

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

FTP Application and Computer Network Configuration

Computer Networks, Project 2

Class 9 Group 1

Bruno MENDES up201906166@edu.fe.up.pt

José COSTA up201907216@edu.fe.up.pt

Friday 28th January, 2022

Contents

1	Summary	1
2	Part 1 - FTP Application	2
2.1	Introduction	2
2.2	The FTP protocol	2
2.2.1	Messages	2
2.2.2	File retrieval	2
2.3	Program modules	2
2.4	Compilation and usage	3
2.5	Conclusion	3
3	Part 2 - Computer Network Configuration	4
3.1	Introduction	4
3.2	Guide 1 - IP Config (Lab)	4
3.2.1	Experiment analysis	5
	What are the ARP packets and what are they used for?	5
	What are the MAC and IP addresses of ARP packets and why?	5
	What packets does the ping command generate?	5
	What are the MAC and IP addresses of the ping packets?	5
	How to determine if a receiving Ethernet frame is ARP, IP, ICMP?	5
	How to determine the length of a receiving frame?	5
	What is the loopback interface and why is it important?	6
3.3	Guide 2 - Virtual LANs (Lab)	6
3.3.1	Experiment analysis	7
	How to configure vlanY0?	7
	How many broadcast domains are there? How can you conclude it from the logs?	7
3.4	Guide 3 - Router Configuration (Remote)	7
3.4.1	Experiment analysis	7
	How to configure a static route in a commercial router?	7
	How to configure NAT in a commercial router?	7
	What does NAT do?	8
	How to configure the DNS service at an host?	8
	What packets are exchanged by DNS and what information is transported?	8
	What ICMP packets are observed and why?	8
3.5	Guide 4 - Router Configuration (Lab)	8
3.5.1	Experiment analysis	10
	What routes are there in the tuxes? What are their meaning?	10
3.6	Conclusion	11

4	Conclusion	12
5	Annex: source code	13
5.1	<i>application.c</i>	13
5.2	<i>message.h</i>	17
5.3	<i>message.c</i>	17
5.4	<i>network.h</i>	19
5.5	<i>network.c</i>	19
5.6	<i>states.h</i>	21
5.7	<i>states.c</i>	21
5.8	<i>url_parser.h</i>	25
5.9	<i>url_parser.c</i>	26
5.10	<i>utils.h</i>	28
5.11	<i>utils.c</i>	28

Chapter 1

Summary

This project is comprised of two parts, with each one regarding a different layer of the computer communication stack. The first part focuses on the application layer, making use of the FTP protocol to build a download application, capable of retrieving files from remote locations. The second part aims for the configuration of a local network, taking advantage of VLANs, NAT and the hardware available at FEUP's NetLab. The use of this technologies introduced us to the Network Layer's set of functionalities.

Chapter 2

Part 1 - FTP Application

2.1 Introduction

The goal of the application is to download a file from an arbitrary FTP server, supporting user login. The implementation revolves around the use of sockets to communicate with the server, relying on the TCP Protocol, and being compatible with both IPv4 and IPv6.

2.2 The FTP protocol

The *File Transfer Protocol* is used for the transfer of files over a TCP connection.

2.2.1 Messages

The communication between the FTP server and the client is done via human-readable messages, following the structure:

```
<code> <description>
```

A program may interpret only the code, the description being left to the human debugger. For complete information about all message details, including multi-line syntax (which is handled by our application, too), please refer to the [RFC](#).

2.2.2 File retrieval

To retrieve a file, one of the two occurs:

- The server connects to the server: **active mode**.
- The client connects to the server: **passive mode**.

In both cases, there is a separate connection for the control flow and the file data transfer. Our program is using the passive mode, which means that it will open the file data connect itself.

2.3 Program modules

Application Executes the flow of the program, being responsible for opening the sockets and performing tasks like logging in in the correct order.

Network The lower-level model, responsible for socket connection details.

URL Parser The URL parser, interpreting the hostname, the file path, the port and the login details using POSIX regular expressions.

Message The message interpreter, for tasks like retrieving the code and parsing the passive mode command reply.

States Controls answer-reply flows for the required tasks at the server, including error handling.

Utils Includes functions for common tasks such as string manipulation.

2.4 Compilation and usage

You can compile the program running *make* at the project root. Usage is as follows:

```
./download ftp://[[user][:password]@]<host>[:port]/<url-to-file>
```

The application will connect to the server, log in, get the file size, set up passive mode, open a new connection to receive the file and close both connections after finishing. A progress bar is shown to illustrate the transfer progress. Any failure will terminate the program with a descriptive error message.

2.5 Conclusion

The developed implementation allows for a reliable file transfer from any FTP-reliant server, with decent average speeds, above 1KB/s. The program performs correctly on the network we configured on the lab from tux3.

Chapter 3

Part 2 - Computer Network Configuration

3.1 Introduction

The objective of this part is to configure a local network, using the hardware available at FEUP's Network Laboratory (NetLab). The group was assigned to workbench 1 of the room I321. This information will influence the IP addresses used for the hosts and Routers configured throughout the following guides.

At each bench there are 3 computers (each named *tux n*, being n the number of the computer), a switch and a router.

Although the first part was developed individually, this second part was done in collaboration with David Preda (up201904726). It is highly likely that one may find similarities between the conclusions taken by each group. This was done with the approval of the practical classes teacher.

3.2 Guide 1 - IP Config (Lab)

The objective of this guide is connecting two computers (tux3 and tux4) using a network switch as a middleman. To do so we needed to assign both computers IP addresses.

In order to watch the Address Resolution Protocol (*ARP*) in action, the *ARP* table of one of the computers was cleared and a ping targeting the peer computer was issued.

The tux4 computer was assigned the IP 172.16.10.254 using the following commands:

```
ip link set up eth0
ip addr flush dev eth0
ip addr add 172.16.10.254/24 dev eth0
```

The tux3 was configured with the IP 172.16.10.1 using the commands below:

```
ip link set up eth0
ip addr flush dev eth0
ip addr add 172.16.10.1/24 dev eth0
ip -family inet neigh flush any
```

3.2.1 Experiment analysis

What are the ARP packets and what are they used for?

ARP, or Address Resolution Protocol, is a protocol used for translating IP addresses into MAC addresses.

ARP packets can be of two types: request and reply. Request packets are first broadcasted by the machine wishing to identify the machine behind a given IP. When the IP owner receives the request, it replies with its MAC address, uniquely identifying itself to the first machine.

What are the MAC and IP addresses of ARP packets and why?

For the request packet, the sender MAC and IP addresses are the ones of the broadcaster. The target IP is the IP of the host to discover and the target MAC field is filled with zeros, the special MAC address used for broadcasting. This MAC address is used given that the machine wishing to have an IP translated has no means of knowing to whom it has to transmit the ARP packet.

As for the reply packet, the target addresses are from the machine who made the initial request, and the sender addresses are from the machine who uses the IP to be translated.

What packets does the ping command generate?

The ping command generates ICMP (Internet Control Message Protocol) packets.

What are the MAC and IP addresses of the ping packets?

The ping command was issued from Tux3 to Tux4. Therefore, the source addresses are the ones of Tux3:

- IP address: 172.16.10.1
- MAC address: 00:21:5a:61:2d:ef

The destination addresses are the ones of Tux4:

- IP address: 172.16.10.254
- MAC address: 00:21:5a:61:2f:24

It is worth noting that the IPs are the ones configured for this experiment.

How to determine if a receiving Ethernet frame is ARP, IP, ICMP?

Through the bytes 12 and 13, we can identify, in an Ethernet Frame, whether the frame is ARP or IP. Furthermore, in an IP packet, through the byte 23 of the Ethernet Frame, we can identify if it is ICMP.

However, this can be more easily done through Wireshark, which indicates, by itself, the type of each packet.

How to determine the length of a receiving frame?

Analyzing the logs on Wireshark, one can verify the length of a frame the Frame Length field.

What is the loopback interface and why is it important?

To loopback means to route a signal from the source to itself, without modifications. A loopback interface is an interface through which a device can communicate with itself while bypassing the local network interface. An example of such on Linux is the *localhost*, which is a virtual loopback interface that allows network communication within the same machine.

3.3 Guide 2 - Virtual LANs (Lab)

The second lab revolved around configuring the network switch to accommodate two separate VLANs. The first Virtual Local Area Network (VLAN) will connect the computers configured in the first guide, while the second VLAN will have a single computer associated to it (tux2), also to be configured during this guide.

To test the lack of connection between two separate VLANs using the same hardware tux3 pinged tux4 but failed to ping tux2. ping broadcast in the 172.16.10.0 network (using *ping -b 172.16.10.255*, the broadcast address), issued by tux3, reached tux4 but not tux2. Any tries to ping another computer from tux2, either directly or using broadcast in its VLAN, were unsuccessful.

To configure the tux2 computer the following commands were issued:

```
ip link set up eth0
ip addr flush dev eth0
ip addr add 172.16.11.1/24 dev eth0
```

The necessary configuration to the switch was done using following these commands:

enable

```
configure terminal
vlan 10
end
```

```
configure terminal
interface fastethernet 0/13
switchport mode access
switchport access vlan 10
end
```

```
configure terminal
interface fastethernet 0/14
switchport mode access
switchport access vlan 10
end
```

```
configure terminal
vlan 11
end
```

```
configure terminal
interface fastethernet 0/12
switchport mode access
switchport access vlan 11
end
```

3.3.1 Experiment analysis

How to configure vlanY0?

The following explanation covers the assignment of static-access port to a VLAN, on a Cisco router. One should have already set the privileged EXEC mode before starting.

Firstly, the *configure terminal* should be used to enter the global configuration mode. Afterwards, the interface to be added to the VLAN should be specified through the *interface <interface-id>* command. The third step is to define the VLAN membership mode for the port. In this case, the mode should be *access*. Therefore, the command *switchport mode access* is used. After this, one must assign the port to the desired VLAN, through the command *switchport access vlan <vlan-id>*. Finally, the *end* command is used to return to the privileged EXEC mode.

How many broadcast domains are there? How can you conclude it from the logs?

There are two broadcast domains in the configured network. Through the logs, by broadcasting on Tux2, we can verify that no other machine receives its pings. However, when broadcasting on Tux3, it is easily verifiable that there exists another broadcast domain, as Tux4 receives Tux3's pings. Given that this does not happen when broadcasting on Tux3, we can conclude that Tux3 and Tux4 are on the same broadcast domain, and that Tux2 is on a separate one. As there are no other hosts in the network, we can conclude that there are only 2 broadcast domains.

This falls in line with the concept of VLAN, creating segregated networks using hardware in common.

3.4 Guide 3 - Router Configuration (Remote)

Due to the regulations in place, this lab was done remotely, without access to FEUP's Net-Lab.

The objectives of the guide were threefold: study Cisco routers' configuration, test DNS entries and explore route configurations on personal machines.

3.4.1 Experiment analysis

How to configure a static route in a commercial router?

For a Cisco commercial router, a static route is configured through the *ip route* command. It is worth noting that, since it is a static route, if the topology of the network changes, the configured route should be updated as well.

How to configure NAT in a commercial router?

For a Cisco commercial router, NAT is configured through a series of commands.

For each interface, IP addresses may be configured through the *ip address <IP address> <Address mask>*. Furthermore, the *ip nat inside* defines an interface as a NAT inside interface and the *ip nat outside* as a NAT outside interface. The definition of internal networks is pretty straightforward with these commands.

NAT can be further configured by making pools of addresses with the *ip nat pool <pool-name> <first-address> <last-address> <prefix> <prefix-length>* command. These pools may then be used to indicate that certain packets from the inside should have their source address translated to an address of the pool. This last configuration is achieved through the *ip nat inside source list <list-number> pool <pool-name>* command.

What does NAT do?

Network Address Translation, or NAT, is a method used to map several IP addresses of a network into a single one, making this single address the only one visible to machines outside the network. The main advantages of NAT is that it allows for a better usage of the amount of IP address available (which was and still is an issue with IPv4) and it allows to hide the IP addresses of the machines in a network behind a public IP address.

How to configure the DNS service at an host?

In a Linux host, it suffices to edit the `/etc/resolv.conf` file. To add a DNS, a line with the word *nameserver* followed by the DNS IP address should be added. Up to 3 nameservers can be added to the configuration file.

What packets are exchanged by DNS and what information is transported?

DNS exchanges IPv4 packets. Inside its packets are a multitude of information, which depends on the type of packet. Along with general information, if it is a DNS request, it contains the query to be made to the server; if it is a response, it contains the answers to the query, which, by itself, contains the IP address of the name requested, the time to live or even the data length.

What ICMP packets are observed and why?

The ICMP packets observed indicate that the time-to-live has exceeded. In the context of this experience, this happens because, by deleting the default gateway, there is no route to the addresses we wish to contact. This also means that incoming packets aren't properly received as they, too, lack a route to their destination.

3.5 Guide 4 - Router Configuration (Lab)

This Guide completes the previous guide, given that access to FEUP's NetLab was granted.

This time the guide helps to configure a linux router and an cisco router. *tux4* will serve as a router, routing traffic between the VLANs set in guide 2. To do so, in addition to the already configured *eth0* interface, the *eth1* interface will be configured, adding it to the same VLAN as *tux2*. It will have the static IP address of 172.16.11.253

```
ip link set up eth1
ip addr flush dev eth1
ip addr add 172.16.11.253/24 dev eth1
```

It was also necessary to configure the switch, to add the newly created connection to *tux2*'s VLAN.

```
configure terminal
interface fastethernet 0/24
switchport mode access
switchport access vlan 11
end
```

To make *tux2* and *tux3* use *tux4*'s interface in their respective VLANs, we needed to add run the following commands:

In *tux2*:

```
ip route add 172.16.10.0/24 via 172.16.11.253
```

In *tux3*:

```
ip route add 172.16.11.0/24 via 172.16.11.254
```

Now, from *tux3*, pinging all other network interfaces is a success, unlike guide 2.

Now, the only thing left to do is to configure the cisco router to allow for internet connection. After checking that the router's *GE0* interface is connected to the internet and the *GE1* interface is connected to the switch, the switch must be configured in order to add the port connected to the router's *GE1* to the same VLAN as *tux2*.

The configuration loaded to the router is as follows:

```
hostname router1

interface GigabitEthernet0/0
ip address 172.16.1.19 255.255.255.0
ip nat outside
no shutdown

interface GigabitEthernet0/1
ip address 172.16.11.254 255.255.255.0
ip nat inside
no shutdown

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.10.0 255.255.255.0 172.16.11.253

ip nat pool ovrld 172.16.1.19 172.16.1.19 prefix-length 24
ip nat inside source list 1 pool ovrld overload

access-list 1 permit 172.16.10.0 0.0.0.7
access-list 1 permit 172.16.11.0 0.0.0.7
```

This configuration also sets up the Network Address Translation (NAT), to using overloading, this is, the hosts in the local network communicate with the outside using the router's IP. The router is then responsible for assigning different ports for each of the hosts requests, forwarding those requests, receiving the response, translating it to the original request port, and forwarding it to the host in the local network.

To add the router to *tux2*'s VLAN, we added in the switch the following configuration:

```
configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan 11
end
```

All the suggested pings were successful. Communication with the tuxes, the lab router and the Internet is established.

To allow other hosts to connect to the internet, routes needed to be added to *tux2* and *tux4*.

In both computers:

```
ip route add default via 172.16.11.254
```

Pinging from *tux3* both the lab router and the Internet were accessible.

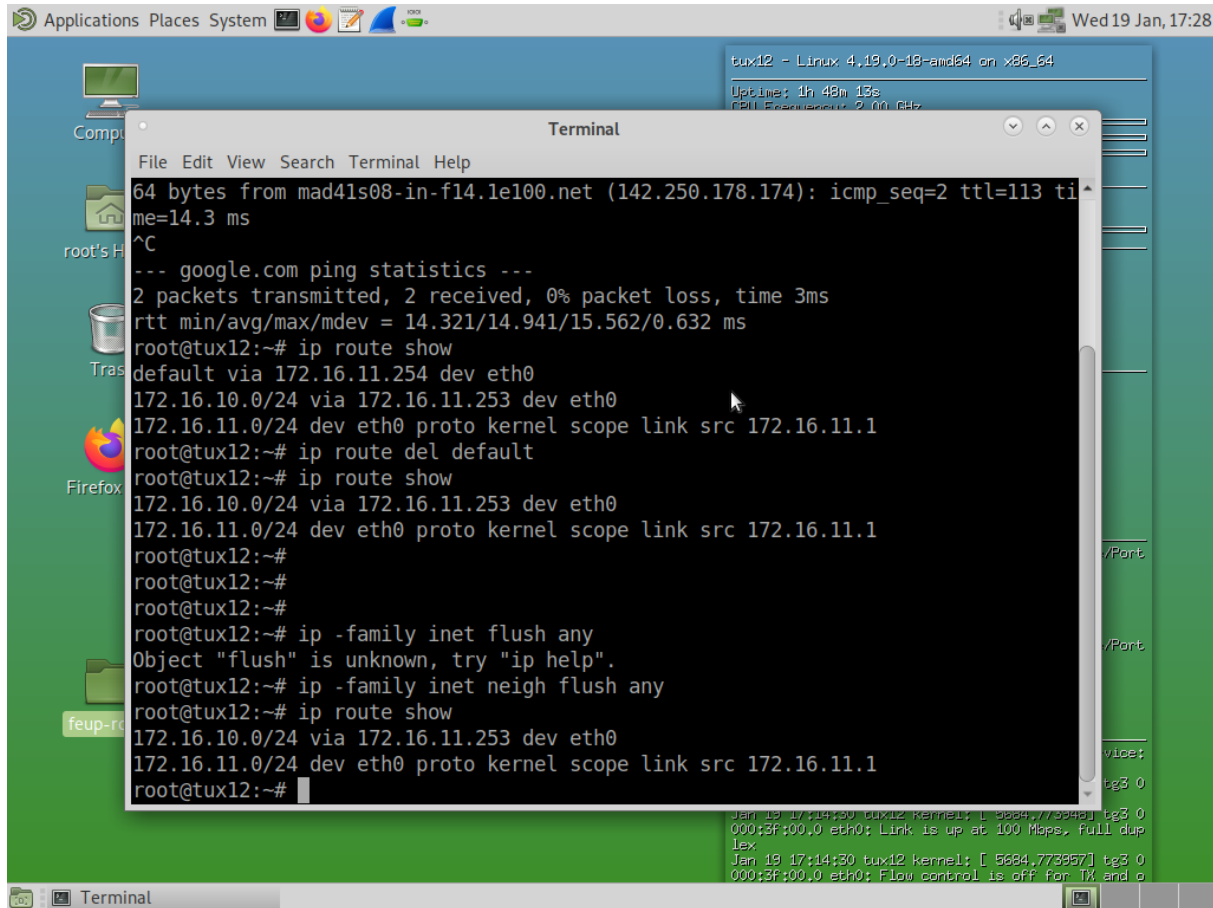
We reached the end of this guide.

3.5.1 Experiment analysis

What routes are there in the tuxes? What are their meaning?

The routes present in the screenshots below are mostly explained in the guides above.

Routes in *tux2*:



The screenshot shows a Linux desktop with a terminal window open. The terminal displays the following commands and output:

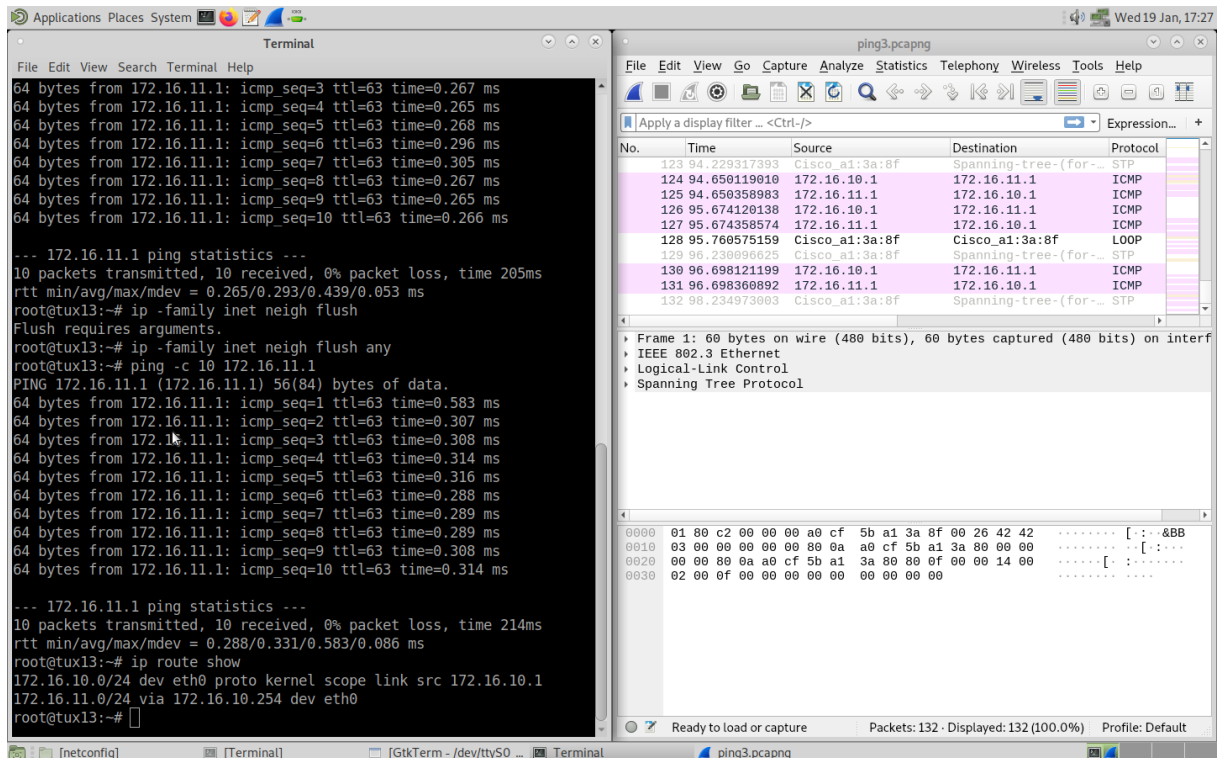
```
tux12 - Linux 4.19.0-18-amd64 on x86_64
Uptime: 1h 48m 13s
CPU Frequency: 2.00 GHz

64 bytes from mad41s08-in-f14.1e100.net (142.250.178.174): icmp_seq=2 ttl=113 time=14.3 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 3ms
rtt min/avg/max/mdev = 14.321/14.941/15.562/0.632 ms
root@tux12:~# ip route show
default via 172.16.11.254 dev eth0
172.16.10.0/24 via 172.16.11.253 dev eth0
172.16.11.0/24 dev eth0 proto kernel scope link src 172.16.11.1
root@tux12:~# ip route del default
root@tux12:~# ip route show
172.16.10.0/24 via 172.16.11.253 dev eth0
172.16.11.0/24 dev eth0 proto kernel scope link src 172.16.11.1
root@tux12:~#
root@tux12:~#
root@tux12:~# ip -family inet flush any
Object "flush" is unknown, try "ip help".
root@tux12:~# ip -family inet neigh flush any
root@tux12:~# ip route show
172.16.10.0/24 via 172.16.11.253 dev eth0
172.16.11.0/24 dev eth0 proto kernel scope link src 172.16.11.1
root@tux12:~#
```

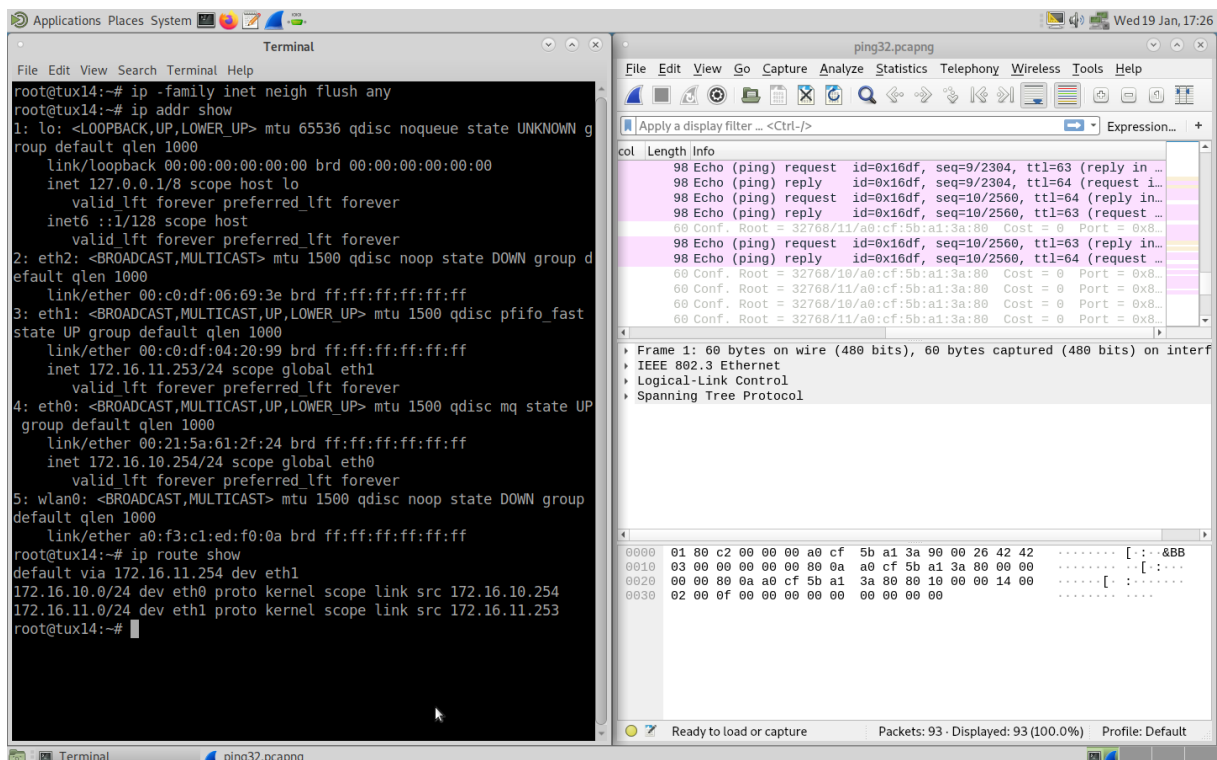
At the bottom of the terminal window, there is a log of kernel messages:

```
Jan 19 17:14:30 tux12 kernel: [ 5684.773948] tg3 0:00:3f:00:00:00 eth0: Link is up at 100 Mbps, Full duplex
Jan 19 17:14:30 tux12 kernel: [ 5684.773957] tg3 0:00:3f:00:00:00 eth0: Flow control is off for TX and o
```

Routes in *tux3*:



Routes in *tux4*:



3.6 Conclusion

This is the first time that any of the group's elements tried to configure a network. This brought us closer to the physical aspect of the backbones in which we usually rely to develop applications. It was, with no doubt, different from what we are used to because it added the physical component to the software heavy syllabus of the course.

Chapter 4

Conclusion

The project was completed successfully, allowing for the transfer of a file using our FTP download application, on top of a manually configured network. It was a great opportunity to learn about the full network stack, mainly the application and network layers.

Chapter 5

Annex: source code

5.1 *application.c*

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>

#include "message.h"
#include "network.h"
#include "states.h"
#include "url_parser.h"
#include "utils.h"

#ifndef PATH_MAX
#define PATH_MAX 4096
#endif

int ctrl_socket_fd = -1;
int data_socket_fd = -1;
bool close_data_file_required = true;
int data_file_fd = -1;

static void print_usage(char *app_name) {
    fprintf(stdout,
            "Usage: %s ftp://[[user][:password]@]<host>[:port]/<url-to-file>\n",
            app_name);
}

static void close_ctrl_socket_fd() {
    close_socket(ctrl_socket_fd);
}

static void close_data_socket_fd() {
    close_socket(data_socket_fd);
}

static void close_data_file_fd() {
    if (close_data_file_required) {
        if (close(data_file_fd) == -1) {
            perror("Failed to close data file FD");
        }
    }
}
```



```

}

static void close_delete_data_file(char *data_file_path) {
    if (close(data_file_fd) == -1) {
        perror("Failed to close data file FD");
        return;
    }
    if (unlink(data_file_path) == -1) {
        perror("Removing created file");
        return;
    }
    close_data_file_required = false;
}

void get_resource_name(char *server_resource_path, int file_name_size,
                      char *file_name) {
    strncpy(file_name, strrchr(server_resource_path, '/') + 1, file_name_size);
}

int create_open_data_file(char *file_name, int file_path_size,
                          char *file_path) {
    /* Create and open file for incoming data*/
    char file_path_[PATH_MAX];
    snprintf(file_path_, PATH_MAX, ":%s",
             ((int)(strlen(file_name, PATH_MAX))), file_name);

    for (int n = 1;; n++) {
        if (access(file_path_, F_OK) == 0) {
            snprintf(file_path_, PATH_MAX, ":%d-%s", n,
                     ((int)(strlen(file_name, PATH_MAX))), file_name);
        } else {
            if (n > 1) {
                strncpy(file_name, strrchr(file_path_, '/') + 1, PATH_MAX / 4);
                fprintf(stdout,
                        "File name already exists, creating new file: %s\n",
                        file_name);
            }
            break;
        }
    }

    if (file_path != NULL) {
        strncpy(file_path, file_path_, file_path_size);
    }

    int fd = -1;
    /* Open new file with received file name */
    if ((fd = open(file_path_, O_WRONLY | O_CREAT | O_TRUNC,
                  S_IRWXU | S_IRWXG | S_IRWXO)) == -1) {
        perror("Open file for writing");
        return -1;
    }
    return fd;
}

int main(int argc, char *argv[]) {
    setbuf(stdout, NULL);

    /* Parse url info */
    if (argc != 2 || validate_ftp_url(argv[1]) != 0) {
        print_usage(argv[0]);
        return -1;
    }
}

```

```

struct con_info con_info;
if (parse_url_con_info(argv[argc - 1], &con_info) != 0) {
    print_usage(argv[0]);
    return -1;
}
if (strlen(con_info.resource, sizeof con_info.resource) < 1) {
    printf("No file specified\n");
    return -1;
}

/* Open data control socket */
int ctrl_socket_family = -1;
if ((ctrl_socket_fd = open_connect_socket(con_info.addr, con_info.port,
                                         &ctrl_socket_family)) == -1) {
    return -1;
}

/* Register atexit function related to control socket FD */
if (atexit(close_ctrl_socket_fd) != 0) {
    fprintf(stderr, "Cannot register close_ctrl_socket_fd to run atexit\n");
    if (close(ctrl_socket_fd) == -1) {
        perror("Close control socket FD");
    }
    return -1;
}

/* Login in the server */
printf("Logging into %s:%s with %s\n", con_info.addr, con_info.port,
       con_info.user);
int ret = 0;
if ((ret = login(ctrl_socket_fd, con_info.user, con_info.pass)) != 0) {
    if (ret == -2) {
        fprintf(stderr, "Unrecognized user or password credentials\n");
    } else if (ret == -3) {
        fprintf(stderr, "Unsupported login with this application\n");
    } else {
        fprintf(stderr, "Error logging in\n");
    }
    return -1;
}

/* Assert regular file type and retrieve size */
size_t size = 0;
if ((ret = get_file_size(ctrl_socket_fd, con_info.resource, &size)) < 0) {
    if (ret == -2) {
        fprintf(stderr, "Path not found in the server\n");
    } else if (ret == -3) {
        fprintf(stderr, "Error: requested a directory\n");
    } else {
        fprintf(stderr, "Could not retrieve file size\n");
    }
    return -1;
}
printf("%s found in the server\n", con_info.resource + 1);

/* Parse file name */
char file_name[PATH_MAX];
get_resource_name(con_info.resource, PATH_MAX, file_name);
printf("Downloading: %s, %ld bytes\n", file_name, size);

/* Set passive mode and open data socket */
char pasv_addr[MAX_ADDRESS_SIZE];
if ((ret = set_pasv_mode(ctrl_socket_fd, ctrl_socket_family, pasv_addr)) ==

```

```

        -1) {
            return -1;
        }
    bool is_ipv4 = (ret == 0);
    if (is_ipv4) {
        char buf[MAX_ADDRESS_SIZE];
        char ip[MAX_URL_LENGTH];
        char port[MAX_PORT_LENGTH];
        strncpy(buf, pasv_addr, MAX_ADDRESS_SIZE);
        snprintf(ip, sizeof ip, "%s", strtok(buf, ":"));
        snprintf(port, sizeof port, "%s", strtok(NULL, ":"));
        if ((data_socket_fd = open_connect_socket(ip, port, NULL)) == -1) {
            return -1;
        }
    } else {
        if ((data_socket_fd =
            open_connect_socket(con_info.addr, pasv_addr, NULL)) == -1) {
            return -1;
        }
    }

    /* Register atexit function related to data socket FD */
    if (atexit(close_data_socket_fd) != 0) {
        fprintf(stderr, "Cannot register close_data_socket_fd to run atexit\n");
        if (close(data_socket_fd) == -1) {
            perror("Close data socket FD");
        }
        return -1;
    }

    /* Create and open file for incoming data */
    char file_path[PATH_MAX];
    if ((data_file_fd =
        create_open_data_file(file_name, PATH_MAX, file_path)) == -1) {
        return -1;
    }
    if (atexit(close_data_file_fd) != 0) {
        fprintf(stderr, "Cannot register close_data_file_fd to run atexit\n");
        close_delete_data_file(file_path);
        return -1;
    }

    /* Initiate transfer */
    if (init_retrieve(ctrl_socket_fd, con_info.resource) == -1) {
        return -1;
    }

    /* Read from socket and write to file */
    struct timespec start_time, end_time;
    if (clock_gettime(CLOCK_MONOTONIC, &start_time) == -1) {
        perror("Clock get start time");
    }
    if (transfer_data(data_socket_fd, data_file_fd, size) == -1) {
        return -1;
    }
    if (clock_gettime(CLOCK_MONOTONIC, &end_time) == -1) {
        perror("Clock get end time");
    }

    /* End transfer */
    if (end_retrieve(ctrl_socket_fd) == -1) {
        return -1;
    }
}

```

```

/* Logout */
if (logout(ctrl_socket_fd) == -1) {
    fprintf(stderr, "Error logging out\n");
    return -1;
}

struct stat st;
if (stat(file_path, &st) == -1) {
    perror("Stat");
} else {
    double elapsed_secs = elapsed_seconds(&start_time, &end_time);
    double kbs = ((double)st.st_size / 1000) / elapsed_secs;
    printf("Elapsed time: %.2fs\n", elapsed_secs);
    printf("Average speed: %.2fKB/s\n", kbs);
}

return 0;
}

```

5.2 *message.h*

```

#pragma once

#include <stdbool.h>
#include <stdlib.h>

#define MAX_ADDRESS_SIZE 30

int ftp_code(char *msg);

bool is_end_reply(char *msg);

bool stat_reply_not_found(char *msg);

bool stat_reply_is_dir(char *msg);

size_t stat_reply_size(char *msg);

int parse_pasv_reply(char *msg, char *out_reply);

void parse_epsv_reply(char *msg, char *out_reply);

```

5.3 *message.c*

```

#include <ctype.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "message.h"
#include "network.h"

int ftp_code(char *msg) {
    char code[4] = "-1";
    int pos = 0;
    for (char *curr = msg; pos < 4 && *curr != '\0'; curr++) {
        if (!isdigit(*curr)) {
            break;
        }
    }
}

```

```

        code[pos++] = *curr;
    }
    code[pos] = '\0';
    return atoi(code);
}

bool is_end_reply(char *msg) {
    bool new_line = true;
    bool found_space = false;
    for (char *curr = msg; *curr != '\0'; curr++) {
        if (*curr == '\n') {
            new_line = true;
            continue;
        }
        if (!isdigit(*curr) && new_line) {
            new_line = false;
            found_space = (*curr == ' ');
        }
    }
    return found_space;
}

bool stat_reply_not_found(char *msg) {
    char msg_token_buf[MAX_BULK_DATA_MSG_SIZE];
    strncpy(msg_token_buf, msg, sizeof msg_token_buf);
    strtok(msg_token_buf, "\n");
    char *line = strtok(NULL, "\n");
    return (line != NULL && line[0] == '2');
}

bool stat_reply_is_dir(char *msg) {
    char msg_token_buf[MAX_BULK_DATA_MSG_SIZE];
    strncpy(msg_token_buf, msg, sizeof msg_token_buf);
    strtok(msg_token_buf, "\n");
    char *line = strtok(NULL, "\n");
    return (line != NULL && line[0] == 'd');
}

size_t stat_reply_size(char *msg) {
    char msg_token_buf[MAX_BULK_DATA_MSG_SIZE];
    strncpy(msg_token_buf, msg, sizeof msg_token_buf);
    strtok(msg_token_buf, "\n");
    char line_token_buf[MAX_BULK_DATA_MSG_SIZE];
    strncpy(line_token_buf, strtok(NULL, "\n"), sizeof line_token_buf);
    strtok(line_token_buf, " ");
    for (int i = 0; i < 3; i++) {
        strtok(NULL, " ");
    }
    char *size_token = strtok(NULL, " ");
    return strtoul(size_token, NULL, 10);
}

int parse_pasv_reply(char *msg, char *out_reply) {
    char buf[MAX_CTRL_MSG_SIZE];
    strncpy(buf, msg, sizeof buf);
    char ip[16];
    int a = -1;
    int b = -1;
    strtok(buf, "(, )");
    strncpy(ip, strtok(NULL, "(, )"), 3);
    for (int i = 0; i < 3; i++) {
        strncat(ip, ".", 2);
        strncat(ip, strtok(NULL, "(, )"), 3);
    }
}

```

```

    }
    a = atoi(strtok(NULL, "(,)"));
    b = atoi(strtok(NULL, "(,)"));
    if (a < 0 || b < 0) {
        return -1;
    }
    a = 256 * a + b;
    snprintf(out_reply, MAX_ADDRESS_SIZE, "%s:%d", ip, a);
    return 0;
}

void parse_epsv_reply(char *msg, char *out_reply) {
    char buf[MAX_CTRL_MSG_SIZE];
    strncpy(buf, msg, sizeof buf);
    strtok(buf, "(|)");
    snprintf(out_reply, MAX_ADDRESS_SIZE, "%s", strtok(NULL, "(|)"));
}

```

5.4 *network.h*

```

#pragma once

#include <stdbool.h>

#ifndef MAX_CTRL_MSG_SIZE
#define MAX_CTRL_MSG_SIZE 4096
#endif

#ifndef MAX_BULK_DATA_MSG_SIZE
#define MAX_BULK_DATA_MSG_SIZE 256000 // Around same as kernel
#endif

int open_connect_socket(char *addr, char *port, int *ai_family);

int close_socket(int fd);

int send_msg(int socket_fd, char *msg);

int receive_data(int socket_fd, int buf_size, char *buf, bool add_terminator);

```

5.5 *network.c*

```

#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#include "network.h"

int open_connect_socket(char *addr, char *port, int *ai_family) {
    struct addrinfo hints;
    struct addrinfo *addrinfo;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    int status;

    if ((status = getaddrinfo(addr, port, &hints, &addrinfo)) != 0) {
        freeaddrinfo(addrinfo);
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(status));
    }
}

```

```

        return -1;
    }

    int fd;
    if ((fd = socket(addrinfo->ai_family, addrinfo->ai_socktype,
                    addrinfo->ai_protocol)) == -1) {
        freeaddrinfo(addrinfo);
        perror("socket");
        return -1;
    }

    if (connect(fd, addrinfo->ai_addr, addrinfo->ai_addrlen) == -1) {
        freeaddrinfo(addrinfo);
        perror("connect");
        if (close(fd) == -1) {
            fprintf(stderr, "Error closing fd\n");
        }
        return -1;
    }

    if (ai_family != NULL) {
        *ai_family = addrinfo->ai_family;
    }

    freeaddrinfo(addrinfo);
    return fd;
}

int close_socket(int fd) {
    if (shutdown(fd, SHUT_RDWR) == -1) {
        perror("Shutdown");
        return -1;
    }
    if (close(fd) == -1) {
        perror("Close");
    }
    return 0;
}

int send_msg(int socket_fd, char *msg) {
    int no_bytes = 0;
    int size = strlen(msg, MAX_CTRL_MSG_SIZE);
    if (msg[size - 1] != '\n') {
        msg[size++] = '\n';
    }
    while (no_bytes != size) {
        int s;
        if ((s = send(socket_fd, msg + no_bytes, size - no_bytes, 0)) == -1) {
            perror("send");
            return -1;
        }
        no_bytes += s;
    }
    return no_bytes;
}

int receive_data(int socket_fd, int buf_size, char *buf, bool add_terminator) {
    int no_bytes;

    if (add_terminator) {
        if ((no_bytes = recv(socket_fd, buf, buf_size - 1, 0)) == -1) {
            perror("recv");
        }
    }
}

```

```

        buf[no_bytes] = '\0';
    } else {
        if ((no_bytes = recv(socket_fd, buf, buf_size, 0)) == -1) {
            perror("recv");
        }
    }

    return no_bytes;
}

```

5.6 *states.h*

```

#pragma once

#include <aio.h>

int login(int ctrl_socket_fd, char *user, char *pass);

int get_file_size(int ctrl_socket_fd, char *path, size_t *size);

int set_pasv_mode(int ctrl_socket_fd, int ctr_socket_family, char *pasv_addr);

int init_retrieve(int ctr_socket_fd, char *path);

int end_retrieve(int ctrl_socket_fd);

int transfer_data(int data_socket_fd, int data_file_fd, size_t size);

int logout(int ctrl_socket_fd);

```

5.7 *states.c*

```

#include <netdb.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <sys/param.h>
#include <unistd.h>

#include "message.h"
#include "network.h"
#include "states.h"
#include "utils.h"

int read_reply_code(int ctrl_socket_fd, char *msg) {
    // Read the current message and get the reply code
    for (int tries = 0; tries < 3;) {
        if (receive_data(ctrl_socket_fd, MAX_CTRL_MSG_SIZE, msg, true) != -1) {
            if (is_end_reply(msg)) {
                return ftp_code(msg);
            }
        } else {
            tries++;
        }
    }
    return -1;
}

int read_msg(int ctrl_socket_fd, int msg_size, char *msg) {
    int total_read_bytes = 0;
    int no_bytes = 0;

```



```

char buf[MAX_CTRL_MSG_SIZE];

for (int tries = 0; tries < 3;) {
    no_bytes = 0;
    if ((msg_size - total_read_bytes - 1) <= 0) {
        return 0;
    }

    int max_len = MIN(msg_size - total_read_bytes - 1, sizeof buf);
    if ((no_bytes = receive_data(ctrl_socket_fd, max_len, buf, true)) ==
        -1) {
        tries++;
        continue;
    }

    strncat(msg, buf, msg_size - total_read_bytes - 1);
    total_read_bytes += no_bytes;
    if (is_end_reply(msg)) {
        return ftp_code(msg);
    }
}

return -1;
}

int login(int ctrl_socket_fd, char *user, char *pass) {
    char msg[MAX_CTRL_MSG_SIZE];
    int reply_code = -1;

    for (;;) {
        reply_code = -1;
        if ((reply_code = read_reply_code(ctrl_socket_fd, msg)) == -1) {
            return -1;
        }
        switch (reply_code) {
            case 220: // Service ready for new user.
                snprintf(msg, sizeof msg, "user %s\n", user);
                break;
            case 331: // User name okay, need password.
                snprintf(msg, sizeof msg, "pass %s\n", pass);
                break;
            case 230: return 0; // Logged in
            case 500: // Syntax error
            case 501: // Syntax error in parameters or arguments
                return -2;
            case 332: // Need account to log in
                return -3;
            case 530: // Not logged in
            case 120: // Service ready in nnn minutes
            case 421: // Service not available, closing control connection.
            default: return -1;
        }
        if (send_msg(ctrl_socket_fd, msg) == -1) {
            return -1;
        }
    }
}

int get_file_size(int ctrl_socket_fd, char *path, size_t *size) {
    char msg[MAX_BULK_DATA_MSG_SIZE];
    int reply_code = -1;

    for (;;) {

```

```

    snprintf(msg, sizeof msg, "stat %s\n", path);
    if (send_msg(ctrl_socket_fd, msg) == -1) {
        return -1;
    }
    msg[0] = '\0';
    if ((reply_code = read_msg(ctrl_socket_fd, sizeof msg, msg)) == -1) {
        return -1;
    }
    switch (reply_code) {
        case 212: // Directory status.
        case 213: // File status.
            if (stat_reply_not_found(msg)) {
                return -2;
            }
            if (stat_reply_is_dir(msg)) {
                return -3;
            }
            *size = stat_reply_size(msg);
            return 0;
        case 450: // Requested file action not taken.
            break;
        case 211: // System status, or system help reply.
        case 500: // Syntax error, command unrecognized. This may include
                // errors such as command line too long.
        case 501: // Syntax error in parameters or arguments.
        case 502: // Command not implemented.
        case 421: // Service not available, closing control connection.
        case 530: // Not logged in.
        default: return -1;
    }
}
}

int set_pasv_mode(int ctrl_socket_fd, int ai_family, char *pasv_addr) {
    char msg[MAX_CTRL_MSG_SIZE];
    int reply_code = -1;

    if (ai_family == AF_INET) {
        snprintf(msg, sizeof msg, "pasv\n");
    } else {
        snprintf(msg, sizeof msg, "epsv\n");
    }
    if (send_msg(ctrl_socket_fd, msg) == -1) {
        return -1;
    }

    if ((reply_code = read_reply_code(ctrl_socket_fd, msg)) == -1) {
        return -1;
    }
    switch (reply_code) {
        case 227: // Entering Passive Mode (h1,h2,h3,h4,p1,p2).
            parse_pasv_reply(msg, pasv_addr);
            return 0;
        case 229: // Entering Extended Passive Mode (|||port|).
            parse_epsv_reply(msg, pasv_addr);
            return 1;
        case 500: // Syntax error
        case 501: // Syntax error in parameters or arguments
        case 502: // Command not implemented
        case 530: // Not logged in
        case 421: // Service not available, closing control connection.
        default: return -1;
    }
}

```

```

}

int init_retrieve(int ctrl_socket_fd, char *path) {
    char msg[MAX_CTRL_MSG_SIZE];
    int reply_code = -1;

    snprintf(msg, sizeof msg, "retr %s\n", path);
    if (send_msg(ctrl_socket_fd, msg) == -1) {
        return -1;
    }

    if ((reply_code = read_reply_code(ctrl_socket_fd, msg)) == -1) {
        return -1;
    }
    switch (reply_code) {
        case 150: // File status okay; about to open data connection.
        case 125: // Data connection already open; transfer starting.
            return 0;
        case 110: // Restart marker reply.
        case 425: // Can't open data connection.
        case 426: // Connection closed; transfer aborted.
        case 530: // Not logged in.
        case 451: // Requested action aborted: local error in processing.
        case 450: // Requested file action not taken. File unavailable
            // (e.g., file busy).
        case 550: // Requested action not taken. File unavailable (e.g.,
            // file not found, no access).
        case 500: // Syntax error, command unrecognized. This may include
            // errors such as command line too long.
        case 501: // Syntax error in parameters or arguments.
        case 421: // Service not available, closing
            // control connection.
        default: return -1;
    }
}

int end_retrieve(int ctrl_socket_fd) {
    char msg[MAX_CTRL_MSG_SIZE];
    int reply_code = -1;

    if ((reply_code = read_reply_code(ctrl_socket_fd, msg)) == -1) {
        return -1;
    }
    switch (reply_code) {
        case 226: // Closing data connection. Requested file action
            // successful (for example, file transfer or file abort)
        case 250: // Requested file action okay, completed.
            return 0;
        case 150: // File status okay; about to open data connection.
        case 125: // Data connection already open; transfer starting.
        case 110: // Restart marker reply.
        case 425: // Can't open data connection.
        case 426: // Connection closed; transfer aborted.
        case 530: // Not logged in.
        case 451: // Requested action aborted: local error in processing.
        case 450: // Requested file action not taken. File unavailable
            // (e.g., file busy).
        case 550: // Requested action not taken. File unavailable (e.g.,
            // file not found, no access).
        case 500: // Syntax error, command unrecognized. This may include
            // errors such as command line too long.
        case 501: // Syntax error in parameters or arguments.
        case 421: // Service not available, closing control connection.
    }
}

```

```

        default: return -1;
    }
}

int transfer_data(int data_socket_fd, int data_file_fd, size_t size) {
    char buf[MAX_BULK_DATA_MSG_SIZE];
    size_t total_bytes_read = 0;
    int bytes_read = -1;
    int no_bytes_written = -1;
    int total_bytes_written = 0;

    for (;;) {
        total_bytes_written = 0;
        bytes_read = -1;
        no_bytes_written = -1;

        if ((bytes_read = receive_data(data_socket_fd, MAX_BULK_DATA_MSG_SIZE,
                                      buf, false)) == -1) {
            return -1;
        }
        if (bytes_read == 0) {
            return 0;
        }

        while (total_bytes_written < bytes_read) {
            if ((no_bytes_written =
                 write(data_file_fd, buf + total_bytes_written,
                       bytes_read - total_bytes_written)) == -1) {
                perror("Write file");
                return -1;
            }
            total_bytes_written += no_bytes_written;
        }
        total_bytes_read += bytes_read;
        print_transfer_progress_bar(total_bytes_read, size);
    }
}

int logout(int ctrl_socket_fd) {
    char msg[MAX_CTRL_MSG_SIZE];
    snprintf(msg, sizeof msg, "quit\n");
    return send_msg(ctrl_socket_fd, msg);
}

```

5.8 *url_parser.h*

```

#pragma once

#define MAX_URL_LENGTH 2048
#define MAX_USER_LENGTH 256
#define MAX_PASS_LENGTH 256
#define MAX_PORT_LENGTH 10

struct con_info {
    char addr[MAX_URL_LENGTH];
    char resource[MAX_URL_LENGTH];
    char user[MAX_USER_LENGTH];
    char pass[MAX_PASS_LENGTH];
    char port[MAX_PORT_LENGTH];
};

int validate_ftp_url(char *url);

```

```
int parse_url_con_info(char *url, struct con_info *con_info);
```

5.9 *url_parser.c*

```
#include <regex.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "url_parser.h"

#define ANONYMOUS_USER "anonymous"
#define ANONYMOUS_USER_PASS "1234"
#define DEFAULT_PORT "21"
#define DEFAULT_RESOURCE "/"

#define URL_CHARS_REGEX \
    "[[:alnum:]]\\$\\\\.\\|\\*\\(\\|\\),!'_-\\+)|(%[[:xdigit:]]{2})"

/**
 * @brief Verifies an URL is a valid FTP URL
 *
 * @param url URL to verify
 * @return int 0 in case of success, -1 otherwise
 */
int validate_ftp_url(char *url) {
    static const char url_regex[] =
        "^(ftp|FTP):/"
        "(((\" URL_CHARS_REGEX \")*(:(\" URL_CHARS_REGEX \")*)?@)?)"
        "((\" URL_CHARS_REGEX \")+((\" URL_CHARS_REGEX \")+)+)"
        "((:[[:digit:]]{1,4})?)"
        "(/(\" URL_CHARS_REGEX \")+)*[/]?$";

    regex_t regex;
    if (regcomp(&regex, url_regex, REG_EXTENDED | REG_NOSUB) != 0) {
        regfree(&regex);
        return -1;
    }
    if (regexexec(&regex, url, 0, NULL, 0) == REG_NOMATCH) {
        regfree(&regex);
        return -1;
    }
    regfree(&regex);
    return 0;
}

int parse_url_con_info(char *url, struct con_info *con_info) {
    regex_t regex;
    regmatch_t pmatch[1];
    regmatch_t last_match;

    /* Parse port */
    static const char port_regex[] = ":[[:digit:]]{1,4}";
    if (regcomp(&regex, port_regex, REG_EXTENDED) != 0) {
        regfree(&regex);
        return -1;
    }

    if (regexexec(&regex, url, 1, pmatch, 0) == 0) {
```

```

        last_match = pmatch[0];
        snprintf(con_info->port, sizeof con_info->port, "%.s",
                pmatch[0].rm_eo - pmatch[0].rm_so - 1,
                &url[pmatch[0].rm_so + 1]);
    } else {
        last_match.rm_so = -1;
        snprintf(con_info->port, sizeof con_info->port, "%s", DEFAULT_PORT);
    }

    /* Parse host address */
    static const char address[] =
        "[/@" URL_CHARS_REGEX ")+(\\" URL_CHARS_REGEX
        ")")+(:[[:digit:]]{1,4})?(/(\" URL_CHARS_REGEX ")+*[/]?";
    if (regcomp(&regex, address, REG_EXTENDED)) {
        regfree(&regex);
        return -1;
    }
    if (regexexec(&regex, url, 1, pmatch, 0) == 0) {
        char buf[MAX_URL_LENGTH];
        if (last_match.rm_so != -1) {
            snprintf(buf, sizeof buf, "%.s%.s",
                    last_match.rm_so - pmatch[0].rm_so - 1,
                    &url[pmatch[0].rm_so + 1],
                    pmatch[0].rm_eo - last_match.rm_eo + 1,
                    &url[last_match.rm_eo]);
        } else {
            snprintf(buf, sizeof buf, "%.s",
                    pmatch[0].rm_eo - pmatch[0].rm_so - 1,
                    &url[pmatch[0].rm_so + 1]);
        }

        char *resource_s = strchr(buf, '/');
        if (resource_s != NULL) {
            snprintf(con_info->resource, sizeof con_info->resource, "%s",
                    resource_s);
            snprintf(con_info->addr, sizeof con_info->addr, "%.s",
                    (int)(resource_s - buf), buf);
        } else {
            snprintf(con_info->addr, sizeof con_info->addr, "%.s",
                    (int)(strlen(buf, MAX_URL_LENGTH)), buf);
            snprintf(con_info->resource, sizeof con_info->resource, "/");
        }
        last_match = pmatch[0];
    }

    /* Parse username */
    static const char username[] = "/" URL_CHARS_REGEX ")*[:@]";
    if (regcomp(&regex, username, REG_EXTENDED)) {
        return -1;
    }
    if (regexexec(&regex, url, 1, pmatch, 0) == 0 &&
        pmatch[0].rm_so != last_match.rm_so) {
        snprintf(con_info->user, sizeof con_info->user, "%.s",
                pmatch[0].rm_eo - pmatch[0].rm_so - 2,
                &url[pmatch[0].rm_so + 1]);
    } else {
        snprintf(con_info->user, sizeof con_info->user, ANONYMOUS_USER);
    }

    /* Parse password */
    static const char password[] = ":(\" URL_CHARS_REGEX ")*@";
    if (regcomp(&regex, password, REG_EXTENDED)) {
        regfree(&regex);
    }

```

```

        return -1;
    }
    if (regexec(&regex, url, 1, pmatch, 0) == 0) {
        snprintf(con_info->pass, sizeof con_info->pass, "%.s",
            pmatch[0].rm_eo - pmatch[0].rm_so - 2,
            &url[pmatch[0].rm_so + 1]);
    } else {
        snprintf(con_info->pass, sizeof con_info->pass, ANONYMOUS_USER_PASS);
    }

    regfree(&regex);
    return 0;
}

```

5.10 *utils.h*

```

#pragma once

#include <time.h>

/**
 * @brief Prints the progress bar.
 *
 * @param curr_byte Current byte in the transfer.
 * @param file_size File size (in bytes)
 */
void print_transfer_progress_bar(unsigned curr_byte, unsigned file_size);

/**
 * @brief Elapsed time during the transfer.
 *
 * @param start struct timespec containing info about the transfer start
 * @param end struct timespec containing info about the transfer end
 * @return double Elapsed time during the transfer (in seconds with a decimal
 * place)
 */
double elapsed_seconds(struct timespec *start, struct timespec *end);

```

5.11 *utils.c*

```

#include "utils.h"
#include <stdio.h>

void print_transfer_progress_bar(unsigned curr_byte, unsigned file_size) {
    static int last_percentage = -1;
    static const int percentage_bar_width = 30;
    float percentage = (float)curr_byte / (float)file_size;
    int percentage_ = (int)(percentage * 100);
    if (last_percentage != percentage_) {
        last_percentage = percentage_;
        printf("\r[");
        int pos = percentage_bar_width * percentage;
        for (int i = 0; i < percentage_bar_width; i++) {
            if (i < pos) {
                printf("=");
            } else if (i == pos) {
                printf(">");
            } else {
                printf(" ");
            }
        }
    }
}

```

```

        printf("] %d%% (%u/%uB) ", percentage_, curr_byte, file_size);
    }
    if (last_percentage == 100) {
        printf("\n");
    }
}

double elapsed_seconds(struct timespec *start, struct timespec *end) {
    double start_secs_decimal = (double)start->tv_nsec / 1000000000;
    double end_secs_decimal = (double)end->tv_nsec / 1000000000;
    double start_secs = (double)start->tv_sec + start_secs_decimal;
    double end_secs = (double)end->tv_sec + end_secs_decimal;
    return end_secs - start_secs;
}

void print_bytes(unsigned char *buf, int size) {
    printf("size: %d\n", size);
    for (int i = 0; i < size; i++) {
        printf("%x ", buf[i]);
    }
    printf("\n\n");
}

```