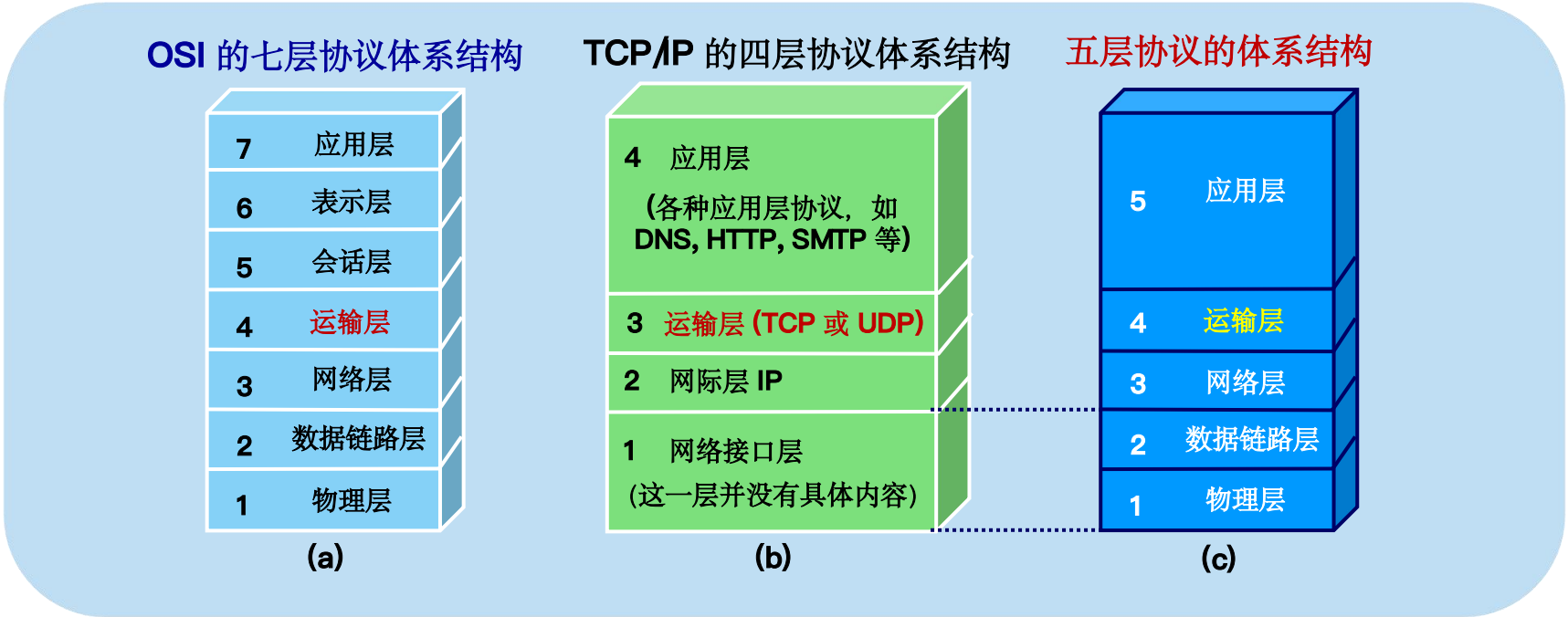
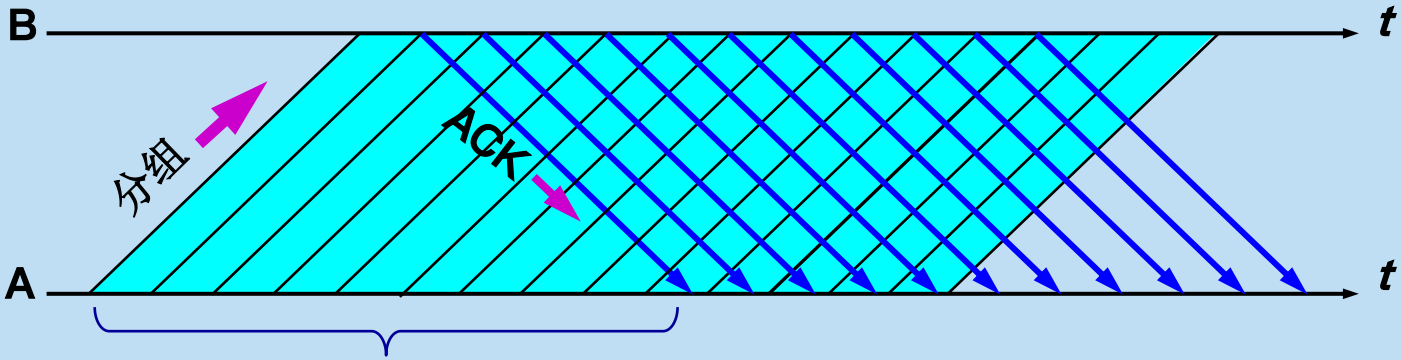
第5章
运输层

计算机网络体系结构


OSI 的七层协议体系结构
7 应用层
6 表示层
5 会话层
4 运输层
3 网络层
2 数据链路层
1 物理层
(a)
TCP/IP 的四层协议体系结构
4 应用层
(各种应用层协议，如 DNS, HTTP, SMTP 等)
3 运输层 (TCP 或 UDP)
2 网际层 IP
1 网络接口层
(这一层并没有具体内容)
(b)
五层协议的体系结构
5 应用层
4 运输层
3 网络层
2 数据链路层
1 物理层
(c)

提高传输效率：流水线传输

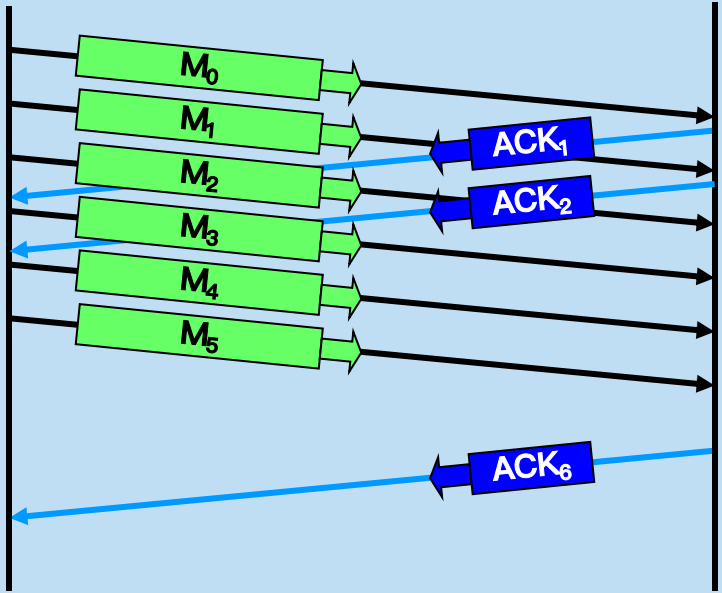


流水线传输：在收到确认之前，发送方连续发出多个分组。

由于信道上一直有数据不间断地传送，
流水线传输可获得很高的信道利用率。

连续 ARQ 协议和滑动窗口协议采用流水线传输方式。

累积确认



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

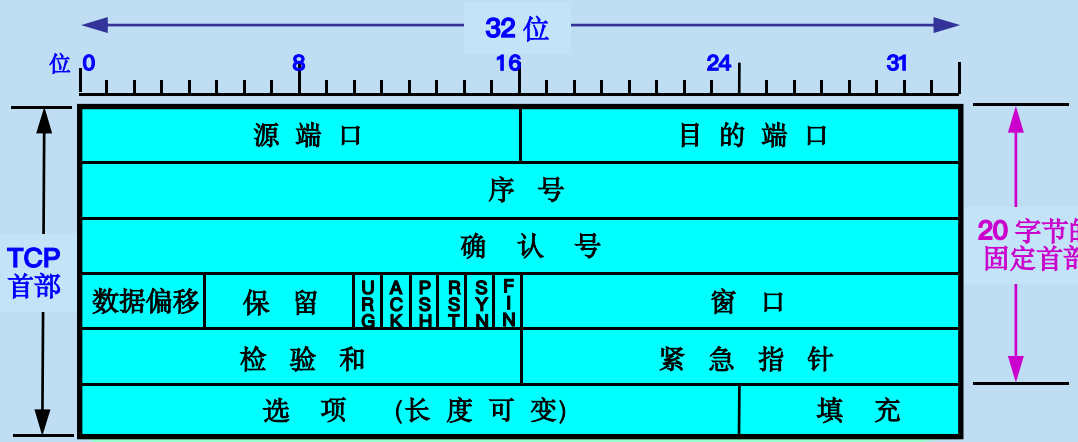
ACK_6 为累积确认，表示 M_5 及之前的 M_2、M_3、M_4 都正确。

将M_2、M_3、M_4、M_5提交给上层协议或用户

5.5 TCP 报文段的首部格式

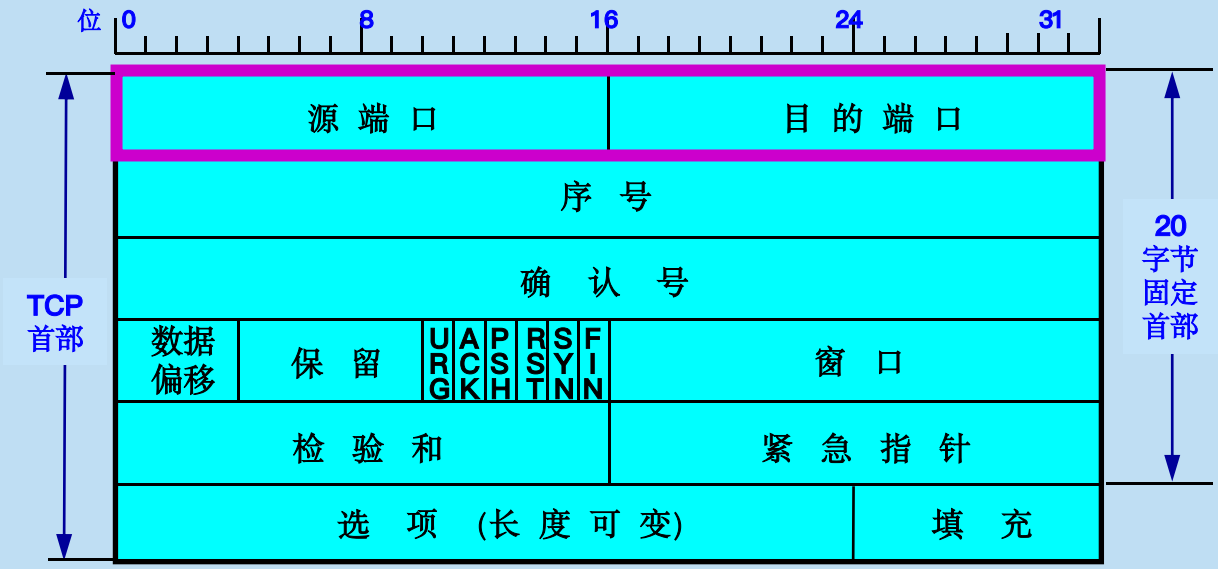
- TCP 虽然是面向字节流的，但 TCP 传送的数据单元却是报文段。
- 一个 TCP 报文段分为首部和数据两部分，而 TCP 的全部功能都体现在它首部中各字段的作用。
- TCP 报文段首部的前 20 个字节是固定的，后面有 4*n* 字节是根据需要而增加的选项 (*n* 是整数)。因此 TCP 首部的最小长度是 20 字节。

5.5 TCP 报文段的首部格式

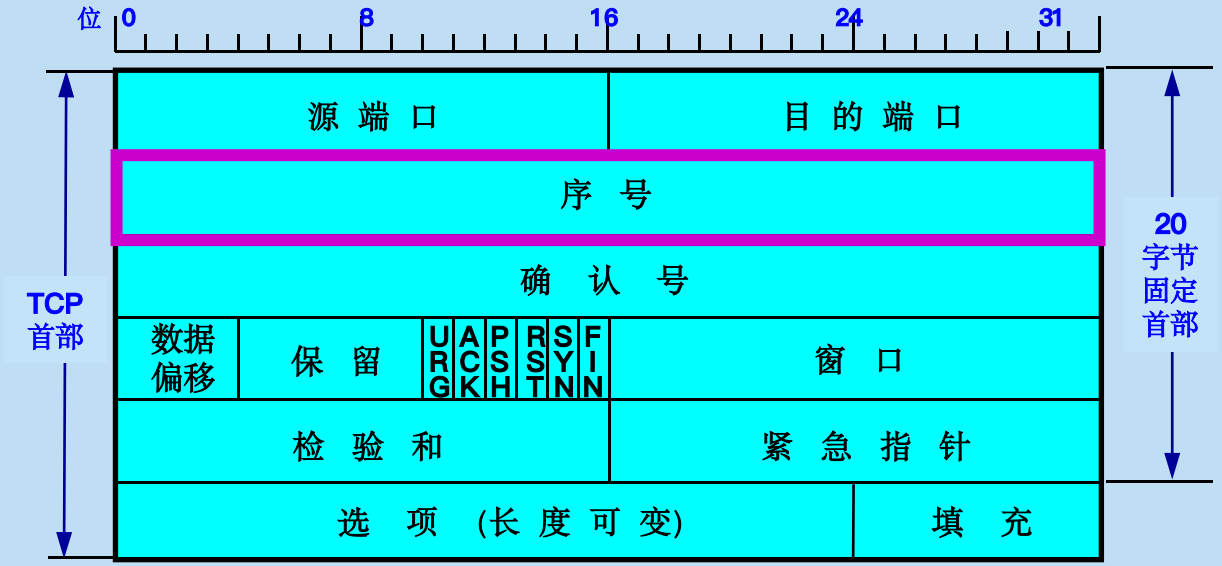


TCP首部的长度是 4n 字节（n 是整数）。

TCP 首部的最小长度是 20 字节。



源端口和目的端口：各占 2 字节。端口是运输层与应用层的服务接口。运输层的复用和分用功能通过端口实现。

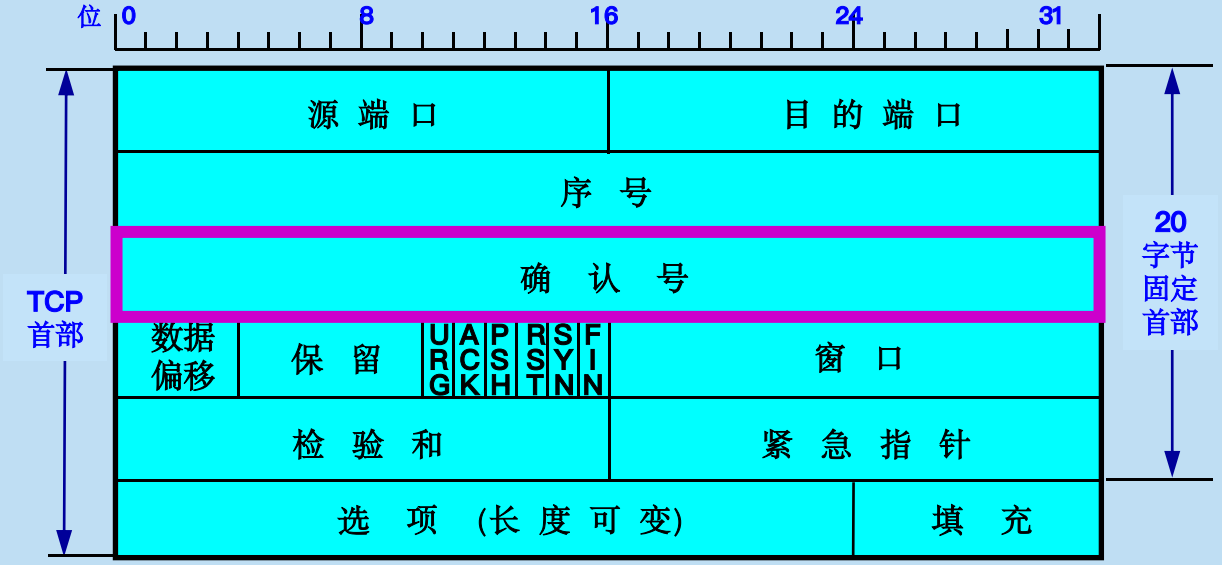


序号：占 4 字节。TCP 连接中传送的数据流中的每一个字节都有一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。

现有 5000 个字节的数据。

假设报文段的最大数据长度为 1000 个字节，初始序号为 1001 。

报文段 1 序号 = 1001　(数据字节序号：1001 ~ 2000)
报文段 2 序号 = 2001　(数据字节序号：2001 ~ 3000)
报文段 3 序号 = 3001　(数据字节序号：3001 ~ 4000)
报文段 4 序号 = 4001　(数据字节序号：4001 ~ 5000)
报文段 5 序号 = 5001　(数据字节序号：5001 ~ 6000)



确认号：占 4 字节，是期望收到对方的下一个报文段的数据的第一个字节的序号。

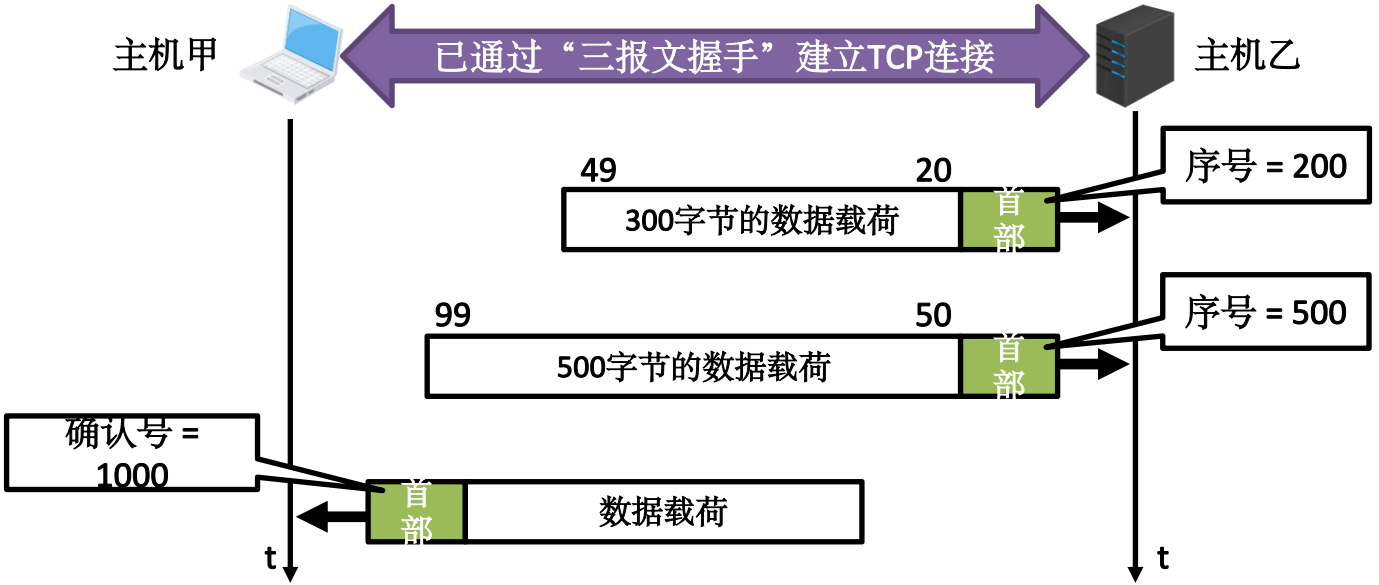
记住：若确认号 = N，则表明：到序号 N −1 为止的所有数据都已正确收到。

01 TCP报文段的首部格式

【2009年 题38】主机甲与主机乙之间已建立一个TCP连接，主机甲向主机乙发送了两个连续的TCP段，分别包 含300字节和500字节的有效载荷，第一个段的序列号为200，主机乙正确接收到两个段后，发送给主机甲的确认序列号是（ D ）。

A. 500　　B. 700　　C. 800　　D. 1000

解析

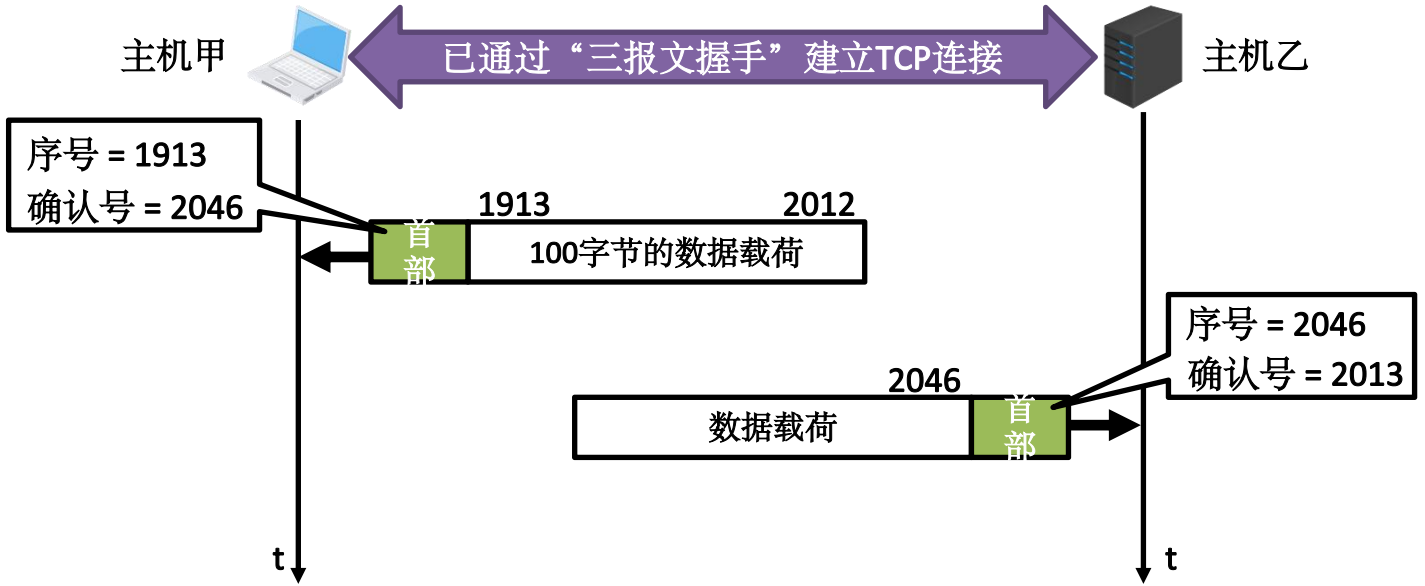


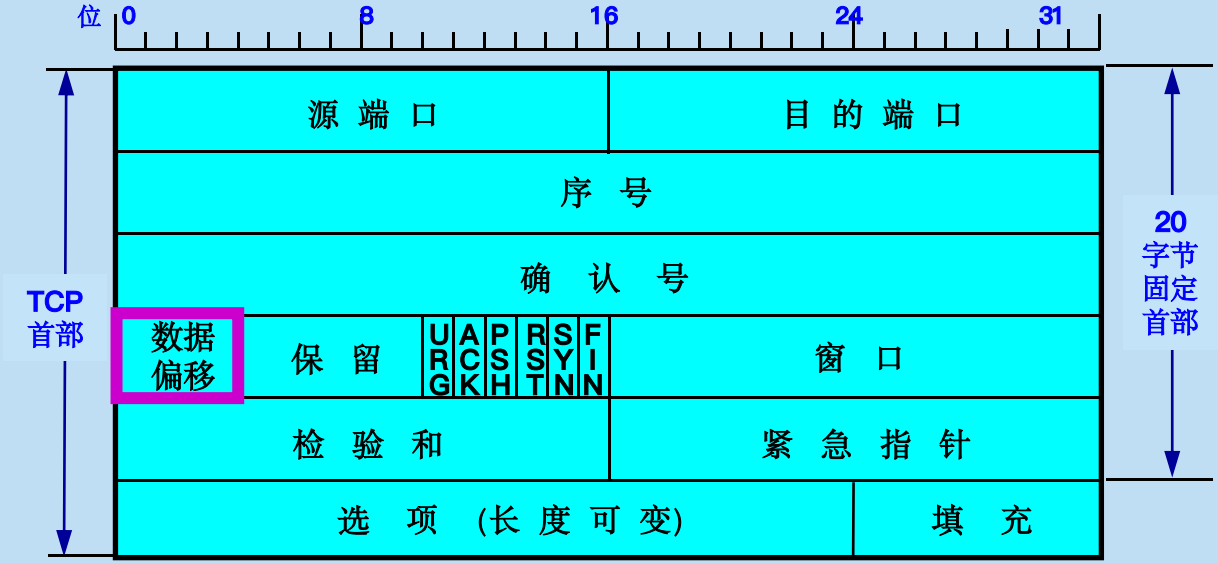
01 TCP报文段的首部格式

【2013年 题39】主机甲与主机乙之间已建立一个TCP连接，双方持续有数据传输，且数据无差错与丢失。若甲收到1个来自乙的TCP段，该段的序号是1913、确认序号为2046、有效载荷为100字节，则甲立即发送给乙的TCP段的序号和确认序号分别是（ B ）。

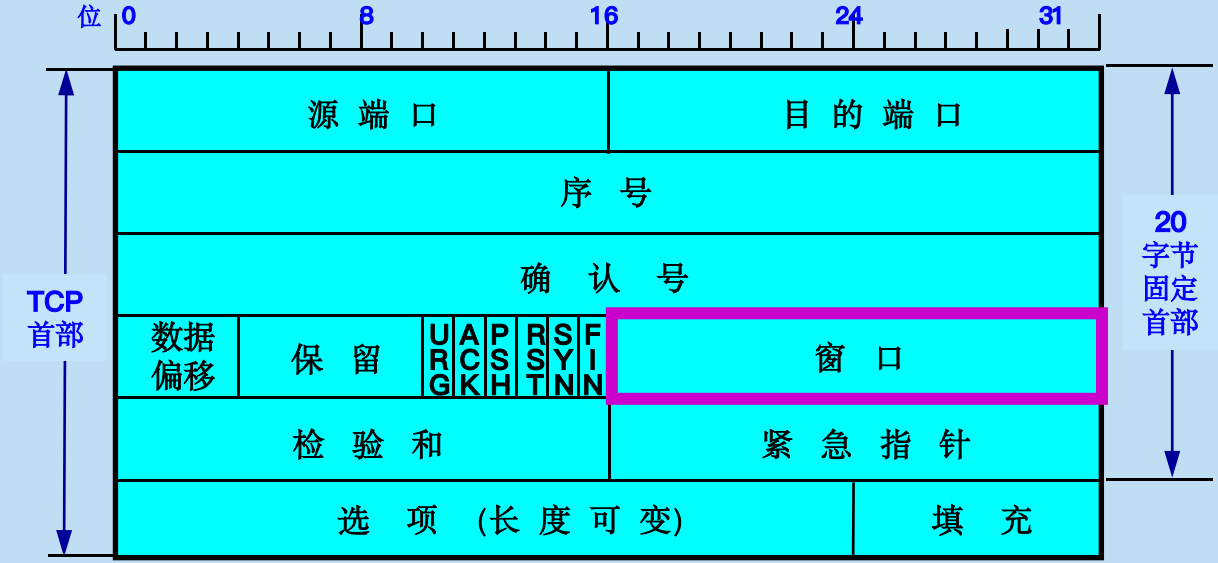
A. 2046、2012　　B. 2046、2013　　C. 2047、2012　　D. 2047、2013

解析





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）。



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）。

记住：窗口字段明确指出了现在允许对方发送的数据量。窗口值经常在动态变化。

选项：长度可变，最长可达 40 字节。

MSS (Maximum Segment Size)

是 TCP 报文段中的数据字段的最大长度。

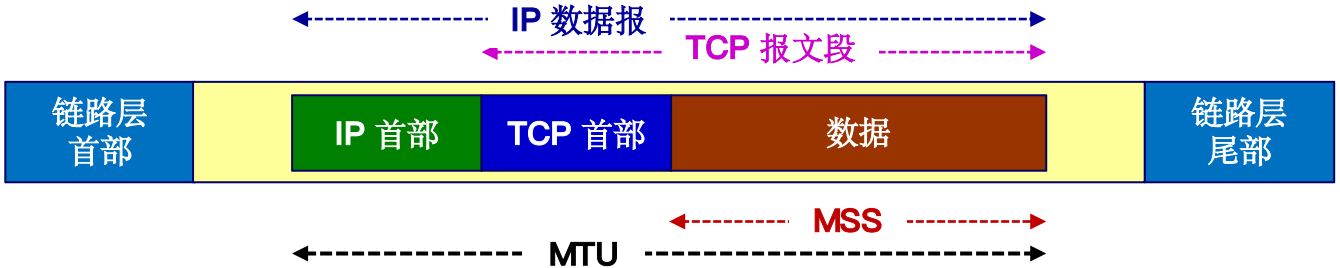
数据字段加上 TCP 首部才等于整个的 TCP 报文段。

所以，MSS是“TCP 报文段长度减去 TCP 首部长度”。

选项：长度可变，最长可达 40 字节。—— 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。”

选项（2）：最大报文段长度 MSS

- 最大报文段长度 MSS（Maximum Segment Size）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。



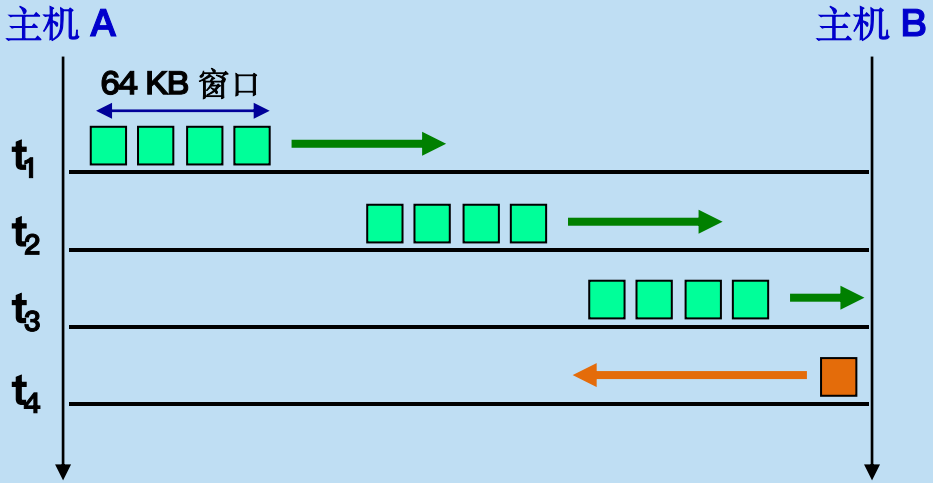
TCP 报文段长度 = 数据字段长度 + TCP 首部长度

数据字段长度 = TCP 报文段长度 −TCP 首部长度

选项（2）：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。	• 开销增大。 • IP 层传输时要分片，终点要装配。 • 分片传输出错时，要整个分组。	• 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。

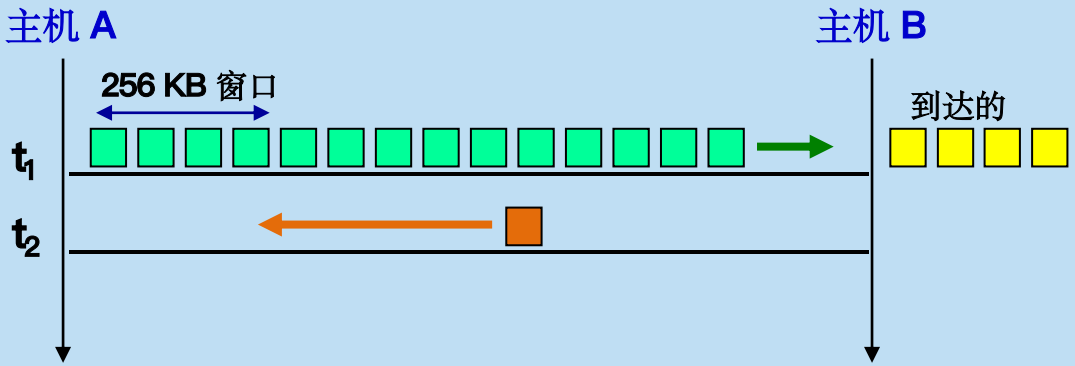
选项（3）：窗口扩大



TCP 窗口字段长度= 16 位，最大窗口大小 = 64 K 字节。

对于传播时延和带宽都很大的网络，为获得高吞吐率较，需要更大的窗口。

选项（3）：窗口扩大



窗口扩大选项：占 3 字节，其中一个字节表示移位值 S。

新的窗口值位数从 16 增大到 (16 + S)，相当于把窗口值向左移动 S 位。

移位值允许使用的最大值是 14，窗口最大值增大到 $2^{(16+14)} - 1 = 2^{30} - 1$。

窗口扩大选项可以在双方初始建立 TCP 连接时进行协商。

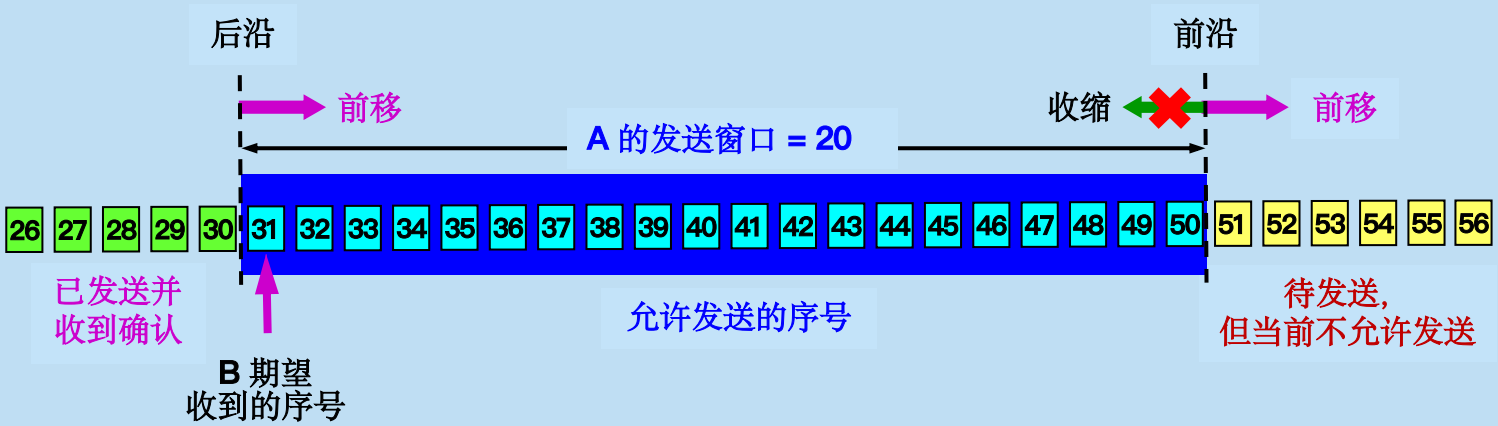
5.6 TCP 可靠传输的实现

5.6.1 以字节为单位的滑动窗口

- TCP 使用流水线传输和滑动窗口协议实现高效、可靠的传输。
- TCP 的滑动窗口是以字节为单位的。
- 发送方 A 和接收方 B 分别维持一个发送窗口和一个接收窗口。
- 发送窗口：在没有收到确认的情况下，发送方可以连续把窗口内的数据全部发送出去。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，以便在超时重传时使用。
- 接收窗口：只允许接收落入窗口内的数据。

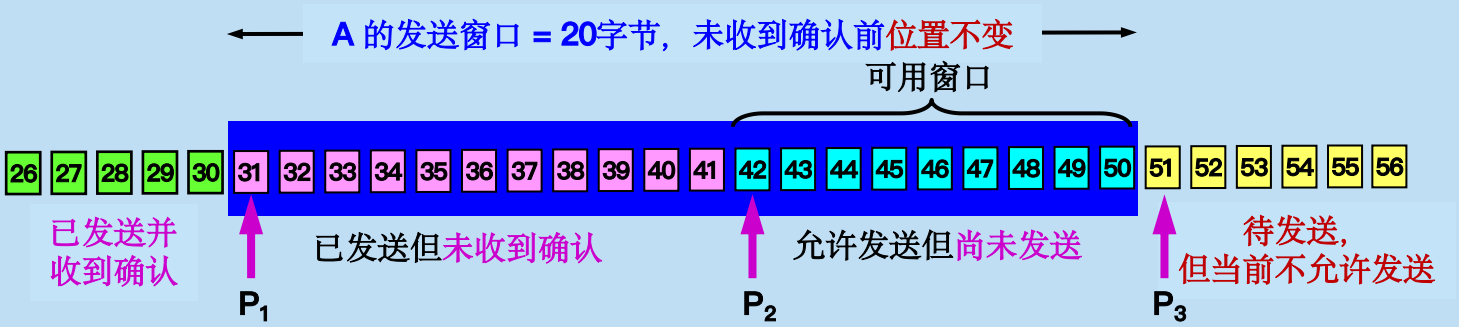
发送窗口

- A 根据 B 给出的窗口值，构造出自己的发送窗口。
- 发送窗口里面的序号表示允许发送的序号。
- 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。



发送窗口

假定 A 发送了序号为 31 ~ 41 共 11 个字节的数据



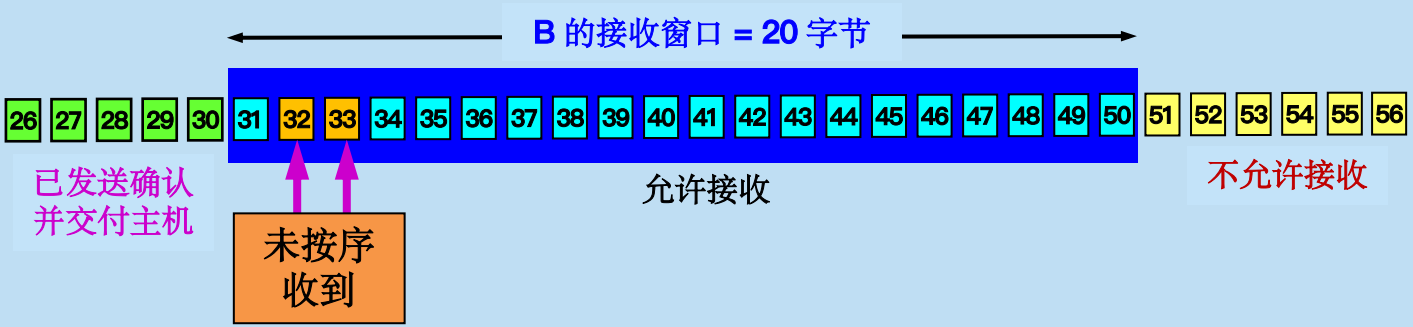
P_1 = 后沿，P2 = 当前，P_3 = 前沿。

$P_3 - P_1$ = 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）

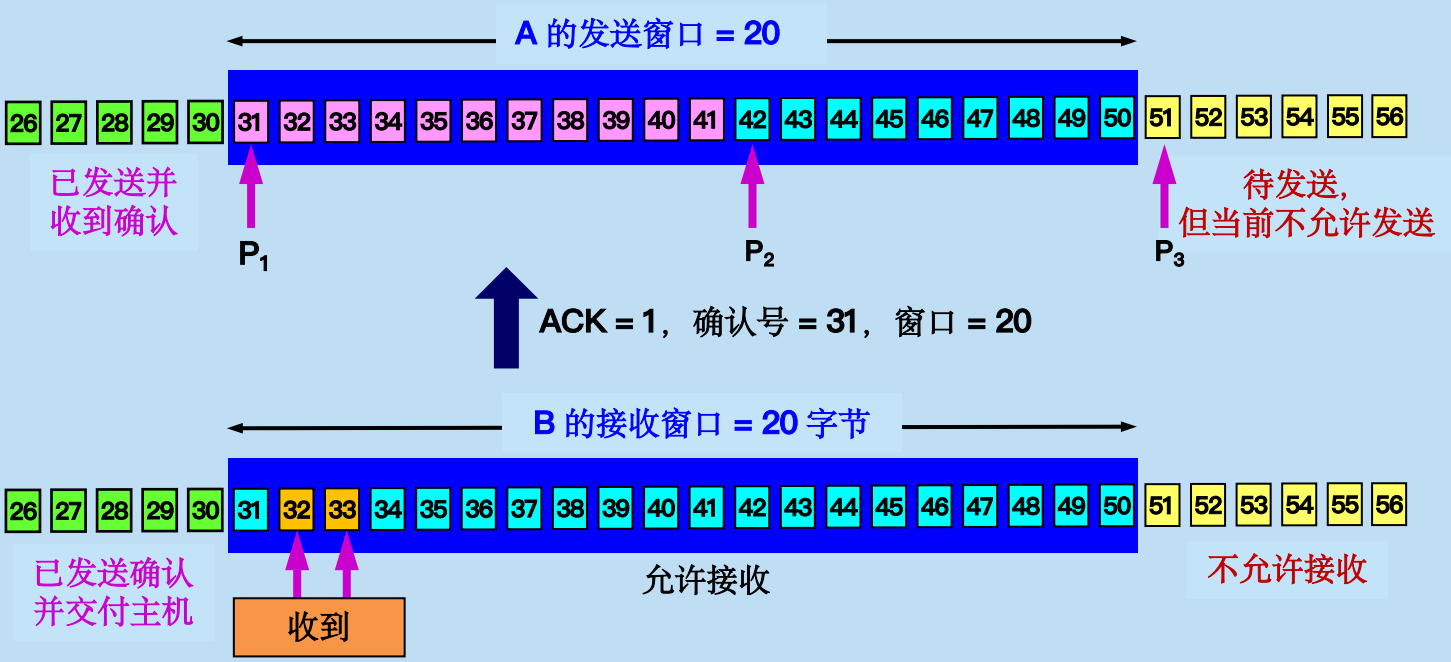
接收窗口



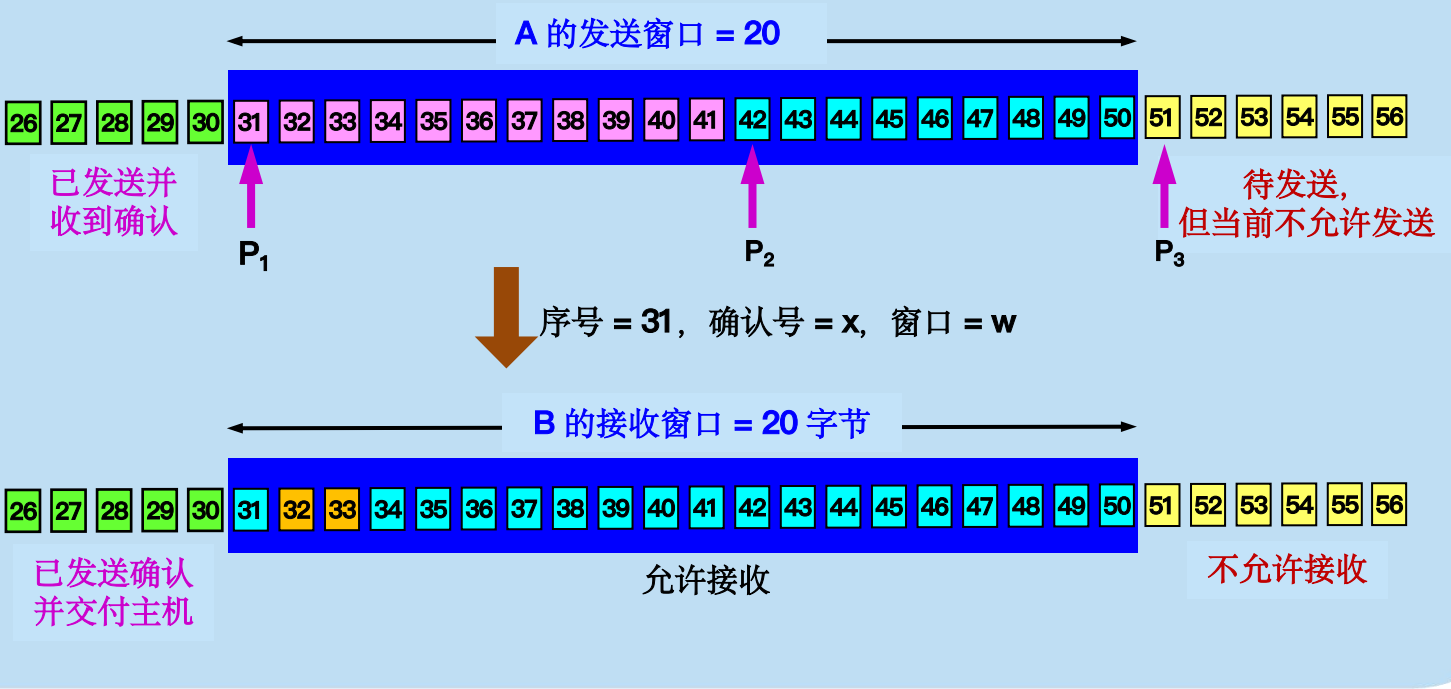
B 收到了序号为 32 和 33 的数据，但未收到序号为 31 的数据。

因此，因此发送的确认报文段中的确认号是 31 （即期望收到的序号）。

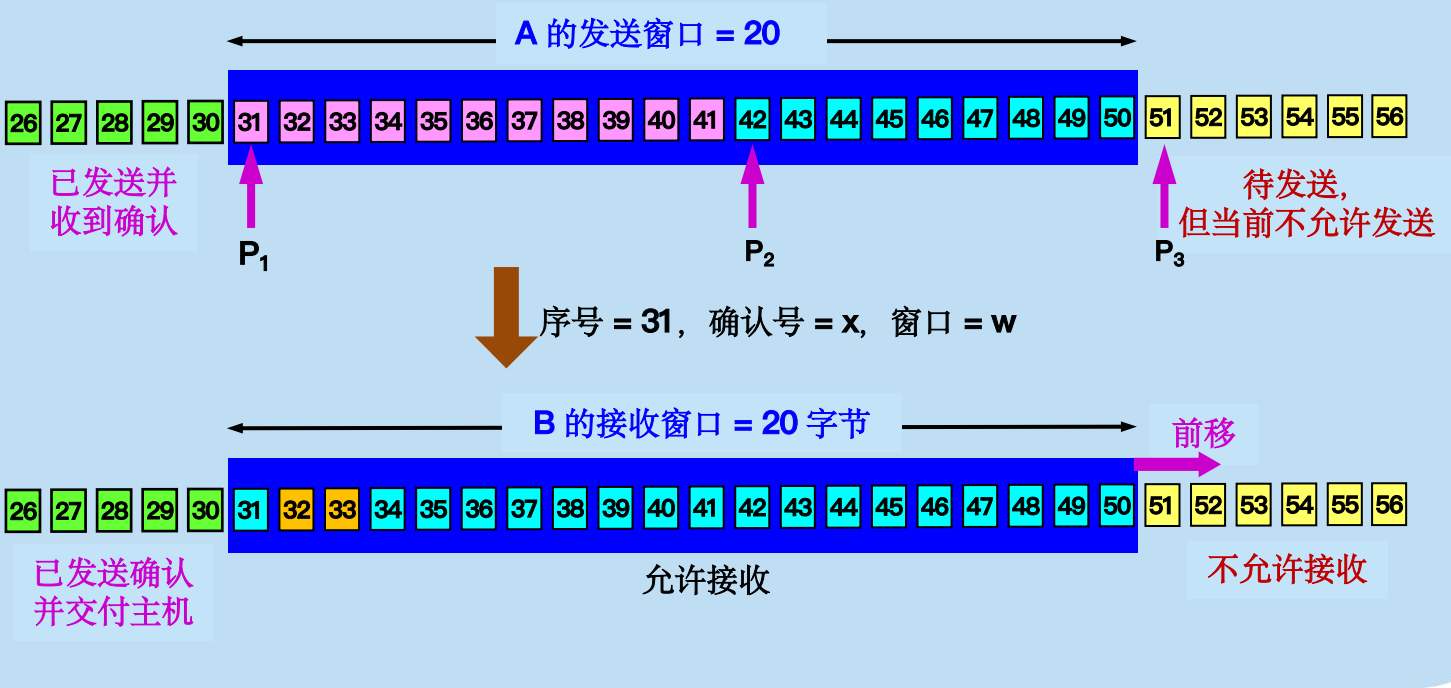
窗口的滑动


A 的发送窗口 = 20
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送并
收到确认
P1
P2
P3
待发送，
但当前不允许发送
ACK = 1，确认号 = 31，窗口 = 20
B 的接收窗口 = 20 字节
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送确认
并交付主机
收到
允许接收
不允许接收

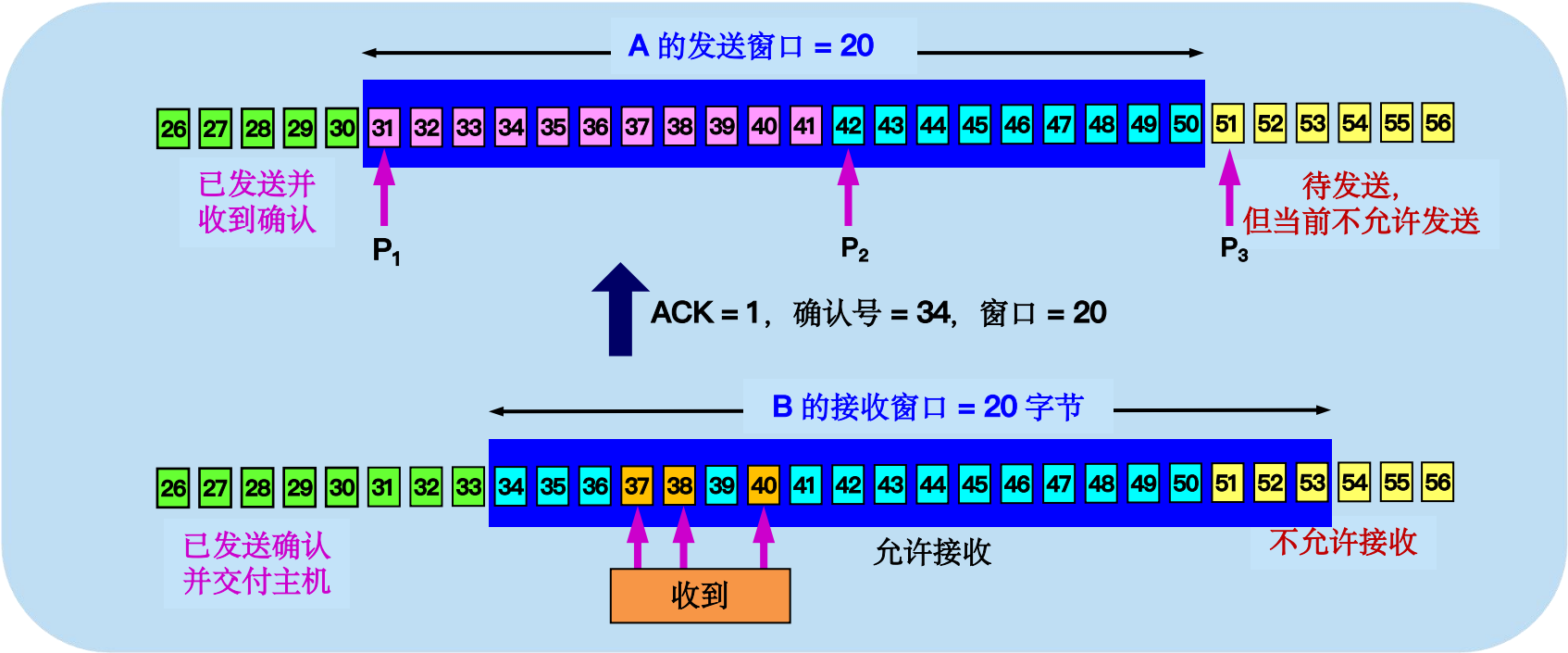
窗口的滑动


A 的发送窗口 = 20
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送并
收到确认
P1
P2
P3
待发送，
但当前不允许发送
序号 = 31，确认号 = x，窗口 = w
B 的接收窗口 = 20 字节
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送确认
并交付主机
允许接收
不允许接收

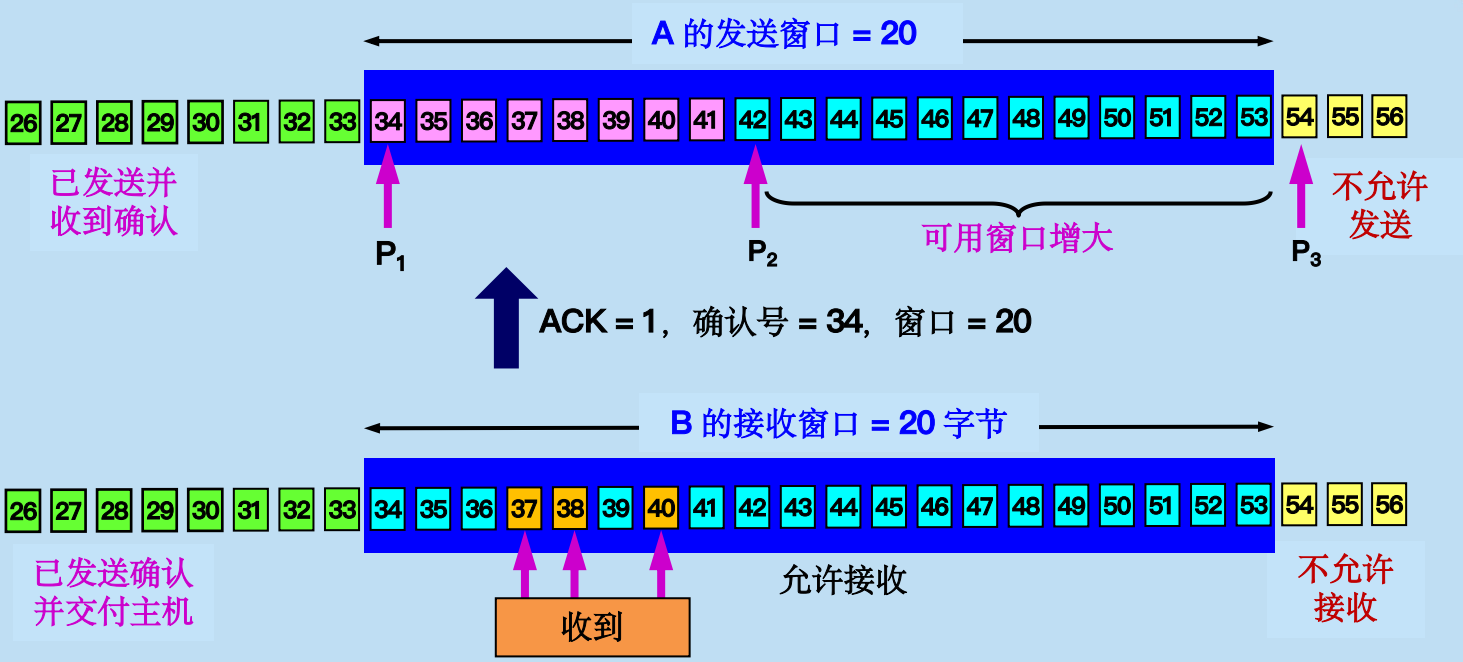
窗口的滑动


A 的发送窗口 = 20
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送并
收到确认
P1
P2
P3
待发送，
但当前不允许发送
序号 = 31，确认号 = x，窗口 = w
B 的接收窗口 = 20 字节
前移
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送确认
并交付主机
允许接收
不允许接收

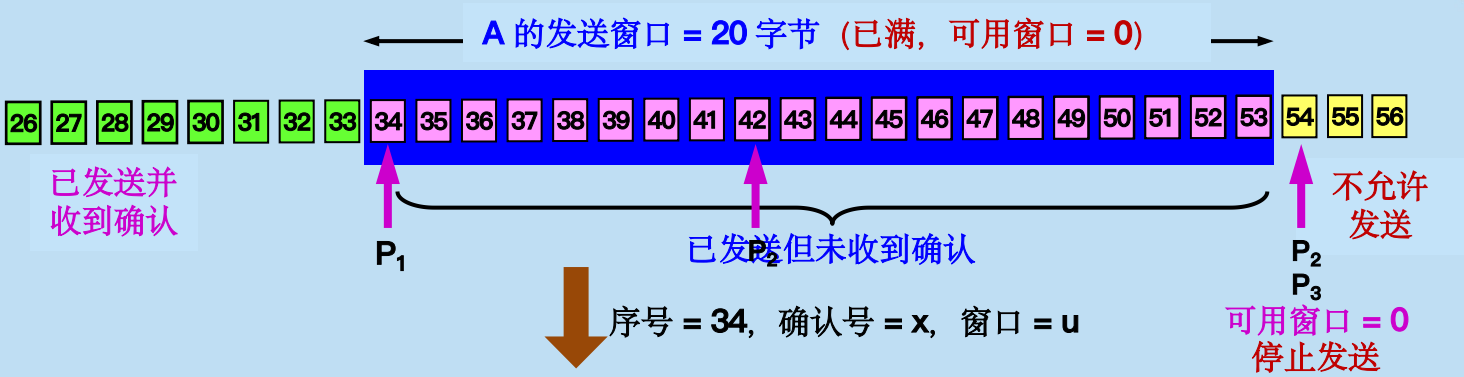
窗口的滑动


A 的发送窗口 = 20
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送并
收到确认
待发送，
但当前不允许发送
P1
P2
P3
ACK = 1，确认号 = 34，窗口 = 20
B 的接收窗口 = 20 字节
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送确认
并交付主机
允许接收
不允许接收
收到

窗口的滑动


A 的发送窗口 = 20
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送并
收到确认
P1
P2
可用窗口增大
P3
不允许
发送
ACK = 1，确认号 = 34，窗口 = 20
B 的接收窗口 = 20 字节
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
已发送确认
并交付主机
收到
允许接收
不允许
接收

窗口的滑动

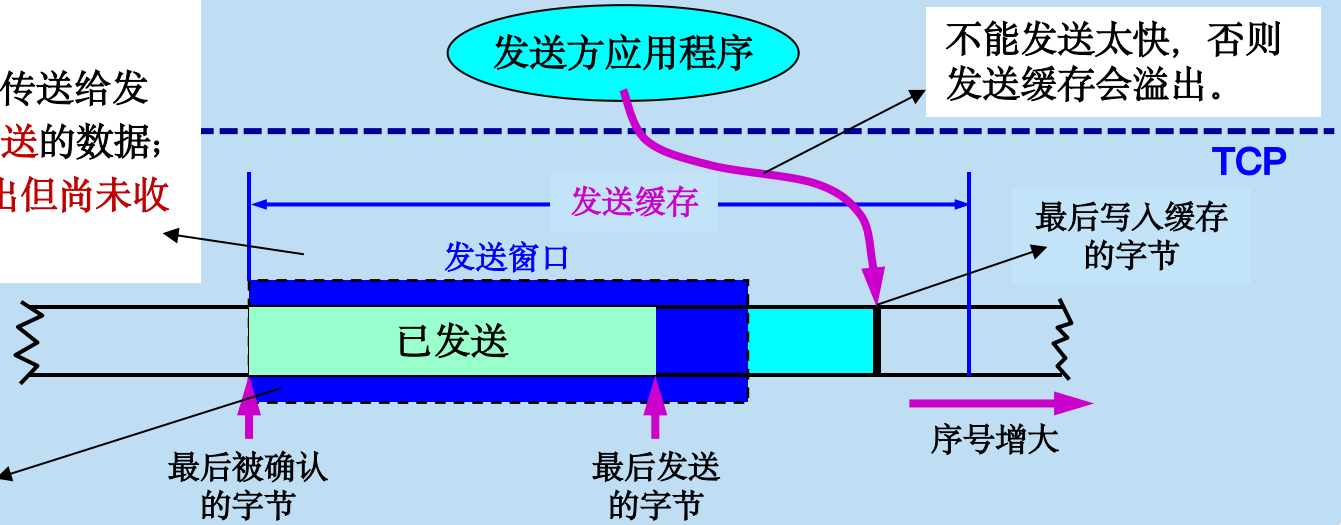


- A 未收到确认的原因有：① B 未发送；② B已发送，但还未到达 A 。
- 为保证可靠传输，A 只能认为 B 还没有收到这些数据。A 经过一段时间后（由超时计时器控制）就重传这部分数据，重新设置超时计时器，直到收到 B 的确认为止。
- 如果 A 按序收到落在发送窗口内的确认号， 就使发送窗口向前滑动，并发送新的数据。

发送缓存与发送窗口

发送方的应用进程把字节流写入 TCP 发送缓存。

暂时存放：
(1) 发送应用程序传送给发送方 TCP 准备发送的数据；
(2) TCP 已发送出但尚未收到确认的数据。



缓存中的字节数 = 发送应用程序最后写入缓存的字节 –最后被确认的字节

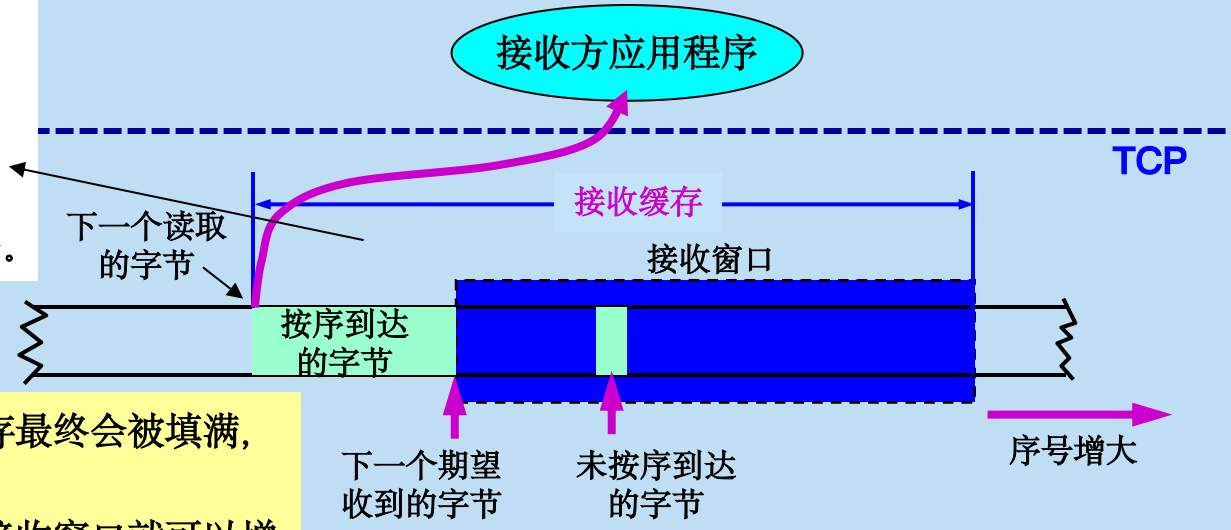
接收缓存与接收窗口

接收方的应用进程从 TCP 接收缓存中读取尚未被读取的字节。

暂时存放：

(1) 按序到达的、但尚未被接收应用程序读取的数据；

(2) 未按序到达的数据。



若不能及时读取，缓存最终会被填满，使接收窗口减小到零。

如果能够及时读取，接收窗口就可以增大，但最大不能超过接收缓存的大小。

TCP 超时重传时间设置

- 不能太短，否则会引起很多报文段的不必要的重传，使网络负荷增大。
- 不能过长，会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

- 加权平均往返时间 RTT_S 又称为平滑的往返时间。

新的 RTT_S = (1 − α) × (旧的 RTT_S) + α × (新的 RTT 样本)　　(5-4)

其中，$0 \leq \alpha < 1$ 。

若 $\alpha \to 0$，表示 RTT 值更新较慢。

若 $\alpha \to 1$，表示 RTT 值更新较快。

RFC 6298 推荐的 α 值为 1/8，即 0.125。

5.7 TCP 的流量控制

5.7.1 利用滑动窗口实现流量控制

- 流量控制 (flow control)：让发送方的发送速率不要太快，使接收方来得及接收。
- 利用滑动窗口机制可以很方便地在 TCP 连接上实现对发送方的流量控制。

利用可变窗口进行流量控制举例

A 向 B 发送数据，MSS = 100 字节。
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

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

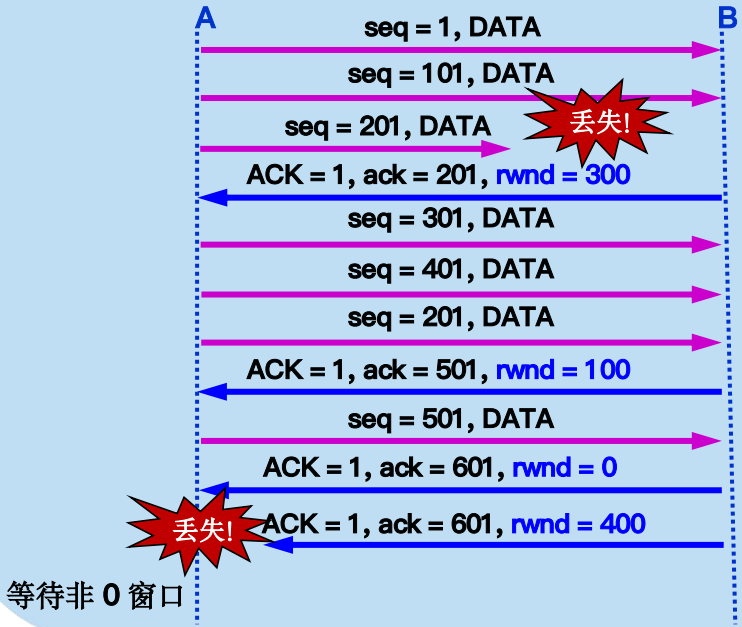
允许 A 发送序号 501 至 600 共 100 字节

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）

可能发生死锁

A 向 B 发送数据，MSS = 100 字节。
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

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

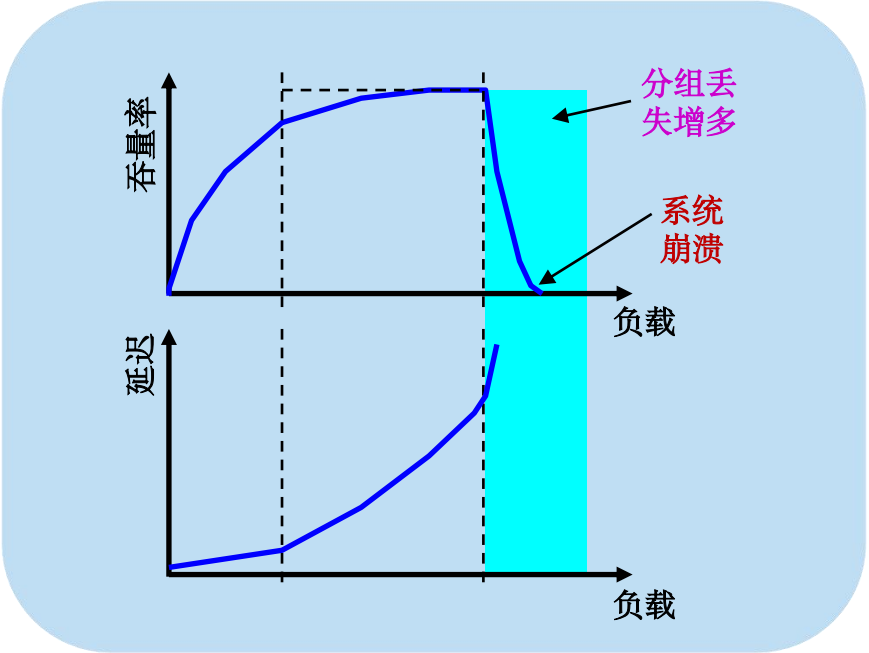
持续计时器

- 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），对方在确认这个探测报文段时给出当前窗口值。
  - 若窗口仍然是零，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。

5.8 TCP 的拥塞控制

5.8.1 拥塞控制的一般原理

- 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，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降。这种现象称为拥塞 (congestion)。
- 最坏结果：系统崩溃。



拥塞产生的原因

- 由许多因素引起。例如：
  1. 节点缓存容量太小；
  2. 链路容量不足；
  3. 处理机处理速率太慢；
  4. 拥塞本身会进一步加剧拥塞；
- 出现网络拥塞的条件：

$$\sum \text{对资源需求} > \text{可用资源} \qquad (5\text{-}7)$$

拥塞控制与流量控制的区别

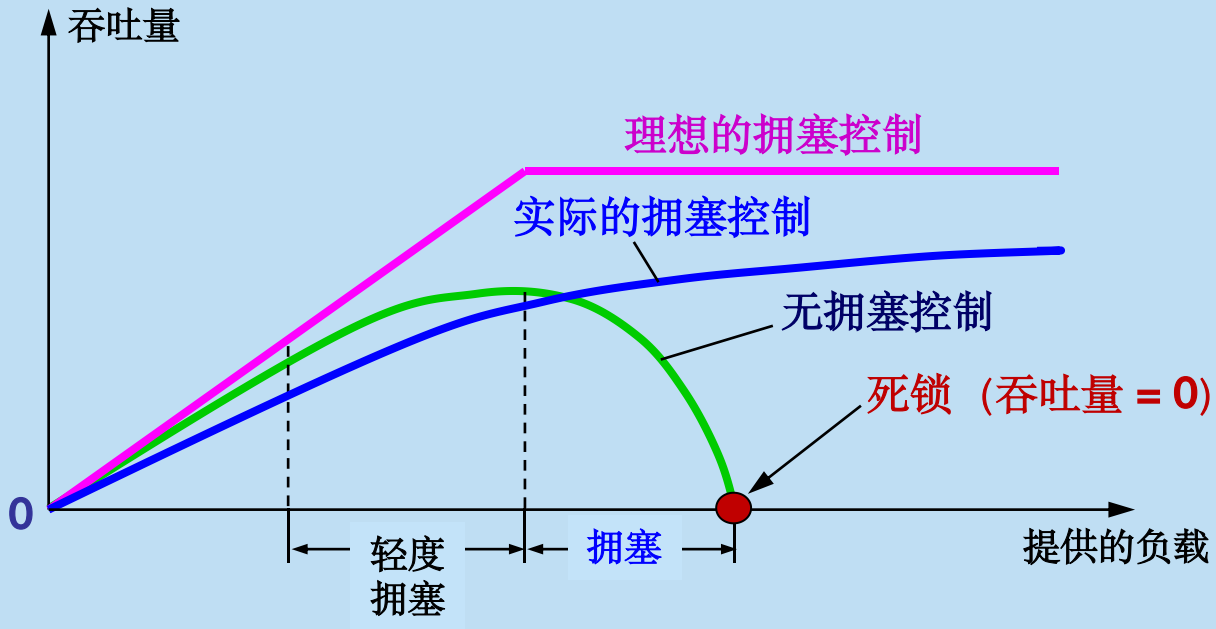
拥塞控制

- 防止过多的数据注入到网络中，避免网络中的路由器或链路过载。
- 是一个全局性的过程，涉及到所有的主机、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

- 抑制发送端发送数据的速率，以使接收端来得及接收。
- 点对点通信量的控制，是个端到端的问题。

拥塞控制所起的作用



拥塞控制的一般原理

- 拥塞控制的前提：网络能够承受现有的网络负荷。
- 实践证明，拥塞控制是很难设计的，因为它是一个动态问题。
- 分组的丢失是网络发生拥塞的征兆，而不是原因。
- 在许多情况下，甚至正是拥塞控制本身成为引起网络性能恶化、甚至发生死锁的原因。

开环控制和闭环控制

开环控制

- 在设计网络时，事先考虑周全，力求工作时不发生拥塞。
- 思路：力争避免发生拥塞。
- 但一旦整个系统运行起来，就不再中途进行改正了。

闭环控制

- 基于反馈环路的概念。
- 根据网络当前运行状态采取相应控制措施。
- 思路：在发生拥塞后，采取措施进行控制，消除拥塞。

闭环控制措施

1，监测

监测网络系统，检测拥塞在何时、何处发生。

2，传送

将拥塞发生的信息传送到可采取行动的地方。

3，调整

调整网络系统的运行以解决出现的问题。

5.8.2 TCP 的拥塞控制方法

- TCP 采用基于滑动窗口的方法进行拥塞控制，属于闭环控制方法。
- TCP 发送方维持一个拥塞窗口 cwnd (Congestion Window)
- 拥塞窗口的大小取决于网络的拥塞程度，并且是动态变化的。
- 发送端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窗口，还取决于网络的拥塞状况。
- 真正的发送窗口值：

真正的发送窗口值 = Min (接收方通知的窗口值，拥塞窗口值)

控制拥塞窗口变化的原则

- 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，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，缓解网络出现的拥塞。

发送方判断拥塞的方法：隐式反馈

超时重传计时器超时	• 网络已经出现了拥塞。
收到 3 个重复的确认	• 预示网络可能会出现拥塞。

因传输出差错而丢弃分组的概率很小（远小于1 %）。

因此，发送方在超时重传计时器启动时，就判断网络出现了拥塞。

TCP 拥塞控制算法

- 四种拥塞控制算法（RFC 5681）：
  - 慢开始 (slow-start)
  - 拥塞避免 (congestion avoidance)
  - 快重传 (fast retransmit)
  - 快恢复 (fast recovery)

1. 慢开始 (Slow start)

- 目的：探测网络的负载能力或拥塞程度。
- 算法：由小到大逐渐增大注入到网络中的数据字节，即：由小到大逐渐增大拥塞窗口数值。
- 2 个控制变量：

拥塞窗口 cwnd

- 初始值：2 种设置方法。
  - 1 至 2 个最大报文段 MSS （旧标准）
  - 2 至 4 个最大报文段 MSS （RFC 5681）

慢开始门限 ssthresh

- 防止拥塞窗口增长过大引起网络拥塞。

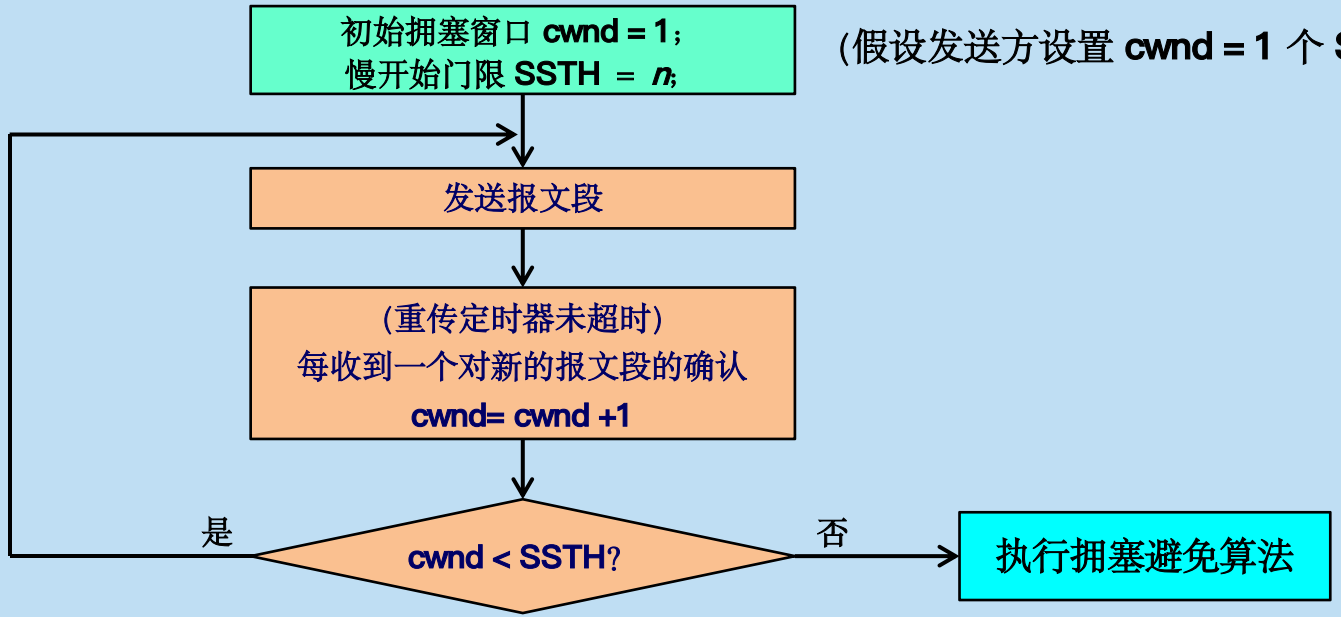
1. 慢开始 (Slow start)

- 拥塞窗口 cwnd 增大：在每收到一个对新的报文段的确认，就把拥塞窗口增加最多一个发送方的最大报文段 SMSS（Sender Maximum Segment Size）的数值。

拥塞窗口 cwnd 每次的增加量 = min (*N*, SMSS)	(5-8)

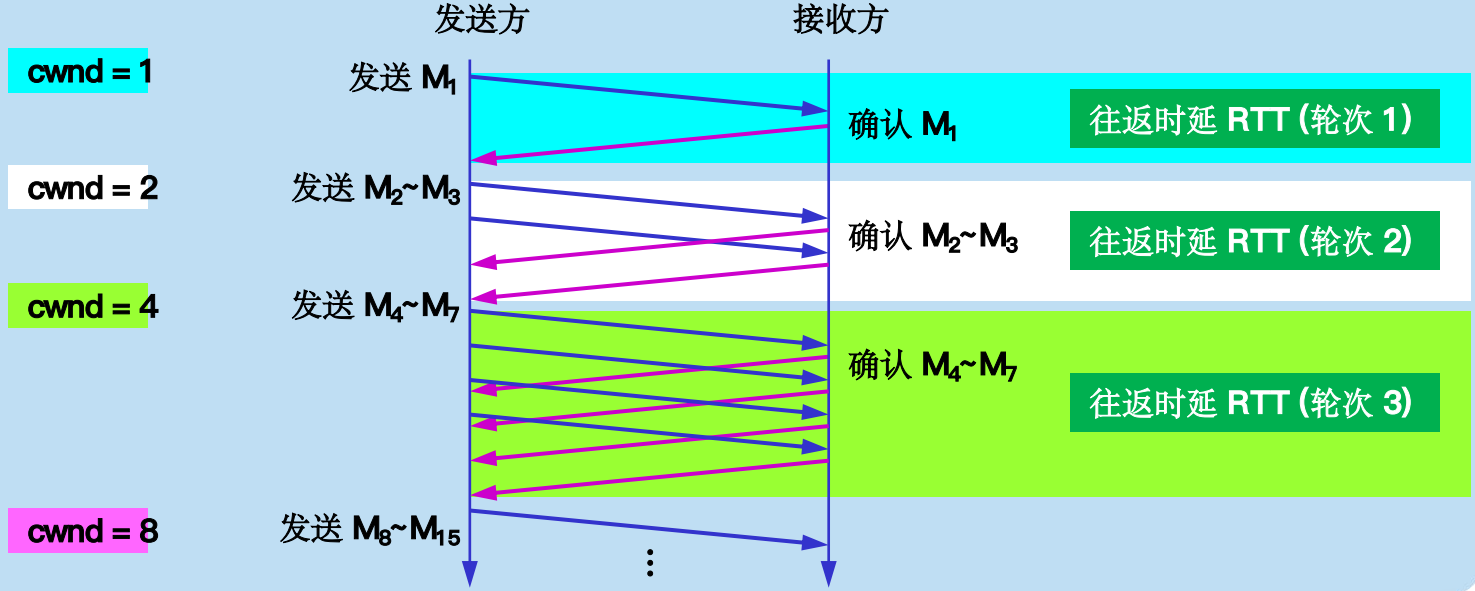
其中 *N* 是原先未被确认的、但现在被刚收到的确认报文段所确认的字节数。

1. 慢开始 (Slow start)

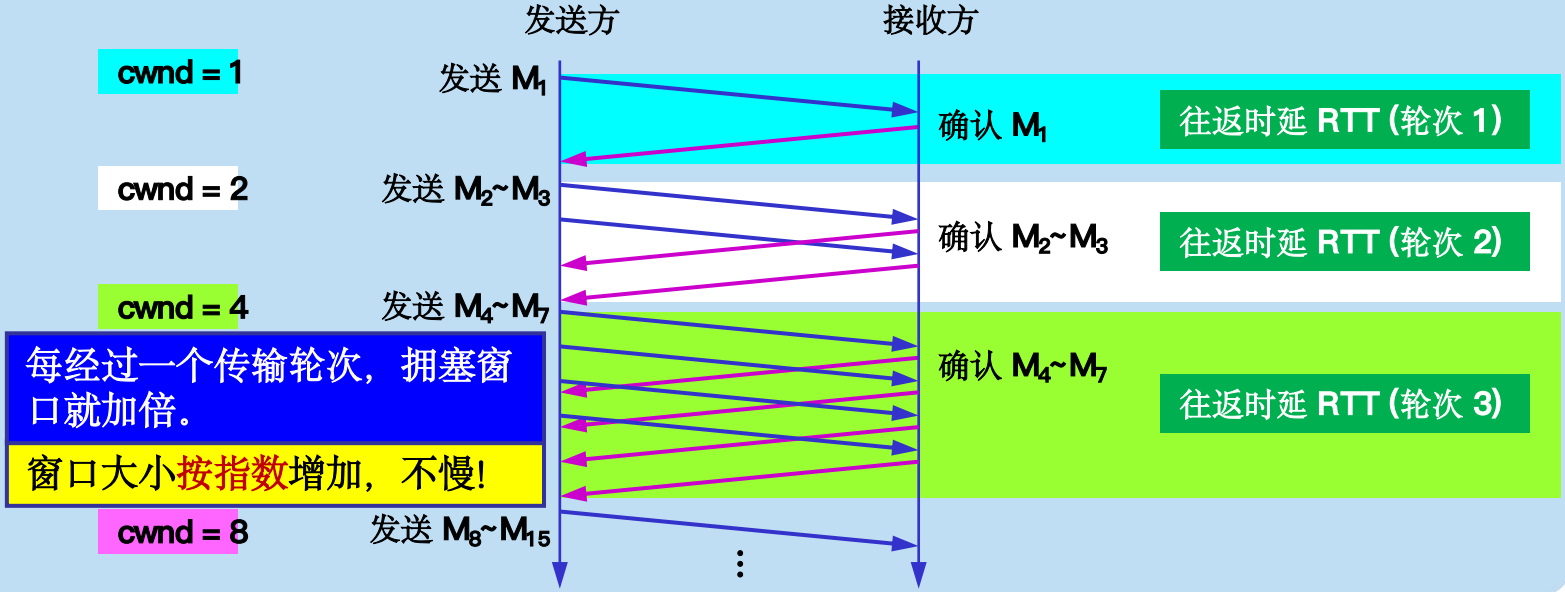


(假设发送方设置 cwnd = 1 个 SMSS)

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 1。



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 1。



传输轮次（transmission round）

- 一个传输轮次所经历的时间其实就是往返时间 RTT。
- 传输轮次强调：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。

慢开始门限 ssthresh

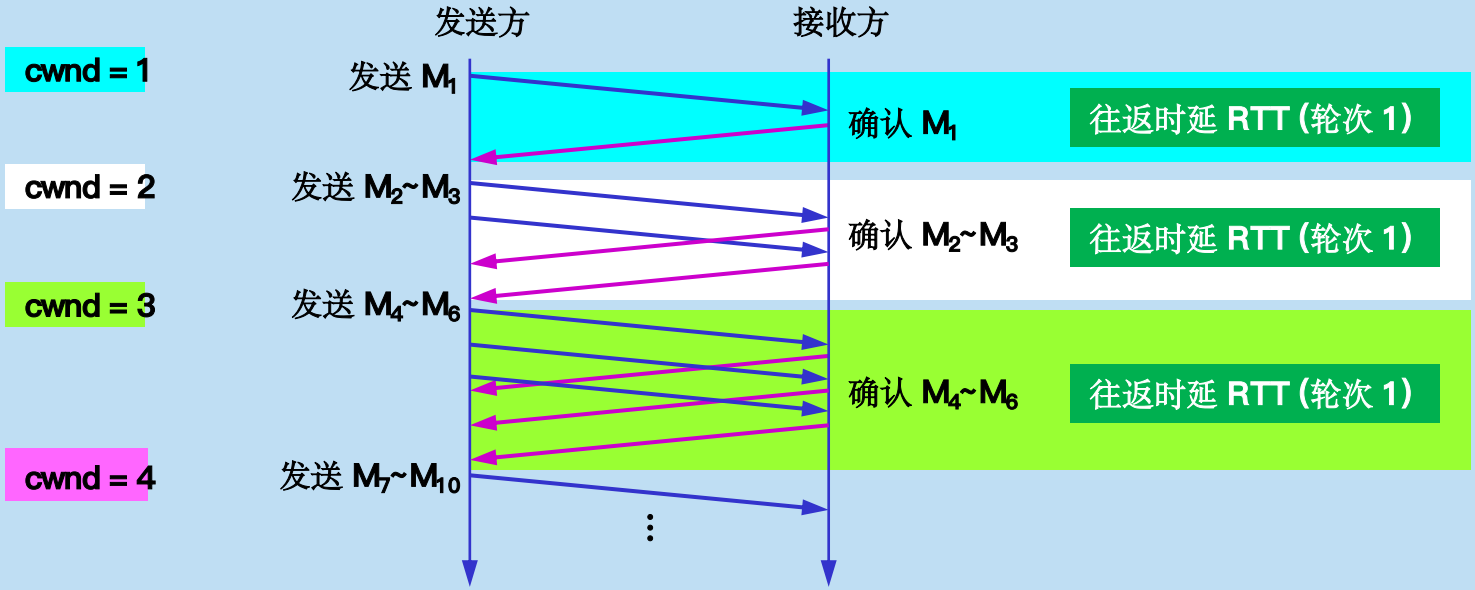
- 防止拥塞窗口 **cwnd** 增长过大引起网络拥塞。
- 用法：
  1. 当 **cwnd < ssthresh** 时，使用慢开始算法。
  2. 当 **cwnd > ssthresh** 时，停止使用慢开始算法，改用拥塞避免算法。
  3. 当 **cwnd = ssthresh** 时，既可使用慢开始算法，也可使用拥塞避免算法。

2. 拥塞避免

- 目的：让拥塞窗口 cwnd 缓慢地增大，避免出现拥塞。
- 拥塞窗口 cwnd 增大：每经过一个往返时间 RTT（不管在此期间收到了多少确认），发送方的拥塞窗口 cwnd = cwnd + 1。
- 具有加法增大 AI (Additive Increase) 特点：使拥塞窗口 cwnd 按线性规律缓慢增长。

注意：
拥塞避免并非完全避免拥塞，而是让拥塞窗口增长得缓慢些，使网络不容易出现拥塞。

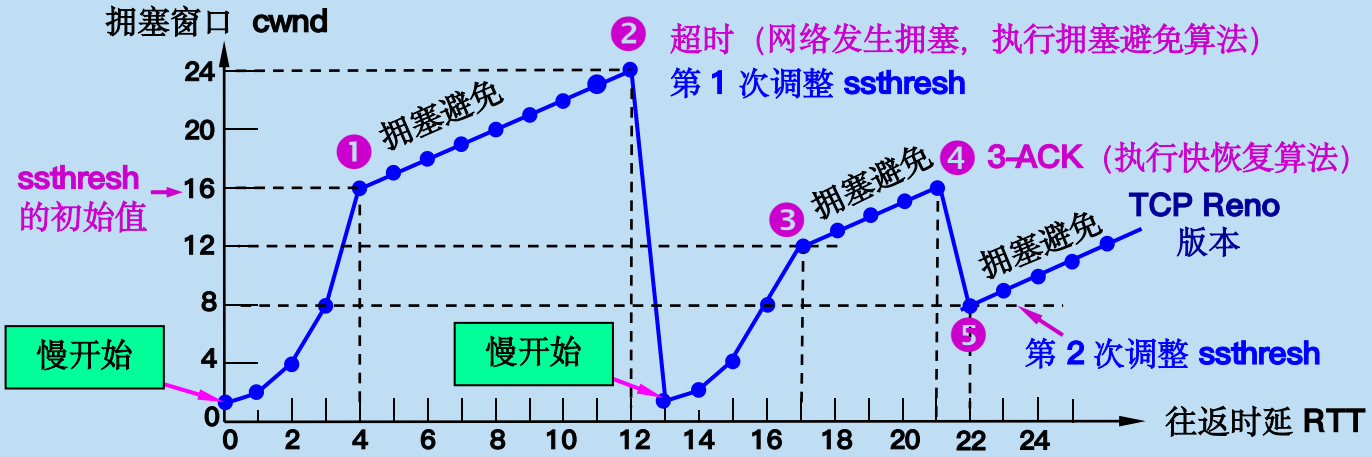
每经过一个往返时间 RTT，发送方就把拥塞窗口 cwnd 加 1。



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）：
  1. ssthresh = max (cwnd/2，2)
  2. cwnd = 1
  3. 执行慢开始算法
- 目的：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。

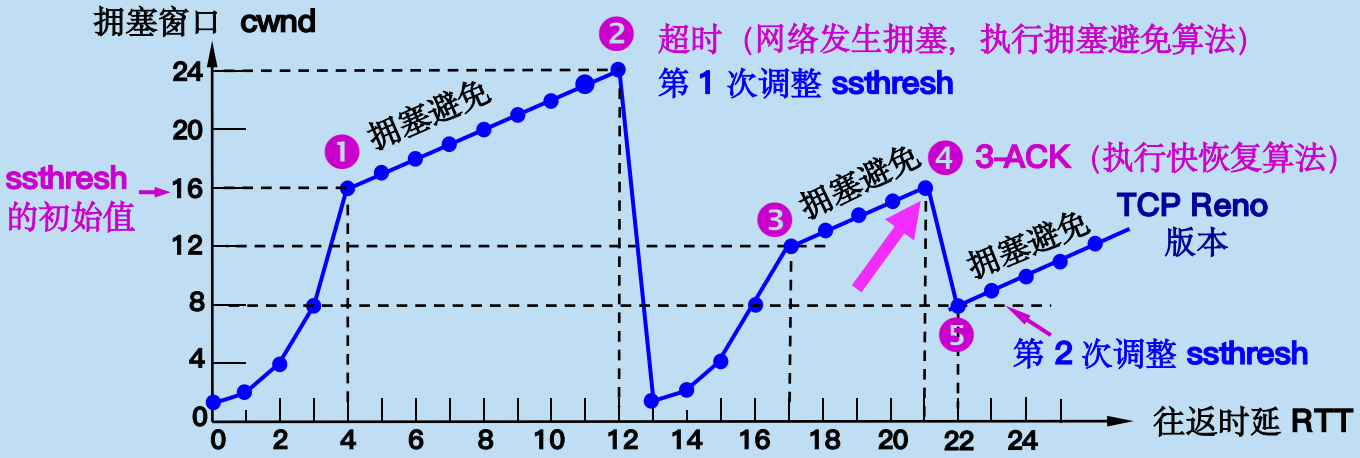
慢开始和拥塞避免算法的实现举例



当 TCP 连接进行初始化时，将拥塞窗口置为 1（窗口单位不使用字节而使用报文段）。

将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



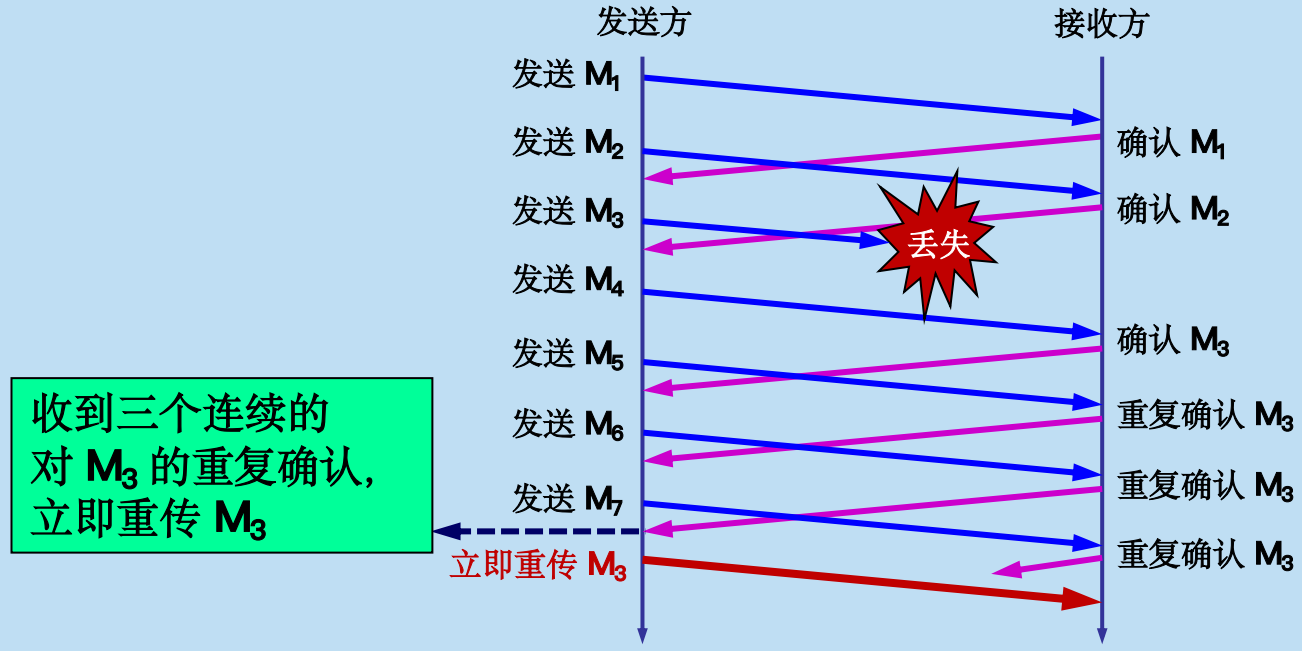
当拥塞窗口 cwnd = 16 时，发送方连续收到 3 个对同一个报文段的重复确认（记为 3-ACK）。发送方改为执行快重传和快恢复算法。

快重传 FR (Fast Retransmission) 算法

- 目的：让发送方尽早知道发生了个别报文段的丢失。
- 发送方只要连续收到三个重复的确认，就立即进行重传（即“快重传”），这样就不会出现超时。
- 使用快重传可以使整个网络的吞吐量提高约 20%。
- 快重传算法要求接收方立即发送确认，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。

注意：
快重传并非取消重传计时器，而是在某些情况下可以更早地（更快地）重传丢失的报文段。

快重传举例

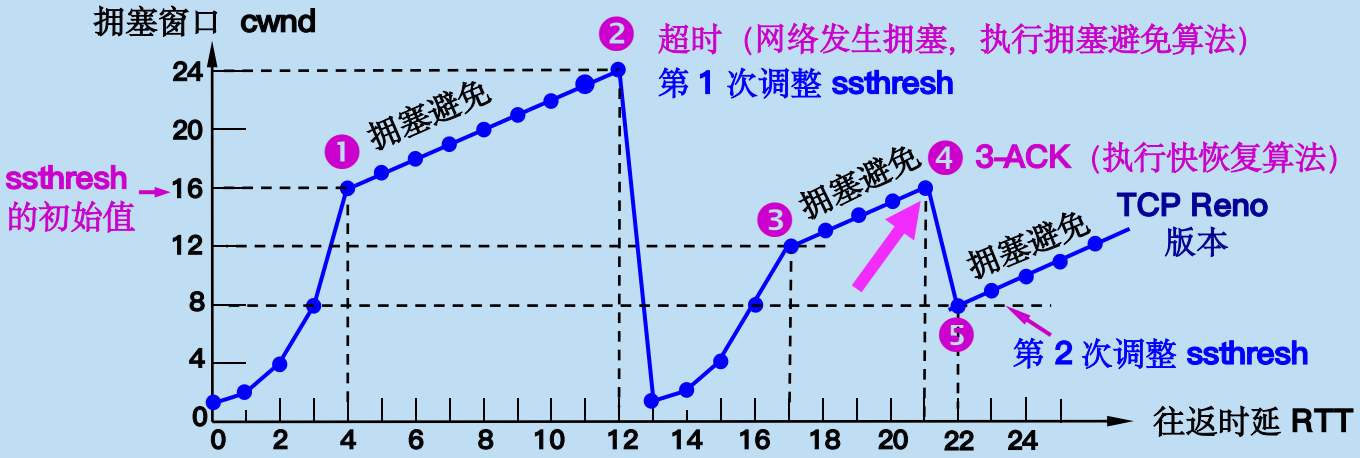

发送方
接收方
发送 M1
发送 M2
发送 M3
发送 M4
发送 M5
发送 M6
发送 M7
确认 M1
确认 M2
丢失
确认 M3
重复确认 M3
重复确认 M3
重复确认 M3
收到三个连续的
对 M3 的重复确认，
立即重传 M3
立即重传 M3

快恢复 FR (Fast Recovery)算法

- 当发送端收到连续三个重复的确认时，不执行慢开始算法，而是执行快恢复算法 FR (Fast Recovery) 算法：
  1. 慢开始门限 ssthresh = 当前拥塞窗口 cwnd / 2；
  2. 乘法减小 MD (Multiplicative Decrease) 拥塞窗口。
     新拥塞窗口 cwnd = 慢开始门限 ssthresh；
  3. 执行拥塞避免算法，使拥塞窗口缓慢地线性增大（加法增大 AI）。

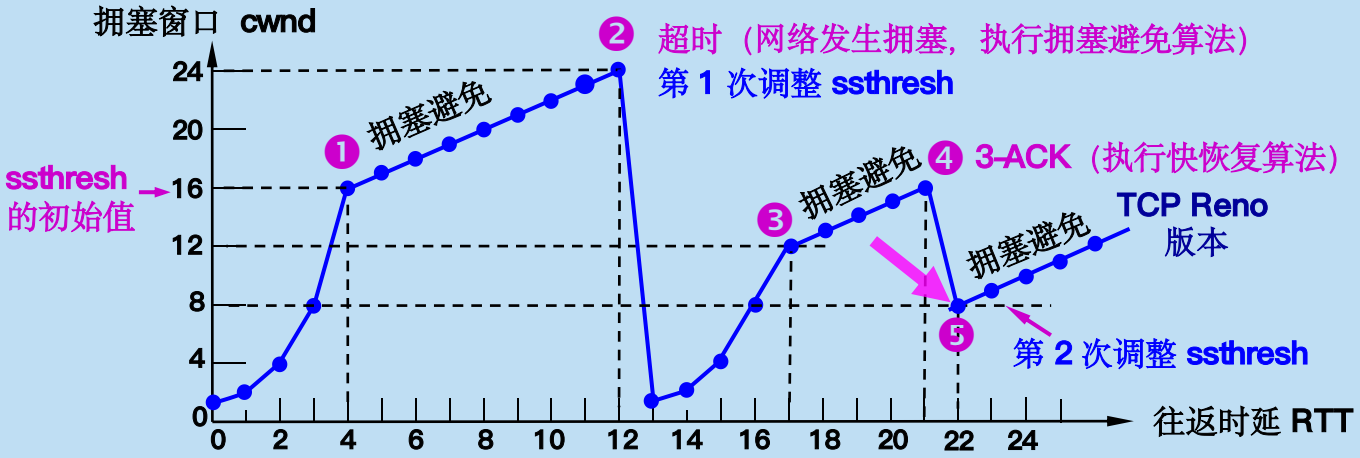
二者合在一起就是所谓的 AIMD 算法，使 TCP 性能有明显改进。

慢开始和拥塞避免算法的实现举例



当拥塞窗口 cwnd = 16 时，发送方连续收到 3 个对同一个报文段的重复确认（记为 3-ACK）。发送方改为执行快重传和快恢复算法。

慢开始和拥塞避免算法的实现举例



执行快重传和快恢复算法：发送方调整门限值 ssthresh = cwnd / 2 = 8，设置拥塞窗口 cwnd = ssthresh = 8，开始执行拥塞避免算法。