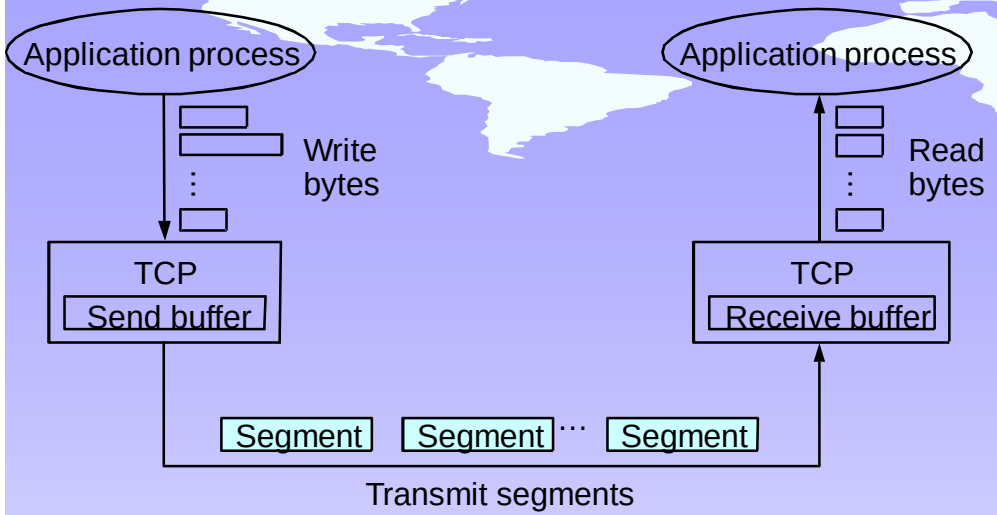
2.3 TCP拥塞控制(闭环，源抑制)

◆源算法

- ➢最广泛使用的TCP协议中的拥塞控制算法.
- ➢根据MCI的统计,总字节数95%和总报文数的90%使用TCP传输.
- ➢下表给出拥塞控制源算法的简单描述

拥塞控制源算法	描述
Tahoe-TCP	慢启动、拥塞避免、快速重传三算法.（早期较为普遍采用的版本）paper： congest avoid.pdf
Reno-TCP	RFC5681加上快速恢复.（当前最为广泛采用的TCP实现版本）
NewReno-TCP	引入了部分确认和全部确认的概念.
SACK-TCP	规范了TCP中带选择的确认消息.
Vegas-TCP	采用带宽估计,缩短了慢启动阶段的时间，源拥塞避免 /net/ipv4/tcp_vegas.h， /net/ipv4/tcp_vegas.c



Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

10Gbps→3.4s

100Gbps →0.34s

源端口Source Port（16 bit）								宿端口Destination Port（16 bit）
序列号Sequence Number（32 bit）								
确认号Acknowledgment Number（32 bit）								
数据偏移（Data Offset（4bit）	保留（为0）Reserved（6 bit）	URG	ACK	PSH	RST	SYN	FIN	Advertised Window（16 bit）
校验和Checksum（16 bit）								紧急指针Urgent Pointer（16 bit）
可选项Option（32 bit）								
数据Data（32 bit）								

TCP序号回绕时长

- 4字节=32比特
- 回绕时段长

$=2^{32} \times 2^3/100 \times 10^6/s=$

$317s \approx 6m$

TCP协议数据格式：

- TCP基本策略：
  - 智端+傻网，端端拥塞控制，
  - 无预约,先渐增发，后观察事件反应
- TCP假设网络中R仅用FIFO排队；但也用FQ规则
- 拥塞包在R中丢弃后，由重传解决
- TCP又叫自定时（self clocking）
- 以下描述3个TCP拥塞控制机制

2.3.1 加法式增加/乘法式减少

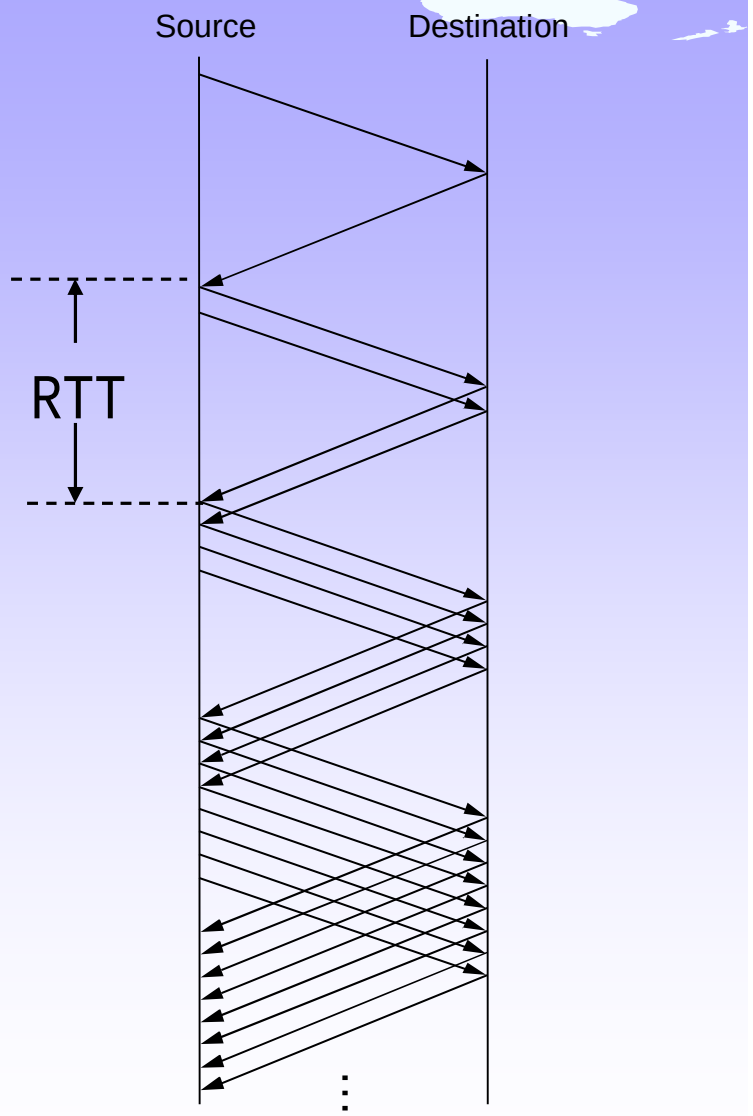
- CongestionWindow
  - **源端TCP为每个连接维护的一个状态变量；**
  - **限制在给定时间内容许传输的数据量**
- **容许的最大未确认字节数就是现在的最小** CongestionWindow**和**AdvertisedWindow
  - MaxWindow = MIN(CongestionWindow, AdvertisedWindow)
  - **可以发送的窗口大小：**EffectiveWindow = MaxWindow - (LastByte - LastByteAcked)

2.3.2 慢启动

- **线性加**
  - **对接近网络可用容量的源来说是一个正确的方法；**
  - **但对从零开始起步的连接来说时间又太长；**
- TCP第2个机制—慢启动（实际相当快）
  - **指数式增加CW，而不是线性。**
  - **开始，源把CW置为1（一个包）；**
  - **然后每收到一个ACK，将CW*2**
  - **当该包的ACK到达后，TCP可以发2个包；**
  - **当收到2个ACK后，CW*2，发4个包，故每个RTT内包数翻一倍**
- **当超过一个阈值：**ssthresh（slow start threshold）cw又变为线性增加，拥塞避免

慢启动传输中的包

- **慢启动称为慢的原因是，该方法未出现前，源按**Advertised Window**可立刻发送所有数据**



TCP怎样学到CW

- AW
  - 靠连接的**接收方来发送**，没有谁发送CW到发送方
- TCP源根据它从现在网络觉察到的拥塞程度来**设置**CW
  - 当拥塞程度**上升**时，就**减少**CW
  - 当拥塞程度**下降**时，就**增加**CW
  - 上述二者合在一起就叫：**加法式增加**/**乘法式减少**

怎样感知是否拥塞？

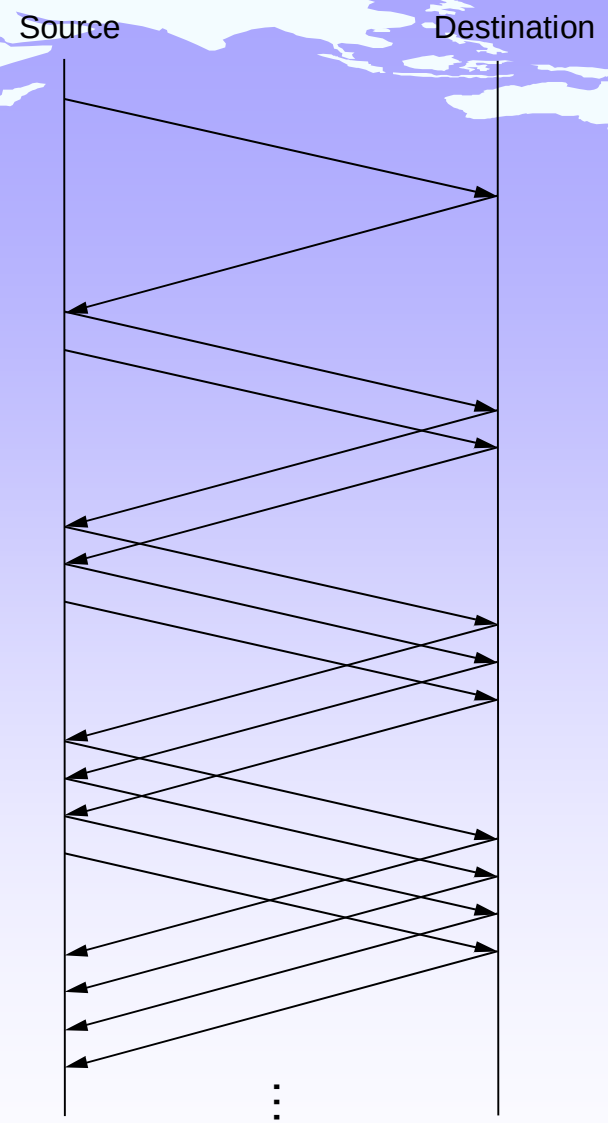
- **观察主要原因：**
  - **包未交付，并导致**duplicated ACK**或超时，判定为拥塞丢包！**
  - **由于传输错误丢包是很少的！**
- TCP**把收到**duplicated ACK**，超时两种现象解释为拥塞的信号**
  - Tahoe TCP**不区分这两种情况**
    - sshthresh = cwnd /2
    - cwnd **重置为** 1
    - **进入慢启动过程**
    - **这就是乘法减少机制的一部分**
  - Reno TCP**面对**duplicated ACK**的实现是：**
    - **进入拥塞避免**
    - cwnd = cwnd /2
    - sshthresh = cwnd
    - **进入快速恢复算法**——Fast Recovery

Duplicated ACK拥塞发生：折半减少CW

- CW定义为字节单位
  - 但也容易以包为单位考虑；例如，
  - CW现在是16个包，若检测到丢包，则设置CW ← 8 = 16/2；
  - 再丢一次包将为8/2 = 4，4/2 = 2；
  - 最后到2/2 = 1；TCP不允许比1个包更小的CW，或最大段尺寸（MSS）

Duplicated ACK拥塞发生：加法式增加CW

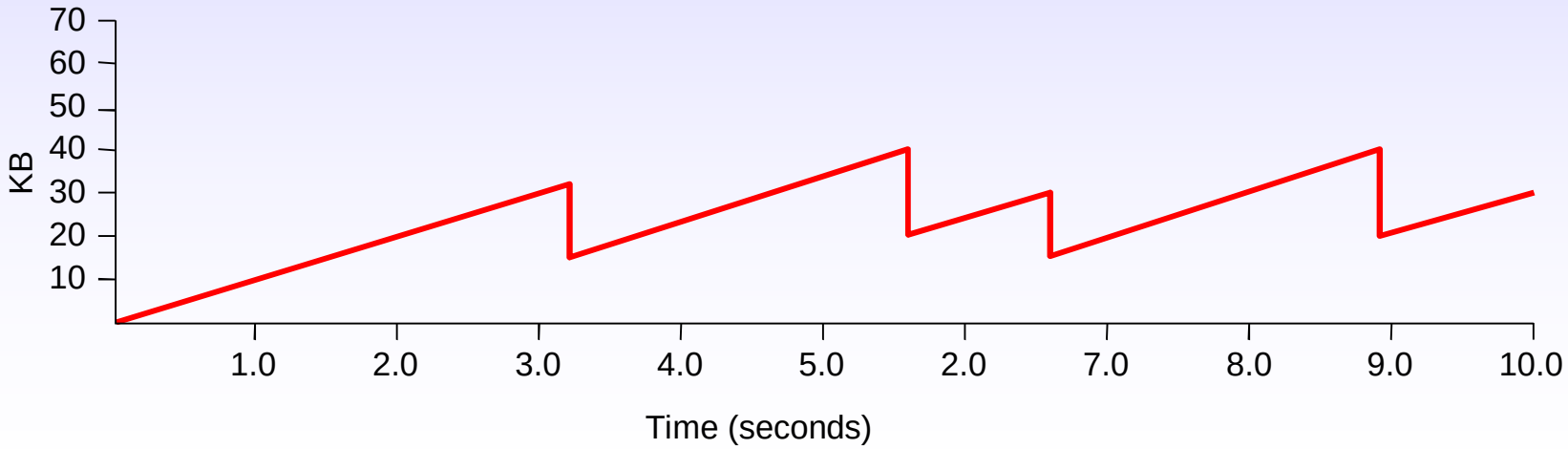
- TCP源
  - 每次成功发送后CW个包后；
  - 即最后一个RTT中每个包都收到了其ACK，
  - 则 CW ← CW+1，这种线性增加



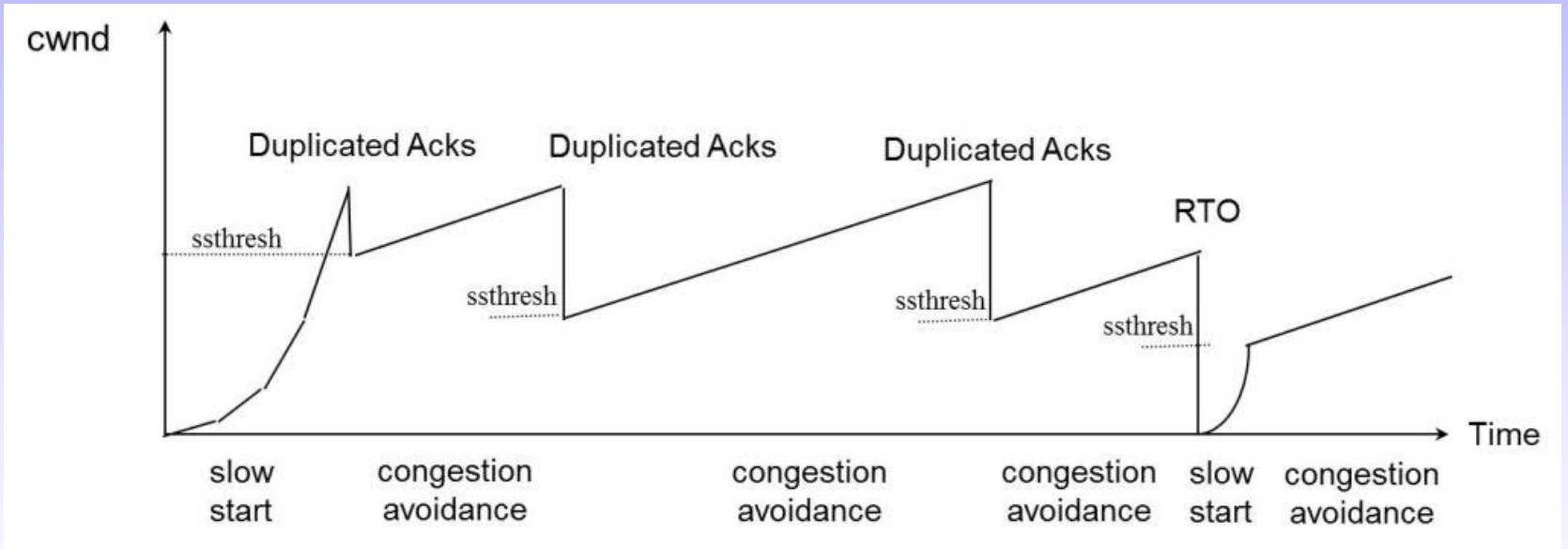
在每个RTT后加1个包

实际是锯齿模式加

- **实际并操作**
  - **不等待全部CW个包的ACK都到后才cw←cw+1，**
  - **而是在每一个ACK到达后给CW增加一点，其增量为**
  - Increment = MSS×(MSS/CW)
  - CW += Increment



◆ TCP跟踪

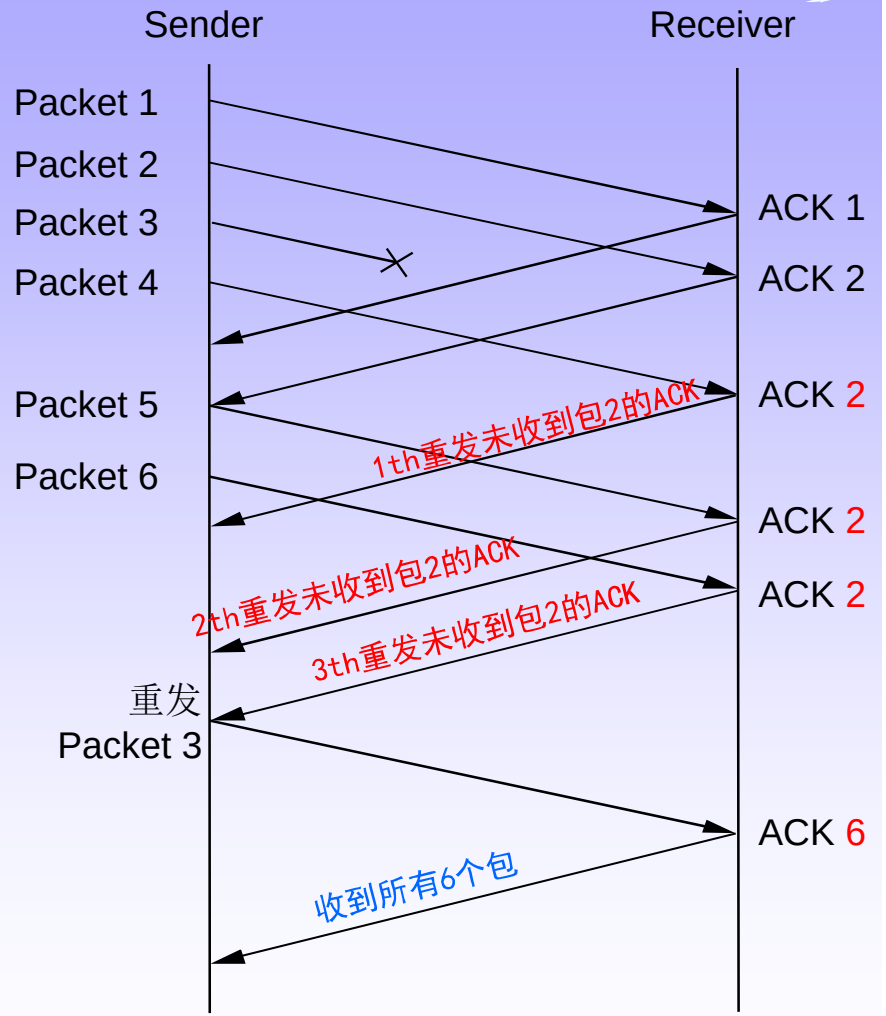

cwnd
Duplicated Acks
Duplicated Acks
Duplicated Acks
RTO
ssthresh
ssthresh
ssthresh
ssthresh
Time
slow start
congestion avoidance
congestion avoidance
congestion avoidance
slow start
congestion avoidance

0.4

2.3.3 快速重发和快速恢复

- 每次接收方收一个数据包
  - 接收方响应一个ACK；（早先）
  - 当包到达失序,TCP因为较早的数据还没有到达,也不能应答含有这些包的数据;
  - TCP发送它上次发送了的那个相同的ACK
  - 相同ACK的第二次发送称着Duplicate ACK,
  - 当发送方看到重复ACK,知道对方收到了一个失序包,这说明比该包更早的包可能被丢失！？
- 实践中
  - TCP等到3个失序包后就重发丢失的包；即快速重发！

- 重复ACK**触发**快速重发
  - 目方收到包1和包2,但包3在网络中丢失
  - 当包4到达时,目方**重发包2的**ACK（1th**重发未收到信息**）
  - 当包5,包6到达时,目方**仍重发包2的**ACK（2th**重发未收到信息**）
  - 当发方收**到包2的第3个**ACK后,它（3th**重发未收到信息**）
  - 当重发包3到达目方后,收方返回一个含有包3、6的**累计**ACK,



改进与快速恢复

当快速重发同时快速恢复

- Fast Recovery**算法如下：**
  - 1.**把**ssthresh**设置为**cwnd**的一半**
  - 2.**把**cwnd**再设置为**ssthresh**的值(具体实现有些为**ssthresh+3)
  - 3.**重新进入拥塞避免阶段。**

实际慢启动

- **仅在连接开始和超时发生时,**
- **所有其它时间,**CW**是纯线性增加/乘式减少模式**

2.4 拥塞避免机制

- TCP的策略
  - 是一旦发生拥塞就来控制拥塞，而不是在第一个地方避免拥塞
  - TCP通过不断增加负载来影响网络的效果，从而找出拥塞发生点，然后从这一点后退。
  - TCP需要创造丢失来发现该连接的可用带宽！
- 拥塞避免策略（但未被广泛采用）
  - 拥塞将要发生时进行预测，在包将被丢弃之前减少主机发送数据的速率，存在3个不同的拥塞避免机制
  - 前2个相似：把少量功能件加入到路由器中，帮助端节点预测拥塞；第三个是纯粹由端节点来避免拥塞

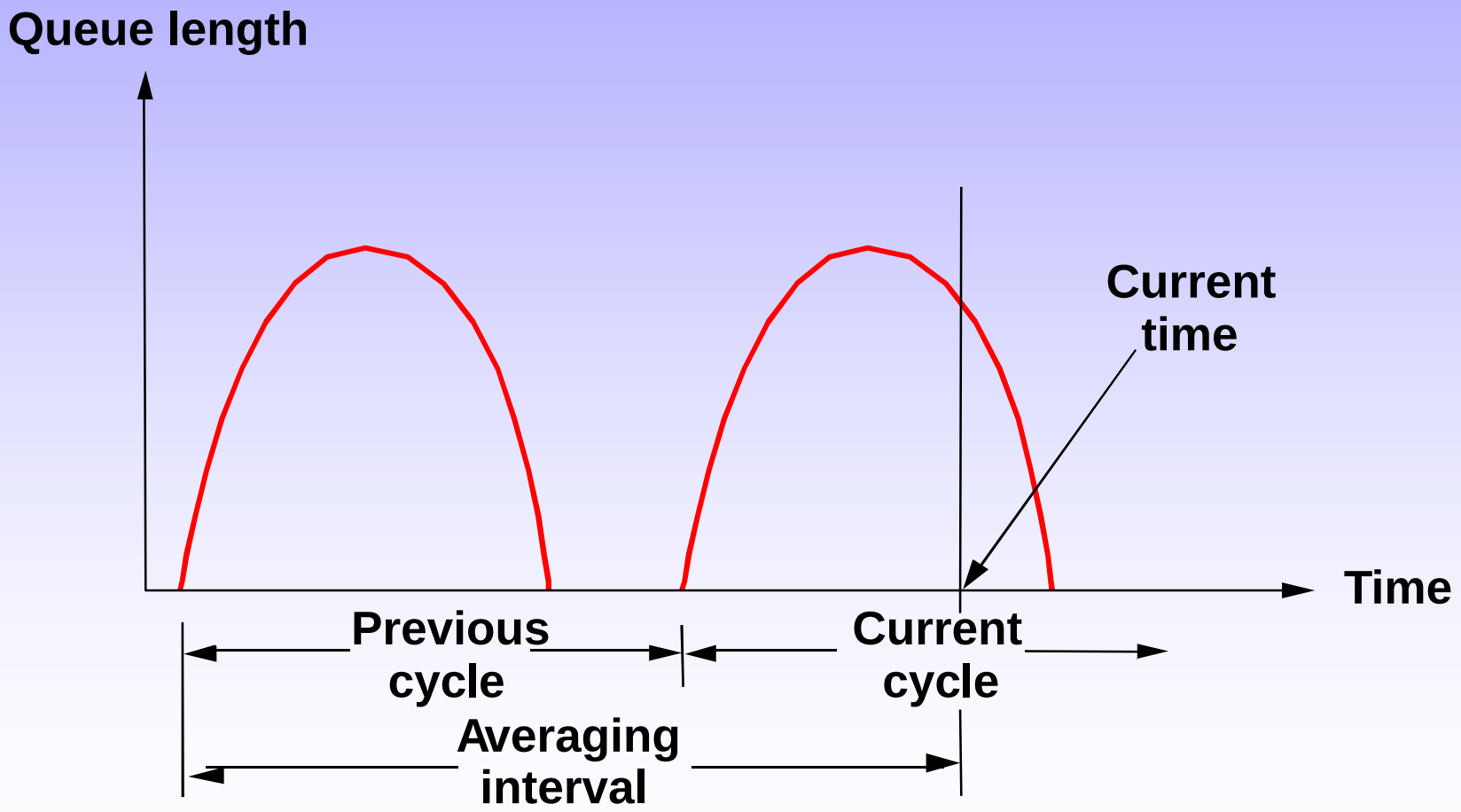
2.4.1 DECbit

- **第一种机制**
  - Digital Network Architecture **上开发使用，故也能用在**TCP/IP**上**
- **基本思想：**
  - **把拥塞任务更均匀分担在路由器和端节点之间**
  - **每个路由器监视它正在经历的负载，在拥塞将要发生时明确地通告端节点**
  - **通过设置流过路由器包中的二进制拥塞位来实现，因由**DEC**公司开发，故称** DECbit
  - **目端把这拥塞位拷贝进**ACK**并返送回源端**
  - **源端调整自己的发送速率以避免拥塞**

拥塞位的设置

- 当包到达
  - 且平均队列长度大于或等于1时，路由器就设置包头中的拥塞位
  - 测量平均队长的时间间隔是
    - 最近的忙期+空闲期+当前忙期（R传输时为忙，不传输时为闲）
  - 路由器计算曲线下面积并除以时间间隔得到平均队列长度
- 用队长1作为设置拥塞位的触发器
  - 是在有效排队(维持高吞吐率)和增加闲期(较低延迟)之间的折中,即队长1是能力（**吞吐率/延迟**）函数的优化

路由器中平均队长的计算



另一半机制--主机的工作

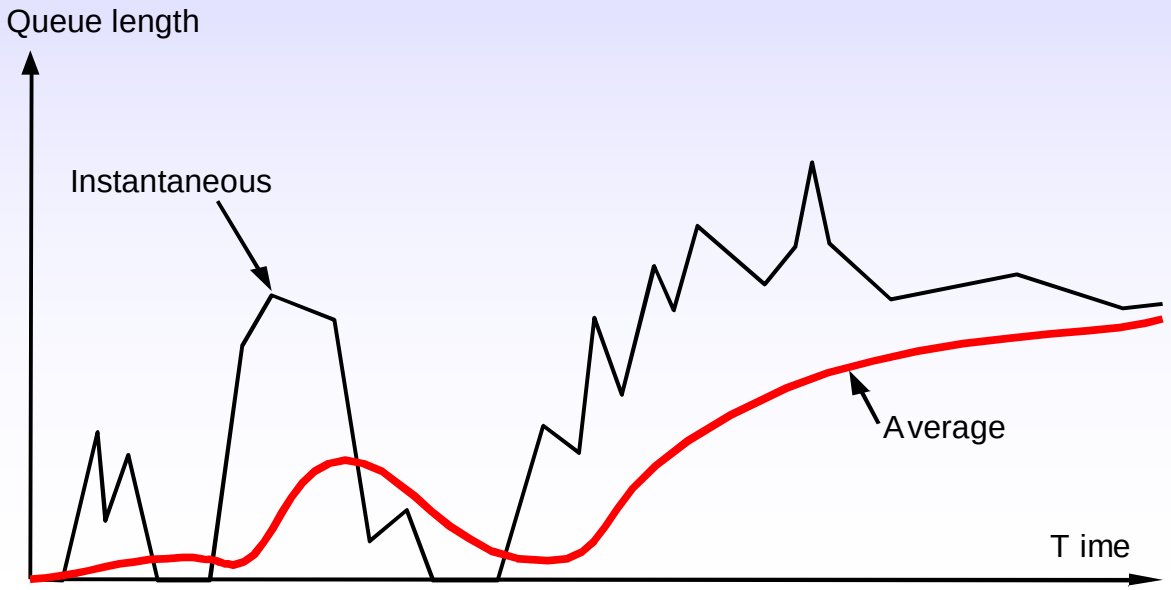
- 源主机的工作
  - 记录有多少包导致其在路由器中记下拥塞位
  - 维护一个拥塞窗口，并观察最近等价窗口值导致设置拥塞的比例，
    - 若比例<50%，源增加拥塞窗口1个包
    - 若比例>=50%，源减拥塞窗口至原值0.875倍
- 这符合累加/倍减机制

2.4.2 随机早检测-RED

- **第二种机制**：Random Early Detecation
  - 与DECbit相似，都是在每个R上编程监视自己队长，检测到拥塞就通知源调整窗口
- Early
  - R在其缓冲被完全填满前就丢少量包，使源方减慢发率
- RED与DECbit的不同
  - RED不明确发拥塞通知到源，而是通过丢弃一个包来隐含已发生拥塞。源会被随后的超时或重复ACK所提示，这可与TCP配合使用
  - 决定何时丢弃及丢弃哪个包的细节上不同，对FIFO，不是等队列完全排满后将每个刚到达包丢弃，而是当队列超过某阈值时按某概率丢弃到达包--

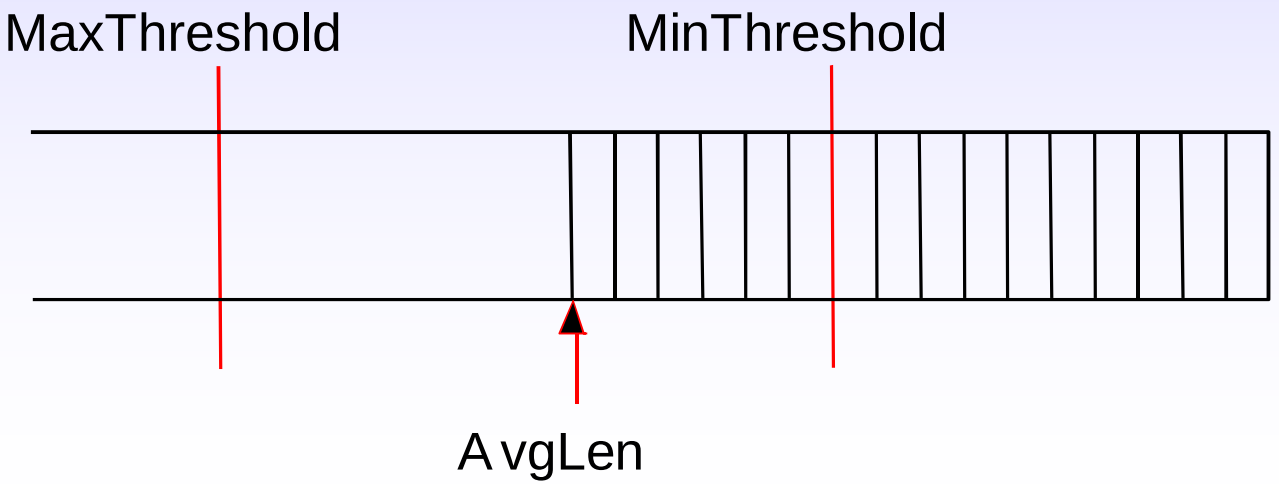
A：RED加权动态平均队长

- 与TCP超时计算中加权值类似：
  - Avglen=(1-Weight)×Avglen+Weight×Samplelen,
  - 其中0＜ Weight ＜1， Samplelen样本测量时对长
- 平均队长
  - 比瞬间队长更能准确捕获拥塞信息，瞬时队列会突然塞满或腾空，依此来通知主机升降发送速率并不合适
- 加权平均值
  - 是一个低通滤波器，加权值决定滤波器的时间常数
  - 图右半部分表示塞满或腾空



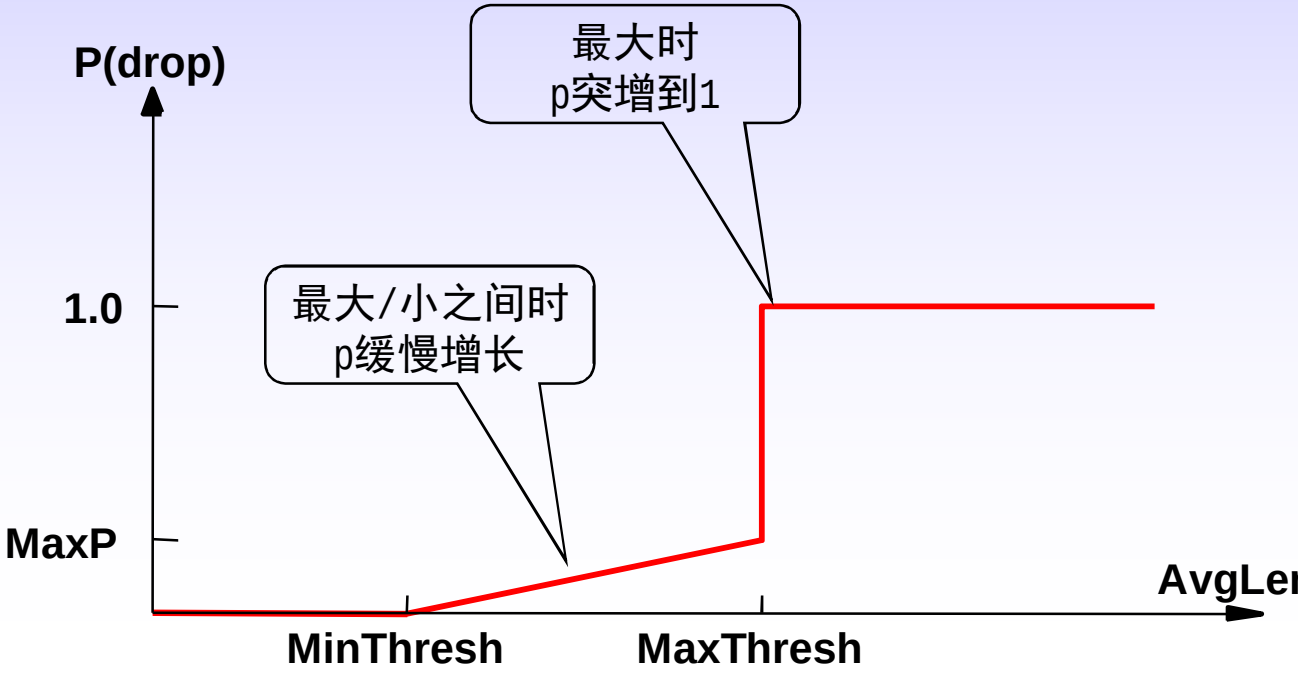
B：FIFO队长阈值的决定

- **最小和最大阈值**MinThreshold/MaxThreshlod
  - **当包到达时，将当前**Avglen**与最小和最大**2**个阈值比较**
    - If Avglen ≤ MinThreshold then queue packet
    - if MinThreshold < Avglen < MaxThreshlod then calculate probability p and drop the packet with probability p
    - if MaxThreshlod < Avglen then drop the arriving packet



主动丢包概率函数

- **实际丢包函数**
  - p**是**Avglen**和上个包被丢后距当前时间的丢包概率函数，**
  - TempP = MaxP×（Avglen-MinThreshold）/(MaxThreshold-MinThreshold)
  - P = TempP/(1-count×TempP)
  - TempP**是图y轴表示的变量**
  - count**记录有多少刚到的包已加入队列**
  - **可使丢弃随时间分布，防止单个连接的包成群丢失，从而导致慢启动**



RED丢包概率随时间均匀分布

- **假定初设置**
  - MaxP=0.02,count=0,Avglen**位于最大最小值中间**,
  - **则**TempP(P**的初值**)= MaxP/2=0.01**即**TempP = 0.02 × **(** Min+**Δ**/2-Min **)** /**Δ**=0.01 P = 0.01/(1-0),
  - **包有**99%**概率进队列**
- P**缓慢增加**,
  - **当**50**个包到达后**,P=0.01/(1-50*0.01)=0.02
  - **当**count=99**时(中间可能出现丢包)**,P=0.01/ (1-99*0.01)= 1,**保证下一分组一定丢失**
- **算法重点**
  - **保障丢弃概率大致随时间均匀分布**
  - **丢弃某特定流包的概率和该流在**R**上获得的带宽分额成比例--保证基本平均分配**

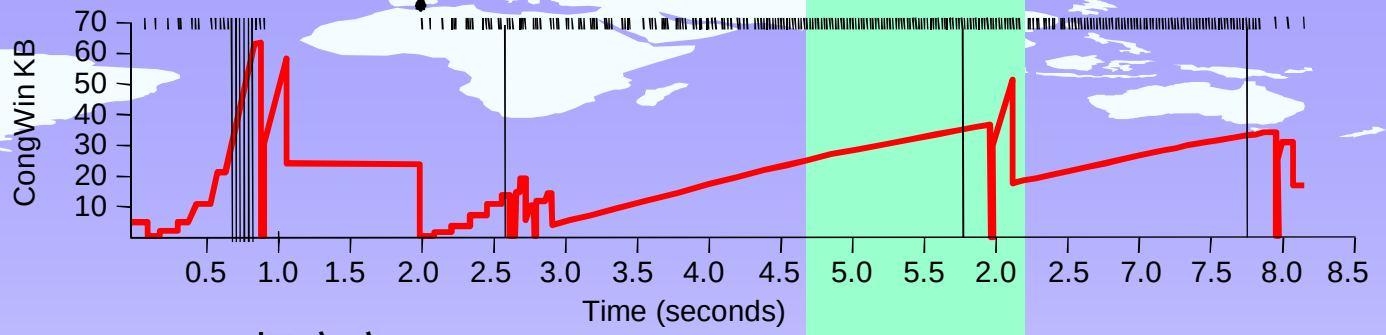
2.4.3 基于源的拥塞避免

- 丢包发生前,源用不同算法检测早期拥塞现象
  - 算法1:源观察到随R中包队列增大,后继包的RTT都会增加,如每隔2个RTT,检测当前RTT是否大于迄今所测RTT的最大最小的平均值,若大则把拥塞窗口减少1/8
  - 算法2:是否改变当前窗口根据RTT和窗口2个因素的变化而决定,每隔2个RTT,计算(CurrentWindow-OldWindow)* (CurrenRTT-OldRTT), 若 > 0 则窗口减少1/8,若≤0 则窗口加1最大包长,每次调整使其围绕最佳点震动
  - 算法3:拥塞前发送率接近平缓.每个RTT内将窗口加1个包,同时将获得的吞吐量与加1包前的吞吐量比较:若差小于只有1个包传输时所得吞吐量的1/2,即和刚建立连接时状态一样,则窗口减1.通过计算未发完字节数/RTT得吞吐量

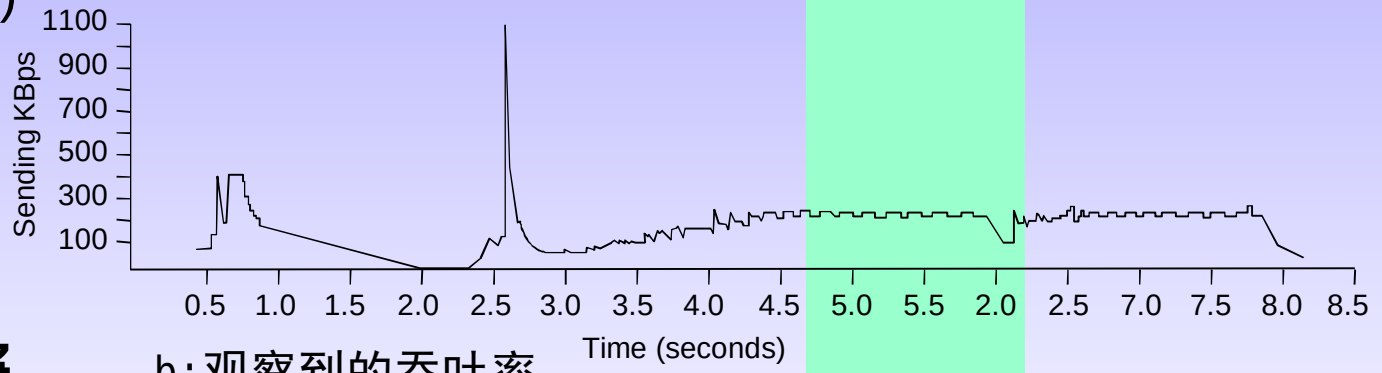
算法4:TCP Vegas

- 类似算法3
  - 查看吞吐率或发送率的变化,
  - 不同的是,它将测量的吞吐量变化率与理想吞吐量变化率比较
- 在第4.5--2.0秒之间(绿色部分):
  - 拥塞窗口的增加,希望吞吐率也随之增加(a),
  - 然它却保持平缓(b),这是因吞吐率不能超过可获得的带宽,
  - 窗口的增加只会导致包在瓶颈R缓冲空间占用(c)

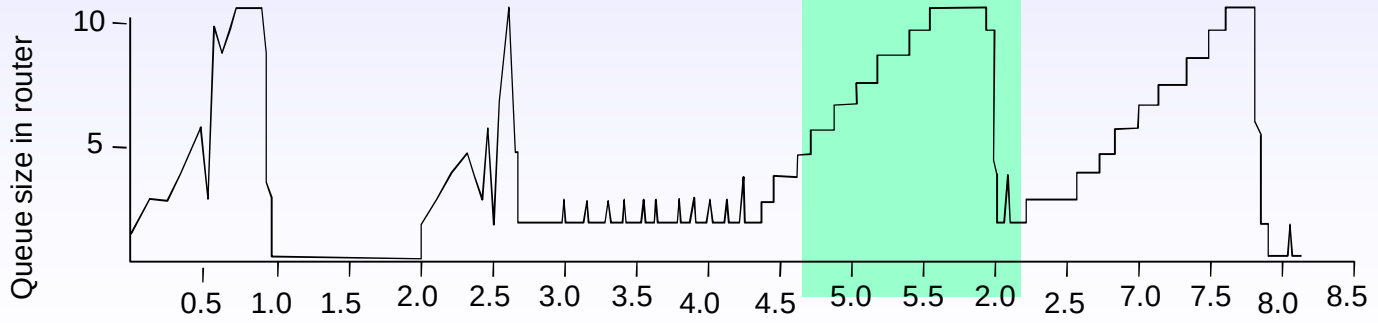
- 拥塞窗口与观察到的吞吐率(3图同步)
- 有色线=拥塞窗口
- 上方黑点=超时
- 散列符号=每个包被发送的时间
- 竖条=丢包/最宗将被传送的包的第一次传送时刻



a:拥塞窗口



b:观察到的吞吐率



c:R上占用的缓冲区

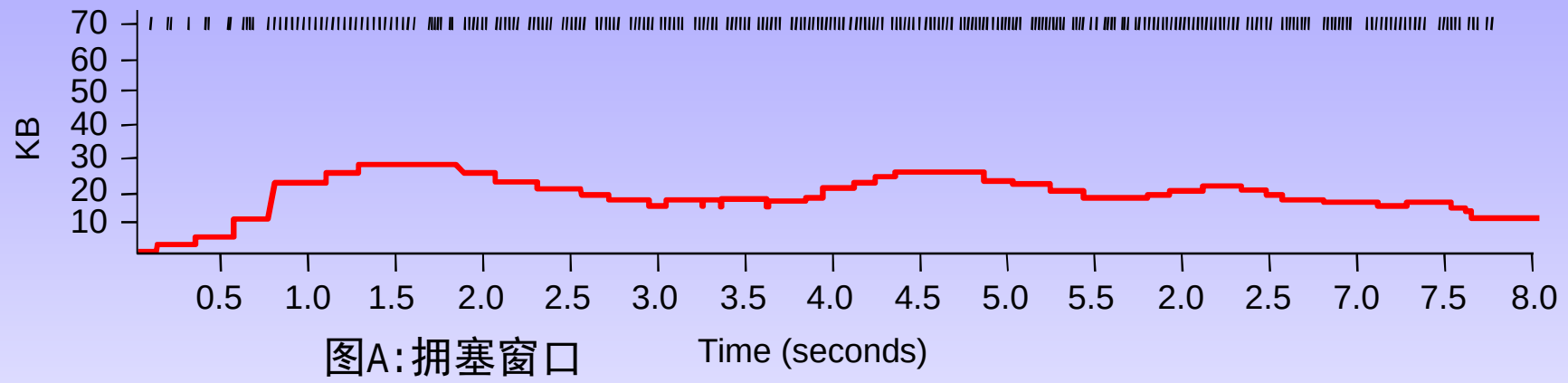
Vegas 算法

- Step1
  - **定义**BaseRTT**为给定流无拥塞时包的**RTT**值，**
  - **实为测得所有往返时间中的最小值，也是**R**溢出前该连接发送的第一包的**RTT**，希望吞吐率为**
  - ExpectedRate=CongestionWindow/BaseRTT
  - **假设**CongestionWindow**等于传送中字节数**
- Step2
  - **计算当前发送率**ActualRate, **记录每个包的发送时间和从发送到确认间包的字节数**,
  - **收到确认后计算**RTT,**相除得发送率**,**每个**RTT**计算一次**

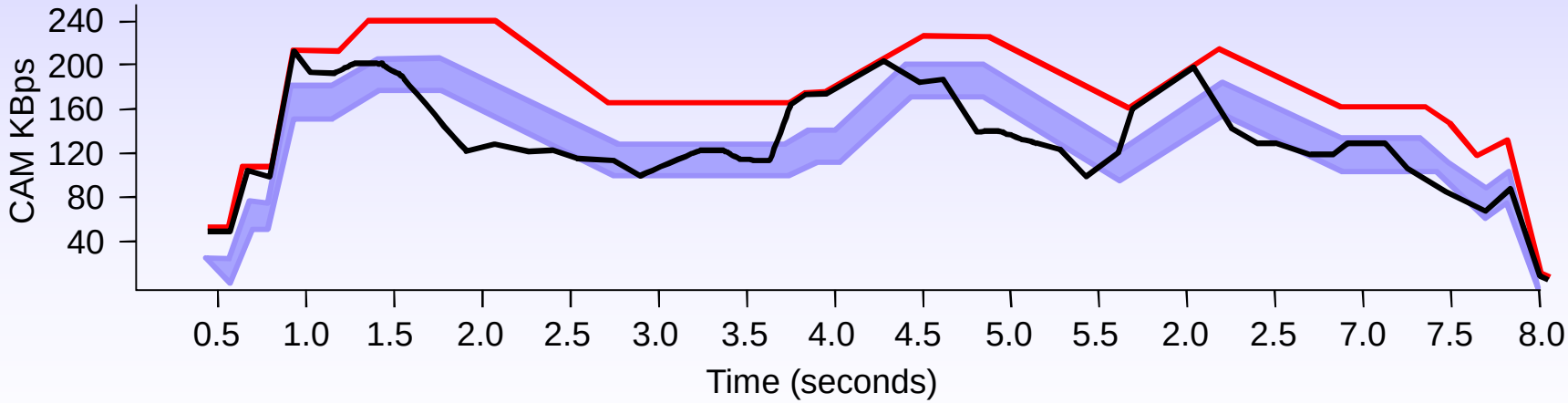
用差值调整拥塞窗口

- Step3
  - **比较**ActualRate**和**ExpectedRate,**并调整窗口**,
    - Diff = ExpectedRate - ActualRate(**按定义应≥0**),**并定义阈值** **α < β**,**大致对应网络中太少和太多的额外数据**
    - **当**Diff < α ,**则**TCP Vegas **在下一个**RTT**中线性增加拥塞窗口**;
    - **当**Diff > β,**则在下一个**RTT**中线性减少拥塞窗口**
    - **当**α < Diff < β TCP Vegas **保持拥塞窗口值不变**
- **可以看出**
  - **实际吞吐率和希望吞吐率差值越大,表明拥塞越大,故β使发送率应下降**;
  - **而当二者太接近时,该连接可能不能充分利用带宽, α使窗口增加**

TCP Vegas拥塞避免机制的过程

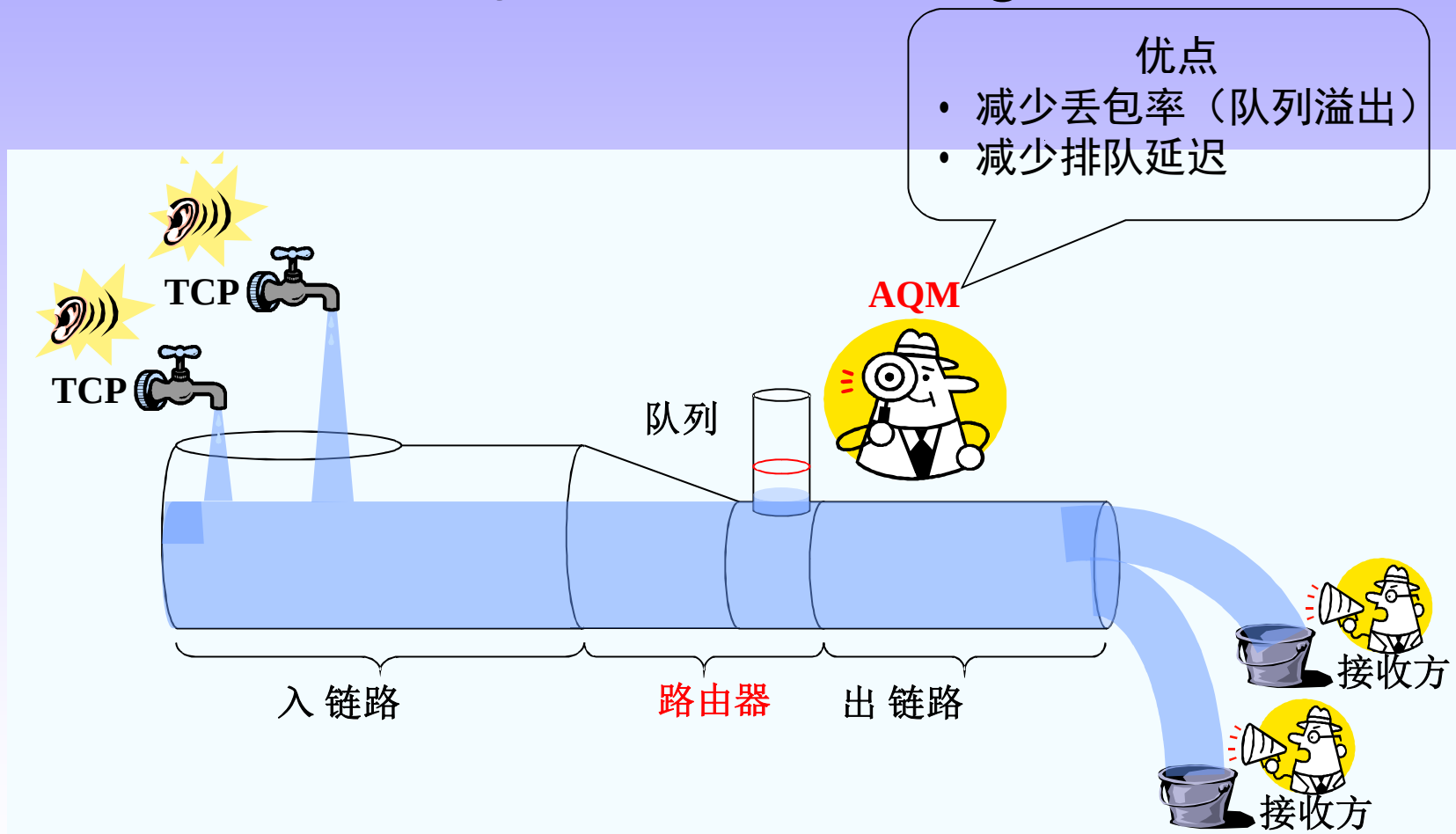


图A:拥塞窗口



图B:期望(红色)和实际(黑色)吞吐量,紫色阴影区是α,β阈值中间的区域

2.5 主动队列管理AQM
--Active Queue Management



为什么要 AQM ？

- 控制平均队长
- 吸收没有丢弃的突发流包
- 防止非TCP友好流仰仗突发连接的锁外行为
- 避免TCP的全局同步现象
- 减少TCP的超时次数
- 惩罚行为不端的流

2.5.1 基于队列的AQM

- RED Random Early Detection [Floyd and Jacobson, 1993]
  - **丢包概率与平均队长（缓冲区占用率）成比例**
- SRED Stabilized RED [Ott et al., 1999]
  - **丢包概率与瞬时缓冲区占用率和活动流的估计数成比例.**
- FRED Flow RED [Lin and Morris, 1997]
  - **使每个流的丢包率与其平均队长和瞬时缓冲区占用率成比例**
- BLUE [Feng et al., 2001]
  - **用丢包事件和链路空闲事件来管理拥塞。**
  - **当丢包率增加时增加丢包率；当链路空闲时减少丢包率。**

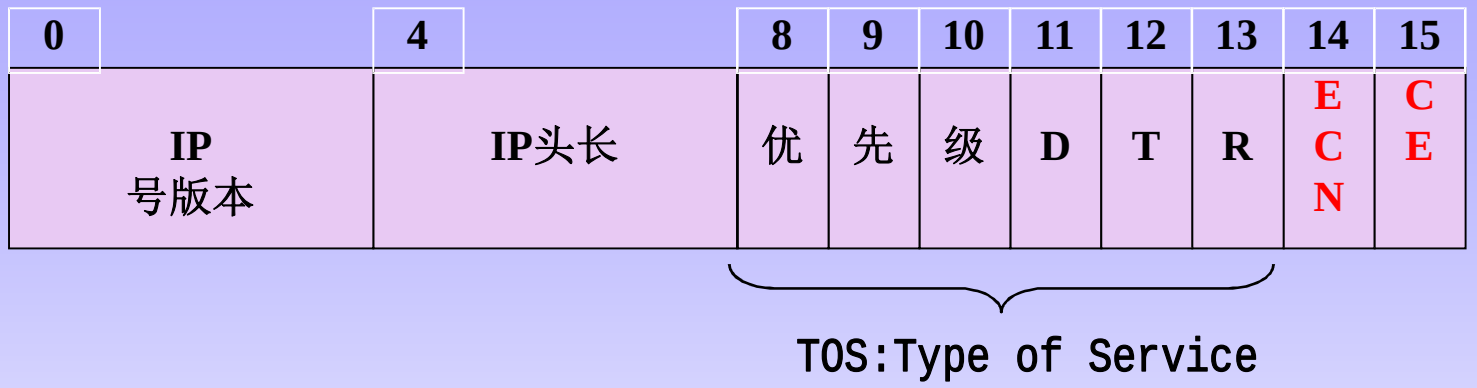
2.5.2 基于负载的AQM

- REM Random Exponential Marking [Athuraliya et al., 1999]
  - **通过输入和输出之间速率差及路由器中当前缓冲区占用率来计算拥塞的尺度**
  - **源的发送速率与拥塞尺度成反比，于是源达到全局优化平衡**
- AVQ Adaptive Virtual Queue [Kunniyur and Srikant, 2001]
  - **维持一个虚拟队列.当其队长溢出时，实际队列中的包就标记或丢弃**
  - **修改虚拟容量使所有流达到希望的链路利用率**
- PI (Proportional Integral) controller[Hollot et al., 2001]
  - **队列长度的变化决定丢包率，队列调节到由锁希望的队长**

2.5.3 显式拥塞控制

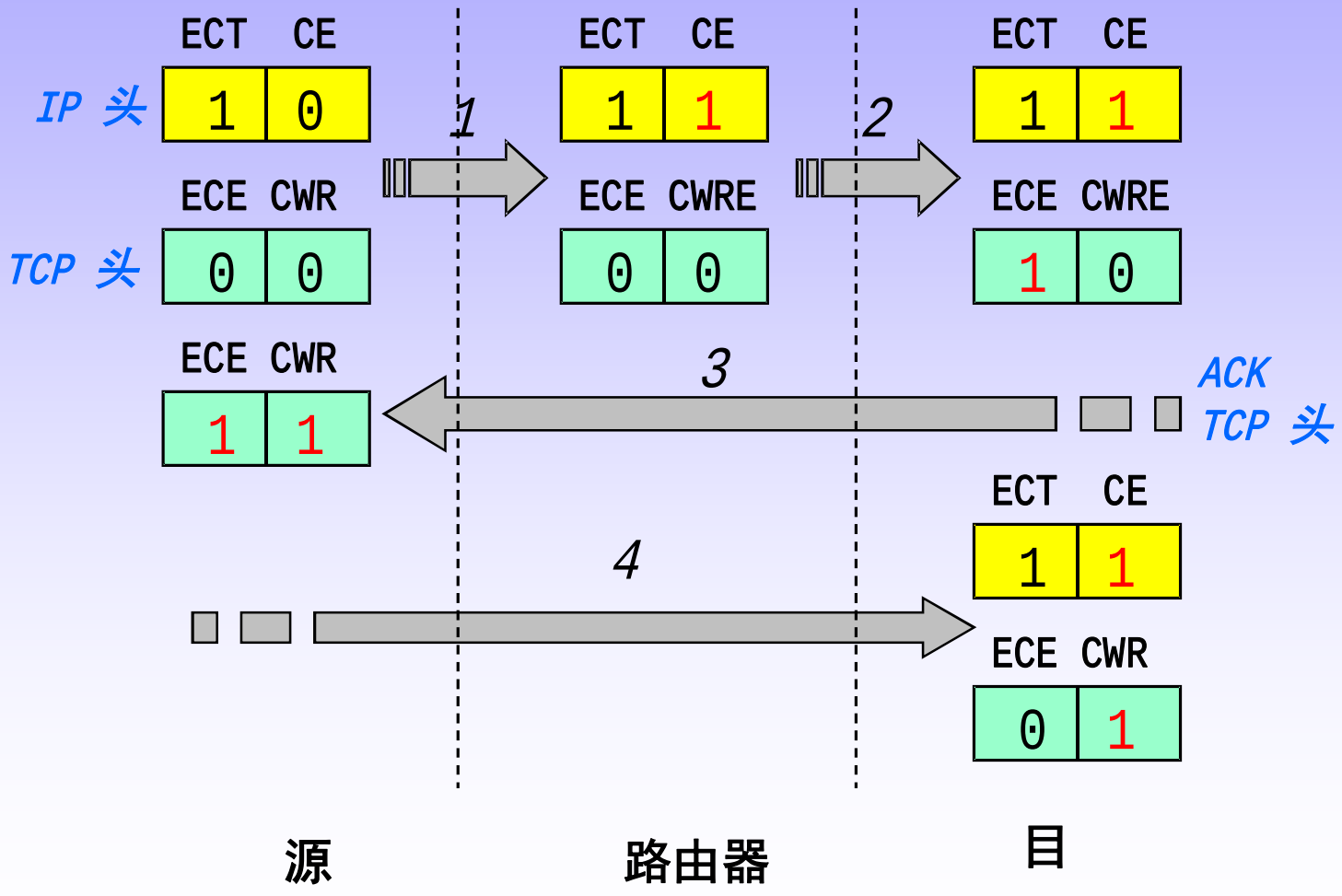
- ECN:Explicit Congestion Notification，IETF RFC 2481
  - **标记包而不是丢弃包**
  - **减少超时长度和重传次数**
- **一种拥塞通知方式；格式分配如下**
  - IP**头中设置** 2 bits **的**ECN**域，**TOS**字段中的第**6、7**位分别定义为**ECT **和** CE **（**V4**的**TOS**中的前**6**位是区分服务）**
    - ECN Capable Transport(ECT):**由源对所有包设定，以指明**ECN **能力**
    - Congestion Experienced(CE)**：由路由器设置作为拥塞的标记（代替丢弃）**
  - TCP**头中设置**2 bits **，**ECE **和** CWR**标志位**
    - ECE**：**Echo Congestion Experienced ,**但接收方收到**CE,**在所有包上设置** ECE**，直到收到源的**CWR
    - CWR:Congestion Window Reduced**，由源设置，以指明它收到了**ECE**并减少了窗口大小**

ECN 的表达

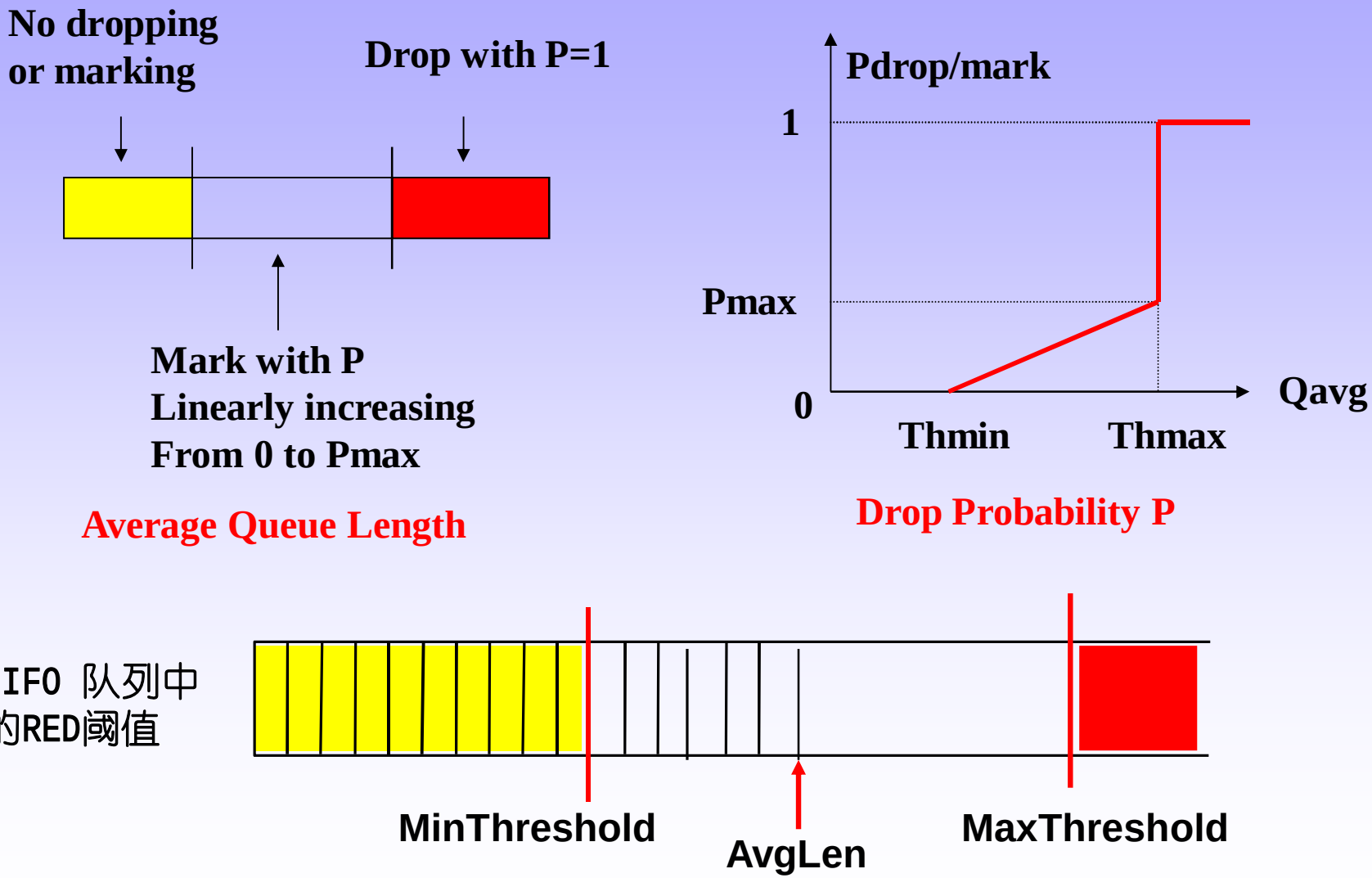


- **需要源与网络合作**
  - **源须用ECN指明开启；目同意使用；**
  - **目须用ECE通知源；源须同丢包一样反应,并置CWR**

ENC 的工作过程


ECT CE
IP 头
1 0
1
ECT CE
1 1
2
ECT CE
1 1
ECE CWR
TCP 头
0 0
ECE CWRE
0 0
ECE CWRE
1 0
ECE CWR
1 1
3
ACK
TCP 头
ECT CE
1 1
4
ECE CWR
0 1
源
路由器
目

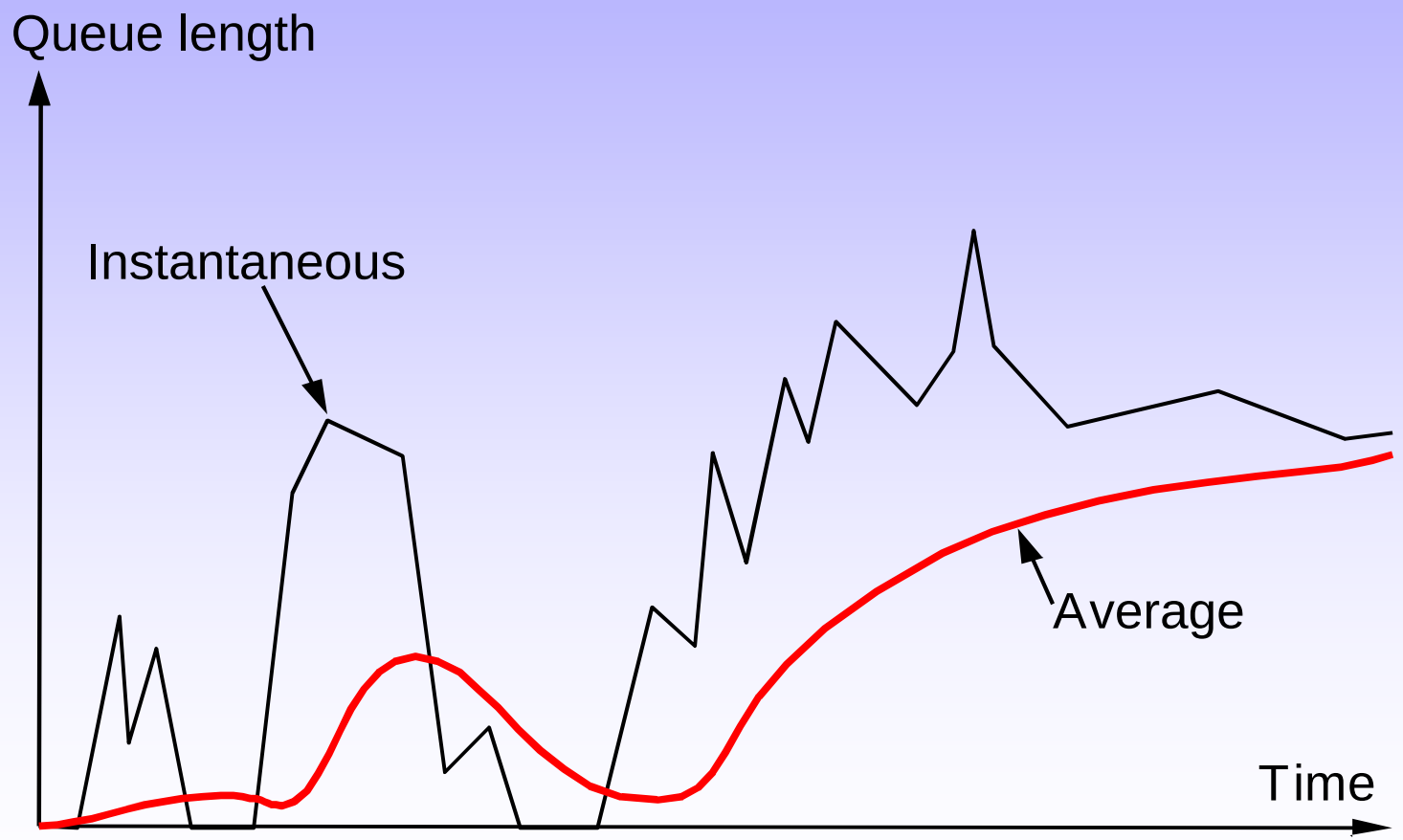
2.5.4 RED 簇基本原理



随机早检测机制

- RED：监控R输出端口队列的平均队长来探测拥塞
- Early:在其缓冲被完全填满前就丢少量包，使源方慢发送速率
- RED与DECbit的不同
  - RED不显式发拥塞通知到源，而是通过丢弃一个包来隐式已发生拥塞。源会被随后的超时或重复ACK所提示，这可与TCP配合使用
  - 决定何时丢弃及丢弃哪个包的细节上不同，对FIFO，不是等队列完全排满后将每个刚到达包丢弃，而是当队列超过某阈值时按某概率丢弃到达包--early random drop
- 在适当时候丢弃正进入R的包，算法实施简单
- 目标:最小化丢包率和排队延迟，避免全局同步和队突发业务的偏见（而去尾算法表现出对突发的偏见）

加权动态平均对长



Stabilized RED (SRED)

- 目标：
  - 保持FIFO队列占用稳定，且与活跃流（Active flows）数量无关
  - 鉴别恶意流，方法是到达的包同Buffer中随机选择的包进行比较，若属同一个流，则为“命中”
- 方法：
  - 在流Cache中记录M个流及其额外信息：“命中计数”和“命中包到达时间戳”
  - 函数Hit(t)记录第t个包是否击中，击中，则Hit(t)取值为1，否则为0
  - 如果一个包“命中”且该count值很大，则可认为它属于恶意流，丢包P增大

Flow RED

- **对每活跃流（per-active-flow）进行记帐（accounting），从而对使用不同带宽的流作出不同的标记包**
- RED**的改进版本，**主要目的是解决**RED**算法的公平性问题
  - RED**是基于带宽使用的比例，随机地选择一个流标记其包，来通知拥塞；**
  - **而**FRED**是从缓冲区中有较多排队包的那些流中选择一个，向其发出拥塞通知。**

ARED: Adaptive RED

- 拥塞通知时应充分考虑到瓶颈链路上流的数量
  - 而RED并没有考虑到这一点。
  - ARED提出了一种自动配置机制，根据流量的变化来配置适当的参数。
  - 根据平均队长是否在min_th和max_th之间，对maxp采用积式增加和减少（Multiplicative Increase Multiplicative Decrease，MIMD）从而尽量保持平均队长在min_th和max_th之间

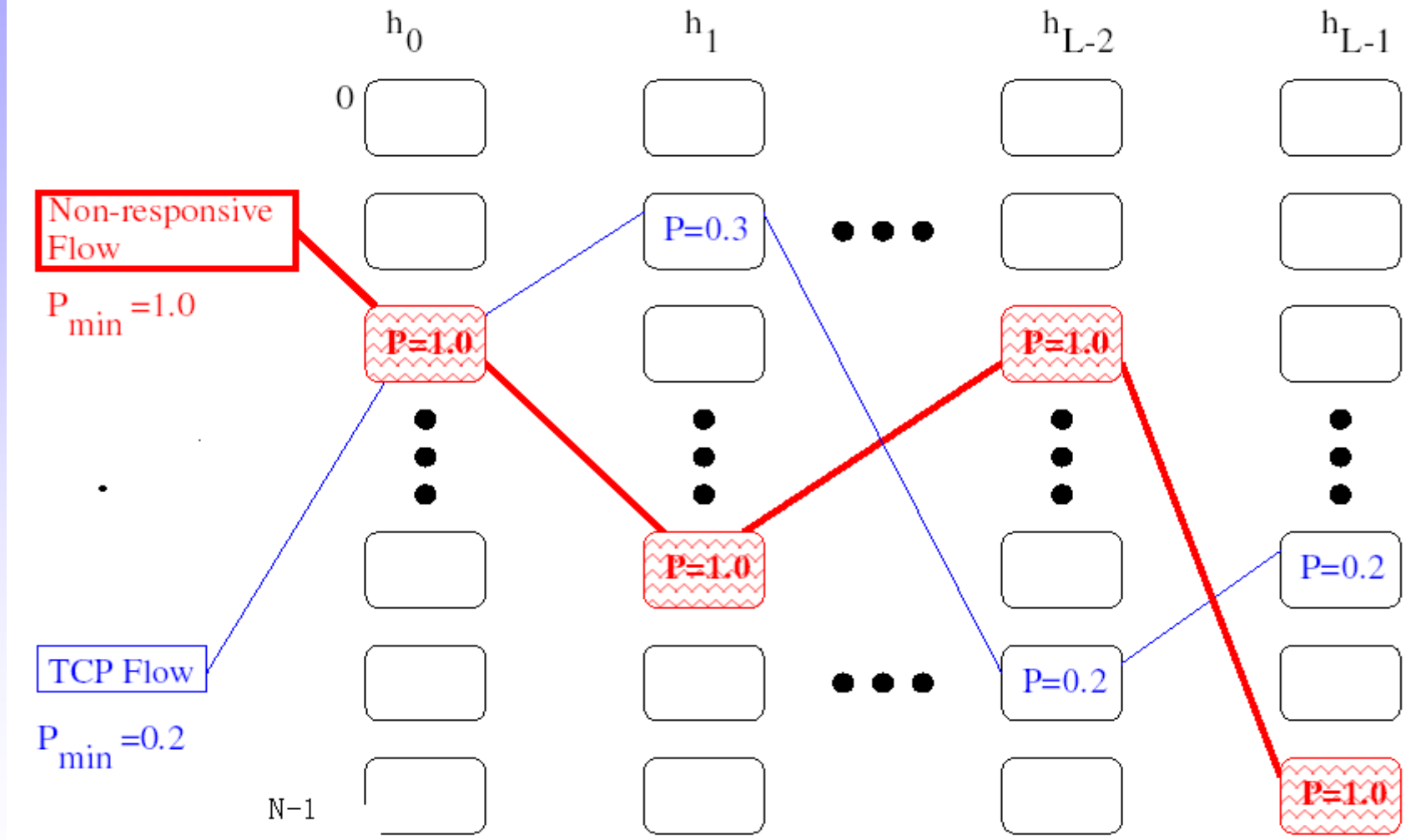
2.5.5 BLUE

- **用丢包事件和链路空闲事件来管理拥塞**
  - 如果由于队列溢出导致持续的丢包，BLUE就增加Pm，**因而增加了向源端发送拥塞通知的速度**
  - 如果由于链路空闲导致队列空，BLUE**就减小**Pm
- BLUE**最大贡献在于：**
  - **使用相对较小缓冲区就能够完成拥塞控制**
  - 减少端到端延迟，提高TCP流的吞吐量
  - 标记包比率相对较稳定，使得队长也相对稳定，减少了延迟抖动

SFB: Stochastic Fair BLUE

- 越来越多的应用不采用TCP的拥塞控制机制，在竞争带宽时使适应流陷入“饥饿”
- SFB：采用记帐的方法鉴别非适应流并对它们进行速度限制
  - 维持N×L层记帐“柜子”（accounting bins），L个Hash函数与每层柜子关联
  - 每个Hash函数将一个流映射到该层的一个柜子。每个柜子有一个标记包的概率Pm（基于队列占用）
    - 若映射到某个柜子的包的数量超过了一定的阈值，则该柜子的Pm便会增加；
    - 如果该柜子中包的数量降为0，Pm便会下降。
    - 取该包所映射到的L个柜子的标记包概率的最小值Pmin，如果该值为1，就认为该包所属的流是非适应流，从而对其限速。否则以Pmin的概率来标记包

SFB 的例子


h_0
h_1
h_{L-2}
h_{L-1}
0
Non-responsive Flow
P_{min} =1.0
TCP Flow
P_{min} =0.2
N-1
P=0.3
P=1.0
P=1.0
P=1.0
P=0.2
P=0.2
P=1.0

2.5.6 GREEN

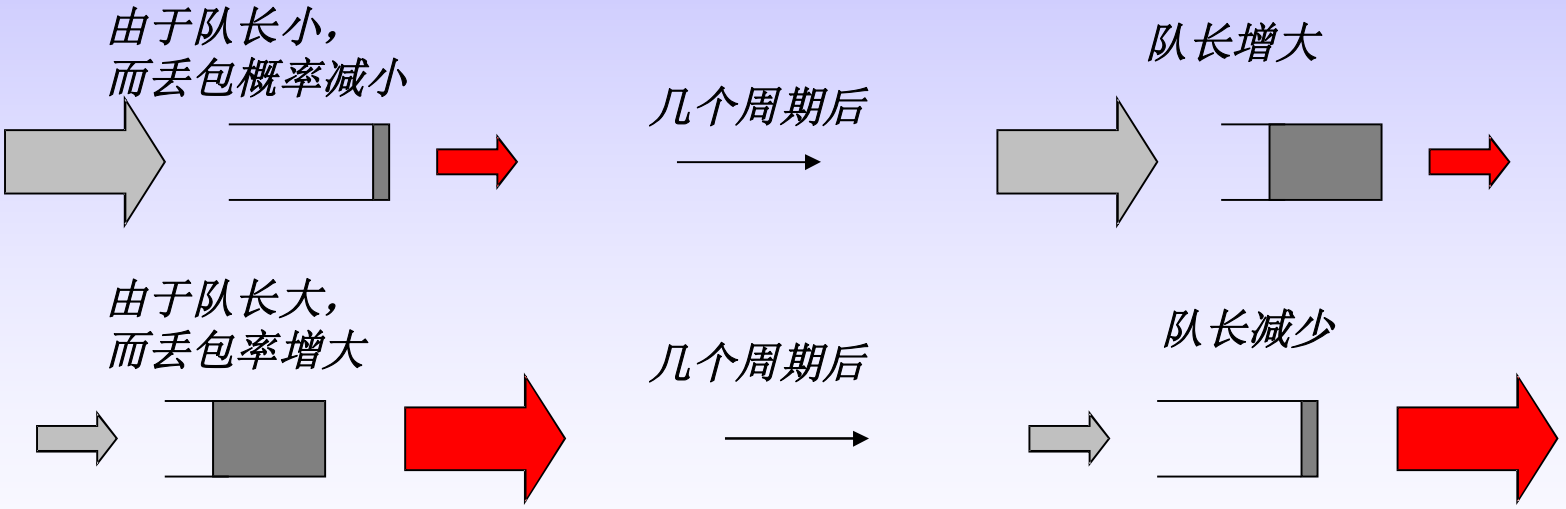
- 包到达路由器的速度来判断拥塞程度，根据拥塞程度来调整发送拥塞通知的速度
- 如果包到达的速度超过了链路的容量，那么在某一时间间隔内，拥塞通知速度增加一次；
- 反之，如果包到达的速度小于了链路的容量，那么在某一时间间隔内，拥塞通知速度减少一次

2.5.7 CHOKe

- CHOose and Keep for responsive flows
- **保护适应流，惩罚非适应流，其基本思想是**
  - **当一个包到达拥塞的路由器时，从FIFO队列中随机地挑出一个包进行比较。如果它们属于同一个流，则这两个包都被丢弃；**
  - **否则，被挑出包依然留下，而刚到达的包则依某种概率被丢弃，此概率的计算和RED中一样**
  - **CHOKe算法是基于以下原因：FIFO队列可能会更多地包含非适应流包，也就意味着这种流的包更有可能被选中进行比较。从而对非适应流进行了惩罚**
- **CHOKe基本上继承了RED的优点，只是少量增加了一些额外开销**

2.5.8 基于负载的AQM

- **基于队列AQM的缺点**
  - **对当前队列到达率和丢弃率非常敏感**
  - **平均队列长度的长周期导致很慢的响应时间**
  - **参数构造困难**



- **只用队长来指示拥塞足够了吗？能否用负载信息来更精确指示拥塞？**
- **用负载信息后，队长信息还重要吗？能达到高响应和高吞吐率吗？**

2.5.9 LDC: Load/Delay Control

- 负载因子
  - R = 队列的到达率/ 队列的服务率
- 特点：同其他load-based 方法不同,LDC不需要ECN
- 思想：用队长和负载因子2种信息来多重指示拥塞
  - 长期：队长给出更稳定（但较慢）拥塞指示
  - 短期：负载因子能加快响应
- 目标:
  - 使排队延迟保持在目标值之下，队列的达到率在队列的服务率之下
  - 提供更好的负载(input/output rate)响应和更直观参数
  - 保持 RED 优点: 惩罚恶意流，并避免全局同步

LDC 的丢包概率

$$P_{\text{drop}} = \frac{3}{4}\min\left(1, \left(\frac{Q_{\text{avg}}}{Q_{\text{target}}}\right)^{n}\right) + \frac{1}{4}\min\left(1, \left(\frac{R_{\text{avg}}}{R_{\text{target}}}\right)^{n}\right)$$