第七章 传输层基础

崔 勇

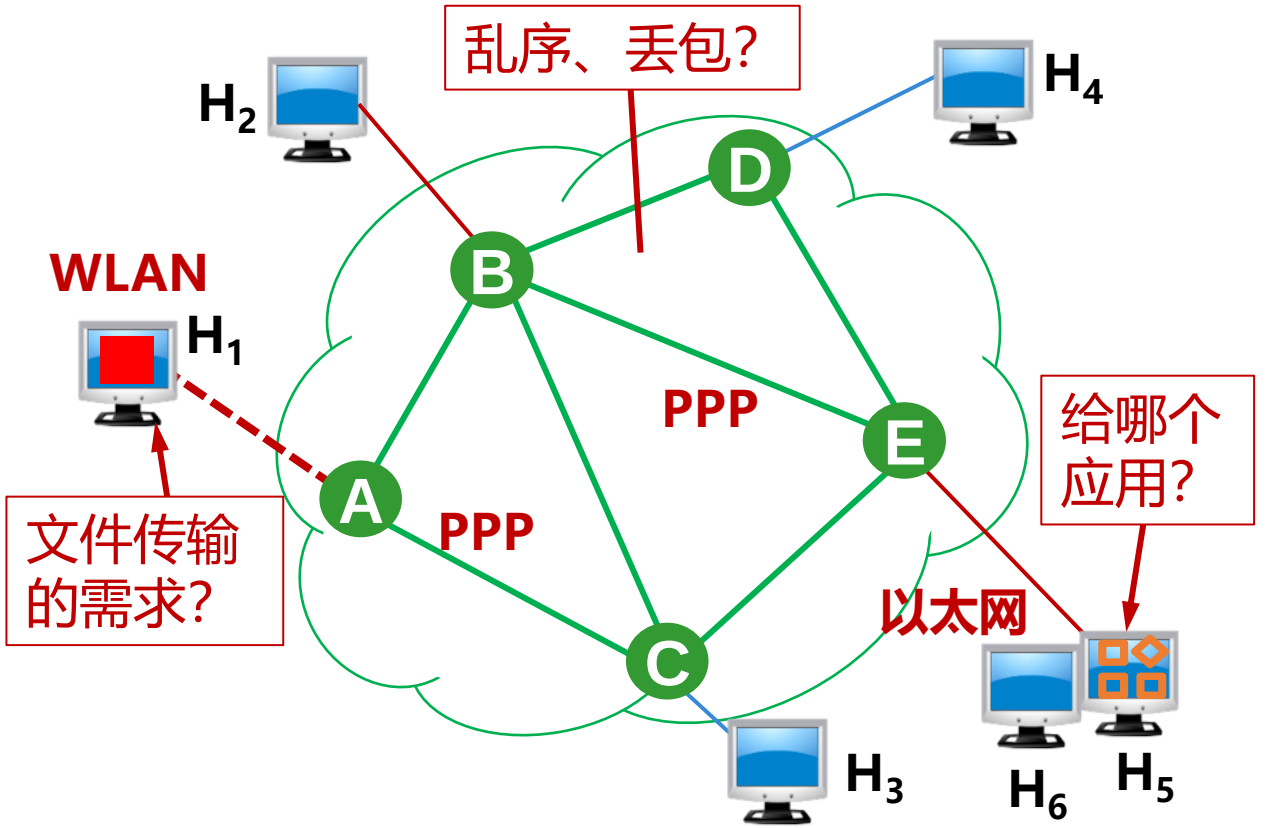
清华大学

计算机网络
教案社区

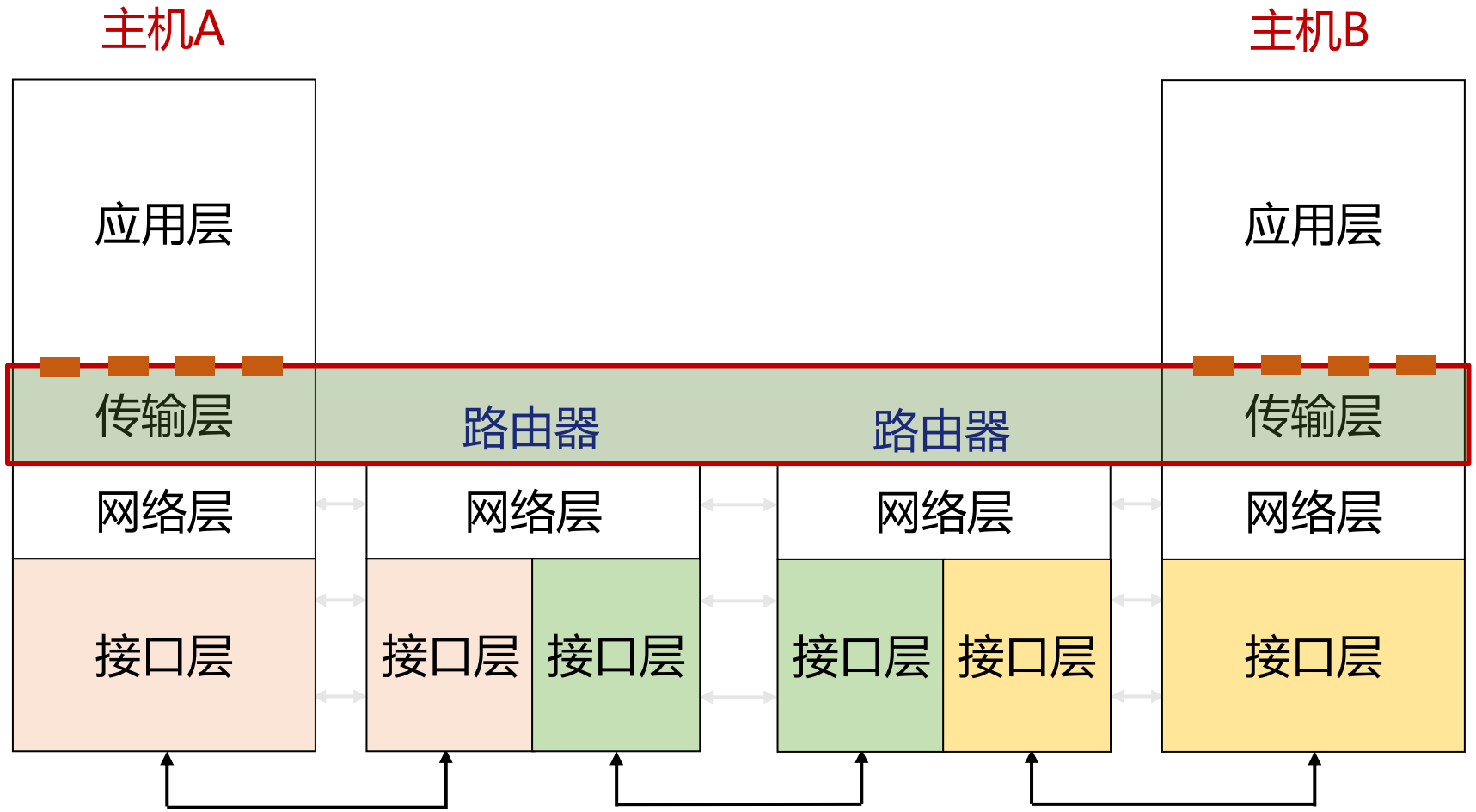
致谢社区成员	
人民大学 何军	清华大学 崔勇

思考与展望

- 网络层：多跳转发
  - 连通下层网络
    - WLAN、PPP、以太网……
  - 控制面：编址&路由
  - 数据面：转发
  - 协议：IP协议簇、路由协议
- 传输层需求分析
  - 如何知道数据属于哪个应用?
  - 向上层提供“更好”的服务?
  - 海量终端如何协同实现公平性?



传输层在哪里？



接口层通常包括数据链路层和物理层

本节目标

1. 了解传输层提供的服务
2. 掌握传输层协议的要素
   - 复用和分用、可靠数据传输
   - 流量控制、拥塞控制
3. 掌握拥塞控制原理
4. 掌握互联网传输层协议
   - UDP协议、TCP协议

需求

原理

设计

TCP/IP的沙漏模型

本节内容

传输层服务概述

- 引入传输层的原因
  - 从主机到主机->进程到进程
  - 消除网络层的不可靠性
  - 提供从源端主机到目的端主机的可靠的、与实际使用的网络无关的信息传输
- 传输层提供给上层的服务
  - 传输层提供两种服务
    - 简单便捷的无连接传输服务
    - 可靠的面向连接传输服务：连接建立，数据传输，连接释放
  - 重点：如何提供可靠的数据传输服务

传输层服务概述

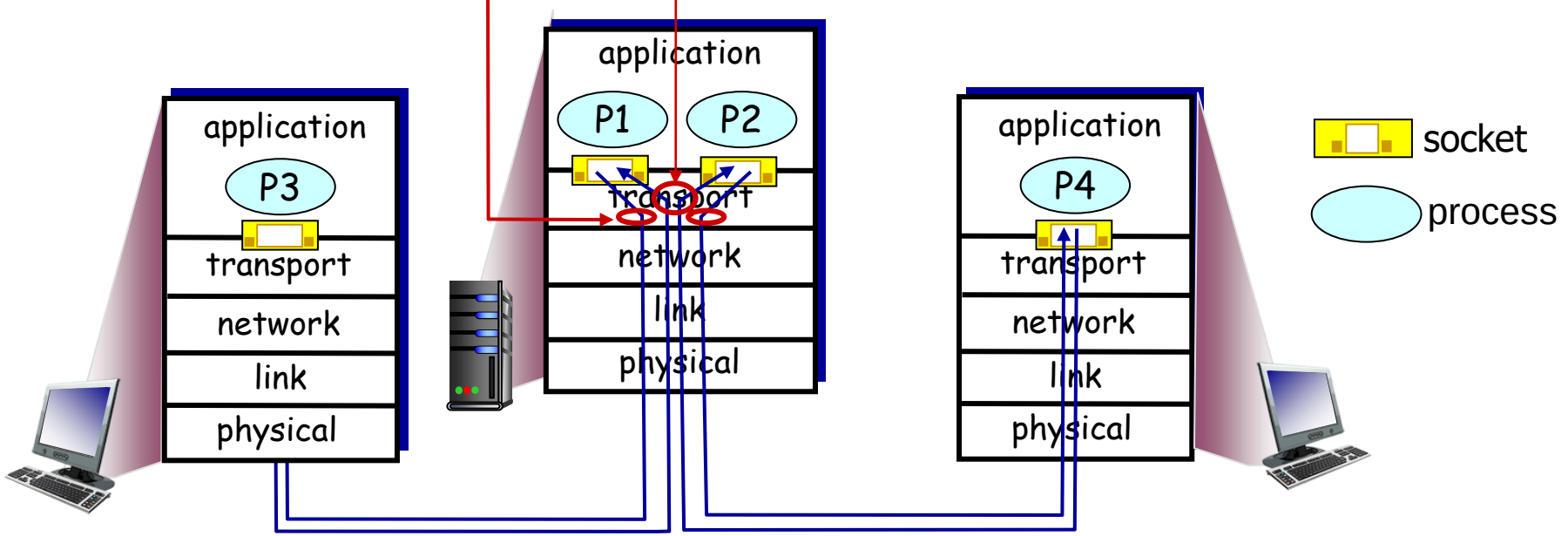
传输层基本服务：将主机间交付扩展到进程间交付，通过复用和分用实现

（发送端）复用:

传输层从多个应用收集数据，交给网络层发送

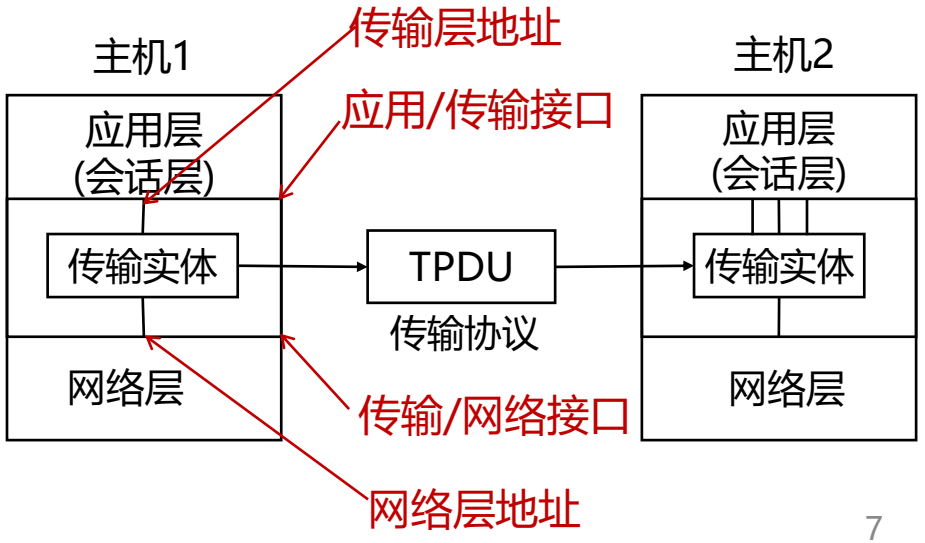
（接收端）分用:

传输层将从网络层收到的数据，交付给正确的应用



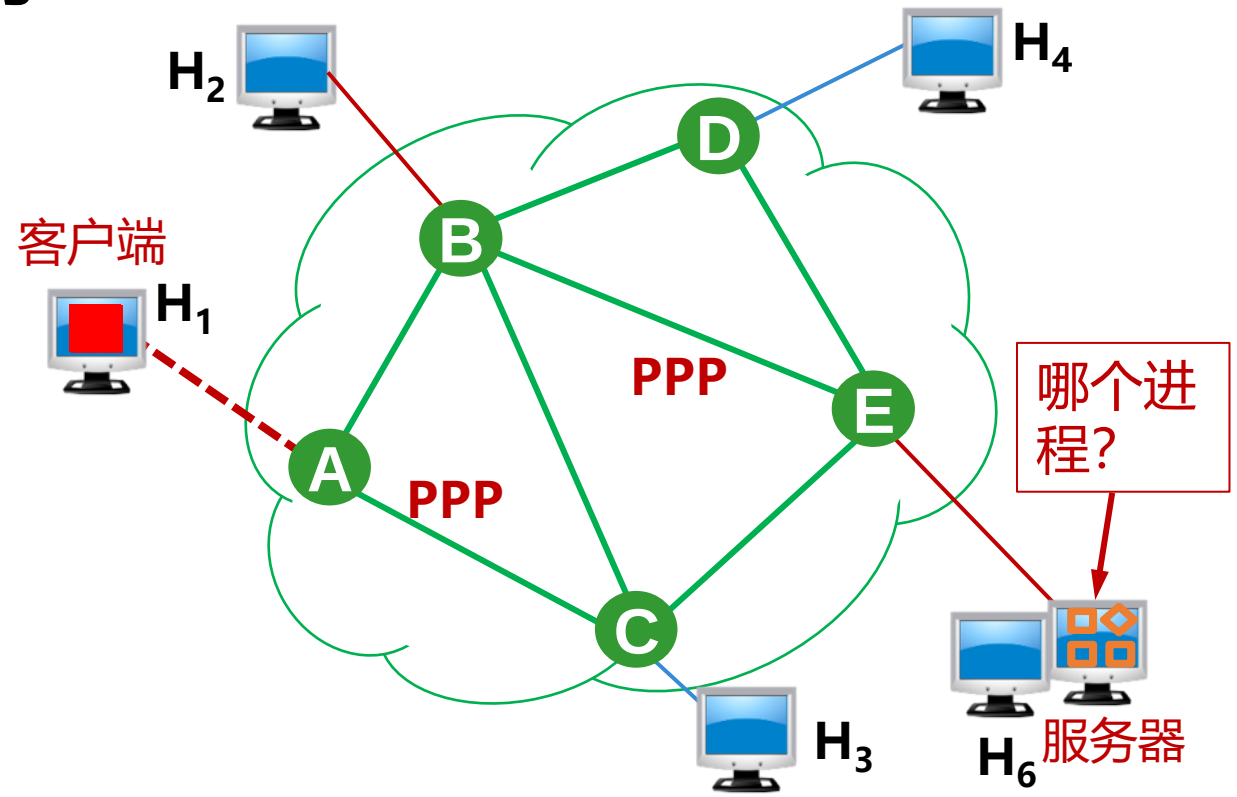
进程到进程: 进程如何标识自己

- 每个进程需要一个标识
  - 问题：可以用进程所在主机的地址来标识进程吗?
  - 回答：不能！因为同一个主机上可能运行着许多进程
- 端口号（port number）
  - 端口号被用来区分同一个主机上的不同进程
- 进程标识包括：
  - 主机地址，主机上与该进程关联的端口号



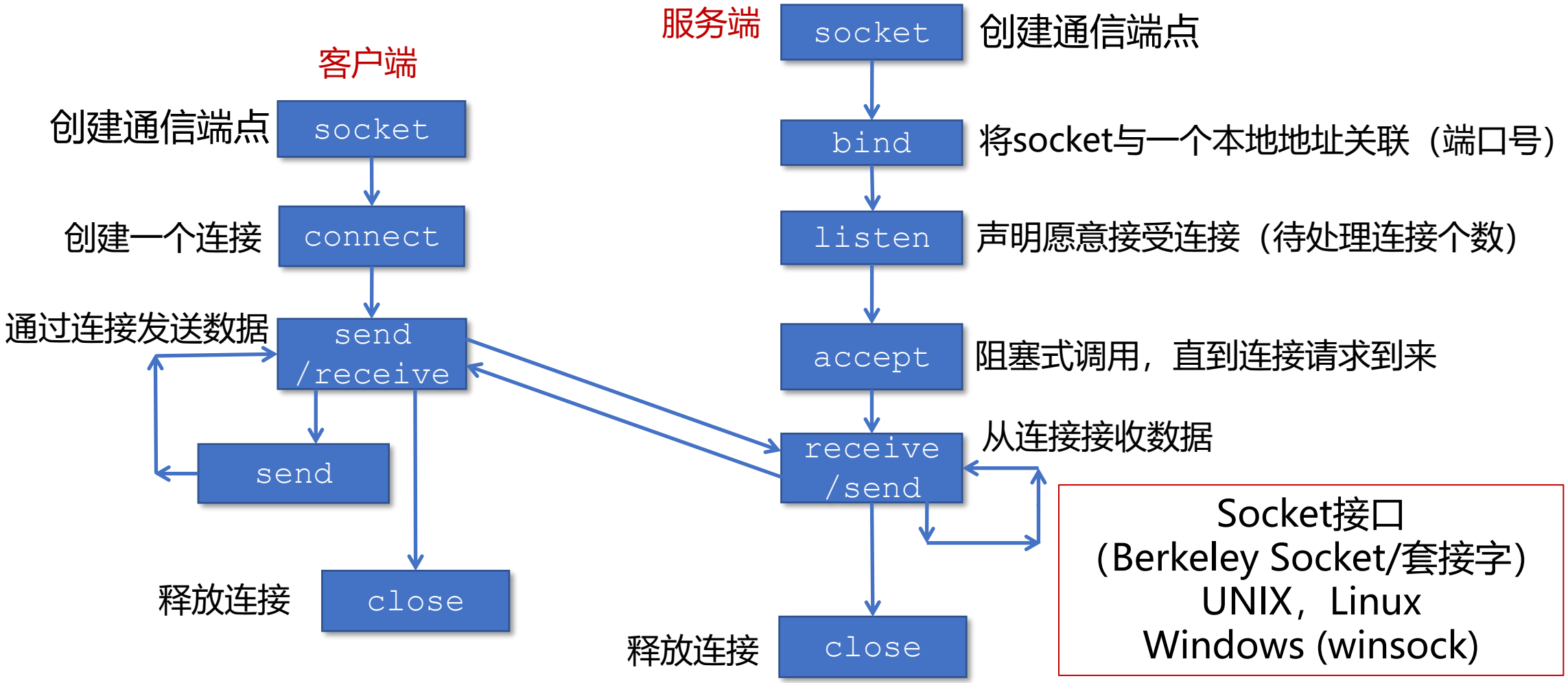
应用层需求分析-以TCP为例

- 传输层给应用层提供的C/S服务
  - 服务器的需求分析
    - 某进程希望使用网络
    - 该进程在什么地址收数据（端口）
    - 开始等待接受连接
    - 接收到连接请求并建立连接
    - 接收数据/发送数据
    - 完成任务，关闭连接，释放资源
  - 客户端的需求分析
    - 某进程希望使用网络
    - 请求连接服务器（地址）
    - 发送数据/接收数据
    - 完成任务，关闭连接，释放资源



传输层好比具有多个互不相通端口的线缆管道

Berkeley 套接字-TCP通信流程



本节内容

7.1 传输服务需求与设计

7.2 传输协议的要素

7.3 拥塞控制

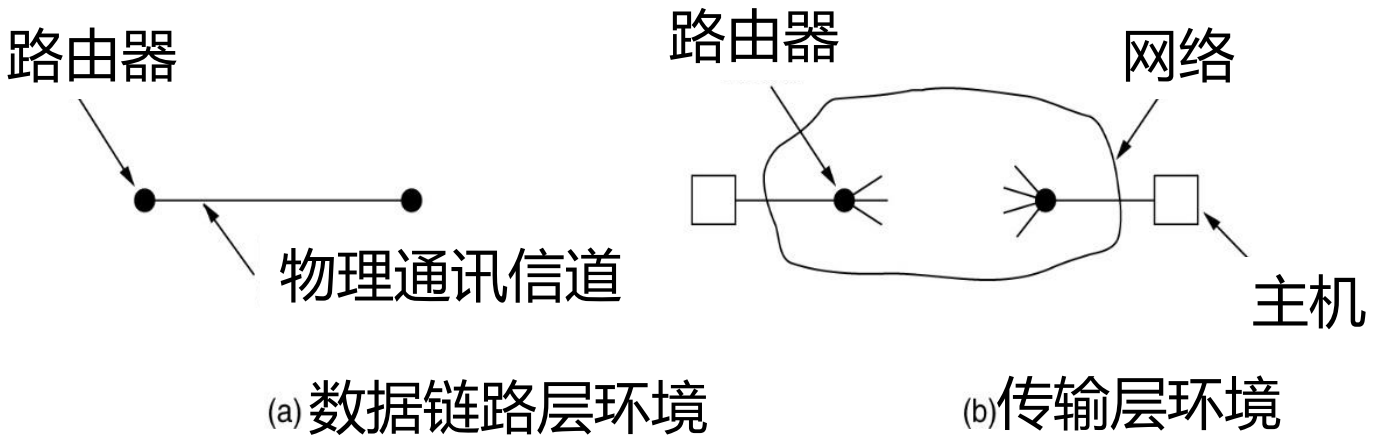
7.4 因特网传输协议UDP

7.5 因特网传输协议TCP

7.2.1 寻址
7.2.2 建立连接
7.2.3 释放连接
7.2.4 流量控制和缓冲策略
7.2.5 多路复用
7.2.6 崩溃恢复

传输协议的要素

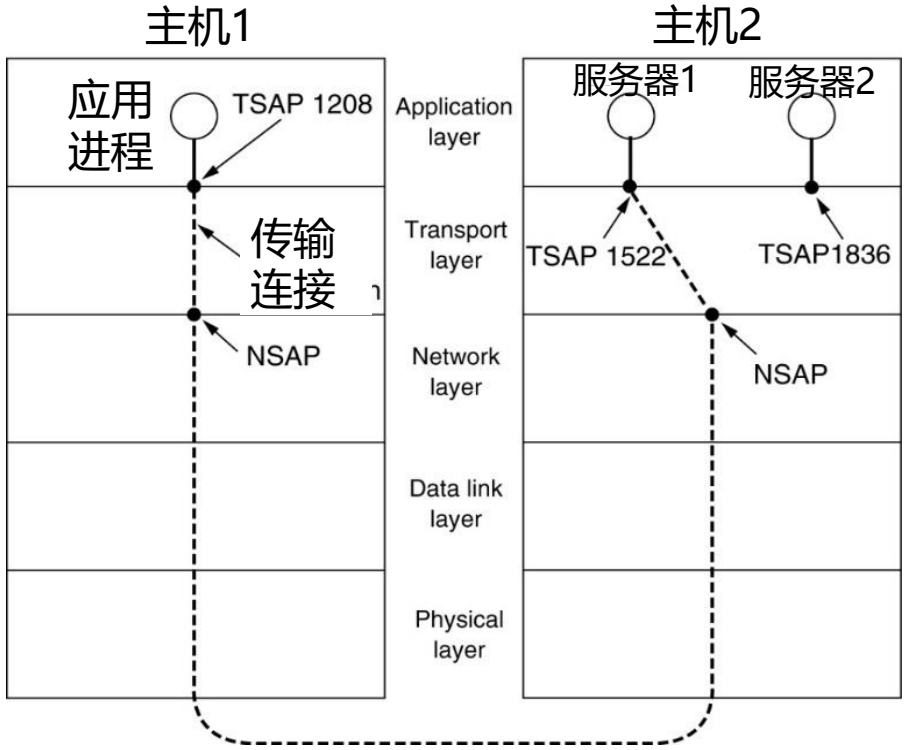
- 与数据链路层类似
  - 相同点：处理错误控制、顺序性、流控制
  - 不同点：指定目的地址、初始连接更为复杂、子网存储能力导致乱序接收、缓冲区数量可变
- 要素
  - 寻址
  - 建立连接
  - 释放连接
  - 差错控制与流量控制
  - 多路复用



(a) 数据链路层环境　　(b)传输层环境

寻址

- 传输服务访问点TSAP
- 网络服务访问点NSAP



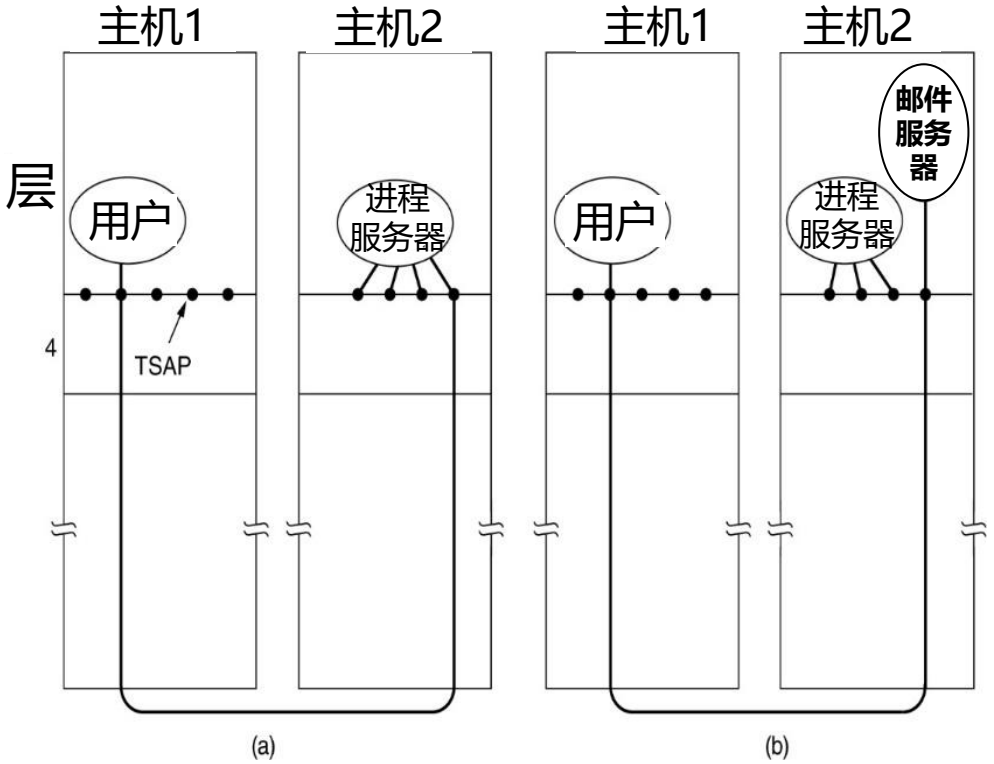
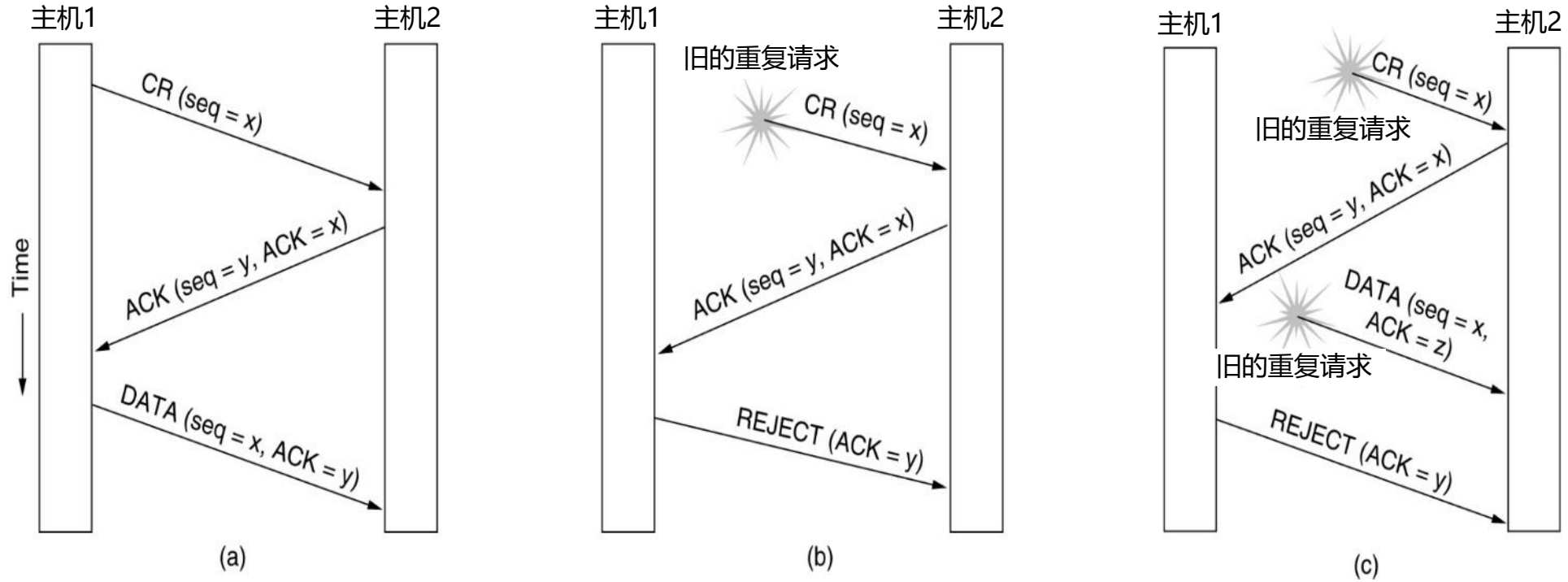


Fig6-9 主机1上的进程通过进程服务器与主机2上的被请求服务器建立连接

建立连接

➢三次握手—建连报文丢失怎么办?

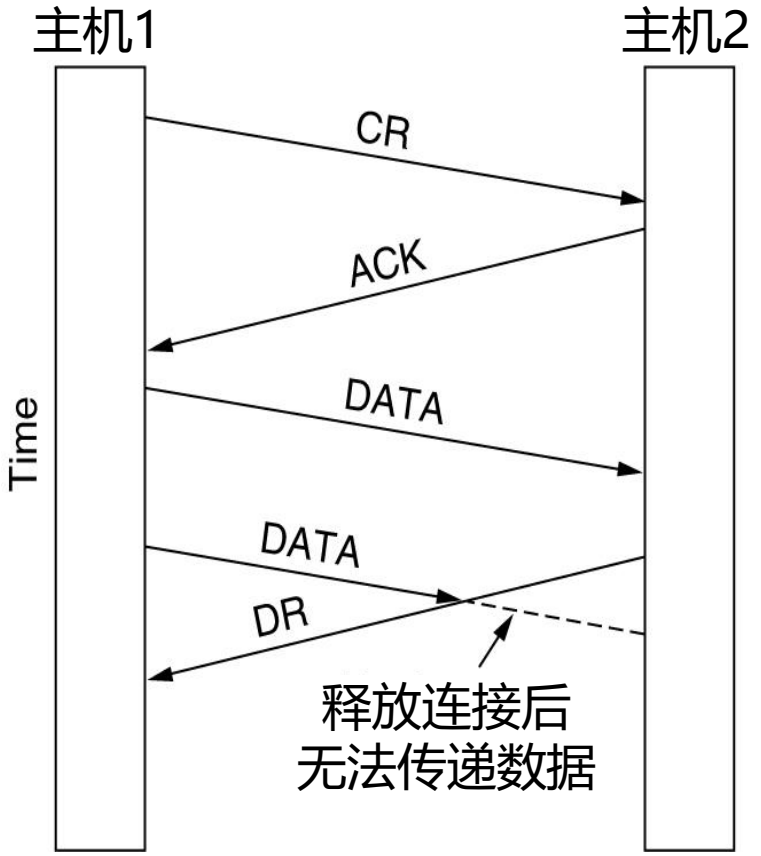
- 时间戳：作为辅助的32位扩展序号：2^32>137×365×24×60×60



释放连接

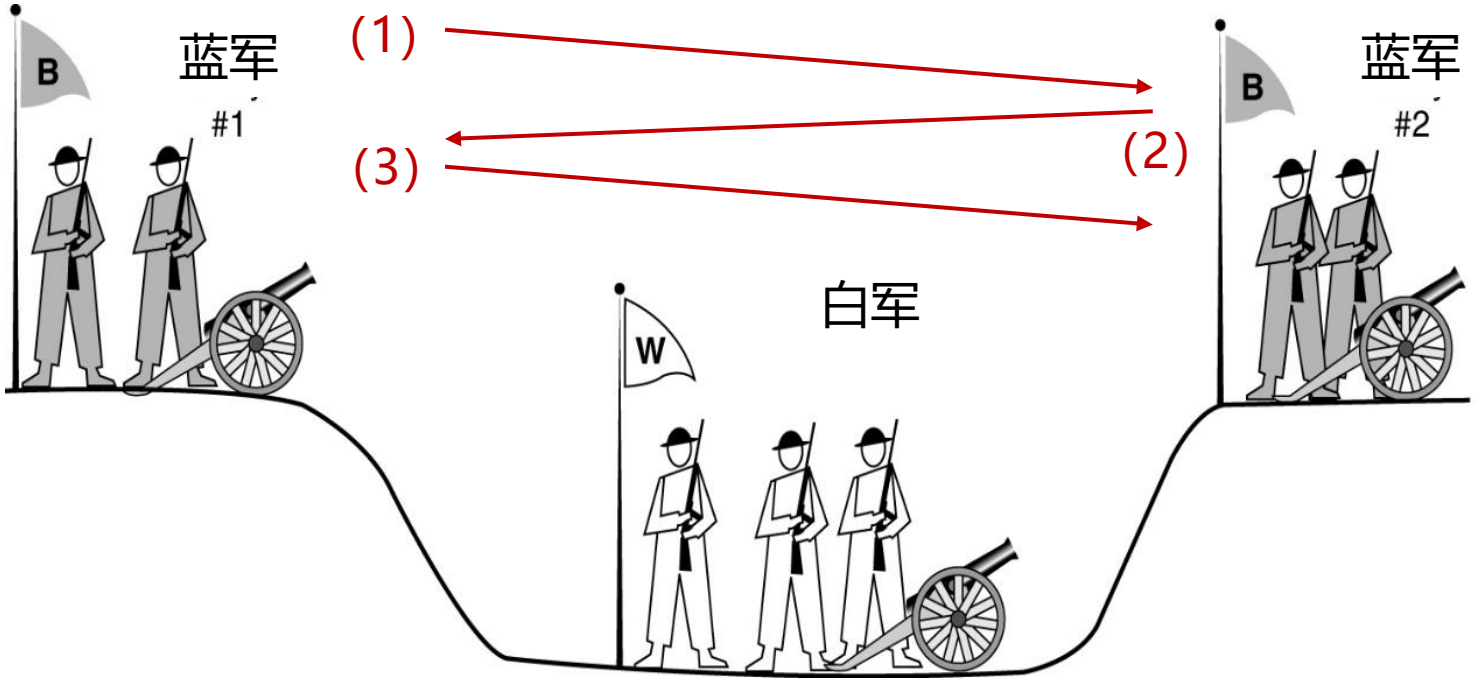
- 释放连接
  - 非对称释放
    - 任何一方可DISCONNECT
    - 右图导致数据丢失
  - 对称释放
    - 各方单独DISCONNECT，停止发送但继续接收
    - 双方均DISCONNECT后释放连接

断连可靠吗?



释放连接(2)

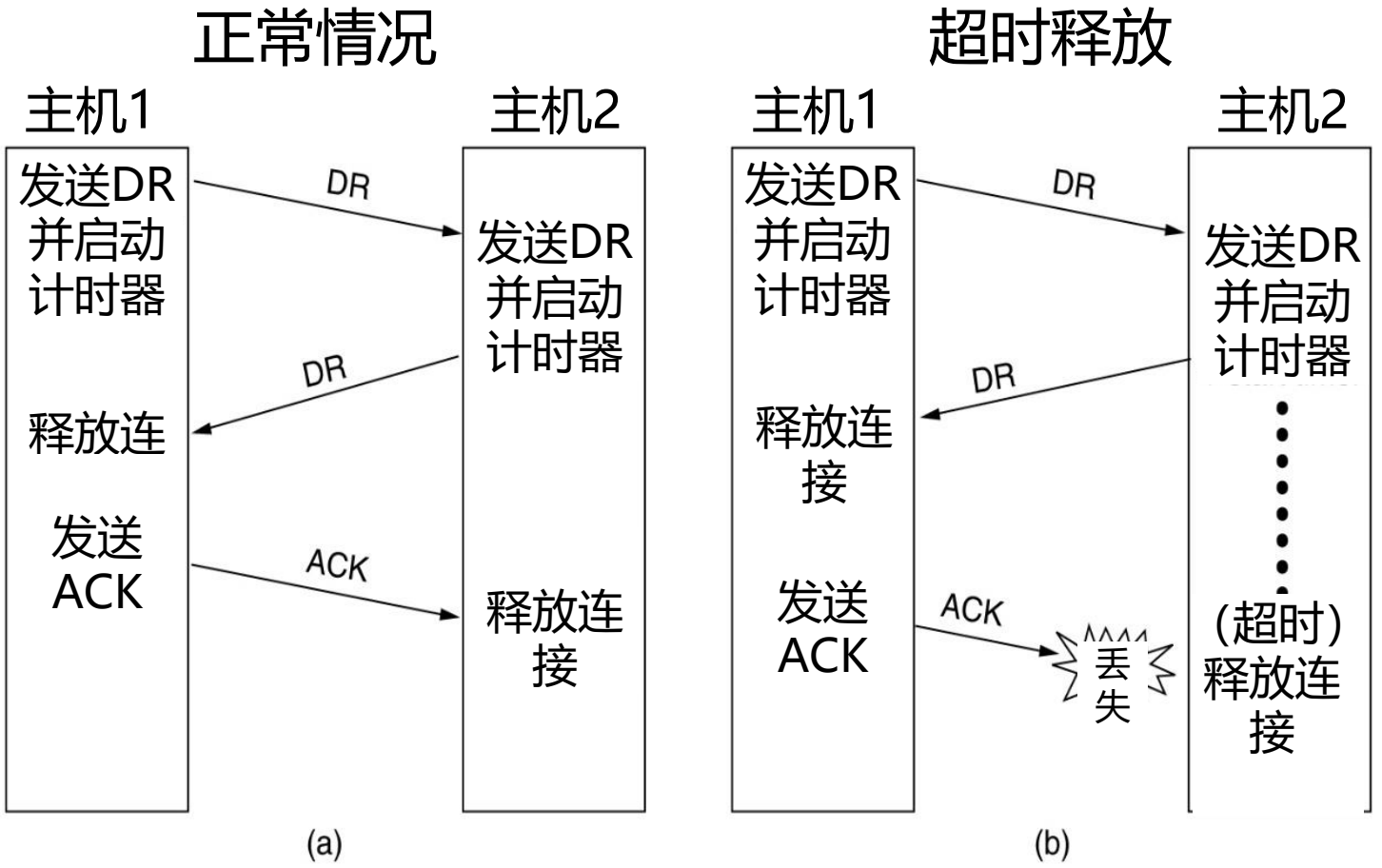
➢两军作战问题



- 谁来确认谁?
- 透过现象看本质：最后一次确认有价值吗?

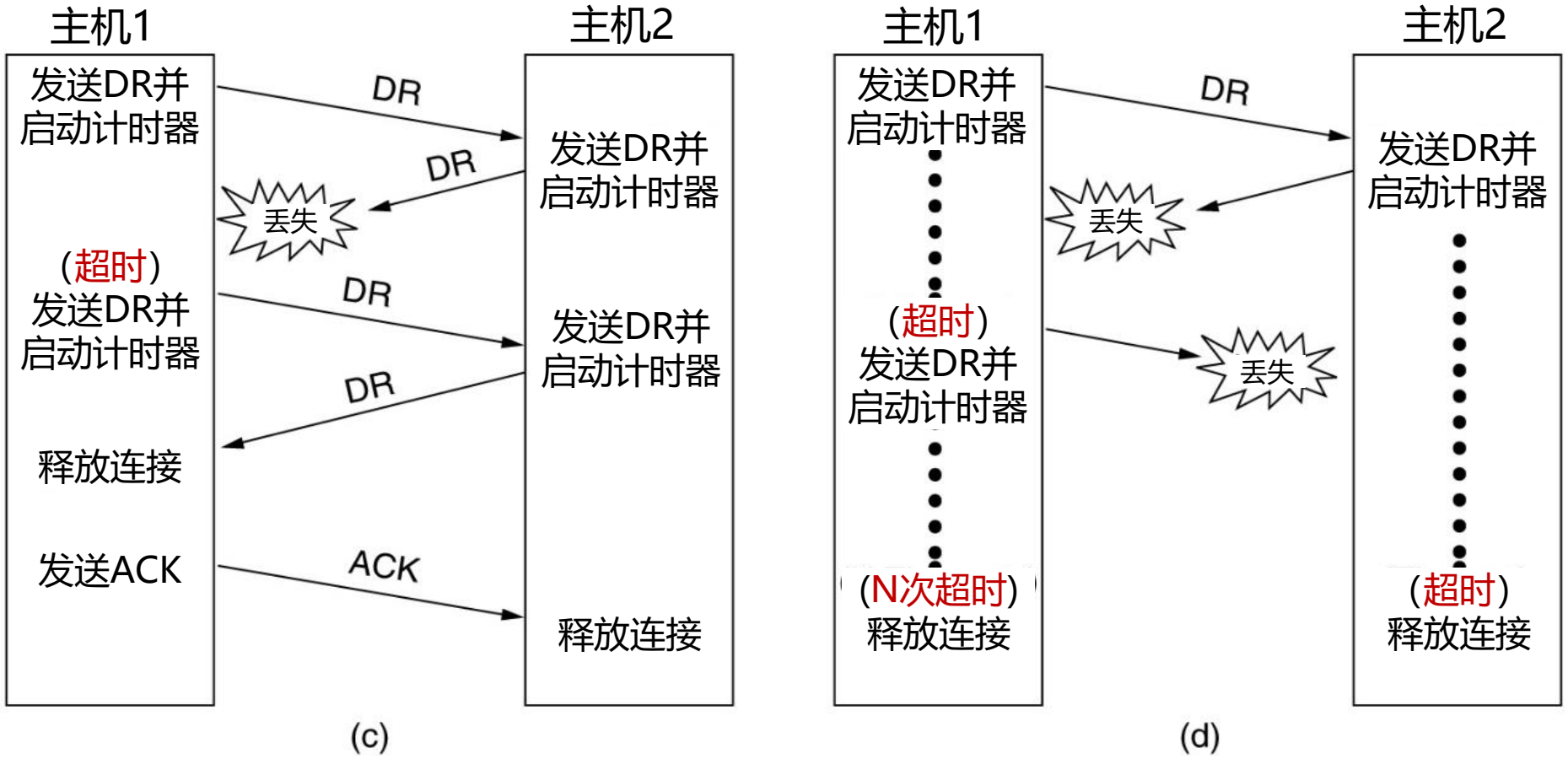
释放连接(3)

- 释放连接的场景



释放连接(4)

- 释放连接的场景



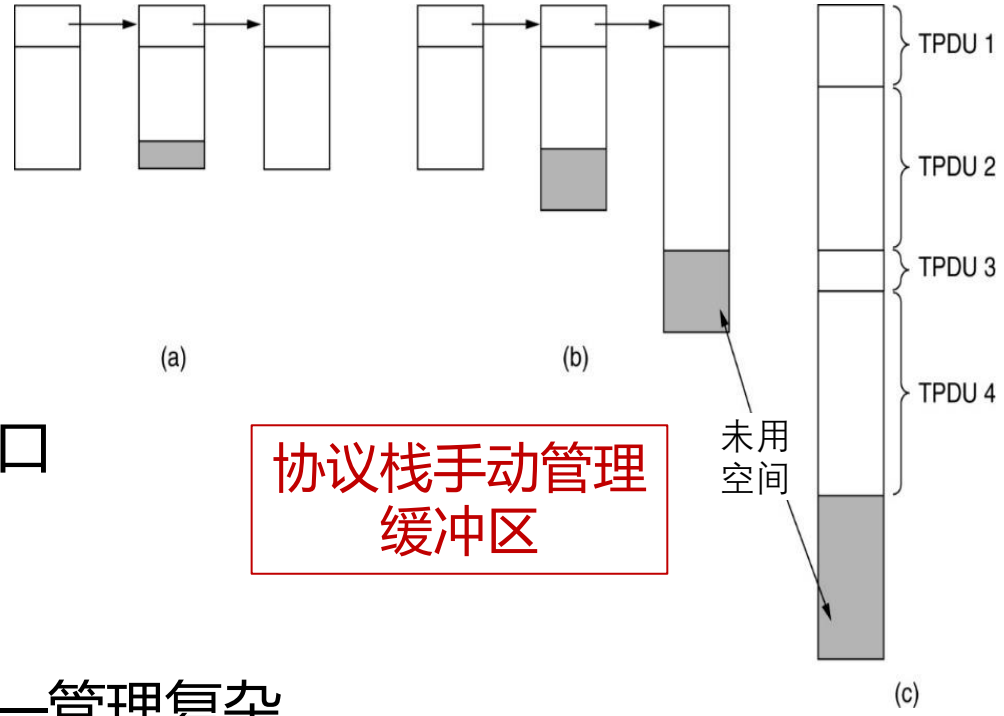
差错控制与流量控制

- 与数据链路层协议不同
  - 链路层校验只保护物理链路，路由器错误在链路层校验之外
  - 端到端的传输层校验至关重要
  - 链路层：时延带宽积小，有线链路误码低
  - 传输层：时延带宽积较大，需使用较大滑动窗口
- 为差错控制而建立的缓冲区

  a. 段的长度都差不多—链接固定大小的缓冲区

  b. 段的长度差异较大— 链接可变大小的缓冲区—管理复杂

  c. 每条连接使用一个大的循环缓冲区



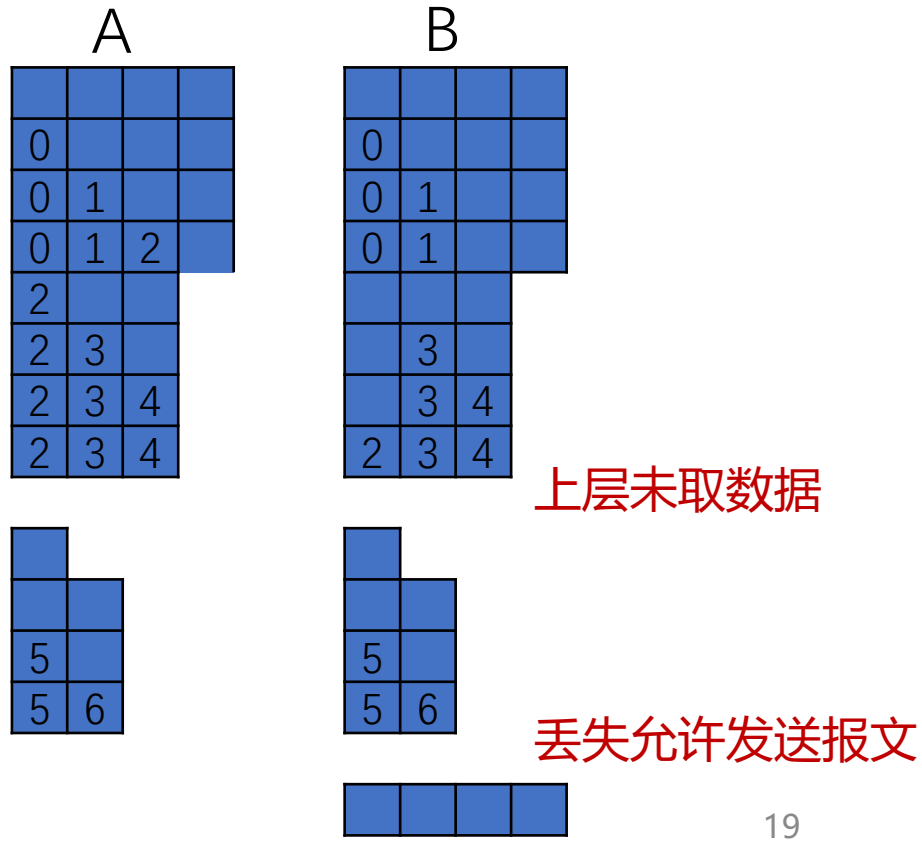
协议栈手动管理缓冲区

差错控制与流量控制

➢动态管理窗口：传输层利用可变滑动窗口协议来实现流控

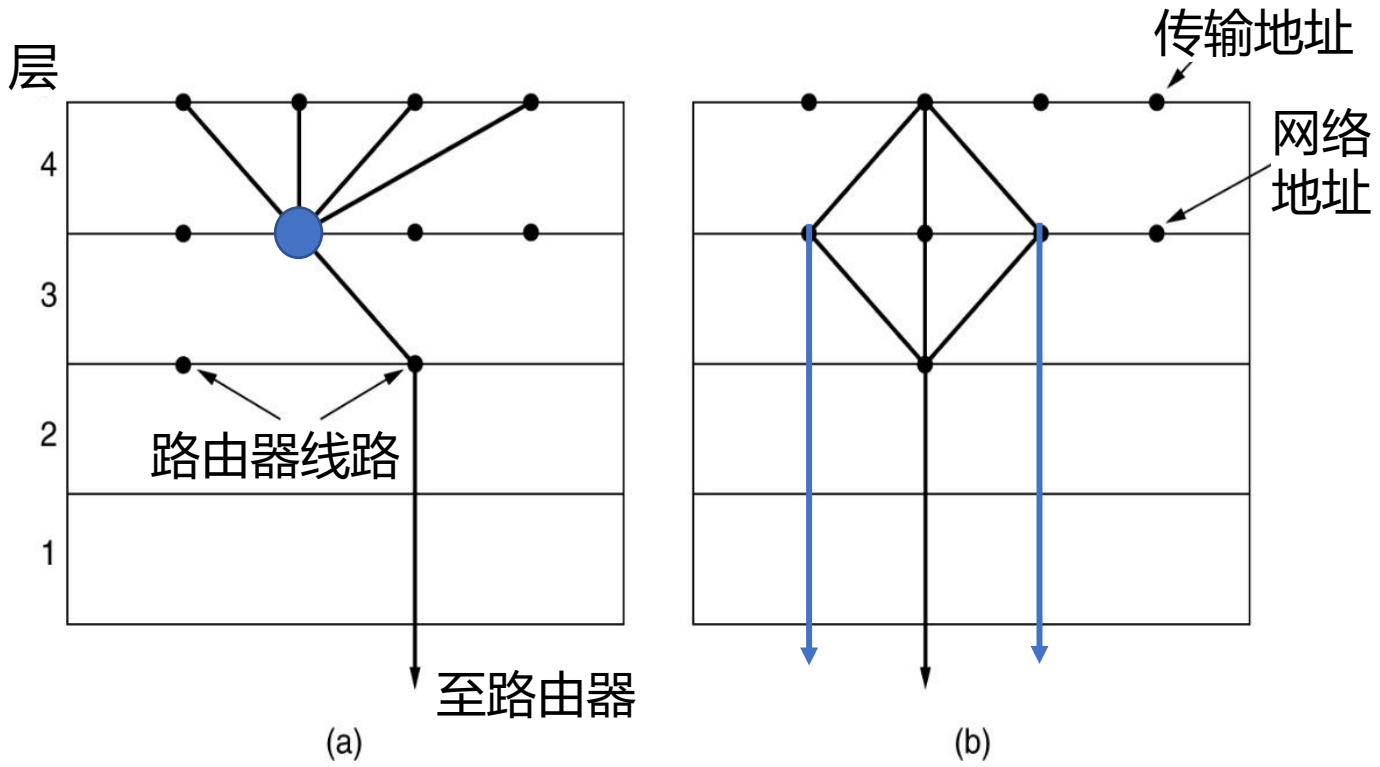
- 第16步丢失导致死锁；周期性发送控制报文

	主机A	消息	主机B	注释
1	→	< request 8 buffers>	→	A需要8个缓冲区
2	←	<ack = 15, buf = 4>	←	B只能提供4个
3	→	<seq = 0, data = m0>	→	A还有3个缓冲区可用
4	→	<seq = 1, data = m1>	→	A还剩2个缓冲区
5	→	<seq = 2, data = m2>	•••	消息丢失，A认为还剩1个缓冲区
6	←	<ack = 1, buf = 3>	←	B确认0和1，允许2-4
7	→	<seq = 3, data = m3>	→	A还剩1个缓冲区
8	→	<seq = 4, data = m4>	→	A还剩0个缓冲区，必须停止
9	→	<seq = 2, data = m2>	→	A超时并重传
10	←	<ack = 4, buf = 0>	←	全部确认，但A仍然阻塞
11	←	<ack = 4, buf = 1>	←	A可以发送m5了
12	←	<ack = 4, buf = 2>	←	B又找到一个新的缓冲区
13	→	<seq = 5, data = m5>	→	A还剩1个缓冲区
14	→	<seq = 6, data = m6>	→	A被再次阻塞
15	←	<ack = 6, buf = 0>	←	A仍然阻塞
16	•••	<ack = 6, buf = 4>	←	潜在的死锁



多路复用

- 向上多路复用：多个传输连接使用同一网络连接
- 向下多路复用：一个传输连接使用多个网络连接



传输协议的要素-小结

- 寻址
  - 通过传输服务访问点TSAP，解决消息发送给谁的问题
  - Internet中，TSAP为：IP地址+协议+Port
- 建立连接
  - 想方设法避免各种错误：多次握手
- 释放连接
  - 非对称释放：太冒失，可能导致数据的丢失
  - 对称释放：多次握手
- 差错控制与流量控制
  - 传输层进行端到端的校验（链路层只保护穿过单跳链路的帧）
  - 带宽时延积比链路层大得多，需要更大的缓存
- 多路复用

本节内容

7.1 传输服务需求与设计

7.2 传输协议的要素

7.3 拥塞控制

7.4 因特网传输协议UDP

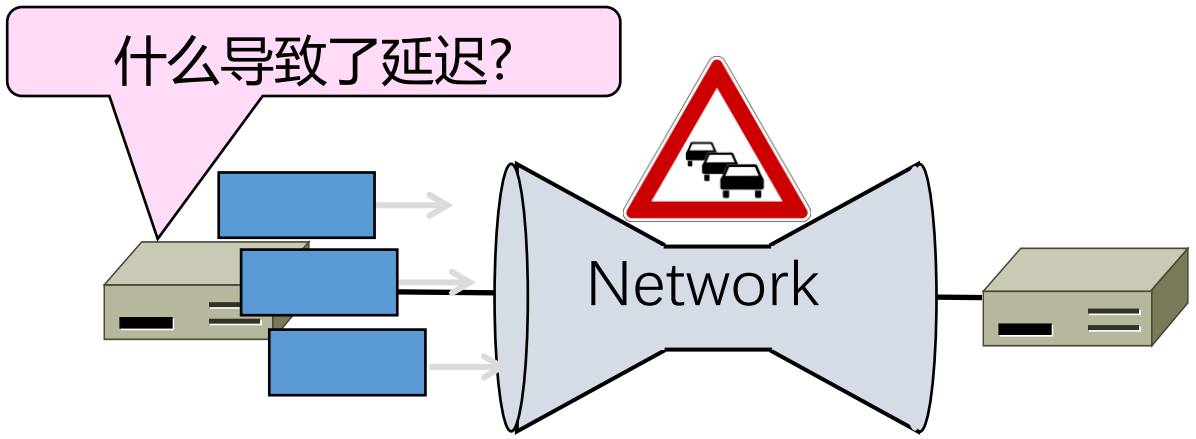
7.5 因特网传输协议TCP

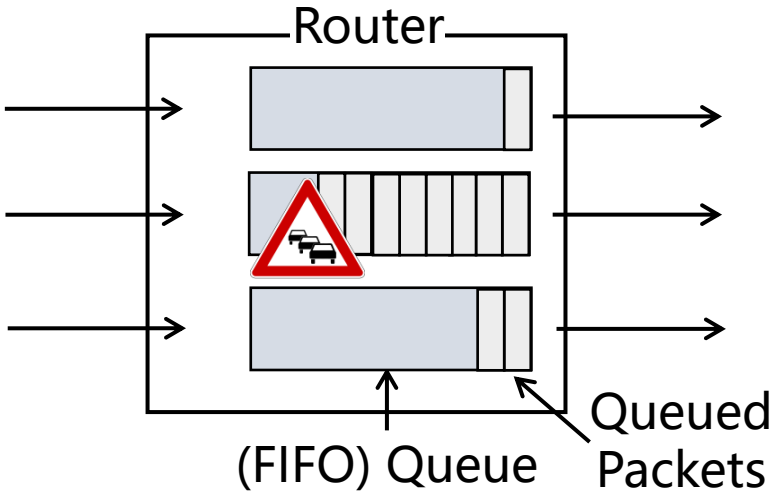
7.3.1 拥塞的影响
7.3.2 调整发送速率
7.3.3 无线问题

拥塞的影响

- 拥塞
  - 网络中的交通阻塞（traffic jam）
  - 路由器/交换机具有用于争用的内部缓冲
- 每个端口输出队列的简化视图
  - 通常为FIFO（先进先出），装满后将其丢弃
  - 当 短暂发生 输入> 输出 速率时，利用队列吸收突发到来的分组
  - 但是如果 输入>输出 速率持续存在，队列将溢出，即发生拥塞

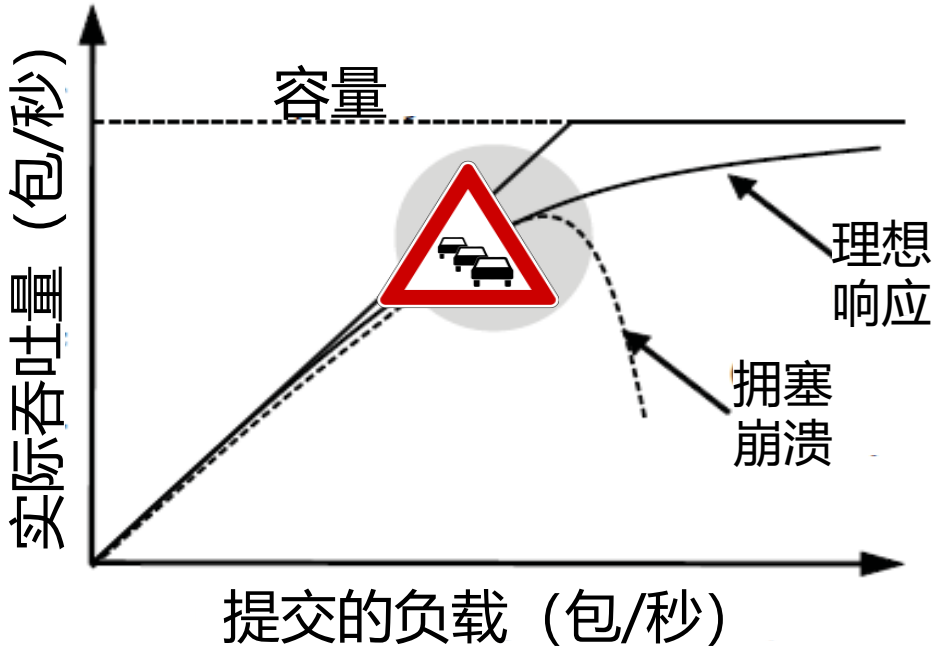
微突发?
缓存好吗?



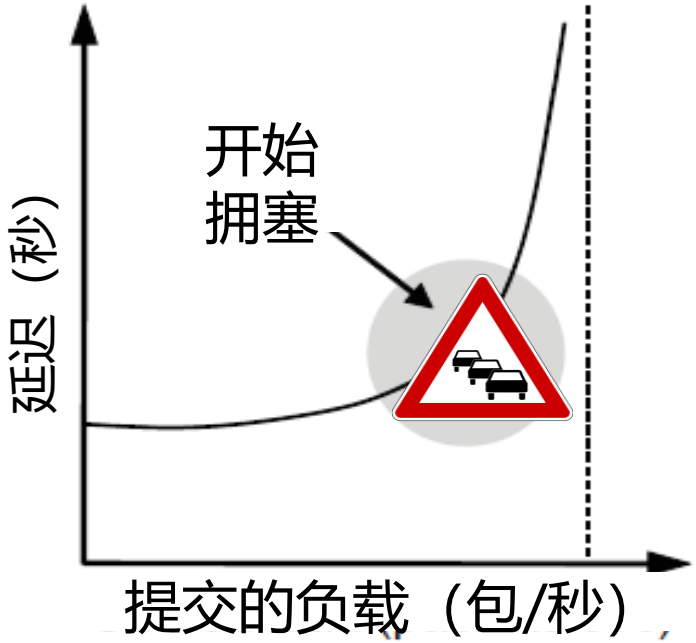


拥塞的影响

- 随着负载的增加，性能会发生什么变化？
  - 有效吞吐量Goodput可能低于吞吐量（由于虚假的重新传输）
- 要避免出现上述情况：在拥塞开始之前就操作网络



有效吞吐量Goodput低于load（由于lost）



delay和lost急剧上升

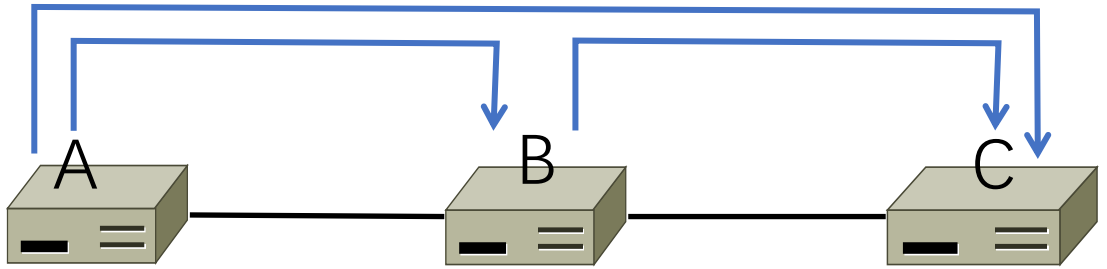
带宽分配

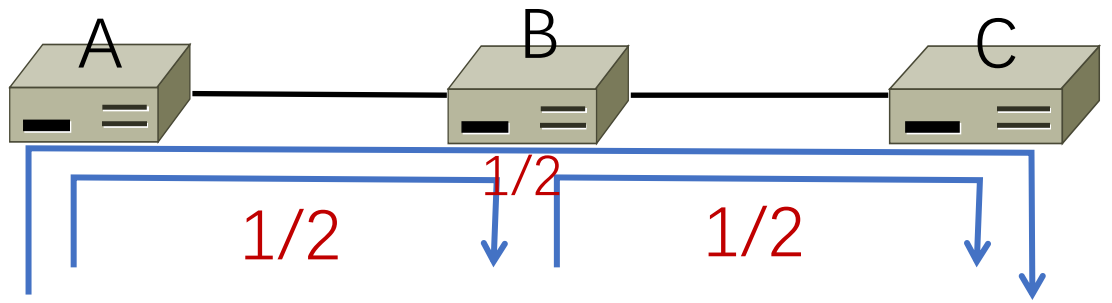
- 网络的重要任务是：将其容量公平分配给大量发送方
  - Good allocation is efficient and fair
  - Efficient 意味着使用了最多的容量，但没有拥塞
  - Fair 意味着每个发送方都能合理共享网络
- 网络层见证拥塞
  - 只有它可以提供直接反馈
- 传输层导致拥塞
  - 只有它可以减少提供的负载
- 关键思路
  - 端网协同且不断调整

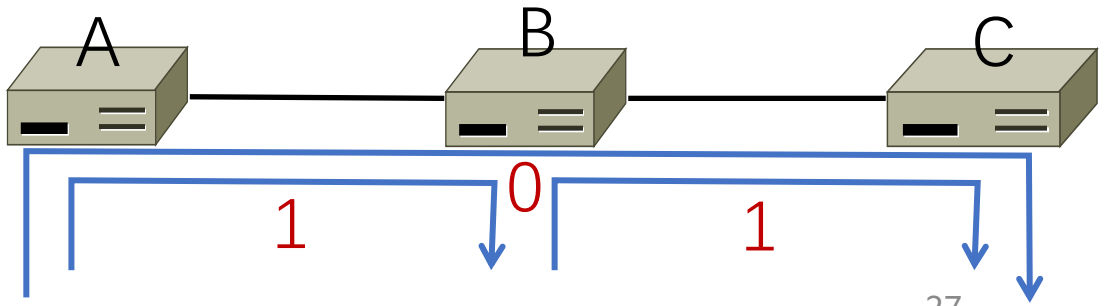
网络是分布式、动态的
没有任何一方可以全面了解其状态

Efficiency vs. Fairness

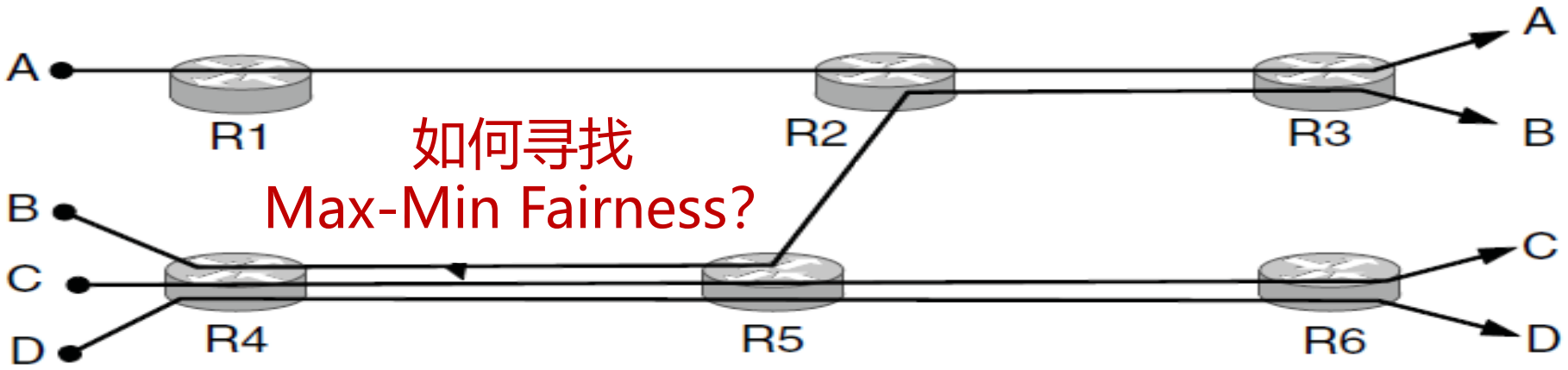
- 效率公平难以兼有!
  - 示例中的网络：A→B, B→C , A→C
  - 可以运载多少流量?
- 如果关心公平性
  - 为每个流分配相等的带宽
  - A→B: ½ unit, B→C: ½, A→C: ½
  - 承载的总流量为 1 ½ 单位
- 如果关心效率
  - 最大化网络中的总流量
  - A→B: 1 unit, B→C: 1, A→C: 0
  - 总流量上升到2个单位







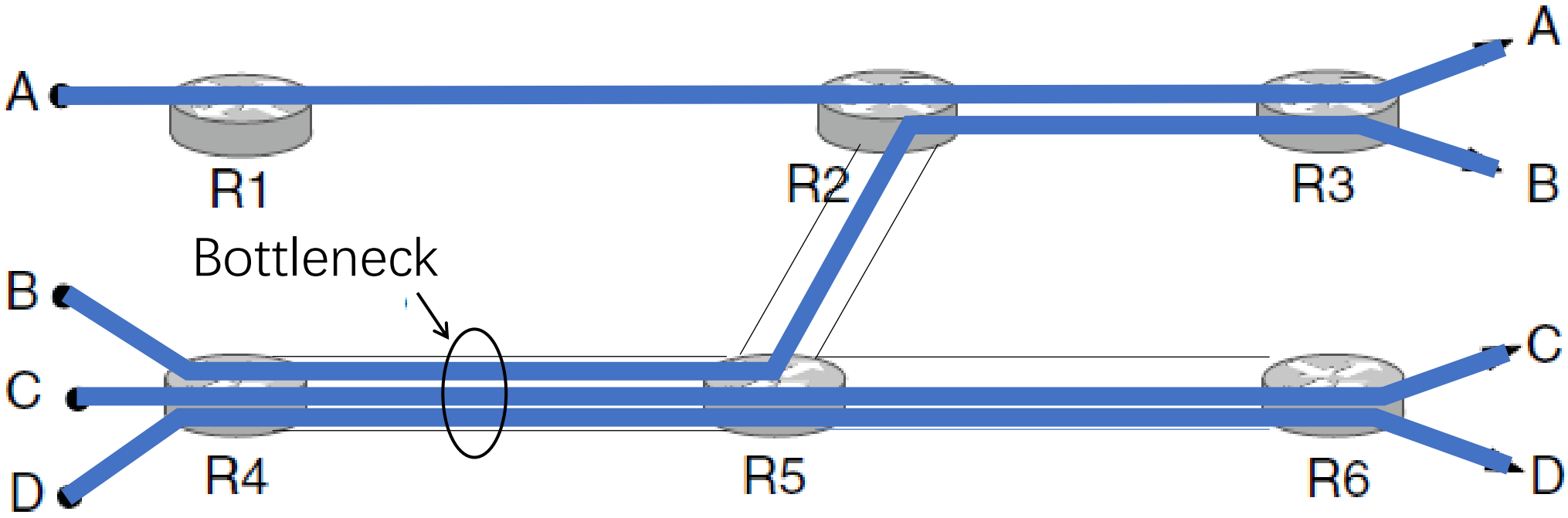
Max-Min Fairness



- 要找到公平的网络，想象“将水倒入网络”的过程
  1. 所有流量从速率为0开始
  2. 增加流量，直到网络中出现新的瓶颈
  3. 固定瓶颈流量的速率
  4. 转到步骤2，查看是否有剩余流量

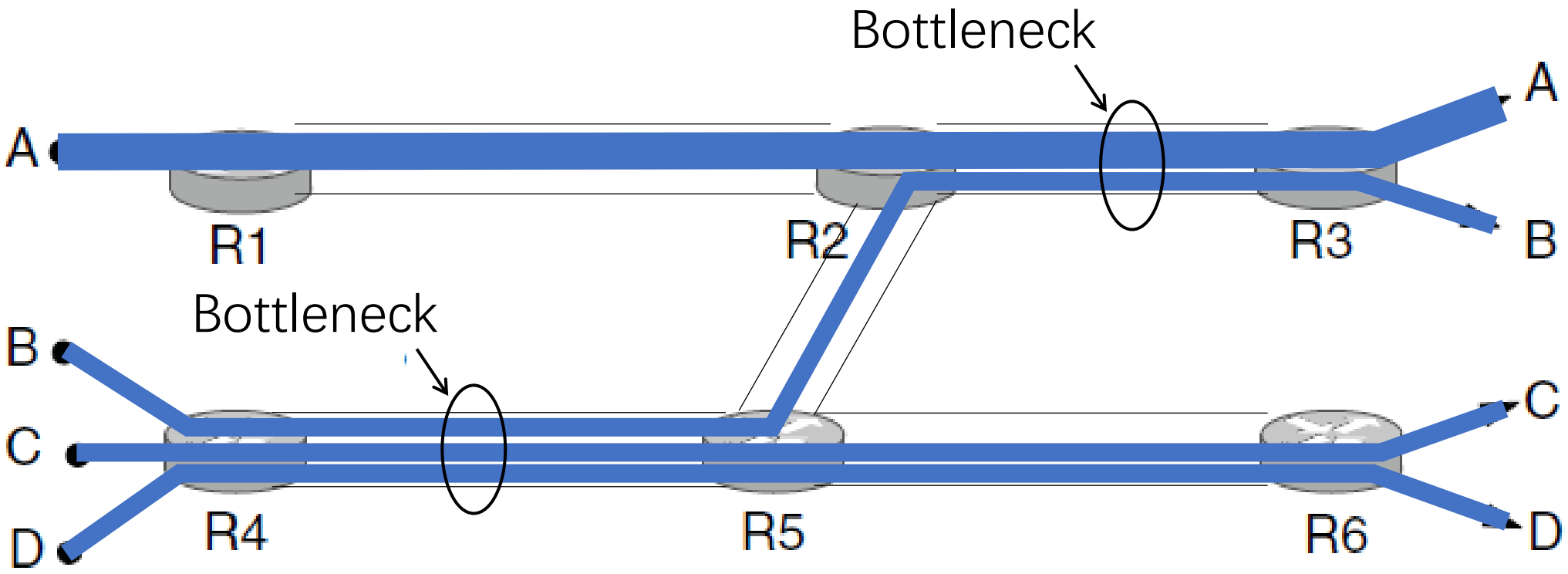
Max-Min Example

- 当 rate=1/3, 流 B, C 和 D 的瓶颈 R4—R5
  - 固定 B, C 和 D, 继续增加 A 的流量



Max-Min Example (2)

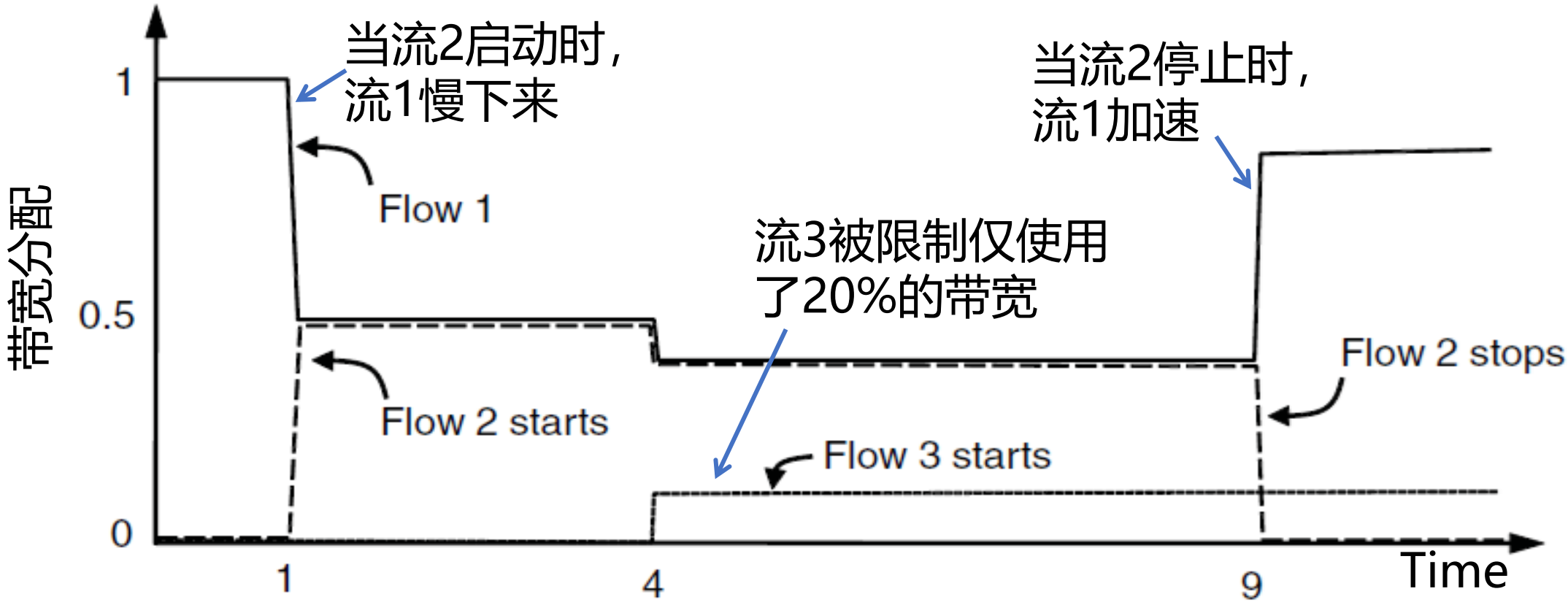
- 当 rate=2/3, 流 A 出现瓶颈 R2—R3



带宽随时间调整

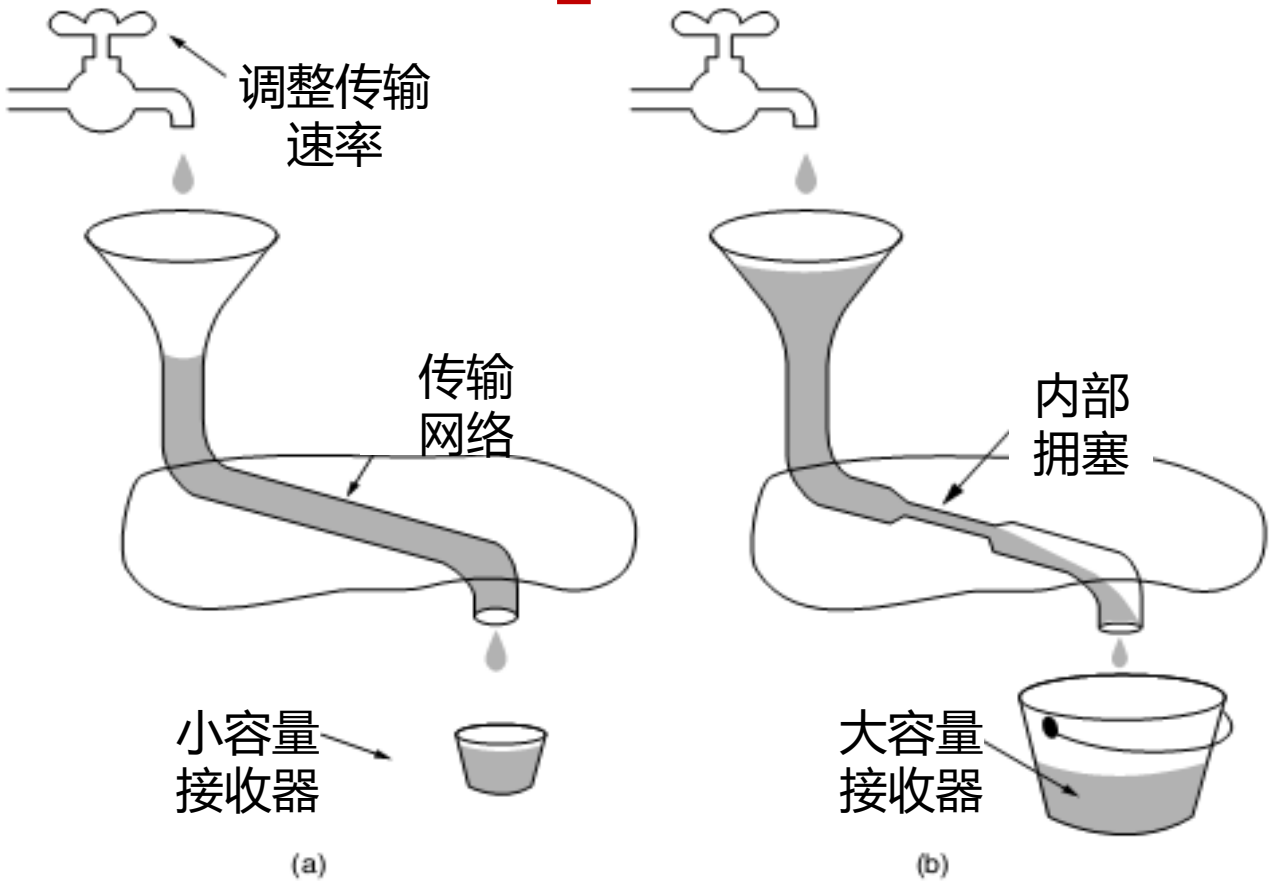
➢带宽分配：随着流的开始和停止而变化

- 流1、2、3共享同一个瓶颈链路



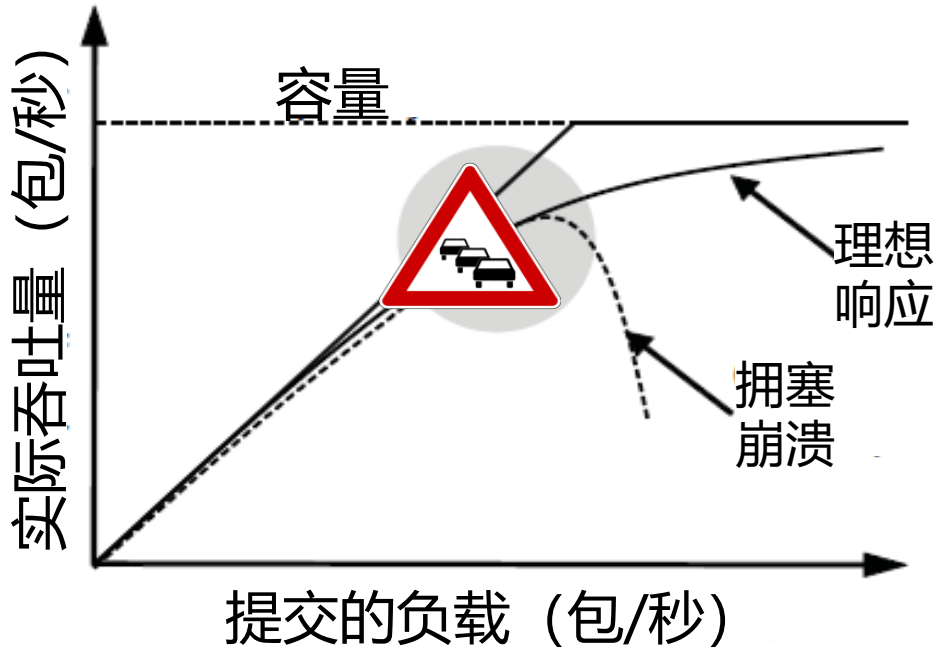
调整发送速率

流量控制 + 拥塞控制



拥塞控制-小结

- 拥塞发生的原因：输入>输出，甚至速率持续存在
- 拥塞的后果
  - 实际吞吐量降低
  - 延迟和丢包急剧上升
- 拥塞控制的目标
  - 使发送方以合适的速度发送
  - 效率与公平难以兼顾
- 带宽调整
  - 随时间动态调整
  - 调整信号、调整策略的选择



本节内容

UDP介绍

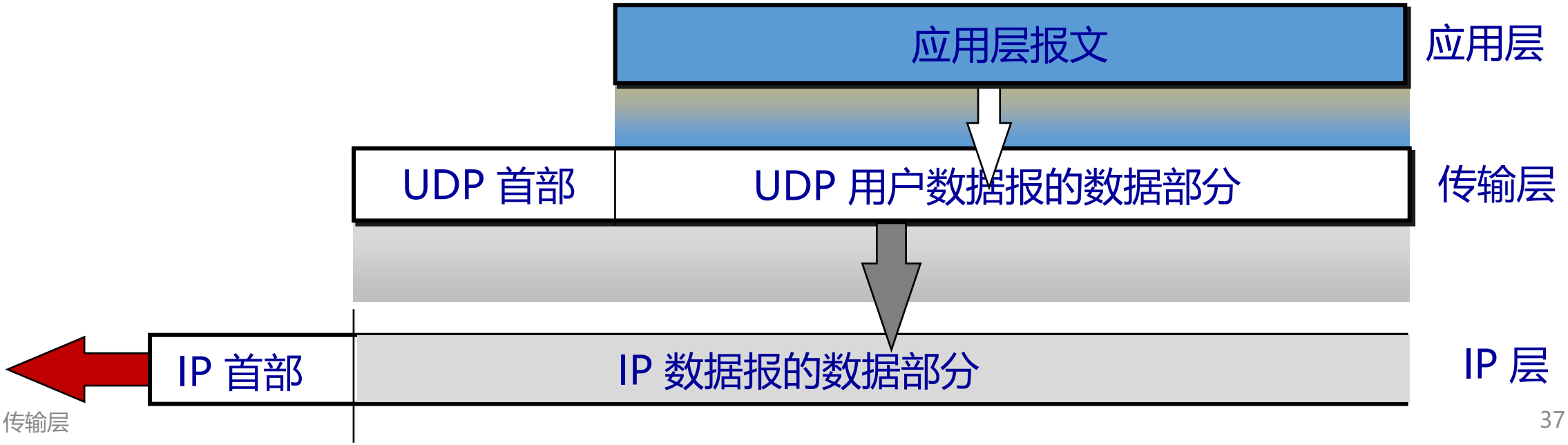
- 设计最最简单的传输层协议UDP
  - 不要连接管理，不要拥塞控制，不要握手……想发就发！
- UDP (User Datagram Protocol)
  - 只在 IP 的数据报服务之上增加了很少一点的功能
  - 增加复用和分用的功能
  - 增加差错检测的功能
- UDP特点
  - 虽然 UDP 用户数据报只能提供不可靠的交付
  - 但 UDP 在某些方面有其特殊的优点

UDP 的主要特点

- UDP 是无连接的
  - 发送数据之前不需要建立连接，因此减少了开销和发送数据之前的时延
- UDP 使用尽最大努力交付
  - 即不保证可靠交付，因此主机不需要维持复杂的连接状态表
- UDP 是面向报文的
  - UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界
  - UDP 一次交付一个完整的报文
- 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：很适合多媒体通信的要求
- UDP 支持一对一、一对多、多对一和多对多的交互通信
- UDP 的首部开销小
  - 只有 8 个字节，比 TCP 的 20 个字节的首部要短

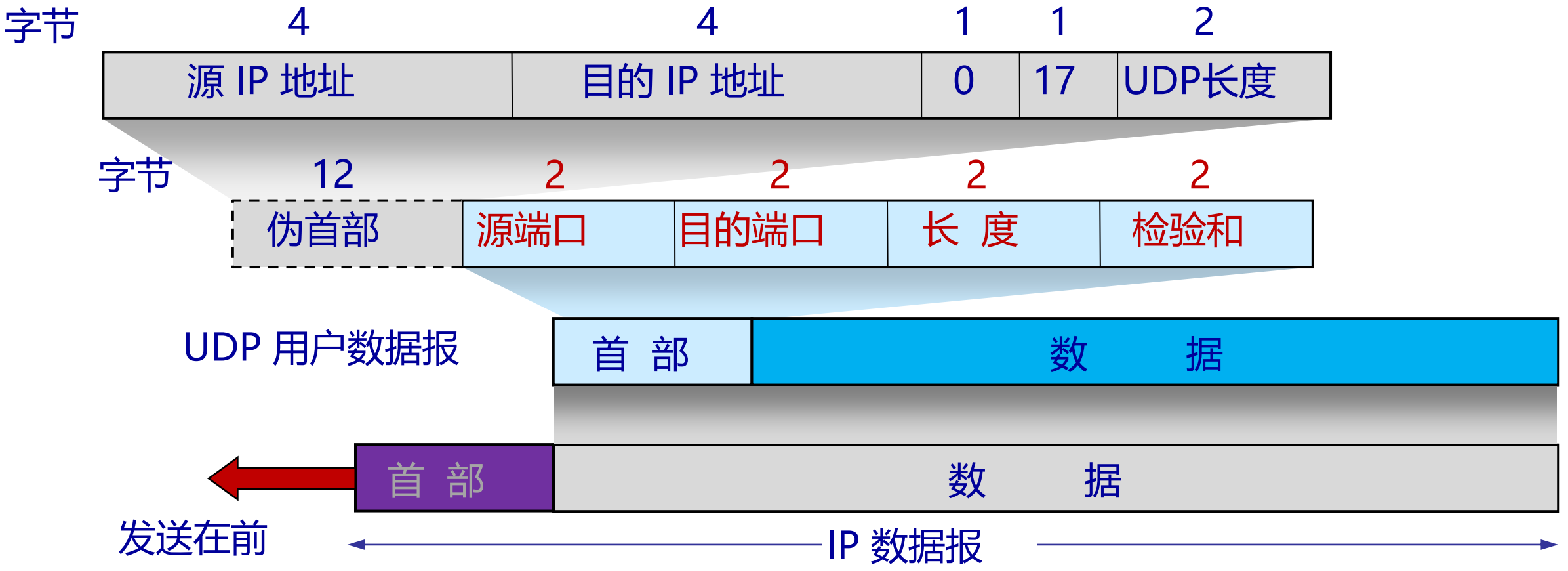
UDP 是面向报文的

- 应用程序必须选择合适大小的报文
  - 若报文太长，UDP 把它交给 IP 层后，IP 层在传送时可能要进行分片，这会降低 IP 层的效率
  - 若报文太短，UDP 把它交给 IP 层后，会使 IP 数据报的首部的相对长度太大，这也降低了 IP 层的效率



UDP 的首部格式

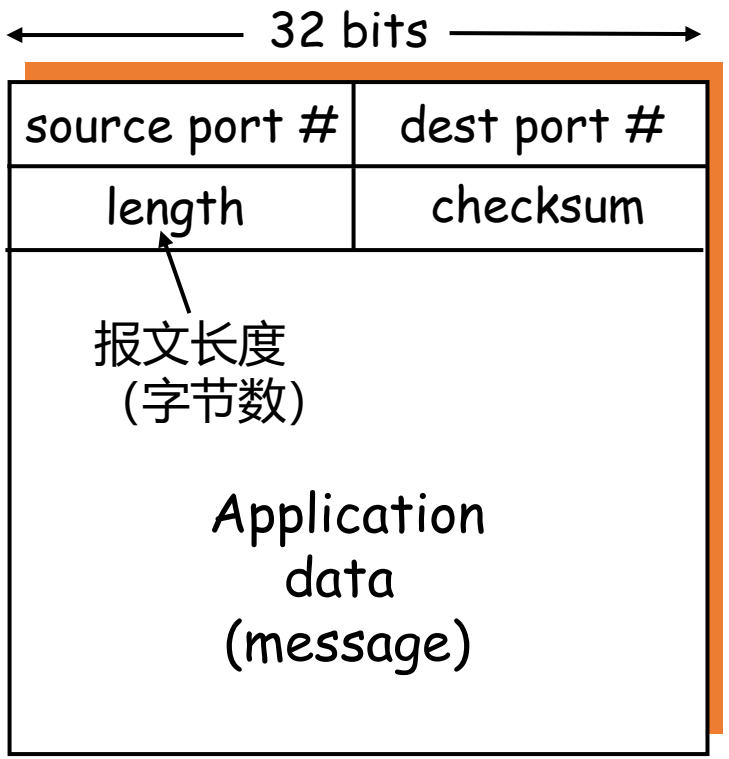
用户数据报 UDP 有两个字段：数据字段+8个字节的首部字段



UDP用户数据报的首部和伪首部

UDP报文格式与IPv4伪头部

UDP报文格式



流Flow的定义：五元组

UDP头部

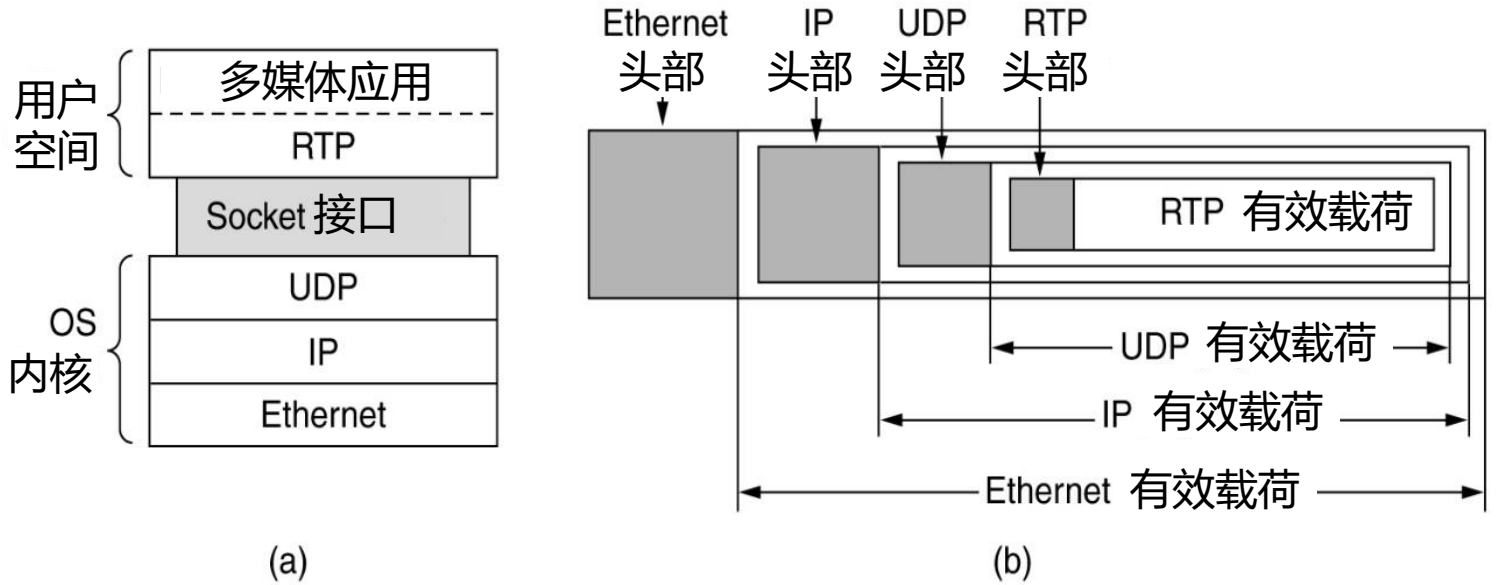
0		16	31
源端口		目的端口	
UDP长度		UDP校验和	
数据			

IPv4伪头部

0	8	16	31
源IP地址			
目的IP地址			
00000000	协议 (=17)	UDP长度	
TCP：	协议字段(=6)	TCP长度	

UDP用例：实时传输协议

- RTP(RFC3550)—广泛应用于多媒体应用程序
  - 运行在用户空间，是一个在应用层上实现的传输协议
  - 以数据包的形式传输音频和视频数据
- 接收端的处理
  - 在正确的时间播放音频和视频



UDP-小结

- UDP协议
  - 无连接、不可靠但简单、快速的传输层协议
  - 应用可尽可能快地发送报文
  - 无建立连接的延迟，不限制发送速率（不进行拥塞控制和流量控制）
  - 报头开销小，协议处理简单
- UDP适合哪些应用?
  - 以单次请求/响应为主的应用：如DNS
  - 容忍丢包但对延迟敏感的应用：如实时会议
- 思考：若应用希望利用UDP的优点，还想要可靠传输呢?
  - 应用层实现可靠性，甚至还会实现拥塞控制

本节内容

7.1 传输服务需求与设计

7.2 传输协议的要素

7.3 拥塞控制

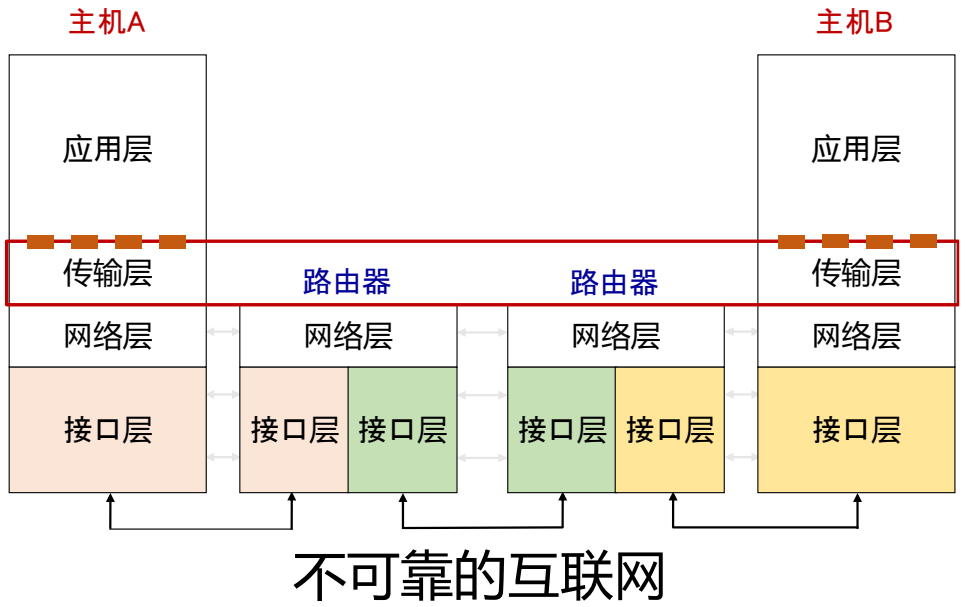
7.4 因特网传输协议UDP

7.5 因特网传输协议TCP

- 7.5.1 TCP概述
- 7.5.2 TCP服务模型
- 7.5.3 TCP协议
- 7.5.4 TCP段的头
- 7.5.5 TCP连接建立
- 7.5.6 TCP连接释放
- 7.5.7 TCP连接管理模型
- 7.5.8 TCP滑动窗口
- 7.5.9 TCP计时器管理
- 7.5.10 TCP拥塞控制

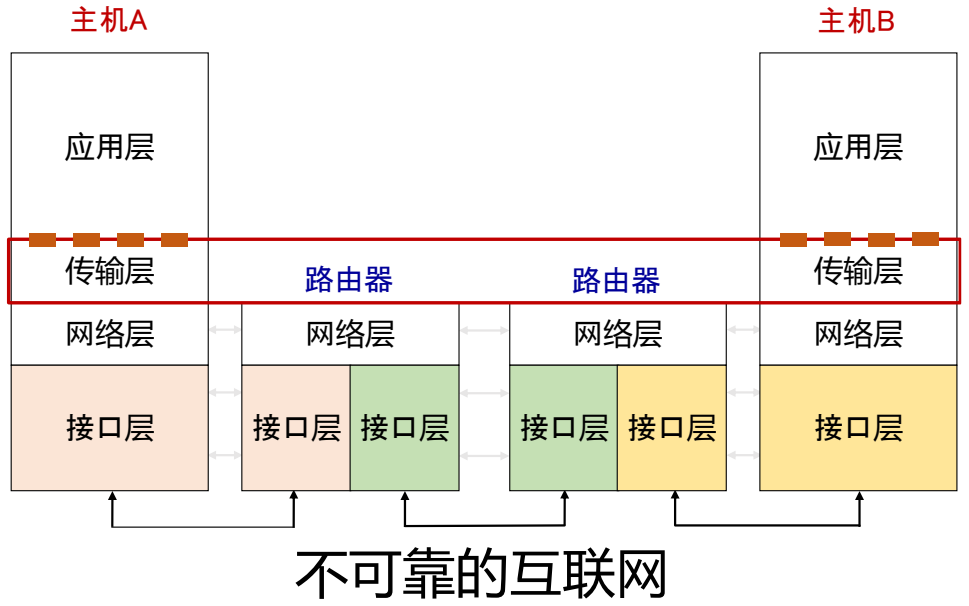
TCP概述

- 传输控制协议TCP（Transmission Control Protocol）
  - 在不可靠的互联网上提供可靠的端到端服务
  - 面向连接的、可靠的、端到端的、基于字节流的传输协议
  - RFC 793+，1122，1323，2018，2581，2873，2988等
- TCP 是面向字节流的
  - TCP不保证接收方应用程序所收到的数据块和发送方应用程序所发出的数据块具有对应大小的关系
  - 接收方应用程序收到的字节流和发送方应用程序发出的字节流完全一样

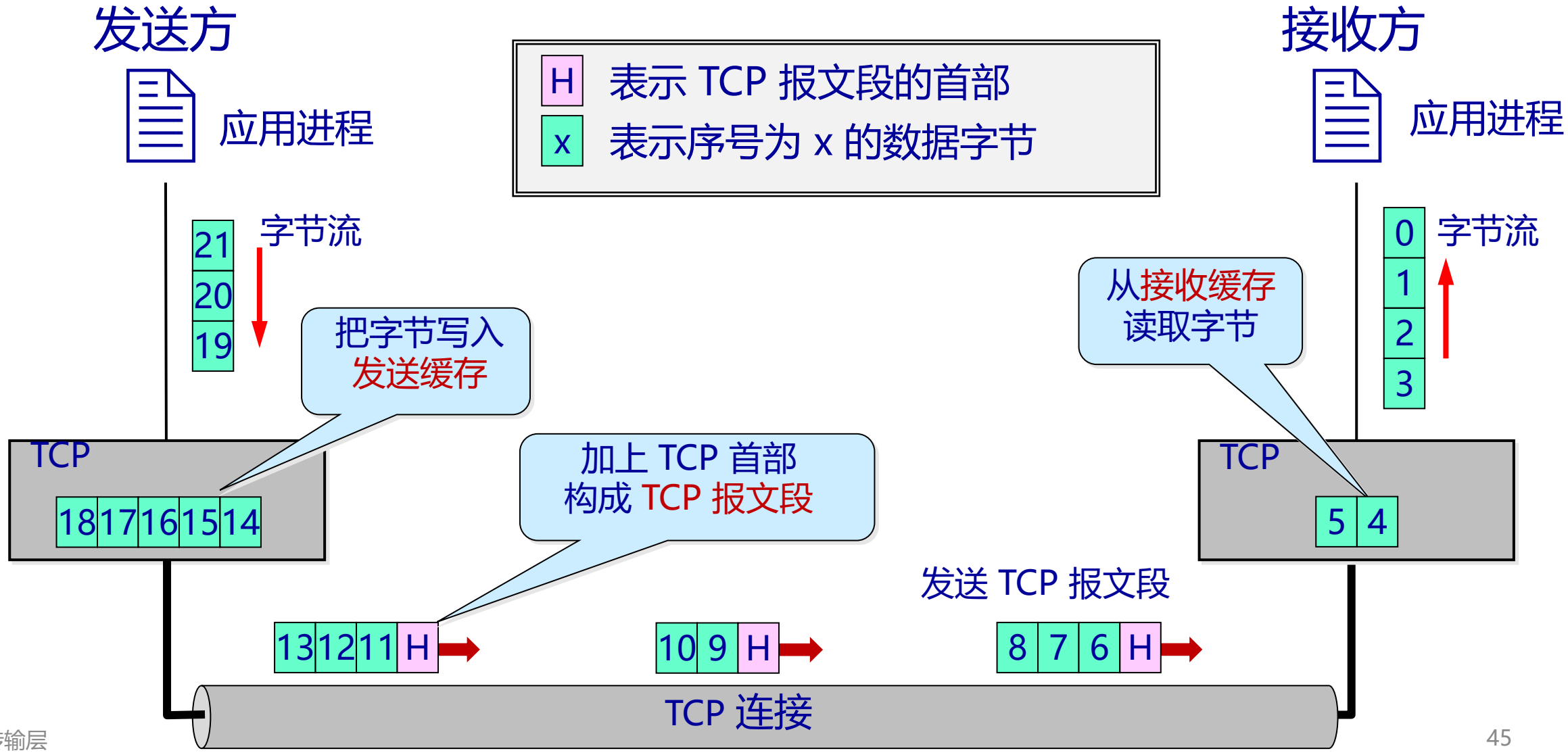


TCP 最主要的特点

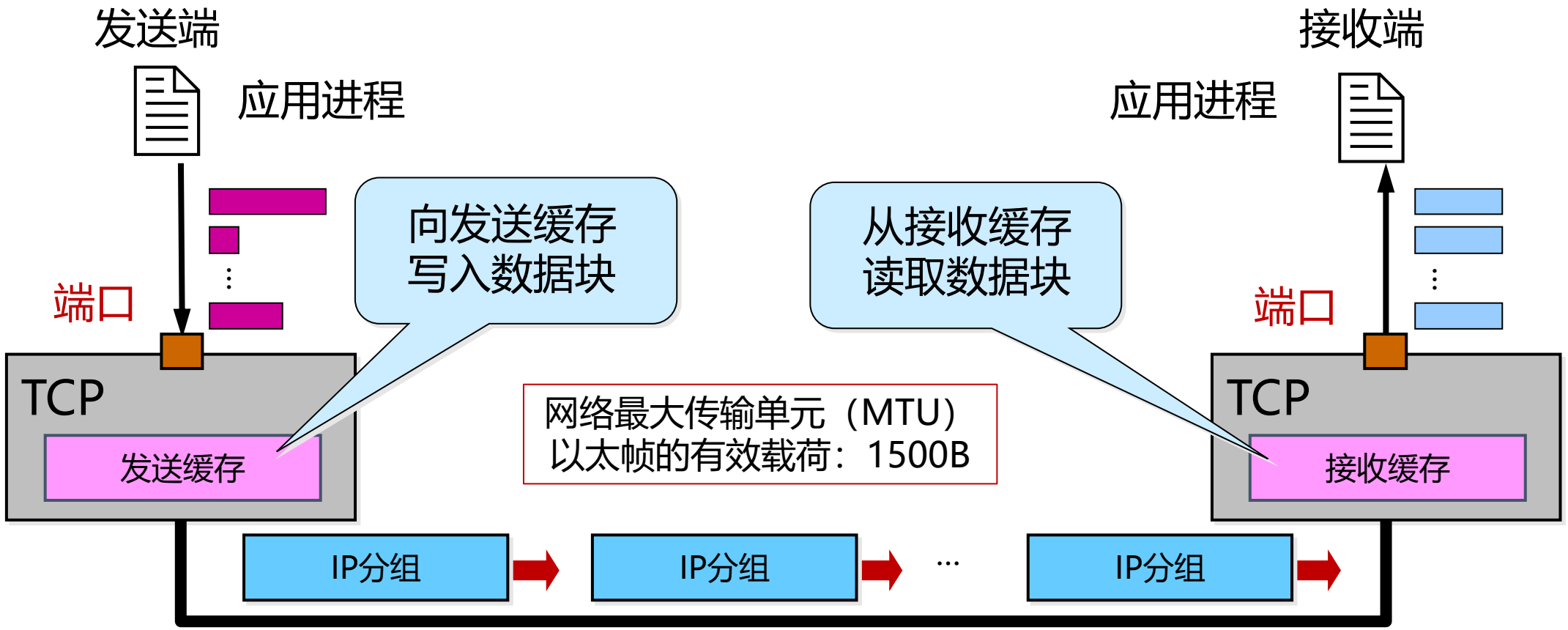
- TCP 是面向连接的传输层协议
  - 每一条 TCP 连接只能有两个端点 (endpoint)
  - 每一条 TCP 连接只能是端到端的（一对一）
  - TCP 提供可靠交付的服务
  - TCP 提供全双工通信
- 面向字节流
  - TCP 中的“流”(stream)指的是流入或流出进程的字节序列
  - “面向字节流”的含义：虽然应用程序和 TCP 的交互是一次一个数据块，但 TCP 把应用程序交下来的数据看成仅仅是一连串无结构的字节流



TCP 面向流的概念


发送方
应用进程
H 表示 TCP 报文段的首部
x 表示序号为 x 的数据字节
接收方
应用进程
21
20
19
字节流
把字节写入
发送缓存
TCP
18 17 16 15 14
加上 TCP 首部
构成 TCP 报文段
13 12 11 H
10 9 H
8 7 6 H
发送 TCP 报文段
TCP 连接
从接收缓存
读取字节
TCP
5 4
0
1
2
3
字节流

TCP 面向流的概念



- TCP 不关心应用进程一次把多长的报文发送到 TCP 缓存
- TCP 对连续的字节流进行分段，形成 TCP 报文段

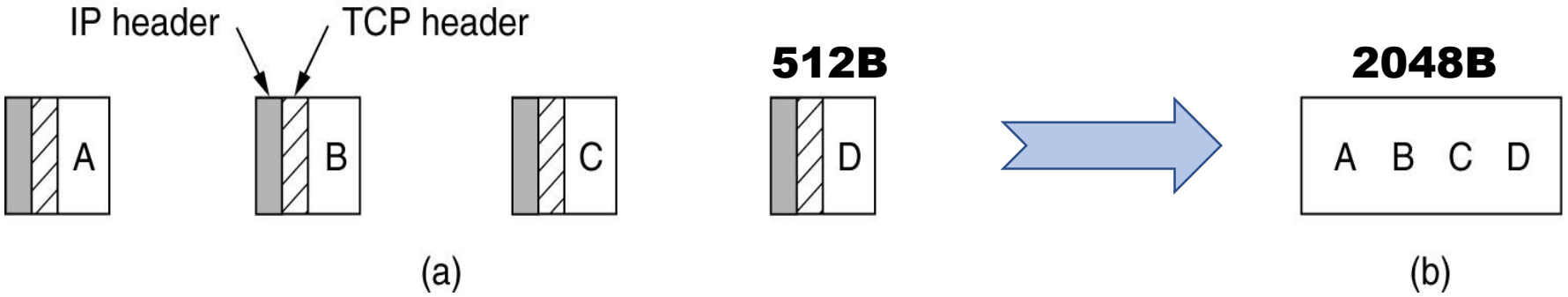
TCP服务模型

- TCP：可靠的端到端的字节流
  - 套接字
    - 主机ip地址+16位数值的端口号
    - 一个套接字可用于多个连接
  - 端到端之间不保留消息边界
    - TCP连接是字节流，不是消息流

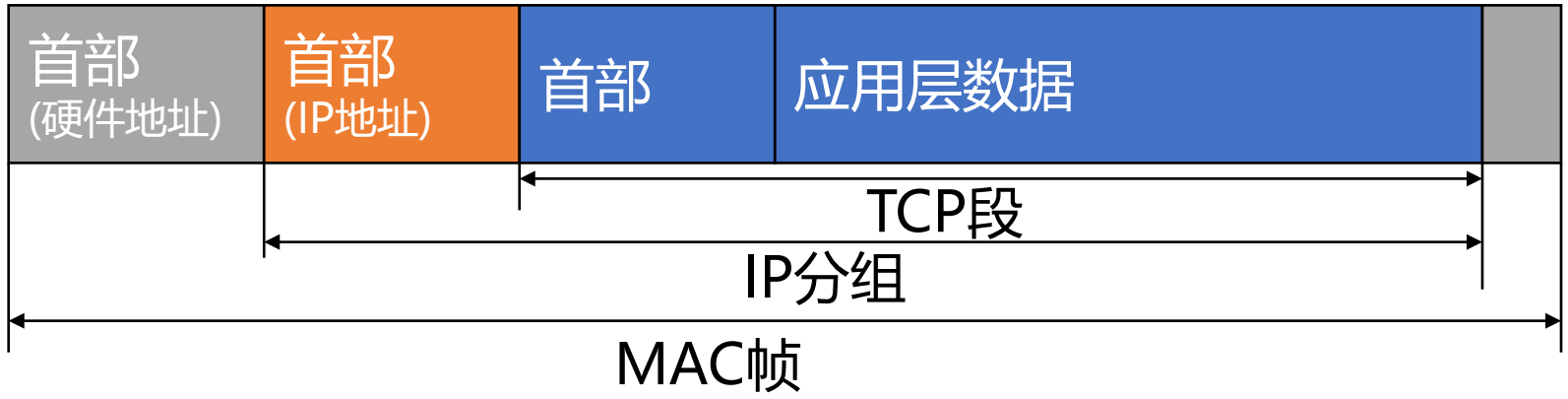
a、发送端4个512字节数据块

b、接收端1个2048字节数据块

端口	协议	用途
20、21	FTP	文件传输
22	SSH	远程控制台（替代远程登录Telnet）
25	SMTP	电子邮件
80	HTTP	万维网
110	POP-3	远程邮件访问
143	IMAP	因特网邮件访问
443	HTTPS	安全web（SSL/TLS上的HTTP）
543	RTSP	媒体播放控制



TCP服务模型(2)



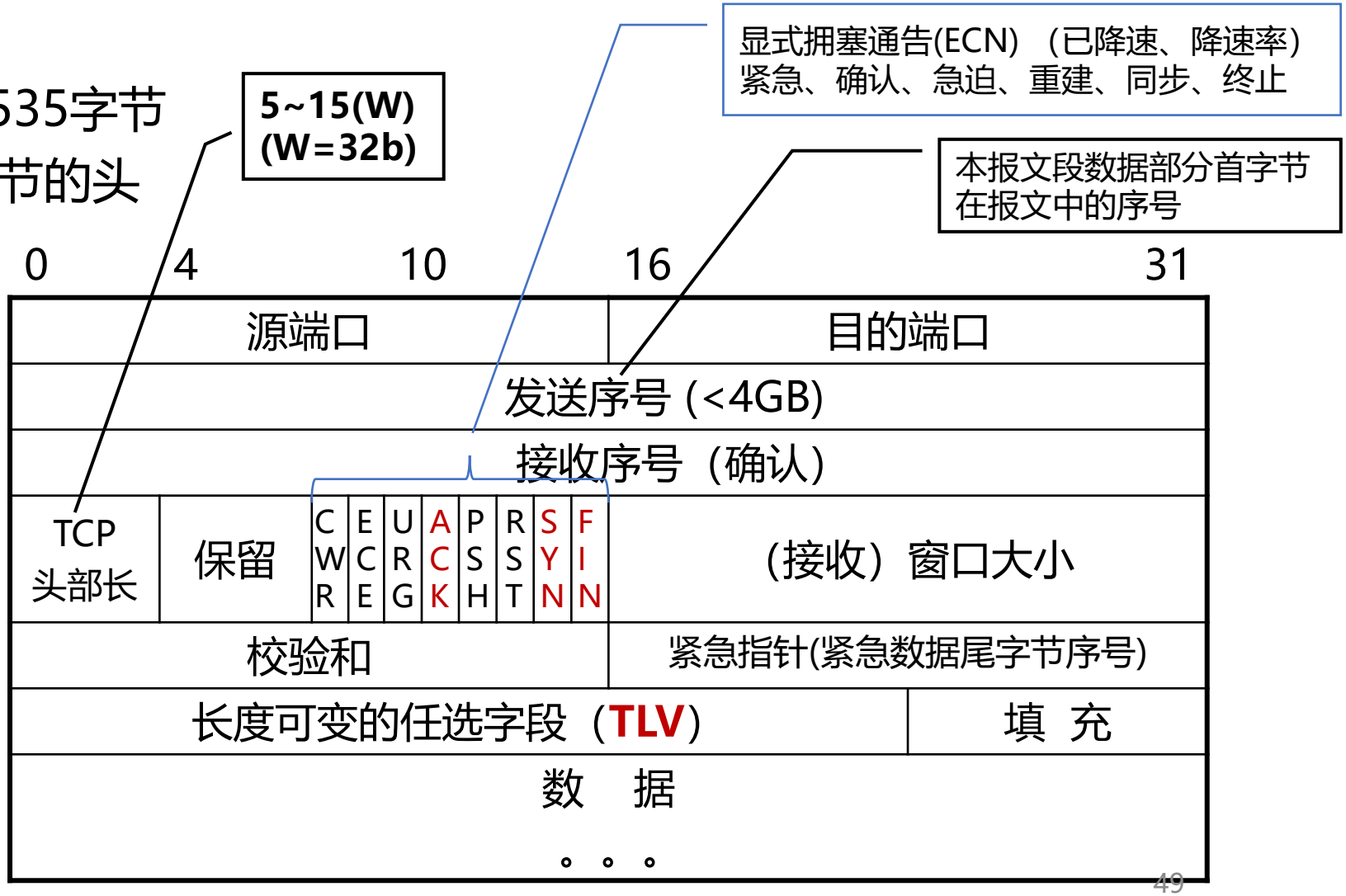
流Flow的定义
五元组

- TCP与UDP
- 端口：16b
- 熟知端口：< 256

应用层	RPC	SNMP	TFTP	SMTP	FTP	SSH
传输层	111	161	69	25	21	22
	UDP			**TCP**		
网络层	**IP**					
	与各种网络接口					

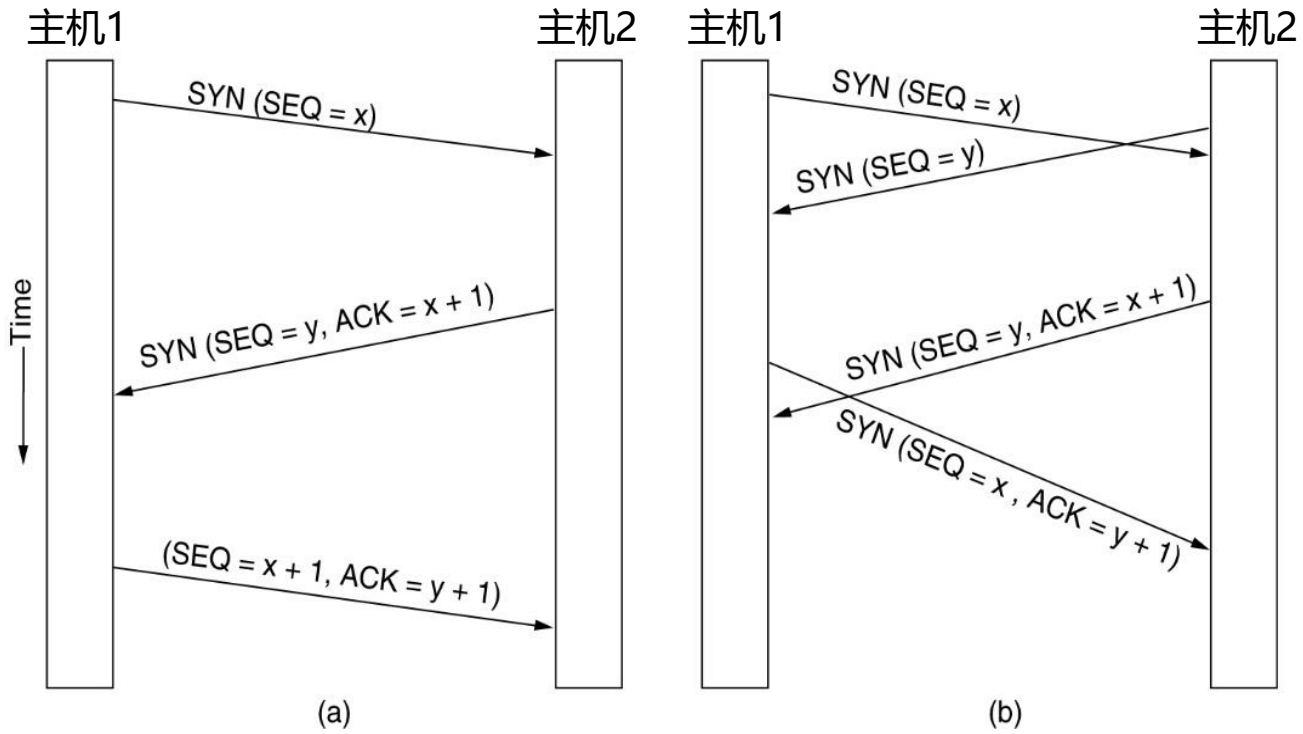
TCP段的头

- TCP段的结构
  - IP头中的length最大为65535字节
  - TCP20字节的头，IP 20字节的头
  - TCP段最大净载荷：65535-20-20=65495
  - SYN、ACK
    - 连接请求：1、0
    - 连接应答：1、1
  - FIN
    - 释放连接
  - 选项
    - 最大段长MSS
    - 时间戳（PAWS）

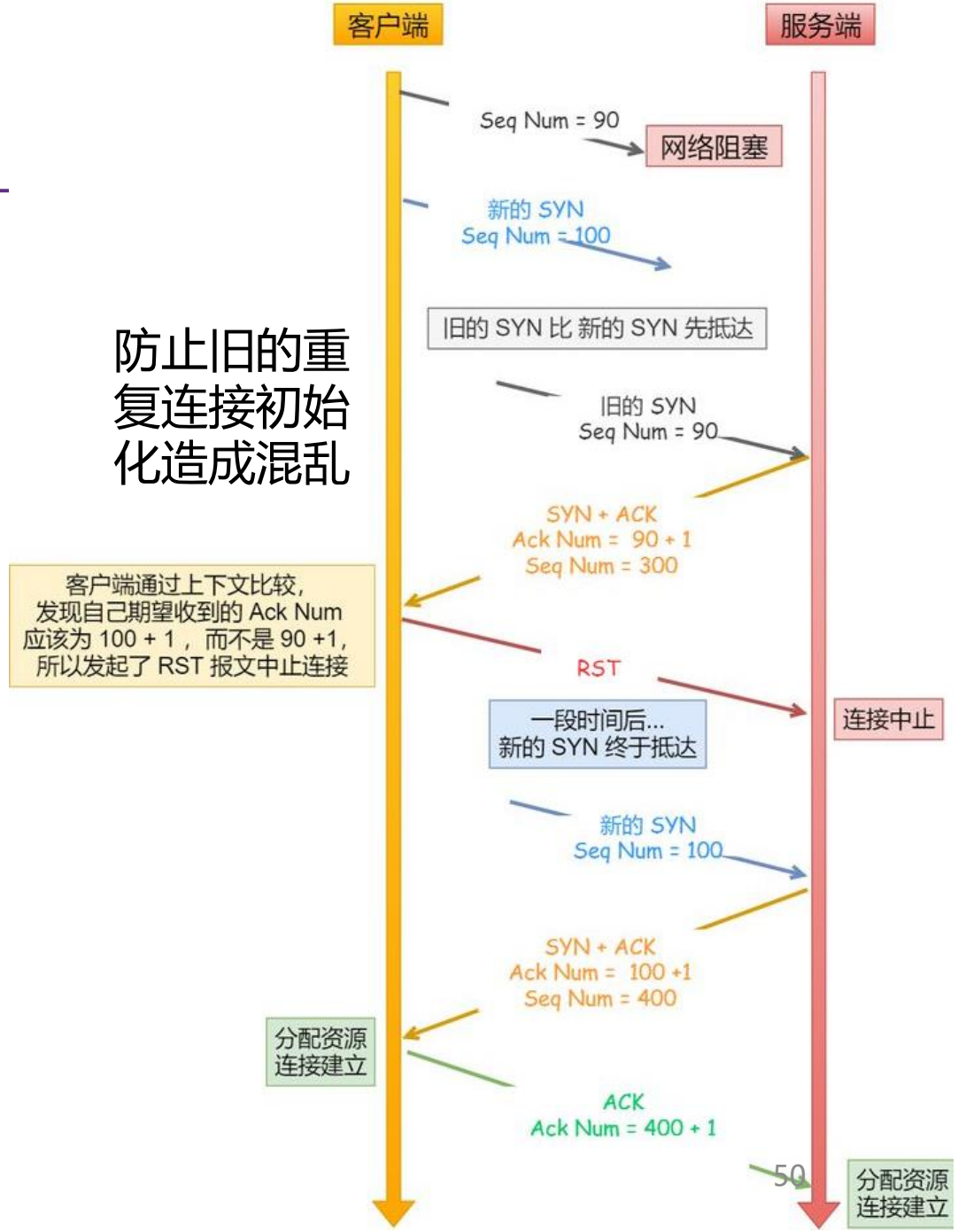


TCP连接建立

- 建立连接采用三次握手方法



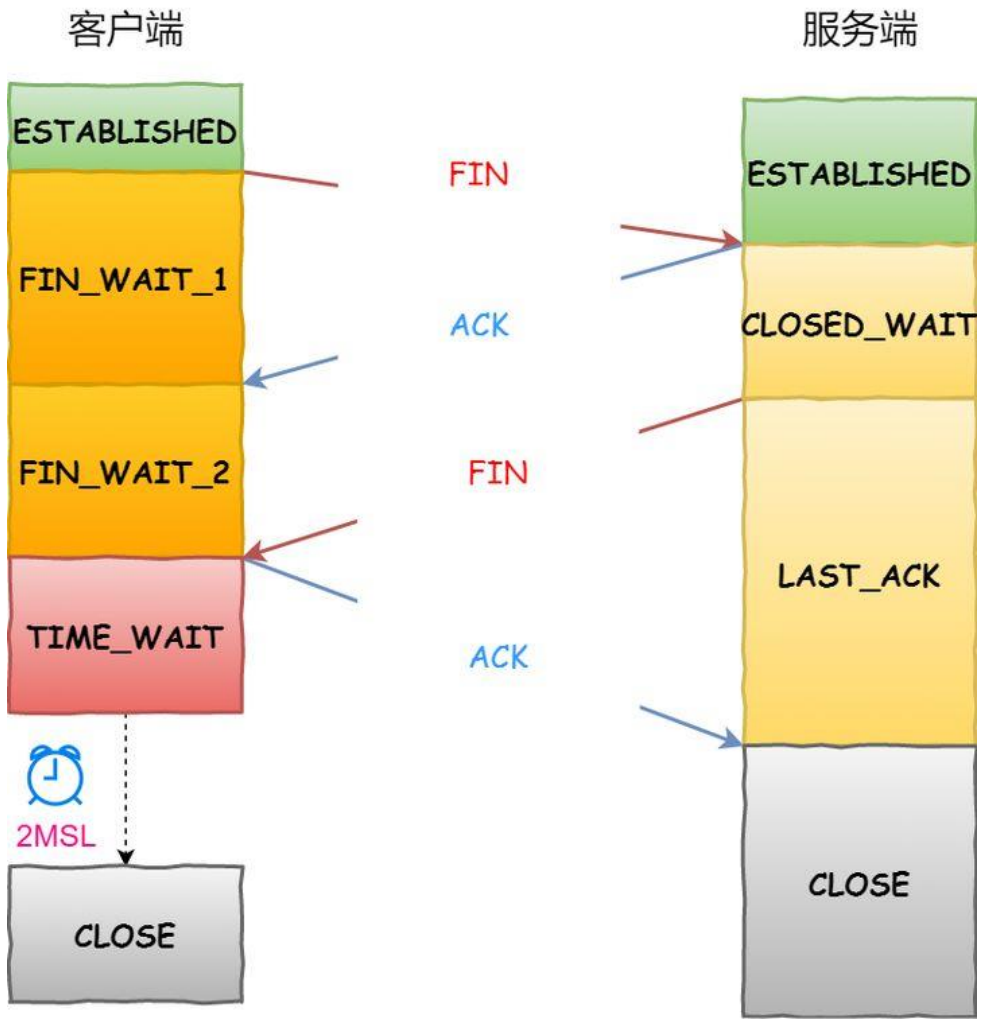
防止旧的重复连接初始化造成混乱



TCP连接释放

- 释放连接（四次挥手）
  - 发送设置了FIN标志位的TCP段
  - 四个TCP段→三个TCP段
    - FIN
    - ACK
    - FIN
    - ACK

    （ACK+FIN）
  - 计时器



TCP连接管理模型

- TCP连接管理有限自动机状态

状态	描述
CLOSED	没有连接，挂起
LISTEN	服务器等待入境呼叫
SYN RCVD	连接请求已到达，等待ACK
SYN SENT	应用层开始打开一个链接
ESTABLISHED	正常数据传送状态
FIN WAIT 1	应用层已完成数据发送
FIN WAIT 2	另一端同意释放连接
TIMED WAIT	等待所有数据包消亡
CLOSING	两端同时关闭连接
CLOSE WAIT	另一端已发起关闭连接
LAST ACK	等待所有数据包消亡

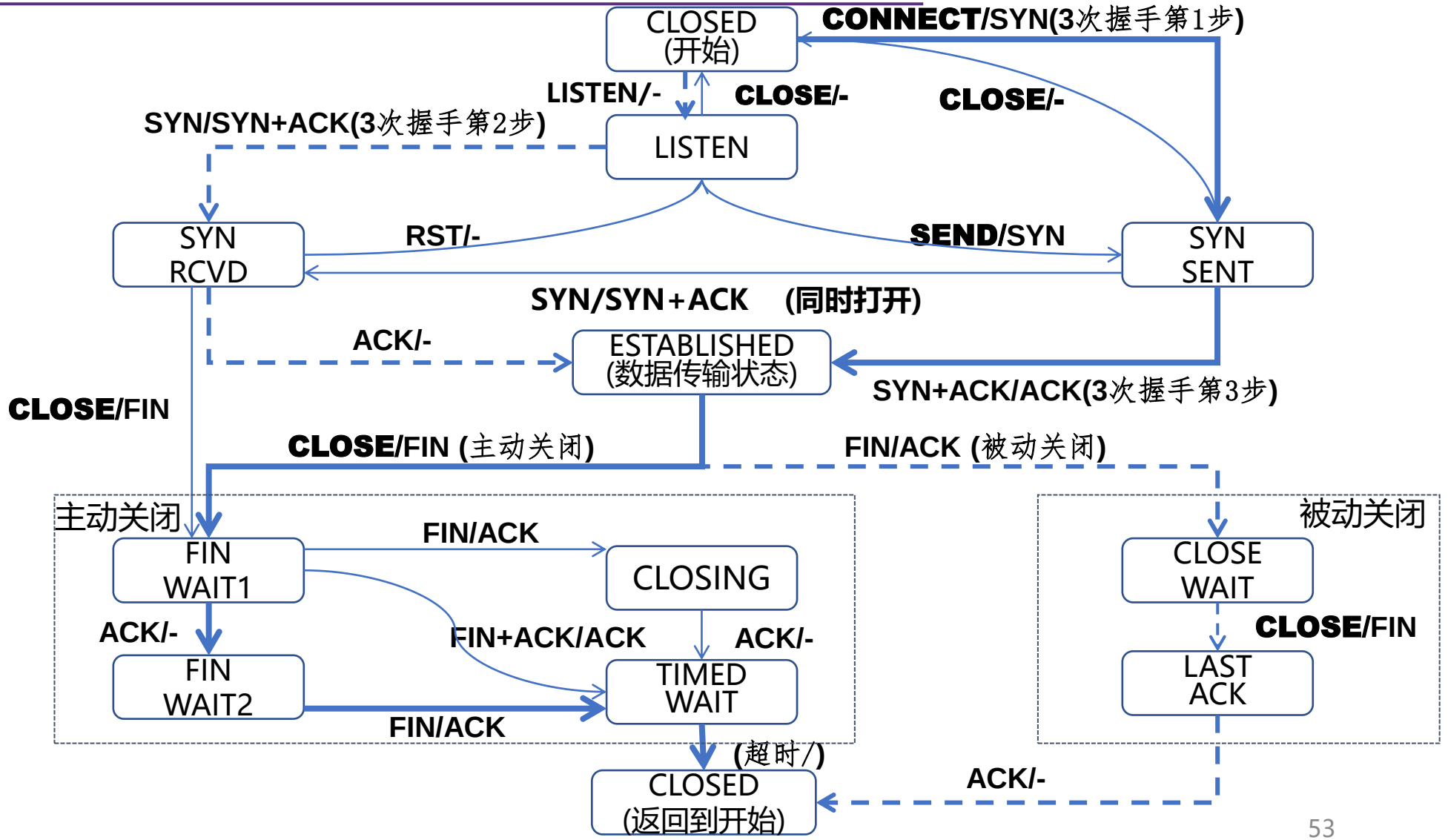
TCP连接管理有限自动机

事件/动作

客户端正常路径

客户端非正常路径

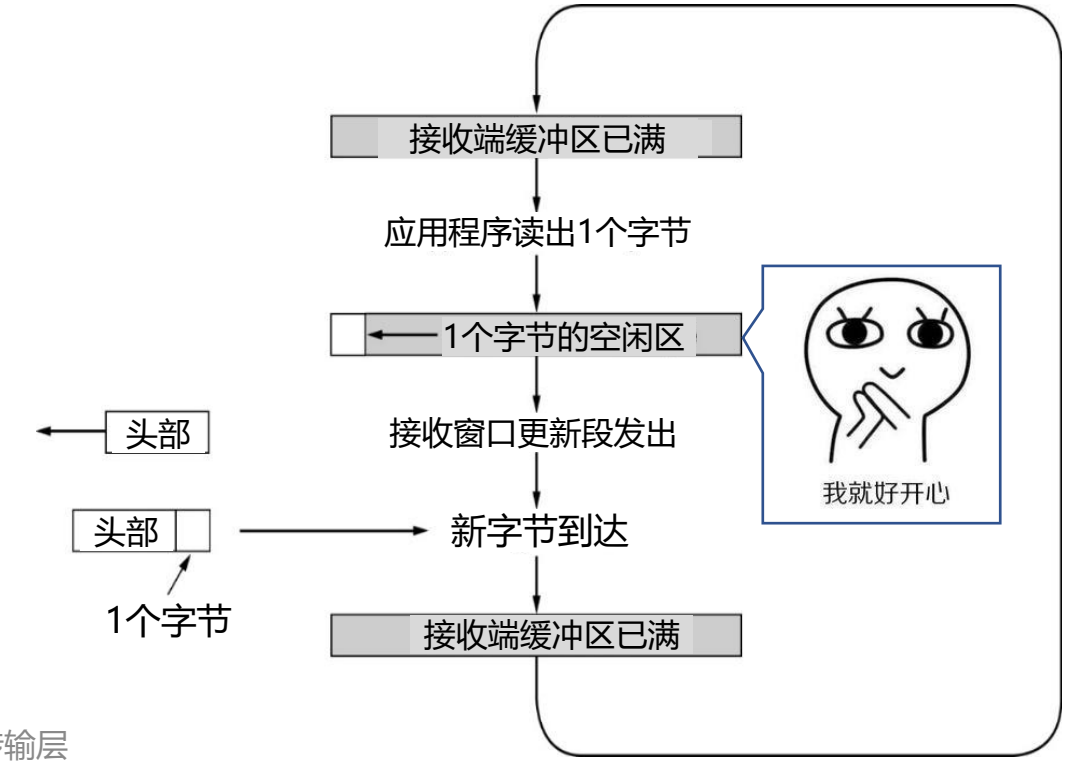
服务器端正常路径

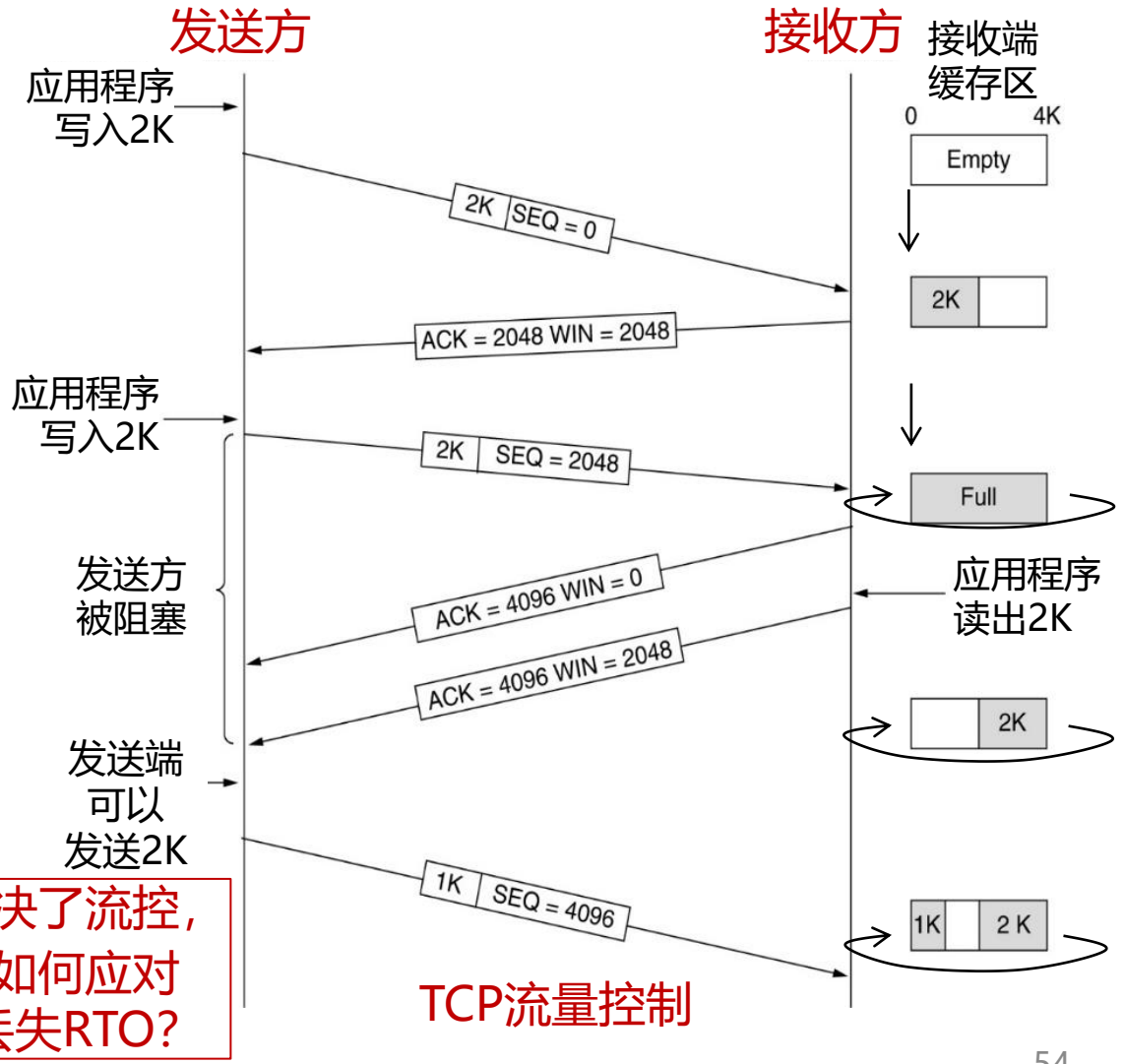


建立了连接，
如何发送数据?

TCP流量控制的滑动窗口

- TCP滑动窗口
  - 接收端流控窗口4096B
- 低能窗口问题
  - 禁止1字节窗口更新段

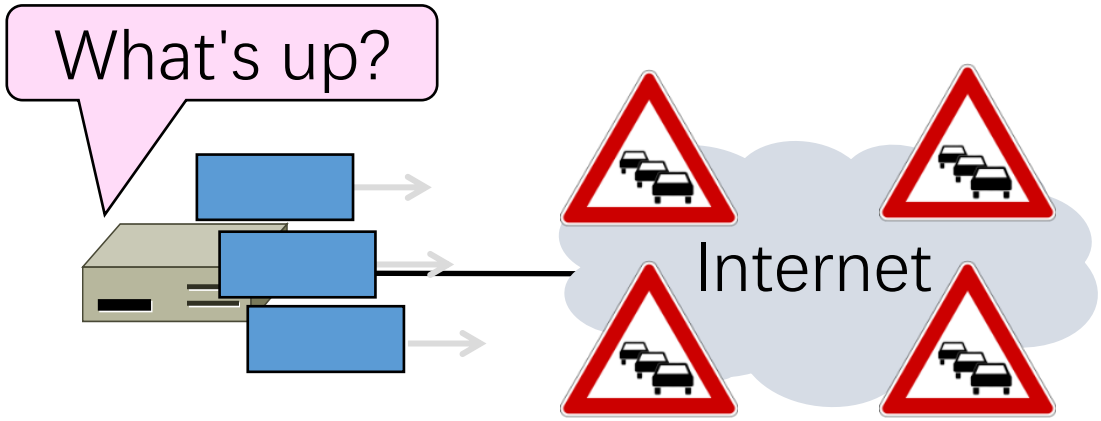


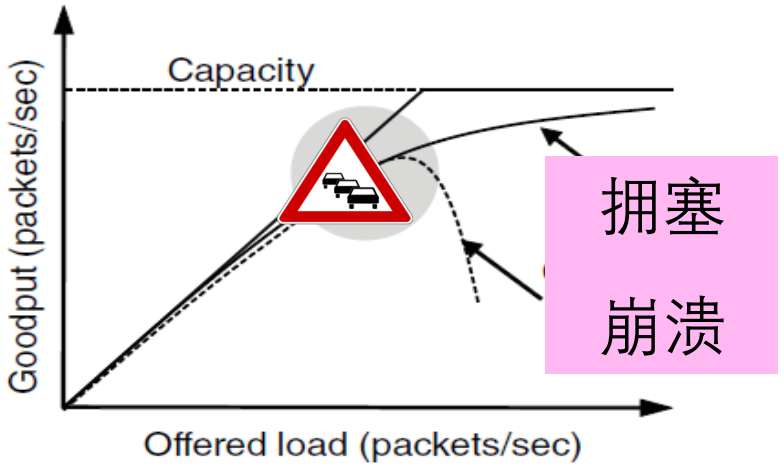


1980s的拥塞崩溃

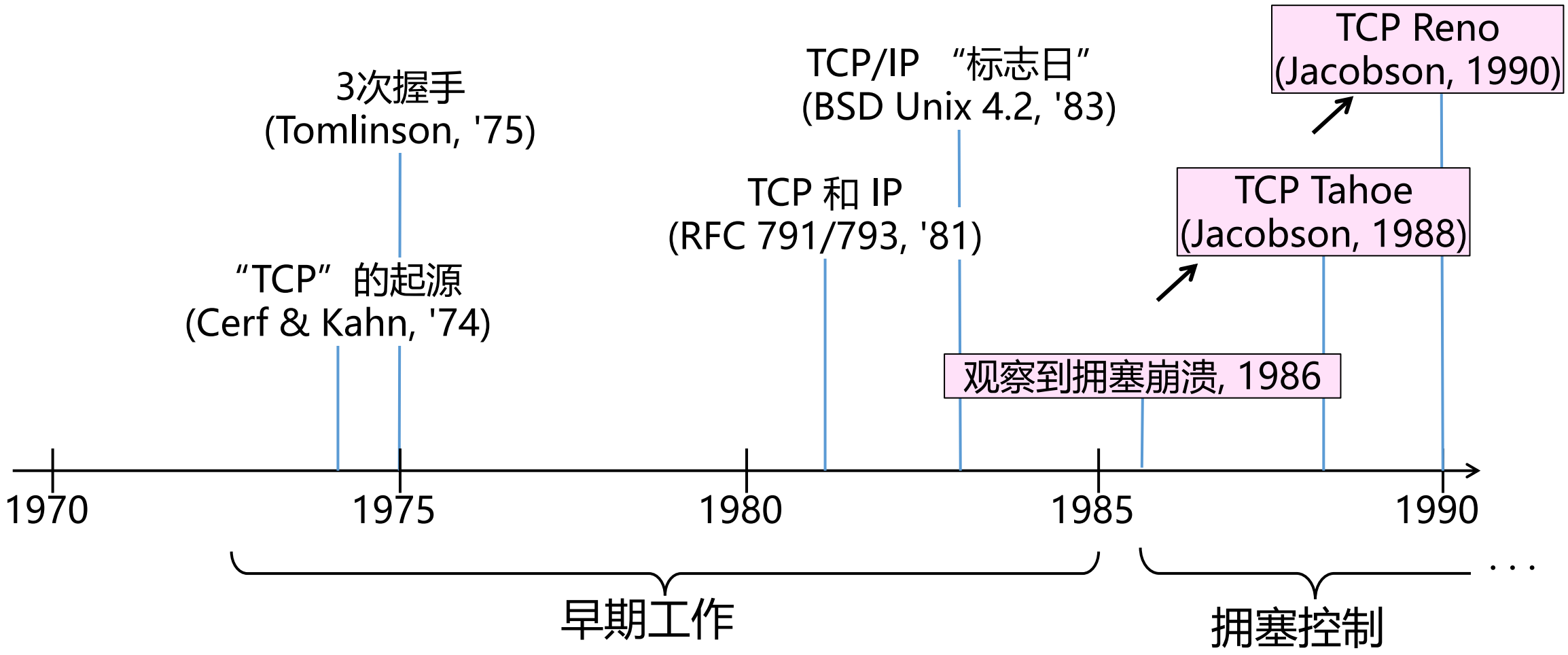
- The story of TCP congestion control
  - 崩溃，控制和多元化
- 早期的TCP使用固定大小的滑动窗口（例如8个数据包）
  - 最初出于可靠性考虑
- 但是随着ARPANET的发展，发生了一些奇怪的事情
  - 链接依然繁忙，但传输速率下降了几个数量级！
  - 队列已满，重传阻塞了网络，实际的吞吐量（goodput）下降

Van Jacobson
思科首席科学家
提出Tahoe/Reno





TCP 发展时间线


TCP Reno
(Jacobson, 1990)
TCP/IP “标志日”
(BSD Unix 4.2, '83)
3次握手
(Tomlinson, '75)
TCP 和 IP
(RFC 791/793, '81)
TCP Tahoe
(Jacobson, 1988)
“TCP”的起源
(Cerf & Kahn, '74)
观察到拥塞崩溃, 1986
1970
1975
1980
1985
1990
...
早期工作
拥塞控制

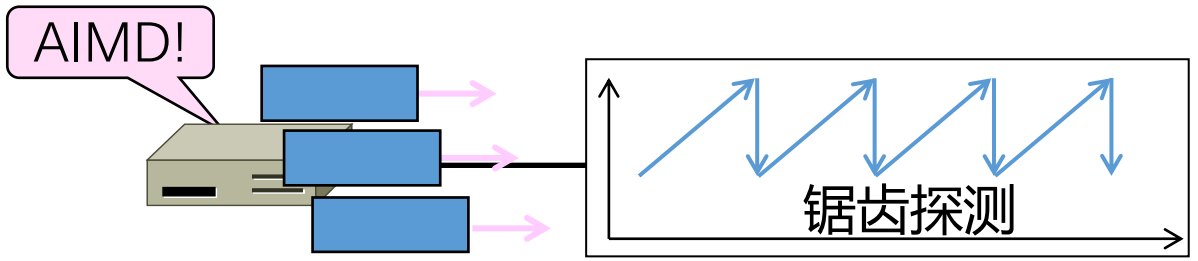
TCP拥塞控制

- 拥塞控制
  - TCP在拥塞控制和可靠传输中发挥主要作用
- 拥塞窗口
  - 发送窗口 = min(拥塞窗口，流控窗口)
  - 根据网络状态调整窗口大小，发送速率与网络带宽匹配
- 拥塞窗口示例
  - 例如，网络：10Mbps，RTT=100ms
  - 目标：拥塞窗口 = 带宽延迟积 =10×0.1= 1Mb = 100(个)×1250(字节)
  - 拥塞窗口从1开始，每RTT加1，100×RTT = 10秒，才能达到100

效率太低怎么办?

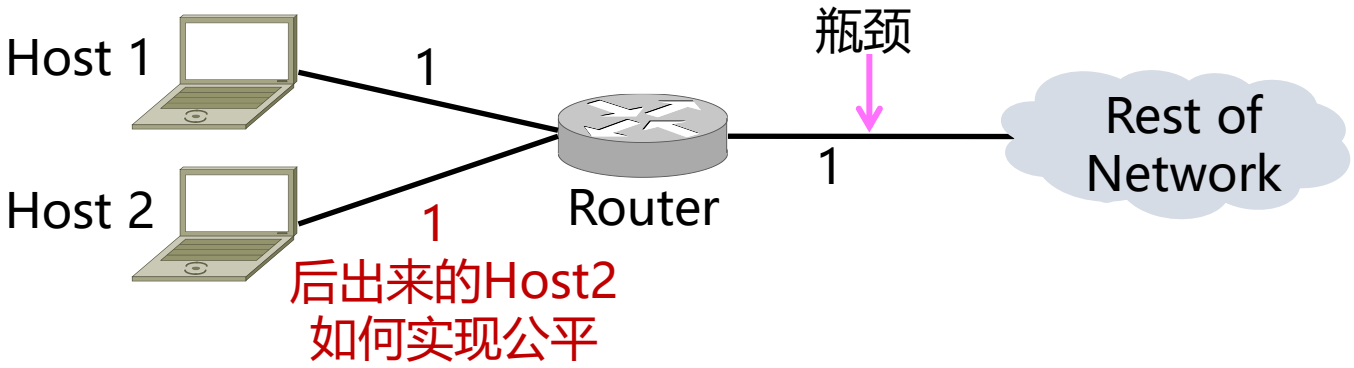
如何确定发送端的发送方案？

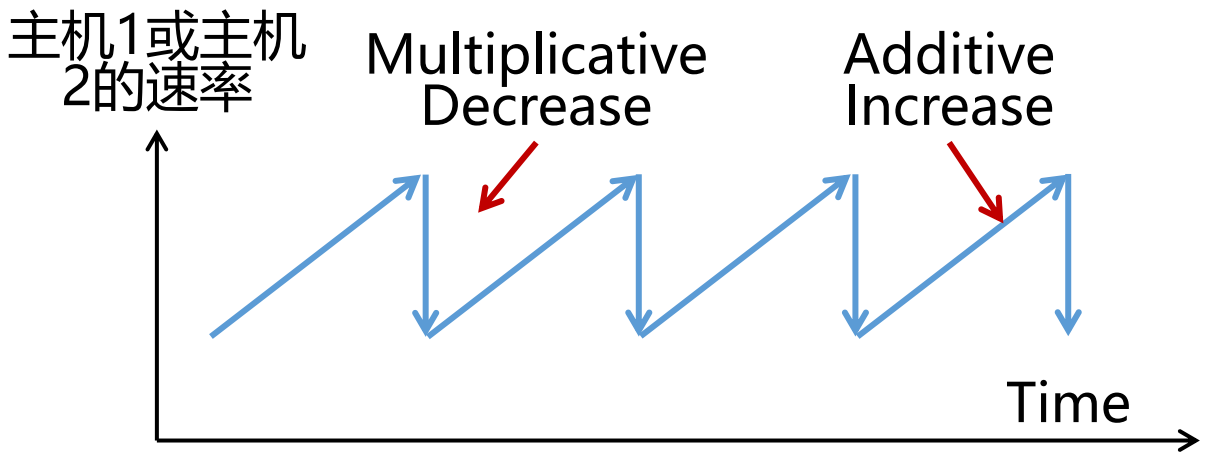
- 发送端如何选取合适的发送速率？目标与挑战
  - 根据网络层的反馈，传输层调整提供的负载
  - 需求：高效，能够紧紧跟随网络状态的变化
  - 需求：公平，先后到达的多个发送端之间
  - 瓶颈多大？在动态变化？漂洋过海哪里是瓶颈？海量终端如何协同？
- 设计思路
  - 快速探测，鼓励后来者（鼓励低速，惩罚高速）
  - 采用AIMD（Additive Increase Multiplicative Decrease) 控制法则



AIMD 法则

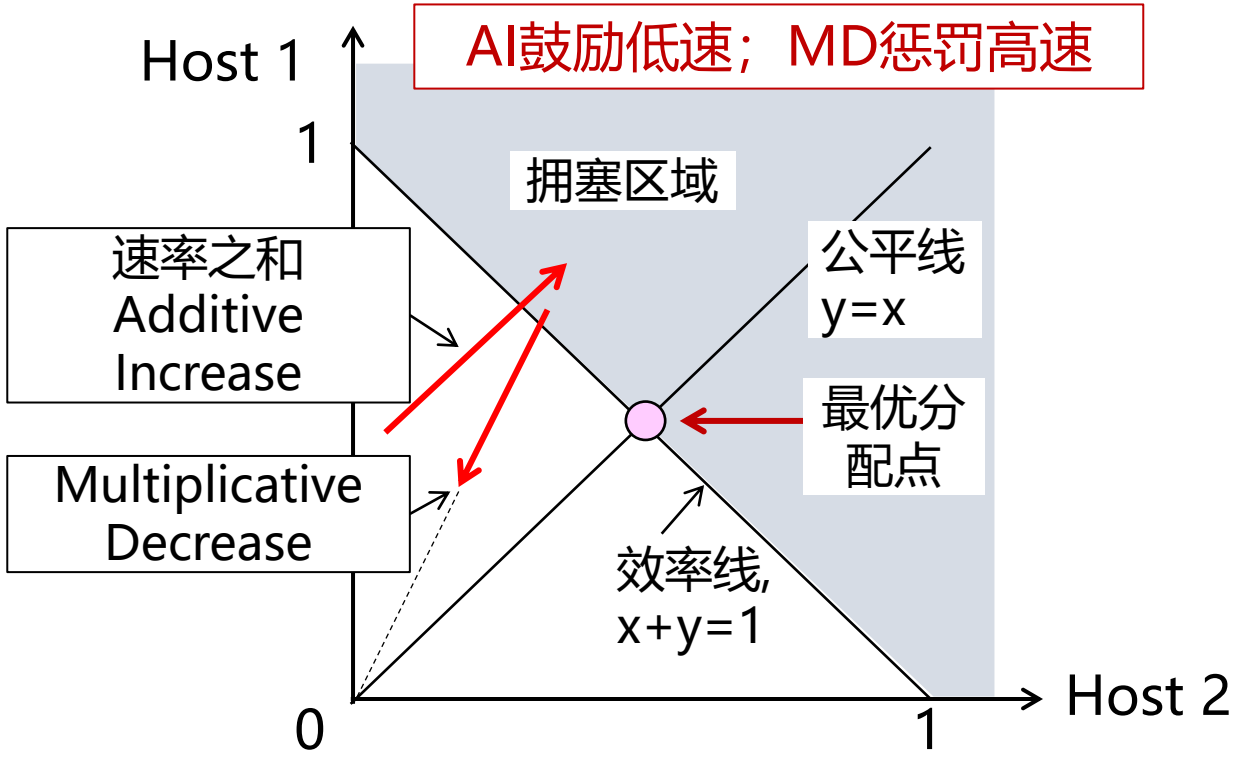
- 主机1和主机2共享瓶颈
  - 主机间并不沟通，如何实现带宽平分的公平性?
- 路由器向二者提供反馈
  - 告知主机，网络是否出现拥塞
- 如何探索并快速规避拥塞
  - 随时间产生每个主机速率的“锯齿(sawtooth)”模式
- 收敛于高效且公平的分配
  - 适用于更一般的拓扑
  - 只需要来自网络的反馈

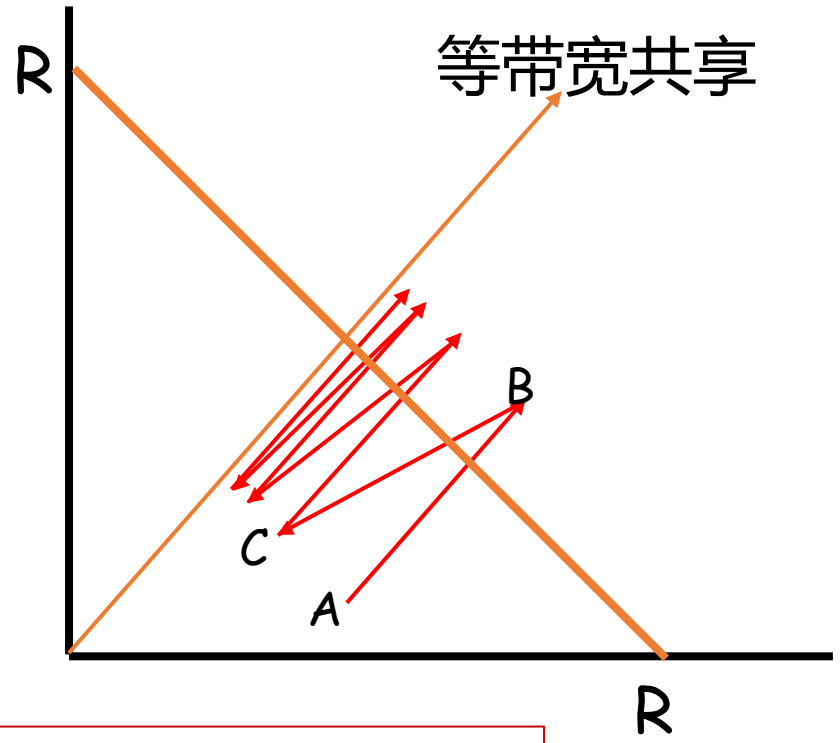




为什么TCP是公平的——AIMD

- AI和MD调整流量的分配
  - 加性增：连接1和连接2按照相同速率增大各自的拥塞窗口，得到斜率为1的直线
  - 乘性减：连接1和连接2将各自的拥塞窗口减半





其他增/减控制法则做不到（如MIAD，MIMD，MIAD）

TCP Tahoe/Reno

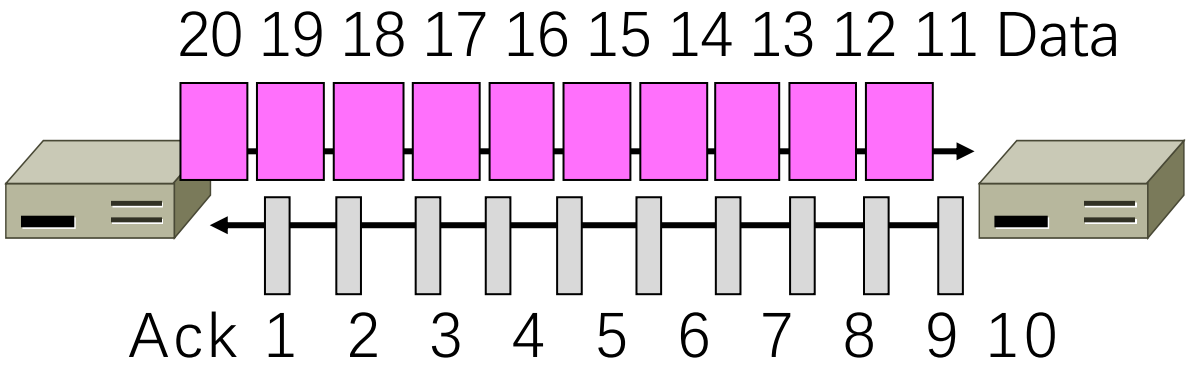
- 广泛使用的TCP拥塞控制算法
  - 在不更改路由器（甚至接收器）的情况下避免拥塞崩溃
  - 修复超时：在流控窗口上引入拥塞窗口（cwnd），以限制队列/丢失
  - TCP Tahoe / Reno使用丢包作为网络反馈信号，调整cwnd来实现AIMD
- TCP的行为

端网协同

  - 确认时钟（ack clocking）
  - 自适应超时（Adaptive timeout）　（均值和方差）
  - 慢启动（Slow-start）
  - 快重传（Fast Retransmission）
  - 快恢复（Fast Recovery）

滑动窗口的ACK时钟

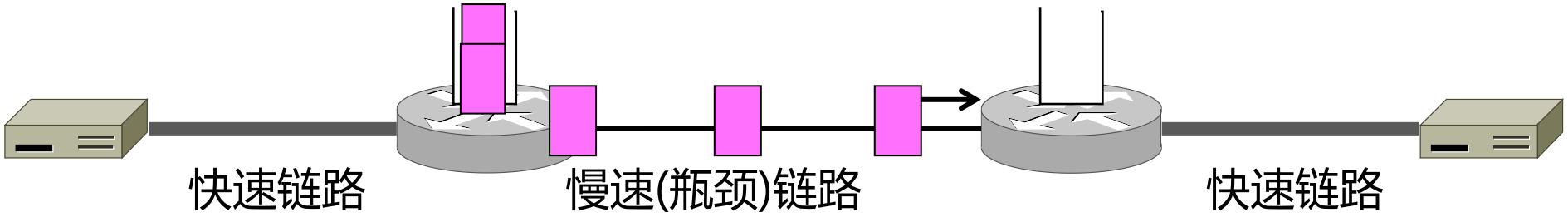
- 如何避免突发，使得流量尽量平稳?
- 滑动窗口的自计时行为以及TCP如何使用它
  - 确认时钟 “ACK clock”
- 每个按序的确认都会推动滑动窗口，并让新的段进入网络
  - 确认数据段的“时钟”



确认时钟的好处

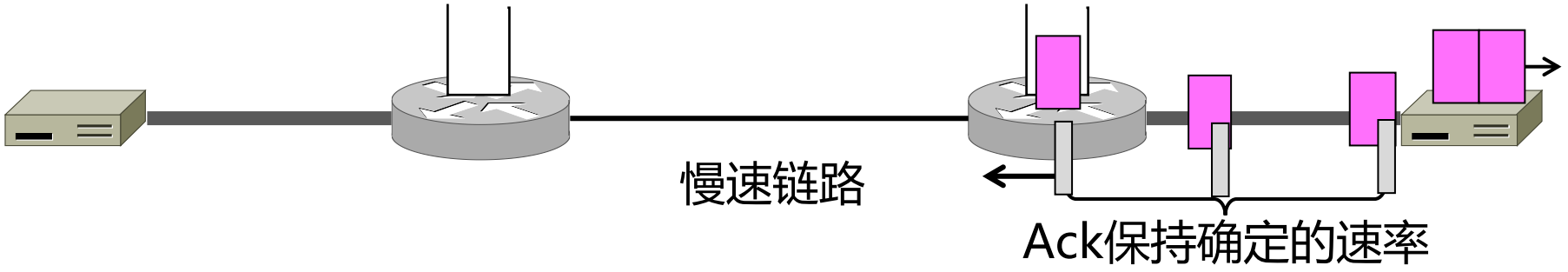
- 考虑当发送端向网络中注入大量段时会发生什么情况

- 段被缓冲并在慢速链接上分散

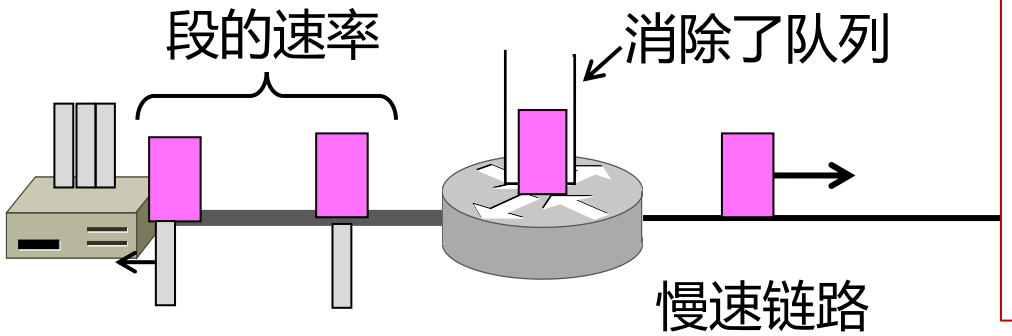


确认时钟的好处

- ACK保持确定的速率传回发送端



- 发送端用该速率做为新段的发送速率
  - 在瓶颈链路的发送就不会产生队列了!



- 帮助网络以低水平丢失和延迟运行
- 网络使数据段的突发变得平稳
- 确认时钟将这种平稳的时序传递回发送端
- 后续数据段不会突发发送，因此不会出现网络队列
- TCP仅发送小量突发段，以使网络保持流量平稳

TCP计时器管理

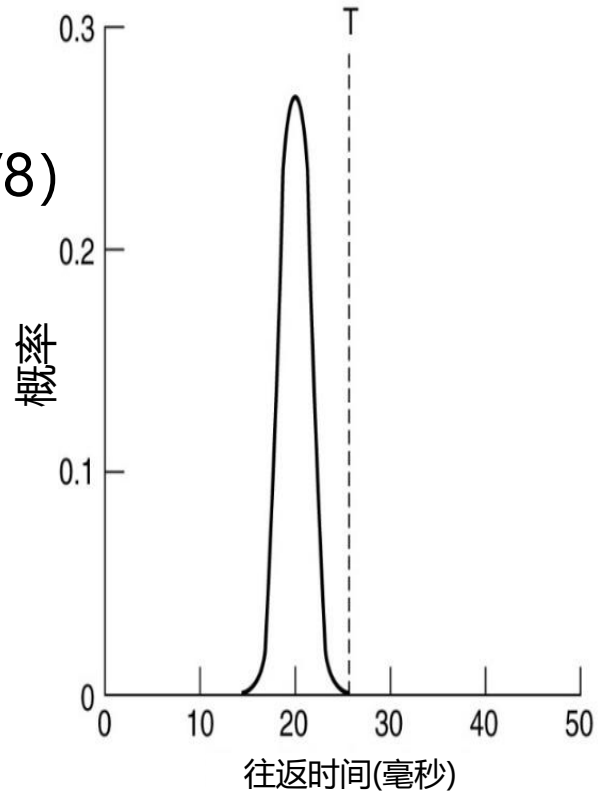
- 重传定时器（时长大小？）
  - 平滑往返时间SRTT

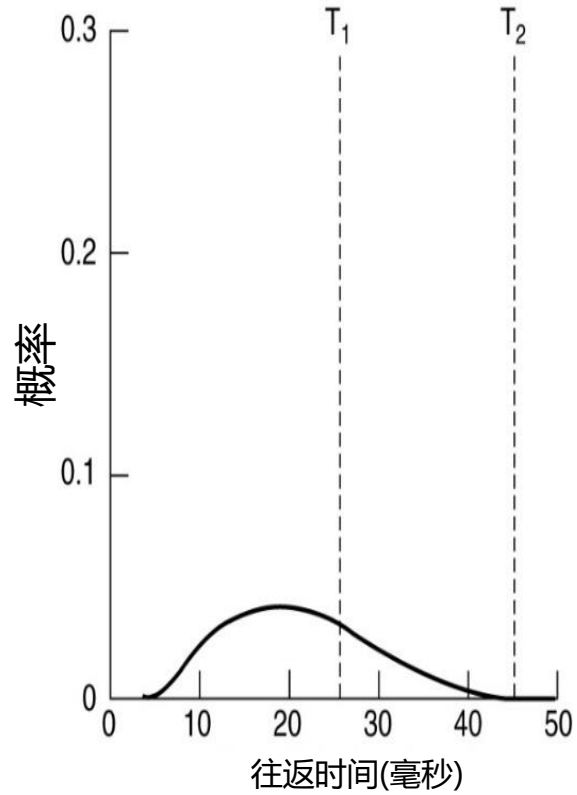
    $SRTT_{新} = \alpha SRTT_{老} + (1-\alpha)R$ （一般$\alpha=7/8$）

    R：本次测量确认所花时间
  - 定时器如何适应RTT快速变化?
    - 原来 = 2×RTT
    - 现在：往返时间变化RTTVAR

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

      超时值: RTO = SRTT + 4×RTTVAR
- 持续计时器：探询接收方，返回窗口大小
- 保活计时器：探测对方是否在线（利弊）



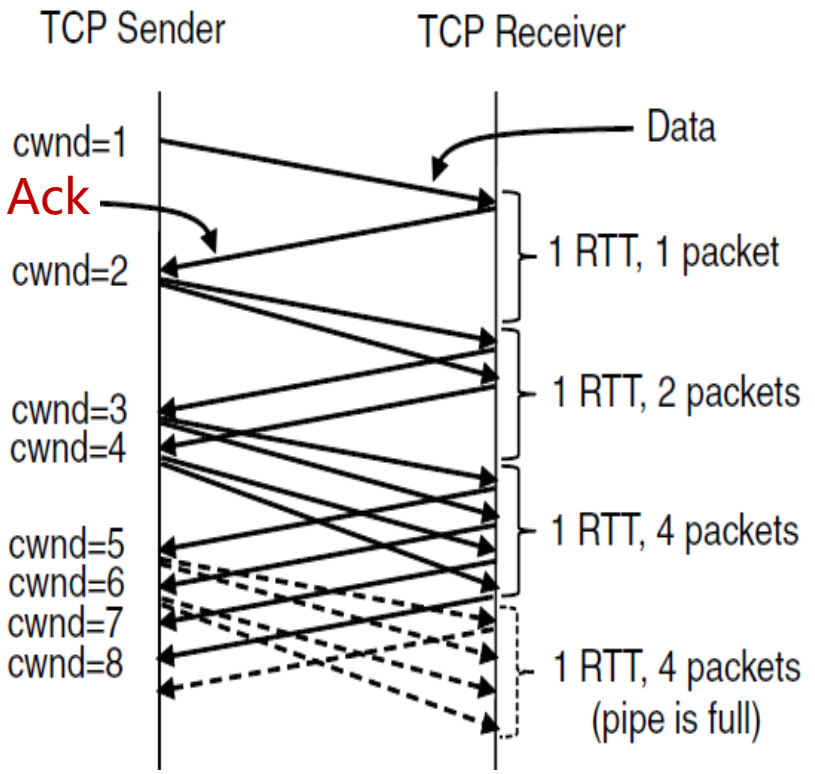


(a)数据链路层的确认到达时间概率密度 (b)TCP的确认到达时间的概率密度

时延v.s.抖动

TCP拥塞控制

- 拥塞控制的目标
  - 速率大致为 cwnd/RTT
  - 快速探测进入高速阶段；尽量长时间维持高速阶段
- 拥塞控制的两个阶段
  - 拥塞窗口cwnd慢启动过程；cwnd线性增加过程
- 慢速启动
  - cwnd初始值：1个段，后来据经验改为4个段
  - 发送端
    - 按初始值发送段
    - 每收到一个未超时确认，cwnd++
  - 慢启动：拥塞窗口对RTT呈指数增长
    - 拥塞窗口：1->2->4->8...

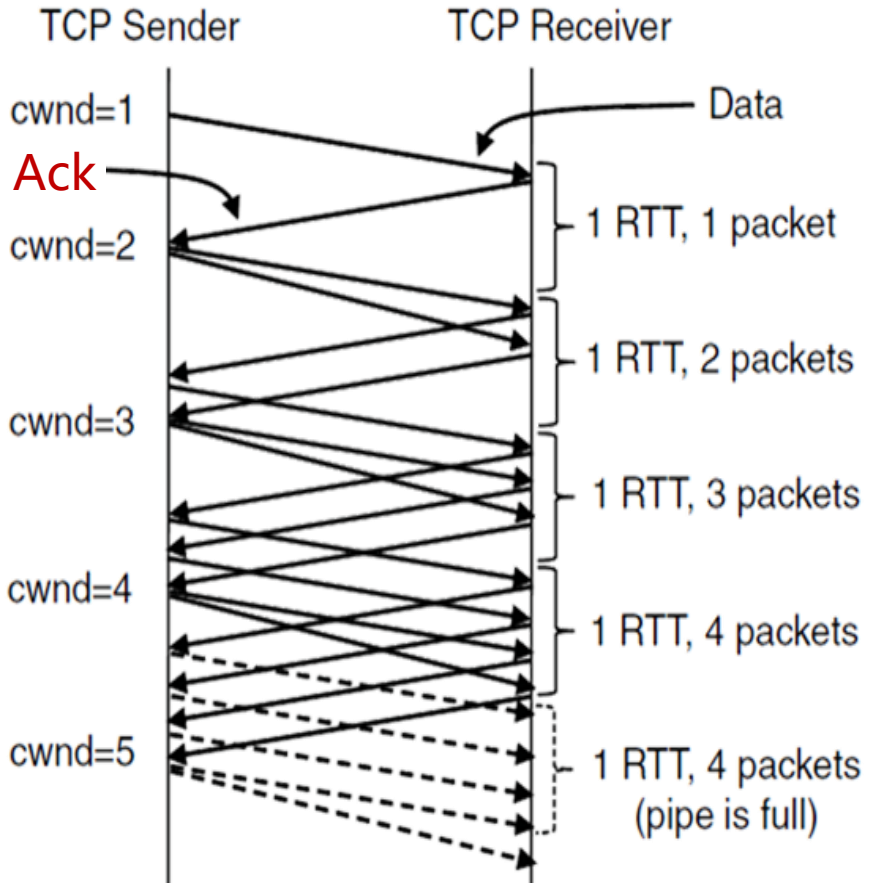


慢速启动增长

TCP拥塞控制

- 线性增加过程
  - 每过一个RTT，cwnd++
- 慢速启动阶段最大阈值
  - 慢启动阈值初始为流量控制窗口大小
  - 发生确认超时后，阈值降为当前拥塞窗口的一半，然后重新启动
  - 一旦慢启动超过阈值，TCP从慢启动切换到线性增加状态

线性增加过程
窗口增速很慢：好/坏?



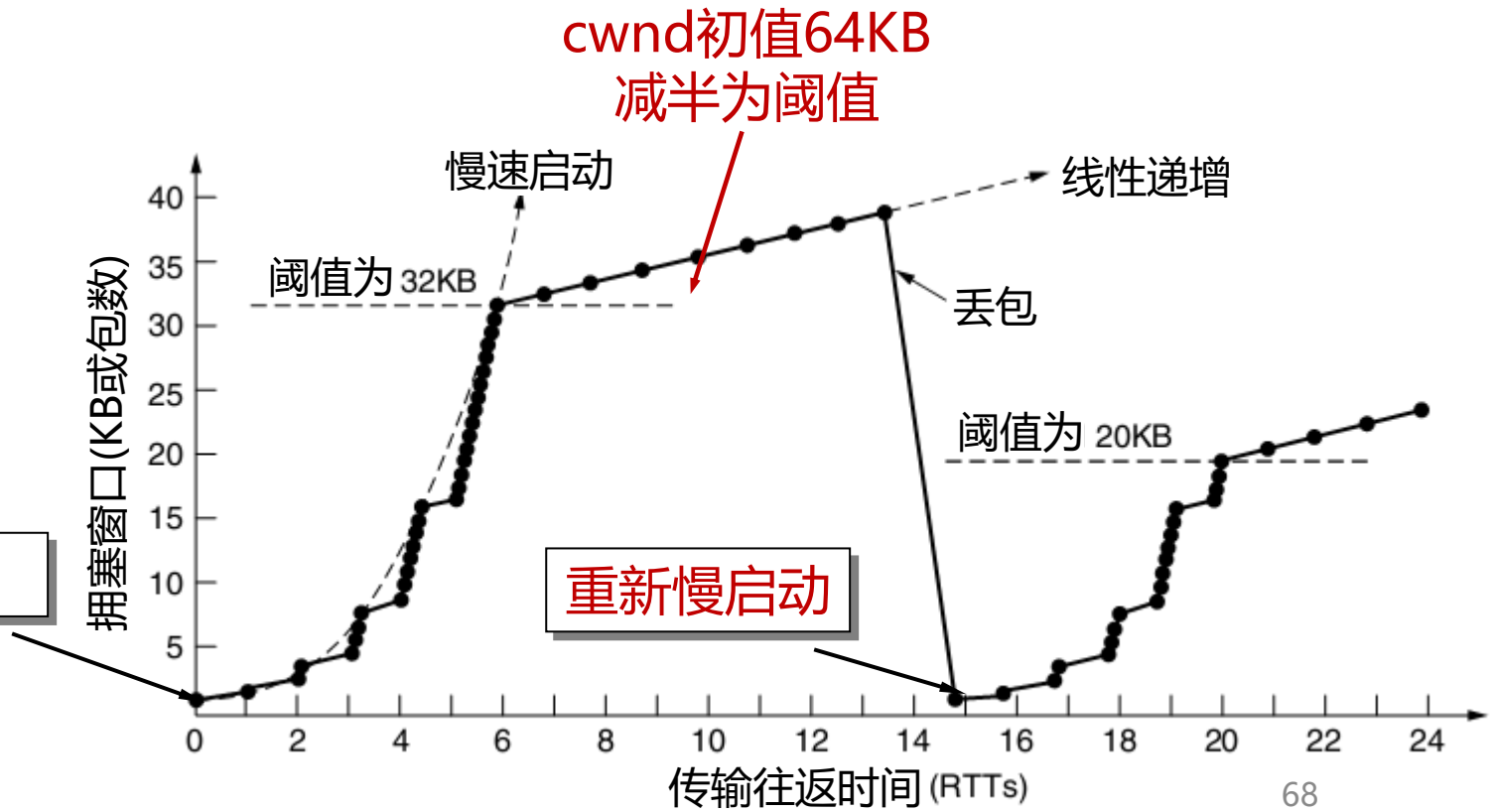
线性增长

TCP拥塞控制

- TCP Tahoe
  - 1988年发布的4.2BSD TCP Tahoe
  - 最大段长：1KB
  - 拥塞窗口初值：64KB
  - 阈值为窗口的一半：32KB
  - 窗口40KB时发生丢失
  - 阈值更新为20KB

慢启动慢吗?
能否少降一些?

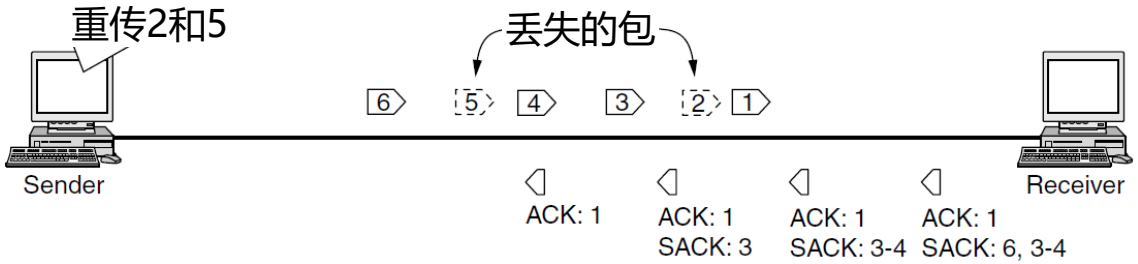
慢启动



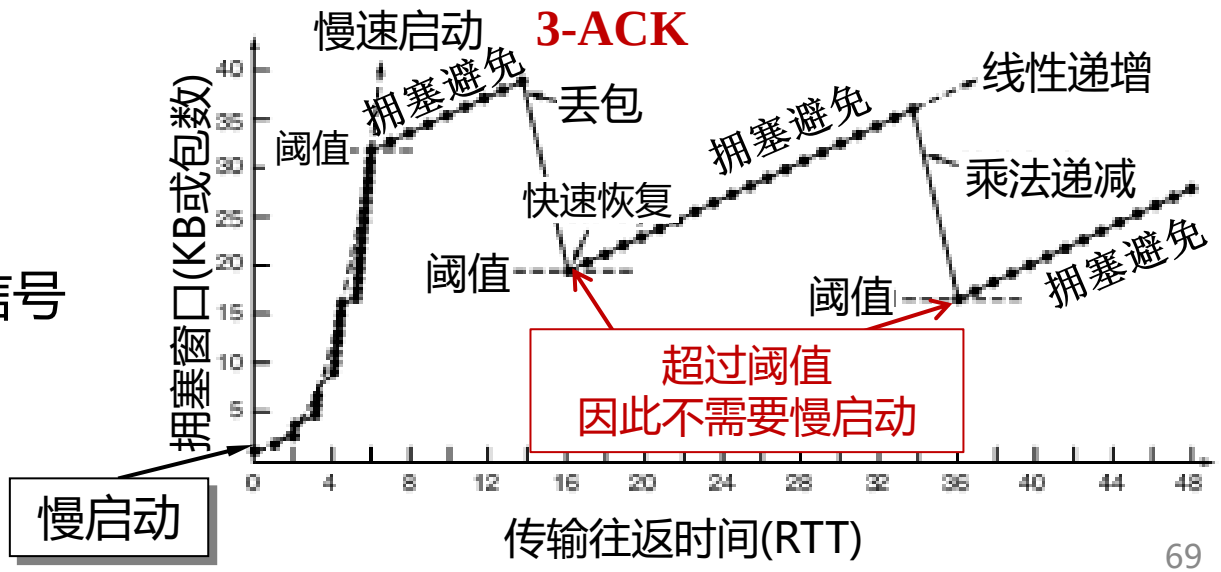
TCP拥塞控制

- TCP Reno的快速恢复
  - 个别丢包，别降低太狠
  - 三个重复确认（如三次确认1）
  - 窗口降到阈值（而不是降为1）
  - 快速重传：重传丢失报文
- TCP的后续发展
  - 选择确认SACK
    - 列出3个已收到的字节范围
  - 丢包前：显示拥塞通知ECN
    - 路由器在分组中设置ECN拥塞信号
    - 接收端设置ECN Echo标志位
    - 发送端减速

端网协同?



如果只有ACK，我们无法知道丢失了2和5

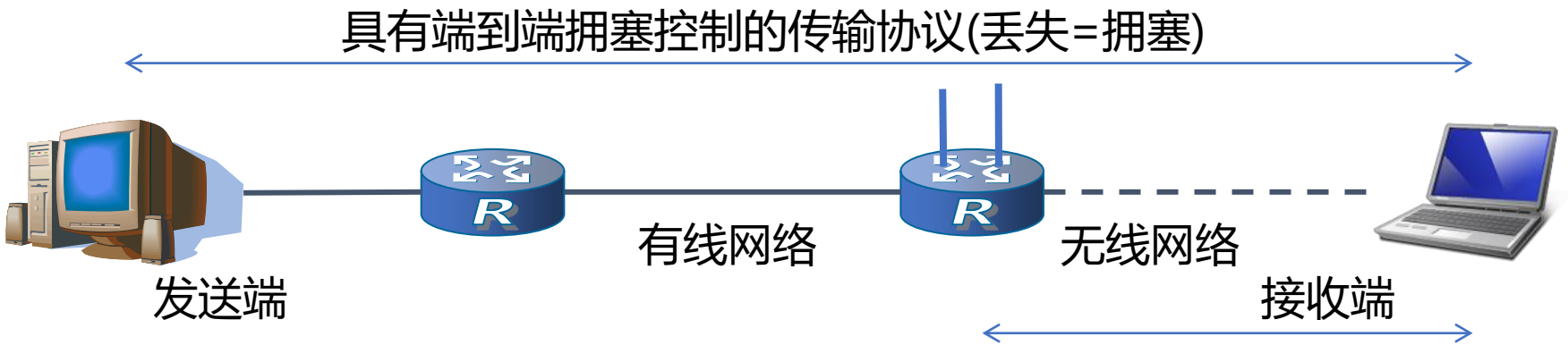


无线问题

- 拥塞与传输错误
  - TCP连接丢失率：1%，极限10%（流量减低至0）
  - 802.11正常丢失率：10%
- 屏蔽策略：在链路层重传
  - 无线链路重传对传输层不可见
  - 链路层重传微秒到毫秒；传输层丢失判断在毫秒到秒

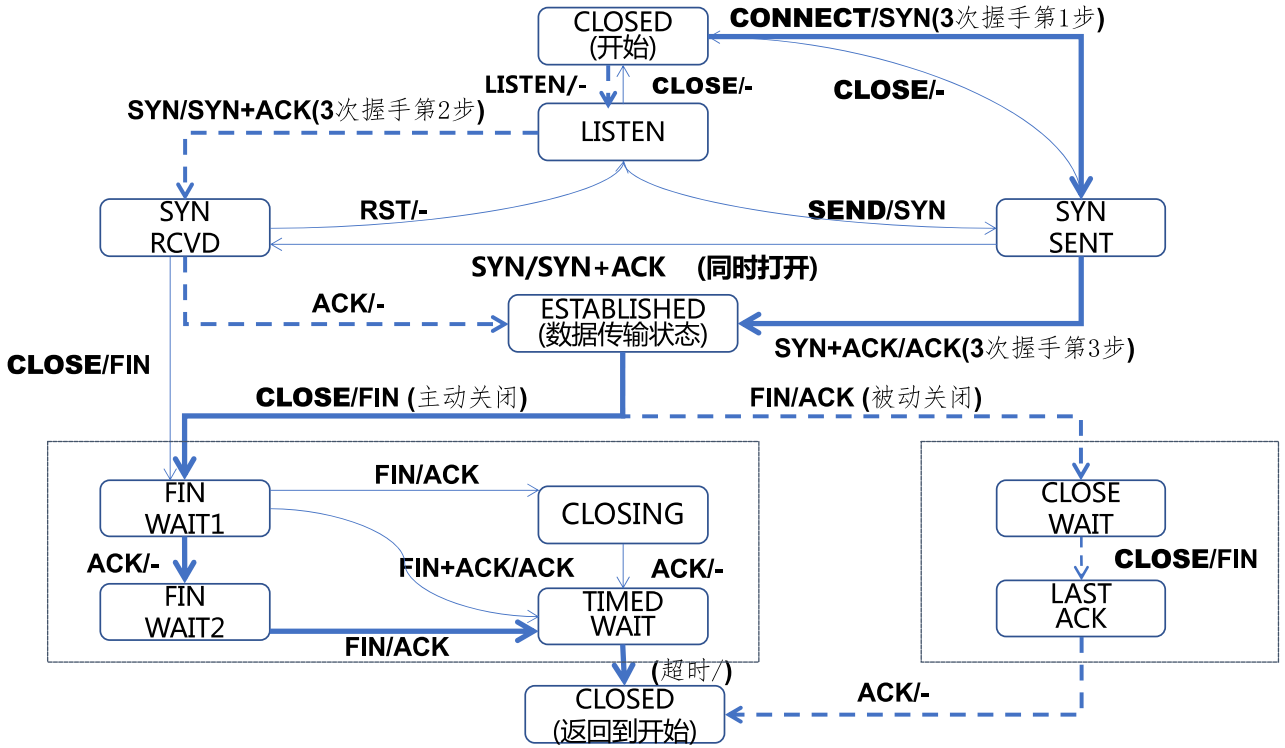
加速重传还是减速重传?

链路层没有重传呢?



传输协议TCP-小结

- TCP提供的服务
  - 面向连接的、可靠的、端到端的、基于字节流的传输协议
- 字节流：流入或流出进程的字节序列
  - TCP保证字节流的可靠交付（保序、不丢失）
- TCP连接
  - 连接建立：三次握手
  - 连接释放：四次握手
  - 连接管理的有限自动机
- TCP的确认与重传机制
  - 确认时钟
  - 基于ACK的滑动窗口
- TCP拥塞控制
  - 拥塞崩溃及其解决(Tahoe, Reno)
  - AIMD机制的公平性



总结

- 传输服务
  - 提供进程到进程的服务
  - Berkeley 套接字
- 传输协议的要素
  - 寻址、连接建立与释放、差错控制与流量控制、多路复用等
- 拥塞控制
  - 引起拥塞的原因：输入>输出持续存在，大量数据在转出前被丢弃
  - 解决方法：使发送方以合适的速度发送
- UDP协议：无连接、不可靠但简单、快速的传输层协议
  - 提供了：进程到进程的服务；复用和分用；差错检测
  - 未提供：拥塞控制；可靠传输

最最最核心的TCP

作业

- 《Computer Networks-5th Edition》章节末习题
  - CHAPTER 6：2（服务原语），9（建立连接），12（带宽分配），14（拥塞控制），16（UDP），19（UDP/TCP）
- 截止时间：下下周三晚11:59，提交网络学堂

致谢社区本章贡献者

华蓓

中国科学技术大学

李沁

安徽工业大学

崔勇

清华大学

参考教材：

[1] Jim Kurose, Keith Ross,《计算机网络：自顶向下方法》(第7版)，机械工业出版社，2018.6
[2] Tanenbaum, Wetherall,《计算机网络》（第5版），清华大学出版社，2012.3
[3] 谢希仁，《计算机网络》（第7版），电子工业出版社，2017.10
[4] 徐敬东，张建忠，《计算机网络》（第3版），清华大学出版社，2013.6

特别致谢：本章课件（6.1-6.7）主要改编自《计算机网络：自顶向下方法》公开的课件