

第五章 传输层

刘 轶

北京航空航天大学 计算机学院

本章内容

- 5.1 传输层协议概述**
- 5.2 用户数据报协议 UDP**
- 5.3 传输控制协议 TCP 概述**
- 5.4 TCP报文段的首部格式**
- 5.5 TCP可靠传输的实现**
- 5.6 TCP的流量控制**
- 5.7 TCP的拥塞控制**
- 5.8 TCP的连接管理**

5.1 传输层协议概述

5.1 传输层协议概述

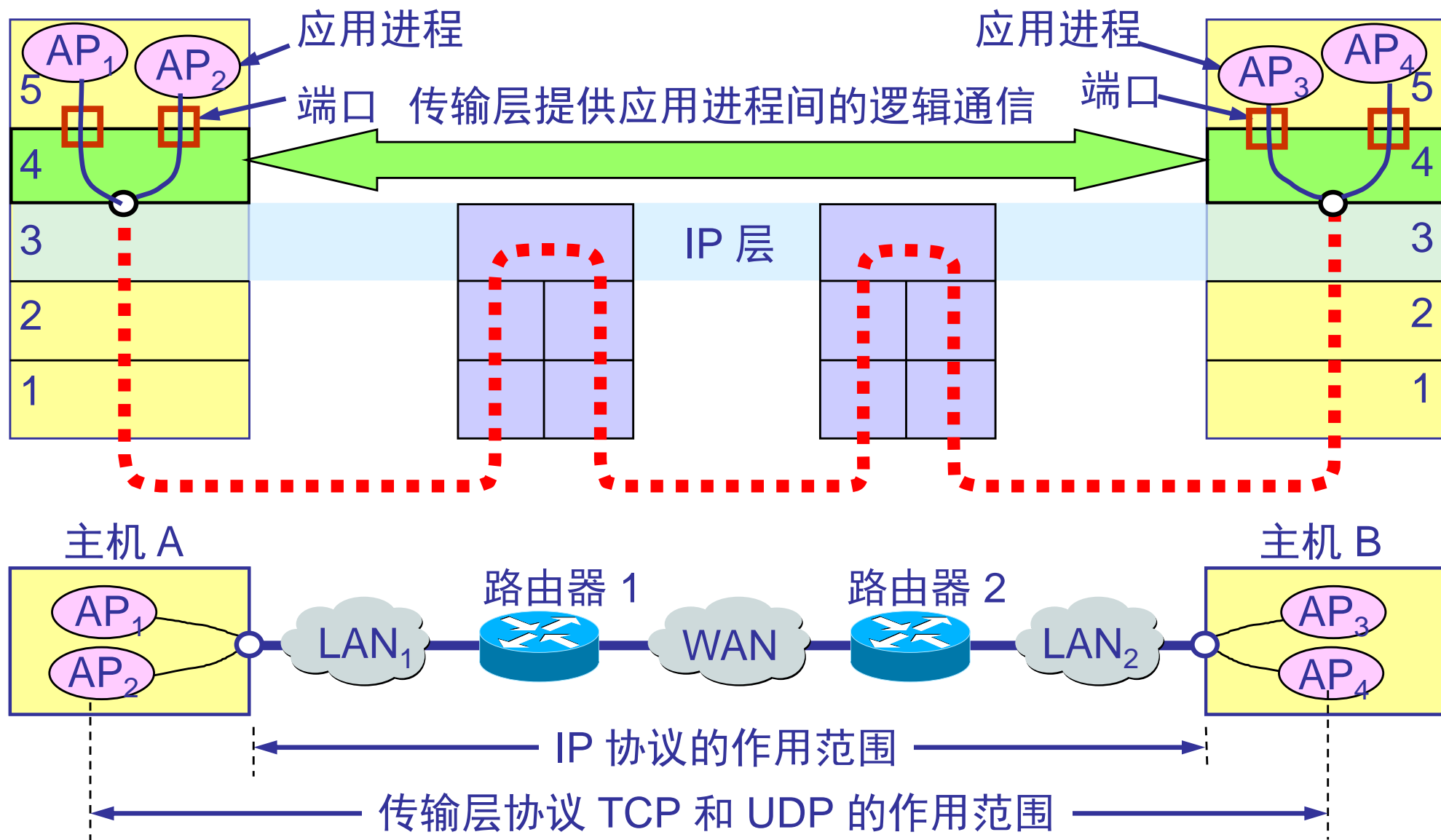
一、进程之间的通信

- 传输层(Transport layer)又称为**运输层**
- 传输层向它上面的应用层提供通信服务
 - 实现**可靠传输**：**差错控制、顺序控制、拥塞控制**等
- 传输层 vs. 网络层
 - 网络层实现**主机之间**的逻辑通信
 - 传输层实现**应用进程之间**的逻辑通信
 - 真正的端到端通信
 - 复用(multiplexing)和解分(demultiplexing)
- 传输层主要协议
 - **TCP** 协议：可靠传输协议
 - **UDP**协议：不可靠传输协议

5.1 传输层协议概述


AP: Application Process

一、进程之间的通信



5.1 传输层协议概述

2009年的一道考研题

- 在OSI参考模型中，自下而上第一个提供端到端服务的层次是：
A. 数据链路层  B. 传输层 C. 会话层 D. 应用层

5.1 传输层协议概述

二、传输层的两个主要协议

- 传输层的两个主要协议
 - 用户数据报协议**UDP**
(User Datagram Protocol)
[RFC 768]
 - 传输控制协议**TCP**
(Transmission Control Protocol)
[RFC 793]
- 传输的数据单位
 - **OSI**术语称为**TPDU**(Transport Protocol Data Unit)
 - **TCP/IP**体系称为**TCP**报文段(segment)或**UDP**用户数据报



5.1 传输层协议概述

二、传输层的两个主要协议

- **TCP**协议

- 可靠传输协议
- 提供面向连接的服务
 - 传送数据前要先建立连接，传送结束后释放连接
 - 需进行确认、流量控制、计时器、连接管理等，处理开销较大
- 基于**TCP**的典型应用协议：**HTTP**、**FTP**、...

- **UDP**协议

- 不可靠传输协议
- 传送数据时无需建立连接
- 与**TCP**相比，效率更高，但可能出现数据错、丢包、顺序错等问题
- 基于**UDP**的典型应用协议：**DNS**、**RIP**、...

5.1 传输层协议概述

三、传输层的端口(1/2)

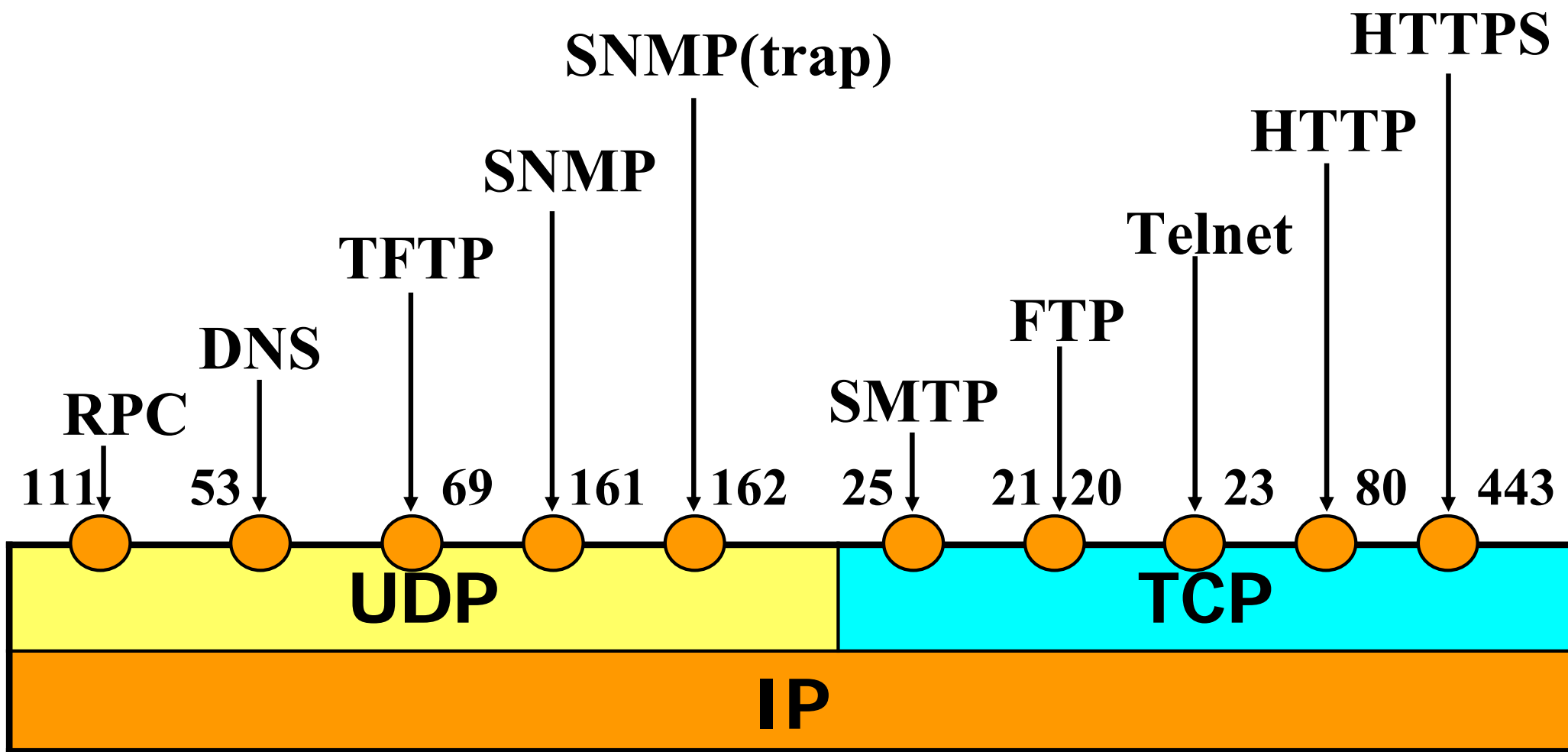
- 传输层需为多个应用进程提供服务 → 复用与分用
 - 应用层多个应用进程通过传输层发送数据 → 复用
 - 传输层收到的数据必须交付给指明的应用进程 → 分用
- 传输层必须提供区分上层应用进程的手段：端口(port)
 - 问题：为什么不用进程ID？
 - 进程ID的定义依赖于特定操作系统
 - 一个应用进程常常需要与网络中的多个主机中的进程通信 → 多条连接
- 注意：此端口为软件端口，不同于计算机中的硬件I/O端口和交换机/路由器上的物理端口

5.1 传输层协议概述

三、传输层的端口(2/2)

- **TCP/IP协议使用16位整数作为端口号**
 - 源端口号、目的端口号
- **端口号分类**
 - ① **熟知(well-known)端口号或系统端口号：数值一般为 0~1023**
 - 例如：HTTP服务使用80，FTP服务使用21，...
 - ② **登记端口号：数值为1024~49151**
 - 供没有熟知端口号的应用程序使用
 - 须在 IANA 登记，以防重复
 - ③ **客户端端口号或短暂端口号：数值为49152~65535**
 - 客户进程临时使用

RFC1700: Assigned Numbers



常用的熟知端口号

5.2 用户数据报协议 UDP

5.2 用户数据报协议UDP

一、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**
 - 端口和差错检测
- **UDP 为不可靠传输协议，但具有自身特点，与TCP分别面对不同的应用**
- **UDP协议的特点**
 - (1) 无连接，即发送数据之前不需要建立连接
 - (2) 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
 - (3) 是面向报文的。没有拥塞控制，很适合多媒体通信的要求
 - (4) 支持一对一、一对多、多对一和多对多的交互通信
 - (5) 首部开销小，只有 8 个字节

二、UDP首部格式

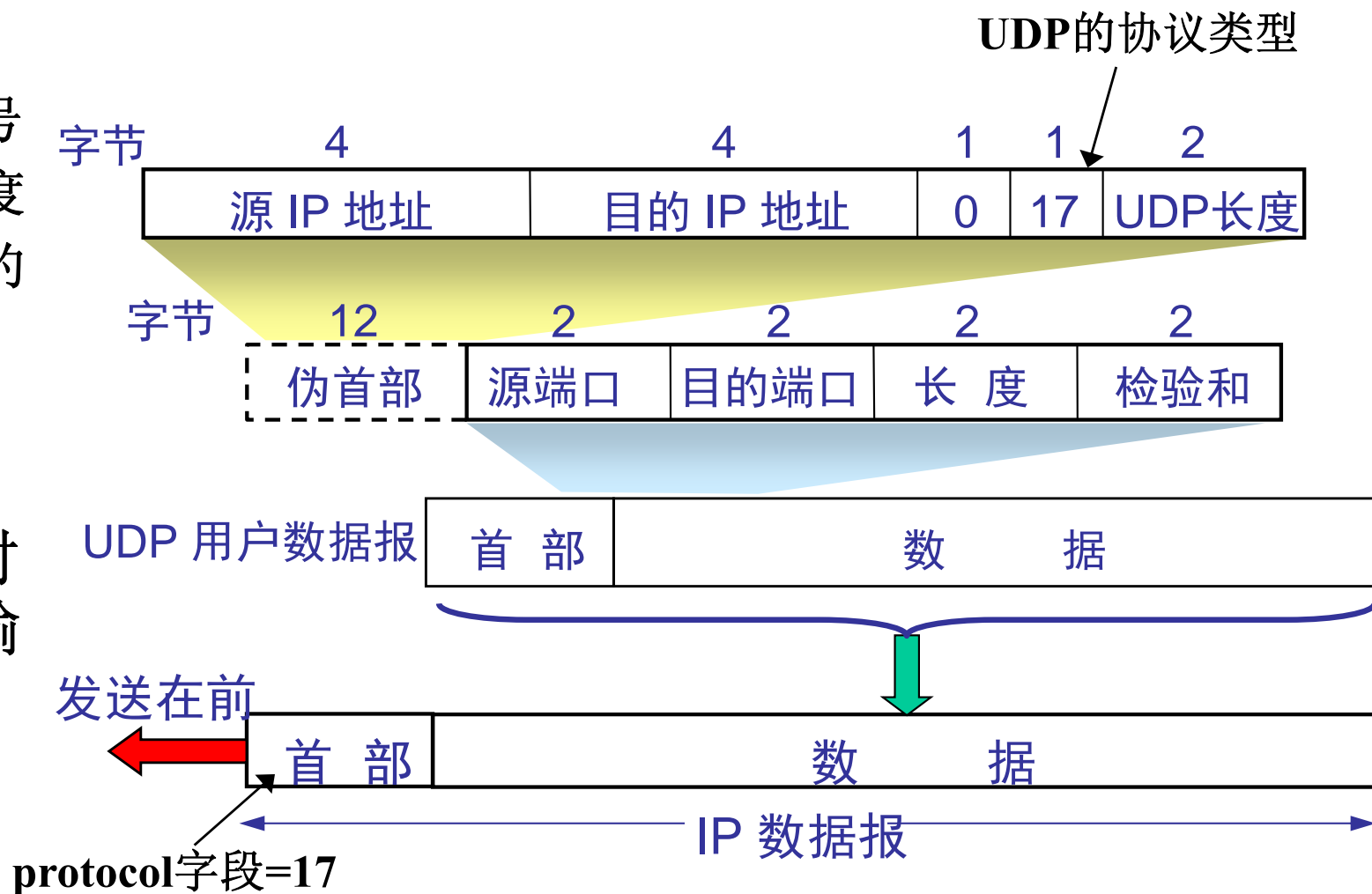
- **UDP数据报**包括2个字段：首部和数据字段
- 首部共4个字段，8字节
- 首部各字段

- (1) 源端口：源端口号
- (2) 目的端口：目的端口号
- (3) 长度：UDP数据报长度
- (4) 校验和：UDP数据报的校验和

- 伪首部

(pseudoheader):

仅在计算校验和时使用，不实际传输



注意**整数**在网络传输过程中的表示:

- 两种字节序
 - **Little endian**: 低位字节在低地址(低字节在前)
 - **Big endian**: 高位字节在低地址(高字节在前)
- 网络中通常用**big endian**
 - 为使程序独立于硬件平台, 编程时注意使用转换字节序的宏: **htonl(), htons(), ntohl(), ntohs()**

UDP报文示例: DNS请求

	Protocol number																
00000000:	00	d0	d0	72	a4	b4	00	16	ea	c3	8c	6c	08	00	45	00	.行rこ..智窘..E.
00000010:	00	3e	00	f4	00	00	80	11	14	71	c0	a8	01	0a	db	8d	.>.?..€...q括..躡
00000020:	88	0a	e1	01	00	35	00	2a	1b	b6	6e	70	01	00	00	01	? ?..5.*..■ np....
00000030:	00	00	00	00	00	00	04	6d	61	69	6c	04	62	75	61	61mail.buaa
00000040:	03	65	64	75	02	63	6e	00	00	01	00	01					.edu.cn.....

源端口号 目的端口号

校验和的计算：与IP包头校验和计算类似

- **IP**只计算包头校验和，**UDP**计算整个数据报的校验和
- 计算校验和时要添加伪头部，奇数字节需填充**0**字节

12 字节 伪首部	153.19.8.104			
	171.3.14.11			
	全 0	17	15	
8 字节 UDP 首部	1087		13	
	15		全 0	
7 字节 数据	数据	数据	数据	数据
	数据	数据	数据	全 0

填充

10011001 00010011 → 153.19

00001000 01101000 → 8.104

10101011 00000011 → 171.3

00001110 00001011 → 14.11

00000000 00010001 → 0 和 17

00000000 00001111 → 15

00000100 00111111 → 1087

00000000 00001101 → 13

00000000 00001111 → 15

00000000 00000000 → 0（校验和）

01010100 01000101 → 数据

01010011 01010100 → 数据

01001001 01001110 → 数据

01000111 00000000 → 数据和 0（填充）

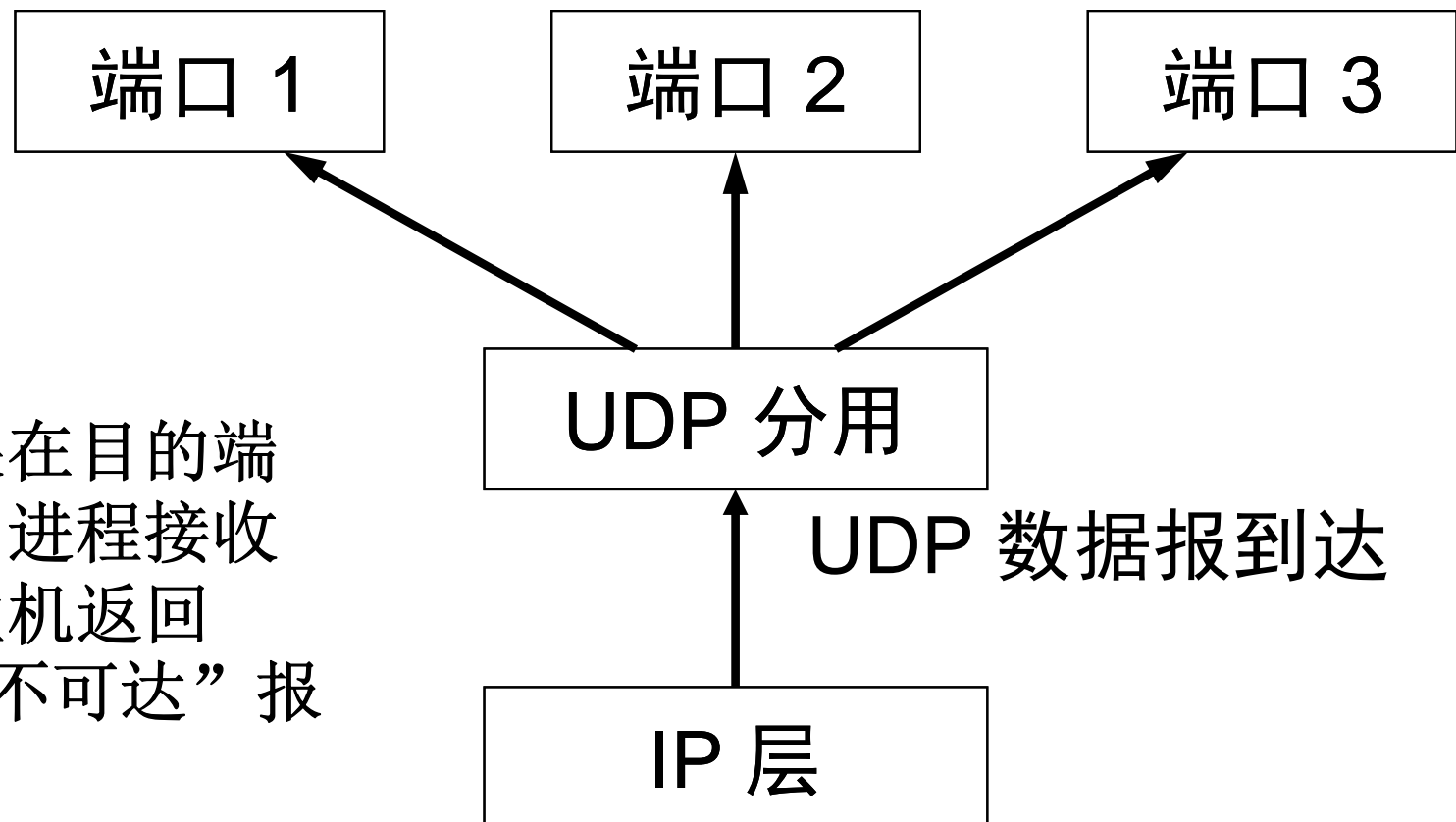
按二进制反码运算求和 10010110 11101101 → 求和得出的结果

将得出的结果求反码 01101001 00010010 → 校验和

5.2 用户数据报协议 UDP

二、UDP首部格式

- 在接收方，如果在目的端口号上没有应用进程接收数据，则向源主机返回 ICMP 的“目的不可达”报文
 - 安全问题？可能被扫描软件利用



5.3 传输控制协议 TCP 概述

5.3 传输控制协议 TCP 概述

一、TCP的主要特点

(1) TCP 是**面向连接**的传输层协议

- 传输数据前必须先建立连接，数据传输完毕后要释放连接

(2) 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)

(3) TCP 提供**可靠**交付的服务

- 无差错、不丢失、不重复、按序到达

(4) TCP 提供**全双工**通信

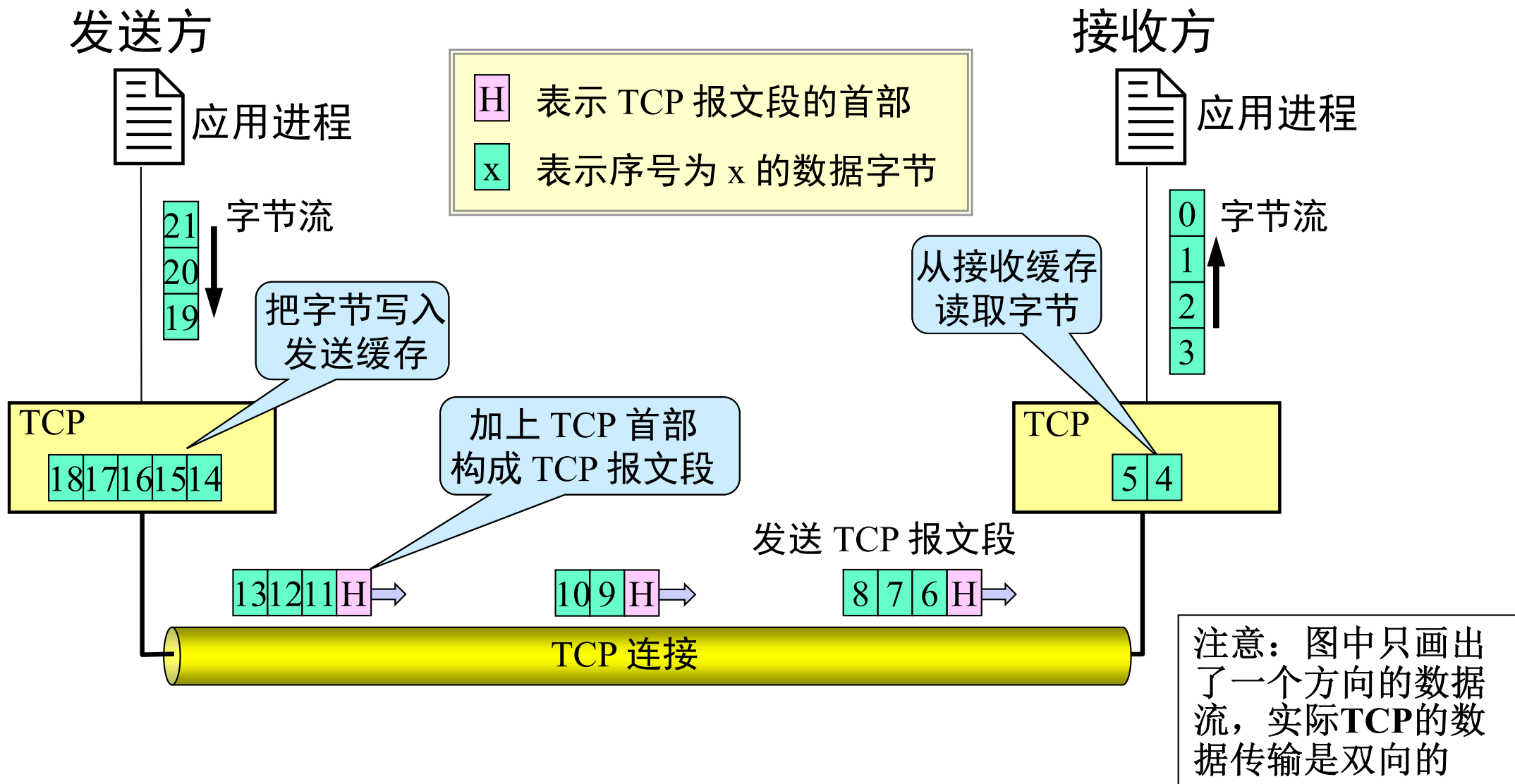
- 在一个连接上，通信双方可同时向对方传输数据

(5) 面向**字节流**

- 认为在TCP连接上传输的是字节流
- 应用程序以数据块为单位与TCP交互，但TCP将其视为无结构的字节流
- 结果：发送方应用进程发出的数据块与接收方应用进程收到的数据块可能没有一一对应关系，但数据保证一致

5.3 传输控制协议 TCP 概述

面向流的概念



5.3 传输控制协议TCP 概述

一、TCP的主要特点

- 要注意的几点:

- 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根据对方给出的窗口值和当前网络拥塞的程度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

5.3 传输控制协议 TCP 概述

二、TCP 的连接

- TCP把连接作为最基本的抽象
- 每一条TCP连接有两个端点
- TCP连接的端点不是主机，不是主机的IP地址，不是应用进程，也不是传输层的协议端口，TCP 连接的端点叫做套接字(socket)或插口
- 端口号拼接到(contatenated with) IP 地址即构成了套接字

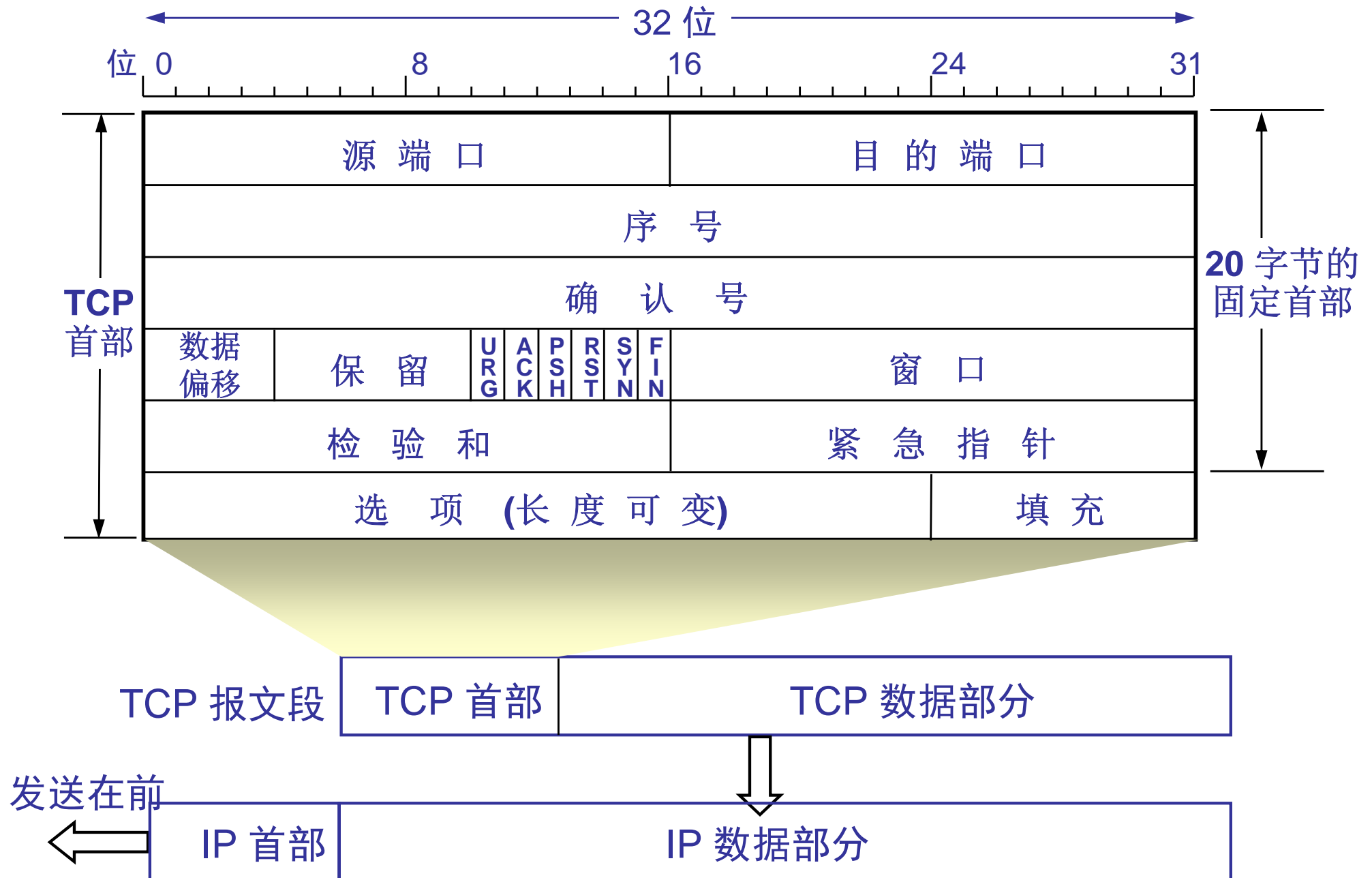
套接字 $\text{socket} ::= (\text{IP地址: 端口号})$

- 每一条 TCP 连接唯一地被通信两端的两个端点(即两个套接字)所确定。即：

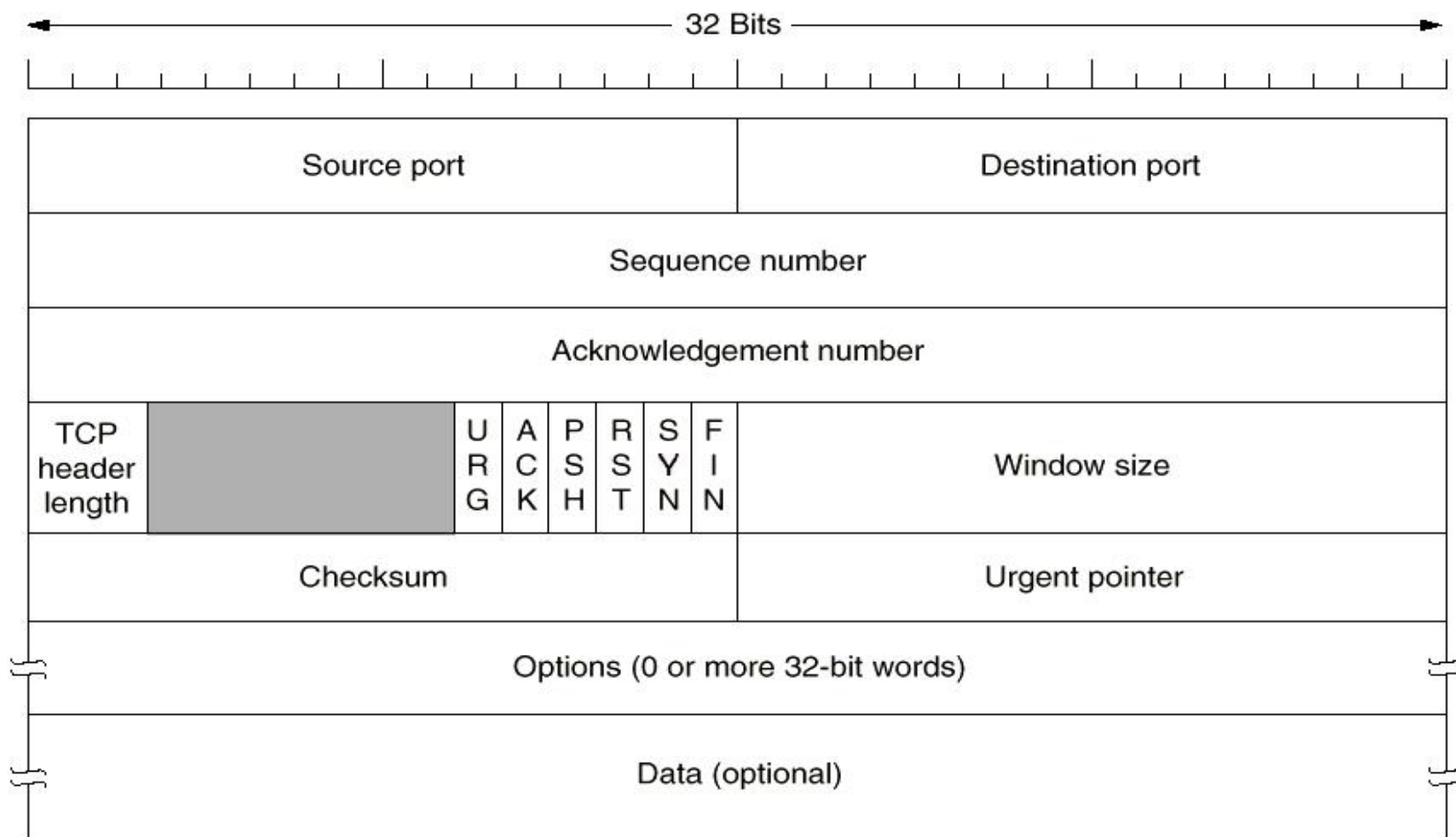
$\text{TCP 连接} ::= \{\text{socket1}, \text{socket2}\} = \{(\text{IP1: port1}), (\text{IP2: port2})\}$

5.4 TCP 报文段的首部格式

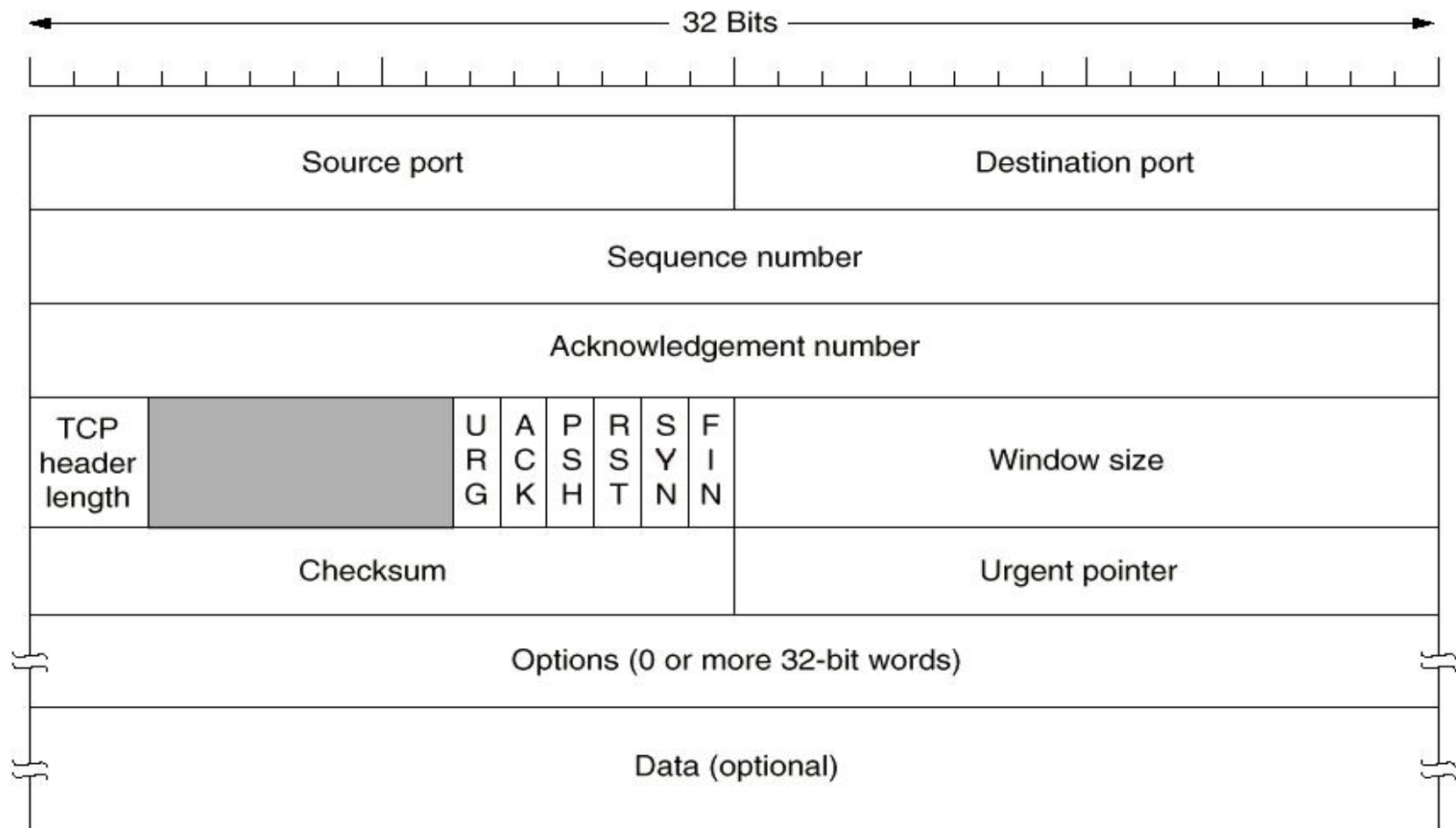
TCP报文段的首部格式



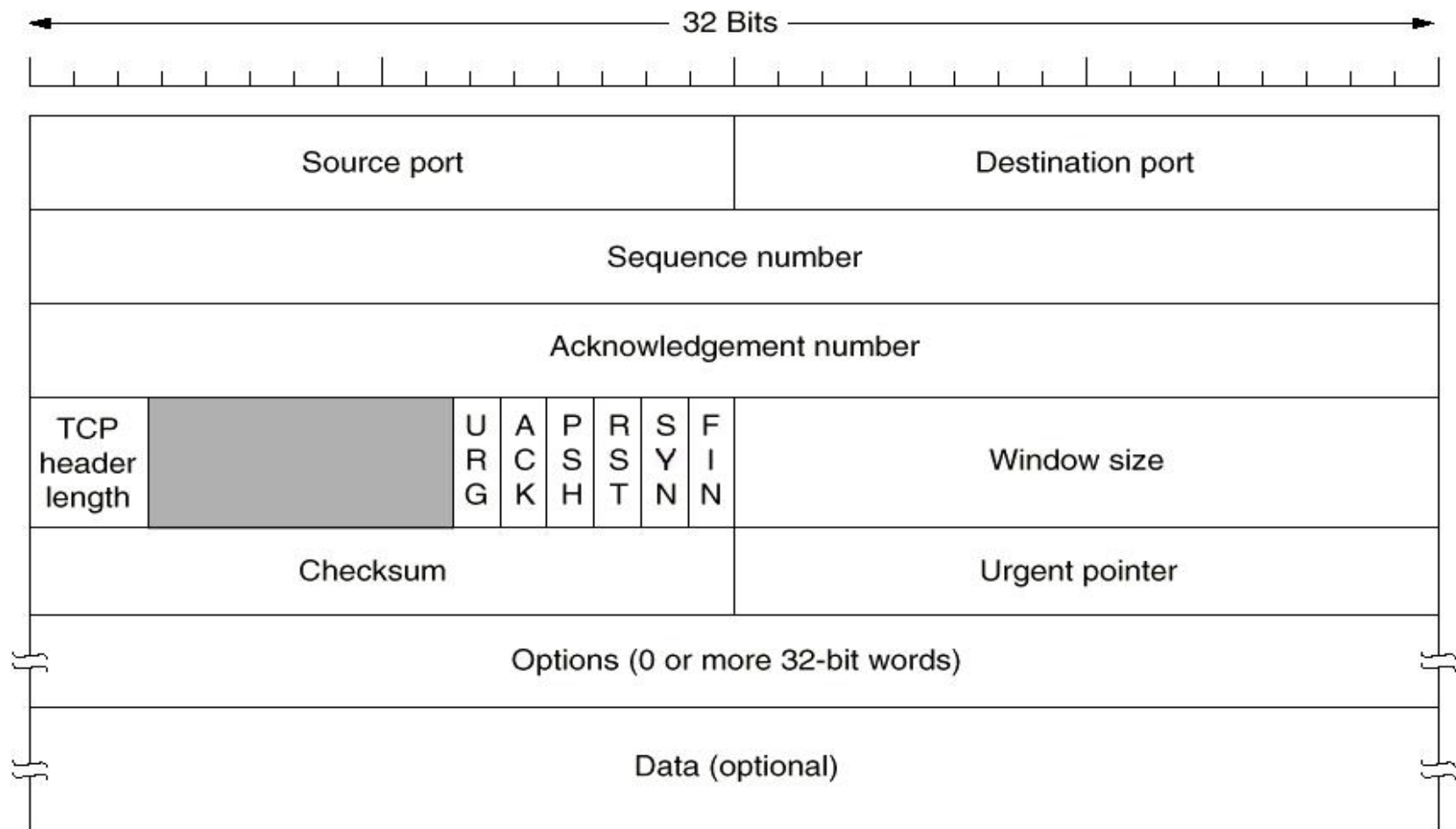
- 源端口和目的端口字段：各2字节
- 序号字段：4字节
 - 在TCP连接中传送的数据流中的每一个字节都有序号
 - 序号字段指本报文段所发送的数据的第一个字节的序号，以字节为单位
- 确认号字段：4字节，期望收到对方的下一个报文段的数据的第一个字节的序号
 - 例：收到对方的报文段中序号为501，数据长度200字节，则返回报文段确认号=701
 - TCP连接是全双工，通信双方可互相发送数据，因此应答与数据一同发送给对方



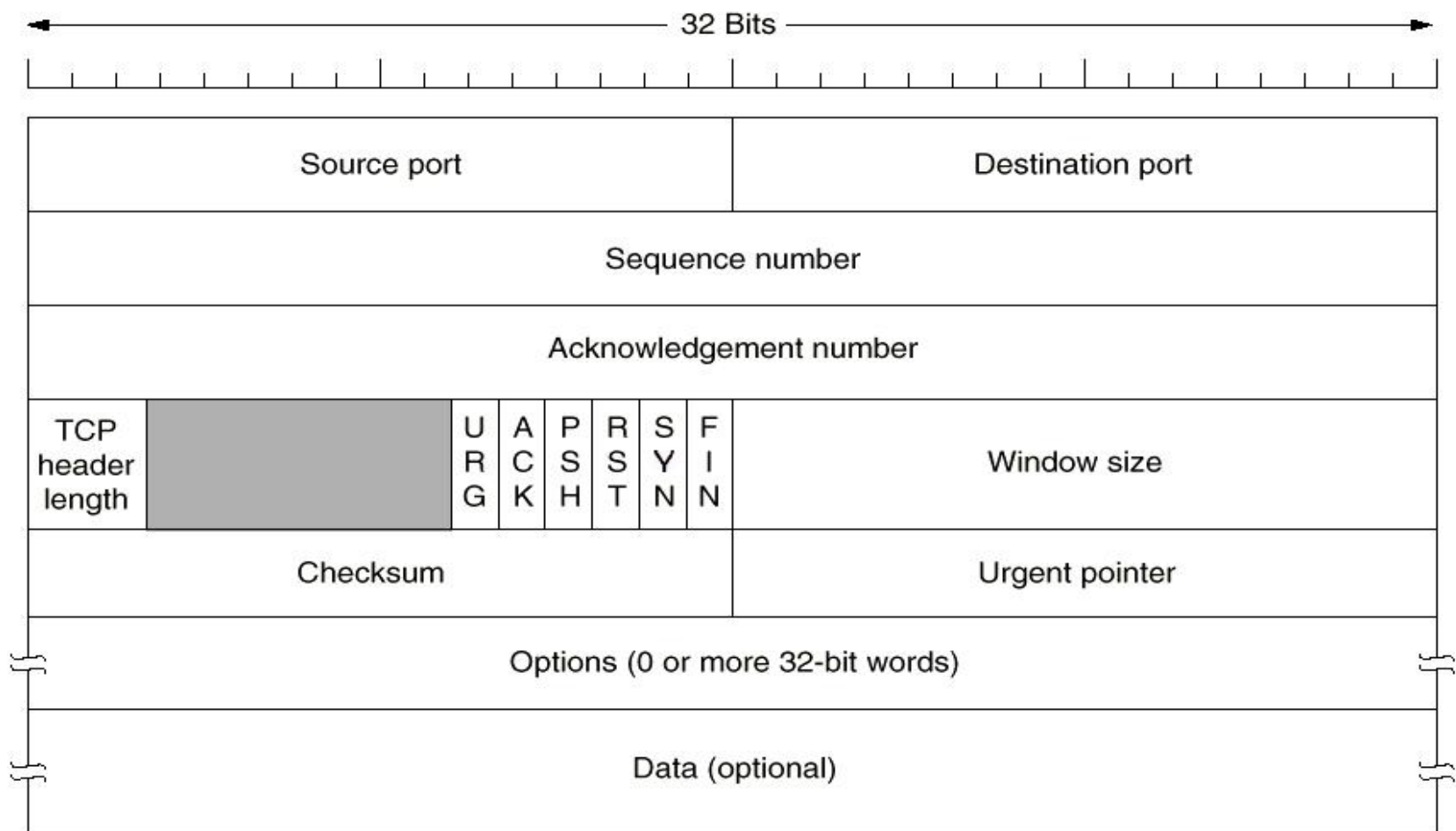
- 数据偏移(首部长度的): **4bit**, TCP报文段的数据起始位置的偏移, 也就是首部的长度, 单位是32位字(4字节)
- 保留字段: **6bit**, 保留
- 紧急URG: **1bit**, 为1时, 紧急指针字段有效, 表明有紧急数据, 应尽快传送
- 确认ACK: **1bit**, 为1时, 确认号字段有效; 为0时, 确认号无效
- 推送PSH: **1bit**, 为1时, 接收方将尽快向应用进程交付此报文段, 而不是等到整个缓存填满



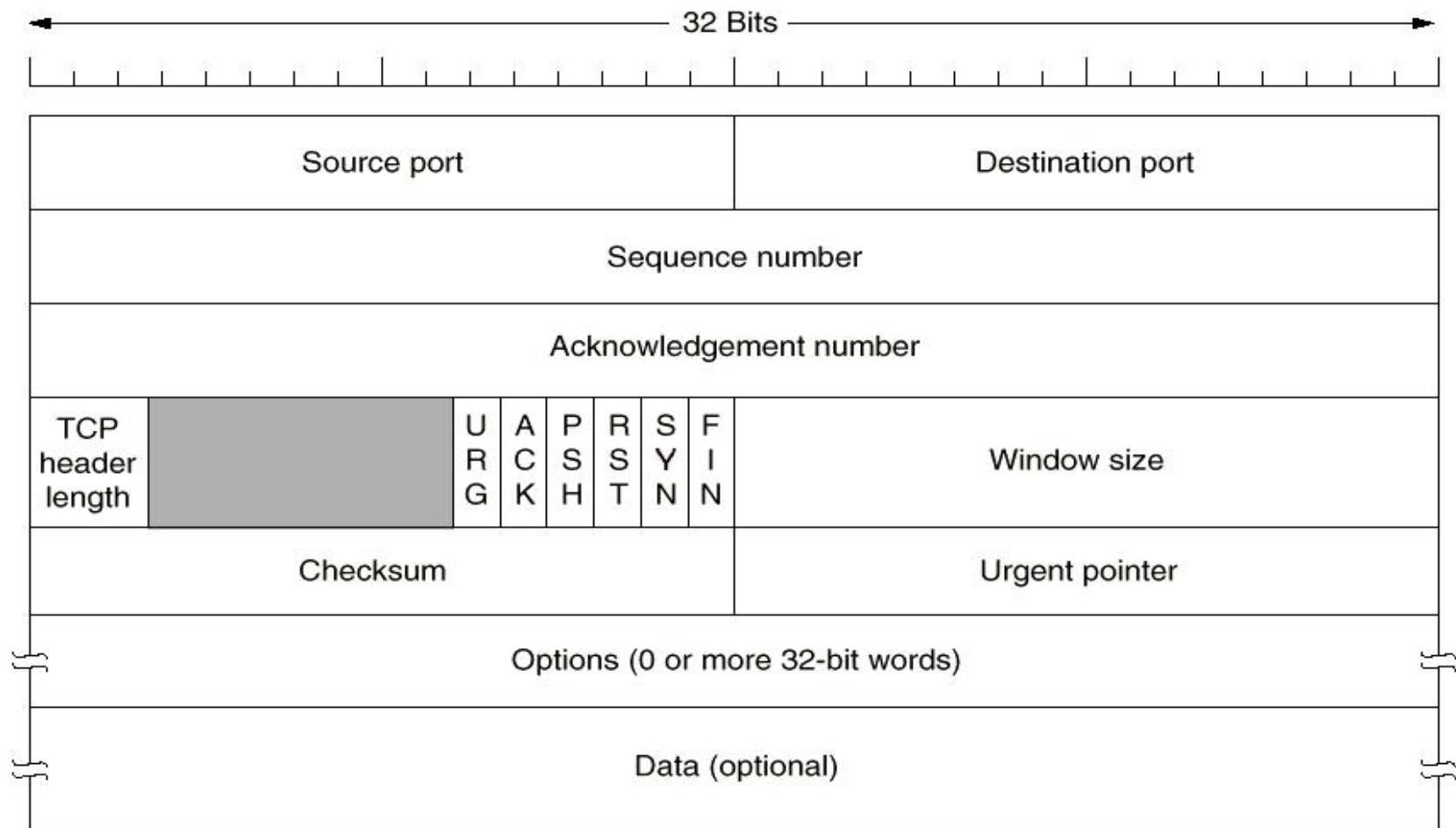
- 复位**RST**: 1bit, 为1时, 表明**TCP**连接出现严重差错(如由于主机崩溃), 须释放连接后重新建立连接
- 同步**SYN**: 1bit, 为1时, 表示这是一个连接请求或连接接受报文
- 终止**FIN**: 1bit, 为1时, 表示要求释放**TCP**连接
- 窗口大小: 2字节, 用来让对方设置发送窗口的依据, 单位为字节



- 检验和：2字节，伪首部+首部+数据的校验和
 - 伪首部(pseudoheader)格式与UDP的伪首部相同



- 紧急指针：2字节，指出本报文段中紧急数据共有多少个字节(紧急数据放在数据的最前面)
- 选项：长度可变，最长40字节
 - 最早定义的一种选项：最大报文段长度MSS(Maximum Segment Size)
 - 告知对方报文段中数据最大长度，双方可使用不同的MSS，缺省MSS=536字节
 - 后续增加的选项：窗口扩大选项、时间戳选项、选择确认选项
- 填充字段：为了使整个首部长度是4字节的整数倍



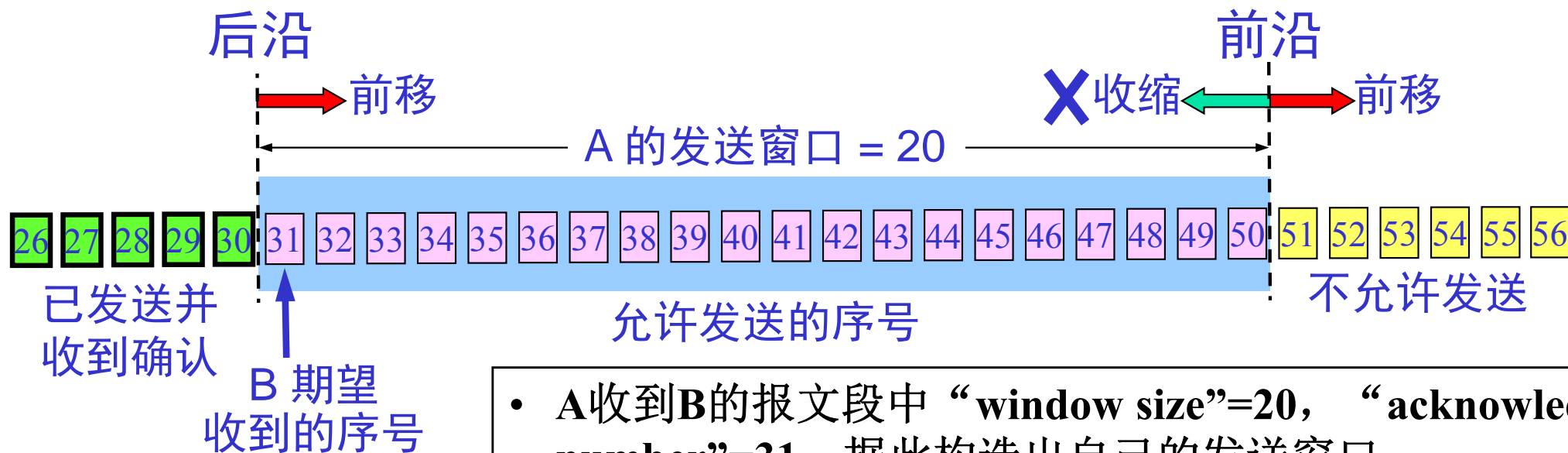
TCP报文示例：HTTP

	源端口号						目的端口号						Protocol number						
00000000:	00	d0	d0	72	a4	b4	00	16	ea	c3	8c	6c	08	00	45	00			.行r...答...E.
00000010:	01	cf	9d	51	40	00	80	06	db	00	c0	a8	01	04	db	ef			.端Q@.?.括...域
00000020:	e3	3a	0b	3d	00	50	91	19	b8	27	5f	bc	d4	ca	50	18			?...=.P??_英...度
00000030:	44	70	a1	4b	00	00	47	45	54	20	2f	20	48	54	54	50			Dp...GET / HTTP
00000040:	2f	31	2e	31	0d	0a	41	63	63	65	70	74	3a	20	69	6d			/1.1..Accept: im
00000050:	61	67	65	2f	67	69	66	2c	20	69	6d	61	67	65	2f	78			age/gif, image/x
00000060:	2d	78	62	69	74	6d	61	70	2c	20	69	6d	61	67	65	2f			-xbitmap, image/
00000070:	6a	70	65	67	2c	20	69	6d	61	67	65	2f	70	6a	70	65			jpeg, image/pjpe
00000080:	67	2c	20	61	70	70	6c	69	63	61	74	69	6f	6e	2f	76			g, application/v
00000090:	6e	64	2e	6d	73	2d	65	78	63	65	6c	2c	20	61	70	70			nd.ms-excel, app
000000a0:	6c	69	63	61	74	69	6f	6e	2f	76	6e	64	2e	6d	73	2d			lication/vnd.ms-
000000b0:	70	6f	77	65	72	70	6f	69	6e	74	2c	20	61	70	70	6c			powerpoint, appl
000000c0:	69	63	61	74	69	6f	6e	2f	6d	73	77	6f	72	64	2c	20			ication/msword,
000000d0:	61	70	70	6c	69	63	61	74	69	6f	6e	2f	78	2d	73	68			application/x-sh
000000e0:	6f	63	6b	77	61	76	65	2d	66	6c	61	73	68	2c	20	2a			ockwave-flash, *
000000f0:	2f	2a	0d	0a	41	63	63	65	70	74	2d	4c	61	6e	67	75			/*..Accept-Langu
00000100:	61	67	65	3a	20	7a	68	2d	63	6e	0d	0a	55	41	2d	43			age: zh-cn..UA-C
00000110:	50	55	3a	20	78	38	36	0d	0a	41	63	63	65	70	74	2d			PU: x86..Accept-
00000120:	45	6e	63	6f	64	69	6e	67	3a	20	67	7a	69	70	2c	20			Encoding: gzip,
00000130:	64	65	66	6c	61	74	65	0d	0a	55	73	65	72	2d	41	67			deflate..User-Ag
00000140:	65	6e	74	3a	20	4d	6f	7a	69	6c	6c	61	2f	34	2e	30			ent: Mozilla/4.0
00000150:	20	28	63	6f	6d	70	61	74	69	62	6c	65	3b	20	4d	53			(compatible; MS
00000160:	49	45	20	37	2e	30	3b	20	57	69	6e	64	6f	77	73	20			IE 7.0; Windows
00000170:	4e	54	20	35	2e	31	3b	20	2e	4e	45	54	20	43	4c	52			NT 5.1; .NET CLR
00000180:	20	31	2e	31	2e	34	33	32	32	3b	20	2e	4e	45	54	20			1.1.4322; .NET
00000190:	43	4c	52	20	32	2e	30	2e	35	30	37	32	37	3b	20	49			CLR 2.0.50727; I
000001a0:	6e	66	6f	50	61	74	68	2e	31	29	0d	0a	48	6f	73	74			nfoPath.1)..Host
000001b0:	3a	20	77	77	77	2e	62	75	61	61	2e	65	64	75	2e	63			: www.buaa.edu.c
000001c0:	6e	0d	0a	43	6f	6e	6e	65	63	74	69	6f	6e	3a	20	4b			n..Connection: K
000001d0:	65	65	70	2d	41	6c	69	76	65	0d	0a	0d	0a						eep-Alive....

5.5 TCP可靠传输的实现

一、以字节为单位的滑动窗口

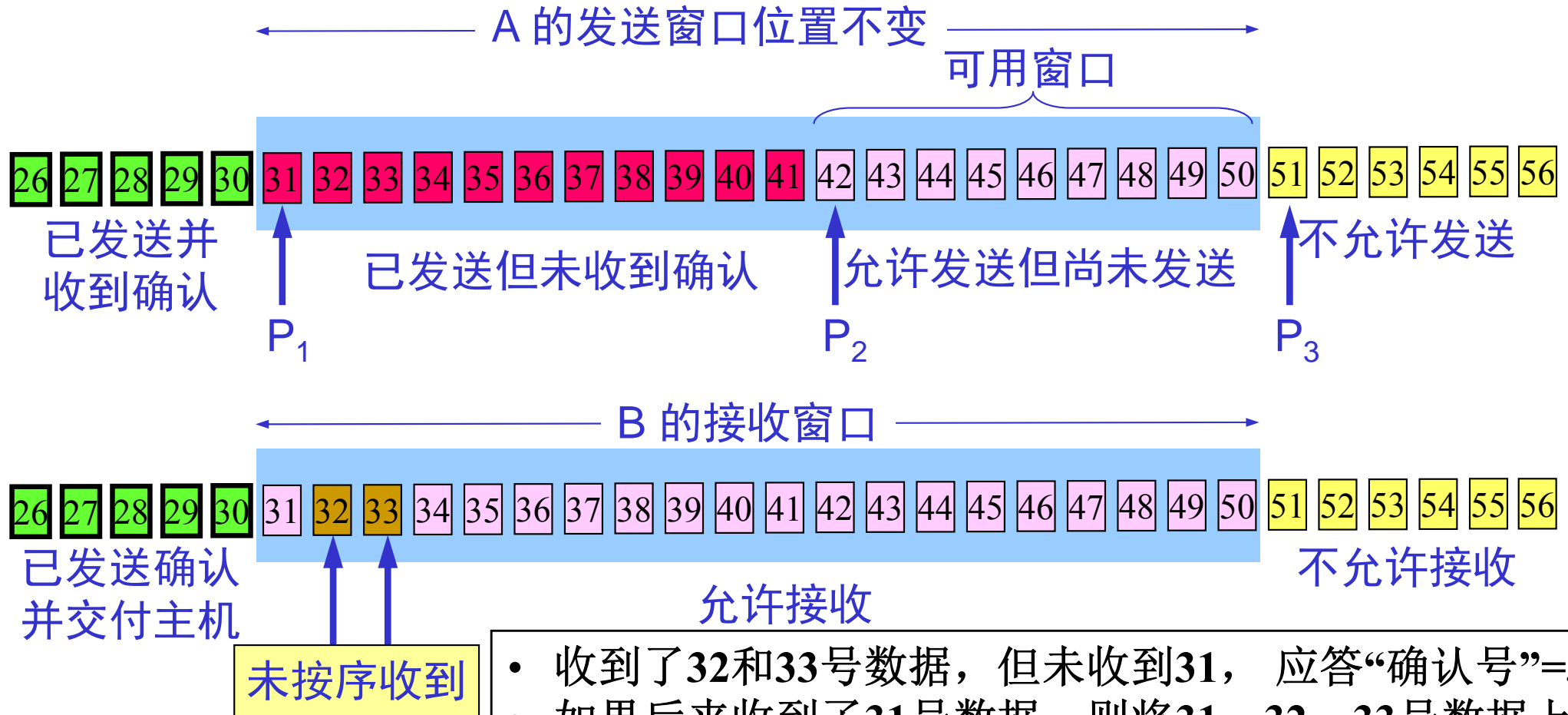
- **TCP基于滑动窗口协议实现可靠传输和流量控制，滑动窗口以字节为单位**
- 发送窗口
 - 在没有收到对方应答的情况下，可以连续把窗口内的数据发送出去
 - 即：窗口内的序号是允许发送的序号
 - 窗口大小的确定：对方发来的窗口大小、拥塞控制
 - 根据收到对方的TCP报文段头部中“**acknowledge number**”字段，窗口向前滑动
 - “**acknowledge number**”字段表示在此序号之前的数据已被正确接收



- A收到B的报文段中“**window size**”=20，“**acknowledge number**”=31，据此构造出自己的发送窗口
- 当对方通知的窗口尺寸变化时，发送窗口前沿可能需要向后收缩，但TCP标准强烈不赞成这样做

- $P3 - P1 = A$ 的发送窗口(又称为通知窗口)
- $P2 - P1 =$ 已发送但尚未收到确认的字节数
- $P3 - P2 =$ 允许发送但尚未发送的字节数(又称为可用窗口)

A发送11字节后

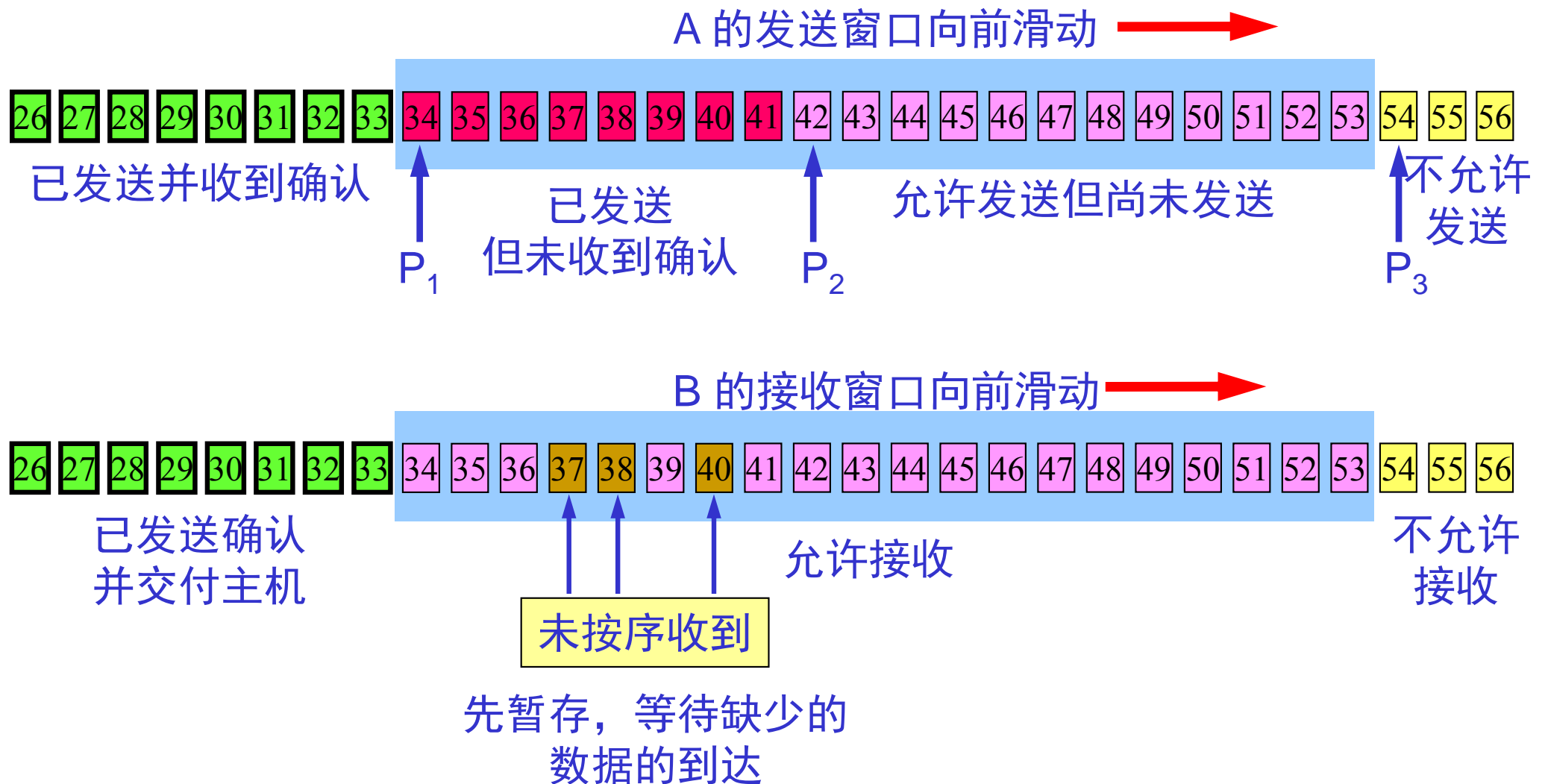


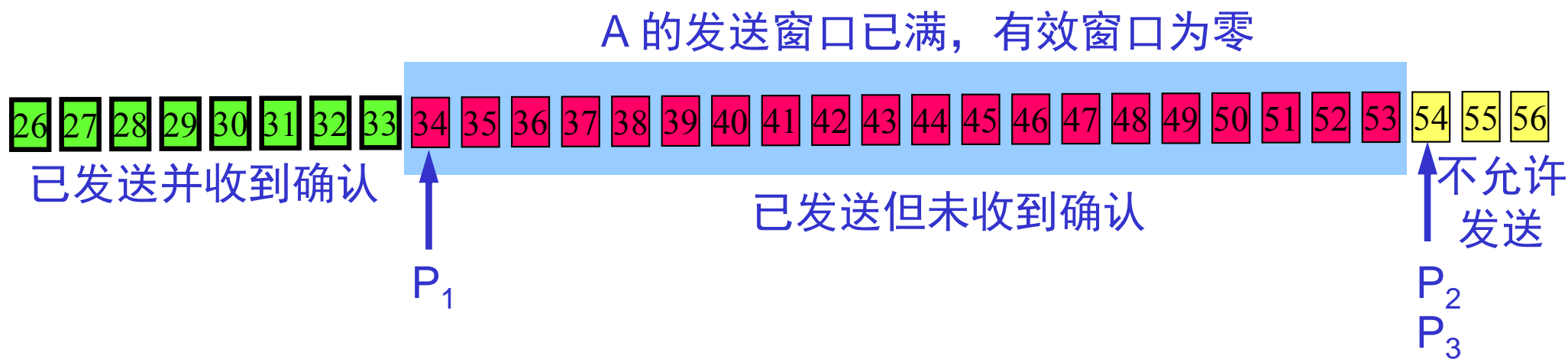
- 收到了32和33号数据，但未收到31， 应答“确认号”=31
- 如果后来收到了31号数据，则将31、32、33号数据上交，窗口后沿前移，应答中“确认号”=34

接收窗口

- 窗口内的数据是允许接收的
- 窗口后沿以外是已正确接收并交付上层的数据

- B收到31号数据后，窗口前移3字节，并返回确认号34，假定窗口宽度仍为20
- A收到确认号34后，窗口前移3字节
- B收到37、38、40号数据，由于数据未按序收全，不能上交

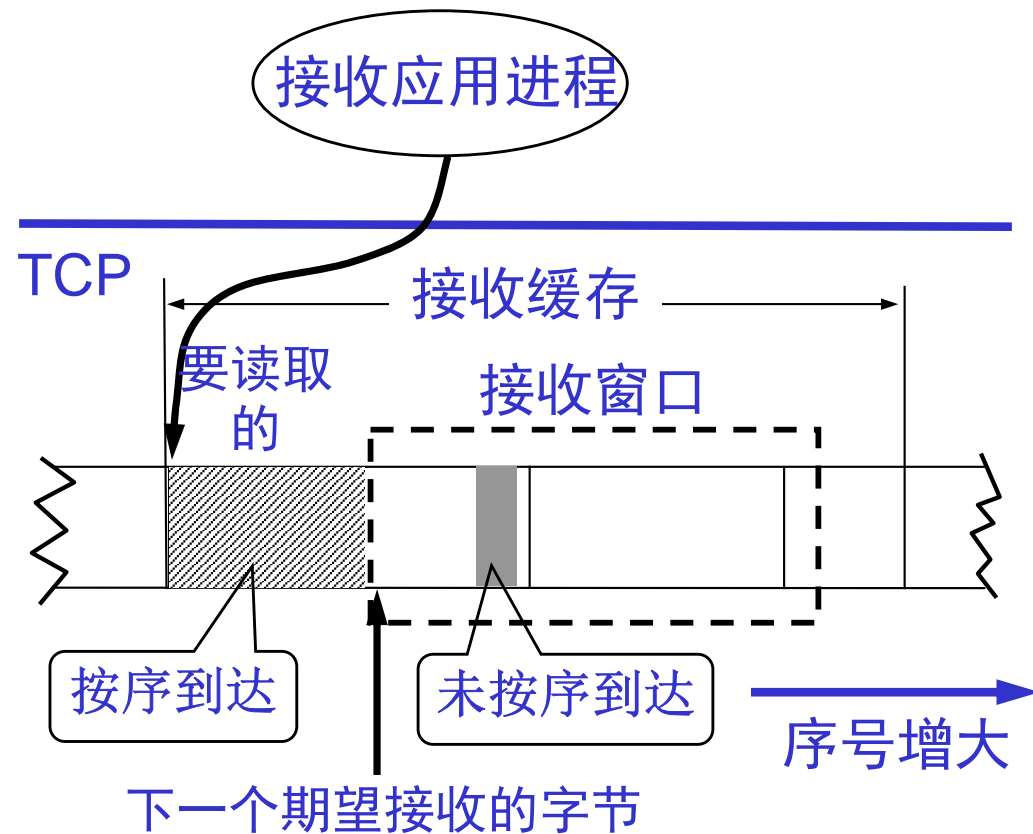
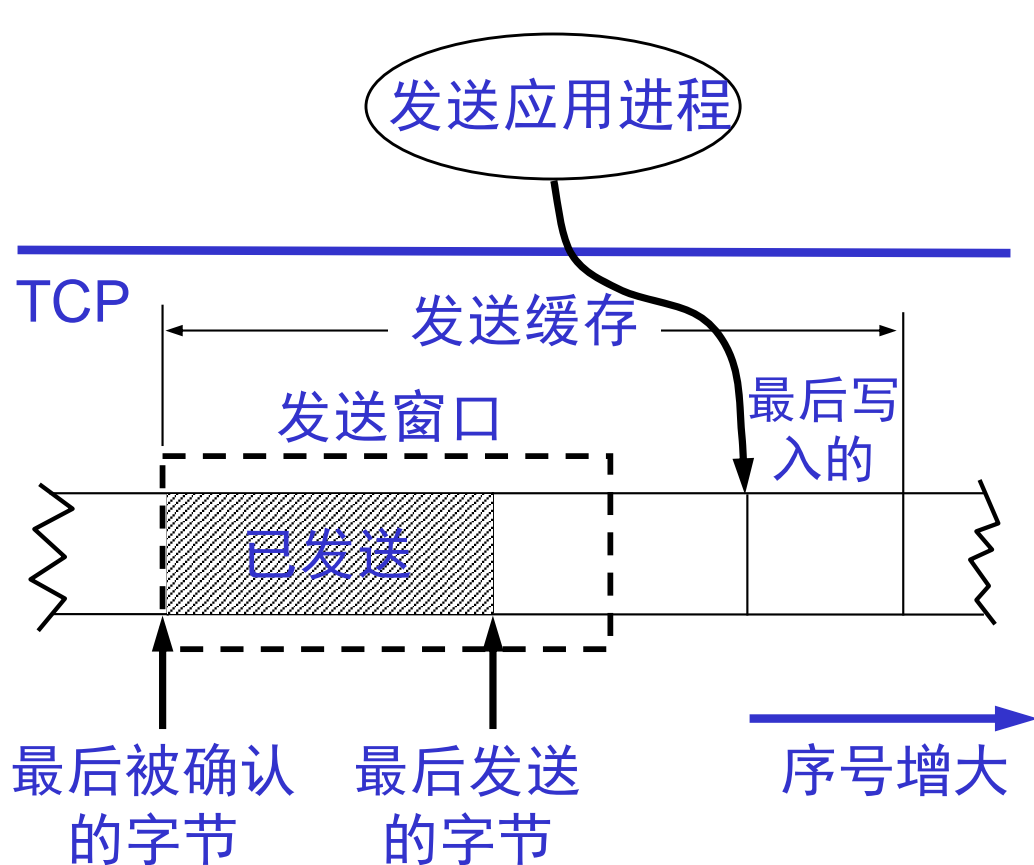




- A将发送窗口中未发送的数据全部发送后，发送窗口满，停止发送
- 如果未收到B返回的应答，A等待超时后将重发部分数据，直到收到应答为止

问：用软件实现TCP协议时，如何实现滑动窗口？

- 发送缓存用来暂时存放：
 - 发送应用进程传送给TCP，但尚未发出的数据
 - 已发送，但尚未收到确认的数据
- 接收缓存用来暂时存放：
 - 按序到达的、但尚未被接收应用进程读取的数据
 - 未按序到达的数据
- A的发送窗口并不总是和B的接收窗口一样大
- TCP要求接收方必须有累积确认的功能，这样可以减小传输开销




2009年考研题

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

A. 500

B. 700

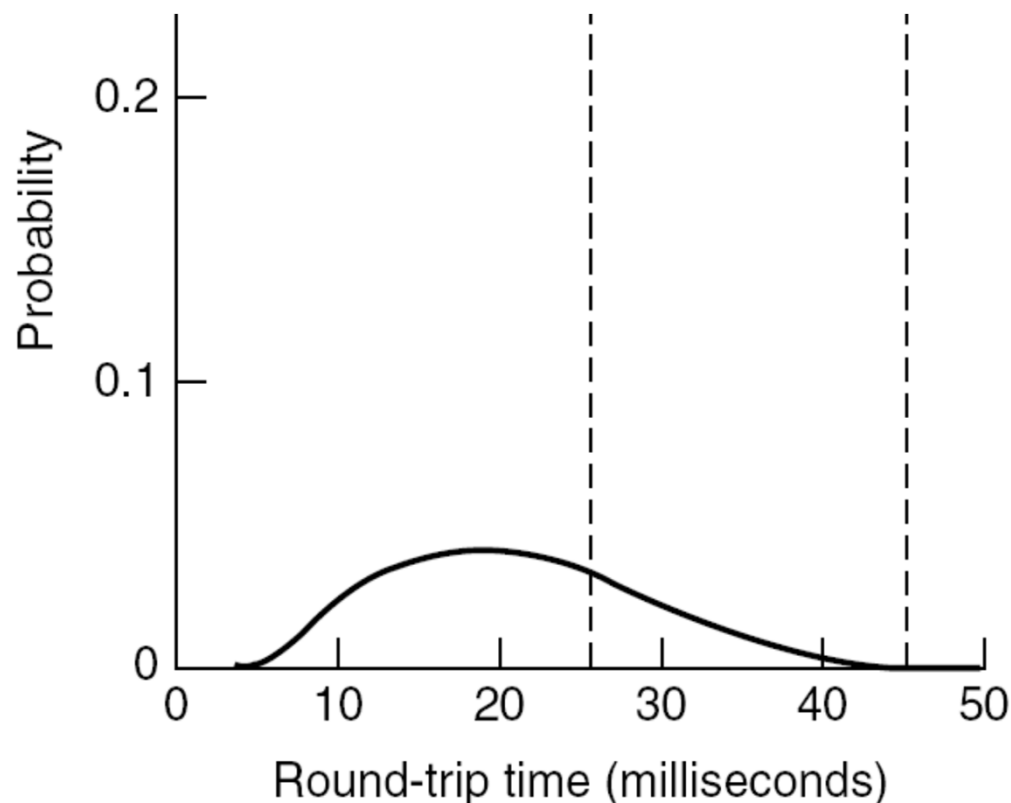
C. 800

 D. 1000

5.5 TCP可靠传输的实现

二、超时重传时间的选择

- TCP的可靠传输通过**校验和+超时重传**实现
 - TCP每发送一个报文段，就对这个报文段设置一次计时器
 - 如果计时器设置的重传时间到，但还没有收到确认，就要重传该报文段
- 超时时间的设置是一个复杂的问题
 - IP层提供数据报服务，每个数据报所选择的路由都可能有变化
 - 传输层的往返时间变化较大



传输层往返时延的概率分布

5.5 TCP可靠传输的实现

二、超时重传时间的选择

- TCP采用一种自适应算法计算超时重传时间
- 加权平均往返时间 RTT_s (注: RTT —*Round Trip Time*)
 - TCP 保留了 RTT 的一个加权平均往返时间 RTT_s (又称为平滑的往返时间)
 - 第一次测量到 RTT 样本时, RTT_s 就取为该值
 - 以后每测量到一个新的 RTT 样本, 按下式重新计算:

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

其中 $0 < \alpha < 1$, α 越小, RTT 值更新越慢, α 越大, RTT 值更新越快

- RFC 2988 推荐 $\alpha=1/8$, 即 0.125

问: 为什么不直接用单次的 RTT 值?

5.5 TCP可靠传输的实现

二、超时重传时间的选择

- 超时重传时间 ***RTO*** (Retransmission Time-Out)

- ***RTO*** 应略大于 ***RTT_S***

问：为什么？

- RFC 2988 建议使用下式计算 ***RTO***:

$$RTO = RTT_S + 4 \times RTT_D$$

- ***RTT_D*** 是 ***RTT*** 的偏差的加权平均值

- RFC 2988 建议这样计算 ***RTT_D***

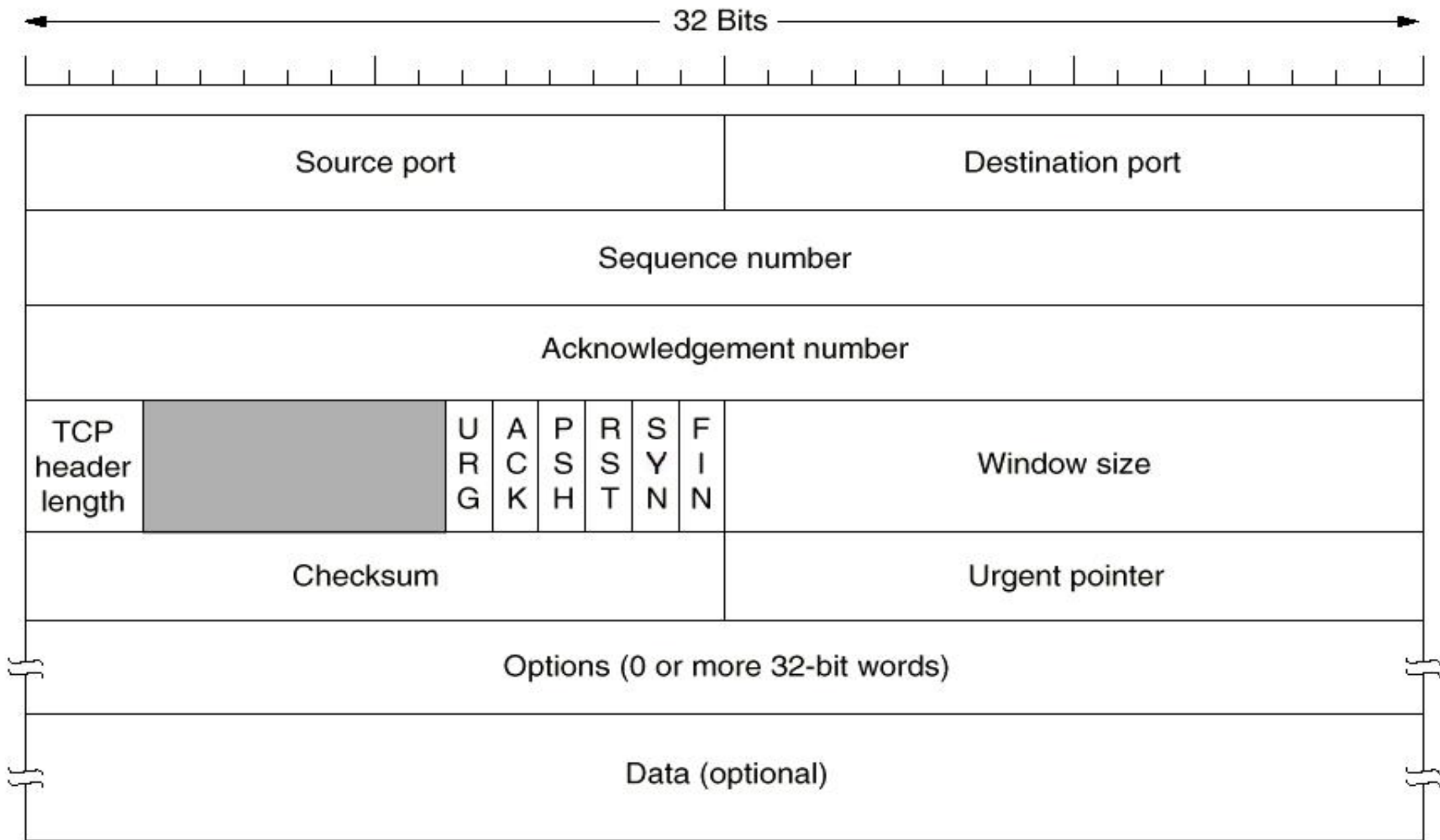
- 第一次测量时, ***RTT_D*** 值取为测量到的 ***RTT*** 样本值的一半

- 在以后的测量中, 使用下式计算加权平均的 ***RTT_D***:

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

其中 $0 < \beta < 1$, 推荐值是 $1/4$, 即 **0.25**

5.6 TCP的流量控制



5.6 TCP的流量控制

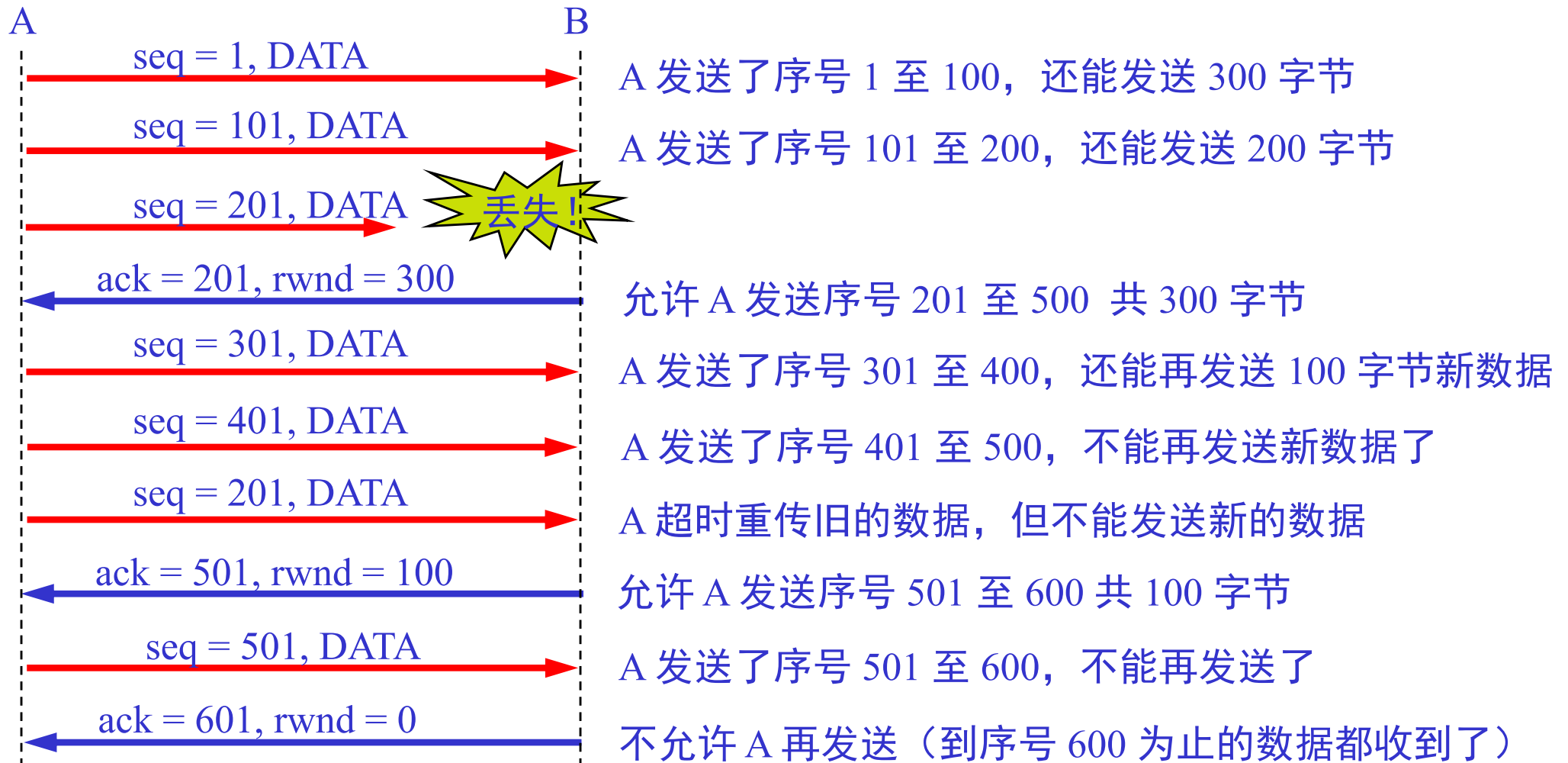
一、利用滑动窗口实现流量控制

- **流量控制(flow control)**就是让发送方的发送速率不要太快，使接收方来得及接收
- 利用滑动窗口机制可以很方便地在 TCP 连接上实现流量控制
 - TCP连接建立过程中，将自己的接收窗口大小告知对方
 - TCP报头中都携带有窗口大小字段，告知对方自己接收窗口的剩余大小(即可接收的字节数)
 - 发送方的发送窗口应不大于对方的接收窗口

Window size字段

TCP流量控制示例

- A 向 B 发送数据
- 在连接建立时, B 告诉 A: “我的接收窗口 **rwnd = 400(字节)**”



B的接收缓冲有空间时, 将向**A**发送**TCP**报文, 告知窗口大小

问题: 如果**B**向**A**发送的这个报文丢失了, 如何避免双方相互等待?

5.6 TCP的流量控制

一、利用滑动窗口实现流量控制

- 持续计时器(persistence timer)
 - TCP为每一个连接设有一个持续计时器
 - 只要一方收到对方的零窗口通知，就启动持续计时器
 - 若持续计时器设置的时间到，就发送一个零窗口探测报文段(仅携带1字节的数据)，而对方在确认这个探测报文段时给出当前窗口值
 - 若窗口仍然是零，则收到这个报文段的一方就重新设置持续计时器
 - 若窗口不是零，则死锁的僵局就可以打破了

5.6 TCP的流量控制

二、传输效率

- 可以用不同的机制来控制TCP报文段的发送时机
- 机制一：缓存数据达到一定量就发送
 - TCP维持一个变量，它等于最大报文段长度MSS
 - 只要缓存中存放的数据达到MSS字节时，就组装成一个TCP报文段发送出去
- 机制二：应用进程控制
 - 由发送方的应用进程指明要求发送报文段，即推送(push)操作
- 机制三：定时发送
 - 发送方的一个计时器期限到了，这时就把当前已有的缓存数据装入报文段(但长度不能超过 MSS)发送出去

5.7 TCP的拥塞控制

5.7 TCP的拥塞控制

一、拥塞控制的一般原理

- 拥塞(congestion)概念和产生原因(1/2)

- 在某段时间，若对网络中某资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏，从而产生拥塞，即拥塞产生的条件：

Σ 对资源的需求 > 可用资源

- 网络资源：链路带宽、路由节点缓存及处理能力等
- 网络拥塞是由多种原因引起的，通过简单地增加某种资源数量往往并不能消除拥塞
 - 例：某路由器缓存容量太小，造成到达该结点的报文丢失，如果增加缓存容量，可能的结果：
 - 报文可在缓存中排队，但如果路由器处理能力和出口链路带宽未增加，报文排队时间过长，发送主机将超时重发→拥塞没有解决

5.7 TCP的拥塞控制

一、拥塞控制的一般原理

- 拥塞(congestion)概念和产生原因(2/2)

- 网络拥塞的表现

- 网络性能下降，具体表现为网络吞吐率下降、报文传输时延增大、丢包率增加、用户端响应时间变长、...

- 拥塞常常会趋于恶化，即恶性循环

- 由**TCP重传机制**引起

- 如：一个路由器由于缓存不足丢弃了部分报文，发送端主机将超时重发，导致网络中被注入更多的报文，从而加剧拥塞

- 拥塞控制(congestion control)与流量控制

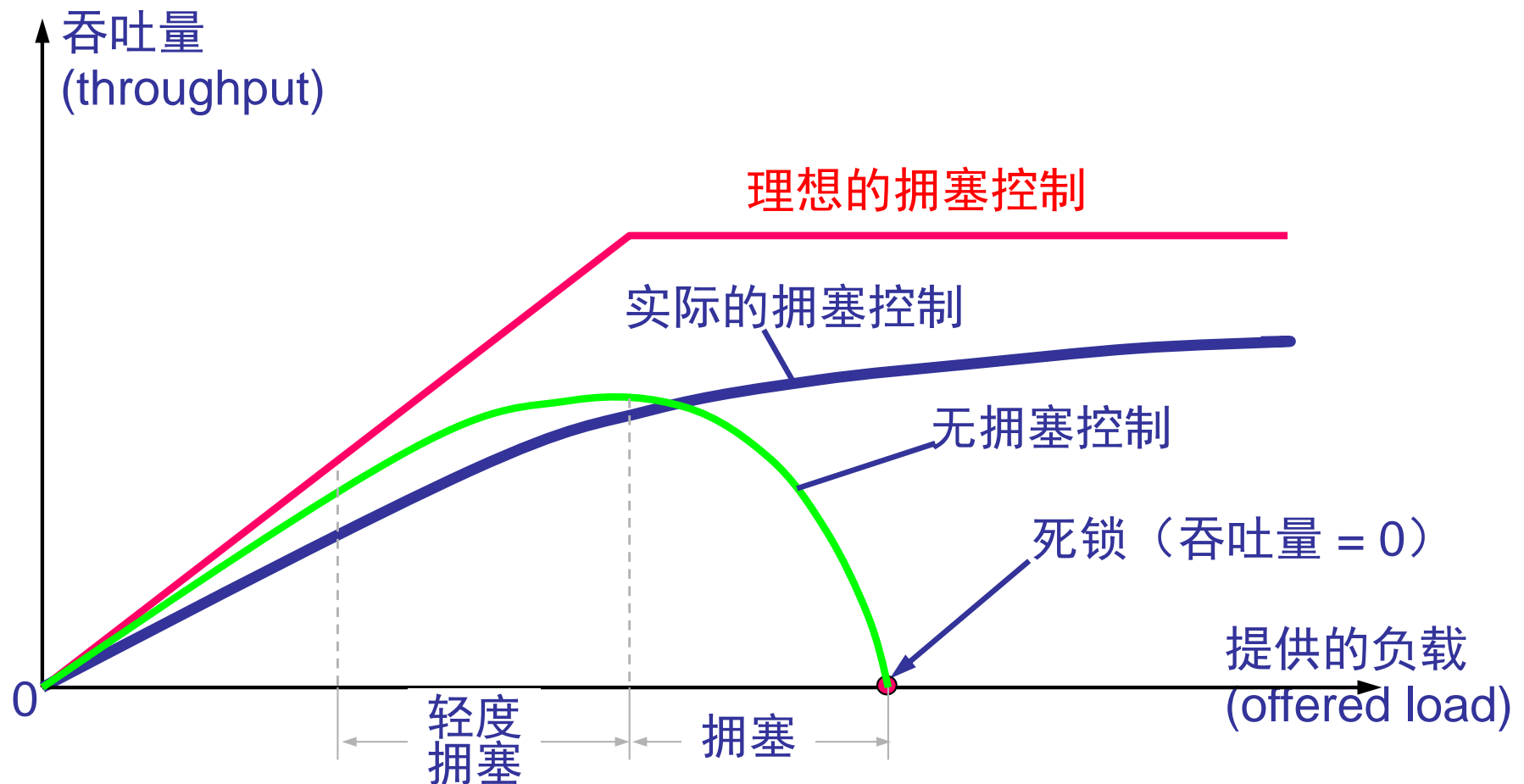
- 拥塞控制：防止过多的数据注入到网络中，使网络中的路由器或链路不致过载

- 拥塞控制是全局性的过程，而流量控制是点对点通信量的控制，是端到端的问题

5.7 TCP的拥塞控制

一、拥塞控制的一般原理

- 拥塞控制的作用



5.7 TCP的拥塞控制

一、拥塞控制的一般原理

- 开环控制与闭环控制

- 开环控制

- 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：基于反馈环路，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

- RFC 2581中定义了四种拥塞控制方法

- ① 慢启动(slow-start)，又称为慢开始

- ② 拥塞避免(congestion avoidance)

- ③ 快重传(fast retransmit)

- ④ 快恢复(fast recovery)

5.7 TCP的拥塞控制

二、拥塞控制方法：慢启动(slow-start)和拥塞避免

- 慢启动进行拥塞控制的基本思路
 - 开始发送时，如果立即把大量数据注入网络，则可能导致拥塞
 - 因为此时不知道网络的负荷状况
 - 较好的方法是：试探性地从小到大逐渐增大发送窗口
- 拥塞窗口cwnd (congestion window)
 - 发送方维持拥塞窗口，它是一个状态变量
 - 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
 - 只要没有出现拥塞，拥塞窗口就增大一些，以便把更多的分组发送出去
 - 一旦出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数

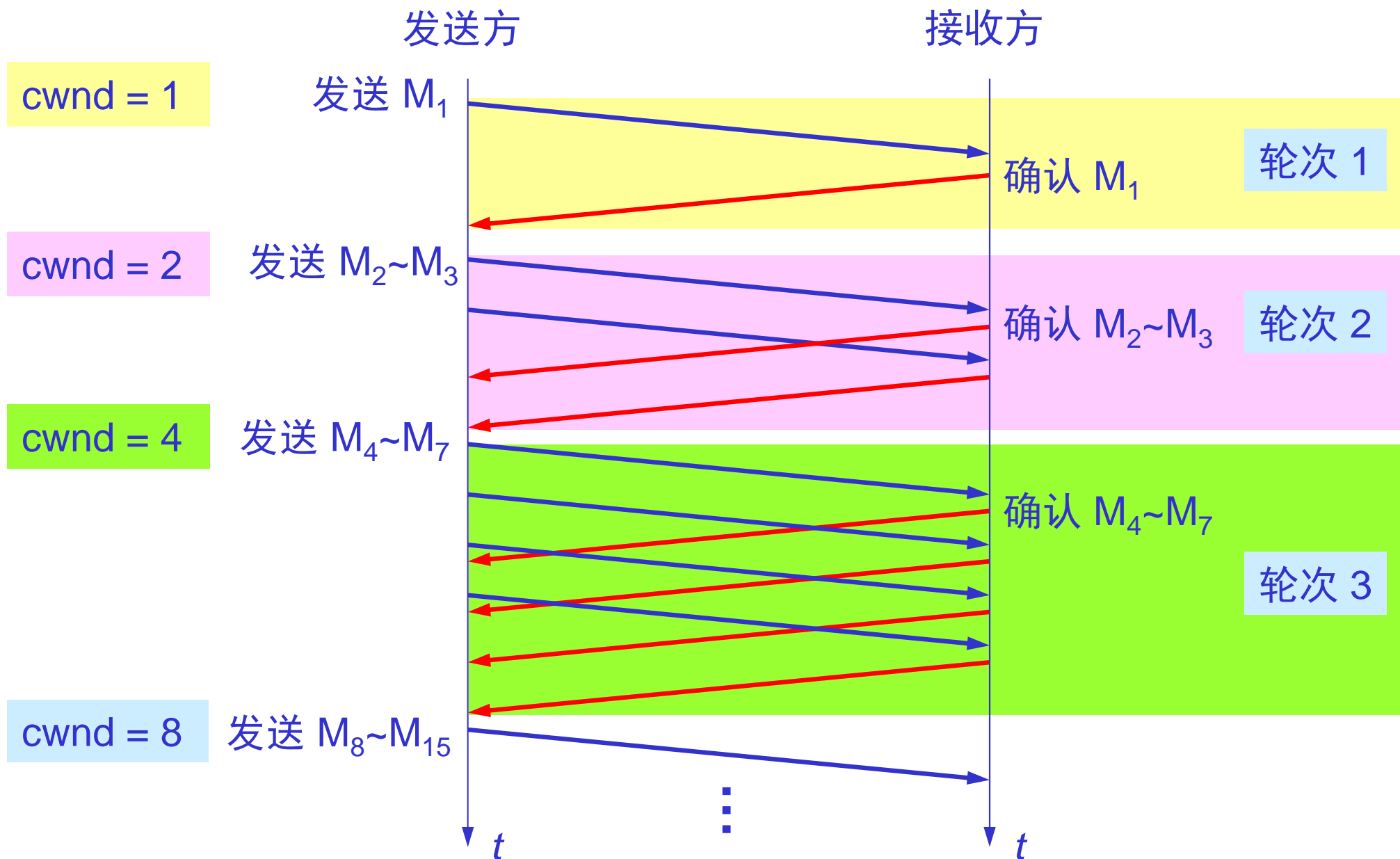
5.7 TCP的拥塞控制

二、拥塞控制方法：慢启动(slow-start)和拥塞避免

- 如何判断是否发生拥塞？
 - 拥塞发生时路由器将丢弃分组
 - 简单的主机端判断方法：只要没有收到确认，就认为发生了拥塞
- 慢启动的工作过程
 - 开始发送报文段时，设置拥塞窗口 $cwnd = 1$ ，即设置为一个最大报文段MSS的数值
 - 每收到一个对新的报文段的确认后，将拥塞窗口加1，即增加一个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
- 注意：TCP中窗口大小是以字节为单位的
为方便起见，这里以MSS为单位讨论拥塞窗口大小

注意: **cwnd**=1表示1个MSS长度, 不是1字节

慢启动工作过程示例



5.7 TCP的拥塞控制

二、拥塞控制方法：慢启动(slow-start)和拥塞避免

- 传输轮次(transmission round)
 - 使用慢启动时，每经过一个传输轮次，拥塞窗口**cwnd**就加倍
 - 一个传输轮次所经历的时间其实就是往返时间**RTT**
 - 传输轮次更加强调：
 - 把拥塞窗口**cwnd**所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认
 - 例如，拥塞窗口 **cwnd** = 4，这时的往返时间**RTT**就是发送方连续发送 4 个报文段，并收到这 4 个报文段的确认，总共经历的时间
- 为防止拥塞窗口**cwnd**增长过大导致拥塞，设置一个慢启动门限**ssthresh**
 - **cwnd** < **ssthresh**时，使用慢启动算法
 - **cwnd** > **ssthresh**时，停止使用慢启动算法而改用拥塞避免算法
 - **cwnd** = **ssthresh**时，可使用慢启动算法或拥塞避免算法

5.7 TCP的拥塞控制

二、拥塞控制方法：慢启动(slow-start)和拥塞避免

- 拥塞避免算法的思路

- 让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间 RTT 就把发送方的拥塞窗口cwnd加1，而不是加倍，使拥塞窗口cwnd按线性规律缓慢增长

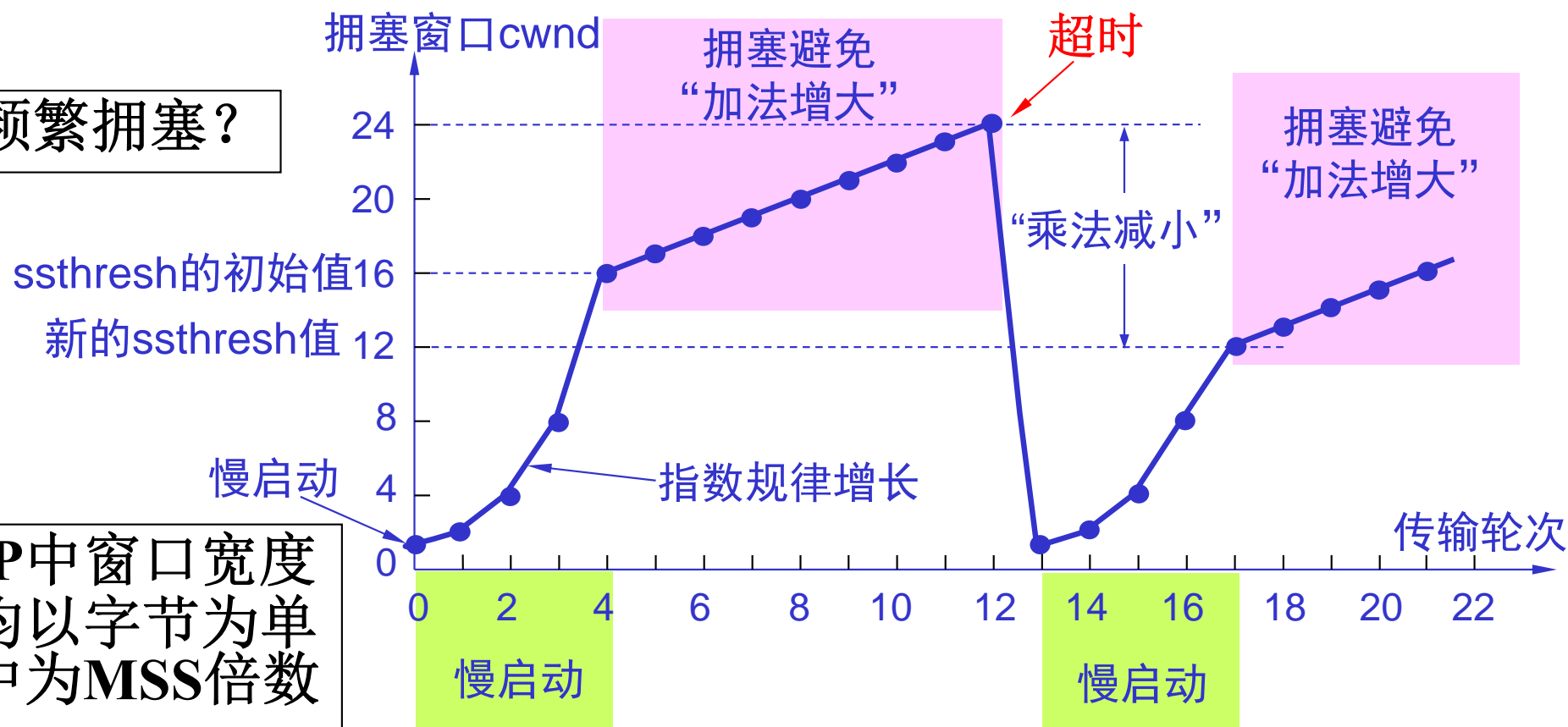
- 拥塞发生时的处理

- 无论在慢启动阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞(没有按时收到确认)，就要把慢启动门限 $ssthresh$ 改为出现拥塞时的发送方窗口值的一半(但不能小于2)
- 然后把拥塞窗口cwnd重新设置为1，执行慢启动算法
- 这样做的目的：迅速减少注入到网络中的分组数，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

慢启动和拥塞避免过程示例

- TCP 连接初始化时, 拥塞窗口 $cwnd=1$, 慢启动门限 $ssthresh=16$, 设发送窗口宽度=拥塞窗口宽度
- 在慢开始算法阶段, 发送端每收到一个确认, 就把 $cwnd$ 加1, 即每经过1个往返时间 RTT , 拥塞窗口 $cwnd$ 加倍
- 当拥塞窗口 $cwnd \geq$ 慢启动门限 $ssthresh$ 时, 进入拥塞避免阶段, 每经过1个往返时间 RTT , 拥塞窗口 $cwnd+1$
- 假定当拥塞窗口 $cwnd=24$ 时发生了拥塞, $ssthresh$ 减为12, 拥塞窗口 $cwnd=1$, 并重新开始慢启动过程

问：如果频繁拥塞？



注意：TCP 中窗口宽度和门限值均以字节为单位，本例中为 MSS 倍数

5.7 TCP的拥塞控制

二、拥塞控制方法：慢启动(slow-start)和拥塞避免

- 乘法减小(multiplicative decrease)
 - 不论在慢启动阶段还是拥塞避免阶段，只要出现一次超时(即网络拥塞)，就把慢启动门限*ssthresh*减小到拥塞窗口的一半
 - 当网络频繁出现拥塞时，*ssthresh*值下降得很快，以大大减少注入到网络中的分组数
- 加法增大(additive increase)
 - 执行拥塞避免算法后，每经过一个往返时间*RTT*，把拥塞窗口*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


5.7 TCP的拥塞控制

2009年的一道考研题

- 一个TCP连接总是以1KB的最大段长度发送TCP段，发送方有足够的~~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)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：

A、7KB


B、8KB

 C、9KB

D、16KB

2010年的一道考研题

- 主机甲和主机乙之间建立一个TCP连接，TCP最大段长度为1000字节，若主机甲的当前拥塞窗口为4000字节，在主机甲向主机乙连续发送2个最大段后，成功收到主机乙发送的第一段的确认段，确认段中通告的接收窗口大小为2000字节，则此时主机甲还可以向主机乙发送的最大字节数是：

 A、1000

B、2000

C、3000

D、4000

5.7 TCP的拥塞控制

三、拥塞控制方法：快重传和快恢复

超时时间较长，待超时后再做拥塞处理有些过晚

- 快重传(fast retransmit)算法思路

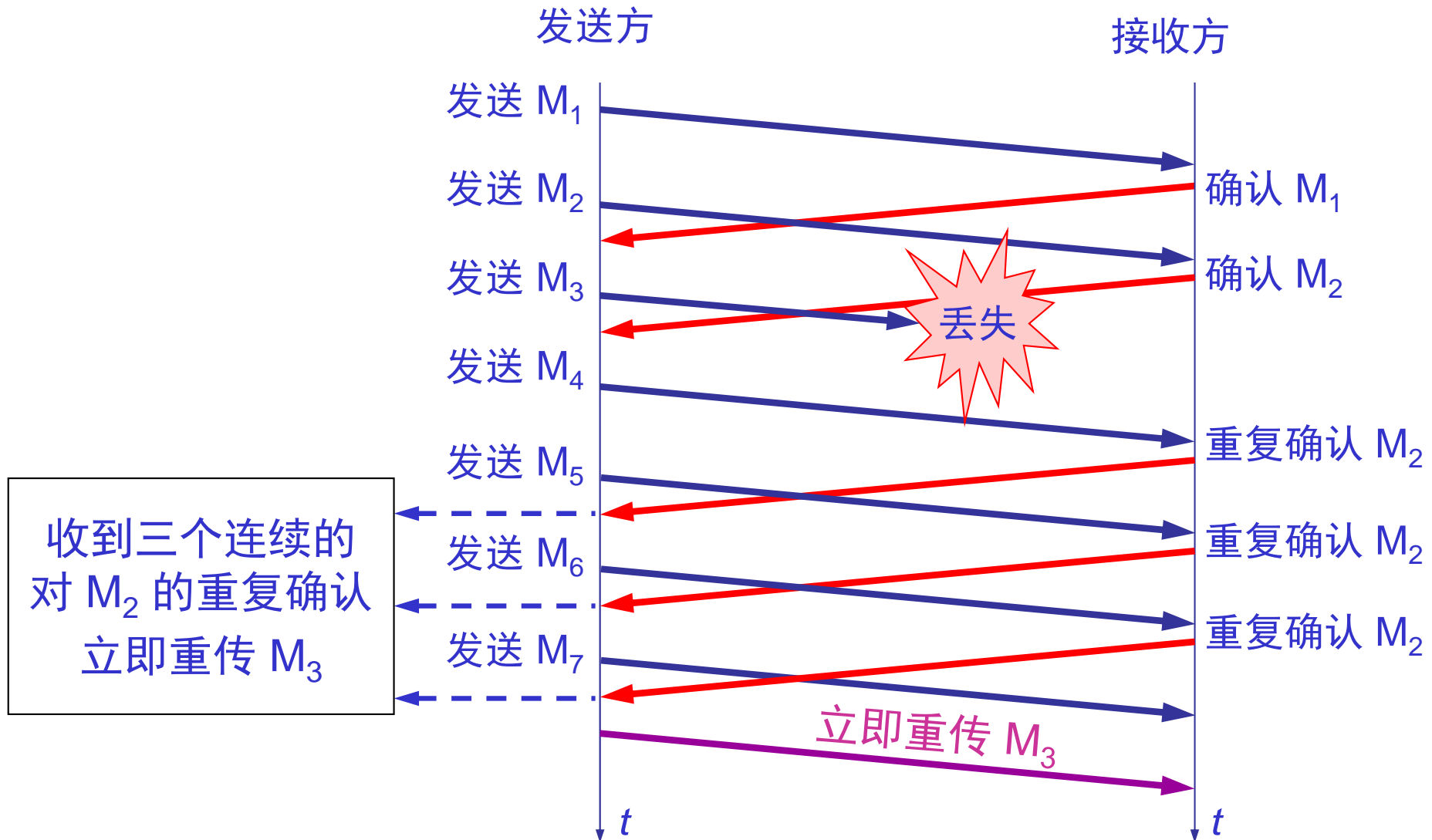
- 如果发送方在超时时间内未收到确认报文，说明网络中发生了拥塞，此时发送方应**尽早地**减小窗口宽度
- 每当接收方收到一个失序的报文段，就发出重复确认，以便使发送方及早知道有报文段没有到达接收方
- 发送方只要接连收到三个重复确认就应当立即重传对方尚未收到的报文段

- 快恢复(fast recovery)算法思路

- 当发送端收到连续三个重复的确认时，就进行“乘法减小”，把慢启动门限 $ssthresh$ 减为当前拥塞窗口宽度的一半
- 接下来不执行慢启动算法，而是设置为慢启动门限 $ssthresh$ 减半后的数值，然后开始执行拥塞避免算法(“加法增大”)，使拥塞窗口缓慢地线性增大
 - 发送方认为现在网络很可能没有发生拥塞(仅有**拥塞征兆**)，因为拥塞时不会有连续多个报文段到达接收方

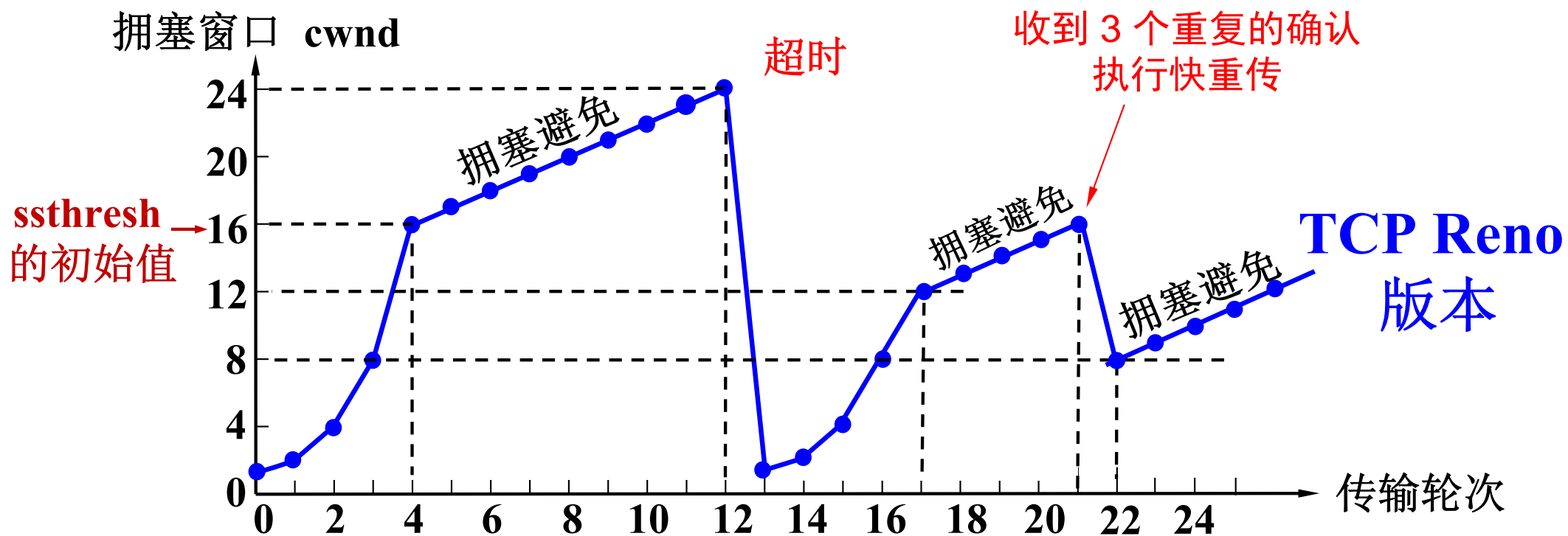
5.7 TCP的拥塞控制

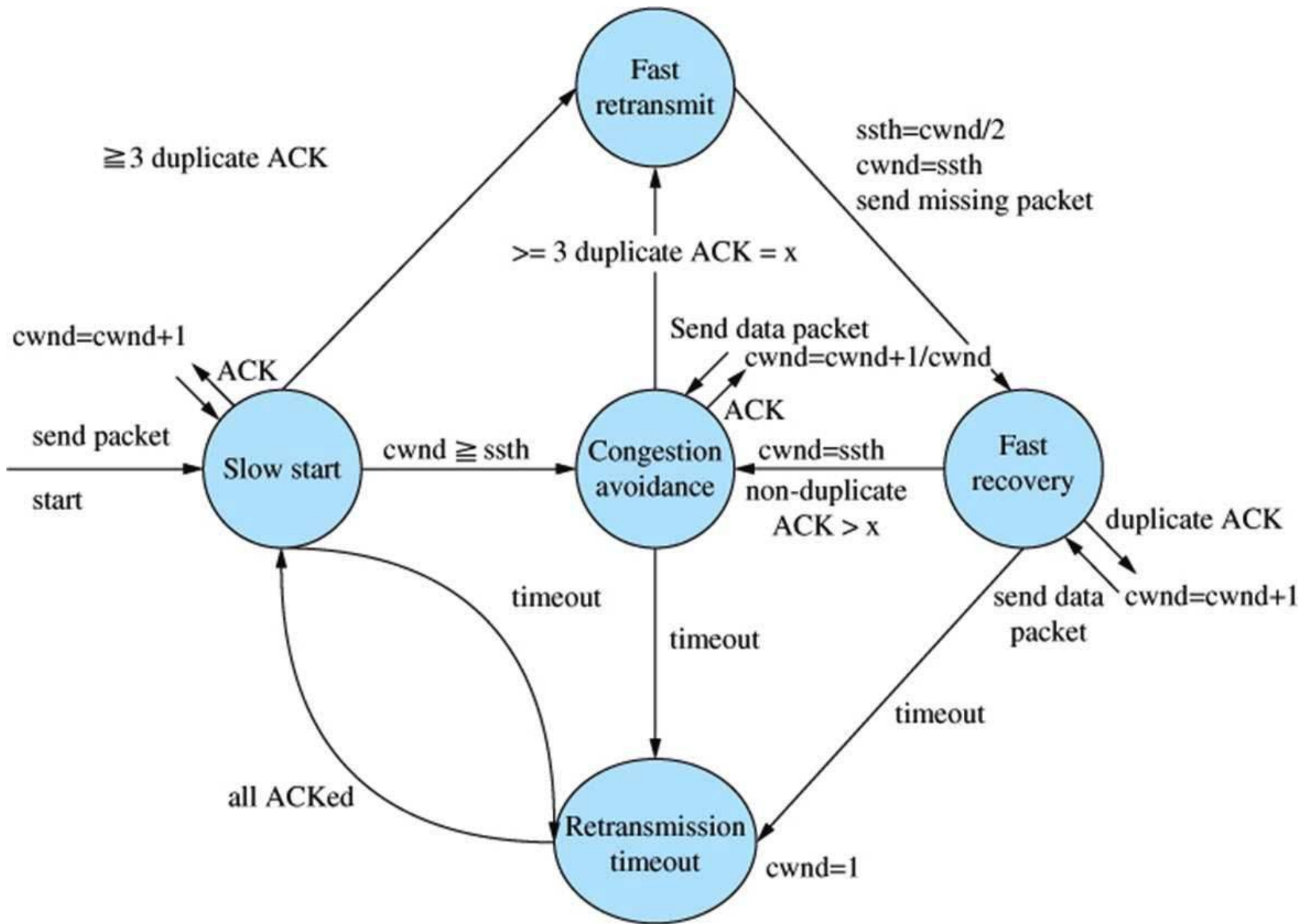
快重传示例



5.7 TCP的拥塞控制

连续收到三个重复确认后转入拥塞避免





TCP Reno拥塞控制的状态转换

5.7 TCP的拥塞控制

三、拥塞控制方法：快重传和快恢复

- 发送窗口的上限值

- 从流量控制角度考虑，发送窗口不能超过对方给出的接收窗口值
- 综合考虑流量控制和拥塞控制，发送方的发送窗口的上限值应当在接收方窗口**rwnd**和拥塞窗口**cwnd**两个者中取较小的一个：

发送窗口的上限值 = $\text{Min}[\text{rwnd}, \text{cwnd}]$

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

5.8 TCP的连接管理

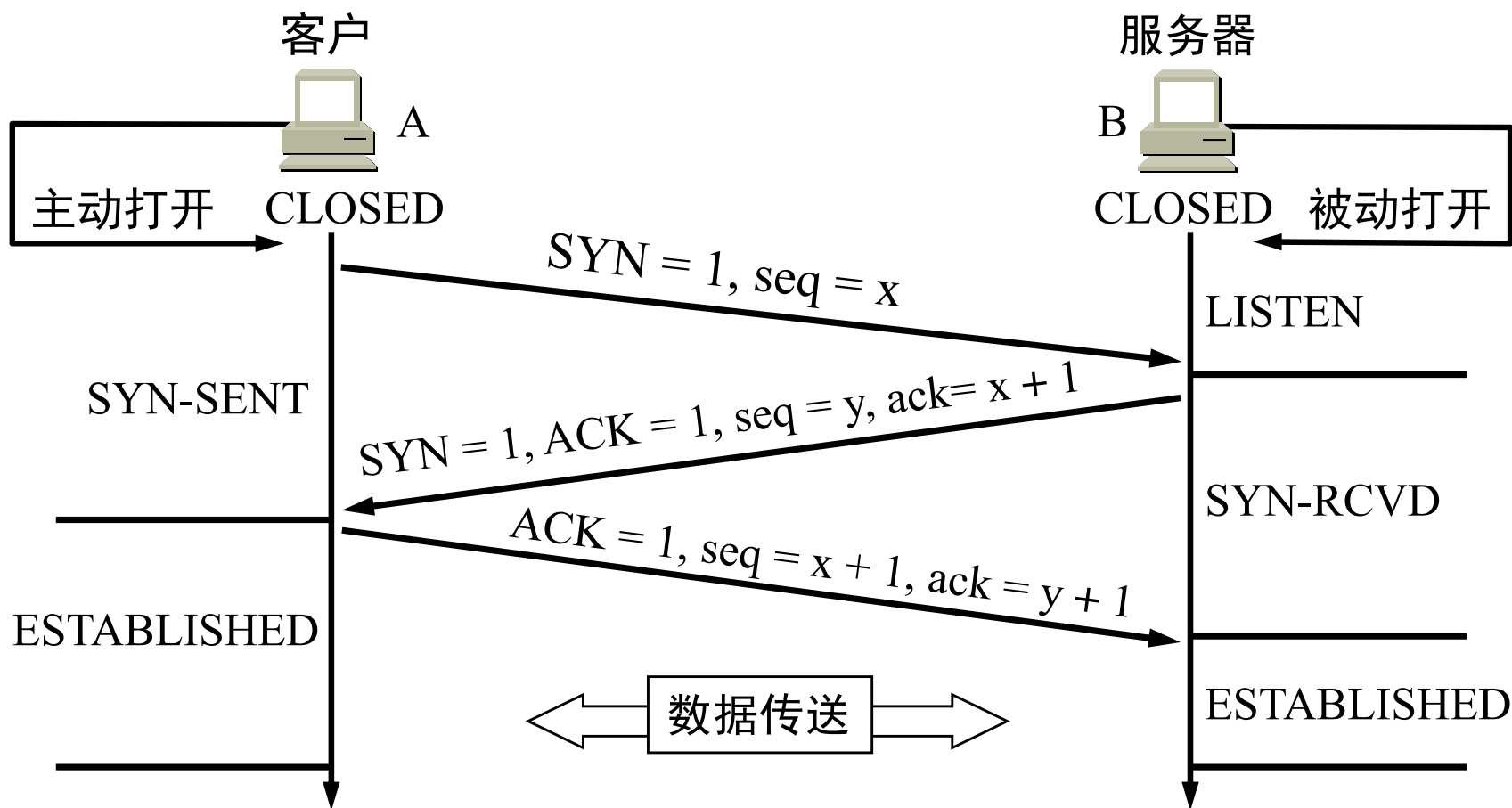
5.8 TCP的连接管理

一、简介

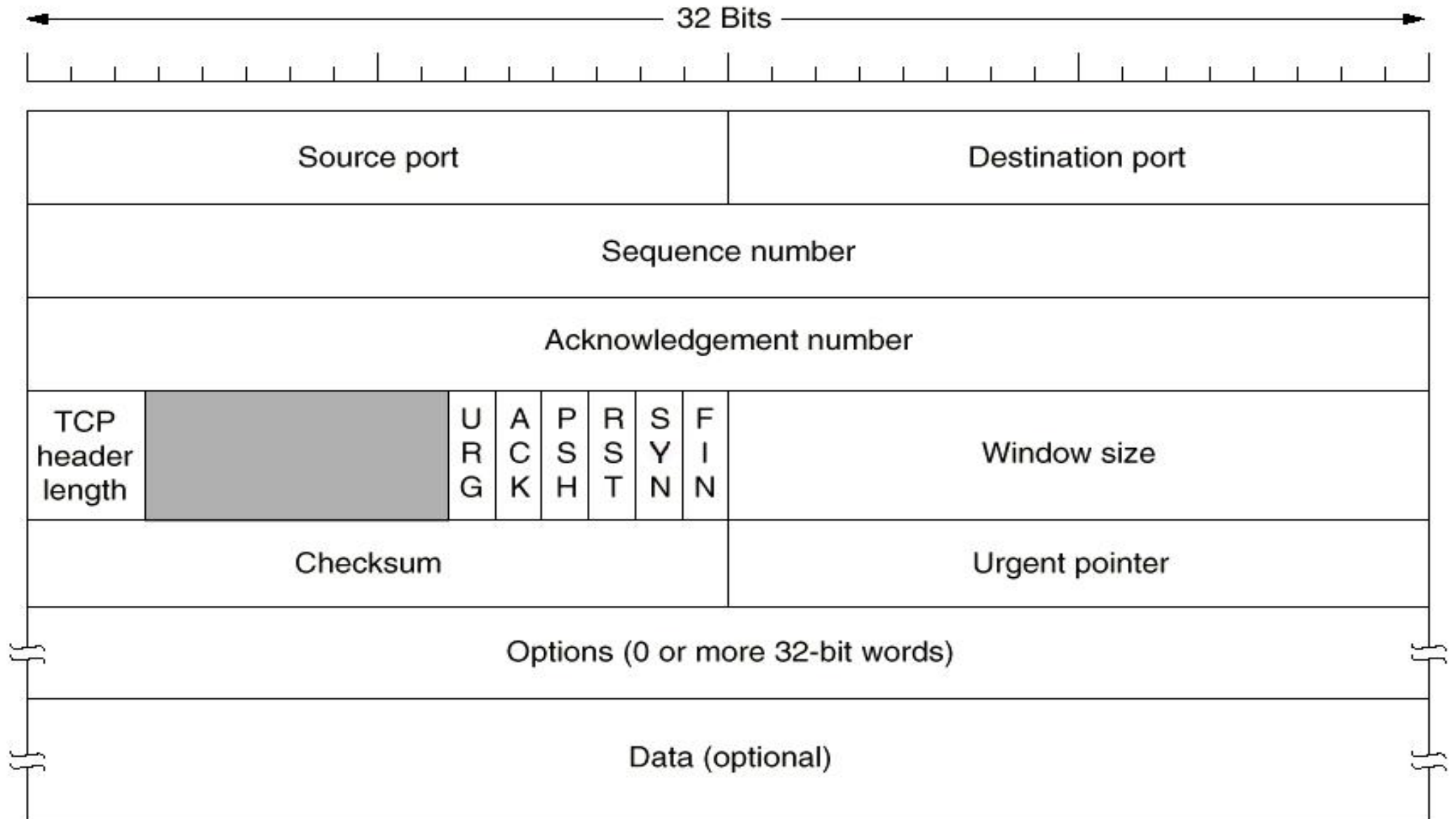
- TCP传输连接有三个阶段
 - ① 连接建立
 - ② 数据传送
 - ③ 连接释放
- 连接建立过程中要解决三个问题：
 - 使每一方能够确知对方的存在
 - 允许双方协商一些参数(如最大报文段长度、最大窗口大小、服务质量等)
 - 能够对传输实体资源进行分配(如缓存、连接表中的项目等)
- TCP 连接的建立都是采用**客户/服务器方式**
 - 客户(client): 主动发起连接建立的应用进程
 - 服务器(server): 被动等待连接建立的应用进程

二、TCP 的连接建立

- TCP连接的建立采用**三次握手(three-way handshake)**
 - ① A向B发出连接请求报文段，其首部中的**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
 - ② B收到连接请求后，如同意，则发回确认，其中**SYN=1**，**ACK=1**，确认号ack=x+1，自己选择的序号seq=y
 - ③ A收到后向B给出确认，其**ACK=1**，确认号ack=y+1
- B收到后，TCP连接建立



- 确认ACK: 1bit, 为1时, 确认号字段有效; 为0时, 确认号无效
- 同步SYN: 1bit, 为1时, 表示这是一个连接请求或连接接受报文
- 终止FIN: 1bit, 为1时, 表示要求释放TCP连接



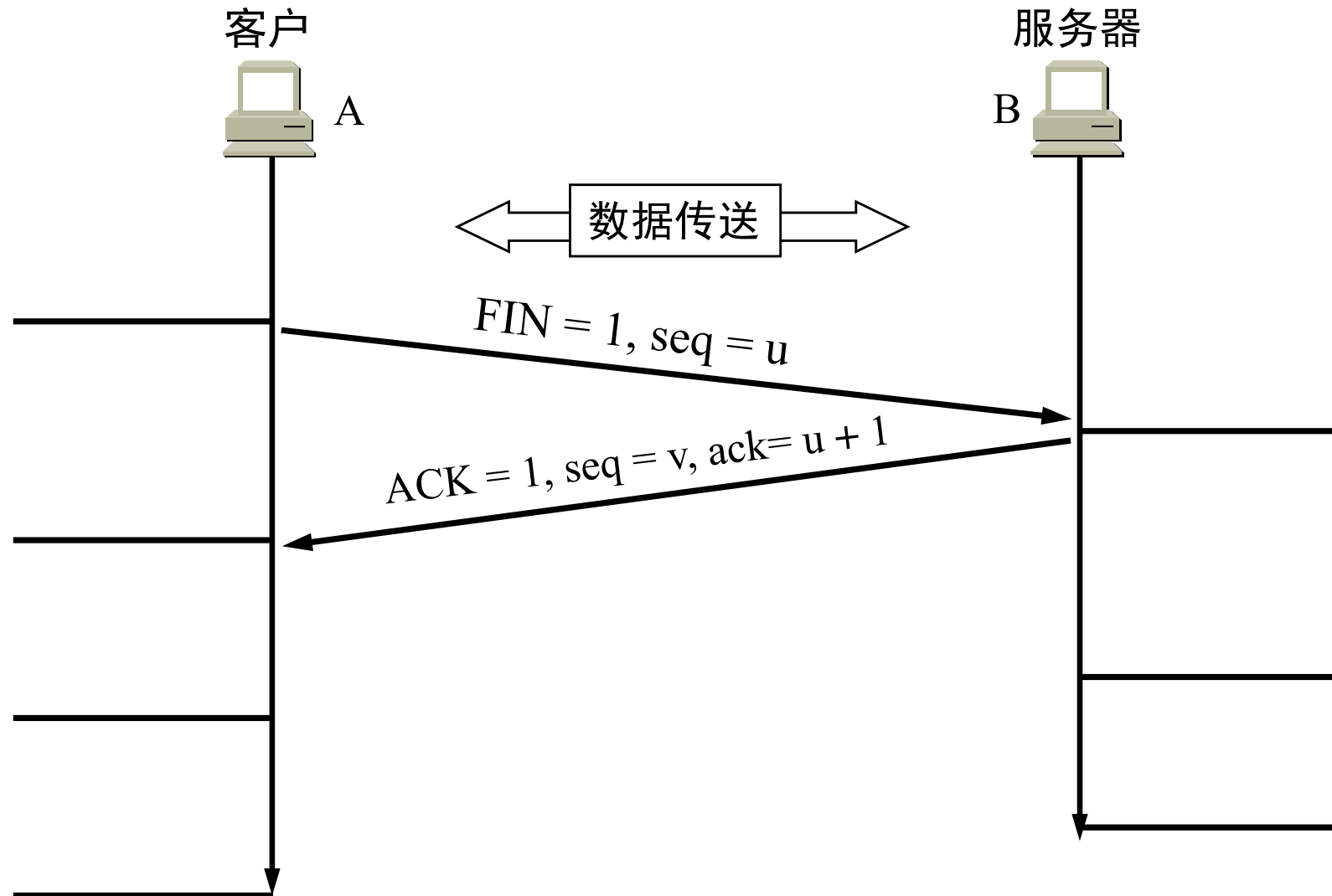
5.8 TCP的连接管理

二、TCP 的连接建立

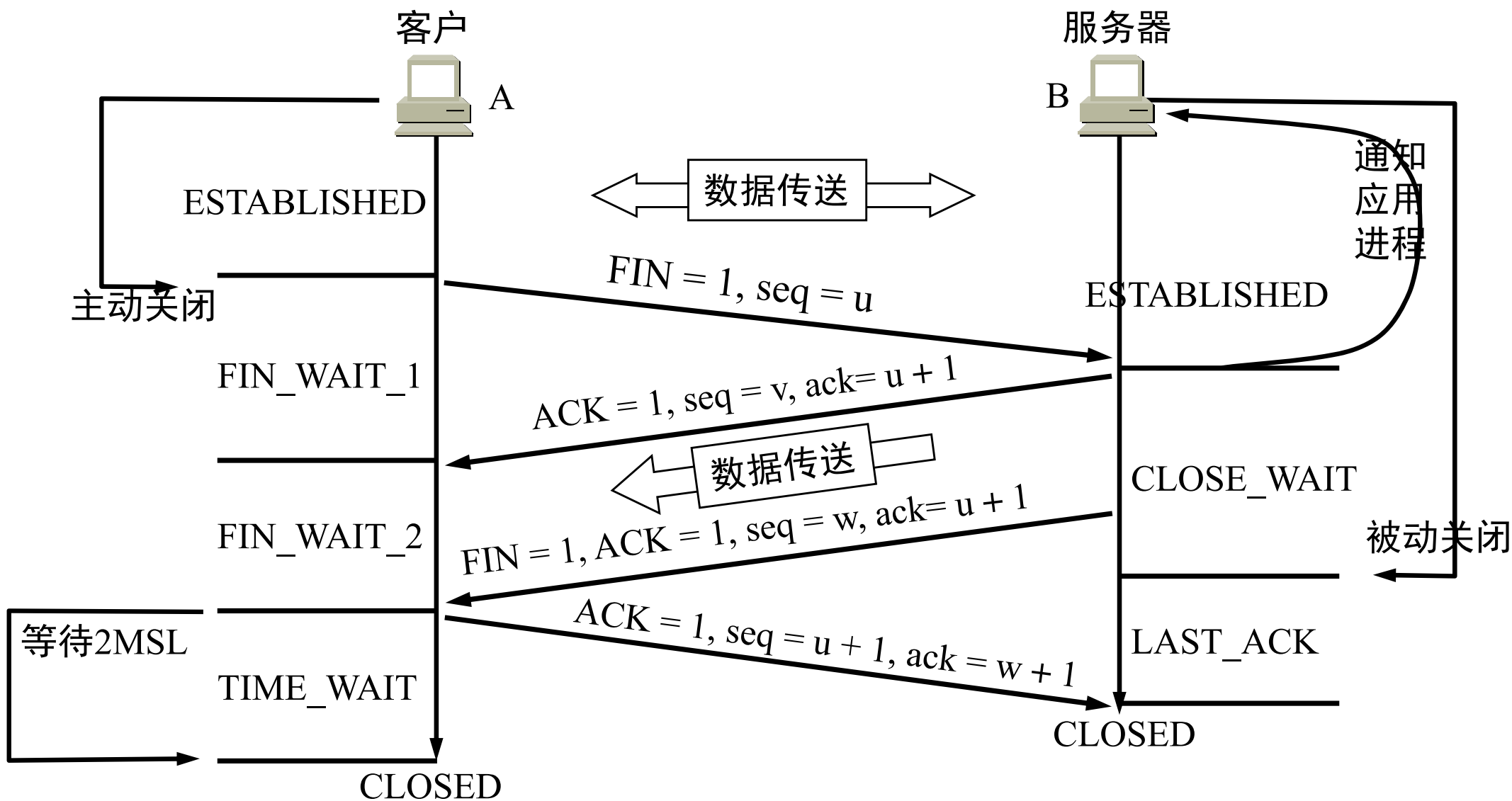
- 为什么采用3次握手而不是2次？
 - 防止“失效的连接请求”在服务器端占用资源
 - 客户端发出了连接请求，但该数据报在网络中某处滞留了
 - 客户端等待超时后，重发连接请求，服务器响应，建立连接
 - 滞留的连接请求又到达服务器端，如果采用2次握手，服务器将建立一个连接，分配资源(缓冲区、定时器、...) → 占用资源且长期存活
- 3次握手带来的安全问题：TCP SYN Flooding攻击
 - 攻击者连续发送大量SYN报文，但却不对SYN ACK报文做出响应
 - 结果：服务器内部数据结构满，无法响应正常用户的TCP连接请求
 - 此类攻击大多使用IP地址伪装，使得对攻击源的定位比较困难
 - Linux内核采用了SYN_Cookies机制应对这种攻击
基本思路：服务器返回SYN+ACK时，根据自身特有信息(时间戳、IP地址、端口号等)计算Cookies，作为seq返回给客户端，并在收到对方应答前，不为该连接分配数据结构

三、TCP的连接释放

- 连接的释放可由任一方发起
- (假定A主动关闭连接)A发送报文段，其首部的 $FIN=1$ ，序号 $seq=u$
- B发出确认， $ACK=1$ ，确认号 $ack=u+1$ ，序号 $seq = v$



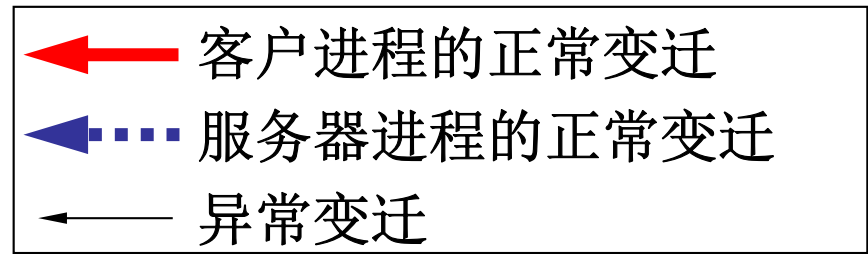
- $A \rightarrow B$ 方向的连接被释放，此时连接处于半关闭状态，B若发送数据，A仍要接收
- 若B已经没有要向A发送的数据，就向A发送连接释放报文($FIN=1, ACK=1$)
- A收到后，发出确认，其中 $ACK=1$ ，确认号 $ack=w+1$ ，序号 $seq=u+1$
- TCP连接必须经过时间**2MSL**后才真正释放掉



5.8 TCP的连接管理

三、TCP 的连接释放

- 关于MSL
 - **Maximum Segment Lifetime**, 报文段最大存活时间
 - **RFC793**中建议MSL=2分钟, 一般使用更小的值
- 关闭连接前等待2MSL时间的原因
 - 为了保证A发送的最后一个ACK报文段能够到达B
 - 防止“已失效的连接请求报文段”出现在本连接中
 - A在发送完最后一个ACK报文段后, 再经过时间2MSL, 就可以使本连接持续的时间内所产生的所有报文段, 都从网络中消失。这样就可以使下一个新的连接中不会出现这种旧的连接请求报文段



TCP 的有限状态机

(Finite State Machine)

