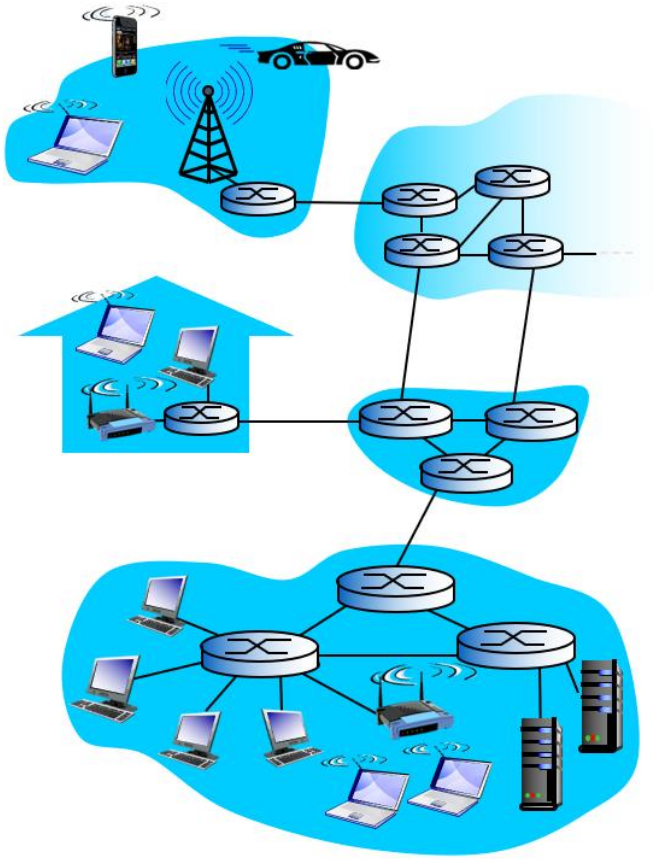
计算机网络

计算机与信息学院
人 工 智 能 学 院

- 网络层：核心IP协议，实现网络中主机之间的通信



本章内容

传输层的功能

1）进程之间的通信

2）可靠传输

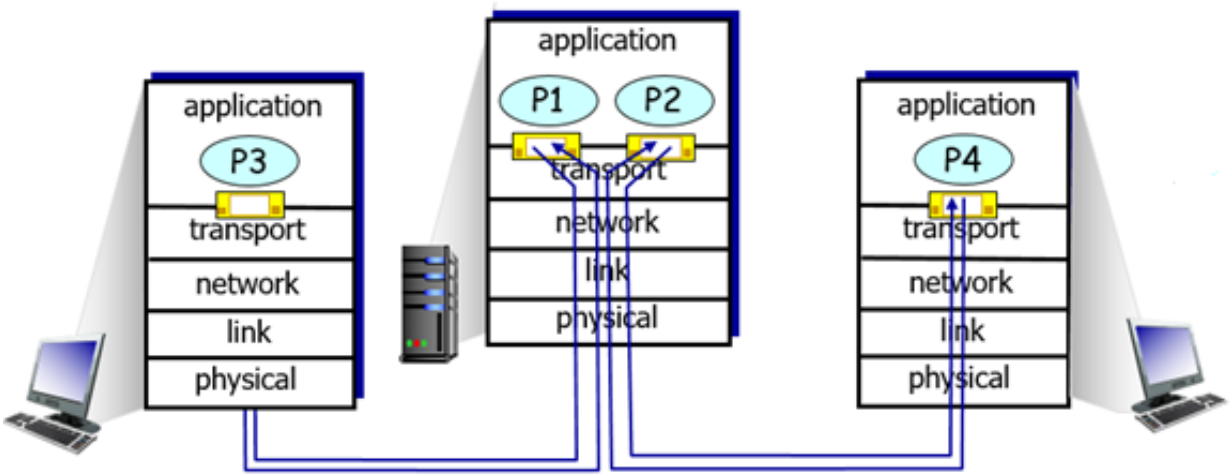
3）流量控制

4）拥塞控制

TCP：传输控制协议，实现功能1-4

UDP：用户数据报协议，实现功能1

不同进程之间的通信，需要对进程进行标识



端口（Port）：应用进程的逻辑编号（16it，0~65535）

如何标识一个通信进程？ <IP 地址， 端口号>

端口号：16位（0~65535）

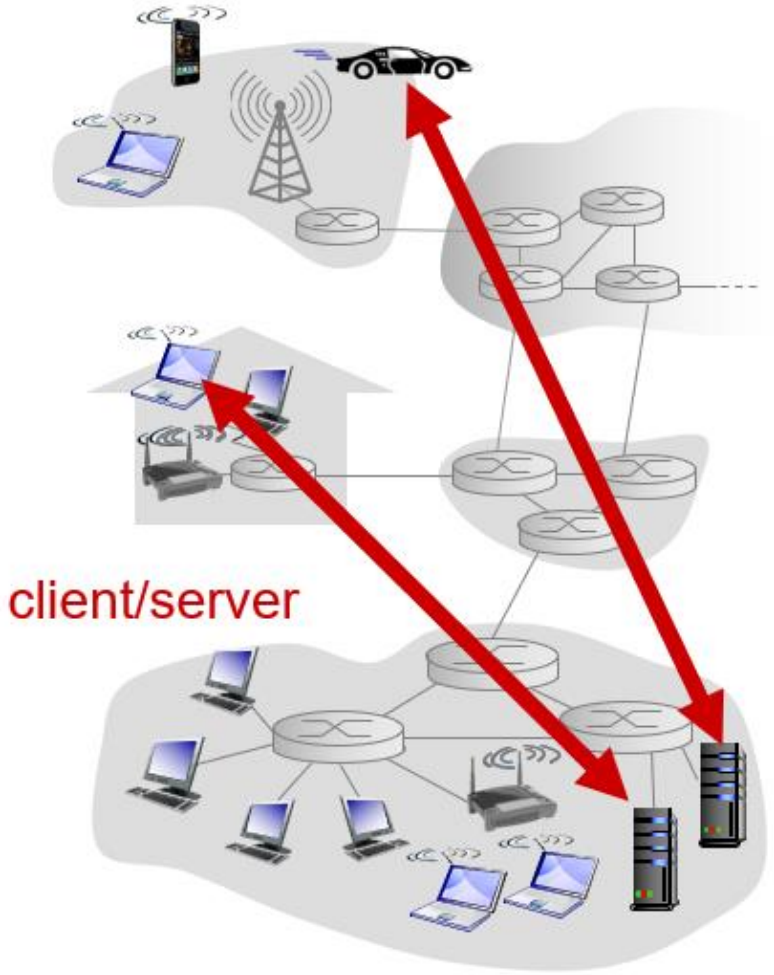
- 服务端使用的端口号

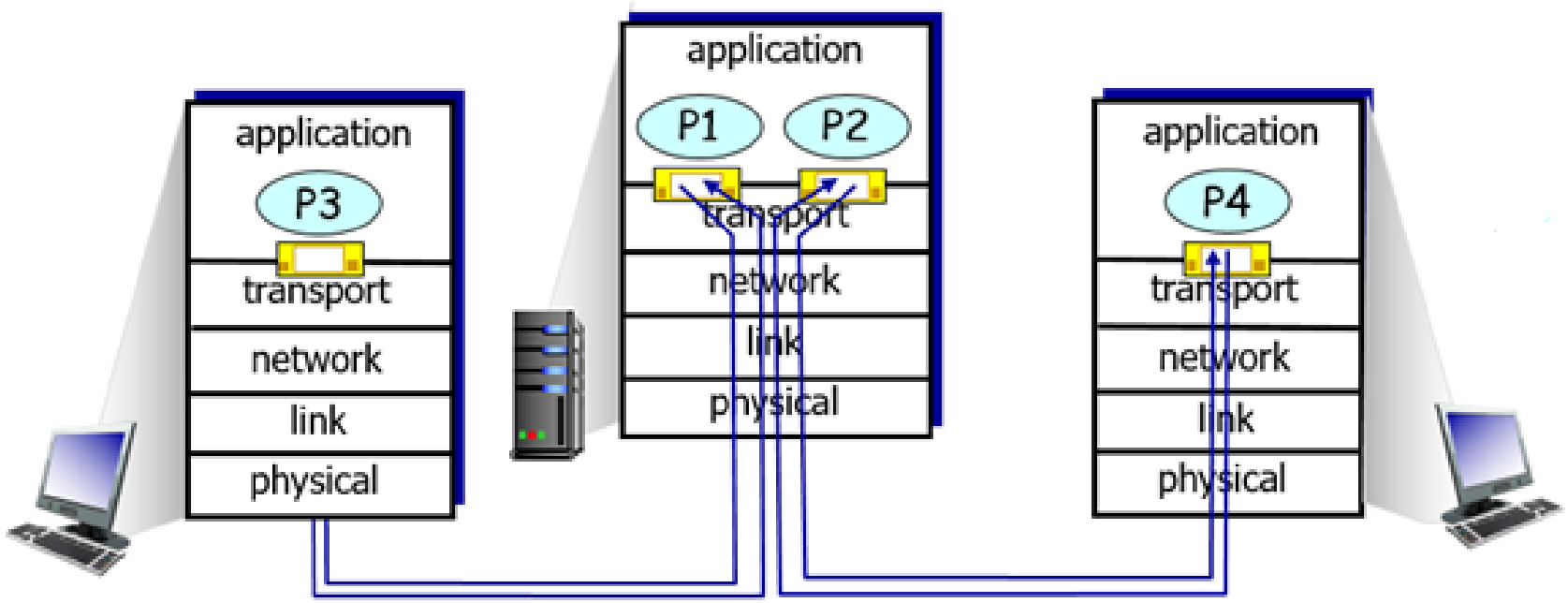
  熟知端口号：（0~1023）

  登记端口号：（1024~49151）

- 客户端使用的端口号

  临时端口号：(49152~65535)





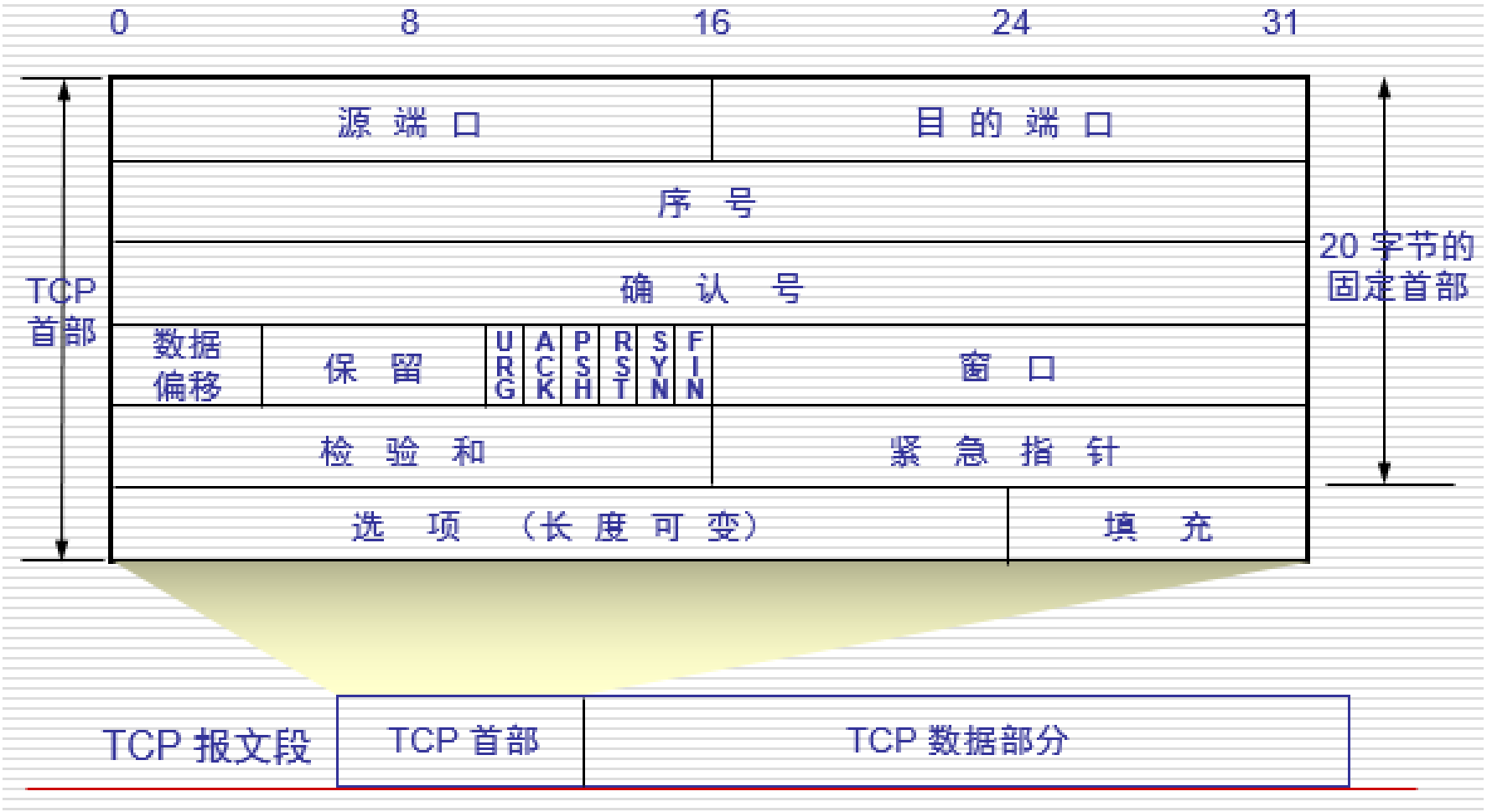
传输层：实现进程的多路复用（ Multiplexing ）和分用（ demultiplexing ）

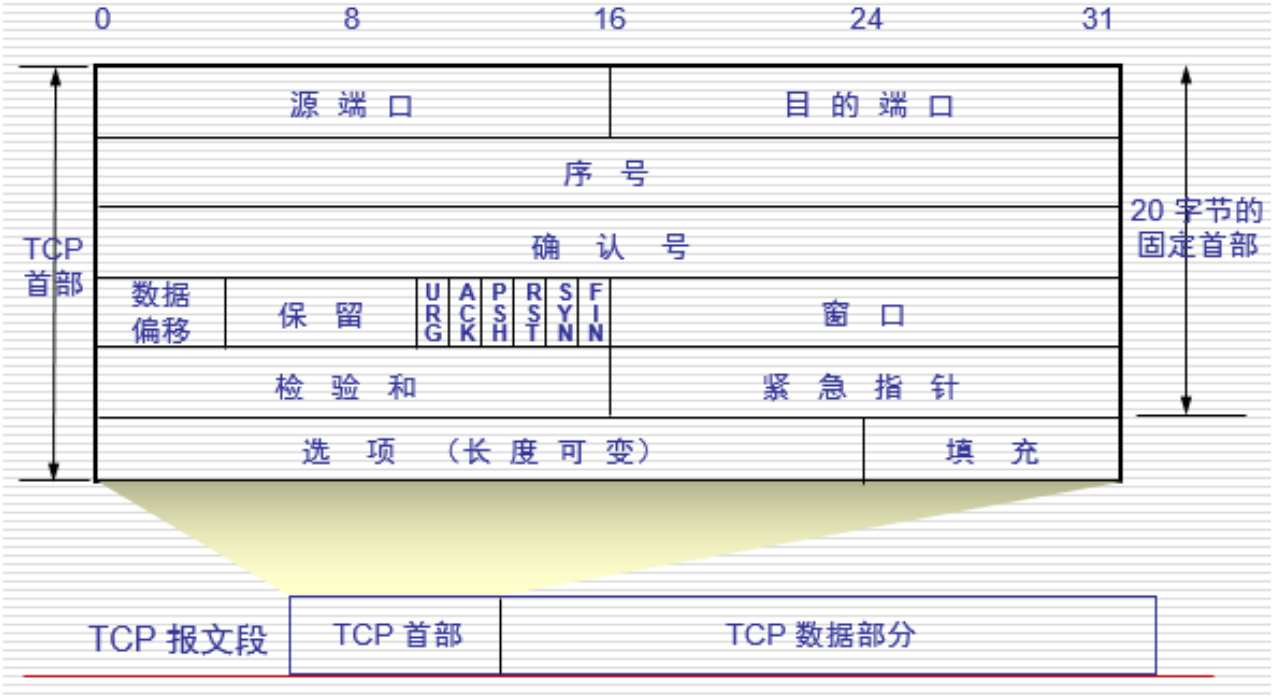
传输层：实现了端(通信进程)到端的通信

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

1. TCP报文格式
2. TCP可靠传输
3. TCP连接管理
4. TCP流量控制
5. TCP拥塞控制

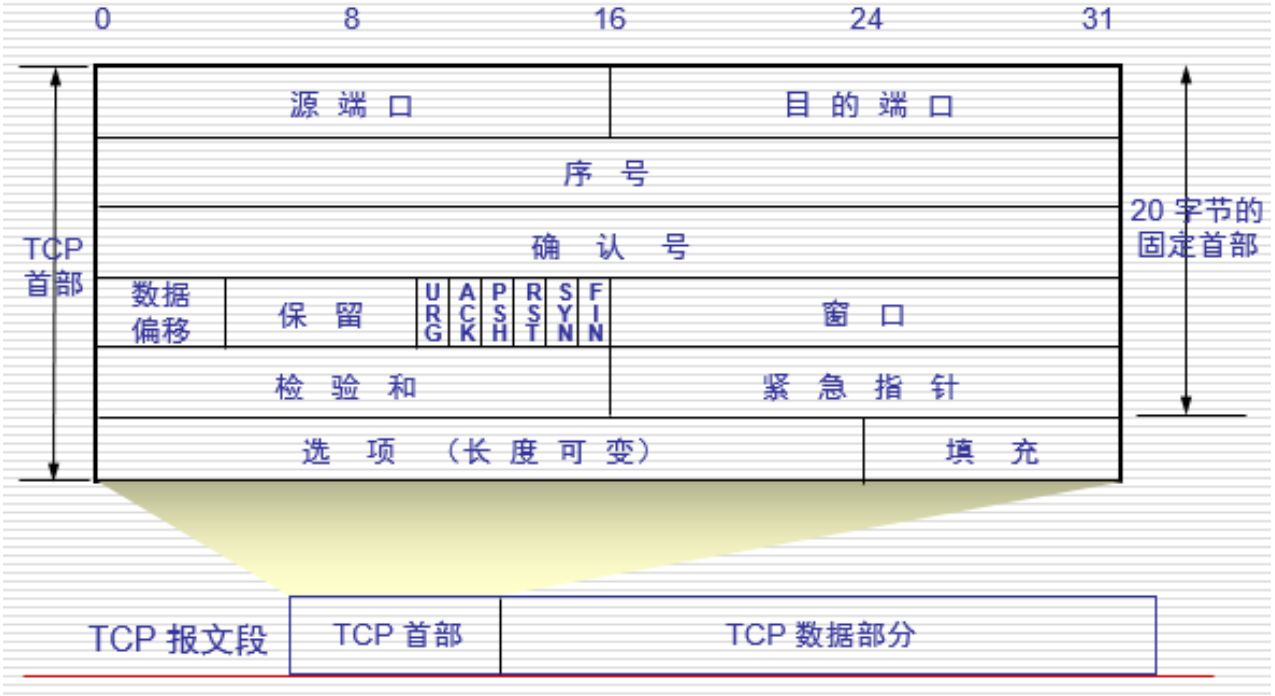
1、TCP报文格式


0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
数据偏移
保 留
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （长 度 可 变）
填 充
TCP 首部
20 字节的固定首部
TCP 报文段
TCP 首部
TCP 数据部分



- **发送方**：应用层需要快速传递关键数据（如交互指令、实时消息）时，PSH置1

- **接收方**：PSH=1 时，无论缓冲区是否已满，必须立即将缓冲区中的所有数据递交给应用层



校验和： TCP首部+TCP数据+伪首部（12字节）

伪首部： IP首部的源/目的IP地址-8字节，协议类型（TCP：6 、UDP：17）1字节

TCP报文段长度-2字节，保留-1字节

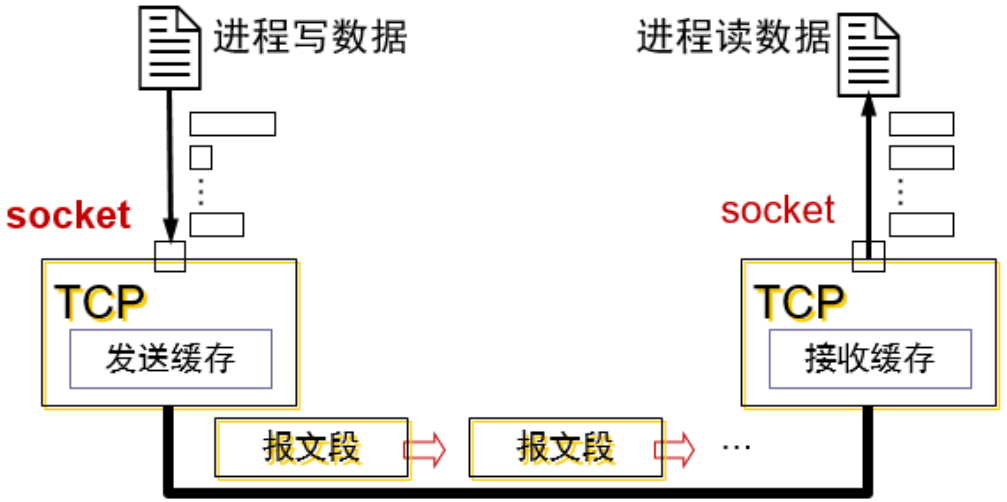
2、可靠传输

在不可靠信道上设计可靠传输协议：复杂

1）停-等协议

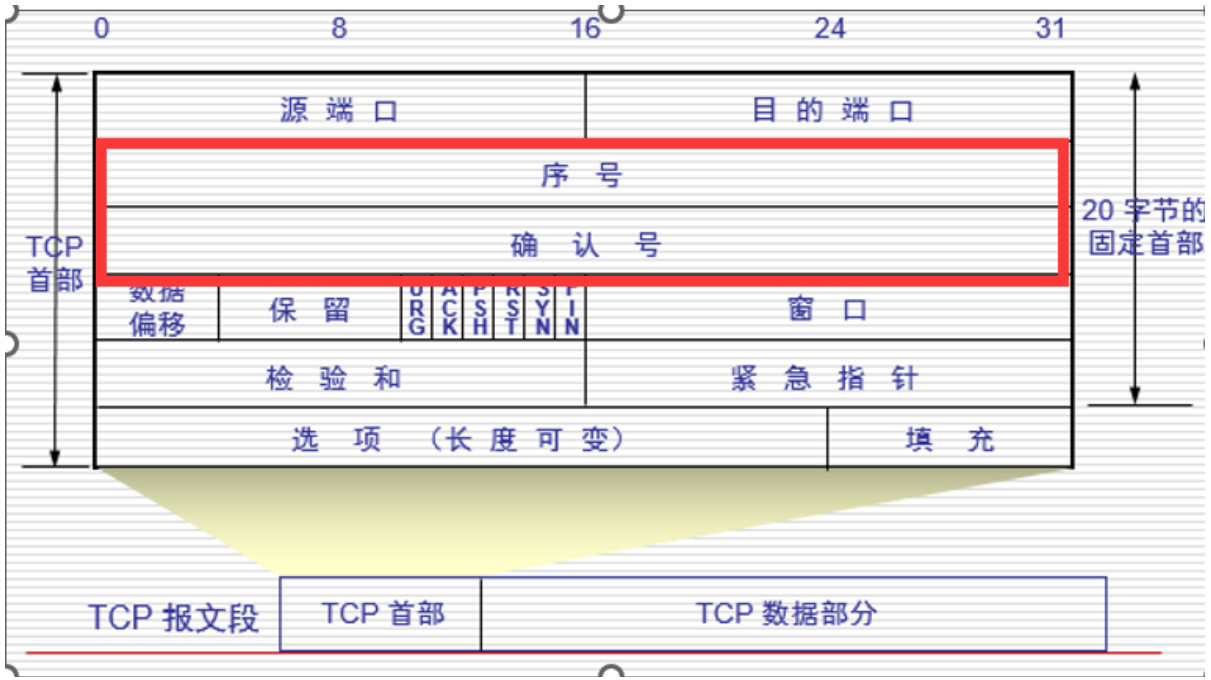
2）回退N

3）选择重传

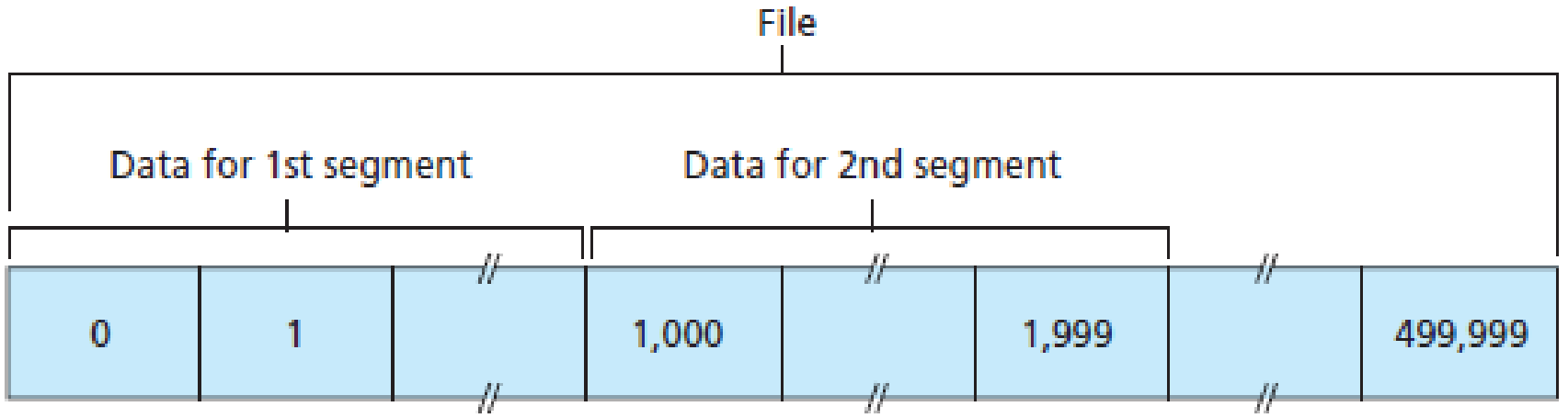


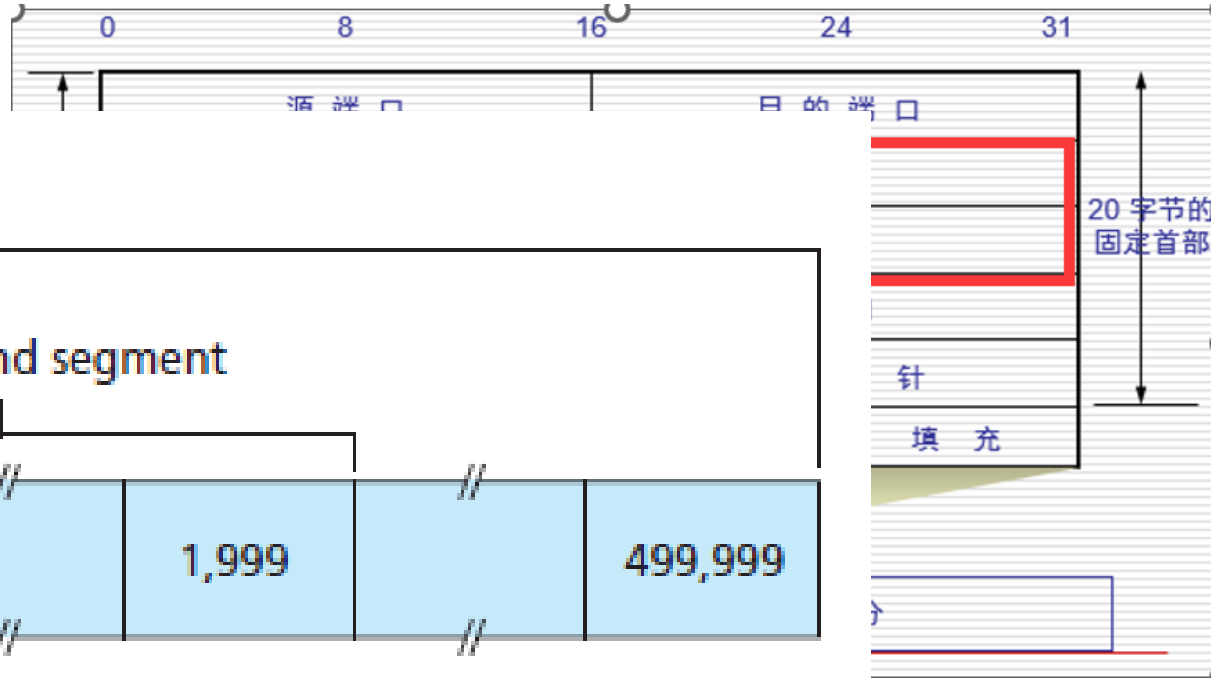
- TCP把上层交付的数据看成字节流，不保证数据块之间的对应关系
- 接收方收到的字节流：无差错、不重复、顺序一致

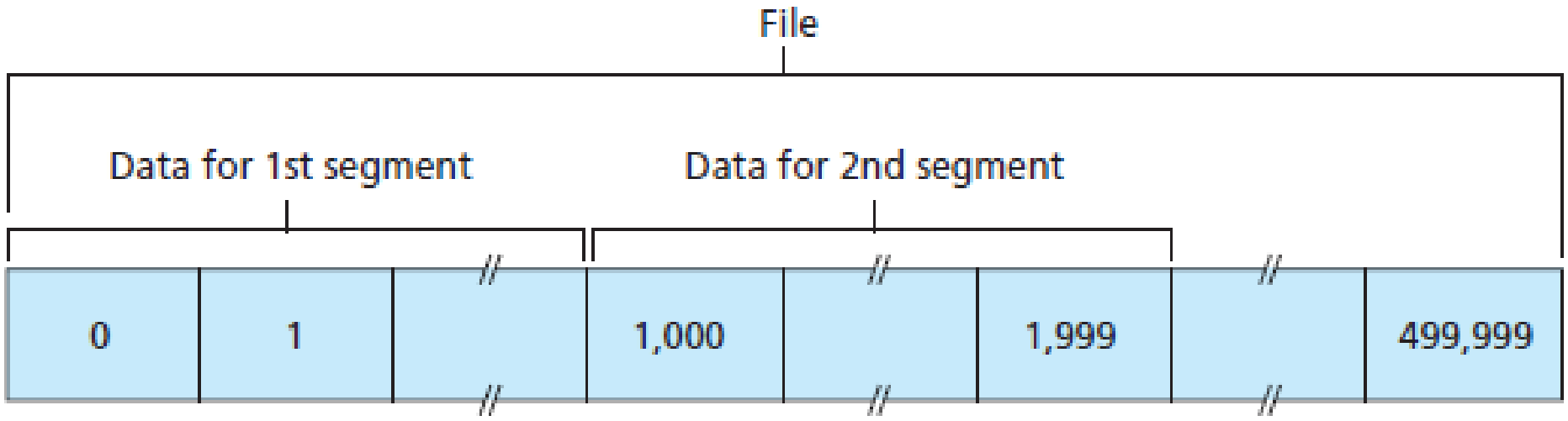
可靠传输


0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
TCP 首部
数据偏移
保 留
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （长 度 可 变）
填 充
20 字节的固定首部
TCP 报文段
TCP 首部
TCP 数据部分

- 序号：发


File
Data for 1st segment
Data for 2nd segment
0
1
1,000
1,999
499,999





确认号：接收方期望收到的下一个字节的编号，同时对该编号之前数据的确认

确认号：接收方期望收到的下一个字节的编号，同时对该编号之前数据的确认（累积确认）

1）A→B，B收到1st报文段0~999，则B返回确认号？

2）A →B，B收到1st报文段0~999，2nd报文段1000~1999，4th报文段3000~3999，则B返回确认号？

- TCP可靠传输
  - ✓ 接收方按序接收，累积确认
  - ✓ 接收方如何处理乱序的报文段-没有规定

——许多TCP实现，将失序的报文段缓存起来（选择确认：SACK选项字段）

- TCP发送窗口和接收窗口大小

1）停-等协议

2）回退N

3）选择重传

两个指针变量：

LastByteSent：最后一个发送的字节

LastByteAcked：最新收到的确认字节

发送窗口>1，具体多大?

1）停-等协议

2）回退N

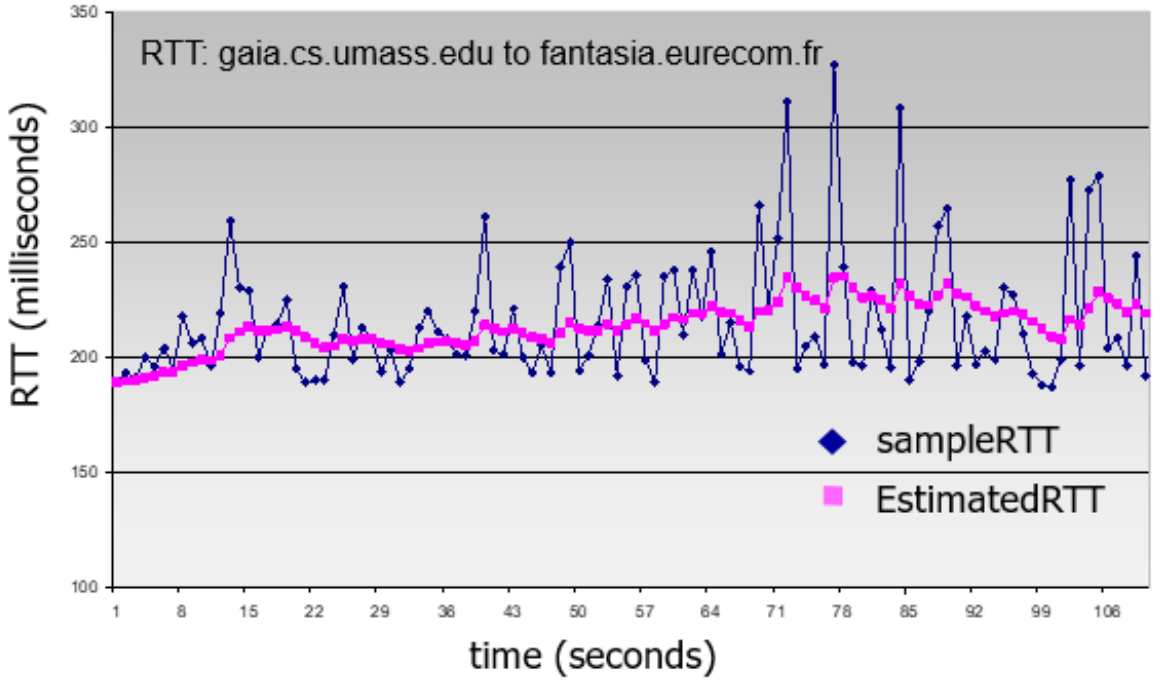
3）选择重传

- 丢失的报文段如何处理?

超时重传：发送方每发送一个报文段，启动计时器（实际实现：1个计时器），在规定时间内没有收到对报文段的确认，则重传该报文段

超时时间的设置：比RTT稍微长一点，长多少?

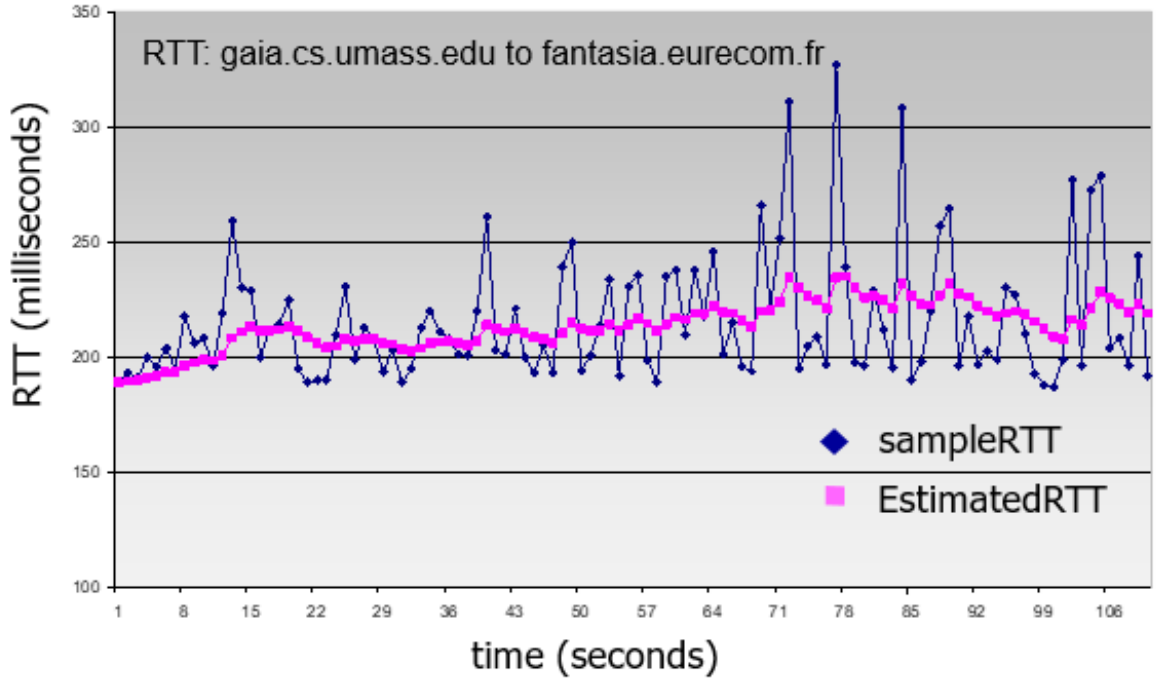
采样报文段的RTT，作为sampleRTT



EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

a的推荐值：0.125

估算偏差



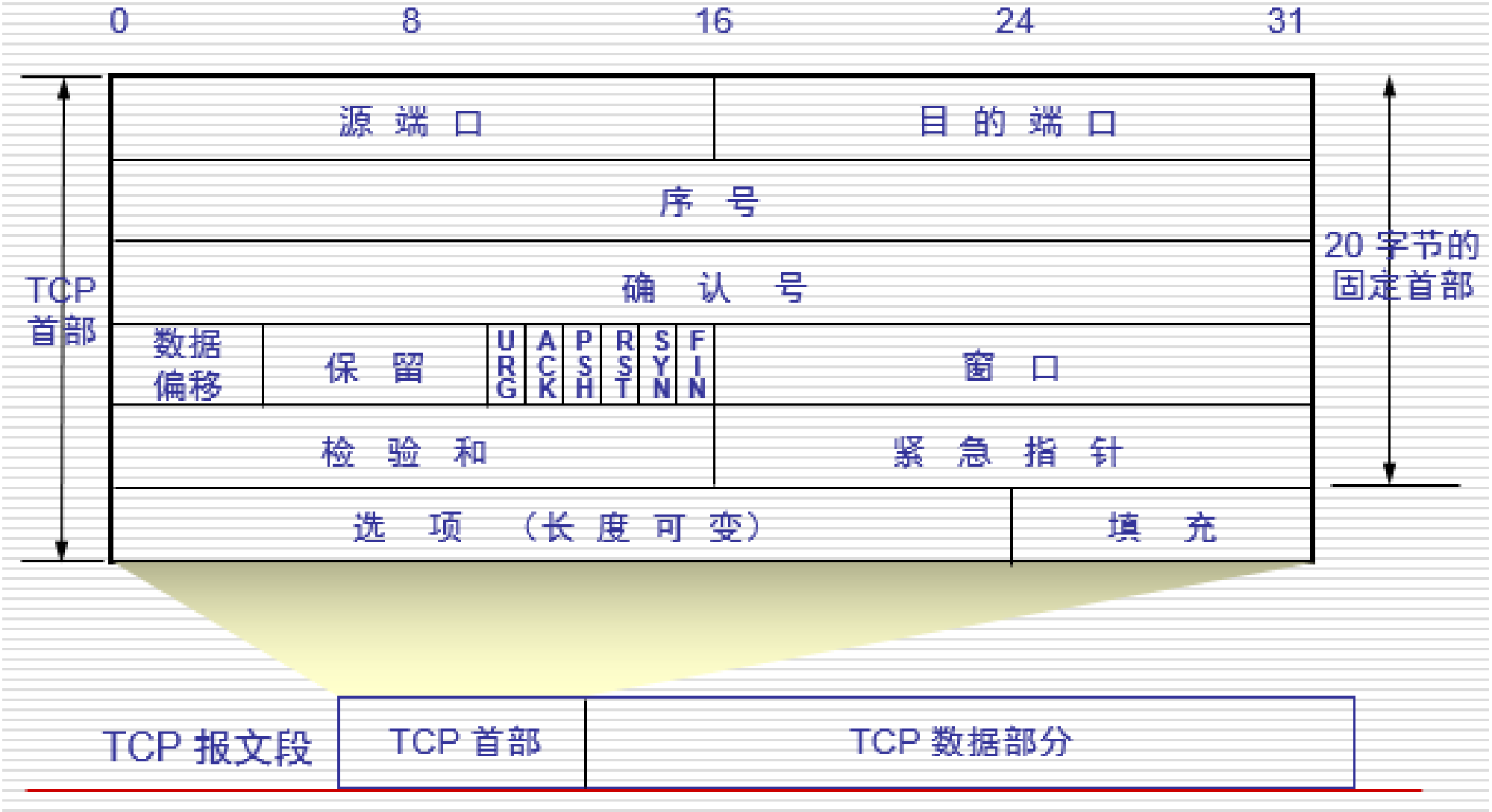
$$DevRTT = (1-\beta)*DevRTT + \beta*|SampleRTT-EstimatedRTT|$$

(typically, $\beta = 0.25$)

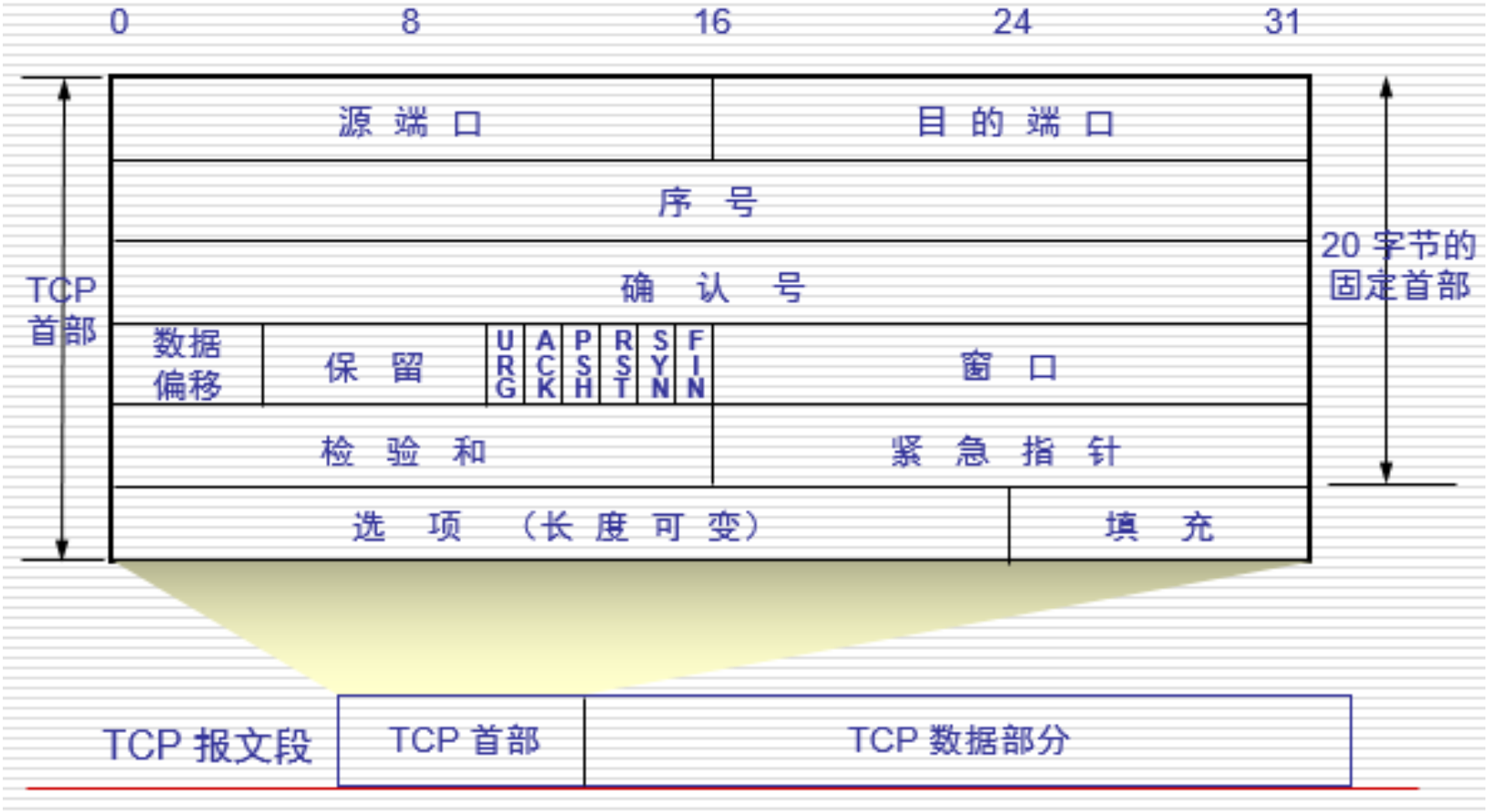
$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

estimated RTT

"safety margin"


0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
数据偏移
保 留
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （长 度 可 变）
填 充
TCP 首部
20 字节的固定首部
TCP 报文段
TCP 首部
TCP 数据部分

1. TCP报文格式
2. TCP可靠传输
3. TCP连接管理
4. TCP流量控制
5. TCP拥塞控制



连接?

收发双方在通信之前需要协商：连接—传输数据—释放连接

以太网的链路层协议：有连接/无连接?

Internet的网络层IP协议：有连接/无连接?

数据报

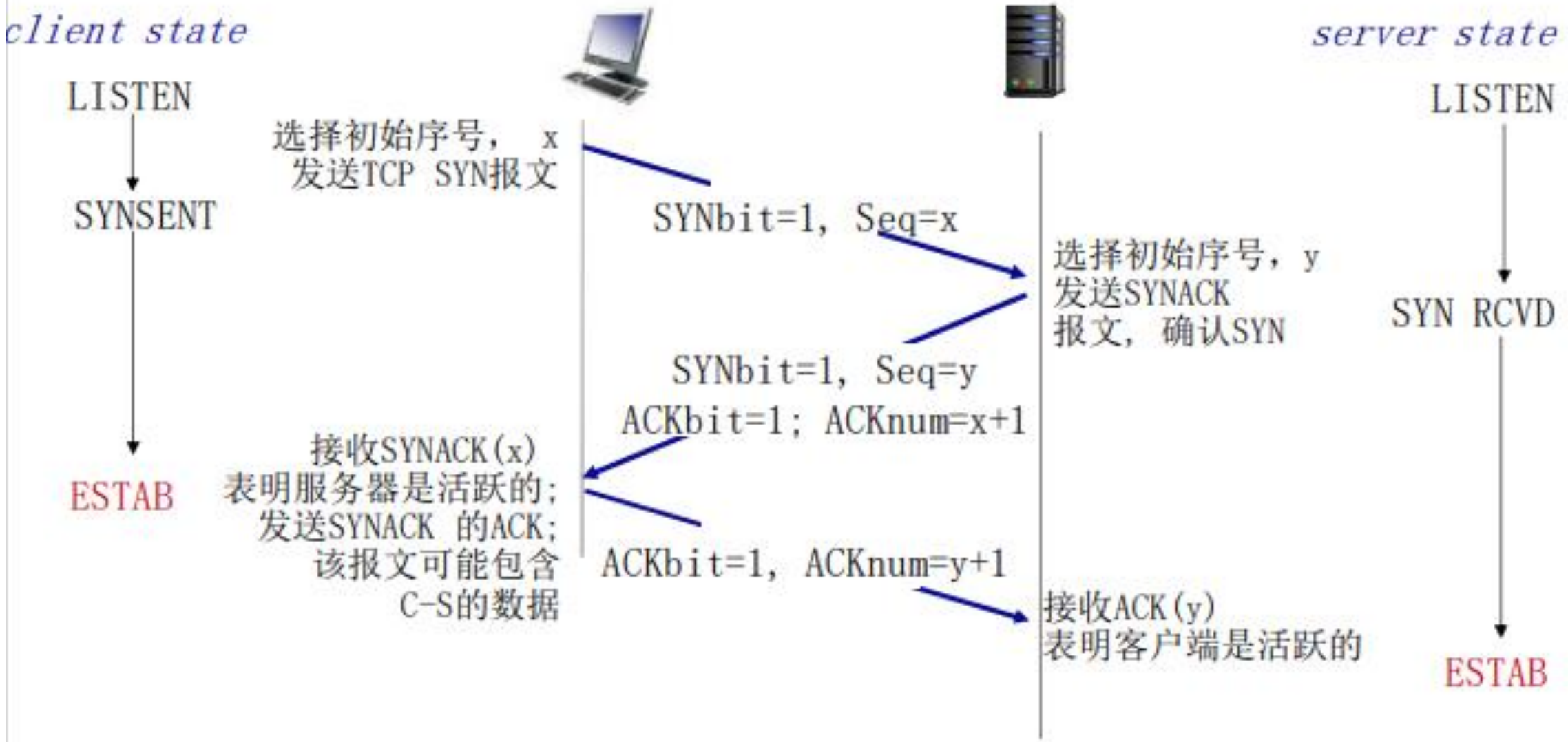
虚电路

采用TCP协议，传输数据前，两个进程必须先建立TCP连接

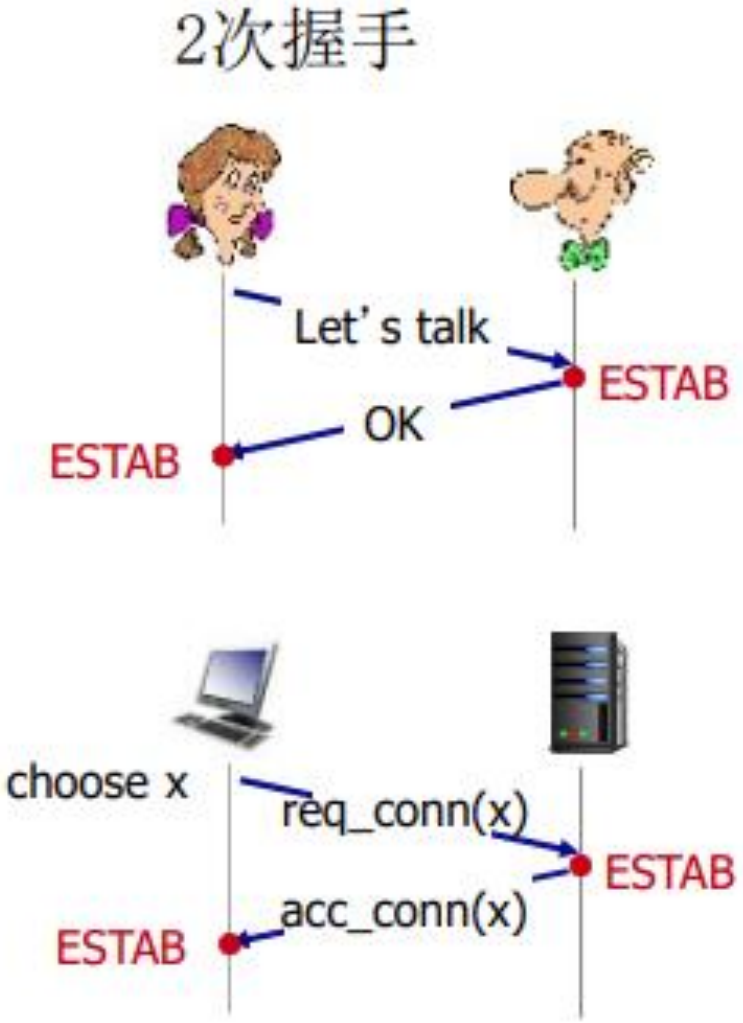
1. 确定接收方的应用进程已经做好接收准备
2. 初始化与TCP连接相关的状态变量

TCP 连接： {(IP1: port1), (IP2: port2)}

建立连接（三次握手）


client state
server state
LISTEN
LISTEN
选择初始序号， x
发送TCP SYN报文
SYNSENT
SYNbit=1, Seq=x
选择初始序号，y
发送SYNACK
报文，确认SYN
SYN RCVD
SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1
接收SYNACK(x)
表明服务器是活跃的;
发送SYNACK 的ACK;
该报文可能包含
C-S的数据
ESTAB
ACKbit=1, ACKnum=y+1
接收ACK(y)
表明客户端是活跃的
ESTAB

建立连接（二次握手）


2次握手
Let’s talk
ESTAB
OK
ESTAB
choose x
req_conn(x)
ESTAB
acc_conn(x)
ESTAB

2次握手建立连接在可靠网络上：可行

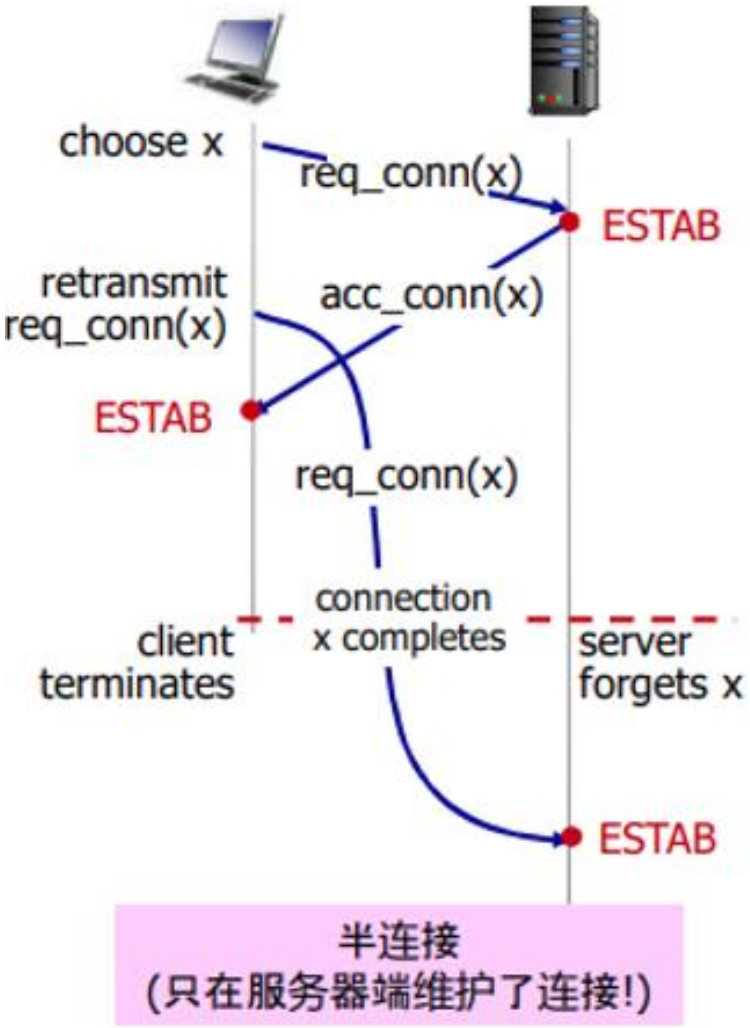
实际的网络：

1）延迟

2）丢失

3）重复

4）失序

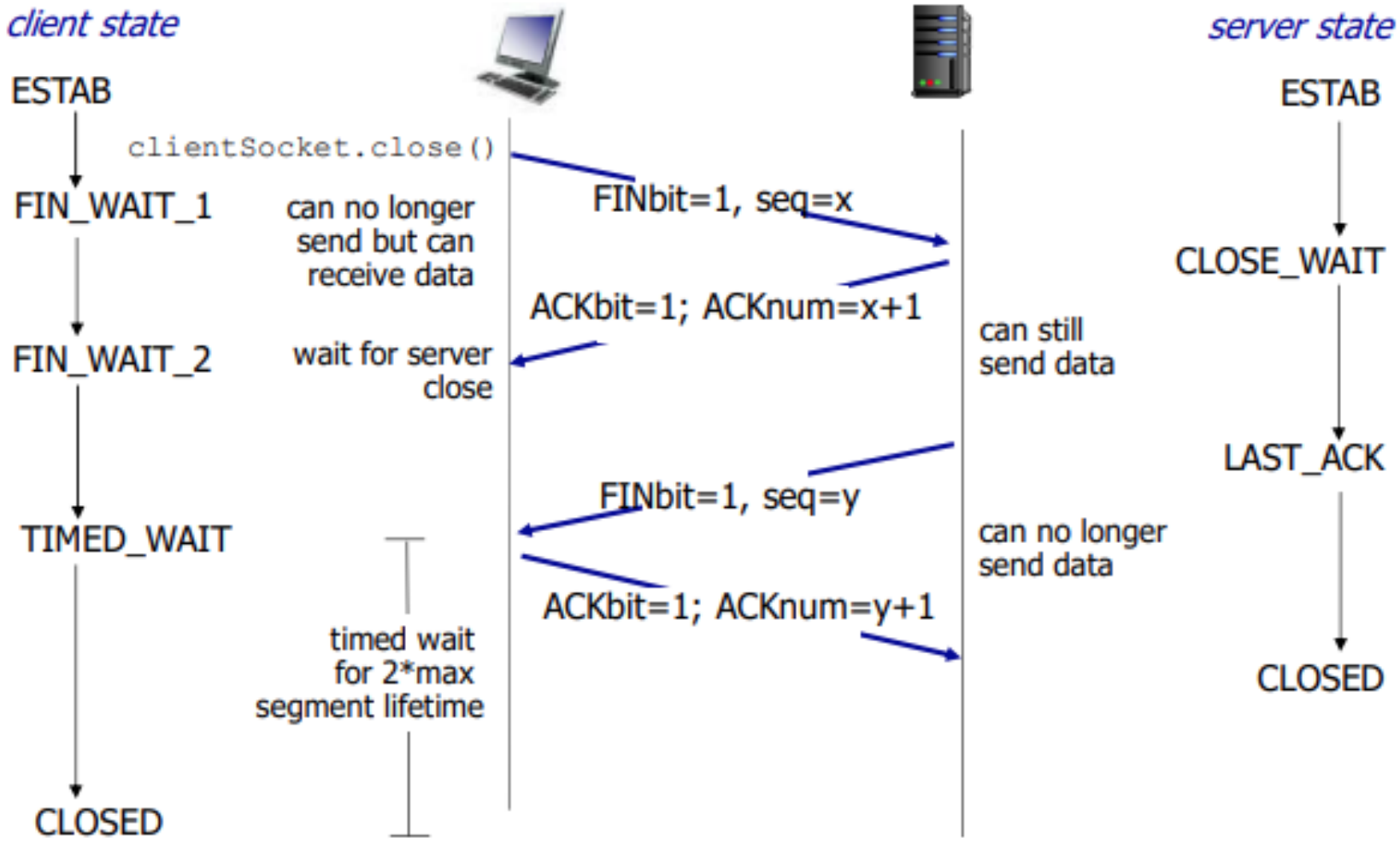


释放连接（4次握手）

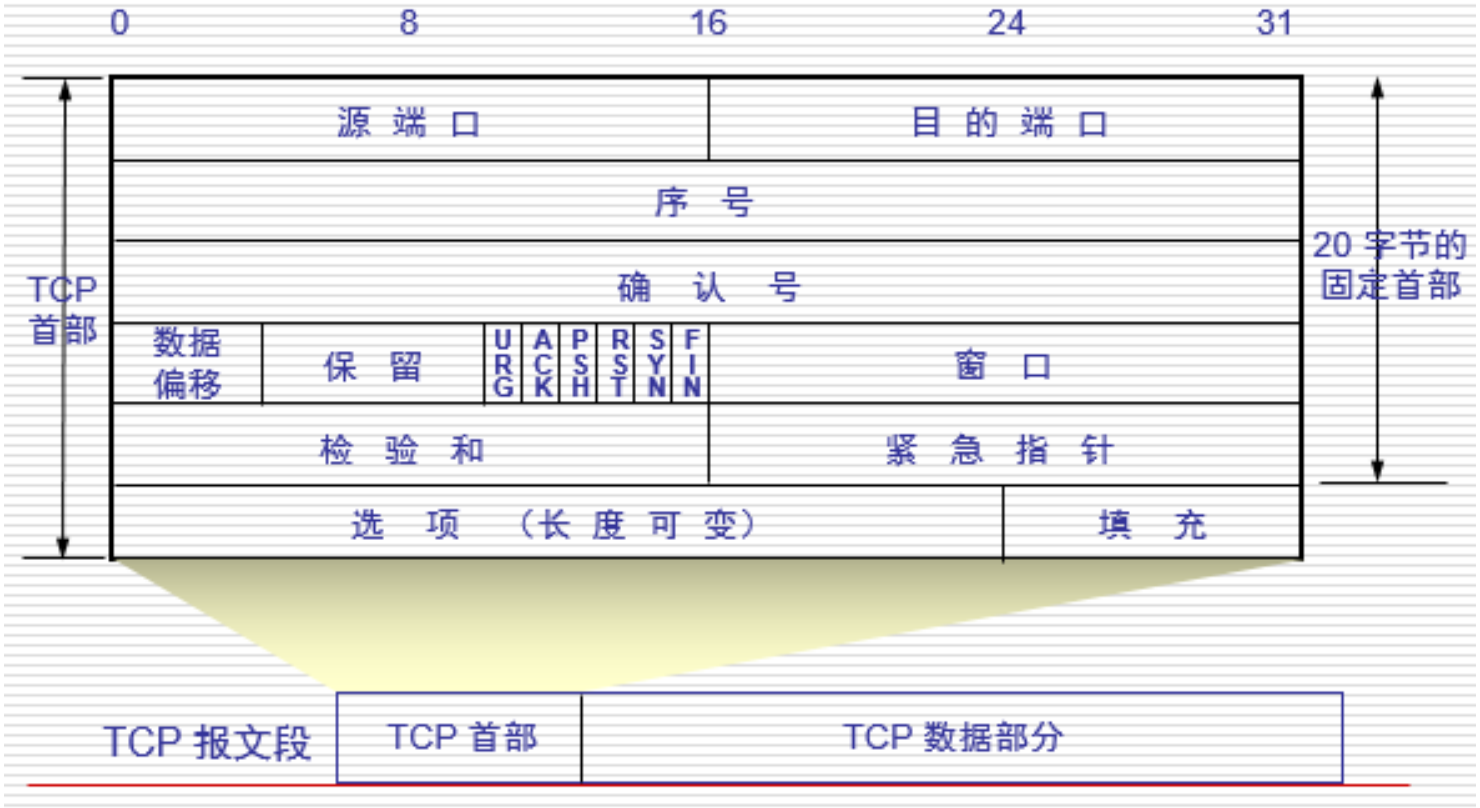
客户端，服务器分别关闭连接：

- ✓ 发送FIN bit = 1的TCP报文段
- ✓ 一旦接收到FIN，ACK回应收到的FIN段
- ✓ ACK可以和它自己发出的FIN段一起发送

释放连接



MSL：最大报文段生命周期



RST：复位标志，处理不存在的连接请求，强制关闭异常连接

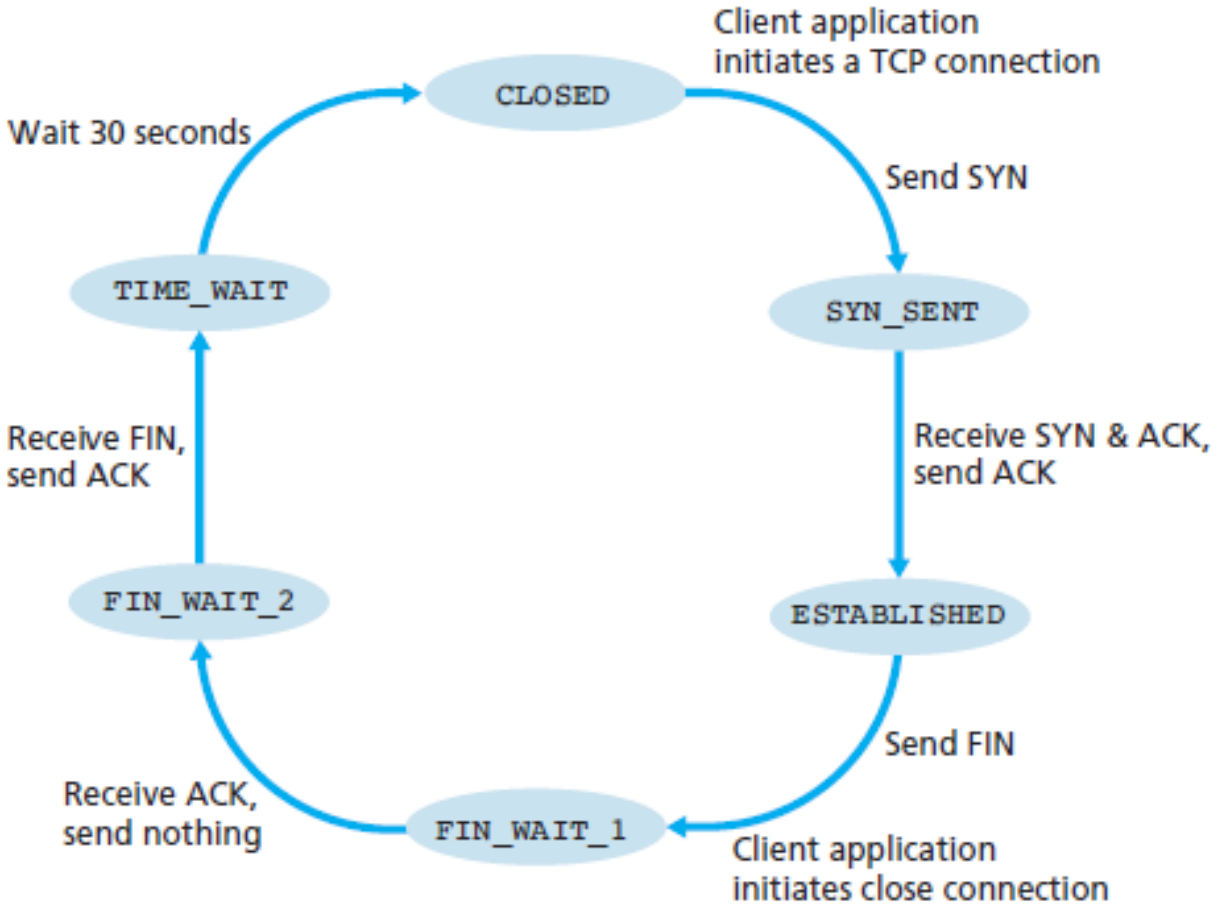


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

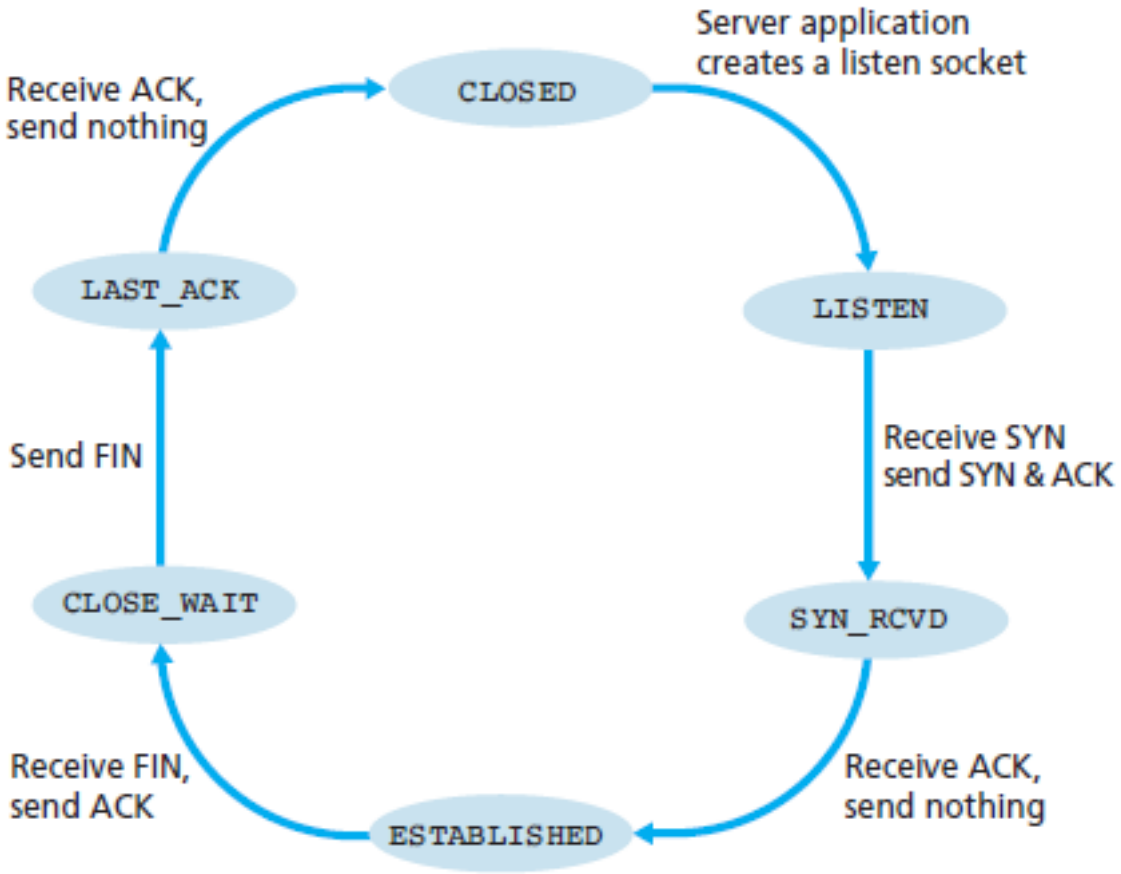
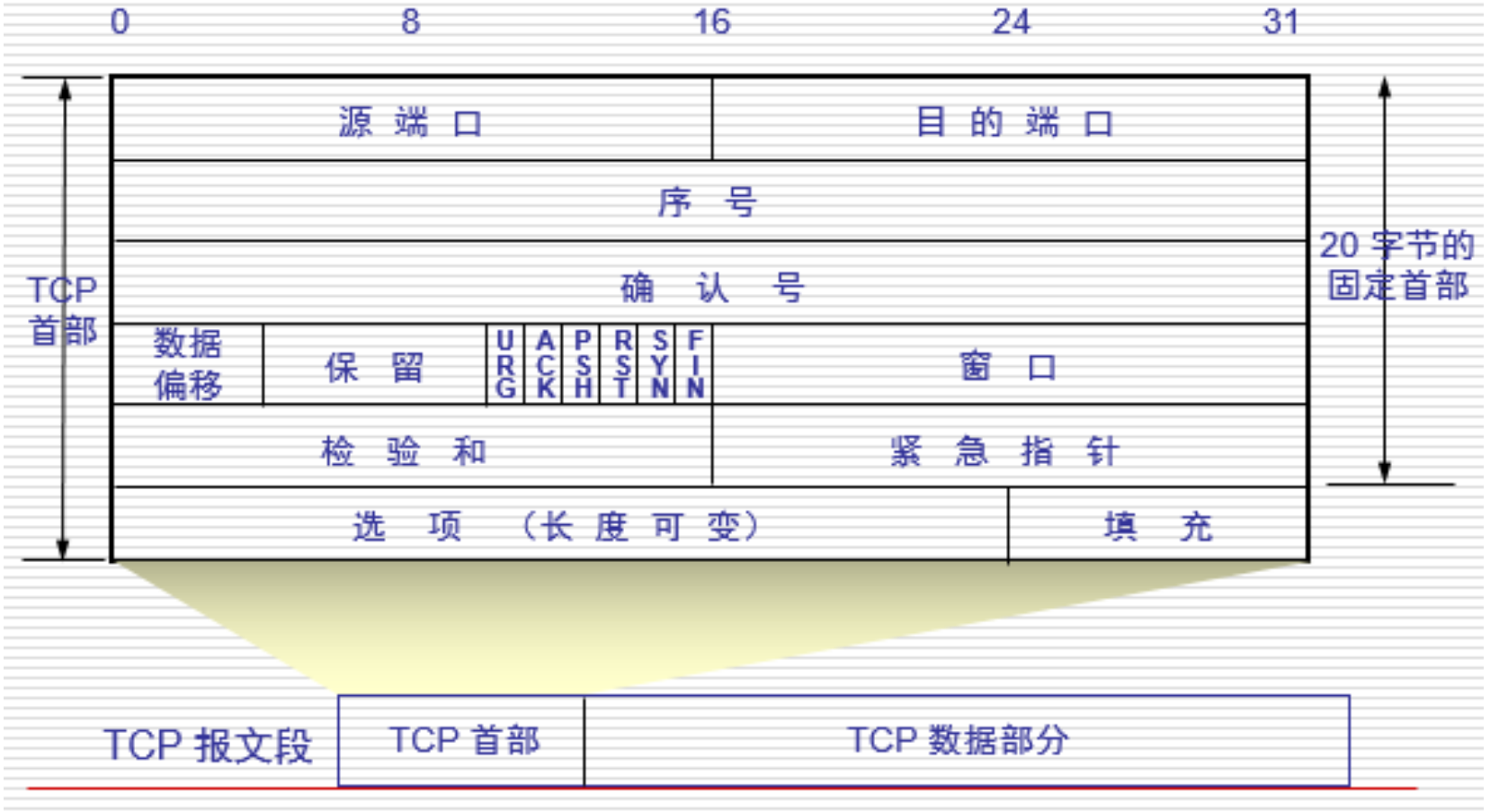


Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP

1. TCP报文格式
2. TCP可靠传输
3. TCP连接管理
4. TCP流量控制
5. TCP拥塞控制

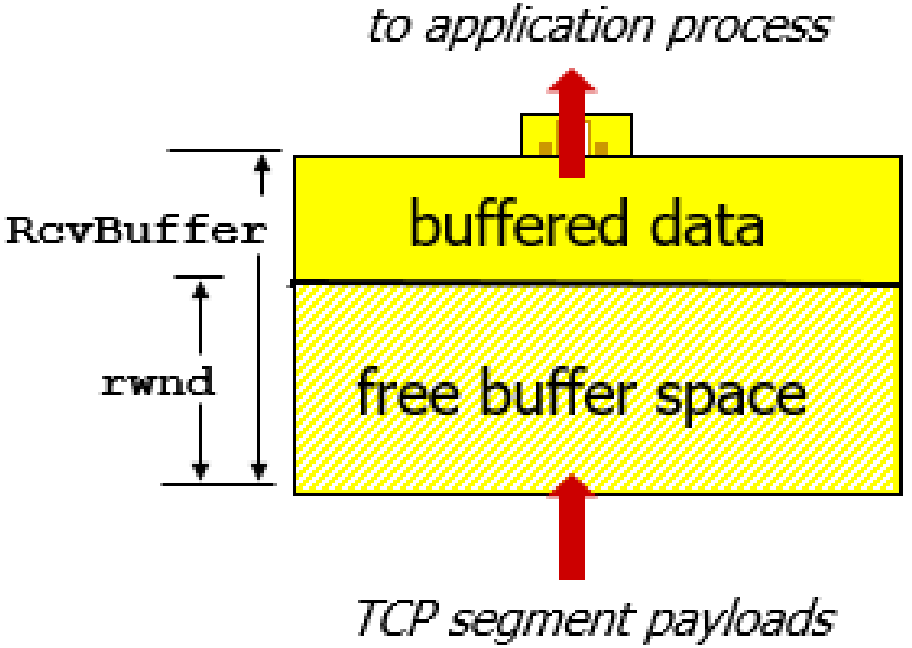
流量控制：控制发送方的发送速度（发送窗口的大小），避免接收方来不及接收

接收方：通过窗口字段，来控制发送窗口的大小



SendWind<=RecvWind

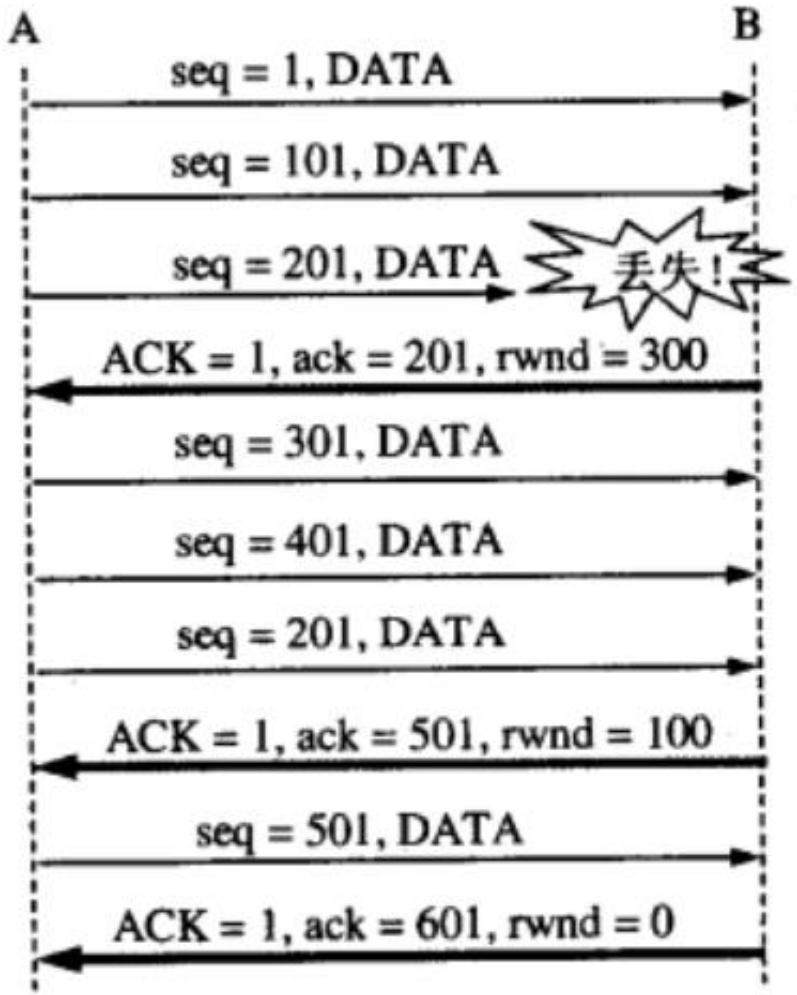
接收方：接收缓冲区



两个指针变量：

LastByteRead：最后一个读取的字节

LastByteRevd：最后一个收到的字节



A 发送了序号 1 至 100，还能发送 300 字节

A 发送了序号 101 至 200，还能发送 200 字节

允许 A 发送序号 201 至 500 共 300 字节

A 发送了序号 301 至 400，还能再发送 100 字节新数据

A 发送了序号 401 至 500，不能再发送新数据了

A 超时重发旧的数据，但不能发送新的数据

允许 A 发送序号 501 至 600 共 100 字节

A 发送了序号 501 至 600，不能再发送了

不允许 A 再发送（到序号 600 为止的数据都收到了）

流量控制中，容易产生“小窗口”和“傻瓜窗口”

1）小窗口： Nagle算法

2）傻瓜窗口：Clark算法

1. TCP报文格式
2. TCP可靠传输
3. TCP连接管理
4. TCP流量控制
5. TCP拥塞控制

大量的源（主机）以过高的速率发送数据

——网络带宽，网络交换设备的处理能力有限

拥塞：

1）丢包：路由器缓冲区溢出

2）长延时：路由器队列排队

拥塞控制方法：

1）端到端的拥塞控制：发送方根据网络拥塞程度限制其发送速率

2）网络辅助的拥塞控制：路由器向发送方反馈网络的拥塞状态（ICMP：源抑制报文）

发送方根据网络拥塞程度限制其发送速率

——没有拥塞，增大发送速率，否则，降低发送速率

1) 发送方如何判断网络拥塞?

拥塞：

1）丢包：路由器缓冲区溢出

2）长延时：路由器队列排队

——发送方超时重传，即判断网络出现了丢包，拥塞发生

端到端的拥塞控制：发送方根据网络拥塞程度限制其发送速率

——没有拥塞，增大发送速率，否则，降低发送速率

2）如何控制发送速率?

控制发送窗口的大小 Sendwind

TCP发送方维持一个变量：拥塞窗口CongWind

SendWind< = Min [RecvWind, CongWind]

端到端的拥塞控制：发送方根据网络拥塞程度限制其发送速率

——没有拥塞，增大发送速率，否则，降低发送速率

3）如何调整拥塞窗口？

拥塞控制算法：

1. 慢启动
2. 拥塞避免
3. 快恢复

端到端的拥塞控制：发送方根据网络拥塞程度限制其发送速率

1）发送方如何判断网络拥塞

2）发送方如何调整发送速率

3）如何调整拥塞窗口？

拥塞控制算法：动态调整拥塞窗口

窗口大小的单位：最大报文段长度 MSS （ Max Segment Size ） ）

MSS： TCP**数据部分**的最大长度

MSS的确定：

1）IP分组大小：65535字节，确保一个TCP报文段能封装在一个IP分组中

2）链路层的MTU（最大传输单元）：MSS+ TCP/IP首部能封装成一个帧

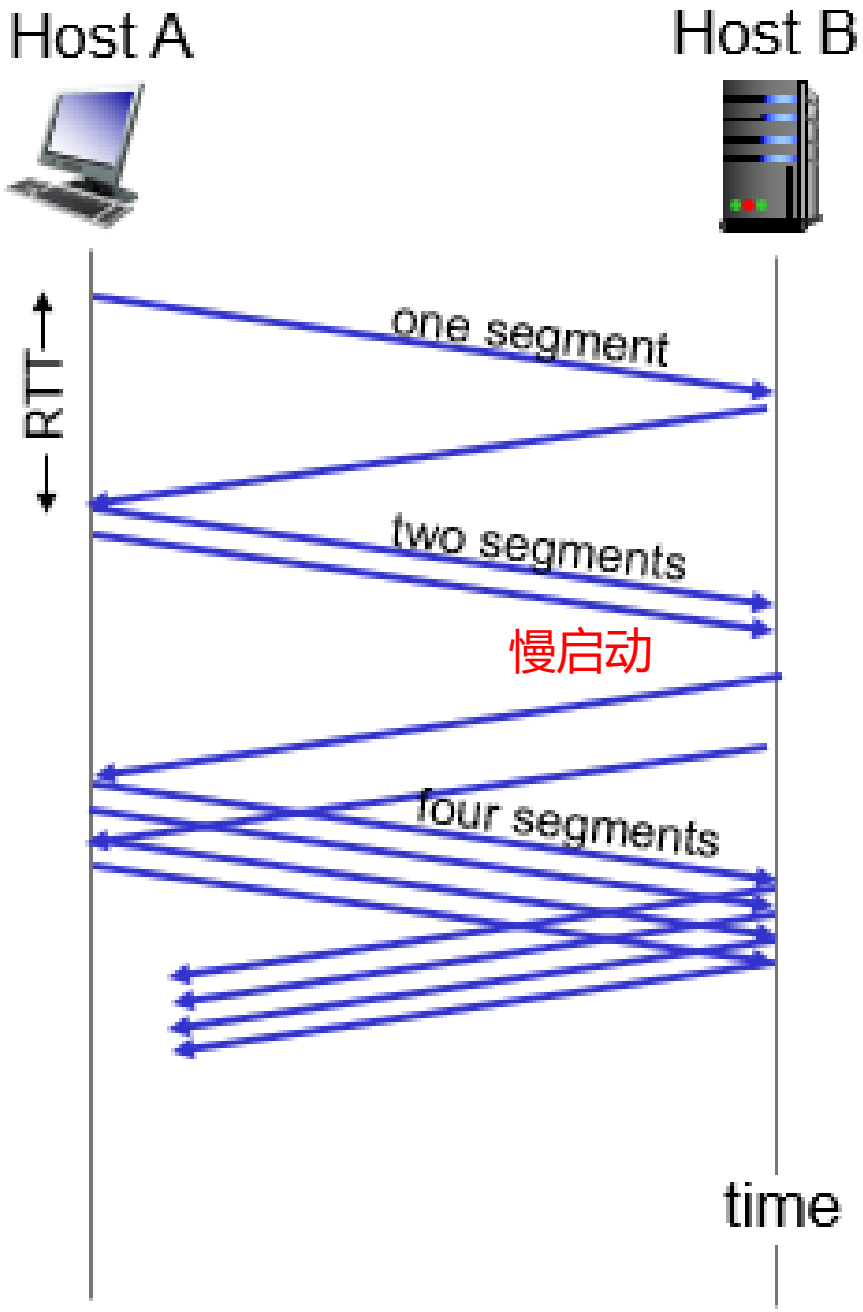
——以太网中MTU：1500B，MSS：1460B

——Internet标准最小MTU：576B，MSS：536B

拥塞控制算法

1）慢启动（slow start）：拥塞窗口cwnd

- ✓ 初始值设为：1个MSS
- ✓ 每收到1个新的报文段的确认：　　cwnd=cwnd+1个MSS



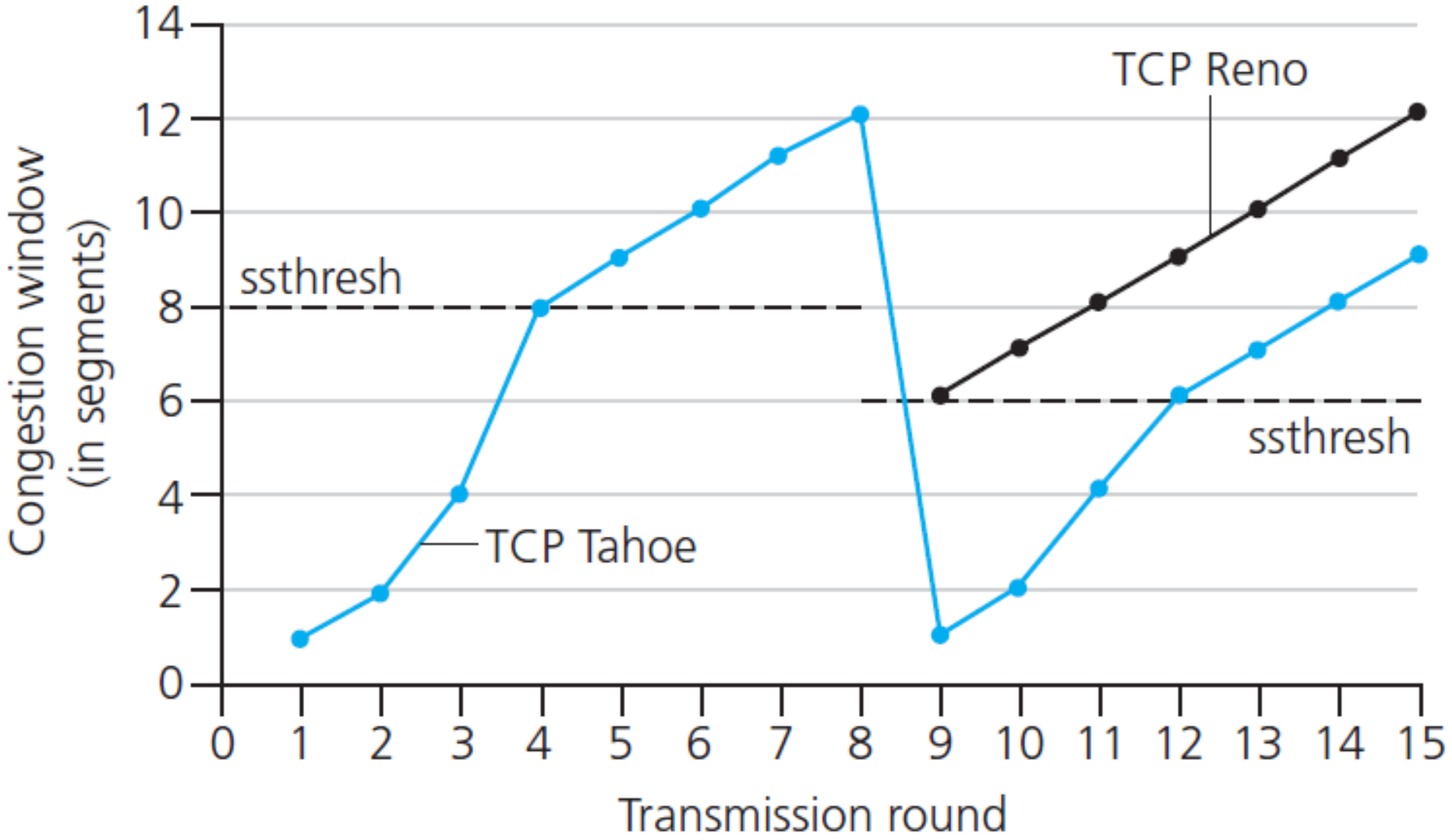
初始速率很慢，但是加速是指数增长

何时停止拥塞窗口的指数增长?

设置一个慢启动阈值： ssthresh （大小取决操作系统和TCP协议实现）

拥塞窗口cwnd>= ssthresh：进入**拥塞避免阶段**

每经过一个RTT： cwnd=cwnd+1个MSS

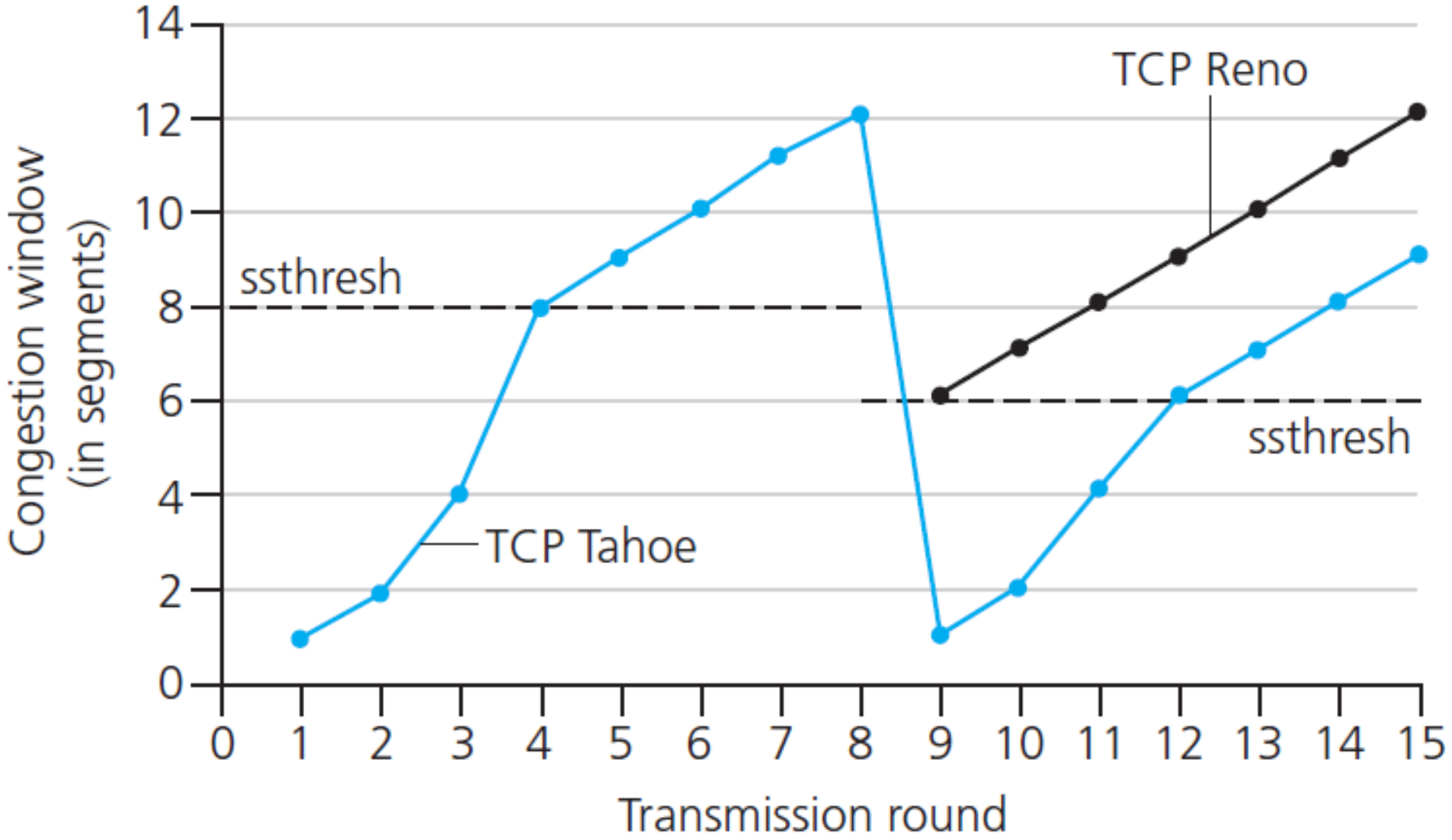

Congestion window
(in segments)
14
12
10
8
6
4
2
0
ssthresh
TCP Reno
ssthresh
TCP Tahoe
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Transmission round

在慢启动及拥塞避免阶段，出现拥塞（即超时重传）：

cwnd=1个MSS

慢启动阈值ssthresh =1/2（拥塞时的cwnd）

——再开始慢启动


Congestion window
(in segments)
14
12
10
8
6
4
2
0
ssthresh
TCP Reno
ssthresh
TCP Tahoe
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Transmission round

拥塞避免：拥塞窗口cwnd

每经过一个RTT：　cwnd=cwnd+1个MSS

实际采用的方法：

每收到一个ACK（假设每个报文段返回1个确认）

cwnd=cwnd+（1MSS/cwnd）*1MSS

3）如何调整拥塞窗口?

拥塞控制算法:

1. 慢启动
2. 拥塞避免
3. 快恢复

快重传机制：

- 考虑：接收方收到失序报文段（即比期望序号大的报文段到达）的情况

——失序可能由于网络设备或链路故障引起，实际：网络没有拥塞

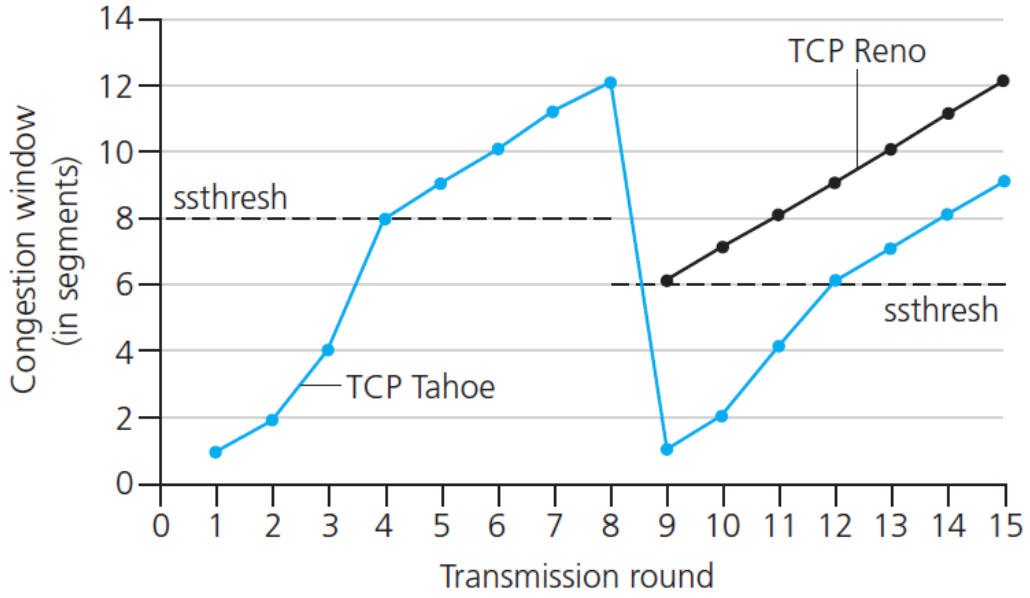
——发送方超时重传，增加了时延

快重传：通知发送方，尽快重传失序的报文段

快重传：通过确认号

——对已经接收的最后一个按序字节进行重复确认，即发送多个ACK

发送方收到多个ACK，在该报文段超时之前重传改报文段



TCP Tahoe版本中，超时和快重传，发送方都认为网络拥塞

- cwnd=1个MSS

改进

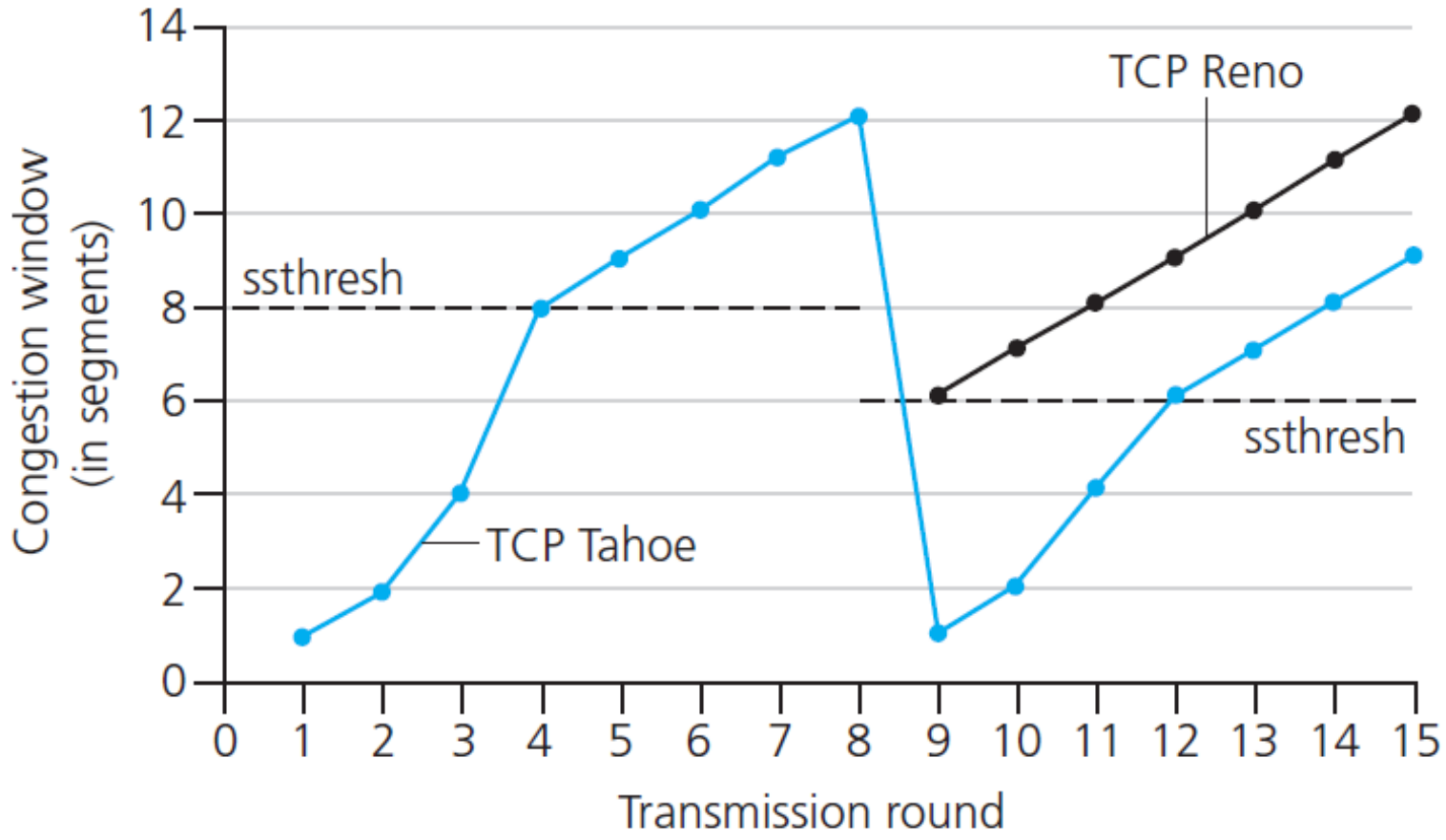
快重传：不认为网络拥塞，窗口可适当减小

快恢复：

- ✓ ssthresh =1/2（拥塞时的Cwnd）
- ✓Cwnd= ssthresh / Cwnd= ssthresh+n*MSS

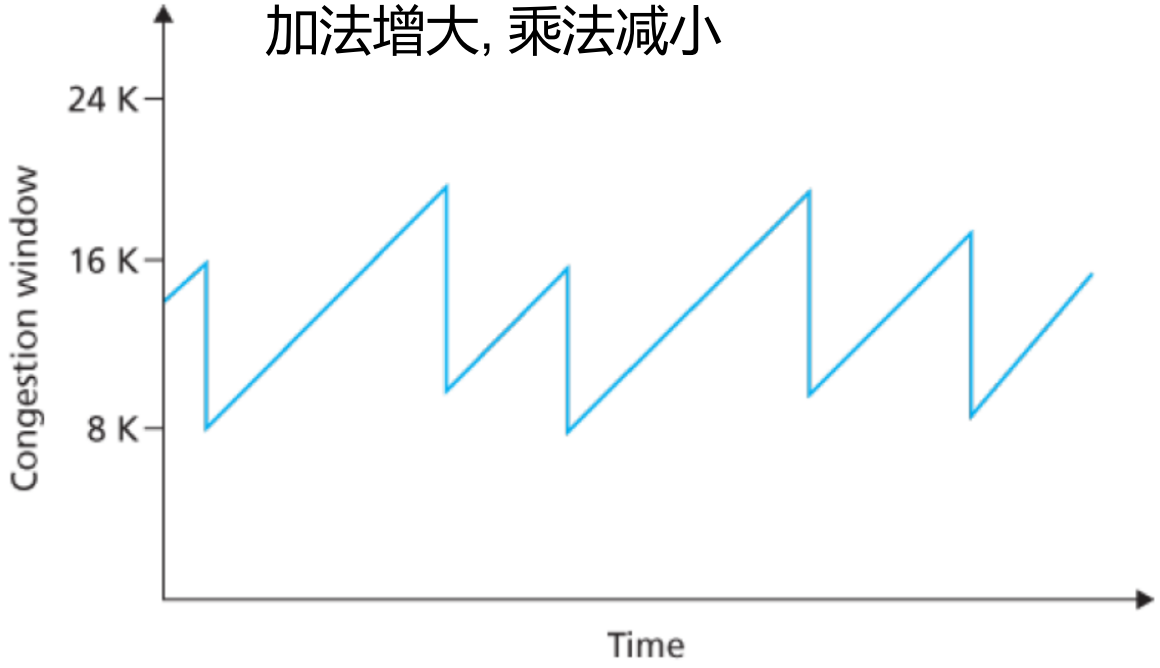
收到新的报文段确认：

✓进入拥塞避免阶段


Congestion window
(in segments)
14
12
10
8
6
4
2
0
ssthresh
TCP Reno
ssthresh
TCP Tahoe
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Transmission round

AIMD（additively increase， *multiplicative decrease*）

加法增大, 乘法减小

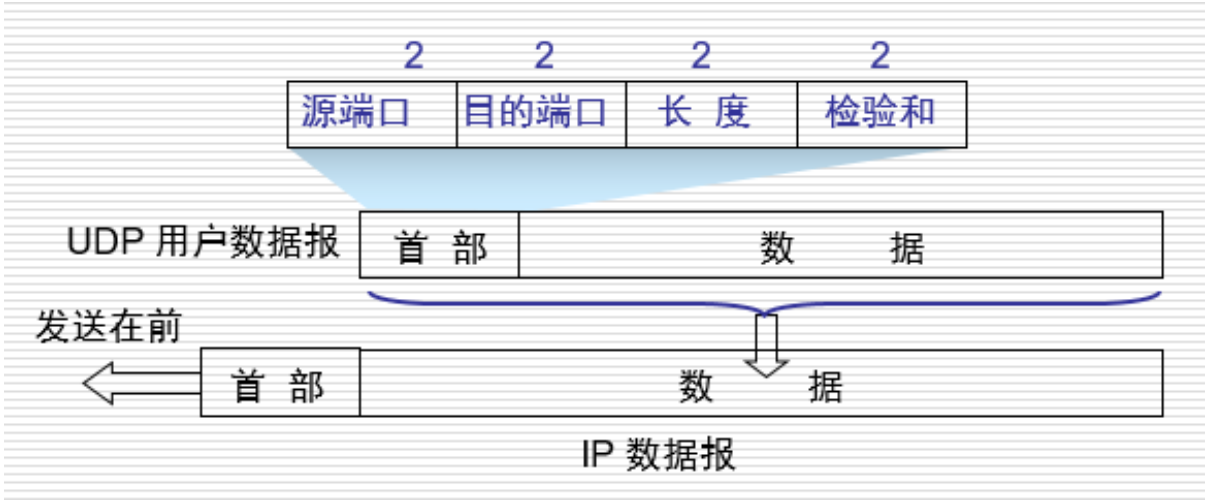


TCP（传输控制协议：Transmission Control Protocol）

1. TCP报文格式
2. TCP可靠传输
3. TCP连接管理
4. TCP流量控制
5. TCP拥塞控制

UDP（用户数据报协议：User Datagram Protocol）

UDP（用户数据报协议：User Datagram Protocol）



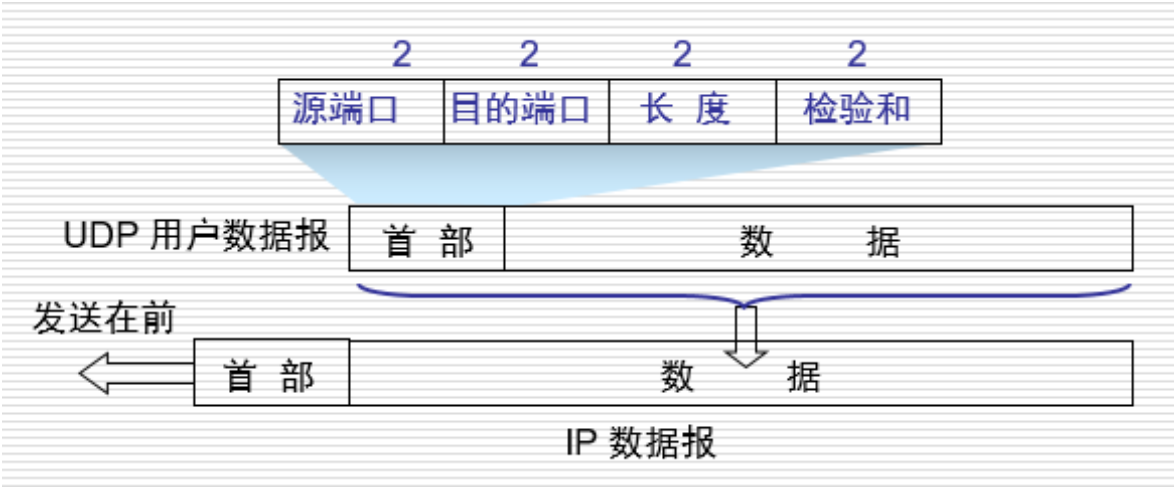
1）首部 8个字节：固定首部

2）长度：首部+数据的长度

3）校验和：UDP首部+UDP数据+伪首部（12字节）

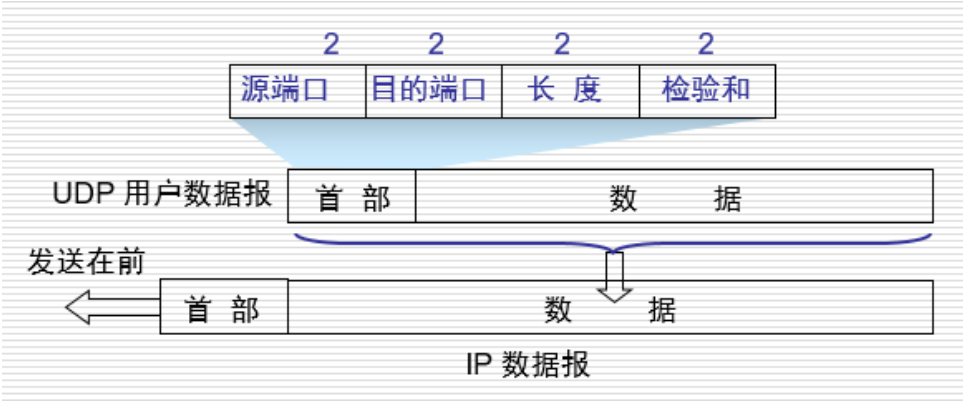
UDP

- ✓ 无连接：发送方和接收方没有握手过程
- ✓ 面向报文：一次封装整个报文（非字节流）



为什么要有UDP

1. 首部开销小（8个字节）
2. 无需建立连接，不会引入建立连接的时延
3. 不维护连接状态（服务端可以支持更多的客户访问）
4. 支持广播和组播



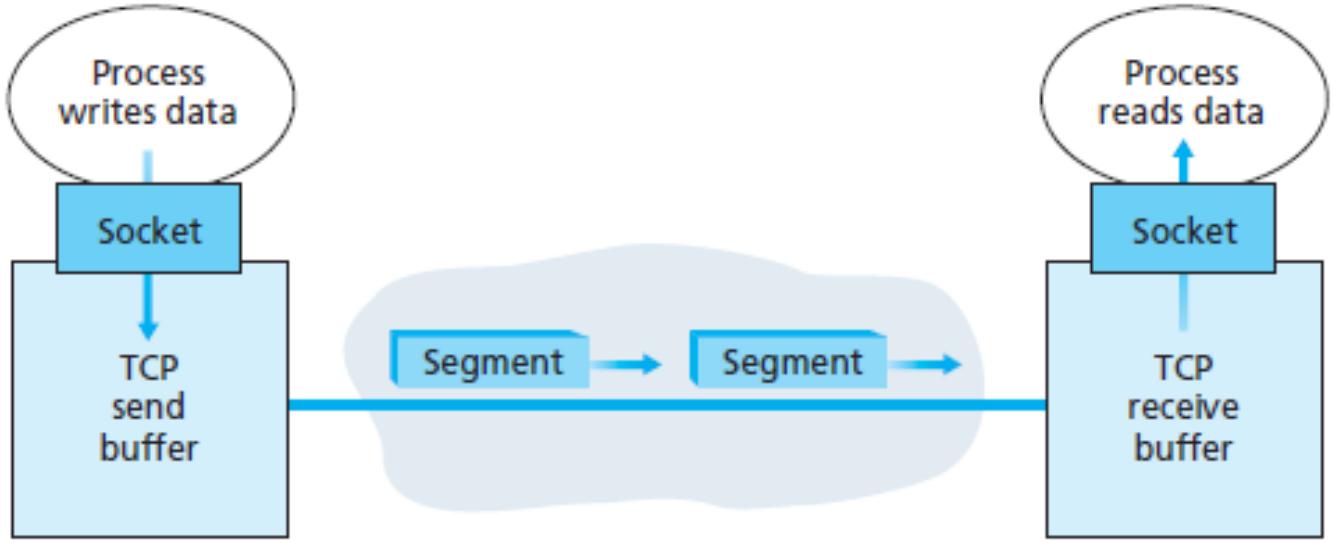
为什么要有UDP

5. 拥塞控制和流量控制，适用于某些实时应用（吞吐量满足一定的要求，容忍一定的数据丢失）——视频、音频等流媒体传输

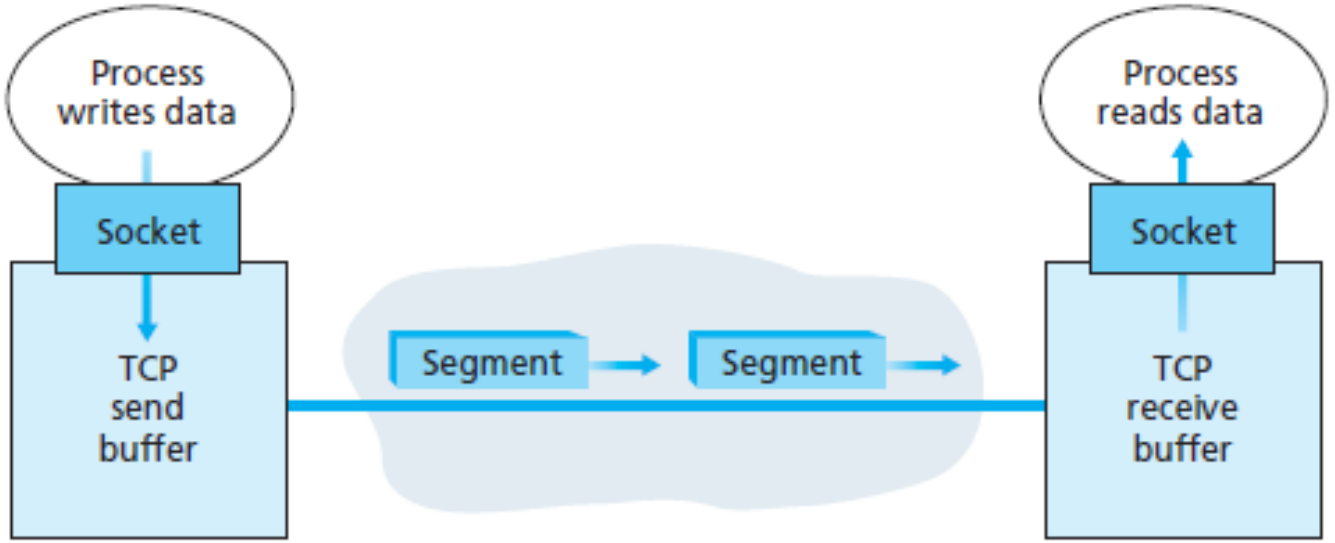
附录：Socket（套接字）

网络编程接口（API），对 TCP/IP 协议栈的封装

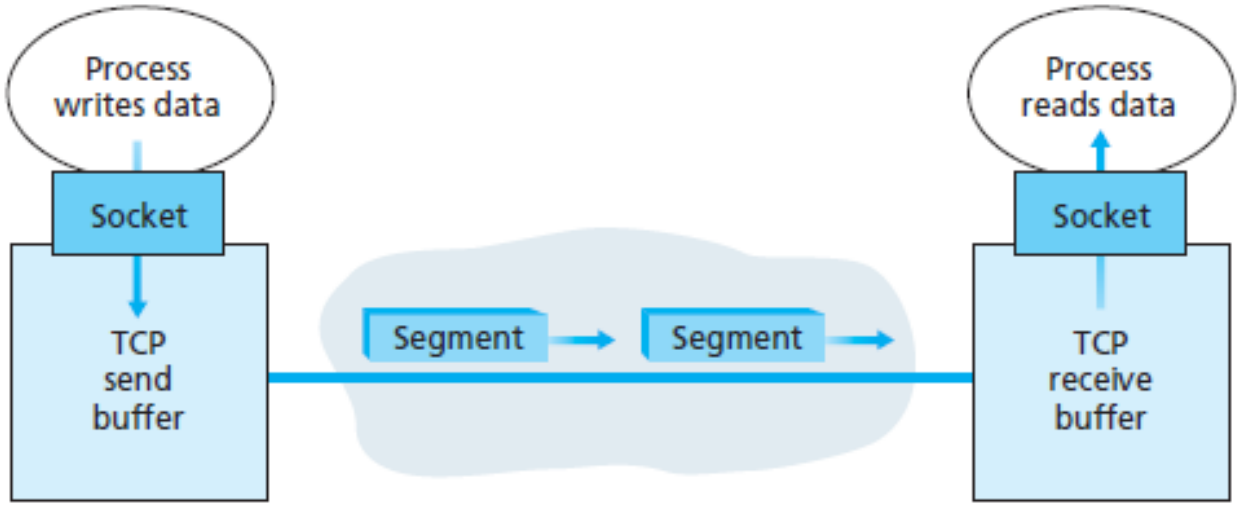
- 最早在BSD UNIX系统中实现， Berkeley Socket API



- 应用程序：1）选择传输层协议 TCP/UDP　2）参数设置（缓冲区大小等）



一个套接字标识：<IP地址，端口号>



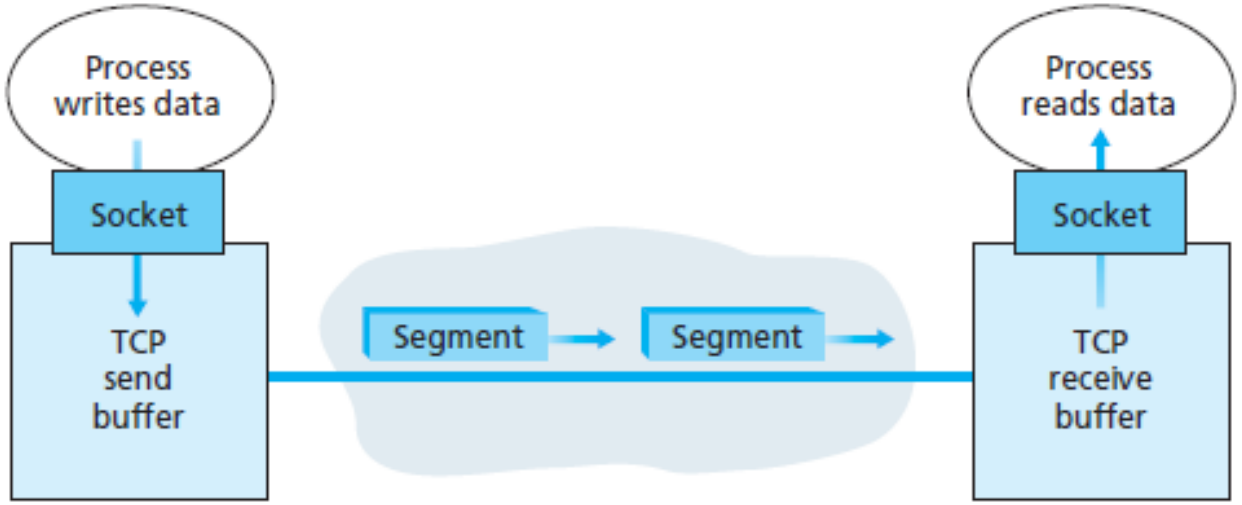
采用TCP协议：

一条连接：4元组表示

<源主机IP地址，源端口号， 目的主机IP地址，目的端口号>

IPA	**1170**	**IPserver**	**80**	**Socket/90000**
IPB	**9157**	**IPserver**	**80**	**Socket/90004**
IPA	**9157**	**IPserver**	**80**	**Socket/90005**

一个套接字标识：<IP地址，端口号>



采用TCP协议：

一条连接：4元组

<源主机IP地址，源端口号， 目的主机IP地址，目的端口号>

采用UDP协议：无连接，支持广播

- 作业
  - 6.8、6.14、6.25、6.27

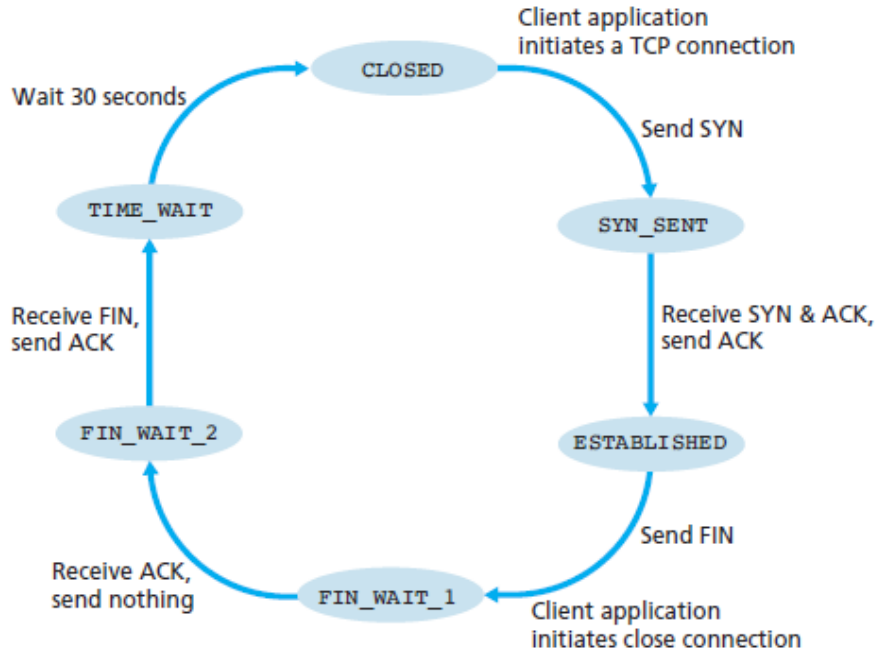


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

  - 补充：

TCP释放连接，Client端为什么不直接close，而是要等2*MSL才close