FireQOS Reference

Version 3.1.6 (Built 16 Feb 2021)

Contents

The latest version of this manual is available online as a PDF, as single page HTML and also as multiple pages within the website.

1 FireQOS Reference

1.1 Who should read this manual

This is a reference guide with specific detailed information on commands and configuration syntax for the FireQOS tool. The reference is unlikely to be suitable for newcomers to the tools, except as a means to look up more information on a particular command.

For tutorials and guides to using FireHOL and FireQOS, please visit the website.

1.2 Where to get help

The FireHOL website.

The mailing lists and archives.

The package comes with a complete set of manpages, a README and a brief INSTALL guide.

1.3 Installation

You can download tar-file releases by visiting the FireHOL website download area.

Unpack and change directory with:

```
tar xfz firehol-version.tar.gz
cd firehol-version
```

From version 3.0.0 it is no longer recommended to install firehol by copying files, since a function library is now used, in addition to the scripts.

Options for the configure program can be seen in the INSTALL file and by running:

```
./configure --help
```

To build and install taking the default options:

```
./configure && make && sudo make install
```

To not have files appear under /usr/local, try something like:

```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var
make
make install
```

If your O/S does not usually have a `/usr/libexec`, you may want to add `--libexecdir=/usr/lib` to the `configure`.

All of the common SysVInit command line arguments are recognised which makes it easy to deploy the script as a startup service.

Packages are available for most distributions and you can use your distribution's standard commands (e.g. aptitude, yum, etc.) to install these.

> **Note**
>
> Distributions do not always offer the latest version. You can see what the latest release is on the FireHOL website.

1.4 Licence

2 Running and Configuring FireQOS

- fireqos(1) - FireQOS program
- fireqos.conf(5) - FireQOS configuration file

3 Organising Traffic with FireQOS

- fireqos-interface(5) - create an interface definition
- fireqos-class(5) - traffic class definition
- fireqos-match(5) - QOS traffic match

4 Optional Parameters for FireQOS Commands

- fireqos-params-class(5) - optional class parameters
- fireqos-params-match(5) - optional match parameters

5 Manual Pages in Alphabetical Order

5.1 fireqos(1)

5.1.1 NAME

fireqos - an easy to use but powerful traffic shaping tool

5.1.2 SYNOPSIS

fireqos *CONFIGFILE* [start | debug] [– conf-arg ...]

fireqos { stop | clear_all_qos }

fireqos status [*name* [dump [*class*]]]

fireqos { dump | tcpdump } *name class* [*tcpdump-arg* ...]

fireqos { drops | overlimits | requeues } *name*

5.1.3 DESCRIPTION

FireQOS is a program that helps you configure traffic shaping on Linux.

Run without any arguments, `fireqos` will present some help on usage.

When given *CONFIGFILE*, `fireqos` will use the named file instead of `/etc/firehol/fireqos.conf` as its configuration.

The parameter *name* always refers to an interface name from the configuration file. The parameter *class* always refers to a named class within a named interface.

It is possible to pass arguments for use by the configuration file separating any conf-arg values from the rest of the arguments with `--`. The arguments are accessible in the configuration using standard bash(1) syntax e.g. $1, $2, etc.

5.1.4 COMMANDS

start; debug Activates traffic shaping on all interfaces, as given in the configuration file. When invoked as `debug`, FireQOS also prints all of the tc(8) commands it executes.

stop Removes all traffic shaping applied by FireQOS (it does not touch QoS on other interfaces and IFBs used by other tools).

clear_all_qos Removes all traffic shaping on all network interfaces and removes all IFB devices from the system, even those applied by other tools.

status Shows live utilisation for the specified interface. FireQOS will show you the rate of traffic on all classes, adding one line per second (similarly to vmstat, iostat, etc.)

If `dump` is specified, it tcpdumps the traffic in the given class of the interface.

tcpdump; dump FireQOS temporarily mirrors the traffic of any leaf class to an IFB device. Then it runs tcpdump(8) on this interface to dump the traffic to your console.

You may add any tcpdump(8) parameters you like to the command line, (to dump the traffic to a file, match a subset of the traffic, etc.), for example this:

```
fireqos tcpdump adsl-in voip -n
```

will start a tcpdump of all traffic on interface adsl-in, in class voip. The parameter `-n` is a tcpdump(8) parameter.

> **Note**
>
> When FireQOS is running in `tcpdump` mode, it locks itself and will refuse to run in parallel with another FireQOS altering the QoS, or tcpdumping other traffic. This is because FireQOS reserves device ifb0 for monitoring. If two FireQOS processes were allowed to `tcpdump` in parallel, your dumps would be wrong. So it locks itself to prevent such a case.

drops Shows packets dropped per second, per class, for the specified interface.

overlimits Shows packets delayed per second, per class, for the specified interface.

requeues Shows packets requeued per second, per class, for the specified interface.

5.1.5 FILES

`/etc/firehol/fireqos.conf`

5.1.6 SEE ALSO

- fireqos.conf(5) - FireQOS configuration file
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation
- tc(8) - show / manipulate traffic control settings
- tcpdump(8) - show / manipulate traffic control settings

5.2 fireqos.conf(5)

5.2.1 NAME

fireqos.conf - FireQOS configuration file

5.2.2 DESCRIPTION

This file defines the traffic shaping that will be applied by fireqos(1).

The default configuration file is `/etc/firehol/fireqos.conf`. It can be overridden from the command line.

A configuration consists of a number of input and output `interface` definitions (see fireqos-interface(5)). Each `interface` can define any number of (optionally nested) `class`es (see fireqos-class(5)) which shape the traffic which they `match` (see fireqos-match(5)).

5.2.3 SPEED UNITS

In FireQOS, speeds can be expressed in the following units:

#bps # bytes per second
#kbps; #Kbps # kilobytes per second
#mbps; #Mbps # megabytes per second
#gbps; #Gbps # gigabytes per second
#bit # bits per second
#kbit; #Kbit; # # kilobits per second (default)
#mbit; #Mbit # megabits per second
#gbit; #Gbit # gigabits per second
#% In a `class`, uses this percentage of the enclosing `rate`.

> **Note**
>
> The default, `kbit` is different to tc(8) which assumes bytes per second when no unit is specified.

5.2.4 EXAMPLE

This example uses match statements.

```
# incoming traffic from my ADSL router
interface eth2 adsl-in input rate 10500kbit adsl remote pppoe-llc
  class voip commit 100kbit pfifo
```

```
    match udp ports 5060,10000:10100 # asterisk sip and rtp
    match udp ports 16393:16402 # apple facetime

  class realtime commit 10%
    match tcp port 22,1195:1198,1753 # ssh, openvpn, pptp
    match udp port 53 # dns
    match proto GRE
    match icmp
    match tcp syn
    match tcp ack

  class clients commit 10%
    match tcp port 20,21,25,80,143,443,465,873,993 # mail, web, ftp, etc

# unmatched traffic goes here ('default' is a special name)
  class default max 90%

# I define torrents beneath the default class, so they slow
# down when the default class is willing to get bandwidth
  class torrents max 90%
    match port 51414 # my torrent client

# outgoing traffic to my ADSL router
interface eth2 adsl-out output rate 800kbit adsl remote pppoe-llc
  class voip commit 100kbit pfifo
    match udp ports 5060,10000:10100 # asterisk sip and rtp
    match udp ports 16393:16402 # apple facetime

  class realtime commit 10%
    match tcp port 22,1195:1198,1753 # ssh, openvpn, pptp
    match udp port 53 # dns
    match proto GRE
    match icmp
    match tcp syn
    match tcp ack

  class clients commit 10%
    match tcp port 20,21,25,80,143,443,465,873,993 # mail, web, ftp, etc

# unmatched traffic goes here ('default' is a special name)
  class default max 90%

# I define torrents beneath the default class, so they slow
# down when the default class is willing to get bandwidth
  class torrents max 90%
    match port 51414 # my torrent client
```

This example uses server/client statements in a bidirectional interface. Of course match statements can also be specified. FireQOS will create 2 interfaces out of this: world-in and world-out.

```
DEVICE=dsl0
INPUT_SPEED="12000kbit"
OUTPUT_SPEED="800kbit"
LINKTYPE="adsl local pppoe-llc"

# a few service definitions
# all the rest that are used in this example
# are defined by FireQOS
server_netdata_ports="tcp/19999"
server_rtp_ports="udp/10000:10100"
server_openvpn_ports="any/1195:1198"
server_mytorrent_ports="any/60000"
server_mytorrenttransfers_ports="any/60001:64999"
server_myssh_ports="tcp/2222"

# League Of Legends game (yes! I have kids)
server_lol_ports="udp/5000:5500 tcp/8393:8400,2099,5223,5222,8088"

interface $DEVICE world bidirectional $LINKTYPE input rate $INPUT_SPEED output rate $OUTPU

  class voip commit 100kbit pfifo
    server sip
    client sip
    server rtp
    client stun

  class interactive input commit 20% output commit 10%
    server icmp limit 50%

    server dns
    client dns

    server ssh
    client ssh

    server myssh
    client myssh

    client teamviewer
    client lol

  class chat input commit 1000kbit output commit 30%
```

```
  client facetime

  server hangouts
  client hangouts

  client gtalk
  client jabber

class vpns input commit 20% output commit 30%
  server pptp
  server GRE
  server openvpn

class servers
  server netdata
  server http

# a class group to favor tcp handshake over transfers
class group surfing prio keep commit 5%
  client surfing
  client rsync

  class synacks
    match tcp syn
    match tcp ack

class group end

class synacks commit 5%
  match tcp syn
  match tcp ack

class default

class background commit 4%
  client torrents
  server mytorrent
  server mytorrenttransfers
```

5.2.5 SEE ALSO

- fireqos(1) - FireQOS program
- fireqos-interface(5) - QOS interface definition
- fireqos-class(5) - QOS class definition

- fireqos-match(5) - QOS traffic match
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation
- tc(8) - show / manipulate traffic control settings

5.3 fireqos-class(5)

5.3.1 NAME

fireqos-class - traffic class definition

5.3.2 SYNOPSIS

{class|class4|class6|class46} [group] *name* [*optional-class-params*]

{class|class4|class6|class46} group end

5.3.3 DESCRIPTION

There is also an optional `match` parameter called `class`; see fireqos-params-match(5).

Writing `class` inherits the IPv4/IPv6 version from its enclosing interface (see fireqos-interface(5)).

Writing `class4` includes only IPv4 traffic in the class.

Writing `class6` includes only IPv6 traffic in the class.

Writing `class46` includes both IPv4 and IPv6 traffic in the class.

The actual traffic to be matched by a class is defined by adding matches. See fireqos-match(5).

The sequence that classes appear in the configuration defines their priority. The first class is the most important one. Unless otherwise limited it will get all available bandwidth if it needs to.

The second class is less important than the first, the third is even less important than the second, etc. The idea is very simple: just put the classes in the order of importance to you.

Classes can have their priority assigned explicitly with the `prio` parameter. See fireqos-params-class(5).

> **Note**
>
> The underlying Linux qdisc used by FireQOS, HTB, supports only 8 priorities, from 0 to 7. If you use more than 8 priorities, all after the 8th will get the same priority (`prio` 7).

All classes in FireQOS share the `interface` bandwidth. However, every class has a *committed* rate (the minimum guaranteed speed it will get if it needs to)

and a *ceiling* (the maximum rate this class can reach, provided there is capacity available and even if there is spare).

Classes may be nested to any level by using the `class group` syntax.

By default FireQOS creates nested classes as *classes directly attached to their parent class*. This way, nesting does not add any delays.

FireQOS can also *emulate new hardware* at the `group class` level. This may be needed, when for example you have an ADSL router that you connect to via Ethernet: you want the LAN traffic to be at Ethernet speed, but WAN traffic at ADSL speed with proper ADSL overheads calculation.

To accomplish hardware emulation nesting, you add a `linklayer` definition (`ethernet`, `adsl`, `atm`, etc.), or just an `mtu` to the `group class`. FireQOS will create a qdisc within the class, where the linklayer parameters will be assigned and the child classes will be attached to this qdisc. This adds some delay to the packets of the child classes, but allows you to emulate new hardware. For linklayer options, see fireqos-params-class(5).

There is special class, called `default`. Default classes can be given explicitly in the configuration file. If they are not found in the config, FireQOS will append one at the end of each `interface` or `class group`.

5.3.4 PARAMETERS

`group` It is possible to nest classes by using a group. Grouped classes must be closed with the `class group end` command. Class groups can be nested.

name This is a single-word name for this class and is used for displaying status information.

optional-class-params The set of optional class parameters to apply to this class.

The following optional class parameters are inherited from the `interface` the class is in:

- `ceil`
- `burst`
- `cburst`
- `quantum`
- `qdisc`

If you define one of these at the interface level, then all classes within the interface will get the value by default. These values can be overwritten by defining the parameter on the class too. The same inheritance works on class groups.

Optional class parameters not in the above list are *not* inherited from interfaces.

FireQOS will by default `commit` 1/100th of the parent bandwidth to each class. This can be overwritten per class by adding a `commit` to the class, or adding a `minrate` at the parent.

5.3.5 EXAMPLES

To create a nested class, called servers, containing http and smtp:

```
interface eth0 lan input rate 1Gbit
  class voip commit 1Mbit
    match udp ports 5060,10000:10100

  class group servers commit 50%  # define the parent class
    match tcp                     # apply to all child classes

    class mail commit 50%         # 50% of parent ('servers')
      match port 25               # matches within parent ('servers')

    class web commit 50%
      match port 80
  class group end                 # end the group 'servers'

  class streaming commit 30%
```

To create a nested class which emulates an ADSL modem:

```
interface eth0 lan output rate 1Gbit ethernet
  class lan
    match src 192.168.0.0/24 # LAN traffic

  class group adsl rate 10Mbit ceil 10Mbit adsl remote pppoe-llc
    match all # all non-lan traffic in this emulated hardware group

    class voip # class within adsl
      match udp port 5060

    class web # class within adsl
      match tcp port 80,443
  class group end
```

5.3.6 SEE ALSO

- fireqos-params-class(5) - QOS class parameters
- fireqos(1) - FireQOS program
- fireqos.conf(5) - FireQOS configuration file
- fireqos-interface(5) - QOS interface definition
- fireqos-match(5) - QOS traffic match
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation

5.4 fireqos-interface(5)

5.4.1 NAME

fireqos-interface - create an interface definition

5.4.2 SYNOPSIS

{ interface | interface4 } *device name direction* [*optional-class-params*] { rate | commit | min } *speed*

interface46 ...

interface6 ...

5.4.3 DESCRIPTION

Writing `interface` or `interface4` applies traffic shaping rules only to IPv4 traffic.

Writing `interface6` applies traffic shaping rules only to IPv6 traffic.

Writing `interface46` applies traffic shaping rules to both IPv4 and IPv6 traffic.

The actual traffic shaping behaviour of a class is defined by adding classes. See fireqos-class(5).

> **Note**
>
> To achieve best results with `incoming` traffic shaping, you should not use 100% of the available bandwidth at the interface level.
>
> If you use all there is, at 100% utilisation of the link, the neighbour routers will start queuing packets. This will destroy prioritisation. Try 85% or 90% instead.

5.4.4 PARAMETERS

device This is the interface name as shown by `ip link show` (e.g. eth0, ppp1, etc.)

name This is a single-word name for this interface and is used for retrieving status information later.

direction If set to `input`, traffic coming in to the interface is shaped.

If set to `output`, traffic going out via the interface is shaped.

if set to `bidirectional` traffic for both input and output can be shaped. If you need to differentiate input and output parameters per statements within the interface, you can prefix them with `input` or `output` like this:

```
interface eth0 lan bidirectional ...
    class voip input commit 1Mbit output commit 2Mbit ...
```

optional-class-params For a list of optional class parameters which can be applied to an interface, see fireqos-params-class(5).

speed For an interface, the committed *speed* must be specified with the `rate` option. The speed can be expressed in any of the units described in fireqos.conf(5).

5.4.5 EXAMPLES

To create an input policy on eth0, capable of delivering up to 1Gbit of traffic:

```
interface eth0 lan-in input rate 1Gbit
```

5.4.6 SEE ALSO

- fireqos.conf(5) - FireQOS configuration file
- fireqos-class(5) - QOS class definition
- fireqos-params-class(5) - QOS class parameters
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation

5.5 fireqos-match(5)

5.5.1 NAME

fireqos-match - QOS traffic match

5.5.2 SYNOPSIS

{match|match4|match6|match46} *optional-match-params*

5.5.3 DESCRIPTION

Writing `match` inherits the IPv4/IPv6 version from its enclosing class (see fireqos-class(5)).

Writing `match4` includes only IPv4 traffic in the match.

Writing `match6` includes only IPv6 traffic in the match.

Writing `match46` includes both IPv4 and IPv6 traffic in the match.

You can add as many `match` statements as you like to a FireQOS configuration. They assign traffic to a class: by default to the class after which they are declared.

The sequence that matches appear in the configuration defines their priority, with the first match being given a `prio` of 10, with 10 added for each subsequent match (10, 20, 30, ...).

Matches can have their priority assigned explicitly with the `prio` parameter. See fireqos-params-match(5).

If one `match` statement generates multiple tc(8) `filter` statements, all filters generated by the same `match` statement will have the same `prio`.

> **Note**
>
> `match` rules are attached to the parent of the `class` they appear in. Within the configuration they are written under a class, but in reality they are attached to their class parent, so that they classify the parent's traffic that they match, into the class.

It is also possible to group all `match` statements together below the classes. This allows them to be arranged in preferred order, without the need for any explicit `prio` parameters. In this case however, each match statement must specify to which class it classifies the packets it matches, using the `class` parameter. See fireqos-params-match(5) and the examples below.

You can also write `client` and `server` statements, much like FireHOL allows, with the same service definitions. For FireQOS however, the client ports are ignored. `server` statements match the server ports on this linux side, while `client` statements match the server ports on the remote side.

Example:

```
server_myrtp_ports="10000:10100"

interface eth0 lan bidirectional rate 1Gbit
  class voip
    server sip
    client sip

    server myrtp

  class dns
    server dns

  class mail
    server smtp
```

5.5.4 PARAMETERS

optional-match-params The set of optional parameters which describe this match. See fireqos-params-match(5).

5.5.5 EXAMPLES

Match traffic within classes:

```
interface eth0 lan output rate 1Gbit
  class voip
    match udp ports 5060,10000:10100
  class dns
    match udp port 53
  class mail
    match tcp port 25
```

Matches split out and explicitly assigning traffic to classes (N.B. without the `class` parameters, all traffic would be classified into 'mail'):

```
interface eth0 lan output rate 1Gbit
  class voip
  class dns
  class mail
```

```
match udp ports 5060,10000:10100 class voip
match tcp port 25 class mail
match tcp port 80 class web
```

5.5.6 SEE ALSO

- fireqos-params-match(5) - QOS match parameters
- fireqos(1) - FireQOS program
- fireqos.conf(5) - FireQOS configuration file
- fireqos-interface(5) - QOS interface definition
- fireqos-class(5) - QOS class definition
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation
- tc(8) - show / manipulate traffic control settings

5.6 fireqos-params(5)

5.6.1 NAME

fireqos-params - shared class/match parameters

5.6.2 SYNOPSIS

prio

priority

5.6.3 DESCRIPTION

Some optional parameter names are the same for both `class` and `match`. This page exists as a placeholder to help you find the appropriate documentation.

If you are searching for FireQOS parameters in general, see both fireqos-params-class(5) and/or fireqos-params-match(5) depending upon your need.

prio For the `class` version, see fireqos-params-class(5).

For the `match` version, see fireqos-params-match(5).

priority For the `class` version, see fireqos-params-match(5).

For the `match` version, see fireqos-params-class(5).

5.6.4 SEE ALSO

- fireqos-params-class(5) - QOS class parameters
- fireqos-params-match(5) - QOS match parameters
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation

5.7 fireqos-params-class(5)

5.7.1 NAME

fireqos-params-class - optional class parameters

5.7.2 SYNOPSIS

rate | commit | min *speed*

ceil | max *speed*

minrate *speed*

{ qdisc *qdisc-name* | pfifo|bfifo|sfq|fq_codel|codel|none } [options "*qdisc-options*"]

prio { 0..7 | keep | last }

{ linklayer *linklayer-name* } | { adsl {local|remote} *encapsulation* } | ethernet | atm

mtu *bytes*

mpu *bytes*

tsize *size*

overhead *bytes*

r2q *factor*

burst *bytes*

cburst *bytes*

quantum *bytes*

priority | balanced

input | output

5.7.3 DESCRIPTION

All of the options apply to `interface` and `class` statements.

Units for speeds are defined in fireqos.conf(5).

5.7.3.1 input, output

For `bidirectional` interfaces, `input` and `output` define the direction for which the parameters following it are applied.

Only the following parameters are affected (all the others are applied to both input and output):

- `minrate`
- `rate`, `min`, `commit`
- `ceil`, `max`

If one of the above is not defined for either `input` or `output`, its default will be used.

5.7.3.2 rate, commit, min

When a committed rate of *speed* is provided to a class, it means that the bandwidth will be given to the class when it needs it. If the class does not need the bandwidth, it will be available for any other class to use.

For interfaces, a rate must be defined.

For classes the rate defaults to 1/100 of the interface capacity.

5.7.4 ceil, max

Defines the maximum *speed* a class can use. Even there is available bandwidth, a class will not exceed its ceil speed.

For interfaces, the default is the `rate` speed of the interface.

For classes, the defaults is the `ceil` of the their interfaces.

5.7.4.1 minrate

Defines the default committed *speed* for all classes not specifically given a rate in the config file. It forces a recalculation of tc(8) r2q.

When minrate is not given, FireQOS assigns a default value of 1/100 of the interface `rate`.

5.7.4.2 qdisc *qdisc-name*, pfifo, bfifo, sfq, fq_codel, codel, none

The qdisc defines the method to distribute class bandwidth to its sockets. It is applied within the class itself and is useful in cases where a class gets saturated. For information about these, see the Traffic Control Howto

A qdisc is only useful when applied to a class. It can be specified at the interface level in order to set the default for all of the included classes.

To pass options to a qdisc, you can specify them through an environment variable or explicitly on each class.

Set the variable FIREQOS_DEFAULT_QDISC_OPTIONS_qdiscname in the config file. For example, for sfq:

`FIREQOS_DEFAULT_QDISC_OPTIONS_sfq="perturb 10 quantum 2000".`

Using this variable each sfq will get these options by default. You can still override this by specifying explicit `options` for individual qdiscs, for example to add some `sfq` options you would write:

```
class classname sfq options "perturb 10 quantum 2000"
```

The `options` keyword must appear just after the qdisc name.

5.7.4.3 prio (class)

> *Note*
>
> There is also a match parameter called `prio`, see fireqos-params-match(5).

HTB supports 8 priorities, from 0 to 7. Any number less than 0 will give priority 0. Any number above 7 will give priority 7.

By default, FireQOS gives the first class priority 0, and increases this number by 1 for each class it encounters in the config file. If there are more than 8 classes, all classes after the 8th will get priority 7. In `balanced` mode (see balanced, below), all classes will get priority 4 by default.

FireQOS restarts priorities for each interface and class group.

The class priority defines how the spare bandwidth is spread among the classes. Classes with higher priorities (lower `prio`) will get all spare bandwidth. Classes with the same priority will get a percentage of the spare bandwidth, proportional to their committed rates.

The keywords `keep` and `last` will make a class use the priority of the class just above / before it. So to make two consecutive classes have the same prio, just add `prio keep` to the second one.

5.7.4.4 linklayer *linklayer-name*, ethernet, atm

The `linklayer` can only be given on interfaces. It is used by the kernel to calculate the overheads in the packets.

5.7.4.5 adsl

`adsl` is a special `linklayer` that automatically calculates ATM overheads for the link.

`local` is used when linux is running PPPoE.

`remote` is used when PPPoE is running on the router.

> **Note**
>
> This special case has not yet been demonstrated for sure. Experiment a bit and if you find out, let us know to update this page. In practice, this parameter lets the kernel know that the packets it sees, have already an ethernet header on them.

encapsulation can be one of (all the labels on the same line are aliases):

- IPoA-VC/Mux or ipoa-vcmux or ipoa-vc or ipoa-mux,
- IPoA-LLC/SNAP or ipoa-llcsnap or ipoa-llc or ipoa-snap
- Bridged-VC/Mux or bridged-vcmux or bridged-vc or bridged-mux
- Bridged-LLC/SNAP or bridged-llcsnap or bridged-llc or bridged-snap
- PPPoA-VC/Mux or pppoa-vcmux or pppoa-vc or pppoa-mux
- PPPoA-LLC/SNAP or pppoa-llcsnap or pppoa-llc or pppoa-snap
- PPPoE-VC/Mux or pppoe-vcmux or pppoe-vc or pppoe-mux
- PPPoE-LLC/SNAP or pppoe-llcsnap or pppoe-llc or pppoe-snap

If your adsl router can give you the mtu, it would be nice to add an `mtu` parameter too. For detailed info, see here.

5.7.4.6 mtu

Defines the MTU of the interface in *bytes*.

FireQOS will query the interface to find its MTU. You can overwrite this behaviour by giving this parameter to a class or interface.

5.7.4.7 mpu

Defines the MPU of the interface in *bytes*.

FireQOS does not set a default value. You can set your own using this parameter.

5.7.4.8 tsize

FireQOS does not set a default *size*. You can set your own using this parameter.

5.7.4.9 overhead

FireQOS automatically calculates the *bytes* `overhead` for ADSL. For all other technologies, you can specify the overhead in the config file.

5.7.4.10 r2q

FireQOS calculates the proper r2q *factor*, so that you can control speeds in steps of 1/100th of the interface speed (if that is possible).

Note

The HTB manual states that this parameter is ignored when a quantum have been set. By default, FireQOS sets quantum to interface MTU, so `r2q` is probably is ignored by the kernel.

5.7.4.11 burst

`burst` specifies the number of *bytes* that will be sent at once, at ceiling speed, when a class is allowed to send traffic. It is like a 'traffic unit'. A class is allowed to send at least `burst` bytes before trying to serve any other class.

`burst` should never be lower that the interface mtu and class groups and interfaces should never have a smaller `burst` value than their children. If you do specify a higher `burst` for a child class, its parent may get stuck sometimes (the child will drain the parent).

By default, FireQOS lets the kernel decide this parameter, which calculates the lowest possible value (the minimum value depends on the rate of the interface and the clock speed of the CPU).

`burst` is inherited from interfaces to classes and from group classes to their subclasses. FireQOS will not allow you to set a `burst` at a subclass, higher than its parent. Setting a `burst` of a subclass higher than its parent will drain the parent class, which may be stuck for up to a minute when this happens. For this check to work, FireQOS uses just its configuration (it does not query the kernel to check how the value specified in the config file for a subclass relates to the actual value of its parent).

5.7.4.12 cburst

`cburst` is like `burst`, but at hardware speed (not just ceiling speed).

By default, FireQOS lets the kernel decide this parameter.

`cburst` is inherited from interfaces to classes and from group classes to their subclasses. FireQOS will not allow you to set a `cburst` at a subclass, higher to its parent. Setting a `cburst` of a subclass higher than its parent, will drain the parent class, which may be stuck for up to a minute when this happens. For this check to work, FireQOS uses just its configuration (it does not query the kernel to check how the value specified in the config file for a subclass relates to the actual value of its parent).

5.7.4.13 quantum

`quantum` specifies the number of *bytes* a class is allowed to send at once, when it is borrowing spare bandwidth from other classes.

By default, FireQOS sets `quantum` to the interface mtu.

`quantum` is inherited from interfaces to classes and from group classes to their subclasses.

5.7.4.14 priority, balanced

These parameters set the priority mode of the child classes.

`priority priority` is the default mode, where FireQOS assigns an incremental priority to each class. In this mode, the first class takes `prio 0`, the second `prio 1`, etc. When a class has a higher prio than the others (higher = smaller number), this high priority class will get all the spare bandwidth available, when it needs it. Spare bandwidth will be allocate to lower priority classes only when the higher priority ones do not need it.

`balanced balanced` mode gives `prio 4` to all child classes. When multiple classes have the same `prio`, the spare bandwidth available is spread among them, proportionally to their committed rate. The value 4 can be overwritten by setting FIREQOS_BALANCED_PRIO at the top of the config file to the `prio` you want the balanced mode to assign for all classes.

The priority mode can be set in interfaces and class groups. The effect is the same. The classes that are defined as child classes, will get by default the calculated class `prio` based on the priority mode given.

These options affect only the default `prio` that will be assigned by FireQOS. The default is used only if you don't explicitly use a `prio` parameter on a class.

Note

There is also a match parameter called `priority`, see fireqos-params-match(5).

5.7.5 SEE ALSO

- fireqos(1) - FireQOS program
- fireqos.conf(5) - FireQOS configuration file
- fireqos-interface(5) - QOS interface definition
- fireqos-class(5) - QOS class definition
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation

5.8 fireqos-params-match(5)

5.8.1 NAME

fireqos-params-match - optional match parameters

5.8.2 SYNOPSIS

at { root | *name* }

class *name*

syn|syns

ack|acks

{ proto|protocol *protocol* [,*protocol*...] } |tcp|udp|icmp|gre|ipv6

{ tos | priority } *tosid* [,*tosid*...]

{ DSCP } *classname* [,*classname*...]

mark *mark* [,*mark*...]

connmark *mark* [,*mark*...]

rawmark *mark* [,*mark*...]

custommark *name mark* [,*mark*...]

{ port | ports } *port*[:*range*] [,*port*[:*range*]...]

{ sport | sports } *port*[:*range*] [,*port*[:*range*]...]

{ dport | dports } *port*[:*range*] [,*port*[:*range*]...]

{ ip | net | host } *net* [,*net*...]

src *net* [,*net*...]

dst *net* [,*net*...]

{ srcmac | smac } *mac*

{ dstmac | dmac } *mac*

prio *id*

input

output

custom '*custom tc parameters*'

estimator *interval decay*

police *police*

insidegre

5.8.3 DESCRIPTION

These options apply to `match` statements.

5.8.3.1 input, output

On `bidirectional` interfaces, `input` and `output` will check the current direction of the interface. If the match is `input` but the interface is `output` the match will be reversed. The same will happen if `output` is given at the match and the interface is `input`.

The parameters that are reversed are:

- `src` and `dst`
- `sport` and `dport`
- `srcmac` and `dstmac`

This allows a definition like this:

```
interface dsl0 world bidirectional ...
  class surfing ...
    match input sport 0:1023
```

The above will match `sport 0:1023` at the `input` interface, and will automatically reverse it to match `dport 0:1023` at the `output` interface.

5.8.3.2 at

By default a `match` is attached to the parent of its parent class. For example, if its parent is a class directly under the interface, then the `match` is attached to the interface and is compared against all traffic of the interface. For nested classes, a `match` of a leaf, is attached to the parent class and is compared against all traffic of this parent class.

With the `at` parameter, a `match` can be attached any class. The *name* parameter should be a class name. The name `root` attaches the `match` to the interface.

5.8.3.3 class

Defines the *name* of the class that will get the packets matched by this `match`.

By default it is the name of the class the `match` statement appears under.

> *Note*
>
> There is also a `class` definition for traffic, see fireqos-class(5).

5.8.3.4 syn, syns

Match TCP SYN packets. Note that the `tcp` parameter must be specified.

If the same match statement includes more protocols than TCP, then this match will work for the TCP packets (it will be silently ignored for all other protocols).

For example, syn is ignored when generating the UDP filter in the below:

```
match tcp syn
match proto tcp,udp syn
```

5.8.3.5 ack, acks

Same as `syn`, but matching small TCP packets with the ACK bit set.

5.8.3.6 proto, protocol, tcp, udp, icmp, gre, ipv6

Match the *protocol* in the IP header.

5.8.3.7 tos, priority

Match to TOS field of ipv4 or the priority field of ipv6. The *tosid* can be a value/mask in any format tc(8) accepts, or one of the following:

- min-delay, minimize-delay, minimum-delay, low-delay, interactive
- maximize-throughput, maximum-throughput, max-throughput, high-throughput, bulk
- maximize-reliability, maximum-reliability, max-reliability, reliable
- min-cost, minimize-cost, minimum-cost, low-cost, cheap, normal-service, normal

  Note

  There is also a class parameter called `priority`, see fireqos-params-class(5).

5.8.3.8 dscp

Match to DSCP value in IP TOS header field. The *classname* has to be one of the following values:

- CS1, CS2, CS3, CS4, CS5, CS6, CS7
- AF11, AF12, AF13
- AF21, AF22, AF23
- AF31, AF32, AF33
- AF41, AF42, AF43
- EF

  Note

tc-filter only supports ToS parameters. That is why a lookaside table is configured within fireqos code to translate the DSCP value to their matching TOS value. See RFC2474 for more information.

5.8.3.9 mark, connmark, custommark, rawmark

Match an iptables(8) MARK. This works the same way it works for FireHOL. FireHOL and FireQOS share the same marks and their masks.

Matching iptables(8) MARKs do not work on input interfaces. You can use them only on output. The IFB devices that are used for shaping inbound traffic do not have any iptables hooks to allow matching MARKs. If you try it, FireQOS will attempt to do it, but currently you will get an error from the tc(8) command executed or they will be silently ignored by it.

On some Linux distributions (e.g. OpenWRT) there is a module called `act_commark` that will enable this feature. Set this within your `fireqos.conf` to enable it:

```
FIREQOS_CONNMARK_RESTORE="act_connmark"
```

Also note that matching marks requires a suitably configured kernel (with `CONFIG_CLS_U32_MARK=y`). There is no error if the kernel is not configured correctly; it just silently drops the rules. For details see this error report.

5.8.3.10 ports, sports, dports

Match ports of the IP header. `ports` will create rules for matching source and destination ports (separate rules for each). `dports` matches destination ports, `sports` matches source ports.

5.8.3.11 ip, net, host, src, dst

Match IPs of the IP header. `ip`, `net` and `host` will create rules for matching source and destination IPs (separate rules for each). `src` matches source IPs and `dst` destination IPs.

Note

If the class these matches appear in are IPv4, then only IPv4 IPs can be used. To override use `match6 ... src/dst *IPV6_IP*`

Similarly, if the class is IPv6, then only IPv6 IPs can be used. To override use `match4 ... src/dst *IPV4_IP*`.

You can mix IPv4 and IPv6 in any way you like. FireQOS supports inheritance, to figure out for each statement which is the default. For example:

```
interface46 eth0 lan output rate 1Gbit # ipv4 and ipv6 enabled
  class voip # ipv4 and ipv6 class, as interface is both
    match udp port 53 # ipv4 and ipv6 rule, as class is both
```

```
    match4 src 192.0.2.1 # ipv4 only rule
    match6 src 2001:db8::1 # ipv6 only rule

  class4 realtime # ipv4 only class
    match src 198.51.100.1 # ipv4 only rule, as class is ipv4-only

  class6 servers # ipv6 only class
        match src 2001:db8::2 # ipv6 only rule, as class is ipv6-only
```

To convert an IPv4 interface to IPv6, just replace `interface` with `interface6`. All the rules in that interface, will automatically inherit the new protocol. Of course, if you use IP addresses for matching packets, make sure they are IPv6 IPs too.

5.8.3.12 prio (match)

> *Note*
>
> There is also a class parameter called `prio`, see fireqos-params-class(5).

All match statements are attached to the interface. They forward traffic to their class, but they are actually executed for all packets that are leaving the interface (note: input matches are actually output matches on an IFB device).

By default, the priority they are executed, is the priority they appear in the configuration file, i.e. the first match of the first class is executed first, then the rest matches of the first class in the sequence they appear, then the matches of the second class, etc.

It is sometimes necessary to control the order of matches. For example, when you want host 192.0.2.1 to be assigned the first class, except port tcp/1234 which should be assigned the second class. The following will *not* work:

```
interface eth0 lan output rate 1Gbit
  class high
    match host 192.0.2.1

  class low
    match host 192.0.2.1 port 1234 # Will never match
```

In this case, the first match is assigned priority 10 and the second priority 20. The second match will never match anything, since all traffic for the host is already matched by the first one.

Setting an explicit priority allows you to change the order in which the matches are executed. FireQOS gives priority 10 to the first match of every interface, 20 to the second match, 30 to the third match, etc. So the default is 10 x the sequence number. You can set `prio` to overwrite this number.

To force executing the second match before the first, just set a lower priority for it. For example, this will cause the desired behaviour:

```
interface eth0 lan output rate 1Gbit
  class high
    match host 192.0.2.1

  class low
    match host 192.0.2.1 port 1234 prio 1 # Matches before host-only
```

5.8.3.13 insidegre

By specifying keyword `insidegre` a GRE (Generic Routing Encapsulation) packet can be matched on the encapsulated IP packet header information.

`insidegre` is available for the following matches:

- src
- dst
- protocol
- port
- tos
- dscp

```
interface eth0 world ...
  class surfing commit 128kbit ceil 1024kbit prio 7
    match src 10.1.128.230 dst 8.8.8.8 insidegre
    match protocol ospf insidegre
    match port 25 insidegre
    match tos 3 insidegre
    match dscp ef insidegre
```

5.8.4 SEE ALSO

- fireqos(1) - FireQOS program
- fireqos.conf(5) - FireQOS configuration file
- fireqos-match(5) - QOS traffic match
- FireHOL Website
- FireQOS Online PDF Manual
- FireQOS Online Documentation