一般来说，我们总是希望数据传输得更快一些，但是往往事与愿违，如果发送方把数据发送得太快，接收方就有可能来不及接收，继而丢弃数据。

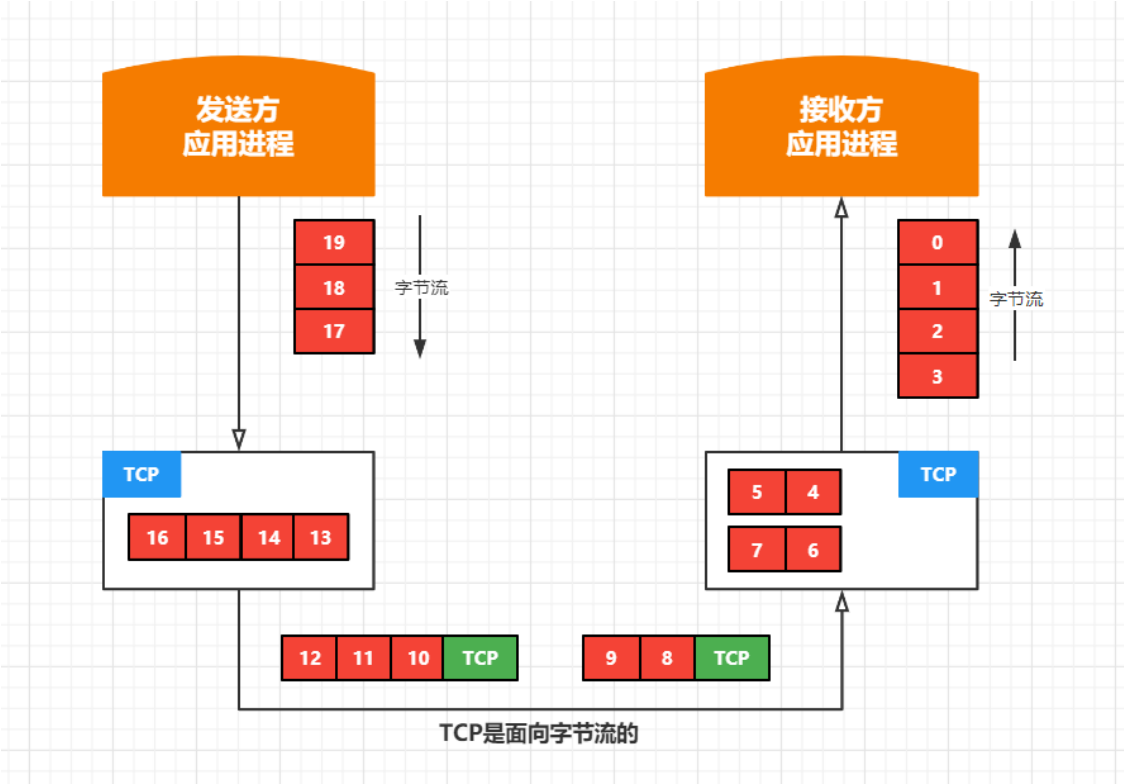
所谓流量控制，就是让发送方的发送速度不要太快，要让接收方来得及接收。

利用滑动窗口机制可以很方便地在TCP连接上实现对发送方的流量控制。

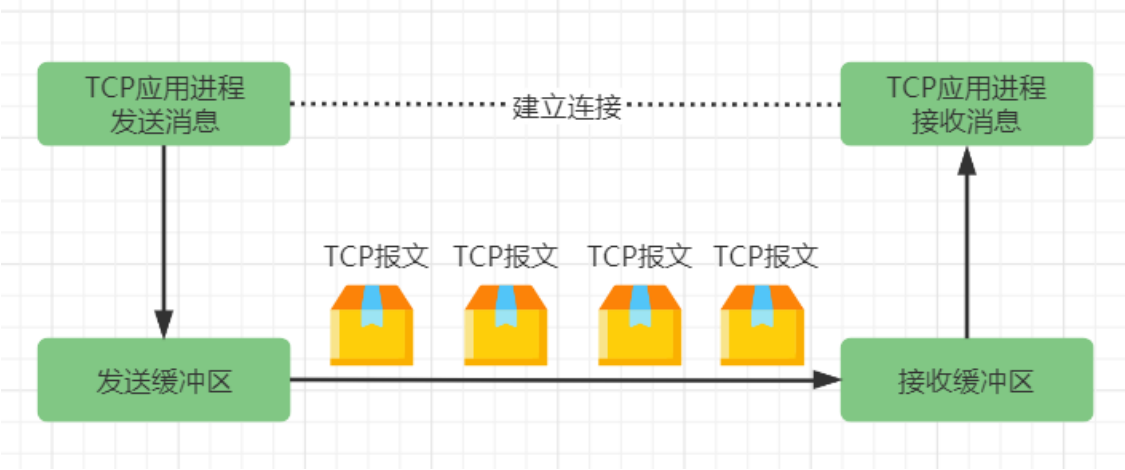
关于窗口的概念，我们在《11｜数据链路层篇：可靠传输问题（下）》中已经介绍过，本篇文章我们再来回过头来说一说，毕竟滑动窗口是TCP中一个非常重要的概念。

一、流量控制

还记得这张图吗?



TCP 会把要发送的数据放入发送缓冲区（Send Buffer），接收到的数据放入接收缓冲区（Receive Buffer），应用程序会不停的读取接收缓冲区的内容进行处理。



流量控制做的事情就是，如果接收缓冲区已满，发送端应该停止发送数据，为了控制发送端的速率，接收端会告知发送端自己的接收窗口（rwnd），也就是接收缓冲区中空闲的部分。

总结：接收方依据自身接收缓冲区的大小告知对方自己的接收窗口，来控制对方发送的速率，避免自己被“喂撑”。

二、发送窗口和接收窗口

很多人认为发送窗口就是接收窗口，其实不然。

实际上发送窗口的大小要在自身拥塞窗口和接收方指定的接收窗口中取小者，关于拥塞窗口将在下篇文章中介绍，只要清楚发送窗口大小可不是等于接收窗口的。

回到接收窗口，在wireshark抓包中可直观看到，即Win字段：

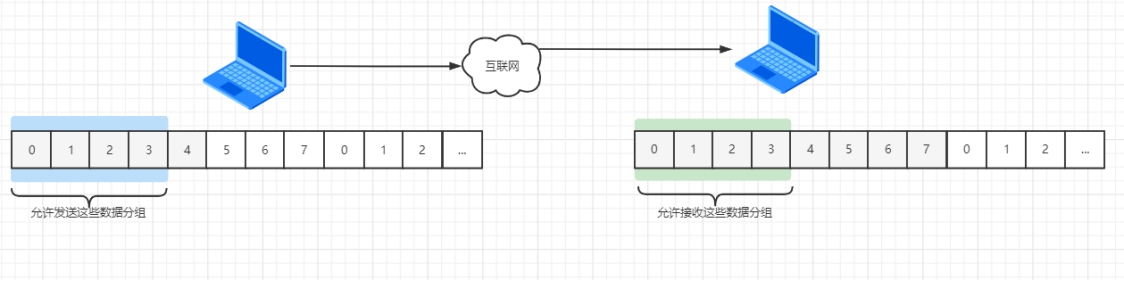
```
25 2021-10-05 13:28:27.182056   192.168.101.2       180.101.49.12       TCP        66 3006 → 80 [SYN] Seq=2384228516 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
26 2021-10-05 13:28:27.191515   180.101.49.12       192.168.101.2       TCP        66 80 → 3006 [SYN, ACK] Seq=2010050501 Ack=2384228517 Win=8192 Len=0 MSS=1412 WS=32 SACK_PERM=1
27 2021-10-05 13:28:27.191687   192.168.101.2       180.101.49.12       TCP        54 3006 → 80 [ACK] Seq=2384228517 Ack=2010050502 Win=131072 Len=0
28 2021-10-05 13:28:27.192063   192.168.101.2       180.101.49.12       HTTP      131 GET / HTTP/1.1
29 2021-10-05 13:28:27.200702   180.101.49.12       192.168.101.2       TCP        60 80 → 3006 [ACK] Seq=2010050502 Ack=2384228594 Win=29056 Len=0
30 2021-10-05 13:28:27.202629   180.101.49.12       192.168.101.2       TCP      1466 80 → 3006 [ACK] Seq=2010050502 Ack=2384228594 Win=29056 Len=1412 [TCP segment of a reassembled PDU
31 2021-10-05 13:28:27.202630   180.101.49.12       192.168.101.2       HTTP     1423 HTTP/1.1 200 OK  (text/html)
32 2021-10-05 13:28:27.202759   192.168.101.2       180.101.49.12       TCP        54 3006 → 80 [ACK] Seq=2384228594 Ack=2010053283 Win=131072 Len=0
33 2021-10-05 13:28:27.203569   192.168.101.2       180.101.49.12       TCP        54 3006 → 80 [FIN, ACK] Seq=2384228594 Ack=2010053283 Win=131072 Len=0
34 2021-10-05 13:28:27.210856   180.101.49.12       192.168.101.2       TCP        60 80 → 3006 [ACK] Seq=2010053283 Ack=2384228595 Win=29056 Len=0
35 2021-10-05 13:28:27.211049   180.101.49.12       192.168.101.2       TCP        60 80 → 3006 [FIN, ACK] Seq=2010053283 Ack=2384228595 Win=29056 Len=0
36 2021-10-05 13:28:27.211126   192.168.101.2       180.101.49.12       TCP        54 3006 → 80 [ACK] Seq=2384228595 Ack=2010053284 Win=131072 Len=0
```

这里的29056表示向对方声明自己的接收窗口的大小，对方收到以后，会把自己的「发送窗口」限制在29056大小之内。如果自己的处理能力有限，导致自己的接收缓冲区满，接收窗口大小为 0，发送端应该停止发送数据。

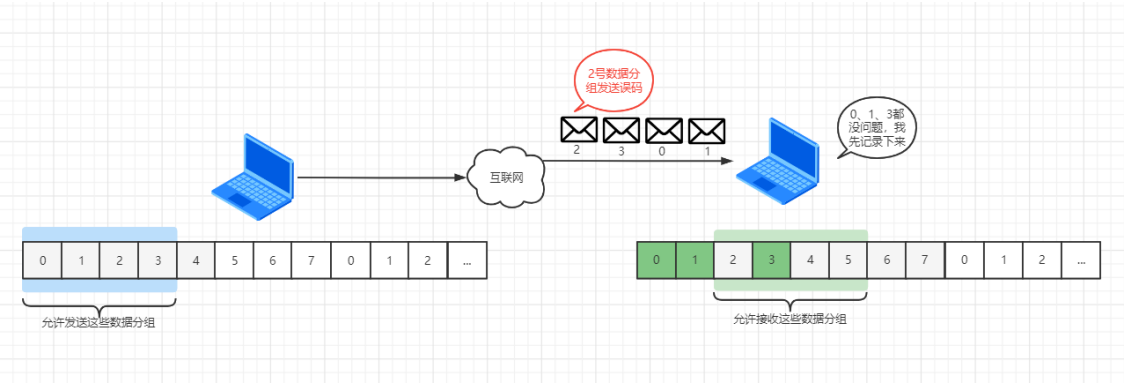
> PS：在本文中，暂不考虑拥塞窗口影响，将发送窗口设置为接收窗口大小。

三、滑动窗口工作的基本过程

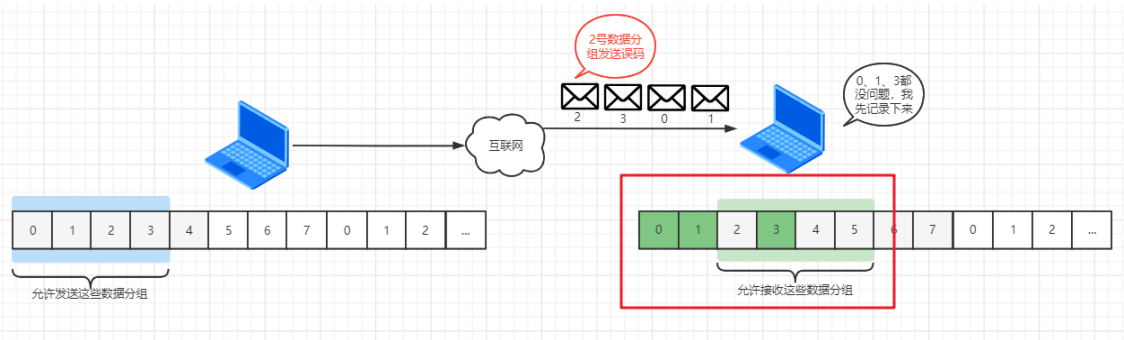
如上所述，窗口分为发送窗口和接收窗口，假设发送窗口和接收窗口大小都为4。



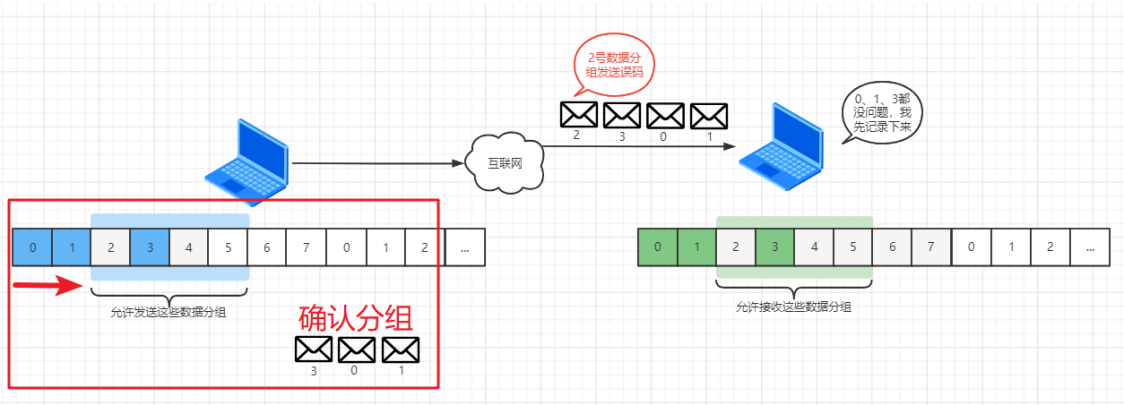
正常情况比较简单，两边窗口同时往前移动即可，但是当某个数据分组出现误码了，如何处理呢？



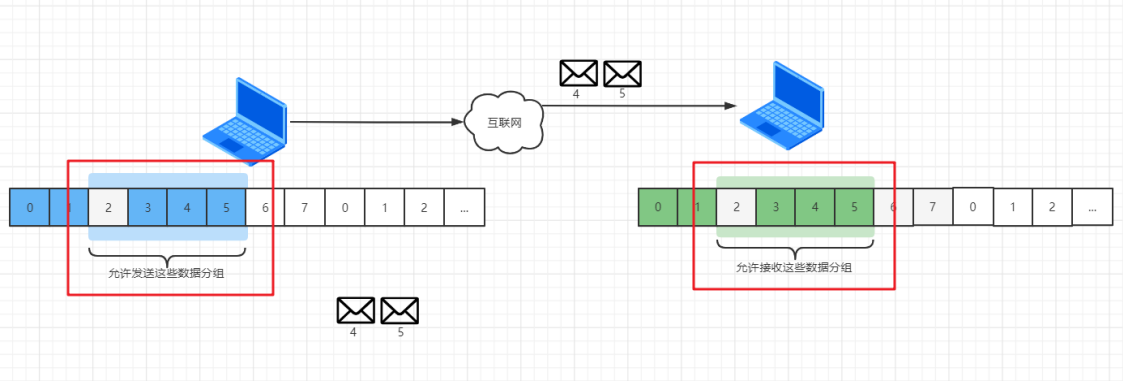
比如本例中序号为2的数据分组出现了误码，接收方判断是误码后进行丢弃，其他的数据分组可正常接收，0和1号分组是正常接收的，那么接收窗口往后移动两个位置：



并且针对0、1、3分别给出确认分组，发送方进行窗口滑动，不过由于2号分组还未收到确认，因此发送窗口只能往后移动两个位置：

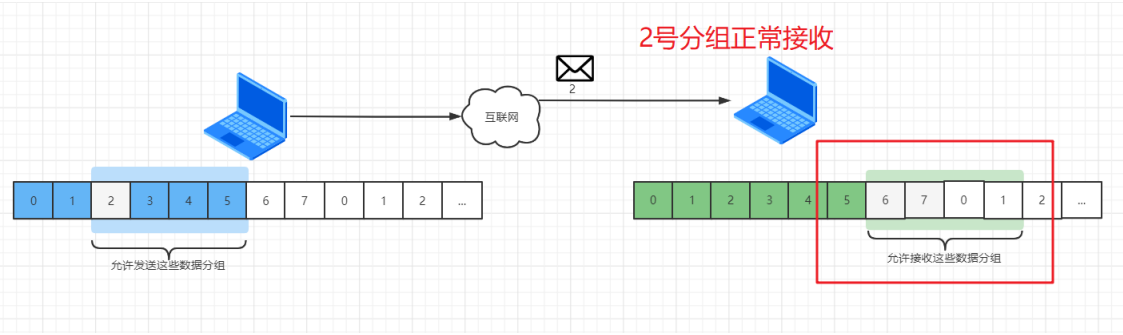


下面发送方将继续发送4、5号数据分组，2号分组需要等待超时重发，可能会发生4、5号数据分组先得到确认的情况：

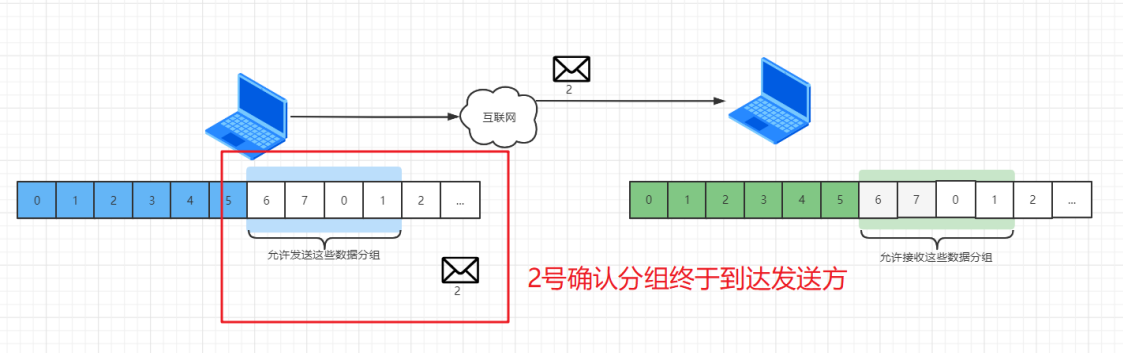


由于2号数据分组此时还未重发，所以发送窗口和接收窗口都不能往后滑动！此时发送方已记录3、4、5数据分组都已正常发送和接收，因此不会触发其重发。

等待一段时间后，2号数据分组超时重发，接收方收到2号分组后，发回2号确认分组，接收窗口往后滑动4个位置：



若发送方收到2号确认分组，那么发送窗口也往后移动4个位置：



下面就有6、7、0、1号数据分组落入发送窗口，那么就继续如上的过程发送即可。

四、滑动窗口区域分类

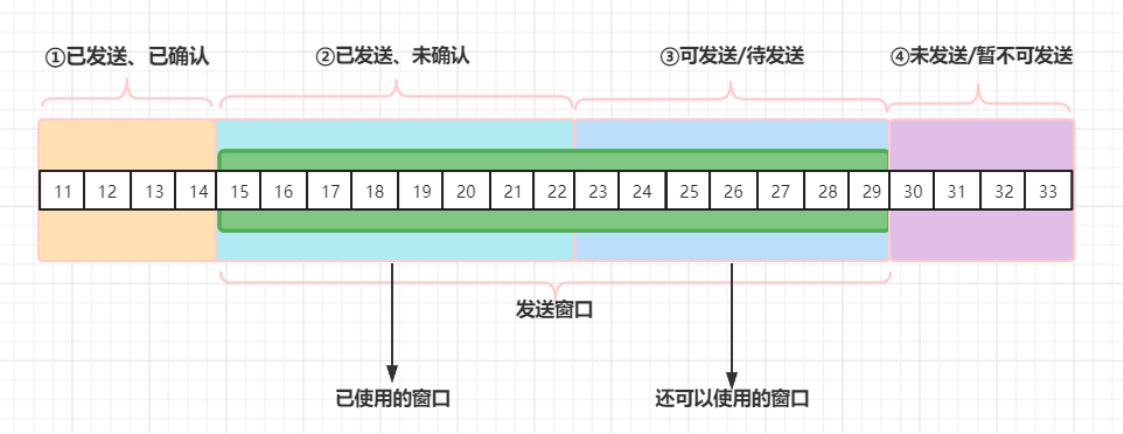
上述过程中，我们可以发现，只有处于发送窗口内的数据才有资格被发送出去，与之对应，只有处于接收窗口内的数据才有资格被确认，说明窗口具有不同功能的区域，下面我们再来看看窗口。

我们以发送窗口为例，发送窗口是发送端被允许发送的最大数据包大小。而处于发送窗口内的数据，可能有两种状态：

- 已发送，但未收到确认
- 即将发送，接收方是有缓存接收的

容易想到，窗口前面的应当是已发送并已确认的数据，窗口后面为现在不能发送的数据。

那么整个发送窗口就可以被划分为四部分：



其中区域②和区域③都属于发送窗口，只有处于这个范围内的数据才可以被发送出去。

其中区域③是可用窗口，很容易理解，可用窗口是发送端还能发送的最大数据包大小。

当可用窗口大小变为0时，只能等待受到接收端的ACK确认报文后，往后移动发送窗口，才能将后续的数据发送出去，否则只能等待。

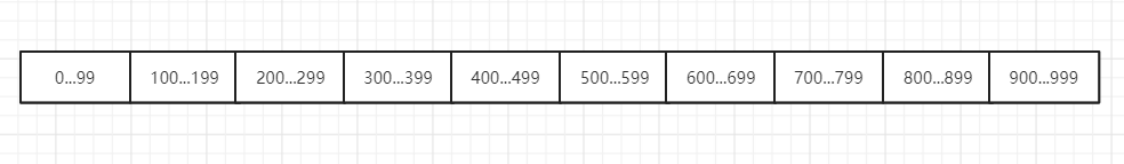
对应着窗口的分类，那么**TCP报文自然也被分为四种**：

- 区域①：**表示已发送且已收到 ACK 确认的数据包。**
- 区域②：**表示已发送但未收到 ACK 的数据包，发送方不确定这部分数据对端有没有收到**，如果在一段时间内没有收到 ACK，发送端需要重传这部分数据包。
- 区域③：**表示未发送但接收端已经准备就绪可以接收的数据包**（有空间可以接收）。
- 区域④：**表示还未发送，且这部分接收端没有空间接收。**

五、流量控制过程详解

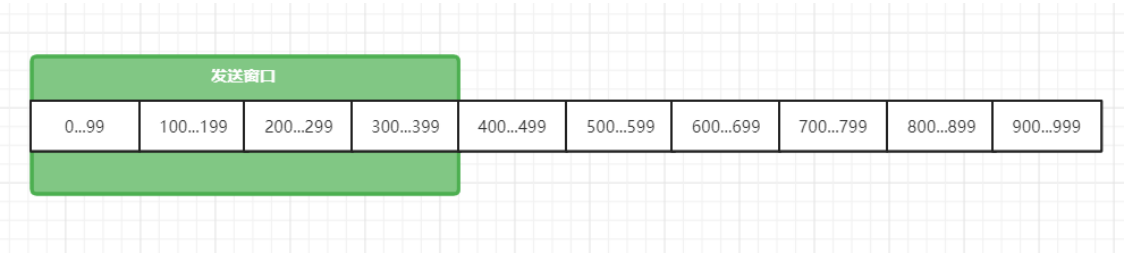
假设有两台主机A和B，目前已建立TCP连接，A往B发送数据，此时B对A进行流量控制。

下图是主机A中待发送数据的字节序号，假设主机A发送的每个TCP报文段可携带100字节数据，因此每个报文段的数据对应每个小格子中的数据序号。

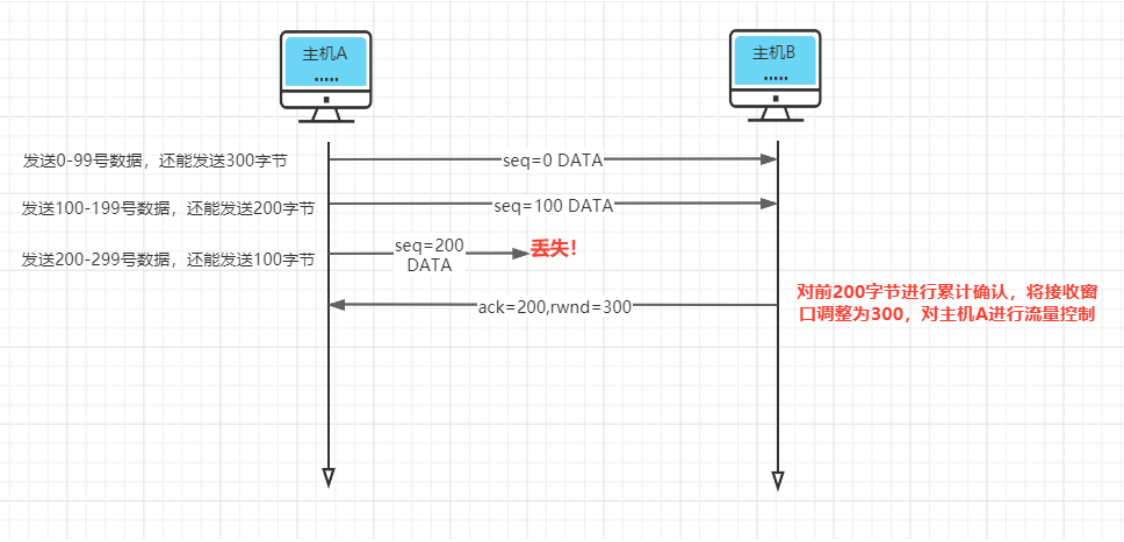


在建立连接时，B告诉A：“我的接收窗口为400字节”，这里我们同样暂时不考虑拥塞窗口等其他因素，那么A的发送窗口被限制为400（0-99序号理解为对应100字节）。

此时，主机A中0-99，100-199，200-299，300-399，这四个格子对应的序号的数据落入发送窗口中，可不等主机B的ACK确认即可一口气发送出去。

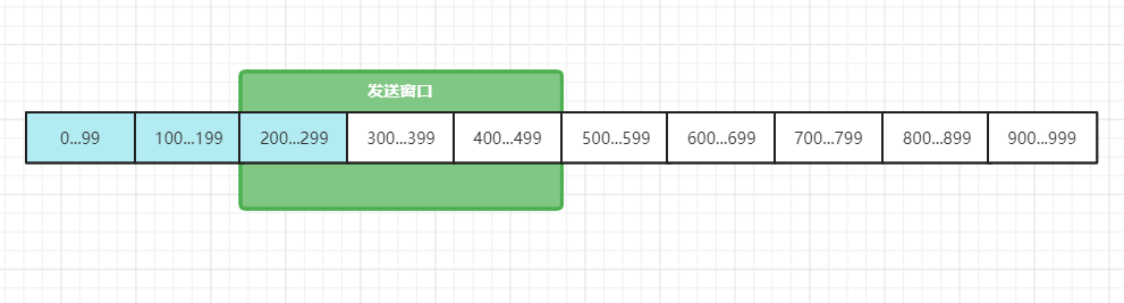


假设此时主机B成功接收到了0-99、100-199，但是主机A在发送200-299数据报时丢失，**主机B对前200个字节进行累计确认，并且告诉主机A，我现在的接收窗口调整为300了**。

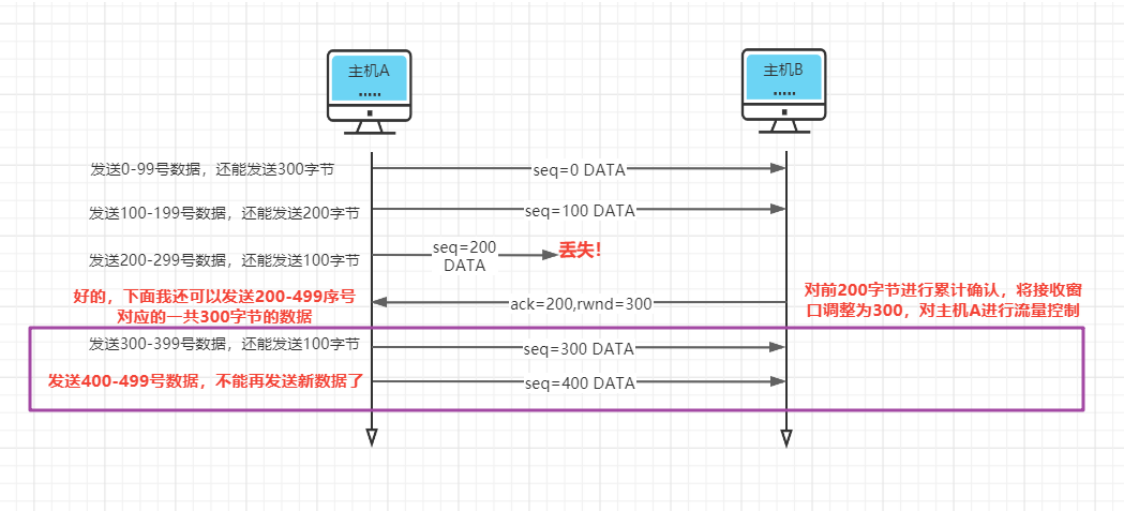


主机A受到来自主机B的ACK确认报文段，发现前200字节的数据已确认接收，那么将发送窗口后移，使得前面已发送并受到确认的数据序号移出发送窗口。

由于主机B在ACK报文中将自己的接收窗口调整为了300，那么主机A相应地将自己的发送窗口调整为300，目前处于主机A发送窗口的数据为200-299、300-399、400-499共300字节的数据，其中200-299是已发送但未收到ACK确认的数据，若重传计时器超时，200-299这100字节将被重传；其他200字节是待发送数据。

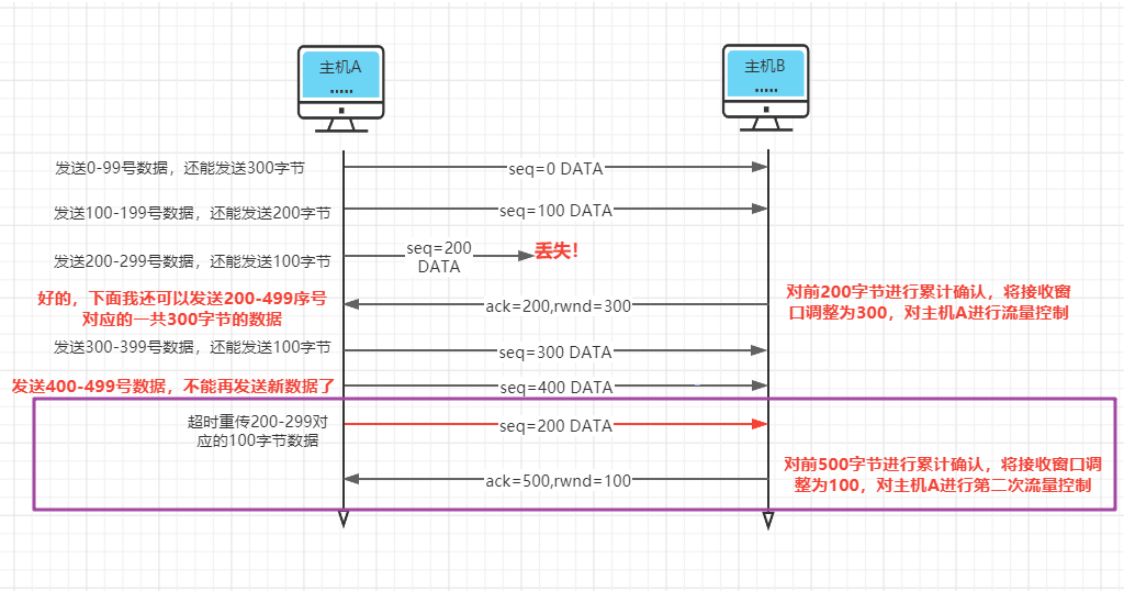


接下来主机A将继续发送300-399、400-499共200字节的数据，发送完后，假如还未收到任何确认报文，这个时候落在发送窗口内的所有数据都已经发送过了，不能继续发送新的数据了。



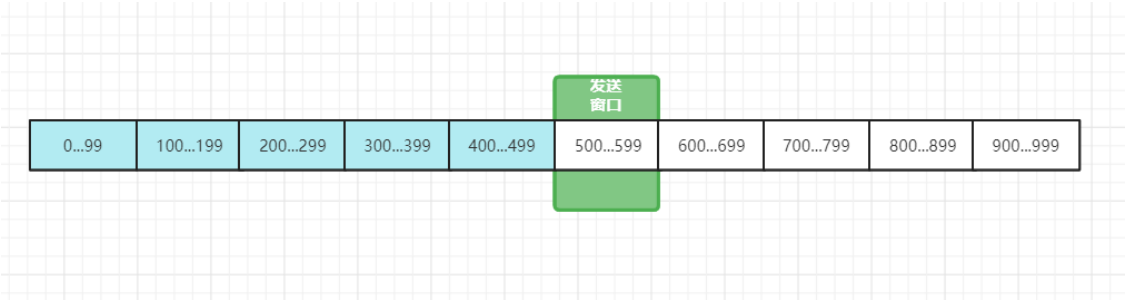
当超时重传计时器到时间后，发现200-299序号的100字节数据仍然未收到来自主机B的确认，进行超时重传，理想情况下，主机B收到主机A发送的重传报文段后，对主机A发送的500号以前的数据进行累计确认，即ack=500。

同时主机B将接收窗口调整为100，对主机A进行第二次流量控制。



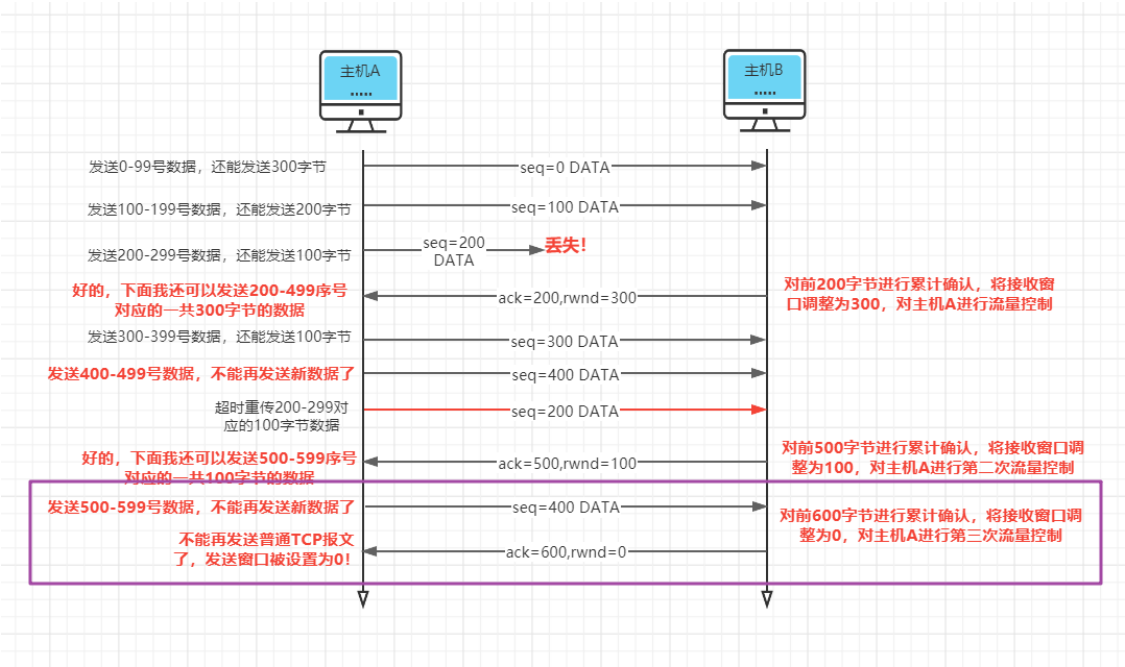
主机A收到累计确认后，发送窗口向后滑动，将前500个已发送并收到确认的这些数据对应的序号从发送窗口中移出出去。并根据最新的接收窗口大小将发送窗口调整为100。

目前处于主机A发送窗口内的数据序号为500-599：



此时主机A只可发送500-599这100字节的数据给主机B，不能发送其他数据了，主机B收到后进行累计确认，返回ack=600，表示序号600前的所有数据都已经接收完毕，下面从600号开始发新数据。

在该累计确认中，**将自己的接收窗口值设置为0，对主机A进行第三次流量控制**。

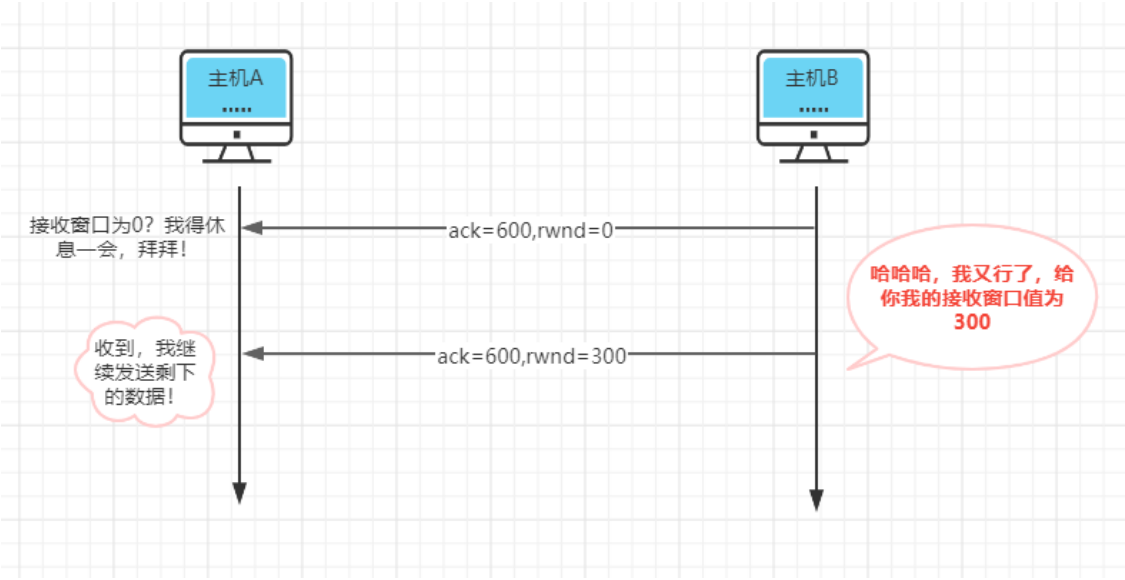


由于主机B返回的接收窗口为0，主机A收到后知道，主机B是遇到困难了，自己的发送窗口也随之调整为0，不再发送普通的TCP数据报文段。

六、零窗口探测报文

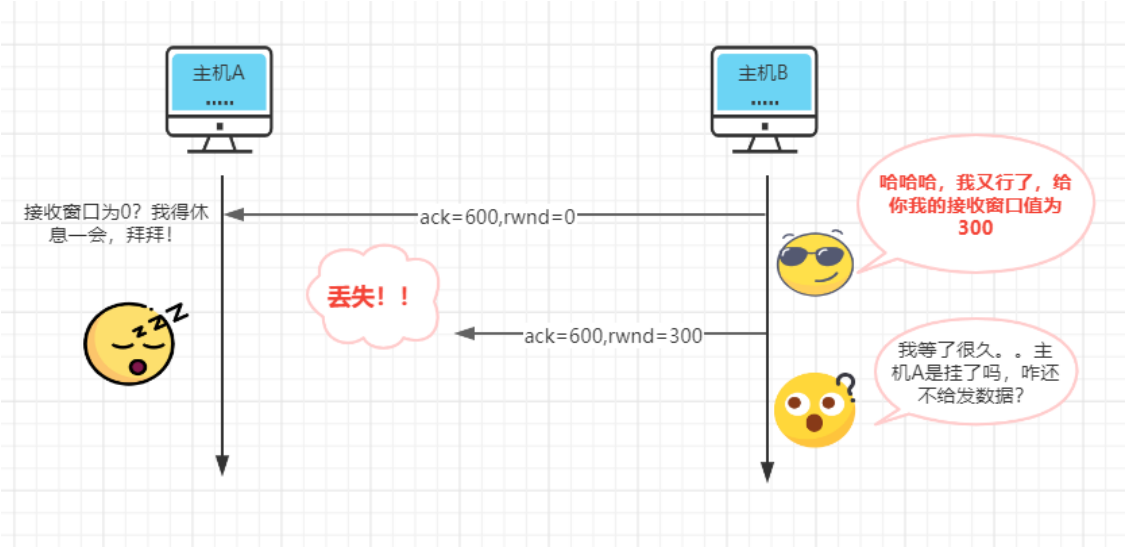
接着上面继续聊，主机A收到了主机B的接收窗口为0的确认报文后，就不能再发送普通的TCP数据报文段了。

不过一段时间后，主机B觉得自己又行了（有接收缓存了），向主机A发送了接收窗口等于300的报文段，正常情况下，主机A收到，就可以设置发送窗口为300并正常发送数据了。



但是如果这个携带非零窗口值的报文段在传输过程中丢失了呢?

主机A一直等待主机B发送的非零窗口的通知，但是主机B已经发送过非零窗口了，只是不知道丢失了，两边都开始进行漫长的互相等待。

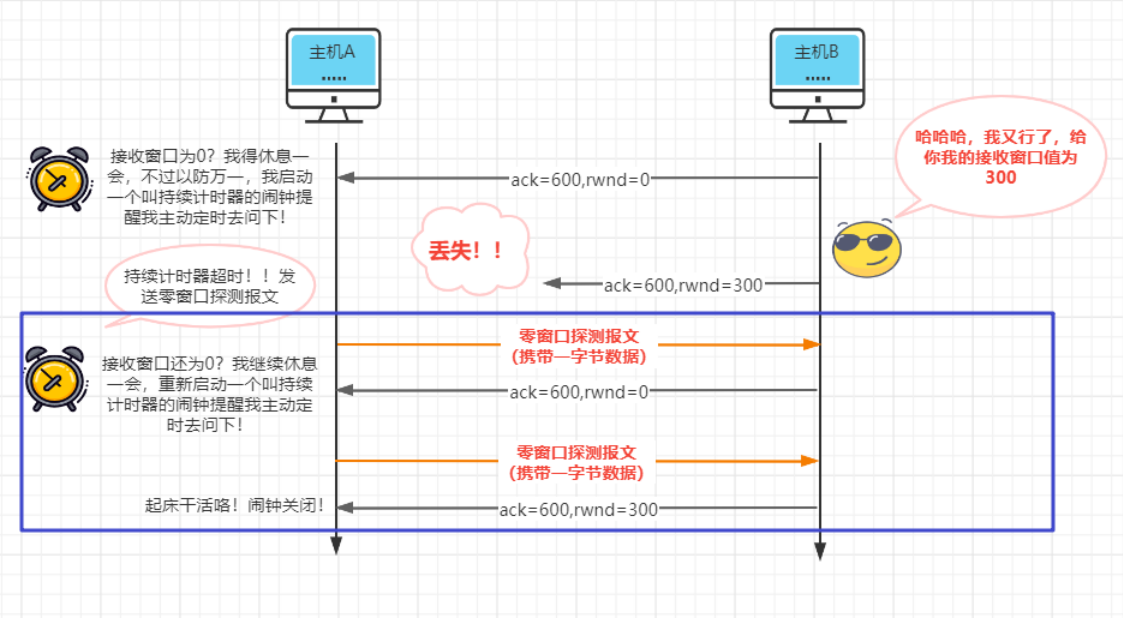


如何打破这个僵局呢?

TCP为每一个连接设有一个持续计时器，只要TCP连接的一方收到对方的零窗口通知，就启动持续计时器，若持续计时器超时，就发送一个零窗口探测报文，仅携带一字节的数据，而对方在确认这个零窗口探测报文段时，给出自己现在的接收窗口值。

如果接收窗口仍然是零，那么收到这个报文段的一方就重新启动持续计时器，如果接收窗口不是0，那么僵局就可以得到打破，当然也就无需再启动持续计时器了！

零窗口探测报文整体工作流程如下图所示：



一个小问题：主机B如果接收窗口为0，还能接收零窗口探测报文吗？答案当然是可以，TCP规定，即使接收窗口为0，也必须接收零窗口探测报文段、确认报文段，以及携带有紧急数据的报文段。

另一个问题：如果零窗口探测报文段丢失呢？是否还能打破以上僵局？答案是可以，因为零窗口探测报文段同样也有超时重传计时器。

OK，本文完，下一篇我们来聊聊拥塞控制的内容。