

Queues

- [Overview](#)
 - [Rate limitation principles](#)
- [Simple Queue](#)
 - [Configuration example](#)
- [Queue Tree](#)
 - [Configuration example](#)
- [Queue Types](#)
 - [Kinds](#)
 - [FIFO](#)
 - [RED](#)
 - [SFQ](#)
 - [PCQ](#)
 - [CoDel](#)
 - [FQ-Codel](#)
 - [CAKE](#)
- [Interface Queue](#)
- [Queue load visualization in GUI](#)

Overview

A queue is a collection of data packets collectively waiting to be transmitted by a network device using a pre-defined structure methodology. Queuing works almost on the same methodology used at banks or supermarkets, where the customer is treated according to its arrival.

Queues are used to:

- limit data rate for certain IP addresses, subnets, protocols, ports, etc.;
- limit peer-to-peer traffic;
- packet prioritization;
- configure traffic bursts for traffic acceleration;
- apply different time-based limits;
- share available traffic among users equally, or depending on the load of the channel

Queue implementation in MikroTik RouterOS is based on Hierarchical Token Bucket (HTB). HTB allows to the creation of a hierarchical queue structure and determines relations between queues. These hierarchical structures can be attached at two different places, the [Packet Flow diagram](#) illustrate both *in* and *postrouting* chains.

There are two different ways how to configure queues in RouterOS:

- **/queue simple** menu - designed to ease configuration of simple, every day queuing tasks (such as single client upload/download limitation, p2p traffic limitation, etc.).
- **/queue tree** menu - for implementing advanced queuing tasks (such as global prioritization policy, user group limitations). Requires marked packet flows from [/ip firewall mangle](#) facility.

RouterOS provides a possibility to configure queue in 8 levels - the first level is an interface queue from "/queue interface" menu and the other 7 are lower-level queues that can be created in Queue Simple and/or Queue Tree.

Rate limitation principles

Rate limiting is used to control the rate of traffic flow sent or received on a network interface. Traffic which rate that is less than or equal to the specified rate is sent, whereas traffic that exceeds the rate is dropped or delayed.

Rate limiting can be performed in two ways:

1. discard all packets that exceed rate limit – **rate-limiting (dropper or shaper)** (100% rate limiter when queue-size=0)
2. delay packets that exceed specific rate limit in the queue and transmit its when it is possible – **rate equalizing (scheduler)** (100% rate equalizing when queue-size=unlimited)

Next figure explains the difference between *rate limiting* and *rate equalizing*:

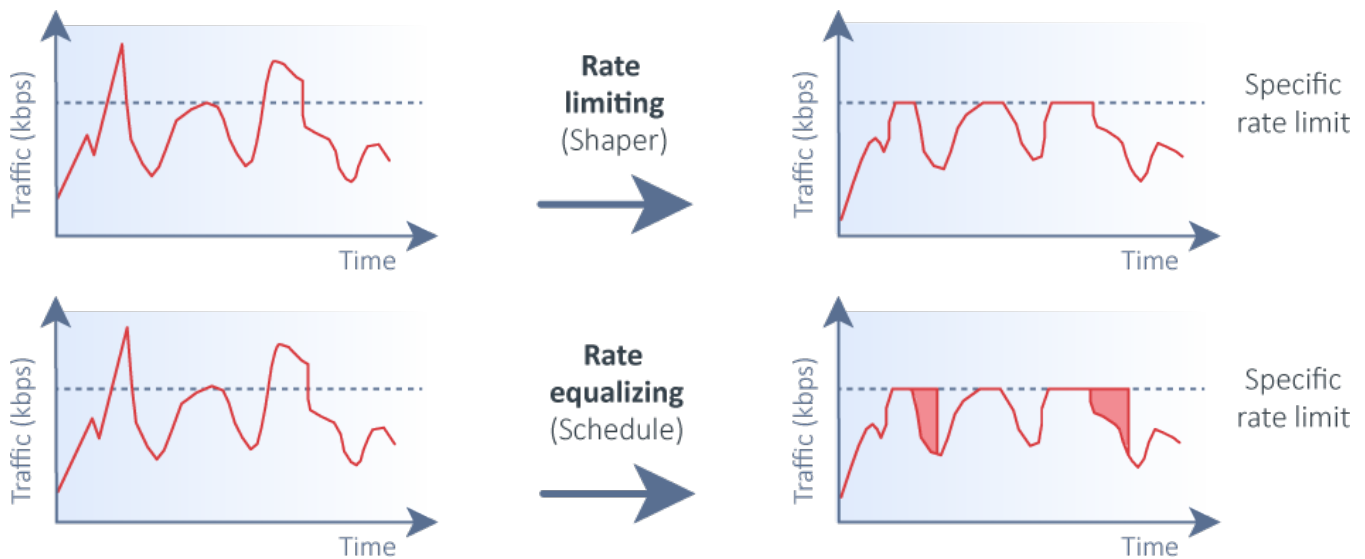


Figure 8.1. Principles of rate limiting and equalizing

As you can see in the first case all traffic exceeds a specific rate and is dropped. In another case, traffic exceeds a specific rate and is delayed in the queue and transmitted later when it is possible, but note that the packet can be delayed only until the queue is not full. If there is no more space in the queue buffer, packets are dropped.

For each queue we can define two rate limits:

- **CIR** (Committed Information Rate) – (**limit-at** in RouterOS) worst-case scenario, the flow will get this amount of traffic rate regardless of other traffic flows. At any given time, the bandwidth should not fall below this committed rate.
- **MIR** (Maximum Information Rate) – (**max-limit** in RouterOS) best-case scenario, the maximum available data rate for flow, if there is free any part of the bandwidth.

Simple Queue

```
/queue simple
```

A simple queue is a plain way how to limit traffic for a particular target. Also, you can use simple queues to build advanced QoS applications. They have useful integrated features:

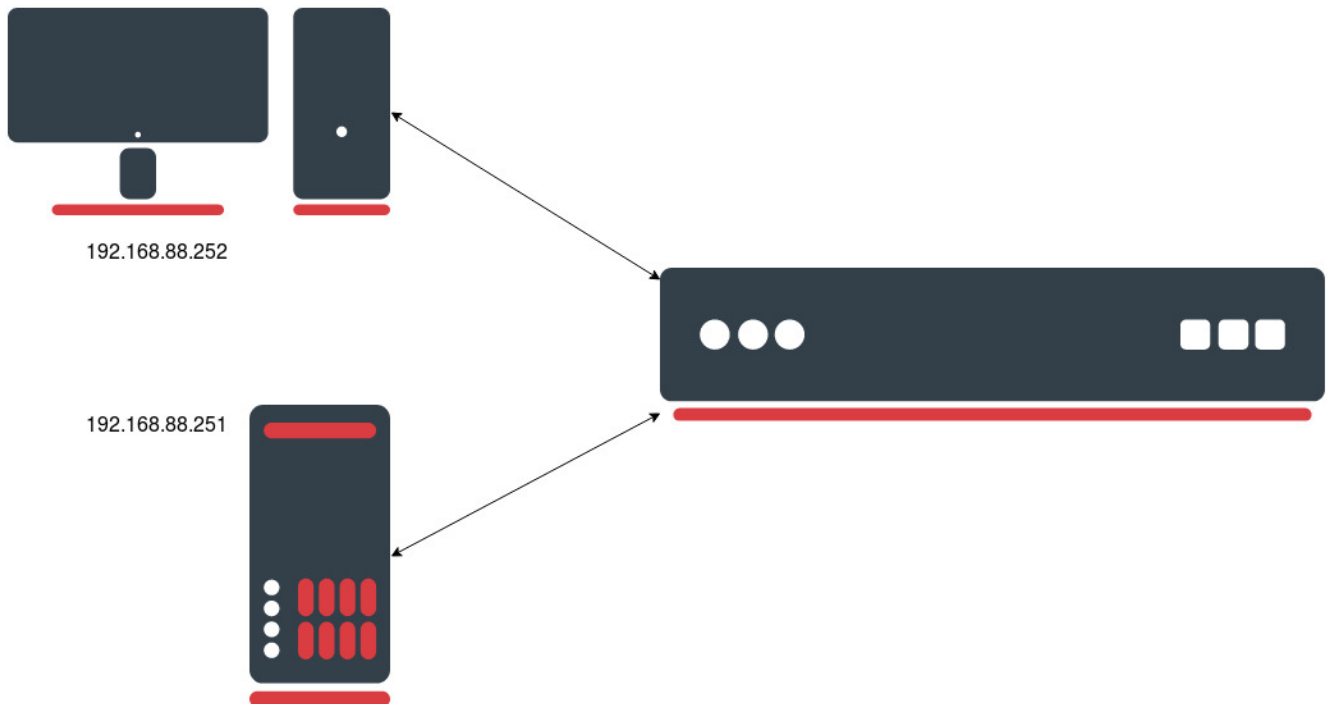
- peer-to-peer traffic queuing;
- applying queue rules on chosen time intervals;
- prioritization;
- using multiple packet marks from */ip firewall mangle*
- traffic shaping (scheduling) of bidirectional traffic (one limit for the total of upload + download)



Simple queues have a strict order - each packet must go through every queue until it reaches one queue which conditions fit packet parameters or until the end of the queues list is reached. For example, In the case of 1000 queues, a packet for the last queue will need to proceed through 999 queues before it will reach the destination.

Configuration example

In the following example, we have one SOHO device with two connected units PC and Server.



We have a 15 Mbps connection available from ISP in this case. We want to be sure the server receives enough traffic, so we will configure a simple queue with a *limit-at* parameter to guarantee a server to receive 5Mbps:

```
/queue simple
add limit-at=5M/5M max-limit=15M/15M name=queue1 target=192.168.88.251/32
```

That is all. The server will get 5 Mbps of traffic rate regardless of other traffic flows. If you are using the default configuration, be sure the FastTrack rule is disabled for this particular traffic, otherwise, it will bypass Simple Queues and they will not work.

Queue Tree

```
/queue tree
```

The queue tree creates only a one-directional queue in one of the HTBs. It is also the only way how to add a queue on a separate interface. This way it is possible to ease mangle configuration - you don't need separate marks for download and upload - only the upload will get to the Public interface and only the download will get to a Private interface. The main difference from Simple Queues is that the Queue tree is not ordered - all traffic passes it together.

Configuration example

In the following example, we will mark all the packets coming from preconfigured *in-interface-list=LAN* and will limit the traffic with a queue tree based on these packet marks.

Let's create a firewall address-list:

```
[admin@MikroTik] > /ip firewall address-list
add address=www.youtube.com list=Youtube
[admin@MikroTik] > ip firewall address-list print
Flags: X - disabled, D - dynamic
#    LIST
ADDRESS                                CREATION-TIME
TIMEOUT
0    Youtube                            www.youtube.
com                                     oct/17/2019 14:47:11
1    D ;;; www.youtube.com
    Youtube                            oct/17/2019 14:47:11
216.58.211.14                           oct/17/2019 14:47:11
2    D ;;; www.youtube.com
    Youtube                            oct/17/2019 14:47:11
216.58.207.238                           oct/17/2019 14:47:11
3    D ;;; www.youtube.com
    Youtube                            oct/17/2019 14:47:11
216.58.207.206                           oct/17/2019 14:47:11
4    D ;;; www.youtube.com
    Youtube                            oct/17/2019 14:47:11
172.217.21.174                           oct/17/2019 14:47:11
5    D ;;; www.youtube.com
    Youtube                            oct/17/2019 14:47:11
216.58.211.142                           oct/17/2019 14:47:11
6    D ;;; www.youtube.com
    Youtube                            oct/17/2019 14:47:21
172.217.22.174                           oct/17/2019 14:47:21
7    D ;;; www.youtube.com
    Youtube                            oct/17/2019 14:52:21
172.217.21.142
```

Mark packets with firewall mangle facility:

```
[admin@MikroTik] > /ip firewall mangle
add action=mark-packet chain=forward dst-address-list=Youtube in-interface-list=LAN new-packet-mark=pmark-
Youtube passthrough=yes
```

Configure the queue tree based on previously marked packets:

```
[admin@MikroTik] /queue tree
add max-limit=5M name=Limiting-Youtube packet-mark=pmark-Youtube parent=global
```

Check Queue tree stats to be sure traffic is matched:

```
[admin@MikroTik] > queue tree print stats
Flags: X - disabled, I - invalid
0    name="Limiting-Youtube" parent=global packet-mark=pmark-Youtube rate=0 packet-rate=0 queued-bytes=0 queued-
packets=0 bytes=67887 packets=355 dropped=0
```

Queue Types

```
/queue type
```

This sub-menu list by default created queue types and allows to add of new user-specific ones.

By default RouterOS creates the following pre-defined queue types:

```
[admin@MikroTik] > /queue type print
Flags: * - default
0 * name="default" kind=pfifo pfifo-limit=50

1 * name="ethernet-default" kind=pfifo pfifo-limit=50

2 * name="wireless-default" kind=sfq sfq-perturb=5 sfq-allot=1514

3 * name="synchronous-default" kind=red red-limit=60 red-min-threshold=10 red-max-threshold=50 red-burst=20
red-avg-packet=1000

4 * name="hotspot-default" kind=sfq sfq-perturb=5 sfq-allot=1514

5 * name="pcq-upload-default" kind=pcq pcq-rate=0 pcq-limit=50KiB pcq-classifier=src-address pcq-total-
limit=2000KiB pcq-burst-rate=0 pcq-burst-threshold=0 pcq-burst-time=10s pcq-src-address-mask=32
pcq-dst-address-mask=32 pcq-src-address6-mask=128 pcq-dst-address6-mask=128

6 * name="pcq-download-default" kind=pcq pcq-rate=0 pcq-limit=50KiB pcq-classifier=dst-address pcq-total-
limit=2000KiB pcq-burst-rate=0 pcq-burst-threshold=0 pcq-burst-time=10s pcq-src-address-mask=32
pcq-dst-address-mask=32 pcq-src-address6-mask=128 pcq-dst-address6-mask=128

7 * name="only-hardware-queue" kind=none

8 * name="multi-queue-ethernet-default" kind=mq-pfifo mq-pfifo-limit=50

9 * name="default-small" kind=pfifo pfifo-limit=10
```

All MikroTik products have default queue type **"only-hardware-queue"** with "kind=none". "only-hardware-queue" leaves interface with only hardware transmit descriptor ring buffer which acts as a queue in itself. Usually, at least 100 packets can be queued for transmit in transmit descriptor ring buffer. Transmit descriptor ring buffer size and the number of packets that can be queued in it varies for different types of ethernet MACs. Having no software queue is especially beneficial on SMP systems because it removes the requirement to synchronize access to it from different CPUs/cores which is resource-intensive. Having the possibility to set **"only-hardware-queue"** requires support in an ethernet driver so it is available only for some ethernet interfaces mostly found on RouterBOARDS.

A **"multi-queue-ethernet-default"** can be beneficial on SMP systems with ethernet interfaces that have support for multiple transmit queues and have a Linux driver support for multiple transmit queues. By having one software queue for each hardware queue there might be less time spent on synchronizing access to them.



Improvement from only-hardware-queue and multi-queue-ethernet-default is present only when there is no "/queue tree" entry with a particular interface as a parent.

Kinds

Queue kinds are packet processing algorithms. Kind describe which packet will be transmitted next in the line. RouterOS supports the following Queueing kinds:

- FIFO (BFIFO, PFIFO, MQ PFIFO)
- RED
- SFQ
- PCQ

FIFO

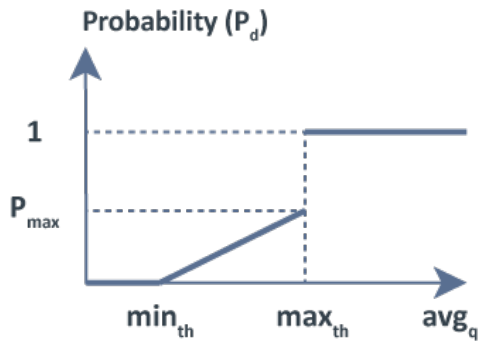
These kinds are based on the FIFO algorithm (First-In-First-Out). The difference between **PFIFO** and **BFIFO** is that one is measured in packets and the other one in bytes. These queues use **pfifo-limit** and **bfifo-limit** parameters.

Every packet that cannot be enqueued (if the queue is full), is dropped. Large queue sizes can increase latency but utilize the channel better.

MQ-PFIFO is *pfifo* with support for multiple transmit queues. This queue is beneficial on SMP systems with ethernet interfaces that have support for multiple transmit queues and have a Linux driver support for multiple transmit queues (mostly on x86 platforms). This kind uses the **mq-pfifo-limit** parameter.

RED

Random Early Drop is a queuing mechanism that tries to avoid network congestion by controlling the average queue size. The average queue size is compared to two thresholds: a minimum (\min_{th}) and maximum (\max_{th}) threshold. If the average queue size (avg_q) is less than the minimum threshold, no packets are dropped. When the average queue size is greater than the maximum threshold, all incoming packets are dropped. But if the average queue size is between the minimum and maximum thresholds packets are randomly dropped with probability P_d where probability is exact a function of the average queue size: $P_d = P_{\max}(\text{avg}_q - \min_{th}) / (\max_{th} - \min_{th})$. If the average queue grows, the probability of dropping incoming packets grows too. P_{\max} - ratio, which can adjust the packet discarding probability abruptness, (the simplest case P_{\max} can be equal to one. The 8.2 diagram shows the packet drop probability in the RED algorithm.



$\text{avg}_q \leq \min_{th}$ - no dropped packets
 $\text{avg}_q \geq \max_{th}$ - dropped all packets
 $\min_{th} \leq \text{avg}_q \leq \max_{th}$ - packets are dropped with probability P_d

Figure 8.2. RED operation

SFQ

Stochastic Fairness Queuing (SFQ) is ensured by hashing and round-robin algorithms. SFQ is called "Stochastic" because it does not really allocate a queue for each flow, it has an algorithm that divides traffic over a limited number of queues (1024) using a hashing algorithm.

Traffic flow may be uniquely identified by 4 options (*src-address*, *dst-address*, *src-port*, and *dst-port*), so these parameters are used by the SFQ hashing algorithm to classify packets into one of 1024 possible sub-streams. Then round-robin algorithm will start to distribute available bandwidth to all sub-streams, on each round giving **sfq-allot** bytes of traffic. The whole SFQ queue can contain 128 packets and there are 1024 sub-streams available. The 8.3 diagram shows the SFQ operation:

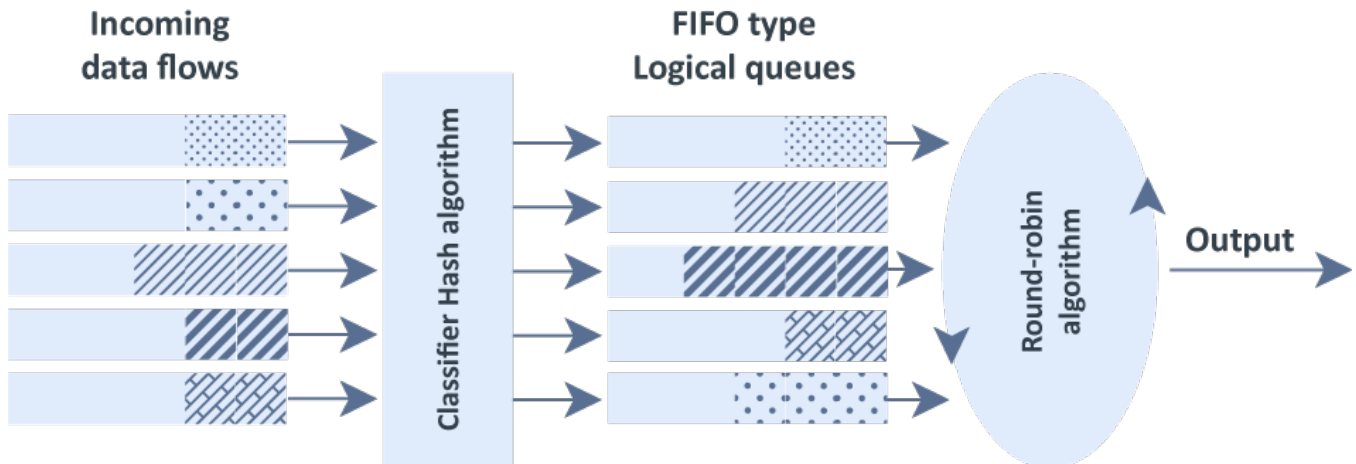


Figure 8.3. SFQ operation

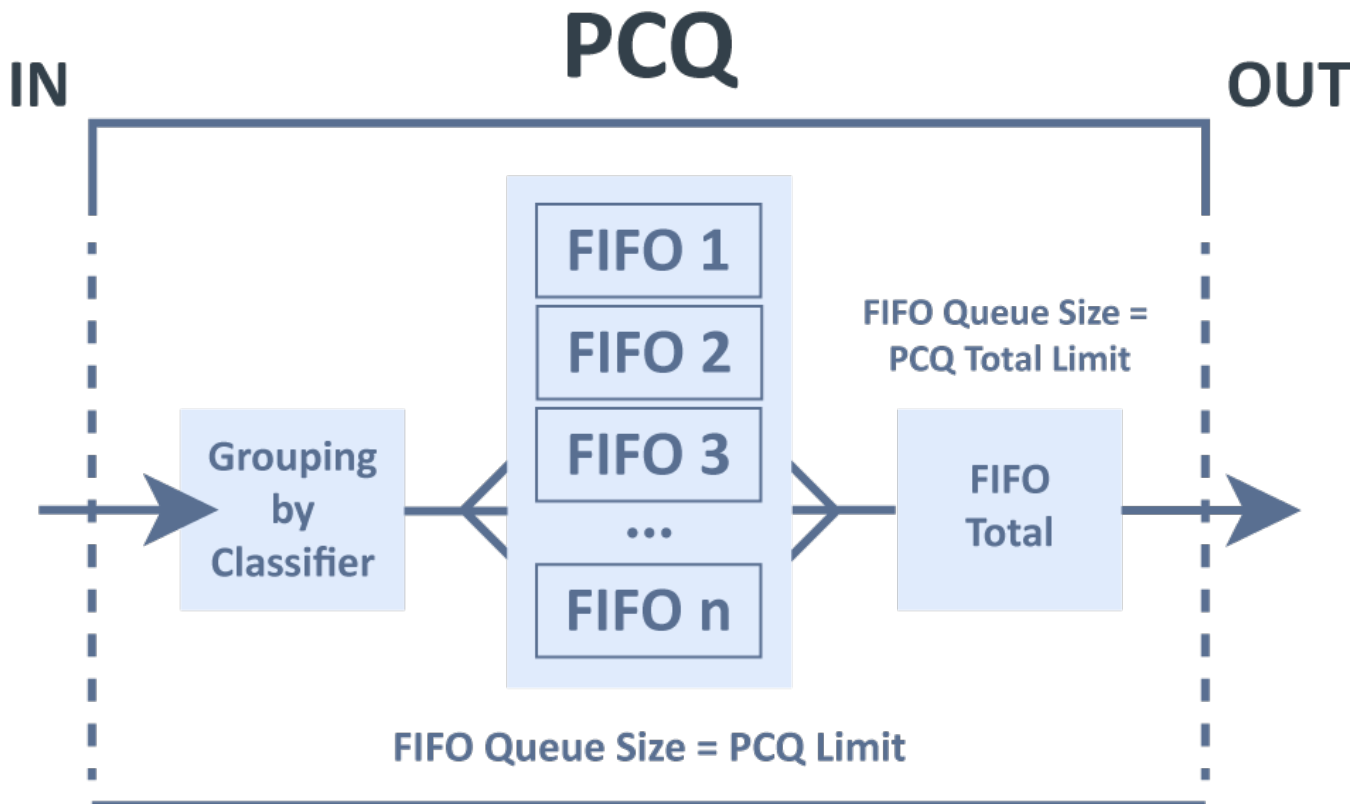
PCQ

PCQ algorithm is very simple - at first, it uses selected classifiers to distinguish one sub-stream from another, then applies individual FIFO queue size and limitation on every sub-stream, then groups all sub-streams together and applies global queue size and limitation.

PCQ parameters:

- **pcq-classifier** (dst-address | dst-port | src-address | src-port; default: "") : selection of sub-stream identifiers
- **pcq-rate** (number): maximal available data rate of each sub-stream
- **pcq-limit** (number): queue size of single sub-stream (in KiB)
- **pcq-total-limit** (number): maximum amount of queued data in all sub-streams (in KiB)

It is possible to assign a speed limitation to sub-streams with the **pcq-rate** option. If "pcq-rate=0" sub-streams will divide available traffic equally.



For example, instead of having 100 queues with 1000kbps limitation for download, we can have one PCQ queue with 100 sub-streams

PCQ has burst implementation identical to Simple Queues and Queue Tree:

- **pcq-burst-rate** (number): maximal upload/download data rate which can be reached while the burst for substream is allowed
- **pcq-burst-threshold** (number): this is the value of burst on/off switch
- **pcq-burst-time** (time): a period of time (in seconds) over which the average data rate is calculated. (This is NOT the time of actual burst)

PCQ also allows using different size IPv4 and IPv6 networks as sub-stream identifiers. Before it was locked to a single IP address. This is done mainly for IPv6 as customers from an ISP point of view will be represented by /64 network, but devices in customers network will be /128. PCQ can be used for both of these scenarios and more. PCQ parameters:

- **pcq-dst-address-mask** (number): the size of the IPv4 network that will be used as a dst-address sub-stream identifier
- **pcq-src-address-mask** (number): the size of the IPv4 network that will be used as an src-address sub-stream identifier
- **pcq-dst-address6-mask** (number): the size of the IPV6 network that will be used as a dst-address sub-stream identifier
- **pcq-src-address6-mask** (number): the size of the IPV6 network that will be used as an src-address sub-stream identifier



The following queue kinds CoDel, FQ-Codel, and CAKE available since RouterOS version 7.1beta3.

CoDel

CoDel (Controlled-Delay Active Queue Management) algorithm uses the local minimum queue as a measure of the persistent queue, similarly, it uses a minimum delay parameter as a measure of the standing queue delay. Queue size is calculated using packet residence time in the queue.

Properties

Property	Description
codel-ce-threshold (default:)	Marks packets above a configured threshold with ECN.
codel-ecn (default: no)	An option is used to mark packets instead of dropping them.
codel-interval (default: 100ms)	Interval should be set on the order of the worst-case RTT through the bottleneck giving endpoints sufficient time to react.
codel-limit (default: 1000)	Queue limit, when the limit is reached, incoming packets are dropped.
codel-target (default: 5ms)	Represents an acceptable minimum persistent queue delay.

FQ-Codel

CoDel - Fair Queuing (FQ) with Controlled Delay (CoDel) uses a randomly determined model to classify incoming packets into different flows and is used to provide a fair share of the bandwidth to all the flows using the queue. Each flow is managed using CoDel queuing discipline which internally uses a FIFO algorithm.

Properties

Property	Description
fq-codel-ce-threshold (default:)	Marks packets above a configured threshold with ECN.
fq-codel-ecn (default: yes)	An option is used to mark packets instead of dropping them.
fq-codel-flows (default: 1024)	A number of flows into which the incoming packets are classified.
fq-codel-interval (default: 100ms)	Interval should be set on the order of the worst-case RTT through the bottleneck giving endpoints sufficient time to react.
fq-codel-limit (default: 10240)	Queue limit, when the limit is reached, incoming packets are dropped.
fq-codel-memlimit (default: 32.0MiB)	A total number of bytes that can be queued in this FQ-CoDel instance. Will be enforced from the <i>fq-codel-limit</i> parameter.
fq-codel-quantum (default: 1514)	A number of bytes used as 'deficit' in the fair queuing algorithm. Default (1514 bytes) corresponds to the Ethernet MTU plus the hardware header length of 14 bytes.
fq-codel-target (default: 5ms)	Represents an acceptable minimum persistent queue delay.

CAKE

CAKE - Common Applications Kept Enhanced (CAKE) implemented as a *queue discipline* (qdisc) for the Linux kernel uses COBALT (AQM algorithm combining Codel and BLUE) and a variant of DRR++ for flow isolation. In other words, Cake's fundamental design goal is user-friendliness. All settings are optional; the default settings are chosen to be practical in most common deployments. In most cases, the configuration requires only a bandwidth parameter to get useful results,

Properties

Property	Description
cake-ack-filter (default: none)	
cake-atm (default:)	Compensates for ATM cell framing, which is normally found on ADSL links.
cake-autorate-ingress (yes/no, default:)	Automatic capacity estimation based on traffic arriving at this qdisc. This is most likely to be useful with cellular links, which tend to change quality randomly. The Bandwidth Limit parameter can be used in conjunction to specify an initial estimate. The shaper will periodically be set to a bandwidth slightly below the estimated rate. This estimator cannot estimate the bandwidth of links downstream of itself.

cake-bandwidth (default:)	Sets the shaper bandwidth.
cake-diffserv (default: diffserv3)	<p>CAKE can divide traffic into "tins" based on the Diffserv field:</p> <ul style="list-style-type: none"> • diffserv4 Provides a general-purpose Diffserv implementation with four tins: Bulk (CS1), 6.25% threshold, generally low priority. Best Effort (general), 100% threshold. Video (AF4x, AF3x, CS3, AF2x, CS2, TOS4, TOS1), 50% threshold. Voice (CS7, CS6, EF, VA, CS5, CS4), 25% threshold. • diffserv3 (default) Provides a simple, general-purpose Diffserv implementation with three tins: Bulk (CS1), 6.25% threshold, generally low priority. Best Effort (general), 100% threshold. Voice (CS7, CS6, EF, VA, TOS4), 25% threshold, reduced Codel interval.
cake-flowmode (dsthost/dual-dsthost/dual-srchohost/flowblind/flows/hosts/srchohost/triple-isolate, default: triple-isolate)	<ul style="list-style-type: none"> • flowblind - Disables flow isolation; all traffic passes through a single queue for each tin. • srchohost - Flows are defined only by source address. • dsthost Flows are defined only by destination address. • hosts - Flows are defined by source-destination host pairs. This is host isolation, rather than flow isolation. • flows - Flows are defined by the entire 5-tuple of source address, a destination address, transport protocol, source port, and destination port. This is the type of flow isolation performed by SFQ and fq_codel. • dual-srchohost Flows are defined by the 5-tuple, and fairness is applied first over source addresses, then over individual flows. Good for use on egress traffic from a LAN to the internet, where it'll prevent anyone LAN host from monopolizing the uplink, regardless of the number of flows they use. • dual-dsthost Flows are defined by the 5-tuple, and fairness is applied first over destination addresses, then over individual flows. Good for use on ingress traffic to a LAN from the internet, where it'll prevent anyone LAN host from monopolizing the downlink, regardless of the number of flows they use. • triple-isolate - Flows are defined by the 5-tuple, and fairness is applied over source *and* destination addresses intelligently (ie. not merely by host-pairs), and also over individual flows. • nat Instructs Cake to perform a NAT lookup before applying flow- isolation rules, to determine the true addresses and port numbers of the packet, to improve fairness between hosts "inside" the NAT. This has no practical effect in "flowblind" or "flows" modes, or if NAT is performed on a different host. • nonat (default) The cake will not perform a NAT lookup. Flow isolation will be performed using the addresses and port numbers directly visible to the interface Cake is attached to.
cake-memlimit (default:)	Limit the memory consumed by Cake to LIMIT bytes. By default, the limit is calculated based on the bandwidth and RTT settings.
cake-mpu (-64 ... 256, default:)	Rounds each packet (including overhead) up to a minimum length BYTES.
cake-nat (default: no)	Instructs Cake to perform a NAT lookup before applying a flow-isolation rule.
cake-overhead (-64 ... 256, default:)	Adds BYTES to the size of each packet. BYTES may be negative.
cake-overhead-scheme (default:)	
cake-rtt (default: 100ms)	Manually specify an RTT. Default 100ms is suitable for most Internet traffic.
cake-rtt-scheme (datacentre /internet/interplanetary/lan /metro/none/oceanic/regional /satellite, default:)	<ul style="list-style-type: none"> • datacentre - For extremely high-performance 10GigE+ networks only. Equivalent to RTT 100us. • lan - For pure Ethernet (not Wi-Fi) networks, at home or in the office. Don't use this when shaping for an Internet access link. Equivalent to RTT 1ms. • metro - For traffic mostly within a single city. Equivalent to RTT 10ms. regional For traffic mostly within a European-sized country. Equivalent to RTT 30ms. • internet (default) This is suitable for most Internet traffic. Equivalent to RTT 100ms. • oceanic - For Internet traffic with generally above-average latency, such as that suffered by Australasian residents. Equivalent to RTT 300ms. • satellite - For traffic via geostationary satellites. Equivalent to RTT 1000ms. • interplanetary - So named because Jupiter is about 1 light-hour from Earth. Use this to (almost) completely disable AQM actions. Equivalent to RTT 3600s.
cake-wash (default: no)	Apply the wash option to clear all extra DiffServ (but not ECN bits), after priority queuing has taken place.

Interface Queue




```
/queue interface
```

Before sending data over an interface, it is processed by the queue. This sub-menu lists all available interfaces in RouterOS and allows to change queue type for a particular interface. The list is generated automatically.

```
[admin@MikroTik] > queue interface print
Columns: INTERFACE, QUEUE, ACTIVE-QUEUE
# INTERFACE QUEUE ACTIVE-QUEUE
0 ether1 only-hardware-queue only-hardware-queue
1 ether2 only-hardware-queue only-hardware-queue
2 ether3 only-hardware-queue only-hardware-queue
3 ether4 only-hardware-queue only-hardware-queue
4 ether5 only-hardware-queue only-hardware-queue
5 ether6 only-hardware-queue only-hardware-queue
6 ether7 only-hardware-queue only-hardware-queue
7 ether8 only-hardware-queue only-hardware-queue
8 ether9 only-hardware-queue only-hardware-queue
9 ether10 only-hardware-queue only-hardware-queue
10 sfp-sfpplus1 only-hardware-queue only-hardware-queue
11 wlan1 wireless-default wireless-default
12 wlan2 wireless-default wireless-default
```

Queue load visualization in GUI

In Winbox and Webfig, a green, yellow, or red icon visualizes each Simple and Tree queue usage based on max-limit.

	0% - 50% of max-limit used
	50% - 75% of max-limit used
	75% - 100% of max-limit used