UNIVERSITY OF PISA

Advanced Network Architectures and Wireless Systems

Year 2019/20

QoS Project

GERARDO ALVARO

LEONARDO FONTANELLI

RICCARDO POLINI

Introduction

The goal of this project is to design and implement the system in Figure 1 using GNS3. Three networks are interconnected through a core network of 5 routers (R1, R2, R3, R4, R5). Each network contains either Client nodes (A1, A2, C1) or Server nodes (B1, B2). Clients send data to servers considering the flow characteristics shown in Table 1.

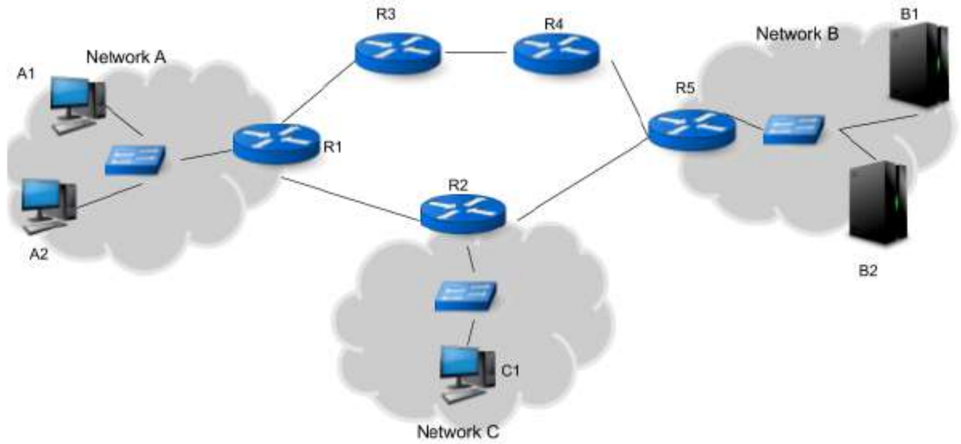


Figure 1: Network Scenario

Source	Destination	Priority	Bandwidth
A1	B1	High	0.9 Mbps
A2	B2	Low	1.2 Mbps
C1	B1	High	0.9 Mbps

Table 1: Characteristics of Data Flow

The whole network must be configured ensuring that:

- Each flow conforms to the characteristics of Table 1, for high-priority flows, the excess traffic is downgraded to low priority; for low-priority flows, the excess traffic is dropped.
- Each high-priority flow is granted at worst 60% of the bandwidth for the link it traverses in the core network.

To test the project, we just need to:

- Open GNS3
- Load **`t3.gns3`** file
- Click the *start* button
- Wait until all the interface LEDs are green

And the network will be ready for use.

Design

The network, built using GNS3, is shown in Figure 2.

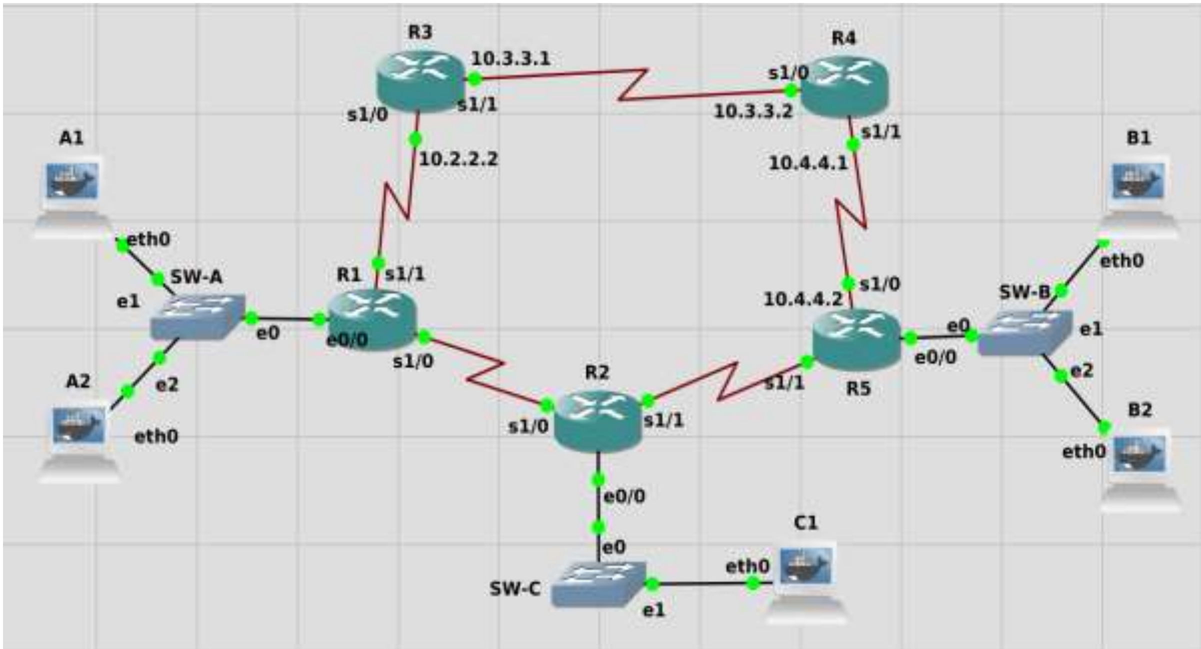


Figure 2: Network with GNS3

All the network interfaces have been configured using a static addressing plan, communications between all hosts belonging to the network take place using OSPF routing protocol.

The whole network can be seen as a unique Autonomous System with a single area with area ID = 1.

Implementation

Classification

An access-list has been defined for each data flow, in particular on router R1 have been defined the access lists for the flow from A1 to B1 and from A2 to B2, while on router R2 the access list for the flow from C1 to B1.

Data flows with high priority (A1-B1 and C1-B1) have been associated with the class-map **`HIGH`** and the one with low priority has been associated to class-map **`LOW`**.

Data flows with high priority have also been associated with another class-map used to guarantee them at worst 60% of the bandwidth for the link they traverse in the core network.

Marking

To mark traffic entering the network we decided to use the following DSCP values:

- **EF** (Expedited Forwarding): to mark high priority flows.
- **AF33** (Assured Forwarding): to mark low priority flows.

In order to improve system performances, the marking phase has been carried out only on the ingress routers R1 and R2.

To verify the correctness of the marking phase, we inspected, using Wireshark, that packets belonging to high priority flows were labelled with the DSCP value **`EF`** and the ones belonging to low priority flows were labelled with the DSCP value **`AF33`**.

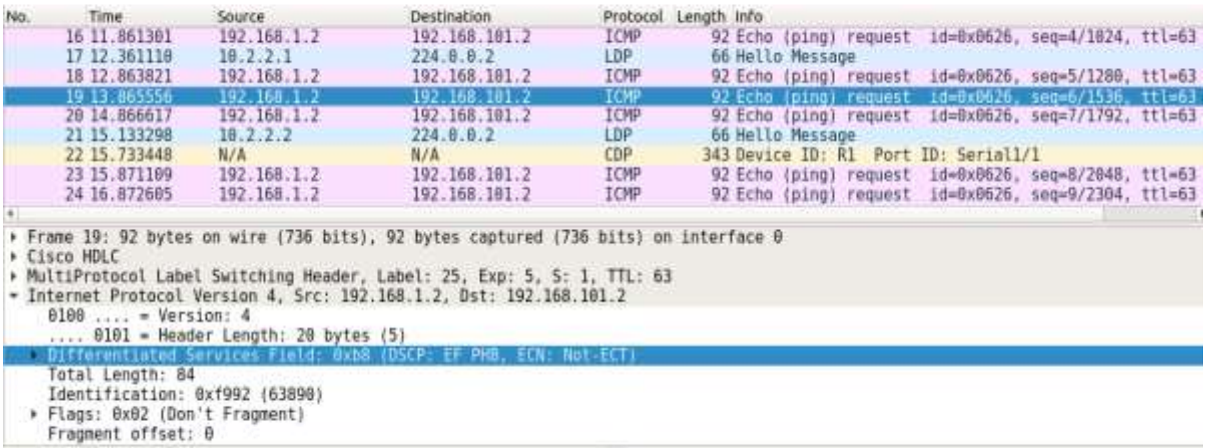


Figure 3: High priority packets marked with EF

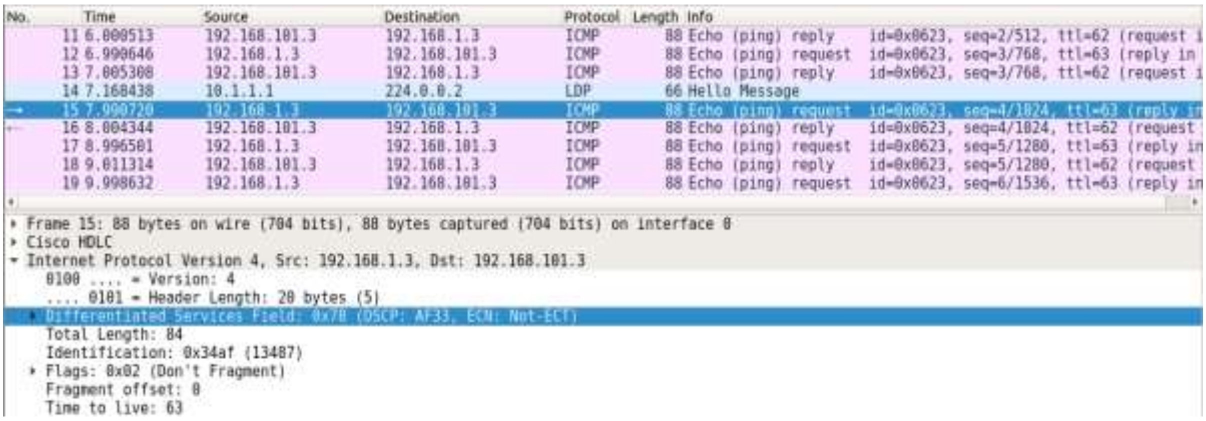


Figure 4: Low priority packets marked with AF33

Policing

To meet the excess traffic requirements, we adopted CAR into the ingress routers of the network.

The high priority traffic is limited to 0.9 Mbps and the excess packets will be downgraded to low priority by setting their DSCP value to 30 (corresponding to AF33) with the action:

```
exceed-action set-dscp-transmit
```

To test it we generated a data traffic of 1 Mbps from host A1 to server B1 by using the **`iperf`** command.

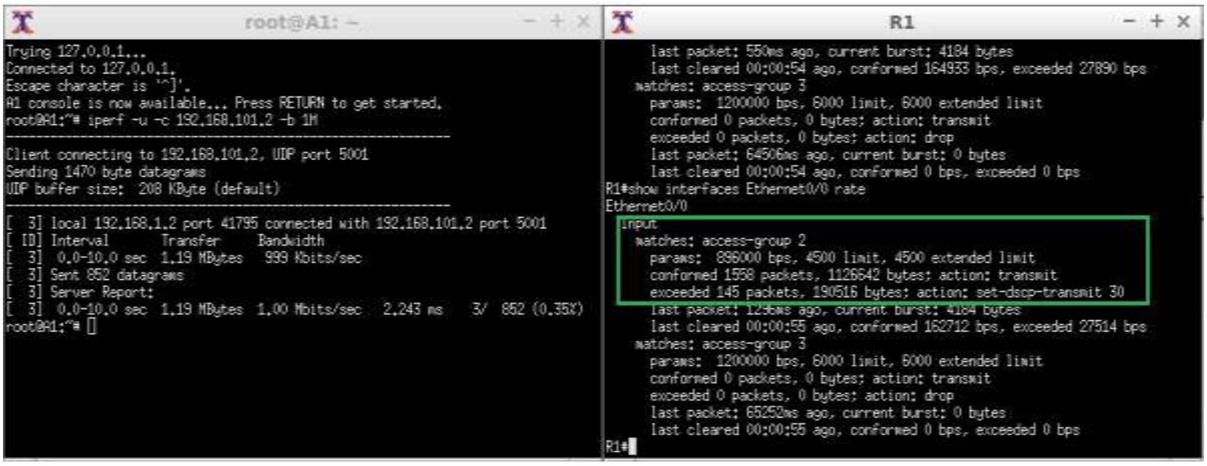


Figure 5: High priority excess traffic

The same reasoning can be done with low priority traffic which exceeds 1.2 Mbps. In this case the packets will be dropped by the ingress router R1.

To test it we generated a data traffic of 2 Mbps from host A2 to server B2 by using the **`iperf`** command.

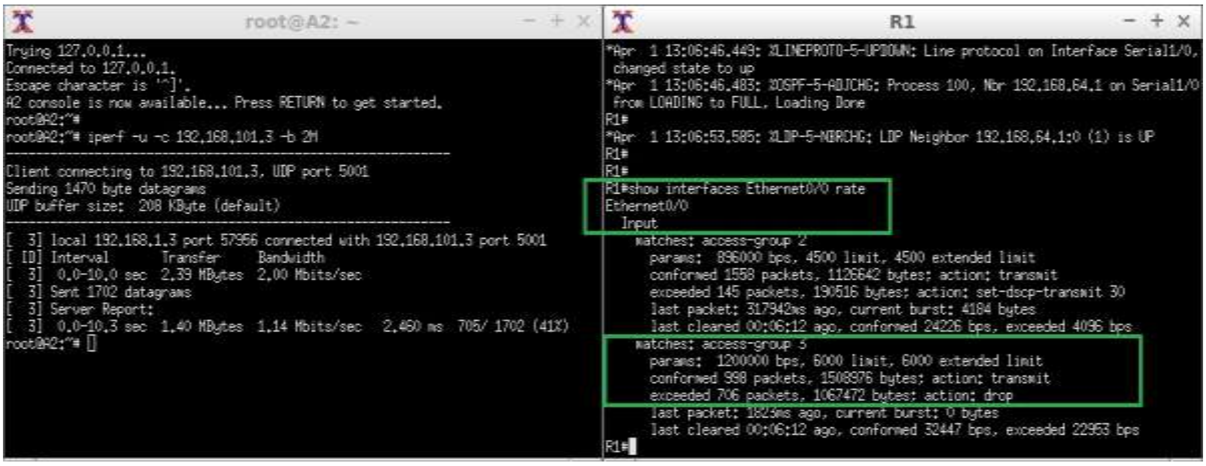


Figure 6: Low priority excess traffic

MPLS-Tunneling

We have to consider that, since we used OSPF protocol, the default path for all data traffic will be the shortest one, hence both data flows from A1 and A2 will pass through router R2.

We can notice that the following data flows will traverse the link that connects routers R2 and R5:

- From A1 to B1
- From C1 to B1
- From A2 to B2

Since the first two must have granted, at worst, 60% of the bandwidth, we need a different path in which traffic from A1 to B1 will flow.

Hence, we will build an explicit tunnel from R1 to R5, through R3 and R4, in which will flow only the traffic from A1 to B1. We can see it in Figure 7.

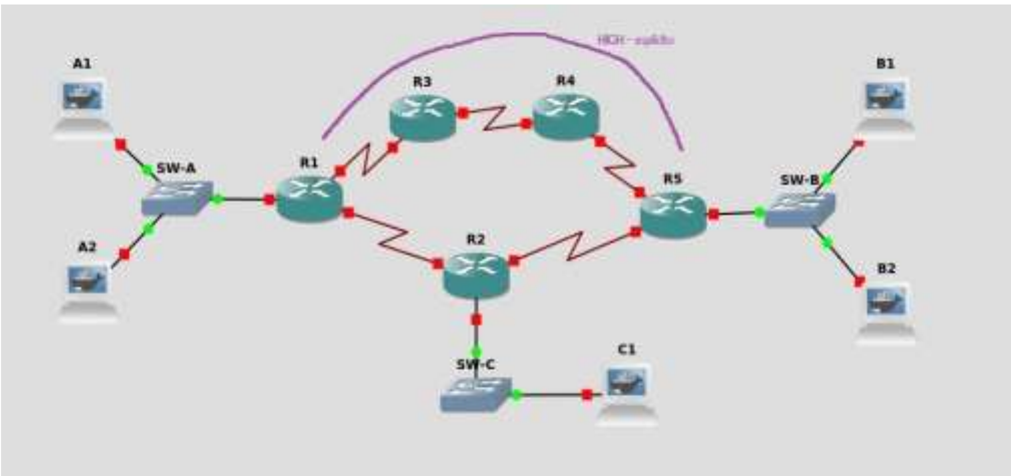


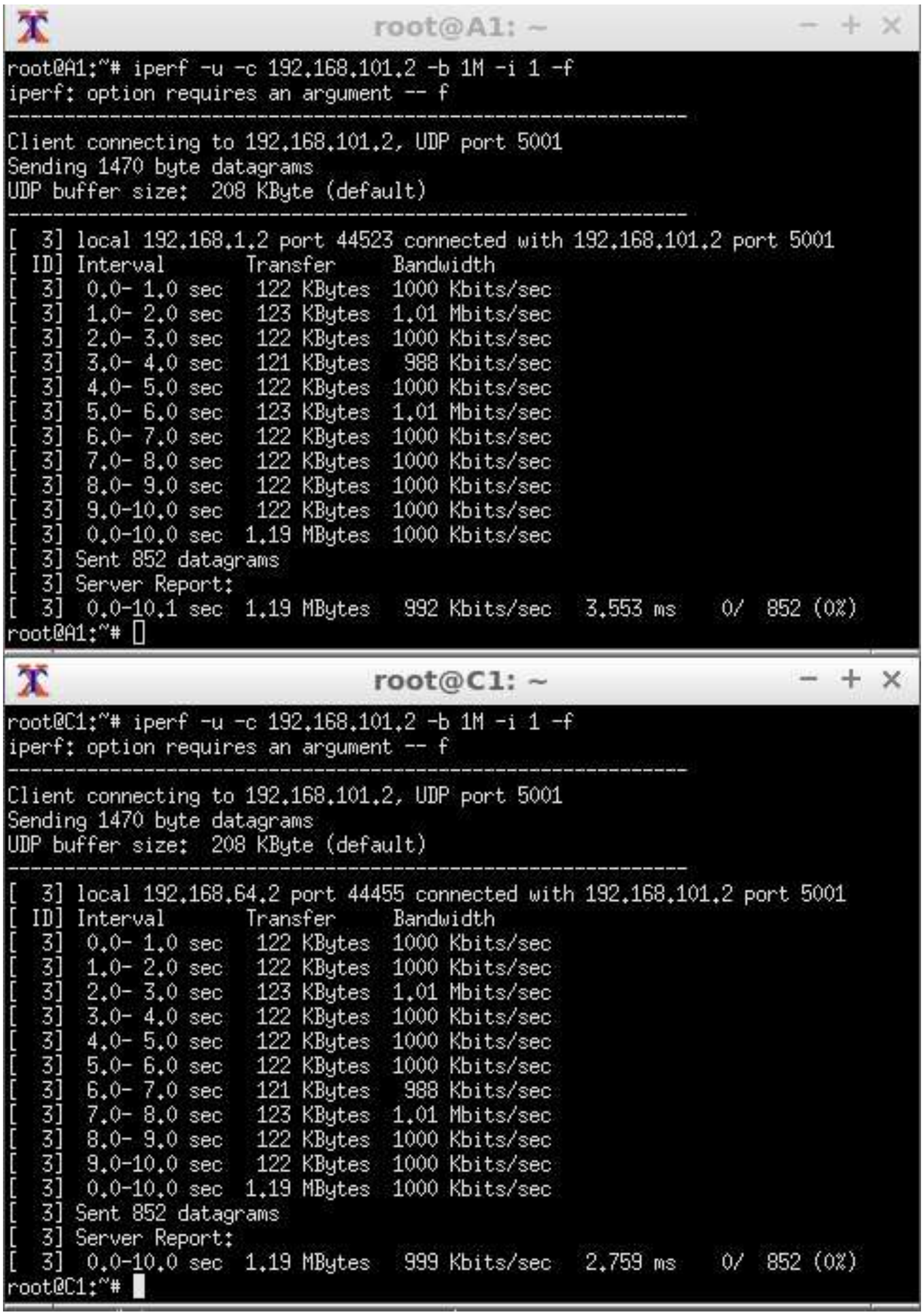
Figure 7: Explicit tunnel from R1 to R5

Testing

To test the correctness of our project we ran two tests.

<u>Test 1</u>

In the first one we generated 1 Mbps of traffic from A1 to B1 and 1 Mbps of traffic from C1 to B1.



```
root@A1: ~
root@A1:~# iperf -u -c 192.168.101.2 -b 1M -i 1 -f
iperf: option requires an argument -- f
------------------------------------------------------------
Client connecting to 192.168.101.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.2 port 44523 connected with 192.168.101.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   122 KBytes  1000 Kbits/sec
[  3]  1.0- 2.0 sec   123 KBytes  1.01 Mbits/sec
[  3]  2.0- 3.0 sec   122 KBytes  1000 Kbits/sec
[  3]  3.0- 4.0 sec   121 KBytes   988 Kbits/sec
[  3]  4.0- 5.0 sec   122 KBytes  1000 Kbits/sec
[  3]  5.0- 6.0 sec   123 KBytes  1.01 Mbits/sec
[  3]  6.0- 7.0 sec   122 KBytes  1000 Kbits/sec
[  3]  7.0- 8.0 sec   122 KBytes  1000 Kbits/sec
[  3]  8.0- 9.0 sec   122 KBytes  1000 Kbits/sec
[  3]  9.0-10.0 sec   122 KBytes  1000 Kbits/sec
[  3]  0.0-10.0 sec  1.19 MBytes  1000 Kbits/sec
[  3] Sent 852 datagrams
[  3] Server Report:
[  3]  0.0-10.1 sec  1.19 MBytes   992 Kbits/sec   3.553 ms    0/  852 (0%)
root@A1:~#
root@C1: ~
root@C1:~# iperf -u -c 192.168.101.2 -b 1M -i 1 -f
iperf: option requires an argument -- f
------------------------------------------------------------
Client connecting to 192.168.101.2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.64.2 port 44455 connected with 192.168.101.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   122 KBytes  1000 Kbits/sec
[  3]  1.0- 2.0 sec   122 KBytes  1000 Kbits/sec
[  3]  2.0- 3.0 sec   123 KBytes  1.01 Mbits/sec
[  3]  3.0- 4.0 sec   122 KBytes  1000 Kbits/sec
[  3]  4.0- 5.0 sec   122 KBytes  1000 Kbits/sec
[  3]  5.0- 6.0 sec   122 KBytes  1000 Kbits/sec
[  3]  6.0- 7.0 sec   121 KBytes   988 Kbits/sec
[  3]  7.0- 8.0 sec   123 KBytes  1.01 Mbits/sec
[  3]  8.0- 9.0 sec   122 KBytes  1000 Kbits/sec
[  3]  9.0-10.0 sec   122 KBytes  1000 Kbits/sec
[  3]  0.0-10.0 sec  1.19 MBytes  1000 Kbits/sec
[  3] Sent 852 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.19 MBytes   999 Kbits/sec   2.759 ms    0/  852 (0%)
root@C1:~#
```

Figure 8: Test 1

As expected, since data flows follow different paths in both hosts A1 and C1 we do not have packet loss.

Test 2

In this case we generated 1 Mbps of traffic from A2 to B2 and 1 Mbps of traffic from C1 to B1.

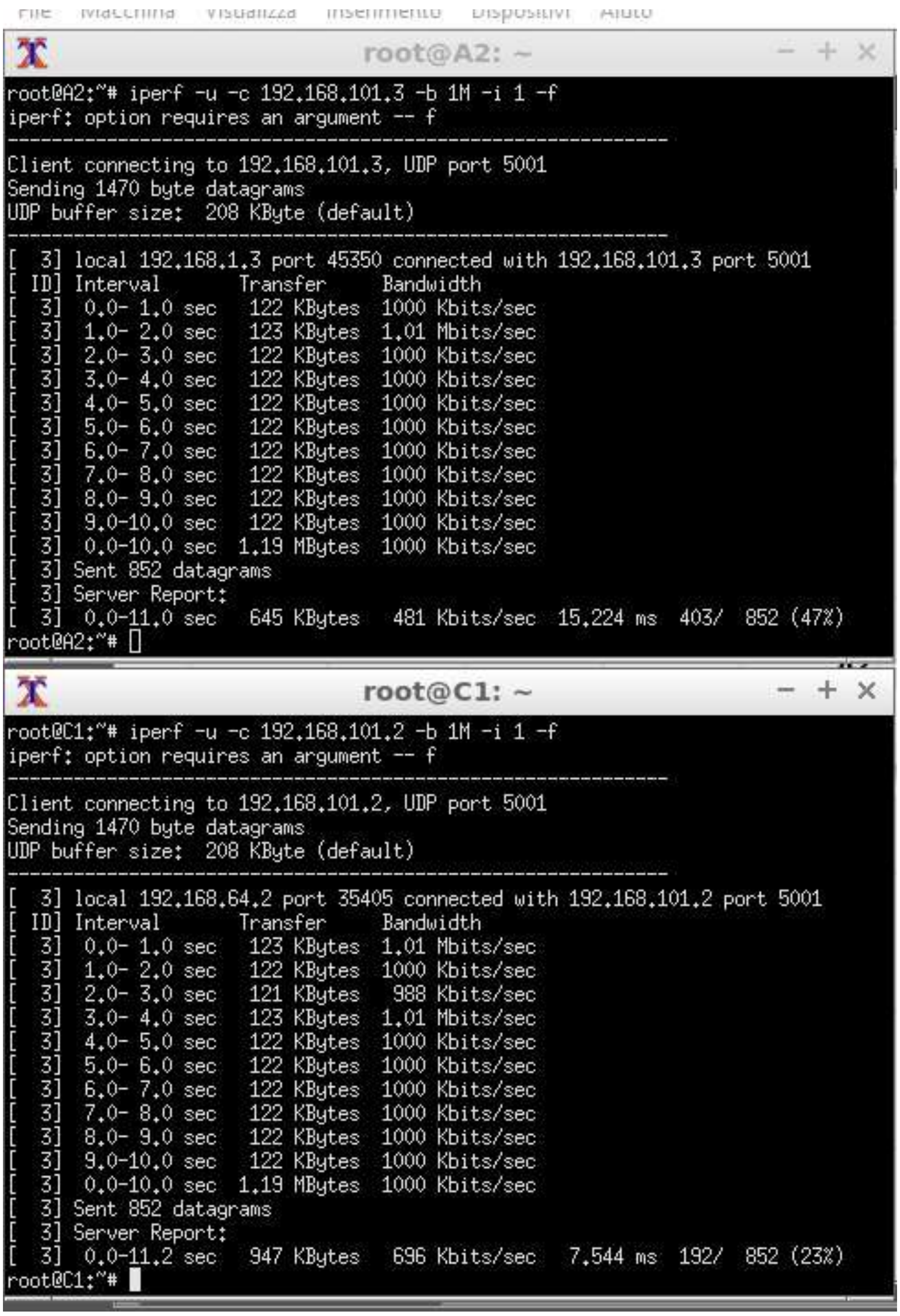


Figure 8: Test 2

As shown in Figure 8, in this case, there will be a packet drop on both hosts since they traverse simultaneously the link that connects R2 and R5; in particular, the percentage of drop packets in C1 is 23% which satisfies the requirements.