Faculdade de Engenharia da Universidade do Porto
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Redes de Computadores

2.º Trabalho Laboratorial
Rede de Computadores

Universidade do Porto

Faculdade de Engenharia

FEUP

Turma 2 – RC

António José Ribeiro Mendes	up201608357
Mariam Ahmed Osman Ahmed Mohamed	up201910423
Sérgio Miguel Carvalho Gonçalves	up201603271
Luís Pedro Silva Ermida Martins de Freitas	up201605641

Porto, 23 de Dezembro de 2019

Sumário

Este 2.º trabalho prático foi desenvolvido para a unidade curricular Redes de Computadores em duas partes: a 1.ª parte com o propósito de implementar um cliente FTP de forma a que seja possível descarregar ficheiros de servidores FTP e a 2.ª parte com o objetivo principal de estudo de uma rede de computadores, fazendo a sua configuração e consequente análise dos resultados obtidos.

O cliente FTP foi implementado com sucesso e todas as experiências foram realizadas e analisadas devidamente.

1. Introdução

O projeto corrente tem como principal intuito a realização de duas partes: a implementação de uma aplicação para download via FTP e o estudo de uma rede de computadores.

Na 1.ª parte o programa construído deverá transferir ficheiros de um servidor para um cliente FTP através da *Internet*. Para que tal seja possível, é necessário conhecer o protocolo FTP, assim como as mensagens necessárias para a sua concretização.

Para a 2.ª parte, foi necessário montar e configurar uma rede que permitisse ligar os computadores de uma determinada forma e estudar as consequências resultantes dessas configurações.

Pretende-se com este relatório demonstrar todas as considerações necessárias para que seja possível a criação do cliente FTP assim como os passos a realizar para preparar as várias redes de computadores apresentadas nas experiências. É a partir delas que obtemos as conclusões pretendidas.

2. Aplicação para *download*

2.1 Arquitetura

Para a implementação da aplicação de download foi necessário perceber como a troca de mensagens se processa para que se faça autenticação e posterior pedido do ficheiro a transferir. De acordo com o *RFC959* percebe-se que o programa deverá seguir a seguinte sequência:

1) Validar e guardar dados introduzidos pelo utilizador respeitantes ao *host*, *path*, *username* e *password*. Estes dois últimos, caso não existam, deverão ser definidos como *anonymous (ou none no caso da password)*.
2) Obter endereço IP em função do *host* com a função *gethostbyname (host).*
3) Abrir *socket* para conexão TCP/IP com o IP obtido e a porta para FTP (21).
4) Obter mensagem de conexão.
5) Enviar “USER *username*” e esperar resposta positiva a pedir *password*.
6) Enviar “PASS *password*” e esperar resposta positiva da autenticação.
7) Enviar “PASV” e esperar resposta positiva, contendo a porta à qual se vai ligar.

8) Abrir novo *socket* para conexão de dados com o IP obtido anteriormente e a porta obtida no passo 7 que resulta da soma *V1*256 + V2*.
9) Fazer pedido de *Retrieve* com "RETR *path*" para o *socket* de controlo e obter resposta positiva.
10) Fazer *read's* sucessivos no *socket* de dados para obter o ficheiro pedido.
11) Receber respostas finais a indicar a conclusão com sucesso da transferência de dados.

Daqui se retiram algumas conclusões acerca das funções necessárias para a implementação correta da aplicação:

```
void AddressVerifier(char *address);
```
Verificar se o endereço está no formato correto e guardar as informações correspondentes ao mesmo no que diz respeito a user,pass,protocol,nome do host, nome do ficheiro e caminho do ficheiro(passo 1)

```
int connectnow(int port,char[] SERVER_ADDR);
```
É responsável por abrir uma conexão TCP/IP com um certo servidor de endereço SERVER_ADDR e uma certa porta(port).(passo 3)

```
int Receive(int sockfd, char expected[]);
```
É responsável por receber os 3 bytes de resposta do servidor aos pedidos do cliente e verificar se estão corretos/são os esperados. Recebe um fd da socket correspondente à ligação à qual a resposta está associada e um vetor de caracteres com a resposta que esperamos receber.(passo 4)

```
int response (int socketfd,char type[], char user[],char filename[],char response
);
```
Envia uma informação de um certo type(user,pass…) para uma certa socket e verifica se esta é a esperada(chamando a função Receive)(passos 5,6,7)

Os restantes passos são efetuados na função main.

Conjugando todas estas funções consegue-se uma aplicação de download bastante eficaz.

2.2 *Downloads* bem-sucedidos

Foram testados diversos links, incluindo:

- ftp://ftp.up.pt/
- http://speedtest.tele2.net/

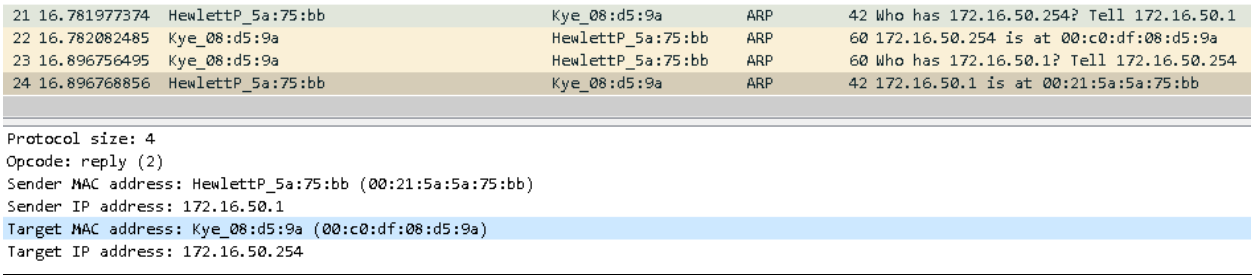
Em todos eles verificou-se que a receção era bem-sucedida e percebeu-se que a mensagem de sucesso de envio.

3. Configuração de redes e análise

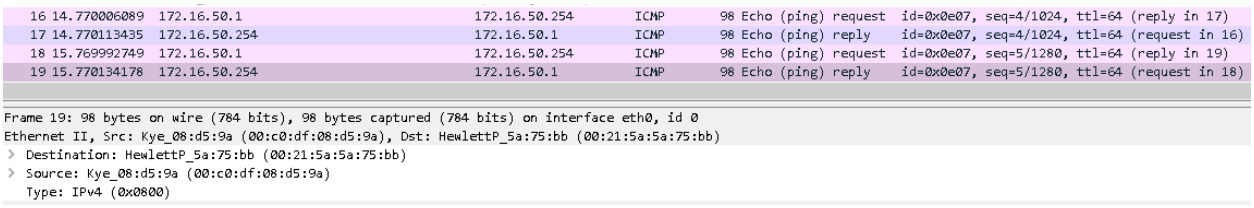
Experiência 1

Nesta primeira experiência começou-se por retirar a ligação dos computadores à rede do laboratório. De seguida, fez-se a configuração destes através do comando *ifconfig*. No final, usamos o comando “ping” para verificar a conectividade dos dois computadores configurados: *tuxy1* e *tuxy4*.

Os pacotes ARP (*Address Resolution Protocol*) são pacotes utilizados para encontrar um endereço da camada de ligação (*MAC*) a partir do endereço da camada de rede (*IP*). *MAC* (*Media Access Control*) é um endereço físico associado à interface de comunicação, que liga um dispositivo à rede enquanto que o endereço *IP* é a identificação de um determinado dispositivo numa rede. O endereço *MAC* dos pacotes *ARP* é *00:21:5a:5a:75:bb* e os endereços *IP* de origem e de destino são, respetivamente, *172.16.50.1* e *172.16.50.254*.



Os pacotes gerados pelo comando *ping* têm como endereço *MAC 00:c0:df:08:d5:9a* e *IP’s* iguais aos anteriores.



Para determinar qual o tipo de pacote recebido temos de verificar os octetos número 25 e 26 da *Ethernet frame*. Através desses octetos vemos se a *frame* corresponde a *ARP*, *IP* ou *ICMP*. Também através desses octetos é possível saber o tamanho da *frame*.

A loopback interface é uma interface de rede à qual só a própria máquina tem acesso. Possui o endereço de *IP* fixo *127.0.0.1*, no caso do *IPv4*, ou *::1*, no caso do *IPv6*. Esta interface é útil na realização de testes à stack TCP/IP mesmo que o computador não esteja ligado a nenhuma rede. Serve também para aceder mais facilmente a serviços de rede instalados na própria máquina, como por exemplo *webservers*.

Experiência 2

Depois de a primeira experiência ter sido concluída com sucesso, o objetivo da segunda experiência é a implementação de duas *virtual LANs* (*VLAN’s*) num *switch*. Nesta experiência a

vlan40 vai ser constituída pelo *tuxy1* e pelo *tuxy4* enquanto a *vlan41* vai ser constituída única e exclusivamente pelo *tuxy2*.

Antes de implementarmos as *VLAN*s considerámos de boa prática fazer *reset* às configurações do *switch* e às configurações de rede dos *tuxy*'s. Para a configuração das *VLAN*s foram utilizados os seguintes comandos:

1) *configure terminal*
2) *vlan 50* (*51* para a segunda vlan)
3) *interface fastethernet 0/P* (P - porta do switch)
4) *switchport mode access*
5) switchport *access vlan 50*
6) *end*

Nesta segunda experiência como existem dois computadores na mesma *VLAN*, e um outro computador no mesmo *switch* mas noutra *VLAN*, podemos concluir que existem dois domínios de *broadcast*.

A partir dos *logs* que capturámos no *tuxy1* e no *tuxy2* concluímos que se fizermos *ping -b 172.16.50.255,* o *tuxy1* obtém uma resposta que provem do *tuxy4*. Isto só é possível pelo facto de estarem na mesma *VLAN*.

```
 9 6.017638996   HewlettP_19:09:5c                 Broadcast           ARP         60 Who has 172.16.51.254? Tell 172.16.51.253
10 7.041662192   HewlettP_19:09:5c                 Broadcast           ARP         60 Who has 172.16.51.254? Tell 172.16.51.253
```

Mas, se no *tuxy2* colocarmos o comando *ping -b 172.16.51.255,* este não obtém qualquer resposta porque é o único computador da segunda *VLAN*.

```
13 16.981240173  172.16.51.1                       172.16.51.255       ICMP        98 Echo (ping) request  id=0x45fc, seq=1/256, ttl=64 (no response found!)
14 18.012914093  172.16.51.1                       172.16.51.255       ICMP        98 Echo (ping) request  id=0x45fc, seq=2/512, ttl=64 (no response found!)
15 18.043988357  Cisco_7b:d5:04                    Spanning-tree-(for-… STP        60 Conf. Root = 32768/51/00:1e:14:7b:d5:00  Cost = 0  Port = 0x8004
16 19.036914381  172.16.51.1                       172.16.51.255       ICMP        98 Echo (ping) request  id=0x45fc, seq=3/768, ttl=64 (no response found!)
17 20.048794981  Cisco_7b:d5:04                    Spanning-tree-(for-… STP        60 Conf. Root = 32768/51/00:1e:14:7b:d5:00  Cost = 0  Port = 0x8004
18 20.060910129  172.16.51.1                       172.16.51.255       ICMP        98 Echo (ping) request  id=0x45fc, seq=4/1024, ttl=64 (no response found!)
19 21.084921731  172.16.51.1                       172.16.51.255       ICMP        98 Echo (ping) request  id=0x45fc, seq=5/1280, ttl=64 (no response found!)
```

Experiência 3

Nesta experiência o objetivo principal é a transformação do *tuxy4* num *router*, permitindo que haja ligação entre as duas *VLANs* anteriormente criadas. Para a realização da experiência foi necessário configurar os endereços IP dos computadores, as suas rotas e as tabelas de *forwarding*. Os comandos mais utilizados para este fim foram: *route -n*, *route add*, *ifconfig* e o comando *ping* (este último para testes).

O passo fundamental nesta experiência foi a atribuição de um endereço IP ao eth1 do *tuxy4* que iria ser ligado ao *switch*, na *VLAN51*. De seguida, para que seja possível a comunicação entre as duas *VLANs* foi necessário configurar as rotas nos computadores *tuxy1* e *tuxy2*. As rotas adicionadas às tabelas destes computadores passavam a indicar que para qualquer pedido o *default gateway* (endereço através do qual deverão ser enviados os pacotes com determinado destino) era o *172.16.50.254* no caso do *tuxy1* e *172.16.51.253* no caso do

tuxy2. Para além disso foi necessário configurar as opções *ip_forward* e *ignore_broadcasts* no *tuxy4 c*om os seguintes comandos:

1) *echo 1 > /proc/sys/net/ipv4/ip_forward*
2) *echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts*

Configurando as tabelas de redirecionamento, verifica-se que por cada entrada são apresentados o *IP* de destino, o *IP* de *gateway*, a máscara do endereço, flags associadas, o custo do caminho, variáveis de estado (*ref* e *use*) e a interface utilizada. As mensagens *ARP* geradas logo após a ligação foram as seguintes:

```
21 16.781977374  HewlettP_5a:75:bb    Kye_08:d5:9a         ARP         42 Who has 172.16.50.254? Tell 172.16.50.1
22 16.782082485  Kye_08:d5:9a         HewlettP_5a:75:bb    ARP         60 172.16.50.254 is at 00:c0:df:08:d5:9a
23 16.896756495  Kye_08:d5:9a         HewlettP_5a:75:bb    ARP         60 Who has 172.16.50.1? Tell 172.16.50.254
24 16.896768856  HewlettP_5a:75:bb    Kye_08:d5:9a         ARP         42 172.16.50.1 is at 00:21:5a:5a:75:bb
```

```
11 13.536504467  HewlettP_61:2c:54                  Broadcast            ARP         42 Who has 172.16.51.1? Tell 172.16.51.253
12 13.536617399  3Com_9f:7e:9c                      HewlettP_61:2c:54    ARP         60 172.16.51.1 is at 00:01:02:9f:7e:9c
```

```
87 61.065807723  HewlettP_5a:7c:e7                  HewlettP_19:09:5c    ARP         42 Who has 172.16.51.253? Tell 172.16.51.1
88 61.065916608  HewlettP_19:09:5c                  HewlettP_5a:7c:e7    ARP         60 172.16.51.253 is at 00:22:64:19:09:5c
```

Estas mensagens aparecem porque os computadores, para comunicarem na rede *Ethernet*, precisam de conhecer quais os endereços *MAC* desses mesmos dispositivos. As mensagens *ARP* são enviadas na esperança de obter uma resposta com o endereço *MAC* pretendido.

São apresentados, de seguida, alguns pacotes *ICMP* encontrados nos *logs* da experiência:

```
11 12.771916806  172.16.50.1      172.16.50.254      ICMP      98 Echo (ping) request  id=0x0e07, seq=2/512, ttl=64 (reply in 12)
12 12.772020171  172.16.50.254    172.16.50.1        ICMP      98 Echo (ping) reply    id=0x0e07, seq=2/512, ttl=64 (request in 11)
13 13.770915143  172.16.50.1      172.16.50.254      ICMP      98 Echo (ping) request  id=0x0e07, seq=3/768, ttl=64 (reply in 14)
14 13.771022209  172.16.50.254    172.16.50.1        ICMP      98 Echo (ping) reply    id=0x0e07, seq=3/768, ttl=64 (request in 13)
```

```
30 25.220920273  172.16.50.1      172.16.51.253      ICMP      98 Echo (ping) request  id=0x0e11, seq=1/256, ttl=64 (reply in 31)
31 25.221069035  172.16.51.253    172.16.50.1        ICMP      98 Echo (ping) reply    id=0x0e11, seq=1/256, ttl=64 (request in 30)
```

```
45 32.867980280  172.16.50.1      172.16.51.1        ICMP      98 Echo (ping) request  id=0x0e15, seq=2/512, ttl=64 (reply in 46)
46 32.868208312  172.16.51.1      172.16.50.1        ICMP      98 Echo (ping) reply    id=0x0e15, seq=2/512, ttl=63 (request in 45)
47 33.866981899  172.16.50.1      172.16.51.1        ICMP      98 Echo (ping) request  id=0x0e15, seq=3/768, ttl=64 (reply in 48)
48 33.867215867  172.16.51.1      172.16.50.1        ICMP      98 Echo (ping) reply    id=0x0e15, seq=3/768, ttl=63 (request in 47)
```

Os endereços IP encontrados apresentam-se acima.

Os pacotes *ICMP* são utlizados para controlo de erros numa rede. Neste caso aparecem no *log* devido aos pedidos feitos pelo comando *ping* no *tuxy1*. Em cada par encontra-se um pedido e uma resposta. Os *MAC addresses* dos computadores *tuxy4*, *tuxy1* e *tuxy2* são, respetivamente, *00:c0:df:08:d5:9a*, *00:21:5a:5a:75:bb* e *00:01:02:9f:7e:9c.*

Experiência 4

A partir da configuração conseguida anteriormente, era necessário fazer uma extensão à *VLAN41*, que consiste em adicionar um *router* para comunicação à *Internet.* Esta configuração teria de ser feita inicialmente sem *NAT* e só depois com *NAT*.

Para configurar o *router* de forma a que sejam possível as conexões com o exterior foram realizados os seguintes passos:

1) interface fastethernet 0/0
2) ip address 172.16.51.254 255.255.255.0
3) no shutdown
4) exit
5) interface fastethernet 0/1
6) ip address 172.16.2.59 255.255.255.0
7) no shutdown
8) exit

Para além disso foi necessário redefinir as rotas para que os vários *tuxys* pudessem ter ligação ao *router* comercial.

Fazendo *ping* do *tuxy1* para o *tuxy4* o caminho seguido pelos pacotes é do *tuxy1* para o *tuxy4*. Se mandarmos um *ping* a partir do *tuxy1* para o *tuxy2*, o trajeto dos pacotes divide-se em dois caminhos: no primeiro, o pacote vai desde o *tuxy1* até ao *tuxy4*; no segundo, devido ao reencaminhamento, o pacote vai desde o *tuxy4* até ao *tuxy2*. No caso em que o *ping* é feito do *tuxy1* para o *router* comercial, o caminho dos dados é similar ao que foi referido anteriormente, só que desta vez o *router* comercial é o local de destino.

Quando o *ping* é feito do *tuxy1* para o endereço do *router* do laboratório, o trajeto seguido pelos pacotes é o mesmo que o trajeto seguido até ao router, adicionando ainda o caminho até ao router do laboratório. No entanto, como os endereços que estamos a utilizar fazem parte da gama de endereços privados, os pacotes não contêm informação suficiente para se saber quem foi o emissor que fez o pedido. Por isso, na experiência realizada, verificou-se que não há qualquer conectividade nos endereços externos à rede configurada. Só ativando corretamente a funcionalidade de *NAT* é que é possível ter acesso a toda a rede externa. O pacote, antes de ser enviado pela rede externa, terá de ser alterado. O endereço *IP* interno é substituído pelo endereço de acesso à rede criada, ou seja, *172.16.2.59* adicionando o número de porta correspondente ao computador que enviou. Posteriormente, quando chegar a resposta ao pedido anterior, o destino do pacote recebido no *router* é substituído pelo endereço *IP* interno do computador ao qual vai ser feita a entrega. Assim, resumidamente, o *NAT* permite que computadores de redes privadas possam aceder a redes externas sem necessitarem de um endereço público para cada um. Um endereço, tipicamente fornecido pelo ISP, é suficiente para que um conjunto de computadores possa comunicar com o exterior.

Para a configuração do *NAT* no *router* comercial do laboratório foram introduzidos os seguintes comandos no seu terminal:

1) *ip nat pool ovrld 172.16.2.59 172.16.2.59 prefix 24*
2) *ip nat inside source list 1 pool ovrld overload*
3) *access-list 1 permit 172.16.50.0 0.0.0.255*
4) *access-list 1 permit 172.16.51.0 0.0.0.255*

Para a configuração das rotas estáticas foram introduzidos os seguintes comandos:

1) *ip route 0.0.0.0 0.0.0.0 172.16.2.254*
2) *ip route 172.16.50.0 255.255.255.0 172.16.51.253*

Experiência 5

Esta experiência tem como principal objetivo a configuração do *DNS* (*Domain Name System* - Sistema de Nomes de Domínios), que é um sistema que obtém endereços *IP* a partir do nome do *host* e vice-versa, permitindo assim que o utilizador não precise de decorar os endereços das máquinas às quais se pretende conectar. Para a concretização dos objetivos delineados configuramos o servidor DNS em cada um dos computadores da nossa rede com o endereço *172.16.1.2 - lixa.netlab.fe.up.pt*. Para isso, foi necessário editar o ficheiro */etc/resolv.conf*, adicionando o seguinte conteúdo:

search netlab.fe.up.pt
nameserver 172.16.1.2

Os pacotes trocados pelo serviço de *DNS* aquando da captura do *log* no *Wireshark* são os seguintes:

```
 7 9.154091195   172.16.51.1                193.136.28.10        DNS        68 Standard query 0x6340 A fe.up.pt
 8 9.154104046   172.16.51.1                193.136.28.10        DNS        68 Standard query 0x6949 AAAA fe.up.pt
 9 9.156706875   193.136.28.10              172.16.51.1          DNS       244 Standard query response 0x6340 A fe.up.pt A 10.227.240.205 NS
10 9.156995465   193.136.28.10              172.16.51.1          DNS       111 Standard query response 0x6949 AAAA fe.up.pt SOA ns1.fe.up.pt
```

Verifica-se nestes pacotes o pedido *DNS* para se obter o endereço da máquina com *hostname google.pt*, tendo obtido o *IP 173.194.41.215*. O tipo de pacotes que são trocados pelo *DNS* são do tipo *IP*, que contêm a origem do pedido e o *hostname* do qual se quer obter o *IP* ou o *IP* caso se pretenda saber o *hostname* correspondente.

Experiência 6

O objetivo da experiência 6 era perceber se o cliente *FTP* tinha sido construído corretamente e se era possível descarregar dados através da rede criada ao longo das experiências anteriores. Para isso foi necessário executar o cliente *FTP* no *tuxy1* e no *tuxy2* e esperar que o ficheiro fosse transferido com sucesso, enquanto decorria a captura no *Wireshark*.

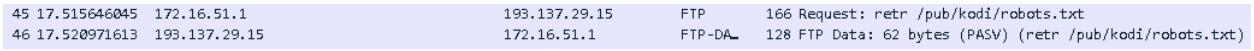
Na aplicação *FTP* construída são criadas duas ligações *TCP*: na primeira, são enviadas as informações de controlo para que seja possível a obtenção dos dados pretendidos; na segunda ligação obtêm-se os pacotes que permitem reconstruir os dados pedidos.

As ligações *TCP* são divididas em três fases: "*connection establishment*", em que é estabelecida a ligação através da troca de pacotes com dados acerca da ligação, "*transfer phase*", na qual é transferida toda a informação e "*connection termination*", que fecha os circuitos virtuais estabelecidos e liberta todos os recursos que foram alocados.

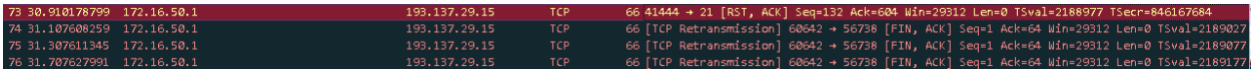
O mecanismo de *ARQ* associado ao *TCP* permite assegurar a entrega segura de dados. Uma falha faz com que haja uma retransmissão de dados. Além disso o *TCP* usa os mecanismos que o *ARQ* disponibiliza para evitar situações de congestionamento da rede. Os campos relevantes para este mecanismo são o *"sequence number", "acknowledgement number", "window size" e o "check sum"* (este ultimo é usado para correção de erros). A partir dos *logs* obtidos consegue-se visualizar facilmente os números de sequência e a troca de mensagens

geradas de forma a que o número de pacotes em processamento não exceda o tamanho máximo da janela definida, que estes não cheguem fora de ordem, que não apareçam pacotes duplicados e que haja correção de erros.

Tentamos então uma transmissão múltipla de ficheiros do servidor em dois tux's 1 e 2. Para garantir que os dois tuxs transmitiam ao mesmo tempo colocamos um sleep entre o meio da transferência de um e início da transferência do outro. Passados alguns segundos verificamos que os ficheiros tinham sido transmitidos com sucesso.



Acima podemos ver o pedido de transferência ao servidor ftp.up.pt, assim como o ficheiro pedido.



Devida à interrupção provocada pelo sleep o servidor teve de retransmitir dados.

Não chegamos a testar com ficheiros com tamanho suficiente para obter gráficos a partir dos quais conseguíamos tirar conclusões (eram de poucos KB), mas o seguinte acontece estando de acordo com o comportamento do mecanismo de controlo de congestionamento do *TCP*:

No decurso da transferência verificar-se-ia que a velocidade aumentaria até atingir um ponto em que a janela seria excedida. Nesse momento, para evitar congestionamento a velocidade diminuiria de forma a que a situação fosse resolvida.

Se fosse iniciada uma transferência no tux 2 verificar-se-ia que a velocidade no *tuxy1* diminuiria(para aproximadamente metade ou para um terço caso fossem 3 computadores), uma vez que a largura de banda é divida pelos computadores a executar a transferência.

4. Conclusões

Todas as experiências propostas para este trabalho laboratorial foram realizadas com sucesso. No final obteve-se uma rede através da qual era possível a comunicação interna entre computadores e a comunicação externa, permitindo, por exemplo, o acesso a páginas *web*.

Através da configuração da rede no laboratório foi possível perceber alguns factos teóricos importantes em redes de computadores, tais como os protocolos *Ethernet* e *TCP/IP*. Para além disso, adquiriu-se conhecimentos necessários para a configuração dos dispositivos que irão fazer parte de uma rede: computadores, *routers* e *switches*.

A realização deste trabalho laboratorial permitiu obter uma nova visão sobre redes de computadores, que cada vez mais estão presentes no nosso dia, quer no computador, quer nos dispositivos móveis.

5. Notas

Apesar de termos uma colega que é estrangeira e comunicarmos em inglês, todo este trabalho foi feito em conjunto em que traduzimos para português todas as contribuições dela.

6.Anexos

Ficheiro_download.c

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

#define SERVER_PORT 21
//#define SERVER_ADDR "192.168.28.96"
//#define SERVER_ADDR "193.137.29.15" // endereço do host name ftp.up.pt
#define MAX_STRING_LENGTH 75
#define h_addr h_addr_list[0]

int Receive(int sockfd, char expected[]);
int connectnow(int port,char[] SERVER_ADDR);
int response(int socketfd,char type[], char user[],char filename[],char response[
]);
void AddressVerifier(char *address);

char Protocol[MAX_STRING_LENGTH];
char User[MAX_STRING_LENGTH];
char Pass[MAX_STRING_LENGTH];
char Host[MAX_STRING_LENGTH];
char Path[MAX_STRING_LENGTH];
char Filename[MAX_STRING_LENGTH];

// ./download ftp://anonymous:1@speedtest.tele2.net/1KB.zip
// ./download ftp://anonymous:none@ftp.up.pt/pub/kodi/robots.txt
```

```
void AddressVerifier(char *address){
    int i=0,state=0;
    memset(Protocol,0,7);
    memset(User,0,MAX_STRING_LENGTH);
    memset(Pass,0,MAX_STRING_LENGTH);
    memset(Host,0,MAX_STRING_LENGTH);
    memset(Path,0,MAX_STRING_LENGTH);
    int j=0;
    while(state != 5){
        switch (state){
           case 0:
               strncpy(Protocol,address,6);
               if(strcmp(Protocol,"ftp://") != 0){
                   printf("Not valid protocol\n");
                }
                i=5;
                state=1;
                break;
           case 1:
               while(address[i] != ':'){
                   User[j]=address[i];
                   ++i;
                   ++j;
               }
               printf("User:%s\n",User);
               state=2;
               break;
           case 2:
               j=0;
               while(address[i] != '@'){
                 printf("%c",address[i]);
                 Pass[j]=address[i];
                   ++i;
                   ++j;
               }
               printf("Pass:%s\n",Pass);
               state=3;
               break;
           case 3:
               j=0;
               while(address[i] != '/'){
                   Host[j]=address[i];
                   ++i;
                   ++j;
```

```
                }
                printf("Host:%s\n",Host);
                state=4;
                break;
            case 4:
                 j=0;
                 Filename[j]=address[i-1];
                 j++;
                 while(address[i] != '\0'){
                    Filename[j]=address[i];
                    ++i;
                    ++j;
                }
                printf("Filename:%s\n",Filename);
                state=5;
                break;

        }
        ++i;
    }
    strcat(Path,Host);
    strcat(Path,Filename);
 }

int Receive(int sockfd, char expected[]){
    char received;
    int estado = 0, totalRead = 0;
    char c;

    while(estado != 3){
        printf("%c",c);
    switch(estado){
        case 0:
            totalRead = read(sockfd,&c,1);
            if(totalRead > 0 && c == expected[0]){
                printf("estado:%d\n",estado);
                estado=1;
            }
            else{
                estado=0;
            }
            break;
        case 1:
            totalRead = read(sockfd,&c,1);
            if(totalRead > 0 && c == expected[1]){
```

```
                printf("estado:%d\n",estado);
                estado=2;
            }
            else{
                estado=0;
            }
            break;
        case 2:
            totalRead = read(sockfd,&c,1);
            if(totalRead > 0 && c == expected[2]){
                printf("estado:%d\n",estado);
                estado=3;
            }
            else{
                estado=0;
            }
            break;
    }
    }
    return 1;
}

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

    int sockfd;
    int sockfdClient;
    struct  sockaddr_in server_addr;
    struct hostent *h;
    int bytes;

      printf("\033[0;31m");
        printf("Escreve o host name desejado(ex: ftp.up.pt)\n");
      printf("\033[0m");
        if (argc != 2) {
            fprintf(stderr,"usage: getip address\n");
            exit(1);
         }

    AddressVerifier(argv[1]);
    printf("%s\n",Host);
        // a struct h contém o host
        if ((h=gethostbyname(Host)) == NULL) {
            herror("gethostbyname");
            exit(1);
        }
```

```
    printf("\033[0;32m");
    printf("Host name  : %s\n", h->h_name);
    printf("IP Address : %s\n",inet_ntoa(*((struct in_addr *)h->h_addr)));
    printf("\033[0m");

       if(strlen(User) == 0){
        printf("Erro num dos parâmetros\n");
    }
       if(strlen(Pass) == 0){
        printf("Erro num dos parâmetros\n");
    }
       if(strlen(Path) == 0){
        printf("Erro num dos parâmetros\n");
    }
       if(strlen(Host) == 0){
        printf("Erro num dos parâmetros\n");
    }
        if(strlen(Filename) == 0){
         printf("Erro num dos parâmetros\n");
    }

    // a função conect now retorna o filedescriptor da socket a que nos conectamo
s
    sockfd = connectnow(SERVER_PORT,inet_ntoa(*((struct in_addr *)h->h_addr)));
    printf("\033[1;34m");
    printf("Conectado Com sucesso\n");
    printf("\033[0m;");
    response(sockfd,"user",User,Filename,"220");
    printf("Passwork required for euproprio\n");
    response(sockfd,"pass",Pass,Filename,"331");
    printf("User logged in\n");
    Receive(sockfd,"230");
    response(sockfd, NULL ,"pasv\r\n",Filename,"227");
    printf("Entering Passive Mode\n");

    int state = 0;
    int index = 0;
    char charResponse[2];
    memset(charResponse, 0, 2);
    char largeByte[4];
    memset(largeByte, 0, 4);
    char smallByte[4];
    memset(smallByte, 0, 4);

    while (state != 7)
```

```
{
    read(sockfd, charResponse, 1);
    charResponse[1] = '\0';
    switch (state)
    {
    case 0:
        if (charResponse[0] == '(')
        {
            state = 1;
        }
        break;
    case 1:
        if (charResponse[0] == ',')
        {
            state++;
        }
        break;
    case 2:
        if (charResponse[0] == ',')
        {
            state++;
        }
        break;
    case 3:
        if (charResponse[0] == ',')
        {
            state++;
        }
        break;
    case 4:
        if (charResponse[0] == ',')
        {

            state++;
        }
        break;
    case 5:
        if (charResponse[0] == ',')
        {
            largeByte[index] = charResponse[0];
            state++;
            index = 0;
        }
```

```
        else{
            largeByte[index] = charResponse[0];
            index++;
        }

        break;
    case 6:
        if (charResponse[0] == ')')
        {
            smallByte[index] = '\0';
            state++;
            index=0;
        }
        else
        {
            smallByte[index] = charResponse[0];
            index++;
        }
        break;

    }
}

int first = atoi(largeByte);
int second = atoi(smallByte);

int port1 =  (first * 256 + second);

printf("first = %d  second %d  port1 %d \n", first, second, port1);

sockfdClient= connectnow(port1,inet_ntoa(*((struct in_addr *)h->h_addr)));

char retr[MAX_STRING_LENGTH];
char meleon[MAX_STRING_LENGTH];
memset(retr,0,MAX_STRING_LENGTH);
memset(meleon,0,MAX_STRING_LENGTH);
int k=0;
int bars=0;

 for(int j=0;j < length;j++){
    if(Filename[j] == '/'){
        ++bars;
    }
}
```

```
   int length = strlen(Filename)-1;
   //caso o ficheiro não esteja em nenhuma pasta, não colocamos a barra antes do
próprio ficheiro
   if(bars == 1){
       char * token = strtok(Filename, "/");
       strcat(retr,"retr ");
       strcat(retr,Filename);
       strcat(retr,"\n");
   }
   else{
       strcat(retr,"retr ");
       strcat(retr,Filename);
       strcat(retr,"\n");
       char * token = strtok(Filename, "/");
   }

   printf("Filename:_%s\n",Filename);

   char * token = strtok(Filename, "/");

   int j=0;
   printf("%s\n",retr);
   write(sockfd,retr,100);

     // loop through the string to extract all other tokens
   while( token != NULL && bars > 1) {
     ++k;
     printf("bars:%d",bars);
     if(k == bars){
         strcpy(meleon,token);
     }
     token = strtok(NULL, "/");
   }

   if(bars == 1){
       strccpy(meleon,token);
   }
   printf("Ficheiro :%s\n",meleon);


   FILE *fd = fopen(meleon,"wb+");
   char mander;
   char izard[512];
   int i=0;
   while(read(sockfdClient,&mander,1) > 0){
```

```
        izard[i]=mander;
        fwrite(&izard[i],1,1,fd);
    }

    printf("Boas festas!\n");
    close(fd);
    close(sockfdClient);
    close(sockfd);
    exit(0);
}

 int connectnow(int port,char[] SERVER_ADDR){

    struct hostent* host;
    int sockfd;

    struct sockaddr_in server_addr;

    /*server address handling*/
    bzero((char*)&server_addr,sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);   /*32 bit Internet add
ress network byte ordered*/
    server_addr.sin_port = htons(port);     /*server TCP port must be network byt
e ordered */

    /*open an FTP socket*/
    if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
            perror("socket()");
            exit(0);
        }

    /*connect to the server*/
    printf("\033[1;33m");
    printf("Connecting...\n");
    printf("\033[0m");
    if(connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0){
            perror("connect()");
            exit(0);
    }

    printf("Conection Estabilished\n");
    return sockfd;

 }
```

```c
int response(int socketfd,char type[], char user[],char filename[],char response[
]){
    int i=0;
    char tosend[50];
    memset(tosend,0,50);
    char buf[50];
    memset(buf,0,50);
    if(type != NULL){
        strcat(tosend,type);//tipo de informação a enviar
        strcat(tosend," ");//tipo de informação a enviar
        strcat(tosend,user);// a propria informação
        strcat(tosend,"\n");// o enter a colocar no terminal
    }
    else{
        strcpy(tosend,user);
    }

    int totalRead = write(socketfd,tosend,strlen(tosend));
    printf("%s enviado :%s\n",type,tosend);

    // ver se a resposta está certa1
    Receive(socketfd,response);
    printf("%s correto %s\n",type,tosend);

    //esta linha de código imprime tudo o que o servidor retorna quando entras
    /*while(read(socketfd,buf,1) != 0){
        printf("%s",buf);
        ++i;
        if(buf[i] == "."){
            break;
        }
    }*/
}
```