第6章 传输层(2)

课程名称：计算机网络

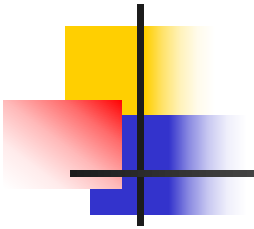
主讲教师：姚烨，朱怡安

课程代码：U10M11016.02

E-MAIL ：yaoye@nwpu.edu.cn

第40-41讲

2021 – 2022 学年第一学期



本节内容提要

6.4 TCP通信三个阶段

6.5 可靠传输工作原理

6.6 TCP 可靠传输的实现

6.7 TCP的流量控制

6.4 TCP通信三个阶段

- TCP协议通信三个阶段：两个端系统之间行为
  - 建立TCP连接
  - 数据通信：维护连接，采用可靠通信机制，确保TCP报文段传输可靠性
  - 释放TCP连接
- 建立连接过程中要解决以下三类问题：
  - 每一方能够确知对方存在；
  - 允许双方协商一些参数（MSS，窗口大小，服务质量等）。
  - 对网络资源（如缓存大小，不同定时器,如重传定时器，保活定时器和坚持定时器等，以及连接状态表等）进行分配并初始化。

6.4 TCP通信三个阶段

- TCP 通信采用客户/服务器方式。
  - 主动发起建立TCP连接的应用进程叫做客户进程(client)，简称客户端
  - 被动等待对方连接建立的进程叫做服务器进程(server)，简称服务器。

- TCP 标准规定1:SYN置1的报文消耗一个序号;
- TCP 标准规定2:单独的确认报文不消耗序号;

用三次握手建立 TCP 连接

连接请求 —— SYN, SEQ = x（MSSX） ——→

←—— SYN, ACK, SEQ = y, ACKn= x + 1（MSSY） ——

反向连接请求
(捎带确认)

确认 —— ACK, SEQ = x + 1, ACKn = y + 1 ——→

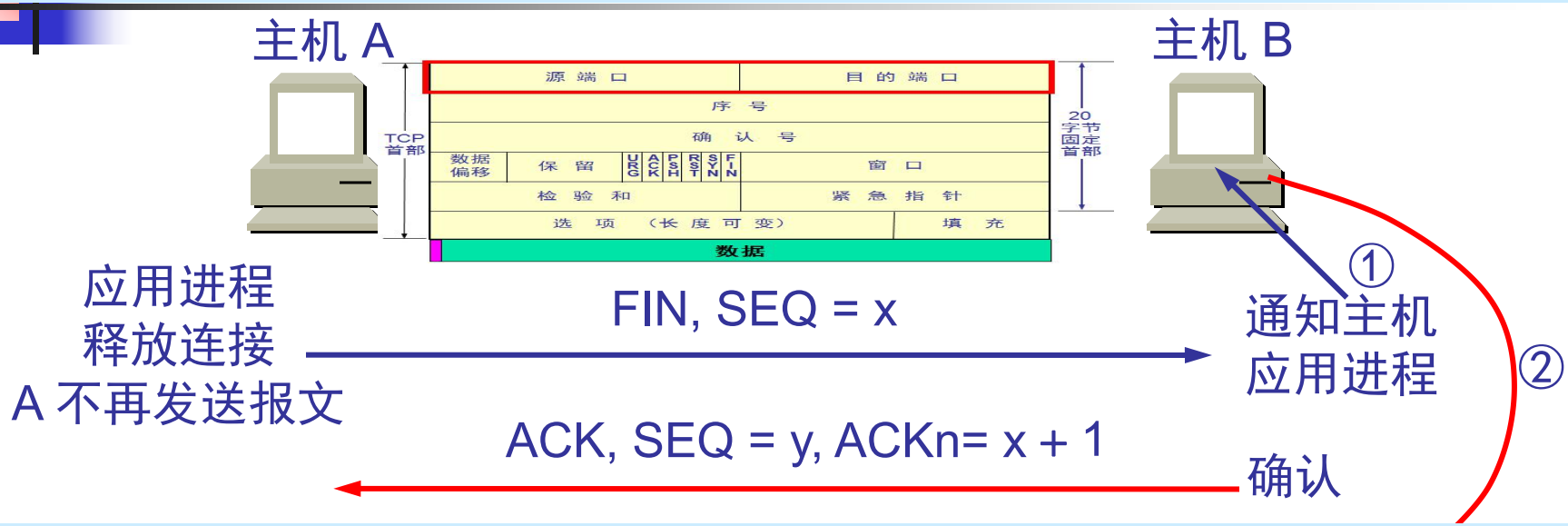
数据 —— ACK, SEQ = x + 1, ACK = y + 1 ——→

←—— ACK, SEQ = y+1, ACK= x + n+1 ——

数据

至此，整个连接已经全部释放。

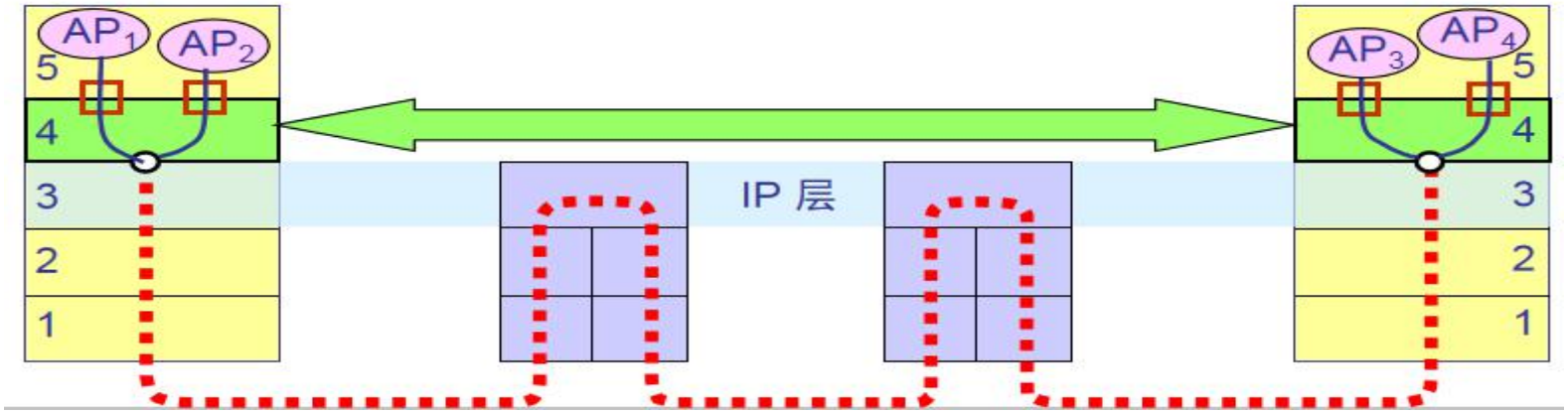
命令:netstat –a,可查看本机连接状态



从 A 到 B 的连接就释放了，连接处于半关闭状态。相当于 A 向 B 说：“我已经没有数据要发送了,并不能发送数据,如果B还发送数据，我仍接收,并给B发送确认。”

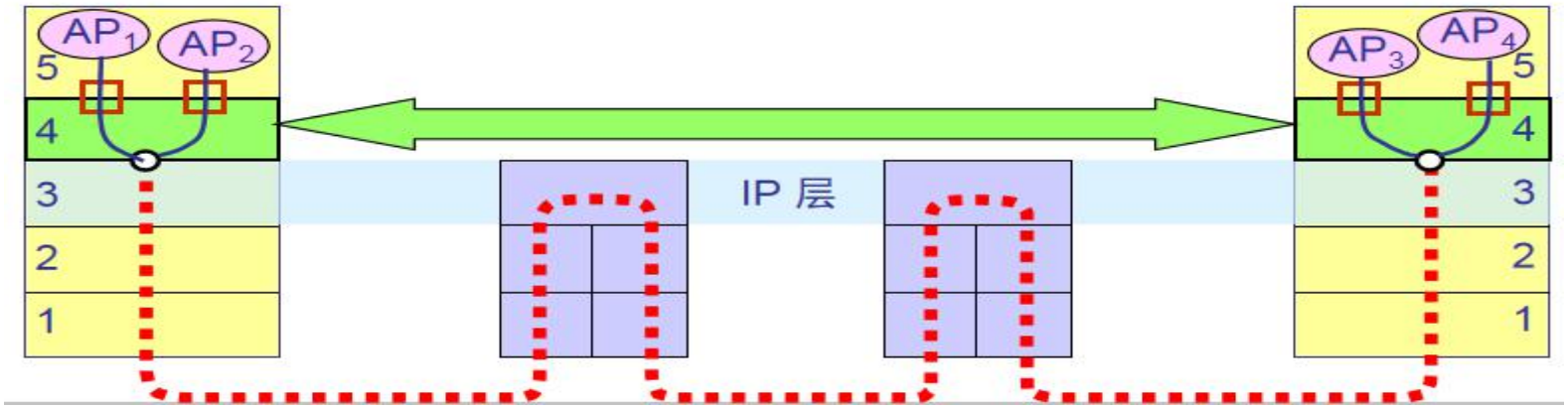
6.5 可靠传输的工作原理

- TCP是一种面向连接(可靠传输)传输层协议
  - 发送方应用进程将数据交付给TCP协议后，TCP协议可无差错的将数据交付给目的应用进程。
- 理想传输条件
  - 传输信道不产生差错(比特差错、传输差错)；
  - 不管发送方发送速率多快，接收方总来的及将接收的数据交付给上层应用进程，避免接收缓存溢出。



6.5 可靠传输的工作原理

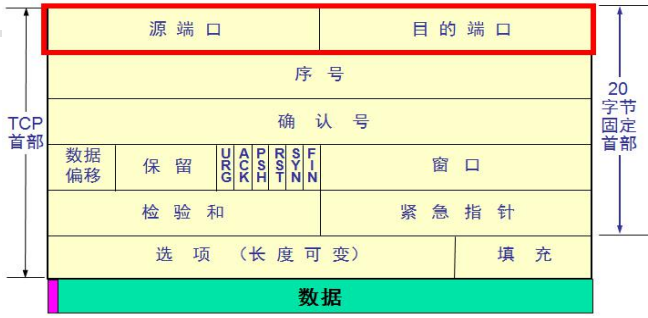
- 解决方法:TCP报文段通信过程中
  - 差错控制：序号+确认应答+超时重发；
  - 流量控制：实现发送速率与接收方应用进程从接收缓冲区交付速率之间的匹配，利用交付速率控制发送速率。
  - 一般采用两个机制同时实现：实现流量控制同时实现差错控制。
  - 拥塞控制：路由器缓冲溢出问题。



6.5 可靠传输的工作原理

- 差错控制
  - （1）序号
  - （2）确认应答
  - （3）超时重发；
- 流量控制
  - （1）停止等待协议
  - （2）连续ARQ-后退N帧协议
  - （3）连续ARQ-选择重发协议

（1）序号 ：字节编号



- TCP协议提供面向字节流传输。
  - TCP将所要传送数据看成是字节组成的字节流。
  - 每一个字节对应于一个序号。
  - 接收方给发送方应答也是按照字节而不是TCP报文段进行确认
    - ACK=1，确认序号有效， ACKn = N；
- 建立连接时，双方首先要协商初始序号。
  - 假设发送方和接收方协商的初始序号别为：x，y。
  - 发送方和接收方开始发送的数据序号分别为： x+1，y+1。
- TCP 发送的TCP报文段首部中序号字段数值表示该TCP段数据字段第一个字节在数据流（字节流）中序号。

举例

- 假设A要发送5000字节的文件给B，建立连接时双方协商MSS=1000字节，在传输层该文件数据被分为5个TCP段进行传输。
  - 假设建立TCP连接时，发送方A随机选择的初始序号x=10000，则数据传输开始序号为x+1=10001；
  - 5个TCP段首部“序号”字段的值分别为：
    - 第一个TCP段：“序号”字段=10001（范围：10001~11000）；
    - 第二个TCP段：“序号”字段=11001（范围：11001~12000）；
    - 第三个TCP段：“序号”字段=12001（范围：12001~13000）；
    - 第四个TCP段：“序号”字段=13001（范围：13001~14000）；
    - 第五个TCP段：“序号”字段=14001（范围：14001~15000）；

（2）ACK确认应答

- TCP确认应答采用按字节序号进行确认应答。
  - ACK=1时，确认序号字段有效，ACKn = N;
  - 确认序号=已收到的数据的最高字节序号+1；
  - 确认序号=接收方期望下次收到的TCP报文段中序号字段的值;
  - 确认序号=接收方期望下次收到的TCP报文段中数据字段第一个字节序号;

（2）ACK确认应答

- TCP确认应答特点
  - （1）单独确认：一方给另一方发送一个确认应答，应答中无数据;
  - （2）捎带确认：一方给另一方发送数据时，捎带应答。

（2）确认应答

- TCP确认应答特点
  - 累积确认：接收方不必对接收到的每一个TCP报文段逐个发送确认，可以在受到几个TCP报文段后，对按序到达的最后一个TCP报文段发送确认即可
  - 累积确认好处：某个ACK确认应答丢失，不影响可靠传输（条件：不导致发送方重发定时器超时，不需要对该确认应答重发）。
  - 举例：
    - 假设发送方发送TCP报文段（数据字节序号：1201~1300），接收方发送ACKn=1301，表明1300以前字节均成功接收；发送方再发送TCP报文段（数据字节序号：1301~1400），接收方发送ACKn=1401，表明1400以前字节均成功接收；
    - 如果ACKn=1301确认丢失，但发送方后来接收到ACKn=1401应答，表明1400以前字节均成功接收；但条件是序号为13001的TCP数据段重发定时器未超时。

（3）TCP超时重发

- 含义：
  - 发送方每发送一个TCP报文段，就启动一个重发定时器，如果定时器超时还没有收到对方相应的ACK确认应答，则重传该TCP报文段。
  - 该重传TCP报文段在网络层被认为是新的IP分组
- 遇到的挑战：如何确定重发定时器的超时时间？
  - 设置过大，发送端系统可能在某时刻有许多重传定时器同时工作，占用了大量的系统资源；
  - 设置过小，网络中存在过多重发TCP报文段，消耗了网络可用带宽。

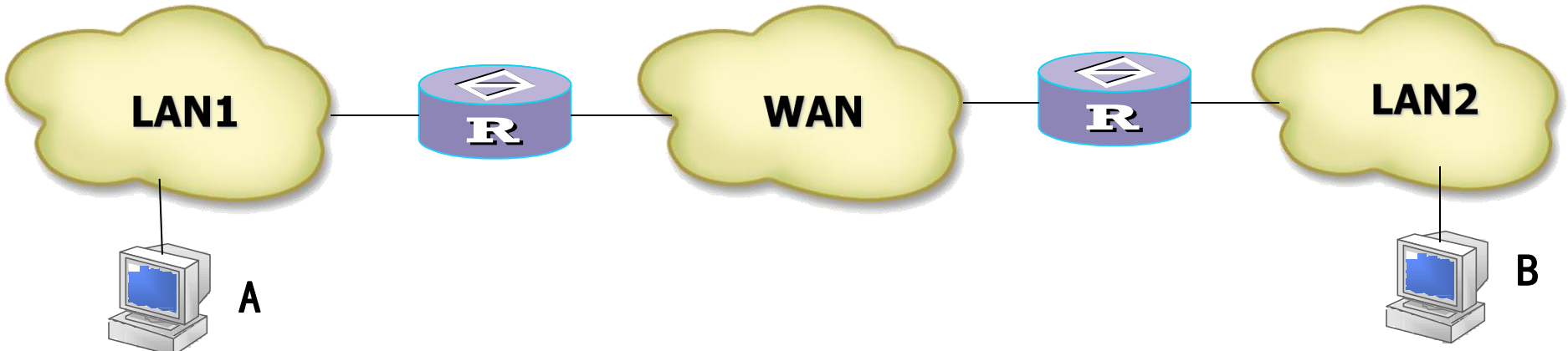
（3）TCP超时重发

- 一般重发定时器时间设置原则为：稍大于RTT
- 很难精确测量到RTT：分组独立路由，（Round Trip Time）.
- 动态变化：由于网络负载随时变化；

- 解决思路：利用以前历史RTT时间预测出当前RTT时间；具体算法：
  - TCP协议标准算法
  - Karn/Partridge算法；
  - Jacobson/Karels算法；

重发定时器-超时时间计算

- 超时重发机制是TCP协议中最重要和最复杂技术之一。
- TCP每发送一个TCP报文段，就对该TCP报文段启动一个重发定时器；重发定时器超时，还没收到确认，就要重传该TCP报文段。
- 重发定时器超时时间设置多少合适？
  - 一般设置为稍大于通信双方的RTT时间。
  - RTT:发送方发送完一个TCP报文段最后一个字节开始计时，到收到对该报文段ACK应答为止的一段时间。



方法：加权平均往返时间

- TCP 采用一种加权平均往返时间 RTT_S（这又称为平滑的往返时间）来计算当前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：

$$\text{新的 } RTT_S = (1-\alpha) \times (\text{旧的 } RTT_S) + \alpha \times (\text{新的 RTT 样本}) \quad (6\text{-}1)$$

- 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。

超时重传时间： RTO (Retransmission Time-Out)

- RTO 应略大于上面得出的加权平均往返时间 RTT_S。
- RFC 2988 建议使用下式计算 RTO：
- $$RTO = RTT_S + 4 \times RTT_D \qquad (6\text{-}5)$$
- RTT_D 是 RTT 的偏差的加权平均值。
- RFC 2988 建议这样计算 RTT_D。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，则使用下式计算加权平均的 RTT_D：

$$\text{新的 } RTT_D = (1 - \beta) \times (\text{旧的} RTT_D) + \beta \times |\text{新} RTT_S - \text{新RTT 样本}| \qquad (6\text{-}6)$$

- β 是个小于 1 的系数，其推荐值是 1/4，即 0.25。

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？产生二义性 。

Karn 算法

- 计算RTT样本时，只要TCP报文段重传了，就不采用该TCP报文段（包括重发的TCP报文段）往返时间样本。
- 这样得出的加权平均往返时间 RTT_S 和超时重传时间 RTO 就较准确。

Karn 算法

- 产生新问题
  - 由于网络发生拥塞，TCP报文段和确认应答在网络上传输延迟时间增大了；
  - 如果重发定时器超时没有收到ACK确认，就需要对该TCP段重传；根据Karn算法，不需要考虑该TCP报文段（由于重发）的RTT样本，则RTO时间无法更新；
  - 如何将网络拥塞对RTO影响考虑进去呢？

修正的 Karn 算法

- TCP报文段每重传一次，就把 RTO 增大一些：

$$\text{新的 RTO} = \gamma \times (\text{旧的 RTO})$$

- 系数 γ 的典型值是 2 。
- 当不再发生TCP报文段的重发时，才根据报文段的往返时延更新平均往返时延 RTTs 和超时重传时间 RTO 的数值。
- 实践证明，这种策略较为合理。

预测当前重发定时器超时时间比较复杂！复杂不一定没有问题！

传输层可靠通信基本原理

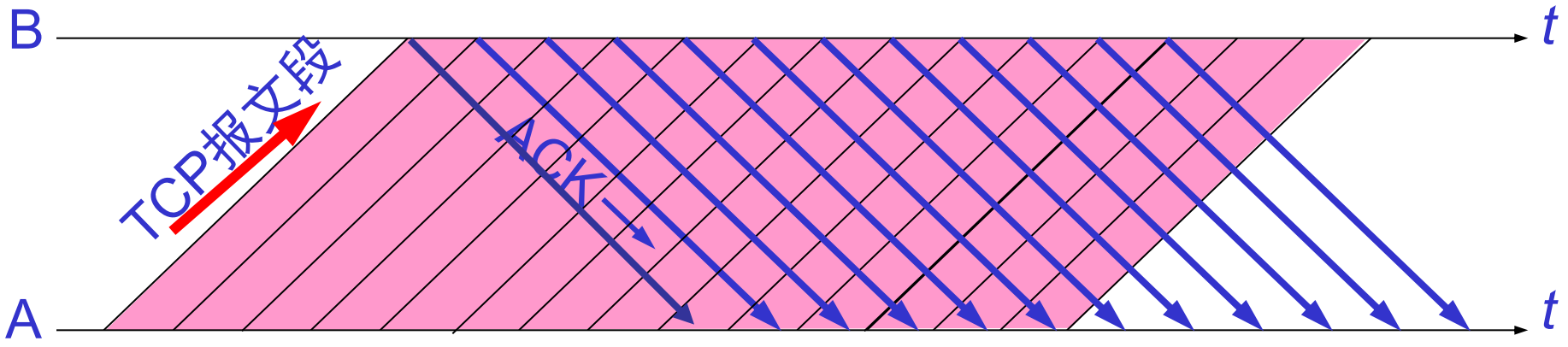
- （1）发送方每发送一个TCP报文段：在首部通过"序号"字段说明数据部分在原始字节流中相对位置；
- （2）发送方启动一个重发定时器；
- （3）接收方每接收到一个TCP报文段，首先进行差错检测：
  - 如果校验正确（伪+首+数），则给发送方发送相应的ACK确认应答(ACK=1，确认序号)，按字节进行确认(可采用累积确认和捎带确认)；
  - 否则：直接丢弃；
- （4）如果发送方重发定时器未超时收到ACK确认，发送窗口向前滑动，发送方可继续发送后面TCP报文段；
- 如果发送方重发定时器超时，未收到接收方对该TCP报文段的确认应答，则重发TCP报文段，分两种情况：
  - 连续ARQ - 后退N帧机制
  - 连续ARQ - 选择重发机制

6.5 可靠传输的工作原理

- 差错控制
  - （1）序号
  - （2）确认应答
  - （3）超时重发；
- 流量控制
  - （1）连续ARQ-后退N帧协议
  - （2）连续ARQ-选择重发协议

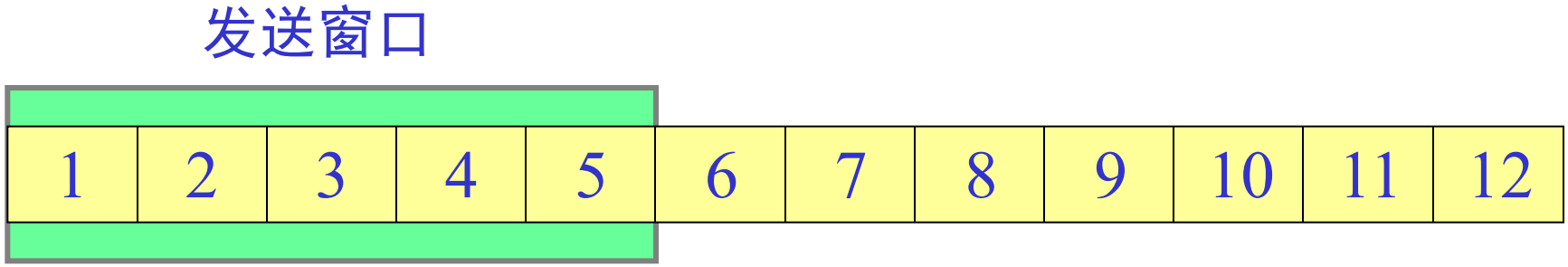
连续ARQ协议

- 发送方可连续发送多个TCP报文段，不必每发完一个TCP报文段就停顿下来等待对方的确认。
- 信道上一直有数据不间断地传送，提高信道利用率。

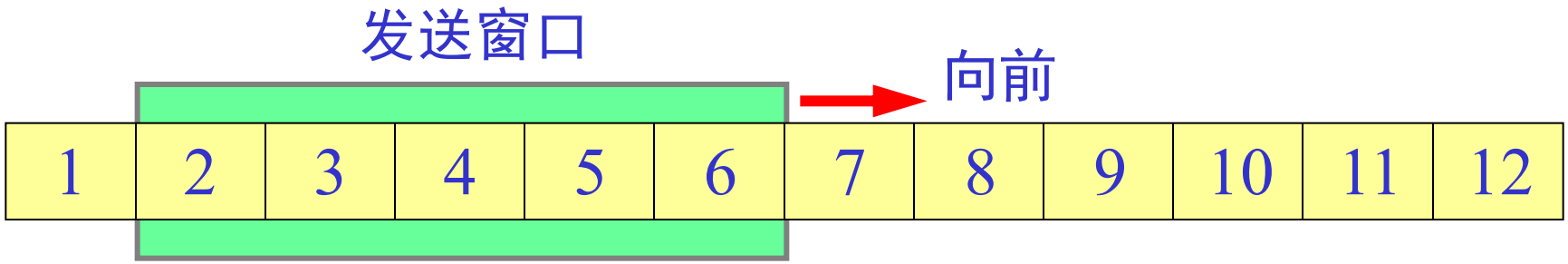


连续 ARQ 协议也称为：流水线传输方式

连续 ARQ 协议



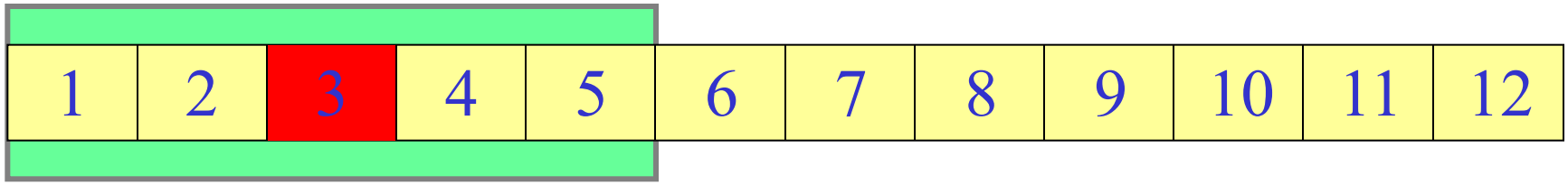
(a) 发送方维持发送窗口（发送窗口是 5）



(b) 收到一个确认后发送窗口向前滑动

连续ARQ协议-举例

- 如果发送方连续发送了前 5 个TCP报文段，中间第 3 个TCP报文段丢失了；这时接收方只能对前两个TCP报文段发出确认；发送方无法知道后面三个TCP报文段传输情况。



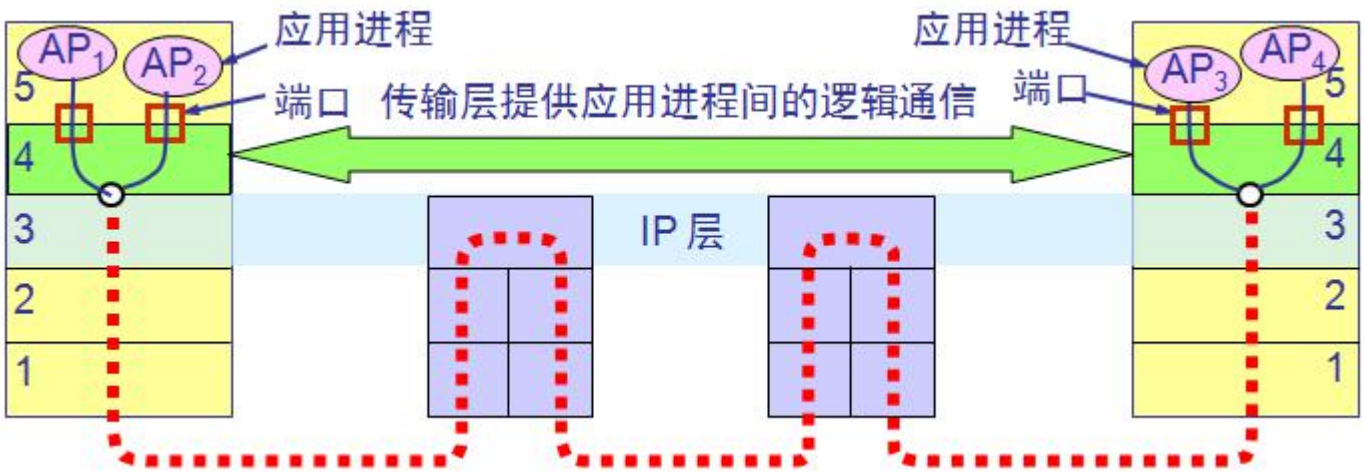
- 方法一：后退 N协议TCP标准协议支持；
  - 把3号(包括)后面所有TCP报文段再重传一次，这就叫做 Go-back-N（后退 N协议）。
  - 当通信线路质量不好时，后退 N协议会带来负面的影响，网络中存在重复的TCP报文段。
- 方法二：
  - 仅把出错的TCP报文段(3号)重发；- 选择重法协议
  - TCP标准协议不支持；需要对其功能扩展才能实现
  - 有些TCP协议实现时采用选择确认（SACK-选择ACK确认选项）；

6.6 TCP 可靠传输的实现

- TCP 可靠传输机制-回忆
  - **差错控制：**字节序号（32bit）；TCP协议ACK确认都是基于序号（确认序号）不是基于报文段；ACK应答，超时重发；
  - 重发定时器超时时间计算比较复杂；

6.6 TCP 可靠传输的实现

- 流量控制：连续ARQ协议
  - 连接每一端都设有两个两个缓存：发送和接收缓存。
  - 连接每一端都设有两个窗口：一个发送窗口和一个接收窗口。
  - TCP发送窗口经常处于动态变化之中，如前移速度和大小。

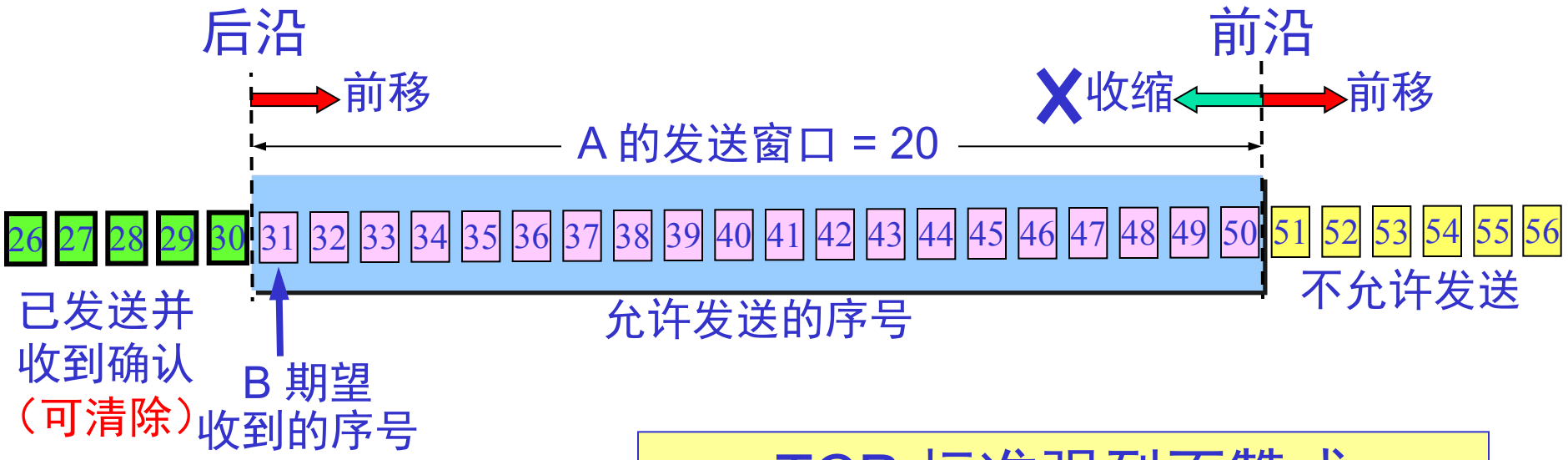


6.6 TCP 可靠传输的实现

- TCP段首部的“窗口”字段数值就是接收方接收窗口有效大小，目的控制发送方发送窗口数值上限。
- 理想情况是发送窗口大小等于接收窗口；

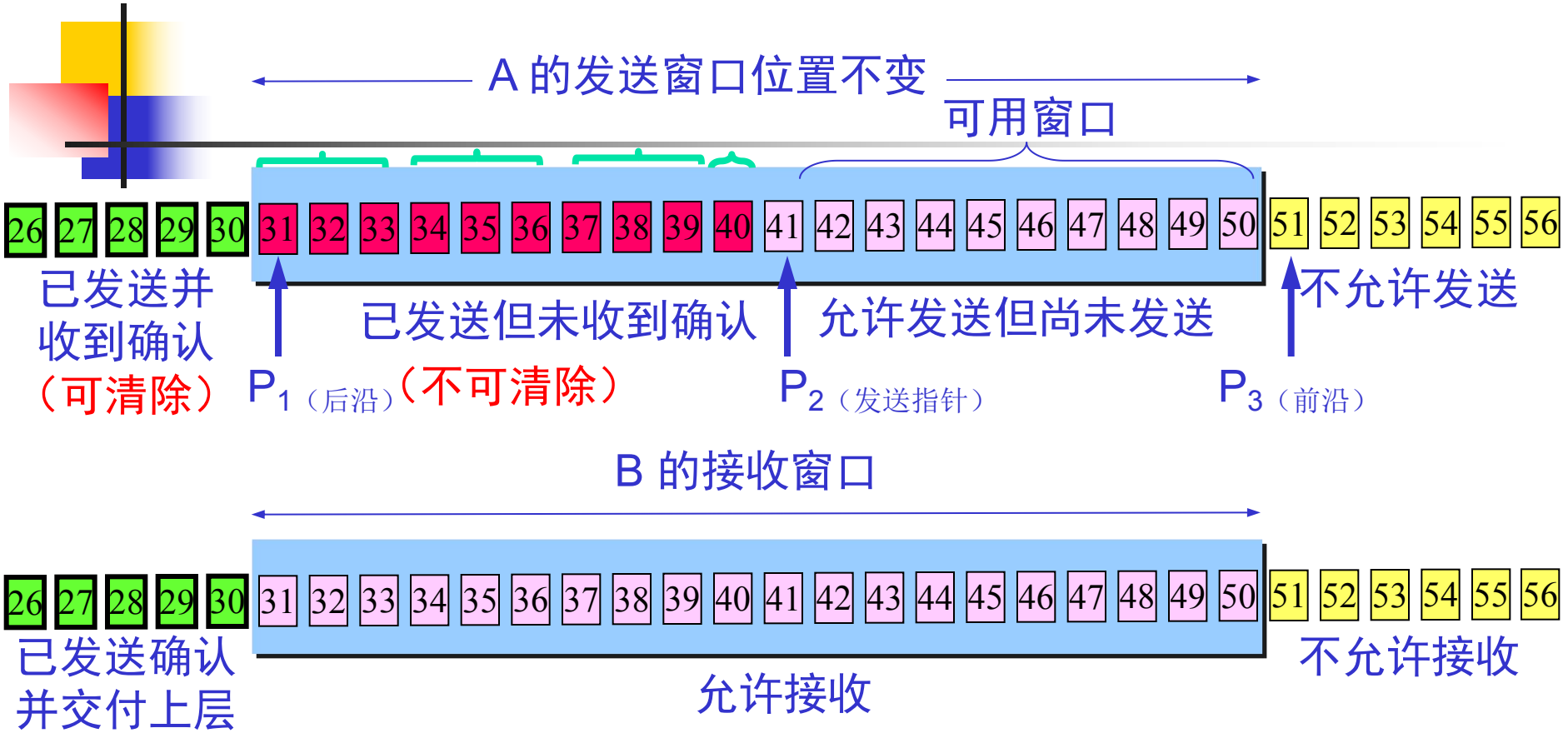
6.6.1 以字节为单位的滑动窗口

A发送方：根据 B 给出的“窗口”值
构造出自己的发送窗口



TCP 标准强烈不赞成
发送窗口前沿向后收缩

A 发送了 10 个字节的数据

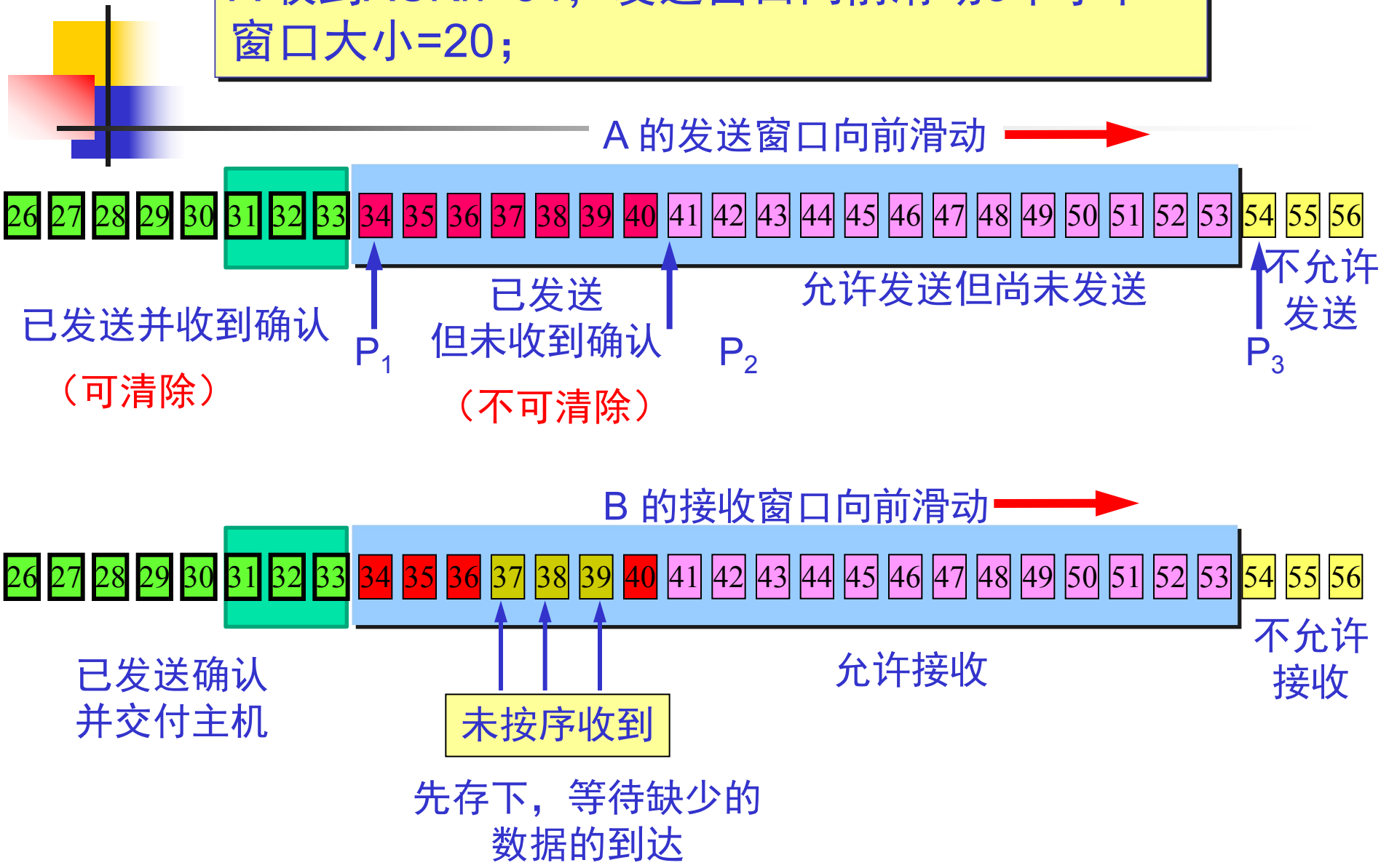


$P_3 - P_1$ = A 的发送窗口（又称为通知窗口）

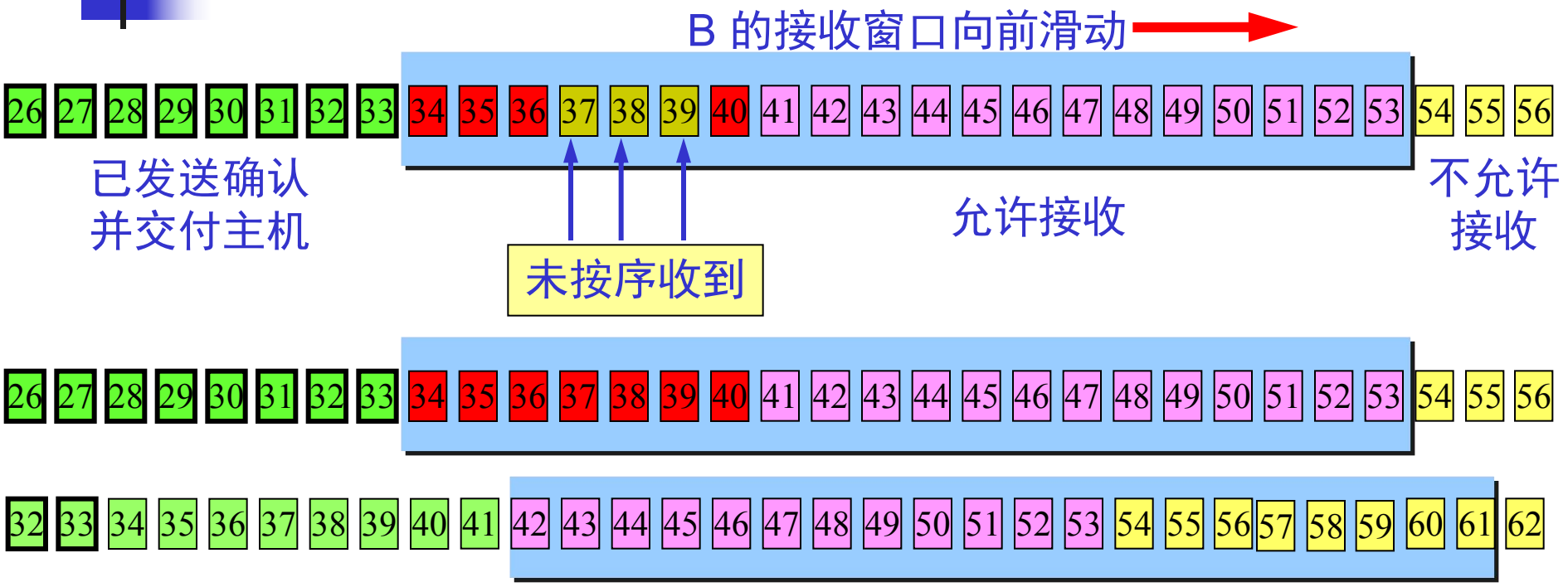
$P_2 - P_1$ = 已发送但尚未收到确认的字节数

$P_3 - P_2$ = 允许发送但尚未发送的字节数（又称为可用窗口）

A 收到ACKn=34，发送窗口向前滑动3个字节
窗口大小=20；



B接收窗口变化情况

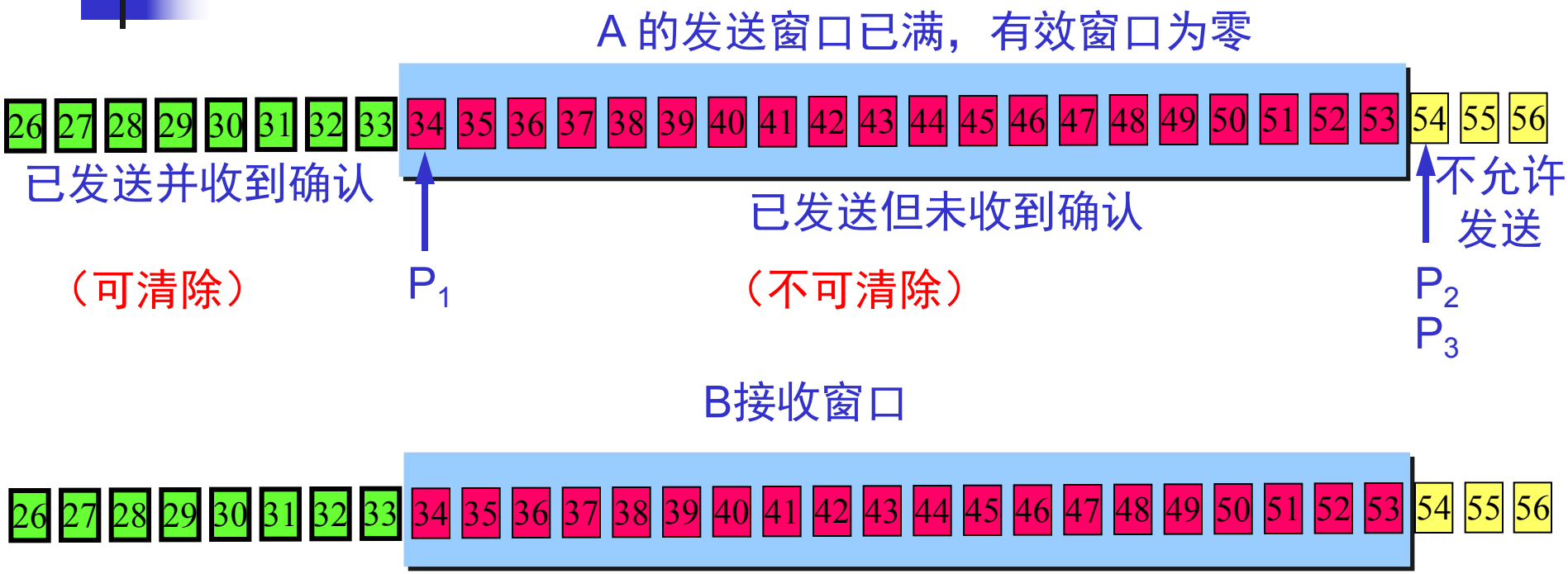


(1) 等到34,35,36号字节到达后，将34~41号字节数据交付上层；

(2) 采用累积应答：ACKn = 41 ；

(3) 接收窗口向前滑动7个字节单位。

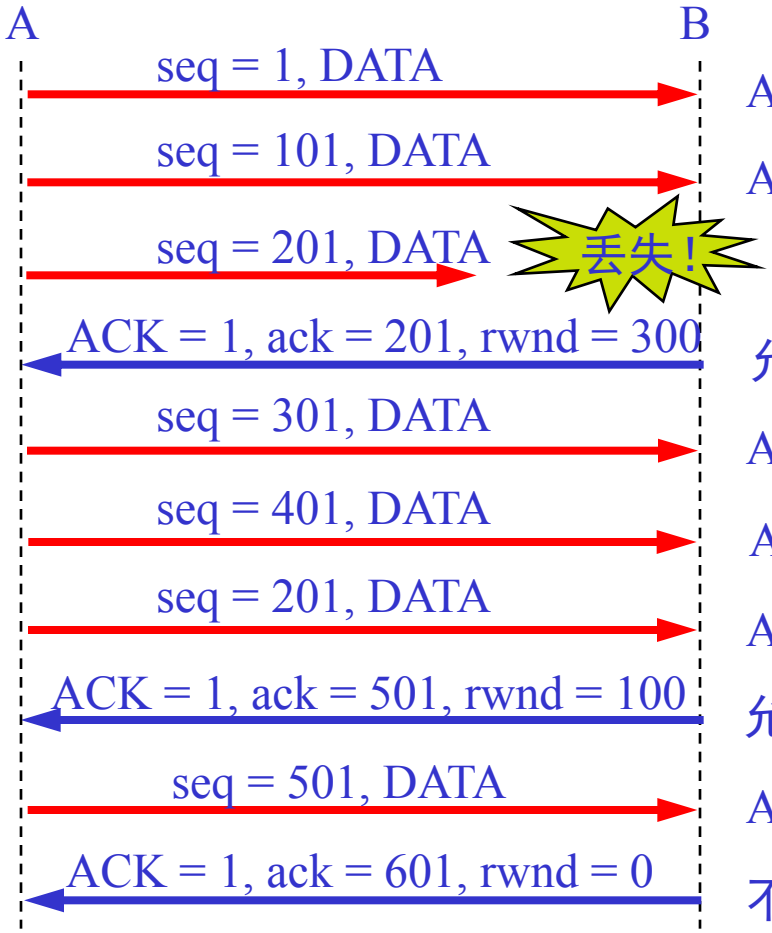
A 的发送窗口内的序号都已用完，
但还没有再收到确认，必须停止发送。



举例：实际流量控制（带差错控制）
发送窗口动态调整（由于接收窗口动态变化）

流量控制举例（接收窗口大小可变）

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初始序号协商为0；MSS=100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

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

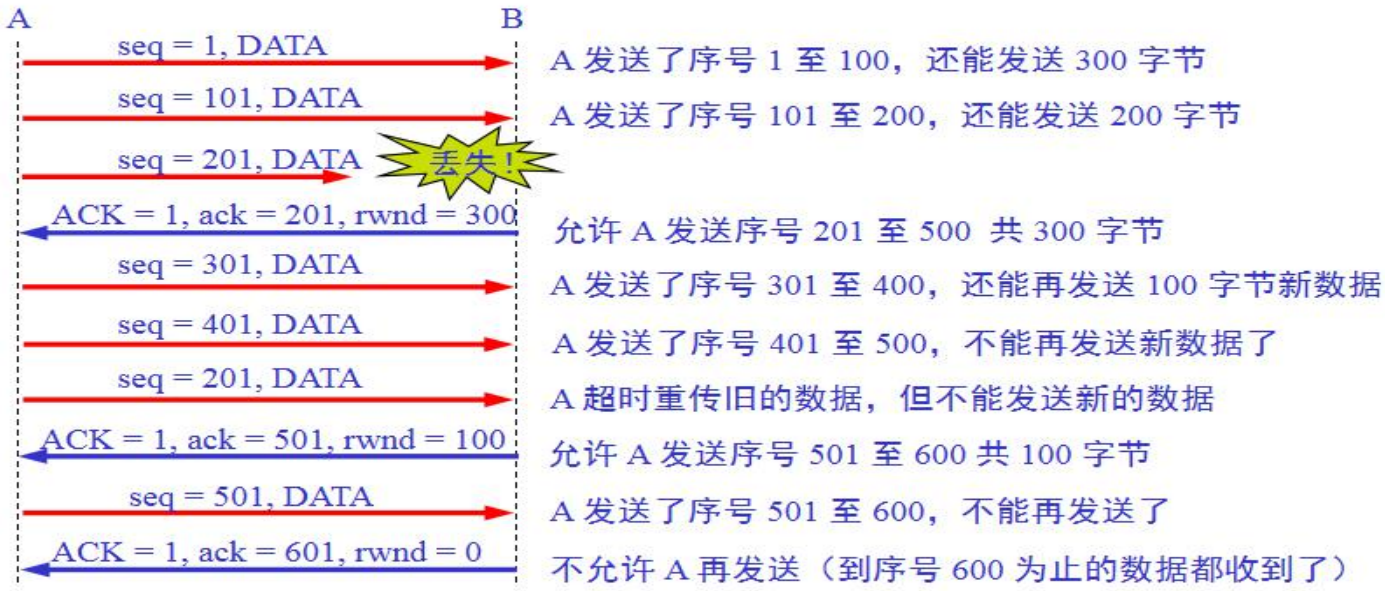
允许 A 发送序号 501 至 600 共 100 字节

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）

讨论1：零窗口问题

- 接收方主机B进行了三次流量控制（接口窗口大小发生可变）：rwnd = 300;rwnd = 100;rwnd = 0;通过接口窗口不仅控制发送窗口向前滑动速度，还控制其大小；
- rwnd = 0时，不允许发送方发送数据，直到B重新发送一个新窗口值不为0为止；



零窗口问题

- 特殊情况：假设B向A发送了rwnd = 0的报文段后不久，B接收缓存又有一些存储空间，于是B向A发送了rwnd = 400的报文段（应答报文或者通告报文形式发送，如果以数据报文形式呢？），但问题是这个通告报文段在传输过程中丢失了；
- 结果：A一直等B发送非零窗口的通知，而B一直等待A发送数据，进入死锁。

零窗口问题解决方法

- TCP为每一个连接设有一个坚持定时器。
- 只要TCP连接一方收到零窗口通知，就启动该定时器；
- 如果定时器超时，就发送一个零窗口探测报文（仅带一个字节的数据）；
- 对方接收到探索报文后，必须给另一方发送报文段，通知其现在窗口大小；
- 如果窗口仍旧为0，则另一方重新设置定时器值（周期性）；否则死锁问题就解决了。

注意以下四点

- 发送方应用进程只要将要发送的字节流写入TCP发送缓存即可，TCP协议来完成可靠通信，用户不需要参与。
- 接收方应用进程从TCP接收缓存中读取字节流，TCP协议保证了所读取的字节流不会有差错、不会丢失、不会重复，也不会乱序。
- 发送方维护和管理发送缓存和发送窗口；接收方维护和管理接收缓存和接收窗口；
- 发送和接收缓存和序号（32比特）都是有限的，一般采用循环使用，所以最好将缓存画为环形。
- 发送或接收窗口只需对相关三个指针进行维护即可。

TCP通信特点

- 发送窗口是根据接收窗口有效大小设置
  - 理想情况：发送窗口等于接收窗口大小；但某一时刻，发送窗口并不总是和接收窗口一样大（因为有一定的时间滞后）；
- TCP 标准没有规定对不按序到达的数据应如何处理。
  - 一般处理方法：只要接收的乱序数据序号在接收窗口范围在，先临时存放在接收窗口，等到前面数据到达后，一起交付上层应用进程。
- TCP 要求接收方采用累积确认，可以减小数据传输开销。
  - 接收方不应过分推迟发送ACK确认，防止发送方不必要重发；
  - TCP标准规定：ACK确认推迟时间不应超过0.5秒；若收到一连串最大的TCP报文段， 则必须每隔一个TCP段发送一个ACK确认[RFC 1122]。

保活定时器作用

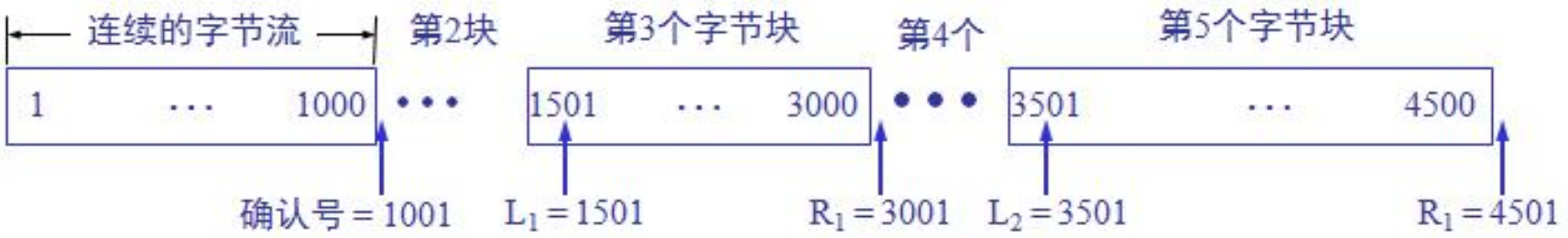
- 问题：
  - 假设客户建立了一个到服务器的连接，并发送一些TCP报文段，然后客户进程出现了故障，而服务器上TCP连接就永远处于打开状态，占用了系统资源。
- 解决方法：服务器上引入保活定时器。
  - 保活定时器初始超时时间一般设置为2小时；
  - 连接建立启动该定时器；
  - 当服务器每收到来自某客户的TCP段，该服务器连接上相应保活定时器复位；
  - 如果服务器上保活定时器超时，服务器不断（每隔75秒）发送一个探测报文给客户端，客户必须发送应答；
  - 当服务器连续发送了10个探测报文还在该连接上没有收到客户端应答，它就关闭该连接。

TCP传输效率问题

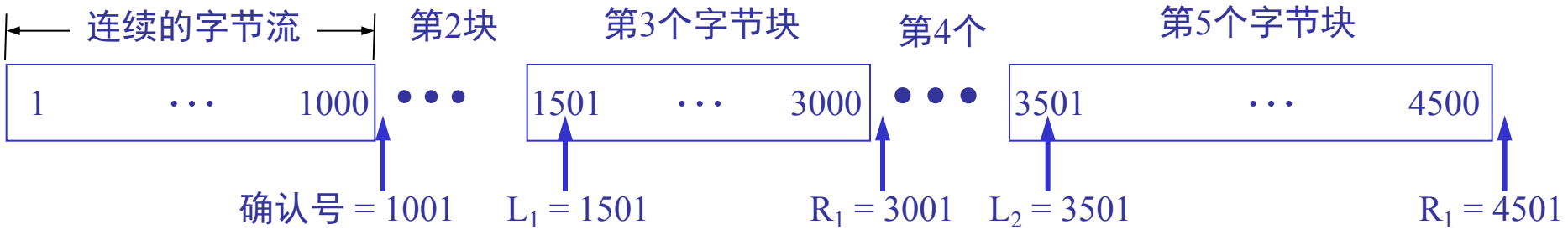
- 采用不同的机制来控制TCP报文段发送时机
- 第一种机制：满足要求就发
  - 发送方维持一个变量-MSS（最大报文段长度 ），只要发送窗口中存放的数据达到 MSS 字节时，就封装成一个 TCP 报文段发送出去。
- 第二种机制：周期性发送
  - 发送方维护一个发送定时器，周期性把当前发送窗口数据封装为不同TCP报文段（但数据长度不能超过 MSS）发送出去。
- 第三种机制：
  - 由发送方的应用进程指明采用推送(push)操作，发送方将发送缓存中数据封装成不同 TCP 报文段发送出去。

讨论2：选择确认 SACK (Selective ACK)

- 接收方无差错的收到了和前面字节流不连续的多个字节块。而且这些字节块的序号都在接收窗口之内，接收方如何处理。
- 方法一：后退N帧，缓存；但重发 时丢失TCP 段以及以后数据全重发；
- 方法二：选择重发，缓存；只重发丢失TCP报文段（TCP扩展后）；
- 方法三：选择确认，接收方先收下这些不连续数据块，但要想办法将字节连续数据块准确地告诉发送方，使发送方不需要等到重发定时器超时，仅把不连续的字节块重发即可。



接收到的字节流序号不连续



- 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。

RFC 2018 的规定

- 如果使用选择确认，在建立 TCP 连接时，在 TCP 首部选项字段中加上“允许 SACK”的选项，而双方必须都事先商定好。
- 此时原来TCP首部中“确认号“字段用法仍不变，只是以后在 TCP 报文段的首部（选项字段）中增加了 SACK 选项，以便报告收到的连续字节块的边界（左右边界各4个字节）。
- 由于首部”选项“的长度最多只有 40 字节，需要一个字节指明要采用SACK选项，另一字节指明该选项实际占用的字节个数。
- 指明一个连续字节块要用 8 字节，因此在选项中最多只能指明 4 个连续字节块的边界信息（共34个字节）。
- SACK文档没有指明发送方应该如何响应SACK，因此大多数TCP协议实现还是采用后退N帧机制。

本节小结

6.5 可靠传输的工作原理

6.5.1 停止等待协议

6.5.2 连续 ARQ 协议

6.6 TCP 可靠传输的实现

6.6.1 以字节为单位的滑动窗口

6.6.2 超时重传时间的选择

6.6.3 选择确认 SACK-选择重发

6.7 TCP的流量控制

6.7.1 利用滑动窗口实现流量控制

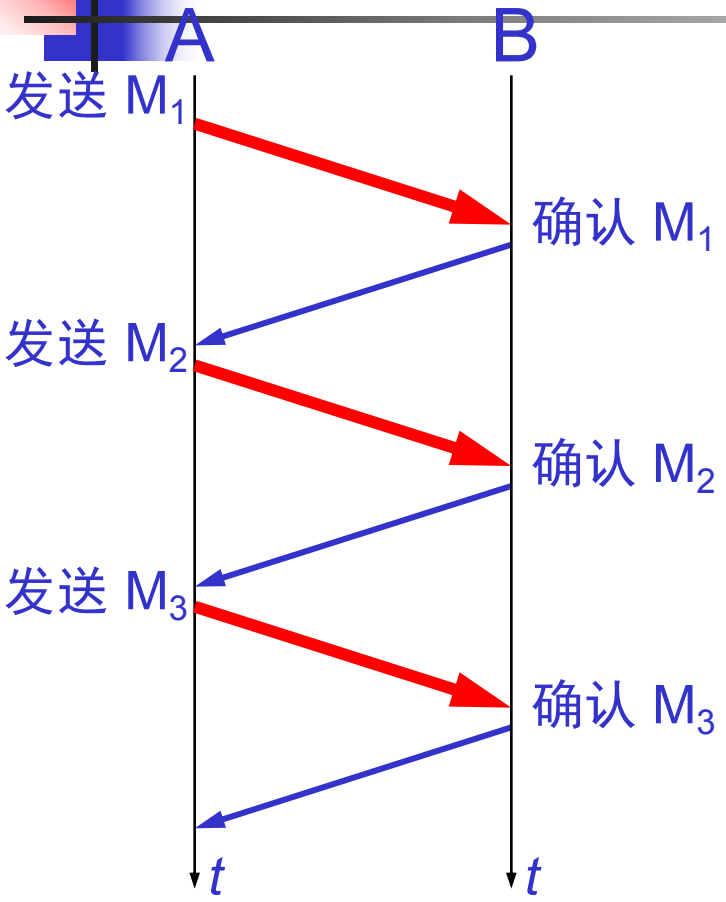
6.7.1 必须考虑传输效率

利用滑动窗口实现流量控制（举例2）

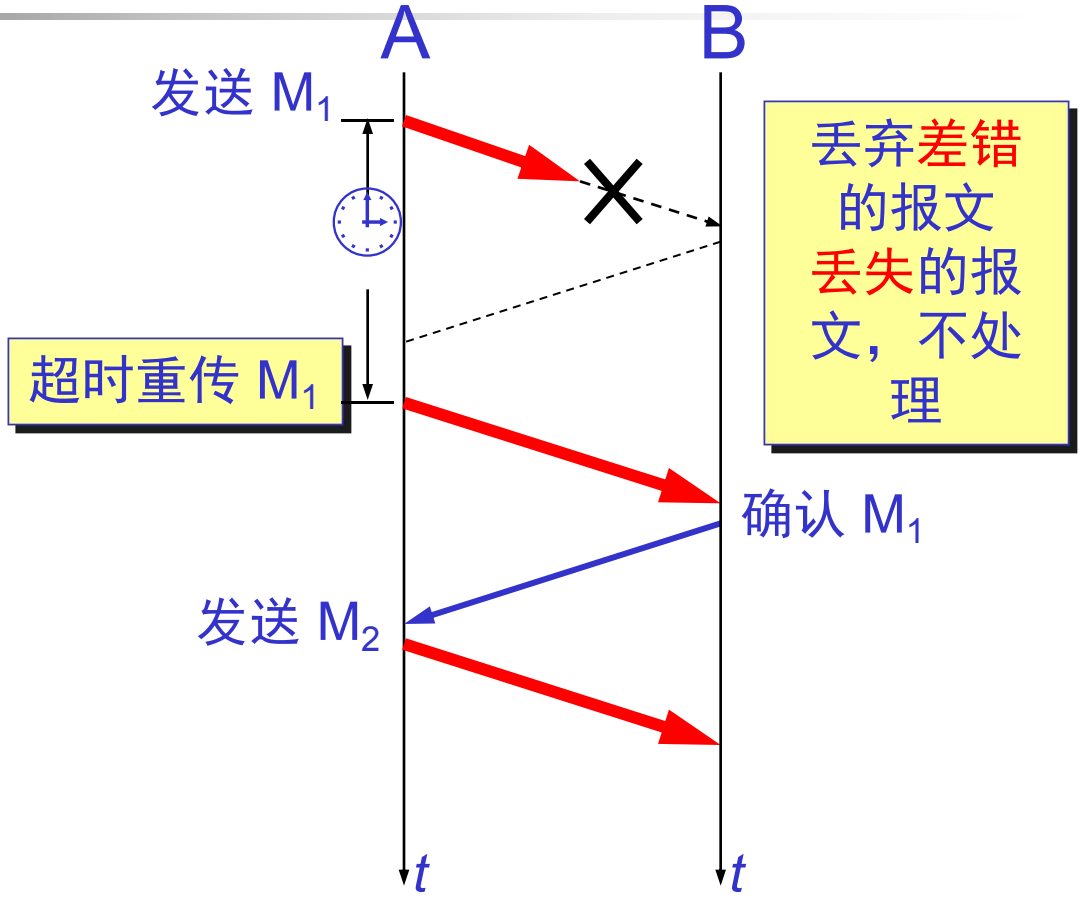
- 一般说来，我们总是希望数据传输得更快一些；但如果发送方把数据发送得过快，接收方就可能来不及接收（具体讲，来不及向上层交付），这就会造成数据的丢失和重传。
- 流量控制(flow control)就是让发送方的发送速率不要太快，让接收方来得及向应用层交付数据。
- TCP利用可变大小滑动窗口机制在TCP连接上实现流量控制。

- 假设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）”，则发送方发送窗口不能超过400字节；
  - TCP窗口计算单位是字节，而不是TCP段；
  - TCP在建立连接时，A设定的序号X=0；真正序号为0+1开始；
  - MSS = 100字节。

（1）停止等待协议



(a) 停止等待



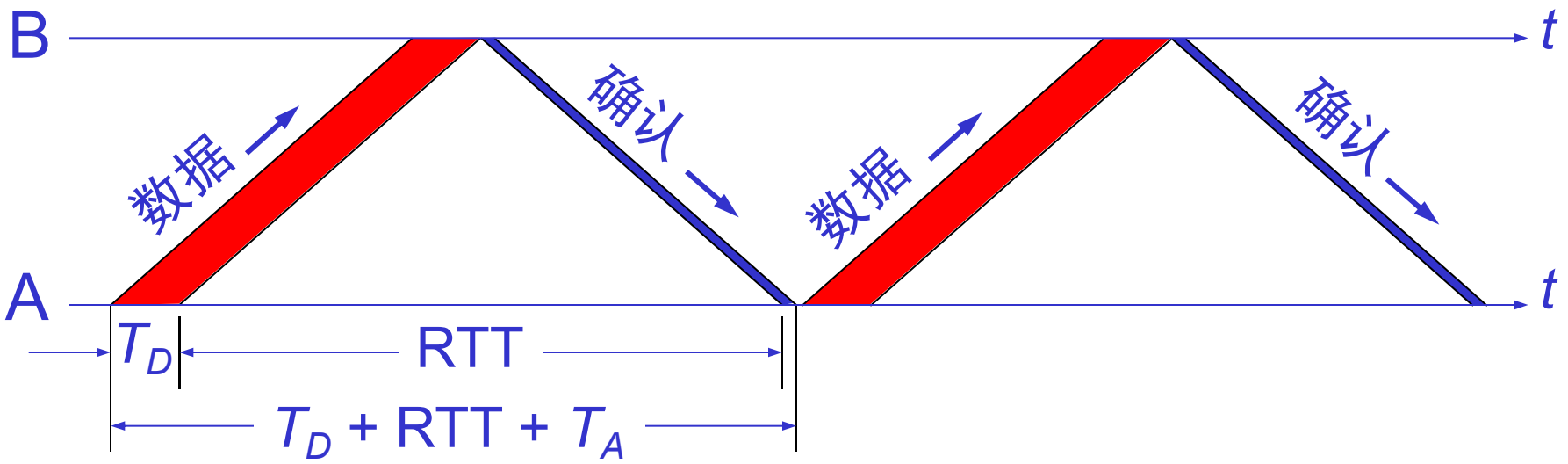
(b) 停止等待-带差错控制

请注意以下三点

- 在发送完一个TCP报文段后，必须暂时保留已发送的TCP报文段，以备重发时使用；只有接收到响应确认后才可清除该TCP报文段。
- TCP报文段（序号）和确认应答（确认序号）都必须有相应序号。
- 超时重发计时器的超时时间应当比数据传输的平均往返时间更长一些，确定起来很难：
  - 网络负载随时发生变化，有时可能发生网络拥塞；
  - 每个TCP报文段以IP分组形式在网络上传输，不同分组可能选择不同路径。

信道利用率

- 停止等待协议的优点是简单，但缺点是信道利用率太低。



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

信道的利用率 U

- 举例：
  - 假设A、B之间信道距离为1200Km，RTT=200ms；数据长度为1200bit；发送速率为1Mb/s；
  - 如果忽略处理时间和 T_A （一般 T_A 远远小于 T_D ）
  - 计算U=5.66%，94%时间信道处于空闲状态。
  - 如果发送速率提高到10Mb/s，则U=0.0571%，99.94%时间信道处于空闲状态。

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