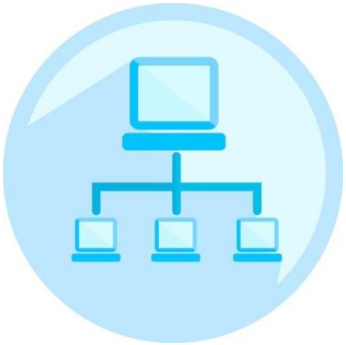
# 计算机网络

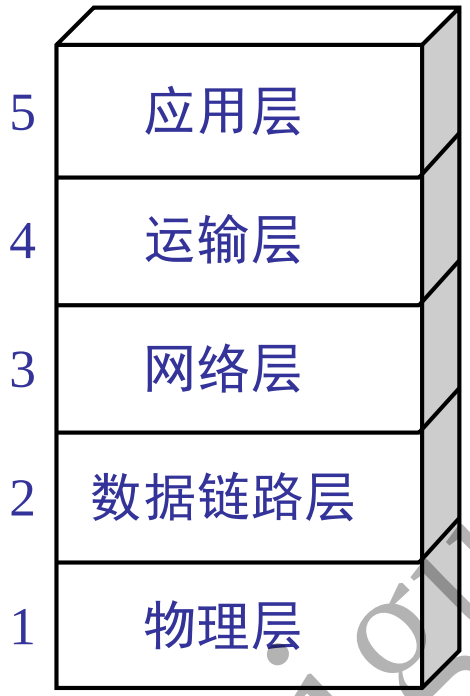
顾 军

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)

# Q21：确认重传机制够用吗？

## TCP 可靠传输的具体实现

- 字节流：所有确认都是基于字节序号而不是报文
- 面向连接：建立一条虚连接，协商参数
- 确认机制：
  - 自动重传请求ARQ（停止等待协议）
  - 连续ARQ（流水线发送、累积确认）
  - 快速确认、延迟确认、选择确认（SACK选项）
- 超时重传机制：
  - Karn修正算法估算较为合理的超时重传时间 RTO
- 窗口机制：
  - TCP 连接的每一端都设有两个窗口（发送和接收窗口）
  - 接收窗口决定发送窗口，四个窗口处于动态变化之中

# 影响网络应用高效实现的原因分析

发送方　传输网络　接收方

## 导致网络不畅的三个潜在因素

- 在传输网络有效工作的情况下，发送方能否高效发送数据、接收方能否高效读取数据——发掘发送方的发送能力和接收方的接收能力
- 在传输网络有效工作的情况下，接收方如何让发送方支持自己高效获取数据——接收方的接收能力自适应调整

单对收发双方的局部角度

- 在发送方和接收方有效工作的情况下，如何让传输网络不发生过载——传输网络的传输能力的有效控制

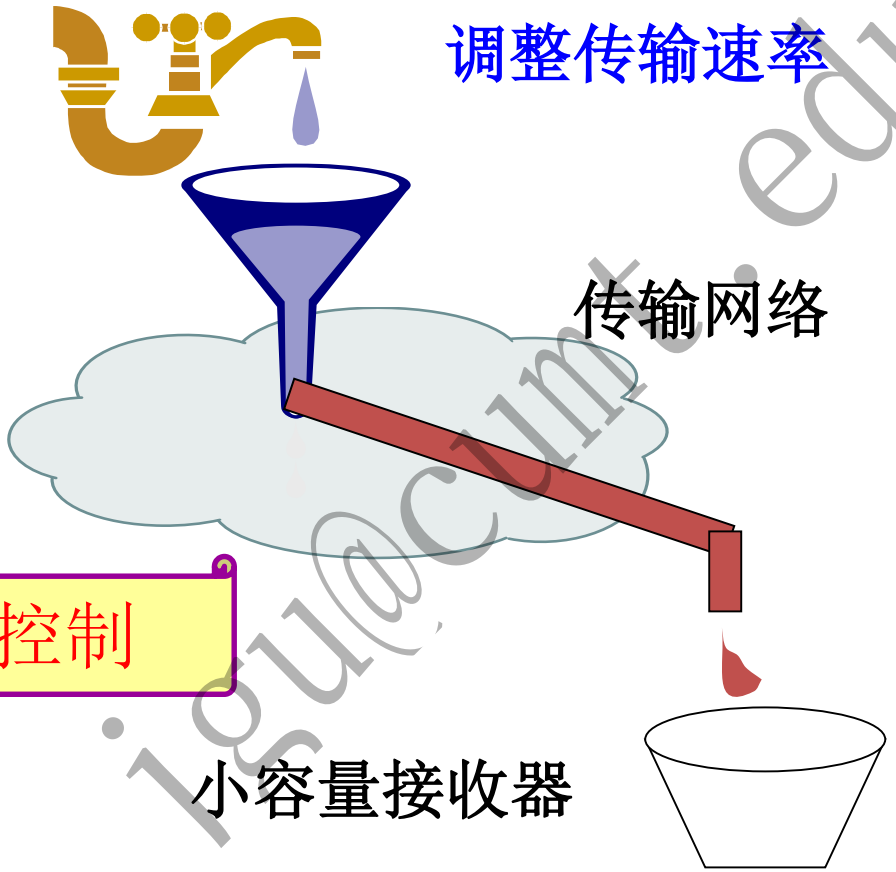
→ 全体收发双方的全局视野

# 一、TCP报文段的发送时机问题

- 应用进程把数据传送到TCP的发送缓存后，剩下的发送任务就由TCP来控制了。
- TCP 报文段的发送时机可以用不同的控制机制：
  - 第一种机制是 TCP 维持一个变量，它等于最大报文段长度 MSS。只要缓存中存放的数据达到 MSS 字节时，就组装成一个 TCP 报文段发送出去。
  - 第二种机制是由发送方的应用进程明确指示TCP何时发送报文段，即 TCP 支持的推送 (push)操作。
  - 第三种机制是发送方设置的计时器的期限到了，这时就把当前已有的缓存数据装入报文段（但长度不能超过 MSS）发送出去。
- 但是，如何控制 TCP 发送报文段的时机，以便提高TCP的传输效率仍然是一个较为复杂的问题。

# 二、端系统接收能力的动态变化问题

发送方



调整传输速率

传输网络

小容量接收器

流量控制

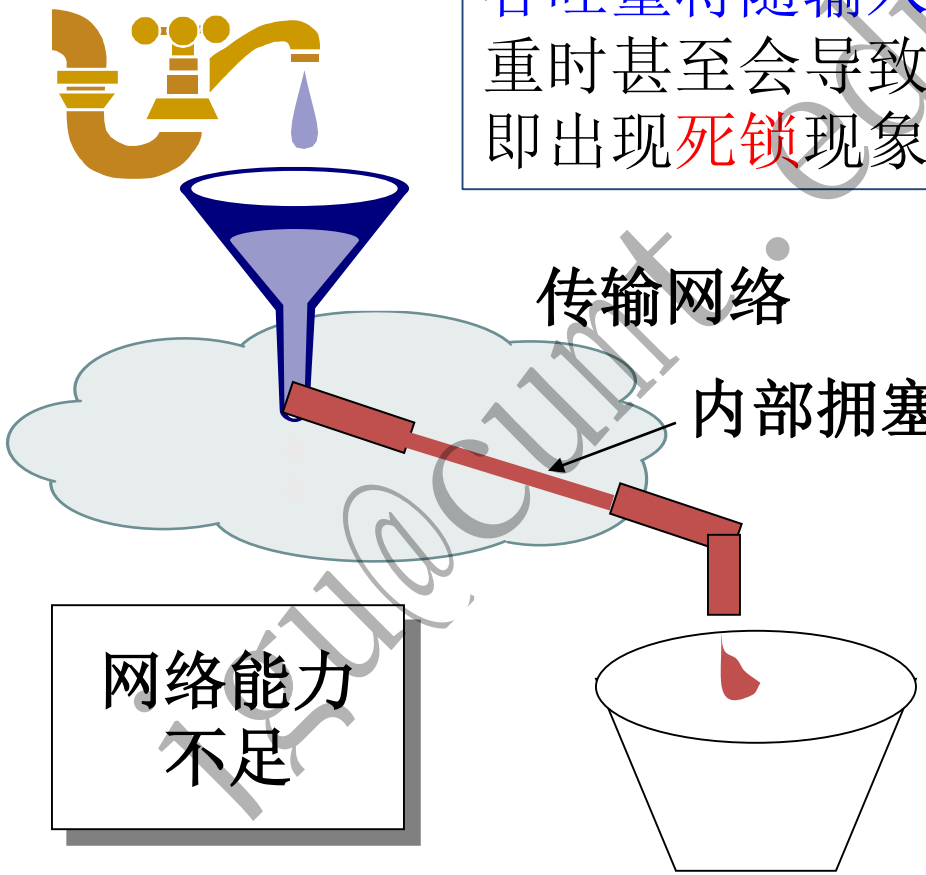
一般说来，总是希望数据传输得更快一些。但如果发送方把数据发送得过快，接收方就可能来不及接收，这就会造成数据的丢失。

接收能力不足

# 三、网络传输能力不足的问题

若网络中有许多资源同时呈现供应不足，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，严重时甚至会导致网络通信业务陷入停顿，即出现死锁现象。

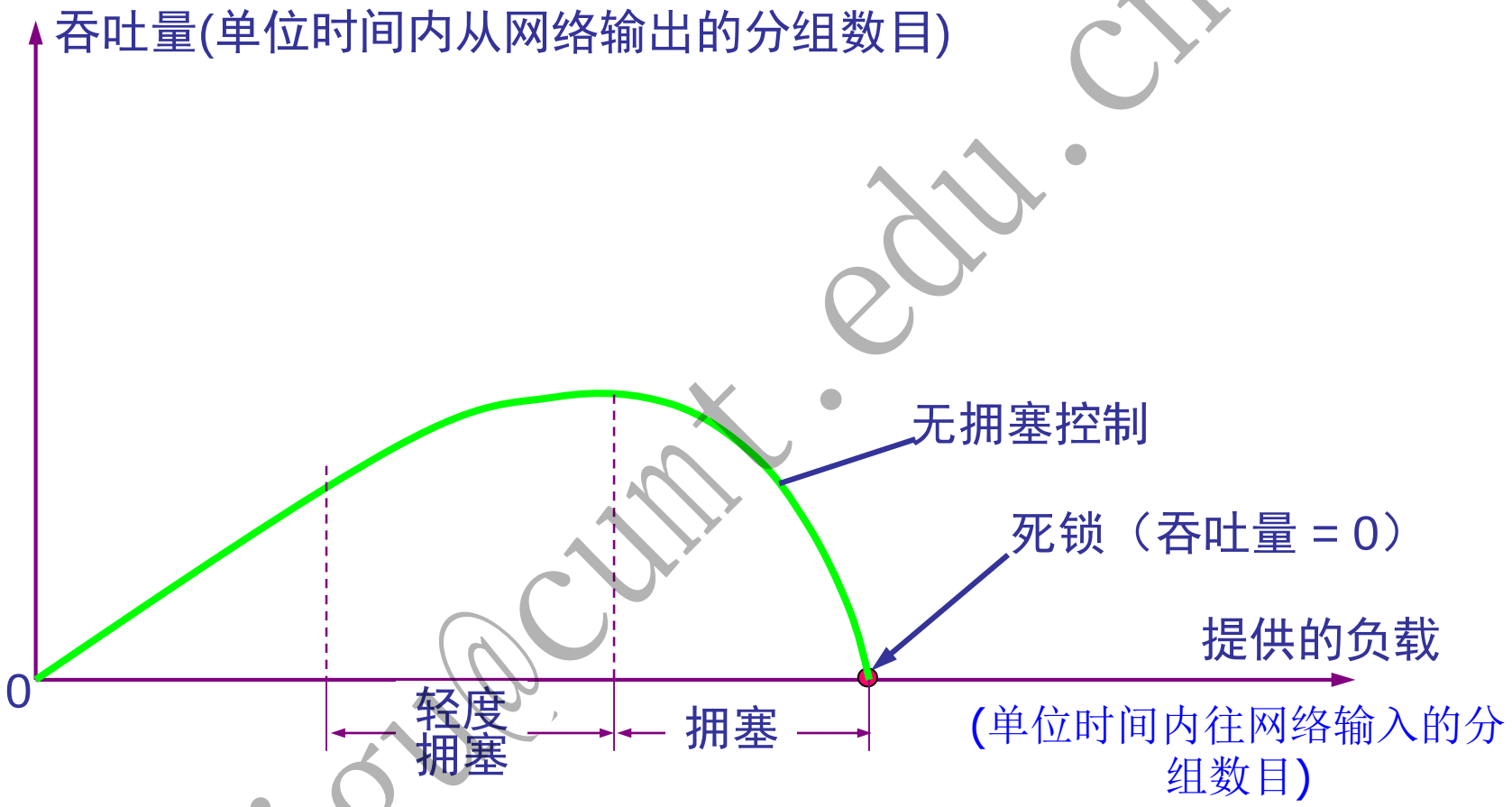
发送方




拥塞控制

大容量接收器

# 网络资源不足引起的后果




出现拥塞的条件：

**∑对资源需求 > 可用资源**

# Q20：TCP 报文段何时发送？

- 糊涂窗口综合症问题：
  - 当发送应用程序产生数据很慢，或接收应用程序读取数据（或消耗数据）很慢，或者两种情况都存在时，发送方和接收方都可能产生糊涂窗口综合症。
- 不管是上述哪一种情况，都会使得发送数据的报文段很小，这就引起操作效率的降低。

# 发送方糊涂窗口综合症的症状

- 如果发送方是产生数据很慢的应用程序服务，例如，一次产生一个字节。
  - 那么TCP发送的报文段只包括一个字节的数据，则意味着发送一个字节需要形成 41 字节长的 IP 数据报（20字节的TCP首部和20字节的IP首部）。
  - 若接收方TCP立即确认，构成的是40字节长的数据报（假定没有数据发送）。
  - 若用户要求远地主机回送这一字节，则用户仅发一个字节时，线路上就需传送总长度为 162 字节（41+40+41+40）共 4 个报文段。
  - 数据的传送效率是1/162，很低。

# 发送方糊涂窗口综合症的解决方法

- 防止发送方的TCP逐个字节地发送数据。
  - ✓ 强迫发送端的TCP收集数据，然后用一个更大的数据块来发送。
  - ✓这就需要推迟发回确认报文，并尽量使用捎带确认的方法。
- 那么，发送端的TCP要等待多长时间呢？
  - ✓如果等待时间过长，它会使整个过程产生较长的时延。
  - ✓如果等待时间不够长，它就可能发送较小的报文段。
- ◆ Nagle找到了一个很好的解决方法。

# Nagle算法

- 若发送应用进程把要发送的数据逐个字节地送到TCP 的发送缓存，则发送方就把第一个数据字节先发送出去，把后面到达的数据字节都缓存起来。
- 当发送方收到对第一个数据字符的确认后，再把发送缓存中的所有数据组装成一个报文段发送出去，同时继续对随后到达的数据进行缓存。
- 继续发送下一个报文段的条件可以是以下三种情况：
  - 收到对前一个报文段的确认后；
  - 到达的数据已达到发送窗口大小的一半；
  - 到达的数据已达到报文段的最大长度。

# Nagle算法的优点

- Nagle算法的优点就是简单，并且它考虑到应用程序产生数据的速率，以及网络运输数据的速率。
  - 若应用程序比网络更快，则报文段就更大（最大报文段）。
  - 若应用程序比网络慢，则报文段就较小（小于最大报文段）。

# 接收方糊涂窗口综合症的症状

- 当接收方的 TCP 缓冲区已满，接收方会向发送方发送窗口大小为 0 的报文。
- 若此时接收方的应用进程以交互方式每次只从接收缓存里读取1个字节，然后接收方会向发送方发送确认，并把发送窗口大小设为1个字节。
- 接着，发送方又发送1个字节的数据（发送的 IP 数据报是 41 字节长），于是接收窗口又满了，如此循环往复。
- 这又是一个低效率问题和糊涂窗口综合症。

# 接收方糊涂窗口综合症的解决方法

- 接收方选择在以下两种情况下延迟发出确认报文，并向发送方通知当前的窗口大小。
  - 让接收方等待一段时间，使得接收缓存已有足够空间容纳一个最长的报文段；
  - 等到接收缓存已有一半空闲的空间。
- 这种延迟的确认不但可以防止这种糊涂窗口综合症，还可以减少通信量——接收端不需要确认每一个报文段。
- 但也有一个缺点，就是可能迫使发送端重传其未被确认的报文段。
- 可以用协议来平衡这个优点和缺点，例如，现在定义了确认的延迟不能超过500毫秒。

# 两种方法可配合使用

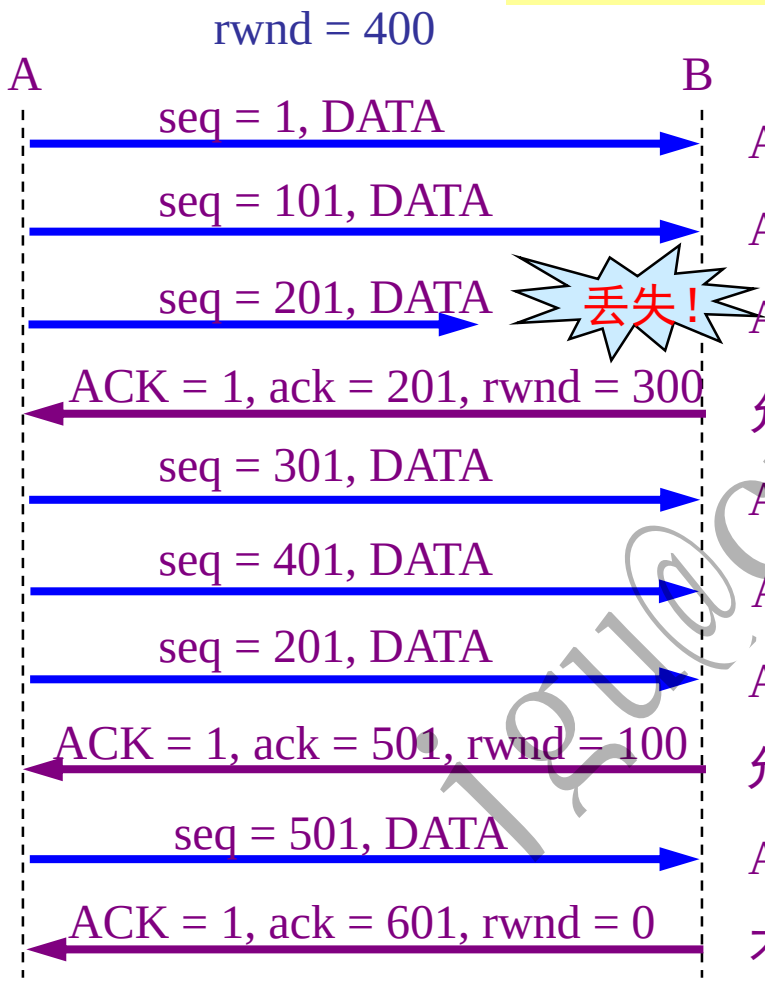
- 上述两种方法可配合使用，使得在发送方不发送很小的报文段的同时，接收方也不要在缓存刚刚有了一点小的空间就急忙把这个很小的窗口大小信息通知给发送方。

# Q21：基于滑动窗口实现流量控制？

- 流量控制用于防止在端口阻塞的情况下丢帧，这种方法是当发送或接收缓冲区开始溢出时，通过将阻塞信号发送回源地址实现的。
  - 通常做法是，接收方向发送方提供某种直接的反馈，以便告诉发送方自己这一端的情形到底怎么样。
- TCP利用滑动窗口机制可以很方便地在 TCP 连接上实现流量控制。
  - 发送窗口除了在建立TCP连接时协商确定外，在通信过程中可以通过接收窗口通告机制对发送窗口进行调整，以便调控发送给接收方的数据流量。

# TCP流量控制举例

A 向 B 发送数据。在连接建立时，B 告诉 A：“我的接收窗口 rwnd = 400（字节）”。




A 发送了序号 1 至 100，还能发送 300 字节

A 发送了序号 101 至 200，还能发送 200 字节

A 发送了序号 201 至 300，还能发送 100 字节

允许 A 发送序号 201 至 500 共 300 字节

A 发送了序号 301 至 400，还能再发送 100 字节新数据

A 发送了序号 401 至 500，不能再发送新数据了

A 超时重传旧的数据，但不能发送新的数据

允许 A 发送序号 501 至 600 共 100 字节

A 发送了序号 501 至 600，不能再发送了

不允许 A 再发送（到序号 600 为止的数据都收到了）

# Q22：流量控制潜在的问题？

- B 向 A 发送了零窗口的报文段后不久，B 的接收缓存又有了一些存储空间。于是 B 向 A 发送了 rwnd = 400 的报文段。
- 但这个报文段在传送过程中丢失了。
- A 一直等待收到 B 发送的非零窗口的通知，而 B 也一直等待 A 发送的数据。
- 如果没有其他措施，这种互相等待的死锁局面将一直延续下去。
- 为了解决这个问题，TCP 为每一个连接设有一个持续计时器 (persistence timer)。

# 持续计时器

- 只要 TCP 连接的一方收到对方的零窗口通知，就启动该持续计时器。
- 若持续计时器设置的时间到期，就发送一个零窗口探测报文段（仅携带 1 字节的数据），而对方就在确认这个探测报文段时给出了现在的窗口值。
- 若窗口仍然是零，则收到这个报文段的一方就重新设置持续计时器。
- 若窗口不是零，则死锁的僵局就可以打破了。

# Q23：什么是拥塞控制？

- 出现拥塞的条件：

**∑对资源需求 > 可用资源**

- 第一种控制方法：同时增加所有部分的可用资源
  - 资源提供远大于资源需求
  - 看上去很美，但是成本太高
- 第二控制方法：给某些部分增加资源
  - 头痛医头，脚痛医脚
  - 例如，把结点的缓存空间扩大，或提高链路速率，或提高结点处理机的运算速度等。

# 任意增加一些资源往往会使网络拥塞恶化

- 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；
- 又如，简单提高处理机的速率可以缓解由于速率太慢而引起的网络拥塞，却会将瓶颈转移到其它地方。
- 再如，如果一个路由器没有足够的缓存空间，它就会丢弃一些新到的分组。但当分组被丢弃时，发送这一分组的源点就会重传这一分组，甚至可能还要重传多次。这样会引起更多的分组流入网络和被网络中的路由器丢弃，反而加剧了网络拥塞。

增加资源的思路都不能解决网络拥塞问题

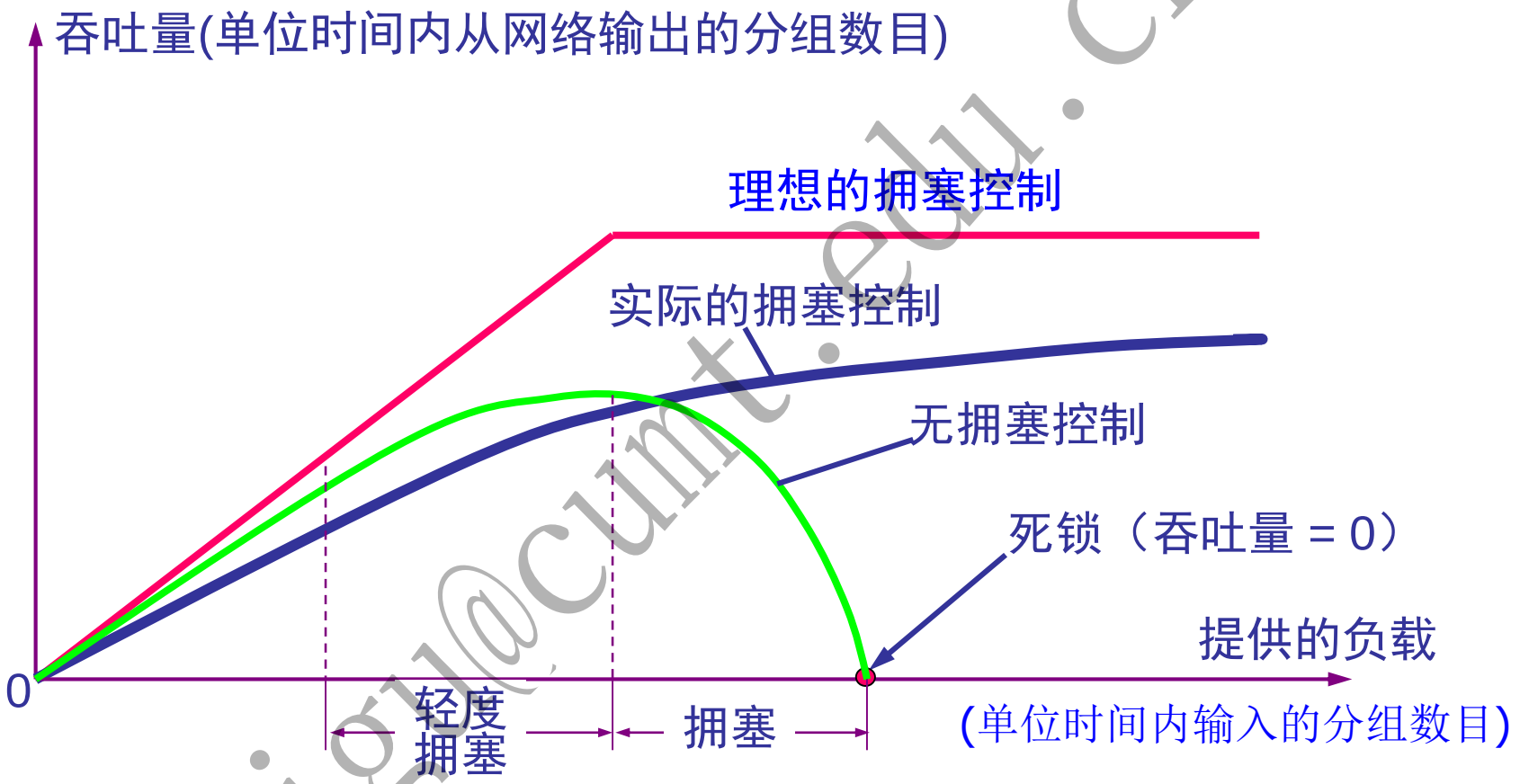
# 拥塞控制的任务

**∑对资源需求 > 可用资源**

**<**

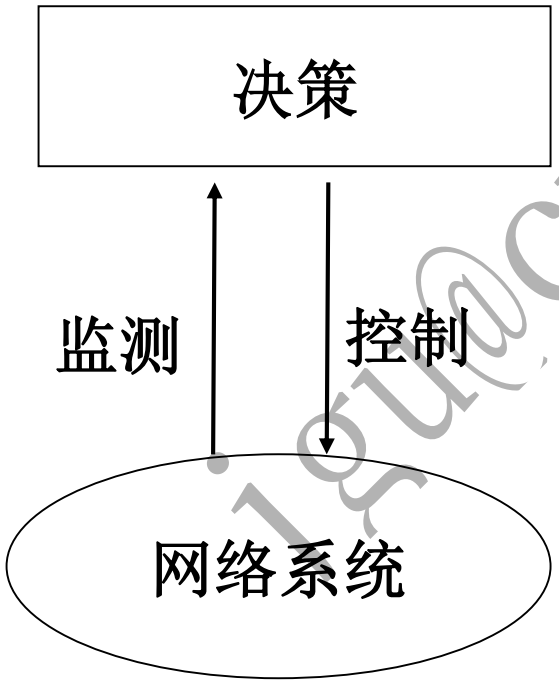
- 若网络中有许多资源同时产生拥塞，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降，因此拥塞控制的任务是确保子网能够承载所到达的流量。
  - 拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负载(负荷)。
- 拥塞控制就是防止过多的数据注入到网络中，使网络中的路由器或链路不致过载。

# 拥塞控制所起的作用



# Q24：拥塞控制的方法论问题？

- 开环控制方法就是在设计网络时事先将有关发生拥塞的因素考虑周到，力求网络在工作时不产生拥塞。
- 闭环控制方法是基于反馈环路的概念。




(1) 监测网络系统以便检测到拥塞在何时、何处发生。

(2) 将拥塞发生的信息传送到可采取行动的地方。

(3) 调整网络系统的运行以解决出现的问题。

# 拥塞发生的监测指标

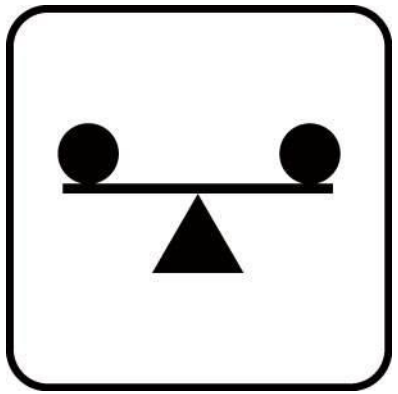
- 主要指标有：
  - 由于缺少缓存空间而被丢弃的分组的百分数；
  - 平均队列长度；
  - 超时重传的分组数；
  - 平均分组时延；
  - 分组时延的标准差，等等。
- 上述这些指标的上升都标志着拥塞的增长。

# 应防止拥塞控制引起网络性能恶化

- 实践证明，网络拥塞是一个非常复杂的问题，是一个动态的（而不是静态的）问题。
  - 比如，当前网络正朝着高速化的方向发展，这很容易出现缓存不够大而造成分组的丢失。但分组的丢失是网络发生拥塞的征兆而不是原因，相应的控制措施是不同的。
  - 简单地采用上述做法，在许多情况下，不但不能解决拥塞问题，而且还可能使网络的性能更坏。
- 在许多情况下，甚至正是拥塞控制本身会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。

# 拥塞控制的目标是系统平衡

- 网络拥塞问题的实质往往是整个系统的各个部分不匹配。
- 拥塞控制是一个全局性问题，涉及到各方面的行为，包括所有的主机、所有的路由器、路由器内部的存储转发处理过程，以及所有可能会削弱子网承载容量的其它因素。
- 只有网络系统的所有部分都平衡了，网络拥塞问题才能得到解决。
- 引入拥塞控制
  - 控制网络处理能力（路由器）
  - 控制流量输入能力（TCP端）



# Q25：TCP的拥塞控制策略是什么？

- TCP不能忽略网络中的拥塞问题。
  - 它不能过分激进地向网络中发送段，这样激进的结果只能伤害TCP自身。
  - TCP也不能过于保守，每个时间间隔只发送少量的段，因为这意味着没有利用好网络可用带宽。
- TCP使用两种不同的策略来处理网络中的拥塞。
  - 当没有拥塞时的加速数据传输策略
  - 当检测到拥塞时的减速策略



# Q26：如何判断拥塞的发生？

- 重传定时器超时
  - 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是很小的（远小于1%）。只要出现了超时，就可以猜想网络可能出现了拥塞。
- 收到三个相同（重复）的 ACK
  - 个别报文段会在网络中丢失，预示可能会出现拥塞（实际未发生拥塞），因此可以尽快采取控制措施，避免拥塞。

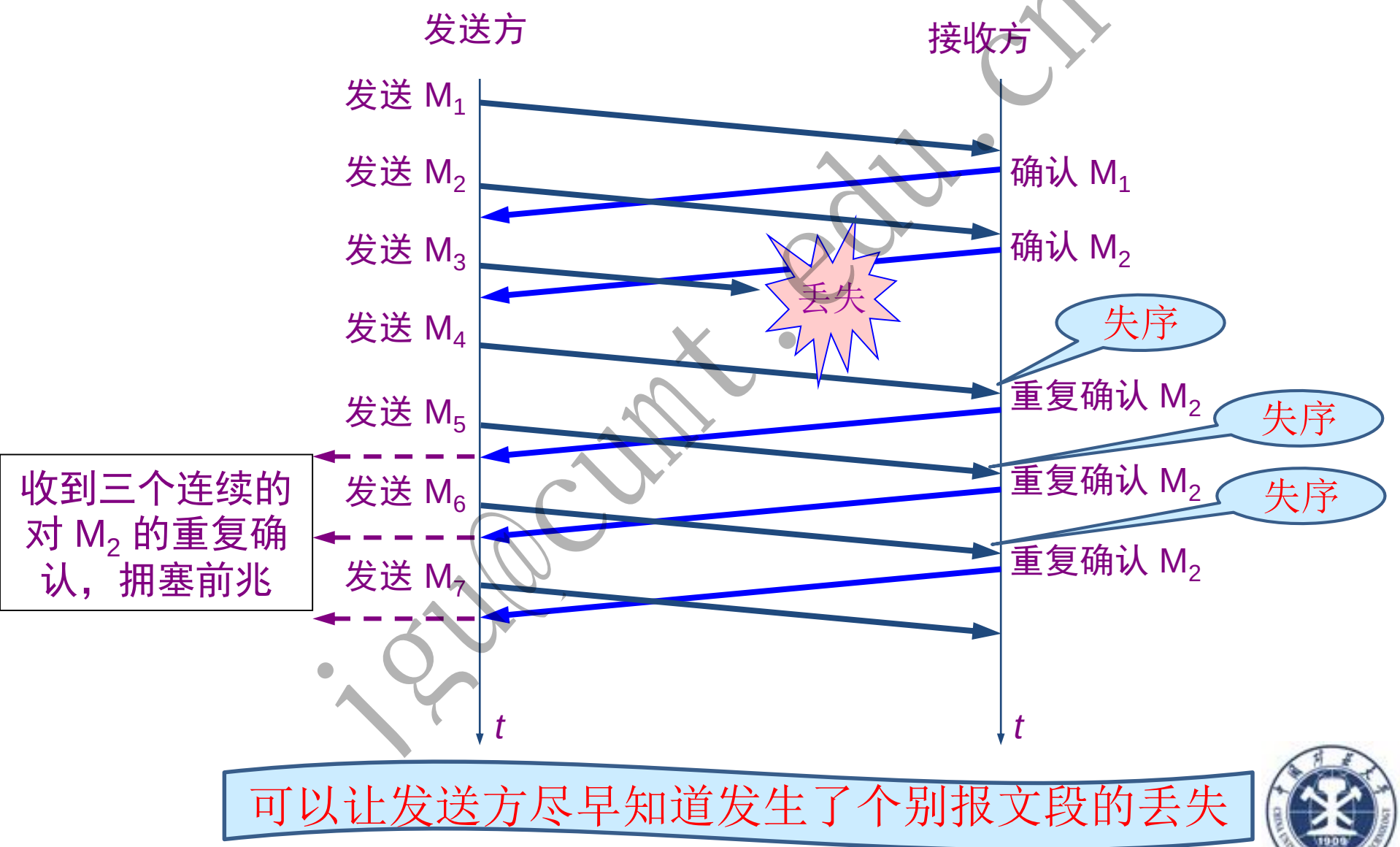
# 重复（冗余）确认的产生原因

- 即使发送端是按序发送，由于TCP报文段是封装在IP包内，IP包在传输时乱序，意味着TCP报文段到达接收端也是乱序的，乱序的话就会造成接收端发送重复（冗余）ACK。
- 至于发送重复ACK是由于乱序造成的还是包丢失造成的，便需要好好权衡一番，因为把3次重复ACK作为判定丢失的准则其本身就是估计值。

# 收到三个重复确认举例

- 接收方每收到一个失序的报文段后就立即发出重复确认。
  - 假设在某个TCP数据传输过程中，接收端依次收到发送端发出的1号和2号数据报文段，并对这两个数据报文段发送确认后，没有按次序收到3号数据报文段，而是收到了4号，这时就需要立即向发送端发送一个2号数据报文段的确认，称为重复确认。
  - 同理，如果继续收到5号、6号数据报文段，接收端仍然要向发送端发出两个2号数据报文段的重复确认。此时，发送端会收到多个2号数据报文段的重复确认，则认为可能发生了拥塞。

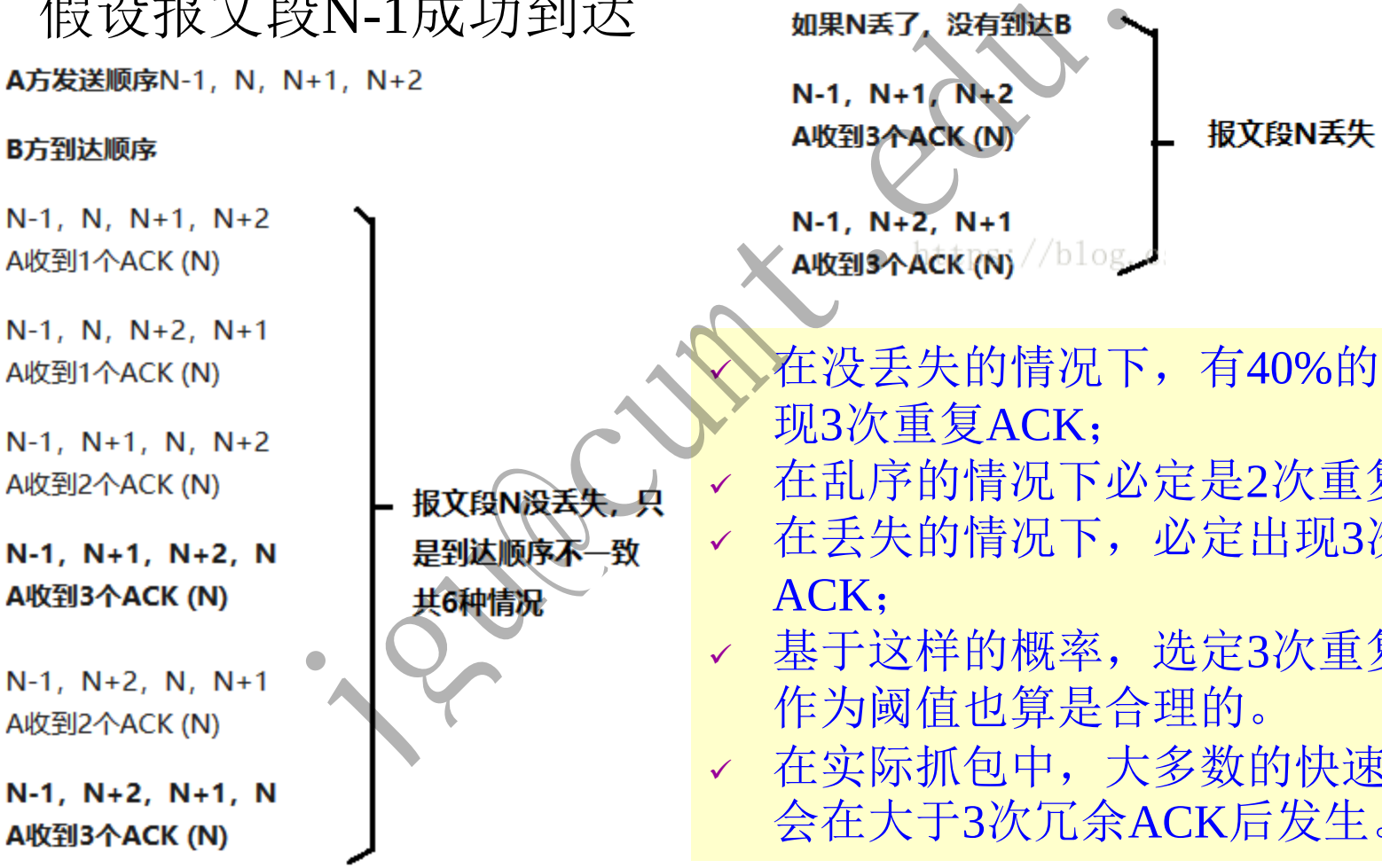
# 收到三个重复确认举例




可以让发送方尽早知道发生了个别报文段的丢失

# 为什么是三个重复确认

A为发送端，B为接收端
A的待发报文段序号为 N-1，N，N+1，N+2
假设报文段N-1成功到达




- ✓ 在没丢失的情况下，有40%的可能出现3次重复ACK；
- ✓ 在乱序的情况下必定是2次重复ACK；
- ✓ 在丢失的情况下，必定出现3次重复ACK；
- ✓ 基于这样的概率，选定3次重复ACK作为阈值也算是合理的。
- ✓ 在实际抓包中，大多数的快速重传都会在大于3次冗余ACK后发生。

# Q27：TCP拥塞窗口的作用？

- TCP基于闭环控制理论，采用基于窗口的方法进行拥塞控制。
- TCP发送方维持一个拥塞窗口 cwnd (congestion window)的状态变量值。
  - 拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化，发送方利用拥塞窗口根据网络的拥塞情况调整发送的数据量。
- 发送窗口大小不仅取决于接收方通知的接收窗口rwnd（通知窗口，advertised window），还取决于反映网络拥塞状况的拥塞窗口cwnd：

**真正的发送窗口值 = Min(通知窗口值，拥塞窗口值)**

# 发送窗口的上限值

- 发送方的发送窗口的上限值应当取为接收方窗口 rwnd 和拥塞窗口 cwnd 这两个变量中较小的一个，即应按以下公式确定：

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 rwnd > cwnd 时，则是网络的拥塞情况限制发送窗口的最大值。

# 拥塞窗口的控制原则

- 只要网络没有出现拥塞，拥塞窗口就可以再增大一些，以便把更多的分组发送出去，这样就可以提高网络的利用率。
- 但只要网络出现拥塞或有可能出现拥塞，就必须把拥塞窗口减小一些，以减少注入到网络中的分组数，以便缓解网络出现的拥塞。

# Q28：TCP的拥塞控制算法有哪些？

- 四种算法（2009年9月，RFC 5681）：
  - 慢开始
  - 拥塞避免

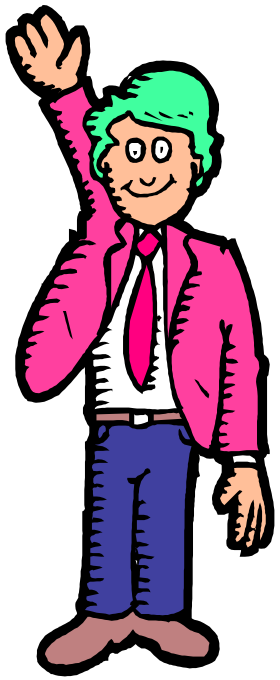
  } 1988年提出，称为Tahoe TCP

  - 快重传
  - 快恢复

  } 1990年出现，TCP Reno版本

假定：

① 数据是单方向传送的，对方只传送确认报文。
② 接收方总是有足够大的缓存空间，因而发送窗口的大小由网络的拥塞程度来决定。



# THANK YOU!