计算机网络基础

第三章 传输层及传输层协议

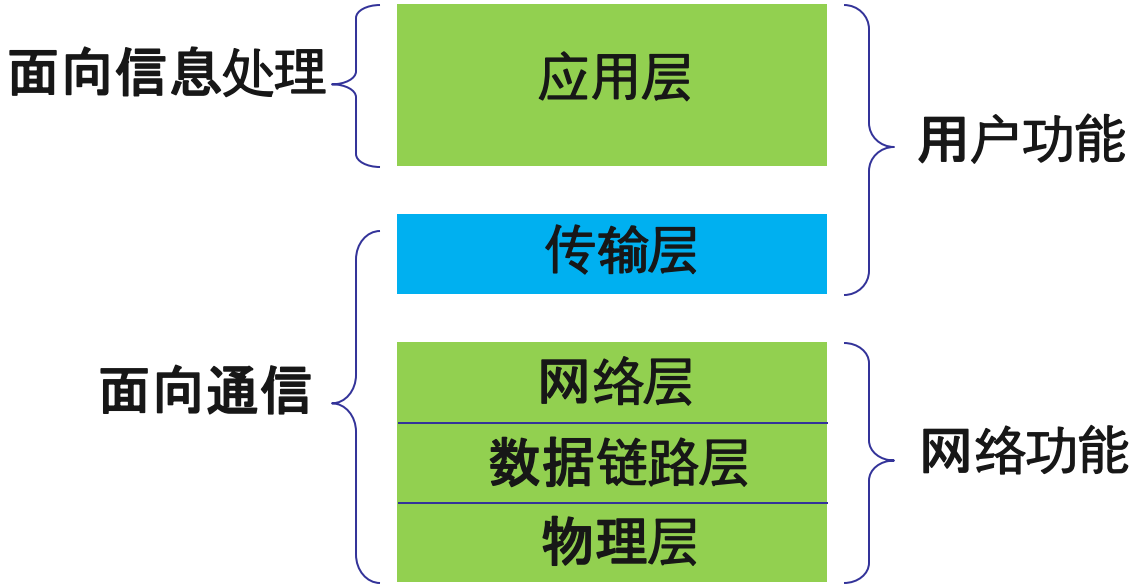
任课教师：马婷婷

第一部分 ▸

传输层协议概述

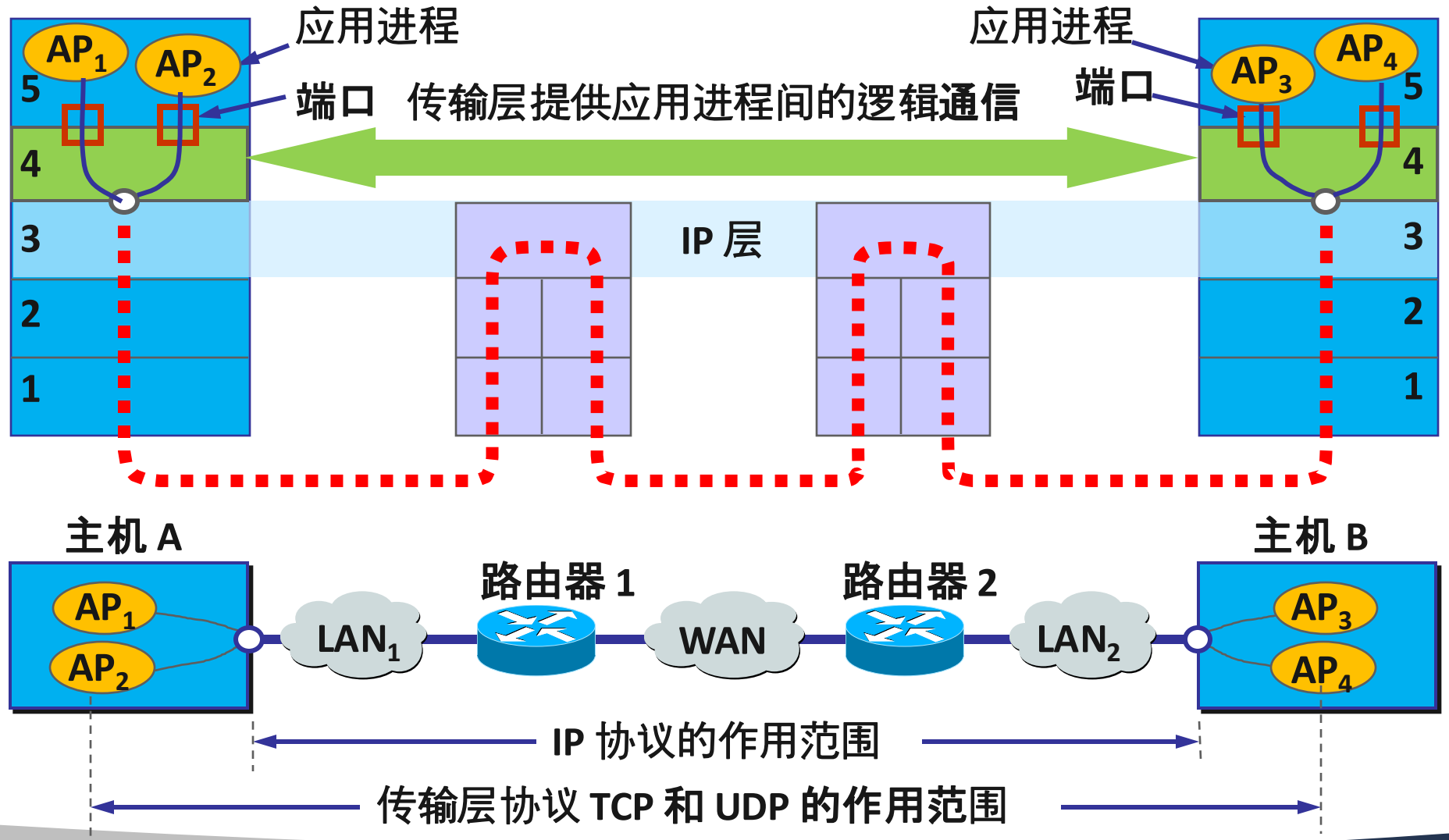
3.1.1 进程之间的通信

从通信和信息处理的角度看，传输层向它上面的应用层提供通信服务，它属于面向通信部分的最高层，同时也是用户功能中的最低层。



3.1.1 进程之间的通信

传输层为相互通信的应用进程提供了逻辑通信

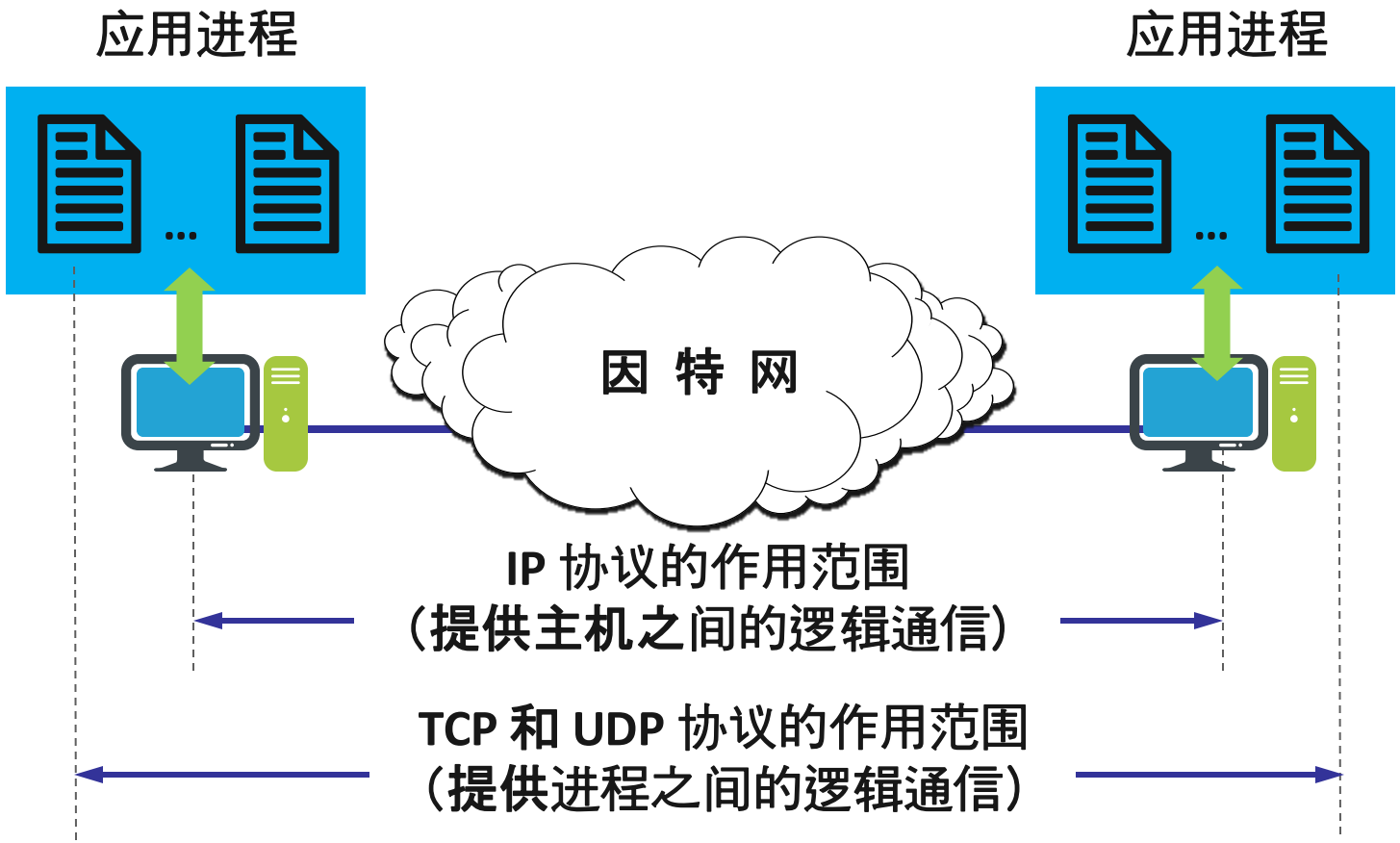


3.1.1 进程之间的通信

- 两个主机进行通信实际上就是两个主机中的应用进程互相通信。
- 应用进程之间的通信又称为**端到端的通信**。
- 传输层的一个很重要的功能就是复用和分用。应用层不同进程的报文通过不同的端口向下交到传输层，再往下就共用网络层提供的服务。
- “传输层提供应用进程间的逻辑通信”。“逻辑通信”的意思是：传输层之间的通信好像是沿水平方向传送数据。但事实上这两个传输层之间并没有一条水平方向的物理连接。

3.1.1 进程之间的通信

传输层协议和网络层协议的**主要区别**



3.1.1 进程之间的通信

传输层的主要功能

- 传输层为应用进程之间提供端到端的逻辑通信（但网络层是为主机之间提供逻辑通信）。
- 传输层还要对收到的报文进行差错检测。
- 传输层可选的功能：
  - 可靠数据传输
  - 流量控制
  - 拥塞控制

3.1.2 传输层协议

因特网的传输层有两个不同的协议：

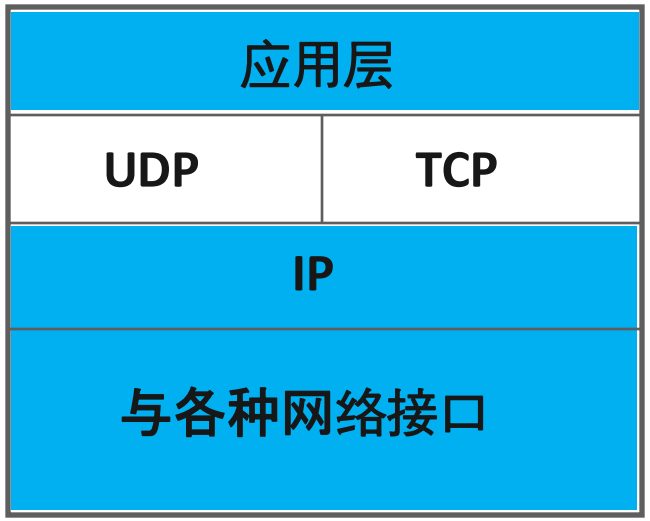
(1) 用户数据报协议 UDP (User Datagram Protocol)

(2) 传输控制协议 TCP (Transmission Control Protocol)

两个对等传输实体在通信时传送的数据单位叫作传输协议数据单元TPDU (Transport Protocol Data Unit)。

TCP 传送的协议数据单元称为 TCP 报文段(segment)

UDP 传送的协议数据单元称为 UDP 报文或用户数据报。



3.1.2 传输层协议

- UDP 在传送数据之前不需要先建立连接。对方的传输层在收到 UDP 报文后，不需要给出任何确认。虽然 UDP 不提供可靠交付，但在某些情况下 UDP 是一种最有效的工作方式。
- TCP 则提供面向连接的服务。TCP 不提供广播或多播服务。由于 TCP 要提供可靠的、面向连接的传输服务，因此不可避免地增加了许多的开销。这不仅使协议数据单元的首部增大很多，还要占用许多的处理机资源。

3.1.2 传输层协议

- 传输层的 UDP用户数据报与网际层的IP数据报有很大区别。IP数据报要经过互连网中许多路由器的存储转发，但 UDP 用户数据报是在传输层的端到端抽象的逻辑信道中传送的。
- TCP 报文段是在传输层抽象的端到端逻辑信道中传送，这种信道是可靠的全双工信道。但这样的信道却不知道究竟经过了哪些路由器，而这些路由器也根本不知道上面的传输层是否建立了 TCP 连接。

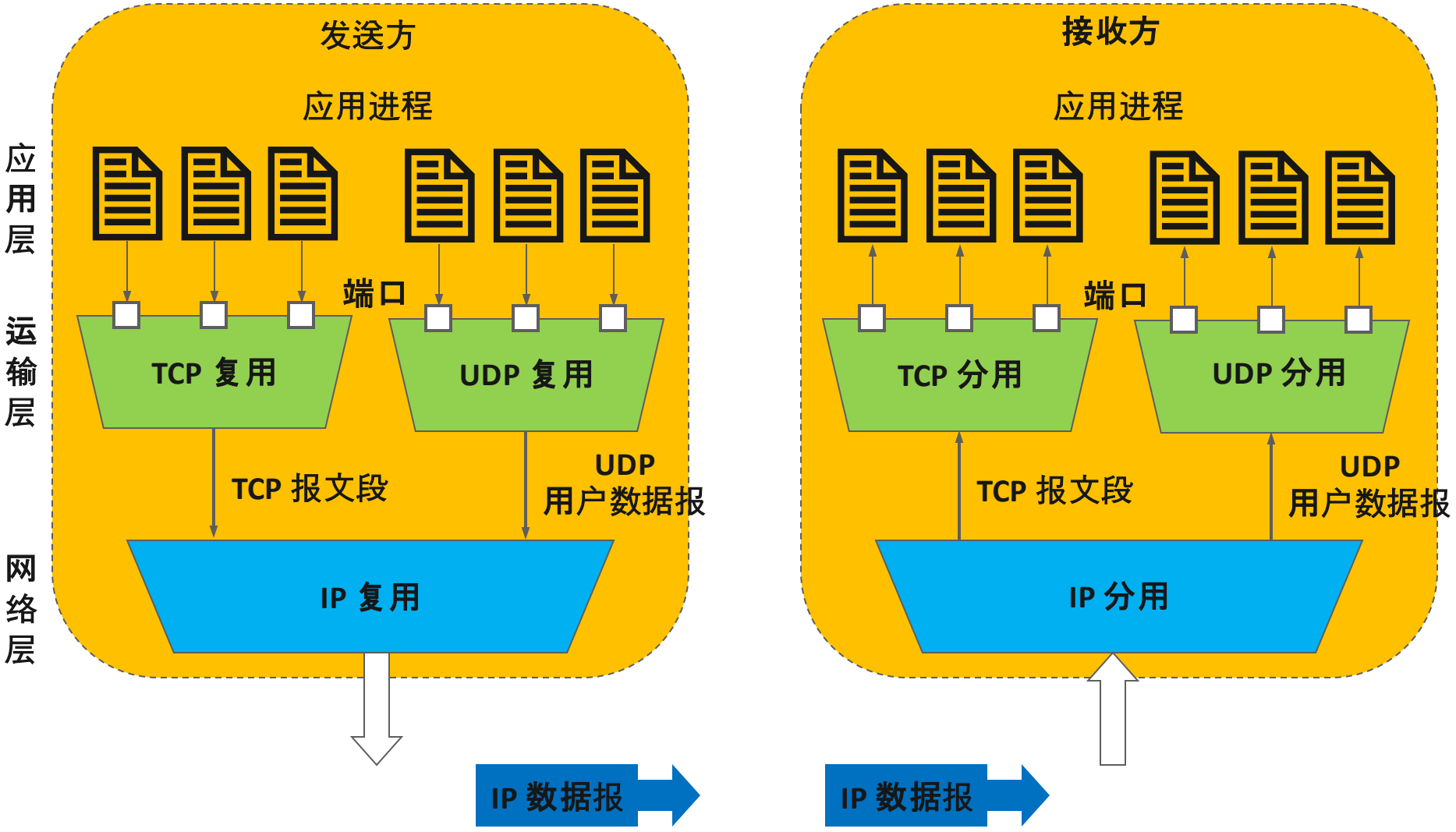
3.1.3 传输层的复用与分用

- **复用**是指在发送方不同的应用进程都可以使用同一个传输层协议传送数据（当然需要加上适当的首部）；
- 而**分用**是指接收方的传输层在剥去报文的首部后能够把这些数据正确交付到目的应用进程。
- 要能正确地将数据交付给指定应用进程，就必须给每个应用进程赋予一个明确的标志。
- 在TCP/IP网络中，使用一种与操作系统无关的协议端口号(protocol port number)（简称端口号）来实现对通信的应用进程的标志。

3.1.3 传输层的复用与分用

- **端口**就是应用进程的**传输层地址**。
- 端口的作用就是让应用层的各种应用进程都能将其数据通过端口向下交付给传输层，以及让传输层知道应当将其报文段中的数据向上通过端口交付给应用层相应的进程。
- 从这个意义上讲，端口是用来标志应用层的进程。

3.1.3 传输层的复用与分用



端口的作用

3.1.3 传输层的复用与分用

端口用一个 16 位端口号进行标志。

端口号只具有本地意义，即端口号只是为了标志本计算机应用层中的各进程。
在因特网中不同计算机的相同端口号是没有联系的。

并且TCP和UDP端口号之间也没有必然联系

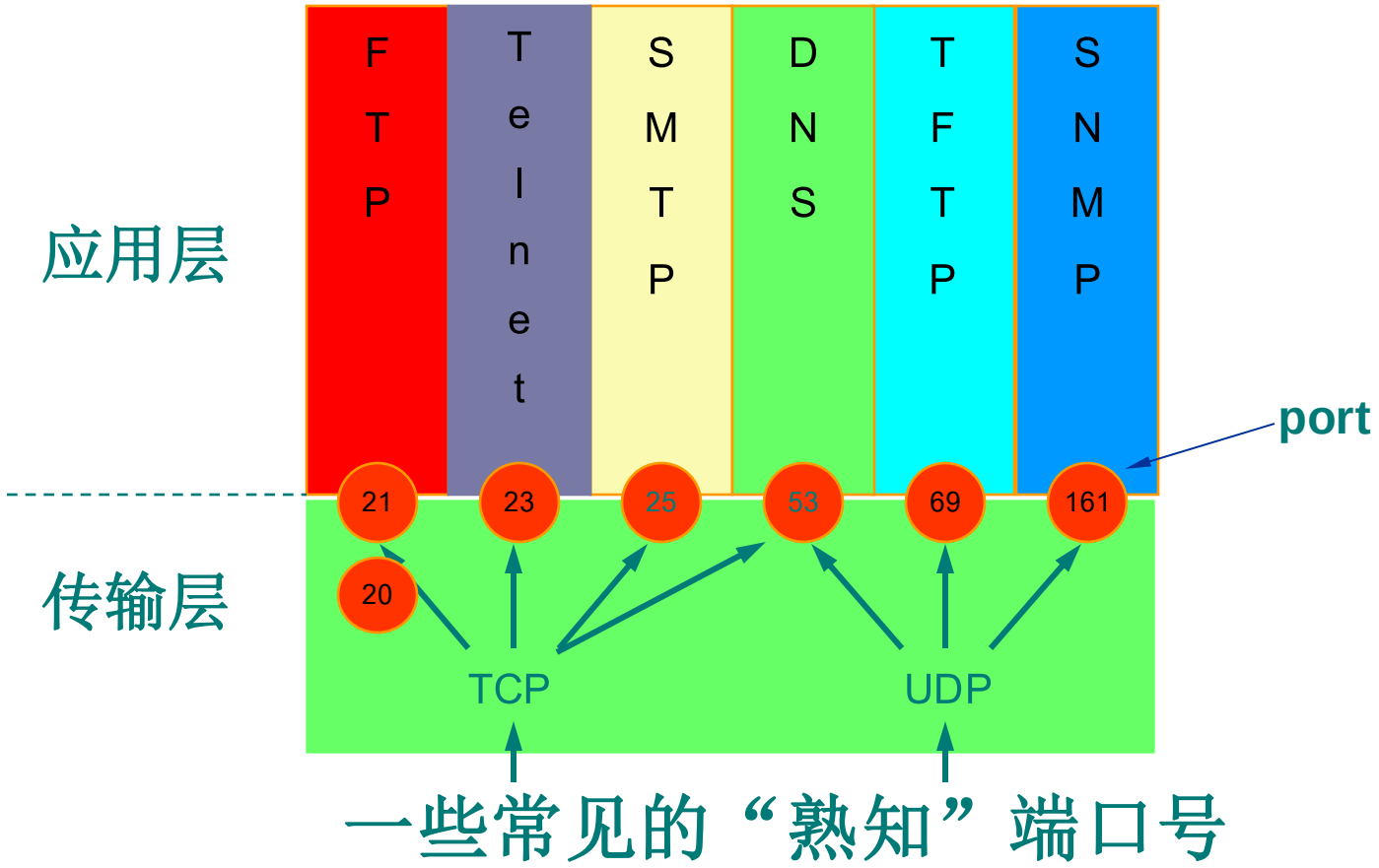
端口的种类

1）**熟知端口** 其数值一般为 0~1023。当一种新的应用程序出现时，必须为它指派一个熟知端口，用于服务器端。

2）**登记端口** 其数值为 1024~49151。这类端口是 ICANN 控制的，使用这个范围的端口必须在 ICANN 登记，以防止重复。

3）**动态端口** 其数值为 49152~65535。这类端口是留给客户进程选择作为临时端口，用于客户端。

3.1.3 传输层的复用与分用



第二部分 ▸

用户数据报协议UDP

3.2.1 UDP概述

UDP 只在 IP 的数据报服务之上增加了很少一点的功能，即端口的功能和差错检测的功能。

虽然 UDP 用户数据报只能提供不可靠的交付，但 UDP 在某些方面有其特殊的优点。

- 发送数据之前不需要建立连接
- UDP 的主机不需要维持复杂的连接状态表。
- UDP 用户数据报只有 8 个字节的首部开销。
- 网络出现的拥塞不会使源主机的发送速率降低。这对某些实时应用是很重要的。

3.2.1 UDP概述

UDP的特点

1) UDP 是无连接的，即发送数据之前不需要建立连接（当然发送数据结束时也没有连接可释放），因此减少了开销和发送数据之前的时延。

2) UDP 使用尽最大努力交付，即不保证可靠交付，同时也不使用拥塞控制，因此主机不需要维持具有许多参数的、复杂的连接状态表。

3) 由于 UDP 没有拥塞控制，因此网络出现的拥塞不会使源主机的发送速率降低。这对某些实时应用是很重要的。很多的实时应用（如 IP 电话、实时视频会议等）要求源主机以恒定的速率发送数据，并且允许在网络发生拥塞时丢失一些数据，但却不允许数据有太大的时延。UDP 正好适合这种要求。

3.2.1 UDP概述

4) UDP 是面向报文的。这就是说，UDP 对应用程序交下来的报文不再划分为若干个分组来发送，也不把收到的若干个报文合并后再交付给应用程序。

- **应用程序交给 UDP 一个报文，UDP 就发送这个报文；而 UDP 收到一个报文，就把它交付给应用程序。**
- **应用程序必须选择合适大小的报文。**

5) UDP 支持一对一、一对多、多对一和多对多的交互通信。

6) 用户数据报只有 8 个字节的首部开销，比 TCP 的 20 个字节的首部要短。

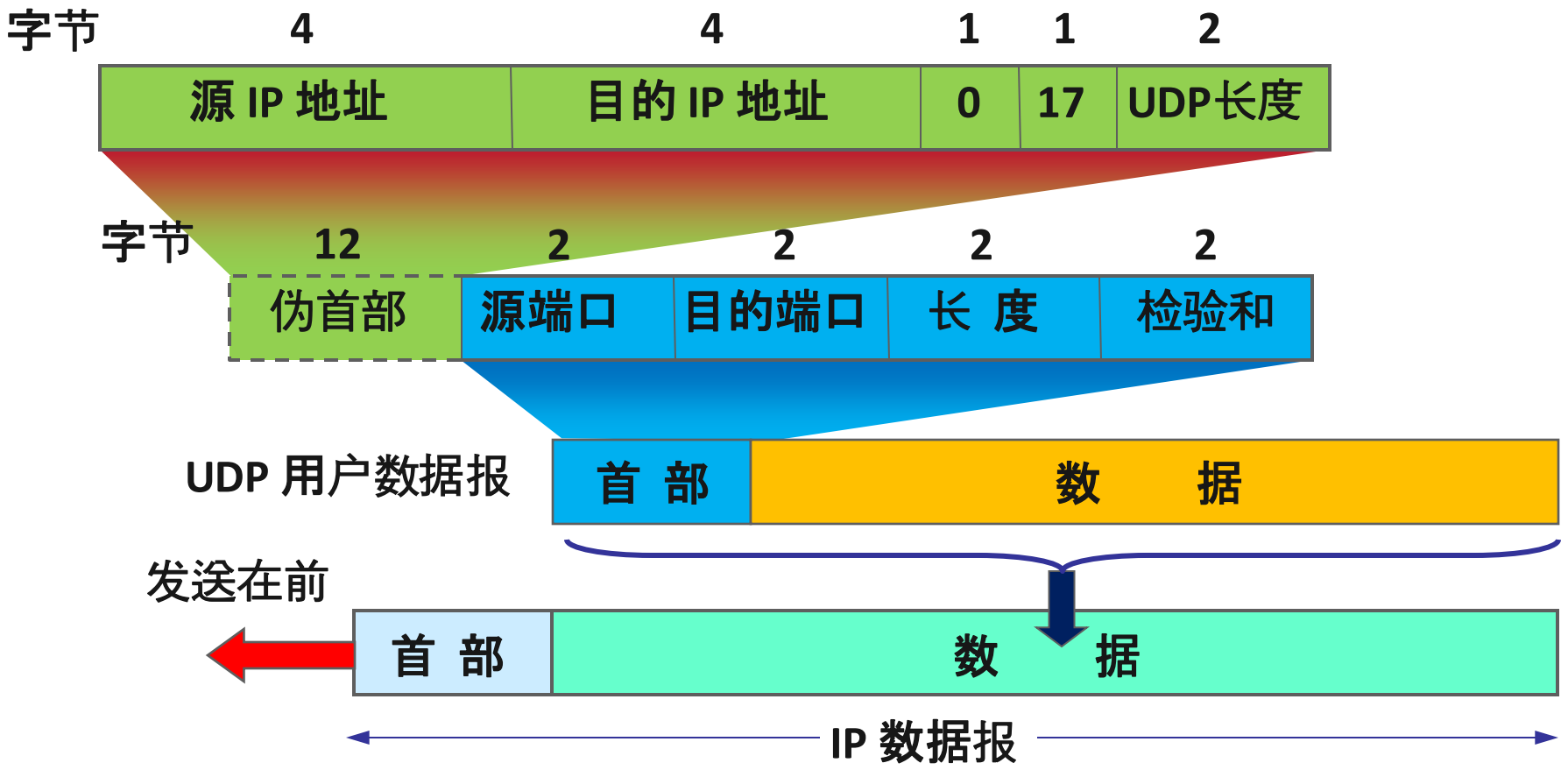
3.2.1 UDP概述

UDP的问题

虽然某些实时应用需要使用没有拥塞控制的UDP，但当很多的源主机同时都向网络发送高速率的实时视频流时，网络就有可能发生拥塞，结果大家都无法正常接收。

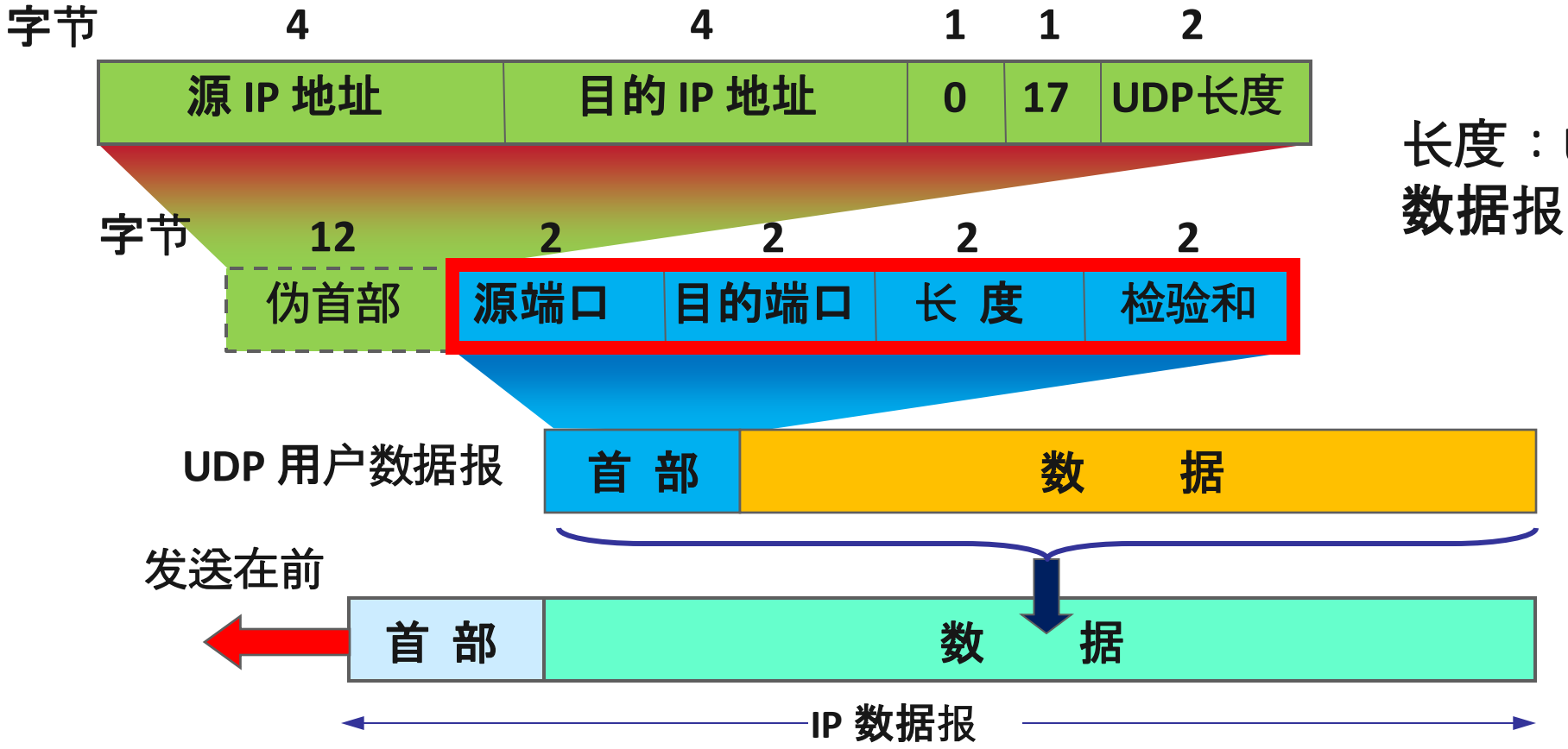
还有一些使用 UDP 的实时应用需要对UDP的不可靠的传输进行适当的改进以减少数据的丢失。

3.2.2 UDP报文格式


字节
4
4
1
1
2
源 IP 地址
目的 IP 地址
0
17
UDP长度
字节
12
2
2
2
2
伪首部
源端口
目的端口
长 度
检验和
UDP 用户数据报
首 部
数 据
发送在前
首 部
数 据
IP 数据报

3.2.2 UDP报文格式

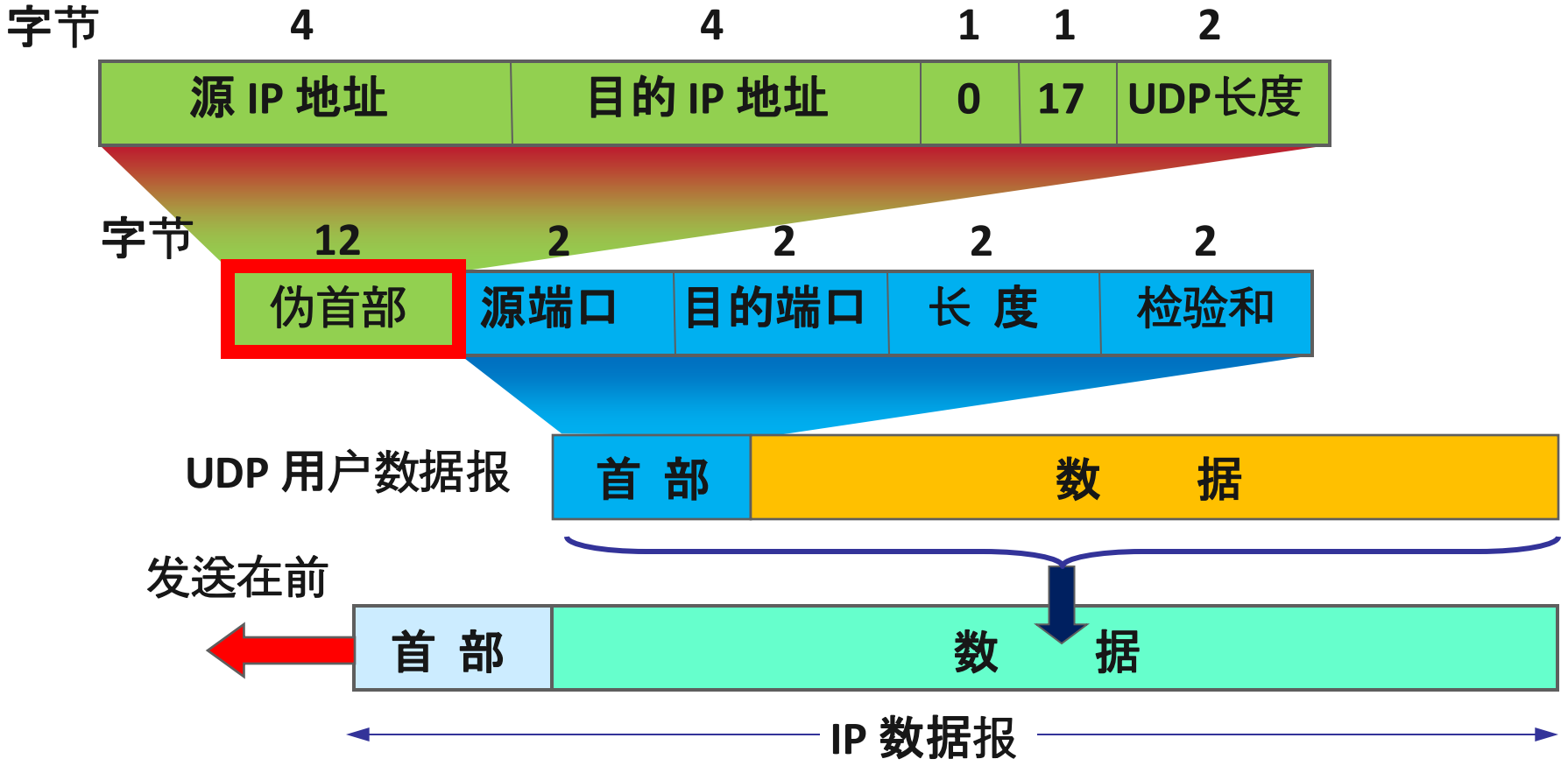
用户数据报 UDP 有两个字段：数据字段和首部字段。首部字段有 8 个字节，由 4 个字段组成，每个字段都是两个字节。



长度：UDP用户数据报的长度。

3.2.2 UDP报文格式

在计算检验和时，临时把“伪首部”和 UDP 用户数据报连接在一起。伪首部仅仅是为了计算检验和。



3.2.3 UDP工作过程

- **数据报发送**
  - **UDP 软件将用户数据封装在 UDP 数据报中**
  - **转交给 IP 软件，进行 IP 封装和转发**
- **数据报的接收**
  - **端口判断该报文的目的端口号是否与当前端口匹配**
    - **若匹配成功，将该数据报保存到相应端口的接收队列中；（若队列已满，则丢弃该数据报）**
    - **若未匹配，则丢弃该数据报，同时向源端发送“端口不可达”的 ICMP 包**

3.2.4 UDP协议适用的范围

确定应用程序在传输层是否采用UDP协议的原则：

- **系统对性能的要求高于对数据完整性的要求；**
- **需要“简短快捷”的数据交换；**
- **需要多播和广播的应用。**

UDP协议是一种适用于实时语音与视频传输的传输层协议。

第三部分 ▶

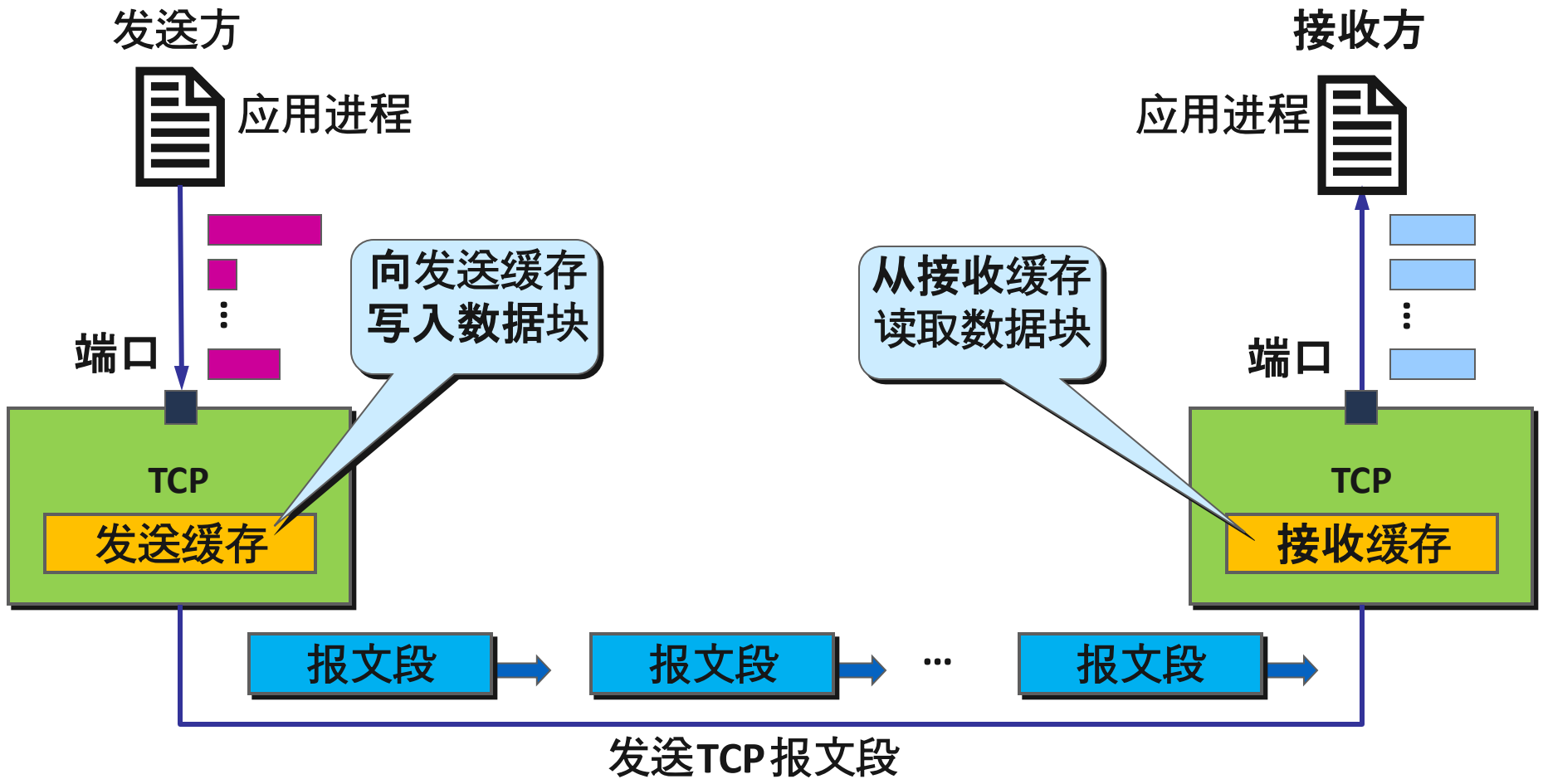
传输控制协议TCP

3.3.1　TCP的主要特点

- TCP 是面向连接的传输层协议。
- 每一条 TCP 连接只能有两个端点(endpoint)，每一条 TCP 连接只能是点对点的（一对一）。
- TCP 提供可靠交付的服务。
- TCP 提供全双工通信。
- 面向字节流。

3.3.1 TCP的主要特点

面向字节流？？——无结构的字节流



3.3.1 TCP的主要特点

- TCP 连接是一条虚连接而不是一条真正的物理连接。
- TCP 对应用进程一次把多长的报文发送到TCP 的缓存中是不关心的。
- TCP 根据网络的具体情况来决定一个报文段应包含多少个字节（UDP 发送的报文长度是应用进程给出的）。
- TCP 可把太长的数据块划分短一些再传送。TCP 也可等待积累有足够多的字节后再构成报文段发送出去。

3.3.1 TCP的主要特点

全双工？？

- **通信是全双工方式。**
- **发送方的应用进程按照自己产生数据的规律，不断地把数据块陆续写入到 TCP 的发送缓存中。TCP 再从发送缓存中取出一定数量的数据，将其组成 TCP 报文段(segment)逐个传送给 IP 层，然后发送出去。**
- **接收方从 IP 层收到 TCP 报文段后，先把它暂存在接收缓存中，然后让接收方的应用进程从接收缓存中将数据块逐个读取。**

3.3.1 TCP的主要特点

一对一？？

由于传输层的通信是面向连接的，因此TCP 每一条连接上的通信只能是一对一的，而不可能是一对多、多对一或多对多的。

TCP 的连接

TCP 把连接作为最基本的抽象。

每一条 TCP 连接唯一地被通信两端的两个端点所确定。即：

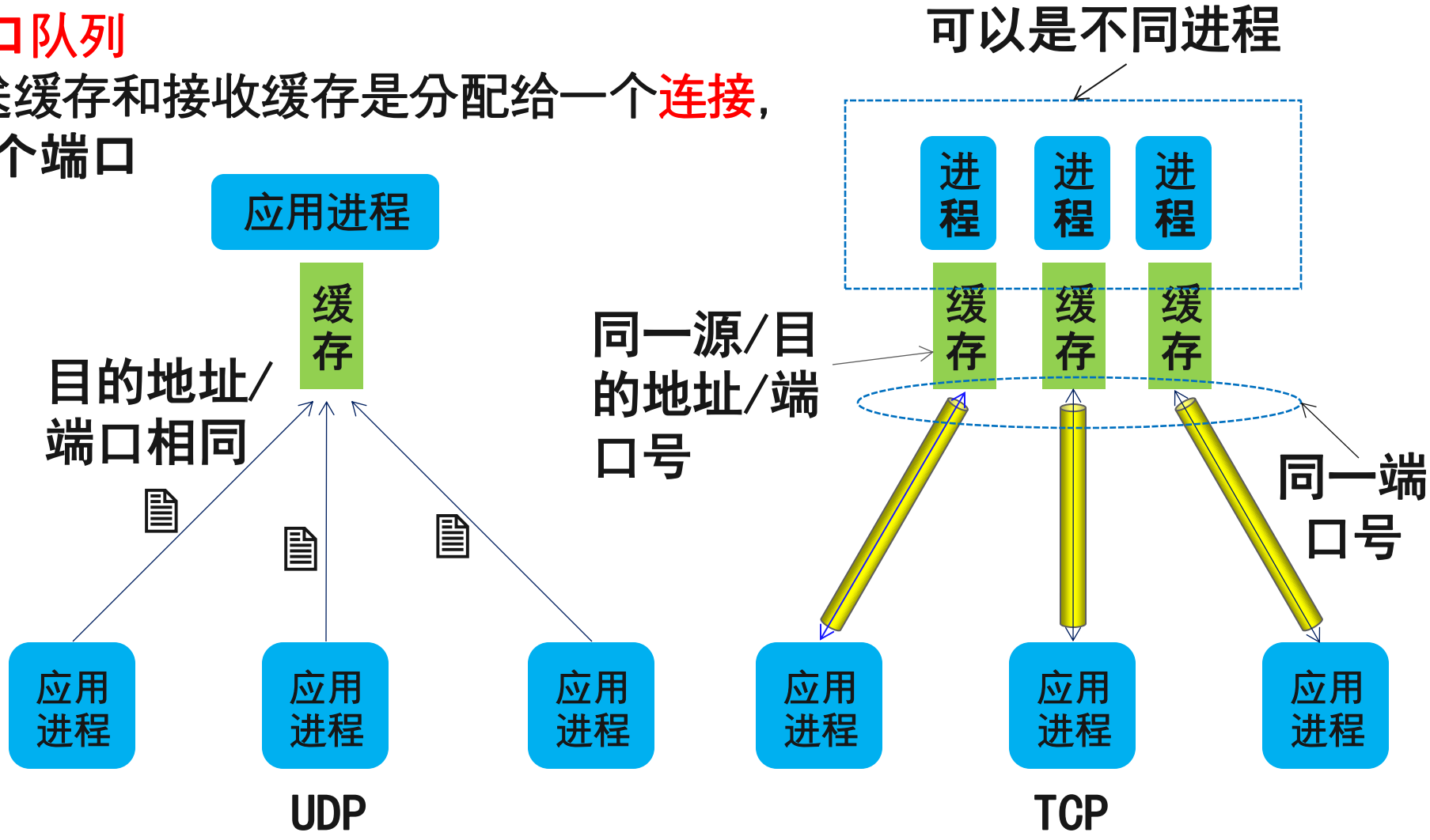
TCP 连接 ::= {(IP1: port1), (IP2: port2)}

3.3.1 TCP的主要特点

UDP与TCP复用的区别

UDP是端口队列

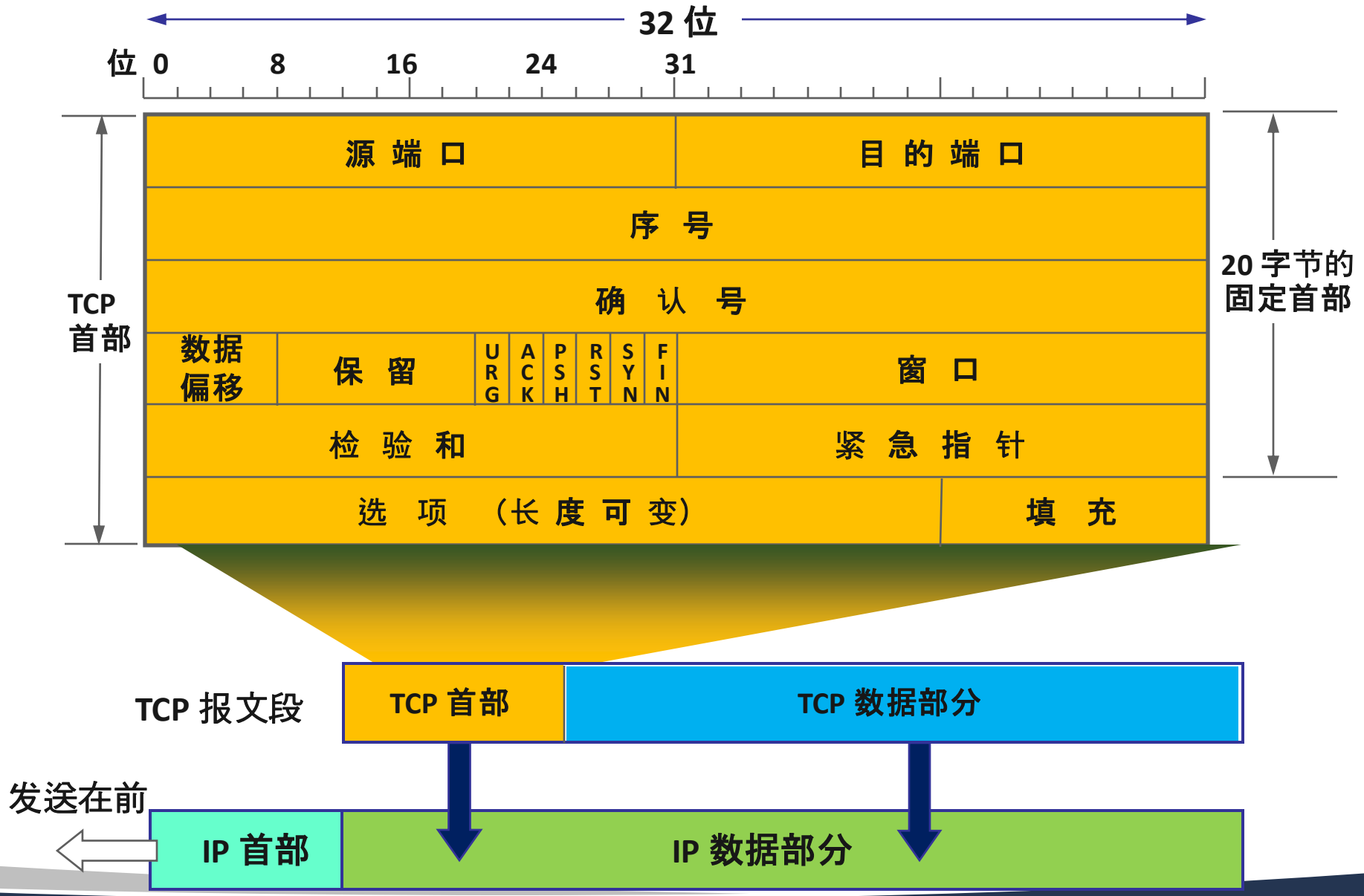
TCP的发送缓存和接收缓存是分配给一个连接，而不是一个端口



3.3.2 TCP报文段结构

- TCP 报文段分为首部和数据两部分。
- 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* 必须是整数)。因此 TCP 首部的最小长度是 20 字节。

3.3.2 TCP报文段结构



3.3.2 TCP报文段结构

源端口和目的端口字段——各占 2 字节。端口是传输层与应用层的服务接口。传输层的复用和分用功能都要通过端口才能实现。

3.3.2 TCP报文段结构

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。

3.3.2 TCP报文段结构

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。

3.3.2 TCP报文段结构

数据偏移——占 4 位，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。

“数据偏移”的单位不是字节而是 32 位字（4 字节为计算单位）。

3.3.2 TCP报文段结构

保留字段——占 6 位，保留为今后使用，但目前应置为 0。

3.3.2 TCP报文段结构

紧急位 URG —— 当 URG = 1 时，表明紧急指针字段有效。它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)。

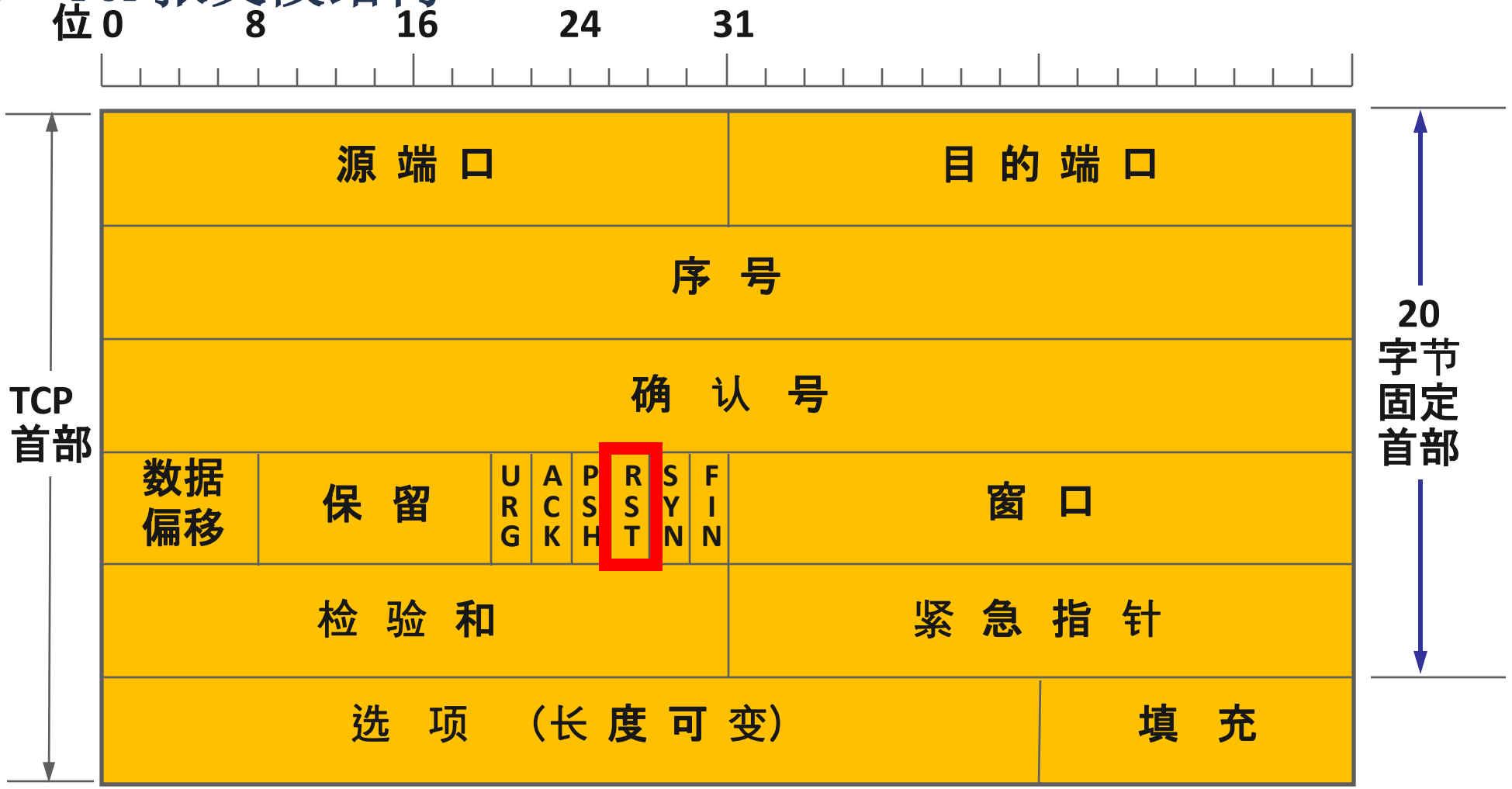
3.3.2 TCP报文段结构

确认位 ACK —— 只有当 ACK = 1 时确认号字段才有效。当 ACK = 0 时，确认号无效。

3.3.2 TCP报文段结构

推送位 PSH (PuSH) —— 接收 TCP 收到 PSH = 1 的报文段，就尽快地交付给接收应用进程，而不再等到整个缓存都填满了后再向上交付。

3.3.2 TCP报文段结构



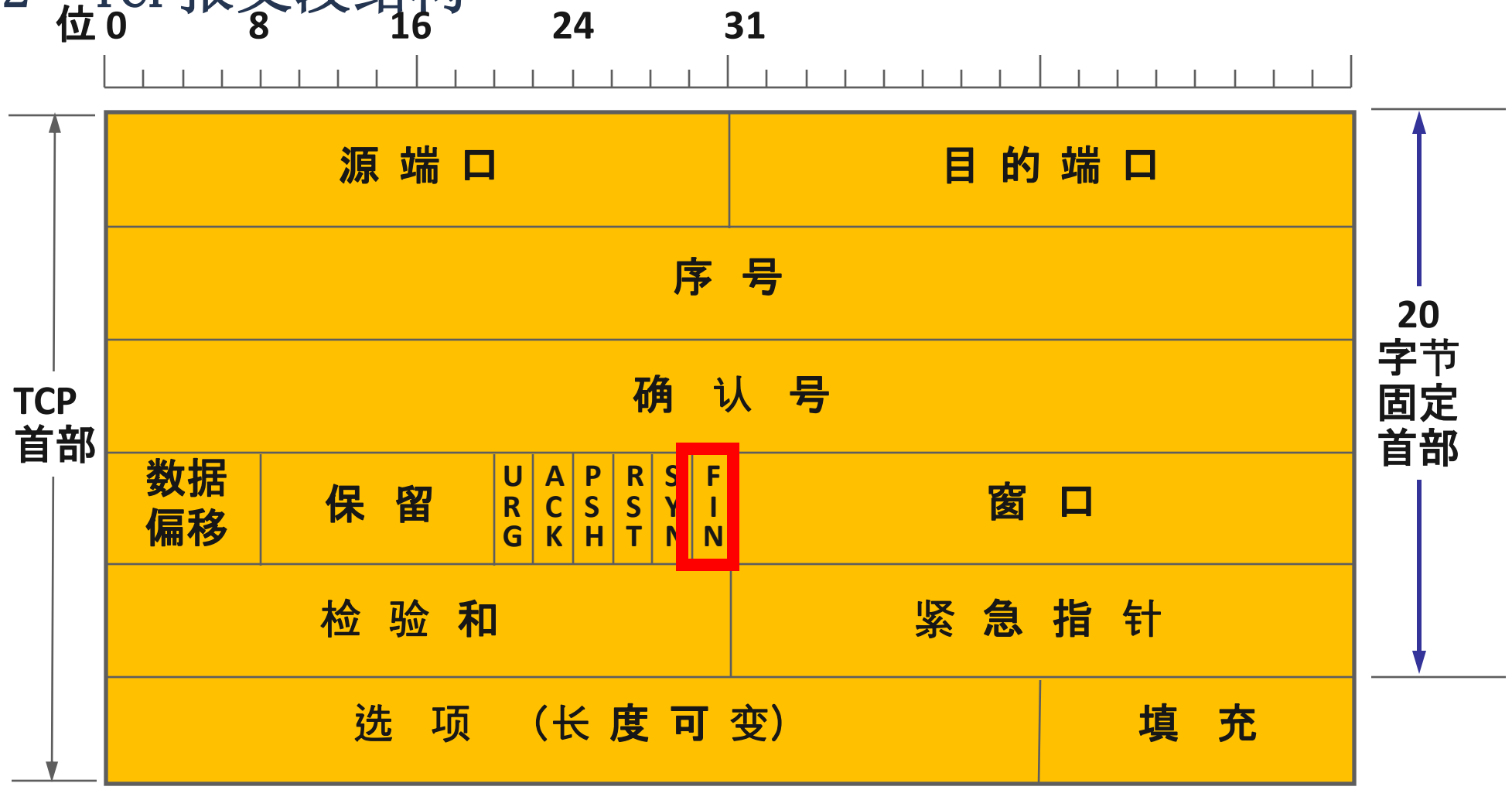
复位位 RST (ReSeT) —— 当 RST = 1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立传输连接。

3.3.2 TCP报文段结构

同步位 SYN —— 当 SYN = 1 时，表示这是一个连接请求或连接接受报文。

SYN=1,ACK=0,表示连接请求报文； SYN=1,ACK=1,表示对方同意建立连接。

3.3.2 TCP报文段结构



终止位 FIN (FINal) —— 用来释放连接。当 FIN = 1 时，表明此报文段的发送方的数据已发送完毕，并要求释放传输连接。

3.3.2 TCP报文段结构

窗口：通知对方自己接收窗口的大小。

确认号和窗口一起确定对方的发送窗口。

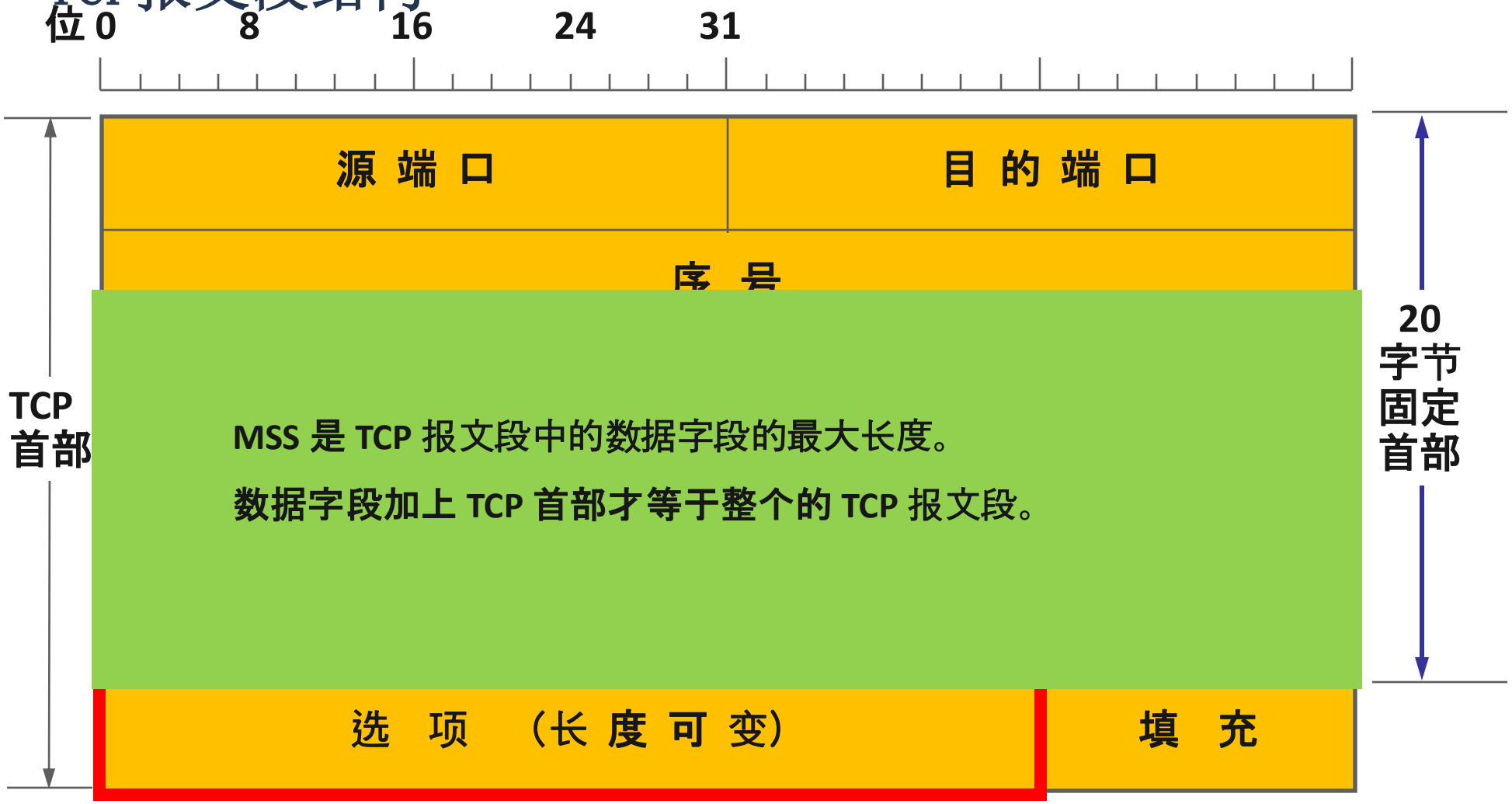
3.3.2 TCP报文段结构

检验和 —— 占 2 字节。检验和字段检验的范围包括首部和数据这两部分。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。

3.3.2 TCP报文段结构

紧急指针字段 —— 占 16 位。紧急指针指出在本报文段中的紧急数据的**最后一个字节**的序号。

3.3.2 TCP报文段结构



选项字段 —— 长度可变。TCP 只规定了一种选项，即最大报文段长度 MSS (Maximum Segment Size)。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。”

3.3.2 TCP报文段结构

填充字段 —— 这是为了使整个首部长度是 4 字节的整数倍。

3.3.3 TCP的可靠传输

- 我们知道因特网的网络层服务是不可靠的，即通过IP传送的数据可能出现差错、丢失、乱序或重复。
- TCP在IP的不可靠的尽最大努力服务的基础上实现了一种可靠数据传输服务，保证数据无差错、无丢失、按序和无重复的交付。
- 由于TCP下面的传输数据的互联网的结构非常复杂，因此不能采用最简单的停止等待协议来实现可靠传输。

3.3.3 TCP的可靠传输

1)数据编号与确认

- TCP 协议是面向字节的。TCP 将所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于一个序号。
- 在连接建立时，双方要商定初始序号。TCP 每次发送的报文段的首部中的序号字段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。接收方返回 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**接收方期望下次收到的数据中的第一个数据字节的序号**。

累计确认

- 由于TCP连接能提供全双工通信，因此通信中的每一方都不必专门发送确认报文段，而可以在传送数据时顺便把确认信息**捎带**传送。

3.3.3 TCP的可靠传输

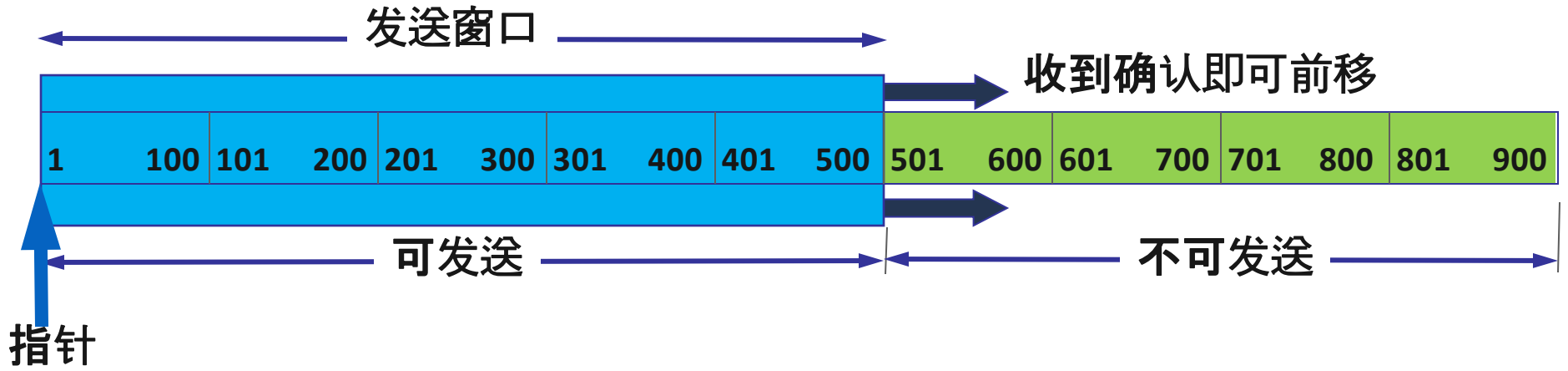
- 为此，**接收方在正确接收到数据**时可能要等待一小段时间（**不能超过0.5秒**）再发送确认，**若**这段时间内有数据要发送给对方，则采用**捎带确认**。这样做可以提高传输效率。
- **接收方若收到有差错的**报文段就丢弃（**不发送否认信息**）。
- **若收到重复的**报文段，**也要丢弃，但要发回（或捎带发回）确认信息。**
- **若收到失序的**报文段，**可**选择将失序报文段丢弃，**或者先将其**暂存于接收缓存内，**待所缺序号的**报文段收齐后再一起上交应用层。**注意，不论采用哪种方法，接收方都要**对已按序接收到的数据进行确认。

3.3.3 TCP的可靠传输

2）以字节为单位的滑动窗口

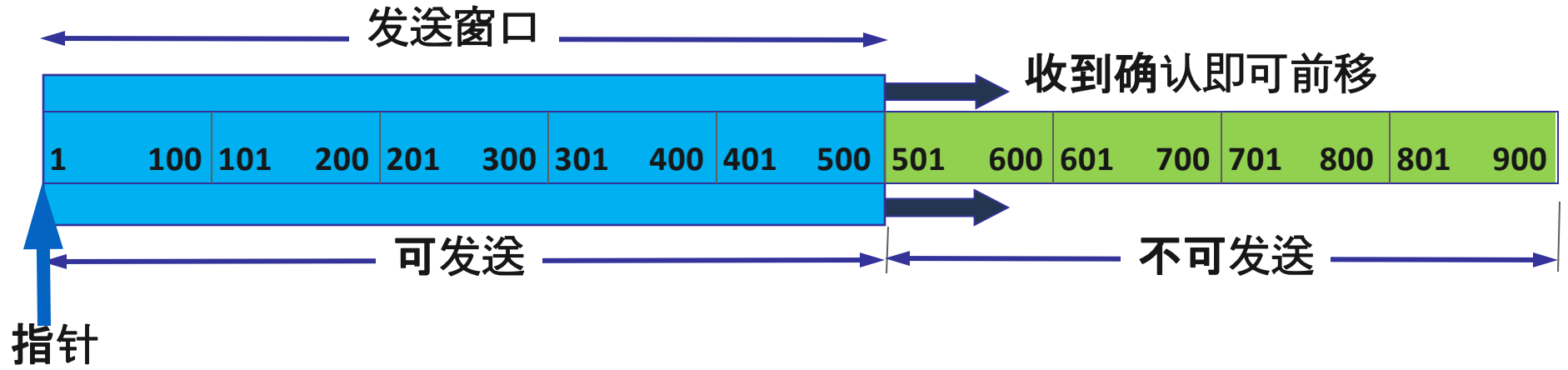
- TCP 采用大小可变的滑动窗口进行**流量控制**。窗口大小的单位是字节。
- 在 TCP 报文段首部的窗口字段写入的数值就是当前给对方设置的**发送窗口数值的上限**。
- 发送窗口在连接建立时由双方商定。但在通信的过程中，接收方可根据自己的资源情况，随时**动态地调整**对方的发送窗口上限值(可增大或减小)。

3.3.3 TCP的可靠传输

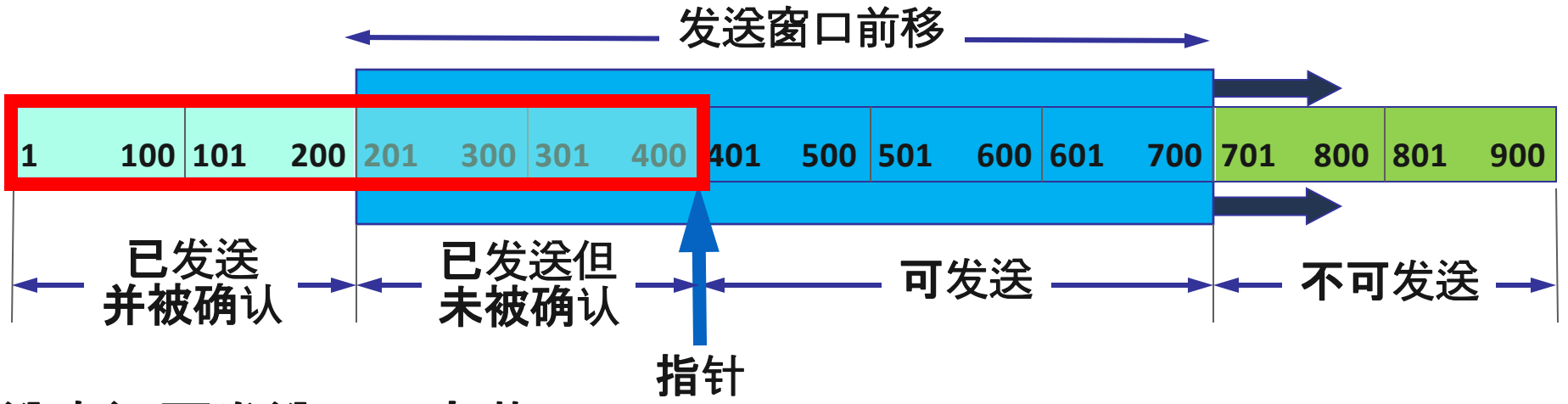


- 发送方要发送 900 字节长的数据，划分为 9 个 100 字节长的报文段，而发送窗口确定为 500 字节。
- 发送方只要收到了对方的确认，发送窗口就可前移。
- 发送 TCP 要维护一个指针。每发送一个报文段，指针就向前移动一个报文段的距离。

3.3.3 TCP的可靠传输



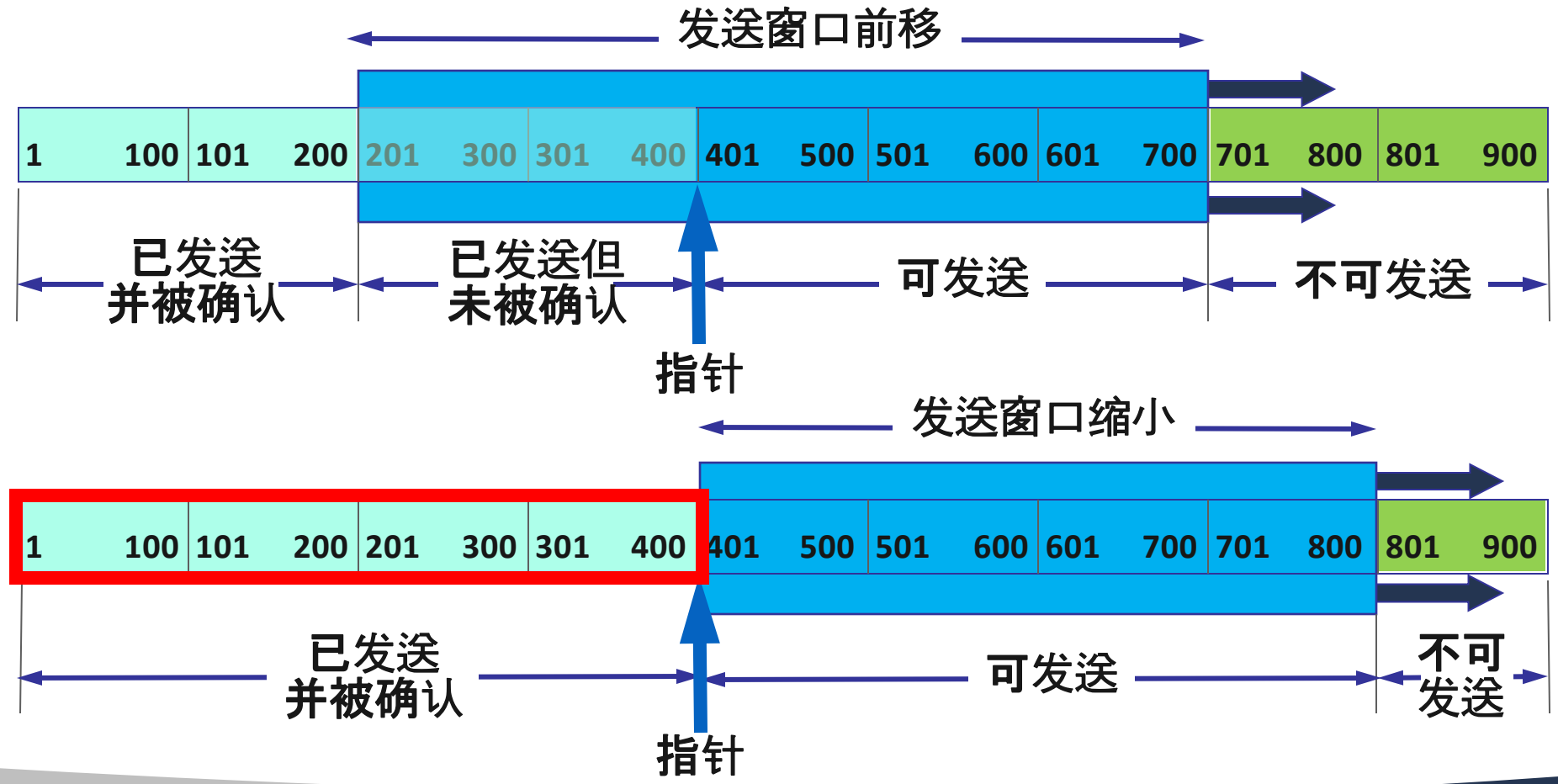
- 发送方已发送了 400 字节的数据，但只收到对前 200 字节数据的确认，同时窗口大小不变。



- 现在发送方还可发送 300 字节。

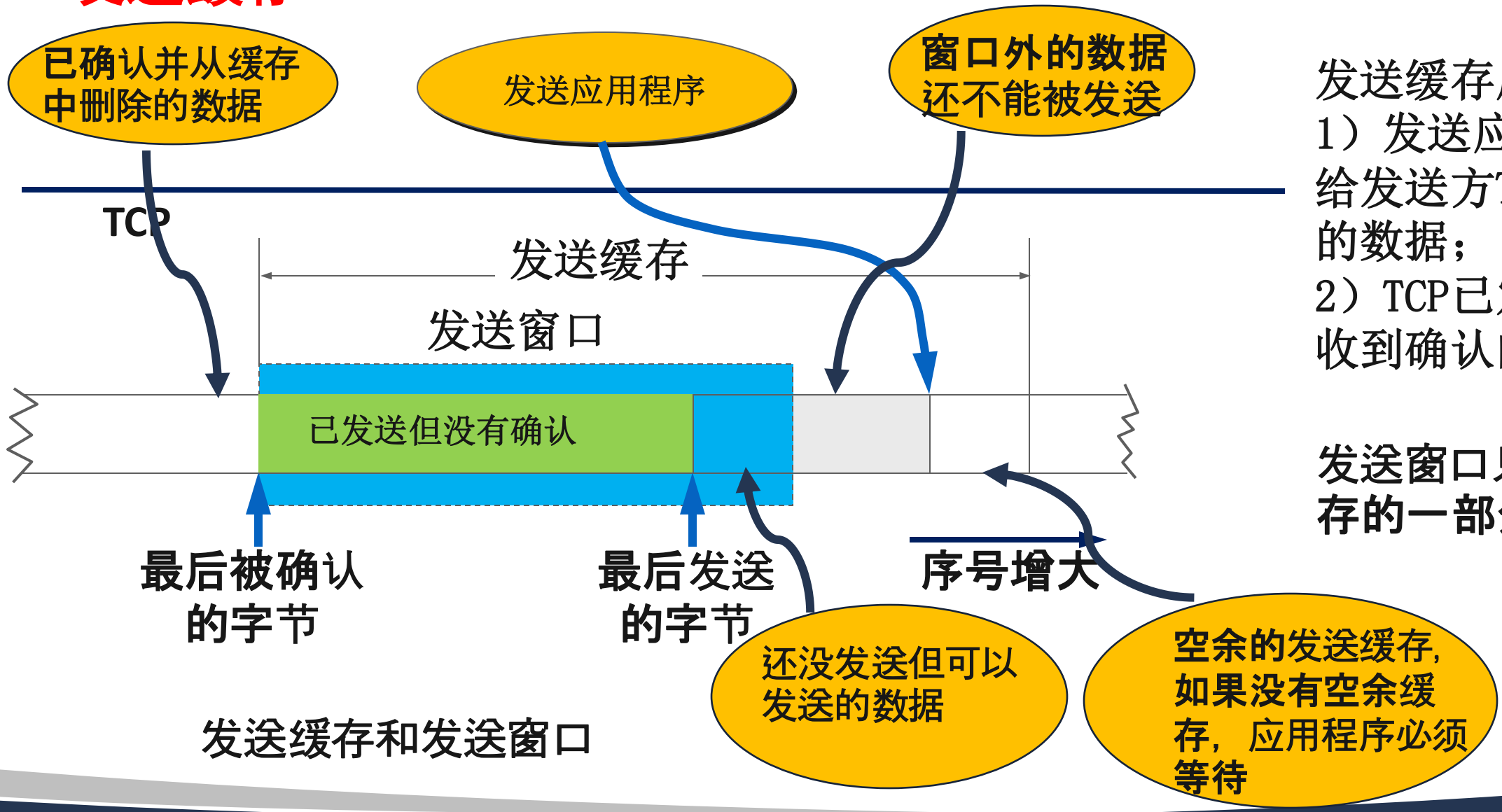
3.3.3 TCP的可靠传输

- 发送方收到了对方对前 400 字节数据的确认，但对方通知发送方必须把窗口减小到 400 字节。
- 现在发送方最多还可发送 400 字节的数据。



3.3.3 TCP的可靠传输

发送缓存



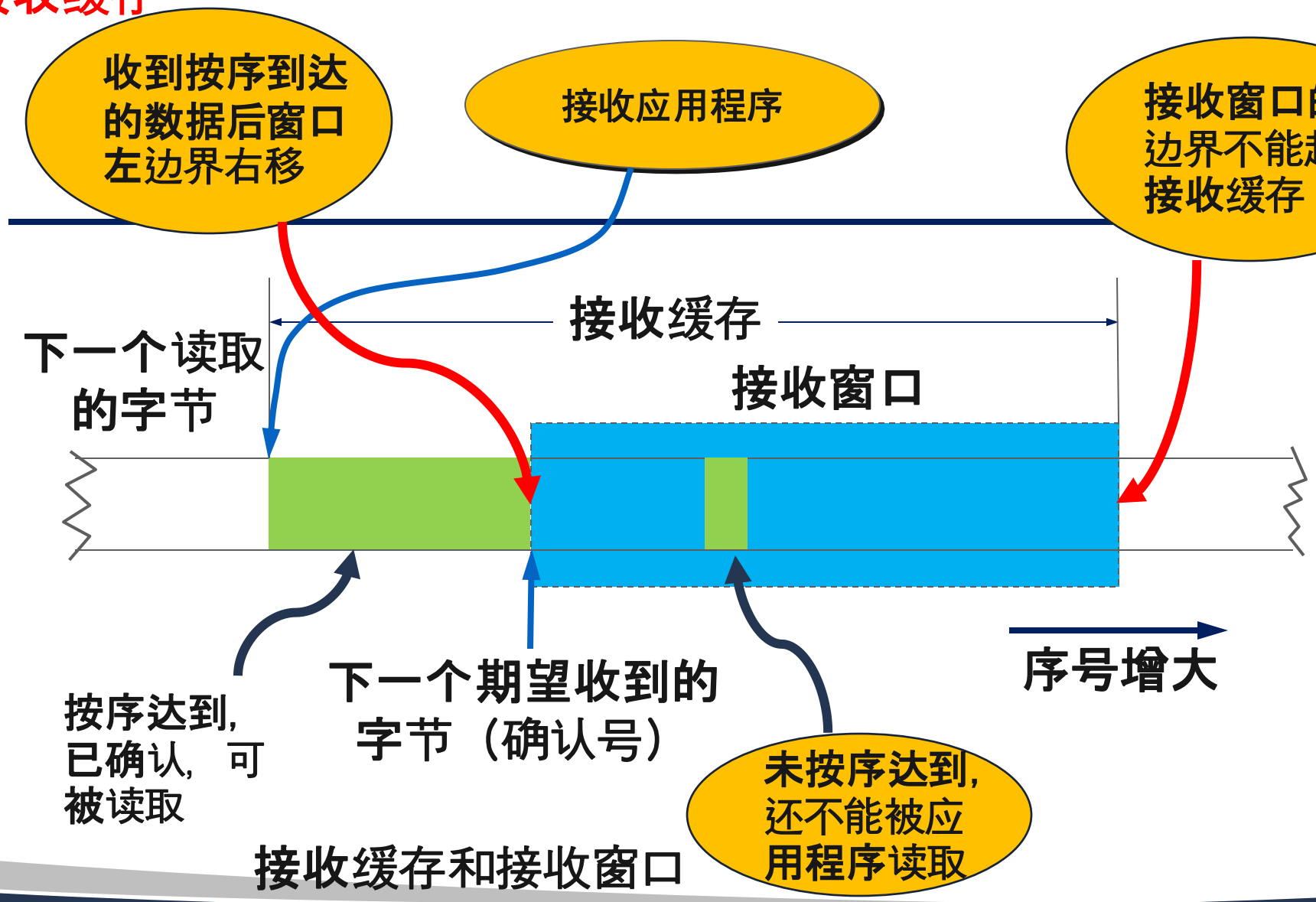
发送缓存和发送窗口

发送缓存用来存放：
1）发送应用程序传送给发送方TCP准备发送的数据；
2）TCP已发送出去但未收到确认的数据。

发送窗口只是发送缓存的一部分。

3.3.3 TCP的可靠传输

接收缓存



接收缓存用来存放：
1）按序到达，但未被接收应用程序读取的数据；
2）未按序到达，但还不能被接收应用程序读取的数据。

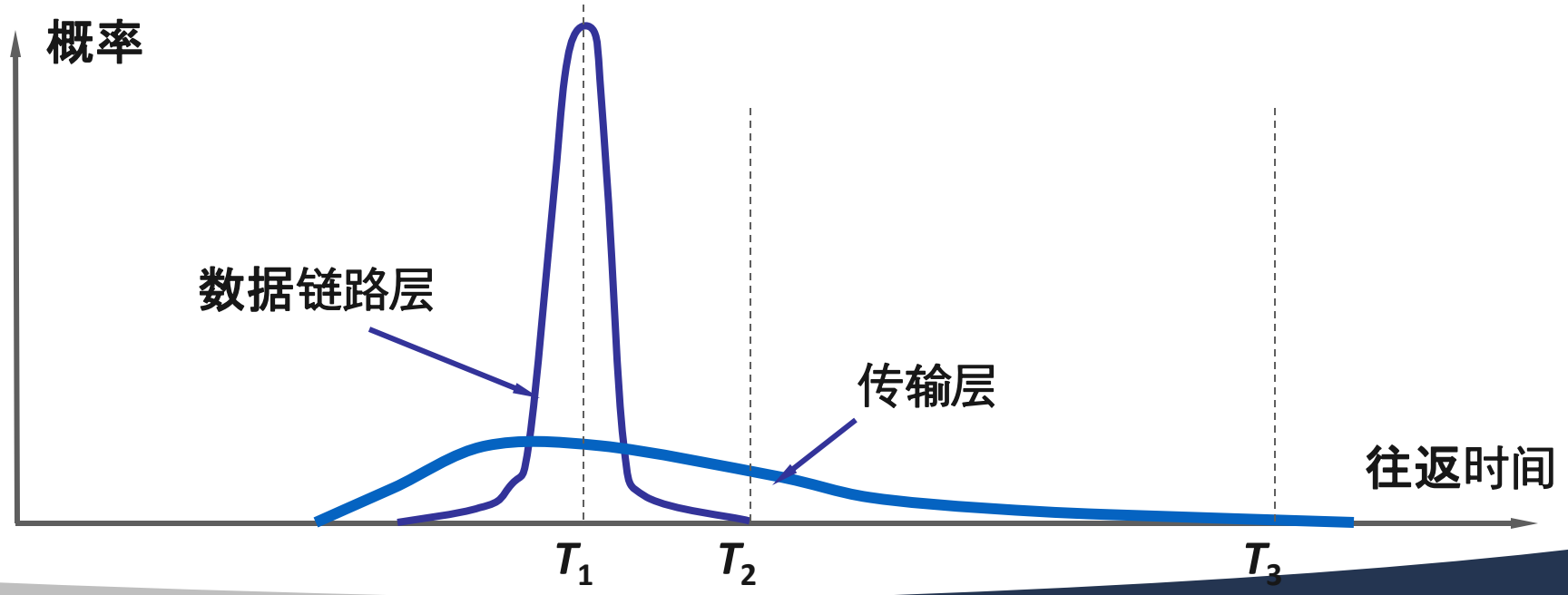
3.3.3 TCP的可靠传输

3) 超时重传时间的选择

- 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。
- 超时重传时间应该稍大于**往返时间RTT**

数字信号通过实际的信道

由于 TCP 的下层是一个互联网环境，IP 数据报所选择的路由变化很大。因而传输层的往返时间的方差也很大。



3.3.3 TCP的可靠传输

指数加权移动平均往返时间

- TCP 用**指数加权移动平均往返时间** RTT_S（这又称为**平滑的往返时间**）来估计当前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：

$$\text{新的 } RTT_S = (1-\alpha) \times (\text{旧的 } RTT_S) + \alpha \times (\text{新的 RTT 样本}) \qquad (3\text{-}1)$$

$$RTT_{S4} = (1-\alpha)^3 \times RTT_1 + (1-\alpha)^2 \times \alpha \times RTT_2 + (1-\alpha) \times \alpha \times RTT_3 + \alpha \times RTT_4$$

随着测量次数的递增，过去的RTT测量值的权重呈指数递减！

最近测量的RTT的权重最大。

- 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。

3.3.3 TCP的可靠传输

超时重传时间 RTO (RetransmissionTime-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：

$$RTO = RTT_S + 4 \times RTT_D \qquad (3\text{-}2)$$

- 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：

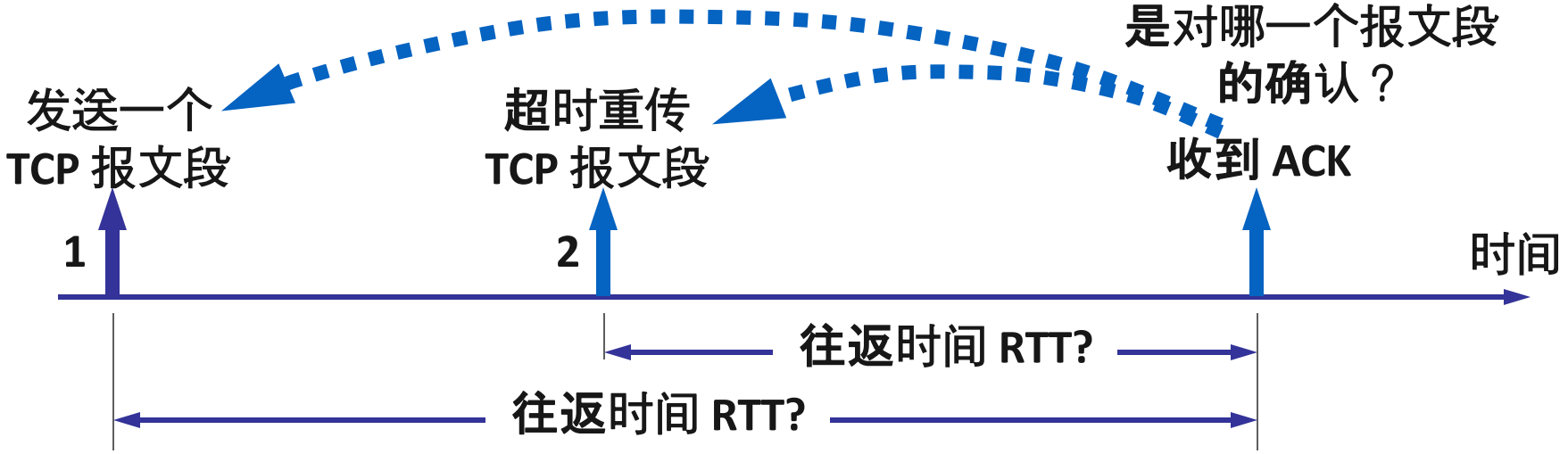
$$\text{新的 } RTT_D = (1 - \beta) \times (\text{旧的} RTT_D) + \beta \times |RTT_S - \text{新的 RTT 样本}| \qquad (3\text{-}3)$$

- β 是个小于 1 的系数，其推荐值是 1/4，即 0.25。

3.3.3 TCP的可靠传输

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？**



结论：如果是重传报文段2的确认，误认为是报文段1的确认，则计算的RTT_S和超时重传时间会偏大。

如果收到的确认是对报文段1的确认，但被误认为是重传报文段2的确认，则计算的RTT_S和超时重传时间会偏小。☞☞导致更多的报文段要重传！！

3.3.3 TCP的可靠传输

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平均往返时间 RTT_S 和超时重传时间 RTO 就较准确。

修正的 Karn 算法

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。

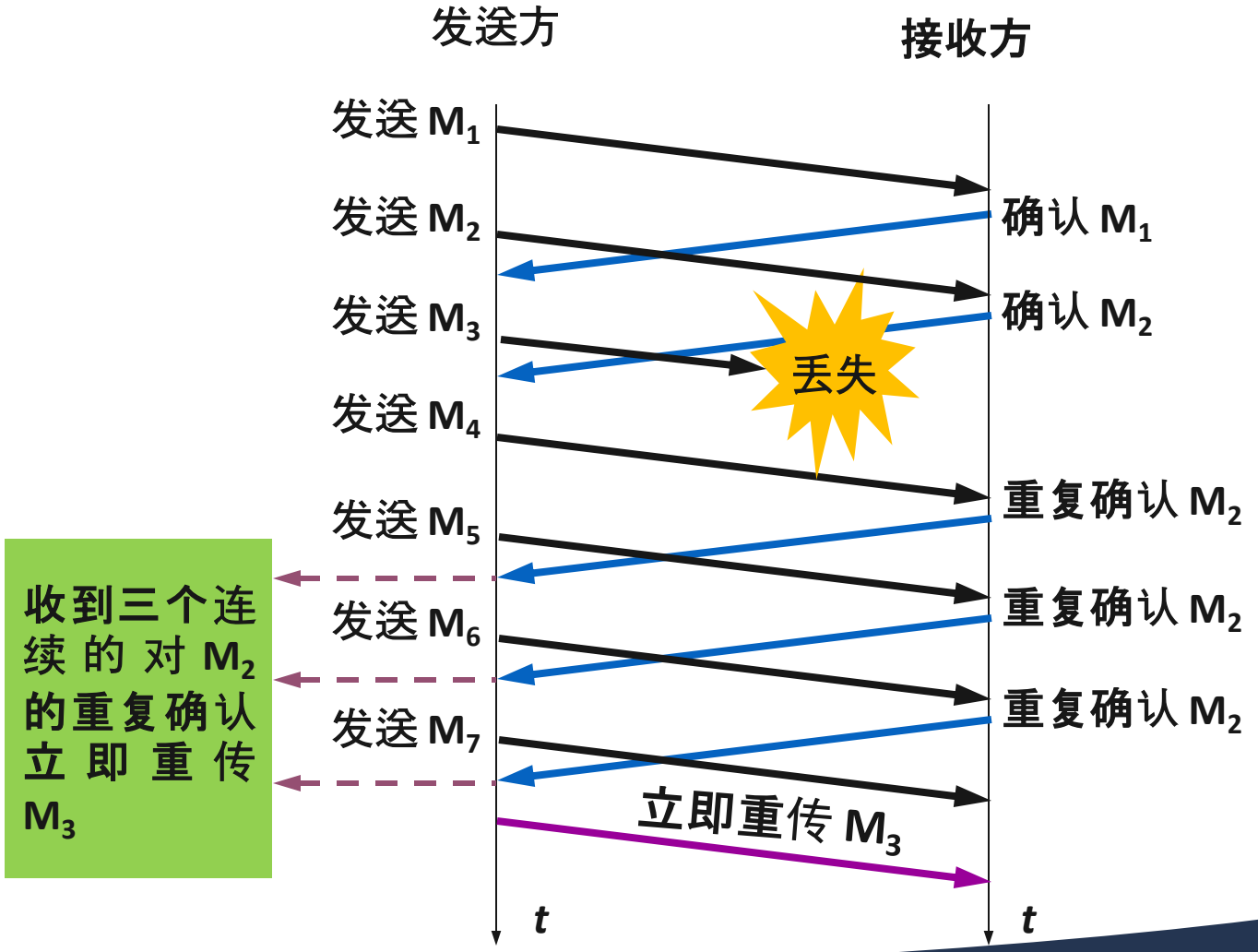
实践证明，这种策略较为合理。

3.3.3 TCP的可靠传输

4）快速重传

- 超时重传存在的问题：超时时间可能会比较长。
- 快要求接收方每收到一个失序的报文段后就立即发出重复确认。
- 发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段。
- 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。

快速重传举例

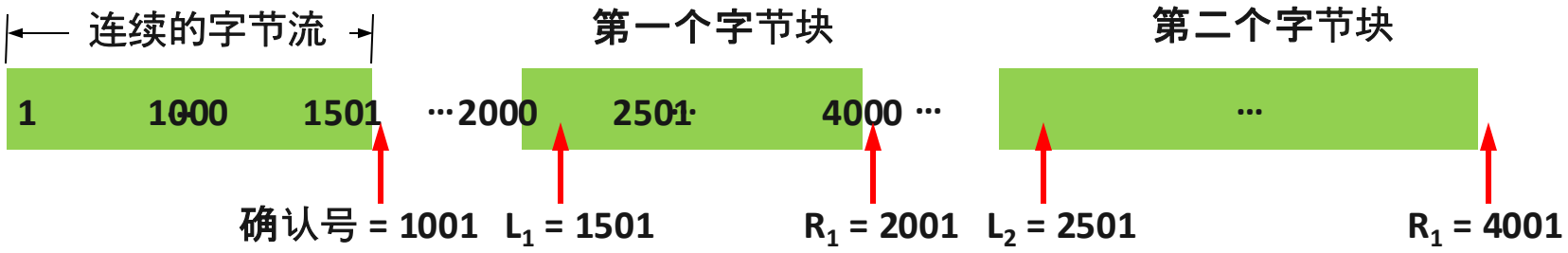


3.3.3 TCP的可靠传输

5) 选择确认

选择确认SACK (Selective ACK) 允许接收方通知发送方所有正确接收了的但是失序的字节块，发送方可以根据这些信息只重传那些接收方还没有收到的字节块。

- TCP在首部中提供了一个可变长的“SACK选项字段”来存放接收到的失序字节块的信息
- 在建立TCP连接时，通过添加“允许SACK选项字段”首部选项，表示都支持选择确认功能



3.3.3 TCP的可靠传输

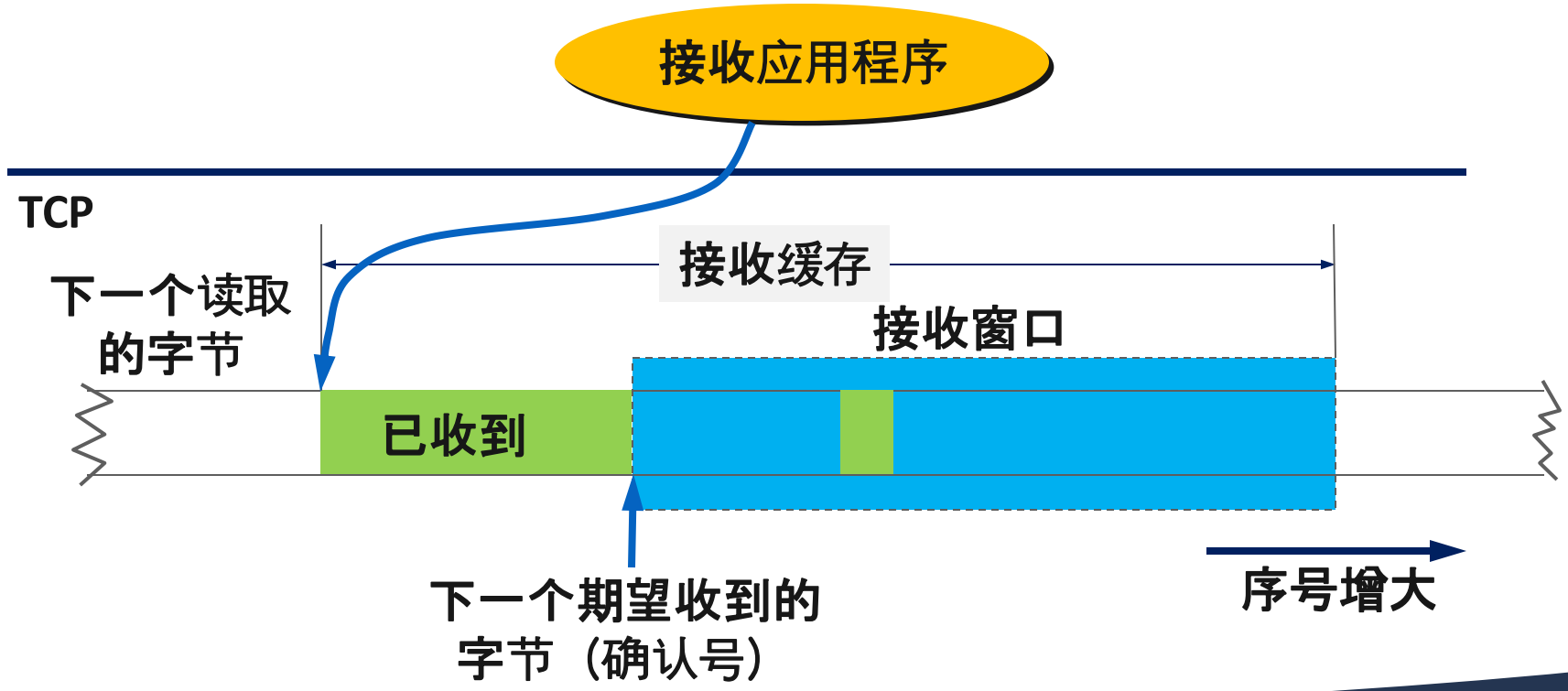
- TCP每发送一个报文段就设置一个超时计时器；
- TCP使用指数加权移动平均算法计算RTT及其偏差的估计值，并据此计算超时重传时间；
- 若超时还没有收到该报文段的确认，则重传该报文段；
- 对重传报文段不使用RTT测量值计算超时重传时间，而是直接对超时重传时间增大一倍；
- 如果连续收到某报文段的3个重复确认，则进行快速重传。

3.3.4 流量控制

- 如果发送方把数据发送得过快，接收方就可能来不及接收，这就会造成数据的丢失。
- **流量控制**(flow control)就是让发送方的发送速率不要太快，即要让接收方来得及接收。
- TCP的接收方将接收窗口大小及时通知发送方，发送方根据对方接收窗口大小调整自己的发送窗口大小。
- 利用**可变大小的滑动窗口机制**可以很方便地在 TCP 连接上实现**流量控制**。

接收缓存

发送方的发送窗口大小不能超过接收方的接收窗口的大小！



3.3.4 流量控制

初始：发送窗口=400， 接收窗口=400

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

假设：该报文段在传输过程中丢失

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

3.3.4 流量控制

接收方发送确认报文给发送方， ack=201,rwin=300
表示允许发送方再发送序号201-500的300字节数据

3.3.4 流量控制

发送窗口右移时同时减小了100个字节， 变成了300

3.3.4 流量控制

3.3.4 流量控制

接收方收到301-400，但还没有收到201-300，因此接收窗口不能右移

3.3.4 流量控制

发送方发送401-500的数据

3.3.4 流量控制

3.3.4 流量控制

发送方超时重传201-300的数据

3.3.4 流量控制

接收方接收到201-300的数据

3.3.4 流量控制

接收窗口右移，此时接收窗口已满，变为0字节

3.3.4 流量控制

应用程序从接收缓存中读取了100个字节， 101-200

3.3.4 流量控制

接收缓存右移，变成201-600；接收窗口右移，变成100

3.3.4 流量控制

3.3.4 流量控制

接收方发送确认， ack=501,rwin=100,
即允许发送方发送501-600的数据

3.3.4 流量控制

3.3.4 流量控制

发送方发送501-600的数据

3.3.4 流量控制

发送方指针右移

3.3.4 流量控制

接收方接收501-600字段的数据，接收窗口左边右移，接收窗口大小变为0

3.3.4 流量控制

3.3.4 流量控制

接收方给发送方发送确认报文， ack=601,rwin=0

3.3.4 流量控制

此时， 发送窗口和接收窗口都为0

3.3.4 流量控制

3.3.4 流量控制

使用了流量控制机制控制了发送方的发送速率，从而保证了接收方缓存不会溢出。

每段100字节，初始窗口和接收缓存为400

3.3.4 流量控制

总结：

1）接收方根据空余的接收缓存大小调整接收窗口的大小；

2）接收方在确认对方数据时将自己的接收窗口大小通知对方；

3）发送方根据收到报文段中确认号和窗口字段调整自己的发送窗口；

4）由于发送方的发送窗口大小总是由接收方的接收窗口大小控制，即接收方通过接收窗口确定发送方的发送速率，保证了接收缓存不会溢出。

3.3.5 TCP的连接管理

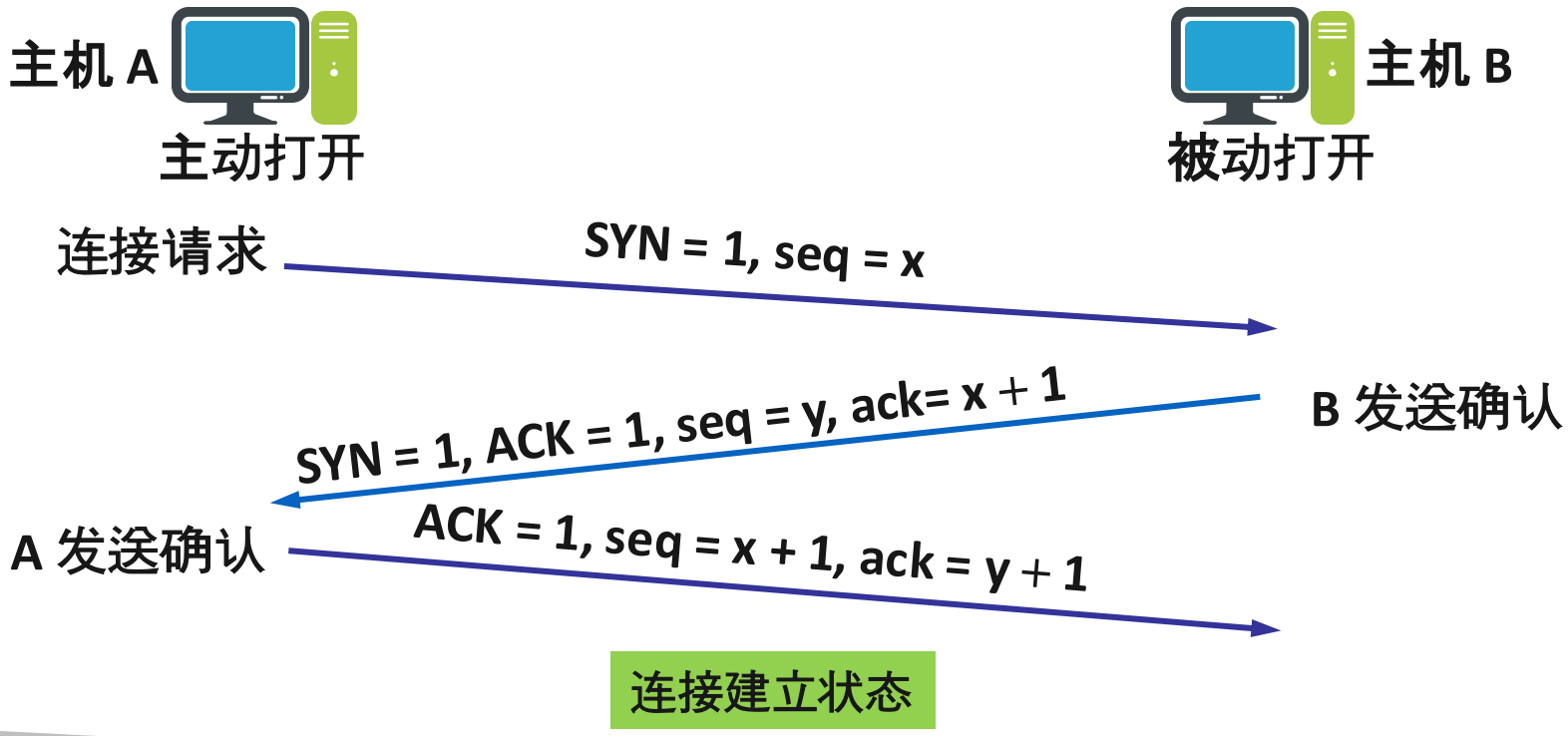
1) TCP的连接建立

- 运输连接就有三个阶段，即：连接建立、数据传送和连接释放。运输连接的管理就是使运输连接的建立和释放都能正常地进行。
- 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，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。

3.3.5 TCP的连接管理

- TCP 的连接和建立都是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)。

用三次握手建立 TCP 连接



3.3.5 TCP的连接管理

- A 的 TCP 向 B 发出连接请求报文段，其首部中的同步位 SYN = 1，并选择序号 x，表明下一个报文段的第一个数据字节的序号是 x + 1。
- B 的 TCP 收到连接请求报文段后，如同意，则发回确认，在确认报文段中使 SYN = 1 和 ACK = 1，其确认号应为 ack = x + 1，并选择序号seq = y。
- A 收到此报文段后，向 B 给出确认，其 ACK = 1，序号应为 seq = x + 1，确认号应为 ack = y + 1。
- A 的 TCP 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 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。

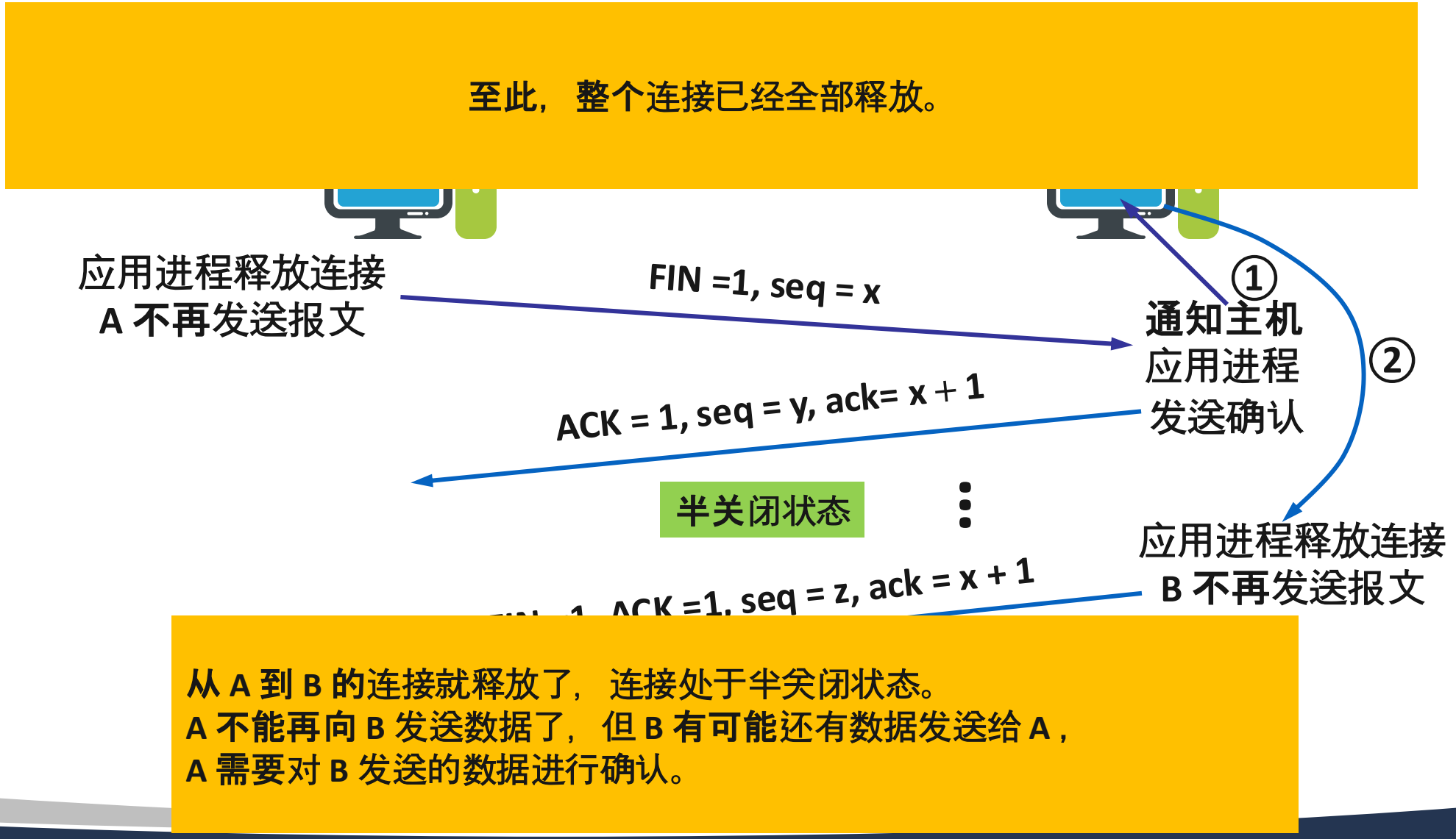
3.3.5 TCP的连接管理

三次握手或三次联络 (three-way handshake)

- 防止已失效的连接请求报文段又传送到B，因而产生错误。
- A 发出连接请求，但因未收到确认而再重传一次。后来收到了确认，建立了连接。数据传输完毕后释放了连接。A 共发送了两个连接请求报文段，其中的第二个到达了 B。
- A 发出的第一个连接请求报文段以后又传送到 B。B 误认为是 A 又发出一次新的连接请求。于是就向 A 发出确认报文段，同意建立连接。
- A 不会理睬 B 的确认 。但 B 却以为运输连接就这样建立了，并一直等待 A 发来数据。 B 的许多资源就这样白白浪费了。

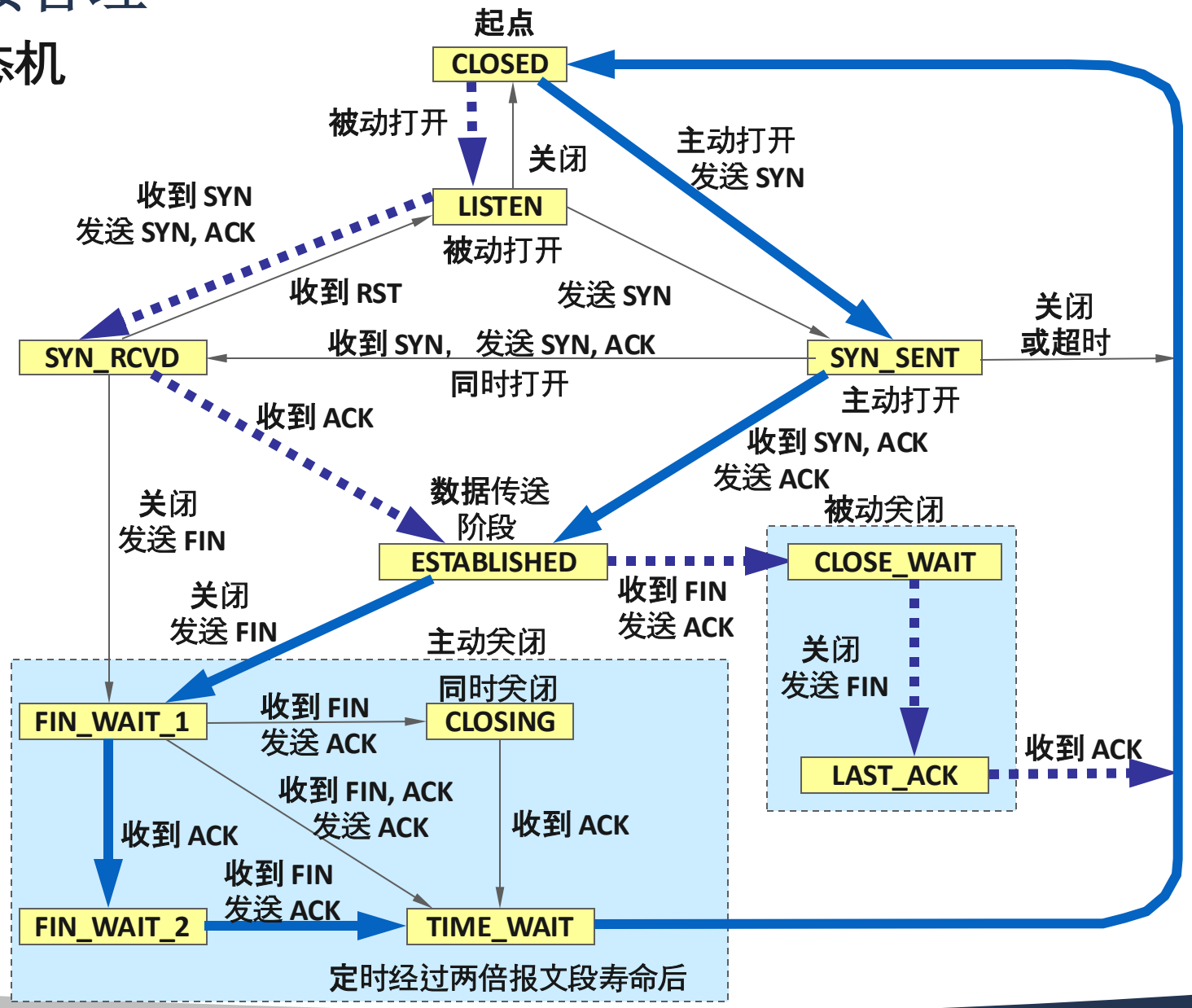
3.3.5 TCP的连接管理

2）TCP的连接释放



3.3.5 TCP的连接管理

3）TCP的有限状态机



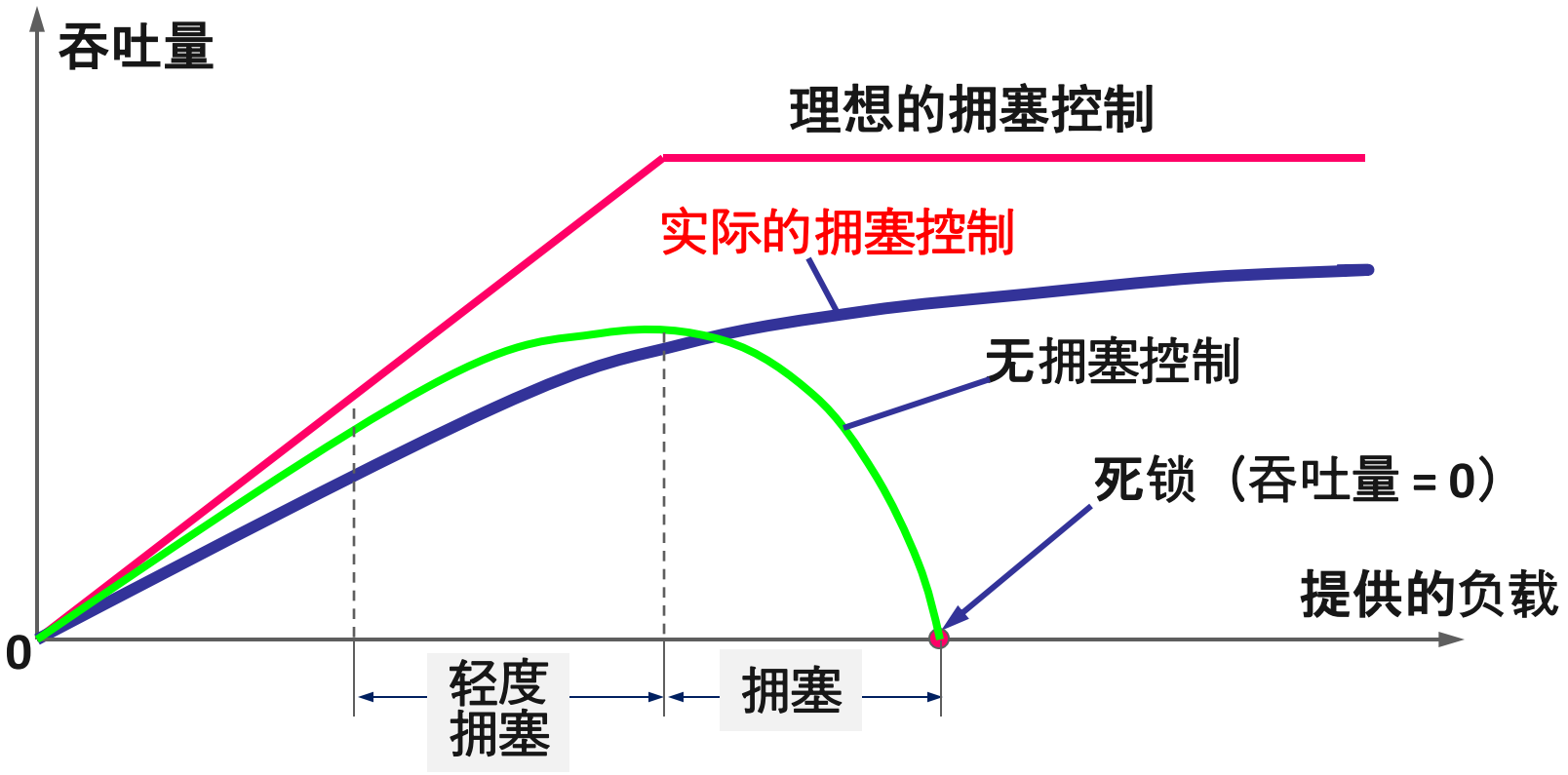
第四部分 ▶

拥塞控制

3.4 拥塞控制

- 如果网络中的负载(load)，即发送到网络中的分组数量，超过了网络的容量，即网络中能处理的分组数量，那么在网络中就会发生拥塞(congestion)。
- 所谓拥塞控制(congestion control)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。

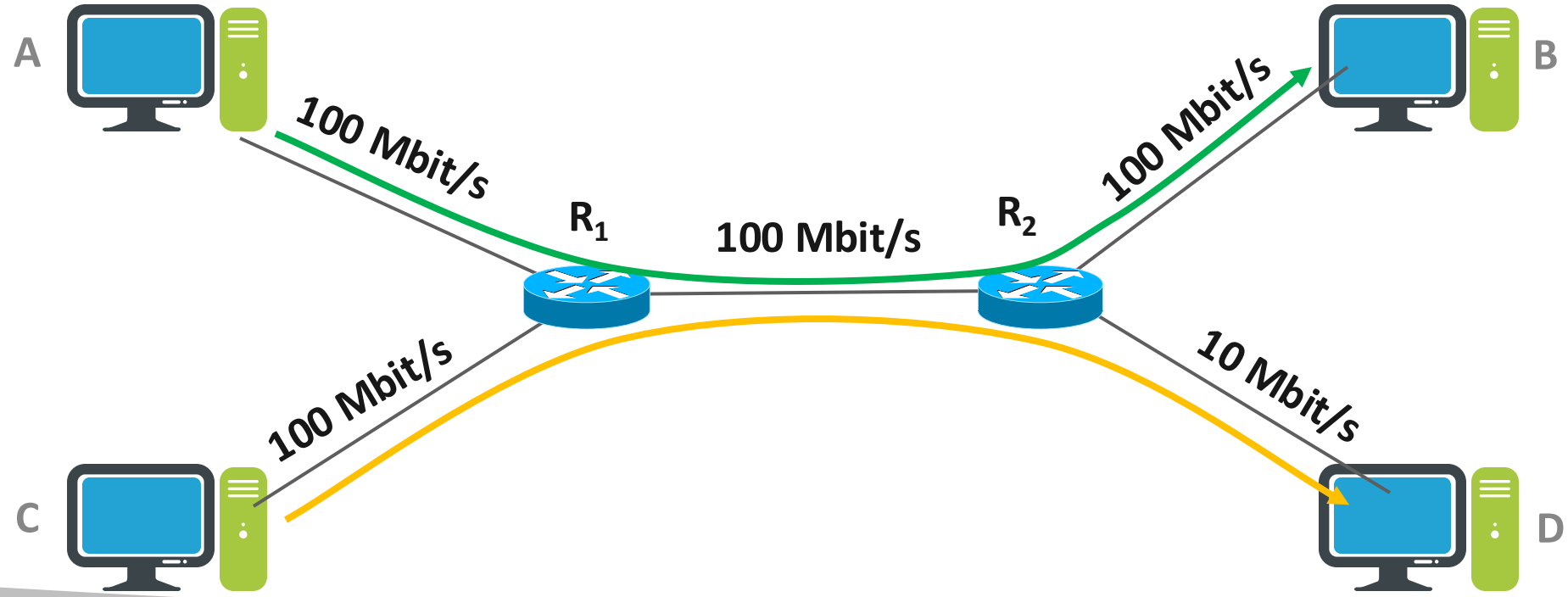
3.4.1 拥塞的原因与危害



3.4.1 拥塞的原因与危害

举例

- 理想吞吐量为100 Mb/s
- 不加任何控制只能达到60 Mb/s
- 当分组丢失时, 任何用于传输该分组的上游传输能力都被浪费!



3.4.2 拥塞控制的基本方法

拥塞控制与流量控制的区别

拥塞控制是一个全局性的过程，涉及到所有的主机、所有的路由器，以及与降低网络传输性能有关的所有因素。

流量控制往往指在给定的发送端和接收端之间的点对点通信量的控制。流量控制所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。

3.4.2 拥塞控制的基本方法

开环控制和闭环控制

- **开环控制方法就是在**设计网络时事先将有关发生拥塞的因素考虑周到，力求网络在工作时不产生拥塞。
- **闭环控制是基于反馈环路的概念。属于闭环控制的有以下几种措施：**
  - 监测网络系统以便检测到拥塞在何时、何处发生。
  - 将拥塞发生的信息传送到可采取行动的地方。
  - 调整网络系统的运行以解决出现的问题。

3.4.2 拥塞控制的基本方法

显式反馈和隐式反馈

- 根据拥塞反馈信息的形式又可以将闭环拥塞控制算法分为显式反馈算法和隐式反馈算法。
- 在显式反馈算法中，从拥塞点（即路由器）向源端提供关于网络中拥塞状态的显式反馈信息。
- 在隐式反馈算法中，源端通过对网络行为的观察（如分组丢失与往返时延）来推断网络是否发生了拥塞。
- TCP采用的就是隐式反馈算法。

3.4.3 TCP的拥塞控制

- 发送方维持一个叫做拥塞窗口 cwnd (congestion window)的状态变量。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞窗口。如再考虑到接收方的接收能力，则发送窗口还可能小于拥塞窗口。
- 发送方控制拥塞窗口的原则是：只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。

3.4.3 TCP的拥塞控制

接收方窗口 rwnd　接收方根据其目前的接收缓存大小所许诺的最新的窗口值，是来自接收方的流量控制。接收方将此窗口值放在 TCP 报文的首部中的窗口字段，传送给发送方。

拥塞窗口 cwnd（congestion window）　是发送方根据自己估计的网络拥塞程度而设置的窗口值，是来自发送方的流量控制。

3.4.3 TCP的拥塞控制

发送窗口的上限值

- 发送方的发送窗口的上限值应当取为接收方窗口 rwnd 和拥塞窗口 cwnd 这两个变量中较小的一个，即应按以下公式确定：

$$\text{发送窗口的上限值} = \text{Min}\,[\text{rwnd}, \text{cwnd}] \qquad (3\text{-}4)$$

- 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，则是网络的拥塞限制发送窗口的最大值。

3.4.3 TCP的拥塞控制

1) 慢启动和拥塞避免

- 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方的拥塞窗口 cwnd，可以使分组注入到网络的速率更加合理。

3.4.3 TCP的拥塞控制

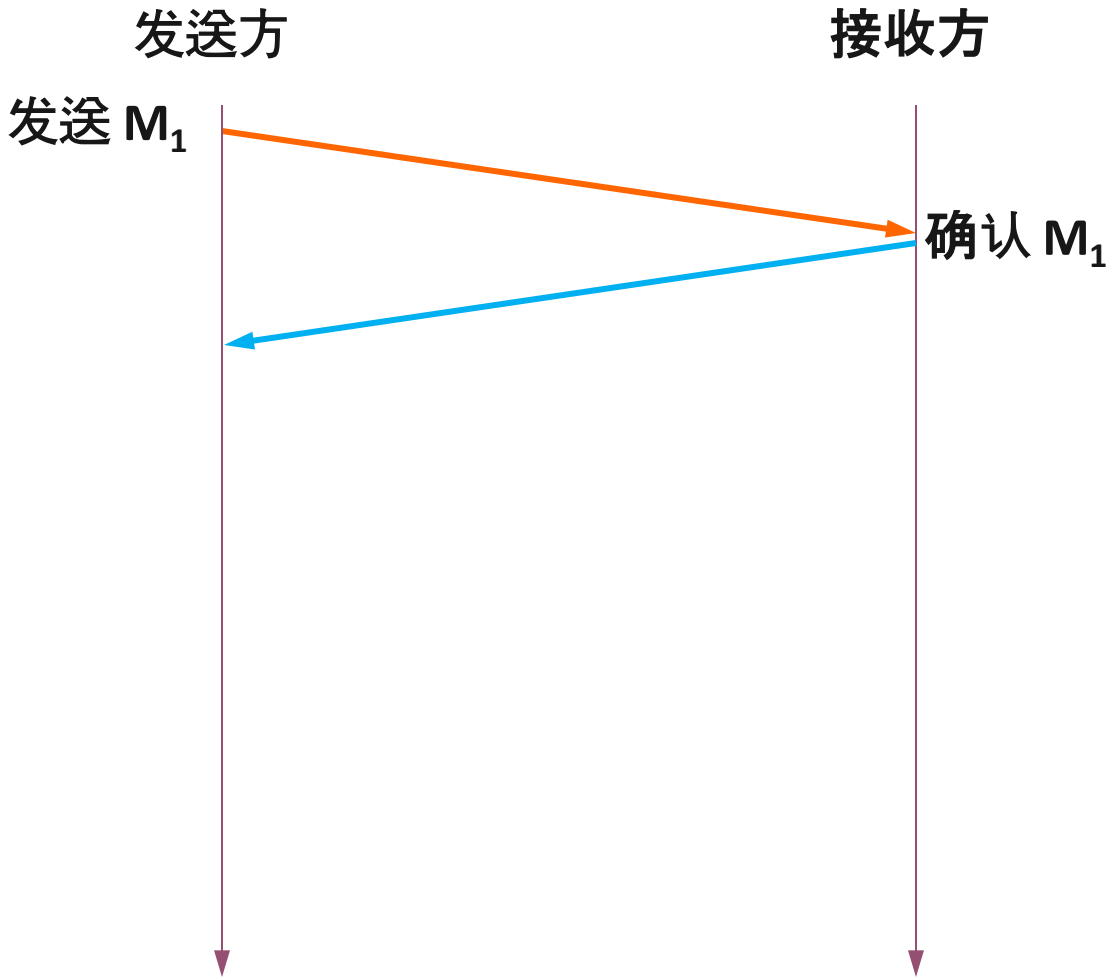
举例

- 用报文段的个数作为窗口大小的单位。还假定接收方窗口 rwnd 足够大，因此发送窗口只受发送方的拥塞窗口的制约。
- 发送方先设置 cwnd = 1，发送 M_0，接收方收到后发回 ACK_1。
- 发送方收到 ACK_1 后，把 cwnd 从 1 增大到 2，发送方接着发送 M_1 和 M_2 两个报文段。
- 接收方收到后发回 ACK_2 和 ACK_3。
- 发送方每收到一个对新报文段的确认 ACK，就使发送方的拥塞窗口加 1，因此现在发送方的 cwnd 又从 2 增大到 4，并可发送 M_3 ~ M_6 共 4 个报文段。

3.4.3 TCP的拥塞控制

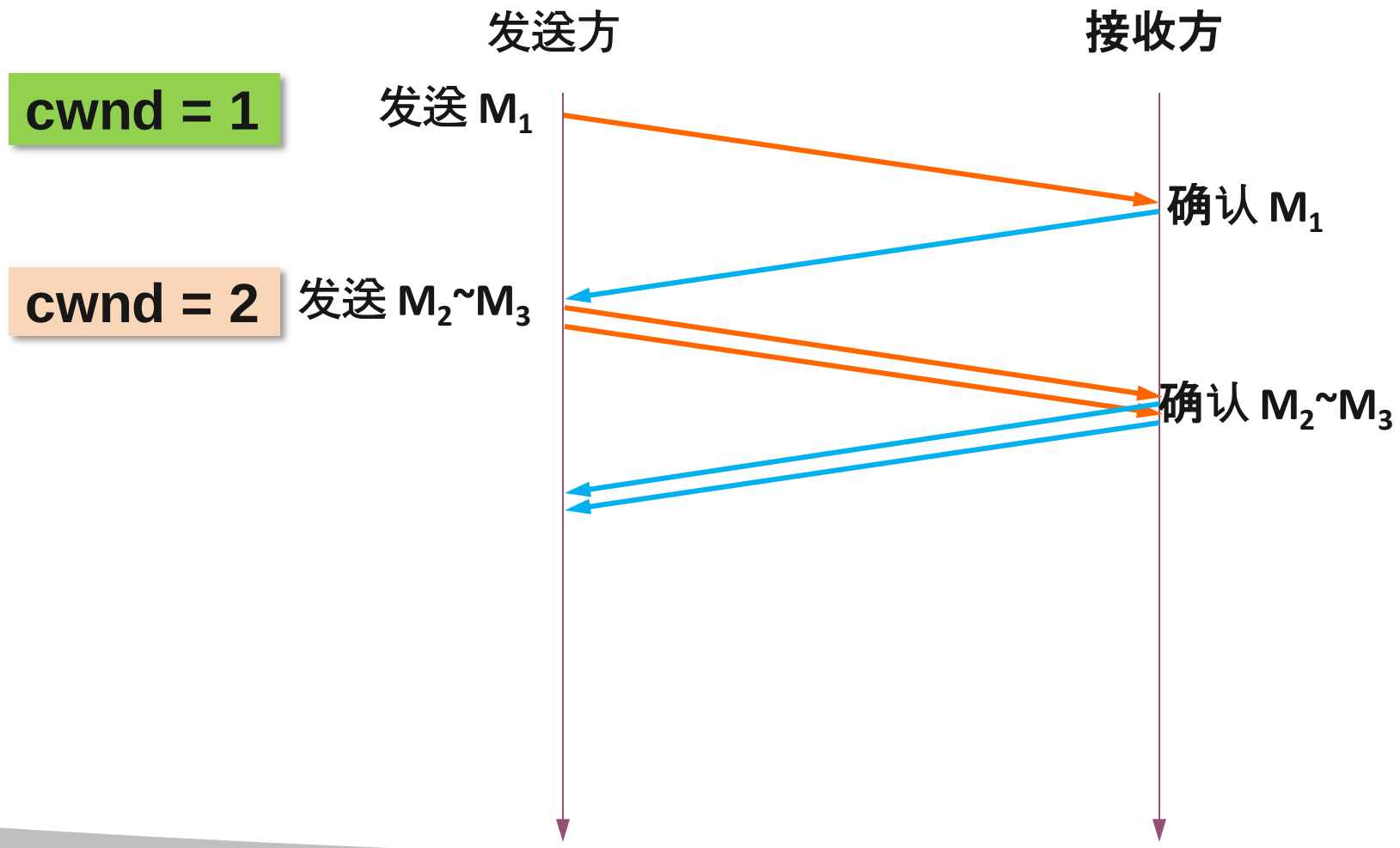
发送方每收到一个对新报文段的确认（重传的不算在内）就使 cwnd 加 1。

cwnd = 1



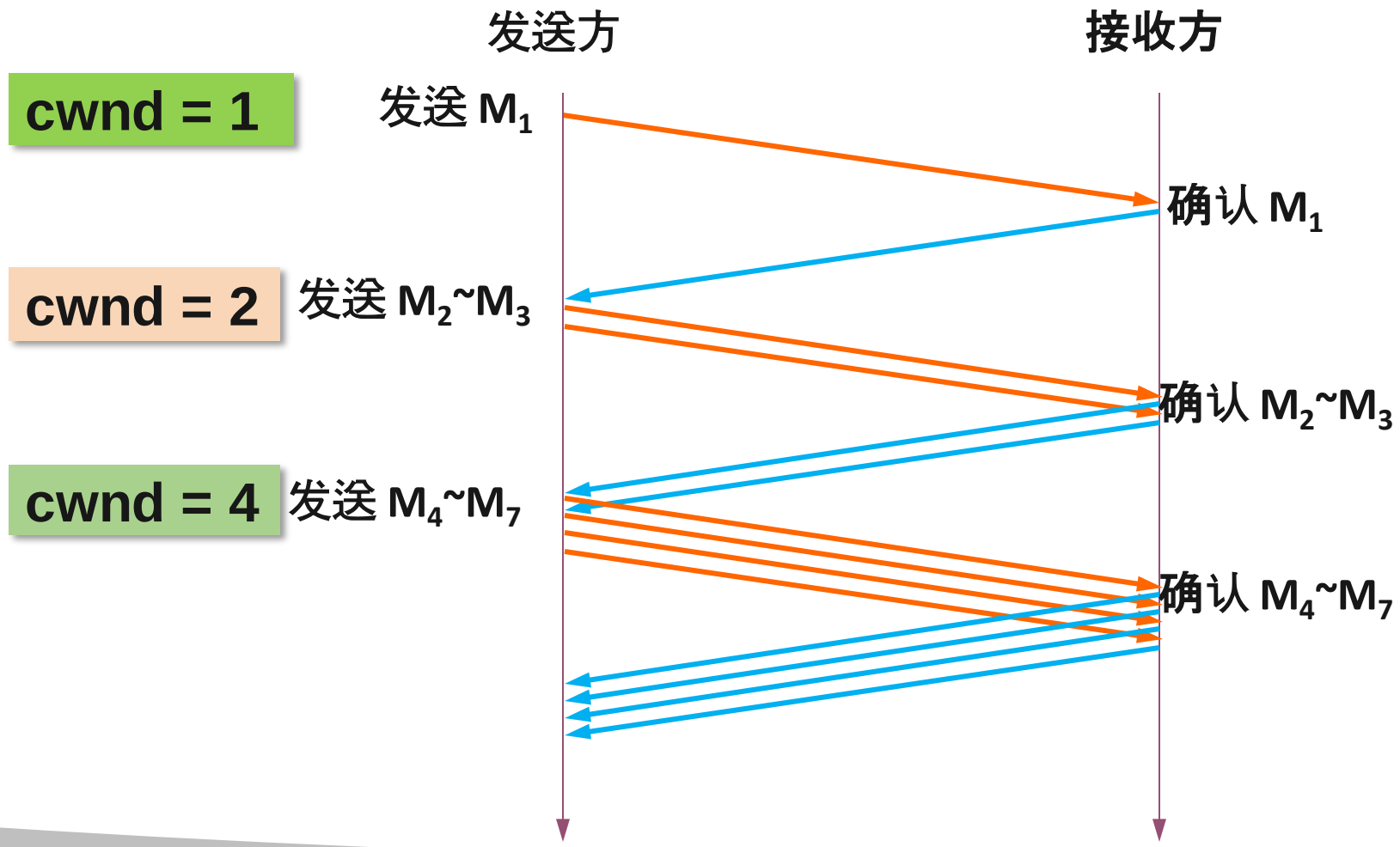
3.4.3 TCP的拥塞控制

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。



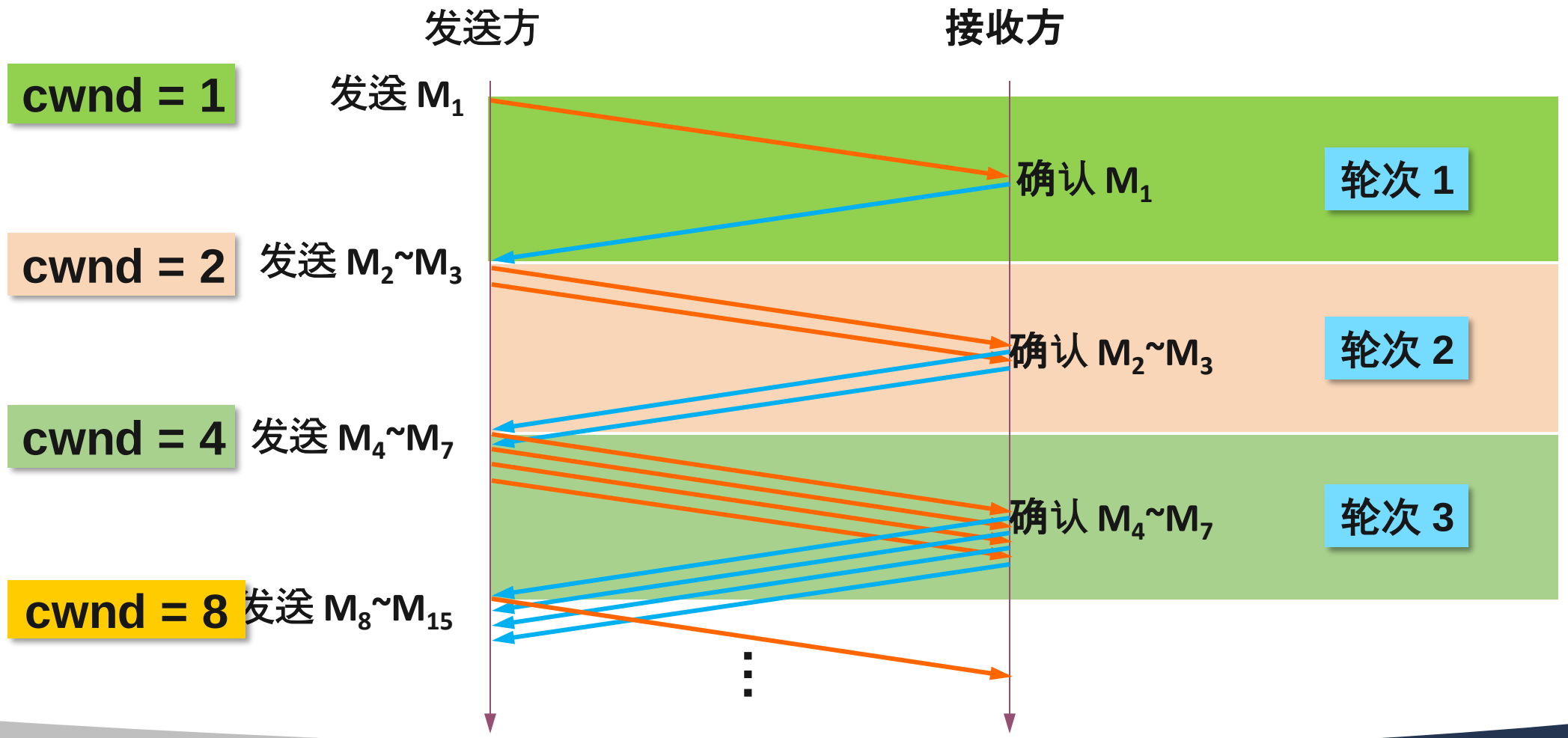
3.4.3 TCP的拥塞控制

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。



3.4.3 TCP的拥塞控制

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。



3.4.3 TCP的拥塞控制

慢启动的作用

- 可见慢启动的“慢”并不是指 cwnd 的增长速率慢，而是指在开始时发送速率“慢”（cwnd = 1）。
- 使用慢启动算法可以使发送方在开始发送时向网络注入的分组数大大减少。这对防止网络出现拥塞是非个常有力的措施。

3.4.3 TCP的拥塞控制

开始门限 ssthresh

- 为了防止拥塞窗口 cwnd 的增长引起网络拥塞，还需要另一个状态变量，即慢启动门限 ssthresh，其用法如下：
  - 当 cwnd < ssthresh 时，使用上述的慢启动算法。
  - 当 cwnd > ssthresh 时，停止使用慢启动算法而改用拥塞避免算法。
  - 当 cwnd = ssthresh 时，既可使用慢启动算法，也可使用拥塞避免算法。

3.4.3 TCP的拥塞控制

拥塞避免算法

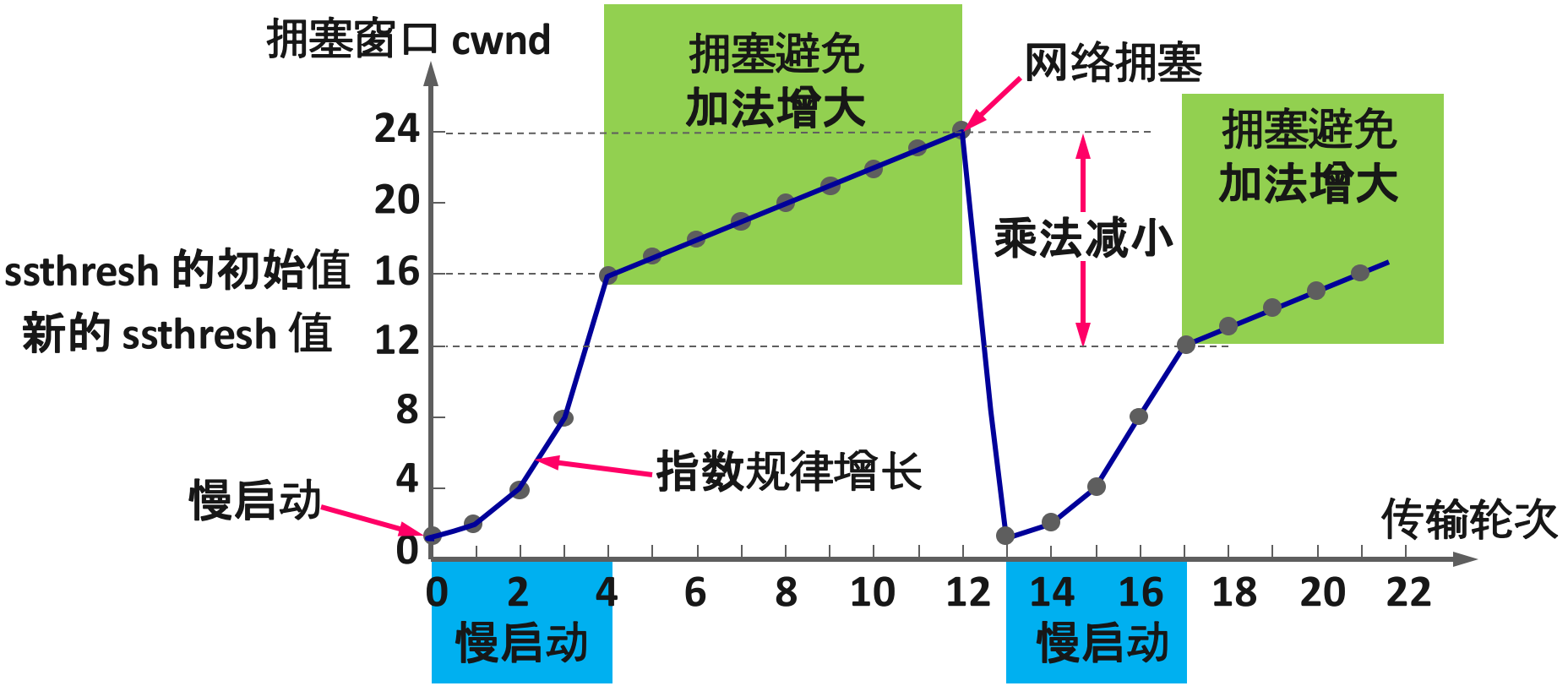
- 拥塞避免算法使发送方的拥塞窗口 cwnd每经过一个往返时延 RTT 就增加一个MSS 的大小（而不管在时间 RTT 内收到了几个 ACK）。
- 拥塞窗口 cwnd 按线性规律缓慢增长，比慢启动算法的拥塞窗口增长速率缓慢得多。

3.4.3 TCP的拥塞控制

当网络出现拥塞时

- 随着 cwnd增大，发送方的发送速率会超过网络可用带宽，导致分组丢失，即出现网络拥塞。
- 为是网络迅速从拥塞状态恢复到正常，发送方把拥塞窗口 cwnd 重新设置为 1，慢启动门限 ssthresh 设置为出现拥塞时的发送窗口值的一半，又执行慢开始算法。
- 这样做的目的就是要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。

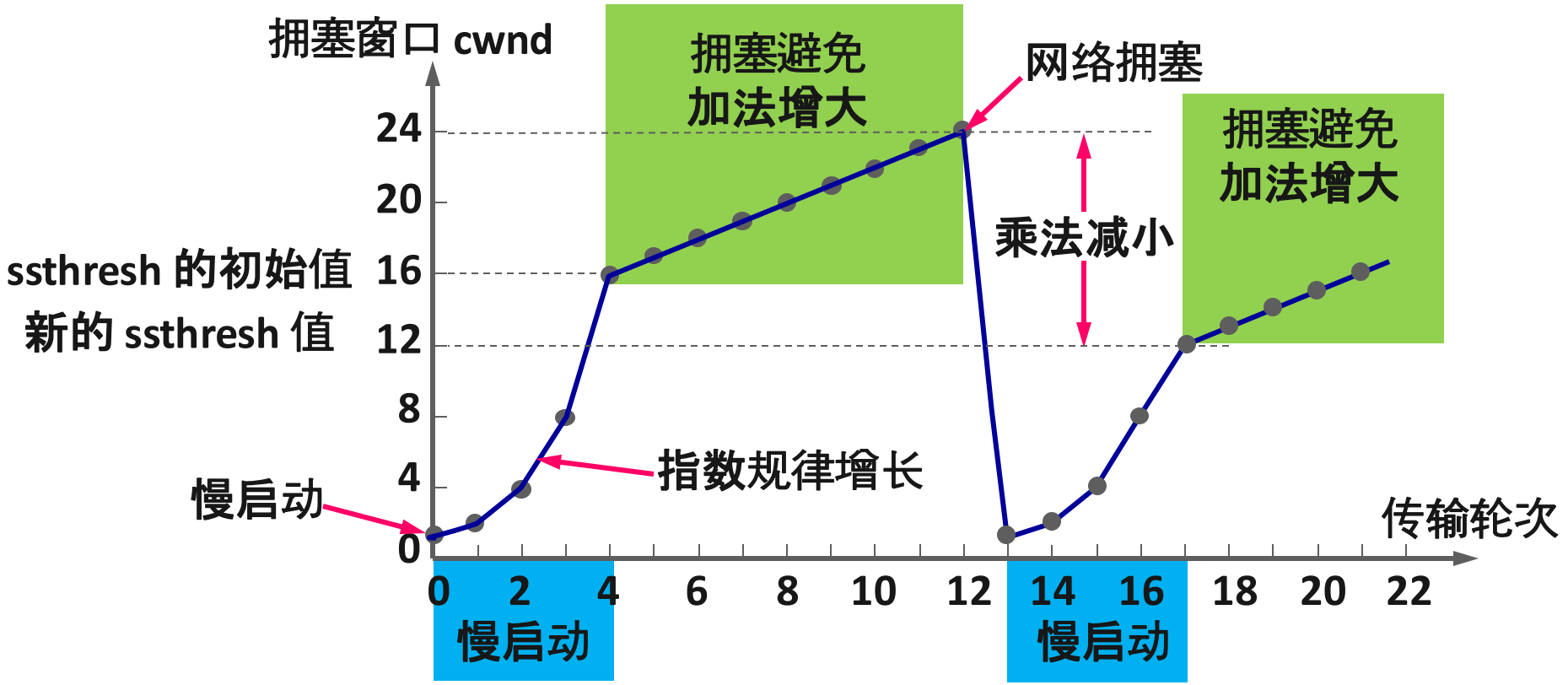
3.4.3 TCP的拥塞控制



当 TCP 连接进行初始化时，将拥塞窗口置为 1。为便于理解，图中的窗口单位不使用字节而使用报文段。

慢启动门限的初始值设置为 16 个报文段，即 ssthresh = 16。

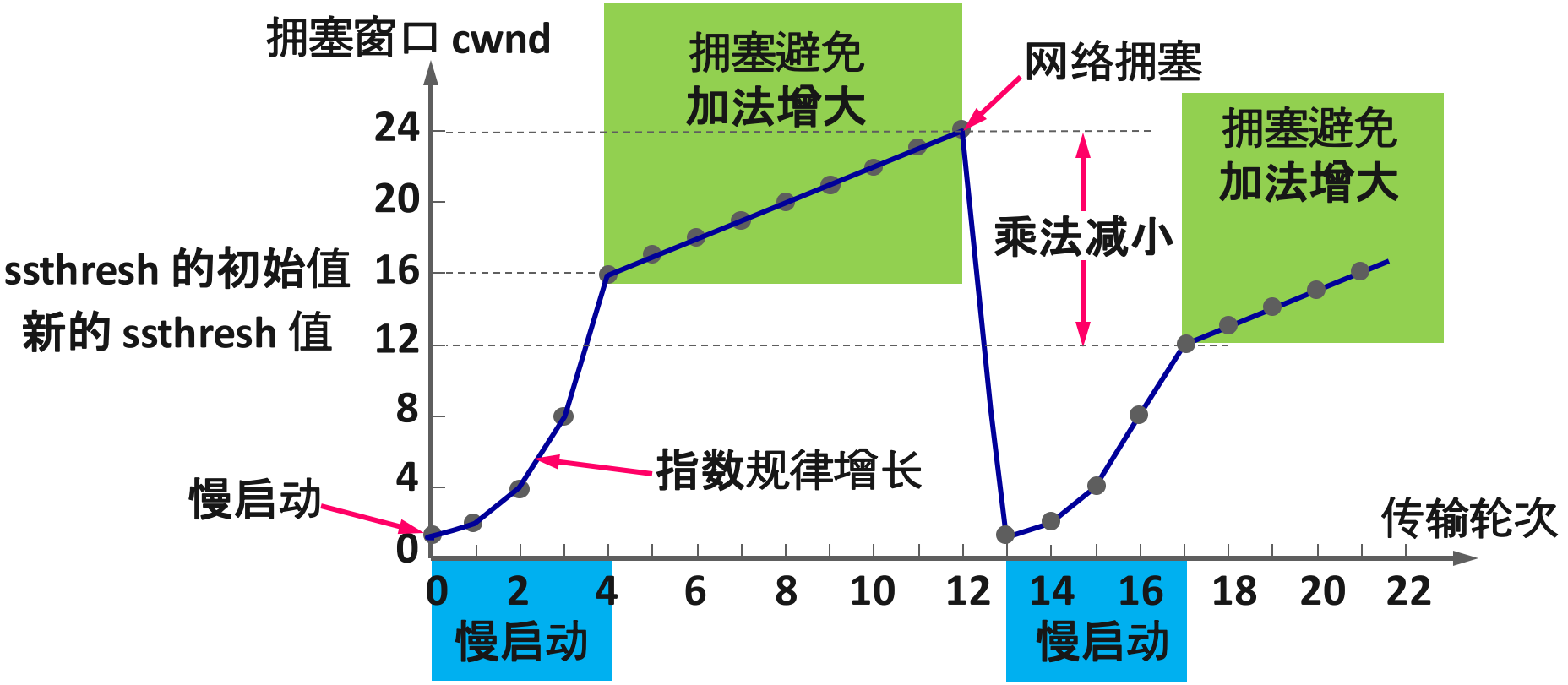
3.4.3 TCP的拥塞控制



请注意横坐标的单位——传输轮次。

所谓“轮次”就是把拥塞窗口 cwnd 所允许发送的报文段都发送出去，并且都收到了对方的确认。“传输轮次”的时间并不是固定不变的。

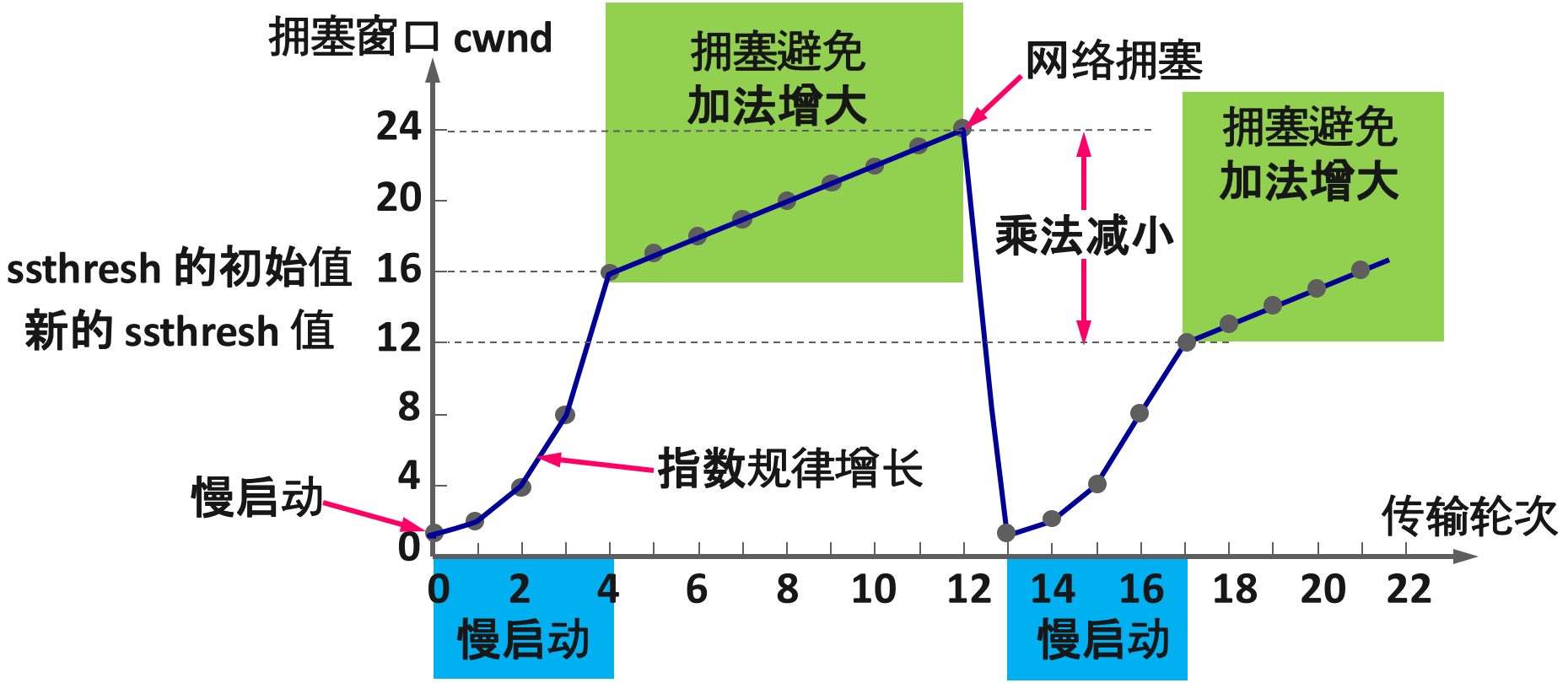
3.4.3 TCP的拥塞控制



发送方的发送窗口不能超过拥塞窗口 cwnd 和接收方窗口 rwnd 中的最小值。

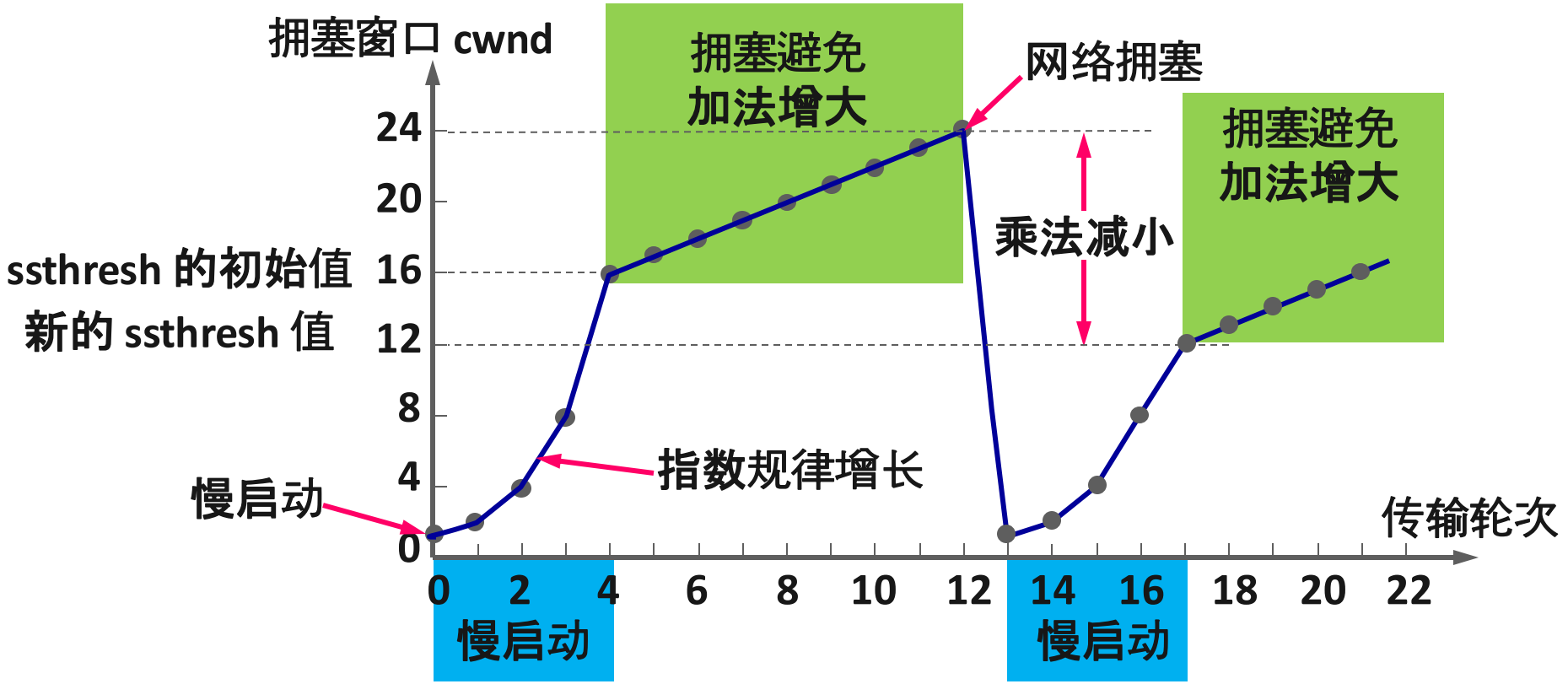
我们假定接收方窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值。

3.4.3 TCP的拥塞控制



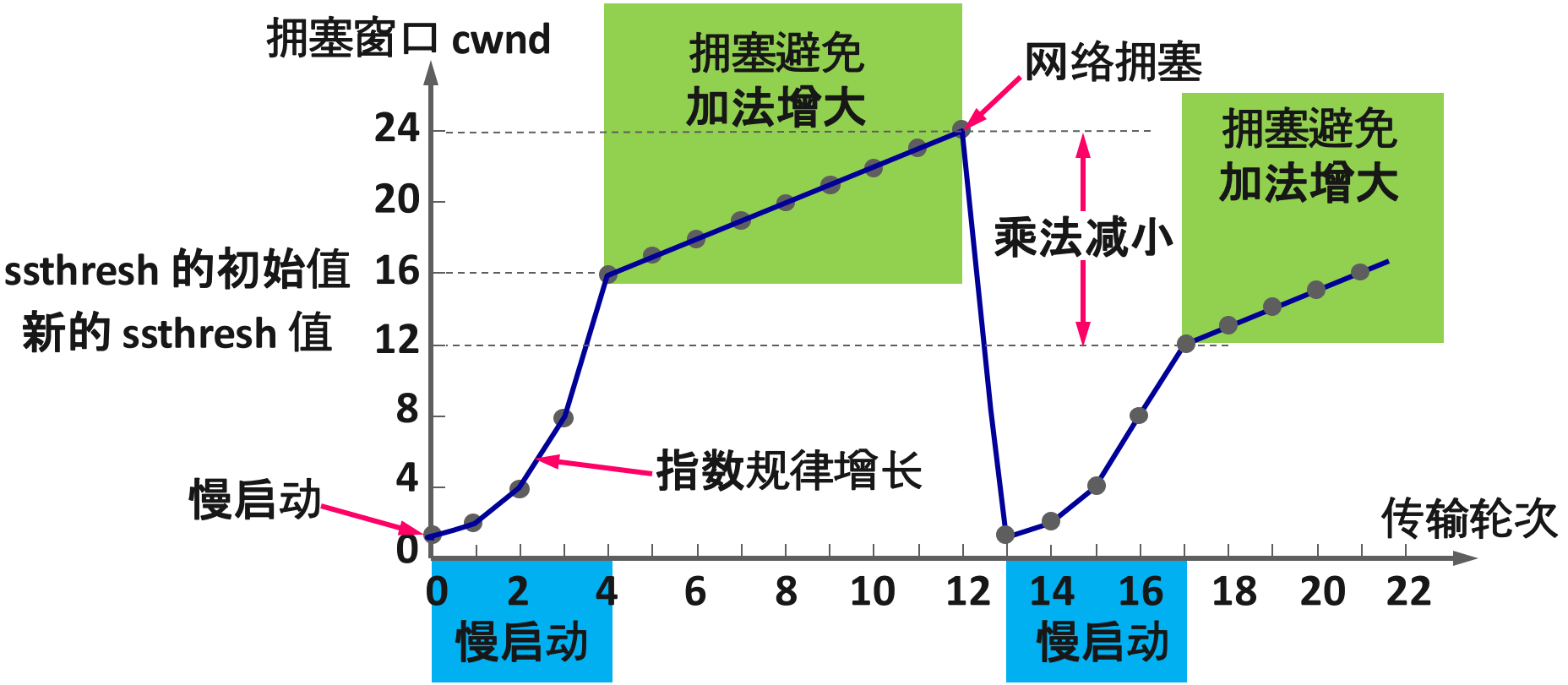
在执行慢启动算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 M_0。以后发送方每收到一个对新报文段的确认，就将发送方的拥塞窗口加 1，然后开始下一次的传输。

3.4.3 TCP的拥塞控制



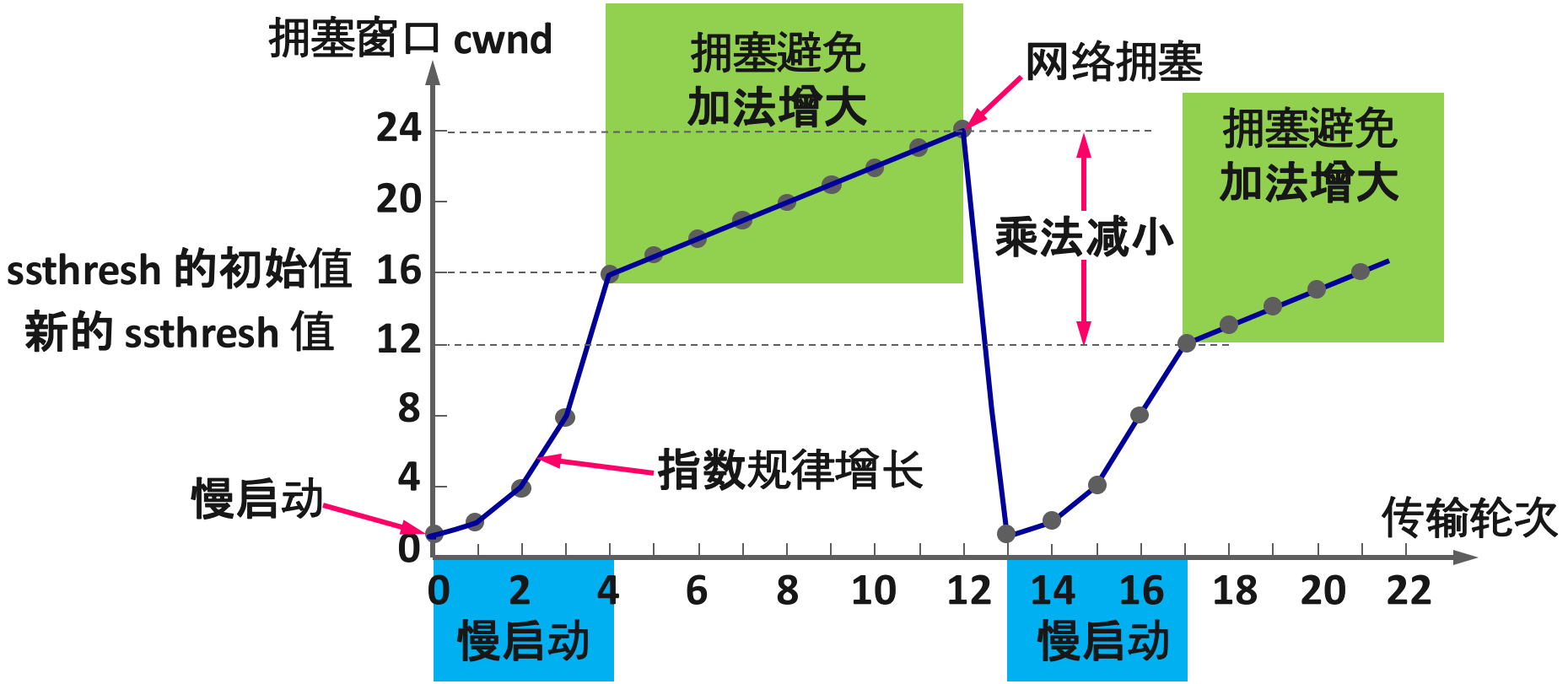
当拥塞窗口 cwnd 增长到慢启动门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

3.4.3 TCP的拥塞控制



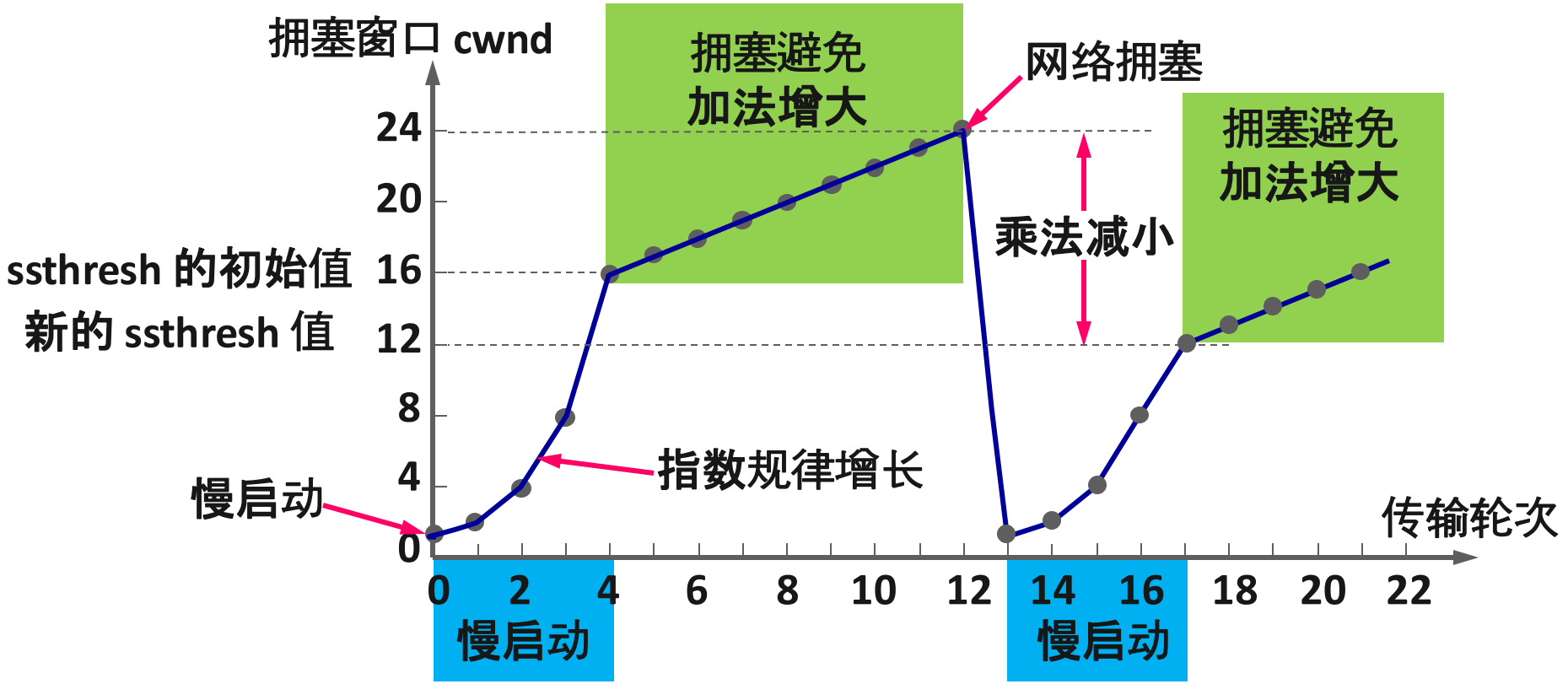
假定拥塞窗口的数值增长到 24 时，网络出现超时（表明网络拥塞了）。

3.4.3 TCP的拥塞控制



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，并执行慢启动算法。

3.4.3 TCP的拥塞控制



当 cwnd = 12 时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。

3.4.3 TCP的拥塞控制

- “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。

3.4.3 TCP的拥塞控制

2） 快恢复算法

- TCP检测到分组丢失有两种情况：重传计时器超时和收到连续三个重复的ACK。
- 当发送方收到连续三个重复的ACK时，说明网络还有一定的分组交付能力，拥塞情况并不严重。
- 这时将拥塞窗口直接降低为1则反应过于剧烈了，因此需要采用能更快恢复发送速率的算法。

(1) 当发送方收到连续三个重复的 ACK 时，就重新设置慢启动门限 ssthresh。

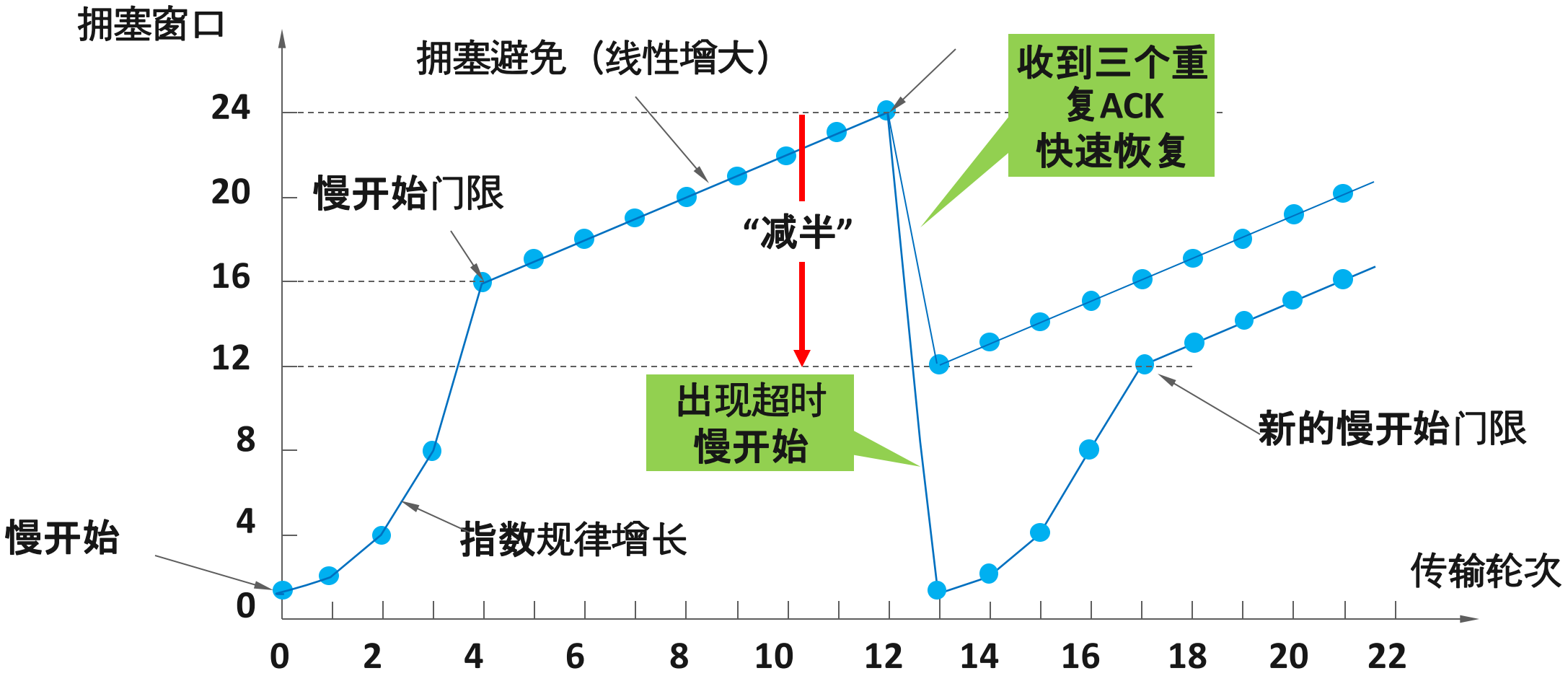
(2) 与慢启动不同之处是拥塞窗口 cwnd 不是设置为 1，而是设置为 ssthresh + 3 × MSS。

(3) 若收到的重复的 ACK 为 n 个（$n > 3$），则将 cwnd 设置为 ssthresh + n × MSS。

(4) 若发送窗口值还容许发送报文段，就按拥塞避免算法继续发送报文段。

(5) 若收到了确认新的报文段的 ACK，就将 cwnd 缩小到 ssthresh。

3.4.3 TCP的拥塞控制



3.4.3 TCP的拥塞控制

“加性增”与“乘性减”

- 对于长时间的TCP连接，在稳定时的拥塞窗口大小呈锯齿状变化“**加性增、乘性减**”
- 发送方的平均发送速率始终保持在较接近网络可用带宽的位置（慢启动门限之上）。