Manual:Queue

Applies to RouterOS: v6.0+

Contents

Sub Categories

List of reference sub-pages	Case studies	List of examples
Queue has no subpages to list.	▪ TE tunnel auto bandwidth ▪ TE Tunnels ▪ Queues - Burst ▪ Queues - PCQ ▪ Queue Size ▪ HTB ▪ Packet Flow v6	▪ TE Tunnels Example ▪ Queues - PCQ Examples ▪ Simple TE ▪ Multiple TE VPLS

Summary

Queues are used to limit and prioritize traffic:

- limit data rate for certain IP addresses, subnets, protocols, ports, and other parameters
- limit peer-to-peer traffic
- prioritize some packet flows over others
- configure traffic bursts for faster web browsing
- apply different limits based on time
- 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 create hierarchical queue structure and determine relations between queues.

In RouterOS, these hierarchical structures can be attached at two different places, packet flow diagram illustrate both Input and Postrouting chains (https://wiki.mikrotik.com/wiki/Manual:Packet_Flow#Packet_Flow_Chains).

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

- **/queue simple** menu - designed to ease configuration of simple, everyday 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.

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 queue and transmit its when it is possible – ***rate equalizing (scheduler) "***(100% rate equalizing when queue-size=unlimited)

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

File:Image8001.gif

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 a 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, 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, maximum available data rate for flow, if there is free any part of bandwidth.

Simple Queues

Sub-menu: `/queue simple`

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 (scheduling) of bidirectional traffic (one limit for the total of upload + download)

Configuration Example

Assume we have network topology like Figure 8.6 and we want to limited download and upload for private network (upload - 256kbps, and download – 512kbps).

File:Image8006.gif

Add a simple queue rule, which will limit the download traffic to 512kbps and upload to 256kbps for the network **10.1.1.0/24**, served by the interface **Ether2**:

```
[admin@MikroTik] /queue simple> add name=private target=10.1.1.0/24 max-limit=256K/512K dst=ether2
```

In this case statement works right also if we indicate only one of parameters: *"target="* or *"dst="*, because both of these define where and for which traffic this queue will be implemented.

Check your configuration:

```
[admin@Augsha] /queue simple> print

Flags: X - disabled, I - invalid, D - dynamic
 0    name="private" target=10.1.1.0/24 dst-address=0.0.0.0/0
      target=ether2 parent=none dst="" priority=8
      queue=default-small/default-small limit-at=0/0 max-limit=256k/512k
      burst-limit=0/0 burst-threshold=0/0 burst-time=0s/0s
      total-queue=default-small
```

The **max-limit** parameter cuts down the maximum available bandwidth. The value **max-limit=256k/512k** means that clients from the private networks will get maximum of 512kbps for download and 256kbps for upload. The **target** allows defining the source IP addresses to which the queue rule will be applied.

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). Move this rule to the beginning of the list, because items in /queue simple are executed in order one by one if router finds rule that satisfy certain packet next rules aren't compared:

```
[admin@MikroTik] /queue simple> add name=server target=10.1.1.1/32 max-limit=0/0 interface=ether2
```

Flow Identifiers

- **target** (multiple choice: IP address/netmask) : list of IP address ranges that will be limited by this queue.
- **interface** (Name of the interface, or *all*) : identifies interface the target is connected to. Useful when it is not possible to specify targets addresses.

Note: Since RouterOS v6 these settings are combined in the option **target** where you can specify either of the above. **Target** is to be viewed from perspective of the target. If you want to limit your users' upload capability, set "target upload".

Each of these two properties can be used to determine which direction is target upload and which is download.

Be careful to configure both of these options for the same queue - in case they will point to opposite directions queue will not work.

If neither value of **target** nor of **interface** is specified, the queue will not be able to make the difference between upload and download and will limit all traffic twice.

Other properties

- **name** (Text) : Unique queue identifier that can be used as **parent** option value for other queues

- *both* - limit both download and upload traffic
- *upload* - limit only traffic to the target
- *download* - limit only traffic from the target

- **time** (*TIME-TIME,sun,mon,tue,wed,thu,fri,sat - TIME* is local time, all day names are optional; default: not set) : allow to specify time when particular queue will be active. Router must have correct time settings.
- **dst-address** (IP address/netmask) : allows to select only specific stream (from target address to this destination address) for limitation explain what is target and what is dst and what is upload and what not
- **packet-marks** (Comma separated list of packet mark names) : allows to use marked packets from **/ip firewall mangle**. Take look at the RouterOS packet flow diagram. It is necessary to mark packets before the simple queues (before *global-in* HTB queue) or else target's download limitation will not work. The only mangle chain before *global-in* is *prerouting*.

HTB Properties

- **parent** (Name of parent simple queue, or *none*) : assigns this queue as a child queue for selected target {{{...}}}. Target queue can be HTB queue or any other previously created simple queue. In order for traffic to reach child queues, parent queues must capture all necessary traffic.
- **priority** (1..8) : Prioritize one child queue over other child queue. Does not work on parent queues (if queue has at least one child). One is the highest, eight is the lowest priority. Child queue with higher priority will have chance to reach its **max-limit** before child with lower priority. Priority have nothing to do with bursts.
- **queue** (*SOMETHING/SOMETHING*) : Choose the type of the upload/download queue. Queue types can be created in **/queue type**.
- **limit-at** (*NUMBER/NUMBER*) : normal upload/download data rate that is guaranteed to a target
- **max-limit** (*NUMBER/NUMBER*) : maximal upload/download data rate that is allowed for a target to reach to reach what
- **burst-limit** (*NUMBER/NUMBER*) : maximal upload/download data rate which can be reached while the burst is active
- **burst-time** (*TIME/TIME*) : period of time, in seconds, over which the average upload/download data rate is calculated. (This is NOT the time of actual burst)
- **burst-threshold** (*NUMBER/NUMBER*) : when average data rate is below this value - burst is allowed, as soon as average data rate reach this value - burst is denied. (basically this is burst on/off switch). For optimal burst behavior this value should above **limit-at** value and below **max-limit** value

And corresponding options for *global-total* HTB queue:

- **total-queue** (*SOMETHING/SOMETHING*): corresponds to **queue**
- **total-limit-at** (*NUMBER/NUMBER*): corresponds to **limit-at**
- **total-max-limit** (*NUMBER/NUMBER*): corresponds to **max-limit**
- **total-burst-limit** (*NUMBER/NUMBER*): corresponds to **burst-limit**
- **total-burst-time** (*TIME/TIME*): corresponds to **burst-time**

- **total-burst-threshold** (*NUMBER/NUMBER*): corresponds to **burst-threshold**

Good practice suggests that:

> Sum of children's **limit-at** values must be less or equal to **max-limit** of the parent.
> Every child's **max-limit** must be less than **max-limit** of the parent. This way you will leave some traffic for the other child queues, and they will be able to get traffic without fighting for it with other child queues.

Statistics

- **rate** (read-only/read-only) : average queue passing data rate in bytes per second
- **packet-rate** (read-only/read-only) : average queue passing data rate in packets per second
- **bytes** (read-only/read-only) : number of bytes processed by this queue
- **packets** (read-only/read-only) : number of packets processed by this queue
- **queued-bytes** (read-only/read-only) : number of bytes waiting in the queue
- **queued-packets** (read-only/read-only) : number of packets waiting in the queue
- **dropped** (read-only/read-only) : number of dropped packets
- **borrows** (read-only/read-only) : packets that passed queue over its "limit-at" value (and was unused and taken away from other queues)
- **lends** (read-only/read-only) : packets that passed queue below its "limit-at" value OR if queue is a parent - sum of all child borrowed packets
- **pcq-queues** (read-only/read-only) : number of PCQ substreams, if queue type is PCQ

And corresponding options for *global-total* HTB queue:

- **total-rate** (read-only): corresponds to *rate*
- **total-packet-rate** (read-only): corresponds to *packet-rate*
- **total-bytes** (read-only): corresponds to *bytes*
- **total-packets** (read-only): corresponds to *packets*
- **total-queued-bytes** (read-only): corresponds to *queued-bytes*
- **total-queued-packets** (read-only): corresponds to *queued-packets*
- **total-dropped** (read-only): corresponds to *dropped*
- **total-lends** (read-only): corresponds to *lends*
- **total-borrows** (read-only): corresponds to *borrows*
- **total-pcq-queues** (read-only): corresponds to *pcq-queues*

Queue Tree

Sub-menu: `/queue tree`

Queue tree creates only one directional queue in one of the HTBs. It is also the only way how to add queue on the separate interface. This way it is possible to ease mangle configuration - you don't need separate marks for download and upload - only upload will get to Public interface and only download will get to Private interface.

Queue tree is not ordered - all traffic pass it together.

Read more about HTB and see configuration examples.

Flow Identifiers

- **name** (Text) : Unique queue identifier that can be used as **parent** option value for other queues
- **packet-marks** (Comma separated list of) : allows to use marked packets from **/ip firewall mangle**. Take look at this packet flow diagram. You need to make sure that packets are marked before the simple queues (before global-in HTB queue)

HTB Properties

- **parent** (Name of , or *none*) : assigns this queue as a child queue for selected target. Target queue can be HTB queue or any other previously created queue
- **priority** (1..8) : Prioritize one child queue over other child queue. Does not work on parent queues (if queue has at least one child). One is the highest, eight is the lowest priority. Child queue with higher priority will have chance to reach its **max-limit** before child with lower priority. Priority have nothing to do with bursts.
- **queue** (*SOMETHING*) : Choose the type of the queue. Queue types can be created here
- **limit-at** (*NUMBER*) : normal data rate that is guaranteed to a target
- **max-limit** (*NUMBER*) : maximal data rate that is allowed for a target to reach
- **burst-limit** (*NUMBER*) : maximal data rate which can be reached while the burst is active
- **burst-time** (*TIME*) : period of time, in seconds, over which the average data rate is calculated. (This is NOT the time of actual burst)
- **burst-threshold** (*NUMBER*) : when average data rate is below this value - burst is allowed, as soon as average data rate reach this value - burst is denied. (basically this is burst on/off switch). For optimal burst behavior this value should above **limit-at** value and below **max-limit**

value

Statistics

Command: `/queue tree print stats`

- **rate** (read-only) : average queue passing data rate in bytes per second
- **packet-rate** (read-only) : average queue passing data rate in packets per second
- **bytes** (read-only) : number of bytes processed by this queue
- **packets** (read-only) : number of packets processed by this queue
- **queued-bytes** (read-only) : number of bytes waiting in the queue
- **queued-packets** (read-only) : number of packets waiting in the queue
- **dropped** (read-only) : number of dropped packets
- **borrows** (read-only) : packets that passed queue over its "limit-at" value (and was unused and taken away from other queues)
- **lends** (read-only) : packets that passed queue below its "limit-at" value OR if queue is a parent - sum of all child borrowed packets
- **pcq-queues** (read-only) : number of PCQ substreams, if queue type is PCQ

Queue Types

Sub-menu: `/queue type`

This sub-menu lists by default created queue types and allow to add new user-specific ones.

By default RouterOS creates following pre-defined queue types:

```
[admin@MikroTik] /queue type> print
 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="only-hardware-queue" kind=none

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

 7 name="default-small" kind=pfifo pfifo-limit=10
```

Note: Starting from v5.8 there is new kind **none** and new default queue **only-hardware-queue**. All RouterBOARDS will have this new queue type set as default interface queue

only-hardware-queue leaves interface with only hw 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 amount 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 expensive.

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 for synchronizing access to them.

Note: having possibility to set only-hardware-queue requires support in ethernet driver so it is available only for some ethernet interfaces mostly found on RBs.

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

Kinds

Queue kinds or Queuing (scheduling) algorithms describe which packet will be transmitted next in line. RouterOS supports several queuing algorithms:

- BFIFO, PFIFO, MQ PFIFO
- RED
- SFQ
- PCQ

PFIFO, BFIFO and MQ PFIFO

These queuing 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.

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

These queues uses **pfifo-limit** and **bfifo-limit** parameters.

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.

mq-pfifo uses `mq-pfifo-limit` parameter.

RED

Random Early Drop is a queuing mechanism which 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 average queue size (avg_q) is less than the minimum threshold, no packets are dropped. When 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: $P_d = P_{max}(avg_q - min_{th})/ (max_{th} - min_{th})$. If average queue grows, the probability for 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 diagram in Figure 8.2. shows the packet drop probability in RED algorithm.

File:Image8002.gif

SFQ

Stochastic Fairness Queuing (SFQ) is ensured by hashing and round-robin algorithms. A traffic flow may be uniquely identified by a 4 options(src-address, dst-address, src-port and dst-port), so these parameters are used by 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.

File:Image8003.gif

SFQ is called "Stochastic" because it does not really allocate a queue for each flow, it has an algorithm which divides traffic over a limited number of queues (1024) using a hashing algorithm.

PCQ

Per Connection Queuing (PCQ) is a similar to SFQ, but it has additional features.

- It is possible to choose flow identifiers (from dst-address | dst-port | src-address | src-port). For example if you classify flows by src-address on local interface (interface with your clients), each PCQ sub-stream will be one particular client's upload.
- It is possible to assign speed limitation to sub-streams with **pcq-rate** option. If **pcq-rate**=0 sub-streams will divide available traffic equally.

More information and examples of PCQ are available here.

Properties

Properties that start with particular queue kind name, is applied only to particular kind. For example all properties starting with pcq applies only to queue **kind=pcq**.

Property	Description
bfifo-limit (*integer [1000..4294967295]*; Default: **15000**)	Maximum number of bytes that the BFIFO queue can hold. Applies if **kind** is bfifo.
kind (*bfifo* | *mq-pfifo* | *none* | *pcq* | *pfifo* | *red* | *sfq*; Default:)	Kind of particular queue type. Read more >>
mq-pfifo-limit (*integer [1..4294967295]*; Default: **50**)	Multi-queue PFIFO limit.
name (*string*; Default:)	Descriptive name of queue type
pcq-burst-rate (*integer [0..4294967295]*; Default: **0**)	Maximal upload/download data rate which can be reached while the burst for substream is allowed
pcq-burst-threshold (*integer [0..4294967295]*; Default: **0**)	This is value of burst on/off switch
pcq-burst-time (*time*; Default: **10s**)	Period of time, in seconds, over which the average data rate is calculated. (This is NOT the time of actual burst)
pcq-classifier (*list of src-address|dst-address|src-port|dst-port*; Default: **""**)	Selection of sub-stream identifiers
pcq-dst-address-mask (*integer [0..32]* | *IPNetmask*; Default: **32**)	size of IPv4 network that will be used as dst-address sub-stream identifier
pcq-dst-address6-mask (*integer [0..128]*; Default: **128**)	size of IPV6 network that will be used as dst-address sub-stream identifier
pcq-limit (*integer [1..4294967295]*; Default: **50**)	Queue size of a single sub-stream (in kilobytes)
pcq-rate (*integer [0..4294967295]*; Default: **0**)	Maximal available data rate of each sub-steam
pcq-src-address-mask (*integer [0..32]* | *IPNetmask*; Default: **32**)	size of IPv4 network that will be used as src-address sub-stream identifier
pcq-src-address6-mask (*integer [0..128]*; Default: **128**)	size of IPV6 network that will be used as src-address sub-stream identifier
pcq-total-limit (*integer [1..4294967295]*; Default: **2000**)	Max amount of bytes queued (in kilobytes) for all sub-streams per PCQ instance. Note that each queue tree entry has its own PCQ instance.
pfifo-limit (*integer [1..4294967295]*; Default: **50**)	Maximum number of packets that the PFIFO queue can hold. Applies if **kind** is pfifo.

`red-avg-packet` (*integer [1..65535]*; Default: **1000**)	Used by RED for average queue size calculations (for packet to byte translation)
`red-burst` (*integer [0..4294967295]*; Default: **20**)	Number of packets allowed for bursts of packets when there are no packets in the queue
`red-limit` (*integer [0..4294967295]*; Default: **60**)	RED queue limit in packets
`red-max-threshold` (*integer [0..4294967295]*; Default: **50**)	The average queue size at which packet marking probability is the highest.
`red-min-threshold` (*integer [0..4294967295]*; Default: **10**)	Average queue size in bytes.
`sfq-allot` (*integer [0..32767]*; Default: **1514**)	Amount of data in bytes that can be sent in one round-robin round
`sfq-perturb` (*integer [0..4294967295]*; Default: **5**)	How often hash function must be refreshed

Interface Queue

Sub-menu: `/queue interface`

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

Note: You cannot add new interfaces to this menu. List is generated automatically.

Properties

Property	Description
`interface` (*string*)	Interface name to which queue is applied. Read-only parameter.
`queue` (*string*; Default:)	Queue type assigned to particular interface.

[Top | Back to Content]