Name 1: Marin THIERCELIN
Name 2: Cyril CADOUX

TCP/IP NETWORKING

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

Friday, November 24, 2017

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].

1 ORGANIZATION OF THE LAB EXERCISE

During this lab, you will use the virtual machine that you setup in the previous labs and you will study various aspects of congestion control in several network topologies constructed by Python scripts in Mininet.

1.1 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 6th, 23:55pm.**

For the research exercise a separate report in pdf should be submitted. Please be short and precise in your answers.

1.2 PRELIMINARY INFORMATION

1. On the virtual machine (VM), you also 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 and uncompress it in the shared folder that you configured in the first lab. The folder `lab5` contains four 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

[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

we will use to measure the goodput of each flow. The folder `lab5/scriptsINF/` contains the python script and a file with configurations that will be used to build the topology of this lab for the experiment that will run in INF019 .

2. Start the virtual PC. 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. Though, 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 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. If the congestion control algorithm is not Cubic, you can change it *until* the next reboot by typing in a terminal:

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

Similarly 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
```

4. For the experiments in INF019, you will use two Minix machines per pair additionally to your laptop. In one of the two machines (that will be indicated in detail in the description of the exercise), you should download the directory *lab5* and you should make the programs executable as in the second step.
5. For the research exercise only, you should download from Moodle the new virtual HD image *Lubuntu16.vdi* and create a new virtual machine following the procedure of Lab 1. *Lubuntu16.vdi* has a Linux version that supports DCTCP, which will be studied in the research exercise.
6. **Note:** In Mininet, between two directly connected hosts/routers there is always a 'transparent' switch, which may not be shown in the figures but it is already added in the code.

2 TCP VS UDP FLOWS

The first two subsections of this section will be performed solely in Mininet on your VM and the last two ones will be performed in INF019 as they require the use of two Minix machines (per group) in INF019. The topology that will be used in this section is shown in Fig. 1. For subsections 2.1, 2.3, you will construct the topology in Mininet by completing the script `lab5/scripts/lab51_network.py`. For subsections 2.3, 2.4, you will use the scripts in the folder `lab5/scriptsINF/` to create part of the topology in Mininet and part of it using the two Minix machines in INF019. More details will be given in Section 2.3. **Note**: We suggest that you have already taken a look at the questions in Section 2 before coming to INF019 for running it, as some questions are theoretical and can be answered in advance.

The python script `lab5/scripts/lab51_network.py` constructs the topology shown in Fig. 1.

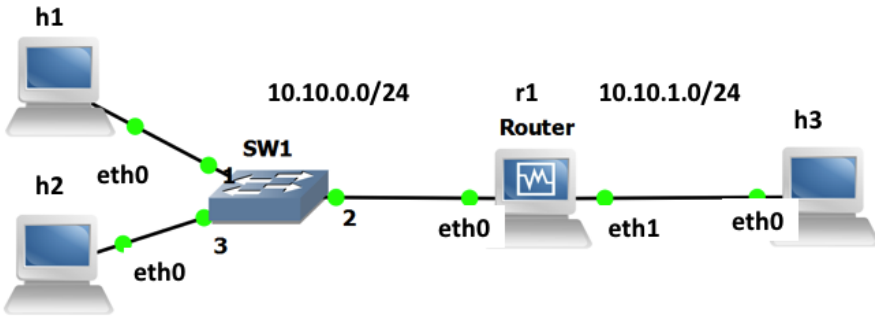


Figure 1: Initial configuration with 3 PCs and one router.

Complete the file `lab51_network.py` (in the two designated spaces) so as to configure the routing tables and the IP addresses according to the following addressing scheme:

- The subnet of hosts h1, h2 and the router (r1) is 10.10.0.0/24. The addresses of h1, h2 and of the router are 10.10.0.1, 10.10.0.2 and 10.10.0.10.
- The subnet of h3 and of the router is 10.10.1.0/24. The addresses of h3 and of the router are 10.10.1.3 and 10.10.1.10.

Open a terminal in your VM and run the script `lab51_network.py`. Test your configuration after running the topology with the *pingall* command.

2.1 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 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 time 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 2.1.1).

Start a UDP server on host h3 that listens on port 10000.

Remark: If you run the experiment multiple times the results may vary 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.

QQ1/ 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?

[A1] Loss : 0.0%
Goodput : 100 kbps

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 time, 862.4 is the average rate of the client: 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 instananeous 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 is the RTT measured by the TCP congestion control algorithm.

Start a TCP server on host h3 that listens on port 10001.

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

[A2] Average goodput on several tries : 9335 kbps

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

The folders `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 one minus the ratio between the number of packet received divided by 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.

2.2 ARTIFICIAL LIMITATION OF THE BANDWIDTH OF THE ROUTER

In order to reproduce experiments where the performance is limited by the network capacities, we will further limit the bandwidth of some interfaces. To do so, in this section we will specify the appropriate parameters for limiting the bandwidth in the topology through the possibilities offered by the TCLink class in Mininet.

In the script $lab51\_network.py$ that you completed and run in the previous section, there exists the command

```
link_r1sw2.intf1.config( bw=10 )
```

This command configures the bandwidth of the interface $r1 - eth1$ of the router to 10 Mbps.

Modify the above command so that the bandwidth of the interface $r1 - eth1$ of the router is bounded to 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 2.1 so that the topology in Fig. 1 is connected and all hosts communicate.

Remark: The limit of 5 Mbps is both for the packets that come in and go out of the interface $r1-eth1$ of the router, since the TCLink class creates symmetric TC link interfaces. In a later section we will explain how to perform bandwidth limitations with the linux tools by using the *class based queueing* (CBQ) scheduler. These tools allow also for non-symmetric bandwidth limitations.

2.2.1 UDP TEST

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.

QQQ3/ What is the size (in bytes) of Ethernet frames that you expect will be used to send the data of the UDP client? Verify with wireshark at the server side.

[A3] We expect a length of frame = 1000 + length(ethernet header) bytes
We see in fact that the length of the ethernet frames is 1042 bytes. It matches our prediction.

QQQ4/ The router is the bottleneck and has a limit of 5 Mbps. What is the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute.

[A4] We can send floor(5*(2^20) / (1042*8)) = 628 packets, each containing 1000 bytes of payload. We obtain a goodput of 628 000 * 8 / 1024 = 4906 kbps

Start a UDP server on host h3 that listens on port 10000.

QQQ5/ Start a UDP client on host h1 that sends data at rate 100 kbps. What are the loss percentage and the goodput observed on h3?

[A5] We still have a loss percentage of 0% and a goodput of 100 kbps

QQQ6/ Repeat the operation with 5 Mbps and 10 Mbps. What are the goodputs and loss percentages? What do you observe (explain)? Compare to the theoretical goodput computed above.

[A6] For 5 Mbps : loss = 3.8 % and goodput = 4800 kbps
For 10 Mbps : loss = 52 % and goodput = 4800 kbps

2.2.2 TCP TEST

QQQ7/ What is the size (in bytes) of Ethernet frames that you expect will be used to send the data by the TCP connection? Verify with wireshark at the server side.

[A7] An ethernet frame is composed of an ethernet header of 14 bytes, an ip header of 20 bytes, and TCP header of 32 bytes. The application data take 1448 bytes, hence we have 1448+20+14+32=1514 bytes

QQQ8/ What is then the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute.

[A8] Out of 1514 bytes of one packet, only 1448 are application data. So if the rate is 5 Mbps, the goodput is 5000*1448/1514 = 4.782 Mbps

Start a TCP server on host h3 that listens on port 10001.

QQQ9/ 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.

[A9] The goodput is equal to 4683 kbps.
We are close to the precomputed value.

2.3 COMPETING UDP FLOWS

The last two parts of this section will be performed in INF019, in a way similar to the corresponding section of Lab 2. To start with, you should disconnect your laptop from the Internet.

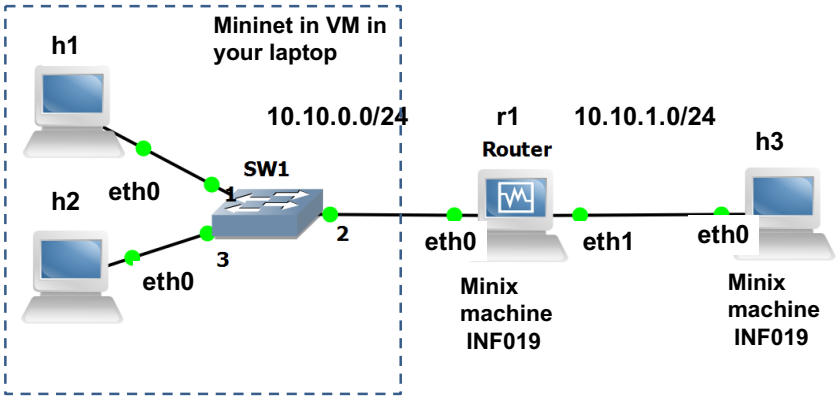


Figure 2: Initial configuration with 3 PCs and one router. It is indicated which hosts/router should be in Mininet in the VM in your laptop and which ones should be Minix machines in INF019.

The python script `lab5/scriptsINF/lab51b_network.py` constructs part of topology shown in Fig. 2. Specifically, it constructs the part of the network that involves the hosts h1, h2 and the switch sw1. You do not need to do any modifications in the script `lab5/scriptsINF/lab51b_network.py`.

Choose one of the Minix machines on your desk in INF019 to be the router r1 and the other Minix machine to be the host h3 (Fig. 2). In the Minix machine playing the role of h3 download the folders `lab5/tcp/` and `lab5/udp/` from Moodle or e.g., from your email. Then, disconnect both Minix machines from the Internet by disconnecting the corresponding Ethernet blue cables. Then, you should connect your laptop

with the Minix machine playing the role of r1 using an Ethernet cable. Next, you should connect the router r1 with the host h3 (i.e., the two Minix machines on your desk together), using another Ethernet cable.

Using as help the commands in the script `lab5/scriptsINF/lab51b_config.txt`, you should appropriately configure your laptop and the two Minix machines in INF019, so that the topology in Fig. 2 is connected and all hosts communicate with each other. To start with, open a terminal in your VM and run the script `lab51b_network.py`. For connecting your virtual environment (Mininet topology) to the router $r1$ in INF019, you need to follow the steps of Lab 2. *First*, your host machine (your laptop) should be able to ping the router $r1$ in INF019 and the opposite while also the router r1 and the host h3 (i.e., the two Minix machines on your desk in INF019) should be able to ping each other. Choose private ip addresses for each of the two sub-networks. *Second*, you should bridge the h1-eth0 and h2-eth0 interfaces of h1 and h2 with a real interface in your host machine, following exactly the same procedure as in Lab 2 for each one of them. Specifically, first remove any ip addresses that are already assigned in h1-eth0 and h2-eth0. Then, from Mininet, execute the following command.

```
mininet> sh ovs-vsctl add-port sw1 <interfacename>
```

Replace `interfacename` with the interface that accesses the Internet in your VM. Then, you should assign valid ip addresses to the hosts h1, h2 with the commands

```
dhclient h1-eth0
dhclient h2-eth0
```

Third, you should make any other necessary configuration in your host machine, and in r1 and h3 in INF019 so that all hosts can communicate with each other. You should also set TCP cubic in every host and router and enable ECN/RED in the router r1. For these configurations use the commands in the script `lab5/scriptsINF/lab51b_config.txt` and any additional command that is needed. Check also the next subsection for the commands that refer to the bandwidth limitation of the router r1.

2.3.1 BANDWIDTH LIMITATIONS USING LINUX COMMANDS

In this section, we will limit the bandwidth of the interfaces using linux commands, i.e., the *class based queueing* (CBQ) scheduler. Specifically, we will limit the bandwidth of the interface r1-eth1 of the router r1 as shown in Fig. 2. *Note that the names of the interfaces in the real machine r1 in INF019 may differ.*

As an example of a CBQ configuration, we postulate and explain the following commands:

```
# tc qdisc add dev r1-eth1 root handle 1:  cbq avpkt 1000 bandwidth
60mbit
# tc class add dev r1-eth1 parent 1:  classid 1:1 cbq rate 2mbit allot
1500 prio 5 bounded isolated
# tc filter add dev r1-eth1 protocol ip parent 1:  prio 16 u32 match ip
src 0/0 flowid 1:1
```

The first line installs a class based queue on r1-eth1, and tells the kernel that for calculations, it can be assumed to be a 60 Mbps interface. The exact value is less important as we will be limiting the rate below 60 Mbps. The second line creates a class, with ID $1:1$, which has 2 Mbit per second limit on the bandwidth allocated to it (and some other default parameters). The third line tells which traffic should go to the shaped class. In particular, it says that all packets going out of r1-eth1 will be stored in the class $1:1$.

For more information see: http://linux.die.net/man/8/tc-cbq or the tutorial in http://lartc.org/howto/lartc.qdisc.classful.html.

In case you need to change the configuration of the CBQ, you first need to delete the previous configuration. Do so by using the command `tc qdisc del dev eth1 root`.

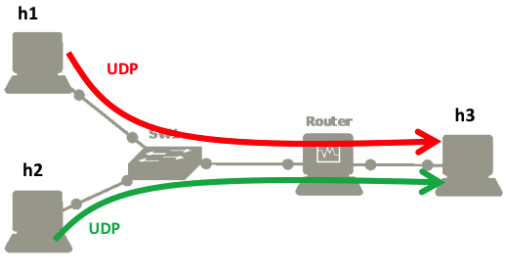
Remark: the limit of 2 Mbps is only for the packets that go out of the interface r1-eth1. Therefore, the packets coming towards the host or router do not interfere with the 2 Mbps limit.

The configuration for r1 is already initiated in the script `lab5/scriptsINF/lab51b_config.txt`. Fill the command needed to set a bandwidth of 5 Mbit per second at the interface r1-eth1 of the router r1 as shown in Fig. 2.

2.3.2 Experiment on competing UDP flows (to be performed in INF019)

We now want to explore what happens when two UDP flows are competing for the same bottleneck. The router has a capacity 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 2 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 2 Mbps, 3 Mbps or 6.5 Mbps.



Scenario	h1 (UDP)	h2 (UDP)
A1	2 Mbps	2 Mbps
A2	2 Mbps	3 Mbps
A3	2 Mbps	6.5 Mbps

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

QQ10/ Based on a 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.

[A10] at R1 h1 sends (2/8.5) % of the flow so it gets (2/8.5)*5 = 1.17 Mb on link r1 - h3
at R1 h2 sends (6.5/8.5) % of the flow so it gets (6.5/8.5)*5 = 3.82 Mb on link r1 - h3

	Goodput h1		Loss percentage h1		Goodput h2		Loss percentage h2	
Q10a/ (A1)	[10a(i)]	2 Mbps	[10a(ii)]	0	[10a(iii)]	2 Mbps	[10a(iv)]	0
Q10b/ (A2)	[10b(i)]	2 Mbps	[10b(ii)]	0	[10b(iii)]	3 Mbps	[10b(iv)]	0
Q10c/ (A3)	[10c(i)]	1.17 Mbps	[10c(ii)]	41.5	[10c(iii)]	3.82 Mbps	[10c(iv)]	41.3

We now want to verify our analysis by combined emulation and experiments. As before, the bandwidth of the interface r1-eth1 of the router is limited to 5 Mbps. For each scenario, start two UDP servers on h3 that listen on ports 10001 and 10002. Then, run a UDP client on h1 that sends data to h3 at 2 Mbps and a UDP client on h2 that sends data to h3 at rate 2, 3 or 6.5 Mbps.

QQQ11/ What are the measured goodputs and loss percentages in scenarios A1, A2 and A3?

[A11]

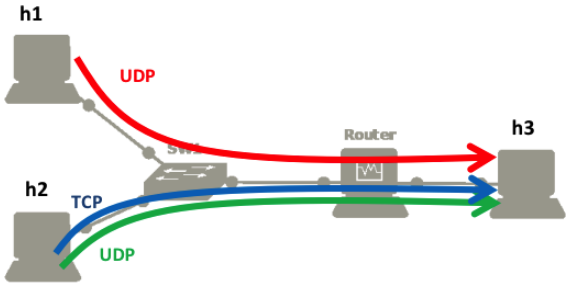
	Goodput h1		Loss percentage h1		Goodput h2		Loss percentage h2	
Q11a/ (A1)	[11a(i)]	2Mbps	[11a(ii)]	0	[11a(iii)]	2 Mbps	[11a(iv)]	0
Q11b/ (A2)	[11b(i)]	1.8 Mbps	[11b(ii)]	6	[11b(iii)]	2.8 Mbps	[11b(iv)]	7
Q11c/ (A3)	[11c(i)]	0.9 Mbps	[11c(ii)]	54.9	[11c(iii)]	3.7 Mbps	[11c(iv)]	41.9

QQQ12/ Do you see a difference between your theoretical analysis and the experiment? If yes, comment on the difference, and try to explain the possible sources.

[A12] We see a bigger loss than expected for h1, this might be because the emulation is not perfect.

2.4 TCP FLOWS COMPETING WITH UDP FLOWS (TO BE PERFORMED IN INF019)

We consider a similar scenario as in the previous case. Host h1 streams PMU data at rate 2 Mbps to host h3 and h2 streams video data to h3 at rate 2 Mbps, 3 Mbps or 6.5 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	2 Mbps	2 Mbps
B2	2 Mbps	3 Mbps
B3	2 Mbps	6.5 Mbps

QQQ13/ Based on a theoretical analysis, what should be the goodputs of the UDP flow of h1, the TCP flow of h2 and the UDP flow h2 in the scenarios B1, B2 and B3 (explain bellow and fill in the table)?

[A13] UDP source are greedy, and TCP has congestion control, so UDP get the same as before and tcp only get what is left.

	UDP flow of h1		UDP flow of h2		TCP flow (h2)	
Q13a/ (B1)	[13a(i)]	2	[13a(ii)]	2	[13a(iii)]	1
Q13b/ (B2)	[13b(i)]	2	[13b(ii)]	3	[13b(iii)]	0
Q13c/ (B3)	[13c(i)]	1.17	[13c(ii)]	3.82	[13c(iii)]	0

We now want to verify our analysis by combined emulation and experiments. As in the previous case, the bandwidth is limited to 5Mbps (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 command `xterm &` at h2's terminal to open a new terminal for h2.

QQQ14/ What are the measured goodputs of the three flows in scenarios B1, B2 and B3?

[A14]

	UDP flow of h1		UDP flow of h2		TCP flow (h2)	
Q14a/ (B1)	[14a(i)]	1.9 Mbps	[14a(ii)]	1.9 Mbps	[14a(iii)]	0.9 Mbps
Q14b/ (B2)	[14b(i)]	1.8 Mbps	[14b(ii)]	2.8 Mbps	[14b(iii)]	50 kbps
Q14c/ (B3)	[14c(i)]	1 Mbps	[14c(ii)]	3.8 Mbps	[14c(iii)]	0 Mbps

QQQ15/ Do you see a difference between your theoretical analysis and the experiment? If yes, comment on the difference, and try to explain the possible sources.

[A15] The value that tcp gets varies if we start the tcp client first ot the udp client first, but in general the tcp reduces its goodput quite fast. If we start the udp clients first, the tcp client stays in slow start and never manage to send a package. If we start the tcp client first, it manages to escape

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 the CUBIC algorithm shares the bandwidth when one or multiple bottlenecks are present in the network. In particular, we will investigate two characteristics: (i) CUBIC provides a fairness *per flow* and (ii) when the RTT or delay-bandwidth product values are low, Cubic is sensitive to delay and when the RTT or delay-bandwidth product values are high Cubic is less sensitive to delay. Note that for low RTT values CUBIC performs similarly to RENO, while this is not the case for higher RTT values as it is explained in the lecture notes. Also, we will perform comparisons between TCP CUBIC and TCP RENO.

For Sections 3.1 and 3.2, we will reuse the setting of Figure 1, as it 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 file as in Section 2.2). 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 2-5 minutes to see the stable results.
For the whole section, in order to enable better convergence that will be less impacted by queueing delays that affect the RTT values, **ECN and RED are already enabled in the script `lab51_network.py`**. More information on the ECN can be found in the research exercise in Section 5.

3.1 ADDING DELAY TO AN INTERFACE

To obtain more realistic and more 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
```

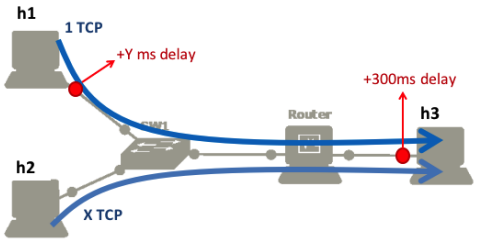
This rule can be changed by typing `tc qdisc change dev eth0 root netem delay 400ms` or deleted by typing `tc qdisc del dev eth0 root`.

QQQ16/ Add 300 ms of delay to the interface h3-eth0 of h3. Ping host h1 from host h3. What is the observed RTT?

[A16] We have an RTT of 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	200 ms
D3	1	400 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 setup D1:

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

QQQ17/ Using a theoretical analysis, what is the total goodput that host h1 and host h2 will get in scenario D1?

[A17] h1 has one connection with rate x and h2 has 3 connections with rate x.So the equation is here 3x + x <= 5, since r1 is limited to 5Mbps.This gives x = 5/4. So the rate at h1 should be around 1.250 kpbs and at h2 it should be 3.750 kbps. If we consider only the goodput (only

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.

QQQ18/ 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?

[A18] h1: 1110 kpbs, h2: 1140 kpbs, 1185 kpbs, 1105 kpbs
This corresponds to what we get in the theoretical approach.

QQQ19/ Can you tell if there is any queuing delay?

[A19] The roundtrip time that we observe in this case is above the one that we precomputed from question 16. So we can say that we have a queuing time (~50 ms)

We now explore scenario D2 and D3, where hosts h1 and h2 both 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.

QQQ20/ For TCP RENO, in theory, what is the goodput that h1 and h2 will get as a function of Y?

[A20] From the formula of the slides
tb1 = (MSS * 1.22)/((300+Y) * sqrt(q))
tb2 = (MSS * 1.22)/(300*sqrt(q))

Numerically, what are these values when:

	Total goodput for h1		Total goodput for h2	
Q20a/ (D2) $Y = 200ms$	[20a(i)]	1793 kbps	[20a(ii)]	2988 kbps
Q20b/ (D3) $Y = 400ms$	[20b(i)]	1434 kbps	[20b(ii)]	3347 kbps

QQQ21/ Run the simulation corresponding to scenarios D2 and D3 for TCP CUBIC. What are the measured aggregate goodputs obtained by h1 and by h2?

[A21] d2 aggregate: 4405 kbps
d3 aggregate: 4503 kbps

	Total goodput for h1		Total goodput for h2	
Q21a/ (D2)	[21a(i)]	2017 kbps	[21a(ii)]	2318 kbps
Q21b/ (D3)	[21b(i)]	2005 kbps	[21b(ii)]	2498 kbps

QQQ22/ Now, run the simulation corresponding to scenarios D2 and D3 for TCP RENO. For TCP RENO what are the measured aggregate goodputs obtained by h1 and by h2.

[A22] d2 aggregate: 4063 kbps
d3 aggregate: 4129 kbps

	Total goodput for h1		Total goodput for h2	
Q22a/ (D2)	[22a(i)]	1894 kbps	[22a(ii)]	2169 kbps
Q22b/ (D3)	[22b(i)]	1740 kbps	[22b(ii)]	2389 kbps

QQQ23/ Do the results for TCP RENO correspond to your theoretical analysis? If there is a difference, can you explain why?

[A23] We can observe a difference between empiric and theoretical results : The goodput of h2 is not as large as it should be, and on the contrary h1 is larger that it should be. First of all, the agglomerate goodput is much smaller than expected, so this reduces the goodput of both

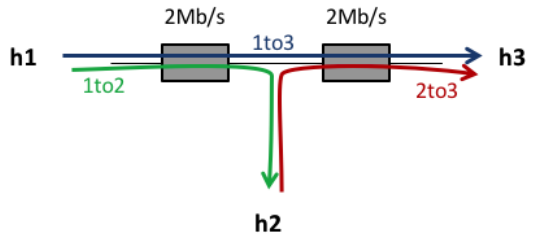
QQQ24/ Can you explain the difference observed between the results of TCP RENO and TCP CUBIC? Can you explain at which regime TCP CUBIC works, namely, (i) in the regime where the RTT or product delay-bandwidth values are considered low or (ii) in the regime where the RTT or product delay-bandwidth

values are considered high? Explain.

[A24] The results between RENO and CUBIC are small in this setup. Since we know that CUBIC is musch more efficient than RENO when RTT is large, we can say that CUBIC is here in the mode that considers low bandwidth values

At this moment, change back your TCP algorithm to TCP CUBIC.

3.3 Fairness of TCP connections traversing multiple bottlenecks



In this part, your goal is to study how the available bandwidth is shared when one TCP connection traverses two bottlenecks that are shared with two TCP connections that each share one of the bottlenecks.

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:
QQQ25/ What is the max-min fair allocation that corresponds to this network (explain)?

[A25] Water filling algorithm gives 1Mb/s for both connections.

QQQ26/ What is the proportionally fair allocation (explain)?

[A26] We must solve max(log(x) + 2*log(2-x)) with x<2
This gives x = 2/3 = rate of (13)
So rate for (12) and (23) = 4/3

3.3.2 Experimental setting

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

Th script `lab53_network.py` constructs the topology according to Figure 3 (note: the router 2 needs to have 3 interfaces available). Complete the script `lab53_network.py` by setting the routing tables at the routers, the default gateways of the hosts and by configuring bandwidth and delay values as shown in the following figure. **ECN and RED are already enabled in the script at the routers 1 and 2 for faster and better convergence**.

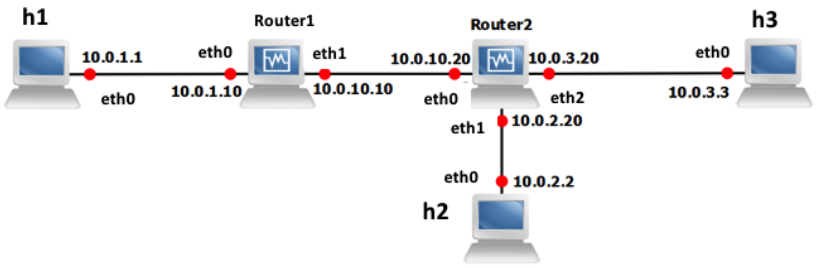
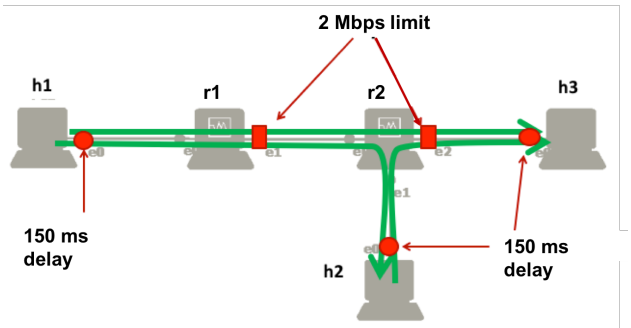


Figure 3: Fairness of TCP connections traversing multiple bottlenecks: wiring and addressing scheme.



The bandwidth values of the interfaces can be configured at the script as in Section 2.2 of this lab, i.e. the interfaces eth1 of Router1 and eth2 of Router2 are limited to 2 Mbps in bandwidth by appropriately setting the bw feature.

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

QQQ27/ Assume that there is no queuing delay. In theory, what should be the RTT of the connections *1to2*, *2to3* and *1to3*?

[A27] The 3 RTT should be around 300 ms

Now, start a TCP server on h2 and 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.

QQQ28/ What is the measured goodput of each one of the three connections?

[A28]

[A28.a] *1to2* 1130 kbps

[A28.b] *1to3* 720 kbps

[A28.c] *2to3* 1126 kbps

QQQ29/ Does this corresponds to your theoretical analysis?

[A29] The rates correspond to something between minmax fairness and proportionnaly fairness. So this correspond to what we saw in class

QQQ30/ What is approximately the average RTT of all three connections? Can you estimate approximately

the queuing delay on Router1 and Router2?

[A30] RTT(23) = 375 ms thus the queuing delay at Router 2 is equal to 75 ms.
RTT(12) = 390 ms thus the queuing delay at Router 1 is equal to 90 ms
RTT(13) = 465 ms thus the queuing delay is 165 ms. This corresponds to the sum of the

Now, modify the delay on h2 to 1000 ms and repeat the same experiment.

QQQ31/ For TCP CUBIC, is the goodput of *1to3* bigger or smaller than the ones of *1to2* and *2to3* (comment the results)?

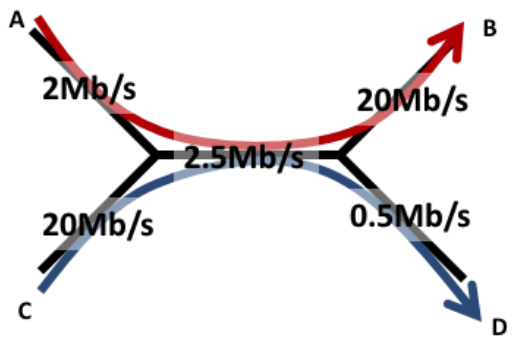
[A31] With cubic we have (13)=715 kbps, (12) = 1150 kbps and (23) = 1145 kbps. This is the same order of magnitude than the previous computation, with goodput(13) being still smaller than the other goodputs

Now, also change your TCP algorithm to RENO and repeat the experiment of the question QQQ31.

QQQ32/ For TCP RENO, is the goodput of *1to3* bigger or smaller than the ones of *1to2* and *2to3*? Comment the results and compare with TCP CUBIC. Explain also at which regime TCP CUBIC works, i.e., are the RTT or product delay-bandwidth values considered low or high?

[A32] (13) = 1120 kbps > (12) = 641 kbps > (23) = 457 kbps. This is because TCP Reno provides more bandwidth when the RTT is smaller. So we can conclude that Cubic is the regime that considers high RTT

4 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.

4.1 THEORETICAL ANALYSIS

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

QQQ33/ 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 links?

[A33] Central link can here transmit 2.5/22 of the traffic.
Red connection can transmit at rate 2*2.5/22 = 0.227 Mbps
The goodput at B (using UDP) is thus 0.227*1000/1042 = 0.218 Mbps

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

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

QQQ34/ a max-min fair allocation?

[A34] By water filling algo : First set both rates at 0.5 Mbs
Then the rate of red cannot be improved
Finally we set the rate of blue to 2 Mbps so that we entirely use central link

QQQ35/ a proportionally fair allocation?

[A35] We must solve max (log(x) + log(2.5-x)) with 0 < x < 0.5 and 0 < 2.5 - x < 2, with x denoting the rate of the red link. We get x = 0.5 Mbps and thusthe rate at A is 2Mbps. This is the same allocation as maxmin

4.2 EXPERIMENTAL SETTING

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

The script $lab54\_network.py$ creates a new topology according to Figure 4.

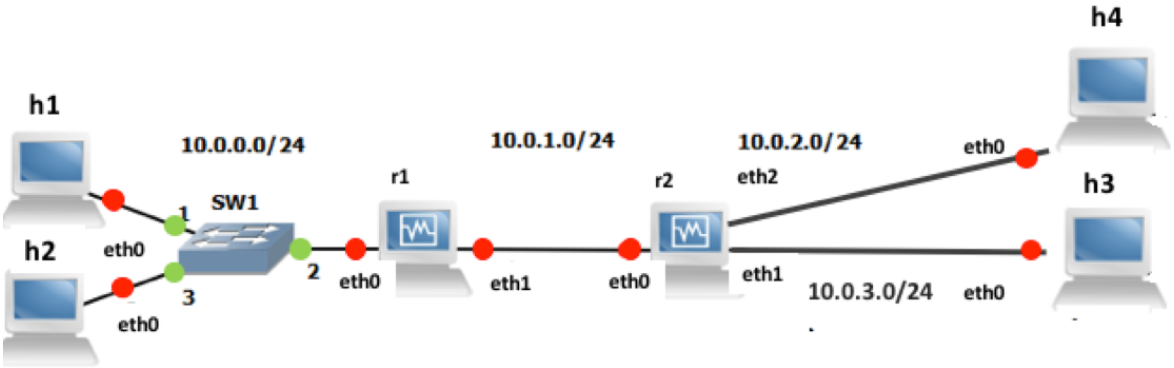


Figure 4: Setting for studying the congestion collapse.

The addressing scheme is as follows:

- The addresses of h1, h2 and h3 end with 1, 2 and 3.
- The addresses of the routers 1, 2 and 3 end with 10, 20 and 30.

The bandwidth limits of the links are already set at the script. ECN and RED are also enabled at the routers.

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

4.2.1 UDP

Launch a udp server on host h3 and another one on host h4 that listen on ports 10001 and 10002 correspondingly. Launch two UDP clients, one on host h1 sending to host h4 and one on host h2 sending to host h3 with rates 2Mbps (h1) and 20Mbps (h2) correspondingly.

QQQ36/ What are the goodputs of the two flows? What are the loss percentages?

[A36]
h1 to h4 = 179 kbps, loss = 91 %
h2 to h3 = 475 kbps, loss = 97.7%

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

QQQ37/ What are the goodputs of the two flows? What are the loss percentages?

[A37]
h1 to h4 = 1890 kbps, loss = 4.7 %
h2 to h3 = 485 kbps, loss = 3.7 %

4.2.2 TCP

Repeat the same process using TCP connections instead of UDP.

QQQ38/ What are the goodputs of the two connections? Did you expect the observed rates?

[A38]
h1 to h4 = 1864 kbps
h2 to h3 = 465 kbps
This corresponds to our theoretical rates, with a small differgence nevertheless. The latter is

QQQ39/ Can you conclude on what are the advantages of having a congestion control mechanism?

[A39]
Congestion control allows the network to be fair and ensures that no host is to greedy. This avoids congestion collapses and thus the network is always efficient.

Congestion control is nice.

5 RESEARCH EXERCISE: STUDY OF DCTCP & COMPARISONS WITH TCP CUBIC & TCP CUBIC WITH EXPLICIT CONGESTION NOTIFICATION (ECN)

(Optional, for bonus)

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 (and in addition RED is enabled), 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 detected a dropped packet and begins fast retransmit.

In this research exercise, we will study the effect of enabling ECN on the RTT and the congestion window by comparing CUBIC without using ECN/RED and CUBIC with ECN/RED. In this research exercise, we will also compare DCTCP with CUBIC.

Important Note 1: You should download from Moodle the new virtual HD image *Lubuntu16.vdi* and create a new virtual machine following the procedure of Lab 1. *Lubuntu16.vdi* has a Linux version that supports DCTCP.

Important Note 2: For this research exercise, you will write your answers and provide the appropriate graphs and explanations in a separate document, which you will upload as pdf together with this pdf file.

QQQ40/ Compute the formula of throughput-loss-congestion probability for DCTCP. Hint: One way could be to repeat the proof of the formula (2.5) (Proposition 2.2.1) of the lecture notes for a multiplicative decrease β and additive increase α, where α, β are appropriately defined for DCTCP.

[A40]

To achieve the above goals, we will use the topology of Fig. 1 which is defined in the script `lab51_network.py`. We should modify the script in order to limit the queue length of the router 1 by appropriately setting the max_queue_size parameter to the value of 100 packets. Furthermore you should limit the bandwidth of the interface 1 (r1-eth1) of the router equal to $5Mbps$.

You will run the topology `lab51_network.py` (with the two modifications above) in the three scenarios that follow.

Scenario 1: TCP CUBIC with **disabled ECN & RED** at the interface 1 of r1.

Scenario 2: TCP CUBIC with **enabled ECN & RED** at the interface 1 of r1.

Scenario 2: DCTCP with **enabled ECN & RED** at the interface 1 of r1. Note that for DCTCP ECN and RED should be always enabled as they are used for the estimation of the probability of congestion.

For each of the above scenarios, you will start a TCP server on h3 and two TCP clients on hosts h1 and h2 sending traffic to h3. Also, you should open wireshark or tshark in both hosts h1 and h2 for capturing the traffic.

Note that for emulating a new scenario, you should exit Mininet, clean up any previous topology, configure the appropriate TCP version and then run the script `lab51_network.py` with the appropriate configurations as described in each scenario. You should always wait until the rates stabilize. **After the end of the emulation for each scenario you should save the wireshark captures (name.pcappng) for hosts h1 and h2. Also note down the rates at which each emulation converged for both hosts h1 and h2 and the**

congestion window ranges. For tshark use the command `tshark -i h1-eth0 -F pcapng -w name.pcapng`.

Now we will use *tcptrace* and *xplot* (which are already installed for you in the virtual HD image) on the files *.pcappng that you saved from wireshark ot tshark for each scenario in order to compare the behavior of CUBIC without using ECN/RED, CUBIC with ECN/RED and DCTCP with ECN/RED.

The command

```
tcptrace -S name.pcapng
```

constructs a file denoted as $a2b\_tsg.xpl$ (and $b2a\_tsg.xpl$ that we will not use), that will be used from *xplot* as:

```
xplot.org a2b_tsg.xpl
```

to construct the Time Sequence Graph from sender a to receiver b. Please check what the Time Sequence Graph shows and how to use it in http://www.tcptrace.org/manual/node12_mn.html.

Similarly, the command

```
tcptrace -R name.pcapng
```

constructs a file denoted as $a2b\_rtt.xpl$ (and $b2a\_rtt.xpl$ that we will not use), that will be used from *xplot* as:

```
xplot.org a2b_rtt.xpl
```

to construct the graph of the RTT over time from sender a to receiver b.

Please use the screenshot tool of Linux (Accessories $\rightarrow$ Screenshot) to save these graphs.

By using all the above emulation results for both hosts and the three scenarios, compare the three scenarios in terms of the values of rates achieved, the congestion window ranges, the RTT values and the number of retransmissions. Provide appropriate explanations and graphs. Zoomed versions of graphs may be needed to show retransmissions in the time-sequence graph. Write your report while being expressed in a short and precise way.

Under some assumptions, which almost hold for a small number of flows as in our case, the maximum queue size that DCTCP can reach is given by $K + N$ where N is the number of flows and K is the queue size threshold after which DCTCP starts marking packets (see M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, Data center TCP (DCTCP). SIGCOMM '10, ACM). For low rates $\leq 1Gbps$, K is empirically set around 20 packets. Based on this formula and the other configurations of this experiment, explain the behavior of DCTCP with respect to the retransmissions.