Laboratory Project 2

Computer Networks

Development of a download application and configuration of a computer network

Class 13

Bernardo Ferreira Campos – up202006056

Ioan-Mircea Radu - up202202727

Summary

This report describes the work that was made during the past semester in the context of the second laboratory project of the Computer Networks curricular unit of the Informatics and Computing Engineering course.

The project consists of the development of a download application, and the configuration and study of a computer network while conducting a series of experiments.

The following sections will go into a more in-depth analysis on the architecture of the download application as well as an analysis of the each of the six experiments, part of the network configuration task.

Contents

Introduction

This project, the focus of our laboratorial classes during the second part of the semester, consists of two main tasks:

At home, we were assigned with the task of developing a simple FTP client in C that should be able to download a file from the university server. This application was partially based on already written programs that were made available to us.

The second task was performed in the laboratories, mainly during classes. In it we were assigned with six experiments, each part of the process of configuration of a computer network. After each experiment we were given a set of questions to ponder upon, to help us get a better grasp of the concepts. Answers to these questions are included in the report.

As a final experiment we were tasked with configurating the network and testing our download application in it. Due to issues with the laboratory computers, we were not able to complete this task as supposed to, but still proved the correct steps on the configuration were followed and that the application was able to fulfil its purpose.

Download application

As already described, one of this project tasks were the development of a download application in C following the FTP (file transfer protocol) as described in RFC959 and adopting the URL syntax described in RFC1738.

Architecture

The download application is divided into two parts.

The first part is responsible for parsing the URL that is passed as argument to the program by the user. We must consider that there are two possible formats for this argument: one of them only containing the host and the URL path to the file, corresponding to the anonymous mode; and another one containing those two arguments in addition to a username and password, corresponding to the normal mode.

The application also has the capacity of throwing an error message if, by mistake, the URL input by the user is not valid in the context of this application.

Parsing this input will allow us to store in proper variables the URL path, the host and if necessary, the username and password.

After this part is concluded, there is an intermediate step, were the function *gethostbyname* is used to retrieve the IP of the host.

Moving on to the second part of the application, responsible for creating the necessary connections to the server through sockets, so we can download the requested file.

The application will start this part off by creating a socket and connecting it to the server using the address we retrieve by using *gethostbyname.*

The next step is sending a sequence of commands to the server: *“user ‘username’\r\n”*, the *“pass ‘password’\r\n”* and the *“pasv\r\n”* commands. These will log the user in the server and

set the server to passive mode. If the URL passed in the beginning of execution corresponds to anonymous mode, ‘anonymous’ and ‘password’ are going to be, respectively, the username and password.

After this step, the server responds with a string containing the server IP address and the server IP port for file transfer. And once again, the program will parse a string, this time to retrieve the IP and port.

The application will create another socket and connect to the server using the address we got in the previous step. Now, to ask the server for the desired file the following command is sent: *“retr ‘path_to_file’\r\n”*, ‘path_to_file’ corresponding to the URL path that was stored into a variable in the first part of the program.

This should allow the start of the file transfer. If the server does not correspond accordingly an error is thrown. Otherwise, a file will be created locally, and the data being read from the server will be written in it, being this the process that actually corresponds to transferring the file.

A vulnerability of this part was pointed out by our teacher. A while loop is used to constantly read from the server, until the reading fails, indicating that there is nothing more to be read and the transfer is concluded. However, this might be a problem if, during that process, for any reason, the connection is corrupted, causing the read to fail. Once the connection is reestablished, instead of resuming the transfer, the program will assume that there was nothing more to be read and interpret the transfer as concluded even though it is not. The correct way to go about this was to check until a message of complete transfer was sent by the server, and only then the program would stop reading.

The concluding step is to simply close both sockets in order to finish the program.

Download Results

The application worked as expected. It is capable of downloading files both in anonymous mode and with the use of login credentials.

Network configuration and analysis

The second task for this project, performed in the laboratory, consisting in configurating a computer network while performing six experiments. The analysis of each of the experiments can be found bellow.

Experiment 1 – Configure an IP Network

The objective for this experiment is to configure two different machines, tux43 and tux44, and establish a connection between them.

To do this, we started by setting the IPs for these machines using the *“ifconfig ip/mask”* command. The IPs used were the ones presented in the guide for this project. The following step was establishing a route between them using the *“route add -net destination/mask gw end_gateway”* command.

With this done, we can use the *“ping ‘destination’”* command to verify if the connection was established successfully.

The Address Resolution Protocol is used to map IP addresses to MAC addresses. When trying to send a packet to a machine in the same network, the sending machine will first send a

broadcast message to all the machines in the network with an ARP package containing the destination IP. The machine with the destination IP will respond, and the association between the IP and MAC addresses will be stored in the ARP table.

The ping command generates Internet Control Message Protocol packets.

To determine the type of the Ethernet frame, the frame header must be analyzed. IP packets contain information about their length, which can be seen using Wireshark.

A loopback interface is a virtual interface used for diagnosis. It allows for the computer to communicate with itself and check for the quality of the connection and the system.

Experiment 2 – Implement two bridges in a switch

The objective for this experiment was to implement two bridges, one containing tux43 and tux44, and the other containing only tux42. This was done through the switch console.

The procedure, considering that the switch is already reset, starts by creating the two bridges necessary to the experiment, bridge40 and bridge41. This was done with the *"/interface bridge add name=bridge40"* command, and similarly done for bridge41.

After this, we removed the ports we were going to use from the bridges. We needed to remove 3 ports, one for each machine, corresponding to the ports we chose to use in the physical configuration. This was done by using the command *"/interface bridge port remove [find interface =etherX]"*, being X the number of a specific port.

The last step for the configuration is to add each port to the corresponding bridge. The ports chosen for tux43 and tux44 should be added to bridge40, and the port for tux43 should be added to bridge41.

By doing some pings between the machines, we can conclude that, as expected, tux43 and tux44 can communicate with each other, since they are connected to the same bridge, but tux42 is isolated, so it is not possible to communicate with that machine without establishing a connection.

There are two broadcast addresses, one for each bridge.

Experiment 3 – Configure a Router in Linux

The objective for this experiment was to configure a router in Linux to enable communication between tux42 and the other machines. We will be doing this by turning tux44 into a router connecting the two bridges created in the previous experiment.

To do this we started by configuring the eth1 port in tux44 with an IP belonging to the same set as tux42, and adding this port to the bridge41, that contains tux42. This was done using the same procedures of IP configuration and addition of ports to bridges presented in the previous experiments. It was also necessary to enable IP forwarding and disabling ICMP echo-ignore-broadcast, which was done with the following commands:
"echo 1 > /proc/sys/net/ipv4/ip_forward" and
"echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts", both executed in tux44.

The next step was setting up the routes in tux43 and tux42, using the command *"route add -net destination/mask gw end_gateway"*. The 'end_gateway' address corresponds to the IP belonging to tux44 that shares a bridge with the respective computer.

In this case, the command ran in tux43 would look like this: *"route add -net 172.16.41.0/24 gw 172.16.40.254"*, meaning that whenever tux43 tries to make contact with an IP belonging to '172.16.41.0/24' set (tux42), that contact will be redirected to '172.16.40.254', the IP of tux44 that belongs to bridge 40, same as tux43. A similar process is followed in tux42.

It is now possible to ping tux43 to tux42 and vice-versa. As already mentioned, tux44 will work as a router, so the pings sent will be redirected to tux44 and only then sent to their actual destination.

The forwarding table will contain information on the destination IP address, the gateway to where the contact will be sent and then redirected to the destination, and the interface (e.g., eth0).

Whenever there is communication between two machines that still "have not met" an ARP message, containing the destination IP address is broadcast, and the machine corresponding to that IP address will respond with another ARP message containing its MAC address.

Since the routes connecting all the machines are set, the ICMP packets sent are of type request and reply.

Experiment 4 – Configure a Commercial Router and implement NAT

The objective for this experiment was the configuration of a commercial router using NAT.

The router was configured through the router console using the following commands: *"/ip address add address=172.16.1.49/24 interface=ether1"* and *"/ip address add address=172.16.41.254/24 interface=ether2"*. NAT is enabled by default.

To setup the routes these were the commands used:
"/ip route add dst-address=172.16.40.0/24 gateway=172.16.41.253"
"/ip route add dst-address=0.0.0.0/0 gateway=172.16.2.254"

Network Address Translation is a way to translate multiple local private addresses to a public one, in order to make possible for our machines to communicate with external networks. This is necessary because IP addresses are a limited resource, so they cannot be freely recognized publicly.

Experiment 5 – DNS

The objective for this experiment is to add our machines into a DNS.

This was done by editing the */etc/resolv.conf* file, at each machine, using the given DNS server.

When contacting an external server, a DNS package is sent asking for the server's IP. The server responds with the requested information.

Experiment 6 – TCP

The objective for this final experiment was to run and study the FTP client we developed in our configured network. No further configuration is necessary for this task.

Two TCP connections will be open by the client. One will be used for authentication and preparing for the transfer, and the second one is responsible for actually transferring the file. So, the first connection is the one used to transport control information.

A TCP connection will have three phases: establishing the connection, trading information,

and closing the connection.

The ARQ TCP mechanism works by using acknowledgment messages and timeouts. Whenever a frame is received and acknowledgment message will be sent, if there is a timeout before the acknowledgement is received, the sending of frames will proceed, and in the ending the frames that did not get received (an acknowledgment message was not sent) will be resent.

The appearance of a second TCP connection will impact the throughput of an already existing TCP connection, that will drop. The TCP connections will be assigned a similar packet transmission rate.

Conclusions

This project allowed us to grasp many concepts on how computer networks and a number of associated protocols and technologies associated with it work. Both the development of the FTP client and the configuration of the network were completed successfully, so we believe its safe to say the project as a whole was a success as well, despite a problem during the presentation that we interpret as a factor out of our control.

References

The main reference during the development of this project was practical guide