据包的时候, 将拥塞窗口和接收端主机反馈的窗口大小做比较, 取较小的值作为实际发送的窗口;

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;

总结：



快重传：在数据丢包的情况下，接收端会发送对视的数据报ASN,如果多次受到ASN,则进行重传。

**延迟应答**：TCP不会以受到数据马上ack

1.后序可能马上还有数据过来（两个ack进行合并，节省带宽）

2.和捎带应答有挂。将ack与向发送端发送的数据合并

**捎带应答**：为了减少发送报文，尽量合并有效数据和ack应答

**粘包问题**：因为TCP面向字节流

在TCP的协议头中, 没有如同UDP一样的 "报文长度" 这样的字段, 但是有一个序号这样的字段.

站在传输层的角度, TCP是一个一个报文过来的. 按照序号排好序放在缓冲区中.

站在应用层的角度, 看到的只是一串连续的字节数据.

