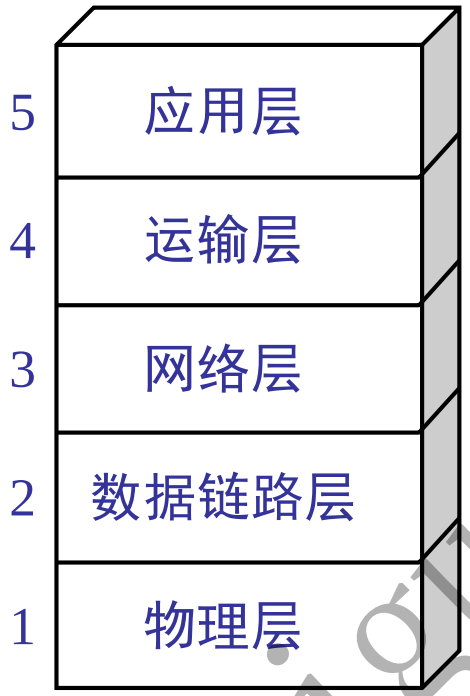
计算机网络

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



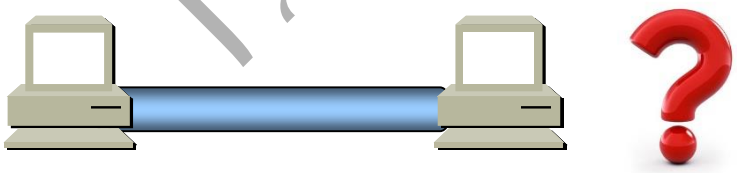
- 应用层(application layer)
- 运输层(transport layer)
- 网络层(network layer)
- 数据链路层(data link layer)
- 物理层(physical layer)

Q6：TCP如何保证端到端可靠交付？

- TCP 下面的IP层只能提供尽力而为(**Best Effort**)服务，实现的传输是不可靠的，因此TCP必须采用适当的措施才能使得两个运输层之间的通信变得可靠。
- 理想的传输条件有以下两个特点：
  - 传输信道不产生差错。
  - 不管发送方以多快的速度发送数据，接收方总是来得及处理收到的数据。
- 实际的网络不具备上述条件。

- ✓ 对收到的报文段进行确认；
- ✓ 当出现差错时重传出错的报文段。

在接收方来不及处理收到的数据时，及时通告发送方适当降低发送数据的速度。



Q7：什么是TCP连接？

- TCP 是面向连接的运输层协议。
  - 应用程序使用TCP协议之前，必须先建立TCP连接。
  - TCP 连接是一条虚连接而不是一条真正的物理连接。
  - TCP 连接就是由协议软件所提供的一种抽象。
- 每一条 TCP 连接只能有两个端点(endpoint)，而且只能是点对点的（一对一）。
- TCP 连接的端点不是主机，不是主机的IP 地址，不是应用进程，也不是运输层的协议端口。

TCP 连接，IP 地址，套接字

- TCP 连接的端点叫做套接字(socket)或插口。
  - 端口号拼接到(concatenated with) IP 地址即构成套接字。

套接字 socket = (IP地址: 端口号)

- 每一条 **TCP** 连接唯一地被通信两端的两个端点（即两个套接字）所确定。即：

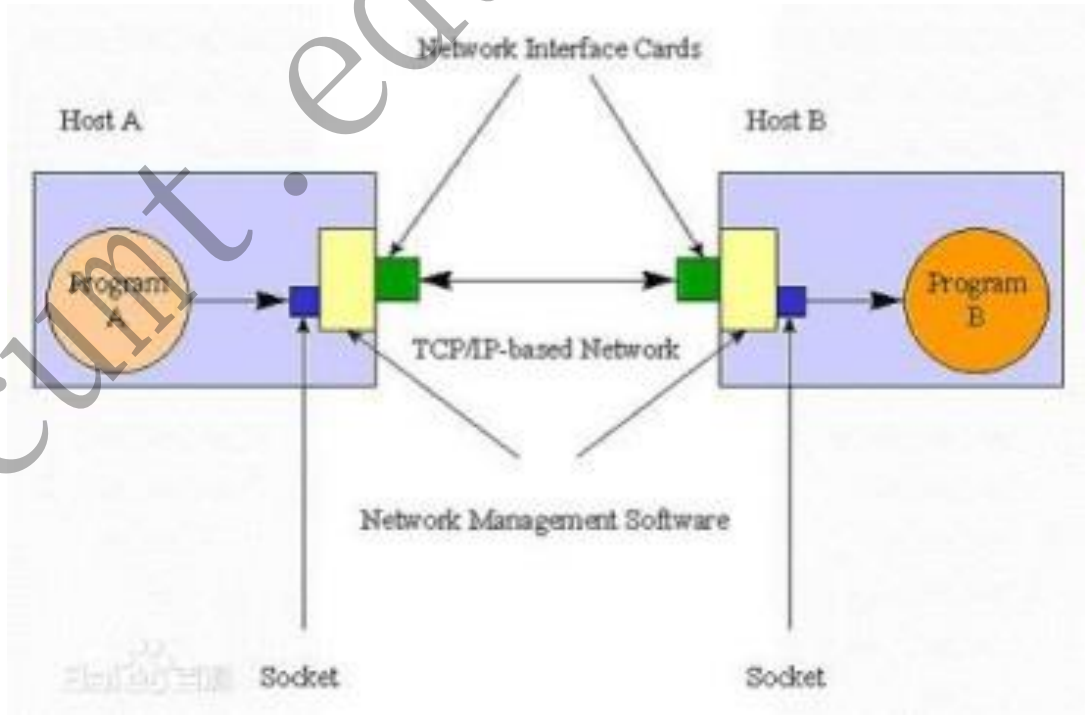
TCP 连接 ::= {socket1, socket2}
= {(IP1: port1), (IP2: port2)}

- 同一个 IP 地址可以有多个不同的 TCP 连接。
- 同一个端口号也可以出现在多个不同的 TCP 连接中。

套接字(socket)

- 套接字是网络通信过程中端点的抽象表示，包含进行网络通信必需的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。

自学
Socket编程！

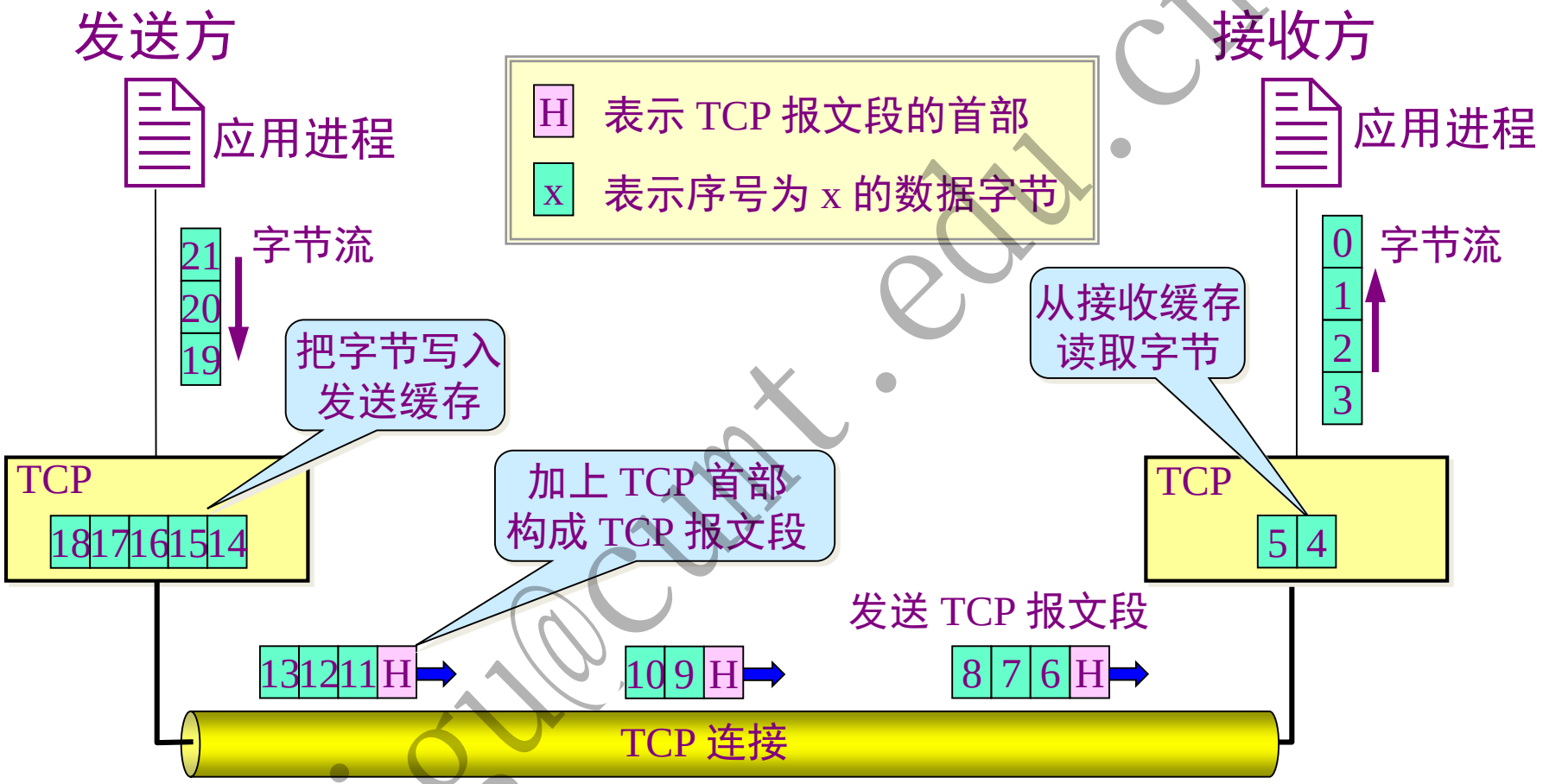


通信关联 association = {Protocol, (IP1: port1), (IP2: port2)}

Q8：如何形成TCP报文段？

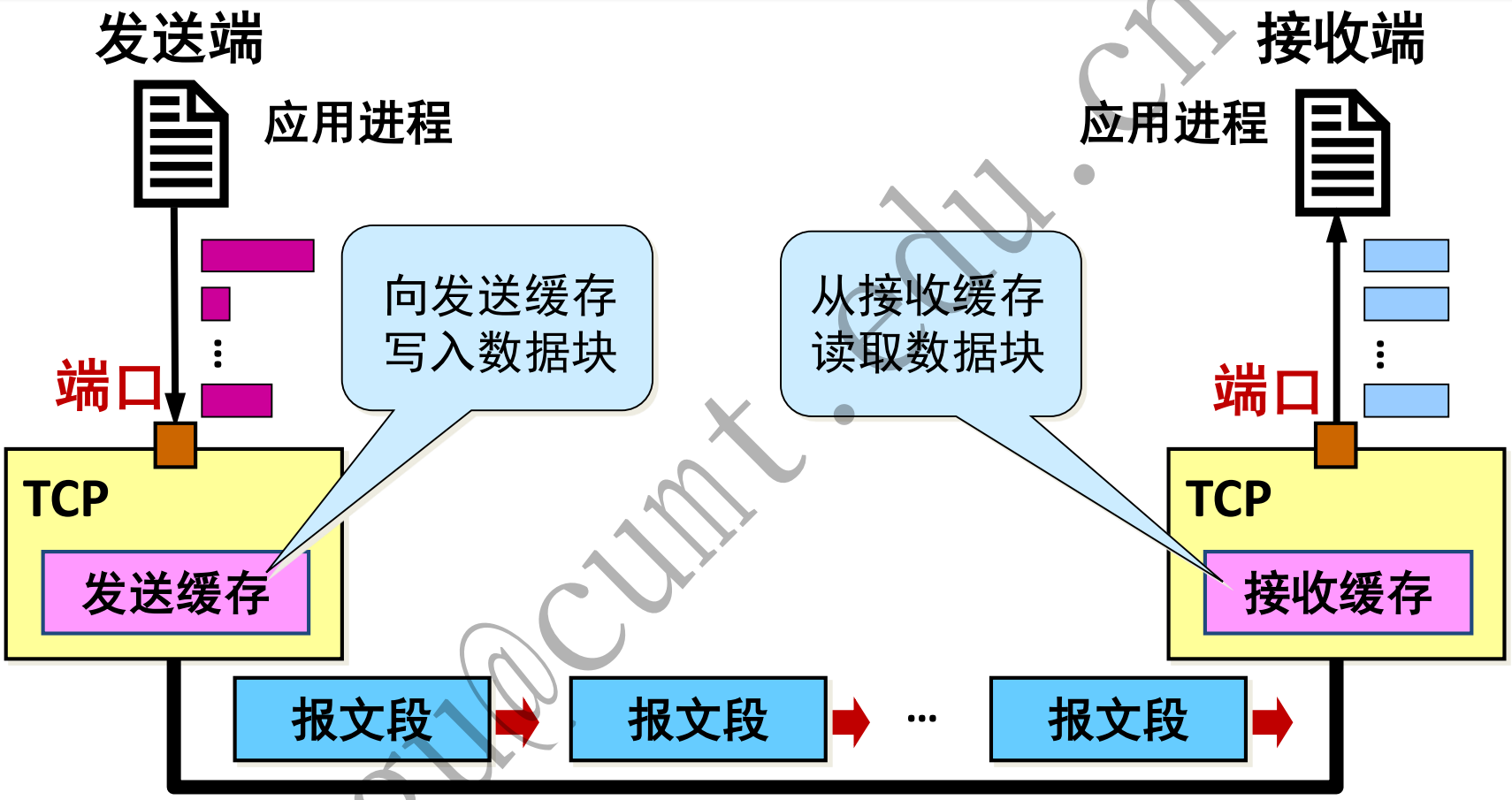
- TCP 中的“流”(stream)指的是流入或流出进程的字节序列。
- “面向字节流”的含义是：虽然应用程序和 TCP 的交互是一次一个数据块，但 TCP 把应用程序交下来的数据看成仅仅是一连串无结构的字节流。
- TCP 不保证接收方应用程序所收到的数据块和发送方应用程序所发出的数据块具有对应大小的关系。
- 但接收方应用程序收到的字节流必须和发送方应用程序发出的字节流完全一样。

面向字节流的概念



TCP 连接是一条虚连接而不是一条真正的物理连接。

TCP 报文段



- TCP 不关心应用进程一次把多长的报文发送到 TCP 缓存。
  - ✓ UDP 发送的报文长度是应用进程给出的。
- TCP 对连续的字节流进行分段，形成 TCP 报文段。

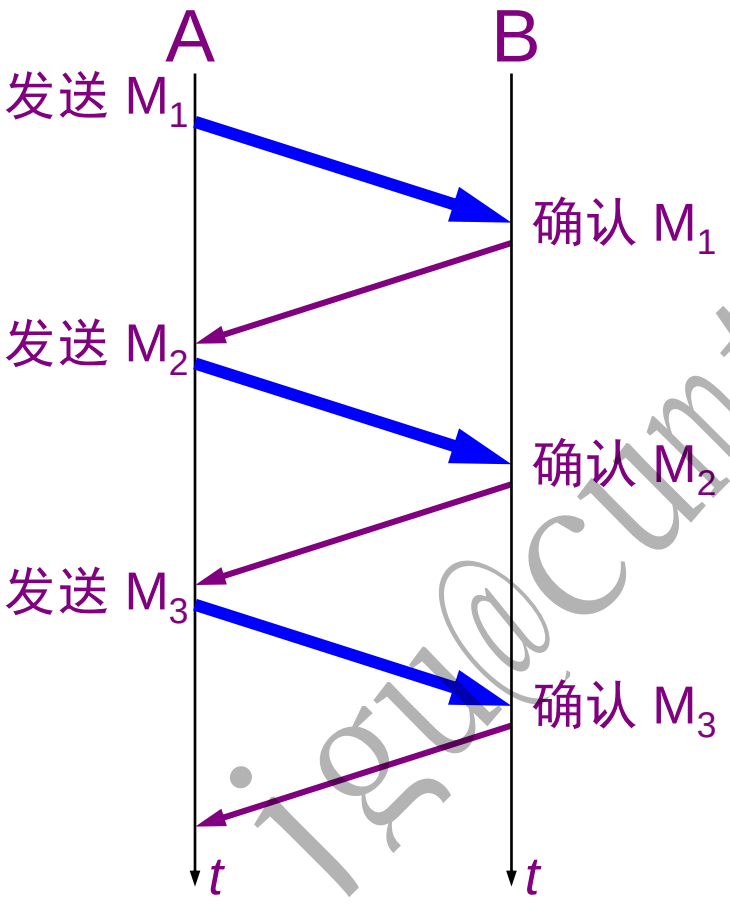
注 意

- TCP 连接是一条虚连接而不是一条真正的物理连接。
- TCP 对应用进程一次把多长的报文发送到TCP的缓存中是不关心的。
- TCP 根据对方给出的窗口值和当前网络拥塞的程度来决定一个报文段应包含多少个字节（UDP 发送的报文长度是应用进程给出的）。
- TCP 可把太长的数据块划分短一些再传送。
- TCP 也可等待积累有足够多的字节后再构成报文段发送出去。

Q9：如何实现确认重传？

- 早期在链路层传输数据时采用的是一种称为“停止等待”的简单协议。
- “停止等待”就是每发送完一个分组就停止发送，等待对方的确认，在收到确认后再发送下一个分组。
- 全双工通信的双方既是发送方也是接收方。
- 为了讨论问题的方便，仅考虑 A 发送数据而 B 接收数据并发送确认。因此 A 叫做发送方，而 B 叫做接收方。

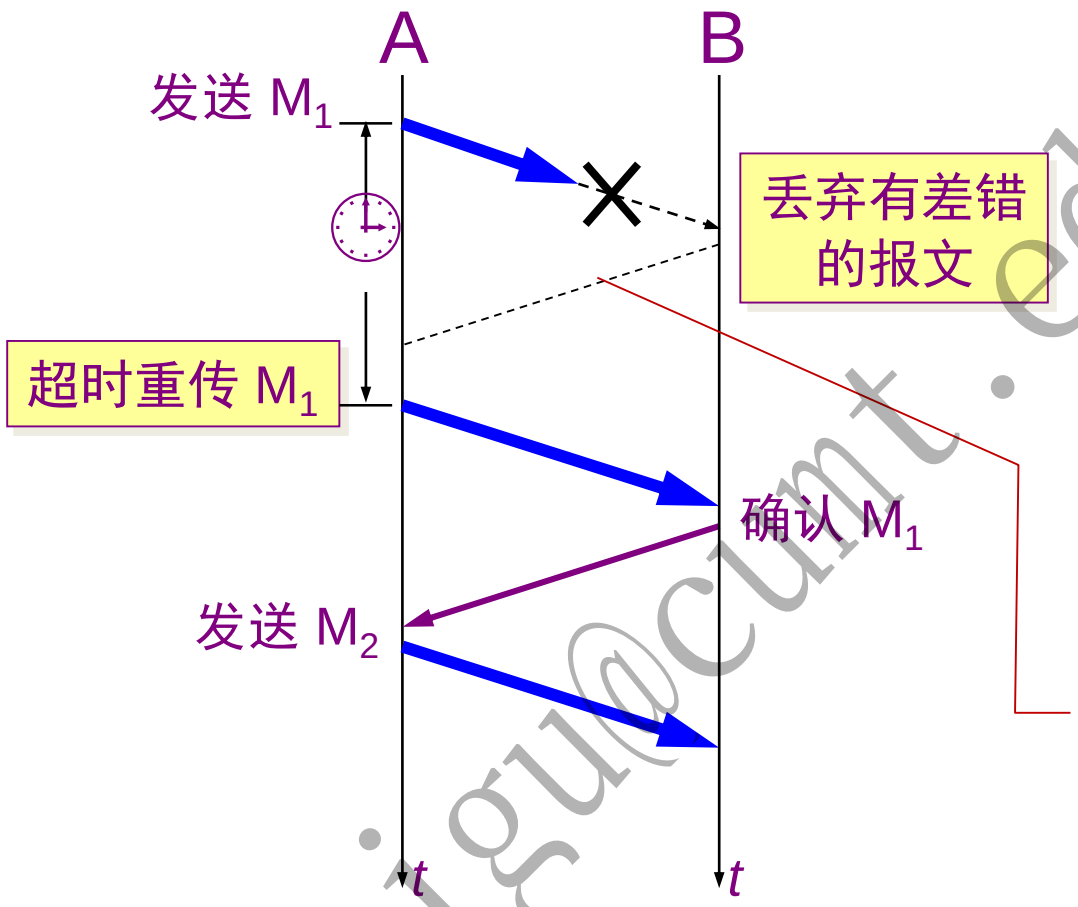
停止等待协议



(a) 无差错情况

- 分组和确认分组都必须进行编号。

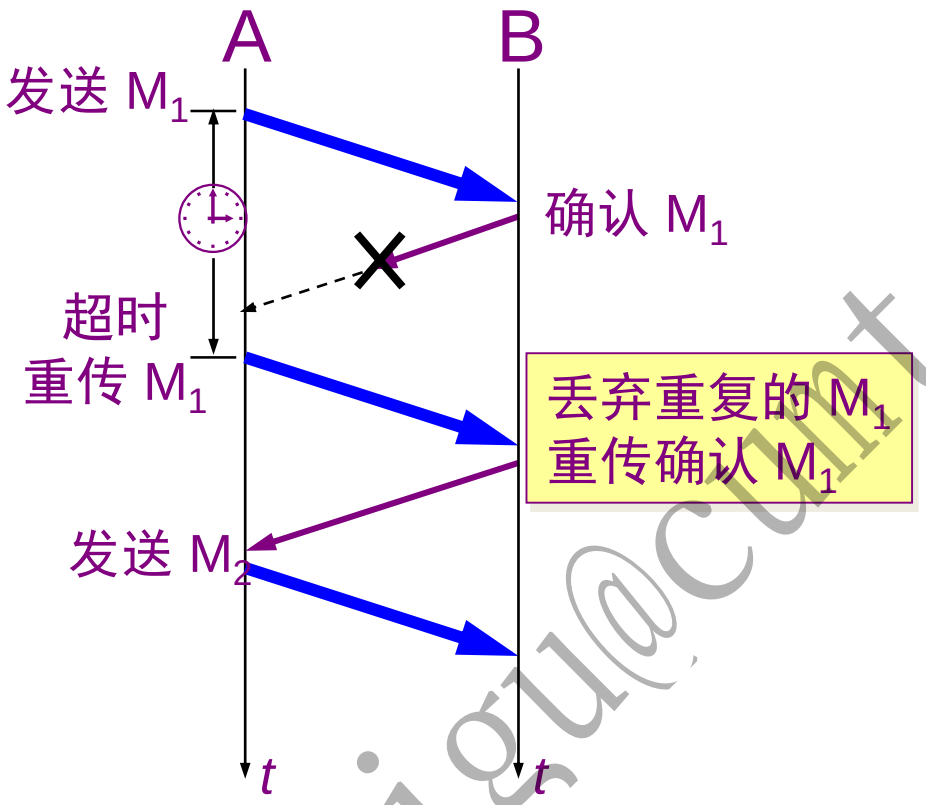
收到确认报文超时，启动重传



(b) 出现差错，超时重传

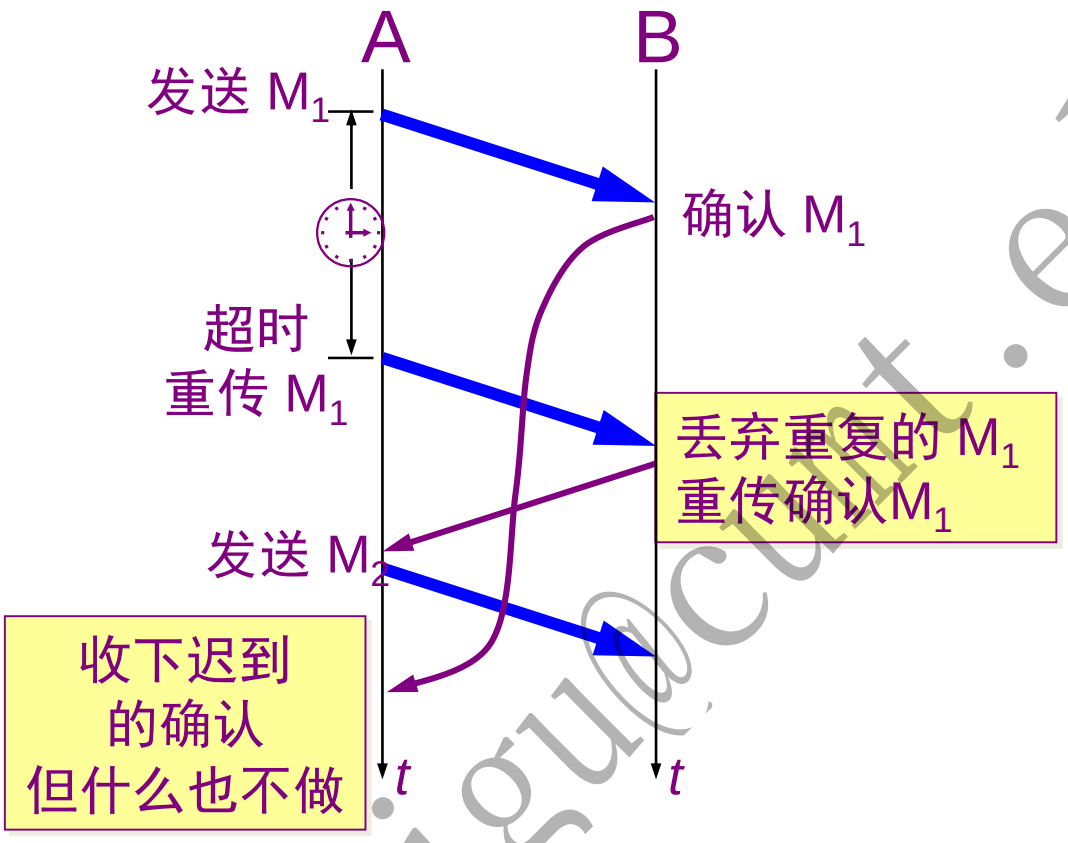
- 分组和确认分组都必须进行编号。
- 在发送完一个分组后，必须暂时保留已发送的分组的副本。
- 超时计时器的重传时间应当比数据在分组传输的平均往返时间(RTT)更长一些。

确认报文丢失



(c) 确认报文丢失

确认报文迟到



(d) 确认报文迟到

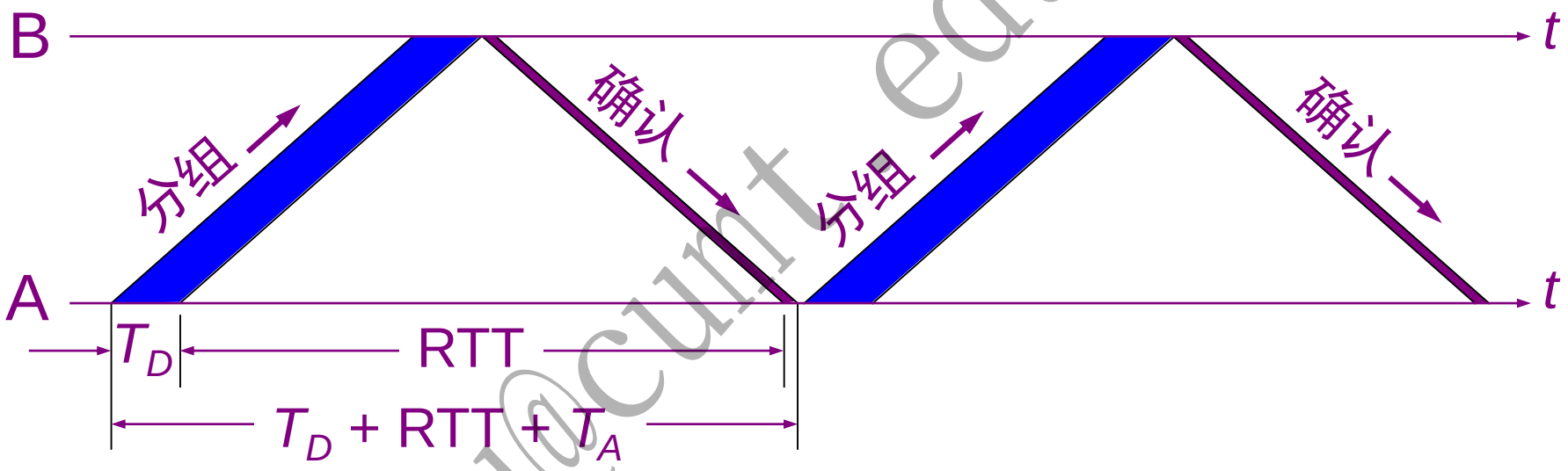
- 通常A最终总是可以收到对所有发出的分组的确认。
- 如果A不断重传分组但总是收不到确认，就说明通信线路太差，不能进行通信。

自动重传请求ARQ

- 使用上述的确认和重传机制，就可以在不可靠的传输网络上实现可靠的通信。
- 这种可靠传输协议常称为自动重传请求ARQ (Automatic Repeat reQuest)。
  - ARQ 表明重传的请求是自动进行的。
  - 发送方在约定时间内收不到确认报文，就会自动开始重传。
  - 接收方不需要请求发送方重传某个出错的分组 。

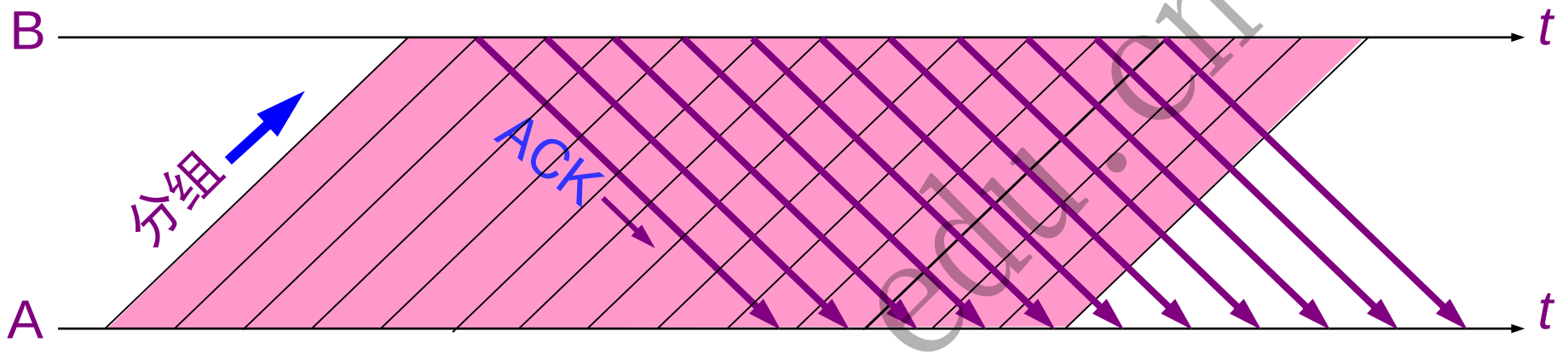
Q10：如何实现高效的确认重传？

- 停止等待协议的优点是简单，但缺点是信道利用率U太低。



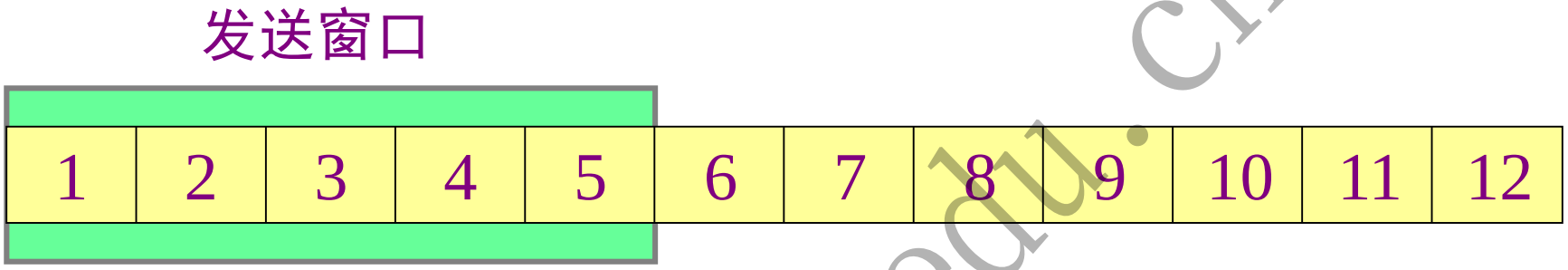
$$U = \frac{T_D}{T_D + \mathrm{RTT} + T_A}$$

流水线传输可提高信道利用率

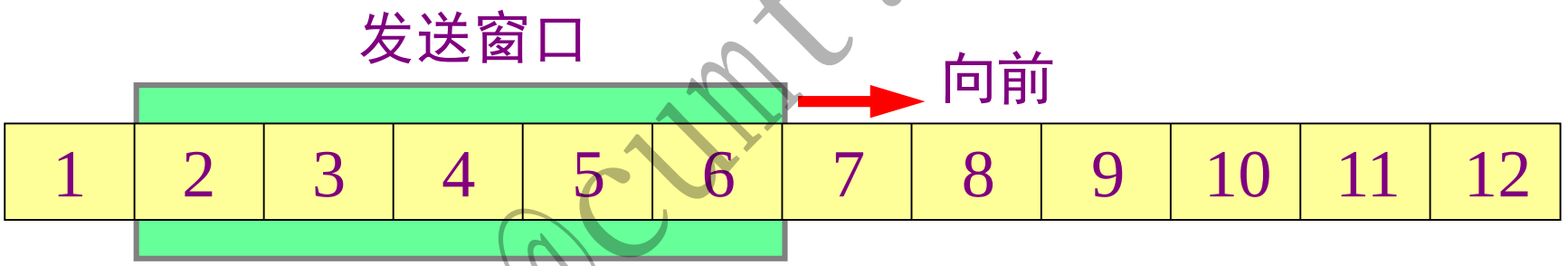


- 发送方可连续发送多个分组，不必每发完一个分组就停顿下来等待对方的确认。这样可使信道上一直有数据不间断地传送。
- 由于信道上一直有数据不间断地传送，这种传输方式可获得很高的信道利用率。
- 当使用流水线传输时，就要使用滑动窗口协议和连续ARQ协议。

滑动窗口协议



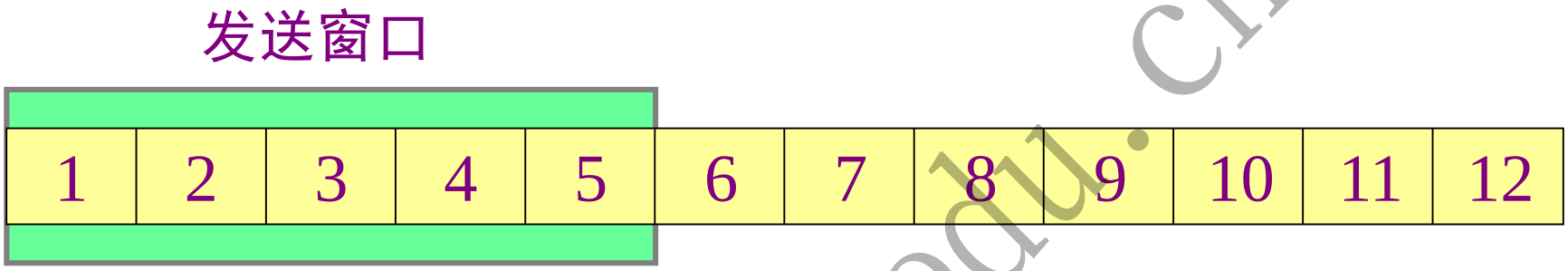
(a) 发送方维持发送窗口（发送窗口是 5）



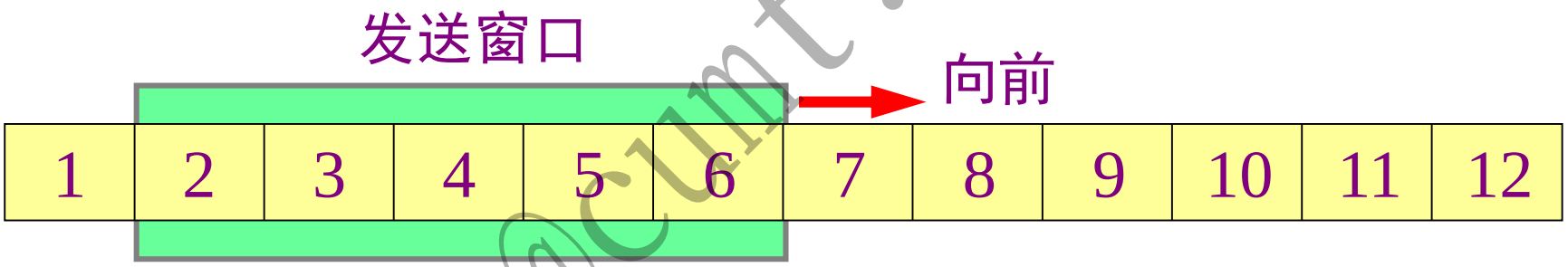
(b) 收到一个确认后发送窗口向前滑动

发送方负责维持发送窗口，位于发送窗口内的分组都可连续发送出去，而不需要等待对方的确认。这样，信道利用率就提高了。

连续ARQ协议



(a) 发送方维持发送窗口（发送窗口是 5）



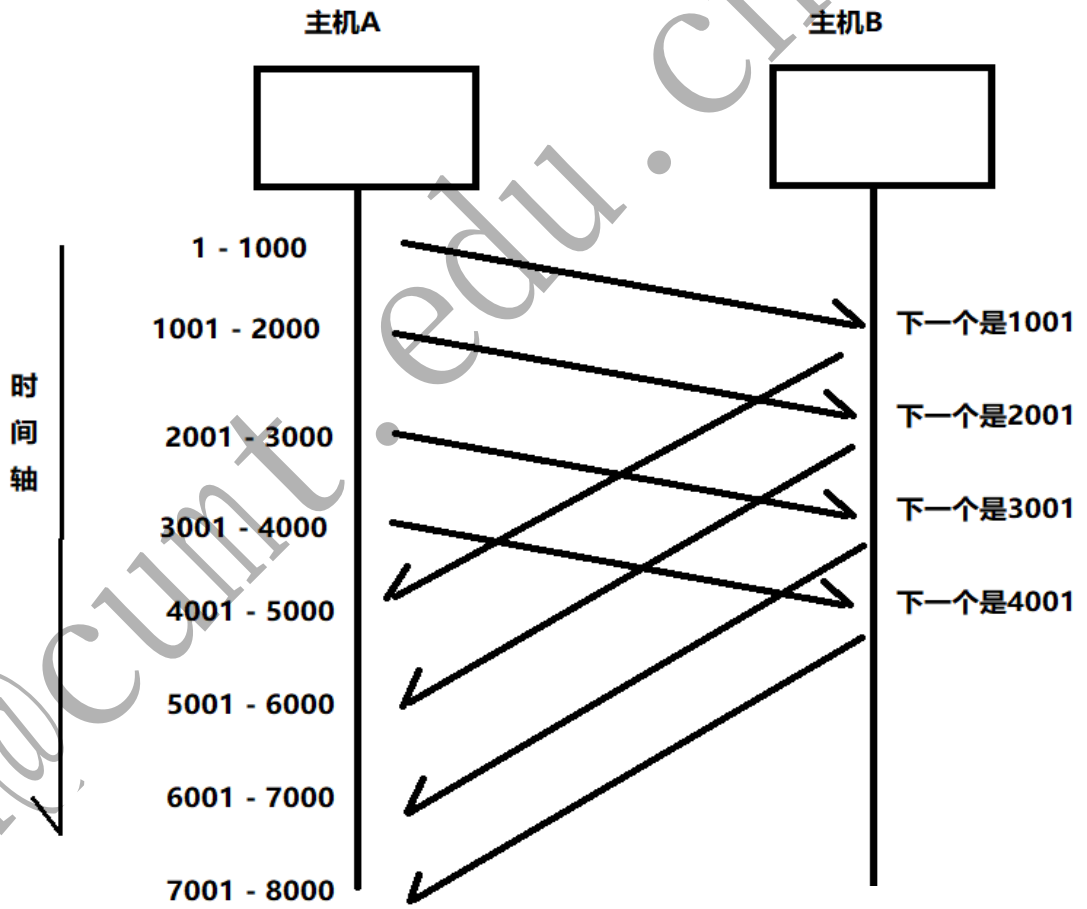
(b) 收到一个确认后发送窗口向前滑动

- 连续 ARQ 协议规定，发送方每收到一个确认，就把发送窗口向前滑动一个分组的位置。
- 如果已经发送了前5个分组，现在就可以发送第6个分组了。

累积确认

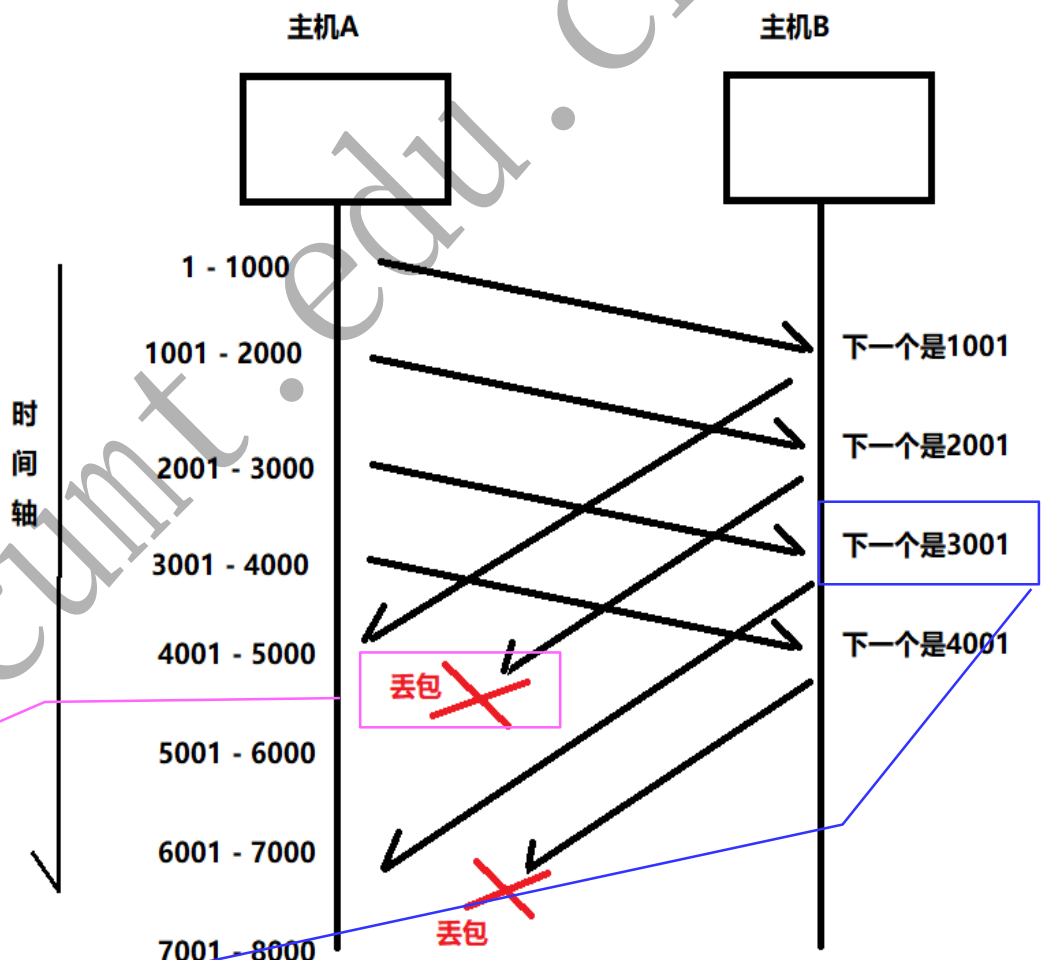
- 接收方一般采用累积确认的方式。
  - 即不必对收到的分组逐个发送确认，而是对按序到达的最后一个分组发送确认。
  - 这样就表示：到这个分组为止的所有分组都已正确收到了。
- 接收方可以在合适的时候发送确认，也可以在自己有数据要发送时把确认信息顺便捎带上。
- 捎带确认实际上并不经常发生，因为大多数应用程序很少同时在两个方向上发送数据。

- 累积确认可以有机会减少确认报文段的发送数量，但是到底能减少多少并不确定，而且会有增大往返时延RTT的风险。
- 所以，接收方不应过分推迟发送确认，否则会导致发送方不必要的重传，反而浪费了网络资源。

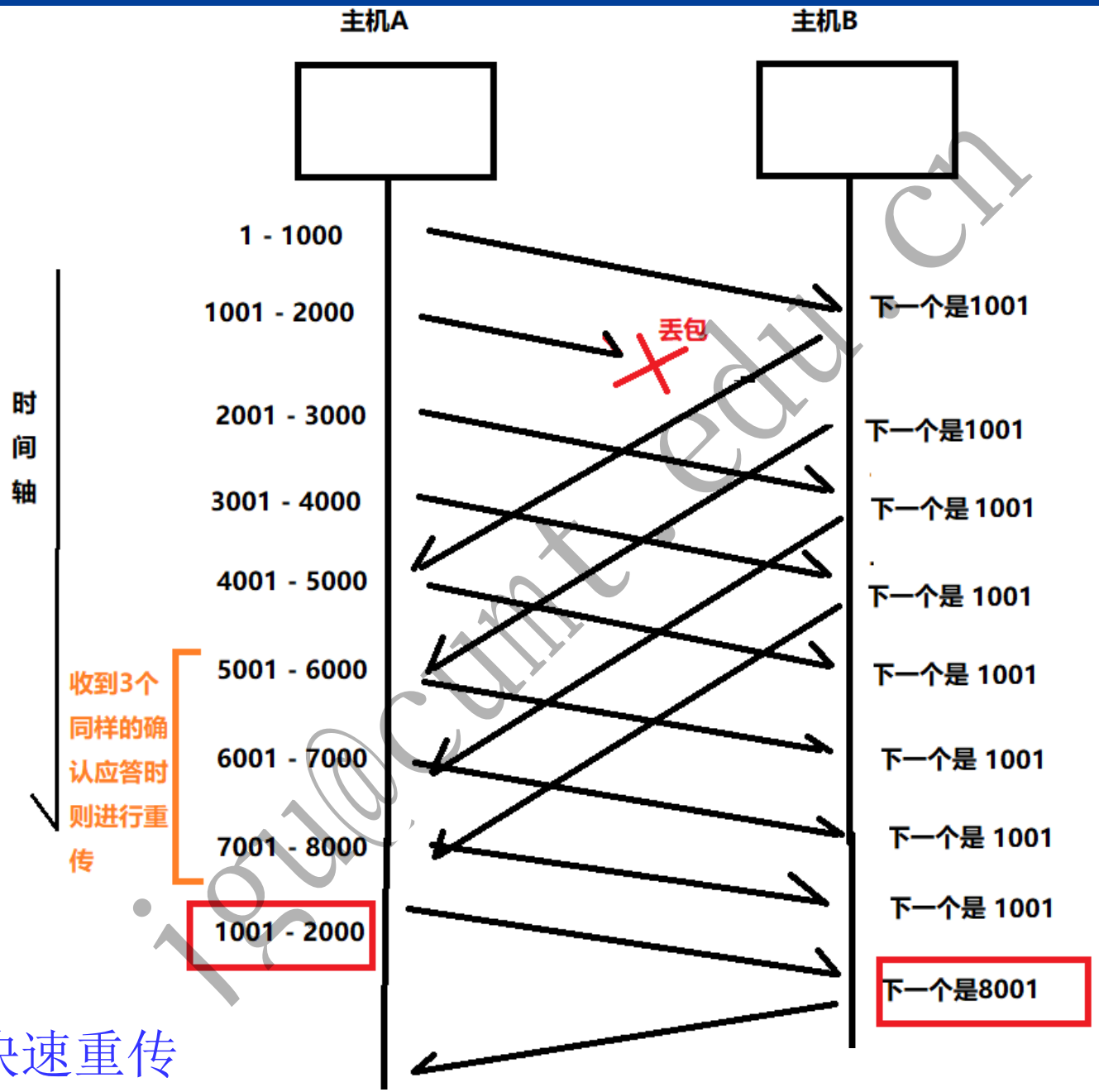


TCP标准规定，确认推迟的时间不应超过0.5秒。若收到一连串具有最大长度的报文段，则必须每隔一个报文段就发送一个确认。

- 累积确认机制下，接收方发出的确认号=N，表明：到序号N-1为止的所有数据都已正确收到。
- 即使确认丢失也不必重传



后一个累计确认挽回了前一个确认丢包的负面影响



快速重传

Go-back-N（回退 N）

- 累积确认机制下，不能向发送方反映出接收方已经正确收到的所有分组的信息，可能导致Go-back-N问题。
- 如果发送方发送了前 5 个分组，而中间的第 3 个分组丢失了。这时接收方只能对前两个分组发出确认。发送方无法知道后面三个分组的下落，而只好把后面的三个分组都再重传一次。
- 这就叫做 Go-back-N（回退 N），表示需要再退回来重传已发送过的 N 个分组。
- 可见，当通信线路质量不好时，连续 ARQ 协议会带来负面的影响。

Q11：TCP何时发送确认？

- TCP协议规定在接受到数据段时需要向对方发送一个确认，但如果只是单纯的发送一个确认，代价会比较高（20字节的IP首部，20字节的TCP首部），最好能附带响应数据一起发送给对方，称为捎带确认。
- 接收方在收到数据包的时候，会检查是否需要发送ACK；如果需要的话，可以选择是快速确认（Quick ACK）还是延迟确认（Delay ACK）。
- 在无法使用快速确认的条件下，就会使用延迟确认。

快速确认（Quick ACK）

- 如果设置了快速确认标志，且快速确认模式中可以发送的ACK数量不为0，就判断连接处于快速确认模式中，允许立即发送ACK。
- 在快速确认模式中，可以发送的ACK数量是有限制的，所以进入快速确认模式时，需要设置可以快速发送的ACK数量，一般允许快速确认半个接收窗口的数据量，但最多不能超过16个，最少为2个。

延迟确认（Delay ACK）

- 在延迟确认下，ACK是可以合并的，即：如果连续收到两个TCP报文段，并不一定需要ACK两次，只要回复最后的ACK就可以了，从而降低网络流量。
- 如果接收方有数据要发送，那么就会在发送数据的TCP报文段里，捎带上ACK信息。这样做，可以避免大量的ACK以一个单独的TCP包发送，减少了网络流量。

优点：减少确认报文段的个数，提高发送效率

缺点：过多的Delay会拉长RTT

延迟确认的规定

TCP对何时延迟发送ACK给对方有以下规定：

- 1、当有响应数据要发送的时候，ACK会随响应数据立即发送给对方，即捎带确认。
- 2、如果没有响应数据，ACK就会有一个延迟，以等待是否有响应数据一块发送，但是这个延迟一般在40ms~500ms之间，一般为200ms。

✓ 这个200ms并非收到数据后需要延迟的时间，而是操作系统内核会启动一个固定的定时器，每隔200ms会来检查是否需要发送ACK包。

延迟确认的规定

- ✓ 比如，定时器在0ms启动，200ms到期。
  - 180ms的时候，一个TCP报文段到达，200ms的时候即使没有响应数据，ACK仍然会被发送，而不是380ms时发送。这个时候就延迟了20ms。

- 3、如果在等待发送ACK期间，对方的第二个数据段又到达了，这时要立即发送ACK。
- ✓ 但是，如果对方的三个数据段相继到达，那么第二个数据段到达时ACK立即发送，但第三个数据段到达时是否立即发送确认，则取决于上面两条。

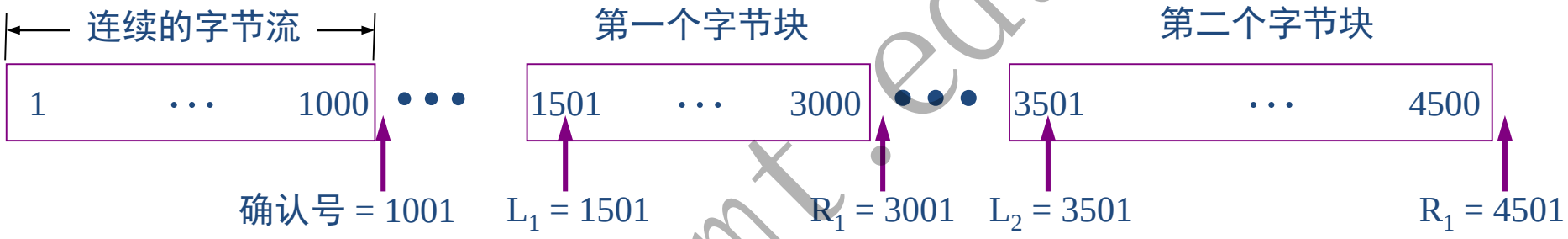
捎带确认实际上并不经常发生

选择确认 SACK (Selective ACK)

- 新问题的提出：
  - 若收到的报文段无差错，只是未按序号，中间还缺少一些序号的数据，那么能否设法只传送缺少的数据而不重传已经正确到达接收方的数据？
- 答案是可以的。选择确认 SACK (Selective ACK) 就是一种可行的处理方法。

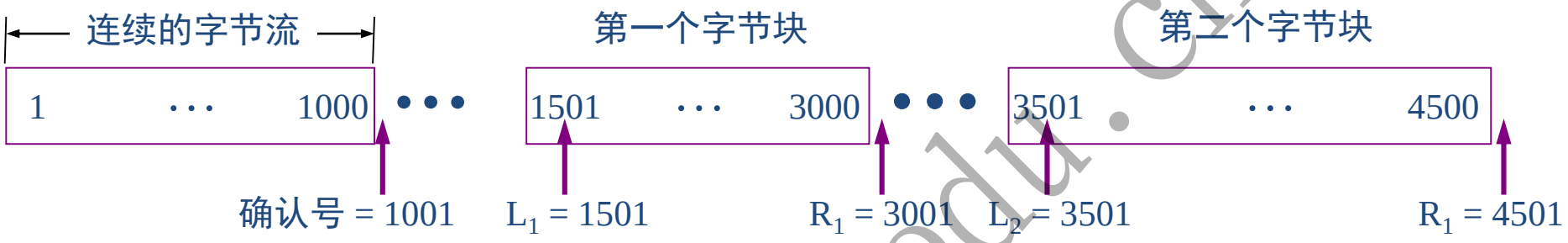
接收到的字节流序号不连续

TCP 的接收方在接收对方发送过来的数据字节流的序号不连续，结果就形成了一些不连续的字节块。



- 和前后字节不连续的每一个字节块都有两个边界：左边界和右边界。图中用四个指针标记这些边界。
- 第一个字节块的左边界 L_1 = 1501，但右边界 R_1 = 3001。
- 左边界指出字节块的第一个字节的序号，但右边界减 1 才是字节块中的最后一个序号。
- 第二个字节块的左边界 L_2 = 3501，而右边界 R_2 = 4501。

选择确认 SACK 扩展选项



- 接收方收到了和前面的字节流不连续的两个字节块。
- 如果这些字节的序号都在接收窗口之内，那么接收方就先收下这些数据，但要把这些信息准确地告诉发送方，使发送方不要再重复发送这些已收到的数据。
- TCP的首部没有哪个字段能够提供上述这些字节块的边界信息。RFC 2018规定，如果要使用SACK，那么在建立TCP连接时，就要在TCP首部的选项中加上“允许SACK”的选项，而双方必须事先商定好。

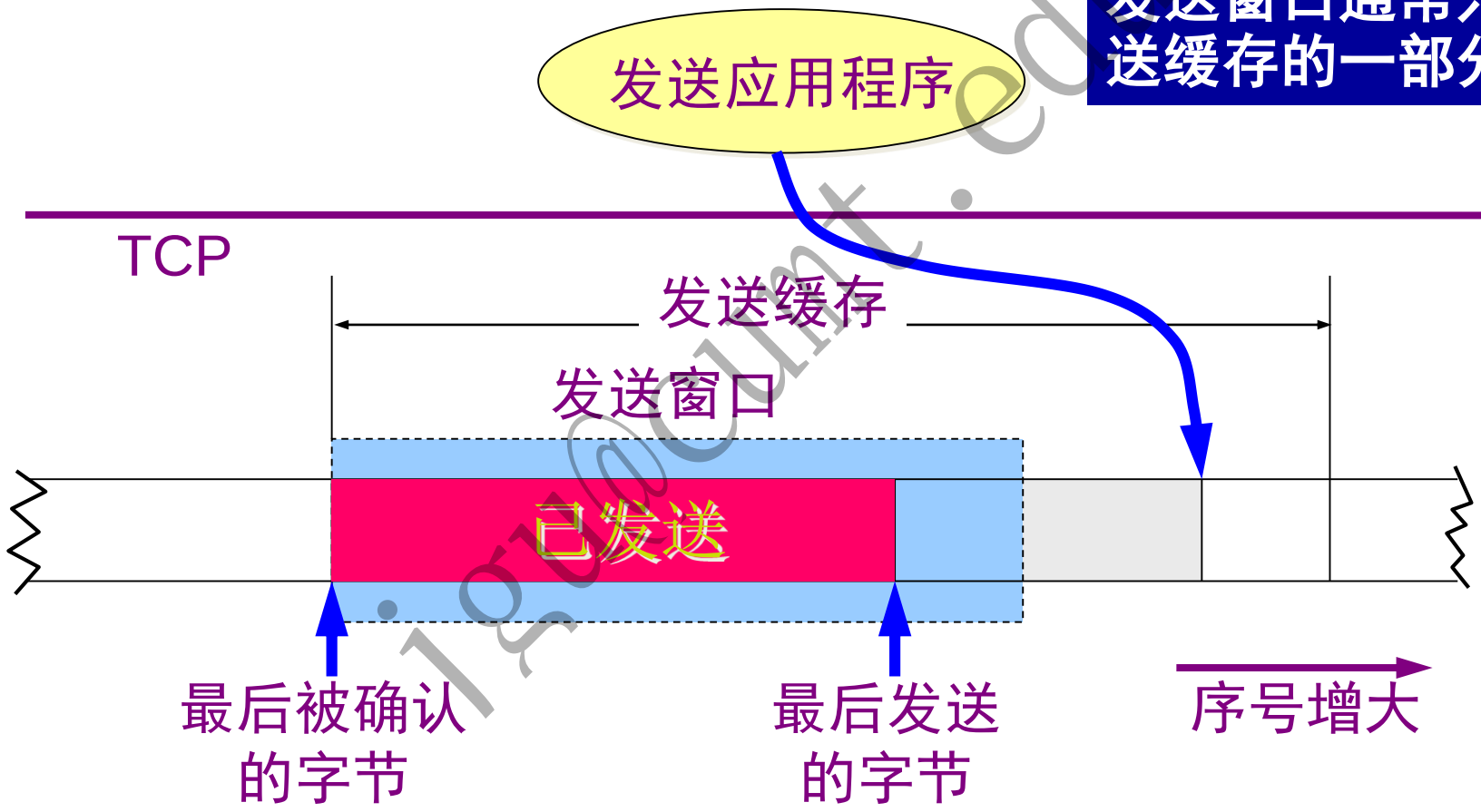
Q12：发送缓存和接收缓存的作用？

- 对于发送端：
  - 已经发送但未被确认的数据应该先被保存下来，以便收不到确认时能够被重传。
  - 需要发送缓存
- 对于接收端：
  - 已经接收但序号不全的数据应该先被保存下来，以便缺少的数据到来后可以一起交付给主机。
  - 需要接收缓存

发送缓存和发送窗口

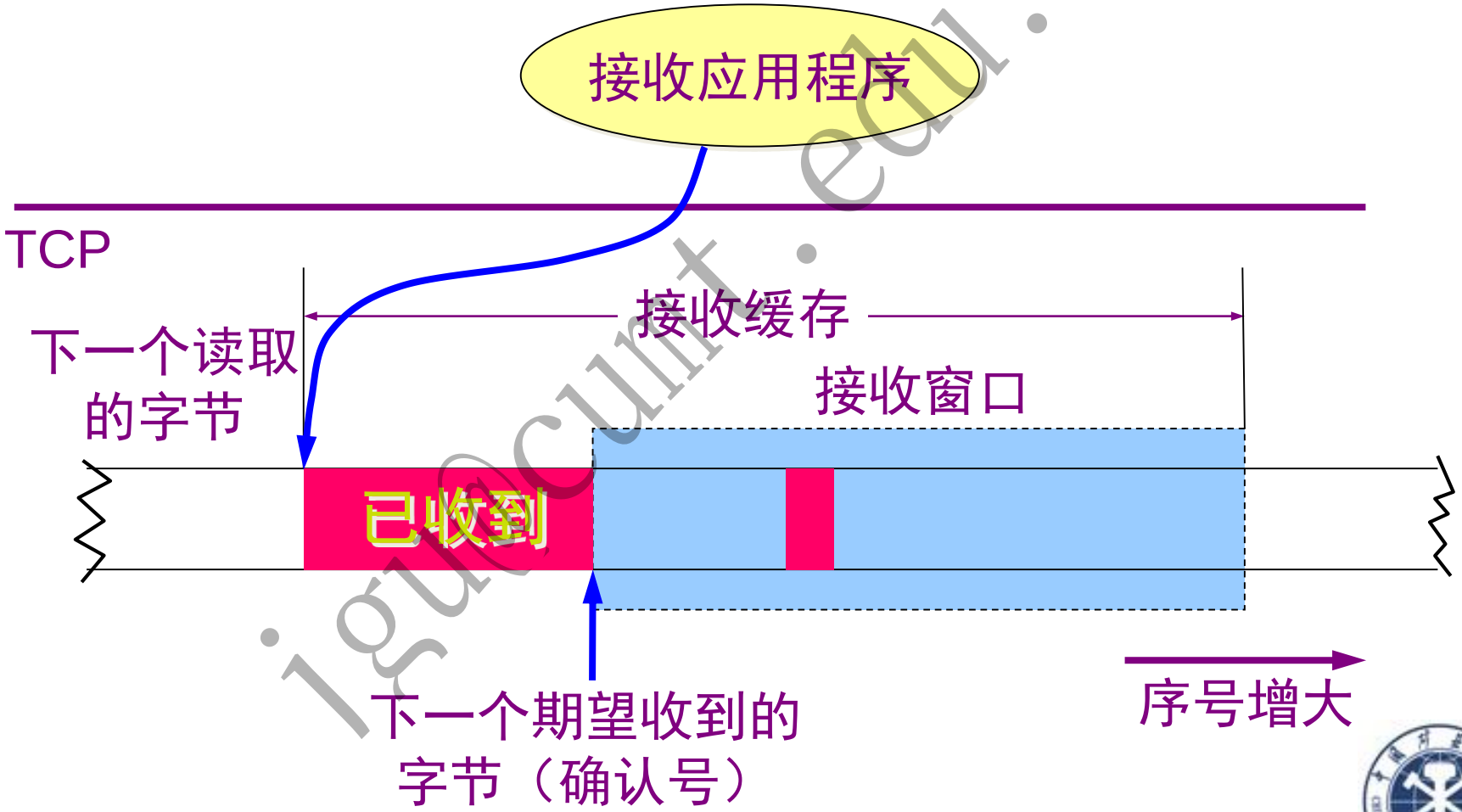
发送方的应用进程把字节流写入 TCP 的发送缓存。

发送窗口通常只是发送缓存的一部分。



接收缓存和接收窗口

接收方的应用进程从 TCP 的接收缓存中读取字节流。

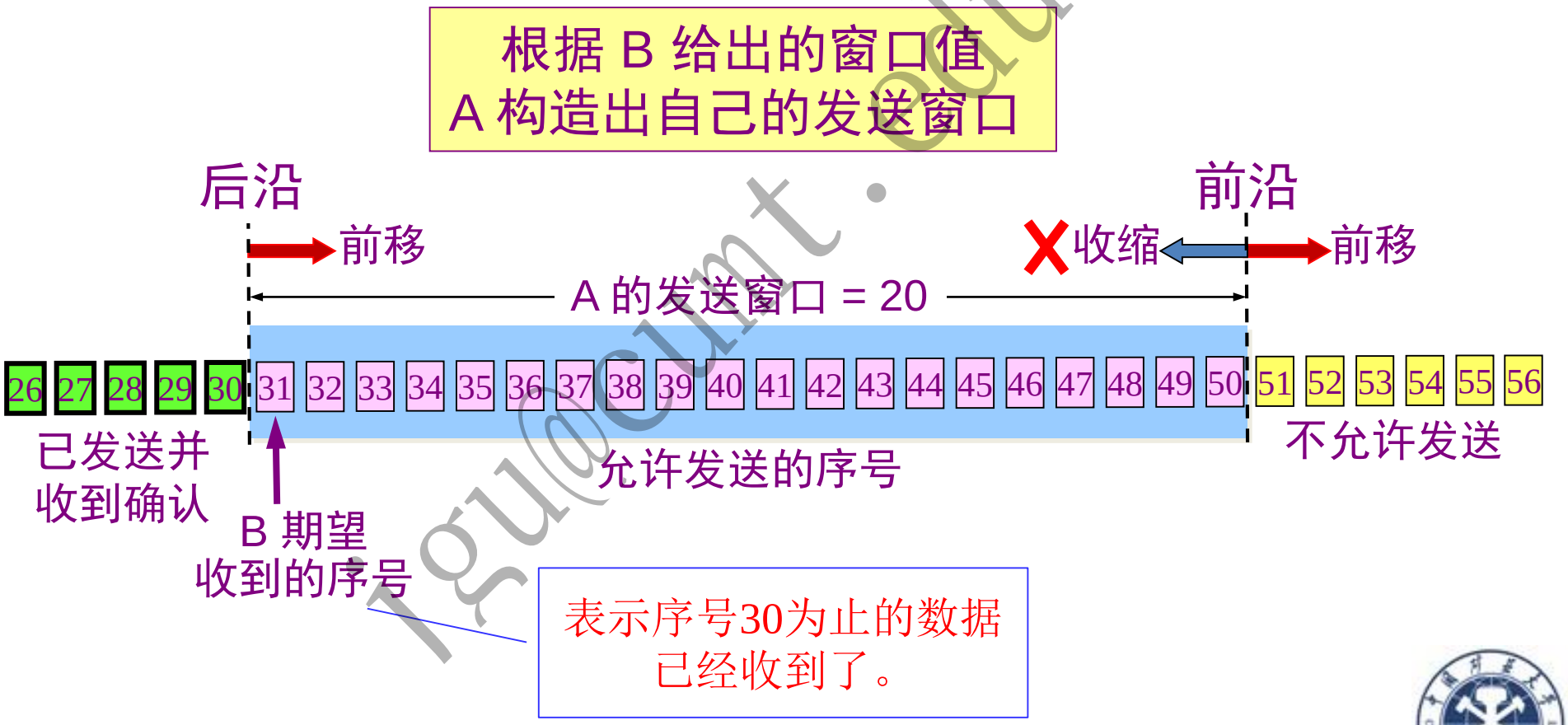


发送缓存与接收缓存的作用

- 发送缓存用来暂时存放：
  - 发送应用程序传送给发送方 TCP 准备发送的数据；
  - TCP 已发送出但尚未收到确认的数据。
- 接收缓存用来暂时存放：
  - 按序到达的、但尚未被接收应用程序读取的数据；
  - 不按序到达的数据。

Q13：如何协调发送和接收窗口？

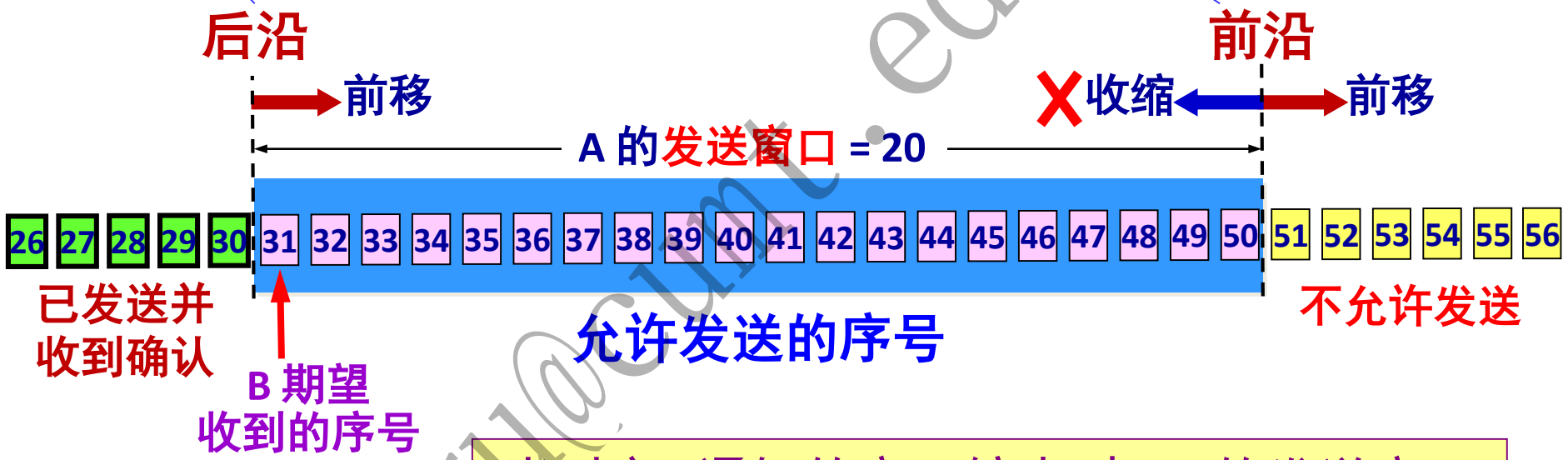
TCP的滑动窗口以字节为单位。



- 发送窗口表示：在没有收到 B 的确认的情况下，A 可以连续把窗口内的数据都发送出去。
- 凡是已经发送过的数据，在未收到确认之前都必须暂时保留在缓存中，以便在超时重传时使用。
- 发送窗口里面的序号表示允许发送的序号。
- 显然，窗口越大，发送方就可以在收到对方确认之前连续发送更多的数据，因而可能获得更高的传输效率。

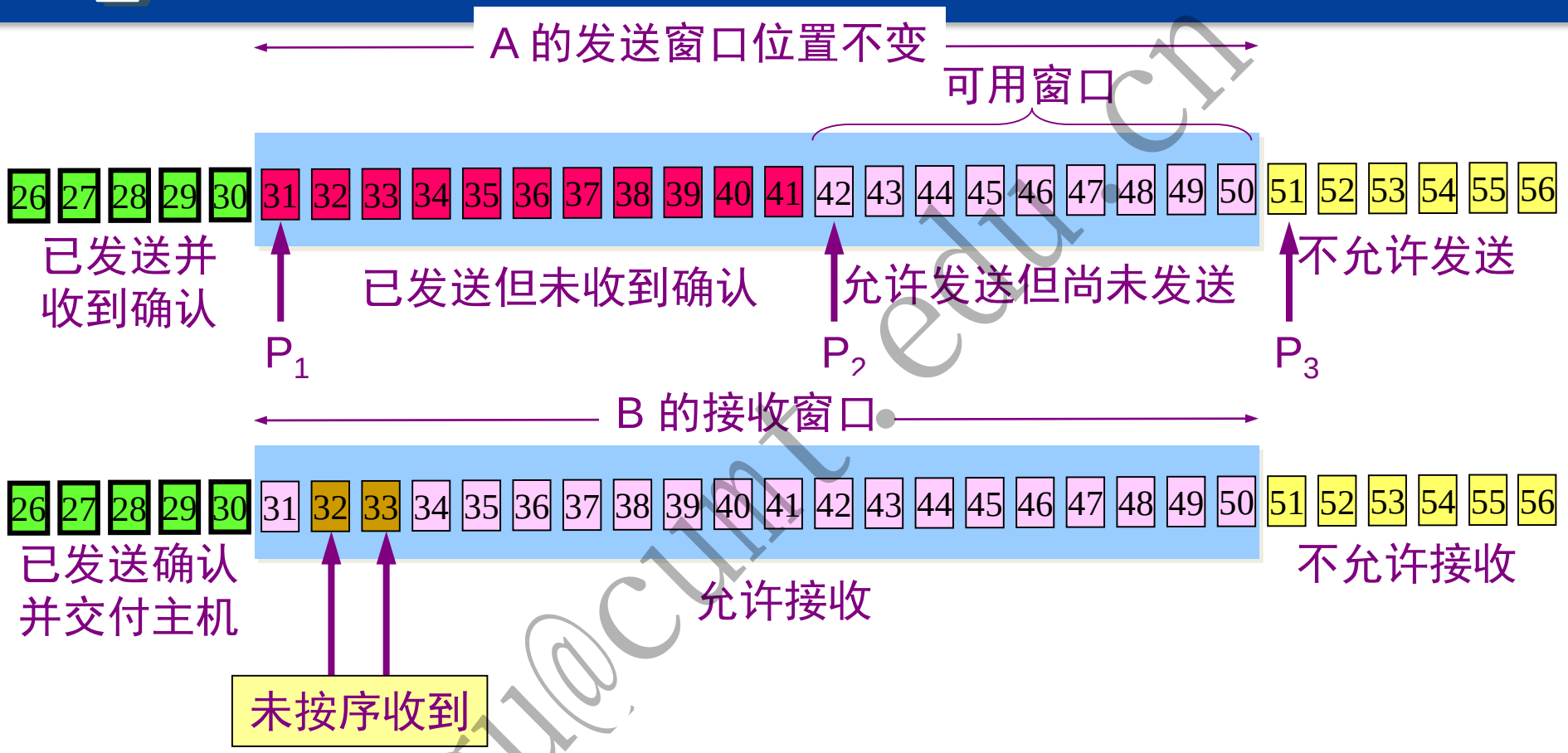
后沿要么不动（没有收到新的确认），要么前移（收到了新的确认），不可能向后移动，因为不能撤销掉已收到的确认。

前沿要么前移，要么不动（没有收到确认且对方通知的窗口大小也不变，或者收到确认但对方通知的窗口缩小了）



当对方B通知的窗口缩小时，A的发送窗口前沿有可能向后收缩。但TCP 标准强烈不赞成发送窗口前沿向后收缩

A 发送了 11 个字节的数据

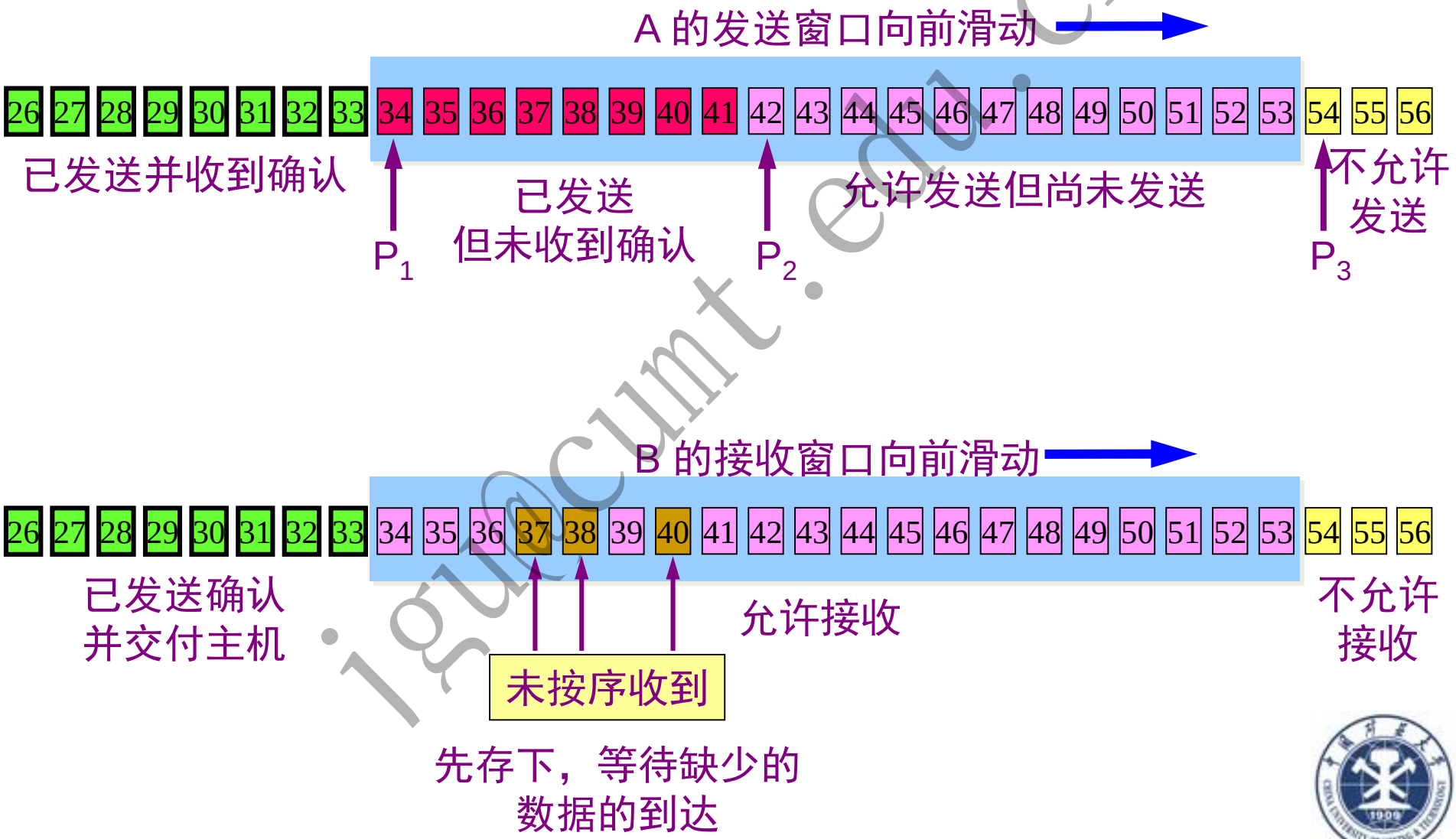


$P_3 - P_1$ = A 的发送窗口（又称为通知窗口）

$P_2 - P_1$ = 已发送但尚未收到确认的字节数

$P_3 - P_2$ = 允许发送但尚未发送的字节数（又称为可用窗口）

A 收到新的确认号，发送窗口向前滑动



A 的发送窗口内的序号都已用完，
但还没有再收到确认，必须停止发送。

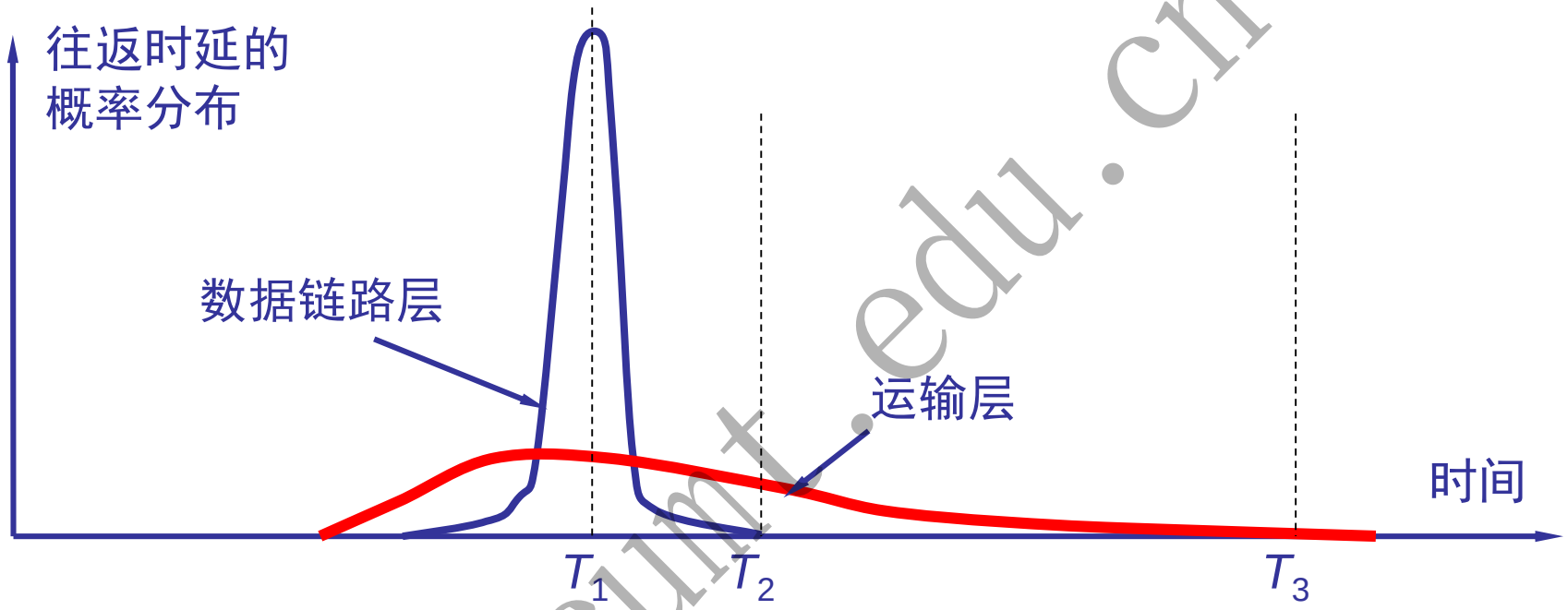
需要强调四点

- 第一，A 的发送窗口并不总是和 B 的接收窗口一样大（因为通过网络传送窗口值需要经历一定的时间滞后）。有时，A还可能根据网络当时的拥塞情况适当减小自己的发送窗口数值。
- 第二，TCP 标准没有规定对于不按序到达的数据应如何处理。通常是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付上层的应用进程。
- 第三，TCP 要求接收方必须有累积确认的功能，这样可以减小传输开销。
- 第四，TCP的通信是全双工通信。通信中的每一方都在发送和接收报文端，因此每一方都有自己的发送窗口和接收窗口。

Q14：TCP报文段重传时机的选择？

- 重传机制是 TCP 中最重要和最复杂的问题之一。
- TCP 每发送一个报文段，就对这个报文段设置一次计时器。
- 只要计时器设置的重传时间到但还没有收到确认，就要重传这一报文段。
- 由于 TCP 的下层是一个互连网环境，IP 数据报所选择的路由变化很大。因而运输层的往返时延的方差也很大。

往返时延的方差很大



- 如果把超时重传时间设置得太短，就会引起很多报文段的不必要的重传，使网络负荷增大。但若把超时重传时间设置得过长，则又使网络的空闲时间增大，降低了传输效率。
- TCP 采用了一种自适应算法，它记录一个报文段发出的时间，以及收到相应的确认的时间。这两个时间之差就是报文段的往返时间 RTT。

加权平均往返时间RTT_S

- TCP 保留了 RTT 的一个加权平均往返时间 RTT_S，即将各个报文段的往返时延样本加权平均，又称为平滑的往返时间（Smoothed RTT）。
- 第一次测量到 RTT 样本时，RTT_S 值就取为所测量到的 RTT 样本值。以后每测量到一个新的 RTT 样本，就按下式重新计算一次 RTT_S：

新的RTT_S = (1 − α) × (旧的RTT_S)
+ α × (新的RTT样本)

- 式中，$0 \le \alpha < 1$。若 α 很接近于零，表示 RTT 值更新较慢。若选择 α 接近于 1，则表示 RTT 值更新较快。
- RFC 2988 推荐的 α 值为 1/8，即 0.125。
- 用这种方法得出的RTTs就比测量出的RTT值更加平滑。

超时重传时间 RTO (Retransmission Time-Out)

- 计时器的 RTO 应略大于上面得出的 RTTs，RFC 2988建议使用下式计算RTO：

$$RTO = RTTs + 4 \times RTT_D$$

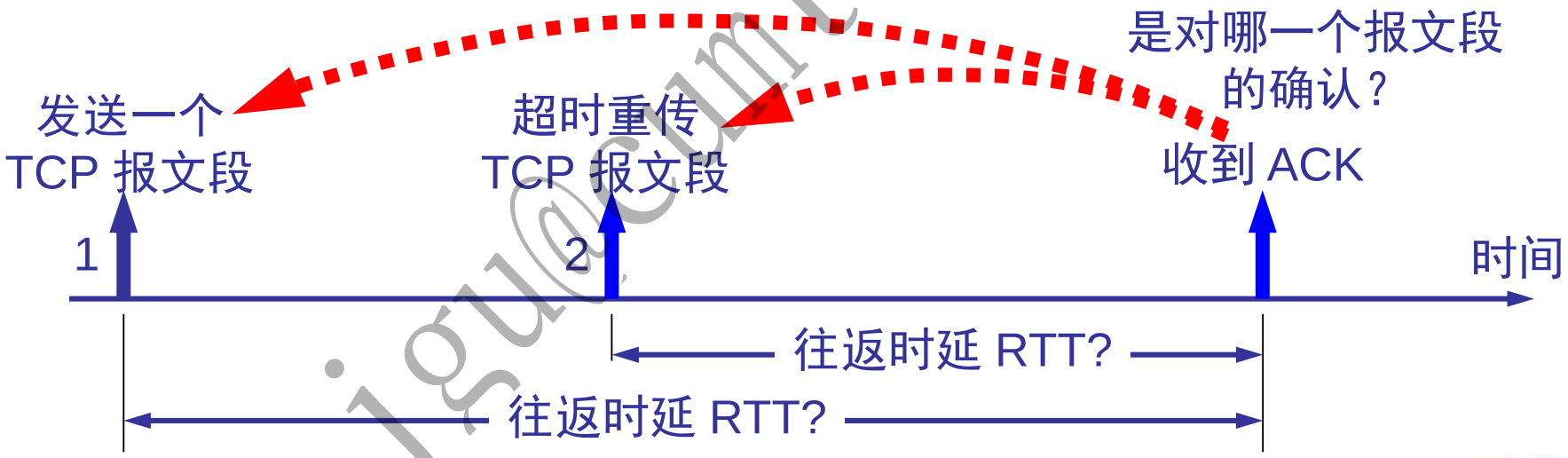
- RTT_D是RTT的偏差的加权平均值，它与RTTs和新的RTT样本之差有关。
- RFC 2988建议，当第一次测量时，RTT_D值取为测量到的RTT样本值的一半。在以后的测量中，使用下式计算加权平均的RTT_D：

$$\text{新的}RTT_D = (1-\beta) \times (\text{旧的}RTT_D) + \beta \times |RTTs - \text{新的RTT样本}|$$

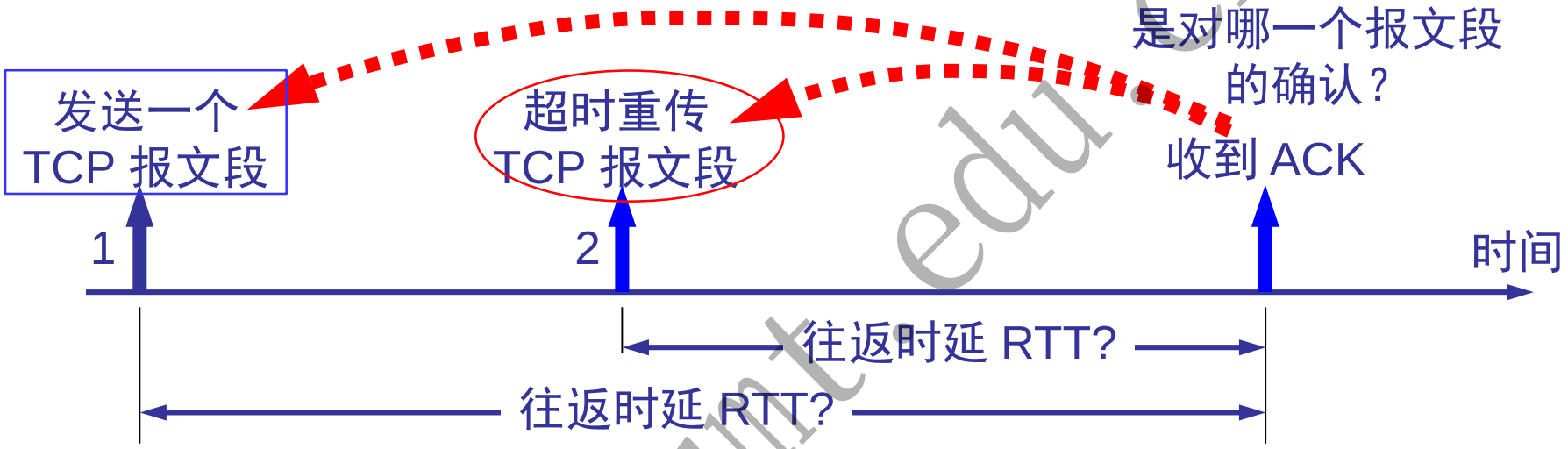
- β 是个大于 1 的系数，它的推荐值是1/4。

往返时间的测量相当复杂

- TCP 报文段 1 没有收到确认。重传（即报文段 2）后，收到了确认报文段 ACK。
- 如何判定此确认报文段是对原来的报文段 1 的确认，还是对重传的报文段 2 的确认？

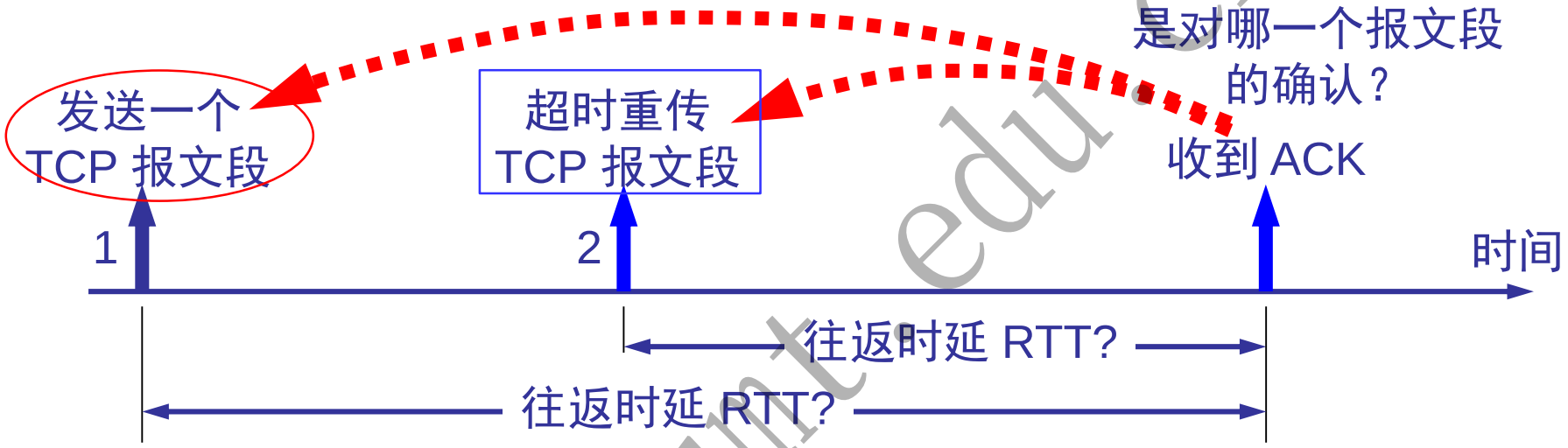


往返时间的测量相当复杂



- 若收到的确认是对重传报文段的确认，但却被源主机当成是对原来的报文段的确认，则这样计算出的RTTs和超时重传时间RTO就会偏大。
- 若后面再发送的报文段又是经过重传后才收到确认报文段，则按此方法得出的超时重传时间RTO就越来越长。

往返时间的测量相当复杂



- 若收到的确认是对原来报文段的确认，但却被当成是对重传报文段的确认，则由此计算出的RTTs和RTO都会偏小。
- 这就必然导致报文段过多地重传。这样就可能使RTO越来越短。

Karn 算法

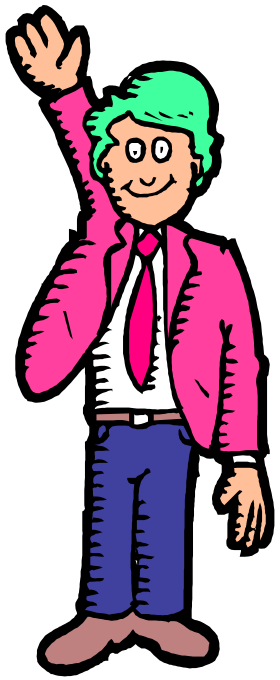
- 在计算平均往返时延 RTTs 时，只要报文段重传了，就不采用其往返时间样本。
- 这样得出的平均往返时延 RTTs 和重传时间RTO就较准确。
- 引起的新问题
  - 设想：报文段的时延突然增大了很多，因此在原来得出的重传时间内，不会收到确认报文段。于是就重传报文段。
  - 但根据Karn算法，不考虑重传的报文段的往返时间样本，这样，超时重传时间就无法更新。

修正的 Karn 算法

- 报文段每重传一次，就将RTO增大一些：

新的RTO = γ × (旧的RTO)

- 系数 γ 的典型值是2 。
- 当不再发生报文段的重传时，才根据报文段的往返时延更新平均往返时延 RTT 和重传时间的数值。
- 实践证明，这种策略较为合理。
- Karn算法能够使运输层区分开有效的和无效的往返时间样本，从而改进了往返时延的估测。



THANK YOU!