第6章 传输层(3)

课程名称：计算机网络

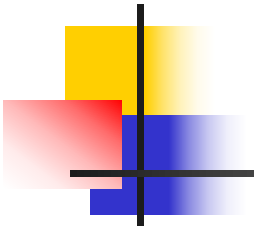
主讲教师：姚烨

课程代码：U10M11016.02

E-MAIL ：yaoye@nwpu.edu.cn

第43-44讲

2021 – 2022 学年第一学期



本节内容提要

6.8 TCP 连接管理

6.8.1 TCP 连接建立

6.8.2 TCP 连接释放

6.8.3 TCP 有限状态机

6.9 TCP 拥塞控制

6.9.1 拥塞控制的一般原理

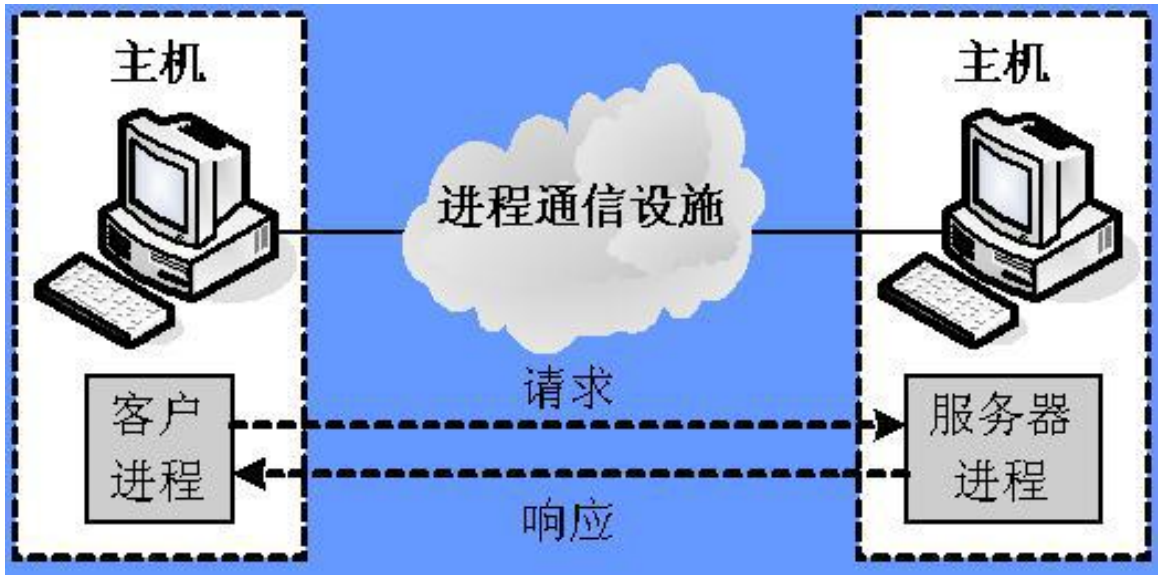
6.9.2 几种拥塞控制方法

6.8 TCP协议连接管理

- TCP协议为端到端不同应用进程间提供可靠通信服务
- TCP通信有三个阶段
  - 建立TCP连接:通过三次握手
  - 数据传输：可靠通信
  - 释放TCP连接：通过四次握手
- 连接的管理:使连接的建立和释放都能正常地进行，在数据传输阶段维护连接处于正常状态。

客户/服务器方式

- TCP 连接的建立、数据传输和释放连接都是采用客户/服务器方式。
  - 主动发起连接请求的应用进程叫做客户(client)。
  - 被动等待连接请求的应用进程叫做服务器(server)。



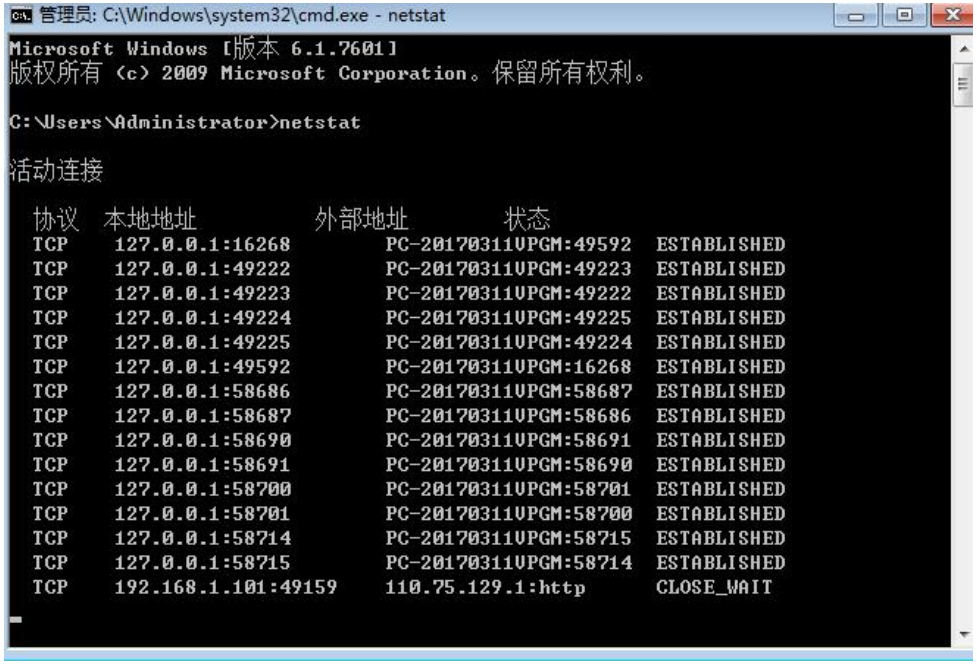
6.8.1 建立连接

- 连接建立过程中要解决以下三个问题：
  - 要使一方能够确知道对方应用进程存在。
  - 允许双方协商一些参数（如最大报文段长度MSS，最大窗口大小，初始序号、是否采用SACK以及其他参数等）。
  - 对网络资源（如缓存大小，各种定时间器：如重发，坚持、保活；连接状态表）进行分配和初始化。

TCP连接表

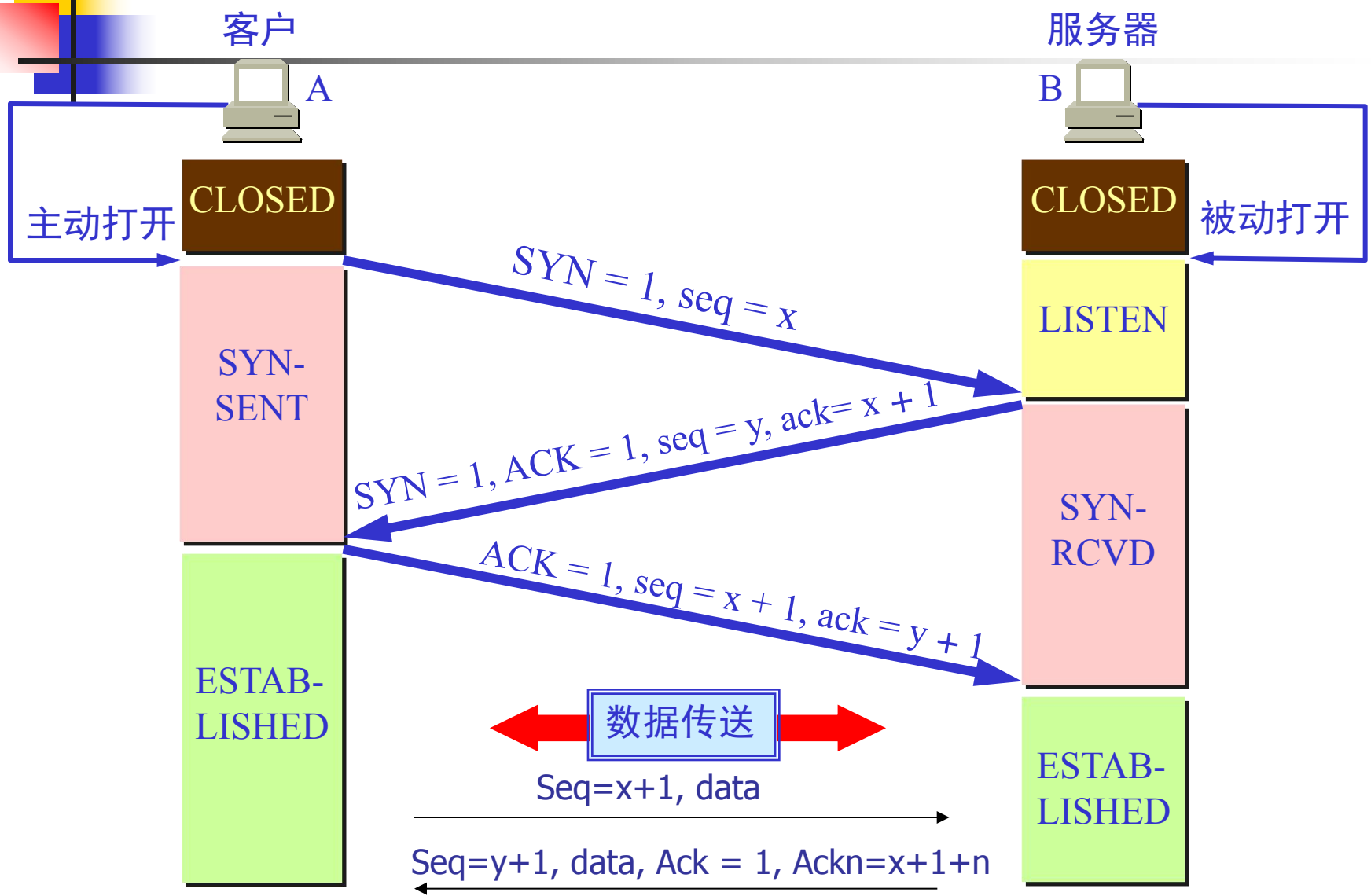
- TCP 连接表对每个连接都登记了其连接信息。除本地和远地的 IP 地址和端口号外，记录每一个连接所处状态。

	本地IP地址	本地端口	远地IP地址	远地端口	连接状态
连接1					
连接2					
连接n					



6.8.1 TCP 的连接建立

用三次握手建立 TCP 连接的各状态



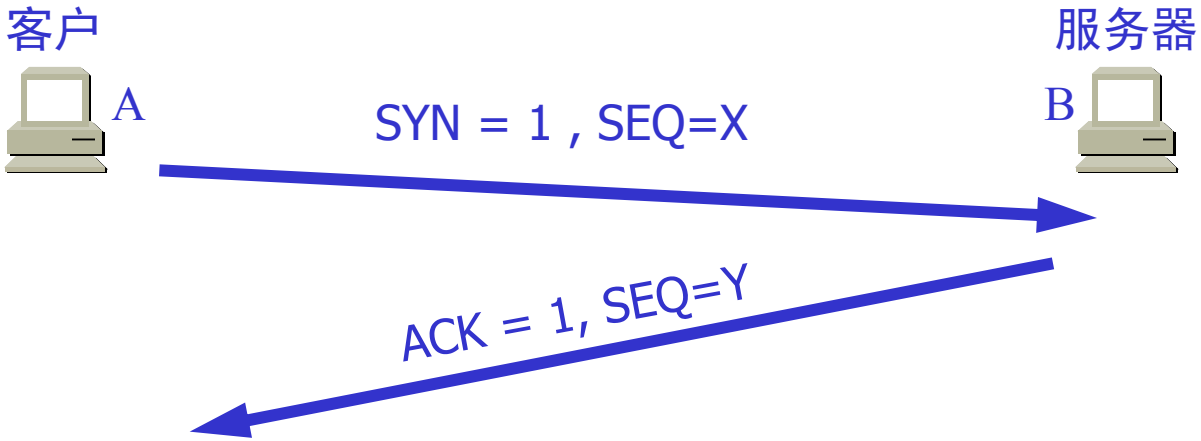
为什么通过三次握手建立连接？

- 问题：防止“已延迟TCP连接请求”发送给对方.
- 现象：假设主机A发出TCP连接请求,但未收到确认,A重传TCP连接请求,后来受到确认,建立连接;数据发送完毕后,释放连接.
- 整个过程中A发送了两个连接请求报文,其中有一个有效.(暂且假设为第二个)

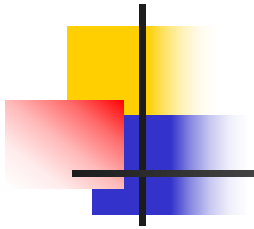
为什么进行三次握手？

- 为什么发送方没有收到第一个连接请求对应确认:
  1) 第一个连接请求丢失;
  2) 第一个TCP连接请求在网络传输延迟时间长;
  3) 确认丢失或出错;
  4) 确认在网络传输延迟时间长.
  如果情况2)，假设采用二次握手，会发生什么情况?
  浪费网络资源；安全问题。

假设A,B之间TCP连接采用两次握手，而不是三次握手建立一个TCP连接？



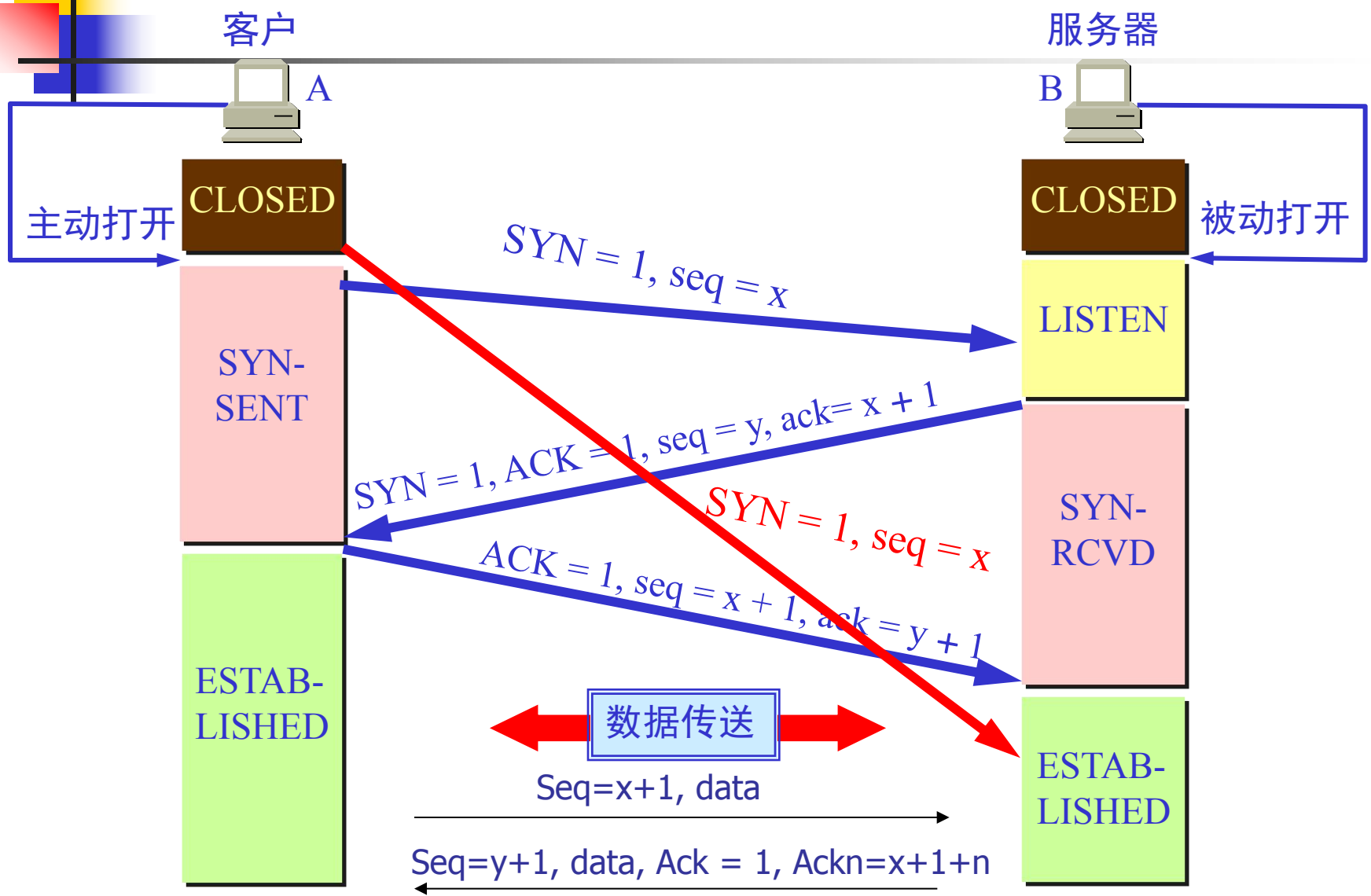
分析：由于A与B之间通信已经结束，客户和服务器进程已经释放连接（释放网络资源）；A进程（或A计算机）无法对ACK应答处理，B以为连接已经建立，分配了网络资源，等待A发送TCP报文段，造成网络资源浪费（DOS攻击），通信不安全性。



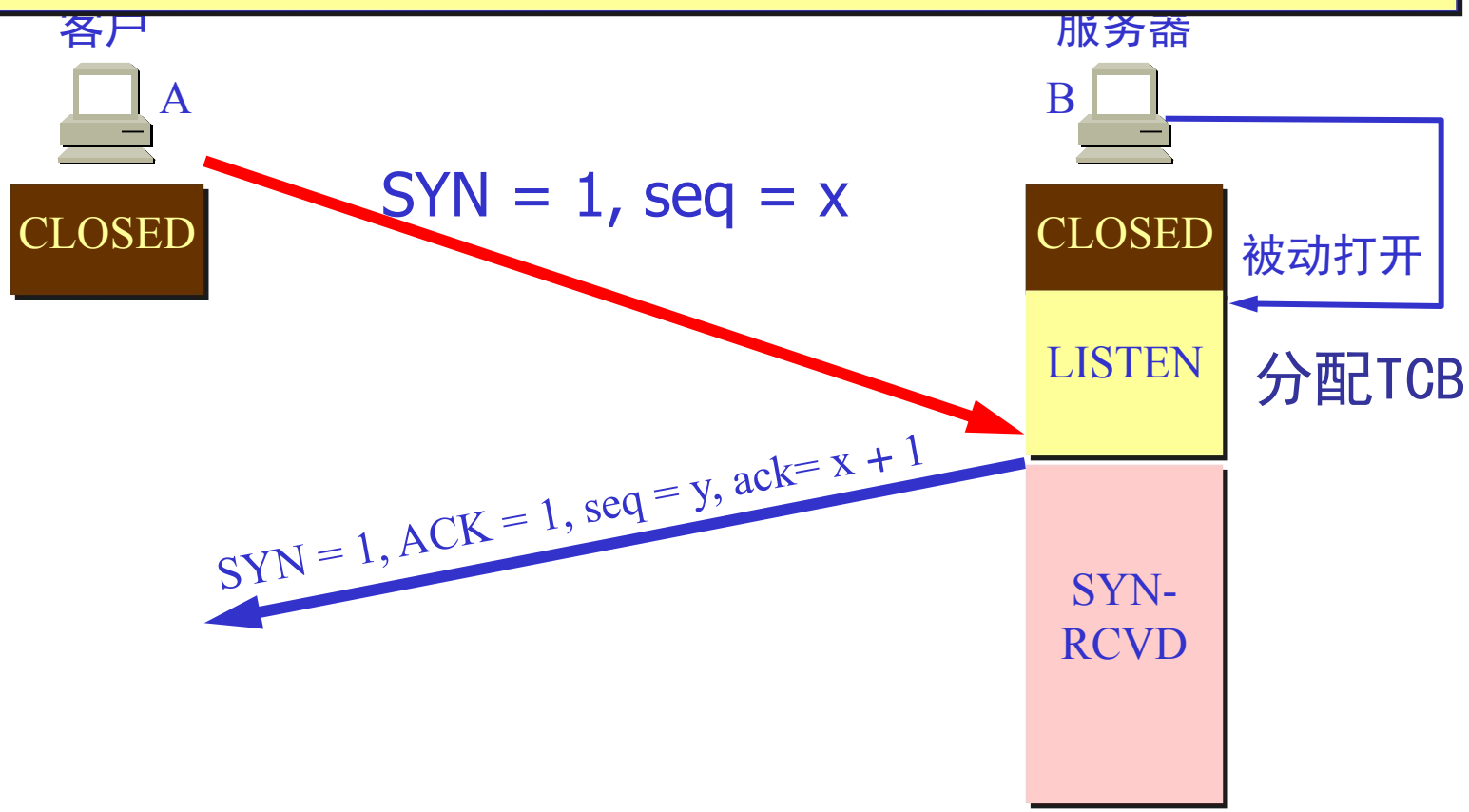
三次握手能解决以上问题吗？

（无法建立连接：克服资源浪费和通信安全问题）

三次握手：A与B通信时，接收到延迟的第一个连接请求解决方法：序号不在接收窗口内，重复段丢弃！

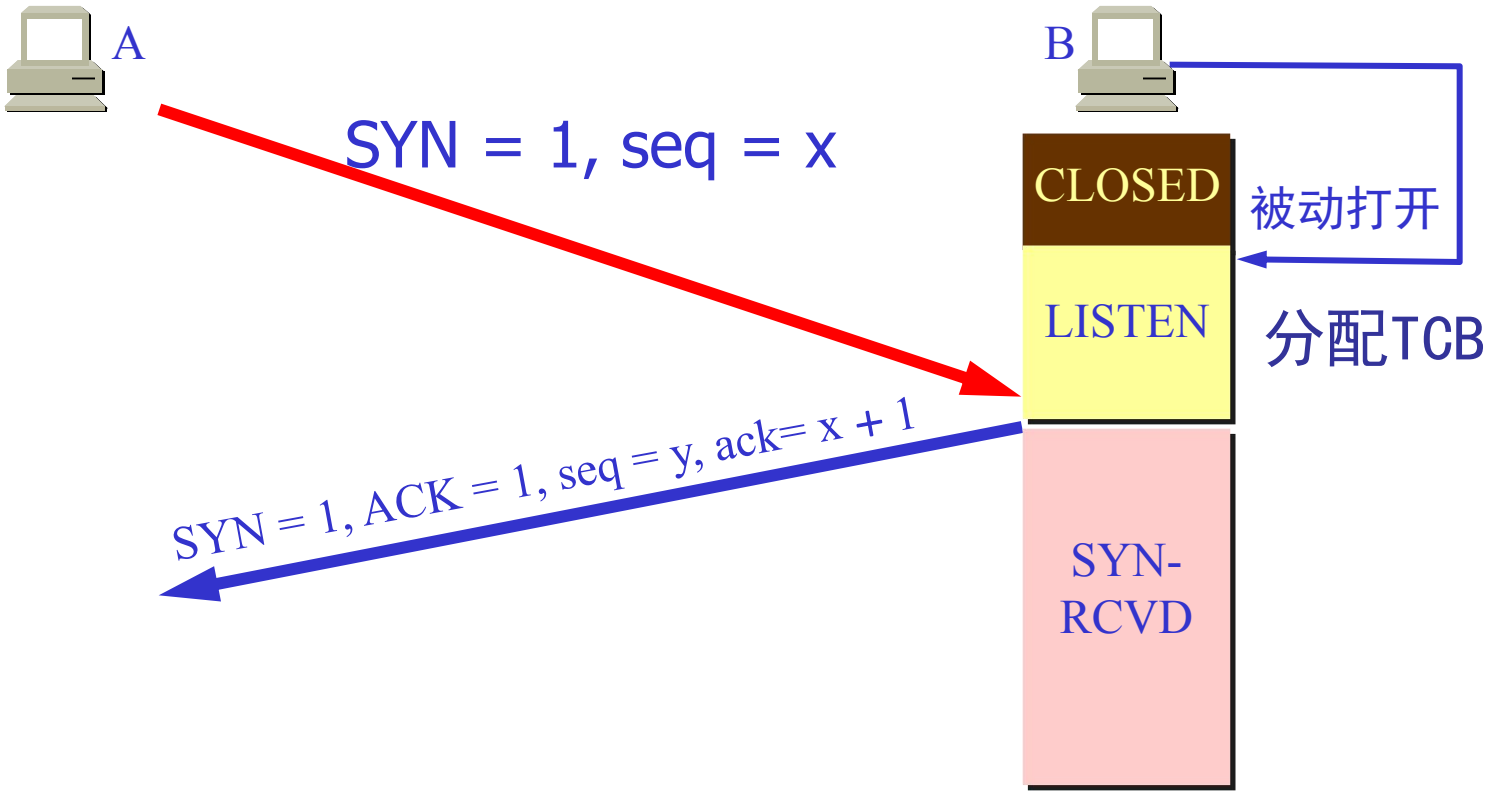


如果采用三次握手：A、B通信结束，连接已关闭,（A进程关闭、B进程运行）。解决方法：由于A进程连接已关闭，A临时端口已经释放；B发送的应答和反向连接请求找不到目的端口对应的接收缓存，A将其丢弃，同时给B发送“端口不可达”ICMP差错报告报文，但不会给B发送ACK应答；



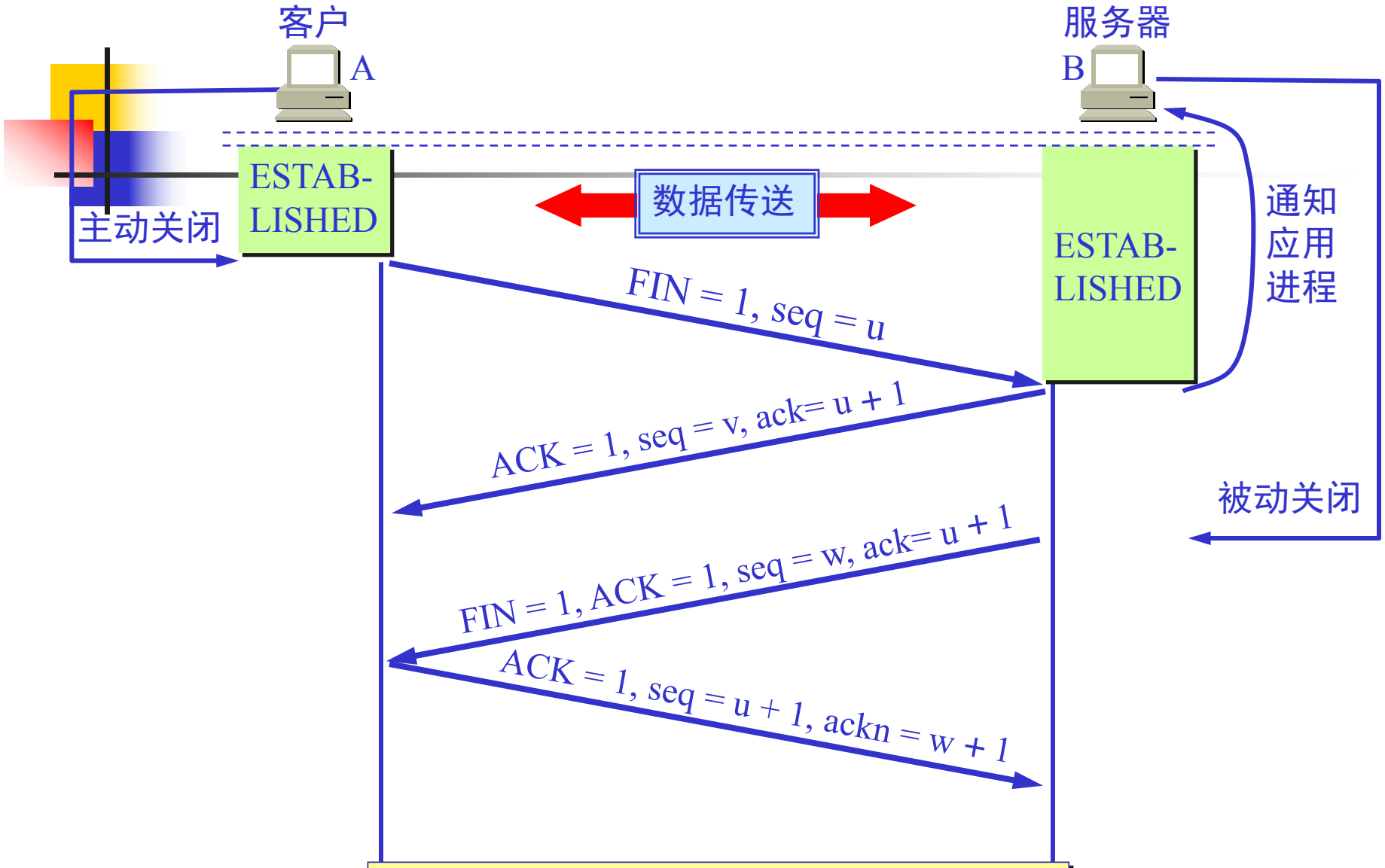
三次握手是原子操作，A没有发送应答给B，原子操作失败，连接无法建立。

如果采用三次握手：A、B通信结束，A关机，B进程还在运行;
解决方法： A关机，应用进程没有运行，B发送的应答和反向连接请求在最后一个路由器不会转发给A（由于ARP协议），该路由器丢弃，并发送“主机不可达”ICMP差错报告报文，不会给B发送ACK应答;



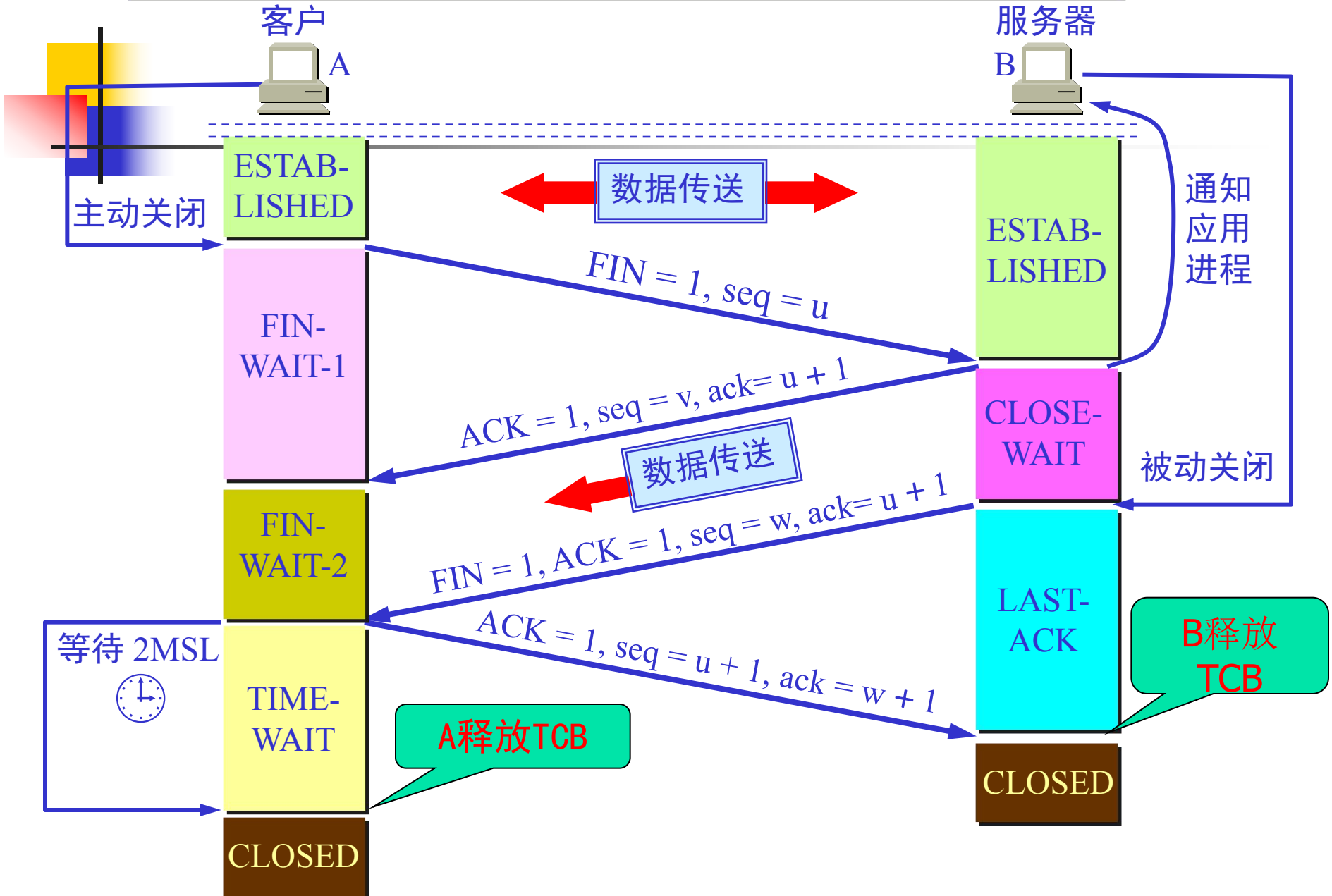
三次握手是原子操作，A没有发送应答给B，原子操作失败，连接无法建立。

6.8.2 TCP 四次挥手释放连接



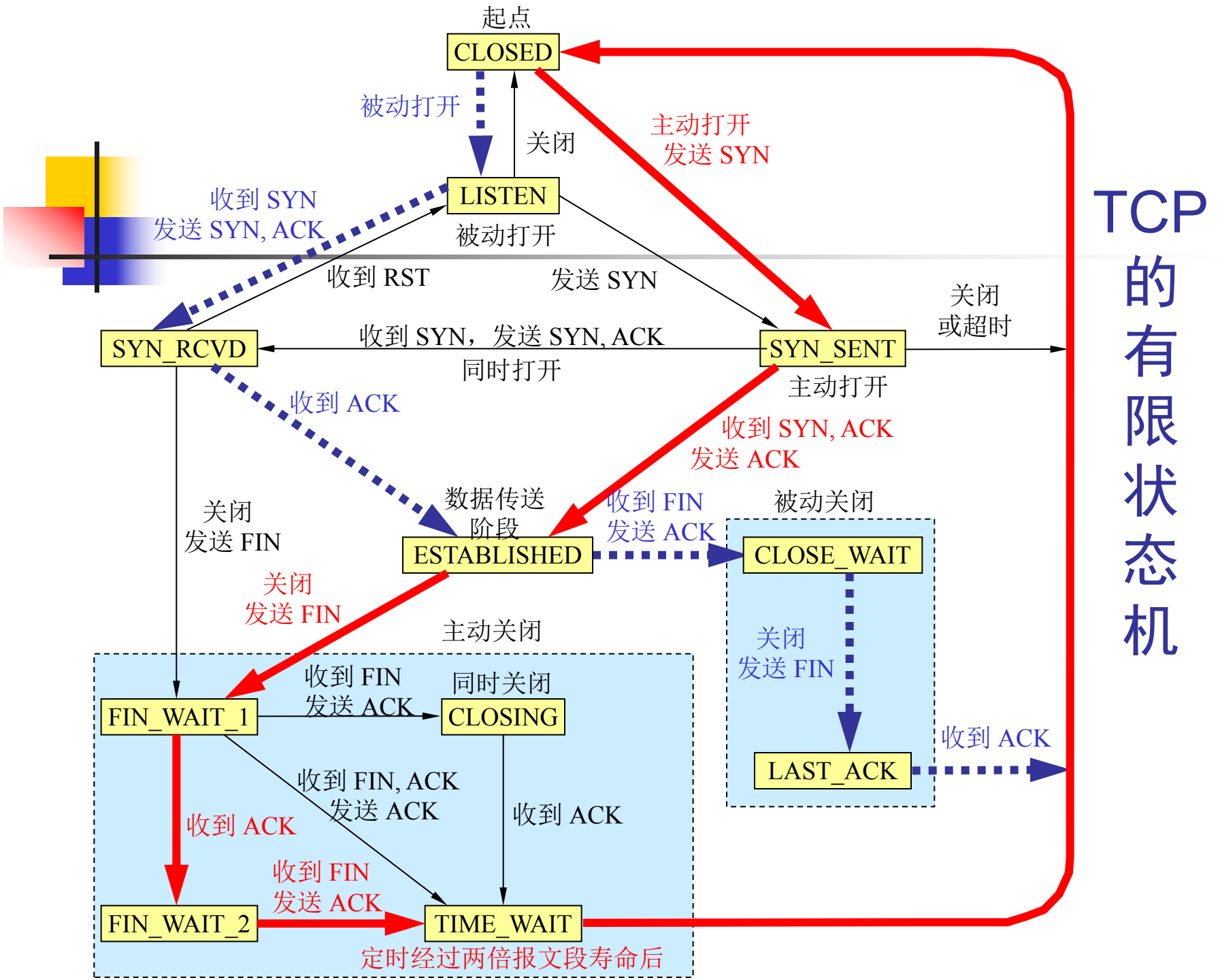
- 四次握手是一个原子操作

TCP 连接必须经过时间 2MSL 后才真正释放掉。
TCP 标准规定:FIN置1的报文消耗一个序号，不能携带数据



6.8.3 TCP 有限状态机

- TCP 有限状态机的图中每一个方框都是 TCP 可能具有的状态。
- 每个方框中的大写英文字符串是 TCP 标准所使用的 TCP 连接状态名。
- 状态之间的带箭头连线表示可能发生的状态变迁。
- 连线旁边的字，表明引起这种变迁的原因（发生了什么事件）。
- 图中有三类不同的带箭头连线。
  - 粗实线箭头表示对客户进程的正常变迁。
  - 粗虚线箭头表示对服务器进程的正常变迁。
  - 另一种细线箭头表示异常变迁-分析为什么会出现？


起点
CLOSED
被动打开
关闭
主动打开
发送 SYN
LISTEN
收到 SYN
发送 SYN, ACK
被动打开
收到 RST
发送 SYN
关闭
或超时
SYN_RCVD
收到 SYN，发送 SYN, ACK
同时打开
SYN_SENT
主动打开
收到 ACK
收到 SYN, ACK
发送 ACK
关闭
发送 FIN
数据传送
阶段
收到 FIN
发送 ACK
被动关闭
ESTABLISHED
CLOSE_WAIT
关闭
发送 FIN
主动关闭
收到 FIN
发送 ACK
同时关闭
FIN_WAIT_1
CLOSING
关闭
发送 FIN
收到 ACK
收到 FIN, ACK
发送 ACK
收到 ACK
LAST_ACK
收到 ACK
收到 FIN
发送 ACK
FIN_WAIT_2
TIME_WAIT
定时经过两倍报文段寿命后

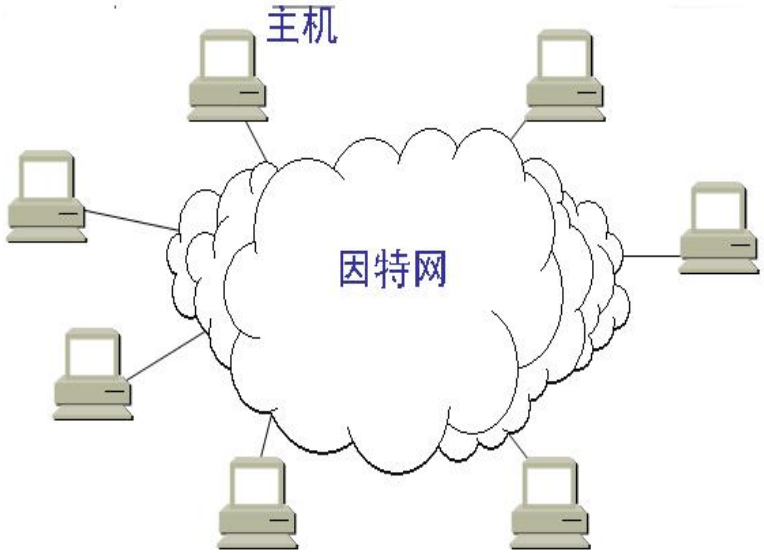
TCP的有限状态机

TCP协议总结

- TCP可靠的数据传输服务
  - 差错控制（序号+确认+超时重传）
  - 流量控制（可变大小滑动窗口）机制
  - 拥塞控制
- TCP三次握手（建立和释放连接），确保通信逻辑连接的安全性，防止主机资源浪费。
- IP地址：寻址到主机，提供端到端数据传输服务。
- IP地址+端口号：寻址到主机上的应用进程，提供端到端不同应用进程间数据传输服务。
- 发送方与接收方之间通过一个五元组来确定之间的通信关系
  《源IP地址，源PORT，目的IP地址，目的端口号，协议》

6.9 TCP的拥塞控制
6.9.1 拥塞控制原理

- 网络资源：网络交换设备的链路容量（带宽）、缓存和处理器等。
- 在某段时间，若网络对交换设备中某资源的需求超过了该设备所能提供的可用部分，网络的性能就要变坏——产生拥塞(congestion)。



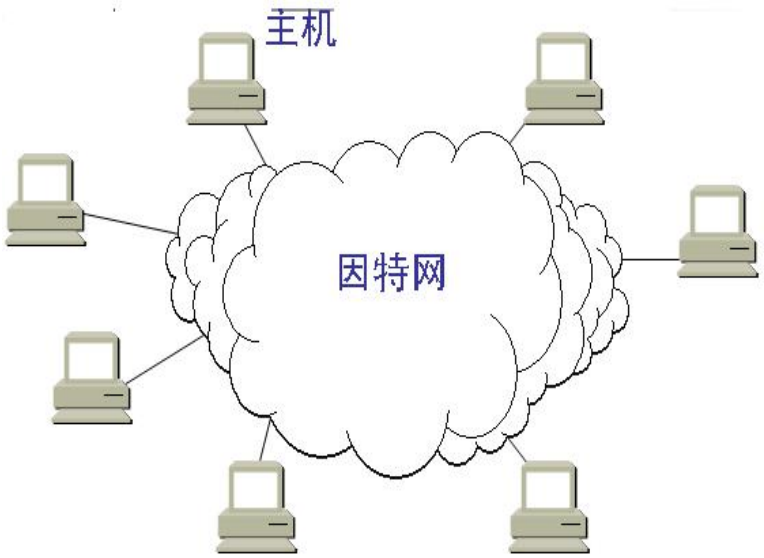
通信资源：带宽；
存储资源：缓存
计算资源：处理器

6.9 TCP的拥塞控制
6.9.1 拥塞控制原理

- 网络拥塞产生条件：

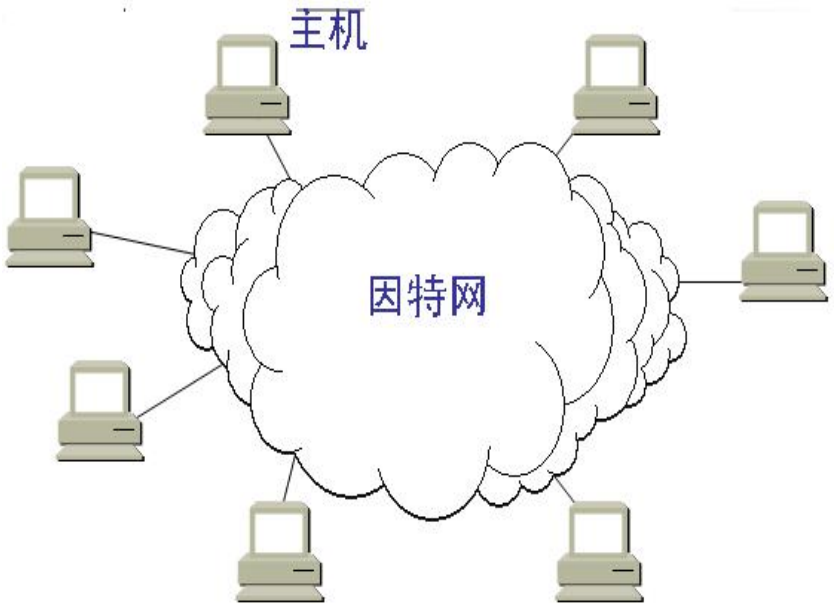
  网络对资源需求总和 >交换设备可用资源

- 若网络中有许多资源同时不能满足需求，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。



拥塞控制与流量控制的关系

- 拥塞控制目的：防止有过多的数据输入到网络，以避免网络中的路由器或链路过载。
- 拥塞控制基本原则：网络能够承受现有的网络负荷，需要提前进行拥塞控制。
- 拥塞控制是一个全局性过程，涉及到网络中所有的主机、网络交换设备。

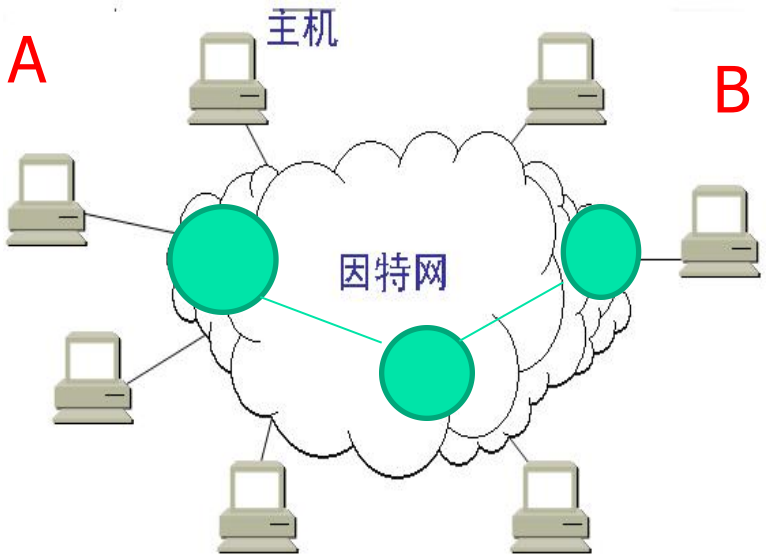


拥塞控制与流量控制的关系

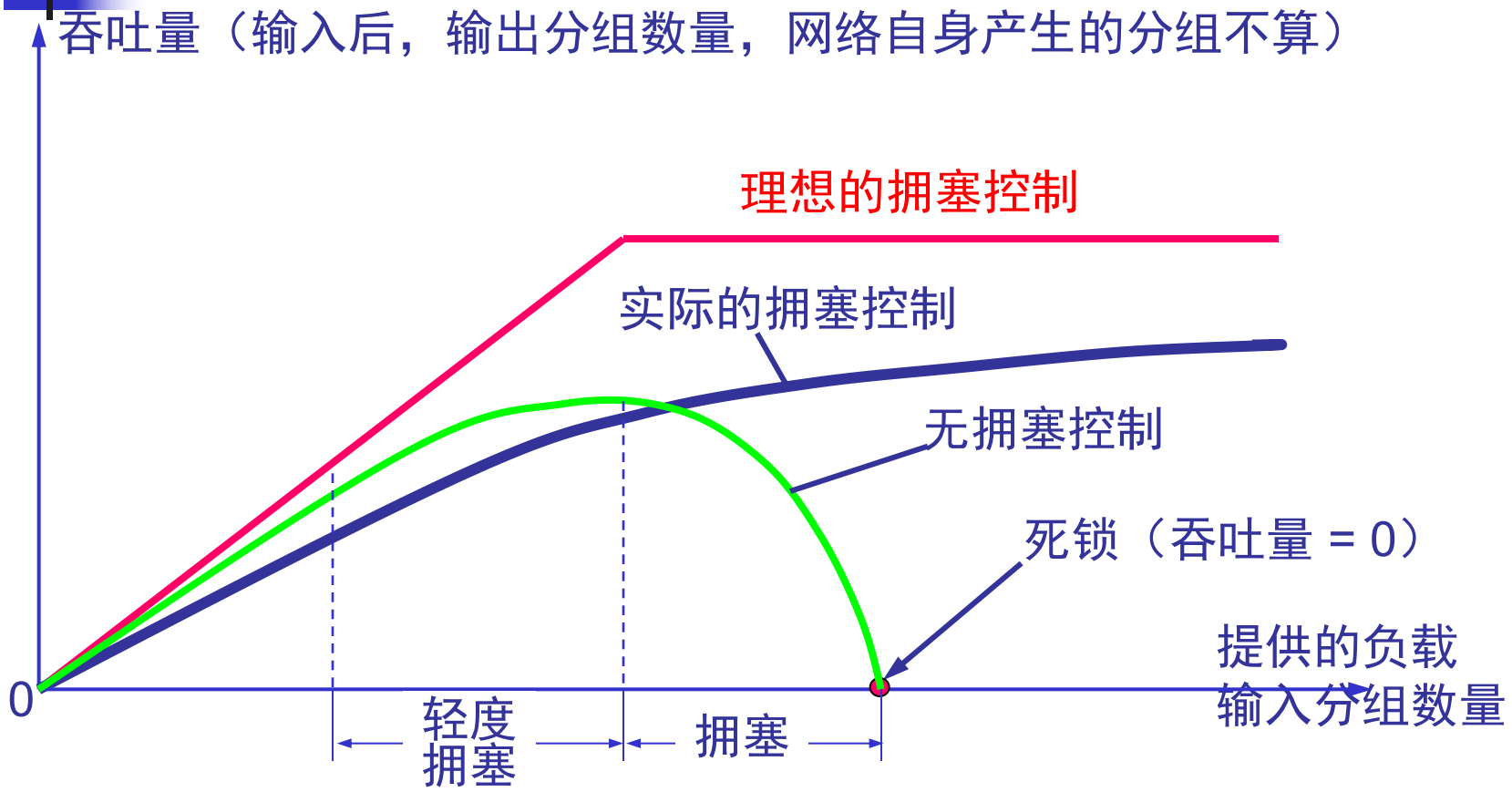
- TCP通信中，发送端如果重发定时器超时还没有收到ACK应答，发送端认为网络发生拥塞；发送端通过拥塞控制降低发送方发送速率解决，但发送方无法知道发生网络拥塞具体情况：何地、什么具体原因引起。

拥塞控制与流量控制的关系

- 流量控制
  - 传输层研究对象：接收端接收缓存；发生在发送端和接收端之间；
  - 数据链路层研究对象：相邻节点接收缓存，发生在相邻节点；
- 流量控制所要做的就是抑制发送端发送数据的速率，使接收端来得及从接收缓冲区中将数据取走，不要让接收缓冲区溢出。



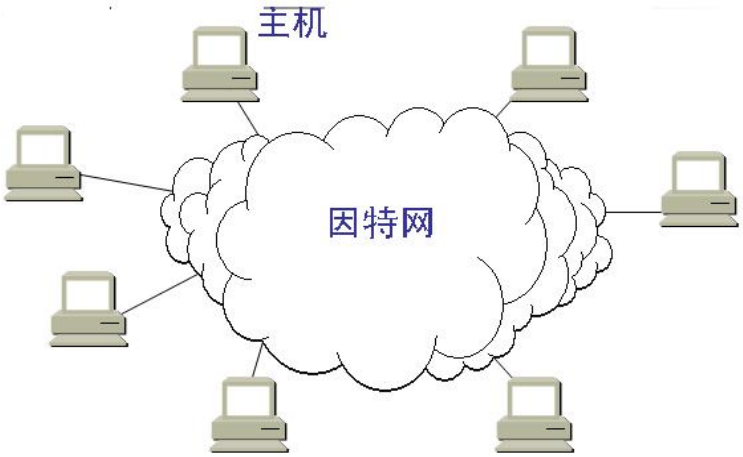
网络拥塞控制作用



- 实际网络中，随着负载增加，网络吞吐量增长率反而减少
- 网络进入拥塞（无控制），再增加负载到一个数值，网络吞吐量反而降为0。

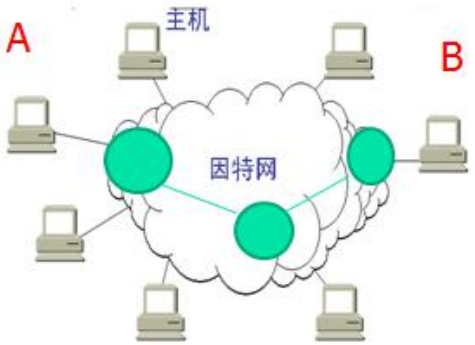
网络拥塞控制原理

- 拥塞控制是一个动态（非静态）过程，很难设计合理方案；
- 当前网络朝着高速化方向发展，很容易出现交换设备接收缓存不够大（或CPU过载）而造成分组的丢失（或来不及处理）。
- 在许多情况下，正是拥塞控制（考虑不周）本身成为引起网络性能恶化甚至发生死锁的原因，这点应特别引起重视。



传输层拥塞控制

- 1999年，RFC2581定义了传输层四种拥塞控制方法：
  - 慢启动（slow start）
  - 拥塞避免(congestion avoid)
  - 快速重传(fast retransmit)
  - 快速恢复(fast recovery)



- 此后，RFC2582，RFC3390对以上方法进行了改进。
- 介绍传输层拥塞控制前，首先提出两个假设：
  - 数据是单向传输，另一方可以发送ACK应答；
  - 接收方有足够大的接收缓存，暂时不考虑流量控制，因而发送窗口主要由网络拥塞程度（拥塞窗口）决定。

6.9.2 TCP拥塞控制方法
1. 慢启动和拥塞避免

- 发送方维护一个拥塞窗口 cwnd（congestion window)的状态变量。
  - 拥塞窗口的大小取决于网络当前拥塞程度，并且动态地变化。
  - 发送方主要通过拥塞窗口大小控制发送窗口；暂不考虑流量控制，则发送窗口一般应该小于等于拥塞窗口（这里取等于）；
- 发送方控制拥塞窗口原则是：
  - 只要网络没有出现拥塞现象，拥塞窗口就增大一些，以便把更多的TCP报文段发送出去；但只要网络出现拥塞，拥塞窗口就减小一些，以减少输入到网络中的TCP报文段数量。

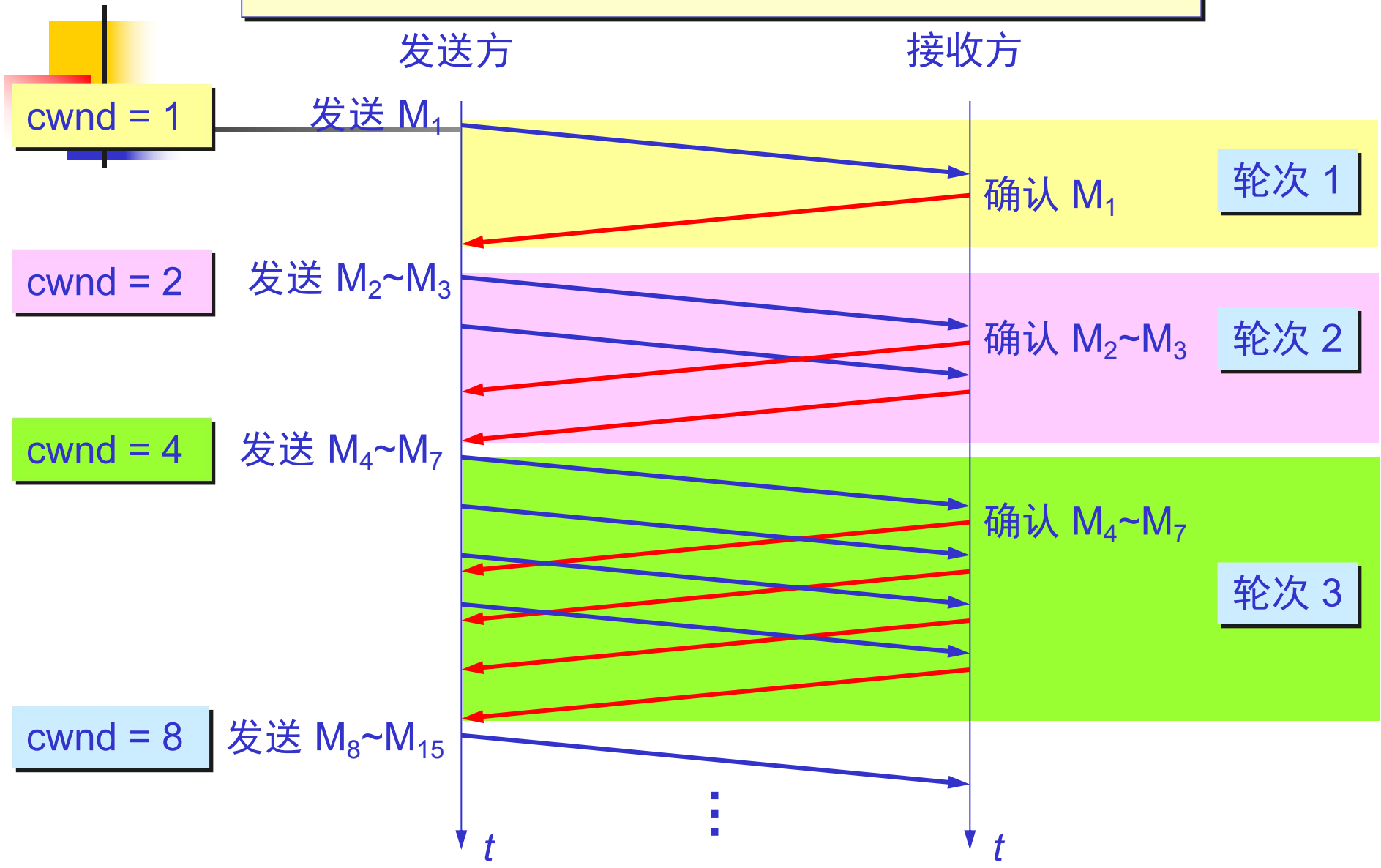
6.9.2 TCP拥塞控制方法
1. 慢启动和拥塞避免

- 发送方如何发现网络发生拥塞？
  - 当网络发生拥塞时，路由器接收缓存溢出，路由器利用拥塞控制策略需要丢弃分组；
  - 发送方不可能收到丢弃分组对应TCP段的确认，重发定时器超时，需要重发该TCP报文段；此时，发送方就认为网络发生了拥塞。
- 注意
  - 发送方没有收到确认原因很多，有可能TCP报文段在网络传输过程中出现了差错，接收方直接丢弃引起；此时网络并没有发生拥塞；或者确认丢失或（差错）。
  - 但现在线路大部分采用光纤，发生此情况概率很小。

方法1：慢启动机制

- 基本思想：当发送端主机刚开始发送TCP报文段，由于不知道网络负载情况，可采用试探的方法，逐渐由小到大增加拥塞窗口（相当发送窗口）大小；
- 发送方通信开始设置拥塞窗口 cwnd = 1，即设置为一个最大报文段 MSS 的数值，第一次只能发送一个TCP报文段；
- 发送方每收到一个对新的TCP报文段的确认后，将拥塞窗口加 1，即多发送一个TCP报文段。
- 实践证明：用这样的方法逐步增大发送端的拥塞窗口 cwnd，可使分组输入到网络的速率更加合 理。
- 注意：为方便起见，用TCP报文段个数作为窗口大小单位（实际上是采用字节）。

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。



方法1：慢启动机制 (transmission round)

- 使用慢启动方法后，每经过一个传输轮次，拥塞窗口 cwnd 就加倍（指数增加）。
- “传输轮次”：TCP把拥塞窗口 cwnd 内TCP报文段连续发送出去，并收到了对已发送的最后一个字节确认。
- 例如，拥塞窗口 cwnd = 4，传输轮次是指发送方连续发送 4 个报文段，并收到这 4 个报文段的确认，总共经历的时间。
- 慢启动中“慢”解释：
  - “慢”并不是指cwnd增长速率慢（一轮后，指数增加），而是指TCP开始发送报文段时， cwnd=1，只能发送一个TCP段，对网络拥塞情况进行试探，然后逐渐增加cwnd，对防止或避免拥塞是有利的。
  - 试想：如果一开始不知道网络拥塞情况下，一下子发送大量TCP报文段是比较危险的，有可能加剧拥塞。

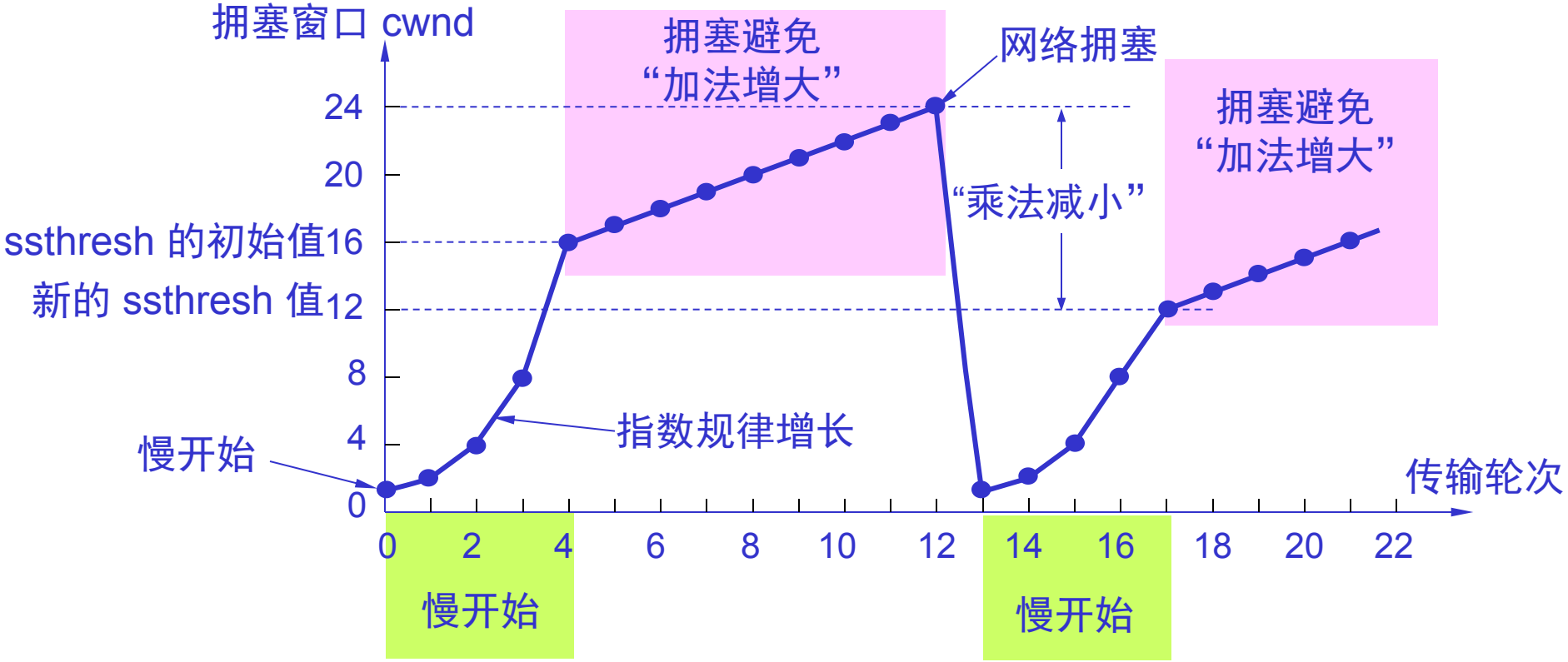
设置慢启动门限状态变量 ssthresh

- 为了防止拥塞cwnd增长过大引起网络拥塞，发送方设置了一个慢启动门限 ssthresh ，用法如下：
- 当 cwnd < ssthresh 时，使用慢启动方法。
- 当 cwnd > ssthresh 时，停止使用慢启动方法而改用拥塞避免方法。
- 当 cwnd = ssthresh 时，或者使用慢启动算法，或者使用拥塞避免算法。
- 拥塞避免机制
  - 让拥塞窗口 cwnd 缓慢增大，即每经过一个传输轮次，把发送方的拥塞窗口 cwnd 加 1（慢启动是加倍），使拥塞窗口 cwnd 按线性规律缓慢增长。

当网络出现拥塞时

- 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（判断依据：重发TCP段），就要把慢启动门限 ssthresh 设置为出现拥塞时的发送方窗口值（拥塞窗口cwnd ）的一半（但不能小于2）。
- 然后把拥塞窗口 cwnd 重新设置为 1（发送窗口=1），执行慢启动方法。
- 目的：迅速减少主机发送到网络中的TCP报文段数量，使得发生拥塞的路由器有足够时间把接收队列中累积的分组处理完毕。

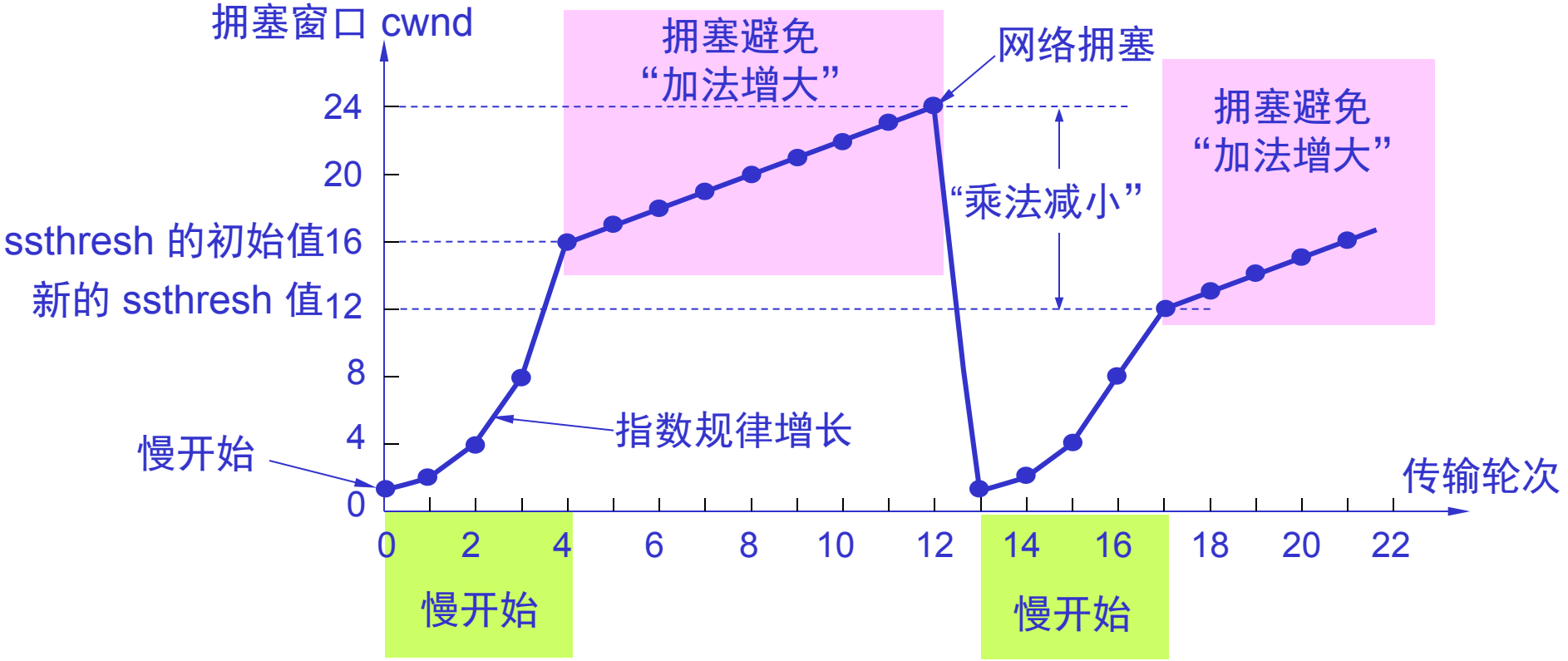
慢启动和拥塞避免算法的实现举例



当 TCP 连接建立初始化时，发送方将拥塞窗口置为 1。图中的窗口单位不使用字节而使用报文段。

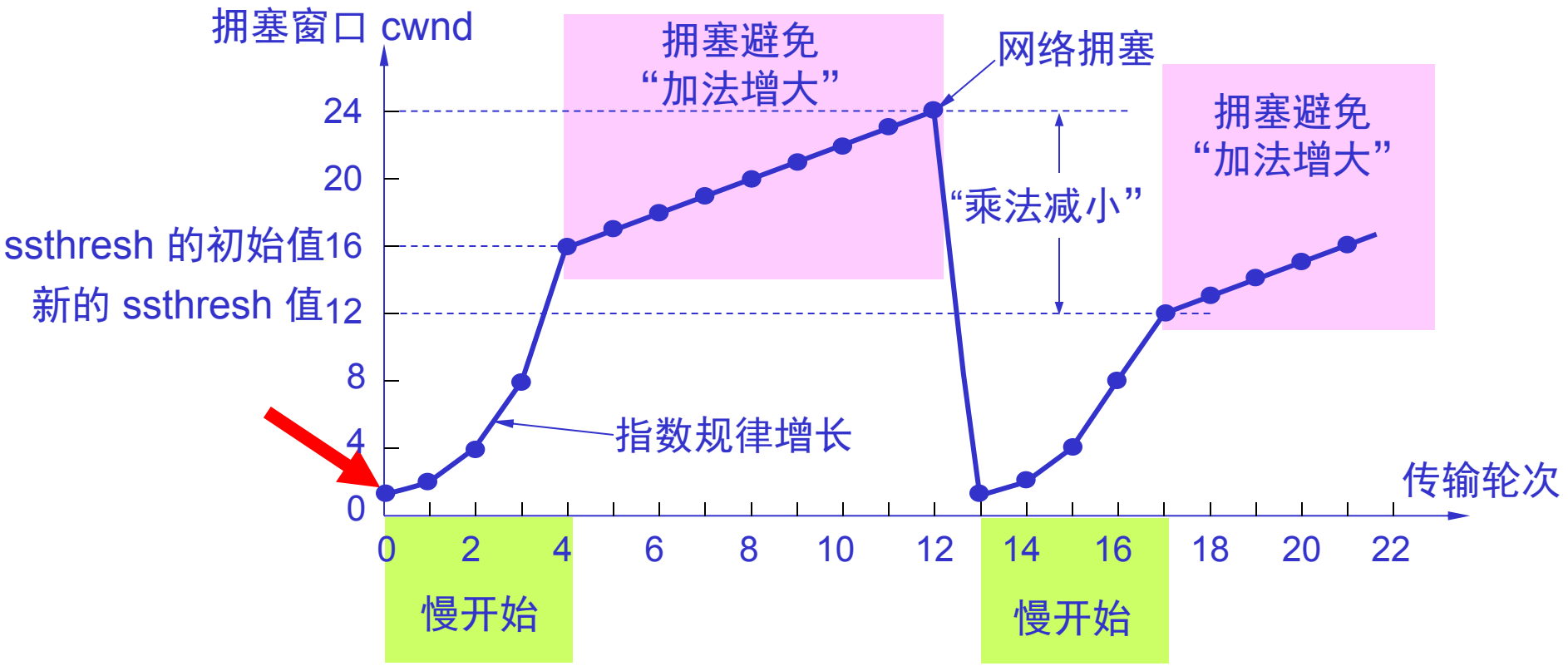
慢启动门限的初始值设置为 16 个TCP报文段（建立连接初始化时设置），即 ssthresh = 16。

慢启动和拥塞避免算法的实现举例



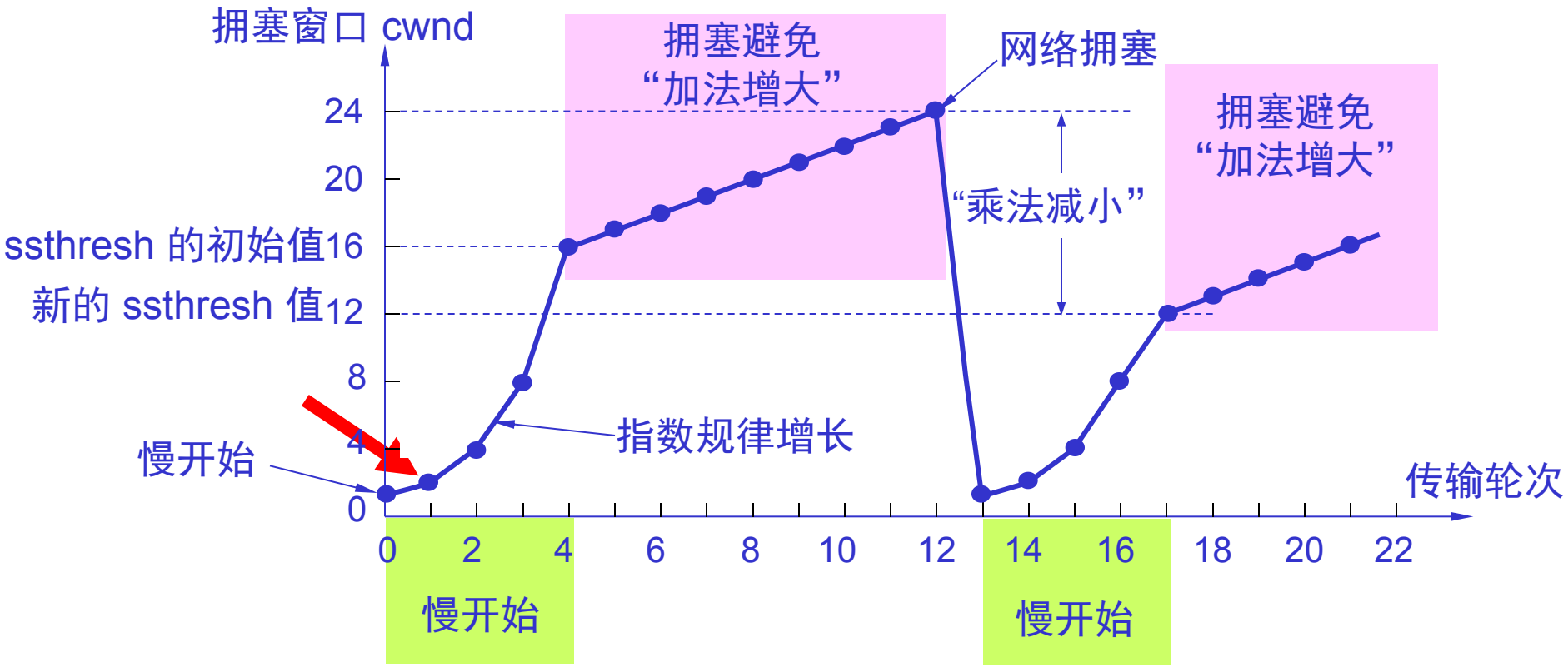
发送端的发送窗口不能超过拥塞窗口 cwnd 和接收端窗口有效大小 rwnd 中最小值；我们假定接收端窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值。

慢启动和拥塞避免算法的实现举例



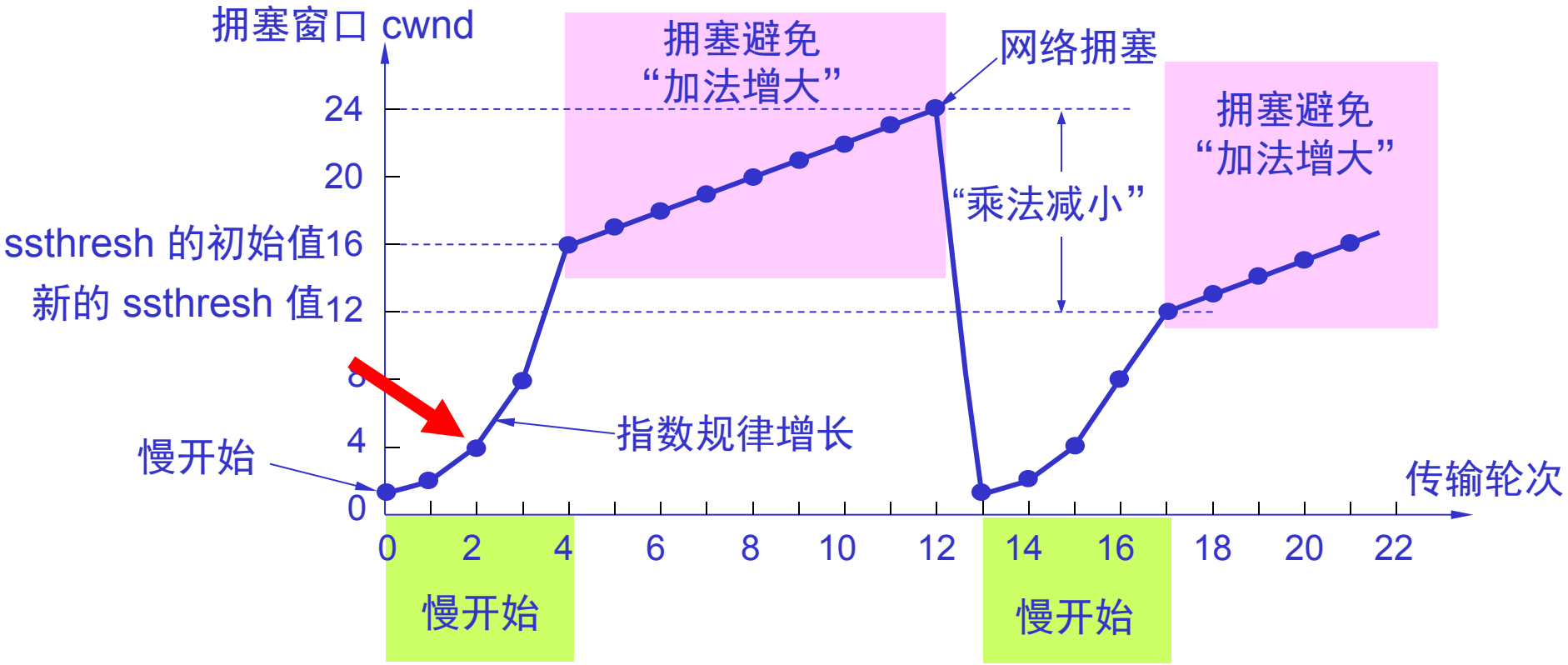
在执行慢启动算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 M_0。

慢启动和拥塞避免算法的实现举例



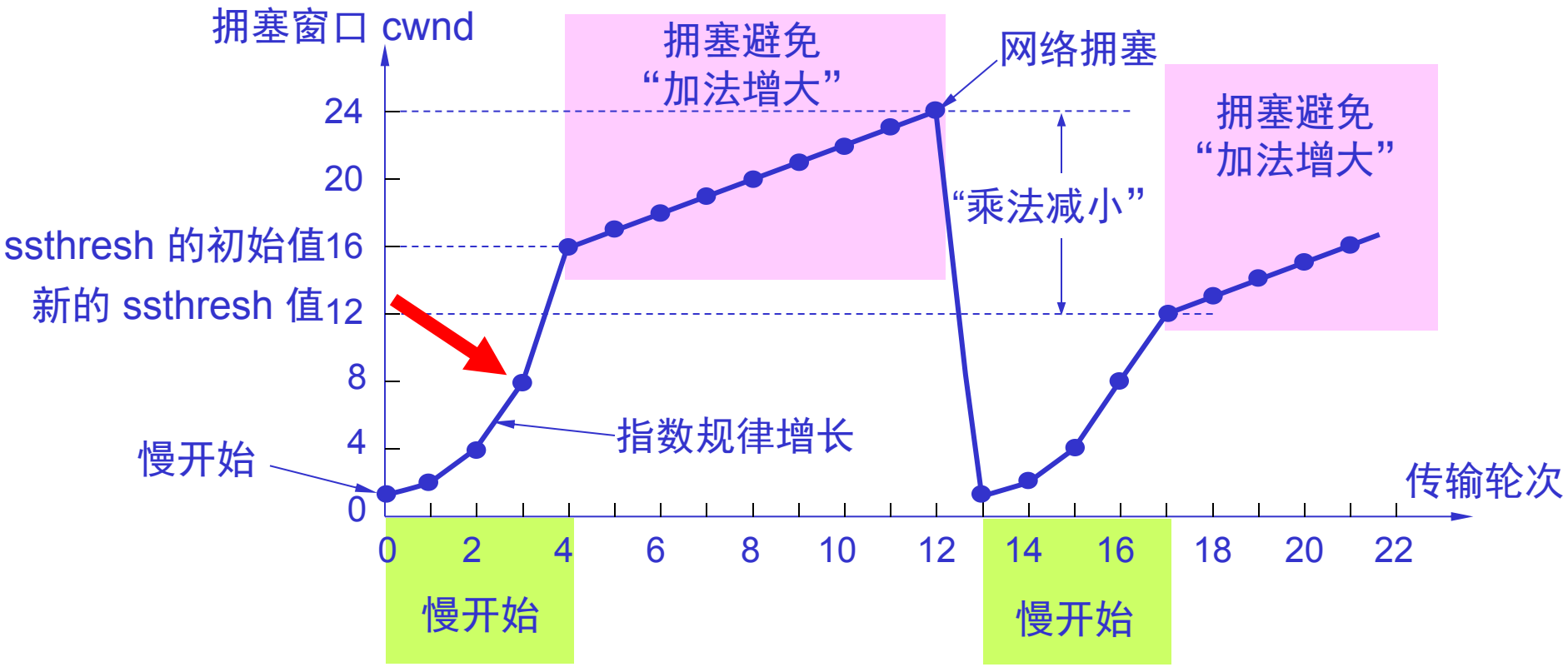
发送端每收到一个确认，就把 cwnd 加 1。于是发送端可以接着发送 M_1 和 M_2 两个报文段。

慢启动和拥塞避免算法的实现举例



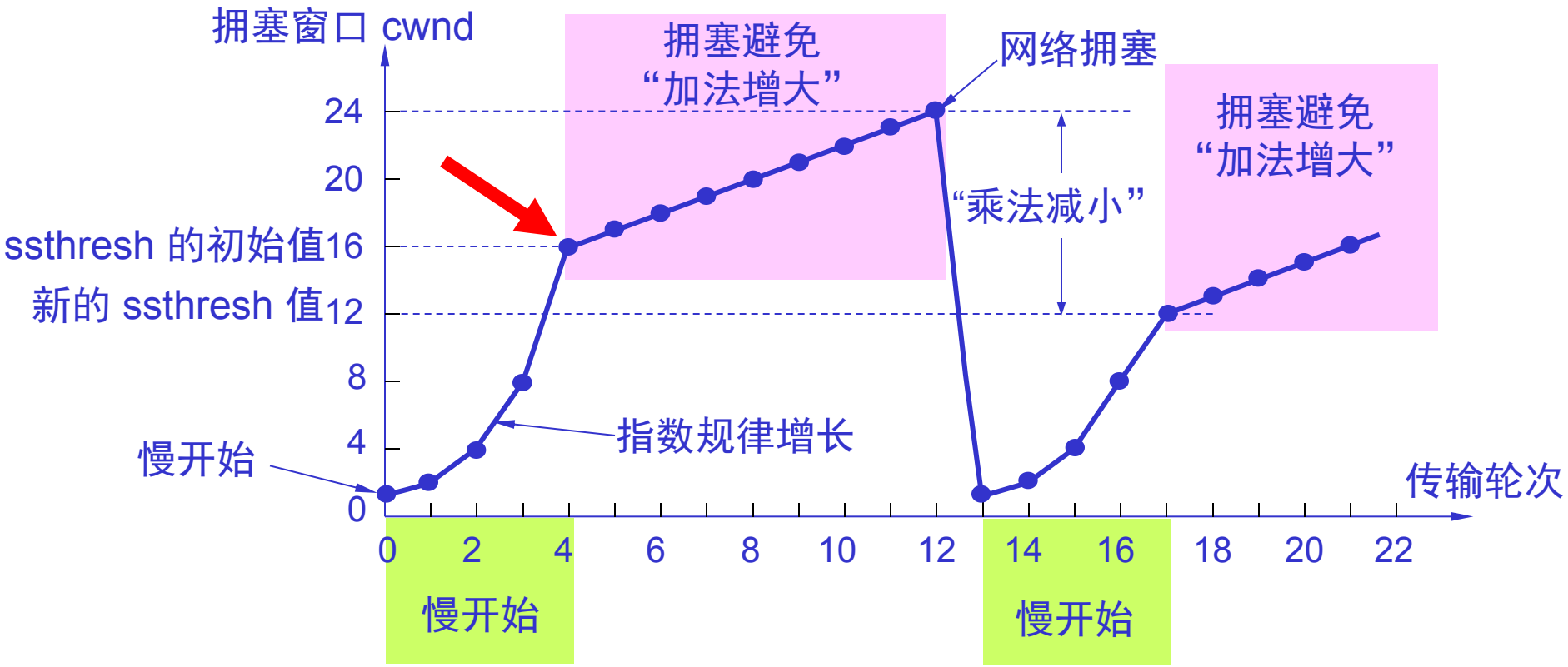
接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的 cwnd 加 1。现在 cwnd 从 2 增大到 4，并可接着发送后面的 4 个报文段。

慢启动和拥塞避免算法的实现举例



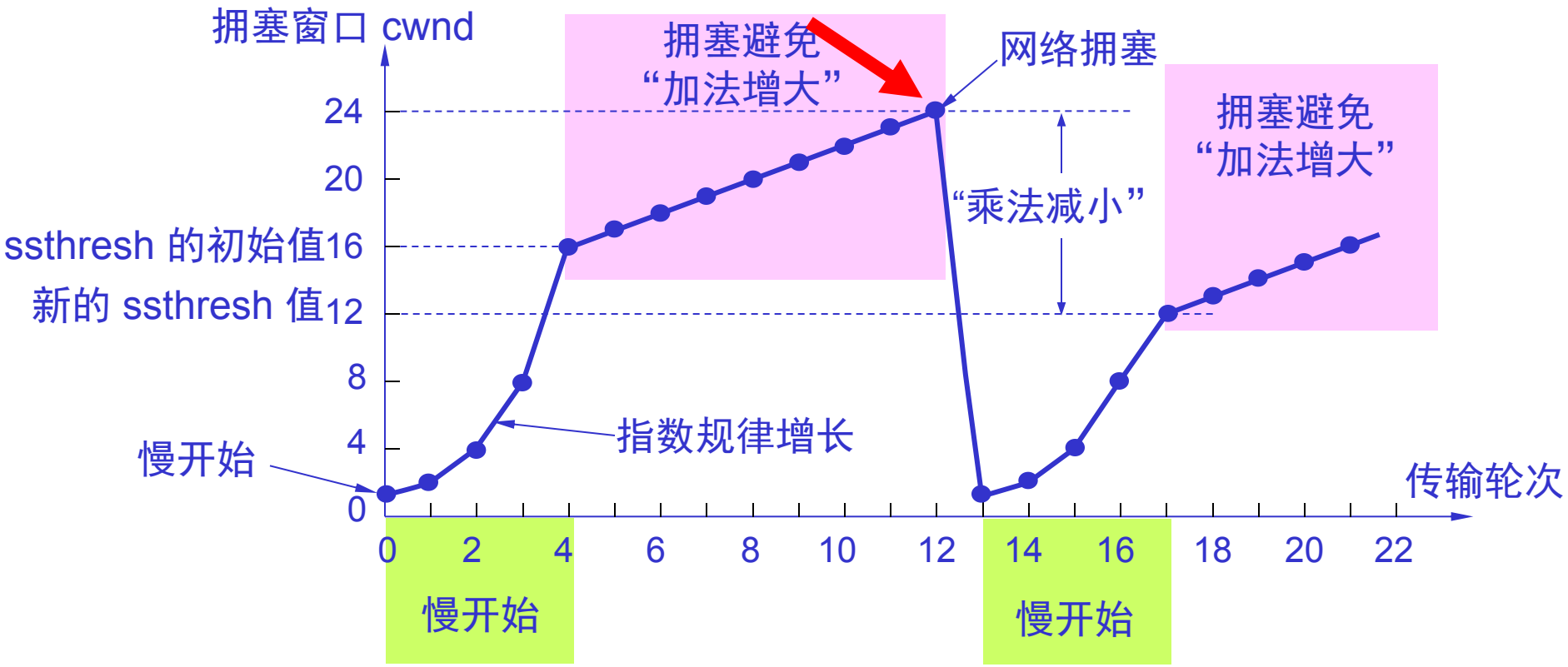
发送端每收到一个对新报文段的确认，就把发送端的拥塞窗口加 1，因此拥塞窗口 cwnd 随着传输轮次按指数规律增长。

慢启动和拥塞避免算法的实现举例



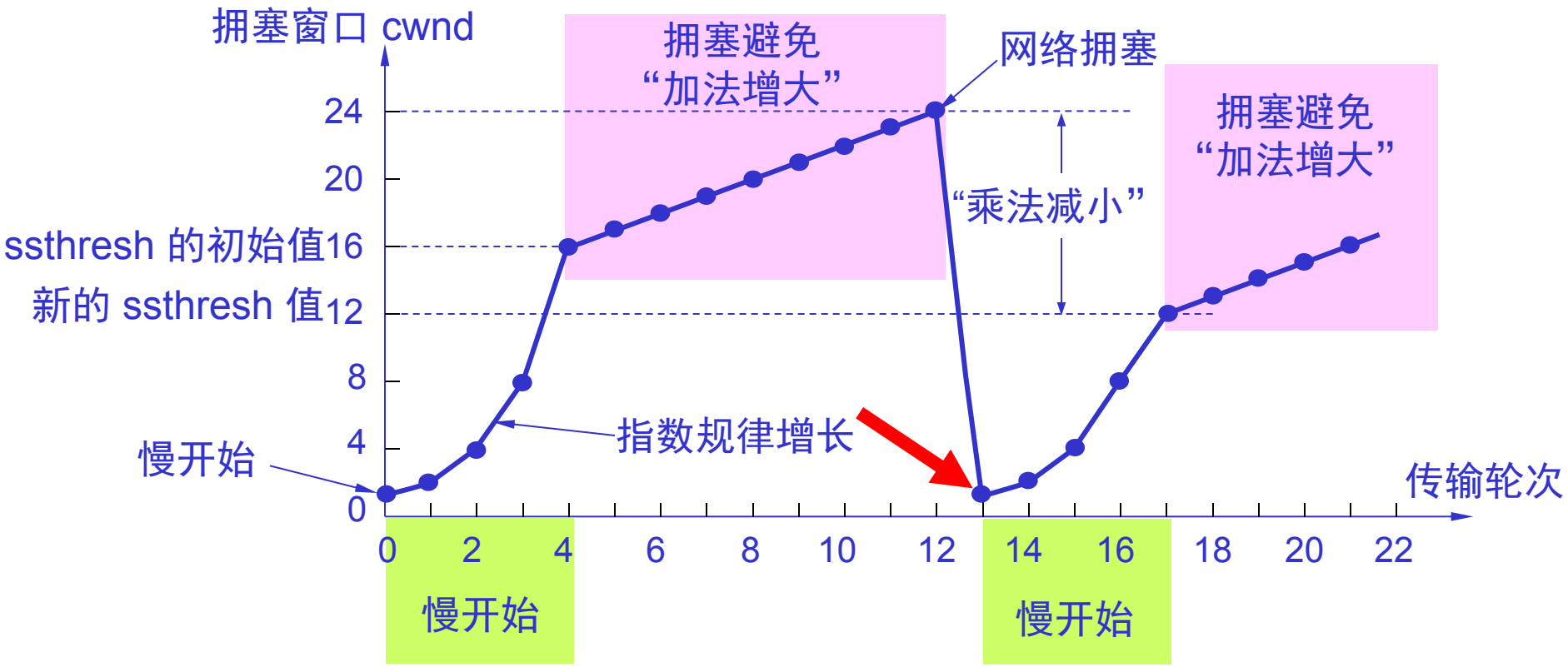
当拥塞窗口 cwnd 增长到慢启动门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

慢启动和拥塞避免算法的实现举例



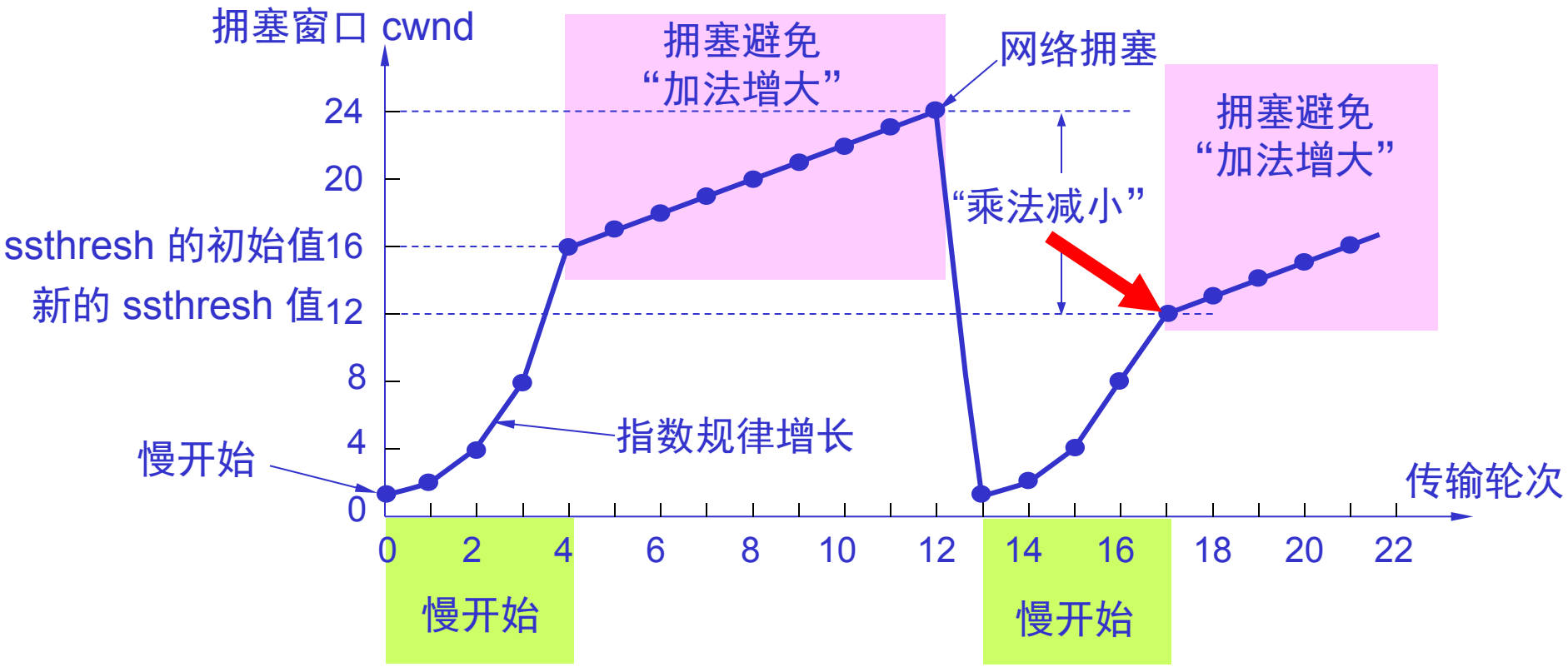
假定拥塞窗口的数值增长到 24 时，重发定时器超时，表明发生网络拥塞， ssthresh和拥塞窗口如何变化？

慢启动和拥塞避免算法的实现举例



拥塞窗口再重新设置为 1，慢启动门限值 ssthresh 值变为 12（发送拥塞时，发送窗口（拥塞窗口）数值 24 的一半），并执行慢启动算法（指数增加cwnd ）。

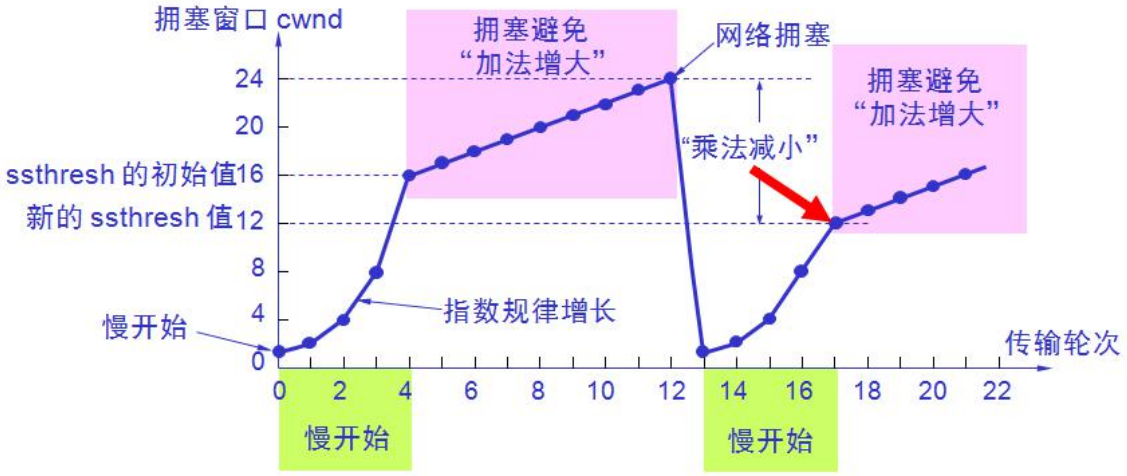
慢启动和拥塞避免算法的实现举例



当 cwnd = 12 时改为执行拥塞避免算法，拥塞窗口按按线性规律增长，每经过一个轮次传输就增加一个 MSS 的大小TCP报文段，依次类推。

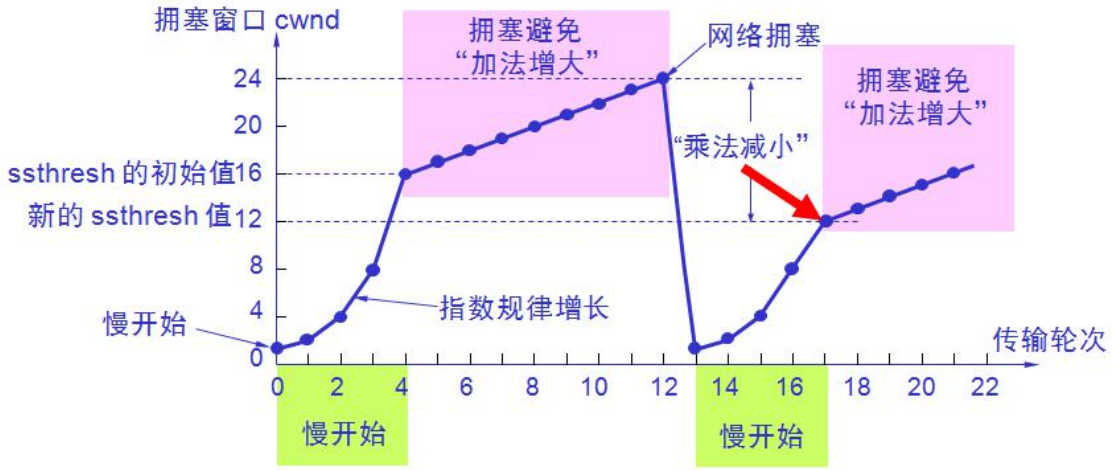
乘法减小 (MD:multiplicative decrease)

- “乘法减小”:是指不论在慢启动阶段还是拥塞避免阶段，只要出现一次重发定时器超时（即出现一次网络拥塞），就把慢启动门限值 ssthresh 设置为当前拥塞窗口值一半。
- 目的：当网络频繁出现拥塞时，ssthresh 值就下降得很快，以大大减少输入到网络中的TCP报文段数。



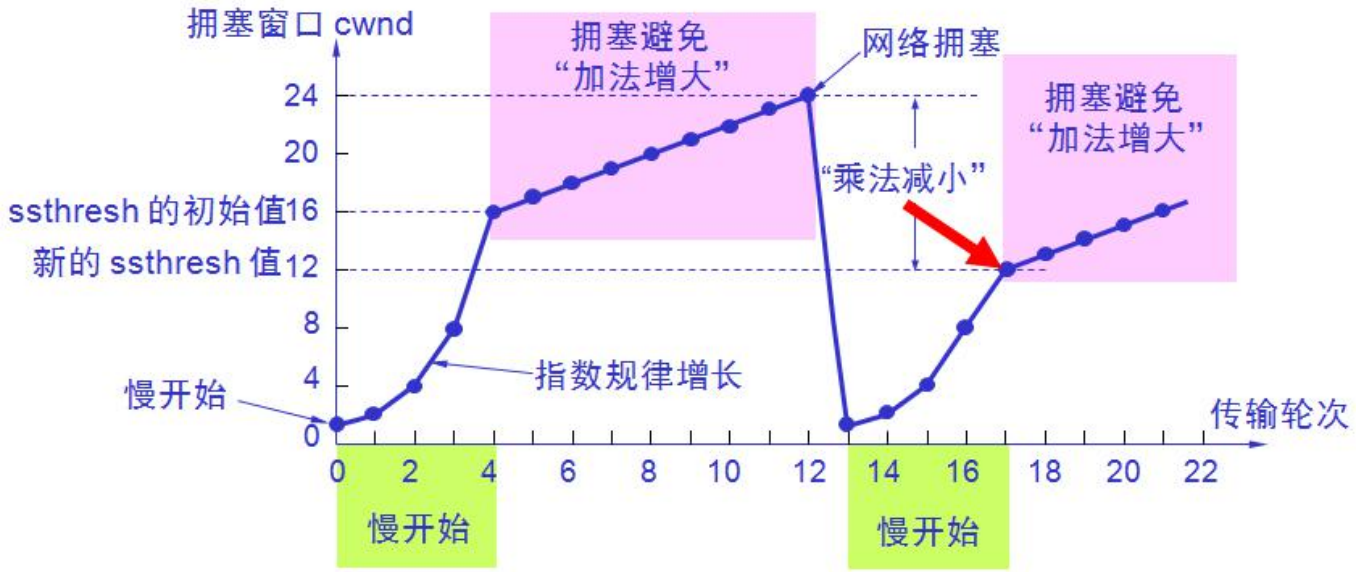
加法增大 (AI:additive increase)

- “加法增大”：指执行拥塞避免算法后，在一个传输轮次中，发送方收到对所有TCP报文段的确认后，就把拥塞窗口cwnd增加一个 MSS 大小的TCP段，使拥塞窗口缓慢增大，以防止网络过早出现拥塞。
- AIMD称为加法增加乘法减小，后来有人对AIMD进行了改进，但现在TCP协议使用最广泛的还是AIMD。



必须强调指出

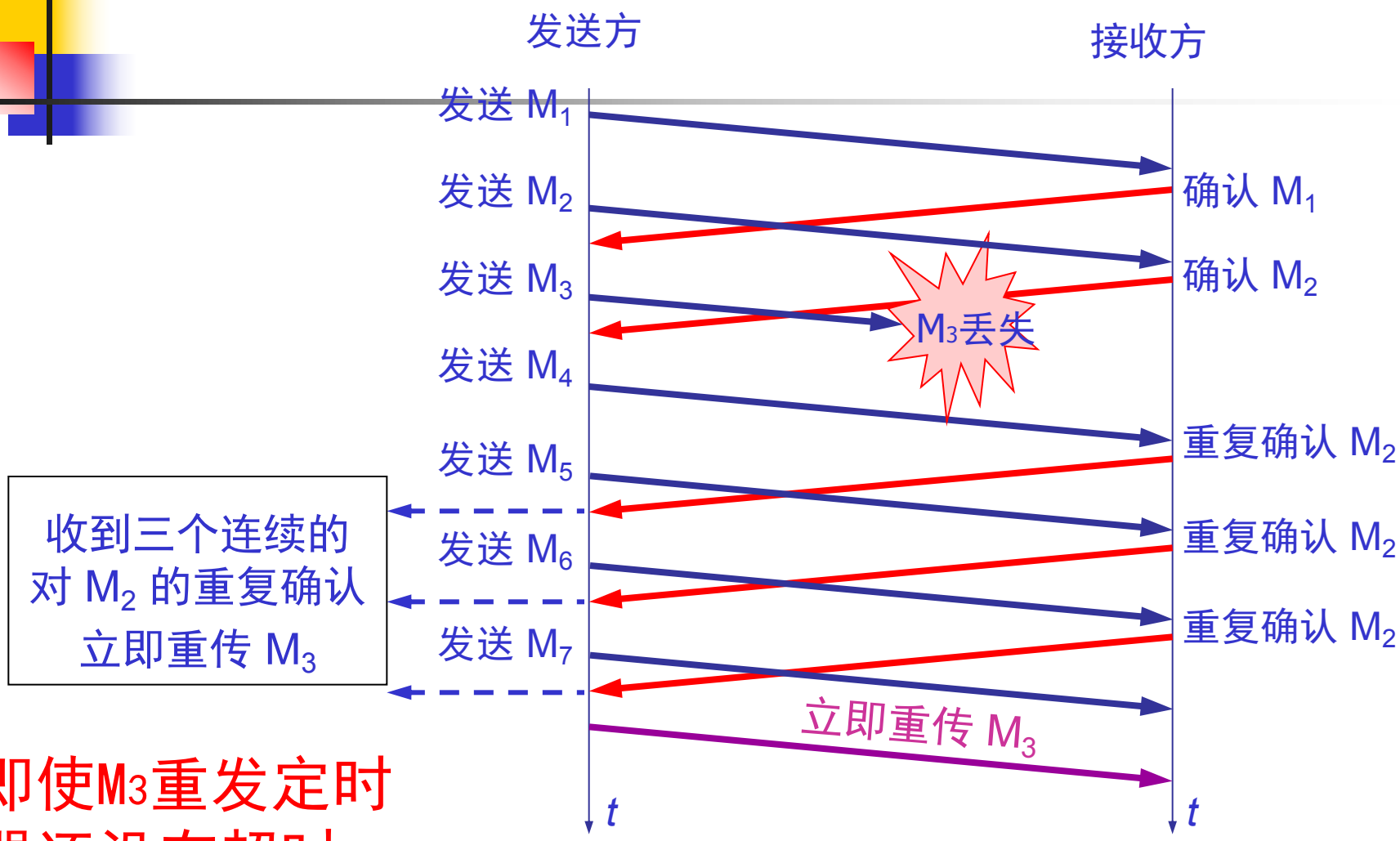
- “拥塞避免”算法并非完全能够避免了拥塞；
- “拥塞避免”是说在拥塞避免阶段把拥塞窗口控制为按线性规律增长，使网络比较不容易出现拥塞,或缓解拥塞程度。



方法2：快速重传和快速恢复

- 慢启动和拥塞避免是1988年提出的拥塞控制方法；
- 1990年又增加了两个新的拥塞控制方法：
  - 快速重传
  - 快速恢复
- 快速重传方法
  - 首先要求接收方每收到一个失序TCP报文段后就立即发出重复确认，让发送方及早知道有报文段没有到达接收方。
  - 发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，不用等到重发定时器超时。
  - 快速重传并非取消重发定时器，而是在某些情况下可更早地重传丢失（或延迟）的TCP报文段。

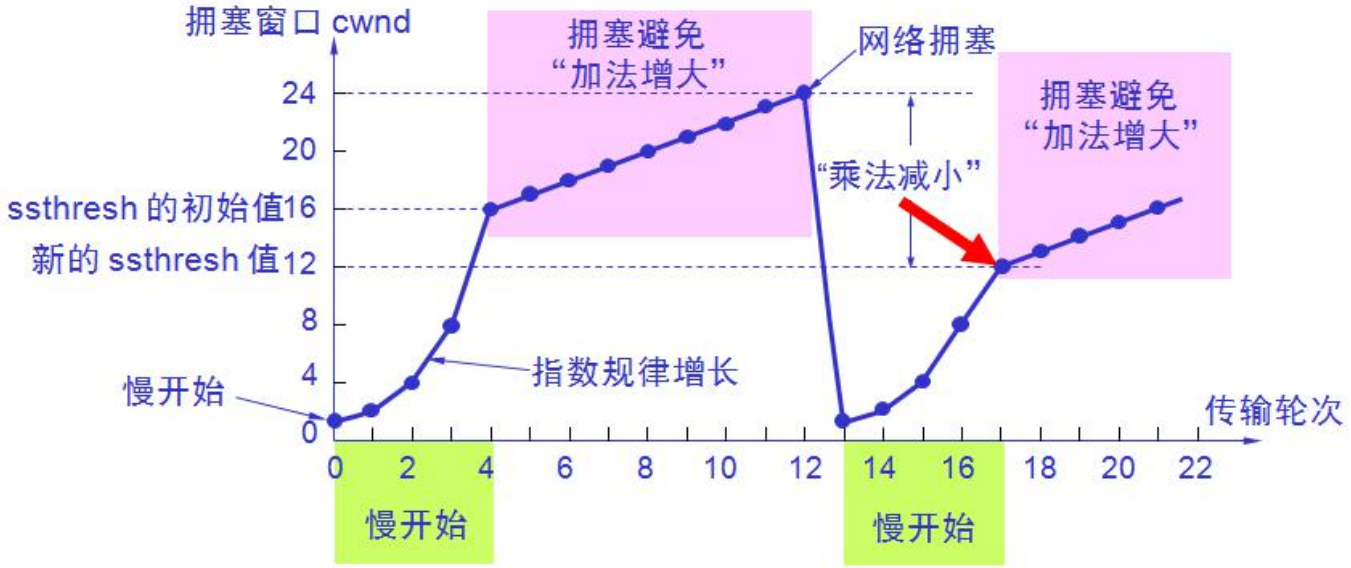
快速重传举例



即使M3重发定时器还没有超时

快速恢复算法

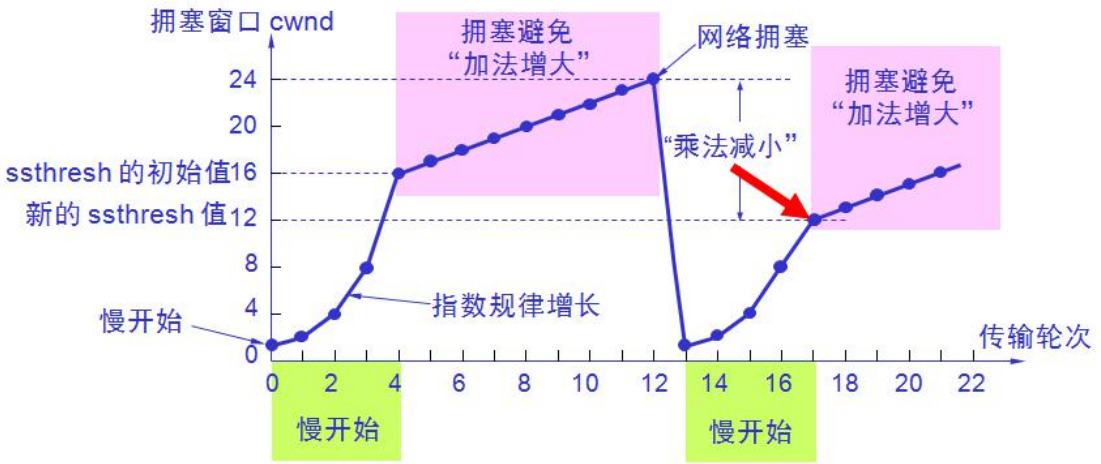
- 快速恢复方法一般与快速重传方法配合使用
  - （1）当发送方收到连续三个重复的确认时，就执行“乘法减小”方法，把慢启动门限 ssthresh 减少为当前cwnd的一半，但接下去不执行慢启动方法。



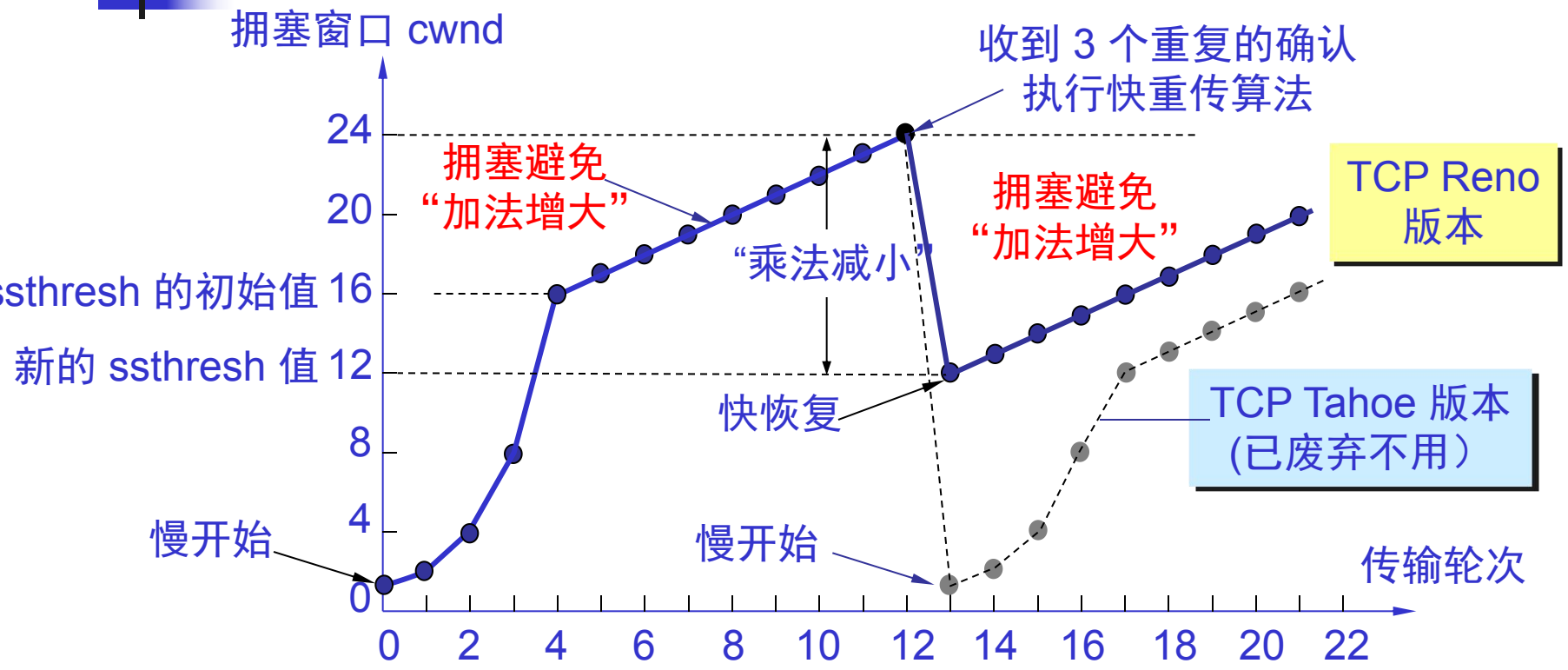
快速恢复算法

快速恢复方法一般与快速重传方法配合使用

- (2) 发送方现在认为网络很可能没有发生拥塞（发送方能收到三个重复确认），因此不执行慢启动算法，拥塞窗口 cwnd 现在不设置为 1，而是将慢启动门限 ssthresh 减少为当前cwnd的一半，同时将拥塞窗口设置为新的慢启动门限 ssthresh 的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。



从连续收到三个重复的确认转入拥塞避免



发送窗口的上限值确定

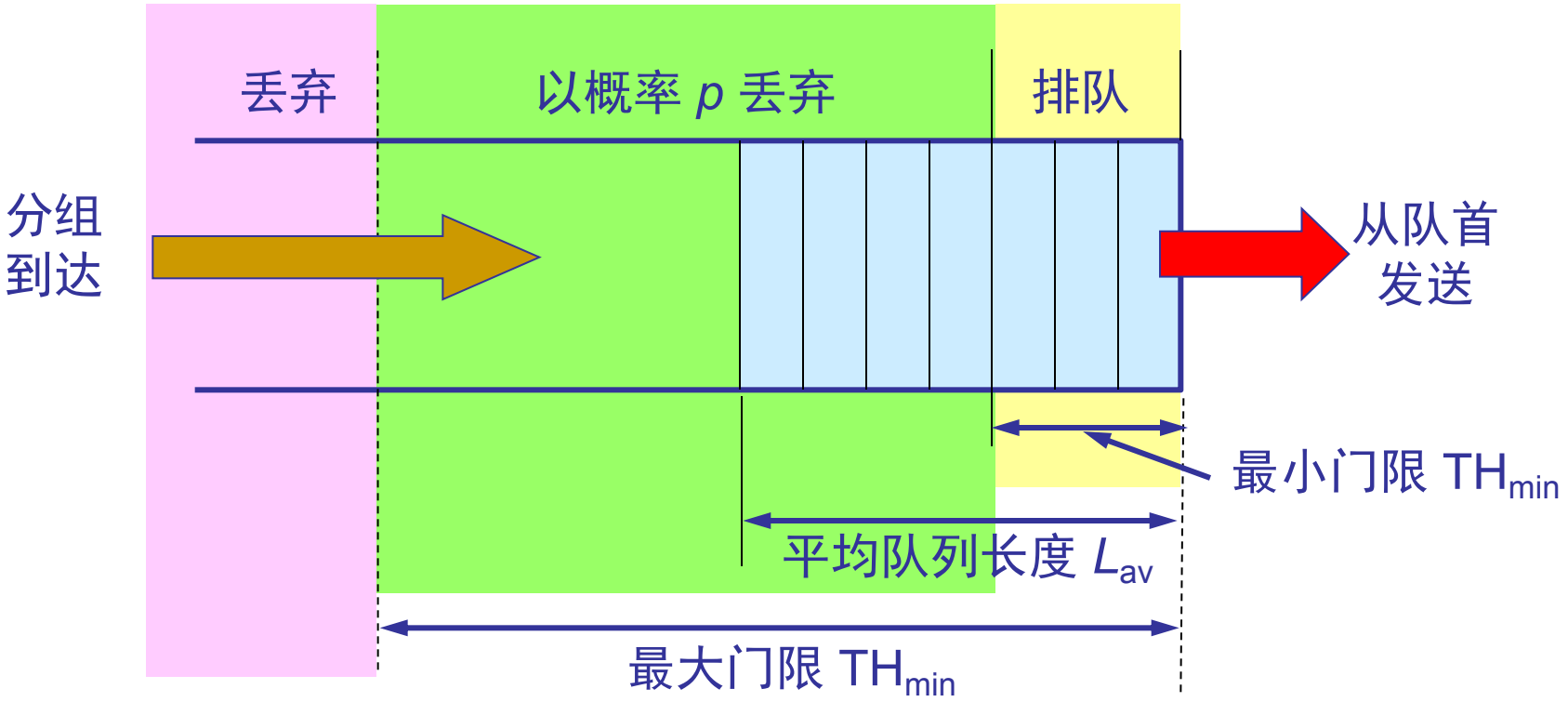
- 本节开始假设：接收方接收缓存有足够大空间，目的是不用考虑流量控制。
- 当实际工程中，由于接收方接收缓存空间有限，所以在考虑拥塞控制同时，也要考虑流量控制。
- 发送窗口大小同时受拥塞窗口cwnd 和接收窗口有效大小rwnd限制，应按以下公式确定：

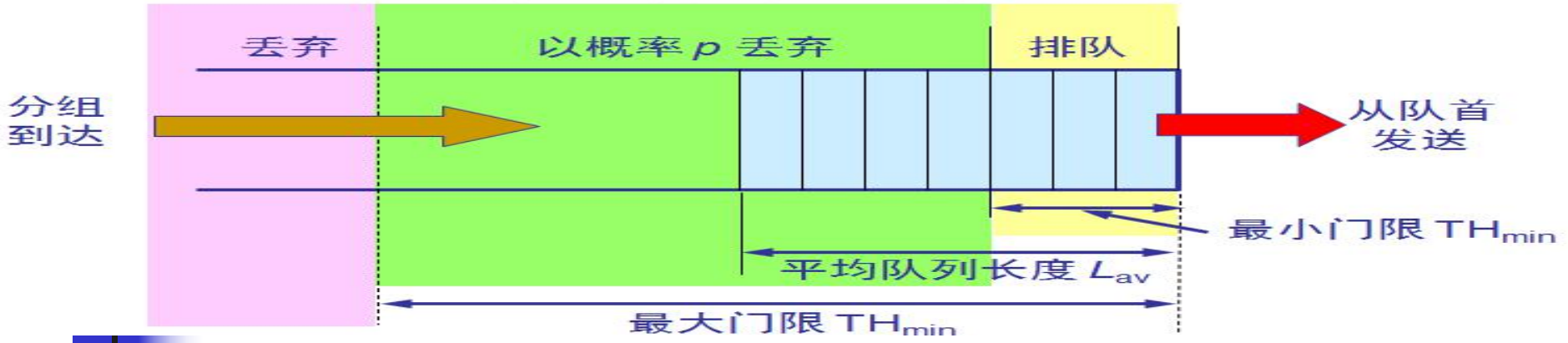
发送窗口的上限值 = Min [rwnd, cwnd]

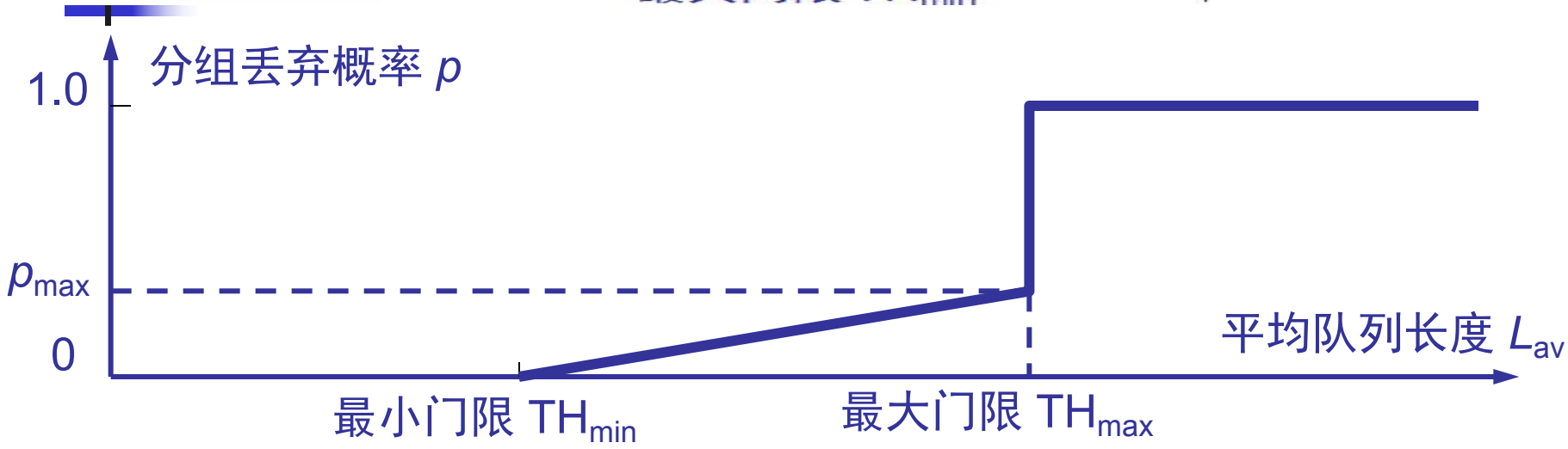
- 当 rwnd < cwnd 时，由接收窗口控制发送窗口的最大值（流量控制起主要作用）。
- 当 rwnd > cwnd 时，则拥塞窗口控制发送窗口的最大值（拥塞控制起主要作用）。

6.9.3 随机早期检测 RED (Random Early Detection)

RED 将路由器的接收缓存队列划分成为三个区域

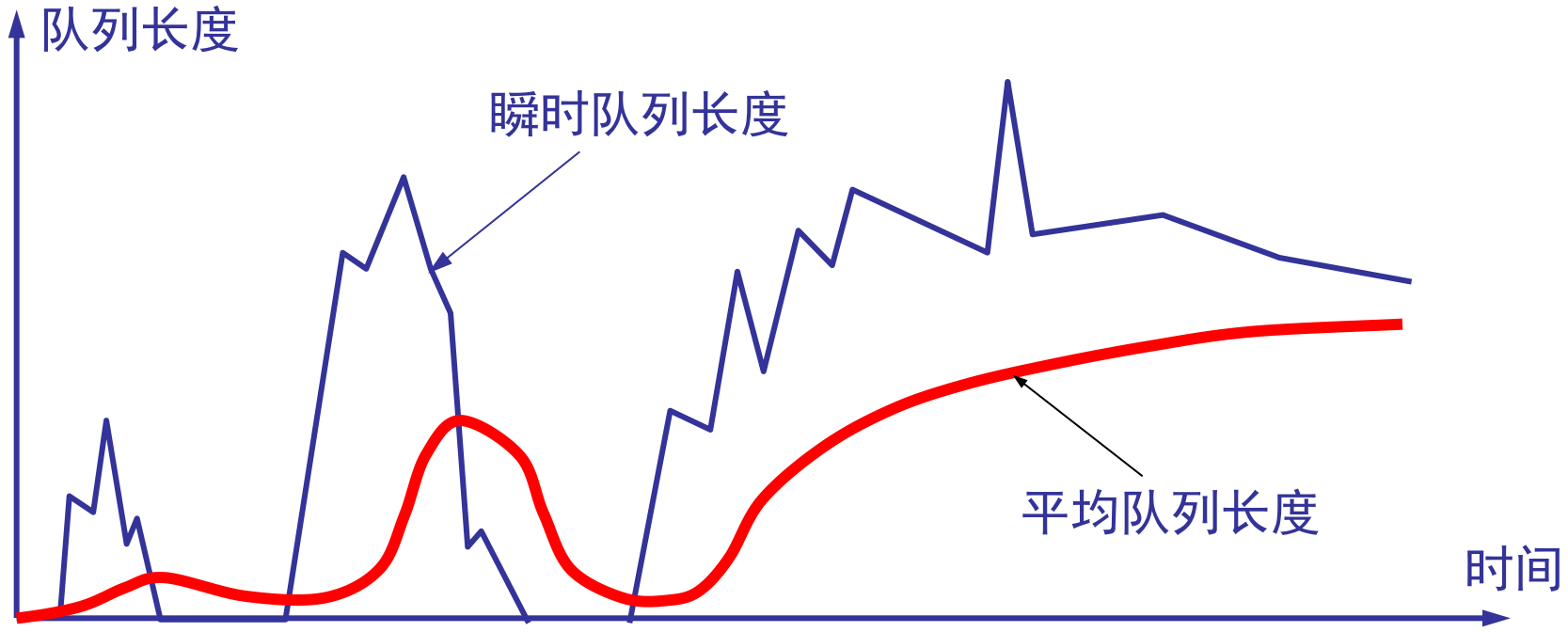






- 当 $L_{AV} < Th_{min}$ 时，丢弃概率 $p = 0$。
- 当 $L_{AV} > Th_{max}$ 时，丢弃概率 $p = 1$。
- 当 $TH_{min} < L_{AV} < TH_{max}$ 时，$0 < p < 1$。
  例如，按线性规律变化，从 0 变到 p_{max}。

瞬时队列长度和平均队列长度的区别



小结

6.8 TCP 的运输连接管理

6.8.1 TCP 的连接建立

6.8.2 TCP 的连接释放

6.8.3 TCP 的有限状态机

6.9 TCP 的拥塞控制

6.9.1 拥塞控制的一般原理

6.9.2 几种拥塞控制方法