MUM-VN-KH-2015
Hanoi – Phnom Penh

QoS on Multicore Router

RouterOS v6.xx

Valens Riyadi (Citraweb)
info@mikrotik.co.id

About Me

Valens Riyadi, Citraweb (ID)

MikroTik Certified Engineer
(MTCNA, MTCWE, MTCRE, MTCTCE, MTCUME, MTCINE)

MikroTik Certified Trainer since 2004

MikroTik Certified Consultant

MikroTik Academy Coordinator

Citra.net.id WISP CEO

Manager for IDNIC (Indonesia National Internet Registry)

IT Expert on Disaster Relief

Proud member of “Routed World” community

MikroTik™

distributor

www.mikrotik.co.id


mikrobits
CELOICA 8101
AINOS 2071
AINOS
ANETO
DINARA
mikrobits

MikroTik Training Center

- The first MikroTik Training Center in Asia Pasific, has more then 3500 participants (151 classes).
- Mikrotik Academy Coordinator.

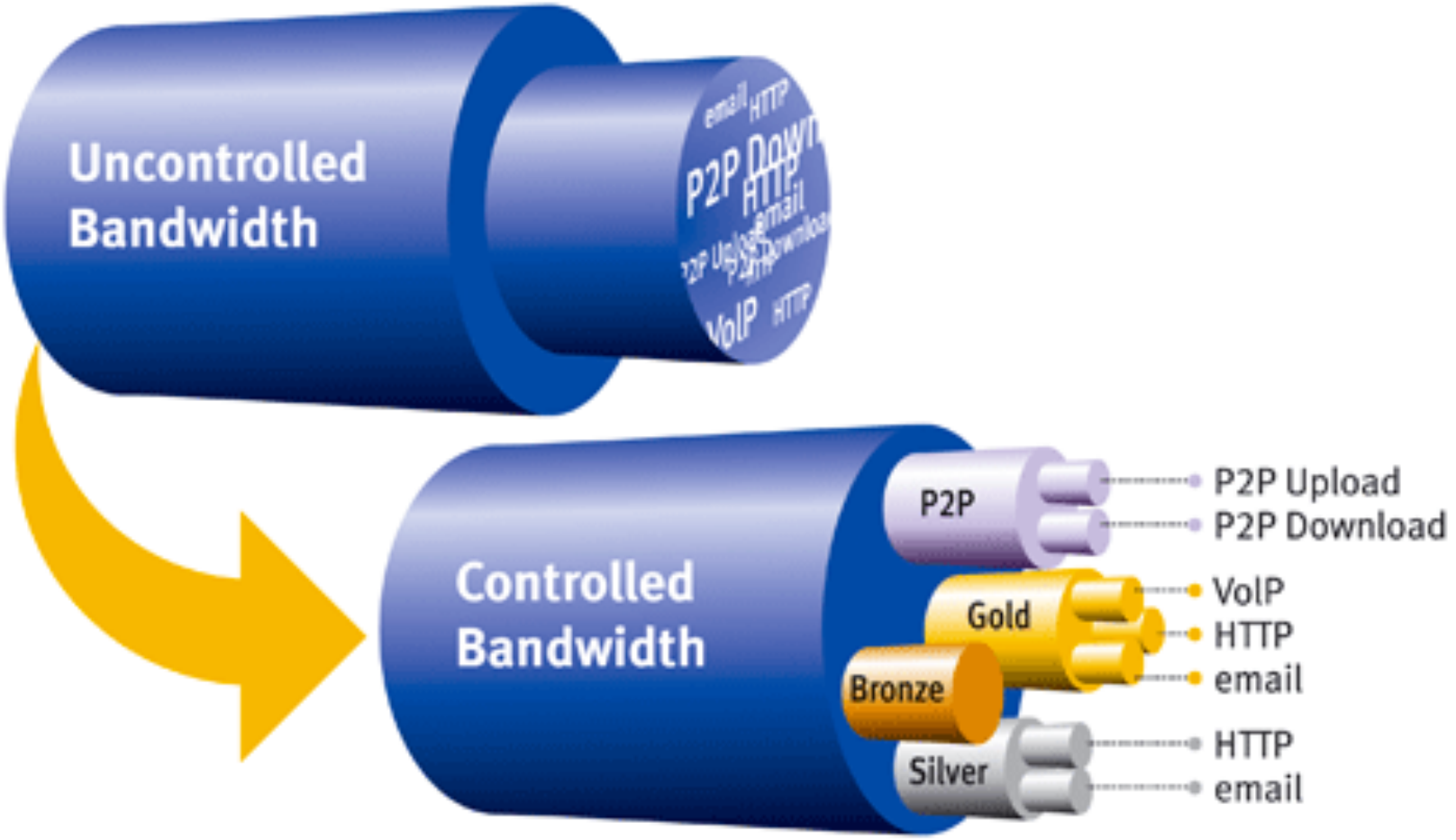
Recommended Resources

- CCR Status Update (Janis Megis – Video)
  http://tiktube.com/video/GJil3aqniCGJCIqqpGnrGznrClGoJGJo=
- Dynamic QoS on RouterOS v6 (Valens Riyadi)
  http://mum.mikrotik.com/presentations/IT14/valens.pdf
- QoS on RouterOSv6 (Valens Riyadi)
  http://mum.mikrotik.com/presentations/HR13/valens.pdf
- HTB vs PCQ (Valens Riyadi)
  http://mum.mikrotik.com/presentations/HU11/valens.pdf
- QoS and Traffic Priorities (Janis Megis)
  http://mum.mikrotik.com/presentations/CZ09/QoS_Megis.pdf
- HTB QoS (Valens Riyadi)
  http://mum.mikrotik.com/presentations/US09/Valens-MUM2009USA.pdf

QoS concept

The process of measuring and controlling the communications (traffic, packets) on a network link, to avoid filling the link to capacity or overfilling the link, which would result in network congestion and poor performance of the network.

Why do we need to manage bandwidth?



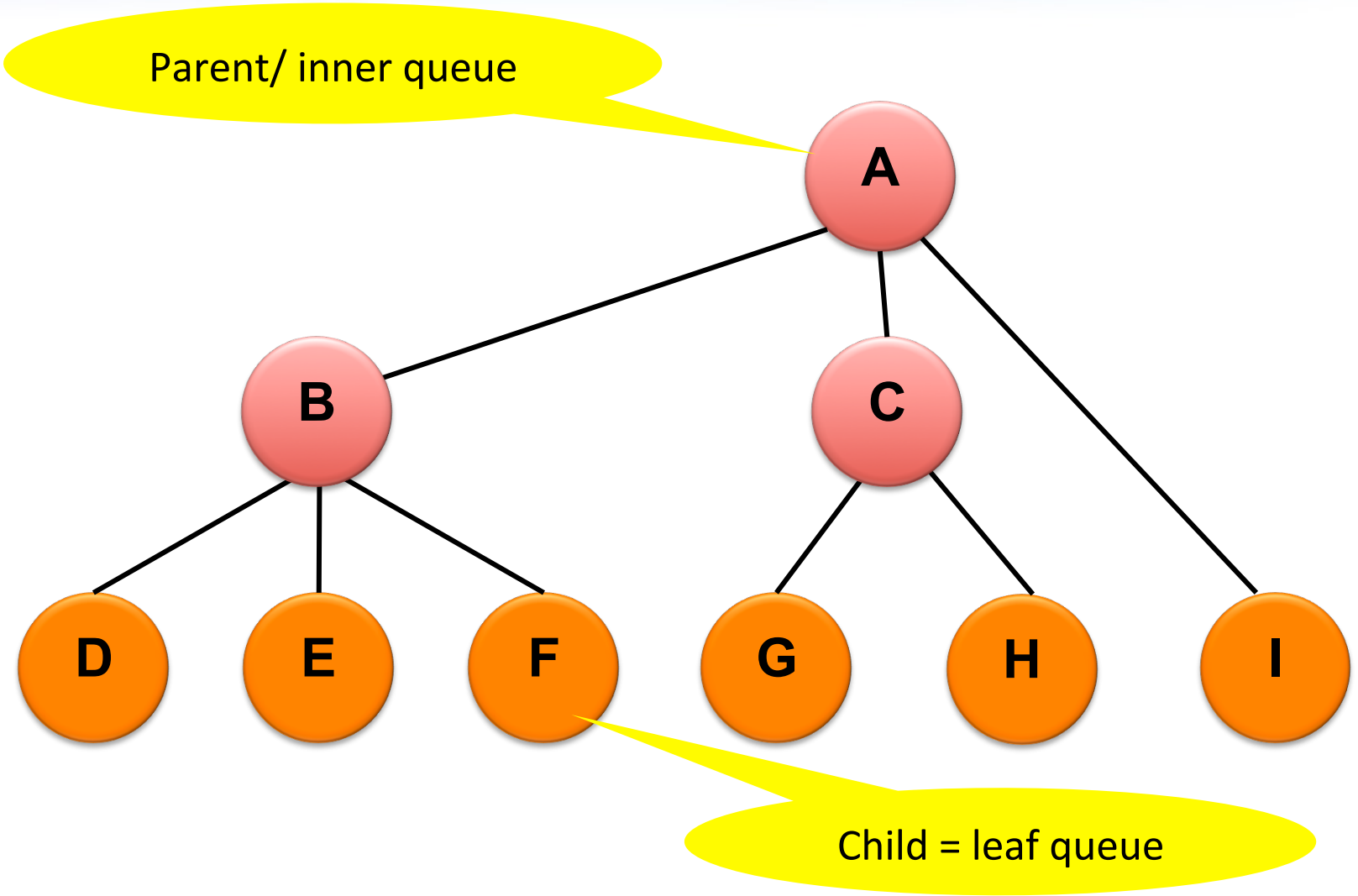
QoS on RouterOS

- MikroTik RouterOS is one of the most advanced bandwidth management, compared to any other brand.
- Why?
  - Advanced HTB configuration
  - Double limitation + Burst
  - A lot of option and parameter → packet-mark

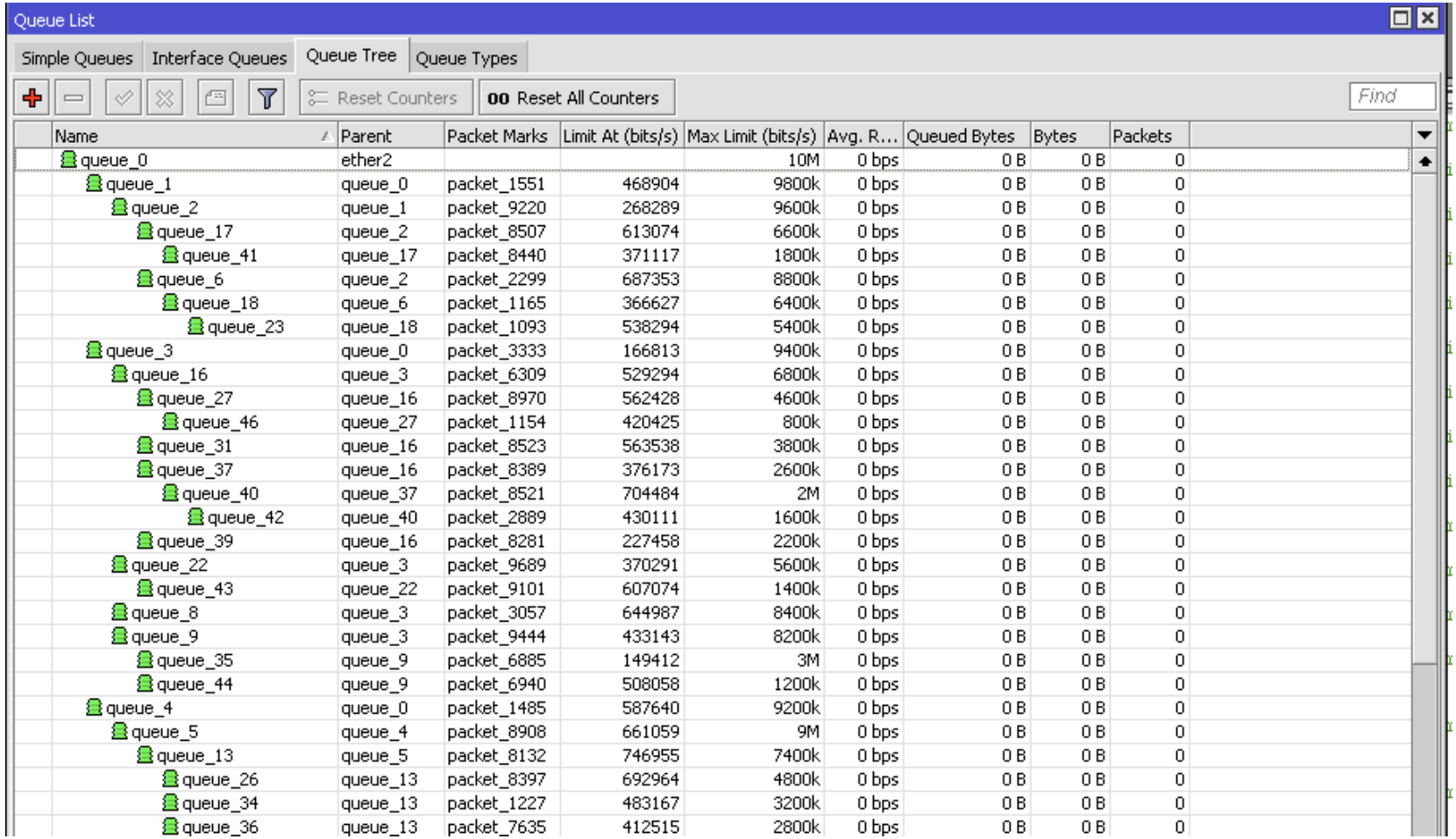
HTB

- Mostly, Quality of Service implementation in RouterOS is based on Hierarchical Token Bucket
- HTB allows to create hierarchical queue structure and determine relations between parent and child queues and relation between child queues
- RouterOS v6 support 1 virtual HTBs (global), and one more just before every interface

HTB Sample



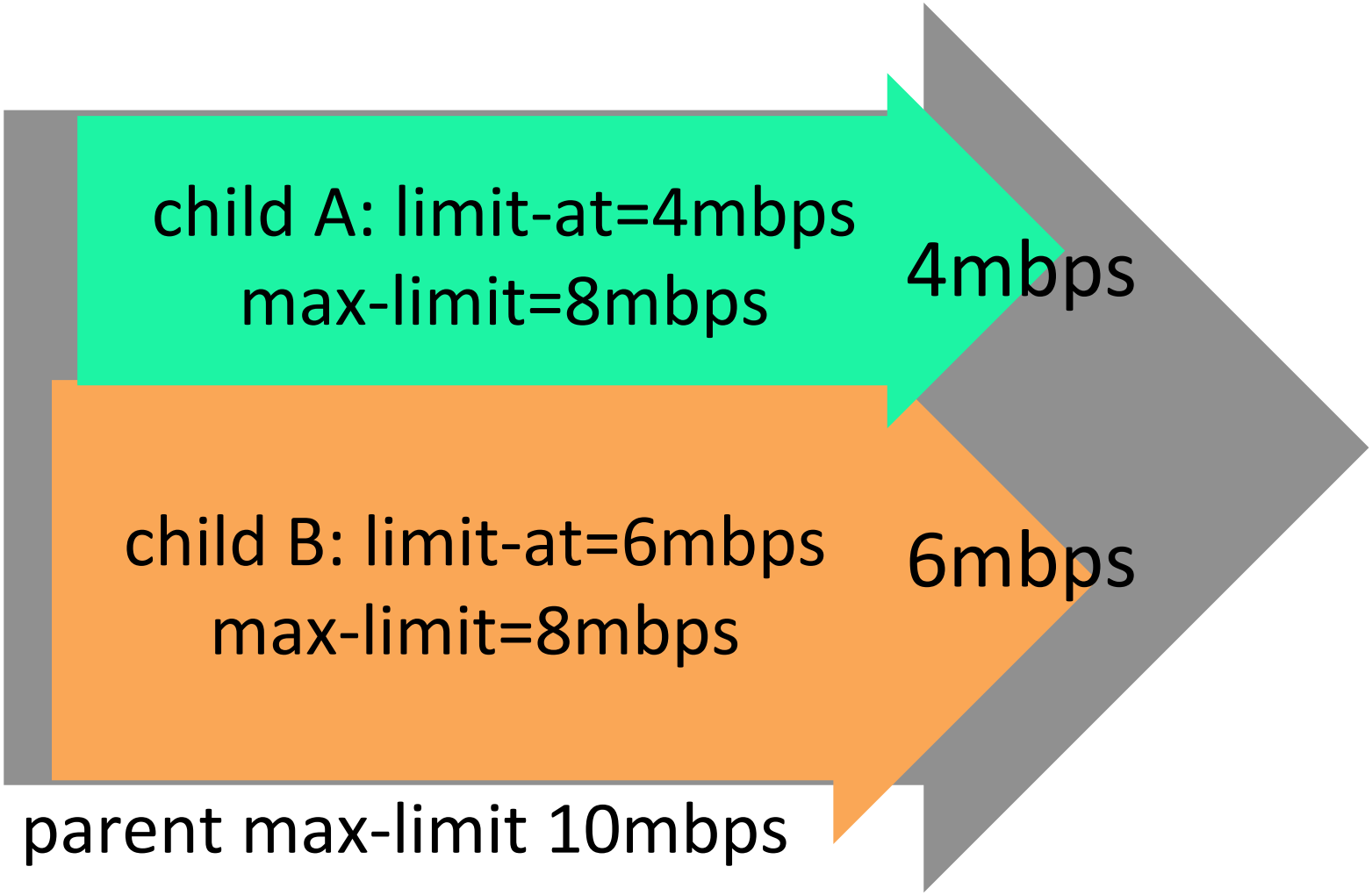
HTB Implementation Example

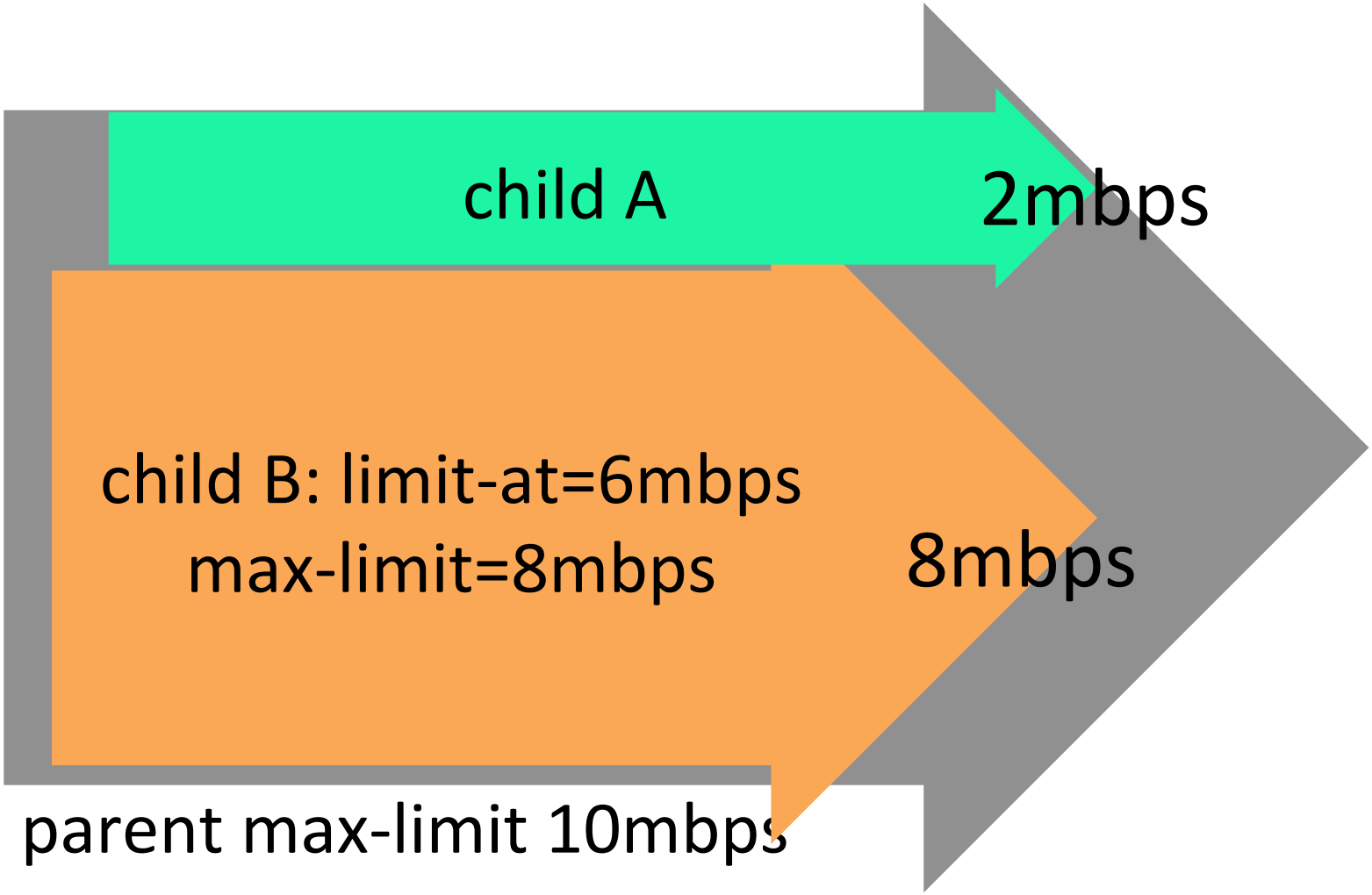


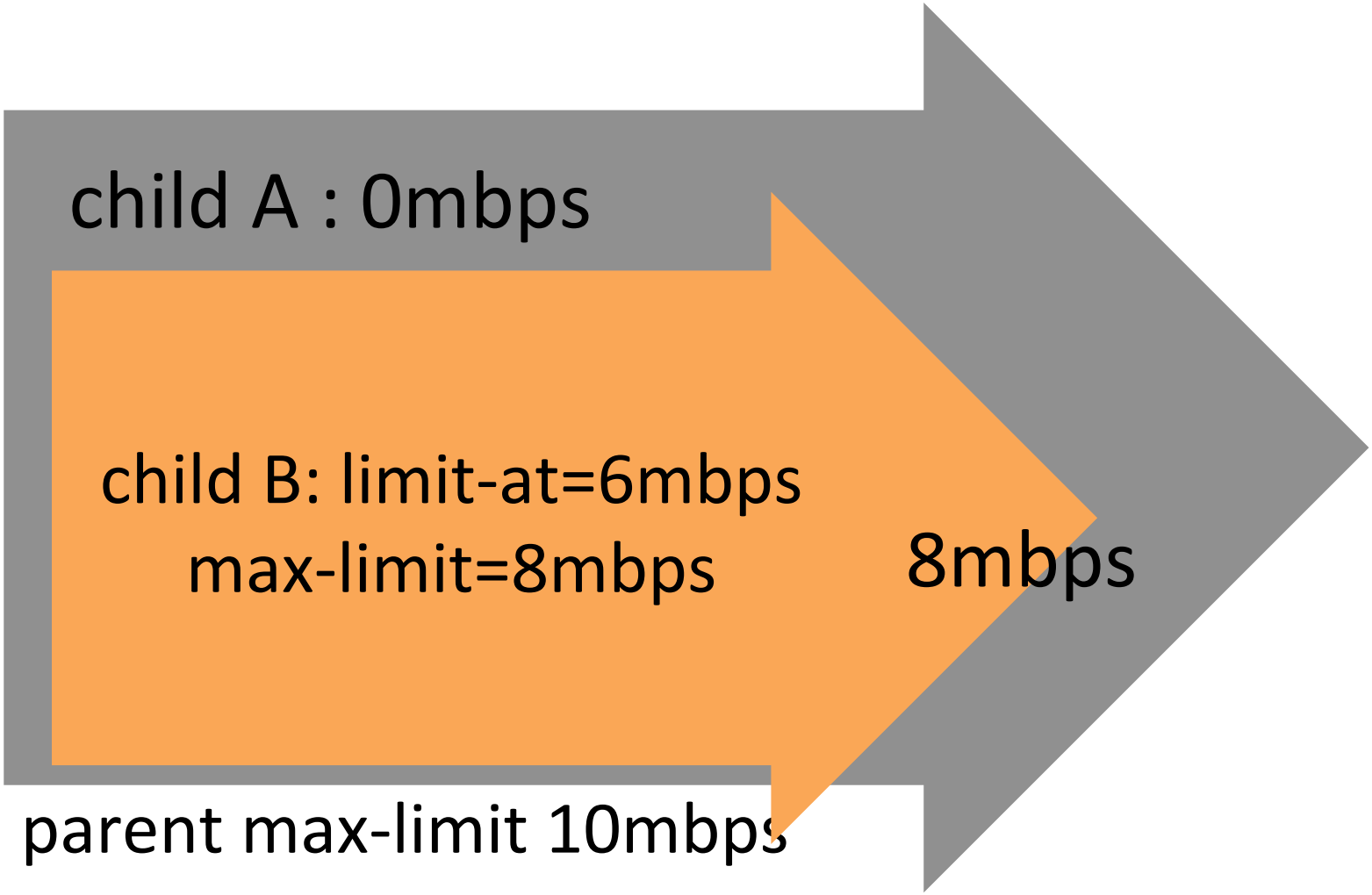
Name	Parent	Packet Marks	Limit At (bits/s)	Max Limit (bits/s)	Avg. R...	Queued Bytes	Bytes	Packets
queue_0	ether2			10M	0 bps	0 B	0 B	0
queue_1	queue_0	packet_1551	468904	9800k	0 bps	0 B	0 B	0
queue_2	queue_1	packet_9220	268289	9600k	0 bps	0 B	0 B	0
queue_17	queue_2	packet_8507	613074	6600k	0 bps	0 B	0 B	0
queue_41	queue_17	packet_8440	371117	1800k	0 bps	0 B	0 B	0
queue_6	queue_2	packet_2299	687353	8800k	0 bps	0 B	0 B	0
queue_18	queue_6	packet_1165	366627	6400k	0 bps	0 B	0 B	0
queue_23	queue_18	packet_1093	538294	5400k	0 bps	0 B	0 B	0
queue_3	queue_0	packet_3333	166813	9400k	0 bps	0 B	0 B	0
queue_16	queue_3	packet_6309	529294	6800k	0 bps	0 B	0 B	0
queue_27	queue_16	packet_8970	562428	4600k	0 bps	0 B	0 B	0
queue_46	queue_27	packet_1154	420425	800k	0 bps	0 B	0 B	0
queue_31	queue_16	packet_8523	563538	3800k	0 bps	0 B	0 B	0
queue_37	queue_16	packet_8389	376173	2600k	0 bps	0 B	0 B	0
queue_40	queue_37	packet_8521	704484	2M	0 bps	0 B	0 B	0
queue_42	queue_40	packet_2889	430111	1600k	0 bps	0 B	0 B	0
queue_39	queue_16	packet_8281	227458	2200k	0 bps	0 B	0 B	0
queue_22	queue_3	packet_9689	370291	5600k	0 bps	0 B	0 B	0
queue_43	queue_22	packet_9101	607074	1400k	0 bps	0 B	0 B	0
queue_8	queue_3	packet_3057	644987	8400k	0 bps	0 B	0 B	0
queue_9	queue_3	packet_9444	433143	8200k	0 bps	0 B	0 B	0
queue_35	queue_9	packet_6885	149412	3M	0 bps	0 B	0 B	0
queue_44	queue_9	packet_6940	508058	1200k	0 bps	0 B	0 B	0
queue_4	queue_0	packet_1485	587640	9200k	0 bps	0 B	0 B	0
queue_5	queue_4	packet_8908	661059	9M	0 bps	0 B	0 B	0
queue_13	queue_5	packet_8132	746955	7400k	0 bps	0 B	0 B	0
queue_26	queue_13	packet_8397	692964	4800k	0 bps	0 B	0 B	0
queue_34	queue_13	packet_1227	483167	3200k	0 bps	0 B	0 B	0
queue_36	queue_13	packet_7635	412515	2800k	0 bps	0 B	0 B	0

Queue Parameter

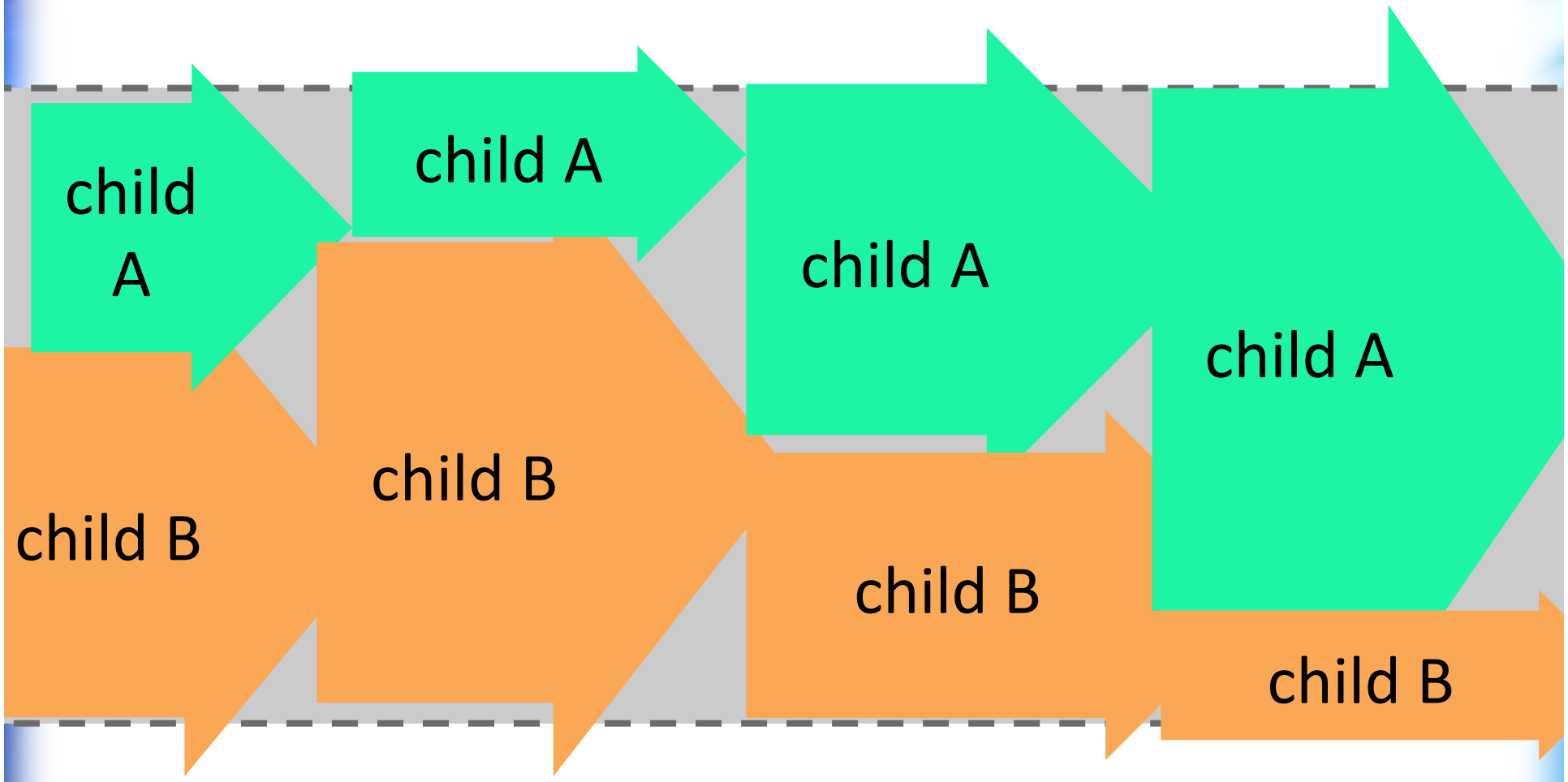
- limit-at (CIR)
- max-limit (MIR)
- burst (threshold, limit, time)
- queue type (FIFO, RED, SFQ, PCQ)
- parent


child A: limit-at=4mbps
max-limit=8mbps
4mbps
child B: limit-at=6mbps
max-limit=8mbps
6mbps
parent max-limit 10mbps


child A
2mbps
child B: limit-at=6mbps
max-limit=8mbps
8mbps
parent max-limit 10mbps

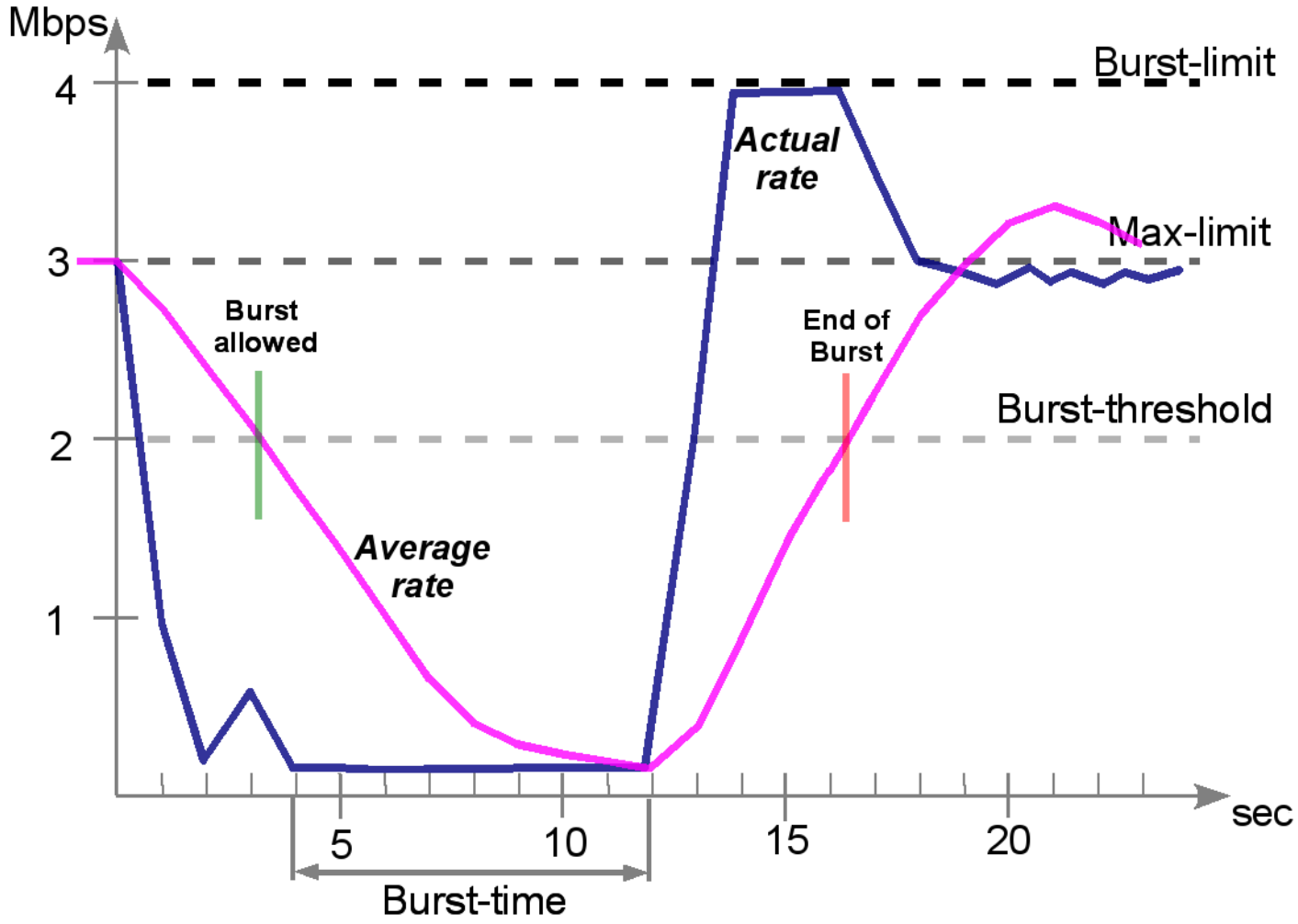

child A : 0mbps
child B: limit-at=6mbps
max-limit=8mbps
8mbps
parent max-limit 10mbps

without parent, with 10mbps link



Without parent, limit-at and priority will be ignored

Burts

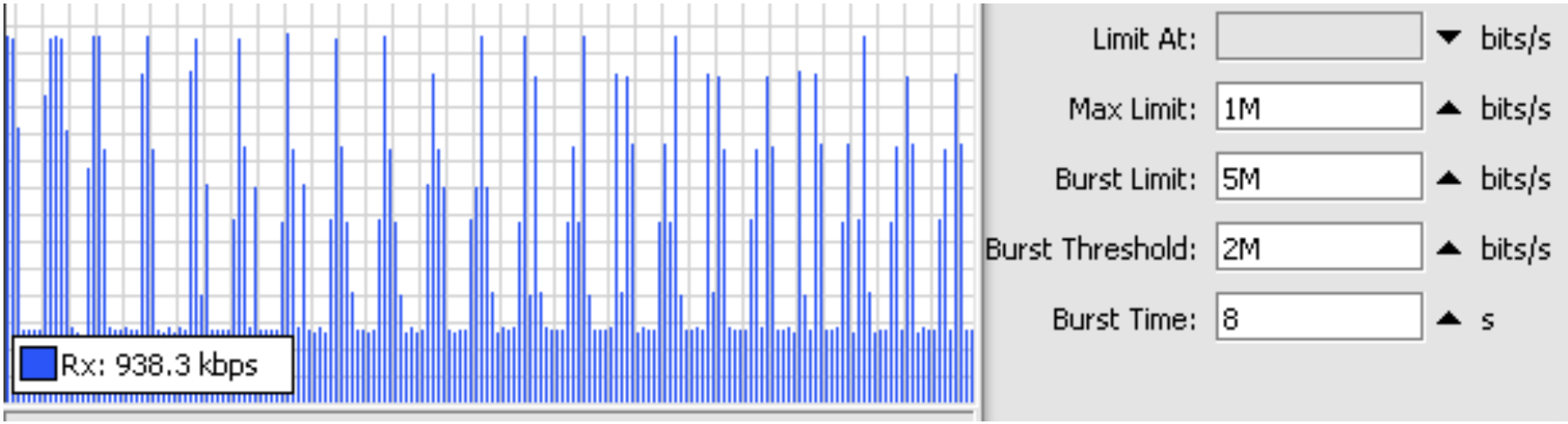

Mbps
4
3
2
1
Burst-limit
Max-limit
Burst-threshold
Actual rate
Average rate
Burst allowed
End of Burst
5
10
15
20
sec
Burst-time

Burst

- Burst is one of the best ways to increase HTTP performance
- Bursts are used to allow higher data rates for a short period of time
- If an *average data rate* is less than **burst-threshold**, burst could be used (*actual data rate* can reach **burst-limit**)
- *Average data rate* is calculated from the last **burst-time** seconds

Burst Example

- We can set different limit and burst for each leaf.
- Certain burst parameter will make “normal” customer think their bandwidth fast.



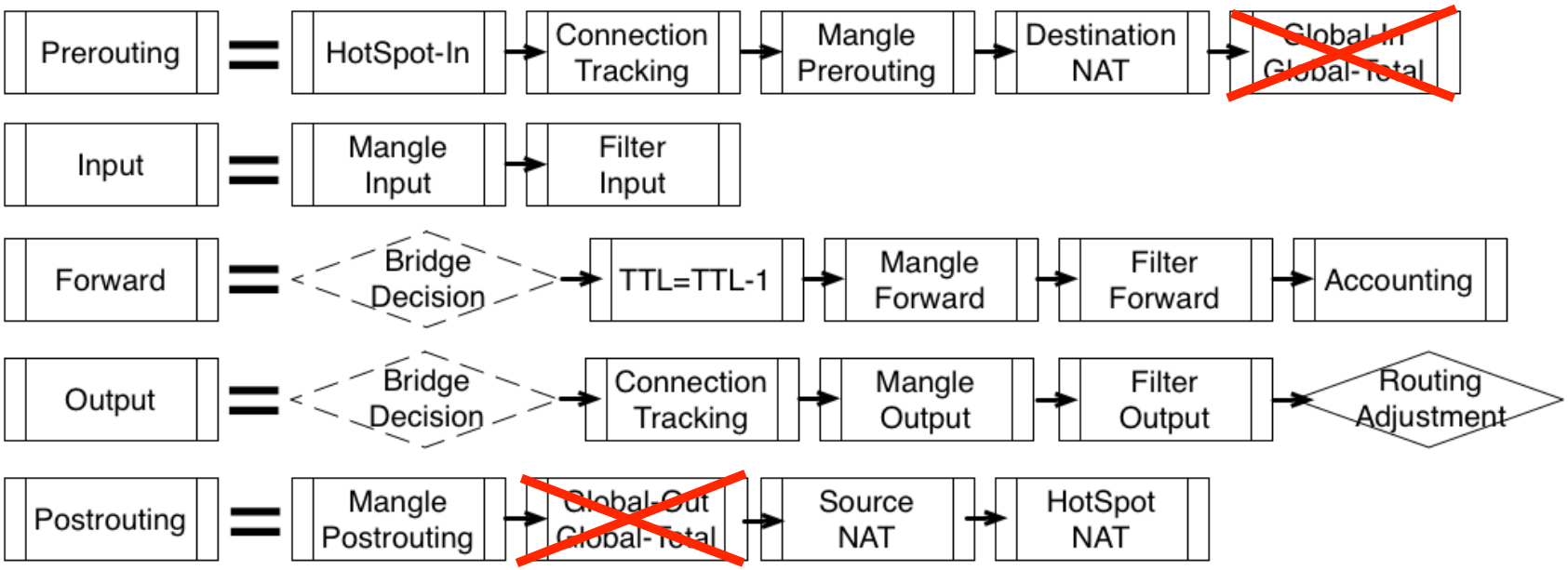
Queue & Multicore Processing

- Packets spend most part of the processing time waiting in queues.
- In order not to waste CPU core cycles on waiting, current core will just leave packet in the queue.
- Packet will be taken out of the queue by random CPU core, that works on that queue at the time.
- In short: queues shuffles packet assignments to CPU cores.

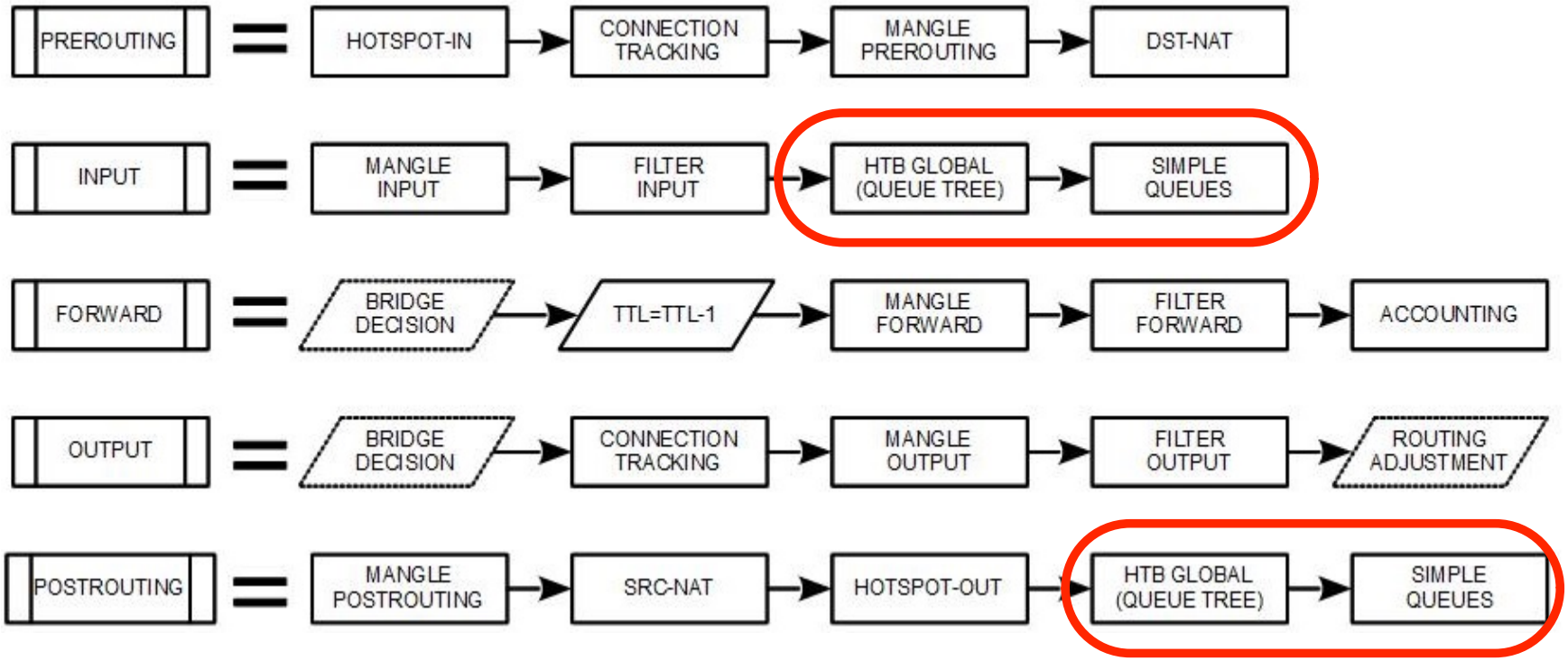
Packet Flow Change

- In RouterOS v5.x there was several places where packets were queued, so CPU core assignments shuffle happened several times
- In RouterOS v6.x QoS system was redesigned so that queuing happens in the same place respectively to other processes in the router – at the end.

HTB in RoS v5


Prerouting
HotSpot-In
Connection Tracking
Mangle Prerouting
Destination NAT
Input
Mangle Input
Filter Input
Forward
Bridge Decision
TTL=TTL-1
Mangle Forward
Filter Forward
Accounting
Output
Bridge Decision
Connection Tracking
Mangle Output
Filter Output
Routing Adjustment
Postrouting
Mangle Postrouting
Source NAT
HotSpot NAT

HTB in RoS v6


PREROUTING
HOTSPOT-IN
CONNECTION TRACKING
MANGLE PREROUTING
DST-NAT
INPUT
MANGLE INPUT
FILTER INPUT
HTB GLOBAL (QUEUE TREE)
SIMPLE QUEUES
FORWARD
BRIDGE DECISION
TTL=TTL-1
MANGLE FORWARD
FILTER FORWARD
ACCOUNTING
OUTPUT
BRIDGE DECISION
CONNECTION TRACKING
MANGLE OUTPUT
FILTER OUTPUT
ROUTING ADJUSTMENT
POSTROUTING
MANGLE POSTROUTING
SRC-NAT
HOTSPOT-OUT
HTB GLOBAL (QUEUE TREE)
SIMPLE QUEUES

Queue Tree on Multicore

- Whole HTB tree from Kernel perspective is and will be one queue so only one CPU core can work on HTB at the same time.
- Same optimization as in simple queues (at least 32 top-level queues, faster matching) will come to queue tree in one of the next versions.
- Suggestions:
  - Use Interface HTB as much as possible to offload traffic from HTB “global”
  - Use simple queues

Queue Change in 6.19

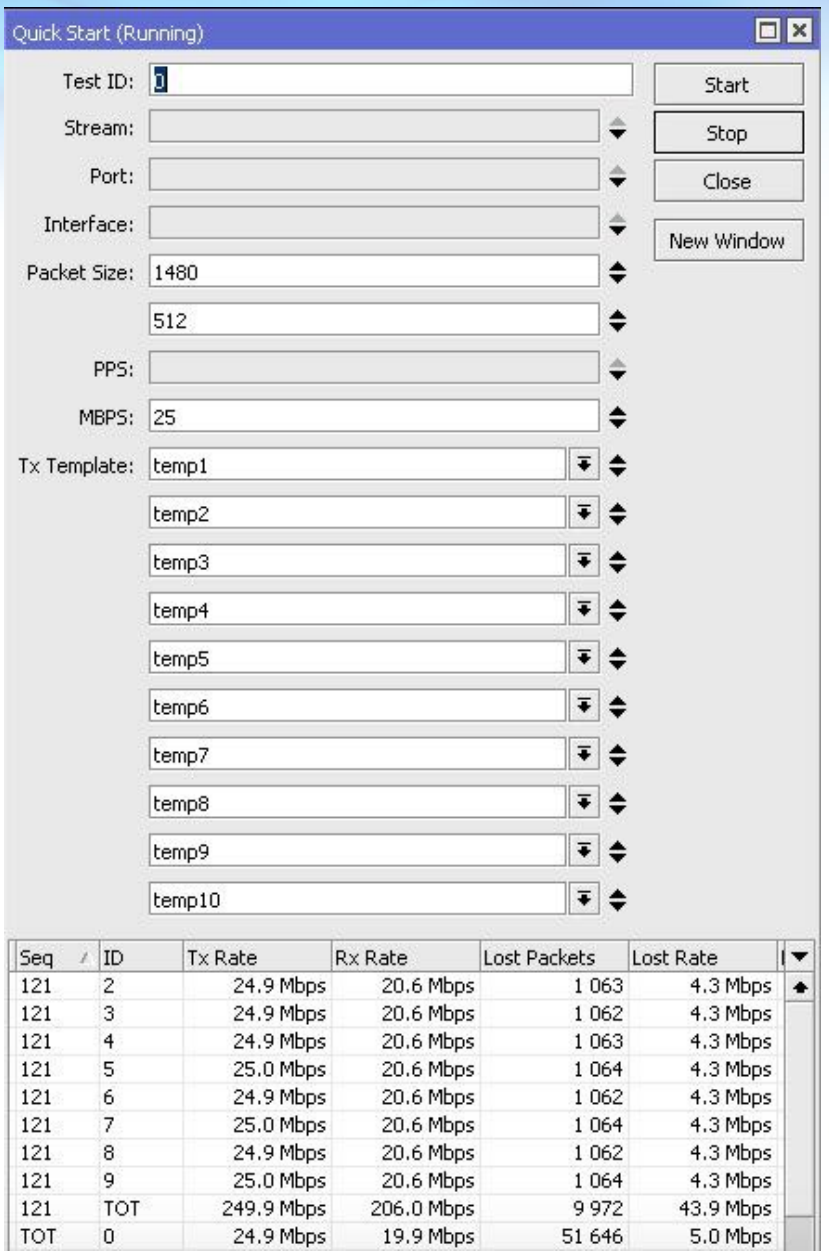
- In RouterOS v6.19 MikroTik introduced a software patch to improve queue performance
- Before CPU core just left packets in the queue and random other core was taking them out “later”.
- Now CPU core that leave packets in the queue will have to take some packets out, in the same instant moment.
- In case when queue limit is not reached, same packets will be left in and taken out of the queue by the same CPU core, making this process seamless

Lab Test

2 CCR 1036 are doing traffic generator, through 1 CCR 1036, routing mode.

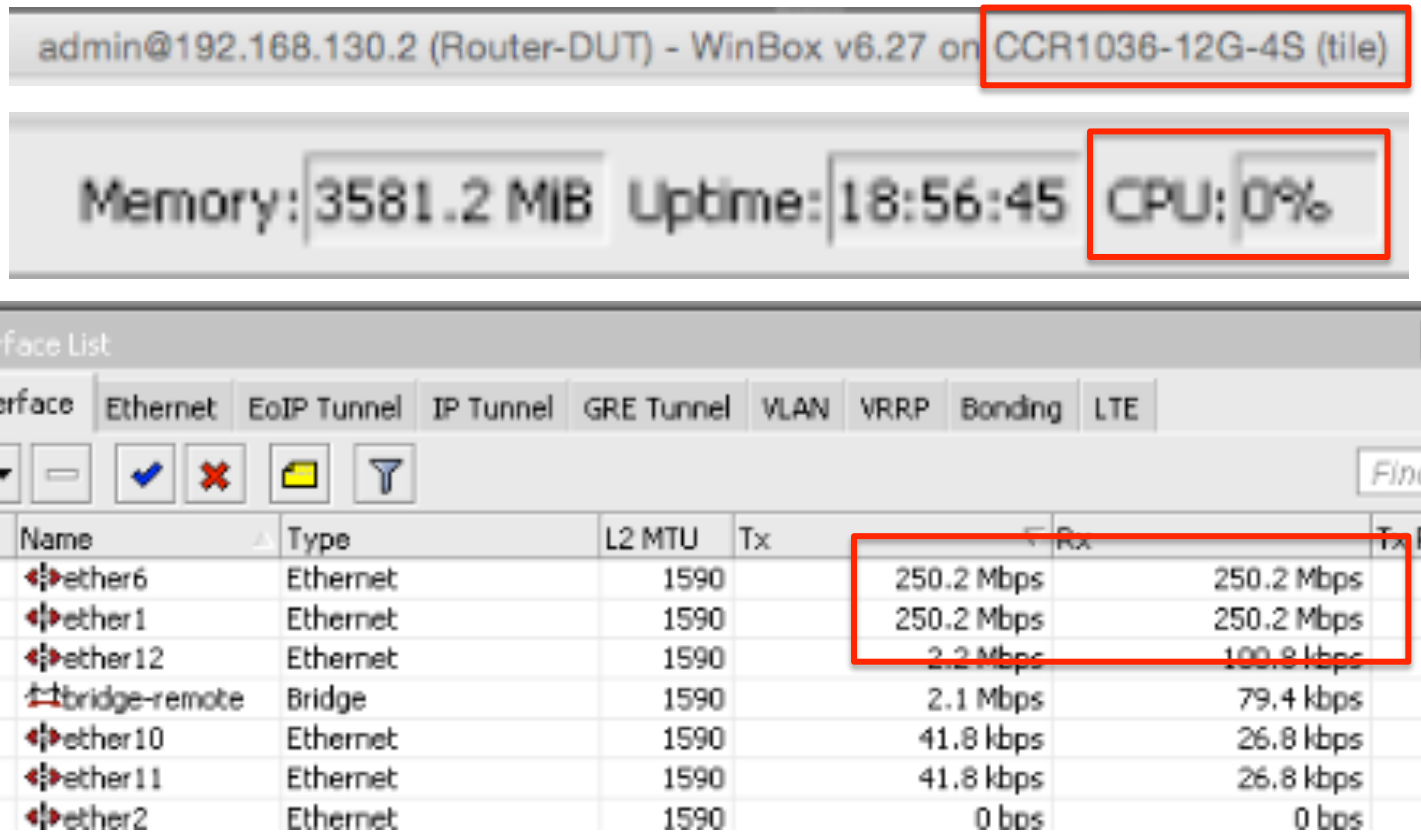
Traffic Generator

Each machine sending
10 streams
(each 25mbps),
from 10 different ip
addresses,
to 10 different ip
addreses.



Seq	ID	Tx Rate	Rx Rate	Lost Packets	Lost Rate
121	2	24.9 Mbps	20.6 Mbps	1 063	4.3 Mbps
121	3	24.9 Mbps	20.6 Mbps	1 062	4.3 Mbps
121	4	24.9 Mbps	20.6 Mbps	1 063	4.3 Mbps
121	5	25.0 Mbps	20.6 Mbps	1 064	4.3 Mbps
121	6	24.9 Mbps	20.6 Mbps	1 062	4.3 Mbps
121	7	25.0 Mbps	20.6 Mbps	1 064	4.3 Mbps
121	8	24.9 Mbps	20.6 Mbps	1 062	4.3 Mbps
121	9	25.0 Mbps	20.6 Mbps	1 064	4.3 Mbps
121	TOT	249.9 Mbps	206.0 Mbps	9 972	43.9 Mbps
TOT	0	24.9 Mbps	19.9 Mbps	51 646	5.0 Mbps

Without any configuration



Mangle

- We need to create firewall mangle if we use queue tree to marking the packet for each queue.

```
/ip firewall mangle

add action=mark-packet chain=prerouting \
new-packet-mark=packet-src-0.255 \
passthrough=no src-address=172.16.0.255

add action=mark-packet chain=prerouting \
new-packet-mark=packet-dst-0.255 \
passthrough=no dst-address=172.16.0.255
```

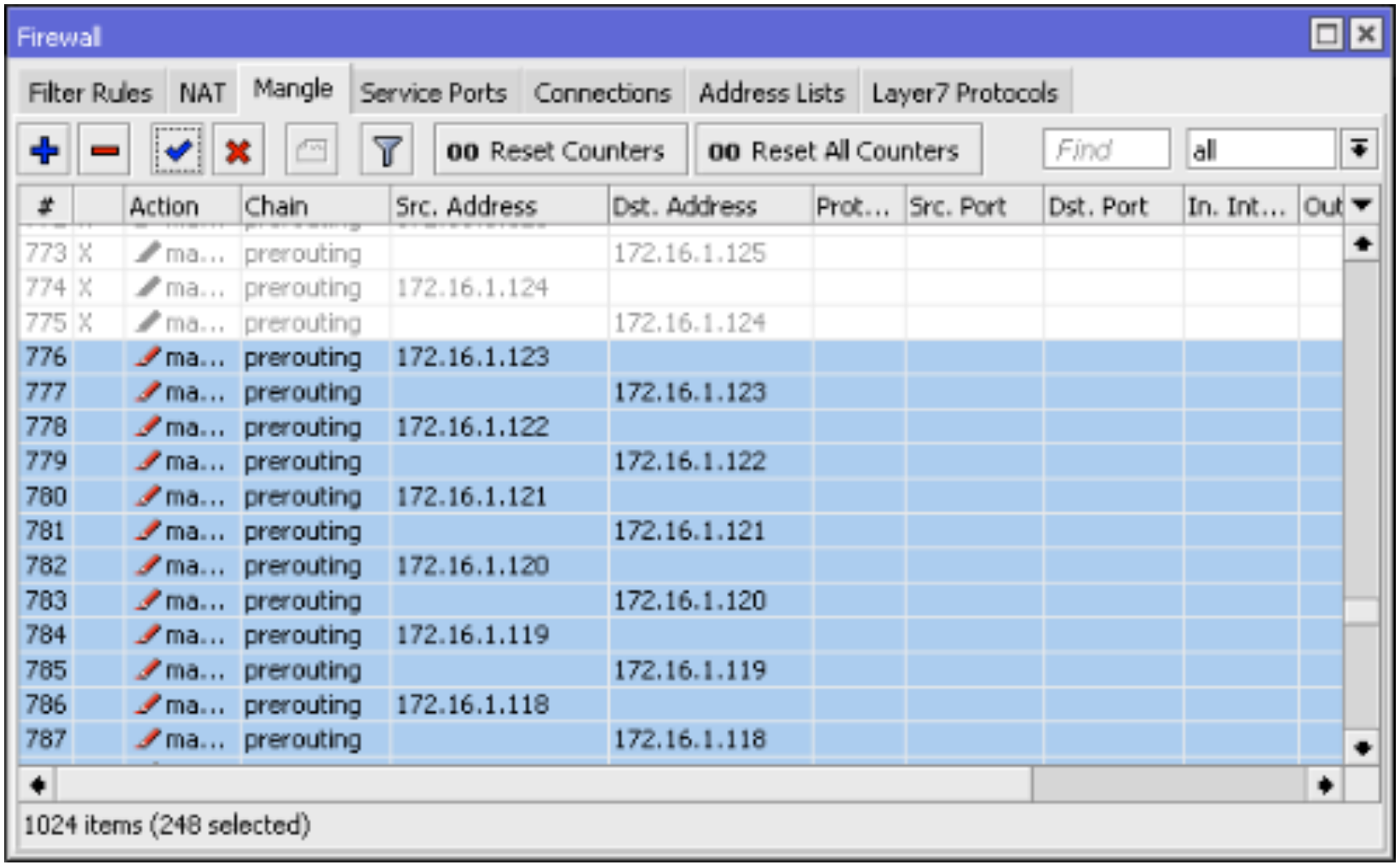
- Correlation between packet-mark and cpu-load?

Where Conn-Mark?

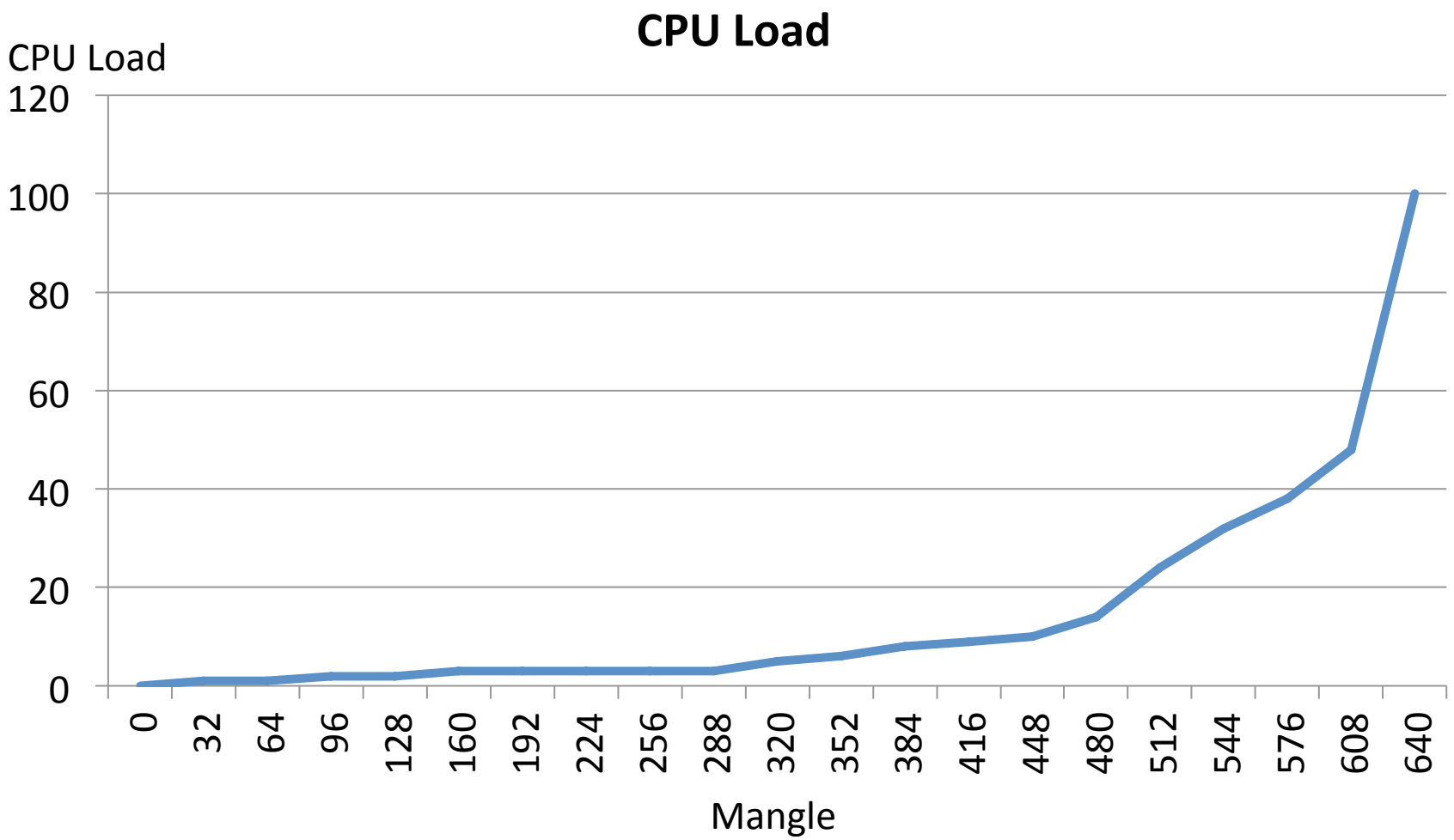
- Should we use connection-mark before packet-mark in firewall mangle?

- Yes.
  But in this lab test, we want to see with how many plain packet-marks a CCR can survive.
  Later, I test with conn-mark, almost same result.

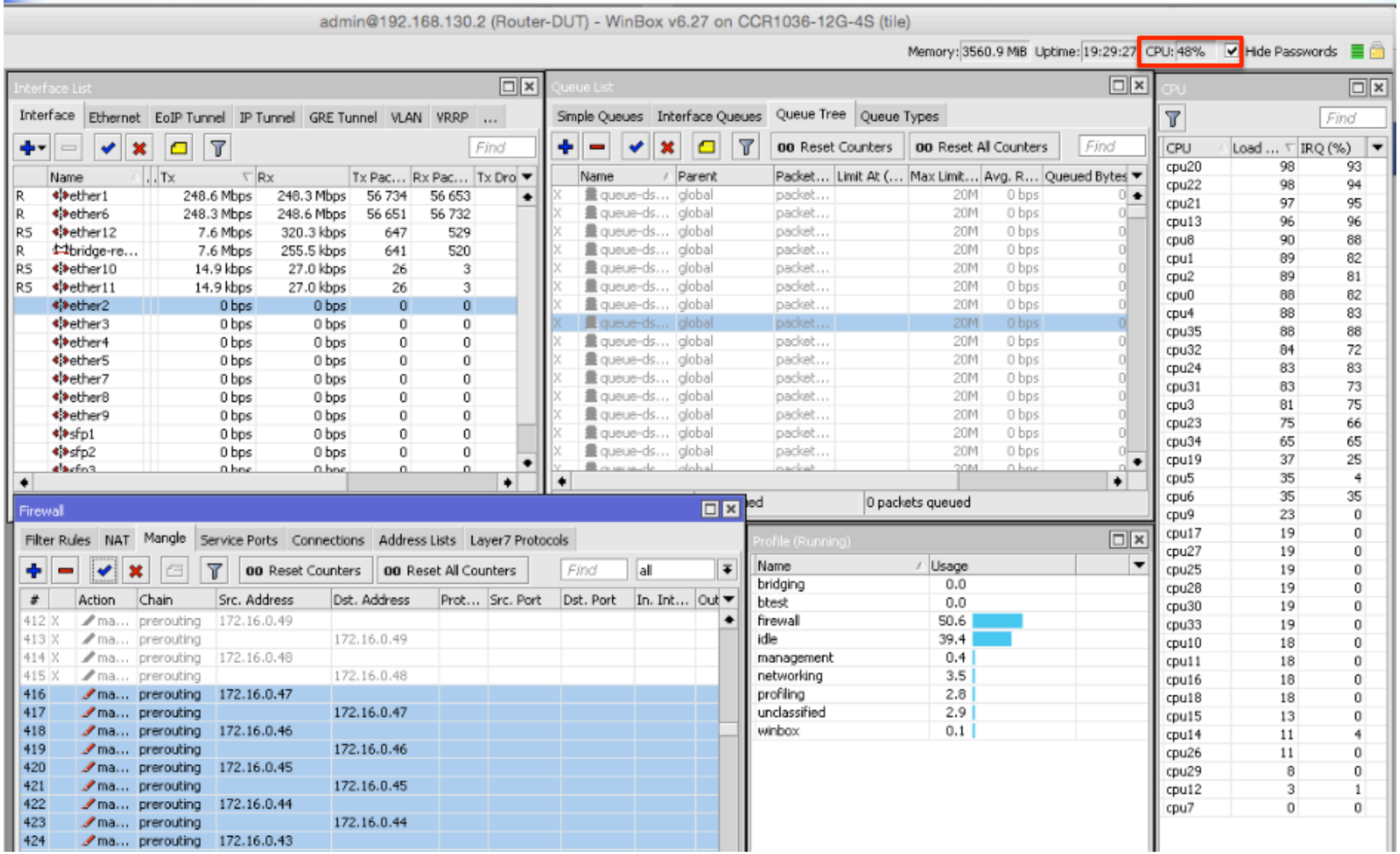
Firewall - Mangle



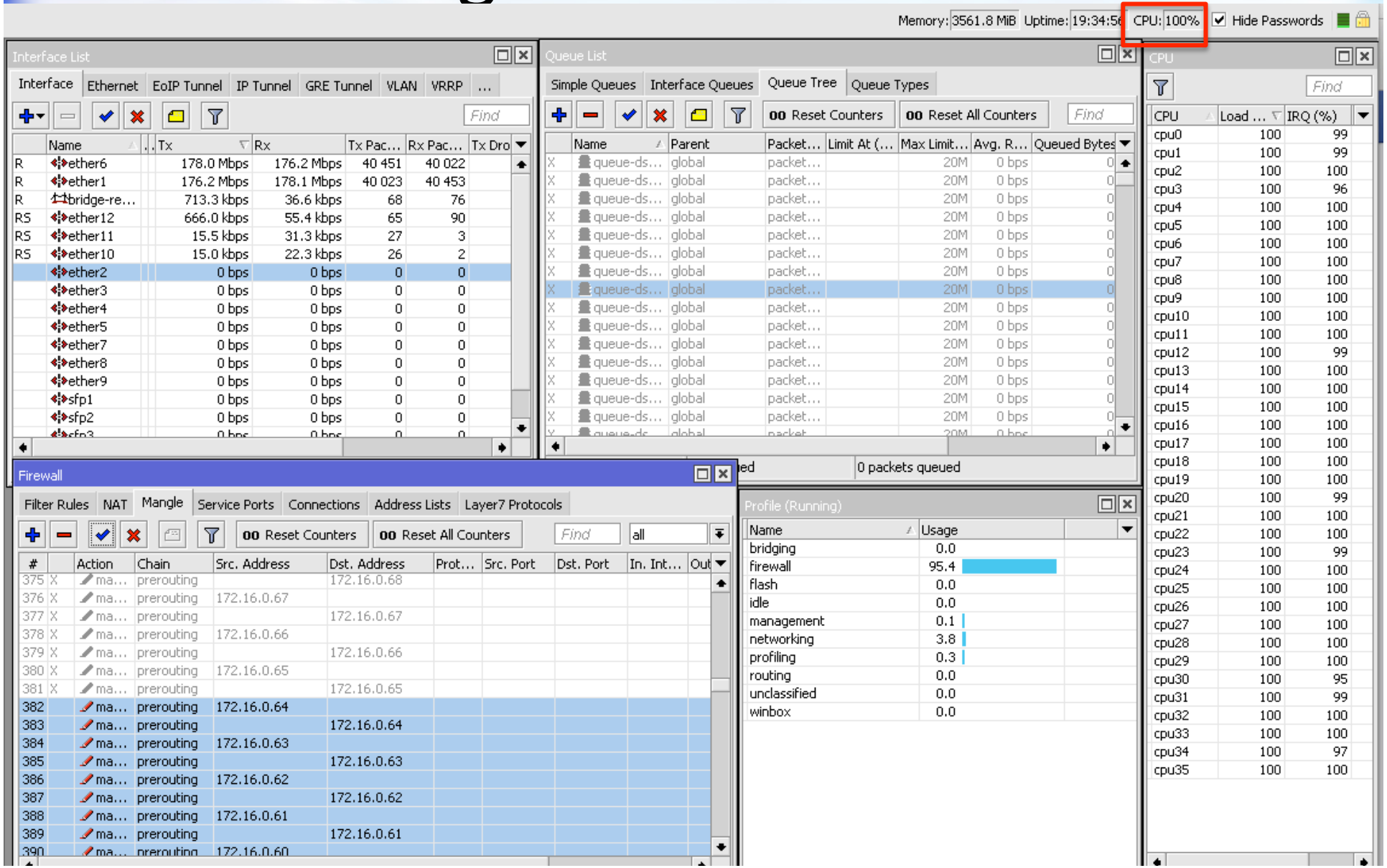
Graphs


CPU Load
CPU Load
120
100
80
60
40
20
0
0
32
64
96
128
160
192
224
256
288
320
352
384
416
448
480
512
544
576
608
640
Mangle

608 Mangle → 48% CPU Load



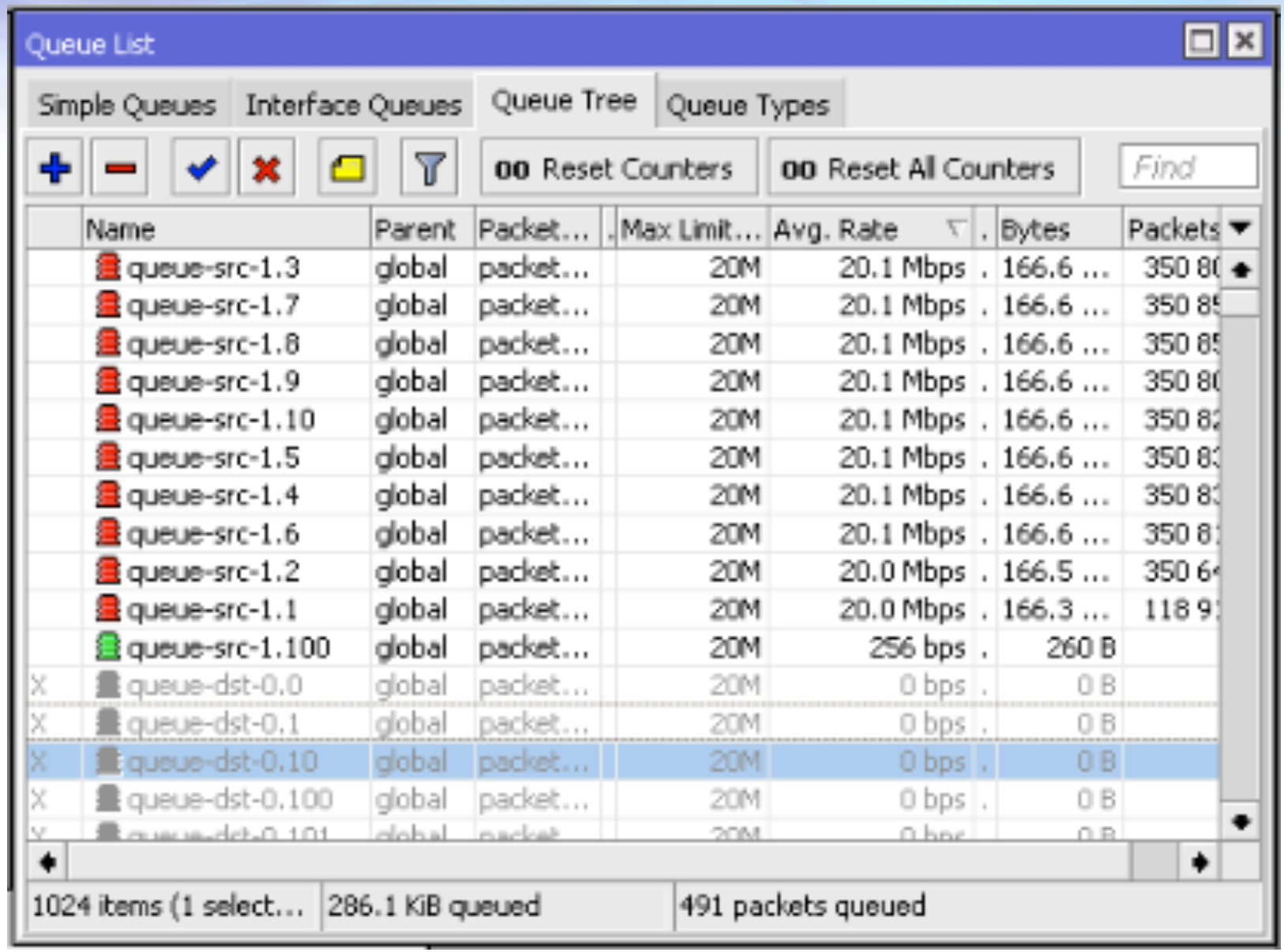
640 mangles → 100% CPU load



What next?

- The previous test is only mangle (packet-mark), we need to test with queue tree (512 rules).
- We use only 512 mangles, it was 24% CPU load.

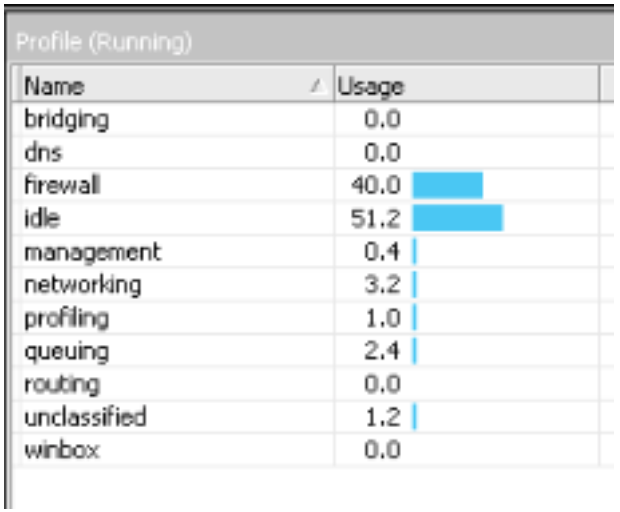
```
/queue tree

add max-limit=20M name=queue-src-1.1
packet-mark=packet-src-1.1 parent=global
queue=default

add max-limit=20M name=queue-dst-1.1
packet-mark=packet-dst-1.1 parent=global
queue=default
```


Queue List
Simple Queues
Interface Queues
Queue Tree
Queue Types
Reset Counters
Reset All Counters
Find
Name
Parent
Packet...
Max Limit...
Avg. Rate
Bytes
Packets
queue-src-1.3 global packet... 20M 20.1 Mbps 166.6 ... 350 80
queue-src-1.7 global packet... 20M 20.1 Mbps 166.6 ... 350 85
queue-src-1.8 global packet... 20M 20.1 Mbps 166.6 ... 350 85
queue-src-1.9 global packet... 20M 20.1 Mbps 166.6 ... 350 80
queue-src-1.10 global packet... 20M 20.1 Mbps 166.6 ... 350 82
queue-src-1.5 global packet... 20M 20.1 Mbps 166.6 ... 350 83
queue-src-1.4 global packet... 20M 20.1 Mbps 166.6 ... 350 83
queue-src-1.6 global packet... 20M 20.1 Mbps 166.6 ... 350 81
queue-src-1.2 global packet... 20M 20.0 Mbps 166.5 ... 350 64
queue-src-1.1 global packet... 20M 20.0 Mbps 166.3 ... 118 91
queue-src-1.100 global packet... 20M 256 bps 260 B
X queue-dst-0.0 global packet... 20M 0 bps 0 B
X queue-dst-0.1 global packet... 20M 0 bps 0 B
X queue-dst-0.10 global packet... 20M 0 bps 0 B
X queue-dst-0.100 global packet... 20M 0 bps 0 B
1024 items (1 select...
286.1 KiB queued
491 packets queued

CPU Load

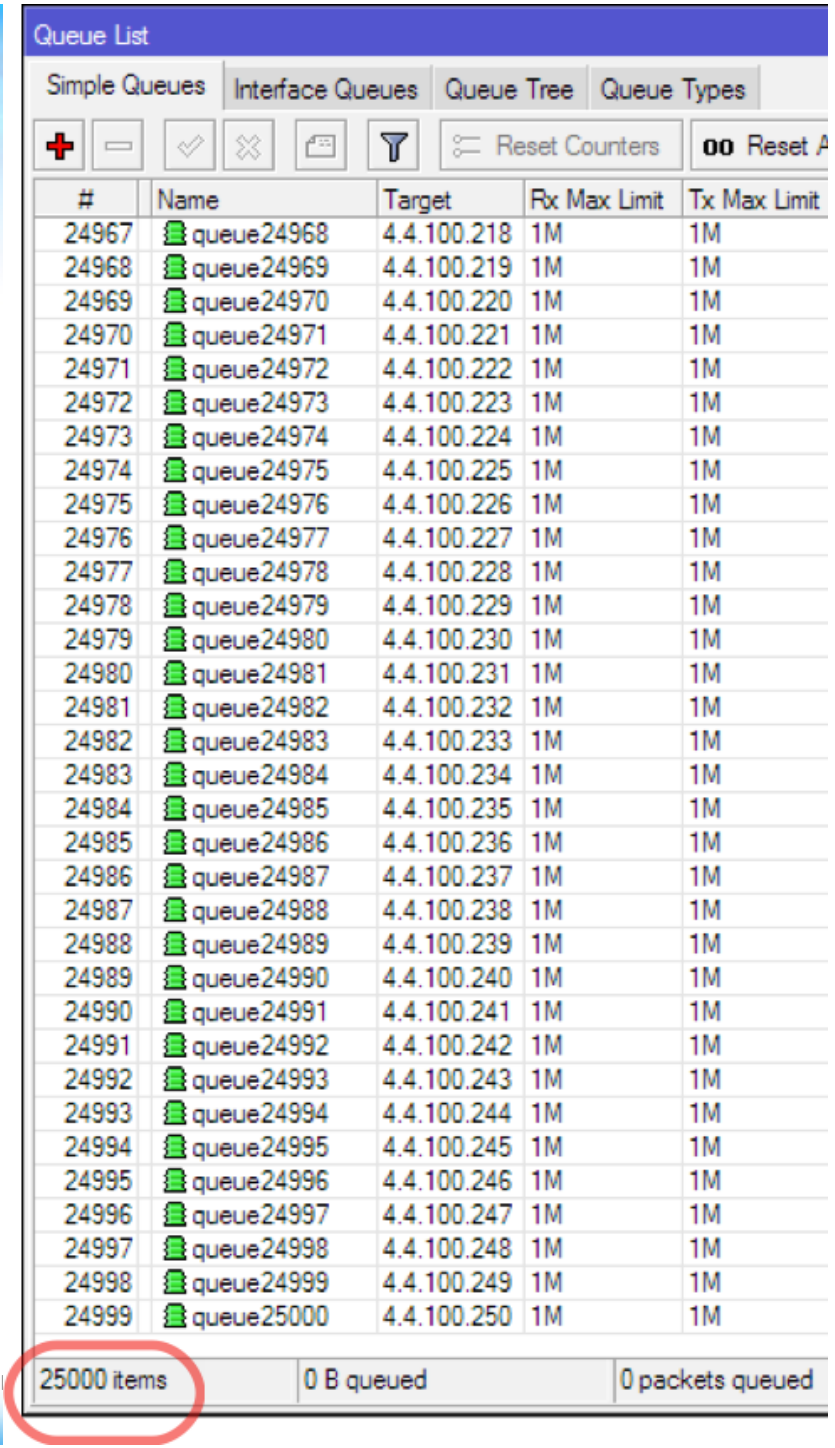
- With 512 mangles + without queue tree, we had 24% CPU load.
- With 512 mangles + 512 queue tree, we had 43% CPU Load (almost double).

- But, on Tools – Profile, load for queue is still low.



Simple Queue

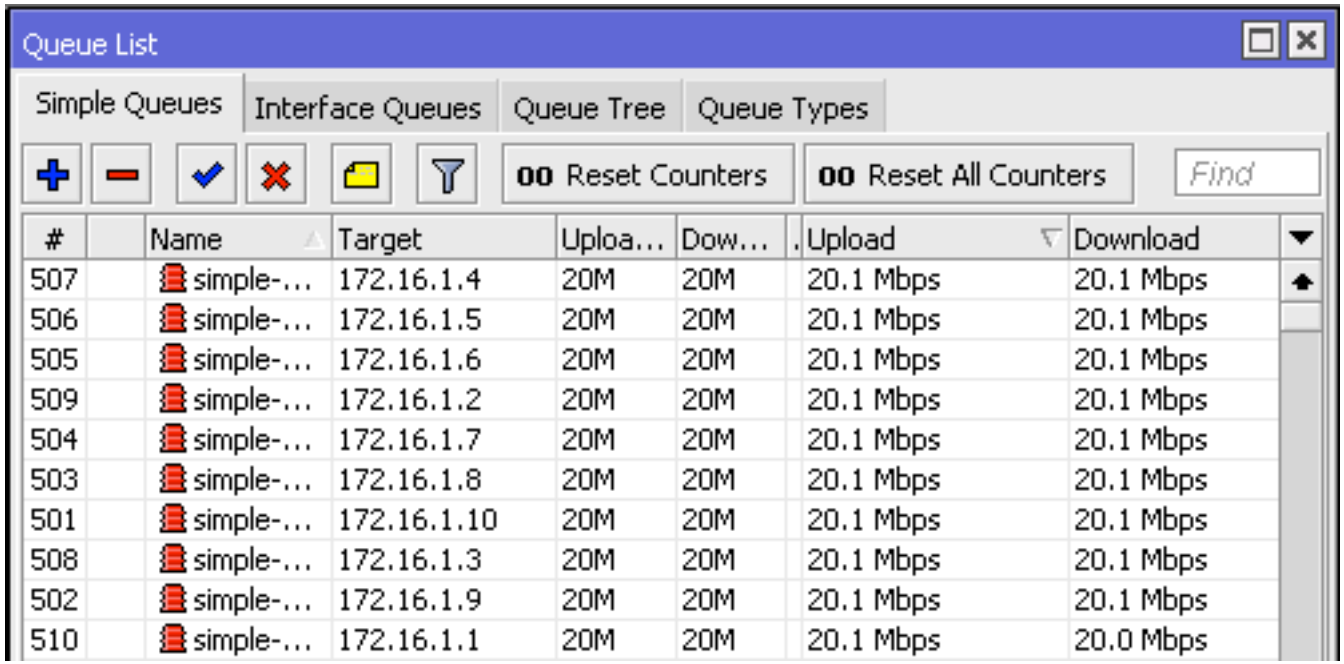
- Matching algorithm has been updated
  - based on hash
  - faster miss-matches
- Optimal performance on Multi-core devices requires at least 32 top level simple queues, so that queuing process can be distributed properly



Let's try Simple Queue

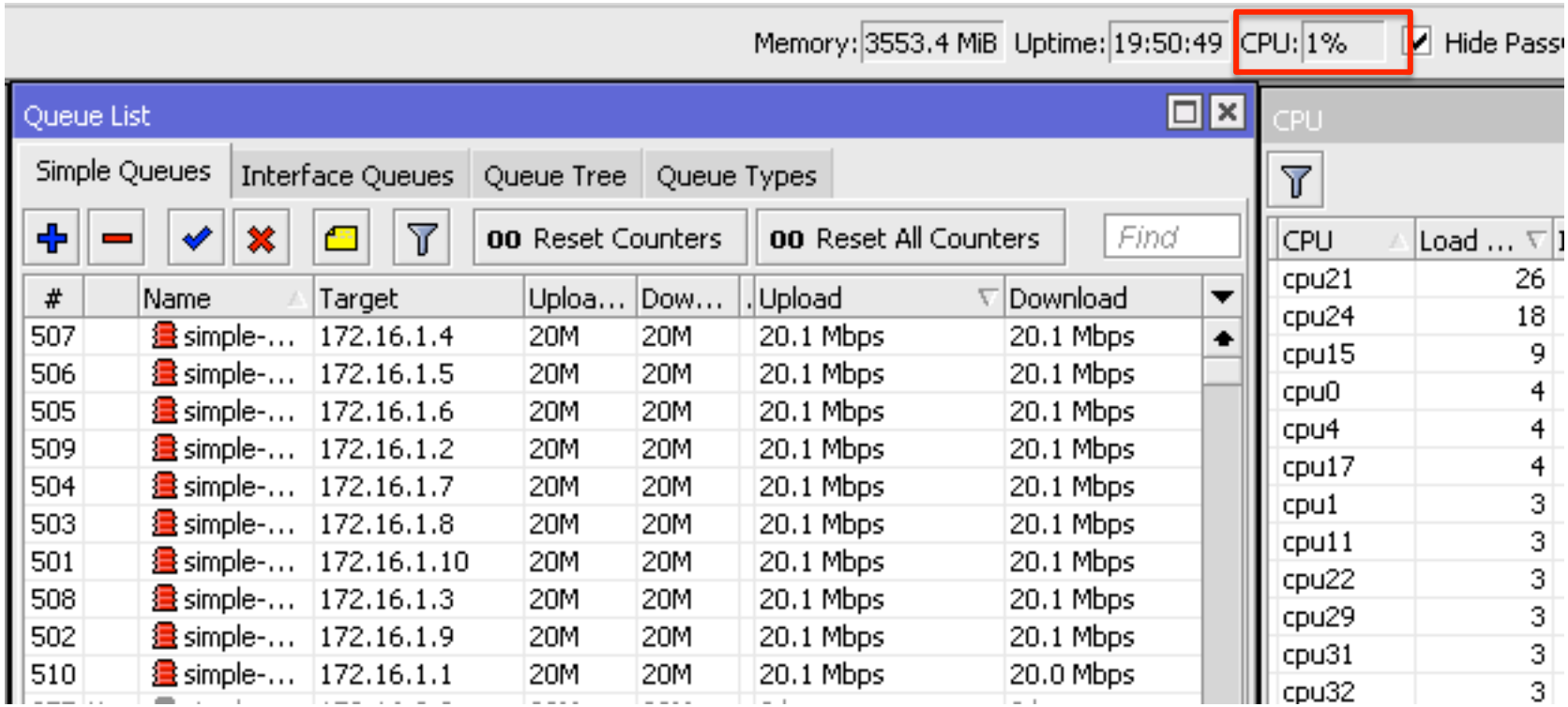
- We add 512 simple queues :

```
/queue simple add max-limit=20M/20M
name=simple-queue-1.1  target=172.16.1.1/32
```



#	Name	Target	Uploa...	Dow...	Upload	Download
507	simple-...	172.16.1.4	20M	20M	20.1 Mbps	20.1 Mbps
506	simple-...	172.16.1.5	20M	20M	20.1 Mbps	20.1 Mbps
505	simple-...	172.16.1.6	20M	20M	20.1 Mbps	20.1 Mbps
509	simple-...	172.16.1.2	20M	20M	20.1 Mbps	20.1 Mbps
504	simple-...	172.16.1.7	20M	20M	20.1 Mbps	20.1 Mbps
503	simple-...	172.16.1.8	20M	20M	20.1 Mbps	20.1 Mbps
501	simple-...	172.16.1.10	20M	20M	20.1 Mbps	20.1 Mbps
508	simple-...	172.16.1.3	20M	20M	20.1 Mbps	20.1 Mbps
502	simple-...	172.16.1.9	20M	20M	20.1 Mbps	20.1 Mbps
510	simple-...	172.16.1.1	20M	20M	20.1 Mbps	20.0 Mbps

With Simple Queue



It's only 1% of CPU Load with Simple Queue

Why Simple Queue?

- With simple queue, we don't need to use mangle (mangle required a lot of CPU resources).
- Simple Queue in v6 has more efficient hashing process.
- For non dedicated services, you can use burst feature.
- In Queue tree, all queue rules under one interface parent will processed by one CPU core.

Conclusions

- For low bandwidth network, overloaded pipe, use:
  - combination of packet-mark
  - queue tree, HTB, and burst
- For high throughput backbone access:
  - use multicore router:
    - CCR 36 – 72 core
    - Intel base quad core Xeon (8 thread)
  - simple queue (no parent)

Thank you

Comments and suggestions:

Valens Riyadi (valens@mikrotik.co.id)

@valensriyadi