**Faculdade de Engenharia da Universidade do Porto**
2021/22


# Ligação de dados

## 2º trabalho laboratorial

Redes de Computadores

Licenciatura em Engenharia Informática e Computação

**Turma 6 grupo 7**
José Macedo (up201705226)
Marta Mariz (201907020)

# Sumário

Este relatório contextualiza todo o segundo trabalho  laboratorial realizado no âmbito da cadeira Redes de Computadores. Este consiste na implementação de um cliente FTP, bem como na configuração de uma rede de computadores , tendo sido utilizados comandos de configuração do Router Cisco e do Cisco Switch. O projeto foi concluído com sucesso, tendo sido desenvolvida uma aplicação capaz de transferir um ficheiro de um protocolo FTP. Para além disso fomos capazes de, com sucesso configurar e apresentar uma rede de computadores em laboratório.

# Introdução

O relatório divide-se em 3 partes:

- **Parte 1 Download application**, onde é descrito como foi implementada a aplicação FTP.
- **Parte 2, Network Configuration and Analysi**s, onde são descritas todas as experiências propostas, assim como os seus resultados.
- **Conclusão**, um breve resumo das ideias principais retidas.

# Parte 1 - Download application

A nossa aplicação **download** é uma simples aplicação FTP que recebe um ficheiro específico usando o protocolo FTP como descrito em RFC959.

### Arquitetura da aplicação de download

A aplicação encontra-se dividida entre duas pastas, **lib** onde estão as *header files* e **src** onde estão os ficheiros de código **.c**.

A execução começa no ficheiro **app.c**, onde valida o número de argumentos da função e procede ao *parser* do *url* através de funções no ficheiro **parser.c**.
Depois do *parser* é chamada a função que vai ser responsável pela logica de realizar downloads através de uma conexão TCP a um *socket* do servidor.
Conexão essa realizada na função do ficheiro **clientTCP.c** com o *ip* identificado pela função do ficheiro **getip.c** que a partir do *host name* fornecido no *url*.
Depois da conexão TCP o programa recebe a *welcome message* do servidor e procede a autenticação com dados default ou inseridos pelo utilizador quando introduz o *url*.
Entra em modo passivo e procede à transferencia do ficheiro byte a byte.
Quando termina a transferência termina todas as conexões TPC.

Relato de um download bem-sucedido

A nossa aplicação deve receber como argumento que adote uma sintaxe de URL, como descrito na RFC1738.

> Exemplo: *download ftp://ftp.up.pt/pub/kodi/timestamp.txt*
> Onde o *path* do URL é *ftp:///[<user>:<password>@]<host>/<url-path>*

Quando os argumentos são corretos a nossa aplicação FTP abre uma ligação TCP para controlo e transfere o ficheiro que se pretende transferir. Esta ligação de dados é feita no modo passivo.

# Parte 2 - Network Configuration and Analysis

## Experiência 1 - Configurar uma rede IP

Esta experiência tem como objetivos perceber o que são pacotes ARP e para que servem, que tipo de pacotes é que o comando ping gera e o que são endereços MAC e endereços IP.
Os endereços MAC são os identificadores das placas de rede, enquanto que os endereços IP servem como identificadores públicos que cada máquina necessita de usar numa rede para poder comunicar com outras máquinas. Uma máquina pode possuir vários endereços IP, mas apenas 1 endereço MAC.
Nesta experiência o comando ping serve para descobrir se existe conectividade entre computadores. Assim, o tux3 sabe o ip do tux4, mas não sabe o seu endereço MAC. Para descobrir, o tux3 envia em broadcast um pacote ARP que contém o endereço de IP e espera uma resposta com o endereço MAC que lhe corresponde.
Foi exatamente isto que observamos na captura de logs. Como está apresentado na imagem, o tux3 envia em broadcast o pedido do endereço MAC correspondente ao IP indicado. A seguir, a ligação está assegurada e conseguimos observar o envio de pacotes do protocolo ICMP.
Para diferenciar os pacotes ARP, IP e ICMP, é necessário aferir o cabeçalho da trama Ethernet, sendo que os pacotes IP contém, ainda, informação acerca do tamanho da trama.
Loopback é uma interface de rede virtual que é sempre conectável, desde que pelo menos uma das interfaces ip no switch esteja operacional. Assim, pode-se sempre dar ping à loopback interface para verificar conectividade, o que é muito útil no debuuging.

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 3.555311041 | HewlettP_a7:32:ab | Broadcast | ARP | 42 | Who has 172.16.60.254? Tell 172.16.60.1 |
| 4 | 3.555452122 | HewlettP_5a:75:bb | HewlettP_a7:32:ab | ARP | 60 | 172.16.60.254 is at 00:21:5a:5a:75:bb |
| 5 | 3.555461202 | 172.16.60.1 | 172.16.60.254 | ICMP | 98 | Echo (ping) request id=0x0b00, seq=1/256, ttl=64 (reply in 6) |
| 6 | 3.555600397 | 172.16.60.254 | 172.16.60.1 | ICMP | 98 | Echo (ping) reply id=0x0b00, seq=1/256, ttl=64 (request in… |

Figura 1- Captura de logs da utilização do comando ping do tux3 para o tux4.

# Experiência 2 - Implementar duas LAN’s virtuais no switch

O objetivo desta experiência é criar duas VLAN’s no switch, uma associada ao tux3 e ao tux4 e a outra apenas ao tux2.
Para configurar VLAN’s, foi necessário primeiro criar uma ethernet vlan, para isso foi usado o comando < **vlan [n]** > , onde n é o identificador da VLAN. Depois é precido adicionar portas do switch às respectivas VLAN’s, o que é feito a partir do comando <interface fastethernet 0/i>, em que i é o número da porta switch, seguidos dos comandos <switchport mode access> <switchport access vlan [n]> em que n é mais uma vez o número identificador da vlan.
Utilizando ping no tuxy3 para o tuxy4 obtivemos respostas o que indica conectividade, como está exemplificado na figura.


Figura 2- Captura de logs da utilização do comando ping do tux3 para o tux4.
No tuxy3 foi feito um ping broadcast, que não obteve resposta, apesar de ser expectável obter resposta do tuxy4 pois estão na mesma sub rede. Pode ser explicado porque echo-ignore-broadcast está ativado.
No tuxy2 foi feito um ping broadcast, que tal como no anterior não obteve resposta, mas aqui já era esperado, este é o único dispositivo ligado à VLANY1.
Assim podemos concluir que há dois domínios de broadcast, um para a VLANY0 e outro para a VLANY1.

# Experiência 3 - Configuração do router

## Configuração do router cisco

Para configurar a rota estática num router comercial é necessário abrir a consola do router e correr os seguintes comandos:
router> config terminal
router> ip route [destino] [máscara] [gateway]
router > exit
Para configurar um router comercial NAT é necessário abrir a consola do router e correr os seguintes comandos: (Sendo Y o numero da bancada e W a sala do laboratório)
router> conf t
router> interface FastEthernet0/0
router> ip address 172.16.2.Y9 255.255.255.0
router> no shutdown
router> ip nat outside
router> exit
router> interface FastEthernet0/1
router> ip address 172.16.Y1.254 255.255.255.0
router> no shutdown
router> ip nat inside
router> exit
router> ip nat pool ovrld 172.16.W.Y9 172.16.W.Y9 prefix 24
router> ip nat inside source list 1 pool ovrld overload
router> access-list 1 permit 172.16.Y0.0 0.0.0.7
router> access-list 1 permit 172.16.Y1.0 0.0.0.7
router> ip route 0.0.0.0 0.0.0.0 172.16.W.254
router> ip route 172.16.Y0.0 255.255.255.0 172.16.Y1.253
router> end

NAT (Network Address Translation) com o nome indica é a tradução de endereços de rede. É uma técnica de mapear vários endereços privados locais para um público antes de transferir as informações.

## DNS config

Executando ping para youtubas foi possível encontrar packets DNS como mostra na figura associada a este ponto.  Para configurar o serviço de DNS no host é necessário alterar o ficheiro etc/resolv.conf . Assim,enquanto que ao dar ping para ensina.europa o endereço de destino da querie DNS era 192.168.1.1, no ponto 4, ao mudar o ficheiro /etc/resolv.conf passou a ser 9.9.9.9.

Os pacotes trocados com o DNS são pacotes com a informação do hostname ou o endereço IP.
O DNS converte hostnames em endereços IP .

## Linux Routing

ICMP (Internet Control Message Protocol) packets são packets IP com ICMP na porção de dados IP.
Os packets ICMP são messagens de controlo e são observados packets de request e reply quando o computador tenta contactar com a rota adicionada, se não conseguir comunicar com essa rota falha e o packet observado é um Host Unreachable .

Os endereços IP e MAC associados ao ICMP são os endereços do computador que realiza o request e o destino para o request é feito.
As mensagens ICMP também contêm todo o cabeçalho IP da mensagem original para que o sistema final saiba qual pacote falhou.
As única rota que existia no computador é uma default gateway, que server de ponto de acesso a outras redes.

## Experiência 4 - Configuração do router (Lab)

O tux2 tem uma default gw para 172.16.61.254 que permite o tux2 comunicar com a rede do Cisco Router, e uma rota para 172.16.60.0/24 atravez da gw 172.16.61.253 que permite ao tux2 saber conectar-se a rede do tux3 a através da gw do tux4. O tux3 tem uma default gw para 172.16.60.254 para o tux4. O tux4 tem uma default gw para 172.16.61.254 que permite o tux4 comunicar com a rede do Cisco Router.
Contem a configuração das rotas do router, no nosso caso: Rota que serve ponto de acesso a outras redes a rede do router: ip route 0.0.0.0 0.0.0.0 172.16.2.254 Rota que serve para o router saber como chegar a VLANX0, através da gw do tux4: ip route 172.16.60.0 255.255.255.0 172.16.61.253.
Verifica-se que a interface eth0 do tuxy4 enviou dois pedidos ARP para determinar o endereço MAC da interface eth0 do tuxy3, enquanto que o tuxy3 mandou um pedido para saber o endereço MAC da interface eth0 do tuxy4. Também existe trocas de mensagens ARP para o tuxy2 tomar conhecimento do endereço MAC da interface eth1 do tuxy4 e vice versa.
Os endereços IP e MAC associados ao ICMP são os endereços do computador que realiza o request e o destino para o request é feito. As mensagens ICMP também contêm todo o cabeçalho IP da mensagem original para que o sistema final saiba qual pacote falhou.

## Conclusão

As experiências desenvolvidas ajudaram para uma melhor compreensão dos objetivos do projeto, a realização de uma aplicação que segue o protocolo ftp para download de ficheiros e a configuração de uma rede para testar a aplicação.

Ambos os objetivos foram concretizados, a aplicação permite transferir ficheiros e a rede foi configurada corretamente.

O desenvolvimento da aplicação permitiu compreender melhor o protocolo ftp, e a configuração da rede permitiu obter um conhecimento mais aprofundado de como funciona uma rede, desde como configurar a ethernet num computador, a configurar VLANs e o router.

# Annexes

## Código da aplicação de download

app.c

```
#include "../lib/app.h"

int main(int argc, char** argv)
{
    if (argc != 2 && strcmp(argv[0], "download") != 0)
    {
        printf("Usage: download
ftp://[<user>:<password>@]<host>/<url-path>\n");
        return -1;
    }

    ftp_url args;

    // parse url
    if(parse_url(argv[1], &args) != 0)
    {
        printf("Error parsing the url\n");
        printf("Usage: download
ftp://[<user>:<password>@]<host>/<url-path>\n");
        return -1;
    }

    printf("\nSuccessful parse URL\n\n");

    // file transfer
    if(file_transfer(args) != 0)
    {
        printf("Error transfering the file\n");
        return -1;
    }

    printf("\nSuccessful file transfer\n");

    return 0;
}
```

parser.c

```c
#include "../lib/parser.h"

int parse_url(const char* url, ftp_url* args){

    char* app_protocol = strtok(url, "//");

    // check if it is an FTP Application Protocol
    if( strcmp(app_protocol, "ftp:") || app_protocol == NULL)
    {
        printf("Application protocol not accepted, pls choose ftp
protocol \n");
        return -1;
    }

    // url first part (user, pass and host)
    char* user_pass_host = strtok(NULL, "/");
    if(user_pass_host == NULL){
        printf("URL wrong format \n");
        return -1;
    }

    // url second part (path)
    char* path = strtok(NULL, " ");
    if(path == NULL){
        printf("URL wrong format \n");
        return -1;
    }

    // parsing the user, password and host
    if(parse_url_first_part(user_pass_host, args) != 0)
    {
        return -1;
    }

    // save the path in the struct
    strcpy(args->path, path);

    return 0;

}
```

```c
int parse_url_first_part(char* user_pass_host, ftp_url* args){

    // storing the first_part to confirm latter
    char first_part[63+63+255];
    strcpy(first_part, user_pass_host);

    char *user = strtok(user_pass_host, ":");

    if(user == NULL)
    {
        printf("Wrong URL format\n");
        return -1;
    }

    // no user if the ':' doesn't exists in the initial phrase
    if(strcmp(user, first_part) == 0)
    {
        // save the user in the struct
        strcpy(args->user, "anonymous");
        // save the pass in the struct
        strcpy(args->pass, "");
        // save the host in the struct
        strcpy(args->host, user);
    }
    else
    {
        // save the user in the struct
        strcpy(args->user, user);

        char *pass = strtok(NULL, "@");

        if(pass == NULL)
        {
            printf("Wrong URL format\n");
            return -1;
        }

        // save the pass in the struct
        strcpy(args->pass, pass);

        char *host = strtok(NULL, "");
        if(user == NULL)
        {
```

```
            printf("Wrong URL format\n");
            return -1;
        }

        // save the host in the struct
        strcpy(args->host, host);
    }
    return 0;
}

int parse_server_response(const char* response_string,
ftp_server_response* response, const char* delimiter){

    char response_string_cpy[SERVER_RESPONSE_LEN];

    // copy response_string to use on strtok
    strcpy(response_string_cpy, response_string);

    // response code
    char* code = strtok(response_string_cpy, delimiter);

    if(code == NULL){
        printf("Server response code error\n");
        return -1;
    }

    // response description
    char* description = strtok(NULL, "");

    if(description == NULL){
        description = "No descripiton";
    }

    // save the code as an integer and description in struct
    response->code = atoi(code);
    strcpy(response->description, description);

    return 0;
}
```

```
int parse_filename(char* path, char* filename){

    char *path_component = malloc(PATH_LEN);
    path_component = strtok(path, "/");

    if(path_component == NULL){
        printf("URL wrong format \n");
        free(path_component);
        return -1;
    }

    do
    {
        strcpy(filename, path_component);
        path_component = strtok(NULL, "/");

    }while(path_component != NULL);

    free(path_component);

    return 0;
}
```

ftp.c

```
#include "../lib/ftp.h"

int file_transfer(ftp_url args){

    char ip[30];
    get_host_ip(args.host, ip);

    int socketfd;
    if((socketfd = connection_TCP(ip, FTP_PORT)) < 0){
        printf("Error in establishing TCP connection\n");
        return -1;
    }

    printf("TCP connection established!\n");


    ftp_server_response welcome_response;

    // receive welcome response

    if(receive_server_response(socketfd, &welcome_response) != 0 ||
welcome_response.code != READY_FOR_NEW_USER) {
        printf("Server Response error\n");
        close(socketfd);
        return -1;
    }

    // authenticate the user

    if(authenticate_user(socketfd, args.user, args.pass))
    {
        printf("Authentication error\n");
        close(socketfd);
        return -1;
    }

    printf("\nSuccessful authentication\n");

    // entering passive mode

    int pasv_socketfd = enter_pasv_mode(socketfd);
```

```
    if(pasv_socketfd < 0)
    {
        printf("Passive Mode error\n");
        close(socketfd);
        return -1;
    }

    printf("\nSuccessful Passive Mode\n");

    // retrieve the file (Download)

    if(retr_file(socketfd, pasv_socketfd, args.path)){
        printf("Retrieving File error\n");
        close(socketfd);
        close(pasv_socketfd);
        return -1;
    }

    printf("\nSuccessful Retrieved File\n");

    // close the TCP connections
    if (close(socketfd)<0 || close(pasv_socketfd)<0) {
        perror("close(socket)");
        exit(-1);
    }


    return 0;

}
```

```
int receive_server_response(int sockfd, ftp_server_response* response){

    char response_string[SERVER_RESPONSE_LEN];

    if(receive_server_response_string(sockfd, response_string))
    {
        return -1;
    }

    printf("\nString received by server:\n%s\n", response_string);


    if(parse_server_response(response_string, response, " ") != 0 &&
parse_server_response(response_string, response, "-") != 0)
    {
        return -1;
    }


    return 0;
}
```

```
// reads a string from the server
int receive_server_response_string(int sockfd, char* response){

    char buf[1]; int i=0;

    // reads a byte from the server
    int ret = read(sockfd, buf, 1);

    if(ret < 0){
        perror("read");
        return -1;
    }

    response[i++] = buf[0];

    // ends when it reads \n
    while(buf[0] != '\n')
    {
        ret = read(sockfd, buf, 1);
        if(ret < 0){
            perror("read");
            return -1;
        }
        else if(ret == 0) break;

        response[i++] = buf[0];
    }

    response[i] = '\0';

    return 0;
}
```

```
int send_command(int sockfd, char* command){

    // sends a string to the server
    int ret = write(sockfd, command, strlen(command));

    if(ret < 0){
        perror("write");
        return -1;
    }

    printf("\nString send to the server:\n%s\n", command);

    return 0;
}

int authenticate_user(int socketfd ,char* user, char* pass){

    char user_command[USER_LEN + 10];
    char pass_command[PASS_LEN + 10];
    ftp_server_response user_input_response;
    ftp_server_response pass_input_response;

    // user and password commands to send to the server
    sprintf(user_command,"user %s\n", user);
    sprintf(pass_command,"pass %s\n", pass);


    // sends the user
    if(send_command(socketfd, user_command))
    {
        return -1;
    }

    if(receive_server_response(socketfd, &user_input_response))
    {
        return -1;
    }


    // Ends after the welcome messages
    while(user_input_response.code == READY_FOR_NEW_USER)
    {
        if(receive_server_response(socketfd, &user_input_response))
```

```
        {
            return -1;
        }

    }

    // User logged in, proceed
    if(user_input_response.code == USER_LOGIN_OK) return 0;

    int num_tries = 0;

    // Tries to receive "User name okay, need password" response
MAX_TRIES
    while(user_input_response.code != NEED_PASSWORD && num_tries <
MAX_TRIES)
    {
        printf("\nUnexpected response from server. Trying again in 1
sec...\n");
        sleep(1);

        // sends the user again
        if(send_command(socketfd, user_command))
        {
            return -1;
        }

        if(receive_server_response(socketfd, &user_input_response))
        {
            return -1;
        }

        num_tries++;
    }


    // sends the password
    if(send_command(socketfd, pass_command))
    {
        return -1;
    }

    if(receive_server_response(socketfd, &pass_input_response))
    {
```

```
        return -1;
    }


    num_tries = 0;

    // Tries to receive "User logged in, proceed." response MAX_TRIES
    while(pass_input_response.code != USER_LOGIN_OK && num_tries <
MAX_TRIES)
    {

        printf("\nUnexpected response from server. Trying again in 1
sec...\n");
        sleep(1);

        // sends the pass again
        if(send_command(socketfd, pass_command))
        {
            return -1;
        }

        if(receive_server_response(socketfd, &pass_input_response))
        {
            return -1;
        }

        num_tries++;

    }

    if(num_tries == MAX_TRIES)
    {
        printf("\nMaximum tries exceded. Exiting...\n");
        return -1;
    }

    return 0;
}
```

```
int enter_pasv_mode(int socketfd){

    ftp_server_response pasv_response;

    // send pasv command to enter passive mode
    if(send_command(socketfd, "pasv \n"))
    {
        return -1;
    }

    if(receive_server_response(socketfd, &pasv_response))
    {
        return -1;
    }

    // printf("pasv response code: %d\n", pasv_response.code);
    // printf("pasv response description: %s\n",
pasv_response.description);

    int num_tries = 0;

    // Tries to receive "Entering Passive Mode (h1,h2,h3,h4,p1,p2)."
response MAX_TRIES
    while(pasv_response.code != ENTER_PASV_MODE && num_tries <
MAX_TRIES)
    {
        printf("\nUnexpected response from server. Trying again in 1
sec...\n");
        sleep(1);

        if(send_command(socketfd, "pasv \n"))
        {
            return -1;
        }

        if(receive_server_response(socketfd, &pasv_response))
        {
            return -1;
        }

        num_tries++;
    }
```

```
    if(num_tries == MAX_TRIES)
    {
        printf("\nMaximum tries exceded. Exiting...\n");
        return -1;
    }


    strtok(pasv_response.description, "(");

    char *six_bytes = strtok(NULL, ")");

    IPv4_address server_ip;

    int p1, p2;
    char ip[30];


    // parse the string into the IP and ports

    sscanf(six_bytes,
            "%d,%d,%d,%d,%d,%d",
            &server_ip.h1,
            &server_ip.h2,
            &server_ip.h3,
            &server_ip.h4,
            &p1,
            &p2);

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

    // calculate the port on the server that is waiting for a
connection
    int port = p1 * 256 + p2;


    int pasv_socketfd;
    if((pasv_socketfd = connection_TCP(ip, port)) < 0){
```

```
        printf("Error in establishing TCP connection with server to
transfer data in passive mode\n");
        return -1;
    }

    printf("TCP connection established with server to transfer data in
passive mode\n");

    return pasv_socketfd;
}

int retr_file(int socketfd, int pasv_socketfd, char* path){

    char retr_command[PATH_LEN + 10];
    ftp_server_response retr_response;

    // retr command to send to the server
    sprintf(retr_command, "retr %s\n", path);

    // Sending retrieve command
    if(send_command(socketfd, retr_command))
    {
        return -1;
    }

    if(receive_server_response(socketfd, &retr_response))
    {
        return -1;
    }


    int num_tries = 0;

    // Tries to receive "File status okay; about to open data
connection." response MAX_TRIES
    while(retr_response.code != FILE_STATUS_OK && num_tries <
MAX_TRIES){
        printf("\nUnexpected response from server. Trying again in 1
sec...\n");
        sleep(1);

        if(send_command(socketfd, retr_command))
        {
```

```
            return -1;
        }

        if(receive_server_response(socketfd, &retr_response))
        {
            return -1;
        }

        num_tries++;
    }

    if(num_tries == MAX_TRIES)
    {
        printf("\nMaximum tries exceded. Exiting...\n");
        return -1;
    }

    char filename[FILENAME_LEN];
    if(parse_filename(path, filename))
    {
        return -1;
    }

    /*
        flag -> 0644
        (owning) User: read & write
        Group: read
        Other: read

        O_WRONLY ->  write-only
        O_CREAT -> If pathname does not exist, create it as a regular
file.

    */
    // open file descriptor
    int filefd = open(filename, O_WRONLY | O_CREAT, 0644);
    if(filefd == -1){
        perror("open file descriptor");
        return -1;
    }

    char buf[1];
    int rd, wt;
```

```
    do{
        // read from the server socker
        rd = read(pasv_socketfd, buf, 1);
        if(rd == -1){
            perror("read");
            return -1;
        }

        // write on file
        wt = write(filefd, buf, 1);
        if(wt == -1){
            perror("write");
            return -1;
        }
    }while(rd != 0);

    close(filefd);

    ftp_server_response transfer_ok_response;

    if(receive_server_response(socketfd, &transfer_ok_response) != 0 ||
transfer_ok_response.code != TRANSFER_COMPLETE)
    {
        return -1;
    }

    return 0;
}
```

clientTCP.c

```c
#include "../lib/clientTCP.h"

int connection_TCP(char* server_ip, int server_port) {

    int sockfd;
    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(server_ip);     /*32 bit
Internet address network byte ordered*/
    server_addr.sin_port = htons(server_port);              /*server TCP
port must be network byte ordered*/

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

    /*connect to the server*/
    if (connect(sockfd,
                (struct sockaddr *) &server_addr,
                sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
    }

    return sockfd;
}
```

getid.c

```
#include "../lib/getip.h"

int get_host_ip(char* host, char* ip) {
    struct hostent *h;

    /**
    * The struct hostent (host entry) with its terms documented

        struct hostent {
            char *h_name;    // Official name of the host.
            char **h_aliases;    // A NULL-terminated array of
alternate names for the host.
            int h_addrtype;    // The type of address being returned;
usually AF_INET.
            int h_length;    // The length of the address in bytes.
            char **h_addr_list;    // A zero-terminated array of
network addresses for the host.
            // Host addresses are in Network Byte Order.
        };

        #define h_addr h_addr_list[0]   The first address in
h_addr_list.
    */
    if ((h = gethostbyname(host)) == NULL) {
        herror("gethostbyname()");
        exit(-1);
    }

    printf("Host name  : %s\n", h->h_name);
    printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *)
h->h_addr)));

    strcpy(ip, inet_ntoa(*((struct in_addr *)h->h_addr)));

    return 0;
}
```

macros.h

```
#ifndef MACROS_H
#define MACROS_H
#define FTP_PORT 21
#define FILENAME_LEN 63
#define USER_LEN 63
#define PASS_LEN 63
#define HOST_LEN 255
#define PATH_LEN 255
#define DESC_LEN 127
#define SERVER_RESPONSE_LEN 255
#define READY_FOR_NEW_USER 220
#define USER_LOGIN_OK 230
#define NEED_PASSWORD 331
#define ENTER_PASV_MODE 227
#define FILE_STATUS_OK 150
#define TRANSFER_COMPLETE 226
#define MAX_TRIES 3

typedef struct
{
    char user[USER_LEN];
    char pass[PASS_LEN];
    char host[HOST_LEN];
    char path[PATH_LEN];
} ftp_url;

typedef struct
{
    int code;
    char description[DESC_LEN];
} ftp_server_response;

typedef struct
{
    int h1;
    int h2;
    int h3;
    int h4;
} IPv4_address;

#endif
```

app.h

```
#ifndef APP_H

#define APP_H

#include "../lib/parser.h"

#include "../lib/ftp.h"

#include "../lib/getip.h"

#include "../lib/clientTCP.h"

#endif
```

parser.h

```
#ifndef PARSER_H
#define PARSER_H

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

#include "macros.h"

int parse_url(const char* url, ftp_url* args);
int parse_url_first_part(char* user_pass_host, ftp_url* args);
int parse_server_response(const char* response_string,
ftp_server_response* response, const char* delimiter);
int parse_filename(char* path, char* filename);

#endif
```

ftp.h

```
#ifndef FTP_H
#define FTP_H

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

#include "macros.h"

int file_transfer(ftp_url args);
int receive_server_response(int sockfd, ftp_server_response* response);
int receive_server_response_string(int sockfd, char* response);
int send_command(int sockfd, char* command);
int authenticate_user(int socketfd ,char* user, char* pass);
int enter_pasv_mode(int socketfd);
int retr_file(int socketfd, int pasv_socketfd, char* path);

#endif
```

clientTCP.h

```
#ifndef CLIENTTCP_H
#define CLIENTTCP_H

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

/**
 * @brief Establish a TCP connection (using SOCKETs) to a server
 *
 * @param server_ip IP address of the server
```

```
 * @param server_port Port of the server, default is FTP_PORT->21
 * @return int negative if an error occured. 0 otherwise.
 */
int connection_TCP(char* server_ip, int server_port);

#endif
```

getip.h

```
#ifndef GETIP_H
#define GETIP_H

#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/**
 * @brief get the IP address from hostname
 *
 * @param host hostname of the server
 * @param ip IP address of the server
 * @return int negative if an error occured. 0 otherwise.
 */
int get_host_ip(char* host, char* ip);

#endif
```

## Comandos de configuração

### TUX2

```
#!/bin/sh

ifconfig eth0 down

# Configuring tux2
ifconfig eth0 up
ifconfig eth0 172.16.61.1/24

# Configure the routes in tuxy2 so that they can reach tuxy3
route add -net 172.16.60.0/24 gw 172.16.61.253

# Set in tuxy2 the default gateway to the Cisco Router
route add default gw 172.16.61.254


# To verify if routes are okay
route -n
```

### TUX3

```
#!/bin/sh

ifconfig eth0 down

# Configuring tux3
ifconfig eth0 up
ifconfig eth0 172.16.60.1/24

# Configure the routes in tuxy3 so that they can reach tuxy2
route add -net 172.16.61.0/24 gw 172.16.60.254

# Set tux3 a default gateway to tux4
route add default gw 172.16.60.254

# To verify if routes are okay
route -n
```

## TUX4

```
#!/bin/sh

ifconfig eth0 down
ifconfig eth1 down

# Configuring tux4
ifconfig eth0 up
ifconfig eth0 172.16.60.254/24

ifconfig eth1 up
ifconfig eth1 172.16.61.253/24

# Enabling forwarding in tuxes
echo 1 > /proc/sys/net/ipv4/ip_forward

# Enabling echo reply to broadcast request
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

# Set in tuxy4 the default gateway to the Cisco Router
route add default gw 172.16.61.254

# To verify if routes are okay
route -n
```

## VLAN

show vlan brief

#Creating an Ethernet VLAN

configure terminal
vlan 60
end

#Add port X to vlan 60  (tux3 and tux4)
configure terminal

interface fastethernet 0/X
switchport mode access
switchport access vlan 60
end

#Creating an Ethernet VLAN

configure terminal
vlan 61
end

#Add port X to vlan 61 (tux2, tux4 and router)
configure terminal

interface fastethernet 0/X
switchport mode access
switchport access vlan 61
end

show vlan brief

ROUTER

```
conf t

interface FastEthernet0/0
ip address 172.16.2.69 255.255.255.0
no shutdown
ip nat outside
exit

interface FastEthernet0/1
ip address 172.16.61.254 255.255.255.0
no shutdown
ip nat inside
exit

ip nat pool ovrld 172.16.2.69 172.16.2.69 prefix 24
ip nat inside source list 1 pool ovrld overload

access-list 1 permit 172.16.60.0 0.0.0.7
access-list 1 permit 172.16.61.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.2.254
ip route 172.16.60.0 255.255.255.0 172.16.61.253

end
```

# Logs captured

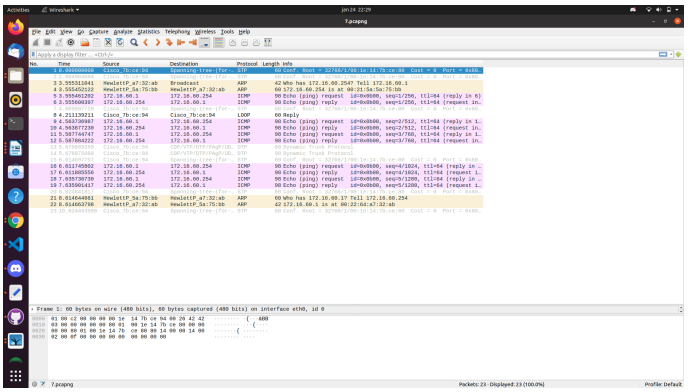

Fig - logs capturados experiência 1

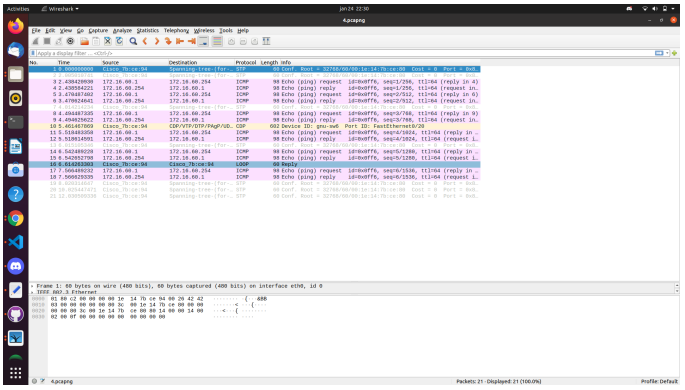

Fig - logs capturados experiência 2 ponto 4

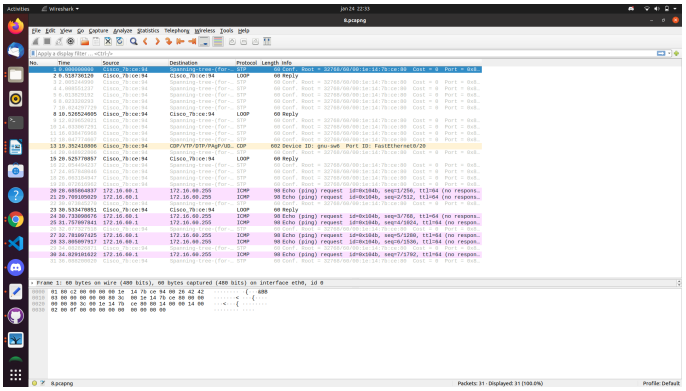

Fig - logs capturados experiência 2 ponto 8

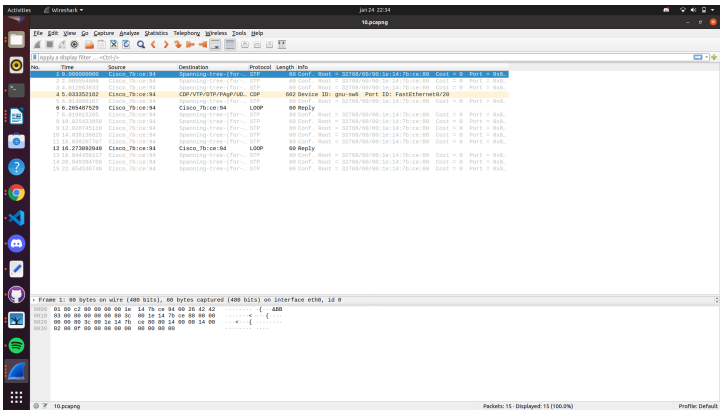

Fig - logs capturados experiência 2 ponto 10

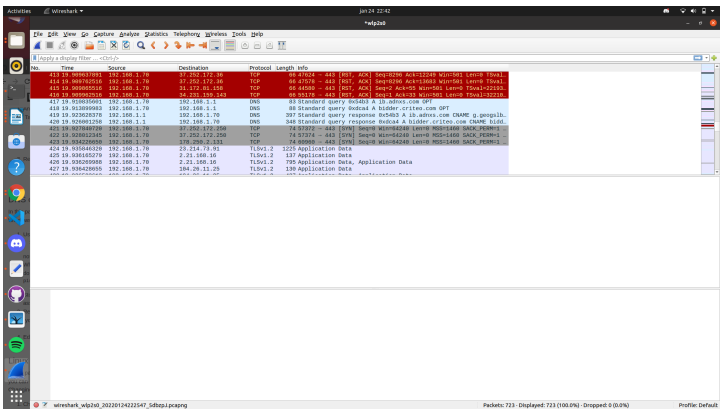

Fig -logs capturados na experiência 3 dns config ponto 2

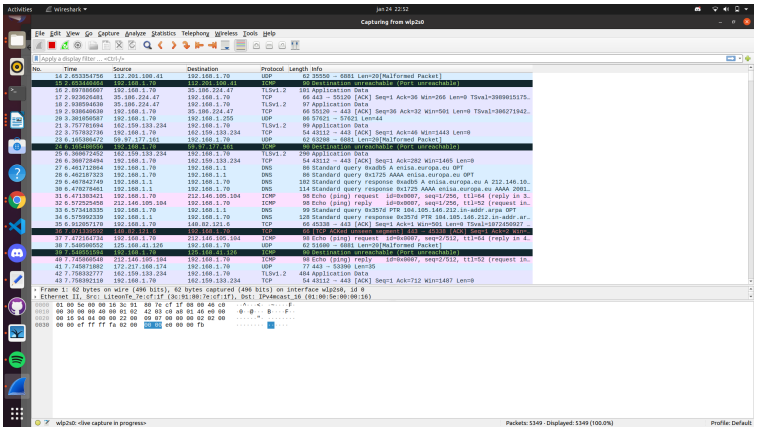

Fig - logs capturados experiência 3 dns config ponto 3

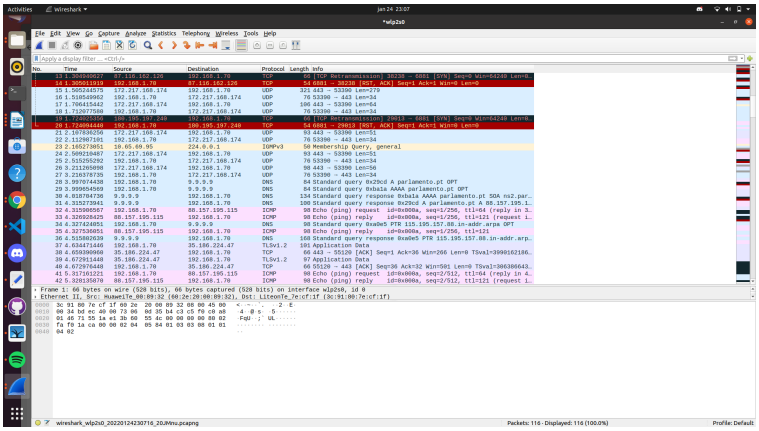

Fig - logs capturados experiência 3 dns config ponto 4

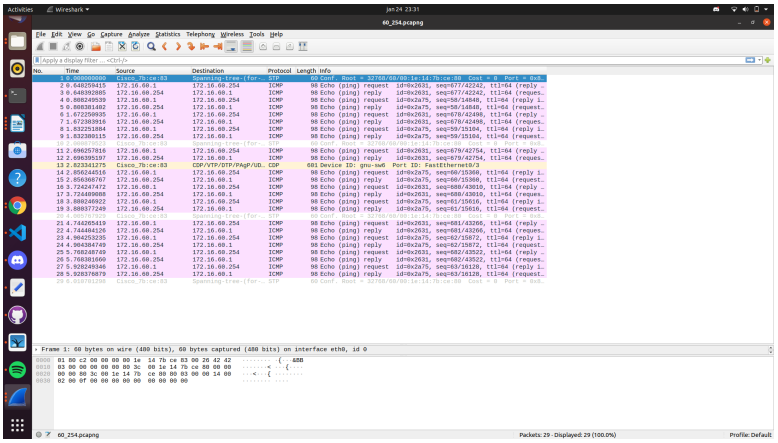

Fig - logs capturados na experiência 3 ponto 9 do tux 3 para 172.16.60.254

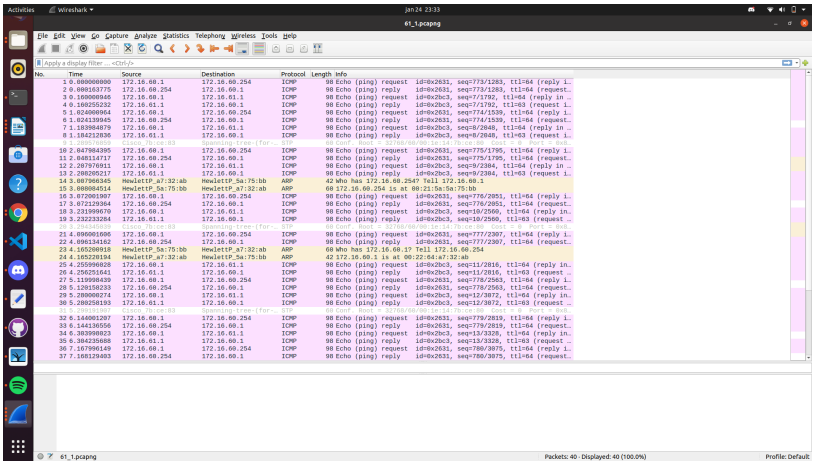

Fig - logs capturados na experiência 3 ponto 9 do tuxy3 para 172.16.61.1

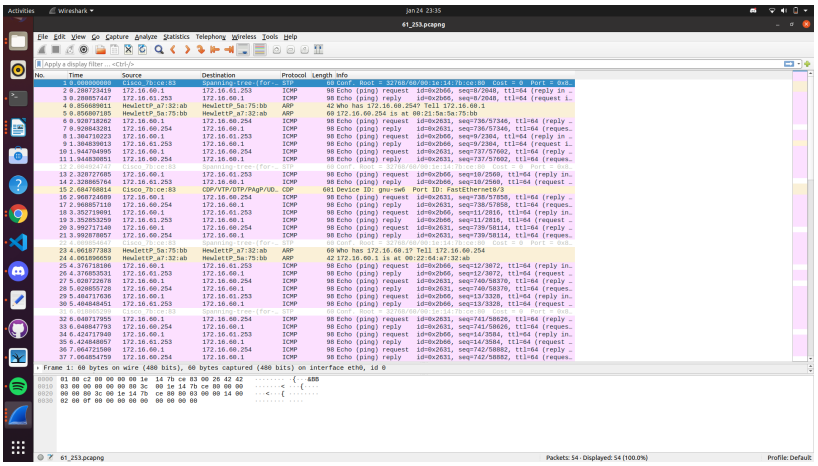

Fig - logs capturados na experiência 3 ponto 9 do tuxy3 para 172.16.61.253

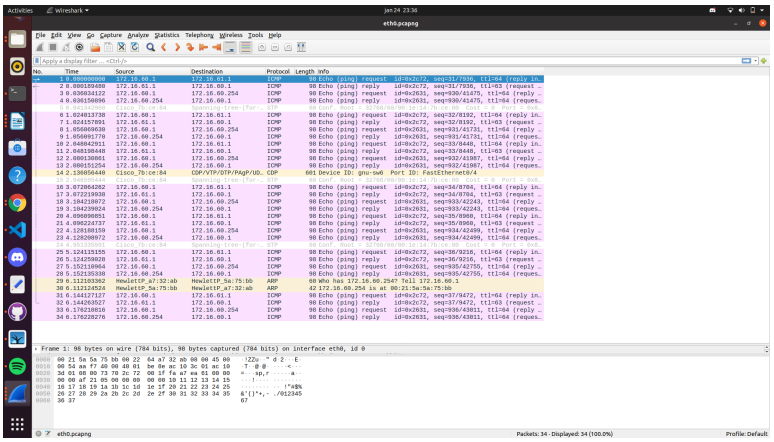

Fig - logs capturados na experiência 3 ponto 11

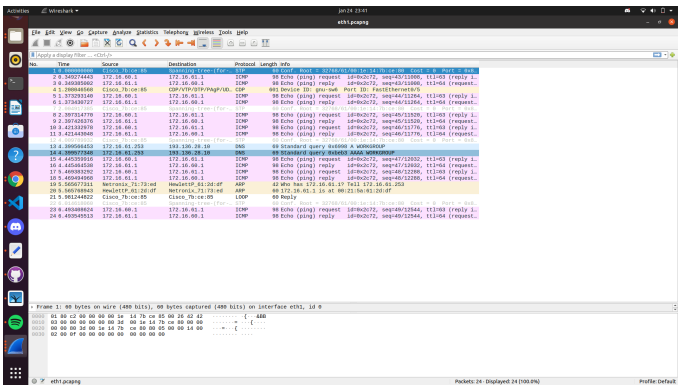

Fig - logs capturados na experiência 3 ponto 11