Name 1:

Name 2:

# TCP/IP NETWORKING

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

Friday, November 25, 2016

**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 7th, 23:55pm.**

> • This lab is designed to be done in order. Each section uses knowledge from the previous section.

## 1.2  PRELIMINARY INFORMATION

On the virtual machine (VM), you also need to download an archive that contains the programs for this lab. To do so,

1. 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

---

[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

`lab5` contains three folders. The folder `lab5/scripts/` contains the python scripts that will be used to build the topologies of this lab. The folders `lab5/tcp/` and `lab5/udp/` contain tcp and udp (correspondingly) clients and servers that we will use to measure the goodput of each flow.

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 RENO congestion control algorithm. If the congestion control algorithm is not RENO, you can change it *until* the next reboot by typing in a terminal:

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

The script `/etc/init.d/local.sh` is executed at each boot. Therefore, this will set the congestion control to RENO at each boot.

# 2 TCP vs UDP FLOWS

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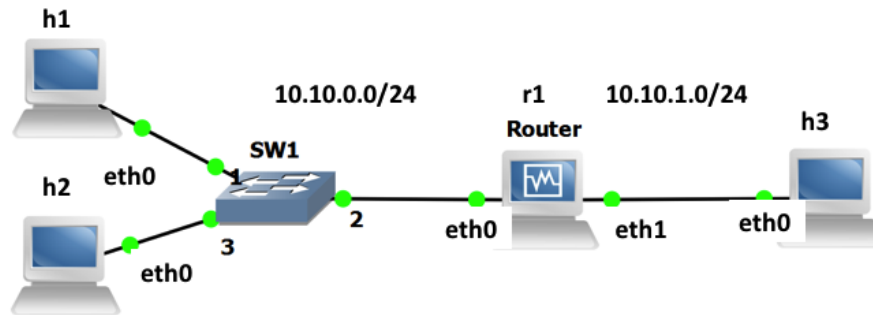


Figure 1: Initial configuration with 3 PCs and one router.

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.

**QQQ1/** Launch a UDP client on host h1 that sends data to this server at rate 100kbps. What are the loss probability and the goodput that you observe on h3?

[A1]

**QQQ2/** Repeat the operation with 1Mbps, 10Mbps, 100Mbps and 1Gbps. What are the goodputs and loss probabilities? At which rate the loss probability is greater than 1%? Explain the results via also carefully observing the code in the script lab51_network.py.

[A2]

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

**QQ3/** Launch a TCP client on host h1 that sends data to this server. What is the goodput of the connection?

[A3]

### 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.

**STOP IMPORTANT: Before You Continue...**

•**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 probability 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 probability 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 (2 minutes is probably OK for most scenarios). We also encourage you to run the experiments multiple times.
- **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. Specifically, the command
`net.addLink( h1, h2, bw=5, delay='2ms', max_queue_size=1000, loss=1)`
adds a bidirectional link between hosts h1 and h2 with bandwidth $5Mbps$, delay $2ms$, loss probability $1\%$, and with a maximum queue size of 1000 packets. The parameter bw is expressed as a number in $Mbps$; delay is expressed as a string with units in place (e.g. $'5ms'$, $'100us'$, $'1s'$); loss probability is expressed as a percentage (between 0 and 100); and $max\_queue\_size$ is expressed in packets. For more information you can access e.g., the webpage http://mininet.org/api/classmininet_1_1link_1_1TCLink.html.

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 $eth1$ of the router to $10Mbps$, instead of limiting the bandwidth of the whole link. However, in terms of the maximum possible traffic transferred, this leads to the same result as setting the bandwidth of the whole link to $10Mbps$, via the command `link_r1sw2=net.addLink( r1, sw2, bw=10)` during the creation of the link.

Modify the above command so that the bandwidth of the interface $eth1$ of the router is bounded to $3Mbps$.

**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.

**Remark**: The limit of $3Mbps$ is both for the packets that come in and go out of the interface $eth1$ 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 50kbps, the client sends packets that contain 1000 bytes of data each.

**QQQ4/** 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.

[A4]

**QQQ5/** The router is the bottleneck and has a limit of 3Mbps. What is the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute.

[A5]

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

**QQQ6/** Start a UDP client on host h1 that sends data at rate 100 kbps. What are the loss probability and the goodput observed on h3?

[A6]

**QQQ7/** Repeat the operation with 3Mbps and 10 Mbps. What are the goodputs and loss probabilities? What do you observe (explain)? Compare to the theoretical goodput computed above.

[A7]

6

### 2.2.2 TCP test

**QQQ8/** 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.

[A8]

**QQQ9/** What is then the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute.

[A9]

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

**QQQ10/** 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.

[A10]

## 2.3 Competing UDP flows

We now want to explore what happens when two UDP flows are competing for the same bottleneck. The router has a capacity 3Mbps 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 1Mbps to host h3 using UDP.
- Host h2 is streaming a video to host h3 using UDP. Depending on the quality, h2 sends at rate 1Mbps, 2Mbps or 4.5Mbps.
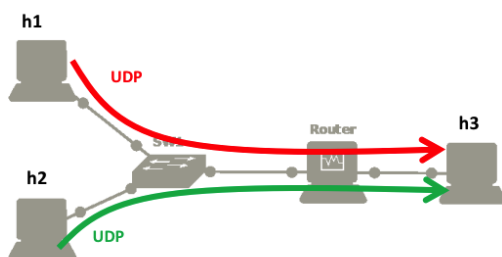


| Scenario | h1 (UDP) | h2 (UDP) |
|----------|----------|----------|
| A1       | 1 Mbps   | 1 Mbps   |
| A2       | 1 Mbps   | 2 Mbps   |
| A3       | 1 Mbps   | 4.5 Mbps |

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).

**QQQ11/** Based on a theoretical analysis, what should be the goodputs and loss probabilities of hosts h1 and h2 in the scenarios A1, A2 and A3. Explain your method below and fill in the table.

|  | Goodput h1 | Loss probability h1 | Goodput h2 | Loss probability 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)] |

We now want to verify our analysis by emulation. As before, the bandwidth of the interface `eth1` of the router is limited to 3Mbps. For each scenario, start two UDP servers on h3 that listen on ports 10001 and 10002. Use the command `xterm &` at h3's terminal to open a new terminal for h3. Please, watch out the arrangement of the terminals so that you keep track of the host they correspond to, as the new terminals will not obtain a name. Then, run a UDP client on h1 that sends data to h3 at 1Mbps and a UDP client on h2 that sends data to h3 at rate 1, 2 or 4.5Mbps.

**QQQ12/** What are the measured goodputs and loss probabilities in scenarios A1, A2 and A3?

[A12]

|  | Goodput h1 | Loss probability h1 | Goodput h2 | Loss probability h2 |
|---|---|---|---|---|
| Q12a/ (A1) | [12a(i)] | [12a(ii)] | [12a(iii)] | [12a(iv)] |
| Q12b/ (A2) | [12b(i)] | [12b(ii)] | [12b(iii)] | [12b(iv)] |
| Q12c/ (A3) | [12c(i)] | [12c(ii)] | [12c(iii)] | [12c(iv)] |

**QQQ13/** 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.

[A13]

## 2.4 TCP FLOWS COMPETING WITH UDP FLOWS

We consider a similar scenario as in the previous case. Host h1 streams PMU data at rate 1Mbps to host h3 and h2 streams video data to h3 at rate 1Mbps, 2Mbps or 4.5Mbps. In addition to this traffic, host h2 is also using a TCP connection to send a software update to h3.
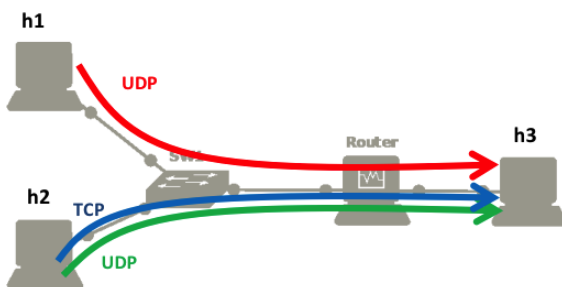


| Scenario | h1 (UDP) | h2 (UDP) |
|---|---|---|
| B1 | 1 Mbps | 1 Mbps |
| B2 | 1 Mbps | 2 Mbps |
| B3 | 1 Mbps | 4.5 Mbps |

**QQQ14/** 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)?

[A14]

|  | 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)] |

We now want to verify our analysis by emulation. As in the previous case, the bandwidth is limited to 3Mbps and we start two UDP servers on h3. Start also a TCP server on h3.

**QQ15/** What are the measured goodputs of the three flows in scenarios B1, B2 and B3?

[A15]

|  | UDP flow of h1 | UDP flow of h2 | TCP flow (h2) |
|---|---|---|---|
| Q15a/ (B1) | [15a(i)] | [15a(ii)] | [15a(iii)] |
| Q15b/ (B2) | [15b(i)] | [15b(ii)] | [15b(iii)] |
| Q15c/ (B3) | [15c(i)] | [15c(ii)] | [15c(iii)] |

**QQQ16/** 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.

[A16]

# 3 STUDY OF THE EFFECT OF THE EXPLICIT CONGESTION NOTIFICATION (ECN) ON THE RTT AND THE CONGESTION WINDOW

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, an ECN-aware router may set a mark in the IP header instead of dropping a packet in order to signal impending congestion. The receiver of the packet echoes the congestion indication to the sender, which reduces its transmission rate as if it detected a dropped packet. In this section, we will study the effect of enabling ECN on the RTT and the congestion window. Towards this goal, we will use the topology of Fig. 1 which is defined again in the script `lab52_network.py`. In order to increase packet dropping we will further limit the queue length of the router 1 by appropriately setting the $max\_queue\_size$ parameter to the value of 1000 packets as explained in Section 2.2 . Further more we define the bandwidth of the interface 1 (`eth1`) of the router equal to $5Mbps$.

After exiting Mininet and cleaning up any previous topology, run the script `lab52_network.py` (appropriately configured as explained above). Open a TCP server on host h3 and start a TCP client on host h1. Wait until the rate stabilizes.

**QQQ17/** Write down the source rate (h1), the approximate range of RTT values and the approximate range of congestion window values.

[A17]

Then, enable ECN by modifying the router's interface 1 (`eth1`) configuration as,
`link_r1sw2.intf1.config( bw=5, max_queue_size=1000, enable_ecn=True)`
The feature $enable\_ecn$ is by default set to the False value, while setting it to the True value enables ECN. Then, exit Mininet, clean up the previous topology and run the modified script `lab52_network.py` with enabled ECN. Open a TCP server on host h3 and start a TCP client on host h1. Wait until the rate stabilizes.

**QQQ18/** Write down the source rate (h1), the approximate range of RTT values and the approximate range of congestion window values.

[A18]

**QQQ19/** Compare with the case that ECN is not enabled. Explain.

[A19]

# 4 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 RENO algorithm shares the bandwidth when one or multiple bottlenecks are present in the network. In particular, we will investigate two characteristics: RENO provides a fairness *per flow* and is sensitive to delay.

> **STOP** For Sections 4.1 and 4.2, we will reuse the setting of Figure 1 (i.e., constructed by the script `lab51_network.py`). The bandwidth of the router should be limited to 3Mbps (use the same configuration file as in Section 2.2).

> **STOP** 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, **you should enable ECN** when you limit the bandwidth of the router similarly to the Section 3.

## 4.1 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 this interface (see http://www.linuxfoundation.org/collaborate/workgroups/networking/netem for more information about `tc` and `netem`).

For example, the following command adds 300ms 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`.

**✎QQ20/** Add 300ms of delay to the interface `eth0` of PC3. Ping host h1 from host h3. What is the observed RTT?

[A20]

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

## 4.2 FAIRNESS BETWEEN TCP CONNECTIONS AND DELAY
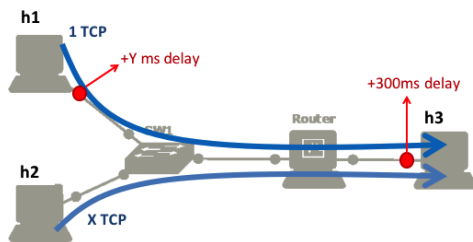


| Scenario | X (# conn. on h2) | Delay Y |
|----------|-------------------|---------|
| D1 | 3 | 0ms |
| D2 | 1 | 150ms |
| D3 | 1 | 300ms |

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 300ms on the interface `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.

**QQQ21/** Using a theoretical analysis, what is the total goodput that host h1 and host h2 will get in scenario D1?

[A21]

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

**QQQ22/** What are the aggregate goodputs obtained by h1 and by h2 in scenario D1?

[A22]
**QQQ23/** Does this correspond to your theoretical analysis and if there is a difference, can you explain why?

[A23]
**QQQ24/** Can you tell if there is any queuing delay?

[A24]

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.

**QQQ25/** In theory, what is the goodput that h1 and h2 will get as a function of $Y$?

[A25]

Numerically, what are these values when:

|  | Total goodput for h1 | Total goodput for h2 |
|---|---|---|
| Q25a/ (D2) $Y = 150ms$ | [25a(i)] | [25a(ii)] |
| Q25b/ (D3) $Y = 300ms$ | [25b(i)] | [25b(ii)] |

Now, run the simulation corresponding to scenarios D2 and D3:

**QQQ26/** What are the measured aggregate goodputs obtained by h1 and by h2.
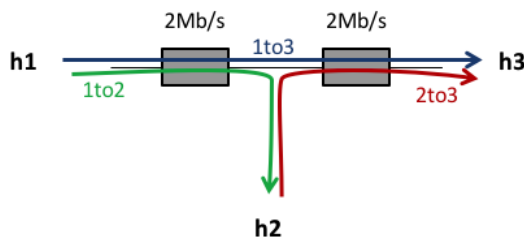
[A26]

|  | Total goodput for h1 | Total goodput for h2 |
|---|---|---|
| Q26a/ (D2) | [26a(i)] | [26a(ii)] |
| Q26b/ (D3) | [26b(i)] | [26b(ii)] |

**QQQ27/** Does this correspond to your theoretical analysis? If there is a difference, can you explain why?

[A27]

## 4.3 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.

### 4.3.1 THEORETICAL ANALYSIS

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

Using a theoretical analysis:
**QQQ28/** What is the max-min fair allocation that corresponds to this network (explain)?

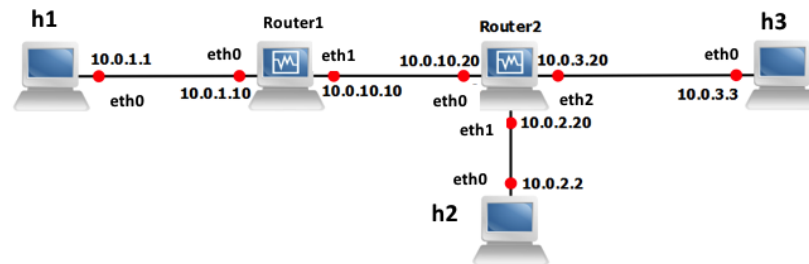[A28]

### 4.3.2 EXPERIMENTAL SETTING



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

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

Th script `lab53_network.py` constructs the topology according to Figure 2 (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. **Remember to enable ECN for faster and better convergence**.
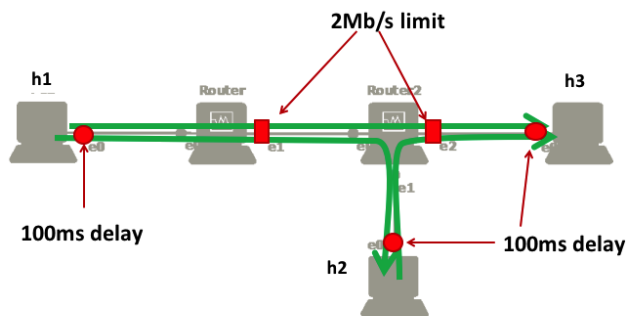


The bandwidth values of the interfaces can be configured at the script as in Section 2.2 of this lab, i.e. the interfaces `eth1` of router1 and `eth2` of router2 should be limited to 2Mbps by appropriately setting the bw feature.

As in Section 4.1 of this lab, use `tc` to add 100ms of delay to the outgoing packets of interfaces `eth0` of h1, h2 and h3.

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.

QQQ31/ What is the measured goodput of the three connections?

[A31]

[A31.a] *1to2*

[A31.b] *1to3*

[A31.c] *2to3*

QQQ32/ Does this corresponds to your theoretical analysis?

[A32]

QQQ33/ What is the average RTT of all three connections? Can you estimate the queuing delay on router1 and router2?
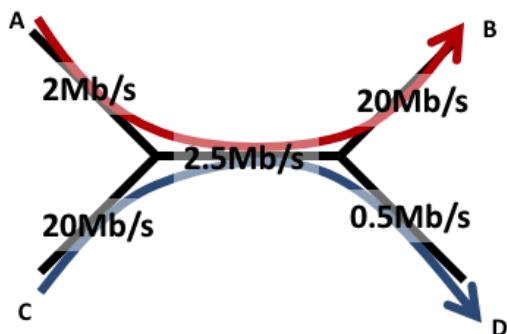
[A33]

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

QQQ34/ Is the goodput of *1to3* bigger or smaller than the ones of *1to2* and *2to3* (comment the results)?

[A34]

# 5 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.

## 5.1 THEORETICAL ANALYSIS

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

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

## 5.2 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 3.
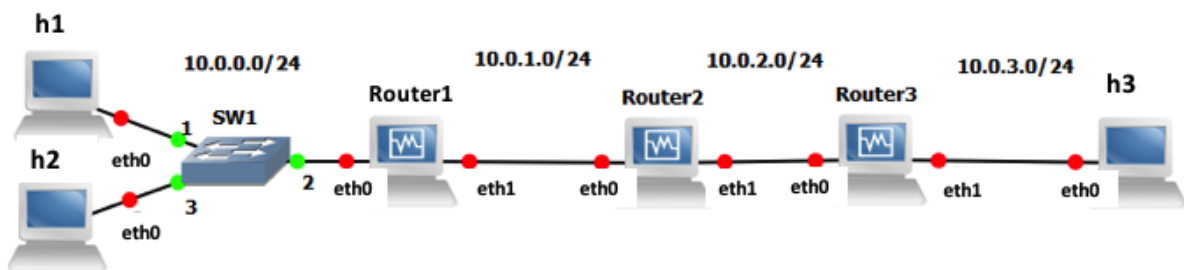


Figure 3: Setting for studying the congestion collapse.

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.

Complete the script $lab54\_network.py$ so as to configure the default gateways of the PCs and the routing tables of the routers in the designated space.

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

### 5.2.1 BANDWIDTH LIMITATIONS USING LINUX COMMANDS

In this section, we will limit the bandwidth of the interfaces using linux commands. Specifically, we will use the *class based queueing* (CBQ) scheduler for this purpose. We recommend to read some tutorials to understand how the CBQ works, in addition to reading the basic instructions below. One recommended source to start with is: http://lartc.org/howto/lartc.ratelimit.single.html.

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

```
# tc qdisc add dev eth1 root handle 1:  cbq avpkt 1000 bandwidth 60mbit
# tc class add dev eth1 parent 1:  classid 1:1 cbq rate 2mbit allot
1500 prio 5 bounded isolated
# tc filter add dev 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 eth1, and tells the kernel that for calculations, it can be assumed to be a 60Mbit interface. The exact value is less important as we will be limiting the rate below 60Mbit per second. The second line creates a class, with ID 1 : 1, which has 2Mbit 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 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 2Mbps is only for the packets that go out of the interface eth1. Therefore, the packets coming towards the host or router do not interfere with the 2Mbps limit.

To emulate the scenario, we will use the CBQ schedular to create:

- two queues on router1 that have bandwidths 2Mbps and 20Mbps and that filter packets from hosts h1 and h2;
- one queue on router2 that has a bandwidth 2.5Mbps.
- two queues on router3 that have bandwidths 20Mbps and 0.5Mbps that filter packets from hosts h1 and h2 correspondingly.

An example of configuration for router1 is as follows:

```
# tc qdisc add dev eth1 root handle 1:  cbq avpkt 1000 bandwidth 60mbit
# tc class add dev eth1 parent 1:  classid 1:1 cbq rate 2mbit allot
1500 prio 5 bounded isolated
# tc class add dev eth1 parent 1:  classid 1:2 cbq rate 20mbit allot
1500 prio 5 bounded isolated
# tc filter add dev eth1 protocol ip parent 1:  prio 16 u32 match ip
src 10.0.0.1 flowid 1:1
# tc filter add dev eth1 protocol ip parent 1:  prio 16 u32 match ip
src 10.0.0.2 flowid 1:2
```

The configuration for router3 is similar and the configuration for router2 is similar to the example provided above.

### 5.2.2 UDP

Launch two udp servers on host h3 that listen on ports 10001 and 10002 correspondingly. Launch two UDP clients, one on host h1 and one on host h2 that send data at rate 2Mbps (h1) and 20Mbps (h2).

**QQ38/** What are the goodputs of the two flows? What are the loss probabilities?

[A38]

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.

**QQ39/** What are the goodputs of the two flows? What are the loss probabilities?

[A39]

### 5.2.3 TCP

Repeat the same process using TCP connections instead of UDP.

**QQ40/** What are the goodputs of the two connections?

[A40]

**QQ41/** Do the above values for both UDP and TCP match your theoretical analysis?

[A41]

18

**QQQ42/** Can you conclude on what are the advantages of having a congestion control mechanism?

[A42]