iu-ne-lab-05-Iskander_Nafikov

- **Name**: Iskander Nafikov
- **E-mail**: i.nafikov@innopolis.university
- **Username**: iskanred
- **Hostname**: `lenovo`

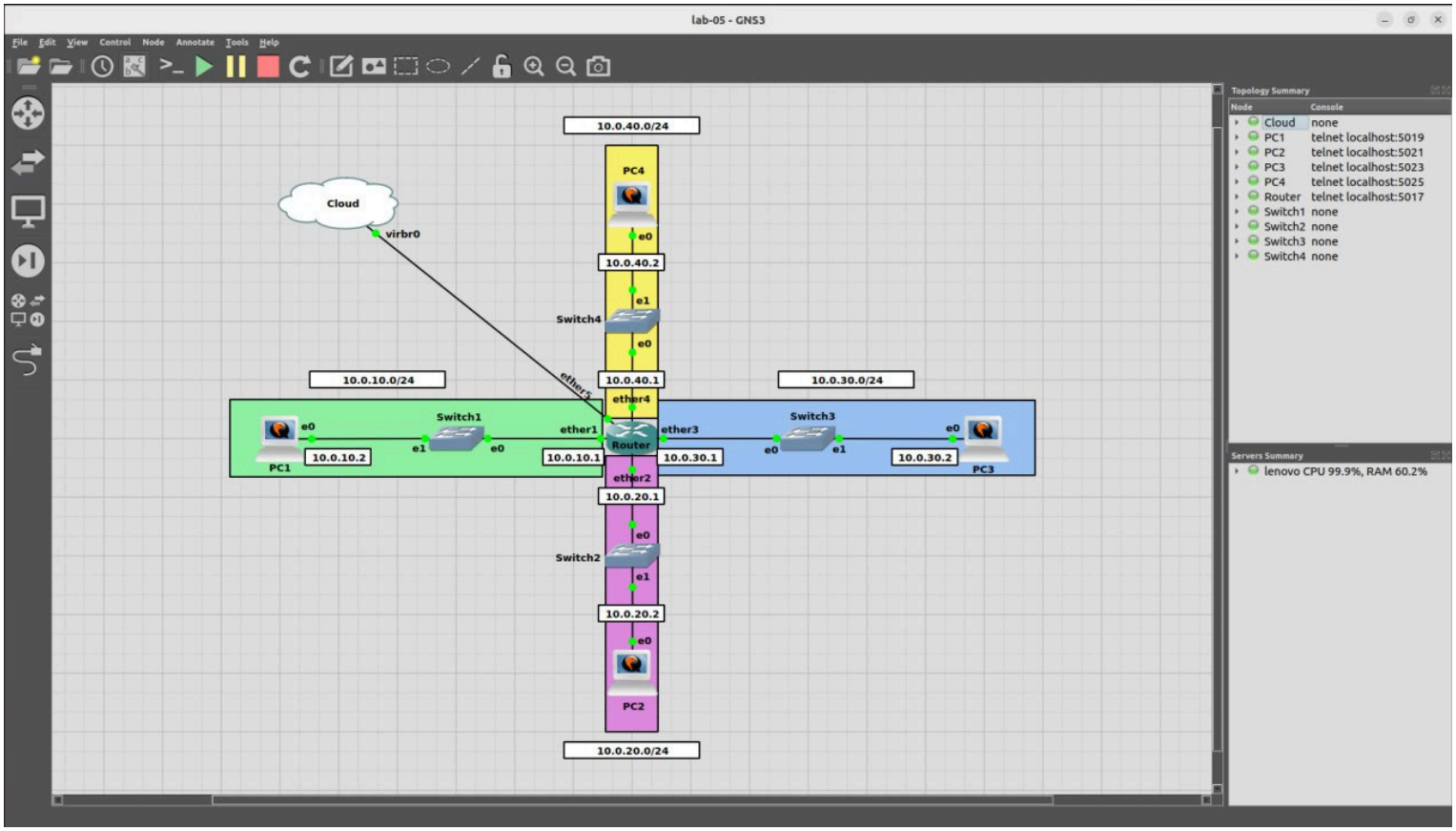
Task 1 - Prepare your network topology

> **1.** In the GNS3 project, select and install a virtual routing solution that you would like to use: Mikrotik (recommended), Pfsense, vyos and so on.`

I decided to use Mikrotik as recommended - `Mikrotik CHR 7.16`

> **2.** Prepare a simple network consisting of at least one router and two hosts. About four hosts in the network are most optimal. You also might need Internet access for the hosts. It may be something like this, just for the imagination:

I configured the following network. The internet access from PCs was configured using NAT (`ip firewall nat`).



Task 2 - QoS learning & configuring

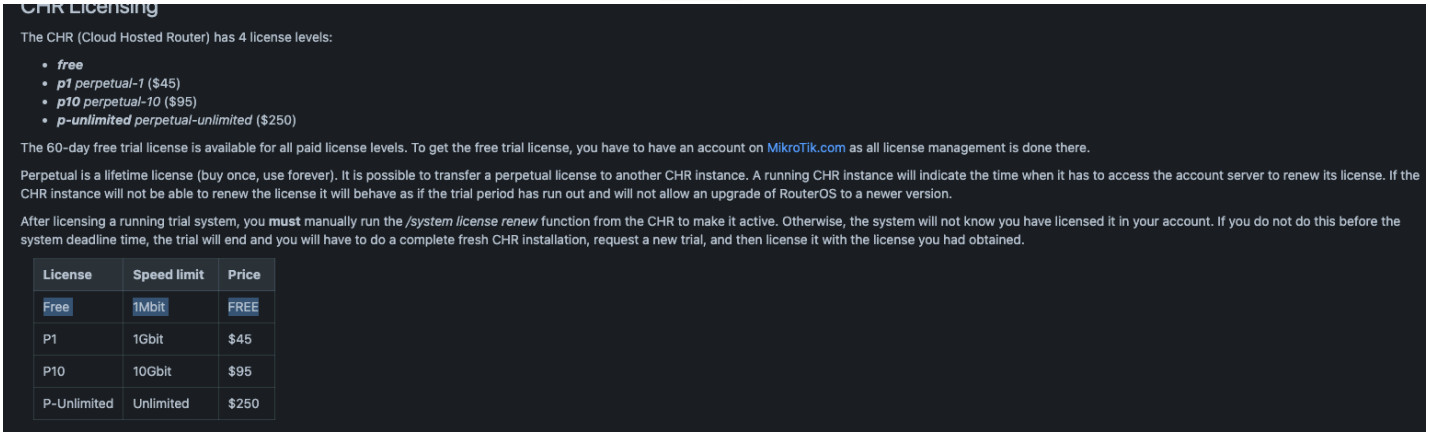
> **1.** Let's start with a little theory. Briefly answer the questions or give one-line description what is it: Class of Service (CoS), ToS (Type Of Service), Differentiated Services Code Point (DSCP), Serialization, Packet Marking, Tail Drop, Head Drop, The Leaky bucket algorithm, The Token Bucket Algorithm, Traffic shaping, Traffic policing

- **Class of Service (CoS)**: A mechanism for prioritizing network traffic by classifying packets into different categories for better Quality of Service (QoS), and it is implemented in Link Layer having special 3-bit field in 802.1Q header.
- **Type of Service (ToS)**: An older field in the IP header used to indicate the priority and treatment of packets in terms of delay, throughput, and reliability.
- **Differentiated Services Code Point (DSCP)**: A field in the IP header used for marking packets to indicate the desired level of service, enabling QoS strategies.
- **Serialization**: The process of converting data structures or objects into a format suitable for transmission over a network.
- **Packet Marking**: The practice of labeling packets with specific information (like priority) to influence their treatment as they traverse the network.
- **Tail Drop**: A queuing discipline where packets are dropped from the end of the queue when it reaches its maximum capacity.
- **Head Drop**: A queuing discipline where packets are dropped from the front of the queue, which can be used in certain network management strategies, though it's less common than tail drop.
- **The Leaky Bucket Algorithm**: A traffic shaping method that controls data output rates by allowing data to "leak" out at a constant rate, smoothing bursts.
- **The Token Bucket Algorithm**: A traffic shaping technique that allows for bursts of data while maintaining a defined average rate by using tokens to regulate packet transmission.
- **Traffic Shaping**: The technique of controlling the volume of traffic being sent into a network to optimize or guarantee performance, improve latency, or manage bandwidth.
- **Traffic Policing**: A method of monitoring network traffic and enforcing bandwidth limits by dropping or marking packets that exceed the agreed rate.

> **2.** Configure your network as you decided above. After your network is configured (don't forget to show the main configuration steps in the report), try to set a speed limitation (traffic shaping) between the two hosts.

I configured my network in the **Task 1**. So now, let define the speed limits.

1. But before, let's investigate what is the current maximum speed without any explicitly set limits.
   - First, it is written in Mikrotik documentation Cloud Host Router (CHR), which is "a RouterOS version intended for running as a virtual machine", has a speed limit `1Mbit` for a free version.



CHR Licensing

The CHR (Cloud Hosted Router) has 4 license levels:

- ***free***
- ***p1*** *perpetual-1* ($45)
- ***p10*** *perpetual-10* ($95)
- ***p-unlimited*** *perpetual-unlimited* ($250)

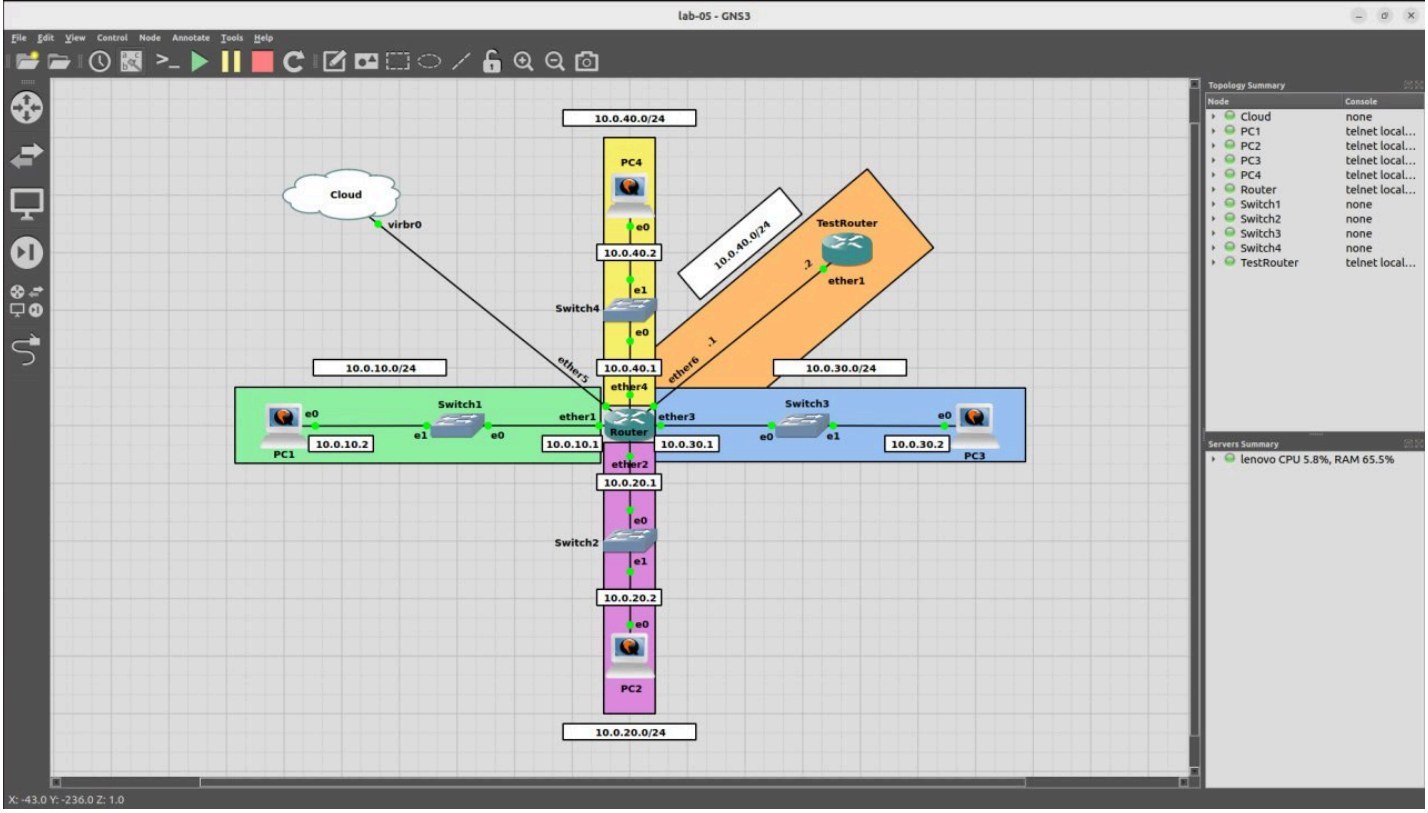
The 60-day free trial license is available for all paid license levels. To get the free trial license, you have to have an account on MikroTik.com as all license management is done there.

Perpetual is a lifetime license (buy once, use forever). It is possible to transfer a perpetual license to another CHR instance. A running CHR instance will indicate the time when it has to access the account server to renew its license. If the CHR instance will not be able to renew the license it will behave as if the trial period has run out and will not allow an upgrade of RouterOS to a newer version.

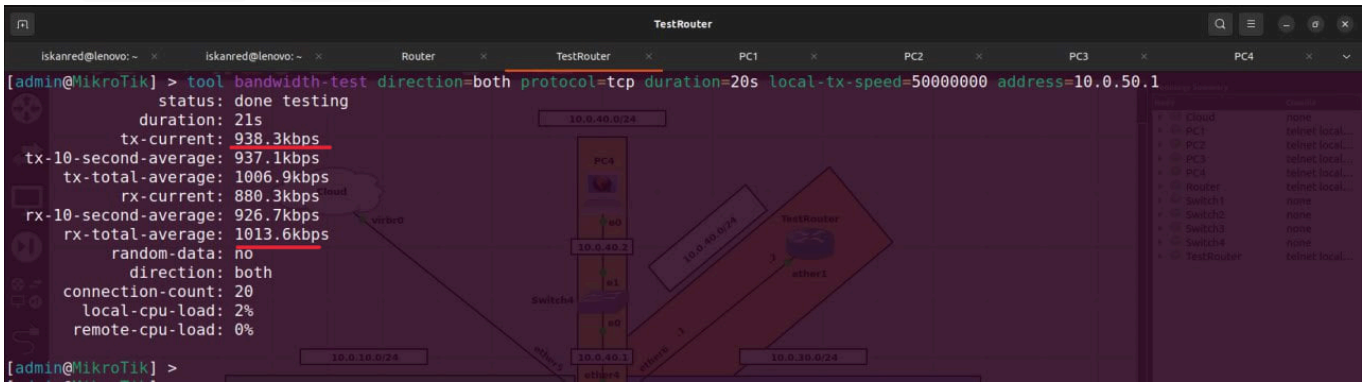
After licensing a running trial system, you **must** manually run the */system license renew* function from the CHR to make it active. Otherwise, the system will not know you have licensed it in your account. If you do not do this before the system deadline time, the trial will end and you will have to do a complete fresh CHR installation, request a new trial, and then license it with the license you had obtained.

License	Speed limit	Price
Free	1Mbit	FREE
P1	1Gbit	$45
P10	10Gbit	$95
P-Unlimited	Unlimited	$250

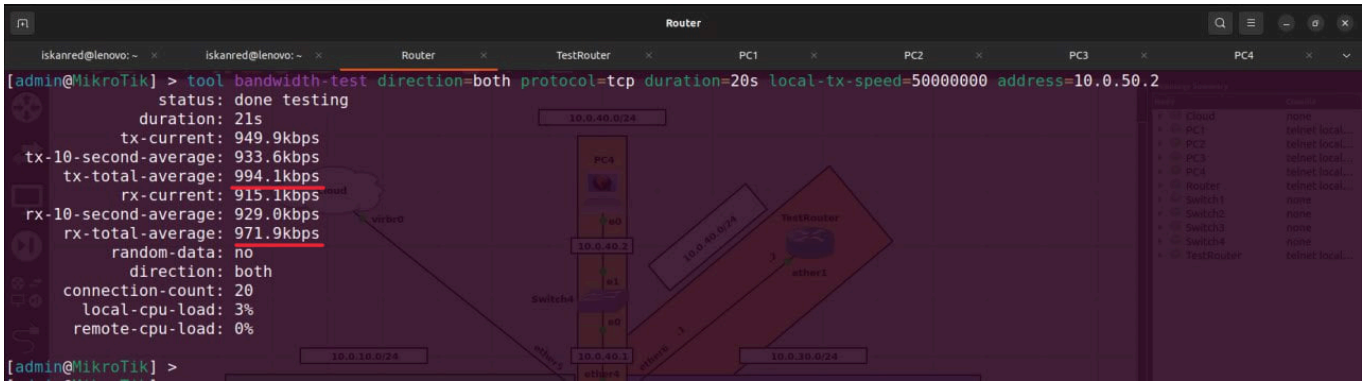
   - Let's verify it
   - To compute the bandwidth of the router and/or of a link, I added another one router `TestRouter` and connect it to my main one named `Router` in a new LAN `10.0.50.0/24`



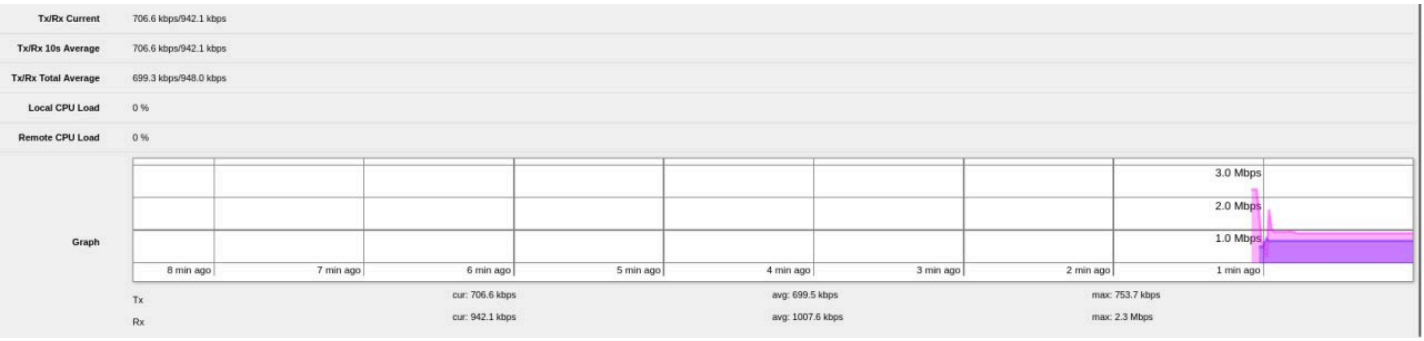
- I did it to run a Mikrotik tool called [bandwidth-tester] which requires a server and a client running on Mikrotik device. Let's run speed tests
  - From `TestRouter` to `Router`



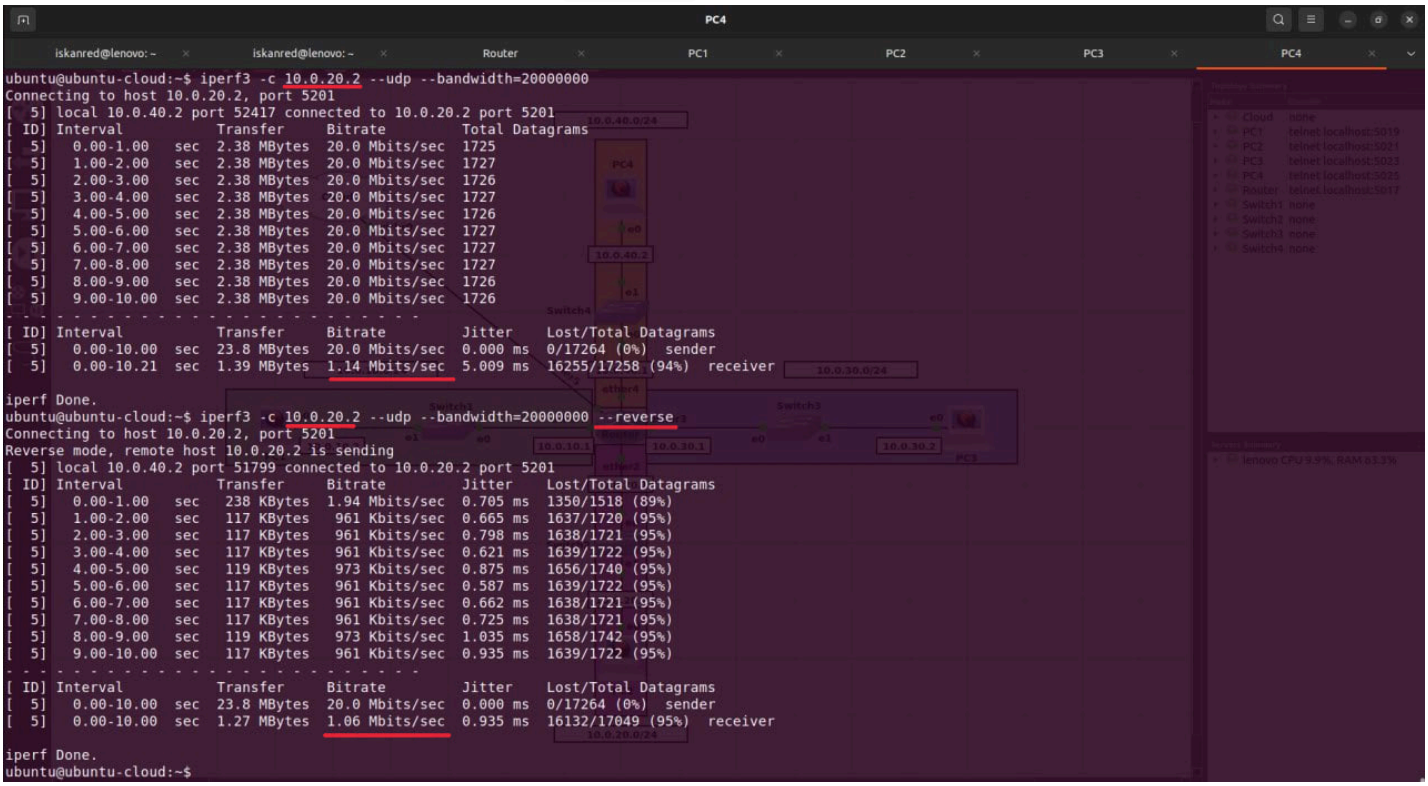
  - From `Router` to `TestRouter`



- And the speed graph which I took from the Mikrotik Web

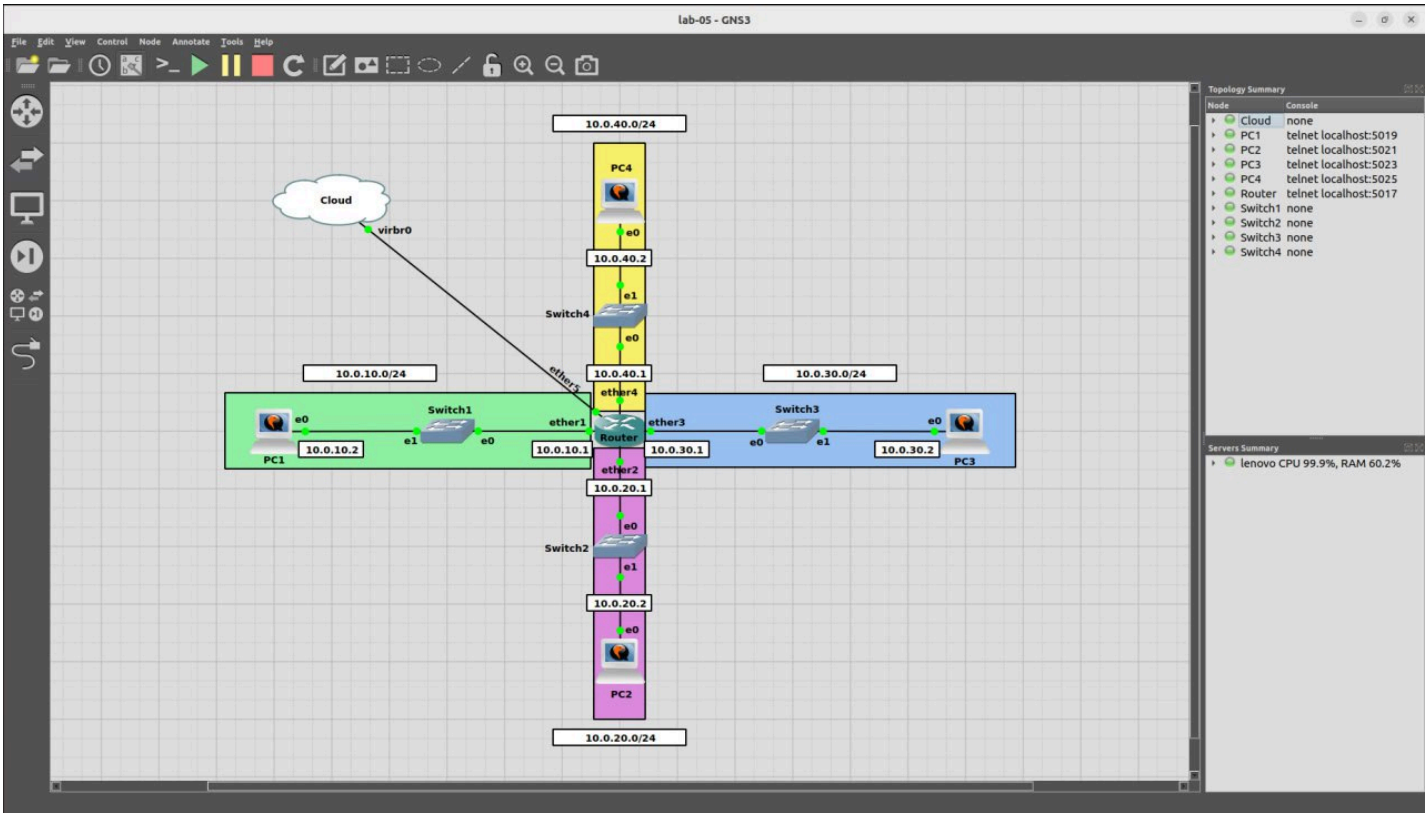


- To verify it completely I run performance test using `iperf` from `PC4` as a client to `PC2` as a server at this time using UDP to avoid losing sending or receiving speed due to TCP's congestion control and its bidirectional nature. To make it more convincing I ran performance testing **2** times: from the client (`PC4`) to the server (`PC2`) and vice a versa from the server to the client using an option `--reverse` (`--bidir` was not an option since it used to me wrong results). Also, I set `--bandwidth` equals to `20 Mbit/s` to be sure that the generating bitrate won't influence on the result (by default it is `1 Mbit/s` which may confuse).



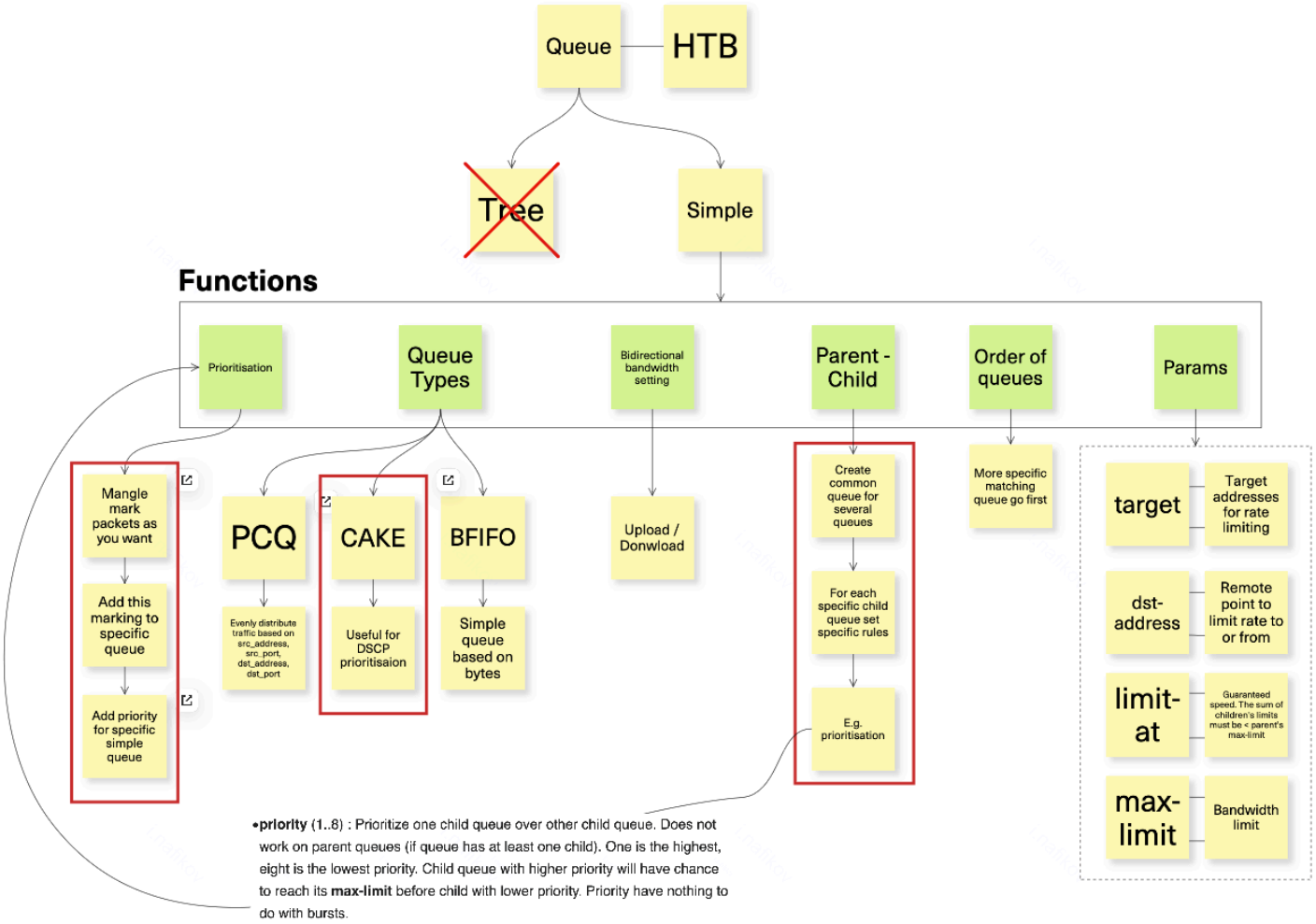
- We see that the bitrate is almost `1 Mbit`. Considering computational errors over this time interval it is easy to conclude that `1 Mbit` is a hard-limit.
- **Conclusion**: Finally, we can conclude that the 🔴 **limit actually exists** 🔴, it is configured to `1 Mbit` and it is configured inside a router's firmware. To increase the limit users should purchase paid versions. Therefore, let's continue using the free version and `1 Mbit` limit.

- I removed `TestRouter` and, thus, returned the topology's original image



2. Now let's set the speed limits using the traffic shaper on our router for the hosts `PC3` and `PC3`
   - I decided to apply traffic shaper to my Mikrotik device for connections between `PC1` - `PC3`.
   - To set traffic shaping or scheduling I had to use queues, so I decided to learn something about them and created such a graphical document for myself to structure my findings and keep my knowledge. I decided not to deep dive into `queue tree` or Hierarchical Token Buckets HTB topics because they are too difficult for me now. Therefore, I tried to keep detauls about `queue simple`. This block may not be detailed or totally correct,

but it is clear for me.

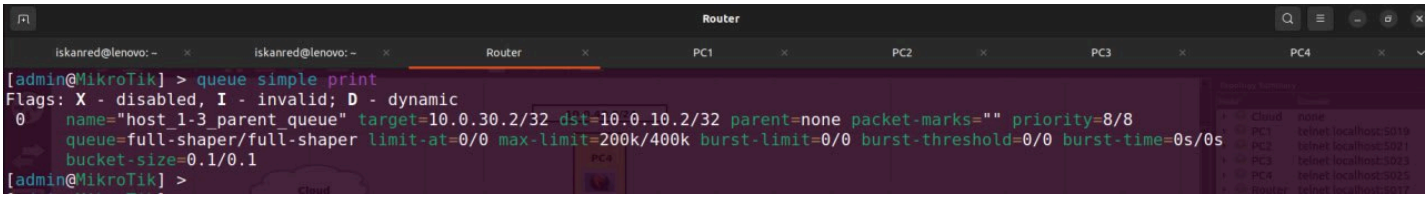


- For this purpose I firstly configured a new queue type `full-shaper` based on the `pfifo` (default packet FIFO queue) and set queue size (`pfifo-limit`) to 1 packet making it almost a full 100% shaper. This means that almost every packet that goes over the predefined limit will be dropped immediately. Therefore, it will cause a **big packet loss**, but **low-latency packets delivery**. I decided to use `pfifo` just because of its simplicity.

```
/queue type add name="full-shaper" kind=pfifo pfifo-limit=1
```

- Then I created a `simple` queue. I decided to use `queue simple` just for the sake of its simplicity. I configured a queue in the following format:

```
/queue simple add name="host_1-3_parent_queue" target="10.0.30.2" dst="10.0.10.2"
queue="full_shaper/full_shaper" max-limit="200K/400K"
```

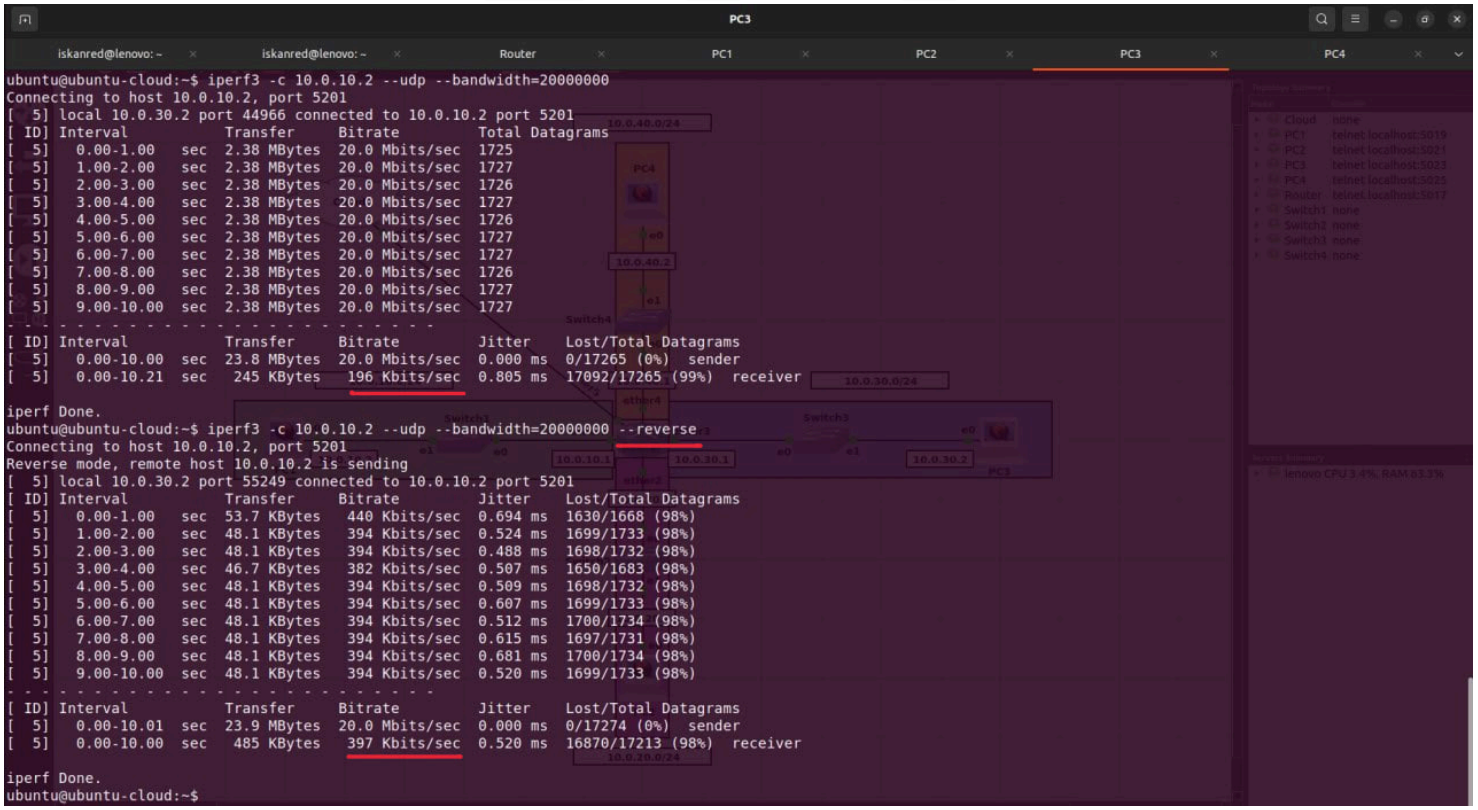


- As you can I configured a queue `host_1-3_parent_queue` for the link between `PC3` (`target=10.0.30.2`) and `PC1` (`dst=10.0.10.2`) with the queue type `queue=full-shaper` for both downloading and uploading

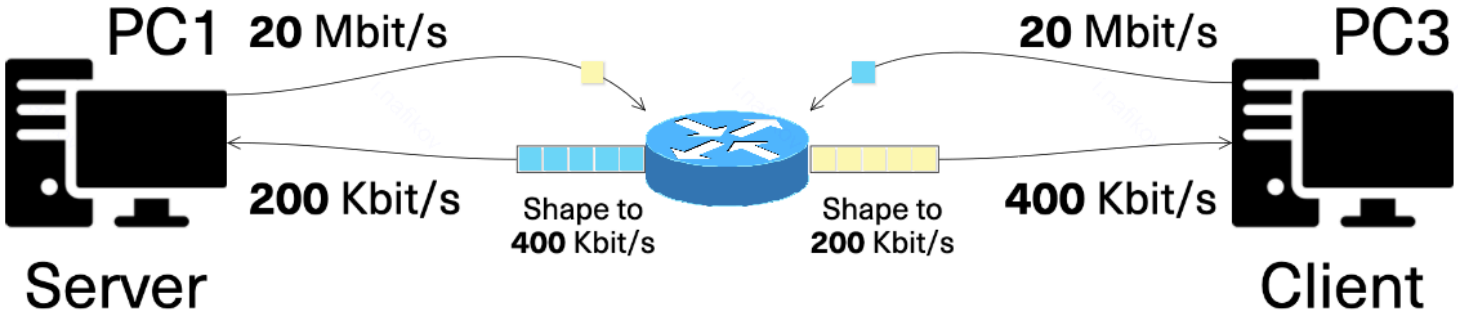
traffic and the actual max bandwidth of `200 Kbit/s` for uploading and of `400 Kbit/s` for downloading (from the client's perspective).

> **3.** Run a bandwidth testing tool, see what is the max speed you can get and verify your speed limitation. Compare the speed between the different hosts.

- Now let's check if it works by running `iperf3` again (`PC1` is a server, `PC3` is a client, UDP traffic)



- We see that it worked and the actual bitrate was definitely about `200 Kbit/s` and `400Kbit/s` for uploading and downloading correspondingly. The scheme of interaction look like:



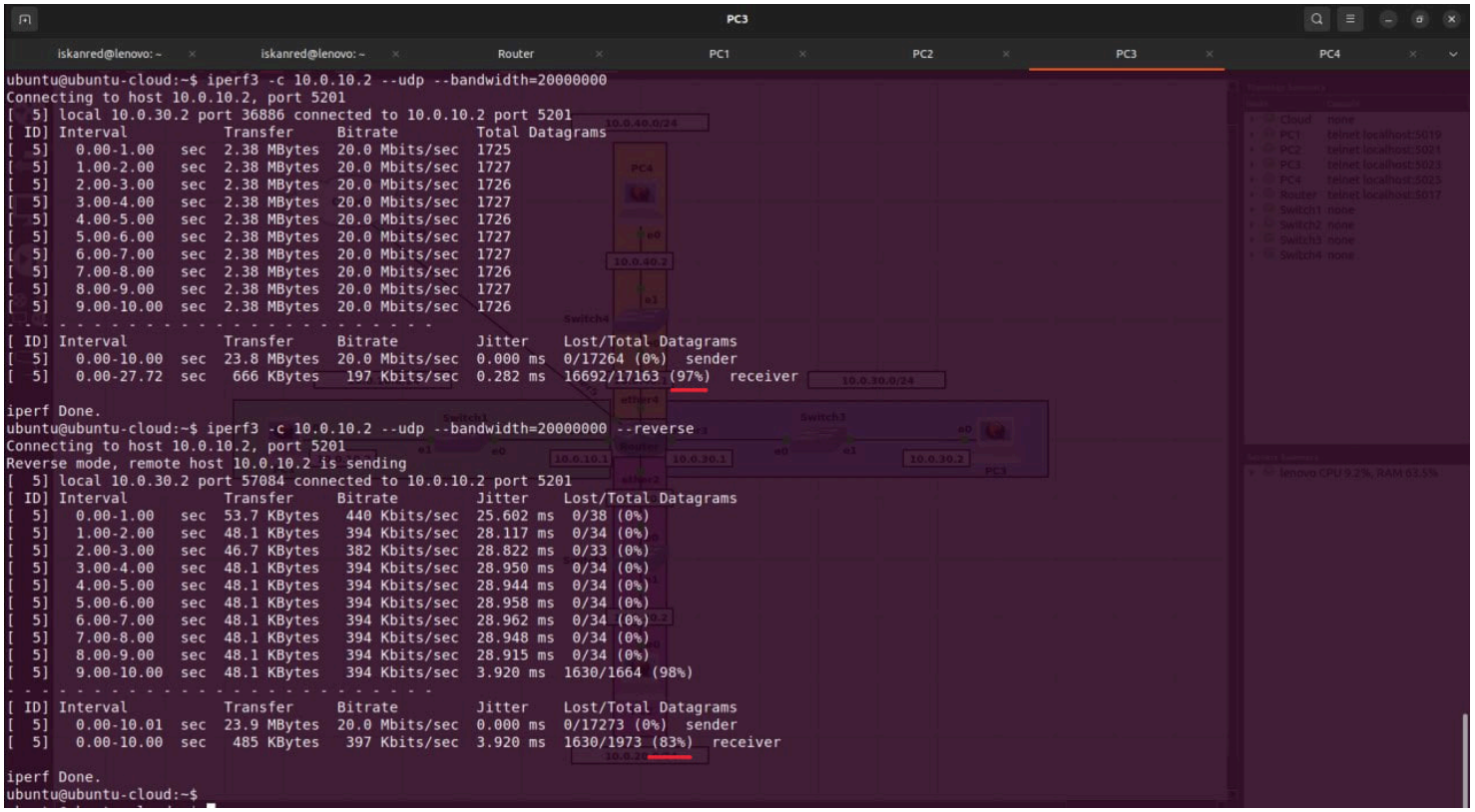
- Nevertheless, we can notice that the packet loss is enormous: **99**% for the client's packets and **98**% for the server's packets.

```
   Lost/Total Datagrams
   0/17265 (0%)  sender
   17092/17265 (99%)  receiver

20000000 --reverse

5201
   Lost/Total Datagrams
   1630/1668 (98%)
   1699/1733 (98%)
   1698/1732 (98%)
   1650/1683 (98%)
   1698/1732 (98%)
   1699/1733 (98%)
   1700/1734 (98%)
   1697/1731 (98%)
   1700/1734 (98%)
   1699/1733 (98%)

   Lost/Total Datagrams
   0/17274 (0%)  sender
   16870/17213 (98%)  receiver
```

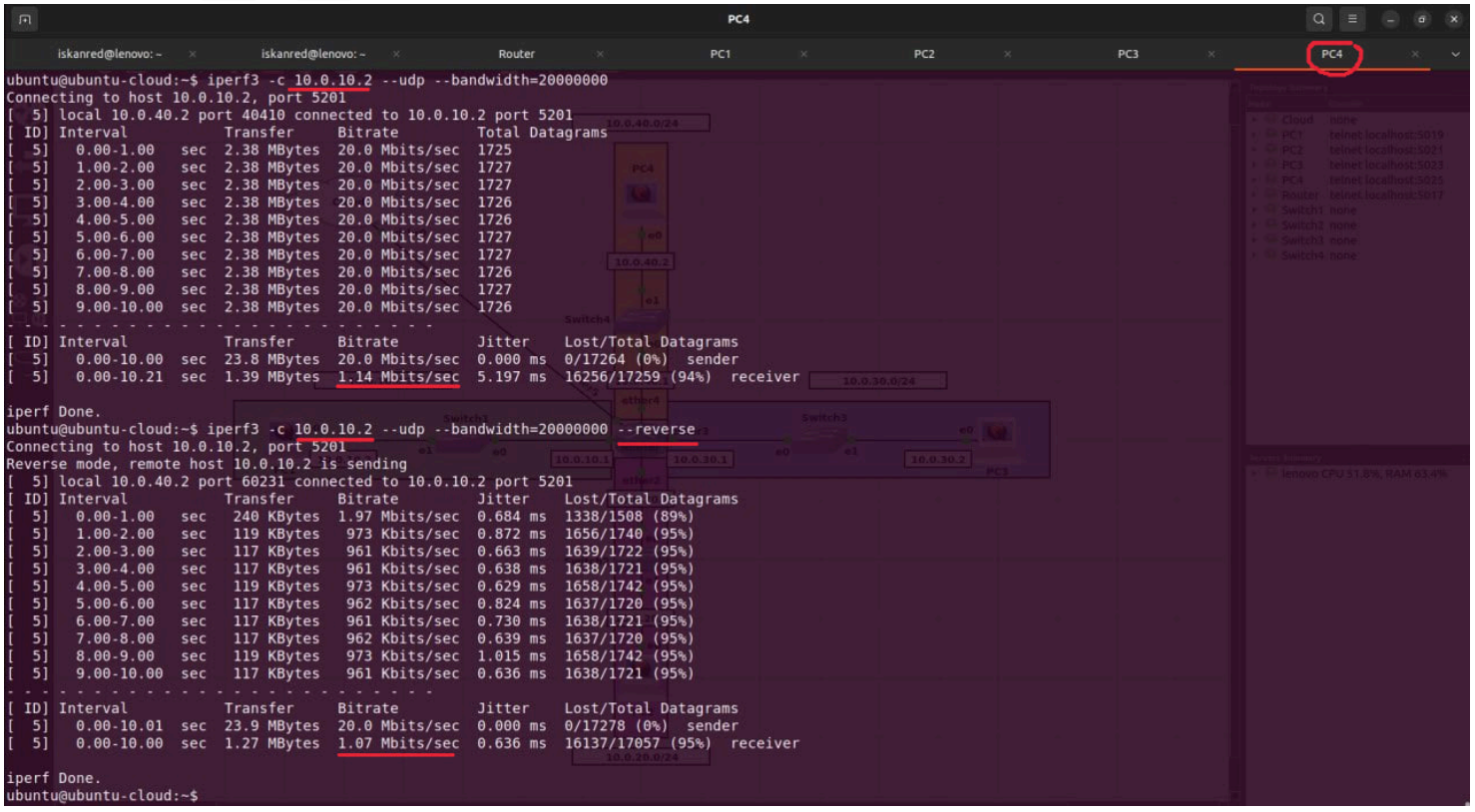
- Such number of lost datagrams happened because UDP is not reliable, especially when the bandwidth and the actual bitrate differ so much. Also, such big numbers occurred because of very aggressive traffic shaping which drops packets almost immediately with queue size = 1 packet.
- Therefore, we can tune it a bit and really slightly decrease packet loss configuring our shaper to be more a scheduler with the queue size = 300 packets.

```
/queue type set full-shaper pfifo-limit=300
```

- Afterward, we can notice this decrease on practice: 99% -> 97% 98% -> 83%

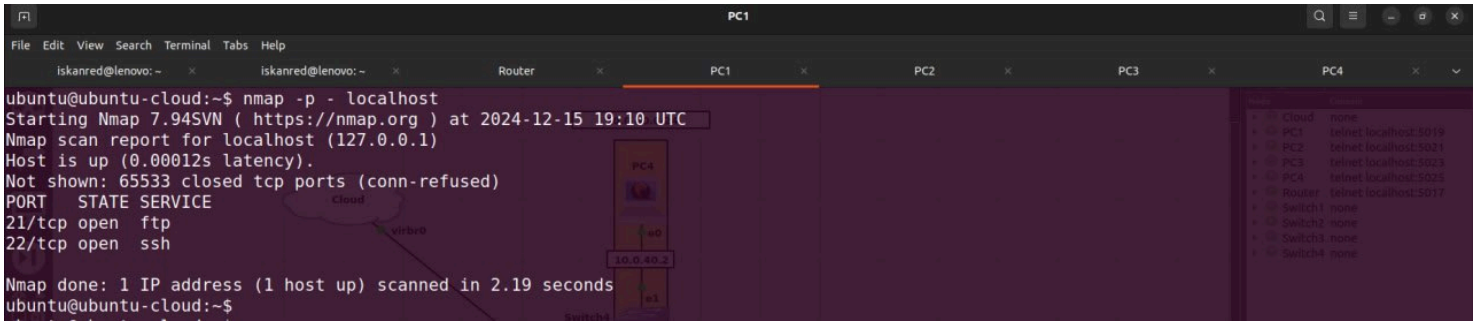


- However, the delay was increased also. I had to wait longer for the end of test.
- Still the speed for other clients remained the same = `1 Mbit/s` even for the same server `PC1`. For example for the connection `PC4` -- `PC1`:



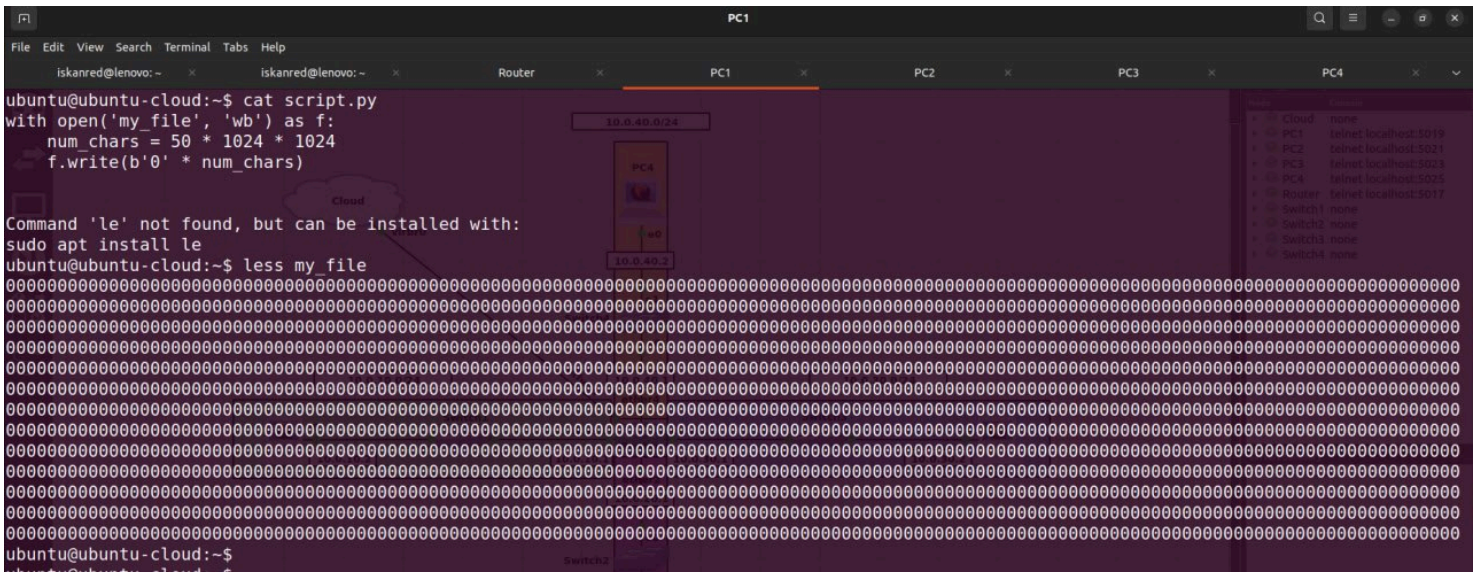
> **4.** While your bandwidth test is still running, try to download a file from one host to the other host and see what is the max speed you can get. If you have more than two hosts on the network, play around with different speed values and show it.

- To complete this task I set up an FTP server on `PC1`. I used pure-ftpd for FTP server

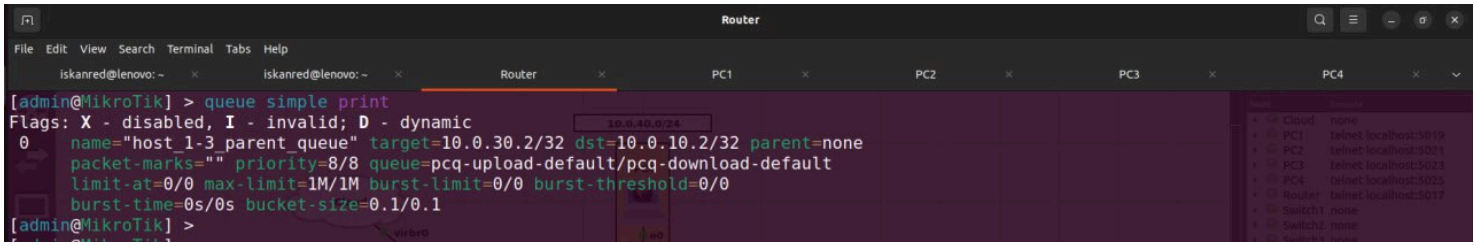


- Also I created a file of 50MB on `PC1` using this Python script:

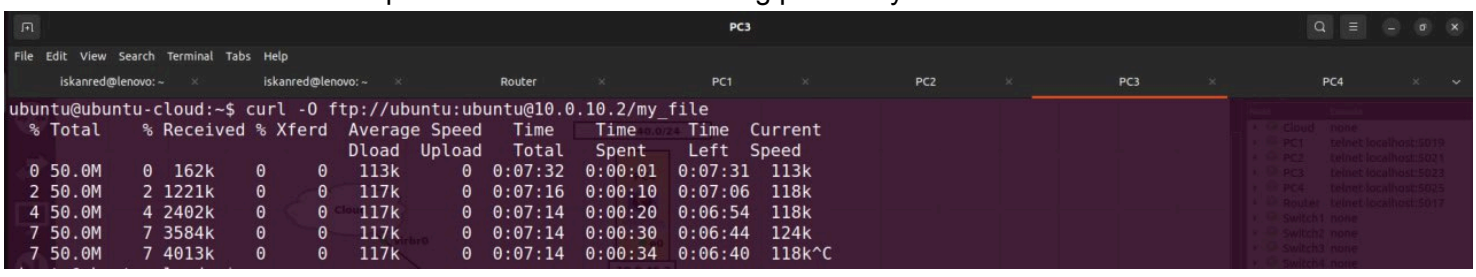
```
with open('my_file', 'wb') as f:
    num_chars = 1024 * 1024 * 1024
    f.write('0' * num_chars)
```



- Before all, I decided to update my queue and set the limit to `1 Mbit/s` for both download and upload. Also, I changed its type to `pcq-upload-default/pcq-download-default` just because I wanted it to try, but there should be no effect since `pcq-classifier` is `dst-address` and it will be the same for both the connections: file download and perf.



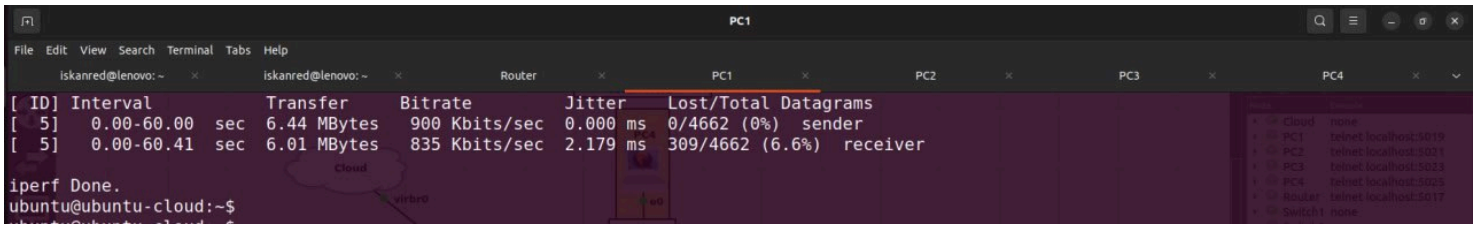
- Then I decided to collect the speed statistics without running perf firstly



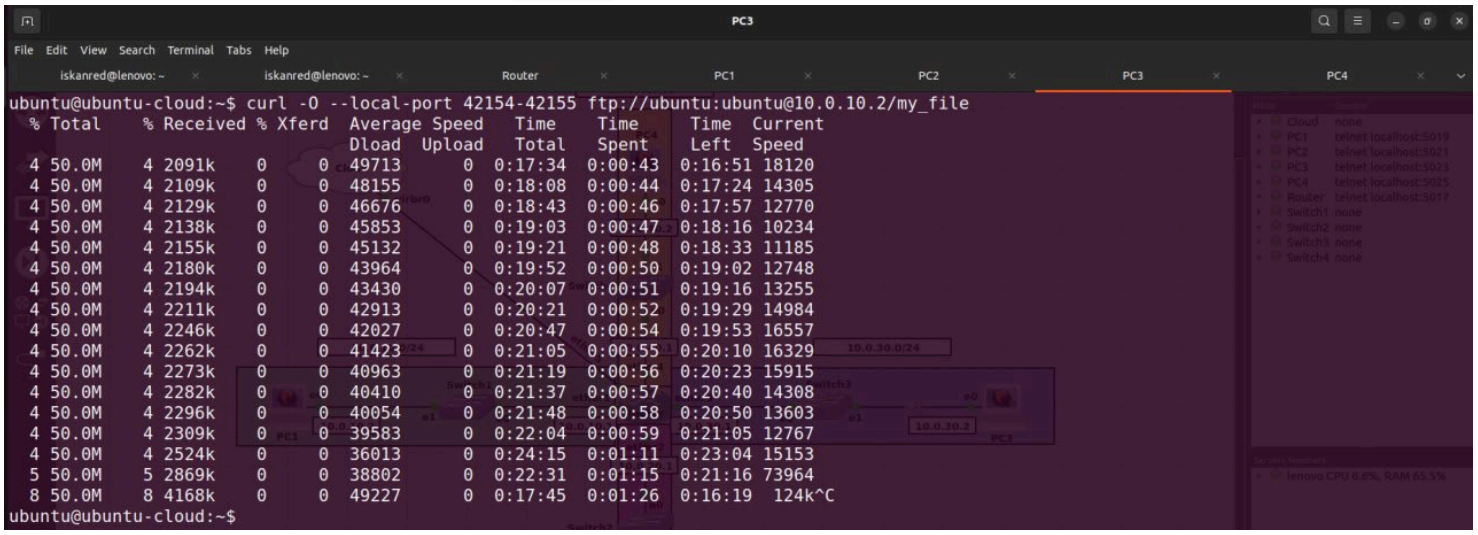
- We see that normally even with `1 Mbit/s` open channel the speed is only about `120 Kbit/s`. This can happen due to TCP traffic has different traffic guarantees which impact the final speed.

- Then I ran `iperf3` from `PC1` to `PC3` : UDP traffic with regular bandwidth = `900 Kbit/s` , not `1 Mbit/s` to leave some "space" for FTP traffic.

```
iperf3 -c 10.0.10.2 --bandwidth=300000 --udp --time=60
```



- Finally, I started FTP file download with `iperf3` running

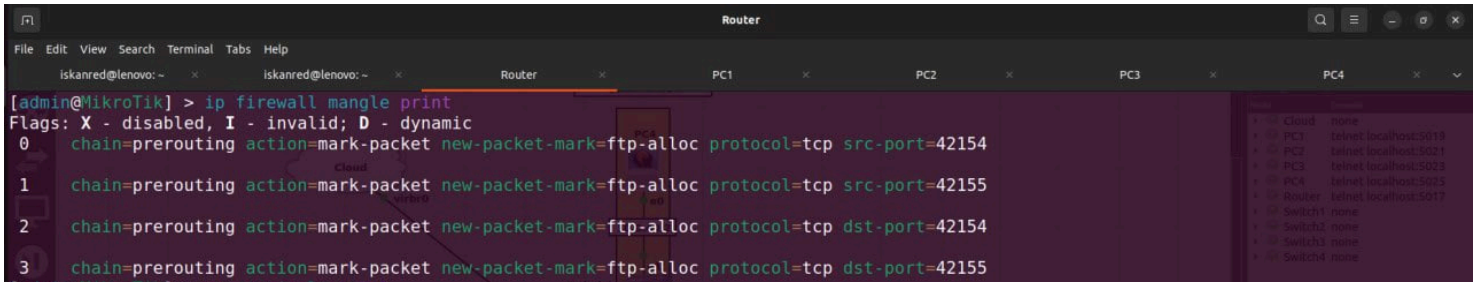


- We can notice that the speed **dropped dramatically** to approximately `15 Kbit/s` and returned to `120 Kbit/s` only after perf ended.

> **5.** Deploy and verify your QoS rules to prioritise the downloading of a file (or any other scenario) over the bandwidth test.

- Now let me show my QoS configuration. I use traffic allocation technique: `limit-at` , child queue, packet marks and etc. Let me show it now
- First, I configured my `ip firewall mangle` to mark all FTP-related packets as `ftp-alloc`

```
ip firewall mangle>
add chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp src-port=42154
add chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp src-port=42155
add chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp dst-port=42154
add chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp dst-port=42155
```

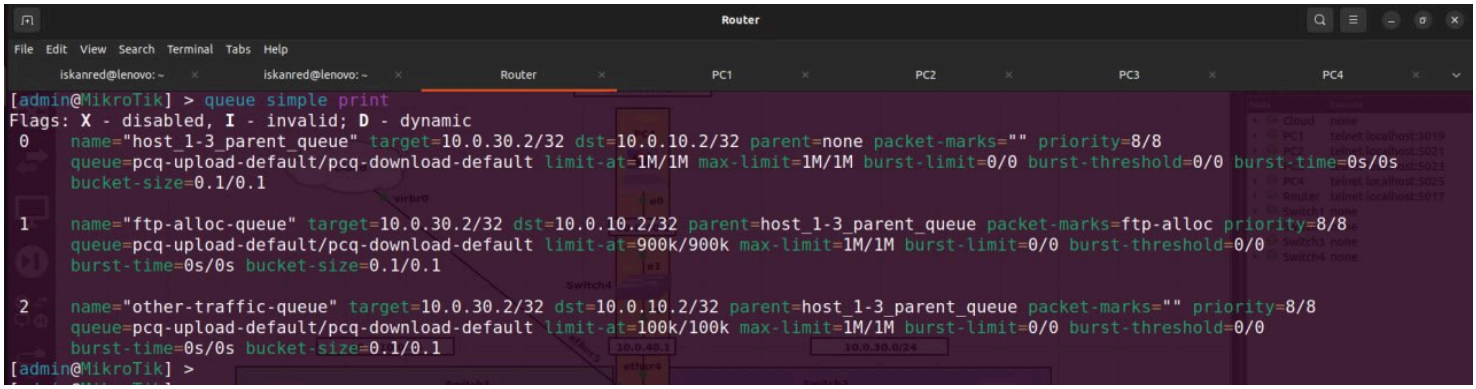


- As you can see I added packet marking on `prerouting` chain (Rules in this chain apply to packets as they just arrive on the network interface) on TCP protocol for different sets of `dst-port` and `src-port` . But why such

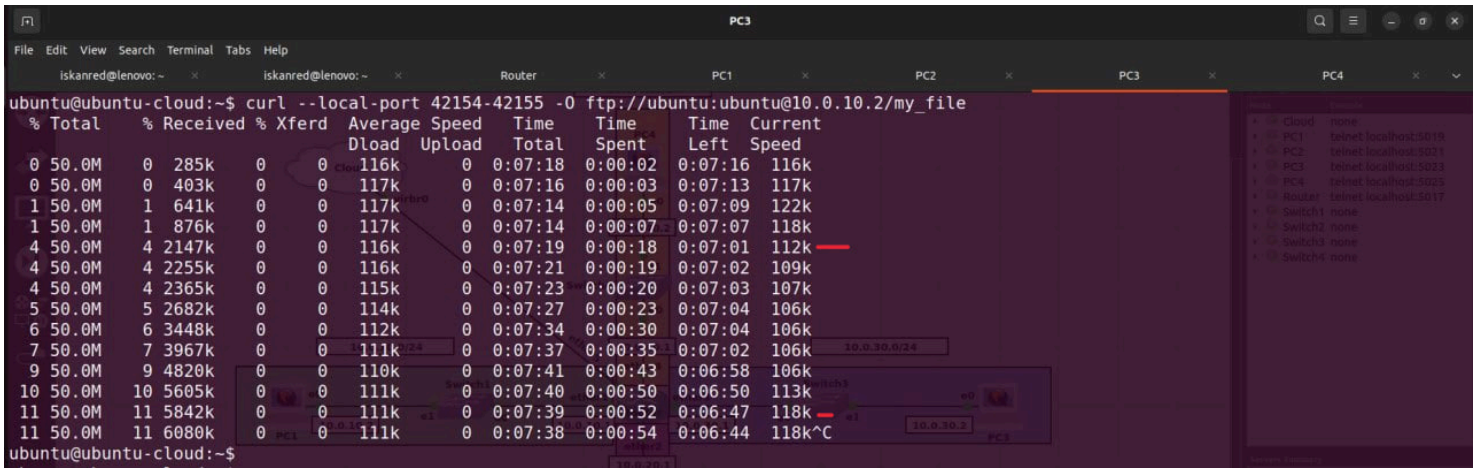
strange numbers: 42154, 42155? This is because I decided to run my FTP client on these ports (one port for a control channel and one for a data channel).

- Then, I added new queues that I made children of my main `host_1-3_parent_queue`.

```
queue simple>
add name=ftp-alloc-queue target=10.0.30.2 dst=10.0.10.2 parent=host_1-3_parent_queue packet-
marks=ftp-alloc queue=pcq-upload-default/pcq-download-default limit-at=900k/900k max-
limit=1M/1M
add name=ftp-alloc-queue target=10.0.30.2 dst=10.0.10.2 parent=host_1-3_parent_queue packet-
marks=ftp-alloc queue=pcq-upload-default/pcq-download-default limit-at=900k/900k max-
limit=1M/1M
```

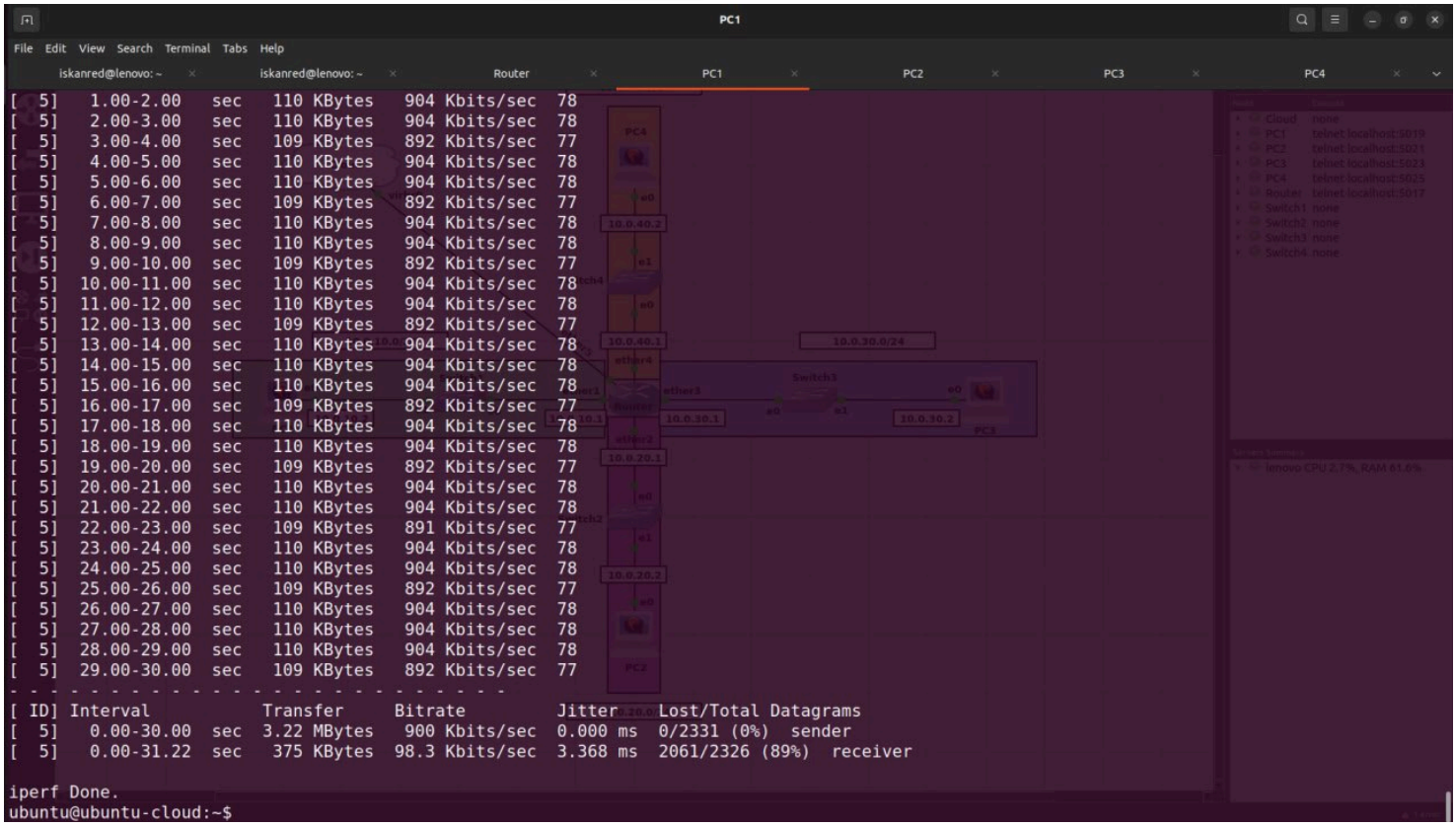


- The first queue `frp-alloc-queue` is configured to be a queue for all the packets that are marked as `ftp-alloc` => all FTP-related packets between these hosts. This queue has `limit-at=900k/900k` meaning that despite all other traffic flows in other queues, this queue will get guaranteed `900 Kbit/s` for both upload/download while the maximum limit leaves the same `1 Mbit/s`.
- The second queue `other-traffic-queue` is also configured for the traffic between these hosts. However, here `limit-at` parameter equals to `100k/100k` meaning that this queue would get only guaranteed `100 Kbit/s` for both upload/download. The traffic would go to this queue only if previous queues were not convenient. Therefore, all non-FTP traffic would go to this queue.
- It's worth to say that in this case it's unnecessary to have a parent queue. However, it's more flexible.
- Finally, we can run our FTP file download



- It's easy to notice that the speed drop in this case is really small, about `12 Kbit/s` from `118 Kbit/s` to `106 Kbit/s`

- It worked! And we can double check it in `PC1`'s `iperf3` stats



- We see that the receiver's bitrate did not exceed `100 Kbit/s`

> **6.** What is the difference between the QoS rules to traffic allocation and priority-based QoS? Try to set up each of them and show then them. In which tasks of this lab do you use one or the other?

- **Traffic allocation** is a technique to define specific policies to control how much bandwidth is assigned to different types of traffic or users within a network. The example is above for downloading files with FTP. It is an efficient way to statically give specific available bandwidth!
- **Priorities** are used classifying packets and assigning them different priorities to ensure critical applications receive better bandwidth and low-latency treatment, regardless of overall usage. Here we cannot guarantee some specific number as a bandwidth value. However, it is better for low-latency applications (such as real-time) since marked packets have a higher probability to be passed first.

Priority-based QoS configuration:

- Since I already deployed traffic allocation technique in the previous task, I decided to deploy priorities using the same queues on Mikrotik.
- First, I removed `limit-at` for the `frp-alloc-queue` to make traffic allocation QoS not having an impact on the results. Nevertheless, I gave `limit-at=10k/10k` for the `other-traffic-queue` to allow non-FTP traffic to have at least some guaranteed speed.
- Then I gave the highest priority to the `ftp-alloc-queue` (1 from 8), while the priority for the `other-traffic-queue` remained the same (8 by default).

```
[admin@MikroTik] > queue simple print
Flags: X - disabled, I - invalid; D - dynamic
 0    name="host_1-3_parent_queue" target=10.0.30.2/32 dst=10.0.10.2/32 parent=none packet-marks="" priority=8/8
      queue=pcq-upload-default/pcq-download-default limit-at=1M/1M max-limit=1M/1M burst-limit=0/0 burst-threshold=0/0 burst-time=0s/0s
      bucket-size=0.1/0.1

 1    name="ftp-alloc-queue" target=10.0.30.2/32 dst=10.0.10.2/32 parent=host_1-3_parent_queue packet-marks=ftp-alloc priority=1/1
      queue=pcq-upload-default/pcq-download-default limit-at=0/0 max-limit=1M/1M burst-limit=0/0 burst-threshold=0/0 burst-time=0s/0s
      bucket-size=0.1/0.1

 2    name="other-traffic-queue" target=10.0.30.2/32 dst=10.0.10.2/32 parent=host_1-3_parent_queue packet-marks="" priority=8/8
      queue=pcq-upload-default/pcq-download-default limit-at=10k/10k max-limit=1M/1M burst-limit=0/0 burst-threshold=0/0 burst-time=0s/0s
      bucket-size=0.1/0.1
[admin@MikroTik] >
```

- I ran started FTP file download and noticed that the speed didn't decrease at all!

```
ubuntu@ubuntu-cloud:~$ curl --local-port 42154-42155 -O ftp://ubuntu:ubuntu@10.0.10.2/my_file
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0 50.0M    0  448k    0     0   117k      0  0:07:16  0:00:03  0:07:13  117k
  1 50.0M    1  803k    0     0   117k      0  0:07:15  0:00:06  0:07:09  118k
  4 50.0M    4 2454k    0     0   117k      0  0:07:14  0:00:20  0:06:54  117k
  5 50.0M    5 2811k    0     0   117k      0  0:07:14  0:00:23  0:06:51  118k
  8 50.0M    8 4459k    0     0   117k      0  0:07:14  0:00:37  0:06:37  117k4
 11 50.0M   11 5753k    0     0   117k      0  0:07:14  0:00:48  0:06:26  118k
 36 50.0M   36 18.1M    0     0   118k      0  0:07:13  0:02:37  0:04:36  117k^C
ubuntu@ubuntu-cloud:~$
```

- Afterwards, I checked `iperf3` stats and found out that it even went wrong at the end because control socket has closed unexpectedly because of too many packets lost.

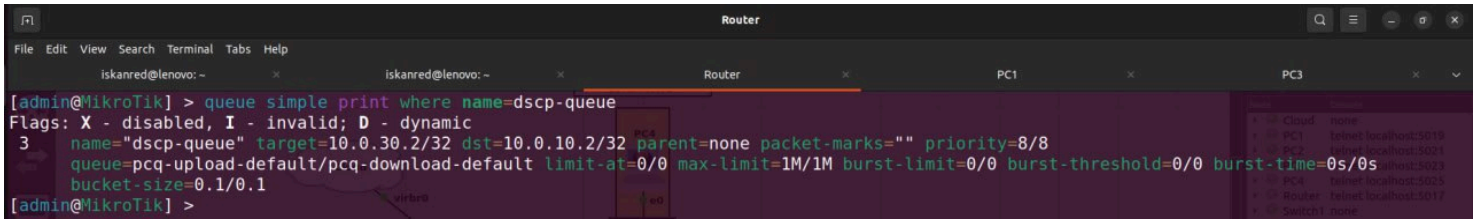
```
ubuntu@ubuntu-cloud:~$ iperf3 -c 10.0.30.2 --udp --time=30 --bandwidth=900000
Connecting to host 10.0.30.2, port 5201
[  5] local 10.0.10.2 port 33296 connected to 10.0.30.2 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec   110 KBytes   903 Kbits/sec  78
[  5]   1.00-2.00   sec   110 KBytes   904 Kbits/sec  78
[  5]   2.00-3.00   sec   110 KBytes   904 Kbits/sec  78
[  5]   3.00-4.00   sec   109 KBytes   892 Kbits/sec  77
[  5]   4.00-5.00   sec   110 KBytes   904 Kbits/sec  78
[  5]   5.00-6.00   sec   110 KBytes   904 Kbits/sec  78
[  5]   6.00-7.00   sec   109 KBytes   892 Kbits/sec  77
[  5]   7.00-8.00   sec   110 KBytes   904 Kbits/sec  78
[  5]   8.00-9.00   sec   110 KBytes   904 Kbits/sec  78
[  5]   9.00-10.00  sec   109 KBytes   892 Kbits/sec  77
[  5]  10.00-11.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  11.00-12.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  12.00-13.00  sec   109 KBytes   892 Kbits/sec  77
[  5]  13.00-14.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  14.00-15.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  15.00-16.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  16.00-17.00  sec   109 KBytes   892 Kbits/sec  77
[  5]  17.00-18.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  18.00-19.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  19.00-20.00  sec   109 KBytes   892 Kbits/sec  77
[  5]  20.00-21.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  21.00-22.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  22.00-23.00  sec   109 KBytes   892 Kbits/sec  77
[  5]  23.00-24.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  24.00-25.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  25.00-26.00  sec   109 KBytes   892 Kbits/sec  77
[  5]  26.00-27.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  27.00-28.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  28.00-29.00  sec   110 KBytes   904 Kbits/sec  78
[  5]  29.00-30.00  sec   109 KBytes   892 Kbits/sec  77
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-30.00  sec  3.22 MBytes   900 Kbits/sec  0.000 ms  0/2331 (0%)  sender
[  5]   0.00-30.00  sec  0.00 Bytes  0.00 bits/sec  0.000 ms  0/0 (0%)  receiver
iperf3: error - control socket has closed unexpectedly
ubuntu@ubuntu-cloud:~$
```

- Therefore, we can conclude that it worked perfectly! In fact, without specified `limit-at` for `other-traffic-queue` `iperf3` couldn't even connect to the `PC3`
- I checked that even with priority = `7` `ftp-traffic-queue` already takes all the traffic, so it is necessary to set the `limit-at`.

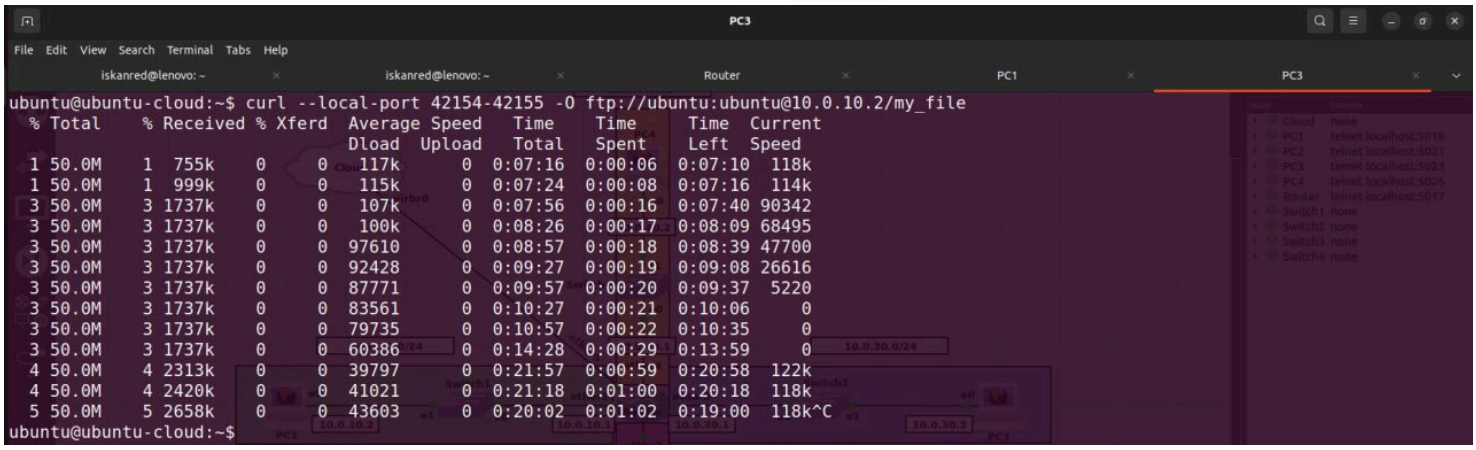
DSCP and CAKE:

- What is more, I tried to use DSCP with the help of a CAKE queueing discipline.
- Firstly, I created a new queue and I wanted to figure out how much speed I will get with for example PCQ discipline.

```
queue add name=dscp-queue target=10.0.30.2 dst=10.0.10.2 queue=pcq-upload-default/pcql-
```

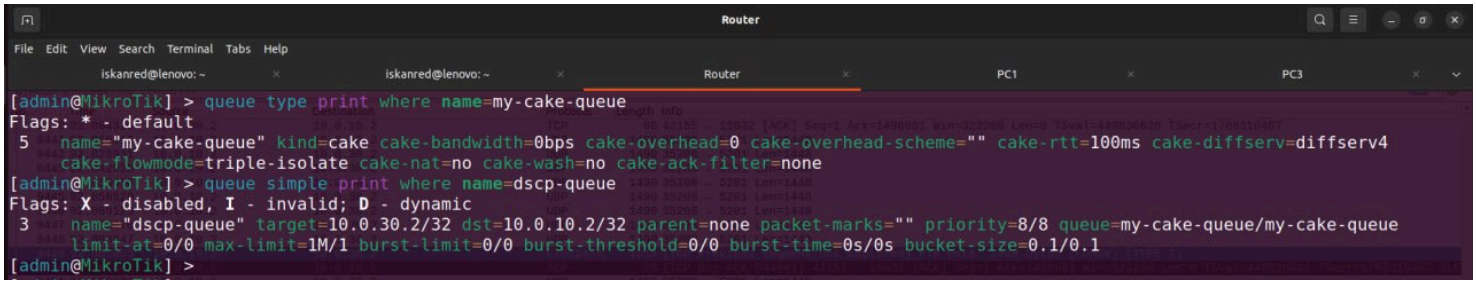


- However, it gave me 0 speed when running together with the `iperf3`



- Therefore I created a new queue type based on CAKE queueing discipline to use it with DSCP

```
queuee type add name=my-cake-queue kind cake cake-diffserv=diffserv4
queue simple set dscp-queue queue=my-cake-queue/my-cake-queue
```



- And of course I needed to configure marking FTP packets with DSCP values, so I configured them to have DSCP = 63 which is the maximum possible DSCP value

```
ip firewall mangle add chain=prerouting action=change-dscp new-dscp=63 protocol=tcp src-
address=10.0.30.2 src-port=42154
ip firewall mangle add chain=prerouting action=change-dscp new-dscp=63 protocol=tcp src-
address=10.0.30.2 src-port=42155
ip firewall mangle add chain=prerouting action=change-dscp new-dscp=63 protocol=tcp dst-
address=10.0.30.2 dst-port=42154
ip firewall mangle add chain=prerouting action=change-dscp new-dscp=63 protocol=tcp dst-
address=10.0.30.2 dst-port=42155
```

```
[admin@MikroTik] > ip firewall mangle print
Flags: X - disabled, I - invalid; D - dynamic
 0    chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp src-port=42154

 1    chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp src-port=42155

 2    chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp dst-port=42154

 3    chain=prerouting action=mark-packet new-packet-mark=ftp-alloc protocol=tcp dst-port=42155

 4    chain=prerouting action=change-dscp new-dscp=63 protocol=tcp src-address=10.0.30.2 src-port=42154

 5    chain=prerouting action=change-dscp new-dscp=63 protocol=tcp src-address=10.0.30.2 src-port=42155

 6    chain=prerouting action=change-dscp new-dscp=63 protocol=tcp dst-address=10.0.30.2 dst-port=42155

 7    chain=prerouting action=change-dscp new-dscp=63 protocol=tcp dst-address=10.0.30.2 dst-port=42154
[admin@MikroTik] >
```

- As a result the speed did reach 0, so it seems that it worked!

```
ubuntu@ubuntu-cloud:~$ curl --local-port 42154-42155 -O ftp://ubuntu:ubuntu@10.0.10.2/my_file
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0 50.0M    0  449k    0     0   117k      0  0:07:14  0:00:03  0:07:11  117k
  1 50.0M    1  564k    0     0   117k      0  0:07:14  0:00:04  0:07:10  117k
  2 50.0M    2 1111k    0     0   103k      0  0:08:17  0:00:10  0:08:07 87546
  2 50.0M    2 1165k    0     0    98k      0  0:08:38  0:00:11  0:08:27 74056
  2 50.0M    2 1224k    0     0 98099       0  0:08:54  0:00:12  0:08:42 61999
  2 50.0M    2 1283k    0     0 95266       0  0:09:10  0:00:13  0:08:57 59463
  2 50.0M    2 1404k    0     0 90553       0  0:09:38  0:00:15  0:09:23 58910
  3 50.0M    3 1633k    0     0 84285       0  0:10:22  0:00:19  0:10:03 59125
  3 50.0M    3 1802k    0     0 80734       0  0:10:49  0:00:22  0:10:27 58331
  3 50.0M    3 1916k    0     0 79110       0  0:11:02  0:00:24  0:10:38 58398
  4 50.0M    4 2150k    0     0 76415       0  0:11:26  0:00:28  0:10:58 59320
  5 50.0M    5 2888k    0     0 74307       0  0:11:45  0:00:39  0:11:06 79531
  5 50.0M    5 3009k    0     0 75512       0  0:11:34  0:00:40  0:10:54 92450
  6 50.0M    6 3126k    0     0 76567       0  0:11:24  0:00:41  0:10:43  102k
  6 50.0M    6 3245k    0     0 77628       0  0:11:15  0:00:42  0:10:33  114k
  6 50.0M    6 3362k    0     0 78618       0  0:11:06  0:00:43  0:10:23  118k
  7 50.0M    7 3601k  ^C      0 79540       0  0:10:59  0:00:44  0:10:15  118k
ubuntu@ubuntu-cloud:~$
```

> **7.** Choice and install any tool that you like for bandwidth control/netflow analysis/network control & monitoring. Play around with the network settings and show the different QoS metrics via UI

> **8.** Try to answer the question: packet drops can occur even in an unloaded network where there is no queue overflow. In what cases and why does this happen?

There are several cases where it is possible that I can figure out:

- **Firewalls** can apply special filtering for the packets and it might be not obvious for the sender what's happening.
- There can be **errors on physical layer** that can lead to packets corruption. This may look as a packet loss from both sides.
- **TTL expiration**. Packets have TTL which is measured as number of hops usually. When TTL reaches zero, a packet is dropped and appropriate ICMP message is sent to the sender.
- **RED scheduling algorithm** can be applied for queuing or a similar algorithm. It has a certain probability to drop packets even if the queue is not full. This is done to prevent the overflow.

Task 3 - QoS verification & packets analysis

> **1.** How can you check if your QoS rules are applied correctly? List and describe the various methods.

I am not sure of what is required, so I just can suggest such methods:

- **Traffic Generators and Test Tools**. Utilizing traffic generation tools (such as `iperf` or **Ostinato**) to simulate different types of traffic on your network. This can help you understand how QoS rules affect performance in a

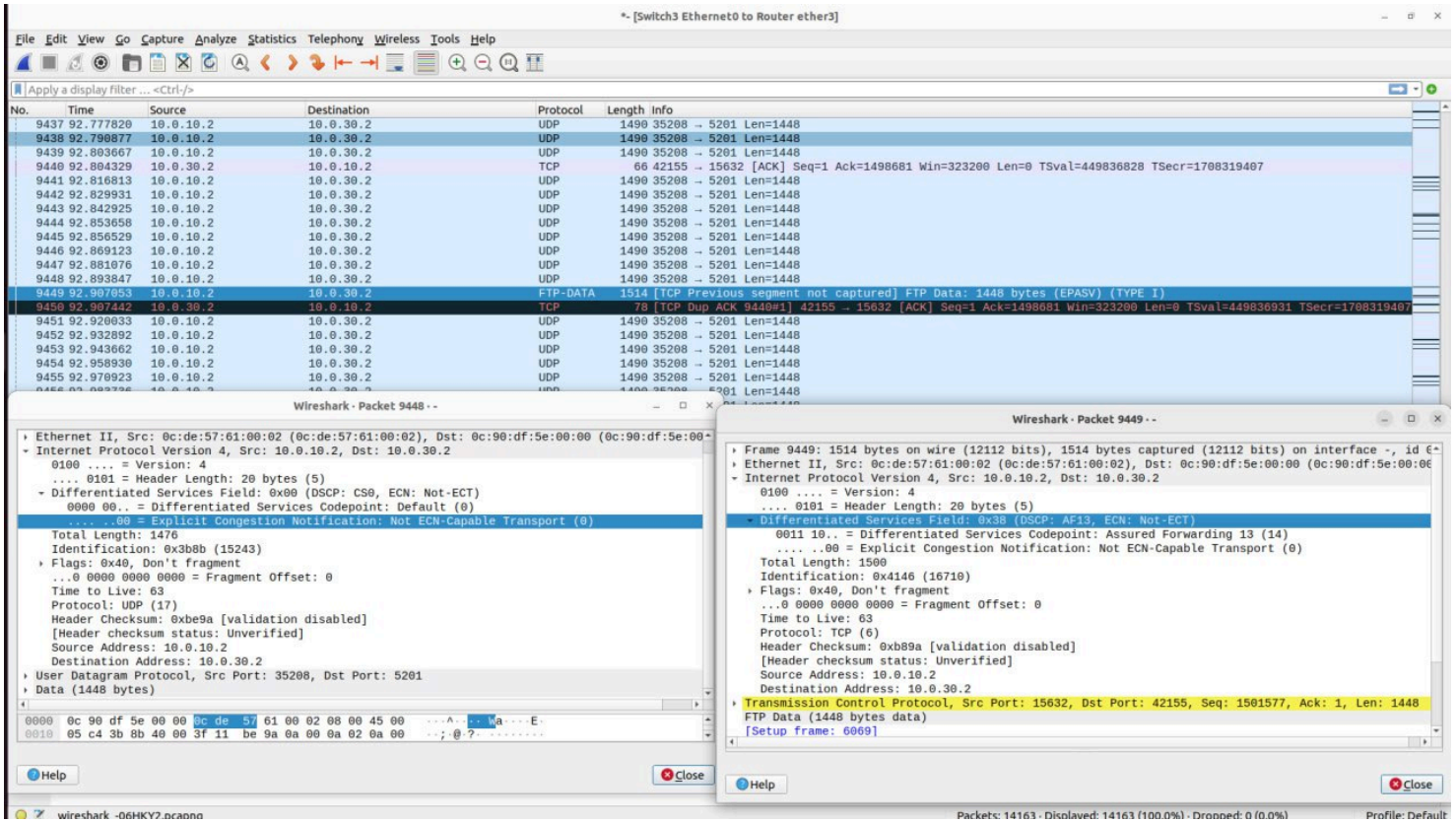
testing environment. Of course, it is better not to use it in production environment together with the production load. The method is pretty simple: generate traffic and verify its distribution with the expected values.

- **Network flow analysis**. We can monitor on how traffic is distributed in production environment by exploring it using different analytical tools, network bitrate graphs and etc, e.g. `itop`.
- **Traffic Monitoring Tools**. Use network monitoring tools to observe real-time traffic flow and check whether the QoS rules are being enforced on the network. Tools like **Wireshark** can capture and help us analysing traffic. For example, in filtering traffic based on the QoS parameters (like DSCP values) to check if the packets are marked and prioritised correctly.
- **Network Device Interfaces**. Check the configuration and status of network devices (routers, switches) where QoS is configured. For example, we can monitor if packets marking is happening using device's monitoring tools or accumulated statistics. For example, in Mirkotik it is possible to look for statistics on dropped packets, queue lengths, and traffic distribution.

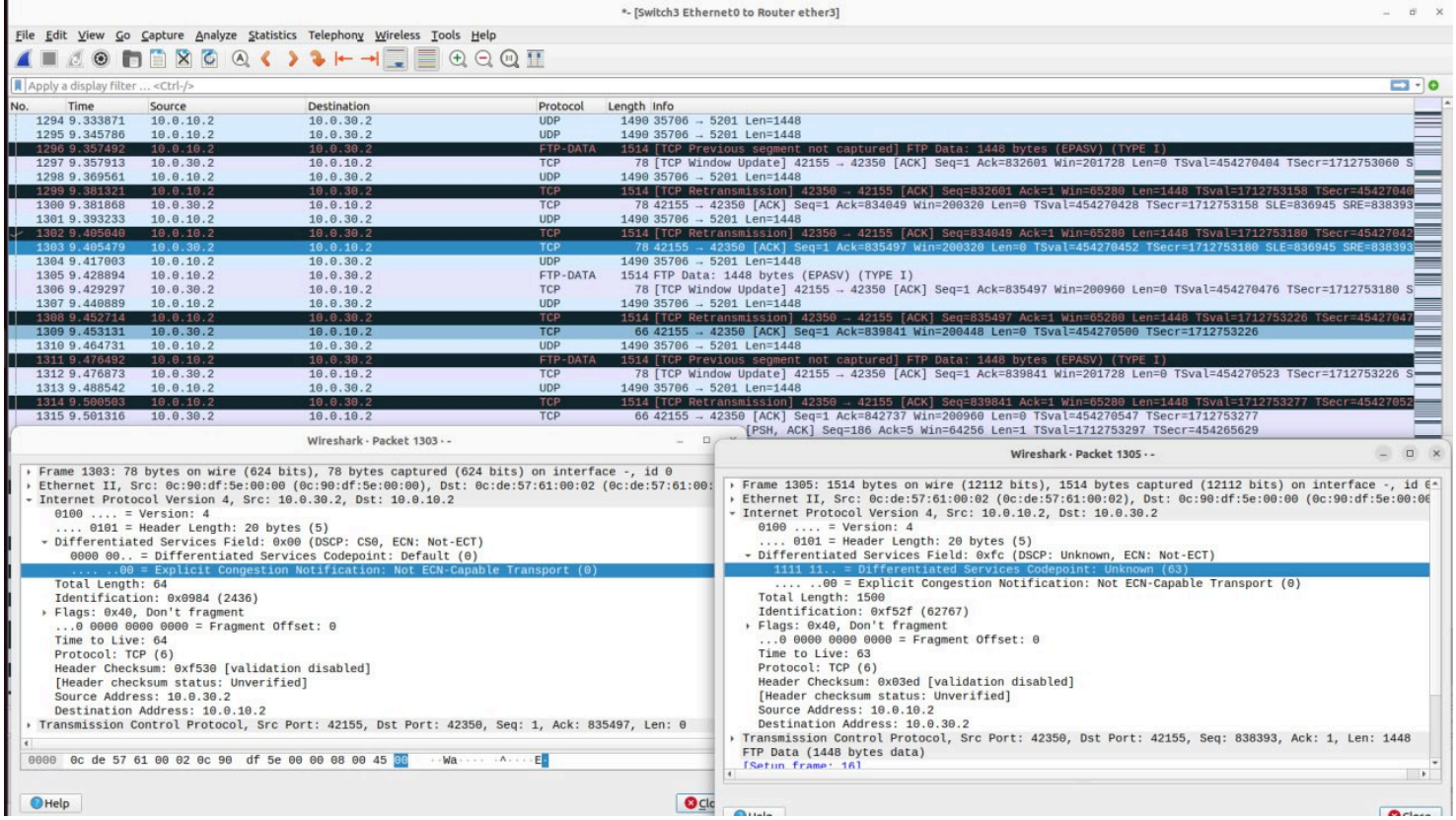
> **2.** Try to use Wireshark to see the QoS packets. How does this depend on the number of routers in the network topology?

It really depends on the method of QoS used. For example, we wouldn't find anything interesting in Wireshark if traffic allocation technique is used. However, for DSCP case we can find.

- Let me show DSCP values that were assigned to FTP traffic from the previous task.



- Here on the left we see a UDP datagram that was generated by `iperf3` and it had the default DSCP value = 0. However, on the right hand-side we can notice our FTP packet which is marked with value = 63



References

- https://help.mikrotik.com/docs/spaces/ROS/pages/18350234/Cloud+Hosted+Router+CHR
- https://help.mikrotik.com/docs/spaces/ROS/pages/7962644/Bandwidth+Test
- https://d2cpnw0u24fjm4.cloudfront.net/wp-content/uploads/iPerf3-User-Documentation.pdf