Relatório do 2º Trabalho Laboratorial

Grupo 5

up201806551@fe.up.pt Beatriz Costa Silva Mendes
up201806528@fe.up.pt Clara Alves Martins

23 de dezembro de 2020

Redes de Computadores - 2020/21 - MIEIC

Professor das Aulas Laboratoriais: Manuel Alberto Pereira Ricardo

Índice

Sumário

Este relatório foi elaborado no âmbito da unidade curricular de Redes de Computadores. O trabalho consiste na configuração e estudo de uma rede de computadores. Este documento subdivide-se em diversas secções, destacando-se duas delas:

- A aplicação de *download*, onde é descrita a sua arquitetura e apresentados os resultados da sua execução;
- A configuração da rede, onde são respondidas várias questões relativamente às experiências realizadas e são analisados os resultados obtidos durante estas.

Assim sendo, o trabalho foi concluido com sucesso, visto que todos os objetivos foram cumpridos e foi finalizada uma aplicação capaz de transferir um ficheiro proveniente de um servidor FTP.

1 Introdução

O segundo projeto de Redes de Computadores tem duas grandes finalidades: o **desenvolvimento de uma aplicação de *download*** e a **configuração e estudo de uma rede de computadores**.

A **aplicação de *download*** foi desenvolvida de acordo com o protocolo FTP (RFC 793) e com a ajuda de ligações TCP (*Transmission Control Protocol*, RFC 793) a partir de *sockets* de *Berkeley*. Já relativamente à **configuração e estudo de uma rede de computadores**, o objetivo desta é permitir a execução de uma aplicação a partir de duas VLAN's dentro de um *switch*.

O relatório divide-se em:

1. **Introdução**, onde são descritos os objetivos do trabalho prático;
2. **Aplicação de Download**, onde é descrita a aplicação de *download*, a sua arquitetura e os respetivos resultados obtidos;
3. **Configuração da Rede e Análise**, onde é descrita a sua arquitetura, objetivos de cada experiência, comandos de configuração e análise dos resultados obtidos durante a sua realização;
4. **Conclusões**, onde é feita uma síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados;
5. **Referências**, onde são colocados todos os documentos consultados para a realização deste trabalho;
6. **Anexos**, onde se encontra o código fonte da aplicação desenvolvida e os resultados obtidos durante as experiências.

2 Aplicação de Download

A primeira parte do segundo projeto de Redes de Computadores consiste no desenvolvimento de uma aplicação de *download* na linguagem de programação C. A aplicação desenvolvida aceita como argumento um link no seguinte formato: `ftp://[<user>:<password>@]<host>/<url-path>`, tendo sido estudado o RFC 959, que aborda o protocolo FTP, para a implementação desta aplicação. O produto final consegue fazer *download* de qualquer ficheiro de um servidor FTP.

Posteriormente iremos descrever de forma sucinta a implementação da aplicação e as suas funcionalidades, assim como os resultados obtidos e a sua análise.

2.1 Arquitetura

A aplicação desenvolvida pelo nosso grupo de trabalho está dividida em duas camadas: a de **processamento do URL** e a do **cliente FTP**. Estas duas camadas relacionam-se apenas pelo facto da camada do cliente FTP utilizar a informação obtida no processamento do URL para estabelecer conexões, enviar comandos e fazer o *download* do ficheiro.

Primeiramente, é feito o processamento do URL. Para tal, são chamadas várias funções auxilares que realizam o *parse* da informação passada pelo argumento. As funções chamadas para este efeito são:

- `parseArgs`: processa o argumento e coloca-o na estrutura de dados `arguments`;
- `parseFilename`: através do *path* obtém-se o nome do ficheiro que se pretende fazer o *download*;
- `getIP`: obtém-se o IP a partir do *hostname* passado como argumento.

Posteriormente, com o processamento do URL terminado, entra-se na camada do cliente FTP. Inicialmente, é estabelecida a conexão, abrindo-se o *socket*. Para abertura deste, é necessário o IP obtido anteriormente e uma porta, que neste caso é a 21, a porta FTP, tal como foi abordado nas aulas teóricas.

Depois da conexão, é necessário fazer-se a comunicação, passando-se vários argumentos através do *socket* pela seguinte ordem:

1. **USER user**, envia-se o nome de utilizador para o servidor;
2. **PASS pass**, envia-se a password para o servidor;
3. **PASV**, entra-se em modo passivo, permitindo-se, assim, uma mútua comunicação entre o servidor e o cliente FTP, sendo estabelecida numa nova conexão TCP. No entando, agora o IP e a porta utilizados são as obtidas no modo passivo;
4. **RETR filename**, é pedido ao servidor o envio do ficheiro para *download*.

Após o envio de todos estes comandos, é necessário fazer o *download* do ficheiro. Para tal, é chamada a função `downloadFile`.

O código fonte da aplicação desenvolvida encontra-se no Anexo I.

2.2 Resultados de Donwload

A aplicação desenvolvida pelo nosso grupo de trabalho foi testada em diversas condições: modo anónimo, modo não anónimo, vários tipos de ficheiros, vários tamanhos de ficheiros, vários servidores FTP, entre outros. Ao longo da execução é impresso no terminal informação sobre o estado do programa para maior controlo por parte do utilizador.

Os resultados da execução do programa de *download* podem ser visualizados em 6.3.

3 Configuração da Rede e Análise

3.1 Experiência 1: Configurar um IP de Rede

O **ARP** (*Address Resolution Protocol*) é um protocolo utilizado para descobrir dinamicamente a relação entre os endereços da camada 3 (protocolo) e da camada 2 (*hardware*). Normalmente esta relação estebelece-se entre um endereço IP e o seu endereço de Ethernet, também designado por endereço MAC, podendo-se afirmar que os pacotes ARP possuem toda esta informação.

O comando ***ping***, após a obtenção do endereço MAC através dos pacotes ARP, gera pacotes do protocolo ICMP (*Internet Control Message Protocol*) e espera por uma resposta do *target host*.

Antes de começar a experiência, foi necessário fazer a configuração das máquinas seguindo os passos explicados em 6.2.

Durante a execução desta experiência, foi necessário fazer o *ping* do gnu 4 a partir do gnu 3. Aquando deste, o gnu 3 envia um pacote ARP (*request*) com o seu endereço IP (172.16.20.1), o seu endereço MAC (00:21:5a:5a:78:c7), o endereço IP do destinatário (172.16.20.254) e o endereço MAC do destinatário. Como o endereço MAC do destinatário lhe é desconhecido, ele preenche este campo com 00:00:00:00:00:00. Posteriormente, obtém um pacote de resposta (*reply*) com os endereços de envio do gnu 4, IP (172.16.20.254) e MAC (00:22:64:a7:26:a2), e os endereços de destino do gnu 3, IP (172.16.20.1) e MAC (00:21:5a:5a:78:c7). Pode-se ver um exemplo destes pacotes em 6.3.

Ao se dar *ping* do gnu 4 a partir do gnu 3, os endereços IP e MAC obtidos serão os correspondentes a cada um destes gnu's. Assim sendo, segue-se detalhadamente a informação em relação aos endereços tanto do pacote de pedido como do pacote de resposta:

Pacote de Pedido (ICMP *request*):

	Endereço
IP de Envio	172.16.20.1
MAC de Envio	00:21:5a:5a:78:c7
IP do destinatário	172.16.20.254
MAC do destinatário	00:22:64:a7:26:a2

Pacote de Resposta (ICMP *reply*):

	Endereço
IP de Envio	172.16.20.254
MAC de Envio	00:22:64:a7:26:a2
IP do destinatário	172.16.20.1
MAC do destinatário	00:21:5a:5a:78:c7

Tal como se pode verificar, os pacotes ARP e os pacotes ICMP (6.3) possuem informações diferentes. Deste modo, para se poder distinguir entre eles, utiliza-se a **desmultiplexagem**. O *Ethernet header* presente nas tramas permite-nos distinguir tramas ARP de tramas IP. Além disso, o IP *header* presente nestas últimas permite-nos fazer a distinção entre tramas ICMP e outras tramas IP. Esta distinção também pode ser feita através da leitura dos resultados obtidos no *wireshark*. Analisando os resultados obtidos, é possível verificar-se que, quando o pacote tiver no parâmetro "*type*"o valor 0x0800 (6.3), significa que o tipo da trama é IP. Posteriormente, é possível analisar o IP *header*: se este tiver o valor 1 (6.3), significa que o tipo de protocolo é ICMP. Por outro lado, se o pacote tiver no parâmetro "*type*"o valor 0x0806 (6.3), então a trama é do tipo ARP.

Para se visualizar o comprimento da trama recetora, é necessário consultar o pacote ICMP *reply* e verificar a *frame length*. No caso na experiência realizada pelo nosso grupo de trabalho, esta tem o valor de 98 *bytes* (784 *bits*) (6.3).

A interface de *loopback* é uma interface de rede virtual que permite ao computador enviar pacotes a si próprio. Esta interface é usada para diagnóstico e identificação de problemas na ligação à rede.

3.2 Experiência 2: Implementar duas LAN's virtuais no *switch*

Nesta experiência, o objetivo principal era criar duas LAN's virtuais (`vlany0` e `vlany1`) no *switch*, sendo que os gnu's 3 e 4 foram associados à `vlany0` e o gnu 2 foi associado à LAN `vlany1`.

Toda a configuração feita nesta experiência pode ser observada em 6.2.

Existem dois domínios de broadcast, correspondentes a cada uma das vlans. Ao efetuarmos um *ping* em *broadcast* (*ping -b (...)*) a partir do gnu 3, não obtemos nenhuma resposta, tal como se pode verificar nos registos (6.3). Isto acontece porque, por predefinição, o *ping* em *broadcast* não obtém nenhuma resposta.

3.3 Experiência 3: Configurar um Router em Linux

Na experiência 3, foi configurado o gnu 4 como um *router*, estabelecendo-se, assim, uma ligação entre as duas VLANs criadas na experiência 2. A configuração necessária para a experiência pode ser observada em 6.2.

GNUY2

Destino	Gateway
172.16.Y0.0	172.16.Y1.253
172.16.Y1.0	0.0.0.0

GNUY3

Destino	Gateway
172.16.Y0.0	0.0.0.0
172.16.Y1.0	172.16.Y1.254

GNUY4

Destino	Gateway
172.16.Y0.0	0.0.0.0
172.16.Y1.0	0.0.0.0

Uma tabela de *forwarding* mapeia endereços de destino para os links de saída destes. Assim, contém os endereços IP, e respetiva máscara, de forma a permitir que o pacote, chegado ao router, seja encaminhado para o router adjacente adequado.

Nos registos do gnu 3, podem ser observadas mensagens de ARP com a finalidade de descobrir o endereço MAC para onde reencaminhar os pacotes. Neste caso, o endereço MAC recebido é, em todos os casos, 00:21:5a:5a:74:3e (o endereço MAC do gnu 4 na VLAN y0). O endereço MAC recebido não é, de facto, o endereço MAC do destino, mas sim o endereço MAC do router que permite alcançar o destino.

No caso do primeiro *ping* 172.16.y0.254, o endereço MAC recebido é o endereço MAC do destino. No entanto, tanto no *ping* 172.16.y1.253 e *ping* 172.16.y1.1, o endereço MAC recebido é o endereço MAC do gnu 4 na VLAN y0, que, apesar de não ser o destinatário, estabelece uma ligação entre ambas as VLANs, fornecendo assim um caminho para o reencaminhamento das mensagens destinadas aos computadores gnu 4 e gnu 2, ambos na VLAN y1, inacessíveis ao gnu 3 na VLAN y0. 6.3

Ao efetuar o comando *ping*, também podemos observar pacotes ICMP. Ao observarmos os pacotes ICMP do gnu 3 iremos reparar que, embora o endereço MAC que lhes está associado seja sempre o mesmo (o endereço MAC do gnu 4 na VLAN y0), já que todos os pacotes são encaminhados para este computador, o endereço IP não é sempre igual, visto este depender do destinatário do pacote. Podemos observar dois tipos de pacotes ICMP, pacotes de *request* e pacotes de *reply*. Desta forma, podemos concluir que foi estabelecida uma ligação entre o gnu 3 na VLAN y0 e todas as outras interfaces da rede.

3.4 Experiência 4: Configurar um Router Comercial e Implementar o NAT

Nesta experiência, o objetivo principal era configurar um *router* comercial, inicialmente sem NAT. Posteriormente, configurou-se o *router* com NAT, de modo a permitir o acesso à rede internet a partir dos computadores.

Toda a configuração feita nesta experiência, tanto do *router* como do NAT, pode ser observada em 6.2.

Após a configuração do *router*, os pacotes acabam por seguir as rotas estabelecidas anteriormente. Se uma destas não existir, os pacotes são redirecionados para o *router* (*default route*), e este informa quem enviou o pacote de que existe o gnu 4, sendo o pacote posteriormente enviado através do gnu 4.

O NAT (*Network Address Translation*) é usado na preservação de endereços, sendo usado de forma a reutilizar endereços em múltiplos locais. Este protocolo permite que endereços de IP privados e não registados se conectem à internet através da transformação desses endereços num único endereço público e registado, antes desses pacotes serem encaminhados para a rede pública.

3.5 Experiência 5: DNS

Na experiência 5, o objetivo principal é configurar o serviço DNS. Esta configuração faz-se através da edição do ficheiro `resolv.conf` em todos os os *hosts* da rede criada e pode ser observada em 6.2. O ficheiro `resolv.conf` consiste num ficheiro utilizado em vários sistemas operativos que configura o DNS. A única edição necessária ao ficheiro é inserir `nameserver 172.16.1.1`, que se trata do endereço IP do servidor que deve ser acedido (netlab.fe.up.pt).

O DNS abstrai endereços IP, tornando a sua utilização mais simples. Desta forma, o utilizador não tem de lidar com os endereços IP numéricos nem com alterações nos mesmos (6.3).

Para testar esta configuração, fez-se *ping* usando *www.google.com*. Nos registos verificou-se que, inicialmente, se envia um *request* (6.3) da informação contida neste domínio e, posteriormente, se recebe um pacote *reply* (6.3) com informação, nomeadamente o endereço IP do domínio, comprovando-se que a configuração foi feita corretamente.

3.6 Experiência 6: Conexões TCP

Nesta experiência, o objetivo principal é utilizar a aplicação *download* descrita na primeira parte do relatório para realizar o *download* de um ficheiro.

A configuração desta experiência é igual à experiência 5, tal como se pode observar em 6.2.

Tal como foi explicado anteriormente, na aplicação desenvolvida pelo nosso grupo de trabalho são abertas duas conexões TCP. A primeira (aberta na porta FTP *default*, 21) permite o envio de comandos FTP ao servidor e receber as respetivas respostas, estabelecendo o controlo da informação. A segunda conexão (aberta na porta recebida como reposta ao comando `pasv`) é usada para receber os dados do ficheiro enviado pelo servidor.

Estas conexões são divididas em 3 fases: a abertura da conexão, o envio/receção de informação e o fecho da conexão.

Inicialmente, o cliente envia um comando SYN para iniciar a conexão com o número de sequência X. Posteriormente, o servidor, ao receber este comando, envia de volta para o cliente um novo comando SYN, mas, desta vez, com um número de sequência Y e uma confirmação positiva (ACK) com o valor X+1. De seguida, o cliente, ao receber a confirmação por parte do servidor, envia um novo comando SYN, desta vez com um número de sequência de valor X+1 e uma confirmação positiva com o valor de Y+1.

Após esta troca de mensagens, a ligação entre o cliente e o servidor está estabelecida e pode inicializar-se a transferência de dados. No final da transferência, é necessário fechar a conexão. Este fecho inicia-se com o envio do comando FIN por parte do cliente com o número de sequência A. O servidor, ao receber este comando, envia uma confirmação positiva de valor A+1. De seguida, envia também o comando FIN para o cliente com uma confirmação positiva igual à enviada anteriormente. Ao receber este último comando, o cliente envia uma confirmação positiva de valor B+1. Por fim, a memória alocada para o processo é libertada.

O mecanismo ARQ (*Automatic Repeat Request*) é importante no controlo de erros durante a transmissão de dados. Para isso, utiliza mensagens de *acknowledge* (mensagem enviada pelo recetor indicando a correta receção dos dados) e erros de *timeout* (erro ocurrido quando o tempo para receção de um pacote ACK expirou). O pacote irá ser retransmitido até se atingir um número máximo de retransmissões ou até ser recebido um ACK antes de acontecer *timeout*. Neste caso, o método utilizado é uma variação do *Go-Back-N*. Assim, nos registos, podemos notar os parâmetros:

- *Sequence Number* - identifica o primeiro *byte* de dados
- *Acknowledgment Number* - identifica o último *byte* de dados sem erros
- *Window* - tamanho da janela, utilizado para o *flow control*
- *Checksum* - utilizado para a verificação da existência de erros

O mecanismo de controlo de congestão TCP (6.3,6.3) utilizado é o *Slow Start*. Começando por enviar apenas um pacote, irá aumentar o número de pacotes enviados simultaneamente de forma exponencial. Desta forma, assim que um pacote seja perdido, o mecanismo irá detetar um congestionamento, procedendo de forma a diminuir esse congestionamento e retransmitindo o pacote perdido. No caso de *timeout*, o congestionamento é considerado severo, recomeçando a transmissão, primeiro, em *Slow Start* até à *threshold*, e, depois, em modo de *Congestion Avoidance* até atingir um novo congestionamento. No caso da receção de 3 ACKs, o congestionamento não é considerado severo, pelo que irá imediatamente entrar no método de *Congestion Avoidance*. O método de *Congestion Avoidance* consiste em ir aumentando, incrementalmente, os pacotes transmitidos, até acontecer um congestionamento, diminuindo, nesse momento, o número de pacotes transmitidos simultaneamente para metade.

Desta forma, tal como podemos verificar nos registos e no gráfico (6.3) o ritmo de transferência vai aumentando até se atingir um máximo, no qual ocorre timeout. Em seguida, o ritmo diminui drasticamente, voltando depois a aumentar lentamente até existir um novo pico, e assim sucessivamente, até ao final da transferência.

O aparecimento de uma segunda conexão TCP leva a uma queda na taxa de transmissão, uma vez que a taxa de transferência passará a ser distribuída igualmente entre as duas conexões. 6.3

4 Conclusões

Após a realização do trabalho prático 2 da unidade curricular Redes de Computadores, podemos afirmar que os conhecimentos adquiridos ao longo das aulas teóricas encontram-se agora consolidados, nomeadamente o protocolo FTP, as conexões TCP, a configuração de endereços de IP, a implementação de VLAN's em *switch* e a configuração de um *router* comercial, com e sem NAT, e a configuração de um serviço de DNS.

Assim sendo, consideramos que os objetivos para este trabalho prático foram atingidos com sucesso, uma vez que implementamos uma aplicação *download* funcional numa rede de computadores configurada ao longo das aulas práticas, apesar de todos as dificuldades que acabaram por surgir tendo em conta o contexto atual de pandemia, nomeadamente em relação ao horário reduzido para utilização dos laboratórios e a dificuldade acrescida de apenas frequentarmos o laboratório em metade das aulas, devido à restrição de apenas um dos membros do grupo poder frequentar simultaneamente o laboratório.

5 Referências

FTP Responde Codes

The Network Layer, Powerpoint Aulas Teóricas

ARP, Wireshark Wiki

Forwarding Table, Computer Networking - a Top-Down Approach, by James F. Kurose and Keith W. Ross

Configuring the Interface Properties, CISCO

NAT, CISCO

resolv.conf, Linux Manual Page

DNS, DNS pt

6 Anexos

6.1 Anexo I - Código Fonte

constraints.h

```
#ifndef CONSTRAINTS_H
#define CONSTRAINTS_H

// ------------------------------------------------
// ------------------ CONSTRAINTS -----------------
// ------------------------------------------------

// ------------------ SERVER INFORMATION ----------------
#define SERVER_ADDR          "192.168.28.96"
#define SERVER_PORT          21
// ------------------ MAX LENGTHS -----------------
#define MAX_STRING_LENGTH    50
#define SOCKET_BUFFER_LENGTH 1000
// ------------------ INDEXES FOR PARSINS -----------------
#define FTP_INDEX            0
#define USER_INDEX           1
#define PASSWORD_INDEX       2
#define HOST_INDEX           3
#define PATH_INDEX           4

typedef struct arguments {
    char* user;
    char* pass;
    char* host;
    char* path;
} arguments;

#endif // CONSTRAINTS_H
```

download.h

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>

#include "constraints.h"
#include "auxfunctions.c"

/*
-------------------------------------------------------------------------
```

```
----                           MAIN                                    ----
-------------------------------------------------------------------------
*/

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

    // ---------------------------- VARIABLES ----------------------------

    int socketfd;
    int socketfdClient =-1;
    struct hostent *h;
    int port;
    char ip[17];

    struct arguments args;

    char responsecode[4], string[4];
    char information[MAX_STRING_LENGTH];
    char filename[MAX_STRING_LENGTH]; memset(filename, 0, MAX_STRING_LENGTH);

    // ----------------------------- PARSING -----------------------------

    parseArgs(argv[1], &args);
    parseFilename(args.path, filename);
    h = getIP(args.host);

    printf("--------------------------------\n");
    printf(" * Username: %s\n", args.user);
    printf(" * Password: %s\n", args.pass);
    printf(" * Host: %s\n", args.host);
    printf(" * Path: %s\n", args.path);
    printf(" * Filename: %s\n", filename);
    printf(" * IP Address: %s\n", inet_ntoa(*((struct in_addr *) h->h_addr)));
    printf("--------------------------------\n");

    // ------------------------ INITIAL CONNECTION ------------------------

    connection(inet_ntoa(*((struct in_addr *)h->h_addr)), &socketfd, SERVER_PORT);
    readResponseCode(socketfd,responsecode);

    // 220 - Service ready for new user.
   stringMaker(string, "220");
   if (strcmp(responsecode, string) != 0) {
      printf("Error when Establishing Connection\n");
      return 1;
   }

    // ------------------------------ LOGIN ------------------------------

    printf("> sending username\n");
    sprintf(information, "user %s\n", args.user);
    sendInformation(socketfd, information);
    readResponseCode(socketfd,responsecode);
    // 331 - User name okay, need password
   stringMaker(string, "331");
   if (strcmp(responsecode, string) == 0) {
```

```
        printf("> sending password\n");
        sprintf(information, "pass %s\n", args.pass);
        sendInformation(socketfd, information);
        readResponseCode(socketfd,responsecode);
    }
     // 230 - User logged in, proceed. Logged out if appropriate.
    stringMaker(string, "230");
    if (strcmp(responsecode, string) != 0) {
       printf("Error when Logging in.\n");
       return 1;
    }

     // ------------------------ ENTER PASSIVE MODE ------------------------

     printf("> sending passive\n");
     sprintf(information, "pasv\n");
     sendInformation(socketfd, information);

     readIPPort(socketfd, responsecode, ip, &port);
     // 227 - Entering Passive Mode (h1,h2,h3,h4,p1,p2).
    stringMaker(string, "227");
    if (strcmp(responsecode, string) != 0) {
       printf("Error Entering Passive Mode.\n");
       return 1;
    }

     // ------------------------- SECOND CONNECTION ------------------------

     connection(inet_ntoa(*((struct in_addr *)h->h_addr)), &socketfdClient, port);

     // --------------------------- DOWNLOAD FILE --------------------------

     printf("> sending retrieve\n");
     sprintf(information, "retr %s\n", args.path);
     sendInformation(socketfd, information);
     readResponseCode(socketfd,responsecode);

     // 150 - File status okay; about to open data connection.
    stringMaker(string, "150");
    if (strcmp(responsecode, string) != 0) {
       printf("Error Sending Retrieve Command.\n");
       return 1;
    }

     printf("> starting download\n");
     downloadFile(socketfdClient, filename);
     printf("> download complete\n");

     // --------------------------- CLOSE SOCKETS --------------------------

     close(socketfd);
     close(socketfdClient);

     return 0;
}
```

auxfunctions.h

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>

#include "constraints.h"
#include "parsing.c"

/**
 * Function that returns the IP auxiliary information from host
 * Based on file: getip.c
 **/
struct hostent * getIP(char host[]) {
    struct hostent * h;
    if ((h = gethostbyname(host)) == NULL) {
        herror("gethostbyname");
        exit(1);
    }
    return h;
}

/**
 * Function that makes the connection with the server address
 * Based on file: clientTCP.c
 **/
int connection(char * ip, int * socketfd, int port) {
   struct sockaddr_in server_addr;

    /* server address handling */
   bzero((char *) & server_addr, sizeof(server_addr));
   server_addr.sin_family = AF_INET;
   server_addr.sin_addr.s_addr = inet_addr(ip);
   /* 32 bit Internet address network byte ordered */
   server_addr.sin_port = htons(port);
    /*server TCP port must be network byte ordered */

    /* open a TCP socket */
   if ((*socketfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      perror("socket()");
      return 1;
   }

    /* connect to the server */
    if (connect(*socketfd, (struct sockaddr *) &server_addr,
        sizeof(server_addr)) < 0) {
      perror("connect()");
      return 1;
   }
    return 0;
```

```
}

/**
 * Response Code String
 */
void stringMaker(char * string, char * maker) {
    string[0] = maker[0];
    string[1] = maker[1];
    string[2] = maker[2];
    string[3] = 0;
}

/**
 * Sends information through socket
 **/
int sendInformation(int fd, char * information){
    int s = send(fd, information, strlen(information), 0);
    return s;
}

/**
 * Reads the response from the socket
 **/
void readResponse(int fd, char * buf) {
    do {
        bzero(buf, SOCKET_BUFFER_LENGTH);
        recv(fd, buf, SOCKET_BUFFER_LENGTH, 0);
    } while (buf[3] != ' ');
}

/**
 * Reads the response code from the socket
 */
void readResponseCode(int fd, char * response) {
    char buf[SOCKET_BUFFER_LENGTH];
    readResponse(fd, buf);
    stringMaker(response, buf);
    printf("\t< %s", buf);
}

/**
 * Reads the response from the socket
 * Determines the port
 */
void readIPPort(int fd, char * response, char * ip, int * port) {
    char buf[SOCKET_BUFFER_LENGTH];
    readResponse(fd, buf);
    stringMaker(response, buf);
    printf("\t< %s", buf);
    parseIPPort(buf, ip, port);
}

/**
 * Creates the file with the data that has to be downloaded
 **/
void downloadFile(int fd, char * filename) {
```

```
    FILE *file = fopen((char *)filename, "wb+");

   char buf[SOCKET_BUFFER_LENGTH];
   int size;
   while ((size = recv(fd, buf, SOCKET_BUFFER_LENGTH, 0)) != 0) {
      fwrite(buf, size, 1, file);
   }

    fclose(file);

    printf("> finished downloading file\n");
}
```

parsing.h

```
#include <string.h>

#include "constraints.h"

// ------------------------------------------------
// ---------------- PARSING FUNCTIONS -------------
// ------------------------------------------------

/**
 * Parse the argument given into user, password, host and path
**/
void parseArgs(char * arguments, struct arguments * args){
    // ./download ftp://[<user>:<password>@]<host>/<url-path>
    char * ftp = strtok(arguments, "/");
    if (strcmp(ftp, "ftp:") != 0) {
        perror("not using ftp");
        exit(1);
    }
    char *aux = strtok(NULL, "/");
    char *path = strtok(NULL, "");
    char *user = strtok(aux, ":");
    char *pass = strtok(NULL, "@");
    char *host;

    if (pass == NULL) {
        //anonymous user
        user = "anonymous";
        pass = "kjewbkwfbe";
        host = aux;
    }
    else {
        host = strtok(NULL, "");
    }
    args->user = user;
    args->pass = pass;
    args->host = host;
    args->path = path;
}

/**
```

```
 * Parse the path given to get the filename
**/
void parseFilename(char * path, char * filename) {
    int indexFilename = 0;
    memset(filename, 0, MAX_STRING_LENGTH);

    for (size_t indexPath = 0; indexPath < strlen(path); indexPath++){
        if (path[indexPath] == '/') {
            indexFilename = 0;
            memset(filename, 0, MAX_STRING_LENGTH);
        }
        else {
            filename[indexFilename] = path[indexPath];
            ++ indexFilename;
        }
    }
}

/**
 * Parse IP and Port read in passive mode
 *
 * FTP server subcommand
 * PORT h1,h2,h3,h4,p1,p2
 *
 * h n
 * Represents the system IP address and is a character string that is a decimal value
     between 0 and 255.
 *
 * p n
 * Represents the TCP port number and is a character string that is a decimal value between
     0 and 255.
 *
 * To convert the p1 and p2 values to a TCP port number, use this formula:
 * port = ( p1 * 256 ) + p2
 *
 * For example, in this PORT subcommand:
 *     PORT 9,180,128,180,4,8
 * the port number is 1032 and the IP address is 9.180.128.180.
 **/
void parseIPPort(char * buf, char * ip, int * port) {
    strtok(buf, "(");
    char* h1 = strtok(NULL, ",");
    char* h2 = strtok(NULL, ",");
    char* h3 = strtok(NULL, ",");
    char* h4 = strtok(NULL, ",");
    char* p1 = strtok(NULL, ",");
    char* p2 = strtok(NULL, ")");

    sprintf(ip, "%s.%s.%s.%s", h1, h2, h3, h4);

    *port = atoi(p1)*256 + atoi(p2);
}
```

6.2 Anexo II - Configurações

Devido às dificuldades que surgiram perante a situação atual de contexto de pandemia, o nosso grupo de trabalho teve dificuldades em obter registos para todas as experiências. Assim sendo, os registos analisados na experiência 1 foram obtidos por nós (Sala 320, Bancada 2) e nas restantes experiências analisamos os registos obtidos pelo grupo T6G8 (Sala 320, Bancada 3).

Experiência 1

Cabos

```
GNUY3E0 -> Switch Port 1
GNUY4E0 -> Switch Port 2
```

gnuY3

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.1/24
```

gnuY4

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.254/24
```

Experiência 2

Cabos

```
GNUY3E0 -> Switch Port 1
GNUY4E0 -> Switch Port 2
GNUY2E0 -> Switch Port 3
```

gnuY2

```
ifconfig eth0 up
ifconfig eth0 172.16.Y1.1/24
```

gnuY3

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.1/24
```

gnuY4

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.254/24
```

Switch

```
configure terminal
vlan y0
end

configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan y0
end
configure terminal
interface fastethernet 0/2
switchport mode access
switchport access vlan y0
end

configure terminal
vlan y1
end

configure terminal
interface fastethernet 0/3
switchport mode access
switchport access vlan y1
end
```

Experiência 3

Cabos

```
GNUY3E0 -> Switch Port 1
GNUY2E0 -> Switch Port 2
GNUY4E0 -> Switch Port 3
GNUY4E1 -> Switch Port 4
```

gnuY2

```
ifconfig eth0 up
ifconfig eth0 172.16.Y1.1/24

route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
```

gnuY3

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.1/24

route add -net 172.16.Y1.0/24 gw 172.16.Y0.254
```

gnuY4

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.254/24
```

```
ifconfig eth1 up
ifconfig eth1 172.16.Y1.253/24

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Switch

```
configure terminal
vlan y0
end

configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan y0
end

configure terminal
interface fastethernet 0/2
switchport mode access
switchport access vlan y0
end

configure terminal
vlan y1
end

configure terminal
interface fastethernet 0/3
switchport mode access
switchport access vlan y1
end

configure terminal
interface fastethernet 0/4
switchport mode access
switchport access vlan y1
end
```

Experiência 4

Cabos

```
GNUY3E0 -> Switch Port 1
GNUY2E0 -> Switch Port 2
GNUY4E0 -> Switch Port 3
GNUY4E1 -> Switch Port 4
ROUTERE0 -> Switch Port 5
ROUTERE1 -> Shelf Port 1
```

gnuY2

```
ifconfig eth0 up
ifconfig eth0 172.16.Y1.1/24

route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
```

gnuY3

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.1/24

route add -net 172.16.Y1.0/24 gw 172.16.Y0.254
```

gnuY4

```
ifconfig eth0 up
ifconfig eth0 172.16.Y0.254/24

ifconfig eth1 up
ifconfig eth1 172.16.Y1.253/24

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Switch

```
configure terminal
vlan y0
end

configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan y0
end

configure terminal
interface fastethernet 0/2
switchport mode access
switchport access vlan y0
end

configure terminal
vlan y1
end

configure terminal
interface fastethernet 0/3
switchport mode access
switchport access vlan y1
end

configure terminal
```

```
interface fastethernet 0/4
switchport mode access
switchport access vlan y1
end

configure terminal
interface fastethernet 0/5
switchport mode access
switchport access vlan y1
end
```

Router

```
conf t
interface fastethernet 0/0
ip address 172.16.y1.254 255.255.255.0
no shutdown
ip nat inside
exit

interface fastethernet 0/1
ip address 172.16.1.y9 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.1.y9 172.16.1.y9 prefix 24
ip nat inside source list 1 pool ovrld overload

access-list 1 permit 172.16.y0.0 0.0.0.7
access-list 1 permit 172.16.y1.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.y0.0 255.255.255.0 172.16.y1.253
end
```

Experiência 5

Repetir passos da experiência 4, mas adicionar os seguintes comandos aos gnu's 2, 3 e 4.

```
echo $'search netlab.fe.up.pt\nnameserver 172.16.2.1' > /etc/resolv.conf
```

Experiência 6

Repetir passos da experiência 5.

6.3 Anexo III - Imagens

```
------------------------------
 * Username: rcom
 * Password: rcom
 * Host: netlab1.fe.up.pt
 * Path: files/pic1.jpg
 * Filename: pic1.jpg
 * IP Address: 192.168.109.136
------------------------------
        < 220 Welcome to netlab-FTP server
> sending username
        < 331 Please specify the password.
> sending password
        < 230 Login successful.
> sending passive
        < 227 Entering Passive Mode (192,168,109,136,162,129).
> sending retrieve
        < 150 Opening BINARY mode data connection for files/pic1.jpg (340603 bytes).
> starting download
> finished downloading file
> download complete
```

Figura 1: Exemplo de execução da aplicação de *download*

```
17 20.404082777  HewlettP_5a:7d:b7   Broadcast           ARP      42 Who has 172.16.30.254? Tell 172.16.30.1
18 20.404221272  HewlettP_5a:74:3e   HewlettP_5a:7d:b7   ARP      60 172.16.30.254 is at 00:21:5a:5a:74:3e
```

Figura 2: Exemplo de pacotes ARP

```
19 20.404236218  172.16.30.1      172.16.30.254    ICMP   98 Echo (ping) request  id=0x249e, seq=1/256, ttl=64 (reply in 20)
20 20.404371849  172.16.30.254    172.16.30.1      ICMP   98 Echo (ping) reply    id=0x249e, seq=1/256, ttl=64 (request in 19)
```

Figura 3: Exemplo de pacotes ICMP

```
Ethernet II, Src: HewlettP_5a:7d:b7 (00:21:5a:5a:7d:b7), Dst: HewlettP_5a:74:3e (00:21:5a:5a:74:3e)
  > Destination: HewlettP_5a:74:3e (00:21:5a:5a:74:3e)
  > Source: HewlettP_5a:7d:b7 (00:21:5a:5a:7d:b7)
    Type: IPv4 (0x0800)
```

Figura 4: *Type* de IP

```
Internet Protocol Version 4, Src: 172.16.30.1, Dst: 172.16.30.254
   0100 .... = Version: 4
   .... 0101 = Header Length: 20 bytes (5)
 > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   Total Length: 84
   Identification: 0xa6ee (42734)
 > Flags: 0x40, Don't fragment
   Fragment Offset: 0
   Time to Live: 64
   Protocol: ICMP (1)
   Header Checksum: 0xfe9a [validation disabled]
   [Header checksum status: Unverified]
   Source Address: 172.16.30.1
   Destination Address: 172.16.30.254
```

Figura 5: *Protocol* de ICMP

```
Ethernet II, Src: HewlettP_5a:7d:b7 (00:21:5a:5a:7d:b7), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
 > Source: HewlettP_5a:7d:b7 (00:21:5a:5a:7d:b7)
   Type: ARP (0x0806)
```

Figura 6: *Type* de ARP

```
Frame 19: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
 > Interface id: 0 (eth0)
   Encapsulation type: Ethernet (1)
   Arrival Time: Nov 24, 2020 15:37:18.719711348 Hora padrão de GMT
   [Time shift for this packet: 0.000000000 seconds]
   Epoch Time: 1606232238.719711348 seconds
   [Time delta from previous captured frame: 0.000014946 seconds]
   [Time delta from previous displayed frame: 0.000014946 seconds]
   [Time since reference or first frame: 20.404236218 seconds]
   Frame Number: 19
   Frame Length: 98 bytes (784 bits)
   Capture Length: 98 bytes (784 bits)
   [Frame is marked: False]
   [Frame is ignored: False]
   [Protocols in frame: eth:ethertype:ip:icmp:data]
   [Coloring Rule Name: ICMP]
   [Coloring Rule String: icmp || icmpv6]
```

Figura 7: Exemplo da *frame length* de um pacote ICMP

```
     9 12.695611572  HewlettP_5a:7d:b7   Broadcast           ARP      60 Who has 172.16.30.254? Tell 172.16.30.1
    10 12.695637902  HewlettP_5a:74:3e   HewlettP_5a:7d:b7   ARP      42 172.16.30.254 is at 00:21:5a:5a:74:3e
    24 17.776757436  HewlettP_5a:74:3e   HewlettP_5a:7d:b7   ARP      42 Who has 172.16.30.1? Tell 172.16.30.254
    25 17.776876096  HewlettP_5a:7d:b7   HewlettP_5a:74:3e   ARP      60 172.16.30.1 is at 00:21:5a:5a:7d:b7
```

Figura 8: Pacotes ARP enviados entre o gnuY4 e o gnuY3

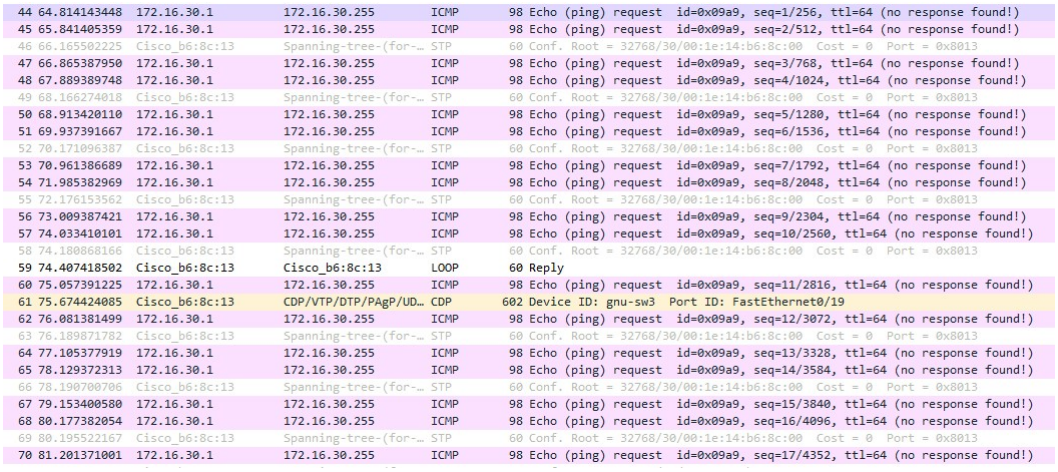

Figura 9: Exemplo de Pacotes ICMP sem resposta

Figura 10: Pacotes DNS

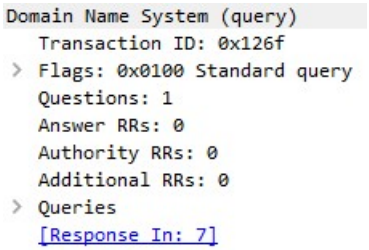

Figura 11: Parâmetros de um pacote DNS (1)

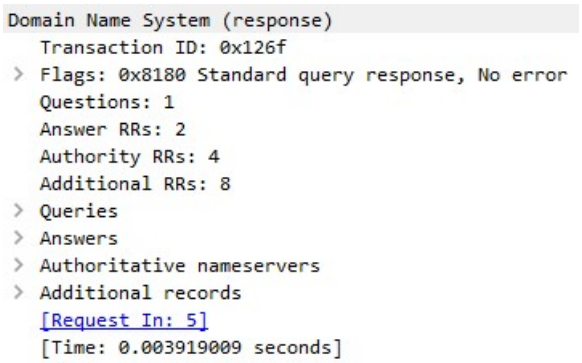

Figura 12: Parâmetros de um pacote DNS (2)

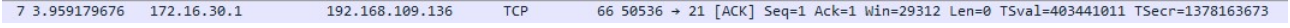

Figura 13: Exemplo de um pacote TCP

```
Transmission Control Protocol, Src Port: 21, Dst Port: 50536, Seq: 92, Ack: 26, Len: 0
   Source Port: 21
   Destination Port: 50536
   [Stream index: 0]
   [TCP Segment Len: 0]
   Sequence Number: 92    (relative sequence number)
   Sequence Number (raw): 3623069829
   [Next Sequence Number: 92    (relative sequence number)]
   Acknowledgment Number: 26    (relative ack number)
   Acknowledgment number (raw): 4272587140
   1000 .... = Header Length: 32 bytes (8)
 ⌄ Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····A····]
   Window: 510
   [Calculated window size: 65280]
   [Window size scaling factor: 128]
   Checksum: 0x2ab7 [unverified]
   [Checksum Status: Unverified]
   Urgent Pointer: 0
```

Figura 14: Informação num pacote TCP

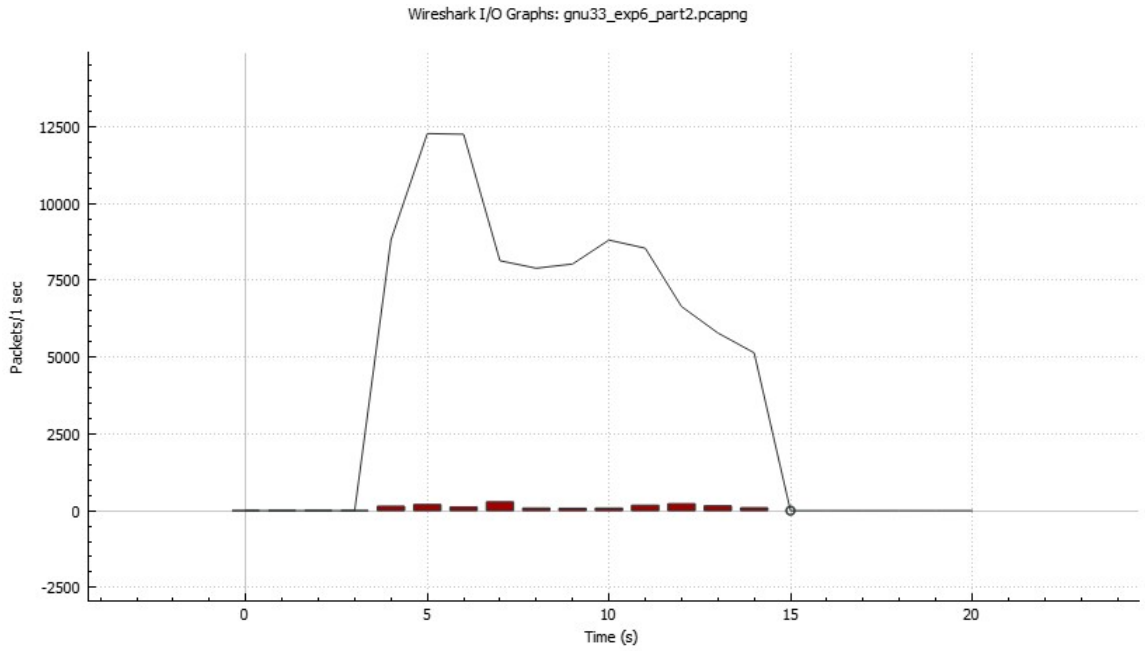

Figura 15: Taxa de transferência FTP