Faculdade de Engenharia da Universidade do Porto

Redes de computadores

2º Trabalho Laboratorial

Redes de Computadores
Turma 5 - Grupo 5

- Fábio Araújo de Sá (up202007658@fe.up.pt)
- Lourenço Alexandre Correia Gonçalves (up202004816@fe.up.pt)

Porto, 12 de Dezembro de 2022

Sumário

Este trabalho realizado no âmbito da Unidade Curricular de Redes de Computadores visa a implementação de um programa de download usando o FTP e a configuração e utilização de uma rede de computadores.

Através deste projeto conseguimos fazer uso da matéria lecionada nas aulas teóricas para implementar o programa com protocolo em questão e configurar a rede.

Introdução

O objetivo do trabalho foi o desenvolvimento e teste de um programa de download usando o FTP e configuração de uma rede de computadores, de acordo com as especificações presentes no guião, para transferir um ficheiro da internet utilizando a rede configurada. O presente relatório está dividido nas seguintes seções:

- **Introdução:** Breve descrição do projeto e os seus objetivos
- ***Downloader*:** Aplicação de download com FTP
    - Arquitetura da aplicação
    - Testes e resultados
- **Configuração e análise de redes:**
    - Configurar uma rede IP;
    - Implementar duas *bridges* num *switch*;
    - Configurar um router em Linux;
    - Configurar um router comercial com NAT;
    - DNS;
    - Ligações TCP;
- **Conclusões:** Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

Downloader

Arquitetura da aplicação

A aplicação desenvolvida realiza o *download* de um ficheiro através do protocolo FTP. Para isso foram consultadas a norma RFC959, funcionamento e arquitetura do protocolo FTP e a norma RFC1738, tratamento das partes constituintes dos endereços URL.

Inicialmente é processado o URL dado como argumento, utilizando expressões regulares, de modo a obter a partir deste uma estrutura de dados com informações necessárias para o estabelecimento da ligação: *host*, nome do servidor onde será criada a comunicação; *resource*, caminho até ao ficheiro pretendido; *file*, nome e extensão do ficheiro a transferir, *user* e *password*, para registo no servidor que podem ser default caso não especificados no input; ip, obtido através do host com o sistema de tradução DNS. Cada campo é ainda submetido a diferentes camadas de validação, desde a correta escrita do URL até à existência do servidor e ficheiro selecionados, garantindo robustez ao nível do processamento dos dados iniciais.

Todas as funções utilizadas para comunicação com o servidor recebem como argumentos o socket a utilizar, um buffer para recepção da mensagem e retornam um inteiro de três dígitos que é o código de resposta do servidor. A receção das mensagens é garantida por uma máquina de estados, implementada na função *readResponse*. Em cada etapa, caso o código recebido do servidor não seja da gama 2XX nem 3XX, algo que acontece por exemplo nas etapas intermédias do *login*, a operação é abortada com uma mensagem de erro expressiva.

Inicialmente é criado um primeiro socket ligado ao IP do *host* através da função *createSocket.* Após o *handshake* é garantida a autenticação no servidor com a função *authConn*, através dos campos *user* e *password*. Nesse momento enviamos o código "*passv*" na função *passiveMode*, para que o servidor entrasse no modo passivo. Assim, obteve-se o IP e a porta para uma segunda conexão FTP a recepção do ficheiro pretendido através da criação de um segundo *socket*. Na fase de download propriamente dito, o ficheiro a transferir é selecionado pela função *requestResource* e transferido a partir de *getResource*.

Finalmente, após a transferência com sucesso, as ligações e os dois *sockets* são encerrados com comandos "quit" durante a execução da função *closeConnection*.

Testes e resultados

De modo a testar o correto funcionamento da aplicação de *download*, foram realizados vários testes com transferências de ficheiros de diferentes tamanhos, desde alguns com apenas umas dezenas de bytes até outros com vários *gigabytes*. Foram também testadas as várias camadas de validação de input, através URLs errados em diversos campos, *resources* não existentes e modo anónimo e não anónimo. Verificamos que em todas as situações apresentadas o programa mostrou-se robusto e eficiente.

Na seguinte imagem observa-se o resultado de uma transferência bem sucedida:

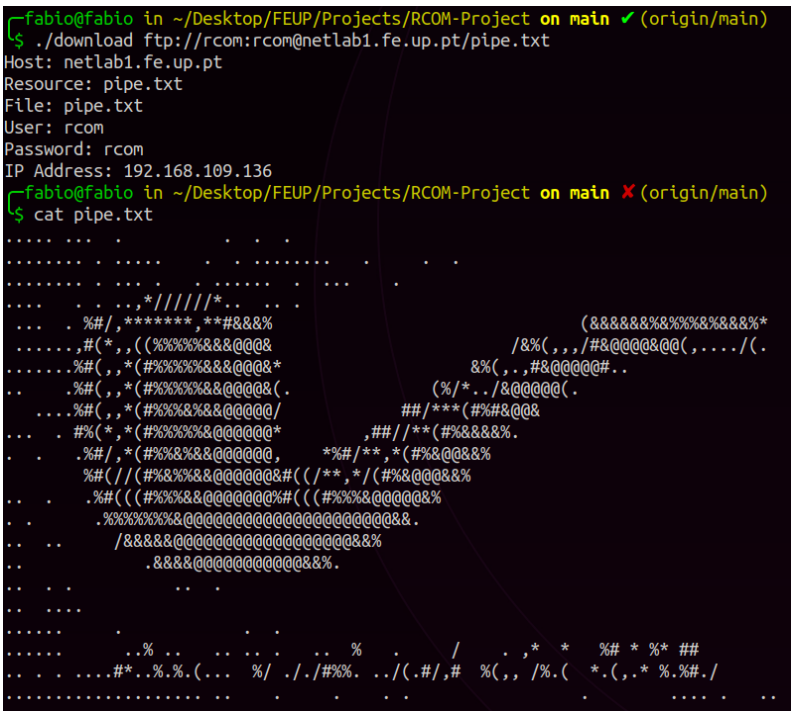


Imagem 1 - Exemplo do funcionamento do *downloader*

Configuração e análise de redes

Experiência 1 - Configurar uma rede IP

O objetivo desta experiência foi a configuração de dois endereços IP de dois computadores diferentes, o Tux53 e Tux54, ligados a um switch. Posteriormente, pretendeu-se analisar o funcionamento do protocolo ARP e verificar o comportamento da ligação quando se eliminam entradas dos IPs configurados nas tabelas ARP.

Para isso, ligou-se o E0 de cada Tux ao switch e utilizou-se o comando *ifconfig* para configurar o IP de cada um. Os endereços MAC e IP de cada computador puderam ser observados nas tabelas ARP criadas após verificar a conexão entre os dois computadores com o comando *ping*. Para o Tux53 foi configurado o IP *172.16.50.1/24,* que correspondia ao MAC *00:21:5a:61:2d:72*, enquanto para o Tux54 o IP foi *172.16.50.254/24* e MAC 00:21:5a:c3:78:70. Analisando os pacotes trocados após a utilização deste comando, verifica-se a presença de pacotes ARP e ICMP.

O protocolo ARP (*Address Resolution Protocol*) relaciona um IP de uma máquina ao respetivo endereço MAC, ou seja, faz a ligação entre a *network layer* e a *link layer*, sendo assim fundamental para o estabelecimento de qualquer ligação. São enviados em *broadcast* 2 endereços IP num pacote: o da máquina de destino e o da máquina de origem. Como resposta, o destinatário envia um pacote do mesmo protocolo que contém o seu endereço MAC. Esta associação é guardada numa das entradas da tabela ARP de ambos os computadores. Verifica-se que quando são eliminadas as entradas das tabelas ARP, existe uma troca de pacotes ARP no início da ligação para repor as associações entre os endereços IP e MAC que foram eliminados. Esta troca de mensagens foi evidenciada nas primeiras duas linhas da captura.

O protocolo ICMP (*Internet Control Message Protocol*) é utilizado para trocar mensagens de controlo, indicando sucesso ou erros durante a comunicação com outro endereço IP. No caso concreto da experiência, foram verificados a transportar mensagens de *reply* e *request.*

O log da captura dos pacotes durante o comando ping está disponível no anexo 3.1.1 e os comandos utilizados no anexo 2.1.

Experiência 2 - Implementar duas *bridges* num switch

O objetivo desta experiência foi a criação de duas LANs (*Local Area Network*), uma com os Tuxs 53 e 54, e outra com apenas o 52, utilizando 2 *bridges* no *switch*.

Inicialmente, foi necessário repetir a configuração de rede da experiência 1 e configurar o endereço IP *172.16.51.1/24* para o Tux52. Para criar as *bridges* foi necessário configurar o *switch*, o que foi feito ligando a consola deste à porta série do Tux52. Na configuração, foram criadas as *bridges* 50 e 51 (com o comando */interface bridge add*), foram removidas as portas às quais as interfaces conectadas a cada um dos Tuxes estavam ligadas por defeito (com o comando */interface bridge port remove*), e foram adicionadas essas interfaces dos Tuxes 53 e 54 à *bridge50*, e a do Tux52 à *bridge51* (com o comando */interface bridge port add*).

Nesta experiência existiam dois domínios de *broadcast*, que correspondiam às duas *bridges*/LANs implementadas. Verificou-se que fazendo *broadcast* a partir do Tux53, o Tux54 era alcançado mas o Tux52 não, visto que só o primeiro se encontra na mesma LAN

que o Tux53. Fazendo o mesmo no Tux52, confirma-se que este não alcança nenhum outro computador, pois encontra-se numa LAN isolada. Isto concluiu-se a partir dos pacotes ICMP capturados nos dois computadores: na primeira situação existiam estados alternados *reply/request* enquanto que na segunda apenas tinham estado *reply* sem resposta.

Os logs da captura dos pacotes durante o comando ping nos Tuxes 53 e 52 estão disponíveis no anexo 3.2.1 e anexo 3.2.2, respetivamente, e os comandos utilizados para a configuração no anexo 2.2.

Experiência 3 - Configurar um router em Linux

Esta experiência foi iniciada com a rede resultante da experiência 2, onde o Tux53 e Tux54 foram ligados a uma *bridge*, e o Tux52 a outra. O objetivo foi transformar o Tux54 num *router* para que o Tux53 e Tux52, que se encontram em LANs diferentes, possam comunicar entre si.

Para isto, conectou-se o E1 do Tux54 ao *switch* e configou-se o IP *172.16.51.253/24*. Depois, removeram-se as portas às quais a interface do *switch* conectada ao Tux54 estava ligada por defeito e adicionou-se essa interface à *bridge51*. Posteriormente desativou-se o *icmp_echo_ignore_broadcasts* e ativou-se o *IP forwarding*. Finalmente, usando o comando *route add*, criaram-se rotas nos Tuxes 52 e 53, considerando como *gateway* o IP do Tux54 acessível por cada um deles através das suas LANs, ou seja, *172.16.51.253/24 e* 172.16.50.254/24, respetivamente.

As rotas configuradas permitiram utilizar o Tux54 como um *router* que faz a ligação entre as duas LANs. Assim, quando utilizamos o comando *ping* entre o Tux53 e Tux52, todos os pacotes ARP e ICMP resultantes chegaram ao destino. Os pacotes ARP e ICMP, computados no Tux54, continham o endereço IP da máquina de destino mas o endereço MAC do Tux54, uma vez que este último, tal como um *router*, trata do redirecionamento da informação entre as duas redes criadas. As tabelas de encaminhamento geradas através da criação das rotas garante que por cada IP de destino, existe outro endereço IP (*gateway*) para onde a máquina de origem deve reencaminhar a informação.

Os logs da captura dos pacotes durante o comando ping no Tux 53 está disponível no anexo 3.3.1 e os comandos utilizados para a configuração no anexo 2.3.

Experiência 4 - Configurar um router comercial com NAT

Nesta experiência usou-se como base a rede resultante da experiência 3, constituída por duas LANs, uma com o Tux53 e outra com o Tux52, em que o Tux54 faz de *router* de modo a ligar as 2 subredes. O objetivo foi configurar e adicionar à bridge51 um router comercial com NAT implementado, de modo a permitir acesso à internet.

Inicialmente ligou-se uma das entradas do *router* comercial à régua e outra ao *switch*, e adicionou-se a interface à *bridge*51 da mesma forma que nas experiências anteriores. Posteriormente, trocou-se o cabo que ligava o Tux52 à consola do *switch* para ligar à consola do *router* comercial de modo a poder configurar o mesmo. Na consola do *router* comercial, configuraram-se os IPs correspondentes a cada interface (com o comando */ip address add*). Por fim foram criadas rotas *default* em cada um dos Tuxes, ligando-os ao router, e uma rota que permitia ao router ligar-se a cada uma das LANs através do Tux54.

Na primeira situação, sem a ligação do Tux52 ao Tux54 e sem ICMP *redirects*, ao fazer *ping* do Tux52 para o Tux53 os pacotes de dados foram reencaminhados através do *router* pela *default route* até ao endereço IP de destino, tal como comprovam as capturas 3.4.1 a 3.4.3. Já na segunda situação, ativando outra vez as rotas e redirecionamentos inibidos, os pacotes não chegaram a passar no *router* pois a ligação mais direta da rede estava disponível, ou seja, foi utilizado simplesmente o Tux54 como intermediário entre as duas LANs. Assim, conclui-se que os pacotes ICMP são responsáveis pelo melhor aproveitamento das ligações existentes na rede, existindo redirects quando necessário, como foi o caso observado na captura 3.4.4.

O NAT (*Network Address Translation*) é um método de mapeamento utilizado para traduzir endereços de uma rede local para um único endereço público, e viceversa. Assim, quando um pacote é enviado para uma rede externa, é enviado com o endereço público como origem. Quando o computador de destino responde, envia a resposta para esse endereço público, que é depois traduzido de volta para o endereço local de destino que enviou o *request*. Deste modo, é possível reduzir o número de endereços públicos utilizados e contornar a escassez de IPs únicos derivados do uso extensivo do IPv4.

Com o NAT ativado verifica-se que existe ligação à internet. No entanto, quando este é desativado, não há forma de o emissor receber a resposta aos pedidos solicitados, uma vez que o router não tem capacidade de traduzir o endereço de destino recebido num endereço da rede interna.

Os comandos utilizados para a configuração estão disponíveis no anexo 2.4, e os logs da captura de pacotes nos seguintes: anexo 3.4.1, anexo 3.4.2, anexo 3.4.3, anexo 3.4.4.

Experiência 5 - DNS

Nesta experiência é utilizada a rede configurada ao longo das experiências anteriores. O objetivo desta experiência é configurar o DNS de modo a, dentro da rede criada, poder aceder a *websites* da internet através do seu nome de domínio. O correto funcionamento foi verificado fazendo ping do *google.com*.

Para realizar esta experiência bastou mudar o conteúdo do ficheiro */etc/resolv.conf* em cada Tux para a seguinte linha: *'nameserver 172.16.1.1*. Esse endereço IP é o do router presente no laboratório e tem acesso à rede externa.

Observou-se que a troca de pacotes DNS (*Domain Name System*) na rede acontece antes de qualquer outro protocolo, já que este realiza a tradução de um nome de domínio para o seu endereço IP que será utilizado por todos os outros protocolos.

Os comandos utilizados para a configuração estão disponíveis no anexo 2.5, e os logs da captura de pacotes no anexo 3.5.1.

Experiência 6 - Ligações TCP

Nesta experiência, além de utilizarmos toda a rede configurada ao longo dos passos anteriores, usamos a aplicação de *download* que desenvolvemos para podermos avaliar o envio e receção de pacotes, a sua constituição interna e os mecanismos de controlo de fluxo e congestionamento caraterísticos das ligações TCP.

Numa primeira fase compilamos o *downloader* no Tux53 e fizemos *download* de um ficheiro de alguns *megabytes*. Observou-se a transferência inicial pacotes DNS para

tradução do nome do servidor no endereço IP, seguida de pacotes FTP *SYN-ACK request/response* para realizar o *handshake* entre o cliente e o servidor e por fim pacotes FTP *Data* com números de sequência incrementais para receção de tramas de informação. A ligação termina com o envio do pacote FTP *FIN-ACK*.

A aplicação FTP usada abre duas ligações TCP (*Transmission Control Protocol*): uma para envio de comandos de controlo ao servidor e outra para a receção do ficheiro selecionado. Todas as ligações TCP utilizam ARQ, são orientadas às comunicações e garantem a transferência de dados incorporando dois mecanismos de controlo: o controlo de fluxo e o controlo de congestionamento.

O mecanismo ARQ (*Automatic Repeat Request*) é um mecanismo também utilizado nas ligações TCP para controlo de erros, usando mensagens ACK (*acknowledge*) para indicar a correta receção do pacote e *timeouts* para averiguar o tempo de receção do mesmo. Sempre que um *timeout* é ultrapassado, considera-se que o pacote foi perdido e posteriormente é retransmitido pela rede.

O mecanismo de controlo de fluxo permite ao receptor controlar o próprio fluxo de receção de pacotes. Os buffers do lado do recetor e emissor são finitos, pelo que não podem receber mais informação do que a sua capacidade caso contrário há perda de pacotes na rede, desencadeando algumas retransmissões. Assim, em cada pacote, é juntamente enviada uma *window size* em bytes, para que o emissor avalie a quantidade de bytes que ainda são possíveis enviar sem comprometer a viabilidade dos mesmos. Foi visível nos pacotes transferidos ao longo desta experiência que o campo *window size* ficava com valores inferiores quando existiam picos de tráfego na rede.

O controlo de congestionamento é efetuado pelos emissores de dados e consiste na utilização do método *Selective Repeat,* ou seja, envio de pacotes pela rede sem esperar pelos respetivos ACK. Uma rede diz-se congestionada a partir do momento em que se perdem pacotes. Para saber o limite da rede na altura da transferência, o emissor transfere vários pacotes de uma vez seguindo uma de duas metodologias até que perceba perda de pacotes na rede: *Additive Increase*, na transferência seguinte envia mais um pacote do que na transferência anterior, ou *Slow Start*, semelhante à anterior mas com incrementos exponenciais na base dois. A perda de um pacote pode ser detetada através de *timeout* ou de três ACK seguidos, causando uma drástica redução do número de pacotes transferidos na seguinte transmissão para valores próximos da metade do valor que despoletou a perda.

No gráfico obtido ao longo desta primeira fase, onde se analisou a velocidade dos pacotes encaminhados pela rede em função do tempo, ficou visível a influência deste mecanismo da ligação TCP. De facto, seguido de períodos de um aumento significativo de pacotes transferidos, que correspondem a *Additive Increase* ou *Slow Start* para *declives* menos ou mais acentuados respetivamente, há um decréscimo desse valor para próximo de metade do valor anterior.

Numa segunda fase, começou-se um *download* de um ficheiro com um tamanho de alguns *megabytes* no Tux53 e, logo a seguir, outro *download* no Tux52. É possível observar no gráfico da velocidade de transferência que esta diminui para cerca de metade, no Tux53, quando o segundo *download* é iniciado. Isto deve-se à ação do controlo de congestionamento por parte dos dois computadores na rede, que converge rapidamente para valores estáveis ao longo da mútua transferência.

Os logs da captura de pacotes nos anexos seguintes: anexo 3.6.1, anexo 3.6.2, anexo 3.6.3, anexo 3.6.4.

Conclusões

Com este projeto, e o alcance de todos os seus objetivos, foi possível consolidar o nosso conhecimento acerca dos protocolos envolvidos na transferência de dados através de redes de computadores, ou seja, como estes se propagam dentro da *network layer* e a sua passagem para a *link layer*.

Referências

1. Beej's Guide to Network Programming Using Internet Sockets
2. Mikrotik RouterOS Wiki
3. RFCs relevantes fornecidos no moodle.

Anexos

Anexo 1 - Código da aplicação de download

1.1 - download.h

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

#define MAX_LENGTH  500
#define FTP_PORT    21

/* Server responses */
#define SV_READY4AUTH           220
#define SV_READY4PASS           331
#define SV_LOGINSUCCESS         230
#define SV_PASSIVE              227
#define SV_READY4TRANSFER       150
#define SV_TRANSFER_COMPLETE    226
#define SV_GOODBYE              221

/* Parser regular expressions */
#define AT              "@"
```

```
#define BAR             "/"
#define HOST_REGEX      "%*[^/]//%[^/]"
#define HOST_AT_REGEX   "%*[^/]//%*[^@]@%[^/]"
#define RESOURCE_REGEX  "%*[^/]//%*[^/]/%s"
#define USER_REGEX      "%*[^/]//%[^:/]"
#define PASS_REGEX      "%*[^/]//%*[^:]:%[^@\n$]"
#define RESPCODE_REGEX  "%d"
#define PASSIVE_REGEX   "%*[^(](%d,%d,%d,%d,%d,%d)%*[^\n$)]"

/* Default login for case 'ftp://<host>/<url-path>' */
#define DEFAULT_USER        "anonymous"
#define DEFAULT_PASSWORD    "password"

/* Parser output */
struct URL {
    char host[MAX_LENGTH];      // 'ftp.up.pt'
    char resource[MAX_LENGTH];  //
'parrot/misc/canary/warrant-canary-0.txt'
    char file[MAX_LENGTH];      // 'warrant-canary-0.txt'
    char user[MAX_LENGTH];      // 'username'
    char password[MAX_LENGTH];  // 'password'
    char ip[MAX_LENGTH];        // 193.137.29.15
};

/* Machine states that receives the response from the server */
typedef enum {
    START,
    SINGLE,
    MULTIPLE,
    END
} ResponseState;

/*
* Parser that transforms user input in url parameters
* @param input, a string containing the user input
* @param url, a struct that will be filled with the url parameters
* @return 0 if there is no parse error or -1 otherwise
*/
int parse(char *input, struct URL *url);

/*
* Create socket file descriptor based on given server ip and port
* @param ip, a string containing the server ip
* @param port, an integer value containing the server port
* @return socket file descriptor if there is no error or -1 otherwise
*/
```

```
int createSocket(char *ip, int port);

/*
* Authenticate connection
* @param socket, server connection file descriptor
* @param user, a string containing the username
* @param pass, a string containing the password
* @return server response code obtained by the operation
*/
int authConn(const int socket, const char *user, const char *pass);

/*
* Read server response
* @param socket, server connection file descriptor
* @param buffer, string that will be filled with server response
* @return server response code obtained by the operation
*/
int readResponse(const int socket, char *buffer);

/*
* Enter in passive mode
* @param socket, server connection file descriptor
* @param ip, string that will be filled with data connection ip
* @param port, string that will be filled with data connection port
* @return server response code obtained by the operation
*/
int passiveMode(const int socket, char* ip, int *port);

/*
* Request resource
* @param socket, server connection file descriptor
* @param resource, string that contains the desired resource
* @return server response code obtained by the operation
*/
int requestResource(const int socket, char *resource);

/*
* Get resource from server and download it in current directory
* @param socketA, server connection file descriptor
* @param socketB, server connection file descriptor
* @param filename, string that contains the desired file name
* @return server response code obtained by the operation
*/
int getResource(const int socketA, const int socketB, char *filename);

/*
```

```
* Closes the server connection and the socket itself
* @param socketA, server connection file descriptor
* @param socketB, server connection file descriptor
* @return 0 if there is no close error or -1 otherwise
*/
int closeConnection(const int socketA, const int socketB);
```

1.2 - download.c

```
#include "../include/download.h"

int parse(char *input, struct URL *url) {

    regex_t regex;
    regcomp(&regex, BAR, 0);
    if (regexec(&regex, input, 0, NULL, 0)) return -1;

    regcomp(&regex, AT, 0);
    if (regexec(&regex, input, 0, NULL, 0) != 0) {
//ftp://<host>/<url-path>

        sscanf(input, HOST_REGEX, url->host);
        strcpy(url->user, DEFAULT_USER);
        strcpy(url->password, DEFAULT_PASSWORD);

    } else { // ftp://[<user>:<password>@]<host>/<url-path>

        sscanf(input, HOST_AT_REGEX, url->host);
        sscanf(input, USER_REGEX, url->user);
        sscanf(input, PASS_REGEX, url->password);
    }

    sscanf(input, RESOURCE_REGEX, url->resource);
    strcpy(url->file, strrchr(input, '/') + 1);

    struct hostent *h;
    if (strlen(url->host) == 0) return -1;
    if ((h = gethostbyname(url->host)) == NULL) {
        printf("Invalid hostname '%s'\n", url->host);
        exit(-1);
    }
    strcpy(url->ip, inet_ntoa(*((struct in_addr *) h->h_addr)));

    return !(strlen(url->host) && strlen(url->user) &&
           strlen(url->password) && strlen(url->resource) &&
```

```
strlen(url->file));
}

int createSocket(char *ip, int port) {

    int sockfd;
    struct sockaddr_in server_addr;

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

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
    }
    if (connect(sockfd, (struct sockaddr *) &server_addr,
sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
    }

    return sockfd;
}

int authConn(const int socket, const char* user, const char* pass) {

    char userCommand[5+strlen(user)+1]; sprintf(userCommand, "user
%s\n", user);
    char passCommand[5+strlen(pass)+1]; sprintf(passCommand, "pass
%s\n", pass);
    char answer[MAX_LENGTH];

    write(socket, userCommand, strlen(userCommand));
    if (readResponse(socket, answer) != SV_READY4PASS) {
        printf("Unknown user '%s'. Abort.\n", user);
        exit(-1);
    }

    write(socket, passCommand, strlen(passCommand));
    return readResponse(socket, answer);
}

int passiveMode(const int socket, char *ip, int *port) {
```

```
    char answer[MAX_LENGTH];
    int ip1, ip2, ip3, ip4, port1, port2;
    write(socket, "pasv\n", 5);
    if (readResponse(socket, answer) != SV_PASSIVE) return -1;

    sscanf(answer, PASSIVE_REGEX, &ip1, &ip2, &ip3, &ip4, &port1,
&port2);
    *port = port1 * 256 + port2;
    sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);

    return SV_PASSIVE;
}

int readResponse(const int socket, char* buffer) {

    char byte;
    int index = 0, responseCode;
    ResponseState state = START;
    memset(buffer, 0, MAX_LENGTH);

    while (state != END) {

        read(socket, &byte, 1);
        switch (state) {
            case START:
                if (byte == ' ') state = SINGLE;
                else if (byte == '-') state = MULTIPLE;
                else if (byte == '\n') state = END;
                else buffer[index++] = byte;
                break;
            case SINGLE:
                if (byte == '\n') state = END;
                else buffer[index++] = byte;
                break;
            case MULTIPLE:
                if (byte == '\n') {
                    memset(buffer, 0, MAX_LENGTH);
                    state = START;
                    index = 0;
                }
                else buffer[index++] = byte;
                break;
            case END:
                break;
            default:
                break;
```

```
        }
    }

    sscanf(buffer, RESPCODE_REGEX, &responseCode);
    return responseCode;
}

int requestResource(const int socket, char *resource) {

    char fileCommand[5+strlen(resource)+1], answer[MAX_LENGTH];
    sprintf(fileCommand, "retr %s\n", resource);
    write(socket, fileCommand, sizeof(fileCommand));
    return readResponse(socket, answer);
}

int getResource(const int socketA, const int socketB, char *filename) {

    FILE *fd = fopen(filename, "wb");
    if (fd == NULL) {
        printf("Error opening or creating file '%s'\n", filename);
        exit(-1);
    }

    char buffer[MAX_LENGTH];
    int bytes;
    do {
        bytes = read(socketB, buffer, MAX_LENGTH);
        if (fwrite(buffer, bytes, 1, fd) < 0) return -1;
    } while (bytes);
    fclose(fd);

    return readResponse(socketA, buffer);
}

int closeConnection(const int socketA, const int socketB) {

    char answer[MAX_LENGTH];
    write(socketA, "quit\n", 5);
    if(readResponse(socketA, answer) != SV_GOODBYE) return -1;
    return close(socketA) || close(socketB);
}

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

    if (argc != 2) {
        printf("Usage: ./download
```

```
ftp://[<user>:<password>@]<host>/<url-path>\n");
        exit(-1);
    }

    struct URL url;
    memset(&url, 0, sizeof(url));
    if (parse(argv[1], &url) != 0) {
        printf("Parse error. Usage: ./download
ftp://[<user>:<password>@]<host>/<url-path>\n");
        exit(-1);
    }

    printf("Host: %s\nResource: %s\nFile: %s\nUser: %s\nPassword: %s\nIP
Address: %s\n", url.host, url.resource, url.file, url.user,
url.password, url.ip);

    char answer[MAX_LENGTH];
    int socketA = createSocket(url.ip, FTP_PORT);
    if (socketA < 0 || readResponse(socketA, answer) != SV_READY4AUTH) {
        printf("Socket to '%s' and port %d failed\n", url.ip, FTP_PORT);
        exit(-1);
    }

    if (authConn(socketA, url.user, url.password) != SV_LOGINSUCCESS) {
        printf("Authentication failed with username = '%s' and password
= '%s'.\n", url.user, url.password);
        exit(-1);
    }

    int port;
    char ip[MAX_LENGTH];
    if (passiveMode(socketA, ip, &port) != SV_PASSIVE) {
        printf("Passive mode failed\n");
        exit(-1);
    }

    int socketB = createSocket(ip, port);
    if (socketB < 0) {
        printf("Socket to '%s:%d' failed\n", ip, port);
        exit(-1);
    }

    if (requestResource(socketA, url.resource) != SV_READY4TRANSFER) {
        printf("Unknown resouce '%s' in '%s:%d'\n", url.resource, ip,
port);
        exit(-1);
```

```
    }

    if (getResource(socketA, socketB, url.file) != SV_TRANSFER_COMPLETE)
{
        printf("Error transfering file '%s' from '%s:%d'\n", url.file,
ip, port);
        exit(-1);
    }

    if (closeConnection(socketA, socketB) != 0) {
        printf("Sockets close error\n");
        exit(-1);
    }

    return 0;
}
```

Anexo 2 - Comandos de configuração

2.1 Experiência 1

Tux53:

ifconfig eth0 up
ifconfig eth0 172.16.50.1/24

Tux54:

ifconfig eth0 up
ifconfig eth0 172.16.50.254/24

Tux53:

ping 172.16.50.254 (ping do Tux54, responde)

Tux54:

ping 172.16.50.1 (ping do Tux53, responde)

Tux53:

arp -a (*?(172.16.50.254) at 00:21:5a:c3:78:70 [ether] on eth0*)
arp -d 172.16.50.254/24
arp -a (vazio)

2.2 Experiência 2

Switch console:

/system reset-configuration

Tux52:

ifconfig eth0 up
ifconfig eth0 172.16.51.1/24

Switch console:

/interface bridge add name=bridge50
/interface bridge add name=bridge51
/interface bridge port remove [find interface=ether1]
/interface bridge port remove [find interface=ether2]
/interface bridge port remove [find interface=ether3]
/interface bridge port add bridge=bridge50 interface=ether1
/interface bridge port add bridge=bridge50 interface=ether2
/interface bridge port add bridge=bridge51 interface=ether3
/interface bridge port print *(verificar que a configuração está correta)*

Tux53:

ping 172.16.50.254 (ping do Tux54, responde)
ping 172.16.51.1 (ping do Tux54, não responde)
ping -b 172.16.50.255 (*Broadcast*)

Tux52:

ping -b 172.16.51.255 (*Broadcast*)

2.3 Experiência 3

Tux54:

ifconfig eth1 up
ifconfig eth1 172.16.51.253/24

Switch console:

/interface bridge port remove [find interface=ether4]
/interface bridge port add bridge=bridge51 interface=ether4

Tux54:

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

Tux52:

route add -net 172.16.50.0/24 gw 172.16.51.253

Tux53:

route add -net 172.16.51.0/24 gw 172.16.50.254

Tux53:

ping 172.16.50.254 (responde)
ping 172.16.51.253 (responde)
ping 172.16.51.1 (responde)

Tux52:

arp -d 172.16.51.253

Tux53:

arp -d 172.16.50.254

Tux54:

arp -d 172.16.50.1
arp -d 172.16.51.1

2.4 Experiência 4

Switch console:

/interface bridge port remove [find interface=ether5]
/interface bridge port add bridge=bridge51 interface=ether5

Router console:

/system reset-configuration
/ip address add address=172.16.1.59/24 interface=ether1
/ip address add address=172.16.51.254/24 interface=ether2
/ip route add dst-address=172.16.50.0/24 gateway=172.16.51.253
/ip route add dst-address=0.0.0.0/0 gateway=172.16.1.254

Tux52:

route add default gw 172.16.51.254

Tux53:

route add default gw 172.16.50.254

Tux54:

route add default gw 172.16.51.254

Tux53:

ping 172.16.50.254 (responde)
ping 172.16.51.1 (responde)
ping 172.16.51.254 (responde)

Tux52:

echo 0 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects
route del -net 172.16.50.0 gw 172.16.51.253 netmask 255.255.255.0
ping 172.16.50.1 (responde)
traceroute -n 172.16.50.1
route add -net 172.16.50.0/24 gw 172.16.51.253
traceroute -n 172.16.50.1
echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects

Tux53:

ping 172.16.1.254 (responde)

Router console:

/ip firewall nat disable 0

Tux53:

ping 172.16.1.254 (não responde)

Router console:

/ip firewall nat enable 0

2.5 Experiência 5

Tux52, Tux53, Tux54:

ping google.com (responde depois de configurar o DNS)

Anexo 3 - Logs capturados

3.1.1 Exp1 - Ping do Tux53 ao Tux54

No.	Time	Source	Destination	Protocol	Length	Info
20	33.286137897	HewlettP_61:2d:72	Broadcast	ARP	42	Who has 172.16.50.254? Tell 172.16.50.1
21	33.286241333	HewlettP_c3:78:70	HewlettP_61:2d:72	ARP	60	172.16.50.254 is at 00:21:5a:c3:78:70
22	33.286259352	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x151a, seq=1/256, ttl=64 (reply in 23)
23	33.286348331	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x151a, seq=1/256, ttl=64 (request in 22)
24	34.036817320	Routerbo_1c:8b:bb	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8b:bb Cost = 0 Port = 0x8001
25	34.311987272	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x151a, seq=2/512, ttl=64 (reply in 26)
26	34.312108587	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x151a, seq=2/512, ttl=64 (request in 25)
27	35.335968583	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x151a, seq=3/768, ttl=64 (reply in 28)
28	35.336075931	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x151a, seq=3/768, ttl=64 (request in 27)

3.2.1 Exp2 - Ping do Tux53 ao Tux54

35	58.063433766	Routerbo_1c:8b:bc	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8b:bb Cost = 0 Port = 0x8002
36	58.638889276	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x08ef, seq=2/512, ttl=64 (reply in 37)
37	58.639020298	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x08ef, seq=2/512, ttl=64 (request in 36)
38	59.662912373	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x08ef, seq=3/768, ttl=64 (reply in 39)
39	59.663043465	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x08ef, seq=3/768, ttl=64 (request in 38)

3.2.2 Exp2 - Ping do Tux53 ao Tux52

26	44.491422548	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x0bbf, seq=1/256, ttl=64 (no response found!)
27	45.517697607	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x0bbf, seq=2/512, ttl=64 (no response found!)
28	46.040871818	Routerbo_1c:8b:be	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8b:bb Cost = 0 Port = 0x8004
29	46.541686691	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x0bbf, seq=3/768, ttl=64 (no response found!)
30	47.565734093	172.16.51.1	172.16.51.255	ICMP	98	Echo (ping) request id=0x0bbf, seq=4/1024, ttl=64 (no response found!)

3.3.1 Exp3 - Pacotes ARP e ICMP, capturados no Tux54 durante o estabelecimento da ligação entre o Tux53 e o Tux52

97	172.473995894	HewlettP_61:2d:72	Broadcast	ARP	60	Who has 172.16.50.254? Tell 172.16.50.1
98	172.474016498	HewlettP_c3:78:70	HewlettP_61:2d:72	ARP	42	172.16.50.254 is at 00:21:5a:c3:78:70
99	172.474128872	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x1532, seq=1/256, ttl=64 (reply in 100)
100	172.474386307	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x1532, seq=1/256, ttl=63 (request in 99)
101	173.500297074	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x1532, seq=2/512, ttl=64 (reply in 102)
102	173.500449049	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x1532, seq=2/512, ttl=63 (request in 101)

3.4.1 Exp4 - Pacotes ARP e ICMP, capturados no Tux53 quando se remove a rota que liga o Tux52 ao Tux54

20	19.634719911	HewlettP_61:2d:72	HewlettP_c3:78:70	ARP	42	Who has 172.16.50.254? Tell 172.16.50.1
21	19.634864485	HewlettP_c3:78:70	HewlettP_61:2d:72	ARP	60	172.16.50.254 is at 00:21:5a:c3:78:70
22	19.725026252	HewlettP_c3:78:70	HewlettP_61:2d:72	ARP	60	Who has 172.16.50.1? Tell 172.16.50.254
23	19.725035122	HewlettP_61:2d:72	HewlettP_c3:78:70	ARP	42	172.16.50.1 is at 00:21:5a:61:2d:72
24	20.021048569	Routerbo_1c:8b:bb	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8b:bb Cost = 0 Port = 0x8001
25	22.023162778	Routerbo_1c:8b:bb	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8b:bb Cost = 0 Port = 0x8001
26	22.247685048	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x588a, seq=1/256, ttl=64 (reply in 27)
27	22.247832276	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x588a, seq=1/256, ttl=64 (request in 26)
28	23.250743909	172.16.50.1	172.16.50.254	ICMP	98	Echo (ping) request id=0x588a, seq=2/512, ttl=64 (reply in 29)
29	23.250871372	172.16.50.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x588a, seq=2/512, ttl=64 (request in 28)

3.4.2 Exp4 - Pacotes ICMP transferidos entre Tux52 e o Router

12	14.575664639	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x5883, seq=1/256, ttl=64 (reply in 13)
13	14.575979419	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x5883, seq=1/256, ttl=63 (request in 12)
14	15.602752619	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) request id=0x5883, seq=2/512, ttl=64 (reply in 15)
15	15.603008592	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) reply id=0x5883, seq=2/512, ttl=63 (request in 14)

3.4.3 Exp4 - Pacotes ICMP transferidos entre Tux52 e o Tux54

37	28.466744110	172.16.50.1	172.16.51.254	ICMP	98	Echo (ping) request id=0x588e, seq=2/512, ttl=64 (reply in 38)
38	28.467024039	172.16.51.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x588e, seq=2/512, ttl=63 (request in 37)
39	29.490742255	172.16.50.1	172.16.51.254	ICMP	98	Echo (ping) request id=0x588e, seq=3/768, ttl=64 (reply in 40)
40	29.491044324	172.16.51.254	172.16.50.1	ICMP	98	Echo (ping) reply id=0x588e, seq=3/768, ttl=63 (request in 39)

3.4.4 Exp4 - Pacotes ICMP de redirecionamento

8	2.859068392	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) request id=0x10ea, seq=2/512, ttl=64 (reply in 10)
9	2.859235871	172.16.51.254	172.16.51.1	ICMP	126	Redirect (Redirect for host)
10	2.859446652	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) reply id=0x10ea, seq=2/512, ttl=63 (request in 8)
11	3.883065158	172.16.51.1	172.16.50.1	ICMP	98	Echo (ping) request id=0x10ea, seq=3/768, ttl=64 (reply in 13)
12	3.883230193	172.16.51.254	172.16.51.1	ICMP	126	Redirect (Redirect for host)
13	3.883433292	172.16.50.1	172.16.51.1	ICMP	98	Echo (ping) reply id=0x10ea, seq=3/768, ttl=63 (request in 11)

3.5.1 Exp5 - Pacotes DNS no ping ao *google.com*

7	5.228046998	172.16.50.1	172.16.1.1	DNS	70	Standard query 0x7f2c A google.com
8	5.228057684	172.16.50.1	172.16.1.1	DNS	70	Standard query 0xf635 AAAA google.com
9	5.230211728	172.16.1.1	172.16.50.1	DNS	86	Standard query response 0x7f2c A google.com A 142.250.200.78
10	5.230225975	172.16.1.1	172.16.50.1	DNS	98	Standard query response 0xf635 AAAA google.com AAAA 2a00:1450:4003:80d::200e
11	5.230517912	172.16.50.1	142.250.200.78	ICMP	98	Echo (ping) request id=0x0836, seq=1/256, ttl=64 (reply in 12)
12	5.248993714	142.250.200.78	172.16.50.1	ICMP	98	Echo (ping) reply id=0x0836, seq=1/256, ttl=112 (request in 11)

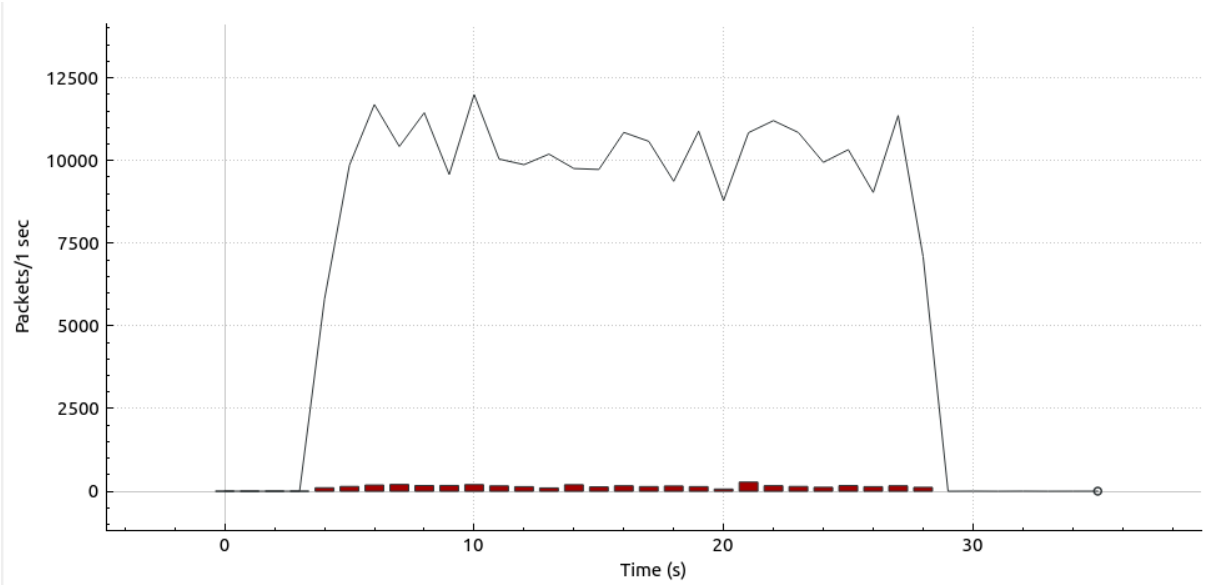
3.6.1 Exp6 - Início da transferência do ficheiro usando FTP

6	4.475250489	172.16.50.1	172.16.1.1	DNS	69	Standard query 0xc1f2 A ftp.up.pt
7	4.476093410	172.16.1.1	172.16.50.1	DNS	107	Standard query response 0xc1f2 A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15
8	4.476256421	172.16.50.1	193.137.29.15	TCP	74	37074 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=778752941 TSecr=0 WS=128
9	4.479885261	193.137.29.15	172.16.50.1	TCP	74	21 → 37074 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM=1 TSval=1740801032 TSecr=778752941 WS=128
10	4.479902442	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=778752945 TSecr=1740801032
11	4.485723128	193.137.29.15	172.16.50.1	FTP	139	Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
12	4.485733884	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [ACK] Seq=1 Ack=74 Win=64256 Len=0 TSval=778752951 TSecr=1740801036
13	4.485761332	193.137.29.15	172.16.50.1	FTP	141	Response: 220----------
14	4.485767827	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [ACK] Seq=1 Ack=149 Win=64256 Len=0 TSval=778752951 TSecr=1740801036
15	4.485831034	193.137.29.15	172.16.50.1	FTP	298	Response: 220-All connections and transfers are logged. The max number of connections is 200.
16	4.485836412	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [ACK] Seq=1 Ack=381 Win=64128 Len=0 TSval=778752951 TSecr=1740801036
17	4.485839275	193.137.29.15	172.16.50.1	FTP	78	Response: 220-
18	4.485843186	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [ACK] Seq=1 Ack=393 Win=64128 Len=0 TSval=778752951 TSecr=1740801037
19	4.486243031	172.16.50.1	193.137.29.15	FTP	81	Request: user anonymous
20	4.488753704	193.137.29.15	172.16.50.1	TCP	66	21 → 37074 [ACK] Seq=393 Ack=16 Win=65280 Len=0 TSval=1740801041 TSecr=778752951
21	4.488799660	193.137.29.15	172.16.50.1	FTP	100	Response: 331 Please specify the password.
22	4.488853927	172.16.50.1	193.137.29.15	FTP	80	Request: pass password
23	4.492568114	193.137.29.15	172.16.50.1	FTP	89	Response: 230 Login successful.
24	4.492611067	172.16.50.1	193.137.29.15	FTP	71	Request: pasv
25	4.495069638	193.137.29.15	172.16.50.1	FTP	118	Response: 227 Entering Passive Mode (193,137,29,15,229,147).
26	4.495165601	172.16.50.1	193.137.29.15	TCP	74	42642 → 58771 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=778752960 TSecr=0 WS=128
27	4.499775439	193.137.29.15	172.16.50.1	TCP	74	58771 → 42642 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM=1 TSval=1740801051 TSecr=778752960 WS=1
28	4.499787941	172.16.50.1	193.137.29.15	TCP	66	42642 → 58771 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=778752965 TSecr=1740801051
29	4.499802887	172.16.50.1	193.137.29.15	FTP	143	Request: retr pub/kodi/test-builds/darwin/tvos/kodi-20201003-09847870-master-tvos.deb
30	4.501992288	193.137.29.15	172.16.50.1	FTP	202	Response: 150 Opening BINARY mode data connection for pub/kodi/test-builds/darwin/tvos/kodi-20201003-09847870-m
31	4.510716438	193.137.29.15	172.16.50.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/kodi/test-builds/darwin/tvos/kodi-20201003-09847870-master-tvos.deb)
32	4.510736901	172.16.50.1	193.137.29.15	TCP	66	42642 → 58771 [ACK] Seq=1 Ack=1369 Win=64128 Len=0 TSval=778752976 TSecr=1740801063
33	4.510822597	193.137.29.15	172.16.50.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/kodi/test-builds/darwin/tvos/kodi-20201003-09847870-master-tvos.deb)

3.6.2 Exp6 - Fim da transferência do ficheiro usando FTP

2…	28.751808803	172.16.50.1	193.137.29.15	TCP	66	42642 → 58771 [ACK] Seq=1 Ack=273283073 Win=985216 Len=0 TSval=778777217 TSecr=1740825302
2…	28.751919781	193.137.29.15	172.16.50.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr pub/kodi/test-builds/darwin/tvos/kodi-20201003-09847870-master-tvos.deb)
2…	28.752048360	193.137.29.15	172.16.50.1	FTP-DA…	1960	FTP Data: 1894 bytes (PASV) (retr pub/kodi/test-builds/darwin/tvos/kodi-20201003-09847870-master-tvos.deb)
2…	28.752065122	172.16.50.1	193.137.29.15	TCP	66	42642 → 58771 [ACK] Seq=1 Ack=273286336 Win=984192 Len=0 TSval=778777217 TSecr=1740825302
2…	28.753859218	193.137.29.15	172.16.50.1	FTP	90	Response: 226 Transfer complete.
2…	28.753867529	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [ACK] Seq=112 Ack=662 Win=64128 Len=0 TSval=778777219 TSecr=1740825306
2…	28.753911041	172.16.50.1	193.137.29.15	FTP	71	Request: quit
2…	28.755921787	193.137.29.15	172.16.50.1	FTP	80	Response: 221 Goodbye.
2…	28.755960339	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [FIN, ACK] Seq=117 Ack=676 Win=64128 Len=0 TSval=778777221 TSecr=1740825308
2…	28.755960339	193.137.29.15	172.16.50.1	TCP	66	21 → 37074 [FIN, ACK] Seq=676 Ack=117 Win=65280 Len=0 TSval=1740825308 TSecr=778777219
2…	28.755968022	172.16.50.1	193.137.29.15	TCP	66	37074 → 21 [ACK] Seq=118 Ack=677 Win=64128 Len=0 TSval=778777221 TSecr=1740825308
2…	28.755980733	172.16.50.1	193.137.29.15	TCP	66	42642 → 58771 [FIN, ACK] Seq=1 Ack=273286336 Win=986240 Len=0 TSval=778777221 TSecr=1740825302
2…	28.757629349	193.137.29.15	172.16.50.1	TCP	66	58771 → 42642 [ACK] Seq=273286336 Ack=2 Win=65280 Len=0 TSval=1740825310 TSecr=778777221
2…	28.757756601	193.137.29.15	172.16.50.1	TCP	66	21 → 37074 [ACK] Seq=677 Ack=118 Win=65280 Len=0 TSval=1740825310 TSecr=778777221
2…	30.032239240	172.16.50.1	172.16.1.1	DNS	85	Standard query 0x4913 PTR 56.75.203.54.in-addr.arpa

3.6.3 Exp6 - Velocidade dos pacotes encaminhados pela rede em função do tempo, numa única transferência do Tux53



3.6.4 Exp6 - Gráfico da velocidade de transferência em função do tempo, no Tux53, começando outra transferência no Tux52 a meio da anterior