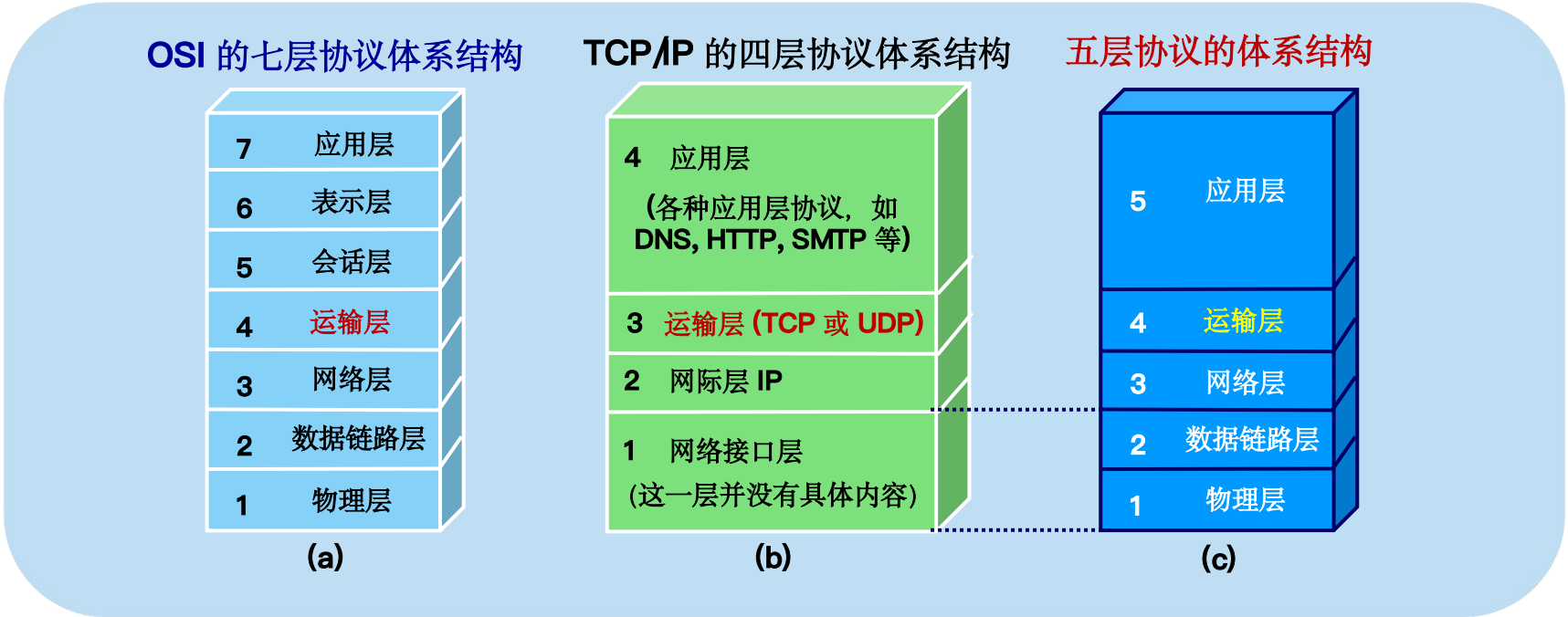
第5章
运输层

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)

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：

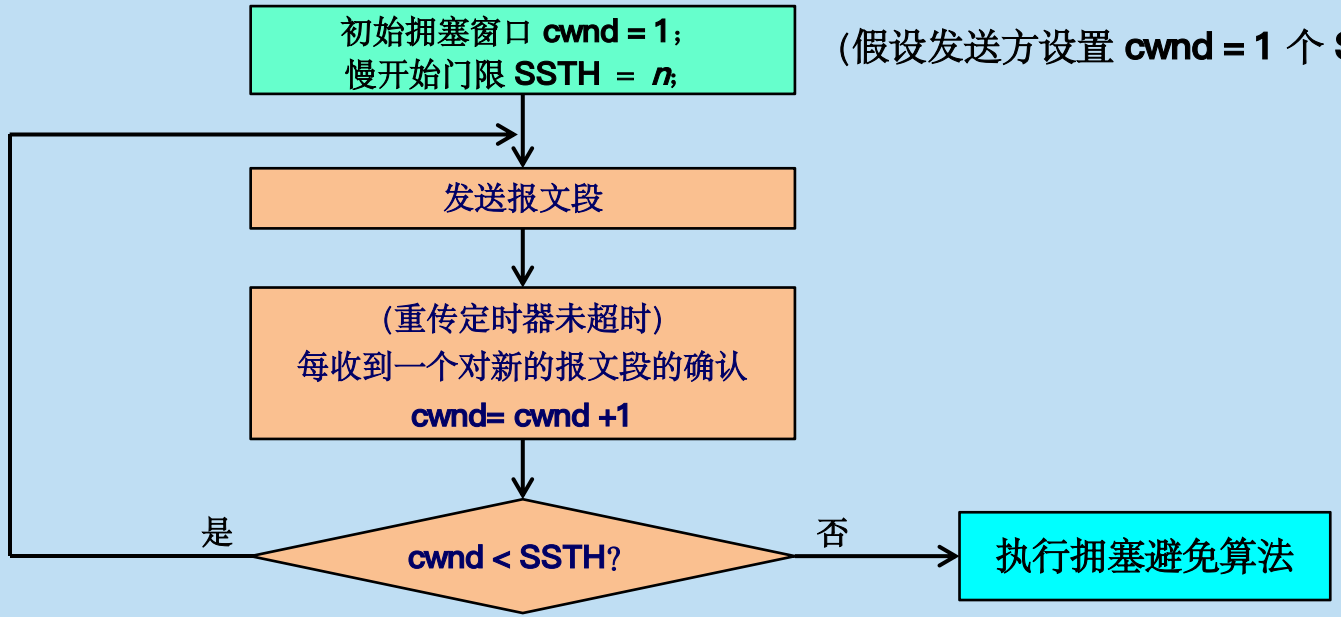
拥塞窗口 cwnd

- 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

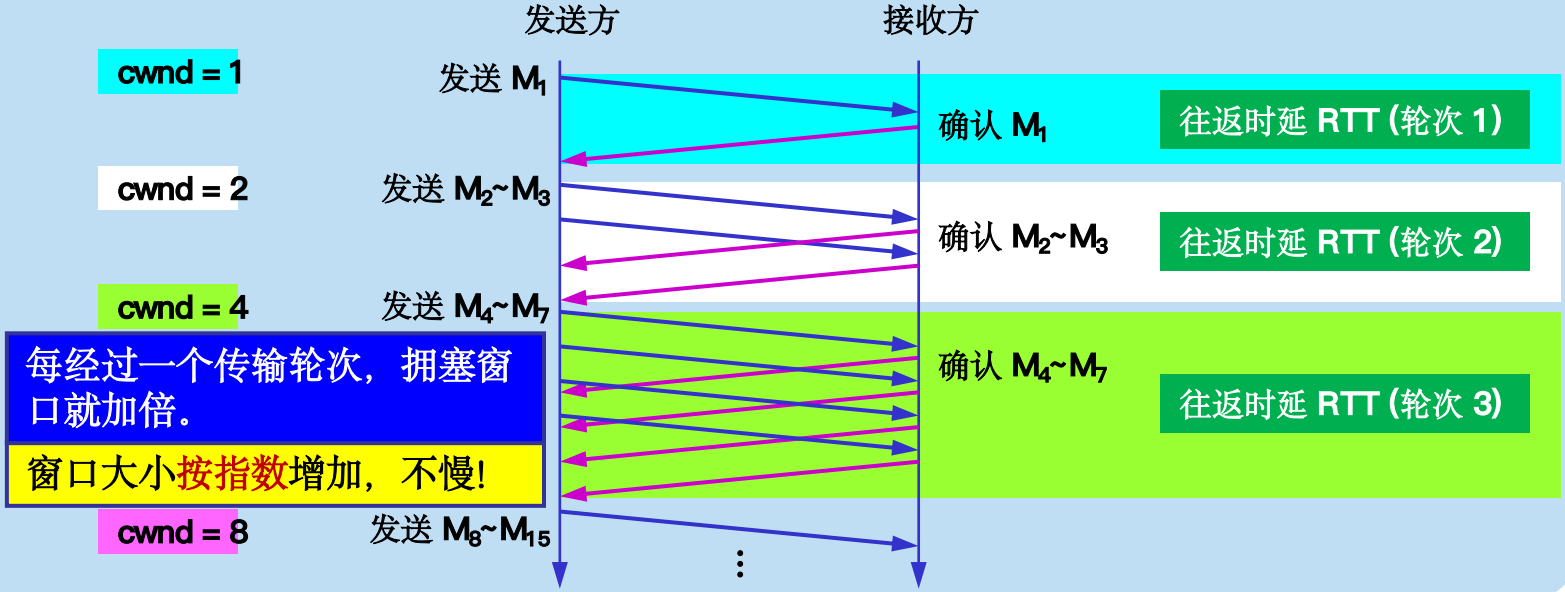
- 防止拥塞窗口增长过大引起网络拥塞。

1. 慢开始 (Slow start)



(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。



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

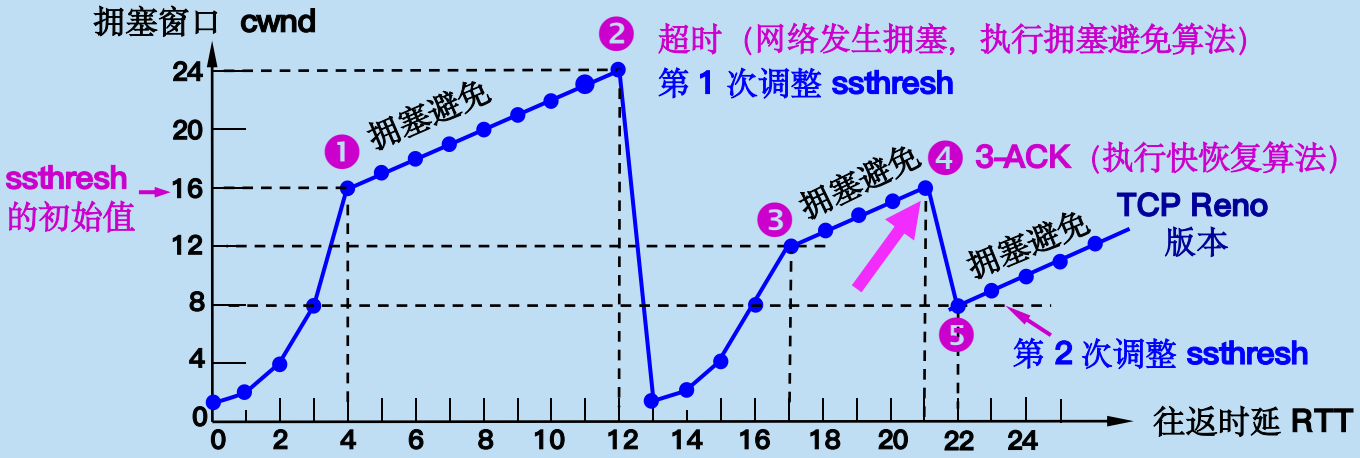
- 目的：让拥塞窗口 cwnd 缓慢地增大，避免出现拥塞。
- 拥塞窗口 cwnd 增大：每经过一个往返时间 RTT（不管在此期间收到了多少确认），发送方的拥塞窗口 cwnd = cwnd + 1。
- 具有加法增大 AI (Additive Increase) 特点：使拥塞窗口 cwnd 按线性规律缓慢增长。

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。

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



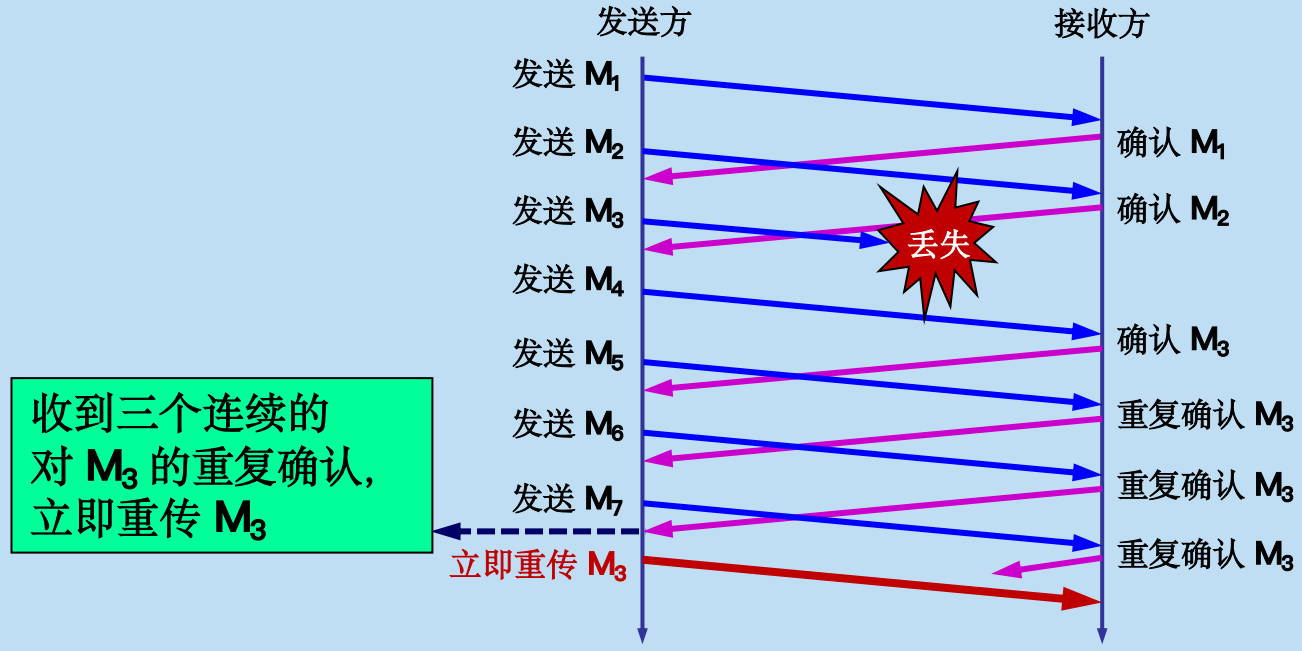
当拥塞窗口 cwnd = 16 时，发送方连续收到 3 个对同一个报文段的重复确认（记为 3-ACK）。发送方改为执行快重传和快恢复算法。

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

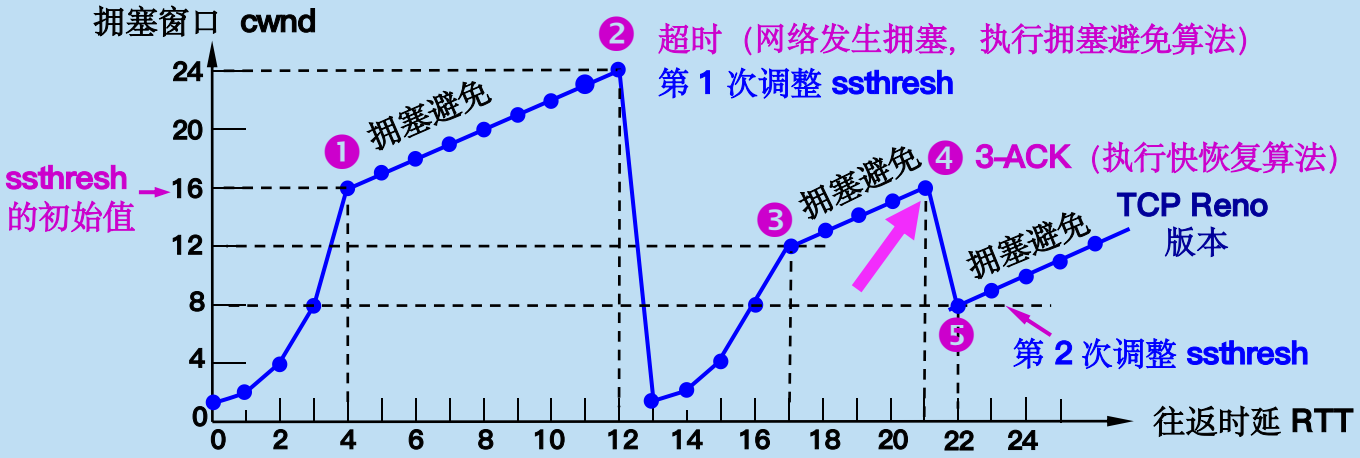

发送方
接收方
发送 M1
发送 M2
发送 M3
发送 M4
发送 M5
发送 M6
发送 M7
确认 M1
确认 M2
丢失
确认 M3
重复确认 M3
重复确认 M3
重复确认 M3
收到三个连续的
对 M3 的重复确认，
立即重传 M3
立即重传 M3

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）。

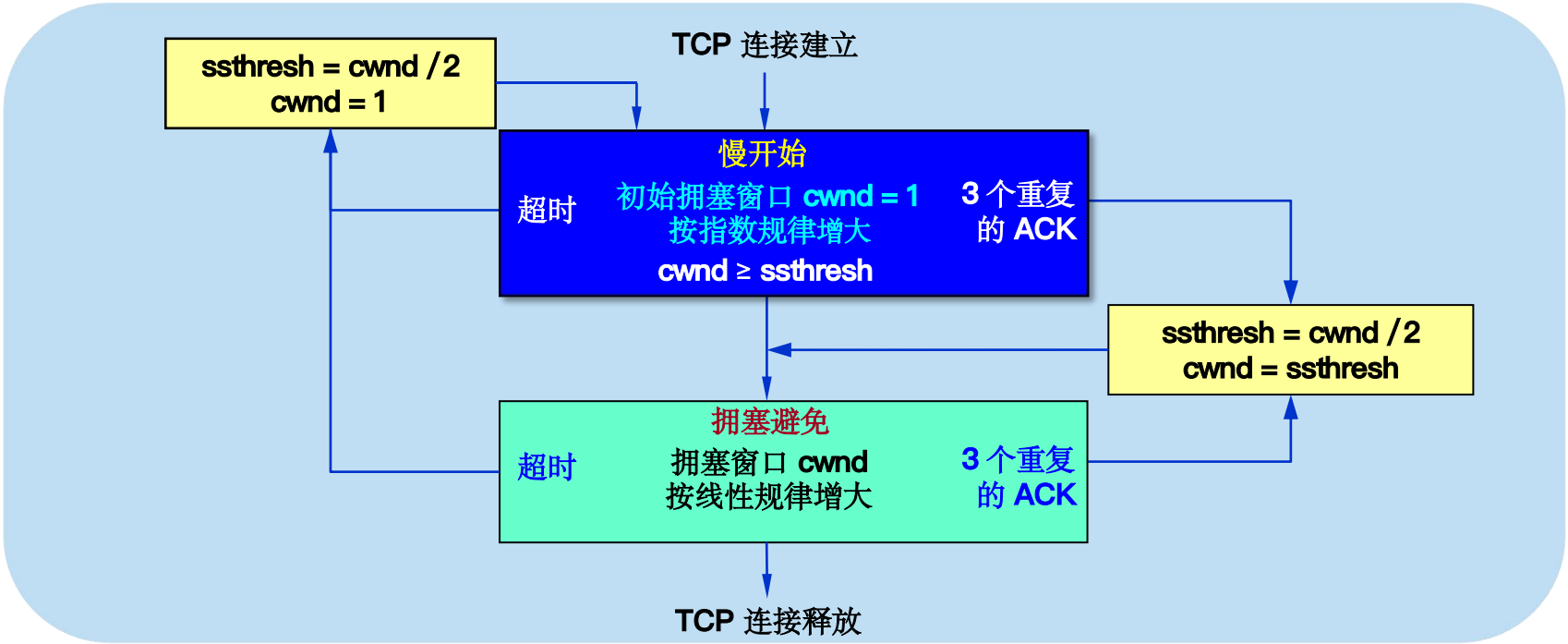
二者合在一起就是所谓的 AIMD 算法，使 TCP 性能有明显改进。

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。

TCP 拥塞控制流程图


TCP 连接建立
ssthresh = cwnd /2
cwnd = 1
慢开始
超时
初始拥塞窗口 cwnd = 1
按指数规律增大
cwnd ≥ ssthresh
3 个重复
的 ACK
ssthresh = cwnd /2
cwnd = ssthresh
拥塞避免
超时
拥塞窗口 cwnd
按线性规律增大
3 个重复
的 ACK
TCP 连接释放

发送窗口的上限值

发送窗口的上限值 = Min [rwnd, cwnd]　　(5-9)

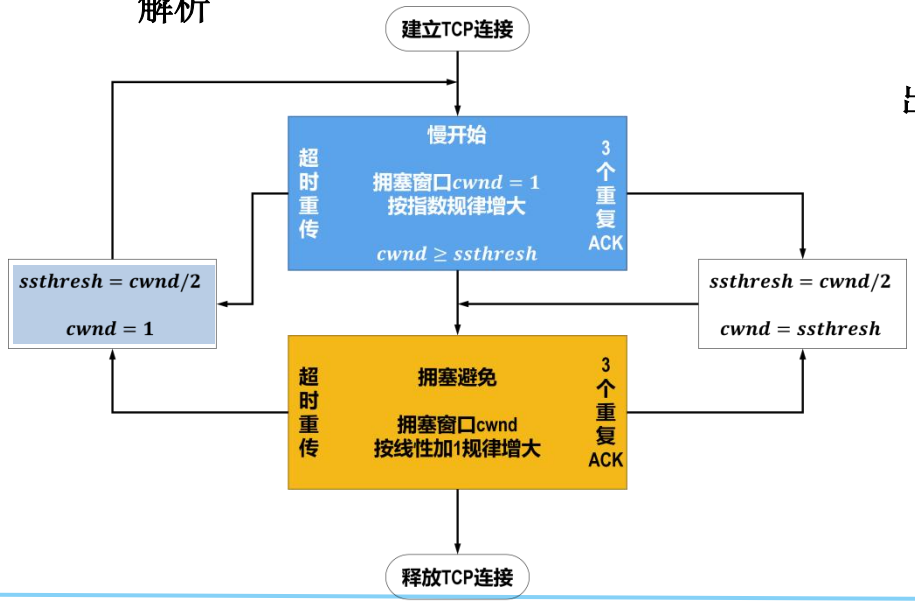
- 当 rwnd < cwnd 时，是接收方的接收能力限制发送窗口的最大值。
- 当 cwnd < rwnd 时，是网络拥塞限制发送窗口的最大值。

02 TCP的四种拥塞控制方法

【2009年 题39】一个TCP连接总是以1KB的最大段长发送TCP段，发送方有足够多的数据要发送。当拥塞窗口为16KB时发生了超时，如果接下来的4个RTT（往返时间）时间内的TCP段的传输都是成功的，那么当第4个RTT时间内发送的所有TCP段都得到肯定应答时，拥塞窗口大小是（ C ）。

A. 7KB　　B. 8KB　　C. 9KB　　D. 16KB

解析



出现超时重传时的处理
- ssthresh = 16 / 2 = 8
- cwnd = 1
- 开始慢开始算法

之后的

第1个传输轮次（RTT）结束　cwnd = 1 + 1 = 2

第2个传输轮次（RTT）结束　cwnd = 2 + 2 = 4

第3个传输轮次（RTT）结束　cwnd = 4 + 4 = 8

由于此时 cwnd = 8 = ssthresh，之后改为拥塞避免算法

第4个传输轮次（RTT）结束　cwnd = 8 + 1 = 9

TCP的四种拥塞控制方法

【2014年 题38】主机甲和主机乙已建立了TCP连接，甲始终以MSS=1KB大小的段发送数据，并一直有数据发送；乙每收到一个数据段都会发出一个接收窗口为10KB的确认段。若甲在t时刻发生超时时拥塞窗口为8KB，则从t时刻起，不再发生超时的情况下，经过10个RTT后，甲的发送窗口是（　　）。

A. 10KB　　B. 12KB　排除　　C. 14KB　排除　　D. 15KB　排除

A

解析

出现超时重传时的处理{ ssthresh = 8 / 2 = 4；cwnd = 1；开始慢开始算法

之后的

第1个传输轮次（RTT）结束　cwnd = 1 + 1 = 2

第2个传输轮次（RTT）结束　cwnd = 2 + 2 = 4

由于此时 cwnd = 4 = ssthresh，之后改为拥塞避免算法

第3个传输轮次（RTT）结束　cwnd = 4 + 1 = 5

第4个传输轮次（RTT）结束　cwnd = 5 + 1 = 6

第5个传输轮次（RTT）结束　cwnd = 6 + 1 = 7

第6个传输轮次（RTT）结束　cwnd = 7 + 1 = 8

第7个传输轮次（RTT）结束　cwnd = 8 + 1 = 9

第8个传输轮次（RTT）结束　cwnd = 9 + 1 = 10

第9个传输轮次（RTT）结束　cwnd = 10 + 1 = 11

第10个传输轮次（RTT）结束　cwnd = 11 + 1 = 12

swnd = min(cwnd, rwnd)

= min(12, 10) = 10

TCP的四种拥塞控制方法

【2015年 题39】主机甲和主机乙新建一个TCP连接，甲的拥塞控制初始阈值为32KB，甲向乙始终以MSS=1KB大小的段发送数据，并一直有数据发送；乙为该连接分配16KB接收缓存，并对每个数据段进行确认，忽略段传输延迟。若乙收到的数据全部存入缓存，不被取走，则甲从连接建立成功时刻起，未发生超时的情况下，经过4个RTT后，甲的发送窗口是（A）。

A. 1KB　　B. 8KB　　C. 16KB　　D. 32KB

解析

swnd = min(cwnd , rwnd)

初始状态　swnd = min(1 , 16) = 1

第1个RTT结束　swnd = min(1+1 , 15) = 2

第2个RTT结束　swnd = min(2+2 , 13) = 4

第3个RTT结束　swnd = min(4+4 , 9) = 8

第4个RTT结束　swnd = min(8+8 , 1) = 1

慢开始

TCP的四种拥塞控制方法

【2017年 题39】若甲向乙发起一个TCP连接，最大段长MSS=1KB，RTT=5ms，乙开辟的接收缓存为64KB，则

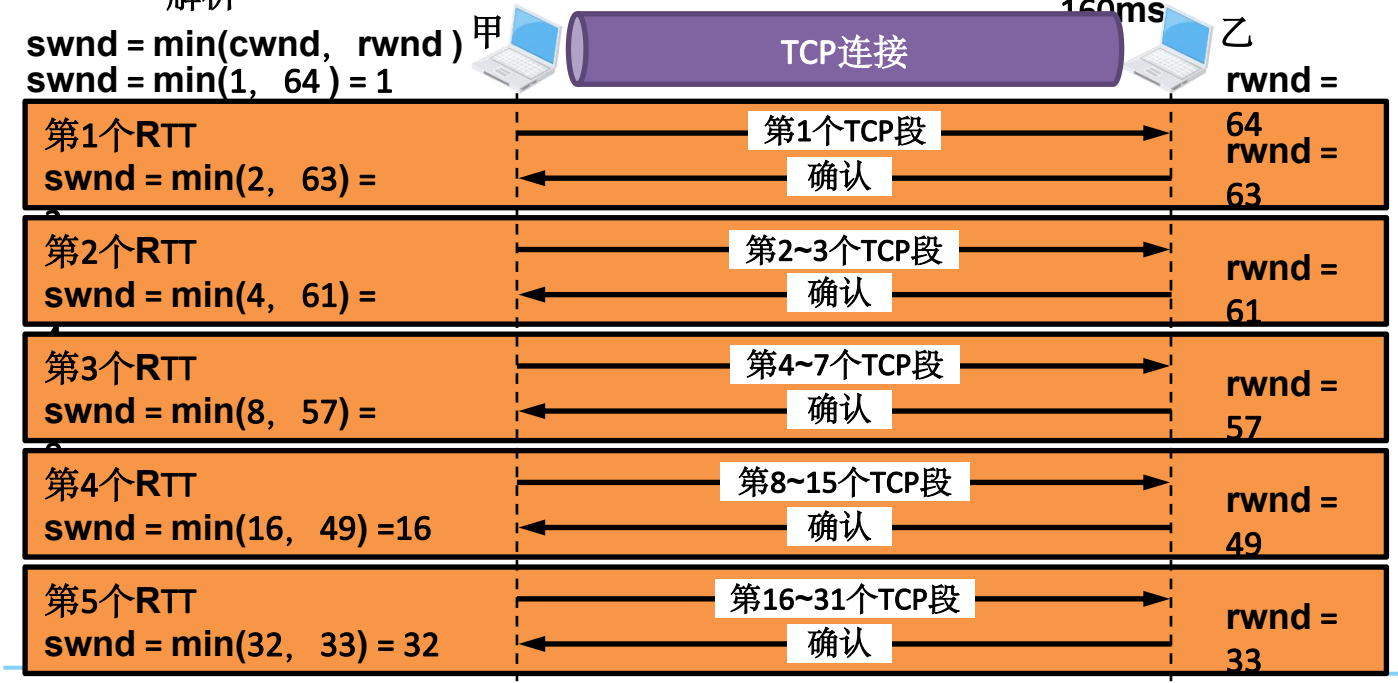
甲从连接建立成功至发送窗口达到32KB，需经过的时间至少是（ A ）。

A. 25ms　B. 30ms　C. 160ms　D. 165ms

解析

swnd = min(cwnd，rwnd)

swnd = min(1，64) = 1



甲从连接建立成功至发送窗口swnd达到32KB，至少（即没有出现超时重传）经历5个RTT。

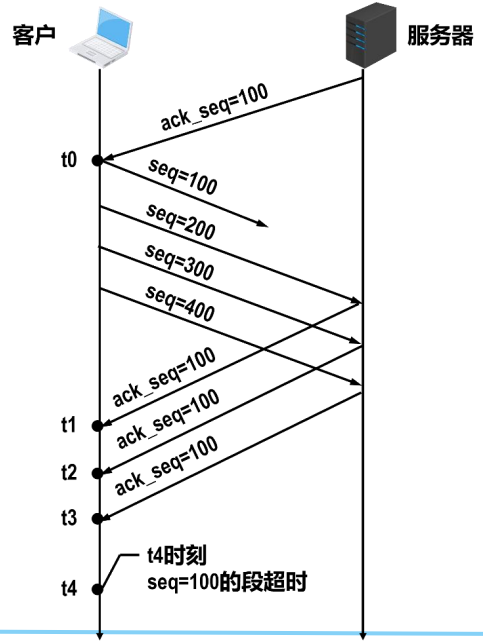
5ms × 5 = 25ms

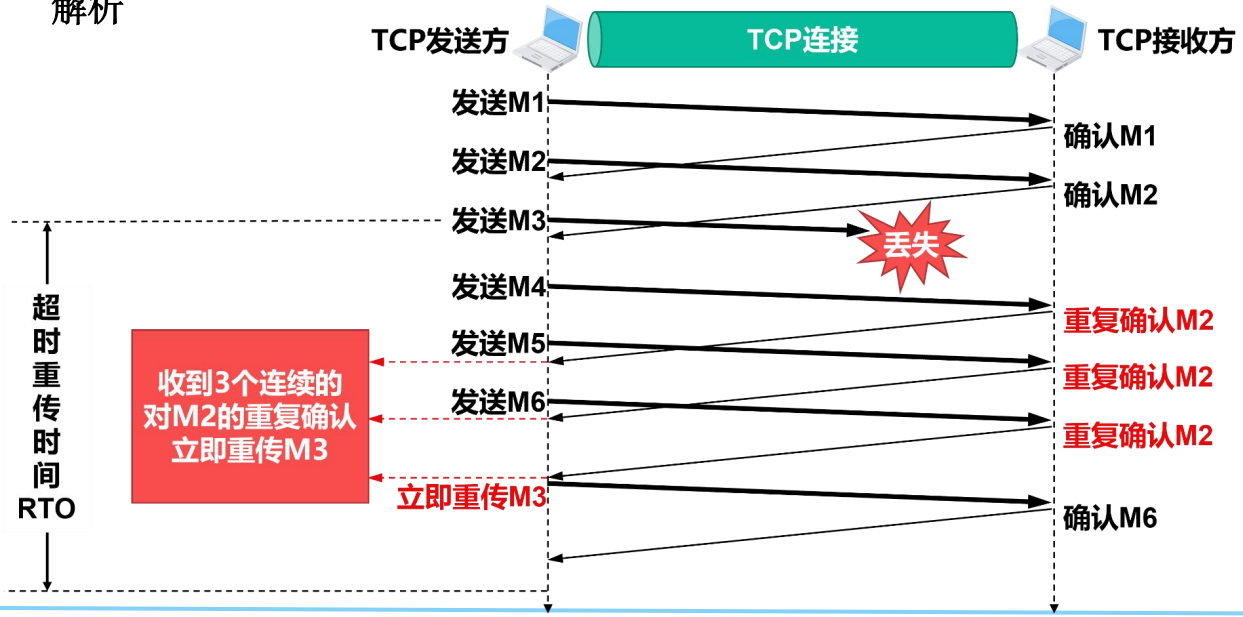
03 TCP的四种拥塞控制方法

【2019年 题38】若客户通过一个TCP连接向服务器发送数据的部分过程如下图所示。客户在t0时刻第一次收到确认序列号ack_seq=100的段，并发送序列号seq=100的段，但发生丢失。若TCP支持快速重传，则客户重新发送seq=100段的时刻是（　）。 C

A. t1　　B. t2　　C. t3　　D. t4

解析





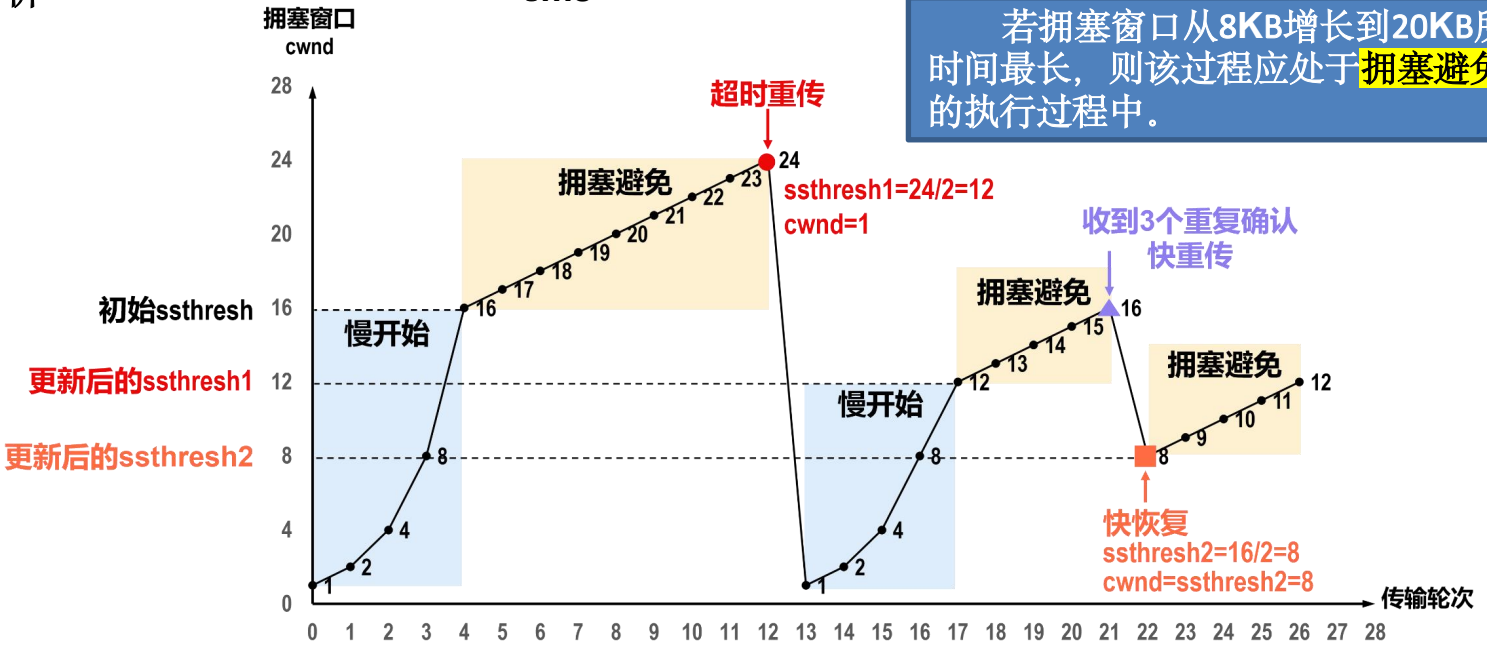
03 TCP的四种拥塞控制方法

【2020年 题38】若主机甲与主机乙已建立一条TCP连接，最大段长MSS为1KB，往返时间RTT为2ms，则在不出 现拥塞的前提下，拥塞窗口从8KB增长到20KB所需的最长时间是C（ ）。

A. 4ms　　B. 8ms　　C. 24ms　　D. 48ms

解析

若拥塞窗口从8KB增长到20KB所需的时间最长，则该过程应处于拥塞避免算法的执行过程中。



TCP的四种拥塞控制方法

【2020年 题38】若主机甲与主机乙已建立一条TCP连接，最大段长MSS为1KB，往返时间RTT为2ms，则在不出现拥塞的前提下，拥塞窗口从8KB增长到20KB所需的最长时间是（ C ）。

A. 4ms　　B. 8ms　　C. 24ms　　D. 48ms

解析

若拥塞窗口从8KB增长到20KB所需的时间最长，则该过程应处于拥塞避免算法的执行过程中。

在上述过程中：

第1个传输轮次（RTT）结束　cwnd = 8 + 1 = 9

第2个传输轮次（RTT）结束　cwnd = 9 + 1 = 10

⋮

第12个传输轮次（RTT）结束　cwnd = 19 + 1 = 20

共经历了12个传输轮次（RTT），总耗时为 2ms × 12 = 24ms

5.9 TCP 的运输连接管理

运输连接的三个阶段

- TCP 是面向连接的协议。
- TCP 连接有三个阶段：
  1. 连接建立
  2. 数据传送
  3. 连接释放
- TCP 的连接管理就是使 TCP 连接的建立和释放都能正常地进行。

TCP 连接建立过程中要解决的三个问题

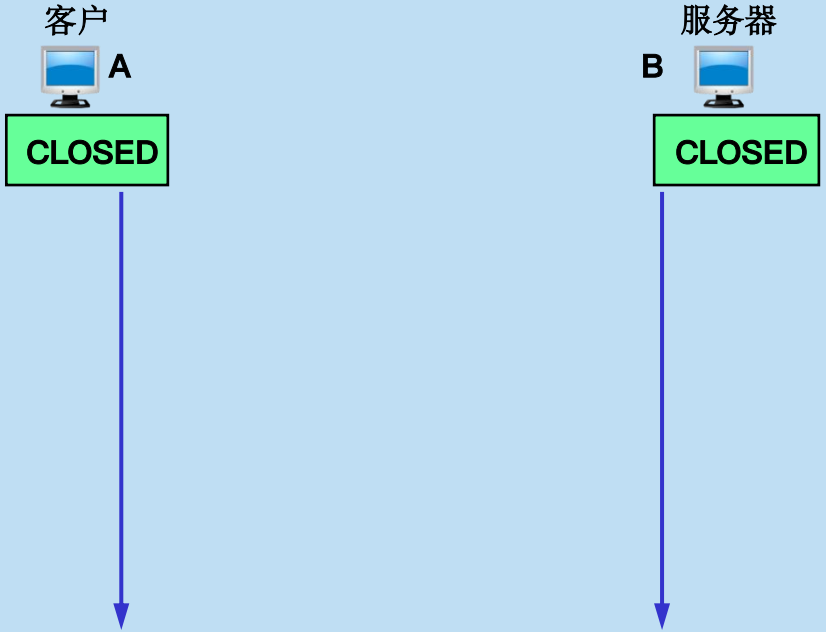
1. 要使每一方能够确知对方的存在。
2. 要允许双方协商一些参数（如最大窗口值、是否使用窗口扩大选项和时间戳选项以及服务质量等）。
3. 能够对运输实体资源（如缓存大小、连接表中的项目等）进行分配。

- 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)。

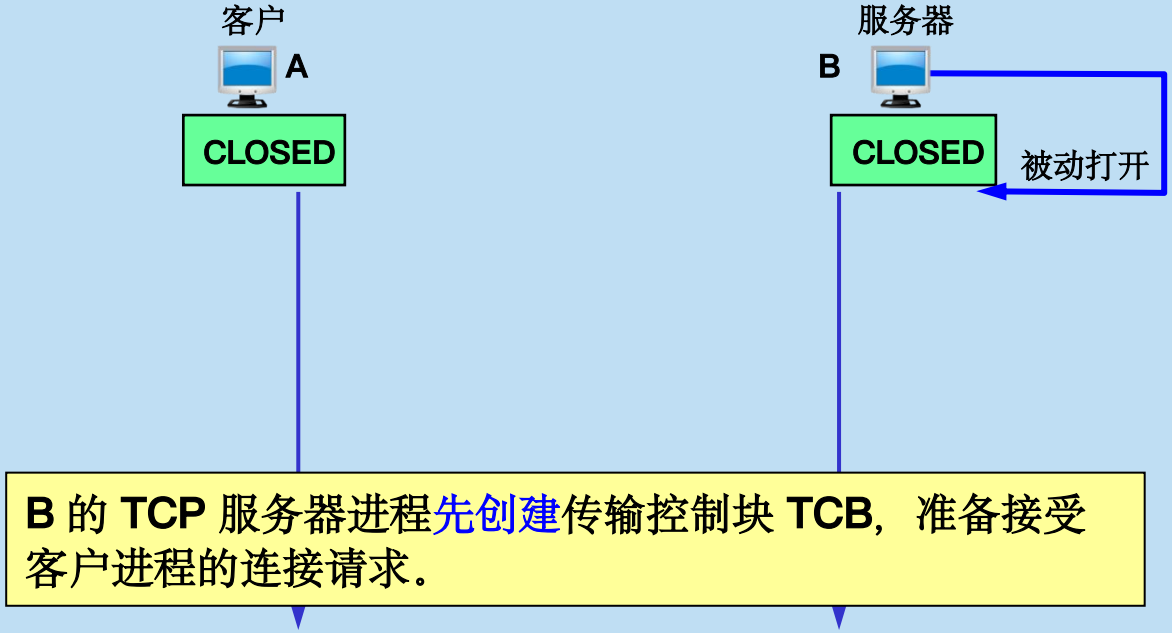
5.9.1 TCP 的连接建立

- TCP 建立连接的过程叫做握手。
- 采用三报文握手：在客户和服务器之间交换三个 TCP 报文段，以防止已失效的连接请求报文段突然又传送到了，因而产生 TCP 连接建立错误。

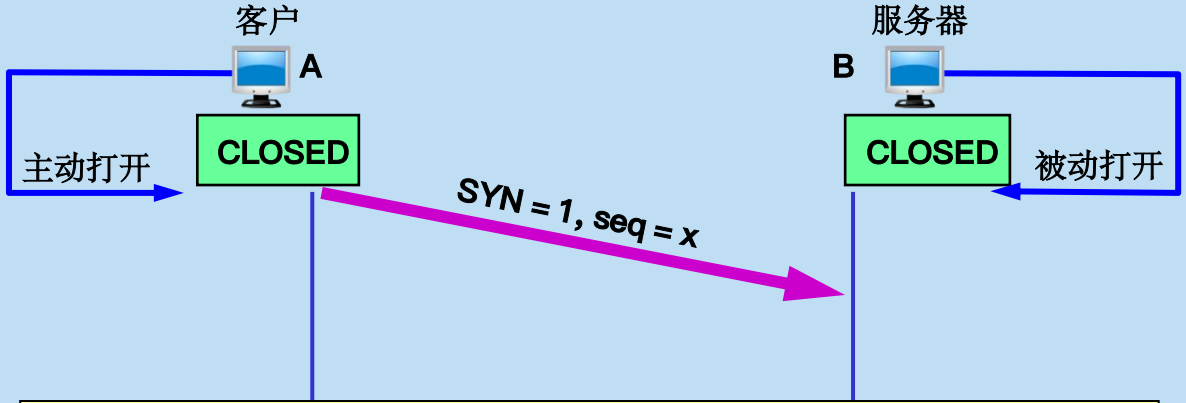
TCP 的连接建立：采用三报文握手



TCP 的连接建立：采用三报文握手



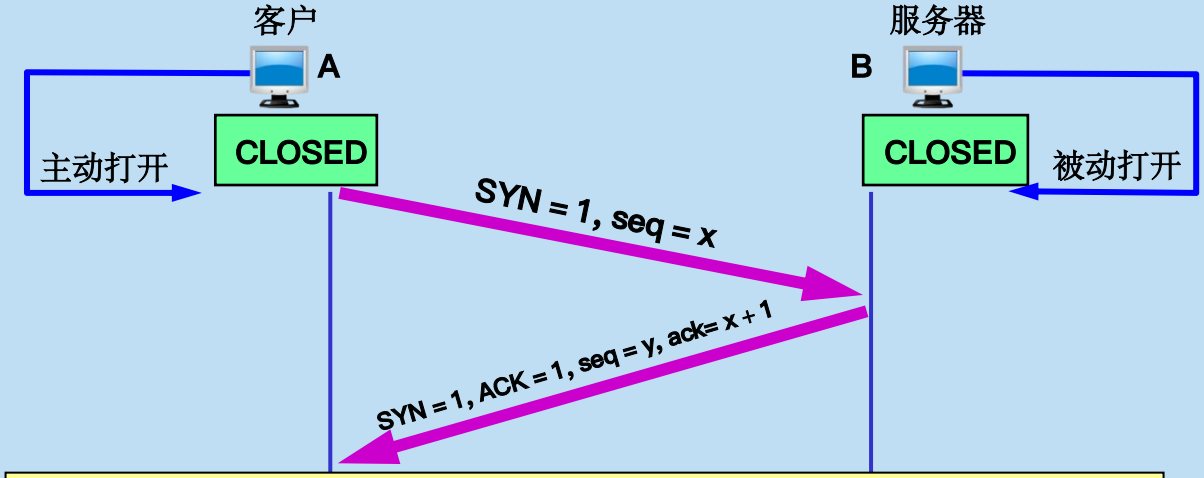
TCP 的连接建立：采用三报文握手



A 的 TCP 向 B 主动发出连接请求报文段，其首部中的同步位 SYN = 1，并选择序号 seq = x，表明传送数据时的第一个数据字节的序号是 x。

注意：TCP规定，SYN 报文段（即SYN = 1的报文段）不能携带数据，但要消耗掉一个序号。

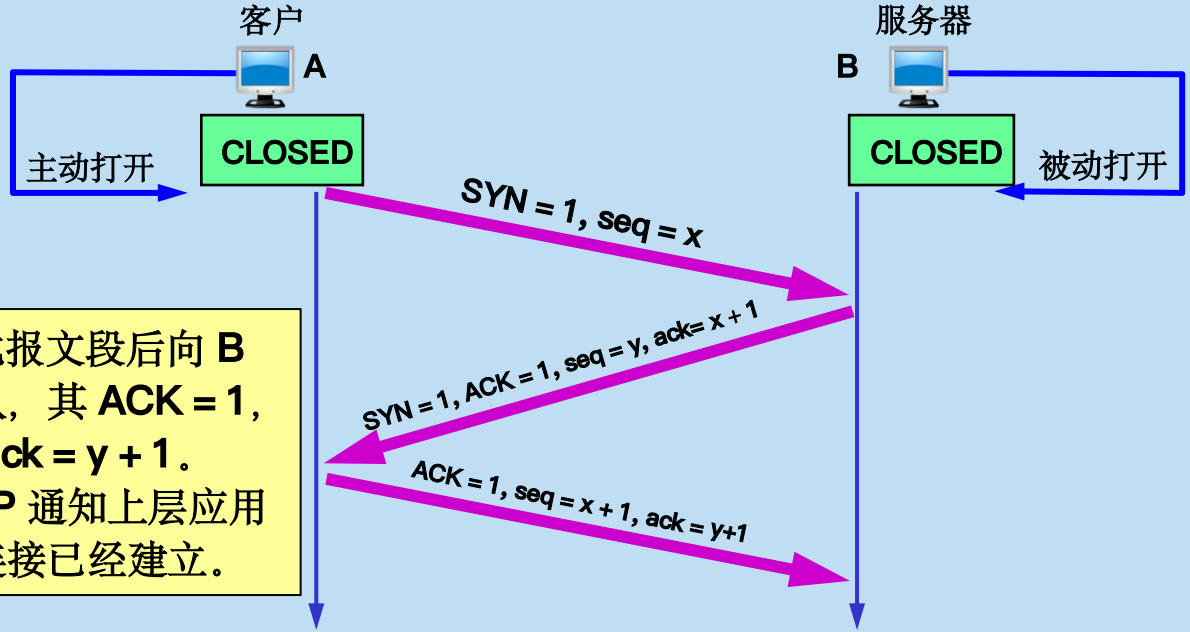
TCP 的连接建立：采用三报文握手



- B 的 TCP 收到连接请求报文段后，如同意，则发回确认。
- B 在确认报文段中应使 SYN = 1，使 ACK = 1，其确认号 ack = x + 1，自己选择的序号 seq = y。

这个报文段也不能携带数据，但同样要消耗掉一个序号。

TCP 的连接建立：采用三报文握手

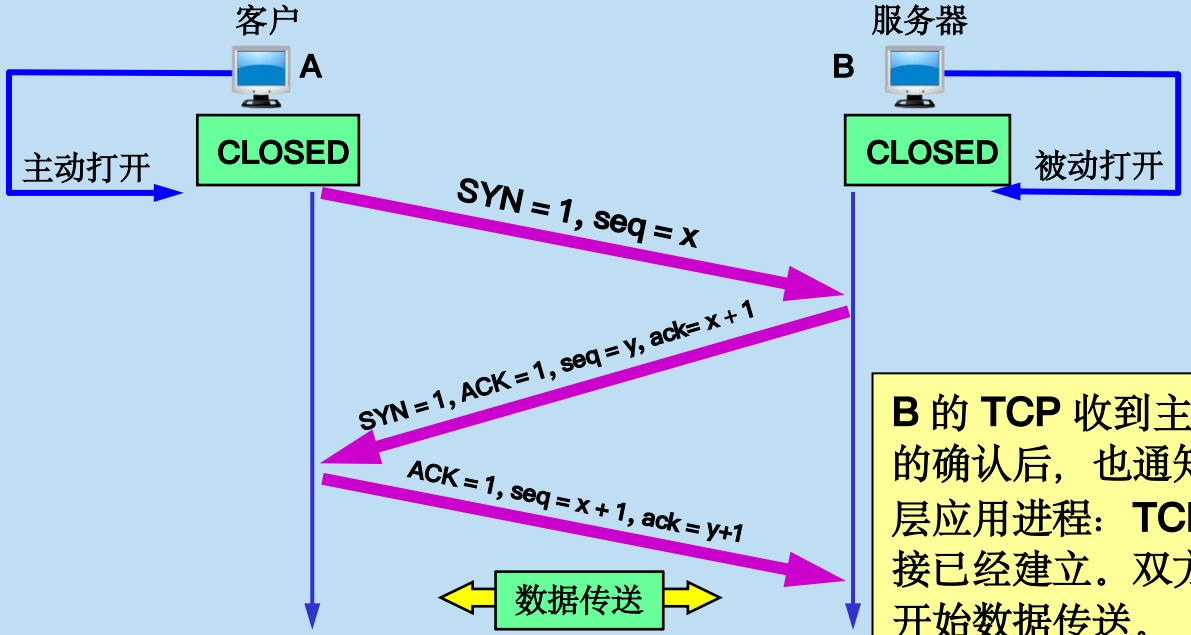


- A 收到此报文段后向 B 给出确认，其 ACK = 1，确认号 ack = y + 1。
- A 的 TCP 通知上层应用进程，连接已经建立。

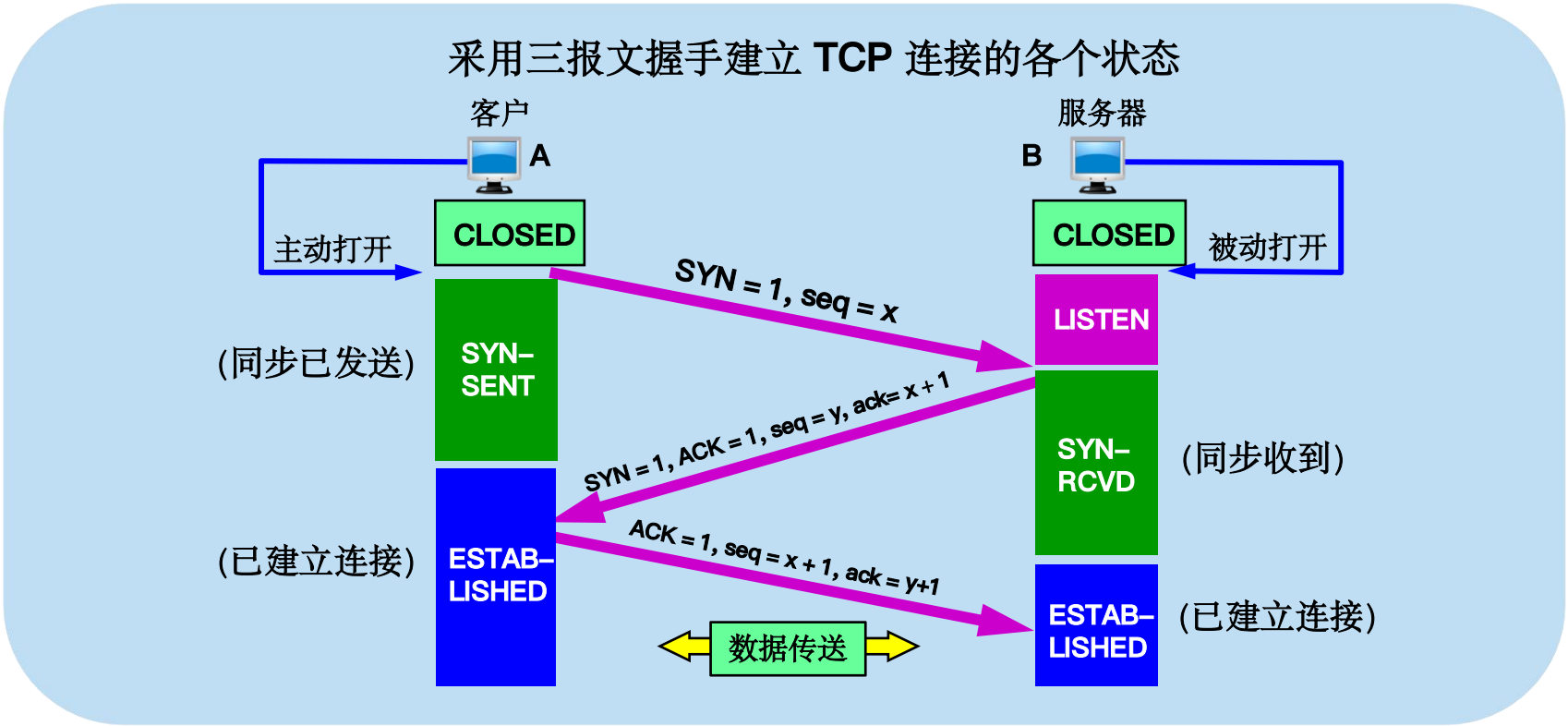
TCP 标准规定：ACK 报文段可以携带数据。

但如果不携带数据，则不消耗序号。下一个数据报文段的序号仍是 seq = x + 1。

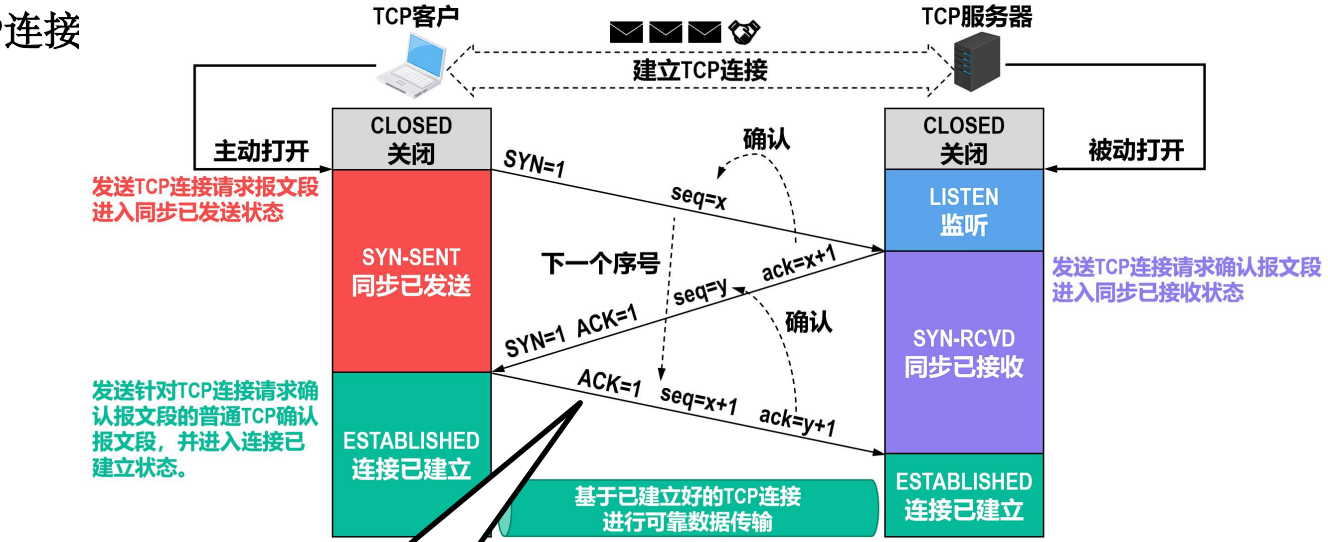
TCP 的连接建立：采用三报文握手



B 的 TCP 收到主机 A 的确认后，也通知其上层应用进程：TCP 连接已经建立。双方可以开始数据传送。

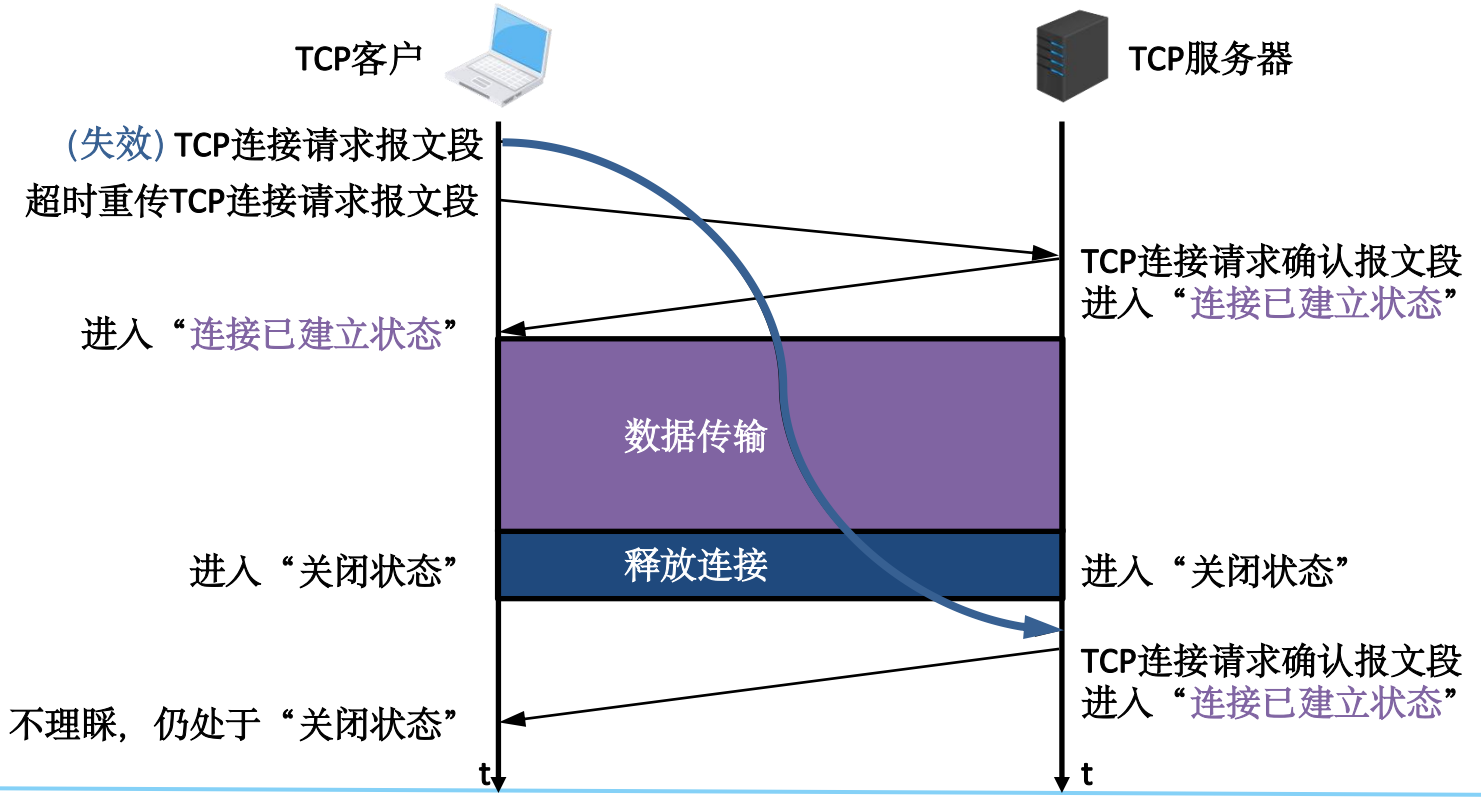

采用三报文握手建立 TCP 连接的各个状态
客户
服务器
A
B
CLOSED
CLOSED
主动打开
被动打开
SYN = 1, seq = x
LISTEN
(同步已发送)
SYN–SENT
SYN = 1, ACK = 1, seq = y, ack= x + 1
SYN–RCVD
(同步收到)
(已建立连接)
ESTAB–LISHED
ACK = 1, seq = x + 1, ack = y+1
ESTAB–LISHED
(已建立连接)
数据传送

“三报文握手”建立TCP连接

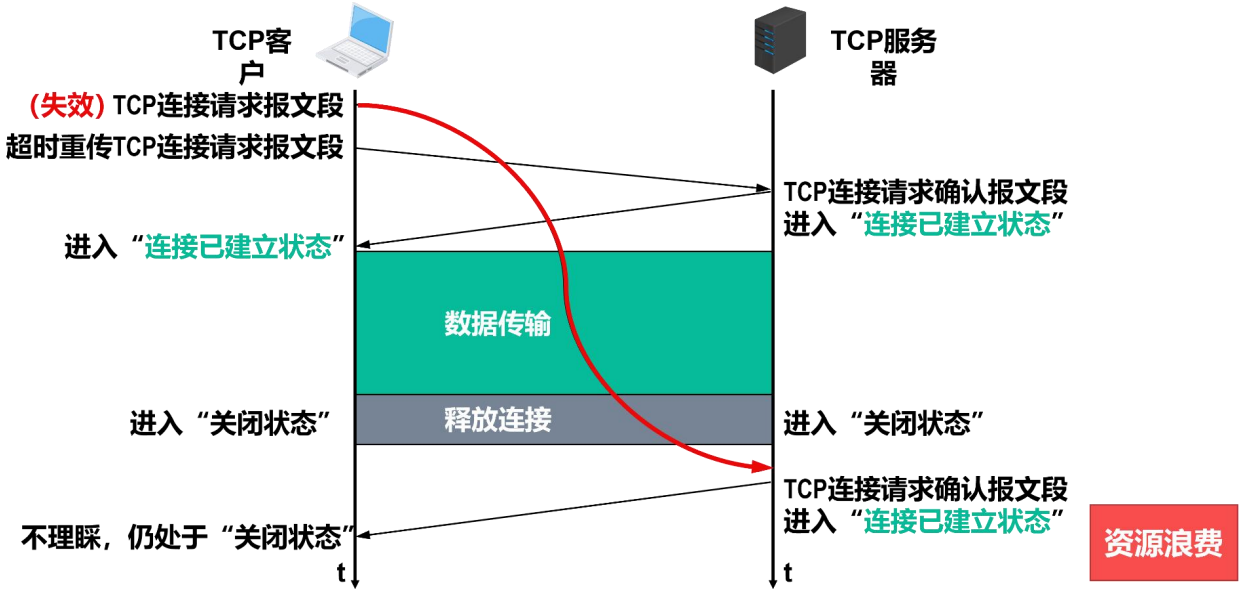


第三个TCP报文段
是否多余？

01 “三报文握手”建立TCP连接 —— 使用“三报文握手”而不是“两报文握手”建立TCP连接的原因



01 “三报文握手”建立TCP连接 —— 使用“三报文握手”而不是“两报文握手”建立TCP连接的原因



采用“三报文握手”而不是“两报文握手”来建立TCP连接，是为了防止已失效的TCP连接请求报文段突然又传送到了TCP服务器进程，因而导致错误。

01 “三报文握手”建立TCP连接

【2011年 题39】主机甲向主机乙发送一个（SYN=1，seq=11220）的TCP段，期望与主机乙建立TCP连接，若主机乙接受该连接请求，则主机乙向主机甲发送的正确的TCP段可能是（ ）。 C

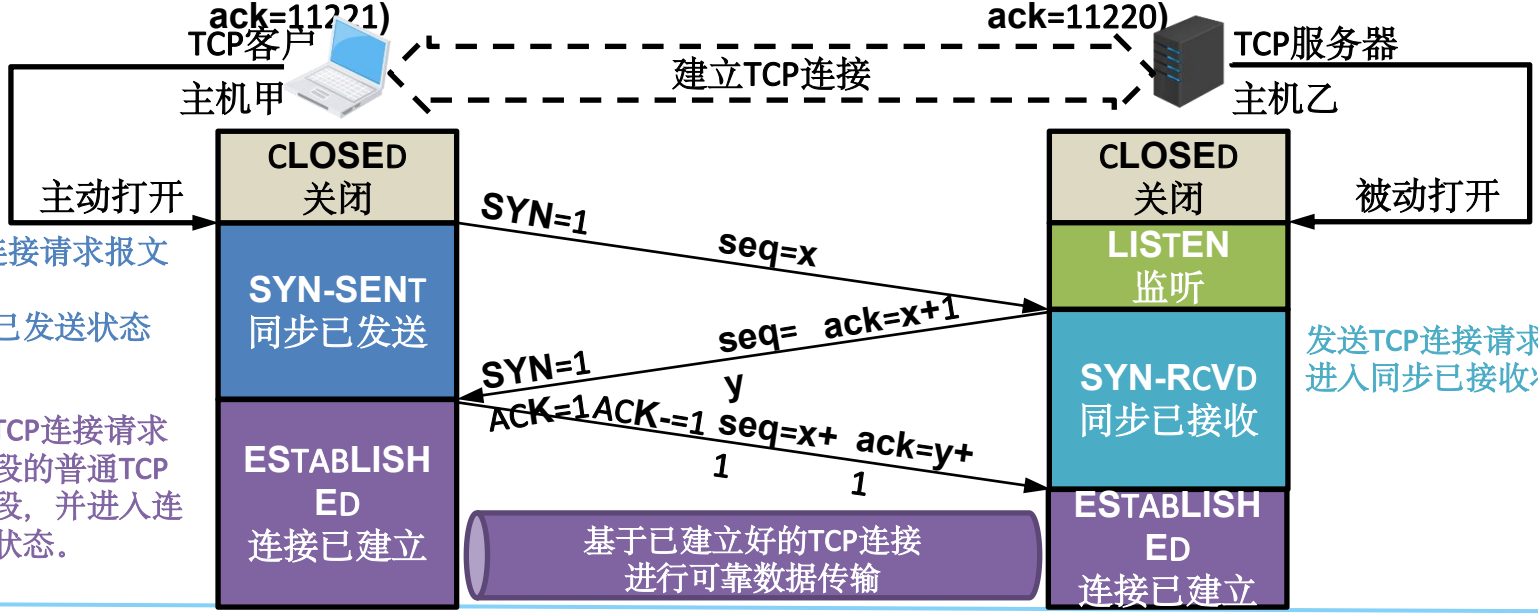
A. (SYN=0，ACK=0，seq=11221，ack=11221)

B. (SYN=1，ACK=1，seq=11220，ack=11220)

C. (SYN=1，ACK=1，seq=11221，ack=11221)

D. (SYN=0，ACK=0，seq=11220，ack=11220)

解析



“三报文握手”建立TCP连接

【2011年 题39】主机甲向主机乙发送一个（SYN=1，seq=11220）的TCP段，期望与主机乙建立TCP连接，若主 机乙接受该连接请求，则主机乙向主机甲发送的正确的TCP段可能是（ ）。 C

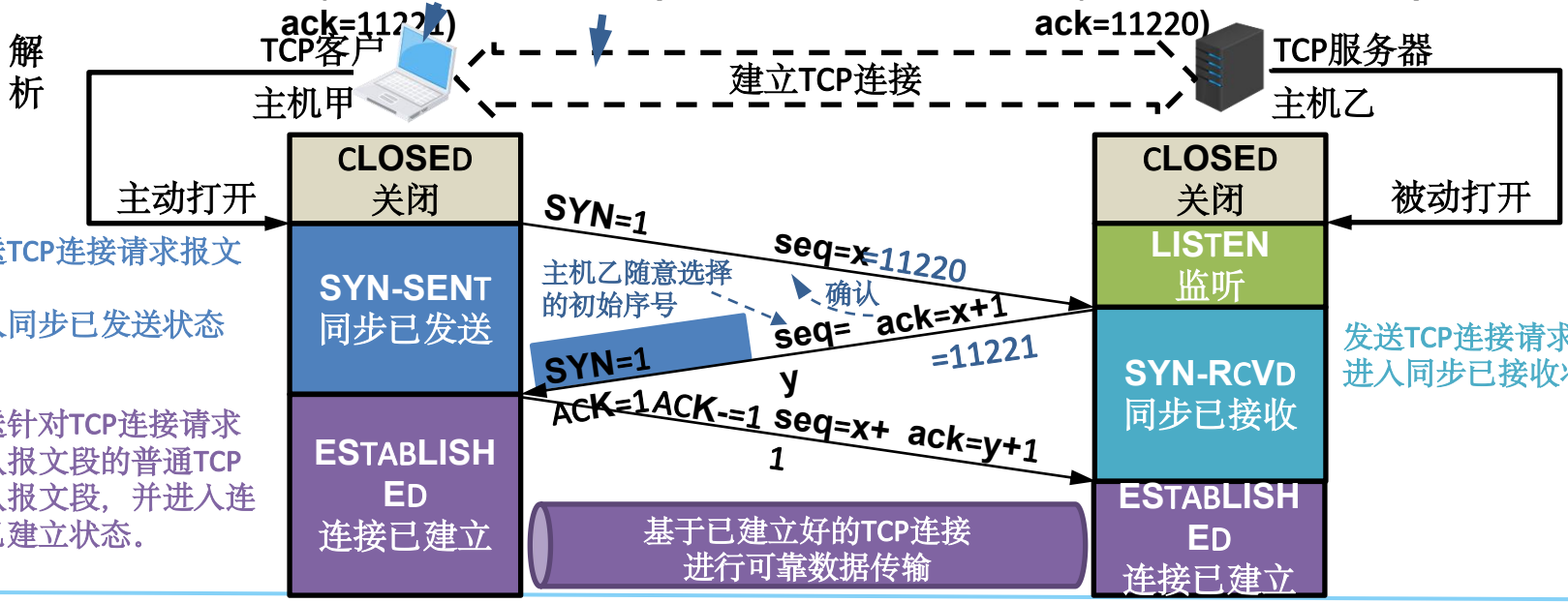
A. (SYN=0，ACK=0，seq=11221，ack=11221)

B. (SYN=1，ACK=1，seq=11220，ack=11220)

C. (SYN=1，ACK=1，seq=11221，ack=11221)

D. (SYN=0，ACK=0，seq=11220，ack=11220)

解析

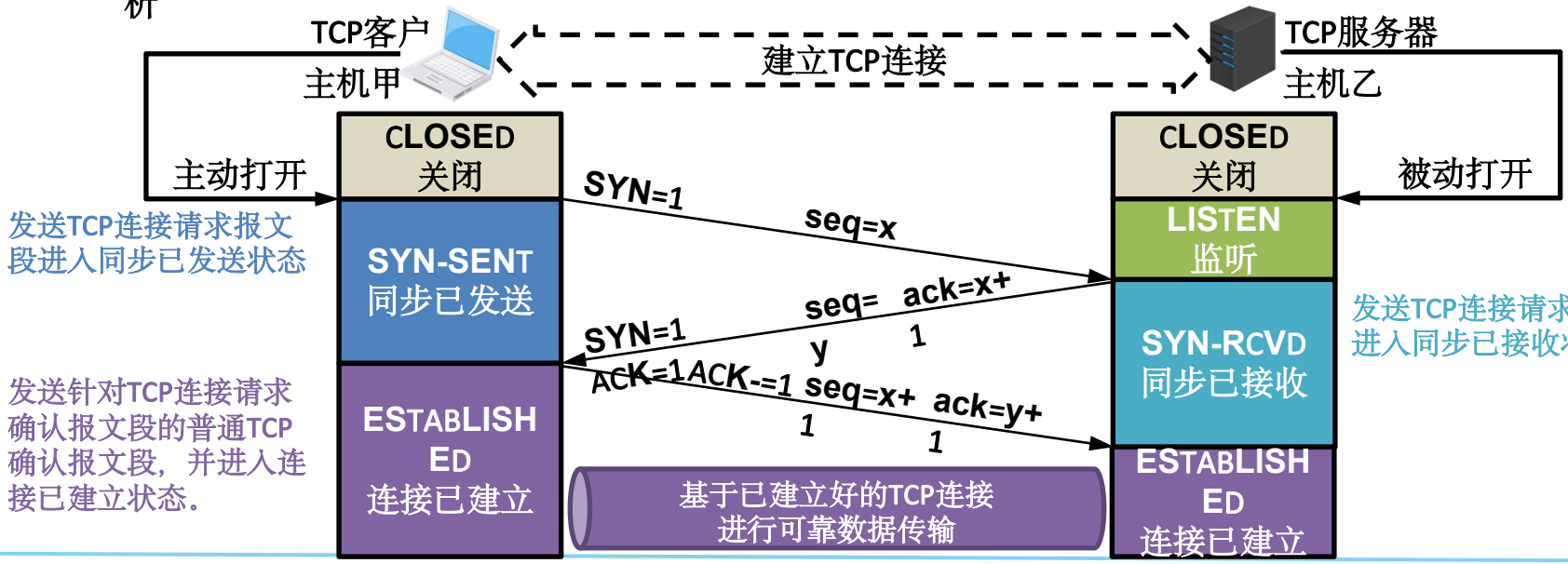


“三报文握手”建立TCP连接

【2019年 题39】若主机甲主动发起一个与主机乙的TCP连接，甲、乙选择的初始序列号分别为2018和2046，则第三次握手TCP段的确认序列号是（D）。

A. 2018　　B. 2019　　C. 2046　　D. 2047

解析

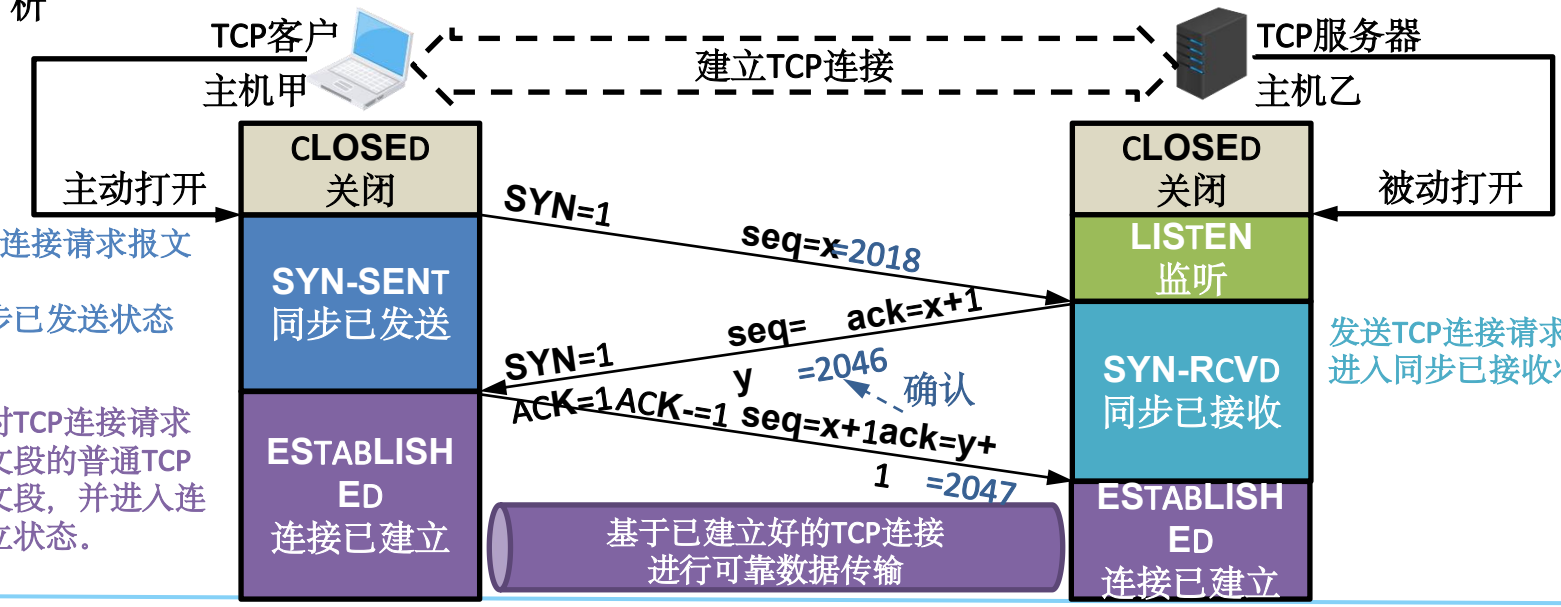


“三报文握手”建立TCP连接

【2019年 题39】若主机甲主动发起一个与主机乙的TCP连接，甲、乙选择的初始序列号分别为2018和2046，则第三次握手TCP段的确认序列号是（D）。

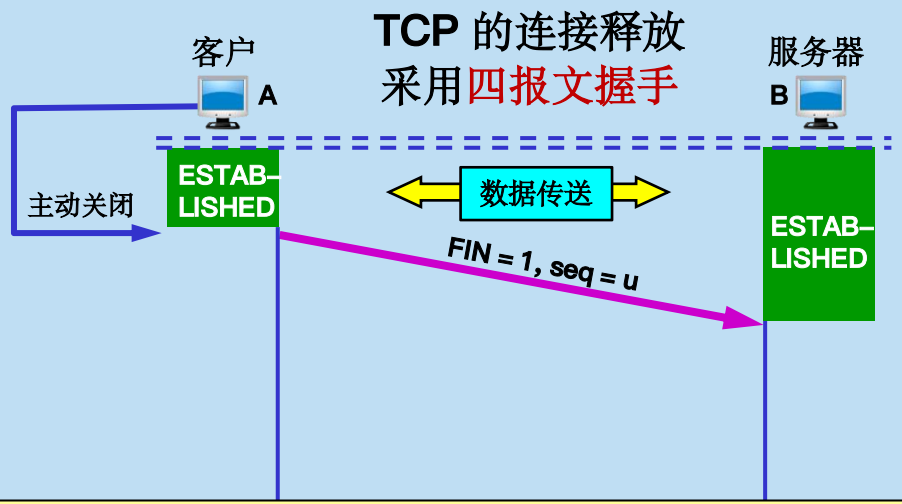
A. 2018　　B. 2019　　C. 2046　　D. 2047

解析



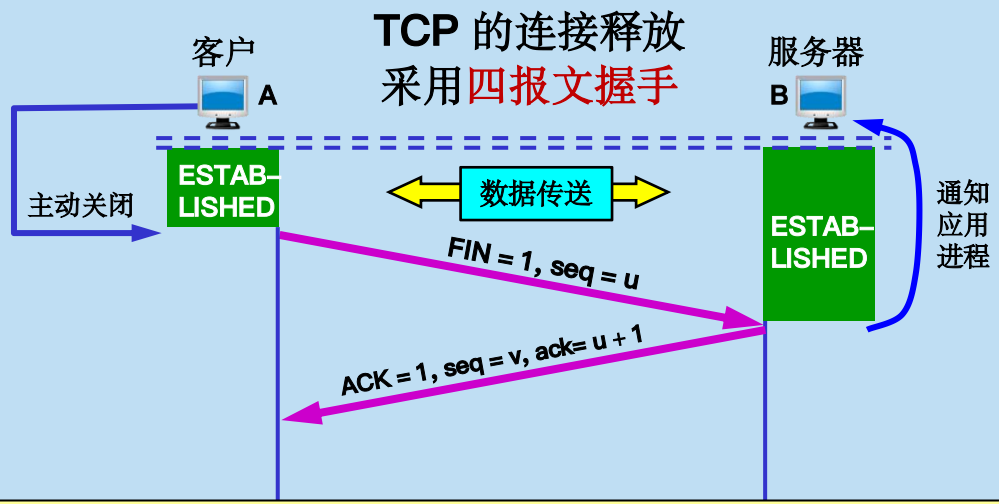
5.9.2 TCP 的连接释放

- TCP 连接释放过程比较复杂。
- 数据传输结束后，通信的双方都可释放连接。
- TCP 连接释放过程是四报文握手。

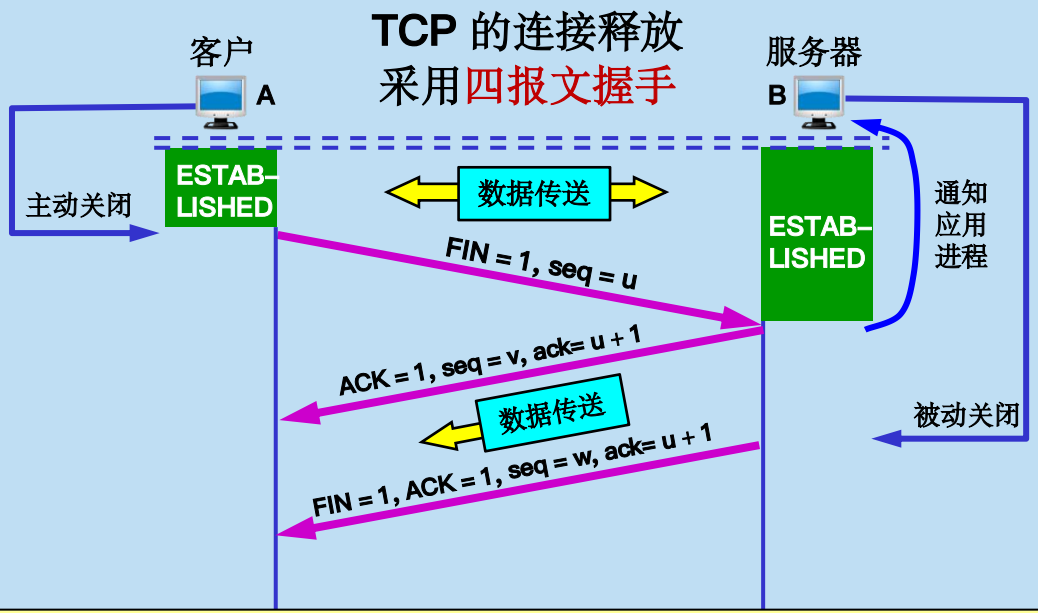


- A 的应用进程先向其 TCP 发出连接释放报文段，并停止再发送数据，主动关闭 TCP 连接。
- A 把连接释放报文段首部的 FIN = 1，其序号seq = u，等待 B 的确认。

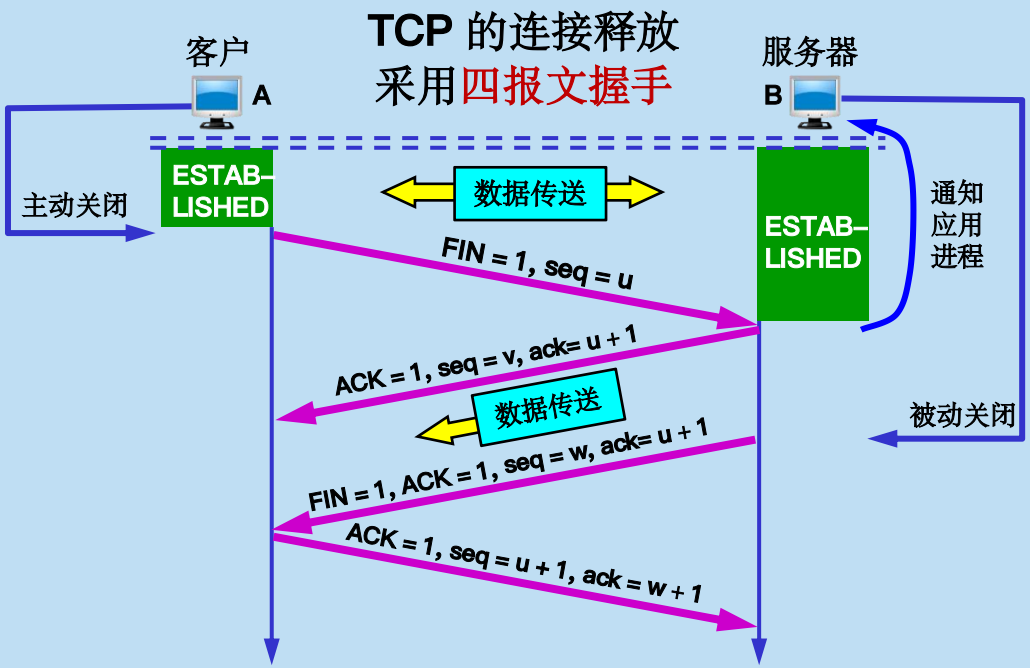
TCP规定：FIN 报文段即使不携带数据，也消耗掉一个序号。



- B 发出确认，ACK=1，确认号 ack = u+1，这个报文段的序号 seq = v。
- TCP 服务器进程通知高层应用进程。
- 从 A 到 B 这个方向的连接就释放了，TCP 连接处于半关闭 (half-close) 状态。B 若发送数据，A 仍要接收。

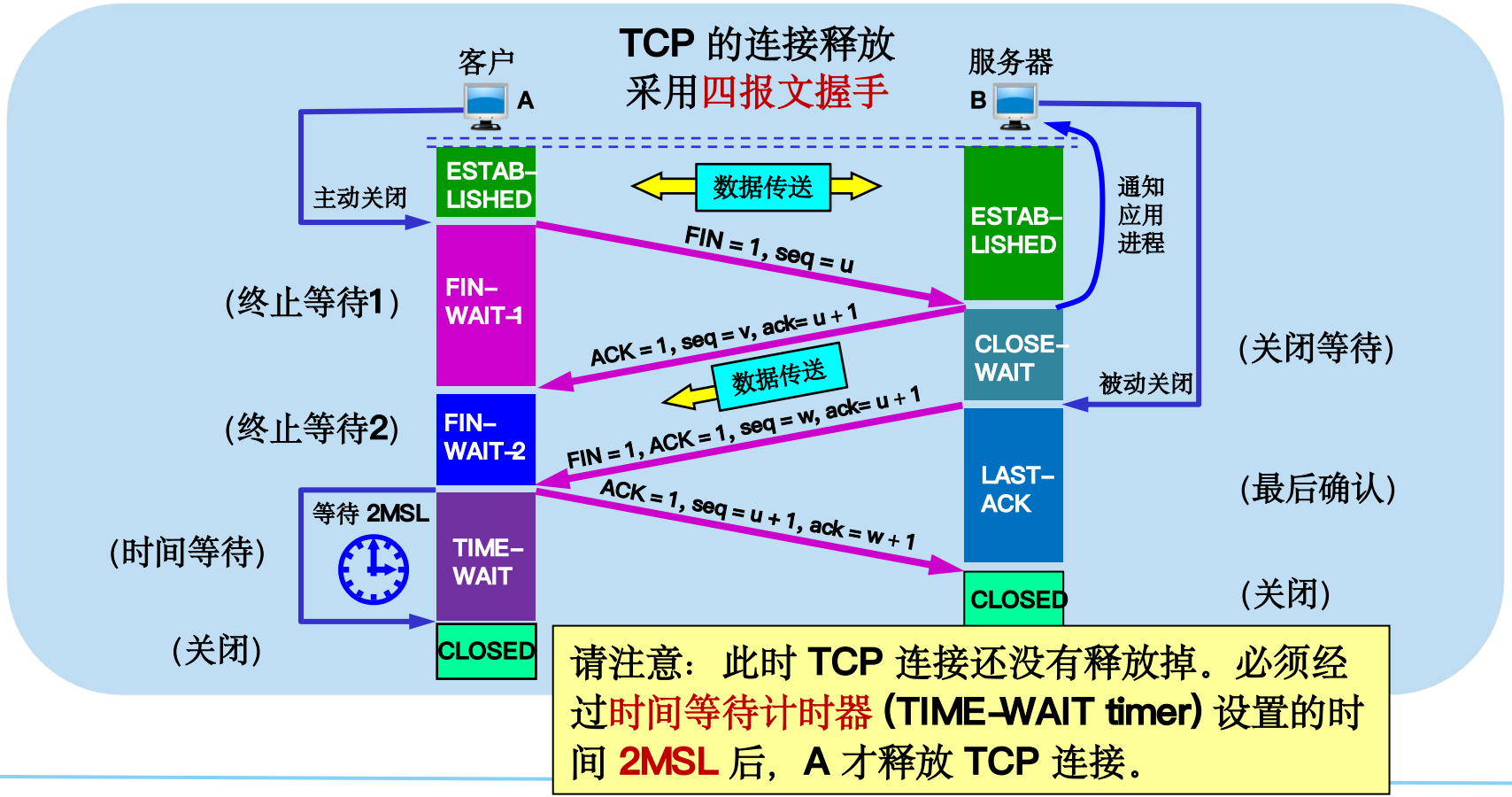


- 若 B 已经没有要向 A 发送的数据，其应用进程就通知 TCP 释放连接。
- FIN=1，ACK=1，确认号 ack = u+1。

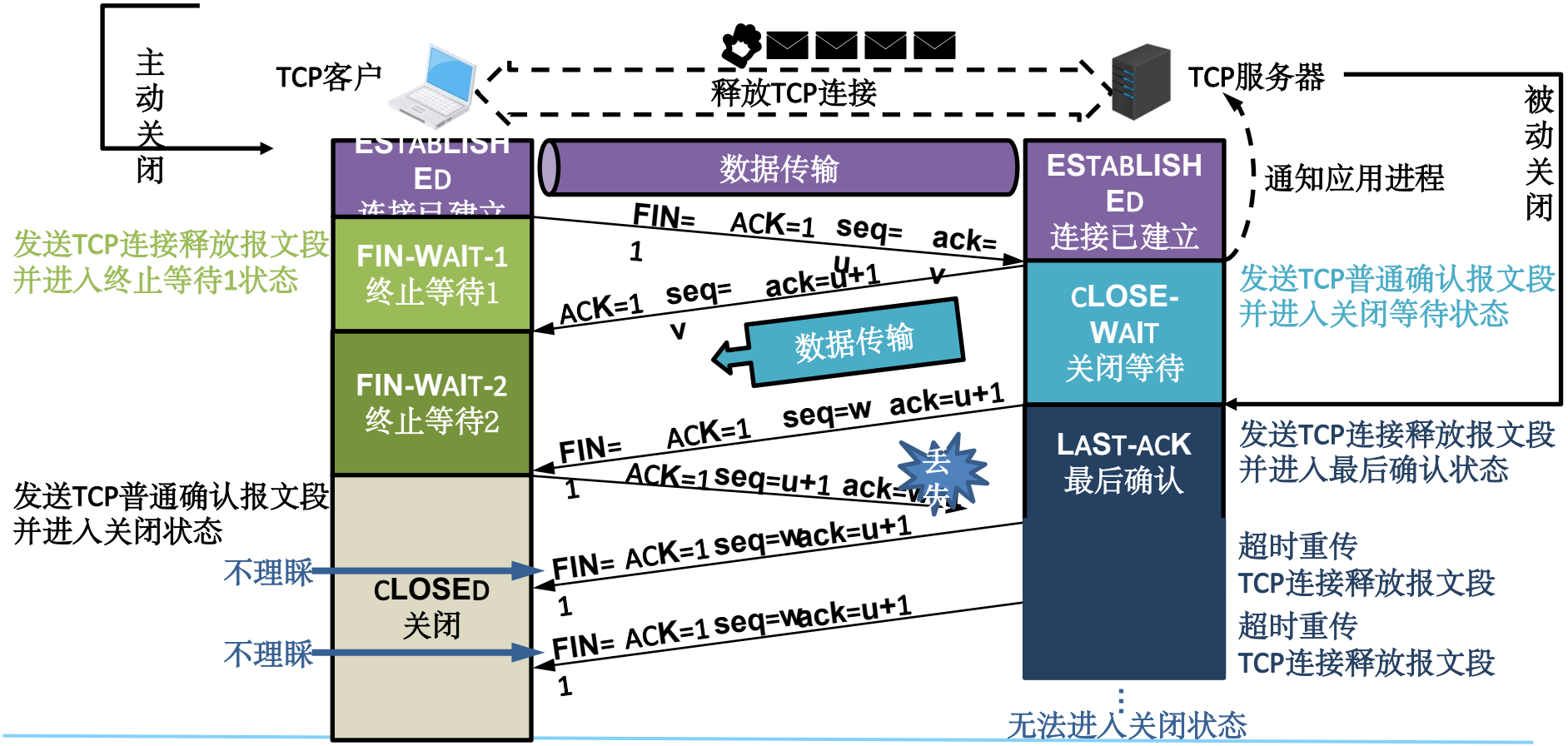


A 收到连接释放报文段后，必须发出确认。

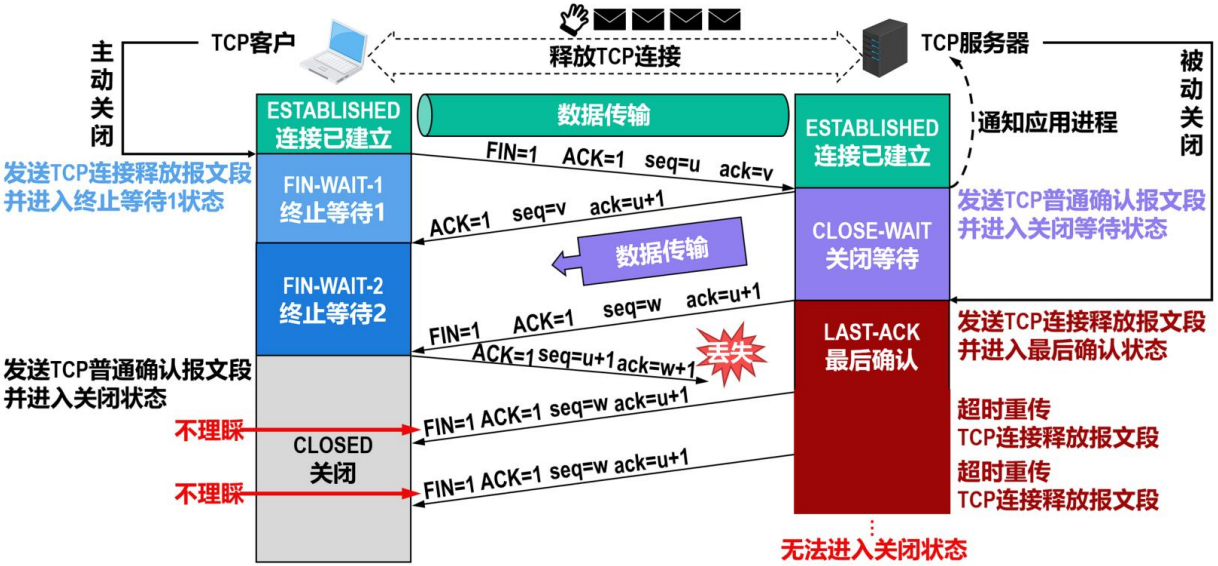
ACK=1，确认号 ack=w+1，自己的序号 seq = u + 1


TCP 的连接释放
采用四报文握手
客户
A
服务器
B
主动关闭
ESTAB-LISHED
数据传送
ESTAB-LISHED
通知应用进程
FIN = 1, seq = u
(终止等待1)
FIN-WAIT-1
ACK = 1, seq = v, ack= u + 1
CLOSE-WAIT
(关闭等待)
数据传送
被动关闭
(终止等待2)
FIN-WAIT-2
FIN = 1, ACK = 1, seq = w, ack= u + 1
LAST-ACK
(最后确认)
等待 2MSL
ACK = 1, seq = u + 1, ack = w + 1
(时间等待)
TIME-WAIT
CLOSED
(关闭)
(关闭)
CLOSED
请注意：此时 TCP 连接还没有释放掉。必须经过时间等待计时器 (TIME-WAIT timer) 设置的时间 2MSL 后，A 才释放 TCP 连接。

“四报文挥手”释放TCP连接



“四报文挥手”释放TCP连接



- 处于时间等待（TIME-WAIT）状态后要经过2MSL时长，可以确保TCP服务器进程能够收到最后一个TCP确认报文段而进入关闭（CLOSED）状态。
- 另外，TCP客户进程在发送完最后一个TCP确认报文段后，再经过2MSL时长，就可以使本次连接持续时间内所产生的的所有报文段都从网络中消失。这样就可以使下一个新的TCP连接中不会出现旧连接中的报文段。

必须等待 2MSL 的时间

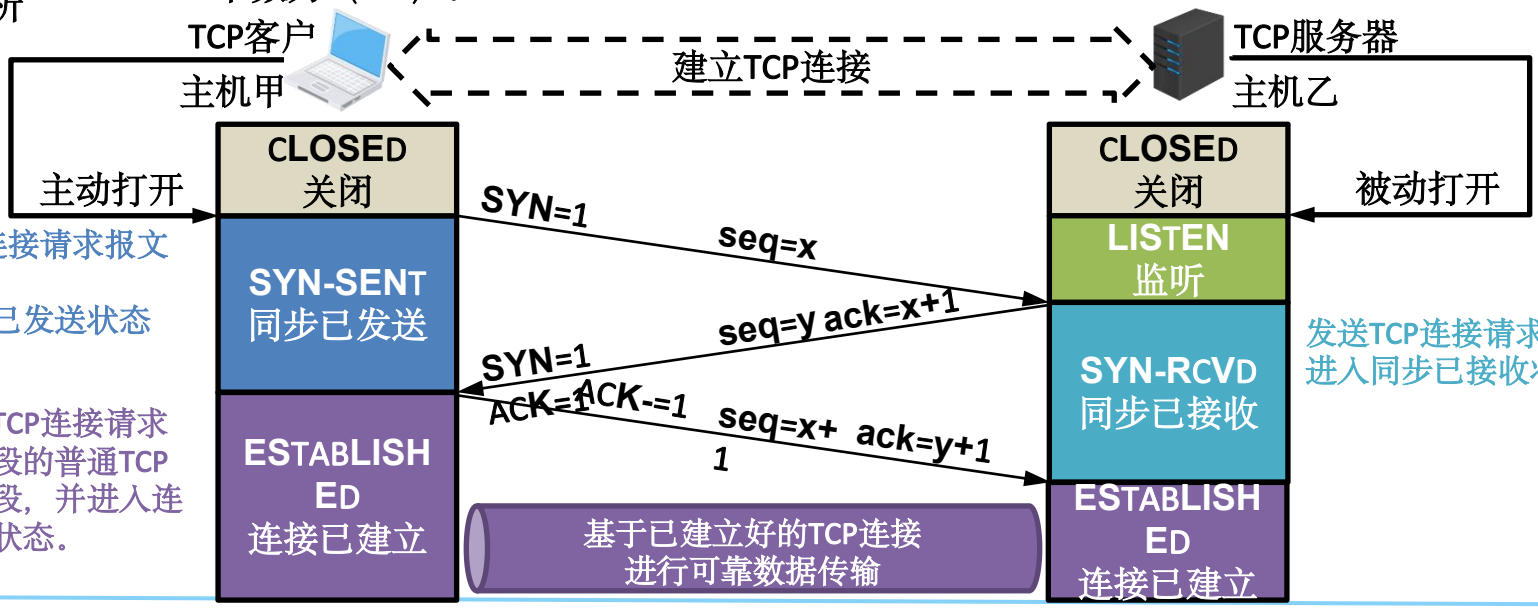
- 第一，保证发送的最后一个 ACK 报文段能够到达 B。
- 第二，防止“已失效的连接请求报文段”出现在本连接中。

“四报文挥手”释放TCP连接

【2020年 题39】若主机甲与主机乙建立TCP连接时发送的SYN段中的序号为1000，在断开连接时，甲发送给乙的FIN段中的序号为5001，则在无任何重传的情况下，甲向乙已经发送的应用层数据的字节数为（　　）。

A. 4002　　B. 4001　　C. 4000　　D. 3999

解析



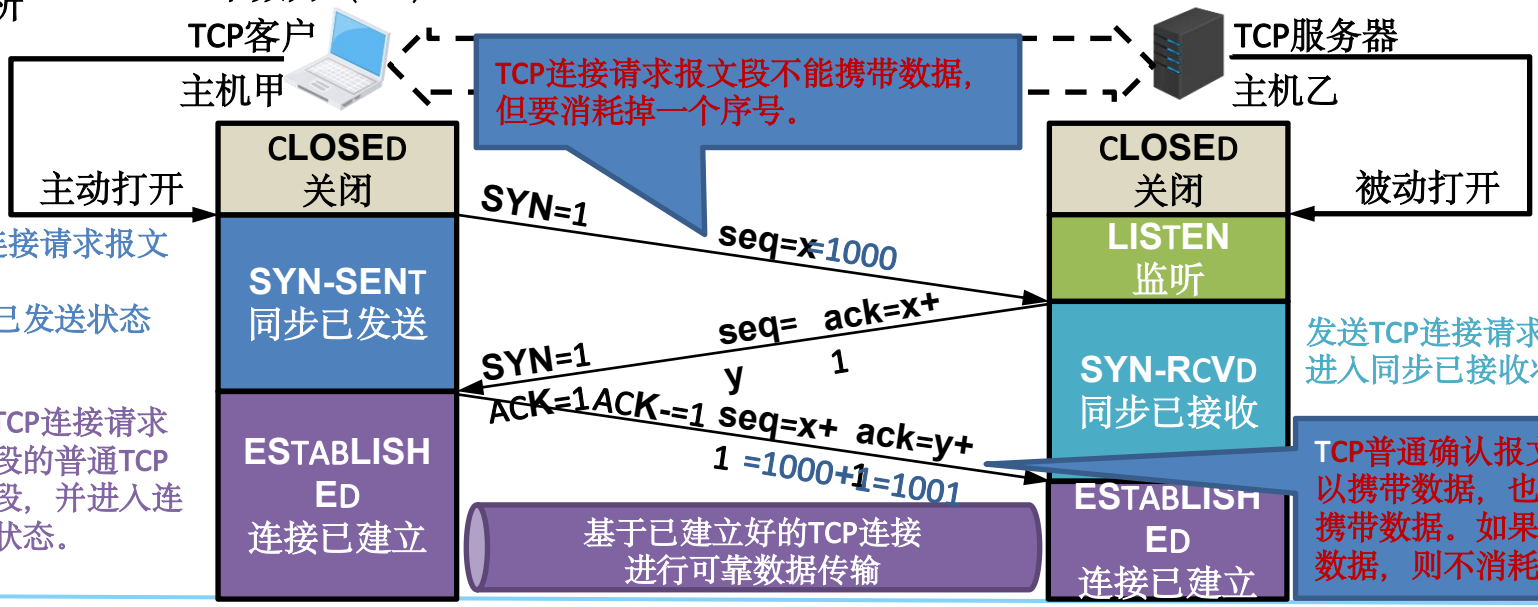
“四报文挥手”释放TCP连接

【2020年 题39】若主机甲与主机乙建立TCP连接时发送的SYN段中的序号为1000，在断开连接时，甲发送给乙的FIN段中的序号为5001，则在无任何重传的情况下，甲向乙已经发送的应用层数据的字节数为（　　）。

A. 4002　　B. 4001　　C. 4000　　D. 3999

C

解析



“四报文挥手”释放TCP连接

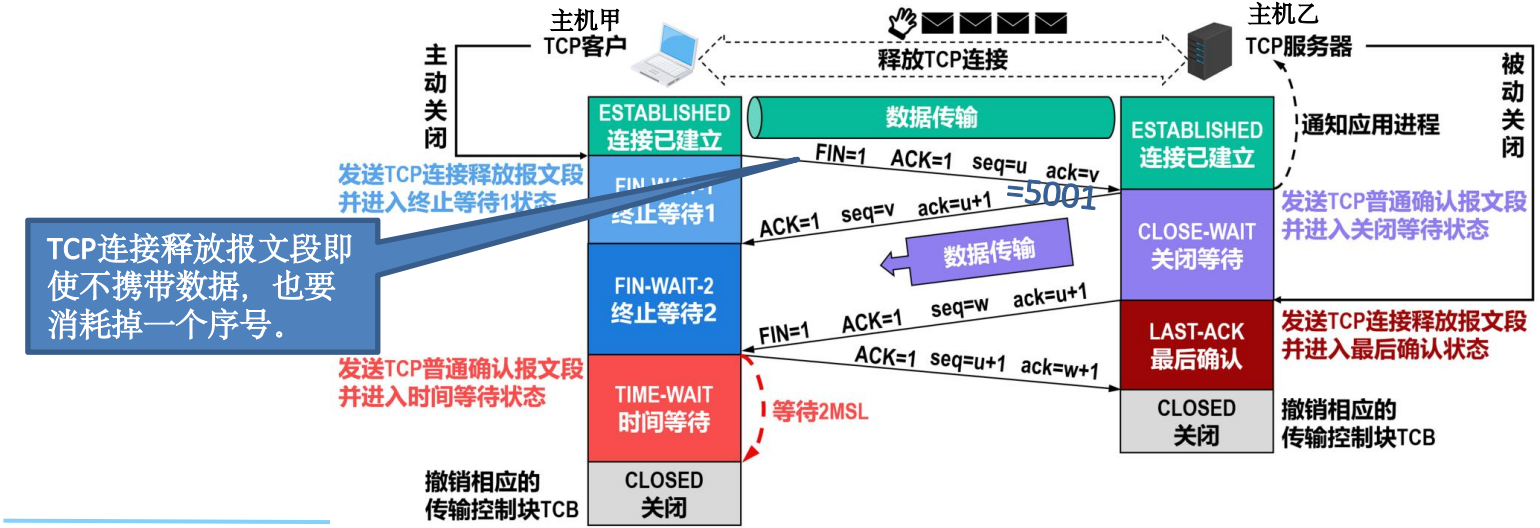
【2020年 题39】若主机甲与主机乙建立TCP连接时发送的SYN段中的序号为1000，在断开连接时，甲发送给乙的FIN段中的序号为5001，则在无任何重传的情况下，甲向乙已经发送的应用层数据的字节数为（ ）。

A. 4002　　B. 4001　　C. 4000　　D. 3999

C

解析

甲给乙发送的第一个应用层数据字节的TCP序号为1001，因为应用层数据作为数据载荷被封装在TCP报文段中。甲在发送FIN段之前，给乙发送的最后一个应用层数据字节的TCP序号为5000。



“四报文挥手”释放TCP连接

【2020年 题39】若主机甲与主机乙建立TCP连接时发送的SYN段中的序号为1000，在断开连接时，甲发送给乙的FIN段中的序号为5001，则在无任何重传的情况下，甲向乙已经发送的应用层数据的字节数为（ C ）。

A. 4002　　B. 4001　　C. 4000　　D. 3999

解析

甲给乙发送的第一个应用层数据字节的TCP序号为1001，因为应用层数据作为数据载荷被封装在TCP报文段中。

甲在发送FIN段之前，给乙发送的最后一个应用层数据字节的TCP序号为5000。

综上所述，甲向乙已经发送了字节序号为1001~5000共4000个字节的应用层数据。