



UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 4: Emulating WAN with NETEM II: Packet Loss, Duplication, Reordering, and Corruption

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction to network emulators and NETEM	3
2 Lab topology.....	5
2.1 Testing connectivity between two hosts	6
3 Adding/changing packet loss	7
3.1 Identify interface of host h1 and host h2.....	8
3.2 Add packet loss to the interface connecting to the WAN	9
3.3 Restore default values.....	12
3.4 Add correlation value for packet loss to interface connecting to WAN	13
4 Adding packet corruption	14
4.1 Add packet corruption to an interface connected to the WAN.....	14
5 Add packet reordering	16
6 Add packet duplication	17
References	18

Overview

This lab continues the description of NETEM and how to use it to emulate Wide Area Networks (WANs). Besides delay, this lab focuses on other parameters such as packet loss, packet duplication, reordering, and packet corruption. These parameters affect the performance of protocols and networks.

Objectives

By the end of this lab, students should be able to:

1. Deploy emulated WANs characterized by parameters such as delay, packet loss, packet corruption, packet reordering, and packet duplication.
2. Measure the performance of WANs characterized by different parameter values.
3. Visualize WAN performance measures.

Lab settings

The information in Table 1 provides the credentials of the machine containing Mininet.

Table 1. Credentials to access Client1 machine.

Device	Account	Password
Client1	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to network emulators and NETEM.
2. Section 2: Lab topology.
3. Section 3: Adding/changing packet loss.
4. Section 4: Adding packet corruption.
5. Section 5: Adding packet reordering.
6. Section 6: Adding packet duplication.

1 Introduction to network emulators and NETEM

Part I of Emulating WAN with NETEM described how to use NETEM to emulate WANs characterized by long delays. Part I also explained how the end-to-end delay can be

dominated by the WAN's propagation delay and how the Round-Trip Time (RTT) estimates this delay.

In addition to delay, many WANs and LANs are subject to packet loss, reordering, corruption, and duplication.

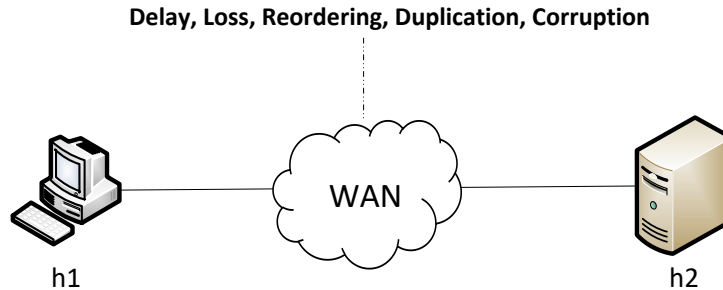


Figure 1. Parameters affecting throughput in a WAN.

The above situations are described follows:

1. Packet loss: a condition that occurs when a packet travelling across a network fails to reach its destination. Packet loss may have a large impact on high-throughput high-latency networks. A common cause of packet loss is the inability of routers to hold packets arriving at a rate higher than the departure rate. Even in cases where the high packet arrival rate is only temporary (e.g., short-term traffic bursts), the router is limited by the amount of buffer memory used to momentarily store packets. When packet loss occurs, TCP reduces the congestion window and consequently the throughput by half. Packet loss must be mitigated by using best-practice network designs, such as Science DMZ.
2. Packet reordering: a condition that occurs when packets are received in a different order from which they were sent. Packet reordering, also known as out-of-order packet delivery, is typically the result of packets following different routes to reach their destination. Packet reordering may deteriorate the throughput of TCP connections in high-throughput high-latency networks. For each segment received out of order, a TCP receiver sends an acknowledgement (ACK) for the last correctly received segment. Once the TCP sender receives three acknowledgements for the same segment (triple duplicate ACK), the sender considers that the receiver did not correctly receive the packet following the packet that is being acknowledged three times. It then proceeds to reduce the congestion window and throughput by half.
3. Packet corruption: corruption of bits comprising a packet may (mostly) occur at the physical layer. Two adjacent devices are connected by a physical channel (e.g., fiber, twisted-pair copper wire, etc). The physical layer accepts a raw bit stream and delivers it to the data-link layer. If corruption occurs, some bits may have different values than those originally sent by the sender node. The receiver node then simply discards the packet. As a result, the TCP sender process will not

receive an acknowledgement for the corresponding segment and will consider it as a lost segment. The TCP sender process will subsequently decrease the congestion window and throughput by half.

4. Packet duplication: a condition where multiple copies of a packet are present in the network and received by the destination. Packet duplication is the result of retransmissions, where a sender node retransmits unacknowledged (NACK) packets.

Packet loss, reordering, and corruption (the last two are interpreted as packet loss also by the TCP sender) lead to a drastic reduction of throughput. In this lab, we will use the NETEM tool to emulate these situations affecting end-to-end performance.

2 Lab topology

Let's get started with creating a simple Mininet topology using MiniEdit. The topology uses 10.0.0.0/8 which is the default network assigned by Mininet.

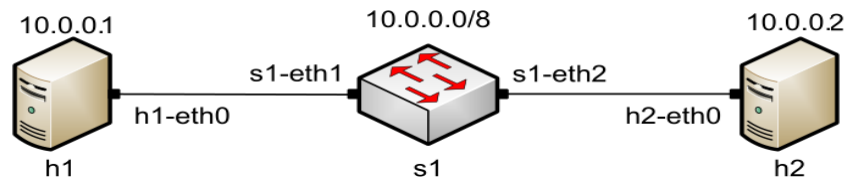


Figure 2. Lab topology.

Step 1. A shortcut to MiniEdit is located on the machine's Desktop. Start MiniEdit by clicking on MiniEdit's shortcut. When prompted for a password, type `password`.

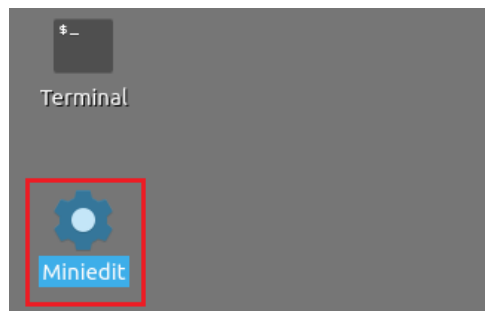


Figure 3. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 4.mn* topology file and click on *Open*.

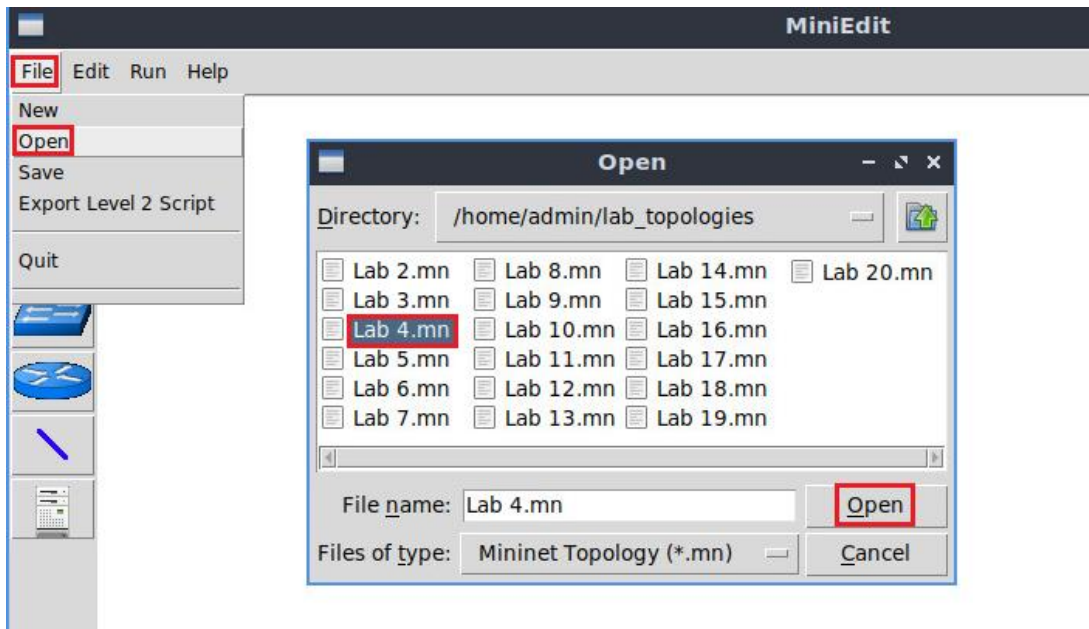


Figure 4. MiniEdit's *Open* dialog.

Step 3. Before starting the measurements between host h1 and host h2, the network must be started. Click on the *Run* button located at the bottom left of MiniEdit's window to start the emulation.

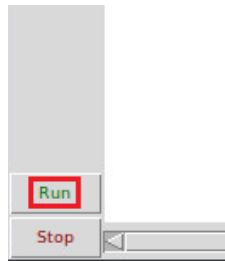


Figure 5. Running the emulation.

The above topology uses 10.0.0.0/8 which is the default network assigned by Mininet.

2.1 Testing connectivity between two hosts

Step 1. Hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on host h1.

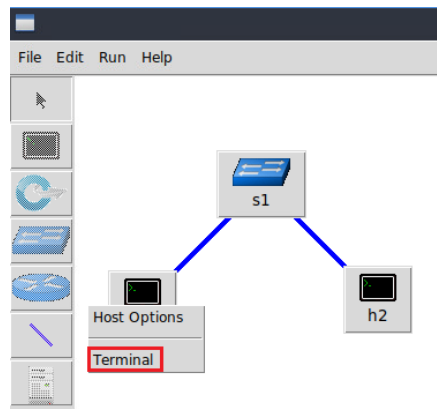


Figure 6. Opening a terminal on host h1.

Step 2. Test connectivity between the end-hosts using the `ping` command. On host h1, type the command `ping 10.0.0.2`. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.

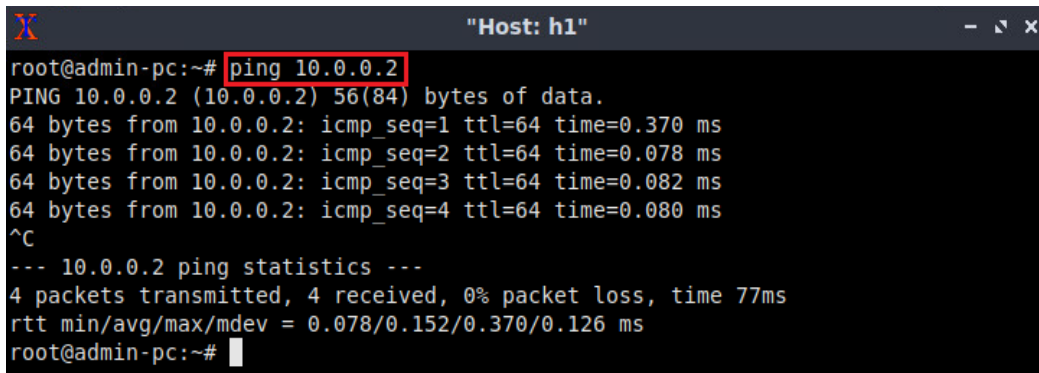


Figure 7. Connectivity test using `ping` command.

The figure above indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

3 Adding/changing packet loss

The user invokes NETEM using the command line utility called `tc`^{4,5}. With no additional parameters, NETEM behaves as a basic FIFO queue with no delay, loss, duplication, or reordering of packets. The basic `tc` syntax used with NETEM is as follows:

```
sudo tc qdisc [add|del|replace|change|show] dev dev_id root netem opts
```

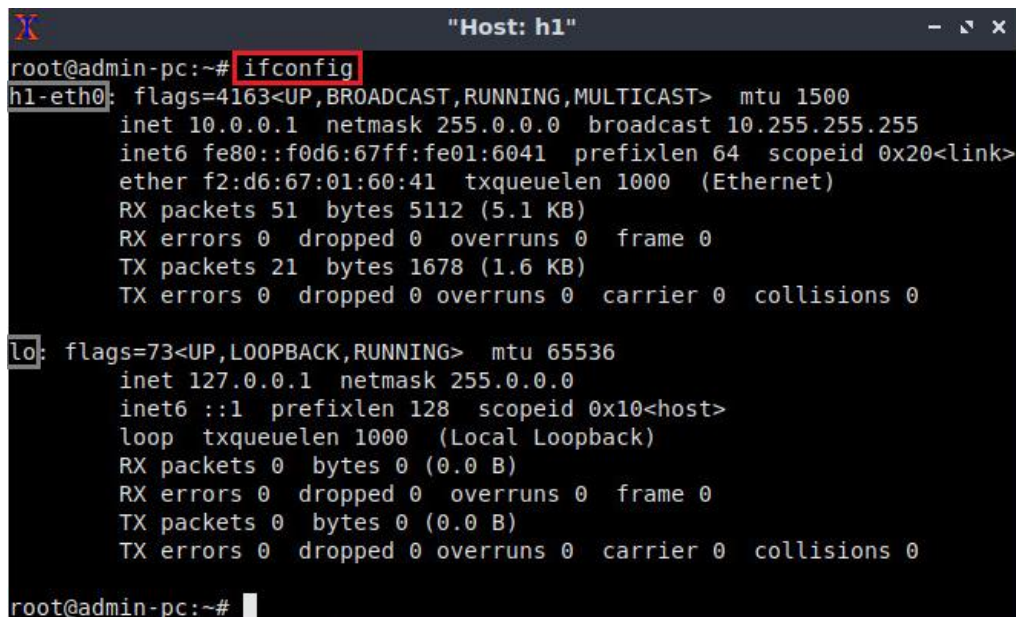
- `sudo`: enable the execution of the command with higher security privileges.
- `tc`: command used to interact with NETEM.
- `qdisc`: a queue discipline (qdisc) is a set of rules that determine the order in which packets arriving from the IP protocol output are served. The queue discipline is applied to a packet queue to decide when to send each packet.

- `[add | del | replace | change | show]`: this is the operation on `qdisc`. For example, to add delay on a specific interface, the operation will be `add`. To change or remove delay on the specific interface, the operation will be `change` or `del`.
- `dev id`: this parameter indicates the interface to be subject to emulation.
- `opts`: this parameter indicates the amount of delay, packet loss, duplication, corruption, and others.

3.1 Identify interface of host h1 and host h2

In this section, we must identify the interfaces on the connected hosts.

Step 1. On host h1, type the command `ifconfig` to display information related to its network interfaces and their assigned IP addresses.



```

root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
inet6 fe80::f0d6:67ff:fe01:6041 prefixlen 64 scopeid 0x20<link>
ether f2:d6:67:01:60:41 txqueuelen 1000 (Ethernet)
RX packets 51 bytes 5112 (5.1 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 21 bytes 1678 (1.6 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
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

root@admin-pc:~#

```

Figure 8. Output of `ifconfig` command on host h1.

The output of the `ifconfig` command indicates that host h1 has two interfaces: `h1-eth0` and `lo`. The interface `h1-eth0` at host h2 is configured with IP address 10.0.0.1 and subnet mask 255.0.0.0. This interface must be used in `tc` when emulating the WAN.

Step 2. In host h2, type the command `ifconfig` as well.


```

Host: h2
root@admin-pc:~# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
        inet6 fe80::8a:3dff:feea:b11d prefixlen 64 scopeid 0x20<link>
        ether 02:8a:3d:ea:b1:1d txqueuelen 1000 (Ethernet)
        RX packets 24 bytes 2851 (2.8 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 7 bytes 586 (586.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        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

root@admin-pc:~#

```

Figure 9. Output of `ifconfig` command on host h2.

The output of the `ifconfig` command indicates that host h2 has two interfaces: `h2-eth0` and `lo`. The interface `h2-eth0` at host h1 is configured with IP address 10.0.0.2 and subnet mask 255.0.0.0. This interface must be used in `tc` when emulating the WAN.

3.2 Add packet loss to the interface connecting to the WAN

In a network, packets may be lost during transmission due to factors such as bit errors and network congestion. The rate of packets that are lost is often measured as a percentage of lost packets with respect to the number of sent packets. In this section, you will use `netem` command to insert packet loss on a network interface.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem loss 10%
```

```

Host: h1
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem loss 10%
root@admin-pc:~#

```

Figure 10. Adding 10% packet loss to host h1's interface `h1-eth0`.

The above command adds a 10% packet loss to host h1's interface `h1-eth0`.

Step 2. The user can verify now that the connection from host h1 to host h2 has packet losses by using the `ping` command from host h1's terminal. The `-c` option specifies the total number of packets to send.

```
ping 10.0.0.2 -c 200
```

```

Host: h1
root@admin-pc:~# ping 10.0.0.2 -c 200
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.408 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=24 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.041 ms

```

Figure 11. `ping` command after introducing packet loss.

In the figure 11, host h1 sends 200 ping packets to host h2. Note the *icmp_seq* values demonstrated in the figure above.

You can see that *icmp_seq*=2, 6, 10 and 17 are missing due to packet losses. Resulting packet loss will likely vary in each emulation.

Figure 12 shows the summary report of the previous command. By default, `ping` reports the percentage of packet loss after finishing the transmission. In our test, ping reported a packet loss rate of 10%. The measured packet loss rate will tend to become closer to the configured loss rate as more trials are performed.

```

--- 10.0.0.2 ping statistics ---
200 packets transmitted, 180 received, 10% packet loss, time 1154ms
rtt min/avg/max/mdev = 0.031/0.055/0.336/0.024 ms
root@admin-pc:~#

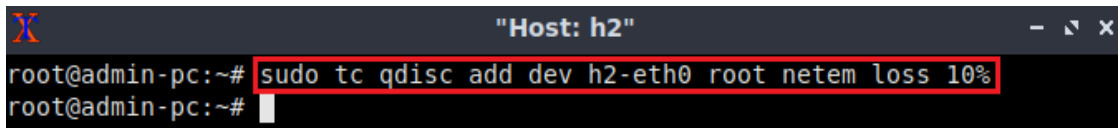
```

Figure 12. `ping` summary report showing 10% packet loss.

Note that the above scenario emulates 10% packet loss on the unidirectional link from host h1 to host h2. If we want to emulate packet loss on both directions, a packet loss of 10% must also be added to host h2.

Step 3. In host h2's terminal, type the following command:

```
sudo tc qdisc add dev h2-eth0 root netem loss 10%
```



```

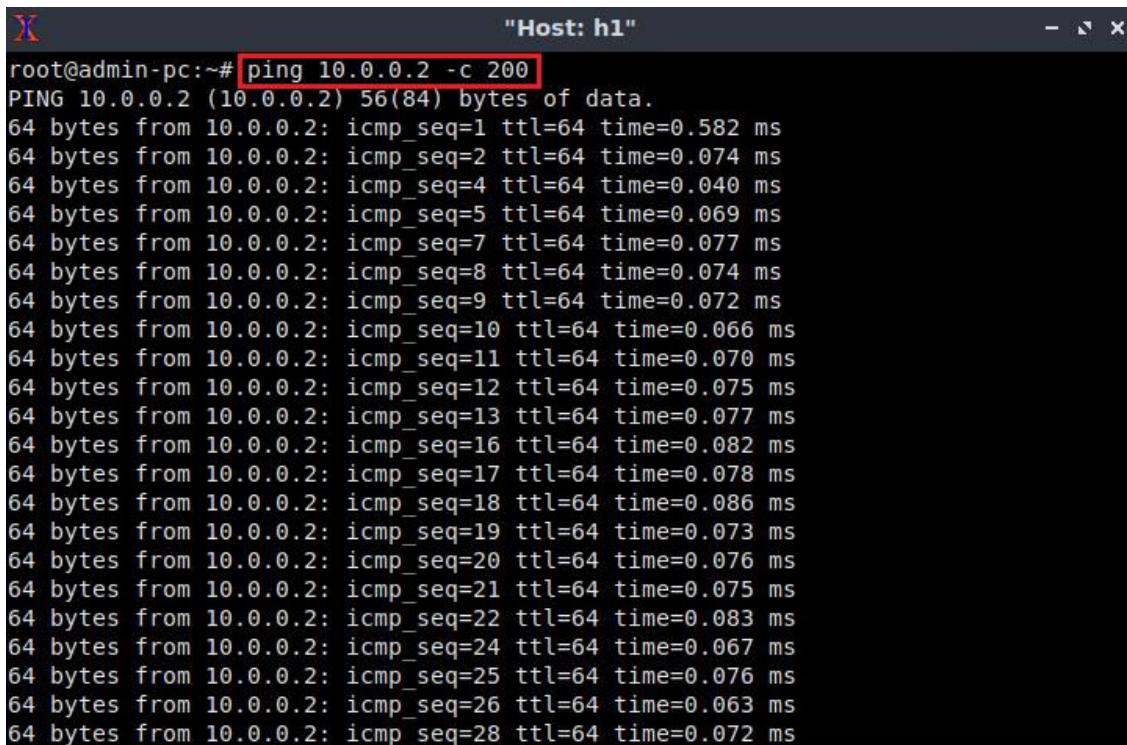
Host: h2
root@admin-pc:~# sudo tc qdisc add dev h2-eth0 root netem loss 10%
root@admin-pc:~#

```

Figure 13. Adding 10% packet loss to host h2's interface *h2-eth0*.

Step 4. The user can verify now that the connection between host h1 and host h2 has more packets losses (10% from host h1 + 10% from host h2) by retying the `ping` command on host h1's terminal:

```
ping 10.0.0.2 -c 200
```



```

Host: h1
root@admin-pc:~# ping 10.0.0.2 -c 200
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.582 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.078 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=24 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=28 ttl=64 time=0.072 ms

```

Figure 14. `ping` command after introducing packet loss.

In the figure 14, host h1 sends 200 ping packets to host h2. Note the *icmp_seq* values demonstrated in the figure above.

You can see that *icmp_seq*=3, 6, 10, 14, 23 and 27 are missing due to packet losses. Resulting packet loss will likely vary in each emulation.

Figure 14 shows the summary report of the previous command. By default, `ping` reports the percentage of packet loss after finishing the transmission. In our test, ping reported a packet loss rate of 10%. The measured packet loss rate will tend to become closer to the configured loss rate as more trials are performed.

```

--- 10.0.0.2 ping statistics ---
200 packets transmitted, 159 received, 20.5% packet loss, time 966ms
rtt min/avg/max/mdev = 0.028/0.054/0.345/0.026 ms
root@admin-pc:~#

```

Figure 15. `ping` summary report showing 20.5% packet loss.

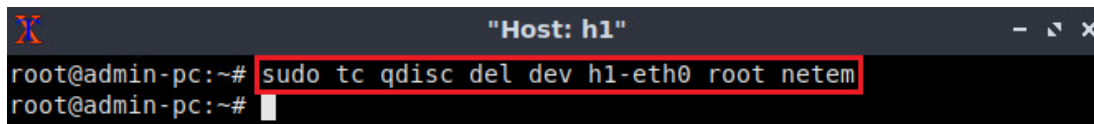
The result above indicates that 159 out of 200 packets were received successfully (20.5% packet loss).

3.3 Restore default values

To remove the packet loss added in Section 3.2 and restore the default configuration, you must delete the rules of the interfaces on host h1 and host h2.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc del dev h1-eth0 root netem
```

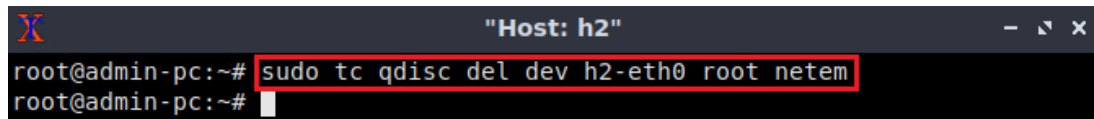


A terminal window titled "Host: h1" showing the command `sudo tc qdisc del dev h1-eth0 root netem` being entered and executed. The prompt is `root@admin-pc:~#`.

Figure 16. Deleting all rules on interface `h1-eth0`.

Step 2. Apply the same steps to remove rules on host h2. In host h2's terminal, type the following command:

```
sudo tc qdisc del dev h2-eth0 root netem
```



A terminal window titled "Host: h2" showing the command `sudo tc qdisc del dev h2-eth0 root netem` being entered and executed. The prompt is `root@admin-pc:~#`.

Figure 17. Deleting all rules on interface `h2-eth0`.

As a result, the `tc` queuing discipline will restore its default values of the device `h2-eth0`.

Step 3. Now, the user can verify that the connection from host h1 to host h2 has no explicit packet loss configured by using the `ping` command from host h1's terminal, press `Ctrl+c` to stop the test:

```
ping 10.0.0.2
```

```

Host: h1
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.357 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.045 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 107ms
rtt min/avg/max/mdev = 0.043/0.112/0.357/0.122 ms
root@admin-pc:~#
    
```

Figure 18. Verifying latency after deleting all rules on both devices.

The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 0.043, 0.112, 0.357, and 0.122 milliseconds respectively.

3.4 Add correlation value for packet loss to interface connecting to WAN

An optional correlation may be added. Adding correlation causes the random number generator to be less random and can be used to emulate packet burst losses¹.

Step 1. In host h1’s terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem loss 50% 50%
```

```

Host: h1
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem loss 50% 50%
root@admin-pc:~#
    
```

Figure 19. Verifying latency after deleting all rules on both devices.

The above command introduces a packet loss rate of 50%, and each successive probability depends 50% on the last one¹. Note that a packet loss rate this high is unlikely.

Step 2. The user can verify now that the connection from host h1 to host h2 has packet losses by using the `ping` command from host h1’s terminal.

```
ping 10.0.0.2 -c 50
```



```

Host: h1
root@admin-pc:~# ping 10.0.0.2 -c 50
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.041 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=27 ttl=64 time=0.058 ms
    
```

Figure 20. `ping` in progress showing successive packet loss.

The result above shows an example where successive packets were dropped: [3, 4, 6, 10,], [13, 14, 16, 17, 20, 21], etc.

Step 3. In host `h1`'s terminal, type the following command to delete previous configurations:

```
sudo tc qdisc del dev h1-eth0 root netem
```

```

Host: h1
root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root netem
root@admin-pc:~#
    
```

Figure 21. Deleting all rules on interface `h1-eth0`.

4 Adding packet corruption

Besides packet loss, packet corruption can be introduced with NETEM.

4.1 Add packet corruption to an interface connected to the WAN

Step 1. In host `h1`'s terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem corrupt 0.01%
```

The new value added here represents packet corruption percentage (0.01%).

```

Host: h1
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem corrupt 0.01%
root@admin-pc:~#
    
```

Figure 22. Adding packets corruption (0.01%) to interface *h1-eth0*.

Step 2. The user can now verify the previous configuration by using the `iperf3` tool to check the retransmissions. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal.

```
iperf3 -s
```

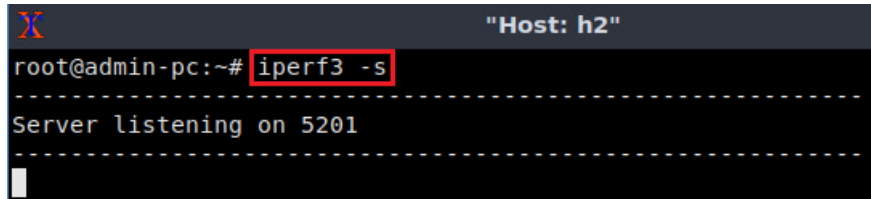


Figure 23. Host h2 running iPerf3 as server.

Step 3. To launch iPerf3 in client mode, run the command `iperf3 -c 10.0.0.2` in host h1's terminal.

```
iperf3 -c 10.0.0.2
```

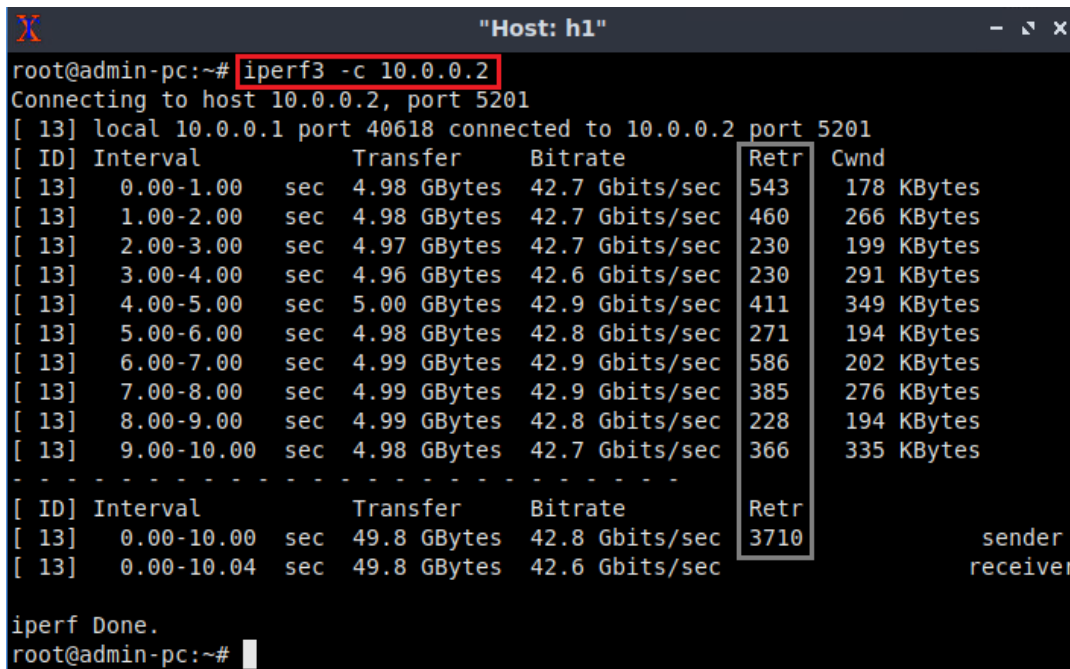
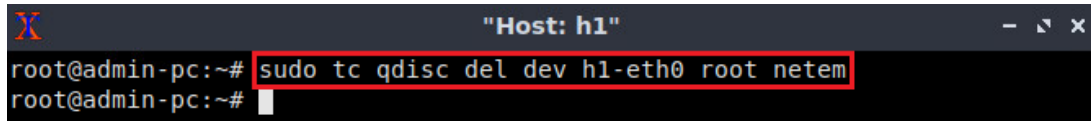


Figure 24. Retransmissions after packets corruption.

The figure above shows the retransmission values on each time interval (1 second). The total number of retransmitted packets, due to packet corruption, is 3710. This verifies that packet corruption was indeed, applied to the interface on host h1.

Step 4. In host h1's terminal, type the following command to delete previous configurations:

```
sudo tc qdisc del dev h1-eth0 root netem
```



```

root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root netem
root@admin-pc:~#

```

Figure 25. Deleting all rules on interface *h1-eth0*.

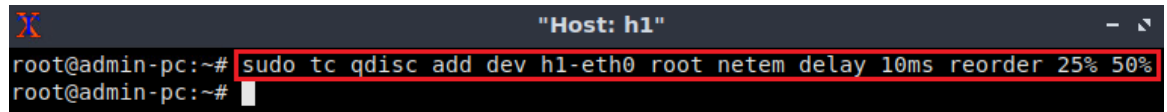
Step 5. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too. The summarized data on the server is similar to that of the client side's and must be interpreted in the same way.

5 Add packet reordering

Packets are sometimes not delivered in the same order they were sent. In order to emulate reordering in NETEM, the `reorder` option is used. Proceed with the steps below.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem delay 10ms reorder 25% 50%
```



```

root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem delay 10ms reorder 25% 50%
root@admin-pc:~#

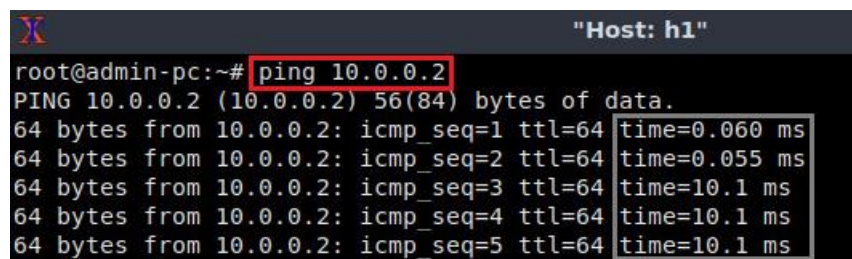
```

Figure 26. Adding packet reordering.

In this command, 25% of the packets (with a correlation value of 50%) will be sent immediately, while the remainder 75% will be delayed by 10ms.

Step 2. The user can verify the effect of packet reorder by using the `ping` command on host h1's terminal, press `Ctrl+c` to stop the test:

```
ping 10.0.0.2
```



```

root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=10.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=10.1 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=10.1 ms

```

Figure 27. `ping` test illustrating the effect of packet reordering.

Consider the first four packets of the figure above. The first and second packets did not experience delay (one out of four, or 25%), while the next three packets experienced a delay of ~10 milliseconds (three out of four, or 75%). The measured reordering rate will tend to become closer to the configured reordering rate as more trials are performed.

It is possible that your first packet will experience delay, but this effect will eventually occur in future tests.

Step 3. In host h1's terminal, type the following command to delete previous configurations:

```
sudo tc qdisc del dev h1-eth0 root netem
```

A terminal window titled "Host: h1" showing a root user at an admin-pc. The command "sudo tc qdisc del dev h1-eth0 root netem" is entered and highlighted with a red box. The prompt returns to root@admin-pc:~#.

Figure 28. Deleting all rules on interface *h1-eth0*.

6 Add packet duplication

Duplicate packets may be present in networks as a result of retransmissions. NETEM provides the option `duplicate` to inject duplicate packets. Before introducing packet corruption, make sure to restore the default configuration of the interfaces on host h1 and host h2 by applying the commands of Section 3.3. Then, proceed with the following steps.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc change dev h1-eth0 root netem duplicate 50%
```

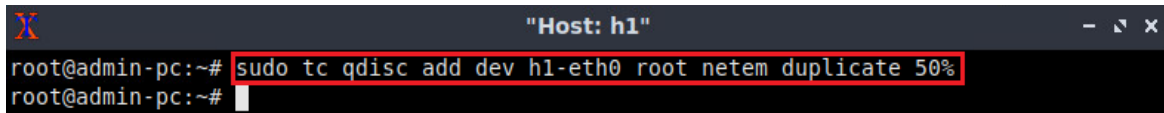
A terminal window titled "Host: h1" showing a root user at an admin-pc. The command "sudo tc qdisc add dev h1-eth0 root netem duplicate 50%" is entered and highlighted with a red box. The prompt returns to root@admin-pc:~#.

Figure 29. Adding packet duplication.

The above command will produce a duplication of 50% (i.e., 50% of the packets will be received twice at the destination).

Step 2. The user can verify the effect of packet duplication by using the `ping` command on host h1's terminal, press `Ctrl+C` to stop the test:

```
ping 10.0.0.2
```

```

root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.077 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.082 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.059 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.075 ms (DUP!)
^C
--- 10.0.0.2 ping statistics ---
9 packets transmitted, 9 received, +4 duplicates, 0% packet loss, time 197ms
rtt min/avg/max/mdev = 0.051/0.071/0.082/0.012 ms
root@admin-pc:~#

```

Figure 30. `ping` test illustrating the effect of packet duplication.

The result above indicates that five duplicate packets were received. Duplicate packets are also marked with (DUP!). The measured rate of duplicate packets will tend to become closer to the configured rate as more trials are performed.

Step 3. In host `h1`'s terminal, type the following command to delete previous configurations:

```
sudo tc qdisc del dev h1-eth0 root netem
```

```

root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root netem
root@admin-pc:~#

```

Figure 31. Deleting all rules on interface `h1-eth0`.

This concludes Lab 4. Stop the emulation and then exit out of MiniEdit.

References

1. Linux foundation. [Online]. Available: <https://wiki.linuxfoundation.org/networking/netem>.
2. S. Hemminger, "Network emulation with NETEM," Linux conf au. 2005, pp. 18-23. 2005.
3. How to use the linux traffic control panagiotis vouzis [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control>.
4. M. Brown, F. Bolelli, N. Patriciello, "Traffic control howto," Guide to IP Layer Network, 2006.