Linux Traffic Control

Traffic control (tc) is a very useful Linux utility that gives you the ability to configure the kernel packet scheduler. If you are looking for reasons to mess with the kernel scheduler, here are a few: Firstly, it's fun to play with the different options and become familiar of all of Linux's features. In addition, you can utilize Linux's helpful tools to simulate packet delay and loss for UDP or TCP applications, or limit the bandwidth usage of a particular service to simulate Internet connections (DSL, Cable, T1, etc).

On Debian Linux, tc comes bundled with iproute, so in order to install it you have to run:

```
apt-get install iproute
```

Network Delay

The first example is how to add constant delay to an interface. The syntax is as follows (run this as root):

```
tc qdisc add dev eth0 root netem delay 200ms
```

Here is what each option means:

qdisc: *modify the scheduler (aka queuing discipline)*
add: *add a* new *rule*
dev eth0: *rules will be applied on device eth0*
root: *modify the outbound traffic scheduler (aka known as t*
netem: *use the network emulator to emulate a WAN prope*
delay: *the network property that is modified*
200ms: *introduce delay of 200 ms*

```
netbeez.net$ ping google.com
PING google.com (172.217.6.78) 56(84) bytes of data.
64 bytes from sfo07s17-in-f78.1e100.net (172.217.6.78):
icmp_seq=1 ttl=53 time=11.9 ms
64 bytes from sfo07s17-in-f78.1e100.net (172.217.6.78):
icmp_seq=2 ttl=53 time=12.0 ms
```

Here is what ping looks like after applying this rule:

```
netbeez.net$ ping google.com
PING google.com (172.217.5.110) 56(84) bytes of data.
64 bytes from sfo03s07-in-f14.1e100.net (172.217.5.110):
icmp_seq=1 ttl=53 time=213 ms
64 bytes from sfo03s07-in-f14.1e100.net (172.217.5.110):
icmp_seq=2 ttl=53 time=210 ms
```

In order to display the active rules use:

```
netbeez.net$ tc qdisc show  dev eth0
qdisc netem 8003: root refcnt 2 limit 1000 delay 200.0ms
```

You can see that details of the existing rules that adds 200.0 ms of latency.

To delete all rules use the following command:

```
netbeez.net$ tc qdisc del dev eth0 root
```

And now we can see what are the default rules of the linux scheduler:

```
netbeez.net$ tc qdisc show  dev eth0
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

Without going into too much detail, we see that the scheduler works under First In First Out (FIFO) rules which is the most basic and fair rule if you don't want to set any priorities on specific packets. You can think about it like the line at the bank: customers are being taken care off in the order they arrive.

Note that if you have an existing rule you can change it by
if you don't have any rules you add rules with " `tc qdisc add`
Here are some other examples:

-Delay of 100ms and random +-10ms uniform distribution:

```
tc qdisc change dev eth0 root netem delay 100ms 10ms
```

-Delay of 100ms and random 10ms uniform variation with correlation value 25% (since network delays are not completely random):

```
tc qdisc change dev eth0 root netem delay 100ms 10ms 25%
```

-Delay of 100ms and random +-10ms normal distribution (other distribution options are pareto, and paretonormal):

```
add dev eth0 root netem delay 100ms 20ms distribution normal
```

Packet Loss and Packet Corruption

Without explaining the syntax in detail here is now to introduce a packet loss of 10%:

```
tc qdisc add dev eth0 root netem loss 10%
```

We can test this by running a ping test with 100 ICMP pacl aggregate statistics look like:

```
netbeez.net$ ping google.com -c 100
PING google.com (216.58.194.174) 56(84) bytes of data.
64 bytes from sfo07s13-in-f174.1e100.net (216.58.194.174): icmp_seq=1 ttl=53 time=10.
.....
64 bytes from sfo07s13-in-f174.1e100.net (216.58.194.174): icmp_seq=99 ttl=53 time=11.
64 bytes from sfo07s13-in-f174.1e100.net (216.58.194.174): icmp_seq=100 ttl=53 time=10.5 ms
```

The following rule corrupts 5% of the packets by introducing single bit error at a random offset in the packet:

```
tc qdisc change dev eth0 root netem corrupt 5%
```

This one duplicates 1% of the sent packets:

```
tc qdisc change dev eth0 root netem duplicate 1%
```

Bandwidth limit

In order to limit the egress bandwidth we can use the following command:

```
tc qdisc add dev eth0 root tbf rate 1mbit burst 32kbit latency 400ms
```

tbf: *use* the *token buffer filter to manipulate traffic rates*
rate: *sustained maximum rate*
burst: *maximum allowed burst*
latency: *packets with higher latency get dropped*

The best way to demonstrate this is with an iPerf test. In my lab I get 95 Mbps of performance before applying any bandwidth rules:

```
netbeez.net$ iperf -c 172.31.0.142
------------------------------------------------------------
Client connecting to 172.31.0.142, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 172.31.0.25 port 40233 connected with 172.31.0.142 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   113 MBytes  95.0 Mbits/sec
```

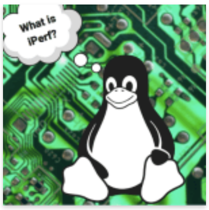
And here is the performance after applying the 1 Mbps lim

```
netbeez.net$ iperf -c 172.31.0.142
------------------------------------------------------------
Client connecting to 172.31.0.142, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 172.31.0.25 port 40232 connected with 172.31.0.142 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-11.0 sec  1.50 MBytes  1.14 Mbits/sec
```

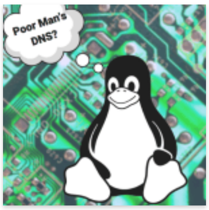
examples of advanced configurations are maximizing TCP throughput on an asymmetric link, prioritizing latency sensitive traffic, or managing oversubscribed bandwidth. Some of these tasks can be performed effectively with other tools or services, but tc is a great utility to have in your arsenal when the need arises.

Related Posts

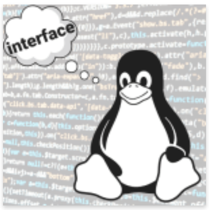
Linux for Network Engineers: How to use nping



Linux for Network Engineers: How to Use iPerf



Linux for Network Engineers: Poor Man's DNS



Linux for Network Engineers: Interface Information

Linux for Network Engineers: iptables

Linux for Network Engineers: What's New in the iPerf 2 Release

Powered by

Search for content

Search...

Search

Request a 15-day free trial, monitor your networks and remote user experience

Welcome to NetBeez, chat with us if you need help.

Categories

Categories

Select Category

25 Comments https://netbeez.net 1 Login

♡ Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Scott Henderson • 2 years ago

I'm using this on a Raspberry Pi 4b to emulate latency between two nodes (tc qdisc add dev eth1 root netem delay 35ms) (and the same for eth2), but whenever I exceed ~30ms latency total across both eth ports, packet loss goes through the roof. I'm using USB-GbE adapters which create eth1 and eth2; eth0 is the on-board NIC (using that for OOB management). Any thoughts on wha might be causing the loss?

1 ^ | ˅ • Reply • Share ›

Welcome to NetBeez, chat with us if you need help.

rick jones ➦ Scott Henderson • 2 years ago

Exceeding 30ms latency via netem may force the queue of delayed packets to hit the maximum and so traffic gets dropped. Presumably the output of:

additional qdisc like "fq_codel" which strives to avoid buffer-bloat by limiting the residency time of packets in a queue. It will then likely start to drop earlier, causing the sending TCP to back-off and keep the queue depths at a manageable level. You might also consider switching your congestion control algorithm on the sending side to "bbr" which also seeks to minimize queue depths at a bottleneck queue.

1 ⌃ | ⌄ • Reply • Share ›

Panos Vouzis ➦ Scott Henderson • 2 years ago

Although 30 ms is not that much, the only thing that comes to my might is that the TCP window on your system needs to be bumped up. Take a look at this on how to do that: https://netbeez.net/blog/tc...

Alternatively, try to test this with two different hosts. I am not sure how the USB-GbE works.

I hope this helps.

⌃ | ⌄ • Reply • Share ›

Orestis Zekai • 2 years ago

root is used to apply rules to the egress qdisc. How can we apply rules to ingress qdics?

1 ⌃ | ⌄ • Reply • Share ›

rick jones ➦ Orestis Zekai • 2 years ago

You need to setup an "ifb" interface and direct inbound traffic through that so the inbound traffic becomes "egress" on the ifb interface so qdisc rules may be applied. A web search or three for "qdisc inbound traffic" will probably find examples.

⌃ | ⌄ • Reply • Share ›

rick jones • 2 years ago

An example of netem's "reorder" would be an excellent addit this page.

⌃ | ⌄ • Reply • Share ›

Dinesh Bhat • 2 years ago

I have used the following command to introduce th delay: tc qdisc

the packets in the receiver buffer length experience the delay? i.e if at time t0 the application sends the 10packets and let's assume that, all 10packets fits in the receiver buffer. Hence, TCP tries to transfer all 10 packets at a time. Now, will my 10th packet exit at time t0 + 2ms x 10 OR t0 + 2ms?
Example:
I have run the scp command to transfer data of 3Gb between VMs
.
Without the latency injected, I saw the copy taking 25seconds and after injecting the latency of 2ms , I see the time taken as 27seconds which is 2 seconds more. While my expectations was, scp to experience more delay.
Can you please help me to understand on how qdisc is applying latency?

Thanks,
Dinesh

^ | ⌄ • Reply • Share ›

rick jones ➦ Dinesh Bhat • 2 years ago

Simply introducing delay will not *necessarily* result in lower TCP bulk throughput performance. One of the commonly-expressed limits to TCP performance is:

Throughput <= EffectiveWindowSize / RoundTripTime

Now, if "EffectiveWindowSize" is large enough to still allow getting (close) to link-rate even when you add the round-trip latency, you won't see a significant change in throughput.

There is a great deal to "EffectiveWindowSize" - that will be the minimum of the receiving TCP's advertised window, the sending TCP's computed congestion window, the limit of the sending socket's SO_SNDBUF size and how muc data the application (eg scp) is willing to have outstan at one time.

^ | ⌄ • Reply • Share ›

Panos Vouzis ➦ Dinesh Bhat • 2 years ago

Hi Dinesh,

scp is probably not the best test to test latency increase since adding a latency of 2ms means that all packets will

The best way to see the effect of latency (or packet loss or jitter) is to use a latency-sensitive test such as VoIP: https://netbeez.net/blog/im...

Reply • Share ›

Dinesh Bhat ➦ Panos Vouzis • 2 years ago

Thanks Panos for the quick response.
With your explanation, I assume that, if there are 10 packets produced by the application and all the packets fits in the tcp receiver buffer (at time t0), the last packet (10th packet) will exit from qdisc at time: t0 + 2ms * 10 where 2ms is the latency added at qdisc. Please correct me otherwise.
If my assumption is correct, in case applications processing huge amount of data (writing to disk or querying data between vms in a application cluster, etc..) will result in lot of packets arriving at qdisc (as the application continues to produce the packets) and chances that, the qdisc might overflow. How qdisc handles the overflow of packets due to added delay using tc command? Will it push back the application in case of TCP traffic which intern slow down the application?

Reply • Share ›

rick jones ➦ Dinesh Bhat • 2 years ago

Linux* has a feature called "TCP small queues" which attempts to keep the number of packets queued to the NIC at the minimum required to keep the NIC busy. It does this by looking a the reference count of (some of) the outstanding TCP segments when TCP goes to send more. Unless NIC driver "cheats" by freeing/derefere the packets early (prior to transmit completion, which the "virtio_net" drive known to do unless someone enables ' tx") this means a single TCP flow/stream/connection is unlikely to overflow the qdisc. If the qdisc *is* being overflowed, that will likely stand-out in some of the output

Dinesh Bhat → nick jones • 2 years ago

Great .. I'm on Ubuntu 16 and believe, there will not be any overflows at qdisc.
Thanks,
Dinesh

^ | ⌄ • Reply • Share ›

Panos Vouzis → Dinesh Bhat • 2 years ago

Dinesh, I am not sure how tc handles overflows. I suspect that the packets that overflow are lost. If that's the case, you might need to change the TCP window size on the system by following this https://netbeez.net/blog/tc...

Product

Product Overview

Network Monitoring

WAN Monitoring

WiFi Monitoring

Multi-Cloud Monitoring

Remote Worker Network Monitoring

Customers

Pricing

Get A Quote

Resources

White Papers

Webinars

Welcome to NetBeez, chat with us if you need help.

Partners

Careers

Support

Contact Us

Sales:
sales@netbeez.net
1-844-NETBEEZ
(638-2339) ext.1

Support:
support@netbeez.net
1-844-NETBEEZ
(638-2339) ext.2

Welcome to NetBeez, chat with us if you need help.