HTB (Hierarchical Token Bucket)

Introduction

HTB (Hierarchical Token Bucket) is a classful queuing discipline that is useful for rate limiting and burst handling. This article will focus on those HTB aspects exclusively in RouterOS, as we use a modified version to deliver features like Simple Queue and Queue Tree.

Token Bucket algorithm (Red part of the diagram)

The Token Bucket algorithm is based on an analogy to a bucket where tokens, represented in bytes, are added at a specific rate. The bucket itself has a specified capacity.

If the bucket fills to capacity, newly arriving tokens are dropped.

Bucket capacity = bucket-size * max-limit

- **bucket size** (0..10, Default:0.1)

Before allowing any packet to pass through the queue, the queue bucket is inspected to see if it already contains sufficient tokens at that moment.

If yes, the appropriate number of tokens are removed ("cashed in") and the packet is permitted to pass through the queue.

If not, the packets stay at the start of the packet waiting queue until the appropriate amount of tokens is available.

In the case of a multi-level queue structure, tokens used in a child queue are also 'charged' to their parent queues. In other words - child queues 'borrow' tokens from their parent queues.

Packet queue (Blue part of the diagram)

The size of this packet queue, the sequence, how packets are added to this queue, and when packets are discarded is determined by:

- **queue-type** - Queue
- **queue-size** - Queue Size

Token rate selection (Black part of the diagram)

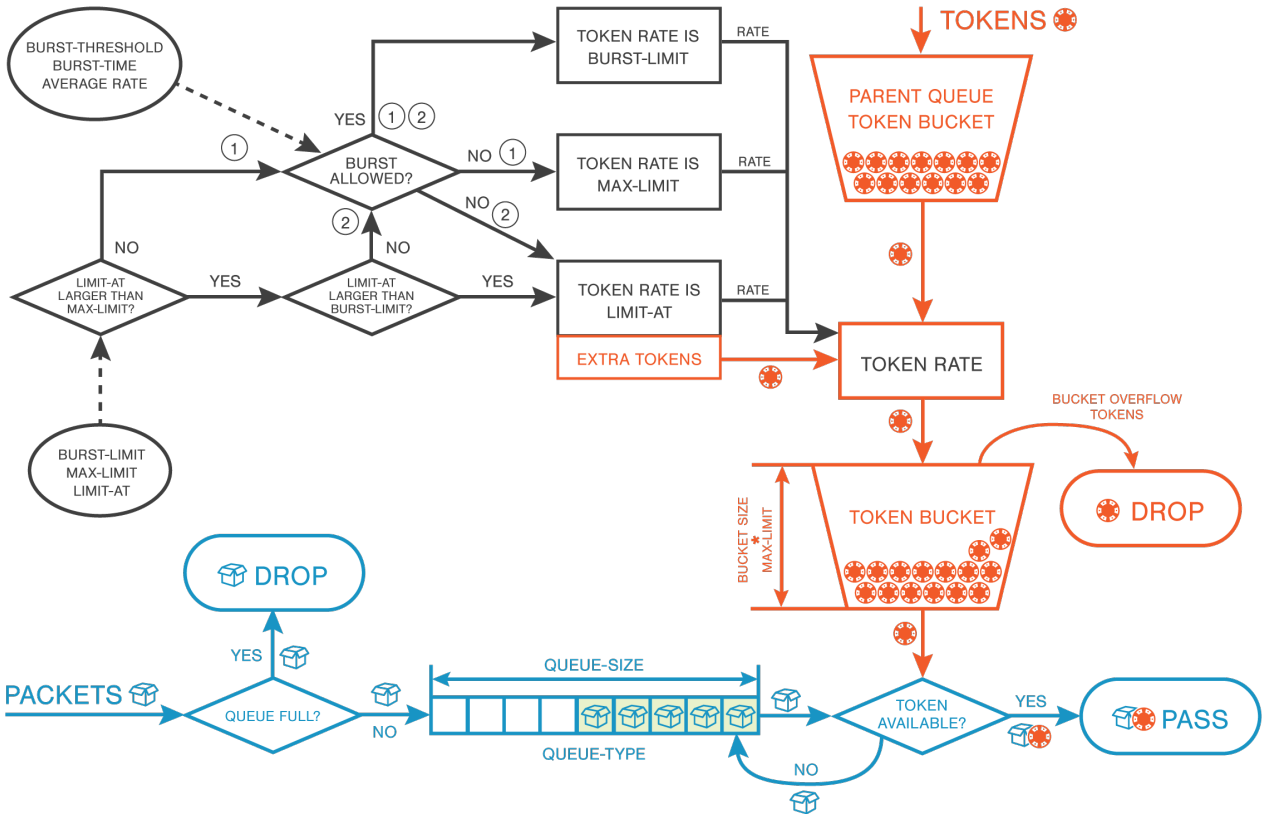
The maximal token rate at any given time is equal to the highest activity of these values:

- **limit-at** (*NUMBER/NUMBER*): guaranteed upload/download data rate to a target
- **max-limit** (*NUMBER/NUMBER*): maximal upload/download data rate that is allowed for a target
- **burst-limit** (*NUMBER/NUMBER*): maximal upload/download data rate that is allowed for a target while the 'burst' is active

burst-limit is active only when 'burst' is in the allowed state - more info here: Queue Burst

In a case where **limit-at** is the highest value, extra tokens need to be issued to compensate for all missing tokens that were not borrowed from its parent queue.

The Diagram



Bucket Size in action

Let's have a simple setup where all traffic from and to one IP address is marked with a packet-mark:

```
/ip firewall mangle
add chain=forward action=mark-connection connection-mark=no-mark src-address=192.168.88.101 new-connection-
mark=pc1_conn
add chain=forward action=mark-connection connection-mark=no-mark dst-address=192.168.88.101 new-connection-
mark=pc1_conn
add chain=forward action=mark-packet connection-mark=pc1_conn new-packet-mark=pc1_traffic
```

Default Queue Bucket

```
/queue tree
add name=download parent=Local packet-mark=PC1-traffic max-limit=10M
add name=upload parent=Public packet-mark=PC1-traffic max-limit=10M
```

In this case bucket-size=0.1, so bucket-capacity= 0.1 x 10M = 1M

If the bucket is full (that is, the client was not using the full capacity of the queue for some time), the next 1Mb of traffic can pass through the queue at an unrestricted speed.

Large Queue Bucket

```
/queue tree
add name=download parent=Local packet-mark=PC1-traffic max-limit=10M bucket-size=10
add name=upload parent=Public packet-mark=PC1-traffic max-limit=10M bucket-size=10
```

Let's try to apply the same logic to a situation when bucket size is at its maximal value:

In this case bucket-size=10, so bucket-capacity= 10 x 10M = 100M

If the bucket is full (that is, the client was not using the full capacity of the queue for some time), the next 100Mb of traffic can pass through the queue at an unrestricted speed.

So you can have:

- 20Mbps transfer speed for 10s
- 60Mbps transfer burst for 2s
- 1Gbps transfer burst for approximately 100ms

You can therefore see that the bucket permits a type of 'burstiness' of the traffic that passes through the queue. The behavior is similar to the normal burst feature but lacks the upper limit of the burst. This setback can be avoided if we utilize bucket size in the queue structure:

Large Child Queue Bucket, Small Parent Queue Bucket

```
/queue tree
add name=download_parent parent=Local max-limit=20M
add name=download parent=download_parent packet-mark=PC1-traffic max-limit=10M bucket-size=10
add name=upload_parent parent=Public max-limit=20M
add name=upload parent=upload_parent packet-mark=PC1-traffic max-limit=10M bucket-size=10
```

In this case:

- parent queue bucket-size=0.1, bucket-capacity= 0.1 x 20M = 2M
- child queue bucket-size=10, bucket-capacity= 10 x 10M = 100M

The parent will run out of tokens much faster than the child queue and as its child queue always borrows tokens from the parent queue the whole system is restricted to token-rate of the parent queue - in this case to max-limit=20M. This rate will be sustained until the child queue runs out of tokens and will be restricted to its token rate of 10Mbps.

In this way, we can have a burst at 20Mbps for up to 10 seconds.

Configuration

We have to follow three basic steps to create HTB:

- **Match and mark traffic** – classify traffic for further use. Consists of one or more matching parameters to select packets for the specific class;
- **Create rules (policy) to mark traffic** – put specific traffic classes into specific queues and define the actions that are taken for each class;
- **Attach a policy for specific interface(-s)** – append policy for all interfaces (global-in, global-out, or global-total), for a specific interface, or for a specific parent queue;

HTB allows to create of a hierarchical queue structure and determines relations between queues, like "parent-child" or "child-child".

As soon as the queue has at least one child it becomes an **inner** queue, all queues without children - are **leaf** queues. **Leaf** queues make actual traffic consumption, **Inner** queues are responsible only for traffic distribution. All **leaf** queues are treated on an equal basis.

In RouterOS, it is necessary to specify the **parent** option to assign a queue as a child to another queue.

Dual Limitation

Each queue in HTB has two rate limits:

- **CIR** (Committed Information Rate) – (**limit-at** in RouterOS) worst case scenario, the flow will get this amount of traffic no matter what (assuming we can actually send so much data);
- **MIR** (Maximal Information Rate) – (**max-limit** in RouterOS) best case scenario, a rate that flow can get up to if their queue's parent has spare bandwidth;

In other words, at first **limit-at** (**CIR**) of all queues will be satisfied, only then child queues will try to borrow the necessary data rate from their parents in order to reach their **max-limit** (**MIR**).

> ⚠ **CIR** will be assigned to the corresponding queue no matter what. (even if max-limit of the parent is exceeded)

That is why, to ensure optimal (as designed) usage of the dual limitation feature, we suggest sticking to these rules:

- The Sum of committed rates of all children must be less or equal to the amount of traffic that is available to parents;

CIR(parent)* ≥ CIR(child1) +...+ CIR(childN)*in case if parent is main parent CIR(parent)=MIR(parent)

- The maximal rate of any child must be less or equal to the maximal rate of the parent

MIR (parent) ≥ MIR(child1) & MIR (parent) ≥ MIR(child2) & ... & MIR (parent) ≥ MIR(childN)

Queue colors in Winbox:

- 0% - 50% available traffic used - green
- 51% - 75% available traffic used - yellow
- 76% - 100% available traffic used - red

Priority

We already know that **limit-at** (**CIR**) to all queues will be given out no matter what.

Priority is responsible for the distribution of remaining parent queues traffic to child queues so that they are able to reach **max-limit**

The queue with higher priority will reach its **max-limit** before the queue with lower priority. 8 is the lowest priority, and 1 is the highest.

Make a note that priority only works:

- for **leaf** queues - priority in the **inner** queue has no meaning.
- if **max-limit** is specified (not 0)

Examples

In this section, we will analyze HTB in action. To do that we will take one HTB structure and will try to cover all the possible situations and features, by changing the amount of incoming traffic that HTB has to recycle. and changing some options.

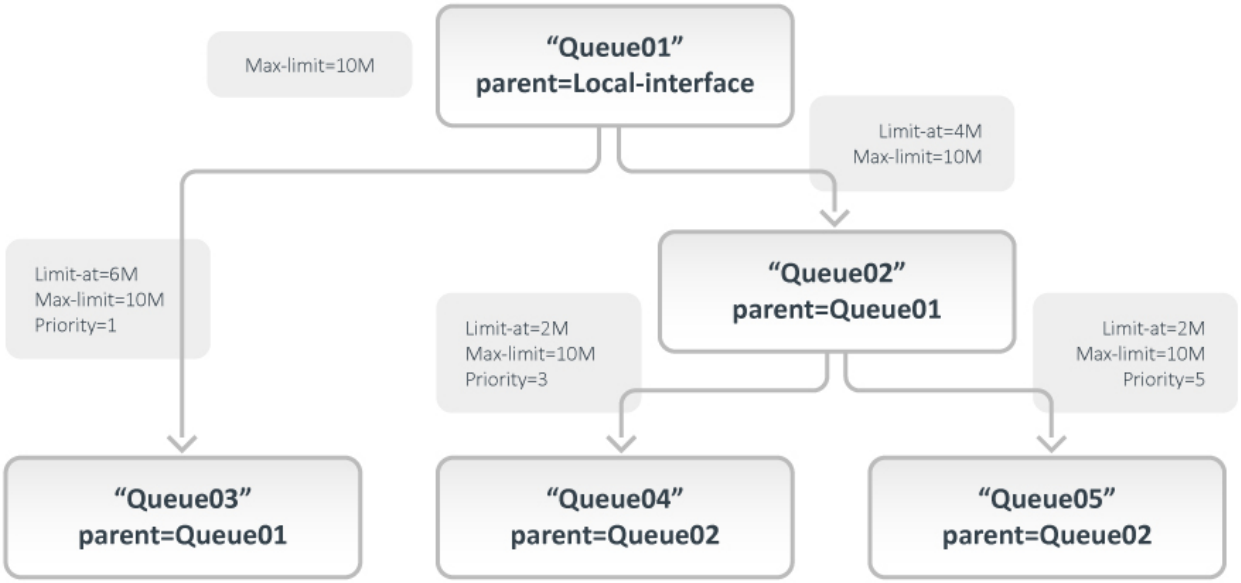
Structure

Our HTB structure will consist of 5 queues:

- **Queue01** inner queue with two children - **Queue02** and **Queue03**
- **Queue02** inner queue with two children - **Queue04** and **Queue05**
- **Queue03** leaf queue
- **Queue04** leaf queue
- **Queue05** leaf queue

Queue03, **Queue04,** and **Queue05** are clients who require 10Mbps all the time Outgoing interface is able to handle 10Mbps of traffic.

Example 1: Usual case

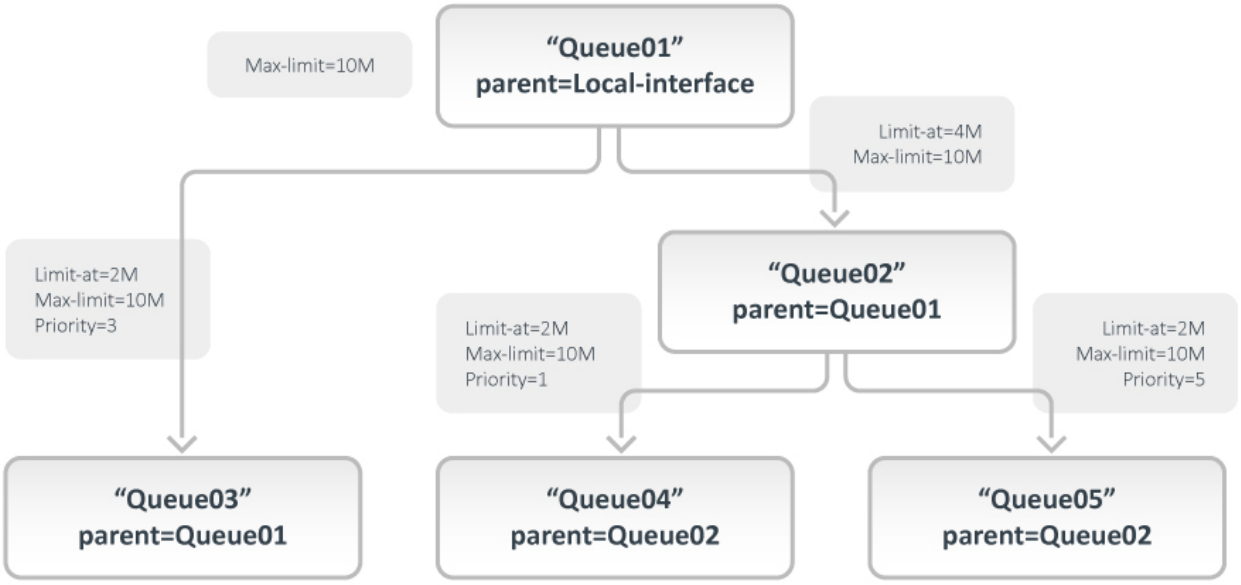


- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=4Mbps max-limit=10Mbps
- **Queue03** limit-at=6Mbps max-limit=10Mbps priority=1
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue05** limit-at=2Mbps max-limit=10Mbps priority=5

Result of Example 1

- **Queue03** will receive 6Mbps
- **Queue04** will receive 2Mbps
- **Queue05** will receive 2Mbps
- **Clarification:** HTB was built in a way, that, by satisfying all **limit-at**s, the main queue no longer has throughput to distribute.

Example 2: Usual case with max-limit

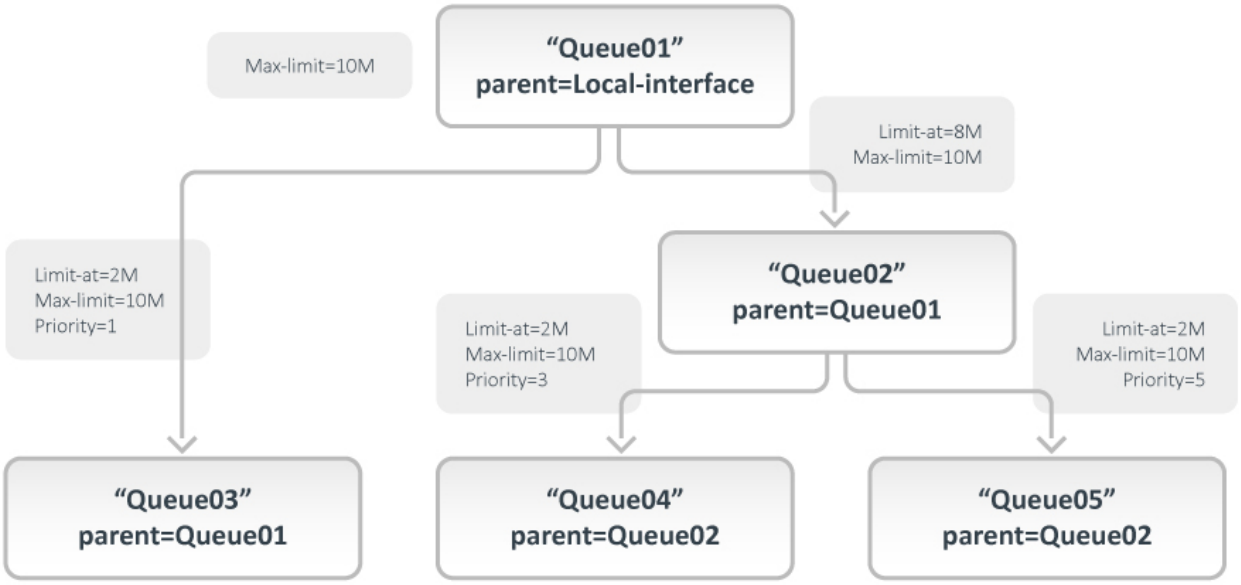


- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=4Mbps max-limit=10Mbps
- **Queue03** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=1
- **Queue05** limit-at=2Mbps max-limit=10Mbps priority=5

Result of Example 2

- **Queue03** will receive 2Mbps
- **Queue04** will receive 6Mbps
- **Queue05** will receive 2Mbps
- **Clarification:** After satisfying all **limit-at**s HTB will give throughput to the queue with the highest priority.

Example 3: Inner queue limit-at

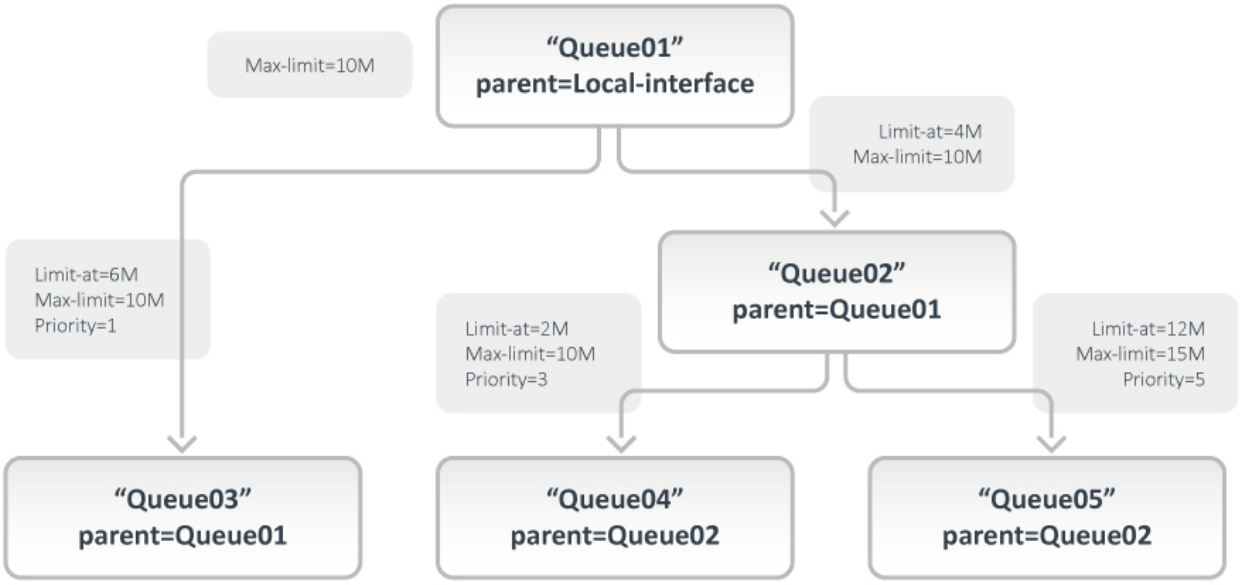


- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=8Mbps max-limit=10Mbps
- **Queue03** limit-at=2Mbps max-limit=10Mbps priority=1
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue05** limit-at=2Mbps max-limit=10Mbps priority=5

Result of Example 3

- **Queue03** will receive 2Mbps
- **Queue04** will receive 6Mbps
- **Queue05** will receive 2Mbps
- **Clarification:** After satisfying all **limit-at**s HTB will give throughput to the queue with the highest priority. But in this case, **inner** queue **Queue02** had **limit-at** specified, by doing so, it reserved 8Mbps of throughput for queues **Queue04** and **Queue05**. Of these two **Queue04** has the highest priority, which is why it gets additional throughput.

Example 4: Leaf queue limit-at



- **Queue01** limit-at=0Mbps max-limit=10Mbps
- **Queue02** limit-at=4Mbps max-limit=10Mbps
- **Queue03** limit-at=6Mbps max-limit=10Mbps priority=1
- **Queue04** limit-at=2Mbps max-limit=10Mbps priority=3
- **Queue05** limit-at=12Mbps max-limit=15Mbps priority=5

Result of Example 4

- **Queue03** will receive ~3Mbps
- **Queue04** will receive ~1Mbps
- **Queue05** will receive ~6Mbps
- **Clarification:** Only by satisfying all **limit-at**s HTB was forced to allocate 20Mbps - 6Mbps to **Queue03**, 2Mbps to **Queue04**, and 12Mbps to **Queue05** , but our output interface is able to handle 10Mbps. As the output interface queue is usually FIFO throughput allocation will keep the ratio 6:2:12 or 3:1:6