tc-netem(8) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | OPTIONS | LIMITATIONS | EXAMPLES | SOURCES | SEE ALSO | AUTHOR | COLOPHON

NETEM(8) Linux *NETEM*(8)

NAME top

netem - Network Emulator

SYNOPSIS top

tc qdisc ... dev *DEVICE*] **add netem** *OPTIONS*

OPTIONS := [*LIMIT*] [*DELAY*] [*LOSS*] [*CORRUPT*] [*DUPLICATION*] [*REORDERING*] [*RATE*] [*SLOT*] [*SEED*]

LIMIT := **limit** *packets*

DELAY := **delay** *TIME* [*JITTER* [*CORRELATION*]]]
[**distribution** { **uniform** | **normal** | **pareto** | **paretonormal** }]

LOSS := **loss** { **random** *PERCENT* [*CORRELATION*] |
state *p13* [*p31* [*p32* [*p23* [*p14*]]]] |
gemodel *p* [*r* [*1-h* [*1-k*]]] } [**ecn**]

CORRUPT := **corrupt** *PERCENT* [*CORRELATION*]]

DUPLICATION := **duplicate** *PERCENT* [*CORRELATION*]]

REORDERING := **reorder** *PERCENT* [*CORRELATION*] [**gap** *DISTANCE*]

RATE := **rate** *RATE* [*PACKETOVERHEAD* [*CELLSIZE* [*CELLOVERHEAD*]]]]

SLOT := **slot** { *MIN_DELAY* [*MAX_DELAY*] |
distribution { **uniform** | **normal** | **pareto** | **paretonormal** | *FILE* } *DELAY JITTER* }
[**packets** *PACKETS*] [**bytes** *BYTES*]

SEED := **seed** *VALUE*

DESCRIPTION top

The **netem** queue discipline provides Network Emulation functionality for testing protocols by emulating the properties of real-world networks.

The queue discipline provides one or more network impairments to packets such as: delay, loss, duplication, and packet corruption.

OPTIONS top

limit *COUNT*
Limits the maximum number of packets the qdisc may hold when doing delay.

delay *TIME* [*JITTER* [*CORRELATION*]]]
Delays the packets before sending. The optional parameters allow introducing a delay variation and a correlation. Delay and jitter values are expressed in milliseconds; Correlation is set by specifying a percent of how much the previous delay will impact the current random value.

distribution *TYPE*
Specifies a pattern for delay distribution.

uniform
Use an equally weighted distribution of packet delays.

normal Use a Gaussian distribution of delays. Sometimes called a Bell Curve.

pareto Use a Pareto distribution of packet delays. This is useful to emulate long-tail distributions.

paretonormal
This is a mix of **pareto** and **normal** distribution which has properties of both Bell curve and long tail.

loss *MODEL*
Drop packets based on a loss model. *MODEL* can be one of

random *PERCENT*
Each packet loss is independent.

state *P13 [P31 [P32 [P23 P14]]]*
Use a 4-state Markov chain to describe packet loss. *P13* is the packet loss. Optional parameters extend the model to 2-state *P31*, 3-state *P23*, *P32* and 4-state *P14*.

The Markov chain states are:

1 good packet reception (no loss).

2 good reception within a burst.

3 burst losses.

4 independent losses.

gemodel *PERCENT [R [1-H [1-K]]]*
Use a Gilbert-Elliot (burst loss) model based on:

PERCENT
probability of starting bad (lossy) state.

R probability of exiting bad state.

1-H loss probability in bad state.

1-K loss probability in good state.

ecn Use Explicit Congestion Notification (ECN) to mark packets instead of dropping them. A loss model has to be used for this to be enabled.

corrupt *PERCENT*
modifies the contents of the packet at a random position based on *PERCENT*.

duplicate *PERCENT*
creates a copy of the packet before queuing.

reorder *PERCENT*
modifies the order of packet in the queue.

gap *DISTANCE*
sends some packets immediately. The first packets *(DISTANCE - 1)* are delayed and the next packet is sent immediately.

rate *RATE [PACKETOVERHEAD [CELLSIZE [CELLOVERHEAD]]]*
Delays packets based on packet size to emulate a fixed link speed. Optional parameters:

PACKETOVERHEAD
Specify a per packet overhead in bytes. Used to simulate additional link layer headers. A negative value can be used to simlate when the Ethernet header is stripped (e.g. -14) or header compression is used.

CELLSIZE

simulate link layer schemes like ATM.

CELLOVERHEAD
specify per cell overhead.

Rate throttling impacted by several factors including the kernel clock granularity. This will show up in an artificial packet compression (bursts).

slot *MIN_DELAY [MAX_DELAY]*
allows emulating slotted networks. Defer delivering accumulated packets to within a slot. Each available slot is configured with a minimum delay to acquire, and an optional maximum delay.

slot distribution
allows configuring based on distribution similar to **distribution** option for packet delays.

These slot options can provide a crude approximation of bursty MACs such as DOCSIS, WiFi, and LTE.

Slot emulation is limited by several factors: the kernel clock granularity, as with a rate, and attempts to deliver many packets within a slot will be smeared by the timer resolution, and by the underlying native bandwidth also.

It is possible to combine slotting with a rate, in which case complex behaviors where either the rate, or the slot limits on bytes or packets per slot, govern the actual delivered rate.

seed *VALUE*
Specifies a seed to guide and reproduce the randomly generated loss or corruption events.

LIMITATIONS top

Netem is limited by the timer granularity in the kernel. Rate and delay maybe impacted by clock interrupts.

Mixing forms of reordering may lead to unexpected results. For any method of reordering to work, some delay is necessary. If the delay is less than the inter-packet arrival time then no reordering will be seen. Due to mechanisms like TSQ (TCP Small Queues), for TCP performance test results to be realistic netem must be placed on the ingress of the receiver host.

Combining netem with other qdisc is possible but may not always work because netem use skb control block to set delays.

EXAMPLES top

```
# tc qdisc add dev eth0 root netem delay 100ms
```
Add fixed amount of delay to all packets going out on device eth0. Each packet will have added delay of 100ms ± 10ms.

```
# tc qdisc change dev eth0 root netem delay 100ms 10ms 25%
```
This causes the added delay of 100ms ± 10ms and the next packet delay value will be biased by 25% on the most recent delay. This isn't a true statistical correlation, but an approximation.

```
# tc qdisc change dev eth0 root netem delay 100ms 20ms distribution normal
```
This delays packets according to a normal distribution (Bell curve) over a range of 100ms ± 20ms.

```
# tc qdisc change dev eth0 root netem loss 0.1%
```
This causes 1/10th of a percent (i.e 1 out of 1000) packets to be randomly dropped.

An optional correlation may also be added. This causes the random number generator to be less random and can be used to emulate packet burst losses.

```
# tc qdisc change dev eth0 root netem duplicate 1%
```
This causes one percent of the packets sent on eth0 to be duplicated.

```
# tc qdisc change dev eth0 root netem loss 0.3% 25%
    This will cause 0.3% of packets to be lost, and each
    successive probability depends is biased by 25% of the
    previous one.

There are two different ways to specify reordering.  The gap
method uses a fixed sequence and reorders every Nth packet.
# tc qdisc change dev eth0 root netem gap 5 delay 10ms
    This causes every 5th (10th, 15th, …) packet to go to be sent
    immediately and every other packet to be delayed by 10ms.
    This is predictable and useful for base protocol testing like
    reassembly.

The reorder form uses a percentage of the packets to get
misordered.
# tc qdisc change dev eth0 root netem delay 10ms reorder 25% 50%
In this example, 25% of packets (with a correlation of 50%) will
get sent immediately, others will be delayed by 10ms.

Packets will also get reordered if jitter is large enough.
# tc qdisc change dev eth0 root netem delay 100ms 75ms
    If the first packet gets a random delay of 100ms (100ms base
    - 0ms jitter) and the second packet is sent 1ms later and
    gets a delay of 50ms (100ms base - 50ms jitter); the second
    packet will be sent first.  This is because the queue
    discipline tfifo inside netem, keeps packets in order by time
    to send.

If you don't want this behavior then replace the internal queue
discipline tfifo with a simple FIFO queue discipline.
# tc qdisc add dev eth0 root handle 1: netem delay 10ms 100ms
# tc qdisc add dev eth0 parent 1:1 pfifo limit 1000

Example of using rate control and cells size.
# tc qdisc add dev eth0 root netem rate 5kbit 20 100 5
    Delay all outgoing packets on device eth0 with a rate of
    5kbit, a per packet overhead of 20 byte, a cellsize of 100
    byte and a per celloverhead of 5 bytes.

It is possible to selectively apply impairment using traffic
classification.
# tc qdisc add dev eth0 root handle 1: prio
# tc qdisc add dev eth0 parent 1:3 handle 30:    tbf rate 20kbit buffer 1600 limit  3000
# tc qdisc add dev eth0 parent 30:1 handle 31:    netem delay 200ms 10ms distribution normal
# tc filter add dev eth0 protocol ip parent 1:0 prio 3 u32    match ip dst 65.172.181.4/32 flowid 1:3
    This example uses a priority queueing discipline; a TBF is
    added to do rate control; and a simple netem delay.  A filter
    classifies all packets going to 65.172.181.4 as being
    priority 3.
```

SOURCES top

1. Hemminger S. , "Network Emulation with NetEm", Open Source Development Lab, April 2005 ⟨http://devresources.linux-foundation.org/shemminger/netem/LCA2005_paper.pdf⟩

2. Salsano S., Ludovici F., Ordine A., "Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel", available at ⟨http://netgroup.uniroma2.it/NetemCLG⟩

SEE ALSO top

tc(8)

AUTHOR top

Netem was written by Stephen Hemminger at Linux foundation and was inspired by NISTnet.

Original manpage was created by Fabio Ludovici <fabio.ludovici at yahoo dot it> and Hagen Paul Pfeifer <hagen@jauu.net>.

COLOPHON top

This page is part of the *iproute2* (utilities for controlling TCP/IP networking and traffic) project. Information about the project can be found at ⟨http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2⟩. If you have a bug report for this manual page, send it to netdev@vger.kernel.org, shemminger@osdl.org. This page was obtained from the project's upstream Git repository ⟨https://git.kernel.org/pub/scm/network/iproute2/iproute2.git⟩ on 2023-12-22. (At that time, the date of the most recent commit that was found in the repository was 2023-12-20.) If you discover any rendering problems in this HTML version of the page, or you believe there is a better or more up-to-date source for the page, or you have corrections or improvements to the information in this COLOPHON (which is *not* part of the original manual page), send a mail to man-pages@man7.org

iproute2 **25 November 2011** ***NETEM*(8)**

Pages that refer to this page: ovs-vswitchd.conf.db(5)