

Ch2: 拥塞控制与QoS

2.1 拥塞控制概念

2.1.1 拥塞控制的基本原理

◆ 问题:

- ♣ 处于竞争的用户集合内**怎样有效和公平地分配**有限资源
- ♣ 怎样共享资源: 包括链路带宽、路由器、交换机中的缓冲区(包在此排队等待传输)和处理机时间等

◆ 拥塞 (congestion)

- ♣ 当过多的包在网络缓冲区中竞争某个相同链路时, 队列会溢出丢包, 当这种丢包成为普通事件时, 则称网络发生**拥塞**

$$\sum_{i=1}^n \text{被请求资源}_i > \sum_{j=1}^m \text{能可用资源}_j$$

♣ Fig. 2.1

◆ 拥塞产生的原因（需求大于供给，无准入控制）

- ♣ 宏观原因：网络资源分布不均匀，流量分布不均匀，
- ♣ 微观原因：聚合报文到达率大于路由器输出链路的带宽

◆ 拥塞的后果

- ♣ 队列长期满，报文排队延迟增加
- ♣ 拥塞时，队尾报文同时丢弃，相关TCP连接同时减少发送窗口；不拥塞时，同时开始增加发送窗口。全网同步振荡，吞吐量下降

◆ 解决办法

- ♣ 告诉几个主机停止发送，从而改变所有主机的情况
- ♣ 针对某个因素的解决方案，只能对提高网络性能起到一点点好处，甚至可能仅仅是转移了影响性能的瓶颈；
- ♣ 需要全面考虑各个因素。

◆ 拥塞控制

- ♣ 描述网络节点为防止或响应超载情况所做的努力
- ♣ 控制的首要任务就是消除拥塞或预防它在第一个地方

拥塞控制和资源分配是硬币之两面

◆ 若**主动分配资源**，则拥塞控制就无必要

- ♣ 如在某时段内，给定物理电路来获得**虚电路**，避免拥塞
- ♣ 但跨域（国家、洲、ISP）精细的资源分配是困难的，因为问题中的资源分布在网络上，需要调度连接**一系列路由器的多条链路**

◆ 若**主动拥塞控制**：发方开始任意，拥塞发生后再将其恢复

- ♣ 简易，但拥塞得到控制之前，网络可能就已开始丢包
- ♣ 网络拥塞时，资源已不足，对竞争用户间的资源分配更**敏感**

拥控与流控之差别

◆ 数据流：

♣ 具有如下若干相同特征的数据包

☞ 路由流：源地址、目标地址

☞ 应用流：源端口号、目的端口号、协议类型、服务类型TOS

☞ Fig. 2.2

◆ 拥控：防止一些发送者**把太多的数据**发送到网路中，因为在某些点上存在资源贫乏

◆ 流控：防止**发送方的发送速度比接收方的接收速度快**



◆ 拥塞控制与流量控制的差别

- ♣ 拥塞控制（congestion control）与全网有关，涉及**多个端到端、主机、路由器**等很多网元；目的是确保通信子网能够承载用户提交的通信量，是一个全局性问题，
- ♣ 流量控制（flow control）只与**一对端到端的通信量**有关，只涉及快速发送方与慢速接收方的问题，是局部问题，一般都是基于反馈进行控制的
- ♣ 要防止这两者的混淆，但它们有些机制是相同的

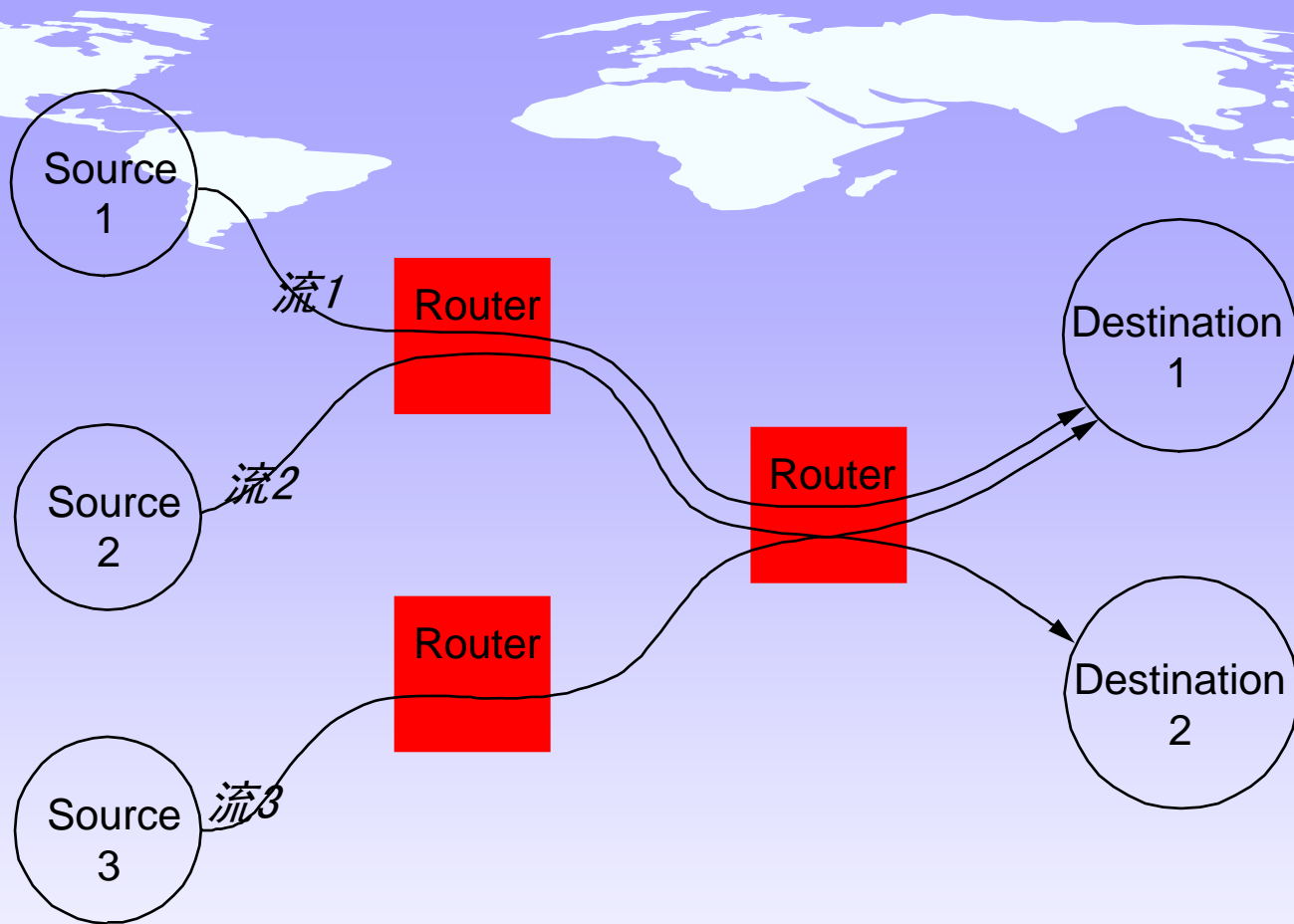
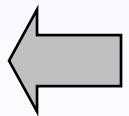


Fig. 2.2 多个流通过一组路由器



影响拥塞的网络策略

Layer	Policies
Transport	<ul style="list-style-type: none">• Retransmission policy• Out-of-order caching policy• Acknowledgement policy• Flow control policy• Timeout determination
Network	<ul style="list-style-type: none">• Virtual circuits versus datagram inside the subnet• Packet queueing and service policy• Packet discard policy• Routing algorithm• Packet lifetime management
Data link	<ul style="list-style-type: none">• Retransmission policy• Out-of-order caching policy• Acknowledgement policy• Flow control policy

2.1.2 拥塞控制方法分类

根据控制是否开闭环分类

◆ 开环控制

- ♣ 预留资源，避免拥塞发生；
- ♣ 开环控制，不考虑网络当前状态；

◆ 闭环控制

- ♣ 基于反馈机制；
- ♣ 工作过程
 - ☞ 监控系统，发现何时何地发生拥塞；
 - ☞ 把发生拥塞的消息传给能采取动作的站点；
 - ☞ 调整系统操作，疏通通道。

根据算法的实现位置分类

◆ 拥塞控制算法分为两大类：

- ♣ 链路算法 (Link Algorithm). 在**网络设备中** (如路由器和交换机) 执行, 作用是检测网络拥塞的发生, 产生拥塞反馈信息
- ♣ 源算法 (Source Algorithm). 源算法**在主机和网络边缘设备中**执行, 作用是根据反馈信息调整发送速率

源拥塞控制算法-TCP层

- 源算法中使用最广泛的是TCP协议中的拥塞控制算法. TCP是目前在Internet中使用最广泛的传输协议.
- 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-早期较为普遍采用版本
Reno-TCP	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

链路拥塞控制算法

- 链路算法的研究目前主要集中在主动队列管理AQM算法方面.
- 下表给出主动队列管理算法的简单描述

主动队列管理算法	描述
RED	比例控制器+低通滤波器.
ARED	根据网络负载的情况调整标记/丢失概率.
SRED	通过估计网络中流的个数来调整报文标记/丢失概率.
BLUE	以“分组丢失率”和“链路有效利用率”作为拥塞是否发生的标准.
REM	利用了网络流量优化理论中“价格”的概念来探测和控制网络拥塞.
AVQ	使用PI控制器, 控制的是队列的入口速率.
PI	使用PI控制器, 控制的是队列长度.

◆ 衡量网络是否拥塞的参数

- ♣ 缺乏缓冲区造成的丢包率；
- ♣ 平均队列长度；
- ♣ 超时重传的包的数目；
- ♣ 平均包延迟；
- ♣ 包延迟变化（Jitter）。

◆ 反馈方法

- ♣ 向负载发生源发送一个告警包；
- ♣ 包结构中保留一个位或域用来表示发生拥塞，一旦发生拥塞，**路由器将**所有的输出包置位，向邻居告警；
- ♣ 主机或路由器主动地、周期性地发送探报（probe），查询是否发生拥塞。

折中方案

- ◆ 不精确的资源分配，但拥塞仍然发生，故仍需要某些解除拥塞的机制
- ◆ 拥塞控制和资源分配包括主机和象路由器等网络元素
 - ♣ 在网络元素中，可用各种排队策略来控制哪些包发送或丢掉
 - ♣ 排队策略还可隔离，即使某个用户的包不过度地影响另一个用户的包
- ◆ 在端主机，拥塞控制可协调源发送包的速度

资源分配中的问题

- ◆ 资源分配和拥塞控制是一个复杂的问题, 从设计的第一个网络以来就一直是研究的课题. 现在仍然是一个活跃的研究领域
 - ♣ 复杂的原因在于它不是只涉及一个孤立的单协议层
 - ♣ 资源分配分别在网络内的路由器或交换机及运行在端主机上的传输协议中实现
 - ♣ 端系统用信令协议把其资源请求传输到网络节点, 这些节点回答资源是否可用的信息

2.2 网络模型

◆ Internet网络结构 3 个显著的特点

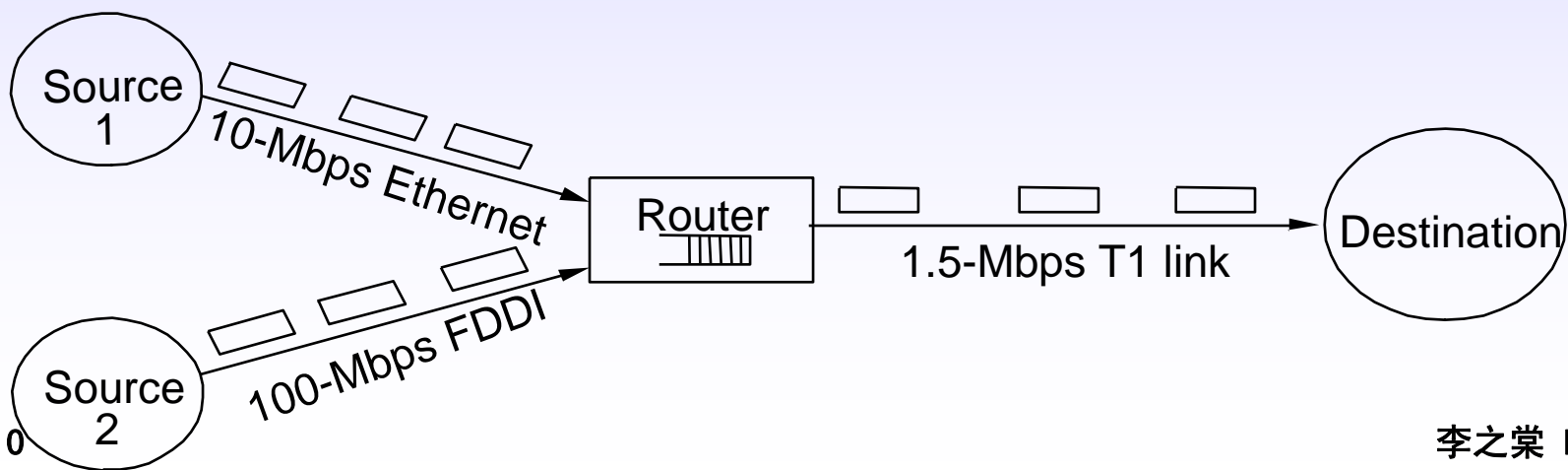
- ♣ **包交换**网络
- ♣ **无连接IP流**
- ♣ **Best-effort服务模型**

◆ 互联网的主要技术特点

- ♣ 分布式分层体系结构
- ♣ 路由器加专线的网络结构
- ♣ 可扩展的路由技术
- ♣ 端到端的网络通信技术
- ♣ 层次结构的域名、网络管理技术
- ♣ 开发通用的应用技术

I 包交换网络中的拥塞

- ◆ 在由多条链路和交换机（路由器）组成的包交换网络中的资源分配
 - ♣ 网中路由器与内联网中的交换机问题相同
- ◆ 在这样环境下：
 - ♣ 某个源在其直连的链路上可能有**足够的容量**去发送包
 - ♣ 但在网络中间的某处该包须经过的链路**正在被许多不同的流**在使用



拥塞控制不同于路由

- ◆ 通过路由传播协议，可把拥塞的链路设计为**大权边而绕过它**。但并没有解决拥塞的问题
- ◆ 稍微复杂的是：所有流量要通过相同R而到达目的
- ◆ 更通常是：**某些R不可能去绕着走**，故在此拥塞，这儿没有任何可行机制
- ◆ 这种R就称为瓶颈路由器

II 无连接流

◆ IP网络本质是无连接的

- ♣ 任何面向连接服务是由运行在端主机上的传输层实现的
- ♣ IP无连接，TCP实现端-端抽象连接

◆ 流

- ♣ 在源目主机对之间，流过网络中相同路由而发送的一系列包
- ♣ 在资源分配的上下文中这是一个重要的抽象
- ♣ 流同通道（Channel）本质上是相同的，不过“流”对网络R是可见的。通道是端-端的抽象

III 服务模式

◆ 因特网Best-effort服务模式

- ♣ 每个包都以相同的方法FIFO处理,
- ♣ 不保障主机对网络流的要求

◆ 多QoS

- ♣ 支持某种有保障的服务模式
- ♣ 如保障视频流的带宽

2.3 资源分配机制

- ◆以路由器为中心与以主机为中心
- ◆基于预留与基于反馈
- ◆基于窗口与基于速率

R/H为中心的分配

- ◆ **以R为中心之设计**，定位在网络内部
 - ♣ R决定何时转发包，决定丢掉哪个包，
 - ♣ 通知正产生网络流量的那些主机允许它们发送包代数目
- ◆ **以H为中心之设计**：定位在网络边缘
 - ♣ 端主机观察网络状态（多少个包成功通过了网络）
 - ♣ 并因而调整其行为
- ◆ **二者并不互斥**，
 - ♣ R中心的拥塞管理仍需要H执行R发送的建议消息
 - ♣ H中心的拥塞管理业需要R丢弃R溢出时那些包

基于预留与基于反馈

◆ 基于预留

- ♣ 建立流时，端主机请求网络给予一定的容量，每个R分配足够的资源（缓冲区或链路带宽的比例）来满足这一请求。
- ♣ 如果由于资源的过量则R不能满足该请求而拒绝该流。这同打电话时碰到忙音一样

◆ 基于反馈

- ♣ 端主机首先没有预留任何容量，而按照它们接收到的反馈来调整其发送。
- ♣ 调整或者是显式的，如拥塞R发送一个“请慢下来”消息到主机
- ♣ 或隐式的：如端主机根据外部可观察的网络行为，如包丢失率，来调整其发送率

两种机制的比较

◆ 基于预留的系统总是意味着

- ♣ 以路由器为中心的资源分配机制：因每个R负责保持跟踪现在有多少容量被预留，并保障在其所做的预留内每个主机是活着的。
- ♣ 如果在作了预留后某主机发送数据快于它曾经所要求的，则该主机的包将是准备丢弃。R也将会拥塞

◆ 基于反馈的系统

- ♣ 可以既是以R为中心也可可是以H为中心的机制。
- ♣ 一般如果反馈是显式的，则，R至少某种程度就被包括在资源分配模式里；
- ♣ 如果反馈是隐式的，则几乎所有的重担都落在端主机上，当R变成拥塞后就默默地丢包。

基于窗口与基于速率

- ◆ 不论流控还是拥控，对发送者，都需要一个方法来表达：多少数据要传输
 - ♣ 用窗口大小来描述：如TCP：流控也可用窗口通告机制来预留缓冲空间，来支持资源分配，X.25基本上是这样做的
 - ♣ 用速率大小来描述：用速率控制发送者的行为，目前可假设有这样的协议支持视频：如接收方可说它可处理1Mbps的视频，而发送方则坚持这一速率

资源分配小结

- ◆ 3个分类，每类有2种，故共有8种不同策略，而实际上看来**只有2个是普通流行的**，并与网络的现行服务模式连在一起
- ◆ Best-effort服务模式意味着
 - ♣ 是**反馈**机制，
 - ♣ 最终由**端用户**来解决拥塞，
 - ♣ 实际上这样的网络使用基于**窗口**的信息
- ◆ QoS服务模式意味着
 - ♣ 资源**预留**，
 - ♣ 需要包含**路由器**，如排队包的不同取决于它所需预留资源的级别，
 - ♣ 这时用**速率**来表示这种预留是很自然的，因为窗口是针对某带宽而言的

2.4 资源分配评价标准

◆ 怎样评价资源分配机制的好坏

- ♣ 有效性
- ♣ 公平性

◆ 资源分配的**有效性**

♣ 2个主要测量指标:

- ☞ 网络吞吐量
- ☞ 延迟

- ♣ 有1方法：增加吞吐率，提高链路利用率，结果增加了R内队列的长度，进而包的平均延迟也变长了

网络能力

◆ 用吞吐率和延迟之比来描述资源分配模式的有效性，该比称为网络能力

♣ Power = 吞吐率/延迟 （实际定义为 $\text{Power} = (\text{吞吐率})^\alpha / \text{延迟}$ ）： $0 < \alpha < 1$ ， $\alpha = 1$ 表示最大化在延迟曲线的拐点，吞吐率是 Bit/s 延迟单位是 s

♣ 该判据对资源分配有效性并不十分明显，但是可接受的

☞ 因为，M/M/1 排队论假设队列是无限的，而实际网络是有限缓冲区并有时丢包

☞ 只定义了单个连接流，对多个连接、竞争未考虑

♣ Fig. 4.3

目标是最大化吞吐/延迟比

- ◆ 网络上有多少负载，这将由**资源分配机制**来设置。
- ◆ 理想情况下应在该**曲线峰顶**上运行。
- ◆ 峰值左边**太保守**，允许包太少、链路利用率低
- ◆ 峰值右边太**过分**，许多包容许到网络，队列的延迟增加超过了吞吐率大增加
- ◆ 一般实行粗略控制

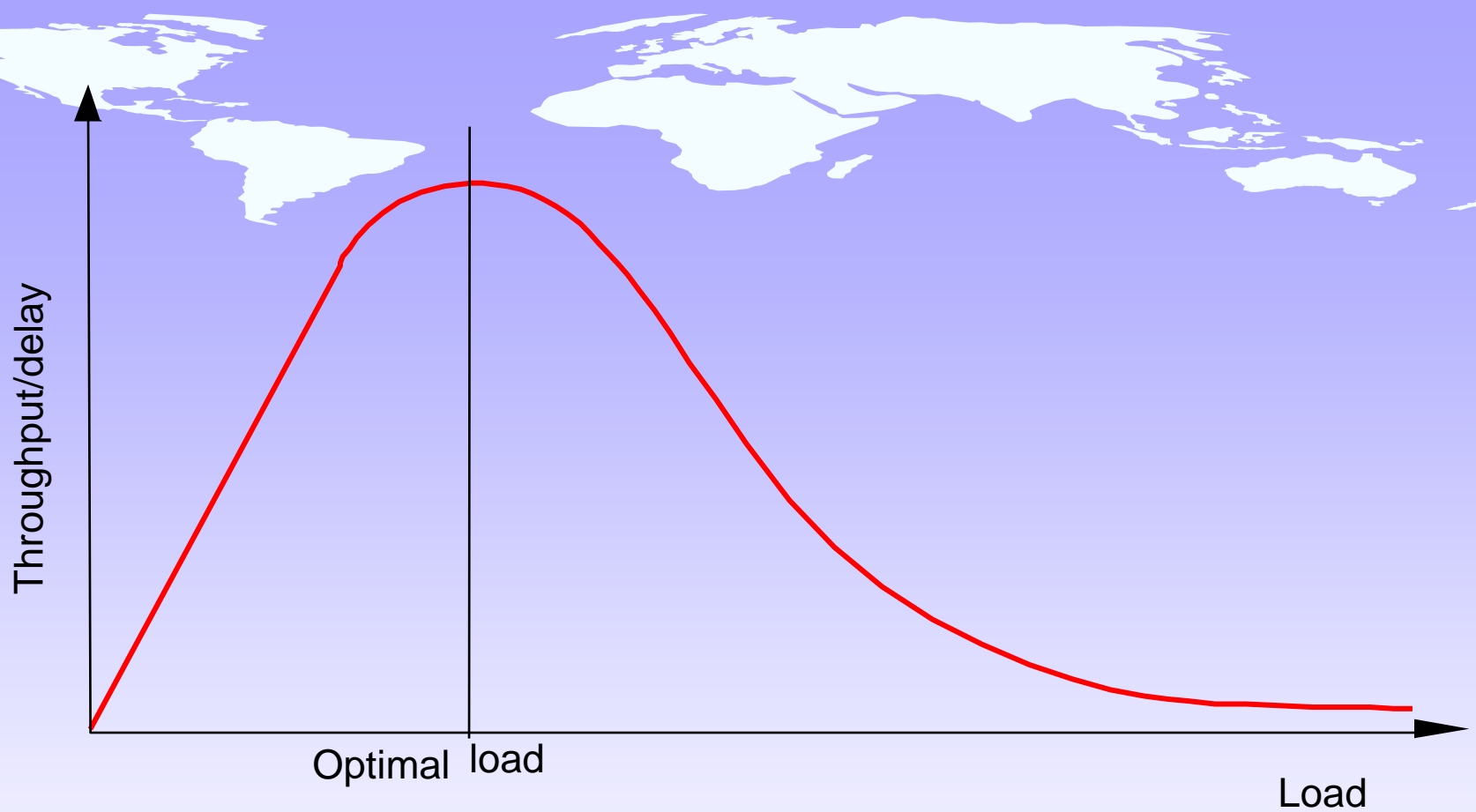
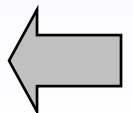


Fig. 2.3 能力曲线—负载与吞吐/延迟比

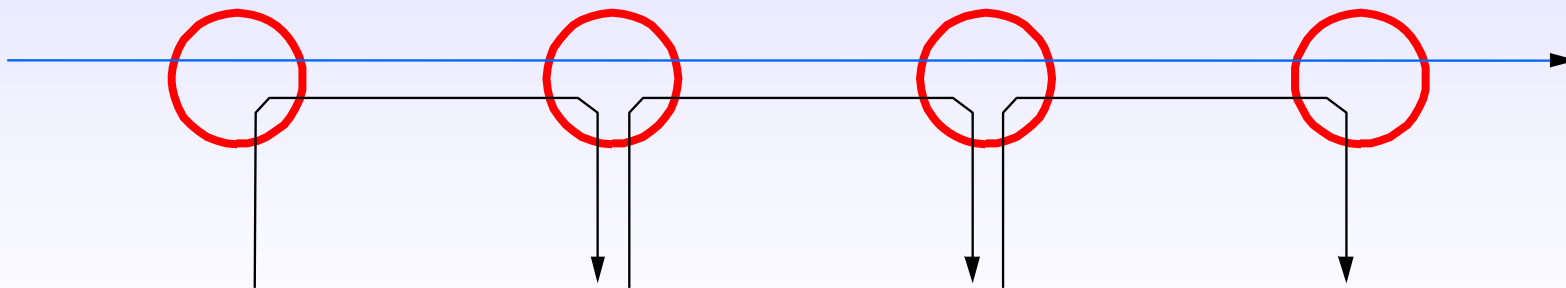


2.5 资源分配的公平性

- ◆ 网络资源分配还要考虑公平性
- ◆ 当试图准确定义公平性时，又陷入了困难
 - ♣ 基于预约的资源分配机制显然导致了可控制的不公平
 - ♣ 该机制可能通过预约使接收视频流在1Mbps，而相同链路中的FTP却只有10Kbps
- ◆ 一般公平原则（若没有明确要求）
 - ♣ 当某条特别链路中存在几个流时，对接收每个流都应该有相等的带宽

公平的定义

- ◆ 假设**公平分享带宽就是分享相等带宽**
 - ♣ 即使不是预约的情况下，平均分配也不等于公平分配，
 - ♣ 是否要考虑流的路径长度？如下图，**1个4跳流同3个1跳流**竞争时，公平性是什么？



Raj Jain 公平指数

- ◆ 假设公平性是指带宽相等且所有通道长度相等，给定一组流的吞吐量 (x_1, x_2, \dots, x_n) [用致单位如bps测量] 下面函数赋予0—1的公平性指数

$$f(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

- ◆ 当n个流都获得1个单位吞吐量时，公平指数

$$f(x_1, x_2, \dots, x_n) = \frac{n^2}{n \times n} = 1$$

- ◆ 当1个流获得1+的吞吐量时，公平指数是

$$f(x_1, x_2, \dots, x_n) = \frac{((n-1) + 1 + \Delta)^2}{n(n-1 + (1 + \Delta)^2)} = \frac{n^2 + 2n\Delta + \Delta^2}{n^2 + 2n\Delta + n\Delta^2} < 1$$

- ◆ N个流中只有k个接收相等等吞吐量，其余为0，公平指数

$$f(x_1, x_2, \dots, x_n) = \frac{k}{n}$$

2.6 排队规则与流量整形

- ◆ 不管其它资源分配机制是多么简单或复杂，每个路由器都必须实现某些排队或调度规则，以便管理等待发送包的缓冲
- ◆ 排队算法可认作是带宽（包得到发送）和缓冲空间的分配（包被丢弃），它还直接影响包经历的延迟（等待多长后发送）
- ◆ 3个排队算法：FIFO和公平排队

2.6.1 FIFO

◆ First In First Out:

- ♣ 首先到达的包首先发送
- ♣ 当R的缓冲空间（本例8个包）满时，尾部的包就丢弃（tail drop），不考虑是否重要等
- ♣ FIFO和tail drop是不同的概念，前者是发送调度策略，后者是丢弃策略，但二者常捆绑在一起叫做FIFO
- ♣ Fig. 2.4

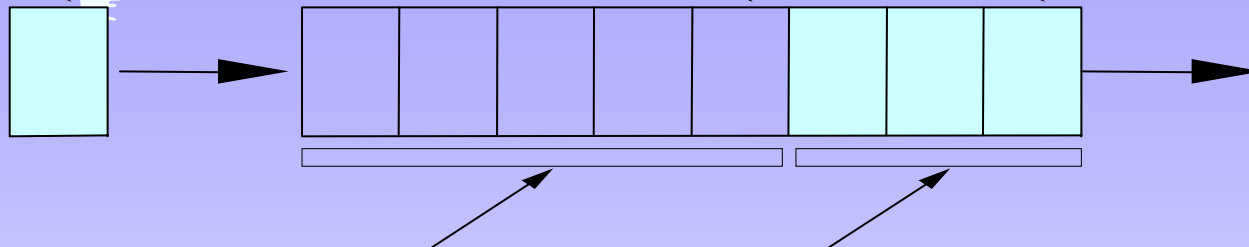
2.6.2 优先排队

- ◆ 路由器简单地把所有拥塞控制和资源分配的责任都推到网络的边沿
- ◆ 故当前因特网流行的拥塞控制方式认为路由器没有取多大作用，TCP担负了所有的责任
- ◆ 对基本FIFO的简单变化是优先排队策略：
 - ♣ 给每个包打上可携带的优先权标记，如IP服务类型TOS（DiffServ.）。
 - ♣ R先发送其优先权高的包
 - ♣ 图2.5
 - ♣ 这一方式离Best-effort delivery并不远

Arriving packet

Next free buffer

Next to transmit



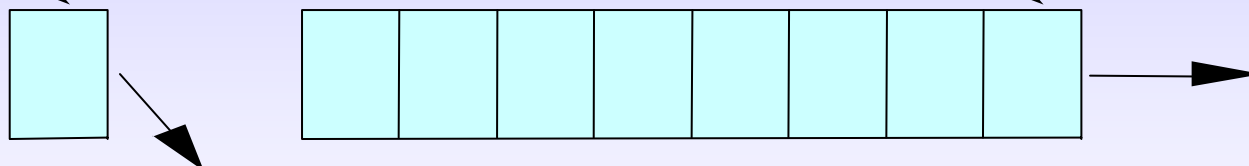
(a)

Free buffers

Queued packets

Arriving packet

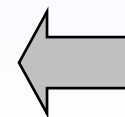
Next to transmit



(b)

Drop

Fig. 2.4 FIFO排队及其截尾



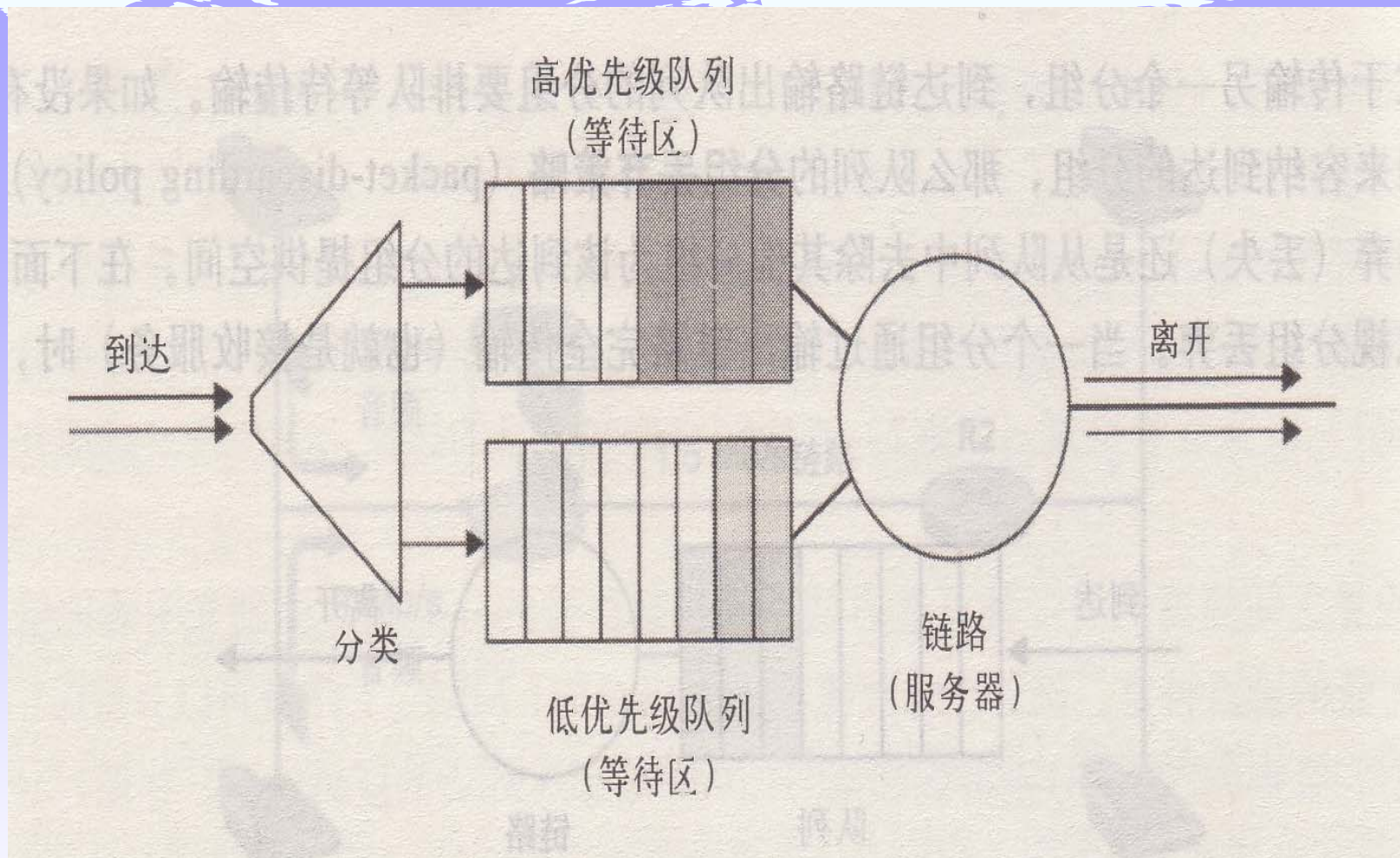


Fig 2.5 优先级排队模型

优先排队策略的问题

- ◆ 高优先权队列可能使其它队列长期等待（发送饿死），即只要至少有一个包在高优先权队列里，较低优先权队列就总得不到服务
- ◆ 对插在队列中的高优先权流要有严格的限制
- ◆ 显然我们不能容许用户自己用不可控制的方式来设置高优先权
- ◆ 既要防止全部由用户做，又要提供给用户某些pushback的格式

优先排队策略的应用

- ◆ 一个显然的方式是运用经济杠杆，包的有限权越高，则付费越高。
- ◆ 然而，富有挑战的是，在因特网这样的分布式环境下如何实现
- ◆ 优先权排队常用在因特网中以保护重要包的传递，
 - ♣ 如RIP/OSPF/BGP等路由更新包，以保障在网络拓扑改变后路由表的稳定。
 - ♣ 特别队列TOS—DiffSer.

2.6.3 公平排队

◆FIFO主要问题

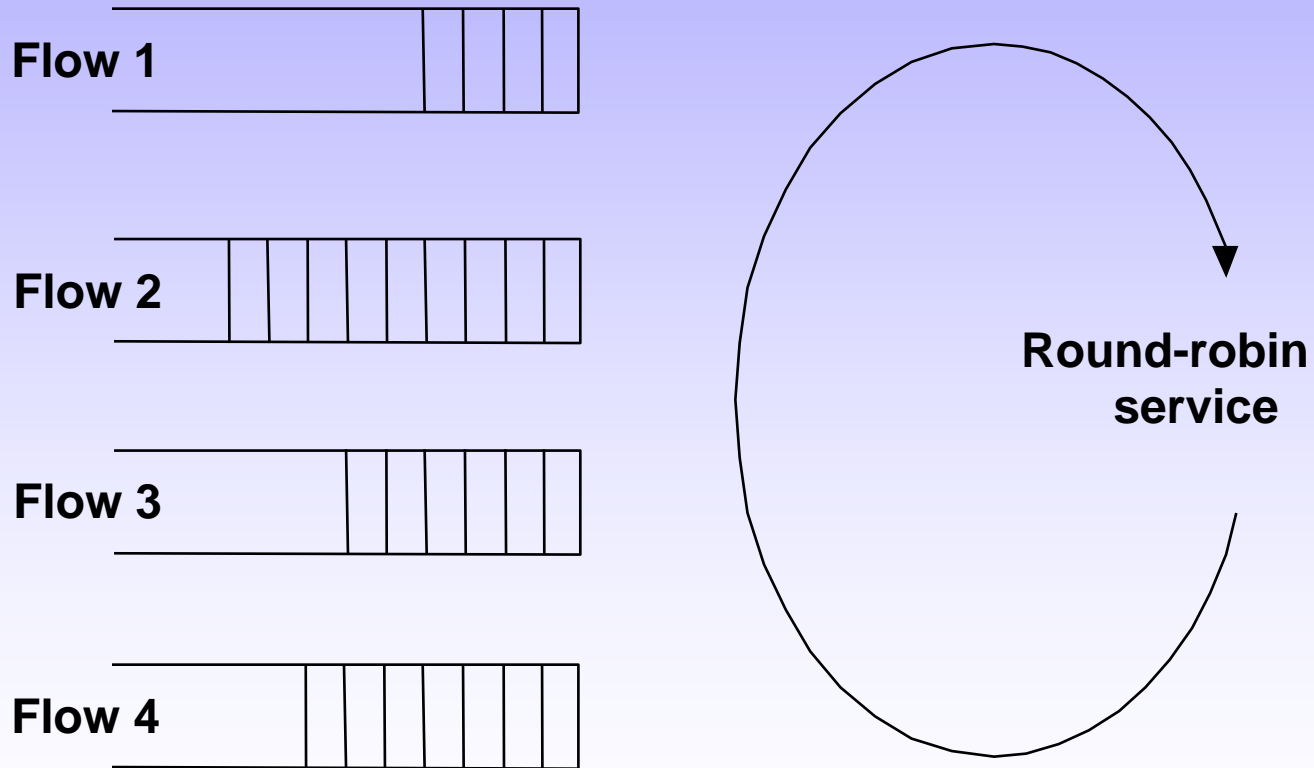
- ♣不能区别不同的流源，即不能按他们所属的流来区分各包。问题在2个不同的层次
 - ☞来自R的少量帮助对任何全部在源端实现的拥塞控制算法都是不适宜的
 - ☞全部拥塞控制机制都在源端实现，FIFO不提供手段引导怎样的资源来坚持这种机制
- ♣非TCP应用，可能把端-端拥塞机制旁路掉
 - ☞如IP电话，这类应用能把自己的包洪泛到网上R，引起丢弃其他应用的包
 - ☞大多数基于UDP对多媒体应用

公平排队算法FQ

◆FQ主要思想:

- ♣ 为每个正在被路由器处理的流分别维护一个队列，路由器以轮循方式服务每个队列。
- ♣ 当一个流发送包太快，则其队列填满；当一个队列到达一个特定的长度，属于该队列的后继包就被丢掉，这样，任何源就不可能多占用其他流的网络容量
- ♣ FQ并不告诉源端任何有关R的状态，或并不限制源端发送怎样快

路由器中的公平排队



FQ机制

- ◆ 主要的复杂性是路由器中处理的**包长度不相同**。
公平须考虑包的长度
 - ♣ 如路由器管理2个流，一个是1000Byte/包，另一个是500Byte/包。
 - ♣ 对每个流队列的简单轮循服务将给第一个流以2/3链路带宽，第二个流仅给1/3的带宽
- ◆ 显然R从不同的包来交叉比特Bit-by-bit轮循是不可行的。
 - ♣ 模拟FQ比特轮循机制
 - ♣ 先判断如果该包按位轮方式发送, 何时可发完; 然后根据完成时间对要发送到包排序

◆ 考虑单个流行为, 并假设有个时钟, **每当所有活动流都传了1位时, 时钟滴答一次**

♣ P_i 是包 i 的长度, S_i 是 R 开始发送包 i 的时间, F_i 是 R 传输包 i 的结束时间

♣ 故有 **$F_i = S_i + P_i$**

♣ 何时开始传输第 i 个包?

☞ 取决于 R 何时完成第 $i-1$ 包的最后1比特

☞ 或第 $i-1$ 包的最后1比特传输完成后很长时间, 第 i 个包才到达。这时轮转算法不从该流发送包

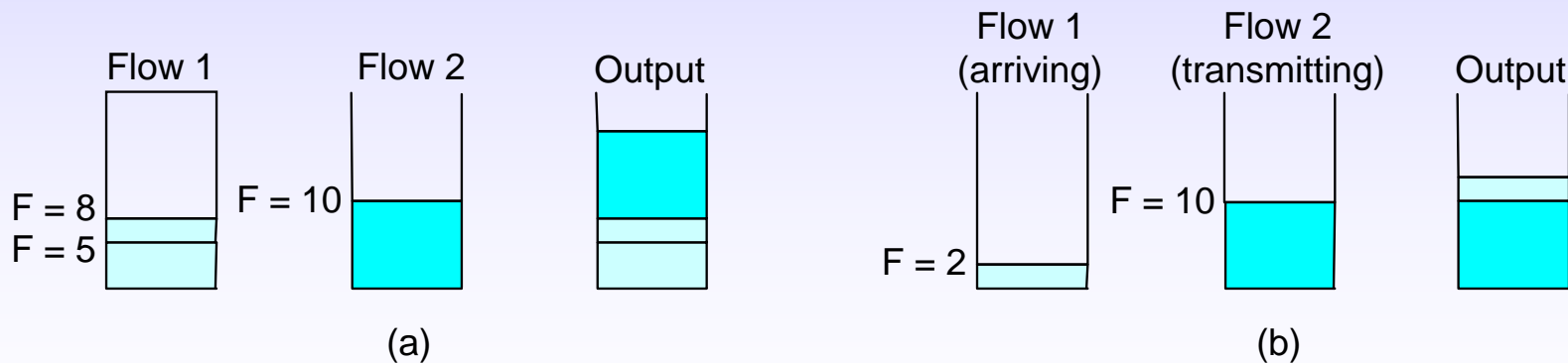
☞ 令 A_i 表示分组 i 到达路由器的时间。则 **$S_i = \max(F_{i-1}, A_i)$**

☞ 故 **$F_i = \max(F_{i-1}, A_i) + P_i$**

◆ 对 n 个活动流, 当 n 个比特传输后时钟就计1次。
并依此来计算 **A_i**

2个流公平排队之例

- ◆图(a)：先选流1后选流2：短包优先
- ◆图(b)：正在发送流2时，流1到达：长包已在发送中，进行中优先。
 - ♣ 尽管发送中可能流1到达完毕，也不强占流2包



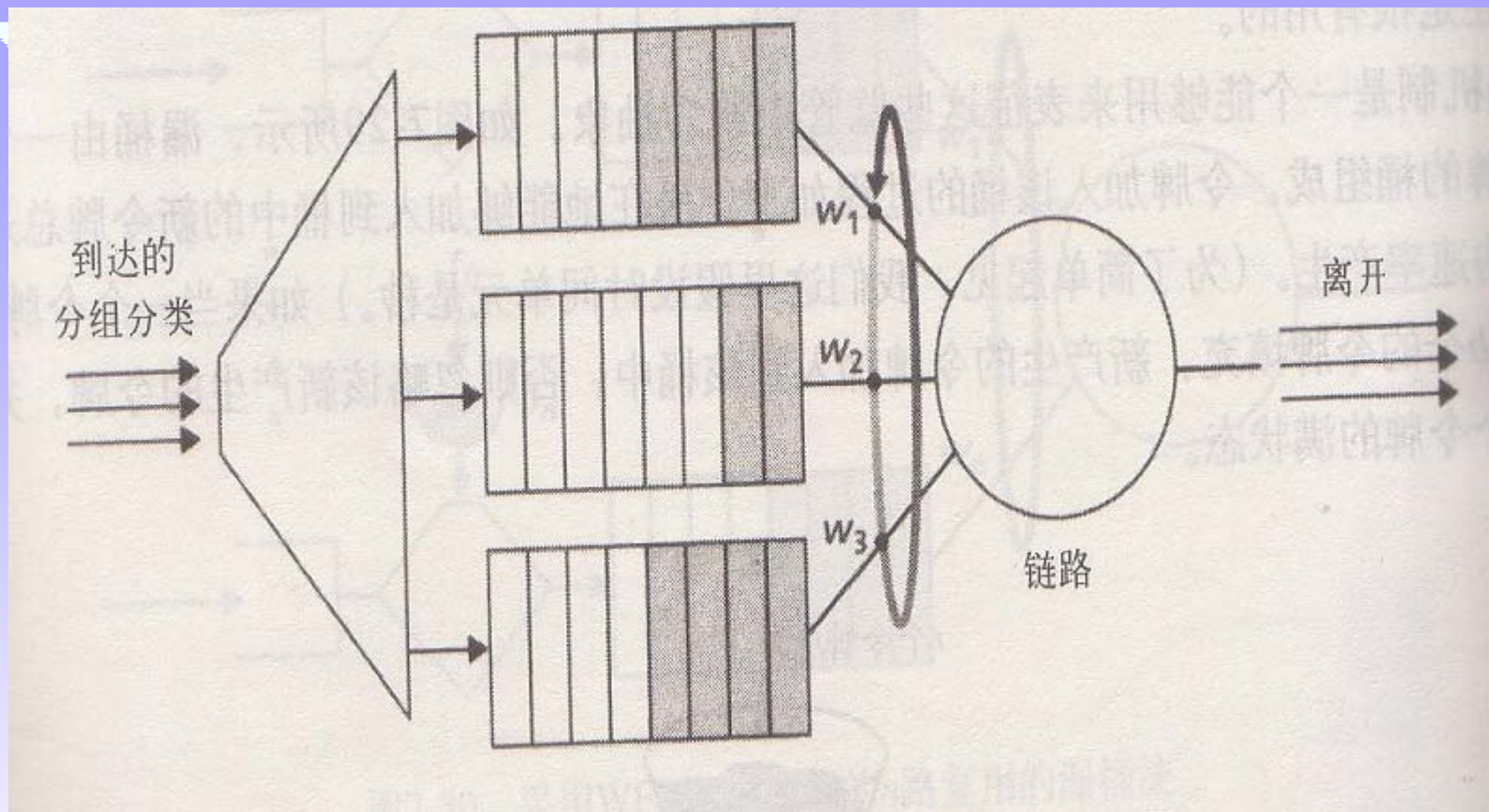
公平排队2点注意

- ◆ 只要至少一个包留在队列中，就决不让链路空闲：具备这一特点的排队称为工作守恒work-conserving。其效果是：
 - ♣ 如果和那些不发送任何数据的流分享链路，我可以
使用链路的全部容量
 - ♣ 一旦那些流开始发送数据，将用它们应分享的容
量，我的流量将削减
- ◆ 若链路满负载且有 n 个流在发送数据，则我不能用大于 $1/n$ 的链路带宽。若硬要做，包的时戳增加，故会在队列中久等，最终队列溢出
- ◆ 故FQ算法也应该有丢包的策略

2.6.4 加权公平排队WFQ

◆ 加权公平队列 (Weighted Fair Queueing)

- ♣ 给排队流加权，权逻辑上说明R服务每个队列时每次发送多少比特，它直接控制每个流将得到多少链路带宽
- ♣ 简单FQ给每个队列权重是1，即每轮每个队列逻辑上仅1比特被传输 $\rightarrow 1/n$
- ♣ WFQ: 可让3个队列分别权重2:1:3 \rightarrow 则导致带宽分别是 $2/6:1/6:3/6$
- ♣ WFQ中的流在使用中可能是“流量类型”，由TOS或DiffSer. 来定义



- 第 i 个类的权 = w_i
- 获得部分服务 = $\frac{w_i}{\sum w_j}$
- 获得到吞吐量 = $R \frac{w_i}{\sum w_j}$

加权公平排队

2.6.7 流量整形 (Traffic Shaping)

◆开环控制

◆基本思想

- ♣造成拥塞的主要原因是网络流量通常是突发性的
- ♣**强迫包**以一种**可预测的速率发送**;
- ♣在ATM网中广泛使用。

◆网络流监管的**三准则**

- ♣平均速率:100bpp比6000bpp的源约束要严格多,希望限制某个流的长期平均速率
- ♣峰值速率:网络可允许平均速率6000bpp,希望限制其峰值速率 $<1500\text{bpp}$
- ♣突发长度:希望限制极短时间间隔内所能发送到网络的最大包数



◆漏桶算法（The Leaky Bucket Algorithm）

- ♣将用户发出的不平滑的数据包流转变成网络中平滑的数据包流；
- ♣可用于固定包长的协议，如ATM；也可用于可变包长的协议，如IP，使用字节计数；
- ♣无论负载突发性如何，漏桶算法强迫输出按平均速率进行，不灵活，**溢出时要丢包**。

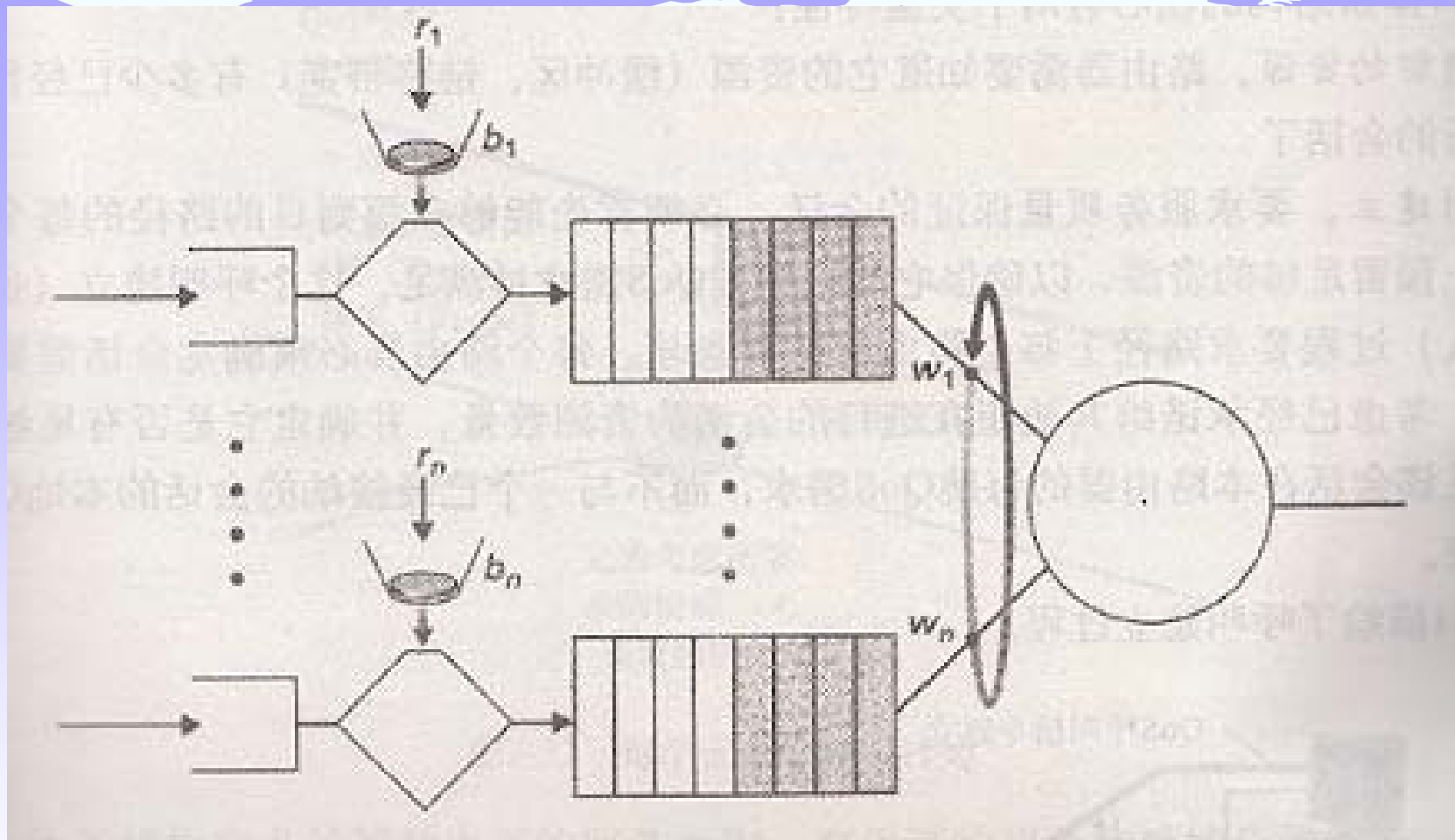
2.6.8 令牌桶算法 (The Token Bucket Algorithm)

◆ 基本思想

- ♣ 漏桶算法不够灵活，因此加入令牌机制；
- ♣ 基本思想：漏桶存放令牌，每 ΔT 秒产生一个令牌，令牌累积到超过漏桶上界时就不再增加。包传输之前必须获得一个令牌，传输之后删除该令牌；

◆ 漏桶算法与令牌桶算法的区别

- ♣ 流量整形策略不同：漏桶算法不允许空闲主机**积累发送权**，以便以后发送大的突发数据；令牌桶算法允许，最大为桶的大小。
- ♣ 漏桶中存放的是**数据包**，桶满了丢弃数据包；令牌桶中存放的是**令牌**，桶满了丢弃令牌，**不丢弃数据包**。



- 采用WFQ的n路复用的漏桶流控
- 可在Diffserv中使用

Thank you!

