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.

2 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>

2.1 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.

2.2 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`)
- *pipe.txt* (Program Usage: `./download ftp://ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt`)
- *crab.mp4* (Program Usage: `ftp://rcom:rcom@netlab1.fe.up.pt/files/crab.mp4`)

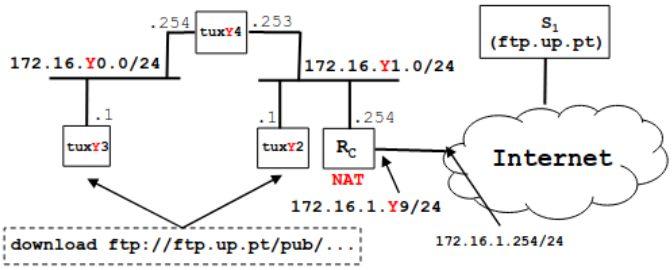
In the Attachments section of this report, a successful download Wireshark log can be found.

3 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.

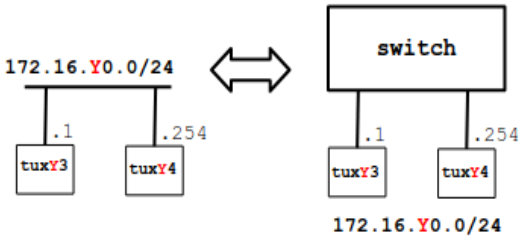
In each section, we will explain how each experiment's network was configured and analyse its usage logs, captured using Wireshark. Any letter Y in the IP addresses represents the workstation number and differentiates between experiment locales. For our experiments, every letter Y was substituted by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

The finalized computer network will look like this:



3.1 Experiment 1 – Configuring an IP Network

3.1.1 Network Architecture



3.1.2 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.

3.1.3 Main Configuration Commands

As the image indicates, the only configuration needed was connecting two computers, tux3 and tux4, to the Cisco switch using any port.
For our experiment, we configured them both using their eth0s and connecting these to ports 2 and 4.
The following commands were issued to make this happen:

(in tux3):

>> ifconfig eth0 up
>> ifconfig eth0 172.16.50.1/24

in (tux4):

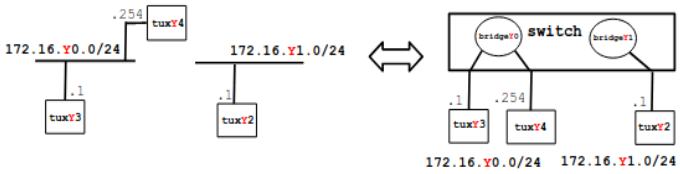
>> ifconfig eth0 up
>> ifconfig eth0 172.16.50.254/24

3.1.4 Logs Analysis

Due to off-schedule lab usage, the workstation used for this experiment was number Y. This change is reflected in the displayed IP addresses retrieved from the logs.
After configuration, we ping tux4 from tux3. Logs show us that both the IP addresses of tux3 and tux4 were setup correctly: 172.16.50.1 for tux3 and 172.16.50.253 for tux4. Tux3's MAC address was 00:21:5a:61:2c:54 and tux4's was 00:22:64:19:09:5c. Since tux3's ARP Table was cleared, the ARP packets sent back and forth aim to convert the unknown IP address of the ping destination (tux4) to a corresponding MAC address. After tux3 sends out an ARP Broadcast Request and receives its acknowledgement and response from tux4, tux4's IP address is saved in tux3's ARP Table for future IP Address – MAC Address conversions, becoming now a known destination in this computer. The numerous ICMP packets use the previously obtained information (IP Addresses and MAC Addresses) to communicate between the two computers.
To determine if a receiving Ethernet frame is ARP or IP frame, we can check the two-octet field inside the ethernet packet (the 12th and 13th bytes). The ARP value of these two bytes is 0x0806 and the IPv4 value is 0x0800. Their packet lengths are calculated as such: for the ARP packets, the request's packet size is the sum of the Ethernet II and ARP layers' size, and the response packet contains the request size with some padding to always reach the minimum frame size needed. The IPv4 packets however are calculated through the two-octet total length field (bytes 2 and 3) of the IPv4 header.

3.2 Experiment 2 – Implement two bridges in a switch

3.2.1 Network Architecture



3.2.2 Experiment Objectives

The main objective of this second experiment is to add a third computer to our network, tux2, and, thus, implementing two bridges in the Mikrotik switch.

3.2.3 Main Configuration Commands

This second experiment builds upon what was already configured previously. In addition to this, we configure a new computer (tux2) with the following command:
(in tux2):

>> ifconfig eth0 up
>> ifconfig eth0 172.16.51.1/24

In the switch's interface, we create two bridges, bridge50 and bridge51, and remove the default bridge (bridge) using the following commands:
>> /interface bridge add name=bridge50
>> /interface bridge add name=bridge51
>> /interface bridge remove bridge

Afterwards, with bridge creation complete, we remove the ports that were being used by the default bridge (using /interface bridge port remove [find interface=etherX]) and add our tux's ports to the two bridges we just created (for our experiment, we used ether3 for tux3, ether7 for tux4, ether2 for tux2) using the following commands:
(in the switch interface):

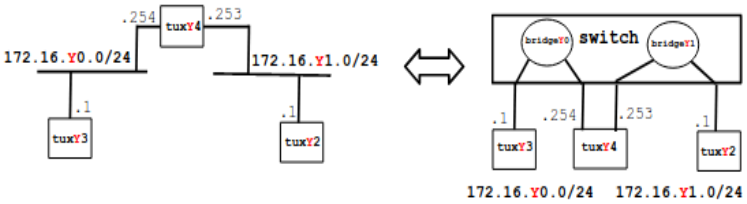
(for bridge50):
>> /interface bridge port add bridge=bridge50 interface=ether3
>> /interface bridge port add bridge=bridge50 interface=ether7
(for bridge51):
>> /interface bridge port add bridge=bridge51 interface=ether2

3.2.4 Logs Analysis

After this second experiment, we can better understand what happened by using information available in the previous one and on the logs,
From the first experiment, we already know that tux3 can ping tux4, but the same cannot be said for tux2: it cannot 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 does not 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.

3.3 Experiment 3 – Configure a Router in Linux

3.3.1 Network Architecture



3.3.2 Experiment Objectives

The main objective of this experiment is to make tux4 a routing mechanism for the network we established in the previous experiments and verifying how it changes connectivity in the network. To reach this goal, we will ping other network interfaces from tux3 and verify if they are reachable.

3.3.3 Main Configuration Commands

Firstly, we need to transform tux4 into a router. To do this, we add tux4's eth1 to a port in the Cisco router, (first removing and then) adding this port to bridge 51 afterwards (in our case, we connected tux4's eth1 to ether3 in the router). Apart from this, we also need to enable IP forwarding so tux4 can route packets coming from bridge 50 and 51, effectively establishing a connection between both bridges. Disabling ICMP echo-ignore-broadcast was also in order. The following commands were used:

(in tux4):

```
>> ifconfig eth1 172.16.51.253/24
>> echo 1 > /proc/sys/net/ipv4/ip_forward
```

(in the switch interface):

```
>> /interface bridge port remove [find
interface=ether8]
>> /interface bridge port add bridge=bridge51
interface=ether8
```

To achieve our objective, we also must make some changes to the configuration of the pre-existing tux3 and tux2 so that they can reach each other using tux4. The following command was used to enable tux2 to connect to tux4:

(in tux2):

```
>> route add -net 172.16.50.0/24 gw
172.16.51.253
```

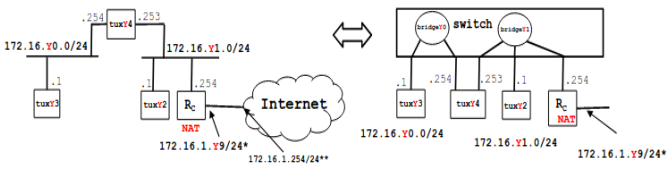
3.3.4 Logs Analysis

This time, tux4 is included in both tux2 and tux3's subnetworks and is configured with IP Forwarding, functioning as a "middleman" between the two other machines by connecting the two subnetworks and routing packets from one to the other.
Pinging all network interfaces from tux3, we observe that every connection is successful. This is due to the default gateway present in both tux2 and tux3 to tux4 eth0 (in tux2) and tux4 eth1 (in tux3). These gateways are used when sending packets to machines outside of the sender's subnetwork, which happens in this experiment. When pinging tux2 from tux3, tux2 doesn't know where tux3 is, since it is outside of its subnetwork (tux2: 172.16.11.0/24; tux3: 172.16.10.1/24), and defaults to sending the packet to tux4 eth0. From here, there is a direct connection to tux2 through tux4's eth1 interface. The reply follows the same process, but in reverse order. We see two pairs of ARP messages in each interface, one when sending the request and another when receiving the reply. For example, in eth0, two ARP messages are sent during the initial ping to connect tux3 to tux4 (Requesting the MAC Address of tux4 and replying) and two when receiving the reply. ICMP packets are sent to control the connection between the IP addresses and looking at the logs we can see the "path" of the connection taking place: first, 172.16.10.1 (tux3) communicates with 172.16.10.254 (tux4 eth0), then with 172.16.11.253 (tux4 eth1) and finally with 172.16.11.1 (tux2). In the reply, tux2 does the same, but in reverse order.

A forwarding table entry contains the network destination, the netmask, the destination gateway and interface and the metric, used when choosing the best route in case of multiple paths.

3.4 Experiment 4 – Configure a Commercial Router & Implement NAT

3.4.1 Network Architecture



3.4.2 Experiment Objectives

As the title of this experiment says, this experiment aims to build a commercial router through NAT implementation.

3.4.3 Main Configuration Commands

To setup the architecture of this experiment's network, we needed to make some changes to the previous configuration. Firstly, we make tux4 the default router for tux3. Secondly, we add a new default gateway in tux2 whose destination is Rc, doing the same for tux4. After this, we add routes for bridge0, 172.15.50.0/24, in tux2 and Rc. We make this happen with the following commands:

(in tux3):

>> route add default gw 172.16.50.254

(in tux2):

>> route add default gw 172.16.51.254

(in tux4):

>> route add default 172.16.51.254

(in the router):

>> /ip address add address=172.16.2.59/24 interface=ether1

>> /ip address add address=172.16.51.254/24 interface=ether2

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

>> /ip route add dst-address=172.16.50.0/24 gateway=172.16.51.253

3.4.4 Logs Analysis

Acessing the internet was possible, both in tux2 and tux3, after finishing the experiment. These two tuxes were already connected to one another due to the previous experiment, only needing to add three gateways: two through 172.16.11.254 (in tux2 and tux4, to Rc) and one through 172.16.11.253 (in tux2, to tux4). Through the same process as in the last experiment, routing defaults to being made through Rc in case of an attempt to access IPs outside of a machine's subnetwork.

Concerning packet paths, with "accept-redirects" disabled and enabled on tux2, they were the following:

disabled: tux2 - Rc - tux4 - tux3

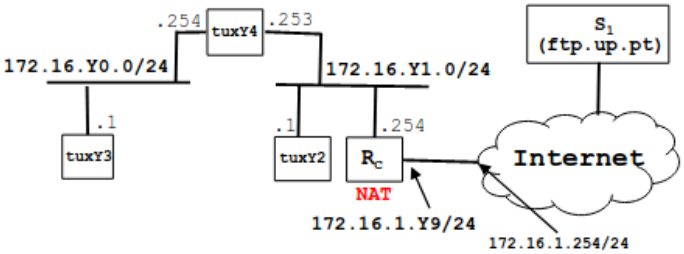
enabled: tux2 - Rc - tux4 - tux3 (first packet), tux2 - tux4 - tux3 (subsequent packets)

This behaviour happens because, with redirecting enabled, after the first packet reaches the Rc, it sends it to tux4 and responds to tux2 saying that the best way to reach tux3 is through tux4. As such, Rc is no longer accessed in subsequent packets.

The Network Address Translation (NAT) is a way of converting a public IP address to a target IP address inside a private network. By doing this, the IP of the destination is kept private, and it allows for multiple machines in a local network to share one IP address publicly.

3.5 Experiment 5 - DNS

3.5.1 Network Architecture



3.5.2 Experiment Objectives

This experiment aims to configure DNS in our network.

3.5.3 Main Configuration Commands

For this experiment, the only configuration needed is the enabling of DNS using *netlab*'s server. We use the following command in **every** tux:

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

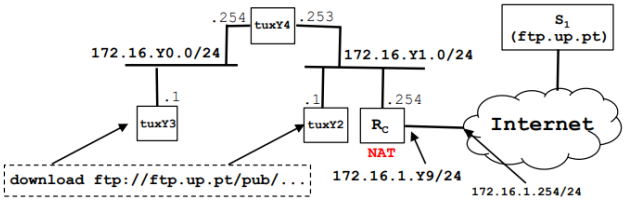
3.5.4 Logs Analysis

In the logs, we can observe two distinct groups of packets. First, a DNS standard query gets sent to 172.16.2.1 (IP address of the router in the lab) with www.google.com as the query. Resolving the DNS yields a standard query response with the translated IP (142.250.184.4), which gets used when exchanging packets with tux3, from and to.

A reverse DNS lookup also happens: translating an IP address to a domain name, in this case, using "in-addr.arpa". In this case, the previous IP with reversed octets (4.184.250.142) and the "in-addr.arpa" added at the end is sent as a DNS standard query to the router's IP address, which returns a standard query response of "mad41s19-in-f4.1e100.net".

3.6 Experiment 6 – TCP Connections

3.6.1 Network Architecture



3.6.2 Experiment Objectives

The final experiment of the project is combining the computer network we have been building and the download application from Part 1 to download a file from FEUP's DNS services.

3.6.3 Main Configuration Commands

No further configuration than what we'd done for experiment 6 is needed.

3.6.4 Logs Analysis

The download application, which uses TCP sockets, opens two TCP connections throughout its lifetime: a main control connection and a data download one. The connection that transports FTP Control information is the is the TCP connection.
TO start the connection, a TCP [SYN] packet is sent by the client to the server, to which the server responds with a TCP [SYN, ACK] frame. With the connection established, commands are sent by the client, with each also receiving a TCP [ACK] packet. After the download starts, the server sends FTP-DATA packets containing the file's data and the client acknowledges them by sending TCP [ACK] packets to the server. Finally, connections are severed when the server sends a TCP [SYN, FIN] packet to the client and receives a TCP [ACK] as response.
The ARQ (Automatic Repeat Request) error-handling mechanism, which is a variation of the Go-Back-N mechanism, is also displayed during the download. When data is not correctly captured, the server sends an FTP-Data packet with a [TCP Previous segment not captured] message. Right after, numerous data packets are sent as the client alerts what frames were missing and acknowledges them when they arrive correctly.
The TCP congestion mechanism is designed to lower throughput when the network is more congested. However, after trying to replicate a situation that would trigger it by starting a download on another tux, we couldn't verify any noticeable change in data transfer, as it always remained steady.

4 Conclusions

The second project was a success, as the download application behaved as expected, with multiple file types and sizes, and error code management. The log analysis also reflected what we learned during the network experiments and theoretical classes. Overall, it was an enjoyable, practical learning experience that harmonized and explained different subjects in a contained and organized manner.