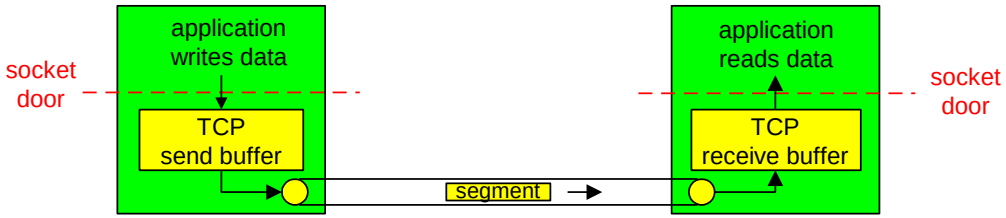
TCP概述 RFCs: 793, 1122, 1323, 2018, 2581

- **点到点:**
  - 一个发送方, 一个接收方
  - 连接状态与端系统有关，不为路由器所知
- **可靠、有序的字节流:**
  - 没有 “报文边界”
- **流水线:**
  - TCP拥塞和流量控制设置滑动窗口协议
- ***发送和接收缓冲区***



- **全双工数据:**
  - 同一连接上的双向数据流
  - MSS: 最大报文段长度
  - MTU:最大传输单元
- **面向连接:**
  - 在进行数据交换前，初始化发送方与接收方状态，进行握手(交换控制信息),
- **流量控制:**
  - 发送方不能淹没接收方
- **拥塞控制:**
  - 抑止发送方速率来防止过分占用网络资源

最大报文段长度MSS

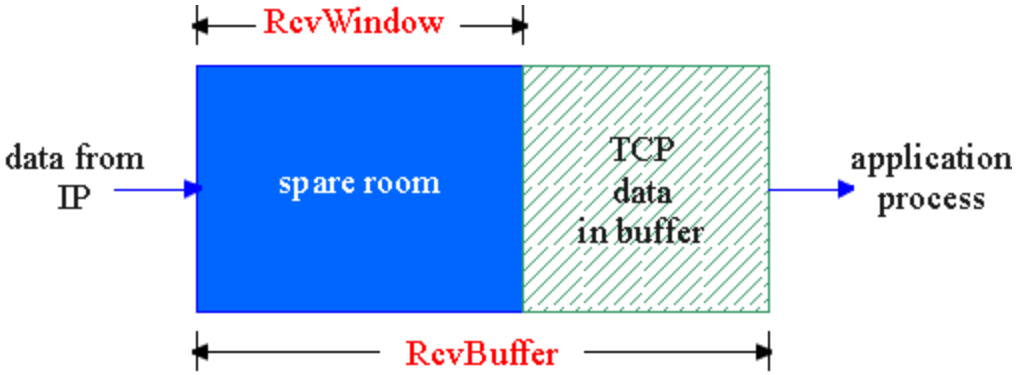
- TCP可从发送缓存中取出并放入报文段(segment)的数据量受限于最大报文段长度MSS。
- MSS通常根据最初确定的本地发送主机发送的最大链路层帧长度MTU来设置。
  - MSS+TCP/IP头部的长度（40字节）= MTU
- 以太网和PPP链路中MTU为1500字节，因此MSS通常为1460字节。

3.5.5 TCP 流量控制

流量控制

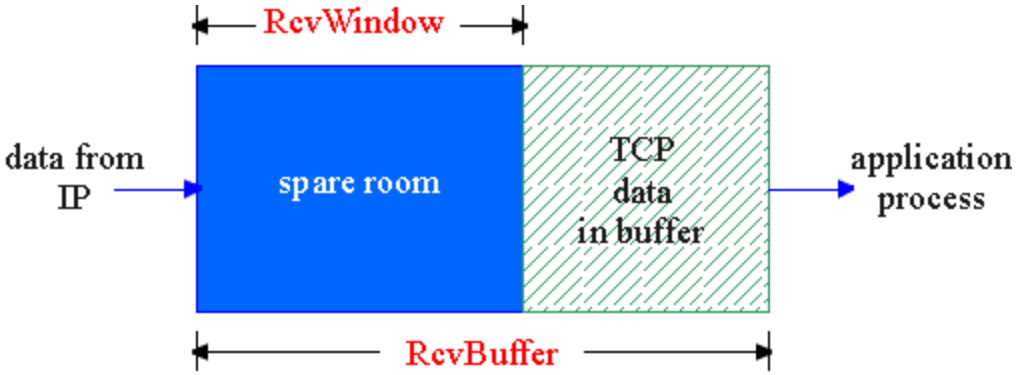
发送方不能发送太多、太快的数据让接收方缓冲区溢出

- TCP连接的接收方有1个接收缓冲区:



- 匹配速度服务: 发送速率需要匹配接收方应用程序的提取速率
- 应用进程可能从接收缓冲区读数据缓慢

3.5.5 TCP流控: 工作原理



(假设 TCP 接收方丢弃失序的报文段)

- 缓冲区的剩余空间

= RcvWindow

= RcvBuffer-[LastByteRcvd - LastByteRead]

- 接收方在报文段接收窗口字段中通告其接收缓冲区的剩余空间
- 发送方要限制未确认的数据不超过**RcvWindow**

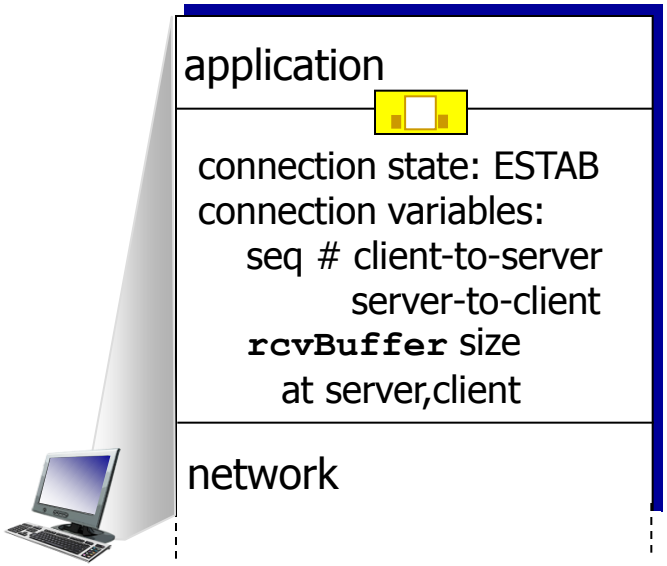
LastByteSent-LastByteAcked <或= RcvWindow

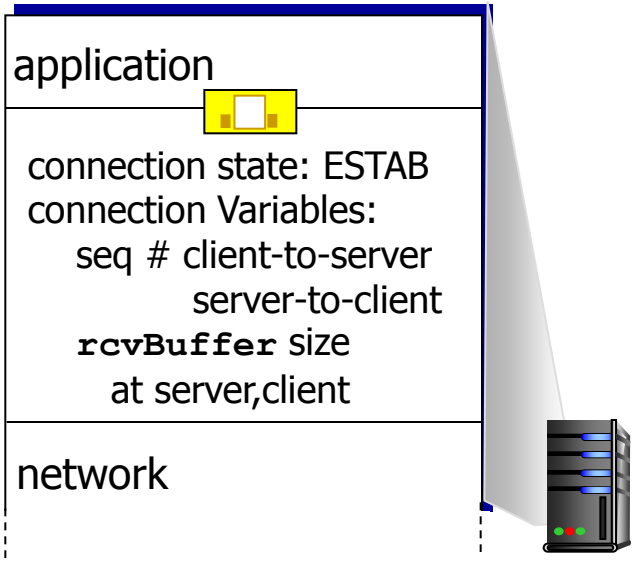
  - 保证接收缓冲区不溢出

3.5.6 连接管理

before exchanging data, sender/receiver “handshake”:

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters






```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

```
Socket connectionSocket =
   welcomeSocket.accept();
```

3.5.6 TCP 3-way handshake

三次握手:

步骤 1: 客户机向服务器发送 TCP SYN报文段

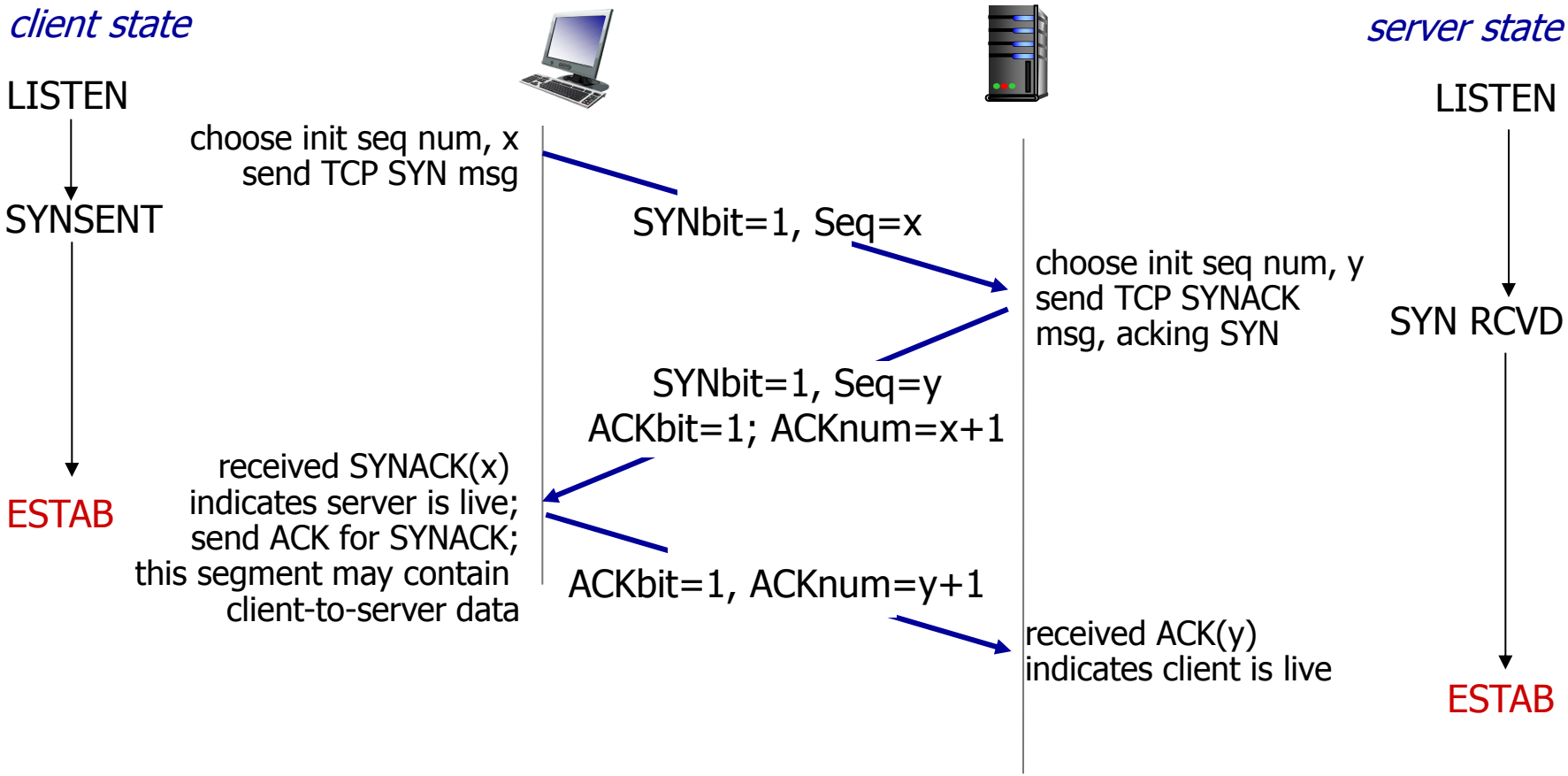
- 指定初始序号（随机产生）
- 没有数据

步骤 2: 服务器收到SYN报文段, 用SYNACK报文段回复

- 服务器为该连接分配缓冲区和变量
- 指定服务器初始序号

步骤 3: 客户机接收到 SYNACK, 用ACK报文段回复,可能包含数据

3.5.6 TCP 3-way handshake



TCP 关闭连接(续)

关闭连接: 服务器和客户端的任何一方都可以发送消息主动关闭连接

send TCP segment with FIN bit = 1

客户关闭套接字: `clientSocket.close();`

步骤 1: 客户机向服务器发送TCP FIN控制报文段

步骤 2: 服务器收到FIN，用ACK回答。关闭连接，发送FIN

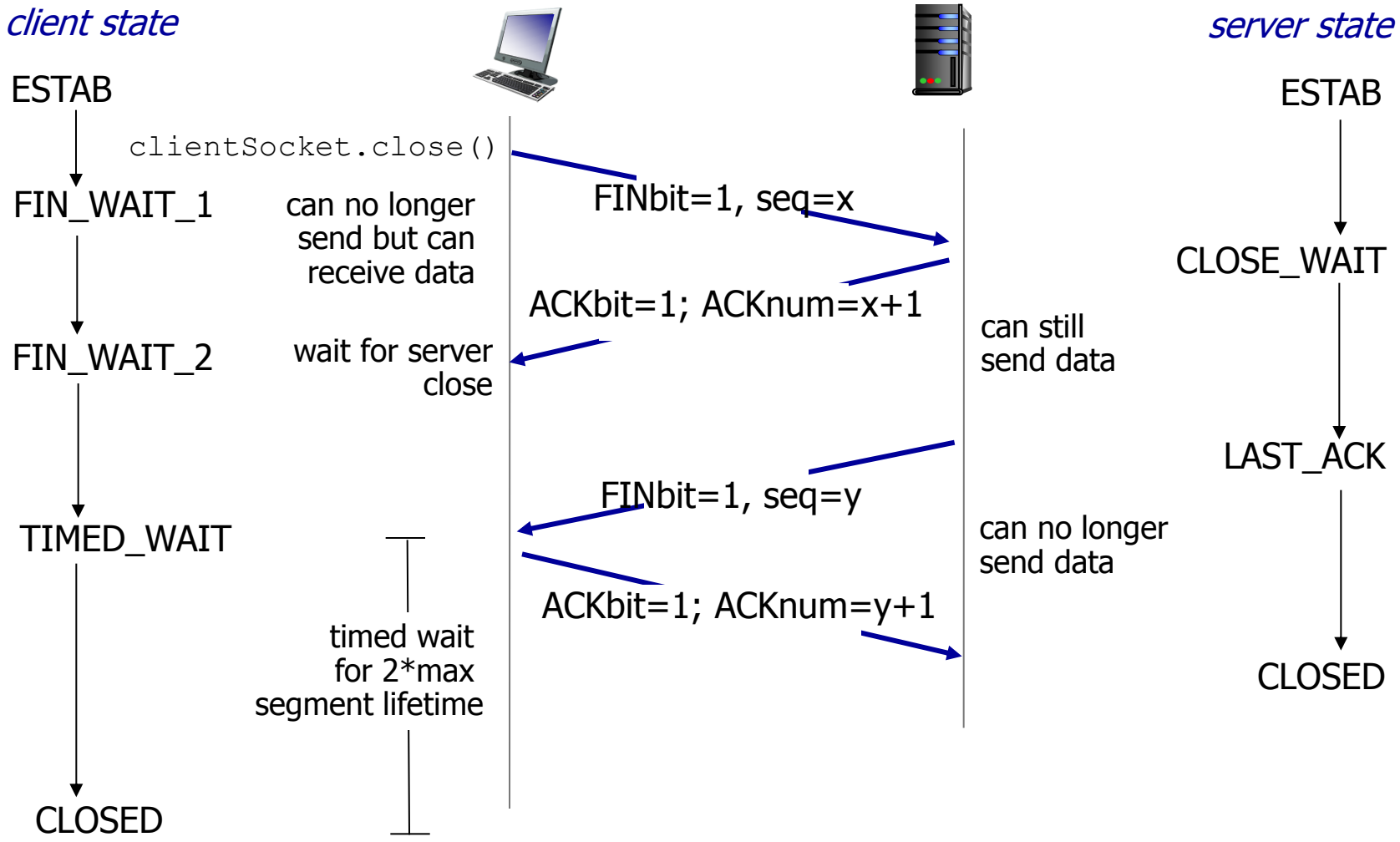
步骤 3: 客户机收到FIN, 用ACK回答

- 进入 “超时等待” - 将对接收到的FIN进行确认

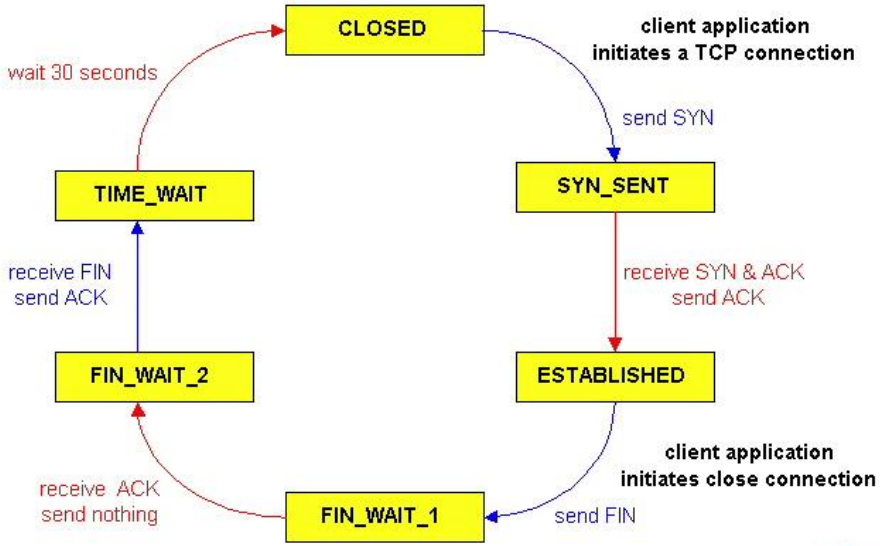
步骤 4: 服务器接收ACK，连接关闭

注意: 少许修改, 可以处理并发的FIN

TCP 关闭连接(续)

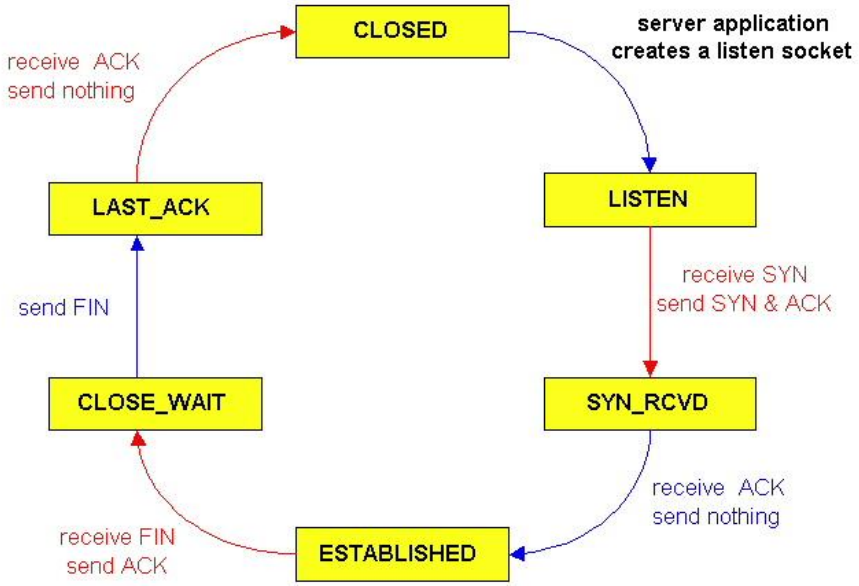

client state
server state
ESTAB
clientSocket.close()
FIN_WAIT_1
can no longer send but can receive data
FINbit=1, seq=x
ACKbit=1; ACKnum=x+1
CLOSE_WAIT
FIN_WAIT_2
wait for server close
can still send data
LAST_ACK
FINbit=1, seq=y
TIMED_WAIT
can no longer send data
ACKbit=1; ACKnum=y+1
timed wait for 2*max segment lifetime
CLOSED
CLOSED

TCP 连接管理 (续)



TCP 客户
生命周期

TCP 服务器
生命周期



开放问题讨论

- TCP为什么需要三次握手，而不采用两次握手机制？或者四次握手机制？
  - **防止失效的连接请求报文段被服务端接收，从而产生错误。**

- 失效的连接请求：若客户端向服务端发送的连接请求丢失，客户端等待应答超时后就会再次发送连接请求，此时，上一个连接请求就是“失效的”。
- 若建立连接只需两次握手，客户端并没有太大的变化，仍然需要获得服务端的应答后才进入ESTABLISHED状态，而服务端在收到连接请求后就进入ESTABLISHED状态。此时如果网络拥塞，客户端发送的连接请求迟迟到不了服务端，客户端便超时重发请求，如果服务端正确接收并确认应答，双方便开始通信，通信结束后释放连接。此时，如果那个失效的连接请求抵达了服务端，由于只有两次握手，服务端收到请求就会进入ESTABLISHED状态，等待发送数据或主动发送数据。但此时的客户端早已进入CLOSED状态，服务端将会一直等待下去，这样浪费服务端连接资源。

SYN洪泛攻击

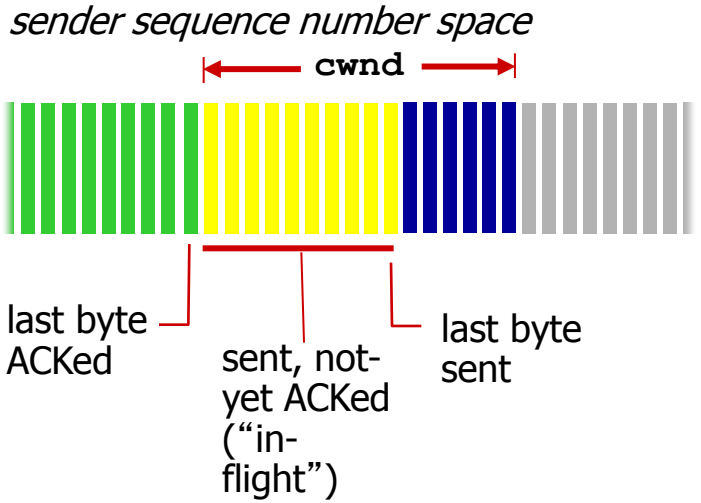
- TCP的三次握手机制为SYN洪泛攻击提供了环境。
  - 攻击者发送大量的TCP SYN报文段，而不去完成第3次握手的步骤，服务器不断为这些半开连接分配资源，导致服务器的连接资源被耗尽。
  - 一种有效的防御系统，称为SYN cookie，已经部署在大多数的主流操作系统中
    - 当服务器收到一个SYN报文时，它并不知道这个报文段是来自一个合法的用户，还是一个SYN洪泛攻击者，因此服务器不会为该报文段生成一个半开连接；
    - 服务器生成一个初始TCP序列号，这个序列号是TCP的源、目的IP地址、源、目的端口号和一个服务器才知道的秘密数字的hash值，称为cookie；
    - 服务器发送携带了cookie作为序列号的SYNACK分组

SYN洪泛攻击

- 如果客户是合法的，则客户将会返回一个ACK报文；
- 服务器收到这个ACK报文后，将其中的源、目的IP地址、源、目的端口号及以前生成的秘密数进行hash，得到的hash值加1，判断是否与该ACK报文的确认号字段是否一致，如果一致，才能判断客户是合法的。
- 最终服务器生成一个具有套接字的全开连接。

如果客户是攻击者，则不会返回ACK报文，那么服务器并不会对该客户的TCP连接事先分配任何资源

3.7 TCP 拥塞控制



- 端到端控制 (没有网络辅助)
- 发送方限制传输:

  LastByteSent-LastByteAcked
  ≤ cwnd

- 粗略地,

$$\text{速率} = \frac{\text{cwnd}}{\text{RTT}} \text{ Bytes/sec}$$

- 拥塞窗口是动态的, 具有感知到的网络拥塞的函数

网络设计十大问题之一

发送方如何感知网络拥塞?

- 丢失事件 = 超时*或者* 3个重复ACK
- 发生丢失事件后，TCP发送方降低速率(拥塞窗口)

TCP加增倍减 AIMD

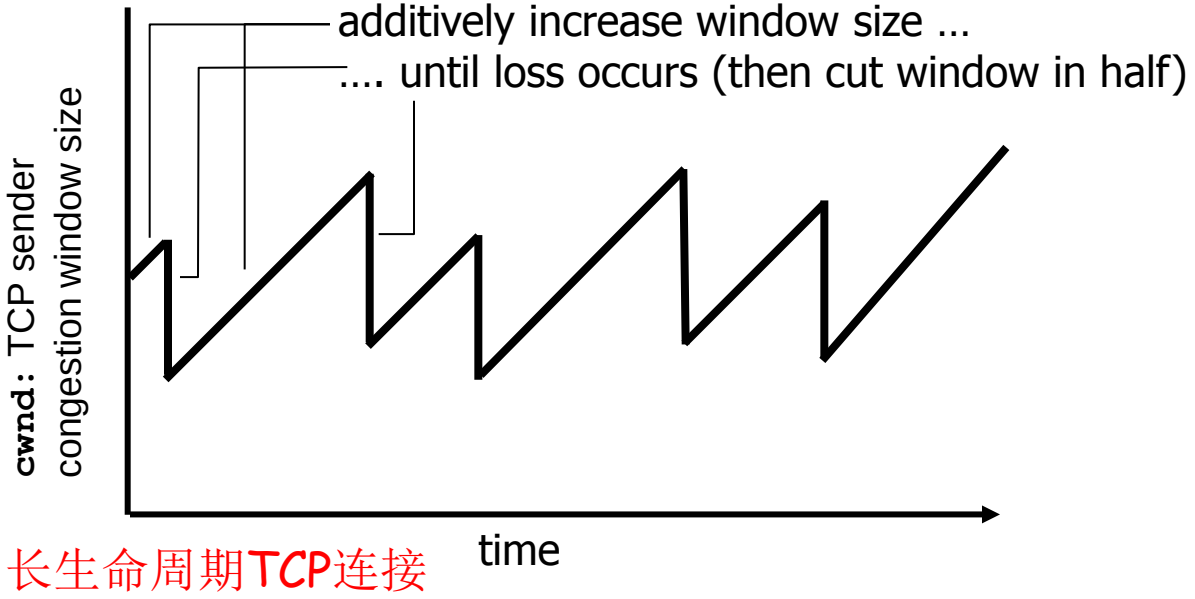
乘性减:

丢包事件后，拥塞窗口值减半

加性增:

如没有检测到丢包事件，每个RTT时间拥塞窗口值增加一个MSS (最大报文段长度)

AIMD saw tooth behavior: probing for bandwidth



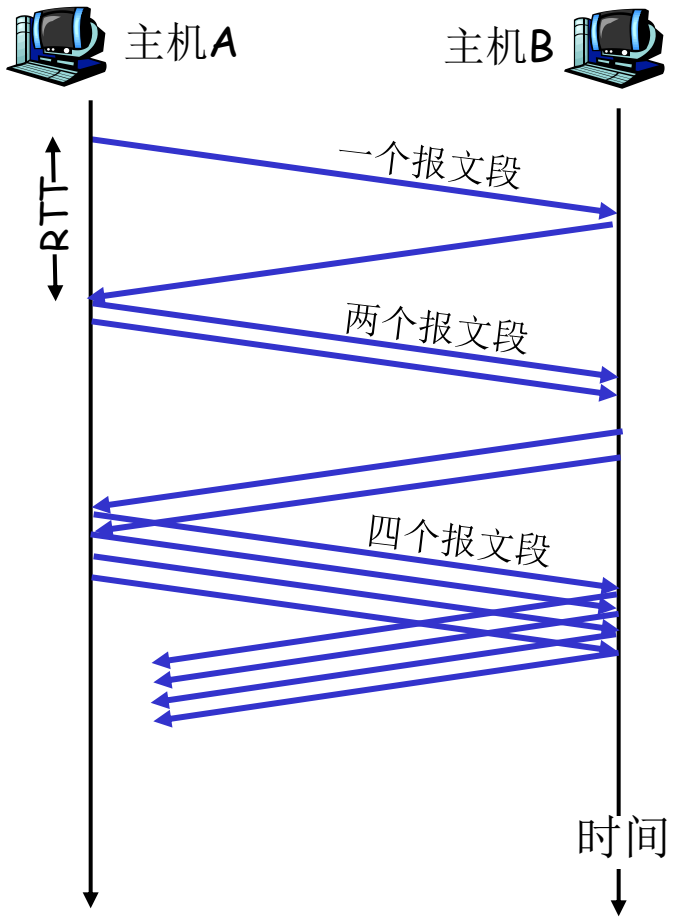
长生命周期TCP连接

TCP慢启动（Slow Start）

- 在连接开始时, 拥塞窗口值 = 1 MSS
  - 例如: MSS= 500 bytes & RTT = 200 msec
  - 初始化速率 = 20 kbps
- 可获得带宽可能 >> MSS/RTT
  - 希望尽快达到期待的速率
- 当连接开始，以指数倍增加速率，直到第一个丢失事件发生

TCP 慢启动(续)

- 当连接开始的时候，速率呈指数式上升，直到第1次报文丢失事件发生为止:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- 总结: 初始速率很低，但指数倍增加



改进

- 收到3个冗余确认后:
  - cwnd减半
  - 窗口再线性增加
- 但是超时事件以后:
  - cwnd值设置为1 MSS
  - 窗口再指数增长
  - 到达一个阈值 (Threshold)后，再线性增长

基本思想:

- 3个冗余ACK指示网络还具有某些传送报文段的能力
- 3个冗余ACK以前的超时,则更为“严重”

从慢启动进入拥塞避免状态(CA)

问题: 什么时候从指数增长转变为线性增长?

回答: **cwnd**达到它超时以前1/2的时候.

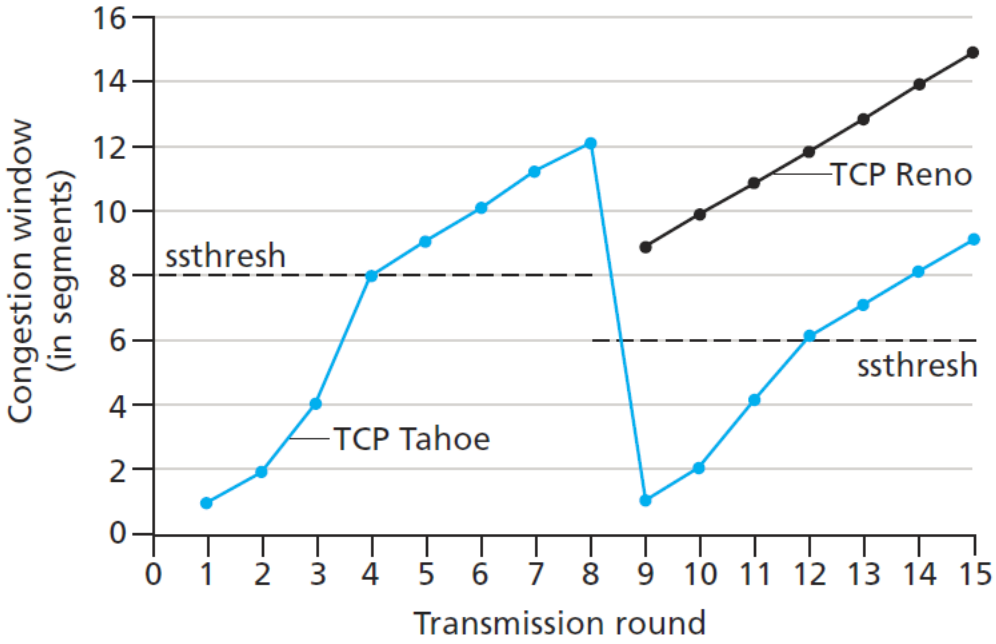


Figure 3.53 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

实现方法:

- 设置一个阈值ssthresh（慢启动阈值）
- 在丢包事件发生时，阈值ssthresh设置为发生丢包以前的cwnd的一半

拥塞避免状态

- 当处于慢启动状态，拥塞控制窗口cwnd大于等于ssthresh时，发送方进入拥塞避免状态；
- 当处于快速恢复状态，如果收到一个新的ACK，进入拥塞避免状态

cwnd = ssthresh, 重复计数ACK清0

- 在该状态拥塞窗口cwnd在每个RTT，值增加一个MSS，即每新收到一个ACK:

cwnd = cwnd + MSS(MSS/cwnd) 注意与慢启动的区别

- 例如，MSS = 1460字节， cwnd = 14600字节，则在一个RTT之内可以发送10个报文段，每个ACK到达，拥塞窗口增加1/10MSS，在收到10个报文段的ACK后，拥塞控制窗口增加1个MSS

快速恢复状态

- 当发送方收到来自于接收方3个重复的ACK报文时，进入快速恢复状态；3个重复的*ACK*报文触发的动作包括：当前的状态可以是慢启动或者拥塞避免

进入

ssthresh = cwnd/2
cwnd = ssthresh+3*MSS
重传丢失的报文

仍然可能处于拥塞状态

- 当处于快速恢复状态时（此时网络可能发生拥塞）：
  - 如果收到一个重复的ACK，**cwnd = cwnd + 1 MSS**
  - **当收到一个新的*ACK*（即丢失报文的*ACK*），TCP在降低拥塞控制窗口后进入拥塞避免状态； cwnd = ssthresh**
  - **当发生超时事件，说明已经发生拥塞，**

ssthresh = cwnd/2
cwnd = 1 MSS
重传丢失的报文

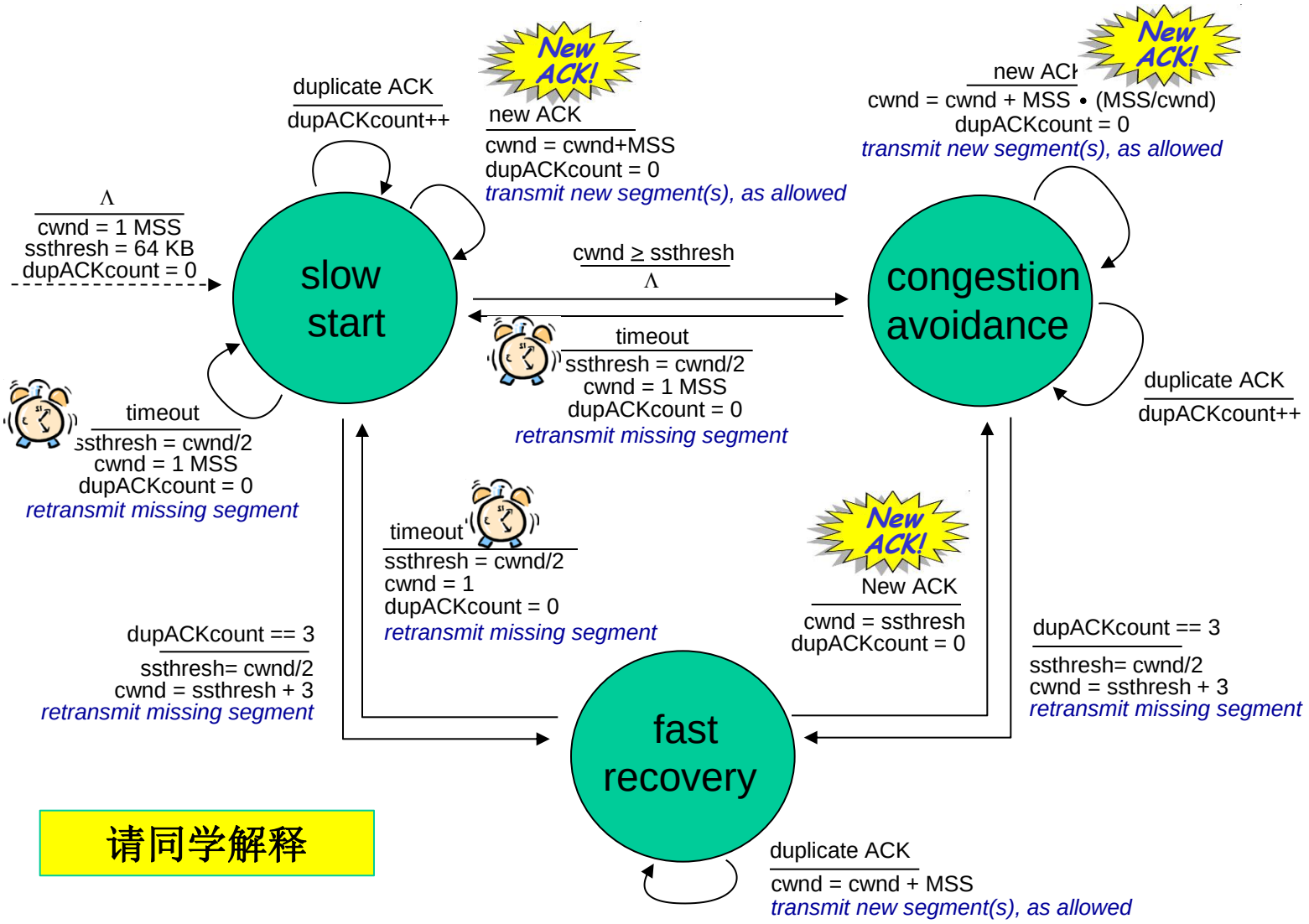
说明拥塞状态已经缓解

进入慢启动状态

TCP 拥塞控制：小结

- 当**cwnd < ssthresh**时，发送者处于**慢启动**阶段, **cwnd**指数增长
- 当**cwnd >= ssthresh**时，发送者处于**拥塞避免**阶段，**cwnd**线性增长
- 当出现**3个冗余确认**时，**ssthresh设置为cwnd/2**，且**cwnd设置为ssthresh+3*MSS**
- 当**超时**发生时， **ssthresh**设置为**cwnd/2**，并且**cwnd**设置为**1 MSS.**

Summary: TCP Congestion Control



请同学解释

TCP 发送方拥塞控制

状态	事件	TCP发送方拥塞控制动作	注释
慢启动（SS）	收到前面未确认数据的ACK	CongWin = CongWin + MSS, 如果（CongWin > 阈值） 设置状态为“拥塞避免”	导致每个RTT CongWin翻倍
拥塞避免（CA）	收到前面未确认数据的ACK	CongWin = CongWin + MSS • MSS/ CongWin	加性增，每RTT导致CongWin增加1个MSS
SS或CA	由3个冗余ACK检测到的丢包事件	阈值 = CongWin/2， CongWin = 阈值+3 MSS，设置状态为“拥塞避免”	快速恢复，实现乘性减。CongWin将不低于1个MSS
SS或CA	超时	阈值 = CongWin/2，CongWin =1 MSS，设置状态为“慢启动”	进入慢启动
SS或CA	冗余ACK	对确认的报文段增加冗余ACK计数	CongWin和阈值不改变

TCP拥塞控制整理

- 当 CongWin 低于阀值, 发送方处于慢启动阶段, 窗口指数增长.
- 当 CongWin 高于阀值, 发送方处于拥塞避免阶段, 窗口线性增长..
- 当三个重复的ACK 出现时,阀值置为CongWin/2 并且CongWin 置为阀值加上3个MSS并进入快速恢复阶段，此时每收到一个重复的ACK拥塞窗口增加1MSS，如果收到新的ACK则拥塞窗口置成阀值）.
- 当超时发生时 ， 阀值置为CongWin/2 并且CongWin 置为1 MSS.

5、对TCP吞吐量的宏观描述

- 一个TCP连接的平均吞吐量(即平均速率)可能是多少。
- 在一个特定RTT的往返时间间隔内，作为窗口长度和RTT的函数，TCP的平均吞吐量是什么？
  - 忽略慢启动

设当丢包发生时窗口长度是W

如果窗口为 W，吞吐量是 W/RTT

当丢包发生后，窗口降为 W/2，吞吐量为 W/2RTT.

一个连接的平均吞吐量为0.75 W/RTT

6、经高带宽路径的TCP

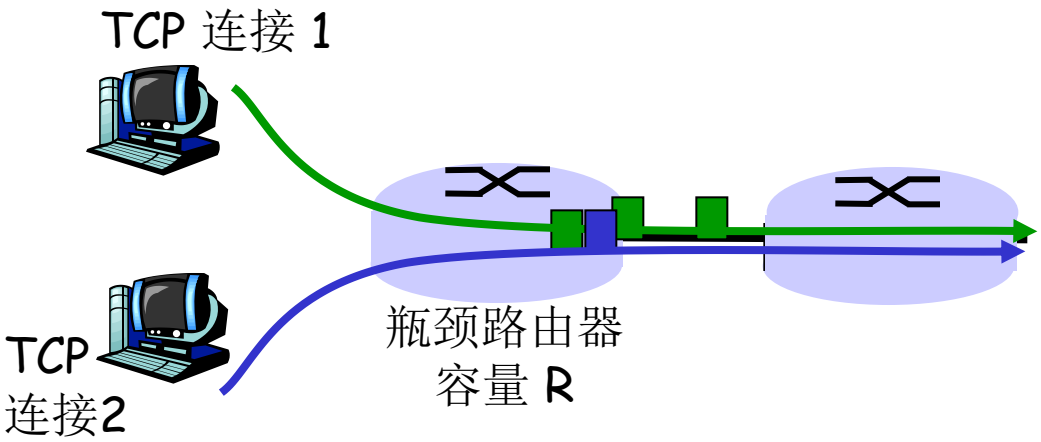
- 举例: 1500 字节的报文段, 100ms RTT, 要达到 10 Gbps 的吞吐量
- 要求窗口长度 W = 83,333 个报文段
- 根据丢包率，则一个连接的平均吞吐量为:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜丢包率 $L = 2 \cdot 10^{-10}$ *(难以达到)*
- **需要高速下的TCP新版本![Ha 2008]**

TCP 公平

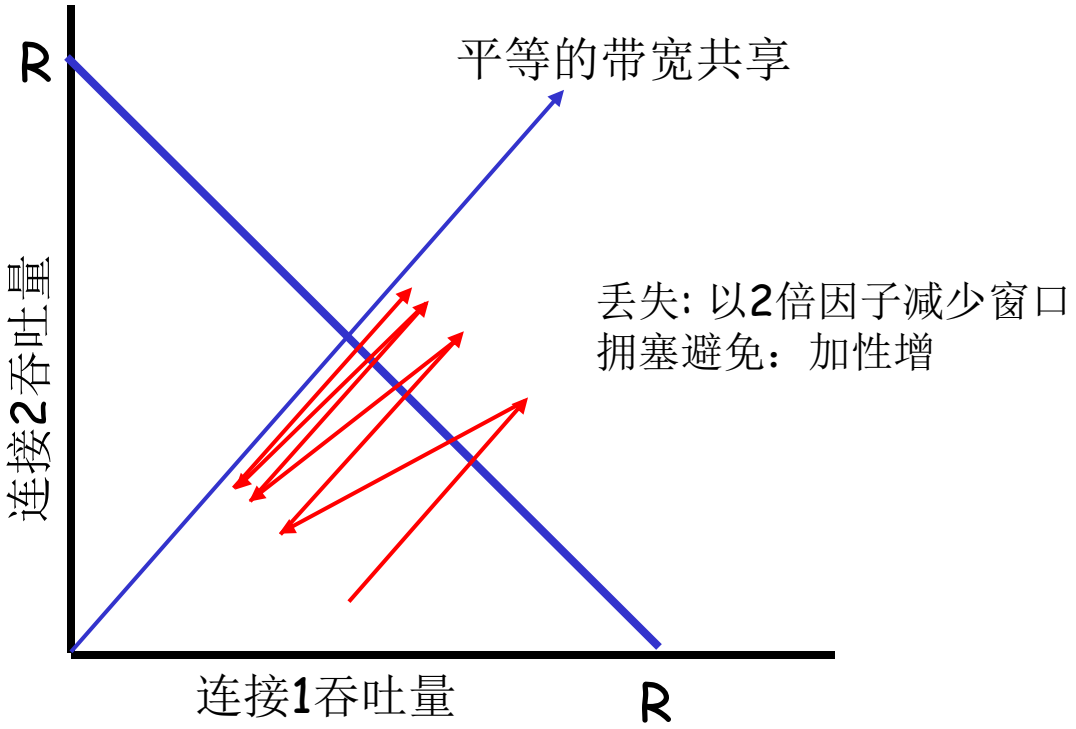
公平目标: 如果K个 TCP 会话 共享带宽为 R的链路瓶颈,
每个会话应有R/K的平均链路速率



为什么TCP能保证公平性?

两个竞争会话:

- 随着吞吐量的增加，按照斜率1加性增加
- 等比例地乘性降低吞吐量



公平性(续)

公平性和UDP

- 多媒体应用通常不用TCP
  - 不希望拥塞控制抑制速率
- 使用UDP
  - 音频/视频以恒定速率发送, 能容忍报文丢失
- 研究领域:

  TCP友好(TCP friendly)

公平性和并行TCP 连接

- 不能防止2台主机之间打开多个并行连接.
- Web浏览器以这种方式工作
- 例子:支持9个连接的速率R的链路:
  - 新应用请求一个TCP连接, 则得到 R/10的带宽。
  - 新应用请求11个TCP连接, 则得到R/2 的带宽!

怎么在UDP的基础上实现与TCP的友好性是曾经的一个热点问题

Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"

Homework Exercises

P40. Consider Figure 3.58. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.

a. Identify the intervals of time when TCP slow start is operating.

b. Identify the intervals of time when TCP congestion avoidance is operating.

c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

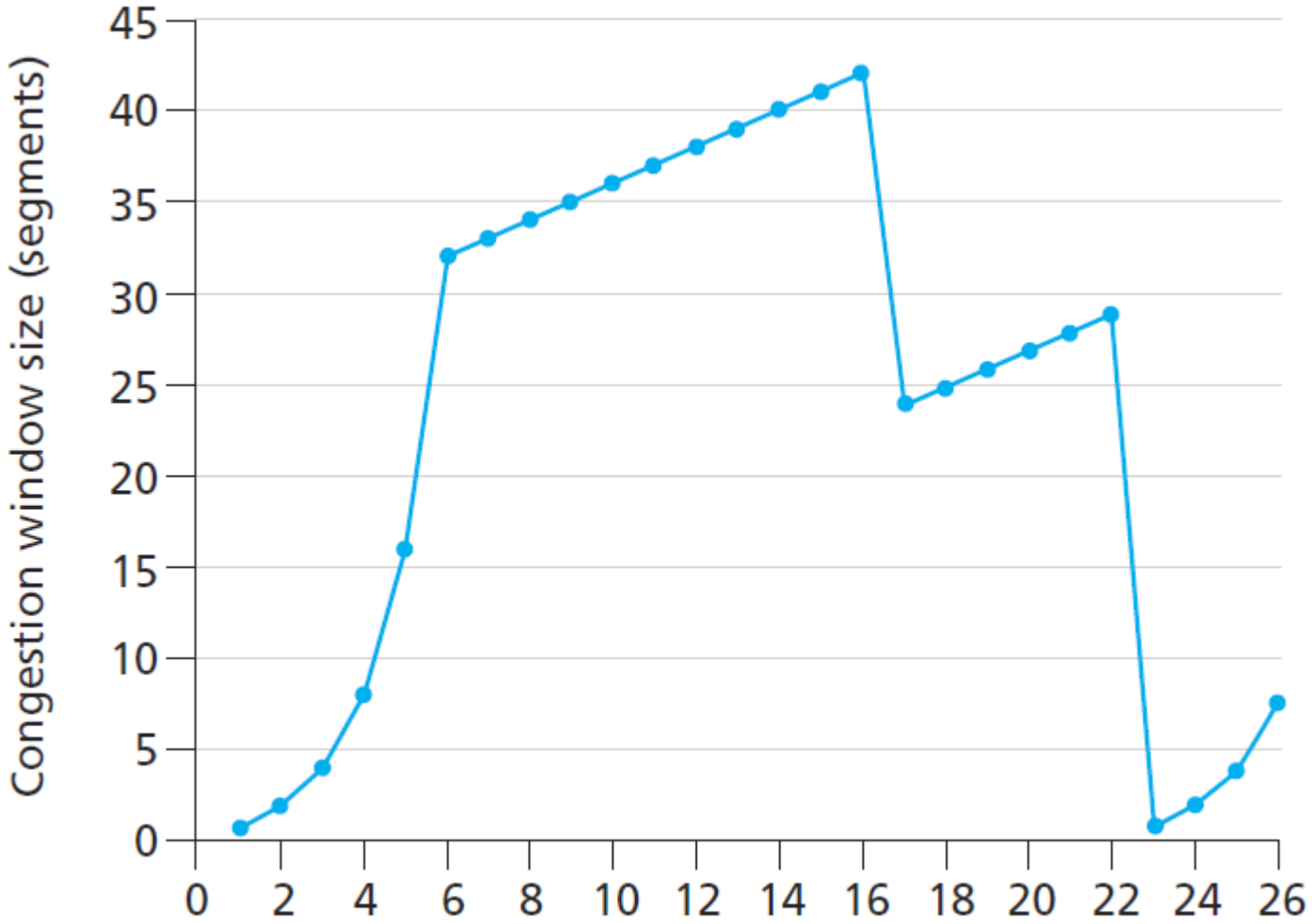
e. What is the initial value of `ssthresh` at the first transmission round?

f. What is the value of `ssthresh` at the 18th transmission round?

g. What is the value of `ssthresh` at the 24th transmission round?

h. During what transmission round is the 70th segment sent?

i. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of `ssthresh`?



j. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the `ssthresh` and the congestion window size at the 19th round?

k. Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?

- 下列关于TCP拥塞控制机制描述错误的是_____。

A. 当TCP连接刚建立时，处于慢启动状态，此时，cwnd的值为1个MSS，每收到1个ACK确认，将cwnd的值增加1个MSS；

B. TCP使用的是端到端拥塞控制机制，而不是网络辅助的拥塞控制机制；

C. 当发送端连续收到3个重复的ACK确认，进入快速恢复状态。

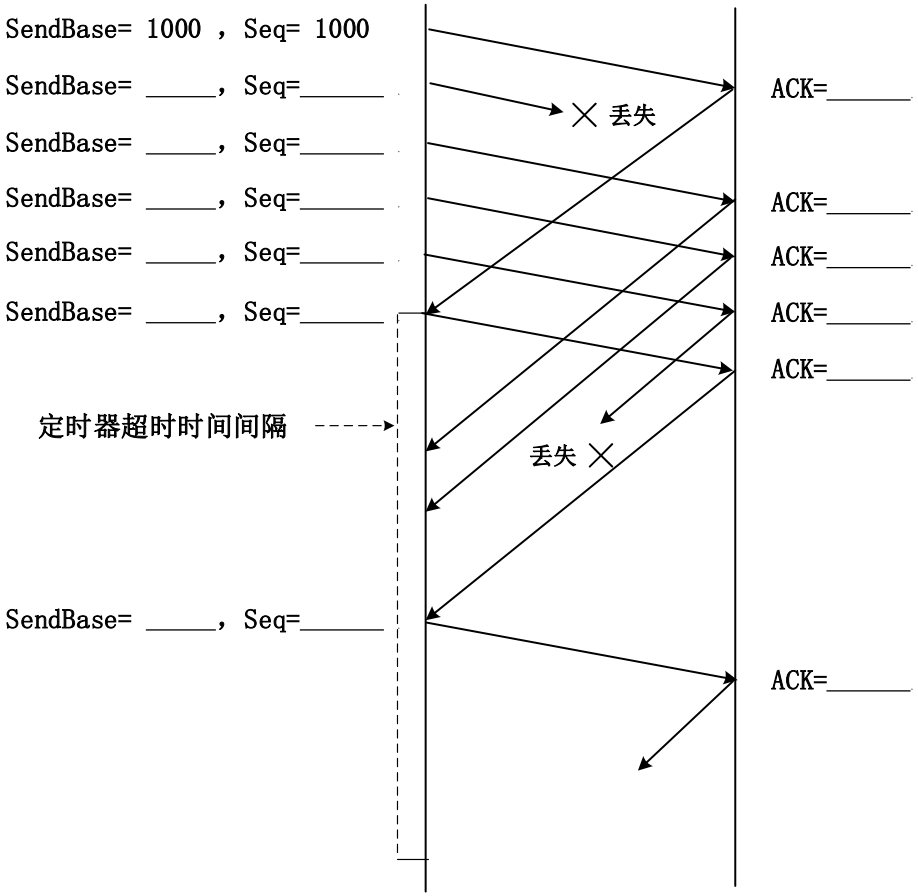
D. 当cwnd的值大于慢启动阈值ssthresh时，进入拥塞避免状态，在该状态，每收到1个ACK确认，将cwnd的值增加1个MSS。

- 下列哪项不是TCP协议的特性_____。

A. 提供可靠服务　　B. 提供无连接服务

C. 提供端到端服务　　D. 提供全双工服务

假设主机A和主机B之间建立了TCP连接，并且主机A有大量的数据需要向B发送。发送发主机A支持快速重传，而接收方B会缓存正确接收但失序的报文段，每次发送的报文段的数据字段长度都为100字节。下面的描述中，Seq代表序号，ACK代表确认号，发送方窗口的基序号为SendBase，发送方窗口长度为500字节，请填写下图中各变量的值



实验2—思考

任务1：任务1（研究TCP)：在这个实验中，同学们将使用web浏览器访问来自某Web服务器的一个文件。能够从该web服务器下载一个Wireshark可读的分组踪迹，记载你从服务器下载文件的过程。在这个服务器踪迹文件里，你将发现并访问该web服务器所产生的分组。你将分析客户端和服务器端踪迹文件，以探索TCP协议的各种细节，特别是将评估在你的计算机与web服务器之间TCP连接的性能。你将跟踪TCP窗口行为、推断分组丢失、重传、流控、拥塞控制行为并估计往返时间。

任务2（研究UDP)：在这个实验中，同学们将进行分组捕获并分析那些使用UDP的你喜爱的应用程序(例如，DNS或skype这样的多媒体应用)。如我们在3.3节中所学的那样，UDP是一种简单的、不提供不必要服务的运输协议。在这个实验中，将研究在UDP报文段中的各首部字段以及校验和计算。