Name 1:
Name 2:

TCP/IP NETWORKING

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

Friday, November 22, 2019

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[1].

0.1 REPORT

This document will be your report (one per group). Type the answers directly in the PDF document. Use Adobe Reader XI, as it supports saving forms. When you finish, upload the report on Moodle. Do not forget to write your names on the first page of the report. **The deadline is Wednesday, December 4th, 23:55pm.**

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 Moodle (`http://moodle.epfl.ch/course/view.php?id=523`), 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. When 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
```

[1] In fact, this is the primary role of a congestion control algorithm. See *Congestion Avoidance and Control.* Van Jacobson and Michael J. Karels. ACM SIGCOMM 1988

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.

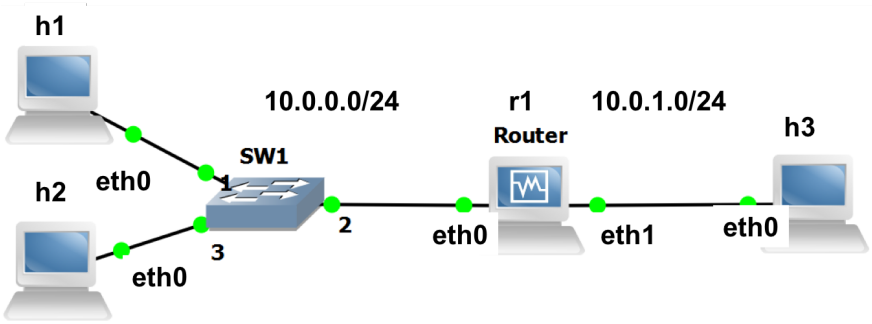


Figure 1: Initial configuration with 3 PCs and one router.

Complete the script `lab5/scripts/lab51_network.py` (in the two designated spaces) so as to configure the IP adresses and the routing tables according to the following addressing scheme:

- The subnet of hosts h1, h2 and the router (r1) is 10.0.0.0/24. The addresses of h1, h2 and of the router are respectively 10.0.0.1, 10.0.0.2 and 10.0.0.10.
- The subnet of h3 and of the router is 10.0.1.0/24. The addresses of h3 and of the router are respectively 10.0.1.3 and 10.0.1.10.

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, 843 and 867 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.

Start a UDP server on host h3 that listens on port 12345.

Q1/ Launch a UDP client on host h1 that sends data to this server at rate 100 kbps. What are the loss percentage and the goodput that you observe on h3?

Solution: *Goodput: 100 kbps, Loss: 0%*

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 transfered 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.

Start a TCP server on host h3 that listens on port 12345.

Q2/ Launch a TCP client on host h1 that sends data to this server. What is the goodput of the connection?

Solution: *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 $100\cdot$ (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 transfered 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 (5 minutes is probably OK for most scenarios). We also encourage you to run the experiments multiple times and provide average values.
- **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 $lab51\_network.py$, which you completed and run in the previous section, 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 3 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/ What is the size (in bytes) of the Ethernet frames that you expect to be used for sending the data of the UDP client? Consult the wikipedia page and wireshark at the server side. Note that some of the bytes of the Ethernet frame cannot be seen in Wireshark because they are only checked by the hardware.

Solution: *The UDP header length is 8 bytes.*
The IP header length is 20 bytes.
The Ethernet header length is 14 bytes.
An Ethernet packet starts with a 7-bytes preamble, which is used for clock synchronization, and an 1-byte start frame delimiter (SFD). The SFD is designed to break the bit pattern of the preamble and signal the start of the actual frame.
The data are 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. In wireshark at the server side we do not see the preamble, the SFD, the FCS and the interpacket gap.*

Q4/ The router has a bandwidth limit of 3 Mbps. What is the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute it.

Solution: *Max total rate = 3Mbps*
*The size of the Ethernet frame is 1066 bytes, 1000 of which correspond to the application data. Therefore, the theoretical max goodput is equal to 3*1000/1066 Mbps = 3000 *0.938 = 2814 kbps*

Start a UDP server on host h3 that listens on port 12345.

Q5/ Start a UDP client on host h1 that sends data at rate 2.5 Mbps. What are the loss percentage and the goodput observed on h3?

Solution: *Goodput: 2.5 Mbps, Loss: 0%*

Q6/ Repeat the operation with a UDP client that sends at rate 3 Mbps, 10 Mbps and 20 Mbps. What are the goodputs and loss percentages? Compare to the theoretical goodput computed above.

Solution: *3 Mbps: Goodput: 2880 kbps, Loss: 4.07%*
10 Mbps: Goodput: 2879 kbps, Loss: 71.49%

20 Mbps: Goodput: 2747 kbps, Loss: 86.2%
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.

Q7/ What is the size (in bytes) of Ethernet frames that you expect to be used for sending the data by the TCP connection? Consult the wikipedia page and wireshark at the server side. (Similarly with the corresponding question for UDP, note that some of the bytes of the Ethernet frame cannot be seen in Wireshark.)

Solution: *The TCP header length is 32 bytes.*
The IP header length is 20 bytes.
The Ethernet header length is 14 bytes.
An Ethernet packet starts with a 7-bytes preamble and an 1-byte SFD.
The length of the data depends on your MSS. We have observed am MSS of 2896 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. In wireshark at the server side we do not see the preamble, the SFD, the FCS and the interpacket gap.*

Q8/ The router has a bandwidth limit of 3 Mbps. What is then the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute it.

Solution: *Max total rate = 3Mbps*
*The size of the Ethernet frame is 2986 bytes, 2896 of which correspond to the application data. Therefore, the theoretical goodput is equal to 3*2896/2986 Mbps = 3000* 0.97= 2909 kbps*

Start a TCP server on host h3 that listens on port 12345.

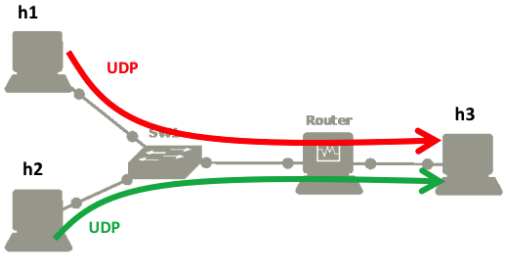
Q9/ Launch a TCP client on host h1 that sends data to this server. What is the goodput of the connection? Compare to the theoretical goodput computed above.

Solution: *2876 kbps (very close to the theoretically expected value).*

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 3 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, 2 Mbps or 5 Mbps.



Scenario	h1 (UDP)	h2 (UDP)
A1	1 Mbps	0.5 Mbps
A2	1 Mbps	2 Mbps
A3	1 Mbps	5 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).

Q10/ Based on theoretical analysis, what should be the goodputs and loss percentages of hosts h1 and h2 in the scenarios A1, A2 and A3. Explain your method below and fill in the table.

	Goodput h1		Loss percentage h1		Goodput h2		Loss percentage h2	
Q10a/ (A1)	[10a(i)]		[10a(ii)]		[10a(iii)]		[10a(iv)]	
Q10b/ (A2)	[10b(i)]		[10b(ii)]		[10b(iii)]		[10b(iv)]	
Q10c/ (A3)	[10c(i)]		[10c(ii)]		[10c(iii)]		[10c(iv)]	

Solution: *X = rate of h1, Y = rate of h2*

$Goodput\ UDP\ h1 = \min([X * 3/(X+Y)] * (1000/1066), X)$,

$Goodput\ UDP\ h2 = \min([Y * 3/(X+Y)] * (1000/1066), Y)$

	Goodput h1	*Loss percentage h1*	*Goodput h2*	*Loss percentage h2*
Q10d/ (A1)	*1 Mbps*	*0%*	*0.5 Mbps*	*0%*
Q10e/ (A2)	*0.938 Mbps*	*6.2%*	*1.876 Mbps*	*6.2%*
Q10f/ (A3)	*0.469 Mbps*	*53.1%*	*2.345 Mbps*	*53.1 %*

We now want to verify our analysis via emulation. For each scenario, start two UDP servers on h3 that listen on ports 12345 and 12346. 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, 2 or 5 Mbps.

Q11/ What are the measured goodputs and loss percentages in scenarios A1, A2 and A3?

	Goodput h1		Loss percentage h1		Goodput h2		Loss percentage h2	
Q11a/ (A1)	[11a(i)]		[11a(ii)]		[11a(iii)]		[11a(iv)]	
Q11b/ (A2)	[11b(i)]		[11b(ii)]		[11b(iii)]		[11b(iv)]	
Q11c/ (A3)	[11c(i)]		[11c(ii)]		[11c(iii)]		[11c(iv)]	

Solution:

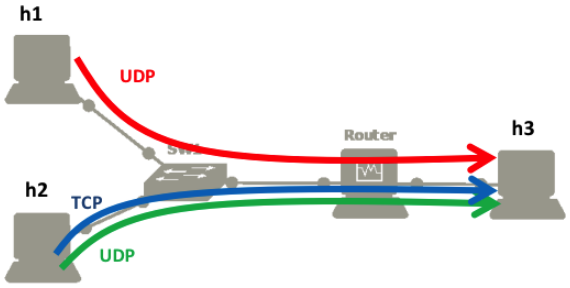
	Goodput h1	*Loss percentage h1*	*Goodput h2*	*Loss percentage h2*
Q11d/ (A1)	*1 Mbps*	*0%*	*0.5 Mbps*	*0%*
Q11e/ (A2)	*999.6 kbps*	*0.3 %*	*1881 kbps*	*6%*
Q11f/ (A3)	*506 kbps*	*49%*	*2378 kbps*	*53%*

Q12/ Do you see a difference between your theoretical analysis and the emulation? If yes, comment on the difference, and try to explain the possible sources.

Solution: *The measurements are quite close to the theoretical computations, similarly to the previous experiments. Differences though occur because our analysis of the losses is approximate. We assume that the bandwidth is proportionally split and the sending rates are constant, both of which may not be exactly true.*

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, 2 Mbps or 5 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	2 Mbps
B3	1 Mbps	5 Mbps

Q13/ Based on a theoretical analysis, what are the expected goodputs of the UDP flow of h1, the TCP flow of h2 and the UDP flow of h2 in the scenario B1 (explain)?

Solution: *Let X = rate of UDP h1, Y = rate of UDP h2, Z = rate of TCP h2. In scenario B1, $X + Y = 1 + 0.5 = 1.5 << 3$. If we also account for the headers, then the total traffic is still much less than the bandwidth, i.e., $(X + Y) * \frac{1066}{1000} << 3$. The TCP rate is then expected to be $Z = 3 - (X + Y)\frac{1066}{1000}$ and the TCP goodput $Z * \frac{2896}{2986}$. Numerically, the goodput values are: 1 Mbps for the UDP of h1, 0.5 Mbps for the UDP of h2 and 1.338 Mbps for the TCP of h2.*

We will now perform an emulation. As in the previous case, the bandwidth is limited to 3Mbps (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.**

Q14/ What are the measured goodputs of the three flows in scenarios B1, B2 and B3?

	UDP flow of h1		UDP flow of h2		TCP flow (h2)	
Q14a/ (B1)	[14a(i)]		[14a(ii)]		[14a(iii)]	
Q14b/ (B2)	[14b(i)]		[14b(ii)]		[14b(iii)]	
Q14c/ (B3)	[14c(i)]		[14c(ii)]		[14c(iii)]	

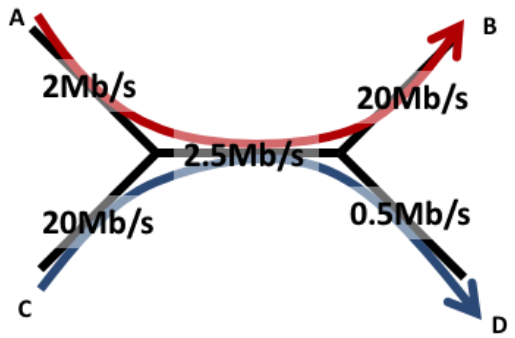
Solution:

	UDP flow of h1	*UDP flow of h2*	*TCP flow (h2)*
Q14d/ (B1)	*870 kbps*	*435 kps*	*1569 kbps*
Q14e/ (B2)	*770 kbps*	*1540 kbps*	*630 kbps*
Q14f/ (B3)	*480*	*2400 kbps*	*15 kbps*

Q15/ Compute the TCP rates given by the TCP RENO's loss-throughput 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. Compare with the rates obtained experimentally in the previous question.

Solution: *Scenario B2: From the emulation of the scenario B2 we observe that the loss rate is* $q = 24\%$ *(UDP at h2) and that* $RTT = 90$ *ms. In addition,* $MSS = 2896$ *bytes,* $c = 1.22$*. Therefore, the rate given by the loss throughput formula is* $\frac{MSS \cdot c}{\sqrt{q}RTT} = 641.6$ *kbps. This is very close to the experimental value (*630 *kbps). Obviously TC CUBIC works in the TCP RENO-friendly regime.*
Scenario B3: From the emulation of the scenario B3 we observe that the loss rate is $q = 52\%$ *(UDP at h2) and that* $RTT = 90$ *ms. In addition,* $MSS = 2896$ *bytes,* $c = 1.22$*. Therefore, the rate given by the loss throughput formula is* $\frac{MSS \cdot c}{\sqrt{q}RTT} = 429$ *kbps. Comparing with the experimental value, we see that 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 the 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 20Mbps. 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.

Q16/ If both sender A and sender C try to send data at maximum speed (i.e. 2Mbps and 20Mbps), what are the goodputs received by B and D? What are the loss percentages of these two flows?

Solution: *First compute* $rateA = 2.5 * 2/22$ *Mbps,* $rateC = 2.5 * 20/22$ *Mbps* $=> rateA = 0.227$ *Mbps,* $rateC = 2.27$ *Mbps*
Then, by further applying the bandwidth constraints on the links the rates of the flows become: $rateAB = 0.227$ *Mbps,* $rateCD = 0.5$ *Mbps,*
Next, obtain goodput by multiplying rates with $1000/1066$ *(UDP). Finally compute the loss percentage as follows:* $lossAB = (2 - 0.227 * 1000/1066)/2 * 100 = 89,3\%$ *and* $lossCD = (20 - 0.5 * 1000/1066)/20 * 100 = 97.6\%$.

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

What is the rate at which A and C will send data if we use

Q17/ a max-min fair allocation?

Solution: $rateAB = 2$ *Mbps,* $rateCD = 0.5$ *Mbps by waterfilling, Goodput is obtained by multiplying the rates with 1000/1066.*

Q18/ a proportionally fair allocation?

Solution: *Same as max-min fair allocation. To obtain it we solve the following optimization problem.* $max(log(rateAB) + log(rateCD))$ *s.t.*
$0 \leq rateCD \leq 0.5$, $0 \leq rateAB \leq 2$, $rateAB + rateCD \leq 2.5$.
For any values of $rateCD$ *and* $rateAB$ *that satisfy the first two constraints, the third constraint is also satisfied, thus it can be ignored. The solution is given by* $rateCD = 0.5$ *Mbps,* $rateAB = 2$ *Mbps. To see this, assume that* $rateCD < 0.5$*, then we can increase it to* 0.5 *and obtain a higher value of the objective function. Similarly, assume that* $rateAB < 2$*, then we can increase it to* 2 *and obtain a higher objective function value.*
Goodput is obtained by multiplying the rates with 1000/1066.

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.

After running the script you can verify that the configuration works with the pingall command.

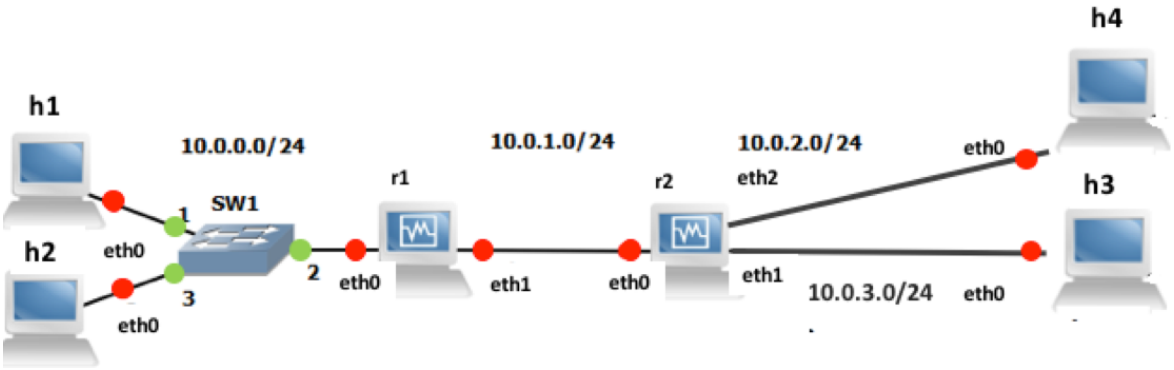


Figure 2: Setting for studying the congestion collapse.

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 2 Mbps and one on host h2 sending to host h3 with rate 20 Mbps.

Q19/ What are the goodputs of the two flows? What are the loss percentages?

Solution: *2to3: 480 kbps goodput and 97.6% losses*
1to4: 320 kbps goodput and 84% losses

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.

Q20/ What are the goodputs of the two flows? What are the loss percentages?

Solution: *2to3: 0.48 Mbps goodput and 4 % losses*
1to4: 1.9 Mbps goodput and 5% losses

2.2.2 TCP

Repeat the same process using TCP connections instead of UDP.

Q21/ What are the goodputs of the two connections? Compare with the theoretical analysis. Did you expect the observed rates?

Solution: *1to4: 1900 kbps goodput*
2to3: 478 kbps goodput
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* 1940 *kbps and for* 2*to*3 484 *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.*

Q22/ Can you conclude on what are the advantages of having a congestion control mechanism?

Solution: *Networks use congestion control and congestion avoidance techniques to try to avoid congestion collapse and packet loss. Also, fair allocation schemes allocate rates to flows according to their specific fairness schemes (e.g., max-min fairness, proportional fairness).*

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 **3 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.

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.

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
```

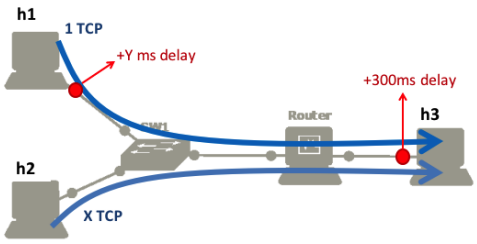
This rule can be changed to e.g., 400ms by typing `tc qdisc change dev eth0 root netem delay 400ms` or deleted by typing `tc qdisc del dev eth0 root`.

Q23/ Add 300 ms of delay to the interface h3-eth0 of h3. Ping host h1 from host h3. What is the observed RTT?

Solution: *300 ms (except from the first packets)*

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.

Q24/ Using a theoretical analysis, what is the total goodput that host h1 and host h2 will get in scenario D1?

Solution: *The throughput of the four flows is theoretically expected to be the same because the RTTs as well as the losses of the flows are expected to be the same since (i) all flows pass from the same queuing points and (ii) the paths of all four flows have the same number of hops (3-hops).*
Let us assume that x is the rate of each flow. Then the bandwidth limitation at the Router r1 gives $4x \leq 3$. However, TCP is Pareto optimal and therefore $x = \frac{3}{4} = 0.75$ Mbps.
(The goodput is obtained by $x * \frac{2896}{2986}$.)

*The resulting aggregated rates for each host are h1: 750 kbps, h2: 3*750=2250 kbps (if completely symmetric without retransmissions)*

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.

Q25/ What are the aggregate goodputs obtained by h1 and by h2 in scenario D1? Does this correspond to your theoretical analysis and if there is a difference, can you explain why?

Solution: *It corresponds to the theoretical analysis. The rates are: h1: 720 kbps, h2:* $720 + 790 + 670 = 2180$ *kbps*

Q26/ Can you tell if there is any queuing delay?

Solution: *Yes, because the RTT is higher than 300ms. The queueing delay is less than 100ms.*

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.

Q27/ In theory, for TCP RENO, what is the goodput that h1 and h2 will get as a function of Y?

Solution: *From the loss-throughput formula, the goodput values of flows 1 and 2 are* $TP1 = \frac{MSS \cdot c}{\sqrt{q1}RTT1}$ *and* $TP2 = \frac{MSS \cdot c}{\sqrt{q2}RTT2}$*, with* $q1$*,* $q2$ *the loss percentages observed for flows 1 and 2, correspondingly and* $RTT1$*,* $RTT2$ *the* RTT*s of flows 1 and 2, correspondingly. Assuming that* $q1 \approx q2$ *(which is logical since the flows have the same queuing points), then* $TP1 = \frac{RTT2}{RTT1}TP2$ *. If we consider that the queuing delays are small, then* $RTT1 = 300$ *and* $RTT2 = 300 + Y$*. As a result, the goodput values, TP1 (h1) and TP2 (h2) derive by solving the system of equations*

$$TP2 = (300 + Y) * TP1/300,\ TP1 + TP2 = 3 * \frac{2896}{2986}$$

Numerically, what are these values when:

	Total goodput for h1		Total goodput for h2	
Q27a/ (D2) $Y = 500$ ms	[27a(i)]		[27a(ii)]	
Q27b/ (D3) $Y = 1000$ ms	[27b(i)]		[27b(ii)]	

Solution:

	Total goodput for h1	*Total goodput for h2*
Q27c/ (D2) $Y = 500$ *ms*	*818 kbps*	*2180 kbps*
Q27d/ (D3) $Y = 1000$ *ms*	*563 kbps*	*2437 kbps*

Q28/ Run the simulation corresponding to scenarios D2 and D3 for TCP CUBIC. Please consult Section 0.2 on how you can set a TCP algorithm. What are the measured goodput values obtained by h1 and by h2?

	Total goodput for h1		Total goodput for h2	
Q28a/ (D2)	[28a(i)]		[28a(ii)]	
Q28b/ (D3)	[28b(i)]		[28b(ii)]	

Solution:

	Total goodput for h1	*Total goodput for h2*
Q28c/ (D2) $Y = 500$ *ms*	*1240 kbps*	*1550 kbps*
Q28d/ (D3) $Y = 1000$ *ms*	*1030 kbps*	*1760 kbps*

Q29/ Now, run the simulation corresponding to scenarios D2 and D3 for TCP RENO. What are the measured goodputs obtained by h1 and by h2.

	Total goodput for h1		Total goodput for h2	
Q29a/ (D2)	[29a(i)]		[29a(ii)]	
Q29b/ (D3)	[29b(i)]		[29b(ii)]	

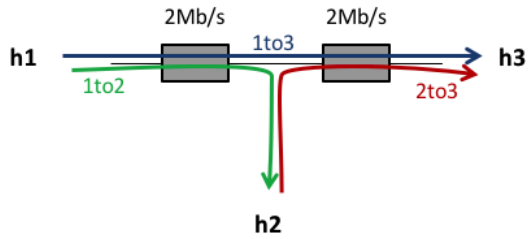
Solution:

	Total goodput for h1	*Total goodput for h2*
Q29c/ (D2)	*750 kbps*	*1850 kbps*
Q29d/ (D3)	*630 kbps*	*1930 kbps*

Q30/ Can you explain the difference observed between the results of TCP RENO and TCP CUBIC? Can you explain at which regime TCP CUBIC works, namely, (i) in the regime where the RTT or product delay-bandwidth values are considered low or (ii) in the regime where the RTT or product delay-bandwidth values are considered high? Explain.

Solution: *The emulation results show that CUBIC is much less sensitive to RTT than RENO is; with TCP Cubic the host h1 who has the higher RTT takes a smaller rate than host h2 but the difference between the two rates is much smaller than for TCP RENO. Therefore, we can conclude that in this example TCP CUBIC operates in the non TCP RENO friendly area, i.e., in the regime where the RTT or product delay-bandwidth values are considered high.*

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.

Using a theoretical analysis:
Q31/ What is the max-min fair allocation that corresponds to this network (explain)?

Solution: 1 *Mbps to all, by waterfilling (multiply by* $1896/1986$ *to obtain goodput)*

Q32/ What is the proportionally fair allocation (explain)?

Solution: $2/3$ *Mbps to* $1to3$ *and* $4/3$ *Mbps to each of* $1to2$ *and* $2to3$ *(multiply by* $1896/1986$ *to obtain goodput),*
To compute it we solve the following optimization problem.
Let x, y, z be the rates of $1to3$, $1to2$, $2to3$, *correspondingly.*
$\max(log(x) + log(y) + log(z))$ *s.t.* $x + y \leq 2$, $x + z \leq 2$, $x, y, z \geq 0$.
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* $x^* + y^* < 2$, $x^* + z^* < 2$. *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* $y = 2 - x$ *and* $z = 2 - x$. *The problem is then transformed as follows*
$\max(log(x) + 2log(2 - x))$ *s.t.* $0 \leq x \leq 2$.
Solving this problem gives $x^* = 2/3$ *and thus,* $y^* = z^* = 4/3$.

3.3.2 EXPERIMENTAL SETTING

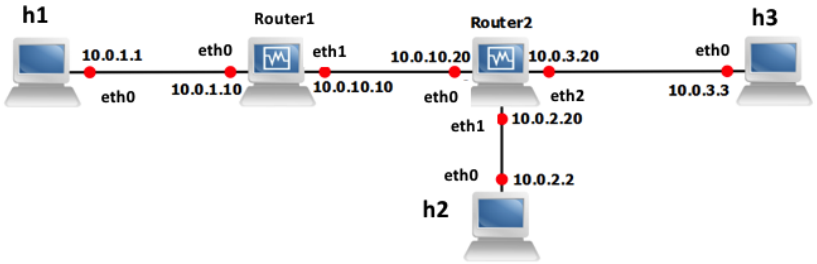
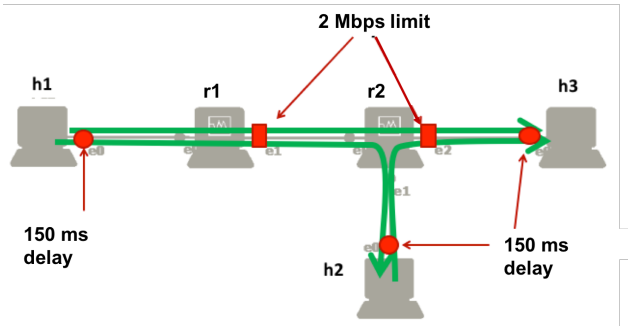


Figure 3: Fairness of TCP connections traversing multiple bottlenecks: wiring and addressing scheme.

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.

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



The bandwidth values of the interfaces r1-eth1 of Router1 and r2-eth2 of Router2 are limited to 2 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.

Q33/ Assume that there is no queuing delay. In theory, what should be the RTT of the connections *1to2*, *2to3* and *1to3*?

Solution: *300ms (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.

Q34/ What is the measured goodput of each one of the three connections?

[A34.a] *1to2*
[A34.b] *1to3*
[A34.c] *2to3*

Solution: *1to2 1250 kbps*
1to3 650 Kbps
2to3 1250 kbps

Q35/ What is approximately the average RTT of all three connections? Can you estimate approximately the queuing delay on Router1 and Router2?

Solution: *RTT 2to3=380 ms*
RTT 1to2= 370 ms
RTT 1to3= 450 ms
Router 2 (at the interface 2) has queueing delay around 80ms (sum for both directions) since the RTT 2to3 is around 380ms.
Router 1 has queueing delay around 70ms (sum for both directions) since the RTT 1to2 is around 370ms.
Note that for the flow 1to2, the bandwidth at its corresponding interface at router 2 is high compared to the value of 2Mbps, thus there is no additional queueing delay.
Verify with RTT from 1to3.

Q36/ Does this corresponds to your theoretical analysis?

Solution: *Yes 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.)

Q37/ What are the observed goodputs for hosts h1 and h2? What are the observed RTTs for the two flows?

Solution: *Rate of h1:* 1350 *kbps*
Rate of h2: 1400 *kbps*
Both RTTs have an average value of 6000 *ms. They fluctuate from* 4000 *ms to* 8000 *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.

Q38/ What are the observed goodputs for hosts h1 and h2? What are the observed RTTs for the two flows? Compare with the previous question (without ECN/RED).

Solution: *Rate of h1:* 1400 *kbps*
Rate of h2: 1450 *kbps*
Both RTTs have an average value of 75 *ms. They fluctuate from* 65 *ms to* 85 *ms. The rates are similar with or without ECN/RED. However, we can see that when using ECN/RED, the RTTs decrease a lot and they fluctuate much less. On the contrary, when not using them, the queues remain close to full all the time, thus*

the queuing delays increase and the RTTs increase.

Q39/ Inspect the packets in wireshark. Do you see any losses? Is the observed behavior expected for ECN? Check the headers of the packets and see if there are any packets where the ECN related Flags are enabled. Based on the comparisons with the case that ECN/RED are not enabled explain what you think is happening.

Solution: *Yes, there are losses because at wireshark we see several retransmissions. When using ECN, we should not observe losses, because it gives notice to the source to reduce its rate before the queue becomes full. We cannot find any packets with an ECN flag enabled. ECN is not working properly with CUBIC in this linux machine. It seams that RED is responsible for the reduction in the RTTs, which probabilistically drops 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 use wireshark in hosts h1 and h2 and in the router for capturing the traffic.

Q40/ What are the observed goodputs for hosts h1 and h2? What are the observed RTTs for the two flows? Compare with the previous cases.

Solution: *Rate of h1:* 1480 *kbps*
Rate of h2: 1400 *kbps*
Both RTTs have an average value of 85 *ms. They fluctuate from* 80 *ms to* 95 *ms. These results are very similar with those of TCP CUBIC with ECN/RED enabled.*

Q41/ Do you see any losses? Does ECN work here? If yes, give an packet ID that is marked by the router and the mark is echoed back to the source by the destination node h3.

Solution: *No, there are no losses as we do not see any retransmissions. ECN works. By inpsecting at the router's eth1 interface, find a packet sent by h1 or by h2 in which the router has set the ECN field in the IP header to 11 as well as h3 has set the ECN field in the TCP header of the corresponding ACK.*