TCP/IP NETWORKING

LAB EXERCISES (TP) 5
CONGESTION CONTROL; TCP, UDP

December 1st, 2022
Deadline: December 14th, 2022 at 23.55 PM

Abstract

In this lab session, you will explore in a virtual environment the effect of the congestion control mechanism of TCP and compare with a situation without congestion control. You will see what types of fairness are achieved by this congestion control mechanism. You will observe that a congestion control mechanism is also essential to avoid congestion collapse.

0.1 ORGANIZATION OF THE LAB

In this document, you will read the lab instructions. You will solve Moodle quizzes, which will be graded. Carefully follow this document while doing the lab.

0.2 PRELIMINARY INFORMATION

1. On the virtual machine (VM), you need to download an archive that contains the programs for this lab. Download the file `lab5.zip` from Lab 5 folder from from Moodle, copy it in the shared folder and then uncompress it. The folder `lab5` contains three folders. The folder `lab5/scripts/` contains the python scripts that will be used to build the topologies of this lab for the experiments that will run in Mininet. The folders `lab5/tcp/` and `lab5/udp/` contain tcp and udp (correspondingly) clients and servers that we will use to create traffic over the topologies.
2. If needed, make the programs executable by going in the directory `lab5` (by typing `cd lab5`) and typing:

```
chmod +x tcp/tcpclient tcp/tcpserver udp/udpclient udp/udpserver
```

Note: All folders contain binary programs as well as the source code. Normally, the binaries should work in your VM and you should not need to recompile the source codes. If you want to (or need to) recompile them, you will probably need to install a few packages, including `gcc`, `make`, and `linux-module-headers`. Then, each program can be compiled by typing `make` in its own directory.

3. Last, we will need to check and/or modify the congestion control mechanism. On a Linux machine, you can test which congestion control mechanism is used by typing in a terminal:

```
cat /proc/sys/net/ipv4/tcp_congestion_control
```

In this lab, we will force TCP to use the CUBIC congestion control algorithm except differently requested. If the congestion control algorithm is not Cubic, you **should** change it to Cubic *until* the next reboot by typing in a terminal:

```
echo cubic >/proc/sys/net/ipv4/tcp_congestion_control
```

You should be in **sudo su** mode to execute this command. Similarly, when requested, you can set the RENO or the DCTCP congestion control algorithm, i.e.,

```
echo reno >/proc/sys/net/ipv4/tcp_congestion_control
```

```
echo dctcp >/proc/sys/net/ipv4/tcp_congestion_control
```

1 TCP VS UDP FLOWS

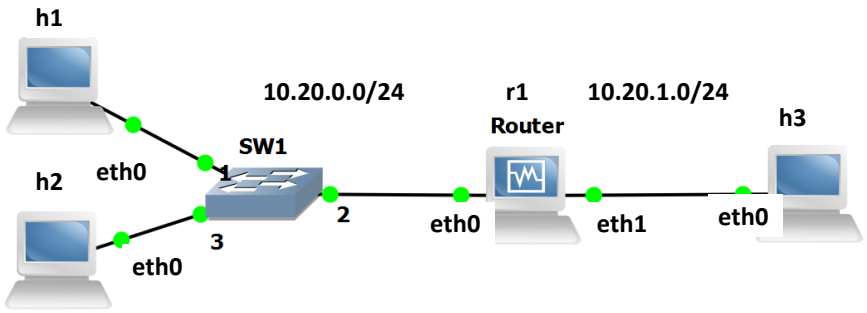


Figure 1: Initial configuration with 3 PCs and one router.

You are given the script `lab5/scripts/lab51_network.py` according to the following addressing scheme:

- The subnet of hosts h1, h2 and the router (r1) is 10.20.0.0/24. The addresses of h1, h2 and of the router are respectively 10.20.0.1, 10.20.0.2 and 10.20.0.10.
- The subnet of h3 and of the router is 10.20.1.0/24. The addresses of h3 and of the router are respectively 10.20.1.3 and 10.20.1.10.

Open a terminal in your VM and run the script `lab51_network.py`. Test the connectivity of the created topology with the *pingall* command.

Answer **Lab 5 - Part 1** on Moodle.

1.0.1 REMARKS ON THE PROGRAMS UDPCLIENT, TCPCLIENT, UDPSERVER AND TCPSERVER

The directories `lab5/tcp/` and `lab5/udp/` contain the executable and the source code of the programs.

•**For each UDP flow, you need one UDP client and one UDP server.** Explanation: each packet sent by a client contains its sequence number (the first packet contains the label "1", the second "2",...) and a lot of "0" to reach a size of 1000 bytes or 125 bytes. The loss percentage printed by the server is given by 100· (one minus the ratio between the number of packets received and the largest sequence number received). Because of this implementation, the loss percentage printed by the server is wrong if two clients talk to the same server.

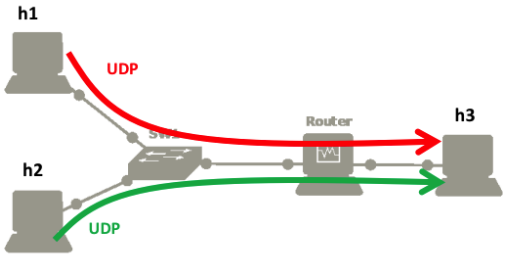
- **For TCP, one server can handle mutiple clients**. The server creates one thread per accepted connection.
- **Before each experiment, kill all clients (TCP and UDP).** You can do that by pressing "Control-C" in the terminal of the client. This will reset the average values printed by the clients. In theory, you can keep the server running but killing them and relaunching them will not harm.
- **The printed rates correspond to application data.** They count the amount of data that was transferred by the TCP/UDP client to the TCP/UDP server. They do not take into account headers.
- **For all experiments, you have to wait until the printed values stabilize**. This is particularly important for TCP. The rate at which TCP sends packets depends on the losses that occur at random. Thus, to obtain deterministic values, you should wait for the average rate to be stable.

- **Goodput**. The goodput of a flow is the rate of *application data* (i.e., useful data) that is successfully transmited. For the theoretical questions, you should take into account that the packets also contain header except from the application data.
- **Units**. In all your answers, indicate in which unit your result is expressed (Mbps, kbps, %, ...).
- **Queueing delay**. It is defined as the time (in the appropriate unit) that a packet waits stored in the queue of a router until it is forwarded.

1.1 Competing UDP flows, and TCP flows competing with UDP flows

Now we will explore what happens when two UDP flows are competing for the same bottleneck. **The router should have a capacity of 5 Mbps and is the bottleneck.** We consider the following scenarios:

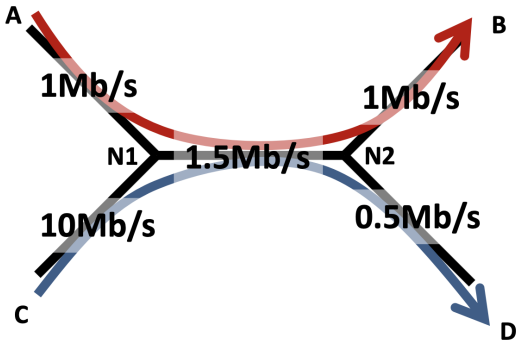
- Host h1 is streaming real-time data (*e.g.*, coming from a Phasor Measurement Unit (PMU)) at rate 1 Mbps to host h3 using UDP.
- Host h2 is streaming a video to host h3 using UDP. Depending on the quality, h2 sends at rate 0.5 Mbps, 4 Mbps or 9 Mbps.



Scenario	h1 (UDP)	h2 (UDP)
A1	1 Mbps	0.5 Mbps
A2	1 Mbps	4 Mbps
A3	1 Mbps	9 Mbps

Answer **Lab 5 - Part 2** on Moodle.

2 The importance of congestion control



In this section, we will explore why having a congestion control mechanism is necessary. The system that we want to emulate is composed of five links depicted on the left. The capacities of the links range from 0.5Mbps to 10Mbps. There are two flows in this network:

- one flow that goes from A to B (in red),
- one flow that goes from C to D (in blue).

We will show evidence of a phenomenon called *congestion collapse*: the more aggressive C is, the smaller the total goodput will be.

Answer **Lab 5 - Part 3** on Moodle.

3 TCP: FAIRNESS AND INFLUENCE OF RTT

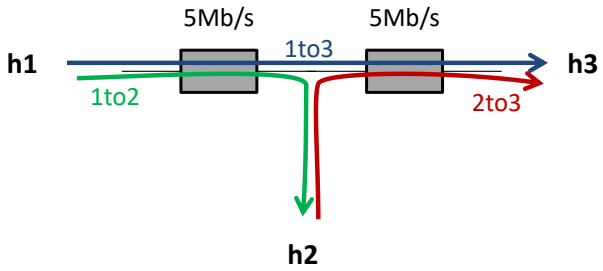
The congestion control algorithm of TCP guarantees that the network resources are shared among the different connections. In this part, we will explore how TCP CUBIC shares the bandwidth when one or multiple bottlenecks are present in the network. Note that for low RTT values CUBIC performs similarly to RENO, whereas this is not the case for higher RTT values as it is explained in the lecture notes. Also, we will perform comparisons between the bandwidth allocations of TCP CUBIC and of TCP RENO.

Fo Part 4 and Part 5, we will reuse the setting of Figure 3, which is created by the script `lab51_network.py`. The bandwidth of the router (r1-eth1) should be limited to **5 Mbps** (use the same configuration as in Section 1.4). Test the connectivity of your topology with the *pingall* command.

In this part in particular, it is important to wait until the printed goodputs stabilize. To speedup the convergence, it is **very** recommended to close all the unnecessary programs on your computer. Especially, you should close the programs that may perform things on background (such as web-browsers, Dropbox synchronization, other virtual machines, etc). In any case, you should wait around 3-5 minutes to see the stable results.
ECN and RED are already enabled in the script `lab51_network.py`, in order to reduce the impact of queueing delays on the RTT values.

Answer **Lab 5 - Part 4** on Moodle.

3.1 FAIRNESS OF TCP CONNECTIONS TRAVERSING MULTIPLE BOTTLENECKS



In this part, your goal is to study how the available bandwidth is shared when (i) one TCP connection traverses two queues (ii) each of the queues is also traversed by another TCP connection, as shown in the figure.

The notion of fairness is difficult. A rate allocation is always a trade-off between maximizing the total rates sent by the connection or trying to equalize the rates of all users. For example, in this scenario, the flow *1to3* uses twice more resources than the flows *1to2* and *2to3*. Thus, the bigger the traffic *1to3* is, the lower the aggregate goodput can be.

We now want to explore what is the allocation provided by TCP.

Answer **Lab 5 - Part 5** on Moodle.

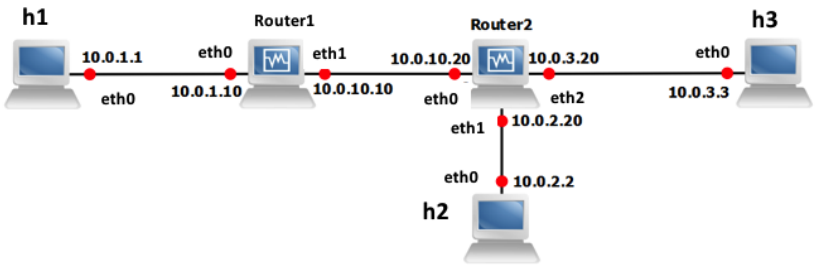


Figure 2: Fairness of TCP connections traversing multiple bottlenecks: wiring and addressing scheme.

4 Research Exercise: Study of the Use of ECN in TCP CUBIC and in DCTCP

ECN allows end-to-end notification of network congestion without dropping packets. Conventionally, TCP/IP networks signal congestion by dropping packets. When ECN is enabled together with some active queue management scheme such as RED, an ECN-aware router may set a mark, i.e., the Congestion Experienced (CE) bit, in the IP header instead of dropping a packet in order to signal impending congestion. The receiver of the packet echoes (by setting the ECN Echo flag) the congestion indication to the sender, which reduces its transmission rate as if it has detected a dropped packet and begins fast retransmit.

In this research exercise, we will study its use in Cubic and in DCTCP. In addition, we will study the effect of enabling ECN/RED on the RTT by comparing CUBIC without using ECN/RED and CUBIC with ECN/RED.

Answer **Lab 5 - Bonus** on Moodle.