



VoIP e mitos: por que a voz picota, atrasa... QoS e seus desafios

Marlon Dutra

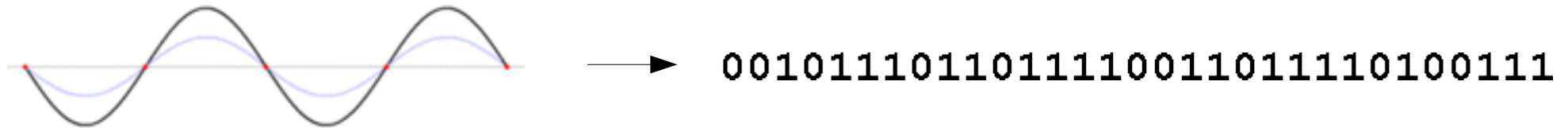
Porto Alegre, 30 de junho de 2011

- Sysadmin desde 1996
Servidores, roteadores, switches
Telefonia IP
- Desenvolvedor Python (ex Perl)
- Propus Informática - diretor
- FISL - ex-presidente
- Piloto privado de avião
- Fotógrafo amador

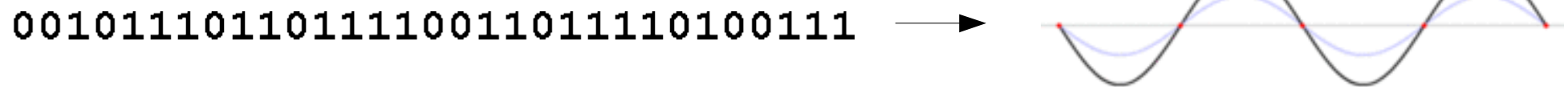


- **Conceitos de voz digital**
- **Gargalos de transmissão - perda de pacotes**
- **Conceito de QoS**
- **Conceito de largura de banda, latência, jitter...**
- **Como QoS funciona**
- **Alguns mecanismos de tratamento**
- **Dicas de implementação**
- **Perguntas e respostas**

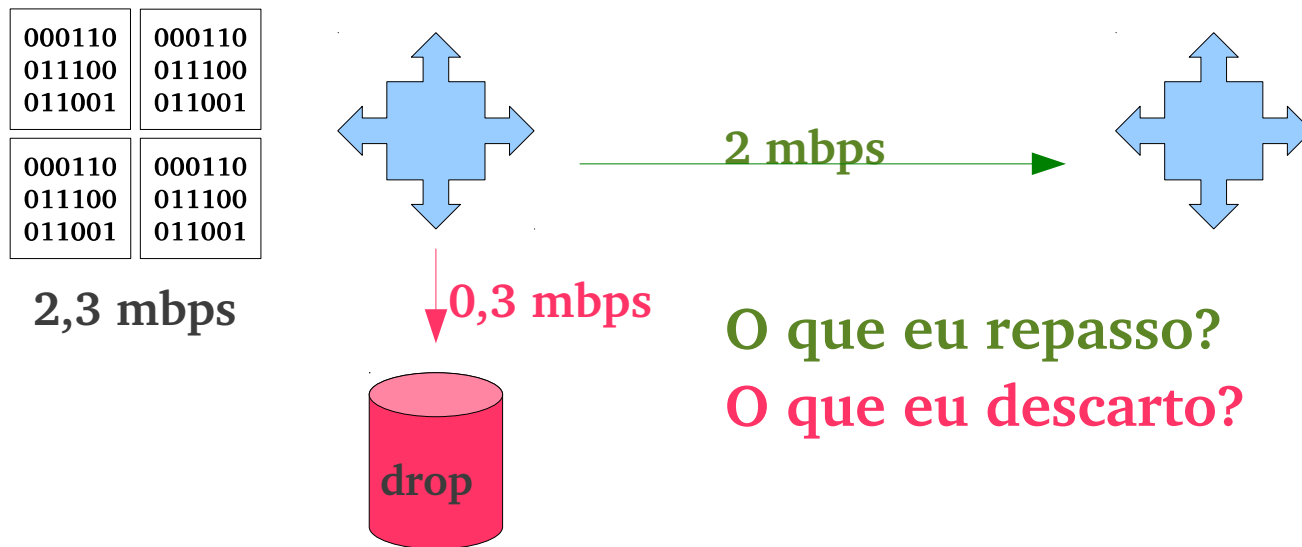
Co dificação



Dec o dificação



Codec - processo que converte a voz em dados, e vice-versa



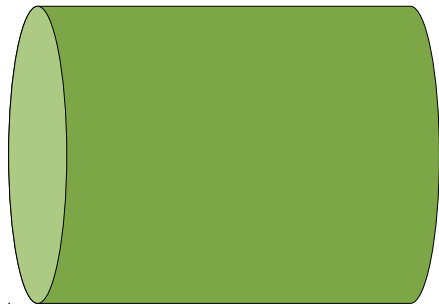
QoS = Quality of Service

Trabalhar na escassez
Prioridades diferentes para serviços diferentes

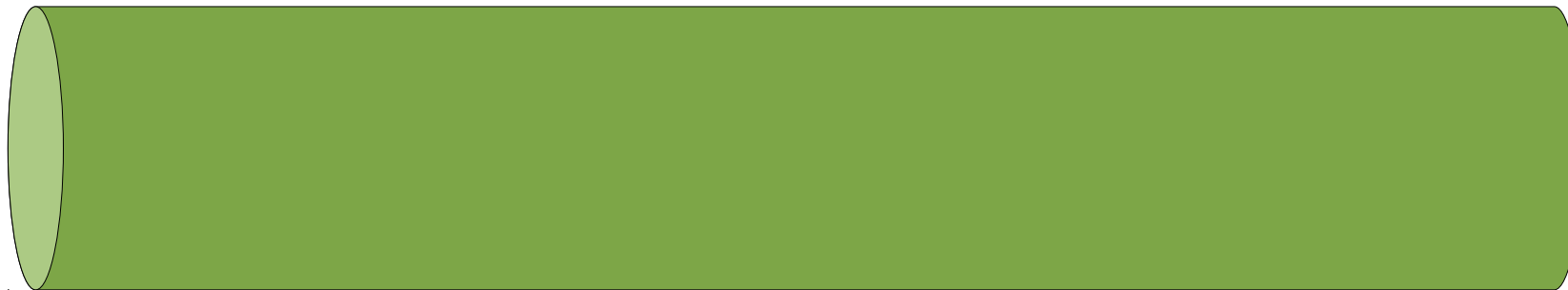
Se não há escassez, esqueça QoS!



velocidade - latência - medida em tempo



largura de banda - vazão (*bandwidth*)
bits / segundo

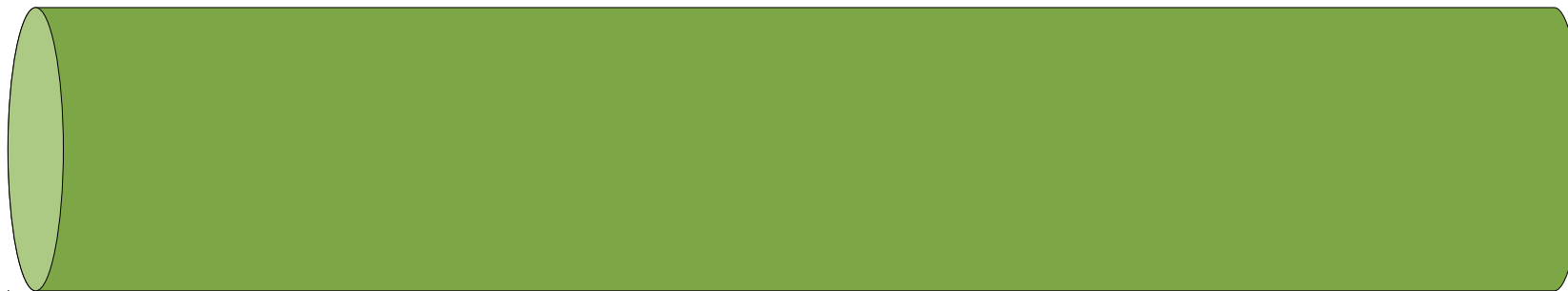


linha de tempo 

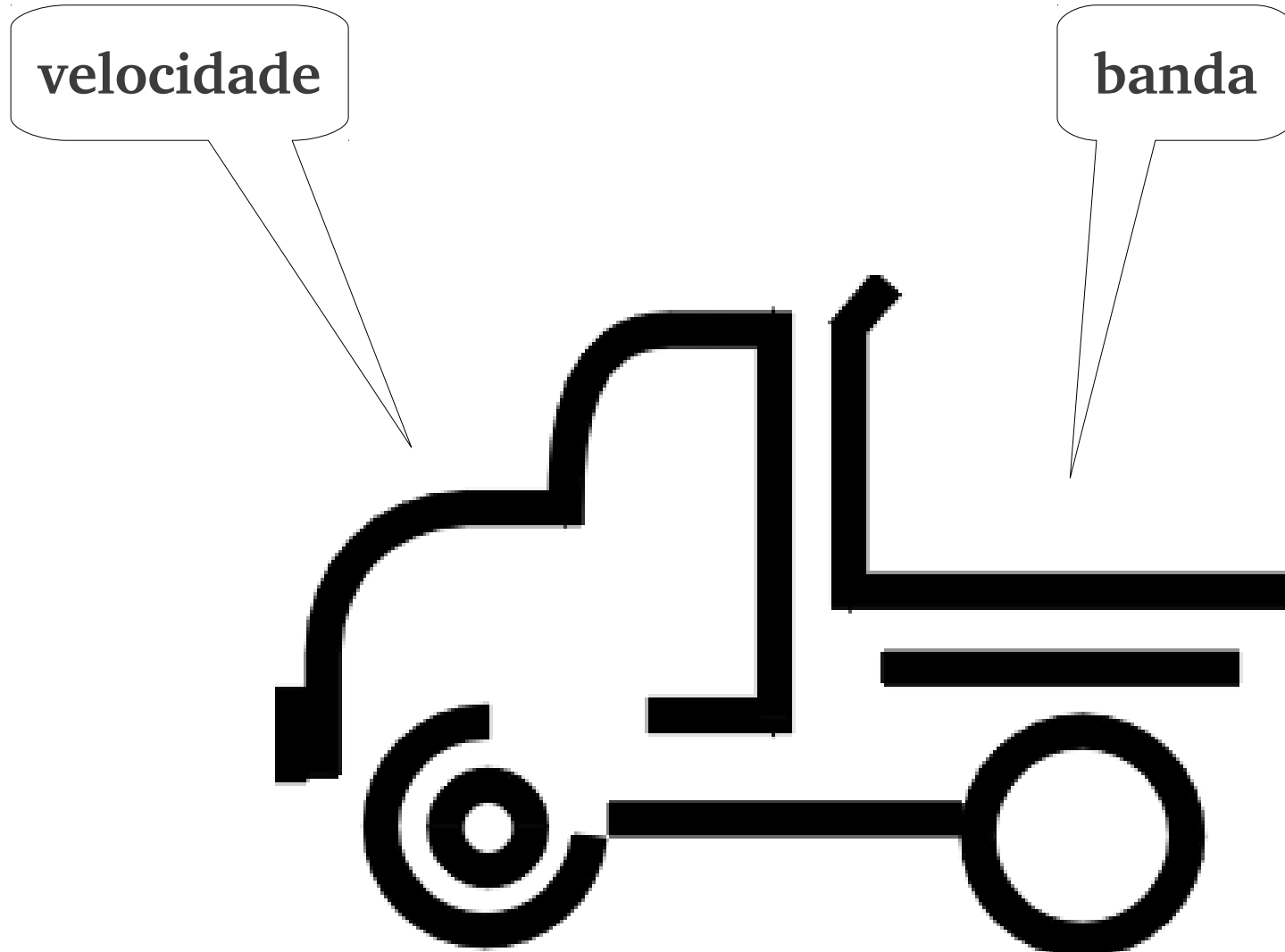


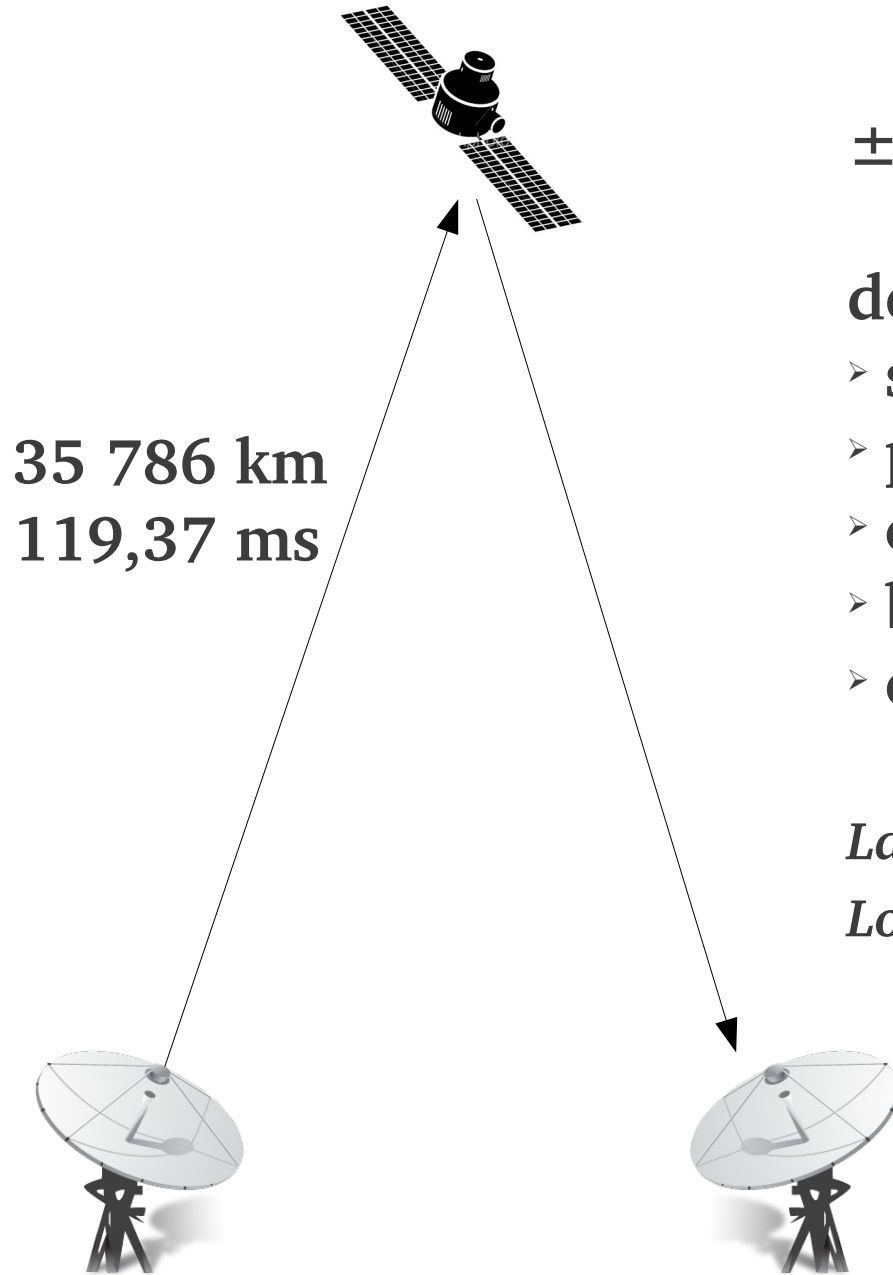
Velocidade ou latência: comprimento do cano

Largura de banda: diâmetro do cano



linha de tempo →





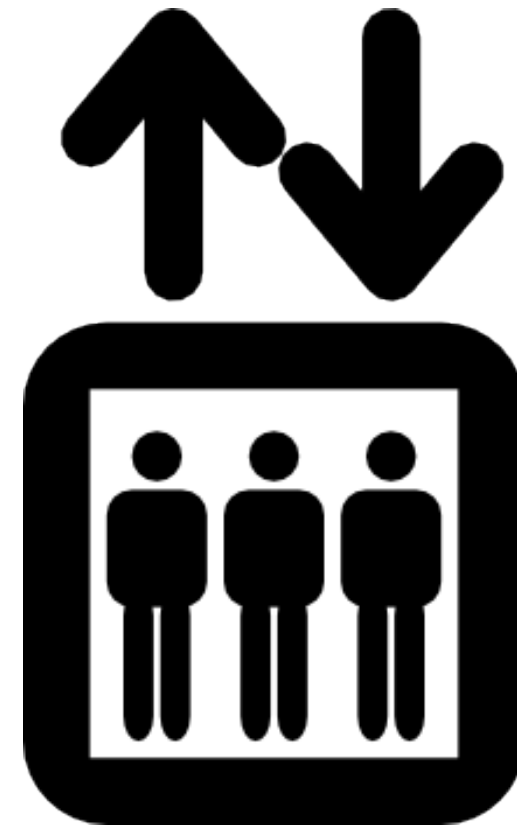
± 240 ms ida-e-volta

desprezando-se outras latências

- sinalização
- processamento
- compressão
- buffering
- etc...

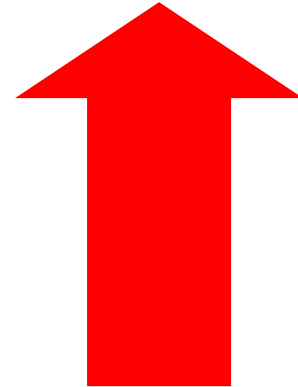
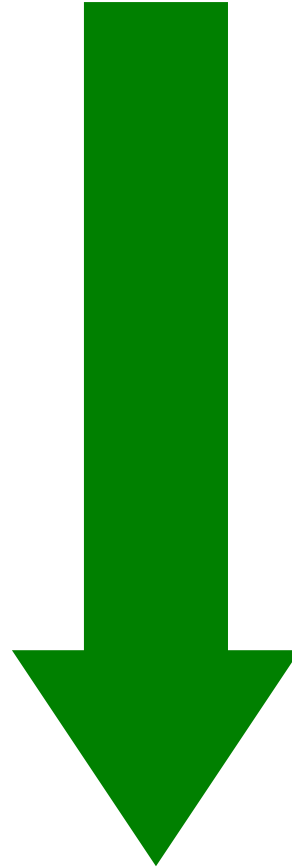


latência conhecida
jitter zero
não muito eficiente
extremamente caro

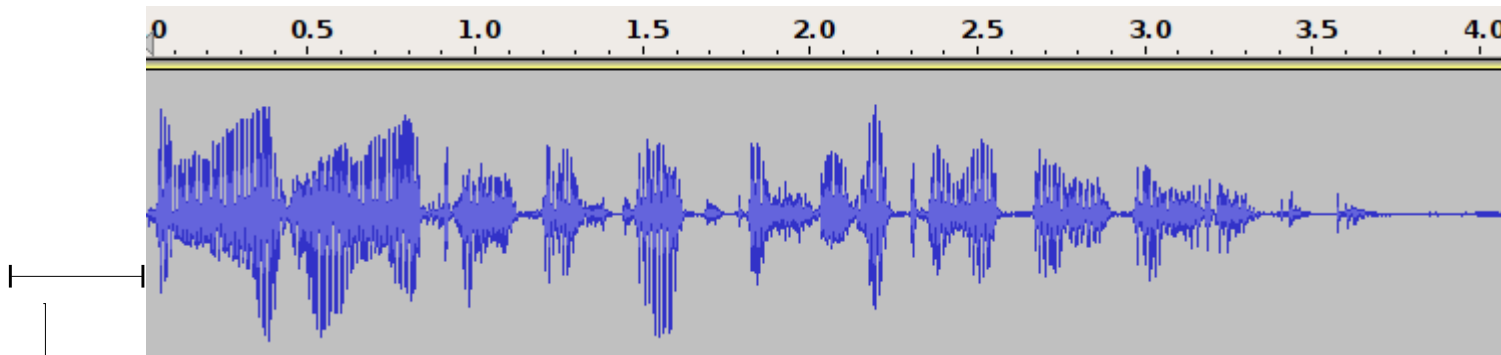
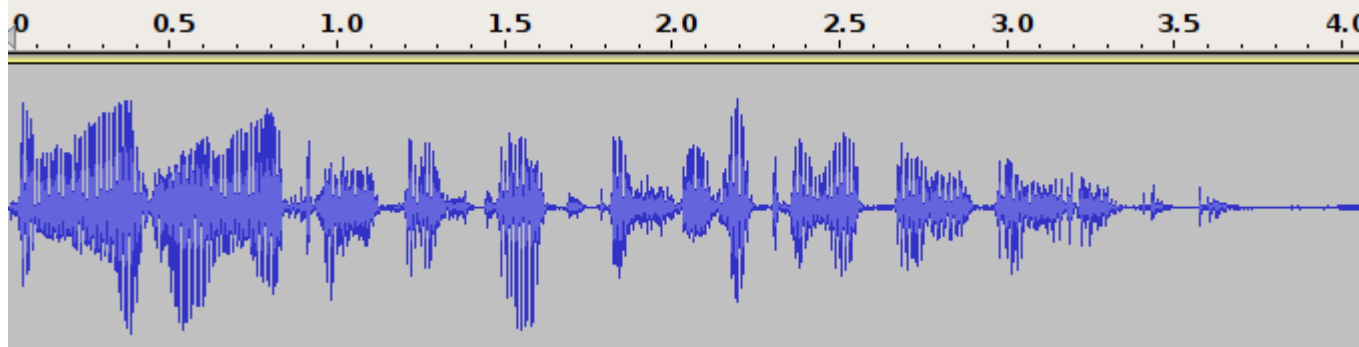


latência variável
jitter variável
eficiente e mais escalável
barato

download



upload



▶ latência: exemplo 150 ms constante

jitter: latência variando - alguns pacotes atrasam mais que outros
(acontece apenas em linhas assíncronas normalmente)

```
% ping -nc 10 1.2.3.4
```

```
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
```

```
64 bytes from 1.2.3.4: icmp_seq=1 ttl=52 time=136 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=2 ttl=52 time=238 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=3 ttl=52 time=129 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=4 ttl=52 time=124 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=5 ttl=52 time=160 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=6 ttl=52 time=183 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=7 ttl=52 time=118 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=8 ttl=52 time=253 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=9 ttl=52 time=230 ms
```

```
64 bytes from 1.2.3.4: icmp_seq=10 ttl=52 time=96.7 ms
```

```
--- 1.2.3.4 ping statistics ---
```

```
10 packets transmitted, 10 received, 0% packet loss , time 9000ms
```

```
rtt min/avg/max/ mdev = 96.743/167.153/253.324/ 53.252 ms
```

mdev = desvio médio

```
% ping -i 0.02 -s 33 -w 10 1.2.3.4
```

```
--- 1.2.3.4 ping statistics ---
```

```
486 packets transmitted, 438 received, 9% packet loss , time 9993ms
```

```
rtt min/avg/max/mdev = 0.134/195.846/504.396/ 152.369 ms
```

-i 0.02 = 50 pacotes por segundo / 1 pacote a cada 0,02 segundos

-s 33 = 33 bytes de dados em cada pacote (simulando GSM)

-w 10 = encerra o teste em 10 segundos

ATENÇÃO: cuidado com ping. Alguns peers limitam a resposta de ping de propósito, para evitar um DoS

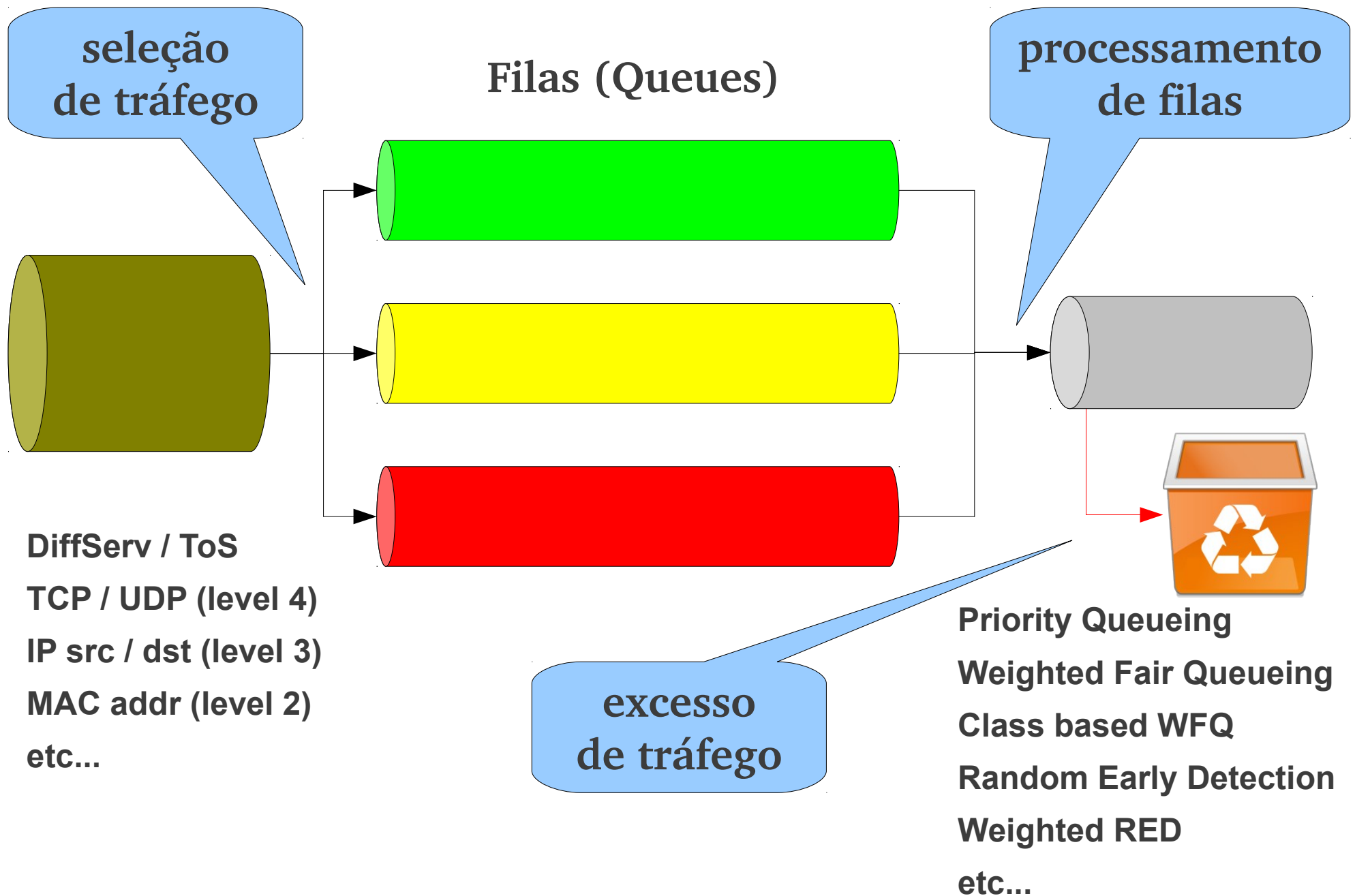
Dica: utilize também o **mtr** , para ver o estado em cada hop

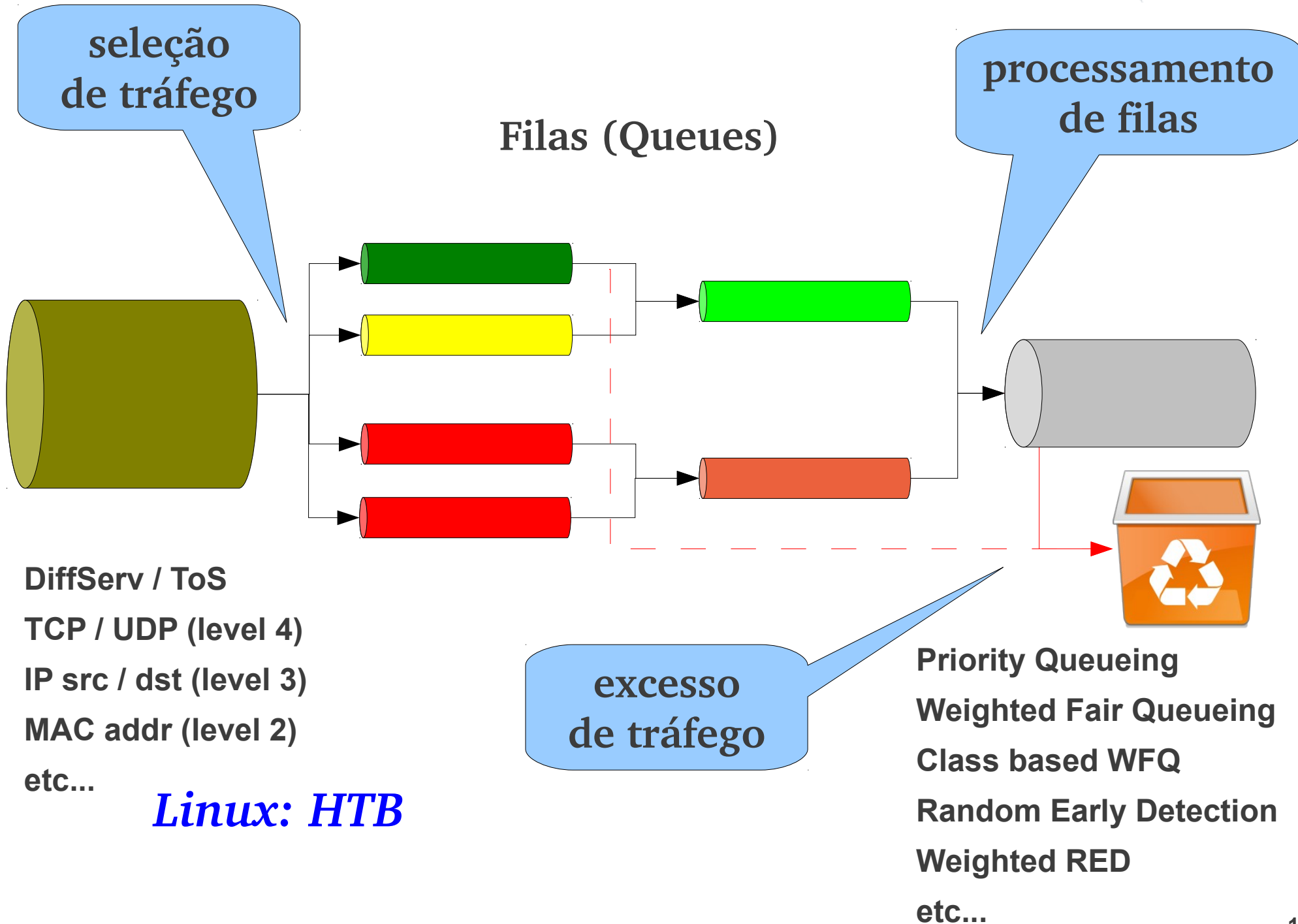
QoS /kju/ou/es/ : ato de ferrar com a vida de alguns pacotes para que alguns outros tenham uma vida boa.

QoS não ajuda em nada, só atrapalha!

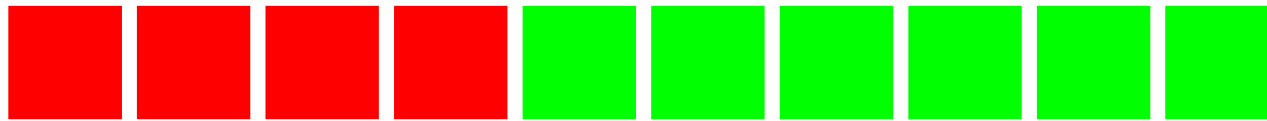
A questão está em: quem vamos atrapalhar hoje?
E qual será nosso grau de malvadeza? :-D

Só existe QoS para o que você envia.
O que você recebe já foi recebido...
(parcialmente verdade)



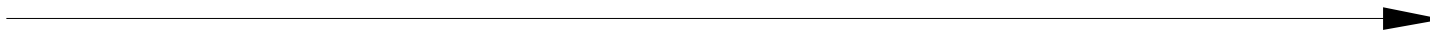


FIFO - Tail drop



acima da
capacidade

buffer 100%

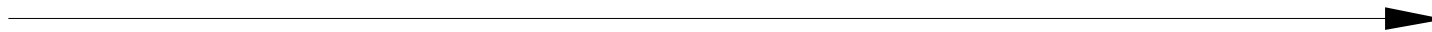


Variante: pFIFO (ToS)

RED - Random early detection



buffer < 100%

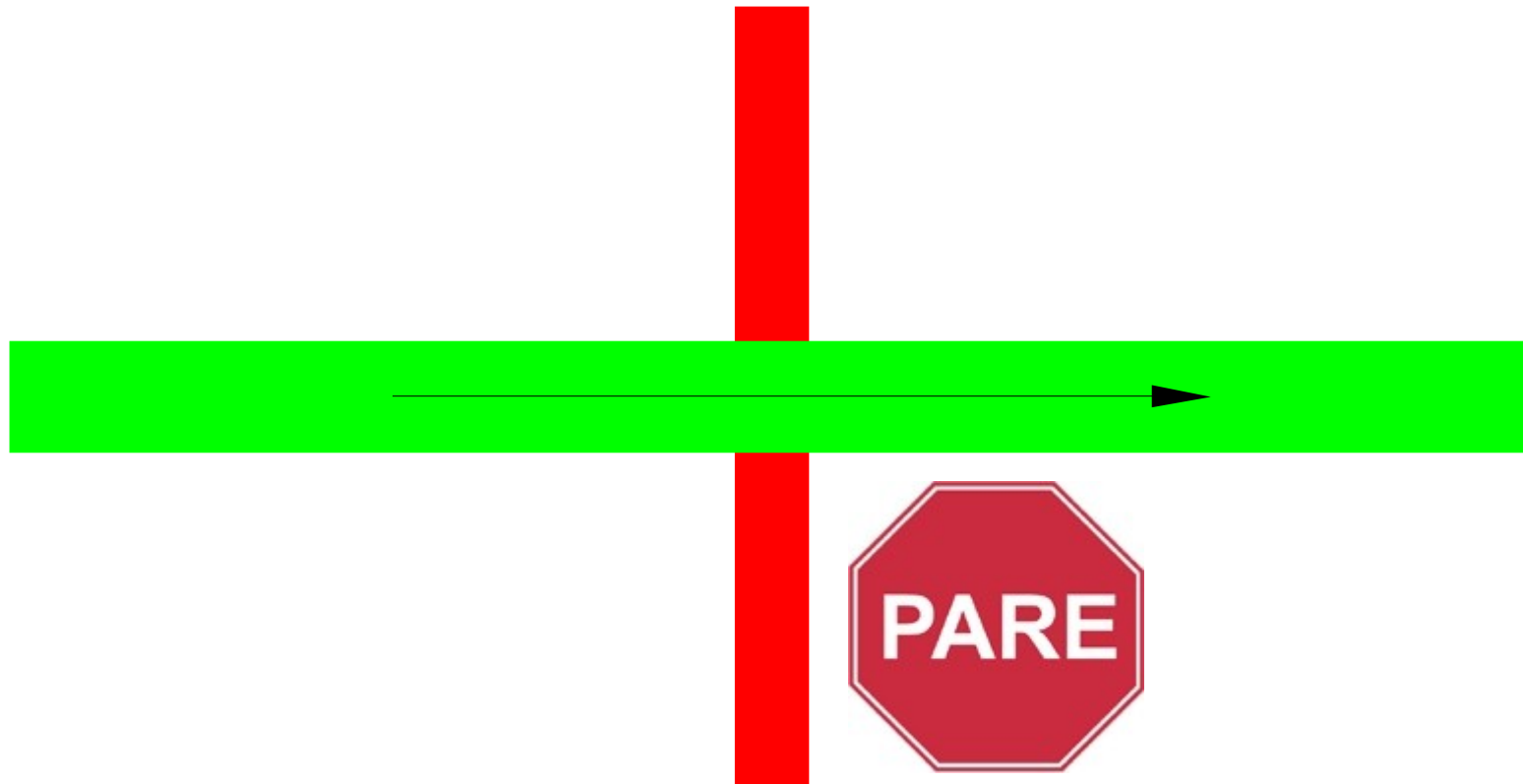


Variantes: Weighted RED, Adaptative RED

PQ - Priority queue

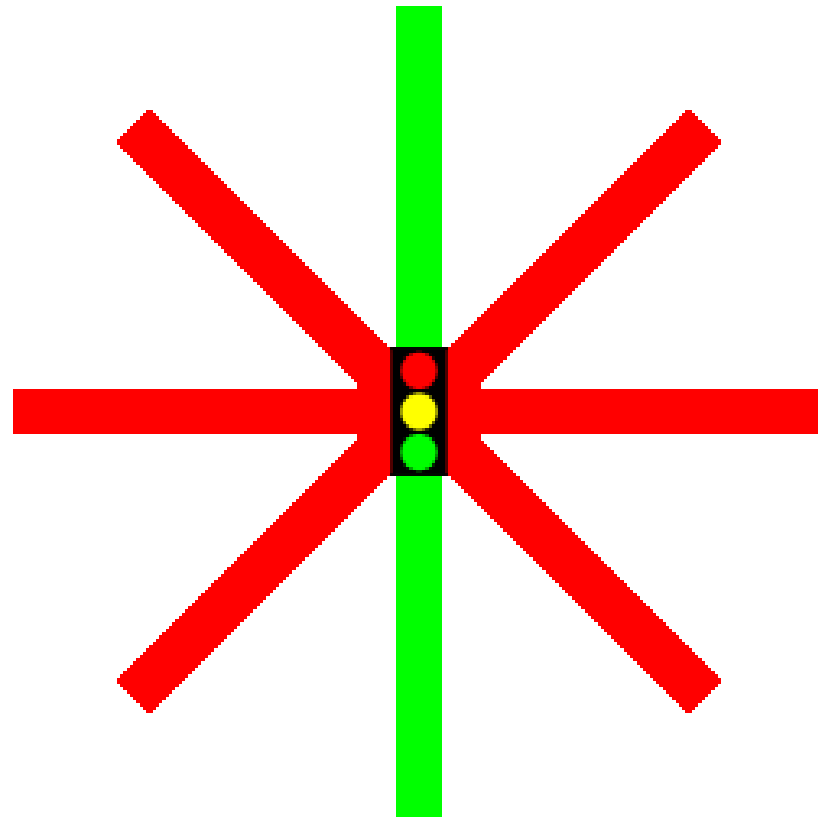
Strict priority

LLQ - low latency queue



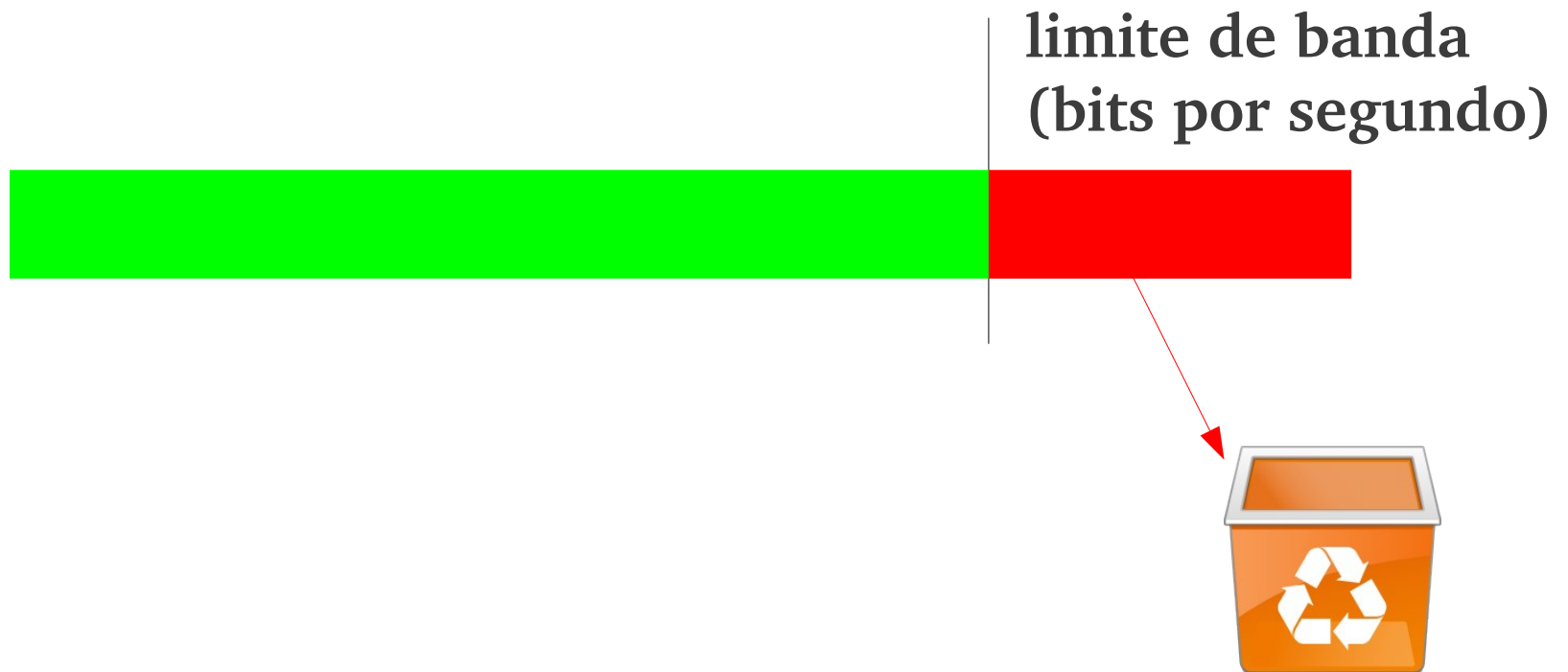
FQ - Fair queue

SFQ - Stochastic fairness queueing



Variante: WFQ, CBWFQ

CAR - Committed access rate *Rate limiting*



Traffic shaping *HTB, CBQ, TBF...*



limite de banda
(bits por segundo)



excesso

EF	AF11	AF21	AF31	AF41	BE
	AF12	AF22	AF32	AF42	
	AF13	AF23	AF33	AF43	

<http://bit.ly/kB5mjR>

Implementação completa de DiffServ em Linux

- Primeiro de tudo: verifique o estado de seu link **sem tráfego**
 - Se ele não for bom, QoS nenhum no mundo vai ajudar
- Determine seu teto de transmissão precisamente - será seu 100%
 - iperf
- Esqueça o termo “reserva de banda” para voz
- Você precisa frear e baixar a prioridade do resto do tráfego
 - Lembre-se: QoS existe para atrapalhar os outros
- Evite fila nos roteadores (FIFO). Se o seu uplink é 400 Kbps, entregue o tráfego a uns 360 Kbps para o roteador
- Alguns tipos de circuitos aumentam a latência chegando próximo do limite de banda, especialmente uplink. Considere isso.
- Experimente, teste, experimente, teste, experimente...
- KISS: keep it short and simple - **não invente moda!**

- 1) Entenda seu problema**
- 2) Entenda os conceitos de QoS**
- 3) Desenhe a solução conceitualmente**
- 4) Implemente**
- 5) Teste**

Sua cabeça está explodindo? Sim, normal!

Perguntas?

Informações sobre QoS em Linux:

<http://lartc.org/>

<http://www.linux.org/docs/ldp/howto/Traffic-Control-HOWTO/>

Marlon Dutra

**<marlon@propus.com.br>
@mfdutra**

<http://hackers.propus.com.br/~marlon/>



<http://www.propus.com.br>

Envie seu curriculum vitae!!