FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

RCOM - 2nd Lab Work

Computer Networks

3LEIC01

(December 23th, 2024)

Filipe Gaio, 202204985
Henrique Fernandes, 202204988

1 - Introduction

This report outlines the development and testing of a download application as part of the RCOM course, providing practical experience with network configuration. The project had two main objectives. The first was to design and implement an application capable of downloading files from specified URLs using the RFC 959 FTP protocol. The second objective was to configure and deploy a network in the NetLab room, giving us hands-on experience with hardware and device configuration.

2 - Development of a Download Application

2.1. Architecture

The download application is developed in C, and it implements the FTP protocol (RFC959). It is able to download a single file from a given URL in the form of ***ftp://[user[:password]@]host[:port]/path***.

It does so by following these steps:

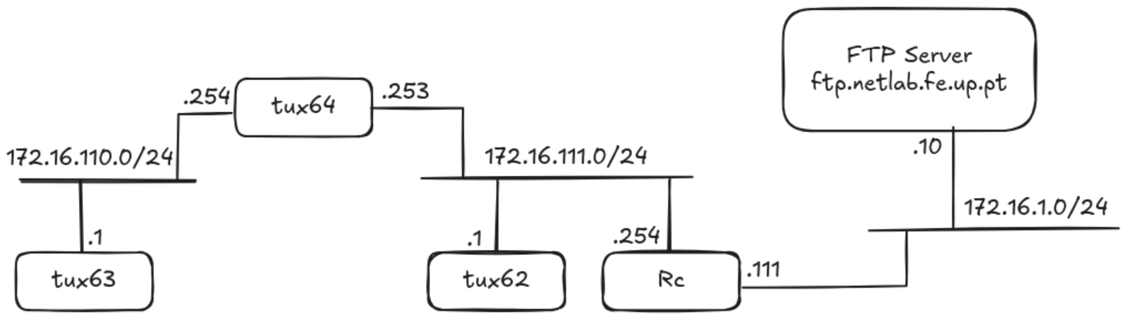
1. Parse the URL to extract the host, port, path, username and password.
2. Make a DNS query to resolve the hostname to an IP address.
3. Establish a TCP connection to the FTP server using sockets.
4. Authenticate with the server using the username and password.
5. Activate passive mode.
6. Open a new socket for data transfer.
7. Send the RETR command to the server via the first connection to request the file.
8. Receive the file data from the server via the second connection.
9. Write the file data to a local file.
10. Close the connections.

The output of the program can be found here.

3 - Network configuration and analysis

A note on the desk we used: although it was labeled as Desk 11, the computers' names are tux6x, so we are going to use 11 for the network configuration but 6 for the computer names.

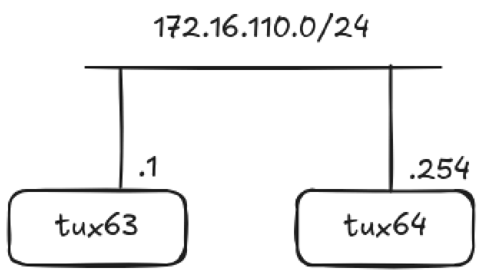
The goal is to configure a network like this:



3.1 Configure an IP Network

3.1.1 Objective

The goal for this task is to connect tux63 and tux64 to the switch, in order to be able to ping each other.



3.1.2 Physical Connections

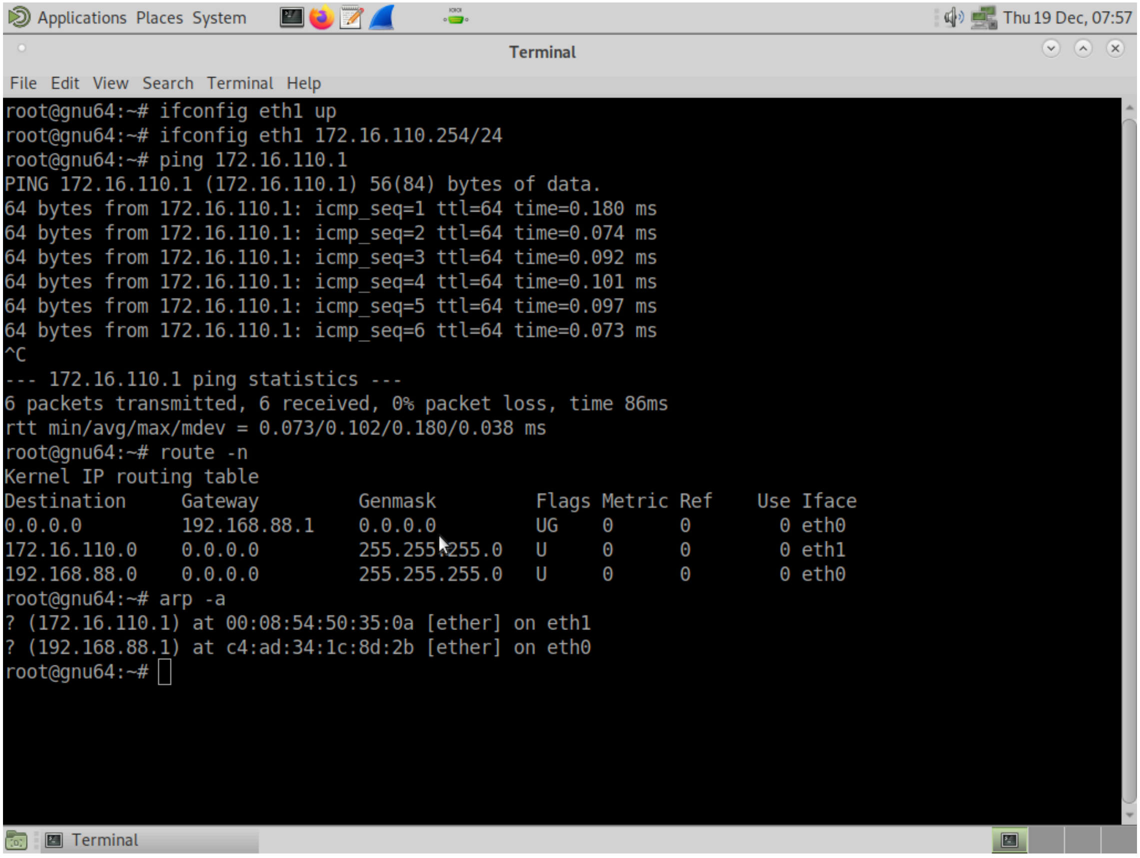
Port eth1 of tux63 connected to port eth3 of the switch and port eth1 of tux64 to port eth4 of the switch.

3.1.3 Configuration and Conclusions

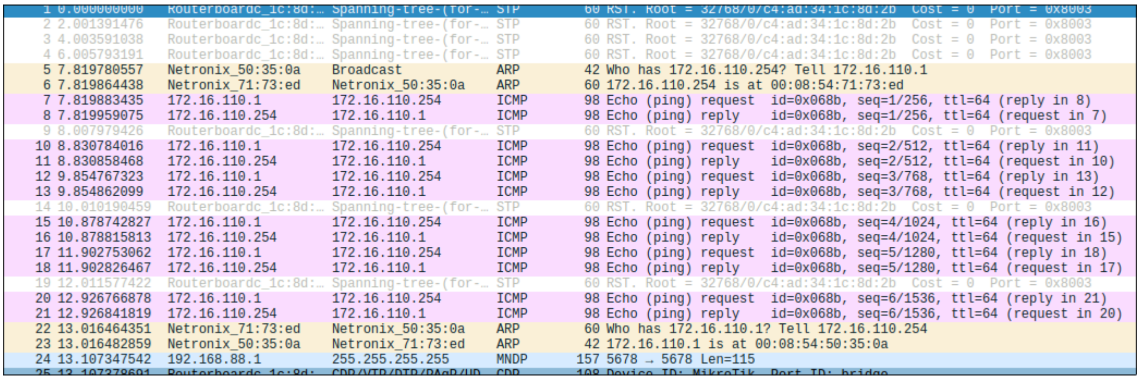
```
Unset
ifconfig eth1 up                   // tux63
ifconfig eth1 172.16.110.1/24      // tux63
ifconfig eth1 up                   // tux64
ifconfig eth1 172.16.110.254/24    // tux64
```

	tux63	tux64
IP	172.16.110.1	1172.16.110.254

MAC	00:08:54:50:35:0a	00:08:54:71:73:ed



After deleting the arp entry with *arp -d 172.16.110.254*, we can start capturing packets on port eth1 of tux63 and pinging tux64 (ping 172.16.110.254) we get the following results:



1	0.000000000	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2b Cost = 0 Port = 0x8003
2	2.001391476	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2b Cost = 0 Port = 0x8003
3	4.003591038	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2b Cost = 0 Port = 0x8003
4	6.005793191	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2b Cost = 0 Port = 0x8003
5	7.819780557	Netronix_50:35:0a	Broadcast	ARP	42	Who has 172.16.110.254? Tell 172.16.110.1
6	7.819864438	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	172.16.110.254 is at 00:08:54:71:73:ed
7	7.819883435	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x068b, seq=1/256, ttl=64 (reply in 8)
8	7.819959075	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x068b, seq=1/256, ttl=64 (request in 7)
9	8.007979426	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2b Cost = 0 Port = 0x8003
10	8.830784016	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x068b, seq=2/512, ttl=64 (reply in 11)
11	8.830858468	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x068b, seq=2/512, ttl=64 (request in 10)
12	9.854767323	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x068b, seq=3/768, ttl=64 (reply in 13)
13	9.854862099	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x068b, seq=3/768, ttl=64 (request in 12)
14	10.010190459	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2b Cost = 0 Port = 0x8003
15	10.878742827	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x068b, seq=4/1024, ttl=64 (reply in 16)
16	10.878815813	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x068b, seq=4/1024, ttl=64 (request in 15)
17	11.902753062	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x068b, seq=5/1280, ttl=64 (reply in 18)
18	11.902826467	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x068b, seq=5/1280, ttl=64 (request in 17)
19	12.011577422	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2b Cost = 0 Port = 0x8003
20	12.926766878	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x068b, seq=6/1536, ttl=64 (reply in 21)
21	12.926841819	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x068b, seq=6/1536, ttl=64 (request in 20)
22	13.016464351	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	Who has 172.16.110.1? Tell 172.16.110.254
23	13.016482859	Netronix_50:35:0a	Netronix_71:73:ed	ARP	42	172.16.110.1 is at 00:08:54:50:35:0a
24	13.107347542	192.168.88.1	255.255.255.255	MNDP	157	5678 → 5678 Len=115

It starts by sending a broadcast using the ARP protocol to find the MAC address of the target (172.16.110.254), which is 00:08:54:71:73:ed. The first ARP packet comes from tux63 and its destination is the broadcast address. Then, the response comes from the device with the IP tux63 asked. The MAC address is needed for devices to communicate in a local network.

Then, the regular ping packets proceed (ICMP). Each ICMP packet is one of 2 types: request or reply, and is usually 98 bytes in length.

3.1.4 Questions

» What are the ARP packets and what are they used for?

R: The ARP (Address Resolution Protocol) is used to map an IP address to a MAC address. An ARP Request is a broadcast packet asking, “Who has this IP address?” and the ARP Reply is a unicast response containing the MAC address associated with the requested IP.

» What are the MAC and IP addresses of ARP packets and why?

R: In an ARP request, the sender IP and MAC correspond to the requesting device, the target IP is the IP of the target, and the MAC address is left as 00:00:00:00:00:00 because it is not known yet. In an ARP reply, the sender IP and MAC are the identifiers of the original target, and the target IP and MAC are the identifiers of the original sender.

» What packets does the ping command generate?

R: The ping command uses ICMP (Internet Control Message Protocol) to test network connectivity and ARP if the IP address is not matched to a MAC address in the cache already.

» What are the MAC and IP addresses of the ping packets?

R: They correspond to the sender and the destination of each packet, and are used to identify each device in the network

» How to determine if a receiving Ethernet frame is ARP, IP, ICMP?

R: Look at the *EtherType* field in the Ethernet header:

```
▾ Ethernet II, Src: Netronix_50:35:0a (00:08:54:50:35:0a), Dst: Netronix_71:73:ed (00:08:54:71:73:ed)
    ▸ Destination: Netronix_71:73:ed (00:08:54:71:73:ed)
    ▸ Source: Netronix_50:35:0a (00:08:54:50:35:0a)
      Type: IPv4 (0x0800)
      [Stream index: 4]
```

```
▾ Ethernet II, Src: Netronix_71:73:ed (00:08:54:71:73:ed), Dst: Netronix_50:35:0a (00:08:54:50:35:0a)
    ▸ Destination: Netronix_50:35:0a (00:08:54:50:35:0a)
    ▸ Source: Netronix_71:73:ed (00:08:54:71:73:ed)
      Type: ARP (0x0806)
      [Stream index: 4]
      Padding: 000000000000000000000000000000000000
```

If it is 0x0800, then it is an IP packet, if it is 0x0806 is is an ARP packet.

» How to determine the length of a receiving frame?

R: The frame length can be found in the frame header or by inspecting the overall packet size in Wireshark.

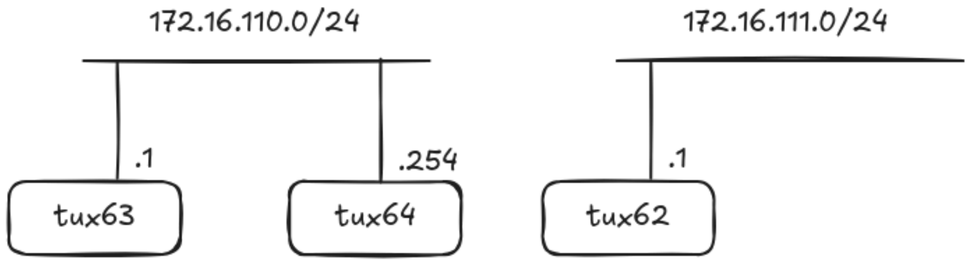
» What is the loopback interface and why is it important?

R: The loopback interface is a virtual network interface that points back to the device itself, and is usually associated with the IP 127.0.0.1 (also known as *localhost*), and it is used for testing internal communication within the device.

3.2 Implement two bridges in a switch

3.2.1 Objective

The goal for this task is to add another device (tux62) and have it on a separate bridge, such that tux63 and tux64 remain connected, but isolated from tux62.



3.2.2 Physical Connection

Port eth1 of tux62 connected to port eth2 of the switch.

3.2.3 Configuration and Conclusions

```
Unset
    ifconfig eth1 up                  // tux62
    ifconfig eth1 172.16.111.1/24     // tux62
```

	tux62	tux63	tux64
IP	172.16.111.1	172.16.110.1	172.16.110.254
MAC	00:c0:df:08:d5:98	00:08:54:50:35:0a	00:08:54:71:73:ed

Using the serial port connection, the following commands were used on the Mikrotik Switch to create and configure the bridges.

```
Unset
    interface bridge add name=bridge110
```

```
interface bridge add name=bridge111
interface bridge port remove [find interface=ether2]
interface bridge port remove [find interface=ether3]
interface bridge port remove [find interface=ether4]
interface bridge port add bridge=bridge110 interface=ether3
interface bridge port add bridge=bridge110 interface=ether4
interface bridge port add bridge=bridge111 interface=ether2
```

*(1st question)

By capturing port eth1 of tux63 and pinging tux64 and tux62 we get the following results:

13	22.977001528	Netronix_50:35:0a	Broadcast	ARP	42	Who has 172.16.110.254? Tell 172.16.110.1
14	22.977174877	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	172.16.110.254 is at 00:08:54:71:73:ed
15	22.977184166	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0b8f, seq=1/256, ttl=64 (reply in 16)
16	22.977291863	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0b8f, seq=1/256, ttl=64 (request in 15)
17	24.001312818	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0b8f, seq=2/512, ttl=64 (reply in 18)
18	24.001445588	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0b8f, seq=2/512, ttl=64 (request in 17)
19	24.018549691	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
20	25.025311075	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0b8f, seq=3/768, ttl=64 (reply in 21)
21	25.025415001	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0b8f, seq=3/768, ttl=64 (request in 20)
22	26.020934454	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
23	26.049343838	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0b8f, seq=4/1024, ttl=64 (reply in 24)
24	26.049449580	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0b8f, seq=4/1024, ttl=64 (request in 23)
25	27.073311859	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0b8f, seq=5/1280, ttl=64 (reply in 26)
26	27.073414528	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0b8f, seq=5/1280, ttl=64 (request in 25)
27	28.007300619	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	Who has 172.16.110.1? Tell 172.16.110.254
28	28.007320035	Netronix_50:35:0a	Netronix_71:73:ed	ARP	42	172.16.110.1 is at 00:08:54:50:35:0a
29	28.023294853	Routerboardc_1c:8d:	Spanning-tree-(for-	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001

Like in the previous task, tux64 is reachable, but, since it is in a separate bridge, tux62 is not accessible.

There are 2 broadcast domains, one for each bridge. Thus, tux63 and tux64 share the same broadcast address, while tux62 has a different one. Wireshark captures can be found on the annexes. *(2nd question)

As expected, when pinging the broadcast address of tux63, tux64 also detects the packets, but tux62 doesn't. When pinging the broadcast address of tux62, neither tux63 nor tux64 detects anything.

3.2.4 Questions

All of the questions were answered throughout the report and tagged with (*1st question) concerning the first question, for example.

The questions are the following:

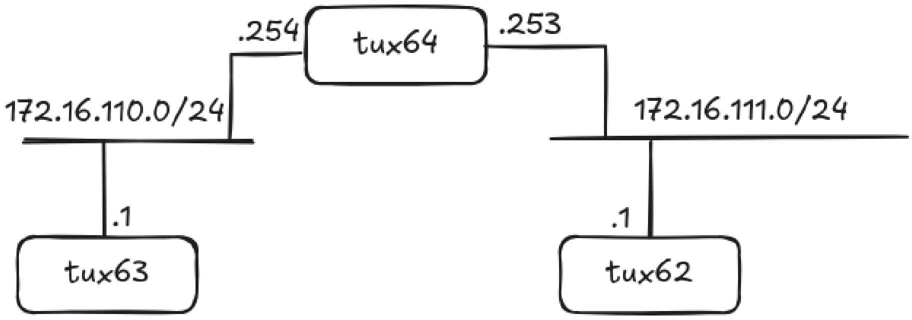
1st - » How to configure bridgeY0?

2nd - » How many broadcast domains are there? How can you conclude it from the logs?

3.3 Configure a Router in Linux

3.3.1 Objective

The goal of this task is to turn tux64 into a router so that tux63 and tux62 can access each other.



3.3.2 Physical Connection

Connect eth2 of tux64 to eth14 of the switch

3.3.3 Configuration and Conclusions

Unset
```
        // in tux64:
        ifconfig eth2 up
        ifconfig eth2 172.16.111.253/24
        sysctl net.ipv4.ip_forward=1                    //    enable    ip
forwarding
        sysctl net.ipv4.icmp_echo_ignore_broadcasts=0  //   disable   ICMP
echo-ignore-broadcast

        route add -net 172.16.110.0/24 gw 172.16.111.253     // tux62
        route add -net 172.16.111.0/24 gw 172.16.110.254     // tux63
        interface bridge port remove [find interface=ether14] // switch
        interface bridge port add bridge=bridge111 interface=ether14
```

	tux62	tux63	tux64	tux64 (eth2)
IP	172.16.111.1	172.16.110.1	172.16.110.254	172.16.111.253
MAC	00:c0:df:08:d5:98	00:08:54:50:35:0a	00:08:54:71:73:ed	00:e0:7d:b4:d1:cd

Now, tux63 can communicate with tux62 and vice-versa, using tux64 as a middle-man.

For that, tux63 needs to know how to reach the subnetwork 172.16.111.0/24, and tus62 needs to know how to reach the subnetwork 172.16.110.0/24. Since tux64 is connected to both networks, it is used to route the packets.

The routes of each computer can be found here.

6	8.617117517	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0976, seq=1/256, ttl=64 (reply in 7)
7	8.617256991	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0976, seq=1/256, ttl=64 (request in 6)
8	9.647464642	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0976, seq=2/512, ttl=64 (reply in 9)
9	9.647570312	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0976, seq=2/512, ttl=64 (request in 8)
10	10.000487864	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
11	10.671460152	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0976, seq=3/768, ttl=64 (reply in 12)
12	10.671568896	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0976, seq=3/768, ttl=64 (request in 11)
13	11.695459923	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x0976, seq=4/1024, ttl=64 (reply in 14)
14	11.695561124	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0976, seq=4/1024, ttl=64 (request in 13)
15	12.002608388	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
16	13.679429495	Netronix_50:35:0a	Netronix_71:73:ed	ARP	42	Who has 172.16.110.254? Tell 172.16.110.1
17	13.679536772	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	172.16.110.254 is at 00:08:54:71:73:ed
18	13.750989029	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	Who has 172.16.110.1? Tell 172.16.110.254
19	13.751002579	Netronix_50:35:0a	Netronix_71:73:ed	ARP	42	172.16.110.1 is at 00:08:54:50:35:0a
20	14.004706842	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
21	16.006834210	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
22	17.656908947	172.16.110.1	172.16.111.253	ICMP	98	Echo (ping) request id=0x097d, seq=1/256, ttl=64 (reply in 23)
23	17.657030960	172.16.111.253	172.16.110.1	ICMP	98	Echo (ping) reply id=0x097d, seq=1/256, ttl=64 (request in 22)
24	18.008927566	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
25	18.671464093	172.16.110.1	172.16.111.253	ICMP	98	Echo (ping) request id=0x097d, seq=2/512, ttl=64 (reply in 26)
26	18.671596513	172.16.111.253	172.16.110.1	ICMP	98	Echo (ping) reply id=0x097d, seq=2/512, ttl=64 (request in 25)
27	19.695462397	172.16.110.1	172.16.111.253	ICMP	98	Echo (ping) request id=0x097d, seq=3/768, ttl=64 (reply in 28)
28	19.695567509	172.16.111.253	172.16.110.1	ICMP	98	Echo (ping) reply id=0x097d, seq=3/768, ttl=64 (request in 27)
29	20.011086363	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
30	20.719459444	172.16.110.1	172.16.111.253	ICMP	98	Echo (ping) request id=0x097d, seq=4/1024, ttl=64 (reply in 31)
31	20.719564416	172.16.111.253	172.16.110.1	ICMP	98	Echo (ping) reply id=0x097d, seq=4/1024, ttl=64 (request in 30)
32	22.013199204	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
33	24.015303455	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
34	24.824934790	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0981, seq=1/256, ttl=64 (reply in 35)
35	24.825223935	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0981, seq=1/256, ttl=63 (request in 34)
36	25.839443911	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0981, seq=2/512, ttl=64 (reply in 37)
37	25.839689963	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0981, seq=2/512, ttl=63 (request in 36)
38	26.017354486	Routerboardc_1c:8d:2d	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
39	26.863460304	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0981, seq=3/768, ttl=64 (reply in 40)
40	26.863689943	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0981, seq=3/768, ttl=63 (request in 39)

After cleaning the ARP tables in the 3 tuxes, we can ping tux62 from tux63 and capture the packets in both tux64 ports:

eth1:

19	34.596922417	Netronix_50:35:0a	Broadcast	ARP	60	Who has 172.16.110.254? Tell 172.16.110.1
20	34.596951820	Netronix_71:73:ed	Netronix_50:35:0a	ARP	42	172.16.110.254 is at 00:08:54:71:73:ed
21	34.597051623	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0a25, seq=1/256, ttl=64 (reply in 22)
22	34.597453629	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0a25, seq=1/256, ttl=63 (request in 21)
23	35.610313296	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0a25, seq=2/512, ttl=64 (reply in 24)
24	35.610493696	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0a25, seq=2/512, ttl=63 (request in 23)
25	36.017602588	Routerboardc_1c:8d:2e	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
26	36.634317535	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0a25, seq=3/768, ttl=64 (reply in 27)
27	36.634496818	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0a25, seq=3/768, ttl=63 (request in 26)
28	37.658314092	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0a25, seq=4/1024, ttl=64 (reply in 29)
29	37.658494702	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x0a25, seq=4/1024, ttl=63 (request in 28)
30	38.009655488	Routerboardc_1c:8d:2e	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
31	39.776905640	Netronix_71:73:ed	Netronix_50:35:0a	ARP	42	Who has 172.16.110.1? Tell 172.16.110.254
32	39.777011938	Netronix_50:35:0a	Netronix_71:73:ed	ARP	60	172.16.110.1 is at 00:08:54:50:35:0a
33	40.011714968	Routerboardc_1c:8d:2e	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
34	42.013805178	Routerboardc_1c:8d:2e	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
35	44.015920252	Routerboardc_1c:8d:2e	Spanning-tree-(for-bridge…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
36	45.215540189	0.0.0.0	255.255.255.255	MNDP	160	5678 → 5678 Len=118
37	45.215570570	Routerboardc_1c:8d:2d	CDP/VTP/DTP/PAgP/UDLD	CDP	94	Device ID: MikroTik Port ID: bridge110
38	45.215618970	Routerboardc_1c:8d:2d	LLDP_Multicast	LLDP	111	MA/c4:ad:34:1c:8d:2d IN/bridge110 120 SysN=MikroTik SysD=MikroTik RouterOS

eth2:

23.200971227	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0a25, seq=1/256, ttl=63 (no response found!)
23.254640692	ASUSTekCOMPU_b3:e9:e8	Broadcast	ARP	60	Who has 192.168.109.1? Tell 192.168.109.113
23.591738936	Cisco_b6:8c:05	Spanning-tree-(for-bridge…	STP	60	Conf. Root = 32768/0/4c:00:82:2e:9a:00 Cost = 19 Port = 0x8005
23.823001584	Cisco_b6:8c:05	Cisco_b6:8c:05	LOOP	60	Reply
24.000392927	HewlettPacka_61:2f:24	Broadcast	ARP	60	Who has 10.227.20.3? Tell 10.227.20.14
24.214110188	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0a25, seq=2/512, ttl=63 (no response found!)
24.255986324	ASUSTekCOMPU_b3:e9:e8	Broadcast	ARP	60	Who has 192.168.109.1? Tell 192.168.109.113
25.024415815	HewlettPacka_61:2f:24	Broadcast	ARP	60	Who has 10.227.20.3? Tell 10.227.20.14
25.105216592	ASUSTekCOMPU_b3:e9:e8	Broadcast	ARP	60	Who has 192.168.109.116? Tell 192.168.109.113
25.105243691	ASUSTekCOMPU_b3:e9:e8	Broadcast	ARP	60	Who has 192.168.109.115? Tell 192.168.109.113
25.105246065	ASUSTekCOMPU_b3:e9:e8	Broadcast	ARP	60	Who has 192.168.109.114? Tell 192.168.109.113
25.105248719	ASUSTekCOMPU_b3:e9:e8	Broadcast	ARP	60	Who has 192.168.109.112? Tell 192.168.109.113
25.105250884	ASUSTekCOMPU_b3:e9:e8	Broadcast	ARP	60	Who has 192.168.109.111? Tell 192.168.109.113
25.238114847	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x0a25, seq=3/768, ttl=63 (no response found!)

As we can see, the requests pass through tux64.

3.3.4 Questions

» What routes are there in the tuxes? What are their meaning?

R: The routes define how packets move through the network. By typing route -n in a tux we can see the table of routes established. We can see two very important columns: Destination and Gateway, which mean, respectively, that if we want to get to a certain destination (or network) (IP) we need to go through (hop) the gateway (IP).

» What information does an entry of the forwarding table contain?

R:

- **Destination**: The target network or host.
- **Gateway**: The next hop (if 0.0.0.0, it's a direct route).
- **Genmask**: Subnet mask for the route.
- **Flags**: Status and behavior of the route (U, G, etc.).
- **Metric**: Priority of the route (smaller is preferred).
- **Iface**: The interface through which packets are sent.

» What ARP messages, and associated MAC addresses, are observed and why?

R:

- **ARP Messages and associated MAC**:
  - Tux63 is broadcasting the message of who owns the IP 172.16.111.0, it it is getting the MAC address as the response.
- **Why?**
  These ARP messages and associated MACs are observed since these tuxes never exchanged information or established connections since the beginning of the network configuration.

ARP traffic is observed whenever:

- A device communicates with another device for the first time.
- The ARP cache has expired.

» What ICMP packets are observed and why?

R:

- **Observed ICMP Packets**:
  - **Echo Reply (Type 0)**: The response to a ping request.
  - **Destination Unreachable (Type 3)**: Indicates that a packet could not reach its destination.
  - **Echo Request (Type 8):** Sent as part of a ping to test connectivity.
  - **Time Exceeded (Type 11)**: Indicates a packet's TTL expired.
- **Why?**
  - ICMP packets like Echo Requests and Replies are part of network diagnostics.
  - Error messages like "Destination Unreachable" and "Time Exceeded" are triggered by routing issues or misconfigurations.

» What are the IP and MAC addresses associated to ICMP packets and why?

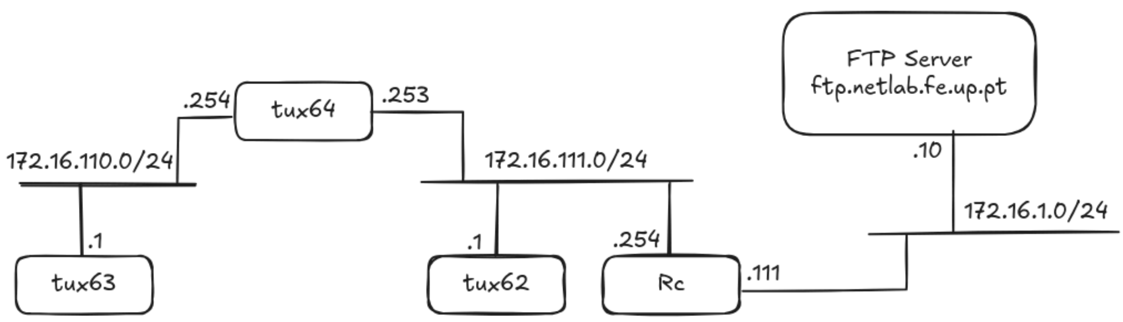
R:

- **IP Addresses**:
    - **Source IP**: The sender of the ICMP packet (e.g., the Tux sending a ping).
    - **Destination IP**: The target of the ICMP packet (e.g., the Tux being pinged).
- **MAC Addresses**:
    - **Source MAC**: The MAC address of the sending Tux's network interface.
    - **Destination MAC**: The MAC address of the receiving Tux's network interface (or a router, if crossing subnets).
- **Why?**
    - The IP addresses ensure end-to-end communication between devices.
    - The MAC addresses are used for local delivery within the same subnet. When packets are routed, the MAC addresses change at each hop, but the IP addresses remain constant.

3.4 Configure a Commercial Router and Implement NAT

3.4.1 Objective

The goal of this task is to connect the current network to the lab network, in order to access the FTP server.



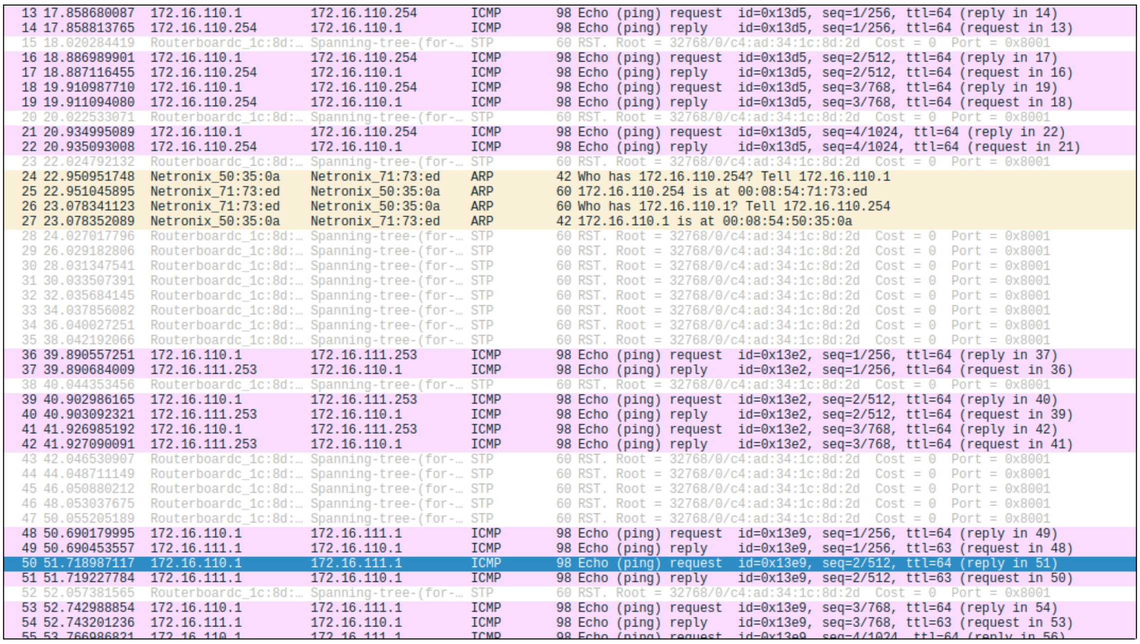
3.4.2 Physical Connection

Connect ether1 of the router to the lab network on port P6.12, and ether2 of the router to port ether10 of the switch

3.4.3 Configuration and Conclusions

```
Unset
        // in the switch, add the interface to brdge111
        interface bridge port remove [find interface=ether10]
        interface bridge port add bridge=bridge111 interface=ether10

        // in the router, assign the IP addresses and the route
        ip address add address=172.16.1.111/24 interface=ether1 // assign
ip to the lab network
        ip  address  add  address=172.16.111.254/24  interface=ether2  //
assign ip to the bridge111
        ip  route  add  dst-address=172.16.110.0/24  gateway=172.16.111.253
// route to tux63's network

        // Add the routes to the lab network
        route  add  -net  172.16.1.0/24  gw  172.16.110.254  //tux63,  route  to
lab network
        route  add  -net  172.16.1.0/24  gw  172.16.111.254  //tux64,  route  to
lab network
        route  add  -net  172.16.1.0/24  gw  172.16.111.254  //  tux62,  route  to
lab network
```

Now, when capturing the packets on tux63.eth1, it can reach every device in the network, including the FTP server. *(1st question - 1/2)



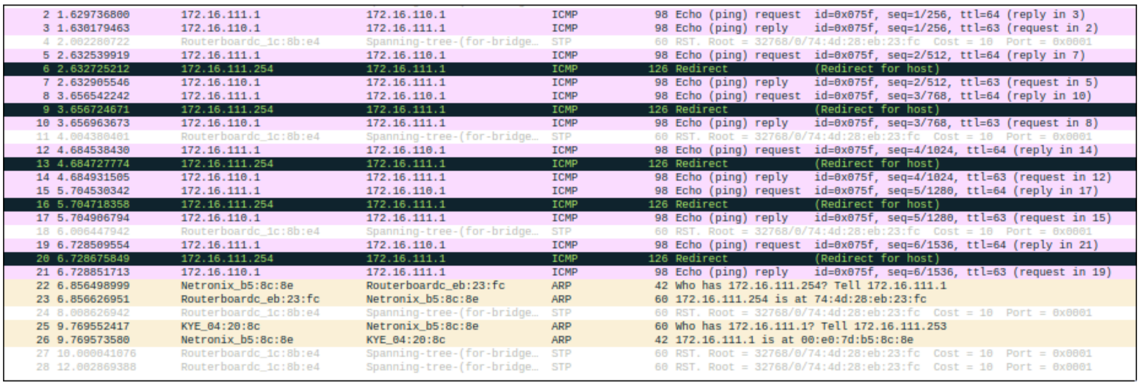
13	17.858680087	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x13d5, seq=1/256, ttl=64 (reply in 14)
14	17.858813765	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13d5, seq=1/256, ttl=64 (request in 13)
15	18.020284419	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
16	18.886989901	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x13d5, seq=2/512, ttl=64 (reply in 17)
17	18.887116455	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13d5, seq=2/512, ttl=64 (request in 16)
18	19.910987710	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x13d5, seq=3/768, ttl=64 (reply in 19)
19	19.911094080	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13d5, seq=3/768, ttl=64 (request in 18)
20	20.022533071	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
21	20.934995089	172.16.110.1	172.16.110.254	ICMP	98	Echo (ping) request id=0x13d5, seq=4/1024, ttl=64 (reply in 22)
22	20.935093008	172.16.110.254	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13d5, seq=4/1024, ttl=64 (request in 21)
23	22.024792132	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
24	22.950951748	Netronix_50:35:0a	Netronix_71:73:ed	ARP	42	Who has 172.16.110.254? Tell 172.16.110.1
25	22.951045895	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	172.16.110.254 is at 00:08:54:71:73:ed
26	23.078341123	Netronix_71:73:ed	Netronix_50:35:0a	ARP	60	Who has 172.16.110.1? Tell 172.16.110.254
27	23.078352089	Netronix_50:35:0a	Netronix_71:73:ed	ARP	42	172.16.110.1 is at 00:08:54:50:35:0a
28	24.027017796	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
29	26.029182806	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
30	28.031347541	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
31	30.033507391	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
32	32.035684145	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
33	34.037856082	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
34	36.040027251	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
35	38.042192066	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
36	39.890557251	172.16.110.1	172.16.111.253	ICMP	98	Echo (ping) request id=0x13e2, seq=1/256, ttl=64 (reply in 37)
37	39.890684009	172.16.111.253	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13e2, seq=1/256, ttl=64 (request in 36)
38	40.044353456	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
39	40.902986165	172.16.110.1	172.16.111.253	ICMP	98	Echo (ping) request id=0x13e2, seq=2/512, ttl=64 (reply in 40)
40	40.903092321	172.16.111.253	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13e2, seq=2/512, ttl=64 (request in 39)
41	41.926985192	172.16.110.1	172.16.111.253	ICMP	98	Echo (ping) request id=0x13e2, seq=3/768, ttl=64 (reply in 42)
42	41.927090091	172.16.111.253	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13e2, seq=3/768, ttl=64 (request in 41)
43	42.046530907	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
44	44.048711149	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
45	46.050880212	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
46	48.053037675	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
47	50.055205189	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
48	50.690179995	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x13e9, seq=1/256, ttl=64 (reply in 49)
49	50.690453557	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13e9, seq=1/256, ttl=63 (request in 48)
50	51.718987117	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x13e9, seq=2/512, ttl=64 (reply in 51)
51	51.719227784	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13e9, seq=2/512, ttl=63 (request in 50)
52	52.057381565	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
53	52.742988854	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x13e9, seq=3/768, ttl=64 (reply in 54)
54	52.743201236	172.16.111.1	172.16.110.1	ICMP	98	Echo (ping) reply id=0x13e9, seq=3/768, ttl=63 (request in 53)
55	53.766986821	172.16.110.1	172.16.111.1	ICMP	98	Echo (ping) request id=0x13e9, seq=4/1024, ttl=64 (reply in 56)

Now, let's disable the acceptance of ICMP redirect messages on tux63. These messages are used to inform the host about a better route to a destination. By disabling it, we can force a packet to take a different route. After that, we can change the routes of tux62 to use the route as a gateway to the subnet 172.16.110/24 instead of tux64. *(1st question 2/2)

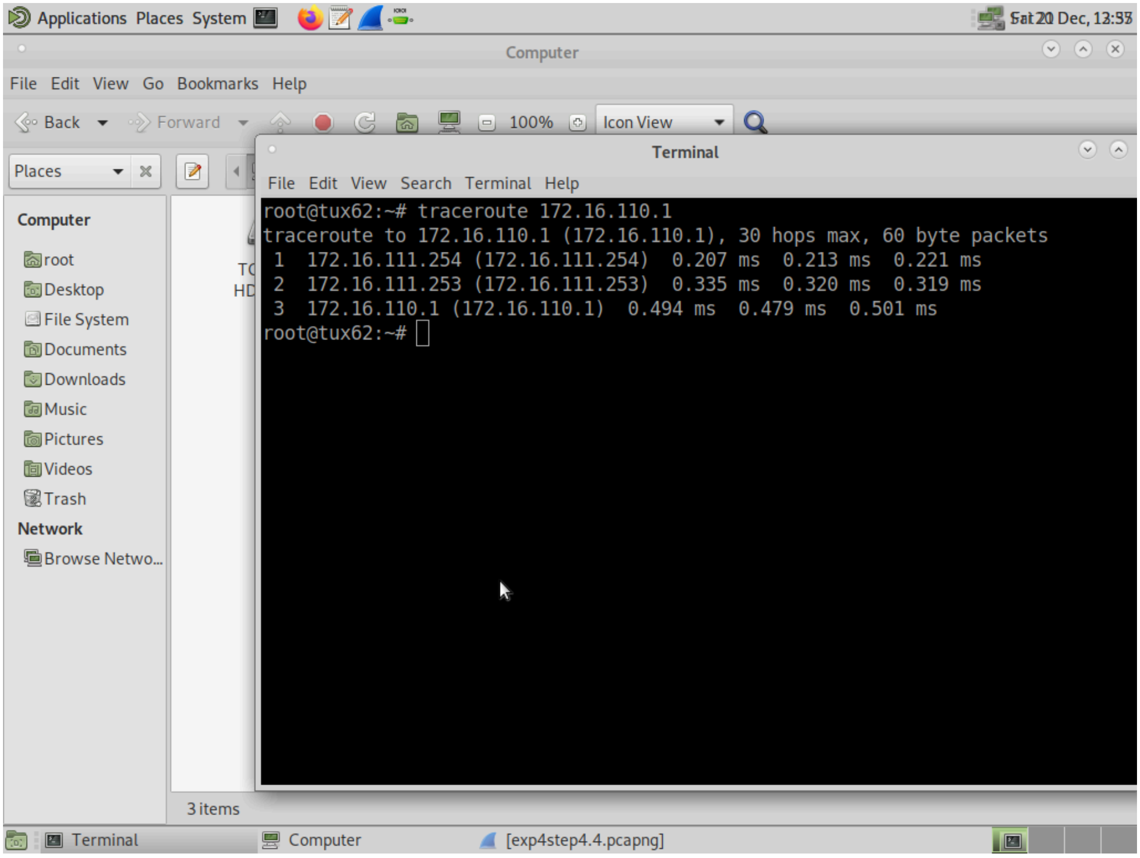
Unset

```
sysctl net.ipv4.conf.eth1.accept_redirects=0
sysctl net.ipv4.conf.all.accept_redirects=0

route del -net 172.16.110.0/24
route add -net 172.16.110.0/24 gw 172.16.111.254
```

By pinging tux63 from tux62, we get the following results:



And by running traceroute 172.16.110.1:



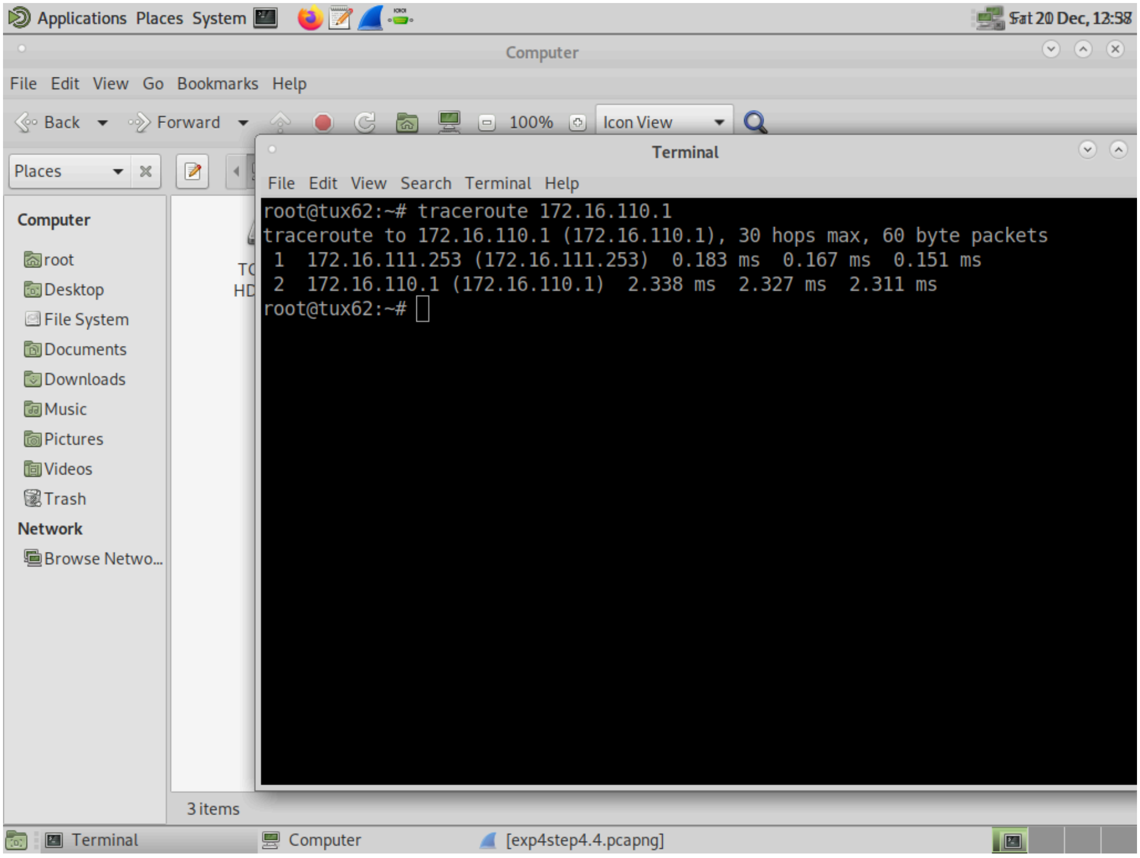
As we can see, for a packet coming from tux62 with the destination being tux63, it hops 3 times:

1. From tux62 to the router
2. From the router to tux64
3. From tux64 to tux63

In addition, it is also possible to verify the existence of ICMP redirect requests, but they are being rejected by tux62 due to the commands that were run previously.

Now, let's change the route from tux62 to subnet 172.16.110.0/24 to its previous path and run traceroute 172.16.110.1 again:

```Unset
route del -net 172.16.110.0/24
route add -net 172.16.110.0/24 gw 172.16.111.253
```

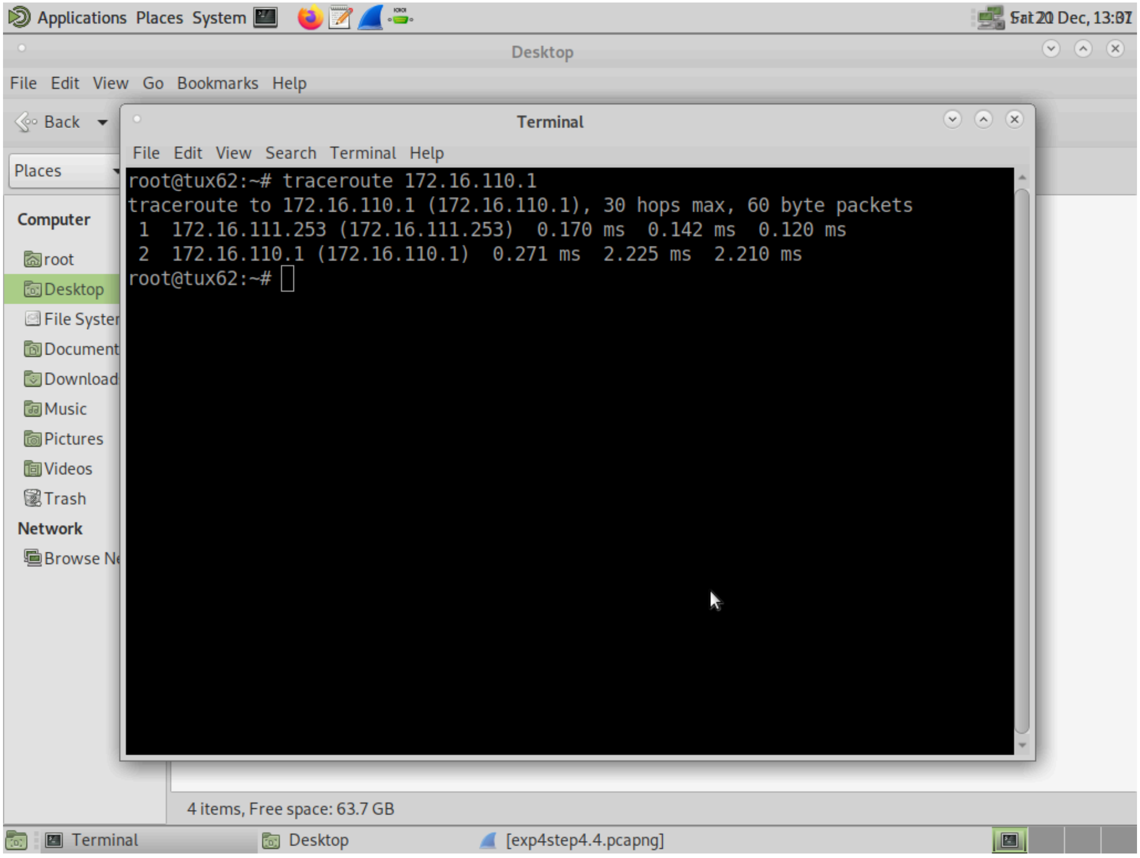


Instead of the 3 hops, now the packet just hops twice, from tux62 to tux64 and from tux64 to tux63.

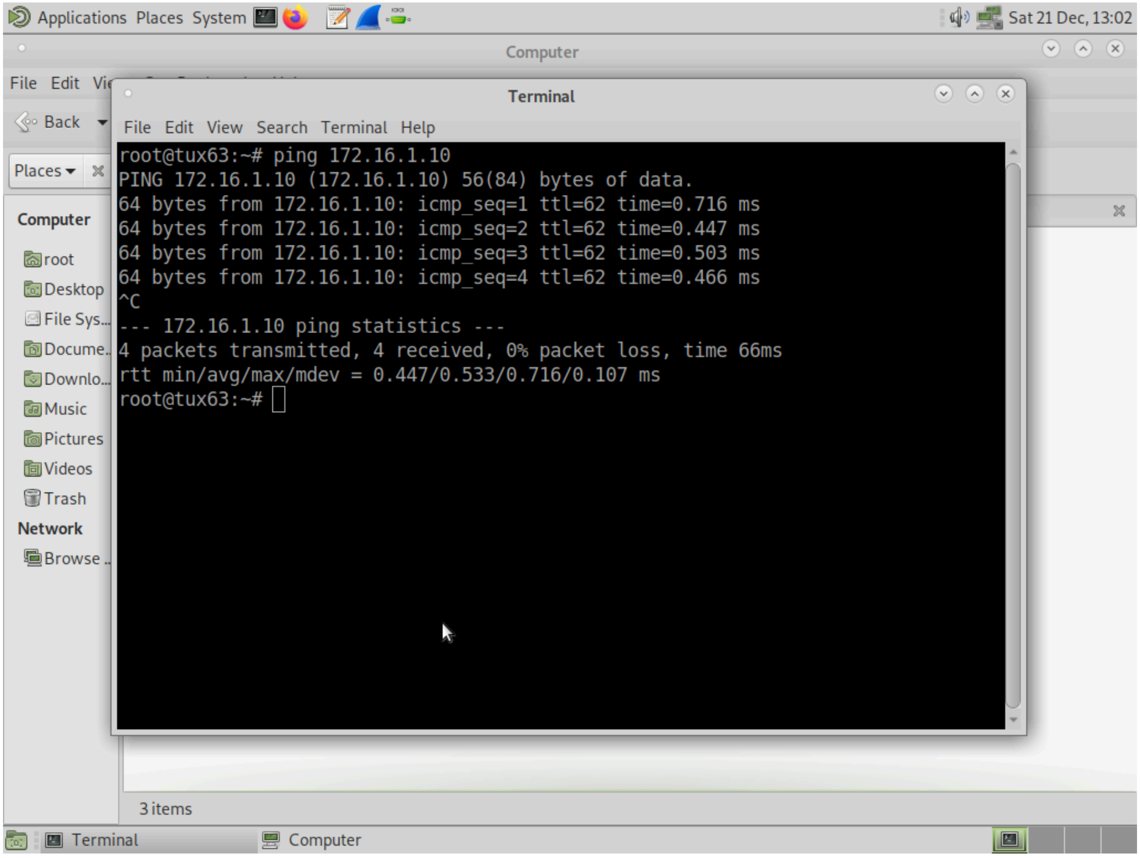
By activating the acceptance of ICMP redirect messages and changing the route once again to go through the router first, we can see that the ICMP redirect messages were accepted, thus the packets will take a shorter route.

Unset

```
sysctl net.ipv4.conf.eth1.accept_redirects=1
sysctl net.ipv4.conf.all.accept_redirects=1
route del -net 172.16.110.0/24
route add -net 172.16.110.0/24 gw 172.16.111.254
```



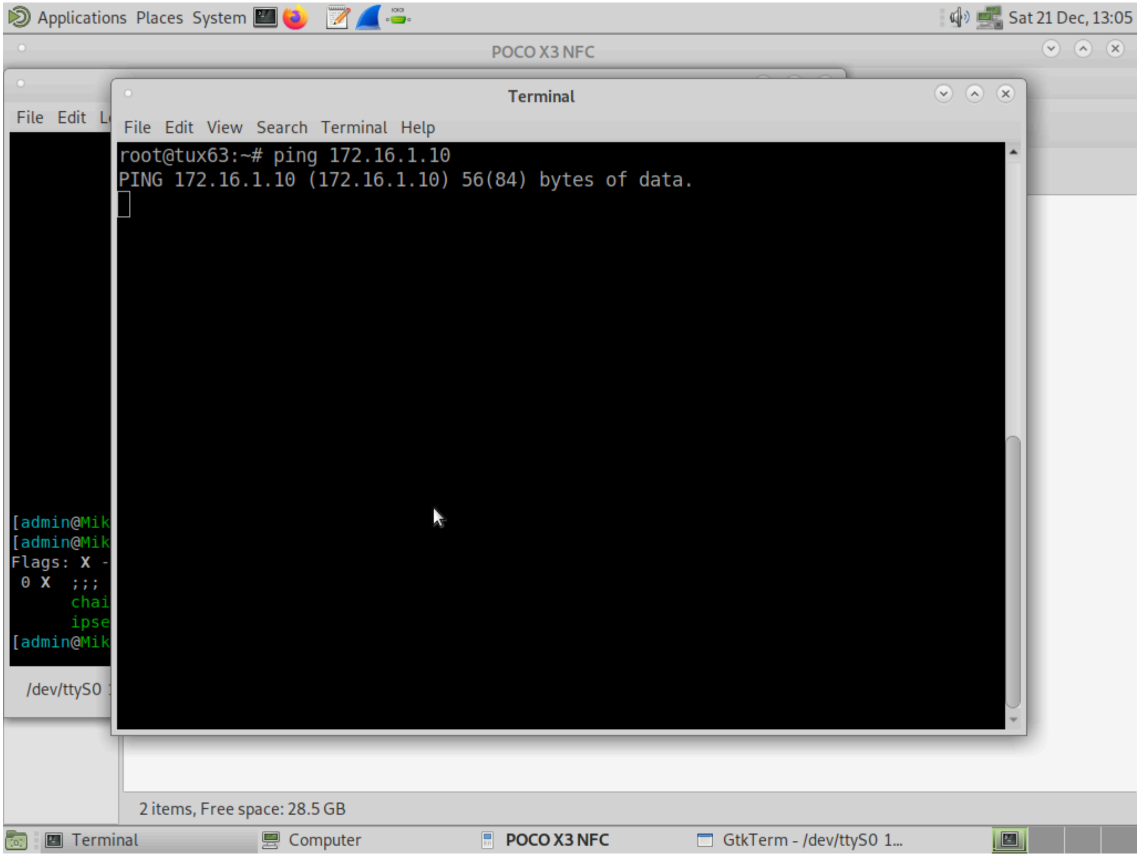
Since we previously added a route from tux63 to the subnet 172.16.1.0/24 (the lab network), we can now ping the FTP server at 172.16.1.10:



By default, the router comes with NAT enabled. NAT is responsible for translating the private IP of each device and changing it to the router's IP, so it is required for devices to communicate with external networks. If we disable it, tux63 can no longer communicate with the FTP server, because the packet's source IP is not being translated to the router's IP address. *(4th and 5th question)

Unset
```
ip firewall nat disable 0 //router
```

*(3rd question)



After this, enable NAT to bring back communication with the external network:

Unset
```
ip firewall nat enable 0 //router
```

3.4.4 Questions

All of the questions were answered throughout the report and tagged with (*1st question) concerning the first question, for example.

The questions are the following:

1st - » How to configure a static route in a commercial router?

2nd - » What are the paths followed by the packets, with and without ICMP redirect enabled, in the experiments carried out and why?

3rd - » How to configure NAT in a commercial router?

4th - » What does NAT do?

5th - » What happens when tuxY3 pings the FTP server with the NAT disabled? Why?

3.5 DNS

If we want to use human-readable names instead of IP addresses to access servers, we need to configure a DNS (Domain Name System). DNS translates these names into IP addresses that computers can use to establish communication.

3.5.1 Objectives

Use a DNS server to access the FPT server using the URL

3.5.2 Physical Connection

N/A

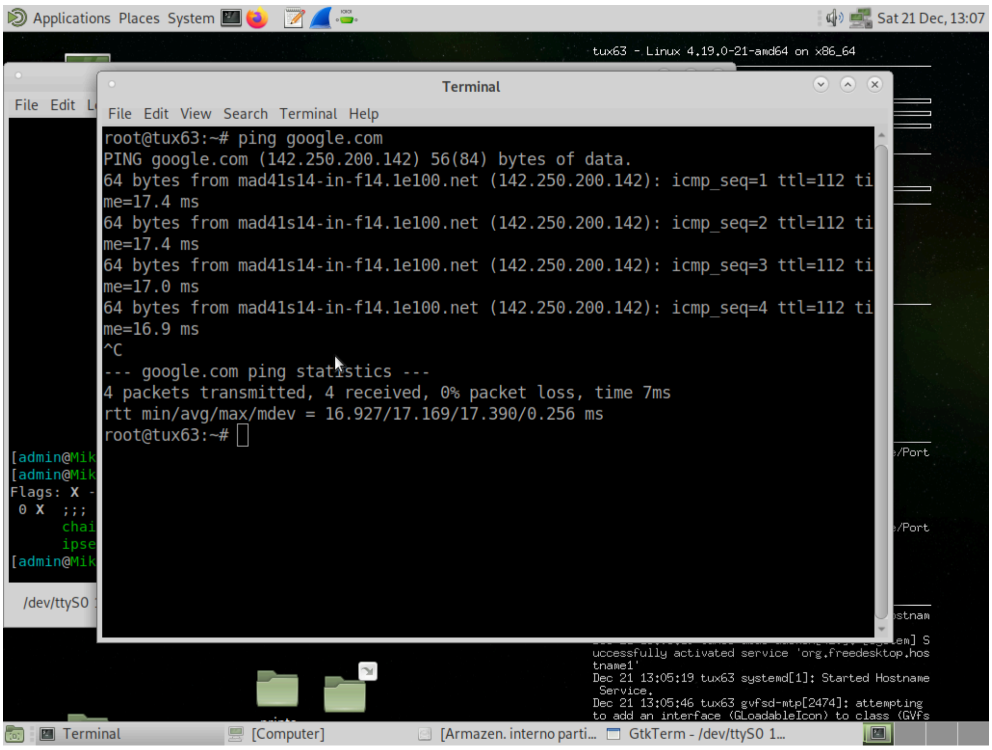
3.5.3 Configuration and Conclusions

We can configure a DNS server by editing the file /etc/resolv.conf:

```
Unset
    sudo vim /etc/resolv.conf
    // add the line "nameserver 10.227.20.3"
```

*(1st question)

Now, we can check if the DNS is working by pinging a common website, like google.com:



If we want to see the DNS messages, we can ping a server that the device has never seen before, such as instagram.com:

3	0.000015714	ASUSTekCOMPU_b3:e9:e8		ARP	62	Who has 192.168.109.114? Tell 192.168.109.113
4	0.000017460	ASUSTekCOMPU_b3:e9:e8		ARP	62	Who has 192.168.109.112? Tell 192.168.109.113
5	0.000019416	ASUSTekCOMPU_b3:e9:e8		ARP	62	Who has 192.168.109.111? Tell 192.168.109.113
6	0.533981582	10.227.20.63	10.227.20.3	DNS	75	Standard query 0x73b1 A instagram.com
7	0.533991709	10.227.20.63	10.227.20.3	DNS	75	Standard query 0xabbc AAAA instagram.com
8	0.591950270	10.227.20.3	10.227.20.63	DNS	103	Standard query response 0xabbc AAAA instagram.com AAAA 2a03:
9	0.591972899	10.227.20.3	10.227.20.63	DNS	91	Standard query response 0x73b1 A instagram.com A 157.240.212
10	0.592373786	10.227.20.63	157.240.212.174	ICMP	100	Echo (ping) request id=0x0b11, seq=1/256, ttl=64 (reply in
11	0.598376344	157.240.212.174	10.227.20.63	ICMP	100	Echo (ping) reply id=0x0b11, seq=1/256, ttl=51 (request i
12	0.598488160	10.227.20.63	10.227.20.3	DNS	90	Standard query 0xf075 PTR 174.212.240.157.in-addr.arpa
13	0.899374447	ASUSTekCOMPU_2e:20:7c		ARP	62	Who has 192.168.109.121? Tell 192.168.109.123
14	0.899388554	ASUSTekCOMPU_2e:20:7c		ARP	62	Who has 192.168.109.122? Tell 192.168.109.123
15	0.899390580	ASUSTekCOMPU_2e:20:7c		ARP	62	Who has 192.168.109.124? Tell 192.168.109.123
16	0.899392396	ASUSTekCOMPU_2e:20:7c		ARP	62	Who has 192.168.109.125? Tell 192.168.109.123

As we can see, the device is asking the DNS server (10.227.20.3) for the IP address corresponding to the domain instagram.com. It then responds with the IP, that is then used to access the destination server. *(2nd question)

3.5.4 Questions

All of the questions were answered throughout the report and tagged with (*1st question) concerning the first question, for example.

The questions are the following:

1st - » How to configure the DNS service in a host?

2nd - » What packets are exchanged by DNS and what information is transported

3.6 TCP connections

3.6.1 Objectives

To test both our FTP download client and the network we configured, we can use our application on tux63 to download a file from the FTP server at ftp.netlab.fe.up.pt.

3.6.2 Physical Connection

N/A

3.6.3 Configuration and Conclusions

1. Edit the Makefile to use the Netlab FTP server:

```
# LAB SERVERS (THESE ONLY WORK INSIDE THE LAB NETWORK)

# URL = ftp://rcom:rcom@ftp.netlab.fe.up.pt/pipe.txt
URL = ftp://rcom:rcom@ftp.netlab.fe.up.pt/files/crab.mp4
# URL = ftp://rcom:rcom@ftp.netlab.fe.up.pt/README
```

2. Run the application with *make run*.
3. Analyse the Wireshark log, and see that tux63 is communicating with the FTP server:

9	0.009427445	172.16.110.1	172.16.1.10	FTP	77	Request: PASS rcom
10	0.052599994	172.16.1.10	172.16.110.1	TCP	66	21 → 34576 [ACK] Seq=83 Ack=23 Win=65280 Len=0 TSval=367601
11	0.158590094	172.16.1.10	172.16.110.1	FTP	113	Response: 230-Welcome, archive user rcom@172.16.1.111 !
12	0.158607345	172.16.1.10	172.16.110.1	FTP	115	Response:
13	0.158751847	172.16.110.1	172.16.1.10	TCP	66	34576 → 21 [ACK] Seq=23 Ack=179 Win=64256 Len=0 TSval=41428
14	0.158774266	172.16.1.10	172.16.110.1	FTP	233	Response:
15	0.159036450	172.16.110.1	172.16.1.10	FTP	72	Request: PASV
16	0.159320214	172.16.1.10	172.16.110.1	TCP	66	21 → 34576 [ACK] Seq=346 Ack=29 Win=65280 Len=0 TSval=36760
17	0.159743592	172.16.1.10	172.16.110.1	FTP	116	Response: 227 Entering Passive Mode (172,16,1,10,139,209).
18	0.160086023	172.16.110.1	172.16.1.10	TCP	74	48688 → 35793 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PER
19	0.160465889	172.16.1.10	172.16.110.1	TCP	74	35793 → 48688 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=14
20	0.160485026	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=41428
21	0.160510518	172.16.110.1	172.16.1.10	FTP	74	Request: TYPE I
22	0.161385628	172.16.1.10	172.16.110.1	FTP	85	Response: 200 Type set to I
23	0.161524961	172.16.110.1	172.16.1.10	FTP	88	Request: SIZE /files/crab.mp4
24	0.162195507	172.16.1.10	172.16.110.1	FTP	80	Response: 213 29803194
25	0.162280504	172.16.110.1	172.16.1.10	FTP	88	Request: RETR /files/crab.mp4
26	0.163184109	172.16.1.10	172.16.110.1	FTP	144	Response: 150 Opening BINARY mode data connection for /file
27	0.163848508	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
28	0.163859753	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=1449 Win=64128 Len=0 TSval=41
29	0.163969403	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
30	0.163975480	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=2897 Win=64128 Len=0 TSval=41
31	0.164092464	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
32	0.164099448	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=4345 Win=64128 Len=0 TSval=41
33	0.164221670	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
34	0.164227537	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=5793 Win=64128 Len=0 TSval=41
35	0.164344171	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
36	0.164349689	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=7241 Win=64128 Len=0 TSval=41
37	0.164467301	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
38	0.164473028	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=8689 Win=64128 Len=0 TSval=41
39	0.164590431	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
40	0.164595879	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=10137 Win=64128 Len=0 TSval=4
41	0.164713631	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
42	0.164719358	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=11585 Win=64128 Len=0 TSval=4
43	0.164835923	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
44	0.164841511	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=13033 Win=64128 Len=0 TSval=4
45	0.164959333	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
46	0.164965339	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=14481 Win=64128 Len=0 TSval=4
47	0.165082952	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
48	0.165088399	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=15929 Win=64128 Len=0 TSval=4
49	0.165205383	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
50	0.165210831	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=17377 Win=64128 Len=0 TSval=4
51	0.165328443	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)
52	0.165334240	172.16.110.1	172.16.1.10	TCP	66	48688 → 35793 [ACK] Seq=1 Ack=18825 Win=64128 Len=0 TSval=4
53	0.165451923	172.16.1.10	172.16.110.1	FTP-DATA	1514	FTP Data: 1448 bytes (PASV) (TYPE I)

(an example of the application output can be found here)

The first step of the TCP connection is establishing the connection between the device and the server. This can be verified by the triple handshake of the SYN, SYN ACK and ACK packets:

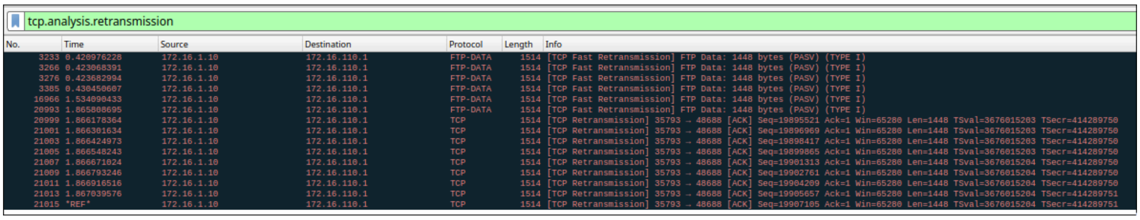
1	0.000000000	172.16.110.1	172.16.1.10	TCP	74	34576 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TS
2	0.000983923	172.16.1.10	172.16.110.1	TCP	74	21 → 34576 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 S
3	0.001007669	172.16.110.1	172.16.1.10	TCP	66	34576 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=414287893
4	0.008167789	172.16.1.10	172.16.110.1	FTP	116	Response: 220 ProFTPD Server (Debian) [::ffff:172.16.1.10]
5	0.008178405	172.16.110.1	172.16.1.10	TCP	66	34576 → 21 [ACK] Seq=1 Ack=51 Win=64256 Len=0 TSval=41428790

Then, the data phase starts. Firstly, the client authenticates with the username and password, followed by requesting the passive mode. In this mode, the server responds with an IP and a port, which is then used by another TCP connection to receive the file.

Finally, the termination phase takes place, using a 4-way handshake with the FYN and ACK packets: *(3rd question)

31080	13.431933393	172.16.110.1	172.16.1.10	TCP	66	34576 → 21 [FIN, ACK] Seq=87 Ack=544 Win=64128 Len=0 TSval=
31081	13.434535676	172.16.1.10	172.16.110.1	TCP	66	21 → 34576 [FIN, ACK] Seq=544 Ack=88 Win=65280 Len=0 TSval=

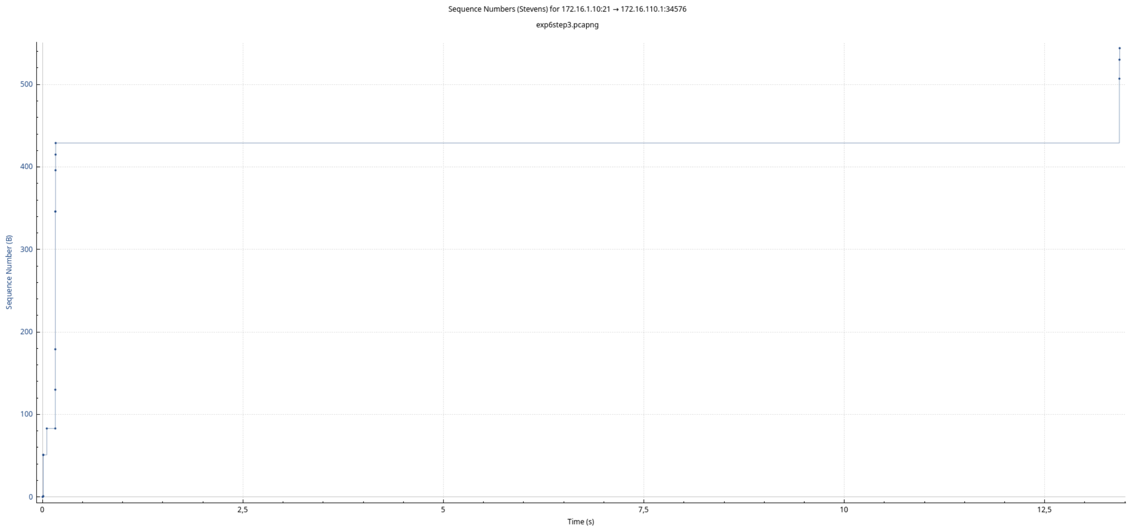
To ensure data consistency during transfer, the TCP protocol employs ARQ (Automatic Repeat Request) mechanisms. If packet loss or errors occur, the sender retransmits packets upon detecting duplicate acknowledgments or timeout events. These mechanisms can be seen by filtering the Wireshark capture to only show retransmission packets:



tcp.analysis.retransmission

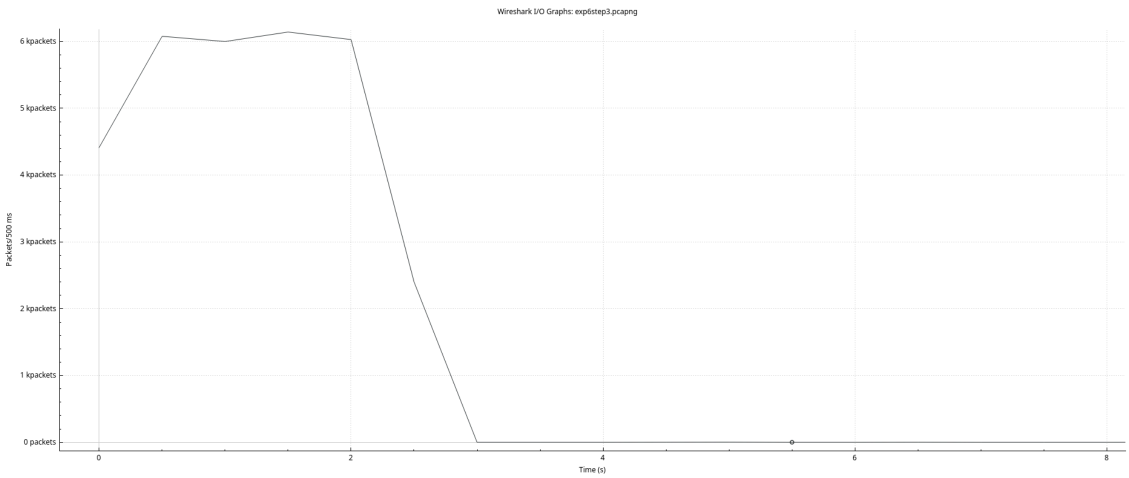
No.	Time	Source	Destination	Protocol	Length	Info
3233	0.420976228	172.16.1.10	172.16.110.1	FTP-DATA	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes (PASV) (TYPE I)
3266	0.423068391	172.16.1.10	172.16.110.1	FTP-DATA	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes (PASV) (TYPE I)
3276	0.423682994	172.16.1.10	172.16.110.1	FTP-DATA	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes (PASV) (TYPE I)
3385	0.430450607	172.16.1.10	172.16.110.1	FTP-DATA	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes (PASV) (TYPE I)
16966	1.534090433	172.16.1.10	172.16.110.1	FTP-DATA	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes (PASV) (TYPE I)
20993	1.865808695	172.16.1.10	172.16.110.1	FTP-DATA	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes (PASV) (TYPE I)
20999	1.866178364	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19895521 Ack=1 Win=65280 Len=1448 TSval=3676015203 TSecr=414289750
21001	1.866301634	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19896969 Ack=1 Win=65280 Len=1448 TSval=3676015203 TSecr=414289750
21003	1.866424973	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19898417 Ack=1 Win=65280 Len=1448 TSval=3676015203 TSecr=414289750
21005	1.866548243	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19899865 Ack=1 Win=65280 Len=1448 TSval=3676015203 TSecr=414289750
21007	1.866671024	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19901313 Ack=1 Win=65280 Len=1448 TSval=3676015204 TSecr=414289750
21009	1.866793246	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19902761 Ack=1 Win=65280 Len=1448 TSval=3676015204 TSecr=414289750
21011	1.866916516	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19904209 Ack=1 Win=65280 Len=1448 TSval=3676015204 TSecr=414289750
21013	1.867039576	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19905657 Ack=1 Win=65280 Len=1448 TSval=3676015204 TSecr=414289751
21015	*REF*	172.16.1.10	172.16.110.1	TCP	1514	[TCP Retransmission] 35793 → 48688 [ACK] Seq=19907105 Ack=1 Win=65280 Len=1448 TSval=3676015204 TSecr=414289751

Alongside ARQ, TCP's congestion control mechanisms like slow start (starting slow and increasing the transmission rate over time, and congestion avoidance, which prevents the transmission rate from exceeding the network capacity) ensure that packets do not overwhelm the network. These mechanisms can be seen by analyzing the time-sequence graph of the Wireshark capture:

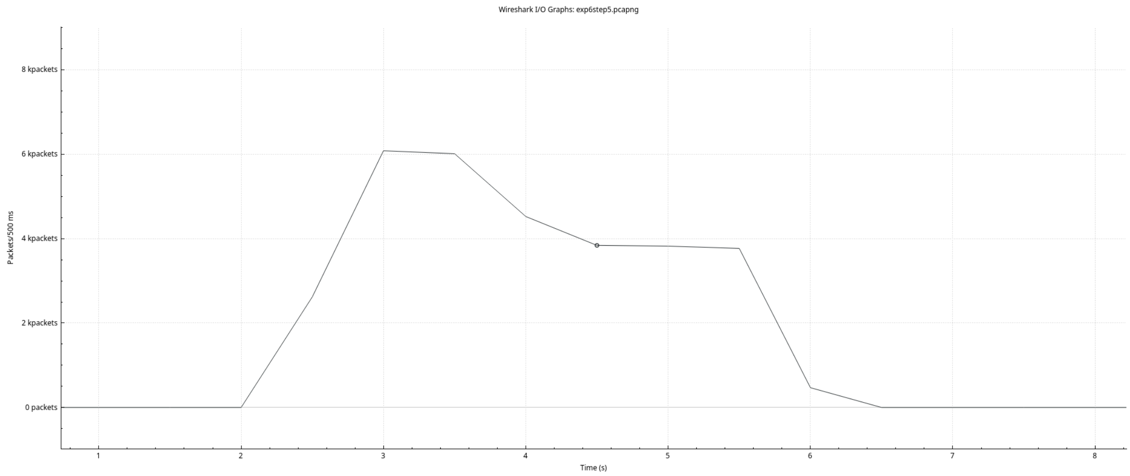


The increases in the sequence number represent the slow start mechanism and the part where the sequence number is constant represents the congestion avoidance mechanism.

The overall transmission rate can be seen by analyzing the I/O graph:



If we want to study the impact of simultaneous connections, we can start another transfer, for example in tux62, while the first transfer is in progress, and analyze the I/O graph:



We can clearly see the moment when only 1 transfer was in progress (around 6000 packets per 500ms) and the moment when there were 2 transfers (around 4000 packets per 500ms.

This is due to the fact that the 2 connections were following a similar path, so we reached the limit of the network. *(6th question)

Questions:

» How many TCP connections are opened by your FTP application?

R: 2, one for the initial communication and control, and another for receiving the file.

» In what connection is transported the FTP control information?

R: The control information is transported in the first connection (the control connection)

» What are the phases of a TCP connection?

R: Answer marked with *(3rd question) above.

» How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?

R: The ARQ is a data reliability mechanism. It works by sending data packets and waiting for an acknowledgment response (ACK). If there is a timeout while waiting for the ACK, or if it receives an ACK that does not correlate with the sent packet, it will mark the packet as a loss, and retransmit it. The relevant TCP fields for AQR are the sequence number (identifies the order of the data packets), the ACK number (indicates the next expected packet), ACK flag (if present, indicates that the ACK is present), windows size (amount of data the receiver accepts) and the checksum (for validating the integrity of the data). The ARQ mechanism was shown in action previously.

» How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according to the TCP congestion control mechanism?

R: The congestion control mechanism was already previously demonstrated. Its relevant fields are, like in the ARQ package, the sequence number, ACK number, windows size, and flags (except in this case the flags can be SYN or ACK, and are used to represent the different events of the connection). In addition, this mechanism can be seen in action in the I/O graph, because when the connection starts, its speed gradually increases until it is constant.

» Is the throughput of a TCP data connections disturbed by the appearance of a second TCP connection? How?

R: Answer marked with *(6th question) above.

4 - Conclusions

To conclude this report, we successfully configured and debugged a network in Linux, as well as configured switches and routers to manage traffic. This project provided us with valuable hands-on experience in networking, particularly in protocols such as FTP, IP, and bridges. We implemented a robust FTP client, worked with Linux-based systems, and explored both commercial routers and network devices. Additionally, we used Wireshark for packet analysis and gained practical insights into network troubleshooting.

5 - References

All references were taken from the UC page at Moodle.

6 - Annexes

1. main.c

C/C++

```c
#include "include/download.h"
#include <stdio.h>
#include <time.h>
#include <unistd.h>
```

```
        int main(int argc, char *argv[]) {
          if (argc != 2) {
                                    fprintf(stderr,      "Usage:      %s
ftp://[<user>:<password>@]<host>/<url-path>\n",
                    argv[0]);
            return -1;
          }

          struct timespec start_time;
          clock_gettime(CLOCK_MONOTONIC, &start_time);

          char *host = argv[1];
          UrlInfo info;
          if (parse_url(host, &info) != 0) {
            perror("Error parsing the URL.\n");
            return -1;
          }

          print_url_info(&info);

          int socket1;

          if (establish_connection(&info, &socket1) != 0) {
            perror("Error establishing connection.\n");
            return -1;
          }

          if (login(socket1, &info) != 0) {
            perror("Error logging in.\n");
            close_connection(socket1, -1);
            return -1;
          }

          if (enter_passive_mode(socket1, &info) != 0) {
            perror("Error entering passive mode.\n");
            close_connection(socket1, -1);
            return -1;
          }

          print_url_info(&info);

          int socket2;
             if   (connect_to_socket(info.passive_ip,   info.passive_port,
&socket2) != 0) {
            perror("Error connecting to the passive socket.\n");
            close_connection(socket1, socket2);
            return -1;
          }

          if (download_file(socket1, socket2, &info) != 0) {
            perror("Error downloading the file.\n");
            close_connection(socket1, socket2);
            return -1;
          }
```

```
    close_connection(socket1, socket2);

    print_url_info(&info);

    print_statistics(&info, &start_time);
    return 0;
  }
```

2. download.c

```
C/C++
  #include "../include/download.h"
  #include <arpa/inet.h>
  #include <fcntl.h>
  #include <netdb.h>
  #include <netinet/in.h>
  #include <stdio.h>
  #include <stdlib.h>
  #include <strings.h>
  #include <sys/socket.h>
  #include <time.h>
  #include <unistd.h>

  #include <string.h>

  void print_url_info(UrlInfo *info) {
    printf("\n========== URL Information ==========\n");
    printf("User          : %s\n", strlen(info->user) ? info->user :
"N/A");
      printf("Password       :  %s\n",  strlen(info->password)  ?
info->password : "N/A");
    printf("Host          : %s\n", strlen(info->host) ? info->host :
"N/A");
     printf("IP             : %s\n", strlen(info->ip) ? info->ip :
"N/A");
    printf("Port         : %d\n", info->port);
    printf("Path          : %s\n", strlen(info->path) ? info->path :
"/");
      printf("Filename       :  %s\n",  strlen(info->filename)  ?
info->filename : "N/A");
    printf("Passive IP   : %s\n",
           strlen(info->passive_ip) ? info->passive_ip : "N/A");
    if (info->passive_port == 0) {
      printf("Passive Port: N/A\n");
    } else {
      printf("Passive Port: %d\n", info->passive_port);
    }
    if (info->file_size == 0) {
```

```
    printf("File Size   : N/A\n");
  } else {
    printf("File Size   : %d bytes\n", info->file_size);
  }
  printf("=====================================\n");
}

int parse_url(char *host, UrlInfo *info) {
  if (host == NULL || info == NULL) {
    perror("Invalid arguments when parsing the url.\n");
    return -1;
  }

  // Validate the prefix.
  const char *prefix = "ftp://";
  if (strncmp(host, prefix, strlen(prefix)) != 0) {
    perror("URL does not start with 'ftp://'.\n");
    return -1;
  }
  const char *cursor = host + strlen(prefix);

  memset(info, 0, sizeof(UrlInfo));

  // Get the username and, optionally, the password.
  const char *at = strchr(cursor, '@');
  if (at) {
    const char *colon = strchr(cursor, ':');
    if (colon && colon < at) {
      // User and password (<user>:<password>@<host>)
      strncpy(info->user, cursor, colon - cursor);
      strncpy(info->password, colon + 1, at - colon - 1);
    } else {
      // No password (<user>@<host>)
      strncpy(info->user, cursor, at - cursor);
    }
    cursor = at + 1; // Move cursor past '@'
  }

  // Get the host and, optionally, the port.
  const char *slash = strchr(cursor, '/');
  const char *colon = strchr(cursor, ':');
  if (colon && (!slash || colon < slash)) {
    // Host:Port
    strncpy(info->host, cursor, colon - cursor);
    info->port = atoi(colon + 1);
  } else {
    // Host (no port)
    info->port = 21;
    if (slash) {
      strncpy(info->host, cursor, slash - cursor);
    } else {
      strcpy(info->host, cursor); // No path; host is the rest of
the URL
    }
  }
```

```
    // Get the path if it exists.
    if (slash) {
      strcpy(info->path, slash);
    } else {
      perror("No path found in the url.\n");
      return -1;
    }

    // Get the filename.
    const char *last_slash = strrchr(info->path, '/');
    if (last_slash) {
      if (*(last_slash + 1) == '\0') {
        perror("No filename found in the url.\n");
        return -1;
      }
      strcpy(info->filename, last_slash + 1);
    } else {
      perror("No filename found in the url.\n");
      return -1;
    }

    // Get the ip address.
    if (get_ip(info->host, info->ip) != 0) {
      return -1;
    }

    return 0;
  }

  int get_ip(char *host, char *ip) {

    struct hostent *h;
    if ((h = gethostbyname(host)) == NULL) {
      herror("gethostbyname()");
      return -1;
    }

      const char *resolved_ip = inet_ntoa(*((struct in_addr
*)h->h_addr));
    if (resolved_ip == NULL) {
      perror("Failed to get the ip address.\n");
      return -1;
    }
    strcpy(ip, resolved_ip);
    return 0;
  }

  int connect_to_socket(const char *ip, const int port, int
*socket_fd) {

    if (ip == NULL || socket_fd == NULL) {
      return -1;
    }
```

```
        int sockfd;
        struct sockaddr_in server_addr;

        /*Server address handling*/
        bzero((char *)&server_addr, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr =
               inet_addr(ip);  /*32 bit Internet address network byte
ordered*/
        server_addr.sin_port =
            htons(port); /*Server TCP port must be network byte ordered
*/

        /*Open a TCP socket*/
        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("socket()");
          return -1;
        }
        /*Connect to the server*/
            if   (connect(sockfd,   (struct   sockaddr   *)&server_addr,
sizeof(server_addr)) <
            0) {
          perror("connect()");
          return -1;
        }

        *socket_fd = sockfd;
        return 0;
      }

      int establish_connection(const UrlInfo *info, int *socket_fd) {
        if (info == NULL || socket_fd == NULL) {
          return -1;
        }

        if (connect_to_socket(info->ip, info->port, socket_fd) != 0) {
          perror("Error connecting to the socket.\n");
          return -1;
        }

        // Read the response.
        char response[1024] = "";
        int response_code = 0;
        if (read_response(*socket_fd, response, &response_code) != 0) {
          return -1;
        }

        if (response_code != 220) {
          perror("Error establishing connection.\n");
          return -1;
        }

        return 0;
      }
```

```
      int  read_response(const  int  socket_fd,  char  *response,  int
*response_code) {
        if (response == NULL || response_code == NULL) {
          return -1;
        }

        enum state current_state = CODE;

        int message_index = 0;
        *response_code = 0;

        while (current_state != STOP) {
          char current_char = 0;
          int bytes_read = read(socket_fd, &current_char, 1);
          if (bytes_read < 0) {
            perror("Error reading from the socket.\n");
            return -1;
          }
          if (bytes_read == 0) {
            break;
          }

          switch (current_state) {
          case CODE:
            if (current_char == '\n') {
              current_char = STOP;
            } else if (current_char == ' ') {
              current_state = RESPONSE;
            } else if (current_char == '-') {
              current_state = MESSAGE;
            } else if (current_char >= '0' && current_char <= '9') {
                *response_code = *response_code * 10 + (current_char -
'0');
            }
            break;
          case MESSAGE:
            if (current_char == '\n') {
              current_state = CODE;
              *response_code = 0;
            }
            break;

          case RESPONSE:
            if (*response_code < 100) {
              current_state = MESSAGE;
              break;
            }

            if (current_char == '\n') {
              response[message_index] = '\0';
              current_state = STOP;
            } else {
              response[message_index++] = current_char;
            }
            break;
```

```
      case STOP:
        break;
      }
    }
    printf("\n======= Response Informations =======\n");
    printf("Response Code    : %d\n", *response_code);
    printf("Response Message: %s\n", response);
    printf("=====================================\n");

    return 0;
}

int send_message(const int socket_fd, const char *message) {
    if (message == NULL) {
      return -1;
    }
    printf("\n========== Sending Message ==========\n");
    printf("Message: %s", message);
    printf("=====================================\n");
    if (write(socket_fd, message, strlen(message)) < 0) {
      perror("Error writing to the socket.\n");
      return -1;
    }
    return 0;
}

int login(const int socket_fd, const UrlInfo *info) {

    if (info == NULL) {
      return -1;
    }

    // Send the username.
    char user[256] = "USER ";
    if (strlen(info->user) > 0) {
      strcat(user, info->user);
    } else {
      strcat(user, "anonymous");
    }
    strcat(user, "\r\n");
    if (send_message(socket_fd, user) != 0) {
      return -1;
    }

    // Read the response.
    char response[1024] = "";
    int response_code = 0;
    if (read_response(socket_fd, response, &response_code) != 0) {
      return -1;
    }

    if (response_code != 331) {
      perror("Error logging in.\n");
      return -1;
```

```c
  }

  // Send the password if it exists.
  char pass[256] = "PASS ";
  if (strlen(info->password) > 0) {
    strcat(pass, info->password);
  } else {
    strcat(pass, "anonymous");
  }

  strcat(pass, "\r\n");
  if (send_message(socket_fd, pass) != 0) {
    return -1;
  }
  if (read_response(socket_fd, response, &response_code) != 0) {
    return -1;
  }
  if (response_code != 230) {
    perror("Error logging in.\n");
    return -1;
  }

  printf("Logged in successfully.\n");

  return 0;
}

int get_file_size(const int socket_fd, UrlInfo *info) {
  if (info == NULL) {
    return -1;
  }

  char retrieve[1024] = "SIZE ";
  strcat(retrieve, info->path);
  strcat(retrieve, "\r\n");
  if (send_message(socket_fd, retrieve) != 0) {
    return -1;
  }

  // Read the response.
  char response[8192] = "";
  int response_code = 0;
  if (read_response(socket_fd, response, &response_code) != 0) {
    return -1;
  }

  if (response_code != 213) {
    perror("Error getting the file size.\n");
    return -1;
  }

  info->file_size = atoi(response);
  return 0;
}
```

```
int enter_passive_mode(const int socket_fd, UrlInfo *info) {
  if (info == NULL) {
    return -1;
  }

  if (send_message(socket_fd, "PASV\r\n") != 0) {
    return -1;
  }

  // Read the response.
  char response[1024] = "";
  int response_code = 0;
  if (read_response(socket_fd, response, &response_code) != 0) {
    return -1;
  }

  if (response_code != 227) {
    perror("Error entering passive mode.\n");
    return -1;
  }

  // Parse the passive mode response.
  char *start = strchr(response, '(');
  char *end = strchr(response, ')');
  if (start == NULL || end == NULL) {
    perror("Error parsing the passive mode response.\n");
    return -1;
  }

  int ip1, ip2, ip3, ip4, port1, port2;
  sscanf(start, "(%d,%d,%d,%d,%d,%d)", &ip1, &ip2, &ip3, &ip4,
&port1, &port2);
  sprintf(info->passive_ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);
  info->passive_port = port1 * 256 + port2;
  return 0;
}

int download_file(const int socket_fd1, const int socket_fd2,
UrlInfo *info) {
  if (info == NULL) {
    return -1;
  }

  // Set the FTP mode to binary.
  if (send_message(socket_fd1, "TYPE I\r\n") != 0) {
    return -1;
  }

  // Read the response.
  char response[8192] = "";
  int response_code = 0;
  if (read_response(socket_fd1, response, &response_code) != 0) {
    return -1;
  }
```

```c
        if (response_code != 200) {
          perror("Error setting the FTP mode to binary.\n");
          return -1;
        }

        if (get_file_size(socket_fd1, info) != 0) {
          perror("Error getting the file size.\n");
          close_connection(socket_fd1, -1);
          return -1;
        }

        // Send the retrieve command.
        char retrieve[1024] = "RETR ";
        strcat(retrieve, info->path);
        strcat(retrieve, "\r\n");
        if (send_message(socket_fd1, retrieve) != 0) {
          return -1;
        }

        // Wait until the file finishes downloading.
        memset(response, 0, sizeof(response));
        response_code = 0;

        if (read_response(socket_fd1, response, &response_code) != 0) {
          return -1;
        }

        if (response_code != 150 && response_code != 125) {
          perror("Error downloading the file.\n");
          return -1;
        }

        // Create the file.
        FILE *file = fopen(info->filename, "wb");
        if (file == NULL) {
          perror("Error creating the file.\n");
          return -1;
        }

        printf("\nDownloading file...\n");
        struct timespec start_time;
        clock_gettime(CLOCK_MONOTONIC, &start_time);

        // Read the file.
        char buffer[1024];
        int bytes_read;
         while ((bytes_read = read(socket_fd2, buffer, sizeof(buffer)))
> 0) {
          fwrite(buffer, 1, bytes_read, file);
                     print_progress_bar(ftell(file),    info->file_size,
&start_time);
        }
        printf("\nDownload complete.\n");

        // Verify if the file was successfully downloaded.
```

```
    response_code = 0;
    memset(response, 0, sizeof(response));
    if (read_response(socket_fd1, response, &response_code) != 0) {
      return -1;
    }

    if (response_code != 226) {
      perror("Error downloading the file.\n");

      return -1;
    }

    // Close the file.
    fclose(file);

    return 0;
  }

  void print_progress_bar(int progress, int total, struct timespec
*start_time) {
    int bar_width = 50;
    float progress_ratio = (float)progress / total;
    int bar_progress = bar_width * progress_ratio;
    struct timespec current_time;
    clock_gettime(CLOCK_MONOTONIC, &current_time);
          double    elapsed_time    =    (current_time.tv_sec    -
start_time->tv_sec) +
                                                (current_time.tv_nsec  -
start_time->tv_nsec) / 1e9;
        double  remaining_time   =  elapsed_time   /  progress_ratio  -
elapsed_time;
    printf("\r[");
    for (int i = 0; i < bar_width; i++) {
      if (i < bar_progress) {
        printf("=");
      } else {
        printf(" ");
      }
    }
     printf("] %.2f%% - Remaining Time: %.2f s", progress_ratio *
100,
           remaining_time);
    fflush(stdout);
  }

  int close_connection(const int socket_fd1, const int socket_fd2)
{

    if (socket_fd2 != -1) {
      if (send_message(socket_fd2, "QUIT\r\n") != 0) {
        return -1;
      }
      if (close(socket_fd2) < 0) {
        perror("Error closing the connection.\n");
        return -1;
```

```
    }
  }

  if (socket_fd1 != -1) {
    if (send_message(socket_fd1, "QUIT\r\n") != 0) {
      return -1;
    }
    char response[1024] = "";
    int response_code = 0;
    read_response(socket_fd1, response, &response_code);
    if (close(socket_fd1) < 0) {
      perror("Error closing the connection.\n");
      return -1;
    }
  }
  return 0;
}

void  print_statistics(const  UrlInfo  *info,  struct  timespec
*start_time) {
  struct timespec end_time;
  clock_gettime(CLOCK_MONOTONIC, &end_time);
  double elapsed_time = (end_time.tv_sec - start_time->tv_sec) +
                        (end_time.tv_nsec - start_time->tv_nsec)
/ 1e9;

  FILE *fp = fopen(info->filename, "r");
  fseek(fp, 0L, SEEK_END);
  int size = ftell(fp);

  printf("\n========== Statistics ==========\n");
  printf("Elapsed Time : %.2f seconds\n", elapsed_time);
  printf("File Size    : %d bytes\n", size);
  printf("Transfer Rate: %.2f bytes/s\n", size / elapsed_time);
  printf("================================\n");
}
```

3. FTP download example

```
Unset
./bin//download
ftp://anonymous:anonymous@ftp.bit.nl/speedtest/100mb.bin

========== URL Information ==========
User          : anonymous
Password      : anonymous
Host          : ftp.bit.nl
IP            : 213.136.12.213
Port          : 21
```

```
        Path          : /speedtest/100mb.bin
        Filename      : 100mb.bin
        Passive IP   : N/A
        Passive Port: N/A
        File Size    : N/A
        =====================================

        ======= Response Informations =======
        Response Code    : 220
        Response Message: Welcome to ftp.bit.nl
        =====================================

        ========== Sending Message ==========
        Message: USER anonymous
        =====================================

        ======= Response Informations =======
        Response Code    : 331
        Response  Message:  Anonymous  login  ok,  send  your  complete  email
address as your password
        =====================================

        ========== Sending Message ==========
        Message: PASS anonymous
        =====================================

        ======= Response Informations =======
        Response Code    : 230
        Response Message: Anonymous access granted, restrictions apply
        =====================================
        Logged in successfully.

        ========== Sending Message ==========
        Message: PASV
        =====================================

        ======= Response Informations =======
        Response Code    : 227
        Response Message: Entering Passive Mode (213,136,12,213,165,119).
        =====================================

        ========== URL Information ==========
        User          : anonymous
        Password      : anonymous
        Host          : ftp.bit.nl
        IP            : 213.136.12.213
        Port          : 21
        Path          : /speedtest/100mb.bin
        Filename      : 100mb.bin
        Passive IP   : 213.136.12.213
        Passive Port: 42359
        File Size    : N/A
        =====================================

        ========== Sending Message ==========
```

```
Message: TYPE I
=====================================

======= Response Informations =======
Response Code   : 200
Response Message: Type set to I
=====================================

========== Sending Message ==========
Message: SIZE /speedtest/100mb.bin
=====================================

======= Response Informations =======
Response Code   : 213
Response Message: 104857600
=====================================

========== Sending Message ==========
Message: RETR /speedtest/100mb.bin
=====================================

======= Response Informations =======
Response Code   : 150
Response  Message:  Opening  BINARY  mode  data  connection  for
/speedtest/100mb.bin (104857600 bytes)
=====================================

Downloading file...
[==================================================]   100.00%   -
Remaining Time: 0.00 s
Download complete.

======= Response Informations =======
Response Code   : 226
Response Message: Transfer complete
=====================================

========== Sending Message ==========
Message: QUIT
=====================================

========== Sending Message ==========
Message: QUIT
=====================================

======= Response Informations =======
Response Code   : 221
Response Message: Goodbye.
=====================================

========== URL Information ==========
User          : anonymous
Password      : anonymous
Host          : ftp.bit.nl
IP            : 213.136.12.213
```

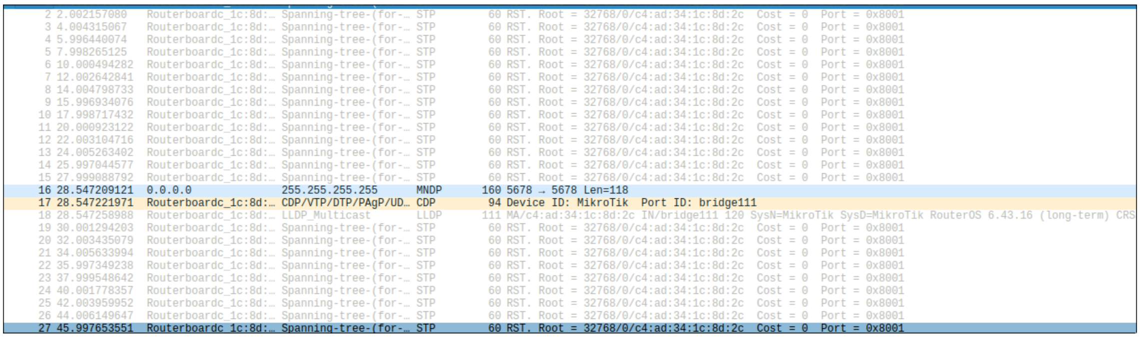
```
Port         : 21
Path         : /speedtest/100mb.bin
Filename     : 100mb.bin
Passive IP   : 213.136.12.213
Passive Port: 42359
File Size    : 104857600 bytes
=====================================

========== Statistics ==========
Elapsed Time : 48.46 seconds
File Size     : 104857600 bytes
Transfer Rate: 2163668.37 bytes/s
================================
```

4. Exp 2 broadcast captures

4.1 Pinging broadcast from tux63

Command: *ping -b 172.16.110.255*

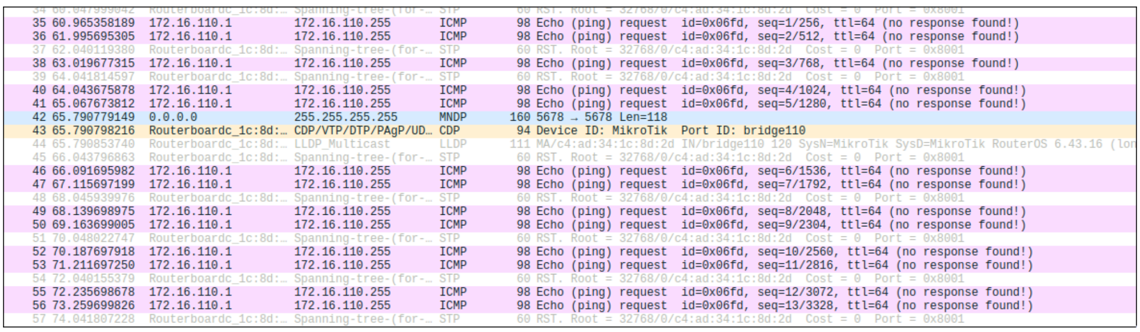
Tux62:



```
 2 2.002157080  Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
 3 4.004315067  Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
 4 5.996440074  Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
 5 7.998265125  Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
 6 10.000494282 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
 7 12.002642841 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
 8 14.004798733 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
 9 15.996934076 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
10 17.998717432 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
11 20.000923122 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
12 22.003104716 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
13 24.005263402 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
14 25.997044577 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
15 27.999088792 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
16 28.547209121 0.0.0.0              255.255.255.255      MNDP 160 5678 → 5678 Len=118
17 28.547221971 Routerboardc_1c:8d:… CDP/VTP/DTP/PAgP/UD… CDP   94 Device ID: MikroTik  Port ID: bridge111
18 28.547258988 Routerboardc_1c:8d:… LLDP_Multicast       LLDP 111 MA/c4:ad:34:1c:8d:2c IN/bridge111 120 SysN=MikroTik SysD=MikroTik RouterOS 6.43.16 (long-term) CRS
19 30.001294203 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
20 32.003435079 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
21 34.005633994 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
22 35.997349238 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
23 37.999548642 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
24 40.001778357 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
25 42.003959952 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
26 44.006149647 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
27 45.997653551 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2c  Cost = 0  Port = 0x8001
```

(Nothing happens)

Tux63:



```
35 60.965358189 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=1/256, ttl=64 (no response found!)
36 61.995695305 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=2/512, ttl=64 (no response found!)
37 62.040119380 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2d  Cost = 0  Port = 0x8001
38 63.019677315 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=3/768, ttl=64 (no response found!)
39 64.041814597 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2d  Cost = 0  Port = 0x8001
40 64.043675878 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=4/1024, ttl=64 (no response found!)
41 65.067673812 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=5/1280, ttl=64 (no response found!)
42 65.790779149 0.0.0.0              255.255.255.255      MNDP 160 5678 → 5678 Len=118
43 65.790798216 Routerboardc_1c:8d:… CDP/VTP/DTP/PAgP/UD… CDP   94 Device ID: MikroTik  Port ID: bridge110
44 65.790853740 Routerboardc_1c:8d:… LLDP_Multicast       LLDP 111 MA/c4:ad:34:1c:8d:2d IN/bridge110 120 SysN=MikroTik SysD=MikroTik RouterOS 6.43.16 (lon
45 66.043796863 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2d  Cost = 0  Port = 0x8001
46 66.091695982 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=6/1536, ttl=64 (no response found!)
47 67.115697199 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=7/1792, ttl=64 (no response found!)
48 68.045939976 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2d  Cost = 0  Port = 0x8001
49 68.139698975 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=8/2048, ttl=64 (no response found!)
50 69.163699005 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=9/2304, ttl=64 (no response found!)
51 70.048022747 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2d  Cost = 0  Port = 0x8001
52 70.187697918 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=10/2560, ttl=64 (no response found!)
53 71.211697250 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=11/2816, ttl=64 (no response found!)
54 72.040155379 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2d  Cost = 0  Port = 0x8001
55 72.235698678 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=12/3072, ttl=64 (no response found!)
56 73.259699826 172.16.110.1         172.16.110.255       ICMP  98 Echo (ping) request  id=0x06fd, seq=13/3328, ttl=64 (no response found!)
57 74.041807228 Routerboardc_1c:8d:… Spanning-tree-(for-… STP   60 RST. Root = 32768/0/c4:ad:34:1c:8d:2d  Cost = 0  Port = 0x8001
```

Tux64:

24	44.962362812	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=1/256, ttl=64 (no response found!)
25	45.992702006	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=2/512, ttl=64 (no response found!)
26	46.037062046	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
27	47.016683477	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=3/768, ttl=64 (no response found!)
28	48.038774275	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
29	48.040668860	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=4/1024, ttl=64 (no response found!)
30	49.064689512	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=5/1280, ttl=64 (no response found!)
31	49.787690950	0.0.0.0	255.255.255.255	MNDP	160	5678 → 5678 Len=118
32	49.787721680	Routerboardc_1c:8d:…	CDP/VTP/DTP/PAgP/UD…	CDP	94	Device ID: MikroTik Port ID: bridge110
33	49.787770150	Routerboardc_1c:8d:…	LLDP_Multicast	LLDP	111	MA/c4:ad:34:1c:8d:2d IN/bridge110 120 SysN=MikroTik SysD=MikroTik RouterOS
34	50.040760212	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
35	50.088705695	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=6/1536, ttl=64 (no response found!)
36	51.112723693	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=7/1792, ttl=64 (no response found!)
37	52.042917470	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
38	52.136719272	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=8/2048, ttl=64 (no response found!)
39	53.160734687	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=9/2304, ttl=64 (no response found!)
40	54.045006283	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
41	54.184724609	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=10/2560, ttl=64 (no response found!)
42	55.208741001	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=11/2816, ttl=64 (no response found!)
43	56.037148021	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
44	56.232740701	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=12/3072, ttl=64 (no response found!)
45	57.256744452	172.16.110.1	172.16.110.255	ICMP	98	Echo (ping) request id=0x06fd, seq=13/3328, ttl=64 (no response found!)
46	58.038808707	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002

4.2 Pinging broadcast from tux62

Command: *ping -b 172.16.111.255*

Tux62:

26	45.368400681	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=1/256, ttl=64 (no response found!)
27	46.008862740	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2c Cost = 0 Port = 0x8001
28	46.395685429	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=2/512, ttl=64 (no response found!)
29	47.419681956	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=3/768, ttl=64 (no response found!)
30	48.011008016	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2c Cost = 0 Port = 0x8001
31	48.443684908	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=4/1024, ttl=64 (no response found!)
32	49.467677873	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=5/1280, ttl=64 (no response found!)
33	50.013183814	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2c Cost = 0 Port = 0x8001
34	50.491689137	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=6/1536, ttl=64 (no response found!)
35	51.515674349	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=7/1792, ttl=64 (no response found!)
36	52.005303861	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2c Cost = 0 Port = 0x8001
37	52.509034290	0.0.0.0	255.255.255.255	MNDP	160	5678 → 5678 Len=118
38	52.509047071	Routerboardc_1c:8d:…	CDP/VTP/DTP/PAgP/UD…	CDP	94	Device ID: MikroTik Port ID: bridge111
39	52.509085973	Routerboardc_1c:8d:…	LLDP_Multicast	LLDP	111	MA/c4:ad:34:1c:8d:2c IN/bridge111 120 SysN=MikroTik SysD=MikroTik RouterOS 6.43.
40	52.539681562	172.16.111.1	172.16.111.255	ICMP	98	Echo (ping) request id=0x0787, seq=8/2048, ttl=64 (no response found!)
41	54.007544612	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2c Cost = 0 Port = 0x8001

Tux63:

24	40.021303545	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
25	42.023429162	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
26	44.025533966	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
27	46.027625080	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
28	48.029741687	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
29	50.031797882	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
30	52.033954997	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
31	54.036057147	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
32	56.038152942	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
33	58.030266960	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
34	60.032419187	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
35	62.034572392	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
36	64.036730416	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
37	66.038892141	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
38	68.041028305	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
39	70.043167402	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001
40	72.045318372	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8001

(Nothing happens)

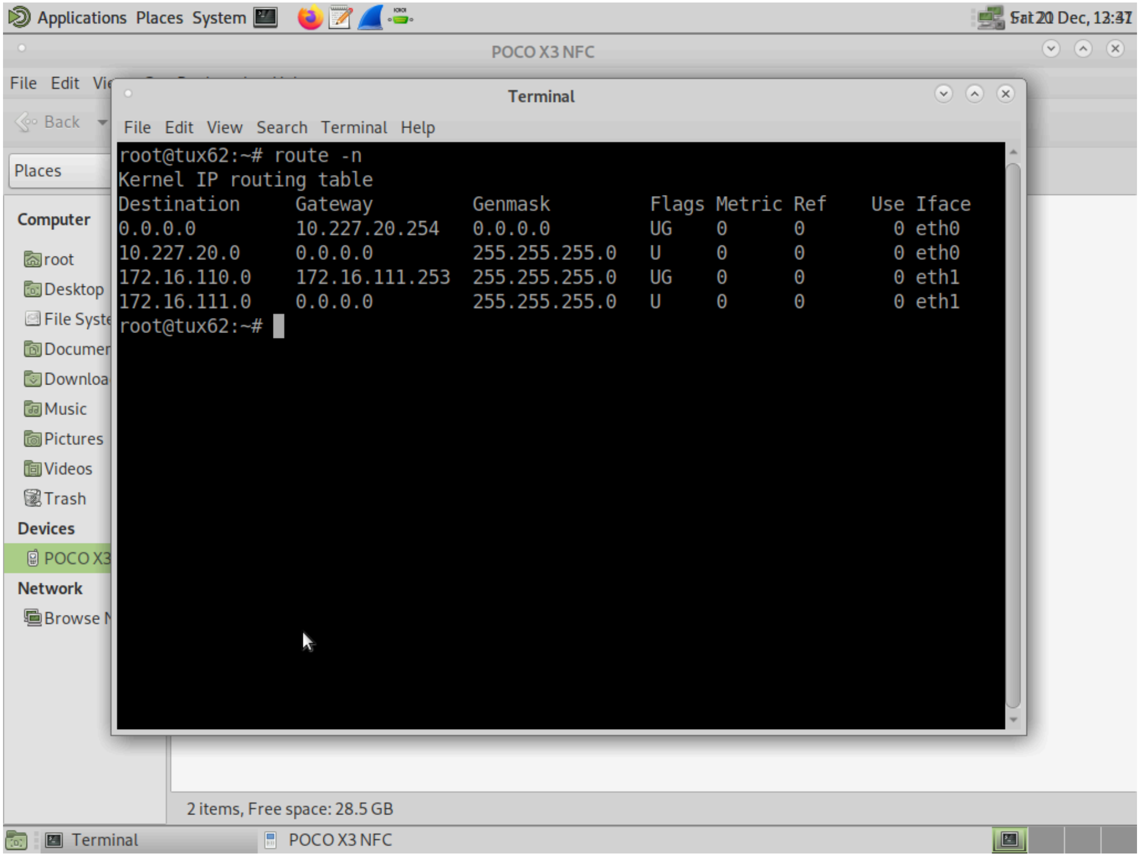
Tux64:

19	30.021185655	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
20	32.023313999	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
21	34.025428653	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
22	36.027525009	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
23	38.029652724	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
24	40.031714648	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
25	42.033878960	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
26	44.035991729	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
27	46.038097373	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
28	48.030217181	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
29	50.032375766	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
30	52.034540287	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
31	54.036704599	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
32	56.038875825	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
33	58.041019812	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
34	60.043163032	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002
35	62.045325807	Routerboardc_1c:8d:…	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2d Cost = 0 Port = 0x8002

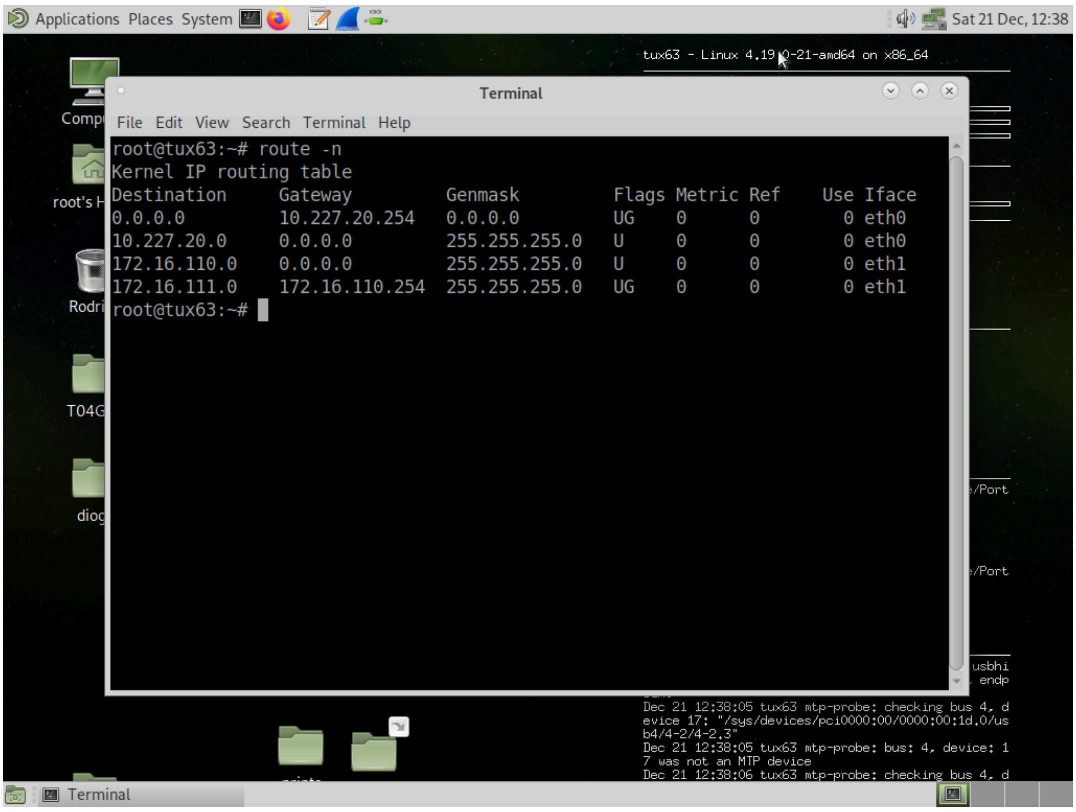
(Nothing happens)

5. Exp 3 routes

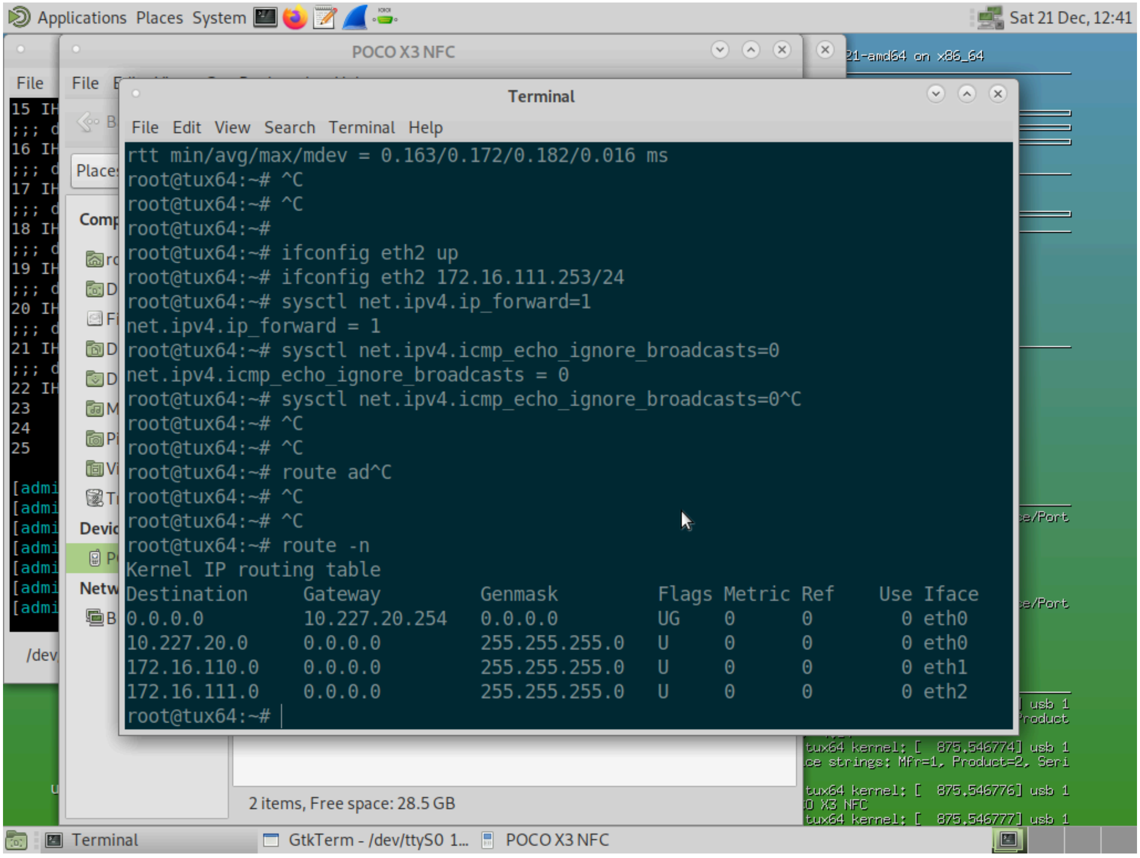
tux62:


Applications Places System
Sat 20 Dec, 13:37
POCO X3 NFC
File Edit
Back
Places
Computer
root
Desktop
Devices
Network
Terminal
File Edit View Search Terminal Help
root@tux62:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 10.227.20.254 0.0.0.0 UG 0 0 0 eth0
10.227.20.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
172.16.110.0 172.16.111.253 255.255.255.0 UG 0 0 0 eth1
172.16.111.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
root@tux62:~#
2 items, Free space: 28.5 GB
Terminal
POCO X3 NFC

tux63:


Applications Places System
Sat 21 Dec, 12:38
Terminal
File Edit View Search Terminal Help
root@tux63:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 10.227.20.254 0.0.0.0 UG 0 0 0 eth0
10.227.20.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
172.16.110.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
172.16.111.0 172.16.110.254 255.255.255.0 UG 0 0 0 eth1
root@tux63:~#
Terminal

tux64:



6. Configuration commands and Logs

All relevant commands and logs have been inserted where appropriate and relevant to the context. To access the file of a Wireshark log, please send an email to up202204988@up.pt.