2º Projeto - Rede de Computadores

Licenciatura em Engenharia Informática e Computação
Redes de Computadores

Turma 9, Grupo 8:

Maria Leonor Monteiro Beirão | 201806798
José Pedro Nogueira Rodrigues | 201708806

28 de Janeiro de 2022

Sumário

O objetivo deste segundo projeto da unidade curricular de Redes de Computadores é desenvolver uma rede de computadores, fazendo um estudo e configuração dos diversos elementos que a compõem, entre os computadores em si, o *Router Cisco* e o *Switch*.

É desenvolvido também um simples programa de download de ficheiros seguindo o protocolo de FTP (File Transfer Protocol).

1. Introdução

Este projeto divide-se em duas partes fundamentais: o desenvolvimento de um programa de download por FTP e a configuração e estudo de uma rede de computadores.

Em primeiro lugar foi desenvolvida a aplicação de download, seguindo o protocolo FTP, e utilizando ligações TCP entre sockets para proceder à comunicação do pedido para o servidor FTP e o processamento das suas respostas.

De seguida foi feita a configuração de uma rede que permitisse a execução desse programa, a partir de duas VLAN's ligadas por um switch.

O objetivo deste relatório é demonstrar e explicar a arquitetura e o processo de desenvolvimento dessas duas componentes, tendo a seguinte estrutura:

2. Parte 1 - Aplicação de download

2.1. Arquitetura da aplicação de download

O primeiro passo que o nosso programa executa é o processamento do URL fornecido como argumento. Este URL vem no formato *ftp://[<user>:<password>@]<host>/<url-path>*, e dele, utilizando a função **processArgument(char *argv[])**, vamos obter e guardar em variáveis o *user*, a *password*, o *host*, o *url_path* e o *file_name*. No caso de não ser fornecido *user* e *password*, uma variável denominada *isAnonymous* é colocada a 1 (TRUE), caso contrário ela mantém se a 0 (FALSE).

Estas variáveis, tal como futuras *ip_addr*, *isAnonymous*, *transferPort* e *respsockfd*, são todas guardadas numa struct denominada **FTPTask**, onde é reunida toda a informação necessária a cada execução de um download.

Após o processamento do URL de argumento, obtém se o endereço IP do host utilizando código previamente fornecido no ficheiro **getip.c**, sendo guardada esta informação na variável *ip_addr*. O port utilizado é sempre o port 21.

Terminado este processamento inicial, vai se iniciar a comunicação com o servidor FTP. Esta comunicação é iniciada utilizando código previamente fornecido no ficheiro **clientTCP.c**, sendo apenas necessário fornece o *ip_addr* e o port (21).

Ao estabelecer conexão, o servidor irá nos responder, pelo que precisamos de uma função que lide com as respostas do servidor e aja em acordância. Essa função é a **processResp(int sockfd, FTPTask *task, int stage)**. Esta função recebe o *sockfd* (file descriptor da ligação ao servidor FTP), a *task* (struct FTPTask preenchida com a informação sobre o download a executar) e a *stage* (variável que representa a fase em que o programa se encontra, o que indica o tipo de resposta esperada). Existem 5 *stages*, são elas *ESTABLISH_CONNECTION*, *SEND_USERNAME*, *SEND_PASSWORD*, *GET_TRANSFER_PORT* e *GET_FILE*.

Após ser estabelecida a conexão (*ESTABLISH_CONNECTION*), são enviados os comandos **USER** *user* (*SEND_USERNAME*) e **PASS** *password* (*SEND_PASSWORD*) para efetuar o login no servidor. Em caso de sucesso, pede se ao servidor a ativação do modo passivo, atráves do comando **PASV**, que vai responder com um port por onde iremos abrir uma segundo ligação para a transferência do ficheiro (*GET_TRANSFER_PORT*).

Por último, após ser aberta essa nova ligação com o port fornecido, irá se executar o pedido do ficheiro, enviando o comando **RETR** *url_path* (*GET_FILE*). Nesta última fase é utilizada a função **receiveFile(FTPTask *task)** para processar e criar o ficheiro localmente, dando por terminado o processo de download.

2.2. Resultados obtidos

O nosso programa foi testado fazendo download de vários ficheiros diferentes, tanto em modo anónimo como em modo autenticado, tendo sempre realizado o download do ficheiro com sucesso. Em caso de falha em alguma das fases do processo (conexão falhada, ficheiro não existe, etc) o programa termina, notificando o utilizador do erro ocorrido.

3. Parte 2 - Configuração e análise de rede

3.1 Experiência 1

O que são os pacotes ARP e para o que são usados?

Os pacotes Adress Resolution Protocol (ARP) são pacotes de comunicação que permitem pedir o endereço MAC de uma máquina, sabendo o endereço IP.

Quais são os endereços MAC e IP de pacotes ARP e porquê?

Os endereços Medium Access Control (MAC) são os identificadores das placas de rede. Os endereços Internet Protocol (IP) são os identificadores públicos de cada máquina que lhe permitem comunicar com outras máquinas. O endereço MAC é único enquanto que uma máquina pode ter vários endereços IP.

Que pacotes são gerados pelo comando *ping*?

O comando ping gera primeiro pacotes ARP para obter endereços MAC associados ao IP fornecido, e depois gera pacotes Internet Control Access Protocol (ICMP), que têm a função de enviar mensagens de controlo, de host para host, ou erros.

Quais os endereços MAC e IP dos pacotes ping?

Pacote de pedido

Endereço MAC da origem do pacote: 00:22:64:a6:a4:f1

Endereço MAC do destino do packet: 00:21:5a:7d:12

Endereço IP da origem do packet: 172.16.20.1

Endereço IP do destino do packet: 172.16.20.254

Pacote de resposta

Endereço MAC da origem do pacote: 00:21:5a:7d:12

Endereço MAC do destino do packet: 00:22:64:a6:a4:f1

Endereço IP da origem do packet: 172.16.20.254

Endereço IP do destino do packet: 172.16.20.1

Como determinar se uma receiving Ethernet frame é ARP, IP, ICMP?

Verificando os 2 bytes Type da trama do tipo Ethernet, e verificando os detalhes dos mesmos, nomeadamente o campo “Protocol”.

Como determinar o comprimento de uma frame recebida?

Usando o WireShark, verificamos um número em bytes em “on wire”, ou no campo “Length”.

O que é a loopback interface e porque é importante?

Loopback interface consiste em enviar através de uma máquina um pacote cujo endereço de destino é o da própria máquina. Isto tem o objetivo de verificar se a rede está bem organizada, dependendo de receber o pacote ou não.

3.2 Experiência 2

Como configurar vlan<u>Y</u>0?

Começámos por criar VLAN 20 e 21, estando na bancada 2. Associamos os ports 1 e 3 à VLAN 20, e os ports 2, 4 e 5 à VLAN 21 no tux23.

Quantos broadcast domains existem? Como se pode concluir isto a partir dos logs?

Existem 2 broadcast domains. Foi feito ping do tux23 a tux24, que ocorreu com sucesso. Ping a partir do tux23 para o tux22 não obteve resposta. No tux23 foi feito ping broadcast, que recebeu resposta do tux24, mas não do tux22. Fazendo novamente ping broadcast desta vez no tux22, não recebemos qualquer resposta. Podemos concluir que existem 2 broadcast domains, um que contém o tux23 e tux24, e outro que contém o tux22.

3.3 Experiência 3

(Cisco) Como configurar uma rota estática num router comercial?

Para se configurar uma rota, após se entrar no modo global de configuração (- configure terminal), é preciso apenas correr os seguintes comandos:

- ip route *prefix mask* { *ip-address* | *interface-type interface-number* [*ip-address*]} (exemplo: ip route *192.168.1.0 255.255.0.0 10.10.10.2*)

- end

(Cisco) Como configurar NAT num router comercial?

Para se configurar Network Address Translation (NAT), após se entrar no modo global de configuração, é preciso correr os seguintes comandos:

- interface *type number*

- ip address *ip-address subnet-mask*

- no shutdown

- ip nat inside/outside

Exemplo:

- interface gigabitethernet 0/0 *

- ip address 172.16.21.254 255.255.255.0

```
- no shutdown

- ip nat inside
```

(Cisco) O que é que NAT faz?

NAT é, na sua base, uma tradução de endereços de rede. É uma forma de mapear vários endereços privados locais num endereço público antes de transferir informações. Organizações que desejem que vários dispositivos empreguem um único endereço IP usam NAT, assim como a maioria dos routers domésticos.

(DNS) Como configurar o serviço DNS num host?

Para configurar e alterar o serviço DNS num host, tudo o que é preciso é alterar o ficheiro hosts, que se encontra em /etc/hosts. Este ficheiro é um ficheiro simples de texto que associa endereços IP a nomes de hosts, uma linha por endereço IP. Para cada host, uma única linha deve estar presente com as seguintes informações:

- *ip-address canonical_hostname* [*aliases...*]

(DNS) Que pacotes são transferidos por DNS e que informação transportam?

Os packets que são transferidos por DNS são packets ICMP (Internet Control Access Protocol), e a informação que transportam, para além da parte do cabeçalho (que inclui o tipo (8 - Echo Request/Reply), o checksum, entre outros dados de identificação), transporta uma pergunta para o name server ou uma resposta a uma pergunta.

(Linux) Que pacotes ICMP podem ser observados e porquê?

Os pacotes ICMP que são observados são pacotes do tipo 11 e sub-tipo 0. Ou seja, são pacotes que indicam *Time to live exceeded in transit*. Isto acontece porque, ao termos apagado a default gateway entry da forwarding table, o caminho designado para o pacote seguir não existe, sendo este “perdido em trânsito”.

(Linux) Quais são os endereços IP e MAC associados com os pacotes ICMP e porquê?

Os endereços IP e MAC associados aos pacotes ICMP são o IP e MAC address fonte (10.0.2.1 e 52:54:00:12:35:00) e o endereço IP de destino (10.0.2.5). E a razão de serem estes é porque o endereço fonte (10.0.2.1) é o nosso *Default Gateway IP* ou o IP do nosso router, enquanto que o endereço de destino (10.0.2.5) somos nós próprios, é o nosso IP privado dentro da nossa rede local.

(Linux) Que rotas estão presentes na tua máquina? Qual é o seu significado?

As rotas presentas na nossa máquina eram as seguintes:

```
- default via 10.0.2.1 dev enp0s9 proto dhcp metric 100
```

```
- default via 10.0.2.2 dev enp0s3 proto dhcp metric 101

- 10.0.2.0/24 dev enp0s9 proto kernel scope link src 10.0.2.5 metric 100

- 10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 101

- 169.254.0.0/16 dev enp0s9 scope link metric 1000

- 192.168.56.0/24 dev enp0s8 proto kernel scope link src 192.168.56.101
metric 102
```

Estas rotas estão presentes na nossa máquina para indicar os caminhos que os pacotes que saem e chegam à nossa máquina devem seguir, e podemos encontrar tanto rotas mais específicas como rotas default, que são seguidas quando nenhuma outra se aplica.

4. Conclusões

A maior parte dos objetivos deste projeto foram alcançados. Devido a dificuldades provocadas pelo vírus da Covid-19 e de algumas dificuldades em participar nas aulas necessárias à conclusão do projeto (tanto pelos vários feriados que coincidiram com as aulas TP como por isolamentos obrigatórios devido a contactos de alto risco ou contração do vírus), não conseguimos terminar todas as experiências propostas neste projeto por ausência de tempo em laboratório.

Mesmo assim, foi possível obter conhecimento sobre o protocolo FTP e programação de comunicação sobre sockets tal como configuração de uma rede de computadores composta por computadores, switches e routers, endereços IP, endereços MAC, routes, ARP e NAT, entre outras especificidades não mencionadas.

De uma forma geral este projeto foi muito enriquecedor na nossa formação como engenheiros informáticos e no nosso conhecimento aprofundado da matéria em questão.

5. Referências

5.1. Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, DOI 10.17487/RFC0959, October 1985, <https://www.rfc-editor.org/info/rfc959>.

5.2. Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, DOI 10.17487/RFC1738, December 1994, <https://www.rfc-editor.org/info/rfc1738>.

5.3. Cisco Systems Inc. "Configuring Network Address Translation: Getting Started", <https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/13772-12.html>

5.4. Cisco Systems Inc. "Cisco 3900 Series, 2900 Series, and 1900 Series Software Configuration Guide", <https://www.cisco.com/c/en/us/td/docs/routers/access/1900/software/configuration/guide/Software_Configuration/routconf.html>

6. Anexos

6.1. Código da aplicação de download

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

#define FALSE 0
#define TRUE 1
#define ServerPort 21
#define MAX_FILE_SIZE 1000000

int debugMode = FALSE;

enum Responses {
    SERVICE_READY,
    SPECIFY_PASSWORD,
    LOGIN_DONE,
    LOGIN_FAILED,
    PASSIVE_MODE,
    SENDING_FILE,
    FILE_UNAVALIABLE,
};

static const char * const ResponseCodes[] = {
    [SERVICE_READY] = "220",
    [SPECIFY_PASSWORD] = "331",
    [LOGIN_DONE] = "230",
    [LOGIN_FAILED] = "530",
    [PASSIVE_MODE] = "227",
    [SENDING_FILE] = "150",
    [FILE_UNAVALIABLE] = "550",
};

enum AccessCommands {
    USER_NAME,
    PASSWORD,
};

static const char * const AccessCommands[] = {
    [USER_NAME] = "USER",
    [PASSWORD] = "PASS",
};

enum ParameterCommands {
    PASSIVE,
};
```

```
static const char * const ParameterCommands[] = {
    [PASSIVE] = "PASV",
};

enum ServiceCommands {
    RETRIEVE,
};

static const char * const ServiceCommands[] = {
    [RETRIEVE] = "RETR",
};

enum Stages {
    ESTABLISH_CONNECTION = 1,
    SEND_USERNAME = 2,
    SEND_PASSWORD = 3,
    GET_TRANSFER_PORT = 4,
    GET_FILE = 5
};

typedef struct {
    unsigned char *user;
    unsigned char *password;
    unsigned char *url_path;
    unsigned char *file_name;
    unsigned char *host;
    unsigned char *ip_addr;
    int isAnonymous;
    int transferPort;
    int respsockfd;
} FTPTask;

void processResp(int sockfd, FTPTask *task, int stage);
void clearReading(int sockfd);
FTPTask processArgument(char *argv[]);
void receiveFile(FTPTask *task);

int main(int argc, char *argv[]) {
    struct hostent *h;
    FTPTask task;

    int sockfd, respsockfd;
    struct sockaddr_in server_addr;
    struct sockaddr_in resp_addr;
    char respCode[3];

    char buf[] = "Mensagem de teste na travessia da pilha TCP/IP\n";
    size_t bytes;

    printf("\n");
    if (argc != 2) {
        fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
argv[0]);
        exit(-1);
    }
```

```
    task = processArgument(argv);

    if ((h = gethostbyname(task.host)) == NULL) {
        herror("gethostbyname()");
        exit(-1);
    }

    task.ip_addr = inet_ntoa(*((struct in_addr *) h->h_addr_list[0]));

    if(debugMode) printf("IP Address: %s\n\n", task.ip_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(task.ip_addr);    /*32 bit Internet
address network byte ordered*/
    server_addr.sin_port = htons(ServerPort);        /*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);
    }

    printf("> Establishing connection\n");
    processResp(sockfd, &task, ESTABLISH_CONNECTION);

    // Login process
    if(!task.isAnonymous) {
        printf("> Sending Username\n");
        write(sockfd, AccessCommands[USER_NAME],
strlen(AccessCommands[USER_NAME]));
        write(sockfd, " ", 1);
        write(sockfd, task.user, strlen(task.user));
        write(sockfd, "\n", 1);
        processResp(sockfd, &task, SEND_USERNAME);

        printf("> Sending Password\n");
        write(sockfd, AccessCommands[PASSWORD], strlen(AccessCommands[PASSWORD]));
        write(sockfd, " ", 1);
        write(sockfd, task.password, strlen(task.password));
        write(sockfd, "\n", 1);
        processResp(sockfd, &task, SEND_PASSWORD);
    } else {
        printf("> Logging in anonymously\n");
        write(sockfd, AccessCommands[USER_NAME],
strlen(AccessCommands[USER_NAME]));
```

```
        write(sockfd, " ", 1);
        write(sockfd, "anonymous", strlen("anonymous"));
        write(sockfd, "\n", 1);
        processResp(sockfd, &task, SEND_USERNAME);

        write(sockfd, AccessCommands[PASSWORD], strlen(AccessCommands[PASSWORD]));
        write(sockfd, " ", 1);
        write(sockfd, "a", 1);
        write(sockfd, "\n", 1);
        processResp(sockfd, &task, SEND_PASSWORD);
    }

    // Request and process data transfer port
    printf("> Requesting port for file transfer\n");
    write(sockfd, ParameterCommands[PASSIVE], strlen(ParameterCommands[PASSIVE]));
    write(sockfd, "\n", 1);
    processResp(sockfd, &task, GET_TRANSFER_PORT);

    printf("< Transfer port: %i\n", task.transferPort);

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

    /*open a TCP socket*/
    if ((respsockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
    }
    /*connect to the server*/
    if (connect(respsockfd, (struct sockaddr *) &resp_addr, sizeof(resp_addr)) < 0)
{
        perror("connect()");
        exit(-1);
    }

    task.respsockfd = respsockfd;

    // Request and process file
    printf("> Retrieving file %s\n", task.file_name);
    write(sockfd, ServiceCommands[RETRIEVE], strlen(ServiceCommands[RETRIEVE]));
    write(sockfd, " ", 1);
    write(sockfd, task.url_path, strlen(task.url_path));
    write(sockfd, "\n", 1);
    processResp(sockfd, &task, GET_FILE);

    if (close(sockfd)<0 || close(respsockfd)<0) {
        perror("close()");
        exit(-1);
    }
```

```
    return 0;
}

FTPTask processArgument(char *argv[]) {
    unsigned char *tmp, *end, *host, *url_path, *file_name, *user, *password =
NULL;
    int isAnonymous = TRUE;

    FTPTask task;

    tmp = argv[1];

    if (!strstr(tmp, "ftp://")) {
        fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
argv[0]);
        exit(-1);
    }

    tmp += + strlen("ftp://");

    if (end = strstr(tmp, ":")) {
        task.isAnonymous = FALSE;

        user = (char *) malloc(strlen(tmp) - strlen(end) + 1);
        memcpy(user, tmp, strlen(tmp) - strlen(end));
        user[strlen(tmp) - strlen(end)] = '\0';

        if(debugMode) if (user) printf("User: %s\n", user);

        tmp += + strlen(user) + 1;

        if (end = strstr(tmp, "@")) {
            password = (char *) malloc(strlen(tmp) - strlen(end));
            memcpy(password, tmp, strlen(tmp) - strlen(end));
            password[strlen(tmp) - strlen(end)] = '\0';

            if(debugMode) if (password) printf("Password: %s\n", password);

            tmp += + strlen(password) + 1;
        }
    } else task.isAnonymous = TRUE;

    if (end = strstr(tmp, "/")) {
        host = (char *) malloc(strlen(tmp) - strlen(end) + 1);
        memcpy(host, tmp, strlen(tmp) - strlen(end) + 1);
        host[strlen(tmp) - strlen(end)] = '\0';

        if(debugMode) if (host) printf("Host: %s\n", host);

        tmp += strlen(host);
    } else {
        fprintf(stderr, "There is information missing in the arguments given!\n");
        fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
argv[0]);
        exit(-1);
```

```
    }

    url_path = strstr(tmp, "/") + 1;

    if(debugMode) if (url_path) printf("Url: %s\n", url_path);

    if (strlen(url_path) <= 0) {
        fprintf(stderr, "There is information missing in the arguments given!\n");
        fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
argv[0]);
        exit(-1);
    }

    file_name = strstr(tmp, "/");

    do {
        file_name = strstr(file_name, "/") + 1;
    } while(strstr(file_name, "/") > 0);

    if(debugMode) if (file_name) printf("Filename: %s\n", file_name);

    task.host = host;
    task.user = user;
    task.password = password;
    task.url_path = url_path;
    task.file_name = file_name;

    return task;
}

void processResp(int sockfd, FTPTask *task, int stage) {
    char c;
    unsigned char respCode[3] = "", p1[4] = "", p2[4] = "";
    int stillReading = TRUE, hasToProcess = FALSE;
    int ignoreCommas = -4;

    if(stage != SEND_USERNAME) printf("< ");
    while(stillReading) {
        read(sockfd, &c, 1);
        if(debugMode) printf("%c", c);

        switch (stage) {
            case ESTABLISH_CONNECTION:
                if(strlen(respCode) < 3) {
                    respCode[strlen(respCode)] = c;
                } else {
                    if(debugMode) printf("Response code: %s\n", respCode);
                    if(!strcmp(respCode, ResponseCodes[SERVICE_READY]))
printf("Connection to server established sucessfully\n");
                    else {
                        printf("Connection to server failed!\n");
                        printf("< Error code: %s\n", respCode);
                        exit(-1);
                    }
                    stillReading = FALSE;
```

```
                }
                break;
            case SEND_USERNAME:
                if(strlen(respCode) < 3) {
                    respCode[strlen(respCode)] = c;
                } else {
                    if(debugMode) printf("Response code: %s\n", respCode);
                    if(strcmp(respCode, ResponseCodes[SPECIFY_PASSWORD])) {
                        printf("Failed to receive valid response to username!\n");
                        printf("< Error code: %s\n", respCode);
                        exit(-1);
                    }
                    stillReading = FALSE;
                }
                break;
            case SEND_PASSWORD:
                if(strlen(respCode) < 3) {
                    respCode[strlen(respCode)] = c;
                } else {
                    if(debugMode) printf("Response code: %s\n", respCode);
                    if(!strcmp(respCode, ResponseCodes[LOGIN_DONE])) printf("Login
process sucessful\n");
                    else {
                        if(!strcmp(respCode, ResponseCodes[LOGIN_FAILED])) {
                            printf("Login failed, credentials might be wrong!\n");
                        } else {
                            printf("Failed to receive valid response to
password!\n");
                            printf("< Error code: %s\n", respCode);
                        }
                        exit(-1);
                    }
                    stillReading = FALSE;
                }
                break;
            case GET_TRANSFER_PORT:
                if(!hasToProcess) {
                    if(strlen(respCode) < 3) {
                        respCode[strlen(respCode)] = c;
                    } else {
                        if(debugMode) printf("Response code: %s\n", respCode);
                        if(!strcmp(respCode, ResponseCodes[PASSIVE_MODE]))
printf("Entered passive mode\n");
                        else {
                            printf("Failed to receive valid response to passive
mode request!\n");
                            printf("< Error code: %s\n", respCode);
                            exit(-1);
                        }
                        hasToProcess = TRUE;
                    }
                } else {
                    if(c == ')') {
                        task->transferPort = atoi(p1) * 256 + atoi(p2);
                        stillReading = FALSE;
```

```
                    } else {
                        if(c == ',')
                            ignoreCommas++;
                        else if(ignoreCommas == 0)
                            p1[strlen(p1)] = c;
                        else if(ignoreCommas == 1)
                            p2[strlen(p2)] = c;
                    }
                }
                break;
            case GET_FILE:
                if(strlen(respCode) < 3) {
                    respCode[strlen(respCode)] = c;
                } else {
                    if(debugMode) printf("Response code: %s\n", respCode);
                    if(strcmp(respCode, ResponseCodes[SENDING_FILE])) {
                        if(!strcmp(respCode, ResponseCodes[FILE_UNAVALIABLE])) {
                            printf("File request failed, file is unavaliable or
does not exist!\n");
                        } else {
                            printf("Failed to receive valid response to file
request!\n");
                            printf("< Error code: %s\n", respCode);
                        }
                        exit(-1);
                        exit(-1);
                    } else {
                        receiveFile(task);
                        return;
                    }
                    stillReading = FALSE;
                }
                break;
            default:
                fprintf(stderr, "Error: Invalid stage given!\n");
                exit(-1);
                break;
        }
    }

    clearReading(sockfd);
}

void receiveFile(FTPTask *task) {
    FILE *file = fopen((char *) task->file_name, "wb+");

    char bufSocket[MAX_FILE_SIZE];
    int bytes;
    while ((bytes = read(task->respsockfd, bufSocket, MAX_FILE_SIZE))>0) {
        bytes = fwrite(bufSocket, bytes, 1, file);
    }

    fclose(file);

    printf("Finished downloading %s from %s!\n", task->file_name, task->host);
```

```
}

void clearReading(int sockfd) {
    char c;
    do {
        read(sockfd, &c, 1);
        if(debugMode) printf("%c", c);
    } while(c != '\n');
}
```

6.2. Comandos de configuração

6.2.1. Experiência nº1

Setup dos cabos:

Tux23E0 -> Switch

Tux24E0 -> Switch

Configuração dos endereços IP e das rotas:

tux23:

- ifconfig eth0 up
- ifconfig eth0 172.16.20.1/24
- ifconfig eth0
- route add -net 172.16.20.0/24 gw 172.16.20.254
- route add default gw 172.16.20.254

tux24:

- ifconfig eth0 up
- ifconfig eth0 172.16.20.254/24
- ifconfig eth0

Apagar entradas na tabela ARP do tux23:

- arp -d 172.16.20.254

6.2.2. Experiência nº2

Setup dos cabos:

Tux23E0 -> Switch Porta 1

Tux23S0 -> T3

```
T4 -> Consola Switch

Tux24E0 -> Switch Porta 2

Tux22E0 -> Switch Porta 3
```

Configuração dos endereços IP e das rotas:

tux23:

```
- ifconfig eth0 up

- ifconfig eth0 172.16.20.1/24

- ifconfig eth0

- route add -net 172.16.20.0/24 gw 172.16.20.254

- route add default gw 172.16.20.254
```

tux24:

```
- ifconfig eth0 up

- ifconfig eth0 172.16.20.254/24

- ifconfig eth0
```

tux22:

```
- ifconfig eth0 up

- ifconfig eth0 172.16.21.1/24

- ifconfig eth0
```

Criação da vlan20 e atribuição das portas:

tux23:

```
- enable

- configure terminal

- vlan 20

-  interface fastethernet 0/1

-  switchport mode access

-  switchport access vlan 20

- end
```

```
- configure terminal
-  interface fastethernet 0/2
-  switchport mode access
-  switchport access vlan 20
- end
```

Criação da vlan21 e atribuição da porta:

tux23:

```
- enable
- configure terminal
- vlan 21
- interface fastethernet 0/3
- switchport mode access
- switchport access vlan 21
```

6.3. Logs obtidos nas experiências

6.3.1. Experiência nº1

Pacote ARP

No.	Time	Source	Destination	Protocol	Length	Info
75	37.653252817	HewlettP_a6:a4:f1	HewlettP_5a:7d:12	ARP	60	Who has 172.16.20.1? Tell 172.16.20.0

Pacote ICMP

No.	Time	Source	Destination	Protocol	Length	Info
77	37.770370292	172.16.20.1	172.16.20.254	ICMP	98	Echo (ping) request id=0x39ba, seq=6/1536, ttl=64 (reply in 78)

Endereços IP e MAC dos pacotes ping

```
No.     Time           Source                Destination           Protocol Length Info
     56 32.648588807   172.16.20.1           172.16.20.254         ICMP     98     Echo (ping) request  id=0x39ba, seq=1/256, ttl=64 (reply in 57)

Frame 56: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Ethernet II, Src: HewlettP_5a:7d:12 (00:21:5a:5a:7d:12), Dst: HewlettP_a6:a4:f1 (00:22:64:a6:a4:f1)
Internet Protocol Version 4, Src: 172.16.20.1, Dst: 172.16.20.254
Internet Control Message Protocol

No.     Time           Source                Destination           Protocol Length Info
     57 32.648724582   172.16.20.254         172.16.20.1           ICMP     98     Echo (ping) reply    id=0x39ba, seq=1/256, ttl=64 (request in 56)

Frame 57: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Ethernet II, Src: HewlettP_a6:a4:f1 (00:22:64:a6:a4:f1), Dst: HewlettP_5a:7d:12 (00:21:5a:5a:7d:12)
Internet Protocol Version 4, Src: 172.16.20.254, Dst: 172.16.20.1
Internet Control Message Protocol
```

6.3.2. Experiência nº2

No tux23, fazer ping a tux24 e depois a tux22

```
 5 3.445200117  172.16.20.1     172.16.20.254     ICMP   98 Echo (ping) request  id=0x39dc, seq=1/256, ttl=64 (reply in 6)
 6 3.445356564  172.16.20.254   172.16.20.1       ICMP   98 Echo (ping) reply    id=0x39dc, seq=1/256, ttl=64 (request in 5)
 7 4.013856734  Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
 8 4.462913257  172.16.20.1     172.16.20.254     ICMP   98 Echo (ping) request  id=0x39dc, seq=2/512, ttl=64 (reply in 9)
 9 4.463072638  172.16.20.254   172.16.20.1       ICMP   98 Echo (ping) reply    id=0x39dc, seq=2/512, ttl=64 (request in 8)
10 5.486912170  172.16.20.1     172.16.20.254     ICMP   98 Echo (ping) request  id=0x39dc, seq=3/768, ttl=64 (reply in 11)
11 5.487046128  172.16.20.254   172.16.20.1       ICMP   98 Echo (ping) reply    id=0x39dc, seq=3/768, ttl=64 (request in 10)
12 6.014675262  Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
13 6.510923515  172.16.20.1     172.16.20.254     ICMP   98 Echo (ping) request  id=0x39dc, seq=4/1024, ttl=64 (reply in 14)
14 6.511057054  172.16.20.254   172.16.20.1       ICMP   98 Echo (ping) reply    id=0x39dc, seq=4/1024, ttl=64 (request in 13)

28 10.765518999 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=1/256, ttl=64 (no response found!)
29 11.771399042 172.16.20.1     172.16.1.1        DNS    97 Standard query 0xc841 A 3.debian.pool.ntp.org.netlab.fe.up.pt
30 11.771408261 172.16.20.1     172.16.1.1        DNS    97 Standard query 0xdf50 AAAA 3.debian.pool.ntp.org.netlab.fe.up.pt
31 11.790896607 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=2/512, ttl=64 (no response found!)
32 12.038054809 Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
33 12.814908231 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=3/768, ttl=64 (no response found!)
34 13.838909030 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=4/1024, ttl=64 (no response found!)
35 14.069634084 Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
36 14.774533260 172.16.20.1     193.136.28.10     DNS    97 Standard query 0xc841 A 3.debian.pool.ntp.org.netlab.fe.up.pt
37 14.774542968 172.16.20.1     193.136.28.10     DNS    97 Standard query 0xdf50 AAAA 3.debian.pool.ntp.org.netlab.fe.up.pt
38 14.862895860 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=5/1280, ttl=64 (no response found!)
39 15.886906646 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=6/1536, ttl=64 (no response found!)
40 16.072404646 Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
41 16.910908352 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=7/1792, ttl=64 (no response found!)
42 17.934913411 172.16.20.1     172.16.21.1       ICMP   98 Echo (ping) request  id=0x39e0, seq=8/2048, ttl=64 (no response found!)
```

Excerto de logs de ping em broadcast no tux23

```
11 15.590500197 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=1/256, ttl=64 (no response found!)
12 16.038237951 Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
13 16.612502796 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=2/512, ttl=64 (no response found!)
14 17.636507087 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=3/768, ttl=64 (no response found!)
15 18.043388123 Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
16 18.660507396 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=4/1024, ttl=64 (no response found!)
17 19.684504633 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=5/1280, ttl=64 (no response found!)
18 20.048409016 Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
19 20.708502009 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=6/1536, ttl=64 (no response found!)
20 21.732504484 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=7/1792, ttl=64 (no response found!)
21 22.053615132 Cisco_5c:4d:84  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8004
22 22.756505003 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=8/2048, ttl=64 (no response found!)
```

Excerto de logs de ping em broadcast no tux24

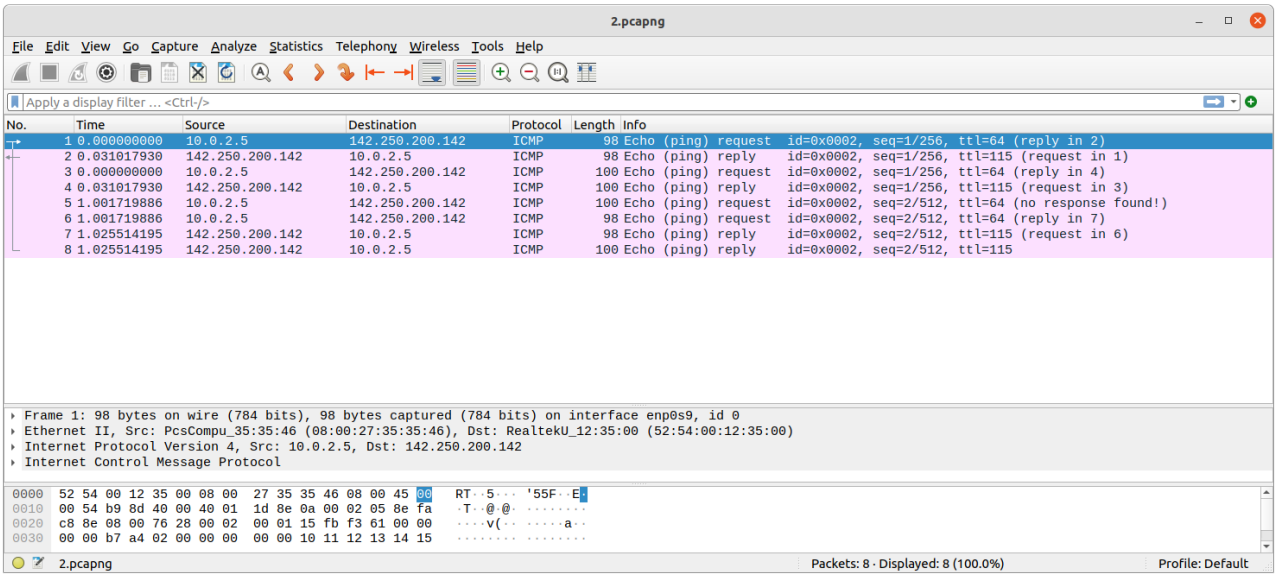
```
 4 3.562729870  172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=1/256, ttl=64 (no response found!)
 5 4.009458466  Cisco_5c:4d:83  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8003
 6 4.584723938  172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=2/512, ttl=64 (no response found!)
 7 5.608714698  172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=3/768, ttl=64 (no response found!)
 8 6.014314880  Cisco_5c:4d:83  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8003
 9 6.632702595  172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=4/1024, ttl=64 (no response found!)
10 7.656693844  172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=5/1280, ttl=64 (no response found!)
11 8.019189801  Cisco_5c:4d:83  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8003
12 8.680677760  172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=6/1536, ttl=64 (no response found!)
13 9.704669638  172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=7/1792, ttl=64 (no response found!)
14 10.024049498 Cisco_5c:4d:83  Spanning-tree-(for-… STP   60 Conf. Root = 32768/20/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8003
15 10.728658373 172.16.20.1     172.16.20.255     ICMP   98 Echo (ping) request  id=0x3a93, seq=8/2048, ttl=64 (no response found!)
```

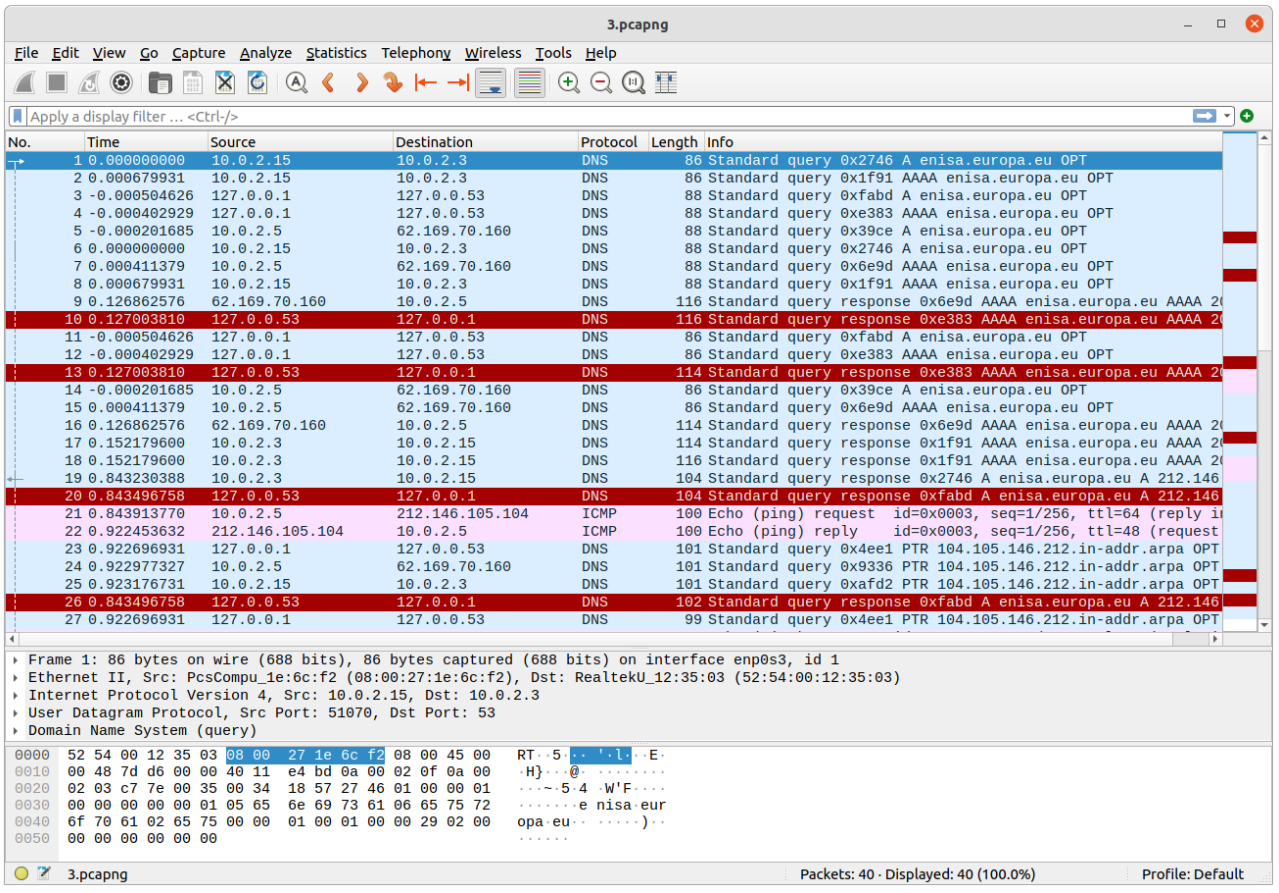
Excerto de logs de ping em broadcast no tux22

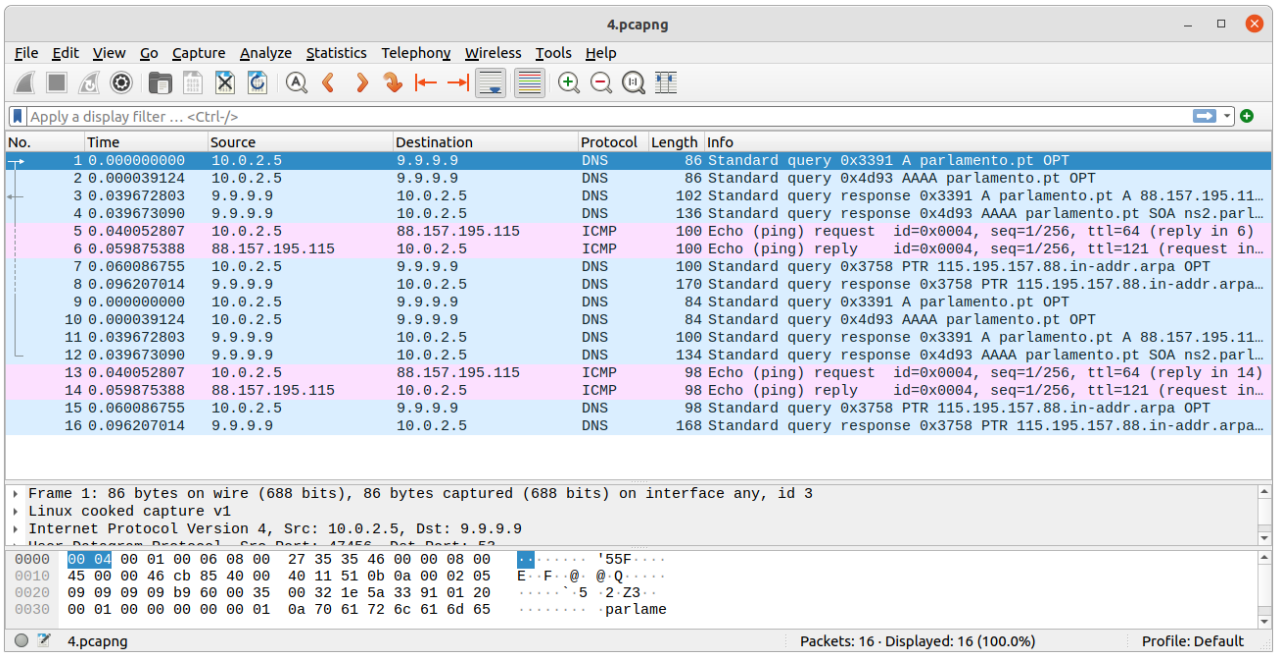
```
 5 7.860615076  Cisco_5c:4d:87  Cisco_5c:4d:87    LOOP   60 Reply
 6 8.036201343  Cisco_5c:4d:87  Spanning-tree-(for-… STP   60 Conf. Root = 32768/21/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8007
 7 10.041002999 Cisco_5c:4d:87  Spanning-tree-(for-… STP   60 Conf. Root = 32768/21/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8007
 8 12.045941592 Cisco_5c:4d:87  Spanning-tree-(for-… STP   60 Conf. Root = 32768/21/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8007
 9 14.050863055 Cisco_5c:4d:87  Spanning-tree-(for-… STP   60 Conf. Root = 32768/21/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8007
10 16.055696008 Cisco_5c:4d:87  Spanning-tree-(for-… STP   60 Conf. Root = 32768/21/fc:fb:fb:5c:4d:80  Cost = 0  Port = 0x8007
11 17.868046642 Cisco_5c:4d:87  Cisco_5c:4d:87    LOOP   60 Reply
```

6.3.3. Experiência nº3

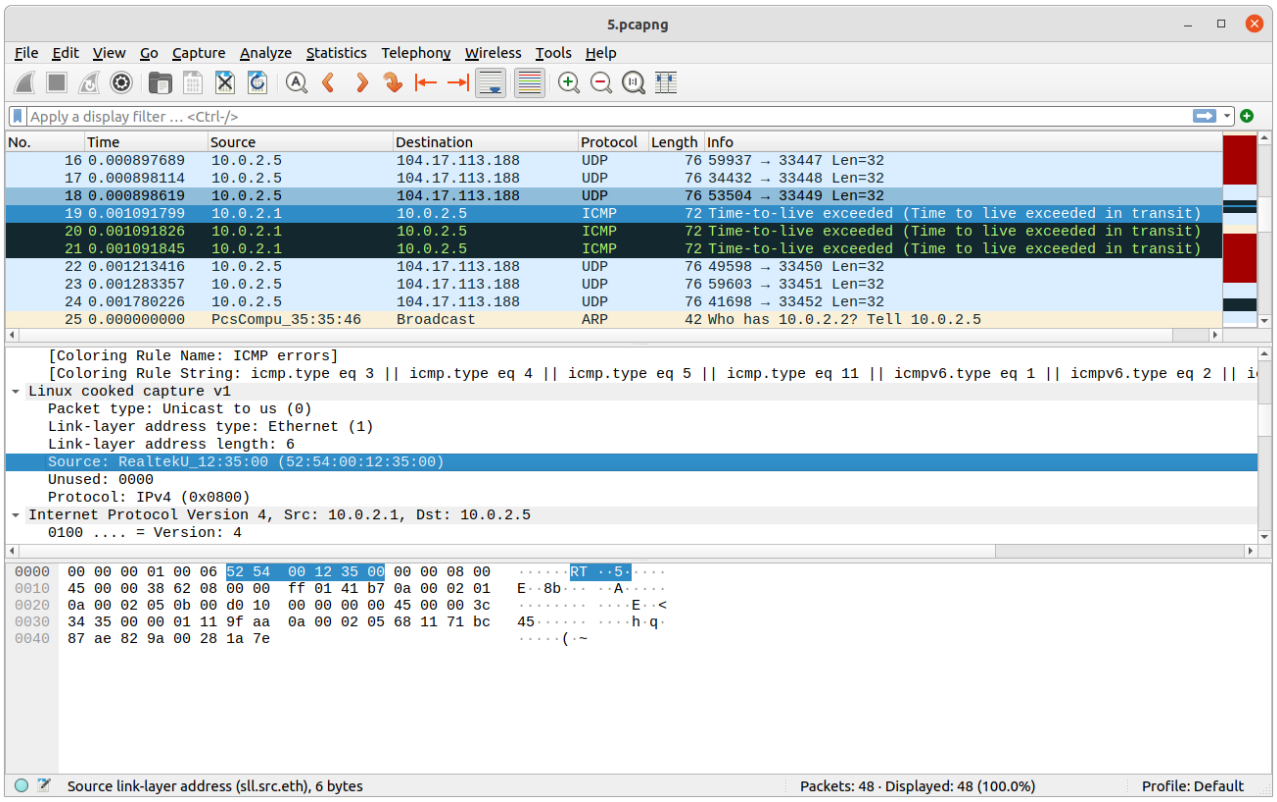
6.3.3.1. DNS Configs

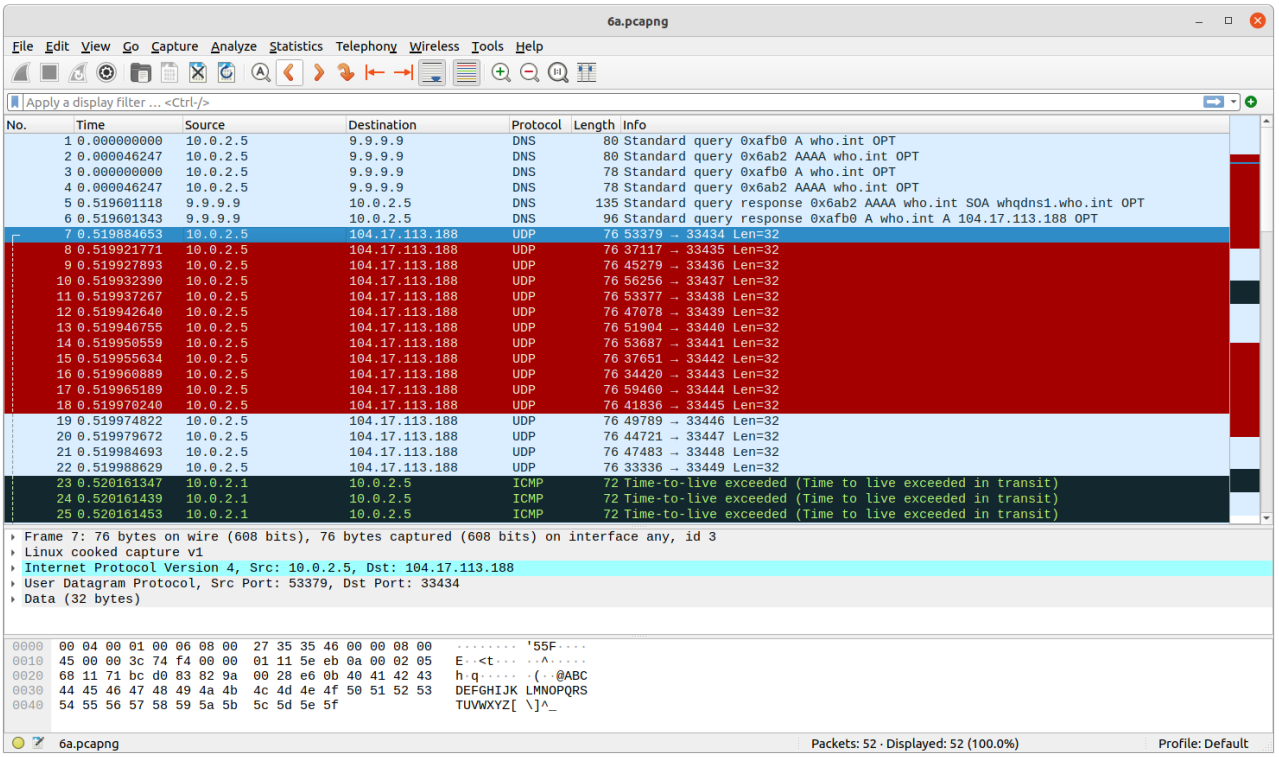






6.3.3.2. Linux Routing




6a.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000000 10.0.2.5 9.9.9.9 DNS 80 Standard query 0xafb0 A who.int OPT
2 0.000046247 10.0.2.5 9.9.9.9 DNS 80 Standard query 0x6ab2 AAAA who.int OPT
3 0.000000000 10.0.2.5 9.9.9.9 DNS 78 Standard query 0xafb0 A who.int OPT
4 0.000046247 10.0.2.5 9.9.9.9 DNS 78 Standard query 0x6ab2 AAAA who.int OPT
5 0.519601118 9.9.9.9 10.0.2.5 DNS 135 Standard query response 0x6ab2 AAAA who.int SOA whqdns1.who.int OPT
6 0.519601343 9.9.9.9 10.0.2.5 DNS 96 Standard query response 0xafb0 A who.int A 104.17.113.188 OPT
7 0.519884653 10.0.2.5 104.17.113.188 UDP 76 53379 → 33434 Len=32
8 0.519921771 10.0.2.5 104.17.113.188 UDP 76 37117 → 33435 Len=32
9 0.519927893 10.0.2.5 104.17.113.188 UDP 76 45279 → 33436 Len=32
10 0.519932390 10.0.2.5 104.17.113.188 UDP 76 56256 → 33437 Len=32
11 0.519937267 10.0.2.5 104.17.113.188 UDP 76 53377 → 33438 Len=32
12 0.519942640 10.0.2.5 104.17.113.188 UDP 76 47078 → 33439 Len=32
13 0.519946755 10.0.2.5 104.17.113.188 UDP 76 51904 → 33440 Len=32
14 0.519950559 10.0.2.5 104.17.113.188 UDP 76 53687 → 33441 Len=32
15 0.519955634 10.0.2.5 104.17.113.188 UDP 76 37651 → 33442 Len=32
16 0.519960889 10.0.2.5 104.17.113.188 UDP 76 34420 → 33443 Len=32
17 0.519965189 10.0.2.5 104.17.113.188 UDP 76 59460 → 33444 Len=32
18 0.519970240 10.0.2.5 104.17.113.188 UDP 76 41836 → 33445 Len=32
19 0.519974822 10.0.2.5 104.17.113.188 UDP 76 49789 → 33446 Len=32
20 0.519979672 10.0.2.5 104.17.113.188 UDP 76 44721 → 33447 Len=32
21 0.519984693 10.0.2.5 104.17.113.188 UDP 76 47483 → 33448 Len=32
22 0.519988629 10.0.2.5 104.17.113.188 UDP 76 33336 → 33449 Len=32
23 0.520161347 10.0.2.1 10.0.2.5 ICMP 72 Time-to-live exceeded (Time to live exceeded in transit)
24 0.520161439 10.0.2.1 10.0.2.5 ICMP 72 Time-to-live exceeded (Time to live exceeded in transit)
25 0.520161453 10.0.2.1 10.0.2.5 ICMP 72 Time-to-live exceeded (Time to live exceeded in transit)
Frame 7: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 3
Linux cooked capture v1
Internet Protocol Version 4, Src: 10.0.2.5, Dst: 104.17.113.188
User Datagram Protocol, Src Port: 53379, Dst Port: 33434
Data (32 bytes)
0000 00 04 00 01 00 06 08 00 27 35 35 46 00 00 08 00 .... '55F....
0010 45 00 00 3c 74 f4 00 00 01 11 5e eb 0a 00 02 05 E..<t..... ..^.....
0020 68 11 71 bc d0 83 82 9a 00 28 e6 0b 40 41 42 43 h.q..... .(..@ABC
0030 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 DEFGHIJK LMNOPQRS
0040 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f TUVWXYZ[\]^_
6a.pcapng
Packets: 52 · Displayed: 52 (100.0%)
Profile: Default