



University of Lleida

Master's Degree in Informatics Engineering

Higher Polytechnic School

Exercise 3

ICT Project: Communication Services and Security

Cèsar Fernández Camón

Albert Pérez Datsira

July 15, 2021

Table of contents

1	Introduction	1
2	Simulation environment	2
3	Problem 1	3
3.1	Setup	3
3.2	Results	8
4	Problem 2	10
4.1	Setup	10
4.2	Results	14
5	Problem 3	15
5.1	Setup	15
5.2	Questions	18

List of Tables

1	Tap interface configuration commands	4
2	Network configuration commads	5
3	R1 router configuration commands	6
4	R2 router configuration commands	7
5	Network configuration commands	10
6	R1 router configuration commands	11
7	Network routing configuration	16
8	R1 router configuration commands	16
9	R2 router configuration commands	17
10	R1 router ip route configuration	18
11	R1 router NAT configuration	18
12	Packet lost by cases	21
13	Packet lost by cases adding a ping transmission	22

List of Figures

1	GNS3 logo	2
2	Problem 1 simulation topology scenario	3
3	Set up tap interfaces	4
4	Check tap interfaces configuration	4
5	Check links tap interfaces	5
6	Set up specific network routing configuration	5
7	Host ip route table	5
8	Configuration file /etc/iproute2/rt_tables	6
9	Ping through tap0 interface	7
10	Ping through tap1 interface	7
11	Problem 1 shell script usage	8
12	Results problem 1 script execution	8
13	Wireshark preferences	9
14	Host ip route table	11
15	packETHcli capture tap0 execution	12
16	packETHcli capture tap1 execution	13
17	Ping through tap0 interface	13
18	Ping through tap1 interface	13
19	Problem 2 shell script usage	14
20	Results problem 2 script execution	14
21	Problem 3 simulation topology scenario	15
22	Host ip/kernel routing table	16
23	R1 router NAT and Route configuration	18
24	R1 router queueing priority configuration	19
25	R1 router serial link 1/0 bandwidth	19
26	Video sample downloaded	20

1 Introduction

This assignment is focused on the one hand on learning how to install, configure and use *GNS3* (Graphical Network Simulator) and defining scenarios to be used through the example.

On the other hand, to put into practice the knowledge about the *Quality of Service* (QoS) and to be able to draw results from the analysis and reading of the network frames using tools such as *Wireshark*, but also to draw conclusions about the behavior of the simulations implemented.

2 Simulation environment

The GNS3 is a Graphical Network Simulator, open source and free software that offers an easy way to design, build and test networks of any size without the need for hardware by using a virtual environment.

Used by hundreds of thousands of network engineers worldwide to emulate, configure, test and troubleshoot virtual and real networks. Allows you to run a small topology consisting of only a few devices on your laptop, to those that have many devices hosted on multiple servers or even hosted in the cloud.

In addition, counts with an extended and active community where to ask questions, start discussions and see what is happening in the world of GNS3, which helps a lot when any doubt arises.



Figure 1: GNS3 logo

3 Problem 1

Based on topology from slide page 21 (QoS, CAR), write a shell script that computes the average rate for the following traffic flows:

- From 11.0.0.1 with precedence 7
- From 11.0.0.1 with precedence 1
- From 12.0.0.1

captured at interface R1-s1/0. Use *tshark* application.

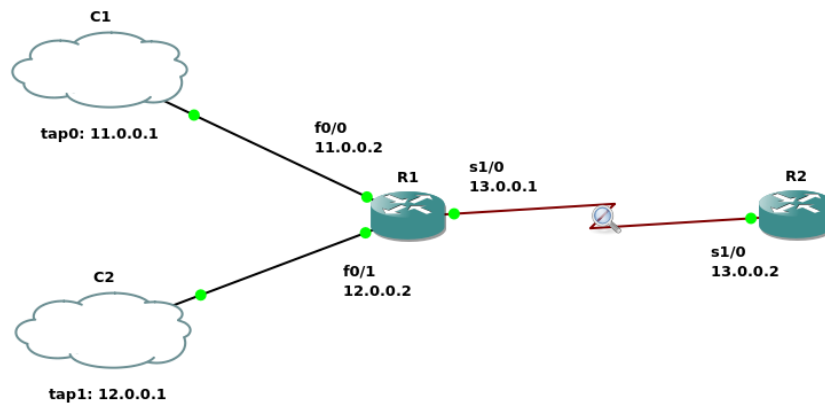


Figure 2: Problem 1 simulation topology scenario

3.1 Setup

This problem aims to show the behaviour of the Committed Access Rate (CAR) feature. As we can see there are two *tap* interfaces that must ping in the same direction. The two traffics pass through an intermediate router (R1) that classifies the traffic (by origin). and sends it to the destination (R2) through a serial link.

By configuration the R1 router classifies the traffic from *tap0*. The conforming traffic is transmitted marked with precedence 7 and the exceeding one with precedence 1. The traffic from *tap1* is not classified (marked).

The necessary configurations to build the topology and the way to generate the traffic in order to capture the frames are specified below.

3.1.1 Host Computer

Tap interfaces configuration

First, we must focus on the two computer interfaces that will be connected/linked to our topology by representing them as clouds as shown in figure 2 on page 3.

	Commands
Configure Tap0	<pre>sudo tuncctl -t tap0 -u <username> sudo ip link set tap0 up sudo ip add add 11.0.0.1/24 dev tap0</pre>
Configure Tap1	<pre>sudo tuncctl -t tap1 -u <username> sudo ip link set tap1 up sudo ip add add 12.0.0.1/24 dev tap1</pre>

Table 1: Tap interface configuration commands

```

albert@albert-VirtualBox: ~
albert@albert-VirtualBox:~$ sudo tuncctl -t tap0 -u albert
Set 'tap0' persistent and owned by uid 1000
albert@albert-VirtualBox:~$ sudo ip link set tap0 up
albert@albert-VirtualBox:~$ sudo ip add add 11.0.0.1/24 dev tap0
albert@albert-VirtualBox:~$ sudo tuncctl -t tap1 -u albert
Set 'tap1' persistent and owned by uid 1000
albert@albert-VirtualBox:~$ sudo ip link set tap1 up
albert@albert-VirtualBox:~$ sudo ip add add 12.0.0.1/24 dev tap1
albert@albert-VirtualBox:~$

```

Figure 3: Set up tap interfaces

Once executed the previous commands we can go on to check the status of our *tap* interfaces using the “*ifconfig <interface>*” command, although we may also use other more modern commands such as “*ip address show*”.

```

albert@albert-VirtualBox: ~
albert@albert-VirtualBox:~$ ifconfig tap0
tap0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 11.0.0.1 netmask 255.255.255.0 broadcast 0.0.0.0
    ether 76:d4:3c:4a:a6:66 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

albert@albert-VirtualBox:~$ ifconfig tap1
tap1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 12.0.0.1 netmask 255.255.255.0 broadcast 0.0.0.0
    ether f6:c8:10:0e:5e:5d txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

albert@albert-VirtualBox:~$

```

Figure 4: Check tap interfaces configuration

In addition, to further check the established links and their status you may use the command “*ip link show <interface>*” as seen in the figure 5 on page 5.

```

albert@albert-VirtualBox: ~
albert@albert-VirtualBox:~$ ip link show tap0
9: tap0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN
mode DEFAULT group default qlen 1000
    link/ether 76:d4:3c:4a:a6:66 brd ff:ff:ff:ff:ff:ff
albert@albert-VirtualBox:~$ ip link show tap1
8: tap1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN
mode DEFAULT group default qlen 1000
    link/ether f6:c8:10:0e:5e:5d brd ff:ff:ff:ff:ff:ff
albert@albert-VirtualBox:~$

```

Figure 5: Check links tap interfaces

Network configuration

Then we should add the corresponding routes to our IP table so that our computer knows how to reach the network 13.0.0.0/24, and thus be able to perform the corresponding pings.

	Commands
Add IP Route Gateways	<pre> sudo route add -net 13.0.0.0/24 gw 11.0.0.2 sudo ip route add 13.0.0.0/24 via 12.0.0.2 table rt2 sudo rule add from 12.0.0.1/32 table rt2 </pre>

Table 2: Network configuration commads

```

albert@albert-VirtualBox: ~
albert@albert-VirtualBox:~$ sudo route add -net 13.0.0.0/8 gw 11.0.0.2
albert@albert-VirtualBox:~$ sudo ip route add 13.0.0.0/24 via 12.0.0.2 table rt2
albert@albert-VirtualBox:~$ sudo ip rule add from 12.0.0.1/32 table rt2
albert@albert-VirtualBox:~$

```

Figure 6: Set up specific network routing configuration

Hence, by means of the command “*ip route*” we will be able to observe our new routes in the computer IP table.

```

albert@albert-VirtualBox: ~
albert@albert-VirtualBox:~$ ip route
default via 10.0.2.2 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100
11.0.0.0/24 dev tap0 proto kernel scope link src 11.0.0.1 linkdown
12.0.0.0/24 dev tap1 proto kernel scope link src 12.0.0.1 linkdown
13.0.0.0/8 via 11.0.0.2 dev tap0 linkdown
169.254.0.0/16 dev enp0s3 scope link metric 1000
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
albert@albert-VirtualBox:~$

```

Figure 7: Host ip route table

But if looking at the commands used we will see that we would not only be using the local IP default table but the *rt2* one for *tap1* interface, so we can show ping responses on it.


```

albert@albert-VirtualBox: /etc/iproute2
albert@albert-VirtualBox:/etc/iproute2$ sudo -i
root@albert-VirtualBox:~# echo "1 rt2" >> /etc/iproute2/rt_tables
root@albert-VirtualBox:~# exit
logout
albert@albert-VirtualBox:/etc/iproute2$ cat rt_tables
#
# reserved values
#
255     local
254     main
253     default
0       unspec
#
# local
#
#1      inr.ruhep
1 rt2
albert@albert-VirtualBox:/etc/iproute2$

```

Figure 8: Configuration file `/etc/iproute2/rt_tables`

Although it is not necessary at all because just adding the gateway without specifying a new IP table as done with `tap0`, would be enough.

3.1.2 Routers

Regards routers, the general configurations are specified below, although if you wish to see them completely, they can be found in the corresponding `.cfg` files of the problem 1 GNS3 project.

Router 1

	Commands
Interface FastEthernet 0/0	ip address 11.0.0.2 255.255.255.0 duplex auto no shutdown
Interface FastEthernet 1/0	ip address 12.0.0.2 255.255.255.0 duplex auto no shutdown
Interface Serial 1/0	ip address 13.0.0.1 255.255.255.0 rate-limit output access-group 10 8000 2000 2000 set-prec-transmit 7 exceed-action set-prec-transmit 1 serial restart-delay 0 no shutdown
Access List	access-list 10 permit 11.0.0.1

Table 3: R1 router configuration commands

Router 2

	Commands
Interface Serial 1/0	ip address 13.0.0.2 255.255.255.0 serial restart-delay 0 no shutdown
Default Gateway	ip route 0.0.0.0 0.0.0.0 13.0.0.1

Table 4: R2 router configuration commands

Just comment the default gateway must be added to the R2 router so that it knows where to answer the pings.

3.1.3 Traffic generation

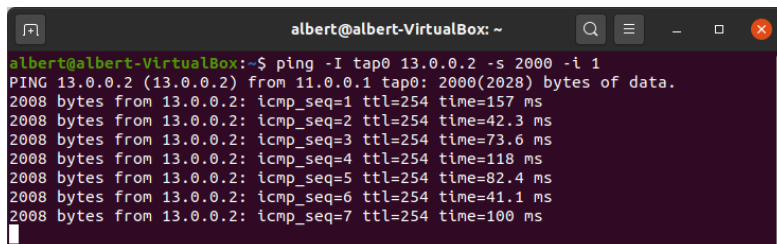
In order to generate traffic between interfaces the following commands were used:

- `$ ping -I tap0 13.0.0.2 -s 2000 -i 1`

sending 2000 bytes every second at a rate of 16Kbps

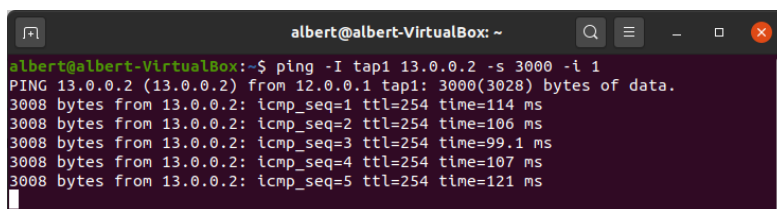
- `$ ping -I tap1 13.0.0.2 -s 3000 -i 1`

sending 3000 bytes every second at a rate of 24Kbps



```
albert@albert-VirtualBox: ~  
albert@albert-VirtualBox:~$ ping -I tap0 13.0.0.2 -s 2000 -i 1  
PING 13.0.0.2 (13.0.0.2) from 11.0.0.1 tap0: 2000(2028) bytes of data.  
2008 bytes from 13.0.0.2: icmp_seq=1 ttl=254 time=157 ms  
2008 bytes from 13.0.0.2: icmp_seq=2 ttl=254 time=42.3 ms  
2008 bytes from 13.0.0.2: icmp_seq=3 ttl=254 time=73.6 ms  
2008 bytes from 13.0.0.2: icmp_seq=4 ttl=254 time=118 ms  
2008 bytes from 13.0.0.2: icmp_seq=5 ttl=254 time=82.4 ms  
2008 bytes from 13.0.0.2: icmp_seq=6 ttl=254 time=41.1 ms  
2008 bytes from 13.0.0.2: icmp_seq=7 ttl=254 time=100 ms
```

Figure 9: Ping through tap0 interface



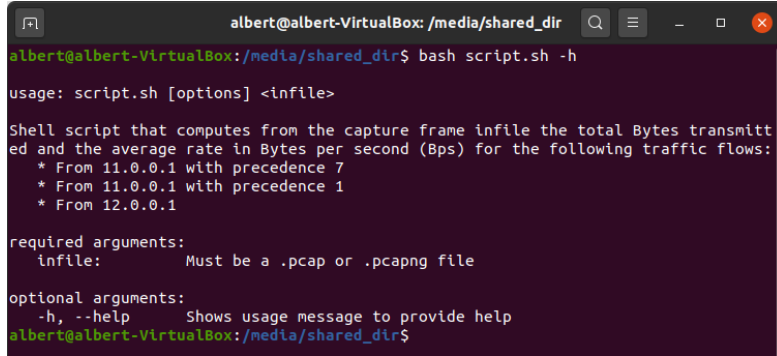
```
albert@albert-VirtualBox: ~  
albert@albert-VirtualBox:~$ ping -I tap1 13.0.0.2 -s 3000 -i 1  
PING 13.0.0.2 (13.0.0.2) from 12.0.0.1 tap1: 3000(3028) bytes of data.  
3008 bytes from 13.0.0.2: icmp_seq=1 ttl=254 time=114 ms  
3008 bytes from 13.0.0.2: icmp_seq=2 ttl=254 time=106 ms  
3008 bytes from 13.0.0.2: icmp_seq=3 ttl=254 time=99.1 ms  
3008 bytes from 13.0.0.2: icmp_seq=4 ttl=254 time=107 ms  
3008 bytes from 13.0.0.2: icmp_seq=5 ttl=254 time=121 ms
```

Figure 10: Ping through tap1 interface

3.2 Results

Once the topology was built and the traffic to the R1-s1/0 interface was captured using the wireshark tool, was able to move on to analyzing the traffic to the R1-s1/0 interface.

A shell script was built that by using the *tshark* application calculates the average rate in Bytes per second (Bps) for the flows specified in the problem statement from reading the *.pcapng* capture file. But also show the bytes transmitted by each of the flows as well as the total bytes of the scenario.



```
albert@albert-VirtualBox: /media/shared_dir
albert@albert-VirtualBox:/media/shared_dir$ bash script.sh -h
usage: script.sh [options] <infile>

Shell script that computes from the capture frame infile the total Bytes transmitted and the average rate in Bytes per second (Bps) for the following traffic flows:
  * From 11.0.0.1 with precedence 7
  * From 11.0.0.1 with precedence 1
  * From 12.0.0.1

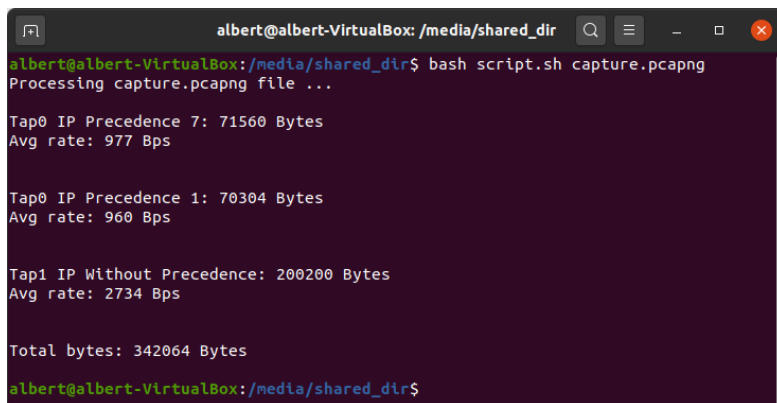
required arguments:
  infile:      Must be a .pcap or .pcapng file

optional arguments:
  -h, --help  Shows usage message to provide help
albert@albert-VirtualBox:/media/shared_dir$
```

Figure 11: Problem 1 shell script usage

The following table helps to better visualize the results:

Interface	IP precedence	Bytes transmitted (B)	Avg rate (Bps)	BW occupation (%)
tap0	7	71560	977	20.92
tap0	1	70304	960	20.55
tap1	Nan	200200	2734	58.52
Total Bytes		342064		



```
albert@albert-VirtualBox: /media/shared_dir
albert@albert-VirtualBox:/media/shared_dir$ bash script.sh capture.pcapng
Processing capture.pcapng file ...

Tap0 IP Precedence 7: 71560 Bytes
Avg rate: 977 Bps

Tap0 IP Precedence 1: 70304 Bytes
Avg rate: 960 Bps

Tap1 IP Without Precedence: 200200 Bytes
Avg rate: 2734 Bps

Total bytes: 342064 Bytes
albert@albert-VirtualBox:/media/shared_dir$
```

Figure 12: Results problem 1 script execution

As we can see, approximately about half of the traffic in *tap0* was marked as surplus traffic. As we know from *tap1*, it is not under CAR's policy and it makes sense that it is sending traffic through R1 with a higher speed rate.

In addition, the same results can be obtained without the need to create a script just by using the powerful wireshark desktop tool.

Firstly must be disabled the “Decode IPv4 TOS field” preferences option to see the statistics and packets by filtering with the variable **ip.tos.precedence**.

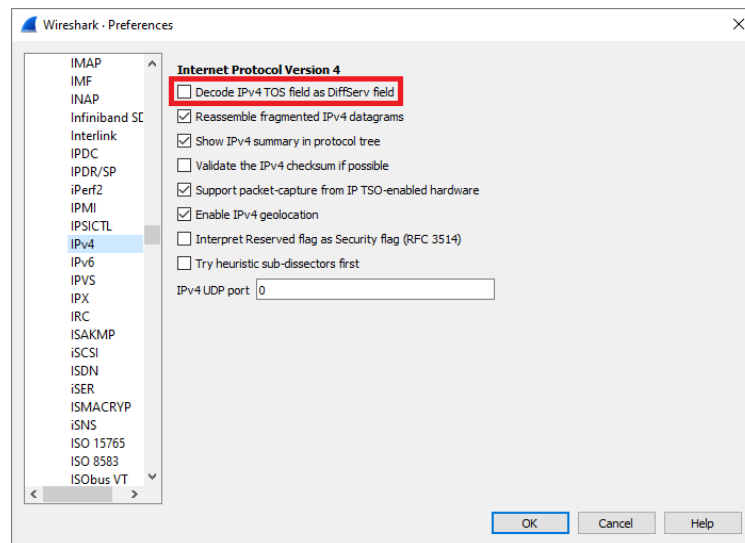


Figure 13: Wireshark preferences

And finally, in order to obtain the frame results, as in our case we are interested in knowing the destination ip, source ip and precedence, the following filters may be applied:

1. **tap0 (prec 7):** `ip.dst == 13.0.0.2 && ip.src == 11.0.0.1 && ip.tos.precedence == 7`
2. **tap0 (prec 1):** `ip.dst == 13.0.0.2 && ip.src == 11.0.0.1 && ip.tos.precedence == 1`
3. **tap1:** `ip.dst == 13.0.0.2 && ip.src == 12.0.0.1`

4 Problem 2

Based on topology from slide page 39 (QoS, CBWFQ), write a shell script that computes the percentage of bandwidth occupation at serial link R1-R2 for each of the streams coming from C1-tap0 and C2-tap1. Use *tshark* application.

4.1 Setup

This problem aims to show the behaviour of the Class-Based Weighted Fair Queuing (CBWFQ).

The topology scenario is the same as in problem 1 which is represented in figure 2 on page 3. There are two *tap* interfaces that generate traffic in the same direction. Both traffic must pass through an intermediate router (R1) that classifies the traffic according to its origin and sends the traffic to the destination (R2) through a serial link.

As described on page 39 of the pdf, *tap0* traffic belongs to the first class, which is assigned a queue that uses 80% of the total bandwidth. On the other hand, the *tap1* traffic belongs to the second class and it uses the remaining 20% of the bandwidth.

When the queue is full, the “tail-drop” is applied to discard the incoming IP packages. To demonstrate this behavior, interfaces generate traffic at a higher speed than they are able to send from an intermediate router, due to the limitations of the serial link.

4.1.1 Host Computer

Tap interfaces configuration

The configuration of the *tap* interfaces (clouds) is the same as in problem 1. You can consult the commands in the table 1 on page 4, as well as the terminal execution example in the figure 3.

Network configuration

	Commands
Add Ip Route Gateways	sudo route add -net 13.0.0.0/24 gw 11.0.0.2 sudo route add -net 13.0.0.0/24 gw 12.0.0.2

Table 5: Network configuration commands

```

albert@albert-VirtualBox:~$ ip route
default via 10.0.2.2 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100
11.0.0.0/24 dev tap0 proto kernel scope link src 11.0.0.1
12.0.0.0/24 dev tap1 proto kernel scope link src 12.0.0.1
13.0.0.0/24 via 12.0.0.2 dev tap1
13.0.0.0/24 via 11.0.0.2 dev tap0
169.254.0.0/16 dev enp0s3 scope link metric 1000
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
albert@albert-VirtualBox:~$

```

Figure 14: Host ip route table

4.1.2 Routers

Router 1

	Commands
Interface FastEthernet 0/0	ip address 11.0.0.2 255.255.255.0 duplex auto no shutdown
Interface FastEthernet 1/0	ip address 12.0.0.2 255.255.255.0 duplex auto no shutdown
Interface Serial 1/0	ip address 13.0.0.1 255.255.255.0 serial restart-delay 0 no shutdown
Acces Lists	access-list 101 permit 11.0.0.0 0.0.0.22 any access-list permit 102 12.0.0.0 0.0.0.255 any
Class Bandwidth	policy-map policy1 class class1 bandwidth percent 80 class class2 bandwidth percent 20
Class Access Group	class-map match-all class1 match access-group 101 class-map match-all class2 match access-group 102

Table 6: R1 router configuration commands

The creation of different policy classes (class1, class2) and the manual allocation of bandwidth can be seen in the applied configuration. As well as the mapping of these with the two access groups (101, 102).

Router 2

It has the same configuration as router 2 in problem 1. You can check its commands at page 4 in the table 1.

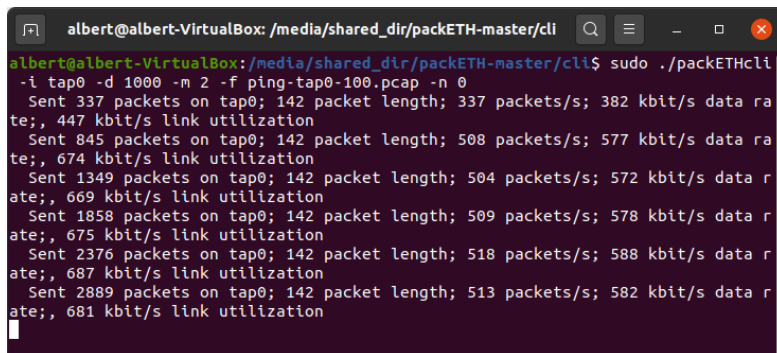
4.1.3 Traffic generation

Packet generator

To generate traffic through the network in order to fill bandwidth, was used the CLI version of the *packETH* packet generator tool software.

Allows you to create and send any possible packet or sequence of packets on the ethernet link. It is very simple to install and use, as it is free and open source just getting from its repository¹ on github, and supports many adjustments of parameters while sending.

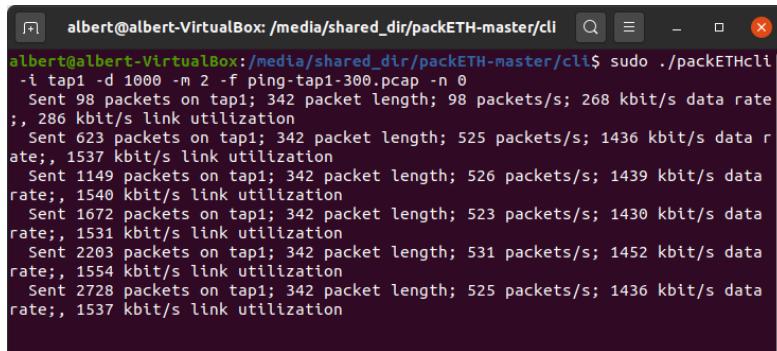
- \$./packETHcli -i tap0 -d 1000 -m 2 -f ping-tap0-100.pcap -n 0
- \$./packETHcli -i tap1 -d 1000 -m 2 -f ping-tap1-300.pcap -n 0



```
albert@albert-VirtualBox: /media/shared_dir/packETH-master/cli$ sudo ./packETHcli
-i tap0 -d 1000 -m 2 -f ping-tap0-100.pcap -n 0
Sent 337 packets on tap0; 142 packet length; 337 packets/s; 382 kbit/s data rate; , 447 kbit/s link utilization
Sent 845 packets on tap0; 142 packet length; 508 packets/s; 577 kbit/s data rate; , 674 kbit/s link utilization
Sent 1349 packets on tap0; 142 packet length; 504 packets/s; 572 kbit/s data rate; , 669 kbit/s link utilization
Sent 1858 packets on tap0; 142 packet length; 509 packets/s; 578 kbit/s data rate; , 675 kbit/s link utilization
Sent 2376 packets on tap0; 142 packet length; 518 packets/s; 588 kbit/s data rate; , 687 kbit/s link utilization
Sent 2889 packets on tap0; 142 packet length; 513 packets/s; 582 kbit/s data rate; , 681 kbit/s link utilization
```

Figure 15: packETHcli capture tap0 execution

¹<https://github.com/albeertito7/packETH>

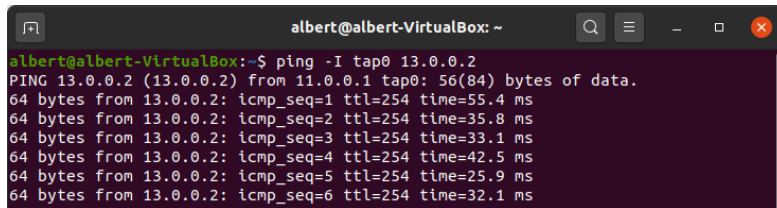
A terminal window titled 'albert@albert-VirtualBox: /media/shared_dir/packETH-master/cli'. The command 'sudo ./packETHcli -i tap1 -d 1000 -m 2 -f ping-tap1-300.pcap -n 0' has been executed. The output shows a series of statistics for packets sent on tap1, including packet length, packets/s, data rate, and link utilization. The statistics are as follows:

Step	Packets sent	Packet length	Packets/s	Data rate	Link utilization
1	98	342	98	268 kbit/s	286 kbit/s
2	623	342	525	1436 kbit/s	1537 kbit/s
3	1149	342	526	1439 kbit/s	1540 kbit/s
4	1672	342	523	1430 kbit/s	1531 kbit/s
5	2203	342	531	1452 kbit/s	1554 kbit/s
6	2728	342	525	1436 kbit/s	1537 kbit/s

Figure 16: packETHcli capture tap1 execution

Set up pings

- \$ ping -I tap0 13.0.0.2
- \$ ping -I tap1 13.0.0.2

A terminal window titled 'albert@albert-VirtualBox: ~'. The command 'ping -I tap0 13.0.0.2' has been executed. The output shows the first six successful ping replies from 13.0.0.2 to 11.0.0.1 via tap0, with 56(84) bytes of data and varying round-trip times between 25.9 ms and 55.4 ms.

```
albert@albert-VirtualBox:~$ ping -I tap0 13.0.0.2
PING 13.0.0.2 (13.0.0.2) from 11.0.0.1 tap0: 56(84) bytes of data.
64 bytes from 13.0.0.2: icmp_seq=1 ttl=254 time=55.4 ms
64 bytes from 13.0.0.2: icmp_seq=2 ttl=254 time=35.8 ms
64 bytes from 13.0.0.2: icmp_seq=3 ttl=254 time=33.1 ms
64 bytes from 13.0.0.2: icmp_seq=4 ttl=254 time=42.5 ms
64 bytes from 13.0.0.2: icmp_seq=5 ttl=254 time=25.9 ms
64 bytes from 13.0.0.2: icmp_seq=6 ttl=254 time=32.1 ms
```

Figure 17: Ping through tap0 interface

A terminal window titled 'albert@albert-VirtualBox: ~'. The command 'ping -I tap1 13.0.0.2' has been executed. The output shows the first ping attempt from 13.0.0.2 to 12.0.0.1 via tap1, with 56(84) bytes of data. The ping is shown as failed, with a single line of output and a cursor following.

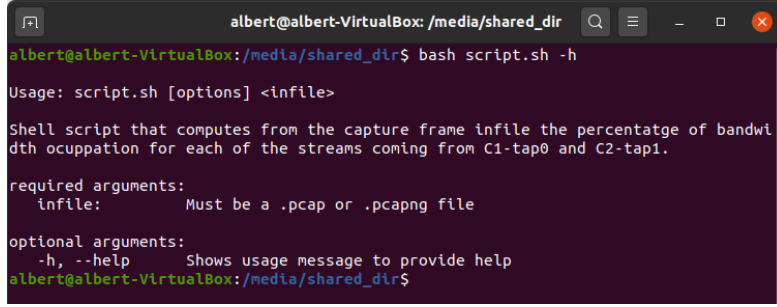
```
albert@albert-VirtualBox:~$ ping -I tap1 13.0.0.2
PING 13.0.0.2 (13.0.0.2) from 12.0.0.1 tap1: 56(84) bytes of data.
_
```

Figure 18: Ping through tap1 interface

As we can see in figure 18, the ICMP² ping replies are not received, because of the 20% of the bandwidth allocated.

²ICMP: Internet Control Message Protocol

4.2 Results



```

albert@albert-VirtualBox: /media/shared_dir
albert@albert-VirtualBox:/media/shared_dir$ bash script.sh -h
Usage: script.sh [options] <infile>

Shell script that computes from the capture frame infile the percentatge of bandwi
dth occupation for each of the streams coming from C1-tap0 and C2-tap1.

required arguments:
  infile:          Must be a .pcap or .pcapng file

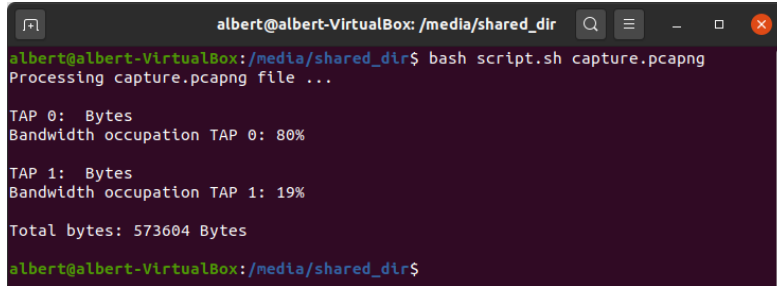
optional arguments:
  -h, --help      Shows usage message to provide help
albert@albert-VirtualBox:/media/shared_dir$

```

Figure 19: Problem 2 shell script usage

The following results have been obtained when the traffic has been captured for a enough period of time:

Interface	Bytes transmitted (B)	BW occupation (%)
tap0	460724	80%
tap1	112880	19%
Total Bytes	573604	



```

albert@albert-VirtualBox: /media/shared_dir
albert@albert-VirtualBox:/media/shared_dir$ bash script.sh capture.pcapng
Processing capture.pcapng file ...

TAP 0: Bytes
Bandwidth occupation TAP 0: 80%

TAP 1: Bytes
Bandwidth occupation TAP 1: 19%

Total bytes: 573604 Bytes
albert@albert-VirtualBox:/media/shared_dir$

```

Figure 20: Results problem 2 script execution

As we can see the results are consistent since approximately 80% of the total traffic has its origin in *tap0* and the remaining 20% has its origin in *tap1*. The bandwidth obviously has the same proportion.

$$\text{Ratio} = \frac{\text{Bytes}(\text{tap0})}{\text{Bytes}(\text{tap1})} = \frac{460724}{112880} \approx 4.08 \approx 4 = \frac{80}{20} \quad (1)$$

5 Problem 3

Answer several questions, considering the following topology:

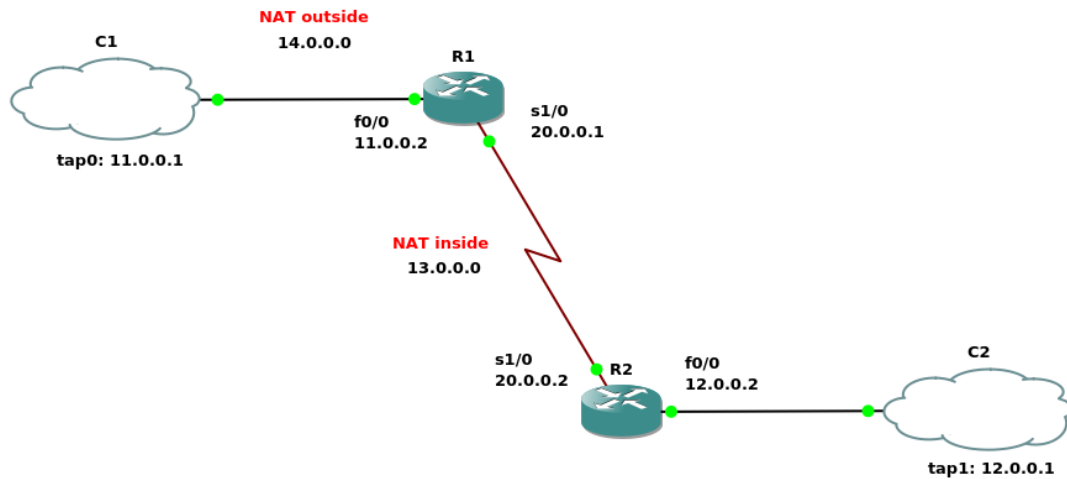


Figure 21: Problem 3 simulation topology scenario

5.1 Setup

This problem aims to demonstrate how Network Address Translation (NAT) works which is a way to map multiple local private addresses to a public one before transferring the information.

As can be seen, there are two *tap* interfaces, each one connected to a router which in turn are interconnected through a serial link.

But, in addition there is a NAT configuration in the R1 router which characterise the behaviour of the whole topology and the internal 20.0.0.0 network.

5.1.1 Host computer

Tap interfaces configuration

Following the same configuration as in problem 1 and 2. You can consult the commands in the table 1 on page 4, as well as the terminal execution example in the figure 3.

Network configuration

	Commands
Add Ip Route Gateways	<pre>sudo route add -net 13.0.0.0/24 gw 11.0.0.2 dev tap0 sudo route add -net 14.0.0.0/24 gw 12.0.0.2 dev tap1</pre>

Table 7: Network routing configuration

```
albert@albert-VirtualBox:~$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default _gateway 0.0.0.0 UG 100 0 0 enp0s3
10.0.2.0 0.0.0.0 255.255.255.0 U 100 0 0 enp0s3
11.0.0.0 0.0.0.0 255.255.255.0 U 0 0 0 tap0
12.0.0.0 0.0.0.0 255.255.255.0 U 0 0 0 tap1
13.0.0.0 11.0.0.2 255.255.255.0 UG 0 0 0 tap0
14.0.0.0 12.0.0.2 255.255.255.0 UG 0 0 0 tap1
20.0.0.0 11.0.0.2 255.255.255.0 UG 0 0 0 tap0
link-local 0.0.0.0 255.255.0.0 U 1000 0 0 enp0s3
192.168.122.0 0.0.0.0 255.255.255.0 U 0 0 0 virbr0
albert@albert-VirtualBox:~$
```

Figure 22: Host ip/kernel routing table

Might be also used the “*ip route*” command to check the ip table configuration as done in the previous problems. But in this case, for a better configuration view, was bet for using the first.

5.1.2 Default routers configuration

Router 1

	Commands
Interface FastEthernet 0/0	<pre>ip address 11.0.0.2 255.255.255.0 ip nat outside ip virtual-reassembly no shutdown duplex half</pre>
Interface Serial 1/0	<pre>ip address 20.0.0.1 255.255.255.0 ip nat inside ip virtual-reassembly serial restart-delay 0 no shutdown</pre>

Table 8: R1 router configuration commands

Router 2

	Commands
Interface Fast Ethernet 0/0	ip address 12.0.0.2 255.255.255.0 no shutdown duplex half
Interface Serial 1/0	ip address 20.0.0.2 255.255.255.0 serial restart-delay 0 no shutdown
Route configuration	ip route 11.0.0.0 255.255.255.0 20.0.0.1 ip route 13.0.0.0 255.255.255.0 12.0.0.1 ip route 14.0.0.0 255.255.255.0 11.0.0.1

Table 9: R2 router configuration commands

5.2 Questions

(Q1) Detail the required NAT related R1 configuration as well as the correct routing directives on your PC. Note that video stream must be delivered from 11.0.0.1 to 13.0.0.1

In first instance, what needs to be done is to define on our host computer IP table configuration the routes that represent the two NAT (inside/outside):

	Commands
R1 route configuration	ip route 12.0.0.0 255.255.255.0 20.0.0.2 ip route 13.0.0.0 255.255.255.0 12.0.0.1 ip route 14.0.0.0 255.255.255.0 11.0.0.1

Table 10: R1 router ip route configuration

Then for R1 router NAT, must be defined the external network as the one connected to the *tap1* interface, and the external one as the one connected to the *tap0* interface.

	Commands
R1 NAT configuration	ip nat inside source static 12.0.0.1 13.0.0.1 ip nat outside source static 11.0.0.1 14.0.0.1

Table 11: R1 router NAT configuration

```
!  
ip route 12.0.0.0 255.255.255.0 20.0.0.2  
ip route 13.0.0.0 255.255.255.0 12.0.0.1  
ip route 14.0.0.0 255.255.255.0 11.0.0.1  
!  
no ip http server  
no ip http secure-server  
!  
ip nat inside source static 12.0.0.1 13.0.0.1  
ip nat outside source static 11.0.0.1 14.0.0.1  
!
```

Figure 23: R1 router NAT and Route configuration

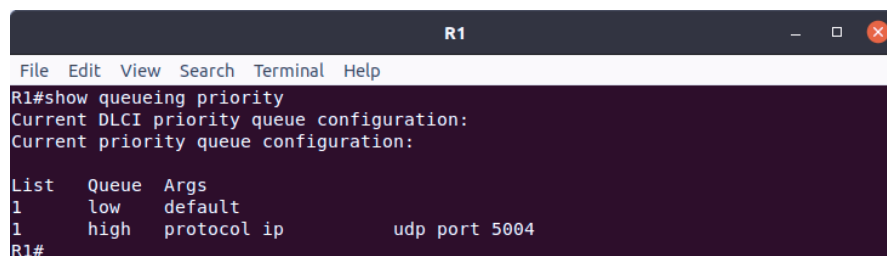
(Q2) Detail the required priority queuing (PQ) related R1 configuration in order to provide high priority to video streaming and low priority to default traffic. Which is the maximum allowed speed transmission rate along the path ?

The first thing to do is to define the priority group on the serial link interface of the R1 router:

- priority-group 1

To set the priority of the video (for the group), it has been established that IP traffic using User Datagram Protocol (UDP) through port 5004 has a high priority 2. Otherwise, the rest of the traffic has low priority.

- priority-list protocol ip high udp 5004
- priority-list 1 default low



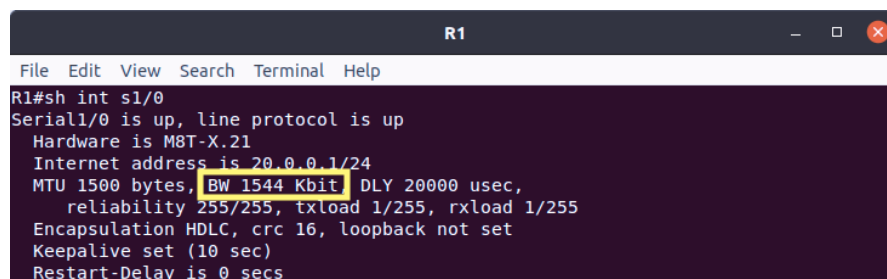
```
R1
File Edit View Search Terminal Help
R1#show queueing priority
Current DLCI priority queue configuration:
Current priority queue configuration:

List  Queue  Args
1     low   default
1     high  protocol ip      udp port 5004
R1#
```

Figure 24: R1 router queueing priority configuration

With respect to the maximum transmission speed along the rotation is 1544 kbps. It has been obtained by executing the following command in R1 for the serial link interface:

- sh int s1/0




```
R1
File Edit View Search Terminal Help
R1#sh int s1/0
Serial1/0 is up, line protocol is up
  Hardware is M8T-X.21
  Internet address is 20.0.0.1/24
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation HDLC, crc 16, loopback not set
  Keepalive set (10 sec)
  Restart-Delay is 0 secs
```

Figure 25: R1 router serial link 1/0 bandwidth

(Q3) Download from [here](#) a video sample with the corresponding video bit rate encoding slightly below your maximum transmission rate. Detail the complete ping to achieve the same transmission rate that your video streaming.

The video that has been downloaded from the indicated link is *Big Buck Bunny* with resolution of 640x360 (562 kpbs, Video bit rate).



Big Buck Bunny

[Encode CodecWorks](#)
[Decode MPEG Player](#)
[Analyze StreamEye](#)

[Duration, minutes: 9:56](#)

Resolution	Format	Video bit rate	Recommended Hardware	Size	Link
640x360	MPEG2 TS, HEVC+AAC	562 kbps	4-core Intel i7 2.5 GHz	61.7Mb	Download

Figure 26: Video sample downloaded

In order to create a ping with a similar bit rate, the following command was used on the host computer terminal:

- `$ sudo ping 13.0.0.1 -s 7025 -i 0,1`

where *-s* refers to the packet size (data bytes to be send), and *-i* the wait interval seconds between sending each packet.

With this we assure sending 7025 bytes every 0.1 second, which is equivalent to sending 562kb every second.

(Q4) While streaming video, measure the number of RTP missed packets (during 1 minute of transmission) by ffplayer when using PQ and without PQ. Explain how you obtain those measurements.

To generate the traffic flow we used *ffmpeg*³, a multimedia framework to record, convert, and stream audio and video via CLI.

Once installed this software, to transmute the video to the 13.0.0.1 ip destination over the UDP 5004 port was used:

- `$ ffmpeg -re -i <video infile> -vcodec copy -an -sdp_file <outfile .sdp> -f rtp rtp://13.0.0.1:5004`

Further, for measuring the missed packets a portable media aplayer included into the *ffmpeg* libraries called *ffplay* was used.

In this way, was recorded the transmission during 1 minute and got the frames written into the corresponding log file, all by using

- `ffplay -protocol_whitelist file,udp,rtp -i <file .sdp> 2> <log file>`

Finally, the counted lost packets for each case stand as

Configuration	Packets lost
PQ	1385
No PQ	1374

Table 12: Packet lost by cases

Observing that there is not much difference between both configurations when the channel is completely free.

³<http://ffmpeg.org>

(Q5) Repeat the last item adding a ping transmission at the same rate that the video streaming.

Now, let's procedure the same way but adding a ping transmission using:

- `$ sudo ping 13.0.0.1 -s 7025 -i 0,1`

The obtained results were:

Configuration	Packets lost
PQ	1647
No PQ	2490

Table 13: Packet lost by cases adding a ping transmission

Clearly, is shown the *priority-queueing* configuration manage way better the situation when there is congestion traffic.

References

- [1] [Quality of Service - Communication Services and Security - César Fernández Camón](#)
- [2] [GNS3 — The software that empowers network professionals](#)
- [3] [GNS3 — Community](#)
- [4] [GNS3-0.5-tutorial.pdf](#)
- [5] [Wireshark · Go Deep](#)
- [6] [tshark - The Wireshark Network Analyzer 3.4.4](#)
- [7] [Why is ip.tos.precedence empty?](#)
- [8] [PackETH Repository by nickrobinson](#)
- [9] [PackETH home web page](#)