重传超时（RTO）：

TCP在发送数据时会设置一个计时器，若超时仍未收到ACK，则会引发对应的重传操作。

快速重传：

通常发生在没有延时的情况下，若TCP累计确认无法返回新的ACK，或SACK表明出现失序报文段时，则会认为丢包，导致重传。

TCP每次重传间隔时间都会x2，称为二进制指数退避。

TCP拥有两个值来决定如何重传同一个报文段：1.尝试重传的次数 2.等待时间

在Linux中

对于一般数据段来说，可以通过net.ipv4.tcp\_retries1（重传次数，默认值3）和net.ipv4.tcp\_retries2（等待时间，默认值15，约为13~30分钟）来设置TCP重传参数；对于SYN报文段来说，通过设置net.ipv4.tcp\_syn\_retries和net.ipv4.tcp\_synack\_retries来设置SYN报文段和SYNACK报文段的重传次数。

### 设置重传超时：

TCP时常工作在动态的网络环境中，其RTT也会动态变化，因此需要动态设置RTO（重传超时）。

若RTO过小则会在网络中引入不必要的重复数据，若RTO过大会降低网络的利用率。

每个TCP连接的RTT均独立估算，并且重传计时器会对任何占用序列号的在传数据计时，包括SYN和FIN报文段。

#### 设置RTO的方法：

1. 经典方法：

平滑的RTT估计值：**SRTT = α \* SRTT+（1-α）\* RTT**

α为平滑因子，推荐值为0.8~0.9，每次得到新的RTT就会更新SRTT，这样新估计值10~20%来自于新测量的RTT值。

RTO的设置：**RTO = min(ubound, max(lbound, (SRTT \* β ))**

β为时延离散因子，ubound为上边界，lbound为下边界。

简单来说RTO为SRTT \* β，β推荐值为1.3~2.0。

这种算法对于比较稳定的网络来说能够取得不错的性能，但是对于RTT变化较大的网络中则无法取得预期的效果。

1. 标准方法：

改善了经典方法，通过记录RTT的变化情况以及均值来进行估值。

因为计算标准差需要对方差及逆行平方根运算，对于TCP来说代价较大，所以选择使用平均偏差来计算。

**srtt = (1-g)(srtt) + (g)rtt**

**rttvar = (1-h)(rttvar) + (h)(|rtt - srtt|)**

**RTO = srtt + 4(rttvar)**

在经典方法的基础上加上了rttvar，RTT变化时，偏差量越大，RTO增长越快。

初始值：**srtt = rtt，rttvar = rtt/2。**

1. 因为重传报文的ACK存在二义性，无法区分是原始报文还是重传报文的ACK，所以出现超时重传时的ACK不进行RTT的计算。

#### 使用时间戳选项测量RTT：

由于可能出现多个报文段使用一个ACK、出现数据丢失失序等情况，TCP报文段与ACK并不是严格的一对一关系。

所以接收端每次需要记录上一个发送的ACK号(LAST ACK)，新的报文段到达时，比较last ack与序列号的值，若相等则证明是“相邻”的数据，此时将时间戳回传。发送端比较时间戳与当前时间，得到RTT。

分组失序或丢失时，发送端会接收到重复ACK，RTT会被过高的估算，导致RTO更大，此时在一定程度上可以避免不必要的重传（后续报文段未接收到ack，避免后续报文的的重传）。

### 基于计时器的重传：

每个TCP连接的发送端会随着报文段的发送，不断的设定和取消一个重传计时器。

若在RTO时间内未收到计时报文段的ACK，则会触发超时重传，此时会降低发送速率来对此进行快速响应。

一种方法是基于拥塞控制机制减少发送窗口的大小。

另一种是暂时增大RTO的退避因子γ，RTO=γRTO。γ通常为1，随着多次重传，γ=2，4，8加倍增长。一旦接收到对应的ACK，γ会重置为1。

### 快速重传：

TCP发送端在接收到至少dupthresh（重复ACK阈值）个重复ACK后，立即重传可能丢失的数据分组，而不必等到重传计时器超时。

TCP接收端在接收到失序报文段时，需要立即生成重复ACK并返回。当采用SACK时，重复ACK通常也包含SACK信息，可以通过SACK获知多个空缺。

### 带选择确认（SACK）的重传：

使用SACK选项可以在一个RTT内获知多个空缺，减少不必要的重传。

1. SACK的接收端行为：
2. SACK的发送端行为：