Using tc for QOS

tropf

ABSTRACT

tc is rather complicated, so I built a wrapper script around it. Here I elaborate a little on the background.

1. Introduction

I built this script, initially for using a NFS via the same link that also provides services. As the services require low latency, and the NFS has the potential to use the entire bandwidth without further notice the NFS traffic has to be restricted.

1.1. Used tools and terminology

This script uses `tc`, which itself is an abbreviation for "traffic control". `tc` is a f airly complicated piece of software—which is appropiate for complicated scenarios, but simple scenarios (like this one) are thus not simple to implement. Its purpose is to reorder and drop packets if they exceed allowed quotas. For classification of packets `iptables` is employed (not shown here at all), as I hope most people reading this already know how to use it.

I will ignore the usual terminology sorrounding `tc` here, as I want to keep the barrier for understanding low.

1.2. Concept

We have a link with a known capacity for outgoing ("egress") traffic (here: `eth0` with 1 gbit/s). To assert control over the traffic, we limit the total available bandwidth to **less than** the total available bandwidth: this way we can decide which traffic is dropped, otherwise the link itself "decides".

All traffic is then put into one of three different classes: High, medium and low priority traffic. By default traffic is medium priority. Each of these classes has a**guaranteed minimum band width**. After the guaranteed minimum bandwidth has been assigned, the remaining bandwidth is split in an unspecified way, generally equally.

As the guaranteed minimum bandwidth is assigned without further checks, the sum of all guaranteed bandwidth must be less than the total available bandwidth. Inside the priority classes all traffic is treated equally.

Incoming traffic ("ingress") is not subject to any form of traffic control, as we can't really control how much traffic is sent to us anyways. And if it passes the link, it would be somewhat wasted to drop it and require the sender to send it again.

1.3. Notes on IPv6

Even though this is specified for IPv6, the server I created this for never gets any IPv6 traffic (people use IPv6 please, adding the AAAA-record was work). So I have not tested the suitability for IPv6 live.

› Don't forget to add rules also for IPv6 when using `iptables`.

1.4. Drawbacks

Note that this just drops traffic and behaves like a unreliable link towards applications. But TCP should be able to deal with that. Also as already stated you have to limit the total bandwidth to less than your actually available bandwidth, so you probably waste around 1% of your bandwidth.

Please also consider this as a new component that can break and is rather difficult to debug.

2. The Script itself

direct link ⟨`../blob/tc.sh`⟩

› Don’t forget `chmod +x` for all these here.

2.1. Full text

```
#!/bin/bash

IFACE="eth0"
RATE_TOTAL="970mbit"
RATE_HIGH="900mbit"
RATE_MEDIUM="50mbit"
RATE_LOW="8mbit"

# please refer to tc-u32(8) for format, rule VAL_MASK_16
# to use a single port use <port, no leading 0> 0xffff
read -r -d "" PORT_PATTERNS_HIGH << EOF
22 0xffff
EOF

read -r -d "" PORT_PATTERNS_LOW << EOF
80 0xffff
443 0xffff
EOF

# clear all qdiscs
tc qdisc del dev $IFACE root 2> /dev/null > /dev/null

# create a qdisc using htb and send traffic to 1:5 by default
tc qdisc add dev $IFACE root handle 1: htb default 5

# define limits for 1: (all packets on $IFACE)
tc class add dev $IFACE parent 1: classid 1:1 htb rate $RATE_TOTAL burst 64k

# define limits for 1:3 (high prio)
tc class add dev $IFACE parent 1:1 classid 1:3 htb rate $RATE_HIGH ceil $RATE_TOTAL
# define limits for 1:5 (medium prio)
tc class add dev $IFACE parent 1:1 classid 1:5 htb rate $RATE_MEDIUM ceil $RATE_TOT
# define limits for 1:9 (low prio)
tc class add dev $IFACE parent 1:1 classid 1:9 htb rate $RATE_LOW ceil $RATE_TOTAL

# add sfq for all subclasses
tc qdisc add dev $IFACE parent 1:3 handle 3: sfq perturb 10
tc qdisc add dev $IFACE parent 1:5 handle 5: sfq perturb 10
tc qdisc add dev $IFACE parent 1:9 handle 9: sfq perturb 10

# add filters for marked packets
tc filter add dev $IFACE protocol ip parent 1: prio 0 handle 3 fw flowid 1:3
tc filter add dev $IFACE protocol ip parent 1: prio 0 handle 5 fw flowid 1:5
tc filter add dev $IFACE protocol ip parent 1: prio 0 handle 9 fw flowid 1:9

tc filter add dev $IFACE protocol ipv6 parent 1: prio 0 handle 3 fw flowid 1:3
tc filter add dev $IFACE protocol ipv6 parent 1: prio 0 handle 5 fw flowid 1:5
tc filter add dev $IFACE protocol ipv6 parent 1: prio 0 handle 9 fw flowid 1:9

# filters for high prio (executed before low prio matching)
echo "$PORT_PATTERNS_HIGH" | while read PORT_PATTERN ; do
    echo "processing (high): $PORT_PATTERN"
    tc filter add dev $IFACE protocol ip   parent 1: prio 0 u32 match ip  sport $PO
    tc filter add dev $IFACE protocol ipv6 parent 1: prio 0 u32 match ip6 sport $PO
```

```
done

# filters for low prio (executes after high prio matching)
echo "$PORT_PATTERNS_LOW" | while read PORT_PATTERN ; do
    echo "processing (low):  $PORT_PATTERN"
    tc filter add dev $IFACE protocol ip   parent 1: prio 0 u32 match ip  sport $PO
    tc filter add dev $IFACE protocol ipv6 parent 1: prio 0 u32 match ip6 sport $PO
done

# remaining traffic is medium priority
```

2.2. Download with wget

```
wget https://www.tropf.io/blob/tc.sh
```

2.3. From copy-paste

Copy-paste this block into a file (here: `/tmp/f`)...

```
H4sIAAAAAAAAA8VWXW/aMBR9z6+4C30AqRUBGtawUgltbK3Wjqpl2yNyEqexmsTUdvoh7cfv2iE0
QCmMTlukIOfa59xzP2xTe9f0Wdb0iYwt6+zz4OOwb1MVO7Z1NRgPJ+PReHDet733TuozNTOenn05
RZtTtV0MP519v+jbbtV4PvrZt48Kg1WDaUKJpCBoRAUoDio4yDvt+lEDIi70mxK1DyJPKPwYnE8u
BtdfJ60uAnFtjkACkmU3ODvlQhnLsR7tQ8YBqUOcBOcEnMcIH0ugBQ4EHIRg23A5uhpPLgfj8fDq
27WJAI6PYTj6bLXbJUJ/vQrDeErUkVOiDg87CwQ1CFCMAJIkcBcyGUhLBcUIQprgew97JtMgOFfQ
PoEm2ppZjoDK2DChGqUDL+C5jh9i5QPJQpAUf5QgUcQCnaJWzwX/Cfkjkifq2SkJwxWnMRJgIls9
wzaDgKt94gfLKCQsZUqayuCquo5mSoJbijaezbga2kmQECmXnUyJoJnSSDPNQhy2jC+hA9p7bi7w
cyEVdA9v1znvQD1mNzFMBeNbeGxVXHaWXZrCB5QlL0swPsBZI8SFekpDlqe7SHGXpRQ7ZqMYd40Y
D+oJf9hFibesRLf1Jhmero5ml9GdUaAbQua+oaVybbfNFXTKnuv0DMeUCpULH1rOZqxbYt0/x3ol
1lvBziKKWKKoKPKaEnFLw7LTNXsxu0IvuOIBT4BNK71edM88UIgeIMIiFa34NjJ3gcx9G5m3QIal
3YbtvvtXI32FbpdYX6Fbiba2UPL52QJ1+kiDXGH9fYozuOFmGwzbQgUxnr0NiwYxB3tv9Tqx4Rc8
xCzR9xveINUF8AFCbgE+BRplB1Sas9ycbI0eLBDaZu1WFYbVqPFKLfSaeWnuygX6aqW2dfRSequO
upschTyjy5mfp7dMPJ5fkdbxXJINiceTa8e8o29M+3/Ku/ev8u7N8y5oSlimQy//MTAJldtMMPVk
/QZFAFdRDQoAAA==
```

...and then extract it with:

```
base64 -d /tmp/f | gzip -d > tc.sh
```

3. Adjusting for your local setup

3.1. In the script

Adjust the beginning areas. You should touch (and modify):

- `IFACE` to your interface with outgoing traffic
- `RATE_TOTAL` to the maximum amount of traffic that should flow (not 100% of your uplink, see above)
- `RATE_{HIGH|MEDIUM|LOW}` to the guaranteed minimum bandwidth per priority class
- `PORT_PATTERNS_{HIGH|LOW}` to the ports you want to use; maybe even delete all of the lines specifying ports and just use `iptables`.

3.2. Automatic execution

Execute this script on startup. I use this systemd-unit:

```
[Unit]
Description=Traffic Control Custom Rules
Wants=network.target
After=network.target

[Service]
Type=oneshot
ExecStart=/root/tc.sh

[Install]
WantedBy=multi-user.target
```

3.3. Classifying traffic

For simple port-based rules you can add these in the script itself (copy the example lines). Otherwise build a iptables rule to match your desired packets and add at the end:

```
-j MARK --set-mark {3|5|9}
```

For high priority use mark 3, medium mark 5 and low mark 9.

Usually those rules belong in the table `mangle` in the chain `postrouting`, though `tc` does not care about how exactly you marked your packets.

› Keep in mind that iptables only works on IPv4 by default.

› Don’t forget to make your iptables rules persistent.

4. See also

- **tc**(8): man page of `tc`; also check the references there
- Advanced Traffic Control in the Arch linux wiki ⟨`https://wiki.archlinux.org/index.php/Advanced_traffic_control`⟩: most comprehensive article on the matter I came across
- Traffic Control HOWTO ⟨`https://tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/`⟩ in the Linux Documentation Project: in-depth and complicated explanation of `tc`