Faculdade de Engenharia da Universidade do Porto

Relatório de Redes de Computadores

Configuração e Análise de uma Rede de Computadores

RCOM 2022/23:

Manuel Alberto Pereira Ricardo
Rui Pedro de Magalhães Claro Prior
Eduardo Nuno Moreira Soares de Almeida

Estudantes & Autores:

José M. Carvalho up202005827@fe.up.pt
Tomás M. Carmo up202007590@fe.up.pt

Sumário

Neste relatório são especificados os passos e experiências feitos para o desenvolvimento de uma rede de computadores constituída por três computadores, um switch e um router e por fim o download de ficheiros através de um protocolo FTP desenvolvido e explicado pelos alunos.

Os resultados e dados obtidos são também explicados no decorrer deste relatório para conseguirmos tirar conclusões e fazer a ponte entre as partes prática e teórica desta Unidade Curricular.

Índice

1. Introdução

Este projeto foi desenvolvido de acordo com as diretrizes disponibilizadas pelos professores da disciplina, assim sendo este está repartido em duas partes .

A primeira propõe o desenvolvimento de uma aplicação de download com o fim de transferir um ficheiro através do protocolo FTP, antes de desenvolver testamos vários protocolos nas máquinas do laboratório como Telnet, SMTP, POP HTTP ,mas com o foco principal em entender o funcionamento geral do FTP. Toda esta parte será explicada em mais detalhe no capítulo seguinte.

A segunda e mais experimental parte deste projeto aborda uma série de experiências a serem realizadas nos laboratórios com o objetivo de configurar passo por passo uma *IP Network*. Estas experiências tem o objetivo de familiarizar os estudantes com as configurações genéricas e mais comuns de uma rede.

Todas estas experiências foram constantemente monitoradas e recortadas para que pudessem ser estudadas e analisadas no decorrer deste relatório.

2. Primeira Parte - Aplicação de Download

A primeira parte deste trabalho consistiu na implementação de uma aplicação de download que implementa um protocolo de aplicação **FTP (File Transfer Protocol)**, descrito no RFC959, através de ligações **TCP (Transmission Control Protocol)** com o uso de sockets. Esta aplicação adota a sintaxe de URL descrita no RFC1738 como a seguinte: **“ftp://[<user>:<password>@]<host>/<url-path>”**.

Arquitetura da aplicação de download

O procedimento da aplicação resume-se nos seguintes passos (cada um destes é acompanhado pela função **read_from_socket(int sockfd, char* buffer, size_t size)** que atualiza uma variável global chamada **buffer** com o output da socket selecionada) :

- Leitura e processamento do argumento dado, dividindo os campos “host”, “url-path” e “user” e “password” se existentes, com a função **parse_arguments(Arguments args, char* input)**, que os guardará numa estrutura **Arguments**;
- Criação da socket de controlo com o método **create_socket()**, seguido da chamada a **connect_socket(int sockfd, char *ip, int port)** para conexão da mesma com IP retornado em **getIP(char* hostname)** e porta 21;
- Login com as credenciais dadas ou com “anonymous” e “pass” na ausência destes utilizando a função **send_credentials(int sockfd, char* user, char* password)**;

- Mudança para modo passivo através do método **enter_passive_mode(int sockfd)**, de modo a que o cliente esteja encarregue de abrir a ligação TCP para receber os dados do ficheiro presente na path do url, guardando o IP e porta dados;
- Cálculo da porta para a segunda socket com a função **get_new_port(char* buffer)** e criação e conexão da mesma com o IP e porta dados anteriormente.
- Envio do ficheiro pela primeira socket com **send_file(int sockfd, char* path)**
- Criação de um ficheiro com o nome dado na path e leitura dos bytes dados pela socket anterior usando o método **save_to_file(int sockfd, char* filename)**;
- Fecho das sockets com **close_connection(int sockfd)**;

Resultados da aplicação

Como podemos ver na figura 1 e 2 em **5. Anexos**, a aplicação funciona como pretendido, sendo que, caso executemos a mesma com argumentos inválidos é nos dada uma mensagem de erro. Também podemos observar que na ausência de campos de nome de utilizador e password, a aplicação infere os mesmos como sendo “anonymous” e “pass”.

3. Segunda Parte - Configuração e Análise de uma Rede

Experiência 1 - Configuração de uma rede IP

Na primeira experiência tínhamos como objetivo configurar o IP de dois computadores, o tux4 e o tux3, ligar os dois a um switch e observar os pacotes transferidos entre eles através do comando **ping**.

Principais comandos:

Para configurar o IP executamos:

- **ifconfig eth0 up <IP-address>,** sendo que o endereço correspondia à rede 172.16.10.0/24 e o último ponto estava associado ao computador (.1 no tux3 e .254 no tux4).

Para o tux3 comunicar com o tux4 executamos no tux3:

- **ping 172.16.10.254**

Análise de Resultados:

Através da captura na figura 3 em 5. Anexos podemos observar **pacotes ARP (Address Resolution Protocol)**, que têm como objetivo mapear endereços de rede IPs com os endereços físicos privados MAC definidos em cada máquina.

O primeiro corresponde ao pacote que o tux3 envia em broadcast (na rede 172.16.10.0/24), uma vez que não sabe o endereço MAC do destinatário (tux4) e o segundo

será a resposta a este, definindo o endereço MAC em falta (Os endereços IP e MAC podem ser vistos nos detalhes dos pacotes em 5. Anexos nas figuras 4 e 5).

Para além deste pacotes, o comando ping gera **pacotes ICMP (Internet control message protocol)** que consistem em enviar mensagens de erro ou trocar informação sobre a conectividade da rede. Neste caso temos presentes pacotes ICMP alternando entre um pedido de comunicação do tux3 para o tux4 e uma resposta do tux4, sendo que cada pacote contém os endereços IP do remetente e destinatário correspondentes.

O Wireshark apresenta **o nome do protocolo correspondente para diferenciar os pacotes de diferentes tipos** com ARP, IP ou ICMP, sendo que estes **tipos são especificados no header de uma ethernet frame** e para além disso, o programa apresenta o **comprimento de qualquer uma destas frames nos detalhes** da respetiva.

Caso seja necessário comunicar com o próprio computador que faz um pedido, temos presente a **interface loopback**. Esta consiste numa **interface virtual da rede que permite ao computador receber respostas de si próprio**, para testar aplicações, serviços de rede ou até a sua configuração, **sem a conexão a uma rede externa**.

Experiência 2 - Implementação de Bridges num Switch

Nesta segunda experiência implementamos duas bridges, uma com as máquinas anteriores e a outra com uma nova máquina, o tux2, com o objetivo de testar comunicações entre máquinas.

Principais comandos:

Para configurar o endereço IP do tux2 usamos o mesmo comando acima com o endereço 172.16.11.1 (temos assim uma nova sub-rede).

Para implementar as bridges:

- **/interface bridge add name=<bridge-name>**, sendo que foram criadas as bridges **bridge10** e **bridge11**.

Para remover as interfaces de cada máquina da bridge default:

- **/interface bridge port remove [find interface=ether<port>]**, substituindo **<port>** pelo número da porta do switch ligada à interface de cada máquina usada.

Para adicionar as interfaces às bridges recentemente criadas:

- **/interface bridge port add bridge=<bridge-name> interface=ether<port>**, sendo **<bridge-name>** “bridge10” no caso do tux3 e do tux4 e “bridge11”

para o tux2.

Para comunicar entre máquinas:

- **ping <target-address>**, substituindo **<target-address>** pelo endereço IP que pretendemos comunicar.

Análise de Resultados:

Após a configuração do endereço IP do tux2, procedemos à configuração das bridges. Veremos especificamente a bridge10, esta consistiu na sequência de comandos apresentada em cima: **criação da bridge10** no terminal do switch, **remoção do tux3 e do tux4 da bridge default** e por fim, **adição das interfaces das mesmas na bridge10**.

Nesta experiência teremos também **dois domínios de broadcast**, um para cada sub-rede dado pelos endereços IP 172.16.10.255 e 172.16.11.255. Podemos concluir isso através das capturas, uma vez que no momento em que **o tux3 faz broadcast, apenas obtém resposta do tux4 e não do tux2**. Do mesmo modo, quando **o tux2 faz broadcast, não obtém resposta de nenhuma outra máquina**. (Figuras 7 e 8 em 5. Anexos).

Experiência 3 - Configuração de um Router em Linux

Na terceira experiência utilizamos o tux4 como router e verificamos de novo a comunicação entre máquinas. Implementando a mesma rede que a anterior e adicionando uma interface eth1 do tux4 à bridge11.

Nota: As capturas desta experiência são relativas à bancada 3, no entanto para coerência refiro os endereços da bancada 1 como anteriormente.

Principais comandos:

Mais uma vez usamos os comandos acima para configuração do endereço IP da nova interface do tux4 (172.16.11.253) e adição do mesmo à bridge11.

Para ativar IP-fowarding:

- **echo 1 > /proc/sys/net/ipv4/ip forward**

Para desativar ICMP echo-ignore-broadcast:

- **echo 0 > /proc/sys/net/ipv4/icmp echo ignore broadcasts**

Para adicionar rotas:

- **route add -net <destination> gw <gateway>**, sendo este comando utilizado para definir a rota do tux2 para o tux3 e vice-versa, através do tux4. **<destination>** será a sub-rede que o tux2 ou tux3 quer aceder e **<gateway>** será o endereço da interface do tux4 correspondente à bridge.

Análise de Resultados:

Uma rota é um caminho definido num computador para este saber onde se dirigir ao comunicar de uma rede para outra.

Cada entrada da tabela de rotas de uma máquina terá o **endereço IP de destino**, **a gateway**, **a máscara do IP**, **flags** que oferecem informações sobre a rota, **metric** que consiste no custo da rota, **ref** que se resume no número de referências para a rota, **use** que consiste num contador de pesquisas para a rota e **Iface que corresponde à interface responsável pela gateway(eth0, eht1,...)**.

Ao analisar as rotas de cada máquina, deparamos-nos com uma rota para a própria bridge (por exemplo no caso do tux3 temos uma rota com destino a 172.16.10.0/24 através da porta 0.0.0.0, a gateway) e, no caso do tux2 e tux3, com uma rota definida por nós com destino 172.16.10.0/24 e gateway 172.16.11.253 e com destino 172.16.11.0/24 e gateway 172.16.10.254 respetivamente.

As mensagens ARP, são, mais uma vez, apenas observadas no caso de uma nova comunicação, como por exemplo na comunicação do tux3 com a interface ether1 do tux4, havendo mapeamento do endereço MAC desta última interface. Quanto aos pacotes ICMP, veremos mensagens de pedido e resposta entre todas as máquinas, visto que, depois de adicionadas as rotas, todos os computadores comunicam entre si e os endereços presentes serão os IP e MAC do remetente e do destinatário.

Experiência 4 - Configuração de um Router Comercial e implementação de NAT

Esta experiência teve como objetivo estabelecer uma ligação à rede do laboratório com um router comercial (configurando uma interface **ether1** para essa mesma rede, **172.16.1.0/24** e outra interface **ether2** para ligar à bridge11) e uma ligação NAT.

Nota: As capturas desta experiência são relativas à bancada 6 e sala I320, no entanto para coerência refiro os endereços da bancada 1 como anteriormente e sala I321.

Comandos principais:

Para a configuração dos endereços IP do router comercial no terminal deste:

- **/ip address add address=172.16.1.19/24 interface=ether1**
- **/ip address add address=172.16.11.254/24 interface=ether2**

Para configurar uma máquina tux como router default:

- **route add default gw <gateway>**, sendo que <**gateway>** será a porta do

router pretendido No caso da configuração do tux4 como router do tux3 será **172.16.10.254**, no caso da configuração do Rc como router do tux2 e do tux4 será **172.16.11.254** executando na respetiva máquina.

Para configurar rotas do router comercial no terminal:

- **/ip address route add dst-address=172.16.10.0/24 gateway=172.16.11.253**, ou seja, se o router comercial quiser aceder à rede 172.16.10.0/24, é redirecionado para a porta do tux4 presente na bridge11.
- **/ip address route add 0.0.0.0/0 gateway=172.16.1.254**, ou seja, configurar como default route a porta da rede de laboratórios.

Para ativar a NAT no router comercial:

- **/ip firewall nat add chain=srcnat action=masquerade out-interface=ether1**, sendo este executado no terminal do router

Para desativar a NAT no router comercial:

- **/ip firewall nat disable 0**, sendo este executado no terminal do router

Análise de Resultados:

Para adicionar as rotas para o router comercial nas outras máquinas, teremos que **definir primeiro os endereços IP no router comercial** através dos comandos acima e de seguida **definimos não só as rotas que permitem usar o router comercial como default router, mas também adicionar rotas ao mesmo**, mais uma vez, com os comandos anteriores.

Como foi dito anteriormente, as rotas default de cada máquina direcionam agora para o router comercial, “diretamente” no caso das máquinas presentes na bridge11 e “indiretamente” no tux3, passando pela porta presente na bridge10 do tux4 antes do router. Através do comando **traceroute <ip-address>** contamos dois **hops** do tux3 até ao tux2, um para a porta do ether0 do tux4 e o final que chega ao tux2. Removemos a rota deste último para a rede 172.16.10.0/24 e desativamos redirecionamentos no mesmo com:

- **route del -net 172.16.10.0/24**
- **echo 0 > /proc/sys/net/ipv4/conf/eth0/accept_redirects**
- **echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects**

Verificamos agora que o tux2 necessita de três **hops** para chegar ao tux3, sendo que retiramos a rota de destino à rede 172.16.10.0/24 pelo eht1 do tux4, ou seja, **um hop para o router comercial** (foi definido como default), outro para o **eth1 do tux4** e por fim um para o **tux3**. Além disso, é possível encontrar novos pacotes ICMP de redirecionamento pelo

router comercial (Figura 15 em 5. Anexos).

De seguida demos **ping** à rede do laboratório, com sucesso, uma vez que a NAT (Network Address Translation) do router comercial estava ativa (Figura 16 em 5. Anexos).

A NAT consiste na **tradução de endereços IP de dispositivos presentes numa rede privada para um único IP público**, permitindo assim que múltiplos dispositivos partilhem uma única conexão à internet mantendo a sua rede interna privada e segura. No contexto da nossa rede, significa que a NAT permite às máquinas na nossa rede local, acesso à rede externa (laboratório). Assim que desativamos a NAT, não é possível verificar a conectividade entre o tux3 e o router do laboratório.

Experiência 5 - DNS

Na quinta experiência, configuramos o sistema DNS com o objetivo de comunicar com outros hosts a partir de um nome ao invés do endereço IP.

Nota: As capturas desta experiência são relativas à bancada 6 e sala I320, no entanto para coerência refiro os endereços da bancada 1 como anteriormente e sala I321.

Comandos principais:

Para alterar o ficheiro **resolv.conf**:

- **echo "nameserver 172.16.1.254" > /etc/resolv.conf**

Para dar ping ao hostname usamos o comando **ping <hostname>** com o nome pretendido tal como temos feito anteriormente com os endereços IP.

Análise de Resultados:

Para configurar o sistema DNS, foi apenas necessário alterar o conteúdo do ficheiro **/etc/resolv.conf** com o nome **nameserver** e o IP requerido, neste caso o do servidor netlab 172.16.1.1. Depois de configurado, observamos que os pacotes enviados de DNS consistem em primeiro lugar a um pedido ao **google.com** do **endereço do host (tipo A)** e de seguida a um pedido do **endereço IPv6 (tipo AAAA)**, sendo o anterior IPv4. Posteriormente cada um destes pedidos recebe resposta, para o primeiro recebemos o endereço **142.250.200.78** e para o segundo o endereço **2a00:1450:4003:80d::200e**. Por fim, após o primeiro pedido de ping e resposta temos mais dois pacotes DNS, o primeiro será mais um pedido mas desta vez do **tipo PTR (Pointer Record)**, um pointer para o "domain name" seguido de uma resposta com o mesmo. Este pedido, ao contrário do de tipo A, mapeia um endereço IP num "domain name", permitindo a partir do endereço obtido inicialmente, aceder a todos os hostnames.

Experiência 6 - Conexões TCP

Nesta última experiência utilizamos a aplicação concebida na parte 1 do trabalho com o objetivo de analisar os pacotes enviados entre o computador e o servidor ftp.

Nota: As capturas desta experiência são relativas à bancada 6 e sala I320.

Comandos principais:

Para executar a aplicação de download:

- **make clean && make**
- **./bin/download <link-ftp>**

Análise de resultados:

Numa primeira fase de conexão FTP, **são abertas duas conexões TCP**, uma na porta 21 do host dado, que por sua vez, envia a informação de controlo FTP e outra a ser definida pela resposta ao comando **pasv** (entrada no modo passivo por parte da primeira socket). Para além desta, temos **a fase de troca de informação e de fecho de ligação**.

O **ARQ TCP é um mecanismo com o intuito de minimizar erros durante a transferência de pacotes**, enviando pacotes de controlo ACK de resposta por cada trama de informação ou NACK em caso de anomalia desta. Como campos TCP relevantes teremos o **número de sequência** e o **número ACK** (esperado), sendo que estes são usados para avaliar anomalias na ordem de receção de pacotes.

Para impedir perdas de pacotes e atrasos quando a rede está sobrecarregada, o protocolo TCP usa um **mecanismo de controlo de congestão**. Neste mecanismo, a **quantidade de pacotes transferidos é avaliada em relação à quantidade de pacotes lidos pelo recetor**, ou seja, são enviados cada vez mais pacotes até que a janela de congestão (campo enviado pelo recetor para informar quantos bytes conseguiu ler) esteja cheia. Caso esta janela exceda a sua capacidade, são descartadas as tramas adicionais e é enviada uma resposta NACK correspondente ao número de sequência destas, ao passo que, no próximo envio, serão mandados menos dados. Como serão dois downloads em simultâneo, é expectável que **o bitrate reduza para metade no momento de congestão**.

Este controlo foi confirmado através da captura de wireshark. Apesar de não muito aparente devido à quantidade de dados lidos de uma vez (2048 bytes) e à diferença de tamanho de ficheiros, é visível uma grande **redução de transferência de pacotes TCP a seguir aos 5 segundos**, que nos indica o **início do download no tux2**, sendo que **pouco depois, há um aumento e estabilização do bitrate** até a transferência do tux3 terminar.

4. Conclusões

Com a realização deste trabalho laboratorial podemos concluir que para um estudante de Engenharia Informática é vital ter uma noção concreta e sólida sobre os tópicos mais básicos de funcionamento de uma rede de computadores.

Dito isto, este trabalho demonstrou ser uma peça chave para consolidar os conhecimentos previamente adquiridos nas aulas teóricas, conseguimos colocar em prática tudo o que tínhamos aprendido assim como visualizar os erros mais comuns do decorrer das experiências, como por exemplo, no decorrer de uma experiências nos deparamos que nada funcionava apesar das configurações estarem 100% corretas. Isto devia-se a um simples RJ45 estar conectado na porta erradamente.

Em tópicos mais específicos podemos também entender exatamente o funcionamento e implementação do protocolo FTP e de que modo a forma a implementação influenciou a execução da experiência 6.

5. Anexos

Código da Aplicação de download:

- main.c

```
#include "utils.h"

int main(int argc, char *argv[]){
    //PROGRAM STEPS
    //1-Parse arguments
    if(argc != 2){
        printf("[ERROR] Invalid argument!\n");
                                printf("***    Usage:    download
ftp://[<user>:<password>@]<host>/<url-path>\n");
        exit(-1);
    }
    Arguments args;
    if(parse_arguments(&args, argv[1]) == -1){
        printf("[ERROR] Invalid argument!\n");
                                printf("***    Usage:    download
ftp://[<user>:<password>@]<host>/<url-path>\n");
```

```
        exit(-1);
    }

    printf("Arguments: \n");
    printf("User: %s\n", args.user);
    printf("Password: %s\n", args.password);
    printf("Hostname: %s\n", args.host);
    printf("Path: %s\n", args.urlPath);

    //2-Create socket and connect to server

    // Get IP with hostname
    char ip_address[16];
    int port = 21;
    strcpy(ip_address, getIP(args.host));
    printf("[LOG] Connecting to %s:%d\n", ip_address,port);

    // Create and connect socket
    int sockfd = create_socket();
    connect_socket(sockfd, ip_address, port);

    // Just some feedback
    char *buffer = (char *)malloc(MAX_LENGTH*sizeof(char));
    size_t size = MAX_LENGTH;
    read_from_socket(sockfd, buffer, size);

    //3-Login with user and password
    send_credentials(sockfd, args.user, args.password);
    //4-Enter passive mode
    enter_passive_mode(sockfd);
    read_from_socket(sockfd, buffer, size);

    //5-Calculate port
    int new_port = get_new_port(buffer);
    if(new_port < 0){
        printf("[ERROR] Invalid port read from passive mode.\n");
        exit(-1);
    }
    //7-Send the file to client with the given port
```

```
        // Create receiver socket and connect
        int rcv_sockfd = create_socket();
        printf("[LOG] Connecting to %s:%d\n", ip_address, new_port);
        connect_socket(rcv_sockfd, ip_address, new_port);

        // Send file from control socket
        send_file(sockfd, args.urlPath);
        read_from_socket(sockfd, buffer, size);
        // Save to new file
        if(buffer[0] != '5' && buffer[0] != '4'){
            char filename[MAX_LENGTH];
            getFilename(filename, args.urlPath);
            if(save_to_file(rcv_sockfd, filename) < 0){
                printf("[ERROR] File download failed.\n");
            }
        }

        //8-Close sockets
        if(close_connection(sockfd) < 0){
                    printf("[ERROR]  Couldn't  close  the  control  socket
connection.\n");
            exit(-1);
        }
        if(close_connection(rcv_sockfd) < 0){
                     printf("[ERROR]  Couldn't  close  the  client  socket
connection.\n");
            exit(-1);
        }

        free(buffer);

        return 0;
    }
```

- utils.h

```
    #pragma once

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

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

// Macros
#define MAX_LENGTH 255

// Structures
typedef struct {
    char user[MAX_LENGTH];
    char password[MAX_LENGTH];
    char host[MAX_LENGTH];
    char urlPath[MAX_LENGTH];
} Arguments;

// Functions
int parse_arguments(Arguments *args, char *input);
int create_socket();
char* getIP(char* hostname);
void connect_socket(int sockfd, char* ip, int port);
void read_from_socket(int sockfd, char* buffer, size_t size);
void send_credentials(int sockfd, char* user, char* password);
void enter_passive_mode(int sockfd);
int get_new_port(char* buffer);
void send_file(int sockfd, char* path);
void getFilename(char* filename, char* path);
int save_to_file(int sockfd, char* filename);
int close_connection(int sockfd);
```

- utils.c

```c
#include "utils.h"

int parse_arguments(Arguments *args, char *input){
    // Parse beggining
    if(strncmp(input, "ftp://", 6) != 0){
            //printf("[ERROR]  The  input  given  must  start  with
```

```
\"ftp://\"\n");
            return -1;
        }

        // Parse user and password if they exist
        if(strchr(input+6, '@') != NULL){
              if(sscanf(input, "ftp://%[^:]:%[^@]@%[^/]/%s", args->user,
args->password, args->host, args->urlPath) != 4){
                printf("[ERROR] Couldn't parse user and pass\n");
                return -1;
            }
            /*printf("Arguments: \n");
            printf("User: %s\n", args->user);
            printf("Password: %s\n", args->password);
            printf("Hostname: %s\n", args->host);
            printf("Path: %s\n", args->urlPath);*/

            return 1;
        }
        else {
                        if(sscanf(input,   "ftp://%[^/]/%s",   args->host,
args->urlPath) != 2){
                printf("[ERROR] Couldn't parse host and path\n");
                return -1;
            }

            strcpy(args->user, "anonymous");
            strcpy(args->password, "pass");

            /*printf("Arguments: \n");
            printf("Hostname: %s\n", args->host);
            printf("Path: %s\n", args->urlPath);*/

            return 1;
        }

        return -1;
    }
```

```
int create_socket(){
    int sockfd;
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
    }

    return sockfd;
}

char* getIP(char* hostname){
    struct hostent *h;

    if ((h = gethostbyname(hostname)) == NULL) {
        printf("[ERROR] Couldn't get IP address.\n");
        exit(-1);
    }

    return inet_ntoa(*((struct in_addr *)h->h_addr));
}

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

    if (connect(sockfd,
                (struct sockaddr *) &server_addr,
                sizeof(server_addr)) < 0) {
        printf("[ERROR] Couldn't connect socket.\n");
        exit(-1);
    }
}
```

```
void read_from_socket(int sockfd, char* buffer, size_t size){
    FILE *fp = fdopen(sockfd, "r");
    printf("[LOG] From Socket #%d: \n", sockfd);
    do {
        buffer = fgets(buffer, size, fp);
        printf("%s", buffer);
    } while (!('1' <= buffer[0] && buffer[0] <= '5') || buffer[3] !=
' ');

    if(buffer[0] == '5' && buffer[0] == '4'){
        printf("[ERROR] Failed action.\n");
    }
}

void send_credentials(int sockfd, char* user, char* password){
    printf("[LOG] Sending user to socket.\n");
    char buffer[MAX_LENGTH];
    char command[MAX_LENGTH];
    sprintf(command, "user %s\n", user);
    size_t bytes;
    if ((bytes = write(sockfd, command, 6+strlen(user))) <= 0){
        printf("[ERROR] Couldn't write to socket.\n");
        exit(-1);
    }
    read_from_socket(sockfd, buffer, MAX_LENGTH);
    //printf("Written %ld bytes\n", bytes);

    printf("[LOG] Sending password to socket.\n");
    sprintf(command, "pass %s\n", password);
    if ((bytes = write(sockfd, command, 6+strlen(password))) <= 0){
        printf("[ERROR] Couldn't write to socket.\n");
        exit(-1);
    }
    read_from_socket(sockfd, buffer, MAX_LENGTH);
    //printf("Written %ld bytes\n", bytes);
}

void enter_passive_mode(int sockfd){
    size_t bytes;
```

```
        printf("[LOG] Sending pasv to socket.\n");
        if ((bytes = write(sockfd, "pasv\n", 5)) <= 0){
            printf("[ERROR] Couldn't write to socket.\n");
            exit(-1);
        }
    }

    int get_new_port(char* buffer){
        int lb, rb;
        int ip_addr[4];
                    if(sscanf(buffer,    "227    Entering    Passive    Mode
(%d,%d,%d,%d,%d,%d).",     &ip_addr[0],     &ip_addr[1],     &ip_addr[2],
&ip_addr[3], &lb, &rb) == 6){
            return lb*256+rb;
        }

        return -1;
    }

    void send_file(int sockfd, char* path){
        size_t bytes;
        char command[MAX_LENGTH];
        printf("[LOG] Sending file...\n");

        sprintf(command, "retr %s\n", path);
        if ((bytes = write(sockfd, command, strlen(command))) <= 0){
            printf("[ERROR] Couldn't write to socket.\n");
            exit(-1);
        }
        //printf("Written %ld bytes\n",  bytes);
    }

    void getFilename(char* filename, char* path){
        char* token;

        token = strtok(path, "/");
        while(token != NULL){
            strcpy(filename,token);
            token = strtok(NULL, "/");
```

```
        }
    }

    int save_to_file(int sockfd, char* filename){
        FILE *f = fopen(filename, "wb");
        char buf[1024];
        size_t bytes;

        printf("[LOG] Starting downloading file.\n");
        while((bytes = read(sockfd, buf, sizeof(buf)))) {
            if(bytes < 0){
                printf("[ERROR] Couldn't read from socket.\n");
                return -1;
            }
            if(fwrite(buf, bytes, 1, f) < 0){
                printf("[ERROR] Couldn't write to target file.\n");
            }

                //printf("[LOG] Written %ld bytes to target file.\n",
bytes);
        }

        printf("[LOG] Finished download.\n");
        return 1;
    }

    int close_connection(int sockfd){
        printf("[LOG] Closing connection #%d.\n", sockfd);
        return close(sockfd);
    }
```

Capturas de Ecrã:

```
tomas@LAPTOP-CGDFVIU0:/mnt/c/Users/tomas/OneDrive/Documentos/GitHub/FEUP-RC-22-23/Projeto 2/Parte 1$ make clean && make
rm -f bin//download
gcc -Wall -o bin//download main.c src//utils.c -Iinclude/
tomas@LAPTOP-CGDFVIU0:/mnt/c/Users/tomas/OneDrive/Documentos/GitHub/FEUP-RC-22-23/Projeto 2/Parte 1$ ./bin/download
[ERROR] Invalid argument!
*** Usage: download ftp://[<user>:<password>@]<host>/<url-path>
tomas@LAPTOP-CGDFVIU0:/mnt/c/Users/tomas/OneDrive/Documentos/GitHub/FEUP-RC-22-23/Projeto 2/Parte 1$ ./bin/download ftp://ftp.gnu.
org/gnu/gcc/README.olderversions
Arguments:
User: anonymous
Password: pass
Hostname: ftp.gnu.org
Path: gnu/gcc/README.olderversions
[LOG] Connecting to 209.51.188.20:21
[LOG] From Socket #3:
220 GNU FTP server ready.
[LOG] Sending user to socket.
[LOG] From Socket #3:
230-NOTICE (Updated October 15 2021):
230-
230-If you maintain scripts used to access ftp.gnu.org over FTP,
230-we strongly encourage you to change them to use HTTPS instead.
230-
230-Eventually we hope to shut down FTP protocol access, but plan
230-to give notice here and other places for several months ahead
230-of time.
230-
230----
230-
230-Due to U.S. Export Regulations, all cryptographic software on this
230-site is subject to the following legal notice:
230-
```

Fig.1 - Execução inicial do programa “download”

```
230-
230-    This site includes publicly available encryption source code
230-    which, together with object code resulting from the compiling of
230-    publicly available source code, may be exported from the United
230-    States under License Exception "TSU" pursuant to 15 C.F.R. Section
230-    740.13(e).
230-
230-This legal notice applies to cryptographic software only. Please see
230-the Bureau of Industry and Security (www.bxa.doc.gov) for more
230-information about current U.S. regulations.
230 Login successful.
[LOG] Sending password to socket.
[LOG] From Socket #3:
230 Already logged in.
[LOG] Sending pasv to socket.
[LOG] From Socket #3:
227 Entering Passive Mode (209,51,188,20,115,48).
[LOG] Connecting to 209.51.188.20:29488
[LOG] Sending file...
[LOG] From Socket #3:
150 Opening BINARY mode data connection for gnu/gcc/README.olderversions (120 bytes).
[LOG] Starting downloading file.
[LOG] Finished download.
[LOG] Closing connection #3.
[LOG] Closing connection #4.
tomas@LAPTOP-CGDFVIU0:/mnt/c/Users/tomas/OneDrive/Documentos/GitHub/FEUP-RC-22-23/Projeto 2/Parte 1$
```

Fig.2 - Finalização da execução do programa “download”

8	8.008561875	Routerbo_1c:8c:b0	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:99 Cost = 0 Port = 0x8018
9	9.234402369	HewlettP_61:2d:ef	Broadcast	ARP	42	Who has 172.16.10.254? Tell 172.16.10.1
10	9.234525219	HewlettP_61:2f:24	HewlettP_61:2d:ef	ARP	60	172.16.10.254 is at 00:21:5a:61:2f:24
11	9.234543867	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x07b4, seq=1/256, ttl=64 (reply in 12)
12	9.234631098	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x07b4, seq=1/256, ttl=64 (request in 11)
13	10.010771067	Routerbo_1c:8c:b0	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8c:99 Cost = 0 Port = 0x8018
14	10.235200363	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x07b4, seq=2/512, ttl=64 (reply in 15)
15	10.235301912	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x07b4, seq=2/512, ttl=64 (request in 14)
16	11.259209238	172.16.10.1	172.16.10.254	ICMP	98	Echo (ping) request id=0x07b4, seq=3/768, ttl=64 (reply in 17)
17	11.259314348	172.16.10.254	172.16.10.1	ICMP	98	Echo (ping) reply id=0x07b4, seq=3/768, ttl=64 (request in 16)

Fig.3 - Captura de Wireshark do ping entre tux3 e tux4 na experiência 1

```
> Frame 9: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_61:2d:ef (00:21:5a:61:2d:ef), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
v 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_61:2d:ef (00:21:5a:61:2d:ef)
    Sender IP address: 172.16.10.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 172.16.10.254
```

Fig.4 - Endereços IP e MAC no primeiro pacote ARP

```
> Frame 10: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_61:2f:24 (00:21:5a:61:2f:24), Dst: HewlettP_61:2d:ef (00:21:5a:61:2d:ef)
v 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_61:2f:24 (00:21:5a:61:2f:24)
    Sender IP address: 172.16.10.254
    Target MAC address: HewlettP_61:2d:ef (00:21:5a:61:2d:ef)
    Target IP address: 172.16.10.1
```

Fig.5 - Endereços IP e MAC no pacote ARP de resposta

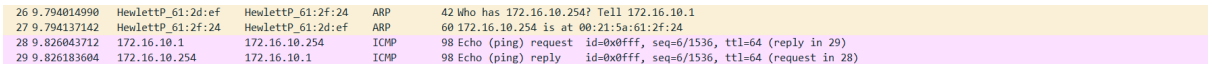

Fig.6 - Captura de Wireshark do ping entre tux3 e tux4 na experiência 2

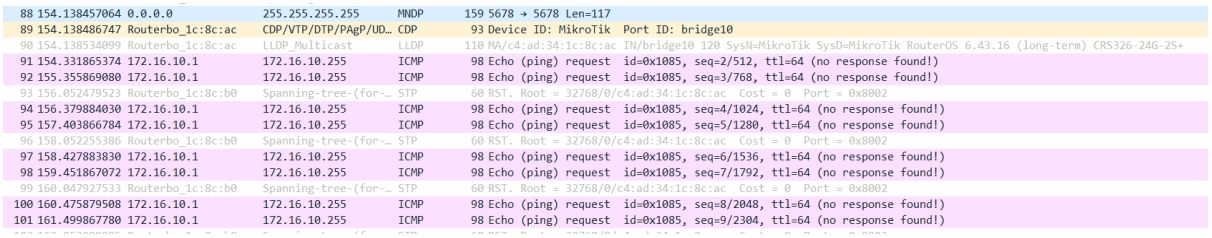

Fig.7 - Captura de Wireshark do ping broadcast no tux3 na experiência 2 (apesar de não aparecer no wireshark, no terminal do tux3 estava presente resposta do tux4)

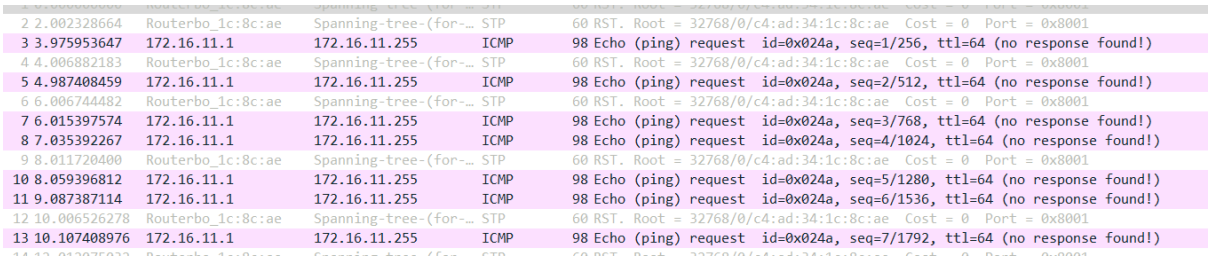

Fig.8 - Captura de Wireshark do ping broadcast do tux2 na experiência 2

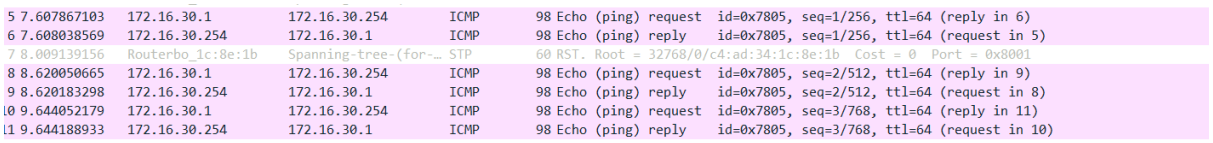

Fig.9 - Captura de Wireshark do ping entre tux3 e tux4 em eth0 na experiência 3

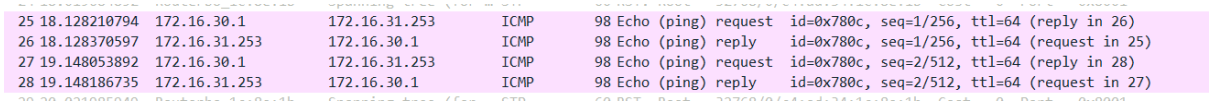

Fig.10 - Captura de Wireshark do ping entre tux3 e tux4 em eth1 na experiência 3

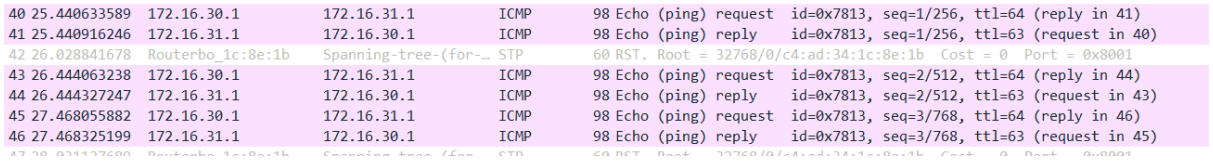

Fig.11 - Captura de Wireshark do ping entre tux3 e tux2 na experiência 3

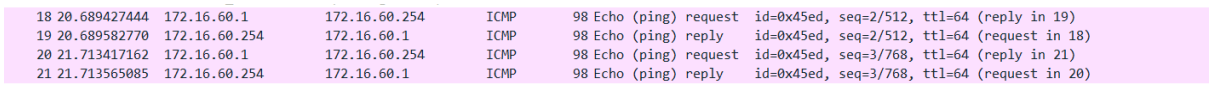

Fig.12 - Captura de Wireshark do ping entre tux3 e tux4 eth0 na experiência 4

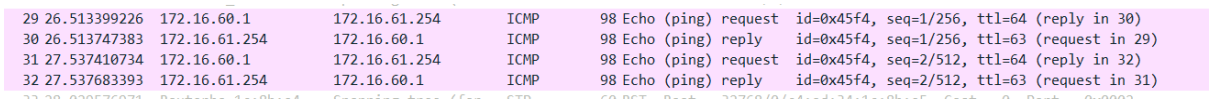

Fig.13 - Captura de Wireshark do ping entre tux3 e o router comercial na experiência 4

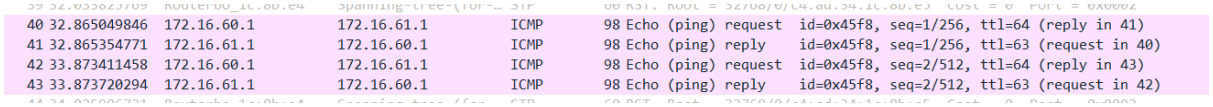

Fig.14 - Captura de Wireshark do ping entre tux3 e tux2 na experiência 4

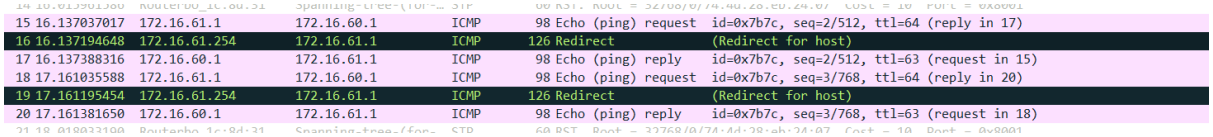

Fig.15 - Captura de Wireshark do ping entre tux3 e tux2 na experiência 4 após remover rota do tux4 à rede da bridge do tux3

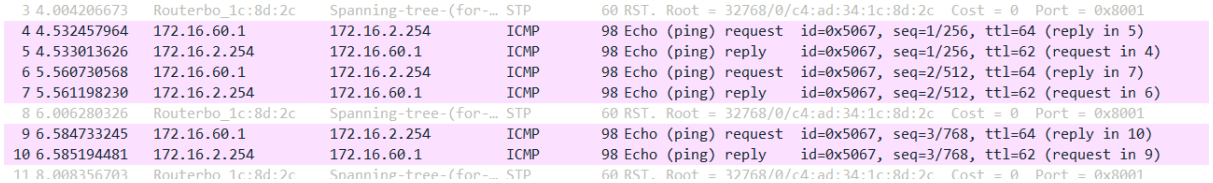

Fig.16 - Captura de Wireshark do ping entre tux3 e router do laboratório na experiência 4

2	0.743367511	172.16.60.1	172.16.2.1	DNS	70	Standard query 0x0002 A google.com
3	0.743377987	172.16.60.1	172.16.2.1	DNS	70	Standard query 0x070a AAAA google.com
4	0.744250453	172.16.2.1	172.16.60.1	DNS	86	Standard query response 0x0002 A google.com A 142.250.200.78
5	0.744272244	172.16.2.1	172.16.60.1	DNS	98	Standard query response 0x070a AAAA google.com AAAA 2a00:1450:4003:80d::200e
6	0.744674394	172.16.60.1	142.250.200.78	ICMP	98	Echo (ping) request id=0x51e5, seq=1/256, ttl=64 (reply in 7)
7	0.762515451	142.250.200.78	172.16.60.1	ICMP	98	Echo (ping) reply id=0x51e5, seq=1/256, ttl=112 (request in 6)
8	0.762625871	172.16.60.1	172.16.2.1	DNS	87	Standard query 0x56c9 PTR 78.200.250.142.in-addr.arpa
9	0.763447632	172.16.2.1	172.16.60.1	DNS	126	Standard query response 0x56c9 PTR 78.200.250.142.in-addr.arpa PTR mad07s24-in-f14.1e100.net
10	1.746528511	172.16.60.1	142.250.200.78	ICMP	98	Echo (ping) request id=0x51e5, seq=2/512, ttl=64 (reply in 11)
11	1.763007999	142.250.200.78	172.16.60.1	ICMP	98	Echo (ping) reply id=0x51e5, seq=2/512, ttl=112 (request in 10)
12	2.002092231	Routerbo_1c:8d:2c	Spanning-tree-(for-…	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8d:2c Cost = 0 Port = 0x8001
13	2.748102911	172.16.60.1	142.250.200.78	ICMP	98	Echo (ping) request id=0x51e5, seq=3/768, ttl=64 (reply in 14)
14	2.765036372	142.250.200.78	172.16.60.1	ICMP	98	Echo (ping) reply id=0x51e5, seq=3/768, ttl=112 (request in 13)

Fig.17 - Captura de Wireshark do ping no tux3 para o "google.com" na experiência 5

2	0.135934302	172.16.60.1	172.16.2.1	DNS	69	Standard query 0x9877 A ftp.up.pt
3	0.136823180	172.16.2.1	172.16.60.1	DNS	107	Standard query response 0x9877 A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15
4	0.136903498	172.16.60.1	193.137.29.15	TCP	74	41706 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2743823056 TSecr=0 WS=128
5	0.140364516	193.137.29.15	172.16.60.1	TCP	74	21 → 41706 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM TSval=2610028735 TSecr=2743823056 WS=128
6	0.140384840	172.16.60.1	193.137.29.15	TCP	66	41706 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2743823059 TSecr=2610028735
7	0.145097985	193.137.29.15	172.16.60.1	FTP	139	Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
8	0.145107623	172.16.60.1	193.137.29.15	TCP	66	41706 → 21 [ACK] Seq=1 Ack=74 Win=64256 Len=0 TSval=2743823064 TSecr=2610028740
9	0.145137166	193.137.29.15	172.16.60.1	FTP	141	Response: 220----------
10	0.145142335	172.16.60.1	193.137.29.15	TCP	66	41706 → 21 [ACK] Seq=1 Ack=149 Win=64256 Len=0 TSval=2743823064 TSecr=2610028740
11	0.145207148	193.137.29.15	172.16.60.1	FTP	310	Response: 220-All connections and transfers are logged. The max number of connections is 200.
12	0.145212456	172.16.60.1	193.137.29.15	TCP	66	41706 → 21 [ACK] Seq=1 Ack=393 Win=64128 Len=0 TSval=2743823064 TSecr=2610028740
13	0.145269307	172.16.60.1	193.137.29.15	FTP	81	Request: user anonymous
14	0.147080170	193.137.29.15	172.16.60.1	TCP	66	21 → 41706 [ACK] Seq=393 Ack=16 Win=65280 Len=0 TSval=2610028742 TSecr=2743823064
15	0.147126545	193.137.29.15	172.16.60.1	FTP	100	Response: 331 Please specify the password.
16	0.147152316	172.16.60.1	193.137.29.15	FTP	76	Request: pass pass
17	0.151459539	193.137.29.15	172.16.60.1	FTP	89	Response: 230 Login successful.
18	0.151487616	172.16.60.1	193.137.29.15	FTP	71	Request: pasv
19	0.154249376	193.137.29.15	172.16.60.1	FTP	118	Response: 227 Entering Passive Mode (193,137,29,15,219,157).
20	0.154294983	172.16.60.1	193.137.29.15	TCP	74	60558 → 56221 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2743823073 TSecr=0 WS=128
21	0.157547523	193.137.29.15	172.16.60.1	TCP	74	56221 → 60558 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM TSval=2610028753 TSecr=2743823073 WS=128
22	0.157559256	172.16.60.1	193.137.29.15	TCP	66	60558 → 56221 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2743823076 TSecr=2610028753
23	0.157574761	172.16.60.1	193.137.29.15	FTP	92	Request: retr pub/CPAN/README.html
24	0.159578806	193.137.29.15	172.16.60.1	FTP	146	Response: 150 Opening BINARY mode data connection for pub/CPAN/README.html (8485 bytes).
25	0.160547654	193.137.29.15	172.16.60.1	FTP-DA…	6906	FTP Data: 6840 bytes (PASV) (retr pub/CPAN/README.html)
26	0.160559108	172.16.60.1	193.137.29.15	TCP	66	60558 → 56221 [ACK] Seq=1 Ack=6841 Win=60544 Len=0 TSval=2743823079 TSecr=2610028756
27	0.160578664	193.137.29.15	172.16.60.1	FTP-DA…	1711	FTP Data: 1645 bytes (PASV) (retr pub/CPAN/README.html)
28	0.160585997	172.16.60.1	193.137.29.15	TCP	66	60558 → 56221 [ACK] Seq=1 Ack=8486 Win=59008 Len=0 TSval=2743823079 TSecr=2610028756
29	0.160589349	193.137.29.15	172.16.60.1	TCP	66	56221 → 60558 [FIN, ACK] Seq=8486 Ack=1 Win=65280 Len=0 TSval=2610028756 TSecr=2743823076
30	0.160642778	172.16.60.1	193.137.29.15	TCP	66	41706 → 21 [FIN, ACK] Seq=57 Ack=582 Win=64128 Len=0 TSval=2743823080 TSecr=2610028755
31	0.160660169	172.16.60.1	193.137.29.15	TCP	66	60558 → 56221 [FIN, ACK] Seq=1 Ack=8487 Win=64128 Len=0 TSval=2743823080 TSecr=2610028756
32	0.162179371	193.137.29.15	172.16.60.1	TCP	66	56221 → 60558 [ACK] Seq=8487 Ack=2 Win=65280 Len=0 TSval=2610028758 TSecr=2743823080
33	0.162575724	193.137.29.15	172.16.60.1	FTP	90	Response: 226 Transfer complete.
34	0.162588226	172.16.60.1	193.137.29.15	TCP	54	41706 → 21 [RST] Seq=58 Win=0 Len=0
35	0.162649198	193.137.29.15	172.16.60.1	TCP	66	21 → 41706 [FIN, ACK] Seq=606 Ack=58 Win=65280 Len=0 TSval=2610028758 TSecr=2743823080
36	0.162655624	172.16.60.1	193.137.29.15	TCP	54	41706 → 21 [RST] Seq=58 Win=0 Len=0

Fig.18 - Captura de Wireshark da execução da aplicação download no tux3 com o ficheiro "README.html" no servidor "ftp.up.pt" na experiência 6

580	5.743933065	172.16.60.1	209.51.188.20	TCP	86	[TCP Window Update] 57534 → 27097 [ACK] Seq=1 Ack=912241 Win=1968896 Len=0 TSval=61076251 TSecr=1943125568 SLE=925273 SRE=1065…
581	5.743981256	209.51.188.20	172.16.60.1	FTP-DA…	2962	FTP Data: 2896 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
582	5.743989288	172.16.60.1	209.51.188.20	TCP	86	[TCP Dup ACK 496#1] 57534 → 27097 [ACK] Seq=1 Ack=912241 Win=1968896 Len=0 TSval=61076251 TSecr=1943125568 SLE=925273 SRE=1068…
583	5.744012406	209.51.188.20	172.16.60.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
584	5.744019250	172.16.60.1	209.51.188.20	TCP	86	[TCP Dup ACK 496#2] 57534 → 27097 [ACK] Seq=1 Ack=912241 Win=1968896 Len=0 TSval=61076251 TSecr=1943125568 SLE=925273 SRE=1070…
585	5.744038108	209.51.188.20	172.16.60.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
586	5.744045022	172.16.60.1	209.51.188.20	TCP	86	[TCP Dup ACK 496#3] 57534 → 27097 [ACK] Seq=1 Ack=912241 Win=1968896 Len=0 TSval=61076251 TSecr=1943125568 SLE=925273 SRE=1071…
587	5.744077149	209.51.188.20	172.16.60.1	FTP-DA…	2962	FTP Data: 2896 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
588	5.744122896	209.51.188.20	172.16.60.1	FTP-DA…	2962	FTP Data: 2896 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
589	5.744178211	209.51.188.20	172.16.60.1	FTP-DA…	4410	FTP Data: 4344 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
590	5.744226332	209.51.188.20	172.16.60.1	FTP-DA…	2962	FTP Data: 2896 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
591	5.744320339	209.51.188.20	172.16.60.1	FTP-DA…	5858	FTP Data: 5792 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
592	5.744372930	209.51.188.20	172.16.60.1	FTP-DA…	2962	FTP Data: 2896 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
593	5.744407572	209.51.188.20	172.16.60.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
594	5.744471547	209.51.188.20	172.16.60.1	FTP-DA…	7306	FTP Data: 7240 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
595	5.744534475	209.51.188.20	172.16.60.1	FTP-DA…	7306	FTP Data: 7240 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
596	5.744594748	209.51.188.20	172.16.60.1	FTP-DA…	7306	FTP Data: 7240 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
597	5.744656279	209.51.188.20	172.16.60.1	FTP-DA…	7306	FTP Data: 7240 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
598	5.744716413	209.51.188.20	172.16.60.1	FTP-DA…	2962	FTP Data: 2896 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
599	5.744723048	209.51.188.20	172.16.60.1	FTP-DA…	4410	[TCP Fast Retransmission] FTP Data: 4344 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
600	5.744739042	172.16.60.1	209.51.188.20	TCP	78	57534 → 27097 [ACK] Seq=1 Ack=923825 Win=1957376 Len=0 TSval=61076252 TSecr=1943125686 SLE=925273 SRE=1126545
601	5.744749658	209.51.188.20	172.16.60.1	FTP-DA…	1514	FTP Data: 1448 bytes (PASV) (retr gnu/gdb/gdb-5.2.1.tar.gz)
602	5.744756502	172.16.60.1	209.51.188.20	TCP	78	[TCP Dup ACK 600#1] 57534 → 27097 [ACK] Seq=1 Ack=923825 Win=1957376 Len=0 TSval=61076252 TSecr=1943125686 SLE=925273 SRE=1127…
603	5.856916673	209.51.188.20	172.16.60.1	TCP	1514	[TCP Out-Of-Order] 27097 → 57534 [ACK] Seq=923825 Ack=1 Win=65280 Len=1448 TSval=1943125799 TSecr=61076249
604	5.856952781	172.16.60.1	209.51.188.20	TCP	66	57534 → 27097 [ACK] Seq=1 Ack=1127993 Win=1929856 Len=0 TSval=61076364 TSecr=1943125799

Fig.19 - Captura de Wireshark da congestão de pacotes na execução da aplicação download no tux3 em simultâneo com o tux4 experiência 6

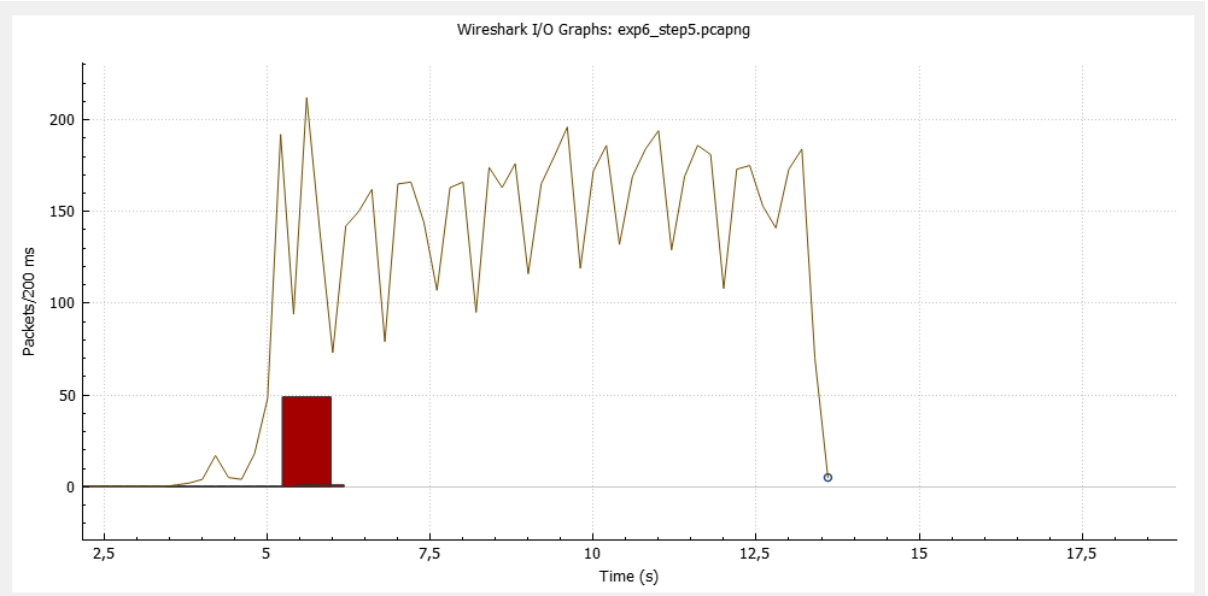

Fig.20 - Gráfico da congestão de pacotes TCP na execução da aplicação download no tux3 em simultâneo com o tux4 experiência 6