

Bandwidth Control

Document revision 2.2 (November 28, 2007, 10:45 GMT)

This document applies to V3.0

Table of Contents

[Table of Contents](#)

[Summary](#)

[Specifications](#)

[Description](#)

[Additional Documents](#)

[Queue Types](#)

[Description](#)

[Property Description](#)

[Interface Default Queues](#)

[Description](#)

[Property Description](#)

[Example](#)

[Simple Queues](#)

[Description](#)

[Property Description](#)

[Queue Trees](#)

[Description](#)

[Property Description](#)

[Example of emulating a 128Kibps/64Kibps Line](#)

[Queue Tree Example With Masquerading](#)

[Equal bandwidth sharing among users](#)

General Information

Summary

Bandwidth Control is a set of mechanisms that control data rate allocation, delay variability, timely delivery, and delivery reliability. The MikroTik RouterOS supports the following queuing disciplines:

- **PFIFO** - Packets First-In First-Out
- **BFIFO** - Bytes First-In First-Out
- **SFQ** - Stochastic Fairness Queuing
- **RED** - Random Early Detect
- **PCQ** - Per Connection Queue
- **HTB** - Hierarchical Token Bucket

Specifications

Packages required: *system*

License required: *level1 (limited to 1 queue), level3*

Home menu level: */queue*

Standards and Technologies: *None*

Hardware usage: *significant*

Description

Quality of Service (QoS) means that the router should prioritize and shape network traffic. QoS is not so much about limiting, it is more about providing quality service to the network users. Some features of MikroTik RouterOS traffic control mechanism are listed below:

- limit data rate for certain IP addresses, subnets, protocols, ports, and other parameters
- limit peer-to-peer traffic
- prioritize some packet flows over others
- use queue bursts for faster web browsing
- apply queues on fixed time intervals
- share available traffic among users equally, or depending on the load of the channel

The queuing is applied on packets leaving the router through a real interface (i.e., the queues are applied on the outgoing interface, regarding the traffic flow), or any of the 3 additional virtual interfaces (global-in, global-out, global-total).

The QoS is performed by means of dropping packets. In case of TCP protocol, the dropped packets will be resent on a slower rate, so there is no need to worry that with shaping we lose some TCP information.

The main terms used to describe the level of QoS for network applications, are:

- **queuing discipline (qdisc)** - an algorithm that holds and maintains a queue of packets. It accumulates the packets and decides the order of the outgoing packets (it means that queuing discipline can reorder packets). Qdisc also decides which packets to drop if there is no space for them.
- **CIR (Committed Information Rate)** - the guaranteed data rate. It means that traffic rate, not exceeding this value should always be delivered
- **MIR (Maximal Information Rate)** - the maximal data rate router will provide
- **Priority** - the order of importance in what traffic will be processed. You can give priority to some traffic in order it to be handled before some other traffic
- **Contention Ratio** - the ratio to which the defined data rate is shared among users (when a certain data rate is allocated to a number of subscribers). It is the number of subscribers that have a single speed limitation, applied to all of them together. For example, the contention ratio of 1:4 means that the allocated data rate may be shared between no more than 4 users

Before sending data over an interface, it is processed with a queuing discipline. There can be only one queuing discipline per interface, which, by default, is set under **/queue interface** for each physical interface (there is no default queuing discipline for virtual interfaces). Once we add a first queue (in **/queue tree** or **/queue simple**) to a physical interface, the interface default queue is replaced by HTB hierarchy with that queue, but the one defined in **/queue interface** for that particular interface, is no more active.

Scheduler and Shaper qdiscs

We can classify queuing disciplines by their influence to packet flow:

- **schedulers** - queuing disciplines only reschedule packets regarding their algorithm and drop packets which 'do not fit in the queue'. Scheduler queuing disciplines are: PFIFO, BFIFO, SFQ, PCQ (both scheduler and shaper), RED
- **shapers** - queuing disciplines that also perform the limitation. Shapers are PCQ (both scheduler and shaper) and HTB

Virtual Interfaces

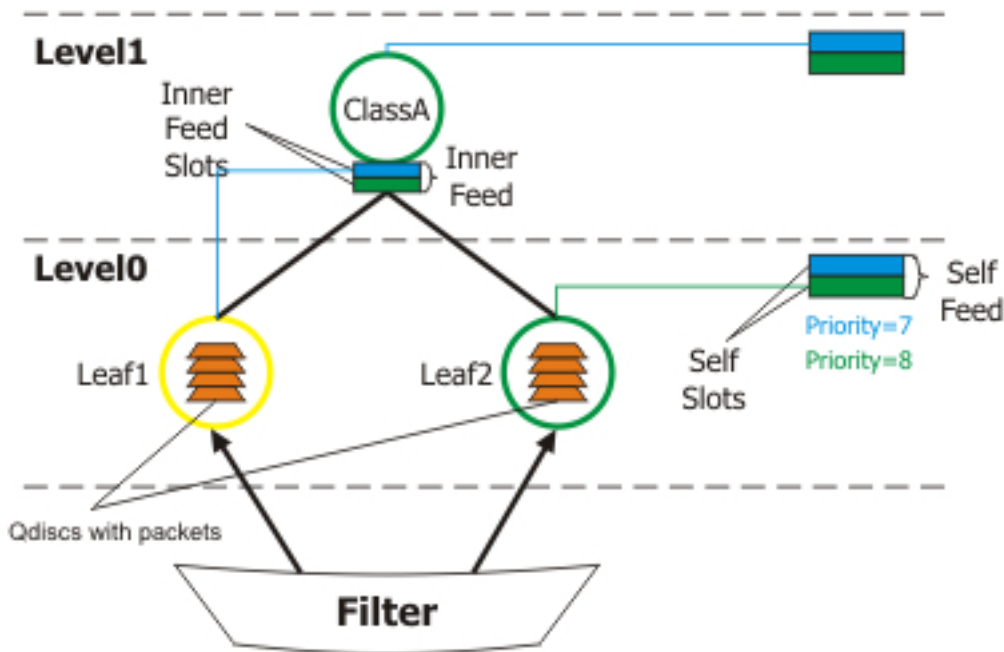
There are 3 virtual interfaces in RouterOS, in addition to real interfaces:

- **global-in** - represents all the input interfaces in general (INGRESS queue). Please note that queues attached to global-in apply to traffic that is received by the router, before the packet filtering. global-in queueing is executed just after mangle and dst-nat
- **global-out** - represents all the output interfaces in general (EGRESS queue). Queues attached to it apply before the ones attached to a specific interface
- **global-total** - represents a virtual interface through which all the data, going through the router, is passing. When attaching a qdisc to global-total, the limitation is done in both directions. For example, if we set a total-max-limit to 256000, we will get upload+download=256kbps (maximum)

Introduction to HTB

HTB (Hierarchical Token Bucket) is a classful queuing discipline that is useful for applying different handling for different kinds of traffic. The queues you add in **/queue simple** and **/queue tree** are attached to the main Hierarchical Token Bucket (HTB). For example, you can set a maximum data rate for a workgroup and then distribute that amount of traffic between the members of that workgroup.

HTB qdisc in detail:



HTB terms:

- **filter** - a procedure that classifies packets. The filters are responsible for classifying packets so that they are put in the corresponding qdiscs. All filters are applied at the HTB root and classify packets directly into the qdiscs, without traversing the HTB tree. If a packet is not classified into any of the qdiscs, it is sent out to the interface directly, traversing the HTB, so no HTB rules are applied to those packets (it would mean effective higher priority than of any packet flow managed by HTB).
- **level** - position of a class in the hierarchy.
- **class** - algorithm for limiting traffic flow to a certain rate. It does not store any packets (this function can only be performed by a queue). A class may contain either one or more subclasses (inner class), or one and only one qdisc (leaf class).
- **inner class** - a class that has one or more child class attached to it. As inner classes do not store any packets, qdiscs can not be attached to them (so their qdisc and filter settings are ignored, although may be still shown in RouterOS configuration), so they only do traffic shaping. Priority setting is ignored as well.
- **leaf class** - a class that has a parent but does not have any child classes. Leaf classes are always located at level 0 of the hierarchy. Each leaf class has one and only one qdisc attached to it, with a certain priority.
- **self feed** - an exit (out of the HTB tree, to the interface) for the packets from all the classes active on its level of the hierarchy. There is one self feed per level, each consisting of 8 self slots that represent priorities.
- **self slot** - an element of a self feed that corresponds to each particular priority. There is one self slot per priority per level. All classes, active at the same level, having the same priority are attached to one self slot that they are using to send packets out through.
- **active class (at a particular level)** - a class that is attached to a self slot at the given level.
- **inner feed** - similar to a self feed object, which consists of inner self slots, present on each inner class. There is one inner feed per inner class.

- **inner feed slot** - similar to self slot. Each inner feed consists of inner slots which represent a priority.

Each class has a parent and may have one or more children. Classes that do not have children, are put at level 0, where queues are maintained, and are called 'leaf classes'.

Each class in the hierarchy can prioritize and shape traffic. There are two main parameters, which refer to shaping and one - to prioritizing:

- **limit-at** - normal data rate that is guaranteed to a class (CIR)
- **max-limit** - maximal data rate that is allowed for a class to reach (MIR)
- **priority** - order in which classes are served at the same level (8 is the lowest priority, 1 is the highest)

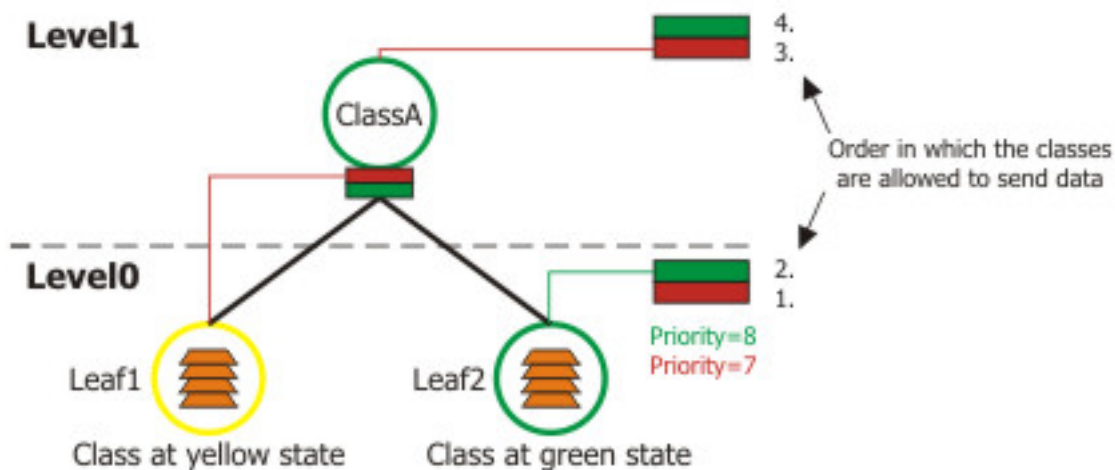
Each HTB class can be in one of 3 states, depending on data rate that it consumes:

- **green** - a class the actual rate of which is equal or less than limit-at. At this state, the class is attached to self slot at the corresponding priority at its level, and is allowed to satisfy its CIR limitation regardless of what limitations its parents have. For example, if we have a leaf class with limit-at=512000 and its parent has max-limit=limit-at=128000, the class will still get its 512kbps! All CIRs of a particular level are satisfied before all MIRs of the same level and any limitations of higher levels.
- **yellow** - a class the actual rate of which is greater than limit-at and equal or less than max-limit (or burst-limit if burst is active). At this state, the class is attached to the inner slot of the corresponding priority of its parent's inner feed, which, in turn, may be attached to either its parent's inner slot of the same priority (in case the parent is also yellow), or to its own level self slot of the same priority (in case the parent is green). Upon the transition to this state, the class 'disconnects' from self feed of its level, and 'connects' to its parent's inner feed.
- **red** - a class the actual rate of which exceeds max-limit (or burst-limit if burst is active). This class cannot borrow rate from its parent class.

Note: as CIRs are always satisfied before MIRs or other limitations of higher levels are consulted, you should always ensure that the **limit-at** property of any inner class is equal or greater than the sum of all **limit-at** parameter of the children of that inner class.

Priorities

When there is a possibility to send out a packet, HTB queries all its self slots in order of priority, starting with highest priority on the lowest level, till lowest priority on highest level. Each leaf class (packets are stored and enqueued only within qdiscs attached to each leaf class) is ultimately connected to a certain self slot, either directly or through a chain of parent classes:



As you can see from the picture, leaf-classes that are in the green state will always have a higher effective priority than those that are yellow (and, thus, borrowing their rate from parent classes), because their priority is at a lower level (level 0). In this picture, **Leaf2** will be served only after **Leaf1**, although it has a higher priority (priority 7) than **Leaf1** (priority 8).

In case of equal priorities and equal states, HTB serves these classes, using round robin algorithm.

HTB Examples

Here are some examples on how the HTB works.

Imagine the following scenario - we have 3 different kinds of traffic, marked in **/ip firewall mangle** (packet_mark1, packet_mark2 and packet_mark3), and now have built a HTB hierarchy:

```
[admin@MikroTik] queue tree> add name=ClassA parent=Local max-limit=2048000
[admin@MikroTik] queue tree> add name=ClassB parent=ClassA max-limit=1024000
[admin@MikroTik] queue tree> add name=Leaf1 parent=ClassA max-limit=2048000 \
\... limit-at=1024000 packet-mark=packet_mark1 priority=8
[admin@MikroTik] queue tree> add name=Leaf2 parent=ClassB max-limit=1024000 \
\... limit-at=256000 packet-mark=packet_mark2 priority=7
[admin@MikroTik] queue tree> add name=Leaf3 parent=ClassB max-limit=1024000 \
\... limit-at=768000 packet-mark=packet_mark3 priority=8
[admin@MikroTik] queue tree> print
Flags: X - disabled, I - invalid
0  name="ClassA" parent=Local packet-mark="" limit-at=0 queue=default
   priority=8 max-limit=2048000 burst-limit=0 burst-threshold=0
   burst-time=0s

1  name="ClassB" parent=ClassA packet-mark="" limit-at=0 queue=default
   priority=8 max-limit=1024000 burst-limit=0 burst-threshold=0
   burst-time=0s

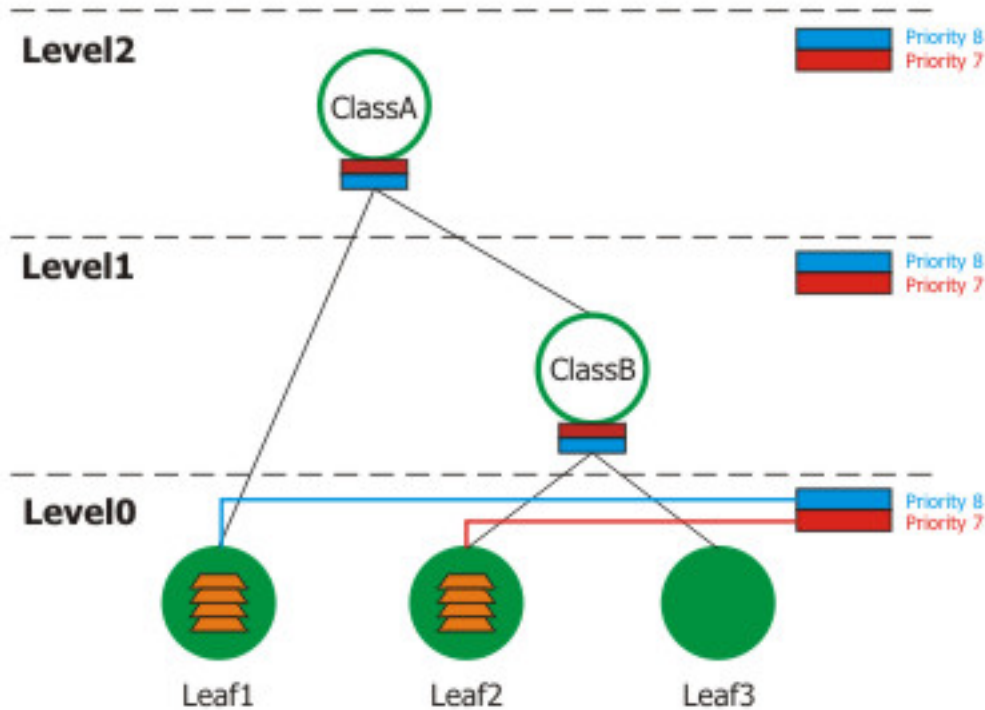
2  name="Leaf1" parent=ClassA packet-mark=packet_mark1 limit-at=1024000
   queue=default priority=8 max-limit=2048000 burst-limit=0
   burst-threshold=0 burst-time=0s

3  name="Leaf2" parent=ClassB packet-mark=packet_mark2 limit-at=256000
   queue=default priority=7 max-limit=1024000 burst-limit=0
   burst-threshold=0 burst-time=0s

4  name="Leaf3" parent=ClassB packet-mark=packet_mark3 limit-at=768000
   queue=default priority=8 max-limit=1024000 burst-limit=0
   burst-threshold=0 burst-time=0s
[admin@MikroTik] queue tree>
```

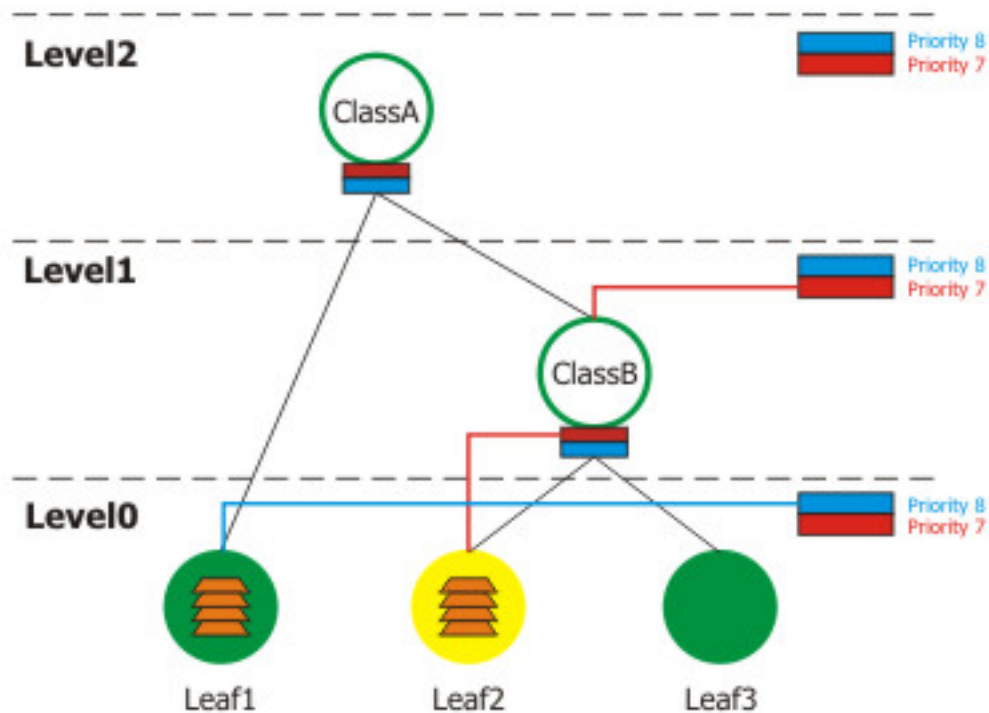
Now let us describe some scenarios, using this HTB hierarchy.

1. Imagine a situation when packets have arrived at Leaf1 and Leaf2. Because of this, Leaf1 attaches itself to this level's (Level 0) self slot with priority=8 and Leaf2 attaches to self slot with priority=7. Leaf3 has nothing to send, so it does nothing.



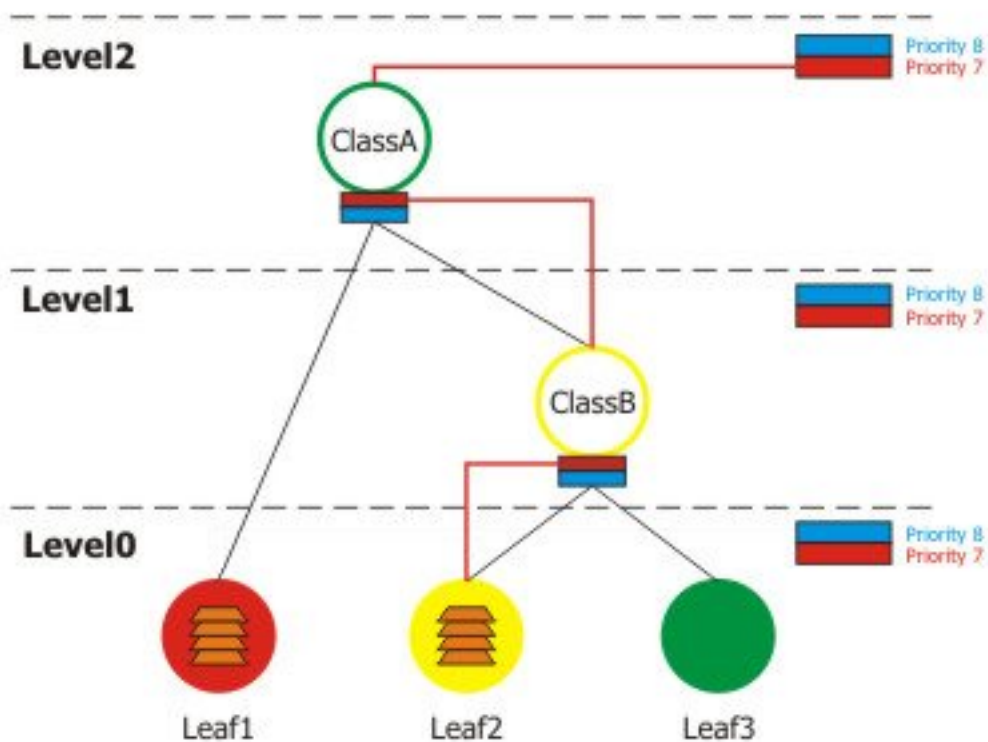
This is a simple situation: there are two active classes (Leaf1 and Leaf2) at Level 0, and as they both are in green state, they are processed in order of their priorities - at first, we serve Leaf2, then Leaf1.

2. Now assume that Leaf2 has to send more than 256kbps, so it needs to go over it's green limit. With the state change, it attaches itself to its parent's (ClassB) inner feed, which recursively attaches itself to Level1 self slot at priority 7. Leaf1 remains in green state - it has packets to send, but their rate is lower than 1Mbps. Leaf3 still has nothing to send.



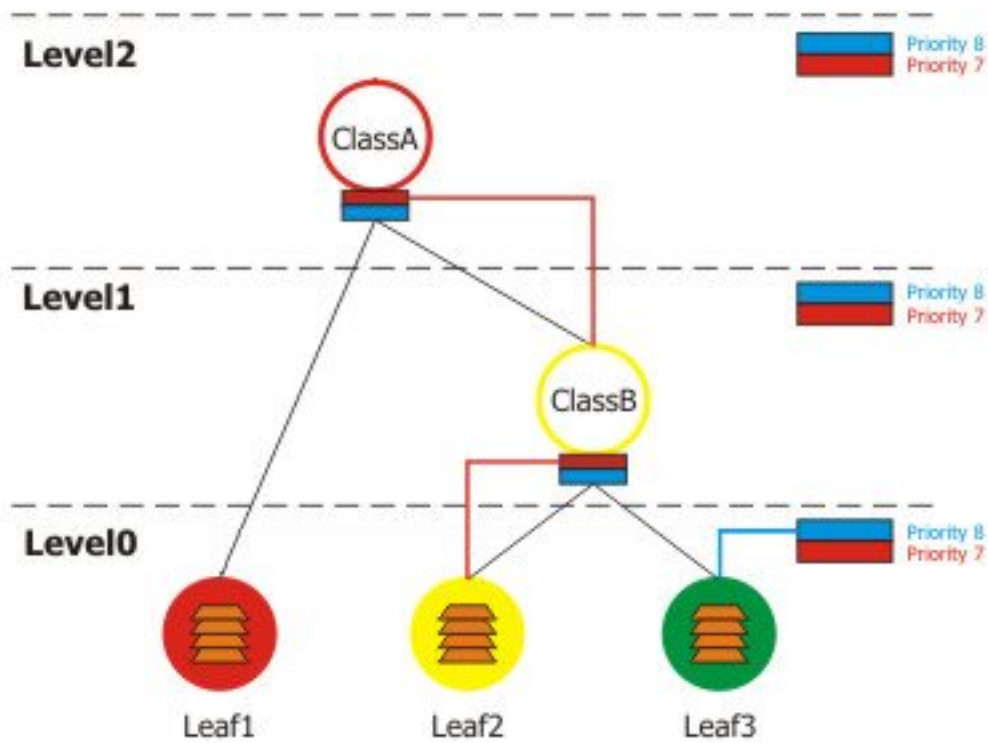
It is very important to understand that Leaf1 now has higher effective priority than Leaf2 (when it is in green state), although we have configured it for a lower priority (8) than Leaf2. It is because Leaf2 has disconnected itself from self feed at Level 0 and is now borrowing rate from its parent (ClassB), which, in turn, has attached to a self feed at Level 1. Thus, the priority of Leaf2 has jumped to Level1. Remember that lowest level is served first, than the next level, and so on, satisfying the attached classes in order of their priority.

3. Consider that Leaf1 has reached its max-limit and changed its state to red, and Leaf2 now uses more than 1Mbps (and less than 2Mbps), so its parent ClassB has to borrow from ClassA and becomes yellow. Leaf3 still has no packets to send.



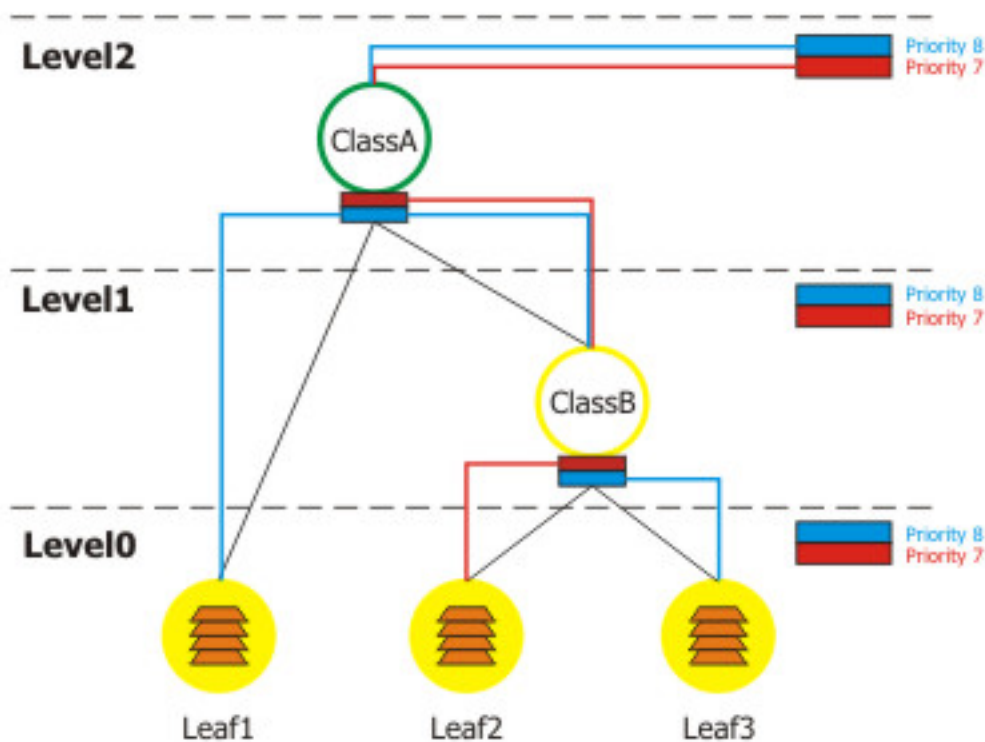
This scenario shows that Leaf1 has reached its max-limit and cannot even borrow from its parent (ClassA), so it is detached from all self slots and inner slots. Leaf2 has recursively reached Level 2, as it borrows from ClassB which, in turn, borrows from ClassA, as it does not have enough rate available. As Leaf3 has no packets to send, the only class that sends is Leaf2.

4. Assume that ClassA reaches its max-limit (2Mbps), so neither ClassB, nor Leaf2 can send as they only rely on borrowing rate, which is impossible as ClassA cannot send. But now, Leaf3 has some packets to send:



In this situation Leaf2 is in yellow state, but it cannot borrow (as Class B cannot borrow from Class A) and Leaf3 is the only class that can send. Note that even though no other classes, including its parents is able to send, Leaf3 can send perfectly well while it is attached to the Level 0 self feed.

5. Finally, let's see what happens, if Leaf1, Leaf2, Leaf3 and ClassB are in the yellow state, and ClassA is green.

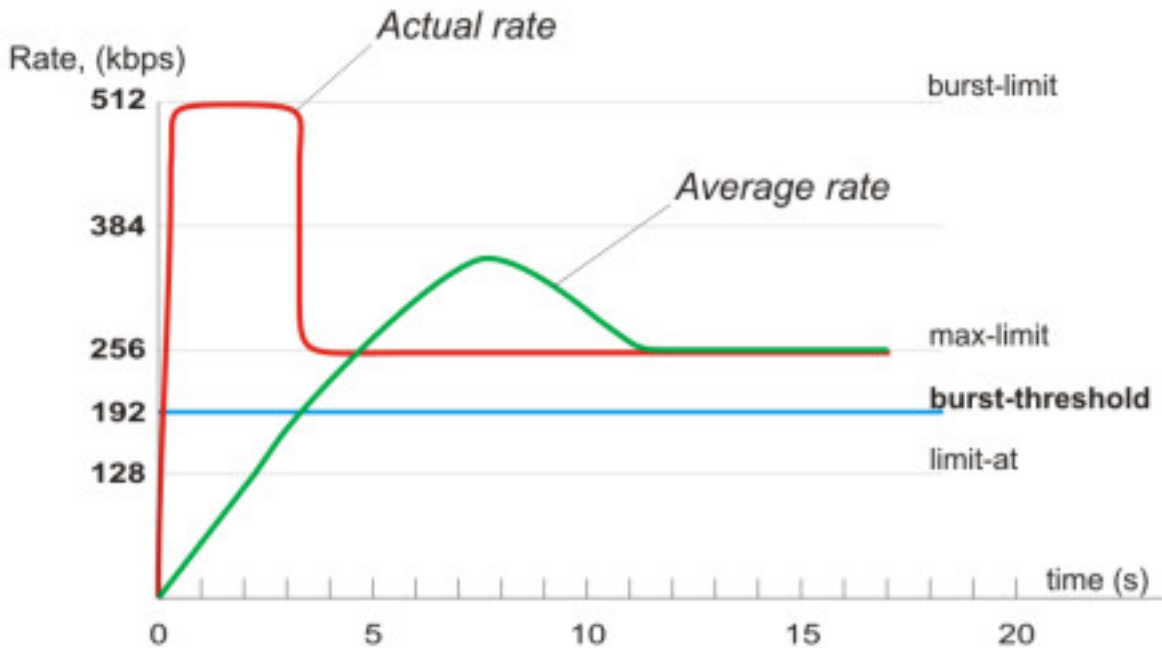


Leaf1 borrows from ClassA, Leaf2 and Leaf3 - from ClassB, and ClassB, in turn, borrows from ClassA. Now all the priorities have 'moved' to Level 2. So Leaf2 is on the highest priority and is served first. As Leaf1 and Leaf3 are of the same priority (8) on the same level (2), they are served using round robin algorithm.

Bursts

Bursts are used to allow higher data rates for a short period of time. Every 1/16 part of the **burst-time**, the router calculates the average data rate of each class over the last **burst-time** seconds. If this average data rate is less than **burst-threshold**, burst is enabled and the effective rate limit (transition to the red state) is set to **burst-limit** bps, otherwise the effective maximal limit falls to **max-limit**.

Let us consider the following setup: **max-limit**=256000, **burst-time**=8, **burst-threshold**=192000 and **burst-limit**=512000. When a user is starting to download a file via HTTP, we can observe such situation:



At the beginning the average data rate over the past 8 seconds is 0bps because no traffic has passed through this rule before it has been created. Since this average data rate is less than **burst-threshold** (192kbps), burst is allowed. After the first second, the average data rate is $(0+0+0+0+0+0+0+512)/8=64\text{kbps}$, which is less than **burst-threshold**. After the second second, average data rate is $(0+0+0+0+0+0+512+512)/8=128\text{kbps}$. After the third second comes the breakpoint when the average data rate becomes larger than **burst-threshold**. At this moment burst is disabled and the effective data rate limitation falls down to **max-limit** (256kbps).

Note how the **burst-time** was used. The actual duration of burst does not depend of **burst-time** alone! It also depends on the **burst-threshold/burst-limit** ratio and the actual data rate passing through the bursty class. In this example the burst ratio was $192000/512000=3/8$, the time was 8, and the queue has been trying to utilize all available rate the class was providing, so the burst was 3 seconds long.

Now you can easily see why the **burst-threshold** should be between **limit-at** and **max-limit** for normal operation. If you specify **burst-threshold** higher than **max-limit**, then the average rate will tend to **burst-threshold**, but the effective maximal limit will jump between **max-limit** and **burst-limit** constantly (depending on the actual traffic rate, it may happen even on each evaluation point (1/16th of **burst-time**)).

HTB in RouterOS

In addition to interface queues (one queue or HTB tree per interface), 3 virtual 4 HTB trees maintained by RouterOS:

- global-in
- global-total
- global-out

When adding a simple queue, it creates 3 HTB classes (in global-in, global-total and global-out), but it does not add any classes in interface queue. Queue tree is more flexible - you can add it to any of these HTB's.

When packet travels through the router, it passes 4 HTB trees - global-in, global-total, global-out and output interface queue. If it is directed to the router, it passes global-in and global-total HTB queues. If packets are sent from the router, they go through global-total, global-out and output interface queues

Additional Documents

- <http://linux-ip.net/articles/Traffic-Control-HOWTO/overview.html>
- <http://luxik.cdi.cz/~devik/qos/htb/>
- <http://www.docum.org/docum.org/docs/>

Queue Types

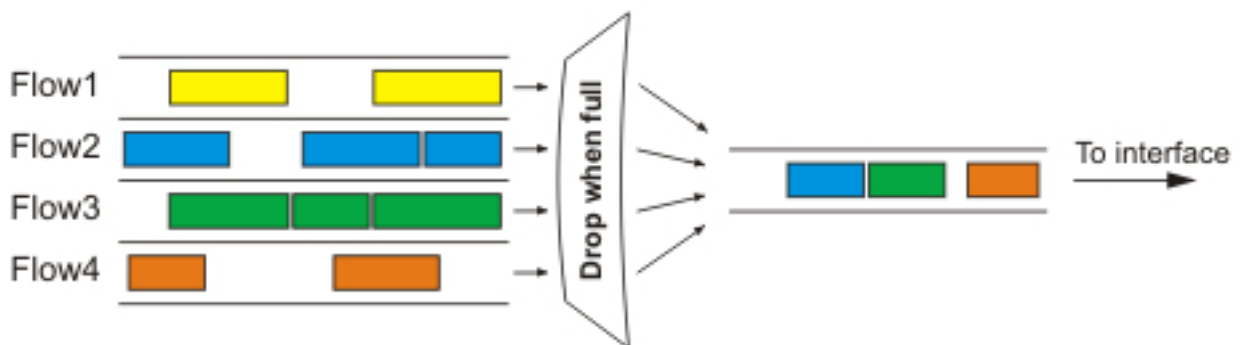
Home menu level: */queue type*

Description

You can create your custom queue types in this submenu. Afterwards, you will be able to use them in **/queue tree**, **/queue simple** or **/queue interface**. Note that these queueing disciplines can not limit data rate at all (except for PCQ) - they only reorganize (schedule) packets and drop excess ones (if the queue is getting too long and the managing class can not send the packets quickly enough), so you won't find any rate limitation parameters here (except for PCQ) - only storage limits. Note also that the scheduling is only taking place when the packets are being enqueued in the qdisc, and this only happens when the packets are coming in at the rate faster than the managing class can provide (so this is only a buffer). There are 5 kinds of qdiscs that can be used for storing packets:

PFIFO and BFIFO

These queueing disciplines are based on the FIFO algorithm (First-In First-Out). The difference between PFIFO and BFIFO is that one is measured in packets and the other one in bytes. There is only one parameter called **pfifo-limit** (or **bfifo-limit** in case of BFIFO) which defines how much data a FIFO queue can hold. Every packet that cannot be enqueued (if the queue is full), is dropped. Large queue sizes can increase latency, but utilize channel better.

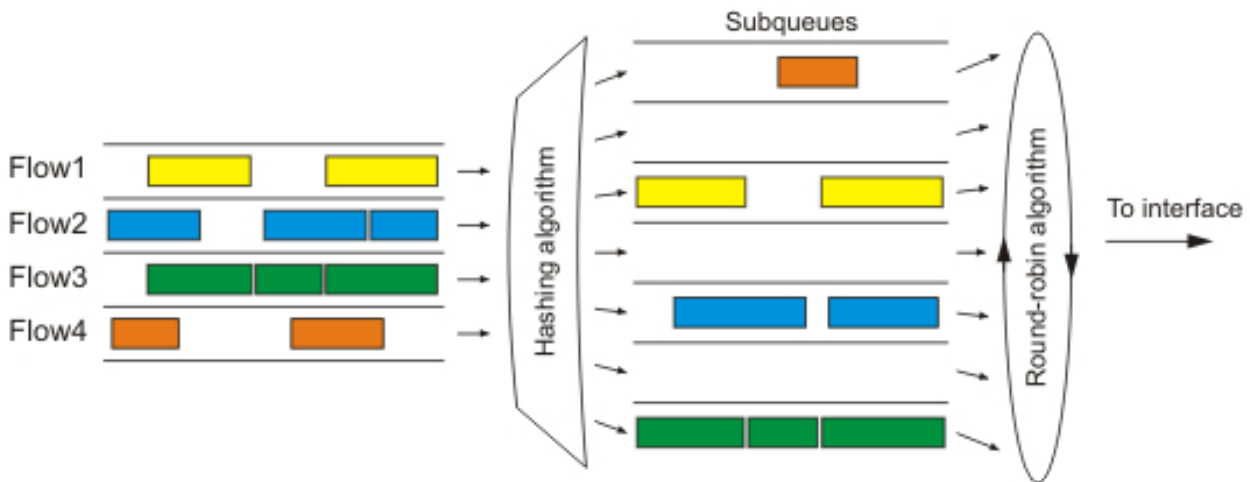


Use FIFO queuing disciplines if you have a noncongested link.

SFQ

Stochastic Fairness Queuing (SFQ) equalizes traffic flows (TCP sessions or UDP streams) when the link is completely full.

The fairness of SFQ is ensured by hashing and round-robin algorithms. Hashing algorithm divides the session traffic over a limited number of subqueues. A traffic flow may be uniquely identified by a tuple (src-address, dst-address, src-port and dst-port), so these parameters are used by SFQ hashing algorithm to classify packets into subqueues.



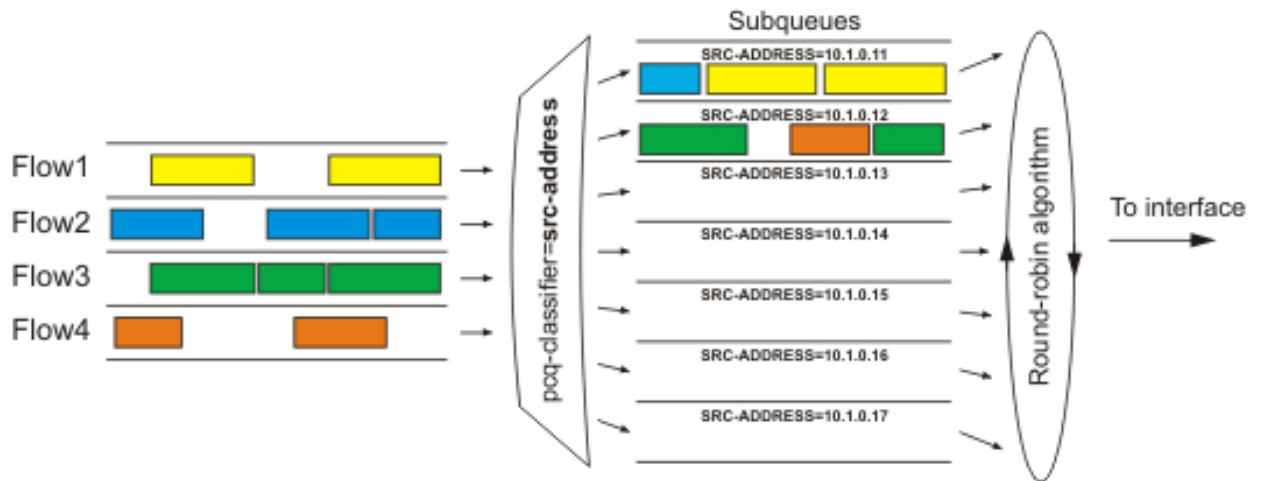
The whole SFQ queue can contain 128 packets and there are 1024 subqueues available for these packets. Each packet stored in a FIFO-like 128 packet buffer, belongs to a certain subqueue, determined by the hash function (a simple function of the tuple values with 10-bit output is used, hence the amount of subqueues is 1024). Stochastic nature of the queueing discipline is observed in that packets from an unpredictable number of flows may actually be classified in the same subqueue. After **sfq-perturb** seconds the hashing algorithm changes and divides the session traffic to other subqueues, so that no separate data flows will be associated with the same subqueue for a long time. The round-robin algorithm dequeues **pcq-allot** bytes from each subqueue in a turn.

Use SFQ for congested links to ensure that connections do not starve. SFQ is especially beneficial on wireless links.

PCQ

To solve some SFQ imperfectness, Per Connection Queuing (PCQ) was created. It is the only classless queueing type in RouterOS that can do rate limitation. It is an improved version of SFQ without its stochastic nature. PCQ also creates subqueues, based on the **pcq-classifier** parameter. Each subqueue has a data rate limit of **pcq-rate** and size of **pcq-limit** packets. The total size of a PCQ queue cannot be greater than **pcq-total-limit** packets.

The following example demonstrates the usage of PCQ with packets, classified by their source address.



If you classify the packets by **src-address** then all packets with different source IP addresses will be grouped into different subqueues. Now you can do the limitation or equalization for each subqueue with the **pcq-rate** parameter. Perhaps, the most significant part is to decide to which interface should we attach this queue. If we will attach it to the Local interface, all traffic from the Public interface will be grouped by src-address (probably it's not what we want), but if we attach it to the Public interface, all traffic from our clients will be grouped by src-address - so we can easily limit or equalize upload for clients. Same can be done for downloads, but in that case **dst-address** classifier will be used, and PCQ put on the locan interface.

To equalize rate among subqueues, classified by the **pcq-classifier**, set the **pcq-rate** to **0**! PCQ can be used to dynamically equalize or shape traffic for multiple users, using little administration. In fact, PCQ always equalizes the subqueues, so the **pcq-rate** is just a cap for equalization - a subqueue may get smaller rate, but will never get higher rate.

RED

Random Early Detection (also known as Random Early Drop, as this is how it actually works) is a queuing mechanism which tries to avoid network congestion by controlling the average queue size. When the average queue size reaches **red-min-threshold**, RED starts to drop packets randomly with linearly increasing probability as the average queue size grows up until the average queue size reaches the **red-max-threshold**. The effective queue size at any moment could be higher than the **red-max-threshold** as the probability does not grow very fast, so it is possible to specify a hard limit for the queue size. When the average queue size reaches **red-max-threshold** or becomes larger, all further packats are dropped until the average queue size does not drop below this value (at which point probalistic calculations will be activated again).

The average queue size **avg** is $(1-W)*avg+W*q$, where

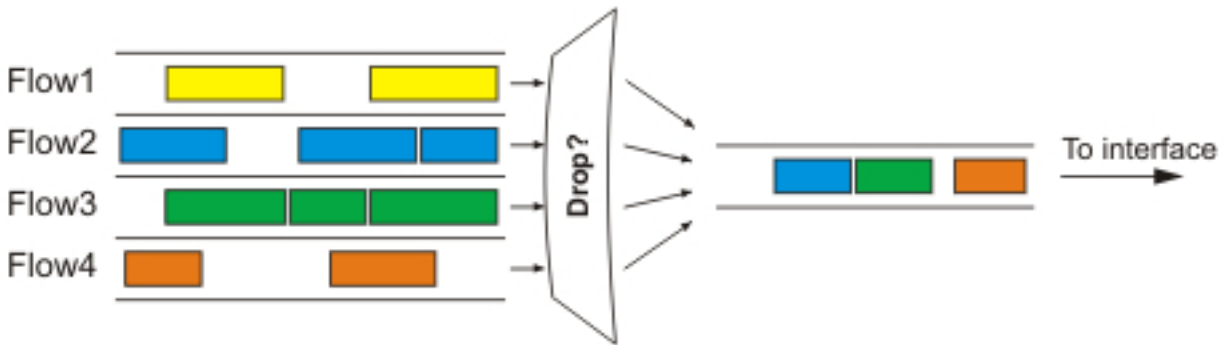
- **q** - current queue length
- **W** - queue weight defined as $burst+1-min=(1-(1-W)^{burst})/W$. Note that $\log(W)$ value is rounded to integer (so W can be 1, 0.1, 0.01, etc.). It is determined experimantally that in many generic cases, W is near to $min/10*burst$

The **pb** probability value is increasing linearly from 0% to 2% as the average queue size grows from **red-min-threshold** to **red-max-threshold**: $pb=0.02*(avg-min)/(max-min)$.

The packet dropping probability **pb** is increasing with **pb** and with enqueued packet count since the last packet was dropped: $pa=pb/(1-count*pb)$.

It is defined experimentally that a good **red-burst** value is $(min+2*max)/3$. And a good **red-max-threshold** is twice **red-min-threshold**.

Note that in the formulas above, **min** means **red-min-threshold**, **max** means **red-max-threshold** and **burst** means **red-burst**.



Used on congested links with high data rates, as it is fast and TCP-friendly.

Property Description

bfifo-limit (*integer*; default: **15000**) - maximum number of bytes that the BFIFO queue can hold

kind (*bfifo | pcq | pfifo | red | sfq*) - which queuing discipline to use

- **bfifo** - Bytes First-In, First-Out
- **pcq** - Per Connection Queue
- **pfifo** - Packets First-In, First-Out
- **red** - Random Early Detection
- **sfq** - Stochastic Fairness Queuing

name (*name*) - reference name of the queue type

pcq-classifier (*dst-address | dst-port | src-address | src-port*; default: `''`) - list classifiers for grouping packets into PCQ subqueues. Several classifiers can be used at once, e.g., src-address,src-port will group all packets with different source address and source-ports into separate subqueues

pcq-limit (*integer*; default: **50**) - number of packets that a single PCQ sub-queue can hold

pcq-rate (*integer*; default: **0**) - maximal data rate allowed for each PCQ sub-queue. This is a rate cap, as the subqueues will be equalized anyway

- **0** - no limitation set (only equalize rates between subqueues)

pcq-total-limit (*integer*; default: **2000**) - number of packets that the whole PCQ queue can hold

pfifo-limit (*integer*) - maximum number of packets that the PFIFO queue can hold

red-avg-packet (*integer*; default: **1000**) - average packet size, used for tuning average queue recalculation time

red-burst (*integer*) - a measure of how fast the average queue size will be influenced by the real queue size, given in bytes. Larger values will smooth the changes, so longer bursts will be allowed

red-limit (*integer*) - hard limit on queue size in bytes. If the real queue size (not average) exceeds this value then all further packets will be discarded until the queue size drops below. This should be higher than red-max-threshold+red-burst

red-max-threshold (*integer*) - upper limit for average queue size, in bytes. When the size reaches this value, all further packets shall be dropped

red-min-threshold (*integer*) - lower limit for average queue size, in bytes. When the size reaches this value, RED starts to drop packets randomly with a calculated probability

sfq-allot (*integer*; default: **1514**) - amount of bytes that a subqueue is allowed to send before the next subqueue gets a turn (amount of bytes which can be sent from a subqueue in a single round-robin turn), should be at least 1514 for links with 1500 byte MTU

sfq-perturb (*integer*; default: **5**) - how often to shake (perturb) SFQ's hashing algorithm, in seconds

Interface Default Queues

Home menu level: */queue interface*

Description

In order to send packets over an interface, they have to be enqueued in a queue even if you do not want to limit traffic at all. Here you can specify the queue type which will be used for transmitting data.

Note that once you configure tree queues for a listed interface, the interface default queue is no longer active for that particular interface, so you need to make sure all packets that goes out through this interface are filtered into some qdiscs inside the HTB tree. Otherwise the packets that are not filtered, are sent out directly (at effective higher priority than any of the packets in the HTB tree), and unbuffered, which ultimately lead to suboptimal performance.

Property Description

interface (*read-only: name*) - name of the interface

queue (*name*; default: **default**) - queue type which will be used for the interface

Example

Set the wireless interface to use **wireless-default** queue:

```
[admin@MikroTik] queue interface> set 0 queue=wireless-default
[admin@MikroTik] queue interface> print
# INTERFACE QUEUE
0 wlan1 wireless-default
[admin@MikroTik] queue interface>
```

Simple Queues

Description

The simplest way to limit data rate for specific IP addresses and/or subnets, is to use simple queues.

You can also use simple queues to build advanced QoS applications. They have useful integrated features:

- Peer-to-peer traffic queuing
- Applying queue rules on chosen time intervals
- Priorities
- Using multiple packet marks from */ip firewall mangle*
- Shaping of bidirectional traffic (one limit for the total of upload + download)

Property Description

burst-limit (*integerinteger*) - maximum data rate which can be reached while the burst is active, in form of in/out (target upload/download)

burst-threshold (*integerinteger*) - average data rate limit, until which the burst is allowed. If the average data rate over the last burst-time seconds is less than burst-threshold, the actual data rate may reach burst-limit. Otherwise the hard limit is reset to max-limit. Set in form of in/out (target upload/download)

burst-time (*integerinteger*) - period of time, in seconds, over which the average data rate is calculated, in form of in/out (target upload/download)

direction (*none | both | upload | download*) - traffic flow directions from the targets' point of view, affected by this queue

- **none** - the queue is effectively inactive
- **both** - the queue limits both target upload and target download
- **upload** - the queue limits only target upload, leaving the download rates unlimited
- **download** - the queue limits only target download, leaving the upload rates unlimited

dst-address (*IP addressnetmask*) - destination address to match

dst-netmask (*netmask*) - netmask for dst-address

interface (*text*) - interface, this queue applies to (i.e., the interface the target is connected to)

limit-at (*integerinteger*) - CIR, in form of in/out (target upload/download)

max-limit (*integerinteger*) - MIR (in case burst is not active), in form of in/out (target upload/download)

name (*text*) - descriptive name of the queue

p2p (*all-p2p | bit-torrent | blubster | direct-connect | edonkey | fasttrack | gnutella | soulseek | winmx*) - which type of P2P traffic to match

- **all-p2p** - match all P2P traffic

packet-marks (*multiple choice: name; default: ""*) - list of packet marks (set by */ip firewall mangle*) to match. Multiple packet marks are separated by commas (",")

parent (*name*) - name of the parent queue in the hierarchy. Can only be another simple queue

priority (*integer: 1..8*) - priority of the queue. 1 is the highest, 8 - the lowest

queue (*namename; default: default/default*) - name of the queue from */queue* type, in form of in/out

target-addresses (*multiple choice: IP addressnetmask*) - limitation target IP addresses (source addresses). Multiple addresses are separated by commas

time (*timetimesat | fri | thu | wed | tue | mon | sun; default: ""*) - limit queue effect to a specified

time period

total-burst-limit (*integer*) - burst limit for global-total (cumulative rate, upload + download) queue

total-burst-threshold (*integer*) - burst threshold for global-total (cumulative rate, upload + download) queue

total-burst-time (*time*) - burst time for global-total queue

total-limit-at (*integer*) - limit-at for global-total (cumulative rate, upload + download) queue

total-max-limit (*integer*) - max-limit for global-total (cumulative rate, upload + download) queue

total-queue (*name*) - queuing discipline to use for global-total queue

Queue Trees

Home menu level: */queue tree*

Description

The queue trees should be used when you want to use sophisticated data rate allocation based on protocols, ports, groups of IP addresses, etc. At first you have to mark packet flows with a mark under **/ip firewall mangle** and then use this mark as an identifier for packet flows in queue trees.

Property Description

burst-limit (*integer*) - maximum data rate which can be reached while the burst is active

burst-threshold (*integer*) - average data rate limit, until which the burst is allowed. If the average data rate over the last burst-time seconds is less than burst-threshold, the actual data rate may reach burst-limit. Otherwise the hard limit is reset to max-limit

burst-time (*time*) - period of time, in seconds, over which the average data rate is calculated

limit-at (*integer*) - CIR

max-limit (*integer*) - MIR (in case burst is not active)

name (*text*) - descriptive name for the queue

packet-mark (*text*) - packet flow mark (set by /ip firewall mangle) to match. This creates a filter that puts the packets with the given mark into this queue

parent (*text*) - name of the parent queue. The top-level parents are the available interfaces (actually, main HTB). Lower level parents can be other tree queues

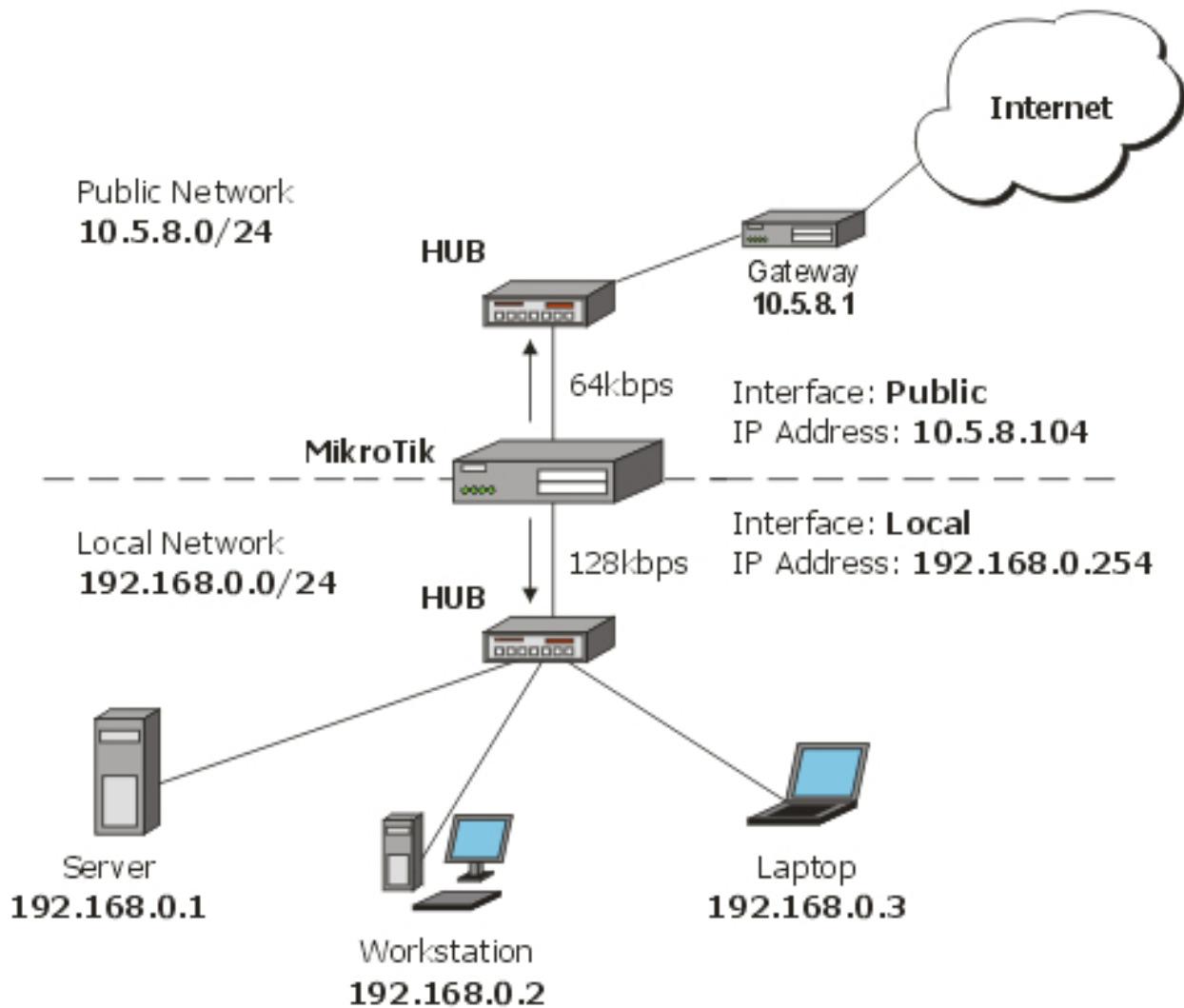
priority (*integer: 1..8*) - priority of the queue. 1 is the highest, 8 - the lowest

queue (*text*) - name of the queue type. Types are defined under /queue type

Application Examples

Example of emulating a 128Kibps/64Kibps Line

Assume, we want to emulate a 128Kibps download and 64Kibps upload line, connecting IP network **192.168.0.0/24**. The network is served through the Local interface of customer's router. The basic network setup is in the following diagram:



To solve this situation, we will use simple queues.

IP addresses on MikroTik router:

```
[admin@MikroTik] ip address> print
Flags: X - disabled, I - invalid, D - dynamic
# ADDRESS NETWORK BROADCAST INTERFACE
0 192.168.0.254/24 192.168.0.0 192.168.0.255 Local
1 10.5.8.104/24 10.5.8.0 10.5.8.255 Public
[admin@MikroTik] ip address>
```

And routes:

```
[admin@MikroTik] ip route> print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
# DST-ADDRESS PREF-SRC G GATEWAY DIS INTE...
0 A S 0.0.0.0/0 r 10.5.8.1 1 Public
1 ADC 10.5.8.0/24 10.5.8.104 0 Public
2 ADC 192.168.0.0/24 192.168.0.254 0 Local
[admin@MikroTik] ip route>
```

Add a simple queue rule, which will limit the download traffic to 128Kib/s and upload to 64Kib/s for clients on the network **192.168.0.0/24**, served by the interface **Local**:

```
[admin@MikroTik] queue simple> add name=Limit-Local interface=Local \
...\ target-address=192.168.0.0/24 max-limit=65536/131072
[admin@MikroTik] queue simple> print
Flags: X - disabled, I - invalid, D - dynamic
0   name="Limit-Local" target-addresses=192.168.0.0/24 dst-address=0.0.0.0/0
    interface=Local parent=none priority=8 queue=default/default
    limit-at=0/0 max-limit=65536/131072 total-queue=default
[admin@MikroTik] queue simple>
```

The **max-limit** parameter cuts down the maximum available bandwidth. From the clients' point of view, the value **65536/131072** means that they will get maximum of 131072bps for download and 65536bps for upload. The **target-addresses** parameter defines the target network (or networks, separated by a comma) to which the queue rule will be applied.

Now see the traffic load:

```
[admin@MikroTik] interface> monitor-traffic Local
received-packets-per-second: 7
received-bits-per-second: 68kbps
sent-packets-per-second: 13
sent-bits-per-second: 135kbps

[admin@MikroTik] interface>
```

Probably, you want to exclude the server from being limited, if so, add a queue for it without any limitation (**max-limit=0/0** which means no limitation) and move it to the beginning of the list:

```
[admin@MikroTik] queue simple> add name=Server target-addresses=192.168.0.1/32 \
...\ interface=Local
[admin@MikroTik] queue simple> print
Flags: X - disabled, I - invalid, D - dynamic
0   name="Limit-Local" target-addresses=192.168.0.0/24 dst-address=0.0.0.0/0
    interface=Local parent=none priority=8 queue=default/default
    limit-at=0/0 max-limit=65536/131072 total-queue=default

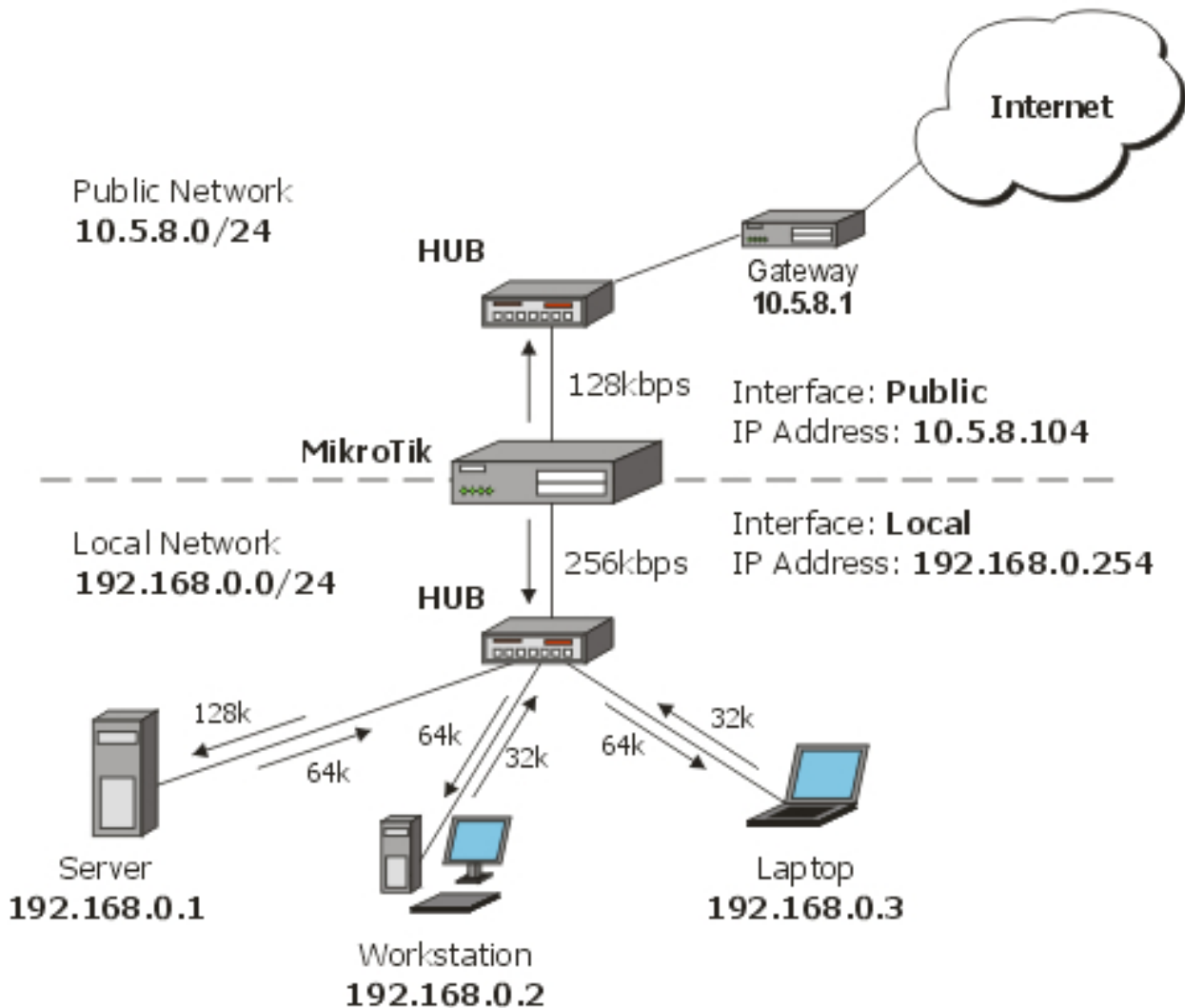
1   name="Server" target-addresses=192.168.0.1/32 dst-address=0.0.0.0/0
    interface=Local parent=none priority=8 queue=default/default
    limit-at=0/0 max-limit=0/0 total-queue=default
[admin@MikroTik] queue simple> mo 1 0
[admin@MikroTik] queue simple> print
Flags: X - disabled, I - invalid, D - dynamic
0   name="Server" target-addresses=192.168.0.1/32 dst-address=0.0.0.0/0
    interface=Local parent=none priority=8 queue=default/default
    limit-at=0/0 max-limit=0/0 total-queue=default

1   name="Limit-Local" target-addresses=192.168.0.0/24 dst-address=0.0.0.0/0
    interface=Local parent=none priority=8 queue=default/default
    limit-at=0/0 max-limit=65536/131072 total-queue=default
[admin@MikroTik] queue simple>
```

Queue Tree Example With Masquerading

In the previous example we dedicated 128Kib/s download and 64Kib/s upload traffic for the local network. In this example we will guarantee 256Kib/s download (128Kib/s for the server, 64Kib/s for the Workstation and also 64Kib/s for the Laptop) and 128Kib/s for upload (64/32/32Kib/s, respectively) for local network devices. Additionally, if there is spare bandwidth, share it among users equally. For example, if we turn off the laptop, share its 64Kib/s download and 32Kib/s upload to the Server and Workstation.

When using masquerading, you have to mark the outgoing connection with **new-connection-mark** and take the **mark-connection** action. When it is done, you can mark all packets which belong to this connection with the **new-packet-mark** and use the **mark-packet** action.



1. At first, mark the Server's download and upload traffic. With the first rule we will mark the outgoing connection and with the second one, all packets, which belong to this connection:

```
[admin@MikroTik] ip firewall mangle> add src-address=192.168.0.1/32 \
...\ action=mark-connection new-connection-mark=server-con chain=prerouting
[admin@MikroTik] ip firewall mangle> add connection-mark=server-con \
...\ action=mark-packet new-packet-mark=server chain=prerouting
[admin@MikroTik] ip firewall mangle> print
Flags: X - disabled, I - invalid, D - dynamic
0 chain=prerouting src-address=192.168.0.1 action=mark-connection
new-connection-mark=server-con

1 chain=prerouting connection-mark=server-con action=mark-packet
new-packet-mark=server
[admin@MikroTik] ip firewall mangle>
```

2. The same for Laptop and Workstation:

```
[admin@MikroTik] ip firewall mangle> add src-address=192.168.0.2 \
...\ action=mark-connection new-connection-mark=lap_works-con chain=prerouting
[admin@MikroTik] ip firewall mangle> add src-address=192.168.0.3 \
...\ action=mark-connection new-connection-mark=lap_works-con chain=prerouting
[admin@MikroTik] ip firewall mangle> add connection-mark=lap_works-con \
...\ action=mark-packet new-packet-mark=lap_work chain=prerouting
[admin@MikroTik] ip firewall mangle> print
Flags: X - disabled, I - invalid, D - dynamic
0 chain=prerouting src-address=192.168.0.1 action=mark-connection
```

```

new-connection-mark=server-con
1 chain=prerouting connection-mark=server-con action=mark-packet
  new-packet-mark=server
2 chain=prerouting src-address=192.168.0.2 action=mark-connection
  new-connection-mark=lap_works-con
3 chain=prerouting src-address=192.168.0.3 action=mark-connection
  new-connection-mark=lap_works-con
4 chain=prerouting connection-mark=lap_works-con action=mark-packet
  new-packet-mark=lap_work
[admin@MikroTik] ip firewall mangle>

```

As you can see, we marked connections that belong for Laptop and Workstation with the same flow.

3. In **/queue tree** add rules that will limit Server's download and upload:

```

[admin@MikroTik] queue tree> add name=Server-Download parent=Local \
\... limit-at=131072 packet-mark=server max-limit=262144
[admin@MikroTik] queue tree> add name=Server-Upload parent=Public \
\... limit-at=65536 packet-mark=server max-limit=131072
[admin@MikroTik] queue tree> print
Flags: X - disabled, I - invalid
0 name="Server-Download" parent=Local packet-mark=server limit-at=131072
  queue=default priority=8 max-limit=262144 burst-limit=0
  burst-threshold=0 burst-time=0s

1 name="Server-Upload" parent=Public packet-mark=server limit-at=65536
  queue=default priority=8 max-limit=131072 burst-limit=0
  burst-threshold=0 burst-time=0s
[admin@MikroTik] queue tree>

```

And similar config for Laptop and Workstation:

```

[admin@MikroTik] queue tree> add name=Laptop-Wkst-Down parent=Local \
\... packet-mark=lap_work limit-at=65535 max-limit=262144
[admin@MikroTik] queue tree> add name=Laptop-Wkst-Up parent=Public \
\... packet-mark=lap_work limit-at=32768 max-limit=131072
[admin@MikroTik] queue tree> print
Flags: X - disabled, I - invalid
0 name="Server-Download" parent=Local packet-mark=server limit-at=131072
  queue=default priority=8 max-limit=262144 burst-limit=0
  burst-threshold=0 burst-time=0s

1 name="Server-Upload" parent=Public packet-mark=server limit-at=65536
  queue=default priority=8 max-limit=131072 burst-limit=0
  burst-threshold=0 burst-time=0s

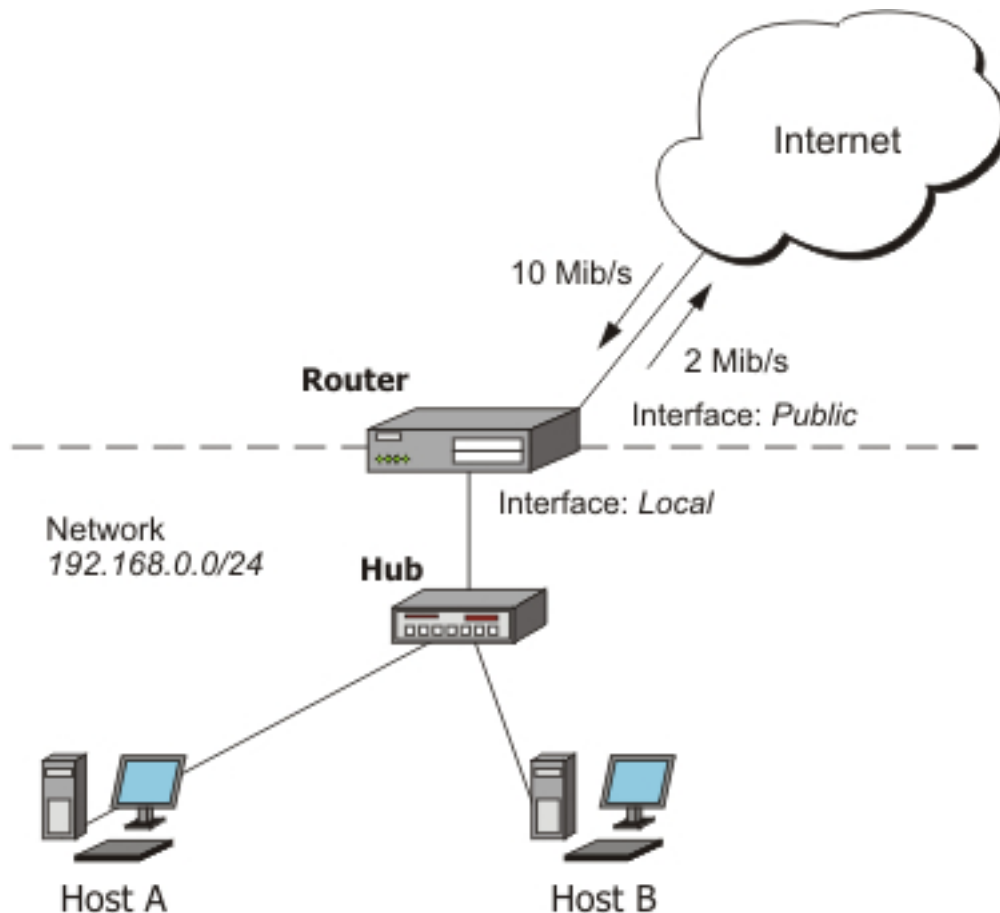
2 name="Laptop-Wkst-Down" parent=Local packet-mark=lap_work limit-at=65535
  queue=default priority=8 max-limit=262144 burst-limit=0
  burst-threshold=0 burst-time=0s

3 name="Laptop-Wkst-Up" parent=Public packet-mark=lap_work limit-at=32768
  queue=default priority=8 max-limit=131072 burst-limit=0
  burst-threshold=0 burst-time=0s
[admin@MikroTik] queue tree>

```

Equal bandwidth sharing among users

This example shows how to equally share 10Mibps download and 2Mbps upload among active users in the network **192.168.0.0/24**. If **Host A** is downloading 2 Mbps, **Host B** gets 8 Mbps and vice versa. There might be situations when both hosts want to use maximum bandwidth (10 Mbps), then they will receive 5 Mbps each, the same goes for upload. This setup is also valid for more than 2 users.



At first, mark all traffic, coming from local network **192.168.0.0/24** with a mark **users**:

```
/ip firewall mangle add chain=forward src-address=192.168.0.0/24 \
action=mark-connection new-connection-mark=users-con
/ip firewall mangle add connection-mark=users-con action=mark-packet \
new-packet-mark=users chain=forward
```

Now we will add 2 new PCQ types. The first, called **pcq-download** will group all traffic by destination address. As we will attach this queue type to the **Local** interface, it will create a dynamic queue for each destination address (user) which is downloading to the network **192.168.0.0/24**. The second type, called **pcq-upload** will group the traffic by source address. We will attach this queue to the **Public** interface so it will make one dynamic queue for each user who is uploading to Internet from the local network **192.168.0.0/24**.

```
/queue type add name=pcq-download kind=pcq pcq-classifier=dst-address
/queue type add name=pcq-upload kind=pcq pcq-classifier=src-address
```

Finally, make a queue tree for download traffic:

```
/queue tree add name=Download parent=Local max-limit=10240000
/queue tree add parent=Download queue=pcq-download packet-mark=users
```

And for upload traffic:

```
/queue tree add name=Upload parent=Public max-limit=2048000
/queue tree add parent=Upload queue=pcq-upload packet-mark=users
```

Note! If your ISP cannot guarantee you a fixed amount of traffic, you can use just one queue for upload and one for download, attached directly to the interface:

```
/queue tree add parent=Local queue=pcq-download packet-mark=users  
/queue tree add parent=Public queue=pcq-upload packet-mark=users
```