ECE 1545

Lab S2

Packet Scheduling in Linux (Part 2)

Purpose:

Become familiar with the configuration of packet scheduling algorithms in Linux. Conduct experiments to evaluate the impact of scheduler configuration in Linux.

This lab assumes that Part 1 of Lab S2 has been completed.

Software Tools:

- The lab/assignment assumes that there is a Ubuntu (or similar) Linux installation, possibly as a virtual machine.
- It is assumed that bokeh and iperf3 are installed on the Ubuntu system.

What to turn in:

- For the lab (to be completed in the lecture) there is nothing to turn in. The assignment consists of parts 6 and 7, which request to prepare a lab report.

Version 2 (November 2019)

Table of Content

Prelab

Before starting this lab, you need to become familiar with the "tc" (traffic control) Linux command and the configuration of a qdisc scheduler.

The following are sources of information:

- The slide set "Packet Scheduling in Linux" available where you found this document.
- "How to Use the Linux Traffic Control" available at

  https://netbeez.net/blog/how-to-use-the-linux-traffic-control/

- "Traffic Control HOWTO" available at

  https://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/

Use this information to provide answers to the problems on the following page.

Questions for Prelab 2

1. Provide the tc command that configures a token bucket filter (TBF) to the interface eth0 with the following parameters:

    - 50KB max. buffer size,
    - bucket of size 1500 bytes,
    - rate is 100 Mbps.

2. Provide the tc command that configures a Priority scheduler to the interface eth0 with three priority levels.

3. Provide the tc commands that configure a Deficit Round Robin (DRR) scheduler to the interface eth0. The DRR scheduler has 2 traffic classes that are each given equal weight.

4. Provide the tc commands that configure a Deficit Round Robin (DRR) scheduler to the interface eth0. The DRR scheduler has 2 traffic classes, where the rate guarantee of the first class is twice the rate guarantee of the second class.

5. For the DRR scheduler with 2 traffic classes above, provide a tc filter that assigns:

    - Traffic with destination port number 5555 to one class;
    - Traffic with destination port number 6666 to the second class.

6. Provide the tc commands that configures a Deficit Round Robin (DRR) scheduler to the interface eth0. The DRR scheduler has 2 traffic classes, where the rate guarantee of the first class is twice the rate guarantee of the second class.

7. For a DRR scheduler, provide the commands that, in addition to the allocation to two classes as given in Problem 5, limit the maximum transmissions of the DRR scheduler by a TBF with the parameters given in Problem 1 above.

8. Provide the qdisc commands that display

    - the current qdisc configuration of interface eth0; **(tc qdisc show dev eth0)**
    - the current configuration of traffic classes on interface eth0. **(tc class show dev eth0)**

Overview

This lab assumes that you have completed Lab S1, and have the components shown in the next figure working:

- A script to start iperf3 clients and servers,
- A visualization, done by the Python bokeh library, that shows the data rate of the iperf3 clients and servers over a moving time window.

The final step is to insert a Qdisc packet scheduler[1] at the ingress of the loopback interface.

In this lab, you will work with the following Qdisc scheduling algorithms:

- PRIO – static priorities,
- DRR – Deficit Round Robin,
- HTB – Hierarchical Token Bucket.

[1] Normally, we would place the scheduler before traffic enters the loopback interface. The reason that we place the scheduler after the loopback interface (when it departs from the loopback interface) is that there are otherwise interactions with higher-layer protocols. By moving the scheduler to the egress of the loopback interface, such interactions are avoided.

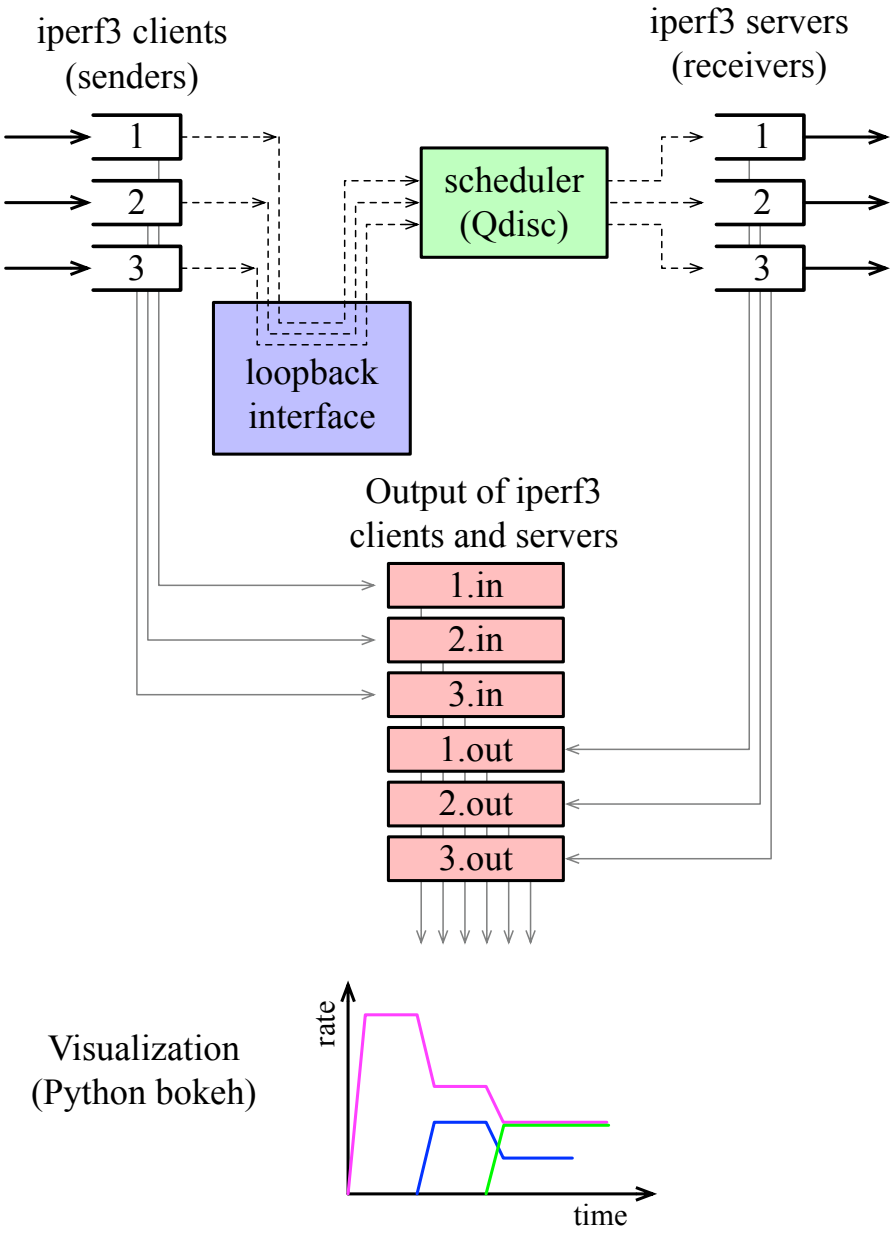


Figure 1. Overview of measurements and their visualization.

Part 1. Scheduling at the Ingress / FIFO

In this part, you will measure scheduling algorithms on a single Unix system. This can be done with the loopback interface (which has domain name "localhost"). All transmissions to the loopback interface are returned to the local system.

Scheduling is normally down for outgoing traffic, that is for traffic that departs from a host or router. From the perspective of the host or router, this the *egress point*. This is shown in Figure 2 for the loopback interface.

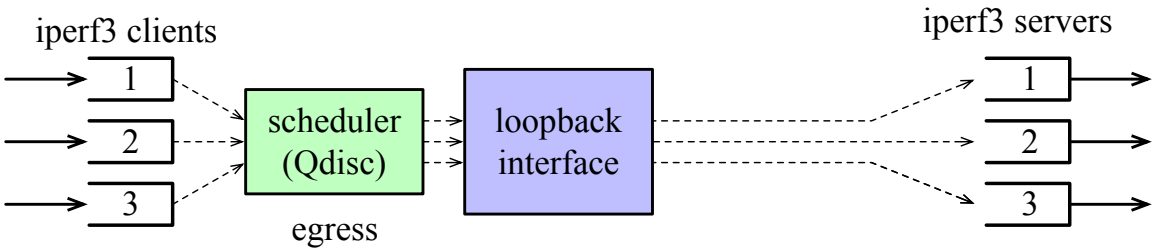


Figure 2. Egress scheduler.

When performing scheduling at the egress, we find that there are interactions with higher-layer protocols. For example, there may be restrictions that a packet from a client can enter the scheduler only when the previous packet from the client has been transmitted. This becomes more of an issue when we want to set the transmission rate of the scheduler to a value that is well below the capacity of the scheduler. For this reason, for the purposes of this lab, you will configure the scheduler after the loopback interface, at the *ingress point*, as shown in Figure 3.

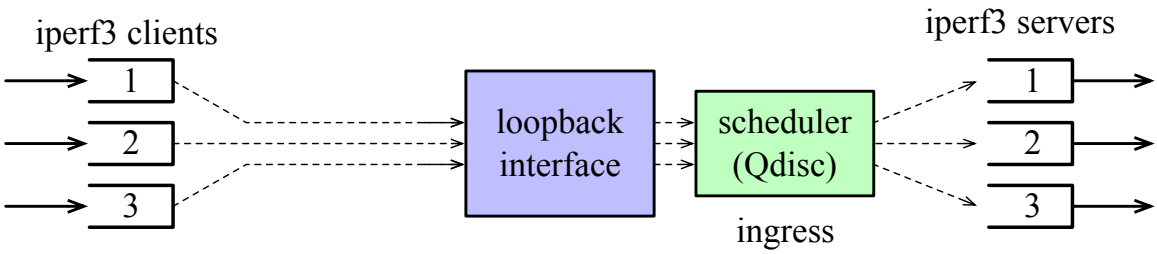


Figure 3. Ingress scheduler.

Since the loopback interface does not impose limits on transmission rates, other than the memory speed of the host, performing scheduling at the ingress does not alter results. On the other hand, ingress scheduling decouples the iper3 clients from the scheduler and prevents any interference by higher-layer protocols.

In Linux, ingress scheduling can be configured using a virtual network interface, called Intermediate Function Block (IFB). The following steps walk you through establishing a token bucket filter at the ingress point.

Exercise 1.1 Configuring a token bucket filter for incoming traffic

1. Reset the loopback interface by deleting the current qdisc configuration.

```
$ sudo tc qdisc del dev lo root
```

2. Create a virtual IFB interface. The command creates the interface ifb0.

```
$ sudo modprobe ifb numifbs=1
```

Enable the interface with

```
$ sudo ip link set dev ifb0 up
```

3. The next step is to add an ingress qdisc to the loopback interface with

```
$ sudo tc qdisc add dev lo handle ffff: ingress
```

4. Then redirect all ingress traffic (except the traffic between the web browser and the bokeh server) from the loopback interface to ifb0.

```
$ sudo tc filter add dev lo parent ffff: prio 1 protocol ip u32 \
                  match ip protocol 6 0xff action connmark pass
$ sudo tc filter add dev lo parent ffff: prio 2 protocol ip u32 \
            match u32 0 0 action mirred egress redirect dev ifb0
```

The first command ensures that the TCP traffic between the web browser and the bokeh server, which is used for visualization, is not redirected to interface *ifb0*. The second command sends all other traffic to interface *ifb0*.

5. Finally, set up the token bucket with

```
$ sudo tc qdisc del dev ifb0 root
$ sudo tc qdisc add dev ifb0 root handle 1:  \
                  tbf rate 10Mbit burst 10k limit 1M
```

The first command deletes any previous configuration of the ifb0 interface.

The second command attaches to interface ifb0 a token bucket filter with a burst size of 10 kB and a rate of 10 Mbps, as illustrated in Figure 4.

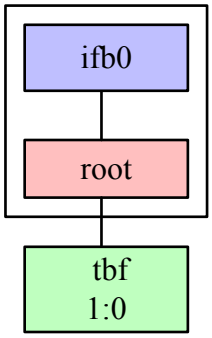


Figure 4. Token bucket filter attached to ifb0.

6. View the results of the configuration with

```
$ tc qdisc show dev ifb0
$ tc class show dev ifb0
```

Exercise 1.2 Measurement Experiments of FIFO

With the configuration from the previous exercise, repeat the measurement of iperf3 transmissions from Exercise 4.2 of Lab S1. Note that without configuring a particular scheduling algorithm, the queueing discipline is FIFO.

1. Start the bokeh server

```
$ bokeh serve serverbokeh.py
```

2. Start a web browser, e.g., Firefox, and type the following URL:

   http://localhost:5006/serverbokeh

3. In the home directory, write the following four lines into a file with name myexperiment.sh:

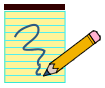
```
./startiperf.sh 5555 60 10 0 &
./startiperf.sh 6666 60 6 10 &
./startiperf.sh 3333 60 3 20 &
./startiperf.sh 2222 60 1 30 &
```

4. Run the script with the command

```
$ bash myexperiment.sh
```

5. Refresh/Reload the above URL in the browser.

6. When the iperf clients are completed, take a screen snapshot of the visualization shown in the browser, and save it to a file.



For the lab report:

1. Turn in the saved screen shot, with a brief description.

2. For the time period where all iperf3 clients are transmitting, compare the data rates of the various iperf3 transmissions, with those from Exercise 4.1 of Lab S1.

3. For the measurements in this exercise (Exercise 1.2), relate the transmission rate of an iperf3 client at the token bucket filter to the rate at which an iperf3 client is transmitting.

Exercise 1.3 Cleanup of the ingress configuration

When you are done with the measurement experiment, you may want to remove the virtual interface ifb0. This is done with the commands

```
# Remove ifb0 by shutting down the module responsible for it
$ sudo modprobe -r ifb
# Reset ingress module
$ sudo tc qdisc del dev lo ingress
```

The first command removes ifb0 by shutting down the relevant module. The second command resets the ingress configuration of the loopback interface.

As an alternative, you can reset all configurations by rebooting Linux.

Part 3. Priority (PRIO)

Next you configure a priority scheduler. Continuing with the configuration in Part 2, the priority scheduler is added to the token filter with rate 10 Mbps to the virtual ifb0 interface. In this fashion the total rate of the priority scheduler does not exceed 10 Mbps. The Qdisc configuration is given in Figure 5.

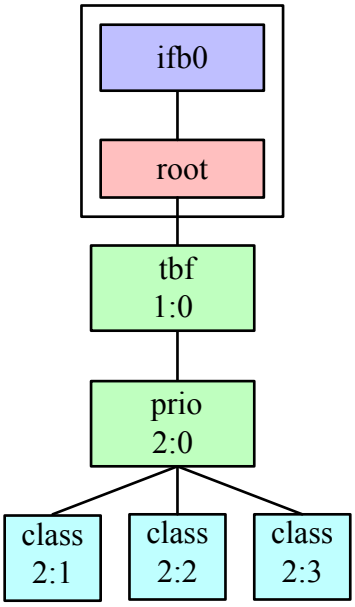


Figure 5. Priority scheduler (prio qdisc).

The command to add a priority scheduler to the configuration from Part 2 is

```
$ sudo tc qdisc add dev ifb0 parent 1: handle 2: prio
```

By default, the configuration of a priority scheduler creates three classes with minor numbers `1', `2', and `3', where `1' is the highest priority, and `3' is the lowest priority.

After setting up the priority scheduler, the next step is to associate traffic flows with traffic classes. Using tc, this is done by specifying so-called `filters'. When generating traffic flows with iperf3, it makes sense to associate the port numbers used for transmissions with a traffic class. The commands that associate destination port 4444 with class 2:1, port 5555 with class 2:2, and port 6666 with class 2:3 are

```
$ sudo tc filter add dev ifb0 parent 2: \
      protocol ip u32 match ip dport 4444 0xffff classid 2:1
$ sudo tc filter add dev ifb0 parent 2: \
      protocol ip u32 match ip dport 5555 0xffff classid 2:2
```

```
$ sudo tc filter add dev ifb0 parent 2: \
      protocol ip u32 match ip dport 6666 0xffff classid 2:3
```

Exercise 3.1 Priority Scheduling

1. Create a token bucket filter on interface ifb0, following the steps of Exercise 1.1.

2. Add a priority scheduler with three priority levels with the command shown above.

3. Create three traffic flows with iperf3, which sent traffic as follows:

    - Flow 1 (class 2:3, low priority): Sends to port 6666 at rate 8 Mbps, starting at t=0 sec, stopping at t=60 sec;

    - Flow 2 (class 2:2, medium priority): Sends to port 5555 at rate 4 Mbps, starting at t=10 sec, stopping at t=50 sec;

    - Flow 3 (class 2:1, high priority): Sends to port 4444 at rate 2 Mbps, starting at t=20 sec, stopping at t=40 sec.

    You can create these flows quickly with the startiperf.sh script. Follow the steps in Exercise 1.2, to create a file that creates the above flows. Give the file the name prio-experiment.sh.

    ```
    ./startiperf.sh 4444 60 2 0 &
    ./startiperf.sh 5555 40 6 10 &
    ./startiperf.sh 3333 60 3 20 &
    ./startiperf.sh 2222 60 1 30 &
    ```

4. Specify the filters as given above, which associate:

    - Port 4444 with class 2:1 (high priority);

    - Port 5555 with class 2:2 (medium priority);

    - Port 6666 with class 2:3 (low priority).

Exercise 1.2 Measurement Experiments

With the configuration from the previous exercise, repeat the measurement of iperf3 transmissions from Exercise 4.2 of Lab S1. Note that without configuring a particular scheduling algorithm, the queueing discipline is FIFO.

7. Start the bokeh server

```
$ bokeh serve serverbokeh.py
```

8. Start a web browser, e.g., Firefox, and type the following URL:

   http://localhost:5006/serverbokeh

9. In the home directory, write the following four lines into a file with name myexperiment.sh:

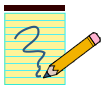
```
./startiperf.sh 5555 60 10 0 &
./startiperf.sh 6666 60 6 10 &
./startiperf.sh 3333 60 3 20 &
./startiperf.sh 2222 60 1 30 &
```

10. Run the script with the command

```
$ bash myexperiment.sh
```

11. Refresh/Reload the above URL in the browser.

12. When the iperf3 clients are completed, take a screen snapshot of the visualization shown in the browser, and save it to a file.



For the lab report:

4. Turn in the saved screen shot, with a brief description.

5. For the time period where all iperf3 clients are transmitting, compare the data rates of the various iperf3 transmissions, with those from Exercise 4.1 of Lab S1.

6. For the measurements in this exercise (Exercise 1.2), relate the transmission rate of an iperf3 client at the token bucket filter to the rate at which an iperf3 client is transmitting.

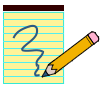
Part 3. Deficit Round Robin (DRR)

We will measure scheduling algorithms on a single Unix system. To do this, we use the loopback interface (which has domain name "localhost"). All transmissions to the loopback interface are returned to the local system.

Exercise 6.3 DRR (Deficit Round Robin)

Repeat the previous experiment with the DRR scheduler:

- Use DRR scheduling;
- Use a token bucket filter (see Part 1) to limit the maximum rate of the scheduler to 10 Mbps;
- You need to define a class as well as filter for each flow.
  Note: If a class has minor number "1", the script "startiperf.sh" sets the port number for that class to "10001". If a class has minor number "33", the script "startiperf.sh" sets the port number for that class to "10033".
- Generate three UDP traffic flows with iperf3 which sent traffic as follows:
  - Flow 1: Sends at rate 2 Mbps, starts at t=0 sec, stops at t=60 sec;
  - Flow 2: Sends at rate 4 Mbps, starts at t=10 sec, stops at t=50 sec;
  - Flow 3: Sends at rate 8 Mbps, starts at t=20 sec, stops at t=40 sec.
  - Save a screen capture of the complete experiment visualization to a file.
  - When all flows are active (20 sec ≤ t ≤ 40 sec), determine the bandwidth obtained by each flow. Relate the bandwidth obtained by a flow to the rate at which a flow is sending.

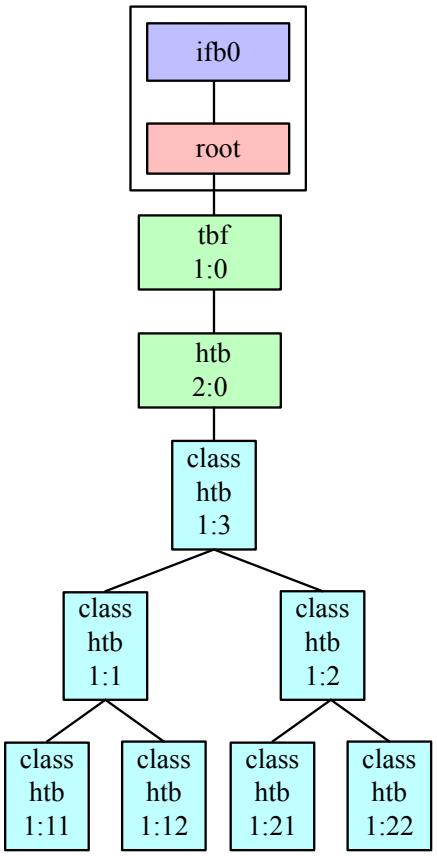


For the lab report:

- Turn in the saved screen shot, with a brief description.

Part 4. Hierarchical Token Bucket (HTB)

The final example creates a hierarchical token bucket with classes as shown here:



Exercise 6.1 HTB

Setup the HTB scheduler with parameters for the classes as follows:

Class Id	Assured Rate (Mbps)	Ceiling Rate (Mbps)	Burst Size (Bytes)
1:3	10	10	4000
1:1	5	10	2000
1:2	5	10	2000
1:11	2.5	10	1000
1:12	2.5	10	1000
1:21	1	10	1000
1:22	4	10	1000

Note: A configuration script for this setup is given at:

https://www.comm.utoronto.ca/~jorg/teaching/ece1545/labs/HTB-example.sh

Your task is to setup (5) different experiments to evaluate whether HTB provides a hierarchical fair bandwidth allocation. Flows may start and stop at different times.

- Include a scenario, where the bandwidth demand of each flow exceeds the assured rate.

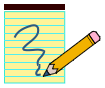
- Include a scenario, where 1:12 has a bandwidth demand of 10 Mbps, and no other flow is transmitting.

- Include a scenario, where the bandwidth demands are:

1:21	10 Mbps	start time: 0 sec
1:22	1 Mbps	start time: 0 sec
1:12	5 Mbps	start time: 20 sec
1:11	does not send	

- Include two other scenarios of your own design.

For each scenario, save a screen capture of the complete experiment visualization to a file.



For the lab report:

- Turn in the saved screen shot, with a brief description.

- Provide discussions of the expected versus the observed outcomes.

Appendix

HTB-example.sh

```
# Editing Loopback Interface
PATH="$PATH:/usr/bin:/sbin:/bin"
INT=ifb0
echo Configuring interface $INT

# Delete current Interface attached to `root`
sudo tc qdisc del dev $INT root

# Add `tbf` with major id 2:
sudo tc qdisc add dev $INT root handle 2: tbf rate 10Mbit burst 10k limit 1M

# Add htb `qdisc` with classid 1:, to class 2: (tbf)
sudo tc qdisc add dev $INT parent 2: handle 1: htb

# Add class 1:3 to qdisc 1: with rate/ceil/burst parameter
sudo tc class add dev $INT parent 1: classid 1:3 htb rate 10Mbit ceil 10Mbit burst 4000

sudo tc class add dev $INT parent 1:3 classid 1:1 htb rate 5Mbit ceil 10Mbit burst 2000
sudo tc class add dev $INT parent 1:1 classid 1:11 htb rate 2.5Mbit ceil 10Mbit burst 1000
sudo tc class add dev $INT parent 1:1 classid 1:12 htb rate 2.5Mbit ceil 10Mbit burst 1000

sudo tc class add dev $INT parent 1:3 classid 1:2 htb rate 5Mbit ceil 10Mbit burst 2000
sudo tc class add dev $INT parent 1:2 classid 1:21 htb rate 1Mbit ceil 10Mbit burst 1000
sudo tc class add dev $INT parent 1:2 classid 1:22 htb rate 4Mbit ceil 10MBit burst 1000

sudo tc filter add dev $INT parent 1: protocol arp u32 match u32 0 0 flowid 1:11 # arp
sudo tc filter add dev $INT parent 1: protocol ip u32 match ip dport 10000 0xffff classid 1:11
sudo tc filter add dev $INT parent 1: protocol ip u32 match ip dport 10001 0xffff classid 1:12
sudo tc filter add dev $INT parent 1: protocol ip u32 match ip dport 10002 0xffff classid 1:21
sudo tc filter add dev $INT parent 1: protocol ip u32 match ip dport 10003 0xffff classid 1:22

# IDs are used to identify qdisc, class and filters
# ID consists of 2 parts; 16bit major number, and 16bit of minor number, written as major:minor.
# Special IDs:
# - root: 65535:65535 (all ones)
# - unspecified: 0:0 (all zeros)
#
# qdisc is identified by only major number (called `handle`)
# - 2: for tbf and 1: for htb above
#
# class id uses both major number and minor number
# - major number is the same as qdisc it belongs to
# - minor number (called classid) can be arbitrarily assigned, there is no rule regarding this.
# - For example
#   - 1:4 corresponds to a class in qdisc 1: with class id 4
#
# Current setting:
#                 *---------------*
#                 |interface: root|
#                 *---------------*
#                         |
#                   *-----------*
#                   |  tbf 2:   |
#                   *-----------*
#                         |
#     *-----------------------------------------*
#     |htb qdisc 1:    -------                  |
#     |                   |                     |
#     |               *---------*               |
#     |               |root(htb)|               |
#     |               |   1:3   |               |
#     |               *---------*               |
#     |                   |                     |
#     |           *--------*----------*          |
#     |           |                   |         |
#     |      *-------*           *-------*      |
#     |      |A      |           |B      |      |
#     |      |  1:1  |           |  1:2  |      |
#     |      *-------*           *-------*      |
#     |          |                   |          |
#     |     *----*----*         *----*----*     |
#     |     |         |         |         |     |
#     |  *-----*   *-----*   *-----*   *--_--*  |
#     |  |A1   |   |A2   |   |B1   |   |B2   |  |
#     |  | 1:11|   | 1:12|   | 1:21|   | 1:22|  |
#     |  *-----*   *-----*   *-----*   *-----*  |
#     *-----------------------------------------*
```