



**TITANIA**<sup>®</sup>  
NETWORKS LTD

# **HTB Network Packet Scheduler implementation:** **experimenting with the new bucket size feature**

©2017 Alfredo Giordano, Matthew Ciantar  
Titania Networks Limited



# Alfredo Giordano

- MikroTik Certified Trainer.
- Certified consultant for MikroTik and other Brands.
- Specialized in ISP, WISP and corporate Network development.
- Working with MikroTik solutions since 2006.
- In Telecommunications since 2001.
- Active Member of RIPE and administrator for several AS
- Master degree in Electronic Engineering at Polytechnic of Turin, IT and university of Illinois of Chicago, USA.



# Matthew Ciantar

- MikroTik Certified Trainer.
- Certified consultant for MikroTik – (MTCNA, MTCTCE, MTCWE, MTCRE, MTCINE, MTCIPv6E)
- Certified consultant for Cisco – (CCNA, CCNP)
- Microsoft Certified Professional - Enterprise Administrator (MCITP)
- Experience with Service Provider, and Betting Industry for providing robust and highly available infrastructures.
- Over 15 years experience with Mikrotik RouterOS and RouterBoards

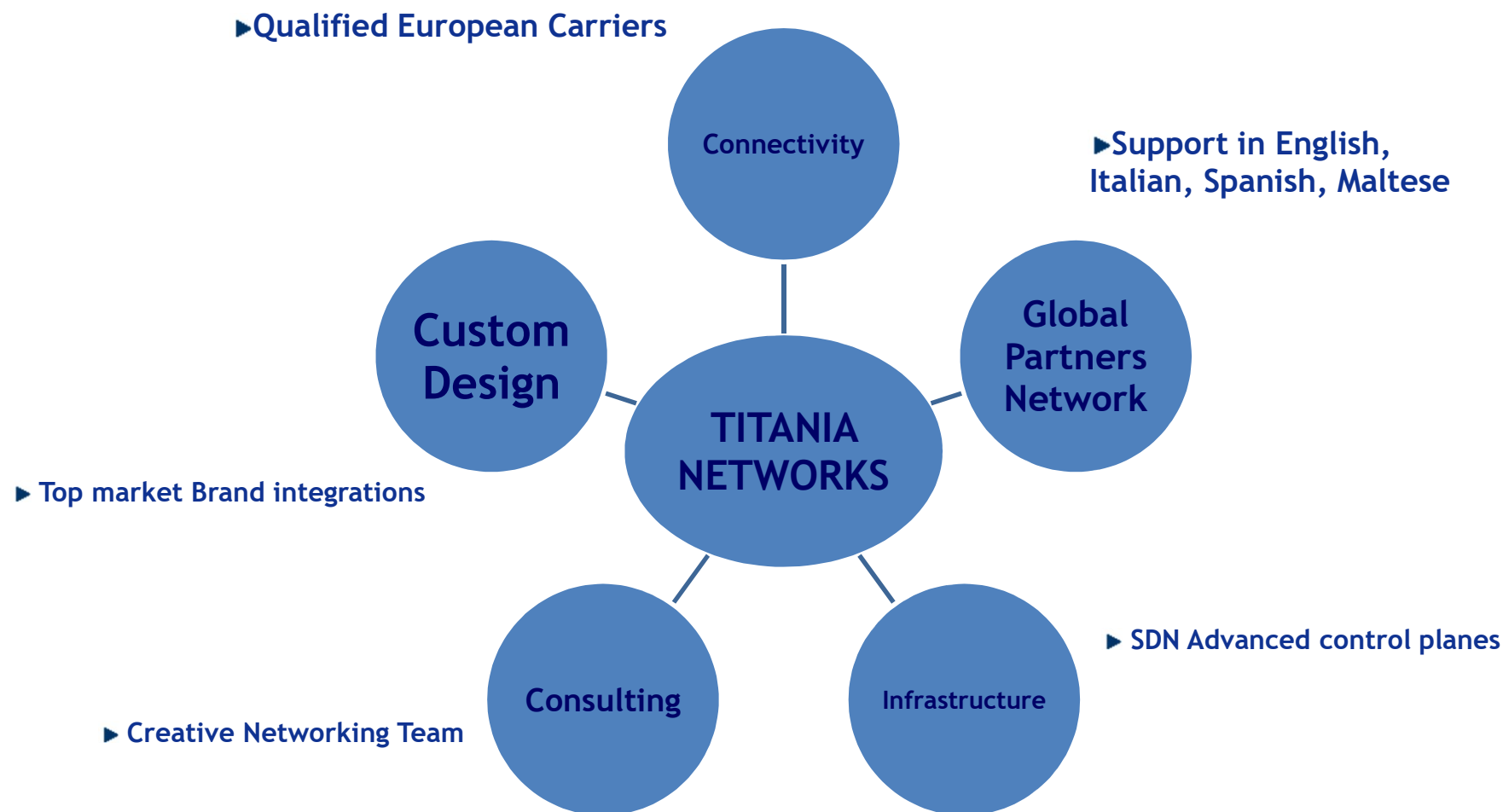


# Titania Networks

- Started in 2014, by providing IT Training and Consultancy under the brand tiktrain.com
- Incorporated in 2015 as a company in Ireland started operations in Europe.
- Most requested services:
  - Mission critical Networking consulting
  - ISP Design
  - Network Training
- Operation area:
  - Europe (Ireland, Malta, Italy, the Netherlands, Spain)
  - Latin America



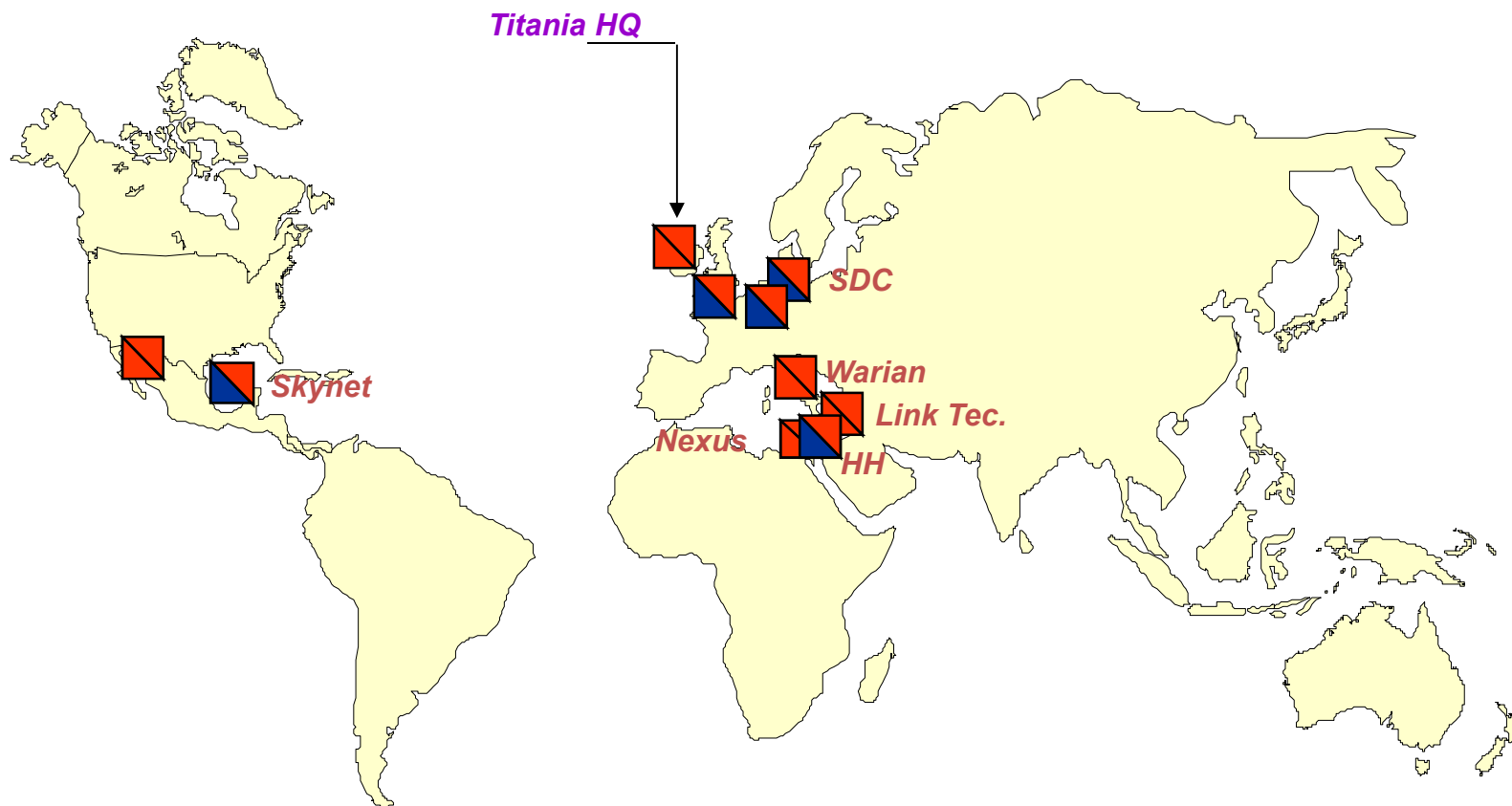
# Operations



- With a single point of contact.



# Customers



-  Consulting Customer
-  Partner with Infrastructure facility



# Goals

- With this presentation we seek to explain:
  - Concepts involving HTB
  - New features introduced
  - What you can achieve
- Trying to get a holistic picture from the available documentation



# Topics

- Concepts
- The linux kernel `queue_run()`
- Queuing
  - Queue type / kind / size
- The HTB Algorithm
  - Token / Buckets
  - Classes
  - Putting everything together
- Configuration basics
- Real-time Lab demonstration





# Concepts

- Scheduling
- Shaping
- Queue



# The truth about Queues

- Queues are located between the system and the interface and determine how data is SENT from the interface itself.
- Queues can be used to buffer the excess of output bandwidth to prevent packet loss in case of bursts – and this is generally GOOD
- TCP/IP, because of the way it works, will try to fill any queue you offer it. Queues create latency that affects interactivity for example when your keystroke must traverse a long queue – and this is generally BAD



# Proof of Concept

The screenshot displays two RouterOS WinBox windows. The left window shows the 'Quick Start (Running)' dialog with a table of traffic statistics. The right window shows the 'Interface List' and 'Simple Queue' configuration.

**Quick Start (Running) Traffic Statistics Table:**

Seq	ID	Tx Packets	Tx Rate	Rx Packets	Rx Rate	Lost Packets	Lost F
55	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
56	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
57	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
58	0	27 985	14.9 Mbps	100	76.0 kbps	27 885	14.9
59	0	27 987	15.0 Mbps	100	76.0 kbps	27 887	14.9
60	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
61	0	27 985	14.9 Mbps	100	76.0 kbps	27 885	14.9
62	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
63	0	27 984	14.9 Mbps	100	76.0 kbps	27 884	14.9
64	0	27 988	15.0 Mbps	100	76.0 kbps	27 888	14.9
65	0	27 985	14.9 Mbps	100	76.0 kbps	27 885	14.9
66	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
67	0	27 987	15.0 Mbps	100	76.0 kbps	27 887	14.9
68	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
69	0	27 985	14.9 Mbps	100	76.0 kbps	27 885	14.9
70	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
71	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
72	0	27 985	14.9 Mbps	100	76.0 kbps	27 885	14.9
73	0	27 986	15.0 Mbps	100	76.0 kbps	27 886	14.9
TOT	0	2 042 835	14.9 Mbps	7 353	76.5 kbps	2 035 482	14.9

**Interface List:**

Name	Type	Actual MTU	L2 MTU	Tx	Rx
ether1	Ethernet	1500	1598	1792 bps	0 bps
ether2	Ethernet	1500	1598	0 bps	0 bps
ether3	Ethernet	1500	1598	65.5 kbps	10.4 kbps
ether4	Ethernet	1500	1598	79.9 kbps	16.0 Mbps
wlan1	Wireless ...	1500	1600	0 bps	0 bps

**Simple Queue Configuration:**

- Rate: 5.4 Mbps (Target Upload), 63.7 kbps (Target Download)
- Packet Rate: 12 840 p/s (Target Upload), 98 p/s (Target Download)
- Legend: Upload: 5.4 Mbps, Download: 63.7 kbps
- Legend: Upload Packets: 12 840 p/s, Download Packets: 98 p/s



# The Linux Kernel

NK Installation of HTB on Slac... × +

lartc.vger.kernel.narkive.com/liqTFums/installation-of-htb-on-slack-v8-0

**NARKIVE**  
MAILINGLIST ARCHIVE

lartc@vger.kernel.org

Search...

Discussion:  
Installation of HTB on Slack v8.0 (too old to reply)

**Grzegorz Skrzypczak** 15 years ago

Hi All,

I'm looking for something to give me possibility to manage the bandwidth inside my FastEthernet LAN. I found HTB code but I don't know how to install it on my Slack v.8.0 with kernel version 2.4.18 (without any QoS functionality compiled in to kernel). Right now my network is running without any bandwidth limitations. Could any one help me to understand the idea of HTB's class hierarchy? What external pckeges should I download and install? Do I really need "iproute2"? How to compile HTB code (where is the source code to compile)?

Hope for helping me

Best regads,  
Grzegorz "Greg" Skrzypczak

**Alfredo Giordano** 15 years ago

Try following the following step:

- 1 Install the kernel source package
- 2 Read all the Doc about how compile the kernel
- 3 download the HTB patch code and apply it to your kernel source tree
- 4 download the patched tc binary and replace your original tc
- 5 compile the kernel with the QoS and HTB options compiled in the kernel (not as module)
- 6 read all the lartc howto

you should get trough

cheers  
alfredo

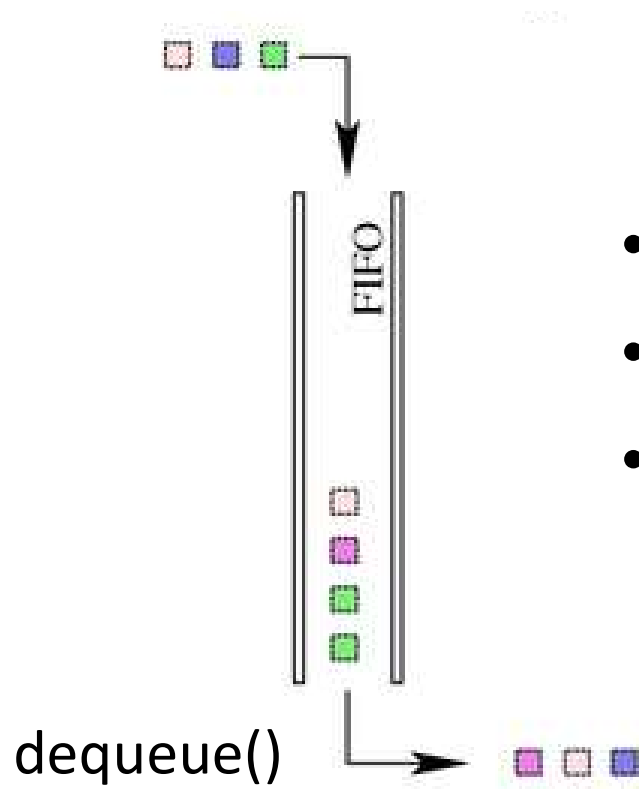


# The Linux Kernel

- The forwarding stack or the local process sends data to the kernel.
- Kernel enqueues data to the *queue-type* selected for the queue and immediately tries to run the queue to the hardware using `queue_run()`
- The function will call `dequeue()` according to the *queue-kind* algorithm to send to hardware (provided hardware can take as much).



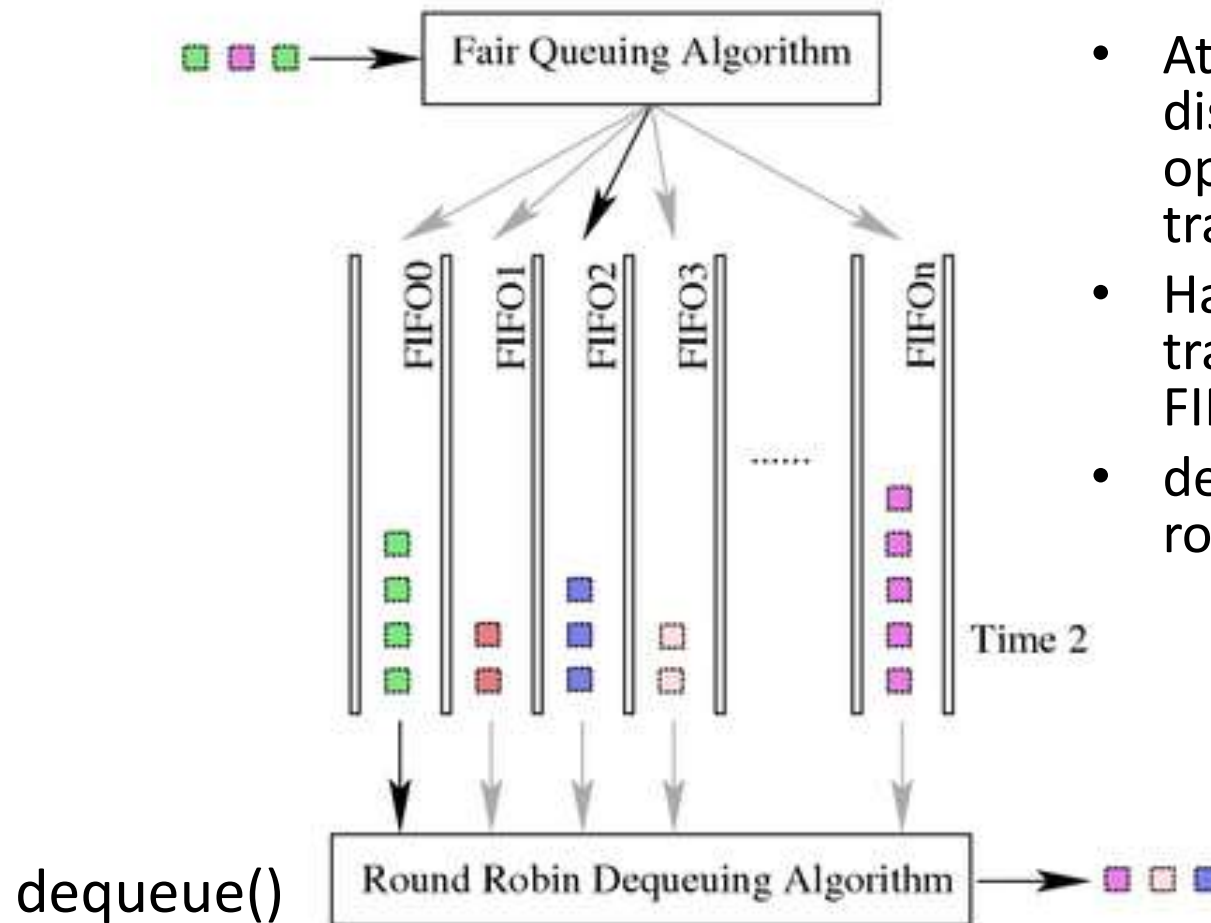
# PFIFO



- Simple Buffer
- Defined by *queue-size*
- Can be bytes or packets



# SFQ



- Attempt to distribute opportunity to transmit fairly
- Hash function to fit traffic in separate FIFOs
- dequeue() with round robin



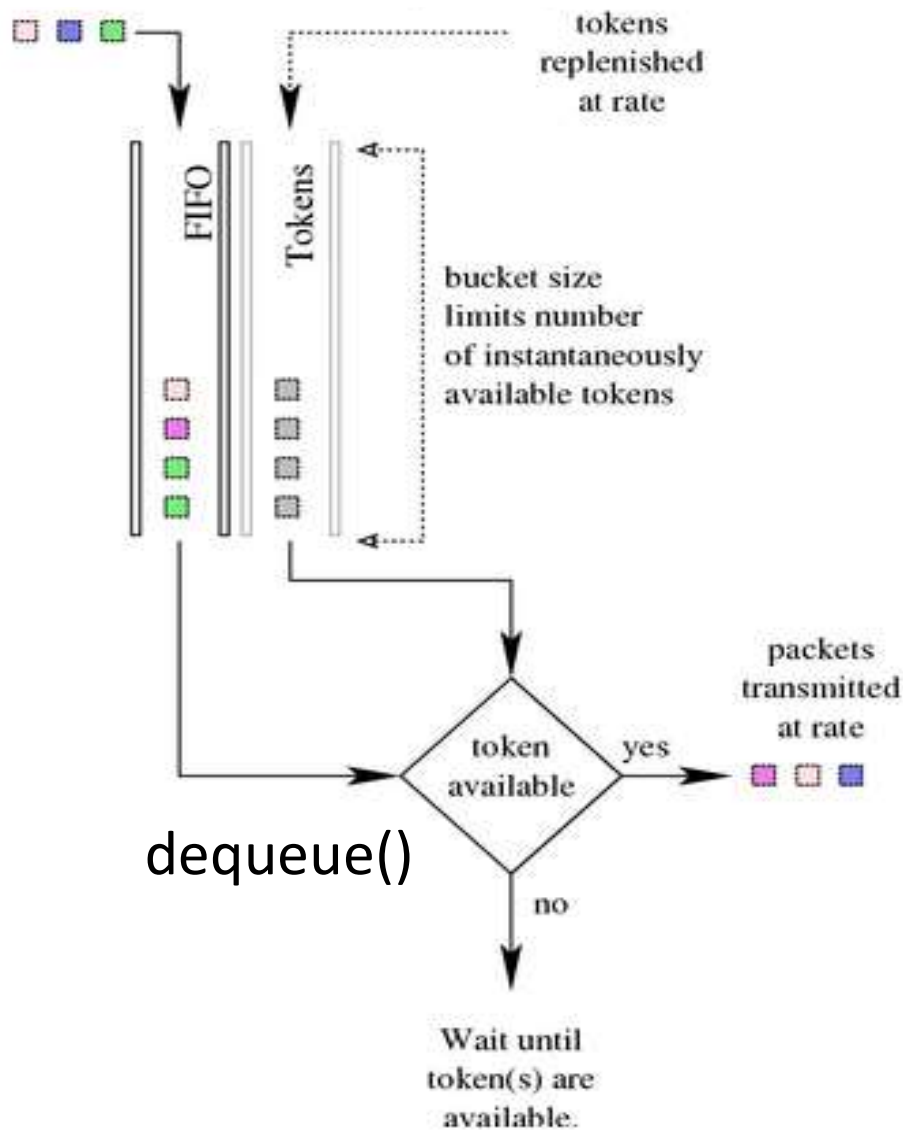
# Tokens

- Control the rate of dequeuing counting the number of packets/bytes dequeued is complex and timers dependent.
- Instead of calculating the current usage, one method, used widely in traffic control, is to generate tokens at a desired rate, and only dequeue packets or bytes if a token is available.





# Simple TBF



- Built on tokens and buckets
- Packets are only transmitted if there are sufficient tokens available.
- Otherwise, packets are deferred.
- It will introduce an artificial latency

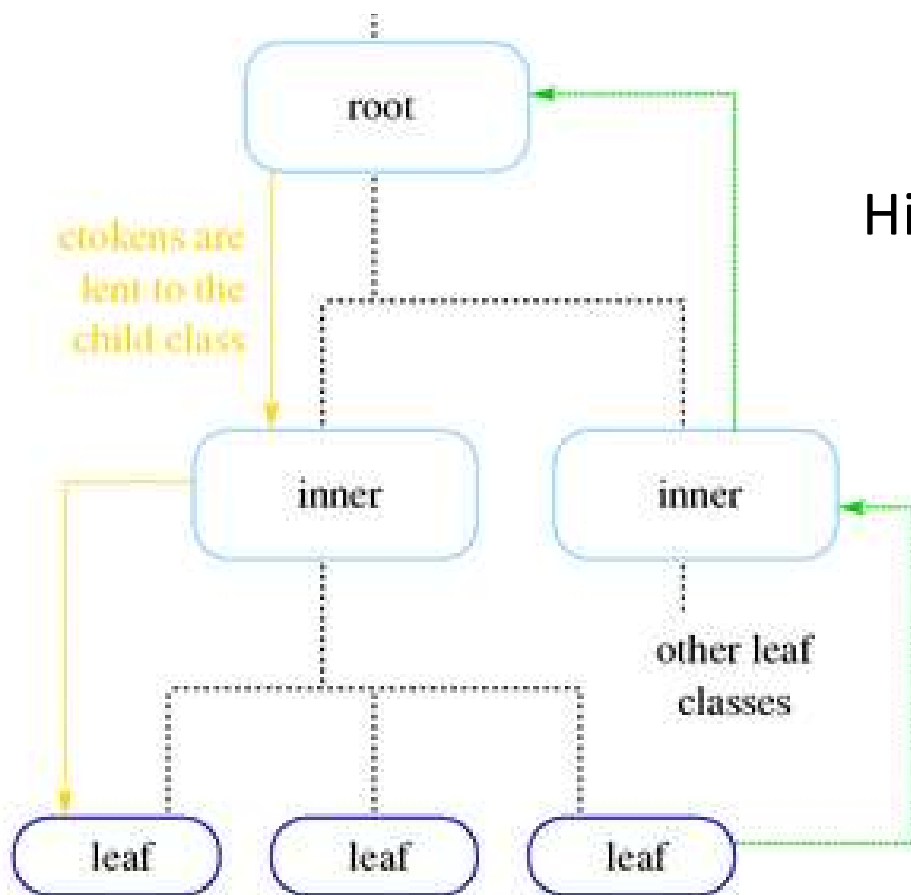


# Buckets

- In the case that a queue does not need tokens immediately, the tokens are collected until they are needed.
- The number of unused tokens that can be stored depends on the size of the bucket.
- A queue that has tokens available can initially dequeue a larger number of packets or bytes before tokens are depleted.



# HTB - Classes



Hierarchical class structure

tokens used in a child class are charged to parent classes

Note: HTB will not delay packets at inner level only leaf

dequeue()



# HTB - Classes

- Children classes borrow tokens from their parents once they have exceeded *limit-at*.
- A child class will continue to attempt to borrow until it reaches *max-limit*
- It will then begin to queue packets for transmission until more tokens are available.

type	rate	kernel action
child	< limit-at	dequeue() based on all available tokens
child	limit-at < rate < max-limit	dequeue() try to borrow tokens from parent
child	> max-limit	delay packets
parent	< limit-at	lend tokens to children
parent	limit-at < rate < max-limit	try to borrow from parent if any, lend to children
parent	> max-limit	no borrow, no lend



# HTB - Burst

- Burst is a feature that allows to satisfy queue requirement for additional bandwidth even if required rate is bigger than *max-limit* for a limited period of time.
- Burst can occur only if average-rate of the queue for the last *burst-time* seconds is smaller than *burst-threshold*.
- Burst mechanism is simple - if burst is allowed queue will receive tokens at *burst-limit* rate. When burst is disallowed queue will receive tokens at *max-limit* rate.

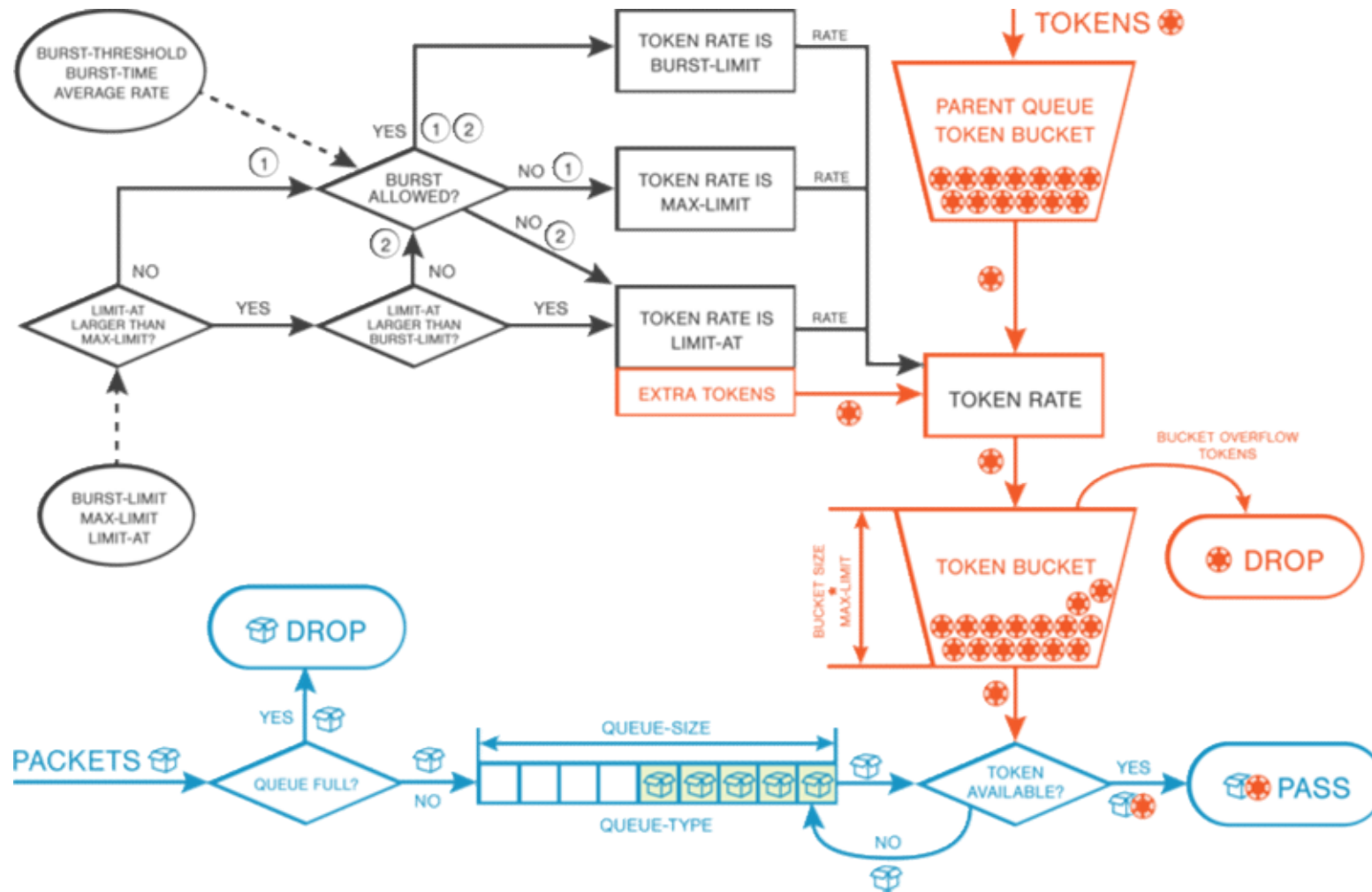


## HTB – Wrap up

- At the end of the day the amount of readily available tokens in the child class will define its behavior.
- We can set how fast the token replenish with the *max-limit* or *burst-limit*.
- Until RouterOS version 6.35 bucket size was hardcoded to  $\text{max-limit}/10$ . This is why default value is set to 0.1.
- **Now it will accept values from 0 to 10**



# The New HTB Diagram





# How much traffic will pass Unrestricted?

This is calculated as follows:

Bucket Capacity = *bucket-size* \* *max-limit* (or *burst-limit*, if *burst-limit* is being used)

Default Bucket with 10M *max-limit*

0.1 \* 10M = 1M

This will allow 1M of Data (Not bandwidth!) to go at  
***unrestricted speed!***





# Default Bucket Size

The screenshot displays two WinBox windows. The left window shows the results of a bandwidth test on the source side (192.168.88.1). The right window shows the interface configuration for ether4 on the target side (6C:3B:6B:1A:7C:5A).

**Terminal Output (Left Window):**

```
[admin@SOURCE] > /tool bandwidth-test address=172.16.1.254 protocol=tcp direction=receive
duration: 25s
rx-current: 9.6Mbps
rx-10-second-average: 9.6Mbps
rx-total-average: 8.9Mbps
random-data: no
direction: receive

[admin@SOURCE] > /tool bandwidth-test address=172.16.1.254 protocol=udp direction=receive
status: running
duration: 14s
rx-current: 9.6Mbps
rx-10-second-average: 9.6Mbps
rx-total-average: 9.7Mbps
random-data: no
direction: receive

[admin@SOURCE] > /tool bandwidth-test address=172.16.1.254 protocol=udp direction=receive
status: running
duration: 30s
rx-current: 11.5Mbps
rx-10-second-average: 10.1Mbps
rx-total-average: 10.3Mbps
lost-packets: 15
random-data: no
direction: receive
rx-size: 1500

[admin@SOURCE] > /tool bandwidth-test address=172.16.1.254 protocol=udp direction=receive
status: running
duration: 27s
rx-current: 10.0Mbps
rx-10-second-average: 10.0Mbps
rx-total-average: 9.9Mbps
lost-packets: 81
random-data: no
direction: receive
rx-size: 1500
```

**Interface Configuration (Right Window):**

Interface	Loop Protect	Overall Stats	Rx Stats	Tx Stats	Status	Traffic
Ethernet						
Tx/Rx Rate:		9.9 Mbps	1216 bps			
Tx/Rx Packet Rate:		823 p/s	2 p/s			
FP Tx/Rx Rate:		10.1 Mbps	1152 bps			
FP Tx/Rx Packet Rate:		843 p/s	2 p/s			
Tx/Rx Bytes:		2656.3 MiB	828.1 MiB			
Tx/Rx Packets:		2 345 923	12 170 570			
Tx/Rx Drops:		0	0			
Tx/Rx Errors:		0	0			

The interface configuration also includes two graphs. The top graph shows Tx (9.9 Mbps) and Rx (1216 bps) rates. The bottom graph shows Tx Packet (823 p/s) and Rx Packet (2 p/s) rates. A red circle highlights the Rx rate in the top graph.



# How much traffic will pass Unrestricted?

Large Bucket with 10M *max-limit*

$$10 * 10M = 100M$$

This will allow 100M of data to go at  
**unrestricted speed!!!**



# Large Bucket Size

The image shows two WinBox windows. The left window is the source router (192.168.88.1) running a bandwidth test. The right window is the target router (6C:3B:6B:1A:7C:5A) showing the interface statistics for ether4.

**Source Router Terminal Output:**

```
MikroTik RouterOS 6.37.5 (c) 1999-2017 http://www.mikrotik.com/

[?] Gives the list of available commands
command [?] Gives help on the command and list of arguments

[Tab] Completes the command/word. If the input is ambiguous,
a second [Tab] gives possible options

/ Move up to base level
.. Move up one level
/command Use command at the base level
[admin@SOURCE] > /tool bandwidth-test address=172.16.1.254 protocol=udp dir
ection=receive

status: running
duration: 37s
rx-current: 10.0Mbps
rx-10-second-average: 10.0Mbps
rx-total-average: 12.6Mbps
lost-packets: 84
random-data: no
direction: receive
rx-size: 1500
[Q quit|D dump|C-z pause]
```

**Target Router Interface Statistics (ether4):**

Category	Value
Tx/Rx Rate	9.9 Mbps / 656 bps
Tx/Rx Packet Rate	822 p/s / 1 p/s
FP Tx/Rx Rate	10.1 Mbps / 1776 bps
FP Tx/Rx Packet Rate	839 p/s / 3 p/s
Tx/Rx Bytes	2930.1 MiB / 828.7 MiB
Tx/Rx Packets	2 535 527 / 12 179 155
Tx/Rx Drops	0 / 0
Tx/Rx Errors	0 / 0

The interface status is shown as **running**. Below the statistics are two traffic graphs. The top graph shows Tx (blue) and Rx (red) rates, with a red circle highlighting a sharp spike in Rx traffic. The bottom graph shows Tx Packet (blue) and Rx Packet (red) rates, also showing a spike in Rx packets.



# What if one wants a ceiling for the burst?

Bucket size work like burst but without a burst-limit. To be able to force a ceiling, we can set a max-limit on the parent queue.

Small Bucket with 20M *max-limit* on the parent with a child having a Large Bucket with 10M *max-limit*.



# Burst with Ceiling

The image shows two WinBox windows. The left window is connected to the source IP 192.168.88.1 and displays the terminal output of a bandwidth test. The test shows a burst of traffic reaching 10.0 Mbps, with a 10-second average of 10.0 Mbps and a total average of 11.6 Mbps. The test parameters are: address=172.16.1.254, protocol=udp, direction=receive, rx-size=1500, and rx-current=10.0Mbps.

```
MikroTik RouterOS 6.37.5 (c) 1999-2017 http://www.mikrotik.com/

[?] Gives the list of available commands
command [?] Gives help on the command and list of arguments

[Tab] Completes the command/word. If the input is ambiguous,
a second [Tab] gives possible options

/ Move up to base level
.. Move up one level
/command Use command at the base level
[admin@SOURCE] > /tool bandwidth-test address=172.16.1.254 protocol=udp dir
ection=receive
status: running
duration: 1m
rx-current: 10.0Mbps
rx-10-second-average: 10.0Mbps
rx-total-average: 11.6Mbps
lost-packets: 82
random-data: no
direction: receive
rx-size: 1500
[Q quit|D dump|C-z pause]
```

The right window is connected to the target IP 6C:3B:6B:1A:7C:5A and displays the interface statistics for ether4. The statistics show a Tx/Rx Rate of 10.2 Mbps and 1216 bps, and a Tx/Rx Packet Rate of 843 p/s and 2 p/s. A red circle highlights a burst in the traffic graph, corresponding to the test results.

Interface	Loop Protect	Overall Stats	Rx Stats	Tx Stats	Status	Traffic
Ethernet						
Tx/Rx Rate:		10.2 Mbps		1216 bps		
Tx/Rx Packet Rate:		843 p/s		2 p/s		
FP Tx/Rx Rate:		10.1 Mbps		1776 bps		
FP Tx/Rx Packet Rate:		839 p/s		3 p/s		
Tx/Rx Bytes:		3427.9 MiB		828.8 MiB		
Tx/Rx Packets:		2 879 980		12 179 984		
Tx/Rx Drops:		0		0		
Tx/Rx Errors:		0		0		

# Burst with Ceiling

Large Bucket at the child with 10M *max-limit* will still allow  $10 * 10M = 100M$  of traffic.

But the Parent is replenishing the bucket at 20Mbit rate. So it will take ~5seconds to be able to empty the 100M bucket, before the queue settles at the actual Token Rate of 10M.





# HTB

- Very predictable regular traffic can be handled by small buckets. Larger buckets may be required for burstier traffic, unless one of the desired goals is to reduce the burstiness of the flows.



# Credits

- [https://wiki.mikrotik.com/wiki/Manual:HTB-Token Bucket Algorithm](https://wiki.mikrotik.com/wiki/Manual:HTB-Token_Bucket_Algorithm)
- [https://wiki.mikrotik.com/index.php?title=Manual:Queues - Burst](https://wiki.mikrotik.com/index.php?title=Manual:Queues_-_Burst)
- <http://linux-ip.net/articles/Traffic-Control-HOWTO>
- <http://www.docum.org/>



# Grazie!

Thank You!

Grazzi!

¡Gracias!

Time for questions, answers and suggestions

*ag@tnxe.net*

*mc@tnxe.net*

