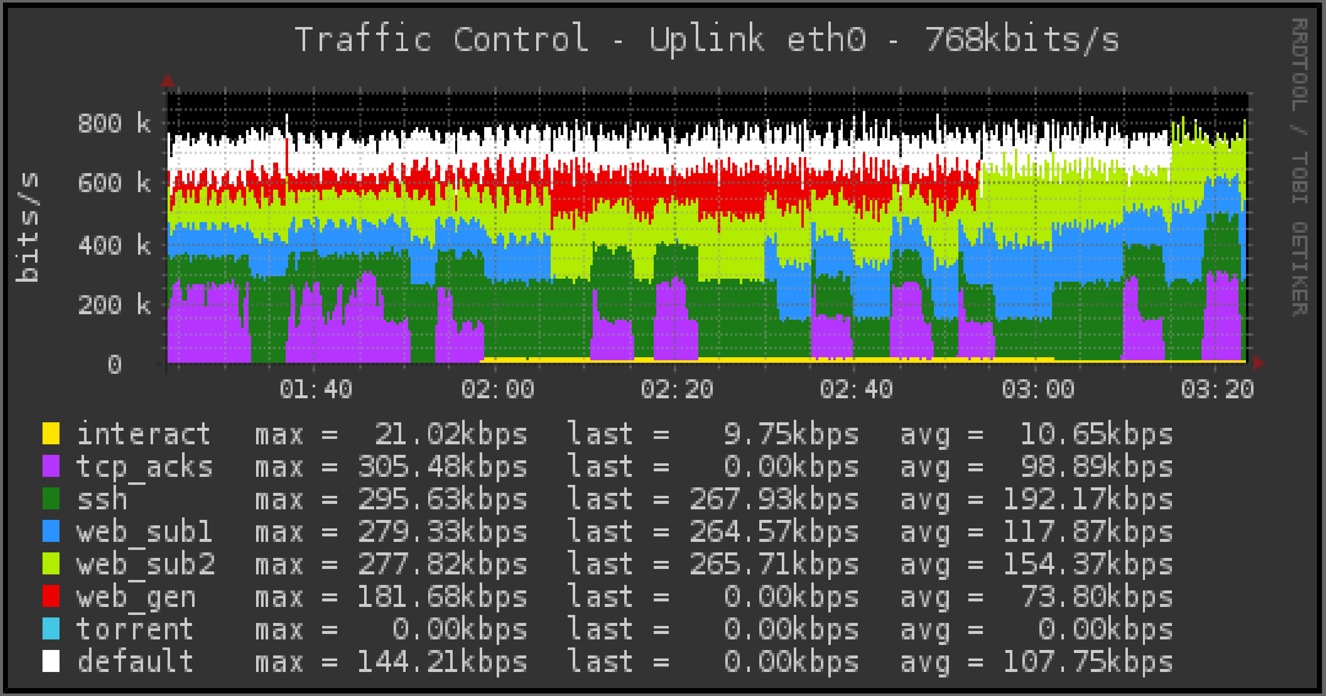
Journey to the Center of the Linux Kernel: Traffic Control, the QoS

- A close look at how Linux transmits packets over the network
- PLUG – Nov.2011
- Julien Vehent - http://jve.linuxwall.info



```
🟨 interact   max =  21.02kbps  last =   9.75kbps  avg =  10.65kbps
🟪 tcp_acks   max = 305.48kbps  last =   0.00kbps  avg =  98.89kbps
🟩 ssh        max = 295.63kbps  last = 267.93kbps  avg = 192.17kbps
🟦 web_sub1   max = 279.33kbps  last = 264.57kbps  avg = 117.87kbps
🟩 web_sub2   max = 277.82kbps  last = 265.71kbps  avg = 154.37kbps
🟥 web_gen    max = 181.68kbps  last =   0.00kbps  avg =  73.80kbps
🟦 torrent    max =   0.00kbps  last =   0.00kbps  avg =   0.00kbps
⬜ default    max = 144.21kbps  last =   0.00kbps  avg = 107.75kbps
```

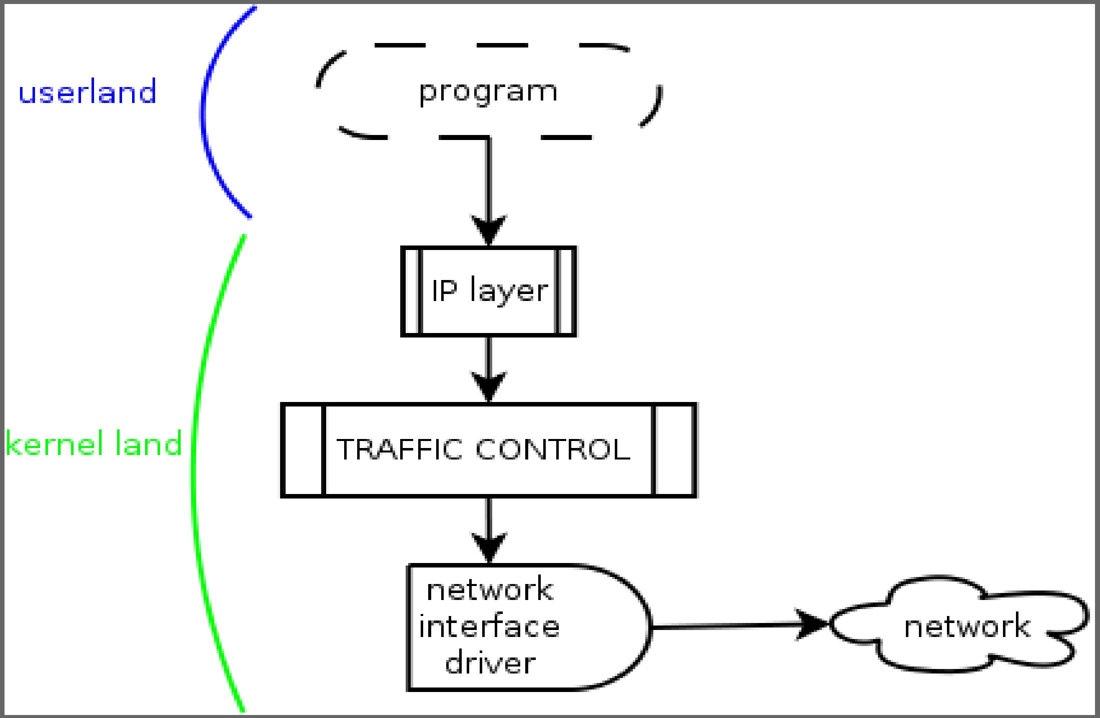
Why should you care ?

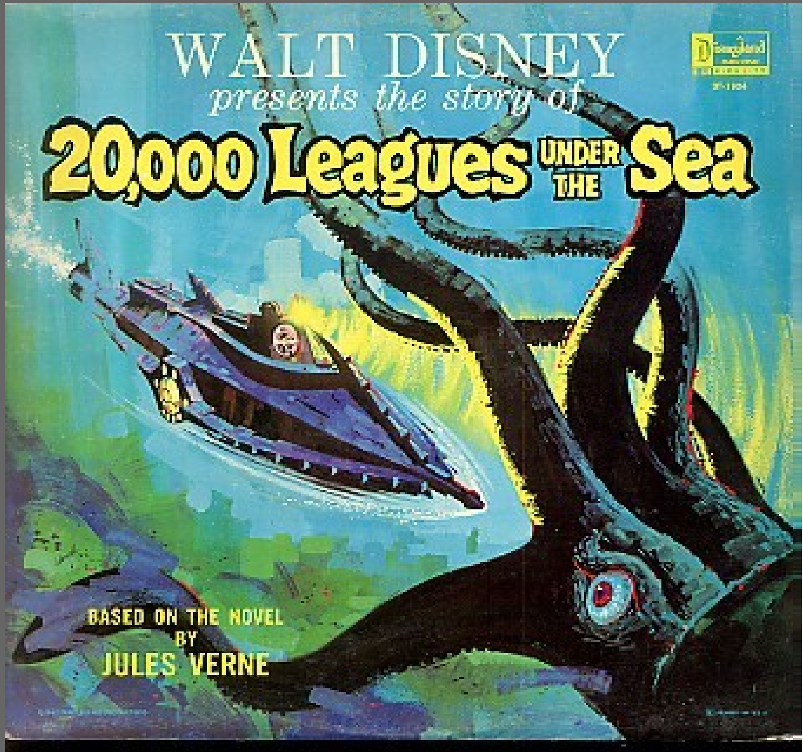
- 1 to 2Mbits per second of uplink on most residential connection
- Can't share it properly without QoS.
  - Netflix + download + skype = angry family
- Your buffers are most likely destroying your latency (bufferbloat)
- But most importantly :
  - Because it's cool stuff ! Really.

Where are we going exactly ?

- All the way at the bottom of the Networking layer.
- TC transmits (egress) packets to the NIC
- In the kernel source: /net/sched/

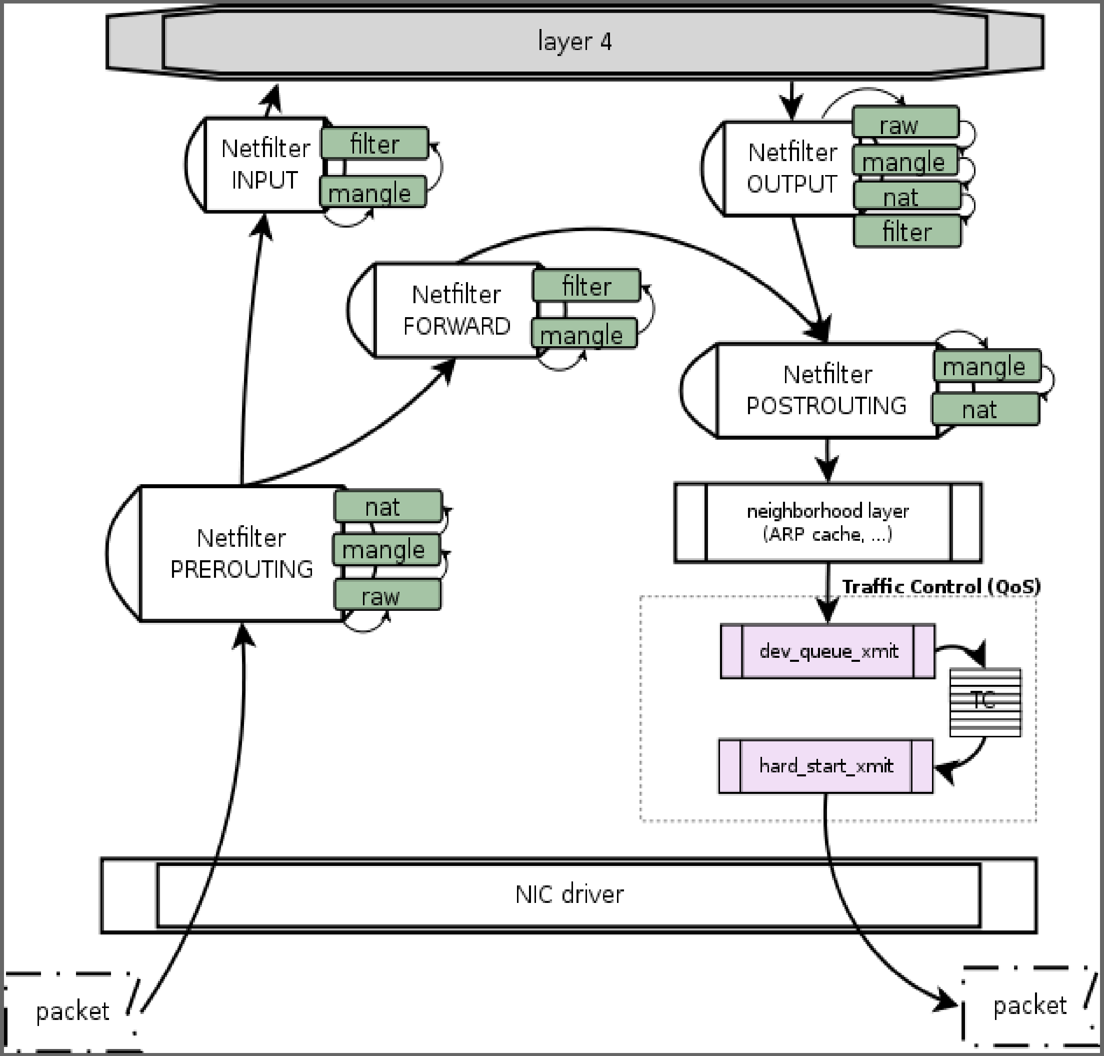
  http://git.kernel.org/?p=linux/kernel/git/next/linux-next.git;a=tree;f=net/sched;hb=HEAD





Packets Transmission

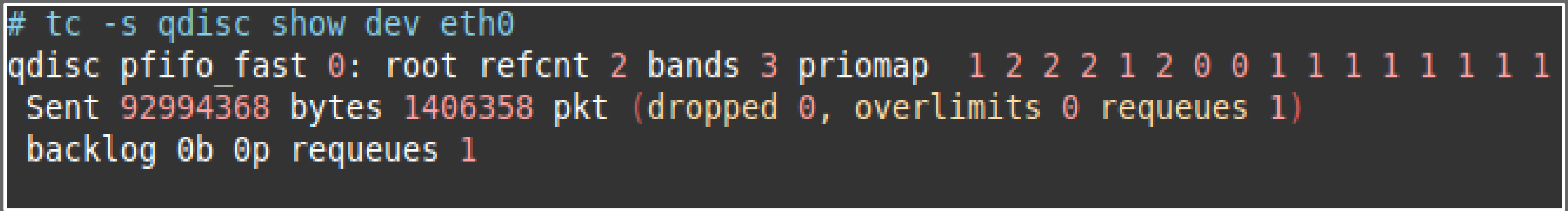
- TC manipulates the sk_buff structure of a packet



```
struct sk_buff {
[...]
  sk_buff_data_t
        transport_header;
  sk_buff_data_t
        network_header;
  sk_buff_data_t
        mac_header;
[...]
union {
  __u32 mark;
[...]
```

TC is a packet scheduler

- TC maintains a list of packets elligible for transmission
- Applies a Queuing Discipline (qdisc) to select the next packet to send
- Transparent when the link is not fully used
- some QDiscs: SFQ, TBF, HTB, HFSC, RED, …
- The default qdisc everybody uses it PFIFO_FAST

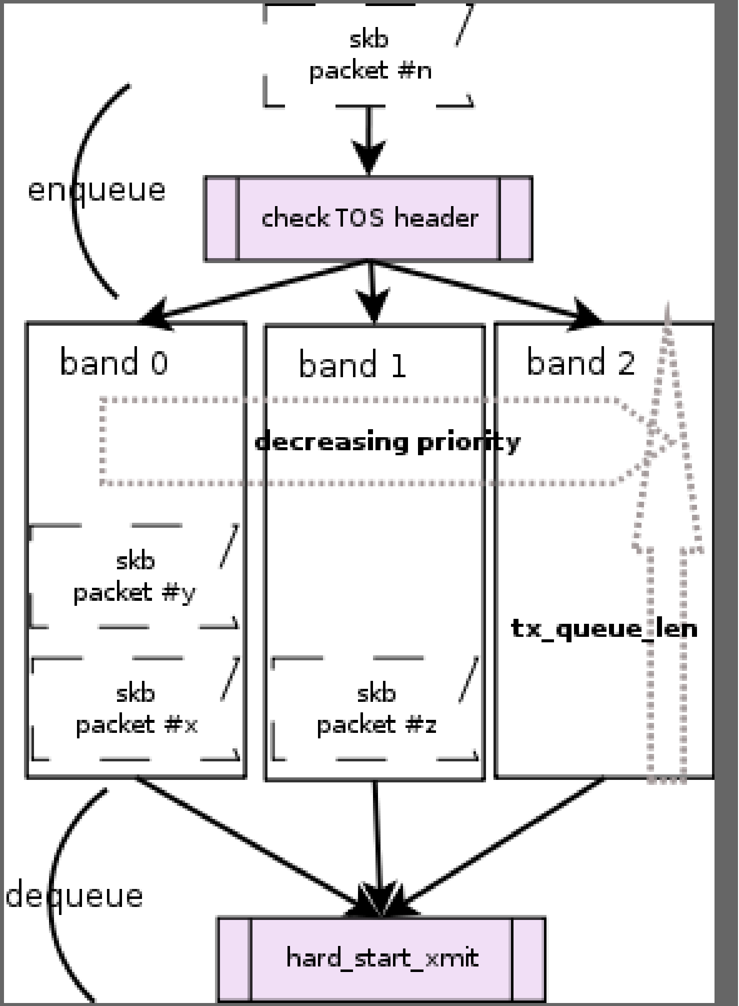


```
# tc -s qdisc show dev eth0
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 92994368 bytes 1406358 pkt (dropped 0, overlimits 0 requeues 1)
 backlog 0b 0p requeues 1
```

PFIFO_FAST and the buffer problem

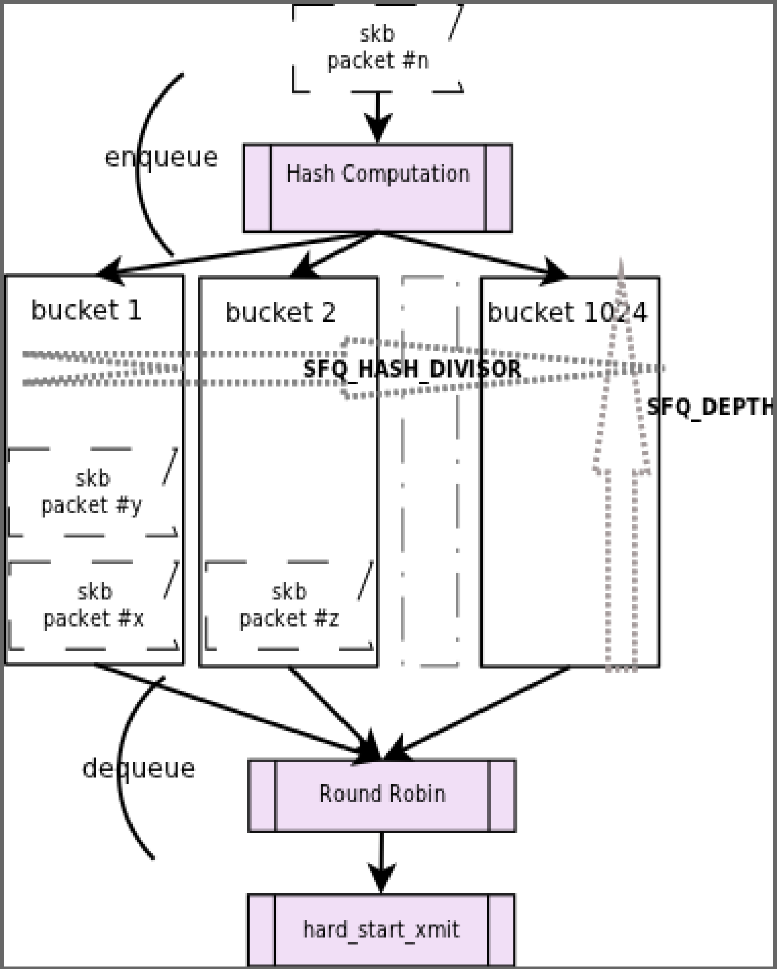
- Simple FIFO queue with 3 bands
- ToS IP headers determines the band
- Default queue size of 1000
- Way too large for small connections
    - BUFFERBLOAT !

```
eth0      Link encap:Ethernet  HWaddr 08:00:27:f1:34:0e

          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0

          [...]

          collisions:0 txqueuelen:1000
```



SFQ, fair sharing of the bandwidth

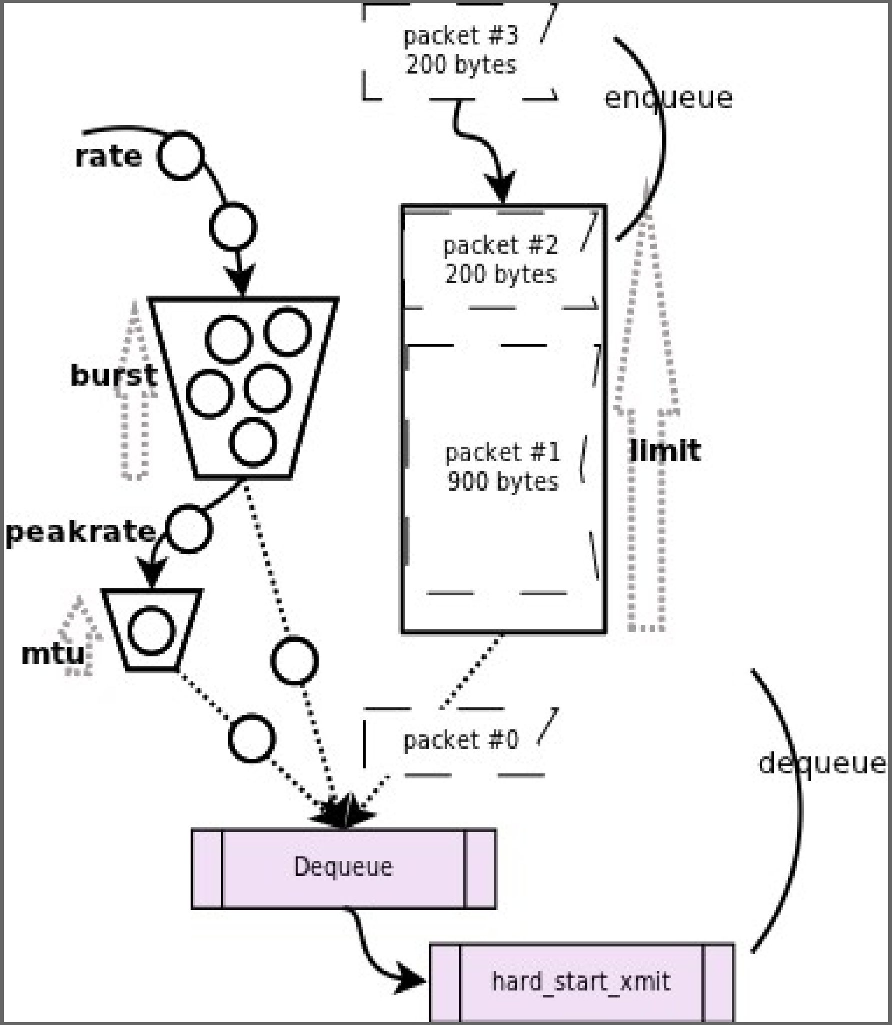
- Send a packet to one of the 1024 buckets based on a hash



- Hash is computed on the packet headers
- Buckets are emptied in a Round Robin fashion
- But....
  - SFQ breaks the sequencing of packets. Not fun for VoIP and Syslog.

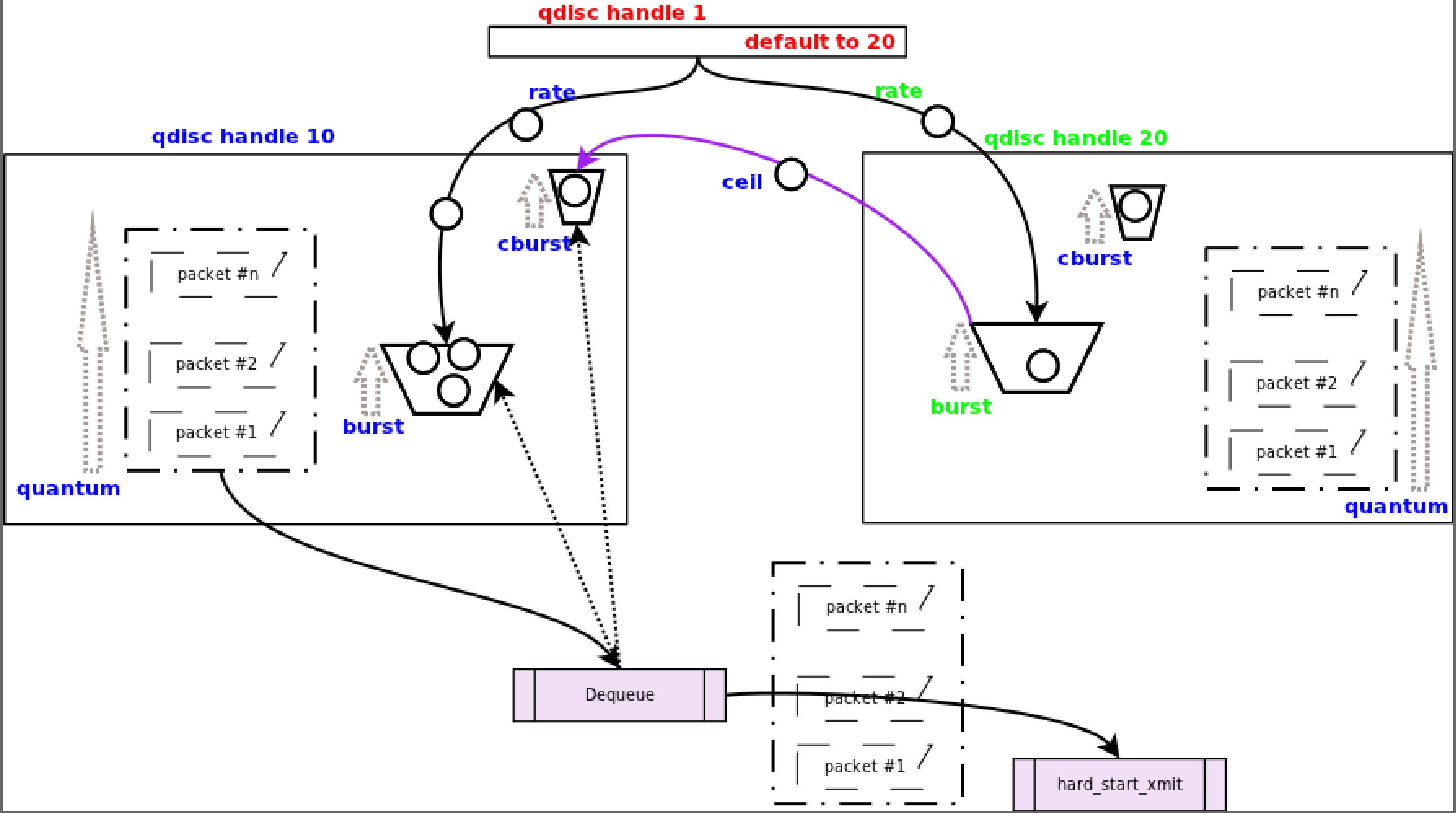
TBF, basic bandwidth accounting

- TBF sends a token into a bucket at a regular frequency
- Entering packets consume the tokens
- Bucket replenishment rate represents the allocated bandwidth
- Bursts are controlled using a second, smaller, bucket



HTB, TBF with priorities

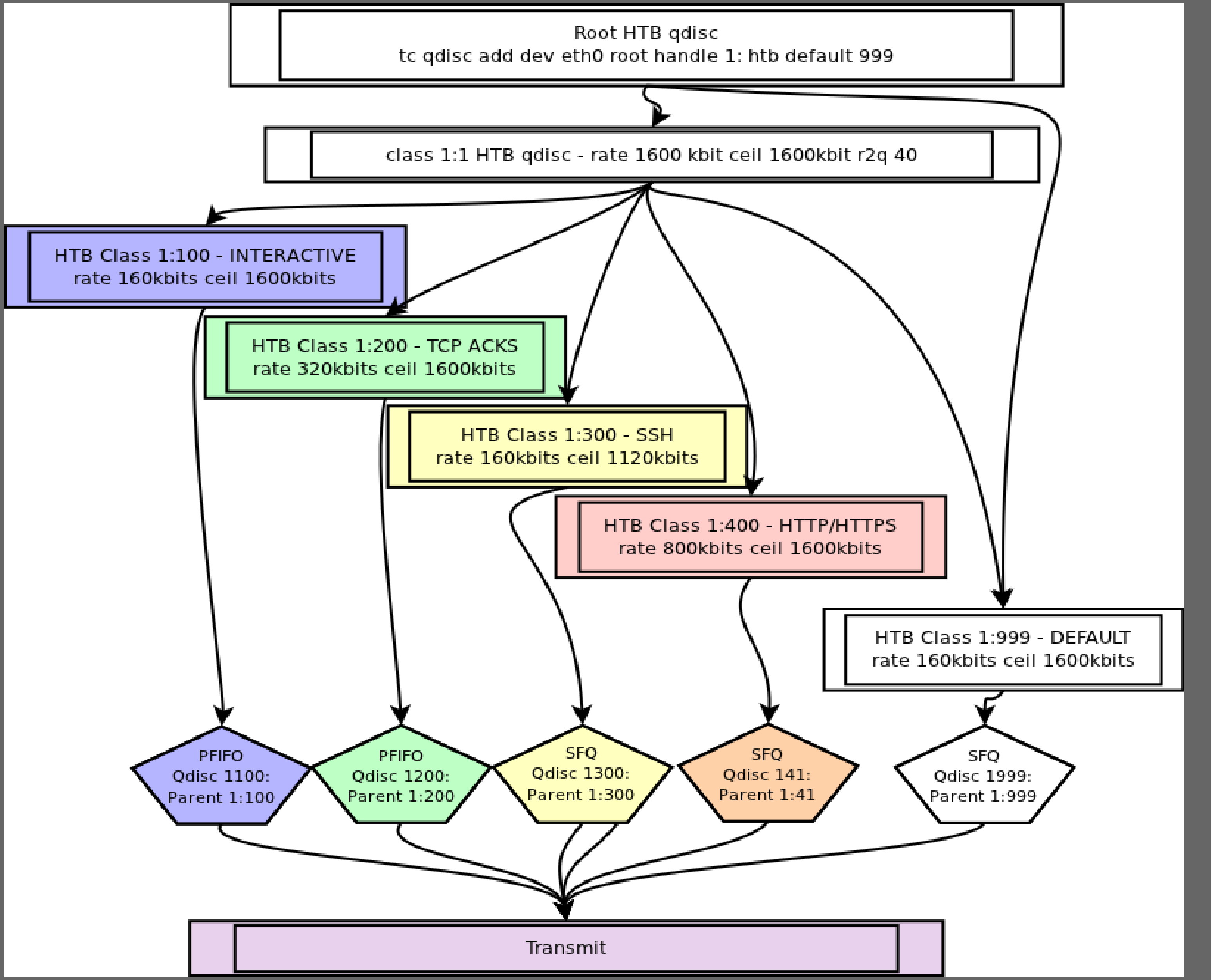
- Similar than TBF, but with several parallel classes that can borrow from the bucket of their neighbors



Journey to the Center of the Linux Kernel: Traffic Control, the QoS

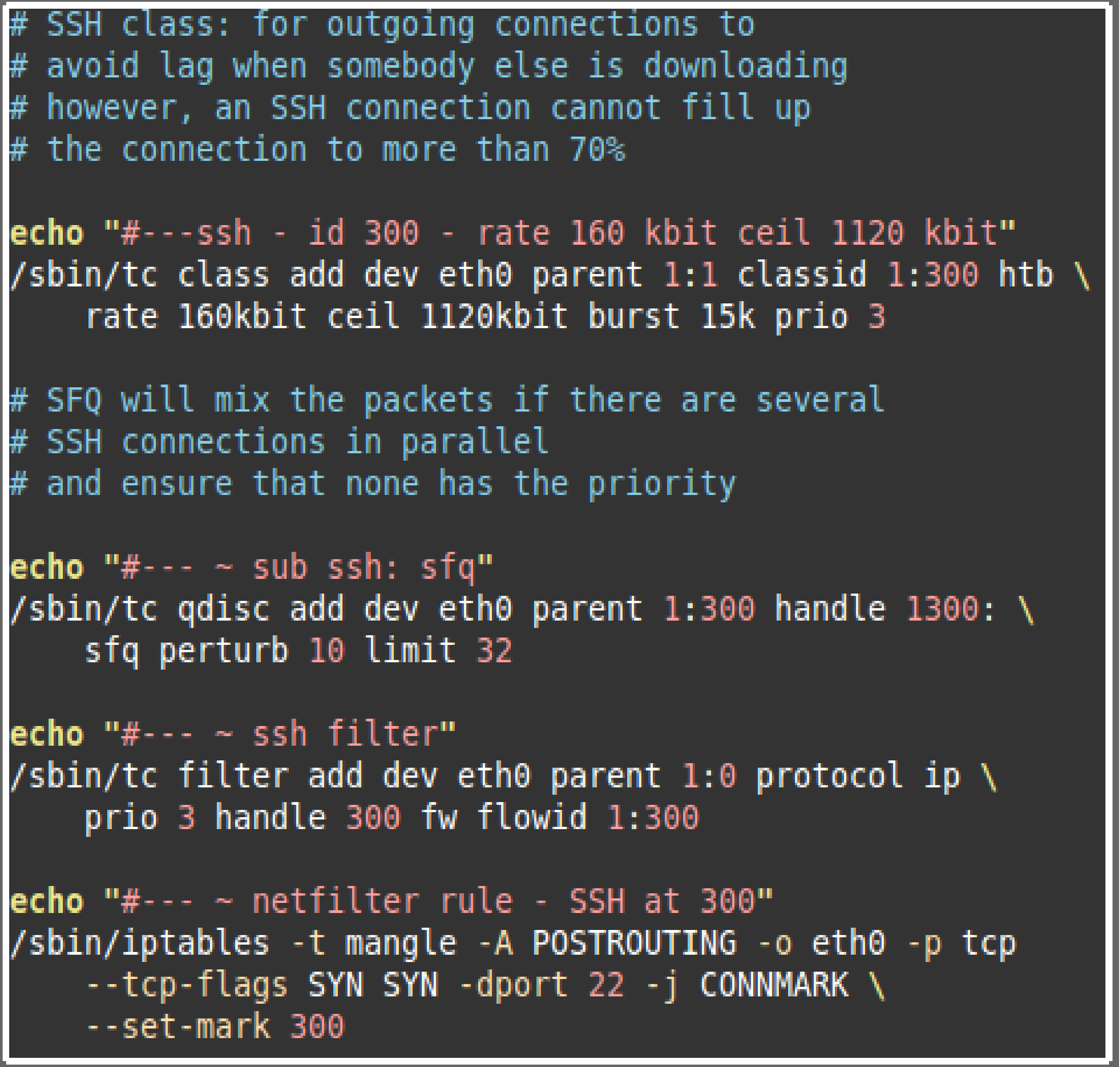
using HTB to build a QoS policy

- 5 classes
- Dedicated rate
- Can borrow
- Use either PFIFO_FAST or SFQ for transmit


Root HTB qdisc
tc qdisc add dev eth0 root handle 1: htb default 999
class 1:1 HTB qdisc - rate 1600 kbit ceil 1600kbit r2q 40
HTB Class 1:100 - INTERACTIVE
rate 160kbits ceil 1600kbits
HTB Class 1:200 - TCP ACKS
rate 320kbits ceil 1600kbits
HTB Class 1:300 - SSH
rate 160kbits ceil 1120kbits
HTB Class 1:400 - HTTP/HTTPS
rate 800kbits ceil 1600kbits
HTB Class 1:999 - DEFAULT
rate 160kbits ceil 1600kbits
PFIFO
Qdisc 1100:
Parent 1:100
PFIFO
Qdisc 1200:
Parent 1:200
SFQ
Qdisc 1300:
Parent 1:300
SFQ
Qdisc 141:
Parent 1:41
SFQ
Qdisc 1999:
Parent 1:999
Transmit

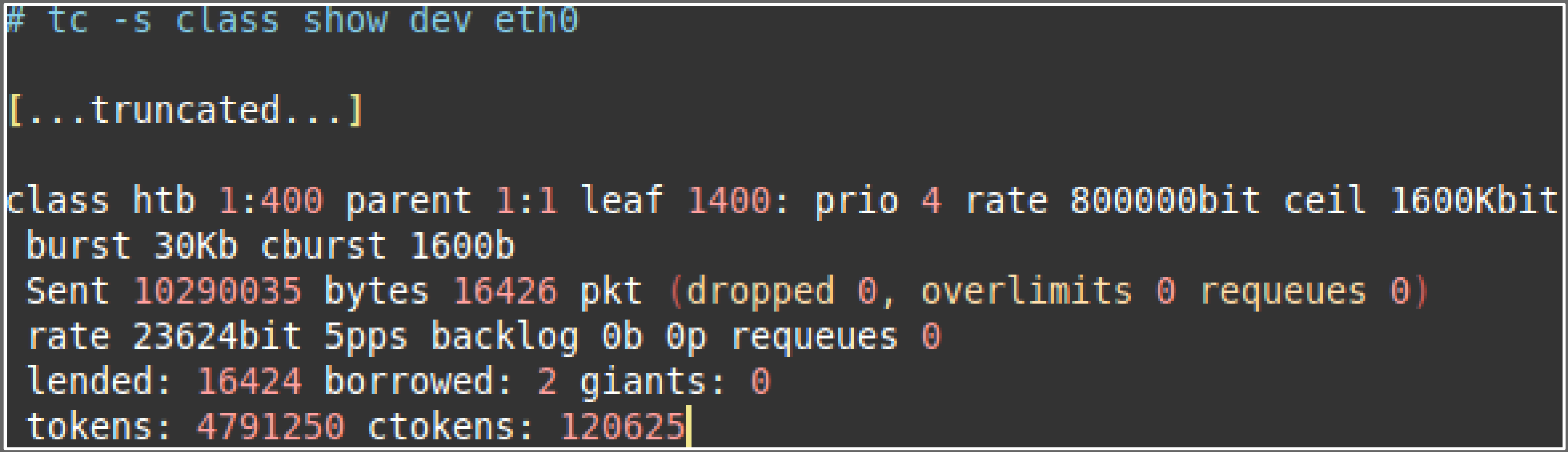
the SSH branch of the HTB tree

- First: the class, with rate and ceil
- Then: the SFQtransmit queue
- And the filter, to route packets to the class based on their netfilter mark
- Finally: the iptables rule to mark packets



```
# SSH class: for outgoing connections to
# avoid lag when somebody else is downloading
# however, an SSH connection cannot fill up
# the connection to more than 70%

echo "#---ssh - id 300 - rate 160 kbit ceil 1120 kbit"
/sbin/tc class add dev eth0 parent 1:1 classid 1:300 htb \
    rate 160kbit ceil 1120kbit burst 15k prio 3

# SFQ will mix the packets if there are several
# SSH connections in parallel
# and ensure that none has the priority

echo "#--- ~ sub ssh: sfq"
/sbin/tc qdisc add dev eth0 parent 1:300 handle 1300: \
    sfq perturb 10 limit 32

echo "#--- ~ ssh filter"
/sbin/tc filter add dev eth0 parent 1:0 protocol ip \
    prio 3 handle 300 fw flowid 1:300

echo "#--- ~ netfilter rule - SSH at 300"
/sbin/iptables -t mangle -A POSTROUTING -o eth0 -p tcp
    --tcp-flags SYN SYN -dport 22 -j CONNMARK \
    --set-mark 300
```

Visualizing TC statistics



```
# tc -s class show dev eth0

[...truncated...]

class htb 1:400 parent 1:1 leaf 1400: prio 4 rate 800000bit ceil 1600Kbit
 burst 30Kb cburst 1600b
 Sent 10290035 bytes 16426 pkt (dropped 0, overlimits 0 requeues 0)
 rate 23624bit 5pps backlog 0b 0p requeues 0
 lended: 16424 borrowed: 2 giants: 0
 tokens: 4791250 ctokens: 120625
```

- Statistics show how much data was sent, but also how many tokens where borrowed, lended, and how many are available now
- Connections/marks are in /proc/net/ip_conntrack

```
tcp      6 299 ESTABLISHED src=68.80.50.84 dst=88.191.125.180 sport=32779 dport=22
packets=455994 bytes=682419052 src=88.191.125.180 dst=68.80.50.84 sport=22
dport=32779 packets=274817 bytes=15267765 [ASSURED] mark=999 secmark=0 use=2
```

Tips and Tricks

- Some tricks:
  - As much as possible, try to mix packets from differents connections using SFQ. It prevents a single connection from filling up the transmit buffer.
  - Put long living connections in a separate class, using the amount of bytes transmitted in the connection and the connbytes netfilter module
  - Remember to divides your rates in the queues to match the global rate of the root

Check out the source code

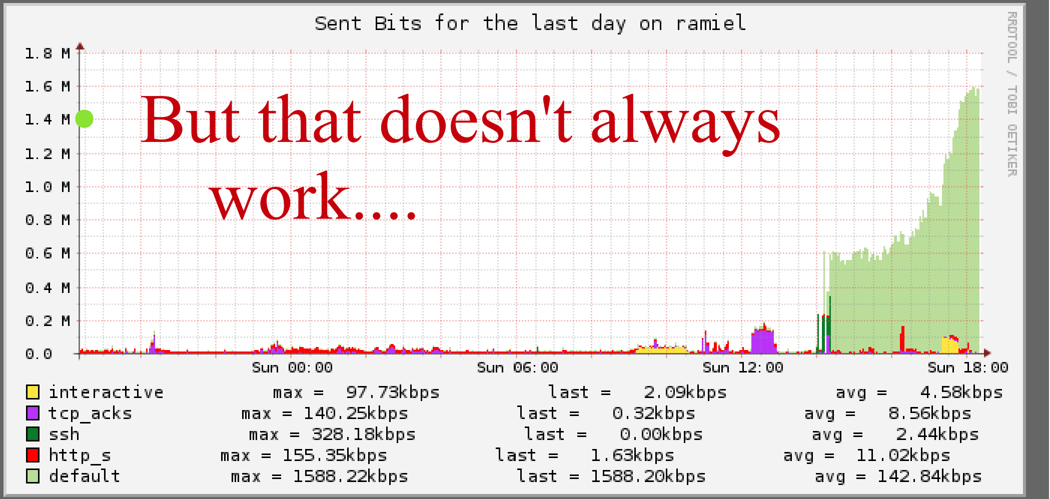
- The main article :
- http://wiki.linuxwall.info/doku.php/en:ressources:dossiers:networking:traffic_control
- Source on GitHub :
- https://github.com/jvehent/lnw-tc

- Come Join Us ! http://www.aweber.com/careers.htm

Comcast Powerboost

- Applies to the first *x* MB of an upload, like a burst
- Make a big file, split it in chunks and upload in parallel
- Test: 100MB file (ideal at 200KB/s: 8:32s, real 175.6KB/s: 09:43s)

```
$ dd if=/dev/urandom of=100MB.dat bs=1M count=100

$ tar -cf - 100MB.dat |split -b 3m - splitvol.tar

$ date;ls splitvol.tar*|xargs -P5 -n 1 -I{} scp '{}' julien@sachiel.linuxwall.info: ;date

Tue Dec 13 17:23:07 EST 2011

splitvol.taraa   100% 3072KB 180.7KB/s   00:17

splitvol.tarad   100% 3072KB 113.8KB/s   00:27

[…...]

splitvol.tarbf  100% 3072KB 109.7KB/s   00:28

Tue Dec 13 17:32:06 EST 2011
```



- But that doesn't always work....