

Redes de Computadores

Protocolo de Ligação de Dados

(2º Trabalho Laboratorial)

Mestrado Integrado em Engenharia Informática e Computação

23 de dezembro de 2020

Autores:

Ana Teresa Feliciano da Cruz | up201806460@fe.up.pt

André Filipe Meireles do Nascimento | up201806461@fe.up.pt

Índice

Sumário	3
Introdução	3
Parte 1: Aplicação de Download	3
Arquitetura	3
Resultados	4
Parte 2: Configuração de Rede e Análise	5
Experiência 1 – Configurar uma rede IP	5
Experiência 2 – Implementar duas LANs virtuais no <i>switch</i>	6
Experiência 3 – Configurar um <i>router</i> em Linux	6
Experiência 4 – Configurar um <i>router</i> comercial e implementar NAT	7
Experiência 5 – DNS	8
Experiência 6 – Conexões TCP	8
Conclusões	10
Referências	10
Anexo I – Parte 1	11
Código Fonte	11
Download bem-sucedido	24
Anexo II – Parte 2: Configurações	25
Configuração Experiência 1	25
Configurações adicionais Experiência 2	26
Configurações adicionais Experiência 3	27
Configurações adicionais Experiência 4	28
Configurações adicionais Experiência 5	29
Anexo II – Parte 2: Logs	30
Experiência 1	30
Experiência 2	33
Experiência 3	35
Experiência 4	36
Experiência 5	37
Experiência 6	38

Sumário

Este relatório foi elaborado no âmbito da cadeira de Redes de Computadores, com o objetivo de complementar o segundo trabalho prático. Este trabalho consiste no desenvolvimento de uma aplicação capaz de transferir ficheiros de acordo com o protocolo FTP (*File Transfer Protocol*), e na configuração e estudo de uma rede utilizando comandos de configuração do *router* e do *switch*.

O trabalho foi concluído com sucesso, visto que foi elaborada uma aplicação capaz de transferir um ficheiro e todas as experiências foram concluídas.

Introdução

O trabalho tem duas grandes finalidades: o desenvolvimento de uma aplicação de *download* e a configuração de uma rede. Quanto ao relatório, o seu objetivo é explorar a parte teórica inerente ao trabalho, cumprindo a seguinte estrutura:

- **Parte 1: Aplicação de *Download***
Arquitetura da aplicação de *download* e respetivos resultados.
- **Parte 2: Configuração da Rede e respetiva análise**
Objetivos de cada experiência e a sua análise.
- **Conclusão**
Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

Parte 1: Aplicação de *Download*

A primeira parte do trabalho prático consistiu no desenvolvimento de uma aplicação de *download* na linguagem de programação C, que aceita um **URL** como argumento (`ftp://[<user>:<password>@]<host>/<url-path>`). A sua implementação exigiu o estudo de vários documentos, principalmente o RFC959 que aborda o protocolo de transferência de ficheiros (FTP).

Arquitetura

Para implementar a aplicação foram criadas duas camadas: a de processamento do URL (ficheiro *url_parser*) e a do cliente FTP (ficheiros *socket* e *ftp*).

A primeira camada trata do processamento da *string* passada como argumento (através da função ***parseURL***) e guarda todos os seus componentes na *struct url_args*. As variáveis *user*, *password* e *host* são obtidas diretamente do URL. O endereço ip e o *filename* são obtidos através de funções auxiliares (***getIP*** e ***getFilename***).

A segunda camada é responsável pela ligação do cliente FTP, começando por se conectar ao servidor FTP, através de um *socket* e, em seguida, envia um conjunto de FTP *requests* para obter o ficheiro desejado. Para envio de comandos é usada a função ***write_to_socket*** e, para a leitura dos códigos de resposta FTP é usada ***read_from_socket***. Após recebida uma resposta, o respetivo código é analisado e caso este não corresponda ao código de sucesso esperado, a aplicação fecha a ligação ao *socket* e termina.

Após a abertura do *socket*, pelo qual será feita a primeira conexão entre o cliente e o servidor, são enviados os comandos **USER <user>** e **PASS <password>** de forma a efetuar o *login*.

Posteriormente, é enviado o comando **TYPE I** para entrar em modo binário, de forma que quando os dados do ficheiro forem obtidos, eles sejam enviados como um fluxo de *bytes* binários, e o comando **PASV** para entrar em modo passivo, obtendo também o endereço da porta necessária para a abertura de um outro *socket* para a transferência de dados. Depois, é enviado o comando **RETR <url-path>** para pedir o ficheiro no caminho indicado e seguidamente é feita a sua transferência (com o auxílio da função **transferFile**). Finalmente, é enviado o comando **QUIT**, e as duas conexões aos *sockets* são fechadas.

Resultados

O nosso programa foi testado em várias condições: modo anónimo e não anónimo, vários tipos e tamanhos de ficheiros, e em vários *hosts*. Termina em caso de códigos de resposta inválida ou caso o ficheiro não exista. Ambos os comandos enviados e respostas recebidas são impressas na consola (comandos enviados são precedidos pelo carácter '>' e respostas recebidas por '<'). Um exemplo de um *download* feito com sucesso encontra-se em [anexo](#).

Parte 2: Configuração de Rede e Análise

Todas as experiências foram realizadas na bancada 4 do laboratório, sendo os endereços utilizados os correspondentes a essa bancada.

Experiência 1 – Configurar uma rede IP

O objetivo desta experiência é ligar o tux43 ao tux44 utilizando um *switch*, estabelecendo assim comunicação entre eles. A configuração efetuada pode ser observada em [anexo](#).

1. O que são os pacotes ARP e para que são usados?

Pacotes ARP (*Address Resolution Protocol*) são usados para obter o endereço **MAC** (*Medium Access Control*) associado ao endereço IP (*Internet Protocol*) fornecido.

2. Quais são os endereços MAC e IP dos pacotes ARP e porquê?

Após fazer *ping* do tux43 para o tux44, o tux43 envia um pacote a perguntar quem é o *tux* que tem o endereço IP de destino fornecido, de forma a obter o endereço MAC associado. Esta pergunta vem na forma de pacote ARP com o IP e MAC do **sender tux** (neste caso do tux43, **172.16.40.1** e **00:21:5a:61:2f:d4** respetivamente) e com o IP do **target tux** preenchido e o MAC, dado que é desconhecido, a 0's (no caso do tux44, **172.16.40.254** e **00:00:00:00:00:00**). Na resposta, o *sender tux* será agora o 44 e o pacote já contém a informação do seu endereço MAC (neste caso, **00:21:5a:5a:7b:ea**). A informação completa destes pacotes pode ser observada nas figuras [2](#) e [3](#) em anexo.

3. Quais os pacotes gerados pelo comando *ping*?

O comando *ping* gera pacotes **ICMP** (*Internet Control Message Protocol*) que são pacotes usados pelo *router* ou *host* para mandar erros da camada 3 ou mensagens de controlo para outros *hosts* ou *routers*.

4. Quais são os endereços MAC e IP dos pacotes *ping*?

Os endereços MAC e IP desses pacotes são os **endereços do tux que enviou o ping e do tux que o recebe**. Pode ser observado o pacote *ping* de pedido do tux43 para tux44 na [figura 4](#) e o pacote de resposta do tux44 ao tux43 na [figura 5](#).

5. Como determinar se a trama Ethernet recetora é ARP, IP, ICMP?

Observando o campo **Type** do **Ethernet header** de um pacote podemos determinar o tipo da trama. Caso este campo tenha o valor **0x0806** será então uma trama **ARP** ([figura 6](#)), caso tenha valor **0x0800** é uma trama **IP** ([figura 7](#)). No caso de tramas IP, observando o campo **Protocol** do **IP header**, podemos determinar o protocolo usado, caso tenha o valor **1** será então do tipo **ICMP** ([figura 7](#)).

6. Como determinar o comprimento da trama recetora?

Observando o campo **Frame Length** de uma trama no *wireshark* podemos obter o seu comprimento, como mostrado na [figura 8](#).

7. O que é a *interface loopback* e porque é tão importante?

A *interface loopback* é uma *interface* virtual da rede que permite ao computador receber respostas de si mesmo (exemplo na [figura 9](#)). É importante pois, permite confirmar que a rede está corretamente configurada caso consiga receber as respostas.

Experiência 2 – Implementar duas LANs virtuais no *switch*

O objetivo desta experiência é configurar duas LANs virtuais no *switch* e observar a comunicação entre os *tuxs*. Em adição às configurações anteriores foram efetuadas as em [anexo](#).

1. Como configurar VLANY0?

De forma a criar as VLANs, foi necessário ligar a porta **T3**, da régua 1, à porta **S0** do *tux* em que se pretende usar o GTKTerm (nas nossas experiências usamos o *tux43*). É também necessário ligar a porta **T4**, da régua 1, à porta do **Switch Console**, da régua 2. Para configurar as duas VLANs criadas, VLAN40 e 41, associamos os *tuxs* 43 e 44 à primeira e o *tux42* à segunda (inserindo no GTKTerm os comandos em [anexo](#)).

2. Quantos domínios de transmissão existem?

Através da análise dos *logs* é possível concluir que existem **dois domínios de transmissão**. Ao executar o comando *ping broadcast* (que envia um *ping* para todos os computadores na mesma rede) no *tux43*, verificamos que apenas o *tux44* recebe os *pings* (figuras [10](#), [11](#) e [12](#)), pois estão ambos ligados à VLAN40. Quando o *tux42* executa o comando nenhum dos outros dois *tuxs* recebe os *pings* (figuras [13](#), [14](#) e [15](#)), uma vez que está ligada a uma rede diferente, VLAN41. Desta forma, concluímos que há dois domínios, um para a VLAN40 e outro para a VLAN41.

Experiência 3 – Configurar um *router* em Linux

O objetivo desta experiência é configurar o *tux44* como um *router* estabelecendo assim uma ligação entre as duas VLANs anteriormente criadas. Em adição às configurações anteriores foram efetuadas as em [anexo](#).

1. Que rotas existem nos *tuxs*? Qual é o seu significado?

Para as VLANs associadas há as seguintes rotas: *tux43* tem uma rota para a VLAN40 (**172.16.40.0**) pela *gateway* **172.16.40.1**; *tux44* tem uma rota para a VLAN40 pela *gateway* **172.16.40.254** e uma rota para a VLAN41 (**172.16.41.0**) pela *gateway* **172.16.41.253**; *tux42* tem uma rota para a VLAN41 pela *gateway* **172.16.41.1**.

Nesta experiência foram também adicionadas as seguintes rotas: *tux43* tem uma rota para a VLAN41 pela *gateway* **172.16.40.254**; *tux42* tem uma rota para a VLAN40 pela *gateway* **172.16.41.253**.

O primeiro endereço da *gateway* identifica a gama de endereços para a qual se quer adicionar a *gateway*, ou seja, os possíveis endereços de destino, e o segundo endereço identifica o IP para o qual se deve encaminhar o pacote. Assim sendo, com as *gateways* criadas, todos os *tuxs* conseguem comunicar entre si.

2. Que informação contém uma entrada da *forwarding table*?

Cada entrada contém o destino da rota (**Destination**), o IP do próximo ponto por onde passará a rota (**Gateway**), a máscara da rede (**Netmask**), informações sobre a rota (**Flags**), o custo da rota (**Metric**), o número de referências para esta rota (**Ref**), contador de pesquisas pela rota (**Use**) e, finalmente, qual a placa de rede responsável (**Interface**) que neste caso será eth0 ou eth1.

3. Que mensagens ARP e endereços MAC associados são observados e porquê?

Com todas as rotas definidas é possível executar o comando *ping* para todas as *interfaces* a partir do tux43, como se pode verificar na [figura 16](#). Nesta figura pode-se também observar a troca de pacotes ARP entre o tux43 e a *interface* eth0 do tux44 para obterem o respetivo endereço MAC de destino.

4. Que pacotes ICMP são observados e porquê?

São observados pacotes ICMP de **request** e **reply**, pois depois de serem adicionadas as rotas, todos os *tuxs* conseguem comunicar entre si. Caso não conseguissem seriam enviados pacotes ICMP de **Destination unreachable**.

5. Quais são os endereços IP e MAC associados aos pacotes ICMP e porquê?

Os endereços IP e MAC associados com os pacotes ICMP são os endereços IP e MAC dos respetivos *tuxs* de origem e destino. Por exemplo, ao executar o comando *ping* do tux43 para o tux44.eth0 os endereços de origem são 172.16.40.1 (IP) e 00:21:5a:61:2f:d4 (MAC), e os de destino são 172.16.40.254 (IP) e 00:21:5a:5a:7b:ea (MAC) como se pode observar na [figura 17](#).

Experiência 4 – Configurar um *router* comercial e implementar NAT

O objetivo desta experiência é configurar inicialmente o *router* do laboratório sem NAT (*Network Address Translation*), e, posteriormente, implementar o NAT, observando a comunicação em ambos os casos. Em adição às configurações anteriores foram efetuadas as em [anexo](#).

1. Como se configura uma rota estática num *router* comercial?

De forma a configurar o *router*, foi necessário ligar a porta **T3**, da régua 1, à porta **S0** do *tux* em que se pretende usar o GTKTerm (nas nossas experiências usamos o tux43). É também necessário ligar a porta **T4**, da régua 1, à porta do **Router Console**, da régua 2. Começamos por configurar a interface GE 0/0 do *router* atribuída à VLAN41, e para a interface GE 0/1 do *router* atribuiu-se o IP **172.16.1.49** para que fosse feita a ligação com a rede dos laboratórios. Configurou-se o *router* definindo as rotas internas e externas com o comando `ip route` na consola do GTKTerm. De seguida, definiu-se o tux44 como *default gateway* do tux43 e o RC como *default gateway* dos *tuxs* 44 e 42.

2. Quais são as rotas seguidas pelos pacotes nas experiências e porquê?

Com todas as rotas configuradas verificou-se que o tux43 conseguia enviar *pings* a todas as outras *interfaces* dos *tuxs* e ao *router* (RC) do laboratório ([figura 18](#)).

Posteriormente, removemos a rota do tux42 para a VLAN40 e verificamos que ao executar *ping* para o tux43, como o tux42 já não tinha uma rota definida, o *router* (RC) com IP

172.16.41.254, definido como *default gateway* do tux42 passou a ser responsável por redirecionar os pacotes ICMP até ao tux43 ([figura 19](#)).

3. Como configurar NAT num *router* comercial?

De forma a configurar NAT no *router* foi seguida a lista de comandos presente no slide 46 do guião do 2º trabalho prático ([figura 20](#)), inseridos no GTKTerm.

4. O que faz o NAT?

O NAT é um mecanismo implementado em *routers* que substitui os endereços IP locais nos pacotes por um endereço IP público de forma a que um computador de uma rede interna tenha acesso ao exterior. Sendo assim, o *router* que implementa o NAT torna-se responsável por encaminhar todos os pacotes para o endereço correto, dentro ou fora da rede local.

Experiência 5 – DNS

O objetivo desta experiência é configurar DNS (*Domain Name System*) nos tuxs 42, 43 e 44. Em adição às configurações anteriores foram efetuadas as em [anexo](#).

1. Como configurar o serviço DNS num *host*?

Para configurar o serviço DNS é necessário editar o ficheiro ***resolve.conf*** no respetivo *tux*. Para tal foi executado o comando `echo '$search netlab.fe.up.pt\nnameserver 172.16.1.1' > /etc/resolv.conf` no terminal de cada um dos 3 tuxs (**netlab.fe.up.pt** é o nome do servidor DNS e **172.16.1.1** é o seu endereço IP).

2. Que pacotes são trocados pelo DNS e que informação é transportada?

Ao realizar um *ping* são enviados 2 pacotes DNS com dois pedidos: um do tipo **A** (*Address Mapping Record*) para pedir o endereço IPv4 do *host*, e outro do tipo **AAAA** (*IP Version 6 Address Record*) para pedir o endereço IPv6. Ambos os pedidos recebem respostas com os endereços IP respetivos (para "ftp.up.pt" o endereço IPv4 é 193.137.29.15 e o endereço IPv6 é 2001:690:2200:1200::15). Após um *ping* ser enviado com sucesso e obter resposta, é enviado outro pedido DNS, agora do tipo **PTR** (*Reverse-lookup Pointer Record*) para, a partir do endereço IP obtido anteriormente, encontrar todos os *hostnames*. Esta troca de pacotes pode ser observada na [figura 21](#).

Experiência 6 – Conexões TCP

O objetivo desta experiência é observar o comportamento do protocolo TCP (*Transmission Control Protocol*) utilizando a aplicação de *download* desenvolvida. Para esta experiência não foram feitas alterações às configurações.

1. Quantas conexões TCP foram abertas pela aplicação FTP?

A aplicação abre **duas conexões** TCP: uma usada para envio de comandos FTP e obter as respostas e outra usada para receber os dados enviados pelo servidor.

2. Em que conexão é transportado o controlo de informação FTP?

O controlo de informação é transportado na conexão responsável pelo envio de comandos.

3. Quais as fases de uma conexão TCP?

Uma conexão TCP tem três fases: estabelecimento da conexão, troca de dados e encerramento da conexão.

4. Como funciona o mecanismo ARQ TCP? Quais os campos TCP relevantes? Qual a informação relevante observada nos logs?

O TCP utiliza o mecanismo **ARQ** (*Automatic Repeat Request*) com o método da janela deslizante, que é um mecanismo de controlo de erros na transmissão de dados. Pacotes de controlo **ACK** são enviados por parte do recetor indicando que a trama foi recebida corretamente.

Como se pode observar pelo cabeçalho TCP, existem permanentemente um par de campos relevantes, sendo eles o **Sequence number**, que identifica o primeiro *byte* dos dados, e o **Acknowledgment number**, que indica o próximo *byte* que o recetor espera. O emissor determina o seu próprio número de sequência e o recetor confirma o segmento usando como número ACK o número de sequência do emissor. Para manter a confiabilidade, o recetor confirma os segmentos indicando que recebeu um determinado número de *bytes* contíguos. Há também o campo **Window Size** que indica a quantidade máxima de dados que o emissor pode enviar. Estes campos podem ser observados nas figuras [22](#) e [23](#).

5. Como é que o mecanismo de controlo de congestão TCP funciona? Como é que o fluxo de dados da conexão evoluiu ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?

Para controlar o congestionamento o TCP usa uma **congestion window** no lado do emissor, que indica a quantidade máxima de dados que podem ser enviados sem receber ACK. Quando o congestionamento na rede aumenta (por ocorrência de *timeout* causando *packet loss*) a *congestion window* diminui, e o inverso também se verifica.

No *download* realizado no tux43, a taxa de transferência foi aumentando até atingir um pico por volta dos 3 segundos (confirmando assim o **slow start**), e manteve-se sem grandes alterações até aos 6 segundos, devido ao início de outro *download* no tux42 que provocou o aumento do congestionamento na rede e causando assim a diminuição da taxa. Quando o segundo *download* acabou, a taxa voltou a subir graças à diminuição de congestionamento. O gráfico destes *downloads* pode ser observado no [gráfico 1](#).

6. De que forma é afetada a conexão de dados TCP pelo aparecimento de uma segunda conexão TCP? Como?

Com o aparecimento de uma segunda conexão TCP, a realização de uma transferência de dados em simultâneo leva inicialmente a uma queda na taxa de transmissão, uma vez que a taxa de transferência é distribuída de igual forma para cada ligação. Após a segunda transferência terminar e a conexão ser fechada a taxa de transmissão volta a subir atingindo novamente um pico.

Conclusões

A aplicação de *download* alcançou os resultados desejados, sendo capaz de transferir ficheiros de vários servidores FTP usando o padrão FTP. O seu desenvolvimento permitiu entender melhor o funcionamento deste protocolo.

As experiências desenvolvidas no laboratório e a análise dos respetivos *logs* permitiram o entendimento e consolidação de diversos aspetos de configuração de redes, como configuração de *interfaces ethernet* em máquinas Linux, configuração de *lans* virtuais com várias máquinas, configurações de *routers* Linux para troca de pacotes entre diferentes *lans* virtuais e configuração de NAT e DNS num *router* comercial.

No entanto, queremos realçar as dificuldades sentidas na execução da Parte 2 deste trabalho, devido à restrição do número de pessoas em laboratório, não permitindo que o grupo estivesse em conjunto a trabalhar. Tal causou um atraso na aprendizagem por parte do membro que ficava em casa. Apesar disso, conseguimos realizar todas as experiências com sucesso e analisar os resultados.

Referências

- *RFC 959: File Transfer Protocol* - <https://www.w3.org/Protocols/rfc959/>
- *IP Addressing: NAT Configuration Guide* - https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/xe-16/nat-xe-16-book/iadnat-addr-consv.html
- *DNS Types* - <https://ns1.com/resources/dns-types-records-servers-and-queries>
- *TCP Congestion Control* - <https://www.noction.com/blog/tcp-transmission-control-protocol-congestion-control>

Anexo I – Parte 1

Código Fonte

- download.c

```
#include "ftp.h"

int main(int argc, char* argv[]) {
    if(argc!=2) {
        fprintf(stderr, "usage: download
ftp://[<user>:<password>@]<host>/<url-path>\n");
        exit(1);
    }

    url_args url;

    if(parseURL(argv[1], &url)!=0){
        fprintf(stderr, "Error parsing url!\n");
        exit(1);
    }

    printURL(url);

    //init
    if(ftp_init(url.ip, 21) != 0){
        fprintf(stderr, "Error service no ready!\n");
        exit(1);
    }

    //login
    if(ftp_login(url.user, url.pass) != 0){
        fprintf(stderr, "Error logging in!\n");
        exit(1);
    }

    //binary mode
    if(ftp_binmode() != 0){
        fprintf(stderr, "Error setting binary mode!\n");
        exit(1);
    }

    //download file
    if(ftp_download(url.path, url.filename) != 0){
        fprintf(stderr, "Error downloading file!\n");
        exit(1);
    }

    //close socket
    if(ftp_quit() != 0){
        fprintf(stderr, "Error closing socket!\n");
        exit(1);
    }

    return 0;
}
```

- url_parser.h

```
#pragma once

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

#define h_addr h_addr_list[0]    //The first address in h_addr_list.

typedef struct {
    char user[256];
    char pass[256];
    char host[512];
    char path[512];
    char filename[512];
    char host_name[512];
    char ip[20];
} url_args;

int parseURL(char* args, url_args *url);
int getIP(url_args *url);
int getFilename(url_args *url);
void printURL(url_args url);
```

- url_parser.c

```
#include "url_parser.h"

int parseURL(char* args, url_args *url) {
    char* ftp = strtok(args, "/");
    char* urlrest = strtok(NULL, "/");
    char* path = strtok(NULL, "");

    if (strcmp(ftp, "ftp:") != 0) {
        fprintf(stderr, "Error: Not using ftp\n");
        return -1;
    }

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

    // default user and pass
    if (pass == NULL) {
        user = "anonymous";
        pass = "pass";
        strcpy(url->host, urlrest);
    }
    else {
        char* host = strtok(NULL, "");
        strcpy(url->host, host);
    }

    strcpy(url->path, path);
    strcpy(url->user, user);
    strcpy(url->pass, pass);

    if (getIP(url) != 0){
        fprintf(stderr, "Error: getIp()\n");
        return -1;
    }

    if (getFilename(url) != 0){
        fprintf(stderr, "Error: getFileName()\n");
        return -1;
    }

    return 0;
}

int getIP(url_args *url){
    struct hostent *h;

    if ((h=gethostbyname(url->host)) == NULL) {
        perror("gethostbyname");
        return -1;
    }

    char* host_name = h->h_name;
    strcpy(url->host_name, host_name);
}
```

```

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

return 0;
}

int getFilename(url_args *url){
char* filename = url->path;
char* p;
for(p = url->path; *p; p++){
    if(*p == '/' || *p == '\\ || *p == ':'){
        filename = p+1;
    }
}

strcpy(url->filename, filename);

return 0;
}

void printURL(url_args url){
printf("\nUser: %s\n", url.user);
printf("Password: %s\n", url.pass);
printf("Host: %s\n", url.host);
printf("Path: %s\n", url.path);
printf("Filename: %s\n", url.filename);
printf("Host name: %s\n", url.host_name);
printf("IP address: %s\n\n", url.ip);
}

```

- ftp.h

```
#pragma once

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

#include "rfc959.h"
#include "url_parser.h"
#include "socket.h"

int ftp_init(char *ip, int port);
int ftp_quit();
int ftp_login(char* user, char* pass);
int ftp_binarymode();
int ftp_pasvmode(char *ip, int *port);
int ftp_retr(char *file);
int ftp_download(char *path, char *filename);
int transferFile(int datafd, char *filename);
```

- ftp.c

```
#include "ftp.h"

static int sockfd;
static int datafd;
static char command[256];

int ftp_init(char *ip, int port){
    int response_code;

    if((sockfd = init_connection(ip, 21))<0){
        fprintf(stderr, "Error initializing connection!\n");
        return -1;
    }

    response_code = read_from_socket(sockfd);
    if(response_code != SERVICE_READY_CODE){
        fprintf(stderr, "Error, not 220\n");
        close(sockfd);
        return -1;
    }

    return 0;
}

int ftp_quit(){
    sprintf(command, "QUIT\r\n");
    if(write_to_socket(sockfd, command) != 0){
        close(sockfd);
        return -1;
    }

    close(sockfd);
    return 0;
}

int ftp_login(char* user, char* pass){
    int response_code;

    //Send user
    sprintf(command, "USER %s\r\n", user);
    if(write_to_socket(sockfd, command) != 0){
        close(sockfd);
        return -1;
    }

    response_code = read_from_socket(sockfd);
    if(response_code == USER_LOGGED_IN_CODE){
        return 0;
    }
    else if(response_code != NEED_PASSWORD_CODE){
        fprintf(stderr, "Error, not 331\n");
        close(sockfd);
        return -1;
    }
}
```



```

//Send pass
sprintf(command, "PASS %s\r\n", pass);
if(write_to_socket(socketfd, command) != 0){
    close(socketfd);
    return -1;
}

response_code = read_from_socket(socketfd);
if(response_code != USER_LOGGED_IN_CODE){
    fprintf(stderr, "Error, not 230\n");
    close(socketfd);
    return -1;
}

return 0;
}

int ftp_binarymode(){
    int response_code;

    sprintf(command, "TYPE I\r\n");
    if(write_to_socket(socketfd, command) != 0){
        close(socketfd);
        return -1;
    }

    response_code = read_from_socket(socketfd);
    if(response_code != COMMAND_OKAY_CODE){
        fprintf(stderr, "Error, not 200\n");
        close(socketfd);
        return -1;
    }

    return 0;
}

int ftp_pasvmode(char *ip, int *port){
    int response_code;

    sprintf(command, "PASV\r\n");
    if(write_to_socket(socketfd, command) != 0){
        close(socketfd);
        return -1;
    }

    response_code = read_passivemode(socketfd, ip, port);
    if(response_code != ENTER_PASSIVE_MODE_CODE){
        fprintf(stderr, "Error, not 227\n");
        close(socketfd);
        return -1;
    }

    return 0;
}

int ftp_retr(char *file){
    int response_code;

```

```

    sprintf(command, "RETR %s\r\n", file);
    if(write_to_socket(socketfd, command) != 0){
        close(socketfd);
        return -1;
    }

    response_code = read_from_socket(socketfd);
    if(response_code != RETR_FILE_OKAY_CODE){
        fprintf(stderr, "Error, not 150\n");
        close(socketfd);
        return -1;
    }

    return 0;
}

int ftp_download(char *path, char *filename){
    char ip[32];
    int port;

    //Enter passive mode
    if(ftp_pasvmode(ip, &port) != 0){
        fprintf(stderr, "Error entering passive mode\n");
        close(socketfd);
        return -1;
    }

    printf("%s %d\n", ip, port);

    //Open new socket for file download
    if((datafd = init_connection(ip, port)) < 0){
        fprintf(stderr, "Error initializing data connection!\n");
        close(socketfd);
        return -1;
    }

    //Retrieve file
    if(ftp_retr(path) != 0){
        fprintf(stderr, "Error file status!\n");
        close(socketfd);
        return -1;
    }

    //Download file
    if(transferFile(datafd, filename) != 0){
        fprintf(stderr, "Error transferring file!\n");
        close(socketfd);
        return -1;
    }

    //Close data socket
    close(datafd);
    int response_code = read_from_socket(socketfd);
    if(response_code != RETR_SUCCESS_CODE){
        fprintf(stderr, "Error, not 226\nError retrieving file\n");
        close(socketfd);
    }
}

```

```

        return -1;
    }

    return 0;
}

int transferFile(int datafd, char *filename) {
    int filefd;

    if ((filefd = open(filename, O_WRONLY | O_CREAT, 0777)) < 0) {
        fprintf(stderr, "Error opening data file!\n");
        return -1;
    }

    char buf[1024];
    int numBytesRead;

    while((numBytesRead = read(datafd, buf, 1024)) > 0) {
        if (write(filefd, buf, numBytesRead) < 0) {
            fprintf(stderr, "Error writing data to file!\n");
            return -1;
        }
    }

    if (close(filefd) < 0) {
        fprintf(stderr, "Error closing file!\n");
        return -1;
    }
    return 0;
}

```

- socket.h

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

int init_connection(char *ip, int port);
int write_to_socket(int sockfd, char *message);
int read_from_socket(int sockfd);
int read_passivemode(int sockfd, char *ip, int *port);
```

- socket.c

```
#include "socket.h"

int init_connection(char* ip, int 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(ip);      /*32 bit Internet
address network byte ordered*/
    server_addr.sin_port = htons(port);              /*server TCP port
must be network byte ordered */

    /*open an TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        fprintf(stderr, "Error opening socket!\n");
        return -1;
    }

    /*connect to the server*/
    if(connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr))
< 0){
        fprintf(stderr, "Error connecting to server!\n");
        return -1;
    }

    return sockfd;
}

int write_to_socket(int sockfd, char *message) {
    int bytes;
    int msg_len = strlen(message);

    if((bytes = write(sockfd, message, msg_len)) != msg_len){
        fprintf(stderr, "Error writing message to socket!\n");
        return -1;
    }

    printf("> %s", message);
    return 0;
}

int read_from_socket(int sockfd){
    FILE* fp = fdopen(sockfd, "r");

    char* buf;
    size_t bytesRead = 0;
    int response_code;

    while(getline(&buf, &bytesRead, fp) > 0){
        printf("< %s", buf);
        if(buf[3] == ' '){
            sscanf(buf, "%d", &response_code);
            break;
        }
    }
}
```

```

        }
    }

    return response_code;
}

int read_passivemode(int sockfd, char *ip, int *port){
    FILE* fp = fdopen(sockfd, "r");

    char* buf;
    size_t bytesRead = 0;
    int response_code;

    while(getline(&buf, &bytesRead, fp) > 0){
        printf("< %s", buf);
        if(buf[3] == ' '){
            sscanf(buf, "%d", &response_code);
            break;
        }
    }

    //parse ip address and port
    char *find;
    int ip1, ip2, ip3, ip4, p1, p2;

    find = strrchr(buf, '(');
    sscanf(find,("(%d,%d,%d,%d,%d,%d)", &ip1, &ip2, &ip3, &ip4, &p1,
&p2);

    sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);

    *port = p1*256+p2;

    return response_code;
}

```

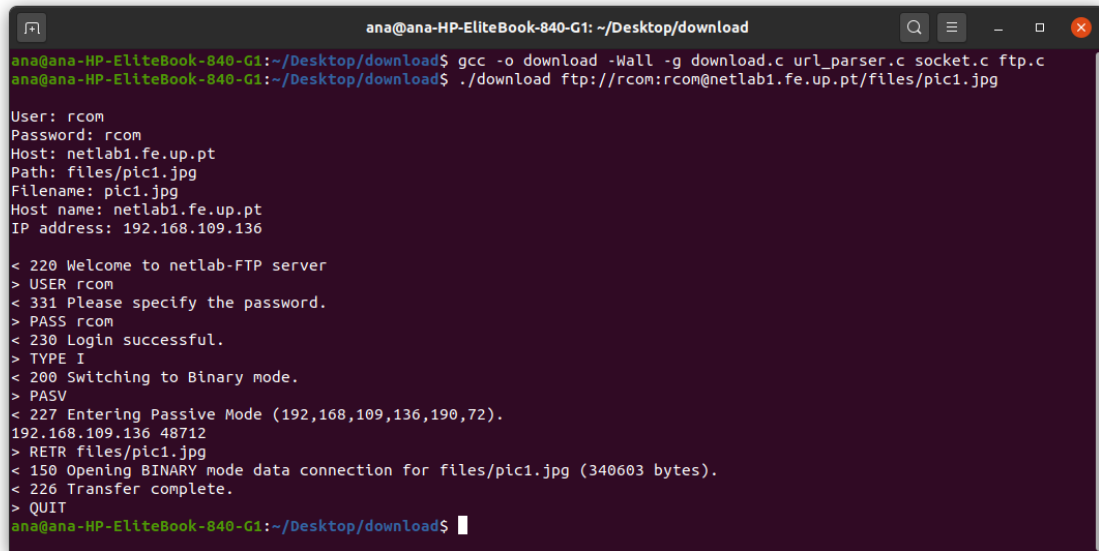
- ffc959.h

```
// 4.2.1 Section - https://www.w3.org/Protocols/rfc959/4\_FileTransfer.html

#define COMMAND_OKAY_CODE      200 //Command okay.
#define SERVICE_READY_CODE     220 //Service ready for new user.
#define SERVICE_CLOSING_CODE   221 //Service closing control connection.
#define NEED_PASSWORD_CODE     331 //User name okay, need password.
#define USER_LOGGED_IN_CODE    230 //User logged in, proceed.
#define CWD_OKAY_CODE          250 //Requested file action okay, completed.
#define ENTER_PASSIVE_MODE_CODE 227 //Entering Passive Mode
(h1,h2,h3,h4,p1,p2).
#define RETR_FILE_OKAY_CODE     150 //File status okay; about to open data
connection.
#define RETR_TRANSF_START_CODE  125 //Data connection already open; transfer
starting.
#define RETR_SUCCESS_CODE       226 //Closing data connection. Requested file
action successful
#define ACTION_FAILED_CODE      550 //Requested action not taken. File
unavailable (e.g., file not found, no access).

// Command Terminator
#define CRLF "\r\n"
```

Download bem-sucedido



```
ana@ana-HP-EliteBook-840-G1: ~/Desktop/download
ana@ana-HP-EliteBook-840-G1:~/Desktop/download$ gcc -o download -Wall -g download.c url_parser.c socket.c ftp.c
ana@ana-HP-EliteBook-840-G1:~/Desktop/download$ ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/pic1.jpg

User: rcom
Password: rcom
Host: netlab1.fe.up.pt
Path: files/pic1.jpg
Filename: pic1.jpg
Host name: netlab1.fe.up.pt
IP address: 192.168.109.136

< 220 Welcome to netlab-FTP server
> USER rcom
< 331 Please specify the password.
> PASS rcom
< 230 Login successful.
> TYPE I
< 200 Switching to Binary mode.
> PASV
< 227 Entering Passive Mode (192,168,109,136,190,72).
192.168.109.136 48712
> RETR files/pic1.jpg
< 150 Opening BINARY mode data connection for files/pic1.jpg (340603 bytes).
< 226 Transfer complete.
> QUIT
ana@ana-HP-EliteBook-840-G1:~/Desktop/download$
```

Figura 1: Download de uma imagem com sucesso

Anexo II – Parte 2: Configurações

Configuração Experiência 1

- **Tux43**

```
ifconfig eth0 up  
ifconfig eth0 172.16.40.1/24
```

- **Tux44**

```
ifconfig eth0 up  
ifconfig eth0 172.16.40.254/24
```

Configurações adicionais Experiência 2

- **Tux42**

```
ifconfig eth0 up  
ifconfig eth0 172.16.41.1/24
```

- **Switch**

```
enable  
(password:) 8nortel  
conf t  
vlan 40  
vlan 41  
interface Fa0/2  
switchport mode access  
switchport access vlan 41  
exit  
interface Fa0/3  
switchport mode access  
switchport access vlan 40  
exit  
interface Fa0/4  
switchport mode access  
switchport access vlan 40  
end
```

Configurações adicionais Experiência 3

- **Tux44**

```
ifconfig eth1 up  
ifconfig eth1 172.16.41.253/24
```

- **Tux43**

```
route add -net 172.16.41.0/24 gw 172.16.40.254
```

- **Tux42**

```
route add -net 172.16.40.0/24 gw 172.16.41.253
```

- **Switch**

```
enable  
(password:) 8nortel  
conf t  
vlan 41  
interface Fa0/5  
switchport mode access  
switchport access vlan 41  
end
```

Configurações adicionais Experiência 4

- **Router**

```
(username:) root
(password:) 8nortel
conf t
interface Gi0/0
ip address 172.16.41.254 255.255.255.0
no shutdown
ip nat inside
exit
interface Gi0/1
ip address 172.16.1.49 255.255.255.0
no shutdown
ip nat outside
exit
ip nat pool ovrld 172.16.1.49 172.16.1.49 prefix 24
ip nat inside source list 1 pool ovrld overload
access-list 1 permit 172.16.40.0 0.0.0.7
access-list 1 permit 172.16.41.0 0.0.0.7
ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.40.0 255.255.255.0 172.16.41.253
end
```

- **Tux42**

```
route add -net 172.16.40.0/24 gw 172.16.41.253
route add default gw 172.16.41.254
```

- **Tux43**

```
route add default gw 172.16.40.254
```

- **Tux44**

```
route add default gw 172.16.41.254
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Configurações adicionais Experiência 5

- **Tux42**

```
echo '$'search netlab.fe.up.pt\nnameserver 172.16.1.1' >  
/etc/resolv.conf
```

- **Tux43**

```
echo '$'search netlab.fe.up.pt\nnameserver 172.16.1.1' >  
/etc/resolv.conf
```

- **Tux44**

```
echo '$'search netlab.fe.up.pt\nnameserver 172.16.1.1' >  
/etc/resolv.conf
```

Anexo II – Parte 2: Logs

Para uma melhor interpretação dos dados foram sublinhados alguns componentes, seguindo o seguinte padrão:

- **tux43.eth0**
- **tux44.eth0**
- **tux44.eht1**
- **tux42.eth0**
- RC

Experiência 1

19	25.567736350	HewlettP_61:2f:d4 Broadcast	ARP	42	Who has <u>172.16.40.254</u> ? Tell <u>172.16.40.1</u>
20	25.567869674	HewlettP_5a:7b:ea HewlettP_61:2f:d4	ARP	60	<u>172.16.40.254</u> is at <u>00:21:5a:5a:7b:ea</u>

```
> Frame 19: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
< Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
  Sender IP address: 172.16.40.1
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 172.16.40.254
```

Figura 2: Pacote ARP para descobrir o endereço MAC associado ao IP de destino fornecido

19	25.567736350	HewlettP_61:2f:d4 Broadcast	ARP	42	Who has <u>172.16.40.254</u> ? Tell <u>172.16.40.1</u>
20	25.567869674	HewlettP_5a:7b:ea HewlettP_61:2f:d4	ARP	60	<u>172.16.40.254</u> is at <u>00:21:5a:5a:7b:ea</u>

```
> Frame 20: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea), Dst: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
< Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)
  Sender IP address: 172.16.40.254
  Target MAC address: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
  Target IP address: 172.16.40.1
```

Figura 3: Pacote ARP com todos os endereços preenchidos

No.	Time	Source	Destination	Protocol	Length	Info
21	25.567888112	<u>172.16.40.1</u>	<u>172.16.40.254</u>	ICMP	98	Echo (ping) request id=0x1150, seq=1/256, ttl=64 (reply in 22)
22	25.568024649	<u>172.16.40.254</u>	<u>172.16.40.1</u>	ICMP	98	Echo (ping) reply id=0x1150, seq=1/256, ttl=64 (request in 21)
23	26.068513227	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/1/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
24	26.581196648	<u>172.16.40.1</u>	<u>172.16.40.254</u>	ICMP	98	Echo (ping) request id=0x1150, seq=2/512, ttl=64 (reply in 25)
25	26.581353299	<u>172.16.40.254</u>	<u>172.16.40.1</u>	ICMP	98	Echo (ping) reply id=0x1150, seq=2/512, ttl=64 (request in 24)
26	27.605197207	<u>172.16.40.1</u>	<u>172.16.40.254</u>	ICMP	98	Echo (ping) request id=0x1150, seq=3/768, ttl=64 (reply in 27)
27	27.605328855	<u>172.16.40.254</u>	<u>172.16.40.1</u>	ICMP	98	Echo (ping) reply id=0x1150, seq=3/768, ttl=64 (request in 26)

```
> Frame 21: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4), Dst: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)
> Internet Protocol Version 4, Src: 172.16.40.1, Dst: 172.16.40.254
> Internet Control Message Protocol
```

Figura 4: Pacote ping do pedido de tux43.eth0 a tux44.eth0

No.	Time	Source	Destination	Protocol	Length	Info
21	25.567888112	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1150, seq=1/256, ttl=64 (reply in 22)
22	25.568024649	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1150, seq=1/256, ttl=64 (request in 21)
23	26.068513227	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/1/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
24	26.581196648	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1150, seq=2/512, ttl=64 (reply in 25)
25	26.581353299	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1150, seq=2/512, ttl=64 (request in 24)
26	27.605197207	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1150, seq=3/768, ttl=64 (reply in 27)
27	27.605328855	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1150, seq=3/768, ttl=64 (request in 26)

> Frame 22: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
 > Ethernet II, Src: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea), Dst: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
 > Internet Protocol Version 4, Src: 172.16.40.254, Dst: 172.16.40.1
 > Internet Control Message Protocol

Figura 5: Pacote ping de resposta do tux44.eth0 a tux43.eth0

No.	Time	Source	Destination	Protocol	Length
19	25.567736350	HewlettP_61:2f:d4	Broadcast	ARP	42

> Frame 19: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
 > Ethernet II, Src: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
 > Source: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
 Type: ARP (0x0806)
 > Address Resolution Protocol (request)
 Hardware type: Ethernet (1)
 Protocol type: IPv4 (0x0800)
 Hardware size: 6
 Protocol size: 4
 Opcode: request (1)
 Sender MAC address: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
 Sender IP address: 172.16.40.1
 Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
 Target IP address: 172.16.40.254

Figura 6: Trama Ethernet recetora do tipo ARP

No.	Time	Source	Destination	Protocol	Length
21	25.567888112	172.16.40.1	172.16.40.254	ICMP	98

> Frame 21: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
 > Ethernet II, Src: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4), Dst: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)
 > Destination: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)
 > Source: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)
 Type: IPv4 (0x0800)
 > Internet Protocol Version 4, Src: 172.16.40.1, Dst: 172.16.40.254
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 84
 Identification: 0xb523 (46371)
 > Flags: 0x40, Don't fragment
 Fragment Offset: 0
 Time to Live: 64
 Protocol: ICMP (1)
 Header Checksum: 0xdc65 [validation disabled]
 [Header checksum status: Unverified]
 Source Address: 172.16.40.1
 Destination Address: 172.16.40.254
 > Internet Control Message Protocol

Figura 7: Trama Ethernet recetora do tipo IPv4, sendo o protocolo usado ICMP

```

v Frame 21: 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 15:43:46.944139471 Hora padrão de GMT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1606232626.944139471 seconds
    [Time delta from previous captured frame: 0.000018438 seconds]
    [Time delta from previous displayed frame: 0.000018438 seconds]
    [Time since reference or first frame: 25.567888112 seconds]
    Frame Number: 21
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)

```

Figura 8: Comprimento da trama

No.	Time	Source	Destination	Protocol	Length	Info
2	0.729665369	Cisco_d4:1c:03	Cisco_d4:1c:03	<u>LOOP</u>	60	Reply

```

> Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface etl
> Ethernet II, Src: Cisco_d4:1c:03 (30:37:a6:d4:1c:03), Dst: Cisco_d4:1c:03 (30:37:a
v Configuration Test Protocol (loopback)
  skipCount: 0
  Relevant function: Reply (1)
  Function: Reply (1)
  Receipt number: 0
> Data (40 bytes)

```

Figura 9: Interface Loopback

Experiência 2

No.	Time	Source	Destination	Protocol	Length	Info
3	3.303593946	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=1/256, ttl=64 (no response found!)
5	4.306285240	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=2/512, ttl=64 (no response found!)
6	5.330287138	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=3/768, ttl=64 (no response found!)
8	6.354283239	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=4/1024, ttl=64 (no response found!)
9	7.378279898	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=5/1280, ttl=64 (no response found!)
12	8.402280050	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=6/1536, ttl=64 (no response found!)
13	9.430284173	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=7/1792, ttl=64 (no response found!)
15	10.450287967	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=8/2048, ttl=64 (no response found!)
16	11.474289167	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=9/2304, ttl=64 (no response found!)
18	12.498283661	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=10/2560, ttl=64 (no response found!)

Figura 10: Ping Broadcast do tux43 registrado no tux43

No.	Time	Source	Destination	Protocol	Length	Info
14	15.332330177	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=1/256, ttl=64 (no response found!)
16	16.334949500	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=2/512, ttl=64 (no response found!)
17	17.358876637	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=3/768, ttl=64 (no response found!)
19	18.382797418	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=4/1024, ttl=64 (no response found!)
20	19.406722808	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=5/1280, ttl=64 (no response found!)
23	20.430638071	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=6/1536, ttl=64 (no response found!)
24	21.458580467	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=7/1792, ttl=64 (no response found!)
26	22.478501842	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=8/2048, ttl=64 (no response found!)
27	23.502430306	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=9/2304, ttl=64 (no response found!)
29	24.526356255	172.16.40.1	172.16.40.255	ICMP	98	Echo (ping) request id=0x0b31, seq=10/2560, ttl=64 (no response found!)

Figura 11: Ping Broadcast do tux43 registrado no tux44

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
2	2.012815314	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
3	4.009890883	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
4	4.186178420	Cisco_d4:1c:05	Cisco_d4:1c:05	LOOP	60	Reply
5	6.014851480	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
6	6.182942488	Cisco_d4:1c:05	CDP/VTP/DTP/PAG...	CDP	432	Device ID: tux-sw4 Port ID: FastEthernet0/3
7	8.024950169	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
8	10.024783221	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
9	12.034884913	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
10	14.034634642	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
11	14.185856330	Cisco_d4:1c:05	Cisco_d4:1c:05	LOOP	60	Reply
12	16.039679470	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
13	18.044539214	Cisco_d4:1c:05	Spanning-tree-(...	STP	60	Conf. Root = 32768/41/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005

Figura 12: Ping Broadcast to tux43 não registrado no tux42

No.	Time	Source	Destination	Protocol	Length	Info
246	393.47748...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=1/256, ttl=64 (no response found!)
247	394.48222...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=2/512, ttl=64 (no response found!)
249	395.50623...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=3/768, ttl=64 (no response found!)
250	396.53023...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=4/1024, ttl=64 (no response found!)
252	397.55423...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=5/1280, ttl=64 (no response found!)
253	398.57822...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=6/1536, ttl=64 (no response found!)
255	399.60622...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=7/1792, ttl=64 (no response found!)
256	400.62622...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=8/2048, ttl=64 (no response found!)
258	401.65022...	172.16.41.1	172.16.41.255	ICMP	98	Echo (ping) request id=0x30a1, seq=9/2304, ttl=64 (no response found!)

Figura 13: Ping Broadcast do tux42 registrado no tux42

No.	Time	Source	Destination	Protocol	Length	Info
165	262.69120...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
166	264.69599...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
167	266.70595...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
168	268.70582...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
169	270.71065...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
170	271.50766...	Cisco_d4:1c:03	Cisco_d4:1c:03	LOOP	60	Reply
171	272.72069...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
172	274.72046...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
173	276.72537...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
174	278.73533...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
175	280.73512...	Cisco_d4:1c:03	Spanning-tree-(for-bridges)_00	STP	60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8003
176	281.50721...	Cisco_d4:1c:03	Cisco_d4:1c:03	LOOP	60	Reply

Figura 14: Ping Broadcast do tux42 não registrado no tux43

No.	Time	Source	Destination	Protocol	Length	Info
82	128.32254...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
83	130.33249...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
84	131.40158...	Cisco_d4:1c:05	Cisco_d4:1c:05	LOOP	60	Reply
85	132.33232...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
86	134.33713...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
87	136.34738...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
88	138.34693...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
89	140.35185...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
90	141.40898...	Cisco_d4:1c:05	Cisco_d4:1c:05	LOOP	60	Reply
91	142.36488...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005
92	144.38149...	Cisco_d4:1c:05	Spanning-tree-(for-bridges)_00 STP		60	Conf. Root = 32768/40/30:37:a6:d4:1c:00 Cost = 0 Port = 0x8005

Figura 15: Ping Broadcast do tux42 não registrado no tux44

Experiência 3

No.	Time	Source	Destination	Protocol	Length	Info
37	44.303591534	<u>172.16.40.1</u>	<u>172.16.40.254</u>	ICMP	98	Echo (ping) request id=0x1aa8, seq=5/1280, ttl=64 (reply in 38)
38	44.303719273	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1aa8, seq=5/1280, ttl=64 (request in 37)
39	45.231569631	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	42	Who has <u>172.16.40.254</u> ? Tell <u>172.16.40.1</u>
40	45.231687871	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	60	<u>172.16.40.254</u> is at <u>00:21:5a:5a:7b:ea</u>
41	45.327594750	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x1aa8, seq=6/1536, ttl=64 (reply in 42)
42	45.327719206	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1aa8, seq=6/1536, ttl=64 (request in 41)
43	45.398761131	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	60	Who has <u>172.16.40.1</u> ? Tell <u>172.16.40.254</u>
44	45.398766928	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	42	<u>172.16.40.1</u> is at <u>00:21:5a:61:2f:d4</u>
143	165.391608027	<u>172.16.40.1</u>	<u>172.16.41.253</u>	ICMP	98	Echo (ping) request id=0x1af5, seq=5/1280, ttl=64 (reply in 144)
144	165.391744286	<u>172.16.41.253</u>	<u>172.16.40.1</u>	ICMP	98	Echo (ping) reply id=0x1af5, seq=5/1280, ttl=64 (request in 143)
145	166.319568803	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	42	Who has 172.16.40.254? Tell 172.16.40.1
146	166.319687252	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	60	172.16.40.254 is at 00:21:5a:5a:7b:ea
148	166.415609706	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x1af5, seq=6/1536, ttl=64 (reply in 149)
149	166.415735349	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1af5, seq=6/1536, ttl=64 (request in 148)
150	166.487694981	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
151	166.487716980	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	42	172.16.40.1 is at 00:21:5a:61:2f:d4
186	194.223604614	<u>172.16.40.1</u>	<u>172.16.41.1</u>	ICMP	98	Echo (ping) request id=0x1b0a, seq=5/1280, ttl=64 (reply in 187)
187	194.223845425	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1b0a, seq=5/1280, ttl=63 (request in 186)
189	195.159916378	HewlettP_5a:7b:ea	HewlettP_61:2f:d4	ARP	60	Who has 172.16.40.1? Tell 172.16.40.254
190	195.159937331	HewlettP_61:2f:d4	HewlettP_5a:7b:ea	ARP	42	172.16.40.1 is at 00:21:5a:61:2f:d4
191	195.247601614	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x1b0a, seq=6/1536, ttl=64 (reply in 192)
192	195.247842704	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1b0a, seq=6/1536, ttl=63 (request in 191)

Figura 16: Ping para todas as interfaces a partir do tux43.eth0

No.	Time	Source	Destination	Protocol	Length	Info
37	44.303591534	<u>172.16.40.1</u>	<u>172.16.40.254</u>	ICMP	98	Echo (ping) request id=0x1aa8, seq=5/1280, ttl=64 (reply in 38)
> Frame 37: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0 > Ethernet II, Src: HewlettP_61:2f:d4 (<u>00:21:5a:61:2f:d4</u>), Dst: HewlettP_5a:7b:ea (<u>00:21:5a:5a:7b:ea</u>) > Internet Protocol Version 4, Src: <u>172.16.40.1</u> , Dst: <u>172.16.40.254</u> > Internet Control Message Protocol						

Figura 17: Ping to tux43.eth0 para o tux44.eth0

Experiência 4

No.	Time	Source	Destination	Protocol	Length	Info
9	2.033503702	172.16.40.1	172.16.40.254	ICMP	98	Echo (ping) request id=0x0d67, seq=3/768, ttl=64 (reply in 10)
10	2.033632978	172.16.40.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0d67, seq=3/768, ttl=64 (request in 9)
38	14.336006066	172.16.40.1	172.16.41.253	ICMP	98	Echo (ping) request id=0x0d72, seq=1/256, ttl=64 (reply in 39)
39	14.336160346	172.16.41.253	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0d72, seq=1/256, ttl=64 (request in 38)
87	36.049480595	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) request id=0x0d80, seq=2/512, ttl=64 (reply in 88)
88	36.049720150	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0d80, seq=2/512, ttl=63 (request in 87)
132	54.801495080	172.16.40.1	172.16.41.254	ICMP	98	Echo (ping) request id=0x0d8a, seq=7/1792, ttl=64 (reply in 133)
133	54.801955264	172.16.41.254	172.16.40.1	ICMP	98	Echo (ping) reply id=0x0d8a, seq=7/1792, ttl=254 (request in 132)

Figura 18: Ping do tux43.eth0 a todas as interfaces dos tuxs e ao router

No.	Time	Source	Destination	Protocol	Length	Info
5	3.088844258	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x0e08, seq=1/256, ttl=64 (reply in 7)
6	3.089190532	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
7	3.089430855	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x0e08, seq=1/256, ttl=63 (request in 5)
9	4.107387259	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x0e08, seq=2/512, ttl=64 (reply in 11)
10	4.107704269	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
11	4.107937888	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x0e08, seq=2/512, ttl=63 (request in 9)
12	5.131381861	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x0e08, seq=3/768, ttl=64 (reply in 14)
13	5.131691327	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
14	5.131921105	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x0e08, seq=3/768, ttl=63 (request in 12)
16	6.155385681	172.16.41.1	172.16.40.1	ICMP	98	Echo (ping) request id=0x0e08, seq=4/1024, ttl=64 (reply in 18)
17	6.155704646	172.16.41.254	172.16.41.1	ICMP	70	Redirect (Redirect for host)
18	6.155917243	172.16.40.1	172.16.41.1	ICMP	98	Echo (ping) reply id=0x0e08, seq=4/1024, ttl=63 (request in 16)

Figura 19: Ping to tux42.eth0, sendo o router quem redireciona os pacotes para o tux43.eth0

Configuração do Router Cisco com NAT

◆ Cisco NAT

http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.shtml

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

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

ip nat pool ovrlld 172.16.1.y9 172.16.1.y9 prefix 24
ip nat inside source list 1 pool ovrlld 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
```

Figura 20: Slide 46 do guião do 2º trabalho prático

Experiência 5

No.	Time	Source	Destination	Protocol	Length	Info
3	1.041927422	172.16.40.1	172.16.1.1	DNS	69	Standard query 0xf66a A ftp.up.pt
4	1.041938666	172.16.40.1	172.16.1.1	DNS	69	Standard query 0x5c75 AAAA ftp.up.pt
5	1.043645451	172.16.1.1	172.16.40.1	DNS	355	Standard query response 0xf66a A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15 NS ns4.up.pt NS r
6	1.043697830	172.16.1.1	172.16.40.1	DNS	367	Standard query response 0x5c75 AAAA ftp.up.pt CNAME mirrors.up.pt AAAA 2001:690:2200:1200::15 NS
7	1.044061200	172.16.40.1	193.137.29.15	ICMP	98	Echo (ping) request id=0x1443, seq=1/256, ttl=64 (reply in 8)
8	1.046717860	193.137.29.15	172.16.40.1	ICMP	98	Echo (ping) reply id=0x1443, seq=1/256, ttl=57 (request in 7)
9	1.046825412	172.16.40.1	172.16.1.1	DNS	86	Standard query 0x9f6e PTR 15.29.137.193.in-addr.arpa
10	1.048022026	172.16.1.1	172.16.40.1	DNS	387	Standard query response 0x9f6e PTR 15.29.137.193.in-addr.arpa PTR mirrors.up.pt NS ns3.up.pt NS

Figura 21: Troca de pacotes DNS

Experiência 6

No.	Time	Source	Destination	Protocol	Length	Info
42	3.772750057	172.16.40.1	192.168.109.136	TCP	66	48740 → 42376 [ACK] Seq=1 Ack=11585 Win=52480 Len=0 TSval=1907394662 TSecr=1977983724
43	3.772866831	192.168.109.136	172.16.40.1	FTP-DATA	15..	FTP Data: 1448 bytes (PASV) (RETR files/crab.mp4)

> Frame 42: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface eth0, id 0

> Ethernet II, Src: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4), Dst: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea)

> Internet Protocol Version 4, Src: 172.16.40.1, Dst: 192.168.109.136

> Transmission Control Protocol, Src Port: 48740, Dst Port: 42376, Seq: 1, Ack: 11585, Len: 0

Source Port: 48740

Destination Port: 42376

[Stream index: 1]

[TCP Segment Len: 0]

Sequence Number: 1 (relative sequence number)

Sequence Number (raw): 1320479832

[Next Sequence Number: 1 (relative sequence number)]

Acknowledgment Number: 11585 (relative ack number)

Acknowledgment number (raw): 302682676

1000 = Header Length: 32 bytes (8)

> Flags: 0x010 (ACK)

Window: 410

[Calculated window size: 52480]

[Window size scaling factor: 128]

Checksum: 0x0269 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

> [SEQ/ACK analysis]

> [Timestamps]

Figura 22: Campos do TCP

No.	Time	Source	Destination	Protocol	Length	Info
42	3.772750057	172.16.40.1	192.168.109.136	TCP	66	48740 → 42376 [ACK] Seq=1 Ack=11585 Win=52480 Len=0 TSval=1907394662 TSecr=1977983724
43	3.772866831	192.168.109.136	172.16.40.1	FTP-DATA	15..	FTP Data: 1448 bytes (PASV) (RETR files/crab.mp4)

> Frame 43: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface eth0, id 0

> Ethernet II, Src: HewlettP_5a:7b:ea (00:21:5a:5a:7b:ea), Dst: HewlettP_61:2f:d4 (00:21:5a:61:2f:d4)

> Internet Protocol Version 4, Src: 192.168.109.136, Dst: 172.16.40.1

> Transmission Control Protocol, Src Port: 42376, Dst Port: 48740, Seq: 11585, Ack: 1, Len: 1448

Source Port: 42376

Destination Port: 48740

[Stream index: 1]

[TCP Segment Len: 1448]

Sequence Number: 11585 (relative sequence number)

Sequence Number (raw): 302682676

[Next Sequence Number: 13033 (relative sequence number)]

Acknowledgment Number: 1 (relative ack number)

Acknowledgment number (raw): 1320479832

1000 = Header Length: 32 bytes (8)

> Flags: 0x010 (ACK)

Window: 510

[Calculated window size: 65280]

[Window size scaling factor: 128]

Checksum: 0x939f [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

> [SEQ/ACK analysis]

> [Timestamps]

TCP payload (1448 bytes)

FTP Data (1448 bytes data)

Figura 23: Campos do TCP

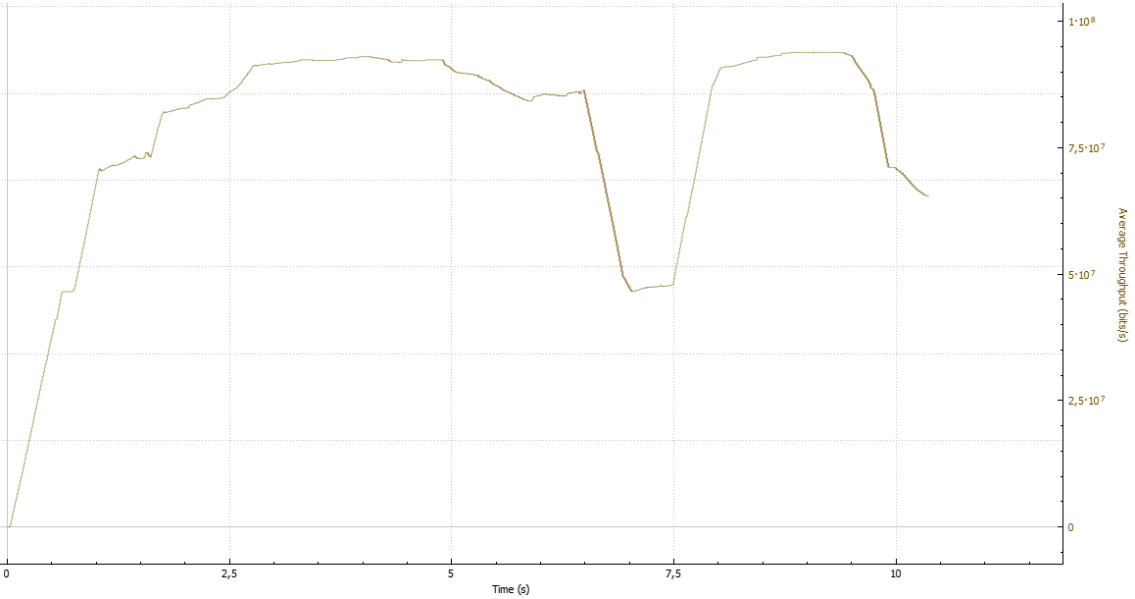


Gráfico 1: Download no tux43.eth0 com um download no tux42.eth0