可靠传输机制

前情回顾

TCP发送的报文段是交给IP层传送的，但IP层只能提供尽最大努力服务。也就是说，TCP下面的网络所提供的是不可靠的传输。因此，TCP必须采用适当的措施才能使得两个传输层之间的通信变得可靠。

理想的传输条件有以下特点：

> (1) 传输信道不产生差错。
> (2) 不管发送方以多快的速度发送数据，接收方总是来得及处理收到的数据。

在这样的情况下，不采取任何措施就能实现可靠传输。

然而实际的网络都不具备上述的两个条件。但是我们可以使用一些可靠传输协议，当出现差错时让发送方重传出现差错的数据，同时在接收方来不及处理收到的数据时，及时告知发送方降低数据发送的速度——**流量控制**。这样本来不可靠的信道就能实现可靠的传输。

> 注：在计算机网络发展初期，通信链路不太可靠，因此在链路层传输数据时都要采用的可靠的传输协议（如停止等待协议），同样出传输层同样也需要某种机制来保证传输层的可靠传输。如今通信链路质量变得可靠，通信出现差错的可能性就变小，所以现在也就不要求数据链路层要实现可靠传输，即数据链路层就抛弃了可靠传输的责任，交给传输层实现。而链路层则主要负责差错控制，这样就会让数据在链路层上传输速度更快，延迟更小。

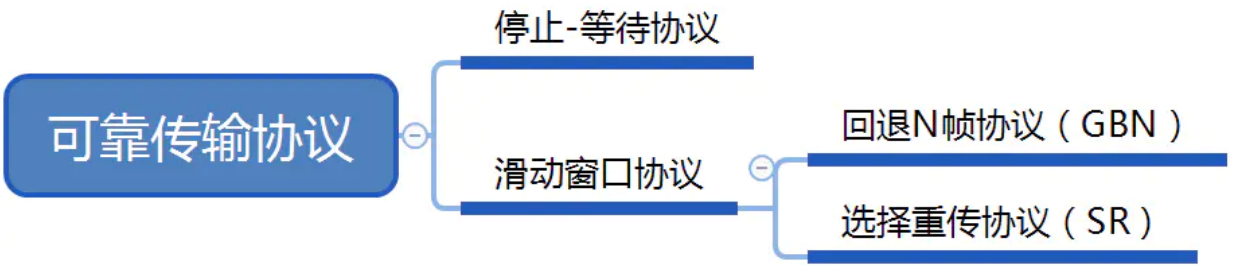
1.可靠传输

可靠传输：即发送端发送的是什么，接收端接收到的就是什么。也就是在数据传输的过程中没有传输差错。

数据传输过程中可能出现的问题：**比特出差错、丢包。**

> 丢包：物理线路故障、设备故障、病毒攻击、路由信息错误等原因，导致的数据包的丢失情况。

可靠传输的方法：**停止-等待协议、滑动窗口协议。**



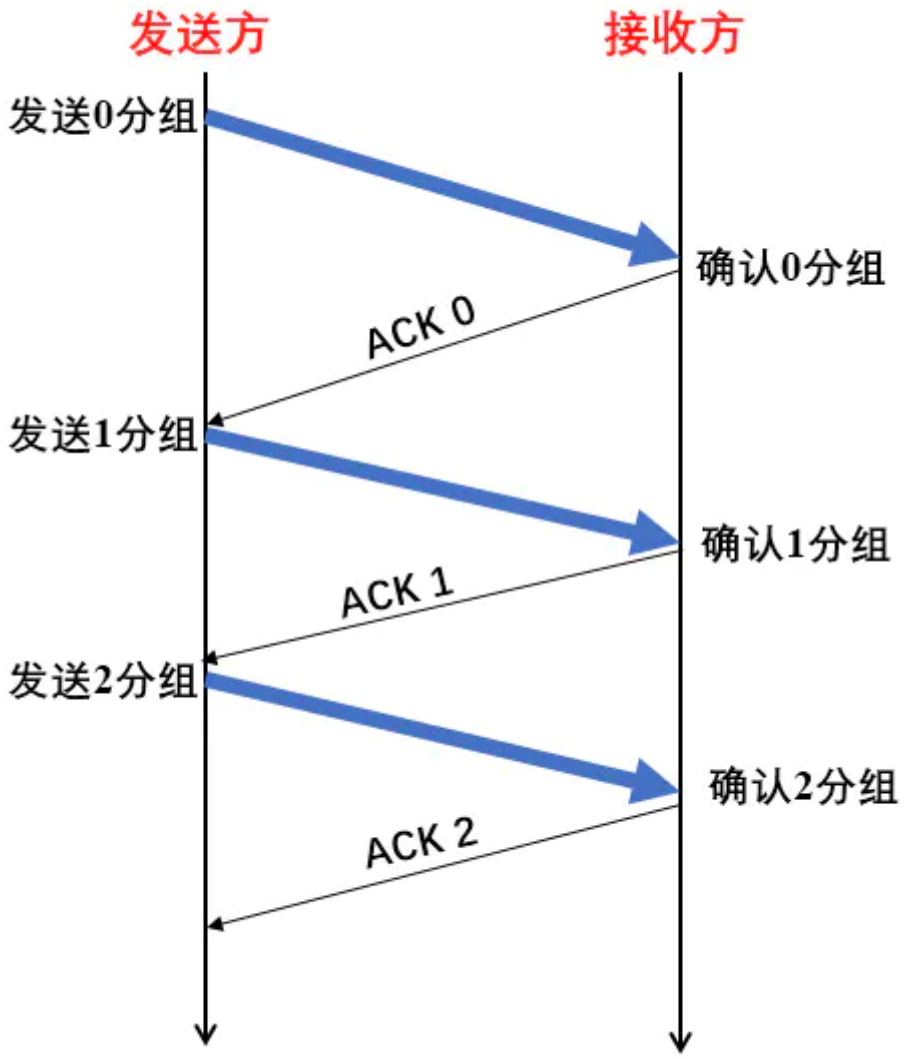
2.停止-等待协议

2.1 前提

(1) 全双工通信的双方既是发送方也是接收方，但是为了讨论问题方便，仅考虑一方发送数据（发送方），一方接收数据（接收方）。
(2) 数据传输的过程中既可能出现差错，又有可能出现丢包的现象。
(3) 所有传输的数据单元称为分组。

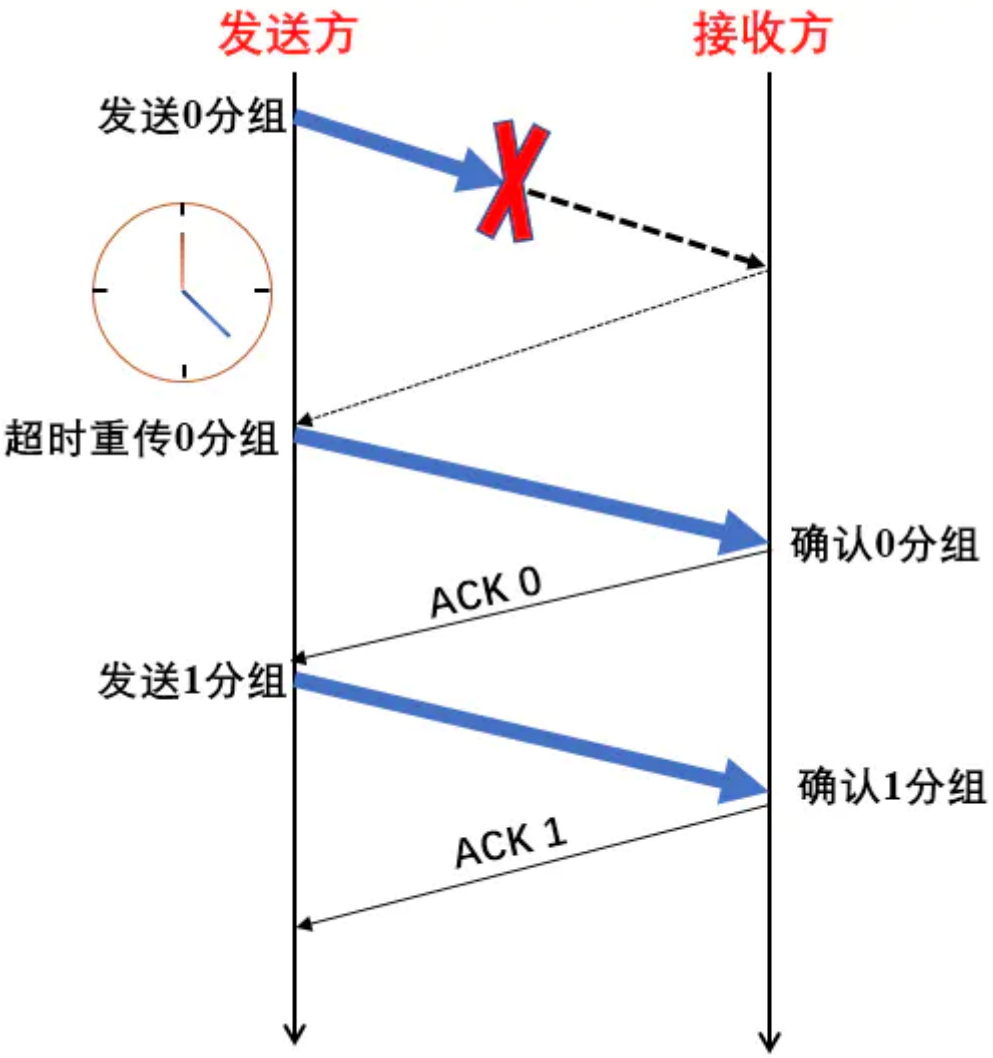
2.2 无差错情况

发送方每发送一分组就停止等待，只有等到了接收方发送来的确认后才可以发送下一个分组。



2.2 有差错情况

(1) **分组丢失或检测到分组出错** 发送方在传输过程中分组丢失，这时接收方没有接收到分组不会知道发送方发送了分组，就不会给发送方返回确认信息（分组出错，接收方会直接丢弃分组也不会返回确认信息），而发送方在等待接收方的确认信息以此来发送下一个分组。所以可靠传输协议这样规定：当发送方**超过一段时间没有仍然没有收到的确认**，就认为刚才的分组丢失了，因而**重传前面的发送过的分组。**这就是**超时重传。**

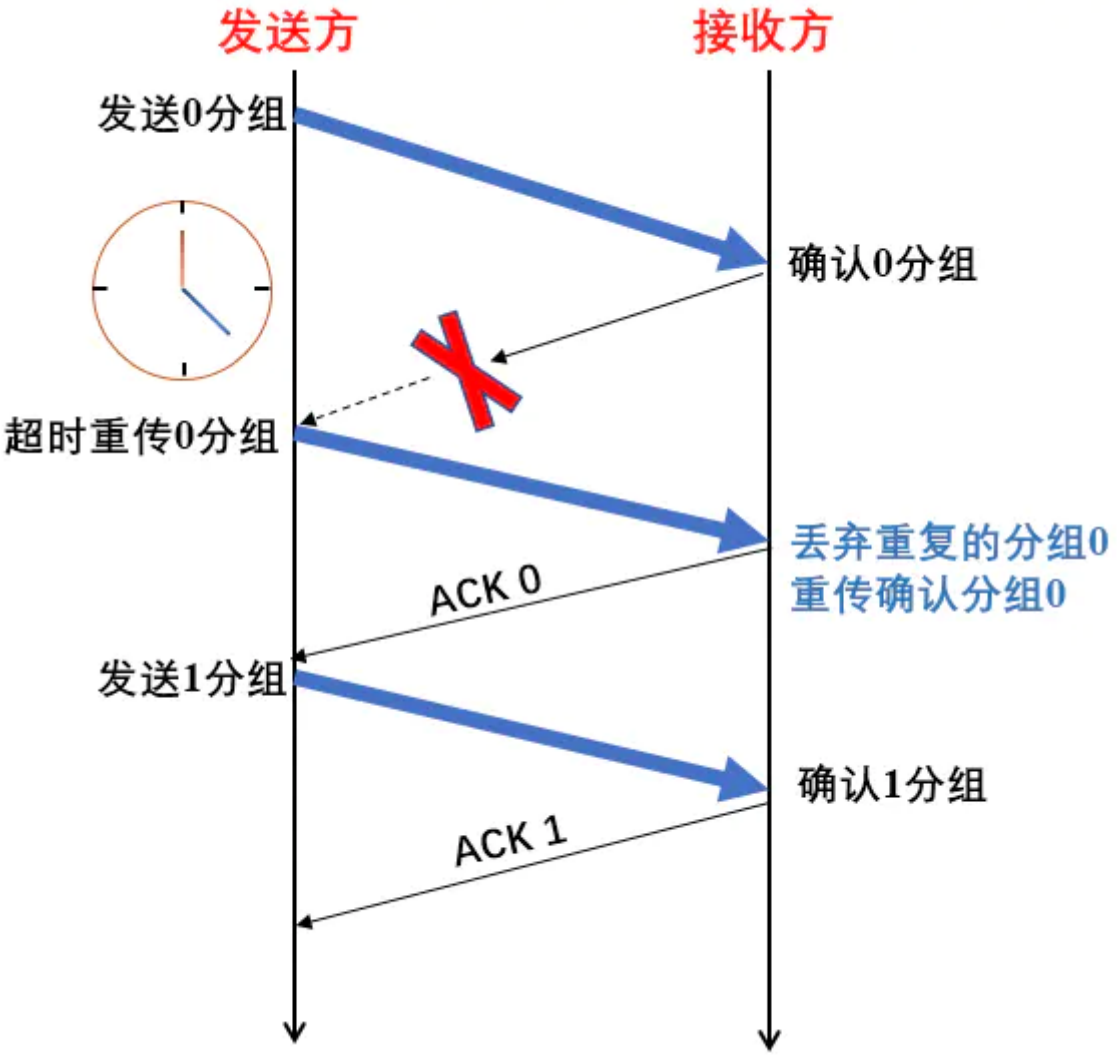


超时重传要求在每次发送完一个分组时设置一个**超时计时器**，如果在超时计时器到期之前收到了对方的确认，就撤销已设置的超时计时器。

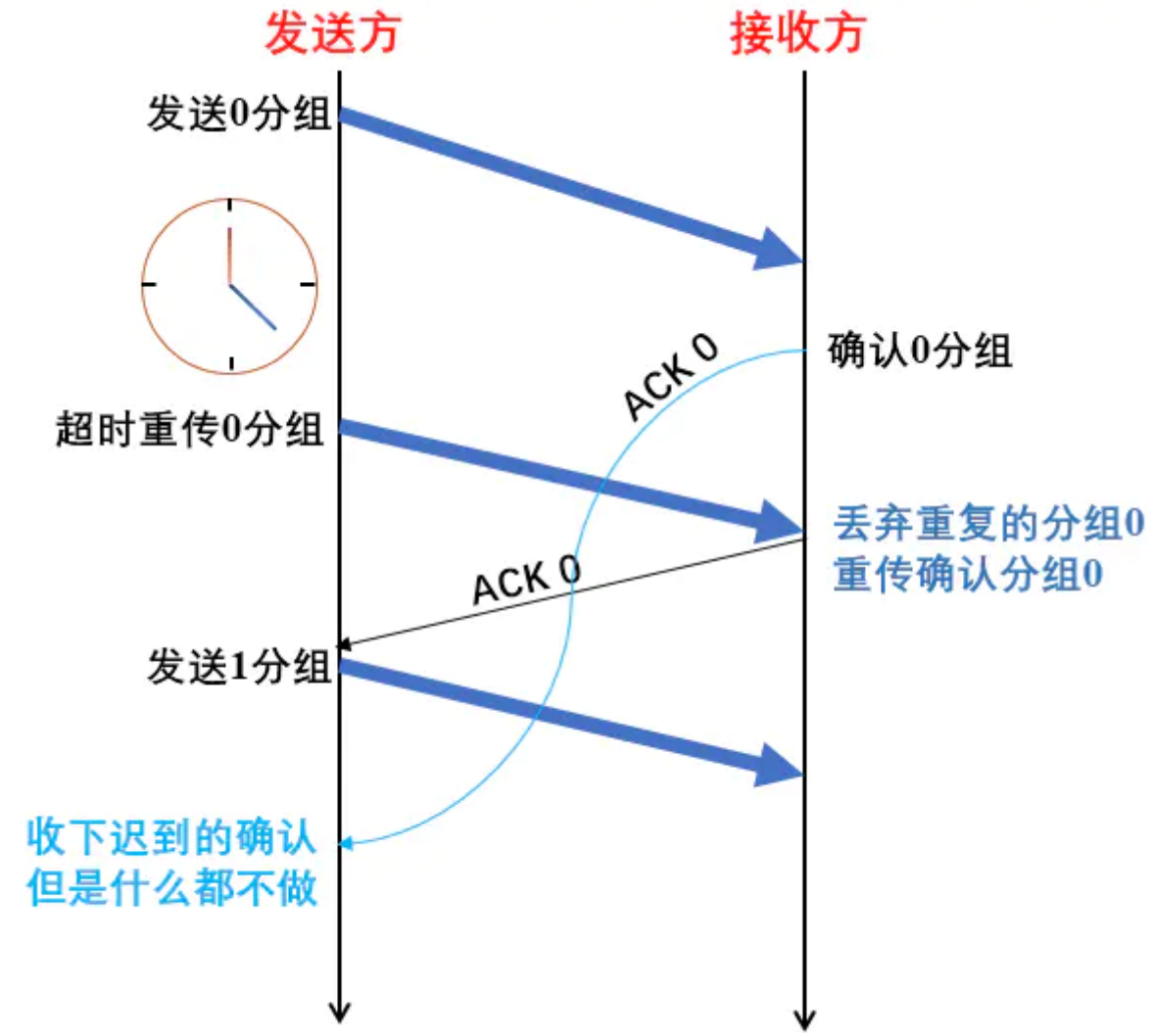
> (1) 超时重传要求发送完一个分组后，**必须暂时保留已发送的分组的副本**（在超时重传时使用），只有收到了相应的确认后才能清除暂时保留的副本。
>
> (2) 超时计时器设置的重传时间应当比数据在分组中传输的平均RTT更长一些。
>
> (3) 分组和确认分组必须进行**编号**。如果接收方发送确认分组丢失（或分组出错），发送方在规定时间内没有收到确认就会超时重传，如果没有编号，那么接收方就不知道这个分组是新的还是已经接收过的，这可能就导致了**分组重复**，所以对分组编号可以很好的解决分组丢失或分组出错等问题。

(2) **确认分组（ACK）丢失**

接收端接收了发送方传输的分组后，向发送方发送ACK，但是ACK在传输过程中丢失了，那么发送方就收不到确认信息，这时发送端超时计时器到期后就要重传分组，接收方仍会重复接收分组，并且丢弃重复的分组，并重传ACK。

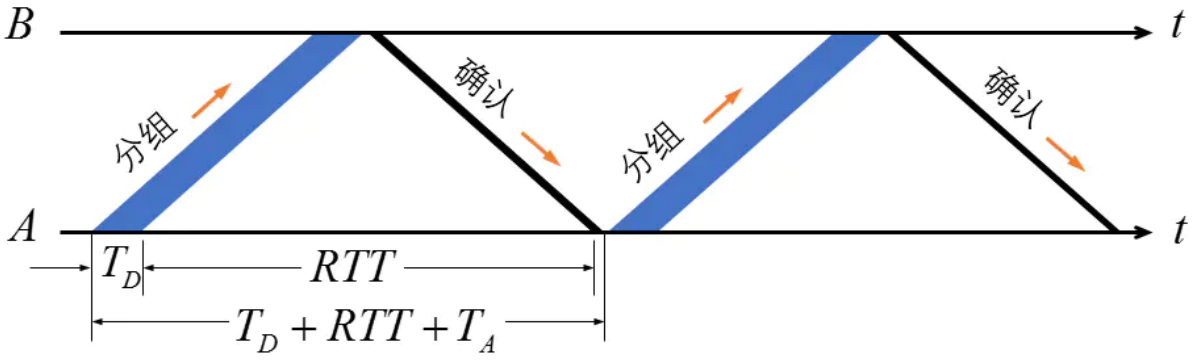


(3) **ACK分组迟到** 下图的传输过程没有出现差错，接收方接收分组后向发送方发送ACK，但是确认信息迟到了，在超时计时器到期后发送方会重新发送分组，接收方仍然会重复接收分组，并且丢弃重复的分组，并重传ACK。当发送方收到ACK后会发送下一个分组，在后来发送方收到迟到的ACK时，发送方会收下并直接丢失。



优缺点：停止等待协议优点是简单，但缺点是信道利用率太低。

2.3 信道利用率



假定A发送分组需要的时间是TD。显然，TD等于分组的长度除以数据率。再假定分组正确到达B后，B处理分组的时间可以忽略不计，同时立即发回确认。假定B发送确认分组需要时间TA，如果A处理确认分组的时间也可以忽略不计，那么A在经过（TD + RTT + TA）后就可以再发送下一个分组，这里的RTT是往返时间。

信道利用率：**发送方在一个发送周期内，有效地发送数据所需要的时间占整个发送周期的比率。**

因为仅仅是在时间TD内才用来传送有用数据（包括分组的首部），因此信道利用率U为：

$$U = \frac{T_D}{T_D + RTT + T_A}$$

> 假定1200km的信道的往返时间RTT = 20ms。分组长度是1200bit，发送速率是1MBit/s。若忽略处理时间和 TA（ TA一般远小于 TD），则信道利用率U = 5.66%。

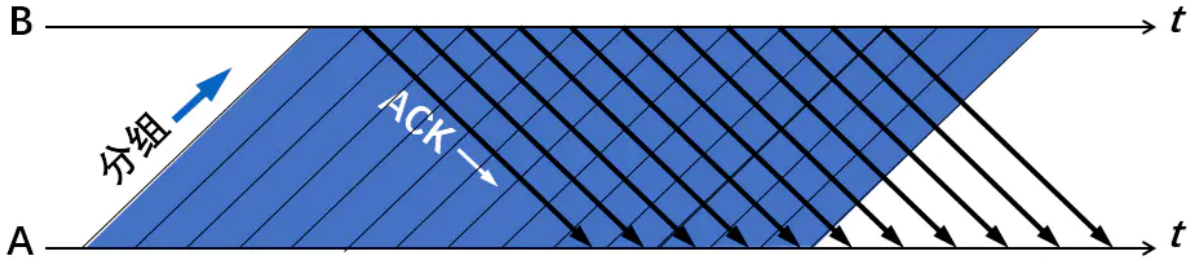
$$T_D = \frac{1200}{10^6} \times 10^3 = 1.2ms$$

$$U = \frac{T_D}{T_D + RTT + T_A} = \frac{1.2}{1.2 + 20} = 0.0566 = 5.66\%$$

2.4 流水线技术

为了提高传输效率，发送方可以不使用低效率的停止等待协议，而是采用**流水线传输。**

流水线传输就是**发送方可以连续发送多个分组，不必每次发完一个分组就停下来等待对方的确认。**这样信道上一直有数据在不断的传送，显然这样会提供信道利用率。



但是流水线技术对可靠传输带来了如下的影响：

(1) **必须增加序号范围。**因为每个传输中的分组必须有一个唯一的序号，而且也许有多个在传输而没有未确认的分组。而在停止等待协议中，由于发送一个分组就必须等待这个分组被确认，所以只需要一个序号即可（如每次传输完成后序号递增）。

(2) 协议的发送方和接收方双方需要缓存多个分组。

(3) 所需的序号范围和对缓存的要求取决于数据传输协议如果处理丢失、损坏及延时过大的分组。解决这些问题有两种基本的方法：**回退N步协议GBN（Go-Back-N）和选择重传协议SR（Selective**

Repeat）。

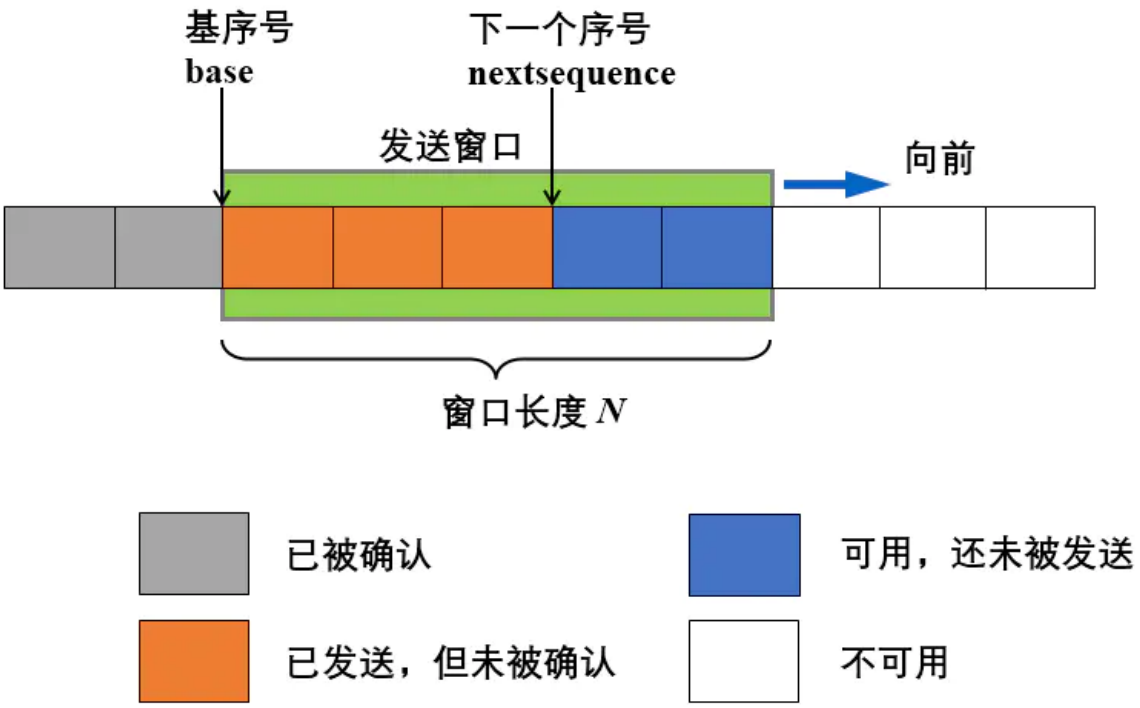
3.回退N步协议GBN

在回退N步协议中，允许发送方发送多个分组而不需等待确认，但它也受限于流水线中未确认的分组数不能超过某个最大允许数N。

发送窗口：**发送方维持的一组连续的允许发送的分组的序号。**凡是发送出现的分组，在没有收到确认之前都需要暂时保留其副本，以便在重传时使用。

> 注：序号是可重用的，这里假设序号的范围是 [0 , 7]。上图中窗口的大小是5，表示最多可以连续发送5个分组而不需等待确认。可见**发送窗口越大，发送方在收到对方确认之前就可以发送更多的数据，因此传输效率也就越高。**但是，窗口的大小也不是无限制的，**窗口的大小受接收方的给出的窗口值和网络拥塞的影响，是动态变化的。**

发送方每收到一个确认，就把发送窗口向前滑动一个分组的位置。如上图所示，发送方收到了对第0分组的确认后，将发送窗口向前移动一个分组的位置。



上图表示了GBN协议的序号范围示意图。其中

> **基序号**：表示最早的未被确认的分组。
> **下一个序号**：表示最小未使用的序号，即下一个待发分组的序号。

序号被分割成了四个部分，各个部分的含义如图所示。

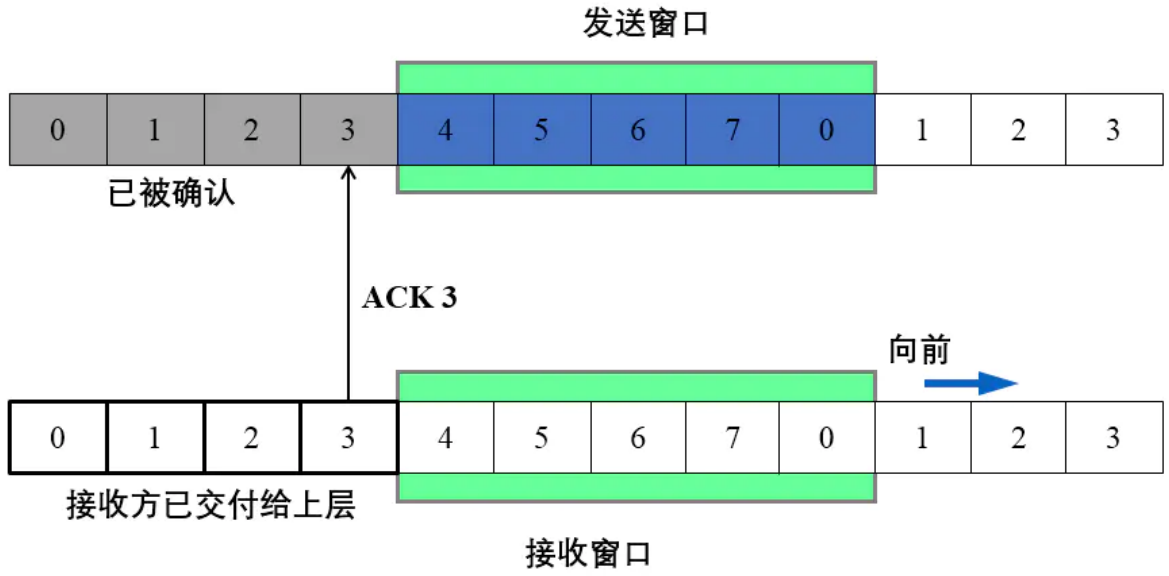
3.1 发送方必须响应的三种类型事件：

(1) 上层的调用。

> 当上层要发送数据时，发送方会先检查发送窗口是否已满，如果未满，则产生一个分组并将其发送；如果窗口已满，发送方只需将数据返回给上层，暗示窗口已满，上层可能要等一会再试。再实际中，发送方可以缓存这些数据，等窗口不满时再发送分组。

(2) **收到一个ACK**：GBN协议中，对序号为n的分组的确认是**采用累积确认**（cumulative acknowledgement）的方式。

> 发送方可以不用每接收一个分组就返回一个确认，当一连串分组按序到达时，发送方可以为这一连串的分组返回一个确认，确认的分组就是按序到达的最大的分组序号。例如，如下图所示，接收方同时接收到0、1、2、3号分组，那么接收方可以就返回一个ACK 3表示接收方已经正确收到了3号分组以及之前的所有分组。
>
> 即接收方返回一个ACK n的分组，那么就表示接收方已经收到了n分组以及n分组之前的所有分组，即使接收方没有返回n之前的确认。



(3) **超时事件**：如果发生超时事件，发送方重传所有已发送但还未被确认过的分组。

后退N步协议名字就来源于出现丢失或时延过程分组发送方的行为。在GBN协议中，**发送方仅仅使用一个定时器**，这个定时器可以当作是最早的已发送但是没有被确认的分组所使用的定时器。如果收到了一个ACK，但仍有已发送但未被确认的分组，则定时器会被重新启动。当没有已发送但是没有被确认的分组时，定时器被终止。

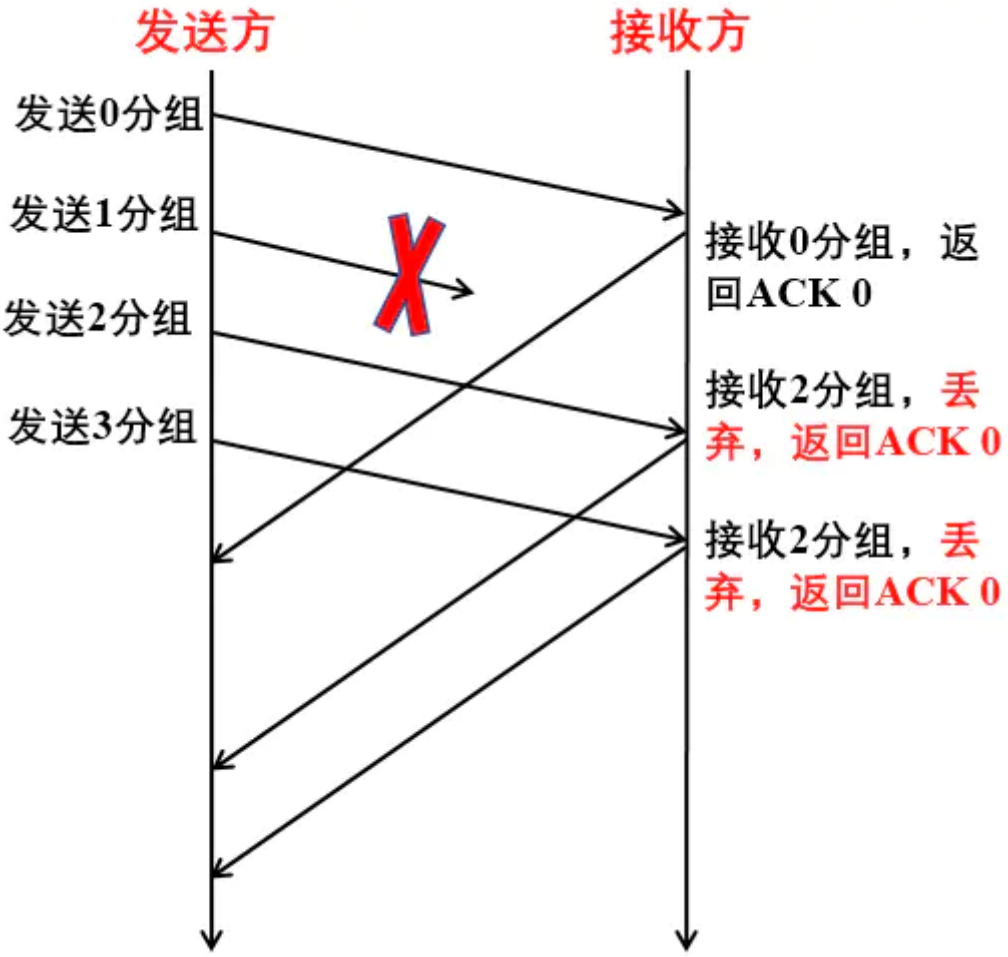
> 举个例子说明一下，假设发送方发送了0号、1号、2号、3号分组，此时0号分组是最早发送但是未被确认的分组，计时器在0号分组发送后启动，之后发送方接收到了0号、1号分组的确认。此时仍有已发送但是没有被确认的分组，即2号分组了，所以计时器重新启动。但2号分组由于网络原因而丢失，所以也就收不到2号分组的确认，由于GBN协议是累积确认机制，自然3号分组即使送达接收方也不会返回ACK 3，等到计时器超时，发送方会把2号分组和3号分组重新再发送一次。

3.2 发送方需要做的事

(1) 如果接收方正确收到n号分组，并且按序（即上次交付给上层的分组序号是n-1），则接收方位分组n发送一个ACK，并将该分组交付给上层。

(2) 其他情况，接收方丢弃分组，并为最近按序接收到的分组重新发送ACK。即GBN协议中，接收方丢失所有失序的分组，接收方不缓存任何的失序分组，只需要维护一个**下一个按序接收的分组序号：expectedseqnum**。

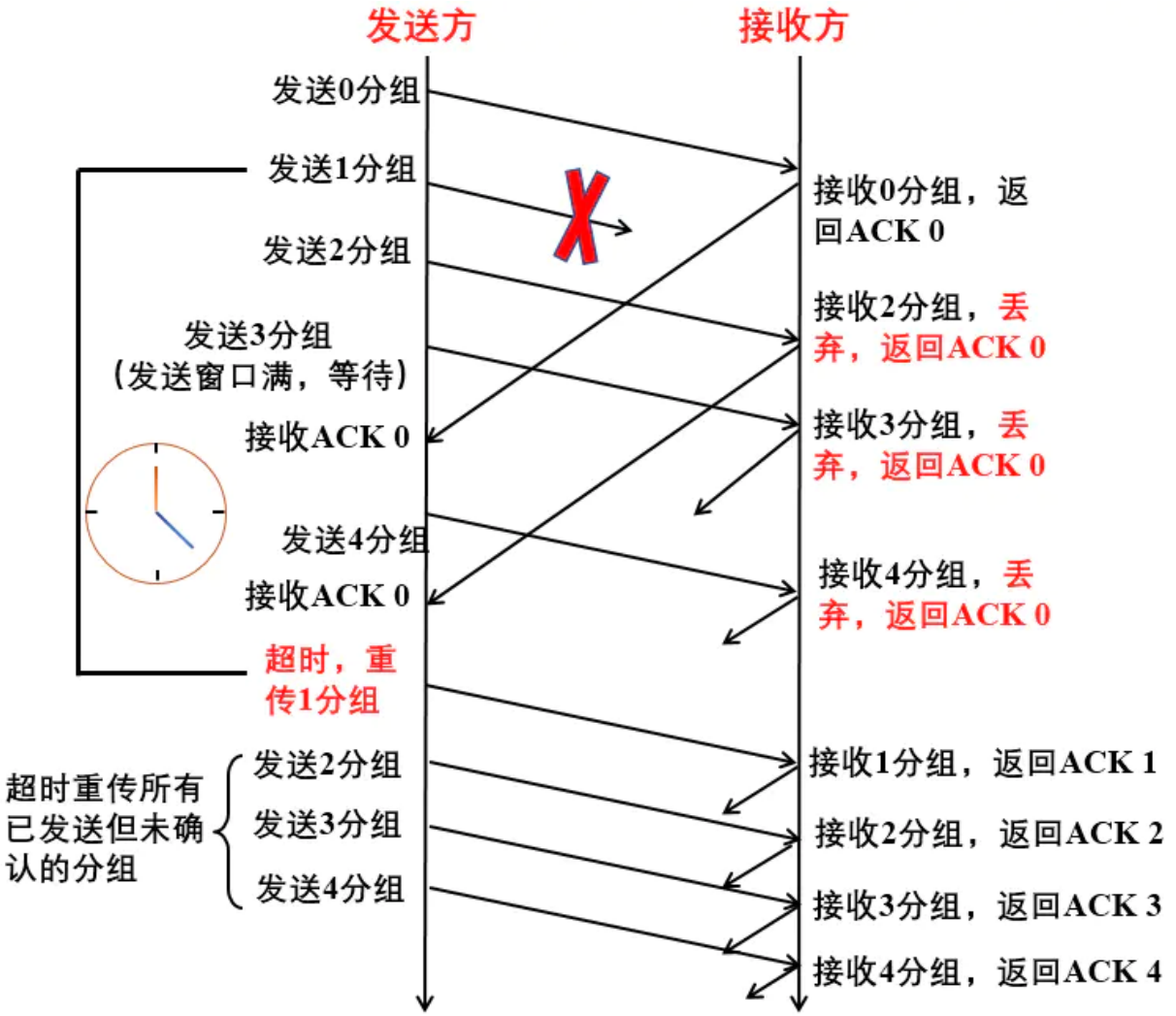
> 如下图所示，发送方发送4个分组，其中2号分组丢失，接收方返回0号分组的确认后，就知道下一个需要接收的分组序号是1，但是没有接收到1号分组，而是后面的3号分组和4号分组，所以接收方发现不是1号分组则简单粗暴的丢失，并且为最近按序收到的分组重新发送ACK，本例中即接收方返回的是ACK 0，它给发送端返回的意思是，我已经正确收到0号分组了，请给我下一个分组即1号分组，其他分组不是我想要的。



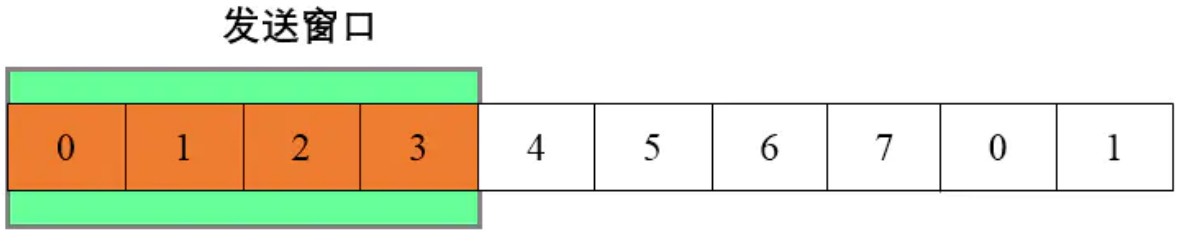
很显然，GBN协议对失序的分组选择直接丢失的行为优点是使得接收缓存简单，即接收方不缓存任何失序的分组，但是缺点也是非常明显的，由于一个分组出错可能需要重传原来已经正确传送的分组，这对资源无疑是一种浪费，其次这些重传的分组也许也会丢失或出错，从而导致更多的重传。

最后，用一个例子总结下一GBN协议的工作流程

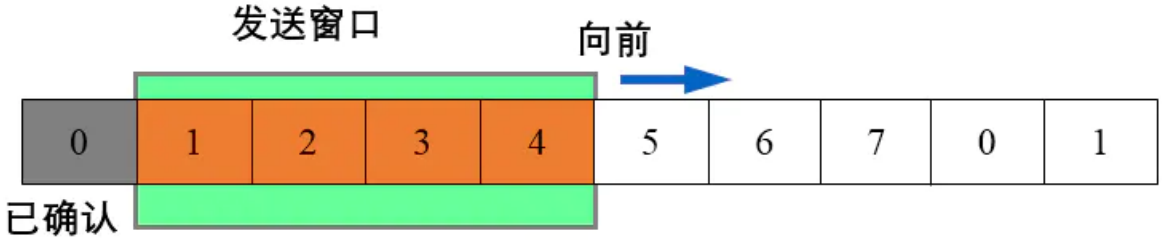
> 假设窗口长度为4，序号范围0~7。



(1) 发送端发送0、1、2、3号分组，此时发送窗口已满，不能在发送新的分组，在4个分组的传输的过程中，其中2号分组丢失。

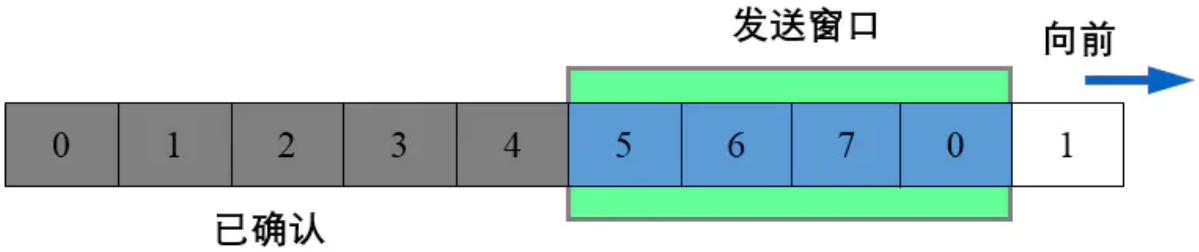


(2) 接收方返回对0号分组的确认，此时窗口向前滑动一个分组位置，并同时发送4号分组。此时，由于0号分组已被确认，计时器重新启动，开始为分组1开始计时....



(3) 接收方在接收到发送方的0号分组后期望接下来收到1号分组，但是没有收到1号分组，但是却收到2号分组和3号分组，所以直接丢失，并向发送方返回ACK 0，同样对于之后收到的4号分组也是直接丢失并返回ACK 0，此时窗口已经满了，也无法发送新的分组。

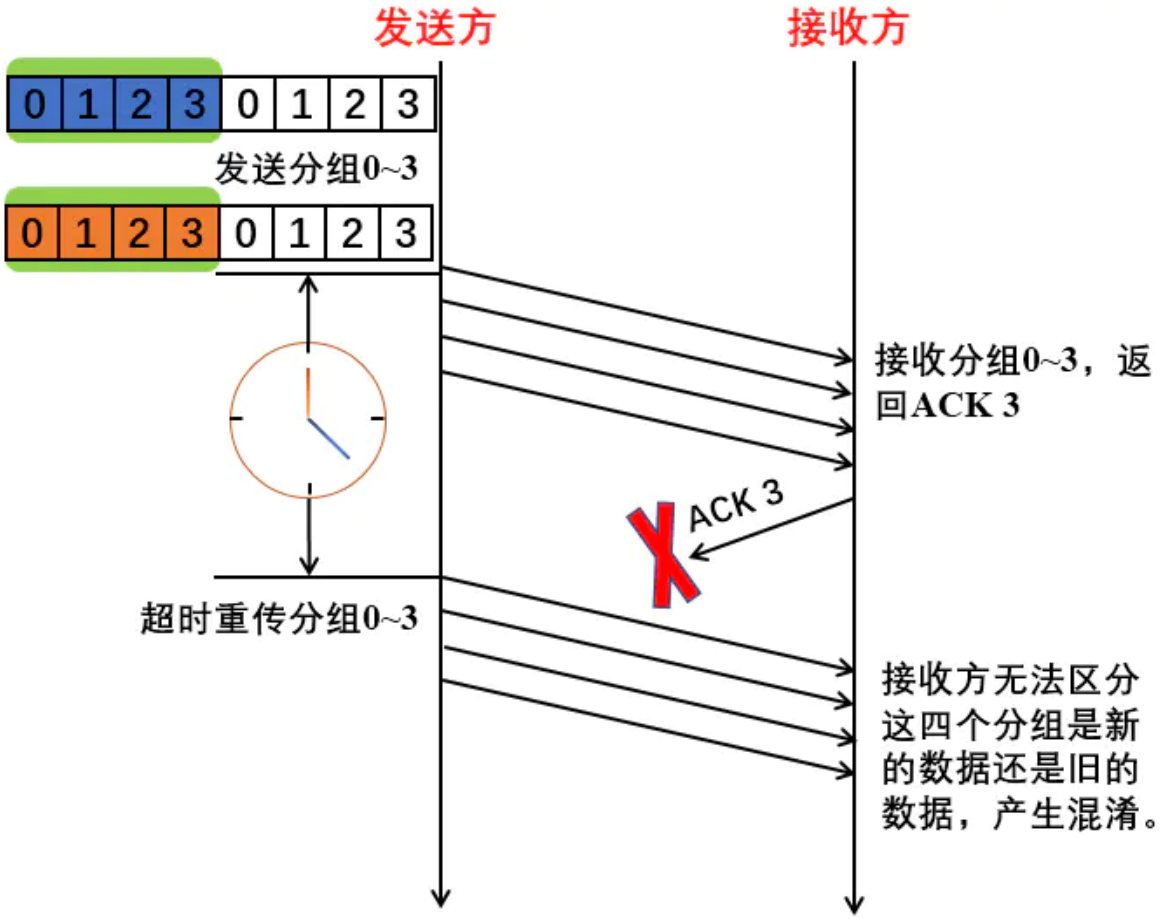
(4) 在计时器超时后，发送方重传所有已发送但是没有被确认的分组，即1号、2号、3号和4号分组，如果这次没有分组丢失，那么接收方会返回这4个分组的确认，发送方在收到确认后，将窗口向前移动4个分组位置。



3.3 GBN协议中滑动窗口长度

窗口的长度越长，发送方可以不需等待确认就发送的分组就越多，信道利用率也就高，但是窗口的长度不能是无限的。若**采用n位比特表示序号，那么发送窗口的尺寸大小W应满足：1 ≤ W ≤ 2^n - 1。因为发送窗口过大，就会使得接收方无法区分新的分组和旧的分组。**

> 假设采用2位来表示序号，则序号的范围就是[0 , 3]，根据上面的公式，发送窗口的最大尺寸是2^2 - 1 = 3。现在如果取发送窗口的大小为4，如下图所示。



> 发送方发送分组0~3，接收方在接收到4个分组后，返回ACK 3（表示收到这4个分组了），但是确认分组在传送的过程中丢失了，发送方在等待一段时间后，计时器超时，重新发送分组0~3，这时接收方就不清楚这四组数据是新的数据还是旧的数据。

3.4 GBN协议总结

(1) GBN协议使用累积确认机制。
(2) 接收方只按顺序接收分组，不按序的分组直接丢失。
(3) 重传时需要重传全部已发送但是没有被确认的分组。
(4) 确认序列号最大的、按序到达的分组。

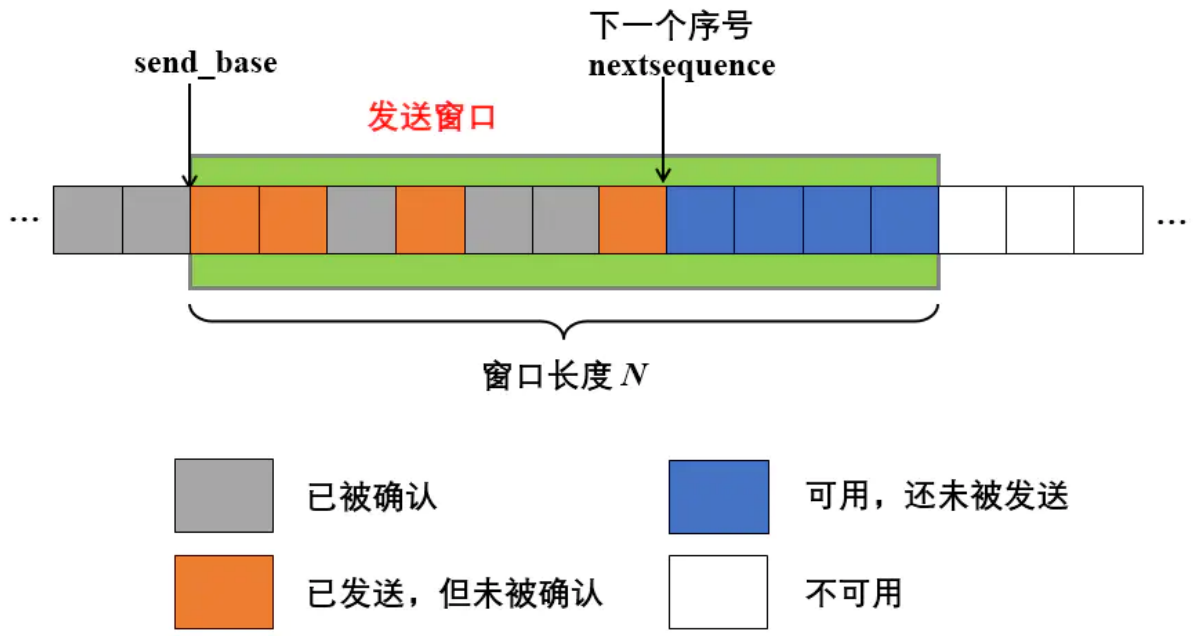
4.选择重传协议SR

选择重传协议是对GBN协议的缺点进行了改进，GNB超时重传会对那些已经正确传送的分组都重新传送，并且对失序到达的分组直接丢失。

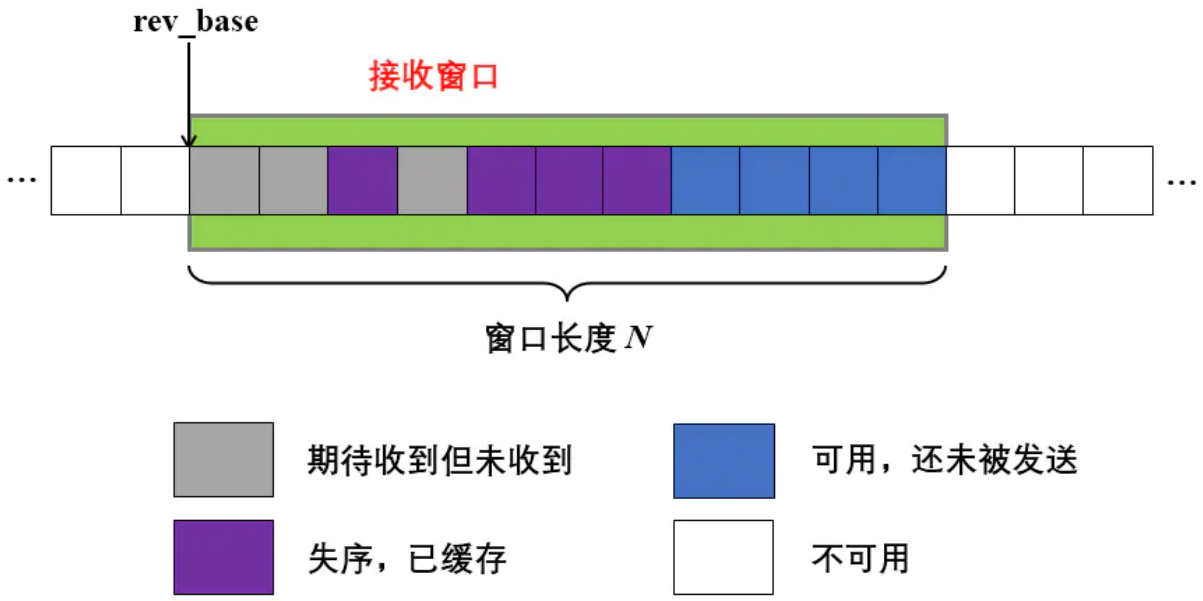
选择重选协议通过让发送方仅重传那些可能在接收方出错（即丢失或受损）的分组而避免了不必须要的重传。SR协议的改进措施：

> 1. 接收方对每个分组**单独进行确认**，并设置**缓存机制**，缓存乱序到达的分组
> 2. 发送方只重传那些没有收到的ACK分组，并为每个分组设置**定时器**。

下图表示选择重选协议的发送窗口和接收窗口。**发送窗口和同步窗口是不同步的。**



> 注：上下两个图并没有对应关系。



4.1 SR发送方必须响应的三件事

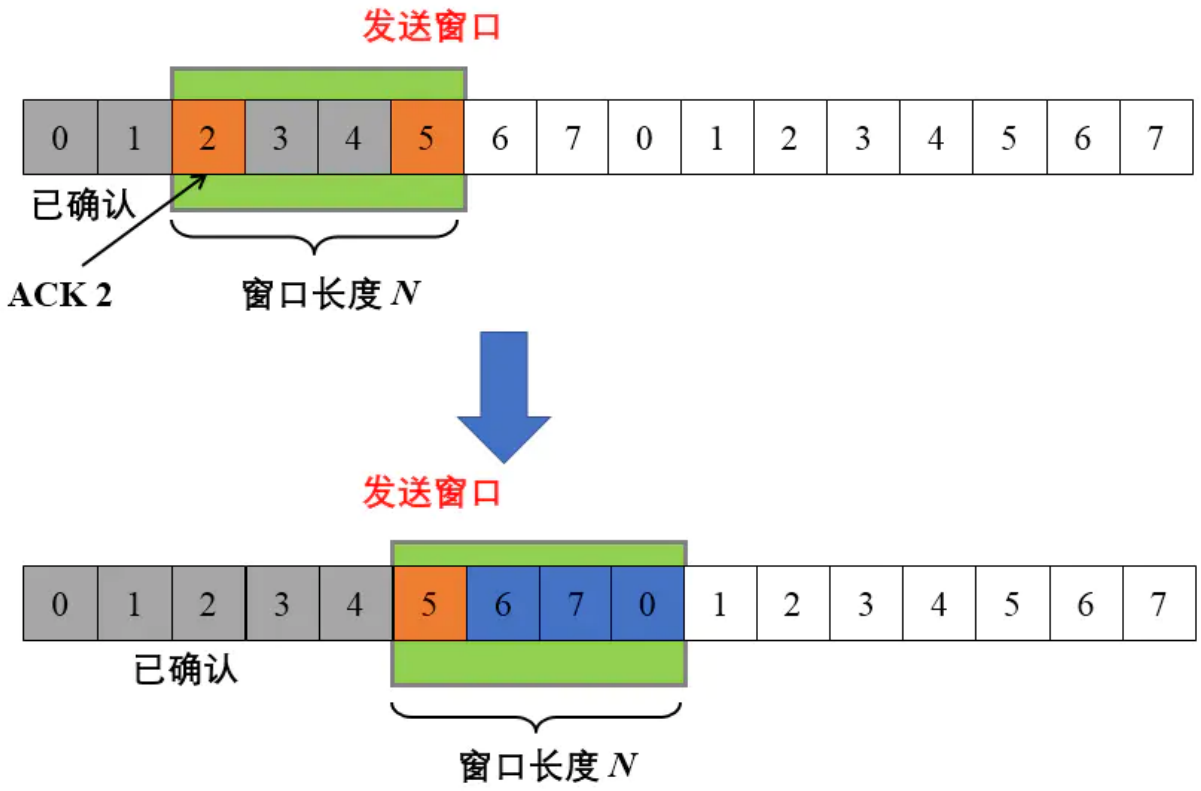
(1) 上层的调用。

> 从上层收到数据后，SR发送方检查下一个可用于该分组的序号，如果序号位于发送窗口之内，则发送分组；否则就像GBN一样，要么将数据缓存，要么返回给上层过一会再传输。

(2) 收到一个ACK。

> 如果收到ACK，假如该分组序号在窗口内，则SR发送方将那个被确认的分组标记为已接收。如果该分组序号是窗口的下界（最左边第一个窗口对应的序号），则窗口向前移动到具有最小序号的未确认分组处。

如下图所示，当收到2号分组的确认后，窗口向前滑动一直到最小序号的未确认的分组处，即5号分组处。

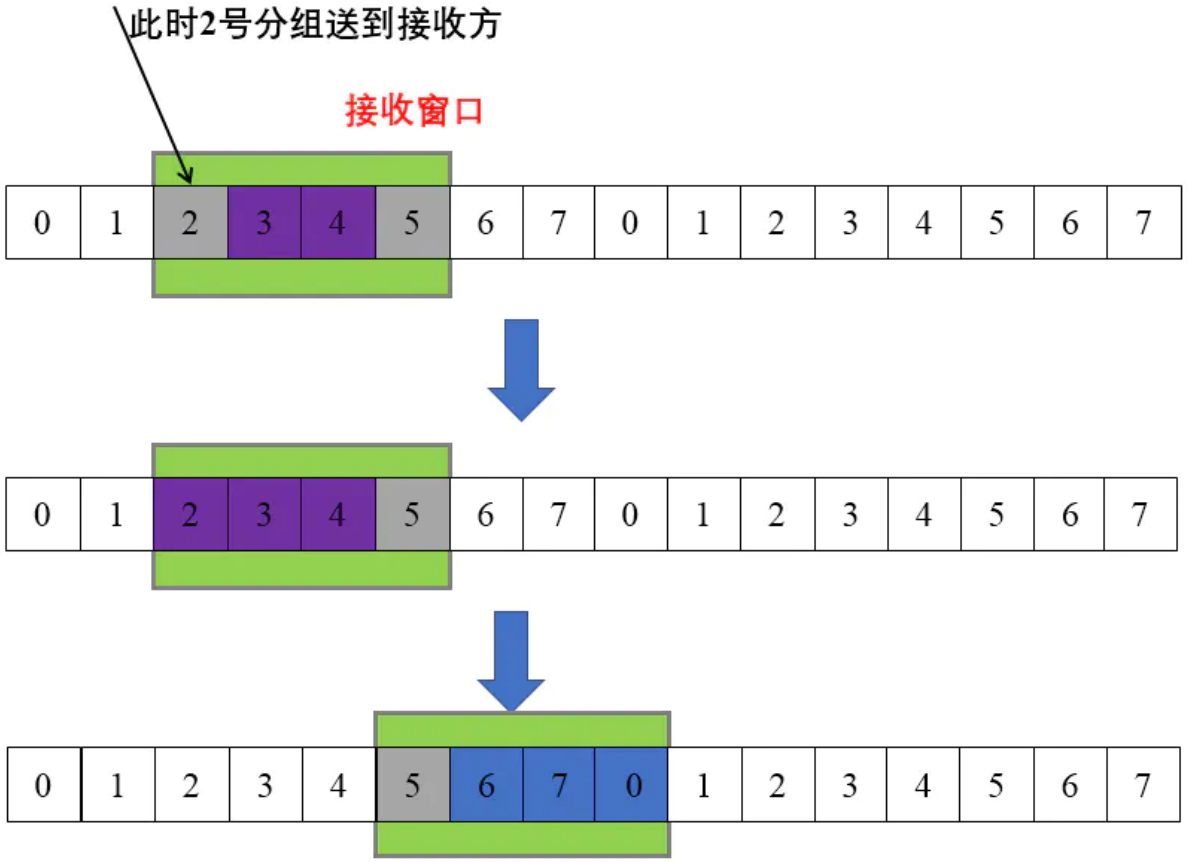


(3) **超时**：定时器再次被用来发送丢失的分组。每个分组都有自己的逻辑定时器，因为超时发生后只能发送一个分组。

4.2 SR接收方必须做的事

(1) SR接收方将确认一个正确接收的分组（在窗口内）而不管其是否按序。失序的分组将被缓存直到所有丢失的分组（即序号更小的分组）都被收到为止，这是才可将一批分组按序交付给上层，然后向前移动滑动窗口。

> 如下图所示，3号、4号以及6号分组失序达到并已经确认，如果此时2号分组到达，那么接收方就将2号、3号和4号分组交付给上层，并将窗口向前滑动。



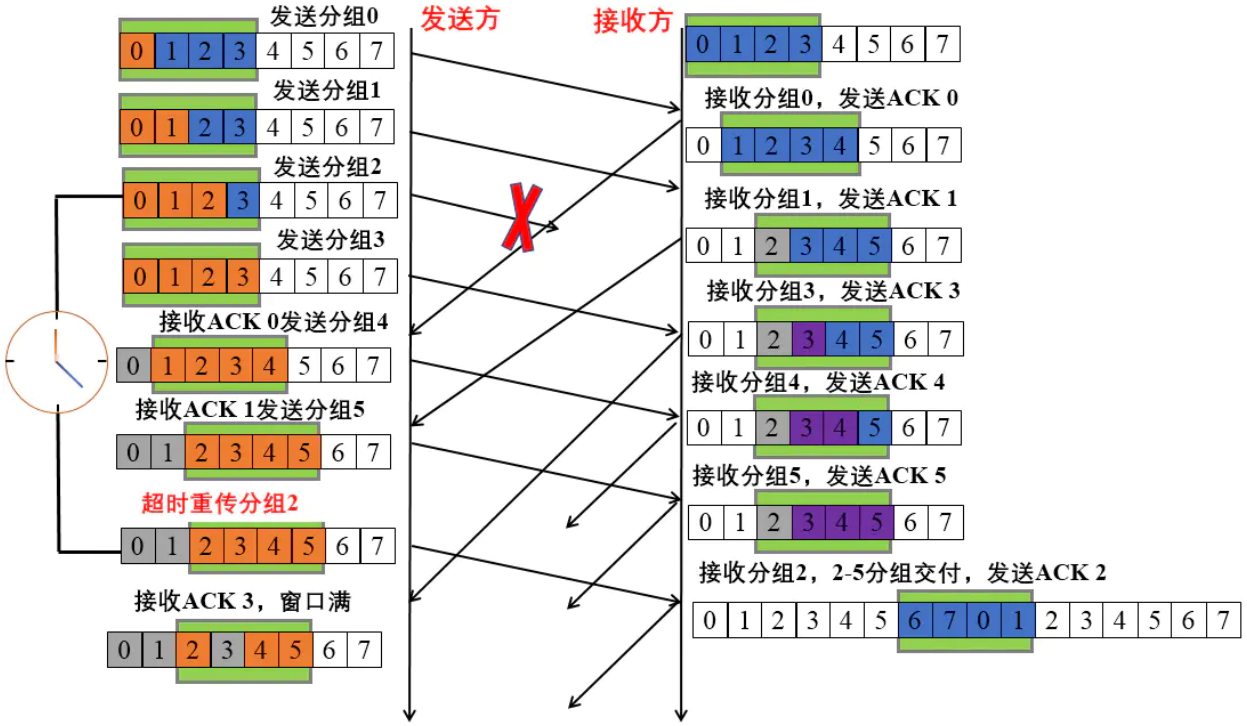
如果接收方收到一个在窗口左侧的分组（确切的来说，左边一个窗口大小范围的序号分组，即rev_base - N~rev_base - 1），在这种情况下，必须发送一个ACk，即使该分组是接收方以前已确认的分组。

> 还是上图，如果接收端过了一会又接收到了2号分组，这就表明接收端发送给发送方的确认分组丢失了，接收方以为接收方没有收到自己分送的分组，这时会超时重传，但是接收方确实已经接收到了2号分组，所以此时接收方会给发送方重新返回一个ACK。

对于其他情况，则忽略分组即可。

下面的图可以表示SR协议的工作过程，具体分析过程和GBN协议相似，这里就不细说了。

> 图中假设了发送窗口和接收窗口的大小都是4，序号范围0~7。



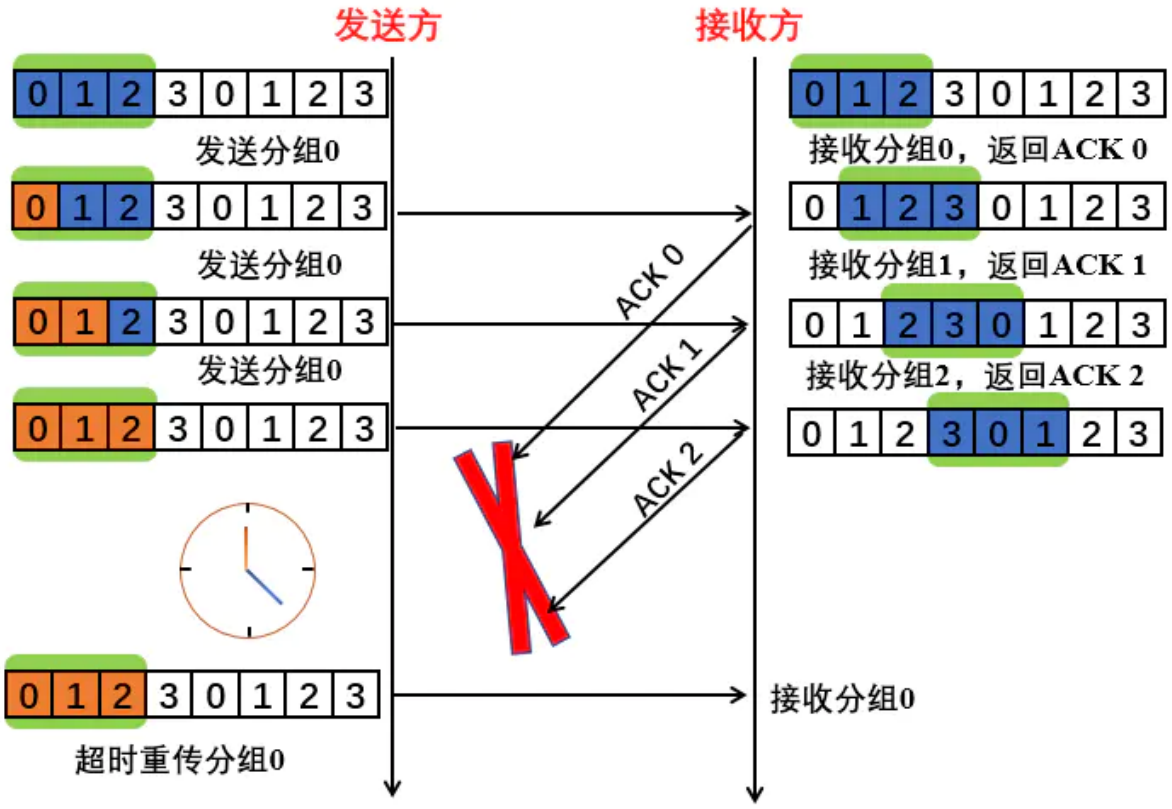
4.3 SR协议中发送窗口和接收窗口

在SR协议中，同样存在窗口大小的问题，在SR协议中发送窗口最好和接收窗口大小相等。如果发送窗口过大，会导致接收窗口溢出，过小同样也不好。

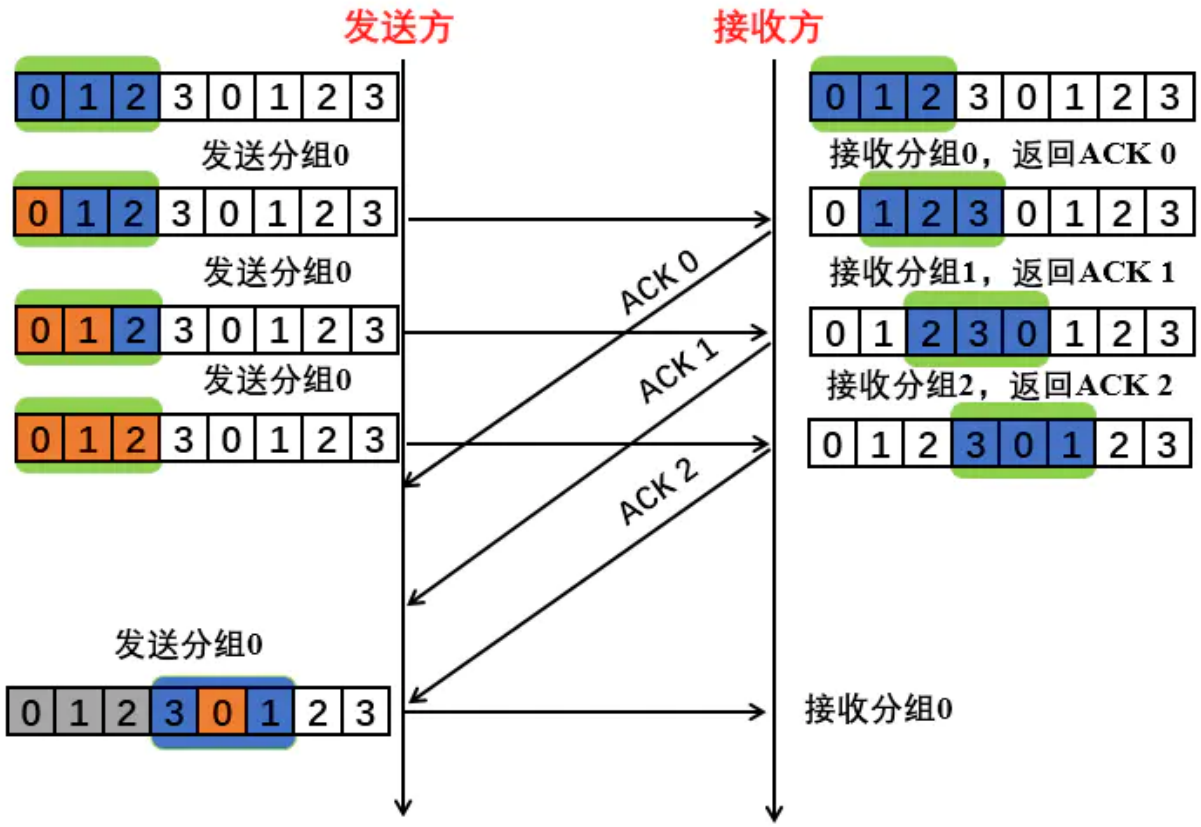
对于采用n位比特表示序号，那么发送窗口和接收窗口的尺寸大小WTmax、WRmax 应满足：WTmax = WRmax = 2^(n-1)。

> 假设采用2位来表示序号，则序号的范围就是[0，3]，根据上面的公式，发送窗口和接收窗口的最大尺寸是2^2-1 = 2。现在如果取发送窗口的大小为3。

如下图所示，发送方发送分组0~3，接收方接收了3个分组并返回了分组的确认，但是三个确认分组在传送过程中都是丢失了，接收方在计时器超时后会重发三个分组，现在仅考虑分组0。



如下图所示，发送方发送分组0~3，接收方接收了3个分组并返回了分组的确认，发送方接收到了确认后将窗口向前滑动，此时发送方乱序发送分组0。



从上帝视角来看，虽然接收方接收的都是分组0，但是第一种情况分组0是旧的数据，而第二种情况是新的数据。即如果窗口的大小设置为3时，如果接收某个时刻接收到分组0，接收方并不能分清接收的

是超时重传的分组还是一个新的分组，从而产生歧义。**所以窗口大小并不是无限制的。**

4.4 SR协议总结

(1) 对窗口内的分组逐一确认，收一个确认一个。
(2) 只传出错的分组。
(3) 接收方有缓存机制，存放失序的分组。

5.总结