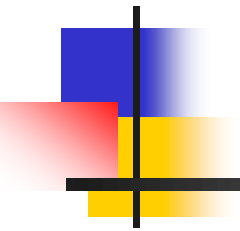
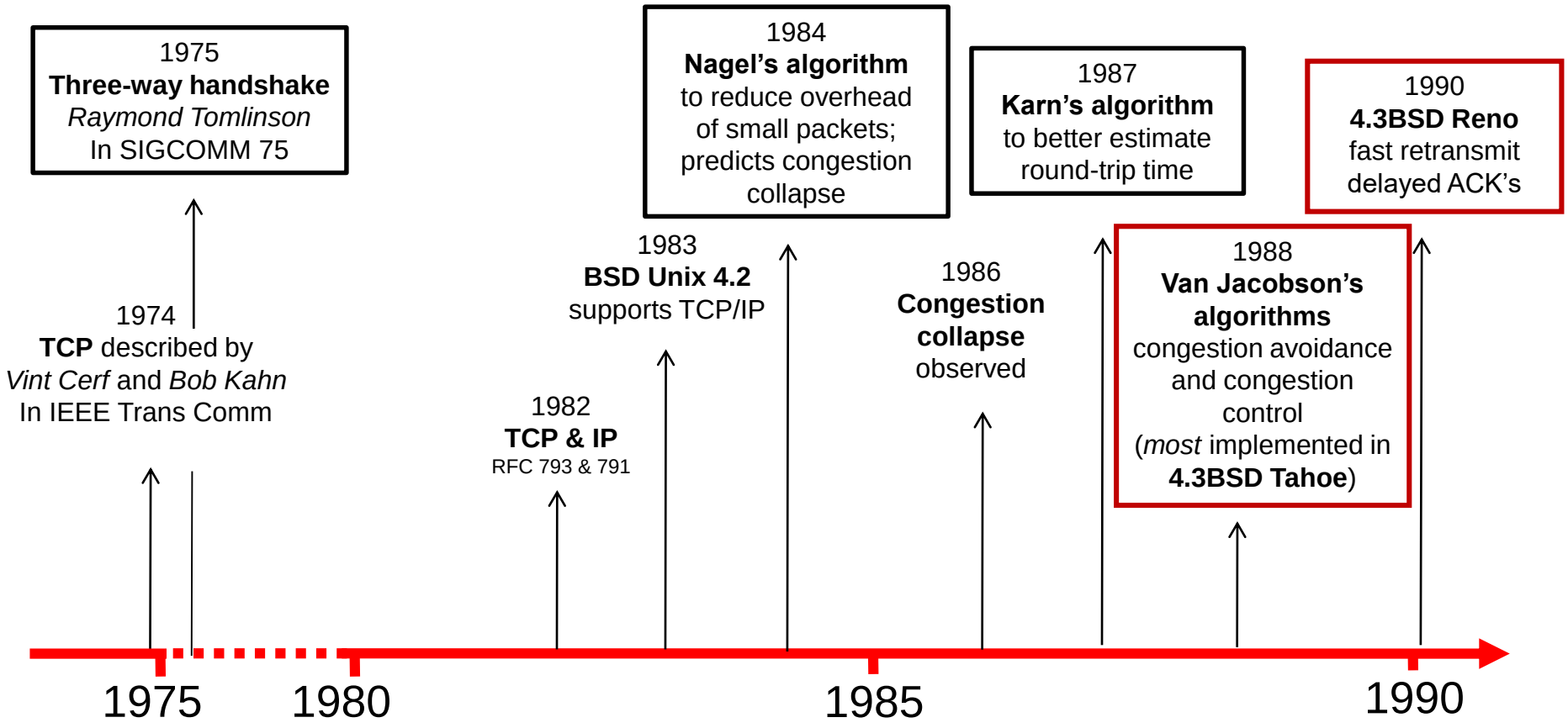


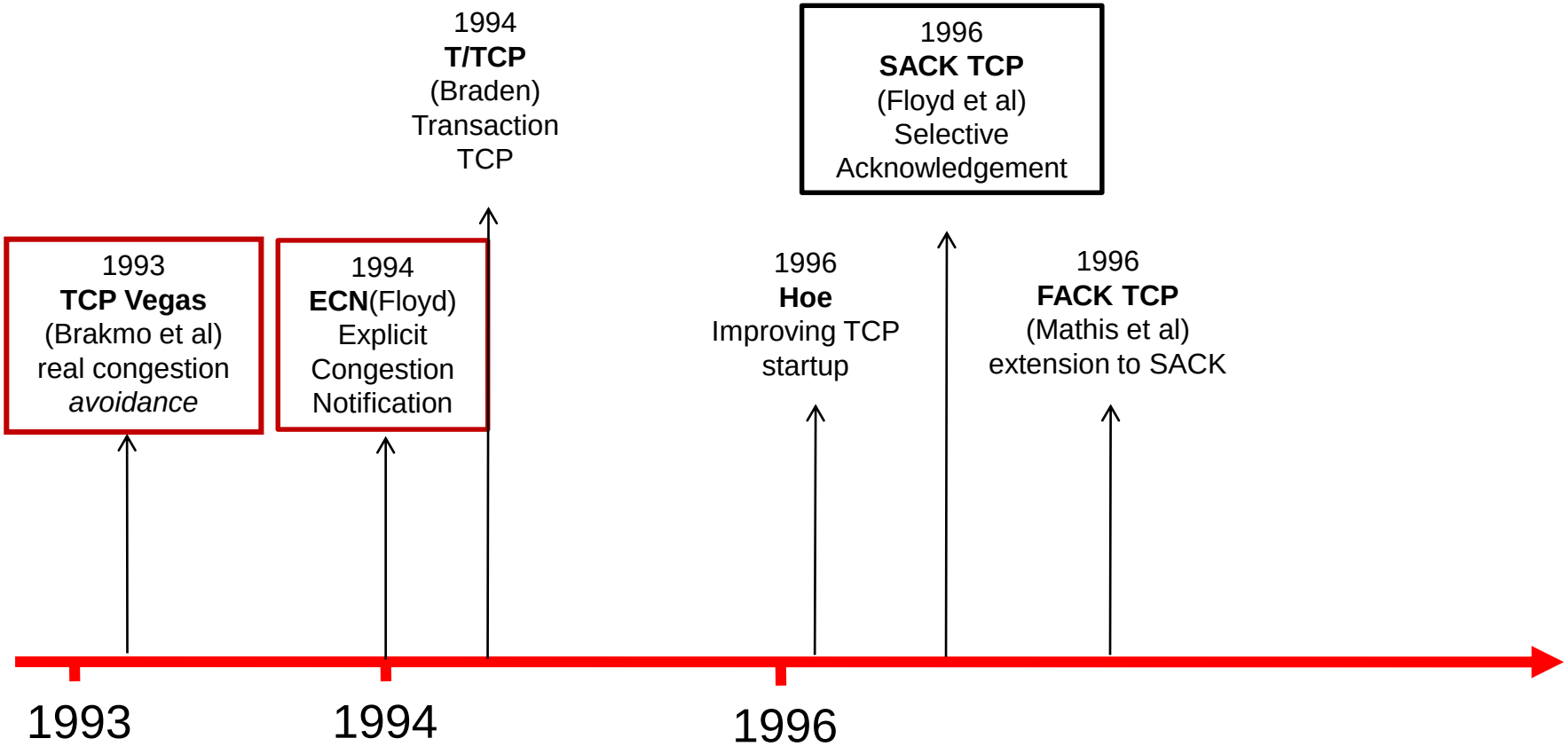
TCP协议与拥塞控制

刘志敏
liuzm@pku.edu.cn

TCP：演进过程


1975
Three-way handshake
Raymond Tomlinson
In SIGCOMM 75
1974
TCP described by
Vint Cerf and Bob Kahn
In IEEE Trans Comm
1982
TCP & IP
RFC 793 & 791
1983
BSD Unix 4.2
supports TCP/IP
1984
Nagel's algorithm
to reduce overhead
of small packets;
predicts congestion
collapse
1986
Congestion
collapse
observed
1987
Karn's algorithm
to better estimate
round-trip time
1988
Van Jacobson's
algorithms
congestion avoidance
and congestion
control
(most implemented in
4.3BSD Tahoe)
1990
4.3BSD Reno
fast retransmit
delayed ACK's
1975
1980
1985
1990

TCP：演进过程

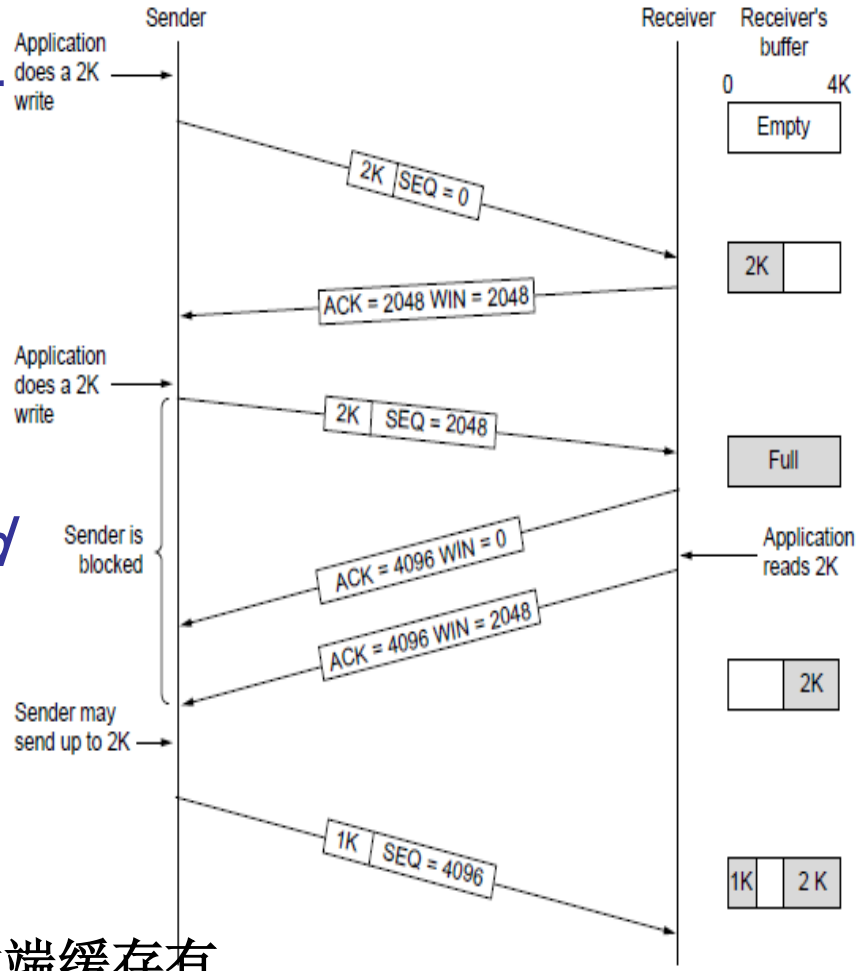

1994
T/TCP
(Braden)
Transaction
TCP
1996
SACK TCP
(Floyd et al)
Selective
Acknowledgement
1993
TCP Vegas
(Brakmo et al)
real congestion
avoidance
1994
ECN(Floyd)
Explicit
Congestion
Notification
1996
Hoe
Improving TCP
startup
1996
FACK TCP
(Mathis et al)
extension to SACK
1993
1994
1996

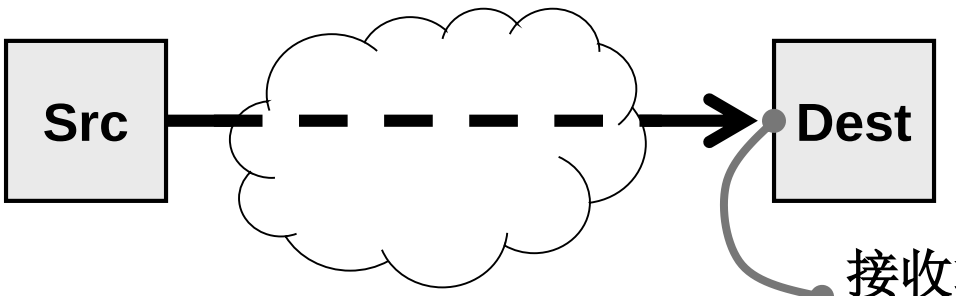
提纲

- 拥塞控制
- TCP的拥塞控制：慢启动、有效的传输控制窗口
- TCP的定时器管理
- 无线链路上的TCP及性能测试

流量控制与拥塞控制

- TCP提供流量控制，防止数据在接收缓存中溢出
- 接收端在确认帧中向发送端声明接收窗口（*wnd*）
- 发送端可发送的未被确认的数据量不能超过*wnd*

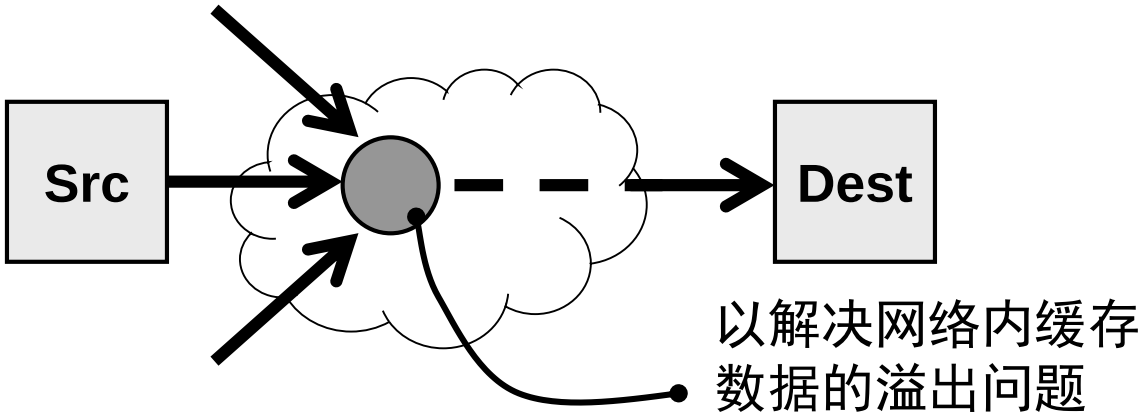




接收端缓存有限，必须限制发端速率

流量控制与拥塞控制

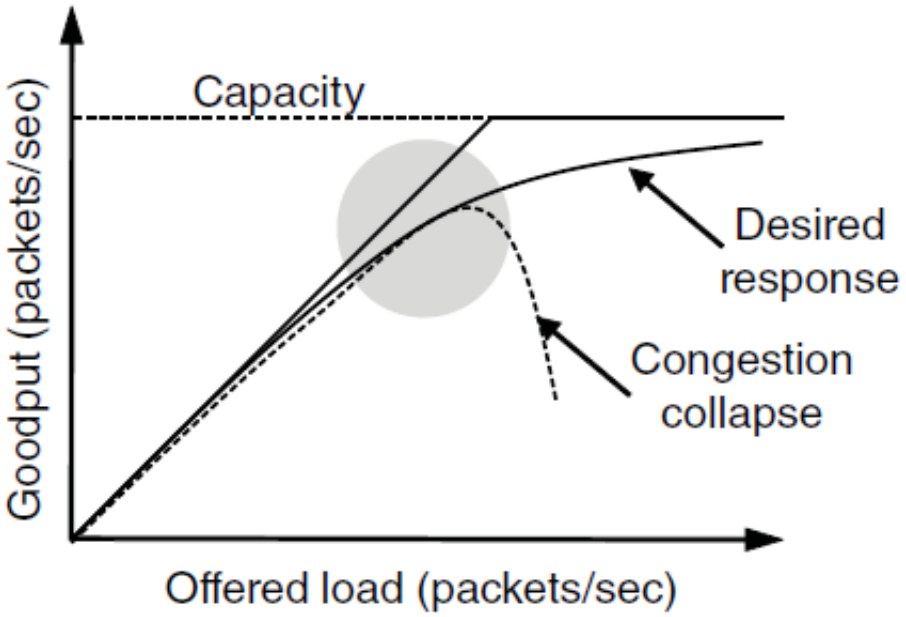
- 为降低网络内部拥塞，在TCP中增加拥塞控制
- 面临的主要问题
  - 网络拥塞是因多个数据源发送数据而引起的，需要协调多个发送端
  - 必须依靠一种间接的拥塞测量方法
- 在发送端实现

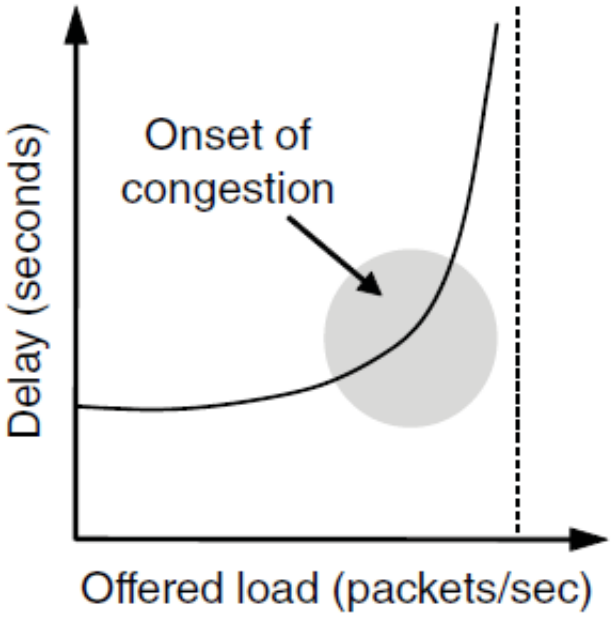


拥塞问题

- 拥塞原因：源主机数太多，向网络发送数据的速率过高
- 这与流量控制不同！流量控制是接收端对发送端的控制，不能解决拥塞问题
- 拥塞控制：
  - 理想的带宽分配
  - 调整发送速率

理想的带宽分配(1)

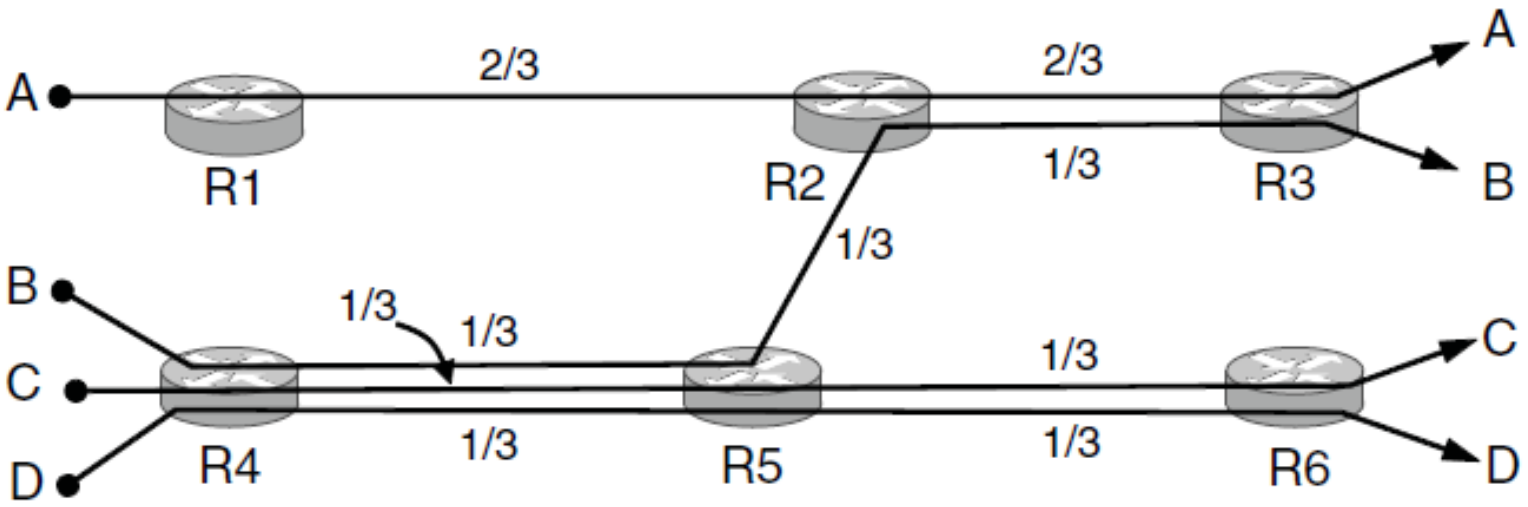




(a)负载与吞吐量的关系

(b) 负载与延迟的关系

理想的带宽分配(2)



理想的带宽分配：**max-min fairness**最大-最小公平性，即分配一个流的带宽时，若不减少另一个流的带宽就无法增加该流的带宽。

例如：流B, C, D的带宽均为1/3；流B在R2-R3的带宽为1/3，则流A的为2/3；若流B增大为1/2则必然降低流C，D的带宽，因此该带宽分配满足最大-最小公平性

最大最小公平算法（max-min fairness ）

- 给每个用户公平分配期望且可满足的最小需求，然后将未使用的资源均分给需要“更多资源”的用户。
- 算法的定义如下：
  - 资源按需求递增的顺序分配
  - 不存在用户得到的资源超过其需求
  - 未得到满足的用户等价分享资源
- 设用户1, …, *n*的资源需求分别为$x_1, x_2, \ldots, x_n$，$x_1<=x_2<=\ldots<=x_n$，服务能力为C；初始将*C/n*分配给n个用户；若$C_1=(C/n-x_1)>0$，则将C_1/(n-1)分配给(n-1)个用户；…… 。该过程结束时，每个用户得到的资源<=其需求，且未满足需求的用户资源>=其他用户的最多资源。
- 不满足需求的用户数最大化的最小资源分配
  - 保证最大公平性的最小资源分配

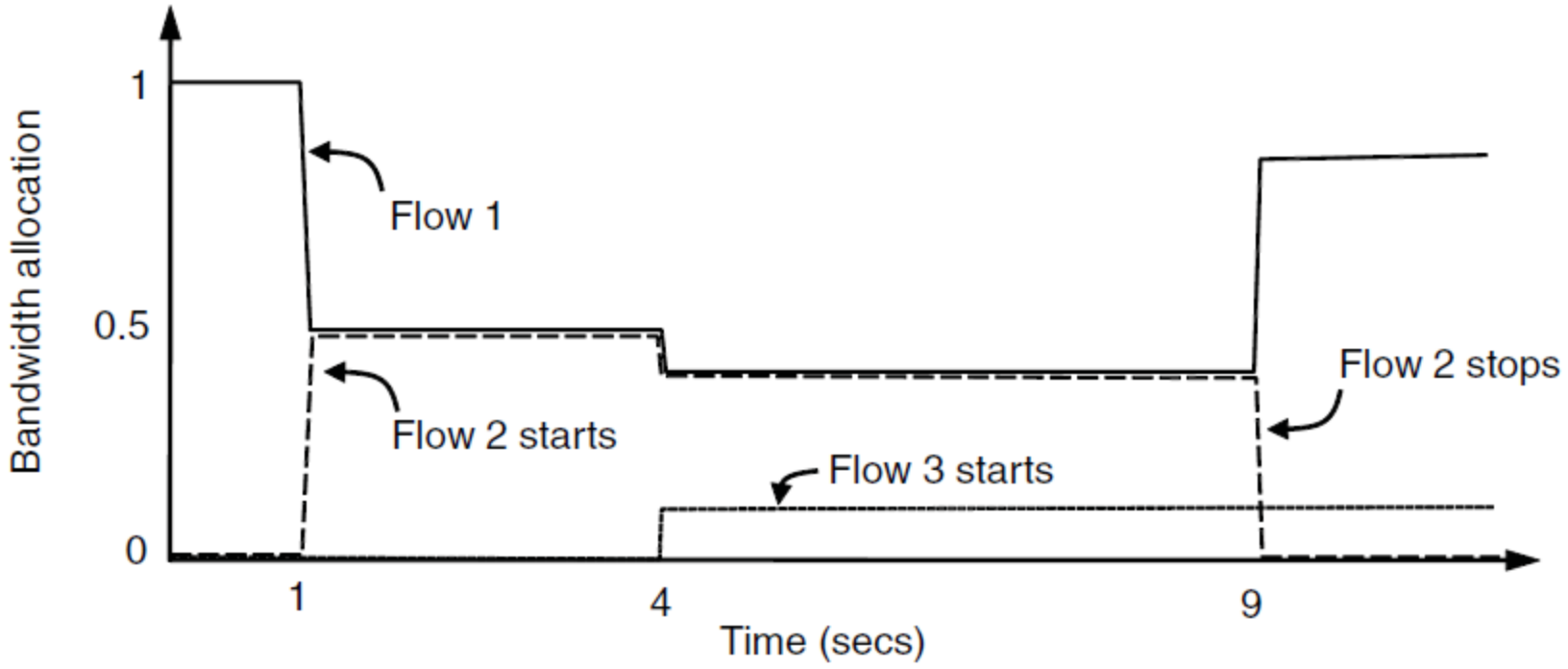
最大最小公平算法示例

- 四个用户集合的资源需求分别是2，2.6，4，5，其资源总数为10。求最大最小公平分配。
- 方法：

	用户1	用户2	用户3	用户4	均分资源
需求	2	2.6	4	5	10/4=2.5
第1轮	2	2.5	2.5	2.5	(2.5-2)/3=0.166
第2轮		0.1	0.166	0.166	(0.166-0.1)/2=0.033
第3轮			0.033	0.033	
结果	2	2.6	2.7	2.7	

- 结果：
  - 得到的<=需求；
  - 未满足需求的>=已分配的；

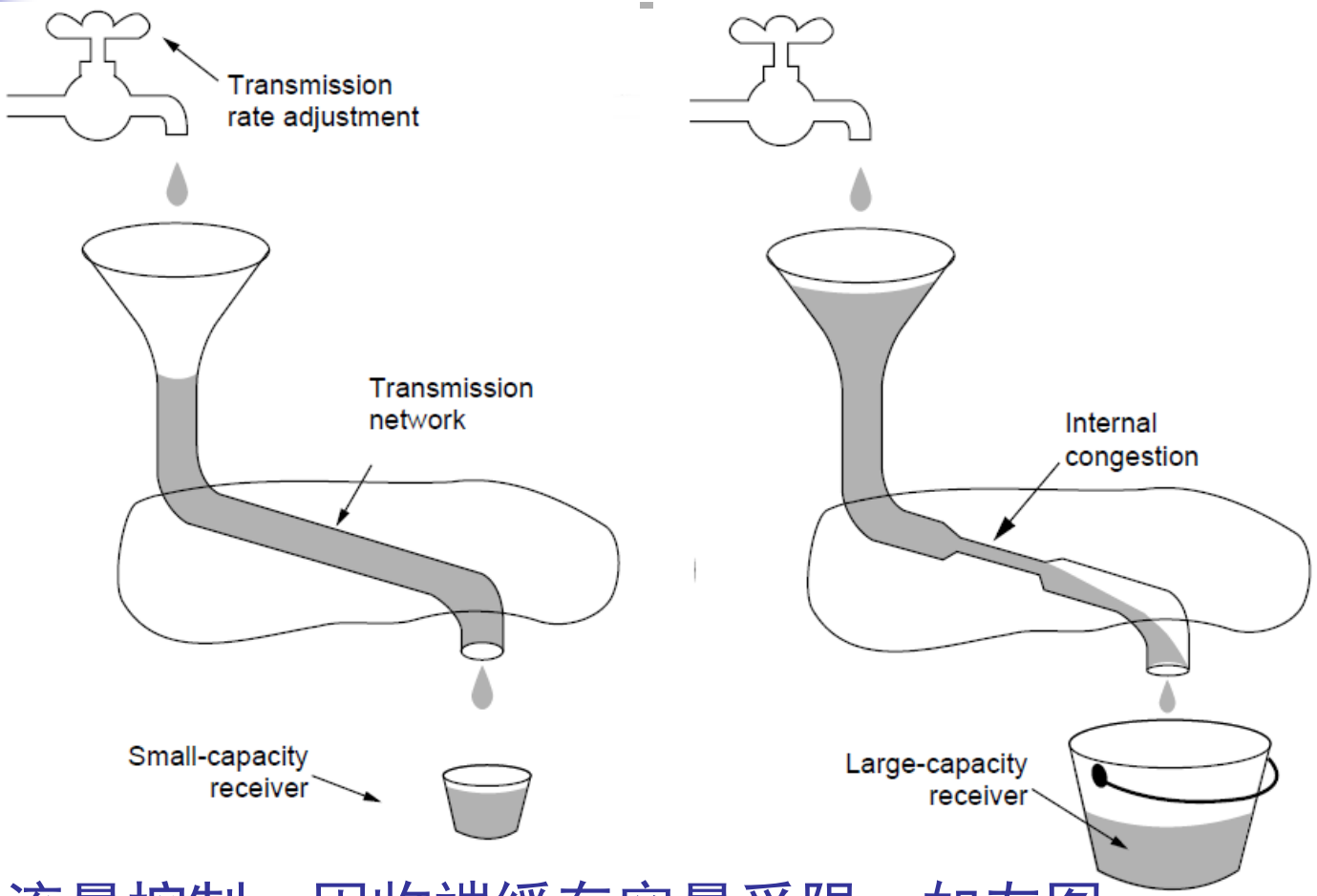
理想的带宽分配(3)



带宽随时间动态变化

当流1,2需要>1/2时，均分；当流3需要较少的，满足

调整发送速率(1)



（1）流量控制，因收端缓存容量受限，如左图

（2）若收端容量很大，但通过低速网络可能发生拥塞，因此也要限制发送速率，避免在流动过程中溢出

调整发送速率(2)

Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

一些TCP的拥塞控制协议

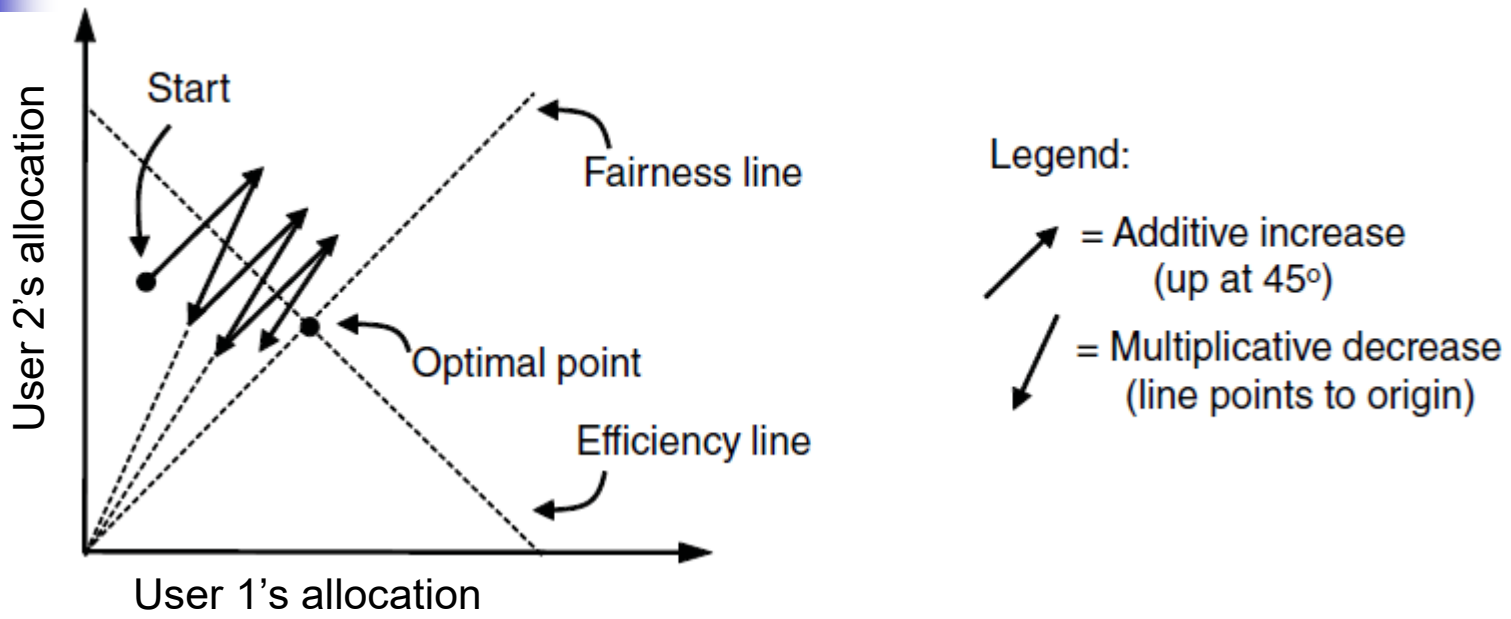
XCP：路由器通知源端拥塞及其发送速率

ECN：路由器仅设置拥塞标志通知源端拥塞

FAST TCP：测量RTT作为拥塞指示

CUBIC TCP：用丢包作为拥塞信号

调整发送速率(3)：调整算法？



最优点：公平线（均分带宽，45°角）与效率线（分配全部带宽）的焦点

带宽加性增减（45°线变化），无法达到最优点

带宽乘性（按比列）增减：沿指向圆点的线变化

只有带宽加性增加乘性减少(Additive Increase Multiplicative Decrease:AIMD)才能从起点到达最优点

拥塞控制

- 如何检测拥塞？拥塞的现象：
  - 分组丢失(因分组在路由器上的缓存不够而导致)，例如；序号不连续、定时器超时
  - 时延增加(数据在路由器缓存中排队)：检测并估计延迟
- 发送端
  - 维护流量控制窗口r*wnd* ，再维护一个拥塞窗口(*cwnd*)，
  - 发送窗口swnd为 swnd = Min [rwnd, cwnd]
  - 需要一种机制设置*cwnd* 以避免网络拥塞
- 主要问题：
  - 如何确定拥塞窗口的值
  - 如何判定网络拥塞的程度

提纲

- 拥塞控制
- TCP拥塞控制：慢启动、有效的传输控制窗口
- TCP的定时器管理
- 无线链路上的TCP及性能测试

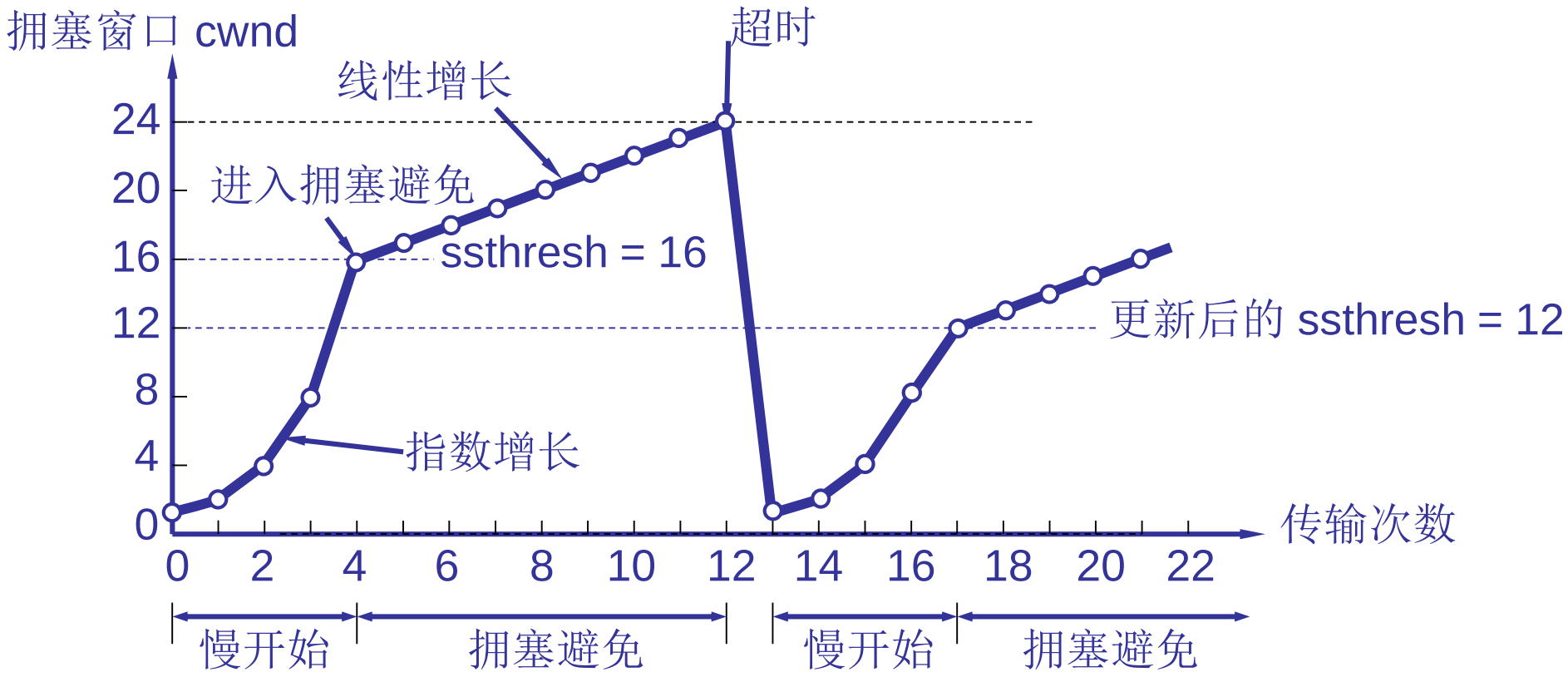
TCP拥塞控制

- 几种解决方法
  - TCP Tahoe（1988）
  - TCP Reno （1990）：增加快重传与快恢复
  - TCP Vegas（1993）

拥塞控制：TCP Tahoe（Jacobson）

- 慢启动（指数增加）
- 拥塞避免(线性增加)
- 发生拥塞时，拥塞门限减半（乘性减少）
- 若数据发送成功，则增加拥塞窗口
  - 慢启动阶段(指数增加)
    - 若*cwnd* ≤ *ssthresh*（慢启动门限），则为慢启动阶段
    - 每次收到ACK，则*cwnd* 增加一个报文段长度（窗口指数增长）
  - 拥塞避免(线性增加)
    - 每次收到确认，*cwnd* = *cwnd* + (1/*cwnd*)
      ⇒ 窗口加性增加（每次RTT，增加一个报文段）
- 若定时器超时——用报文段丢失来指示网络拥塞，则
  - 慢启动门限 *ssthresh*= *cwnd* /2, *cwnd* =1 进入慢启动过程

慢启动和拥塞避免算法的举例

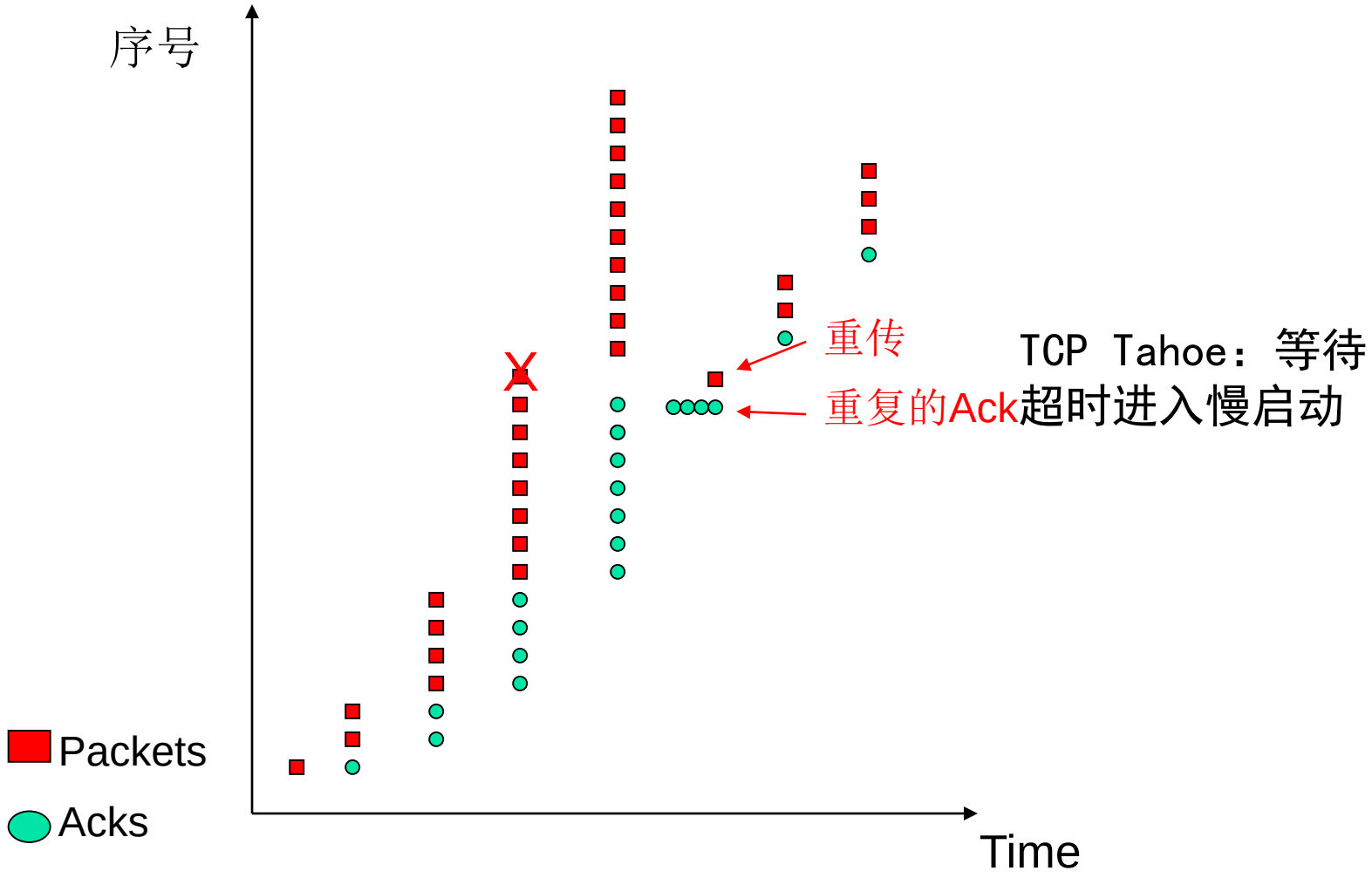


实际拥塞窗口值为cwnd*MSS，此处省略了MSS

初始时 cwnd=1，慢启动门限 ssthresh=16

swnd=Min[rwnd, cwnd]，若接收窗口rwnd足够大，则swnd=cwnd

分组丢失，导致重复的ACK



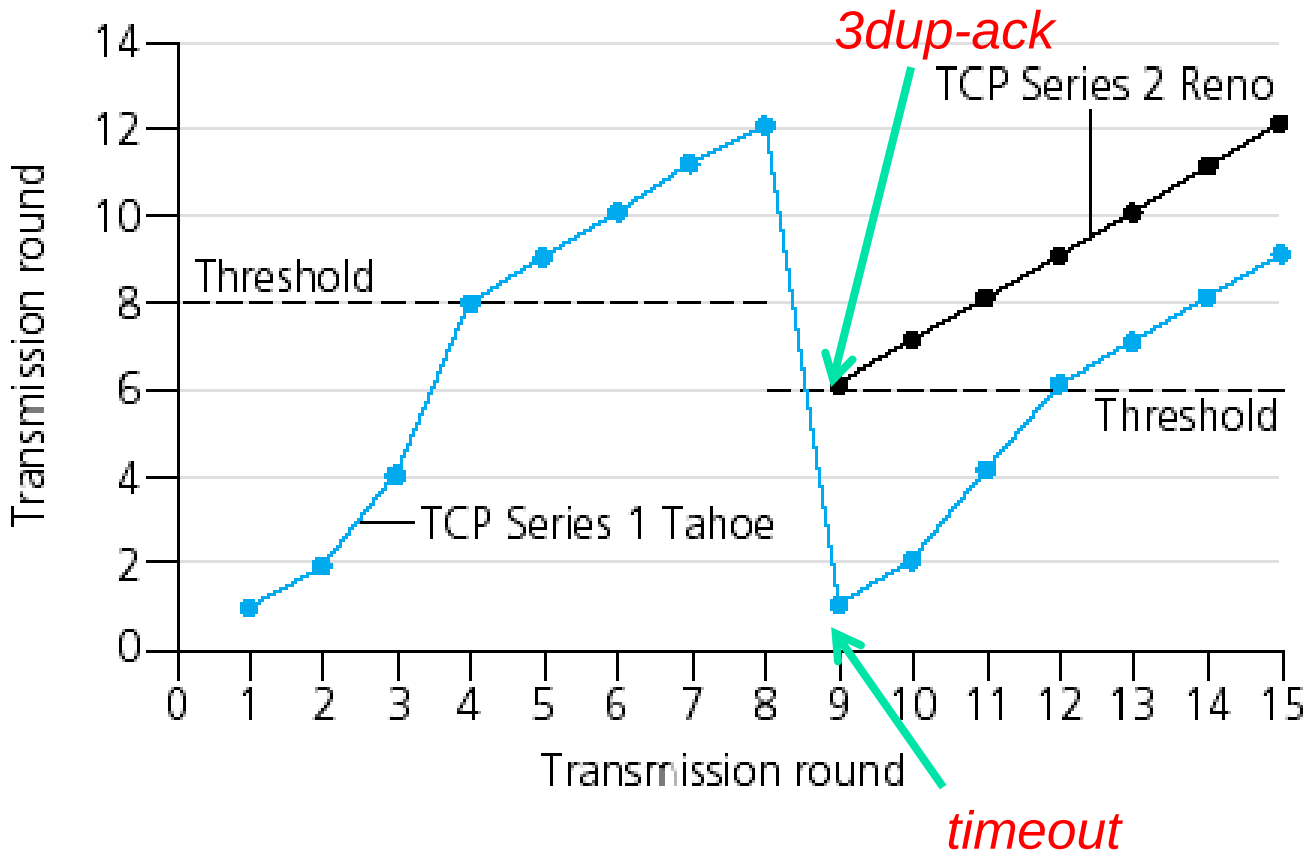
TCP Reno (1990)

- 增加快重传及快恢复
- 何谓重复ACK?
  - 重复地确认同一报文段序号
- 何时发生重复的ACK?
  - 报文丢失
  - 报文错序(假设分组错序现象不常见)
- 快重传
  - 用3次重复的ACK指示报文段丢失
  - 不等超时定时器到就重传丢失的报文段
  - 慢启动门限值为拥塞窗口的1/2
- 快恢复
  - 快重传之后的临时模式，拥塞窗口减半，对重复确认的计数，直到拥塞网络内的报文数降为新阈值

TCP Reno

- 启动拥塞的方法
  - 用定时器超时或收到重复的ACK来指示分组丢失
  - 若定时器超时，则乘性减少cwnd
  - 若收到3次重复的ACK，则适当调整 cwnd
- 若发生分组丢失，则减少拥塞窗口
  - 因超时： *cwnd* = 1 并进入慢启动过程
  - 因重复的ACK： *cwnd* = *cwnd*/2
- 若数据发送成功（收到ACK），则增加拥塞窗口
  - 慢启动阶段(指数增加)
  - 拥塞避免(线性增加)

TCP Reno的拥塞窗口

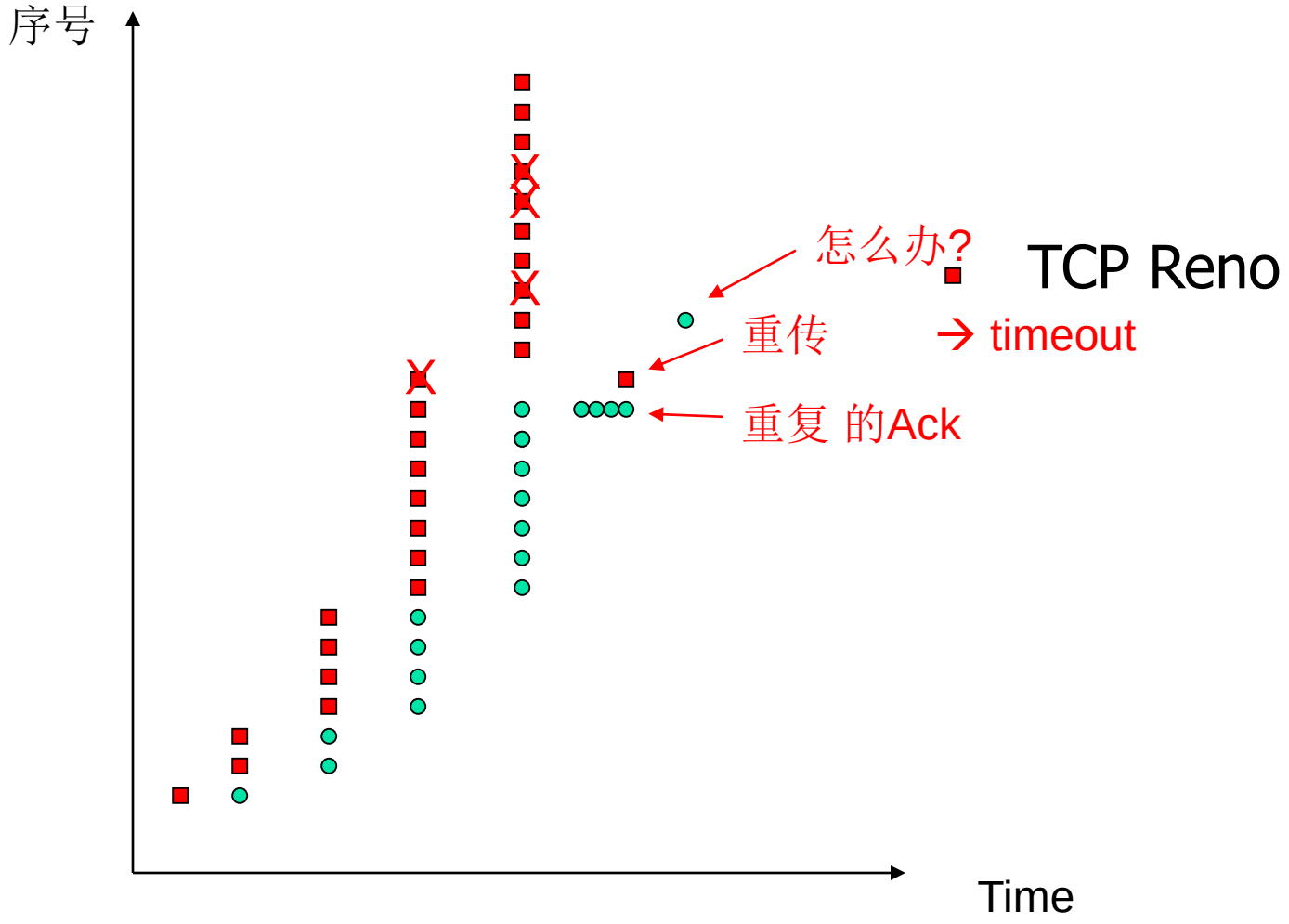


快重传与快恢复之后，慢启动的门限值减半

快恢复算法

- 当收到3个重复ACK时，则慢启动门限 ssthresh= cwnd /2，拥塞窗口 cwnd = ssthresh + 3 × MSS
- 若收到*n* 个重复 ACK （*n* > 3），则 cwnd =ssthresh + *n*×MSS
- 若发送窗口还允许发送报文段，就进入拥塞避免阶段
- 若收到新的ACK，cwnd=ssthresh，进入拥塞避免阶段

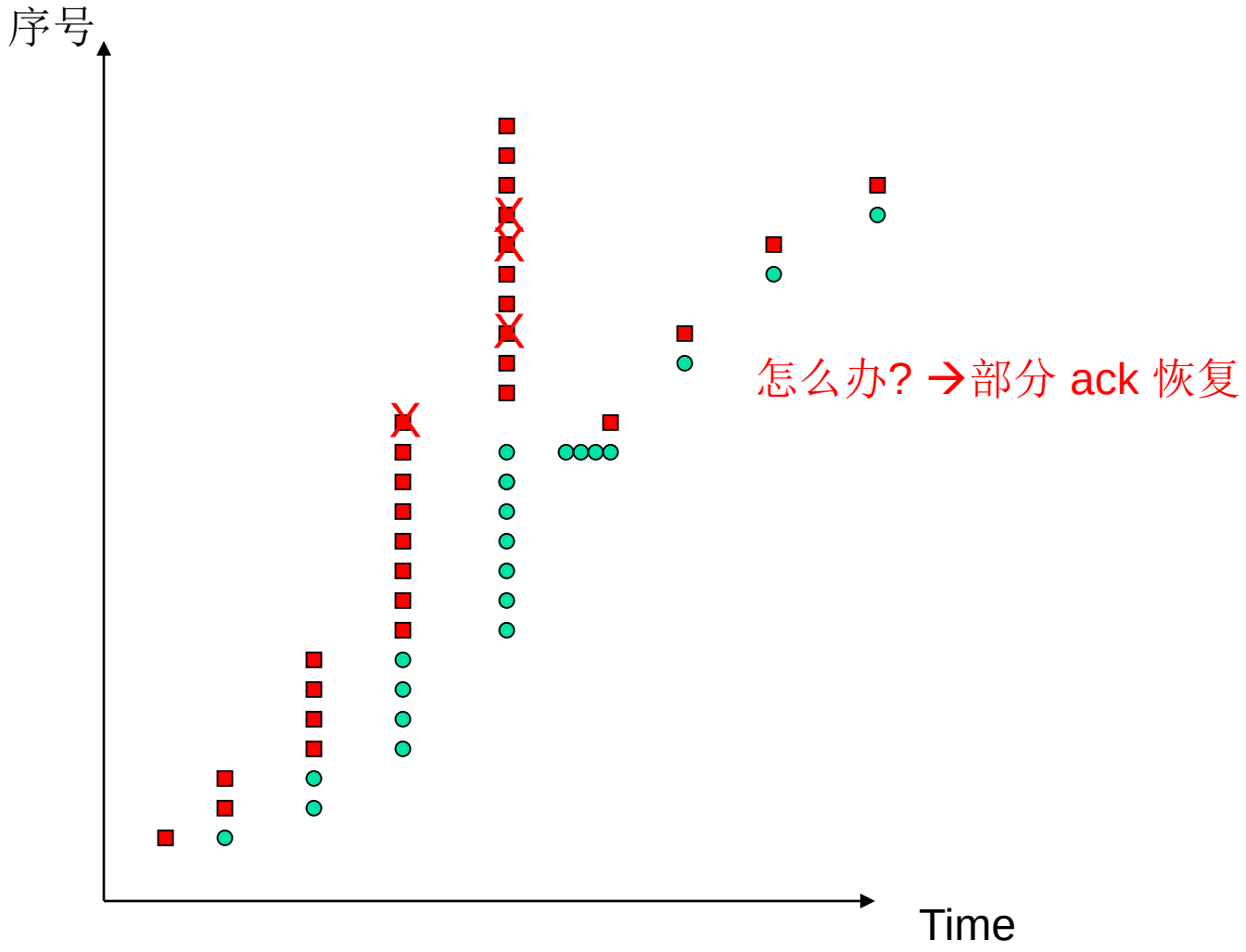
丢失多个报文的情况



TCP NewReno

- 何时出现Reno超时?
  - 首次报文段丢失后重复的ACK少于3个
  - 当ACK丢失时
- 快重传之后收到ACK，应视为再次发生分组丢失
- NewReno快恢复丢失的报文段
  - 快重传之后收到ACK，则立即发送当前报文
  - 恢复时间仅需要一个RTT

TCP NewReno



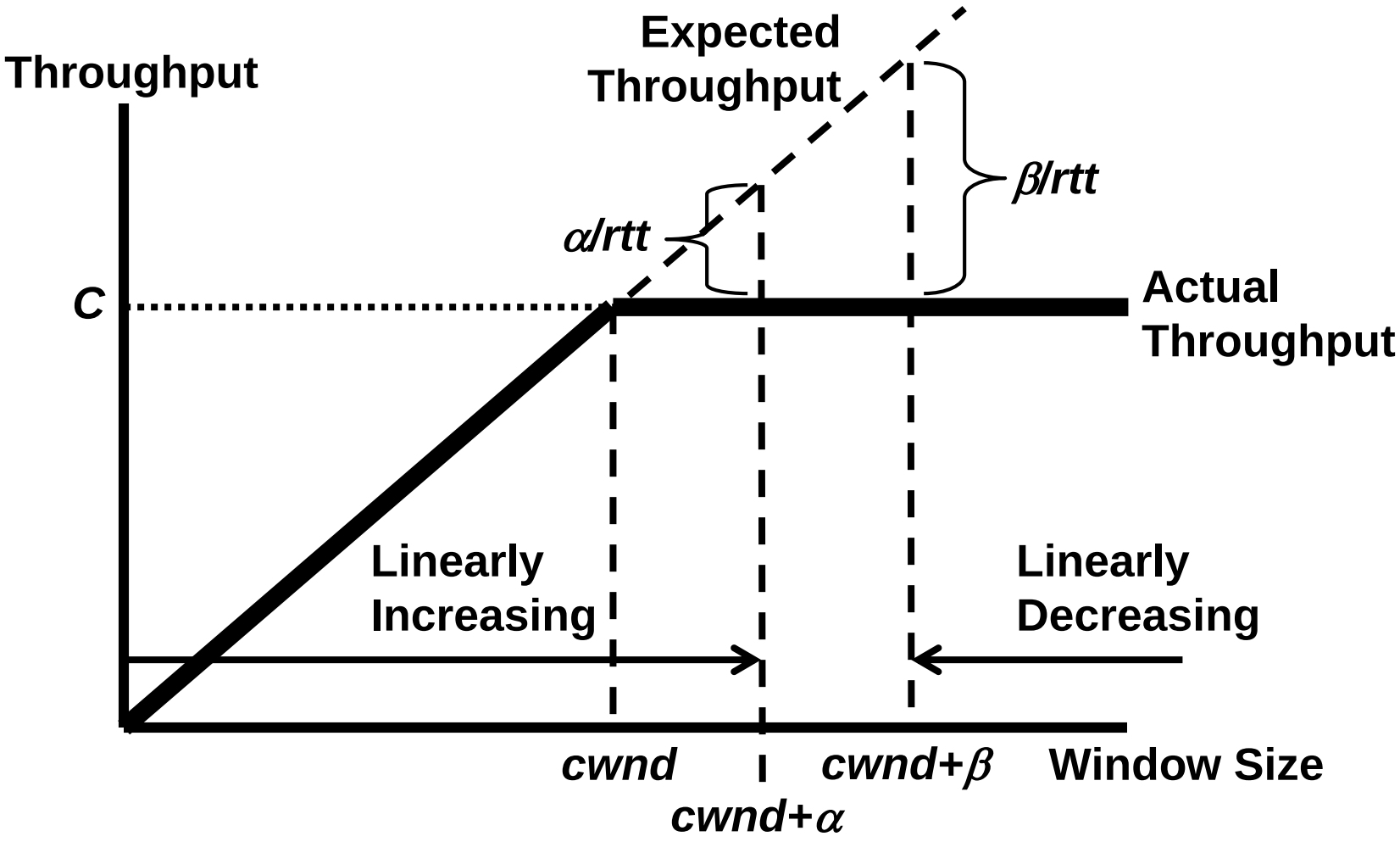
TCP Vegas (1)

- 拥塞窗口为期望速率与实际数据速率之差
  - 目的是控制网络缓存中的数据量
- 定义
  - *Cwnd*: 拥塞窗口
  - *rtt**: 最小RTT (无拥塞情况)
  - *Rtt*: 实际的RTT (有拥塞时)
  - *Diff*: 队列中缓存数据的估计
  - α: *diff*的低门限 ($diff > \alpha$)
  - β: diff高低门限 ($diff < \beta$)
- $diff = (cwnd/rtt^* - cwnd/rtt) \times rtt^*$

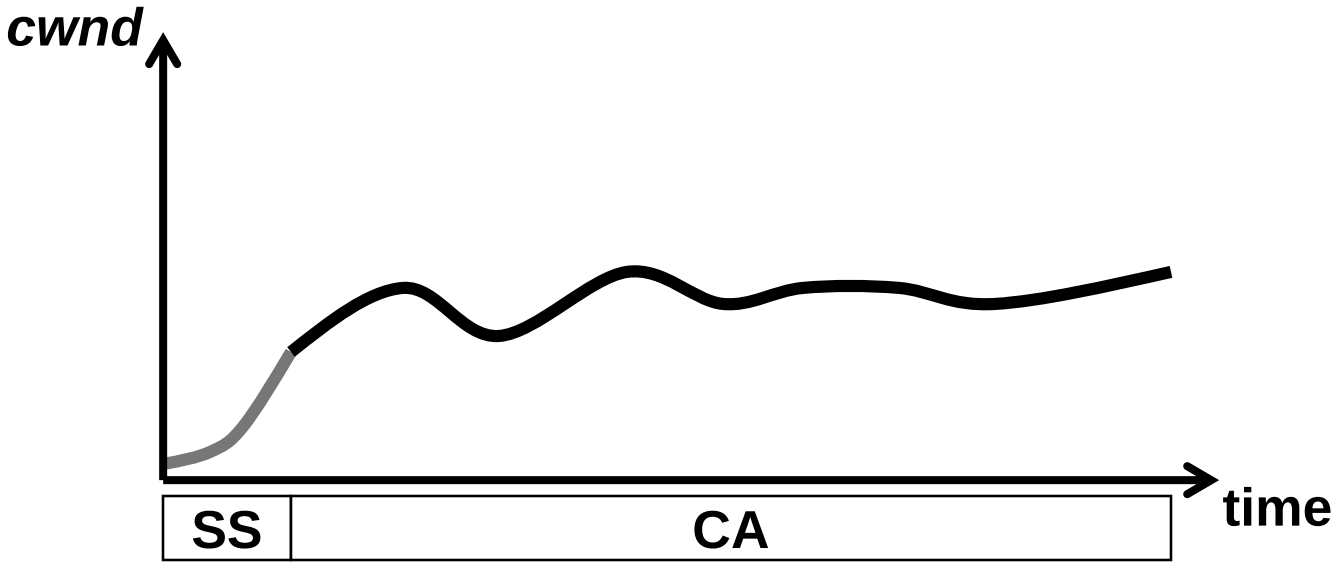
TCP Vegas (2)

- $diff = (cwnd/rtt^* - cwnd/rtt) \times rtt^*$
- TCP Vegas
  - If $diff < \alpha$, increase *cwnd* by 1
  - If $diff > \beta$, decrease *cwnd* by 1
  - Otherwise ($\alpha \leq diff \leq \beta$), *cwnd* is not changed
- TCP Vegas提供一种对拥塞的主动响应机制
  - 根据延迟时间的变化逐渐调整

拥塞控制: TCP Vegas (3)

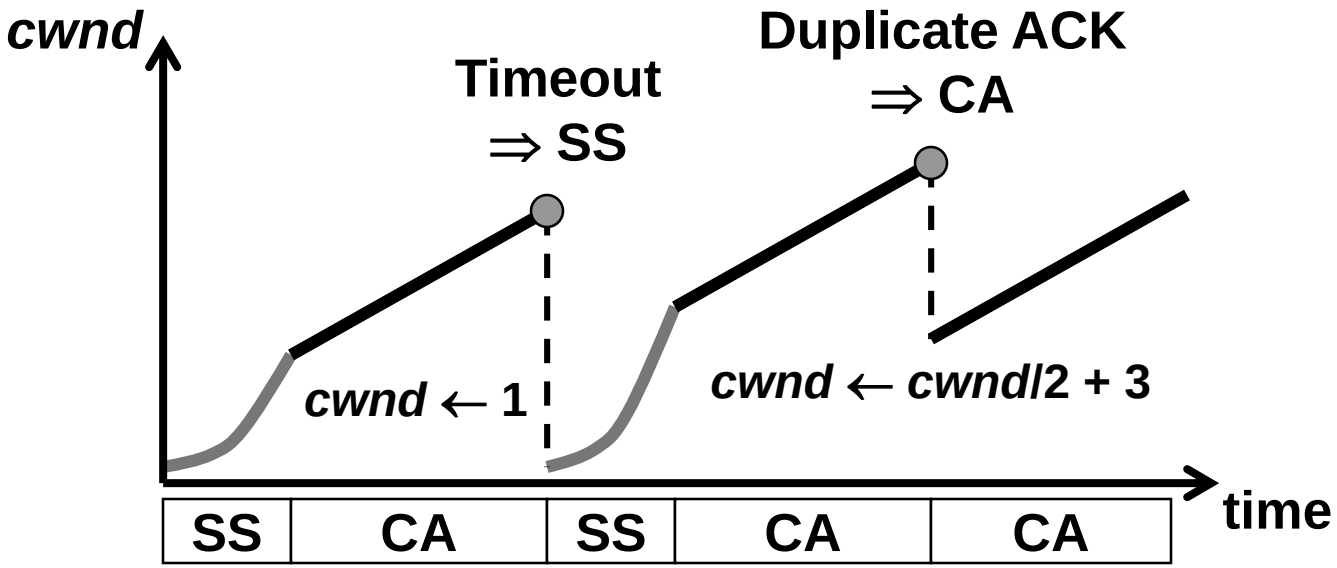


TCP Vegas



- 拥塞更为平滑… 假设缓存足够大

TCP Reno



SS: Slow start
CA: Collision avoidance

提纲

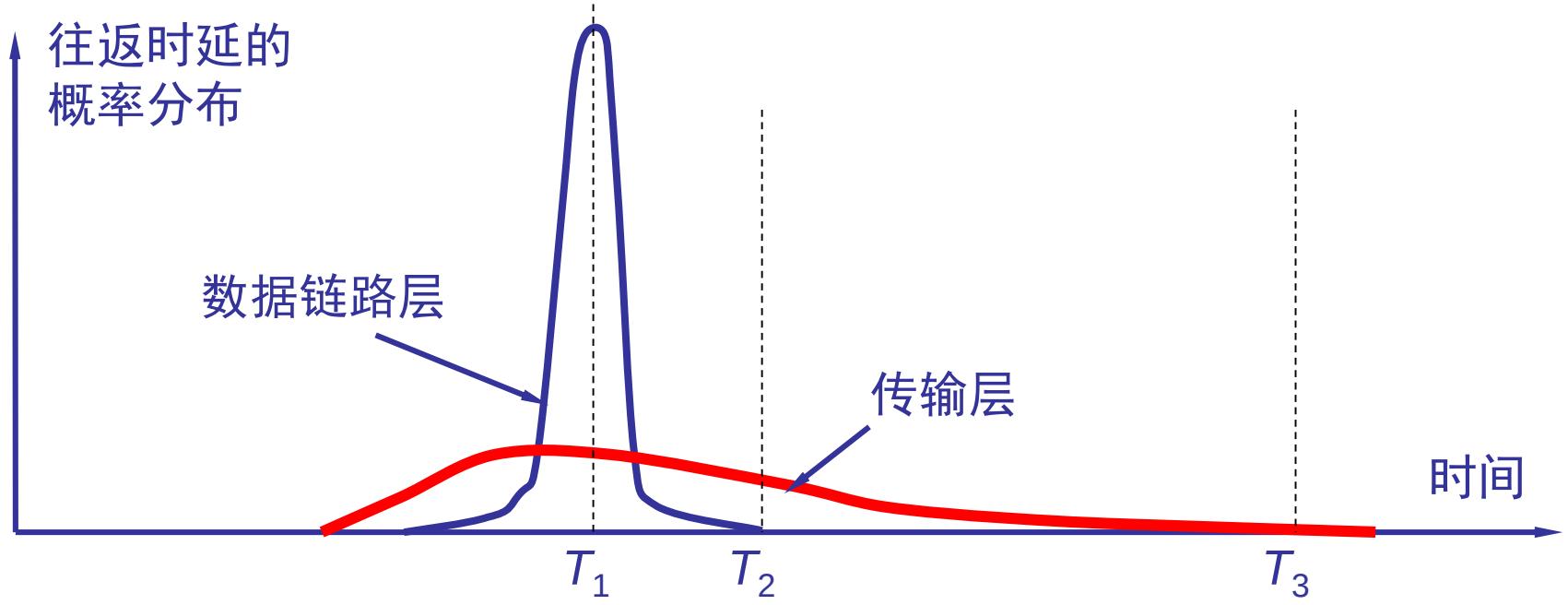
- 拥塞控制
- TCP的拥塞控制机制：慢启动、有效的传输控制窗口
- TCP定时器管理
- 无线链路上的TCP及性能测试

TCP定时器管理

- 发送一个报文段，就设置一次重传定时器
- 在重传定时到之前收到确认，则取消定时；
- 否则，就重传报文段；
- 此外，超时还用于指示网络拥塞；
- 重传定时器如何设置？
  - 设置过小，导致不必要的重传
  - 设置过大，导致对丢失的报文处理不及时

网络往返时延：方差很大

- 在链路上的延迟的均值是可估计的
- 在网络层上，IP 分组的路由经过多条链路，存在排队及拥塞，传输层报文的往返时延的变化很大
- 若设置重传时间为T_2则太小，若为T_3则太大



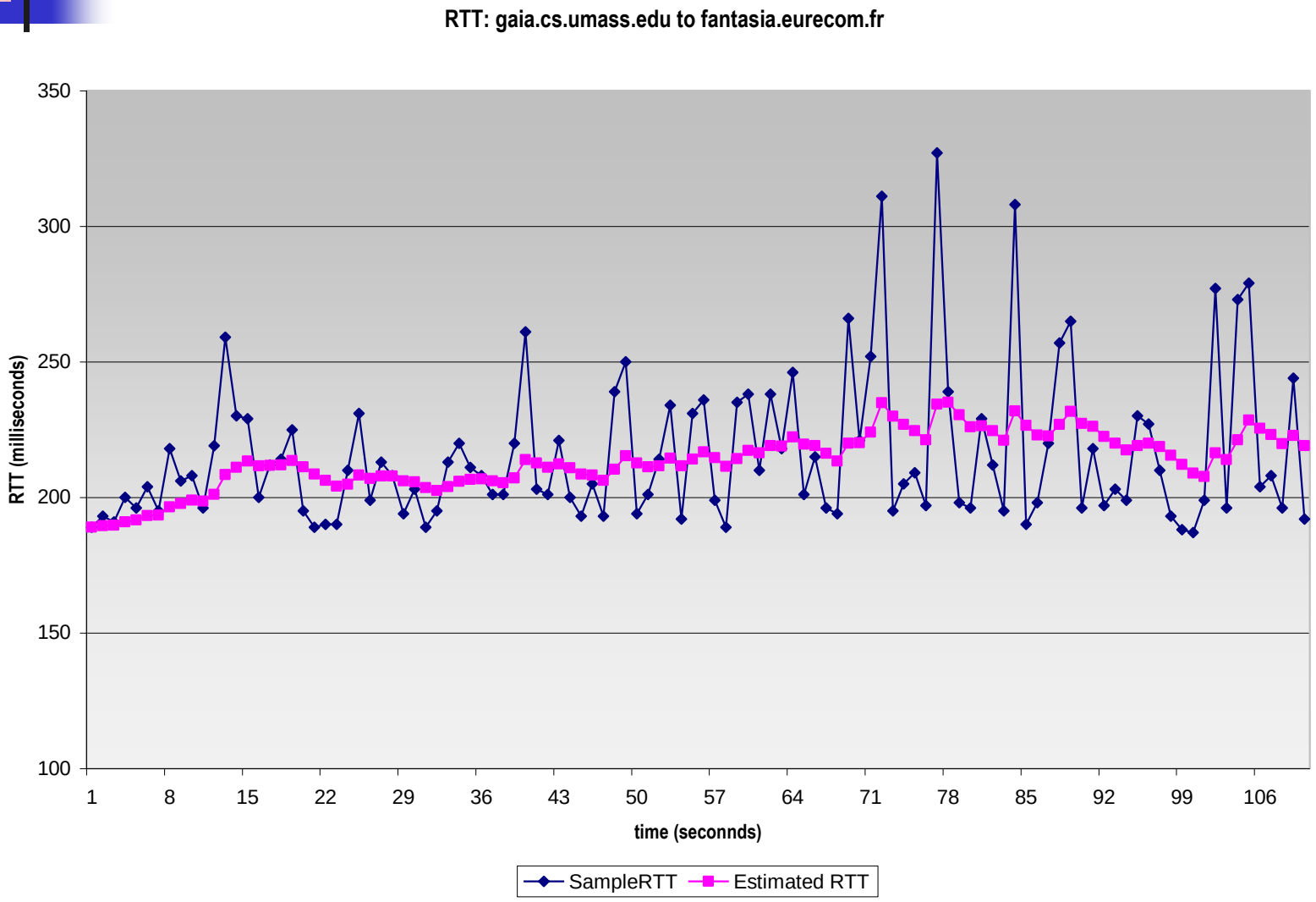
往返时延的估计

- 用发送报文定时器，计算报文往返时延M
- 计算往返时延RTT：采用指数加权移动平均（EWMA）

$$RTT = \alpha \times RTT + (1 - \alpha) \times M \qquad 0 \le \alpha < 1$$

- 参数 α ：
  - 若α接近1，则新的往返时延对RTT影响不大
  - 若α接近0，则新的往返时延对RTT影响较大
  - 典型的 α=7/8

举例：RTT 估计


RTT: gaia.cs.umass.edu to fantasia.eurecom.fr
RTT (milliseconds)
350
300
250
200
150
100
1
8
15
22
29
36
43
50
57
64
71
78
85
92
99
106
time (seconnds)
SampleRTT
Estimated RTT

超时重传时间 RTO

- 计时器 RTO (RetransmissionTime-Out)

$$RTO = \beta \times RTT \qquad \beta > 1$$

- 若 β 接近1，发送端可及时重传丢失报文段，效率高，但若报文并未丢失而只是时延增加了，过早重传则会加重网络负担；同时进入拥塞控制的慢启动
- TCP 标准推荐 β=2。

超时重传时间 RTO

- 延迟的变化为 RTT-M
- 计算平均延迟的变化D

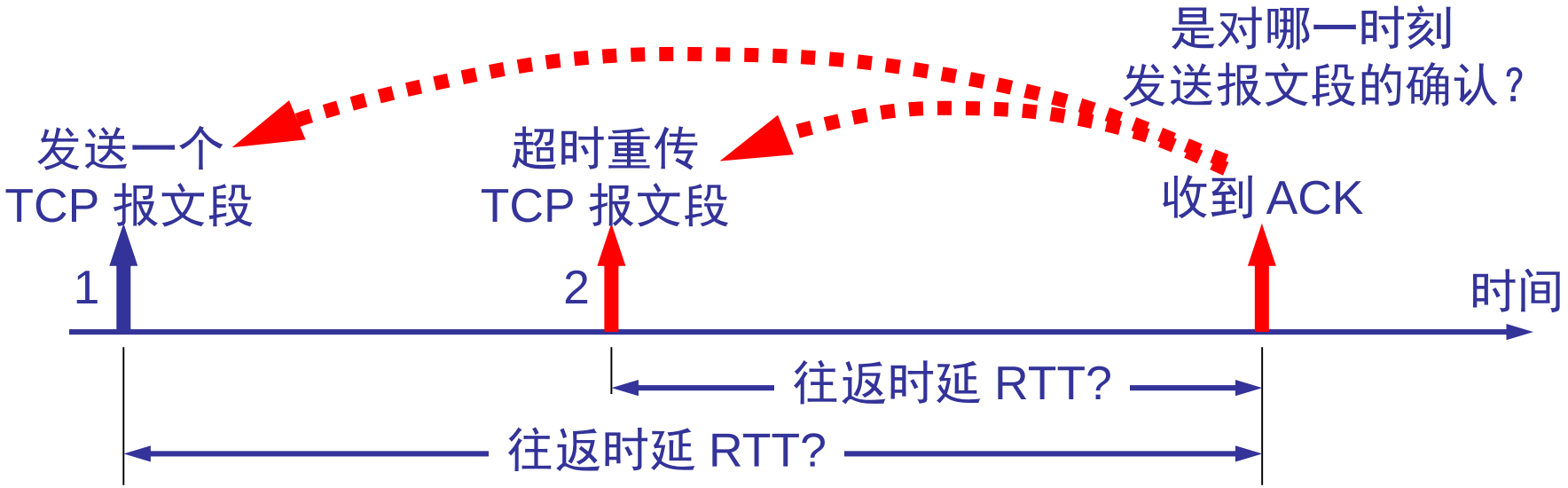
$$D = \alpha \times D + (1-\alpha) \times |RTT-M|$$

  - 其中 α =3/4
- 超时定时器

$$RTO = RTT+4 \times D$$

测量往返时间的问题

- 在时刻1发送报文段，未收到ACK，在时刻2重传报文段，之后收到ACK
- 如何判定ACK是对时刻1的原报文段的确认，还是对时刻2的确认？



Karn 算法及其修正

- Karn 算法：计算平均RTT时，若报文段重传，就不采用新的时延值
- 修正的Karn算法：报文段每重传一次，就增大重传时间RTO：

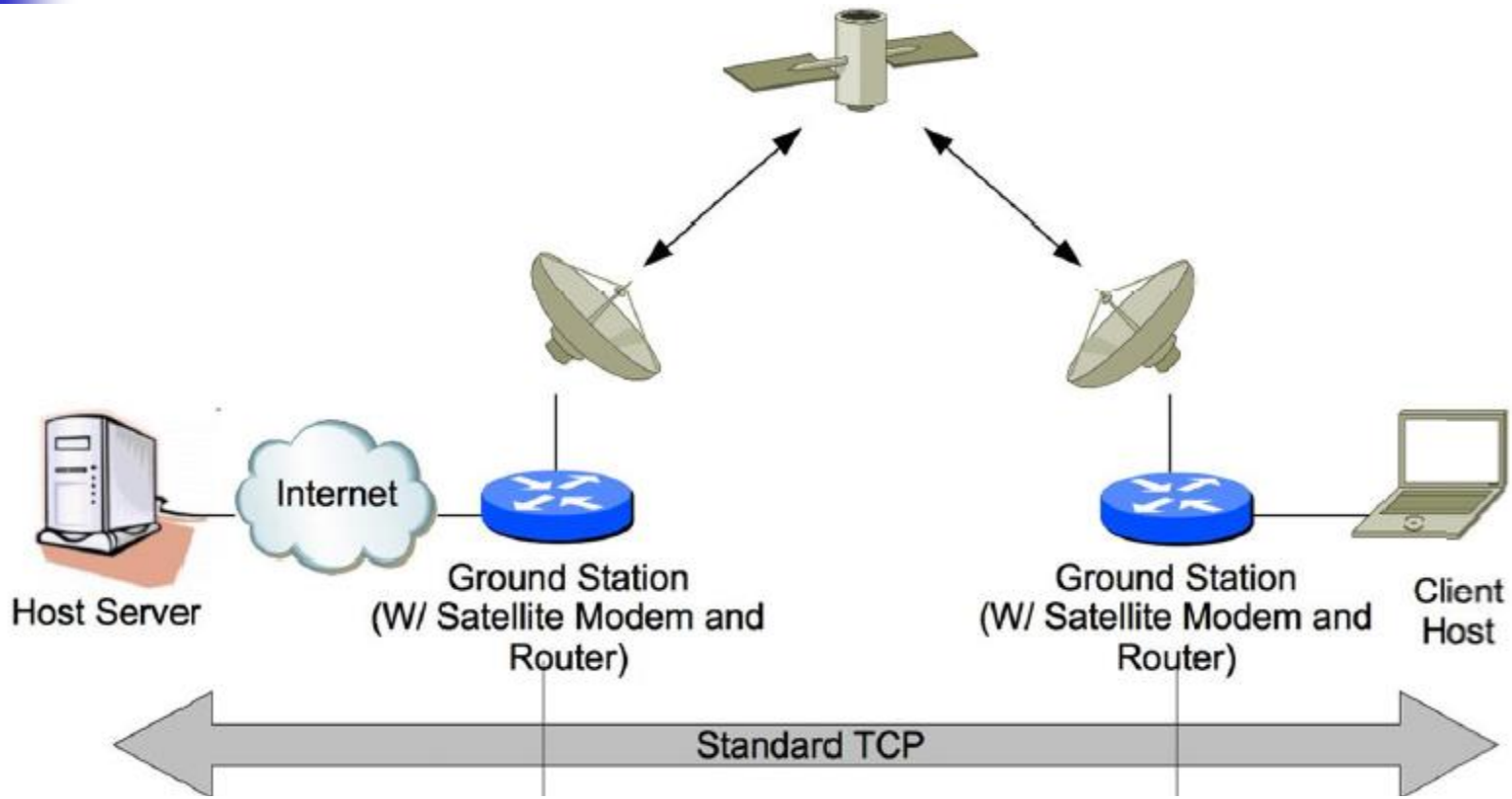
$$RTO = \gamma \times (RTO)$$

- γ =2；当不再发生报文段重传时，才根据报文段往返时延M更新RTT和重传时间RTO。
- 实践证明，这种策略较为合理。

提纲

- 拥塞控制
- TCP的拥塞控制机制：慢启动、有效的传输控制窗口
- TCP超时定时器
- 无线链路上的TCP及性能测试
  - 无线链路的特点：
    - RTT更大：慢启动
    - 误码率高：容易发生超时重传
    - 移动性及切换：存在潜在的丢帧问题

典型的卫星接入网拓扑



问题：

1）在卫星链路上采用哪种TCP拥塞控制方法，性能更好？

2）如何在实验室环境下测验及验证？

无线TCP拥塞控制算法及其实验

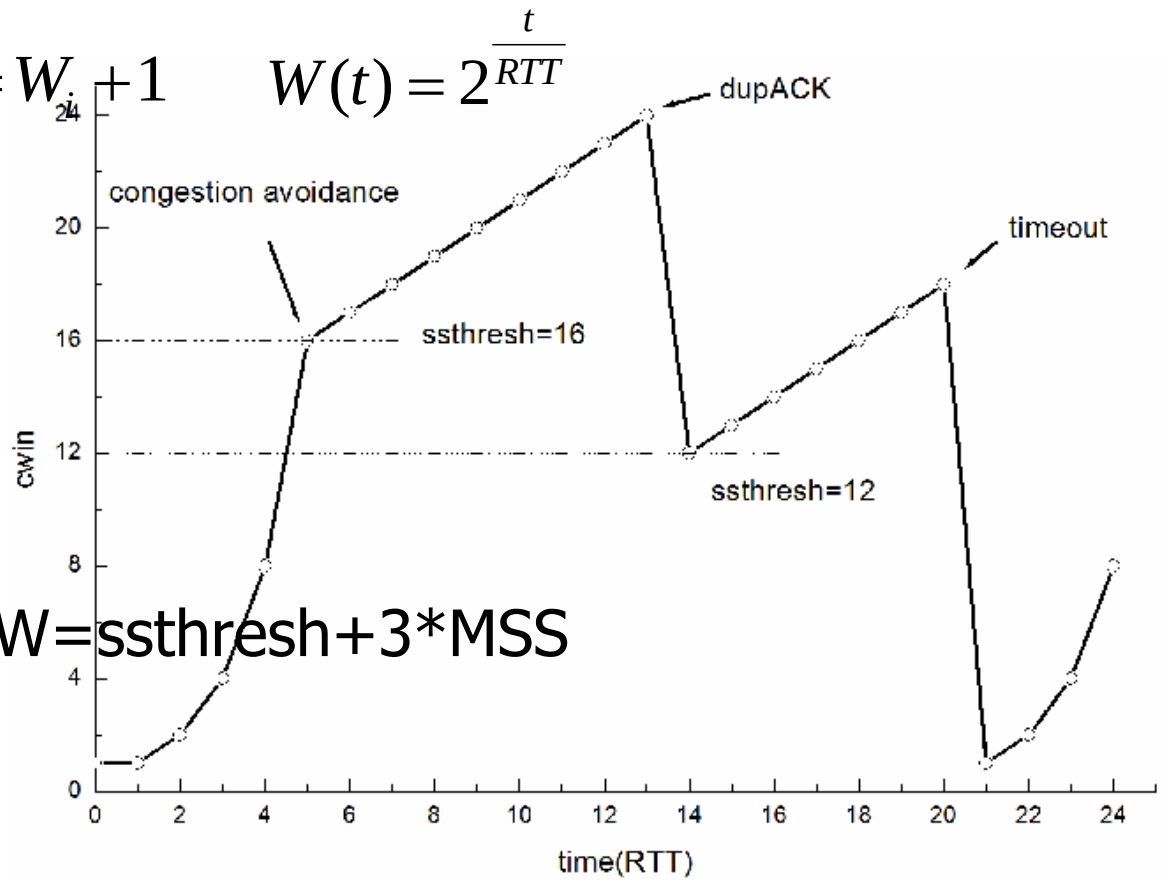
- W为拥塞窗口，RTT为往返时间，γ为拥塞后的窗口，t_γ为拥塞发生时刻
- Reno算法
  - 慢启动 $W_{i+1} = W_i + 1$ $W(t) = 2^{\frac{t}{RTT}}$
  - 拥塞避免

$$W_{i+1} = W_i + 1/W_i$$

$$W(t) = \frac{t - t_\gamma}{RTT} + \gamma$$

  - 其他

ssthresh=W/2，W=ssthresh+3*MSS



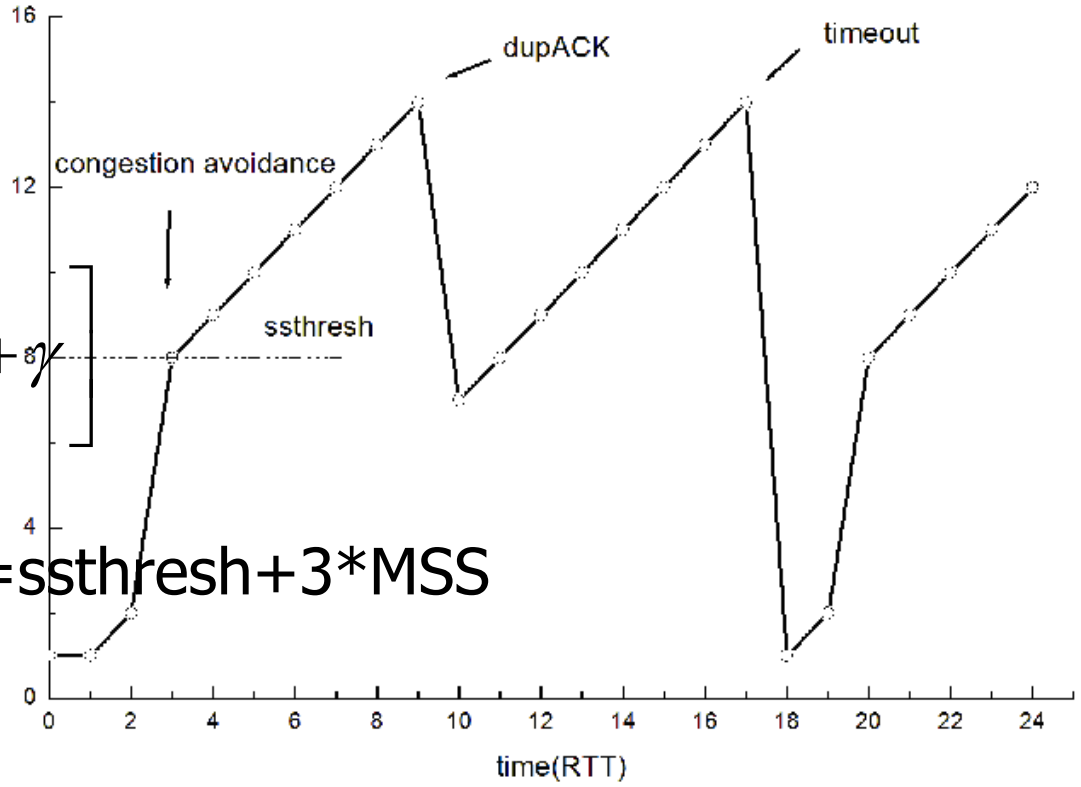
无线TCP拥塞控制算法及其实验

- W为拥塞窗口，RTT为往返时间，γ为拥塞后的窗口，t_γ为拥塞发生时刻
- Hybla算法：相对RTT参数 $\rho = RTT / RTT_0$
  - 慢启动 $W_{i+1}^{'} = W_i^{'} + 2^{\rho} - 1$ $W^{'}(t) = \rho 2^{\rho \frac{t}{RTT}}$
  - 拥塞避免

$$W_{i+1}^{'} = W_i^{'} + \rho^2 / W_i^{'}$$

$$W^{'}(t) = \rho\left[\rho \frac{t - t_{\gamma,0}}{RTT} + \gamma\right]$$

  - 其他 ssthresh=W/2，W=ssthresh+3*MSS



无线TCP拥塞控制算法及其实验

- W为拥塞窗口，RTT为往返时间，γ为拥塞后的窗口，t_γ为拥塞发生时刻
- Bic/Cubic算法
  - 慢启动：$K = \sqrt[3]{W_{\max}\beta / C}$

    Wmax是减小之前的窗口，β是一个乘性因子
  - 拥塞避免

$$W_{cubic} = C(t-K)^3 + W_{\max}$$

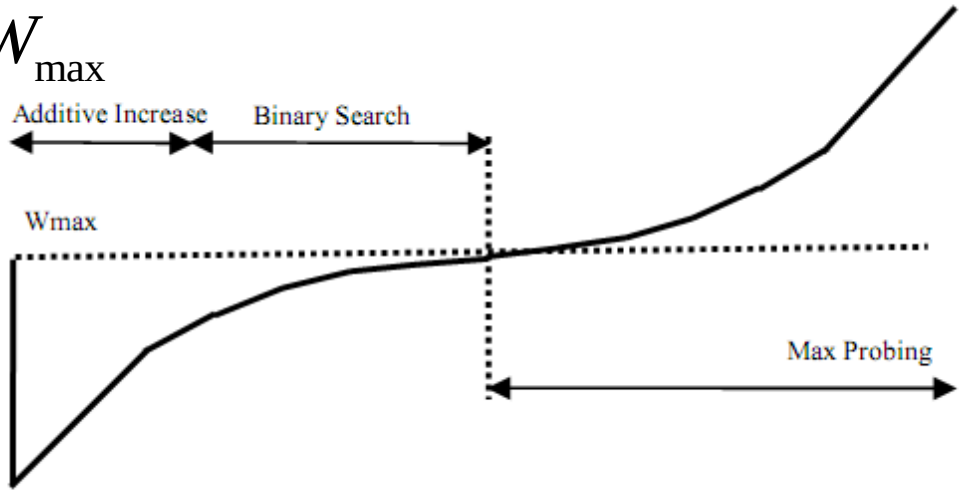


Fig. 1: The Window Growth Function of BIC

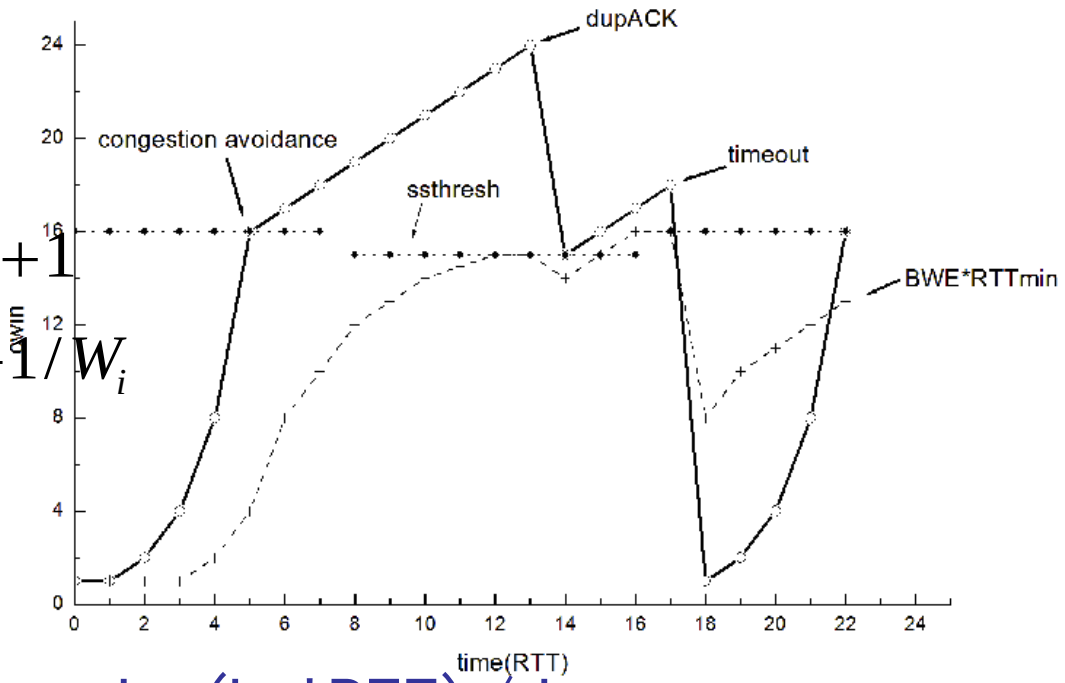
无线TCP拥塞控制算法及其实验

- W为拥塞窗口，RTT为往返时间，t_k接收ACK的时刻，d_k为分组长度
- Westwood+算法
  - 慢启动
  - 拥塞避免
  - 估计带宽：

$$b_k = \frac{d_k}{t_k - t_{k-1}}$$

$$W_{i+1} = W_i + 1$$

$$W_{i+1} = W_i + 1/W_i$$



  - 3次重复的ACK sthresh=(b_k*RTT)/d_k
    cwnd=ssthresh
  - 超时：ssthresh =(b_k*RTT)/d_k c*wnd* =1

模拟卫星链路

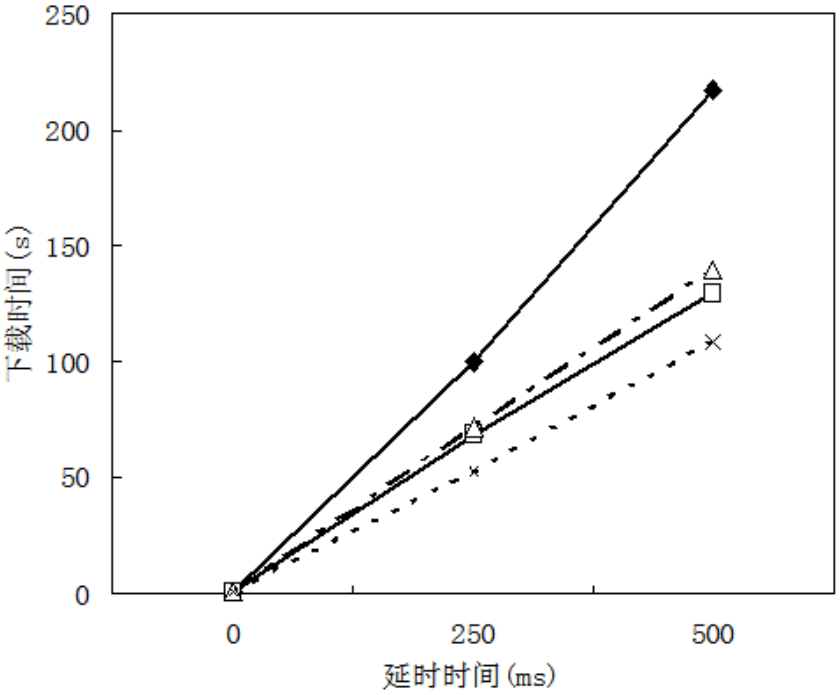
- 卫星链路的特点：
  - 单向传播延时为250ms-280ms
  - 误码率$P_b=10^{-3}\sim10^{-7}$，分组长L=100B，丢包率 $PLR=1-(1-P_b)^L$
- 模拟卫星链路：两台Linux系统，利用traffic control（TC）来控制数据包发送速率、时延、优先级等。
  - tc qdisc add dev eth0 root netem delay 500ms loss 1%
- 设置Linux内核TCP拥塞算法：
  - 在文件/etc/sysctl.conf中添加 sys.net.ipv4.tcp_congestion_control = Hybla
  - 运行命令 sysctl –p

实验结果

- TCP拥塞控制算法：reno，Hybla，Westwood+，Bic/Cubic
- 利用TC工具设计6种链路场景
- 下载6MB文件，计算文件下载时间

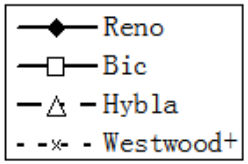
TCP拥塞算法	1%丢包无延时	1%丢包250ms 延时	1%丢包500ms 延时	5%丢包无延时	5%丢包250ms 延时	5%丢包500ms 延时
Reno (s)	0.47	99.6	217	6.16	252	632
Bic(s)	0.47	68.6	129	4.25	217	458
Hybla(s)	0.47	71.7	140	3.85	54.2	218
Westwood+(s)	0.47	53	109	2.69	224	298

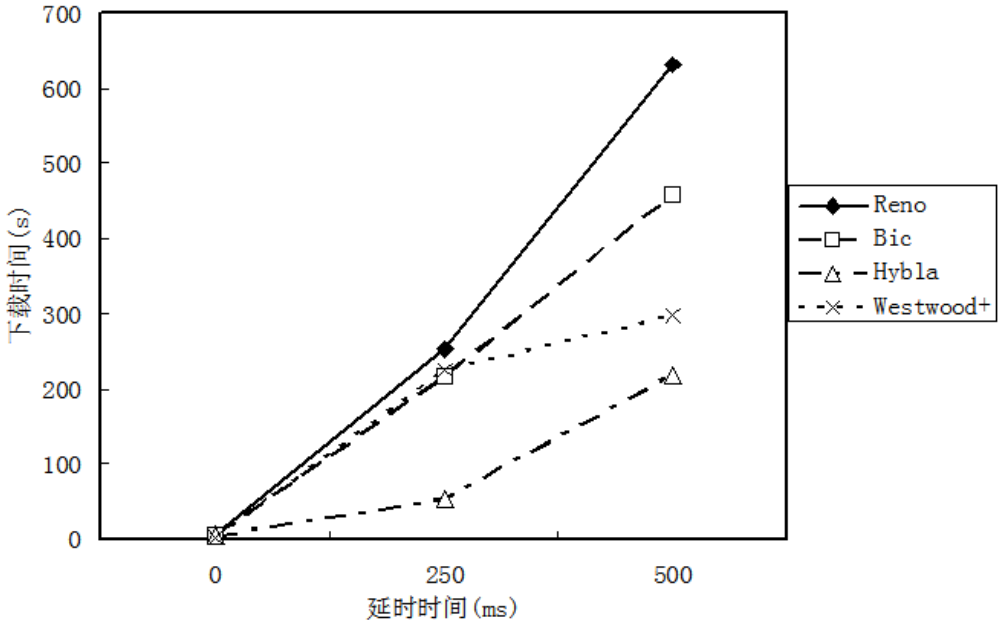
实验结果



1%丢包率：时延——下载时间

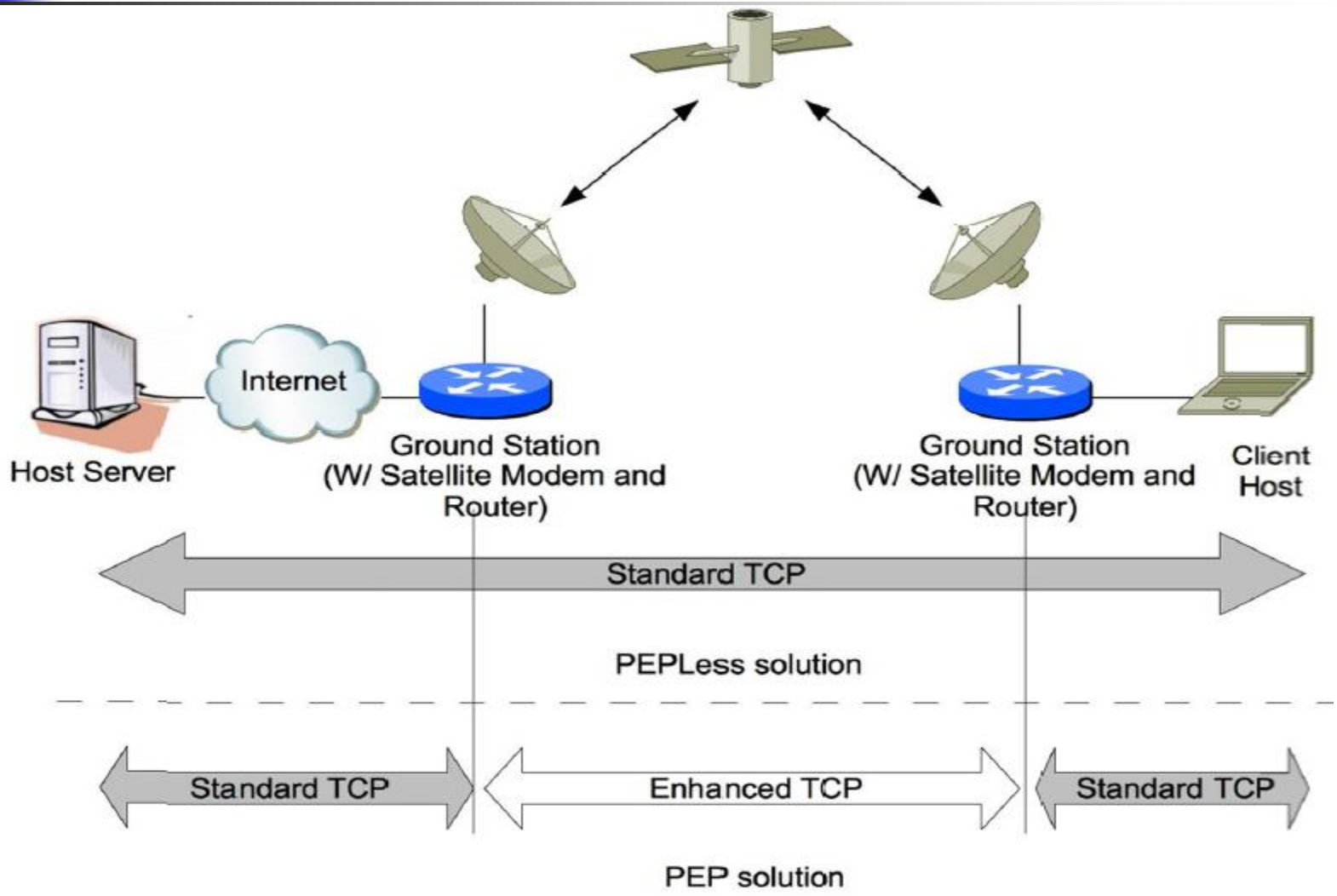
结论：在卫星链路上，采用Westwood+及Hybla改进TCP性能更好





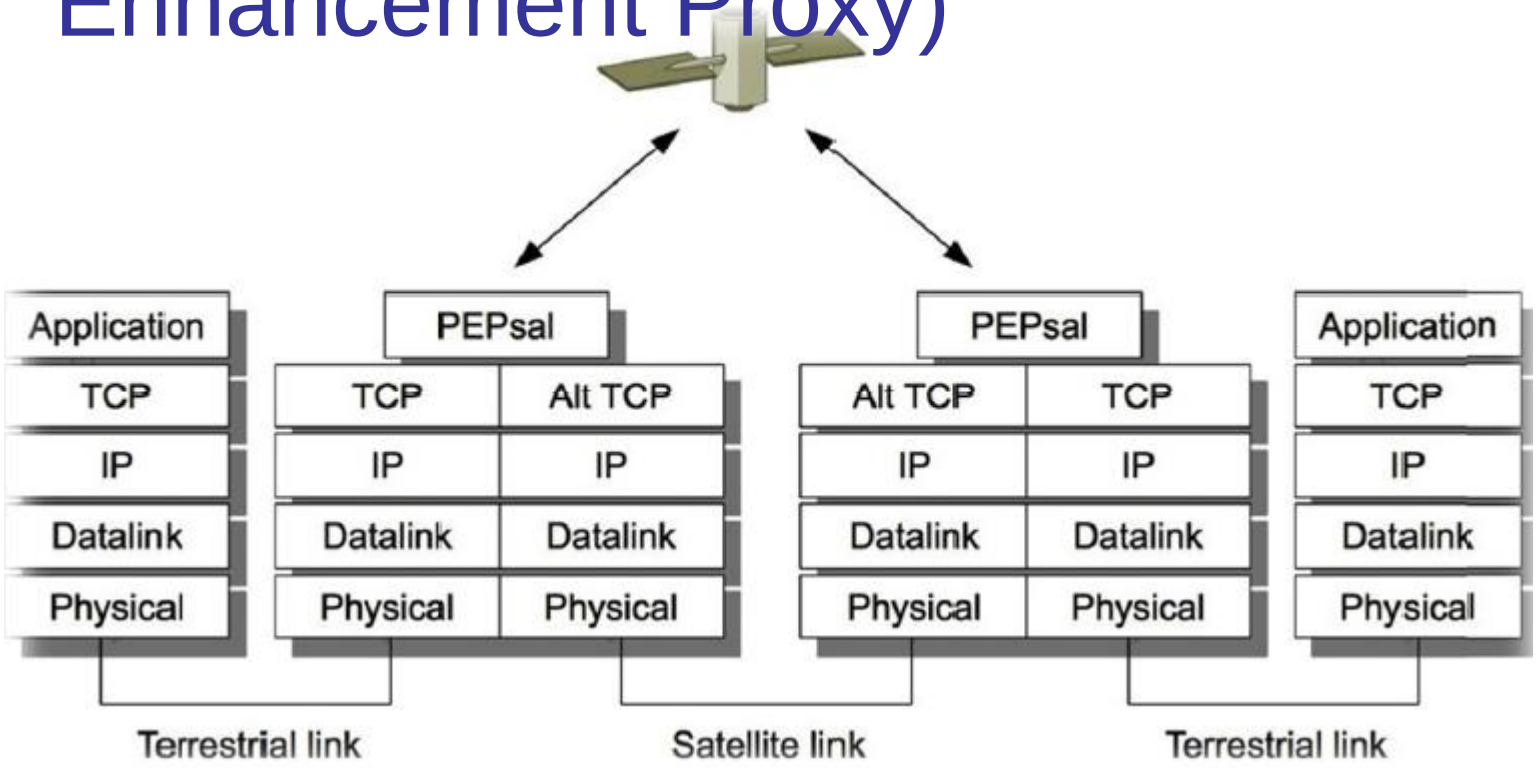
5%丢包率: 时延——下载时间

典型的卫星接入网拓扑



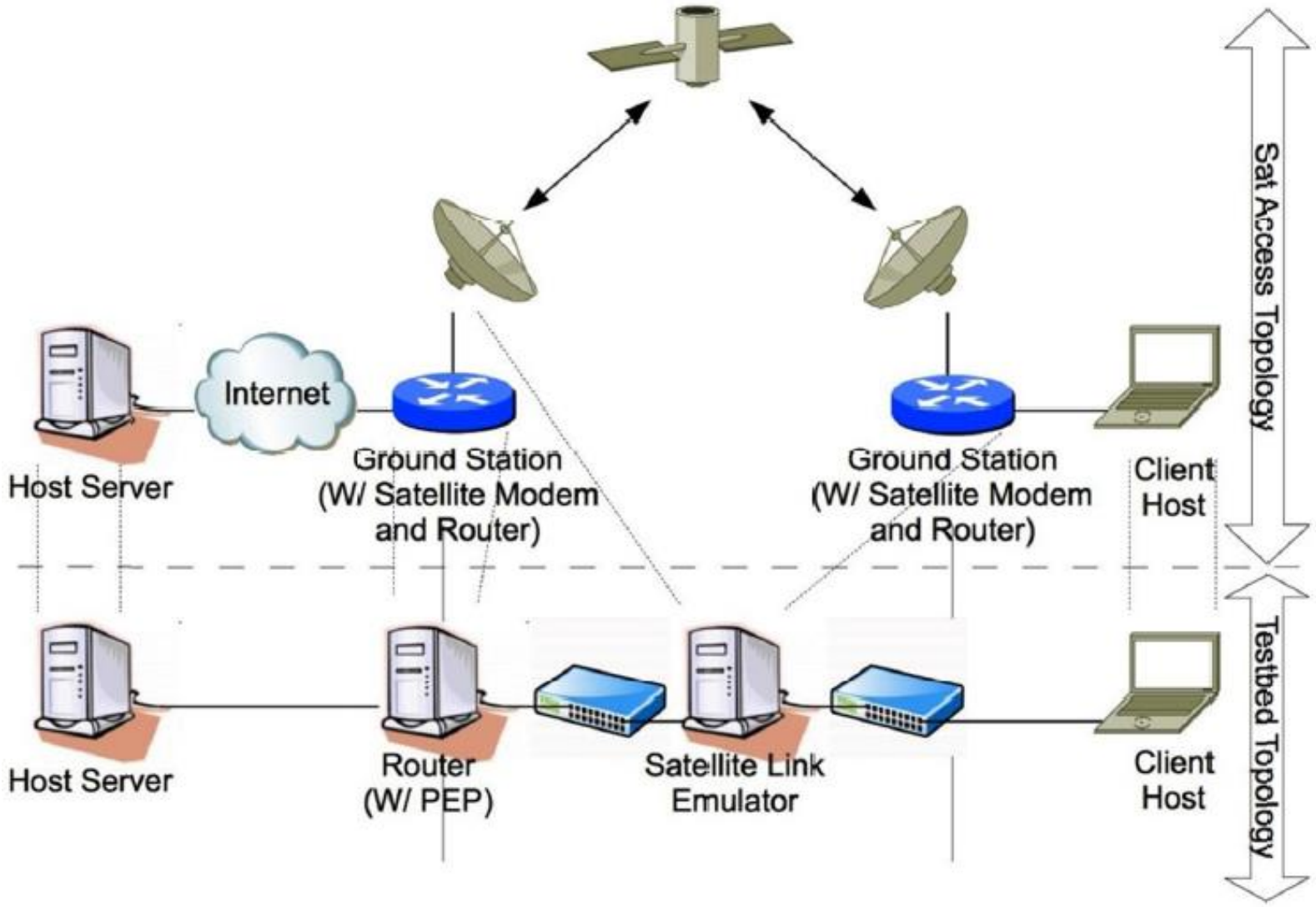
问题：是否有增强的TCP实现，开源且可移植？

TCP 加速或TCP PEP (Performance Enhancement Proxy)

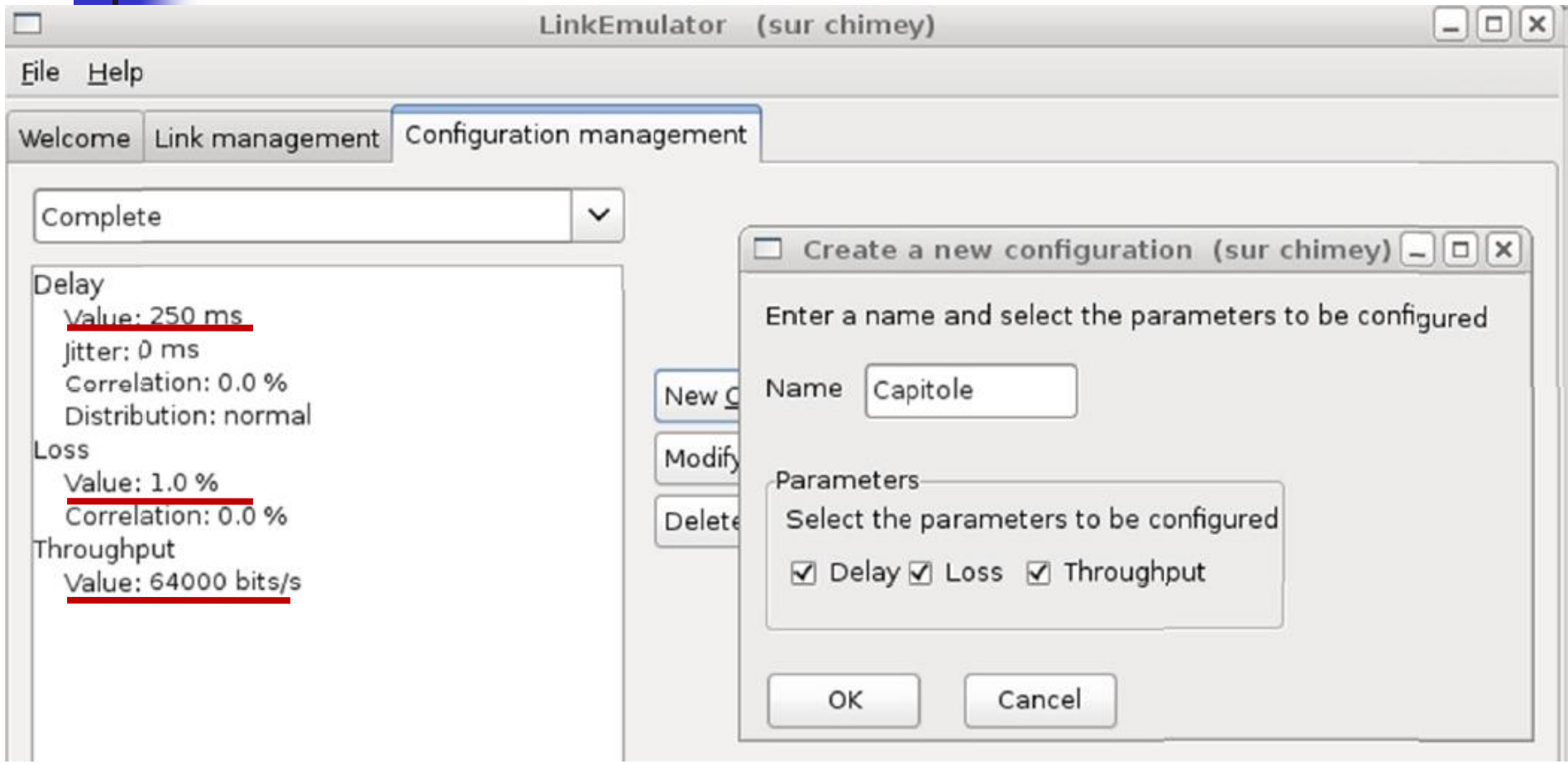


"A New Survey on Improving TCP Performances over Geostationary Satellite Link", Network and Communication Technologies; Vol. 2, No. 1; 2013,ISSN 1927-064X E-ISSN 1927-0658,Published by Canadian Center of Science and Education

试验方法及工具


Sat Access Topology
Internet
Host Server
Ground Station
(W/ Satellite Modem
and Router)
Ground Station
(W/ Satellite Modem
and Router)
Client
Host
Testbed Topology
Host Server
Router
(W/ PEP)
Satellite Link
Emulator
Client
Host

卫星链路模拟软件


LinkEmulator (sur chimey)
File Help
Welcome
Link management
Configuration management
Complete
Delay
Value: 250 ms
Jitter: 0 ms
Correlation: 0.0 %
Distribution: normal
Loss
Value: 1.0 %
Correlation: 0.0 %
Throughput
Value: 64000 bits/s
New C
Modify
Delete
Create a new configuration (sur chimey)
Enter a name and select the parameters to be configured
Name
Capitole
Parameters
Select the parameters to be configured
Delay
Loss
Throughput
OK
Cancel

卫星链路TCP性能测试

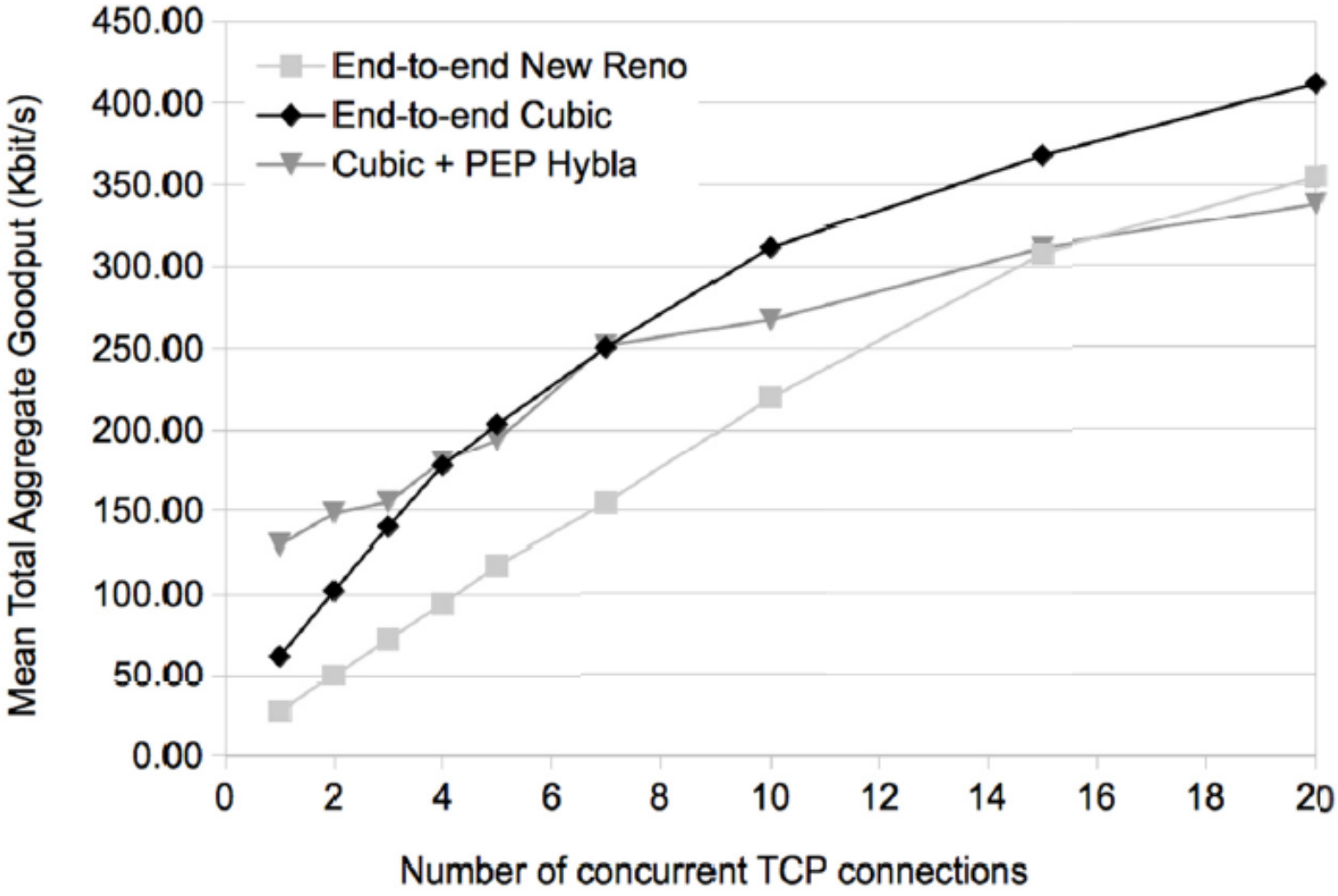
- 三种协议：*e2e_Nreno,(*end to end TCP NewReno)、*e2e_Cubic*、*Cubic & PEPSal*
- 表1：网页下载

Testbed 配置	网页下载时间(s)	吞吐量 (KB/s)
e2e_NewReno	158	31.4
e2e_Cubic	72	68.8
Cubic & PEPsal	37	133.9

- 表2：文件下载

Testbed 配置	传输时间(s)	吞吐量(KB/s)
e2e_NewReno	77.5	26.9
e2e_Cubic	34.8	61.9
Cubic & PEPsal	18.8	121.3

平均聚合吞吐量


End-to-end New Reno
End-to-end Cubic
Cubic + PEP Hybla
Mean Total Aggregate Goodput (Kbit/s)
450.00
400.00
350.00
300.00
250.00
200.00
150.00
100.00
50.00
0.00
0
2
4
6
8
10
12
14
16
18
20
Number of concurrent TCP connections

小结

- TCP建立连接，协商参数
  - 接收窗口WIN，最大报文段MSS，SACK
- TCP的定时器管理
  - 计算RTT并更新
  - RTO = β × RTT
- TCP的拥塞控制
  - 发送窗口swnd=Min[rwnd,cwnd]，rwnd接收窗口
  - 拥塞窗口cwnd：慢启动、线性增加、乘性减少
- 无线链路上的TCP及性能测试

作业：TCP协议实验

- 1) 设计保存TCP 连接信息的数据结构（TCB）
- 2) 实现stud_tcp_input()函数进行TCP的接收：完成校验和检查、字节序转换，实现TCP客户机报文接收的有限状态机。
- 3) 实现stud_tcp_output()函数进行TCP的发送。
- 4)实现TCP客户机的5个Socket接口函数，stud_tcp_socket()、stud_tcp_connect()、stud_tcp_recv()、stud_tcp_send()和stud_tcp_close()，并与TCP发送和接收流程有机地结合起来

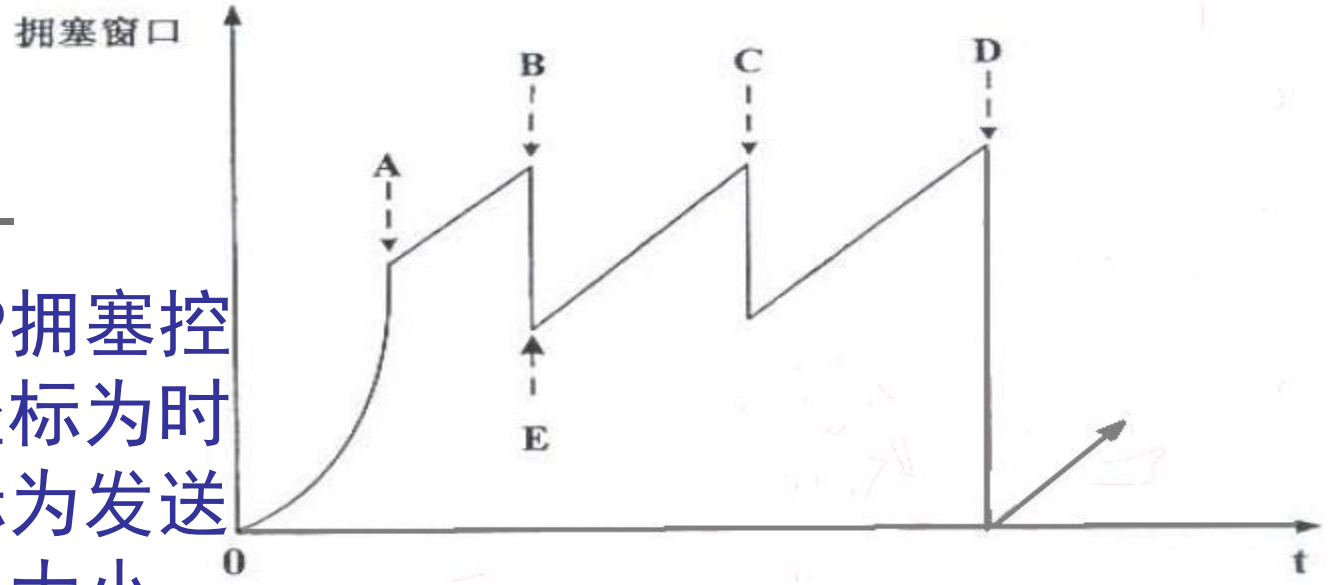
提交截止时间：5月30日

练习题

- 计算下列网络的时延带宽积（1）T1（1.5Mbps）（2）以太网（10Mbps）（3）T3（45Mbps）（4）STS-3（155Mbps）假设RTT为100ms，TCP头部中窗口大小为16位。根据你的计算其含义是什么？
- 如果TCP的RTT为30ms，确认帧分别在26ms, 32ms, 24ms到达，若采用指数加权移动平均算法，问新的RTT估计值是多少？ α=0. 9

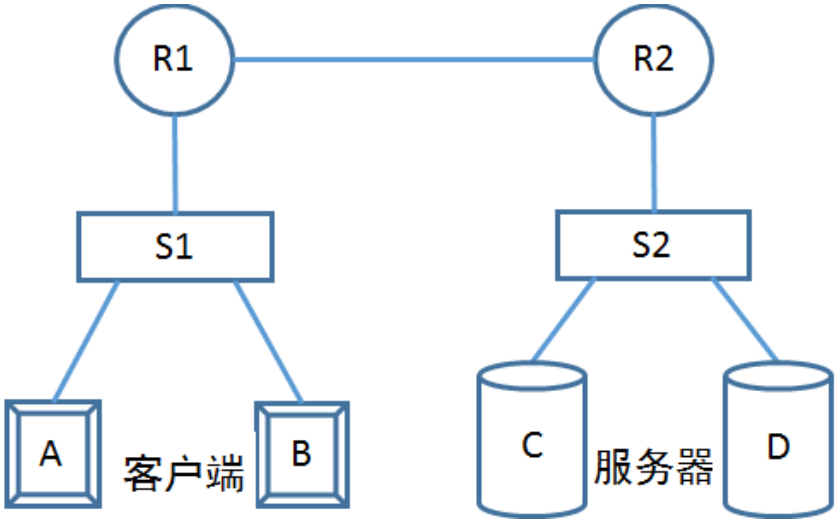
练习题

- 右图给出TCP拥塞控制过程。横坐标为时间轴，纵坐标为发送方的拥塞窗口大小。



1. 假设最大段长MSS为1000字节。当拥塞窗口大小达到A点时发送方共向网络中传输了15000字节。试计算A点对应的拥塞窗口值（假设发方在t=0时刻建立TCP连接，数据发送时延忽略不计）
2. 根据TCP的拥塞控制机制说明A、B和E点拥塞窗口变化的原因。
3. C、D处分别是因何原因导致拥塞窗口减小？

练习题



- 在如图所示的网络中A和B是客户机，C是流媒体服务器，D是文件服务器。A正在点播C上的某个流媒体节目，此时主机B拟与服务器D建立TCP连接，下载D上的文件。假设客户机和服务器的往返时间RTT=100ms，待传文件长为500KB，TCP的MSS为5KB，B的接收缓存为100KB，拥塞窗口初始化为32KB 。（TCP采用传统的拥塞控制策略，1M=1000K）。假设B的处理速度足够快其接收缓存不会堆积报文 。试问经过多少轮的发送，D才能将整个文件发送完毕？请给出D每轮发送的有效窗口大小。

练习题

- 设TCP拥塞控制算法中，拥塞窗口cwnd的初始值为1(报文段)，慢开始值ssthresh的初始值为8(报文段)。当拥塞窗口cwnd上升到14(报文段)，网络发生超时，TCP启动拥塞避免过程。试分别计算TCP建立连接后第1轮次到第15轮次的拥塞窗口cwnd大小(报文段)，并写出计算过程。

轮次	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cwnd/ MSS															