U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Computer Networks – 2nd Project Report

Luís Duarte - 202108734
Madalena Ye - 202108795

December 23, 2023

Abstract

The goal of this project is to establish a compact, decentralized computer network with the purpose of downloading a file from an FTP server. This objective was successfully attained, with both the network and the download application meeting the specified requirements in the guide provided. The hands-on approach, coupled with additional reflection on the proceedings, provided opportunities for more profound learning.

1 Introduction

The second lab assignment is segmented into two components: the creation of an FTP download application and the configuration and study of a computer network. Further details on each aspect will be explored below.

2 Download Application

The first part of the project involves creating a straightforward FTP application tasked with fetching a designated file utilizing the FTP protocol outlined in RFC959. This application is designed to accept an argument following the URL syntax as specified in RFC1738. For example:

```
download
ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
```

where the URL path is

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

This allowed us to learn about the following points:

- Explain the client-server concept and its nuances in TCP/IP
- Locate and read RFCs
- Define application protocols in general, outline URL characteristics, and provide a detailed explanation of FTP behavior.
- Implement a simple FTP client in C language
- UNIX utilities to communicate with remote services like sockets
- Gain insight into the functionality of the DNS service and apply it in a client program

2.1 Architecture of the download application

The application's flow can be described by a series of consecutive steps, as follows:

1. Initially, the URL passed as the argument to the program is parsed and processed, using regular expressions, to obtain a data structure with essential information for establishing the connection. This includes the username, password, host, port, and path.
2. We establish the connection to the provided host using the specified port, which defaults to 21, the standard FTP control port.
3. After logging in, we enter passive mode and retrieve the host and port information for the data connection.

4. We retrieve the requested file's size in the main connection. Subsequently, we can verify if the file exists, as it would trigger an error.

5. The establishment of a connection to the data connection's host and port is done next so that we can initiate a signal to instruct the remote server to start the transfer of the file specified in the path segment of the URL string.

6. We read a data buffer from the data connection and save it to a local file with the same name as the one being transferred. We can determine the completion of the file transfer by referencing the previously obtained file size.

7. Finally, we close the connection.

2.2 Report of a Successful Download

In order to assess the proper functionality of the download application, several tests were conducted involving file transfers of varying sizes, ranging from small files with only a few dozen bytes to larger ones measuring multiple gigabytes. Numerous layers of input validation were also tested, including incorrect URLs in various fields, non-existent resources, and both anonymous and non-anonymous modes. In all presented scenarios, the program demonstrated resilience and efficiency. The following figure illustrates the outcome of a successful transfer:

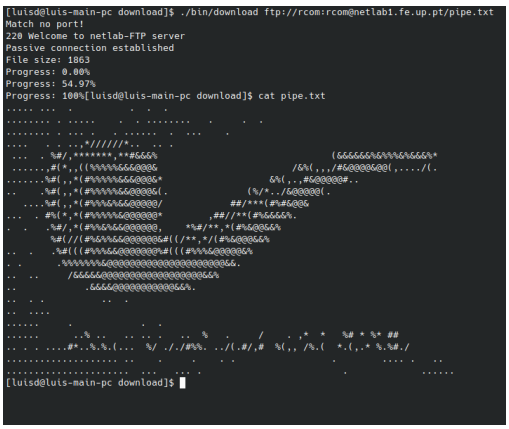

Fig. 1. File transfer example

The log of a successful download is attached as annex to experiment 6.

3 Network Configuration and Analysis

This part of the project focused on developing a small network from a computer that normally doesn't have internet to a fully-fledged network with internet access to test our FTP download application developed in the first part of the project.

In order to achieve that, the networking configuration was done incrementally, subdivided into small experiments that led up to a network that we could test our FTP application.

All experiments were done on bench 1 of Lab2.

3.1 Experiment 1

3.1.1 Network Configuration

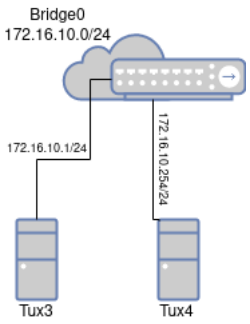

Fig. 2. Experiment 1 network architecture

In order to achieve this network configuration, we use the *MikroTik* switch to connect between the two computers: tux3 and tux4, as you can see in the figure above.

3.1.2 Experiment Objectives

The purpose of this experiment is to learn how to make two computers communicate through a network and also how to learn the way the communicate.

3.1.3 Main Configuration Commands

```
#switch
/interface bridge
remove 0
add name=bridge0
/interface bridge port
remove numbers=8,9
add bridge=bridge0 interface=ether9
add bridge=bridge0 interface=ether10
#tux 3
ip a add 172.16.10.1/24 dev eth0
```

```
ip route add 172.16.10.0/24 dev
    eth0 src 172.16.10.1
#tux 4
ip a add 172.16.10.254/24 dev eth0
ip route add 172.16.10.0/24 dev
    eth0 src 172.16.10.254
```

3.1.4 Log analysis

As we can see from our captured data (which is annexed to this report), using *Wireshark* to capture and view the data from both machines.

On Tux3, we make the *ping* command to Tux4 IP address, *172.16.10.254*. Because Tux4 still doesn't know which MAC address to send the packet to, it needs a way to "*translate*" an IP address to a MAC address. In order to do that, it uses the ARP protocol, which consists of sending a packet to every computer on the network, also called a *broadcast* packet, in which only the computer with the desired IP address will respond with an ARP answer packet. *Broadcast* frames, on the Ethernet protocol, have a MAC address of *FF:FF:FF:FF:FF:FF* which are accepted on all computers. To avoid loopback, the switch sends the packet to everyone except the sender of that packet.

In this situation, Tux3 sends an ARP probe with the source MAC address of *00:21:5A:5A:7D:16* and the destination MAC of *FF:FF:FF:FF:FF:FF*, the packet sends the source IP, which is *172.16.10.1* and the desired IP being probed of *172.16.10.254*. When Tux4 receives the ARP probe of Tux3 it will send a reply to Tux3's MAC address with its own MAC address, *00:C0:DF:25:13:65*. When Tux3 receives the ARP response, it will store the MAC address of Tux4 associated with its IP address.

After discovering the destination MAC address, Tux3 can now proceed with sending *ping* requests which in turn Tux4 responds, each packet exchanged contains the IP address and the MAC address of both. These *ping* requests are done on the ICMP protocol.

ICMP and ARP are completely different protocols they are distinguished by the Ethernet frame using the type parameter, which can be located on bytes 13 and 14 on the frame. In this case, ARP has a type of *0x0806* and the ICMP protocol has no type exclusive to it (because it uses the IPV4/IPV6 protocol) so in this case, it used the IPV4 type, *0x8000*.

The length of each Ethernet frame is determined by the underlying protocol: ARP has a fixed byte length of 28 bytes, but because an Ethernet frame must always be at least 64 bytes long, it adds a padding of 0s until it meets the criteria. In the case of the IPV4 protocol, the size is determined by bytes 3 and 4 at the beginning of the IPV4 packet, we have to sum this size by the frame size (and possible padding).

The loopback interface is a virtual interface implemented by the kernel's networking stack, it has no purpose of communicating with other machines but can be used for testing purposes, and testing the validity of the networking stack and its applications. It can also be used for communication of different processes on a machine, eg.: a browser communicating with a local web server.

3.2 Experiment 2

3.2.1 Network Architecture

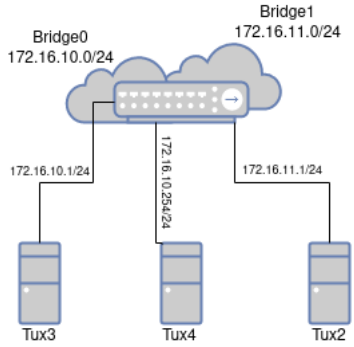

Fig. 3. Experiment 2 network architecture

Adding to the previous network architecture, we connect Tux2 to the same switch but on a different domain *172.16.11.0* with the IP address *172.16.11.1*. On the switch side, we create a separate bridge for the port that Tux2 is connected to. This means that even though bridge0 and bridge1 are connected to the same device is as though bridge0 and bridge1 are not physically connected.

3.2.2 Experiment Objectives

In this experiment, we are expected to learn and configure two different networks that are connected through the same device, in this case, our *MikroTik* switch, and verify that there is no communication between the two networks.

3.2.3 Main Configuration Commands

```
#switch
/interface bridge
remove 0
add name=bridge0
add name=bridge1


/interface bridge port
remove numbers=0,1,2,8,9
add bridge=bridge0 interface=ether9
add bridge=bridge0 interface=ether10
add bridge=bridge1 interface=ether1

#tux2
ip a add 172.16.11.1/24 dev eth0
```

3.2.4 Log analysis

As expected, Tux4 and Tux3 can still reach each other by the *ping* command because their configurations have not changed from the previous experiment, as we can prove from the packet capture logs.

We were also asked to do the *ping* command again on Tux3 but instead of pinging a single host, we broadcast it on an entire domain. In this case, we get a response from Tux4 but we don't have any response on Tux2 (and we don't receive anything on Tux2, as you can see by the captures). This indicates that Tux2 is not connected physically to the other 2 machines.

Such behavior is validated by our *MikroTik* switch configuration, that we have two separate networks operating through the same switch.

3.3 Experiment 3

3.3.1 Network Architecture

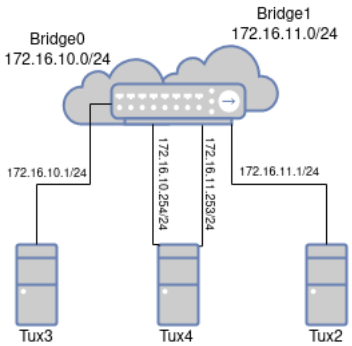

Fig. 4. Experiment 3 network architecture

In this experiment, we add a connection to the second Ethernet port of Tux4, *eth1*, that is connected to *bridge1* of the *MikroTik* router. In summary, we have two bridges: *bridge0* that is connected to Tux3 and the first Ethernet port of Tux4, we also have *bridge1* that is connected to the second Ethernet port of Tux4 and Tux2.

3.3.2 Experiment Objectives

In this experiment, we are expected to learn how a router works and how to configure it properly, to allow communication between two different networks, *172.16.10.0/24* and *172.16.11.0/24*. In order words, the objective is to allow communication between Tux3 and Tux4.

3.3.3 Main Configuration Commands

```
#switch
/interface bridge
remove 0
add name=bridge0
add name=bridge1


/interface bridge port
remove numbers=0,1,2,8,9
add bridge=bridge0 interface=ether9
add bridge=bridge0 interface=ether10
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2

#tux2
ip route add 172.16.10.0/24 dev
   eth0 via 172.16.11.253

#tux3
ip route add default dev eth0 via
   172.16.10.254

#tux4
ip a add 172.16.11.253/24 dev eth1
ip route add 172.16.11.0/24 dev
   eth1 src 172.16.11.253
sysctl net.ipv4.ip_forward=1
sysctl
   net.ipv4.icmp_echo_ignore_broadcasts=0
```

3.3.4 Log analysis

As we can see from our network captures Tux3 can successfully ping Tux2, which means that the network configuration is correct and the router is doing its job as a *Layer 3* routing device.

This is only possible because Tux3 and Tux2 have routing entries on the table pointing at

Tux4, which serves as a router for this purpose. In this case, Tux3 has a default gateway pointing at *172.16.10.254* and Tux2 has a route for the subnet *172.16.10.0/24* that points at *172.16.11.253* (it's not the default gateway because the default gateway will be set to something else on later experiments). A routing table is fundamental to the networking stack of every computer because it needs to know how to route each packet in order to communicate with other networks (as it happens frequently with the internet). We can set a routing table for a specific subnet as we did on Tux4 or we can configure a *default* route, as we did on Tux3, that if there isn't any other match on the routing table it will send the packet to that default route (which is common to happen in commercial routers). Usually, the forwarding table contains the destination subnet or IP address, and the network interface that will redirect the packet toward its destination but it can also contain some other configuration parameters like the *metric* parameter which defines the priority of the route in case there are multiple matches for that packet.

After deleting the ARP tables for all machines, and issuing a *ping* command from Tux3 to Tux2, we can see that the ARP instead of asking for *172.16.11.1* (which is impossible because ARP only works inside of a subnet) it asks for *172.16.10.254* because the Kernel's networking stack knows that in order to reach any other subnet the packet should go through *172.16.10.254* because of the route that we've configured. Therefore, in Tux3, the destination MAC for the ICMP packets has the Tux4 MAC address and not the Tux2 MAC address.

So, after watching this localized behavior we can generalize for the whole packet path: Tux3 sends an ARP requesting the MAC of Tux4's eth0 IP address, after the Tux4's ARP response, Tux3 can now send the ping request to Tux4 MAC. However after Tux4 processes the packet, it will know that the recipient IP address is not destined for itself, and because the Kernel is enabled to forward packets, will try to re-route the packet to a suitable destination. Because Tux4 doesn't know whose MAC address belongs to *172.16.11.1* and it has a route configured that matches, it will send an ARP request to the subnet *172.16.11.0/24*. Finally, Tux2 answers the ARP request and Tux4 can finally send the ICMP request packet to its recipient. The response is faster, however, because we don't need to repeat the ARP request.

However, the packet has been rewritten once at Tux4, because Tux3 sends the packet to Tux4's MAC address and now Tux4 will need to alter the packet to give it a new and final recipient. This is why routers are slower than switches because there's more processing involved with the rewriting of the packet.

3.4 Experiment 4

3.4.1 Network Architecture

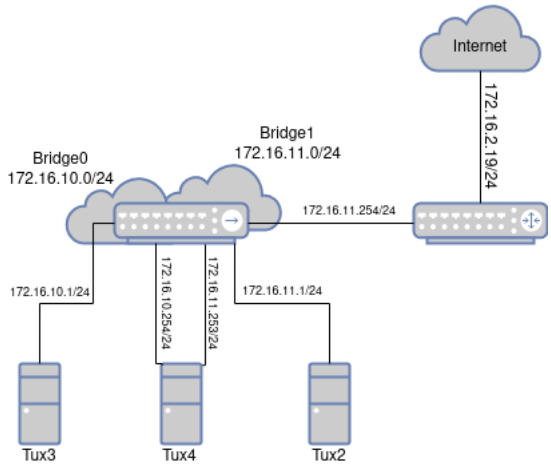

Fig. 5. Experiment 4 network architecture

From the last experiment, we add a *MikroTik* commercial router to the subnet *172.16.1.0/24* or bridge1 and connect the router to the rest of FEUP's network which in turn connects the lab to the internet.

3.4.2 Experiment Objectives

In this experiment, we want to connect the lab computers to the Internet because of that we also need to learn how to NAT works and how to configure it.

3.4.3 Main Configuration Commands

```
#router
/ip address remove 0

/ip address add
    address=172.16.2.19/24
    interface=ether1
/ip address add
    address=172.16.11.254/24
    interface=ether2

/ip route add dst-address=0.0.0.0/0
    gateway=172.16.2.254
```

```
/ip route add
    dst-address=172.16.10.0/24
    gateway=172.16.11.253

/ip firewall nat add chain=srcnat
    action=masquerade
    out-interface=ether1

#switch
add bridge=bridge1 interface=ether3

#tux4
ip route add default dev eth1 via
    172.16.11.254

#tux2
ip route add default dev eth0 via
    172.16.11.254
```

3.4.4 Log analysis

If we delete the subnet *172.16.10.0/24* route from Tux2, we can still *ping* Tux3, which is located on that subnet. However, it can do it at the cost of an extra hop at the commercial route (because the commercial router still has that route and it's the default route of Tux2). We can prove this through our logs (because now if we look at the logs from Tux2 that the destination MAC belongs to the router) and through the *traceroute* command and the path goes from Tux2 →router →Tux4 →Tux3.

If we leave NAT disabled on the router, we cannot ping the lab's router from Tux2 but, if we enable it again we can now ping the lab's router. This is due to the fact NAT is responsible for rewriting the outgoing packets' source IP address to its own "*external*" IP, or WAN IP, and when a response from the outside comes rewriting the ingress packets' destination IP address to the destination machine or route. This is possible since the router keeps a NAT table on its memory that keeps track of every connection made (it stores the source IP address, source port, destination IP address, and destination port). We use destination NAT which means it will only make the NAT table for initial outbound connections. There's also source NAT, also known as port forwarding, that we don't need to use but it's made on a static table that can redirect initial ingress connections.

NAT (Network Address Translation) is important because it can isolate multiple subnets from one another and therefore it allows us to avoid collisions between subnets. And as a consequence, allows us to have many more machines than IPV4 theoretically allows connected to the Internet.

3.5 Experiment 5

3.5.1 Network Architecture

As this experiment doesn't introduce any new node to the evolving network, the architecture remains consistent with that of the previous one.

3.5.2 Experiment Objectives

This experiment aims to enable the pinging of host names from Tux3 by leveraging the Domain Name System (DNS).

3.5.3 Main Configuration Commands

```
#tux3, tux4, tux2
echo 'nameserver 193.136.28.10' >
    /etc/resolv.conf
```

3.5.4 Log Analysis

In this experiment, we use *tux2* to ping *ni.fe.up.pt* since it's not a valid IP address, the system will automatically use DNS (Domain Name System) to query the nameserver of what IP address is *ni.fe.up.pt's* hostname. Note that, in the logs, we use *1.1.1.1*, *Cloudflare's* DNS service, as a nameserver because at the time the Lab's DNS was broken. The DNS lookup starts by discovering which nameservers the host machines use, in Linux, it's usually in the */etc/resolv.conf*. After this, we send two DNS queries to a nameserver of type A or AAAA (IPV4 or IPV6 respectively, and in this case, only the A type will be applied because IPV6 is not configured) and we get two answers from the nameserver containing the hostname's IP address and also the queried hostname.

3.6 Experiment 6

3.6.1 Network Architecture

As this experiment doesn't introduce any new node to the evolving network, the architecture of the network in this experiment is the same as in the previous two.

3.6.2 Experiment Objectives

The goal of this experiment is to validate the utilization of the TCP protocol by employing the FTP client developed in the initial phase of the project to download a file over the network established in the second part of the project.

3.6.3 Main Configuration Commands

There are no configuration commands because the network is identical to experiment 5.

3.6.4 Log Analysis

We can see from the logs two TCP connections are being established to host *193.137.29.15* on port *21* and on port *52839*, one for the FTP control connection and one for the FTP file data (using PASSIVE mode), respectively. We can also prove that by analyzing the data transmitted on the two connections because the first only contains FTP queries and responses.

A TCP connection consists of three phases: a *Connection Establishment* phase, where occurs a 3-way handshake, the client sends an SYN the server responds with an SYN-ACK and the client responds with an ACK (if any of these steps fail the client/server will try to retransmission). After this, we have a *Data transfer* where the applications can freely send packets to each other however, on each packet, we have a *Sequence number*, and that guarantees the reliability of a TCP connection. The *Sequence numbers* play a huge role in TCP retransmission because it can detect if a packet arrives out-of-order or never arrives and automatically ask the other side to retransmit the data. There's also *Acknowledgement numbers* that check if the other side received the sent packet.

The TCP congestion control mechanism is usually activated when a packet is sent 3 times but not acknowledged (it might indicate if a network is congested), and it will send a "TCP Window Update" indicating the new connection congestion window that should be reduced to reduce the effect on congestion, this will control how much data can be sent before an ACK.

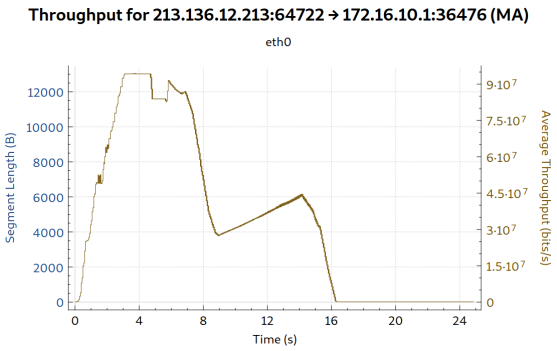

Fig. 6. Experiment 6 throughput on 2 concurrent FTP downloads

We can see the TCP congestion control in action on the throughput graph above initially we have a high throughput but when we initiate another FTP download, the throughput suddenly drops to less than half and slowly creeps up to half (which is near the theoretical throughput of the connection). This makes sense because the router will suddenly become congested due to the second download and will reject some packets of the first connection therefore, the first connection will try to reduce the window size until it stabilizes. When that happens, the TCP congestion control will try to increase the window size slowly until it meets the theoretical maximum.

4 Conclusions

Every objective outlined in the project was successfully accomplished. All experiments were executed with success, and the download application, fulfilling all requirements, is comprehensive, efficient, and functions as intended. Furthermore, the analysis of the logs aligns with and reinforces what was discussed during theoretical classes. Overall, the development of this project, along with the process of writing this report, proved to be a robust hands-on, practical learning experience.

A Appendix

A.1 Code

In folder `download/`.

A.2 Wireshark Captures

In folder `experiments/pcap/`.

A.3 Setup scripts

In folder `experiments/`.