

**2º LAB WORK**

**Autores**:

Guilherme Magalhães

Paula Arias Fernández

Pedro Cerejeira

**Turma**: 3LEIC06

**UC**: RCOM

1. **Summary**

This report explains the network configuration work our group developed in class and a program we developed to establish an FTP connection with a server.

1. **Introduction**

Our project is divided in two parts: a first part concerning a download application, where we explain the code and show a use example and a second part concerning a network in configured in the practical classes, where we mostly reflect on how our work relates to what we learned in the theoretical class. We don’t give a step-by-step guide on how to repeat our experiments, instead we explain what is going on in the network.

1. **Download Application** 
   1. **Architecture Overview**

We divided FTP code in three modules: input-output (.h and .c), connection (.h and .c) and main.c.

* + 1. **Input-Output**

This module handles all input and output to the user’s machine.

The function parseInput parses the ftp instruction written by the user when calling the program. It receives as arguments: input, the input string and info, an output struct wo which the parsed data will be written. Using scanf and simple logic, user, password, file name, file path and host are extracted from the string. A bool indicating whether a username was provided is also saved. If a username is provided, but not a password, the password is filled with “nopassword”. At this point, getIP is called, after which the function flow ends.

The function getIP receives an Info struct as an argument. Gethostbyname is called using the host indicated in the structed and the result, an IP and host name, is written to the struct.

The function writeFile receives as arguments the name of the file to be created and the ID of the socket from which it will read the contents of this file. The file is opened and 1000 bytes are written at a time in a loop, until the socket has no more bytes left. The file is then saved

* + 1. **Connection**

This module handles the connection with the FTP server.

The function openSocket establishes a connection with the server. It receives as arguments the IP address and the port in which the connection will be opened and a pointer to an integer to which the socket ID will be written.

The function writeRequest creates a string request. It receives as arguments two strings: argument and result and a request type integer, depending on the desired type it outputs a request string in the result, possibly using the argument string.

The function sendRequest sends a request to the server. It receives a request string and a socket ID as arguments. Using the function send it simply writes the request to the socket.

The function readResponse reads the server’s response to a request. It receives a pointer to the file where the server response is written as argument. The response is written to the user’s interface and the success/failure of the command sent determined.

The function setupPasv reads the server’s response to a pasv request and sets up a second connection in passive mode, from which an output can be read. It receives a pointer to the file where the server response is written and pointers to two integers as arguments. This means that, after running a similar code to readResponse, it parses the response to identify an IP and a port, which is then outputed to the argument integers.

* + 1. **Main**

This is the main file, that handles the flow of the program and calls other functions as needed. All requests are created by calling writeRequest() to write the desired command, followed by sendRequest() to send it to the server.

The very first thing the program does is call parseInput() to parse the user input and save its contents to the scruct “info”, after which a socket is opened via the openSocket function, in order to communicate with the server.

If the user has provided a username and a password, they are sent in two consecutive requests. If not, then username “ftp” and password “nopassword” are sent instead.

With the connection established the protocol activates passive mode, sending a specific request to the server. Until now responses were processed with the simple readResponse method. But this specific response is processed with setupPasv, that saves the IP and port to which the server will write the ouput. A socket is created connecting the program to this indicated port, using openSocket.

A final command is sent to the original socket: “Retr”. The server then writes the contents of the desired file to the second socket, which is read with writeFile. This marks the end of the program flow.

* 1. **Usage**

To compile our code use “gcc main.c connection.c inpuT-output.c”. We tested our code with files up to about 200 MB of size.

Graphical user interface, text, application

Description automatically generated with medium confidenceGraphical user interface, text

Description automatically generated

Figure 1: Simple Usage example

1. **Configuration and Study of a Network**
   1. **Experience 1**

IP addresses are public addresses that a machine can generate to allow other machines to contact it in a network. MAC addresses are physical identifiers, that a machine simply has and doesn’t control. In order for computer A to send a message to computer B, it needs to know its MAC address. But in most cases only the IP address is available.

The Address Resolution Protocol is used in these instances. First, ARP packets are broadcasted to the entire LAN. This initially ARP packet contains both addresses of the sender and only the IP address of the receiver (otherwise there would be no need to use ARP).

Text, letter

Description automatically generated

Figure 2: ARP IP and MAC addresses

The machines that don’t have the target IP address ignore the request. When it eventually reaches the desired receiver, it replies only to the sender with its MAC address, which can then be added to the ARP table

Text, letter

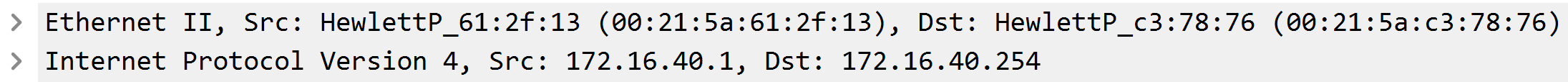
Description automatically generated

Figure 3: ARP reply IP and MAC addresses

The ping command generates ICMP packets. The sender sends a request and, if the receiver receives it, it responds with a reply. These packets contain IP and MAC address of both the receiver and the sender. If the ARP table in the sender doesn’t contain the MAC address of the receiver, an ARP packet will have to be sent first.

Text

Description automatically generated



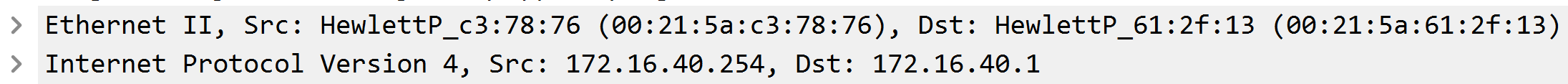


Figure 4: Ping packets and ICMP request and reply addresses (Dst and Src inverted)

In Wireshark, the type and length of a packet are readily available in the details section. The length is simply the sum of its IP packet payload length and the Ethernet specific components: the Ethernet Header, 14 bytes and the Frame Check Sequence Field, 4 bytes. To manually check the packet type, we’d need to look at the Ethernet Header and compare its value to that of the different types of packets.

A loopback interface is a virtual interface that is always reachable if at least one of the IP interfaces on the switch is working. For that reason, a loopback interface is useful for debugging, since its IP address can always be pinged if any other switch interface is up.

* 1. **Experience 2**

The ethernet ports of our computers need to be connected for the machines to communicate with eachother. For that we can create bridges in our switch.

To configure bridgeY0 in a switch we need to first create it by using the command "/interface bridge add name=bridgeY0". Afterwards, the ports where the aimed resources are connected must be removed by using "/interface bridge port remove [find interface =ether{X}]". Finally, we add the now free resource to the corresponding port to the bridge by using "/interface bridge port add bridge=bridgeY0 interface=ether{X}". Obviously, replace {X} with the context specific port identifier.

In our experience there were two broadcast domains. We can verify this by running “ping broadcast” and seeing which computers receive this request. We ran the command in both tuxY3 and tuxY2 and verified if it reached tuxY4. We also tried to ping the machines from different origins and reached the same conclusion.



Figure 5: tuxY4 receiving the ping broadcast from tuxY3, nothing was received from tuxY2

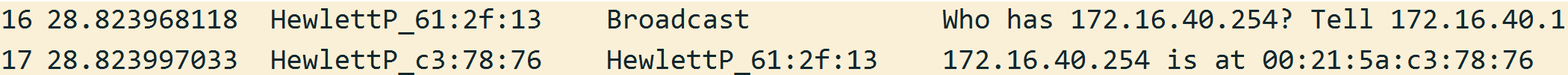
* 1. **Experience 3**

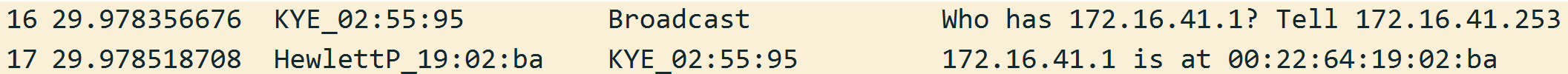
We can use a router to connect machines from different bridges. To do it we only need to add it to different bridges and setup routes in every device.

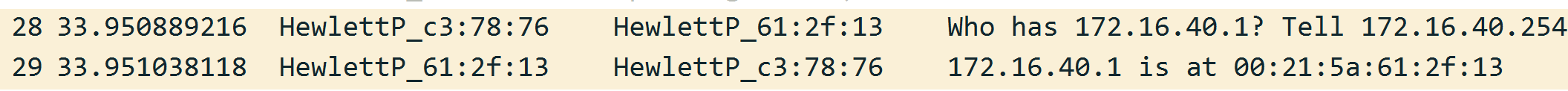
There is no need to configure routes to the bridge a port is connected to. They are represented by the default gateway 0.0.0.0. However, you need to add routes to machines in other bridges with the command “route add -net {target subnetwork} gw {router IP}”. This means that, to send packets to any IP in the target subnetwork, you can send them to the gateway, in this case the router. In tuxY3 we added a route to 172.16.41.0/24 with 172.16.40.254 as the gateway and in tuxY2 we added a route to 172.16.40.0/24 with 172.16.41.253 as the gateway.

The forwarding table contains: the ingress port number, the source MAC address, the destination MAC address, the Ether Type, the VLAN ID, the VLAN priority, the source IP address, the destination IP address, the IP protocol, the IP ToS, the TCP source port, and the TCP destination port. After this table is deleted, everytime a machine needs to send a message to an IP address an ARP packet must be sent.

In our experience, we pinged tuxY2 from tuxY3. In order to explain what we did, I will call the tuxY4 ports connected to bridgeY0 and bridgeY1 tuxY4.0 and tuxY4.1 respectfully. Starting with ARP requests: first, tuxY3 asks for tuxY4.0’s MAC address; followed by tuxY4.1 asking for tuxY2’s address; followed by tuxY2 asking for tuxY4.1’s address until finally tuxY4.0 asks for tuxY3’s MAC address.







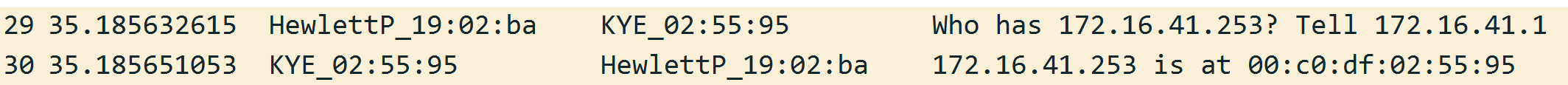


Figure 6: ARP packets captured in tuxY4

After these requests were made, ICMP packets followed. The request packet made its way from tuxY3 to tuxY2, passing through the router and the response packet made the reverse path. They used the IP addresses indicated in their routes and the MAC addresses obtained via ARP.

* 1. **Experience 4**

In order to configure a static route in our commercial routers, we used the command "/ip route add dst-address=[destination] gateway=[gateway]".

In our experience we tested 4 different situations. In the first test, we removed the route in tuxY2 to tuxY3, so a request would need to use the default gateway. When we pinged tuxY3 from tuxY2 it followed the path tuxY2-Router-tuxY4-tuxY3. When we readded the route in the second test, it followed the direct path tuxY2-tuxY4-tuxY3.

Calendar

Description automatically generated with low confidence

Figure 7: results of traceroute

Network Address Translation (NAT) is used to map local private addresses to a public one. If we want to connect our LAN to other networks we need to use NAT, otherwise the target machine has no way of responding to our requests, as they can’t identify the specific machine that made them. To configure the NAT in our commercial router, we ran the command "/ip firewall nat add chain=srcnat action=masquerade out-interface=ether1", with ether1 being the port connected to the outside world. To disable it: “/ip firewall nat disable 0”.

When we pinged the lab router from tuxY3 in our third experiment, the ping followed the path tuxY3-tuxY4-Router-Lab Router. After we disabled NAT in our final test, the ping command failed to get a reply. This is because, even though the lab router receive the ICMP packet, it couldn’t respond since it had no way of knowing the source of the packet.

* 1. **Experience 5**

The DNS service is configured at the file "/etc/resolv.conf". Inside this file only “nameserver [server address]” should be written. DNS uses TCP packets for Zone transfer and UDP packets for name, and queries. UDP packets can be used to exchange small information whereas TCP packets must be used to exchange information larger than 512 bytes.

* 1. **Experience 6**

The FTP protocol uses TCP to connect the server to the client. A TCP connection has three phases: connection establishment, data transfer and connection termination.

The TCP AQR uses sliding window flow control, a forgiving error control method. All packets have a sequence number, an acknowledge number, the window size and other parameters not relevant to the TCP ARQ.



Figure 8: Some TCP parameters

The window size is the size of the sliding window and is dynamically updated, to achieve better efficiency. In the beginning of the connection a process called slow start is used, where the TCP protocol tries to identify a lower bound for the window size. This is done by increasing exponentially the window size until packet loss occurs. Afterwards the connection is maintained more or less stable, with the protocol constantly increasing and reducing throughput to try to reach an optimal spot.

Graphical user interface, application

Description automatically generated

Figure 8: The evolution of a simple FTP connection’s throughput

The throughput of an FTP connection should be reduced when another connection is established, as the resources are divided equally amongst the two connections. We didn’t see this happen in our experiment, so we have to conclude there is a bottleneck somewhere else in the process, perhaps on the server itself.

A picture containing text, boat, water, several

Description automatically generated

Figure 9: Effect of a simultaneous connection in an FTP transfer

1. **Conclusion**

The goal of this project was achieved: we understood both how the FTP protocol and communication with sockets work in part 1 and learned some general network configuration in part 2. The hands on experience was particularly useful for understanding how the ARP protocol works.

1. **Annexes**

FTP Code <https://github.com/guilhermewastaken/RC02/tree/main/Download>

Logs captured <https://github.com/guilhermewastaken/RC02/tree/main/Logs>