TCP协议

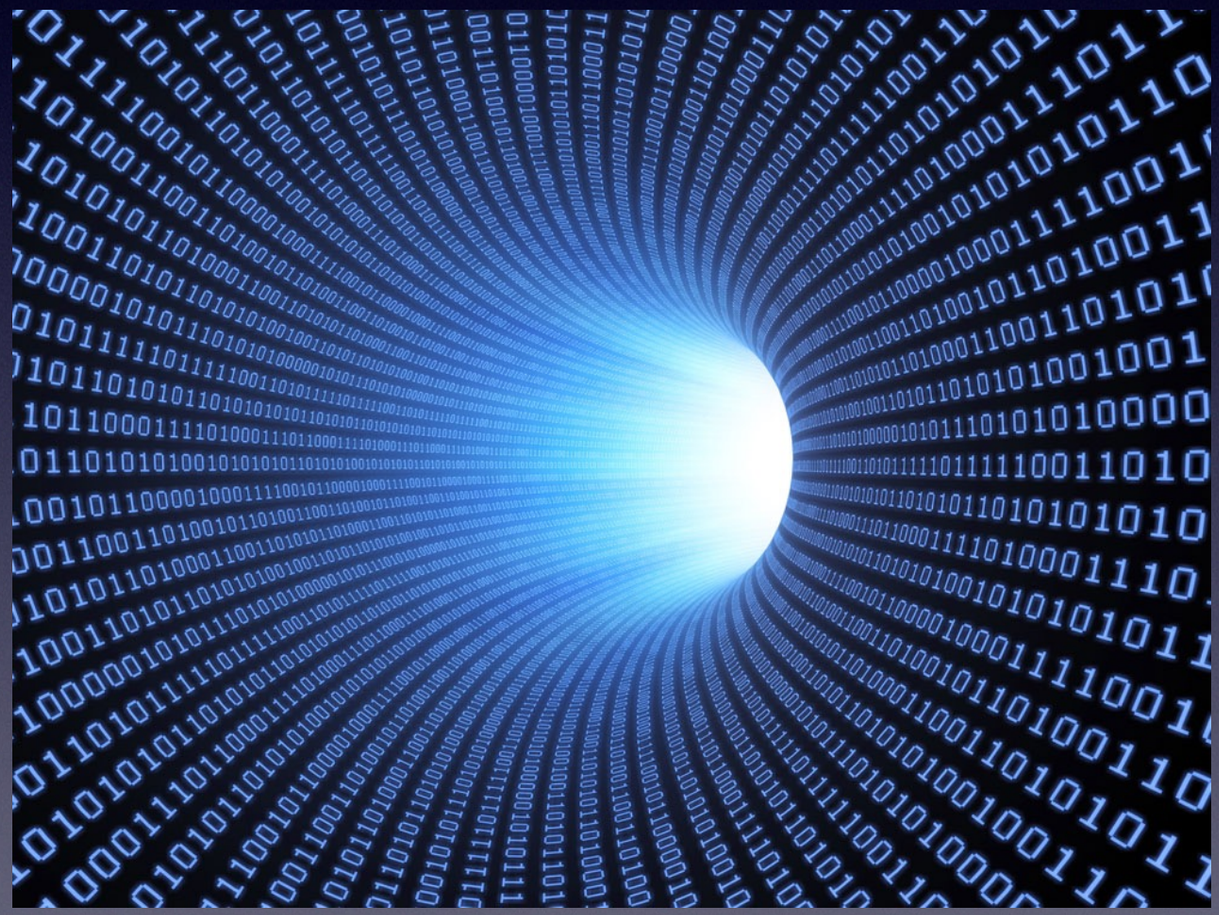
Transmission Control Protocol

产品开发-运维

陆培庆 Marco Lu

小调查

- 听说过OSI模型或者TCP/IP协议?
- 学过TCP/IP协议簇?
- 写过网络编程相关代码?



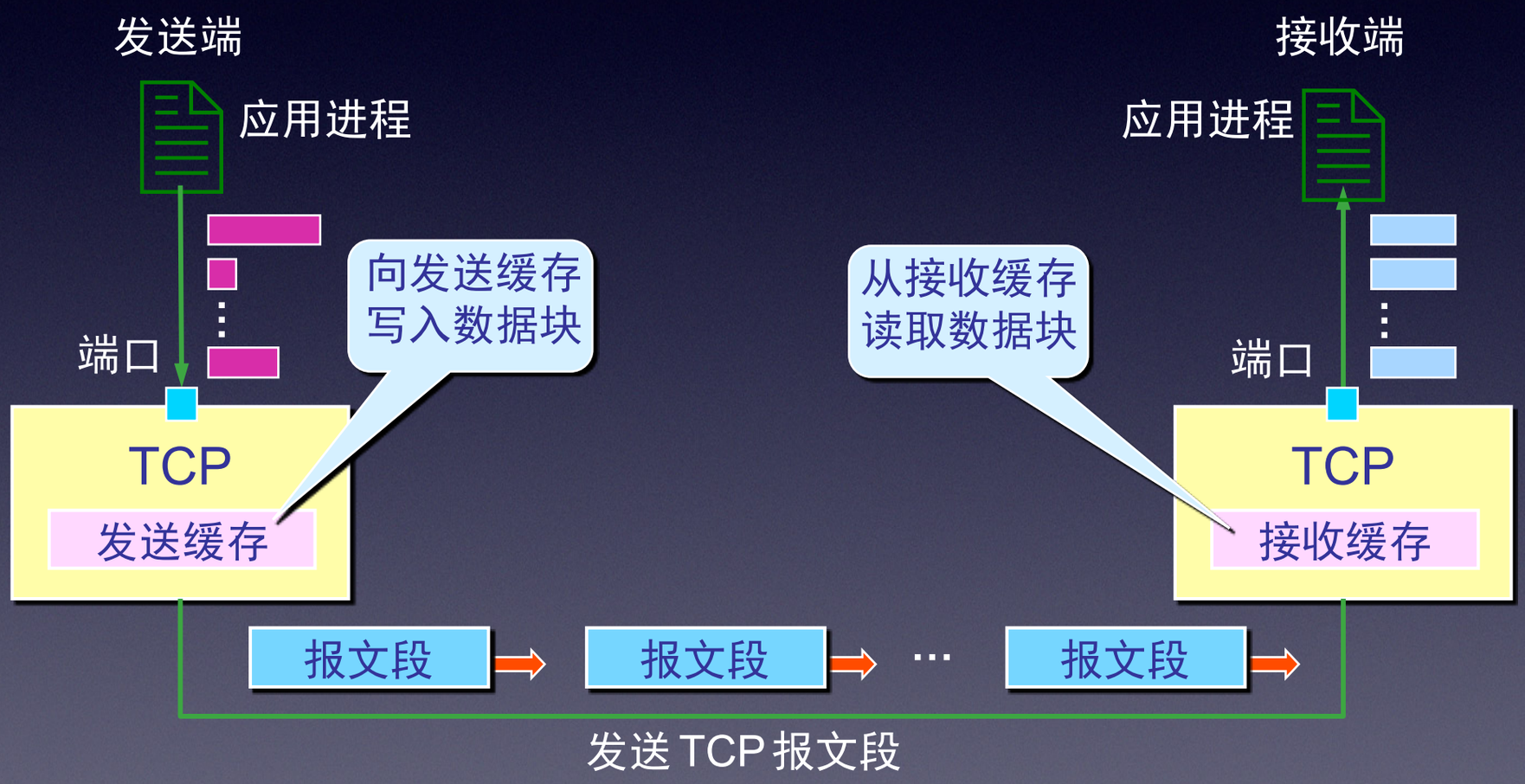
Agenda

- TCP协议概述
- TCP传输控制实现方法
- 三次握手
- 正常的连接建立和关闭
- 流量控制和拥塞控制
- 重传机制

TCP概述

- 为应用进程提供可靠的、端到端的、面向连接的字节流通信的协议
- 利用网络层IP协议提供的不可靠的分组传输服务，解决分组的重传和排序问题
- 由RFC793正式定义（http://tools.ietf.org/rfc/index)
- 为Internet的许多著名应用提供传输服务（HTTP、HTTPS、FTP、TELNET、POP3、SSH、SMTP ……）
- 知名端口号及注册端口号：(http://www.iana.org/assignments/port-numbers）

TCP 概述示图


发送端
应用进程
端口
向发送缓存
写入数据块
TCP
发送缓存
接收端
应用进程
端口
从接收缓存
读取数据块
TCP
接收缓存
报文段
报文段
…
报文段
发送 TCP 报文段

TCP 报文段的首部

首部 | 数 据

源 端 口 16 bits								目 的 端 口 16 bits
序 号 32 bits								
确 认 号 32 bits								
数据偏移 4 bits	保 留 6 bits	u r g	a c k	p s h	r s t	s y n	f i n	窗 口 16 bits
校 验 和 16 bits								紧 急 指 针 16 bits
选 项（长度可变）								

TCP 报文段 | TCP 首部 | TCP 数据部分

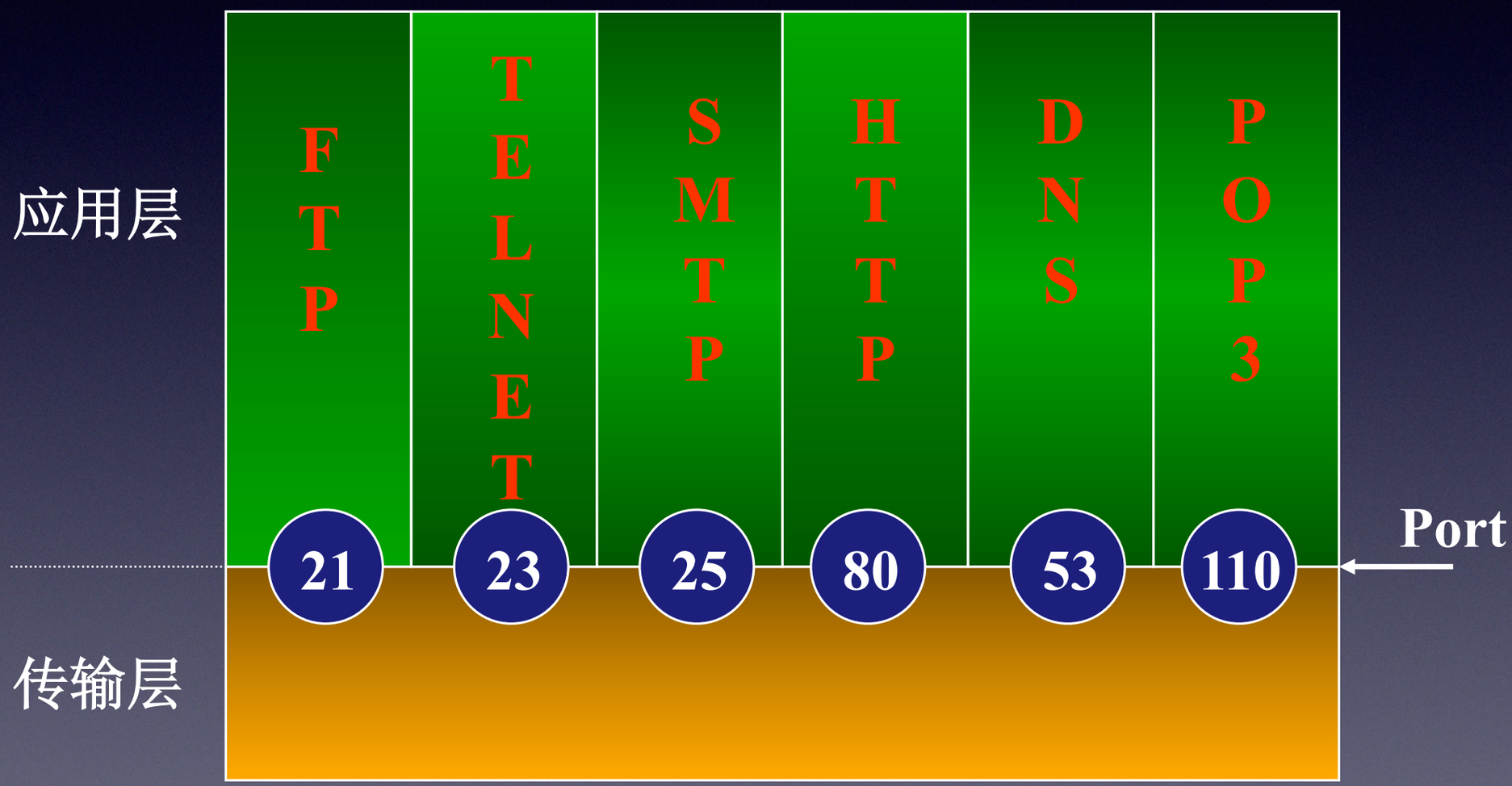
首部各字段及作用

源端口和目的端口字段——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都要通过端口才能实现。

序号字段——占 4 字节。TCP 连接中传送的数据流中的每一个字节都编上一个序号。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。

确认号字段——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。

TCP 保留端口举例



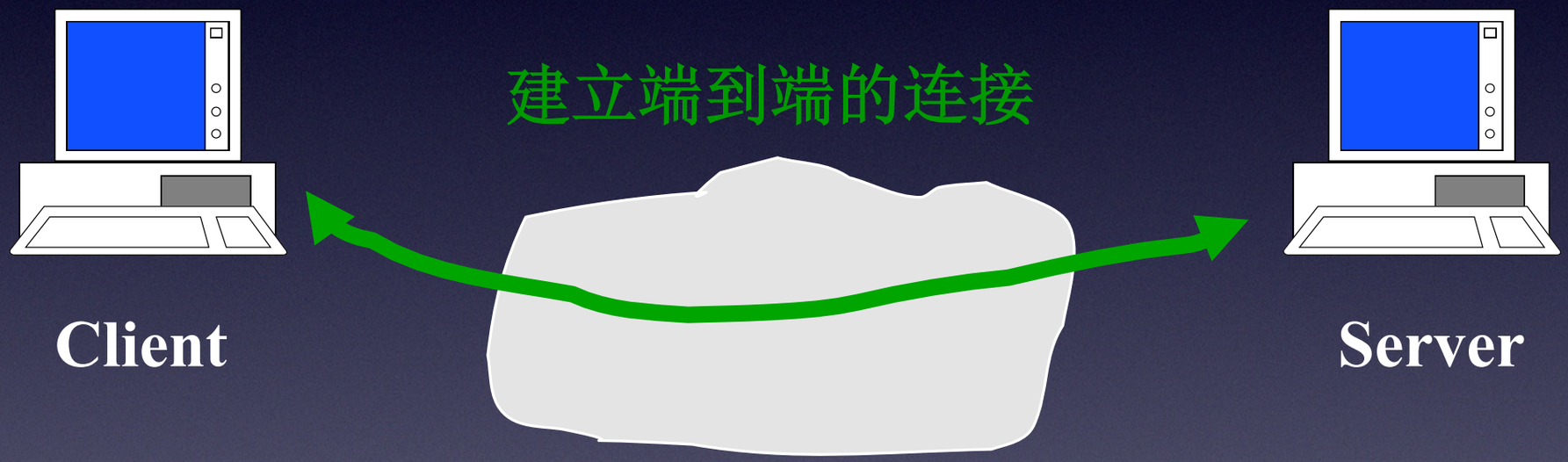
FTP 应用连接端口举例

应用：**ftp 114.80.235.7**

主机A

主机B

建立端到端的连接



Client

Server

(13.13.13.13: 1234)

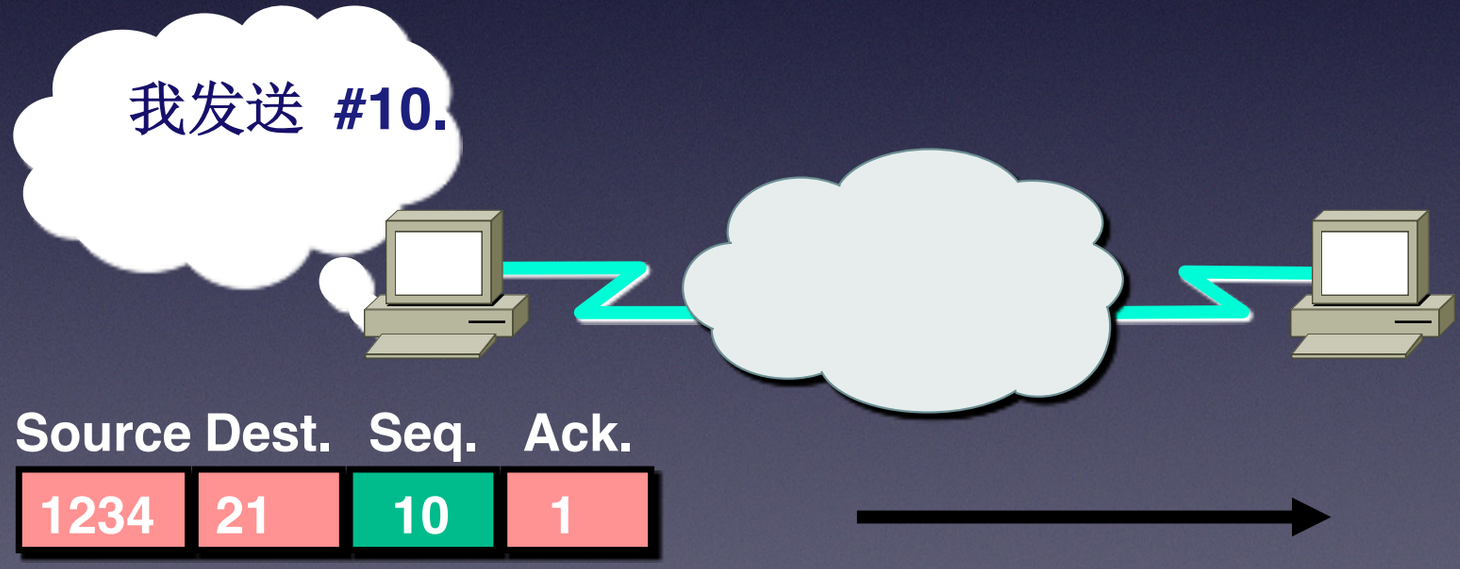
(114.80.235.7: 21)

临时端口

众所周知端口

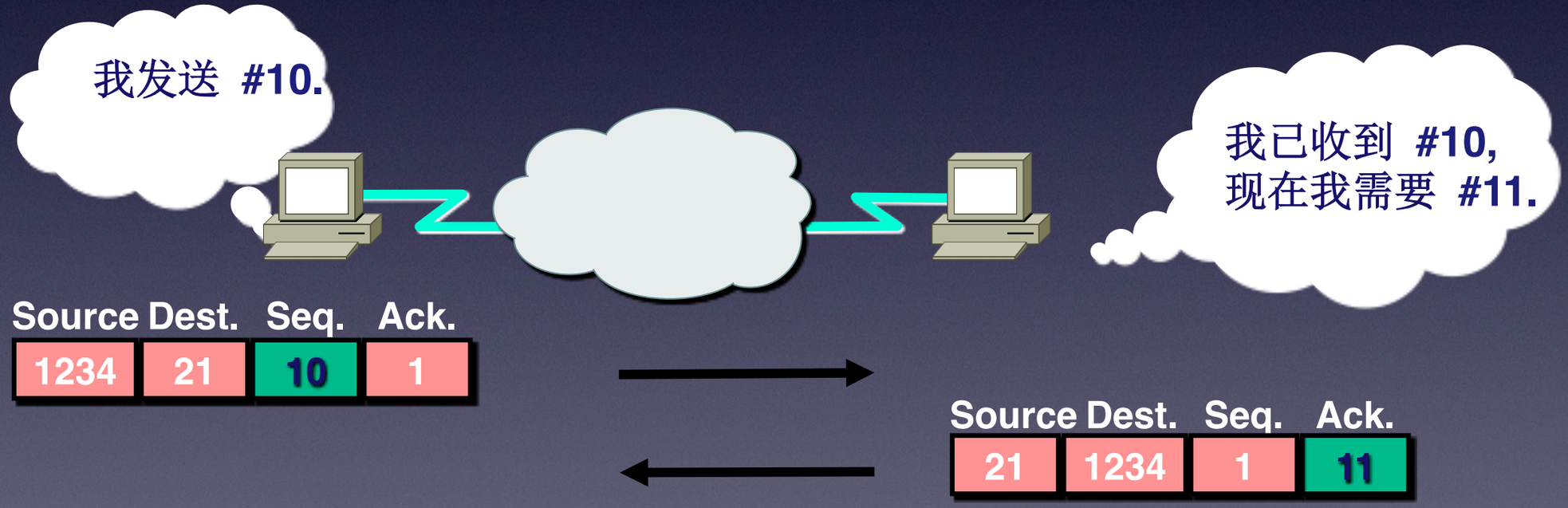
TCP 序号和确认号

源端口	目标端口	顺序号 #	确认号 #	...



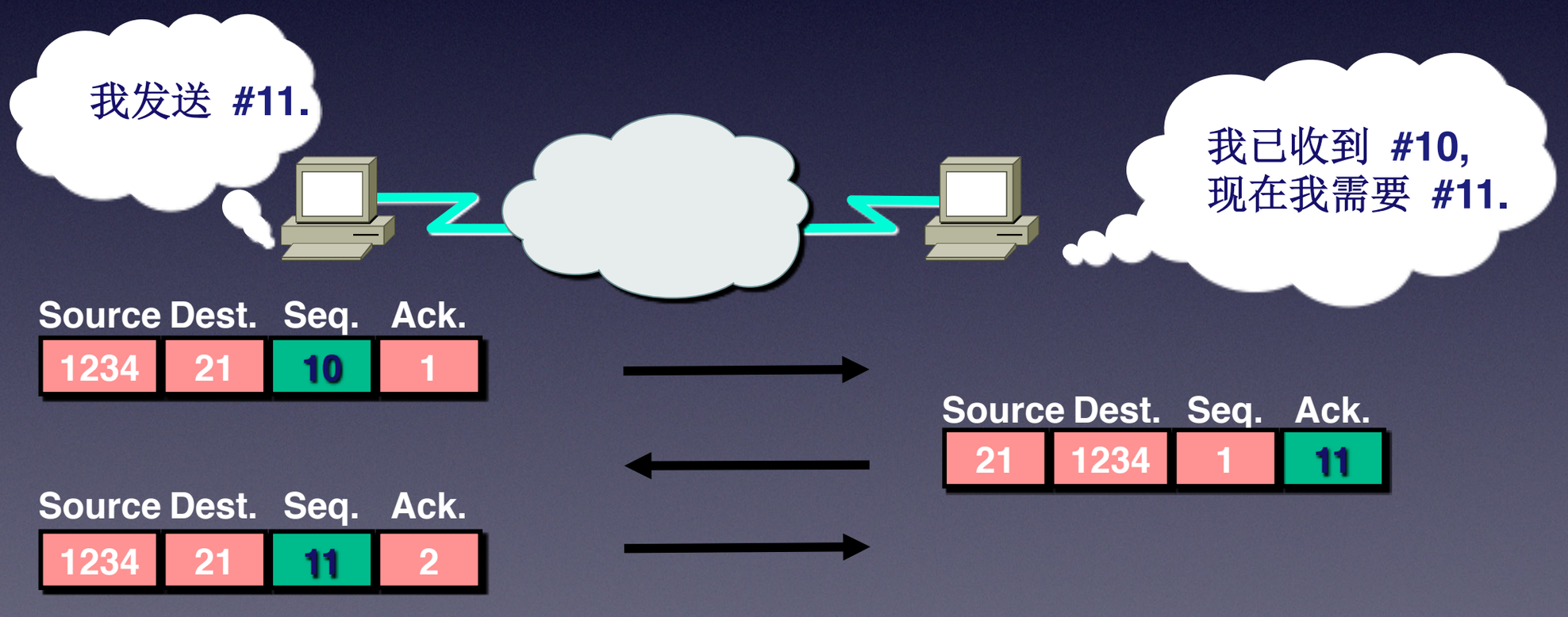
TCP 序号和确认号


源端口
目标端口
顺序号
#
确认号
#
...


我发送 #10.
我已收到 #10,
现在我需要 #11.
Source Dest. Seq. Ack.
1234 21 10 1
Source Dest. Seq. Ack.
21 1234 1 11

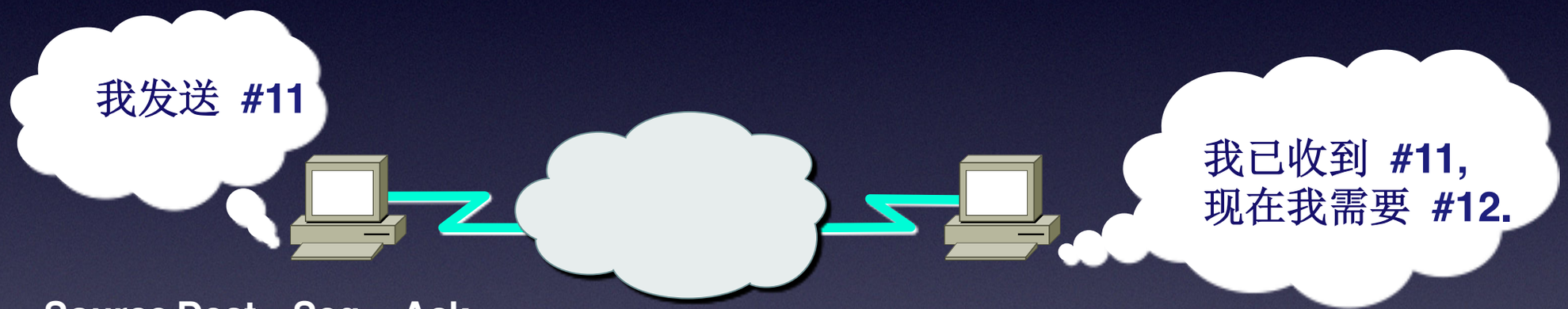
TCP 序号和确认号

源端口	目标端口	顺序号 #	确认号 #	…



TCP 序号和确认号

源端口	目标端口	顺序号 #	确认号 #	…



Source	Dest.	Seq.	Ack.
1234	21	10	1

Source	Dest.	Seq.	Ack.
21	1234	1	11

Source	Dest.	Seq.	Ack.
1234	21	11	2

Source	Dest.	Seq.	Ack.
21	1234	2	12

TCP的数据编号与确认

- TCP协议是面向字节的。将要传送的报文看成是字节组成的数据流，并使每一个字节对应一个序号

- 在连接建立时，双方商定初始序号。TCP每次发送的报文段的首部中的序号字段的数值表示该报文段中的数据部分的第一个字节的序号。

- TCP的确认是对接收到的数据的最高序号表示确认。接收端返回的确认号是已收到的数据的最高序号加1.因此确认号表示接收端期望下次收到的数据中第一个数据字节的序号。

TCP控制报文段发送的机制

- TCP维持一个变量，等于最大报文段长度MSS。发送缓冲区的数据达到MSS字节，就组成一个报文段发送出去。

- 利用TCP支持的推送（PUSH）操作，由发送端的应用进程指明要求发送的报文段。

- 发送端的计时器时间到了，就把当前已有的缓冲数据装入报文段发送出去。

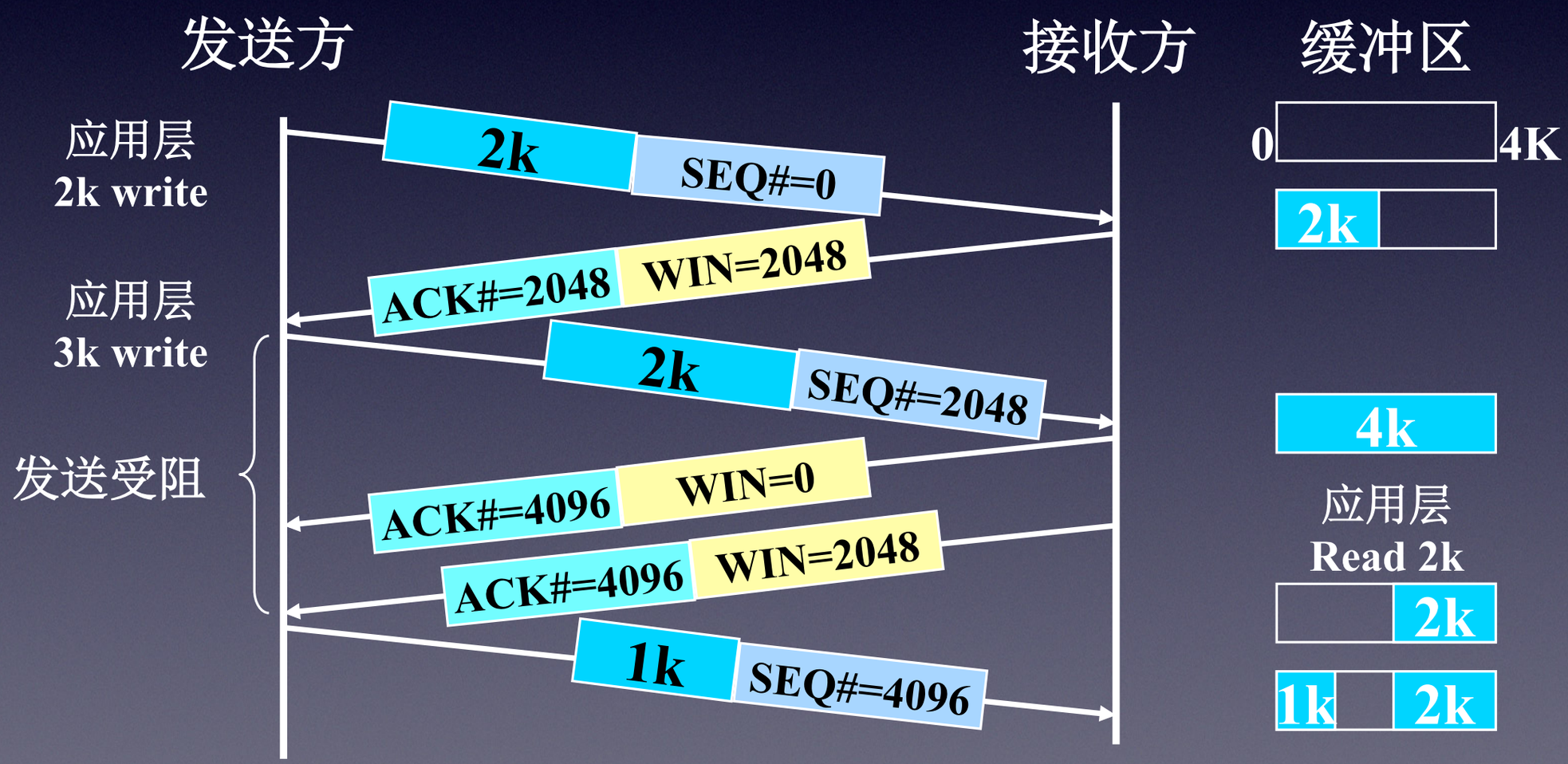
TCP的传输连接管理

- 传输连接三个阶段：连接建立、数据传输和连接释放。传输连接的管理就是使传输连接的建立和释放都能正常的进行。

- 连接建立过程中要解决以下三个问题：
  - 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（如最大报文段长度，最大窗口大小等）
  - 能够对传输实体资源（如缓存大小，连接表中的项目）进行分配

TCP传输控制的实现


头部
源端口
1028
宿端口
21
序列号
SEQ#
确认号
ACK#
...
窗口
WIN
...


发送方
接收方
缓冲区
应用层
2k write
2k
SEQ#=0
ACK#=2048
WIN=2048
应用层
3k write
2k
SEQ#=2048
发送受阻
ACK#=4096
WIN=0
ACK#=4096
WIN=2048
1k
SEQ#=4096
0
4K
2k
4k
应用层
Read 2k
2k
1k
2k

三次握手建立TCP连接

主机 A

主动打开

主机 B

被动打开

连接请求

SYN, SEQ = x

SYN, ACK, SEQ = y, ACK= x + 1

确认

ACK, SEQ = x + 1, ACK = y + 1

确认

连接成功！

建立TCP连接

- A 的 TCP 实体向 B的 TCP 实体发出连接请求报文段，其首部中的同步比特 SYN 应置为 1，并选择一个随机序号 x，表明传送数据时的第一个数据字节的序号是 x。
- B 的 TCP 收到连接请求报文段后，如同意，则发回确认。
- B 在确认报文段中应将 SYN 置为 1，ACK置为1，其确认号应为 x+1，同时也为自己选择随机序号 y。
- A 收到此报文段后，向 B 给出确认， ACK置为1，其确认号应为 y+1。
- A 的 TCP实体通知上层应用进程，连接已经建立。
- 当运行服务器进程的主机 B 的 TCP实体收到主机 A 的确认后，也通知其上层应用进程，连接已经建立。

三次握手

DDOS攻击


SERVER
ONLINE
SYN
SYN
SYN
SYN
SYN
SYN


SERVER
ONLINE
SYN
SYN
SYN
SYN
SYN
SYN


SYN

TCP连接释放的过程

主机 A

主机 B

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。但你如果还发送数据，我仍接收。”

应用进程
释放连接

A 不再发送报文

FIN, SEQ = x

①
通知主机
应用进程

②

ACK, SEQ = y, ACK= x + 1

确认

FIN, ACK, SEQ = y, ACK = x + 1

应用进程
释放连接

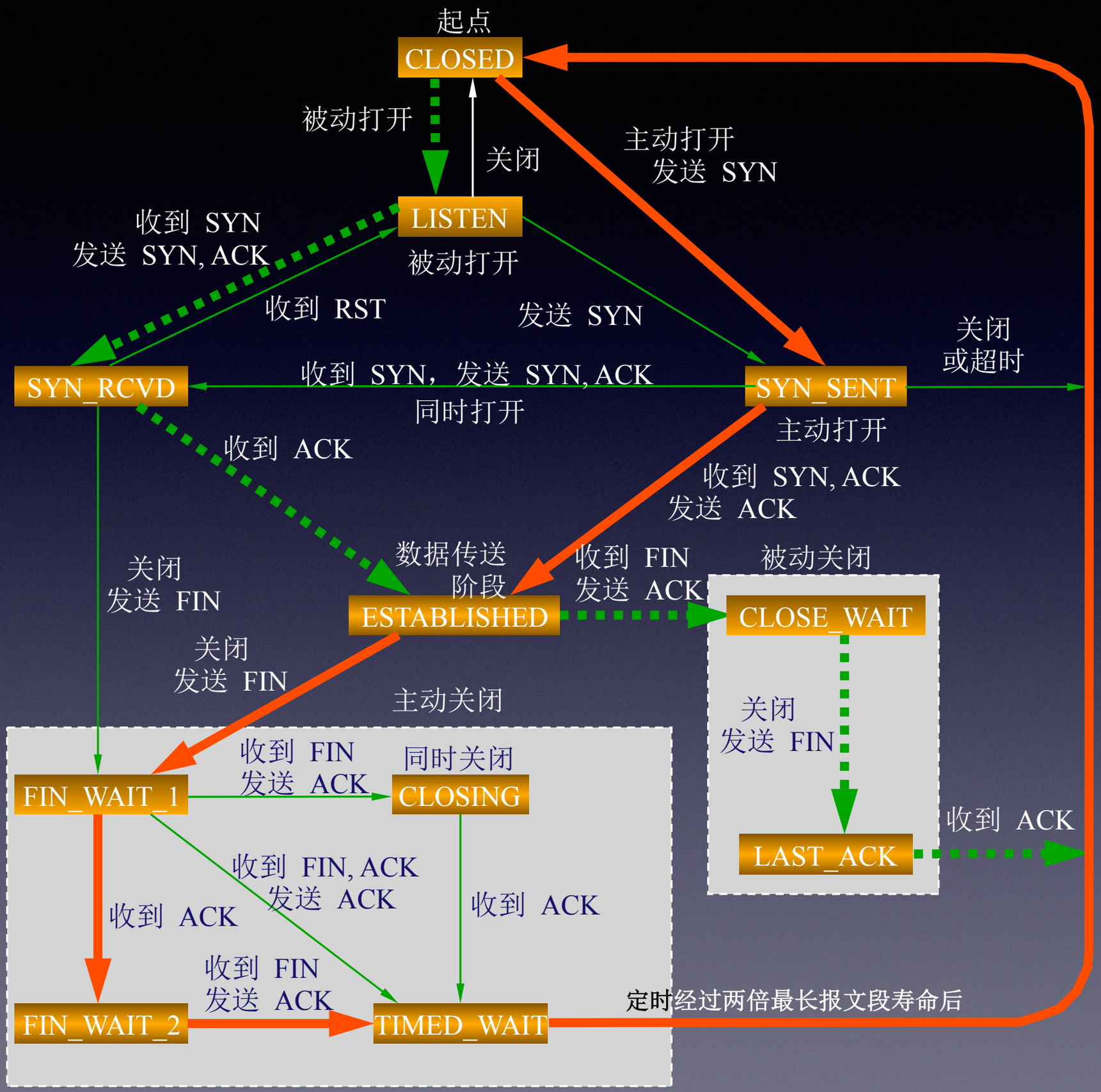
B 不再发送报文

ACK, SEQ = x + 1, ACK = y + 1

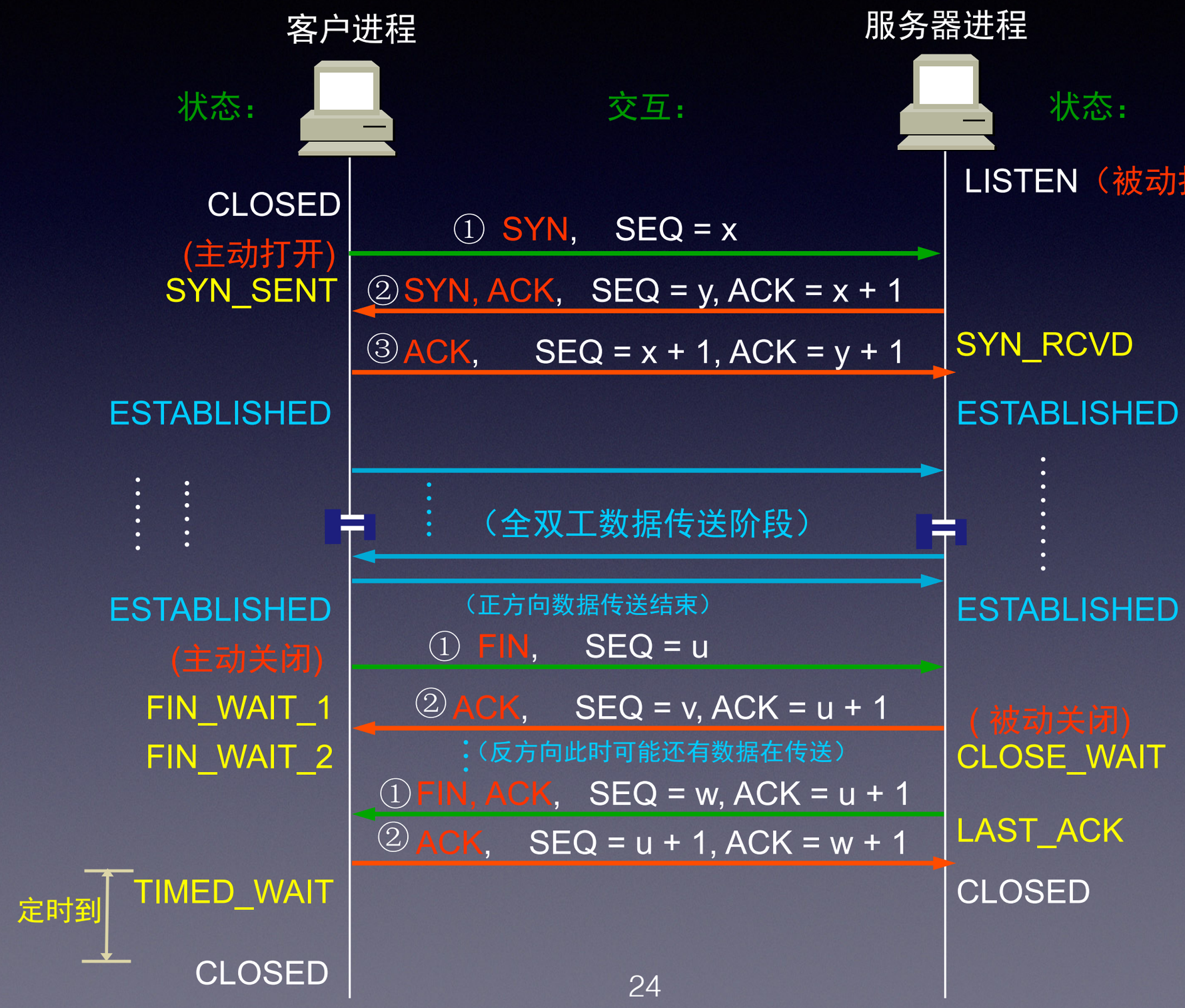
确认

至此，整个连接已经全部释放。

TCP的有限状态机


起点
CLOSED
被动打开
关闭
主动打开
发送 SYN
收到 SYN
发送 SYN, ACK
LISTEN
被动打开
收到 RST
发送 SYN
关闭
或超时
SYN_RCVD
收到 SYN，发送 SYN, ACK
同时打开
SYN_SENT
主动打开
收到 ACK
收到 SYN, ACK
发送 ACK
关闭
发送 FIN
数据传送
阶段
收到 FIN
发送 ACK
被动关闭
ESTABLISHED
CLOSE_WAIT
关闭
发送 FIN
关闭
发送 FIN
主动关闭
收到 FIN
发送 ACK
同时关闭
FIN_WAIT_1
CLOSING
收到 FIN, ACK
发送 ACK
收到 ACK
收到 ACK
收到 FIN
发送 ACK
收到 ACK
FIN_WAIT_2
TIMED_WAIT
定时经过两倍最长报文段寿命后

TCP的正常连接建立和关闭



TCP的有限状态机

- 为了管理因特网，在网络管理中心设有管理信息库 MIB (Management Information Base)。
- 管理信息库存放着各主机的 TCP 连接表。
- TCP 连接表对每个连接都登记了其连接信息。除本地和远地的 IP 地址和端口号外，还要记录每一个连接所处的状态。

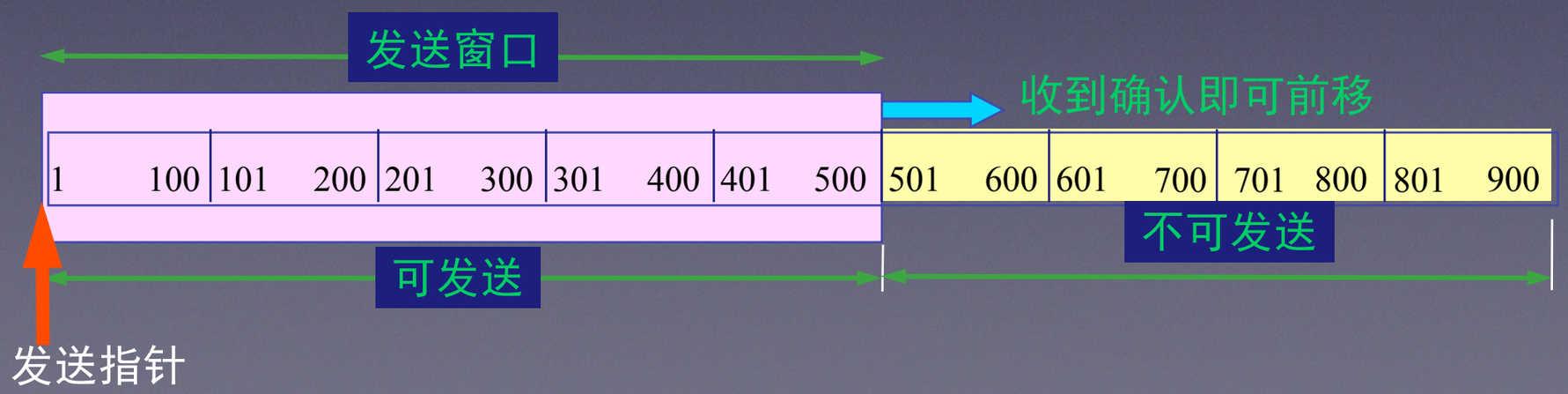
	连接状态	本地 IP 地址	本地端口	远地 IP 地址	远地端口
连接1					
连接2					
⋮					
连接n					

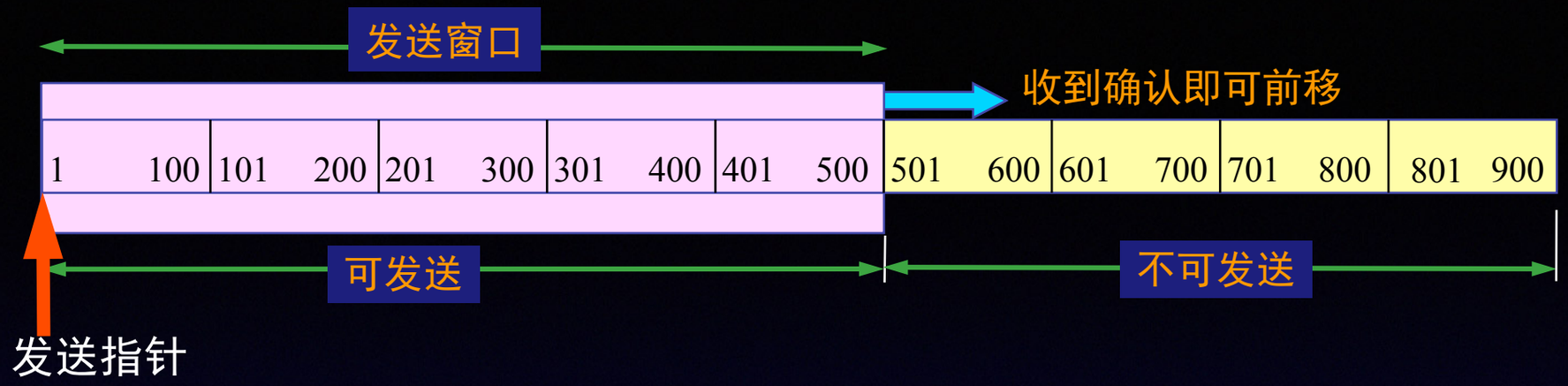
TCP的流量控制与拥塞控制

滑动窗口的概念

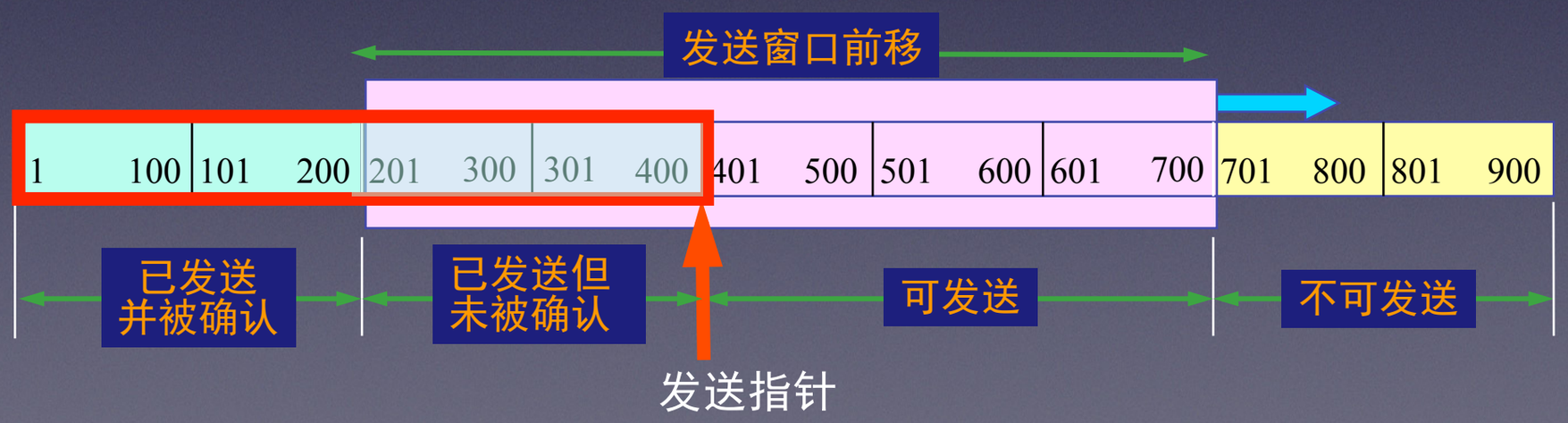
- TCP采用大小可变的滑动窗口进行流量控制。窗口大小的单位是字节。
- 在TCP报文段首部的窗口字段写入的数值就是当前给对方设置的发送窗口数值的上限。
- 发送窗口在连接建立时由双方商定。但在通信的过程中，接收端可根据自己的资源情况，随时动态地调整对方的发送窗口上限值(可增大或减小)。

- 设发送端要发送 900 字节长的数据，划分为 9 个 100 字节长的报文段，并设发送窗口为 500 字节。
- 发送窗口的左边沿对应已发送数据中被确认的最高序号 + 1，其右边沿对应左边沿的序号加上发送窗口的大小。
- 发送 TCP 要维护一个指针。每发送一个报文段，发送指针就向前移动一个报文段的距离。
- 发送TCP每收到对方对一个报文段的确认，便释放该报文段所占的缓冲，将发送窗口左沿向右移动一个报文段的距离。若发送窗口大小不变，则窗口右沿也向右移，即发送窗口向右滑动。
- 当接收方接收缓冲区变小、向发送方发出的TCP报文首部的“窗口”字段值变小时，使发送窗口变小，发送窗口会收缩。

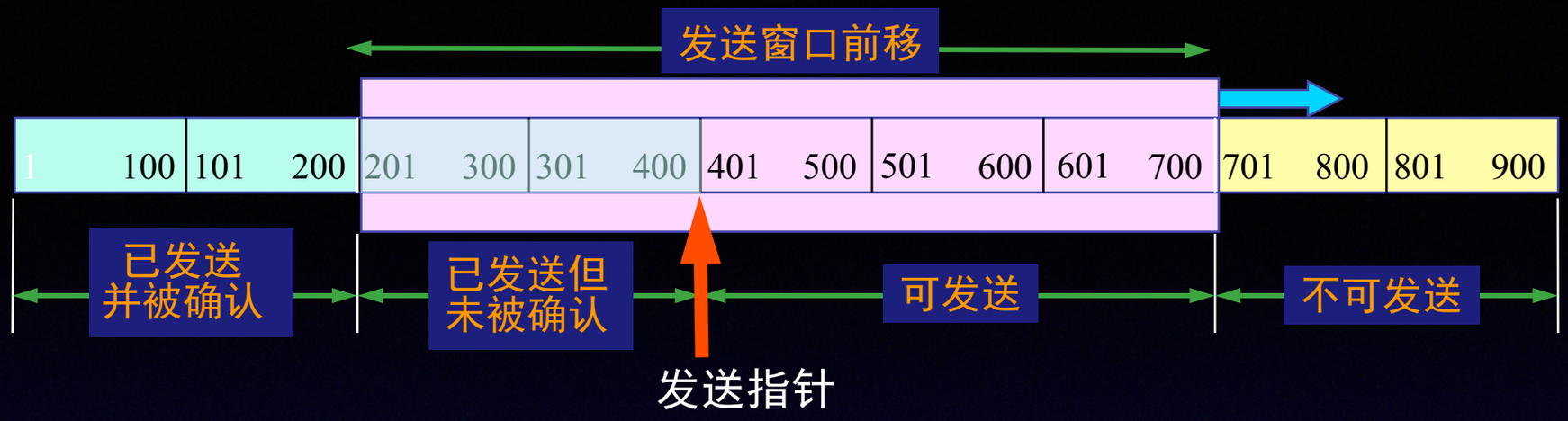




- 发送端发送了 **400** 字节的数据，但只收到对前 **200** 字节数据的确认，同时窗口大小不变，因此发送窗口向右平移。

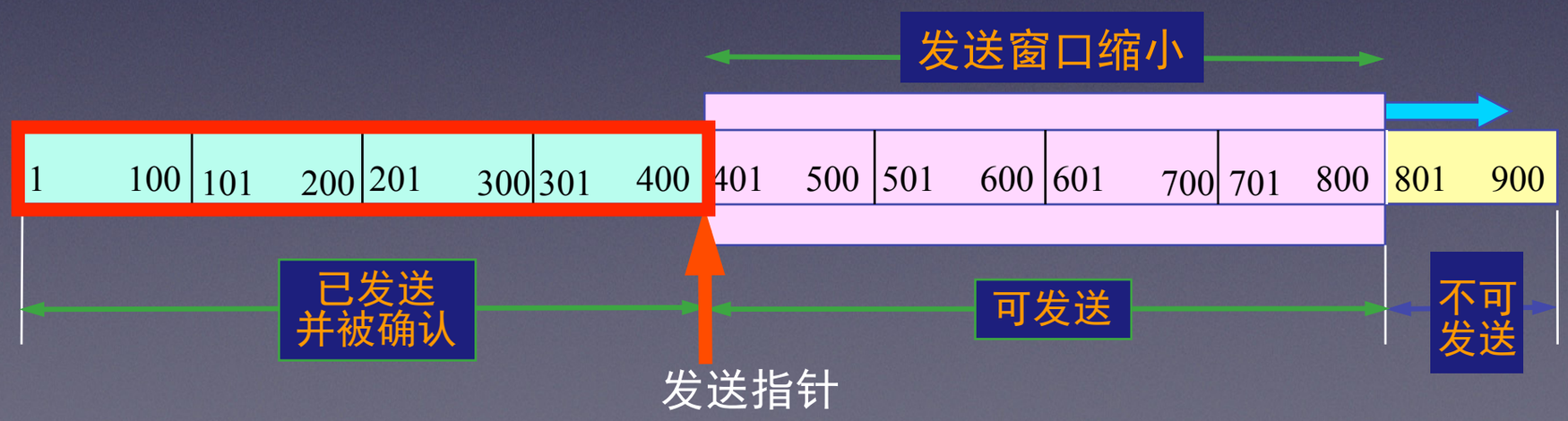


- 现在发送端还可发送 **300** 字节。



- 发送端收到了对方对前 400 字节数据的确认，但对方通知发送 端必须把窗口减小到 400 字节。

- 现在发送端最多还可发送 400 字节的数据。



利用可变窗口大小进行流量控制

设双方确定的窗口值为400

TCP拥塞控制

- 拥塞原因：
  - 加载到网络的负载大于网络自身的处理能力，若再大量向网络中输入负载，网络性能会明显变坏，极端情况下便会死锁。出现网络拥塞的条件：

    ∑(对网络中资源的需求) > 可用资源

- 解决办法：
  - 开环控制：设计网络时事先将发生网络拥塞的因素考虑周到，力求网络在工作时不产生拥塞。
  - 闭环控制：基于反馈环路概念。监测拥塞在何时何地发生，将消息传到可采取行动的地方，继而调整网络系统的运行以解决出现的问题。
  - 例如：TCP发送方迟迟不能收到对方的确认信息，便认为网络中发生了拥塞，于是减少注入网络中的信包数。

- 流量控制----往往指接收方对发送方通信量的控制，是个端到端的问题。流量控制要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。
- 拥塞控制----就是防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。

- 发送方控制发送流量必须同时考虑接收端的存储容量和网络的传输能力，可以如下形象地说明：

接收方的处理能力不足：
--接收容器小
网络不够通畅：
--传输管道细

接收方处理能力不足

发送方

调整传输速率

传输网络

小容量接收器

接收能力

网络不够畅通

发送方

调整传输速率

网络能力

传输网络

内部拥塞

大容量接收器

rwnd和cwnd

- 接收端窗口 rwnd (receiver window) 这是接收端根据其目前的接收缓存大小所许诺的最新的窗口值，是来自接收端的流量控制。接收端将此窗口值放在 TCP 报文的首部中的窗口字段，传送给发送端。
- 拥塞窗口 cwnd (congestion window) 是发送端根据自己估计的网络拥塞程度而设置的窗口值，是来自发送端的流量控制。

拥塞的解决办法

- 发送方保持两个窗口
  - 对方的接收窗口 rwnd
  - 拥塞窗口 cwnd
- 在一个实际网络中，发送方控制发送流量必须同时考虑接收端的存储容量和网络的传输能力，发送流量应该取接收端和通信子网所能允许的流量值中的较小值。因此，取两者的最小值为发送窗口的上限值，即:

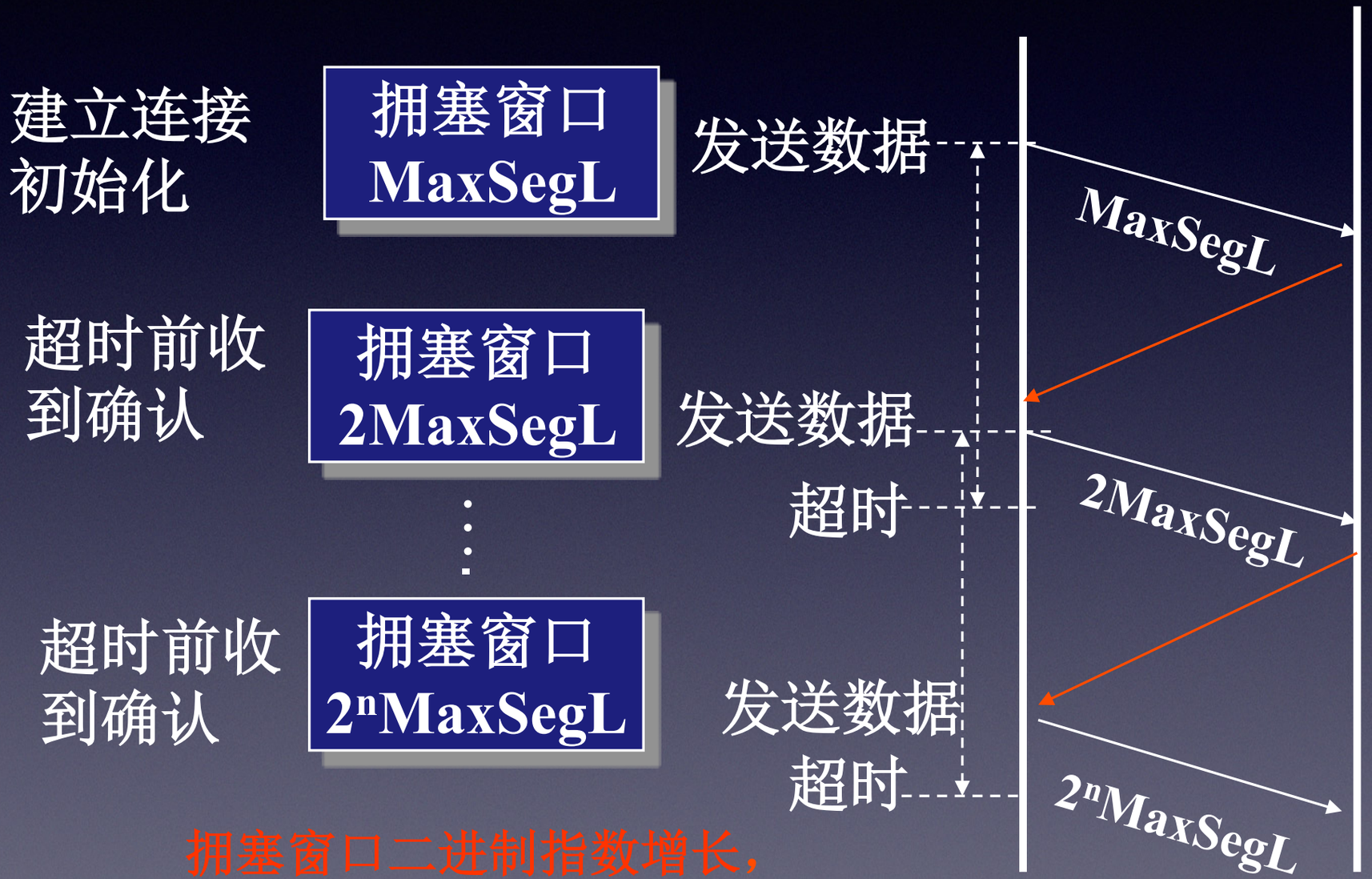
发送窗口的上限值 = Min [rwnd, cwnd]　　(7-1)

- 如何确定拥塞窗口的大小?
  - 慢开始算法
  - 拥塞控制算法

慢开始算法的原理

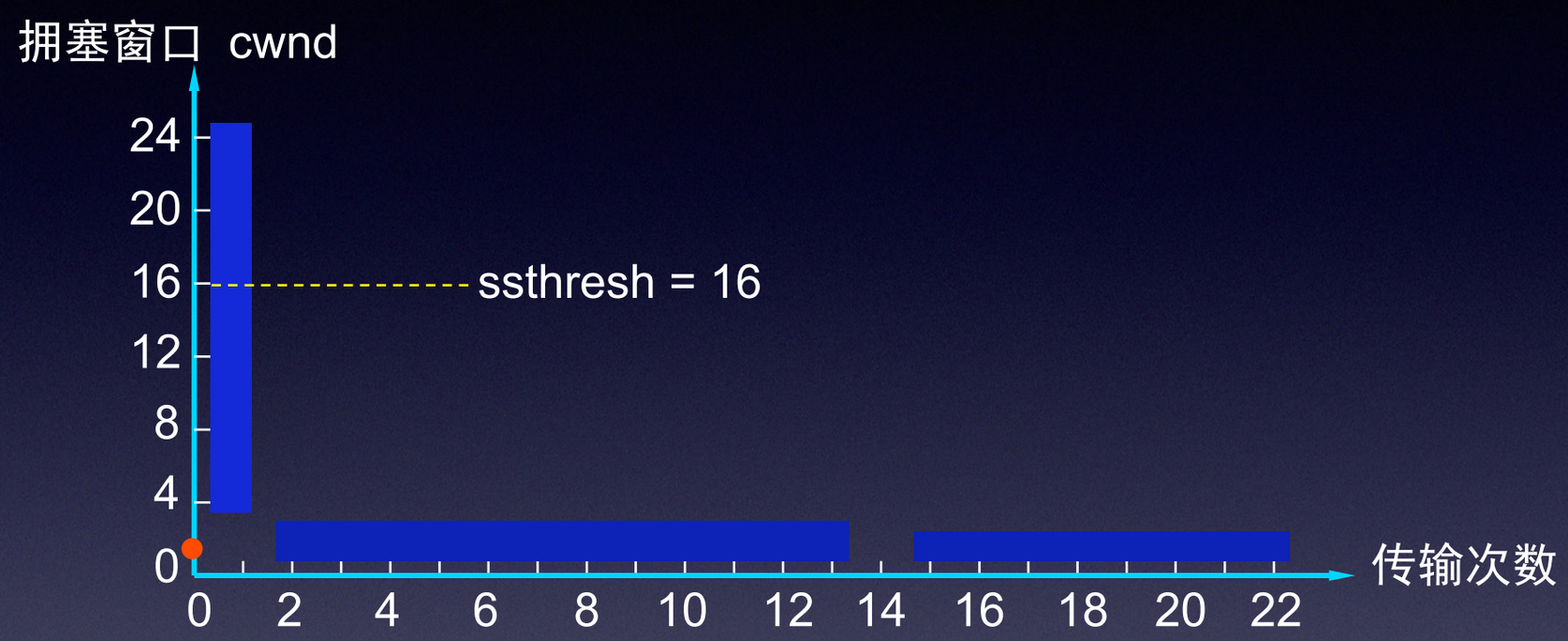
- 在主机刚刚开始发送报文段时可先将拥塞窗口 cwnd 设置为一个最大报文段 MSS 的数值。
- 在每收到一个对新的报文段的确认后，将拥塞窗口增加至多一个 MSS 的数值。
- 用这样的方法逐步增大发送端的拥塞窗口 cwnd，可以使分组注入到网络的速率更加合理。

慢开始算法



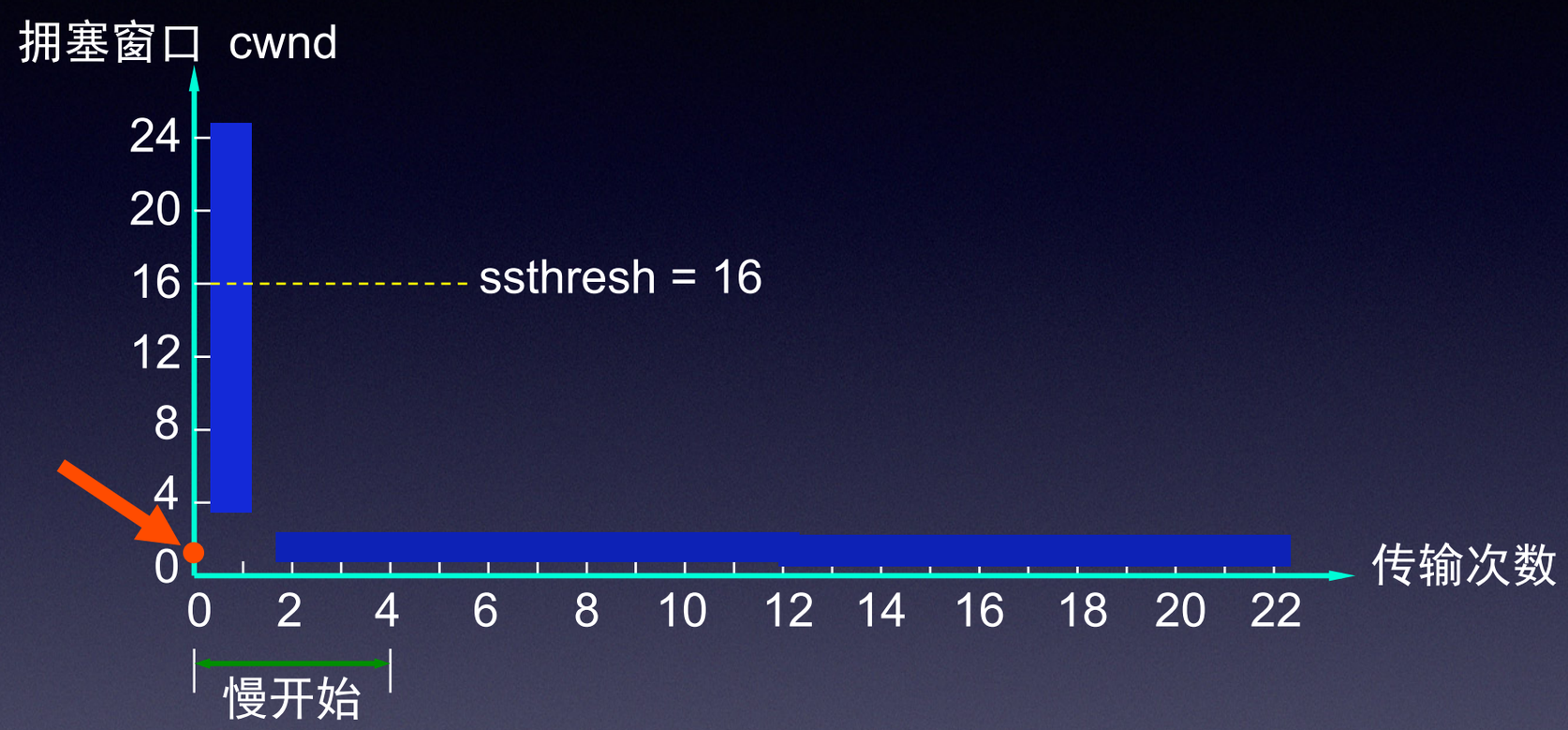
拥塞窗口二进制指数增长，
直至发送数据后超时无确认

慢开始算法举例



- 当 TCP 连接进行初始化时，将拥塞窗口置为 1。（图中的窗口单位不使用字节而使用报文段）
- 设慢开始门限的初始值设置为 16 个报文段，即 **ssthresh = 16**
- 发送端的发送窗口不能超过拥塞窗口 **cwnd** 和接收端窗口 **rwnd** 中的最小值。我们假定接收端窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值

慢开始算法举例

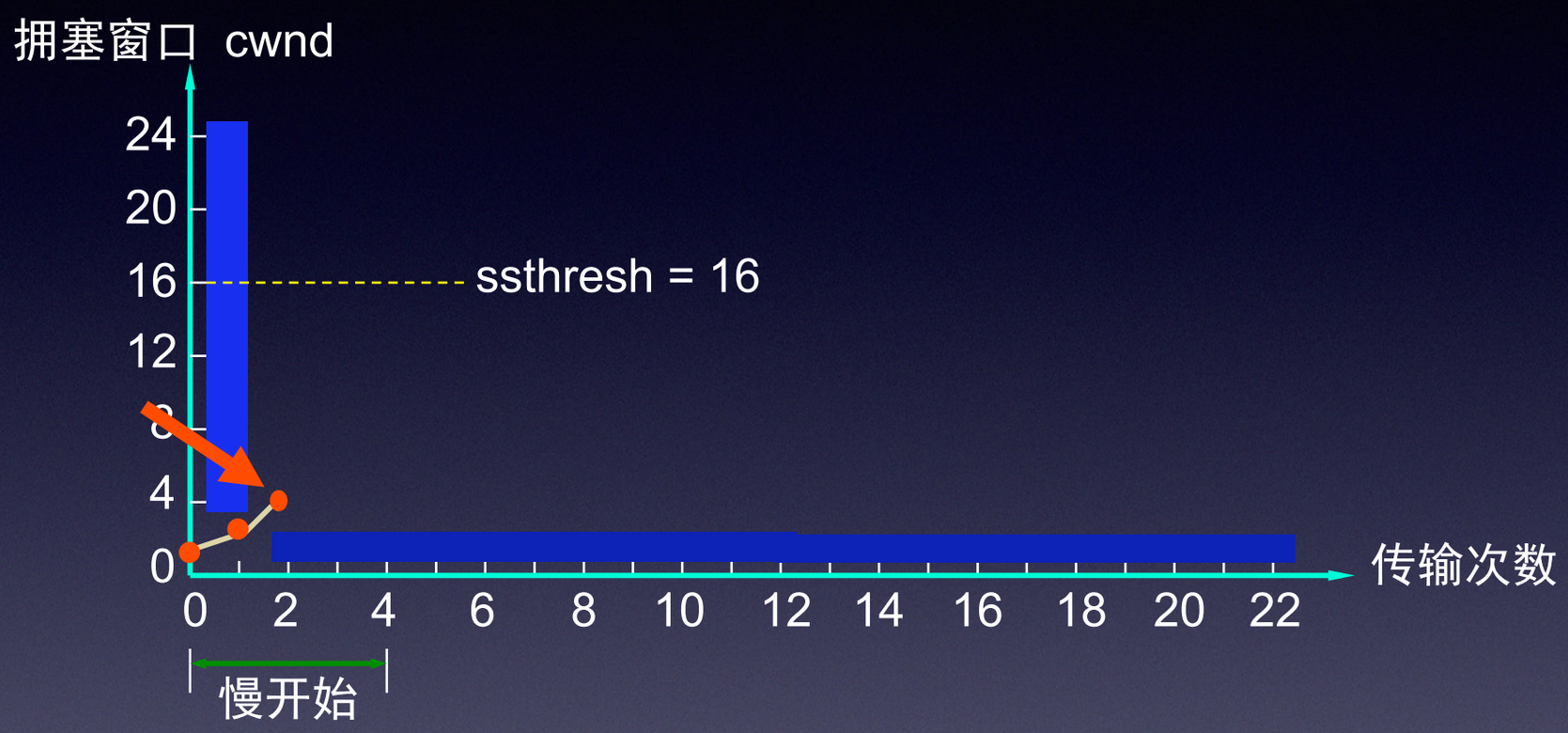


在执行慢开始算法时，拥塞窗口 **cwnd** 的初始值为 **1**，发送第一个报文段 M_0。

慢开始算法举例

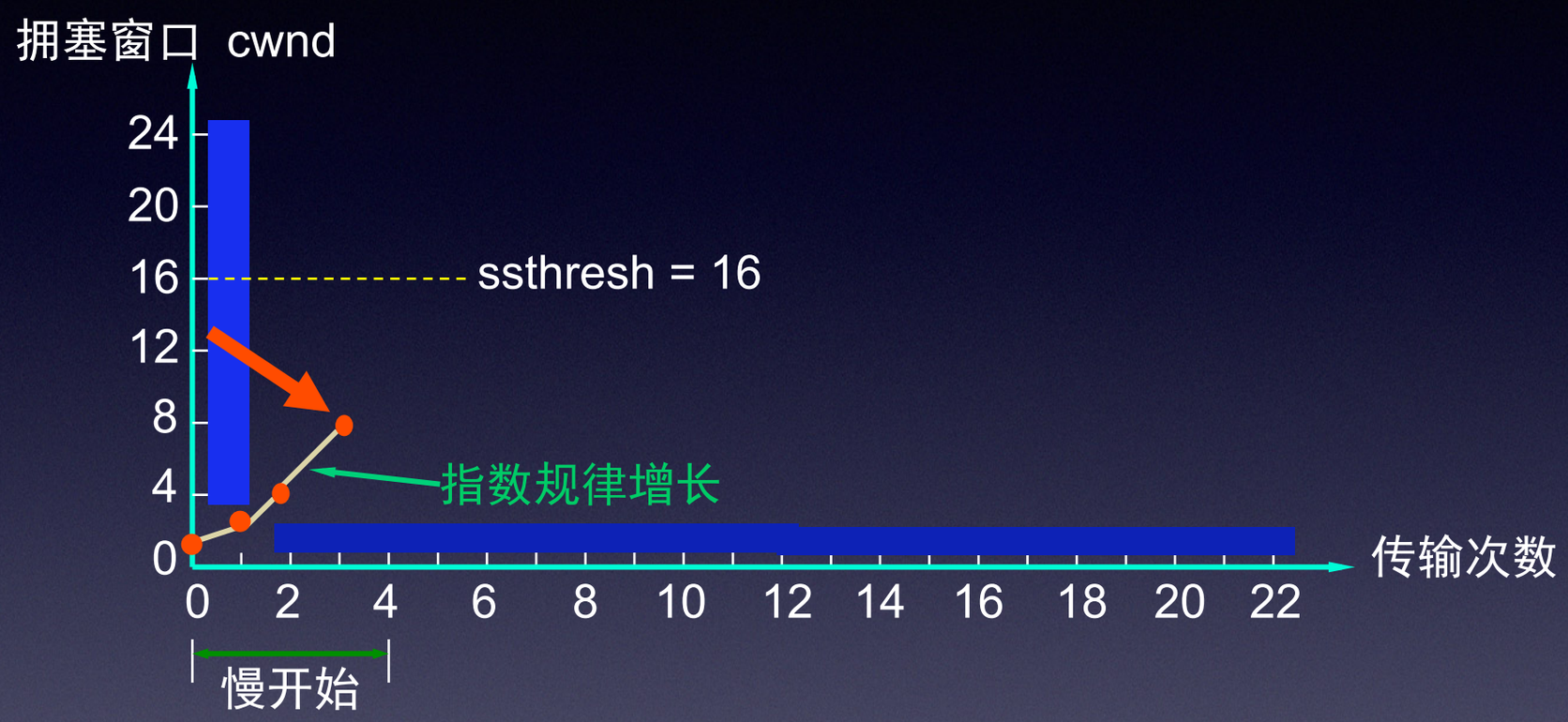
发送端收到 ACK_1（确认 M_0，期望收到 M_1）后，将 cwnd 从 1 增大到 2，于是发送端可以接着发送 M_1 和 M_2 两个报文段。

慢开始算法举例



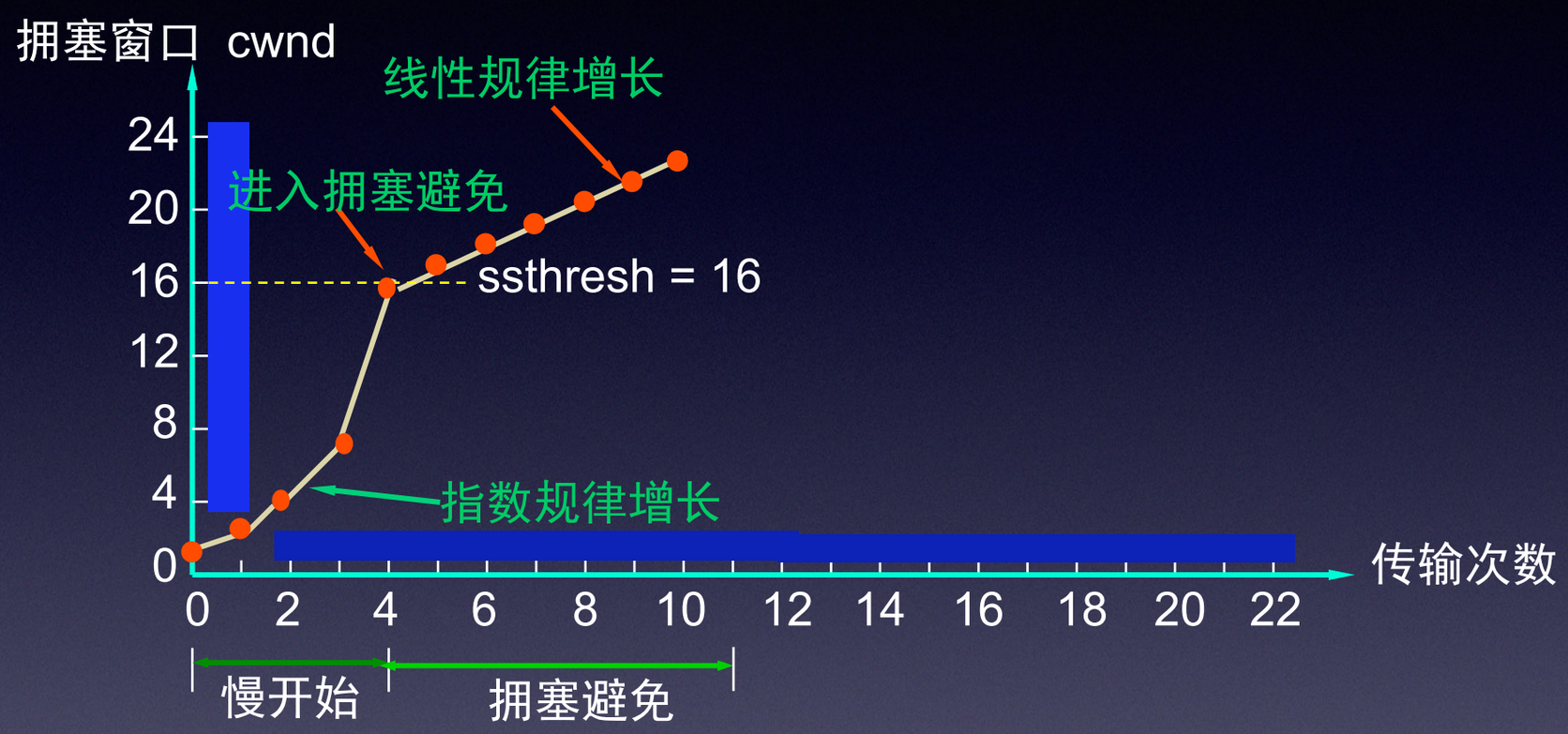
接收端发回 ACK_2 和 ACK_3。发送端每收到一个对新报文段的确认 ACK，就把发送端的拥塞窗口加 1。现在发送端的 cwnd 从 2 增大到 4，并可发送 $M_4 \sim M_6$共 4个报文段。

慢开始算法举例



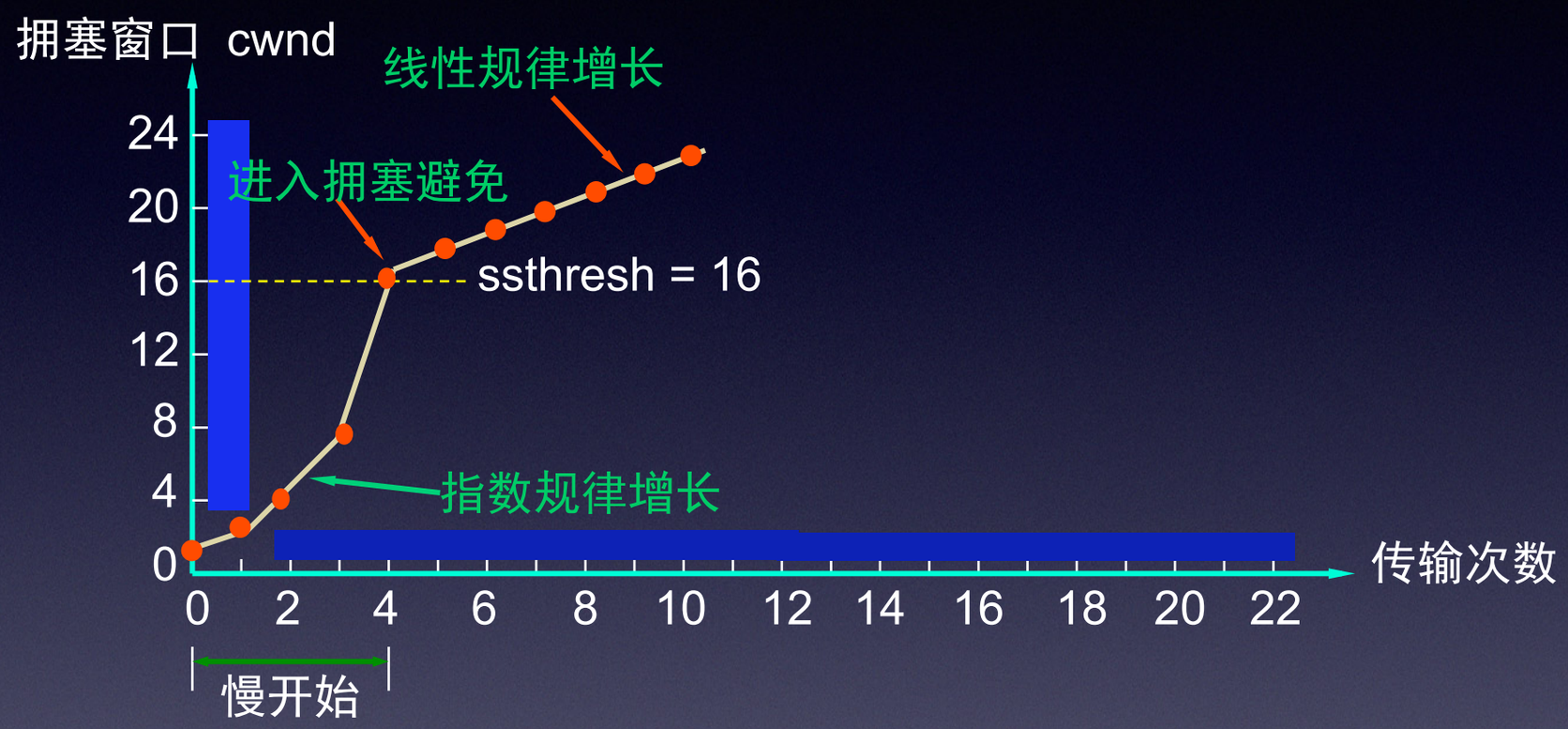
发送端每收到一个对新报文段的确认 **ACK**，就把发送端的拥塞窗口加 **1**，导致每经过一个传输轮次，拥塞窗口就加倍。因此拥塞窗口 **cwnd** 随着传输次数按指数规律增长。

慢开始算法举例



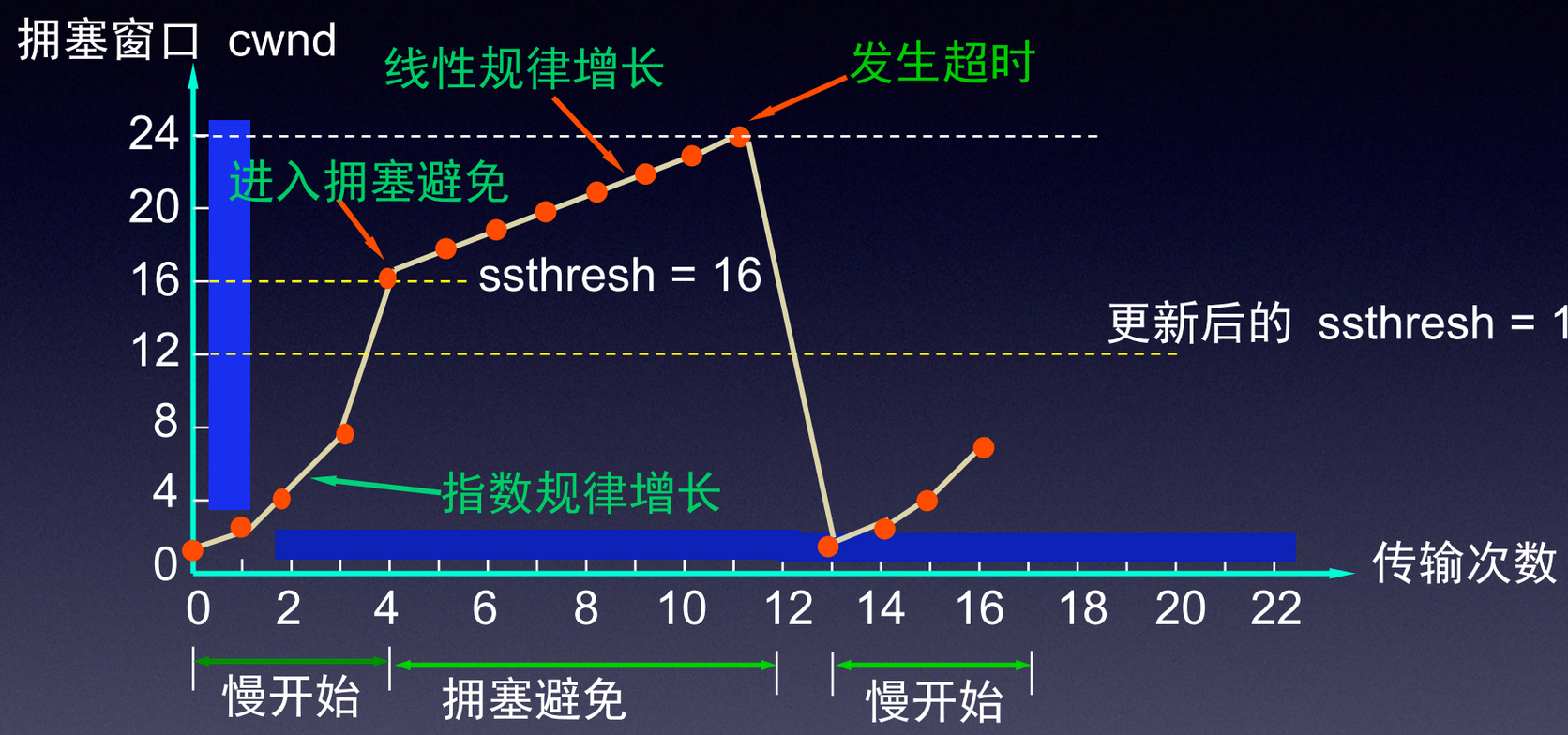
当拥塞窗口 **cwnd** 增长到慢开始门限值 **ssthresh** 时（即当 **cwnd = 16** 时），就改为执行拥塞避免算法，每经一传输轮次拥塞窗口只加1，因此按线性规律增长。

慢开始算法举例



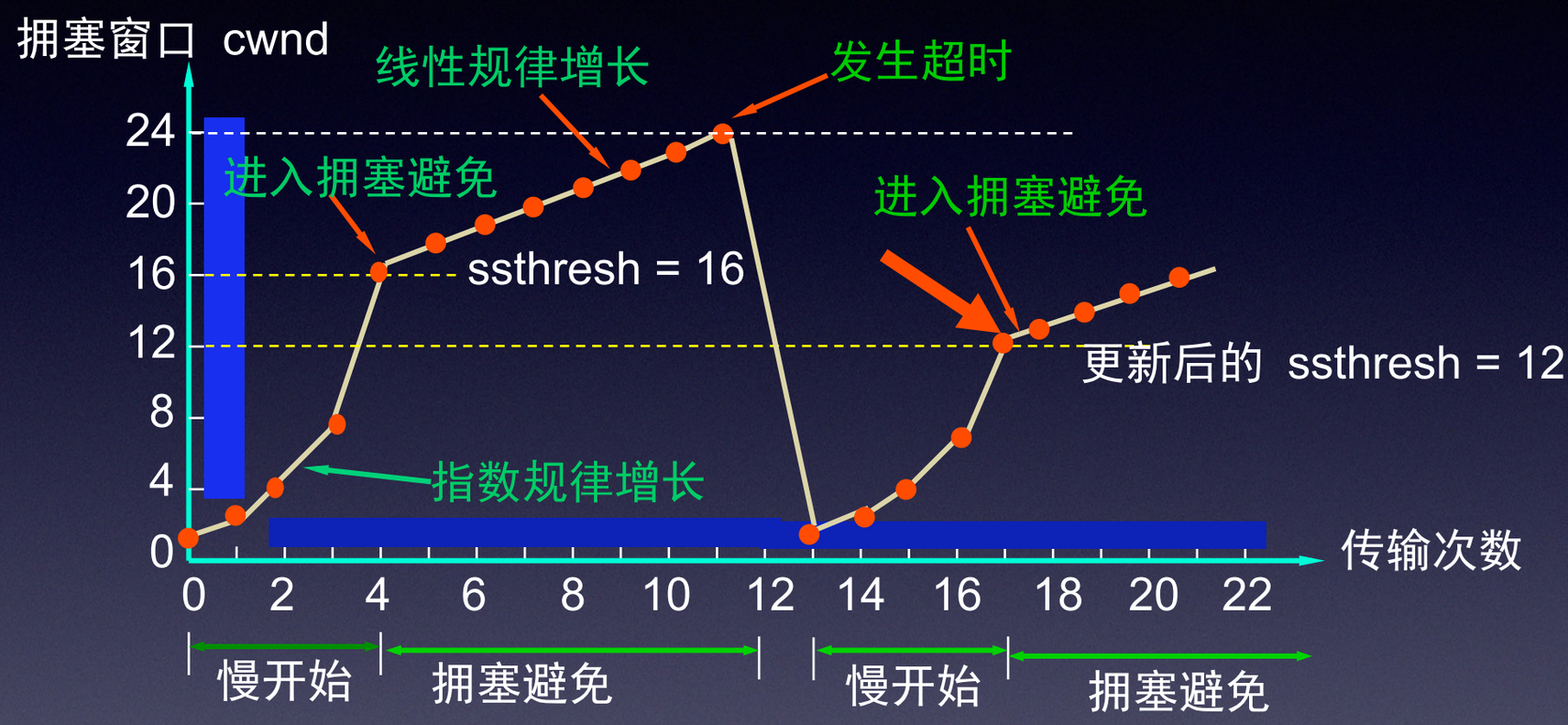
假定拥塞窗口的数值增长到 24 时，网络出现超时（表明网络拥塞了），便执行“乘法减小”策略：

慢开始算法举例



更新后的 ssthresh 值变为 12（即发送窗口数值 24 的一半），拥塞窗口再重新设置为 1，并重新执行慢开始算法。

慢开始算法举例



当 **cwnd = 12** 时又改为执行拥塞避免算法，拥塞窗口按按线性规律增长，每经过一个往返时延就增加一个 **MSS** 的大小。

乘法减小
Multiplicative Decrease

- “乘法减小“是指不论在慢开始阶段还是拥塞避免阶段，只要出现一次超时（即出现一次网络拥塞），就把慢开始门限值 ssthresh 设置为当前的拥塞窗口值乘以 0.5。
- 当网络频繁出现拥塞时，ssthresh 值就下降得很快，以大大减少注入到网络中的分组数。

加法增大
Additive Increase

“加法增大”是指执行拥塞避免算法后，当收到对所有报文段的确认就将拥塞窗口 cwnd增加一个 MSS 大小，使拥塞窗口缓慢增大，以防止网络过早出现拥塞。

慢开始和拥塞避免算法的说明

- “拥塞避免”并非指完全能够避免了拥塞。利用以上的措施要完全避免网络拥塞还是不可能的。

- “拥塞避免”是说在拥塞避免阶段把拥塞窗口控制为按线性规律增长，使网络比较不容易出现拥塞。

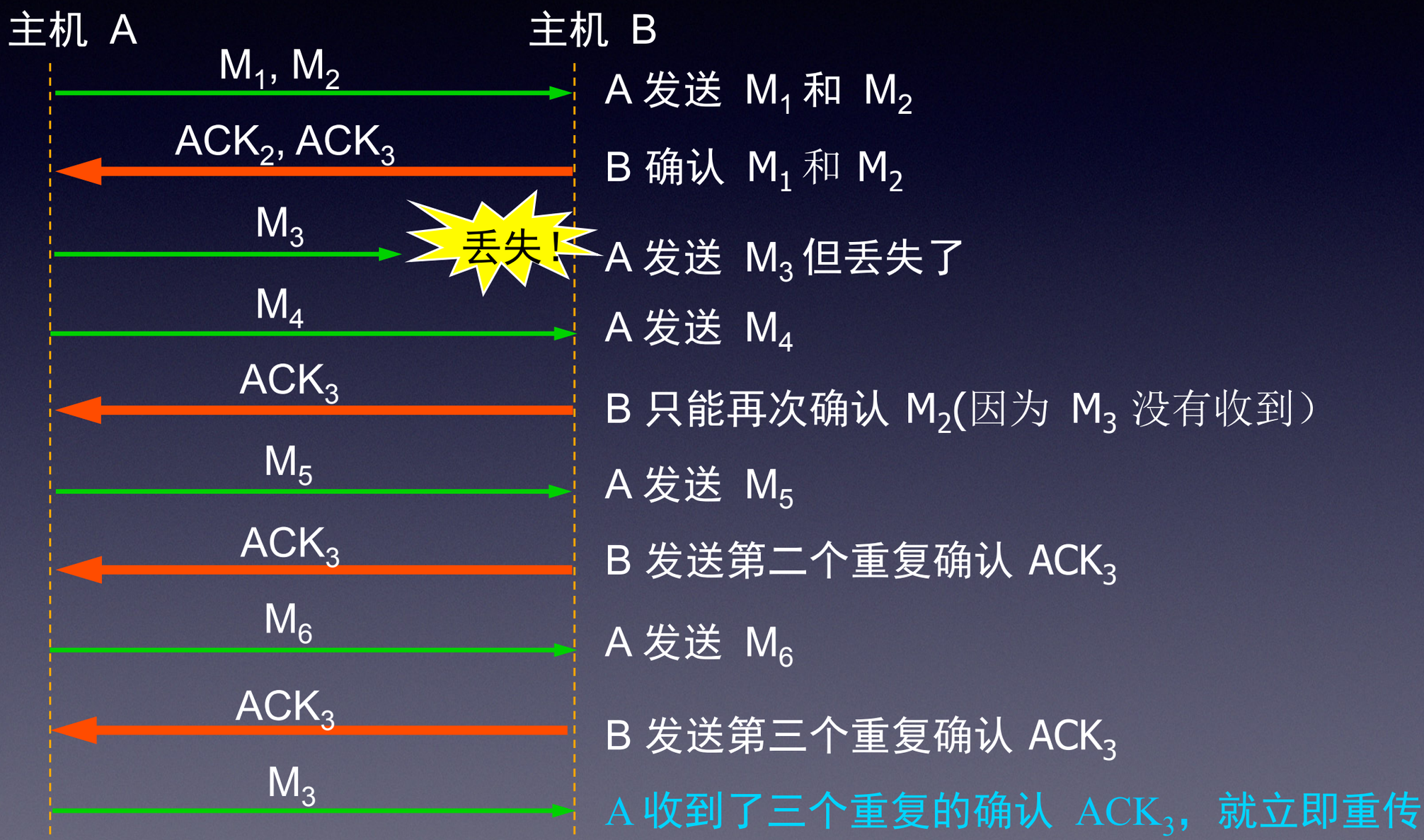
快重传和快恢复

- 快重传算法规定，接收方每收到一个失序的报文段就立即发出重复确认，而不要等到自己发送数据时才稍带确认，这样可以使发送方及早知道某一报文段没有到达接收方。

- 发送端只要一连收到三个重复的 ACK 即可断定有分组丢失了，就应立即重传丢失的报文段而不必继续等待为该报文段设置的重传计时器的超时。

- 不难看出，快重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段，由于发送方尽早重传未被确认的报文段，采用快重传后可以使整个网络的吞吐量提高约20%。

快重传举例

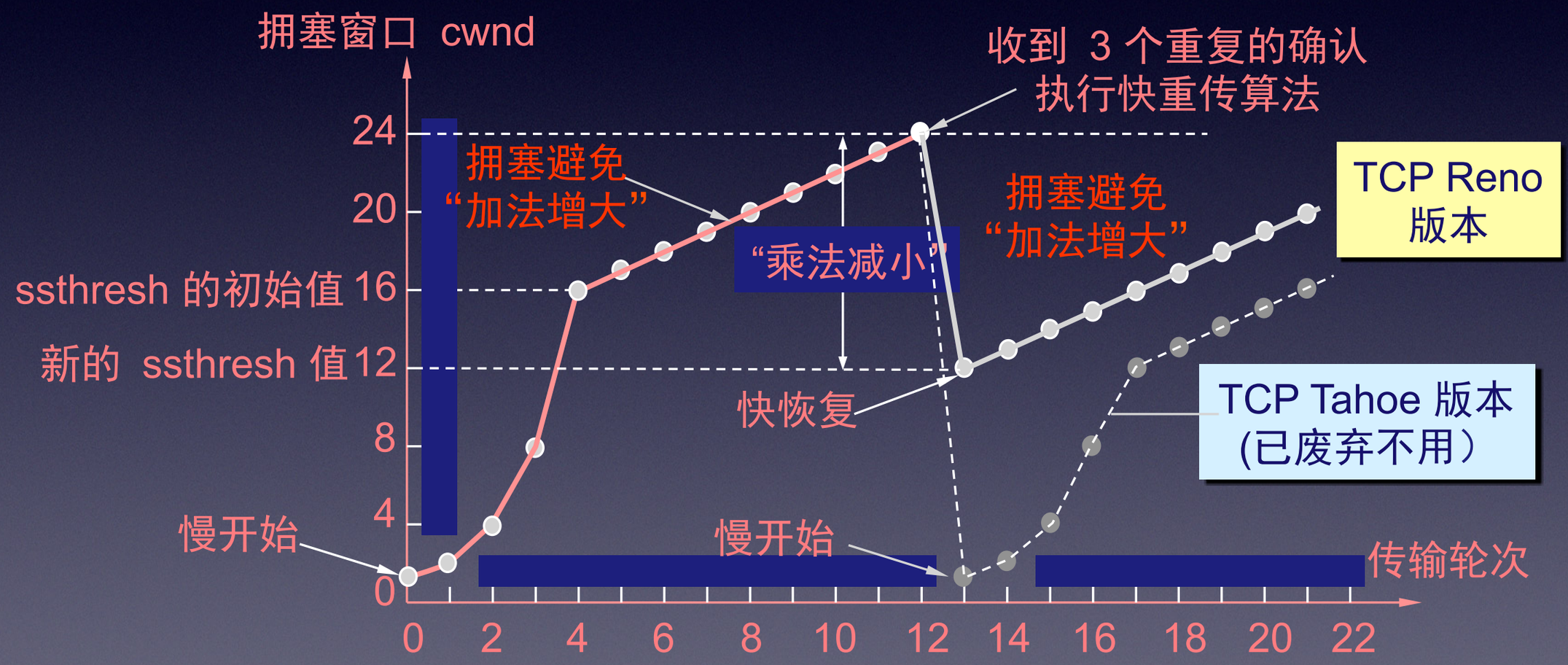

主机 A
主机 B
M_1, M_2
A 发送 M_1 和 M_2
ACK_2, ACK_3
B 确认 M_1 和 M_2
M_3
丢失！
A 发送 M_3 但丢失了
M_4
A 发送 M_4
ACK_3
B 只能再次确认 M_2(因为 M_3 没有收到）
M_5
A 发送 M_5
ACK_3
B 发送第二个重复确认 ACK_3
M_6
A 发送 M_6
ACK_3
B 发送第三个重复确认 ACK_3
M_3
A 收到了三个重复的确认 ACK_3，就立即重传 M_3，而不必等待超时重传。

快恢复算法

- 当发送端连续收到三个重复的 ACK 时，进行快重传的同时，还执行“乘法减小”算法，把慢开始门限ssthresh减半，以此预防网络发送拥塞。
- 由于发送方现在认为网络很可能没有发生拥塞(否则不可能连续收到三个ACK)，因此接下去不执行慢开始算法(cwnd不置为1)，而是把cwnd设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法，即每经一个传输轮次拥塞窗口cwnd只加1，按线性规律增长(注:也有的快重传此时设置cwnd为 ssthresh + 3 MSS)。
- 在采用快重传快恢复算法后，慢开始算法只是在TCP连接建立时和网络出现超时的时候才使用。
- 采用快开始和快恢复算法后，使得TCP的性能有明显改进。

目前广泛使用的TCP Reno

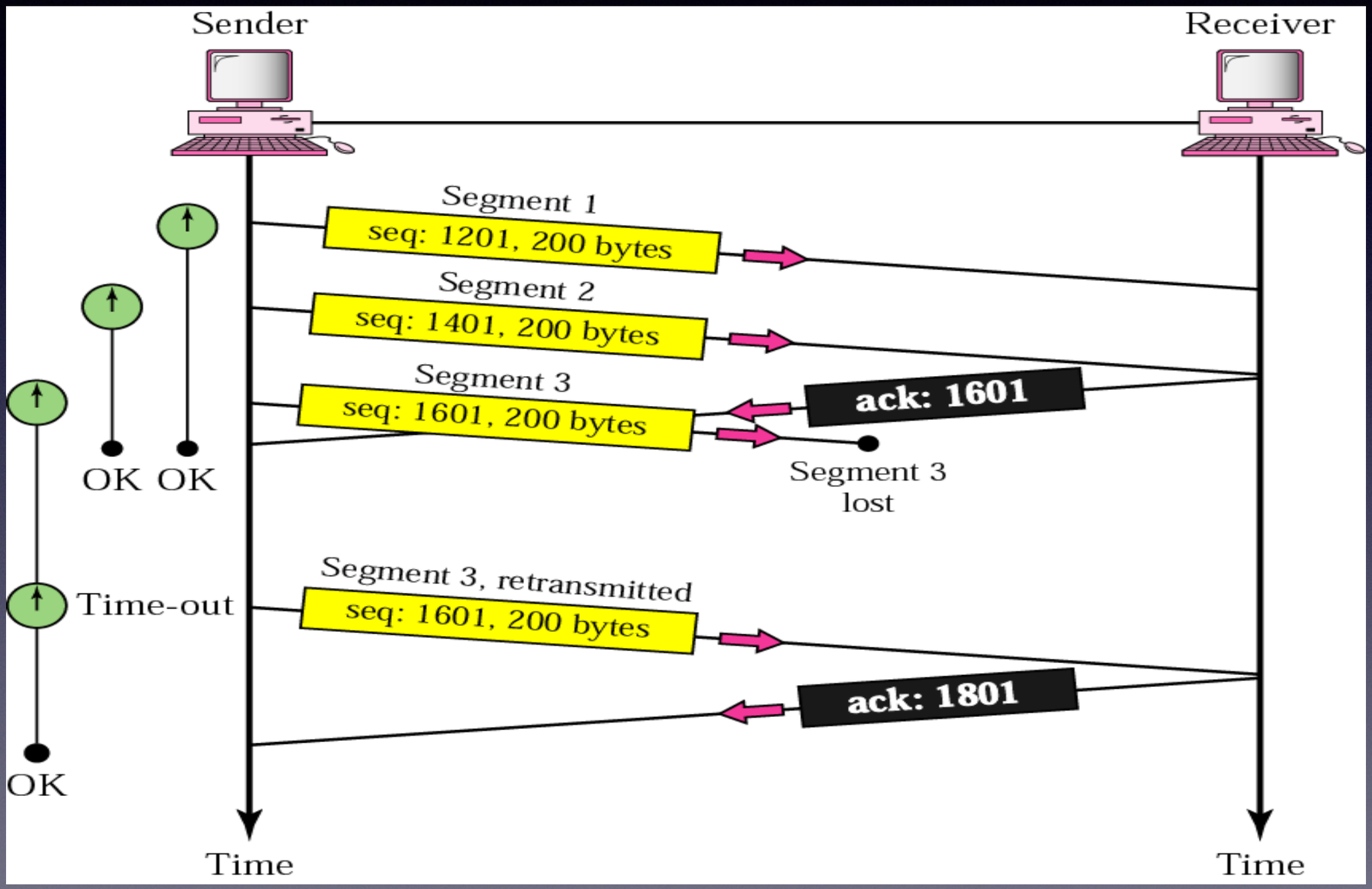
(除了连接建立及发送超时执行慢开始外，
在连续收到三个重复的ACK后还进行快重传、快恢复)



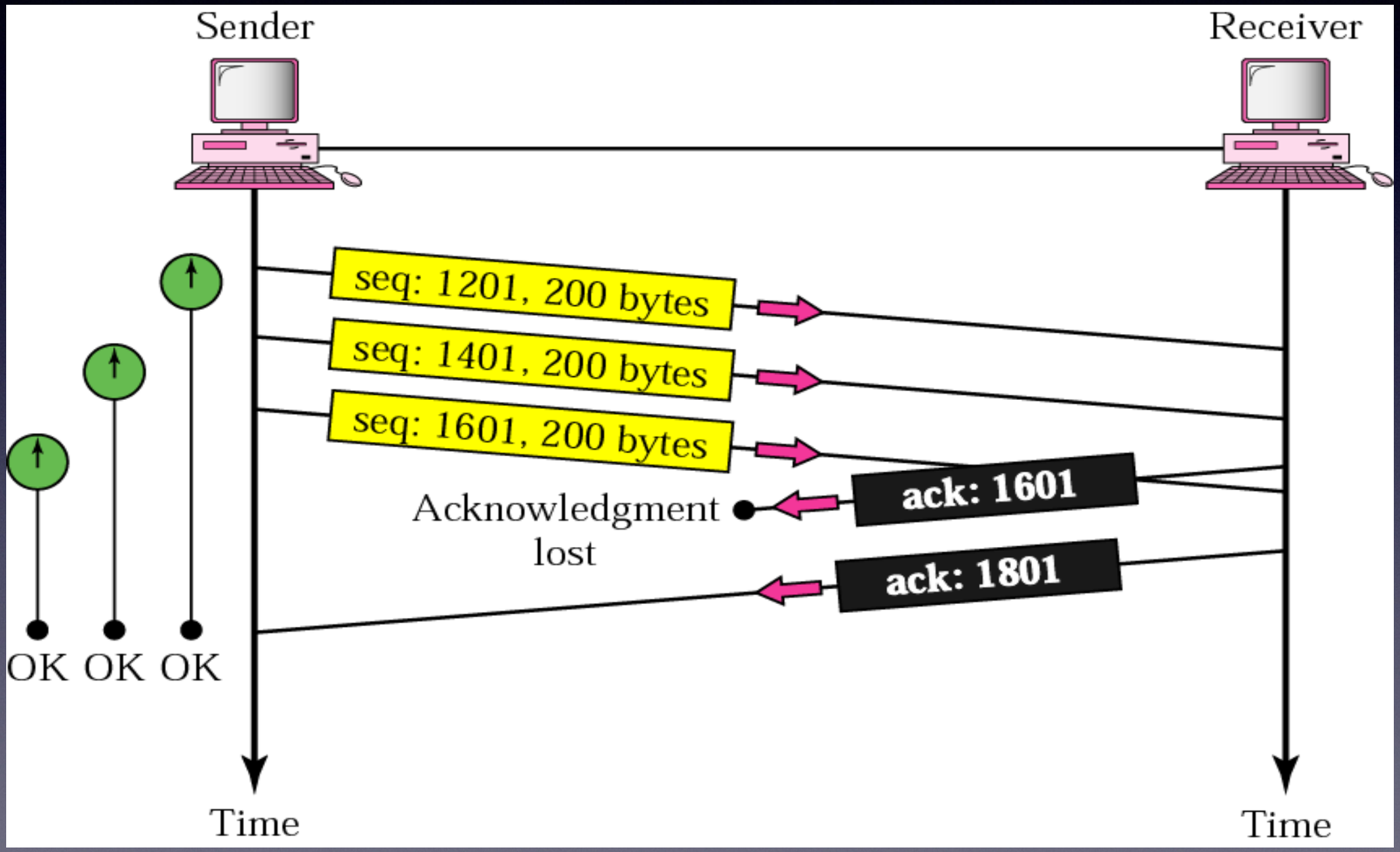
TCP重传机制

- 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。

报文丢失重传


Sender
Receiver
Segment 1
seq: 1201, 200 bytes
Segment 2
seq: 1401, 200 bytes
Segment 3
seq: 1601, 200 bytes
ack: 1601
OK OK
Segment 3
lost
Time-out
Segment 3, retransmitted
seq: 1601, 200 bytes
ack: 1801
OK
Time
Time

确认丢失重传


Sender
Receiver
seq: 1201, 200 bytes
seq: 1401, 200 bytes
seq: 1601, 200 bytes
ack: 1601
Acknowledgment lost
ack: 1801
OK OK OK
Time
Time

修正的Karn算法

- 报文段每重传一次，就将重传时间增大一些：

新的重传时间 = γ × (旧的重传时间)

- 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RTO的数值。
- 实践证明，这种策略较为合理。

“下课”

Q&A