

Quality of Service in IP Networks

by

Harald Welte <laforge@conectiva.com>

Contents

- Definition of QoS
- Why QoS
- IP Networks are not designed for QoS
- How to do the impossible
- What can Linux based systems help
- Advanced Concepts (DiffServ, IntServ, RSVP, ...)
- References / Further Reading

Definiton of QoS

- Provide Service Differentiation
- Performance Assurance by
 - **Bandwith guarantees**
 - ▷ for streaming multimedia traffic
 - ▷ prioritizing certain important applications
 - **Latency guarantees**
 - ▷ for voice over IP
 - ▷ for interactive character-oriented applications (ssh,telnet)
 - **Packet-loss guarantees**
 - ▷ for unreliable layer-4 protocols
 - ▷ to avoid retransmits

Why QoS

- Decide how and who available bandwidth is divided
- Limit available bandwidth for certain users / applications
- Guarantee bandwidth for certain users / applications
- Divide bandwidth more equally between users / applications

IP networks not designed for QoS

Properties of IP-based networks:

- offer a "best-effort" service
- make NO guarantees about
 - bandwidth
 - latency
 - packet loss
- provide a non-reliable packet transport

Conclusion: IP networks are not suitable for QoS

How to do the Impossible

As IP Networks including Hardware (Routers, ...) are widely deployed, all QoS efforts have to layer on top of the existing technology.

- There's no real solution to control latency
 - latency widely dependent on routing, which may be dynamic
- There's no real solution to control packet loss
 - packet loss may occur on any intermediate router
- But we can control bandwidth usage!
 - The sender can limit bandwidth for outgoing streams
 - Intermediate routers BEFORE a bottleneck can control bandwidth usage

What can Linux systems do?

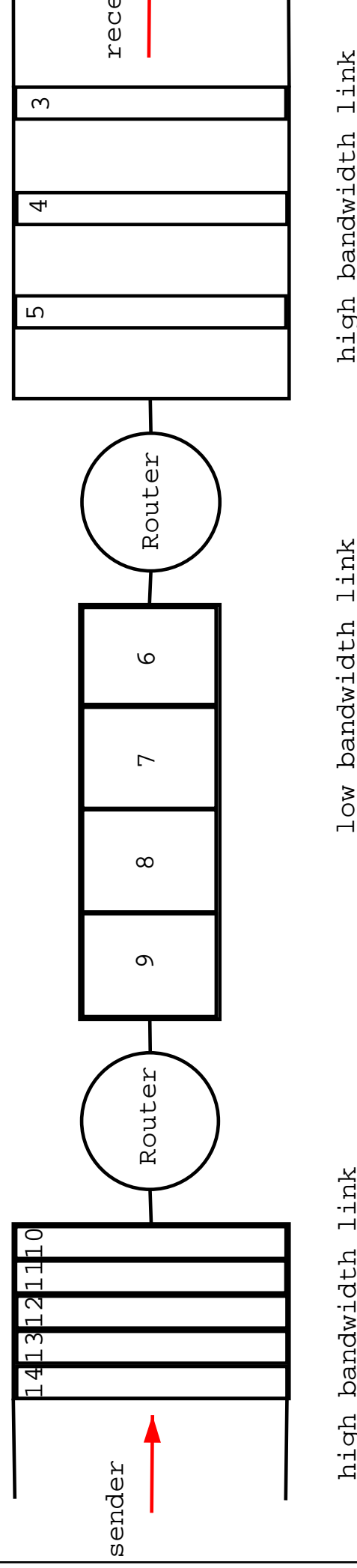
- Bandwidth limiting at the sender application
 - not many applications support it
 - server often out of control (on Internet, ...)
 - server doesn't know what's between him and the client
- Bandwidth control on intermediate router before bottleneck
 - Ideal case because this is where packet loss would occur
 - Sophisticated queue scheduling on the outgoing queue
 - Variety of different queue scheduling algorithms
- Flow throttling at the Receiver
 - Worst case, because influence is limited
 - Theoretically possible for TCP, no implementation yet.
 - Ingress qdisc might help

Bandwidth limiting at server

- Some Internet Servers support bandwidth limiting
 - ProFTPD (builtin support)
 - Apache (using contributed mod_bandwidth)
- Using those features it is easy to limit
 - maximum bandwidth used per connection
 - maximum bandwidth used per client (IP/network)
 - maximum bandwidth used by one virtual host (webserver/ftpserver)

Router before bottleneck

The router receives more packets on his incoming interface(s) than it can send out on the outgoing interface. It has to build a queue of packets (usually a FIFO one) and starts dropping packets as soon as the queue is full



The idea is to change this queue, thus decide

- which packets get enqueued in which order
- how many packets get queued
- which packets get dropped in case of a filling queue

The Linux 2.2 / 2.4 Solution

- Packet Scheduling algorithms in the Kernel
 - CBQ - Class Based Queue
 - RED - Random Early Drop
 - SFQ - Stochastic Fairness Queueing
 - TEQL - True Link Equalizer
 - TBF - Token Bucket Filter

- tc command of iproute2 package for configuration
 - almost no documentation
 - very few examples on the internet

- Packet Classification
 - tc builtin classes (route, u23, ...)
 - all iptables/netfilter matches by using fwmark

Conclusion: Linux is the best suited general-purpose operating system for QoS, but almost nobody is using it because lack of knowledge.

Available queuing algorithms

- CBQ - Class Based Queue
 - hierarchical bandwidth classes
 - used as basis in almost all cases
- TBF - Token Bucket Filter
 - really accurate algorithm
 - uses a lot of CPU
 - not possible for high bandwidth links (>1MBit)
- SFQ - Stochastic Fairness Queueing
 - less accurate algorithm
 - tries to distinguish between individual streams
 - does round robin between those streams
- TEQL - True Link Equalizer
 - allows to 'bundle' interfaces
- RED - Random Early Detect / Drop
 - simulates congested link by statistic packet dropping
 - uses almost no CPU
 - recommended for high-bandwidth backbones
- others (WRR, TCINDEX, DSMARK, ..)
 - WRR not officially included in kernel, similar to CBQ
 - others mostly used for DiffServ

The big picture

Overview of the a packet's journey



Example scenario using CBQ

Let's assume we have a link with 10 MBit maximum available bandwidth. We offer two major services to the outside world: Anonymous FTP and a Webserver offering important Information.

FTP Bulk data transfers are using up almost all available bandwidth, thus slowing down accesses to our website :(

We want to have FTP transfers use up to 8MBit and reserve 2MBit for WWW. Implementation uses CBQ for bandwidth divisions.

Example scenario

- attach a CBQ to the device

```
tc qdisc add dev eth0 root handle 10: cbq
  bandwidth 10Mbit avpkt 1000
```

- create CBQ classes

```
tc class add dev eth0 parent 10:0 classid 10:1 cbq
  bandwidth 10Mbit rate 10Mbit allot 1514
  weight 1Mbit prio 8 maxburst 20 avpkt 1000
```

```
tc class add dev eth0 parent 10:1 classid 10:100 cbq
  bandwidth 10Mbit rate 8Mbit allot 1514
  weight 800kbit prio 5 maxburst 20 avpkt 1000 bounded
```

```
tc class add dev eth0 parent 10:1 classid 10:200 cbq
  bandwidth 10Mbit rate 2Mbit allot 1514
  weight 200kbit prio 5 maxburst 20 avpkt 1000 bounded
```

- add filter rules

```
tc filter add dev eth0 parent 10:1 protocol ip handle 6 fw classid 10:100
```

```
iptables -t mangle -A PREROUTING -j MARK -p tcp --sport 20 --set-mark 6
```

Further optimization

Now we have achieved bandwidth division between two services.

Within one service, however, one individual user with a high bandwidth link can still use up most of our bandwidth, slowing down other user.

We can improve this behaviour of changing the scheduling algorithm from it's default (fifo)

```
tc qdisc add dev eth0 parent 10:100 sfq quantum 1514b perturb 15
tc qdisc add dev eth0 parent 10:200 sfq quantum 1514b perturb 15
```

Further reading / Links

- Bandwidth limiting on Servers
 - ProFTPd
 - ▷ <http://www.proftpd.net/>
 - Apache mod_bandwidth / mod_bwshare
 - ▷ ftp://ftp.cohprog.com/pub/apache/module/mod_bandwidth.c
 - ▷ <http://www.topology.org/src/bwshare/>
- Queue scheduling
 - Advanced Routing HOWTO
 - ▷ <http://www.ds9a.nl/2.4Routing/>
 - Linux QoS HOWTO
 - ▷ <http://www.ittc.ukans.edu/~rsarav/howto/>
 - iproute2+tc
- This presentation
 - Authors Homepage
 - ▷ <http://www.gnumonks.org/>