TCP/IP NETWORKING

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

December 2nd, 2021
Deadline: December 15th, 2021 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

The topology that is used in this section is shown in Fig. 1.

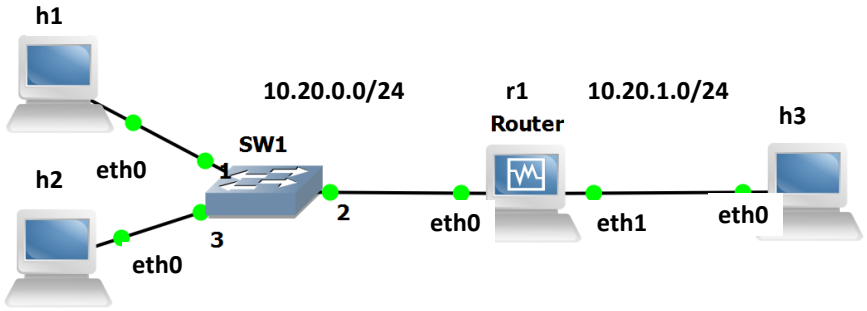


Figure 1: Initial configuration with 3 PCs and one router.

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.

1.1 TESTING THE CONNECTIVITY WITH BASIC UDP AND TCP CLIENTS/SERVERS

The directory `lab5/udp` contains two programs: `udpserver` and `udpclient`. Their usage is:

```
# ./udpserver PORT
# ./udpclient IP_SERVER PORT RATE
```

For the server, `PORT` is the port number on which the server listens. For the client `IP_SERVER` and `PORT` are the IP address and port of the machine to which the packets are sent and `RATE` is the rate at which the client sends data (in kilobits per seconds). The client sends packets of size 125 bytes if the rate is lower than 50kbps and of size 1000 bytes otherwise.

The output of the UDP client has the following format:

```
4.0s - sent:  503 pkts, 1000.0 kbits/s
5.0s - sent:  629 pkts, 1000.6 kbits/s
```

5.0 is the number of seconds since the launching time of the client, 629 is the total number of packets sent by the client and 1000.6 is the sending rate during the last second (in kilobits per second).

The output of the UDP server has the following format:

```
169.5s - received:  723/ sent:  741 pkts (loss 2.429%), 959.6 kbit/s
170.5s - received:  843/ sent:  867 pkts (loss 2.768%), 957.7 kbit/s
```

The values of the second line are explained as: 170.5 is the number of seconds since the launching of the server, 867 and 843 are the total number of packets sent by the client and received by the server, 2.768% is the percentage of packets that were lost and 957.7 is the rate at which packets were received during the last second. The latter value is defined as the goodput at the last second (see also the remarks in Section 1.1.1).

Q1/ Answer **Question 1 in Lab 5 - Part 1** on Moodle.

The directory `lab5/tcp` contains two programs: `tcpserver` and `tcpclient`. Their usage is similar to `udpserver` and `udpclient`, except that we do not specify a rate to the client: the client has an unlimited amount of data to send and uses TCP congestion control algorithm to control at which rate it sends the data to the server.

```
# ./tcpserver PORT
# ./tcpclient IP_SERVER PORT
```

The output of a TCP client looks like this:

```
6.3:  854.0 kbps avg ( 944.5[inst], 926.5[mov.avg]) cwnd  9 rtt 83.9ms
7.3:  862.4 kbps avg ( 914.6[inst], 925.3[mov.avg]) cwnd  9 rtt 86.8ms
```

The values of the second line are explained as: 7.3 is the number of seconds since the launching of the client, 862.4 is the average rate of the client, i.e., the total amount of data that was successfully transferred by the client divided by the total time (this is defined as goodput - average value - for the TCP), 914.6 is the instantaneous rate (approximately over the last second) and 925.3 is a moving average of this value. The value 9 is the congestion window of the TCP connection and 86.8 ms is the RTT measured by the TCP congestion control algorithm.

Q2/ Answer **Question 2 in Lab 5 - Part 1** on Moodle.

1.1.1 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.

1.2 ARTIFICIAL LIMITATION OF THE BANDWIDTH OF THE ROUTER

In order to produce experiments where the performance is limited by the network capacities, we will limit the bandwidth of some interfaces in the topology. To do so, in this section, we use the possibilities offered by the TCLink class in Mininet.

In the script $lab51\_network.py$, you can find the command

```
link_h3r1.intf1.config( bw=10, enable_red=True, enable_ecn=True )
```

The part $bw = 10$ of this command configures the bandwidth of the interface $r1 - eth1$ of the router to be 10 Mbps. **Modify** the command so that the bandwidth of the interface $r1 - eth1$ of the router is **5 Mbps**.

Note: You should exit Mininet, clean up the topology of the previous section and run the script `lab51_network.py` with the new bandwidth configuration. The script should involve the additions made in Section 1.1 so that the topology in Fig. 1 is connected and all hosts communicate.

1.2.1 UDP TEST

Assume a UDP server on host h3 and a UDP client on host h1. When the `RATE` at which the UDP client sends data is greater that 50 kbps, the client sends packets that contain 1000 bytes of data each.

Q3/ Answer **Question 3 in Lab 5 - Part 1** on Moodle.

Q4/ Answer **Question 4 in Lab 5 - Part 1** on Moodle.

Q5/ Answer **Question 5 in Lab 5 - Part 1** on Moodle.

Start a UDP server on host h3 that listens on port 1.

Q6/ Answer **Question 6 in Lab 5 - Part 1** on Moodle.

Q7/ Answer **Question 7 in Lab 5 - Part 1** on Moodle.

1.2.2 TCP TEST

Assume a TCP server on host h3 and a TCP client on host h1.

Q8/ Answer **Question 8 in Lab 5 - Part 1** on Moodle.

Q9/ Answer **Question 9 in Lab 5 - Part 1** on Moodle.

Start a TCP server on host h3 that listens on port 1.

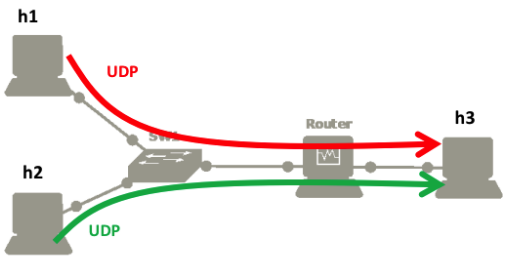
Q10/ Answer **Question 10 in Lab 5 - Part 1** on Moodle.

Q11/ Answer **Question 11 in Lab 5 - Part 1** on Moodle.

1.3 COMPETING 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

Before doing the measurements, we want to predict the amount of data that will be sent and received in the three scenarios (denoted A1, A2 and A3).

Q12/ Answer **Question 1 in Lab 5 - Part 2** on Moodle.

Q13/ Answer **Question 2 in Lab 5 - Part 2** on Moodle.

We now want to verify our analysis via emulation. For each scenario, start two UDP servers on h3 that listen on ports 1 and 2. Use the command `xterm h3` in mininet to open a new terminal for h3. Then, run a UDP client on h1 that sends data to h3 at 1 Mbps and a UDP client on h2 that sends data to h3 at rate 0.5, 4 or 9 Mbps.

Q14/ Answer **Question 3 in Lab 5 - Part 2** on Moodle.

Q15/ Answer **Question 4 in Lab 5 - Part 2** on Moodle.

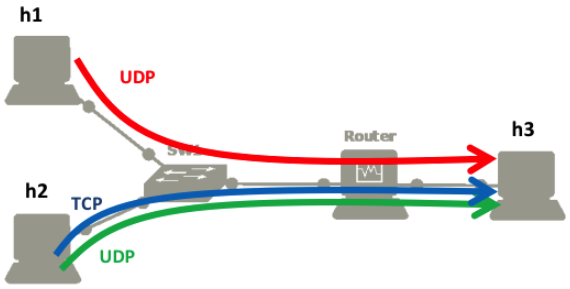
.

Q16/ Answer **Question 5 in Lab 5 - Part 2** on Moodle.

.

1.4 TCP FLOWS COMPETING WITH UDP FLOWS

We consider a similar scenario as in Section 1.3. Host h1 streams PMU data at rate 1 Mbps to host h3 and h2 streams video data to h3 at rate 0.5 Mbps, 4 Mbps or 9 Mbps. In addition to this traffic, host h2 is also using a TCP connection to send a software update to h3.



Scenario	h1 (UDP)	h2 (UDP)
B1	1 Mbps	0.5 Mbps
B2	1 Mbps	4 Mbps
B3	1 Mbps	9 Mbps

Q17/ Answer **Question 6 in Lab 5 - Part 2** on Moodle.

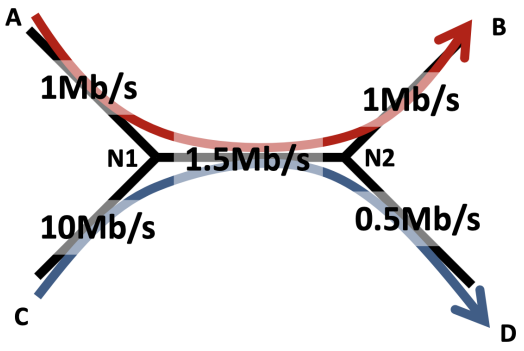
We will now perform an emulation. As in the previous case, the bandwidth is limited to 5Mbps (outgoing packets of r1-eth1) and we start two UDP servers on h3. Start also a TCP server on h3. Then, start the corresponding clients in hosts h1, h2. Use the commands `xterm h2`, `xterm h3` in mininet to open new terminals for h2, h3. **In experiments B2 and B3, take notes of the UDP loss rates and the TCP RTTs and congestion windows because you will need them in the next question.**

Q18/ Answer **Question 7 in Lab 5 - Part 2** on Moodle.

Q19/ Answer **Question 8 in Lab 5 - Part 2** on Moodle.

Q20/ Answer **Question 9 in 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.

2.1 THEORETICAL ANALYSIS

We first assume that there is no congestion control and that there exist two senders, A and C, which send data using UDP.

Q21/ Answer **Question 1 in Lab 5 - Part 3** on Moodle.

Q22/ Answer **Question 2 in Lab 5 - Part 3** on Moodle.

We now assume that sender A and sender C use a congestion control mechanism.

Q23/ Answer **Question 3 in Lab 5 - Part 3** on Moodle.

Q24/ Answer **Question 4 in Lab 5 - Part 3** on Moodle.

2.2 EXPERIMENTAL SETTING

We now want to verify these results in the virtual environment.

The script $lab52\_network.py$ creates a new topology according to Figure 2.

The addressing scheme is as follows:

- The addresses of h1, h2, h3 and h4 end with 1, 2, 3 and 4.
- The addresses of the routers 1, 2 end with 10, 20.

The bandwidth limits of the links are already set at the script. ECN and RED are also enabled at the routers. More information on the ECN can be found at the research exercise in Section 4.

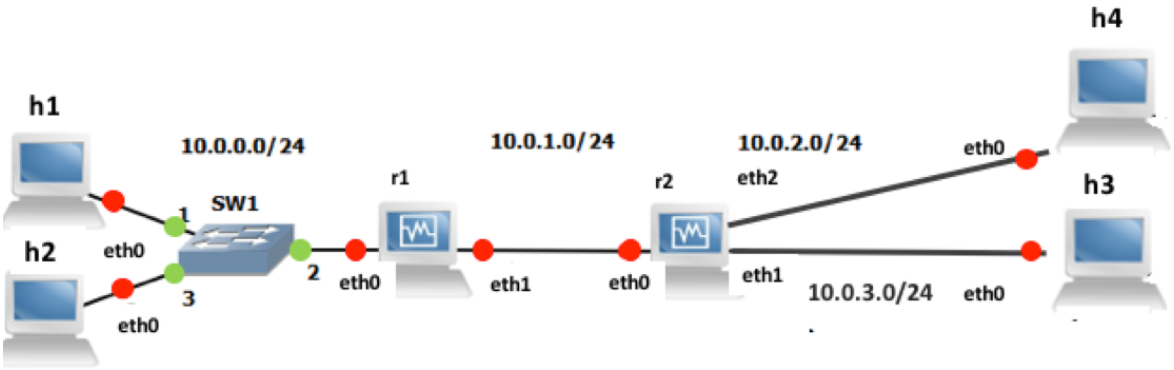


Figure 2: Setting for studying the congestion collapse.

2.2.1 UDP

Launch a UDP server on host h3 and another one on host h4. Launch two UDP clients, one on host h1 sending to host h4 with rate 1 Mbps and one on host h2 sending to host h3 with rate 10 Mbps.

Q25/ Answer **Question 5 in Lab 5 - Part 3** on Moodle.

Now, launch two UDP clients, one on host h1 and one on host h2, which send data according to the max-min fair allocation that you computed before.

Q26/ Answer **Question 6 in Lab 5 - Part 3** on Moodle.

2.2.2 TCP

Repeat the same process using TCP connections instead of UDP.

Q27/ Answer **Question 7 in Lab 5 - Part 3** on Moodle.

Q28/ Answer **Question 8 in 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.

For Sections 3.1 and 3.2, we will reuse the setting of Figure 1, 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 1.2). 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.

3.1 Adding delay to an interface

To obtain more realistic and reproducible experiments, we will add delay in the network. To do so, we will use the module `netem` of the software *traffic control* that exists in Linux. We can use the command `tc` to add a rule in order to delay packets on an interface (see http://www.linuxfoundation.org/collaborate/workgroups/networking/netem for more information about `tc` and `netem`).

For example, the following command adds 300 ms of delay to all packets going out of the interface `eth0` (one direction only, not applied to the packets coming in!!!):

```
# tc qdisc add dev eth0 root netem delay 300ms
```

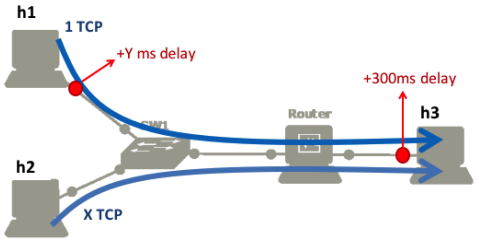
This rule can be changed to e.g., 400ms by typing `tc qdisc change dev eth0 root netem delay 400ms` or deleted by typing `tc qdisc del dev eth0 root`.

Q29/ Answer **Question 1 in Lab 5 - Part 4** on Moodle.

.

Remark: if you flush the ARP table (for example, by using the commands `ifconfig h3-eth0 down && ifconfig h3-eth0 up`) and you reconfigure `netem` to add 300 ms, the RTT of the first packet should be larger than the RTT of the second packet, because of the ARP request.

3.2 Fairness between TCP connections and delay



Scenario	X (# conn. on h2)	Delay Y
D1	3	0 ms
D2	1	500 ms
D3	1	1000 ms

TCP provides a fair sharing of the bandwidth at the flow level. Therefore, a machine that opens several TCP connections will obtain more bandwidth. To verify that, we will use the scenario D1:

- There is an additional delay of 300 ms on the interface h3-eth0 of host h3 (already added before) but none on hosts h1 or h2.
- Host h1 opens one TCP connection to h3 and host h2 opens three TCP connections to h3.

Q30/ Answer **Question 2 in Lab 5 - Part 4** on Moodle.

Q31/ Answer **Question 3 in Lab 5 - Part 4** on Moodle.

Start a TCP server on host h3. Run the 3 TCP clients on host h2 and one TCP client on h1. Wait until the rates stabilize.

Q32/ Answer **Question 4 in Lab 5 - Part 4** on Moodle.

Q33/ Answer **Question 5 in Lab 5 - Part 4** on Moodle.

We now explore scenarios D2 and D3, where both hosts h1 and h2 open 1 TCP connection to host h3. Assume that the RTT is 300ms for the connections coming from h2 and $(300 + Y)$ms for the connections coming from h1.

Q34/ Answer **Question 6 in Lab 5 - Part 4** on Moodle.

Q35/ Answer **Question 7 in Lab 5 - Part 4** on Moodle.

.

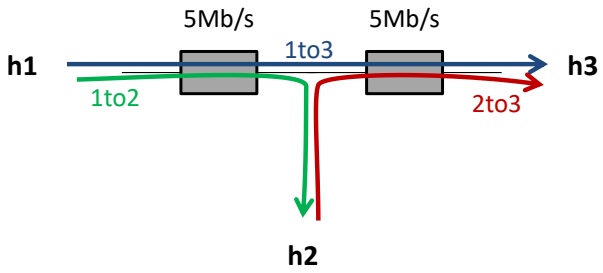
Q36/ Answer **Question 8 in Lab 5 - Part 4** on Moodle.

Each time you change the congestion control algorithm, you should exit and clean Mininet and restart your experiment. While Mininet is running, the congestion control algorithm is considered as being the same as the one set when Mininet was initiated the last time.

Q37/ Answer **Question 9 in Lab 5 - Part 4** on Moodle.

Q38/ Answer **Question 10 in Lab 5 - Part 4** on Moodle.

3.3 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.

3.3.1 THEORETICAL ANALYSIS

We first perform a theoretical analysis to compute two *fair* allocations corresponding to this network.

Q39/ Answer **Question 1 in Lab 5 - Part 5** on Moodle.

.

Q40/ Answer **Question 2 in Lab 5 - Part 5** on Moodle.

3.3.2 EXPERIMENTAL SETTING

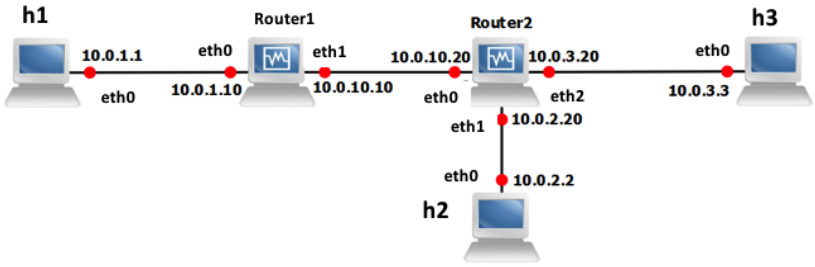
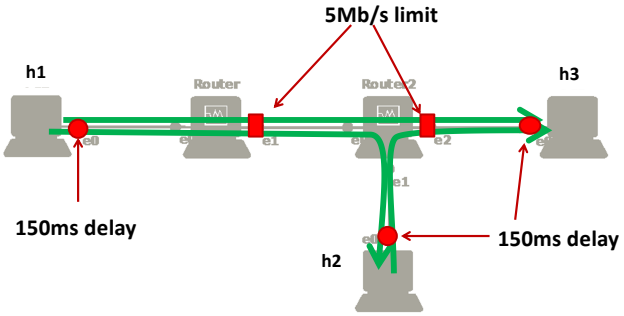


Figure 3: Fairness of TCP connections traversing multiple bottlenecks: wiring and addressing scheme.

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

At this moment, verify that the TCP algorithm in your virtual machine is TCP CUBIC.

Th script `lab53_network.py` constructs the topology according to Figure 3. **ECN and RED are already enabled in the script at routers 1 and 2**.



The bandwidth values of the interfaces r1-eth1 of Router1 and r2-eth2 of Router2 are limited to 5 Mbps.

As in Section 3.1 of this lab, **use `tc` to add 150 ms of delay** to the outgoing packets of the interfaces h1-eth0, h2-eth0 and h3-eth0.

Q41/ Answer **Question 3 in Lab 5 - Part 5** on Moodle.

Now, start one TCP server on h2 and one on h3. On host h1, open one TCP connection to h2 and one TCP connection to h3. On host h2, open one TCP connection to h3. Wait until the rates stabilize.

Q42/ Answer **Question 4 in Lab 5 - Part 5** on Moodle.

Q43/ Answer **Question 5 in Lab 5 - Part 5** on Moodle.

Q44/ Answer **Question 6 in Lab 5 - Part 5** on Moodle.

Q45/ Answer **Question 7 in Lab 5 - Part 5** on Moodle.

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. To achieve the above goals, we will use the topology of Fig. 1 which is defined in the script `lab51_network.py`.

First, set your TCP algorithm to TCP CUBIC.

We will now modify the script in order to limit the queue length of the router 1 by setting the max_queue_size parameter to the value of 1000 packets. In the script, you can find the command:

```
link_h3r1.intf1.config( bw=5, enable_red=True, enable_ecn=True )
```

You should add a parameter $max\_queue\_size = 1000$.

Next, disable ECN/RED at the interface eth1 of the Router in the script. by setting `enable_red=False, enable_ecn=False`. Run the script. Start a TCP server on h3 and two TCP clients, each one on hosts h1 and h2 sending traffic to h3. (Note: Start the TCP clients almost at the same time for faster convergence of the rates.)

Q46/ Answer **Question 1 in Lab 5 - Bonus** on Moodle.

Now, enable ECN/RED at the interface eth1 of the router by setting `enable_red=True, enable_ecn=True` in the script `lab51_network.py`. Repeat the previous experiment. Also, you should open wireshark in hosts h1 and h2 and in the router for capturing the traffic.

Q47/ Answer **Question 2 in Lab 5 - Bonus** on Moodle.

Q48/ Answer **Question 3 in Lab 5 - Bonus** on Moodle.

Q49/ Answer **Question 4 in Lab 5 - Bonus** on Moodle.

Q50/ Answer **Question 5 in Lab 5 - Bonus** on Moodle.

Q51/ Answer **Question 6 in Lab 5 - Bonus** on Moodle.

Set your TCP algorithm to TCP DCTCP. Enable ECN/RED at the interface eth1 of the Router. Repeat the experiment and open wireshark on r1-eth1, h1 and h2.

Q52/ Answer **Question 7 in Lab 5 - Bonus** on Moodle.

Q53/ Answer **Question 8 in Lab 5 - Bonus** on Moodle.