

POLITECNICO DI TORINO



**Politecnico
di Torino**

Lab #2 on Traffic Scheduling

“Computer network design and control” module of
Communication and network systems

Academic Year 2025/26

Andrea Bianco, Paolo Giaccone,
Alessandro Cornacchia, Matteo Sacchetto

Version: December 2, 2025

© 2025-26

Chapter 1

Laboratory #2

The aim of this lab is to experiment with fairness issues and Quality of Service (QoS) support through scheduling algorithms in a Mininet SDN network emulator running on a Virtual Machine within CrownLabs. The Linux Traffic Control (`tc`) tool will be used to define and configure QoS algorithms on the virtual switch interfaces.

The lab's primary goal is to familiarize students with QoS techniques, their impact on network performance, and various scheduling and buffer management algorithms.

1.1 Starting the Lab

To start the lab, follow the same procedure outlined in Lab 1 (also provided below for convenience).

1. Navigate to the CrownLabs website: <https://crownlabs.polito.it/>
2. Click on the "Login @Polito" button.
3. On the login page, click on the "PoliTo SSO" button at the bottom of the form.
4. Log in using your PoliTo credentials (the same credentials used to access the "Portale della didattica").
5. Once logged in, you will be on the "Dashboard" tab. You should see the "Computer Network Design" workspace on the left side of the user interface (UI).
6. Click on that workspace to reveal a new section containing the VM called "Lab".
7. Click the "Create" button to instantiate a new VM. Once creation is complete, the "Connect" button will become active.
8. When the "Connect" button becomes active, click it. A new browser tab will open, connecting you to the VM desktop.

For detailed instructions on using CrownLabs, please refer to the full guide provided in Lab 1.

Before continuing the lab, open a terminal and run the following command:

```
xrdb -merge /home/netlab/Desktop/.Xresources
```

1.2 Scheduling Algorithms

The objective of this section is to compare the performance of various scheduling algorithms, including FIFO, Round-Robin (RR), Weighted Round-Robin (WRR), and Strict Priority (SP), across different network topologies.

1.2.1 Two-Flows Scenario

Let's start with a simple topology featuring two traffic flows under varying input loads.

The topology is shown in Fig. 1.1 and consists of two hosts, `h1` and `h2`, serving as traffic generators, and two hosts, `h3` and `h4`, acting as traffic destinations. The hosts are interconnected through two switches, `s1` and `s2`, in series. All links between hosts and switches operate at 2 Mbps with a delay of 40 ms. The link between the two switches (denoted as the “bottleneck”) runs at 1 Mbps with a delay of 10 ms.

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

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

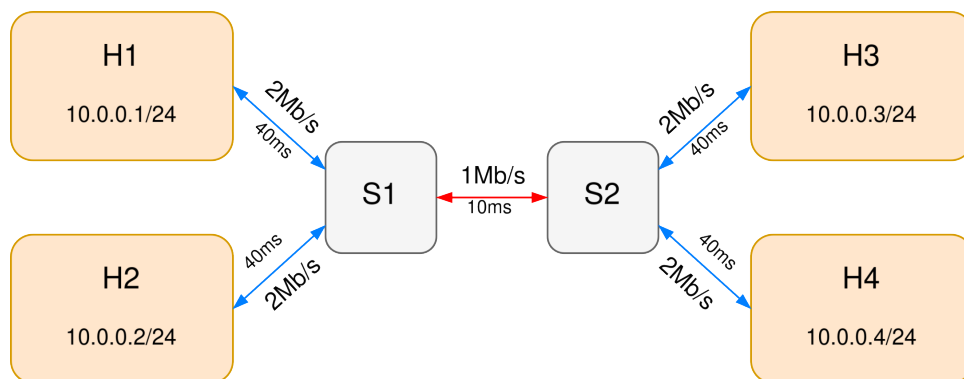


Figure 1.1: Linear topology with two flows

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

1.2.1.1 FIFO

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: The window label (e.g., `Node:h1`) helps identify the host associated with the terminal.

Now start the `iperf3` servers by typing in both terminals of `h3` and `h4`:

```
iperf3 -s
```

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

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

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

NOTE: You can stop the source when the results stabilize by pressing `CTRL+C`.

Explain below the meaning of all the above options (`-s`, `-c`, `-u`, `-t`, `-b`).

Type `man iperf3` to get this information.

Run the experiments for the various input loads reported in the table below. Recall that when the results stabilize and losses are observed (if any), you can stop the experiment. Before running each experiment, compute the expected throughput (take note of how the computation is done). When the run ends, observe the measured throughput (denoted as “Bitrate”) and losses measured in `h3` (not in `h1`).

Report the results in the table below:

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

Are the results as expected? If not, why?

Now run a scenario with 2 UDP flows (from h1 to h3, and from h2 to h4) for all combinations of input loads shown in the table below. Before running each experiment, compute the expected throughput and report it below.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Loss probability	Exp. 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? If not, why?

In summary, what are the most evident effects of the FIFO policy?

1.2.1.2 Round-Robin (RR)

Load the second scenario with two flows and an RR scheduler using the command:

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

Open a terminal for each of the four hosts. Before running each experiment, compute the expected throughput and report it below. Start the two UDP traffic flows as shown in the previous scenario and fill in the following table. From now on, we will not consider the loss probability.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Expected 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? If not, why?

In summary, what are the most evident effects of the RR policy?

1.2.1.3 Weighted Round-Robin (WRR)

Load this scenario with the following command:

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

Then, open a terminal for each of the four hosts.

Before running each experiment, compute the expected throughput and report it below. Start the two UDP traffic flows as described in the previous scenario, and complete the table below.

Recall that the weights for the two flows generated by h1 and h2 are set at 4:1, respectively.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Expected 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? If not, why?

In summary, what are the most evident effects of the WRR policy?

1.2.1.4 Strict Priority (SP)

Load this scenario with the following command:

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

Open a terminal for each of the four hosts. Before running each experiment, compute the expected throughput and report it below. Start the two UDP traffic flows as shown in the previous scenario and fill in the following table.

Remember that the traffic generated by h1 is assigned the highest priority.

Exp.	Host	Input load [kbit/s]	Throughput [kbit/s]	Expected 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? If not, why?

In summary, what are the most evident effects of the SP policy?

1.2.1.5 Max-min fairness

In the scenario of Fig. 1.1, 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 in which the throughput at the receiver is the same as the one computed according to the max-min fair allocation:

Which scheduling policy (FIFO, RR, WRR, SP) provides, at the receivers, the same rate as the max-min fair allocation when the bottleneck link is overloaded?

Is there any difference among schedulers in underload? Why?

1.2.2 Multiple Flows Scenarios

The objective of this section is to compare the performance of different scheduling algorithms, namely FIFO, Round-Robin (RR), Weighted Round-Robin (WRR), and Strict Priority (SP), with multiple flows across two topologies: a single bottleneck and a meshed topology.

For this part of the lab, you will run the script `run.py`, providing a configuration file that defines the topology, scheduler, and flows to simulate. Look at the example configuration file `linear5_underload_fifo.yml` in the `lab2/configs/` folder and familiarize yourself with the main options available:

- `experiment_name`: Can be anything you like. This is the name of the output directory where the results will be stored.
- `topology`: The topology to use:
 - `linear_topology`
 - `linear_topology_5`
 - `mesh_topology_5`
- `scheduler`: The scheduler to use:
 - `fifo`
 - `rr`
 - `wrr`
 - `strict_priority`
- `flows`: The various flows to generate. For each flow, the most important options are:
 - `rate`: The rate to generate.
 - `start_time`: The starting time at which the flow should start generating traffic.

You will have to create a configuration file for each experiment. It is recommended to make multiple copies of the above-mentioned example file.

NOTE: Remember to change the experiment name; otherwise, output files will be overwritten. You can keep the default values of all other parameters for the moment.

To run the experiment through the `run.py` program, use the following command:

```
sudo python -m lab2.run lab2/configs/<config_file_name>
```

where `<config_file_name>` is a placeholder that you need to replace with the file name of the experiment you want to run. For example:

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

All experiments last for the duration set in the config file (suggested value: 60 seconds minimum).

The program generates a folder with the experiment name on the Desktop, which contains the following:

- 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 both the offered load and the throughput of each flow.

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

1.2.2.1 Single Bottleneck

In the first scenario, the topology consists of two switches, with 5 hosts connected to each of them, as shown in Fig. 1.2. All links between hosts and switches operate at 2 Mbps with a delay of 40 ms. The link between the two switches (denoted as the “bottleneck”) runs at 1 Mbps with a delay of 10 ms.

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

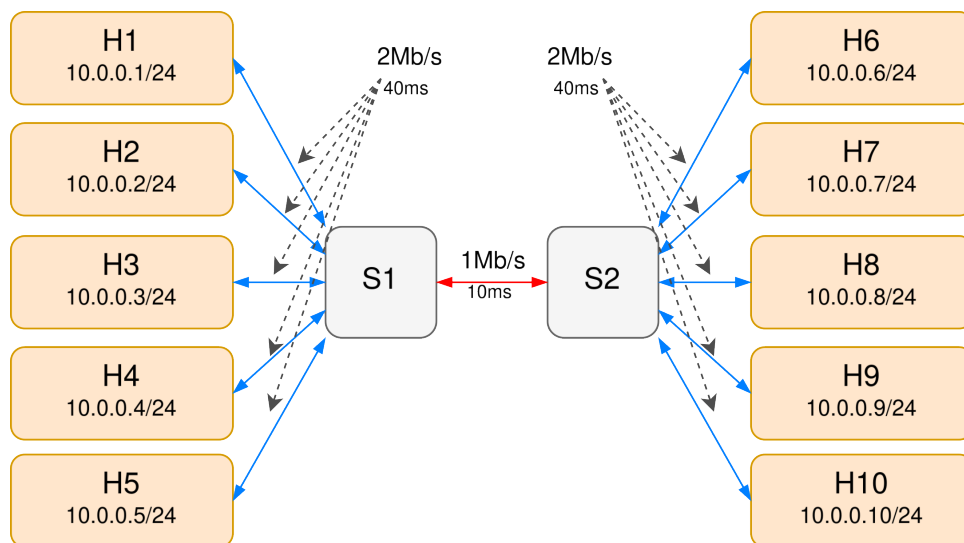


Figure 1.2: Linear topology with 5 flows

You must compare the effect of different schedulers with different offered loads.

Underload

First, run the experiments under low-traffic conditions. Set the rates of the five flows to 100, 100, 100, 200, 350 kbit/s, respectively, and complete the table below with the average throughput values obtained from the plot.

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

Repeat the experiment for the remaining three schedulers: Round-Robin, Weighted Round-Robin, and Strict Priority, and report the results. The weights for the WRR scheduler are set to 1, 1, 1, 2, and 5, respectively. For the Strict Priority scheduler, the priorities are set to 1, 2, 2, 2, and 3, where a lower number indicates higher priority.

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		

Strict Priority - 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 the plots.



Overload

In overloaded conditions, the available capacity is not sufficient to serve all flows with their input rates. Modify the configuration file and change the rates of the five flows from 100, 100, 100, 200, 350 kbit/s to 200, 200, 200, 300, 600 kbit/s, respectively.

Fill in the table below by extracting the information from the plots.

FIFO - Overload		
Flow ID	Input load	Throughput
1		
2		
3		
4		
5		

Repeat the experiments for the other three schedulers: Round-Robin, Weighted Round-Robin,

and Strict Priority, and report the results.

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		

Strict Priority - 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 the graphs.



Max-min fairness

In the scenario of Fig. 1.2, compute the max-min fair allocation for the five flows.

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

Now list the experiment number in which the throughput at the receiver is the same as the one computed according to the max-min fair allocation.

Which scheduling policy (FIFO, RR, WRR, SP) provides the same rate as the max-min fair allocation? Why?

1.2.2.2 Mesh topology

For this section, you will use the topology shown in Fig. 1.3 with the depicted flows and routes. All links between hosts and switches operate at **20 Mbps** with a delay of 40 ms. The links between all switches run at **10 Mbps** with a delay of 10 ms.

To use this topology, specify `mesh_topology_5` as the topology in the configuration file.

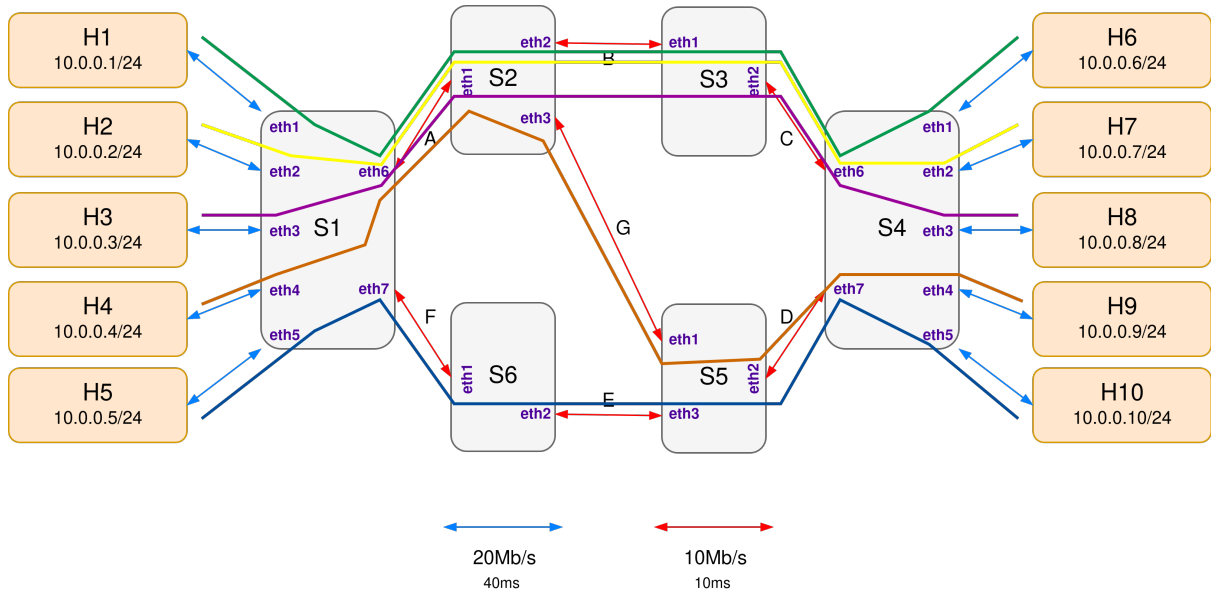


Figure 1.3: Mesh topology with 5 flows

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		Expected Losses [%]	
	FIFO	Round-Robin	FIFO	Round-Robin
1				
2				
3				
4				
5				

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

Overload

We now increase the rate of each flow to **10 Mbit/s**. Is some link in the topology overloaded? 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. Then, run the experiments and fill in the table.

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

Compare and explain the results.

1.2.2.3 Rate limiting

For this step, use the same configuration as in Sec. 1.2.2.2 but limit 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 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			

Run the experiments and report the throughput achieved by the FIFO and Round-Robin schedulers in the above table. Do the results of the tests with the two different schedulers differ?

Discuss and explain the results.

1.2.3 Two-flows scenario - Transient (Optional)

For this section, you will reuse the same topology used in Sec. 1.2.1 (reported below for your convenience).

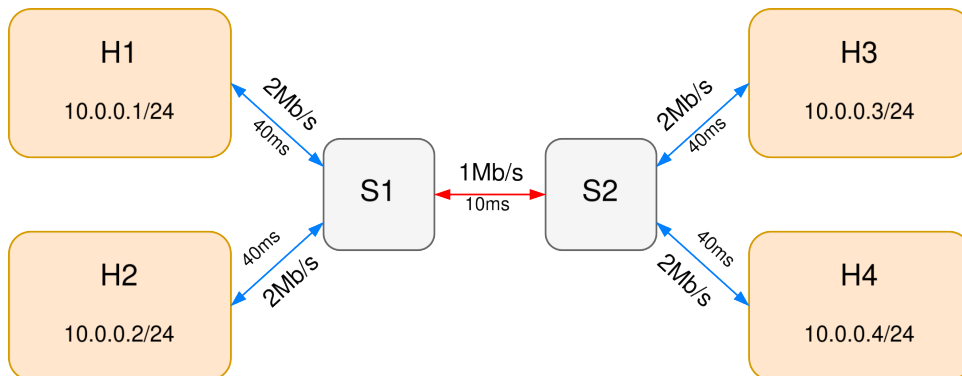


Figure 1.4: Linear topology with two flows

The script `run.py` provides a configuration file that defines the topology, scheduler, flows, and forwarding tables to simulate the scenario under study.

The objective of this experiment is to analyze how different scheduling algorithms behave in a transient scenario where one flow starts later than the other. This allows us to observe the impact of various schedulers on network performance during dynamic traffic changes.

Look at the example configuration file `linear_underload_fifo_transient.yml` in the `lab2/configs/` folder. To use this topology, specify `linear_topology` as the `topology` in the configuration file.

In each experiment, you must change both the `rate` and the `start_time` for each flow.

To run the experiment through the `run.py` program, use the following command:

```
sudo python -m lab2.run lab2/configs/<config_file_name>
```

where `<config_file_name>` is a placeholder that you need to replace with the file name of the experiment you want to run. For example:

```
sudo python -m lab2.run lab2/configs/linear_underload_fifo_transient.yml
```

All experiments last for the duration set in the config file (suggested simulation length: 30 seconds).

The program generates a folder with the experiment name on the `Desktop`, which contains the following:

- 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 both the offered load and the throughput of each flow.

For this exercise, you will mostly look at the throughput graph.

1.2.3.1 FIFO

In this first scenario, we will use the FIFO scheduler.

We report below the various configurations (i.e., input loads) you should run. Before running the experiments, write the expected throughput for the various flows. Then, complete the table by reporting the offered load and the average throughput achieved toward the end of the simulation, once the simulation reaches a stable condition:

Exp #	Host	Input load [kbit/s]	Start time [s]	Expected throughput [kbit/s]	Offered load [kbit/s]	Throughput [kbit/s]
1	h1	200	0			
	h2	400	10			
2	h1	400	0			
	h2	800	10			
3	h1	800	10			
	h2	400	0			
4	h1	800	0			
	h2	800	10			

Discuss and explain below the results and flow behavior.

1.2.3.2 Round-Robin

In this second scenario, we will use the Round-Robin scheduler.

We report below the various configurations (i.e., input loads) you should run. Before running the experiments, write the expected throughput for the various flows. Then, complete the table by reporting the offered load and the average throughput achieved toward the end of the simulation, once the simulation reaches a stable condition:

Exp #	Host	Input load [kbit/s]	Start time [s]	Expected throughput [kbit/s]	Offered load [kbit/s]	Throughput [kbit/s]
1	h1	200	0			
	h2	400	10			
2	h1	400	0			
	h2	800	10			
3	h1	800	10			
	h2	400	0			
4	h1	800	0			
	h2	800	10			

Discuss and explain below the results and flow behavior.

1.2.3.3 Weighted Round-Robin

In this third scenario, we will use the Weighted Round-Robin scheduler.

Remember that the weights for the two flows generated by h_1 and h_2 are set at 4:1, respectively. We report below the various configurations (i.e., input loads) you should run. Before running the experiments, write the expected throughput for the various flows. Then, complete the table by reporting the offered load and the average throughput achieved toward the end of the simulation, once the simulation reaches a stable condition:

Exp #	Host	Input load [kbit/s]	Start time [s]	Expected throughput [kbit/s]	Offered load [kbit/s]	Throughput [kbit/s]
1	h1	200	0			
	h2	400	10			
2	h1	400	0			
	h2	800	10			
3	h1	800	10			
	h2	400	0			
4	h1	800	0			
	h2	800	10			

Discuss and explain below the results and flow behavior.

1.2.3.4 Strict priority

In this last scenario, we will use the Strict Priority scheduler.

Remember that the traffic generated by h1 is assigned the highest priority. We report below the various configurations (i.e., input loads) you should run. Before running the experiments, write the expected throughput for the various flows. Then, complete the table by reporting the offered load and the average throughput achieved toward the end of the simulation, once the simulation reaches a stable condition:

Exp #	Host	Input load [kbit/s]	Start time [s]	Expected throughput [kbit/s]	Offered load [kbit/s]	Throughput [kbit/s]
1	h1	200	0			
	h2	400	10			
2	h1	400	0			
	h2	800	10			
3	h1	800	10			
	h2	400	0			
4	h1	800	0			
	h2	800	10			

Discuss and explain below the results and flow behaviour.