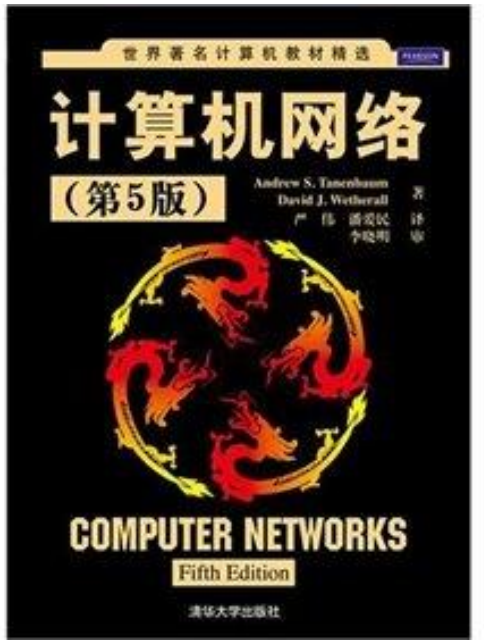
计算机网络

Andrew S. Tanenbaum（5 Edition）



安徽大学 互联网学院
School of Internet Anhui University

计算机网络

第1章 引言

第2章 物理层

第3章 数据链路层

第4章 介质访问控制子层

第5章 网络层

第6章 传输层

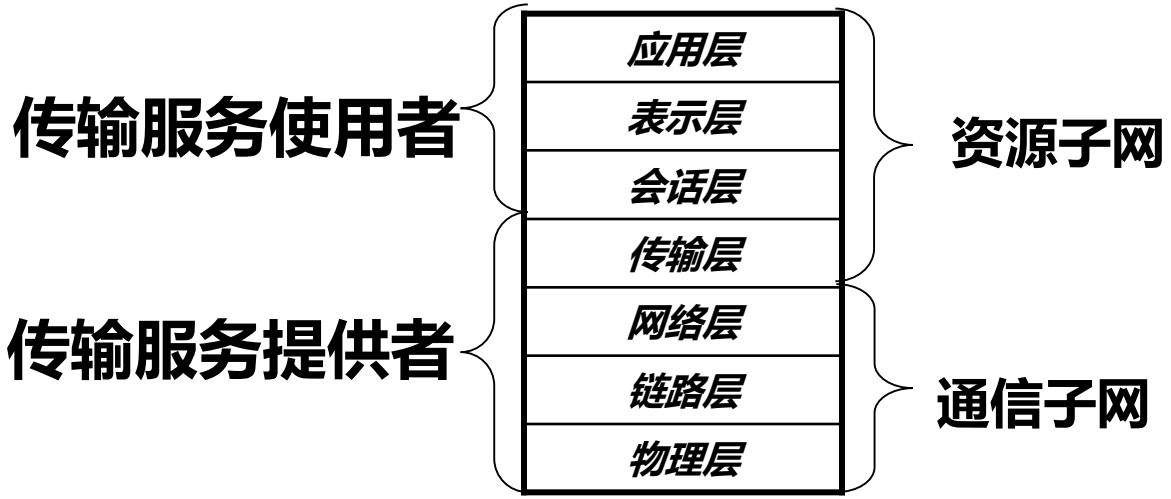
第7章 应用层

第8章 网络安全

第6章 传输层

6.1 传输服务

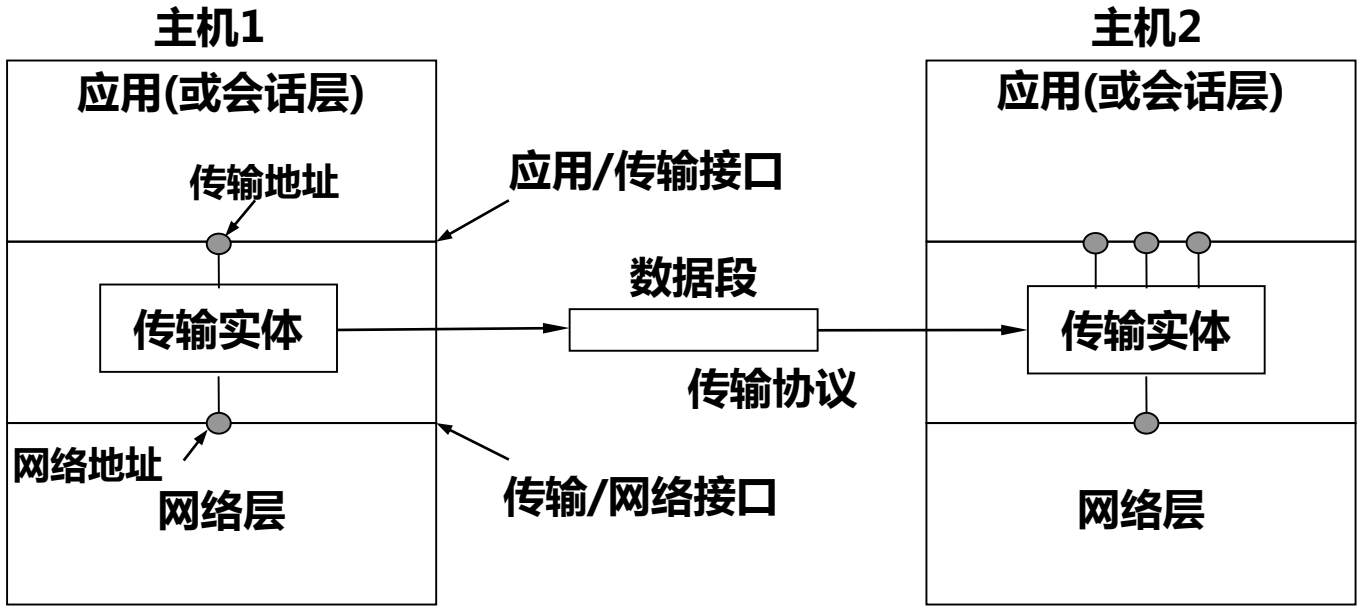
传输层在OSI模型中的位置



- 介于通信子网和资源子网之间，对高层用户屏蔽了通信的细节
- 弥补了通信子网所提供服务的差异和不足，提供端到端之间的无差错保证
- 传输层工作的简繁取决于通信子网提供服务的类型

6.1 传输服务

传输层与上下层之间的关系



- 传输层提供面向连接和无连接的服务，需要对流量和拥塞做控制，看似与网络层功能相近
- 网络层主要功能运行在路由器上，传输层则运行在用户终端，使通信服务更可靠。传输层还可以调用库函数实现特定的服务功能。

6.1.2 传输服务原语

- 网络层服务和传输层服务的区别

（1）**可靠性：**网络层服务一般是不可靠的，可能会丢失数据包。传输层同时提供可靠的面向连接的服务和不可靠的无连接服务，并能够在不可靠的网络上提供可靠的服务。

（2）**服务的对象：**网络层服务被传输实体所使用，一般不直接面向用户。传输层服务能够被用户直接调用。

（3）**运行位置：**网络层服务程序主要运行在路由器上，传输层代码完全运行在用户的机器上。

（4）**通信实体：**网络层服务通信实体为主机之间，传输层为两个进程之间。

（5）**传输的数据：**网络层传输的是IP数据包，传输层为段或传输协议数据单元（TPDU）

6.1.2 传输服务原语

用户可以使用传输服务原语，调用系统函数库，实现传输层服务功能。

传输服务原语是应用程序和传输服务之间的接口

原　语	TPDU发送的	含　　义
LISTEN	(无)	阻塞，直到某个过程试图连接
CONNECT	CONNECTION REQ	主动尝试建立一个连接
SEND	DATA	发送信息
RECEIVE	(无)	阻塞，直到一个DATA TPDU到达
DISCONNECT	DISCONNECTION REQ	请求释放连接

- 一个典型的面向连接的服务原语

6.1.2 Berkeley套接字

伯克利套接字（**Berkeley Sockets**），是软件系统的一部分，可被调用实现**TCP**传输功能

原　语	含　　义
SOCKET	创建一个新的通信端点
BIND	将本地地址关联到套接字上
LISTEN	声明愿意接受连接，给出队列长度
ACCEPT	被动创建一个入境连接
CONNECT	主动建立连接的尝试
SEND	在指定连接上发送数据
RECEIVE	从指定连接中接收数据
CLOSE	释放指定的连接

JAVA中的SOCKET套接字方法

ServerSocket类

Socket accept() 产生阻塞，监听指定的端口，直至有客户端发来连接请求

getInputStream() 网络输入流，同时返回一个InputStream对象实例。

getOutputStream()网络输出流，同时返回一个OutputStream对象实例。

close() 关闭当前ServerSocket

void bind(SocketAddress endpoint）绑定IP和端口

InetAddress getInetAddress() 返回ServerSocket监听的，本机的IP地址

getLocalPort() 返回ServerSocket监听的，本机的IP地址上指定的端口号

int getSoTimeout()/void setSoTimeout(int timeout) 设置读取数据时阻塞链路的超时时间

connect(SocketAddress endpoint)主动连接目的主机套接字

6.2 传输协议的要素

- **传输层与数据链路层对比**

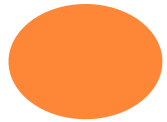
相似点：都能够提供错误控制，顺序性处理和流量控制的功能等

不同点：数据链路层通过(有/无线)物理通道直接通信，传输层面向的传输通道是整个网络；

- 数据链路层的连接建立较简单，传输层要复杂得多；
- 数据链路层的通信是点对点的，无中间存储环节，每条输出线对应了唯一的一个设备。传输层需要设定目的端地址 ，数据传输可能经过多个路由器的存储、寻址、转发，每次传输的线路都不固定。
- 数据链路层通常使用一对发送缓冲区和接收缓冲区，在传输层对每个连接都必须分配一定的缓冲区，缓冲区管理复杂得多 。

6.2 传输协议的要素

- 为了实现可靠传输，传输层（TCP）需要对通信连接进行管理，保证连接过程中不会出现差错，具体过程包括
  - **建立连接**
  - 传输数据
  - **释放连接**



6.2 传输协议的要素

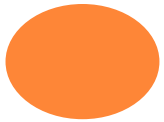
- 为了实现可靠传输，传输层需要对通信连接进行管理，保证连接过程中不会出现差错，具体过程包括：

  为了确保数据传输具备所需的可靠性，进行**差错控制**；

  为了防止快速发送端淹没慢速接收端，进行**流量控制**；

  为了放缓往网络中发送数据包的速度，进行**拥塞控制**。

- 不可靠传输不需要建立连接，更不需要管理连接。



6.2.1 寻址

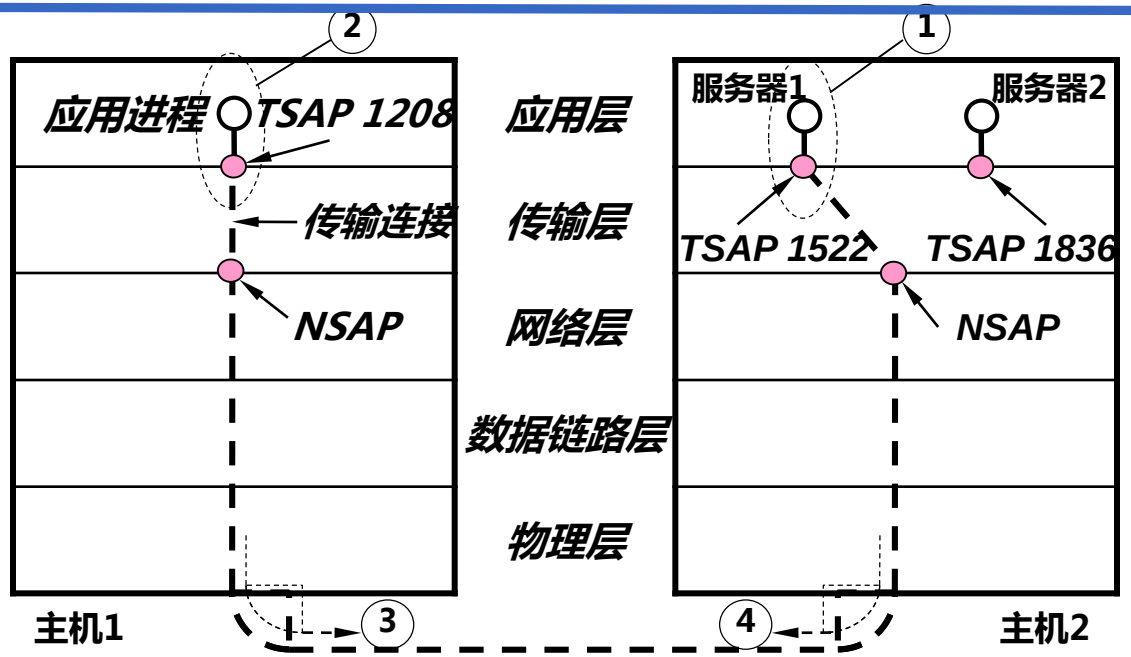
- 传输服务访问点TSAP（Transport Service Access Point）
- 当一个应用进程与远程应用进程建立连接时，必须指明需要连接到哪个进程上，这个标记称为传输层地址，更通用的称为传输服务访问点（TSAP），可以看作是在特定协议层通信实体的ID标志符。
  - 在TCP协议中传输层地址即TCP的端口号
  - 网络层地址称为网络服务访问点NSAP（Network Service Access Point），NSAP在IP协议中即IP地址

6.2.1 寻址

- 端口port
- 运行的进程在计算机中用进程标识符来标志的。但不能用于网络中通信实体的标识，因为在因特网上使用的计算机的操作系统种类很多，而不同的操作系统又使用不同格式的进程标识符。
- 传输层用一个 16 位二进制码代表端口号标识不同进程。端口号只具有本地意义，只是为了标识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。
- 端口号范围0-65535

6.2.1 寻址

- 访问一个邮件服务器



- ✓ 1、主机2上的邮件服务进程将自己关联到1522号TSAP上，等待即将到来的请求，例如，可以用LISTEN调用。
- ✓ 2、主机1上的一个应用进程发送邮件，便发出一个CONNECT请求，将1208号TSAP设定为源地址，将1522号TSAP设定为目的地址 ；
- ✓ 3、主机1上的应用进程发送邮件 ；
- ✓ 4、主机2上的服务器进程响应该邮件；
- ✓ 5、传输连接释放 。

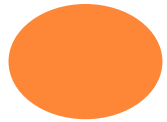
6.2.1 寻址

- 如何知道对方的TSAP
- well-known TSAP

  每个服务都有自己固定的TSAP，所有网络用户都知道

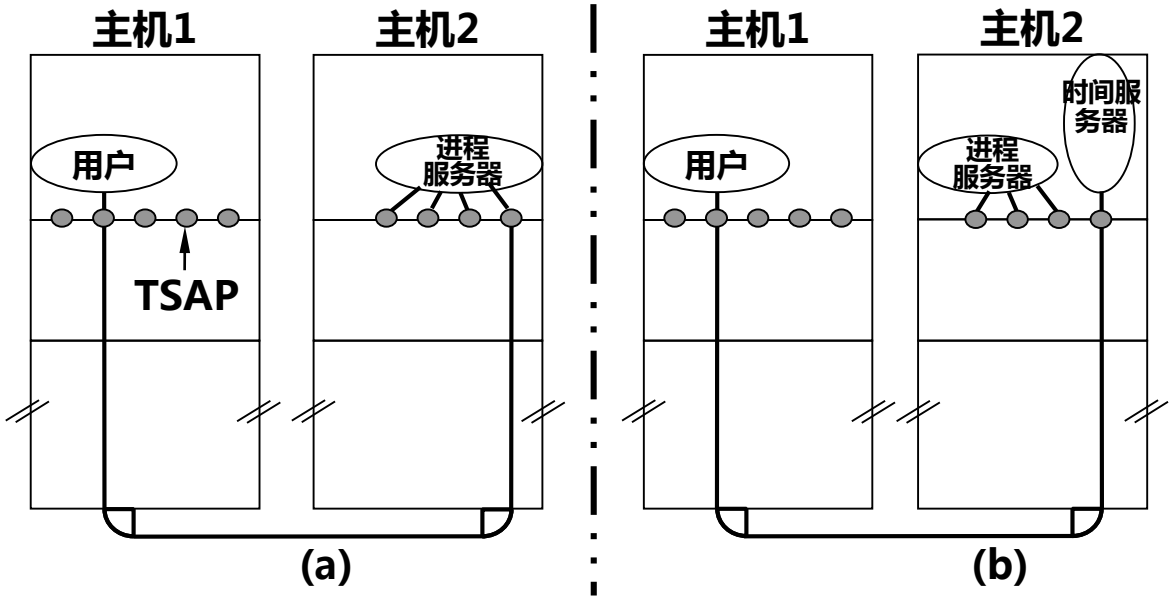
FTP	TELNET	SMTP	DNS	TFTP	HTTP	SMTP
21	23	25	53	69	80	161

- 采用端口映射器，将特定服务的名字与一个TSAP相连接。
- 用户向服务器发送一个报文，指明服务的名称，映射器将该服务对应的TSAP返回给用户



6.2.1 寻址

- 初始连接协议（initial connection protocol）：用于监听并接受用户各种端口的的请求 。基本原理如下。
- 服务器（主机2）上运行一个特殊的进程服务器，同时监听一系列的端口，等待用户的TCP连接请求
- 用户（主机1）设定所需服务的TSAP地址，发出一个CONNECT请求
- 主机2的进程服务器在收到请求后，便将请求装入被请求服务器，被请求服务器将继承已建立的连接并为用户提供服务
- 主机2的进程服务器又继续监听新的请求

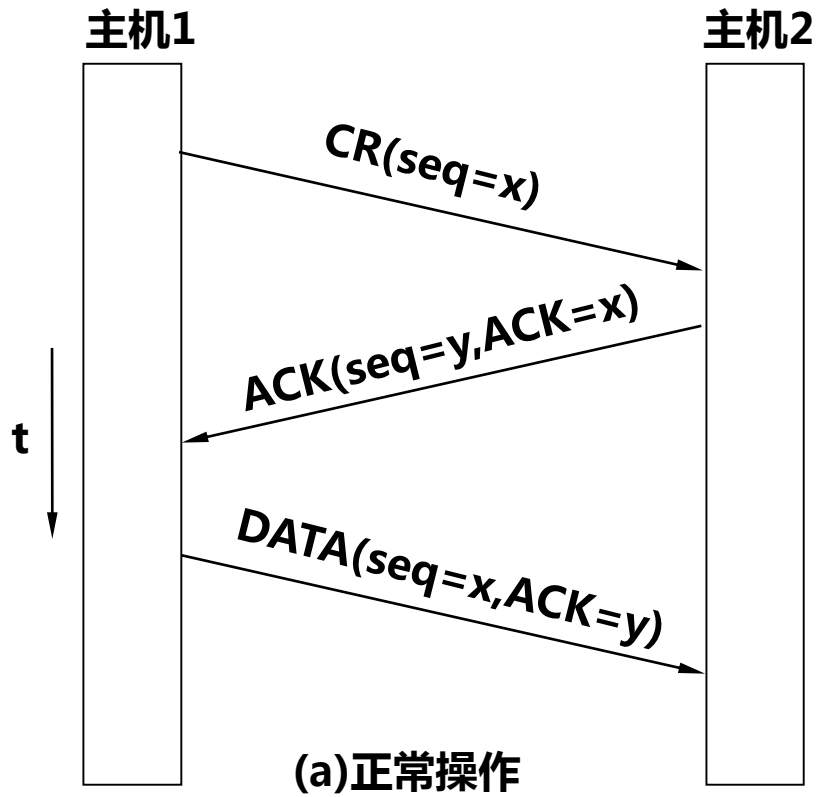


6.2.2 连接建立

- 通信子网中存在着延时和分组的丢失，以及由于延时和丢失而带来的重复分组
- 由于通信子网的尽力而为的传输原则，一个早已超时的分组最终还是到达了目的端，所以有必要将分组的生命周期限制在一个适当的范围内,连接建立时，如何处理过期分组，保证连接的唯一性是连接建立过程中首要考虑的问题
- 常用的方法是： 三次握手法
- 相关内容包括序号、计时器

6.2.2 连接建立

- 正常连接的三次握手过程



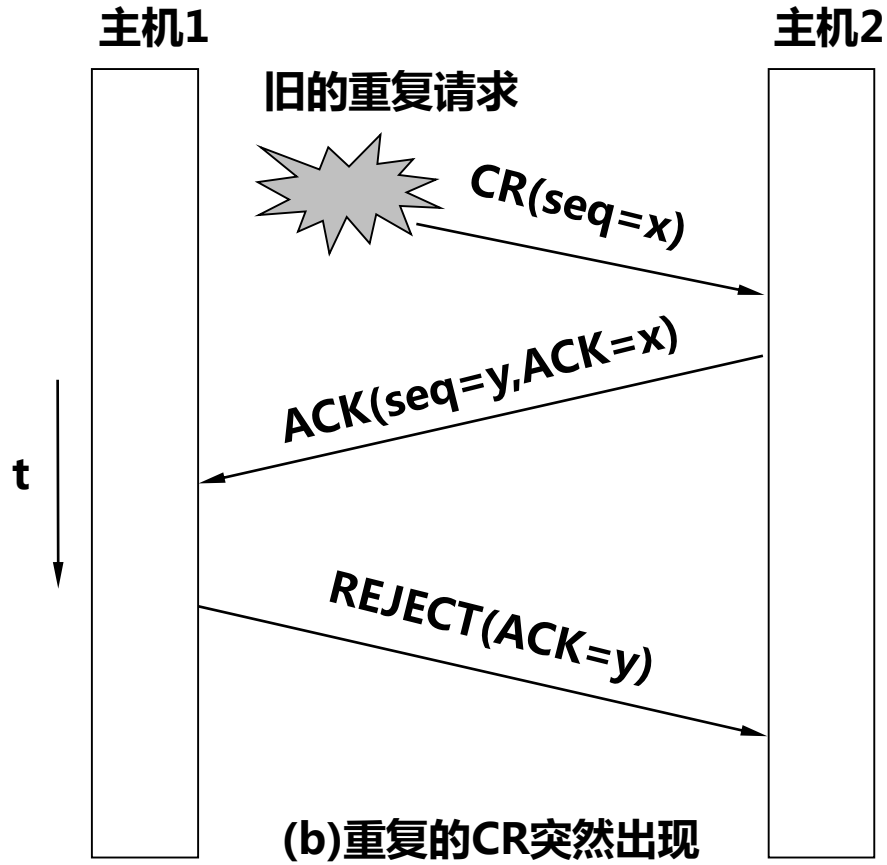
(a)正常操作

- ✓ 主机1发出的连接请求序号为x(seq=x)
- ✓ 主机2应答接受主机1的连接请求，并声明自己的序列号为y(seq=y,ACK=x)，
- ✓ 主机1收到确认后发送第一个数据TPDU并确认主机2的序列号(seq=x,ACK=y)
- ✓ 至此，整个连接建立过程正常结束，数据传输已正式开始

CR：Connection Request（连接请求）

6.2.2 连接建立

- 出现延迟的重复TPDU时三次握手的工作过程

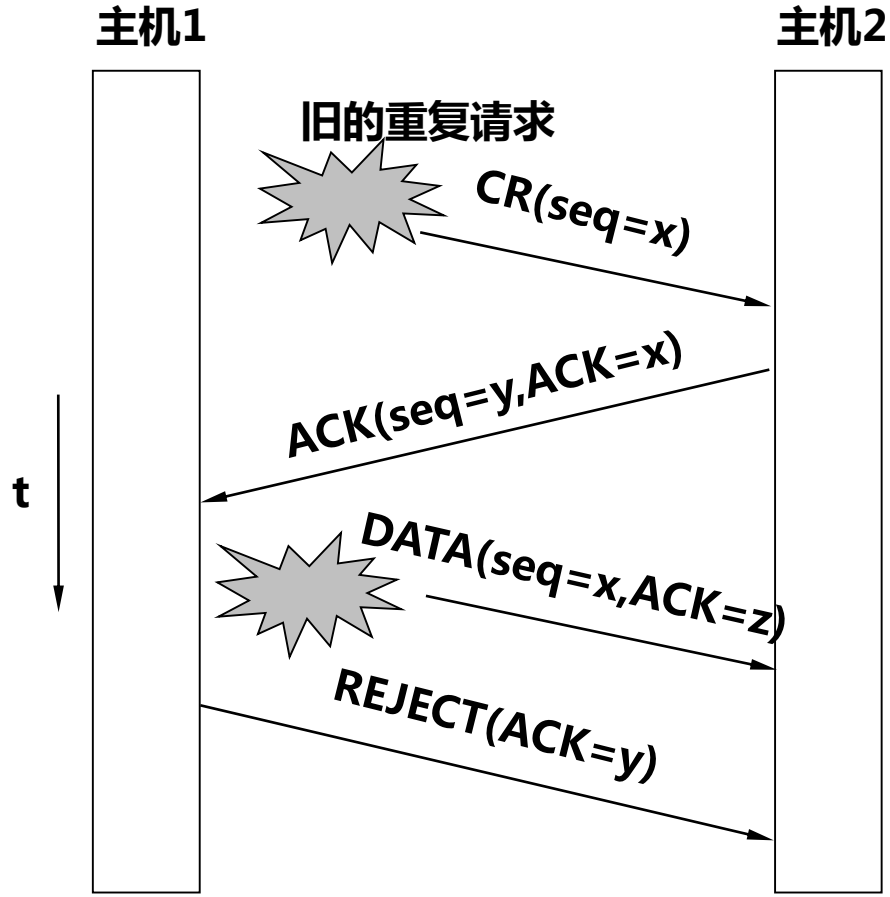


(b)重复的CR突然出现

- ✓ 一个已经释放连接的主机1的延时重复的连接请求，该TPDU在主机1毫不知晓的情况下到达主机2
- ✓ 主机2通过向主机1发送一个接受连接请求的TPDU来响应该TPDU，并声明自己的序号为y(seq=y,ACK=x)
- ✓ 主机1收到这个确认后拒绝建立连接，
- ✓ 主机2收到了主机1 的拒绝才意识到自己受到了延时的重复TPDU的欺骗并放弃该连接
- ✓ 延时的重复请求将不会产生不良后果

6.2.2 连接建立

- 子网中同时有作废的CR和ACK的情况



(c)重复的CR和重复的ACK

- ✓ 主机2收到了一个延时的CR并做了确认应答，注意主机2已经声明使用y作为从到主机1进行数据传输的初始序号，因此主机2十分清楚在正常情况下，主机1的数据传输应捎带对y确认的TPDU，
- ✓ 当第二个延时的TPDU到达主机2时，主机2根据它确认的是序号z而不是y知道这也是一个过时的重复TPDU，因此也不会无故建立无人要求的连接

6.2.3 连接释放

- **非对称释放**

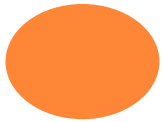
  一方中止连接，则连接即告中断

  缺陷：可能导致数据丢失

- **对称释放**

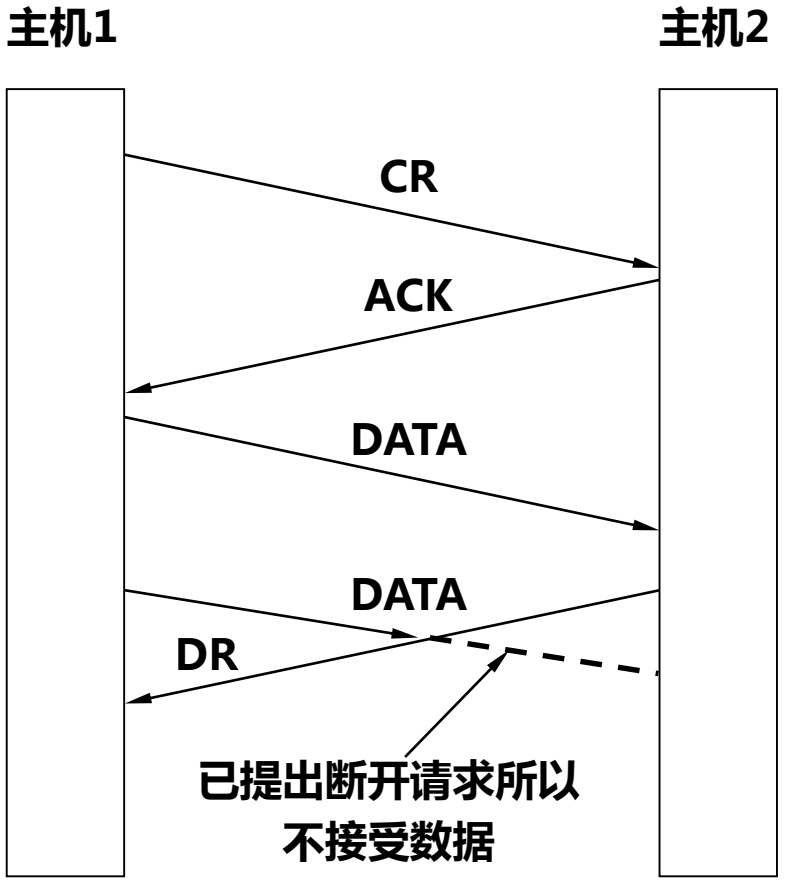
  A提出中止请求，B同意即中止

  问题：B如何知道A 收到了它的确认(蓝白军问题)



6.2.3 连接释放

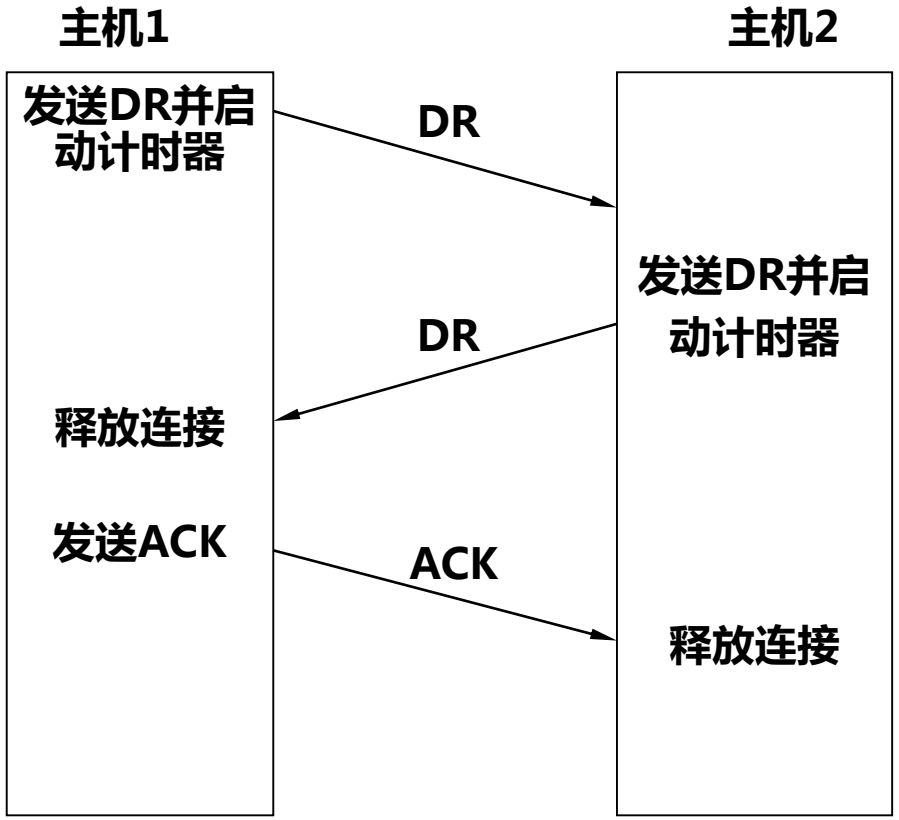
- 非对称释放：一方中止连接，则连接即告中断



- ✓ 当连接建立后，主机1发送了一个数据TPDU并正确抵达主机2；
- ✓ 接着，主机1发送了第二个数据TPDU；
- ✓ 主机2在收到第二个TPDU之前先突然发出了释放连接请求 DISCONNECT，
- ✓ 结果是连接立即被释放，数据被丢失

6.2.3 连接释放

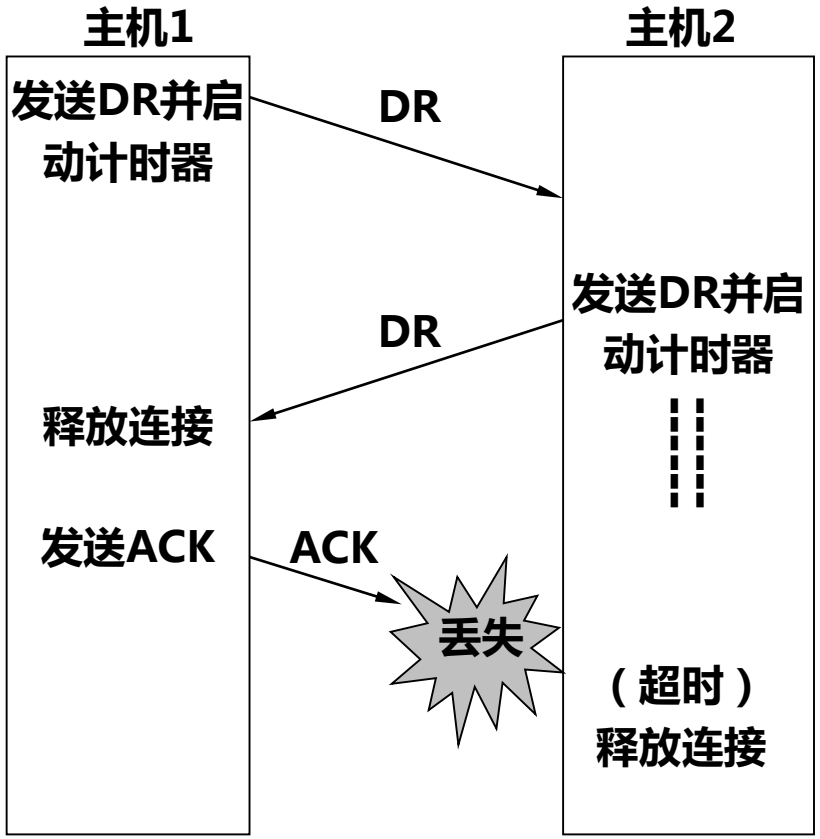
- 对称释放:三次握手的正常情况



- ✓ 主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器;
- ✓ 主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器;
- ✓ 主机1 在计时器没有超时前收到主机2 的DR，便正式释放连接并发送ACK，主机2也在计时器没有超时前收到主机1 的ACK，于是也释放了连接；
- ✓ 至此整个数据传输过程，包括建立连接、传输数据和释放连接的过程正常结束

6.2.3 连接释放

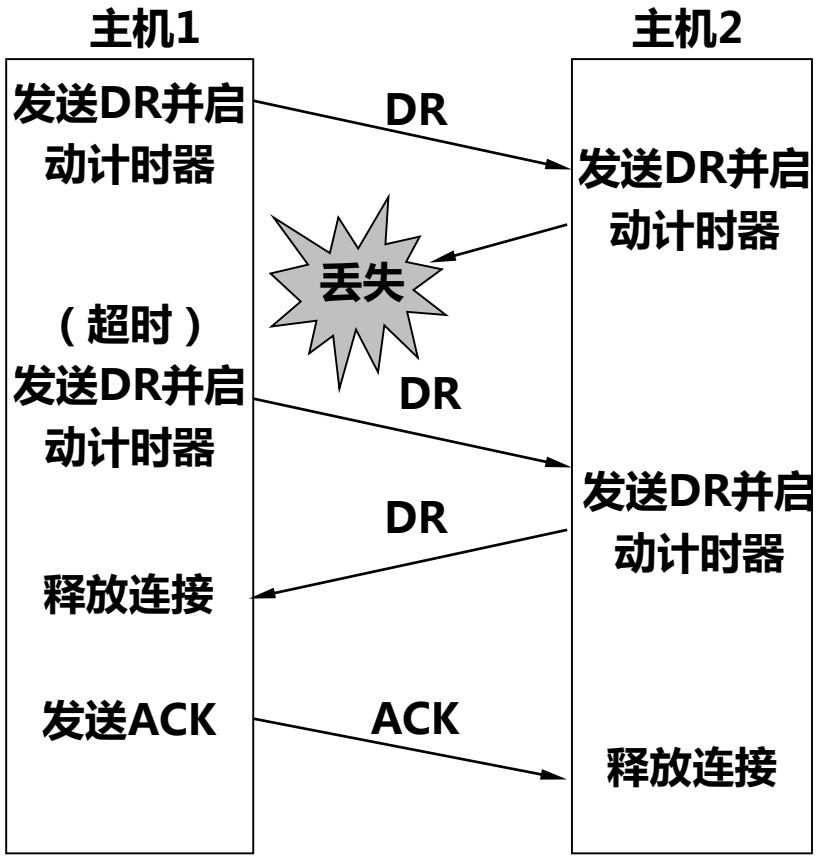
- 对称释放2：最后的确认TPDU丢失



- ✓ 主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器;
- ✓ 主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器;
- ✓ 主机1 在计时器没有超时前收到主机2 的DR，便正式释放连接并发送ACK;
- ✓ 主机2在计时器超时后还未收到主机1 的ACK，但是由于已经超时，于是也释放了连接

6.2.3 连接释放

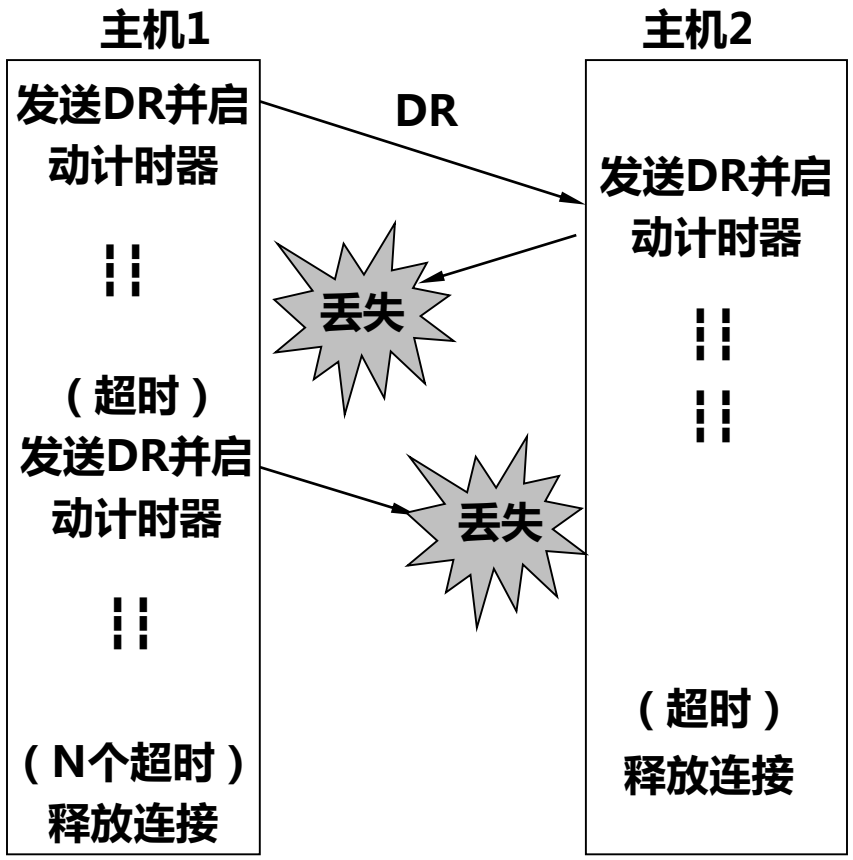
对称释放3:ACK丢失



- 主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器;
- 主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器;
- 主机1 在计时器超时后还未收到主机2的DR，于是又重新发送DR并启动计时器，下面便是一个正常的三次握手，并最后正常释放连接，即整个数据传输过程正常结束

6.2.3 连接释放

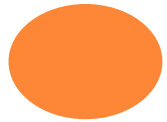
- 对称释放4:ACK丢失以及后续的DR丢失



- ✓ 主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器;
- ✓ 主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器;
- ✓ 无论是哪一方的超时重发的TPDU都不能到达对方
- ✓ 最终，接收方计时器的超时而也释放连接，发送方经过n次重发和超时后释放连接 .

6.2.4 差错控制和流量控制

- 方法：1、检错码和纠错码
  2、自动重传请求ARQ
  3、停止等待协议
  4、滑动窗口协议
- 流量控制是发送方和接收方之间的传输速率上的匹配，为使没有得到确认的PDU在超时后的重发，通常必须在缓冲区中暂存 。



6.2.4 差错控制和流量控制

- 流量控制
  - ✓ 将缓冲与确认机制分离。发送端请求缓冲区大小，接收端给予分配一定缓冲区空间。
  - ✓ 发送端每发送一段数据，就计算剩余缓冲区大小，如果减小到0则停止发送。
  - ✓ 接收方在回复的消息中捎带上确认和剩余的缓冲区数量。

6.2.4 差错控制和流量控制

缓冲区的大小

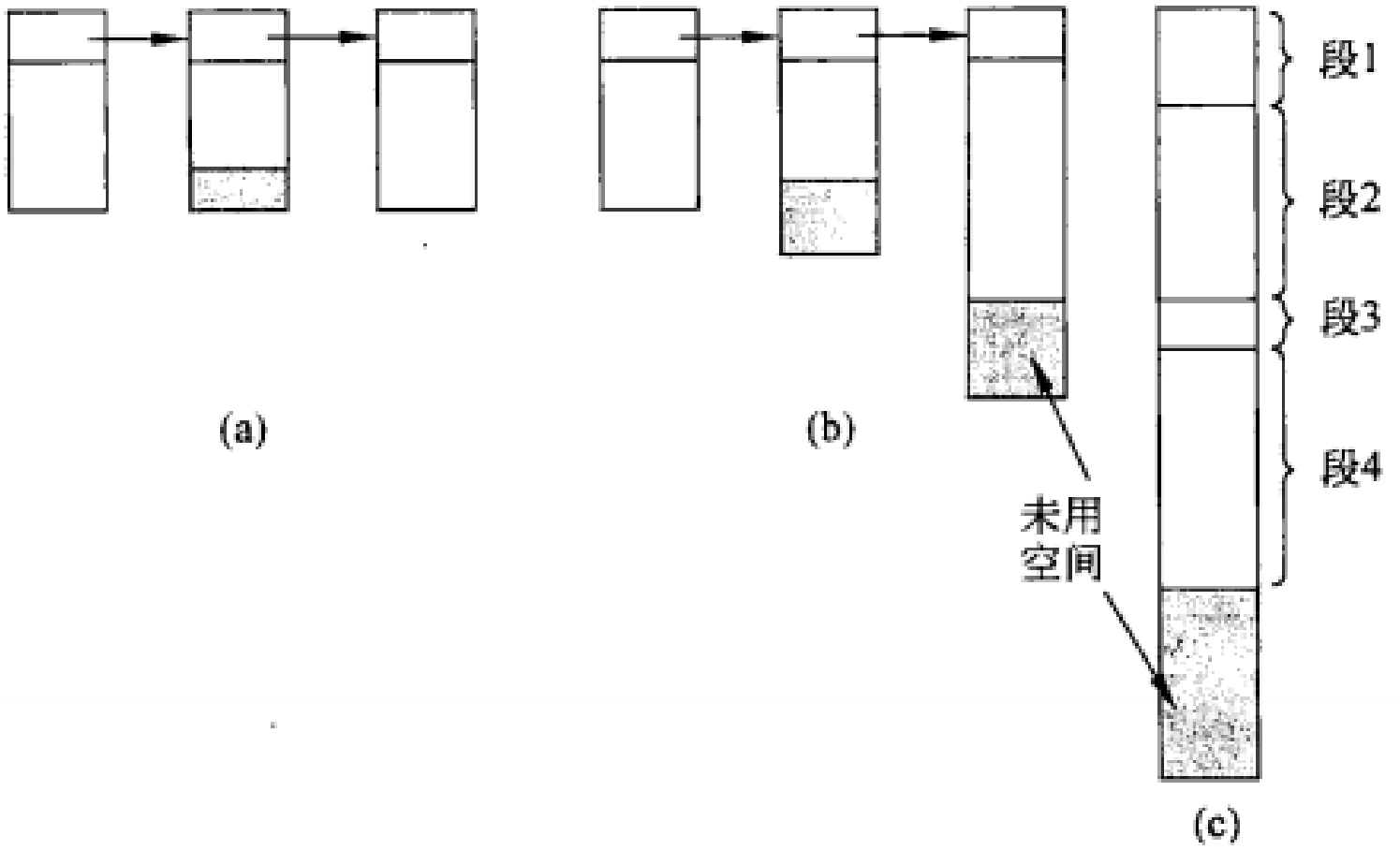
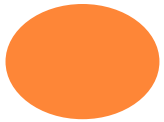


图 6-15

（a）链式固定大小的缓冲区；（b）链式可变大小的缓冲区；（c）每个连接一个大循环缓冲区

6.2.4 差错控制和流量控制

- 动态缓冲区的分配管理：常用做法是将缓冲与确认机制分离。
- 动态缓冲区管理实际上意味着一个可变大小的窗口。
  - ✓ 初始时，发送端根据它期望的需求，请求一定数量的缓冲区
  - ✓ 接收端根据它的能力分配尽可能多的缓冲区。
  - ✓ 每次发送端传输一段，它必须减小分配给它的缓冲区数。
  - ✓ 当分配给发送端的缓冲区数达到0时，完全停止发送。
  - ✓ 接收端在逆向流量中捎带上单独的确认和缓冲区数。TCP将缓冲区的分配捎带在头的window size字段中。



6.2.4 差错控制和流量控制

TPDU序号	主机A	消息	主机B	注　释
1	→	<请求8个缓冲区>	→	A想要8个缓冲区
2	←	<ack=15,buf=4 >	←	B只有4个缓冲区，只同意接收0-3
3	→	<seq=0,data=m0>	→	A发送了m0，A剩下3个缓冲区
4	→	<seq=1,data=m1>	→	A发送了m1，A剩下2个缓冲区
5	→	<seq=2,data=m2>	✗	A发送了m2，并以为B剩下1个缓冲区
6	←	<ack=1,buf=3>	←	B确认了m0,m1，并剩下3个缓冲区
7	→	<seq=3,data=m3>	→	A发送了m3，A剩下1个缓冲区
8	→	<seq=4,data=m4>	→	A发送了m4后剩下0个缓冲区，暂停
9	→	<seq=2,data=m2>	→	m2超时A重发m2，A剩下0个缓冲区
10	←	<ack=4,buf=0>	←	B确认了m3，并剩下0个缓冲区
11	←	<ack=4,buf=1>	←	B确认了m4，并剩下1个缓冲区
12	←	<ack=4,buf=2>	←	B确认了m2，B有了2个缓冲区
13	→	<seq=5,data=m5>	→	A发送了m5，A剩下1个缓冲区
14	→	<seq=6,data=m6>	→	A发送了m6后剩下0个缓冲区，暂停
15	←	<ack=6,buf=0>	←	B确认了m6，并剩下0个缓冲区
16	✗	<ack=6,buf=4>	←	B确认了m6，并剩下4个缓冲区，潜在死锁

动态缓冲区分配

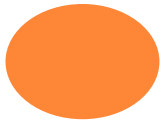
6.2.5 多路复用

- **多路复用**

多个传输层的连接共用一个网络层的连接（地址），称为多路复用，能提高网络层连接的利用率

- **逆向多路复用**

一个传输层的连接合用多个网络层的连接来发送，可增加其有效带宽，以提高传输速率



6.2.5 多路复用

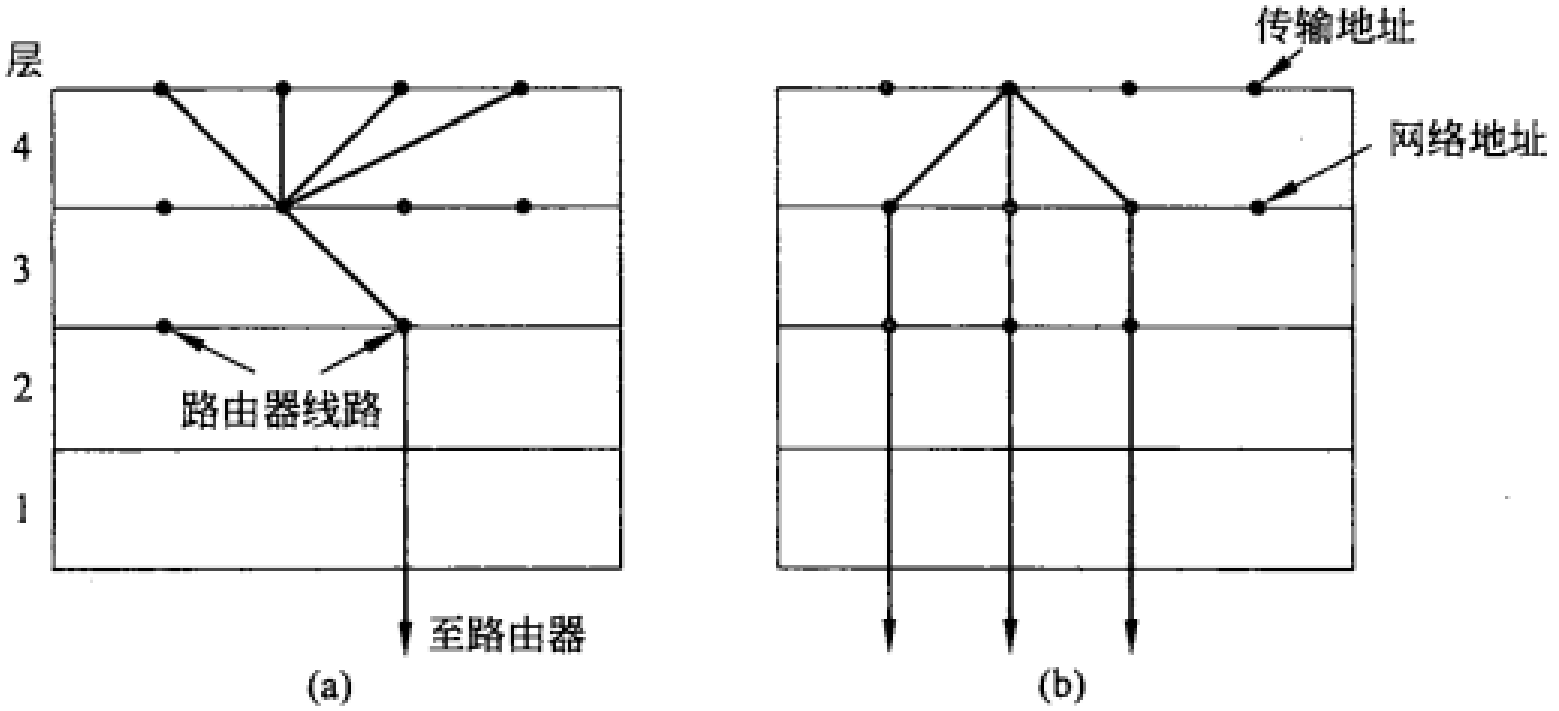


图 6-17

（a）多路复用；（b）逆向多路复用

6.2.6 崩溃恢复

- 服务器与崩溃相关的两个状态：s1（未确认）和s0（没有未完成）
- 服务端的三种事件：确认A，写入W，崩溃C

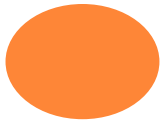
	接收主机采用的策略					
	先ACK，后写			先写，后ACK		
发送主机采用的策略	AC(W)	AWC	C(AW)	C(AW)	WAC	WC(A)
总是重传	OK	DUP	OK	OK	DUP	DUP
永远不重传	LOST	OK	LOST	LOST	OK	OK
在状态S0时重传	OK	DUP	LOST	LOST	DUP	OK
在状态S1时重传	LOST	OK	OK	OK	OK	DUP

OK＝协议功能正确
DUP＝协议产生了一个重复消息
LOST＝协议丢失了一个消息

图 6-18　客户和服务器策略的不同组合

6.3 拥塞控制

- 带宽分配
- 调整发送速率



6.3.1 理想的带宽分配

- 1、效率和功率　　功率=负载/延迟

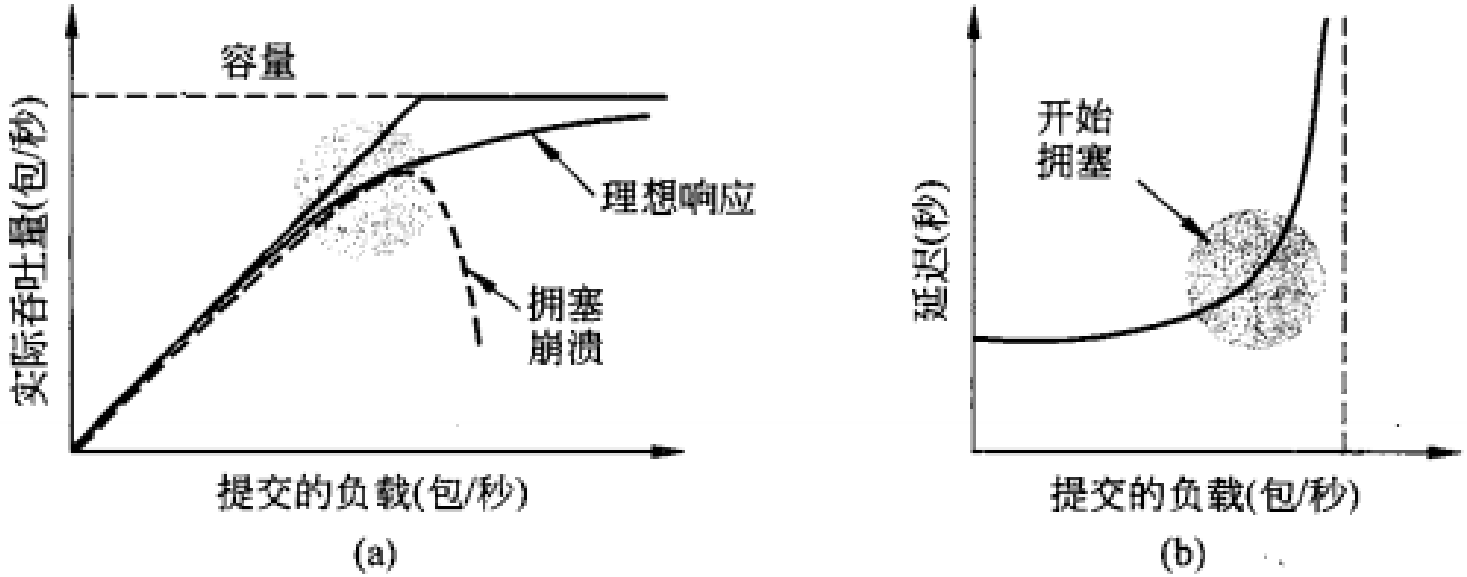


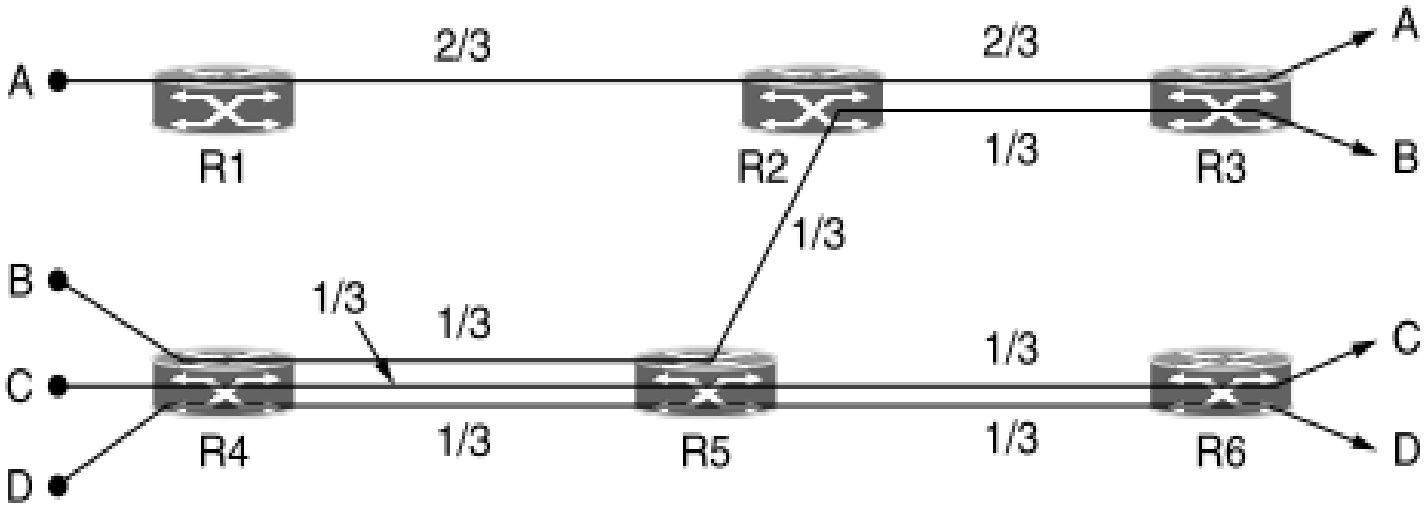
图　6-19

（a）实际吞吐量；（b）延迟作为提交负载的函数

6.3.1 理想的带宽分配

2、最大-最小公平性：加权公平队列WFQ

- 路由器本身会平均分配带宽，而拥塞控制机制会合理分配
- 拥塞控制会从整体上考虑带宽的分配
- 最大最小公平原则：资源按照需求递增的顺序进行分配；不存在用户得到的资源超过自己的需求；未得到满足的用户等价的分享资源



6.3.1 理想的带宽分配

- 3、收敛

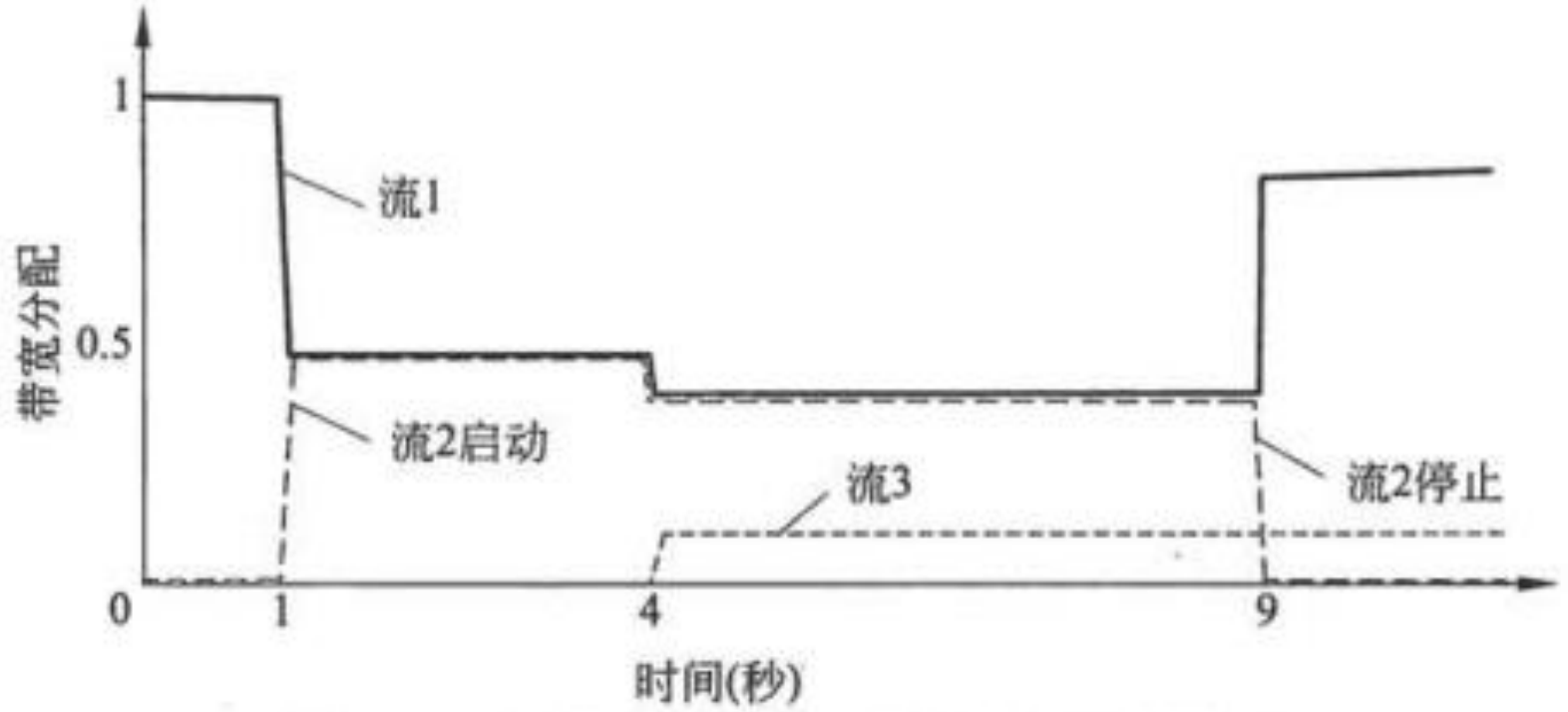


图 6-21　随着时间的推移而变化的带宽分配

6.3.2 调整发送速率

- 有两种方式调整发送速率：流量控制和拥塞控制。
- 流量控制用于接收端没有足够缓冲区的情况，拥塞控制用于网络容量不足的情况。

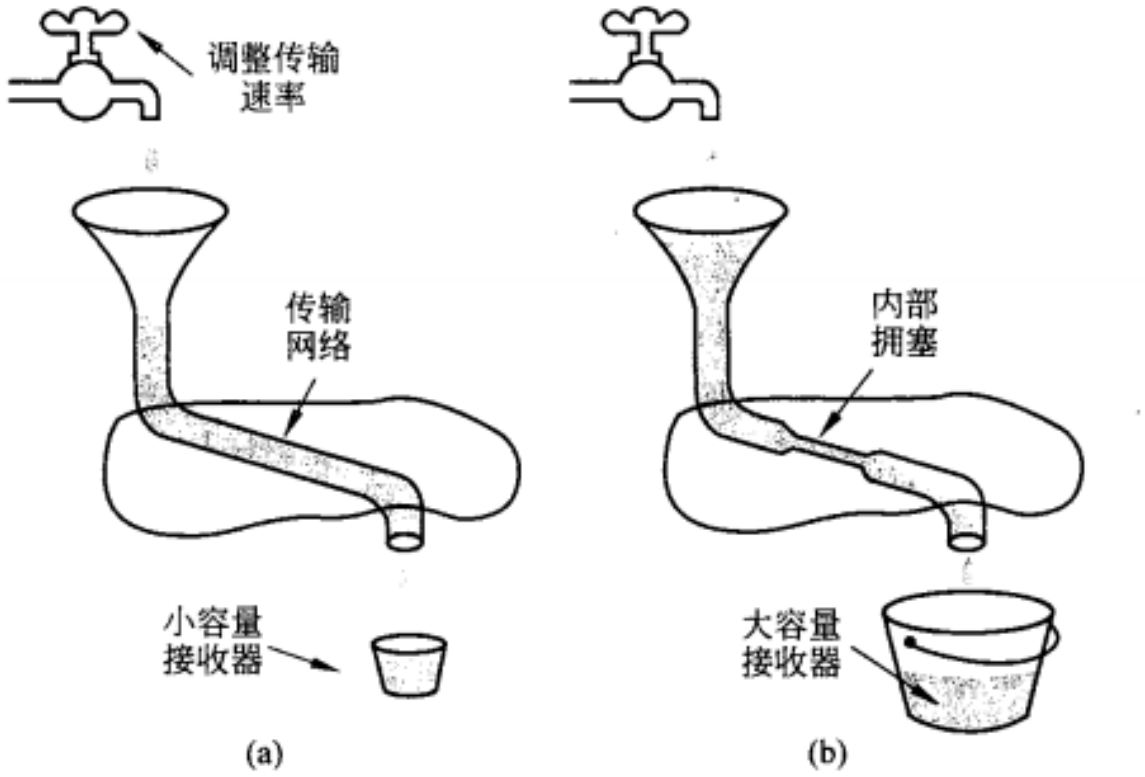


图 6-22

（a）一个快速网络往小容量接收器中注水；（b）一个小容量网络往大容量接收器中注水

6.3.2 调整发送速率

- 判断网络拥塞的方法包括显式拥塞协议XCP（路由器通知发送端），TCP显式拥塞通知ECN（路由器设置警告标识告诉发送端），FAST TCP（测量端到端延迟），当前使用丢包来判断。

协议	信号	显式否	精确否
XCP	使用速率	是	是
TCP	拥塞警告	是	否
FAST TCP	端到端延迟	否	是
Compound TCP	数据包丢失&端到端延迟	否	是
CUBIC TCP	数据包丢失	否	否
TCP	数据包丢失	否	否

图 6-23　一些拥塞控制机制的信号

6.3.2 调整发送速率

- 拥塞控制使用加法递增乘法递减（AIMD）规则：加速收敛，避免震荡

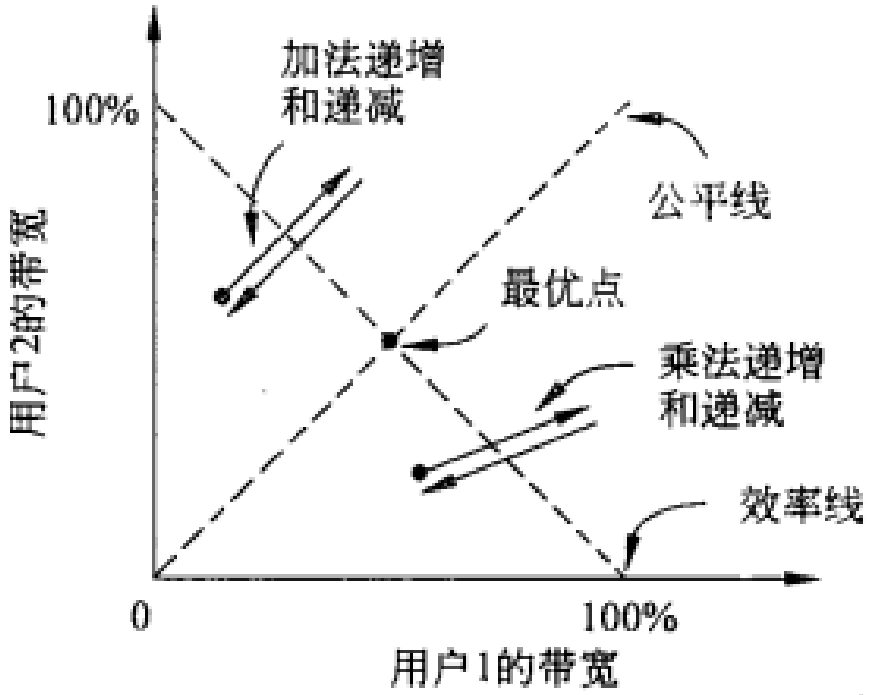


图 6-24　加法递增乘法递减的带宽调整

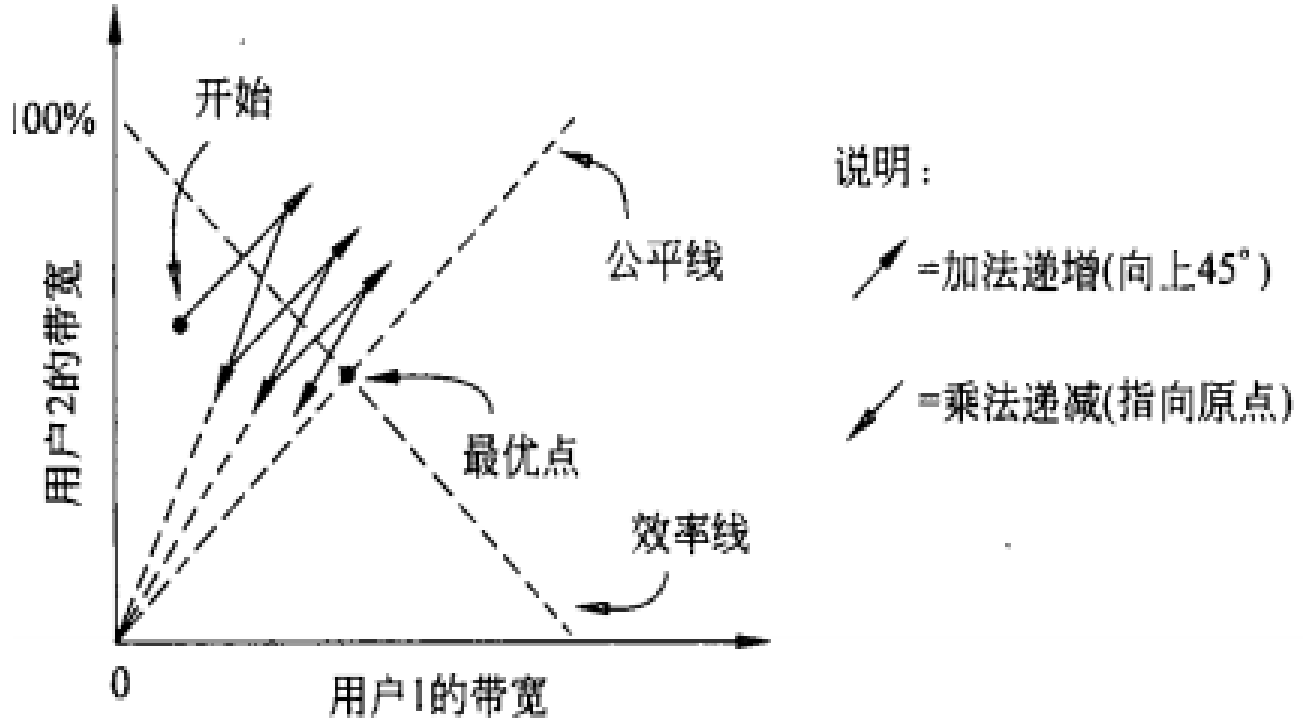


图 6-25　加法递增乘法递减（AIMD）控制法则

6.3.3 无线问题

- 无线网络的特征决定应使用丢包情况代表拥塞
- 无线网络使用前向纠错FEC：在数据包中加入冗余数据进行纠错

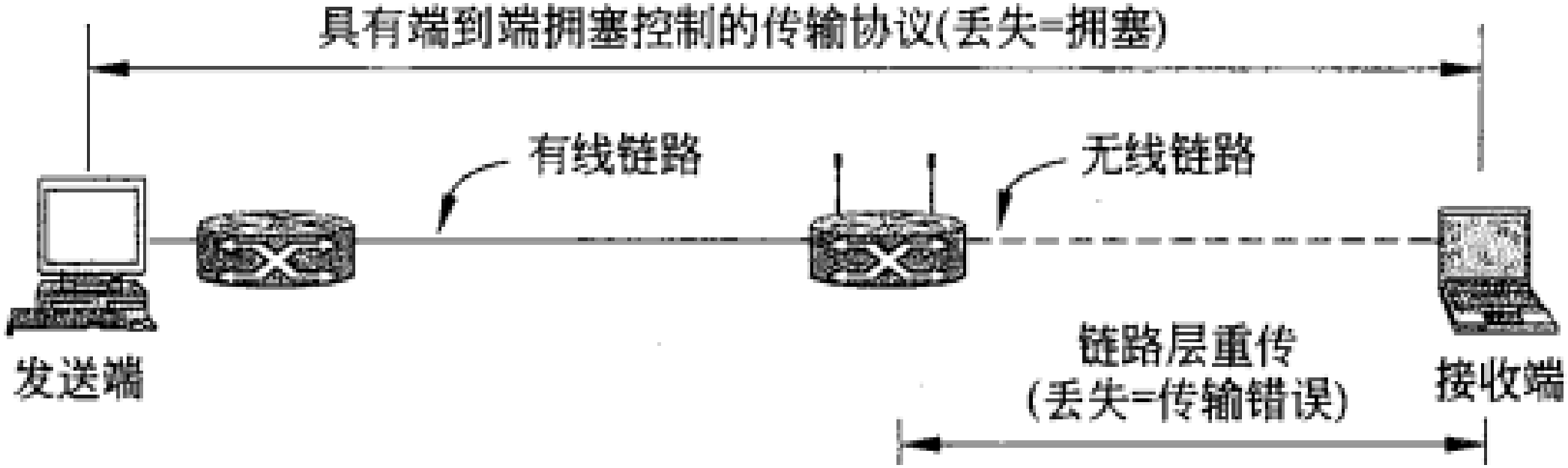
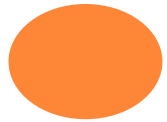


图 6-26　针对具有无线链路的路径的拥塞控制

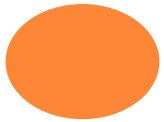
6.4 Internet的传输协议：UDP

- TCP和UDP都是Internet提供的传输层协议
  - **UDP** (User Datagram Protocol)用户数据报协议
  - **TCP** (Transmission Control Protocol) 传输控制协议



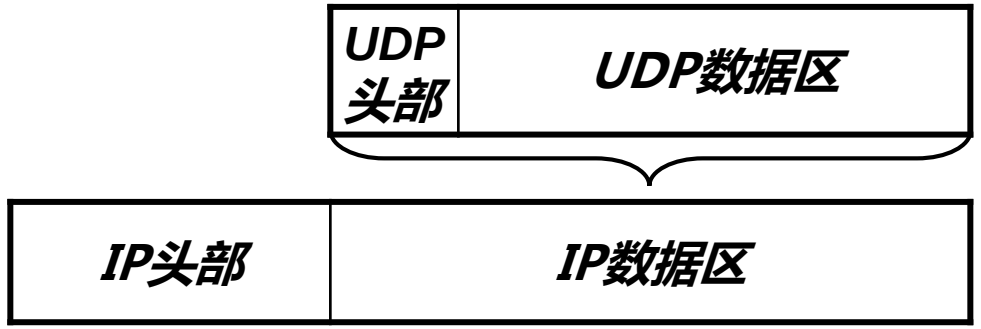
6.4.1 UDP概述

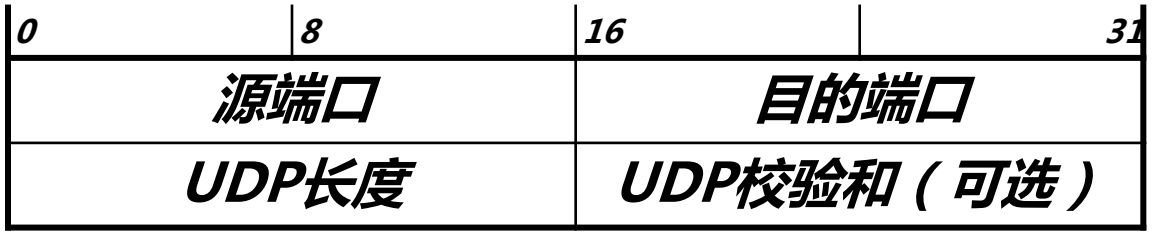
- UDP协议是无连接的数据报协议，它不提供可靠性服务，但其相应的协议开销也较小、效率较高
- 与IP协议相比，UDP主要增加的功能是提供端口协议，以保证进程通信。UDP没有提供流量控制、拥塞控制、重传机制。传输的可靠性要靠应用程序来解决。
- 域名系统DNS、简单网络管理协议SNMP使用的传输层协议是UDP



6.4.1 UDP概述

UDP的数据报格式





- 源端口：UDP端口号，当不需要返回数据时，源端口域置0
- 目的端口：UDP端口号
- UDP长度：整个数据段的长度，包括头部和数据部分以字节计，最小值为8（仅头部长度）
- UDP校验和：可选域，全0为未选，全1表示校验和为0

6.4.2 远程过程调用RPC

- 尽可能的使得一个远程过程调用看起来像本地过程调用
  - 1、客户存根---列集：记录经常访问的服务器地址和端口号，方便客户端访问
  - 2、服务器存根---散集：记录客户端的访问特性，识别客户端，

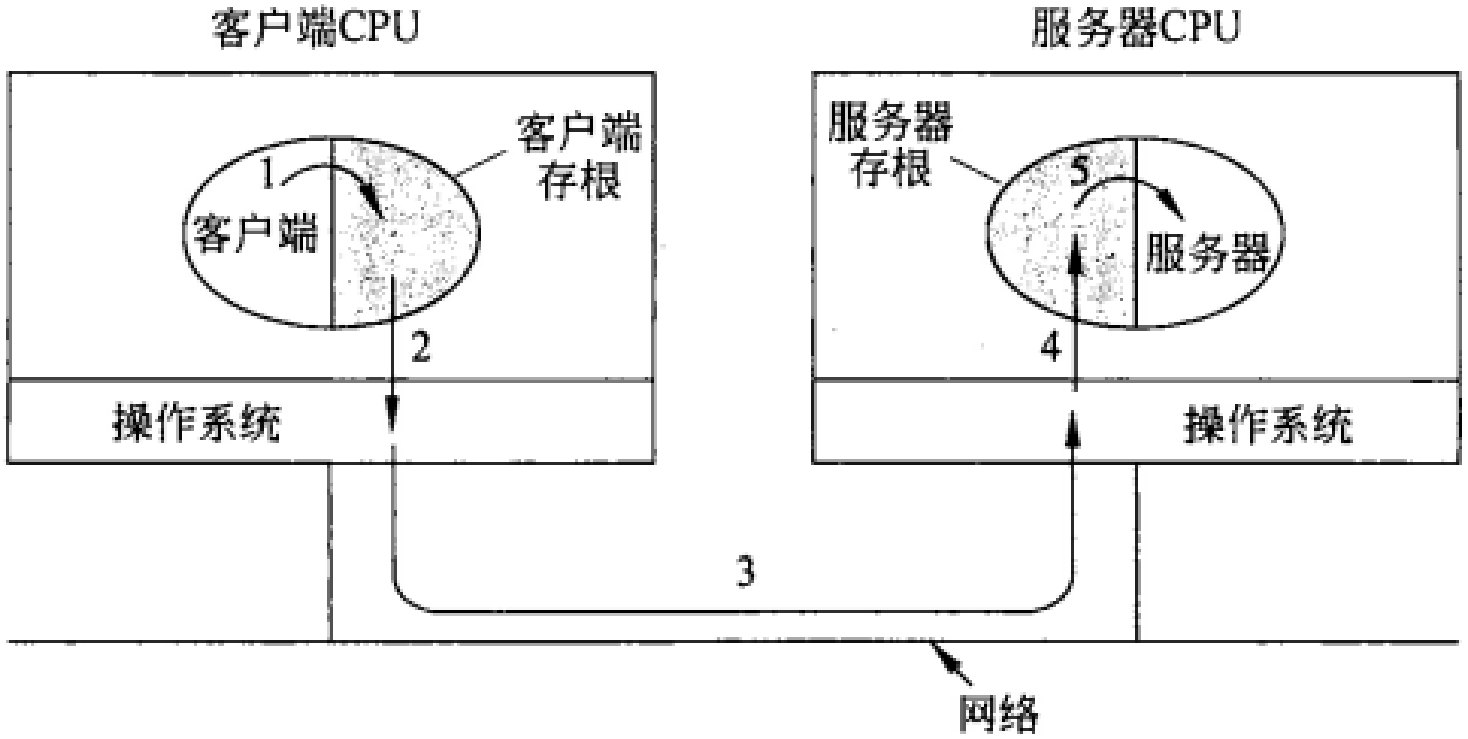


图 6-29　远程过程调用的步骤（阴影部分表示存根）

6.4.3 实时传输协议

- **1.实时传输协议RTP---应用于实时多媒体应用程序**

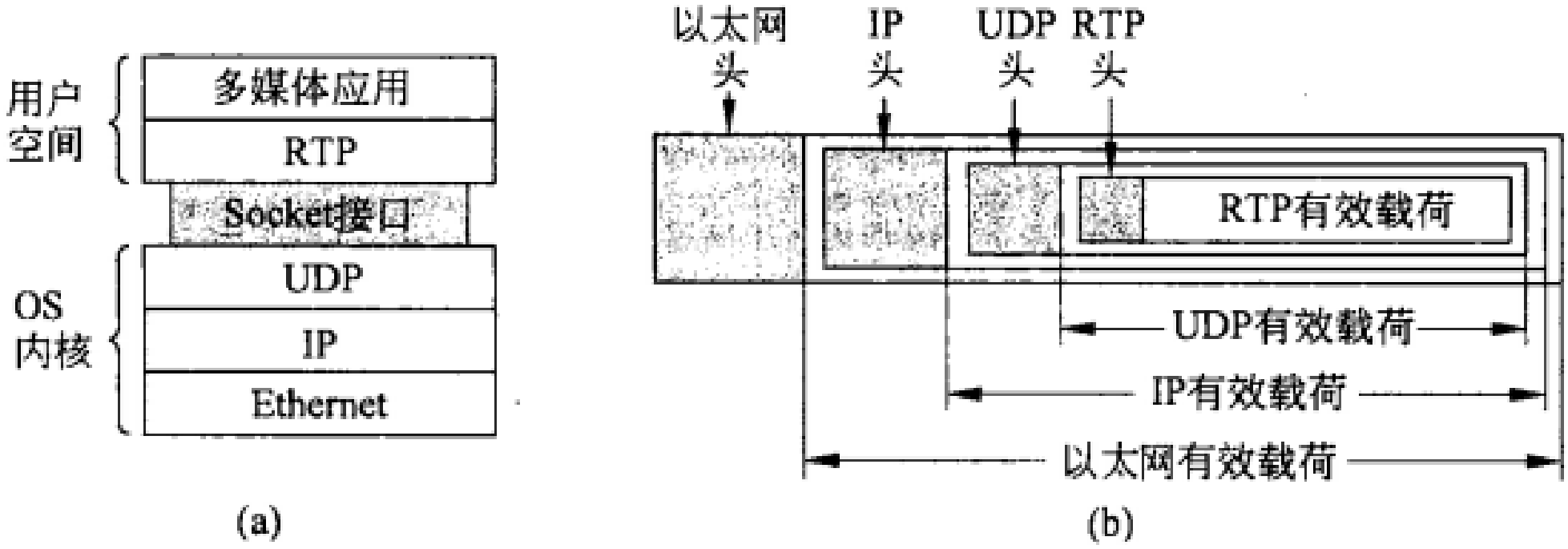


图 6-30

（a）RTP 在协议栈中的位置；（b）数据包的嵌套

6.4.3 实时传输协议

- **1.实时传输协议RTP**

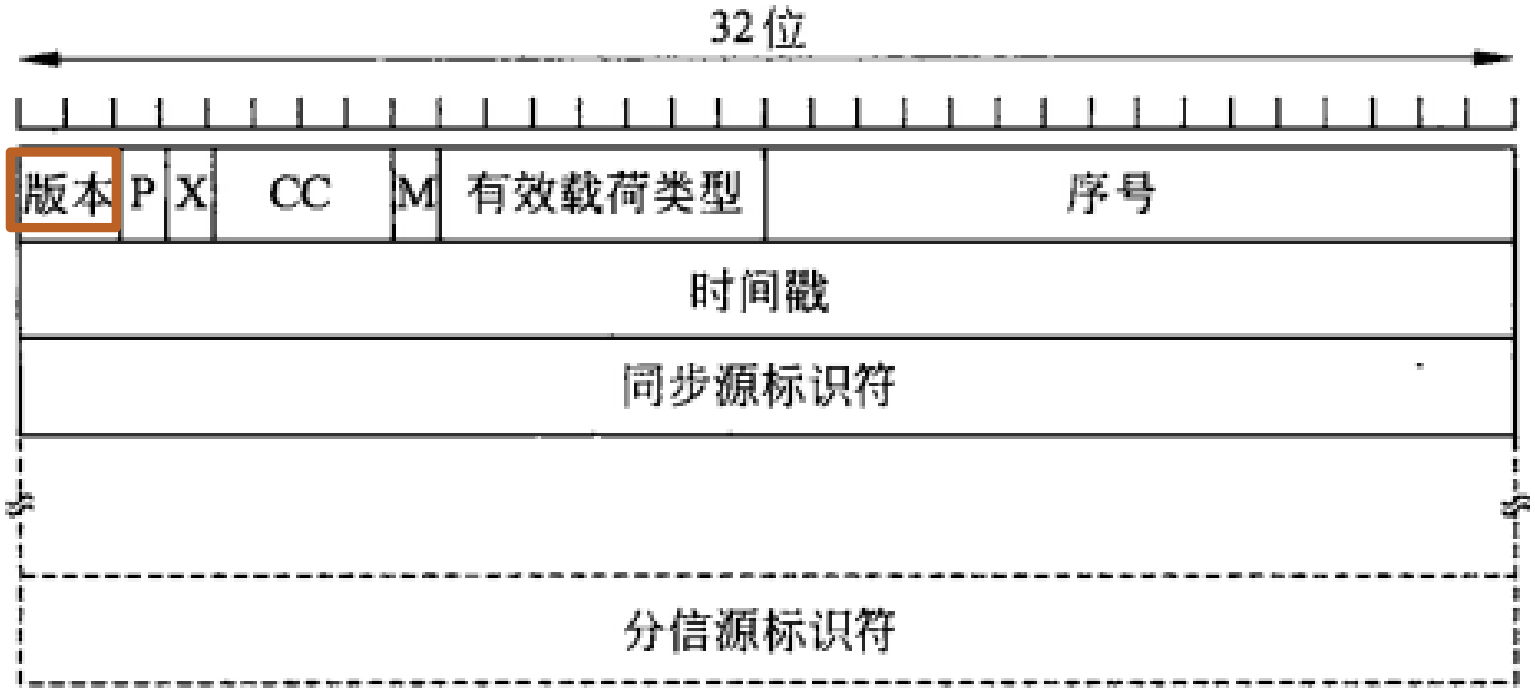


图 6-31　RTP 头格式

版本：版本号

6.4.3 实时传输协议

- **1.实时传输协议RTP**

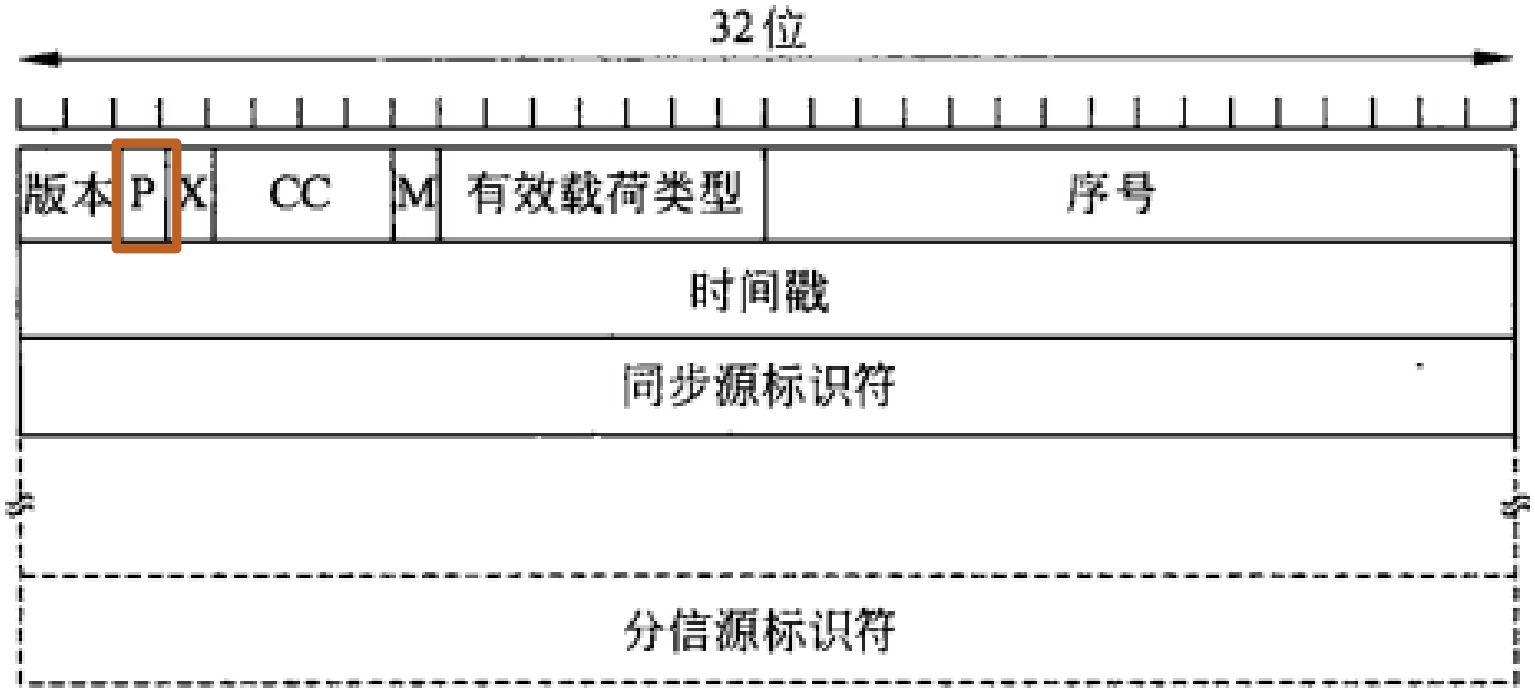


图 6-31　RTP 头格式

P位：表示数据包填充到整4字节倍数

6.4.3 实时传输协议

- **1.实时传输协议RTP**

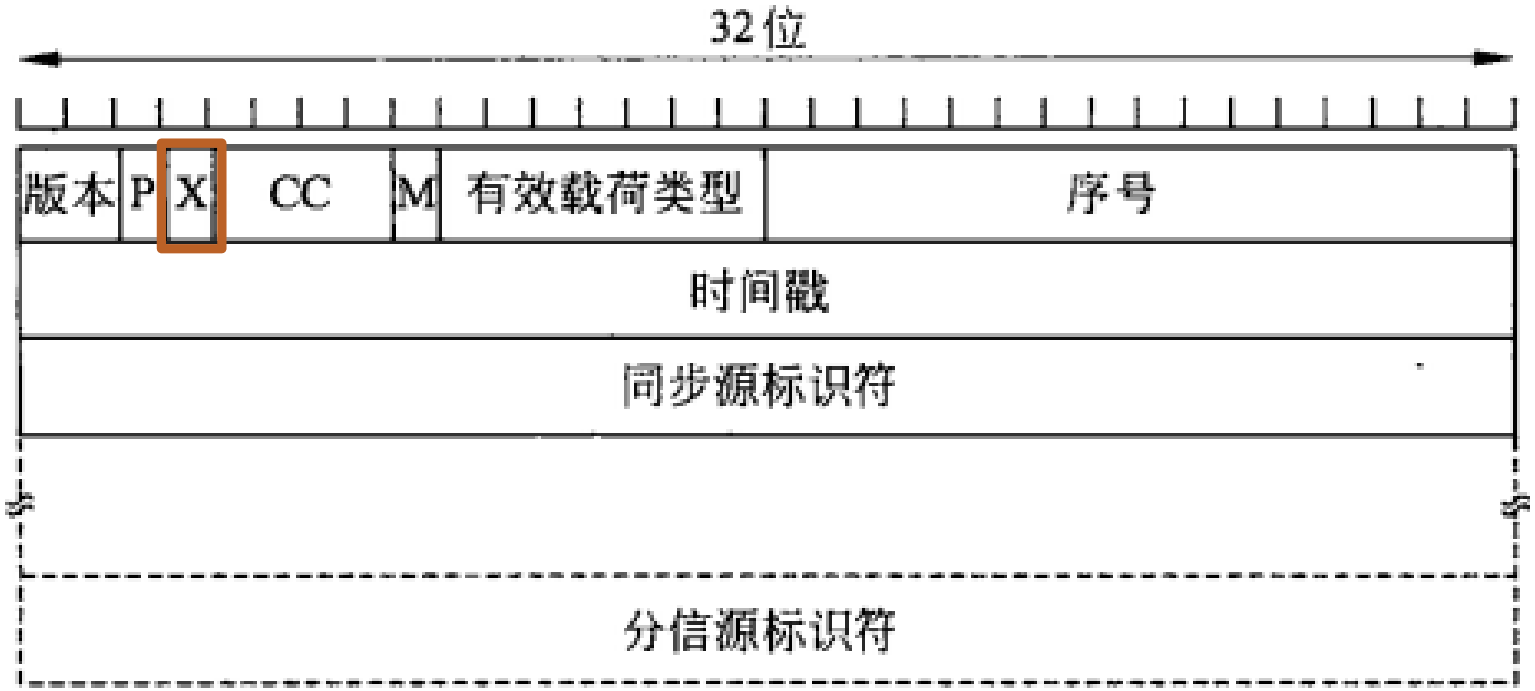


图 6-31　RTP 头格式

X位：标识有扩展头

6.4.3 实时传输协议

- **1.实时传输协议RTP**

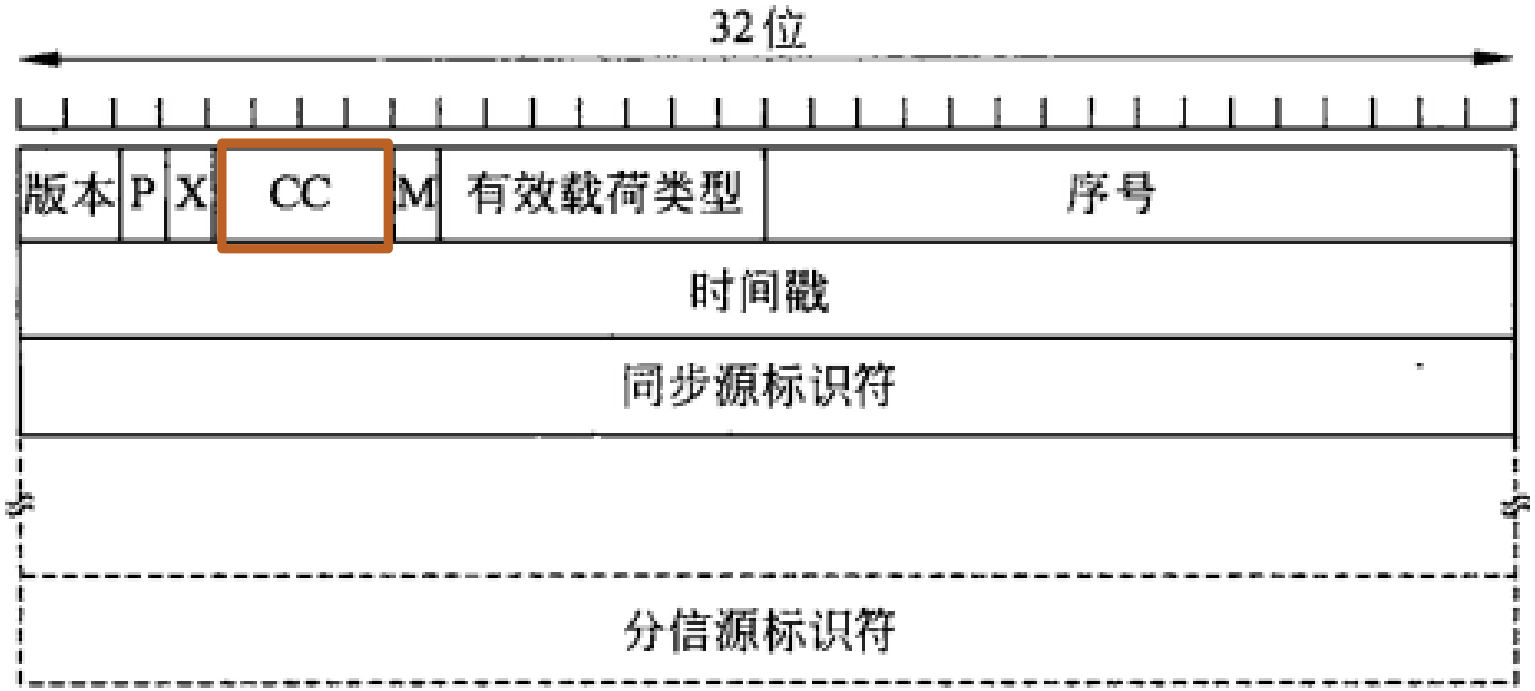


图 6-31　RTP 头格式

CC字段：贡献源数量，0—15

6.4.3 实时传输协议

- **1.实时传输协议RTP**

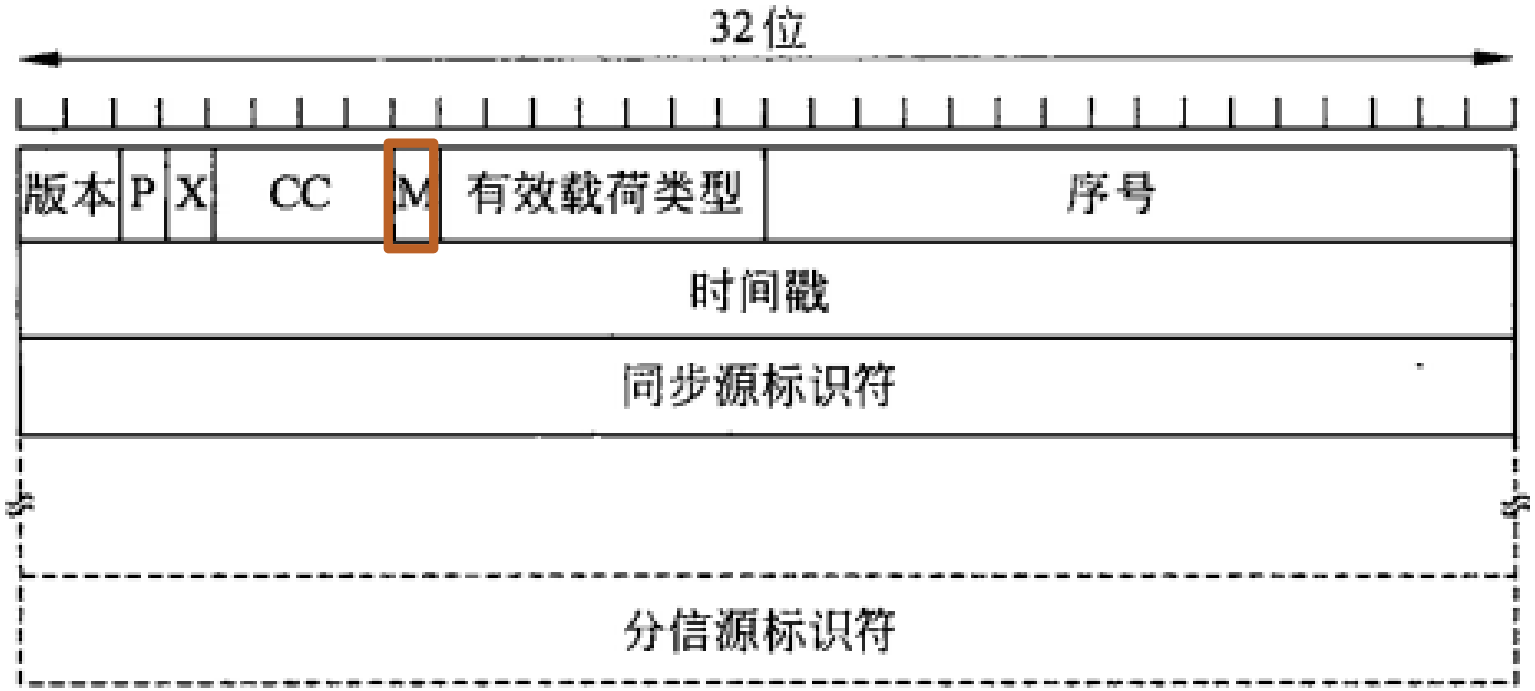


图 6-31　RTP 头格式

M位：应用相关标记

6.4.3 实时传输协议

- **1.实时传输协议RTP**

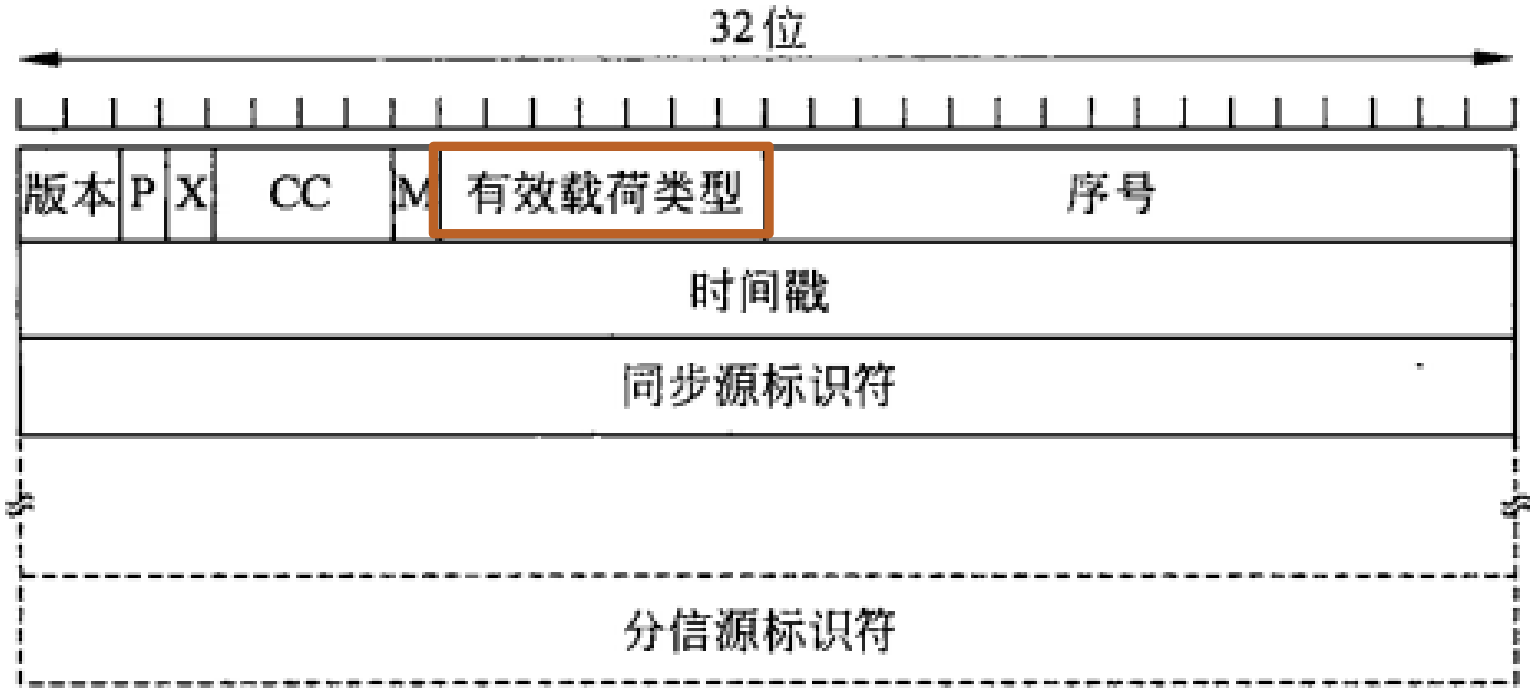


图 6-31　RTP 头格式

有效载荷类型：编码算法（应用类型）

6.4.3 实时传输协议

- **1.实时传输协议RTP**

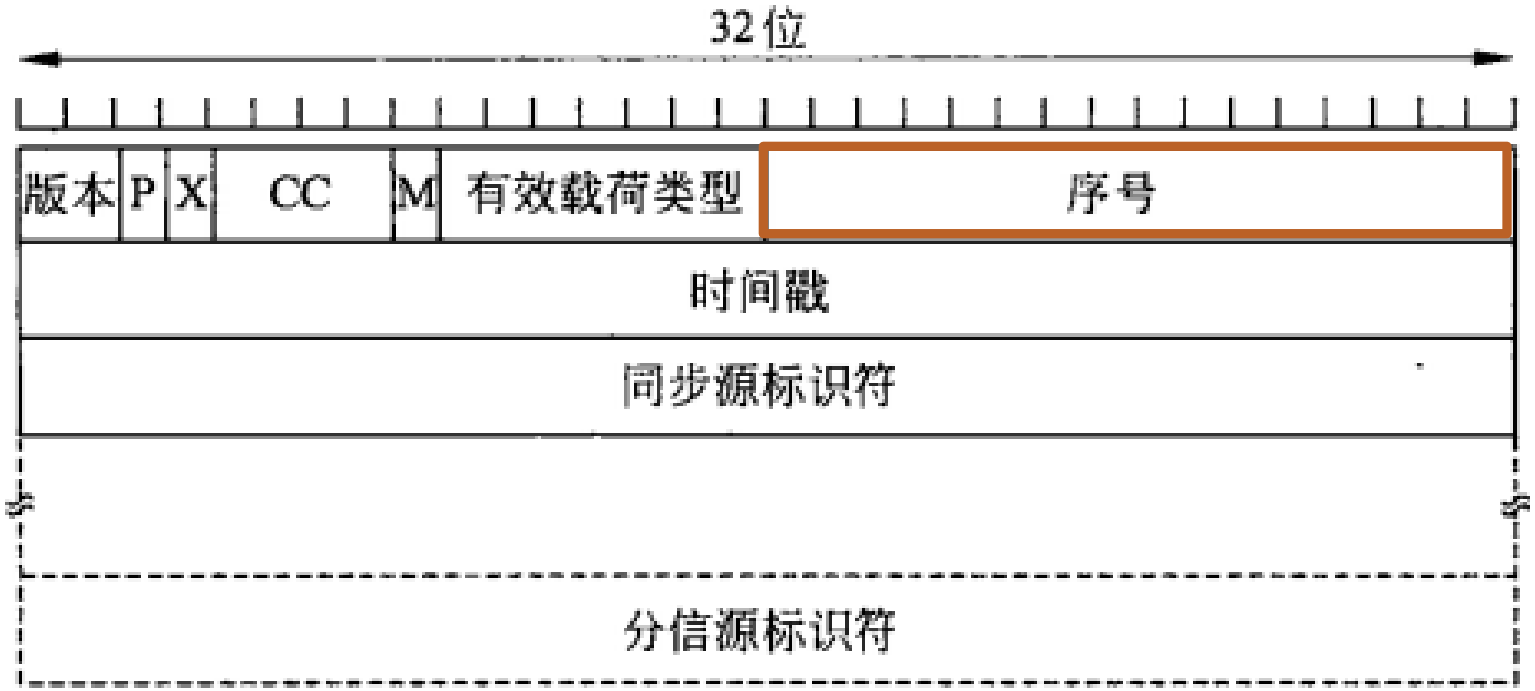


图 6-31　RTP 头格式

序号：数据包计数器，可用于检测包丢失

6.4.3 实时传输协议

- **1.实时传输协议RTP**

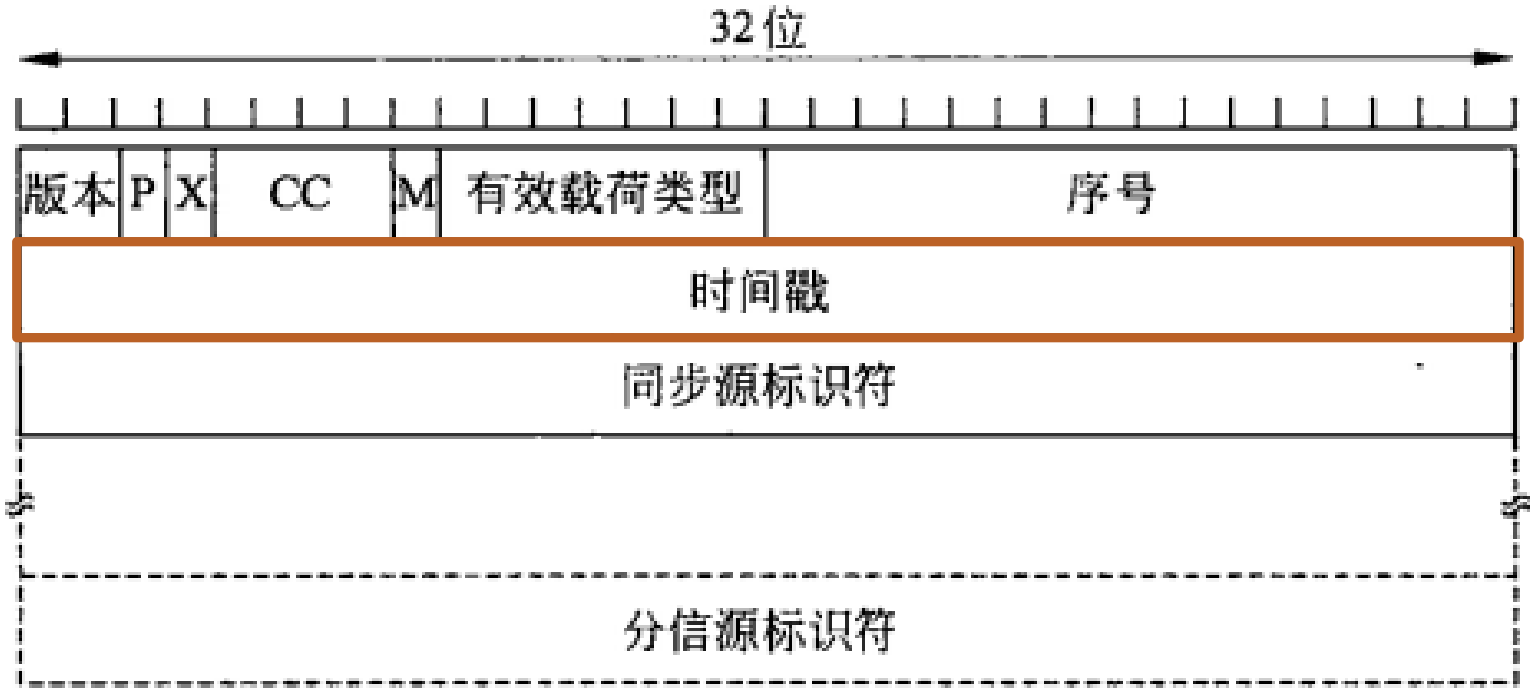


图 6-31　RTP 头格式

时间戳：第一个样本产生时间，用于缓解抖动

6.4.3 实时传输协议

- **1.实时传输协议RTP**

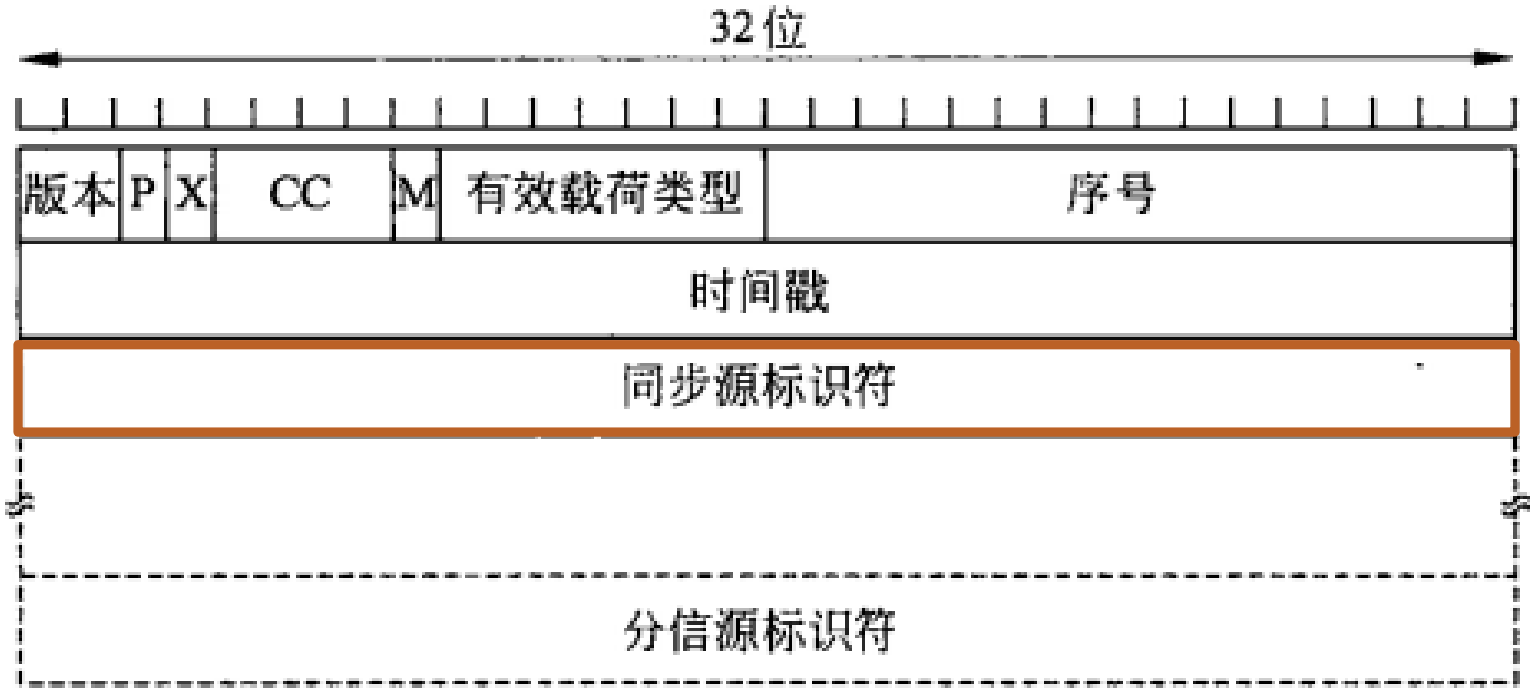


图 6-31　RTP 头格式

同步源标识符：指明数据包属于哪个数据流

6.4.3 实时传输协议

2.实时传输控制协议RTCP

RTCP是RTP的姐妹协议，增加了三个功能：

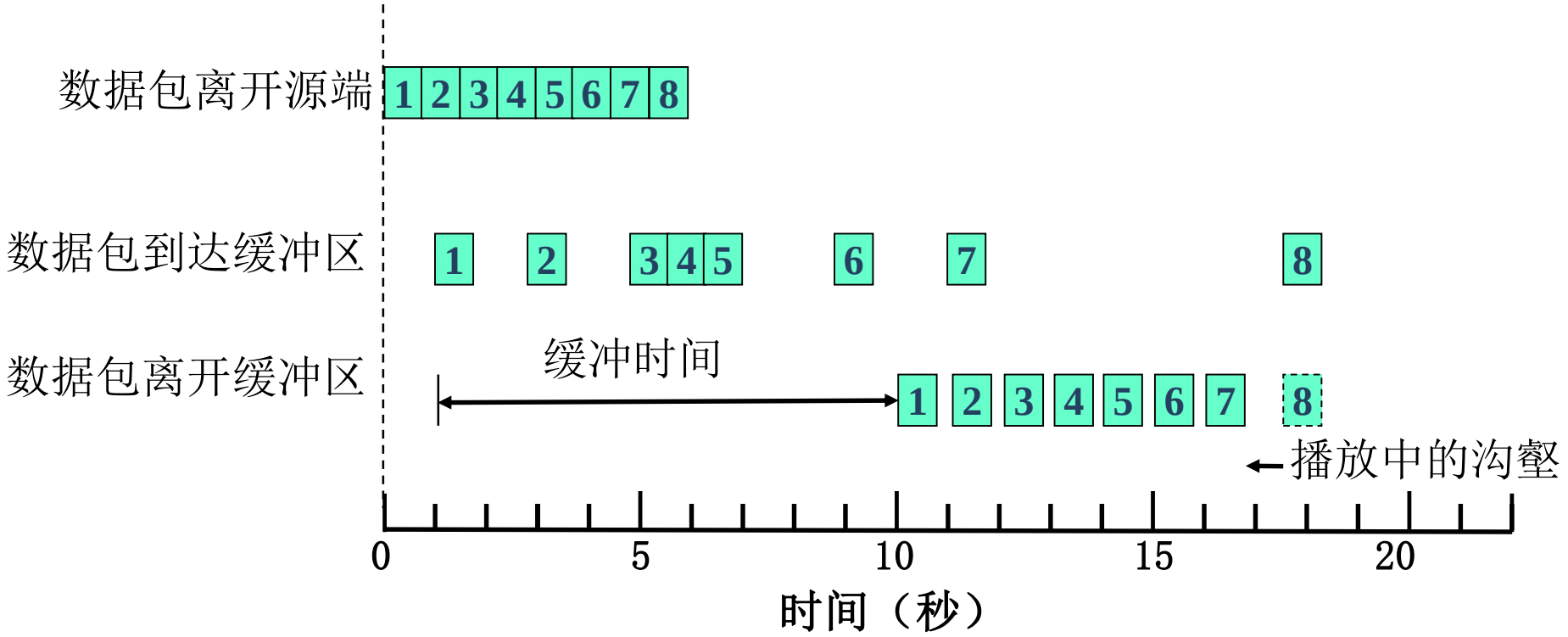
（1）提供网络特性信息

（2）处理流同步

（3）源命名管理

6.4.3 实时传输协议

- **3.带有缓冲和抖动控制的播放**



通过缓冲数据包来平滑输出流

6.4.3 实时传输协议

- 3.带有缓冲和抖动控制的播放

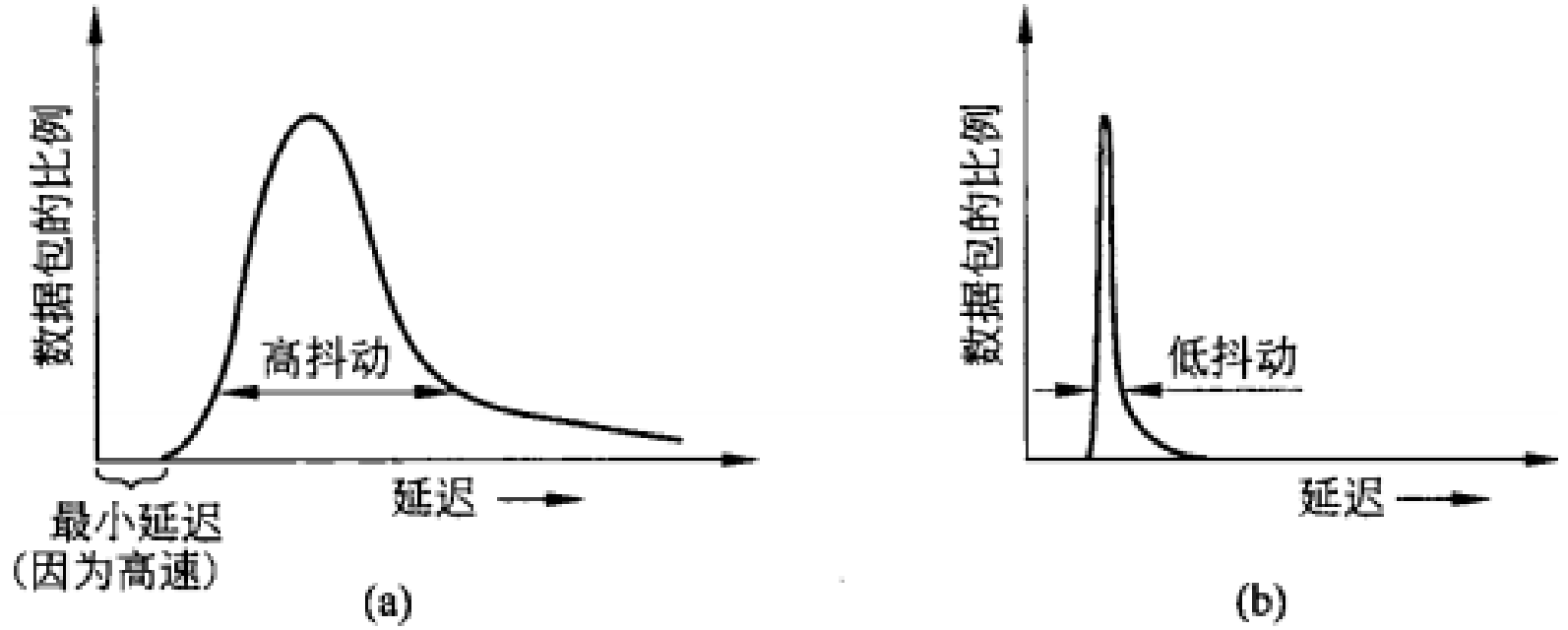
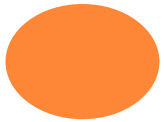


图 6-33

(a) 高抖动; (b) 低抖动

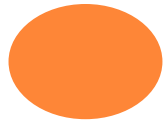
6.5 Internet传输协议：TCP

- TCP服务模型
- TCP数据段头
- TCP连接和释放管理
- TCP传输策略
- TCP拥塞控制
- TCP计时器管理



6.5 Internet传输协议：TCP

- TCP (Transmission Control Protocol) 传输控制协议是为了在不可靠的互联网络上，提供**面向连接**的、**可靠**的、**全双工**的、**端到端**的、**字节流**的通信服务。
- RFC793 /RFC1122 /RFC1323 /RFC2018 /RFC2581
- RFC2873 /RFC2988 /RFC3168 /RFC4614



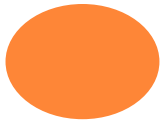
6.5.2 TCP服务模型

- TCP发送和接收端各由一对套接字（Socket）构成，一对套接字包括一个IP和一个端口

TCP 连接 ::= {socket1, socket2}

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

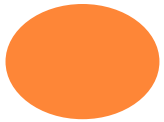
- 端口port是16位的数值，用于标识通信进程，知名端口（well-known port） 端口号小于1024，其他可由各主机自己定义



6.5.2 TCP服务模型

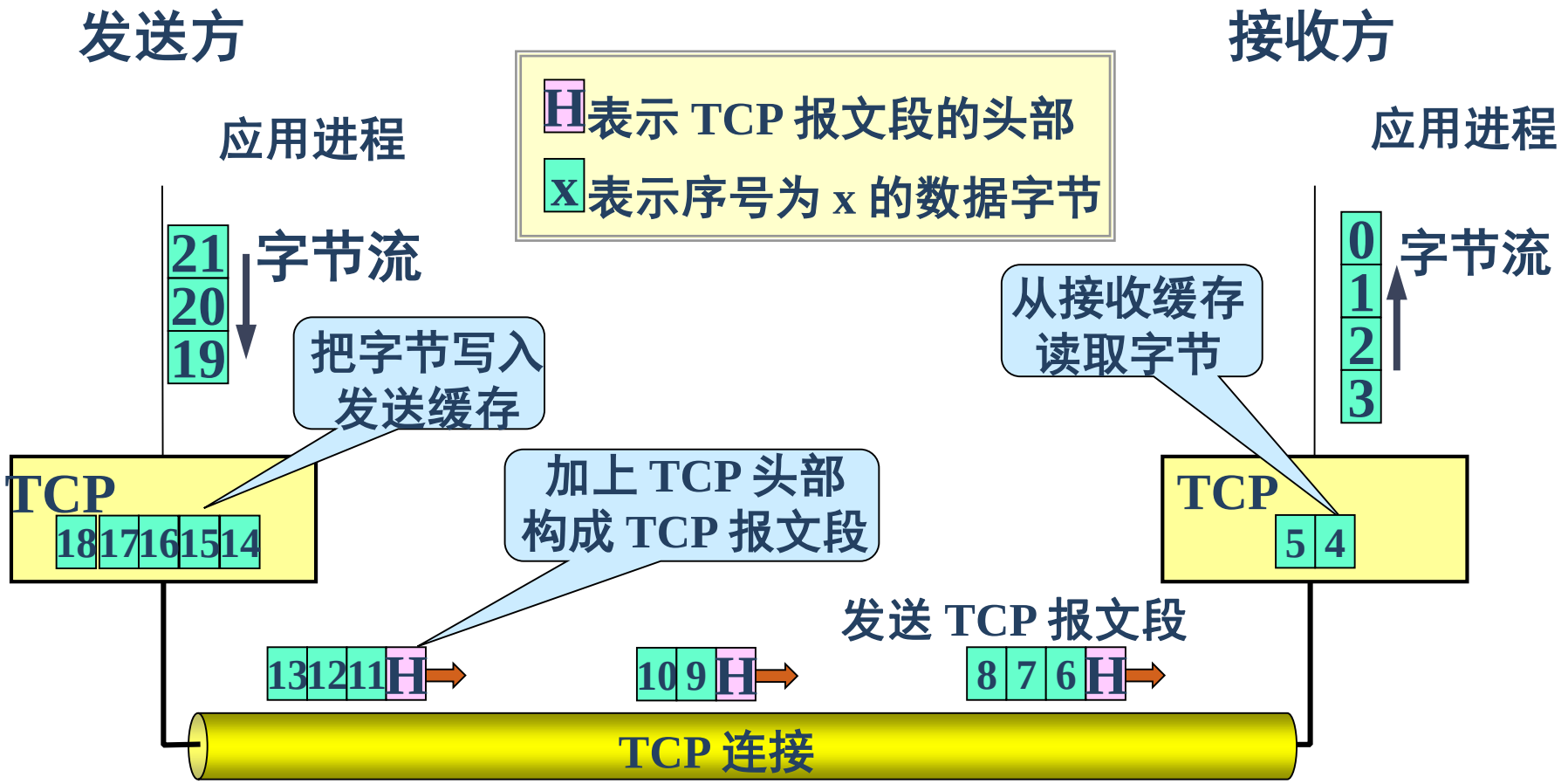
常用TCP端口表

端口号	关键字	服务说明
20,21	FTP	文件传输协议
23	TELNET	终端连接
25	SMTP	简单邮件传输协议
53	DNS	域名解析协议
80	HTTP	超文本传输协议
110	POP-3	远程E-mail访问
143	IAMP	访问远程邮件
443	HTTPS	安全WEB

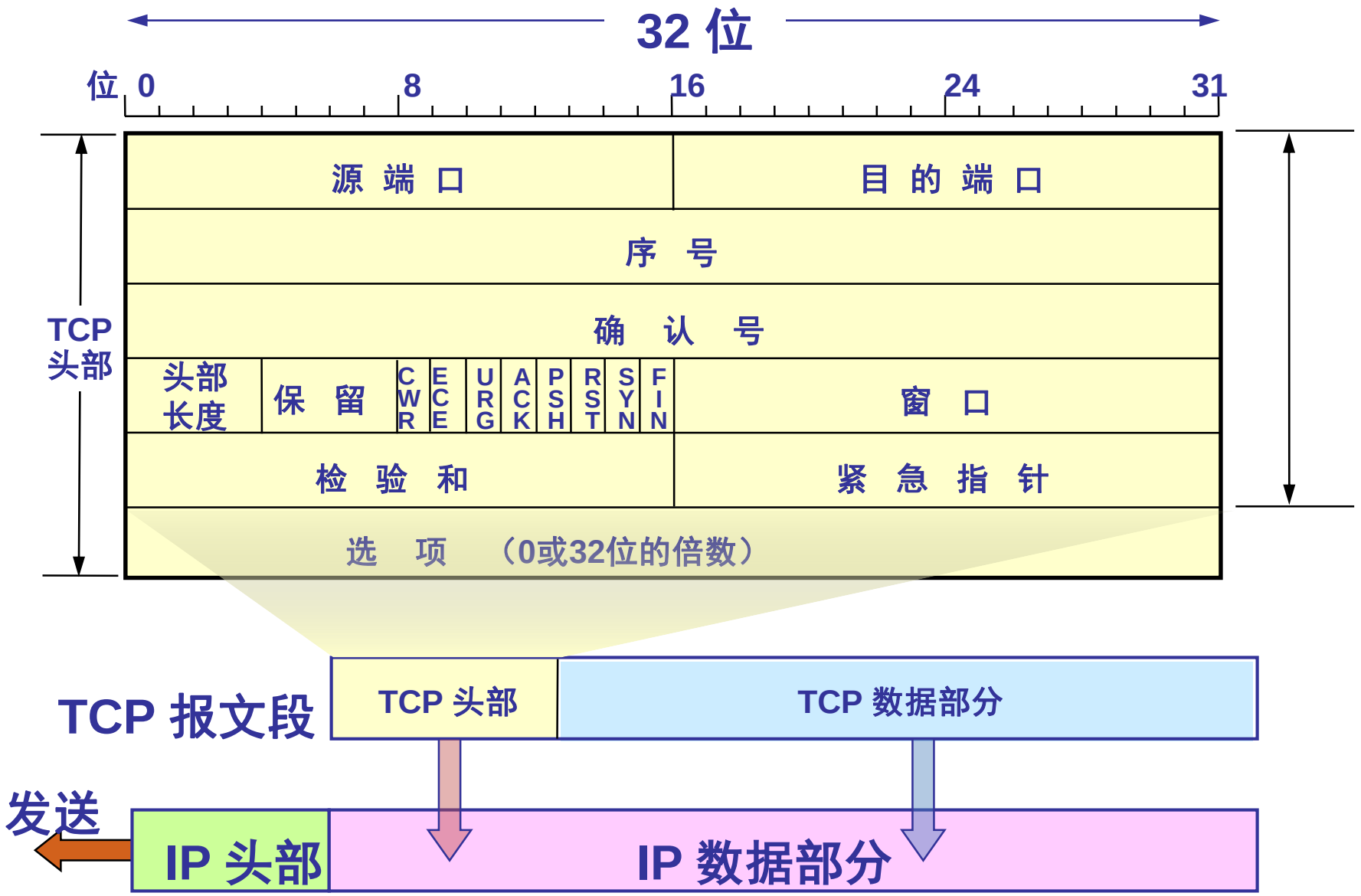


6.5.2 TCP服务模型

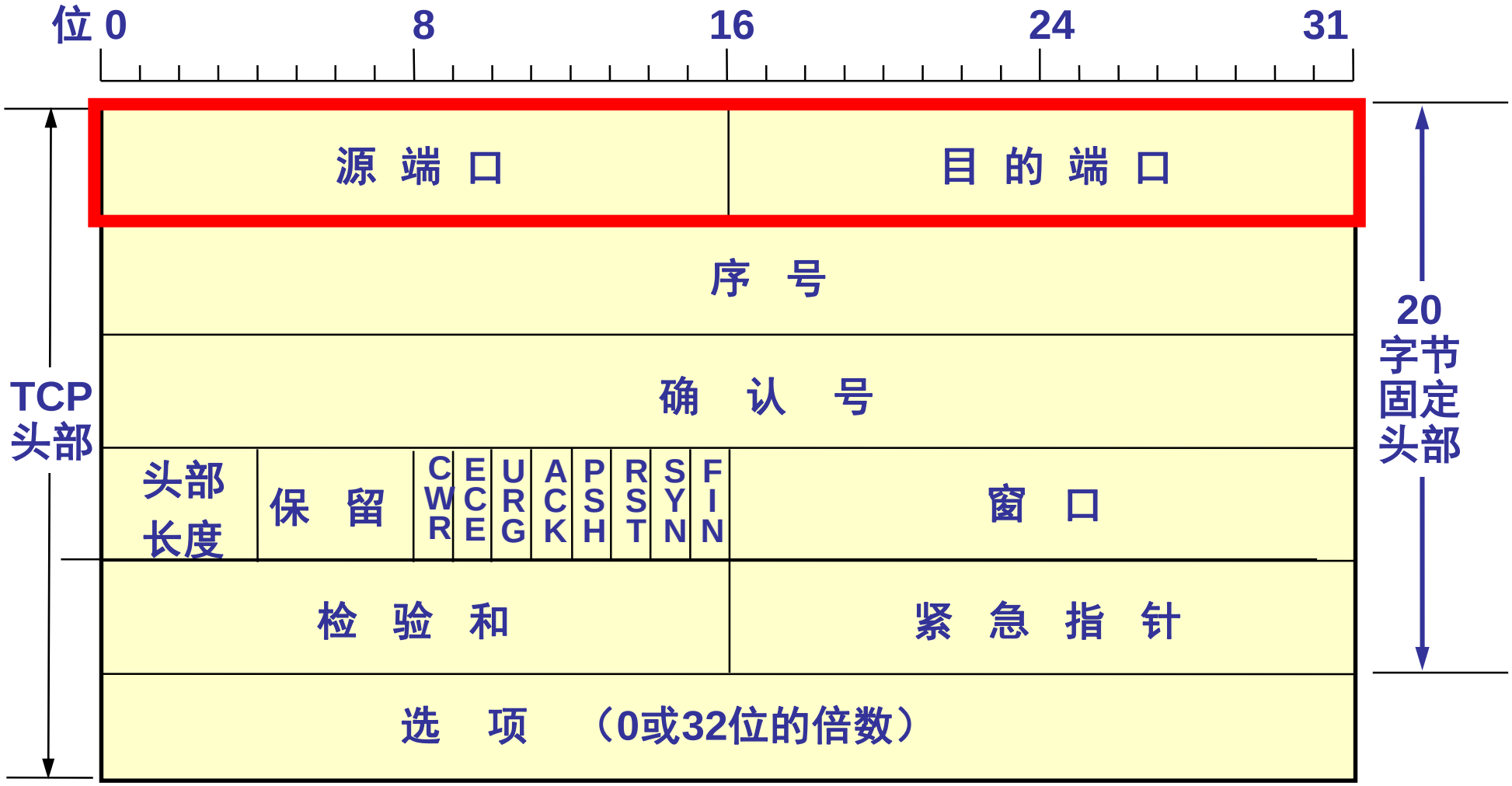
- TCP面向字节流，流是指流入到进程或从进程流出的字节序列。不管应用进程交付什么样的数据，TCP协议都看成是无结构的字节流



6.5.4 TCP头段


32 位
位 0 8 16 24 31
源 端 口
目 的 端 口
序 号
确 认 号
头部长度
保 留
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （0或32位的倍数）
TCP 头部
TCP 报文段
TCP 头部
TCP 数据部分
发送
IP 头部
IP 数据部分

6.5.4 TCP头段



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。

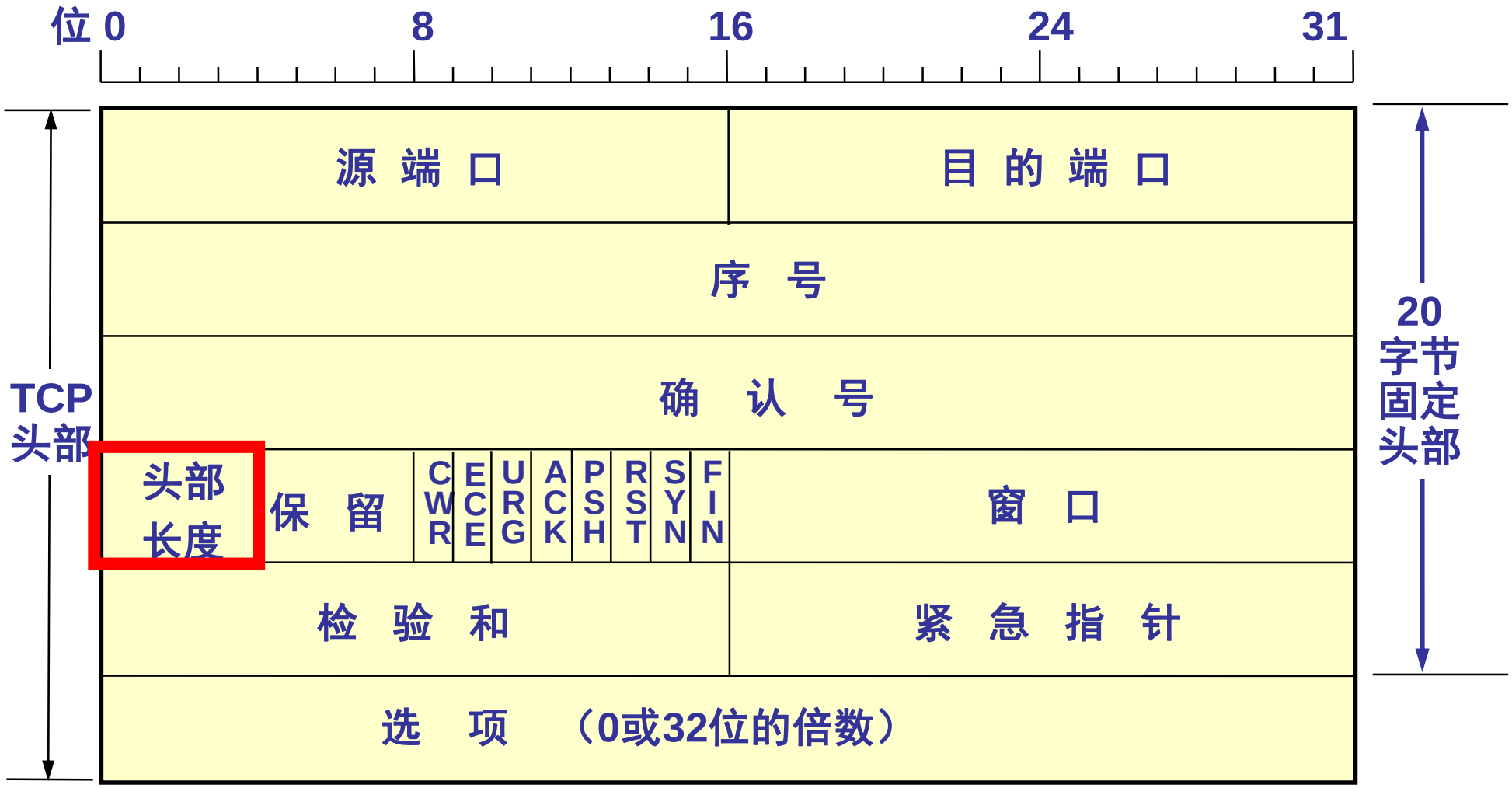
6.5.4 TCP头段

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。

6.5.4 TCP头段

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。

6.5.4 TCP头段



头部长度——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”的单位是 32 位字（以 4 字节为计算单位）。

6.5.4 TCP头段

保留字段——占 4 位，保留为今后使用，但目前应置为 0。

6.5.4 TCP头段

CWR—— 使用显式拥塞通知ECN时的拥塞控制信号，发送端设置。

6.5.4 TCP头段

ECE——使用显式拥塞通知ECN时的拥塞控制信号，接收端设置。

6.5.4 TCP头段

紧急 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)。

6.5.4 TCP头段

确认 ACK —— 只有当 ACK = 1 时确认号字段才有效。

当 ACK = 0 时，确认号无效。

6.5.4 TCP头段

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

6.5.4 TCP头段

复位 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，然后再重新建立运输连接。

6.5.4 TCP头段

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

6.5.4 TCP头段

终止 FIN (FINis) —— 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。

6.5.4 TCP头段

窗口字段 —— 占 2 字节，用来让对方设置发送窗口的依据，单位为字节。

检验和 —— 占 2 字节。检验和字段检验的范围包括头部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪头部。

紧急指针字段 —— 占 16 位，指出在本报文段中紧急数据共有多少个字节（紧急数据放在本报文段数据的最前面）。

6.5.4 TCP头段

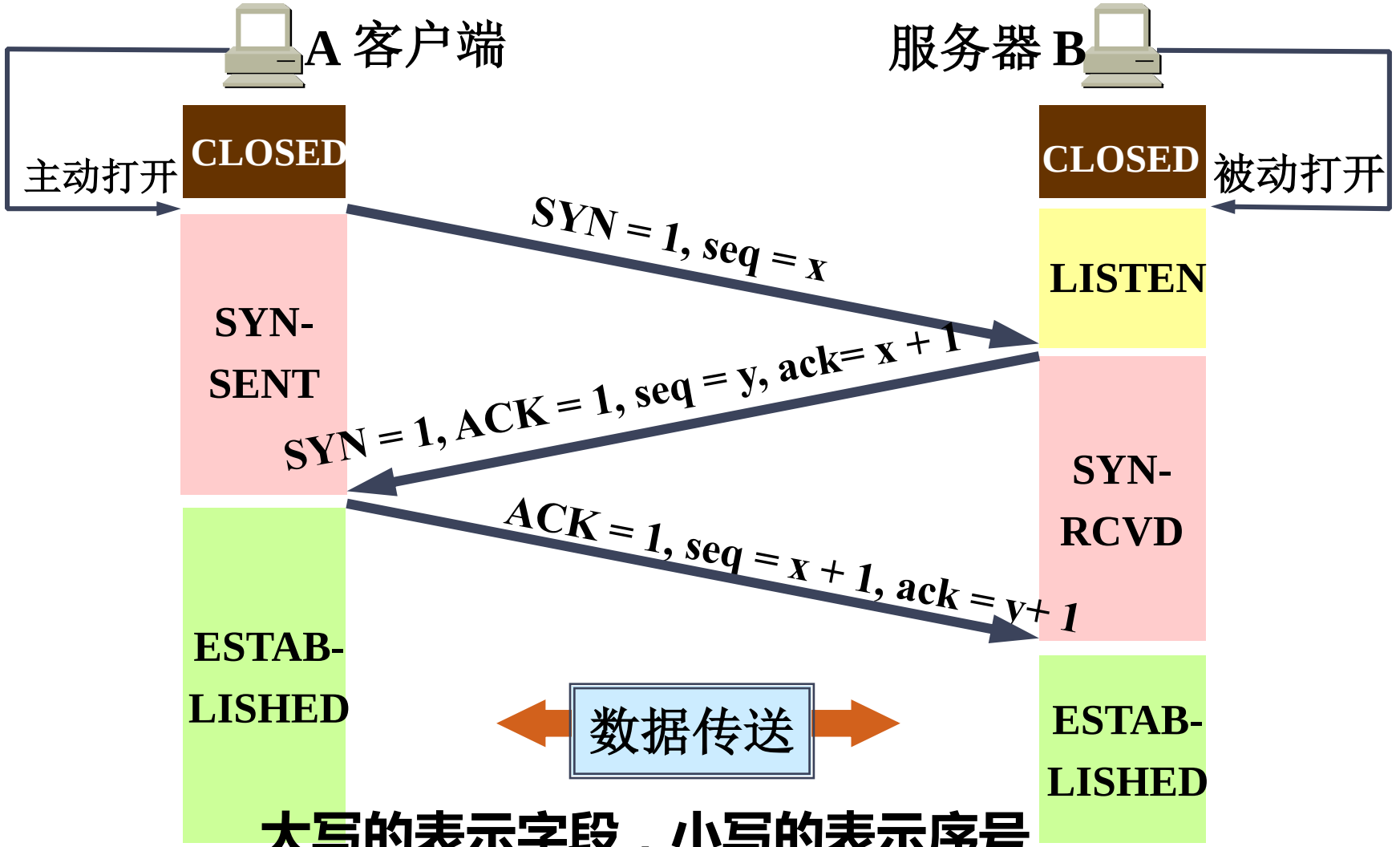
选项字段 —— 长度可变。TCP 最初只规定了一种选项，即**最大报文段长度** MSS。MSS 告诉对方 TCP：“我的缓存所能接收的报文段的数据字段的最大长度是 MSS 个字节。”

6.5.4 TCP头段

- 其他选项
- 窗口扩大选项 ——占 3 字节，其中有一个字节表示移位值S。新的窗口值等于TCP 头部中的窗口位数增大到(16 + S)，相当于把窗口值向左移动 S 位后获得实际的窗口大小。
- 时间戳选项——占10 字节，其中最主要的字段是时间戳值字段（4 字节）和时间戳回送回答字段（4 字节）。
- 选择确认选项——接收端告诉发送端已经收到的序号范围，从而重传部分数据。

6.5.5 TCP连接建立

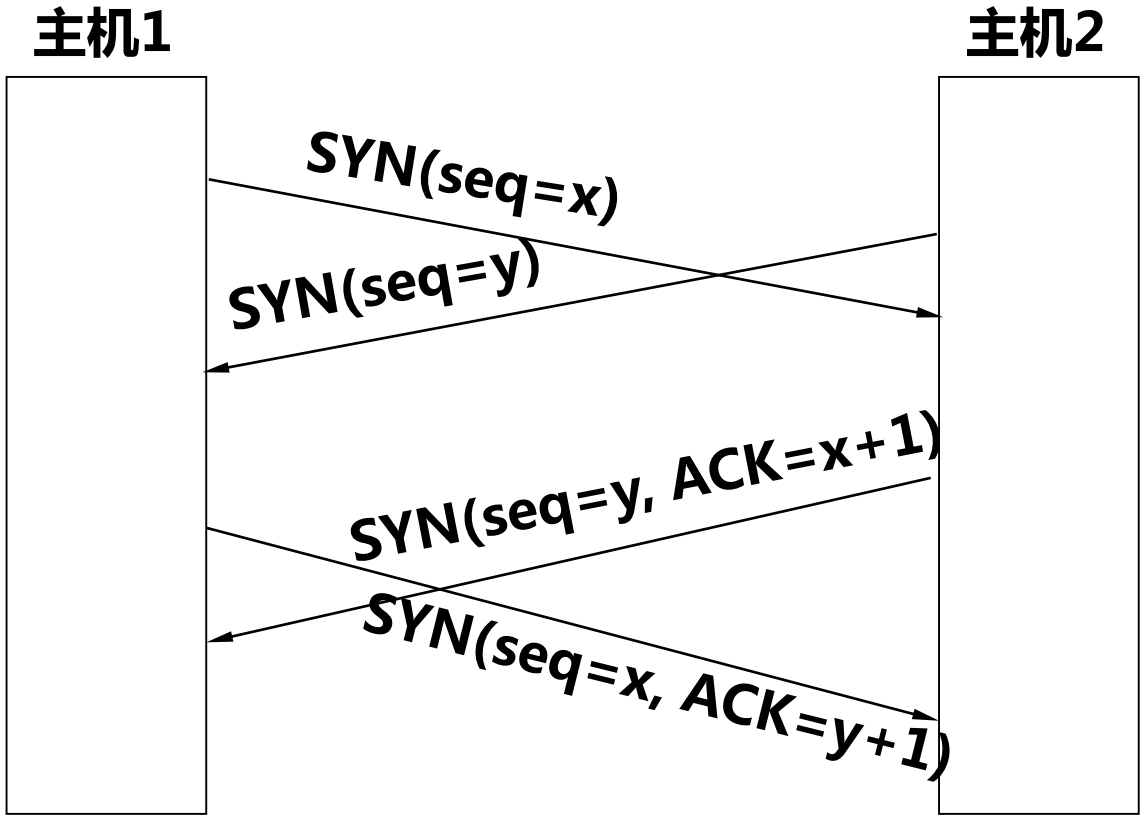
三次握手（3-way handshake）建立连接，用TCP包头中的同步位（SYN）和结束位（FIN）描述连接建立和终止时的三次握手消息



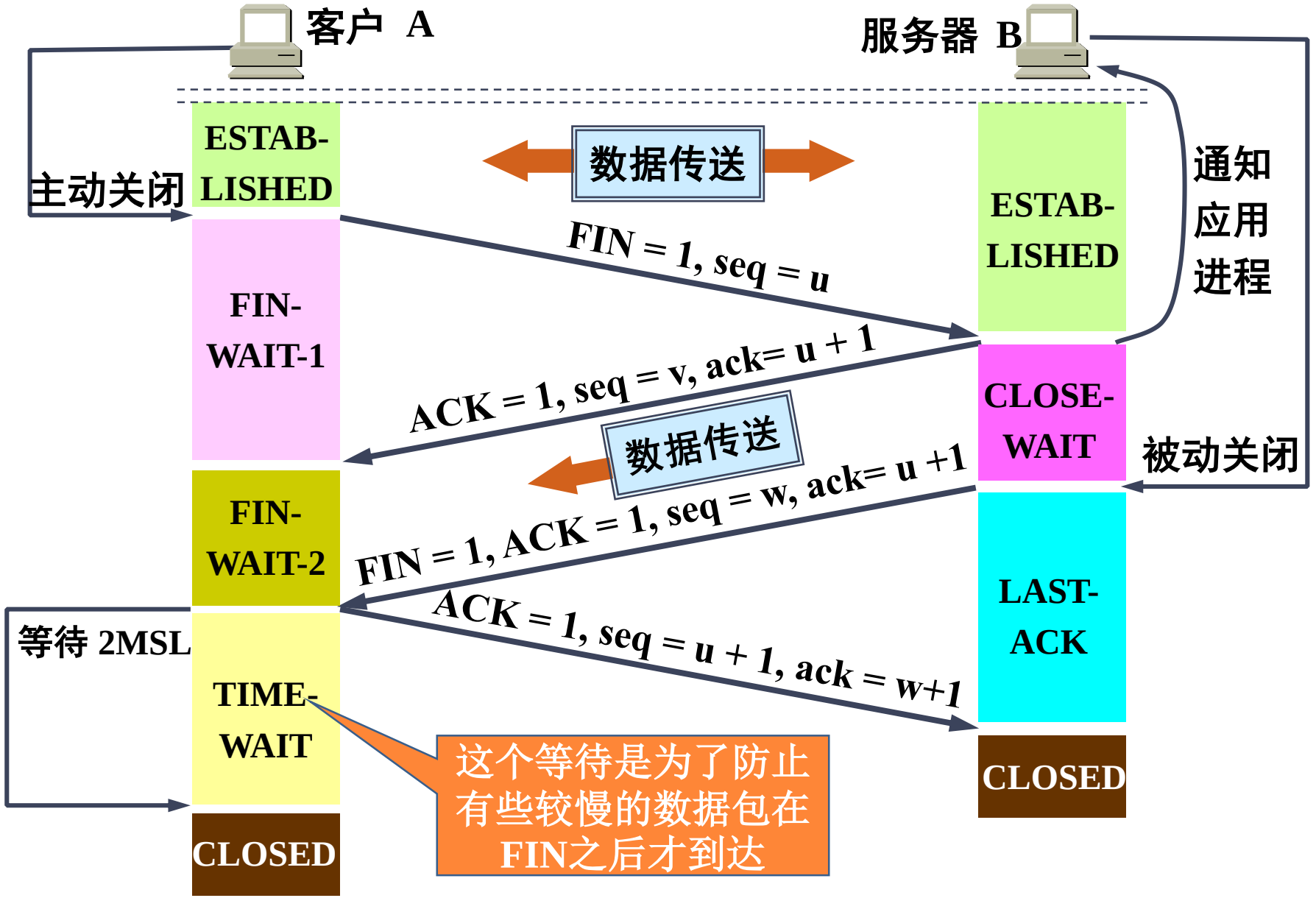
6.5.5 TCP连接建立

连接以端地址-端地址为标识

即使两边各自发起一个连接，也不会建立两个连接。最终建立的连接仍然是（x，y）



6.5.5 TCP连接释放

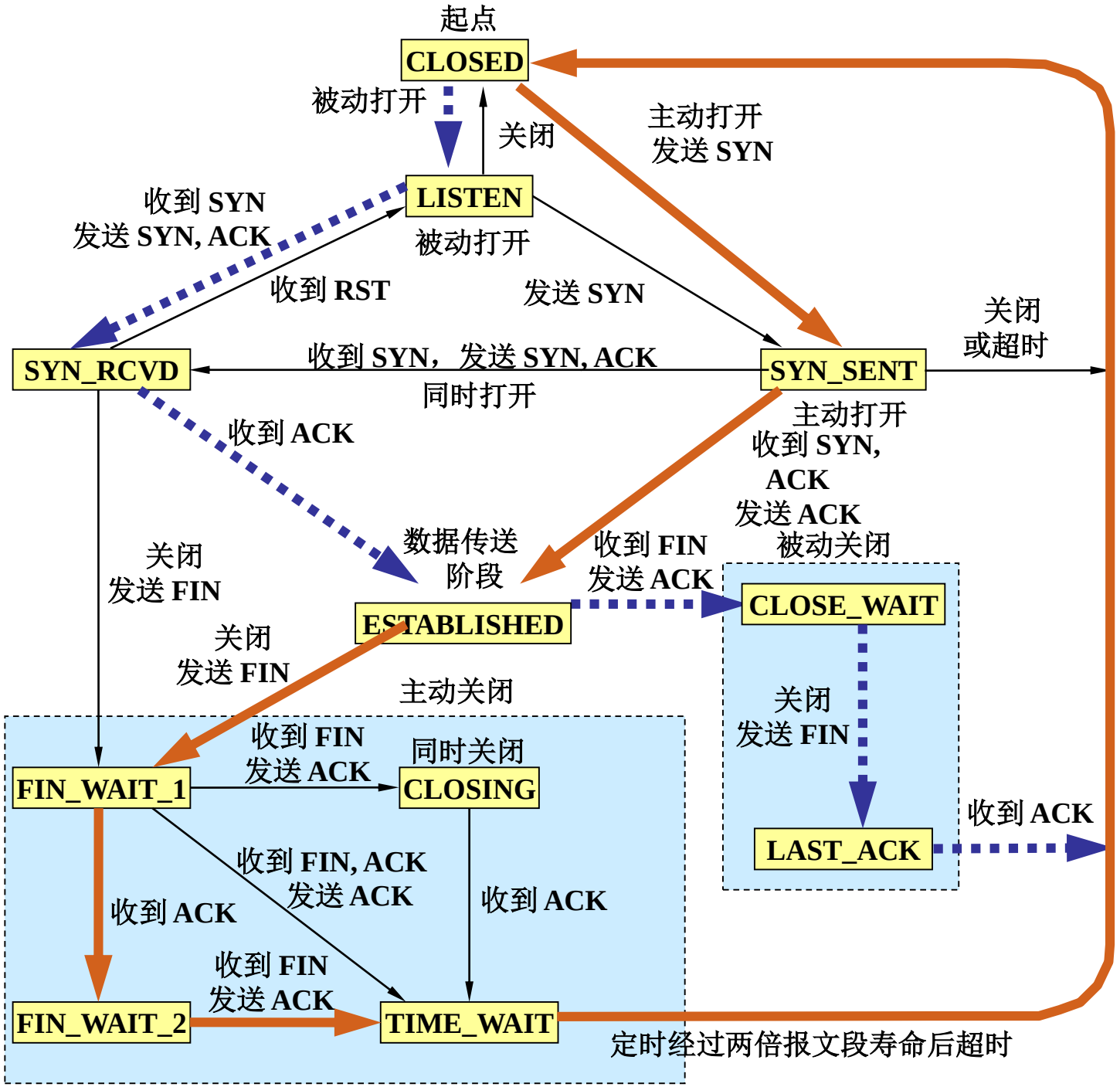

客户 A
服务器 B
ESTAB-LISHED
数据传送
ESTAB-LISHED
主动关闭
通知应用进程
FIN = 1, seq = u
FIN-WAIT-1
ACK = 1, seq = v, ack= u + 1
CLOSE-WAIT
数据传送
被动关闭
FIN = 1, ACK = 1, seq = w, ack= u +1
FIN-WAIT-2
LAST-ACK
ACK = 1, seq = u + 1, ack = w+1
等待 2MSL
TIME-WAIT
CLOSED
这个等待是为了防止有些较慢的数据包在FIN之后才到达
CLOSED

6.5.7 TCP连接管理模型

状态	描述
CLOSED	没有活跃的连接或者挂起
LISTEN	服务器等待入境呼叫
SYN RCVD	到达一个连接请求；等待ACK
SYN SENT	应用已经启动了打开一个连接
ESTABLISHED	正常的数据传送状态
FIN WAIT1	应用没有数据要发了
FIN WAIT2	另一端同意释放连接
TIME WAIT	等待所有数据包寿终正寝
CLOSING	两端同时试图关闭连接
CLOSE WAIT	另一端已经发起关闭连接
LAST ACK	等待所有数据包寿终正寝

图 6-38　TCP 连接管理有限状态机使用的状态

TCP连接管理的有限状态机

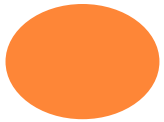


6.5.8 TCP滑动窗口

- TCP使用与数据链路层不同的窗口协议来实现流量控制

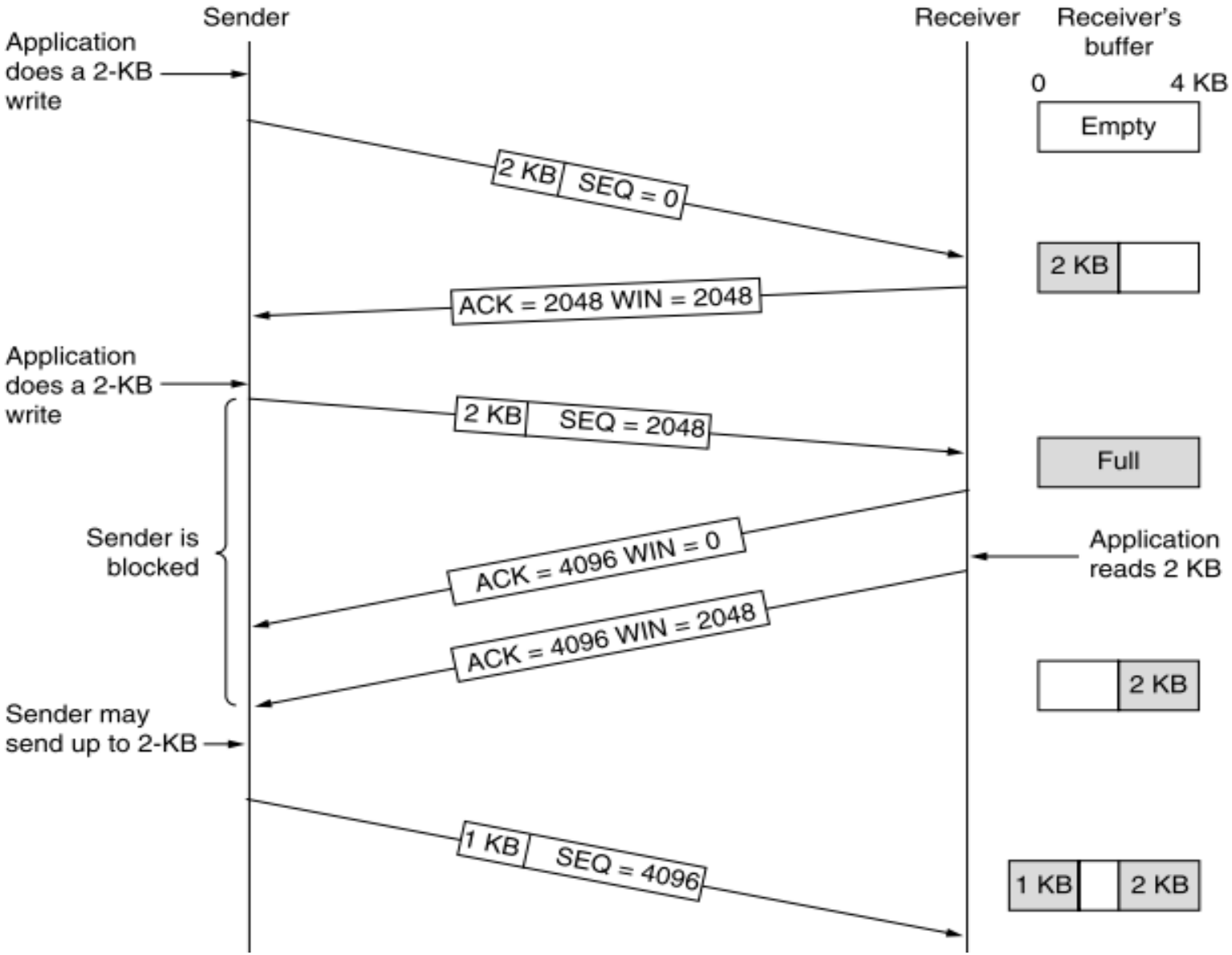
- 数据链路层：在数据链路层的滑动窗口协议中，发送窗口的滑动依赖于确认的到达

- TCP层：TCP的窗口大小是其缓冲区的尺寸，建立连接的双方都将分配一个缓冲区作为接收数据的存放空间，并相互通知对方，此后，每次对接收数据确认的同时发布一个报文报告剩余窗口的大小，任何时刻，剩余的缓冲区空间的数量叫窗口大小



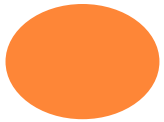
6.5.8 TCP滑动窗口

TCP的窗口管理



6.5.8 TCP滑动窗口

- 发送方收到一个零窗口为0的报文时，必须停止发送，直到接收方重新发送一个非零的窗口报文，但有两种情况除外
  - 发送紧急数据，如允许用户终止（kill）当前正在远端机上运行的进程
  - 发送方可以发送一个字节的数据段通知对方，要求接收方重申希望接收的下一字节及当前的窗口大小，以防止可能的窗口公告丢失而导致的死锁



6.5.9 TCP计时器管理

- TCP最重要的计时器是重传计时器RTO。每发送一个段时，同时启动一个RTO。如果RTO超时前被确认，则停止。否则重传。则RTO的时间间隔如何设置？

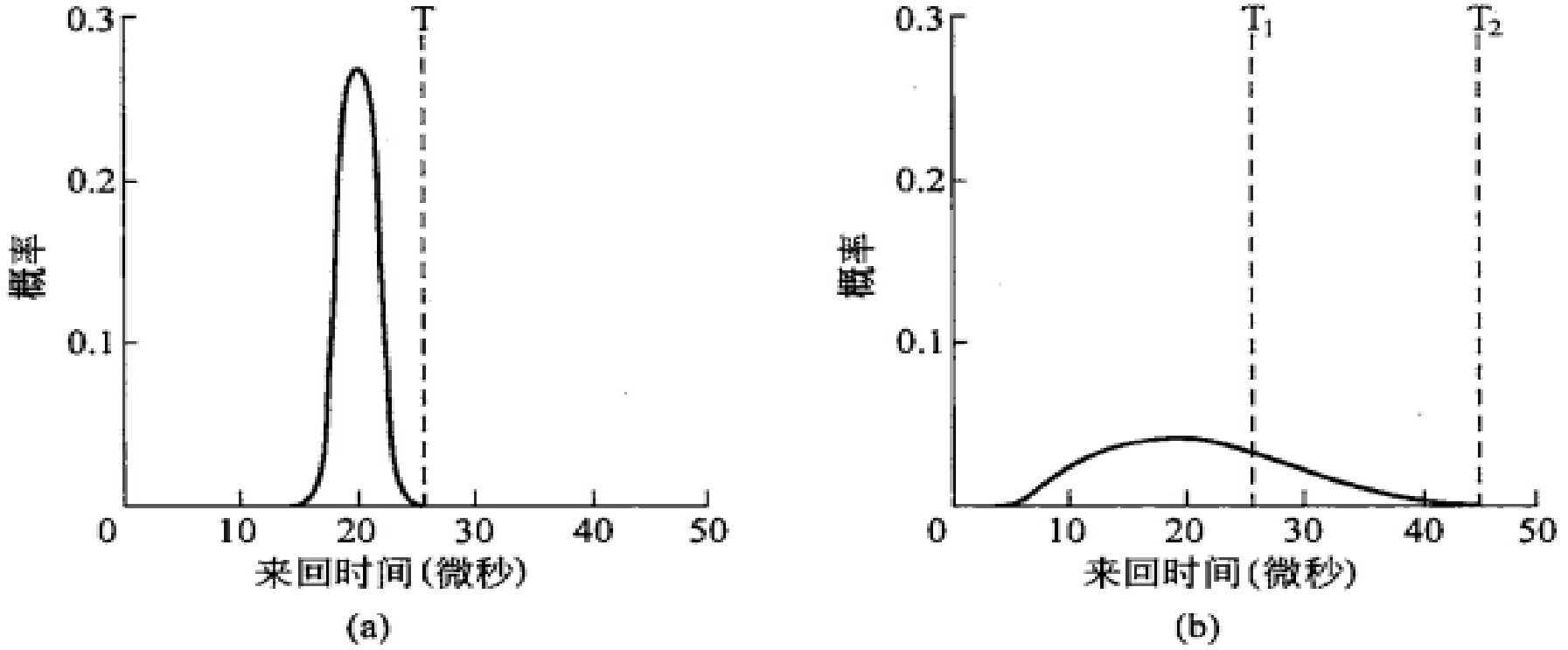


图 6-42

(a) 数据链路层的确认到达时间的概率密度；(b) TCP 确认到达时间的概率密度

6.5.9 TCP计时器管理

- RTT：往返时间（Round-Trip Time）
- SRTT:平滑往返时间(Smooth Round-Trip Time)
- RTTVAR：往返时间变化（ Round-Trip Time VARiation）
- SRTT: = α× SRTT + (1 − α) × R

新的 SRTT: = α×（旧的 SRTT）+（1 − α）×（新的 RTT 样本）

$$RTO = SRTT + 4 \times RTTVAR$$

RTTVAR = β×（旧的RTTVAR）+（1 − β） × |SRTT − R|

6.5.10 TCP拥塞控制

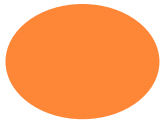
- 数据的丢失有两种情况

  1.接收方的容量太小（接收缓冲区）

  2.网络的容量太小（网络带宽，路由器缓存等）

  当负载超过网络处理能力就会发生拥塞，导致数据包丢失，而TCP的重发机制进一步加剧了拥塞

- TCP协议通过两个窗口控制发送速度：拥塞窗口和流量控制窗口。窗口的大小动态调整。一旦发现数据包丢失，则降低发送数据包的速率。两个窗口取较小的值。



6.5.10 TCP拥塞控制

- 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，以便把更多的数据发送出去。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数据（段数量）。
- 早期的TCP协议采用慢速启动和加法递增的机制（4.2BSD TCP Tahoe版本）。后来加上了快速恢复机制（4.3BSD TCP Reno版本）。

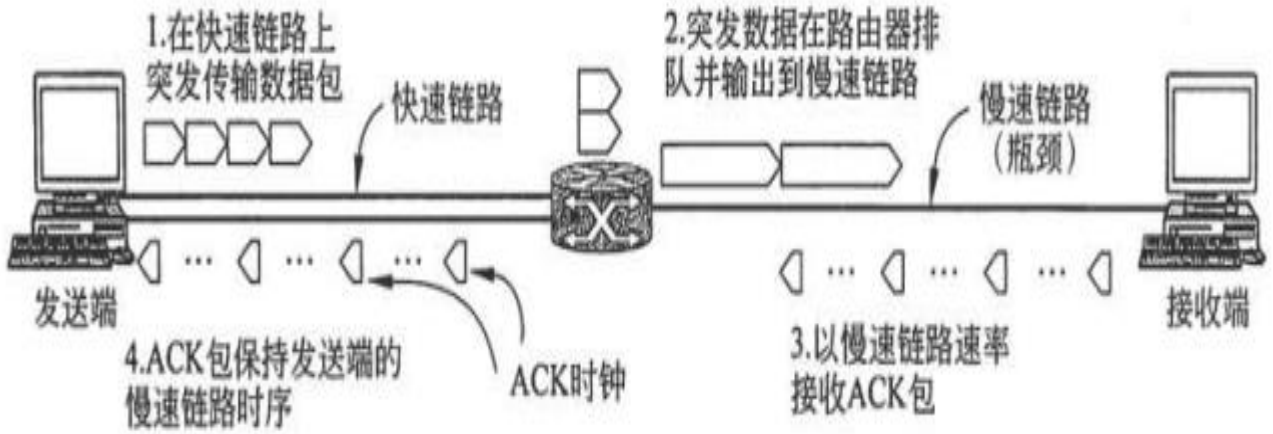
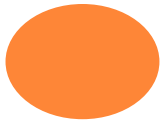


图 6-43　发送端的突发数据包以及返回的确认时钟

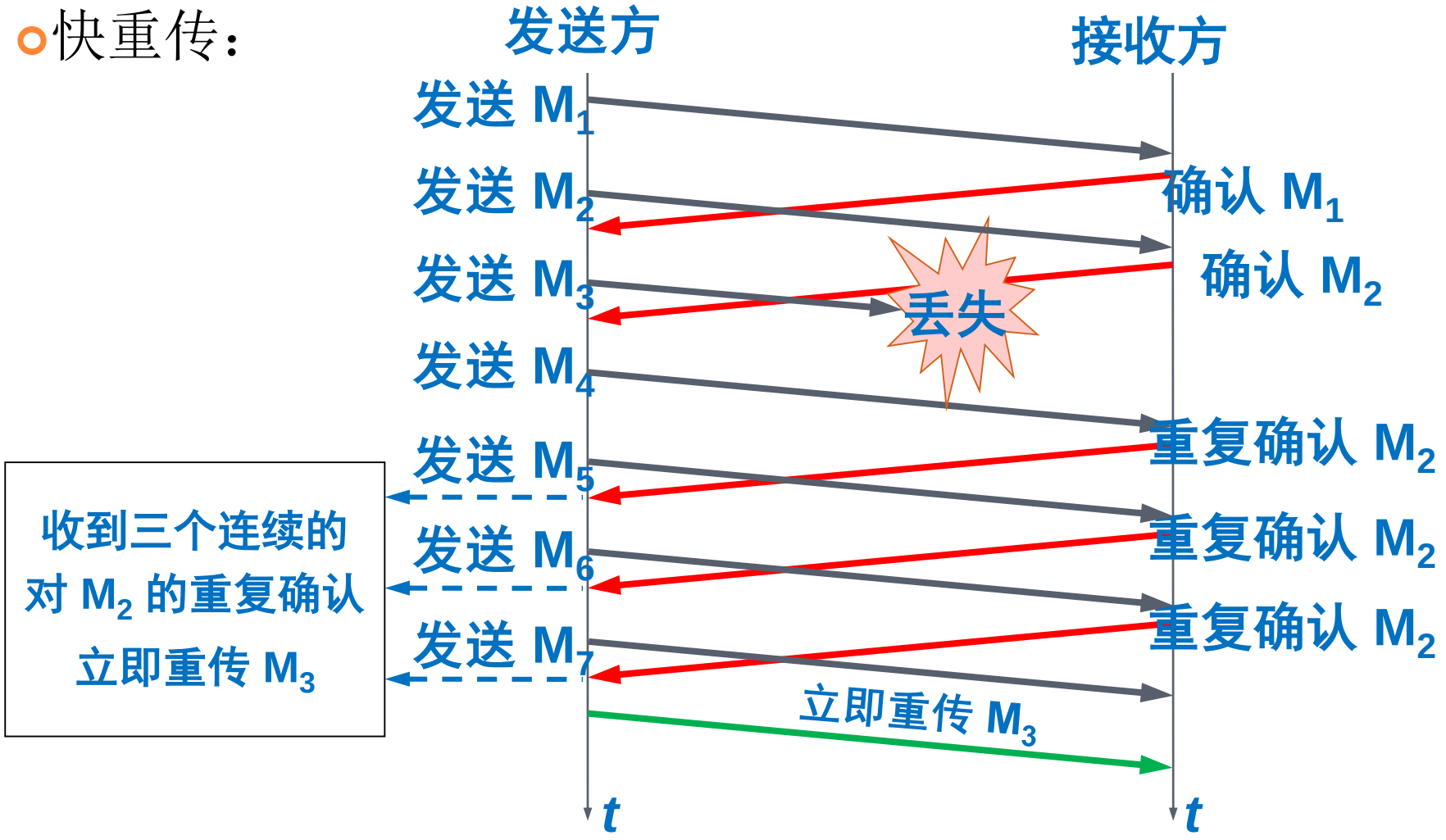
6.5.10 TCP拥塞控制

- 连接建立时，发送方用一个很小的值初始化拥塞窗口（1或者4个段），然后发送端按照该窗口大小发送数据；
- 对于每个确认的段，发送端窗口再增加一个段的字节量，即正常确认后（每个往返时间/轮次），窗口值加倍。该阶段称为慢速启动，实际按照指数增长的，并不慢（慢只是相对于后面的“归零”操作）
- 为了控制慢速启动窗口的增长速度，发送端为每个连接设置了一个慢启动阈值。开始这个阈值可以设置得较高。当发送端的窗口大小达到阈值时，TCP拥塞窗口的增长方式变为线性增加（加法递增），即每个往返时间只增加一个数据单位（段）。
- 当检测到丢包时，慢启动阈值设置为当前拥塞窗口的一半，拥塞窗口恢复到初始值，然后再慢启动。
- TCP规定当收到三个重复确认，认为已经丢失一个包。发送端将立即重传该数据包（即快速重传）。重传后，慢启动的阈值降为当前拥塞窗口的一半，并重新开始慢启动过程，拥塞窗口恢复到初始值。



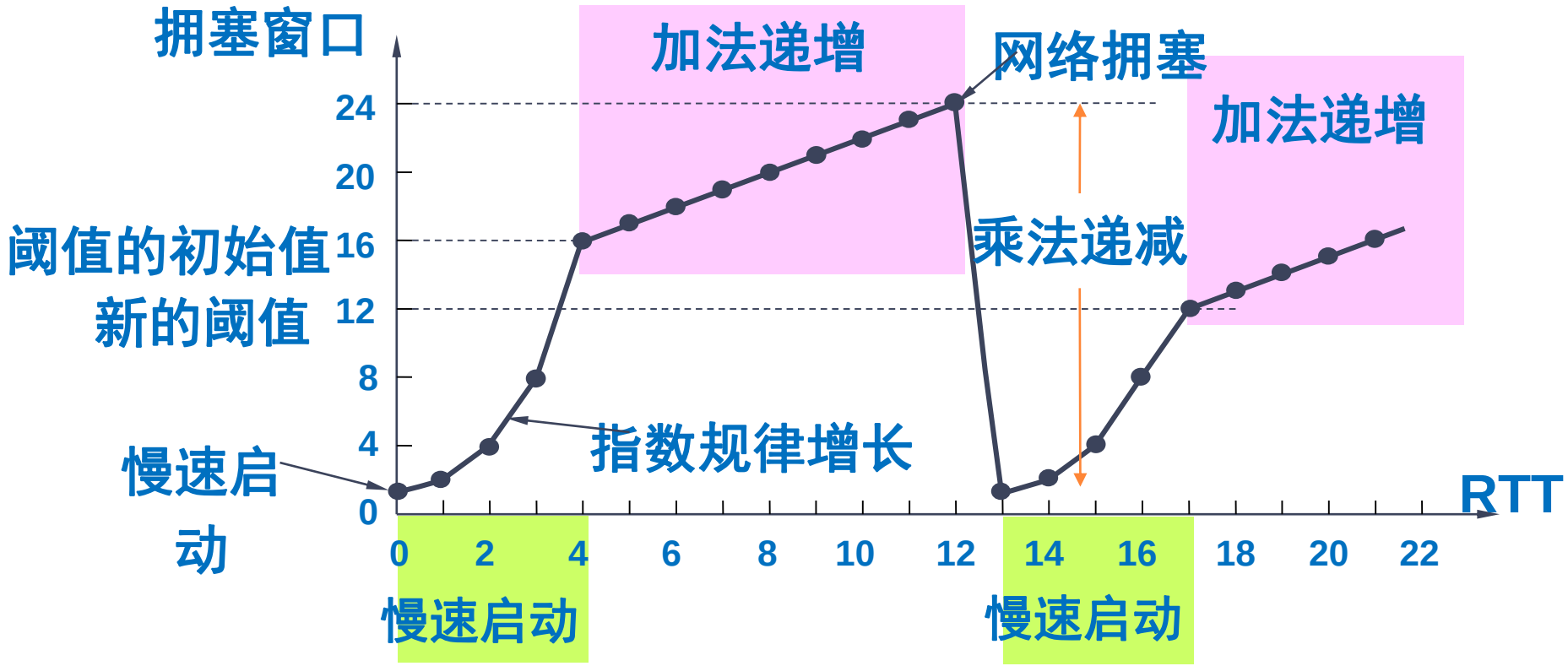
6.5.10 TCP拥塞控制

- 快重传：



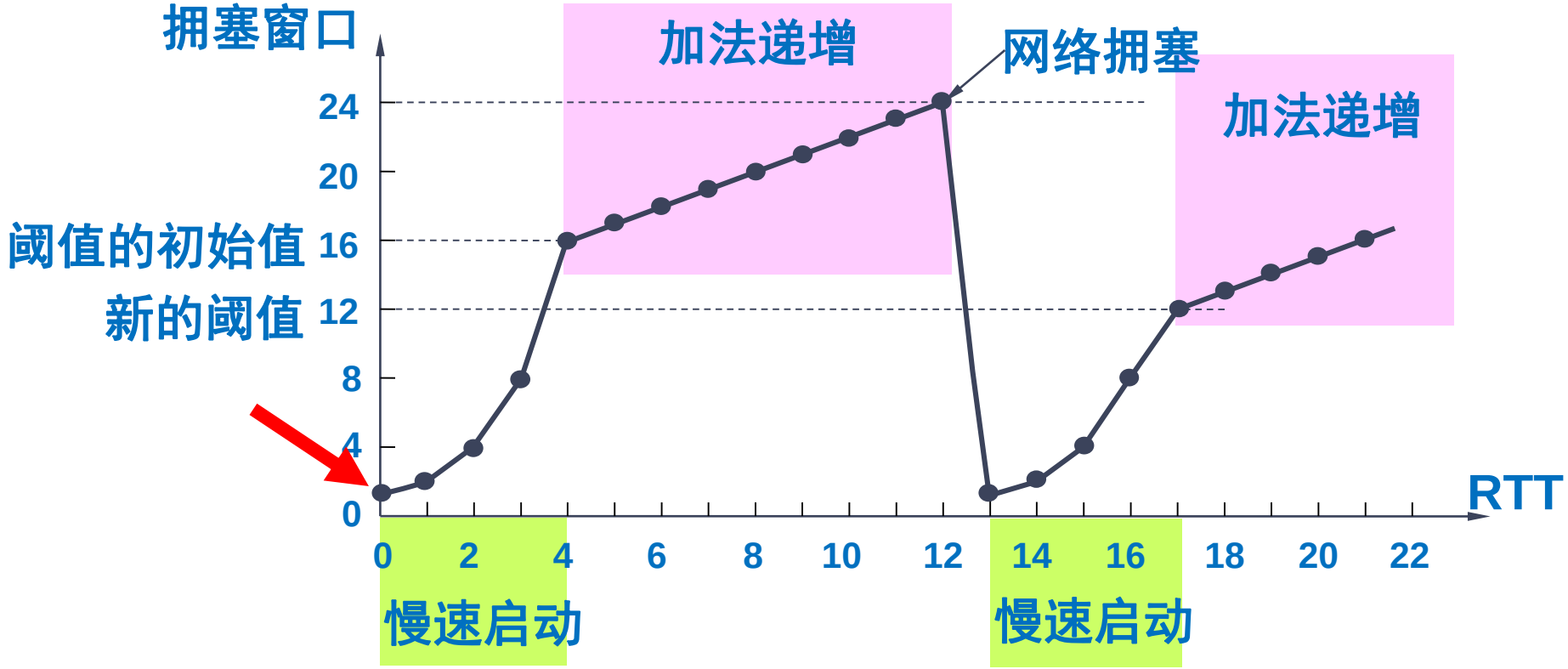
6.5.10 TCP拥塞控制

- 慢速启动-拥塞避免的过程



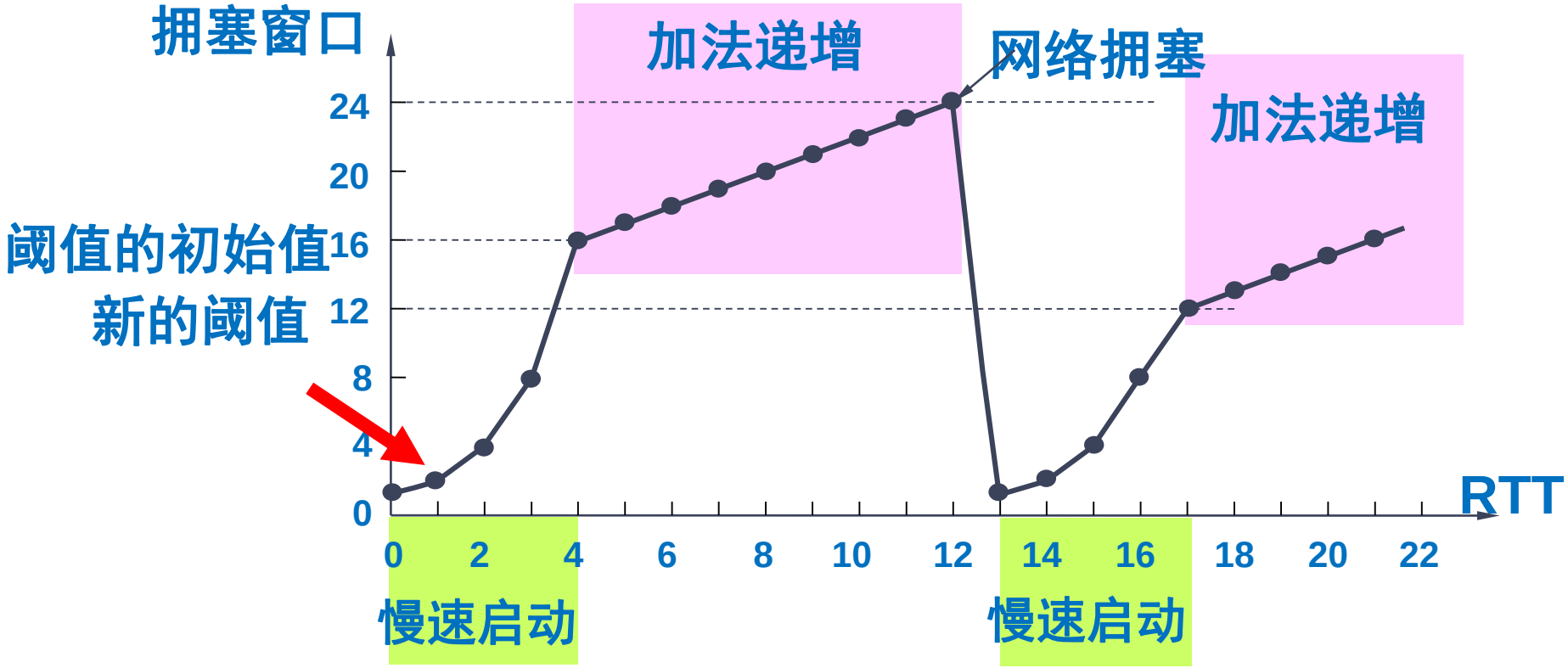
6.5.10 TCP拥塞控制

在执行慢启动算法时，拥塞窗口的初始值为 1，发送第一个报文段 M_0。



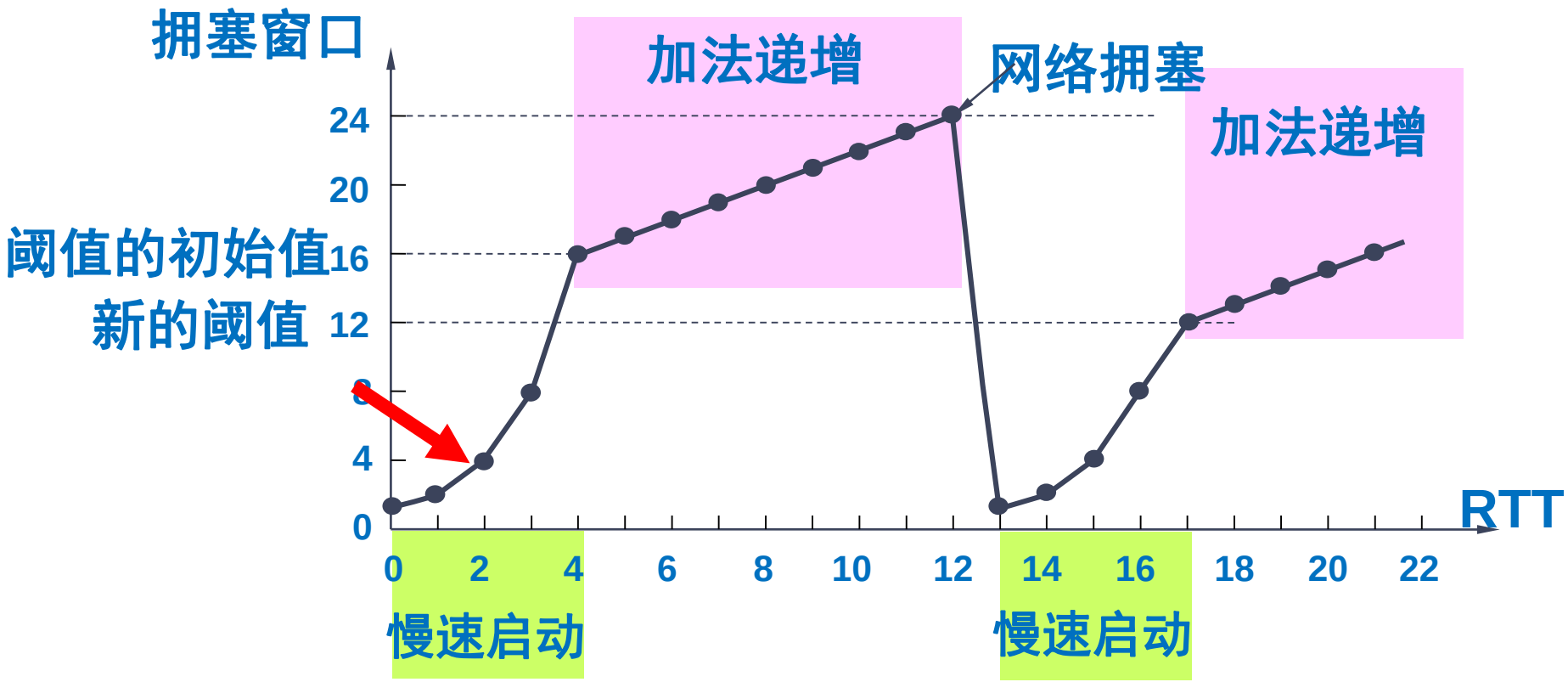
6.5.10 TCP拥塞控制

发送端每收到一个确认 ， 就把拥塞窗口值加 1。于是发送端可以接着发送 M_1 和 M_2 两个报文段。



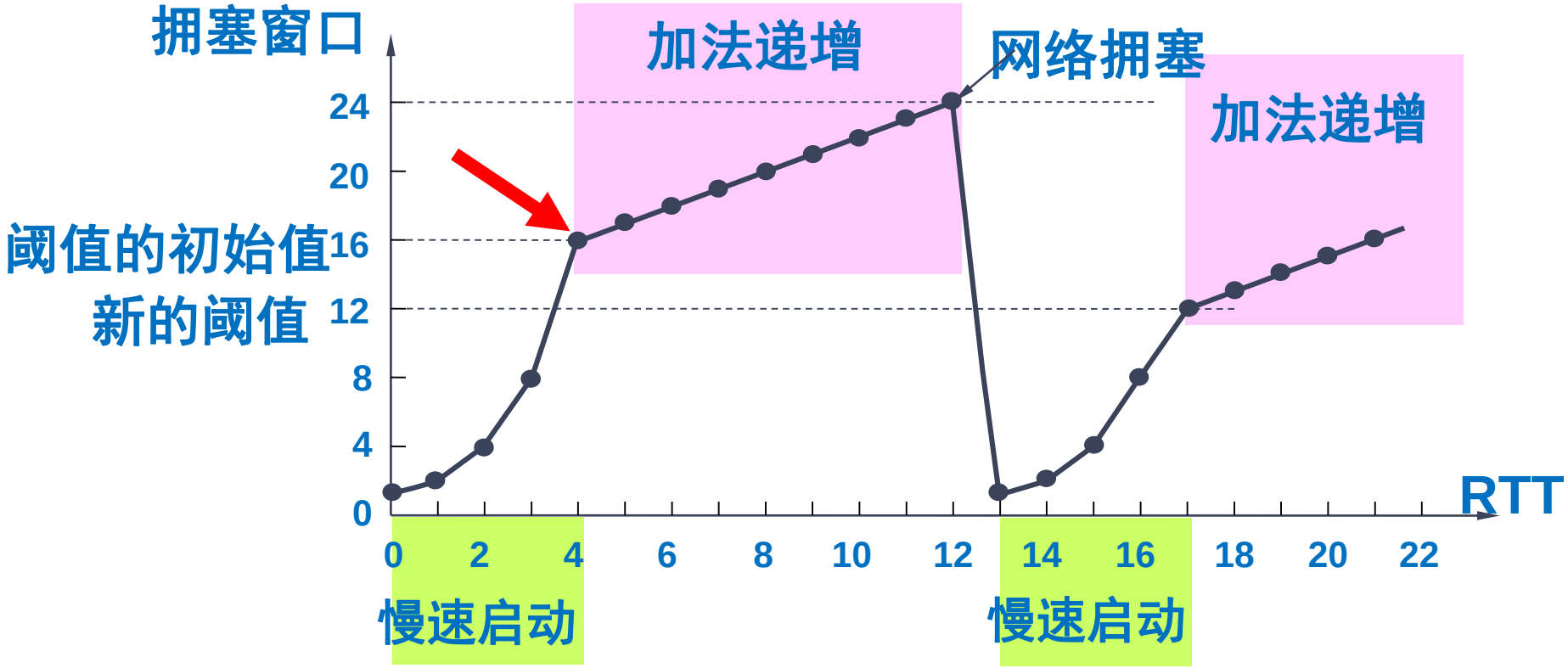
6.5.10 TCP拥塞控制

接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的窗口加 1。现在窗口从 2 增大到 4，并可接着发送后面的 4 个报文段。



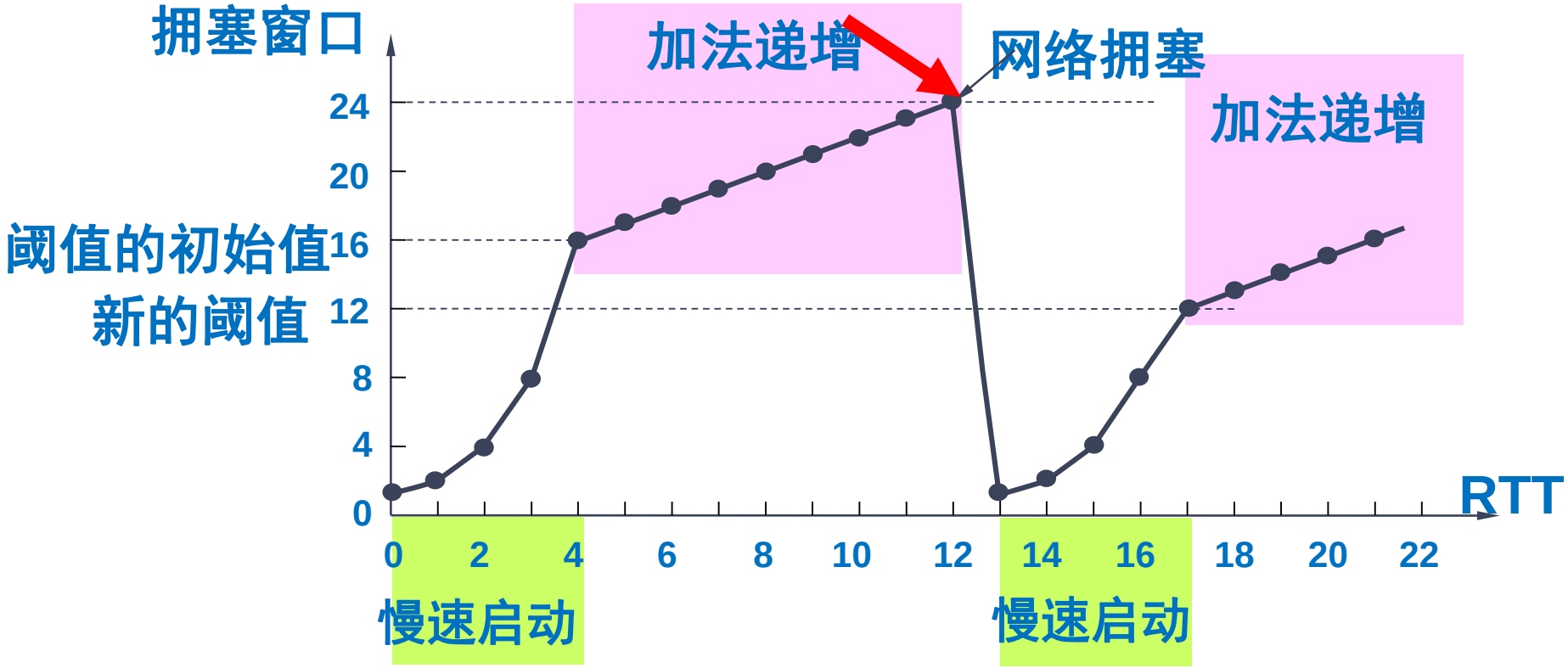
6.5.10 TCP拥塞控制

当拥塞窗口增长到慢开始阈值时（即当 16 时），就改为执行加法递增算法，拥塞窗口按线性规律增长。



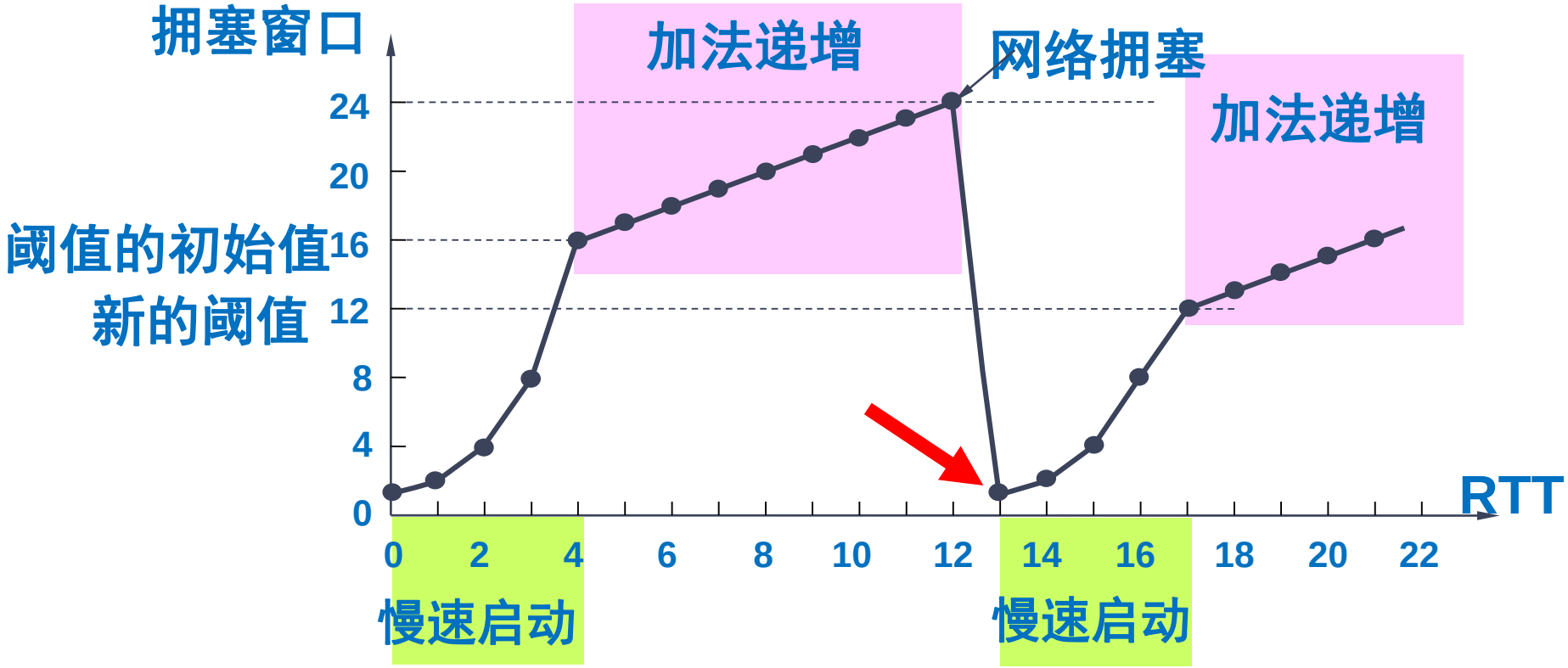
6.5.10 TCP拥塞控制

假定拥塞窗口的数值增长到 24 时，网络出现超时，表明网络出现拥塞了，需要调整拥塞窗口大小和阈值大小。



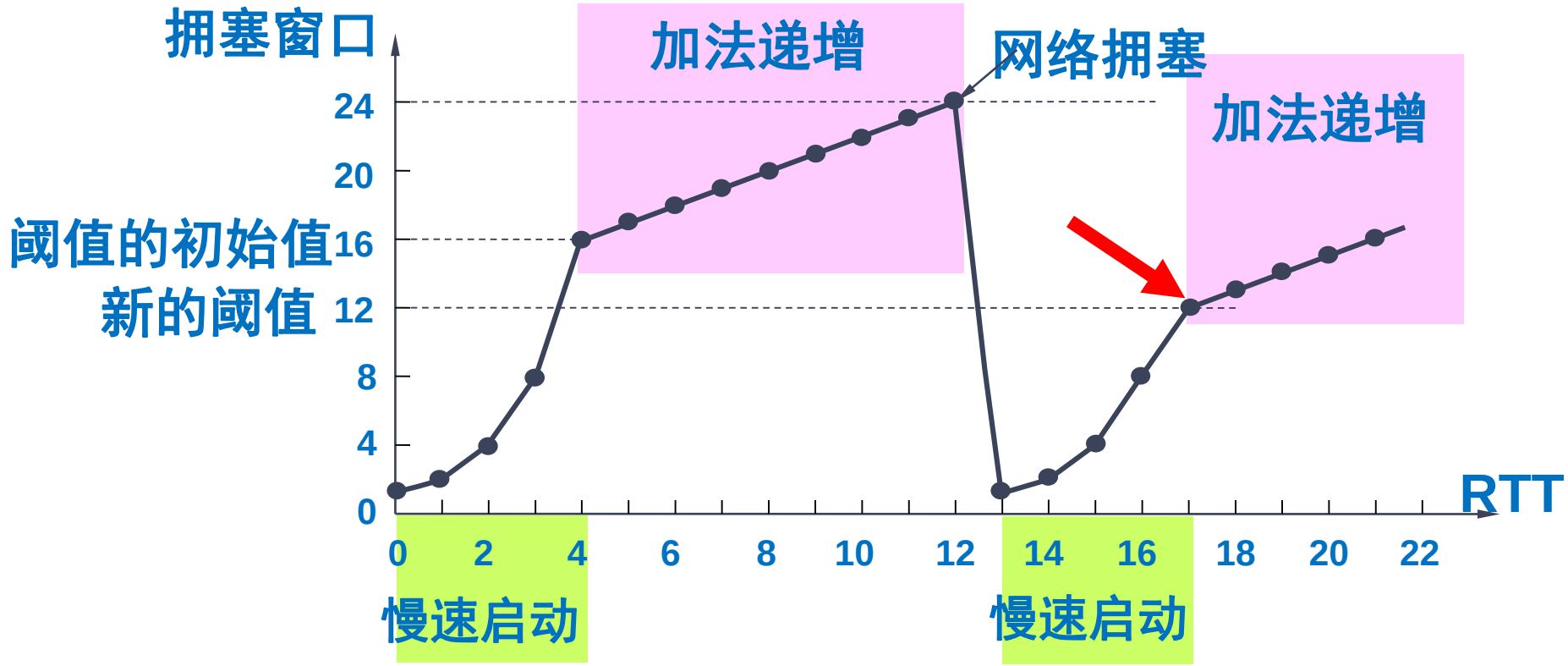
6.5.10 TCP拥塞控制

更新后的阈值值变为 12（即当时发送窗口数值 24 的一半），拥塞窗口再重新设置为 1，并执行慢启动算法。



6.5.10 TCP拥塞控制

当 窗口值= 12 时改为执行加法递增算法，拥塞窗口按线性规律增长

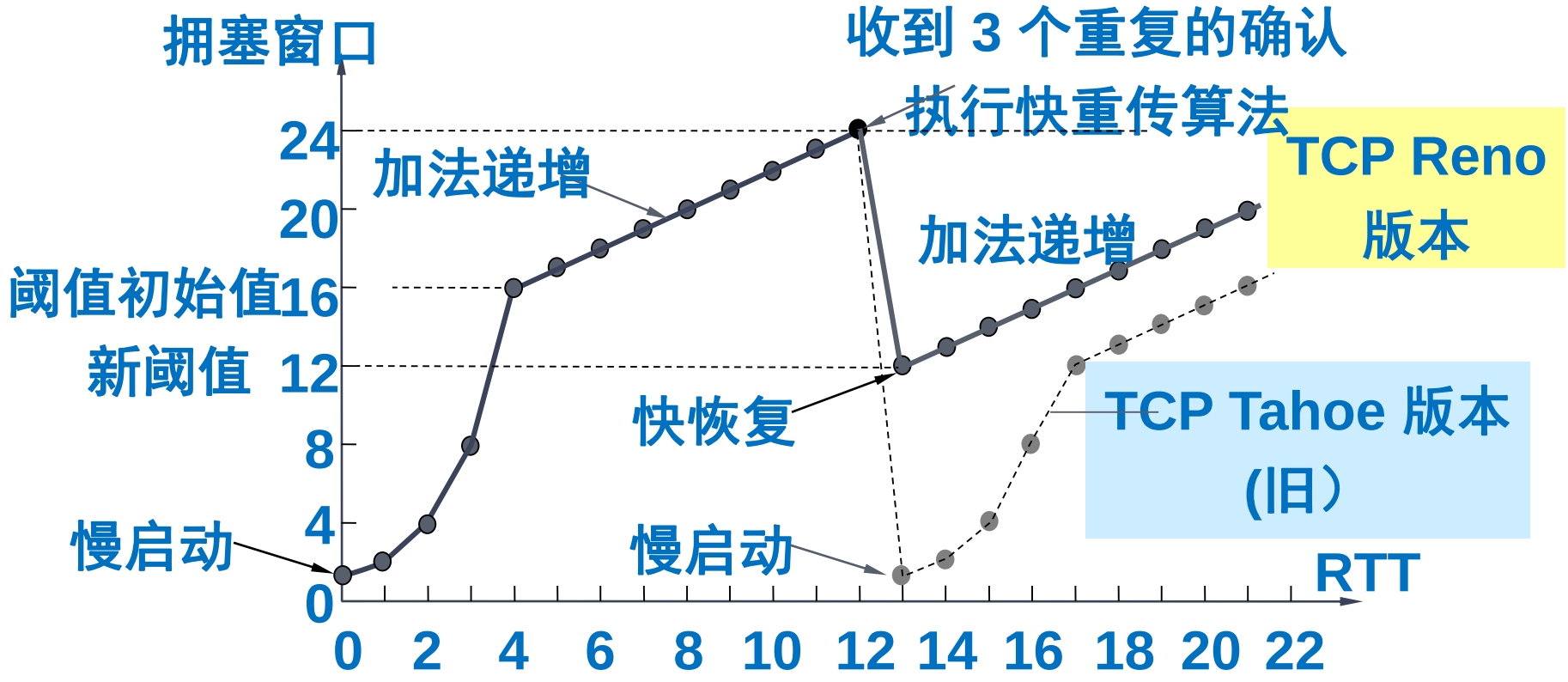


6.5.10 TCP拥塞控制

- 快速恢复：当网络的拥塞控制处于快速重传情况下时，网络可能并未发生拥塞，而只是部分数据包丢失。
- 4.3BSD TCP Reno对4.2BSD TCP Tahoe版本做了改进，当发送端收到三个重复确认时，认为网络可能并未发生严重拥塞。因此只是将阈值将为当前窗口的一半，采用线性增加的方式增大拥塞窗口值，而并不是将拥塞窗口值降低到初始值并开始慢启动。
- 只有当系统第一次运行，或检测到超时才使用慢启动。收到三个重复确认时采用快速恢复。
- 如果快速恢复不起作用，仍然会发生超时，从而进入慢启动。

6.5.10 TCP拥塞控制

- 快速恢复：



6.5.10 TCP拥塞控制

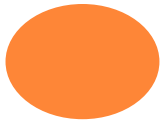
- 示例：慢速启动阈值=8，RTT为6的时候检测到三个重复确认，从而使用快速重传和快速恢复. 第10个RTT检测到超时。到请给出各RTT的窗口值大小和阈值大小，并指明各阶段时间区间

6.6 性能问题

- 性能问题非常重要
- 给出一些来自于实践的经验规则和实例
- 之所以放在传输层来讲性能，是因为应用程序获得的性能依赖于传输层、网络层和链路层的结合
- 考察网络性能问题的6个方面：
  - ✓ 性能问题
  - ✓ 网络性能的测量
  - ✓ 快速网络的主机设计
  - ✓ 快速处理段
  - ✓ 头的压缩
  - ✓ “长肥”网络协议

6.6.1 计算机网络中的性能问题

- 临时资源过载
- 网络中存在结构性的资源不平衡---可能同时触发过载，广播风暴。
- 同时触发过载----掉电
- 缺少系统总体协调而使得性能退化
- 超时间间隔的设置不合理
- 音频和视频的抖动问题



6.6.2 网络性能测量

- 确保样值空间足够大
- 确保样值具有代表性
- 缓存可以破坏测量结果
- 确保测试期间不会发生不可知事情
- 小心使用粗粒度时钟
- 小心推断结果

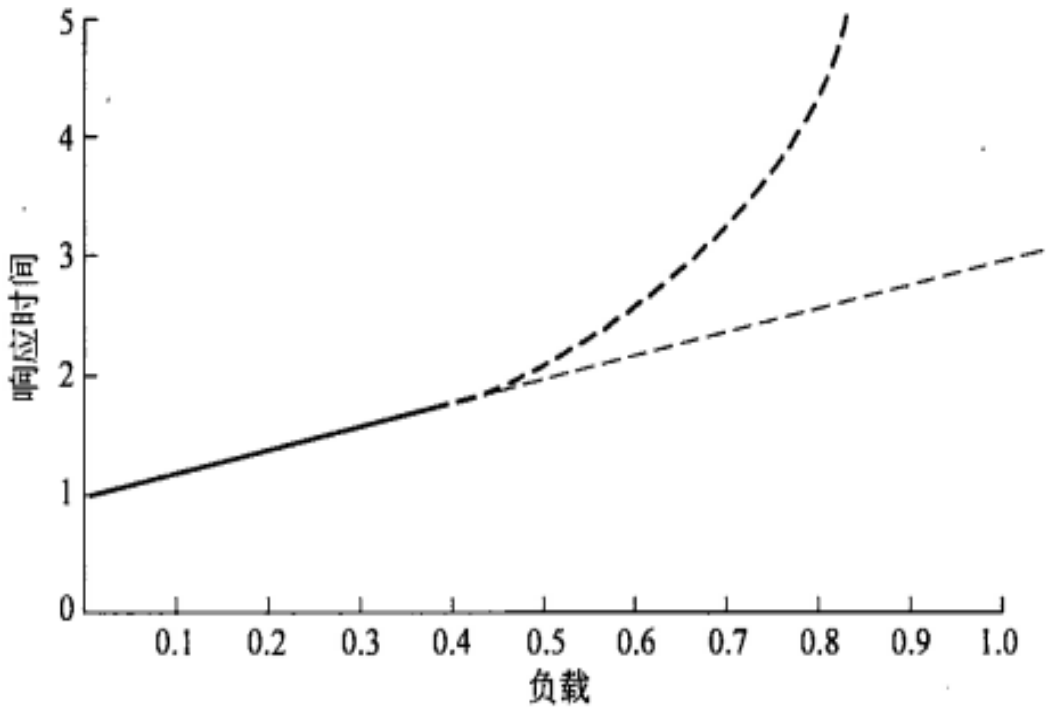
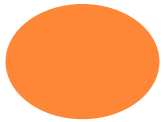


图 6-49　响应时间是负载的函数

6.6.3 针对快速网络的主机设计

- 经验表明通常的性能瓶颈不在快速网络而在于主机
  - 原因：1.网络接口卡和路由器在公车上早已能以“线速率”运行---他们处 理数据包的速度和数据包到达链路的速度一样快。2.相关的性能是指应用程序能获得的，是经过网络和传输层处理之后的吞吐量和延迟，而非链路容量。
  - 减少软件开销可以提高吞吐量和减少延迟。能耗这个因素对于移动计算机时很重要考虑的。
- 主机速度比网络速度更重要
  - 在1Gpbs上运行的最大问题是应该足够快地将比特从用户缓冲区中取出放到网络上，以及接受主机以比特到达的速度来处理它们。



6.6.3 针对快速网络的主机设计

- 减少包技术来降低开销
- 最小化数据预取---把多个层次的处理结合在一起，一次来最小化复制操作
- 最小化上下文切换---通过内部缓冲机制可以减少上下文切换
- 避免拥塞比从中恢复更好
- 避免超时---应该尽量少用计时器，使用时尽量设置成稍稍超过一点保守时间边界

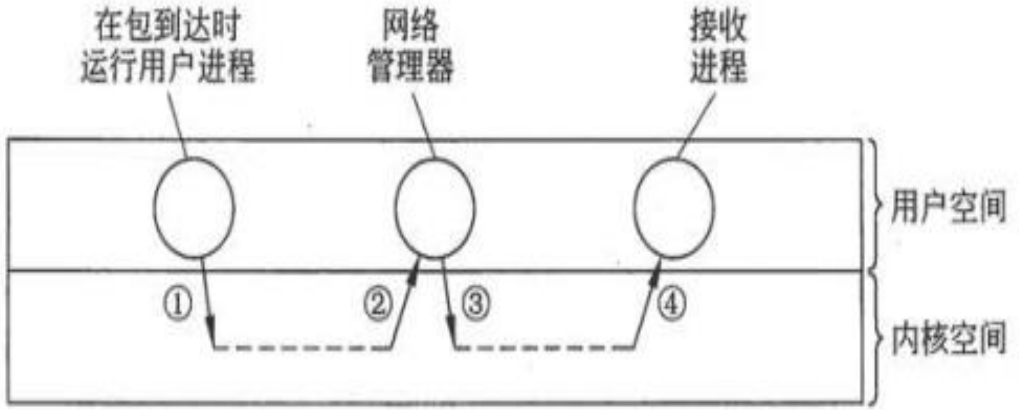


图 6-50　处理一个数据包要经历从用户空间到网络管理器的 4 次上下文切换

6.6.4 快速处理段

- 段的处理开销由每个段的处理开销和每个字节的处理开销两部分组成
- 快速处理段的关键是分离出一般情况和成功情况（单向数据传输），并对它们作特殊处理。

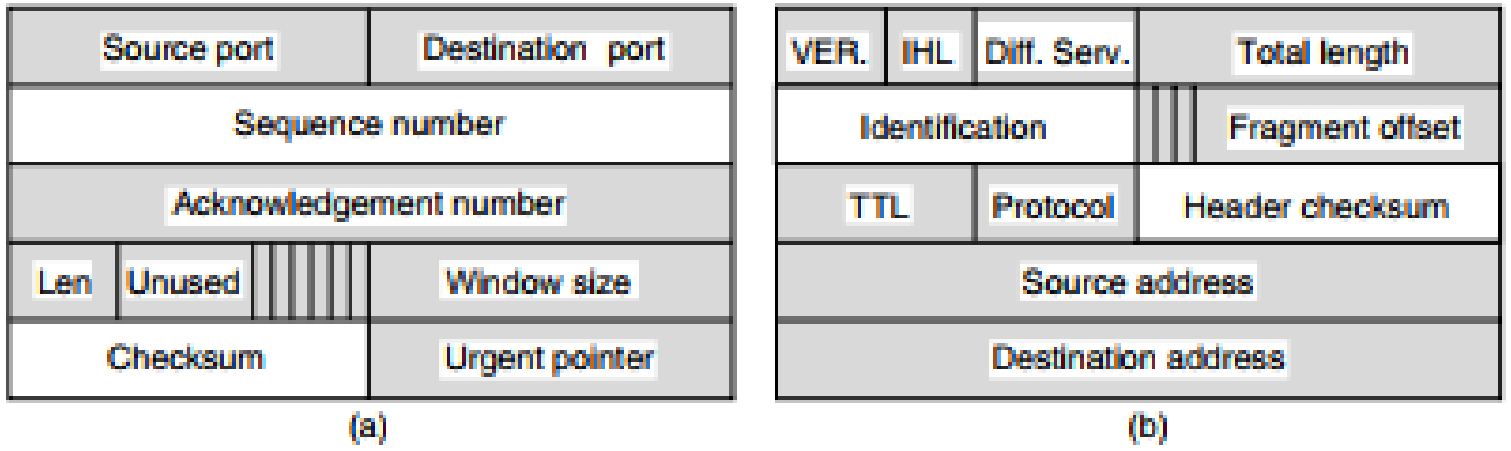


Figure 6-52. (a) TCP header. (b) IP header. In both cases, they are taken from

6.6.4 快速处理段

- 快速路径处理过程：
  - ✓ **1.找到与入境段相对应的连接记录**
  - ✓ **2.接收端检查入境段，看是否正常：连接状态是ESTABLISHED、连接的两端均不打算关闭连接、段完好无缺、段没有特殊的标志位，并且它的序号正好是接收端所期望的。如果所有条件均满足，则调用特殊快速路径TCP过程。**
  - ✓ **3.快速路径更新连接记录，并将数据复制给用户**

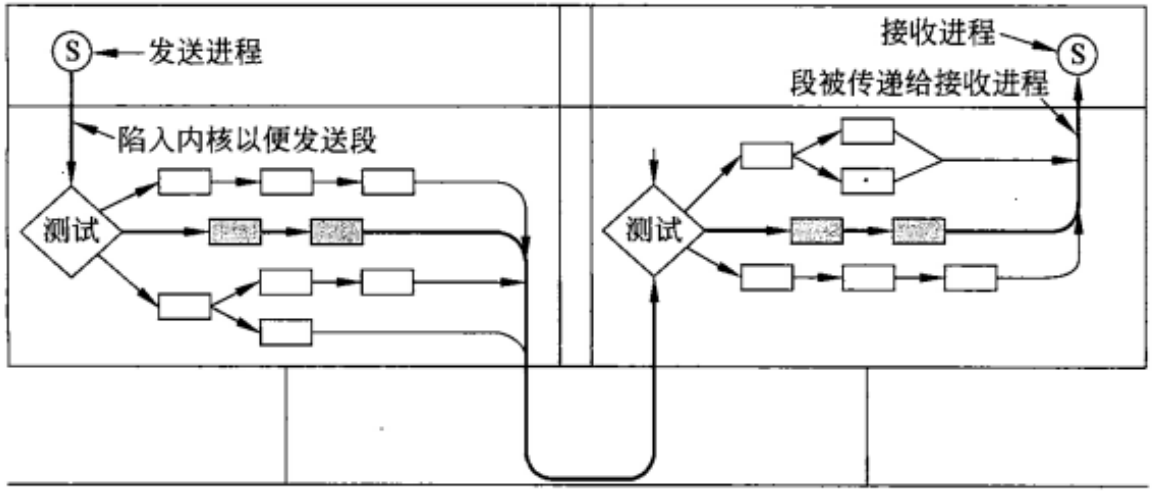


图 6-51　一条重载线路上从发送端到接收端的快速路径（阴影标示了这条路径上的处理步骤）

6.6.4 快速处理段

- 两个方面的性能有可能获得较大幅度的提高：
  - ✓ 缓冲区管理————主要问题是避免不必要的复制操作
  - ✓ 计时器管理————用链表结构将计时器事件按超时值进行排序保存

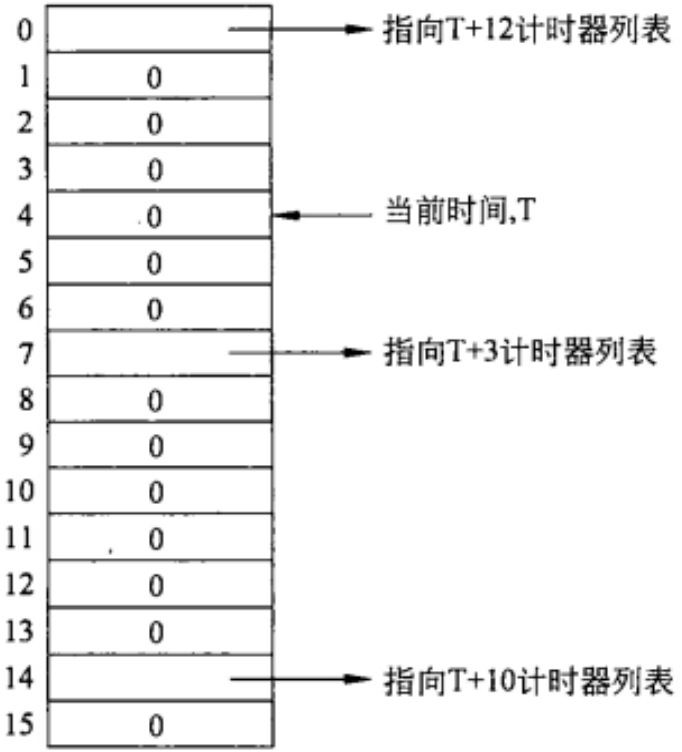
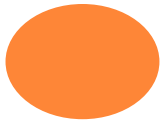


图 6-53　计时轮

6.6.5 头压缩

- 为了更好地使用贷款，协议头和有效载荷应该携带尽可能少的比特
- 有效载荷必须使用信息的压缩编码—意味着应用程序级的缓存机制应该摆在首位，比如用降低传输的Web缓存机制。
- 无线网络中的链路层，典型的头是压缩的。因为它们始终秉承着专为稀缺带宽而设计的思想。如：802.16
- 为减少软件开销而采用的流水型处理往往导致头无法压缩。
- 头压缩技术可用来降低高层协议头消耗的链路带宽，通常采用专门的设计方案。



6.6.5 头压缩

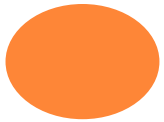
- 头压缩方案：对通过低速串行链路上的TCP/IP头进行也说---(Van,1990)

40→3个字节。

- 低带宽链路上进行压缩编码----鲁棒头压缩设计目标:能够容忍发生无线链路上的丢失。每一组被压缩的协议都有一个轮廓文件。最终传输的压缩头是携带一个指向上下文的引用，它本质上是一个连接
- 头压缩目标主要是针对减少带宽需求，同时可以有助于降低延迟。
- 头压缩技术有助于减少发送的数据，从而降低传输延迟。

6.6.7 长肥网络的协议

- 长肥网络：长距离传输数据的千兆网络，由于结合了快速网络或“肥管道”而被称为“长肥网络”
- 可能的问题：
  - 序号回绕
  - 流量控制窗口的大小必须增长得很快
  - 简单重传策略
  - 长距离千兆位线路的延迟
  - 通信速度提高，远远超越了计算速度



6.7 延迟容忍网络

- 消息交换：在偶尔连接的网络中，数据存储在某些节点，随后有工作链路时再转发这些数据，这种技术称为消息交换
- 延迟容忍网络：基于消息减缓通信方法构建的网络体系结构

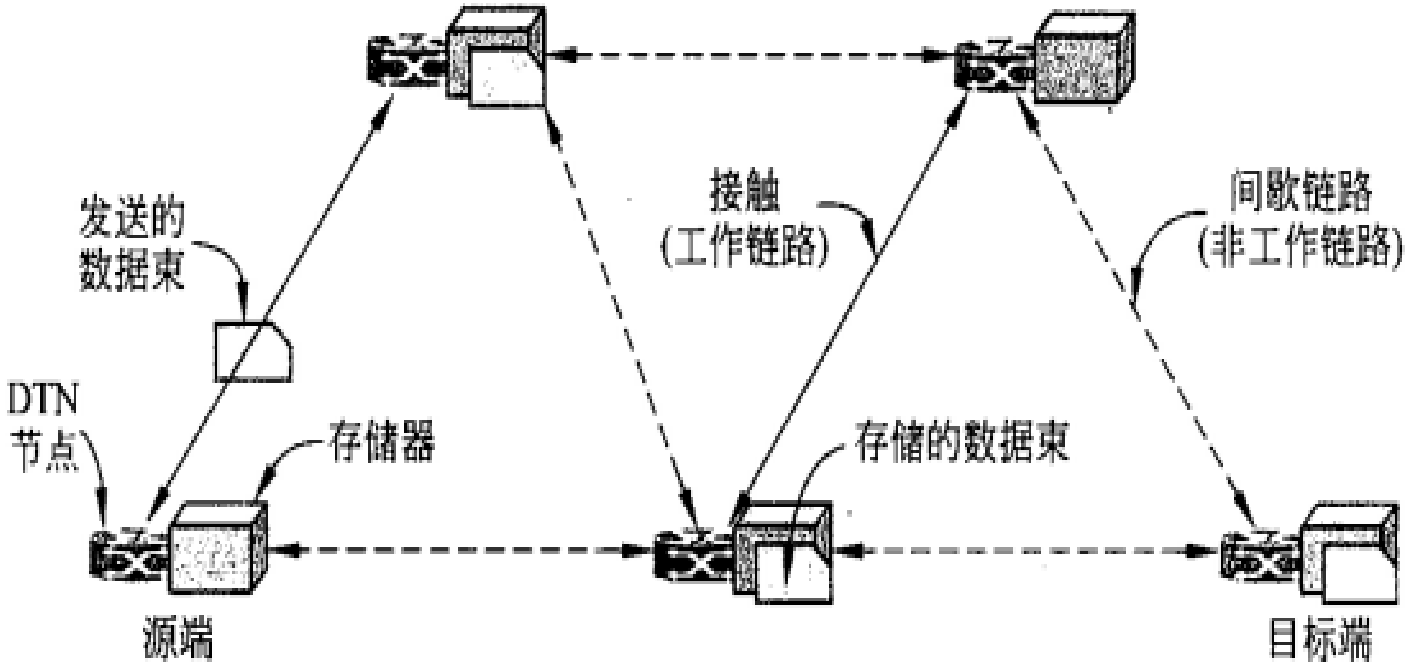


图 6-56　延迟容忍网络体系结构

第六章作业:

12，14，15，19，22，23，26，32，38，39