Configuração de uma rede e desenvolvimento de uma aplicação de *download*

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Turma 2 / Grupo 1:
António Pedro Araújo Fraga – 201303095
Luís Miguel da Costa Oliveira – 201304515
Miguel Guilherme Perestrelo Sampaio Pereira – 201305998

22 de dezembro de 2015

Sumário

Este relatório tem o propósito de cimentar o trabalho realizado e serve como material de apoio ao projeto que se trata de configurar e estudar uma rede, utilizando comandos de configuração do *router* e do *switch* e de desenvolver uma aplicação capaz de fazer *download* de um ficheiro através de um FTP.

O trabalho foi terminado com sucesso. A aplicação desenvolvida realiza a transferência do ficheiro sem erros e a foi possível configurar corretamente a rede.

Índice

1. Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Redes de Computadores e tem como metas fundamentais o desenvolvimento de uma aplicação capaz de realizar o *download* de um ficheiro utilizando o *File Transfer Protocol* (FTP) e a configuração e análise de uma rede.

Para a configuração da rede seguiram-se os passos descritos no guião que envolviam a configuração do IP de cada máquina, das LANs virtuais dentro do *switch*, do *router* com a implementação de NAT e, por fim, do DNS.

Estando a rede configurada, testou-se a aplicação de *download*. A aplicação foi desenvolvida em C recorrendo à utilização de *sockets* para a comunicação com o servidor, enviando comandos e recebendo respostas.

Este relatório começa por descrever como foi desenvolvida a aplicação e como ela está estruturada. Depois, serão abordadas cada uma das experiências realizadas ao longo das aulas práticas, explicando o objetivo de cada uma e os comandos utilizados para a realizar a configuração. Por último, serão apresentadas algumas conclusões.

2. Aplicação de *download*

2.1. Arquitetura

A aplicação está dividida em duas partes, a primeira parte trata do processamento da *string* que é passada como argumento e guarda todos os seus componentes numa estrutura *url* através da função *init_url*. Esses componentes fazem parte da informação necessária para que a transferência ocorra, sendo que entre eles estão a *string* de utilizador e palavra-passe assim como a *string* que contém o *hostname* e o caminho do ficheiro a transferir.

```
url *new_url = malloc(sizeof(url));

if (init_url(new_url, argv[1], debug_mode) == ERROR)
    return ERROR;
```

Figura 1 - Chamamento de *init_url*

A segunda parte tem como objetivo estabelecer duas ligações (ligação A e ligação B) com o *host* definido anteriormente, fazendo uso de várias tarefas para o conseguir. A ligação A será a ligação de controlo, por isso é inicializada uma estrutura *connection* que guardará os dados necessários.

```
connectionA->fd = connect_to(connectionA->ip, connectionA->port, debug_mode);
```

Figura 2 - Ligação A

A primeira coisa a fazer é resolver o IP a que o *host* está associado através da função *get_ip*. Só assim será possível estabelecer uma ligação. Esta função contém maioritariamente código de um exemplo de obtenção de IP a partir da *string* de *host* que estava presente no *moodle* da unidade curricular. Além desse código, é definida a porta a usar na ligação de controlo, a porta 21. A *string* que contém o IP e a porta é guardada na estrutura *connection* da ligação A.

```
if (get_ip(connectionA, new_url, debug_mode) == ERROR)
    return ERROR;
```

Figura 3 - Chamamento de *get_ip*

Depois dessas informações estarem guardadas é possível estabelecer uma ligação com o *host* através da função *connect_to* que também contém o código disponibilizado no *moodle* de um exemplo de conexão a um determinado IP. Depois de uma tentativa de conexão com o *host*, no caso da ligação de controlo, é recebida uma resposta com um determinado código; a função *read_from_host* verifica se o código recebido é o código esperado. Esta função retorna o descritor de ficheiro da ligação aberta, neste caso a ligação A.

Quando a ligação estiver completa é altura de o programa se autenticar fazendo uso do utilizador e palavra-passe guardados anteriormente na estrutura *url*. O programa chama assim a função *log_in_host*, que envia uma mensagem com a informação da

string de utilizador através do método *send_to_host*; é recebida a resposta e o código dessa mesma resposta é verificado. Repete-se o mesmo procedimento para enviar a informação sobre a *string* da *password* e no fim deve receber-se o código correspondente à resposta de um uma autenticação bem sucedida.

```
if (log_in_host(connectionA, new_url, debug_mode) == ERROR)
    return ERROR;
```

Figura 4 - Chamamento de *log_in_host*

É então enviado o comando para que o servidor de FTP transfira dados em modo passivo. É executada a função *pasv_host*, que envia a mensagem e interpreta a reposta do *host*, para que o IP e a porta a serem utilizados pela ligação B possam ser guardados. Esta resposta é interpretada pelo método *get_pasv_from_host* e estes dados são guardados numa nova estrutura *connection*, estrutura essa que se refere à ligação B.

```
if (pasv_host(connectionA, new_url, debug_mode, connectionB) == ERROR)
    return ERROR;
```

Figura 5 - Chamamento de *pasv_host*

```
if( get_pasv_from_host(connectionA->fd, ip, &port, debug_mode) < 0 ) {
    printf("\t->Error interpreting passive message.\n");
    return ERROR;
}
```

Figura 6 - Chamamento de *get_pasv_from_host*

Já que o servidor entrou no modo passivo e existem todos os dados de ligação da conexão B (calculados e interpretados anteriormente), é executada a função *connect_to*, para que haja uma ligação de transferência de ficheiro. Quando a ligação estiver completa é enviada uma mensagem a partir da ligação A que define o caminho do ficheiro dentro do servidor; a resposta do servidor define se o ficheiro existe ou não.

```
if (def_path(connectionA, new_url->path, debug_mode) == ERROR)
    return ERROR;
```

Figura 7 - Chamamento de função *def_path*

Os dados para que a transferência ocorra estão enviados. Então, a ligação B começa a transferir o ficheiro (*download_from_host*), abrindo um ficheiro para escrita e escrevendo os dados recebidos a cada 1024 *bytes*, fechando-o quando este estiver terminado. A ligação B é fechada e passa-se à desconexão da ligação A, enviando-se uma mensagem de terminação ao servidor e libertando a memória alocada anteriormente.

```
debug_msg(1, "Downloading file...");

if (download_from_host(connectionB, new_url->path, debug_mode) == ERROR)
    return ERROR;

debug_msg(1, "Download completed...\n");

debug_msg(debug_mode, "Disconnecting...");

if (disconnect_host(connectionA, new_url, debug_mode) == ERROR)
    return ERROR;

debug_msg(debug_mode, "Diconnected.\n");
```

Figura 8 - Terminação da ligação

2.2. Resultados

Para uma melhor análise de resultados o grupo decidiu implementar um modo de *debug* que pode ou não ser ativado. Este modo *debug* imprime várias informações sobre toda a execução do programa desenvolvido. Uma transferência só é bem-sucedida se todas as respostas por parte do servidor contiverem o código de resposta positivo em relação à mensagem enviada pelo programa. Se por qualquer motivo o código de resposta não for aquele que é suposto ser então o programa termina a sua execução e imprime a respetiva mensagem de erro, uma possibilidade seria a falha de autenticação. Esta verificação é feita pelo método *read_from_host* que recebe como argumento o código que é suposto receber em cada chamada. De seguida apresenta-se o resultado de uma execução com sucesso com o modo *debug* ativado.

```
Initializing a url struct.
    ->Getting url strings...
    ->Completed!
Url struct initialized.

Getting ip by host name.
    ->Getting host ip by name...
    ->Completed!
Host is valid, and it returned a valid ip.

Connecting to A '193.136.37.8' through port 21...
    ->Creating a socket...
    ->Socket creatd.
    ->Connecting...
    ->Connected!
    ->Receiving response message...
    ->Message Code: 220
    ->Message received!
Connection was successfull.

Logging in.
    ->Sending user to host...
    ->User sent!
    ->Receiving message from host...
    ->Message Code: 331
    ->Message received!
    ->Sending passoword to host...
    ->Password sent!
    ->Receiving messagem from host...
    ->Message Code: 230
    ->Message received!
Logged in messages were sent.

Entering passive mode...
    ->Sending passive message to host...
    ->Passive message sent!
    ->Interpreting passive message from host...
    ->Passive Message: 227 Entering Passive Mode
(193,136,37,8,251,201)
    ->Interpreted IP: 193.136.37.8
    ->Interpreted Port: 64457
    ->Completed!
Completed!

Connecting to B '193.136.37.8' through port 64457...
```

```
    ->Creating a socket...
    ->Socket created.
    ->Connecting...
    ->Connected!
Connected!

Sending path...
    ->File: pub/CPAN/RECENT-1M.json
    ->Sending 'retr' command to host...
    ->Command sent!
    ->Receiving message from host...
    ->Message Code: 150
    ->Message received!
Path was sent!

Downloading file...
    ->Creating file with defined path...
    ->Created!
    ->Downloading...
    ->Completed!
Download completed...

Disconnecting...
    ->Sending 'quit' command to host...
    ->Closing socket...
Disconnected.
```

3. Estudo da configuração da rede

3.1. Configurar um IP de rede

O objetivo desta experiência foi configurar os endereços de IP de dois computadores para que estes conseguissem comunicar. Depois de configurar as portas eth0 e adicionar as rotas necessárias, utilizou-se o comando *ping* para verificar a existência de uma ligação entre os dois computadores.

O *Address Resolution Protocol* (ARP) é um protocolo utilizado para a resolução de endereços da camada de rede (endereços IP) em endereços da camada de ligação de dados (endereços Ethernet). Para enviar uma trama para um computador na rede, o emissor tenta descobrir o endereço MAC correspondente ao endereço IP, difundindo em *broadcast* um pacote ARP que contém o endereço de IP e espera uma resposta com o endereço MAC que lhe corresponde.

O comando *ping* gera pacotes do protocolo ICMP. Para distinguir os pacotes ARP, IP e ICMP, é necessário verificar o cabeçalho da trama Ethernet, sendo que os pacotes IP contêm, ainda, informação acerca do tamanho da trama.

A interface *loopback* é uma interface de rede virtual que o computador utiliza para realizar testes de diagnóstico. Esta interface permite ter um endereço de IP no *router* que está sempre ativo, não dependendo de uma interface física.

Para esta experiência, utilizou-se o comando *ifconfig* para configurar os endereços de IP nos dois computadores.

3.2. Implementar duas LANs virtuais no *switch*

O objetivo da experiência foi criar duas LANs virtuais no *switch*, uma com os computadores 1 e 4 e a outra com o computador 2. Assim, o computador 2 deixaria de ter acesso aos computadores 1 e 4, visto que se encontram em sub-redes diferentes.

Para configurar o *switch*, entrou-se na sua consola de configuração e executaram-se os seguintes comandos:

- *vlan i*, para indicar que se estaria a atuar sobre a VLAN identificada por *i*;
- *interface fastethernet 0/j*, para adicionar a porta *j* à VLAN;
- *switchport mode access;*
- *switchport access vlan i.*

Existem dois domínios de *broadcast* que correspondem a cada umas das VLANs criadas.

3.3. Configurar um *router* em Linux

Esta experiência tinha como objetivo configurar o computador 4 para funcionar como um *router* que permitisse a comunicação entre as duas VLANs criadas.

Para o efeito, foi necessário configurar porta eth1 do computador 4 com um IP na mesma gama que o computador 2. De seguida, adicionaram-se as rotas corretas aos computadores com o comando *route add*. No computador 1 adicionou-se a rota que indica que os pacotes devem ser reencaminhados para o endereço IP do computador 4, sendo feito o mesmo para o computador 2, mas com IP do computador 4 da sub-rede 1.

Assim, é possível fazer *ping* do computador 2 para o 1. Os pedidos são reencaminhados para o computador 4 que está ligado à sub-rede de ambos os computadores, conseguindo comunicar com os dois.

Analisando os *logs*, é possível perceber que, quando é feito um ping do computador 1 ao computador 2, o pacote ICMP contém como endereço de destino, o endereço MAC do computador 4. Na resposta do computador 2, o pacote contém como endereço de origem o endereço MAC do computador 4.

Para além do que foi feito na experiência 2, foi também preciso adicionar a porta do *switch* ligada a eth1 à VLAN com o computador 2, utilizando os comandos de configuração do *switch*.

3.4. Configurar um *router* comercial e implementar NAT

O objetivo da experiência foi configurar um *router* comercial com NAT implementado.

O NAT – *Network Address Translation* – possibilita que os computadores de uma rede interna, como as que foram criadas, tenham acesso ao exterior. Para tal, é gerado um número de 16 bits, utilizando-se uma *hash table*, e escrevendo-o no campo da porta de origem. Na resposta, realiza-se o processo inverso, para que o *router* saiba para qual computador deve enviar a resposta.

Configurou-se o *router* definindo as rotas internas e externas com o comando *ip route* na consola de configuração do *router*. De seguida, definiu-se o computador 4 como *default gateway* do computador 1 e o *router* como *default gateway* dos computadores 2

e 4. Assim, os pacotes enviados pelo computador 1 seguem para o computador 4 e depois para o *router* ou para o computador 2.

3.5. DNS

O objetivo desta experiência foi configurar o *Domain Name System* (DNS), para ser possível aceder a redes externas.

Para configurar o DNS, basta editar o ficheiro *resolv.conf*, indicando os parâmetros do DNS fornecido no guião.

Quando se faz *ping* a um servidor externo, é enviado um pacote de DNS que pede o IP do servidor. Em resposta, chega outro pacote DNS que contém a informação pedida.

3.6. Ligações TCP

Depois de a rede estar completamente configurada, procedeu-se ao teste da aplicação desenvolvida.

O teste foi feito com recurso à transferência de um ficheiro através de um servidor FTP. A transferência foi bem-sucedida, mostrando que a configuração da rede foi feita corretamente.

O *Transmission Control Protocol* (TCP) utiliza o mecanismo *Automatic Repeat Request* (ARQ) que é um método de controlo de erros na transmissão de dados que utiliza *acknowledgments* (mensagens enviadas pelo recetor indicando que a trama de dados foi recebida corretamente) e *timeouts* (tempo permitido para esperar por um *acknowledgment*), de forma a garantir uma transmissão confiável através serviço não confiável. Se não for recebido um *acknowledgment* antes do *timeout*, a trama é retransmitida até ser recebido um *acknowledgment*.

Para fazer o controlo de congestão, o TCP mantém uma janela de congestão que consiste numa estimativa do número de octetos que a rede consegue encaminhar, não enviando mais octetos do que o mínimo da janela definida pelo recetor e pela janela de congestão.

Uma vez que a taxa de transferência é distribuída de forma igual para cada ligação, a transferência de dados em simultâneo leva a uma queda na taxa de transmissão.

4. Conclusão

Com o desenvolvimento deste trabalho, foi possível interiorizar os conceitos necessários e perceber melhor como funciona algo que está presente no dia-a-dia de todos.

Por outro lado, a aplicação de *donwload* e o seu desenvolvimento permitiu-nos perceber como funcionam as transferências por FTP e o próprio protocolo.

Podemos concluir que o projeto foi terminado com sucesso, visto que o grupo conseguiu implementar tudo o que era proposto, dando uma perspetiva diferente de como funcionam os dispositivos utilizados.

Referências

1. Address Resolution Protocol, https://en.wikipedia.org/wiki/Address_Resolution_Protocol
2. Transmission Control Protocol, https://en.wikipedia.org/wiki/Transmission_Control_Protocol
3. Automatic repeat request, https://en.wikipedia.org/wiki/Automatic_repeat_request
4. Controlo de congestão, http://www.gsd.inesc-id.pt/~ler/docencia/prd0304/handouts09.pdf

Anexos

Código da aplicação

main.c

```
#include "utilities.h"
#include "url.h"
#include "connection.h"


int main (int argc, char** argv) {

    if (argc != 3) {
        printf("\nUsage Error!\n");
        printf("\n\nUsage:\n");
        printf("| name |\t\t url\t\t\t| debug mode |\n");
        printf(" ./app ftp://[<user>:<password>@]<host>/<url-path>
<ON/OFF>\n\n");
        return ERROR;
    }

    if (strcmp(argv[2], "ON") != 0 && strcmp(argv[2], "OFF") != 0) {
        printf("Error!\n");
        printf("\nPlease give a valid debug mode: ON/OFF\n");
        return ERROR;
    }

    int debug_mode = strcmp(argv[2], "ON") == 0 ? 1 : 0;

    debug_msg(debug_mode, "Initializing a url struct.");

    url *new_url = malloc(sizeof(url));
    if (init_url(new_url, argv[1], debug_mode) == ERROR)
        return ERROR;

    debug_msg(debug_mode, "Url struct initialized.\n");
    debug_msg(debug_mode, "Getting ip by host name.");

    connection * connectionA = malloc(sizeof(connection));

    if (get_ip(connectionA, new_url, debug_mode) == ERROR)
        return ERROR;

    debug_msg(debug_mode, "Host is valid, and it returned a valid
ip.\n");

    char * host_info = malloc(30 * sizeof(char));
    strcpy(host_info, "Connecting to A '");
    strcat(host_info, connectionA->ip);
    strcat(host_info, "' through port ");
    char portA_str[15];
    sprintf(portA_str, "%d", connectionA->port);
    strcat(host_info, portA_str);
    strcat(host_info, "...");

    debug_msg(debug_mode, host_info);

    connectionA->fd = connect_to(connectionA->ip, connectionA->port,
debug_mode);
```

```
    if (connectionA->fd == ERROR)
        return ERROR;

    debug_msg(debug_mode, "Connection was successfull.\n");

    debug_msg(debug_mode, "Logging in.");

    if (log_in_host(connectionA, new_url, debug_mode) == ERROR)
        return ERROR;

    debug_msg(debug_mode, "Logged in messages were sent.\n");

    debug_msg(debug_mode, "Entering passive mode...");

    connection * connectionB = malloc(sizeof(connection));

    if (pasv_host(connectionA, new_url, debug_mode, connectionB) ==
ERROR)
        return ERROR;

    debug_msg(debug_mode, "Completed!\n");

    char * data_host_info = malloc(30 * sizeof(char));
    strcpy(data_host_info, "Connecting to B '");
    strcat(data_host_info, connectionA->ip);
    strcat(data_host_info, "' through port ");
    char portB_str[15];
    sprintf(portB_str, "%d", connectionB->port);
    strcat(data_host_info, portB_str);
    strcat(data_host_info, "...");

    debug_msg(debug_mode, data_host_info);

    connectionB->fd = connect_to(connectionB->ip, connectionB->port,
debug_mode);

    if (connectionB->fd == ERROR)
        return ERROR;

    debug_msg(debug_mode, "Connected!\n");

    debug_msg(debug_mode, "Sending path...");

    if (def_path(connectionA, new_url->path, debug_mode) == ERROR)
        return ERROR;

    debug_msg(debug_mode, "Path was sent!\n");

    debug_msg(1, "Downloading file...");

    if (download_from_host(connectionB, new_url->path, debug_mode) ==
ERROR)
        return ERROR;

    debug_msg(1, "Download completed...\n");

    debug_msg(debug_mode, "Disconnecting...");

    if (disconnect_host(connectionA, new_url, debug_mode) == ERROR)
        return ERROR;
```

```
    debug_msg(debug_mode, "Diconnected.\n");

    return OK;
}
```

connection.h

```
#pragma once

#include "utilities.h"
#include "url.h"


typedef struct Connection {
    int fd;
    char * ip;
    int port;
} connection;

int get_ip(connection * connection, url* url, int debug_mode);

int connect_host(connection * connection, url * url, int debug_mode);

int connect_to(char * ip, int port, int debug_mode);

int log_in_host(connection * connection, url * url, int debug_mode);

int pasv_host(connection * connectionA, url * url, int debug_mode,
connection * connectionB);

int get_pasv_from_host(int connection_fd, char* ip, int * port, int
debug_mode);

int def_path(connection * connectionA,char * path, int debug_mode);

int download_from_host(connection * connectionB, char* path, int
debug_mode);

int disconnect_host(connection * connectionA, url * url, int
debug_mode);

int send_to_host(int connection_fd, const char* msg);

int read_from_host(int connection_fd, char* msg, int debug_mode, char
* code);
```

connection.c

```
#include "connection.h"


int get_ip(connection * connection, url* url, int debug_mode) {
    struct hostent* h;

    debug_sub_msg(debug_mode, "Getting host ip by name...");

    if ((h = gethostbyname(url->host)) == NULL) {
        herror("Error, could not execute gethostbyname()");
        return ERROR;
    }
```

```
    debug_sub_msg(debug_mode, "Completed!");

    char* ip = inet_ntoa(*((struct in_addr *) h->h_addr));

    connection->ip = malloc(strlen(ip));
    strcpy(connection->ip, ip);

    connection->port = 21;

    return OK;
}


int connect_to(char * ip, int port, int debug_mode) {

    struct sockaddr_in server_addr;
    bzero((char*)&(server_addr),sizeof((server_addr)));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ip);
    server_addr.sin_port = htons(port);

    debug_sub_msg(debug_mode, "Creating a socket...");

    int fd;

    if ((fd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
            perror("\t->Error, could not execute socket()");
            exit(ERROR);
    }

    debug_sub_msg(debug_mode, "Socket created.");

    debug_sub_msg(debug_mode, "Connecting...");

    if (connect(fd, (struct sockaddr *)&(server_addr),
sizeof(server_addr)) < 0){
        perror("\t->Error, could not execute connect()");
        exit(ERROR);
    }

    debug_sub_msg(debug_mode, "Connected!");

    if (port == 21) {

        char * msg = malloc(5 * sizeof(char));

        debug_sub_msg(debug_mode, "Receiving response message...");

        if (read_from_host(fd, msg, debug_mode, "220") == ERROR) {
            printf("\nNot a valid connect message!\n\n");
            return ERROR;
        }

        debug_sub_msg(debug_mode, "Message received!");

        free(msg);

    }

    return fd;
}
```

```
int log_in_host(connection * connection, url * url, int debug_mode) {

    char * user = malloc(sizeof(url->user) + 5 * sizeof(char));
    sprintf(user, "user %s\r\n", url->user);

    debug_sub_msg(debug_mode, "Sending user to host...");

    if (send_to_host(connection->fd, user) == ERROR) {
        printf("\t->Error sending a message to host.");
        return ERROR;
    }

    debug_sub_msg(debug_mode, "User sent!");

    debug_sub_msg(debug_mode, "Receiving message from host...");

    if (read_from_host(connection->fd, user, debug_mode, "331") ==
ERROR) {
        printf("\nNot a valid user message!\n\n");
        return ERROR;
    }


    debug_sub_msg(debug_mode, "Message received!");


    char * password = malloc(sizeof(url->password) + 5 *
sizeof(char));
    sprintf(password, "pass %s\r\n", url->password);

    debug_sub_msg(debug_mode, "Sending password to host...");

    if (send_to_host(connection->fd, password) == ERROR) {
        printf("\t->Error sending a message to host.");
        return ERROR;
    }

    debug_sub_msg(debug_mode, "Password sent!");

    debug_sub_msg(debug_mode, "Receiving message from host...");

    if (read_from_host(connection->fd, password, debug_mode, "230") ==
ERROR) {
        printf("\nLog in failed!\n\n");
        return ERROR;
    }

    free(password);

    debug_sub_msg(debug_mode, "Message received!");


    return OK;
}


int pasv_host(connection * connectionA, url * url, int debug_mode,
connection * connectionB) {

    char * pasv = malloc(7 * sizeof(char));
```

```
    sprintf(pasv, "pasv \r\n");

    debug_sub_msg(debug_mode, "Sending password to host...");

    if (send_to_host(connectionA->fd, pasv) == ERROR) {
        printf("\t->Error sending a message to host.");
        return ERROR;
    }

    debug_sub_msg(debug_mode, "Password sent!");

    debug_sub_msg(debug_mode, "Interpreting passive message from
host...");

    char * ip = malloc(50 * sizeof(char));

    int port;

    if( get_pasv_from_host(connectionA->fd, ip, &port, debug_mode) < 0
) {
        printf("\t->Error interpreting passive message.\n");
        return ERROR;
    }

    char * ip_info = malloc(1024 * sizeof(char));

    strcpy(ip_info, "Interpreted IP: \0");
    strcat(ip_info, ip);

    debug_sub_msg(debug_mode, ip_info);

    char * port_info = malloc(1024 * sizeof(char));

    if (sprintf(port_info, "Interpreted Port: %d", port) < 0)  {
        printf("\t->Error printing port to string.\n");
        return ERROR;
    }

    connectionB->ip = ip;
    connectionB->port = port;

    debug_sub_msg(debug_mode, port_info);

    debug_sub_msg(debug_mode, "Completed!");

    return OK;
}

int def_path(connection * connectionA, char * path, int debug_mode) {

    char * retr = malloc(1024 * sizeof(char));
    sprintf(retr, "retr %s\r\n", path);

    char * path_info = malloc(1024 * sizeof(char));
    sprintf(path_info, "File: %s", path);

    debug_sub_msg(debug_mode, path_info);

    debug_sub_msg(debug_mode, "Sending 'retr' command to host...");

    if (send_to_host(connectionA->fd, retr) == ERROR) {
```

```
        printf("\t->Error sending a message to host.");
        return ERROR;
    }

    debug_sub_msg(debug_mode, "Command sent!");

    debug_sub_msg(debug_mode, "Receiving message from host...");

    if (read_from_host(connectionA->fd, retr, debug_mode, "150") ==
ERROR) {
        printf("\nPath is not valid!\n\n");
        return ERROR;
    }

    free(retr);
    free(path_info);

    debug_sub_msg(debug_mode, "Message received!");

    return OK;
}



int disconnect_host(connection * connectionA, url * url, int
debug_mode) {

    char * quitA = malloc(6 * sizeof(char));

    sprintf(quitA, "quit\r\n");

    debug_sub_msg(debug_mode, "Sending 'quit' command to host...");

    if (send_to_host(connectionA->fd, quitA) == ERROR) {
        printf("\t->Error sending a message to host A.");
        return ERROR;
    }

    debug_sub_msg(debug_mode, "Closing socket...");

    if (connectionA->fd) {
        close(connectionA->fd);
        free(connectionA);
    }

    free(url);

    return OK;
}



int send_to_host(int connection_fd, const char* msg) {

    int written_bytes = 0;

    written_bytes = write(connection_fd, msg, strlen(msg));

    int return_value = (written_bytes == strlen(msg)) ? OK : ERROR;

    return return_value;
```

```
}


int read_from_host(int connection_fd, char* msg, int debug_mode, char
* code) {

    FILE* fp = fdopen(connection_fd, "r");
    int size = 4;

    if(debug_mode)
        printf("\t->Message Code: ");

    do {
        memset(msg, 0, size);
        msg = fgets(msg, size, fp);

        if(debug_mode)
            printf("%s", msg);

    } while (!('1' <= msg[0] && msg[0] <= '5'));

    if(debug_mode)
        printf("\n");

    if (strcmp(msg, code) != 0) {
        char * error_msg = malloc(1024 * sizeof(char));
        strcpy(error_msg, "\n\nError!! It was suposed to receive a
message with the '");
        strcat(error_msg, code);
        strcat(error_msg, "' code, and a it was received a message
with the '");
        strcat(error_msg, msg);
        strcat(error_msg, "' code...\n\n");

        if(debug_mode)
            printf(error_msg, "\n\nError! You received a wrong code
message\n");

        return ERROR;
    }

    return OK;
}



int get_pasv_from_host(int connection_fd, char* ip_str, int * port,int
debug_mode) {

    FILE* fp = fdopen(connection_fd, "r");
    int size = 1024;
    char * msg = malloc(size * sizeof(char));

    if(debug_mode)
        printf("\t->Passive Message: ");

    do {
        memset(msg, 0, size);
        msg = fgets(msg, size, fp);
```

```
        if(debug_mode)
            printf("%s", msg);

    } while (!('1' <= msg[0] && msg[0] <= '5'));

    int ip[4];
    int port_arr[2];

    if ((sscanf(msg, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)",
        &ip[0], &ip[1], &ip[2], &ip[3], &port_arr[0], &port_arr[1])) <
0)
        return ERROR;

    if (sprintf(ip_str, "%d.%d.%d.%d", ip[0], ip[1], ip[2], ip[3]) <
0)
        return ERROR;

    *port = 256 * port_arr[0] + port_arr[1];


    return OK;
}


int download_from_host(connection * connectionB, char* path, int
debug_mode) {
    FILE* file;
    int bytes;

    char * filename = basename(path);

    debug_sub_msg(1, "Creating file with defined path...");

    if (!(file = fopen(filename, "w"))) {
        printf("ERROR: Cannot open file.\n");
        return ERROR;
    }

    debug_sub_msg(1, "Created!");

    debug_sub_msg(1, "Downloading...");

    char buf[1024];
    while ((bytes = read(connectionB->fd, buf, sizeof(buf)))) {
        if (bytes < 0) {
            printf("ERROR: Nothing was received from data socket
fd.\n");
            return ERROR;
        }

        if ((bytes = fwrite(buf, bytes, 1, file)) < 0) {
            printf("ERROR: Cannot write data in file.\n");
            return ERROR;
        }
    }

    debug_sub_msg(1, "Completed!");

    if(connectionB->fd) {
        close(connectionB->fd);
        free(connectionB);
```

```
    }

    if(file)
        fclose(file);

    return 0;
}
```

url.h

```
#pragma once

typedef struct Url {
    char * user;
    char * password;
    char * host;
    char * path;
    char * filename;
} url;


int init_url(url * url, char * url_str, int debug_mode);
```

url.c

```
#include "utilities.h"
#include "url.h"


int init_url(url * url, char * url_str, int debug_mode) {

    debug_sub_msg(debug_mode, "Getting url strings...");

    char * str_beg = malloc(6 * sizeof(char));
    memcpy(str_beg, url_str, 6);

    if (strcmp(str_beg, "ftp://\0") != 0) {
        printf("\nError! Please start your url by 'ftp://'...\n");
        return ERROR;
    }

    char ** sub_str = malloc(5 * sizeof(char*));

    int size =  strlen(url_str) - 6;
    sub_str[0] = malloc(size);
    memcpy(sub_str[0], url_str + 6, size);
    sub_str[0][size] = '\0';


    url->user = malloc(strlen(sub_str[0]));
    memcpy(url->user, sub_str[0], strlen(sub_str[0]));
    strtok(url->user, ":");

    if (strlen(url->user) == strlen(sub_str[0])) {
        printf("\nError!!! Please declare an user...\n");
        return ERROR;
    }

    size = strlen(sub_str[0]) - strlen(url->user) - 1;
    sub_str[1] = malloc(size);
    memcpy(sub_str[1], sub_str[0] + strlen(url->user) + 1, size);
    sub_str[1][size] = '\0';
```

```
    url->password = malloc(strlen(sub_str[1]));
    memcpy(url->password, sub_str[1], strlen(sub_str[1]));
    strtok(url->password, "@");

    if (strlen(url->password) == strlen(sub_str[1])) {
        printf("\nError!!! Please declare a password...\n");
        return ERROR;
    }

    size = strlen(sub_str[1]) - strlen(url->password) - 1;
    sub_str[2] = malloc(size);
    memcpy(sub_str[2], sub_str[1] + strlen(url->password) + 1, size);
    sub_str[2][size] = '\0';

    url->host = malloc(strlen(sub_str[2]));
    memcpy(url->host, sub_str[2], strlen(sub_str[2]));
    strtok(url->host, "/");

    if (strlen(url->host) == strlen(sub_str[2])) {
        printf("\nError! Please declare a host...\n");
        return ERROR;
    }


    size = strlen(sub_str[2]) - strlen(url->host) - 1;
    sub_str[3] = malloc(size);
    memcpy(sub_str[3], sub_str[2] + strlen(url->host) + 1, size);
    sub_str[3][size] = '\0';

    url->path = malloc(strlen(sub_str[3]));
    size = strlen(sub_str[3]);
    memcpy(url->path, sub_str[3], size);
    url->path[size] = '\0';

    if (!strlen(url->path)) {
        printf("\nError! Please declare a path...\n");
        return ERROR;
    }


    debug_sub_msg(debug_mode, "Completed!");

    return OK;
}
```

utilities.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <signal.h>
```

```
#include <libgen.h>


#define ERROR -1
#define OK 0

void debug_msg (int debug_mode, char * msg);
void debug_sub_msg (int debug_mode, char * msg);
```

utilities.c

```
#include <stdio.h>

void debug_msg (int debug_mode, char * msg) {
    if (debug_mode) {
        printf(msg, "Error printing msg, missing argument!");
        printf("\n");
    }
}


void debug_sub_msg (int debug_mode, char * msg) {
    if (debug_mode) {
        printf("\t->");
        printf(msg, "\t->Error printing msg, missing argument!");
        printf("\n");
    }
}
```

Comandos de configuração

Computadores

```
#!/bin/bash

ifconfig eth0 up 172.16.10.1/24
route add -net 172.16.11.0/24 gw 172.16.10.254
route add default gw 172.16.10.254
printf "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" >
/etc/resolv.conf
echo "tux1 configured"
```

```
#!/bin/bash

ifconfig eth0 up 172.16.11.1/24
route add -net 172.16.10.0/24 gw 172.16.11.253
route add default gw 172.16.11.254
printf "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" >
/etc/resolv.conf
echo "tux2 configured"
```

```
#!/bin/bash

ifconfig eth0 up 172.16.10.254/24
ifconfig eth1 up 172.16.11.253/24
route add default gw 172.16.11.254
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
printf "search lixa.fe.up.pt\nnameserver 172.16.1.1\n" >
/etc/resolv.conf
echo "tux4 configured"
```

Router

```
conf t
interface gigabitethernet 0/0
ip address 172.16.11.254 255.255.255.0
no shutdown
ip nat inside
exit

interface gigabitethernet 0/1
ip address 172.16.1.19 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.1.19 172.16.1.19 prefix 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

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
end
```

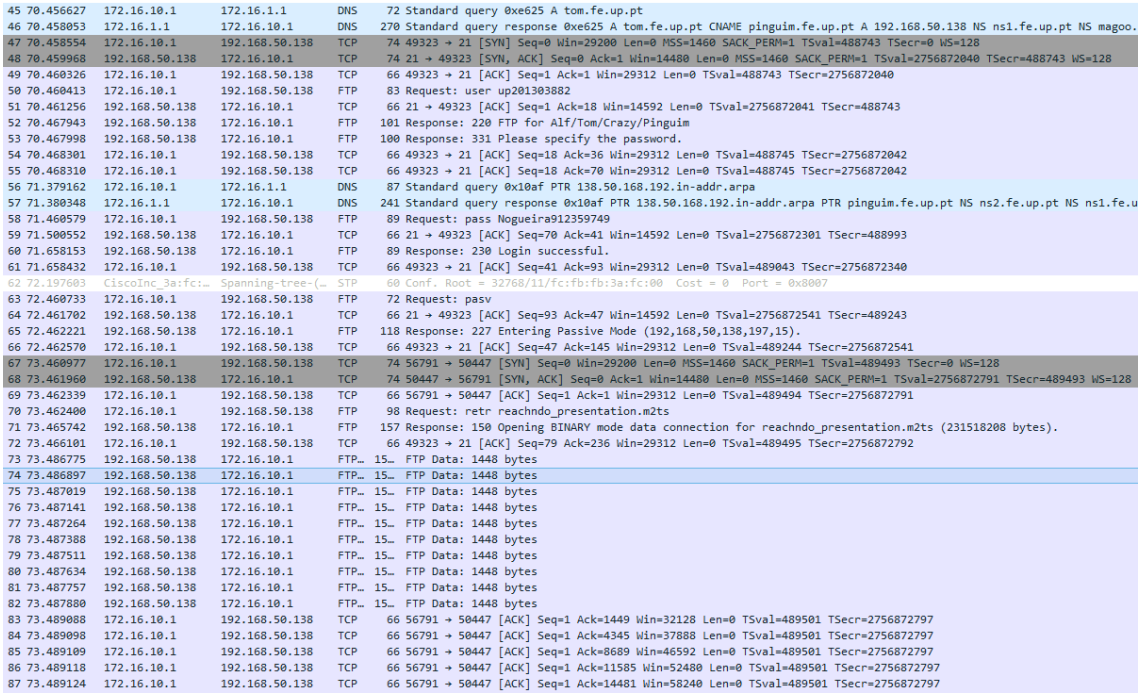
Switch

```
conf t
vlan 10
end

conf t
vlan 11
end

conf t
interface fastethernet 0/1
switchport mode access
switchport access vlan 10
end

conf t
interface fastethernet 0/3
switchport mode access
switchport access vlan 10
end

conf t
interface fastethernet 0/2
switchport mode access
switchport access vlan 11
end

conf t
interface fastethernet 0/4
switchport mode access
switchport access vlan 11
end

conf t
interface gigabitethernet 0/1
switchport mode access
```

```
switchport access vlan 20
end
```

Logs

Os logs encontram-se na pasta *logs* enviada em conjunto com este relatório, à exceção do *log* da experiência 6, devido ao seu tamanho. Por esse motivo, apresenta-se, de seguida, um excerto desse *log*.



45	70.456627	172.16.10.1	172.16.1.1	DNS	72 Standard query 0xe625 A tom.fe.up.pt
46	70.458053	172.16.1.1	172.16.10.1	DNS	270 Standard query response 0xe625 A tom.fe.up.pt CNAME pinguim.fe.up.pt A 192.168.50.138 NS ns1.fe.up.pt NS magoo.
47	70.458554	172.16.10.1	192.168.50.138	TCP	74 49323 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=488743 TSecr=0 WS=128
48	70.459968	192.168.50.138	172.16.10.1	TCP	74 21 → 49323 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=2756872040 TSecr=488743 WS=128
49	70.460326	172.16.10.1	192.168.50.138	TCP	66 49323 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=488743 TSecr=2756872040
50	70.460413	172.16.10.1	192.168.50.138	FTP	83 Request: user up201303882
51	70.461256	192.168.50.138	172.16.10.1	TCP	66 21 → 49323 [ACK] Seq=1 Ack=18 Win=14592 Len=0 TSval=2756872041 TSecr=488743
52	70.467943	192.168.50.138	172.16.10.1	FTP	101 Response: 220 FTP for Alf/Tom/Crazy/Pinguim
53	70.467998	192.168.50.138	172.16.10.1	FTP	100 Response: 331 Please specify the password.
54	70.468301	172.16.10.1	192.168.50.138	TCP	66 49323 → 21 [ACK] Seq=18 Ack=36 Win=29312 Len=0 TSval=488745 TSecr=2756872042
55	70.468310	172.16.10.1	192.168.50.138	TCP	66 49323 → 21 [ACK] Seq=18 Ack=70 Win=29312 Len=0 TSval=488745 TSecr=2756872042
56	71.379162	172.16.10.1	172.16.1.1	DNS	87 Standard query 0x10af PTR 138.50.168.192.in-addr.arpa
57	71.380348	172.16.1.1	172.16.10.1	DNS	241 Standard query response 0x10af PTR 138.50.168.192.in-addr.arpa PTR pinguim.fe.up.pt NS ns2.fe.up.pt NS ns1.fe.u
58	71.460579	172.16.10.1	192.168.50.138	FTP	89 Request: pass Nogueira912359749
59	71.500552	192.168.50.138	172.16.10.1	TCP	66 21 → 49323 [ACK] Seq=70 Ack=41 Win=14592 Len=0 TSval=2756872301 TSecr=488993
60	71.658153	192.168.50.138	172.16.10.1	FTP	89 Response: 230 Login successful.
61	71.658432	172.16.10.1	192.168.50.138	TCP	66 49323 → 21 [ACK] Seq=41 Ack=93 Win=29312 Len=0 TSval=489043 TSecr=2756872340
62	72.197603	CiscoInc_3a:fc:…	Spanning-tree-(…	STP	60 Conf. Root = 32768/11/fc:fb:fb:3a:fc:00 Cost = 0 Port = 0x8007
63	72.460733	172.16.10.1	192.168.50.138	FTP	72 Request: pasv
64	72.461702	192.168.50.138	172.16.10.1	TCP	66 21 → 49323 [ACK] Seq=93 Ack=47 Win=14592 Len=0 TSval=2756872541 TSecr=489243
65	72.462221	192.168.50.138	172.16.10.1	FTP	118 Response: 227 Entering Passive Mode (192,168,50,138,197,15).
66	72.462570	172.16.10.1	192.168.50.138	TCP	66 49323 → 21 [ACK] Seq=47 Ack=145 Win=29312 Len=0 TSval=489244 TSecr=2756872541
67	73.460977	172.16.10.1	192.168.50.138	TCP	74 56791 → 50447 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=489493 TSecr=0 WS=128
68	73.461960	192.168.50.138	172.16.10.1	TCP	74 50447 → 56791 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=2756872791 TSecr=489493 WS=128
69	73.462339	172.16.10.1	192.168.50.138	TCP	66 56791 → 50447 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=489494 TSecr=2756872791
70	73.462400	172.16.10.1	192.168.50.138	FTP	98 Request: retr reachndo_presentation.m2ts
71	73.465742	192.168.50.138	172.16.10.1	FTP	157 Response: 150 Opening BINARY mode data connection for reachndo_presentation.m2ts (231518208 bytes).
72	73.466101	172.16.10.1	192.168.50.138	TCP	66 49323 → 21 [ACK] Seq=79 Ack=236 Win=29312 Len=0 TSval=489495 TSecr=2756872792
73	73.486775	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
74	73.486897	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
75	73.487019	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
76	73.487141	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
77	73.487264	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
78	73.487388	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
79	73.487511	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
80	73.487634	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
81	73.487757	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
82	73.487880	192.168.50.138	172.16.10.1	FTP…	15… FTP Data: 1448 bytes
83	73.489088	172.16.10.1	192.168.50.138	TCP	66 56791 → 50447 [ACK] Seq=1 Ack=1449 Win=32128 Len=0 TSval=489501 TSecr=2756872797
84	73.489098	172.16.10.1	192.168.50.138	TCP	66 56791 → 50447 [ACK] Seq=1 Ack=4345 Win=37888 Len=0 TSval=489501 TSecr=2756872797
85	73.489109	172.16.10.1	192.168.50.138	TCP	66 56791 → 50447 [ACK] Seq=1 Ack=8689 Win=46592 Len=0 TSval=489501 TSecr=2756872797
86	73.489118	172.16.10.1	192.168.50.138	TCP	66 56791 → 50447 [ACK] Seq=1 Ack=11585 Win=52480 Len=0 TSval=489501 TSecr=2756872797
87	73.489124	172.16.10.1	192.168.50.138	TCP	66 56791 → 50447 [ACK] Seq=1 Ack=14481 Win=58240 Len=0 TSval=489501 TSecr=2756872797

Figura 9 - Excerto do log da experiência 6 (captura no computador 1)