FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Parallel and Distributed Computing – 1st Project Report

Guilherme Almeida, João Malva, Nuno Pereira

BACHELOR IN INFORMATICS AND COMPUTER ENGINEERING

Mestrado Integrado em Engenharia Informática e Computação

March 6, 2023

Contents

0.1 Introduction

The main goal of this project was to set up a small computer network and showcase that it is correctly configured using a "download application", also developed for the purpose of this project.

0.2 Download Application

Part 1 consists in the development of a small application that downloads files using the FTP protocol as described in RFC959. It takes an argument that adopts the *URL* syntax as described in RFC1738. Example:

```
download ftp://ftp.up.pt/pub/hello.txt
```

where the *URL* path is:

```
ftp://[auth@]domain_plus_path

having:
auth = <username>[:<password>]
domain_plus_path = <host>[:<port>]/<path>
```

This allowed us to learn things such as:

- UNIX utilities for handling *URL*s like `getaddrinfo()`;
- Searching, reading and interpreting RFCs, like the one that describes de FTP protocol;
- UNIX utilities to communicate with remote services like sockets;
- how to develop a simple FTP client in C;
- DNS peculiarities when developing a network-driven application;

0.2.1 Architecture of the download application

The download application's flow consists of a series of steps, as follows:

1. Parsing the *URL* string given as the argument to the program to extract the various fragments that make it up, such as the username, password, host, port, and path.
2. Connecting to the supplied host on the specified port (defaulting to 21, the standard FTP control port).
3. Entering passive mode and receiving the data connection's host and port information.
4. Connecting to the data connection's host and port.

5. Signaling that the remote server should start the transfer of the file specified in the *path* portion of the *URL* string.

6. Reading a data buffer from the data connection and writing that to a local file whose name is the same as the one being transferred.

7. Closing the connection.

Network communication is mediated by TCP sockets. Making use of the custom *URL* parser and `getaddrinfo()`, the FTP server's IP and control port are obtained (defaulting to 21 as per the FTP standard).

A TCP connection is established and the program starts sending commands through the control connection. All lines that are/will be sent back by the server are parsed and the given response code is extracted: if the code does not match what is expected, the program terminates with a status code of -1.

The program performs a login against the remote server. If no username is given in the *URL* string then an anonymous login is attempted.

Next, passive mode is entered: in case of success the server responds with **227 Entering Passive Mode (h1,h2,h3,h4,p1,p2)**, where **h1** through **h4** are the host's IP address bytes from highest to lowest and **p1** and **p2** are, respectively, the high byte and low byte of the port to connect to.

After a data connection is established, the `retr` command is sent and afterwards the program starts reading buffers of data from the data connection to a file with the same name as the one being downloaded.

After all these steps, the connection is closed and the program terminates.

0.2.2 Successful download

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

0.3 Network Configuration and Analysis

The main purpose of this part of the project was to setup a small computer network to allow a computer that normally does not have internet access to download a file from a remote server using the FTP client developed in the first part of the project.

This was accomplished by performing a series of small steps leading up the creation of the complete network, which consists of 2 "leaf" computers, 2 sub-networks implemented on a Switch, a 3rd computer serving as a router between both subnets and finally a commercial network router to provide internet access through the lab's own network.

All experiments were made on bench 6 of Lab 2.

0.3.1 Experiment I

0.3.1.1 Network Architecture

At the end of the experiment, the network configuration should consist of tux63 and tux64 connected through the MicroTik switch, as follows:

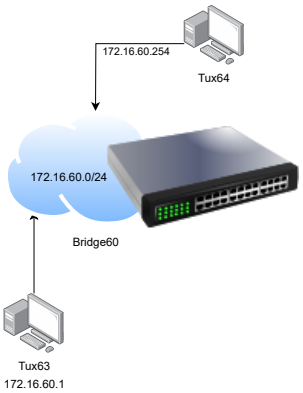

Figure 1: Network configuration for experiment I

0.3.1.2 Experiment Objectives

This experiment had the purpose of teaching how to make 2 separate machines communicate over a network.

0.3.1.3 Main Configuration Commands

```
# tux3:
ifconfig eth1 up
ifconfig eth1 172.16.60.1/24

# tux4:
ifconfig eth0 up
ifconfig eth0 172.16.60.254/24
```

In this experiment, the computers were connected to the switch with no additional configuration needed (using the switch's default bridge).

Note: due to a problem with the bench's tux3 ethernet ports, tux3 had to use the `eth1` interface and have the physical cable connected to the bench's `E2` port. This is the same for the remaining experiments.

0.3.1.4 Log analysis

The corresponding Wireshark log is included as an annex.

In tux3 we issued a ping command to *172.16.60.254*. Since tux3 does not know the MAC address of the machine with this IP address, and as such does not know where to send the ping

requests, it first makes an ARP request to resolve the MAC address corresponding to the pinged IP address.

ARP stands for Address Resolution Protocol, used by computers to associate an IP address with a MAC address in their respective ARP tables, where all associations "IP Address - MAC Address" are stored for future use.

In this case, tux3 , with MAC address *00:22:64:a7:32:ab*, sends a broadcast ARP packet to discover who has IP address *172.16.60.254*; this can be identified in the underlying Ethernet frame that includes the packet, as the "destination" field is *ff:ff:ff:ff:ff:ff*.

Upon receiving this broadcast packet, tux4, with MAC address *00:21:5a:5a:75:bb*, sends a response to tux3 indicating that it has the IP address specified in the ARP request and returning its MAC address, which in turn is stored in tux3's ARP table.

After getting tux4's MAC address, tux3 continuously sends PING requests and tux4 sends PING responses, each containing the previously mentioned MAC and IP addresses.

The type of packet sent in a frame can be differentiated using the 'type' field (bytes 13 and 14): IPv4 corresponds to type `0x0800`, ARP corresponds to type `0x0806`. ICMP packets are stored inside the IPv4 packet so they do not have a specific type code, even though the IPv4 packet itself has a field identifying the format of its data.

The length of each received frame can be calculated on a per-type basis:

- IPv4 packets have a field named Total Length that represents the length of the IPv4 packet and its contents and is stored at bytes 3 and 4 of the packet header. So the total frame length is the packet's total length plus the length of the ethernet frame header.

- ARP packets themselves have a fixed length (in our experiments, and according to several online resources): always 28 bytes long. However, some padding can be added, making the frame length vary between 46 and 64 bytes (Wireshark omits the 4 CRC bytes included in the Ethernet frame and as such the lengths displayed vary between 42 and 60 bytes).

Besides the Ethernet interfaces present in both machines, there is also a "Loop-back" interface, which is a virtual network interface, in other words, it is implemented in software at the kernel level.

The primary purpose of this interface is to check the validity of the installed "network stack", which is necessary for every other type of network communication. This is achieved by sending frames from a machine to itself: if either the frame's destination or source field is a loopback interface, the frame should not leave the current machine, instead being sent back up the network stack.

Other purposes include routing packets from a client (browser, for example) to a server running on the same machine.

0.3.2 Experiment II

0.3.2.1 Network Architecture

At the end of the experiment, the network configuration should consist of tux63 and tux64 connected by a bridge (bridge0) and tux62 connected to a different bridge (bridge1) through the MicroTik switch, as follows:

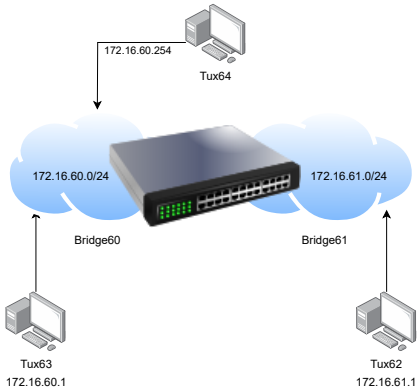

Figure 2: Network configuration for experiment II

0.3.2.2 Experiment Objectives

The purpose of this experiment is to teach us how to setup 2 different network domains on the same switch, and verify that no communication occurs between these domains by default.

0.3.2.3 Main Configuration Commands

```
# These commands are performed after the configuration in experiment I

# tux2
ifconfig eth0 up
ifconfig eth0 172.16.61.1/24

# Switch, connected to through a serial cable
/system reset-configuration

/interface bridge add name=bridge60
/interface bridge add name=bridge61

# assuming tux2, tux3 and tux4 are connected to, respectively, switch
    ports 10, 1 and 2:
/interface bridge port remove [find interface=ether1]
/interface bridge port remove [find interface=ether2]
/interface bridge port remove [find interface=ether10]
/interface bridge port add bridge=bridge60 interface=ether1
```

```
/interface bridge port add bridge=bridge60 interface=ether2
/interface bridge port add bridge=bridge61 interface=ether10
```

0.3.2.4 Log Analysis

The corresponding Wireshark logs are included as annexes.

From the log analysis we can see that tux63 and tux64 can send and receive ping requests/responses from each other, since they are connected to the same bridge on the switch (bridge0).

However tux62 is still unreachable from the 2 other machines. This hypothesis is backed by the capture logs taken from tux62: when pinging from tux63, both tux63 and tux64 show ICMP packets corresponding to PING requests/responses. However, tux62's logs never show any ICMP packets, indicating that this machine is not connected to either one of the other computers. The same thing happens when sending ping broadcasts from tux63 and tux62: tux63 is connected to tux64 and vice-versa, while tux62 is disconnected from the other two.

This also tells us that there are now 2 separate sub-networks operating through the switch. One way to confirm this is to verify the IP address in the 'sender' field in Ethernet frames sent from the 3 computers: tux64 and tux63 belong to the same network (172.16.60.0/24) while tux62 is connected to another network (172.16.61.0/24).

0.3.3 Experiment III

0.3.3.1 Network Architecture

At the end of the experiment, the network configuration should consist of tux63 and tux64 connected by a bridge (bridge0) and tux62 and tux64 connected by a different bridge (bridge1) through the MicroTik switch, as follows:

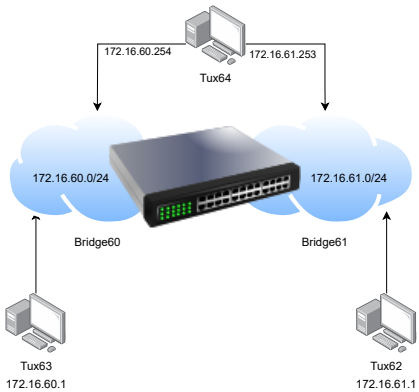

Figure 3: Network configuration for experiment III

0.3.3.2 Experiment Objectives

The purpose of this experiment is to turn tux64 into a router in order to allow communication between tux63 and tux62.

0.3.3.3 Main Configuration Commands

```
# these commands are performed after the configuration in experiment II

# Switch
# these commands assume that tux64 is also wired to socket 11 of the
    switch
/interface bridge port remove [find interface=ether11]
/interface bridge port add bridge=bridge61 interface=ether11

# tux4
ifconfig eth1 up
ifconfig eth1 172.16.61.253/24
echo 1 > /proc/sys/net/ipv4/ip_forward

# tux3
route add -net 172.16.61.0/24 gw 172.16.60.254

# tux2
route add -net 172.16.60.0/24 gw 172.16.61.253
```

0.3.3.4 Log Analysis

The corresponding Wireshark logs are included as annexes.

After connecting tux4 to both sub-networks and configuring routes through tux4 in both tux3 and tux2, these can send PING requests/replies to each other, as can be seen in tux3's logs. This confirms that the network configuration is correct.

If we inspect the routing tables in both tux3 and tux2 we can understand why this happens: tux4 (which is connected to both sub-networks) is the default gateway for both machines to access the sub-network they are not a part of.

After deleting the ARP tables in tux4 and re-running the ping command, the 3 tuxes do not know how to reach each other, as they do not know each other's MAC addresses. Because of this, and because tux3 is pinging a machine on another network, tux3 sends an ARP broadcast to subnet 172.16.60.0/24 requesting the MAC address of its default gateway, in this case tux4. An ARP response is generated and sent to tux3 which in turn stores the MAC address of tux4 in its ARP table for later use (later tux4 does the inverse process to store the MAC address of tux3). Since the ping is going through tux4 into tux2, the same process happens in the second sub network, ending in tux4 storing the MAC addresses of both tux3 and tux2.

After that, routing of ICMP packets occurs normally. It should be noted that these packets have the source field set to the IP address of tux3 and destination field set to the IP address of tux2. However, the underlying frame's destination address (when the packet is sent by either tux3 or tux2) is set to tux4's MAC address. This makes sense on a theoretical level since Ethernet frames

are a Layer 2 data model and as such should be transmitted between connected nodes, where as IP packets are a layer 3 data model and store the transmission endpoints.

0.3.4 Experiment IV

0.3.4.1 Network Architecture

At the end of the experiment, the network configuration should consist of tux63 and tux64 connected by a bridge (bridge0) and tux62, tux64 and the commercial MicroTik router, which is connected to the internet, connected by a different bridge (bridge1) through the MicroTik switch, as follows:

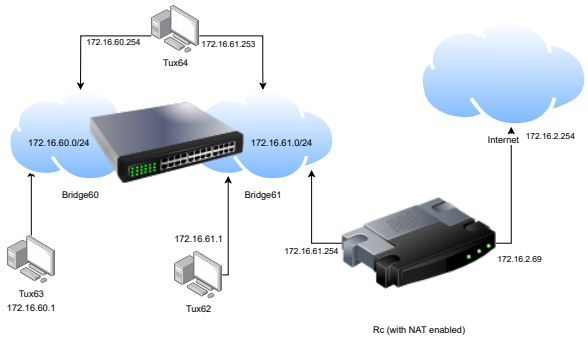

Figure 4: Network configuration for experiment IV

0.3.4.2 Experiment Objectives

The purpose of this experiment is to add internet connectivity to the developed network through a commercial router connected to the labs network.

0.3.4.3 Main Configuration Commands

```
# these steps are performed after the ones described in experiment III

# these steps assume that the router is connected to the switch on socket
    12
# Switch
/interface bridge port remove [find interface=ether12]
/interface bridge port add bridge=bridge61 interface=ether12

# Router
/system reset-configuration
/ip address add address=172.16.2.69/24 interface=ether1
/ip address add address=172.16.61.254/24 interface=ether2
/ip route add dst-address=0.0.0.0/0 gateway=172.16.2.254
/ip route add dst-address=172.16.60.0/24 gateway=172.16.61.253
```

```
# tux4
route add default gw 172.16.61.254

# tux3
route add default gw 172.16.60.254

# tux2
# in the beginning of the experiment
route del default gw 172.16.61.254
# in the middle of the experiment
route add default gw 172.16.61.254
```

0.3.4.4 Log Analysis

The corresponding Wireshark logs are included as annexes.

After deleting tux2's route through tux64, tux62 can only access the 172.16.60.0/24 sub-network through its default router, in this case the bench's router. This can be confirmed by analyzing the capture logs taken from tux62: the IPv4 packets have the addresses of tux62 an tux63 but the Ethernet frames have different MAC addresses, belonging to the router and tux62. Besides this, there were 2 ARP unicast requests made by tux62 and tux64 to resolve the MAC addresses of, respectively, the router and tux62.

All of this tells us that the ping request followed the following path: tux62 → router → tux64 → tux63 → tux64 → tux62.

Afterwards, tux63 made pings to the labs router. These requests succeeded when NAT was enabled on the router and failed when it wasn't. This is due to the fact that the router was responsible for translating addresses outside the sub-networks to valid addresses on the sub-networks using NAT (Network Address Translation).

NAT is the mechanism through which a router machine (commercial router or not, in our case tux64 could be responsible for performing NAT if correctly configured) translates an incoming IP address into an outgoing IP address belonging to a sub-network. This can be used to avoid address collisions with external addresses (from a sub-network's "point of view").

0.3.5 Experiment V

0.3.5.1 Network Architecture

Since this experiment does not add any node to the network being developed, the architecture of the network in this experiment is the same as in the previous one.

0.3.5.2 Experiment Objectives

The purpose of this experiment is to allow us to ping host names from tux63 by taking advantage of a Domain Name System.

0.3.5.3 Main Configuration Commands

```
# these steps are performed after the ones described in experiment IPv4

# tux2
echo 'nameserver 172.16.2.1' > /etc/resolv.conf

# tux3
echo 'nameserver 172.16.2.1' > /etc/resolv.conf
```

0.3.5.4 Log Analysis

The corresponding Wireshark logs are included as annexes.

In this experiment tux62 pinged *google.com*. Since this is not a valid IP address, the computer needs to know what IP address corresponds to this host. To do that, a DNS is used.

DNS stands for Domain Name System and is a mapping from host/domain names and valid IP addresses.

When pinging *google.com*, the computer first performs a DNS lookup to the specified nameserver. This nameserver is defined by adding an entry to the */etc/resolv.conf* file. This file is responsible for specifying up to 3 IP addresses of nameservers, i.e., servers with DNS capabilities.

The DNS lookup generates 2 types of packets: queries and query responses. Both have fields in common but query responses have an additional "Answers" field. These packets transport the number of queries made and the number of typed responses, along with each query/response itself: a query stores each requested host, the length of the host name, and finally the type and class of record. An answer holds the same information plus the requested address and its length in bytes.

Once the IP address of the given host is retrieved, the ping can proceed as usual.

These statements are all corroborated by the Wireshark logs.

0.3.6 Experiment VI

0.3.6.1 Network Architecture

Since this experiment does not add any node to the network being developed, the architecture of the network in this experiment is the same as in the previous two.

0.3.6.2 Experiment Objectives

The purpose of this experiment is to allow us to verify the usage of the TCP protocol when using the FTP client developed for the first part of the project to download a file over the network developed for the second part of the project.

0.3.6.3 Log Analysis

The corresponding Wireshark logs are included as annexes.

Note: these logs refer IP and MAC addresses on bench 4 of lab 2 since that was the one available at the time of this capture. However, these values are not important for this analysis.

From the logs, we can clearly see, in the first packets captured, the configuration of 2 different communication channels to host *192.168.109.136*, one on port 21 (the default port for the FTP control connection) and one on port 47124 (belonging to the data connection in this case).

We can see that the connection open on port 21 is the control connection because of the content of the FTP packets which contain FTP commands sent to the remote server.

This way, its clear that these 2 connections are in their "Connection establishment" phase, because of the TCP handshake that starts with each one (packets 2-4 for the control connection, packets 20-22 for the data connection).

From analyzing the TCP packets, we can see some of the information that they store, such as fields called "Sequence Number" and "Sequence number (raw)". These fields play an important role in TCP's retransmission mechanism, which is one of the motives that makes TCP a reliable Transport Layer protocol. These sequence numbers are used to check the reception order of packets and to figure out errors in the transmission, such as connection timeouts or other factors that render a packet invalid. Besides the sequence numbers, TCP packets also contain an "Acknowledgement Number" and "Acknowledgement Number (raw)" which are TCP's way of accepting a sequenced packet. If it was not accepted/acknowledged, the program automatically retransmits the same packet.

However, a packet might not be sent after 3 retries, which can indicate a connection timeout or that the network is congested. In these cases, a "TCP Window Update" is sent indicating that the connection's congestion window, which in sum control how much data can be sent before receiving an ACK, should be reduced to reduce the congestion effect on the connection.

In addition to that, the throughput graphs show that there was an initial high throughput which suffered a quick drop and then decreased more slowly, until finally getting back up. This makes sense because, following the experiment, the download/capture on tux3 occurred concurrently with a download in tux2, thus making the bench router have to distribute its load between the 2 machines.

0.4 Conclusions

All of the project's goals were achieved - all experiments were executed with success, and the required download application is complete, well designed and working as intended; the analysis of the logs matches and reinforces what was discussed in class on a theoretical level and the development of the project as a whole was a solid practical learning experience.

Member list & Workload

No.	Full name	Student ID	Percentage of work
1	Guilherme Almeida	202006137	33.3%
2	João Malva	xxxxxxx	32.3%
3	Nuno Pereira	202007865	33.3%

.1 Appendix

.1.1 Code

In folder `code/`.

.1.2 Wireshark Captures

In folder `captures/`.

.1.3 Setup scripts

In folder `lab-setup/`.