
TCP/IP NETWORKING

LAB EXERCISES (TP) 5

CONGESTION CONTROL; TCP, UDP

With Solutions

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 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.

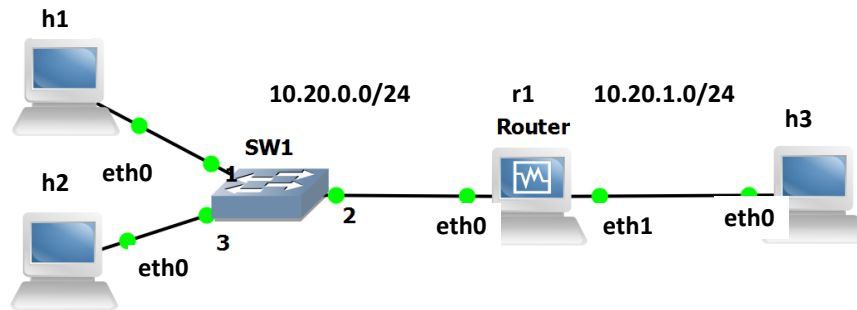


Figure 1: Initial configuration with 3 PCs and one router.

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).

Remark for the experiments that follow: If you run the same experiment multiple times the results may vary among different runs since Mininet is a network emulator. Thus, it is highly recommended that you run each experiment multiple times (e.g., 5) and provide the averaged values as the answer.

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

***Solution:** Start a UDP server on host h3 that listens on port 1. Launch a UDP client on host h1 that sends data to this server at rate 1 Mbps. The loss percentage observed on h3 is Loss: 0%. The goodput observed on h3 is 1 Mbps.*

The directory `lab5/tcp` contains two programs: `tcpserver` and `tcpclient`. Their usage is similar to `udpsrvr` 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.

***Solution:** Start a TCP server on host h3 that listens on port 1. Launch a TCP client on host h1 that sends data to server h3. The goodput is 9300 Kbps, close to the bandwidth of the interface 1 (`r1-eth1`) of the router.*

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 $100 \cdot (1 - \frac{\text{number of packets received}}{\text{largest sequence number received}})$. 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 multiple 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 transmitted. 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 than 50 kbps, the client sends packets that contain 1000 bytes of data each.

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

Solution: An Ethernet packet starts with a 7 bytes preamble, which is used for clock synchronization, and an 1 byte start frame delimiter (SFD), which is designed to break the bit pattern of the preamble and signal the start of the actual frame.

The Ethernet header length is 14 bytes.

The IP header length is 20 bytes.

The UDP header length is 8 bytes.

The data contains 1000 bytes.

After the payload there exists a frame check sequence (FCS), which is a 4-bytes cyclic redundancy check (CRC) that allows detection of corrupted data within the entire frame as received on the receiver side.

At the end there is an interpacket gap which represents the idle time between packets. After a packet has been sent, transmitters are required to transmit a minimum of 12-bytes of idle line state before transmitting the next packet.

In total the Ethernet frame is expected to have 1066 bytes.

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

Solution: Assume that one router has a bandwidth limit of 5 Mbps. In that situation, the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved is **4690 kbps**.

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

Solution: The maximum theoretical aggregate application throughput calculated in the previous question is equal to $5000 * \text{Application data size} / \text{Ethernet frame size}$.

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

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

Solution: Start a UDP client on host h1 that sends data at rate 4.5 Mbps. What are the loss percentage and the goodput observed on h3? The loss percentage observed on h3 is 0 %. The goodput observed on h3 is 4.5 Mbps.

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

Solution: Repeat the operation with a UDP client that sends at rate: 5 Mbps: Goodput: 4798 kbps, Loss: 4.03%

10 Mbps: Goodput: 4798 kbps, Loss: 52.03%

20 Mbps: Goodput: 4798 kbps, Loss: 76.43%

The results are **close** to the theoretical max goodput value.

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.

Solution: What is the size (in bytes) of Ethernet frames that you expect to be used for sending the data by the TCP connection? An Ethernet packet starts with a 7-bytes **preamble** and an 1-byte **start frame delimiter (SFD)**.

The **TCP header** length is 32 bytes.

The **IP header** length is 20 bytes.

The **Ethernet header** length is 14 bytes.

The length of the data depends on your **MSS**. Its size is **2896:500** bytes in Wireshark.

After the payload there exists a 4-bytes **frame check sequence (FCS)** and at the end there exists a **12-bytes interpacket gap**.

In total the Ethernet frame is expected to have **2986** bytes.

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

Solution: The router has a bandwidth limit of 5 Mbps. In that situation, when using TCP, the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved is **4850 kbps**.

Explanation: The size of the Ethernet frame is 2986 bytes, 2896 of which correspond to the application data. Therefore, the theoretical goodput is equal to $5 * 2896 / 2986 \text{ Mbps} = 5000 * 0.97 = 4850 \text{ kbps}$.

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

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

Solution: Launch a TCP client on host h1 that sends data to this server. The observed goodput of the connection is Answer **4782 kbps**. The results are **close** to the theoretically expected value.

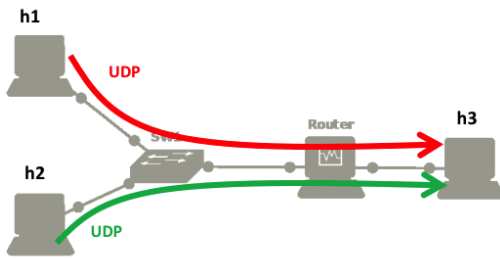
Q11/ Answer Question 11 in Lab 5 - Part 1 on Moodle.

Solution: The maximum theoretical aggregate application throughput calculated in the previous question is equal to: $5 * \text{MSS size} / \text{Ethernet frame size}$

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.

Solution: Based on theoretical analysis, give the general formula for the goodputs of hosts h1 in the scenarios A1, A2 and A3: Goodput UDP h1:

$\min(5 * \text{rate of h1} / \text{rate of h1} + \text{rate of h2} * \text{Application data size} / \text{Ethernet frame size}, \text{rate of h1})$

And the goodput of flow h2 in the scenarios A1, A2 and A3:

$\min(5 * \text{rate of h2} / \text{rate of h1} + \text{rate of h2} * \text{Application data size} / \text{Ethernet frame size}, \text{rate of h2})$.

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

Solution: Based on theoretical analysis done in previous question, what should be the theoretical numerical value for the goodputs and loss percentages of hosts h1 and h2 in the scenarios A1, A2 and A3.

Scenario A1: The goodput is 1 Mbps for host h1 and 0.5 Mbps for host h2. The loss percentage is 0% for host h1 and 0% for host h2.

Scenario A2: The goodput is 0.938 Mbps for host h1 and 3.752 Mbps for host h2. The loss percentage is 6.2% for host h1 and 6.2% for host h2.

Scenario A3: The goodput is 0.469 Mbps for host h1 and 4.221 Mbps for host h2. The loss percentage is 53.1% for host h1 and 53.1% for host h2.

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.

Solution: What are the measured goodputs and loss percentages in scenarios A1, A2 and A3?

Scenario A1: The goodput is 1 Mbps for host h1 and 0.5 Mbps for host h2. The loss percentage is 0% for host h1 and 0% for host h2.

Scenario A2: The goodput is 0.964 Mbps for host h1 and 3.83 Mbps for host h2. The loss percentage is 3.58% for host h1 and 4.11% for host h2.

Scenario A3: The goodput is 0.487 Mbps for host h1 and 4.314 Mbps for host h2. The loss percentage is 51.26% for host h1 and 52.06% for host h2.

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

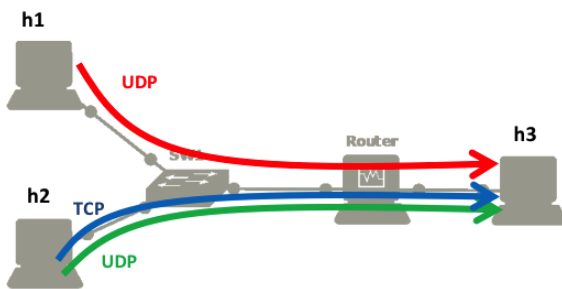
Solution: The measurements are quite close to the theoretical computations

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

Solution: Could you explain potential differences between the theoretical analysis (Q2 vs. Q3)? Specifically, What are the assumptions, we assumed in our theoretical computations (i.e., Q1 and Q2), but might not be exactly true in practice (i.e., Q3)? bandwidth is proportionally split and sending rates are constants.

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.

Solution: Based on a theoretical analysis of the scenario B1, the expected goodputs of the UDP flow of h1 is 1 Mbps and 0.5 Mbps for the UDP flow of h2. This is due to the fact that the total traffic of UDP flows is 1.599 Mbps. It is smaller than the bandwidth.

For the TCP flow of h2, the expected TCP rate is 3.401 Mbps.

*The expected goodput of the TCP flow of h2 is: expected TCP rate of h2 * MSS size/Ethernet frame size. Numerically, the goodput value for the TCP flow of h2 is 3.298 Mbps.*

*Explanation: If we also account for the headers, then the total traffic is still much less than the bandwidth, i.e., $(X + Y) * \frac{1066}{1000} \ll 5$. The TCP rate is then expected to be $Z = 5 - (X + Y) * \frac{1066}{1000}$ and the TCP goodput $Z * \frac{2896}{2986}$.*

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.

Solution: What are the measured goodputs in scenarios B1, B2 and B3? Scenario B1: The goodput is 952 kbps for UDP flow of host h1. The goodput is 449 kbps for UDP flow of host h2. The goodput is 3403 kbps

for TCP flow of host h2.

Scenario B2: The goodput is 833 kbps for UDP flow of host h1. The goodput is 3136 kbps for UDP flow of host h2. The goodput is 920 kbps for TCP flow of host h2.

Scenario B3: The goodput is 475 kbps for UDP flow of host h1. The goodput is 4291 kbps for UDP flow of host h2. The goodput is 45 kbps for TCP flow of host h2.

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

Solution: The TCP RENO's loss-throughput formula of a non-ECN long-lived TCP connection with round trip time RTT and constant segment size L that experiences loss probability q is

$$\frac{cL}{RTT\sqrt{q}}$$

(where c is a constant equal to 1.22). Here, $L = MSS$.

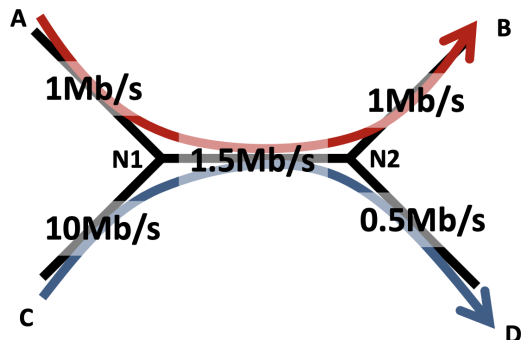
Q20/ Answer Question 9 in Lab 5 - Part 2 on Moodle.

Solution: Compute the TCP rates given by the previous formula for scenarios B2 and B3, assuming that the loss rate is the same for both TCP and UDP at h2. Use the loss rates and RTTs observed in each experiment.

Scenario B2: The observed UDP loss rate at h2 is 22% and the TCP RTT is 60 ms. With the loss throughput formula, the rate given is 1004 kbps. This is quite close to the experimental value. TCP CUBIC works in the TCP RENO-friendly regime.

Scenario B3: The observed UDP loss rate at h2 is 52% and the TCP RTT is 54 ms. With the loss throughput formula, the rate given is 726 kbps. This is far to the experimental value. The loss-throughput formula is completely wrong here. This can be explained, if we further observe the congestion window; we see that it is almost always equal to 1, thus TCP is always in slow start.

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.

Solution: If both sender A and sender C try to send data at maximum speed (i.e. 1Mbps and 10Mbps), the ratio of accepted traffic at N1 is **13.6%**.

Then, the rates of the flows are:

For flow AB, $rate_{AB} = 0.136$ Mbps. For flow CD, $rate_{CD} = 0.5$ Mbps.

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

Solution: In the situation of the previous question, the goodput received by B is **0.128**, and the goodput received by D is **0.469** Mbps.

And the formula of the loss percentage for flow AB is

$100 * (\text{Rate that A sends data} - \text{Goodput received by B}) / \text{Rate that A sends data}$.

Numerically, the loss percentages are:

For flow AB, $loss_{AB} = 87.2\%$ For flow CD, $loss_{CD} = 95.3\%$.

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.

Solution: Max-min fair allocation

If we use a max-min fair allocation, the rate at which A will send data is **1** Mbps and for C is **0.5** Mbps.

What algorithm do you use ? **Waterfilling**

The goodput received by B is **0.938** Mbps, and received by D is **0.469** Mbps.

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

Solution: Proportionally fair allocation

If we use a proportionally fair allocation, to determine the rate at which A and C will send data, we have to solve the following optimization problem: **$\max \log(rate_{AB}) + \log(rate_{CD})$** subject to:

$0 \leq rate_{CD} \leq 0.5$, $0 \leq rate_{AB} \leq 1$ $0 \leq rate_{AB} + rate_{CD} \leq 1.5$

Then, the rate at which A will send data is **1** Mbps, and for C is **0.5** Mbps.

Explanation: Assume that $rate_{CD} < 0.5$, then we can increase it to 0.5 and obtain a higher value of the objective function. Similarly, assume that $rate_{AB} < 1$, then we can increase it to 1 and obtain a higher objective function value.

2.2 EXPERIMENTAL SETTING

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

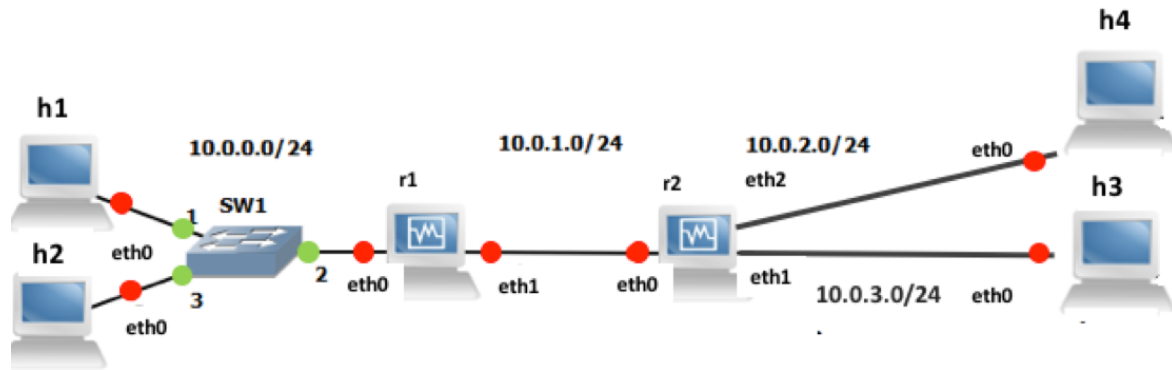


Figure 2: Setting for studying the congestion collapse.

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.

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.

Solution: *What are the goodputs and loss percentages of the two flows?
Flow h2 to h3: Goodput of 480 kbps and 95.2% loss. Flow h1 to h4: Goodput of 145 kbps and 85.5% loss.*

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.

Solution: *What are the goodputs and loss percentages of the two flows?
Flow h2 to h3: Goodput of 480 kbps and 3.9% loss. Flow h1 to h4: Goodput of 950 kbps and 5% loss.*

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.

Solution: What are the goodputs of the two connections with TCP connection instead of UDP ?
Flow h2 to h3: Goodput of **476 kbps**. Flow h1 to h4: Goodput of **946 kbps**. In Questions 15, 16 we obtained the rates given by the max-min fair allocation and by the proportional fair allocation, which, in this case, coincide. The goodputs are given by multiplying with $\frac{2896}{2986}$ and are: for 1to4 970 kbps and for 2to3 485 kbps. These values are very close to the values we obtained with TCP for this experiment. This is expected because the TCP rate allocation is between the max-min fair and the proportional fair allocations, which coincide in this case.

Q28/ Answer Question 8 in Lab 5 - Part 3 on Moodle.

Solution: These values are very **close** to the theoretical values we obtained with TCP for this experiment. This is expected because the TCP rate allocation is **between the max-min fair and the proportional fair allocations, which coincide in this case**.

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
```

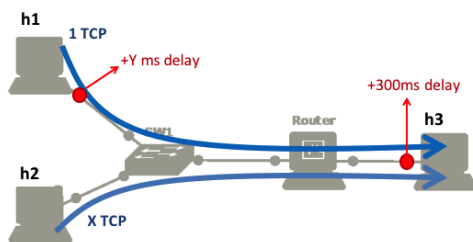
This rule can be changed to e.g., 400ms by typing `tc qdisc change dev eth0 root netem delay 400ms` or deleted by typing `tc qdisc del dev eth0 root`.

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

Solution: Add 300 ms of delay to the interface h3-eth0 of h3. Ping host h1 from host h3. The observed RTT is 300 ms.

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.

Solution: Theoretically, in scenario D1, the throughput of flow h1 is equal to the throughput of the other flows. This is because a fair sharing of the bandwidth is at the flow level.

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

Solution: Therefore, by using a theoretical analysis in scenario D1, the rate of the TCP flow of h1 is 1.25 . For host h2, the aggregated rate is 3.75 Mbps. The rate allocation of TCP is Answer Pareto efficient.

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.

Solution: The goodput obtained by h1 is **1.21 Mbps**. The aggregate goodputs obtained by h2 is **3.52 Mbps**. These results are **close** to the theoretical analysis done in the previous question.

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

Solution: Is there any queuing delay? **Yes**. This is due to the fact that the RTT is **higher than 300ms**.

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.

Solution: From the loss-throughput formula, the goodput values of flow 1 and 2 satisfy the following equation: Goodput of flow h1 = $300 / (300+Y) * \text{Goodput of flow h2}$. In addition, the Pareto optimality of TCP gives the following equation: **Goodput of flow h1 + Goodput of flow h2 = 5 * MSS size / ethernet frame size**

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

Solution: By solving the system of equations of the previous question, the theoretical total goodput obtained for h1 is: In scenario D2: **1320 kbps**.

In scenario D3: **910 kbps**.

The theoretical total goodput obtained for h2 is: In scenario D2: **3530 kbps**.

In scenario D3: **3940 kbps**.

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

Solution: Run the simulation corresponding to scenarios D2 and D3 for TCP CUBIC.

What is the measured total goodput obtained for h1 ? In scenario D2: **1660 kbps**.

In scenario D3: **1606 kbps**.

What is the measured total goodput obtained for h2 ?

In scenario D2: **2858 kbps**.

In scenario D3: **2930 kbps**.



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.

Solution: Run the simulation corresponding to scenarios D2 and D3 for TCP RENO.

What is the measured total goodput obtained for h1 ? In scenario D2: **885 kbps**.

In scenario D3: **651 kbps**.

What is the measured total goodput obtained for h2 ?

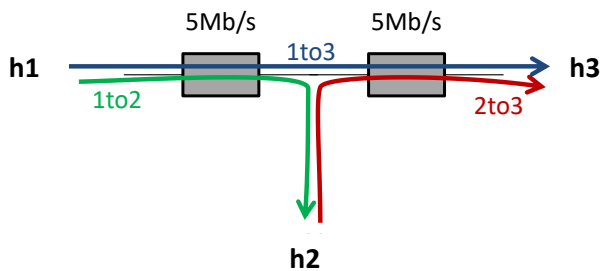
In scenario D2: **3107 kbps**.

In scenario D3: **3300 kbps**.

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

Solution: The emulation results show that CUBIC is *less sensitive to RTT* than RENO is. We can conclude that in this example TCP CUBIC operates in the regime where the RTT or product delay-bandwidth values are considered **high (non TCP RENO friendly area)**.

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.

Solution: What is the max-min fair allocation that corresponds to this network?

For flow h1 to h3, the rate that the traffic is sent is **2.5 Mbps**.

For flow h1 to h2, the rate that the traffic is sent is **2.5 Mbps**.

For flow h2 to h3, the rate that the traffic is sent is **2.5 Mbps**.

What algorithm do you use ? **Waterfilling**

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

Solution: If we use a proportionally fair allocation, to determine the rates of h1 to h3 (noted $rate_{1to3}$), of h1 to h2 (noted $rate_{1to2}$), of h2 to h3 (noted $rate_{2to3}$), we have to solve the following optimization problem:
 $\max \log(rate_{1to3}) + \log(rate_{1to2}) + \log(rate_{2to3})$

subject to:

$$0 \leq rate_{1to3} + rate_{1to2} \leq 5$$

$$0 \leq rate_{1to3} + rate_{2to3} \leq 5$$

$$rate_{1to3}, rate_{1to2}, rate_{2to3} \geq 0$$

The resolution of this problem gives $rate_{1to3} = 1.67$ Mbps, $rate_{1to2} = 3.33$ Mbps and $rate_{2to3} = 3.33$.

Explanation: Let $x = rate_{1to3}$, $y = rate_{1to2}$ and $z = rate_{2to3}$. The first two constraints should be satisfied with equality. To show this, let us suppose that the optimal solution x^*, y^*, z^* , is such that $x^* + y^* < 5$, $x^* + z^* < 5$. Then, since $x^*, y^*, z^* \geq 0$, we can increase y^*, z^* and obtain a higher value of the objective function, which contradicts the fact that x^*, y^*, z^* is an optimal solution. Thus, we can do the following replacements $y = 5 - x$ and $z = 5 - x$. The problem is then transformed as follows

$$\max(\log(x) + 2\log(5 - x)) \text{ s.t. } 0 \leq x \leq 5.$$

Solving this problem gives $x^* = 5/3$ and thus, $y^* = z^* = 10/3$.

3.3.2 EXPERIMENTAL SETTING

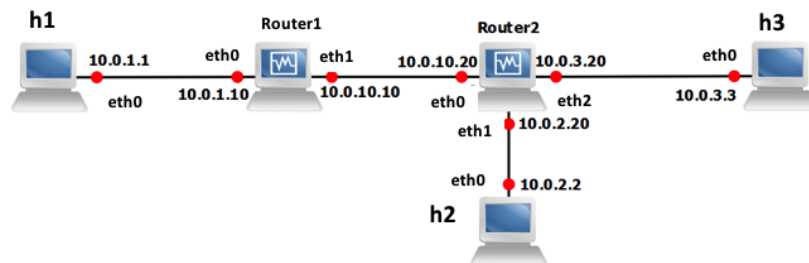
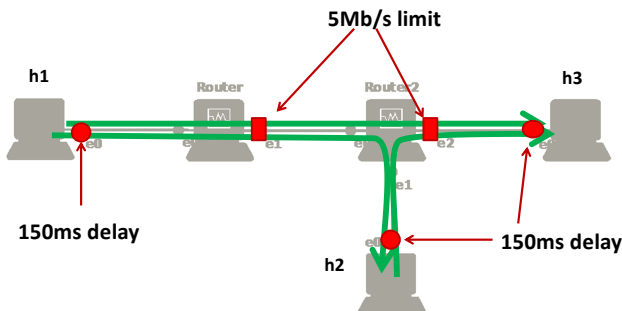


Figure 3: Fairness of TCP connections traversing multiple bottlenecks: wiring and addressing scheme.

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.

The 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.

***Solution:** Assume that there is no queuing delay. In theory, the RTT of the connections 1to2, 2to3 and 1to3 is 300 ms. Delays are only on one direction.*

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.

***Solution:** What is the measured goodput of each one of the three connections ?
For flow h1 to h2: Goodput = **2920** kbps.
For flow h1 to h3: Goodput = **1740** kbps.
For flow h2 to h3: Goodput = **2970** kbps.*

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

***Solution:** The average RTT are: For flow h1 to h2: RTT = **340** ms.
For flow h1 to h3: RTT = **370** ms.
For flow h2 to h3: RTT = **330** ms.*

Explanation: Router 2 (at the interface 2) has queueing delay around 30ms (sum for both directions) since the RTT 2to3 is around 330ms. Router 1 has queueing delay around 40ms (sum for both directions) since the RTT 1to2 is around 340ms. Note that for the flow 1to2, the bandwidth at its corresponding interface at router 2 is high compared to the value of 5Mbps, thus there is no additional queueing delay. Verify with RTT from 1to3.

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

***Solution:** By observing the RTTs, we can deduce that Router 2 (at the interface 2) has queueing delay around **30** ms (sum for both directions).
By observing the RTTs, we can deduce that Router 1 has queueing delay around **40** ms (sum for both directions).
Note that this correspond to the theoretical analysis: It is very close to the proportional fair allocation as expected from the theory of TCP and given the fact that the RTTs of all three flows are very close.*

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`. We will 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.

Set your TCP algorithm to TCP CUBIC. Disable ECN/RED at the interface `eth1` of the Router in the script `lab51_network.py` by setting `enable_red=False`, `enable_ecn=False`. Run the script `lab51_network.py`. 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.)

Q45/ Answer Question 1 in Lab 5 - Bonus on Moodle.

Solution: *The observed goodputs for host h1 is 2100 kbps and 2650 for h2. The observed RTT of h1 has an average value of 3000 ms and 3000 ms for h2. They fluctuate from 2000 ms to 5000 ms.*

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.

Q46/ Answer Question 2 in Lab 5 - Bonus on Moodle.

Solution: *The observed goodputs for host h1 is 2373 kbps and 2411 for h2. The observed RTT of h1 has an average value of 50 ms and 50 ms for h2.*

Q47/ Answer Question 3 in Lab 5 - Bonus on Moodle.

Solution: *The observed rates are similar with or without ECN/RED. When using ECN/RED, the observed RTTs decrease and fluctuate less than without ECN/RED.*

Q48/ Answer Question 4 in Lab 5 - Bonus on Moodle.

Solution: *Inspect the packets in Wireshark. Do you see any losses? Yes. Indeed, in Wireshark we see several retransmissions.*

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

Solution: *When using ECN we **should not** observe losses. This is because the notice **is** given to the source to reduce its rate when the queue is becoming full.*

Q50/ Answer Question 6 in Lab 5 - Bonus on Moodle.

Solution: *By checking the headers of the packets, are there any packets where the ECN related Flags are enabled ? **No**.*

*ECN is **not** working properly with CUBIC in this linux machine.*

*It seems that **RED** is responsible for RTT reduction, which drops a packet with probability proportional to how full the queue is.*

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.

Q51/ Answer Question 7 in Lab 5 - Bonus on Moodle.

Solution: *The observed goodputs for host h1 is **2390** kbps and **2390** for h2.*

*The observed RTT of h1 has an average value of **55** ms and **55** ms for h2.*

*These results are **close to** those of TCP CUBIC with ECN/RED enabled.*

Q52/ Answer Question 8 in Lab 5 - Bonus on Moodle.

Solution: *Inspect the packets in Wireshark. Do you see any losses? **No***

*Indeed, in Wireshark we see **no re-transmission***

Q53/ Answer Question 9 in Lab 5 - Bonus on Moodle.

Solution: *Observe that ECN works here.*

*Is the ECN Echo (ECE) flag for a packet that is echoed back to the source (h1 and h2) by the destination node h3 is set? **Yes**.*

*If congestion is encountered, on r1-eth1 the ECN field in the IP header is **11**.*