NAME

tc−stab − Generic size table manipulations

SYNOPSIS

tc qdisc add ... stab
[**mtu** BYTES] [**tsize** SLOTS]
[**mpu** BYTES] [**overhead** BYTES]
[**linklayer** { adsl | atm | ethernet }] ...

OPTIONS

For the description of BYTES − please refer to the **UNITS** section of **tc**(8).

mtu
maximum packet size we create size table for, assumed 2048 if not specified explicitly

tsize
required table size, assumed 512 if not specified explicitly

mpu
minimum packet size used in computations

overhead
per−packet size overhead (can be negative) used in computations

linklayer
required linklayer specification.

DESCRIPTION

Size tables allow manipulation of packet sizes, as seen by the whole scheduler framework (of course, the actual packet size remains the same). Adjusted packet size is calculated only once − when a qdisc enqueues the packet. Initial root enqueue initializes it to the real packet's size.

Each qdisc can use a different size table, but the adjusted size is stored in an area shared by whole qdisc hierarchy attached to the interface. The effect is that if you have such a setup, the last qdisc with a stab in a chain "wins". For example, consider HFSC with simple pfifo attached to one of its leaf classes. If that pfifo qdisc has stab defined, it will override lengths calculated during HFSC's enqueue; and in turn, whenever HFSC tries to dequeue a packet, it will use a potentially invalid size in its calculations. Normal setups will usually include stab defined only on root qdisc, but further overriding gives extra flexibility for less usual setups.

The initial size table is calculated by **tc** tool using **mtu** and **tsize** parameters. The algorithm sets each slot's size to the smallest power of 2 value, so the whole **mtu** is covered by the size table. Neither **tsize**, nor **mtu** have to be power of 2 value, so the size table will usually support more than is required by **mtu**.

For example, with **mtu** = 1500 and **tsize** = 128, a table with 128 slots will be created, where slot 0 will correspond to sizes 0−16, slot 1 to 17 − 32, ..., slot 127 to 2033 − 2048. Sizes assigned to each slot depend on **linklayer** parameter.

Stab calculation is also safe for an unusual case, when a size assigned to a slot would be larger than 2ˆ16−1 (you will lose the accuracy though).

During the kernel part of packet size adjustment, **overhead** will be added to original size, and then slot will be calculated. If the size would cause overflow, more than 1 slot will be used to get the final size. This of course will affect accuracy, but it's only a guard against unusual situations.

Currently there are two methods of creating values stored in the size table − ethernet and atm (adsl):

ethernet

This is basically 1−1 mapping, so following our example from above (disregarding **mpu** for a moment) slot 0 would have 8, slot 1 would have 16 and so on, up to slot 127 with 2048. Note, that **mpu** > 0 must be specified, and slots that would get less than specified by **mpu** will get **mpu** instead. If you don't specify **mpu**, the size table will not be created at all (it wouldn't make any difference), although any **overhead** value will be respected during calculations.

atm, adsl

ATM linklayer consists of 53 byte cells, where each of them provides 48 bytes for payload. Also all the cells must be fully utilized, thus the last one is padded if/as necessary.

When the size table is calculated, adjusted size that fits properly into lowest amount of cells is assigned to a slot. For example, a 100 byte long packet requires three 48−byte payloads, so the final size would require 3 ATM cells − 159 bytes.

For ATM size tables, 16 bytes sized slots are perfectly enough. The default values of **mtu** and **tsize** create 4 bytes sized slots.

TYPICAL OVERHEADS

The following values are typical for different adsl scenarios (based on **[1]** and **[2]**):

LLC based:

PPPoA − 14 (PPP − 2, ATM − 12)
PPPoE − 40+ (PPPoE − 8, ATM − 18, ethernet 14, possibly FCS − 4+padding)
Bridged − 32 (ATM − 18, ethernet 14, possibly FCS − 4+padding)
IPoA − 16 (ATM − 16)

VC Mux based:

PPPoA − 10 (PPP − 2, ATM − 8)
PPPoE − 32+ (PPPoE − 8, ATM − 10, ethernet 14, possibly FCS − 4+padding)
Bridged − 24+ (ATM − 10, ethernet 14, possibly FCS − 4+padding)
IPoA − 8 (ATM − 8)

There are a few important things regarding the above overheads:

- IPoA in LLC case requires SNAP, instead of LLC−NLPID (see rfc2684) − this is the reason why it actually takes more space than PPPoA.
- In rare cases, FCS might be preserved on protocols that include Ethernet frames (Bridged and PPPoE). In such situation, any Ethernet specific padding guaranteeing 64 bytes long frame size has to be included as well (see RFC2684). In the other words, it also guarantees that any packet you send will take minimum 2 atm cells. You should set **mpu** accordingly for that.
- When the size table is consulted, and you're shaping traffic for the sake of another modem/router, an Ethernet header (without padding) will already be added to initial packet's length. You should compensate for that by subtracting 14 from the above overheads in this case. If you're shaping directly on the router (for example, with speedtouch usb modem) using ppp daemon, you're using raw ip interface without underlying layer2, so nothing will be added.

  For more thorough explanations, please see **[1]** and **[2]**.

ETHERNET CARDS CONSIDERATIONS

It's often forgotten that modern network cards (even cheap ones on desktop motherboards) and/or their drivers often support different offloading mechanisms. In the context of traffic shaping, 'tso' and 'gso' might cause undesirable effects, due to massive TCP segments being considered during traffic shaping (including stab calculations). For slow uplink interfaces, it’s good to use **ethtool** to turn off offloading features.

SEE ALSO

tc(8), **tc−hfsc**(7), **tc−hfsc**(8),
[**1**] http://ace−host.stuart.id.au/russell/files/tc/tc−atm/
[**2**] http://www.faqs.org/rfcs/rfc2684.html

Please direct bugreports and patches to: <net...@vger.kernel.org>

AUTHOR

Manpage created by Michal Soltys (sol...@ziu.info)