NAME
sfq − Stochastic Fairness Queueing

SYNOPSIS
tc qdisc ... [**divisor** hashtablesize] [**limit** packets] [**perturb** seconds] [**quantum** bytes] [
flows number] [**depth** number] [**headdrop**] [**redflowlimit** bytes] [**min** bytes] [**max**
bytes] [**avpkt** bytes] [**burst** packets] [**probability** P] [**ecn**] [**harddrop**]

DESCRIPTION
Stochastic Fairness Queueing is a classless queueing discipline available for traffic control with the
tc(8) command.

SFQ does not shape traffic but only schedules the transmission of packets, based on 'flows'. The
goal is to ensure fairness so that each flow is able to send data in turn, thus preventing any single
flow from drowning out the rest.

This may in fact have some effect in mitigating a Denial of Service attempt.

SFQ is work-conserving and therefore always delivers a packet if it has one available.

ALGORITHM
On enqueueing, each packet is assigned to a hash bucket, based on the packets hash value. This
hash value is either obtained from an external flow classifier (use **tc filter** to set them), or a
default internal classifier if no external classifier has been configured.

When the internal classifier is used, sfq uses

(i)   Source address

(ii)   Destination address

(iii)   Source and Destination port

If these are available. SFQ knows about ipv4 and ipv6 and also UDP, TCP and ESP. Packets
with other protocols are hashed based on the 32bits representation of their destination and source.
A flow corresponds mostly to a TCP/IP connection.

Each of these buckets should represent a unique flow. Because multiple flows may get hashed to
the same bucket, sfqs internal hashing algorithm may be perturbed at configurable intervals so
that the unfairness lasts only for a short while. Perturbation may however cause some inadvertent
packet reordering to occur. After linux-3.3, there is no packet reordering problem, but possible
packet drops if rehashing hits one limit (number of flows or packets per flow)

When dequeuing, each hashbucket with data is queried in a round robin fashion.

Before linux-3.3, the compile time maximum length of the SFQ is 128 packets, which can be
spread over at most 128 buckets of 1024 available. In case of overflow, tail-drop is performed on
the fullest bucket, thus maintaining fairness.

After linux-3.3, maximum length of SFQ is 65535 packets, and divisor limit is 65536. In case of
overflow, tail-drop is performed on the fullest bucket, unless headdrop was requested.

PARAMETERS
divisor   Can be used to set a different hash table size, available from kernel 2.6.39 onwards. The
           specified divisor must be a power of two and cannot be larger than 65536. Default value:
           1024.

limit   Upper limit of the SFQ. Can be used to reduce the default length of 127 packets. After
           linux-3.3, it can be raised.

depth   Limit of packets per flow (after linux-3.3). Default to 127 and can be lowered.

perturb
>   Interval in seconds for queue algorithm perturbation. Defaults to 0, which means that no
>   perturbation occurs. Do not set too low for each perturbation may cause some packet
>   reordering or losses. Advised value: 60 This value has no effect when external flow classifi-
>   cation is used. Its better to increase divisor value to lower risk of hash collisions.

quantum
>   Amount of bytes a flow is allowed to dequeue during a round of the round robin process.
>   Defaults to the MTU of the interface which is also the advised value and the minimum
>   value.

flows   After linux-3.3, it is possible to change the default limit of flows. Default value is 127

headdrop
>   Default SFQ behavior is to perform tail-drop of packets from a flow. You can ask a head-
>   drop instead, as this is known to provide a better feedback for TCP flows.

redflowlimit
>   Configure the optional RED module on top of each SFQ flow. Random Early Detection
>   principle is to perform packet marks or drops in a probabilistic way. (man tc-red for
>   details about RED)
>   redflowlimit configures the hard limit on the real (not average) queue size per SFQ flow in bytes.

min   Average queue size at which marking becomes a possibility. Defaults to **max** /3

max   At this average queue size, the marking probability is maximal. Defaults to **redflowlimit**
>   /4

probability
>   Maximum probability for marking, specified as a floating point number from 0.0 to 1.0.
>   Default value is 0.02

avpkt   Specified in bytes. Used with burst to determine the time constant for average queue size
>   calculations. Default value is 1000

burst   Used for determining how fast the average queue size is influenced by the real queue size.
>   Default value is :
>   **(2 * min + max) / (3 * avpkt)**

ecn   RED can either 'mark' or 'drop'. Explicit Congestion Notification allows RED to notify
>   remote hosts that their rate exceeds the amount of bandwidth available. Non-ECN capa-
>   ble hosts can only be notified by dropping a packet. If this parameter is specified, packets
>   which indicate that their hosts honor ECN will only be marked and not dropped, unless
>   the queue size hits **depth** packets.

harddrop
>   If average flow queue size is above **max** bytes, this parameter forces a drop instead of ecn
>   marking.

EXAMPLE & USAGE

To attach to device ppp0:

tc qdisc add dev ppp0 root sfq

Please note that SFQ, like all non-shaping (work-conserving) qdiscs, is only useful if it owns the
queue. This is the case when the link speed equals the actually available bandwidth. This holds
for regular phone modems, ISDN connections and direct non-switched ethernet links.

Most often, cable modems and DSL devices do not fall into this category. The same holds for
when connected to a switch and trying to send data to a congested segment also connected to the
switch.

In this case, the effective queue does not reside within Linux and is therefore not available for

scheduling.

Embed SFQ in a classful qdisc to make sure it owns the queue.

It is possible to use external classifiers with sfq, for example to hash traffic based only on source/destination ip addresses:

tc filter add ... flow hash keys src,dst perturb 30 divisor 1024

Note that the given divisor should match the one used by sfq. If you have changed the sfq default of 1024, use the same value for the flow hash filter, too.

Example of sfq with optional RED mode :

tc qdisc add dev eth0 parent 1:1 handle 10: sfq limit 3000 flows 512 divisor 16384
  redflowlimit 100000 min 8000 max 60000 probability 0.20 ecn headdrop

SOURCE
      o   Paul E. McKenney "Stochastic Fairness Queuing", IEEE INFOCOMM'90 Proceedings, San Francisco, 1990.

      o   Paul E. McKenney "Stochastic Fairness Queuing", "Interworking: Research and Experience", v.2, 1991, p.113-131.

      o   See also: M. Shreedhar and George Varghese "Efficient Fair Queuing using Deficit Round Robin", Proc. SIGCOMM 95.

SEE ALSO
tc(8), **tc-red**(8)

AUTHORS
Alexey N. Kuznetsov, <kuznet@ms2.inr.ac.ru>, Eric Dumazet <eric.dumazet@gmail.com>.

This manpage maintained by bert hubert <ahu@ds9a.nl>