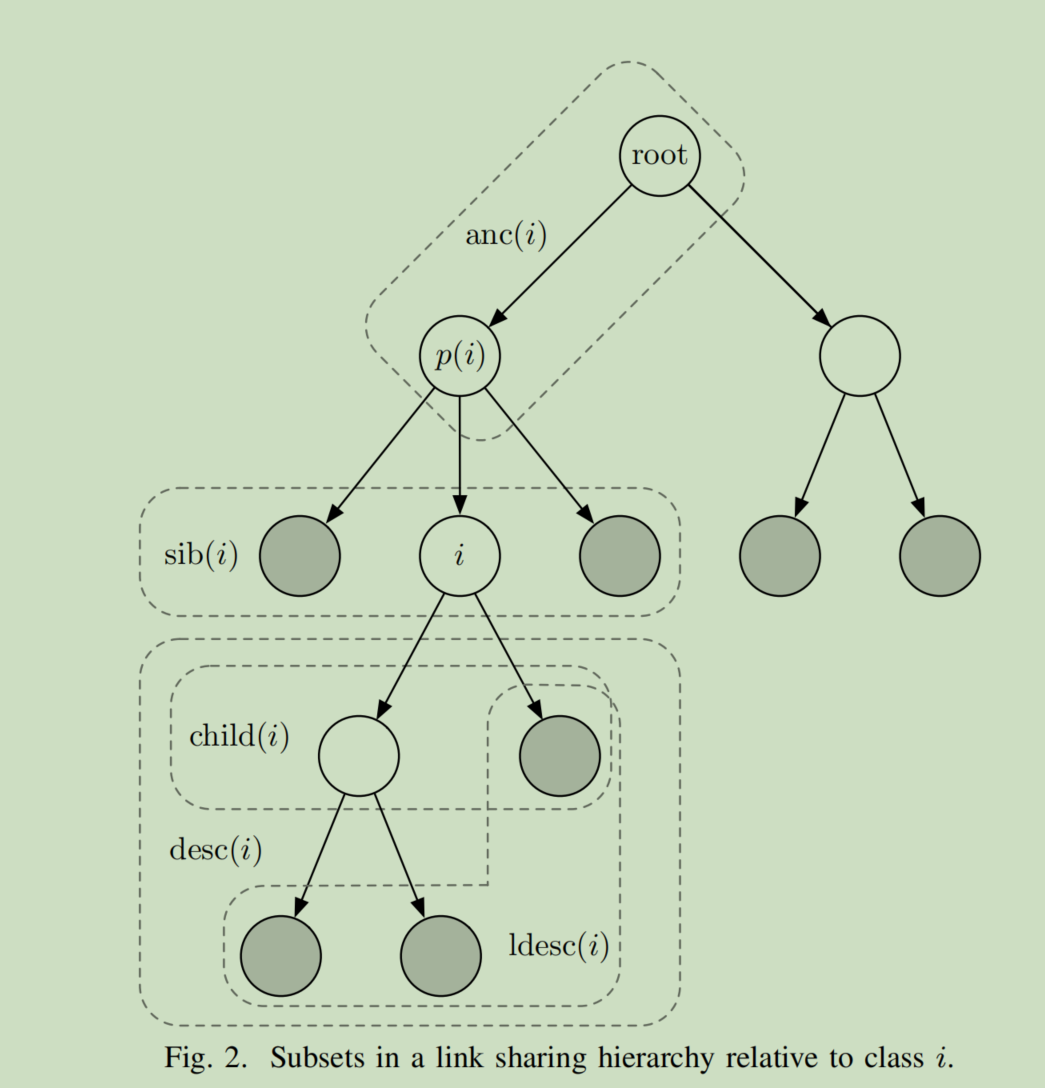
# Tree介绍

i的祖先节点anc(i)包含p(i) 和root

i的子孙节点包含4个节点，即child和child的child，且递归。



# Link-Sharing and Resource Management Models for Packet Networks

## Abstract

本文讨论链路共享需求和一种潜在的链路共享机制。如果多种客户，多种协议族，或者多种优先级共享一个网络，路由器必须提供一种共享机制，控制网络。及时业务是一种紧急业务，需要路由器在共享网络上保留一定的资源，而不是要求实现一种独立的网络。

## Introduction

网络中已经出现几种紧急需求，包括共享服务和及时服务。在一个带宽有限的网络中，像audio和video需要严格控制延迟。本文讨论链路共享需求，实现算法，可扩张的架构，以适应网络的多变性，高效性，健壮性。

多个客户，每个客户因为费用的不同而引起带宽不一样。但是每个用户的费用都要保证一定的带宽，即网络需要为每个客户保留一定的带宽。但是，当只有唯一的一个客户在线时，需要可以使用网络的最大带宽。其它的共享，例如不同的协议族，不同的协议族承载不同的网络内容，对网络的要求不一。同一个协议族下，还有TC的需求，例如telent/audio/video。

这些不同的需求导致层次调度的产生（Hierarchical link-sharing），例如多个客户的多个不同业务共享一个网络。每个业务可能需要不同的带宽，每个业务可能分时在线等。这种网络现实导致了单个链接汇聚成Hierarchical link-sharing structure。每个节点（class）是一个拥塞控制节点。不同的链接有不同的共享框架，在某个class共享的两个链接，在下一个节点可能分成两个class。

同时在线的共享服务和及时服务，需要一系列的规则，才能获得相对满意的服务。及时服务需要保证一定的吞吐和一定的延迟承诺，即需要保证一个精确的调度。对于长时间业务（大象流）却可以维护一个不太准确的调度。本文解决了及时服务和通用服务之间的关系。不是为两类服务分别设计机制，而是一套机制可以服务两种业务。

假设一个网络中有两种服务，及时服务和非及时服务。在没有拥塞的状态下，调度器可以采用通用的策略，例如fifo，sp或者rr。但是，在拥塞情况下，即一种服务用了过多的带宽（可能是及时服务使用了过多的带宽造成非及时服务饿死，或者非及时服务使用过多的带宽，造成及时服务服务质量下降），此时，link-sharing机制需要发现这种问题，并通过一定的方式解决这个问题。此时调度器可能对overlimit 队列采用额外的调度策略。因此，及时服务可以采用SP调度。即有机制保证其它业务不会饿死。

## The Link-Sharing Goals

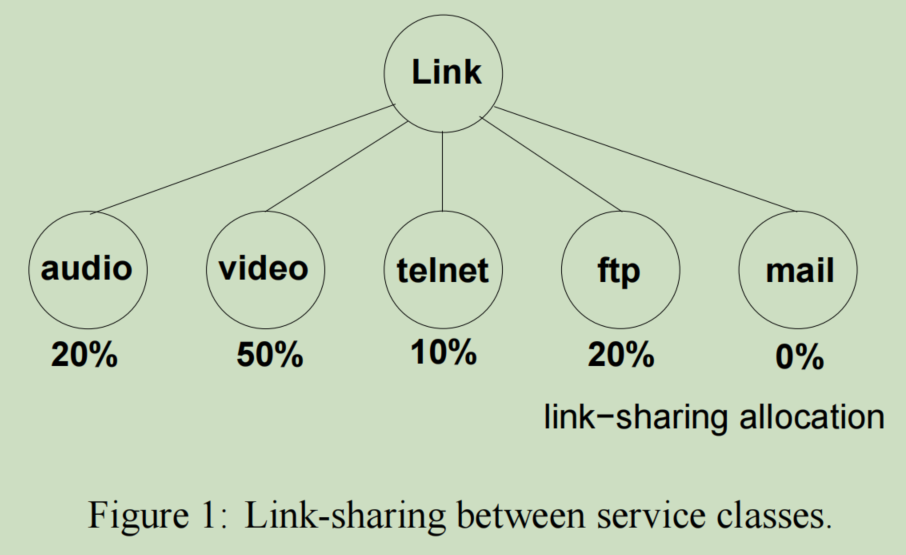
多个客户，其中每个客户可能使用不同的协议族，协议族中可能有不同等级的业务需要在一个网络中共享带宽。客户之间可能共享的是付费网络，即有承诺带宽需求。不同的业务可能对拥塞的敏感程度不一，共享机制需要实现“在拥塞时，对拥塞敏感低的业务不能占用所有带宽”。对于实时业务和非实时业务，如果采用SP调度，需要做到，“在一定的时间周期内，不能造成非实时业务饿死”。

Link-sharing中有各种不同的需求，需求来自不同的客户，不同的协议族，不同的业务。我们不是分立的实现这种需求，而是采用一种方案来满足不同的需求，这种方案具有一定的灵活性。

首先，我们考虑flat link-sharing，图1中，有多种实时和非实时业务，每一个叶子节点就是一个classes。汇聚的节点（aggregated link）就是内部节点（interior class）。在拥塞下，每个叶子节点期望获得的流量(AR，assured rate）。图一中，telnet class可能是一种延迟敏感服务。但是mail确实一种延迟非敏感服务。

图1中，root的总带宽按照比例分配个不同的业务。共享链路的第一目标是：在一定的时间间隔内，如果产生拥塞，保证每个叶子节点获得承诺带宽（allocated bandwidth）。即在拥塞时，某些class将严格限制到承诺带宽。例如，没有承诺带宽的mail，需要其它机制保证服务，在拥塞时，不会保证mail的任何流量。

链路共享是一种粗糙的网络流量量化。是一定时间间隔内的测量。在图一中，如果telnet和ftp的延迟高达分钟，这认为是不可接受的。不管如何，一个非常小的时间间隔内，需要有调度策略保证每一个class都要能获得调度，而不是在任意小的间隔内，都要满足分配带宽（allocated bandwidth）



### Definitions：exempt classes

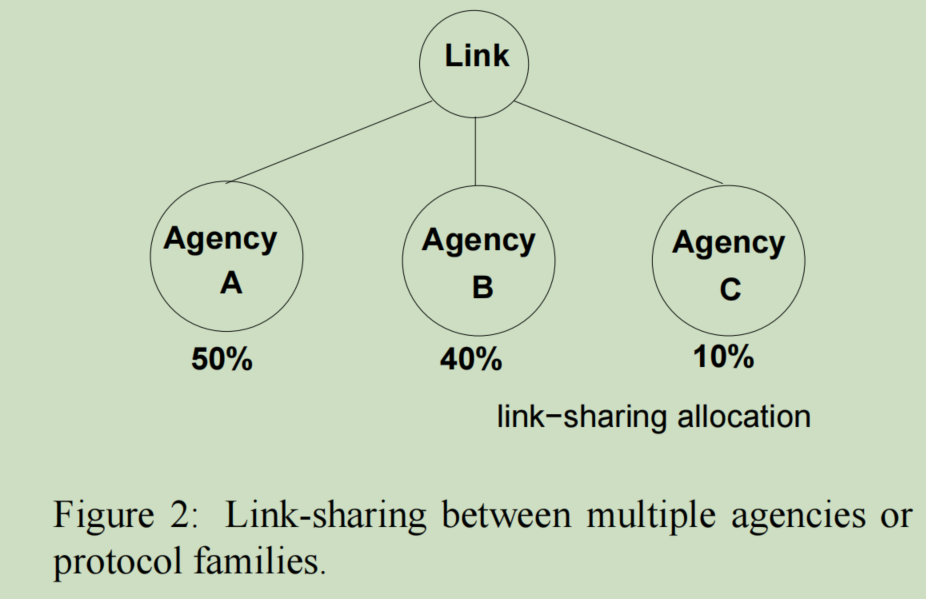
例外队列，任何时候，都需要直接调度？SP。此时需要管理员，配置SP的总带宽不能超过Root带宽？

### Definitions：restricted classes

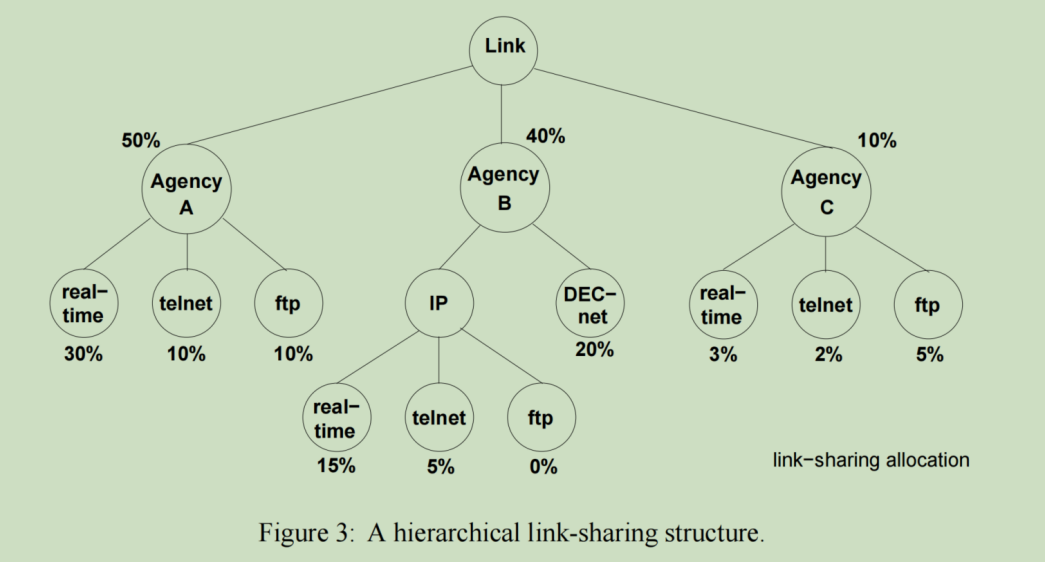
任何时候，该队列的带宽都是保留的，无论是否在线，其它人都不允许抢这个带宽。

共享链路的第二个目标是：当某些队列没有达到分配带宽时（allocated bandwidth），其它队列不能任意分配剩余带宽，需要遵循某些分配策略。对于图1，flat架构不能决定如何分配共享带宽，如何分配剩余带宽，是一种调度策略。利于图一可以采用相对优先级分配剩余带宽。

在图2中，例如Agency A只有很少的流量，Agency B可能认为不公平，假如所有的剩余带宽都分配给了Agency C。一种比较理想的测量是，按照active队列的权重重新分配剩下带宽。例如当Agency A不用时，此时B：C的带宽比例还是4：1。即80%：20%。



不同的link-sharing 约束可以通过Hierarchical link-sharing架构来实现。所有的报文分配一个叶子节点（leaf），中间节点（interior class）设计成如何分配剩余带宽。



### Link-Sharing goals

1. 在一定的时间间隔内，叶子节点或者内部节点都应该大致获得分配带宽。
2. 当每个叶子节点和内部节点都获得了分配带宽时，不能任意分配剩余带宽，需要遵循某种原则在激活队列中按照某种有限策略分配。

### Definitions

General Scheduler：通用调度器，非拥塞场景下，调度。

Link-sharing Scheduler：某些队列超过了一定的吞吐率或者低于一定的吞吐率，采用的调度策略。

Regulated classes：采取了link-sharing Scheduler调度策略的队列，即要么超发，要么欠服务且有排队。

本文并没有解决如何分配剩余带宽的问题。但是通用调度器general Scheduler考虑了这种策略。根据优先级，采用相对比例分配剩余带宽（一种理想的分配方式）

本文的link-sharing目标相对简单，采用较少的机制就能保证每个链接的最小带宽需求。

一个采用最少行动的例子。一个网络可能将80%的带宽分配给实时业务，20%的带宽分配给非实时业务。采用WRR或者类似调度，并附加管理员控制策略，可能控制实时业务到总体带宽的50%。Link-sharing Scheduler可能只是在实时业务超过了80%,且非实时业务有排队时，才介入。但是，采用了link-sharing Scheduler策略后，能保证一定的时间，不饿死非实时业务。

Link-sharing structure可能有静态和动态两种，管理员已经分配了静态的层次关系，并分配了流量。

Link-sharing的目标不是解决网络拥塞，只是解决了不同链接之间的流量分配问题。假如一个leaf是一个链接，作为中间节点（router），需要解决这个链接需要多少带宽和buffer的问题。如果leaf是aggregated的class，则需要端到端的拥塞控制协议，例如TCP。

我理解：link-sharing Scheduler不解决网络拥塞，只是按照规则为每个链接提供尽力而为的服务，拥塞是L4，即点到点的问题。如果router有拥塞，可以反馈拥塞情况，即队列情况到端点。

## Formal link-sharing guidelines

假设每个调度器（sched）有两种调度，一种是general Scheduler，完成非拥塞情况下的调度。一种是link-sharing Scheduler，对某些队列附加调度规则（regulated）的调度。

我理解两种调度是：一种就是正常情况的调度，按照图1，假设每个队列都有任务，就是按照AR，即保证带宽调度。但是，任何时候，不是每个队列都有任务，这样子就涉及到剩余带宽分配，没有任务的队列突然有任务，低优先级队列长时间没有服务时，就涉及到link-sharing Scheduler，即改变某些队列的状态。将overlimit的队列限流，为低优先级长时间没有服务的队列分配带宽，将突然有任务的队列排队等，将他拉回到图1的按照某种比例分配带宽的模式上。

本文给出一种草案建议，满足实时任务和共享任务同时存在的带宽分配问题。本文讨论general Scheduler和link-sharing Scheduler的关系。但是不涉及Scheduler具体实现形式。

Link-sharing Scheduler可能对一个队列采用多种规则中的某一个。即当有剩余带宽时，如何分配剩余带宽，如果有某些其它队列处于服务不满意状态时，如何将超发的流量收回等。以及丢包策略等，或者降低队列优先级，或者动态调整每个队列的分配带宽等。

Link-sharing Scheduler和general Scheduler不是独立的两段代码，而是一个整体。例如，我们的示例中，就是在general Scheduler额外增加了time-to-send域。

### Definitions

Overlimit:超过分配带宽

Underlimit：低于分配带宽

At-limit：刚好等于分配带宽

Satisfied：非unstatisfied

Unsatisfied：underlimit & persistent backlog（注意不是低于分配带宽就是服务不满意，是低于分配带宽的同时还有排队，如何判断是否有排队也是个问题）

需要在一定的时间间隔内，通过estimator评估每个队列的状态。时间间隔是一个非常重要的参数，这个间隔决定了link-sharing介入的频率。Root只有At-limit和underlimit两种状态。有多重方式实现estimator，目前流行的是bucket token。

|  |
| --- |
| Formal link-sharing guidelines：A class can continue unregulated if one of the flowing conditions hold: |
| 1. The class is not overlimt, or   本队列处于欠吞吐状态，那么本队列必须要获得服务，采用general Scheduler |
| 1. The class has a not-overlimt ancestor with no unsatisfied descendants   如果本队列已经超发，超发即抢了别人的带宽，超发是借用了父节点以及祖先节点的带宽。   1. 如果本队列的祖先节点处于underlimit，则该祖先节点下欠服务的队列有带宽可以发送，即不需要限制本队列带宽 2. 如果本队列的祖先节点处于atlimit或者overlimit，该祖先节点下有欠服务的队列，则需要限制本队列的带宽。   大白话解释：就是某个队列超发，会朝root方向一层一层借用带宽，抢了这个叶子节点到root节点上所有中间节点（包括root节点）的带宽，造成这些中间节点或者下面的其它叶子unsatisfied，就需要限制超发。 |
| Otherwise，The class will be regulated by the link-sharing scheduler. |

Case1中节点处于overlimit，即没有某个队列需要regulated。

Case2中Agency A的realtime 处于overlimit，nonrealtime业务处于underlimit，没有unstatisfied队列，即没有class需要regulated。

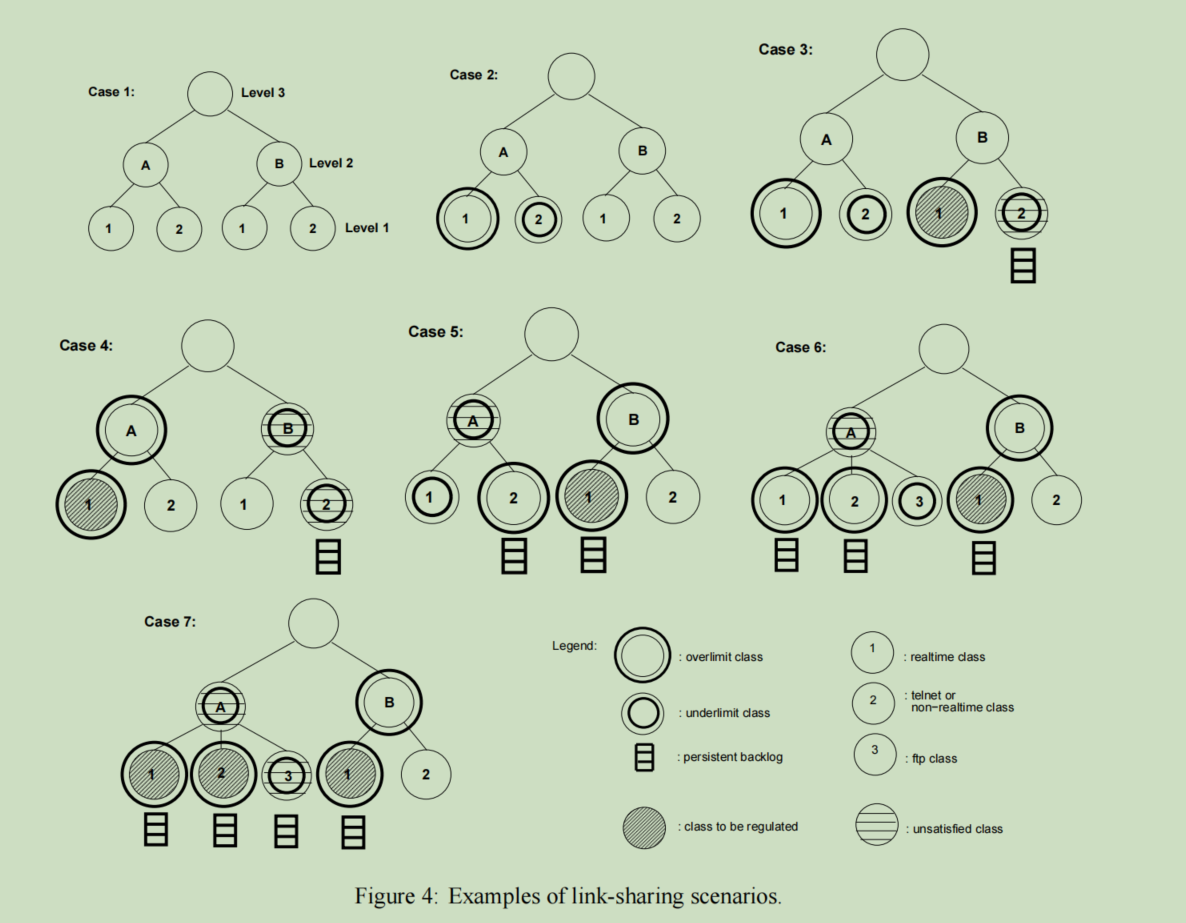
Case3中AgencyA和AgencyB都处于atlimit状态，其中agencyA的1处于超发，2处于欠吞吐。AgencyB的1处于超发，2处于服务不满意。我们可以看到Agency A的1虽然处于超发，但是在层次上，他没有影响Agency B，此处只需要限制Agency B 的1的带宽。

Case4中Agency A和AgencyA.1都处于超发状态，同时Agency B和Agency B.2处于欠吞吐状态，则我们判断Agency A通过Root抢占了Agency B的部分带宽，此处我们应该现在Agency A.1的带宽。

Case5中，Agency A处于服务不满意状态，且AgencyA.2处于超发状态，且有排队，即AgencyA.2队列短时间获得了大量的任务，排队时间过长？Agency B处于超发，此处没有限制AgencyA.1的带宽，限制了Agency B.1的带宽。是合理的。

Case6中，现在Agency B.1的带宽。

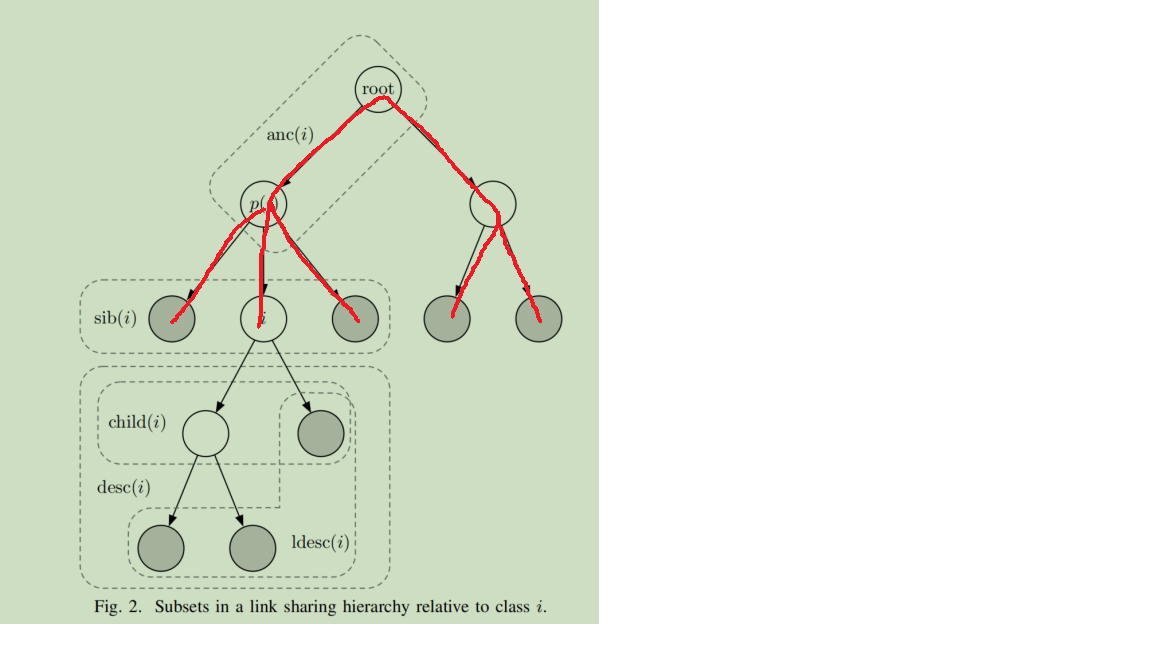
Case7中，现在AgencyA.1/AgencyA.2/AgencyB.1的带宽。



## Approximations to the Formal Link-Sharing guidelines

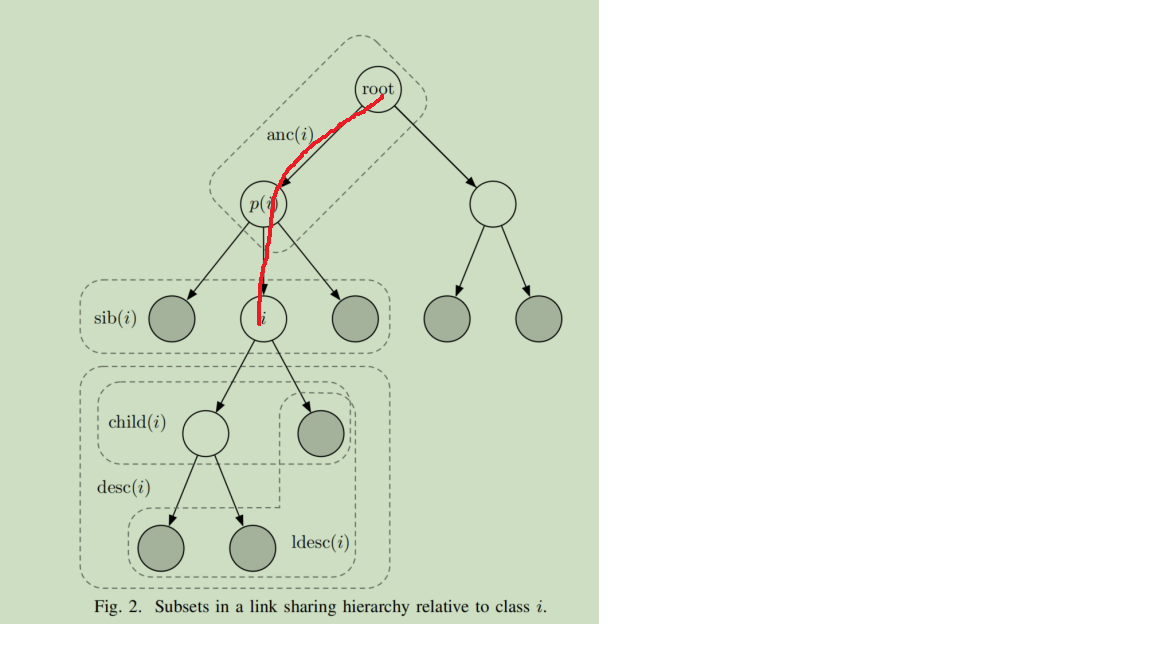
通过以上分析，我们可以知道formal 方式的复杂度，overlimit的影响需要分析所有和本class有路径的class的情况，是一种穷尽搜索。

例如当i超发后，需要判断sib（i）/anc(i)/以及root下的另外一个分支情况。



### Ancestors-only link sharing guidelines

|  |
| --- |
| A Class can continue unregulated if one of the following conditions hold: |
| 1. The Class is not overlimit，OR   本队列没有超发 |
| 1. The Class has an underlimit ancestor   本队列overlimit了，但是祖先处于underlimit状态，且不用管祖先的其它分支节点的信息。 |

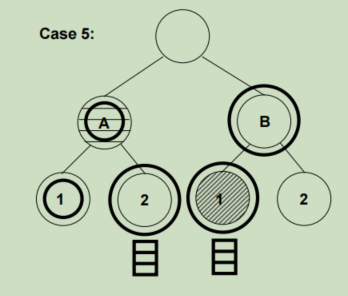


该方式只需要处理上图的每个class的状态。图4的case3说明了Ancestors-only的过程。由于无法处理本class影响到所有的节点，有些class可能被错误regulated或者link-sharing 目标无法满足。

Ancestors-only link-sharing 能满足大部分的场景，但是没有formal健壮。Ancestors-only link-sharing 需要一些机制保证，当存在unsatisfied队列时，阻止较高优先级的队列仍然处于unregulated状态。该机制是Ancestor 处于underlimit，注意不是at limit。例如对于flat结构，root只有atlimit和underlimit两种状态。因此estimator需要严格区分atlimit和underlimit状态。即quantitative敏感。

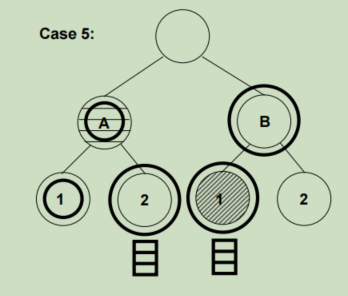
由于Ancestor-only 判断队列是否需要管制，只判断Ancestors是否处于underlimit。如果减少ancestors的某种credits，似的ancestors在子节点超发时，快速处于overlimit状态，则可以限制子节点的超发时间，较快的结束某个队列的超发状态。因为Ancestor-only 没有办法优先处于父节点处于underlimit时，某个不需要borrow的队列拥有较高的优先级发送，造成不需要borrow的队列长时间等待。

Case5中，假设realtime的优先级高于non-realtime，假设未来某段时间AgencyA.1有少量数据需要发送，AgencyB.1处于unstatisfied状态。按照Ancestor-only规则，Root只要处于underlimit，就不会处理AgencyA的状态。这样AgencyB.1会一直处于超发状态。这样造成AgencyA 的网络一直处于一种突发状态。如果Agency A在idle的时候，获得较少的“credits”，Agency A则获得比分配带宽较少的带宽。？



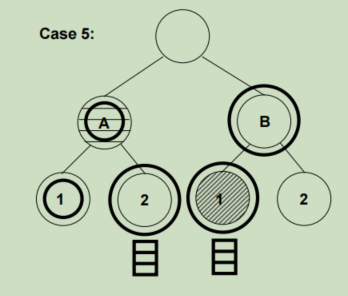
### Top-Level link-sharing guidelines

设置每个叶子节点能够borrow的层级。如Case5，假如我们设置借用的最大Level =2 ，则A和B都不能超发，超发限制到level1，这样子减少了错判的概率。



|  |
| --- |
| A Class can continue unregulated if one of the following conditions hold: |
| 1、The Class is not overlimt, OR  本队列没有超发，即不用任何限制 |
| 2、The Class has an underlimit ancestor whose level is at most Top-Level  本队列处于超发状态，且祖先处于underlimit，注意此处不是一直判断到root，而是可能中间就终止。 |
| Otherwise，the class will be regulated by the link sharing scheduler. |
| The following list gives the set of heuristics used in our simulator for setting the Top-Level variable |
| **Heuristics for setting the Top-Level variable:** |
| 1. if a packet arrives for a not-overlimit class,set Top-Level to 1   如果报文从叶子节点到root的移动过程中，来自一个not-overlimt class,则设置Top-level为1，即无需借用父节点的流量。 |
| 1. If Top-Level is i，and a packet arrives for an overlimt class with an underlimit parent at a lower level than i (say j)，then set Top-Level to j   此处，AgencyA.2 处于超发，AgencyA处于underlimit，设置Top\_level 到2，可以限制AgencyA 进入超发状态，进而影响Agency B的流量？ |
| 1. After a packet is sent from a class,and that class now either has an empty queue,or is unable to continue unregulated, the set Top-Level to **Infinity**   如果队列发送报文后，队列处于IDLE或者不能unregulated，即需要采取规则时，Top-Level变成Infinity？ |

换个方式理解Top-Level，是不是某个父节点下的子节点处于unsatisfied时，就把Top-Level 设置成子节点的层数，这样子子节点就不能借用父节点的流量，慢慢的超发的子节点就消失了。Case5 为例，当某个报文入队时，看到自己的父节点Agency A处于unsatisfied，则将Level改为2，则B就不能超发了？



# Hierarchical Token bucket theory（devik）

## Definitions

每个Class 用于assured rate AR, Ceil Rate CR, priority P, level and quantum Q. 每个Class可能有父节点（parent）或者子节点（child）。实际数量actual Rate 或者R。对于内部节点，R是子孙节点R的之和。

Leaf是没有child的节点，只有leaf队列才能存储packet。

Level表示本class处于的层次。Leaf表示level = 0，root是Level\_count - 1。

Mode表示队列状态。Red：R>CR/Yellow:R<CR & R>AR/Green:R<=AR。

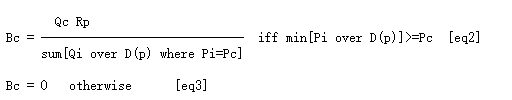
D(c)，C的有任务排队的子孙节点，当C处于yellow时，D(c)表示所有想从C处borrow流量的所有class。

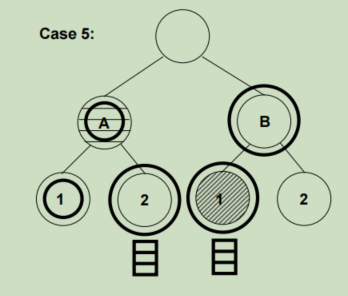
## Link Sharing Goal

公式1表示，class的active rate 不能超过Ceil Rate。当队列处于Assured rate和Ceil Rate之间时，可能从父队列borrow流量。



注意Priority随着调度的进行，可能会改变。此处的意思是剩余带宽具有相同优先级的队列直接共享，且是子队列的最高优先级。此处只是剩余带宽的共享方式。





该处对Quantum有介绍，我理解就是每次调度最多允许超发多少报文。http://intronetworks.cs.luc.edu/1/html/queuing.html。每个队列配置一个quantum即可。注意此处是配置，假设某个父节点下有10个子队列时，某些子队列短时间没有流量，有任务的子队列会不停的参与调度，则剩余带宽分配情况就是有任务队列的quantum(i)的比值。Quantum【0】：quantum【1】等。即解决了剩余带宽如何分配的问题。

### CBQ Note

HTB比CBQ更加严格，是一种CBQ的实现方式。

### HTB Scheduler

根据Top-Level link-Sharing guidelines，每一步只需要将子节点的信息刷新到父节点。只有调度出的那个节点信息需要回刷。即每次调度从叶子到root，然后从root到叶子的过程。我们需要清晰定义队列的class信息即可。

#### Bucket Token

当调度到这个队列时，刷新Token值。Token值的刷新不是任意时刻的，而是按照时间段。维护一个全局的count【47:0】，假设时间精度是1024个cycle，则class里面的time【13:10】，即1024个cycle内，token只更新一次。如果某个队列长期不调度，或者调度时time【13:10】刚好绕转，则这个时间段内，token有抖动。分时间段更新token，可以方便的左移配置速率rate。

例如rate = 100kbps时，软件需要根据硬件主频折算到1024或者2^n个Cycle上，假设F = 200MHz， rateH 1024= 100\*1000\*1024/200M = 0.512 bit/c。量化此值。

例如某个队列在某个1024段调度到，则更新token，如果还是在这个1024段内调度到，则不更新token，更新的时刻在下一个1024时间段。



调度原则：

每一个class内，小于CIR的放如standon链，大于CIR的放入standby链。只有standon链为空后，才能调度standby链。

Backlog判断条件为 standon链非空但是父节点到CBS且连续4次。

#### DWRR

基于报文字节数的加权，每个队列拥有的调度带宽和该队列的权重成比例。实现方法：为每个队列设置一个计数器deficit，deficit的初始值为Weight\*MTU；每次轮询到一个队列时，输出一个报文且队列的计数器减去报文长度，直到计数器减为0时停止该队列的调度；当所有队列的计数器都为0时，根据Weight\*MTU重新初始化每个队列的计数器，开始新的一轮调度。这里疑问？当某个队列一直没有任务时，该队列的计数器一直为初始值，但是另外一个队列有任务且计数器为0？

#### Class

|  |  |  |
| --- | --- | --- |
| 信号 | 位宽 | 描述 |
| CIR | 16 | Committed information Rate，承诺速率，单位是bit/s,即向令牌桶中投放令牌的速率。 |
| CBS | 16 | 承诺突发尺寸，单位是bit，用来定义在流量超过CIR之后，最大的突发流量，即为令牌桶的容量。CBS一般配置为Root的CBS，即允许突发到最大带宽。 |
| Quantum | 16 | 当流量大于CIR之后，每一次调度能发送的字节数。至少大于1MTU |
| Borrow\_en | 1 | 是否借用父节点流量。 |
| Time | 16 | 叶子节点调度的时刻 |
| Priority | 4 | 优先级，支持4个优先级。由DSCP映射。 |
| Parent | 12 | 父节点 |
| Children | 12 | 子节点 |
| Leaf\_state | 2 | 00: can send  01:borrow  02:wait |

#### Sched

此处参考sch\_hbt.c规格。叶子节点使用8K深度的sram，存放叶子节点的class信息。Cos层使用4K深度的sram存放class信息。Vm使用1K深度的sram存放class信息，host的深度为4 class，port为4class，root 为1class。

支持HBT模型，执行统一调度。从Level0开始执行top-level link-sharing调度。即叶子节点也执行top-level link-sharing调度。当存在流控调度时，设置CIR和borrow\_en即可，注意网络拥塞是很少的场景。大部分的时候依靠HBT模型完成每个队列的业务调度，并响应PFC反压。

预估每层缓存320份调度结果，每层红黑树支持4个优先级，每个优先级有一个active链和一个standby链，则每一次中间节点有8queue，每个queue深度为320。即red\_black\_tree。



#### request\_dequeue

|  |  |
| --- | --- |
| Level0 | Queue的class信息按照1K\*8存放，即每读一个sram地址，获取8个queue的class信息。此处为叶子节点，叶子节点按照流量桶模型调度，小于CIR的是active，大于CIR的是borrow，即standby。由于调度时，不知道发包情况，此处采用预扣模式，可以根据优先级配置几个MTU。默认值是固定扣除8个MTU。   1. 扫描叶子Class信息，每一次读取获取8个class 2. 按照＜CIR & Priority+burst\_size调度。 3. 按照 ＜CBS & Priority＋quantum调度。 4. 记录调度的时刻time,leaf\_state，叶子节点的wait状态不能出队。 5. 调度的queue进入vm\_pri\_active & vm\_pri\_standby。 |
| Level1 | 每4个cycle执行红黑树扫描，即8个队列同时扫描。根据vm\_pri信息读取相应的class信息，如果小于Level1 CBS的出队，如果大于Level1 CIR的，需要借用Level2的流量。  A、读取vm\_pri0\_active队列，根据vm\_pri0作为指针读取Level1的class的信息，如果小于CBS则出队，并扣除相关流量信息；如果该class信息已经到达CBS，则设置Heuristics，即禁止属于vmxx\_prio0\_class的子节点之间相互借用，并将该队列放入vm\_prio\_active尾部，等待vmxx\_pri0\_class 的CBS恢复。  B、读取vm\_pri0\_standby队列，根据vm\_pri0指针读取Level1的class信息，如果小于CBS且vm\_pri0\_active为空，则可以出队。如果当前时刻current\_time - leaf\_sched\_time并判断该叶子节点已经小于AR，则放入active链。  按照A/B规范执行其它优先级调度。 |
| Level2 | 按照Level1传递的active和standby进入active和standby链，注意只有standby随着时间流逝会进入active链。  按照AB执行优先级调度。 |
| Level3~5 | 按照上述流程执行调度 |
|  |  |



#### Red-black\_tree

使用Level 1举例，Level0\_pri0队列会将所有VM下的属于pri0的队列入队，例如入队顺序入下列箭头所示，当vm0\_pri0\_q3入队时，vm0\_pri0\_class由于刚调度了一个巨型报文，导致大于CIR，则进入了level1\_pri0\_standby队列。由于Level1\_pri0\_active 非空时，Level1\_pri0\_standby不能出队，且队列是混杂模式，此处会相互影响流量。但是，我们认为CBS流量很容易满足，即满足CBS是一件很容易的事件。即active无事实上的对头阻塞。

Level1\_pri0\_standby由于是borrow型调度，可能存在调度到某个vm\_pri0\_qxx时，刚好满足了CBS不能出队，甚至多次连续撞见CBS，则根据调度time，如果该队列的流量已经小于AR，则进入active队列。



#### Response\_enqueue

当PIPE根据计算出最终报文token时，使用{qid，token}刷新每一层的流量桶信息。从Level0更新到root。

#### PB调度

PB根据自身规则出队，PB事实上受HTB优先级和流量控制，但是DMA PLD的完成时间可能不受HTB调度顺序控制，即允许PB按照先完成先出队，事实上往MAC口调度报文。