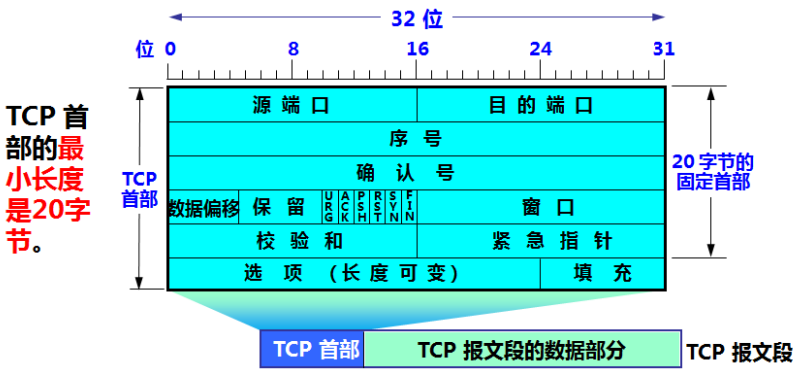
课程回顾

- TCP报文段格式



- TCP连接管理
  - TCP连接的建立：三报文握手
  - TCP连接的释放：四报文握手

- TCP流量控制

  采用滑动窗口机制

- TCP可靠传输 —— ARQ协议
  - 停止等待ARQ协议
  - 连续ARQ协议

    以字节为单位的窗口滑动
    - 回退N（Go-Back-*N*）

      累积确认
    - 选择性重传

      SACK选项

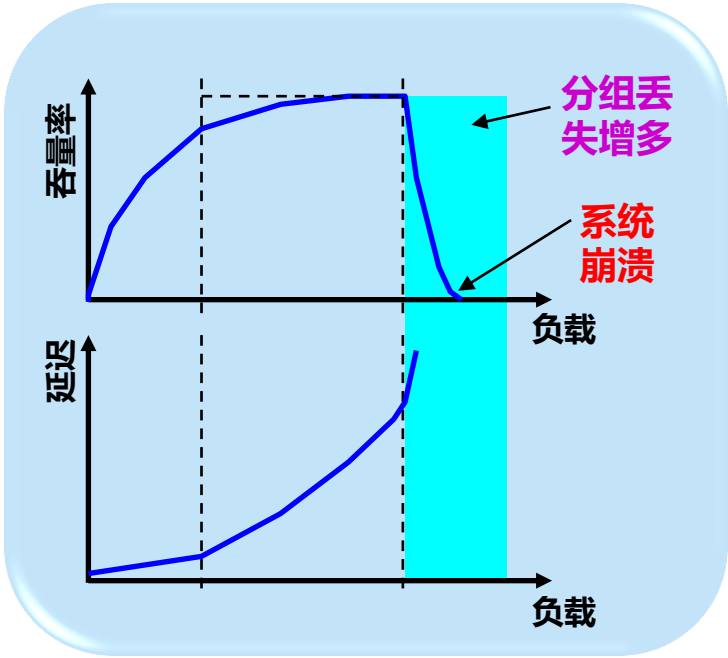
	连续ARQ协议	停止等待协议
发送的分组数量	一次发送多个分组	一次发送一个分组
传输控制	滑动窗口协议	停止、等待
确认	单独确认，累积确认，累积确认+选择性确认	单独确认
超时计时器	每个发送的分组	每个发送的分组
序号	每个发送的分组	每个发送的分组
重传	回退N个分组，多个分组	一个分组

- 超时重传

  根据 RTT 设置 RTO

TCP 拥塞控制

- 在某段时间，若对网络中某资源（如带宽、缓存、处理机等）的需求超过了该资源所能提供的可用部分，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降，这种现象称为拥塞。
- 最坏结果：系统崩溃。
- 网络拥塞产生的原因
  - 链路的容量不足，资源分配不均衡
  - 路由器缓存空间小，流量分布不均衡
  - 处理机速率太慢
  - 拥塞本身会进一步加剧拥塞



∑ 对资源需求 > 可用资源

TCP 拥塞控制

增加资源能解决拥塞吗？

- **不能**，还可能使网络的性能**更坏**。
- 网络拥塞往往是由许多因素引起的。例如：
  - **增大缓存**，**但未提高输出链路的容量和处理机的速度**，排队等待时间将会大大增加，引起大量超时重传，**解决不了网络拥塞**；
  - **提高处理机处理的速率**会将**瓶颈转移**到其他地方。
  - 拥塞引起的**重传**并不会缓解网络的拥塞，反而会加剧网络的拥塞。

拥塞控制与流量控制的区别

拥塞控制

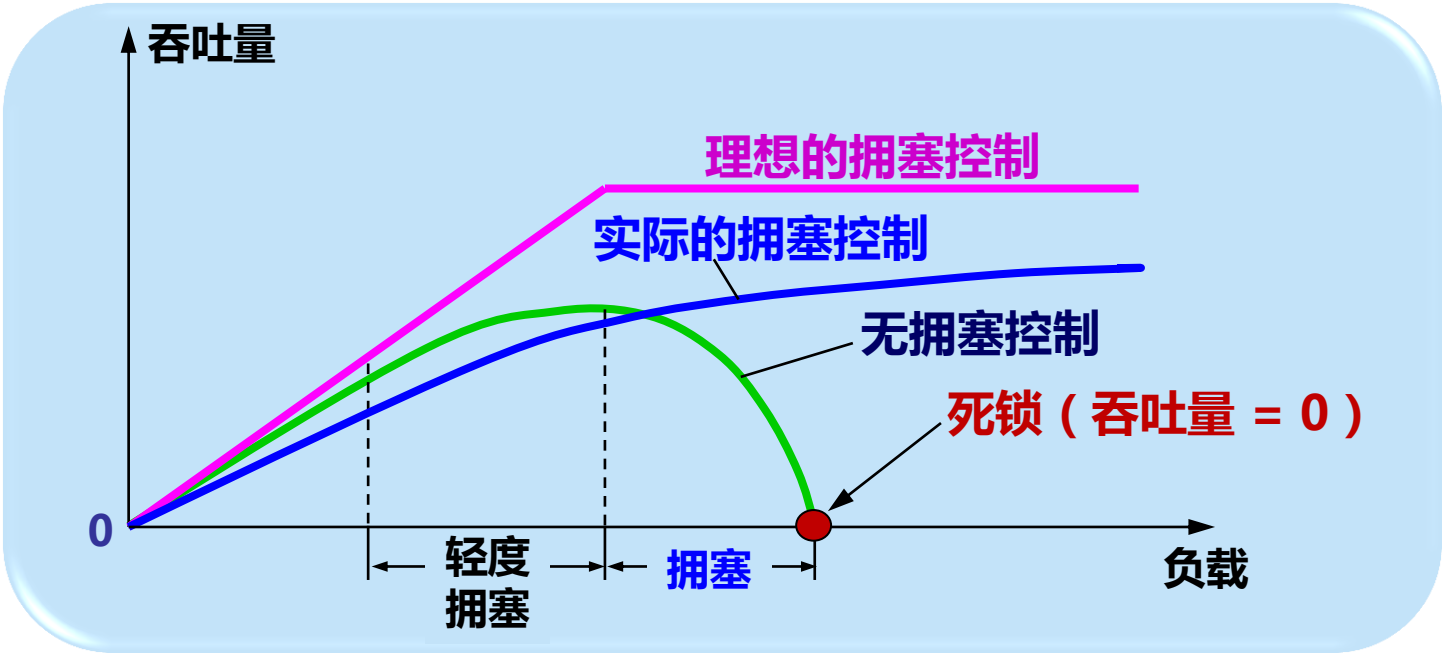
- 防止过多的数据注入到网络中，避免网络中的路由器或链路过载。
- 是一个全局性的过程，涉及到所有的主机、路由器以及与降低网络传输性能有关的所有因素。

流量控制

- 抑制发送端发送数据的速率，以使接收端来得及接收。
- 点对点通信量的控制，是个端到端的问题。

TCP 拥塞控制

- 拥塞控制的前提：网络能够承受现有的网络负荷。
- 实践证明，拥塞控制是很难设计的，因为它是一个动态问题。
- 分组丢失是网络发生拥塞的征兆而不是原因。
- 在许多情况下，甚至正是拥塞控制本身成为引起网络性能恶化、甚至发生死锁的原因。



TCP 拥塞控制

开环控制

- 在设计网络时，事先考虑周全，力求工作时不发生拥塞。
- 一旦整个系统运行起来，就不再中途进行改正。
- 思路：力争避免发生拥塞。

闭环控制

- 基于反馈环路的概念。
- 根据网络当前运行状态采取相应控制措施。
- 思路：在发生拥塞后，采取措施进行控制，消除拥塞。

TCP 拥塞控制

闭环控制措施

1. 监测 — 监测网络，检测拥塞在何时、何处发生。
2. 传送 — 将拥塞发生的信息传送到可采取行动的地方。
3. 调整 — 调整网络的运行以解决出现的问题。

- **监测网络拥塞的主要指标**
  - 由于缺少缓存空间而被丢弃的分组的百分数；
  - 平均队列长度；
  - 超时重传的分组数；
  - 平均分组时延；
  - 分组时延的标准差，等等。

- **拥塞通知的传递与时机**
  - 过于频繁，会使系统产生不稳定的振荡。
  - 过于迟缓，不具有任何实用价值。
  - 采取的策略
    - ✓ 发送“通知拥塞发生”的分组。
    - ✓ 在分组中保留表示拥塞状态的字段。
    - ✓ 周期性地发出探测分组等。

TCP 拥塞控制

- 既成事实标准的TCP/IP软件实现来自于伯克利的加利福尼亚大学的计算机系统研究小组，软件是随同4.x BSD （Berkeley Software Distribution）系统的网络版一起发布的。
- 几个TCP/IP版本
  - 4.2 BSD：第一个广泛可用的TCP/IP版本
  - 4.3BSD：性能得到改善
  - TCP Tahoe：慢启动，拥塞避免，快速重传
  - TCP Reno：慢启动，拥塞避免，快速重传，快速恢复
  - TCP NewReno
  - TCP SACK
  - TCP Vegas
  - TCP Westwood

TCP 拥塞控制

几个重要的参数：

- **拥塞窗口**（cwnd）：发送端在拥塞控制情况下一次最多能发送的报文段数量。
- **通告窗口**（rwnd）：接收端给发送端预设的发送窗口大小。
- **慢启动阈值**（ssthresh）：拥塞控制中慢启动阶段和拥塞避免阶段的分界点，**防止拥塞窗口增长过大**引起网络拥塞。
- **往返时间**（RTT）：一个报文段从发送端发出直到收到接收端返回的确认的时间间隔。
- **重传超时**（RTO）：一个报文段从发送到失效的时间间隔，是判断报文段丢失与否、网络是否发生拥塞的重要参数。该值通常设为2RTT或5RTT。
- **快速重传阈值**（tcprexmtthresh）：能触发发送端进入快速重传的同一报文段重复确认的数目。当此数目超过快速重传阈值时，就进入快速重传阶段。快速重传阈值的缺省值为3。

TCP 拥塞控制

- TCP 采用基于窗口的方法进行拥塞控制，该方法属于闭环控制方法。
- TCP 发送方维持一个拥塞窗口（cwnd）
  - cwnd 的大小取决于网络的拥塞程度，并且是动态变化的。
  - 只要网络没有出现拥塞， cwnd 就可以再增大一些，以便把更多的分组发送出去，提高网络的利用率。
  - 只要网络出现拥塞或有可能出现拥塞，就必须把 cwnd 减小一些，以减少注入到网络中的分组数，缓解网络出现的拥塞。
- 发送方根据网络的拥塞情况，利用 cwnd 调整发送的数据量。
- 发送窗口大小不仅取决于接收方窗口，还取决于网络的拥塞状况。
- 真正的发送窗口值：

真正的发送窗口值 = min（接收方通知的窗口值，拥塞窗口值）

TCP 拥塞控制

发送方是如何判断网络出现拥塞的呢？

超时重传计时器超时	• 网络已经出现了拥塞。
收到 3 个重复的确认	• 预示网络可能会出现拥塞。

现在通信线路的传输质量一般都很好，因传输出差错而丢弃分组的概率很小（远小于1 %），因此只要出现了超时，就判断网络出现了拥塞。

TCP 拥塞控制

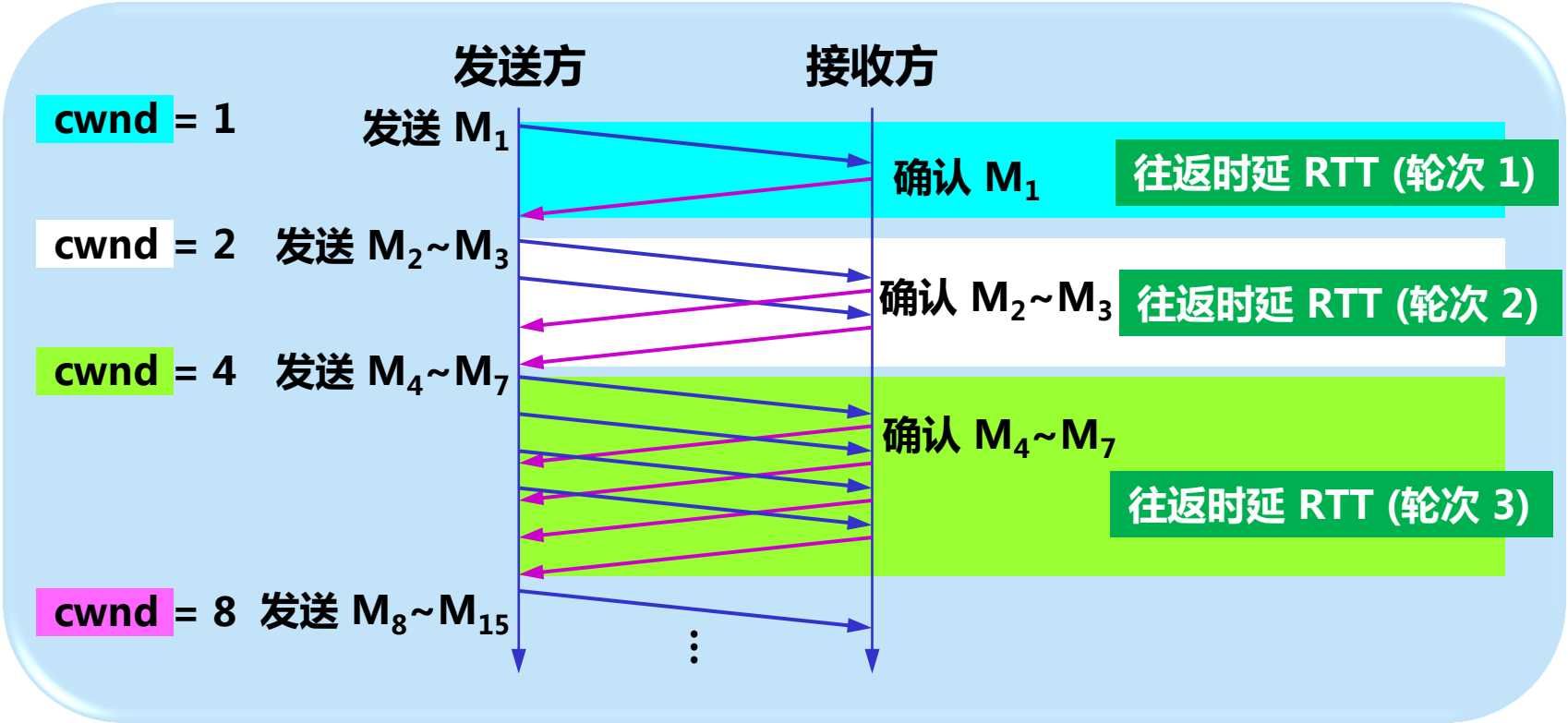
- TCP 拥塞控制算法
  - 慢启动
  - 拥塞避免
  - 快速重传
  - 快速恢复
- 讨论的前提条件
  - 单向数据传送，另一方只发送确认
  - 接收方接收缓存无限大， cwnd 的大小由网络拥塞程度来决定。

TCP 拥塞控制 -- 慢启动

- **目的**：探测网络的负载能力或拥塞程度。
- **算法思路**：由小到大逐渐增大注入到网络中的数据字节，即由小到大逐渐增大拥塞窗口（cwnd）数值。
- **两个控制变量**

拥塞窗口（cwnd）	慢启动阈值（ssthresh）
初始值两种设置方法： • 1 至 2 个最大报文段 MSS（旧标准） • 2 至 4 个最大报文段 MSS（RFC 5681）	防止cwnd增长过大引起网络拥塞。 • cwnd<ssthresh，使用慢启动算法； • cwnd>ssthresh，停用慢启动算法，使用拥塞避免算法； • cwnd=ssthresh，既可使用慢开始算法，也可使用拥塞避免算法。

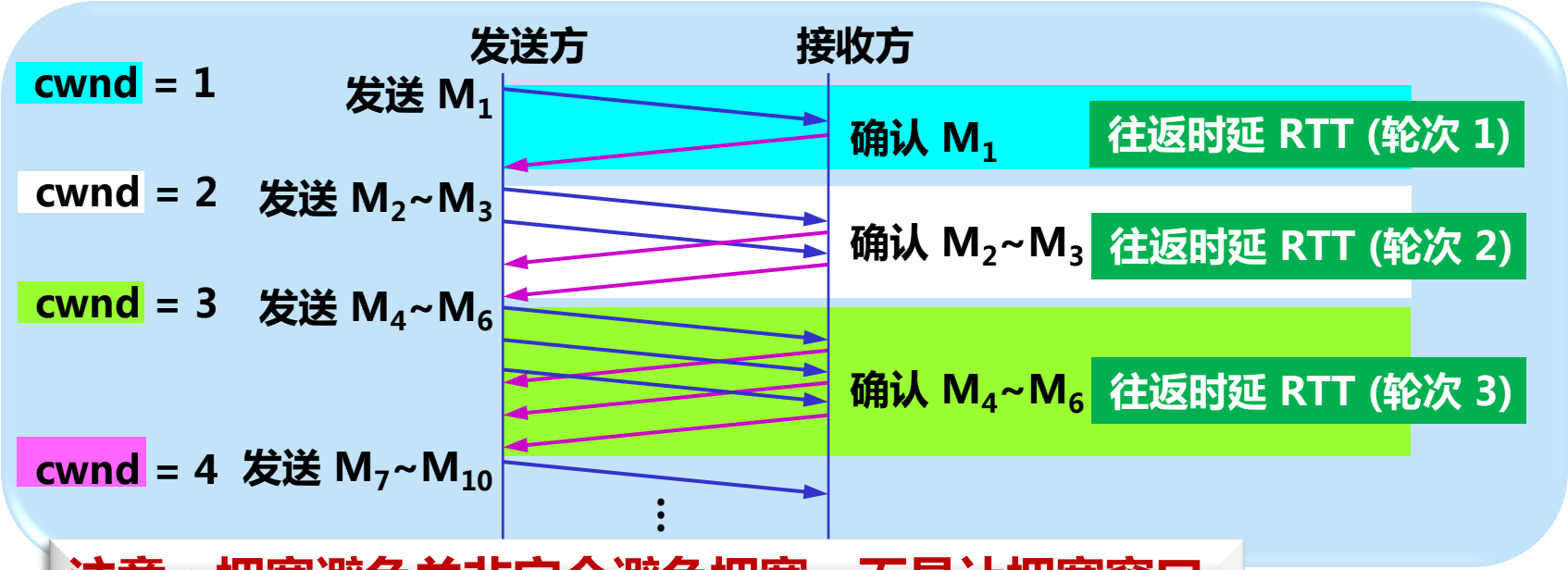
TCP 拥塞控制 -- 慢启动



- 发送方每收到一个对新报文段的确认（重传的不算在内）就使 cwnd 加 1。
- 每经过一个传输轮次， cwnd 就加倍。
- 窗口大小按指数增加，不慢！

TCP 拥塞控制 -- 拥塞避免

- **目的**：让拥塞窗口（cwnd）**缓慢地增大**，**避免**出现拥塞。
- **算法思路**：使 cwnd 按**线性规律缓慢增长**。**每经过一个往返时间 RTT**（不管在此期间收到了多少确认），发送方的 **cwnd = cwnd + 1（一个 报文段）**。
- 在拥塞避免阶段，具有**加法增大**（**AI**，Additive Increase）的特点。



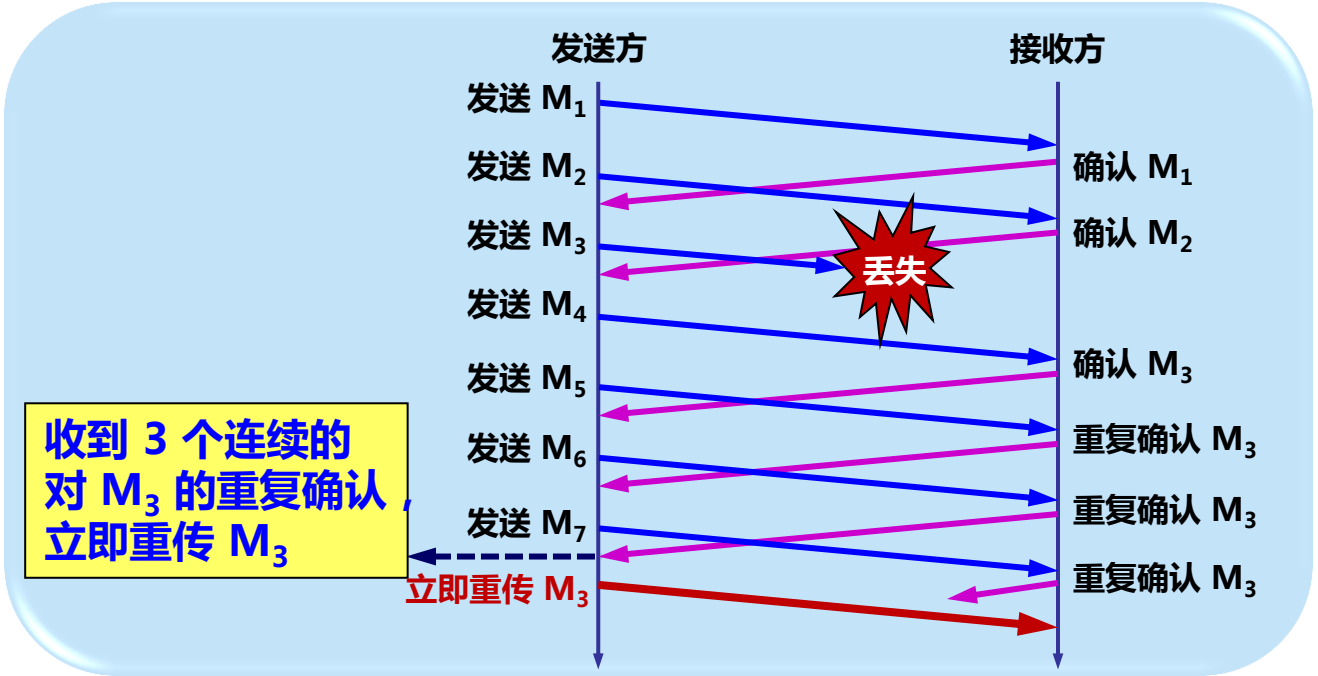
注意：拥塞避免并非完全避免拥塞，而是让拥塞窗口增长得缓慢些，使网络不容易出现拥塞。

TCP 拥塞控制 -- 慢启动和拥塞避免

- 无论在慢启动阶段还是在拥塞避免阶段，只要**发送方判断网络出现拥塞**（**重传定时器超时**）：
  - ssthresh = max (cwnd/2，2)
  - cwnd = 1
  - 执行慢启动算法
- **目的：**迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

TCP 拥塞控制 -- 快速重传

- **目的**：让发送方**尽早**知道发生了个别报文段的丢失。
- **算法思路**：快速重传算法要求接收方**立即发送确认**，即使收到了失序的报文段，也要立即发出对已收到的报文段的重复确认。发送方只要连续收到 **3 个重复确认**，就**立即进行重传（即“快速重传”）**，这样就不会出现超时，发送方也不会误认为网络出现了拥塞。
- 使用快速重传可以使整个网络的吞吐量提高约 20%。



注意：快速重传并非取消重传计时器，而是在某些情况下可以更早地（更快地）重传丢失的报文段。

TCP 拥塞控制 -- 快速恢复

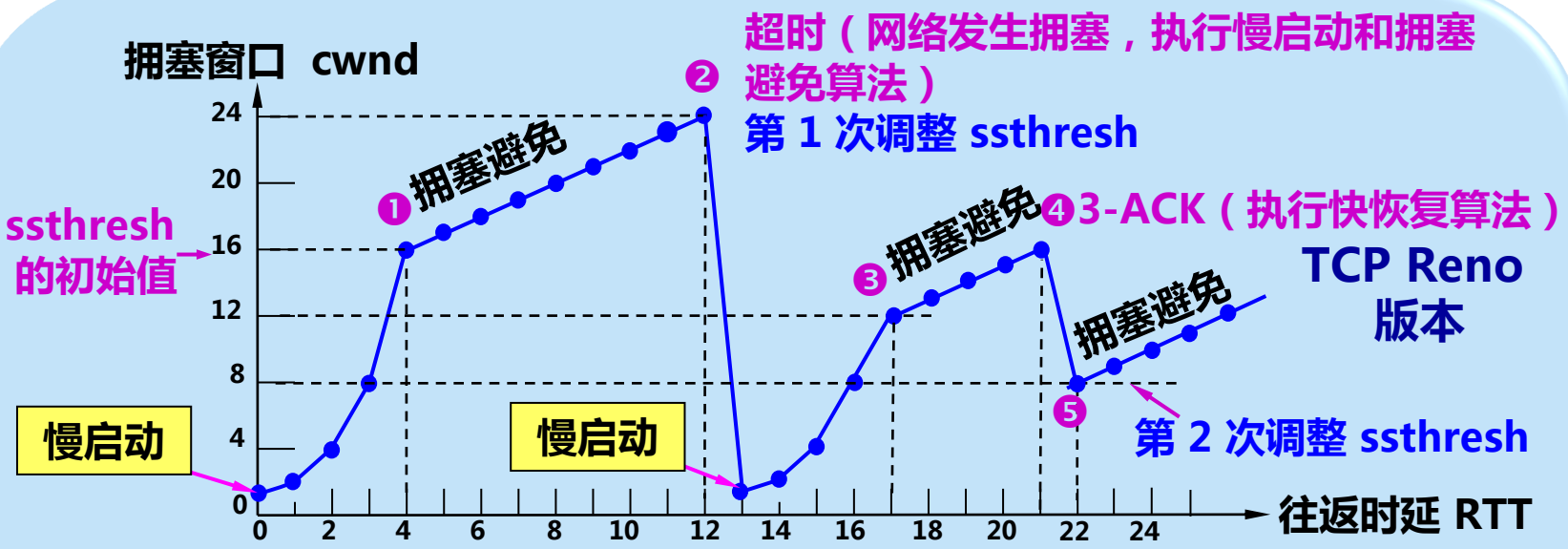
- 当发送端收到3个连续的重复确认时，由于发送方现在认为网络**很可能没有发生拥塞**，因此现在**不执行慢启动算法**，而是**执行快速恢复**算法。
  - 慢启动阈值 ssthresh = 当前拥塞窗口 cwnd / 2
  - 乘法减小拥塞窗口，新的拥塞窗口 cwnd = ssthresh 或 ssthresh+3（3个报文段）（RFC 5681）
  - 开始执行拥塞避免算法，使拥塞窗口缓慢地**线性增大**。

二者合在一起就是**加法增大、乘法减小（AIMD）**算法。

TCP 拥塞控制算法小结

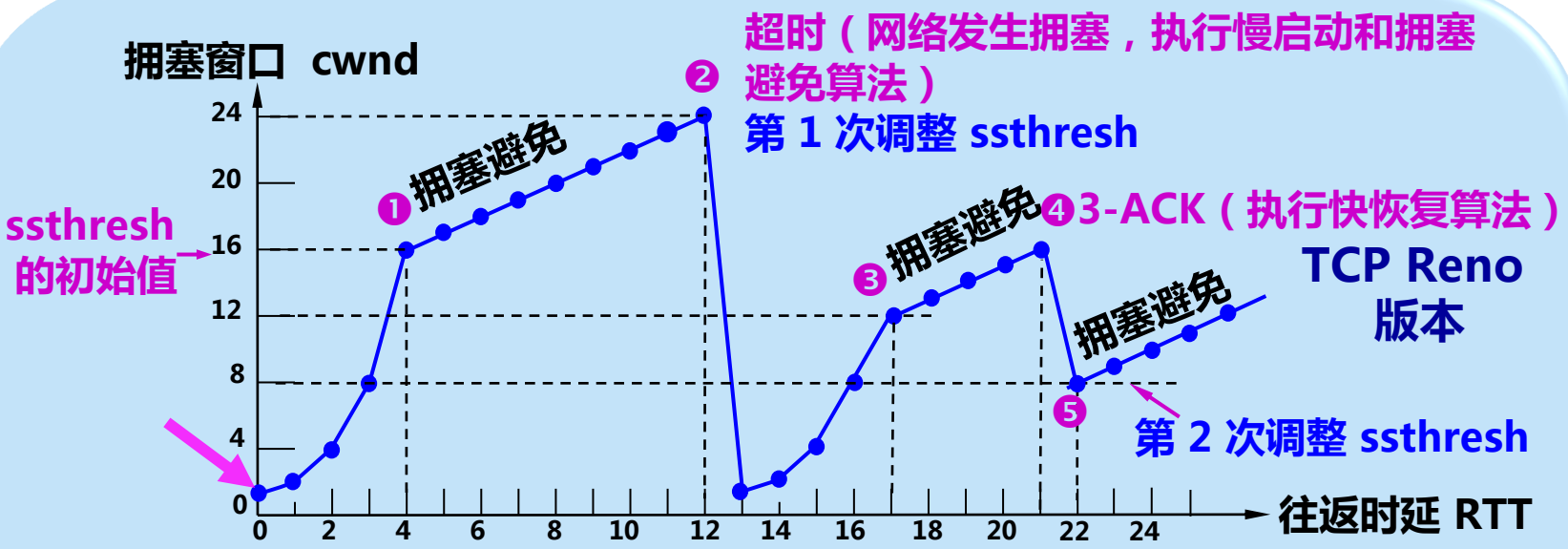
- 慢启动，每个 RTT 内，每收到一个确认，cwnd + 1；
- 拥塞避免，每个 RTT 内，无论收到多少确认，cwnd +1 (加法增大)；
- 乘法减小，超时重传或收到3个连续重复确认，慢启动阈值=cwnd/2；
- 快速重传，收到 3 个连续重复确认，立即重传丢失的报文段；
- 快速恢复，收到 3 个连续重复确认，慢启动阈值 = cwnd/2 ，执行拥塞避免算法。

TCP 拥塞控制算法举例



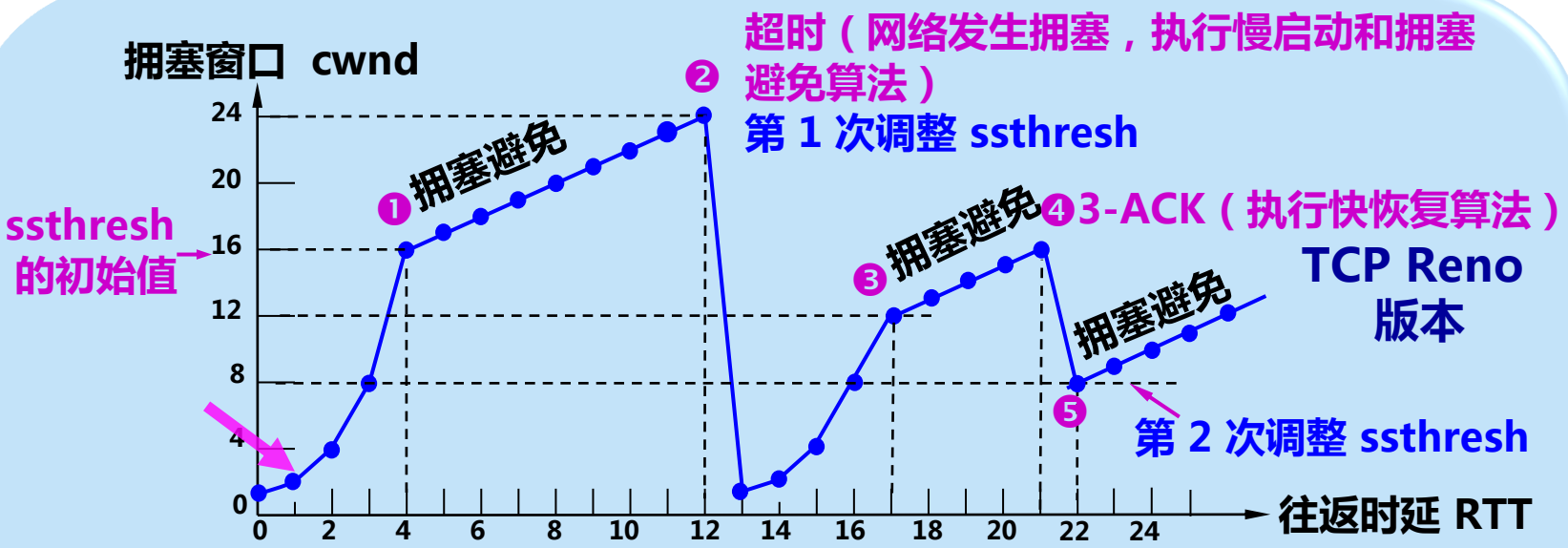
- 当 TCP 连接进行初始化时，将拥塞窗口置为 1（窗口单位不使用字节而使用报文段）。
- 将慢启动阈值的初始值设置为 16 个报文段，即 ssthresh = 16。

TCP 拥塞控制算法举例



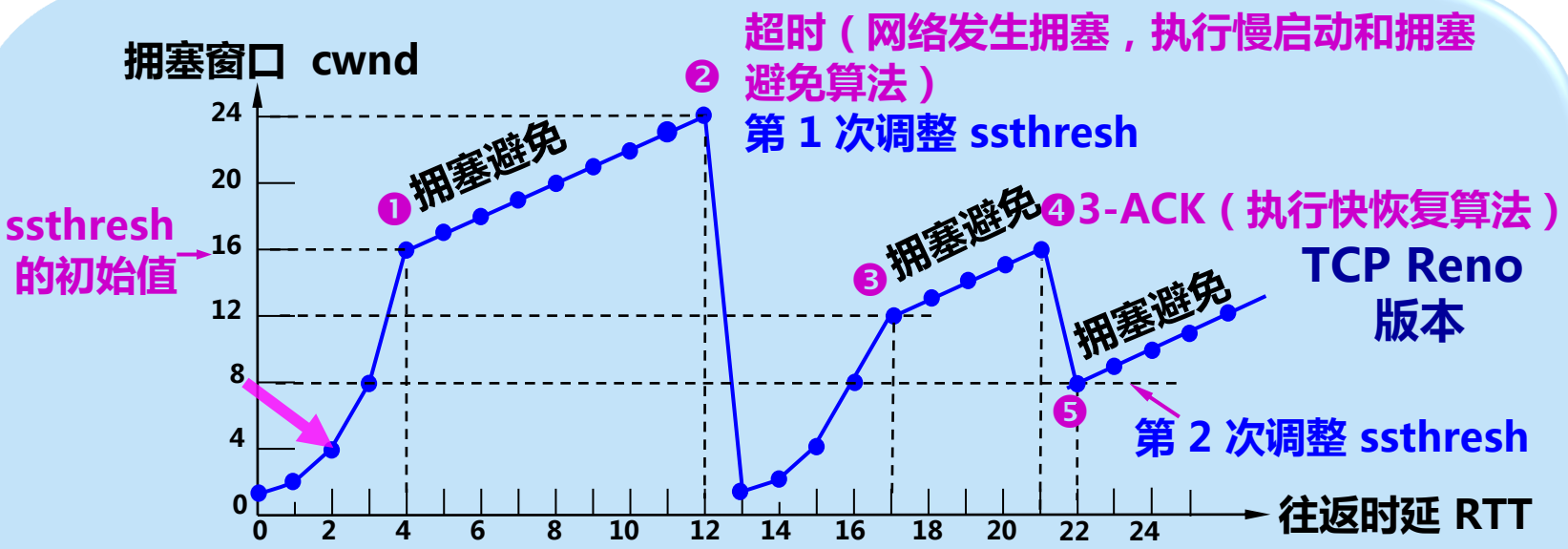
开始执行慢启动算法时，拥塞窗口 cwnd=1，发送第一个报文段。

TCP 拥塞控制算法举例



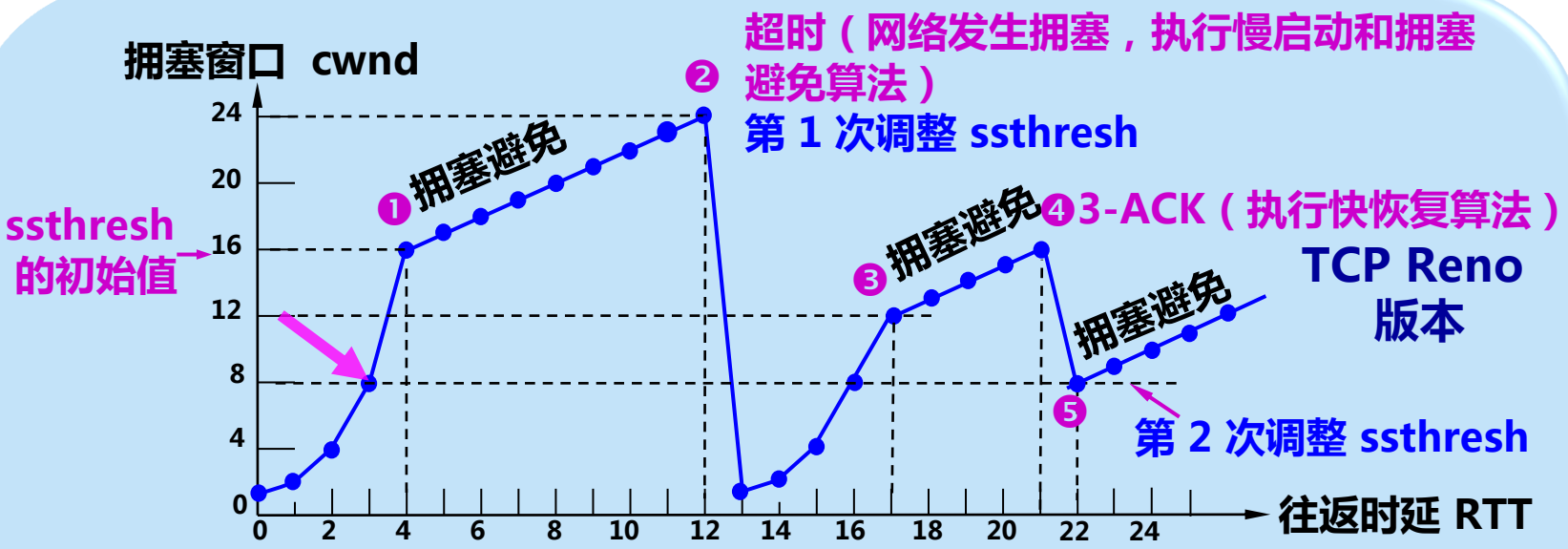
发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1，因此拥塞窗口 cwnd 随着往返时延 RTT 按指数规律增长。

TCP 拥塞控制算法举例



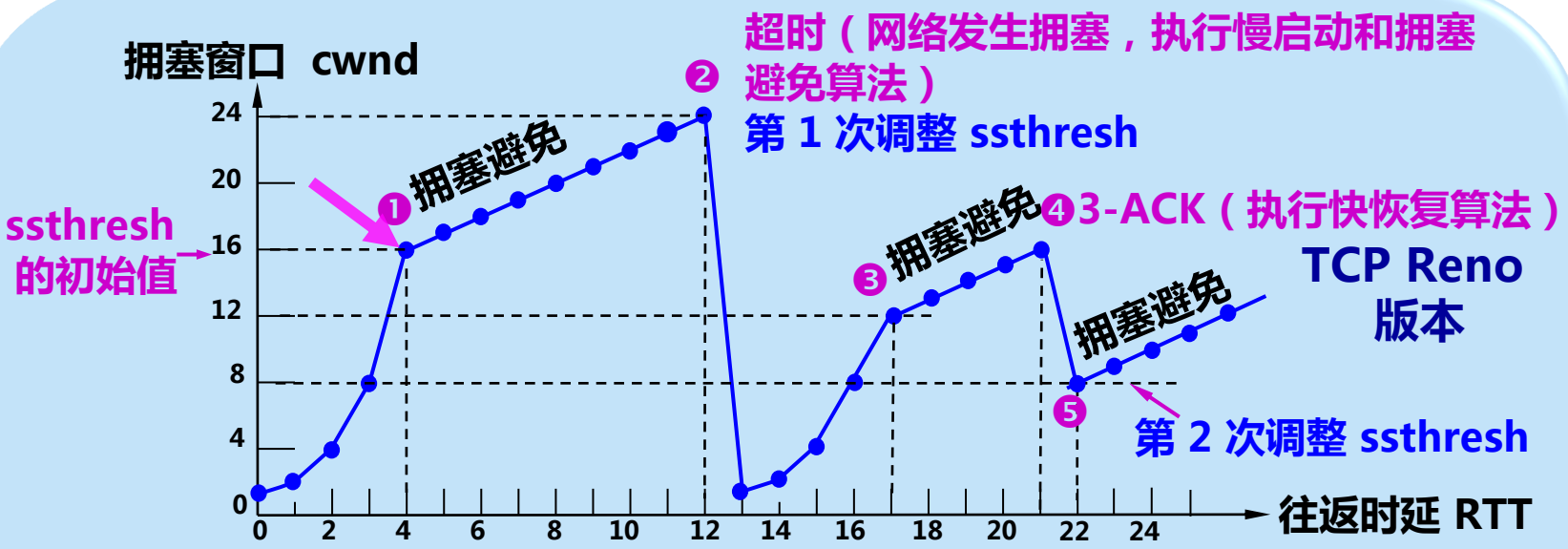
发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1，因此拥塞窗口 cwnd 随着往返时延 RTT 按指数规律增长。

TCP 拥塞控制算法举例



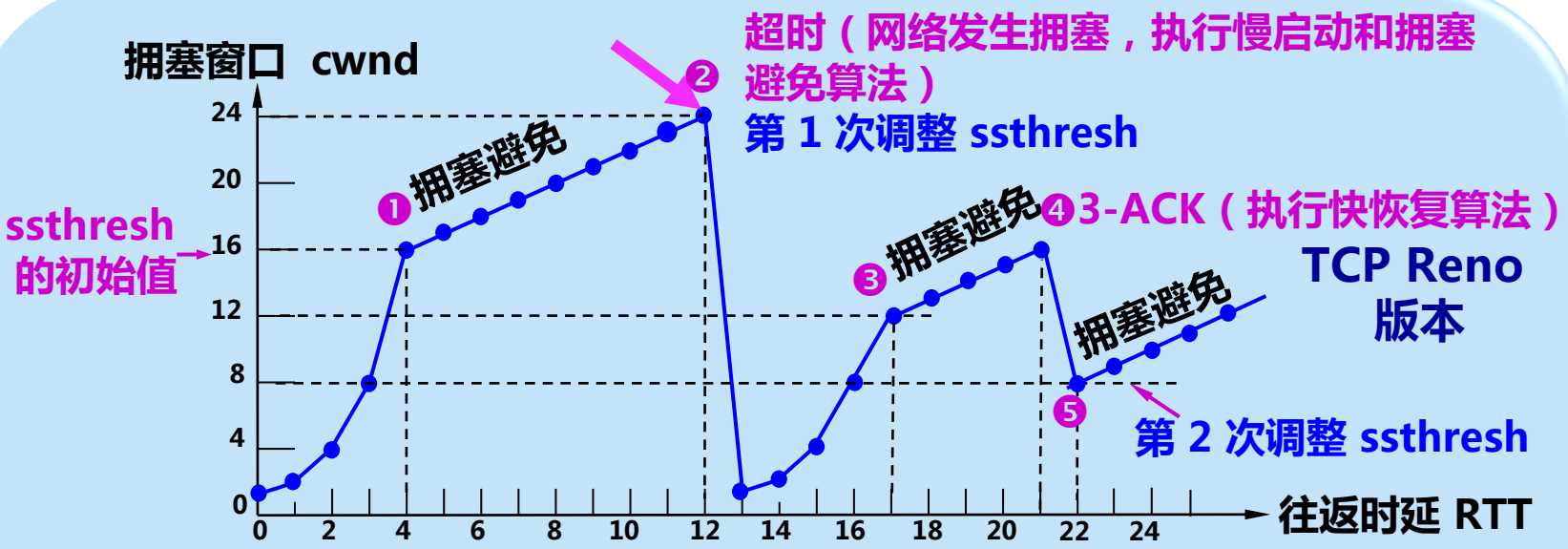
发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1，因此拥塞窗口 cwnd 随着往返时延 RTT 按指数规律增长。

TCP 拥塞控制算法举例



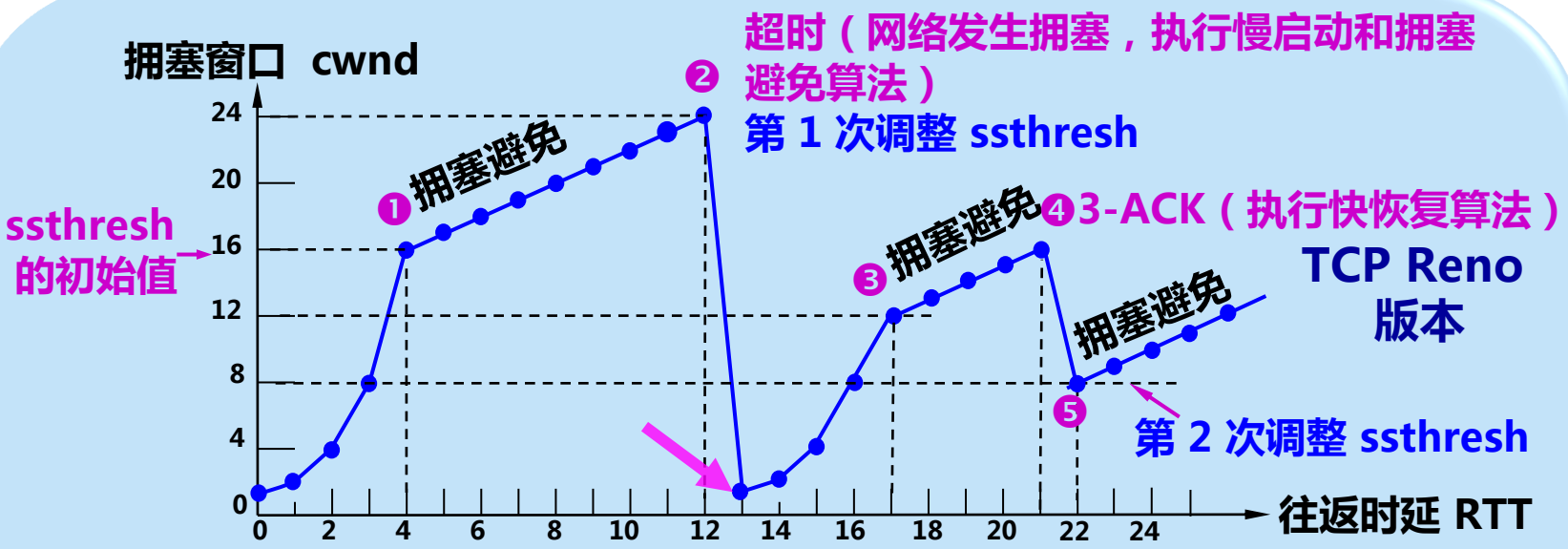
当拥塞窗口 cwnd 增长到慢启动阈值 ssthresh 时，改为执行拥塞避免算法，拥塞窗口按线性规律增长。

TCP 拥塞控制算法举例



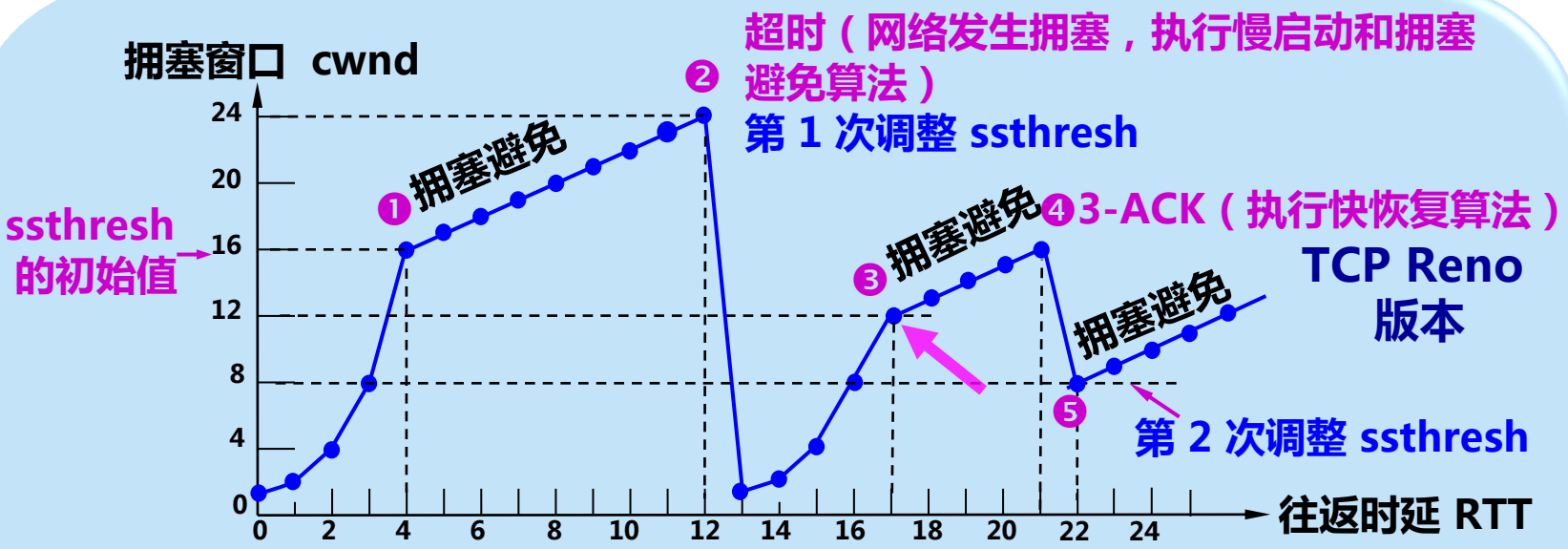
当拥塞窗口 cwnd = 24 时，网络出现了超时，发送方判断为网络拥塞。调整慢启动阈值 ssthresh = cwnd / 2 = 12，同时设置拥塞窗口 cwnd = 1，进入慢启动阶段。

TCP 拥塞控制算法举例



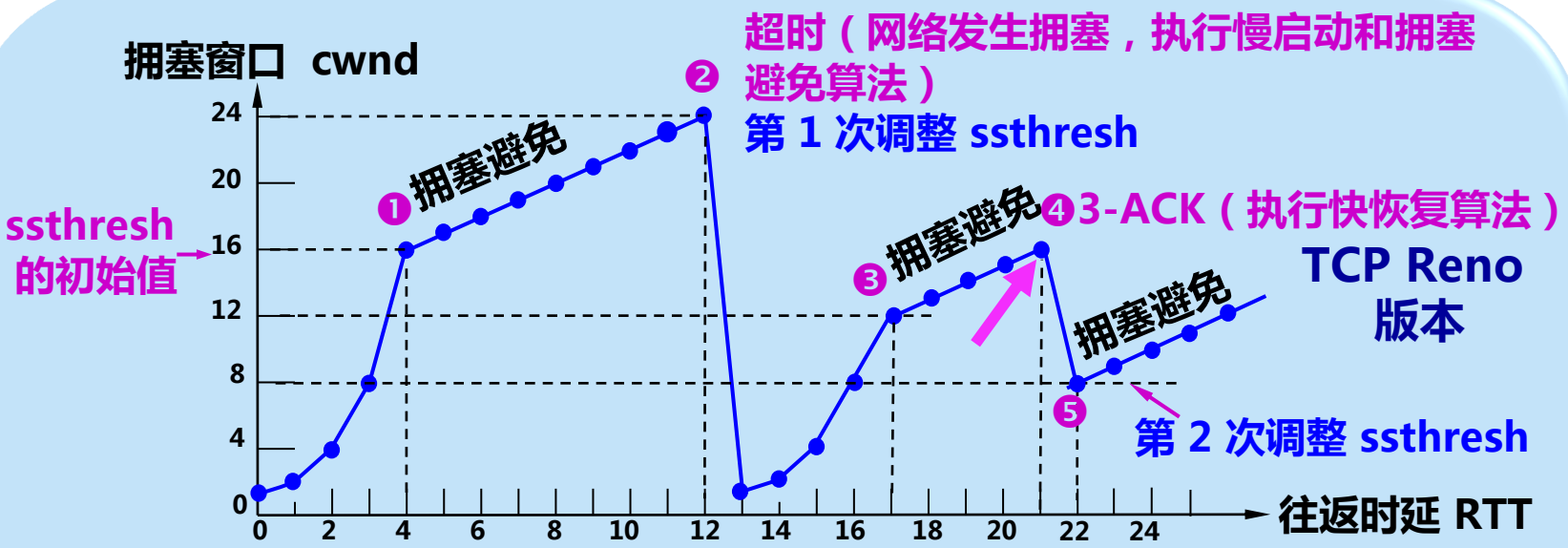
当拥塞窗口 cwnd = 24 时，网络出现了超时，发送方判断为网络拥塞。调整慢启动阈值 ssthresh = cwnd / 2 = 12，同时设置拥塞窗口 cwnd = 1，进入慢启动阶段。

TCP 拥塞控制算法举例



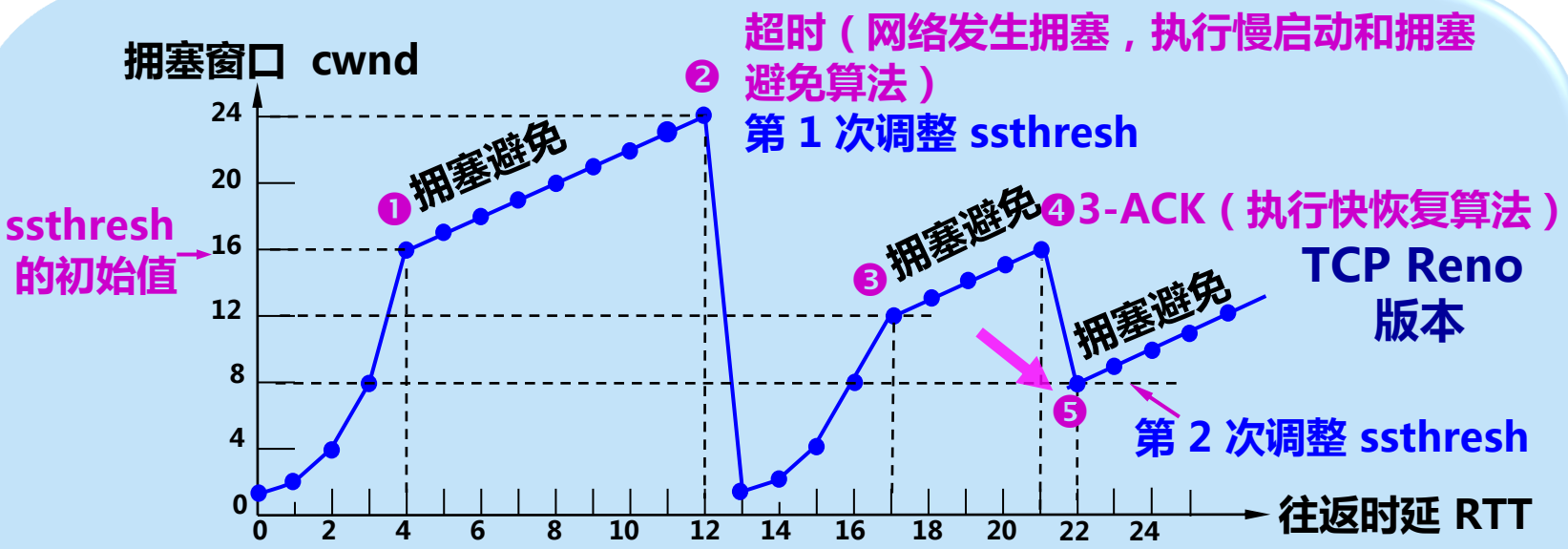
按照慢启动算法，发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1。当拥塞窗口 cwnd = ssthresh = 12 时，改为执行拥塞避免算法，拥塞窗口按线性规律增大。

TCP 拥塞控制算法举例



当拥塞窗口 cwnd = 16 时，发送方连续收到 3 个对同一个报文段的重复确认（记为 3-ACK）。发送方改为执行快速重传和快速恢复算法。

TCP 拥塞控制算法举例



执行**快速重传**和**快速恢复**算法：发送方调整慢启动阈值 ssthresh = cwnd / 2 = 8，设置拥塞窗口 cwnd = ssthresh = 8，开始执行拥塞避免算法。

TCP 拥塞控制

- 实际上，接收方的缓存空间是有限的，接收方要根据自己的接收能力设定接收窗口（rwnd）大小（以报文段为单位）。
- 发送窗口的上限值

发送窗口的上限值 = min [rwnd, cwnd]

- 当 rwnd < cwnd 时，是接收方的接收能力限制发送窗口的最大值。
- 当 cwnd < rwnd 时，是网络拥塞限制发送窗口的最大值。

主要内容