POLITECNICO DI TORINO

Lab #2 on Traffic Scheduling and Buffer Management

"Computer network design and control" module of
Communication and network systems

Academic year 2023/24

Andrea Bianco, Paolo Giaccone,
Alessandro Cornacchia, Iman Lotfimahyari, Matteo Sacchetto

Version: December 1, 2023

1 Introduction

The lab deals with fairness issues and QoS support through scheduling and buffer management algorithms. The activities carried on in this lab are based on Mininet and the Linux Traffic Control (`tc`) tool, which permits defining and configuring the QoS algorithms on the virtual switch interfaces.

1.1 Before starting

To start this lab you need to perform the following steps:

1. Navigate to Crownlabs and start the ”Lab1” VM (If you do not remember how to use Crownlabs, please refer to the Introduction section of the Lab 1)
2. Once you are connected to the VM you need to download all the necessary material needed to run this lab. So, within the VM, open a web browser and either
   - navigate to `https://www.telematica.polito.it/course/computer-network-design-and-control`
   - Download the ”lab2.zip” archive
3. or directly download from `https://www.telematica.polito.it/app/uploads/2023/11/lab2.zip`
4. Open the File manager and navigate to the ”Downloads” folder. Here you should find the ”lab2.zip” archive. Right-click on it and select the ”Extract to” option. In the pop-up window, select ”Desktop” and then click on the ”Extract” button you can find at the bottom right of such window. You will find a `lab2` folder with multiple subfolders dedicated to the different scenarios we will explore as well as the `install.sh` which contains the additional dependencies needed for this lab.
5. Close the File manager, then open a terminal on the Desktop (Right-click on the Desktop and select ”Open terminal here”)
6. On the terminal type `bash install.sh` and press Enter.
7. Once the script finishes, you are all set and ready to start the Lab.

2 Scheduling algorithms

The objective of this part of the laboratory is to compare the performance of different scheduling algorithms, namely FIFO, Round-Robin (RR), Weighted Round-Robin (WRR), and Strict-Priority on different topologies.

2.1 Two-flows scenario

We focus on a simple topology with two traffic flows, for variable input loads.

The topology is depicted in Fig. 1 and comprises two hosts `h1` and `h2` acting as traffic generators and two hosts `h3` and `h4` acting as traffic destinations. The hosts are connected through a sequence of two switches `s1` and `s2`. All the links are running at 2Mbps, except for the link (denoted as “bottleneck”) between the two switches running at 1Mbps. The delays of all the links are set equal to 10ms.

In the folder `lab2/linear_topology` you find the scripts to configure the topology and run different scheduling algorithms on the output interface connecting `s1` to `s2`:

- `fifo.py` implements FIFO queueing;
- `rr.py` implements a Round-Robin (RR) policy;
- `wrr.py` implements a Weighted Round-Robin (WRR) with weights 4 (for the traffic generated by `h1`) and 1 (for the traffic generated by `h2`);
- `strict-priority.py` implements a strict priority (SP) scheduler, with the traffic generated by `h1` at the highest priority.

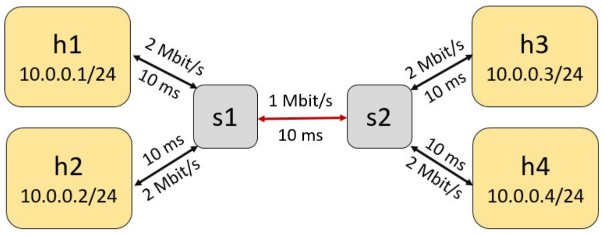

Figure 1: The scenario with two flows.

All the commands reported below need to be run with the current working directory set to `/home/netlab/Desktop`. If you are not sure which directory your terminal is currently in, you can check it by running `pwd`. Otherwise, simply run `cd /home/netlab/Desktop` to navigate to the correct location.

2.1.1 FIFO scheduler

Run the first scenario with a FIFO scheduler, using the command:

```
sudo python -m lab2.linear_topology.fifo
```

Open the terminal of all four hosts by typing, within the mininet window:

```
xterm h1 h2 h3 h4
```

Note that the window label, e.g., `Node:h1`, permits the identification of the host whose terminal is running on the window. Now start the `iperf3` servers at the destination hosts by typing on both terminals of `h3` and `h4`:

```
iperf3 -s
```

The command should print that port 5201 is used to receive the traffic.

To begin with, we consider a single flow scenario. Start UDP traffic from `h1` to `h3` with 100kbit/s load by typing on the terminal of `h1`:

```
iperf3 -c 10.0.0.3 -u -t 100 -b 100k
```

Note that you can stop the source when the results stabilize by typing `CTRL-C`.

Explain below what is the meaning of all of the above options (`-s`, `-c`, `-u`, `-t`, `-b`). Type `man iperf3` to get this info.

Observe the measured throughput (denoted as "Bitrate") and losses measured in `h3` (not in `h1`) and fill the following table by repeating the experiment for different input loads, as reported in the following table. Note that the losses are observed, when occurring, after 30-60 seconds of the experiment, due to the internal buffering. When the results stabilize and the losses are observed (if any), you can stop the experiment. Compute also the theoretical throughput, eventually showing the way the computation is done.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Loss probability	Theo. Thr.
1	h1	100			
2	h1	500			
3	h1	900			
4	h1	1200			
5	h1	1500			

Are the results as expected? Why?

Now run a scenario with 2 UDP flows (from `h1` to `h3`, and from `h2` to `h4`) for all the combinations of input loads shown in the table.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Loss probability	Theo. Thr. [kbit/s]
6	h1	100			
	h2	200			
7	h1	400			
	h2	800			
8	h1	400			
	h2	1200			
9	h1	600			
	h2	1200			

Are the results as expected? Why? In summary, what is the effect of the FIFO policy?

2.1.2 Round robin (RR)

Start the scenario with

```
sudo python -m lab2.linear_topology.rr
```

Then open the terminal of all four hosts. Start the two UDP traffic flows, as in the previous scenario, and fill the following table. From now on, we will not consider the loss probability, since it may require some time to observe it, due to the internal buffering. The results should converge very fast.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Theo. Throughput [kbit/s]
10	h1	100		
	h2	200		
11	h1	400		
	h2	800		
12	h1	400		
	h2	1200		
13	h1	600		
	h2	1200		

Are the results as expected? Why? In summary, what is the effect of the RR scheduling policy?

2.1.3 Weighted round robin (WRR)

Start the scenario with the command

```
sudo python -m lab2.linear_topology.wrr
```

Then open the terminal of all four hosts. Start the two UDP traffic flows, as in the previous scenario, and fill in the following table. Recall that the weights are 4:1, for the two flows generated by `h1` and `h2`, respectively.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Theo. Throughput [kbit/s]
14	h1	100		
	h2	200		
15	h1	400		
	h2	800		
16	h1	600		
	h2	800		
17	h1	800		
	h2	800		
18	h1	1000		
	h2	800		

Are the results as expected? Why? In summary, what is the effect of the WRR scheduling policy?

2.1.4 Strict priority (SP)

Start the scenario with

```
sudo python -m lab2.linear_topology.strict_priority
```

Then open the terminal of all four hosts. Start the two UDP traffic flows, as in the previous scenario, and fill in the following table. As a reminder, the traffic generated at `h1` is at highest priority.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Theo. Throughput [kbit/s]
19	h1	100		
	h2	200		
20	h1	400		
	h2	800		
21	h1	600		
	h2	800		
22	h1	800		
	h2	800		
23	h1	1200		
	h2	800		

Are the results as expected? Why? In summary, what is the effect of the SP scheduling policy?

2.1.5 Max-min fairness

In the scenario of Fig. 1, compute the max-min fair allocation for the two flows.

Flow	Max-min fair rate allocation [kbit/s]
h1-h3	
h2-h4	

Now list the experiment number for the throughput results that are compatible with a max-min fair allocation:

Which scheduling policy (FIFO, RR, WRR, SP) is compatible with a max-min fair allocation when the bottleneck link is overloaded?

2.2 Multiple flows scenario

The objective of this step is to compare the performance of different scheduling algorithms, namely FIFO, Round-Robin (RR), Weighted Round-Robin (WRR), and Strict-Priority, with multiple flows on two topologies. For this section of the lab, you will be running the program `run.py` passing a configuration file describing the topology, scheduler, and flows to simulate.

Look at the example configuration file `linear5_underload_fifo.yaml` you can find in the folder `lab2/configs/` and familiarize yourself the main options, i.e.:

- the `experiment_name` (can be anything you like);
- the `topology` to use (either `linear_topology_5` or `mesh_topology_5`);
- the `scheduler` to use, one of:
  - `fifo`
  - `rr`
  - `wrr`
  - `strict_priority`
- the `rate` of each of the five flows.

You will have to create a configuration file for each experiment. You can copy the above mentioned example file, remember though to change the experiment name, otherwise output files will be overwritten. You can keep the default values of all other parameters for now.

You run the experiment through the `run.py` program, e.g.,

```
sudo python -m lab2.run lab2/configs/linear5_underload_fifo.yaml
```

All experiments last for the duration set in the config file (suggested at least 60 seconds). The program generates on the desktop a folder with the experiment name in which you will find:

- a `.csv` file containing the input and output rates of all interfaces;

- a `.txt` file for each server, containing the `iperf3` output;
- a `.png` plot showing the *offered load* of each flow;
- a `.png` plot showing the *throughput* of each flow.

For the remainder of this lab, you will mostly look at the throughput graph.

2.2.1 Multiflow single bottleneck network

In this scenario, the topology is made of two switches, with 5 hosts connected to each of them, as shown in Fig. 2. All link rates are fixed at **2 Mbit/s** except the link between the switches which is fixed at **1 Mbit/s** to be a possible bottleneck.

To use this topology, specify `linear_topology_5` as the `topology` of the configuration file.

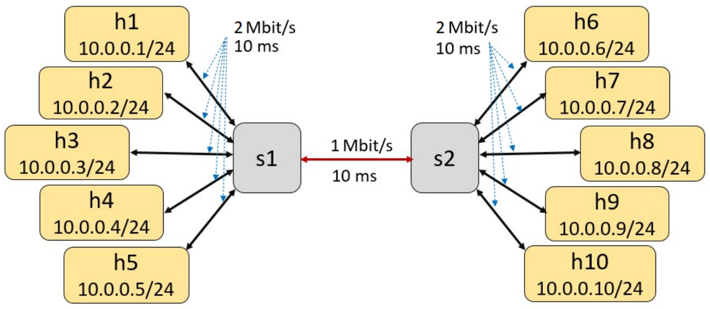

Figure 2: Multiflow single bottleneck network topology.

You must compare the effect of using different schedulers in the switches while transmitting with i) different offered loads and ii) different starting times of traffic generation..

2.2.1.1 Under-load First, we run the experiments in low traffic conditions, where flows are unlikely to experience losses. Fix the rates of the five flows to (respectively) `100, 100, 100, 200, 350 kbit/s`, and fill the table below with the throughput reported in the plot.

FIFO scheduler - Underload		
Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Repeat the experiments for the other three schedulers: strict priority, round robin, weighted round robin, and report the results. Weights for the WRR scheduler, are set respectively to 1,1,1,2,5. For strict priority, priorities are set respectively 1, 2, 2, 2, 3 (the lower number the higher priority)

Strict priority - Underload

Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Round robin - Underload

Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Weighted round robin - Underload

Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Briefly discuss per-flow throughput and fairness properties comparing the various test cases, with the help of graphs.

2.2.1.2 Over-load We now move to overloaded conditions, where there might not be enough capacity to serve all flows. Change the configuration file and set the rates of the five flows from (respectively) `100, 100, 100, 200, 350 kbit/s` to `200, 200, 200, 300, 600 kbit/s`.

Fill the table below by extracting the info from the plots.

FIFO scheduler - Overload

Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Repeat the experiments for the other three schedulers: strict priority, round robin, weighted round robin, and report the results.

Strict priority - Overload

Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Round robin - Overload		
Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Weighted round robin - Overload		
Flow id	Input load	Throughput
1		
2		
3		
4		
5		

Briefly discuss per-flow throughput and fairness properties by comparing the various test cases, with the help of graphs.

2.2.2 Max-min fairness

In the scenario of Fig. 2, compute the max-min fair allocation for the two flows.

Flow	Max-min fair rate allocation [kbit/s]
h1-h6	
h2-h7	
h3-h8	
h4-h9	
h5-h10	

Now list the experiment number for the throughput results that are compatible with a max-min fair allocation:

Which scheduling policy (FIFO, RR, WRR, SP) is compatible with a max-min fair allocation when the bottleneck link is overloaded?

2.3 Multiflow mesh network.

For this step, you will use the topology in Fig. 3 with the depicted flows and routes. All the links between the switches (i.e., red links) are fixed at **10 Mbit/s**. To use this topology, specify `mesh_topology_5` as the `topology` in the configuration file.

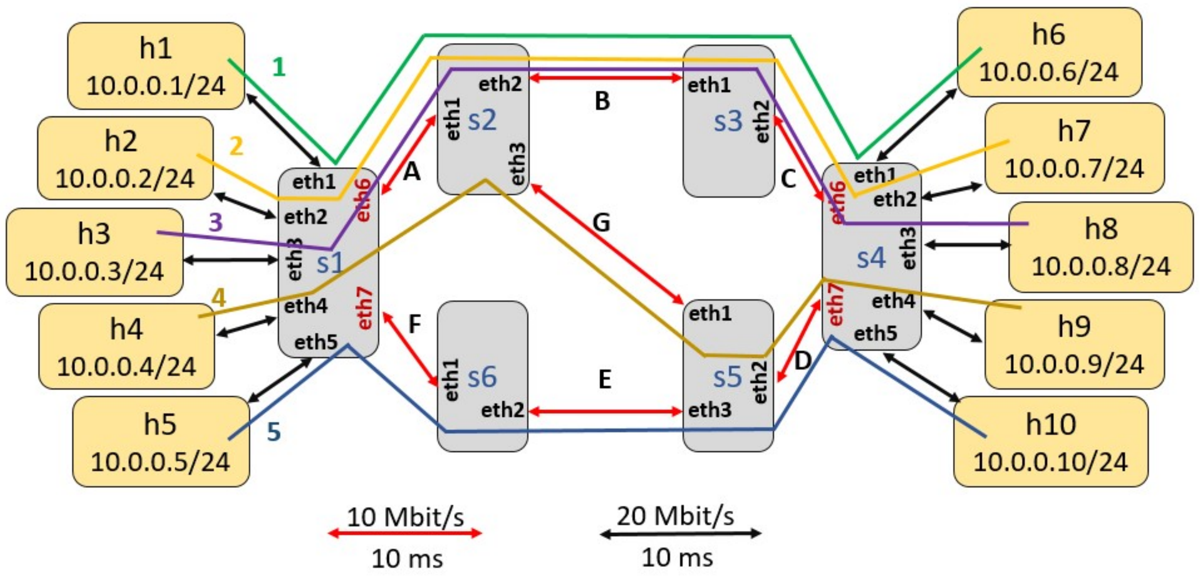

Figure 3: Multiflow mesh network topology.

2.3.1 Underload

Set the input rates of all flows to `1 Mbit/s` (you will specify that as `1M` in the file).

Report below and compare the results

Underload				
Flow id	Throughput		Losses [%]	
	FIFO	Round Robin	FIFO	Round Robin
1				
2				
3				
4				
5				

Would you expect to see any difference? Explain the results.

2.3.2 Overload

We now increase the rate of all flows to **10 Mbit/s**.

Before running the experiment, compute the expected rates that a **max-min fairness** algorithm would assign to the five flows and write the values in the table below.

Overload - Throughput			
Flow id	MAX MIN fair rate	FIFO rate	Round Robin rate
1			
2			
3			
4			
5			

Explain the results.

2.3.3 Rate limiting

For this step, repeat the step in Sec. 2.3.2 with limiting the link rates for flow 1 and flow 2 to **1 Mbit/s**. Before running the experiment, compute the expected rates that a **max-min fairness** algorithm would assign to the new five flows and write the values in the table below.

Rate limiting - Throughput			
Flow id	MAX MIN fair rate	FIFO rate	Round Robin rate
1			
2			
3			
4			
5			

Also, report in the table the throughput achieved by the FIFO and Round Robin schedulers. Do the results of the tests with the two different schedulers differ? Explain the results.