做了一些关于tc red htb tbf的调研如下。

Linux流量控制控发不控收,所以只能对产生瓶颈网卡处的发包速率进行控制..而网络瓶颈分析亦为Linux网络流控的第一步。

SHAPING（流量整形也叫限制）用于处理向外的流量，而POLICIING(流量监管也叫策略)用于处理接收到的数据。

之前见到诸位在邮件里有讨论到底是出还是收，故在此做以上补充。（如我理解错请忽略）

一些关于linux tc的简介：

<http://blog.csdn.net/qinyushuang/article/details/46611709>

涉及到linux与网卡，tc命令添加htb队列，浅显易懂，不需要可略

<http://blog.csdn.net/qinyushuang/article/details/46611923>

<http://www.cnblogs.com/endsock/archive/2011/12/09/2281519.html>

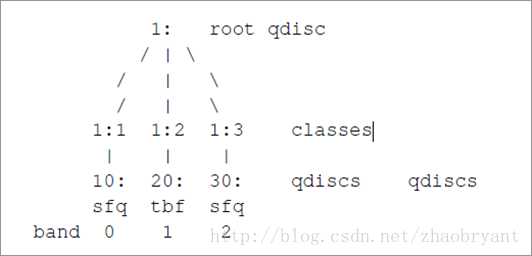
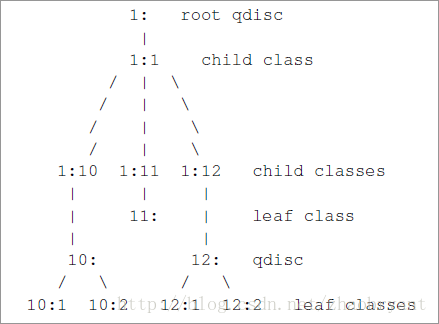
<http://blog.csdn.net/ysdaniel/article/details/7905879>

<http://www.php-oa.com/2009/06/23/linux_tc.html>

这些有部分是重复内容，供参考。

**流量的处理由三种对象控制，它们是：qdisc(排队规则)、class(类别)和filter(过滤器)。**  
  
QDISC(排队规则)  
QDisc(排队规则)是queueing discipline的简写，它是理解流量控制(traffic control)的基础。无论何时，内核如果需要通过某个网络接口发送数据包，它都需要按照为这个接口配置的qdisc(排队规则)把数据包加入队列。然后，内核会尽可能多地从qdisc里面取出数据包，把它们交给网络适配器驱动模块。   
最简单的QDisc是pfifo(packet first in first out)它不对进入的数据包做任何的处理，数据包采用先入先出的方式通过队列。不过，它会保存网络接口一时无法处理的数据包。   
  
CLASS(类)  
某些QDisc(排队规则)可以包含一些类别，不同的类别中可以包含更深入的QDisc(排队规则)，通过这些细分的QDisc还可以为进入的队列的数据包排队。通过设置各种类别数据包的离队次序，QDisc可以为设置网络数据流量的优先级。  
  
FILTER(过滤器)  
filter(过滤器)用于为数据包分类，决定它们按照何种QDisc进入队列。无论何时数据包进入一个划分子类的类别中，都需要进行分类。分类的方法可以有多种，使用fileter(过滤器)就是其中之一。使用filter(过滤器)分类时，内核会调用附属于这个类(class)的所有过滤器，直到返回一个判决。如果没有判决返回，就作进一步的处理，而处理方式和QDISC有关。   
需要注意的是，filter(过滤器)是在QDisc内部，它们不能作为主体。 

总结：tc整个结构是树，每个网卡一般会配置一个根队列规则，根队列规则可以向下分类（分出来的类可以在子节点上使用无分类队列规则，如果根队列规则使用不分类的就不能再往下有分支），不同的队列规则最后对应不同的路由，过滤器即为分类报文服务，将不同的报文分到不同的类，如果该类还有过滤器就再继续过滤分给子类。



结构可以是图示各种。

Tc就是linux的一个指令或是应用，用来对网卡添加规则之类的，达到限制网卡速率等目的。

red random early detection 随机早期检测 不可分类

大概介绍：<http://man7.org/linux/man-pages/man8/tc-red.8.html>

<https://linux.die.net/man/8/tc-red> 这两个是不同网站的会略有不同

相关源码：<http://lxr.free-electrons.com/source/net/sched/sch_red.c>

Htb hierarchical token bucket htb是可分类Qdisc

大概介绍：<http://man7.org/linux/man-pages/man8/tc-htb.8.html>

<https://linux.die.net/man/8/tc-htb>

相关源码：<http://lxr.free-electrons.com/source/net/sched/sch_htb.c>

Tbf token bucket filter令牌桶过滤器 man7上是classful（01年的），有的网页（包括die.net）写的是无分类，可能是版本不同。HTB和TBF核心算法应该是差不多的，我也看到有说法是HTB比TBF多在引入了classes。Linux cross reference上tbf的开头注释部分有写with classful tbf，blablabla。在linux上用tc实际配置中尝试了下tbf是classless。

大概介绍：<http://man7.org/linux/man-pages/man8/tc-tbf.8.html>

<https://linux.die.net/man/8/tc-tbf>

相关源码：<http://lxr.free-electrons.com/source/net/sched/sch_tbf.c>

从算法上看htb和tbf是基本一致的，主要区别应该就是htb可以向下分类，tbf不可以。如果是这样的话，那tbf就还是先入先出不过有限速，htb有分类的话子类间共享htb的带宽，即如果有某一子类带宽不够时可以借用父类下空闲带宽。（可设置其限值）

您在邮件中提到的4个问题

Maybe you can teach us:

1)is it possible to let HTB call RED? What is the relationship between HTB and RED?

2)you can learn how to configure tc red and measure the latency.

3)find how to get the instantaneous queue length in red.

4)finally, we may need to add the logic in sch\_htb.c, but sch\_red.c seems to be easier.

1. 确切的说HTB和RED的关系不是调用，上文说了HTB和RED都是队列规则，RED是无类规则，HTB是有类规则，所以RED可以作为HTB的叶规则来使用。

比如如下代码：

root@node-0:/users/Hunter# tc qdisc del dev eth2 root handle 1: htb

root@node-0:/users/Hunter# tc qdisc add dev eth2 root handle 1: htb default 11

root@node-0:/users/Hunter# tc class add dev eth2 parent 1:0 classid 1:1 htb rate 20000mbit

root@node-0:/users/Hunter# tc qdisc add dev eth2 parent 1:1 handle 10: red avpkt 1000 min 1000000 max 2000000 limit 3000000 burst 1000

三行代码，第一行创建根规则，第二行创建根类，第三行创建叶规则（RED隶属在HTB之下）

1. 配置tc规则请先查看tc-red那个网页，看下red的参数，然后照着tc的正常配置方法配置就可以。

我没有设OVS的限速直接测了延迟，有需要之后再改，先做试验性。

未设置tc规则时，sockperf数据：

sockperf: Summary: Latency is 114.898 usec

sockperf: Total 3000 observations; each percentile contains 30.00 observations

sockperf: ---> <MAX> observation = 137.956

sockperf: ---> percentile 99.999 = 137.956

sockperf: ---> percentile 99.990 = 137.956

sockperf: ---> percentile 99.900 = 129.279

sockperf: ---> percentile 99.000 = 125.440

sockperf: ---> percentile 90.000 = 117.852

sockperf: ---> percentile 75.000 = 115.082

sockperf: ---> percentile 50.000 = 114.448

sockperf: ---> percentile 25.000 = 113.976

sockperf: ---> <MIN> observation = 86.867

对node0（客户端）设置tc规则后（就按照上面的参数随便设的，这个之后按需求改）：

sockperf: Summary: Latency is 1143.553 usec

sockperf: Total 3000 observations; each percentile contains 30.00 observations

sockperf: ---> <MAX> observation = 1350.822

sockperf: ---> percentile 99.999 = 1350.822

sockperf: ---> percentile 99.990 = 1350.822

sockperf: ---> percentile 99.900 = 1280.910

sockperf: ---> percentile 99.000 = 1248.499

sockperf: ---> percentile 90.000 = 1193.017

sockperf: ---> percentile 75.000 = 1173.543

sockperf: ---> percentile 50.000 = 1150.281

sockperf: ---> percentile 25.000 = 1124.382

sockperf: ---> <MIN> observation = 979.028

这里有个问题，我设了之后好像并没有影响到iperf，还是按照9.4Gb/s速率发送，但是sockperf测的延迟确实明显变大了（删除了tc规则后又测了一次回归第一次测试的水平），这个有待研究。

对node1（服务器端）设置相同的tc规则：

sockperf: Summary: Latency is 115.596 usec

sockperf: Total 3000 observations; each percentile contains 30.00 observations

sockperf: ---> <MAX> observation = 222.724

sockperf: ---> percentile 99.999 = 222.724

sockperf: ---> percentile 99.990 = 222.724

sockperf: ---> percentile 99.900 = 148.108

sockperf: ---> percentile 99.000 = 127.407

sockperf: ---> percentile 90.000 = 118.335

sockperf: ---> percentile 75.000 = 115.529

sockperf: ---> percentile 50.000 = 114.868

sockperf: ---> percentile 25.000 = 114.403

sockperf: ---> <MIN> observation = 85.591

这个和第一次没配置的时候测的结果是一个水平的，不知道具体是什么原因。这里可以再研究下。

1. 可以参考我在附件中放的pdf是对red代码的一点标注，便于理解。这里吹一波linux cross reference（就老师给的那个red源码的网站），可以直接找函数或结构体的定义和引用，非常方便，省了不少事。

您给的那个网页是Linux/net/sched/sch\_red.c，里面引用了库#include <net/red.h>，在这个库里有如何计算qavg（平均队列长度），在您给的那个网页sch\_red.c里是直接使用了计算平均队列的函数，所以如果只想找瞬时队列长度的话，不太麻烦，大部分代码都没用。

[http://lxr.free-electrons.com/source/include/net/red.h#L294](http://lxr.free-electrons.com/source/include/net/red.h%23L294)这是red.h。

sch\_red.c中第66行的函数用了参数qstats.backlog，我一路查它数据结构备注是@backlog: backlog size of queue，应该就是瞬时队列长度。具体的平均队列长度算法和计算则在red.h中，主要从230行开始往下。

[http://lxr.free-electrons.com/source/net/sched/sch\_red.c sch\_red.c](http://lxr.free-electrons.com/source/net/sched/sch_red.c%20%20%20sch_red.c)

1. 这个我理解您的意思就是修改red的算法，但是具体想怎么修改我还不太理解，或者说你们paper的核心是什么我还不清楚。目前理解是从平均队列长度标ecn变成瞬时队列长度标ecn？

<http://cxw06023273.iteye.com/blog/867326>

此外我还参考了上面这个链接，也是说red代码的，虽然有出入但我觉得可参考性很大。