

# 流量控制指南

版本 1.0.2

Martin A. Brown

原文地址: [http://www.tldp.org/HOWTO/html\\_single/Traffic-Control-HOWTO/](http://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/)

翻译 : ziven (<http://blog.csdn.net/zgangz>)

水平有限, 仅供参考!

## 修订记录

修订号 1.0.2	2006-10-28	修订者: MAB
增加对 HFSC 的说明, 更改作者的电子邮箱		
修订号 1.0.1	2003-11-17	修订者: MAB
对引用的 Leonardo Balliache 的文献增加链接		
修订号 1.0	2003-09-24	修订者: MAB
被 TLDP 审查且批准		
修订号 0.7	2003-9-14	修订者: MAB
增加修订号, 校对, 为添加进 TLDP 做好准备		
修订号 0.6	2003-09-09	修订者: MAB
来自 Stef Coene 的细微修正		
修订号 0.5	2003-09-01	修订者: MAB
HTP 章节基本完成, 更多的图表, LARTC 预览版		
修订号 0.4	2003-08-30	修订者: MAB
增加图表		
修订号 0.3	2003-08-29	修订者: MAB
真正的完成无等级, 软件, 规则, 要素及组件(classless, software, rules, elements and components) 章节		
修订号 0.2	2003-08-23	修订者: MAB
主要集中在概览, 要素, 组件和软件(overview, elements, components and software) 章节		
修订号 0.1	2003-08-15	修订者: MAB
初始版本(完成大纲)		

流量控制包含一组机制和操作, 这些机制和操作作用于某个网口上被排队等待发送/接收的报文。其中操作包括入队, 监管, 分类, 调度, 整流, 丢弃。本指南介绍了 Linux 下流量控制的能力及其运用。

本文档允许在自由软件基金的 GNU 1.1 版或以后发布的条款下复制，分发和修改。本文档没有不变的部分，没有封面文字，没有封底文字。许可证副本在：

<http://www.gnu.org/licenses/fdl.html>

## 目录

1. Linux 流量控制介绍.....	4
1.1. 目标及假定读者 .....	4
1.2. 排版约定 .....	4
1.3. 推荐学习方法 .....	4
1.4. 待补充内容，错误指正及读者反馈 .....	5
2. 流量控制思想概述 .....	5
2.1. 什么是流量控制？ .....	5
2.2. 为什么要应用流量控制？ .....	6
2.3. 优点 .....	7
2.4. 缺点 .....	7
2.5. 队列 .....	7
2.6. 数据流 .....	8
2.7. 令牌桶 .....	8
2.8. 包和帧 .....	9
3. 流量控制中几个必备元素 .....	9
3.1. 整流 .....	9
3.2. 调度 .....	10
3.3. 分类 .....	10
3.4. 监管 .....	11
3.5. 丢弃 .....	11
3.6. 标记 .....	11
4. 流量控制组件 .....	11
4.1. 排队规则(qdisc) .....	12
4.2. 分类(class) .....	12
4.3. 过滤器(filter) .....	13
4.4. 分类器(classifier) .....	13
4.5. 监管器(policer) .....	13
4.6. 丢包(drop) .....	14
4.7. 句柄(handle) .....	14
5. 软件包和工具集 .....	14
5.1. 内核版本要求 .....	15
5.2. iproute2 工具 (tc) .....	16
5.3. tcng, 下一代流量控制 (Traffic Control Next Generation) .....	18
5.4. IMQ, 中间队列设备 .....	18
6. 无分类排队规则 (qdiscs) .....	18
6.1. FIFO, 先进先出 (pfifo and bfifo) .....	18
6.2. pfifo_fast, Linux 默认排队排队 .....	20
6.3. SFQ, 随机公平队列 .....	21

6.4.	ESFQ, 扩展的随机公平队列.....	23
6.5.	GRED, 通用早期随机丢包.....	23
6.6.	TBF, 令牌桶过滤器.....	23
7.1.	HTB, 分层令牌桶.....	25
7.1.1.	软件要求.....	26
7.1.2.	整流.....	26
7.1.3.	租借.....	26
7.1.4.	HTB 分类参数.....	28
7.1.5.	规则.....	29
7.3.	PRIQ, 优先级调度.....	29
7.4.	CBQ, 基于类的队列.....	30
8.	流量控制的常见规则和方法.....	30
8.1.	Linux 流量控制的一般规则.....	30
8.2.	在已知带宽的线路上实施流量控制.....	30
8.4.	基于流来分享、划分带宽.....	31
8.5.	基于 IP 来分享、划分带宽.....	31
9.	用于 Qos 和流量控制的脚本.....	31
9.1.	wondershaper.....	31
9.2.	用于 ADSL 宽带的指南脚本 (myshaper).....	31
9.3.	htb.init.....	31
9.4.	teng.init.....	31
9.5.	cbq.init.....	32
10.	图表.....	32
10.1.	总图.....	32
11.	有关流量控制的资源链接.....	33

# Linux 流量控制介绍

Linux 提供了丰富的工具来管理和控制报文的发送。Linux 社区对于在 Linux 下对报文进行修改、应用防火墙策略（netfilter，以及之前的 ipchains）以及大量的可以运行在操作系统上的网络服务工具非常熟悉。但只有小部分 Linux 社区内和更少的 Linux 社区外人员了解 Linux 流量控制子系统的强大功能，并且该系统已经在 2.2 和 2.4 内核中变得更为成熟。

本指南试图介绍流量控制的思想，传统的流量控制要素，Linux 流量控制组件，同时提供了一些参考。这份指南是通过对 LARTC 指南上的文档以及重要的 LARTC 邮件列表学习、整理出来的。

对于没有耐心阅读本指南，想要立即体验流量控制的用户，推荐你们阅读 [Traffic Control using tcng and HTB HOWTO](#) 和 [LARTC HOWTO](#)，这两份文档会很有用。

## 1.1. 目标及假定读者

本指南的目标读者是想对流量控制及对 Linux 下的流量控制工具有所了解的网络管理人员或者有一定基础的家庭用户

本指南的读者应该熟悉 UNIX 操作环境及命令行且具备一定的 IP 网络知识。如果要对流量控制进行应用，还需具有对 linux 内核或软件进行打补丁，编译及安装的能力。对于拥有较新内核版本 (2.4.20 及以上，参见 5.1 节) 的用户，只需要会安装和使用软件就够了。

一般来说，本指南应该适合各类用户。虽然有些读者可能已经有在 Linux 下进行流量控制的经验，但我仍假设所有读者都没有相关经验

## 1.2. 排版约定

本文档是在 DocBook 中通过 vim 来编写的。所有格式由基于 DocBook XSL 和 LDP XSL 表格样式的 xsltproc 来控制。字体格式和显示样式和大多数印刷及电子技术文档相似。

## 1.3. 推荐学习方法

强烈建议读者先通过 tc 和 tcng 来突击学习下流量控制的规则，在集中学习 tcng 之前，只需对 tc 命令行工具有一个大致的了解。tcng 软件包定义了一个完整的语言来描述流量控制结构。第一眼看上去，这门语言会让人生畏，但当你掌握之后，它会使你能很容易使用流量控制，而这是直接使用 tc 工具不能办到的。

如果可能的话，我将以一种抽象的方式来描述 Linux 下流量控制系统的行为，但有些时候我也会借用其他类似的系统来描述这些结构。同一个例子，我不会既用 tcng 语言又用 tc 命令行来实现，聪明的读者会将两者融会贯通。

## 1.4. 待补充内容，错误指正及读者反馈

本指南仍然缺少些内容。下面的内容会在未来某个时间点添加进本文档。

- 有关 GRED, WRR, PRIQ 和 CBQ 的描述及图表。
- 使用示例
- 对分类器(classifier)进行详细介绍的小节。
- 对流量测量技术进行讨论的小节。
- 对标记有覆盖的小节。
- 更多关于 tcng 的细节。

欢迎将建议，错误指正及反馈发送至<martin@linux-ip.net>。理论上来说，所有的错误和疏忽都是我的错，虽然我已经尽力确保了当前文档的正确性，但我不能承担读者根据此文档进行操作后的任何后果。

## 2. 流量控制思想概述

本节将会对流量控制进行介绍并且说明为什么要使用流量控制，然后指出流量控制的一些优缺点，最后介绍在流量控制中涉及的一些关键思想。

### 2.1. 什么是流量控制？

流量控制是路由器上报文的接收和发送机制及排队系统的统称。这包括在一个输入接口上决定以何种速率接收何种报文，在一个输出接口上以何种速率、何种顺序输出何种报文。

在绝大多数情况下，流量控制只包含一个单一队列，此队列以硬件所能处理的最大能力接收报文并尽快输出它们，这种队列就叫 FIFO (first in, first out)。



Linux 下的默认排队规则是 pfifo\_fast，此规则比 FIFO 稍复杂。

在很多类型的软件中都有使用队列的例子。队列是组织未定任务或数据流的一种方法（参考 2.5 节）。因为网络链路层单纯的以串行方式发送数据流，所以队列对于管理输出报文是必要的。

如果一桌面电脑和一台高效 web 服务器共享同一条上行链路，它们对带宽的竞争就可能发生。web 服务器可能会填满路由器的发送队列，当 web 服务器发送数据

的速率大于路由器上数据的输出速率时，路由器就会开始丢包(它的缓冲区满了)。这个时候，桌面电脑(拥有一个交互用户)就会面临丢包及高延时的风险。注意，高延迟有时候会让用户抓狂！通过划分两个内部队列给这两种不同类型的应用使用，这两种应用就可以较好的分享网络资源。

流量控制是一个工具集，它能够让用户通过它在网络设备上应用不同的队列和排队机制。虽然利用这些工具来重新划分数据流和报文的功能很强大，并且可以进行复杂的应用，但最好有足够多的带宽。

术语服务质量（QoS）通常是流量控制的同义词。

## 2.2. 为什么要应用流量控制？

分组交换网络和电路交换网络的一个重要不同之处是：分组交换网络是无状态的，而电路交换网络（比如电话网）必须保持其状态。分组交换网络和 IP 网络一样被设计成无状态的，实际上，无状态是 IP 的一个根本优势。

无状态的缺陷是不能对不同类型数据流进行区分。但通过流量控制，管理员就能够基于报文的属性对其进行排队和区别。它甚至能够被用于模拟电路交换网络，将无状态网络模拟成有状态网络。

有很多实际的理由去考虑使用流量控制，并且流量控制也有很多有意义的应用场景。下面是一些利用流量控制可以解决或改善的问题的例子

下面的列表不是流量控制可以解决的问题的完整列表，此处仅仅介绍了一些能通过流量控制来解决的几类问题

### 常用的流量控制解决方案

- 通过 TBF 和带子分类的 HTB 将带宽限制在一个数值之下
- 通过 HTB 分类(HTB class)和分类(classifying)并配合 filter，来限制指定用户、服务或客户端的带宽。
- 通过提升 ACK 报文的优先级，以及使用 wondershaper 来最大化非对称线路上的 TCP 吞吐量。
- 通过带子分类的 HTB 和分类(classifying)为某个应用或用户保留带宽。
- 通过 HTB 分类(HTB class)中的(优先级)PRIO 机制来提高延时敏感型应用的性能。
- 通过 HTB 的租借机制来管理多余的带宽。
- 通过 HTB 的租借机制来实现所有带宽的公平分配。
- 通过监管器 (policer) 加上带丢弃动作的过滤器(filter)来使某种类型的流量被丢弃。

请记住，很多时候，最好去购买更大的带宽，流量控制并不能解决所有的问题。

## 2.3. 优点

当正确的使用流量控制时，就能提高网络资源的利用率，并能减少潜在的竞争。正确使用后，网络就会符合流量控制的目标，比如能给大流量的下载分配一个合适的带宽然后使高优先级的交互报文也能进行正常的传送，低优先级的数据（比如邮件）也能正常的传输而不明显的影响其它数据传送。

广义上来说，如果流量控制能符合已和用户约定好的策略，那么用户就能最大限度的使用网络资源。

## 2.4. 缺点

复杂性是流量控制的主要缺点。而流量控制工具降低了学习流量控制及其机制的难度，虽然有一些方式来熟悉这些流量控制工具，但是从流量控制中找出不正确的配置也是一个挑战

恰当的使用流量控制可以使网络资源更加公平的被分配，但流量控制很容易被使用不当，如果使用不当就会导致对网络资源的分配更不公，对网络资源的竞争会更大。

为了维护流量控制结构，需要路由器上有足够的计算资源。幸运的是，它只会消耗较小的计算资源，不过当流量控制规则变得庞大且复杂时它所消耗的资源就不容忽视。

对于个人用户来说，流量控制几乎没什么成本。但对于公司来说，应用流量控制所带来的费用可能会比购买更多的带宽还多，因为培训一个精通流量控制的人员会比购买更多的带宽花费多。

尽管分组交换网上的流量控制是一种很宽泛的概念，但你可以认为流量控制提供了一种将电路交换网的有状态属性应用到分组交换网的方式。

## 2.5. 队列

队列是流量控制的基础，而且是调度的实现思想。一个队列中包含有限个对象，这些对象将在队列中等待被处理。在网络中，这些对象就是数据包，这些数据包在队列中等待被硬件传输。在最简单的模式中，数据包按照先进先出的原则进行传输，这种队列在计算机网络（或更广泛的计算机科学）中被称为 FIFO。

当没有和其它机制配合使用时，队列并不能提供流量控制功能。实际上，队列只有两种操作：将对象加入到队列中，即入队，将对象从队列中移除，即出队。

当和其他组件配合时队列就会具有复杂的功能，比如在多队列中对报文进行延迟，对报文进行重新排队，丢弃报文，对报文进行分级。队列中也可以包含子队列，以便使用更为复杂的调度方式。

对于高层应用来说，报文就是入队然后发送，对于报文是以何种顺序、何种方式被发送的，高层应用并不关心。因此，对于高层应用来说，流量控制表现出来的就是一个单一队列。只有对应用流量控制的那一层来说，流量控制结构才是可见的。

## 2.6. 数据流

一条数据流就是两台主机之间的连接或会话。两台主机之间的任何数据交互报文可以被认为一条数据流。在 TCP 中，源 IP 加源端口和目的 IP 加端口的连接就决定了一条数据流，UDP 也类似。

流量管理机制常常会将流量划分为不同类的数据流，这些流可以被聚合起来然后作为一条聚合流被传输（DiffServ，差分服务）。不同的流量控制机制能基于不同的流将带宽进行均等的划分。

如果想对带宽进行均分，然后分配给对带宽有竞争关系的数据流，那么对数据流进行处理就很重要。特别是有些应用会产生大量的数据流，这时的流量管理就更为重要。

## 2.7. 令牌桶

整形的两个关键要素就是令牌桶。

为了控制出队的速率，一种方式就是直接统计队列中出队的报文或字节数，但是为了保证精确性就需要复杂的计算。在流量控制中广泛应用的另一种方式就是令牌桶，令牌桶以一定的速率产生令牌，报文或字节出队时从令牌桶中取令牌，只有取到令牌后才能出队。

我们可以打一个比方，一群人正排队等待乘坐游乐场的游览车。让我们想象现在有一条固定的道路，游览车以固定的速度抵达，每个人都必须等待游览车到达后才能乘坐。游览车和游客就可以类比为令牌和报文，这种机制就是速率限制或流量整形，在一个固定的时间段内只有一部分人能乘坐游览车。

继续上面的比方，设想有大量的游览车正停在车站等待游客乘坐，但现在没有一个游客。如果现在有一大群游客同时过来了，那么他们都可以马上乘上游览车。在这里，我们就可以将车站类比为桶，一个桶中包含一定数量的令牌，桶中的令牌可以一次性被使用完而不管数据包到达的时间。



让我们来完成这个比方，游览车以固定的速率抵达车站，如果没人乘坐就会停满车站，即令牌以一定的速率进入桶中，如果令牌一直没被使用那么桶就可以被装满，而如果令牌不断的被使用那么桶就不会满。令牌桶是处理会产生流量突发应用（比如 HTTP）的关键思想。

使用令牌桶过滤器的排队规则（TBF qdisc, Token Bucket Filter）是流量整形的一个经典例子（在 TBF 小节中有一个图表，通过该图表可以形象化的帮助读者理解令牌桶）。TBF 以给定的速度产生令牌，当桶中有令牌时才发送数据，令牌是整流的基本思想。

在某些情况下，队列中可能无数据包，因此不会立即需要令牌，这些令牌就会被收集起来直到队列需要。但无限制的积累令牌可能会失去流量整形的意义，所以令牌积累到一定量就会停止。由于积累了令牌，当有大量的报文或字节需要出队时就有足够的令牌可用。这种虚拟的令牌被存储在虚拟的桶中，一个桶中能存放多少令牌取决于桶的大小。

这意味着如果桶的大小设计不合理，在任何时刻桶中都可能充满令牌。小型的令牌桶适合流量较为稳定的网络，大型的令牌桶适合有较大突发流量的网络，除非你的目的就是限制数据流的突发传输。

总之，令牌以固定的速度产生，桶中可以积满令牌，这样就可以处理突发流量，使网络流量更为平滑。

令牌和桶的概念紧密相联，它们被用于 TBF（一种无类 qdiscs）和 HTB（一种分类 qdiscs）中。在 tcng 语言中，二色和三色标识法就是令牌桶的应用。

## 2.8. 包和帧

在网络上传输的数据该被称为什么，其术语取决于其所在的网络层。本文档将不会区分两者在技术上的差别（不区分是错的），尽管在这里列出了这两个术语。

帧通常是用来描述第 2 层（数据链路层）上发送给下一接收者的数据。以太网接口，PPP 接口，T1 接口都把他们的第 2 层数据叫作帧。在流量控制中，帧其实就是其处理单位。

包就是对更高层数据单元的称呼，即第 3 层（网络层），在本文档中我们更多的会使用包这一术语，虽然这么用不够准确。

（译注：在本译文中，数据包和报文混用）

# 3. 流量控制中的几个必备元素

## 3.1. 整流

整流器通过延迟数据包来使流量保持在一定速率。

整流就是让包在输出队列上被发送之前进行延时，然后一定的速率发送，使网络流量保持在一定的速率之下，这是大部分用户进行流量控制的目的。报文延时作为流量控制的一部分，使得所有整流算法都是非工作保留模式的，大致解释就是“必须保持工作来延迟报文”。

非工作保留模式提供了整形的功能，反过来看，工作保留排队机制（参考 PRIO，优先级调度）就不具备流量整形的功能，因为其不能对报文进行延时。

整流器将流量控制在一个给定速率下（通常以 pps, bps, Bps 为单位）。当然其也有副作用，那就是整流器平滑了突发流量。对带宽进行整流的好处是可以控制报文的延时，整流的思想就是利用令牌桶，可以阅读 2.7 节获取更多有关令牌桶的信息。

（译注：工作保留就是为较高优先级的报文保留部分资源，在非工作保留模式下，当有报文可处理时就处理，在工作保留模式下，即使有报文需要处理，就算资源空闲，其也会为高优先级的报文保留部分资源）

## 3.2. 调度

调度器(scheduler)排列或重排输出报文。

调度就是对队列中的输入输出报文进行排列。最常的调度方法就是 FIFO（先进先出），更广泛的来说，在输出队列上的任何流量控制都可以被称作调度，因为报文被排列以被输出。

有很多调度算法被用于不同的网络环境，一种公平调度算法（参考 SFQ）用来阻止某一个客户端或数据占用太多网络资源，一种轮循调度算法（参考 WRR）让每条数据流轮流出队，还有一些比较复杂的调度算法用来防止主干网流量过载（参考 GRED）或还有一些用来改进其它调度算法的算法（参考 ESFQ）。

## 3.3. 分类

分类器(classifier)将流量分类、划分到不同队列中。

分类就是将流量进行划分以便区别处理，例如拆分后放到不同的输出队列中。在报文的接收、路由、发送过程中，网络设备可以用多种方式来分类报文。分类包括对报文进行标记，标记可以在边缘网络中由一个单一的控制单元来完成，也可以在每一跳中都进行标记。

Linux 允许报文依次通过一系列分类器，而且还可以通过和监管器(policer)配合来对报文进行分类。

## 3.4. 监管

监管器 (Policer) 计算和限制队列中的流量。

监管作为流量控制的一部分，就是用于限制流量。监管常用于网络边缘设备，使某个节点不能使用多于分配给它的带宽。监管器以特定的速率接收数据包，当流量超过这一速率时就对接收的数据包执行相应的动作。最严格的动作就是丢弃数据包，尽管该数据包可以被重新分类。

当流量以一定的速率进入队列时，监管器只有两个动作：当数据包以低于给定速率进入队列时，执行一个动作（允许入队），当数据包以高于给定速率进入队列时，执行另一个动作。尽管监管器内部使用了令牌桶，但是它不具备整流器延迟报文的能力。


## 3.5. 丢弃

丢弃就是丢掉整个数据包，整条流或一类流量。

丢弃就是通过某种机制来选择哪个数据包被丢掉。

## 3.6. 标记

标记对数据进行更改。

 这里不是指 fwmark, iptables 和 ipchains 的标记都是用来修改数据包的元数据，而不是数据包本身。

流量控制在数据包中插入了 DSCP 部分，在一个可管理网络中，其可被其它路由器利用和识别（通常用于 DiffServ，差分服务）。

# 4. 流量控制组件

表 1. 流量控制元素和 Linux 组件之间的联系

传统元素	Linux 组件
整流(shaping)	class 组件提供整流功能。
调度(scheduling)	qdisc 是一个调度器。调度器可简可繁，简单的如 FIFO，复杂的如带分类的 qdisc，其他的如 HTB。
分类(classifying)	filter 加上 classifier 提供了分类功能，严格来说，Linux 的 classifier 离不开 filter。

传统元素	Linux 组件
监管 (policing)	在 Linux 流量控制机制中，policer 仅作为 filter 一个功能。
丢弃(dropping)	为了运用丢弃，需要一个 filter 和一个动作是“drop”的 policer 相配合。
标记(marking)	qdisc 提供标记的功能。

## 4.1. 排队规则(qdisc)

简单来说，qdisc 是一个调度器（参见 3.2 节）。每个输出接口都有一个调度器，默认的调度器就是 FIFO。Linux 下可用的调度器会按照调度器的规则对进入队列的包进行重新排列。

qdisc 是 Linux 流量控制设计的主要部分，也被称为排队规则 (quenuing discipline)。

分类 qdisc (classful qdisc) 可以包含多个类别 (class)，其也提供了可以附加到 filter 的句柄。使用不带子类的 classful qdisc 是允许的，尽管这会造成系统资源的浪费。

非分类 qdisc 是不带 class 的，所以其也不能被附加到一个 filter，由于非分类 qdisc 不带任何子对象，所以不能用来对流量进行分类。这就意味着没有 filter 能被附加到一个非分类 qdisc。

术语 root qdisc 和 ingress qdisc 常使人混淆，其实它们都不是真正的排队规则，它们只是流量控制可以被实施的地方，即入口（输入流量）和出口（输出流量）处。

每个接口既包含 root qdisc 又包含 ingress qdisc。最主要也是最常见的就是入口排队规则 (egress qdisc)，即 root qdisc。它能包含任何带分类或分类结构的排队规则。绝大多数文档都是针对 root qdisc 和它的子集的，接口上的流量都得通过 egress 或 root qdisc。

对于在接口上接收的流量，会先通过 ingress qdisc，由于组件的限制，ingress qdisc 不能创建子分类，而且只能和一个 filter 相关联。在实际运用中，ingress qdisc 仅仅和一个 policer 相关联来方便的限制网络接口上接收的数据量。

简而言之，你可以利用 egress 实现更为强大的功能，因为它包含一个真正的 qdisc 并且具有流量控制系统的完整功能。一个 ingress qdisc 仅仅支持一个 policer。本文档的余下部分除非有特殊说明，将只关心和 root qdisc 有关的流量控制结构。

## 4.2. 分类(class)

分类只存在于可分类排队规则（`classful qdisc`）（例如，HTB 和 CBQ）中。分类可以很复杂，它可以包含多个子分类，也可以只包含一个子 `qdisc`。在超级复杂的流量控制应用场景中，一个类中再包含一个可分类 `qdisc` 也是可以的。

任何一个分类都可以和任意多个 `filter` 相关联，这样就可以选择一个子分类或运用一个 `filter` 来重新排列或丢弃进入分类中的数据包。

叶子分类是 `qdisc` 中的最后一个分类，它包含一个 `qdisc`（默认是 FIFO）并且不包含任意子分类。任何包含子分类的分类都是内部分类而不是子分类。

### 4.3. 过滤器(filter)

过滤器是流量控制系统中最复杂的部分。过滤器提供了一种方便的机制将流量控制中的几个关键要素集合在了一起。过滤器最简单且最常见的功能就是分类数据包（参见 3.3 节）。Linux 的过滤器可以允许用户利用一个或多个过滤器将数据包分类至输出队列上。

- 一个过滤器必须包含一个分类器(classifier)。
- 一个过滤器可以包含一个监管器(policer)。

过滤器既可以和可分类的 `qdisc` 相关联，也可以和分类（`class`）相关联。已入队的数据包首先通过 `root qdisc`，当数据包通过了和 `root qdisc` 相关联的过滤器后，数据包可以被送入子分类(subclass)（子分类也可以有自己的过滤器）以进行进一步的分类。

### 4.4. 分类器(classifier)

分类器(filter)可以通过 `tc` 工具来管理，它可以运用不同的分类器，最常用的就是 `u32` 分类器，`u32` 分类器可以允许用户基于数据包的属性来选择数据包。

分类器作为 `filter` 的一部分，可以用来识别数据包或数据包元数据的特征。Linux 分类器是 Linux 流量控制中有关分类的直接实现。

### 4.5. 监管器(policer)

在 Linux 中，监管器这个关键部分只能作为 `filter` 的一部分来使用，监管器在速率超过指定速率时执行一个动作，在速率低于指定速率时执行另一个动作。可以巧妙的利用监管器来模拟三色标识法，参考第 10 节。

当利用监管器来限制带宽时，尽管监管和整形都是流量控制的基本元素，但它们从不延迟任何流量。基于指定的准则，监管器只能执行一个动作。可参考例 5。

## 4.6. 丢包(drop)

在 Linux 流量控制中, drop 仅能作为 policer 的一部分。任何和 filter 相关联 policer 都可以有 drop 动作。



在 Linux 流量控制系统中, 数据包只能在 policer 中被显示的丢弃, policer 能以指定的数率限制数据包入队, 也能丢弃匹配某种模型的所有数据包。

然而, 在流量控制系统的某些地方, 数据包可能作为一个副作用被丢弃。例如, 如果调度器(scheduler)使用 and GRED 一样的方式来控制数据流时, 数据包就可能被丢弃。

另外, 当流量突发或流量过载的时候, 整形器(shaper)或调度器(scheduler)可能会耗尽分配给它的缓冲区而不得不丢弃数据包。

## 4.7. 句柄(handle)

在流量控制结构中, 每个分类或可分类 qdisc(参见第 7 节)都需要一个唯一的标识符。这个唯一标识符就是我们所说的句柄, 一个句柄由两组数字组成: 一个主编号和一个次编号。这些编号可以由用户根据下面的规则任意分配。

### class 和 qdisc 句柄的编号准则

#### 主编号(*major*)

这个编号对内核来说完全没有意义。用户可以使用任意一种方式来编号, 但是在流量控制系统中, 具有相当父对象的子对象应该使用相同的主句柄号。和 root qdisc 直接关联的对象, 习惯上从 1 开始给它们编号。

#### 次编号(*minor*)

如果次编号为 0, 那么就明确的表明这个对象是一个 qdisc, 其他非零值则表示该对象是 class。所有拥有同一父 class 的 class 必须使用同一个次编号。

ffff:0 这个句柄号保留给 ingress qdisc 使用。

在 tc 的 filter 中, 句柄被作为 classid 和 flowid 的参数使用。句柄是对象的外部标识符, 在用户侧应用中被使用, 而在内核中, 内核为每个对象都分配了一个对应的内部标识符。

## 5. 软件包和工具集

## 5.1. 内核版本要求

许多 Linux 发行版都提供了带流量控制 (QoS) 功能的内核，该功能作为一个单独的模块存在，或直接编译在内核中。定制化的内核可能不包含流量控制功能，这时就可以按照下面列出的内核选项配置内核。

对于编译内核没有或只有一点经验的用户，建议参考 [Kernel HOWTO](#)。对于编译内核有经验的用户，在对流量控制有一定了解后，应该知道下面的选项哪一些应该被选择。

### 例 1. 内核编译选项

```
#
# QoS 和或公平排队(fair queueing)
#
CONFIG_NET_SCHED=y
CONFIG_NET_SCH_CBQ=m
CONFIG_NET_SCH_HTB=m
CONFIG_NET_SCH_CSZ=m
CONFIG_NET_SCH_PRIO=m
CONFIG_NET_SCH_RED=m
CONFIG_NET_SCH_SFQ=m
CONFIG_NET_SCH_TEQL=m
CONFIG_NET_SCH_TBF=m
CONFIG_NET_SCH_GRED=m
CONFIG_NET_SCH_DSMARK=m
CONFIG_NET_SCH_INGRESS=m
CONFIG_NET_QOS=y
CONFIG_NET_ESTIMATOR=y
CONFIG_NET_CLS=y
CONFIG_NET_CLS_TCINDEX=m
CONFIG_NET_CLS_ROUTE4=m
CONFIG_NET_CLS_ROUTE=y
CONFIG_NET_CLS_FW=m
CONFIG_NET_CLS_U32=m
CONFIG_NET_CLS_RSVP=m
CONFIG_NET_CLS_RSVP6=m
CONFIG_NET_CLS_POLICE=y
```

按上面的选项编译后的内核将能以模块化的方式提供本文档所讨论的任何组件。当需要使用相应功能时，用户可以使用 modprobe 来加载对应模块。再说一次，推荐有困惑的用户去阅读 [Kernel HOWTO](#)，通过本文档并不能解决有关使用 Linux 内核的问题。

## 5.2. iproute2 工具 (tc)

iproute2 是一组命令行工具组件，用来操作一台机器上有关 IP 网络配置的内核结构。有关这些工具的技术文档，参见 [iproute2 documentation](#)，如果想了解更多，可以参考 [linux-ip.net](#) 上的文档。在 iproute2 软件包的众多工具中，tc 是唯一用来进行流量控制的工具，本指南将忽略其它工具。

由于 tc 直接作用于内核来创建、删除、修改流量控制结构，所以在编译 tc 时需要提前安装你想使用的所有 qdisc。特别的，HTB qdisc 还未在上游 iproute2 软件包中被支持，更多信息请参见 7.1 小节。

### 例 2. 使用 tc 命令

```
[root@leander]# tc
Usage: tc [ OPTIONS ] OBJECT { COMMAND | help }
where  OBJECT := { qdisc | class | filter }
        OPTIONS := { -s[tatistics] | -d[etails] | -r[aw] }
```

每个对象都有更多且不同的选项，本文档不会完整涉及到所有选项，下面的例子将尽可能展现 tc 命令的使用。可以访问 [LARTC HOWTO](#) 来获取更多的例子，如果想更好的理解 tc，可以参考内核和 iproute2 的代码。

### 例 3. tc qdisc

```
[root@leander]# tc qdisc add \ ❶
> dev eth0 \ ❷
> root \ ❸
> handle 1:0 \ ❹
> htb ❺
```

❶添加一个排队规则，也可以使用 del 来删除。

❷指定将和新排队规则相关联的设备。

❸对 tc 来说，root 就是指 “egress”，但是这里必需使用单词 “root”。对于另一个有功能限制的 qdisc: ingress qdisc，其可以被关联到同一设备上。

❹handle 是用户指定的一个编号，其格式是 主编号:次编号。对任何排队规则句柄来说，次编号必需是 0。排队规则(qdisc)句柄的一种可用简写形式是“1:”，当没有指定次编号时就默认为 0。

❺需要使用的排队规则，本例中是 HTB。排队规则需要的一些参数可以跟在后面，本例中，我们没有其他参数。



上例是使用 tc 工具给一个设备添加排队规则的简单例子，下面是使用 tc 给一个存在的父 class 添加子 class 的例子。

#### 例 4. tc class

```
[root@leander]# tc class add \ ❶
> dev eth0 \ ❷
> parent 1:1 \ ❸
> classid 1:6 \ ❹
> htb \ ❺
> rate 256kbit \ ❻
> ceil 512kbit ❼
```

❶添加一个 class，也可以用 del 删除。

❷指定我们将要关联新 class 的设备。

❸指定我们将要关联新 class 的父 class 句柄。

❹标识此 class 的唯一句柄(主编号：次编号)。次编号必需为非零值。

❺带分类的 qdisc 需要所有的子 class 都和它是相同的类型，因此，HTB qdisc 将包含 HTB classes。

❻❼class 的参数，查看 7.1 节来获取关于这些参数的更多详细信息。

#### 例 5. tc filter

```
[root@leander]# tc filter add \ ❶
> dev eth0 \ ❷
> parent 1:0 \ ❸
> protocol ip \ ❹
> prio 5 \ ❺
> u32 \ ❻
> match ip port 22 0xffff \ ❼
> match ip tos 0x10 0xff \ ❽
> flowid 1:6 \ ❾
> police \ ❿
> rate 32000bps \ (11)
> burst 10240 \ (12)
> mpu 0 \ (13)
> action drop/continue (14)
```

❶添加一个 filter，也可以使用 del 来删除。

❷指定新的 filter 将要关联的设备。

❸指定新的 filter 将要关联的父句柄。

❹这个参数是必要的，它应该被显示的使用，尽管我不知道它还有什么其他功能。

❺prio 参数能使一个 filter 在另一个之前被使用，pref 是 prio 的同义词。

❽这是一个分类器(classifier)，所有 tc filter 命令都需包含该选项。

❼❽❾❿(11)(12)(13)(14)这些是分类器的参数。在本例中，带有服务标识且端口号是 22 的数据包将会被选择。

⑨ `flowid` 指定了一个目标 `class`(或 `qdisc`)句柄, `filter` 所选择的数据包将发往这个句柄所指向的 `class`(或 `qdisc`)。

⑩ 这是一个监管器(`policer`), 在 `tc filter` 命令中, 这是一个可选项。

(11) 当超过这个速率时, 监管器将执行一个动作, 低于这个速率时, 监管器将执行另一个动作。

(12) `burst` 相当于 HTB 中的 `burst` (`burst` 采用的是桶(`bucket`)原理)。

(13) 被监管的最小数据单位。为了统计所有流量, 使用 `mpu 0`。

(14) `action` 指定了当流量大于给 `policer` 所配置的 `rate` 时应采取的动作。第一个词指明了当流量超过 `rate` 时所执行的操作, 第二个词指明了其他情况下应该采取的操作。

正如你在上面所看到的, `tc` 命令行工具语法晦涩且复杂, 即使是一个简单的操作都要进行复杂的配置。当然你不必感到吃惊, 因为在 Linux 下, 我们有更容易的方式来配置流量控制结构。参见下一小节 5.3。

## 5.3. tcng, 下一代流量控制 (Traffic Control Next Generation)

此节有待补充; 让我们为 `tcng` 喝彩。有关 `tcng` 的信息可参见 [Traffic Control using tcng and HTB HOWTO](#) 和 [tcng documentation](#)。

下一代流量控制(也就是这里的 `tcng`)将会以一种非常简单的方式提供 Linux 下流量控制所有的功能。


## 5.4. IMQ, 中间队列设备

此节有待补充; 对于 IMQ 的探讨是必须的。可以参考 Patrick McHardy 网站上有关 IMQ 的内容。

# 6. 无分类排队规则 (qdiscs)

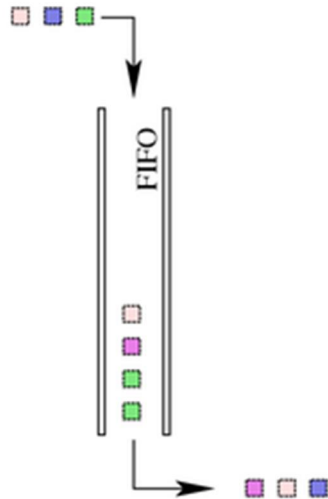
下面的任一无类排队规则都可以作为一个网络接口的主要排队规则, 或者作为分类排队规则的叶子分类(`leaf class`)。下面的这些排队规则都是在 Linux 下使用的基本规则。

## 6.1. FIFO, 先进先出 (pfifo and bfifo)

 FIFO 不是 Linux 网络接口上使用的默认排队规则(`qdisc`), 有关默认排队规则(`pfifo_fast`)的完整信息请参考 6.2 节。

FIFO 算法是所有 Linux 网络接口默认排队规则的基本组成形式。FIFO 既不能对流量进行整形也不能对数据包进行重新排列,当收到数据包后它只能尽快的传输它们。FIFO 也是所有新创建的 class 内部使用的排队规则,直到被其他排队规则或 class 替换。

## First-in First-out (FIFO)



实际使用的 FIFO 都有一个缓冲区来防止溢出,当 FIFO 不能及时将它所收到的数据包出队时就使用缓冲区将数据包暂时缓存起来。Linux 运用了两种类型的 FIFO,一种基于字节,一种基于报文。不管是哪种类型的 FIFO,队列的大小都由参数 limit 来定义。基于报文的 fifo(pfifo)其单位就是报文,基于字节的 fifo(bfifo)其单位就是字节。

### 例 6. 为基于报文或字节的 FIFO 设置一个大小

```
[root@leander]# cat bfifo.tcc
/*
 *在 eth0 上创建一个大小为 10k 字节的 FIFO 队列
 *
 */

dev eth0 {
    egress {
        fifo (limit 10kB );
    }
}
[root@leander]# tcc < bfifo.tcc
# =====Device eth0 =====

tc qdisc add dev eth0 handle 1:0 root dsmark indices 1 default_index 0
```

```

tc qdisc add dev eth0 handle 2:0 parent 1:0 bfifo limit 10240
[root@leander]# cat pfifo.tcc
/*
 *在 eth0 上创建一个可容纳 30 个数据包的 FIFO 队列
 *
 */

dev eth0 {
    egress {
        fifo (limit 30p );
    }
}
[root@leander]# tcc < pfifo.tcc
# =====Device eth0 =====

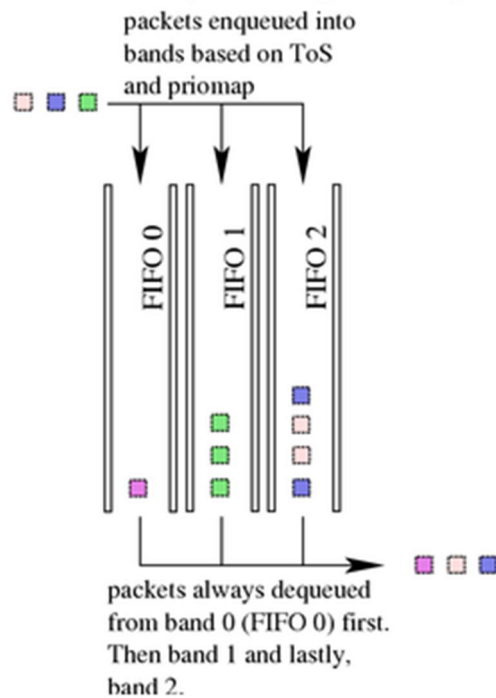
tc qdisc add dev eth0 handle 1:0 root dsmark indices 1 default_index 0
tc qdisc add dev eth0 handle 2:0 parent 1:0 pfifo limit 30

```

## 6.2. pfifo\_fast, Linux 默认排队排队

pfifo\_fast 是 Linux 上所有接口的默认排队规则(qdisc)。pfifo\_fast 基于传统的 FIFO 排队规则，但它支持优先级，它为分离的流量提供了三种不同等级的通道(三条独立的 FIFO)。最高等级的流量（交互数据流）被放入 0 号通道且被优先处理，通道 1 中的数据包优先于通道 2 中的数据包被处理。

## pfifo\_fast queuing discipline

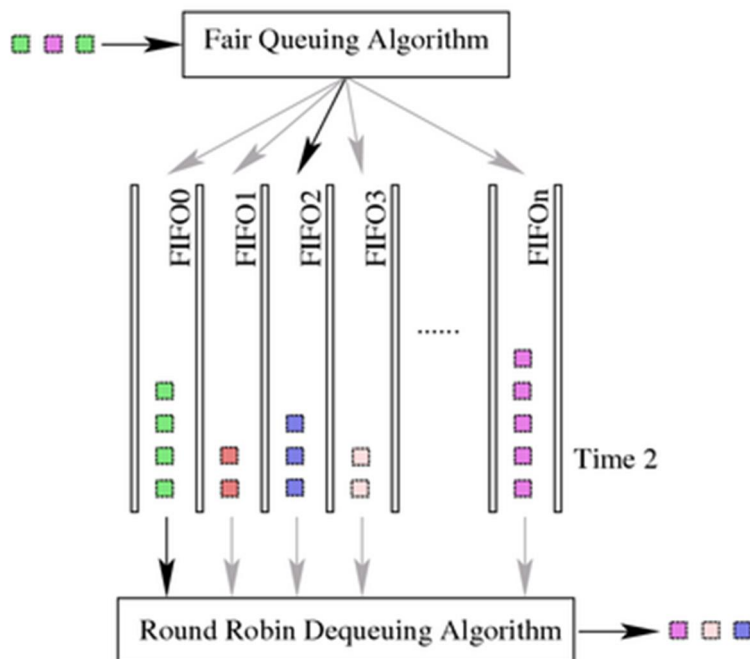


pfifo\_fast 对用户来说没有什么需要配置的。有关 priomap 和 Tos 使用的更详细信息请参考 [pfifo-fast section of the LARTC HOWTO](#)。

### 6.3. SFQ, 随机公平队列

SFQ 试图让任意数量的数据流都有平等的机会来向网络传输数据。它通过一个哈希函数将数据流分配到不同的 FIFO 队列中，然后以轮循的方式依次出队这些队列中的数据。由于所选择的哈希函数可能会产生明显的不公平性，哈希函数会被定期的更换，参数 perturb 用来设置哈希函数的更换周期。

## Stochastic Fair Queuing (SFQ)



### 例 7. 创建一个 SFQ

```
[root@leander]# cat sfq.tcc
/*
*在 eth0 上创建一个周期为 10 的 SFQ 队列
*
*/

dev eth0 {
    egress {
        sfq( perturb 10s );
    }
}

[root@leander]# tcc < sfq.tcc
# ===== Device eth0 =====

tc qdisc add dev eth0 handle 1:0 root dsmark indices 1 default_index 0
tc qdisc add dev eth0 handle 2:0 parent 1:0 sfq perturb 10
```

不幸的是，一些自作聪明的软件（例如 Kazaa，eMule 等等）破坏了公平排队的好处，它们持续不断的创建尽可能多的 TCP 会话。在许多网络中，如果应用能遵

守规范，SFQ 能很好的将网络资源分给对资源有竞争的数据流，但当某些讨厌的应用产生大量的数据流时就必需采用其它措施了。

参考 6.4 节，了解一种可供用户使用的，拥有更多参数的 SFQ 排队规则。

## 6.4. ESFQ，扩展的随机公平队列

从概念上来讲，ESFQ 除了拥有更多的可用参数外，和它的同辈 SFQ 没有什么不同。ESFQ 只是为了克服上节所述的 SFQ 的缺点而被提出。它允许用户选择哈希算法来分配网络带宽，使带宽的分配更公平。

### 例 8. ESFQ 的使用

```
Usage: ... esfq [ perturb SECS ] [ quantum BYTES ] [ depth FLOWS ]  
          [ divisor HASHBITS ] [ limit PKTS ] [ hash HASHTYPE]
```

Where:

```
HASHTYPE := { classic | src | dst }
```

待补充，需要补充实际例子和解释。

## 6.5. GRED，通用早期随机丢包

待补充，我从没有使用过 GRED，需要实用例子和解释。

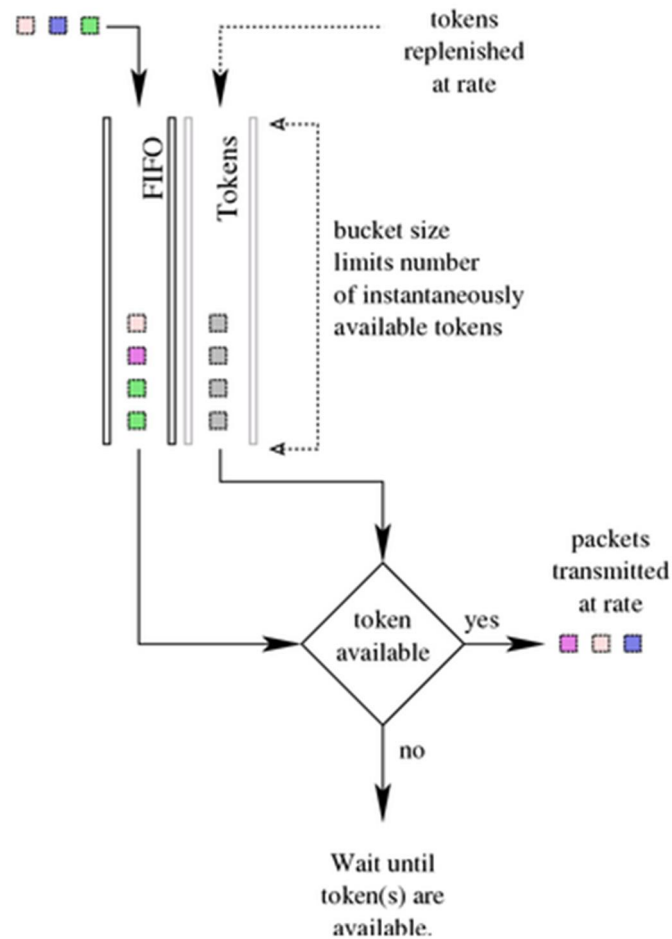
RED 算法的原理表明，它在骨干网和核网上非常有用，在末级用户处几乎无用。查看数据流这一节来简单了解下 TCP

## 6.6. TBF，令牌桶过滤器

TBF 是建立在令牌桶之上，它对接口上的流量进行整形。为了限制某一特定接口上报文的出队速率，TBF 排队规则将是一个不错的选择，它能将报文的发送速率降到指定值。

如果有足够的令牌，数据包就被传输，否则，数据包会被延迟。通过这种方式延迟报文将会给报文的往返造成人为的延时。

## Token Bucket Filter (TBF)



### 例 9. 创建一个速率为 256kbit/s 的 TBF

```
[root@leander]# cat tbf.tcc
/*
 * 在 eth0 上创建一个速率为 256kbit/s 的 TBF
 *
 */

dev eth0 {
    egress {
        tbf( rate 256 kbps, burst 20 kB, limit 20 kB, mtu 1514 B );
    }
}

[root@leander]# tcc < tbf.tcc
# ===== Device eth0 =====

tc qdisc add dev eth0 handle 1:0 root dsmark indices 1 default_index 0
tc qdisc add dev eth0 handle 2:0 parent 1:0 tbf burst 20480 limit 20480 mtu 1514
```



```
rate 32000bps
```

## 7. 分类排队规则(qdiscs)

通过分类排队规则可以使流量控制更易操作、更灵活。记住，分类排队规则可以和 filter 相关联，这样就可以让数据包被送往特定的分类或子队列。

有几个常见的术语来描述和 root qdisc 以及末梢分类直接相关联的分类(class)。和 root qdisc 相关联的分类，通常被称为根分类(root class)，或者更通用的被称为内部分类(inner class)。在特定排队规则中的末梢分类(terminal class)通常被称为叶子分类(leaf class)，就好比一棵树的叶子一样。在本文中，除了将这种结构比喻成一棵树外，还会运用家庭成员关系来描述。

### 7.1. HTB, 分层令牌桶

HTB 利用令牌和桶的概念，并基于分类和 filter 实现了一个复杂且精细的流量控制方法。通过复杂的租借机制(borrowing model)，HTB 能实现复杂且多样的流量控制机制。HTB 最简单、最快速的应用就是整形(shaping)。

如果理解了令牌和桶，或者了解 TBF 的机制，那么学习 HTB 就轻而易举。HTB 允许用户自己定义令牌和桶的属性，然后按照需求对这些桶分类。当和分类机制(classifying scheme)配合使用时，就能将流量控制得更为精细。

下面是 tc 工具在命令行上输出的 HTB 用法，尽管 tcng 有它自己的语法，但对于 HTB 的使用都是相同的。

#### 例 10. tc 的 HTB 用法

```
Usage: ... qdisc add ... htb [default N] [r2q N]
  default  minor id of class to which unclassified packets are sent {0}
  r2q      DRR quantums are computed as rate in Bps/r2q {10}
  debug    string of 16 numbers each 0-3 {0}

... class add ... htb rate R1 burst B1 [prio P] [slot S] [pslot PS]
           [ceil R2] [cburst B2] [mtu MTU] [quantum Q]
  rate     rate allocated to this class (class can still borrow)
  burst    max bytes burst which can be accumulated during idle period {computed}
  ceil     definite upper class rate (no borrows) {rate}
  cburst   burst but for ceil {computed}
  mtu      max packet size we create rate map for {1600}
  prio     priority of leaf; lower are served first {0}
```

```
quantum  how much bytes to serve from leaf at once {use r2q}
```

TC HTB version 3.3

### 7.1.1. 软件要求

不像这里讨论的其它规则，HTB 是一个比较新的排队规则，你所使用的 Linux 发行版可能没有相应的操作工具或者没有 HTB 的功能。要使用 HTB，就必需使用支持 HTB 的内核，内核主干版本 2.4.20 及其以上版本支持 HTB，较低版本的内核需要打相应的补丁。为了在应用中使用 HTB，参见 [HTB](#) 中有关 tc 的 iproute2 补丁信息。

### 7.1.2. 整流

HTB 最常用的用法包含将网络流量限制在一个指定速率之下。

所有的整形都发生在叶子分类中(leaf class)，在根(root)或内部(inner)分类(class)中不会进行整形，根分类仅仅决定租借机制如何分发可用令牌。

### 7.1.3. 租借

HTB 排队规则的一个基本组成部分就是租借机制。当子分类的流量超过了 rate 之后，就可以向父分类借用令牌。一个子分类会一直尝试向父分类借用令牌，直到该子分类的流量达到 ceil，这个时候子分类就会将数据包入队，以等待有更多的可用令牌。HTB 只创建两种基本类型的分类，下面的图表列出了在租借机制影响下，两种分类在各种可能状态可能会产生的相应动作。

表 2. HTB 分类在不同状态下可能会采取的动作

分类类型	类状态	HTB 内部状态	动作
叶子 (leaf)	$< rate$	<i>HTB_CAN_SEND</i>	只要有可用令牌就发送数据包（不超过 burst 个数据包）。
叶子 (leaf)	$> rate,$ $< ceil$	<i>HTB_MAY_BORROW</i>	叶子分类会尝试向父分类借用令牌(token 或 ctoken)，如有父分类有足够的令牌，叶子分类会以 quantum 的整数倍借用令牌发送数据，但发送的数据包总数不能超过 cburst 字节。
叶子 (leaf)	$> ceil$	<i>HTB_CANT_SEND</i>	将不会发送任何数据包，为了将速率限制在 rate 之下，会对数据包造成延时。

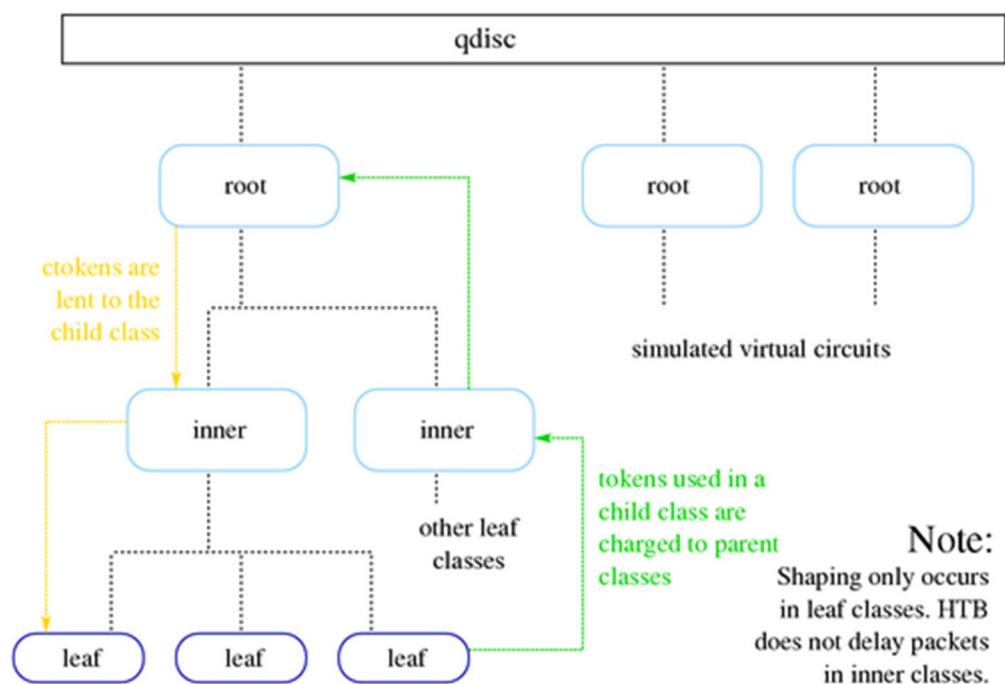
分类类型	类状态	HTB 内部状态	动作
inner, root	$< rate$	<i>HTB_CAN_SEND</i>	内部分类向子分类借出令牌。
inner, root	$> rate, < ceil$	<i>HTB_MAY_BORROW</i>	内部分类会尝试向父分类借用令牌(token 或 ctoken), 然后以 quantum 的整数倍将借用的令牌再租借给它的子分类。
inner, root	$> ceil$	<i>HTB_CANT_SEND</i>	内部分类既不会向父分类借用令牌, 也不会向它的子分类借出令牌。

下图描绘了借用令牌时令牌的流向, 以及向父分类归还令牌的方法。为了使租借模型能正常工作, 必须明确的指定每个分类及其该分类所有子类的可用令牌数量。每个子分类都向它的父分类归还令牌, 这样一级级归还, 直到根分类。

任何想要借用令牌的子分类都向它的父分类借用, 如果父分类由于超过了它的 rate 而没有令牌, 父分类就会向它的父分类借用, 这样一级级直到借到令牌或到达根分类为止。因此, 借用令牌时, 令牌从根向叶子的方向流动; 归还令牌时, 令牌从叶子向根的方向流动。

## Hierarchical Token Bucket (HTB)

### Class structure and Borrowing



注意, 该图中有多个根分类, 每个根分类都可以模拟成一条虚拟线路。

#### 7.1.4. HTB 分类参数

*default*

这是 HTB 排队规则的一个可选参数，默认值为 0，当值为 0 时意味着会绕过所有和 root qdisc 相关联的分类，然后以最大的速度出队任何未分类的流量。

*rate*

这个参数用来设置流量发送的最小期望速率。这个速率可以被当作承诺信息速率(CIR)，或者给某个叶子分类的保证带宽。

*ceil*

这个参数用来设置流量发送的最大期望速率。租借机制将会决定这个参数的实际用处。这个速率可以被称作“突发速率”。

*burst*

这个参数是 rate 桶的大小(参见令牌桶这一节)。HTB 将会在更多令牌到达之前将 burst 个字节的数据包出队。

*cburst*

这个参数是 ceil 桶的大小(参见令牌桶这一节)。HTB 将会更多令牌(ctoken)到达之前将 cburst 个字节的数据包出队。

*quantum*

这个是 HTB 控制租借机制的关键参数。正常情况下，HTB 自己会计算合适的 quantum 值，而不是由用户来设定。对这个值的轻微调整都会对租借和整形造成巨大的影响，因为 HTB 不仅会根据这个值向各个子分类分发流量(速率应高于 rate, 小于 ceil)，还会根据此值输出各个子分类中的数据。

*r2q*

通常, quantum 的值由 HTB 自己计算，用户可以通过此参数设置一个值来帮助 HTB 为某个分类计算一个最优的 quantum 值。

*mtu*

*prio*

### 7.1.5. 规则

下面是一些从 <http://docum.org/> 和 [LARTC mailing list](http://lartc mailing list) 上挑选出来的使用 HTB 的一般性提示。这些只是针对初学者的提示，以便初学者能最大化的掌握 HTB，如果你有 HTB 的实际使用经验，就没必要看这些建议了。

- 整形只会发生在 HTB 的叶子分类中，参见第 7.1.2 小节。
- 由于只会在 HTB 的叶子分类中进行整形，所以所有叶子分类的速率(rate)之和应该不大于父分类速率的最大值(ceil)。在理想情况下，叶子分类的总速率应该和父分类的速率相同，这样父分类总是会有剩余的带宽分配给子分类。

这是 HTB 的一个关键点，我们一再重复。只有叶子分类能对流量进行整形，数据包也只会是在叶子分类中被延时，所有的内部分类(从叶子分类开始一直到根分类的所有分类)只是决定如何进行租借。

- quantum 只会在某个分类的速率大于 rate 小于 ceil 时起作用。
- quantum 应该被设置成 MTU 或更大。即使 quantum 的值被设置得过小，HTB 只要有机会也会出队一个数据包，不过这样就不能准确的计算真实的带宽消耗了。
- 父分类向子分类租出的令牌以 quantum 的倍数增加，所以为了最大粒度、最为迅速的公平分发带宽，quantum 的值应该尽可能的小，但不能小于 MTU。
- token 和 ctoken 只有在叶子分类中才有区别，因为非叶子分类只是向子分类租借令牌。
- HTB 的租借模型应该更为准备的被称为“使用”。

## 7.2. HFSC, 分层公平服务曲线

HFSC 分类排队规则在延时敏感型流量和吞吐敏感型流量之间做了一个平衡。在拥塞状态下，HFSC 排队规则会根据服务区线的定义来决定是否发送延时敏感型数据包。访问 [HFSC Scheduling mit Linux](#) 参阅在 Linux 下使用 HFSC 的德文文档，或者访问 [HFSC Scheduling with Linux](#) 阅读该德文文档的英译版。你也可以通过访问 [A Hierarchical Fair Service Curve Algorithm For Link-Sharing, Real-Time and Priority Services](#) 来获取最初的研究论文。

本节将会在稍后完善。

## 7.3. PRI0, 优先级调度

PRI0 分类排队规则有一个非常简单的准则，当它准备发送数据包时，它会检查第一个分类中有没有数据包，如果有就发送数据包，如果没有就检查下一个分类，直到队列中的最后一个分类。

本节将会在稍后完善。

## 7.4. CBQ, 基于类的队列

CBQ 是流量控制系统的典型排队算法，本节稍后会完善。

# 8. 流量控制的常见规则和方法

## 8.1. Linux 流量控制的一般规则

下面的一些规则可以使流量控制的学习更简单，不管是使用 `tcng` 来配置或 `tc` 来配置，Linux 下的流量控制结构都是相同的。

- 应当只有在路由器出现瓶颈时才去实施流量整形，而且流量整形值应该比最大带宽稍低，这样就能防止从其它路由器过来的大量数据流阻塞路由器，同时也能保证路由器的最大处理能力。
- 只能对设备发送的流量进行整形，当流量被入口接收后，就不能对其进行整形了。该问题的传统解决方案是利用 `ingress policer`。
- 每个接口都有一个默认排队规则，当接口上没有显示指明使用哪一种 `qdisc` 时就会使用默认 `qdisc`。
- 给接口配置一个没有子 `class` 的分类 `qdisc` 是毫无用处的，仅仅消耗了 CPU 资源。
- 每个新创建的 `class` 内部默认都是使用的 `FIFO` 排队规则，可以用其它排队规则来代替 `FIFO`，如果有一个子 `class` 和这个新建的 `class` 相关联，那么 `FIFO` 排队规则将会被删除。
- 可以将分类(`class`)和根排队规则(`root qdisc`)直接关联起来，用来模拟一个虚拟线路。
- `filter` 可以和分类(`class`)或者一个可分类排队规则(`classful qdisc`)关联起来。

## 8.2. 在已知带宽的线路上实施流量控制

在已知带宽的线路上实施流量控制，`HTB` 是一个完美的选择。最内部(最顶端)的分类可以配置成允许最大带宽的流量进入，然后数据流可以被划分到各个子分类(`children class`)中，我们可以为某些子类保留一定带宽，或者让某些特殊的流量优先传送。

## 8.3. 在可变（或未知）带宽的线路上实施流量控制

理论上来说, PRI0 调度器(scheduler)是在可变带宽上实施流量控制的完美方案,因为它是一个工作保留型的排队规则(这也意味着 PRI0 没有整形功能)。在一个带宽未知或有波动的网络中, PRI0 调度器简单的先将最高优先级通道中的数据包包出队,然后再将次优先级队列中的数据包包出队。

## 8.4. 基于流来分享、划分带宽

对网络带宽的竞争有多种形式, SFQ 是解决竞争的最易方式之一。通过使用 SFQ, 流量能被划分为不同的流, 每条流都会被公平的服务。对于行为良好的应用和用户来说, 使用 SFQ 和 ESFQ 将能高效的完成大多数有带宽分享需求的情形。

这些公平排队算法的死穴就是某些行为不端的应用或用户会同时打开太多的连接(例如 emule, eDonkey, Kazaa)。通过创建大量的数据流, 这些应用会占用公平排队算法中的大多数资源。换种方式来说, 就是公平排队算法对那些创建大量数据流的应用毫无办法, 它也不能去惩罚该用户, 你只能采取其它的对应措施。

## 8.5. 基于 IP 来分享、划分带宽

对于网络管理员来说, 这是将带宽划分给用户的一个很好的方式。但不幸的是, 没有简单的方案来实现它, 随着网络中设备的增加它会变得更复杂。

如果要将网络带宽公平的分给 N 个 IP 地址, 那么就需要 N 个分类(classss)。

# 9. 用于 Qos 和流量控制的脚本

## 9.1. wondershaper

待完善, 参见 [wondershaper](#)。

## 9.2. 用于 ADSL 宽带的指南脚本 (myshaper)

待完善, 参见 [myshaper](#)。

## 9.3. htb.init

待完善, 参见 [htb.init](#)。

## 9.4. tcng.init

待完善，参见 [tcng.init](#)。

## 9.5. cbq.init

待完善，参见 [cbq.init](#)。

# 10. 图表

## 10.1. 总图

下图是分类排队规则各组件之间的关系总图（图中是 HTB 排队规则）。点[这里](#)可以查看大图

### 例 11. tcng 配置 HTB 示例

```
/*
 *
 * 可以在下面网址中获取模拟图
 * http://linux-ip.net/traffic-control/htb-class.png
 *
 */

$m_web = trTCM (
    cir 512 kbps, /* 承诺信息速率 */
    cbs 10 kB,    /* 承诺突发信息速率 */
    pir 1024 kbps, /* 峰值信息速率 */
    pbs 10 kB     /* 突发峰值信息速率 */
) ;

dev eth0 {
    egress {

        class ( <$web> ) if tcp_dport == PORT_HTTP &&
__trTCM_green( $m_web );
        class ( <$bulk> ) if tcp_dport == PORT_HTTP &&
__trTCM_yellow( $m_web );
        drop if
__trTCM_red( $m_web );
        class ( <$bulk> ) if tcp_dport == PORT_SSH ;

        htb () { /* root qdisc */
```



```

class ( rate 1544kbps, ceil 1544kbps ) { /* root class */

    $web = class ( rate 512kbps, ceil 512kbps ) { sfq ; } ;
    $bulk = class ( rate 512kbps, ceil 1544kbps ) { sfq ; } ;

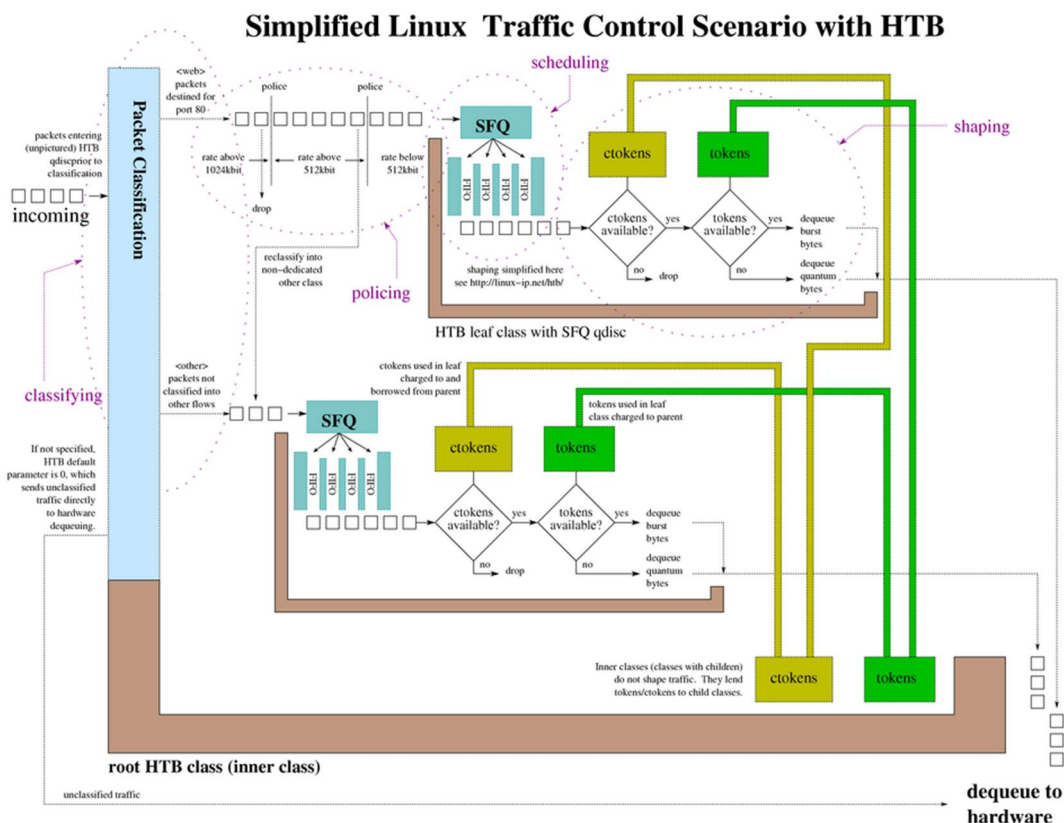
}

}

}

}

```



## 11. 有关流量控制的资源链接

本部分列出了一些有关流量控制和流量控制软件的文档链接。每个链接下面都有一个该链接内容的简单描述。

- [HTB site](#), [HTB user guide](#) and [HTB theory](#) (*Martin "devik" Devera*)

分层令牌桶，也就是 [HTB](#)，是一个分类排队规则。它被广泛的使用和支持，其拥有完善的用户使用手册和网站 [Stef Coene's site](#)。

- [General Quality of Service docs](#) (*Leonardo Balliache*)

在这个网站上有许多易懂的介绍性文档，特别是有很多优秀的概述文章。

- [tcng \(Traffic Control Next Generation\)](#) and [tcng manual](#) (*Werner Almesberger*)

tcng 软件包包含了一个语言，以及一组用来创建和测试流量控制结构的工具。除了能以 tc 命令的方式输出外，它还能够为非 Linux 应用提供输出。在本文档中被忽视掉的关键部分是 tscim 流量控制模拟器

tcng 软件包提供的帮助手册已经通过 latex2html 工具转换成了 HTML 格式，该版本将通过 TeX 文档来发布。

- [iproute2](#) and [iproute2 manual](#) (*Alexey Kuznetsov*)

这里是 iproute2 套件的源代码，包含了必不可少的 tc 二进制可执行命令。注意，iproute2-2.4.7-now-ss020116-try.tar.gz 并不支持 HTB，需要从 [HTB](#) 上获取一个支持 HTB 的补丁。

这里有该工具套件中所有工具的帮助文档，尽管有关 tc 的文档可能不完整。当对帮助文档中的内容有疑惑时，最好去 LARTC HOWTO 寻找相关答案。

- [Documentation, graphs, scripts and guidelines to traffic control under Linux](#) (*Stef Coene*)

Stef Coene 已经收集了在 Linux 下使用 QoS 的一些脚本、建议、统计数据 and 测试结构。在 Stef 的站点上还有一些非常实用的图表和指南。

- [LARTC HOWTO](#) (*bert hubert, et. al.*)

Linux 高级路由和流量控制指南 (Linux Advanced Routing and Traffic Control HOWTO) 是讲解 Linux 下复杂流量控制技术的有用文档。流量控制入门指南 (Traffic Control Introduction HOWTO) 应该向读者提供足够的应用背景及流量控制思想，LARTC HOWTO 是读者查找流量控制信息的好地方。

- [Guide to IP Networking with Linux](#) (*Martin A. Brown*)

虽然和流量控制没有直接联系，但这个站点上有一些和 Linux IP 层有关的文章和常见资源。

- [Werner Almesberger's Papers](#)

Werner Almesberger 是 Linux 流量控制的主要开发者和拥护者之一 (他也是 tcng 的作者)。他的文章《Linux 流量控制-应用概述》，是描述 Linux

内核中流量控制架构的一份重要文档，你可以获取该文档的 [PDF](#) 或 [PS](#) 版。

- [Linux DiffServ project](#)

下面这段内容是从 DiffServ 网站主页上毫不犹豫的剪切过来的...

差分服务 (简称: **Diffserv**) 为网络流量提供不同类或不同等级的服务。差分服务的一个关键特征是: 流可以在网络中聚合在一起形成一条聚合流, 因此核心路由器只需要识别少量的聚合流就行了, 即使一条聚合流中可能包含成千上万条独立的流。