**RC - Download Application &**

**Configuration of a Computer Network**

by José Castro and Pedro Silva - T14G9

Faculty of Engineering of the University of Porto

1. **Introduction**

There are two goals to this project: the first is to develop a socket-based application that uses the FTP protocol to download a file remotely, and the second is to configure a computer network able to interact with each other and access the internet using 3 different computers, analysing its usage. As a final experiment, a computer in the previously mentioned computer network will use the download application to communicate with the internet and download a file.

This report will feature both the development aspects of each desired goal and an examination of the computer network’s usage throughout the 6 experiments that took place.

1. **Download Application**

The download application is a C language program that utilizes the FTP standard protocol (RFC959) and makes use of the URL syntax (RFC1738) that allows users to download a single file remotely, supporting credential inputting. It uses TCP sockets to communicate with the servers and utilizes the standard FTP control port to achieve this.

Its usage is as follows:

**./download ftp://[<user>:<password>@]<host>/<url-path>**

* 1. **Architecture**

The architecture of the application is based around a series of steps

used in the default flow of the application. They are as follows:

1. Establishment of connection to the server

2. Login and activation of passive mode

3. Connection to the new server-given data port

4. File request and download

5. Finalizing the connection

The flow of the application through these steps is explained as follows:

1. Firstly, it opens a connection to the target server using a ***main*** socket.

2. After establishing connection and receiving an acknowledgement, it sets up for passive data download by logging into the server with the user-given credentials and engaging passive mode. It does this by sending the following FTP commands in order:

1. user <user>
2. pass <password>
3. pasv

3. Subsequently, and if these requests are successful, the server will respond with a sequence of numbers containing the IP address and, finally, two numbers which make up a new port pertaining to the data which will be transferred by the server, and to which the client should connect to. This port will be parsed and connected to using a new ***download*** socket.

4. Before the data download begins, the application opens a new file locally, where the transferred data will be written. Afterwards, the application sends a request through the ***main*** socket to receive the file, using the following command:

retr <url-path>

If the server recognizes this command successfully, it will return a reply specifying the file size of the requested file. This size will be parsed accordingly and used to finalize the data reading when this file size is reached.

Now, the download will now formally begin. Data packets with maximum size of 512 bytes are read by the application from the ***download*** socket until finished.

5. When data transfer is done, both the ***main*** and ***download*** sockets are closed, and the application exits accordingly.

The developed download application is split into 3 main files:

* *main.c*, which englobes the steps of the main flow of the application (estabilishing the initial connection to the server, for example). It is the highest level and least specific layer of the program.
* *connection.c*, which takes care of specific functionalities of the program, (reading a socket reply, for example). This module acts as an API to the *main.c* file and consists mostly of FTP command sending and handling.
* *socket.c,* which is comprised only of socket-related functions (opening a socket, closing a socket, etc.). It is the lowest level layer of the program.
  1. **Download Testing Report**

Numerous tests were made to make sure the download application returned a transferred file correctly independently of the file size. The following files were downloaded:

* *timestamp.txt* (Program Usage: ./download <ftp://ftp.up.pt/pub/kodi/timestamp.txt>)
* *crab.mp4* (Program Usage:<ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4>)

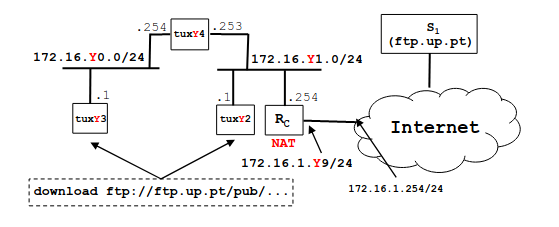
In the Attachments section of this report, both of the downloads’ console logs can be found, boasting displayed server replies and step-by-step prints.

1. **Computer Network Configuration and Analysis**

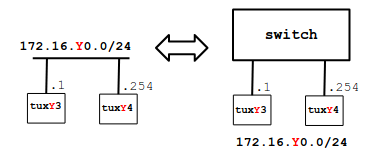
The second part of the project’s goal is to build a local network of computers in which one of them will function as a router linking two computers to the laboratory’s router. With this achieved, it is possible to use the download application concocted for part 1 in any computer in this network.

In each section, we will explain how each experiment’s network was configured and analyse its usage logs, captured using Wireshark. Any letter **Y** in the IP addresses represents the workstation number and differentiates between experiment locales. For our experiments, every letter **Y** was substituted by the number **5.**

The finalized computer network will look like this:



* 1. **Experiment 1 – Configuring an IP**
     1. **Network Architecture**



* + 1. **Experiment Objectives**

The main objective of this experiment is to understand how computers connected to the same network can communicate with each other, and how this connection functions. To achieve this, after setting up our network, we ping the other computer with the ping command to test connectivity.

* + 1. **Main Configuration Commands**

As the image indicates, the only configuration needed was connecting two computers, tuxY3 and tuxY4, to the MikroTik switch using any port.

For our experiment, we configured them using the eth0 ports.

The following commands were issued to make this happen:

**(in tuxY3):**

**>>** ifconfig eth0 up

**>>** ifconfig eth0 172.16.40.1/24

* + 1. **Logs Analysis**
  1. **Experiment 2**

**Network Architecture**

* + 1. **Experiment Objectives**
    2. **Main Configuration Commands**
    3. **Logs Analysis**

After this second experiment, using information available in the previous one and on the logs, we can better understand what happened.

From the first experiment, we already know that tux3 can ping tux4, but the same cannot be said for tux2: it can't ping or be pinged by tux3 or tux4. A natural occurrence, considering we did not connect the machines or the VLANs.

Regarding the broadcast pings, we can conclude that two broadcast domains exist, one where tux3 and tux4 belong and another including tux2 exclusively. This becomes clear when we realise that the broadcast frame sent from tux2 doesn't reach any other tux nor does he receive any one of the other frames, but the one sent from tux3 reaches tux4 and vice-versa.

* 1. **Experiment 3**
     1. **Network Architecture**
     2. **Experiment Objectives**
     3. **Main Configuration Commands**
     4. **Logs Analysis**
  2. **Experiment 4**
     1. **Network Architecture**
     2. **Experiment Objectives**
     3. **Main Configuration Commands**
     4. **Logs Analysis**
  3. **Experiment 5**
     1. **Network Architecture**
     2. **Experiment Objectives**
     3. **Main Configuration Commands**
     4. **Logs Analysis**
  4. **Experiment 5**
     1. **Network Architecture**
     2. **Experiment Objectives**
     3. **Main Configuration Commands**
     4. **Logs Analysis**