Redes de Computadores
2º Trabalho Laboratorial – Rede de computadores

Mestrado Integrado em Engenharia Informática e computação
21 de dezembro de 2020

Turma 3 grupo 7:

- **Diogo Guimarães do Rosário - up201806582**
- **Henrique Melo Ribeiro - up201806529**

Sumário

Este projeto teve como objetivo criar uma rede de computadores e um programa de download de ficheiros por FTP.

Este relatório divide-se em duas partes:

- Descrição da aplicação de download

- Configuração da rede em função das 6 experiências propostas

As experiências acima referidas consistem em configurar um **IP de rede,** um **router** em Linux, um router comercial juntamente com um sistema de **DNS** e na implementação de duas **Virtual LANs** através do switch e num teste de download usando a configuração final e a aplicação de download criada.

Introdução

Este projeto foi realizado no âmbito da disciplina Redes de Computadores com o objetivo de obter um melhor entendimento sobre o desenvolvimento de uma aplicação de download em modo FTP e sobre a criação/análise de uma rede.

O trabalho baseou-se na criação de uma rede de computadores e posterior ligação da aplicação de download desenvolvida pelo nosso grupo. Desta forma o nosso relatório estará dividido da seguinte forma:

-Parte 1 - Aplicação de download

- Arquitetura da aplicação de download

- Analise de um download com sucesso

- Parte 2 - Configuração da rede e análise:

- Arquitetura da configuração, objetivos da experiência, comandos das configurações principais

- Análise dos logs obtidos através do *Wireshark*

- Conclusões

- Anexos

Nota: Logs utilizados para as experiências 4, 5 e 6 foram obtidos pelo grupo 3 da turma 3.

Parte 1 – Aplicação de download

Para testar o bom funcionamento da rede configurada foi preciso criar uma aplicação de download capaz de criar uma conexão FTP com um endereço fornecido pelo utilizador e transferir o ficheiro especificado utilizando as normas RFC959 (leitura e analise das respostas do servidor) e RFC1738 (tratamento e utilização dos endereços URL).

De seguida iremos descrever com mais detalhe a implementação do programa bem como algumas das suas funcionalidades e algumas analises dos resultados.

Para compilar basta fazer o comando: ***gcc -Wall -o download download.c***

1.1 – Arquitetura da aplicação de download.

Para a melhor distribuição de funcionalidades o grupo decidiu dividir a aplicação em duas partes, uma em que faz o tratamento do input do utilizador e outra para realizar a conexão FTP com o servidor fornecido. O servidor é fornecido como único argumento de execução sendo que pode ou não conter também um utilizador e password para este.

O formato para execução da aplicação deve ser o seguinte:

```
ftp://[<user>:<password>@]<host>/<url-path>
```

Este argumento é tratado na função getURL(username, password, returnHost, file, argv).

Após o tratamento do input, os dados da função que são recebidos são preenchidos com os valores corretos de forma a conseguir realizar a conexão FTP.

De seguida é feita a conexão com o servidor, utilizando funções fornecidas no guião da 1ª aula **getHostName(char *host)** que é responsável por obter a informação sobre o host, como o seu IP, e um conjunto de funções como **socket** e **connect** que criam um TCP socket e conectam-no com o servidor desejado).

Após esta conexão é feito um while loop que lê a mensagem enviada pelo servidor como resposta à ligação do socket com este, sendo esta mensagem processada na função **parseConnection(char* buf**). Esta função verifica se os três primeiros bytes da mensagem correspondem ao código 220 que simboliza que a conexão foi estabelecida corretamente. Para alem disto, também verifica se, após o código 220 existe um '–'. Caso exista é sinal que o servidor ainda vai enviar mais mensagens, caso não exista significa que o servidor está à espera de um input.

Quando a função **parseConnection** indicar que a resposta do servidor terminou é utilizada a função **sendUserPass(int sockfd, char *user, char *pass)** que é responsável por enviar as credenciais ao servidor caso estes tenham sido indicados, caso não tenham sido indicados é indicado que a conexão é feita em modo anónimo.

Em primeiro lugar é enviado o utilizador para o servidor através da função **write** para o file descriptor do socket. Após a escrita do user é esperada uma resposta do servidor com o código 331. Caso este código esteja presente é enviada a password ao servidor, caso seja outro código a execução da aplicação é cancelada. Se todos os credenciais forem aceites é enviado o comando de modo passivo para o servidor, retornando o **serverPort** para a segunda conexão.

Caso o servidor consiga entrar em modo passivo é feita uma nova conexão para o download do ficheiro ser realizado, utilizando o valor de retorno da função anterior. Este download é feito na função **getFile** na qual é enviado o comando **retr** para o servidor, juntamente com o path para o ficheiro a ser transferido. Caso o ficheiro exista, é iniciado o download do ficheiro através de várias chamadas à função **read,** utilizando o file descriptor da conexão de dados como fonte da leitura.

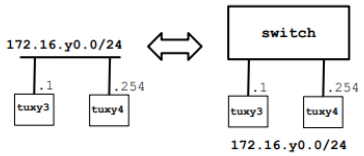
1.2 – Resultados do download.

Caso o programa tenha conseguido executar sem problemas o terminal deve indicar todas as respostas obtidas do servidor tal como algumas mensagens da aplicação que servem como forma de seguir a execução do programa e saber que passo está a executar.

```
^C
diogo@diogo-up201806582:~/Desktop/FEUP/3Ano/RCOM/Lab2$ ./download ftp://ftp.up.pt/pub
/debian/README
user: anonymous
pass:
host: ftp.up.pt
file: pub/debian/README
        220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
        220----------------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
        220-
        220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
        220-
        220-
        220
parsed starting connection
        331 Please specify the password.
        230 Login successful.
        227 Entering Passive Mode (193,137,29,15,197,240).
connected to data server-port
getting file:pub/debian/README
        150 Opening BINARY mode data connection for pub/debian/README (1190 bytes).
Finished creating file
diogo@diogo-up201806582:~/Desktop/FEUP/3Ano/RCOM/Lab2$
```

Parte 2 – Configuração da rede

2.1 – Experiência 1 – Configuração de um network IP



2.1.1 – Objetivos principais

O objetivo principal desta experiência era configurar os endereços ip dos 2 tux, de modo a que estes consigam comunicar.

2.1.2 – Comandos de configuração

Tux3 –

ifconfig eth0 172.16.y0.1/24

route add -net 172.16.y0.0/24 gw 172.16.y0.1 dev eth0

Tux4 –

ifconfig eth0 172.16.y0.254/24

route add -net 172.16.y0.0/24 gw 172.16.y0.254 dev eth0

2.1.3 – Questões

1. **What are the ARP packets and what are they used for?**

   Os ARP packets são utilizados para traduzir os endereços IP em endereços MAC

```
▾ 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_5a:7d:16 (00:21:5a:5a:7d:16)
    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
```

2. **What are the MAC and IP addresses of ARP packets and why?**

É enviado um Broadcast a “perguntar” quem é aquele endereço IP, recebendo uma respostas com o seu MAC address.

```
    13 18.750604998  HewlettP_5a:7d:16    Broadcast           ARP         42 Who has 172.16.10.254? Tell 172.16.10.1
    14 18.750744820  HewlettP_5a:7b:3f    HewlettP_5a:7d:16   ARP         60 172.16.10.254 is at 00:21:5a:5a:7b:3f
```

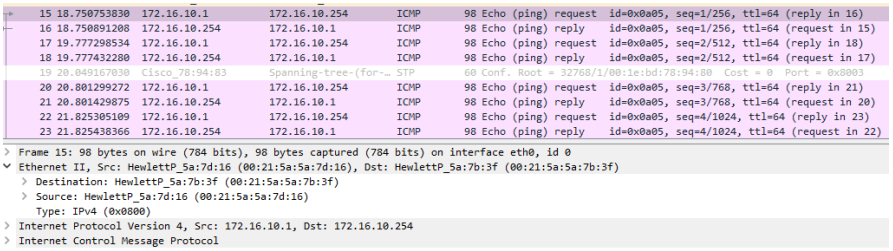
3. **What packets does the ping command generate?**

   The ping command generates ICMP packets.

```
    15 18.750753830  172.16.10.1       172.16.10.254     ICMP      98 Echo (ping) request  id=0x0a05, seq=1/256, ttl=64 (reply in 16)
    16 18.750891208  172.16.10.254     172.16.10.1       ICMP      98 Echo (ping) reply    id=0x0a05, seq=1/256, ttl=64 (request in 15)
    17 19.777298534  172.16.10.1       172.16.10.254     ICMP      98 Echo (ping) request  id=0x0a05, seq=2/512, ttl=64 (reply in 18)
    18 19.777432280  172.16.10.254     172.16.10.1       ICMP      98 Echo (ping) reply    id=0x0a05, seq=2/512, ttl=64 (request in 17)
```

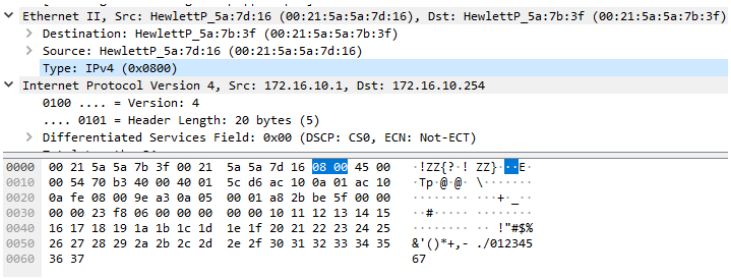
4. **What are the MAC and IP addresses of the ping packets?**

   Neste exemplo é possível verificar que o endereço MAC de destino é o endereço do tux4 e o endereço MAC source é o do tux3. Assim concluímos que o ping foi efetuado de tux3 para o tux4.

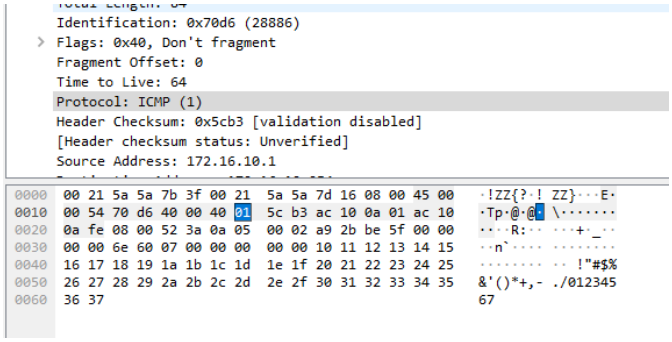


5. **How to determine if a receiving Ethernet frame is ARP, IP, ICMP?**

   Para distinguir tramas ARP de IP e ICMP é necessário analisar os bytes 12-13 da trama ethernet. Neste seguinte exemplo o valor 0x0800 representa uma trama IP.

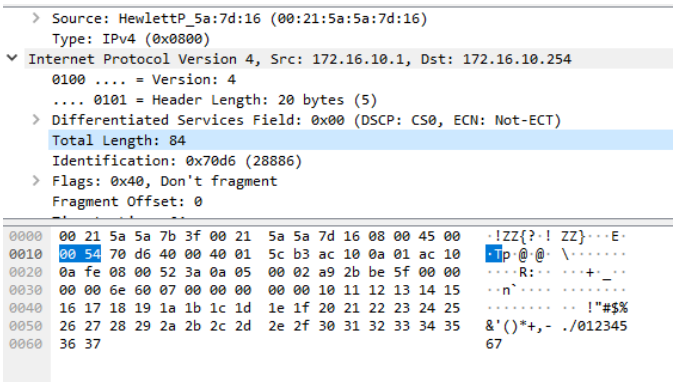


   Para distinguir as tramas IP de ICMP é possível analisar o byte 23 da trama ethernet.



6. **How to determine the length of a receiving frame?**

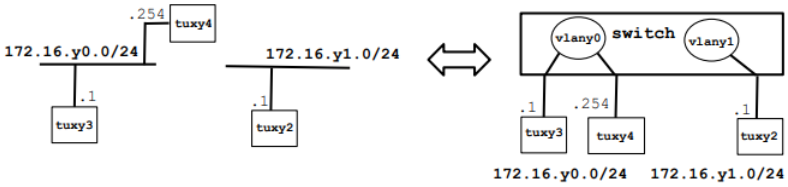
   Nas tramas IP o tamanho total da trama é representado nos bytes 16-17.



7. **What is the loopback interface and why is it importante**

A interface de loopback é um canal no qual todas as mensagens enviadas são instantaneamente recebidas. Isto é importante para verificar se a rede se encontra configurada corretamente.

2.2– Experiência 2 – Implementar duas VLANs através de um switch



2.2.1 – Objetivos principais

O objetivo principal desta experiência era configurar duas VLANs separadas e entender que estas não conseguiam comunicar entre elas.

2.2.2 – Comandos de configuração

Switch (gtkterm) –

Start by creating VLANs

Configure terminal

vlany0

end

Repeat for other VLAN

Add switch ports to respective VLAN

Configure terminal

Interface fastethernet 0/porta

Switchport mode access

Switchport access vlan y0

End

Repeat for other ports and for both VLANs

2.2.3 – Questões

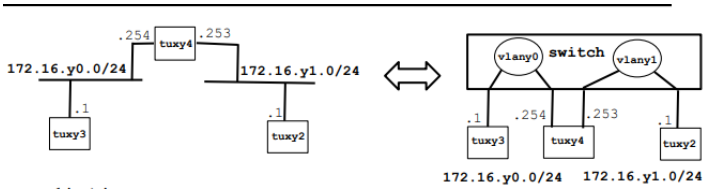
1. **How to configure vlany0?**

   Respondido em 2.2.2

2. **How many broadcast domains are there? How can you conclude it from the logs?**

   É possível concluir que existem 2 domínios de broadcast uma vez que o tux2 que está numa VLAN separada não recebe nenhum dos pings enviados pelo tux3

2.3 – Experiência 3 – Configurar um router em linux



2.3.1 – Objetivos principais

O objetivo principal desta experiência era configurar um router em Linux, de modo a conseguir comunicar entre os tux3 e tux2.

2.3.2 – Comandos de configuração

Tux2 –

ifconfig eth0 172.16.y1.1/24

route add -net 172.16.y0.0/24 gw 172.16.y1.253 dev eth0

Tux4 –

Ifconfig eth1 172.16.y1.253/24

Echo 1 > /proc/sys/net/ipv4/ip_forward

Echo 0 > /proc/sys/net/ipv4/ecmp_echo_ignore_broadcasts

2.3.3 – Questões

1. **What routes are there in the tuxes? What are their meaning?**
   Existem 2 rotas em todos os tux. Isto é, tux3 tem rota para chegar a tux4 e tux2 na outra vlan, acontecendo o oposto no tux2 e tux4 tem as 2 rotas tanto para tux3 como tux2. Estas rotas são importantes para cada computador saber para onde mandar os pacotes.
2. **What information does an entry of the forwarding table contain?**
   A tabela de routing tem informação acerca do endereço de destino, bem como o gateway e a máscara de rede. Além disso tem informação acerca da interface usada, como por exemplo, eth0.

3. **<u>What ARP messages, and associated MAC addresses, are observed and why?</u>**
   É possível observar mensagens ARP para mapear os endereços IP em endereços MAC. Isto deve-se às ARP tables terem sido limpas antes da captura dos logs pelo Wireshark.

4. **<u>What ICMP packets are observed and why?</u>**

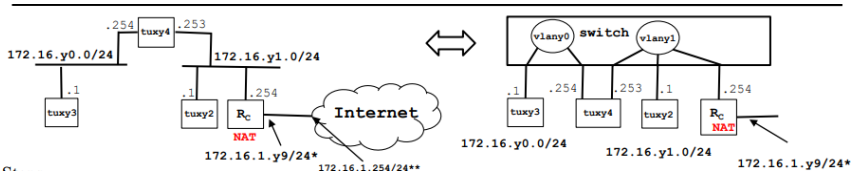
26	16.082164448	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x60af, seq=1/256, ttl=64 (reply in 27)
27	16.082328087	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x60af, seq=1/256, ttl=64 (request in 26)
28	17.091267305	172.16.10.1	172.16.11.253	ICMP	98	Echo (ping) request id=0x60af, seq=2/512, ttl=64 (reply in 29)
29	17.091401751	172.16.11.253	172.16.10.1	ICMP	98	Echo (ping) reply id=0x60af, seq=2/512, ttl=64 (request in 28)

   Neste exemplo podemos ver pings entre tux3 e tux2 com sucesso.
   Assim, concluímos que o tux4 está a servir de router entre tux3 e tux2 ao contrário da experiência anterior em que não havia conexão.

5. **<u>What are the IP and MAC addresses associated to ICMP packets and why?</u>**

   Os pacotes ICMP têm como endereço MAC associado o tux4, visto que este serve de router entre tux3 e tux2.

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



2.4.1 – Objetivos principais

O objetivo principal desta experiência era configurar um router comercial e implementar NAT de modo a ter acesso ao "exterior" da rede.

2.4.2 – Comandos de configuração

Router –

Comandos do slide 46

Tux3 –

Route add default gw 172.16.y0.254

Tux4 –

Route add default gw 172.16.y1.254

Tux2 –

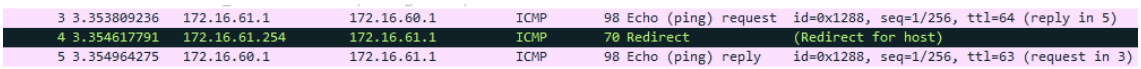
Route add default gw 172.16.y1.254

2.4.3 – Questões

1. **How to configure a static route in a commercial router?**

   Através dos seguintes comandos:
   ip route 0.0.0.0 0.0.0.0 172.16.2.254
   ip route 172.16.y0.0 255.255.255.0 172.16.y1.253

2. **What are the paths followed by the packets in the experiments carried out and why?**



   Podemos observar que os *packets* são redirecionados para o tux3 ao atingir tux4, uma vez que estes têm o *default gateway* para o endereço 172.16.y1.254. Após o primeiro, os próximos já não são redirecionados.

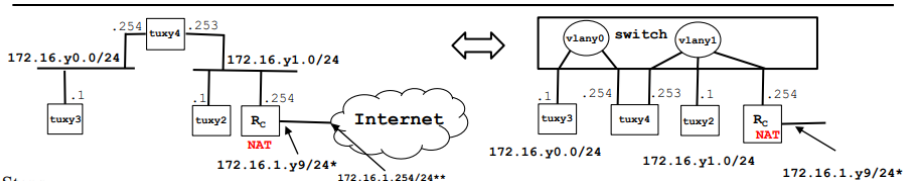
3. **How to configure NAT in a commercial router?**

   Respondido no ponto 2.4.2

4. **What does NAT do?**

   O NAT é utilizado para "reescrever" o endereço IP de origem de um pacote, de modo a que seja usado apenas 1 endereço público para vários endereços privados internos à rede.

2.5– Experiência 5 – DNS



2.5.1 – Objetivos principais

O objetivo principal desta experiência era configurar o DNS.

2.5.2 – Comandos de configuração

Alterar ficheiro /etc/resolv.conf com as seguintes linhas

search netlab.fe.up.pt

nameserver 172.16.1.1

2.5.3 – Questões

1. **How to configure the DNS service at an host?**

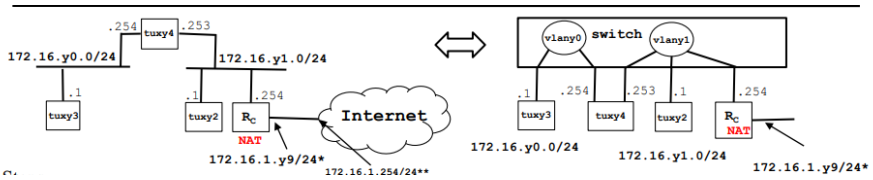
   Respondido no ponto 2.5.2

2. **What packets are exchanged by DNS and what information is transported?**

   Os pacotes de resposta DNS contêm informação sobre o domain, como por exemplo o IP.

```
> Frame 14: 291 bytes on wire (2328 bits), 291 bytes captured (2328 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_5a:79:97 (00:21:5a:5a:79:97), Dst: HewlettP_61:2d:df (00:21:5a:61:2d:df)
> Internet Protocol Version 4, Src: 172.16.2.1, Dst: 172.16.60.1
> User Datagram Protocol, Src Port: 53, Dst Port: 41242
v Domain Name System (response)
    Transaction ID: 0xc0f3
  > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 3
    Additional RRs: 5
  > Queries
  > Answers
  > Authoritative nameservers
  > Additional records
    [Request In: 12]
    [Time: 0.062847923 seconds]
```

2.6– Experiência 6 – TCP



2.6.1 – Objetivos principais

O objetivo principal desta experiência era transferir um ficheiro através do protocolo FTP.

2.6.2 – Comandos de configuração

Correr o programa de download. Ex: ./download ftp://ftp.up.pt/pub/debian/README

2.6.3 – Questões

1. **How many TCP connections are opened by your ftp application?**

   São abertas 2 conexões. A primeira de controlo que comunica com o servidor por comandos e a segunda para transferência do ficheiro

2. **In what connection is transported the FTP control information?**

A informação de controlo é transportada na 1ª conexão aberta.

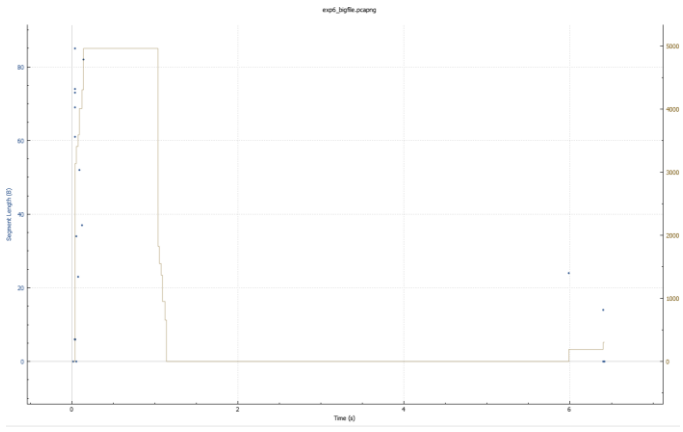
3. **What are the phases of a TCP connection?**

   A conexão TCP tem 3 fases. Através da análise dos logs é possível identificar a 1ª fase de ligação ao servidor, a 2ª após iniciar a conexão para a transferência de dados e a 3ª no fim da transferência dos dados.

4. **How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?**
   O mecanismo ARQ TCP usa respostas do tipo ACK e timeouts. Desta forma, os packets serão reenviados se não for recebido um ACK antes do tempo de timout. Usa também selective repeat, isto é, envia vários packets seguidos sem esperar resposta imediata. No cabeçalho da trama TCP é possível identificar as portas de origem e destino, número de sequência, ACK number, window e checksum.

```
Transmission Control Protocol, Src Port: 54896, Dst Port: 53122, Seq: 0, Ack: 1, Len: 0
    Source Port: 54896
    Destination Port: 53122
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 4005075128
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
```

5. **How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according the TCP congestion control mechanism?**
   O congestion control da conexão TCP é baseado no RTT.
   No seguinte gráfico podemos verificar que o throughput está de acordo com o congestion control mechanism do TCP, isto é, a cada RTT vai aumentando um "passo". Após a transmissão do ficheiro por volta do segundo 1 o throughput foi diminuindo gradualmente até que aumenta no fim com, os packets de "goodbye".



6. **Is the throughput of a TCP data connection disturbed by the appearance of a second TCP connection? How?**

   Sim. O throughput será dividido pelas 2 conexões.

Conclusão

Em suma, consideramos que os objetivos do trabalho foram atingidos. Apesar de todas as dificuldades devido ao regime de aulas adotado achamos que ficamos a compreender melhor a criação e o funcionamento de uma rede, bem como o protocolo FTP.

Anexos

```c
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <regex.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define SERVER_PORT 21
#define SERVER_ADDR "192.168.28.96"

void getURL(char *username, char *password, char *returnHost, char *file,
 char *argv);
struct hostent *getHostname(char *host);
int parseStartConnection(char* buf);
int sendUserPass(int sockfd, char * user,char * pass);
void getFile(int sockfd,int sockfdData,char * file);

// download ftp://[<user>:<password>@]ftp.up.pt/pub

int main(int argc, char **argv)
{
    struct hostent *h;
    int sockfd;
    int sockfdData;
    struct sockaddr_in server_addr;

    if (argc != 2)
    {
        perror("usage : ./download ftp://[url]");
        exit(1);
    }

    char *username = malloc(sizeof(char) * 256);
    char *password = malloc(sizeof(char) * 256);
    char *file = malloc(sizeof(char) * 256);
    char *host = malloc(sizeof(char) * 256);
    getURL(username, password, host, file, argv[1]);

    printf("user: %s\n", username);
    printf("pass: %s\n", password);
    printf("host: %s\n", host);
    printf("file: %s\n", file);

    h = getHostname(host);

    /*server address handling*/
    bzero((char *)&server_addr, sizeof(server_addr));
```

```
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(inet_ntoa(*((struct in_addr *
)h->h_addr))); /*32 bit Internet address network byte ordered*/
    server_addr.sin_port = htons(SERVER_PORT);
              /*server TCP port must be network byte ordered */

    /*open an TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket()");
        exit(0);
    }
    /*connect to the server*/
    if (connect(sockfd,
                (struct sockaddr *)&server_addr,
                sizeof(server_addr)) < 0)
    {
        perror("connect()");
        exit(0);
    }

    char buf[256];
    int bytesRead = 0;
    int readStatus = 1;
    while (readStatus)
    {
        bytesRead = read(sockfd, buf, 256);
        printf("\t");
        fflush(stdout);
        write(1, buf,bytesRead);
        readStatus = parseStartConnection(buf);
        buf[0] = '\0';
    }

    printf("parsed starting connection\n");

    int serverPort = sendUserPass(sockfd,username,password);
    if(serverPort == -1){
        printf("failed when receiving port from server");
    }

    /*server address handling*/
    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(inet_ntoa(*((struct in_addr *
)h->h_addr))); /*32 bit Internet address network byte ordered*/
    server_addr.sin_port = htons(serverPort);
              /*server TCP port must be network byte ordered */

    /*open an TCP socket*/
    if ((sockfdData = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket()");
        exit(0);
```

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

    printf("connected to data server-port\n");
    printf("getting file:%s\n",file);

    getFile(sockfd,sockfdData,file);


    free(username);
    free(password);
    free(host);
    free(file);
}

int parseStartConnection(char* buf){
    char status[4];
    strncpy(status,buf,3);
    fflush(stdout);
    if(strcmp(status,"220")){
        printf("wrong message received, not code 220");
        exit(1);
    }
    if(buf[3] != '-')
        return 0;
    return 1;
}

void getFile(int sockfd,int sockfdData,char * file){
    char * cmd = malloc(sizeof(char)*50);
    int bytesRead = 0;
    cmd[0] = '\0';
    strcat(cmd,"retr ");
    strcat(cmd,file);
    strcat(cmd,"\n");
    write(sockfd,cmd,strlen(cmd));
    char * buf = malloc(sizeof(char)*256);
    bytesRead = read(sockfd,buf,256);
    printf("\t");
    fflush(stdout);
    write(1,buf,bytesRead);
    if(strncmp(buf,"150",3)){
        printf("couldn't open binary data connection for this file\n");
    }
    else
    {
        char *token;
```

```
        token = strtok(buf, "(");
        token = strtok(NULL,")");
        char * aux_token;
        aux_token = strtok(token," ");
        int bytesToRead = atoi(aux_token);

        char * auxFile;
        char * lastToken;
        auxFile = strtok(file,"/");
        while(auxFile != NULL){
            lastToken = auxFile;
            auxFile = strtok(NULL,"/");
        }

        FILE * fileToCreate = fopen(lastToken,"wb");
        if(fileToCreate == NULL){
            printf("error opening file\n");
            exit(1);
        }
        char auxBuf[256];
        bytesRead = 0;
        int totalBytesRead = 0;
        while(totalBytesRead < bytesToRead){
            bytesRead = read(sockfdData,auxBuf,256);
            fwrite(auxBuf,sizeof(char),bytesRead,fileToCreate);
            totalBytesRead += bytesRead;
        }
        fclose(fileToCreate);
        printf("Finished creating file\n");

    }
}

int sendUserPass(int sockfd, char * user,char * pass){
    char * cmd = malloc(sizeof(char)*50);
    int bytesRead = 0;
    cmd[0] = '\0';
    strcat(cmd,"user ");
    strcat(cmd,user);
    strcat(cmd,"\n");
    write(sockfd,cmd,strlen(cmd));
    char * buf = malloc(sizeof(char)*256);
    bytesRead = read(sockfd,buf,256);
    printf("\t");
    fflush(stdout);
    write(1,buf,bytesRead);
    if(strncmp(buf,"331",3)){
        printf("couldn't specify password\n");
    }
    else
    {
        cmd[0] = '\0';
        strcat(cmd,"pass ");
        strcat(cmd,pass);
```

```
        strcat(cmd,"\n");

        write(sockfd,cmd,strlen(cmd));
        bytesRead = read(sockfd,buf,256);

        printf("\t");
        fflush(stdout);
        write(1,buf,bytesRead);
    }
    if(strncmp(buf,"230",3)){
        printf("didn't login successfully\n");
    }
    else{
        cmd[0] = '\0';
        strcat(cmd,"pasv\n");
        write(sockfd,cmd,strlen(cmd));
        bytesRead = read(sockfd,buf,256);

        printf("\t");
        fflush(stdout);
        write(1,buf,bytesRead);
    }
    if(strncmp(buf,"227",3)){
        printf("couldn't enter passive mode\n");
    }
    else{
        char *token;
        token = strtok(buf, "(");
        token = strtok(NULL,")");
        char * aux_token;
        aux_token = strtok(token,",");
        aux_token = strtok(NULL,",");
        aux_token = strtok(NULL,",");
        aux_token = strtok(NULL,",");
        aux_token = strtok(NULL,",");
        int port;
        int aux;
        port = atoi(aux_token)*256;
        aux_token = strtok(NULL,",");
        aux = atoi(aux_token);
        port += aux;
        return port;
    }
    return -1;

}

void getURL(char *username, char *password, char *returnHost, char *file,
 char *argv)
{
    regex_t regex;
    if (regcomp(&regex, "^ftp://((a-zA-Z0-9)+:(a-zA-Z0-
9)*@)?.+(/[^/]+)+/?+$", REG_EXTENDED) != 0)
    {
```

```
        perror("Url regex");
        exit(1);
    }

    int match = regexec(&regex, argv, 0, NULL, 0);
    if (match == REG_NOMATCH)
    {
        puts("invalid url");
        free(username);
        free(password);
        free(returnHost);
        free(file);
        exit(1);
    }

    regfree(&regex);

    char *token;
    token = strtok(argv, "/");
    token = strtok(NULL, "/"); //Skip ftp://
    int isHost = 0;
    char *testHost = malloc(sizeof(char) * 50);
    char *filepath = malloc(sizeof(char) * 256);
    strcpy(filepath, "");
    while (token != NULL)
    {
        if (isHost == 0)
        {
            strcpy(testHost, token);
            isHost = 1;
        }
        else
        {
            strcat(filepath, token);
            strcat(filepath, "/");
        }
        token = strtok(NULL, "/");
    }

    char *pPosition = strchr(testHost, '@');
    char *user = malloc(sizeof(char) * 50);
    char *pass = malloc(sizeof(char) * 50);
    char *host = malloc(sizeof(char) * 50);

    if (pPosition == NULL)
    {
        strcpy(user, "anonymous");
        strcpy(pass, "");
        strcpy(host, testHost);
    }
    else
    {
        char *pointsPosition = strchr(testHost, ':');
        if (pointsPosition == NULL)
```

```
        {
            perror("No division between the user and password\n");
            exit(1);
        }

        const char dividePass = '@';
        const char divideUser = ':';

        user = strtok(testHost, &divideUser);
        pass = strtok(NULL, &dividePass);
        host = strtok(NULL, &dividePass);

        if (user == NULL || pass == NULL || host == NULL)
        {
            perror("Please use the following format for the introduction
of username and password : username:password@host\n");
            exit(1);
        }
    }

    strcpy(username, user);
    strcpy(password, pass);
    strcpy(file, filepath);
    file[strlen(file) - 1] = '\0'; // remove '/' from the end of the path
    strcpy(returnHost, host);
    //free(user);
    //free(pass);
    //free(host);
    //free(filepath);
}

struct hostent *getHostname(char *host)
{
    struct hostent *h;
    if ((h = gethostbyname(host)) == NULL)
    {
        herror("gethostbyname");
        exit(1);
    }

    return h;
}
```