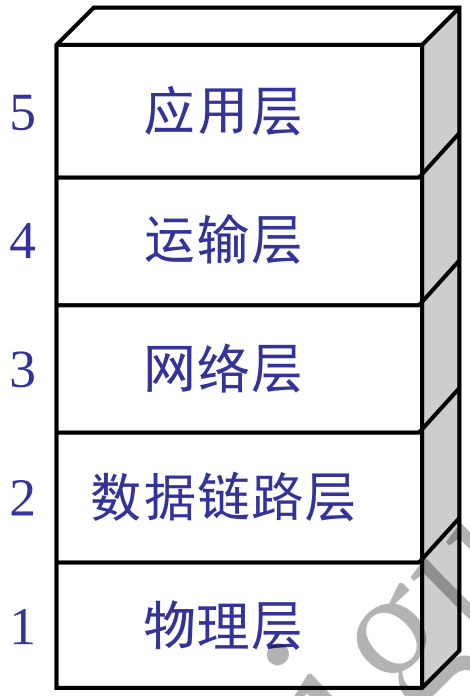
计算机网络

顾 军

计算机学院

jgu@cumt.edu.cn

专题5：如何保证端到端的可靠传输



- 应用层(application layer)
- 运输层(transport layer)
- 网络层(network layer)
- 数据链路层(data link layer)
- 物理层(physical layer)

Q29：慢开始和拥塞避免算法？

- 慢开始(slow start)/慢启动算法和拥塞避免(congestion avoidance)算法是两个目的不同、独立的算法，在实际中这两个算法通常在一起实现。
- 当拥塞发生时，希望降低分组进入网络的传输速率，于是可以调用慢开始/慢启动来做到这一点。
- 当主机开始发送数据时，由于并不清楚网络的负荷情况，所以如果立即把大量数据字节注入到网络，那么就有可能引起网络发生拥塞。
- 慢开始算法的思路：以探测的方式由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大拥塞窗口数值。

初始拥塞窗口 cwnd 设置

- 旧的规定：在刚刚开始发送报文段时，先把初始拥塞窗口cwnd 设置为 1 至 2 个发送方的最大报文段 SMSS (Sender Maximum Segment Size) 的数值（字节大小）。
- 新的 RFC 5681 把初始拥塞窗口 cwnd 设置为不超过2至4个SMSS 的数值。

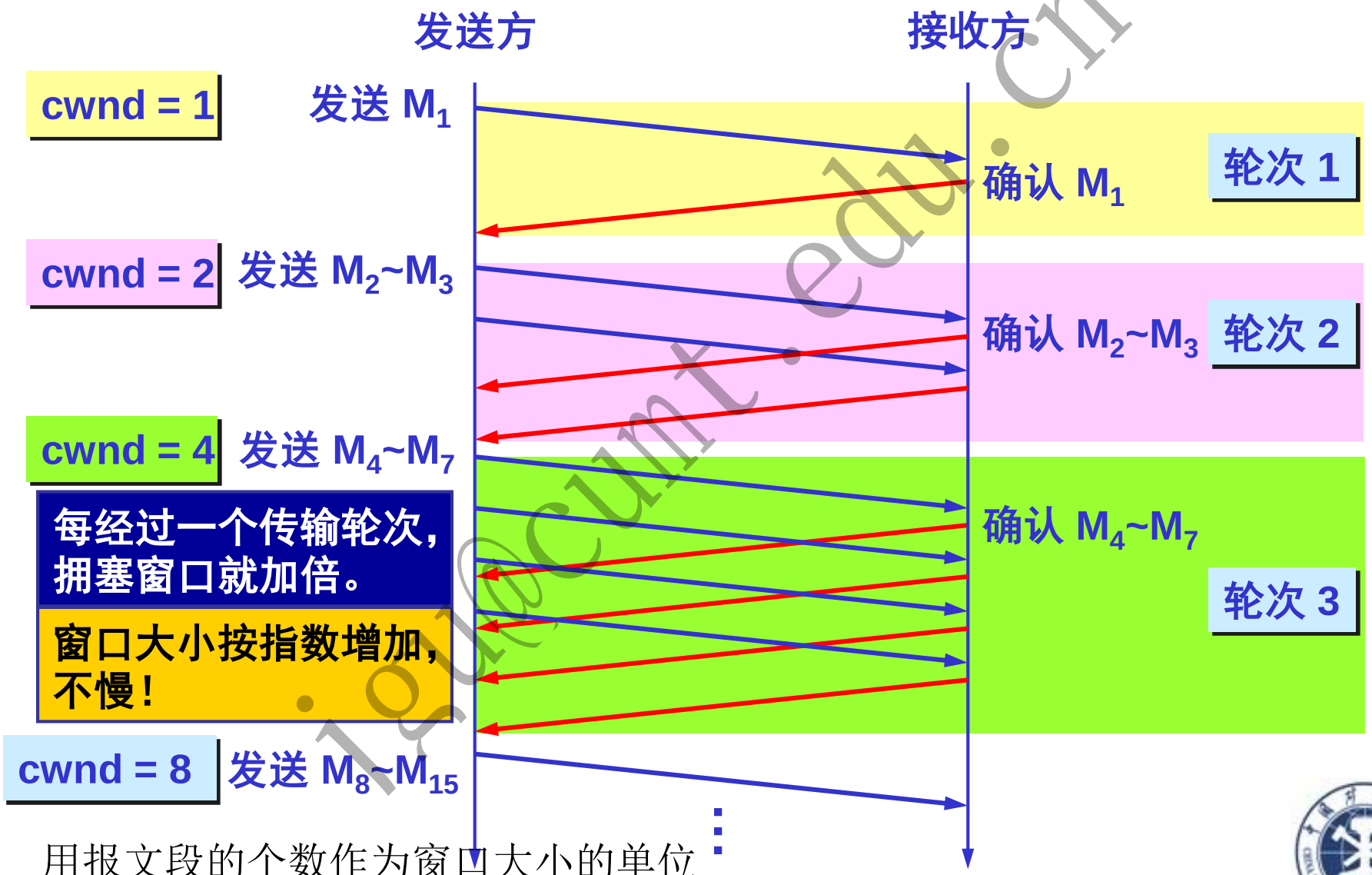
慢开始的拥塞窗口控制方法

- 拥塞窗口 cwnd 控制方法：在每收到一个对新的报文段的确认后，可以把拥塞窗口增加最多一个 SMSS 的数值。

拥塞窗口cwnd每次的增加量 = min (*N*, SMSS)

- 其中，*N* 是原先未被确认的、但现在被刚收到的确认报文段所确认的字节数。
- 当 *N* < SMSS 时，拥塞窗口每次的增加量要小于 SMSS。
- 用这样的方法逐步增大发送方的拥塞窗口 cwnd，可以使分组注入到网络的速率更加合理。

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 加倍。



传输轮次

- 使用慢开始算法后，每经过一个传输轮次(transmission round)，拥塞窗口 cwnd 就加倍。
- 一个传输轮次所经历的时间其实就是往返时间 RTT（注意：RTT并非是恒定的数值）。
- 使用“传输轮次”是更加强调：把拥塞窗口 cwnd 所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。
  - 例如，拥塞窗口 cwnd = 4，这时的往返时间 RTT 就是发送方连续发送 4 个报文段，并收到这 4 个报文段的确认，总共经历的时间。

正确理解慢开始的“慢”

- 慢开始的“慢”并不是cwnd的增长速率慢，而是指在TCP开始发送报文段时先设置cwnd=1，使得发送方在开始时只发送一个报文段，然后再逐渐增大cwnd。
- 这比设置大的cwnd值一下子把许多报文段注入到网络中要“慢得多”。

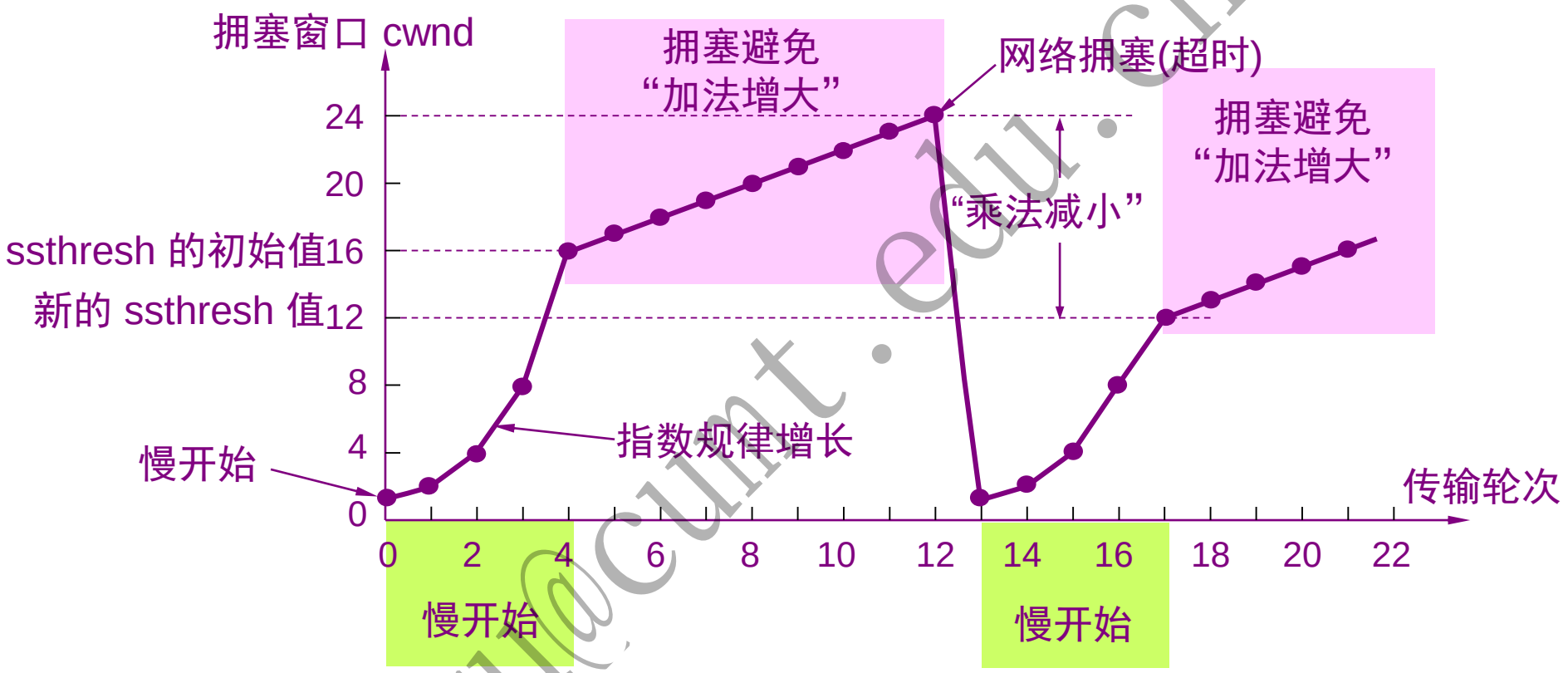
设置慢开始门限状态变量 ssthresh

- 慢开始门限 ssthresh（状态变量）：防止拥塞窗口cwnd 增长过大引起网络拥塞。
- 慢开始门限 ssthresh 的用法如下：
  - 当 cwnd < ssthresh 时，使用慢开始算法。
  - 当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。
  - 当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。

拥塞避免(congestion avoidance)算法

- 拥塞避免算法的思路：让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd 加 1，而不是加倍，使拥塞窗口 cwnd 按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。
- 可以看出，在拥塞避免阶段，拥塞窗口是按照线性规律增大的。这常称为“加法增大”AI (Additive Increase)。

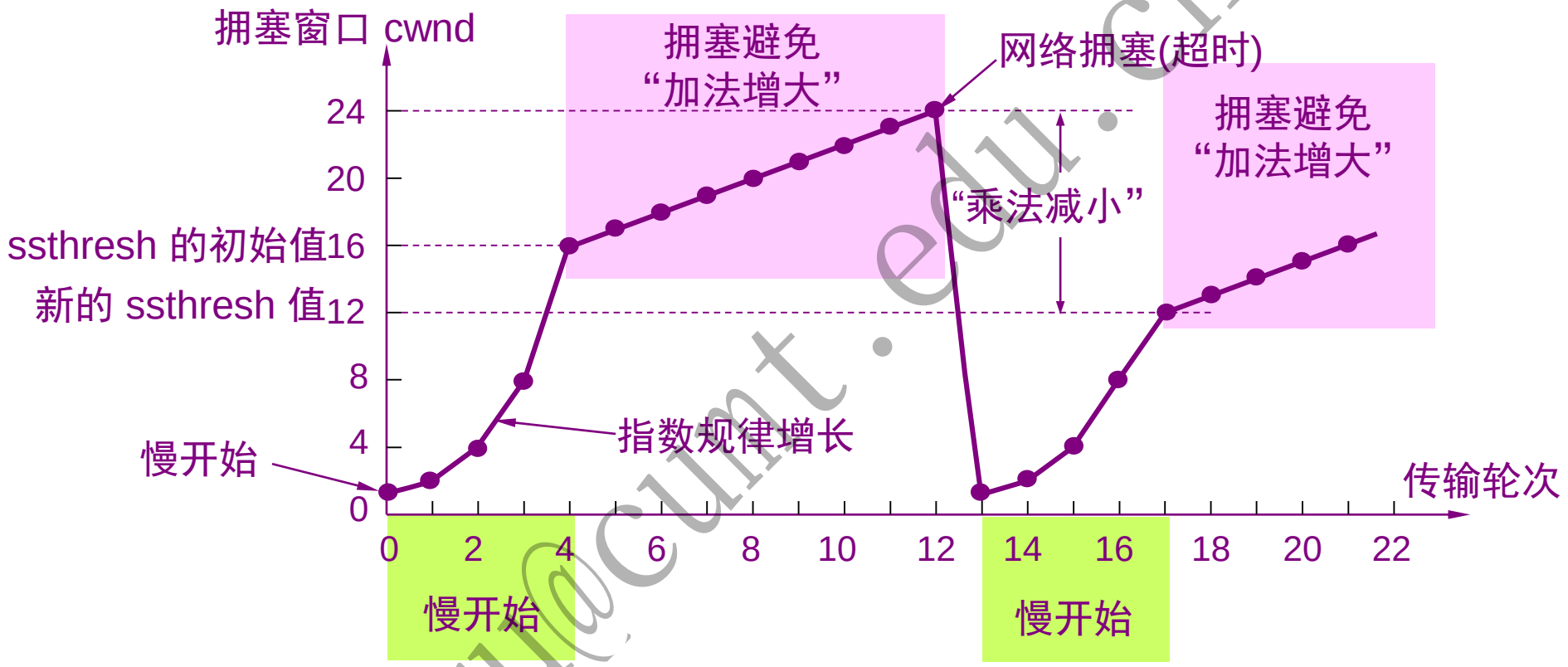
慢开始和拥塞避免算法的实现举例



当 TCP 连接进行初始化时，将拥塞窗口置为 1。为了便于理解，图中的窗口单位不使用字节而使用报文段。

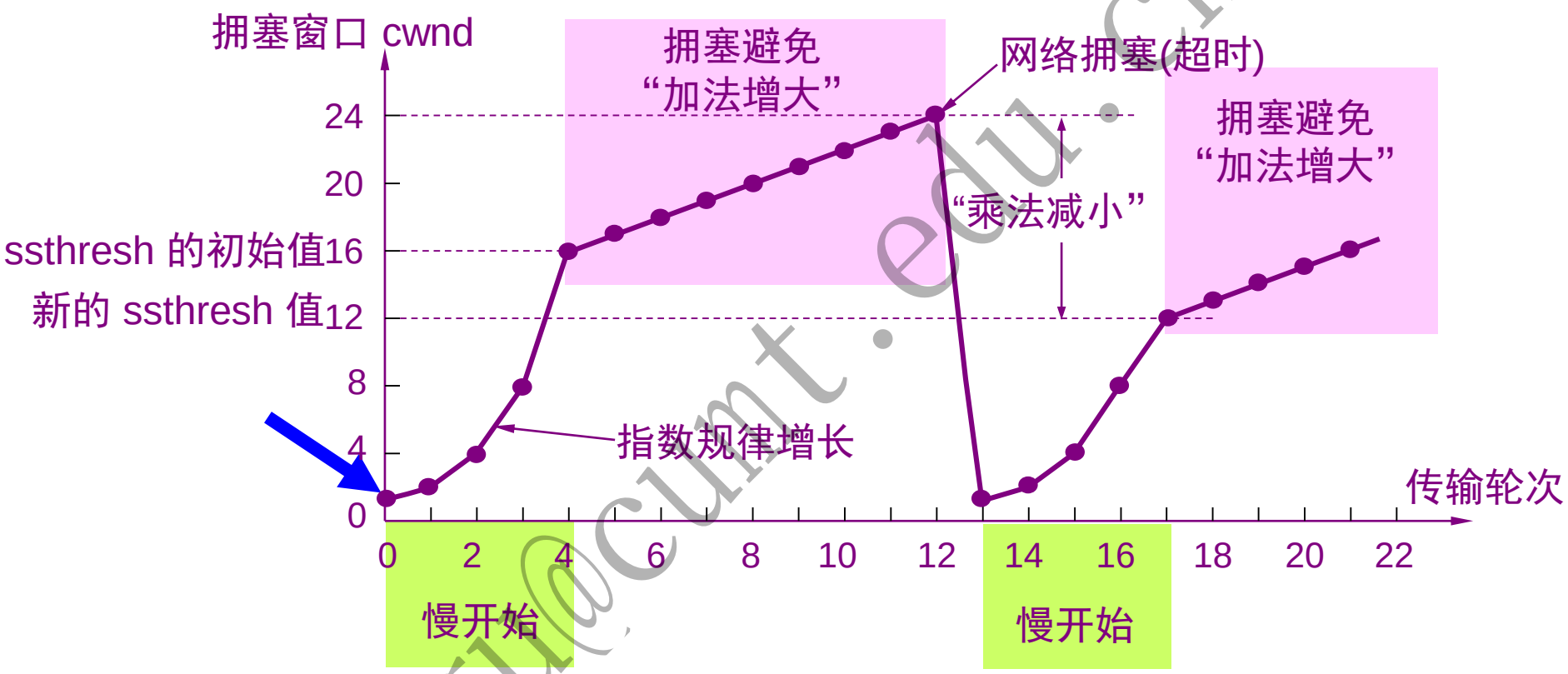
慢开始门限的初始值设置为 16 个报文段，即 ssthresh = 16。

慢开始和拥塞避免算法的实现举例



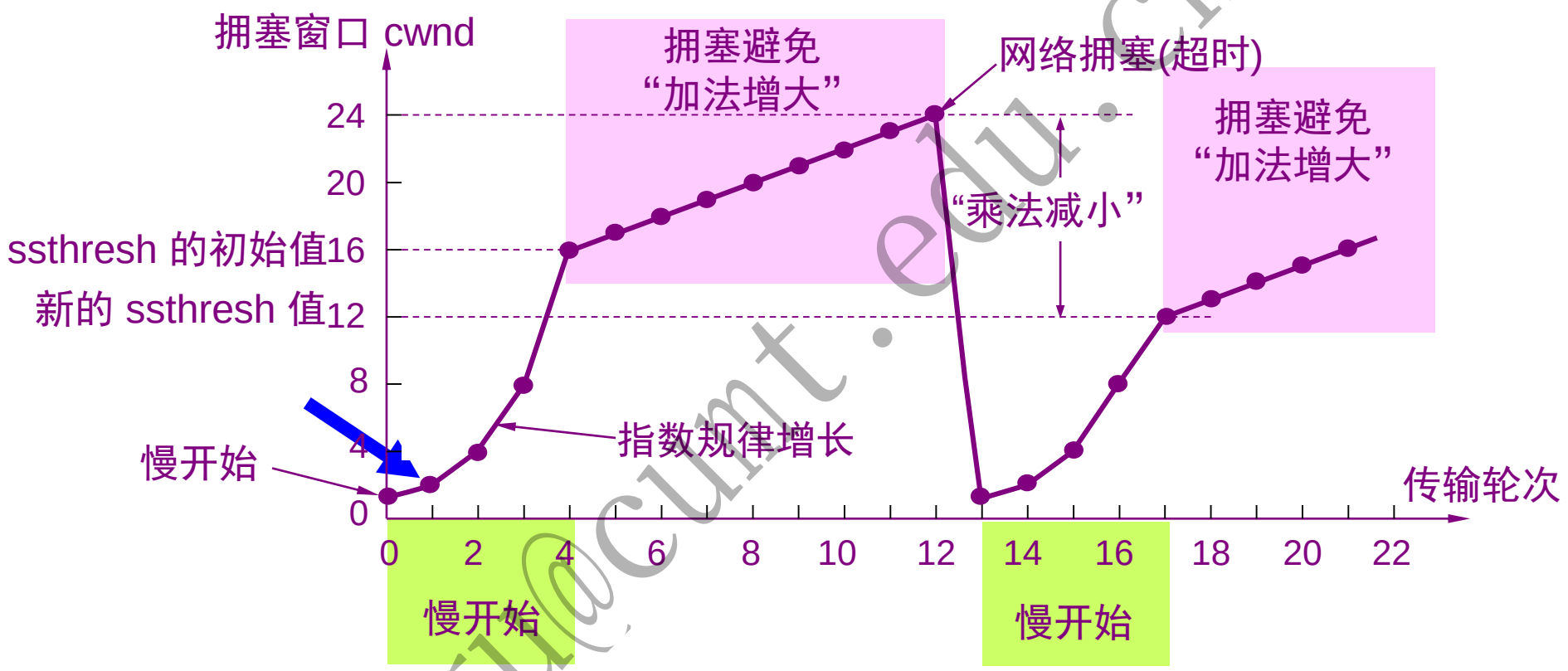
发送端的发送窗口不能超过拥塞窗口 cwnd 和接收端窗口 rwnd 中的最小值。我们假定接收端窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值。

慢开始和拥塞避免算法的实现举例



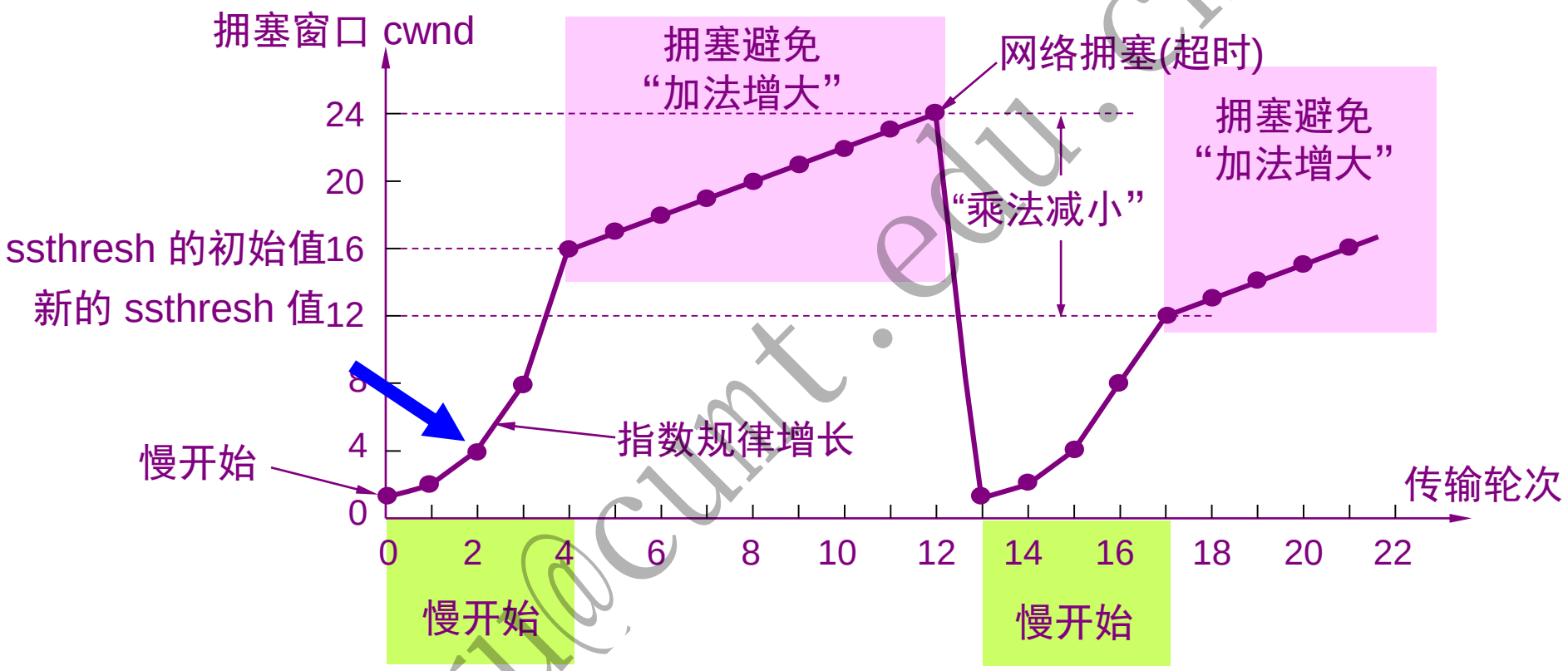
在执行慢开始算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 M_0。

慢开始和拥塞避免算法的实现举例



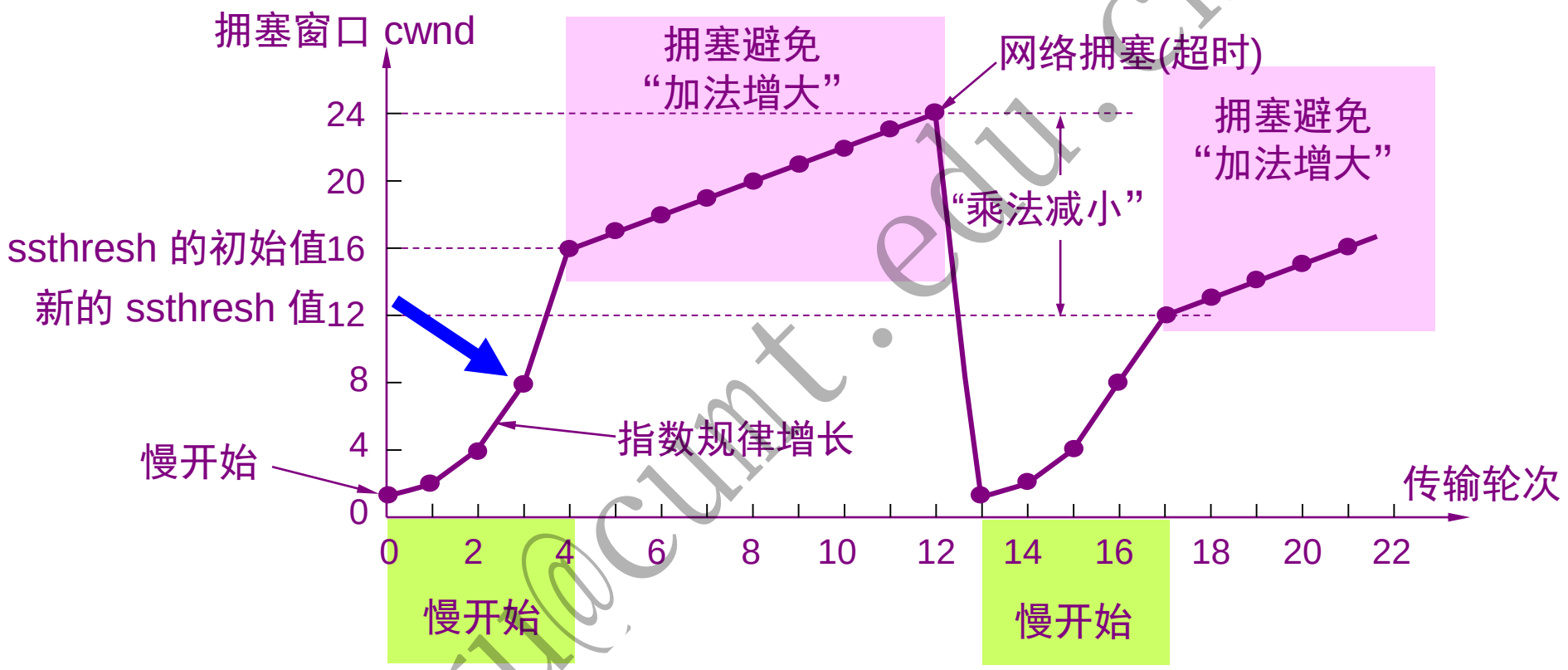
发送端每收到一个确认，就把 cwnd 加倍。于是发送端可以接着发送 M_1 和 M_2 两个报文段。

慢开始和拥塞避免算法的实现举例



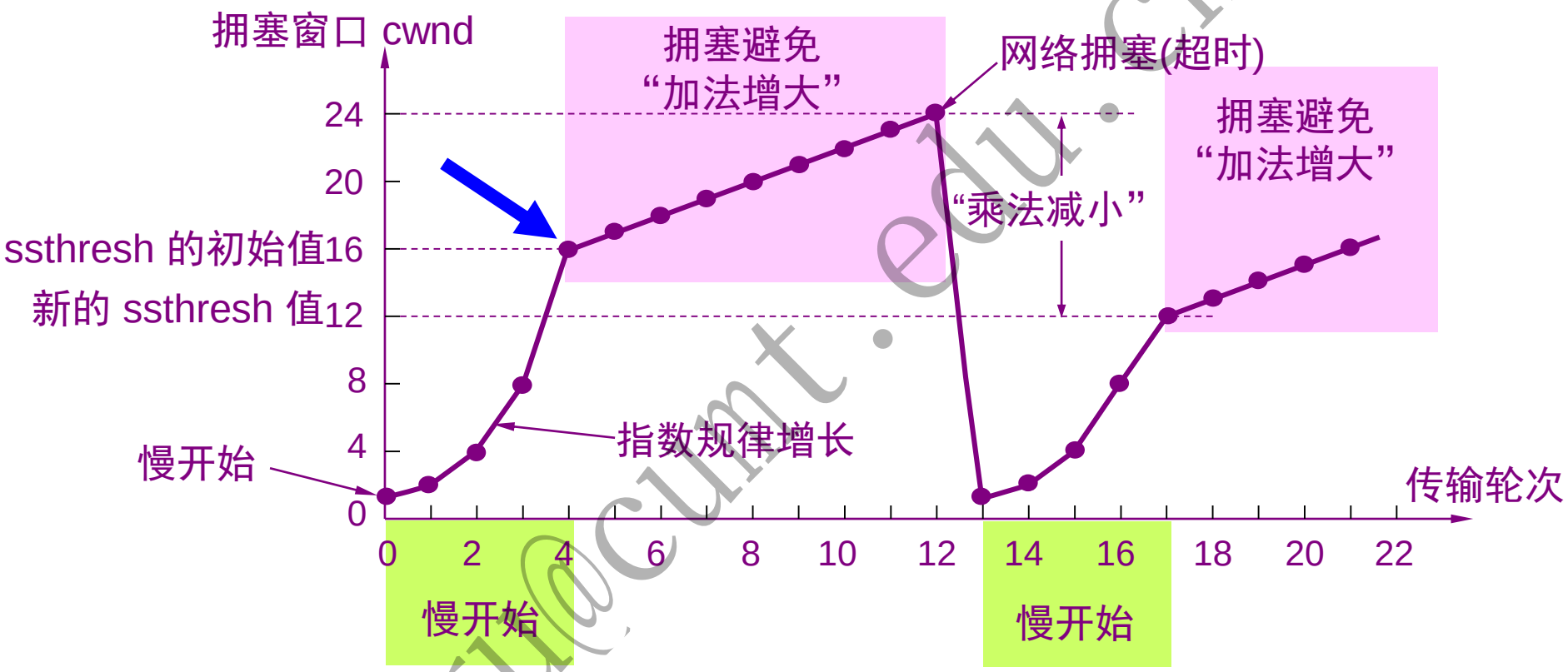
接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的 cwnd 加倍。现在 cwnd 从 2 增大到 4，并可接着发送后面的 4 个报文段。

慢开始和拥塞避免算法的实现举例



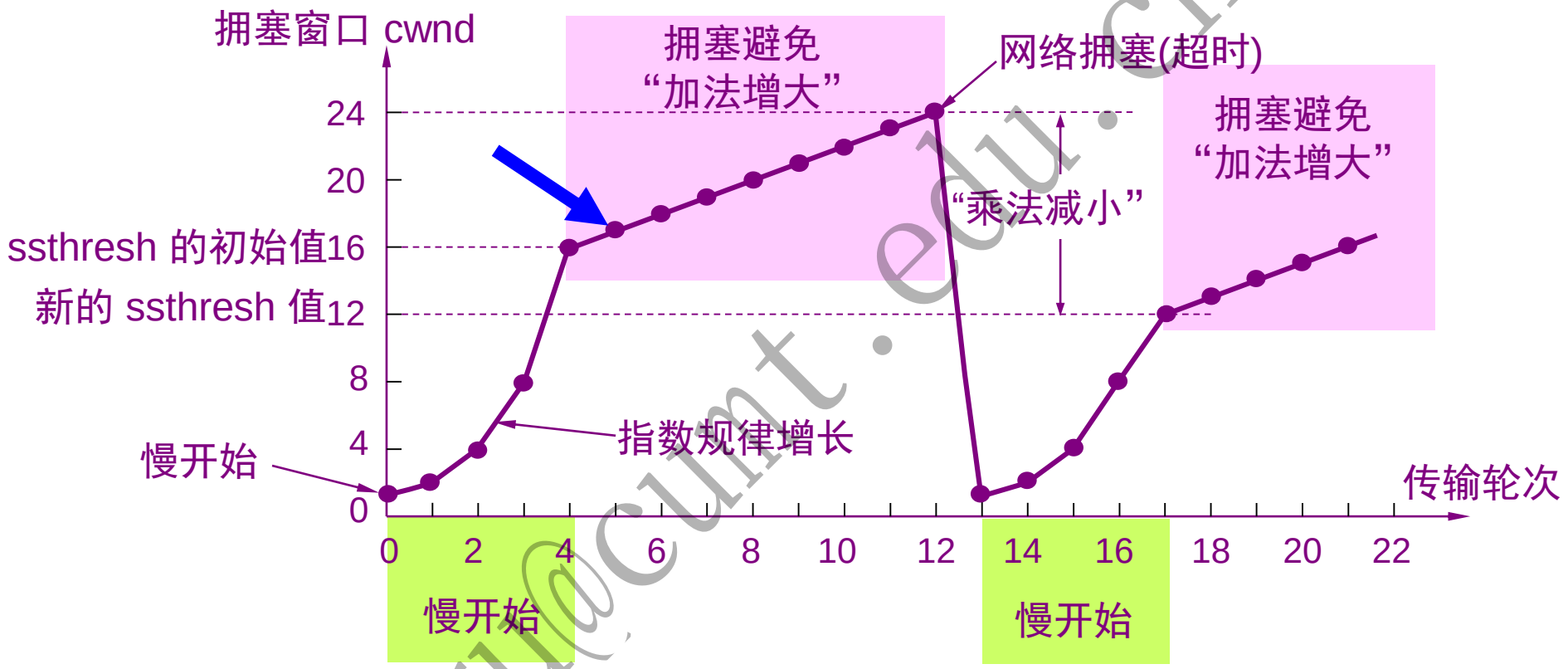
发送端每收到一个对新报文段的确认，就把发送端的拥塞窗口加倍，因此拥塞窗口 cwnd 随着传输轮次按指数规律增长。

慢开始和拥塞避免算法的实现举例



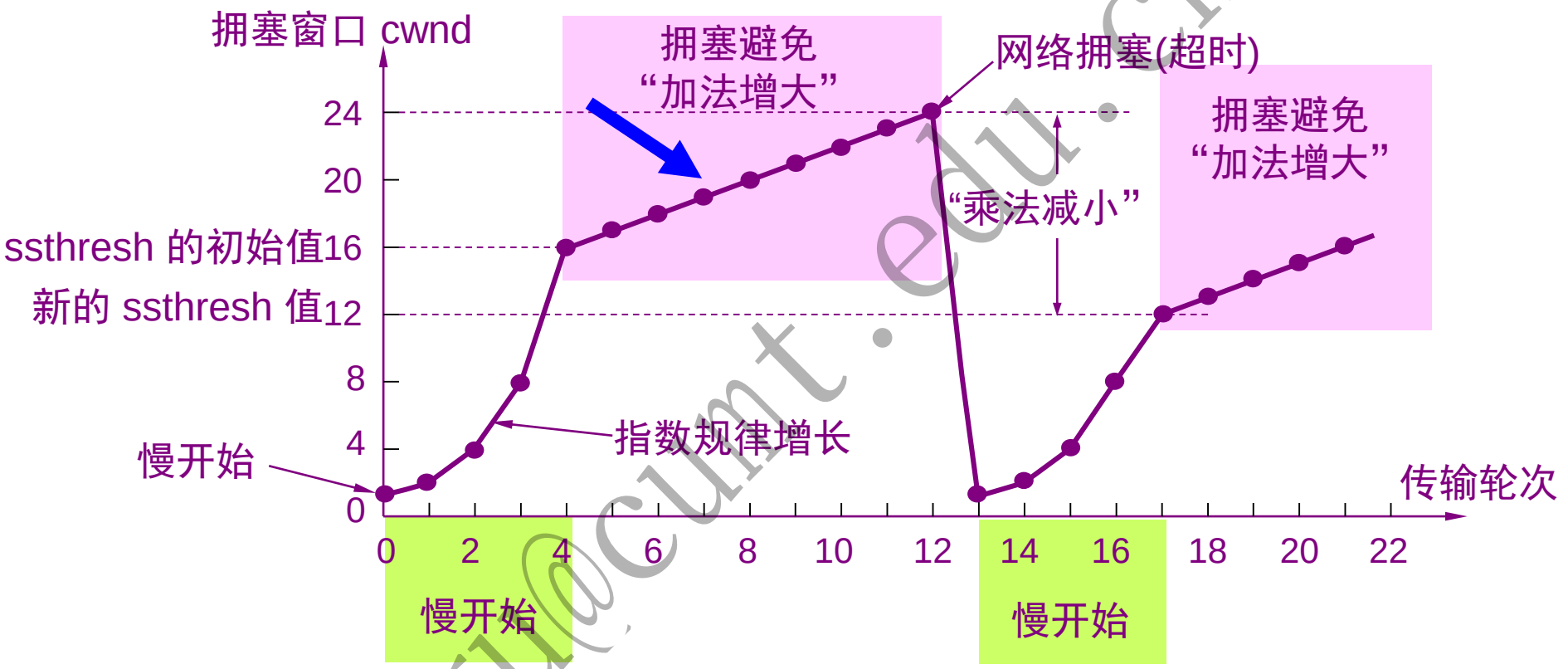
当拥塞窗口 cwnd 增长到慢开始门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

慢开始和拥塞避免算法的实现举例



“加法增大(additive increase)”是指执行拥塞避免算法后，当收到对所有报文段的确认就将拥塞窗口cwnd增加一个 MSS 大小，使拥塞窗口缓慢增大，以防止网络过早出现拥塞。

慢开始和拥塞避免算法的实现举例

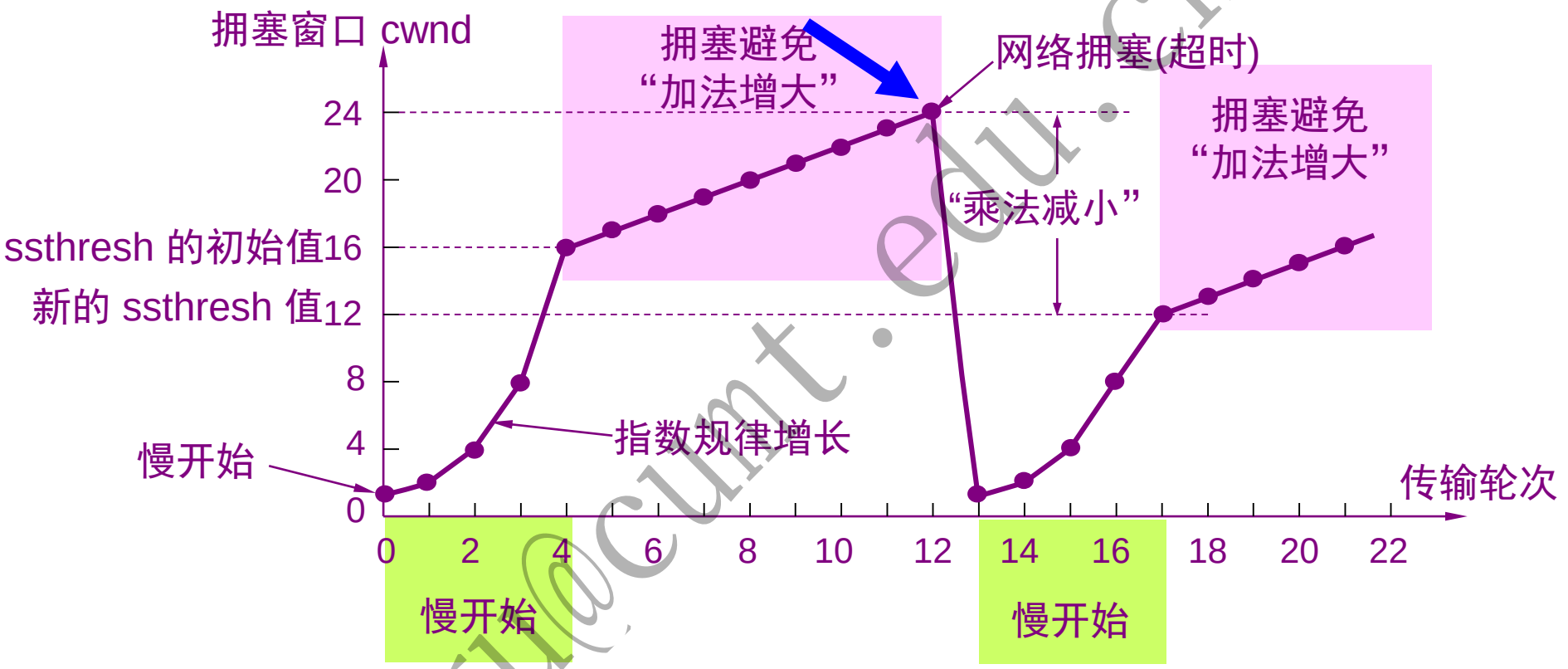


拥塞避免阶段有“加法增大” (Additive Increase) 的特点。这表明在拥塞避免阶段，拥塞窗口 cwnd 按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。

拥塞避免算法

- “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，并非指完全能够避免了拥塞，而且利用以上的措施要完全避免网络拥塞还是不可能的。

慢开始和拥塞避免算法的实现举例

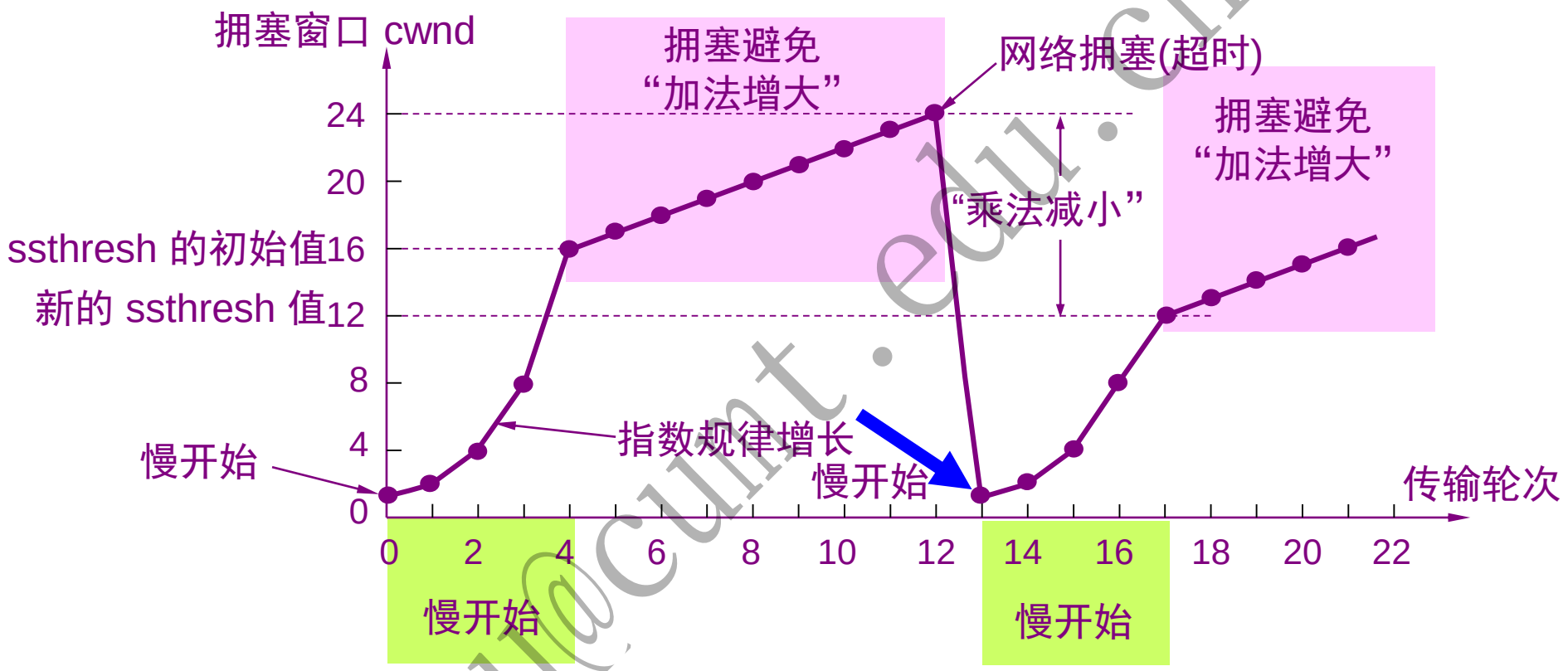


假定拥塞窗口的数值增长到 24 时，网络出现超时，表明网络拥塞了。

当网络出现拥塞时

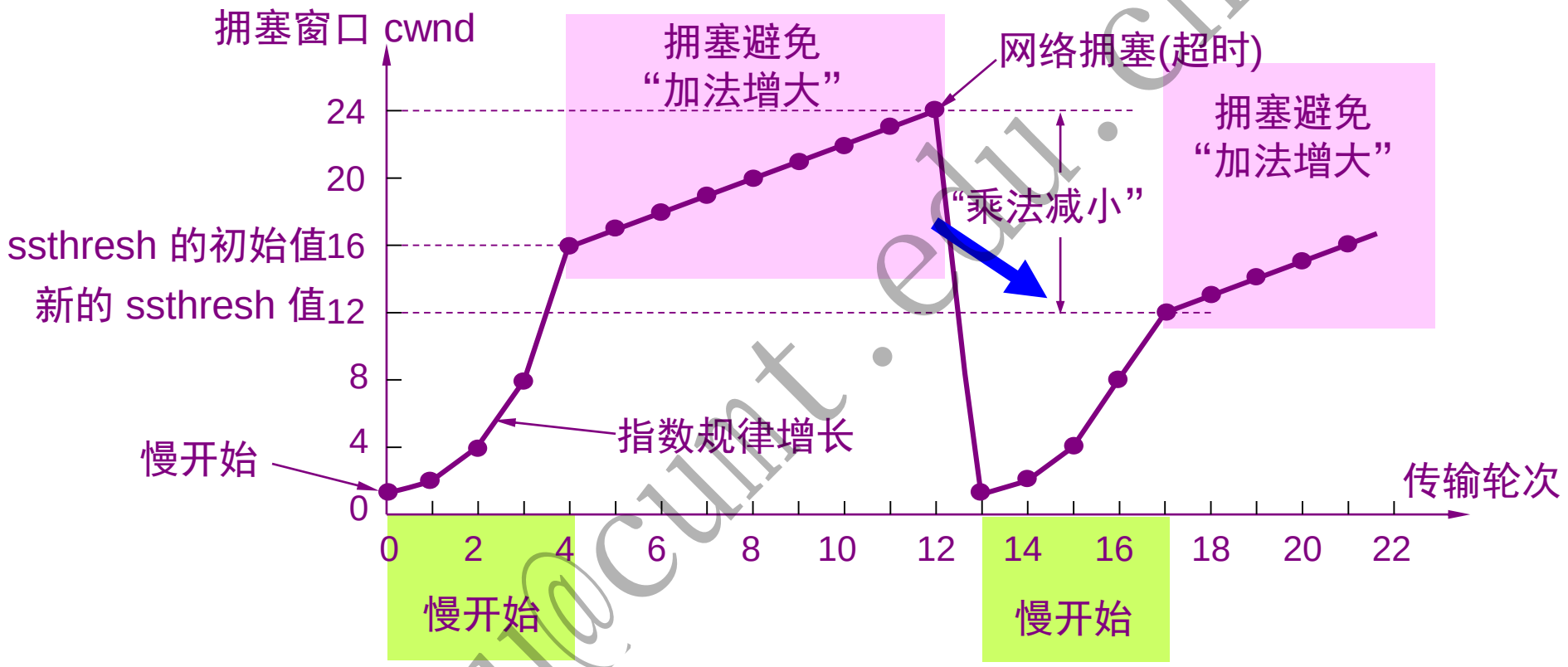
- 无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（重传定时器超时）：
  - ssthresh = max(cwnd/2，2)
  - cwnd = 1
  - 执行慢开始算法
- 这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

慢开始和拥塞避免算法的实现举例



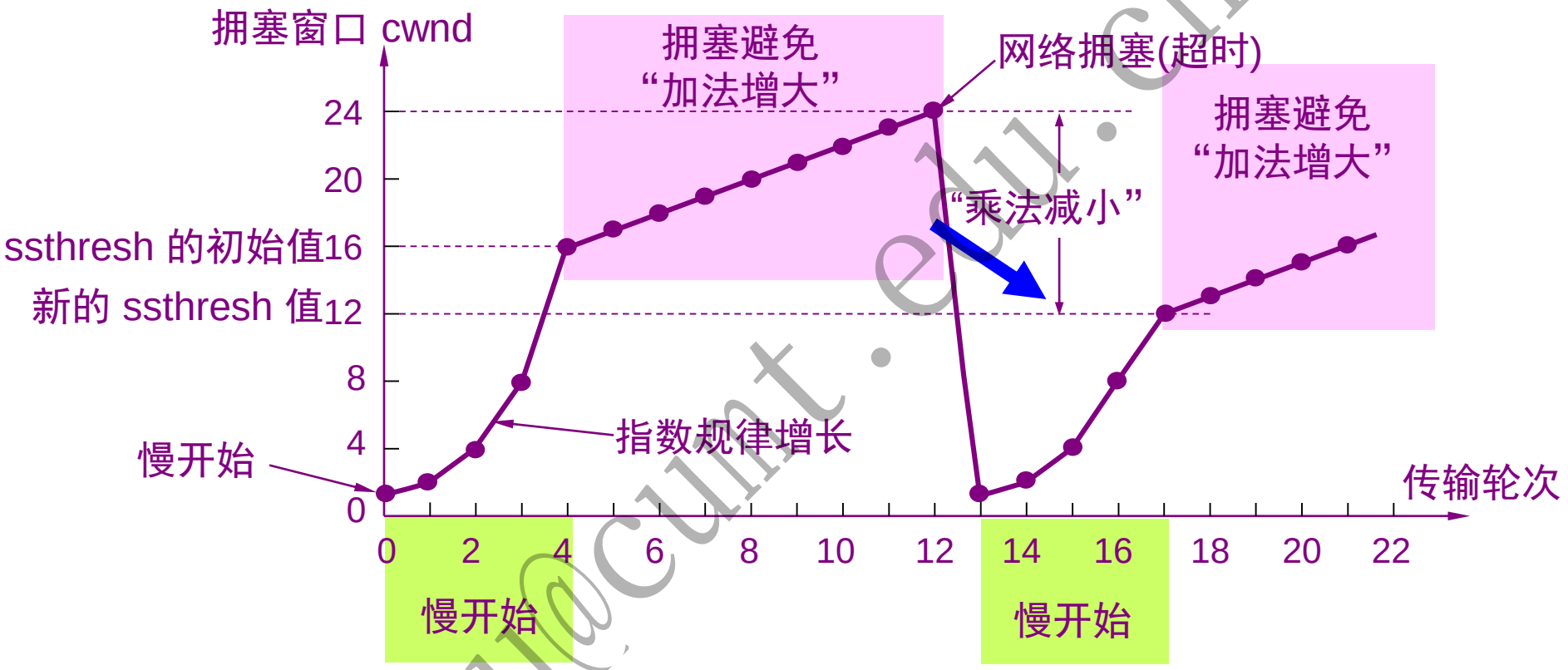
更新后的 ssthresh 值变为 12（即发送窗口数值 24 的一半），拥塞窗口再重新设置为 1，并执行慢开始算法。

慢开始和拥塞避免算法的实现举例



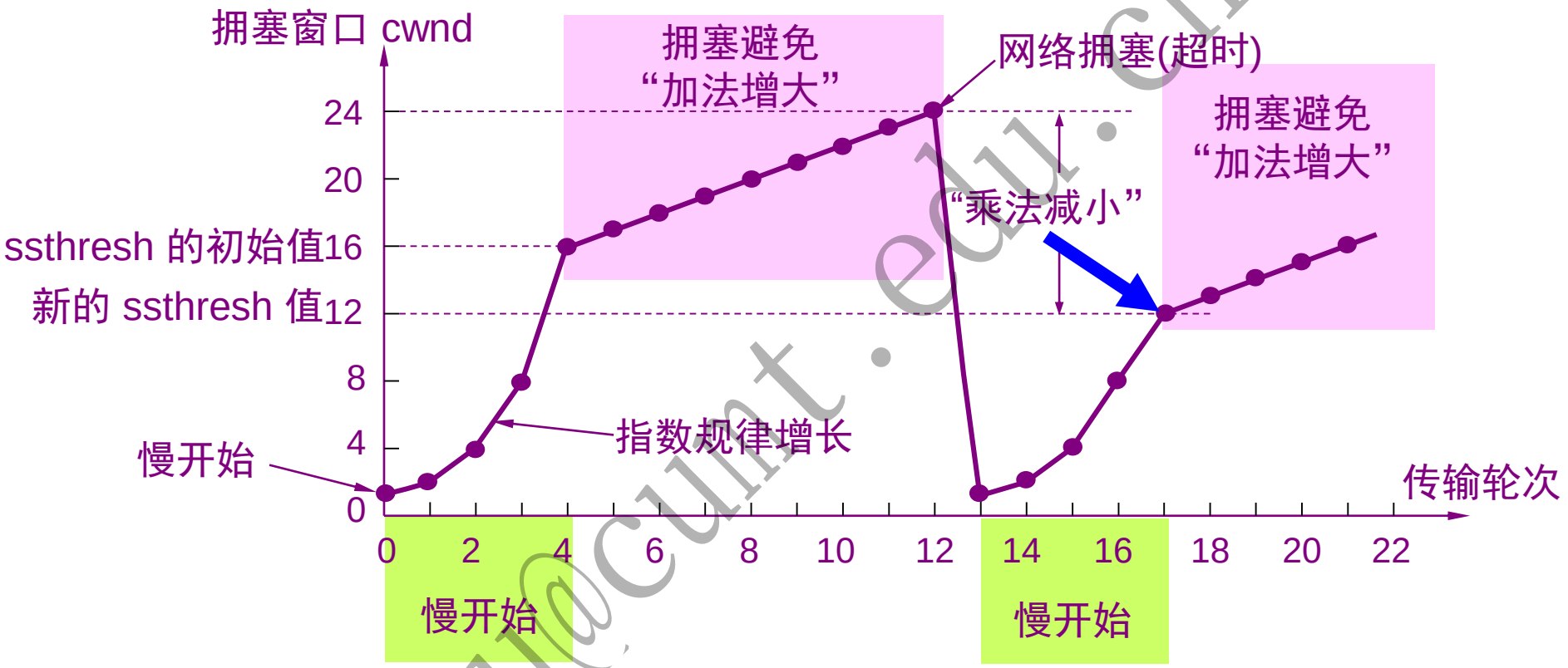
“乘法减小(multiplicative decrease)”是指不论在慢开始阶段还是拥塞避免阶段，只要出现一次超时（即出现一次网络拥塞），就把慢开始门限值 ssthresh 设置为当前的拥塞窗口值乘以 0.5。

慢开始和拥塞避免算法的实现举例



当网络频繁出现拥塞时，ssthresh 值就下降得很快，以便大大减少注入到网络中的分组数。

慢开始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，拥塞窗口按线性规律增长，每经过一个往返时延就增加一个 MSS 的大小。

拥塞控制习题1：

- 设TCP 的ssthresh的初始值为8(单位为报文段)。
- 当拥塞窗口上升到12时该网络发生了超时，TCP使用慢开始和拥塞避免。
- 试分别求出第1 次到第15 次传输的各拥塞窗口大小并说明拥塞控制窗口每一次变化的原因。

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cwnd															

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cwnd	1	2	4	8	9	10	11	12	1	2	4	6	7	8	9

- TCP使用慢开始，第一次发送1段，窗口大小为1；
- 随后按2的指数增长，增长到ssthresh的初始值8，需要经过$\log_2 8=3$次，即第4次；
- 随后进入第一轮拥塞避免算法，按线性增长到12，需要12-8=4次，即第8次；
- 此时，发生超时将开始新一轮的慢开始，窗口重新设置为1，同时新的ssthresh值更新为12/2=6；
- 新一轮慢开始阶段由1按指数增长到大于6，需要3次（$2^3=8>6$），即发生超时后的第4次，总第8+4=12次。
- 进入第二轮拥塞避免，窗口值由新的ssthresh值6开始线性增长，传输到第15次时，线性增长了15-12=3次，此时窗口值为6+3=9。

存在的问题

慢启动和拥塞避免是1988年提出的拥塞控制算法，称为Tahoe TCP，可能存在以下不足：

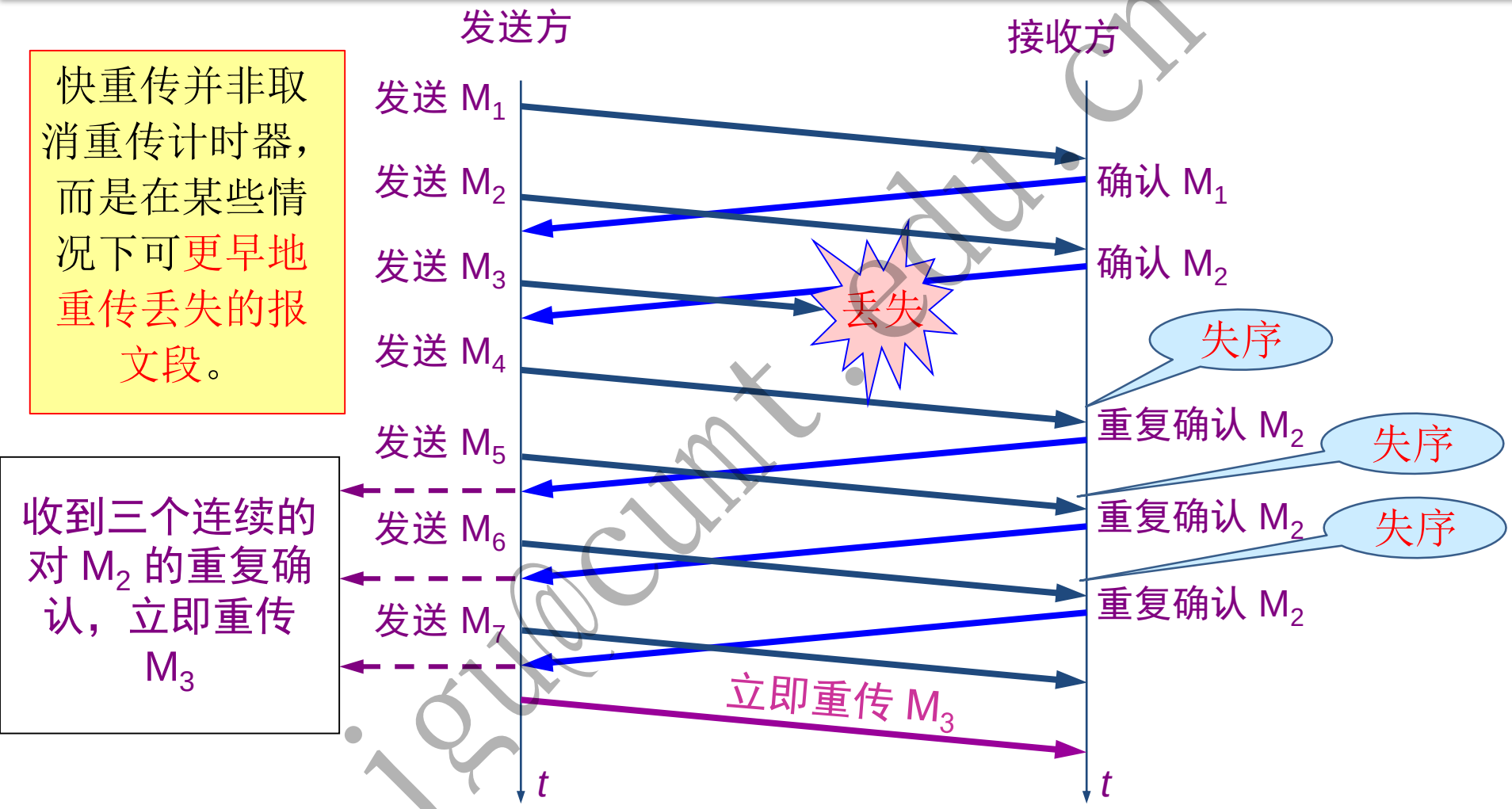
- 有时，个别报文段会在网络中丢失，但实际上网络并未发生拥塞。
- 然而，发送方因为迟迟收不到确认，就会产生超时，从而误认为网络发生了拥塞。
- 结果就是发送方会错误地启动慢开始，把拥塞窗口cwnd又设置为1，导致传输效率降低。

Q30：快重传和快恢复算法？

- 1990年出现的TCP Reno版本增加了“快速重传”算法和“快速恢复”算法，避免了当网络拥塞不够严重时采用“慢启动”算法而造成过大地减小发送窗口尺寸的现象。
- 采用快重传算法可以让发送方尽早知道发生了个别报文段的丢失。
- 快重传算法首先要求接收方不要等待自己发送数据时才进行捎带确认，而是要立即发送确认，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。

发送方只要一连收到某TCP报文段的三个重复确认，就立即进行重传(即快重传)，这样就不会把出现超时误认为发生拥塞。

快重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段。



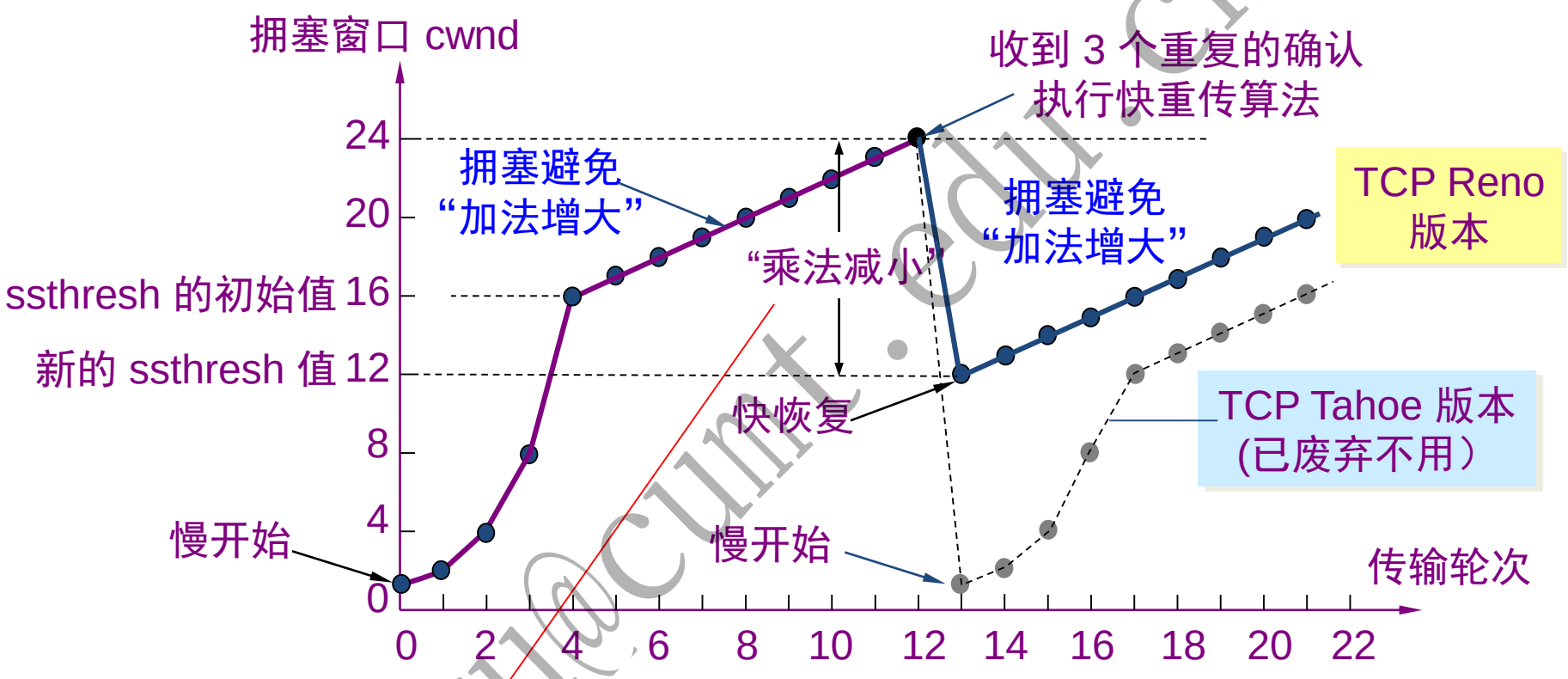
收到三个连续的对 M_2 的重复确认，立即重传 M_3

使用快重传可以使整个网络的吞吐量提高约20%。

快恢复算法

- 快速恢复是配合快速重传使用的算法。
- 当发送端收到连续三个重复的确认时，发送方认为现在只是丢失了个别的报文段，网络很可能没有发生拥塞，因此不执行慢开始算法，而是执行快恢复算法。
- 拥塞窗口 cwnd不设置为 1，而是执行“乘法减小”算法，把慢开始门限 ssthresh 减半，即为cwnd/2。同时设置cwnd的新值为减半后的 ssthresh 数值，即cwnd = ssthresh。

从连续收到三个重复的确认转入拥塞避免



执行“乘法减小”算法，把慢开始门限 ssthresh 减半，但是并不立即执行慢开始算法，然后开始执行拥塞避免算法，使拥塞窗口缓慢地线性增大。

拥塞控制习题2：

- 设TCP 的ssthresh的初始值为8(单位为报文段)。
- 采用Reno TCP拥塞控制方法，当拥塞窗口上升到12时收到 3 个重复的确认，在第13次传输后发生了超时，试分别求出第1 次到第15 次传输的各拥塞窗口大小，给出不同算法的执行阶段，说明不同阶段的ssthresh门限值的大小。

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cwnd															

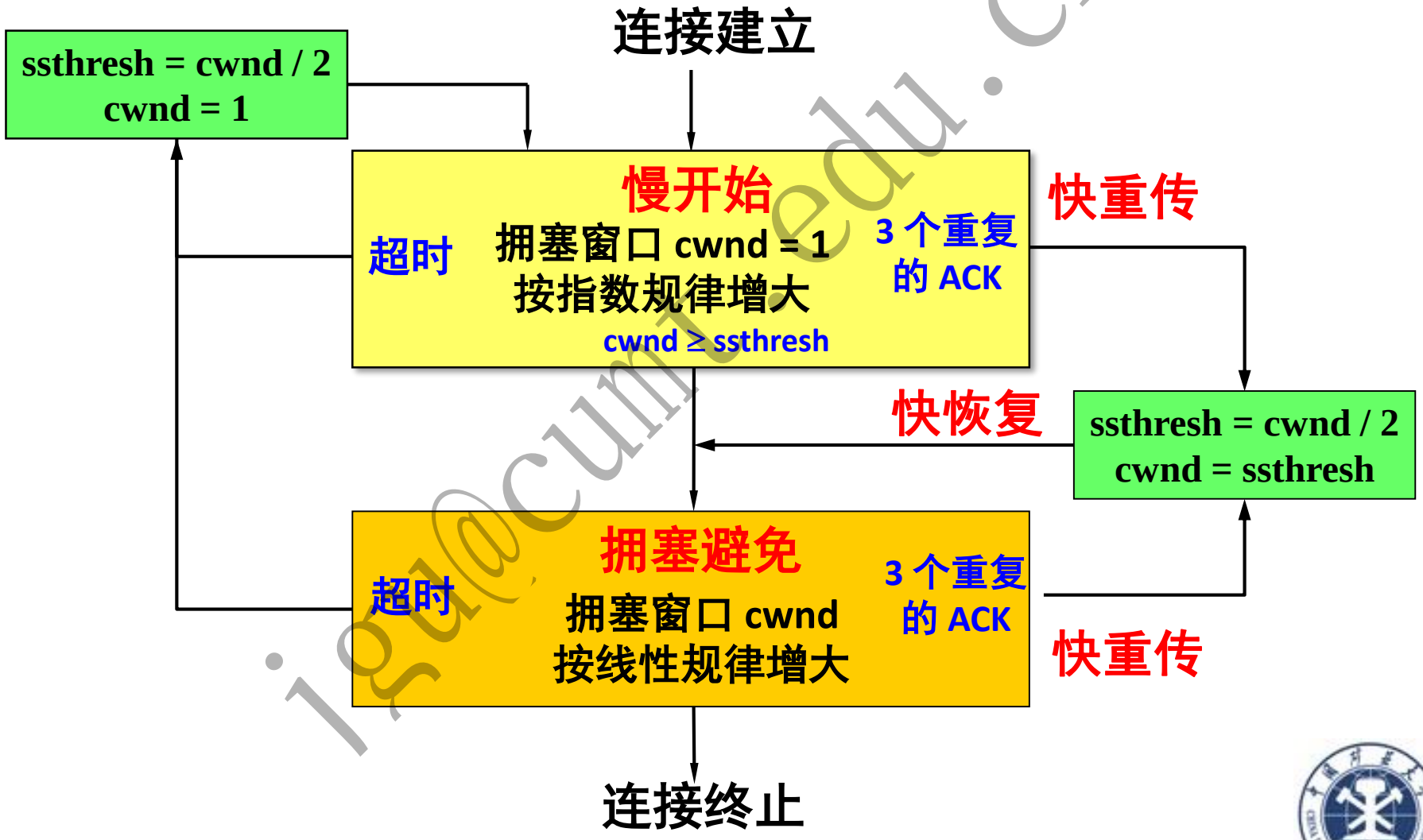
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cwnd	1	2	4	8	9	10	11	12	6	7	8	9	10	1	2

- TCP使用慢开始，第一次发送1段，窗口大小为1；
- 随后按2的指数增长，增长到ssthresh的初始值8，需要经过$\log_2 8=3$次，即第4次；
- 随后进入第一轮拥塞避免算法，按线性增长到12，需要12-8=4次，即第8次；
- 此时，收到 3 个重复的确认，窗口cwnd和ssthresh值更新为原来cwnd的一半，即12/2=6，开始快重传和快恢复算法；
- 进入新一轮拥塞避免阶段，按线性增长到10；
- 在第13次传输后发生超时，开始新一轮的慢启动，窗口重新设置为1，同时新的ssthresh值更新为10/2=5；
- 新一轮慢启动阶段，在第15次传输时由1按指数增长到2。

快恢复算法的一种改进

- 把快恢复开始时的拥塞窗口cwnd值再增大一些（增大3个报文段的长度），即等于新的ssthresh + 3 × MSS。
- 这样做的理由是：既然发送方收到3个重复的确认，就表明有3个分组已经离开了网络。这3个分组不再消耗网络的资源而是停留在接收方的缓存中。可见此时的网络中并不是堆积了分组而是减少了3个分组。因此可以适当把拥塞窗口扩大些。

Q31：TCP拥塞控制流程图？



Q32：拥塞控制与流量控制的关系？

- TCP流量控制由接收端通过窗口机制限制发送端向网络中输入更多报文段，可以在一定程度上减缓拥塞的负面影响。
- TCP拥塞控制由发送端通过对当前网络拥塞状况的监测实施有节制的报文段发送。

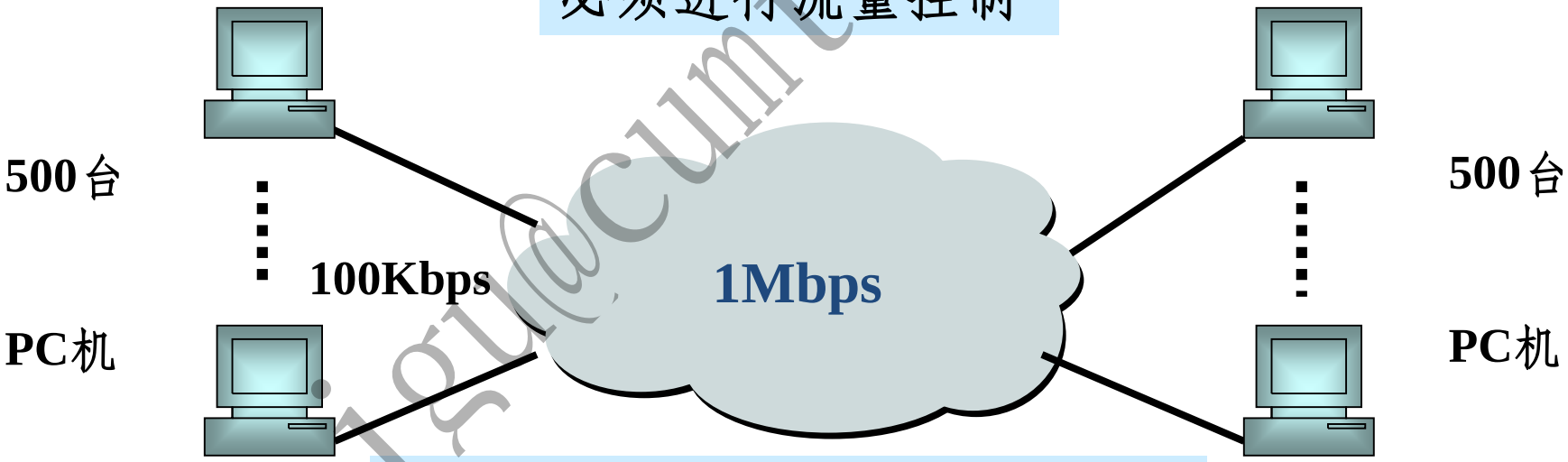
拥塞控制和流量控制之所以常常被弄混，是因为某些拥塞控制算法是向发送端发送控制报文，并告诉发送端，网络已出现麻烦，必须放慢发送速率。这点又和流量控制是很相似的。

流量控制与拥塞控制

- 流量控制(flow control)只与特定的发送方和特定的接收方之间的点到点流量有关，它的任务是让发送方的发送速率不要太快，既要让接收方来得及接收，也不要使网络发生拥塞，以便确保一个快速的发送方不会持续地以超过接收方吸收能力的速率传输数据。
- 流量控制可以有效的防止由于网络中瞬间的大量数据对网络带来的冲击，保证用户网络高效而稳定的运行。但是，在某段时间，若对网络中某资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏——产生拥塞(congestion)。

拥塞控制与流量控制的适用场景

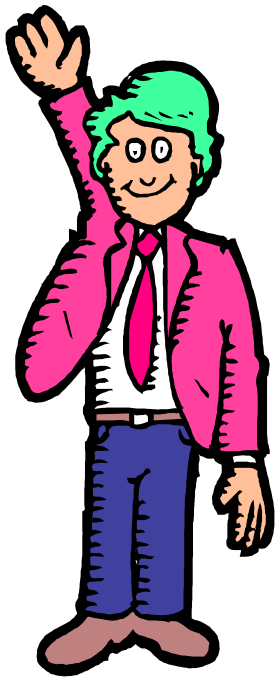
必须进行流量控制



无需流量控制，但要拥塞控制

拥塞控制与流量控制的异同

拥塞控制	流量控制
拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。	流量控制所要做的前提，就是确保发送端和接收端的数据传输速率的匹配，同时也要尽可能提高传输效率。
拥塞控制就是防止过多的数据注入到网络中，使网络中的路由器或链路不致过载。	流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。
拥塞控制是一个全局性的过程，涉及到所有的主机、所有的路由器，以及与降低网络传输性能有关的所有因素。	流量控制往往指点对点通信量的控制，是个端到端的问题（接收端控制发送端）。



THANK YOU!