课程回顾

- 运输层的位置

  属于面向通信功能的最高层，同时也是面向用户功能的最低层。

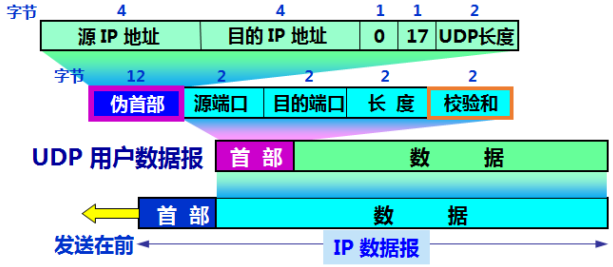
- 运输层的作用
  - 对于低层用户而言，其主要起管控作用，实现运输连接管理、可靠传输、流量控制、拥塞控制等。
  - 对于高层用户而言，其主要起屏蔽作用，屏蔽网络核心建立端到端的逻辑通信信道。
- 运输层与网络层的区别
  - 网络层为主机之间提供逻辑通信，通信的两端是两台主机。
  - 运输层为应用进程之间提供端到端的逻辑通信，通信的真正端点是主机中的进程。
- TCP/IP 运输层端口
  - 标识本计算机应用层中的各进程，具有本地意义。
  - 类型：服务器端口号，注册端口号，客户端口号

进程标识符：操作系统给每个进程定义的一个唯一标识该进程的非负正数。

复用
分用

课程回顾

- 运输层协议
  - UDP
    - 特点：无连接；使用尽最大努力交付；面向报文；没有拥塞控制 ；支持一对一、一对多、多对一、多对多等交互通信
    - UDP数据报

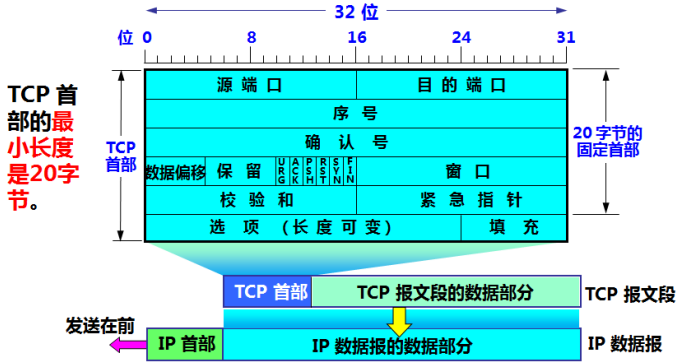


  - TCP
    - 特点：面向连接的运输层协议；实现点对点通信；提供可靠交付的服务；提供全双工通信；面向字节流的协议
    - TCP连接

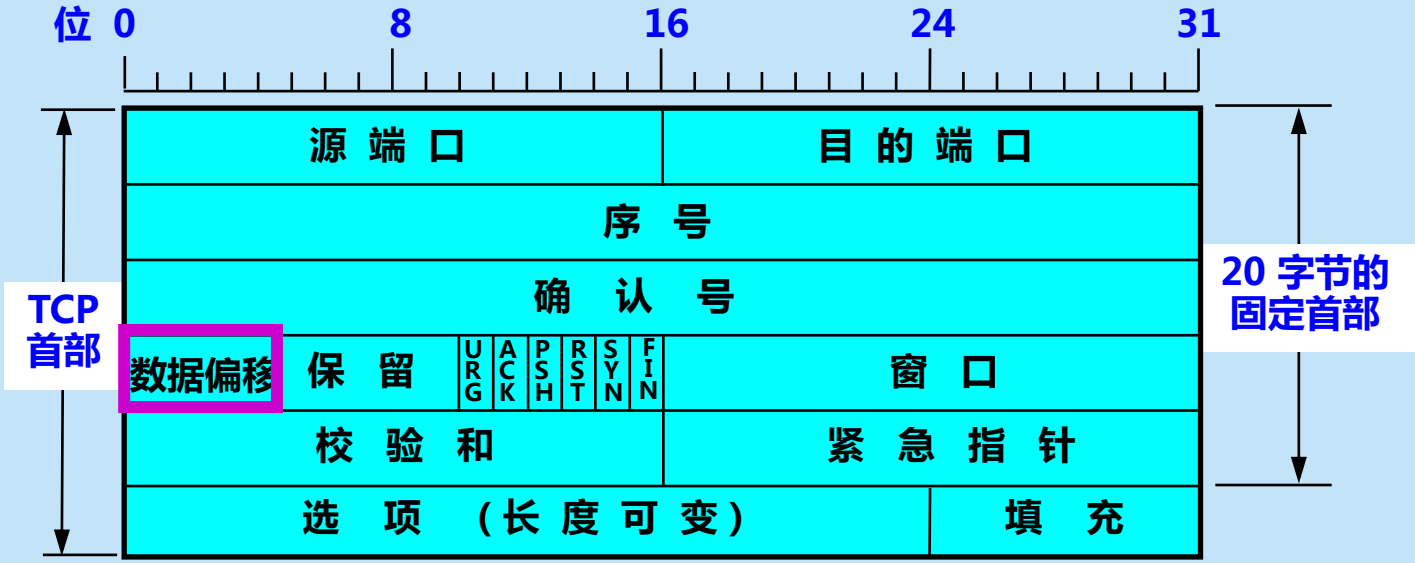
套接字 socket ::= （IP地址 : 端口号）

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

    - TCP报文段

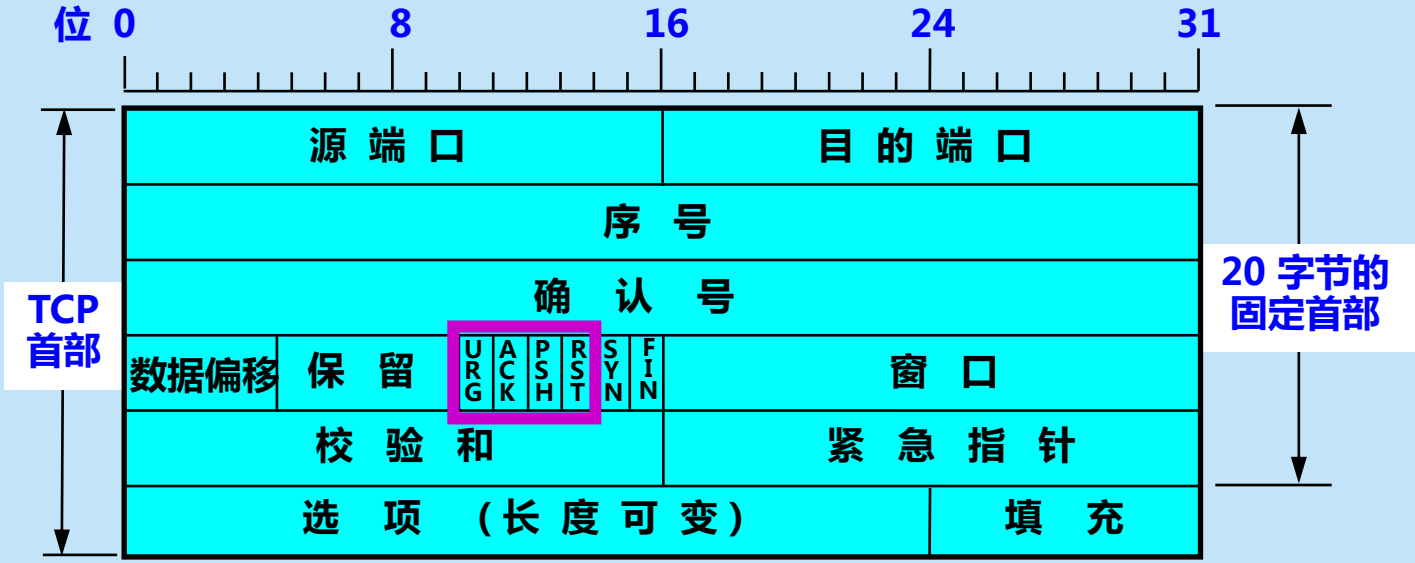


TCP 报文段格式



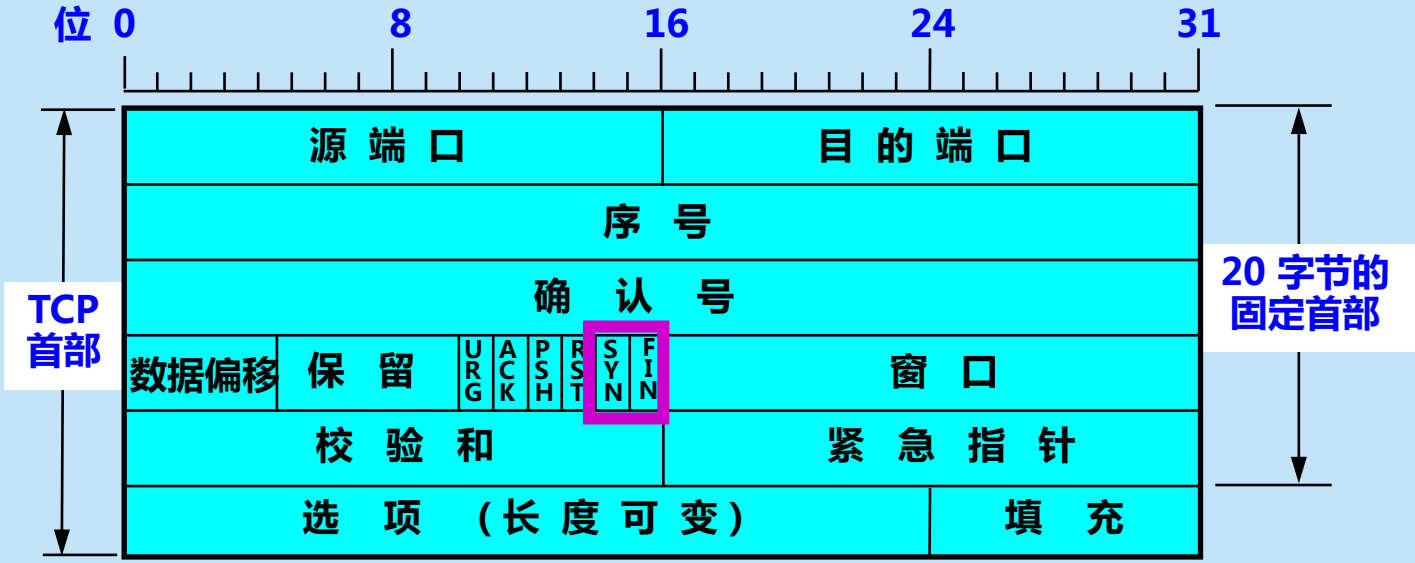
数据偏移（即**首部长度**）：占 4 位，指出 TCP 报文段的**数据起始处**距离 TCP **报文段的起始处**有多远。单位是 32 位字（以 4 字节为计算单位）。

TCP 报文段格式



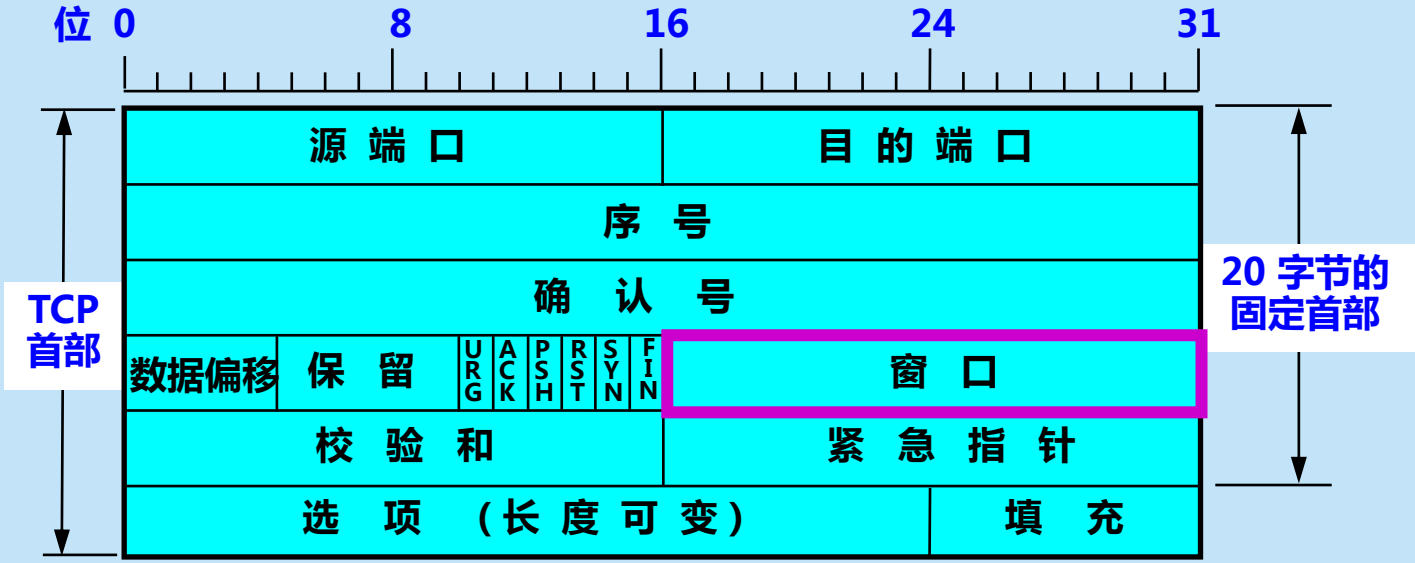
- **紧急 URG**：控制位。当 URG = 1 时，表明紧急指针字段有效，告诉系统此报文段中有紧急数据，应尽快传送 (相当于高优先级的数据)。
- **确认 ACK**：控制位。只有当 ACK =1 时，确认号字段才有效；当 ACK =0 时，确认号无效。
- **推送 PSH** ：控制位。接收 TCP 收到 PSH = 1 的报文段后，就尽快（即“推送”向前）交付接收应用进程，而不再等到整个缓存都填满后再交付。
- **复位 RST**：控制位。当 RST=1 时，表明 TCP 连接中出现严重差错（如主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。

TCP 报文段格式



- **同步 SYN** ：控制位。连接建立时用来同步序号。
  - SYN = 1 表示这是一个连接请求或连接接受报文。
  - 当 SYN = 1，ACK = 0 时，表明这是一个**连接请求报文段**。
  - 当 SYN = 1，ACK = 1 时，表明这是一个**连接接受报文段**。
- **终止 FIN**：控制位。用来**释放一个连接**。FIN=1 表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。

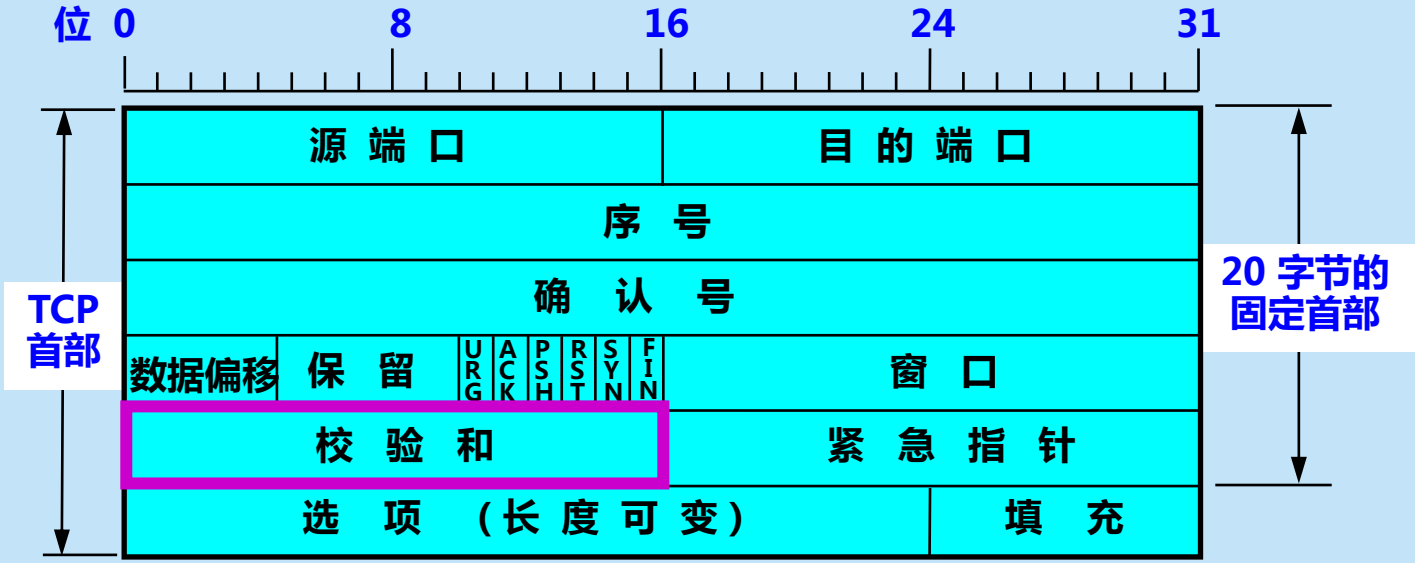
TCP 报文段格式



窗口：占 2 字节。
窗口值告诉对方：从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量（以字节为单位）。

注：窗口字段明确指出了现在允许对方发送的数据量，即让对方设置发送窗口的依据。窗口值经常在动态变化。

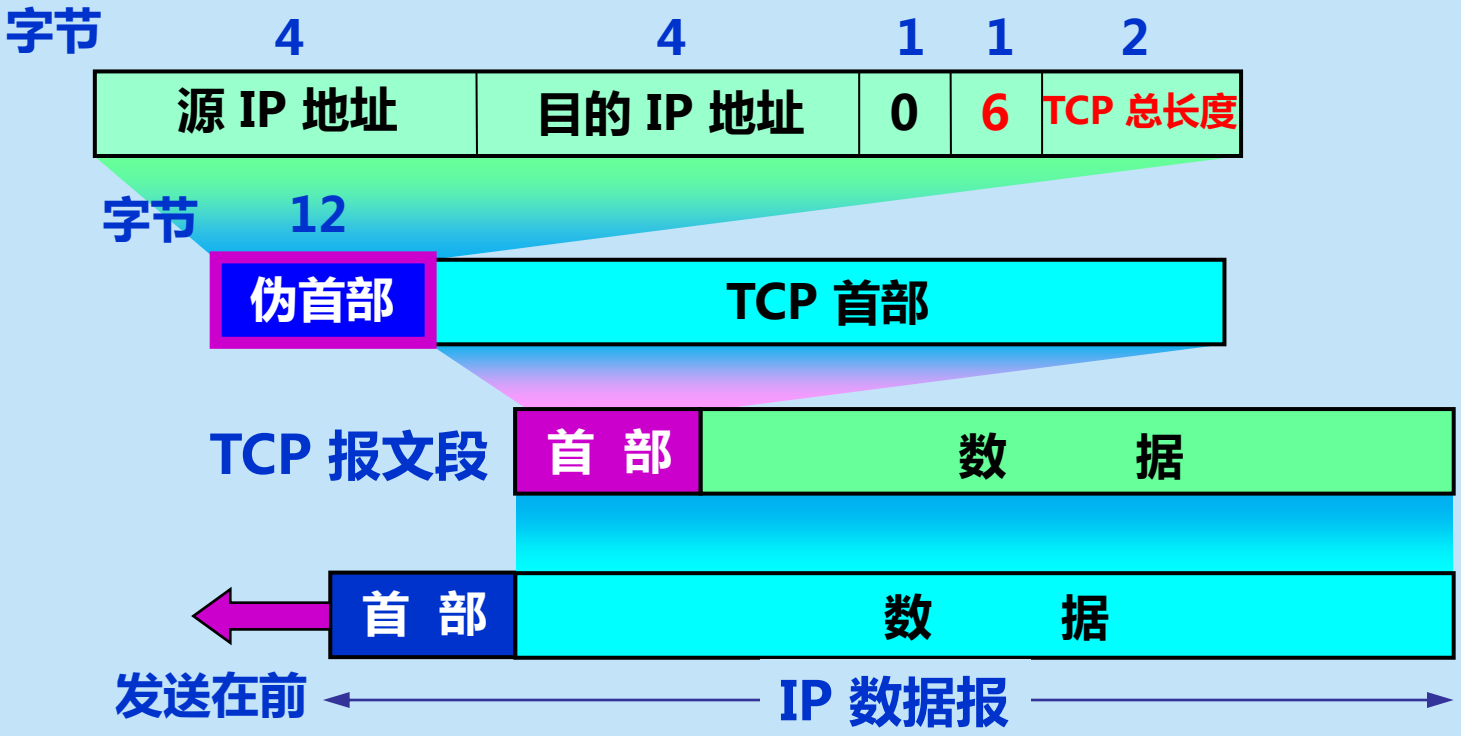
TCP 报文段格式



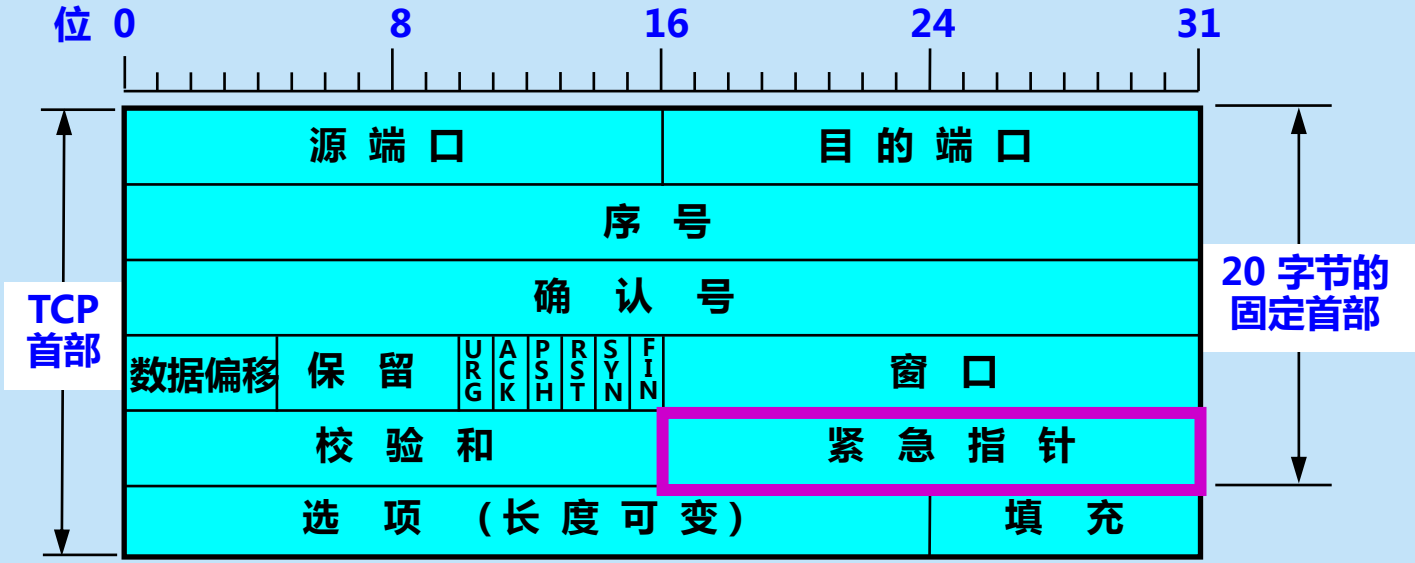
校验和：占 2 字节。检验和字段检验的范围包括**首部和数据**两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的**伪首部**。

TCP 报文段格式

在计算校验和时，临时把 12 字节的“伪首部”和 TCP 报文段连接在一起。伪首部仅仅是为了计算校验和。

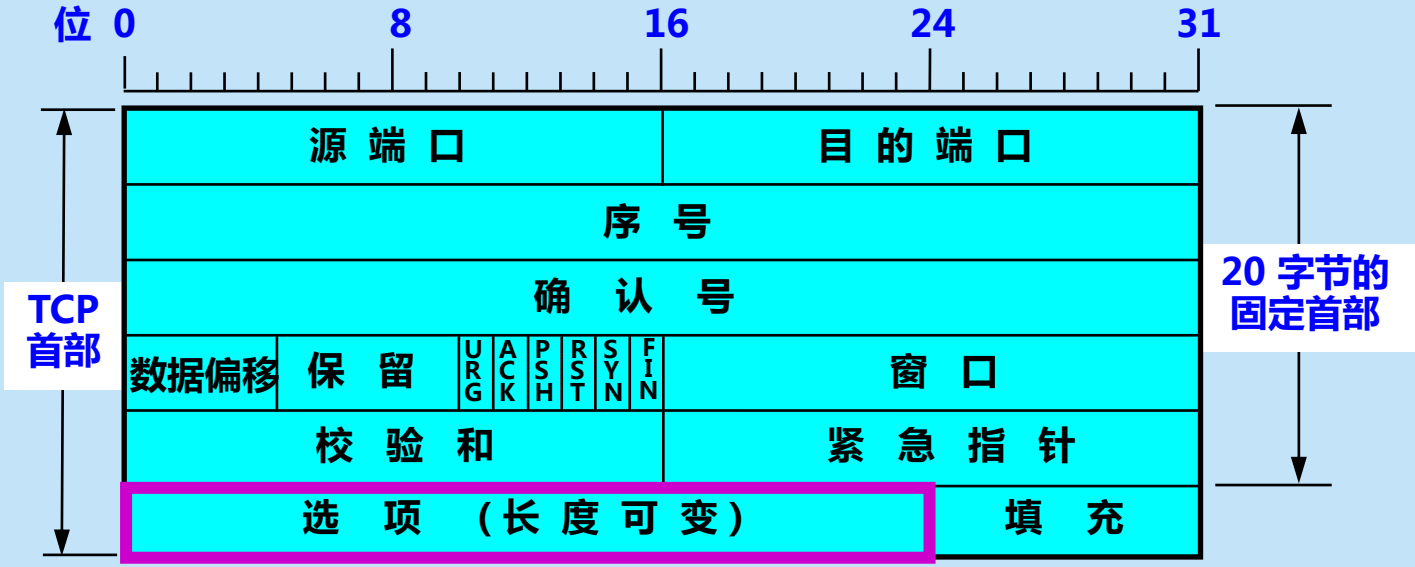


TCP 报文段格式



紧急指针：占 2 字节。在 URG = 1 时有效，指出本报文段中的紧急数据的字节数（紧急数据结束后就是普通数据），指出了紧急数据的**末尾**在报文段中的位置。

TCP 报文段格式



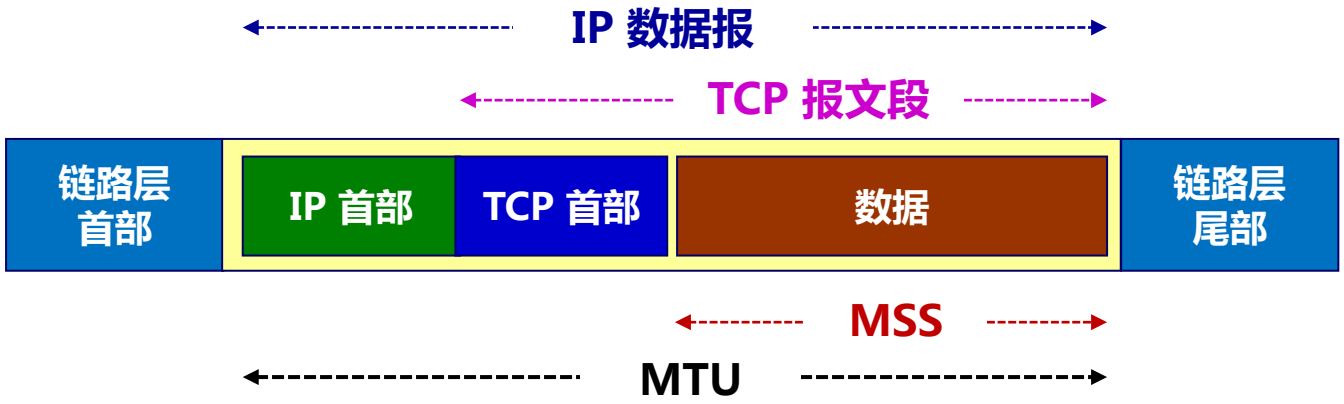
选项：长度可变。TCP 最初只规定了一种选项，即最大报文段长度 MSS。MSS 告诉对方 TCP：“我的缓存所能接收的报文段的数据字段的最大长度是 MSS 个字节。”

填充：使整个 TCP 首部长度是 4 字节的整数倍。

选项+填充： 最长为 40 字节。

TCP 选项 -- 最大报文段长度(MSS)

- **最大报文段长度 MSS** 是每个 TCP 报文段中的**数据字段**的最大长度。
- **与接收窗口值没有关系。**



TCP 报文段长度 = TCP 首部长度 + MSS
MSS = TCP 报文段长度 – TCP 首部长度

为什么要规定MSS呢？

TCP 选项 -- 最大报文段长度(MSS)

不能太小	不能太大	应尽可能大
◆ 网络利用率降低。 ◆ 例如：仅 1 个字节，利用率就不会超过 1/41。	◆ IP 层传输时要分片，终端要重组。 ◆ 分片传输出错时，要重传整个分组。 ◆ 开销增大。	◆ 只要在 IP 层传输时不再分片。 ◆ 默认值= 536 字节。 • 报文段长度 = 536 + 20 = 556 字节。 • IP 数据报长度 = 576 字节。

TCP 选项

- 窗口扩大
- 时间戳
- 选择性确认

TCP 连接管理

- TCP 是面向连接的协议，连接是用来传送TCP报文段的。
- TCP 连接管理的目的是要保证运输连接的建立和释放均能正常进行。

如何打招呼？如何说再见？

- TCP 连接有三个阶段
  - 连接建立 —— 交流之前打招呼。
  - 数据传送 —— 信息交流。
  - 连接释放 —— 交流完成说再见。

类似的例子——购物前与卖家交流：发什么物流？有什么优惠？是否有赠品？

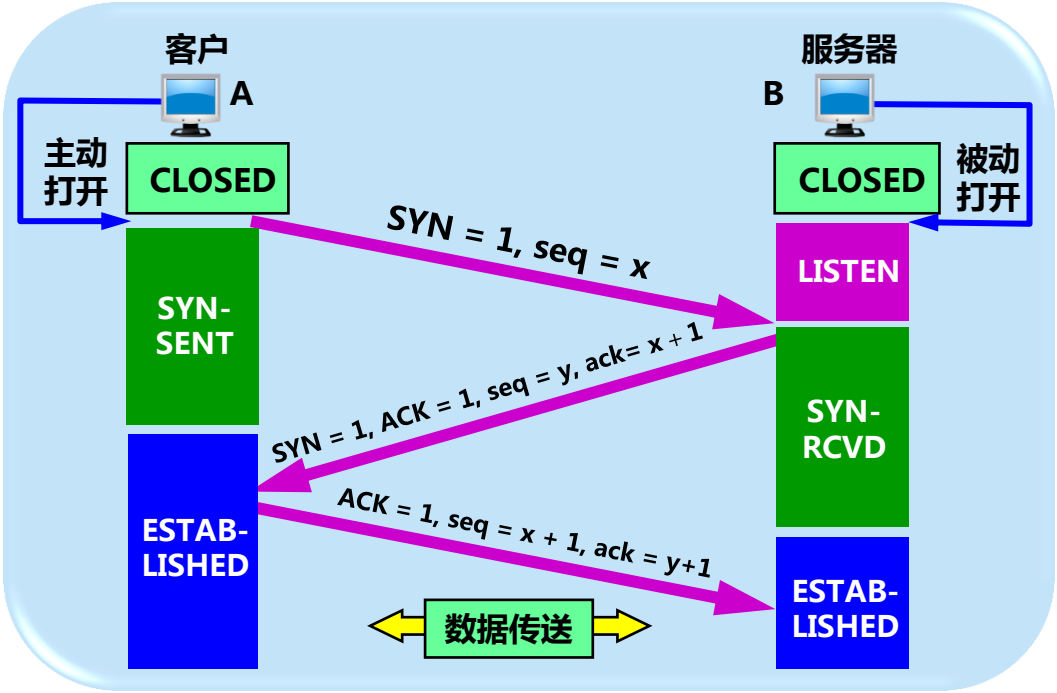
- TCP 连接建立过程中要解决的问题：
  - 要使每一方能够确知对方的存在。
  - 要允许双方协商一些参数（如最大窗口值、是否使用窗口扩大选项和时间戳选项以及服务质量等）。
  - 能够对运输实体资源（如缓存大小、连接表中的项目等）进行分配。

TCP 连接管理

- TCP 连接的建立采用客户-服务器方式。
  - 主动发起连接建立的应用进程叫做客户 (client)。
  - 被动等待连接建立的应用进程叫做服务器 (server）。
- TCP 规定
  - SYN 报文段（即 SYN = 1 的报文段）不能携带数据，但要分配一个序号。
  - ACK报文段可以携带数据，若不携带数据，则不分配序号。
  - FIN 报文段（即 FIN = 1 的报文段）即使不携带数据，也要分配一个序号。

TCP 连接的建立

- TCP 建立连接的过程叫做**握手**。
- 握手需要在客户和服务器之间**交换三个TCP报文段**，称之为**三报文握手**。



第一个报文：
A 的 TCP 向 B 发出连接请求报文段：首部中的 SYN = 1，seq = x，表明传送数据时的第一个数据字节的序号是 x。

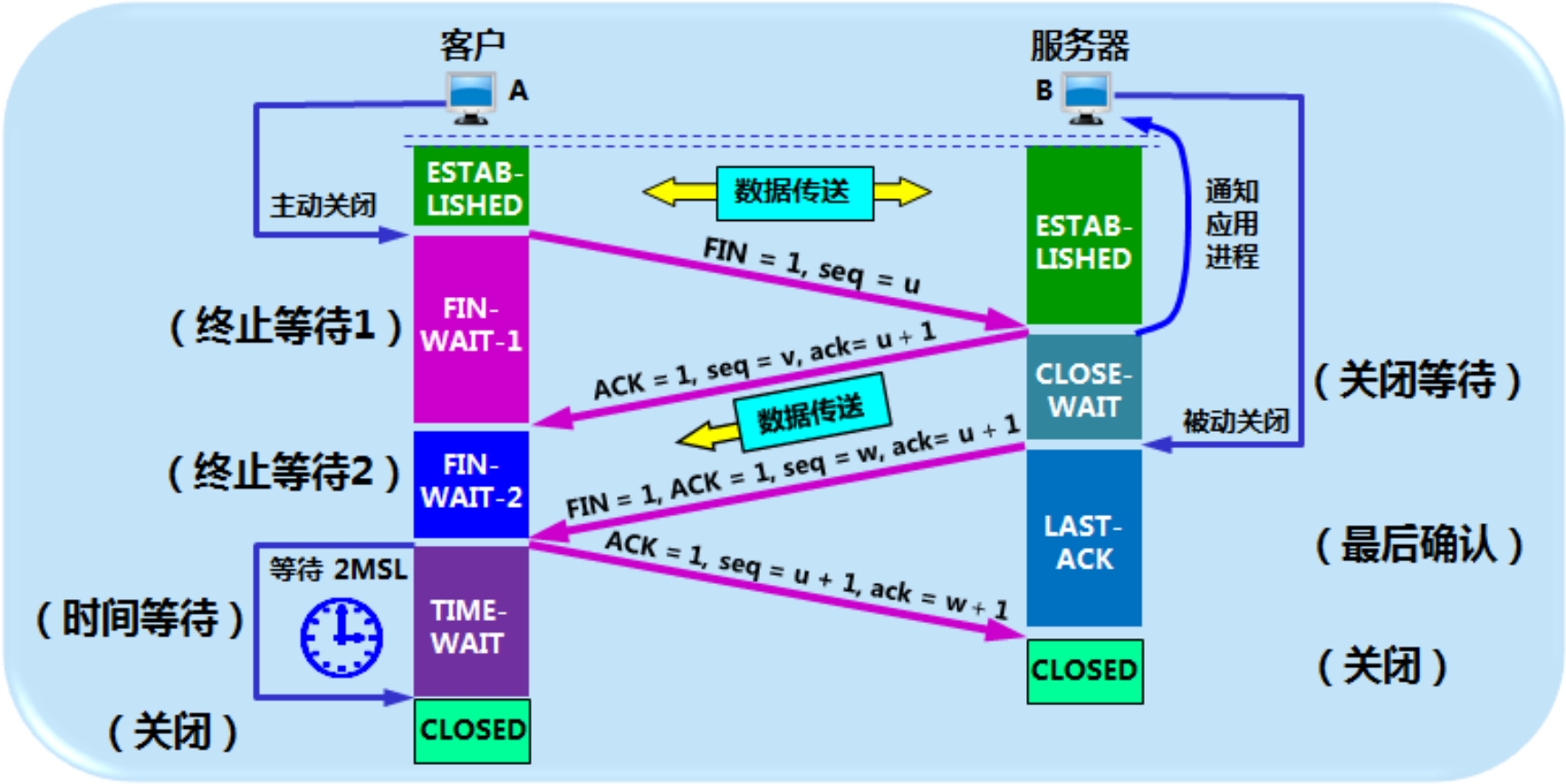
第二个报文：
B 收到连接请求报文段后，若同意，返回确认。确认报文段中的 SYN = 1，ACK = 1，其确认号 ack = x +1，自己的序号 seq=y。

第三个报文：
A 收到 B 的确认报文后，向 B 发出确认，ACK = 1，确认号 ack = y + 1。A 的 TCP 通知上层应用进程，连接已经建立。

采用三报文握手主要是为了防止已失效的连接请求报文段突然又传送到了，从而产生错误。

TCP 连接的释放

- TCP 连接释放过程是四报文握手。



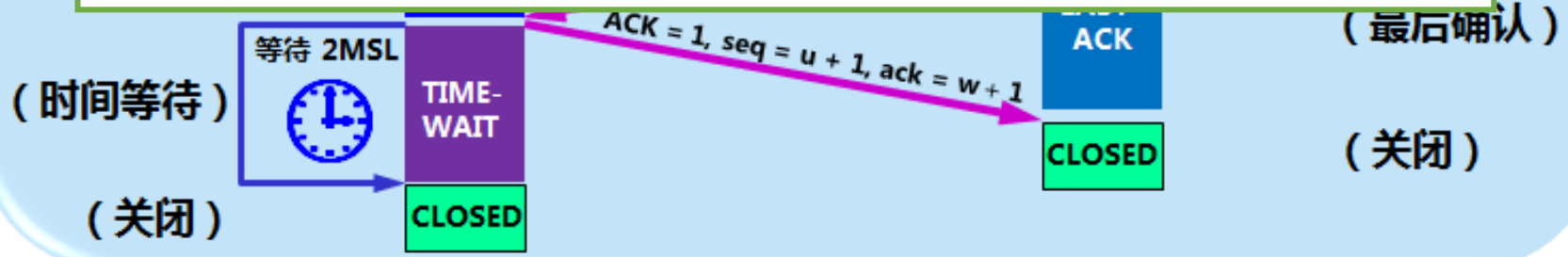
MSL（Maximum Segment Lifetime）：最长报文段寿命

TCP 连接的释放

- TCP 连接释放过程是**四报文握手**。

必须等待 2 MSL：

- MSL 由**时间等待计时器**设定，必须等待的原因是：
  - 为了保证 A 发送的最后一个 ACK 报文段能够到达 B。
  - 防止 “已失效的连接请求报文段” 出现在本连接中。
- RFC 793 建议设置为 2 分钟，实际应用中常用的是30秒、1分钟和2分钟等。



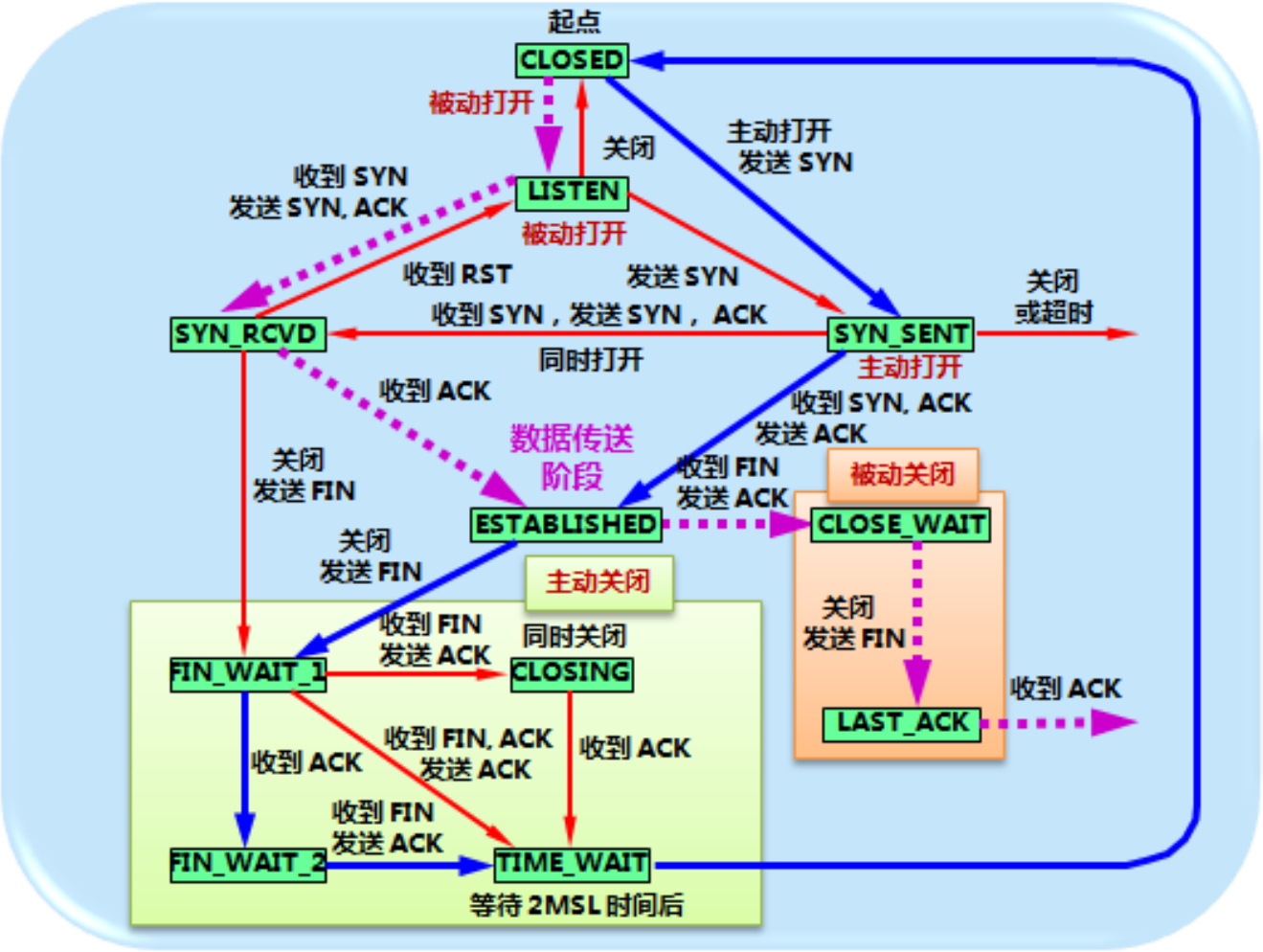
MSL（Maximum Segment Lifetime）：最长报文段寿命

保活计时器

- 用来防止在 TCP 连接出现长时期的空闲。
- 保活计时器通常设置为2小时。
- 若服务器过了2小时还没有收到客户的数据，它就发送探测报文段。若发送了10个探测报文段（每一个相隔75秒）还没有响应，就认为客户的主机出了故障，因而就终止该连接。

TCP 的有限状态机

- 表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。



对客户进程的正常变迁

对服务器进程的正常变迁

异常变迁

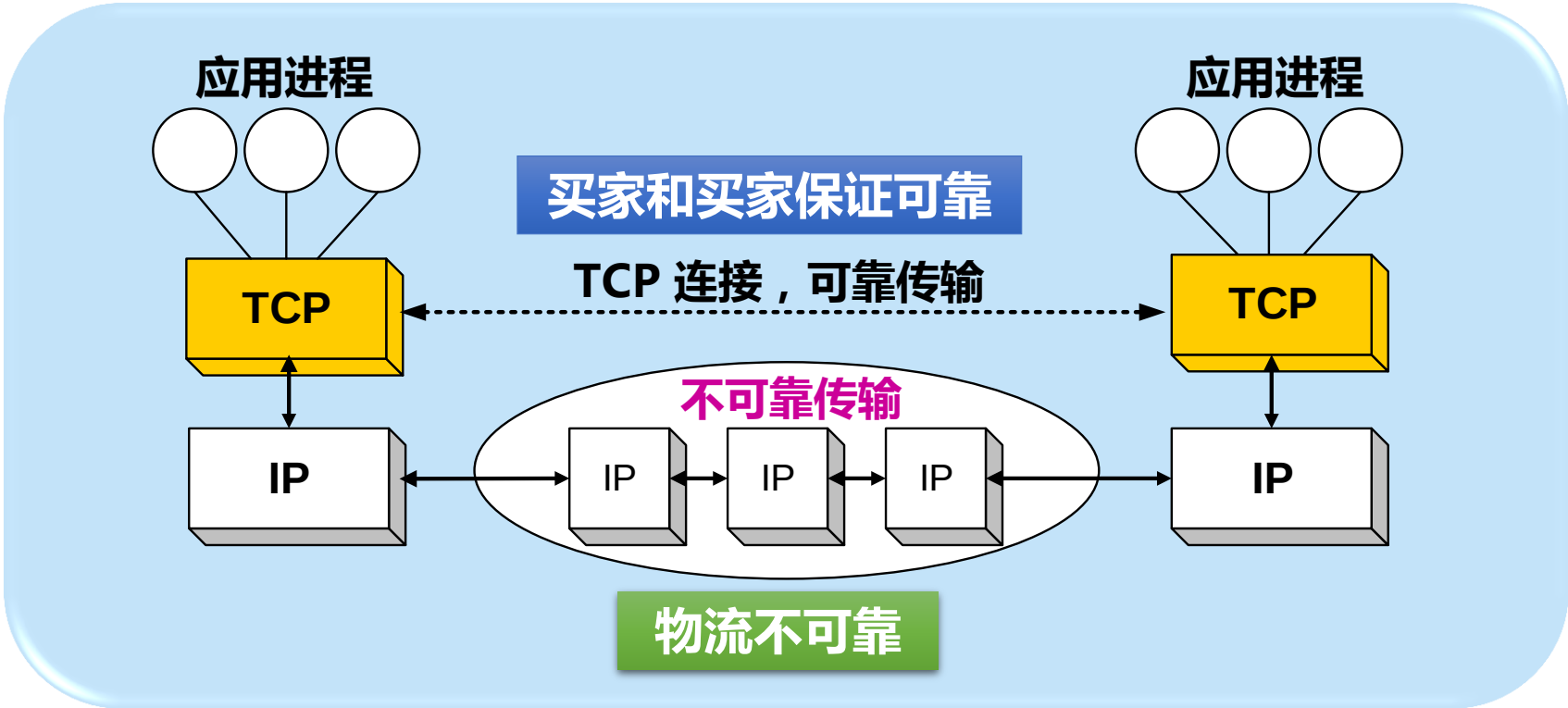
箭头旁边的字表明引起这种变迁的原因，或表明发生状态变迁后又出现什么动作。

TCP 传输控制

- 可靠传输
- 流量控制
- 超时重传

TCP 传输控制 —— 可靠传输

- 类似网购
  - 物流运输网络不可靠；
  - 可靠的买卖由买卖双方保证。



TCP 传输控制 —— 可靠传输

- 理想的传输条件的特点
  - 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。
- 在这样的理想传输条件下，**不需要采取任何措施**就能够**实现可靠传输**。
- 实际的网络都不具备以上两个理想条件。**必须使用一些可靠传输协议**，在不可靠的传输信道**实现可靠传输**。
  - ✓ **自动重传请求** (**ARQ**，Automatic Repeat reQuest)：重传的请求是自动进行的，接收方不需要请求发送方重传某个出错的分组。
    - **停止等待ARQ协议**
    - **连续ARQ协议**
      - **回退N（Go-Back-*N*）ARQ协议**
      - **选择性重传ARQ协议**

停止等待协议要点

- **停止等待**。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。
- **暂存**。在发送完一个分组后，发送方必须暂存已发送的分组的副本，以备重发。
- **编号**。对发送的每个分组和确认都进行编号。
- **超时重传**。发送方为发送的每个分组设置一个超时计时器。若超时计时器超时位收到确认，发送方会**自动**超时重传分组。
- 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**更长一些**，防止不必要的重传。
- **简单，但信道利用率太低。**

连续 ARQ 协议

- 发送方维持一个一定大小的发送窗口，位于窗口内的所有数据都可以连续地发送出去，中途不需要等待对方的确认。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，以便在超时重传时使用。
- 如果收到了接收方发来的确认，则可以继续发送数据。TCP所有的确认都是基于序号而不是基于报文段的。
- 使用滑动窗口协议控制发送方和接收方所能发送和接收的分组的数量和编号，每收到一个确认，发送方就把发送窗口向前滑动。
- 每个 TCP 连接都维持一个超时重传计时器，每发送一个报文段，就设置一次计时器，如果确认超时，则重传这个报文段。

要点：窗口，序号，确认，超时重传 —— 实现可靠传输

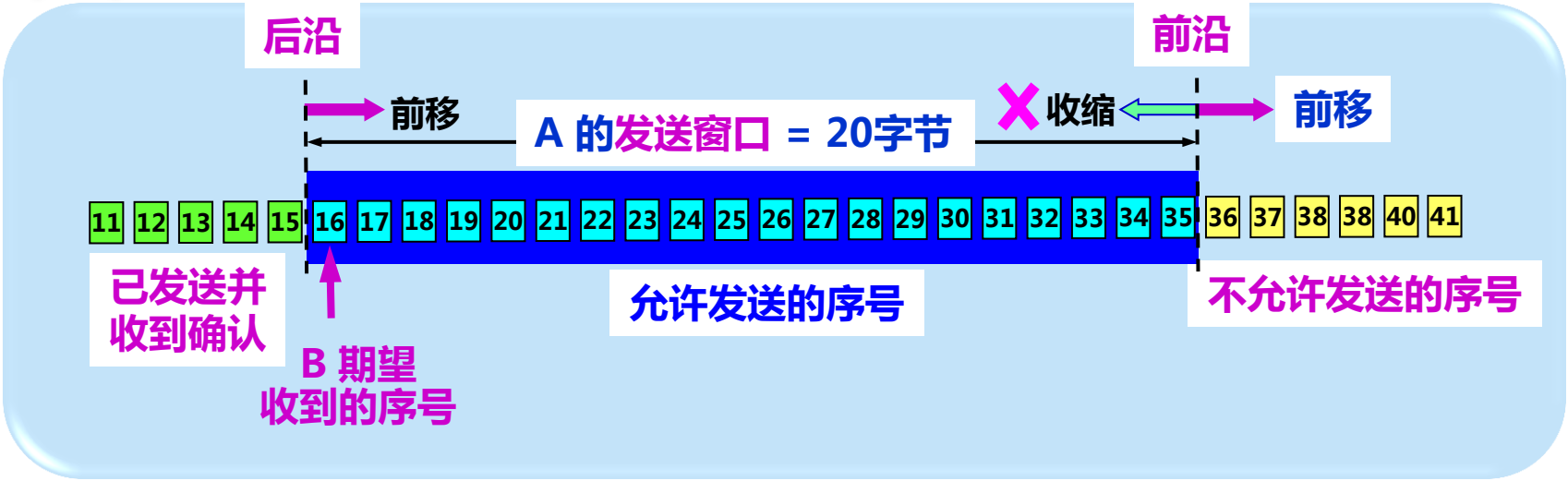
连续 ARQ 协议 -- 以字节为单位的窗口滑动

- TCP连接的两个端点各有两个窗口

窗口经常处于动态变化之中

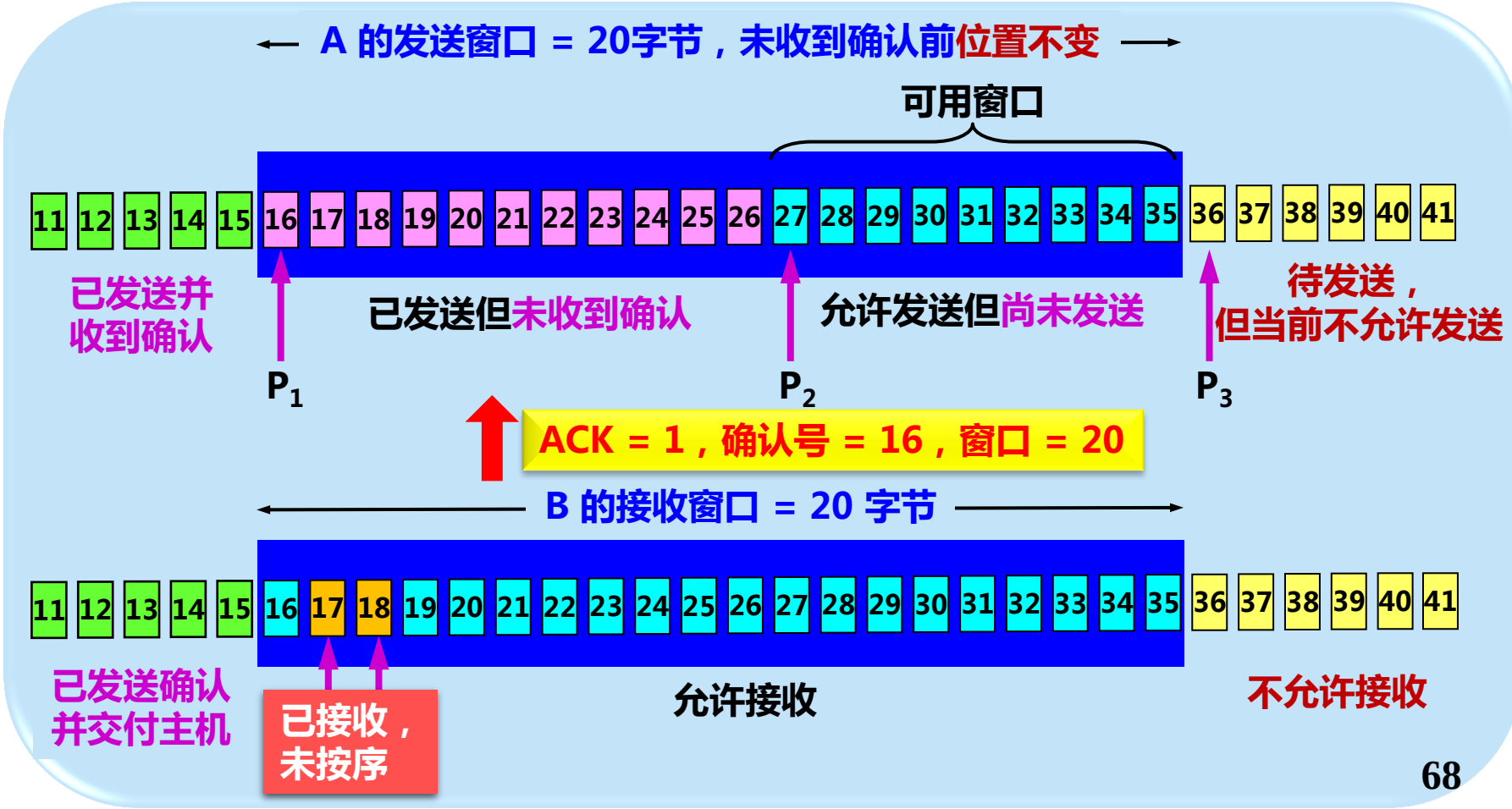
  - **发送窗口**：已发送但未收到确认的数据和准备发送的数据。
  - **接收窗口**：按序到达但未被应用程序接收的数据和未按序到达的数据。

例 假设A收到了B的确认报文段：窗口值20字节，确认号为16，请确定A的发送窗口和可发送的数据。



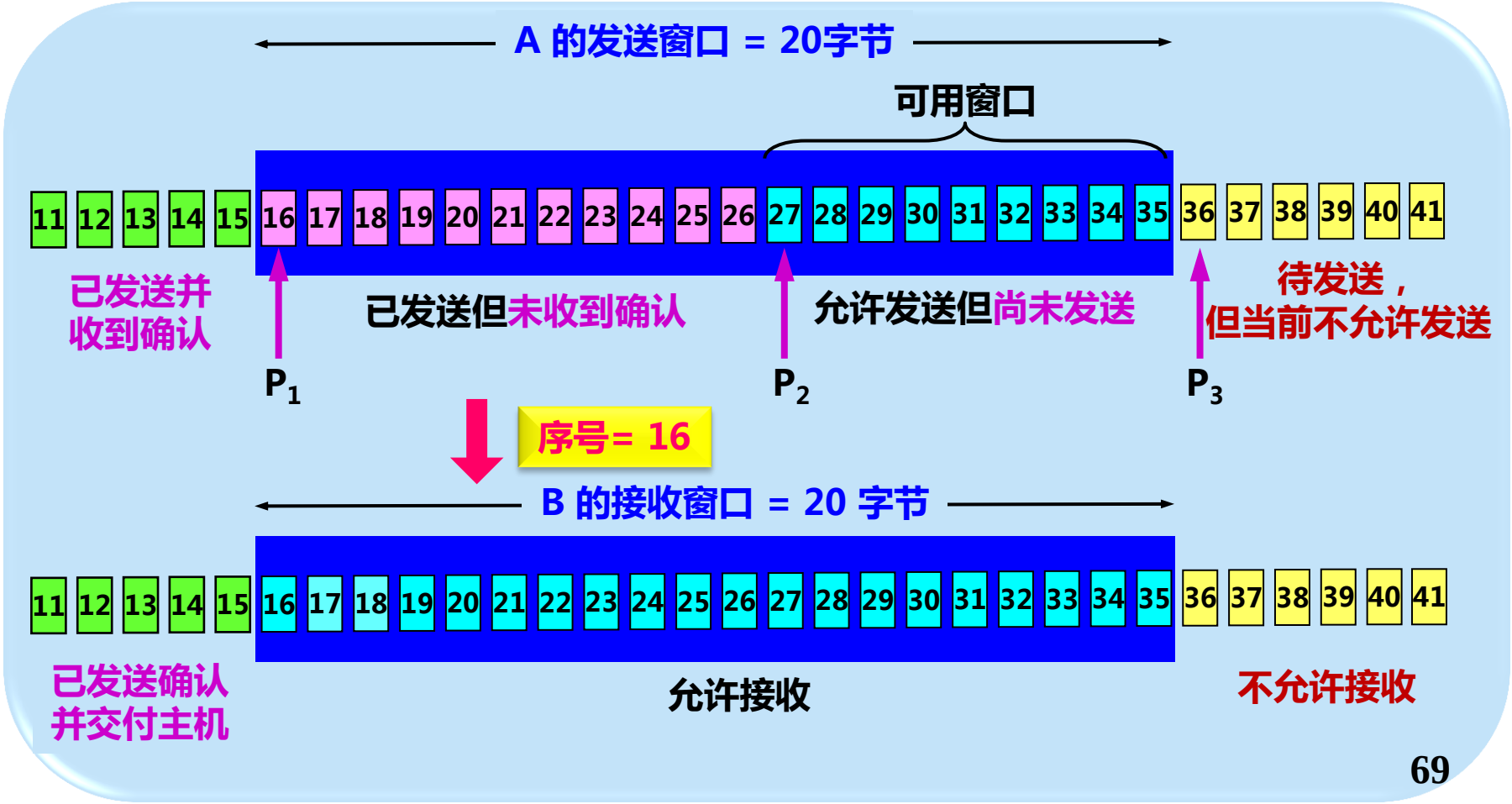
连续 ARQ 协议 -- 以字节为单位的窗口滑动

假设B收到了序号15之前的数据并向A返回了确认，然后A发送序号为16-35的数据，但未收到确认，B收到了序号为17和18的数据。



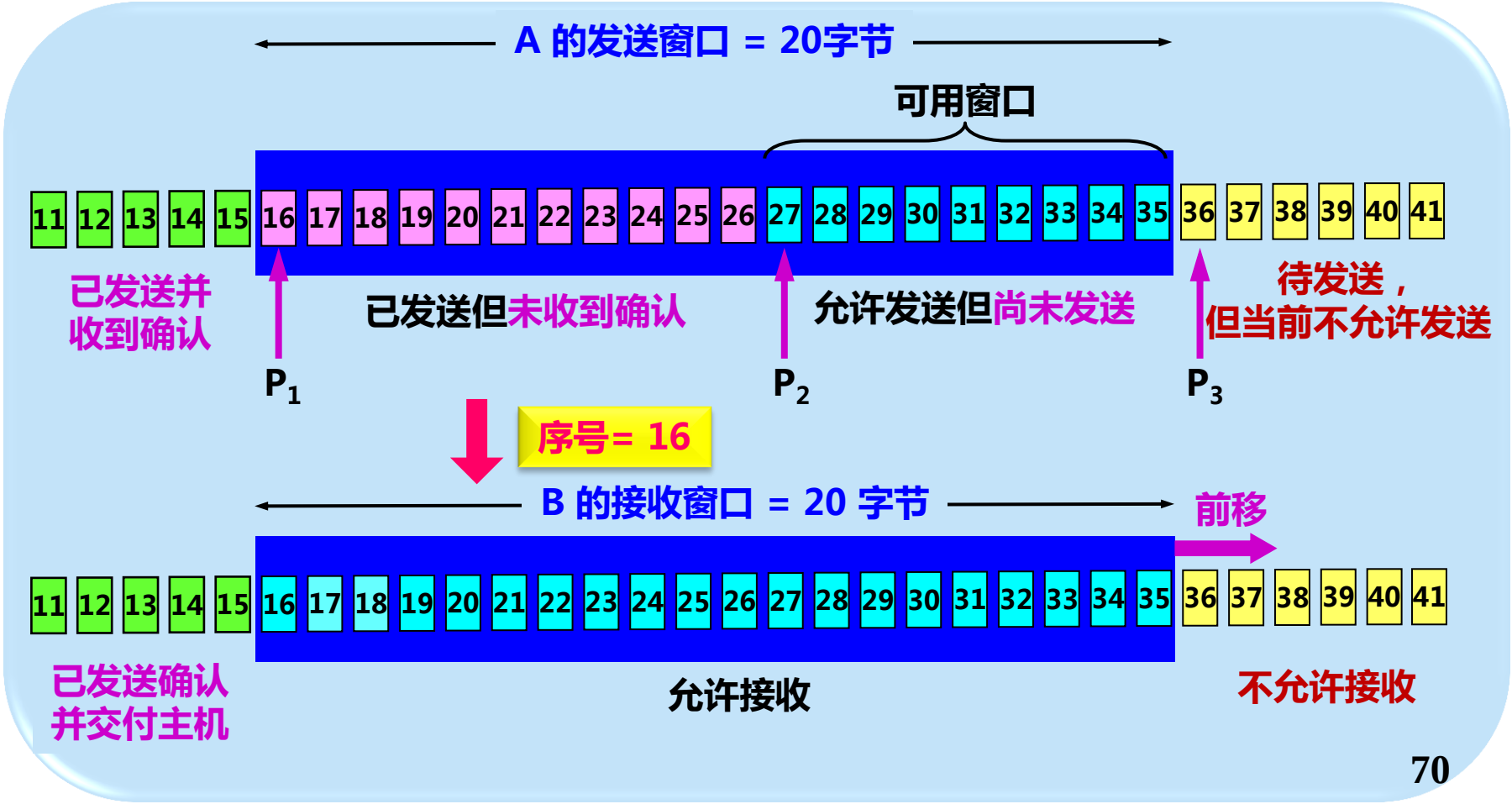
连续 ARQ 协议 -- 以字节为单位的窗口滑动

假设B收到了序号为16的数据，并把序号为16-18的数据交付给主机，然后B从缓存中删除这些数据，把窗口向前移动3个序号。



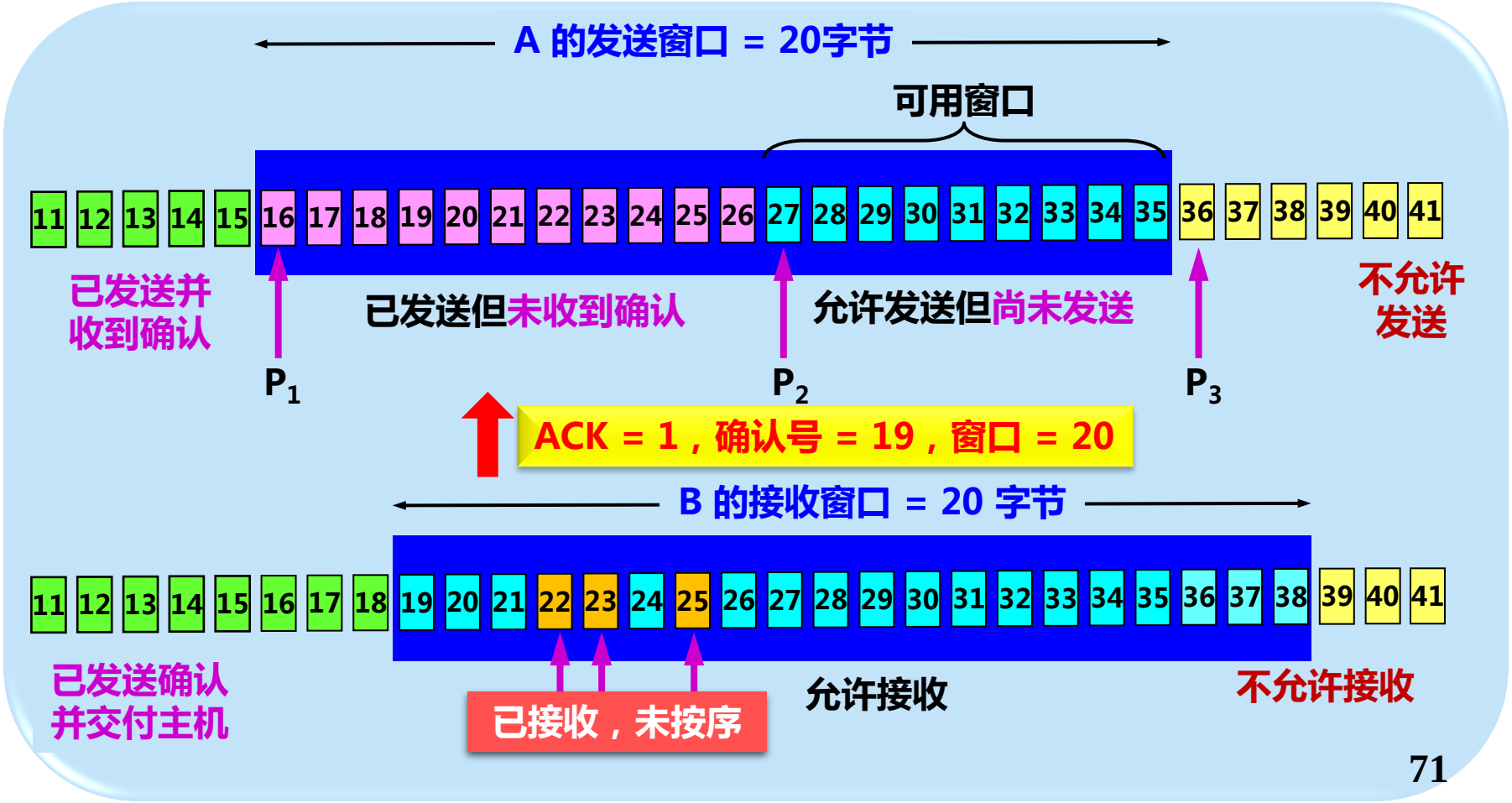
连续 ARQ 协议 -- 以字节为单位的窗口滑动

假设B收到了序号为16的数据，并把序号为16-18的数据交付给主机，然后B从缓存中删除这些数据，把窗口向前移动3个序号。



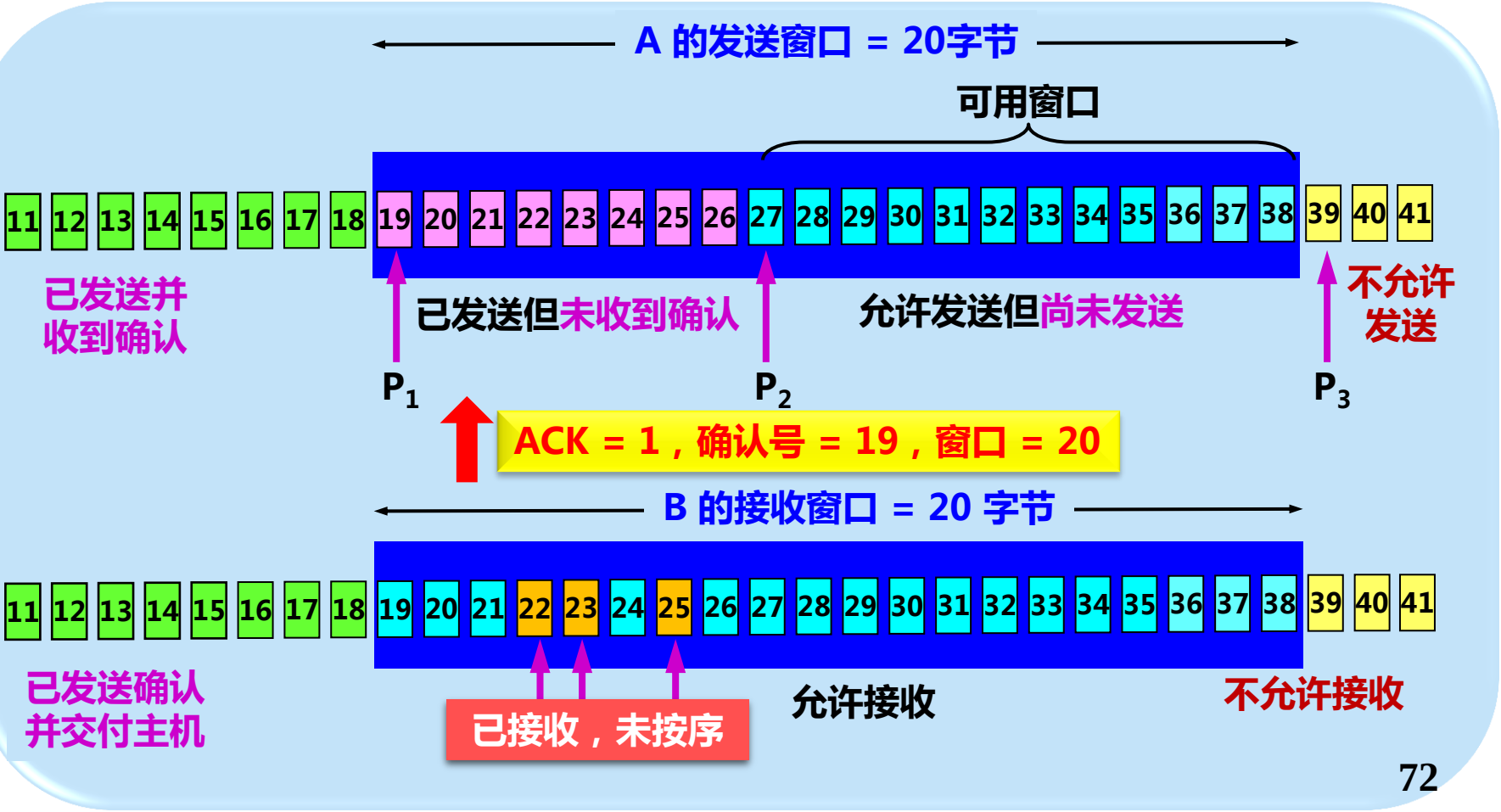
连续 ARQ 协议 -- 以字节为单位的窗口滑动

B收到序号18之前的数据，还收到了未按序到达的序号为22、23和25的数据。A收到B的确认后，将其发送窗口向前移动3个序号，P_2不动。



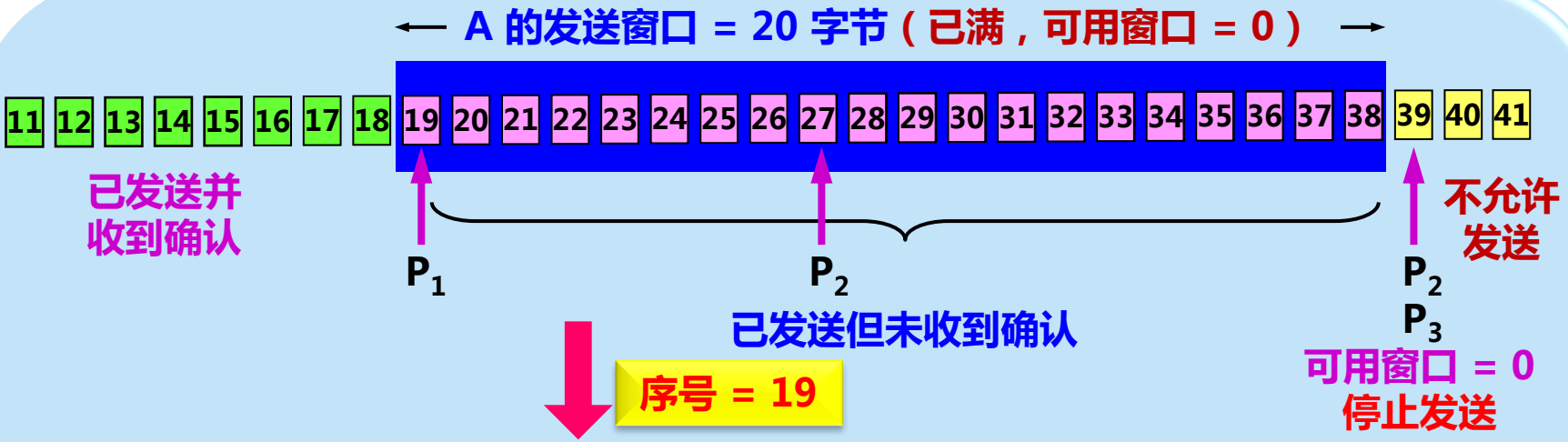
连续 ARQ 协议 -- 以字节为单位的窗口滑动

B收到序号18之前的数据，还收到了未按序到达的序号为22、23和25的数据。A收到B的确认后，将其发送窗口向前移动3个序号，P_2不动。



连续 ARQ 协议 -- 以字节为单位的窗口滑动

A继续发送完序号为27-38的数据后，指针P_2向前移动，和指针P_3重合。发送窗口内的序号都已用完，但还没有再收到确认，停止发送。



- A 未收到确认的原因有：① B 未发送；② B已发送，但还未到达 A 。
- 为保证可靠传输，A 只能认为 B 还没有收到这些数据。A 经过一段时间后（由超时计时器控制）就重传这部分数据，重新设置超时计时器，直到收到 B 的确认为止。
- 如果 A 按序收到落在发送窗口内的确认号， 就使发送窗口向前滑动，并发送新的数据。

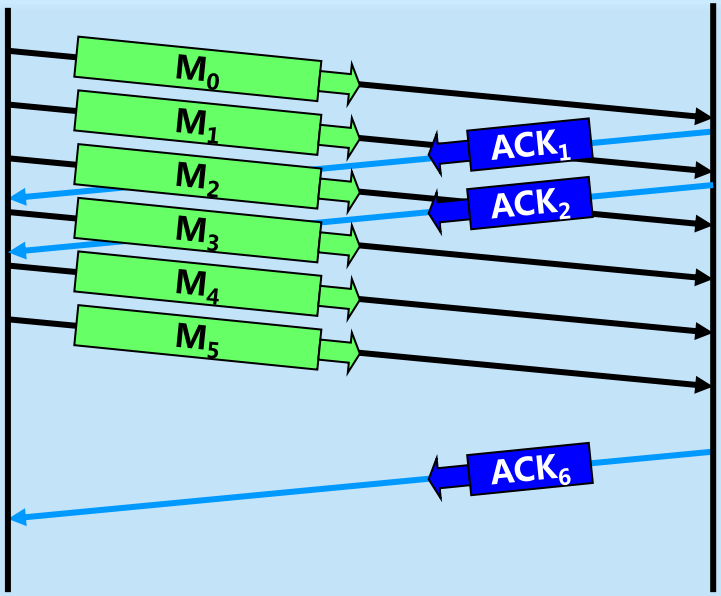
连续 ARQ 协议 -- 以字节为单位的窗口滑动

说明：

- 发送窗口根据接收窗口进行设置，但在同一时刻，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。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。
- TCP 要求接收方必须有累积确认的功能，以减小传输开销。但接收方不应过分推迟发送确认，确认推迟的时间不应超过 0.5秒，否则会导致发送方不必要的重传。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。

连续 ARQ 协议 -- 累积确认

- 接收方不必对收到的分组逐个发送确认，而是在收到几个分组后**对按序到达的最后一个分组发送确认**，表明到这个分组为止的所有分组都已被正确接收。



ACK_1 确认 M_0，将 M_0 提交给上层协议或用户

ACK_2 确认 M_1，将 M_1 提交给上层协议或用户

M_2 正确

M_3 正确

M_4 正确

M_5 正确

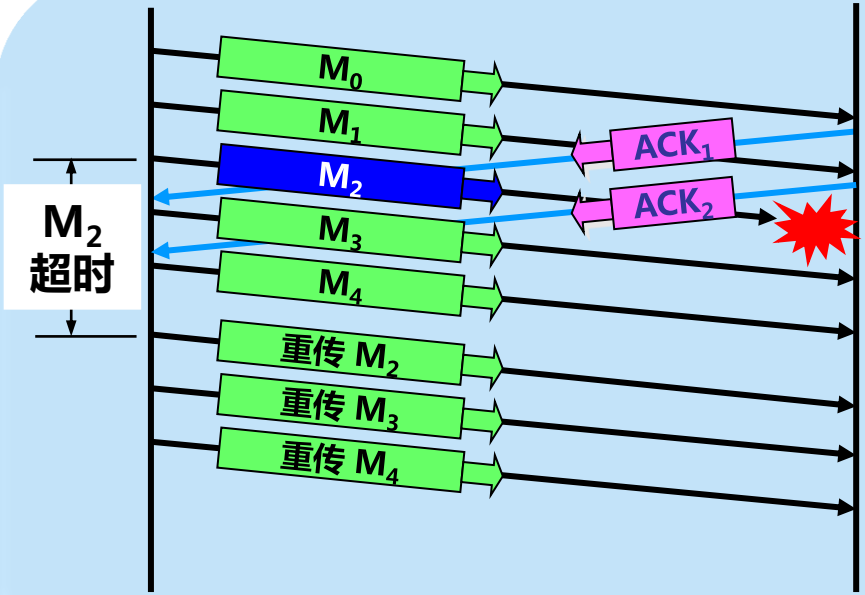
ACK_6 为**累积确认**，表示 M_5 及之前的 M_2、M_3、M_4 都正确。

将M_2、M_3、M_4、M_5提交给上层协议或用户

优点：易于实现，即使确认丢失也不必重传。

连续 ARQ 协议 -- 累积确认

- 假设发送方发送了 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。



ACK_1 确认 M_0，将M_0提交给上层协议或用户

ACK_2 确认 M_1，将M_1提交给上层协议或用户

M_2 丢失

M_3 未按序到达，丢弃

M_4 未按序到达，丢弃

回退N（Go-Back-*N*）ARQ协议

缺点：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。

如果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？

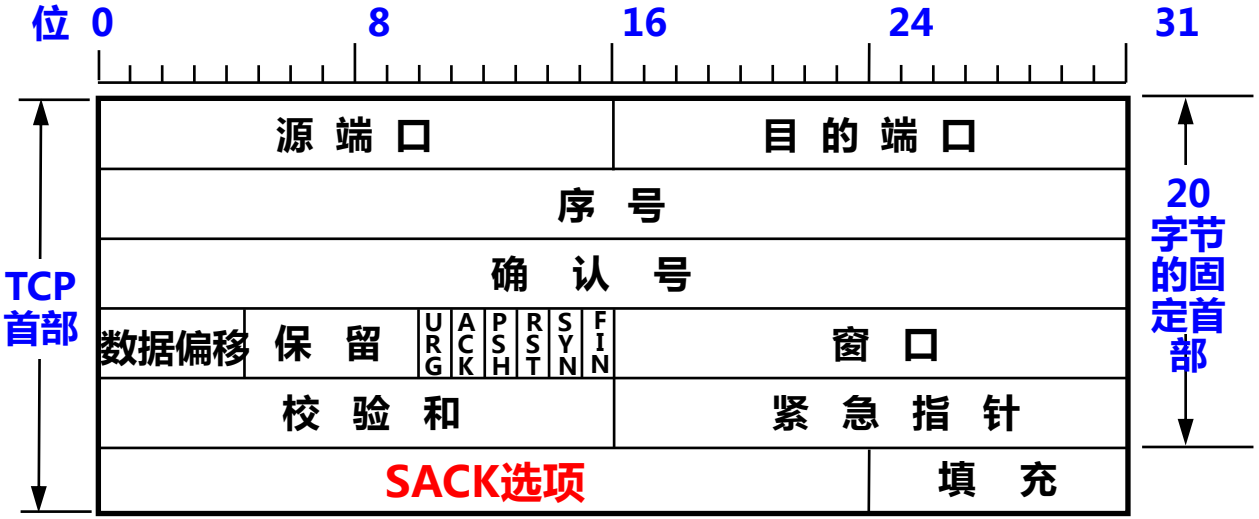
解决方法：选择性确认（SACK）

连续 ARQ 协议 -- 选择性确认（SACK）

- TCP SACK 选项可以告知发送方接收方收到了哪些数据，这样发送方收到这些信息后就会知道哪些数据丢失了，然后立即重传丢失的部分。
- RFC 2018 定义了两种 TCP SACK 选项
  - **SACK 允许选项**：标识是否支持 SACK。
  - **SACK 信息选项**：包含了具体的 SACK 信息。

连续 ARQ 协议 -- 选择性确认（SACK）

- **SACK 允许选项**
  - 在建立 TCP 连接时，通信双方在 SYN 报文段或 SYN+ACK 报文段中添加 SACK允许 选项，通知对方本端是否支持SACK。
  - 原来首部中的“确认号”字段的用法仍然不变（累积确认），只是在 TCP 首部中都增加了 SACK允许 选项，以便报告收到的不连续的字节块的边界。



类别 = 4	长度 =2

连续 ARQ 协议 -- 选择性确认（SACK）

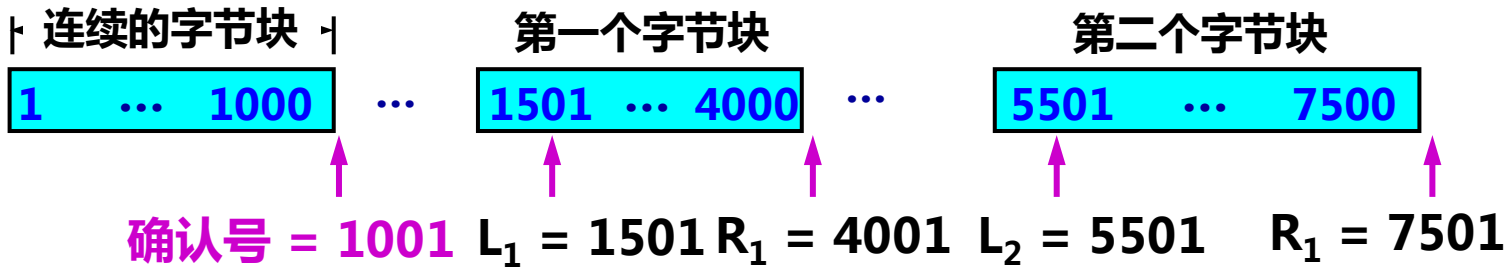
- **SACK 信息选项**
  - 连接建立后，接收方可以通过 SACK 选项告知发送方已经接收并缓存的不连续的字节块。
  - 首部选项长度最大40字节，指明一个字节块用8字节，选项中最多只能指明4个字节块的边界信息。

左边界 = 第一个字节的序号
右边界 = 最后一个字节序号 + 1

连续 ARQ 协议 -- 选择性确认（SACK）

例

假设接收方收到了发送方发过来的和前面的字节块不连续的两个字节块，通过 SACK 信息选项，接收方应该告诉发送方哪些信息？



确认号 =1001

类别 = 5	长度 =18
1501	
4001	
5501	
7501	

小结：连续 ARQ 协议与停止等待协议对比

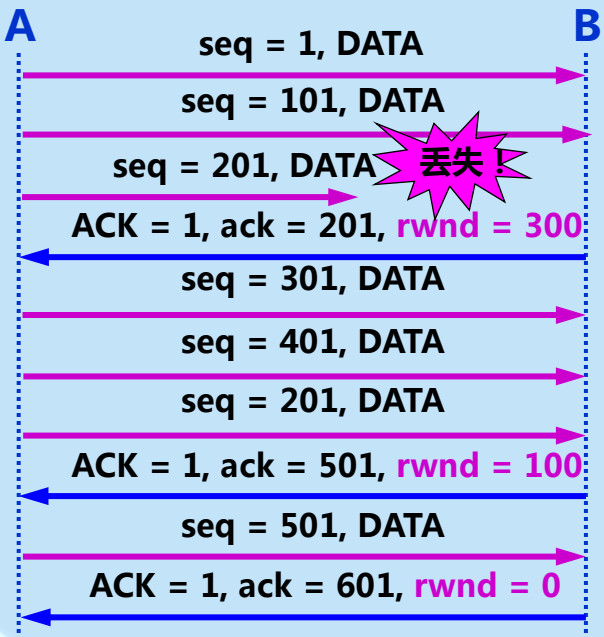
	连续ARQ协议	停止等待协议
发送的分组数量	一次发送多个分组	一次发送一个分组
传输控制	滑动窗口协议	停止、等待
确认	单独确认，累积确认， 累积确认+选择性确认	单独确认
超时计时器	每个发送的分组	每个发送的分组
序号	每个发送的分组	每个发送的分组
重传	回退N个分组，多个分组	一个分组

TCP 传输控制 —— 流量控制

- **流量控制**：让发送方的发送速率不要太快，既要让接收方来得及接收，也不会使网络发生拥塞。
- 利用**滑动窗口机制**可以在 TCP 连接上实现对发送方的流量控制。

例

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

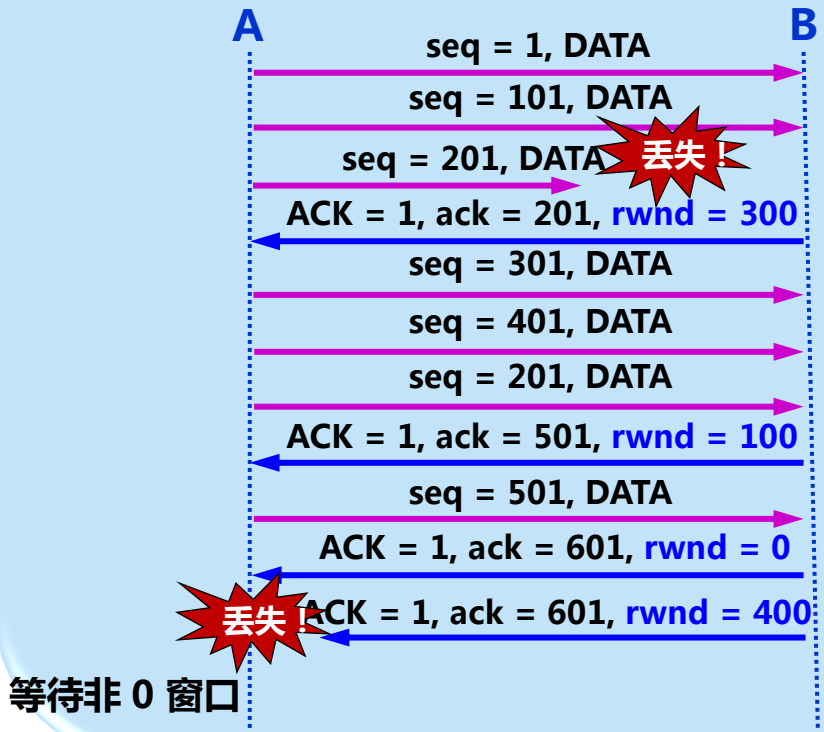


A 发送了序号 1 至 100，还能发送 300 字节
A 发送了序号 101 至 200，还能发送 200 字节

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）

TCP 传输控制 —— 流量控制

A向B发送数据。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

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）
允许 A 发送序号 601 至 1000 共 400 字节
等待 A 发送

死锁问题

TCP 传输控制 —— 流量控制

解决方法：设置持续计时器 (persistence timer) ：

- 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，就可以打破死锁僵局。

TCP 传输控制 —— 超时重传

- 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了，但还没有收到确认，就要重传这一报文段。
- 超时重传时间（RTO）的设置
  - 不能太短，否则会引起很多报文段的不必要重传，增大网络负荷。
  - 不能过长，会使网络的空闲时间增大，降低了传输效率。
- 根据往返时间（RTT）设置 RTO。
  - TCP保留了RTT的一个加权平均往返时间 RTT_S（又称为平滑的 RTT）

$$RTT_{S\_new} = (1 - \alpha) \times RTT_{S\_old} + \alpha \times RTT_{new}$$

式中，$0 \le \alpha < 1$。若 $\alpha \to 0$，表示 RTT 值更新较慢；若 $\alpha \to 1$，则表示 RTT 值更新较快。

RFC 6298 推荐的 α 值为 1/8，即 0.125。

TCP 传输控制 —— 超时重传

- RTO 应略大于加权平均往返时间 RTT_S。
- RFC 6298 建议使用如下方法计算 RTO：

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

式中，RTT_D 是 RTT 的偏差的加权平均值。

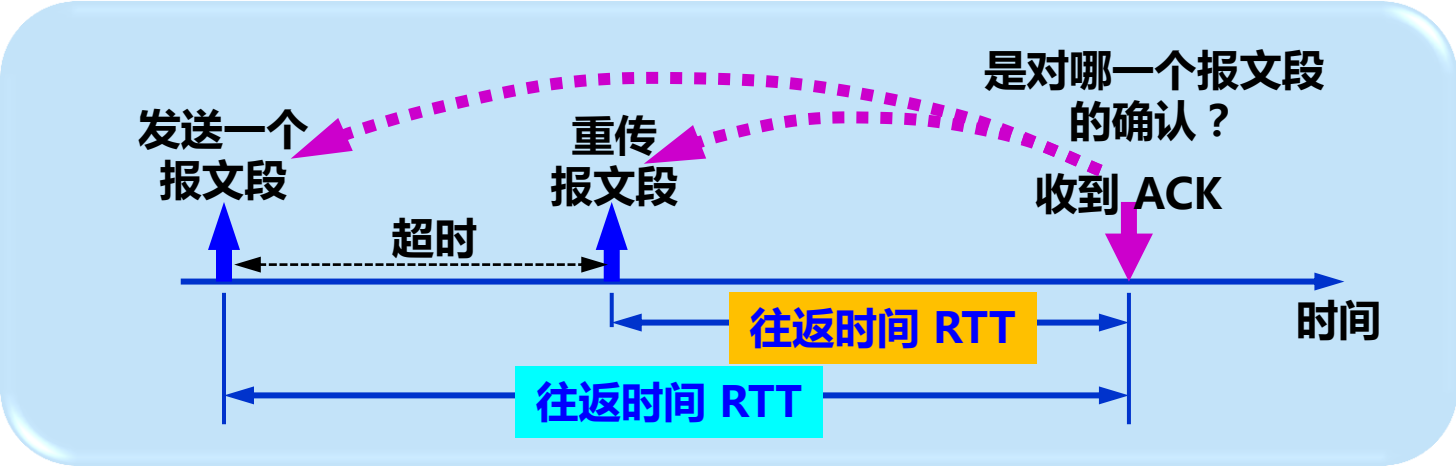
- RFC 6298 建议这样计算RTT_D。第一次测量时，RTT_D 值取为测量到的 RTT 样本值的一半。在以后的测量中，则使用下式计算加权平均的 RTT_D：

$$RTT_{D\_new} = (1 - \beta) \times RTT_{D\_old} + \beta \times |RTT_S - RTT_{new}|$$

式中，β 是个小于 1 的系数，其推荐值是 1/4，即 0.25。

TCP 传输控制 —— 超时重传

- 往返时间（RTT）的测量相当复杂。
  - TCP 报文段没有收到确认，重传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的确认，还是对重传的报文段的确认？



Karn算法：在计算平均往返时间 RTT 时，只要报文段重传了，就不采用其往返时间样本，这样得出的加权平均往返时间 RTT_S 和超时重传时间 RTO 就较准确。

TCP 传输控制 —— 超时重传

- **新问题：**当报文段的时延突然**增大**很多时，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，造成很多不必要的重传。**

修正的 Karn 算法：

- 报文段**每重传一次，就把 RTO 增大一些，即**

$$RTO_{new} = \gamma \times RTO_{old}$$

- 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 和超时重传时间 RTO 的数值。
- **实践证明，这种策略较为合理。**