Computer Networks

Redes de Computadores 2024-2025

2st Lab Work

3LEIC09

Gabriel da Quinta Braga (up202207784)

Guilherme Silveira Rego (up202207041)

December 23, 2024

Summary

This report details the development and testing of an FTP download application and a series of network configuration experiments. The report includes the architecture and implementation of the application in C, network setup procedures, logs of experimental outcomes, and an analysis of results.

Introduction

This document outlines two primary objectives: creating and testing a functional FTP download application and configuring network systems to simulate and analyze various scenarios, according to the guide’s specifications. The project explores network communication, routing, and FTP protocol implementation to transfer a file from the internet using the networking we configured, supplemented by hands-on experiments and data analysis.

The report is divided in three parts:

- **FTP download application**: Architecture and report of a successful download.
- **Network configuration and analysis**: Network architecture, experiment objectives, main configuration commands, relevant logs and its analysis.
- **Conclusions**: Answering important questions about the network configuration in order to better understand it and the conclusions we got.

FTP download application

The developed application facilitates the download of a file using the FTP protocol, as specified in RFC959 and RFC1738. The program's logic is divided into six main components:

1. Initialization

This phase initializes variables (command buffer and response buffer which, respectively, are used to write commands to the socket and read its response). It also includes parsing the URL input provided by the user and extracting critical information such as the host's IP address, file path, and credentials (username and password) and storing it in a URL struct we created (see the annex for more information).

The parse of the URL involves:

- Checking the format of the URL to ensure it complies with:

  ```
  ftp://[<user>:<password>@]<host>/<url-path>
  ```

- Extract optional user credentials (if they are not in the URL, the default is user=”anonymous” and password=”” for public addresses, user=”rcom” and password=”rcom” otherwise), host information, and file path components.

Then we resolve the host to its corresponding IP address using DNS functions in getip() and finally extract the filename from the URL path using parse_filename().

Example usage of the program:
./download <URL>
The input URL is parsed and stored in a custom `URL` structure. This structure includes fields for the username, password, host, path, filename, and resolved IP address, which are filled during the parsing process.

2. Create the Socket

The application establishes a primary control connection to the FTP server by creating a TCP socket using our function create_socket().

This involves:

- Using the IP address resolved earlier.
- Creating a sockaddr_in struct and configure it to set the properties of the connection, we set sin_family to AF_NET (IPv4), its IP to the IP we got before using the inet_addr() function to convert the IP to string format, and finally the sin_port to the port we passed through create_socket() as an argument, using htons() function to convert the format to big-endian to ensure consistency.
- Creating the Socket using the socket() function with AF_NET, SOCK_STREAM.
- Using connect() function to connect the socket to the IP address and port set on the sockaddr_in struct we created before.
- Validating the server's response to ensure the connection was successful

With the socket created, we need to read the server's response to the creation of the socket using the read_socket() function we created, and if it was successful we should get the following answer:

```
"220 If you don't have an account, log in as <user> or <other user>."
```

Which, according to RFC959, is the message that indicates that the server is waiting for authentication, meaning that the creation of the socket was successful.

3. Authenticate

With the socket created, we need to authenticate to the server, and to accomplish that we created the function authenticate() that has three arguments, the URL struct we created before, a pointer to the command buffer that we will fill, a pointer to the response buffer, and the file descriptor of the socket we created.

This function aims to authenticate the client into the FTP server and validate its response to ensure that the login was successful, and it is done with the following steps:

- Construct the USER command that is used in FTP to inform the user to the server. To do this we use the sprintf() function that concatenates "USER" with the user in the URL with the correct format (ends with \r\n), and stores it in the command buffer. The usage of this function to have the commands in the correct string format will be used from now on.
- Use the function write_command() that sends the command buffer with the USER command through the socket to the FTP server.
- Use bzero() to clear the command and response buffers, and then construct the PASS command the same way it was made with USER. We will use this function from now on every time we use these buffers.
- Again, use the function write_command() to send the PASS command.

If everything goes as it is supposed, the client is logged in with the given user and password, which we can confirm by reading the socket and getting the following server's response:

```
“Response: 230 User <user> logged in.”
```

Which is the message indicating a successful authentication on the server, stated in the RFC959.

4. Passive mode

Putting the client in passive mode is a critical step to establish a connection with the FTP server. This mode enables the client to control the connection **opening a new socket** to the file transfer. We created a function **passive_mode()** to ensure this is done correctly, obtaining the IP address and the port of the FTP server from the server's response to the PASV command. The function follows the steps:

- Get the passive mode command the same way it was made before.
- Call the **passive_mode()** function with the first socket we created, the second socket we want to create, the buffer with the passive mode command, the length of this command, and the buffer with the response from the server.
- Write the passive mode command to the server through the first socket, requesting the passive mode. The server's response will include the IP address and the Port to establish connection.
- Initialize the variables to store the IP (ip1, ip2, ip3 and ip4) and the Port (port1 and port2).
- Use sscanf() to extract the individual values of the IP and the Port and write it to the variables we initialized. We will use this function from now on every time we need to extract values and write them to variables.
- Use sprintf() to create the string corresponding to the IP and save it in another variable, that is the result of every ip combined.
- Calculate the Port's number and store it in another variable, port_num, that is the final result. The Port's number is divided in two values (port1 and port2) that were given by the server, to calculate it we apply the following formula: port1 * 256 + port2, that combines the two values in one number of 16 bits. The Port's number is necessary to create the Socket in passive mode.
- Create a second Socket (sockfdB) in passive mode, the same way we created the first one (sockfdA), with create_socket() function with the ip and port_num we calculated before.

At the end, if everything went well, we should get the following response from the server:

```
“227 Entering Passive Mode (<ip1>,<ip2>,<ip3>,<ip4>,<port1>,<port2>)”
```

Which, according to RFC959, indicates that sockfdB entered passive mode.

5. Retrieve File

Retrieving a file from the FTP server is the core objective of the program, as it initiates the process of downloading the specified file from the server to the client. The **retrieve_file()** function plays a key role in sending the appropriate FTP command (RETR) to the server, which triggers the transfer of the file's contents via the data connection established earlier in passive mode (sockfdB). This function ensures that the retrieval process is correctly requested and handled. Below, we outline the steps involved in this process:

- Extract the file path from the URL provided by the user during the initialization phase.
- Use sprintf() to create the RETR command by appending the file path to the FTP command, formatted as: RETR <filepath>\r\n.

- Call the **retrieve_file()** function, passing as arguments sockfdA, the file path, the command buffer and the response buffer.
- Call the **retr()** function to send the RETR command through the control socket (sockfdA), previously established with the server.
- The function uses the write() system call to send the command, ensuring that the server is informed of the desired file to retrieve.
- Check the return value of the **retr()** function. If the command fails to send (e.g., due to a socket error or invalid file path), an error message is displayed, and the process is terminated.
- The server processes the RETR command and begins sending the file's contents through the passive mode socket (sockfdB).

If executed successfully, this process informs the server to begin streaming the file data over the pre-established data connection.

6. Download File

The **download_file()** function is the final step in the FTP client's workflow, as it handles the actual transfer of the requested file's data from the server to the client's local storage. Once the RETR command has been successfully issued and the server begins transmitting file data through the passive mode socket (sockfdB), this function ensures that the data is received, written to a local file, and properly managed. Below is a detailed breakdown of how this process works:

- A new file is created (or an existing file is overwritten) using the open() system call with the O_WRONLY | O_CREAT flags.
- File permissions are set to 0666 to ensure the file is readable and writable by the user and group.
- The function enters a loop to read chunks of data from the passive mode socket (sockfdB) using the read() system call.
- A buffer is used to store each chunk of data temporarily.
- If the read() system call fails, the function logs an error message and terminates. This ensures that incomplete or corrupted file transfers are not silently ignored.
- Each chunk of data read from the socket is written to the local file (where is the final result of the transfer) using the write() system call.
- After all the data has been read and written, the function closes both the local file descriptor and the passive mode socket (sockfdB) to release resources and clean up.

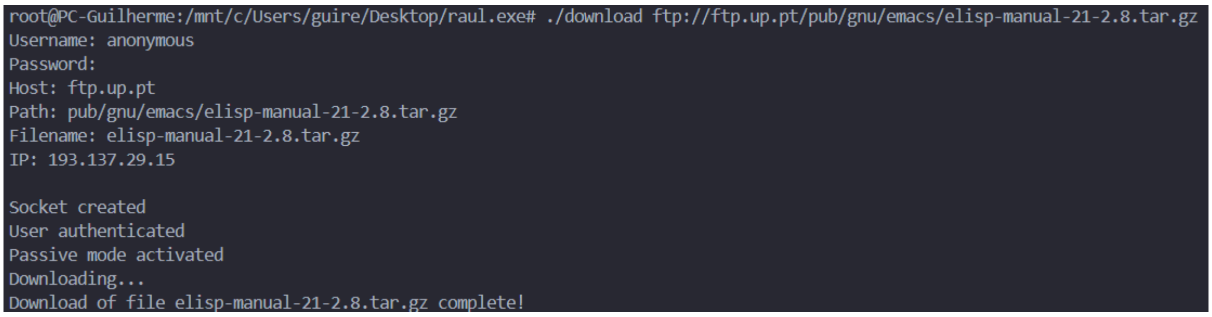
By ensuring robust error handling, resource cleanup, and proper file operations, the **download_file()** function guarantees a reliable and efficient file download process.

It is important to highlight that our application has a **robust error handling** (e.g., checks if every pointer is NULL and the return values of system calls), resource cleanup (e.g., the use of the bzero() function to cleanup the buffers), as we can see throughout the whole C code, ensuring a reliable and efficient file download process following RFC959 and RFC1738.

Results

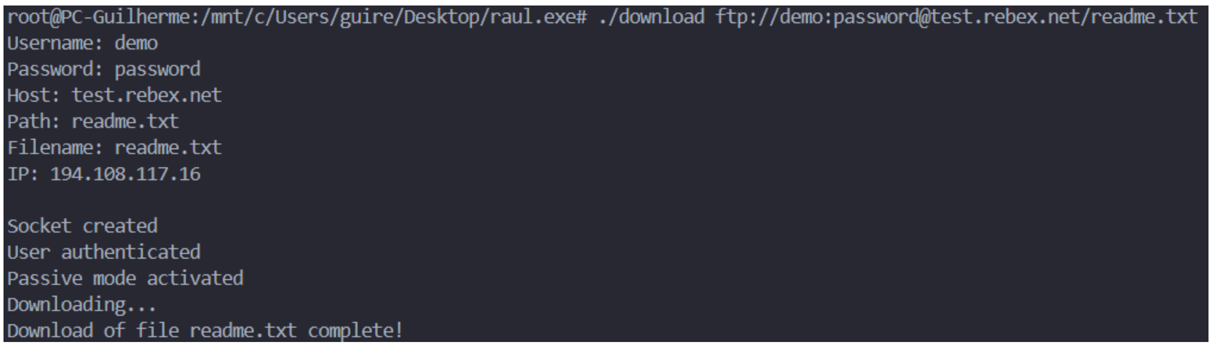
In order to test our download application, we tried to download different files with different lengths, and wrong file names to ensure it would work in the same way. Below, we will show the results of the download of different files:

URL 1:



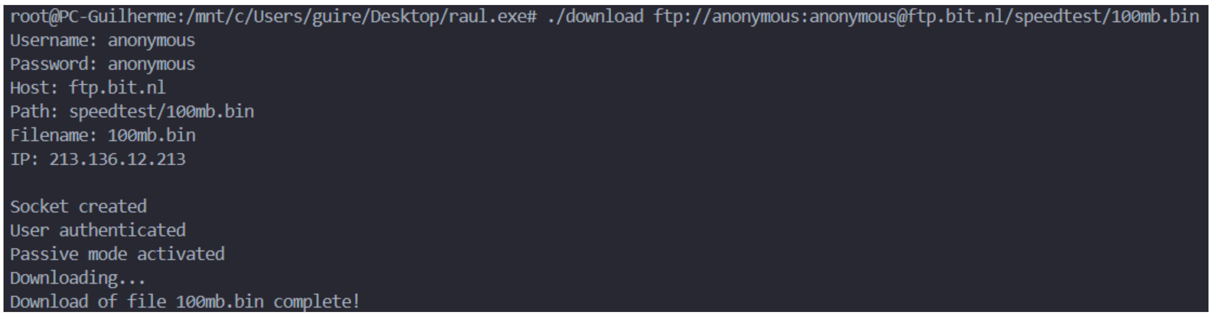
```
root@PC-Guilherme:/mnt/c/Users/guire/Desktop/raul.exe# ./download ftp://ftp.up.pt/pub/gnu/emacs/elisp-manual-21-2.8.tar.gz
Username: anonymous
Password:
Host: ftp.up.pt
Path: pub/gnu/emacs/elisp-manual-21-2.8.tar.gz
Filename: elisp-manual-21-2.8.tar.gz
IP: 193.137.29.15

Socket created
User authenticated
Passive mode activated
Downloading...
Download of file elisp-manual-21-2.8.tar.gz complete!
```

URL 2:



```
root@PC-Guilherme:/mnt/c/Users/guire/Desktop/raul.exe# ./download ftp://demo:password@test.rebex.net/readme.txt
Username: demo
Password: password
Host: test.rebex.net
Path: readme.txt
Filename: readme.txt
IP: 194.108.117.16

Socket created
User authenticated
Passive mode activated
Downloading...
Download of file readme.txt complete!
```

URL 3:



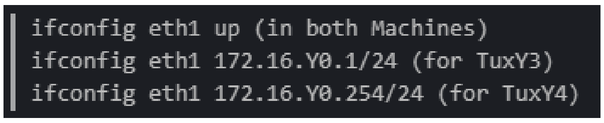
```
root@PC-Guilherme:/mnt/c/Users/guire/Desktop/raul.exe# ./download ftp://anonymous:anonymous@ftp.bit.nl/speedtest/100mb.bin
Username: anonymous
Password: anonymous
Host: ftp.bit.nl
Path: speedtest/100mb.bin
Filename: 100mb.bin
IP: 213.136.12.213

Socket created
User authenticated
Passive mode activated
Downloading...
Download of file 100mb.bin complete!
```

Configuration and Analysis of Computer Networks - Class Experiments

Experiment 1 - Configure an IP Network

We initiate every class by running the command 'systemctl restart networking' on each one of the TUXs. This experience's goal is to configure the IP addresses of two different computers, connected to the switch, and to study their interaction.

We start by connecting the E1 port of both TUXY3 and TUXY4 to any port on the switch (the ports to which the machines are connected may vary throughout the experiments since not always were the same computers available). After this, we configure the eth1 interface of TUXY3 and TUXY4 using ifconfig in the console.



```
ifconfig eth1 up (in both Machines)
ifconfig eth1 172.16.Y0.1/24 (for TuxY3)
ifconfig eth1 172.16.Y0.254/24 (for TuxY4)
```

With the both eth1 interfaces configured, we now ping TUXY4 from TUXY3, while capturing the results by running Wireshark in TUXY3's eth1.

Experiment 2 - Implement two bridges in a switch

In this experiment, the objective was to create two bridges on a switch: one connecting TUXY3 and TUXY4, and the other exclusively connecting TUXY2.

We began by configuring TUXY2 using the following commands:

- ifconfig eth1 up
- ifconfig eth1 172.16.Y1.1/24

Next, we modified the physical connections as follows:

- Connect the E1 port of TUXY2 to ether2.
- Connect the E2 port of TUXY3 to ether3.
- Connect the E3 port of TUXY4 to ether4.

Using GTKTerm, we accessed the switch console with the following settings:

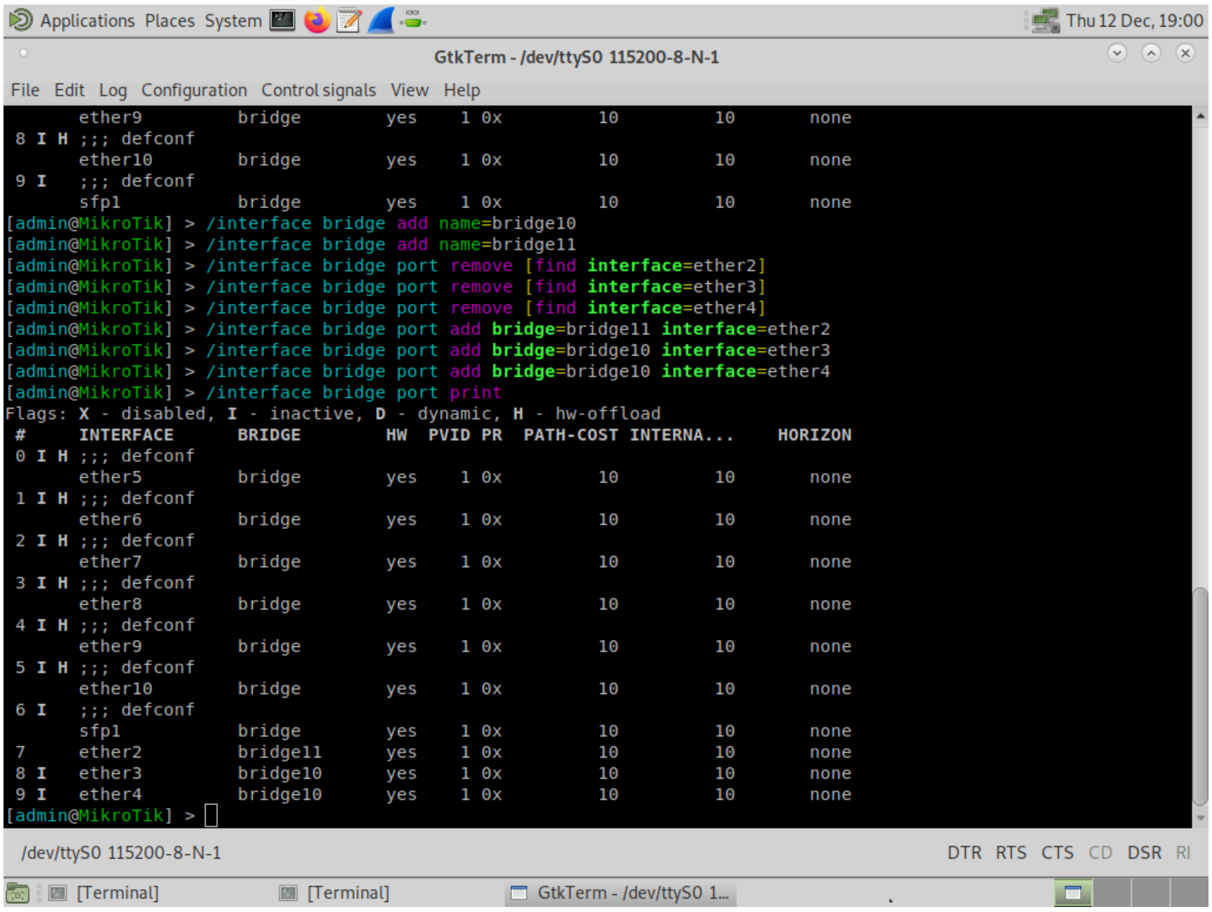
- **Baudrate:** 115200 bps
- **Username:** admin
- **Password:** (leave blank)

Once logged in, we removed the default port configurations for the connected interfaces using the commands:

- /interface bridge port remove [find interface=ether2]
- /interface bridge port remove [find interface=ether3]
- /interface bridge port remove [find interface=ether4]

We created two bridges and assigned the ports as follows:

- /interface bridge add name=bridgeY0
- /interface bridge add name=bridgeY1
- /interface bridge port add bridge=bridgeY0 interface=ether3
- /interface bridge port add bridge=bridgeY0 interface=ether4
- /interface bridge port add bridge=bridgeY1 interface=ether2


Applications Places System
Thu 12 Dec, 19:00
GtkTerm - /dev/ttyS0 115200-8-N-1
File Edit Log Configuration Control signals View Help
/dev/ttyS0 115200-8-N-1
DTR RTS CTS CD DSR RI
[Terminal]
[Terminal]
GtkTerm - /dev/ttyS0 1...

```
       ether9         bridge        yes    1 0x          10          10        none
 8 I H ;;; defconf
       ether10        bridge        yes    1 0x          10          10        none
 9 I   ;;; defconf
       sfp1           bridge        yes    1 0x          10          10        none
[admin@MikroTik] > /interface bridge add name=bridge10
[admin@MikroTik] > /interface bridge add name=bridge11
[admin@MikroTik] > /interface bridge port remove [find interface=ether2]
[admin@MikroTik] > /interface bridge port remove [find interface=ether3]
[admin@MikroTik] > /interface bridge port remove [find interface=ether4]
[admin@MikroTik] > /interface bridge port add bridge=bridge11 interface=ether2
[admin@MikroTik] > /interface bridge port add bridge=bridge10 interface=ether3
[admin@MikroTik] > /interface bridge port add bridge=bridge10 interface=ether4
[admin@MikroTik] > /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE      BRIDGE        HW  PVID PR  PATH-COST INTERNA...    HORIZON
 0 I H ;;; defconf
       ether5         bridge        yes    1 0x          10          10        none
 1 I H ;;; defconf
       ether6         bridge        yes    1 0x          10          10        none
 2 I H ;;; defconf
       ether7         bridge        yes    1 0x          10          10        none
 3 I H ;;; defconf
       ether8         bridge        yes    1 0x          10          10        none
 4 I H ;;; defconf
       ether9         bridge        yes    1 0x          10          10        none
 5 I H ;;; defconf
       ether10        bridge        yes    1 0x          10          10        none
 6 I   ;;; defconf
       sfp1           bridge        yes    1 0x          10          10        none
 7     ether2         bridge11      yes    1 0x          10          10        none
 8 I   ether3         bridge10      yes    1 0x          10          10        none
 9 I   ether4         bridge10      yes    1 0x          10          10        none
[admin@MikroTik] > 
```

With this setup, we tested the bridges using broadcasts. When pinging from TUXY3, we observed that TUXY4 was reachable, as expected, because both devices were part of the same bridge (bridgeY0). However, TUXY2 was not reachable, as it was isolated within bridgeY1.

Similarly, when testing from TUXY2, it could not communicate with any other machine, confirming that the bridges were configured correctly to isolate the networks.

Doing the same in TUXY2, we can confirm that it does not reach any other machine, as we said before.

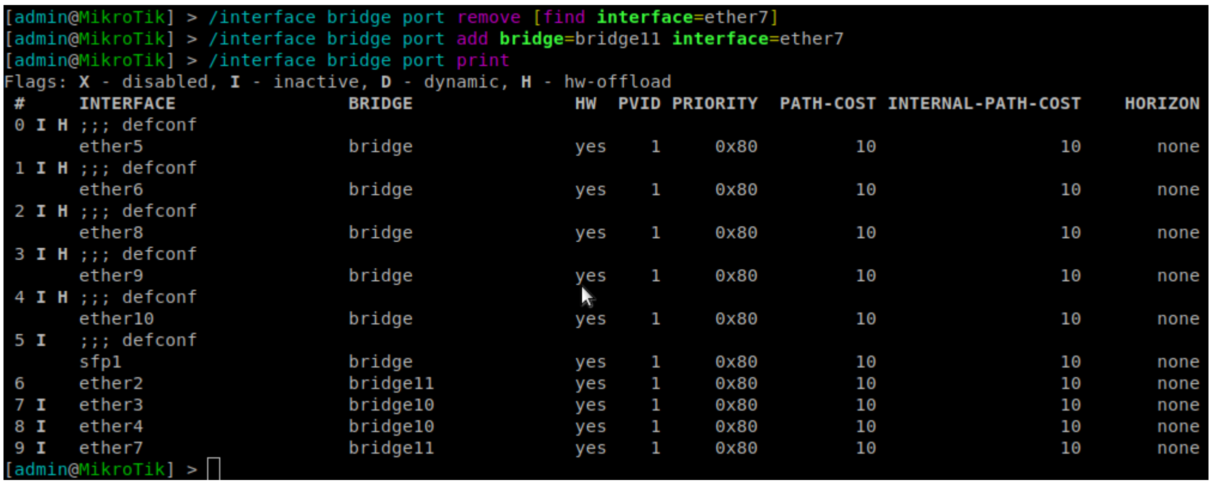
Experiment 3 - Configuring a Router in Linux

With the network resulting from the previous experiment, this experience has the objective to to create a Router (using TUX4) to connect the IPs in the address space 172.16.Y0.0 and 172.16.Y1.0, making it possible for TUXY2 and TUXY3 to reach eachother.

We start by connecting the E1 port of TUXY4 to a port in the switch and configure its IP using ifconfig, in the following manner:

```
ifconfig eth2 up
ifconfig eth2 172.16.Y1.253/24
```

We then remove any of the ports to which the switch might have been connected so that there were no possible errors, and we add the interface corresponding to the E1 port of TUXY4 to the bridgeY1.



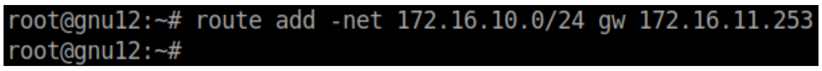
```
[admin@MikroTik] > /interface bridge port remove [find interface=ether7]
[admin@MikroTik] > /interface bridge port add bridge=bridge11 interface=ether7
[admin@MikroTik] > /interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
 #     INTERFACE                 BRIDGE                HW  PVID PRIORITY  PATH-COST INTERNAL-PATH-COST    HORIZON
 0 I H ;;; defconf
       ether5                    bridge                yes    1     0x80         10                 10       none
 1 I H ;;; defconf
       ether6                    bridge                yes    1     0x80         10                 10       none
 2 I H ;;; defconf
       ether8                    bridge                yes    1     0x80         10                 10       none
 3 I H ;;; defconf
       ether9                    bridge                yes    1     0x80         10                 10       none
 4 I H ;;; defconf
       ether10                   bridge                yes    1     0x80         10                 10       none
 5 I   ;;; defconf
       sfp1                      bridge                yes    1     0x80         10                 10       none
 6     ether2                    bridge11              yes    1     0x80         10                 10       none
 7 I   ether3                    bridge10              yes    1     0x80         10                 10       none
 8 I   ether4                    bridge10              yes    1     0x80         10                 10       none
 9 I   ether7                    bridge11              yes    1     0x80         10                 10       none
[admin@MikroTik] >
```

Following the guide, we enable ‘*IP Forwarding*’ and disable ‘*ICMP Echo Ignore Broadcasts*’

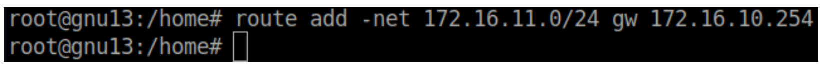
```
sysctl net.ipv4.ip_forward=1
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
```

Now, to make sure TUXY3 and TUXY2 can reach each other, we reconfigure them and create routes to make sure they can communicate with the IP Address space of one another using TUXY4 as a gateway:

```
route add -net 172.16.Y0.0/24 gw 172.16.Y1.253 (In TUX2)
route add -net 172.16.Y1.0/24 gw 172.16.Y0.254 (In TUX3)
```

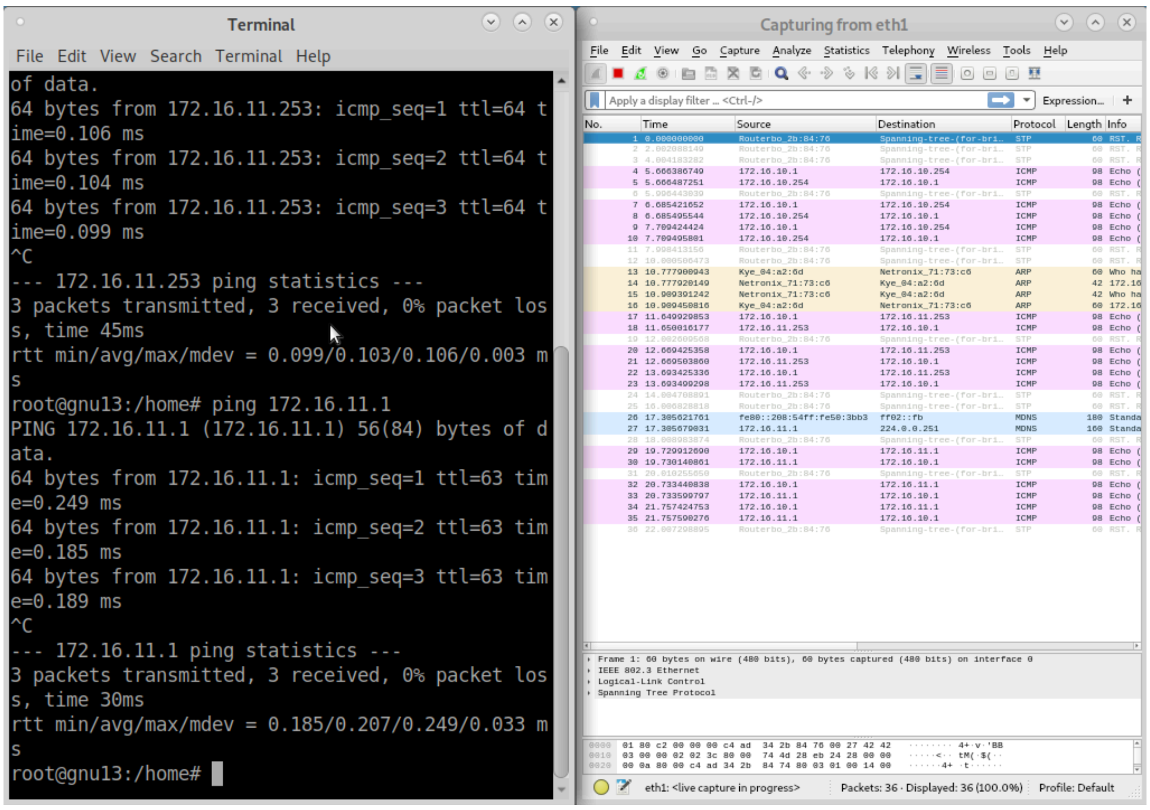


```
root@gnu12:~# route add -net 172.16.10.0/24 gw 172.16.11.253
root@gnu12:~#
```



```
root@gnu13:/home# route add -net 172.16.11.0/24 gw 172.16.10.254
root@gnu13:/home#
```

From TUXY3, we now ping TUXY4's eth1 and eth2, and TUXY2 to verify if everything is well set up:

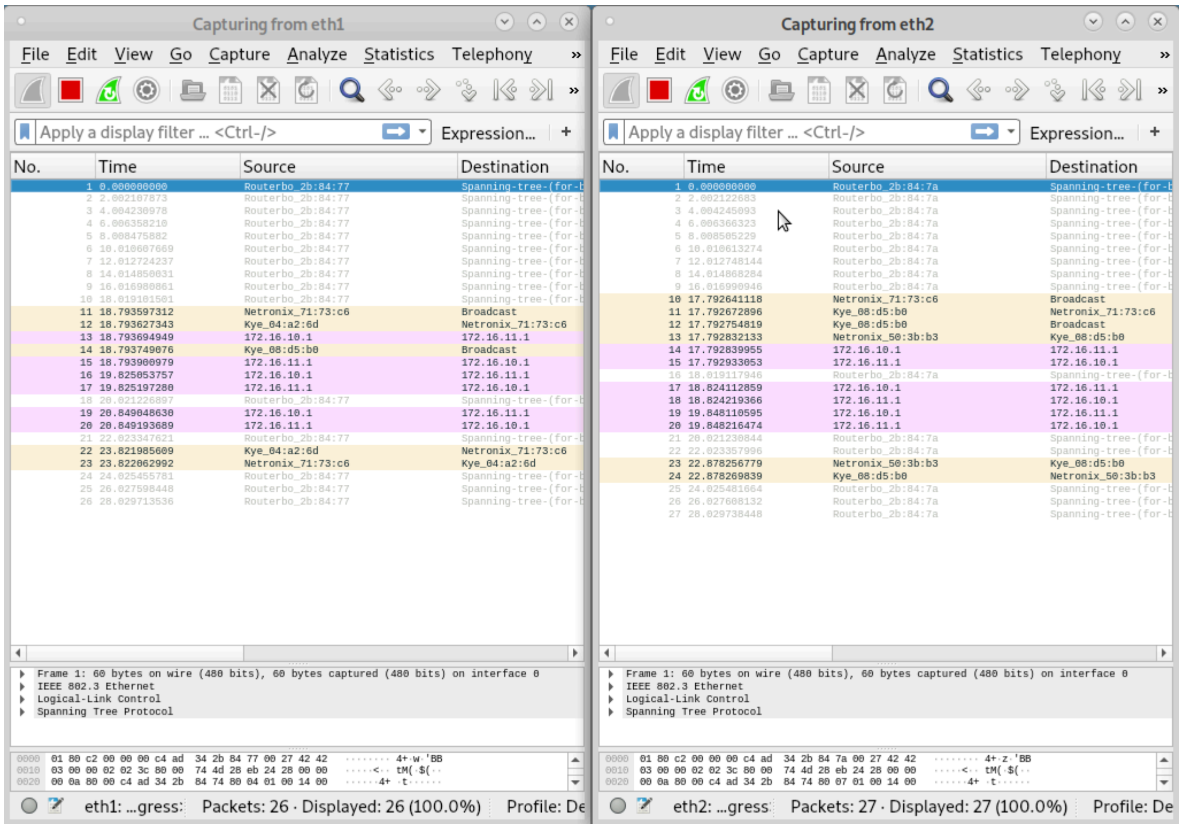


```
of data.
64 bytes from 172.16.11.253: icmp_seq=1 ttl=64 time=0.106 ms
64 bytes from 172.16.11.253: icmp_seq=2 ttl=64 time=0.104 ms
64 bytes from 172.16.11.253: icmp_seq=3 ttl=64 time=0.099 ms
^C
--- 172.16.11.253 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 45ms
rtt min/avg/max/mdev = 0.099/0.103/0.106/0.003 ms
root@gnu13:/home# ping 172.16.11.1
PING 172.16.11.1 (172.16.11.1) 56(84) bytes of data.
64 bytes from 172.16.11.1: icmp_seq=1 ttl=63 time=0.249 ms
64 bytes from 172.16.11.1: icmp_seq=2 ttl=63 time=0.185 ms
64 bytes from 172.16.11.1: icmp_seq=3 ttl=63 time=0.189 ms
^C
--- 172.16.11.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 30ms
rtt min/avg/max/mdev = 0.185/0.207/0.249/0.033 ms
root@gnu13:/home#
```

```
No.  Time         Source                    Destination                Protocol Length Info
  1  0.000000000  Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
  2  2.002088149  Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
  3  4.004183282  Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
  4  5.666386749  172.16.10.1               172.16.10.254              ICMP     98     Echo (
  5  5.666487251  172.16.10.254             172.16.10.1                ICMP     98     Echo (
  6  5.996443039  Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
  7  6.685421652  172.16.10.1               172.16.10.254              ICMP     98     Echo (
  8  6.685495544  172.16.10.254             172.16.10.1                ICMP     98     Echo (
  9  7.709424424  172.16.10.1               172.16.10.254              ICMP     98     Echo (
 10  7.709495881  172.16.10.254             172.16.10.1                ICMP     98     Echo (
 11  7.998413156  Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
 12  10.000506473 Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
 13  10.777900943 Kye_04:a2:6d              Netronix_71:73:c6          ARP      60     Who ha
 14  10.777920149 Netronix_71:73:c6         Kye_04:a2:6d               ARP      42     172.16
 15  10.989391242 Netronix_71:73:c6         Kye_04:a2:6d               ARP      42     Who ha
 16  10.989450816 Kye_04:a2:6d              Netronix_71:73:c6          ARP      60     172.16
 17  11.649929853 172.16.10.1               172.16.11.253              ICMP     98     Echo (
 18  11.650016177 172.16.11.253             172.16.10.1                ICMP     98     Echo (
 19  12.002609568 Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
 20  12.669425358 172.16.10.1               172.16.11.253              ICMP     98     Echo (
 21  12.669503860 172.16.11.253             172.16.10.1                ICMP     98     Echo (
 22  13.693425336 172.16.10.1               172.16.11.253              ICMP     98     Echo (
 23  13.693499298 172.16.11.253             172.16.10.1                ICMP     98     Echo (
 24  14.004708891 Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
 25  16.006828818 Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
 26  17.305621761 fe80::208:54ff:fe50:3bb3  ff02::fb                   MDNS     180    Standa
 27  17.305679031 172.16.11.1               224.0.0.251                MDNS     160    Standa
 28  18.008983874 Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
 29  19.729912690 172.16.10.1               172.16.11.1                ICMP     98     Echo (
 30  19.730140861 172.16.11.1               172.16.10.1                ICMP     98     Echo (
 31  20.010255650 Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R
 32  20.733440838 172.16.10.1               172.16.11.1                ICMP     98     Echo (
 33  20.733509797 172.16.11.1               172.16.10.1                ICMP     98     Echo (
 34  21.757424753 172.16.10.1               172.16.11.1                ICMP     98     Echo (
 35  21.757590276 172.16.11.1               172.16.10.1                ICMP     98     Echo (
 36  22.007298895 Routerbo_2b:84:76         Spanning-tree-(for-bri…    STP      60     RST. R

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Spanning Tree Protocol

0000  01 80 c2 00 00 00 c4 ad  34 2b 84 76 00 27 42 42   ········ 4+·v·'BB
0010  03 00 00 02 02 3c 80 00  74 4d 28 eb 24 28 00 00   ·····<·· tM(·$(··
0020  00 0a 80 00 c4 ad 34 2b  84 74 80 03 01 00 14 00   ······4+ ·t······

eth1: <live capture in progress>   Packets: 36 · Displayed: 36 (100.0%)   Profile: Default
```

Terminal
File Edit View Search Terminal Help
Capturing from eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

We can make sure everything is working as intended, by starting a Wireshark capture of eth1 and eth2 on TUXY4 and pinging TUXY2 from TUXY3 and checking the results:



```
No.  Time          Source               Destination
  1  0.000000000   Routerbo_2b:84:77    Spanning-tree-(for-b
  2  2.002107873   Routerbo_2b:84:77    Spanning-tree-(for-b
  3  4.004230978   Routerbo_2b:84:77    Spanning-tree-(for-b
  4  6.006358210   Routerbo_2b:84:77    Spanning-tree-(for-b
  5  8.008475882   Routerbo_2b:84:77    Spanning-tree-(for-b
  6  10.010607669  Routerbo_2b:84:77    Spanning-tree-(for-b
  7  12.012724237  Routerbo_2b:84:77    Spanning-tree-(for-b
  8  14.014850031  Routerbo_2b:84:77    Spanning-tree-(for-b
  9  16.016980861  Routerbo_2b:84:77    Spanning-tree-(for-b
 10  18.019101501  Routerbo_2b:84:77    Spanning-tree-(for-b
 11  18.793597312  Netronix_71:73:c6    Broadcast
 12  18.793627343  Kye_04:a2:6d         Netronix_71:73:c6
 13  18.793694949  172.16.10.1          172.16.11.1
 14  18.793749076  Kye_08:d5:b0         Broadcast
 15  18.793900979  172.16.11.1          172.16.10.1
 16  19.825053757  172.16.10.1          172.16.11.1
 17  19.825197280  172.16.11.1          172.16.10.1
 18  20.021226897  Routerbo_2b:84:77    Spanning-tree-(for-b
 19  20.849048630  172.16.10.1          172.16.11.1
 20  20.849193689  172.16.11.1          172.16.10.1
 21  22.023347621  Routerbo_2b:84:77    Spanning-tree-(for-b
 22  23.821985609  Kye_04:a2:6d         Netronix_71:73:c6
 23  23.822062992  Netronix_71:73:c6    Kye_04:a2:6d
 24  24.025455781  Routerbo_2b:84:77    Spanning-tree-(for-b
 25  26.027598448  Routerbo_2b:84:77    Spanning-tree-(for-b
 26  28.029713536  Routerbo_2b:84:77    Spanning-tree-(for-b

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Spanning Tree Protocol

0000  01 80 c2 00 00 00 c4 ad  34 2b 84 77 00 27 42 42   ········ 4+·w·'BB
0010  03 00 00 02 02 3c 80 00  74 4d 28 eb 24 28 00 00   ·····<·· tM(·$(··
0020  00 0a 80 00 c4 ad 34 2b  84 74 80 04 01 00 14 00   ······4+ ·t······

eth1: ...gress   Packets: 26 · Displayed: 26 (100.0%)   Profile: De
```

```
No.  Time          Source               Destination
  1  0.000000000   Routerbo_2b:84:7a    Spanning-tree-(for-b
  2  2.002122683   Routerbo_2b:84:7a    Spanning-tree-(for-b
  3  4.004245093   Routerbo_2b:84:7a    Spanning-tree-(for-b
  4  6.006366323   Routerbo_2b:84:7a    Spanning-tree-(for-b
  5  8.008505229   Routerbo_2b:84:7a    Spanning-tree-(for-b
  6  10.010613274  Routerbo_2b:84:7a    Spanning-tree-(for-b
  7  12.012748144  Routerbo_2b:84:7a    Spanning-tree-(for-b
  8  14.014868284  Routerbo_2b:84:7a    Spanning-tree-(for-b
  9  16.016990946  Routerbo_2b:84:7a    Spanning-tree-(for-b
 10  17.792641118  Netronix_71:73:c6    Broadcast
 11  17.792672896  Kye_08:d5:b0         Netronix_71:73:c6
 12  17.792754819  Kye_08:d5:b0         Broadcast
 13  17.792832133  Netronix_50:3b:b3    Kye_08:d5:b0
 14  17.792839955  172.16.10.1          172.16.11.1
 15  17.792933053  172.16.11.1          172.16.10.1
 16  18.019117946  Routerbo_2b:84:7a    Spanning-tree-(for-b
 17  18.824112859  172.16.10.1          172.16.11.1
 18  18.824219366  172.16.11.1          172.16.10.1
 19  19.848110595  172.16.10.1          172.16.11.1
 20  19.848216474  172.16.11.1          172.16.10.1
 21  20.021230844  Routerbo_2b:84:7a    Spanning-tree-(for-b
 22  22.023357996  Routerbo_2b:84:7a    Spanning-tree-(for-b
 23  22.878256779  Netronix_50:3b:b3    Kye_08:d5:b0
 24  22.878269839  Kye_08:d5:b0         Netronix_50:3b:b3
 25  24.025481664  Routerbo_2b:84:7a    Spanning-tree-(for-b
 26  26.027608132  Routerbo_2b:84:7a    Spanning-tree-(for-b
 27  28.029738448  Routerbo_2b:84:7a    Spanning-tree-(for-b

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Spanning Tree Protocol

0000  01 80 c2 00 00 00 c4 ad  34 2b 84 7a 00 27 42 42   ········ 4+·z·'BB
0010  03 00 00 02 02 3c 80 00  74 4d 28 eb 24 28 00 00   ·····<·· tM(·$(··
0020  00 0a 80 00 c4 ad 34 2b  84 74 80 07 01 00 14 00   ······4+ ·t······

eth2: ...gress   Packets: 27 · Displayed: 27 (100.0%)   Profile: De
```

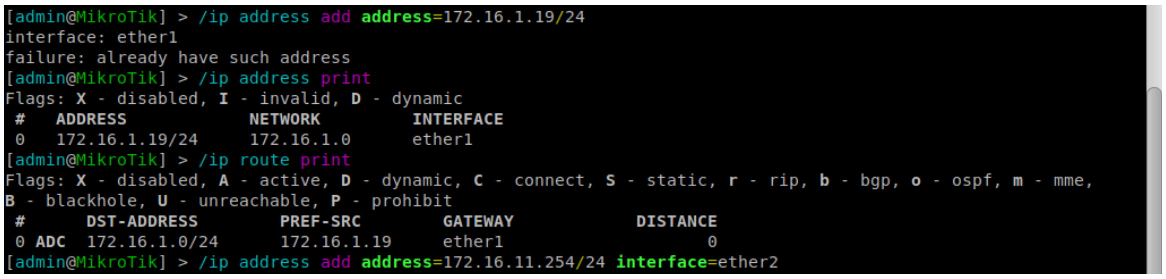
Capturing from eth1
Capturing from eth2
File Edit View Go Capture Analyze Statistics Telephony
Apply a display filter ... <Ctrl-/>
Expression...

Experiment 4 - Configuring a Commercial Router and Implementing NAT

In this experiment, we make use of the network set up in the last experiment, with the goal of configuring and adding a Commercial Router to BridgeY1 and Implementing NAT, in a way where the TUXs can now access the internet.

We start by connecting the eth1 port of the Router to the lab network on the PY.12, and the eth2 port of the Router to a port on the switch, who's interface we will add to BridgeY1. We now configure the router:

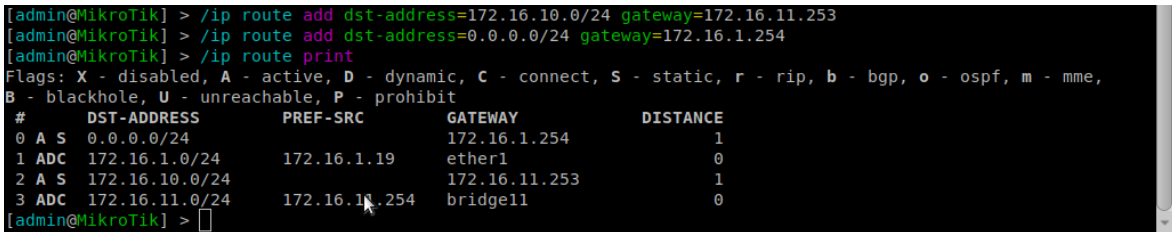
```
/interface bridge port remove [find interface=ether6]
/interface bridge port add bridge=bridgeY1 interface=ether6
```



```
[admin@MikroTik] > /ip address add address=172.16.1.19/24
interface: ether1
failure: already have such address
[admin@MikroTik] > /ip address print
Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         INTERFACE
 0   172.16.1.19/24     172.16.1.0      ether1
[admin@MikroTik] > /ip route print
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 ADC  172.16.1.0/24      172.16.1.19     ether1                    0
[admin@MikroTik] > /ip address add address=172.16.11.254/24 interface=ether2
```

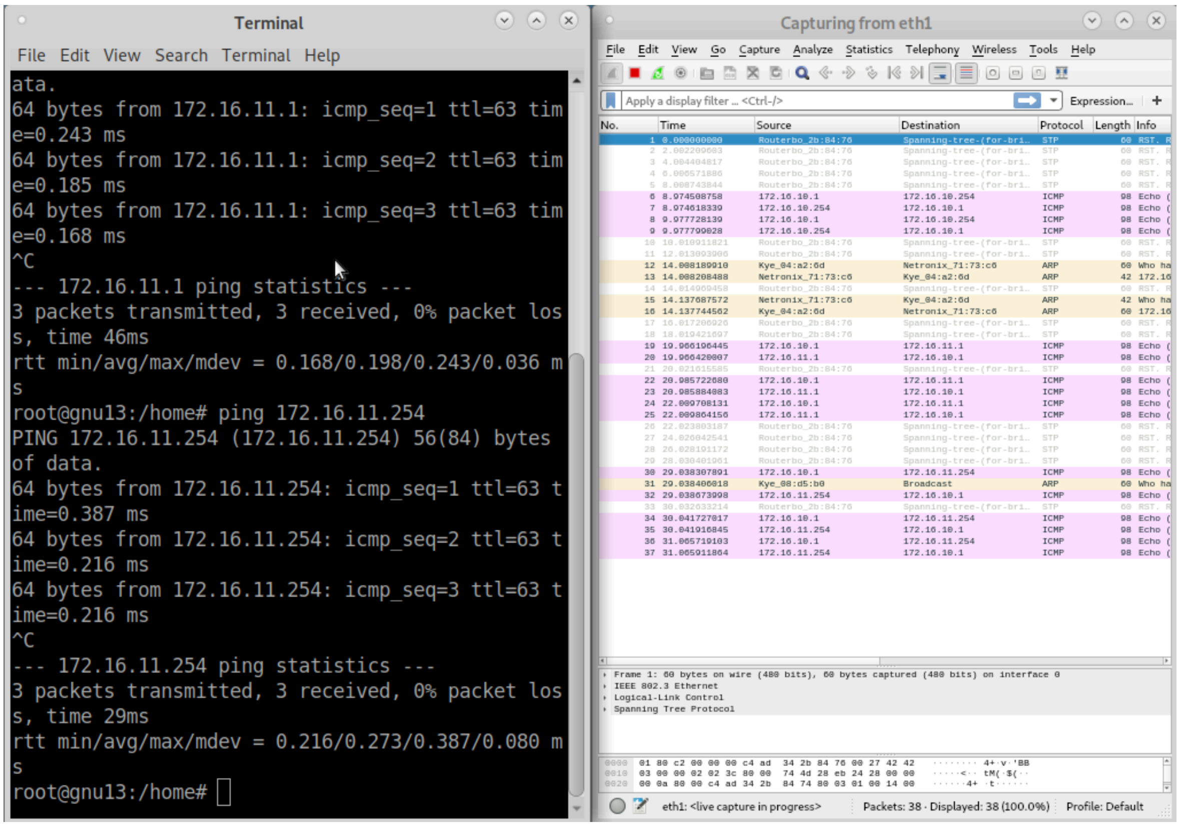
We follow by adding routes in each TUX and the Router, to make possible communicating with the router:

```
route add default gw 172.16.Y1.254 (in Tux2)
route add default gw 172.16.Y0.254 (in Tux3)
route add default gw 172.16.Y1.254 (in Tux4)
```



```
[admin@MikroTik] > /ip route add dst-address=172.16.10.0/24 gateway=172.16.11.253
[admin@MikroTik] > /ip route add dst-address=0.0.0.0/24 gateway=172.16.1.254
[admin@MikroTik] > /ip route print
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 A S  0.0.0.0/24                         172.16.1.254              1
 1 ADC  172.16.1.0/24      172.16.1.19     ether1                    0
 2 A S  172.16.10.0/24                     172.16.11.253             1
 3 ADC  172.16.11.0/24     172.16.11.254   bridge11                  0
[admin@MikroTik] >
```

To verify if everything is correctly setup, we ping TUXY2, TUXY4, and the Router from TUXY3, and capture everything in Wireshark:



```
ata.
64 bytes from 172.16.11.1: icmp_seq=1 ttl=63 tim
e=0.243 ms
64 bytes from 172.16.11.1: icmp_seq=2 ttl=63 tim
e=0.185 ms
64 bytes from 172.16.11.1: icmp_seq=3 ttl=63 tim
e=0.168 ms
^C
--- 172.16.11.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet los
s, time 46ms
rtt min/avg/max/mdev = 0.168/0.198/0.243/0.036 m
s
root@gnu13:/home# ping 172.16.11.254
PING 172.16.11.254 (172.16.11.254) 56(84) bytes
of data.
64 bytes from 172.16.11.254: icmp_seq=1 ttl=63 t
ime=0.387 ms
64 bytes from 172.16.11.254: icmp_seq=2 ttl=63 t
ime=0.216 ms
64 bytes from 172.16.11.254: icmp_seq=3 ttl=63 t
ime=0.216 ms
^C
--- 172.16.11.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet los
s, time 29ms
rtt min/avg/max/mdev = 0.216/0.273/0.387/0.080 m
s
root@gnu13:/home# 
```

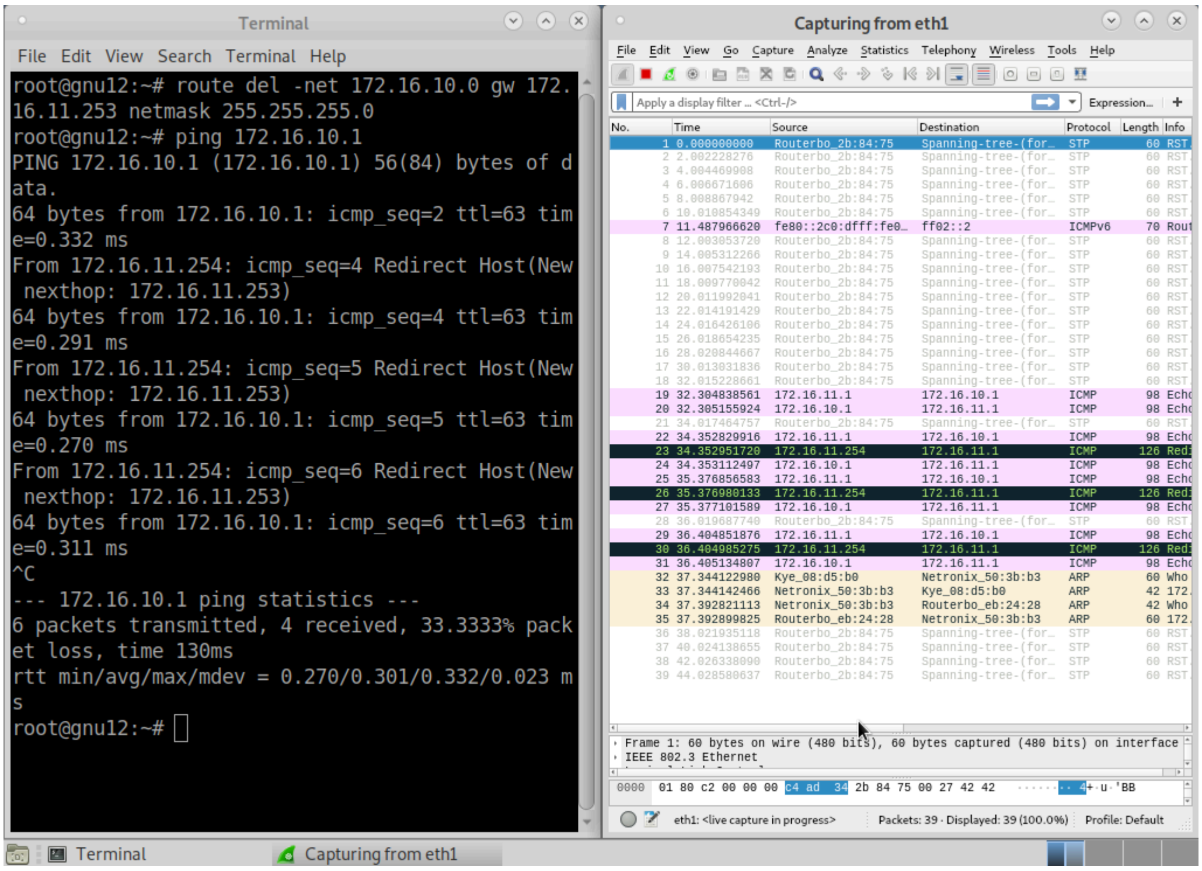
Terminal
File Edit View Search Terminal Help
Capturing from eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time         Source              Destination               Protocol Length Info
  1  0.000000000  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
  2  2.002209603  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
  3  4.004404817  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
  4  6.006571886  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
  5  8.008743844  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
  6  8.974508758  172.16.10.1         172.16.10.254             ICMP     98 Echo (
  7  8.974618339  172.16.10.254       172.16.10.1               ICMP     98 Echo (
  8  9.977728139  172.16.10.1         172.16.10.254             ICMP     98 Echo (
  9  9.977799028  172.16.10.254       172.16.10.1               ICMP     98 Echo (
 10 10.010911821  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 11 12.013093906  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 12 14.008189910  Kye_04:a2:6d        Netronix_71:73:c6         ARP      60 Who ha
 13 14.008208488  Netronix_71:73:c6   Kye_04:a2:6d              ARP      42 172.16
 14 14.014969458  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 15 14.137687572  Netronix_71:73:c6   Kye_04:a2:6d              ARP      42 Who ha
 16 14.137744562  Kye_04:a2:6d        Netronix_71:73:c6         ARP      60 172.16
 17 16.017206926  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 18 18.019421697  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 19 19.966196445  172.16.10.1         172.16.11.1               ICMP     98 Echo (
 20 19.966420007  172.16.11.1         172.16.10.1               ICMP     98 Echo (
 21 20.021615585  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 22 20.985722680  172.16.10.1         172.16.11.1               ICMP     98 Echo (
 23 20.985884083  172.16.11.1         172.16.10.1               ICMP     98 Echo (
 24 22.009708131  172.16.10.1         172.16.11.1               ICMP     98 Echo (
 25 22.009864156  172.16.11.1         172.16.10.1               ICMP     98 Echo (
 26 22.023803187  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 27 24.026042541  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 28 26.028191172  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 29 28.030401961  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 30 29.038307891  172.16.10.1         172.16.11.254             ICMP     98 Echo (
 31 29.038406018  Kye_08:d5:b0        Broadcast                 ARP      60 Who ha
 32 29.038673998  172.16.11.254       172.16.10.1               ICMP     98 Echo (
 33 30.032633214  Routerbo_2b:84:76   Spanning-tree-(for-bri…   STP      60 RST. R
 34 30.041727017  172.16.10.1         172.16.11.254             ICMP     98 Echo (
 35 30.041916845  172.16.11.254       172.16.10.1               ICMP     98 Echo (
 36 31.065719103  172.16.10.1         172.16.11.254             ICMP     98 Echo (
 37 31.065911864  172.16.11.254       172.16.10.1               ICMP     98 Echo (

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Spanning Tree Protocol

0000  01 80 c2 00 00 00 c4 ad  34 2b 84 76 00 27 42 42   ········ 4+·v·'BB
0010  03 00 00 02 02 3c 80 00  74 4d 28 eb 24 28 00 00   ·····<·· tM(·$(··
0020  00 0a 80 00 c4 ad 34 2b  84 74 80 03 01 00 14 00   ······4+ ·t······

eth1: <live capture in progress>   Packets: 38 · Displayed: 38 (100.0%)   Profile: Default
```

Following the guide, we disable Accept Redirects and delete the route connecting TUXY2 to 172.16.Y0.0 through TUXY4, and now use the Router as the Default Gateway:

```
sysctl net.ipv4.conf.eth1.accept_redirects=0
sysctl net.ipv4.conf.all.accept_redirects=0
route del -net 172.16.Y0.0 gw 172.16.Y1.253 netmask
255.255.255.0
route add -net 172.16.Y0.0/24 gw 172.16.Y1.254
```

We now ping TUXY3 to check the results:

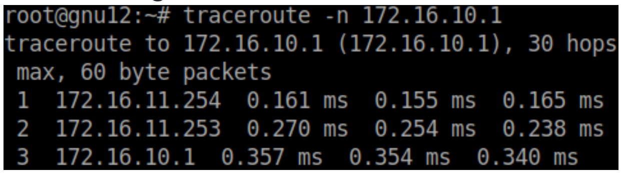


```
root@gnu12:~# route del -net 172.16.10.0 gw 172.
16.11.253 netmask 255.255.255.0
root@gnu12:~# ping 172.16.10.1
PING 172.16.10.1 (172.16.10.1) 56(84) bytes of d
ata.
64 bytes from 172.16.10.1: icmp_seq=2 ttl=63 tim
e=0.332 ms
From 172.16.11.254: icmp_seq=4 Redirect Host(New
 nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=4 ttl=63 tim
e=0.291 ms
From 172.16.11.254: icmp_seq=5 Redirect Host(New
 nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=5 ttl=63 tim
e=0.270 ms
From 172.16.11.254: icmp_seq=6 Redirect Host(New
 nexthop: 172.16.11.253)
64 bytes from 172.16.10.1: icmp_seq=6 ttl=63 tim
e=0.311 ms
^C
--- 172.16.10.1 ping statistics ---
6 packets transmitted, 4 received, 33.3333% pack
et loss, time 130ms
rtt min/avg/max/mdev = 0.270/0.301/0.332/0.023 m
s
root@gnu12:~#
```

```
No.  Time          Source                 Destination             Protocol  Length Info
 1  0.000000000   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
 2  2.002228276   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
 3  4.004469908   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
 4  6.006671606   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
 5  8.008867942   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
 6 10.010854349   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
 7 11.487966620   fe80::2c0:dfff:fe0…    ff02::2                 ICMPv6     70 Rout
 8 12.003053720   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
 9 14.005312266   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
10 16.007542193   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
11 18.009770042   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
12 20.011992041   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
13 22.014191429   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
14 24.016426106   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
15 26.018654235   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
16 28.020844667   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
17 30.013031836   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
18 32.015228661   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
19 32.304838561   172.16.11.1            172.16.10.1             ICMP       98 Echo
20 32.305155924   172.16.10.1            172.16.11.1             ICMP       98 Echo
21 34.017464757   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
22 34.352829916   172.16.11.1            172.16.10.1             ICMP       98 Echo
23 34.352951720   172.16.11.254          172.16.11.1             ICMP      126 Redi
24 34.353112497   172.16.10.1            172.16.11.1             ICMP       98 Echo
25 35.376856583   172.16.11.1            172.16.10.1             ICMP       98 Echo
26 35.376980133   172.16.11.254          172.16.11.1             ICMP      126 Redi
27 35.377101589   172.16.10.1            172.16.11.1             ICMP       98 Echo
28 36.019687740   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
29 36.404851876   172.16.11.1            172.16.10.1             ICMP       98 Echo
30 36.404985275   172.16.11.254          172.16.11.1             ICMP      126 Redi
31 36.405134807   172.16.10.1            172.16.11.1             ICMP       98 Echo
32 37.344122980   Kye_08:d5:b0           Netronix_50:3b:b3       ARP        60 Who
33 37.344142466   Netronix_50:3b:b3      Kye_08:d5:b0            ARP        42 172.
34 37.392821113   Netronix_50:3b:b3      Routerbo_eb:24:28       ARP        42 Who
35 37.392899825   Routerbo_eb:24:28      Netronix_50:3b:b3       ARP        60 172.
36 38.021935118   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
37 40.024138655   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
38 42.026338090   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST
39 44.028580637   Routerbo_2b:84:75      Spanning-tree-(for…     STP        60 RST

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface
IEEE 802.3 Ethernet

0000  01 80 c2 00 00 00 c4 ad  34 2b 84 75 00 27 42 42   ········ 4+·u·'BB

eth1: <live capture in progress>   Packets: 39 · Displayed: 39 (100.0%)   Profile: Default
```

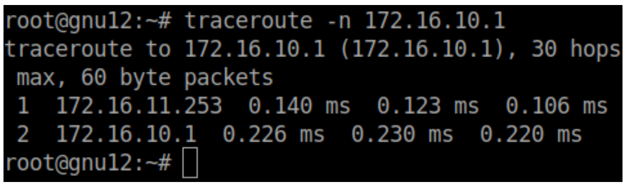
Terminal
File Edit View Search Terminal Help
Capturing from eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
Terminal
Capturing from eth1

Using traceroute -n 172.16.Y0.1:



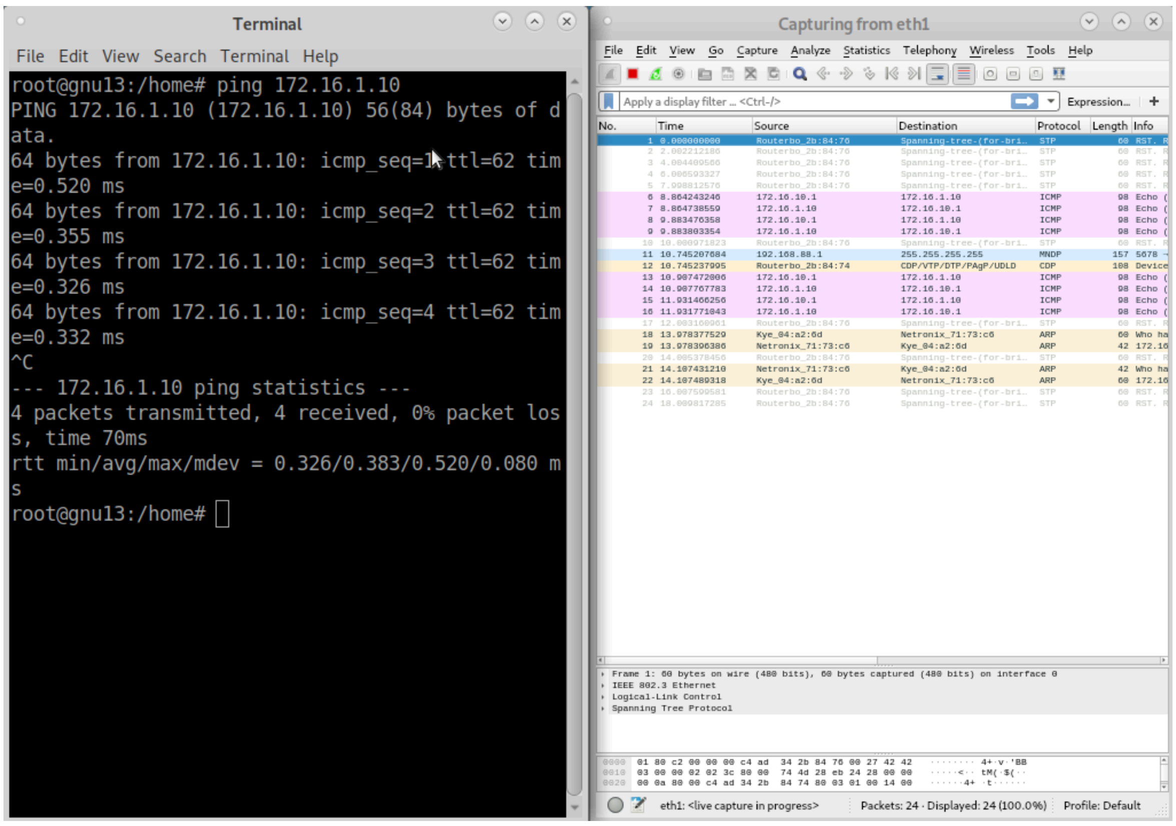
```
root@gnu12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops
 max, 60 byte packets
 1  172.16.11.254  0.161 ms  0.155 ms  0.165 ms
 2  172.16.11.253  0.270 ms  0.254 ms  0.238 ms
 3  172.16.10.1  0.357 ms  0.354 ms  0.340 ms
```

Now, we change the routes to use TUXY4 as the gateway to 172.16.Y0.0 and use traceroute-n 172.16.Y0.1 again:



```
root@gnu12:~# traceroute -n 172.16.10.1
traceroute to 172.16.10.1 (172.16.10.1), 30 hops
 max, 60 byte packets
 1  172.16.11.253  0.140 ms  0.123 ms  0.106 ms
 2  172.16.10.1  0.226 ms  0.230 ms  0.220 ms
root@gnu12:~#
```

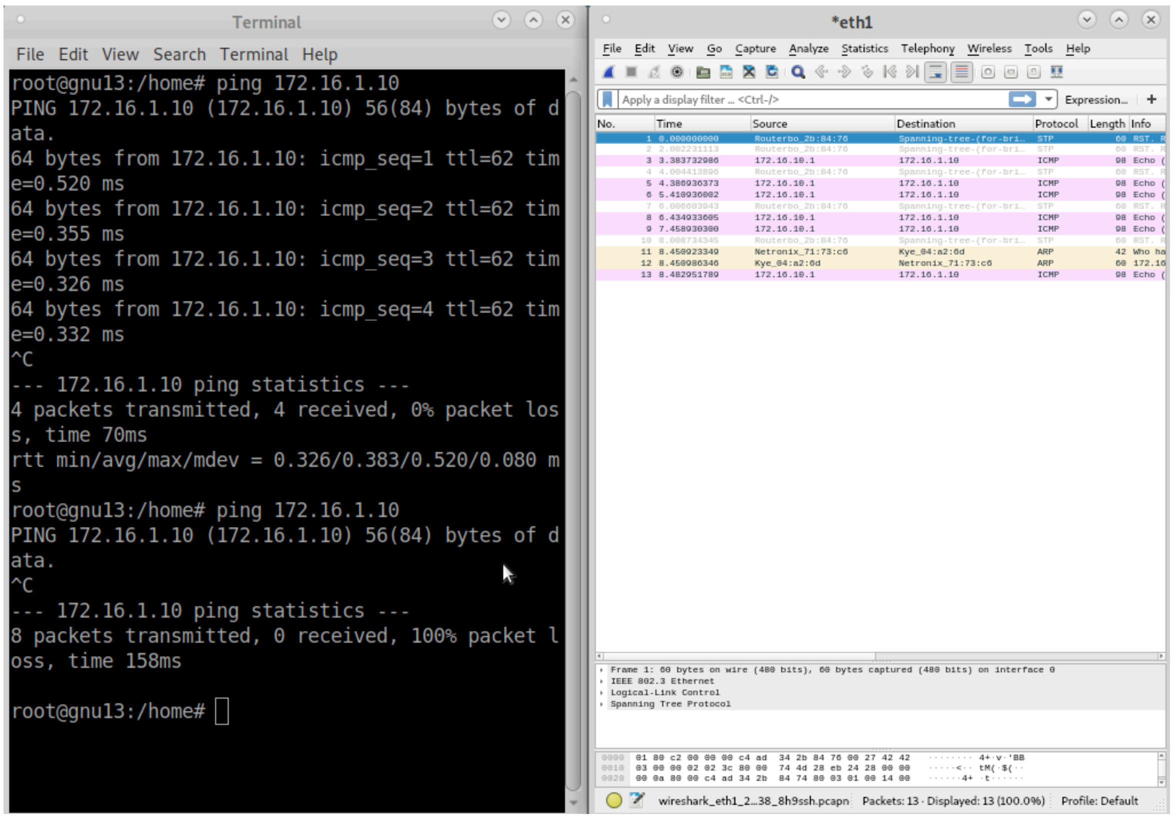
Finally, we try to ping the FTP server (with NAT Enabled by default) and check the results on Wireshark:


Terminal
File Edit View Search Terminal Help
Capturing from eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
root@gnu13:/home# ping 172.16.1.10
PING 172.16.1.10 (172.16.1.10) 56(84) bytes of d
ata.
64 bytes from 172.16.1.10: icmp_seq=1 ttl=62 tim
e=0.520 ms
64 bytes from 172.16.1.10: icmp_seq=2 ttl=62 tim
e=0.355 ms
64 bytes from 172.16.1.10: icmp_seq=3 ttl=62 tim
e=0.326 ms
64 bytes from 172.16.1.10: icmp_seq=4 ttl=62 tim
e=0.332 ms
^C
--- 172.16.1.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet los
s, time 70ms
rtt min/avg/max/mdev = 0.326/0.383/0.520/0.080 m
s
root@gnu13:/home# 
```

```
No.  Time          Source               Destination               Protocol Length Info
  1  0.000000000   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  2  2.002212186   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  3  4.004409566   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  4  6.006593327   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  5  7.998812576   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  6  8.864243246   172.16.10.1          172.16.1.10               ICMP       98   Echo (
  7  8.864738559   172.16.1.10          172.16.10.1               ICMP       98   Echo (
  8  9.883476358   172.16.10.1          172.16.1.10               ICMP       98   Echo (
  9  9.883803354   172.16.1.10          172.16.10.1               ICMP       98   Echo (
 10  10.000971823  Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
 11  10.745207684  192.168.88.1         255.255.255.255           MNDP      157   5678 →
 12  10.745237995  Routerbo_2b:84:74    CDP/VTP/DTP/PAgP/UDLD     CDP       108   Device
 13  10.987472006  172.16.10.1          172.16.1.10               ICMP       98   Echo (
 14  10.987767783  172.16.1.10          172.16.10.1               ICMP       98   Echo (
 15  11.931466256  172.16.10.1          172.16.1.10               ICMP       98   Echo (
 16  11.931771043  172.16.1.10          172.16.10.1               ICMP       98   Echo (
 17  12.003160961  Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
 18  13.978377529  Kye_04:a2:6d         Netronix_71:73:c6         ARP        60   Who ha
 19  13.978396386  Netronix_71:73:c6    Kye_04:a2:6d              ARP        42   172.16
 20  14.005378456  Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
 21  14.107431210  Netronix_71:73:c6    Kye_04:a2:6d              ARP        42   Who ha
 22  14.107489318  Kye_04:a2:6d         Netronix_71:73:c6         ARP        60   172.16
 23  16.007599581  Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
 24  18.009817285  Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Spanning Tree Protocol

0000  01 80 c2 00 00 00 c4 ad  34 2b 84 76 00 27 42 42   ········ 4+·v·'BB
0010  03 00 00 02 02 3c 80 00  74 4d 28 eb 24 28 00 00   ·····<·· tM(·$(··
0020  00 0a 80 00 c4 ad 34 2b  84 74 80 03 01 00 14 00   ······4+ ·t······

eth1: <live capture in progress>   Packets: 24 · Displayed: 24 (100.0%)   Profile: Default
```

Now, we disable NAT in the MikroTIK console and attempt to ping the FTP Server once again, while capturing everything on Wireshark:


Terminal
File Edit View Search Terminal Help
*eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
root@gnu13:/home# ping 172.16.1.10
PING 172.16.1.10 (172.16.1.10) 56(84) bytes of d
ata.
64 bytes from 172.16.1.10: icmp_seq=1 ttl=62 tim
e=0.520 ms
64 bytes from 172.16.1.10: icmp_seq=2 ttl=62 tim
e=0.355 ms
64 bytes from 172.16.1.10: icmp_seq=3 ttl=62 tim
e=0.326 ms
64 bytes from 172.16.1.10: icmp_seq=4 ttl=62 tim
e=0.332 ms
^C
--- 172.16.1.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet los
s, time 70ms
rtt min/avg/max/mdev = 0.326/0.383/0.520/0.080 m
s
root@gnu13:/home# ping 172.16.1.10
PING 172.16.1.10 (172.16.1.10) 56(84) bytes of d
ata.
^C
--- 172.16.1.10 ping statistics ---
8 packets transmitted, 0 received, 100% packet l
oss, time 158ms

root@gnu13:/home# 
```

```
No.  Time          Source               Destination               Protocol Length Info
  1  0.000000000   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  2  2.002231113   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  3  3.383732986   172.16.10.1          172.16.1.10               ICMP       98   Echo (
  4  4.004413896   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  5  4.386936373   172.16.10.1          172.16.1.10               ICMP       98   Echo (
  6  5.410936002   172.16.10.1          172.16.1.10               ICMP       98   Echo (
  7  6.006603943   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
  8  6.434933605   172.16.10.1          172.16.1.10               ICMP       98   Echo (
  9  7.458930300   172.16.10.1          172.16.1.10               ICMP       98   Echo (
 10  8.008734345   Routerbo_2b:84:76    Spanning-tree-(for-bri…   STP        60   RST. R
 11  8.450923349   Netronix_71:73:c6    Kye_04:a2:6d              ARP        42   Who ha
 12  8.450986346   Kye_04:a2:6d         Netronix_71:73:c6         ARP        60   172.16
 13  8.482951789   172.16.10.1          172.16.1.10               ICMP       98   Echo (

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Spanning Tree Protocol

0000  01 80 c2 00 00 00 c4 ad  34 2b 84 76 00 27 42 42   ········ 4+·v·'BB
0010  03 00 00 02 02 3c 80 00  74 4d 28 eb 24 28 00 00   ·····<·· tM(·$(··
0020  00 0a 80 00 c4 ad 34 2b  84 74 80 03 01 00 14 00   ······4+ ·t······

wireshark_eth1_2…38_8h9ssh.pcapn   Packets: 13 · Displayed: 13 (100.0%)   Profile: Default
```

Experiment 5 - DNS

In this experience, the goal is to configure the DNS in a way that we could access internet websites through the name of its domain. To configure it we just had to edit the file in the path “/etc/resolv.conf” in every TUX adding the line “nameserver 10.227.netlab.fe.up.pt”. This IP address is of the router present in the lab room.

After this, we could test if the configuration was done correctly by making a ping of google.com. We observed that the exchange of DNS packets happens **before** any other protocol, since it is the one that performs the translation of the name of a domain to its IP address, that will be used by every other protocol.

Experiment 6 - TCP connections

In this experience we tested our download application in C using the private network of the lab environment, that we already explained every detail above.

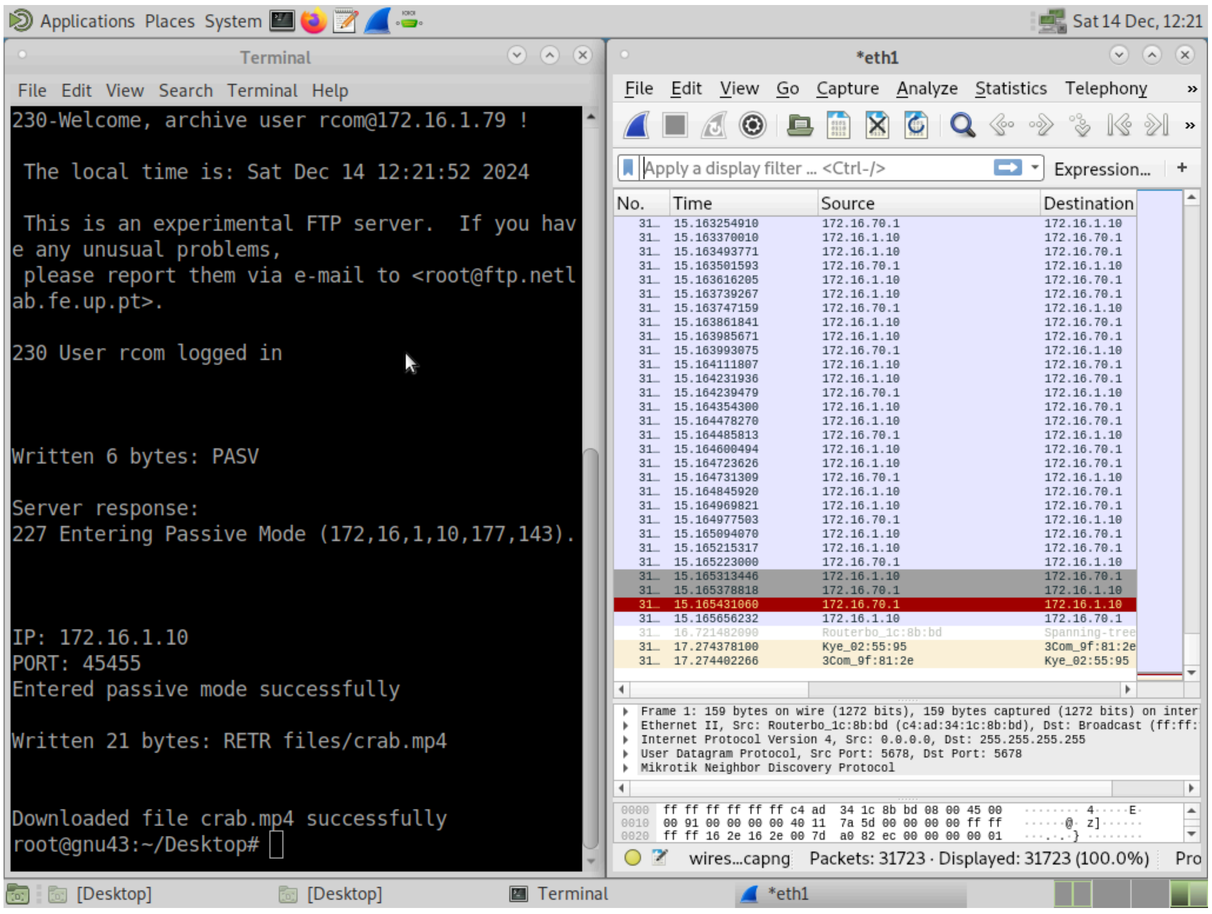
To do this, we first need to compile the C code with:

- gcc -o download download.c

And start the transfer of a file using our application:

- ./download <URL>

To check the file transfer we start a Wireshark capture in TUXY3:


Applications Places System
Sat 14 Dec, 12:21
Terminal
File Edit View Search Terminal Help
*eth1
File Edit View Go Capture Analyze Statistics Telephony
Apply a display filter ... <Ctrl-/>
Expression...
[Desktop]
[Desktop]
Terminal
*eth1

```
230-Welcome, archive user rcom@172.16.1.79 !

 The local time is: Sat Dec 14 12:21:52 2024

 This is an experimental FTP server.  If you hav
e any unusual problems,
 please report them via e-mail to <root@ftp.netl
ab.fe.up.pt>.

230 User rcom logged in



Written 6 bytes: PASV

Server response:
227 Entering Passive Mode (172,16,1,10,177,143).



IP: 172.16.1.10
PORT: 45455
Entered passive mode successfully

Written 21 bytes: RETR files/crab.mp4


Downloaded file crab.mp4 successfully
root@gnu43:~/Desktop# 
```

```
No.   Time           Source              Destination
 31…  15.163254910   172.16.70.1         172.16.1.10
 31…  15.163370010   172.16.1.10         172.16.70.1
 31…  15.163493771   172.16.1.10         172.16.70.1
 31…  15.163501593   172.16.70.1         172.16.1.10
 31…  15.163616205   172.16.1.10         172.16.70.1
 31…  15.163739267   172.16.1.10         172.16.70.1
 31…  15.163747159   172.16.70.1         172.16.1.10
 31…  15.163861841   172.16.1.10         172.16.70.1
 31…  15.163985671   172.16.1.10         172.16.70.1
 31…  15.163993075   172.16.70.1         172.16.1.10
 31…  15.164111807   172.16.1.10         172.16.70.1
 31…  15.164231936   172.16.1.10         172.16.70.1
 31…  15.164239479   172.16.70.1         172.16.1.10
 31…  15.164354300   172.16.1.10         172.16.70.1
 31…  15.164478270   172.16.1.10         172.16.70.1
 31…  15.164485813   172.16.70.1         172.16.1.10
 31…  15.164600494   172.16.1.10         172.16.70.1
 31…  15.164723626   172.16.1.10         172.16.70.1
 31…  15.164731309   172.16.70.1         172.16.1.10
 31…  15.164845920   172.16.1.10         172.16.70.1
 31…  15.164969821   172.16.1.10         172.16.70.1
 31…  15.164977503   172.16.70.1         172.16.1.10
 31…  15.165094070   172.16.1.10         172.16.70.1
 31…  15.165215317   172.16.1.10         172.16.70.1
 31…  15.165223000   172.16.70.1         172.16.1.10
 31…  15.165313446   172.16.1.10         172.16.70.1
 31…  15.165378818   172.16.70.1         172.16.1.10
 31…  15.165431060   172.16.70.1         172.16.1.10
 31…  15.165656232   172.16.1.10         172.16.70.1
 31…  16.721482090   Routerbo_1c:8b:bd   Spanning-tree
 31…  17.274378100   Kye_02:55:95        3Com_9f:81:2e
 31…  17.274402266   3Com_9f:81:2e       Kye_02:55:95

 Frame 1: 159 bytes on wire (1272 bits), 159 bytes captured (1272 bits) on inter
 Ethernet II, Src: Routerbo_1c:8b:bd (c4:ad:34:1c:8b:bd), Dst: Broadcast (ff:ff:
 Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
 User Datagram Protocol, Src Port: 5678, Dst Port: 5678
 Mikrotik Neighbor Discovery Protocol

0000  ff ff ff ff ff ff c4 ad  34 1c 8b bd 08 00 45 00   ········ 4·····E·
0010  00 91 00 00 00 00 40 11  7a 5d 00 00 00 00 ff ff   ······@· z]······
0020  ff ff 16 2e 16 2e 00 7d  a0 82 ec 00 00 00 00 01   ···.·.·} ········

wires...capng   Packets: 31723 · Displayed: 31723 (100.0%)   Pro
```

And verify that the file is being transferred correctly.

For the other experience we did the same but in the middle of the download, initiated a transfer in TUXY2 and checked the capture of both machines.

For more details or doubts, check the download application's architecture above and the annexes below.

Conclusions

The lab work presented in this report successfully met its objectives, showcasing the practical implementation of both an FTP client application and a series of network experiments.

The FTP client, developed in C, demonstrated robust adherence to protocol standards (RFC959 and RFC1738) and performed reliably under various test scenarios. The download tests validated its functionality across multiple file types and sizes, proving its efficacy in file retrieval over FTP.

The network configuration experiments provided hands-on insights of what was learned from the theoretical and practical classes, applying critical networking concepts, such as ARP resolution, bridging, routing, NAT implementation, DNS configuration, and TCP congestion control.

Key takeaways from the experiments include:

- The importance of ARP in resolving MAC addresses for IP communication.
- The segmentation of broadcast domains using bridges and their impact on network isolation.
- The role of routing in enabling communication across distinct subnets and the significance of proper route configuration.
- The functionality and advantages of NAT in translating private to public addresses for internet access.
- The intricate mechanisms of TCP, including connection establishment, congestion control, and the influence of multiple simultaneous connections on bandwidth allocation.
- Etc.

Through systematic testing, analysis, and troubleshooting, this lab work not only achieved its technical goals but also reinforced foundational concepts of the RCOM UC.

Below are our answers to the questions asked in this lab's guide:

Experiment 1 - Configure an IP Network:

Q: What are the ARP packets and what are they used for?

A: ARP packets are packets used to establish a connection between an IP and an MAC address.

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

A: Two IP addresses are sent in broadcast mode: source TuxY3 (172.16.Y0.1) and destination TuxY4 (172.16.Y0.254) in the same ARP packet.

Q: What are the MAC and IP addresses of the ping packets?

A: IP and MAC addresses are used as a whole packet (ICMP) to the communication between TuxY3 and TuxY4.

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

A: We can distinguish by looking at the protocol of a specific frame, that is indicated in the first 2 bytes of the frame.

Q: How to determine the length of a receiving frame?

A: There is a column in Wireshark that shows the length in bytes of the specific frame.

Q: What is a loopback interface and why is it important?

A: It's a virtual interface that is always reachable if at least one of the switches are operational. Useful to check if the network is well implemented.

Experiment 2 - Implement two bridges in a switch

Q: How to configure bridgeY0?

A: BridgeY0 was configured to serve as a means to connect computers TuxY3 and TuxY54. We removed the switch's default configurations and connections to configure new connections in every machine.

Q: How many broadcast domains are there? How can you conclude it from the logs?

A: There are two broadcast domains because there are two bridges implemented. Analysing Wireshark, we can see that we can ping TuxY4 from TuxY3 but not TuxY2 because they are in two different bridges.

Experiment 3 - Configuring a Router in Linux

Q: What routes are there in the tuxes? What are their meaning?

A: There are routes in TuxY2 and TuxY3, and both of them have as their gateway TuxY4 because it is the router.

Q: What information does an entry of the forwarding table contain?

A: Each table's entry contains the destination address and its gateway.

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

A: The ARP packets that are exchanged from TuxY3 to TuxY2 contain the MAC addresses, but not the destination. This is because TuxY3 knows the gateway's address (router) that leads it to TuxY2, but doesn't know the TuxY2 address.

Q: What ICMP packets are observed and why?

A: ICMP packets contain the source address and the destination address, and if we can see them, that means the network is correctly configured.

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

A: Each ICMP packet observed through ping from TuxY3 contains the source address (IP of TuxY3) and the destination address (IP of TuxY2).

Experiment 4 - Configure a Commercial Router and Implement NAT

Q: How to configure a static route in a commercial router?

A: Reset its configurations, add it to the respective bridge and give it an external and internal IP.

Q: What are the paths followed by the packets in the experiments carried out and why?

A: Initially, packets sent by TUXY2 to TUXY3 or vice-versa, passed through TUXY4. However, with the router as the default gateway, packets sent from TUXY3 to TUXY2 now reached the router before going through TUXY4 and finally TUXY2 and the other way around for packets sent from TUXY2 to TUXY3. Any packet sent to the FTP Server passed through the router before being sent to the FTP Server

Q: How to configure NAT in a commercial router?

A: In the mikrotik's terminal of the router, running the command "/ip firewall nat enable 0".

Q: What does NAT do?

A: A NAT is a Network Address Translation, that (as the name suggests) translates addresses of the local network to a public address, and the other way around. That way, when a packet is sent to an external network, it is sent as a public address **as source**, and when the destination computer answers to that public address, that is translated afterwards back to the local address.

Experiment 5 - DNS

Q: How to configure the DNS service at a host?

A: The DNS service can be configured by adding "nameserver <IP address>" in the file /etc/resolv.conf in every Tux.

Q: What packets are exchanged by DNS and what information is transported?

A: The packets are initially DNS, to make the router translate them to the destination IP.

Experiment 6 - TCP connections

Q: How many TCP connections are opened by your ftp application?

A: Two.

Q: In what connection is transported the FTP control information?

A: The one where there is transfer of control commands and reception of the server's messages.

Q: What are the phases of a TCP connection?

A: 1. DNS, 2. Connection, 3. Configuration, 4. File transfer, 5. Ending.

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

A: Automatic Repeat Request is used to make retransmission on a network that is losing packets, and it is necessary to send many packets at once until that happens.

Q: How does the TCP congestion control mechanism work? What are the relevant fields. How did the through put of the data connection evolve along the time? Is it according the TCP congestion control mechanism?

A: Each transmitter determines the capacity of the communication to know how many packets it could send. For this, there is one more connection parameter on the connection, Congestion Window, that if the congestion level increases, the CongestionWindow decreases.

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

A: Yes, making more than one TCP connection, the bandwidth is splitted between the multiple connections, decreasing the speed of each one of them.

Annexes

download.c:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FTP_PORT 21
#define MAX_BUFFER_SIZE 255
#define COMMAND_SIZE 512

typedef struct {
    char user[MAX_BUFFER_SIZE];
    char password[MAX_BUFFER_SIZE];
    char host[MAX_BUFFER_SIZE];
    char path[MAX_BUFFER_SIZE];
    char filename[MAX_BUFFER_SIZE];
    char ip[MAX_BUFFER_SIZE];
} URL;

// Get the IP address of a host
int getip(char host[], URL* url) {
    if (url == NULL) {
        printf("ERROR: url is NULL\n");
        return 1;
    }

    struct hostent *h;
    if ((h = gethostbyname(host)) == NULL) {
        printf("ERROR: gethostbyname() failed, h is NULL\n");
        return 1;
    }

    strcpy(url->ip, inet_ntoa(*((struct in_addr *)h->h_addr_list[0])));
    return 0;
}

// Create a struct URL with all fields set to 0
int create_url(URL* url) {
    if (url == NULL) {
        printf("ERROR: url is NULL\n");
        return 1;
    }

    memset(url->user, 0, MAX_BUFFER_SIZE);
    memset(url->password, 0, MAX_BUFFER_SIZE);
    memset(url->host, 0, MAX_BUFFER_SIZE);
    memset(url->path, 0, MAX_BUFFER_SIZE);
    memset(url->filename, 0, MAX_BUFFER_SIZE);
    memset(url->ip, 0, MAX_BUFFER_SIZE);
    return 0;
}

// Parse the filename from the URL path
int parse_URL(char *input, URL *url) {
    if (input == NULL || url == NULL) {
        printf("ERROR: input or url is NULL\n");
        return 1;
    }

    char *user, *password, *host, *path;
    int length = strlen(input);

    if (strncmp(input, "ftp://", 6) != 0) {
        printf("ERROR: parse_URL, URL Doesn't start with 'ftp://'\n");
        return 1;
    }

    int hasUserAndPassword = 0;
    for (int i = 0; i < length; i++) {
        if (input[i] == '@') {
            hasUserAndPassword = 1;
            break;
        }
    }

    if (hasUserAndPassword) {
        char user_delimiter[] = ":";
        user = strtok(&input[6], user_delimiter);
        strcpy(url->user, user);

        char pass_delimiter[] = "@";
        int password_index = strlen(user) + 7;
        password = strtok(&input[password_index], pass_delimiter);
        strcpy(url->password, password);

        char host_delimiter[] = "/";
        int host_index = password_index + strlen(password) + 1;
        host = strtok(&input[host_index], host_delimiter);
        strcpy(url->host, host);

        char url_delimiter[] = "\n";
        int url_index = host_index + strlen(host) + 1;
        path = strtok(&input[url_index], url_delimiter);
```

```c
        strcpy(url->path, path);
    } else {
        strcpy(url->user, "anonymous");
        strcpy(url->password, "");

        const char host_delimiter[] = "/";
        host = strtok(&input[6], host_delimiter);
        strcpy(url->host, host);

        const char url_delimiter[] = "\n";
        int url_index = strlen(host) + 7;
        path = strtok(&input[url_index], url_delimiter);
        strcpy(url->path, path);
    }

    return 0;
}

// Parse the filename from the path
int parse_filename(char *path, URL *url) {
    if (path == NULL || url == NULL) {
        printf("ERROR: path or url is NULL\n");
        return 1;
    }

    char filename[MAX_BUFFER_SIZE];
    strcpy(filename, path);

    char remove[MAX_BUFFER_SIZE];
    while (strchr(filename, '/')) {
        char path_delimiter[] = "/";
        strcpy(remove, strtok(&filename[0], path_delimiter));
        strcpy(filename, filename + strlen(remove) + 1);
    }

    strcpy(url->filename, filename);

    if (url->filename == NULL || strlen(url->filename) == 0) {
        printf("ERROR: url->filename is NULL\n");
        return 1;
    }

    return 0;
}

// Create a socket and connect it to the server
int create_socket(char* ip, int port) {
    if (ip == NULL) {
        printf("ERROR: ip is NULL\n");
        return -1;
    }

    int sockfd;
    struct sockaddr_in server_addr;
    bzero((char*) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ip);
    server_addr.sin_port = htons(port);

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("ERROR: Failed to Open TCP Socket socket()\n");
        exit(-1);
    }

     if (connect(sockfd, (struct sockaddr *) &server_addr, sizeof(server_addr)) <
0) {
        printf("ERROR: Failed to connect socket to server connect()\n");
        exit(-1);
    }

    return sockfd;
}

// Read the response from the server
int read_socket(int sockfd, char* response) {
    if (response == NULL) {
        printf("ERROR: response is NULL\n");
        return 1;
    }

    memset(response, 0, MAX_BUFFER_SIZE);
    FILE* file = fdopen(sockfd, "r");
    response = fgets(response, MAX_BUFFER_SIZE, file);

    while (!(response[0] <= '5' && '1' <= response[0]) || response[3] != ' ') {
        memset(response, 0, MAX_BUFFER_SIZE);
        response = fgets(response, MAX_BUFFER_SIZE, file);
    }

    return 0;
}

// Send a command to the server writing it to the socket
int  write_command(int  sockfd,  const  char*  command,  int  command_size,  char*
response) {
    if (command == NULL || response == NULL) {
        printf("ERROR: command or response is NULL\n");
        return 1;
    }

    int res = write(sockfd, command, command_size);
    if (res <= 0) {
        printf("ERROR: write()\n");
        return 1;
    }

    if (read_socket(sockfd, response) != 0) {
        printf("ERROR: read_socket()\n");
        return 1;
    }

    return 0;
}
```