Desenvolvimento de uma aplicação *download* e configuração de uma rede

Relatório do Trabalho Laboratorial nº2

Redes de Computadores

Catarina Justo dos Santos Fernandes - up201806610
Gustavo Sena Mendes - up201806078

Turma 6 - Grupo 12

Índice

Sumário

Este relatório tem como objetivo complementar o segundo trabalho laboratorial da Unidade Curricular de Redes de Computadores do Mestrado Integrado em Engenharia Informática e Computação.

O trabalho consistiu no desenvolvimento de uma aplicação *download* que descarrega um ficheiro seguindo o File Transfer Protocol e no estudo e configuração de uma rede privada de computadores.

Sendo assim, podemos afirmar que o trabalho foi concluído com sucesso, pois completámos todos os objetivos estabelecidos.

Introdução

O trabalho tem dois grandes objetivos, o desenvolvimento de uma aplicação *download* e a configuração e estudo de uma rede privada de computadores.

A nível do desenvolvimento da aplicação, o objetivo era desenvolver um cliente FTP que fosse capaz de comunicar com um servidor usando sockets TCP e que, dado um URL, conseguisse fazer a transferência de um ficheiro da internet. O nosso programa foi desenvolvido em C e em ambiente Linux.

Quanto à configuração, o objetivo era fazer as seis experiências descritas no guião para que no final fosse possível correr aplicação *download* na rede, a partir da criação de duas VLANs dentro de um switch.

O objetivo deste relatório é abordar os passos e a estrutura de ambas as componentes, fazendo uma análise teórica da sua arquitetura e resultados. Assim, o relatório está dividido em três secções principais:

1. **Aplicação *download***
2. **Configuração e estudo da rede**
3. **Conclusões**

Parte 1 - Aplicação *download*

1. Arquitetura da aplicação *download*

A aplicação foi desenvolvida em duas camadas: a de processamento do URL e a do cliente FTP. A primeira parte processa uma string passada como argumento com o formato:

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

e guarda os seus componentes na estrutura de dados **urlData** através da função **url_parser**. Os campos **user** e **password** não são obrigatórios e caso não estejam presentes a aplicação assume os valores “anonymous” e “password” respetivamente. Após o processamento do URL, o atributo **ip** é preenchido pela função **getIP**.

A segunda camada trata de toda a comunicação usando o protocolo FTP. Todos os comandos a serem enviados ou recebidos utilizam as funções **ftp_send_command** ou **ftp_rcv_response**. Após cada resposta, é feita uma verificação do código recebido, terminando o programa caso não seja o código esperado.

1. É chamada a função **init**, que cria uma socket e conecta-a com o servidor usando o **ip** e **port** qie recebe como argumento. Ao **port** é dado o valor 21, número da porta de controlo do protocolo FTP. De seguida, a aplicação aguarda pela resposta 220 (SERV_READY) do servidor.
2. Através da função **ftp_login** faz-se o log in com as credenciais passadas no url:
    a. Envia-se o comando USER <user> e aguarda-se pela resposta 331 (USER_LOGIN)
    b. Envia-se o comando PASS <password> e aguarda-se pela resposta 230 (PASS_LOGIN)
    c. Envia-se o comando TYPE I para ativar o modo binário e aguarda-se pela resposta 200 (BIN_READY)
3. Inicia-se o download através da função **ftp_download:**
    a. Com a função **ftp_enter_passive** pede-se ao servidor FTP para transferir os dados em modo passivo, enviando-se o comando PASV e calculando o **port** e o **ip** com a resposta. O **ip** corresponde aos primeiro 4 bytes e o **port**, assumindo o byte 5 como x e o 6 como y, calcula-se fazendo x*256 + y.
    b. Abre-se então a ligação de dados através do **ip** e do **port** calculado anteriormente.
    c. Envia-se o comando RETR <url_path> e aguarda-se pela resposta 150 (RETRV_READY).
    d. Cria-se um ficheiro onde guardar os dados lidos do servidor
    e. Lê-se os dados da ligação de dados aberta neste passo até não haverem mais bytes a ler
    f. Finalmente fecha-se o ficheiro de destino e ligação de dados
    g. Envia-se então o comando QUIT para a socket e fecha-se a mesma.

2. Resultados

A aplicação foi testada com ficheiros de vários formatos como txt, jpg, png, mp4, pdf e iso, cujos tamanhos variaram entre 672 a 714858496 Kb. Foi também testada com modo anónimo e modo não anónimo tanto no servidor de teste netlab1.fe.up.pt como no servidor da UPORTO ftp.up.pt. Na figura 1 do anexo II podemos observar o exemplo do download bem sucedido de um ficheiro .iso de 714858496 Kb.

Para além disso foram testados os casos de erro de dar um link cujo ficheiro não exista ou link inválido tendo o programa terminado com uma mensagem de erro, como previsto.

Parte 2 - Network configuration and analysis

Experiência 1 - Configuração de um endereço IP de rede

O objetivo desta experiência era configurar os endereços de IP de dois computadores (tux3 e tux4) para que pudessem comunicar entre si.

A arquitetura da rede pode ser encontrada na figura 2 do anexo II.

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

ARP (*Address Resolution Protocol*) é um protocolo da camada de rede (protocolo de comunicação) utilizado para converter um endereço IP num endereço físico, chamado DLC (*Data Link Control*). O endereço DLC diz respeito à camada de ligação de dados, sendo normalmente chamado de endereço MAC (*Media Access Control*). Por sua vez, o endereço MAC é um endereço que identifica um nó numa rede, sendo este único de hardware.

Os pacotes ARP são usados para o host obter um endereço MAC de outro host usando apenas o seu endereço IP (fazendo broadcast de um pacote para uma rede (TCP/IP) e recebendo uma resposta do host dessa rede com o endereço MAC.

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

Nos pacotes ARP estão contidos os endereços MAC e IP do transmissor e do recetor. No caso do transmissor querer descobrir o endereço MAC do recetor este será enviado com o valor “00:00:00:00:00:00”, obtendo como resposta um pacote ARP semelhante ao endereço MAC do recetor.

» ***What packets does the ping command generate?***

O comando ping gera pacotes ARP2 caso o endereço físico do recetor não esteja registado (na tabela ARP). Estando este registado, o comando ping gerará pacotes ICMP (*Internet Control Message Protocol*), sendo estes usados por dispositivos de rede para gerar mensagens de erro quando, devido a problemas de rede, não lhes é permitida a transferência de pacotes IP.

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

Os pacotes ICMP contém o endereço MAC e IP do transmissor e do recetor.

Neste caso, os pacotes enviados pelo Tux3 terão endereço de origem MAC: 172.16.60.1 e IP: 00:21:5a:61:2f:4e (correspondentes ao próprio Tux) e de destino MAC: 172.16.60.254 e IP: 00:21:5a:c5:61:bb (correspondentes ao Tux4).

No caso dos pacotes recebidos pelo Tux3 terão os endereços de origem correspondentes ao Tux4 e os de destino correspondentes ao Tux3.

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

Usando o campo *EthernetType* presente no *Ethernet header* da própria trama, podemos determinar o protocolo utilizado:

- 0x0806: ARP;
- 0x0800: IPv4 (IP version 4);
- 0x86dd: IPv6 (IP version 6);

Para determinar se se trata de ICMP é necessário analisar o IPv4 *header*, tendo o *Protocol Number* que corresponder a 0x01.

» ***How to determine the length of a receiving frame?***

O terceiro e quarto octetos do *Ethernet Header* de uma trama contém a informação relativa ao tamanho total da mesma (usando o *Wireshark* basta inspecionar o campo *Frame Length* da trama).

» ***What is the loopback interface and why is it important?***

A interface loopback é uma interface virtual da rede que permite ao computador comunicar consigo mesmo. Esta interface é usada para averiguar o estado de configuração da rede.

Experiência 2 - Implementar duas LANs virtuais num switch

O objetivo desta experiência era configurar duas LANs virtuais (vlan0 e vlan1) num switch. O tux3 e o tux4 foram associados à vlan0 e o tux2 foi associado à vlan1.

A arquitetura da rede pode ser encontrada na figura 3 do anexo II.

» ***How to configure vlan60?***

Configuração das ligações:

- Régua 1 porta T4 < > Régua 2 porta *switch console*;
- Régua 1 porta T3 < > Porta S0 do Tux usado para ligação à consola *switch* (neste caso Tux3).

Após estas configurações abrir o *GTKTerm* no tux escolhido e escrever os seguintes comandos:

- *configure terminal*
- *vlan 60*
- *end*

Em seguida, termos de associar as portas switch à vlan criada com os seguintes comando (novamente no GTKTerm):

- *configure terminal*
- *interface fastethernet 0/1* (correspondendo o 1 ao nr da porta)
- *switchport mode access*
- *switchport access vlan 60*
- *end*

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

Existem dois domínios de *broadcast*, visto que, quando o Tux3 faz broadcast, o próprio e o Tux4 recebem o ping de *broadcast* (mas não o Tux2). Quando o Tux2 faz broadcast nenhum dos outros Tuxs recebem o ping de *broadcast*. São assim os dois domínios: o que contém o Tux3 e o Tux4 e o que contém o Tux2.

Experiência 3 - Configuração de um router em Linux

O objetivo desta experiência era criar configurar o tux4 como um router e, assim, estabelecer uma ligação entre as duas vlans criadas anteriormente.

A arquitetura da rede pode ser encontrada na figura 4 do anexo II.

» ***What routes are there in the tuxes? What are their meaning?***

Rotas do Tux3:

- 192.16.60.0 (vlan60) pela gateway 192.16.60.1
- 192.16.61.0 (vlan61) pela gateway 192.16.60.254 (ligação ao Tux4)

Rotas do Tux2:

- 192.16.60.0 (vlan60) pela gateway 192.16.61.253 (ligação ao Tux4)
- 192.16.61.0 (vlan61) pela gateway 192.16.61.1

Rotas do Tux4:

- 192.16.60.0 (vlan60) pela gateway 192.16.60.254
- 192.16.61.0 (vlan61) pela gateway 192.16.61.253

Estas rotas são a ligação entre os diferentes tuxes e as vlans. O Tux4 está ligado a ambas as vlans então age como intermediário para a ligação do Tux3 com a vlan61 e do Tux2 com a vlan60 (tal como representado na figura acima)

» ***What information does an entry of the forwarding table contain?***

A tabela contém a seguinte informação:

- *Destination*: destino da rota
- *Gateway*: IP do ponto onde a rota passará a seguir
- *Netmask*: junto com o campo *Destination* permite determinar o ID da rede
- *Flags*: informação sobre a rota
- *Metrix*: custo de cada rota
- *Interface*: placa de rede responsável pela *gateway* (neste caso *eth0* ou *eth1*)

» ***What ARP messages, and associated MAC addresses, are observed and why?***

Quando um tux dá ping a outro, o tux recetor que não conhece o MAC Address do que enviou o ping envia uma mensagem ARP para “perguntar” qual é esse tal MAC Address.

Essa mensagem vai ter o MAC Address do tux de origem associado 00:00:00:00:00:00 (mensagem enviada em modo de broadcast), pois ainda não sabe qual o tux de destino. De seguida, o tux de destino responde uma mensagem ARP a dizer o seu MAC Address.

```
30 28.679814547  HewlettP_61:2f:4e    HewlettP_c5:61:bb    ARP         42 Who has 172.16.60.254? Tell 172.16.60.1
31 28.679947454  HewlettP_c5:61:bb    HewlettP_61:2f:4e    ARP         60 172.16.60.254 is at 00:21:5a:c5:61:bb
32 28.724950788  HewlettP_c5:61:bb    HewlettP_61:2f:4e    ARP         60 Who has 172.16.60.1? Tell 172.16.60.254
33 28.724958750  HewlettP_61:2f:4e    HewlettP_c5:61:bb    ARP         42 172.16.60.1 is at 00:21:5a:61:2f:4e
34 28.743860599  172.16.60.1          172.16.60.254        ICMP        98 Echo (ping) request  id=0x11e9, seq=6/1536, ttl=64 (r
35 28.744001050  172.16.60.254        172.16.60.1          ICMP        98 Echo (ping) reply    id=0x11e9, seq=6/1536, ttl=64 (r
36 29.767851469  172.16.60.1          172.16.60.254        ICMP        98 Echo (ping) request  id=0x11e9, seq=7/1792, ttl=64 (r
37 29.767982142  172.16.60.254        172.16.60.1          ICMP        98 Echo (ping) reply    id=0x11e9, seq=7/1792, ttl=64 (r
```

Figura 5 - tux3 pings tux4 and after that tux4 pings tux3

Na figura podemos observar o tux3 (IP: 172.16.60.1 e MAC: 00:21:5a:61:2f:4e) a perguntar a MAC adress ao tux4 (IP: 172.16.60.254 e MAC:00:21:5a:c5:61:bb) e a mesma pergunta inversamente, ou seja, do tux4 para o tux3.

» ***What ICMP packets are observed and why?***

São observadas tramas ICMP de *request* e *reply*, devido a após a ligação de todos os tuxs, estes encontram se visíveis. Se não se encontrassem visíveis, as tramas ICMP enviadas seriam de *Host Unreachable.*

» ***What are the IP and MAC addresses associated to ICMP packets and why?***

Os endereços IP e MAC associados com as tramas ICMP são os dos tuxs de origem e destino. EX: Tux3 dá *ping* ao Tux4 os endereços de origem vão ser MAC: 172.16.60.1 e IP: 00:21:5a:61:2f:4e e de destino MAC: 172.16.60.254 e IP: 00:21:5a:c5:61:bb.

Experiência 4 - Configuração de um router comercial e implementação de NAT

O objetivo desta experiência era, primeiro, configurar um router comercial sem NAT, ligando-o à rede do laboratório e, depois disso, adicionar a funcionalidade NAT ao router.

A arquitetura da rede pode ser encontrada na figura 6 do anexo II.

» ***How to configure a static route in a commercial router?***

Para configurar o router é necessário fazer as seguintes ligações:

- Régua 1 porta T4 < > Régua 2 porta *router console*
- Régua 1 porta T3 < > S0 Tux3

Comandos no *GTKTerm* do Tux3 (Tux ligado ao *router*):

- *configure terminal*
- *ip route 0.0.0.0 0.0.0.0 172.16.1.254*
- *end*

Estes comandos fazem com que qualquer pacote com destino a 192.16.60.0 com a máscara 255.255.255.0 terá que passar pela *gateway* 172.16.61.253.

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

Se for possível utilizar uma rota existente, os pacotes utilizam essa. Caso não seja possível, estes são redirecionados para a rota *default* (neste caso a do *router*). A partir deste momento os pacotes utilizam o Tux3 para chegar ao destino.

» ***How to configure NAT in a commercial router?***

Para configurar o router é necessário configurar a interface interna do processo NAT, usando, para tal, os seguintes no *GTKTerm* do Tux3 (tux ligado ao *router*):

- *configure terminal*
- *ip nat pool ovrld 172.16.1.69 172.16.1.69 prefix 24*
- *ip nat inside source list 1 pool ovrld overload*
- *access-list 1 permit 172.16.60.0 0.0.0.7*
- *access-list 1 permit 172.16.61.0 0.0.0.7*
- *end*

Sendo ainda necessário reconfigurar as rotas ip

- *configure terminal*
- *interface gigabitethernet 0/0 **
- *ip address 172.16.61.254 255.255.255.0*
- *no shutdown*
- *ip nat inside*
- *exit*

- *configure terminal*
- *interface gigabitethernet 0/1 **
- *ip address 172.16.1.69 255.255.255.0*
- *no shutdown*
- *ip nat inside*
- *exit*

» ***What does NAT do?***

O NAT(Network Address Translation) é um método de remapeamento de um endereço IP, que tem como objetivo a conservação do mesmo, permitindo, assim, que redes IP privadas usem endereços IP não registados para se conectarem à Internet ou a uma rede pública. O NAT opera num router, onde conecta duas redes, traduzindo endereços privados (na rede interna) para endereços legais, antes que as tramas sejam encaminhadas para a outra rede. NAT pode também ser implementado em ambientes de acesso remoto, oferecendo também funções de segurança.

Experiência 5 - DNS

O objetivo desta experiência era configurar o DNS nos tux3, tux4 e tux2 para conseguir aceder a redes externas, consequentemente à internet.

A arquitetura da rede pode ser encontrada na figura 7 do anexo II.

» ***How to configure the DNS service at an host?***

Para configurar um serviço DNS num host é necessário aceder ao ficheiro *resolv.conf*, responsável pela configuração de *name servers* de DNS, que está presente na pasta etc. (em Linux).

No nosso caso usamos o comando: *echo $'search netlab.fe.up.pt\nnameserver 172.16.1.1' > /etc/resolv.conf*

Este comando faz com que o computador use o domínio desejado (neste caso *netlab.fe.up.pt*) nos websites que serão acedidos posteriormente.

» ***What packets are exchanged by DNS and what information is transported***

O Tux (host) envia, para o server com o hostname desejado, uma trama pedindo o seu endereço IP, recebendo em seguida a resposta do servidor com o endereço pedido:

No.	Time	Source	Destination	Protocol	Length	Info
43	23.105054804	172.16.60.1	193.137.29.15	ICMP	98	Echo (ping) request id=0x1dc6, seq=9/2304, ttl=64 (reply in 44)
44	23.106903216	193.137.29.15	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dc6, seq=9/2304, ttl=57 (request in 43)
46	24.106992504	172.16.60.1	193.137.29.15	ICMP	98	Echo (ping) request id=0x1dc6, seq=10/2560, ttl=64 (reply in 47)
47	24.109150033	193.137.29.15	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dc6, seq=10/2560, ttl=57 (request in 46)
53	33.722103815	172.16.60.1	172.16.1.1	DNS	70	Standard query 0xc652 A google.com
54	33.722113523	172.16.60.1	172.16.1.1	DNS	70	Standard query 0x455b AAAA google.com
55	33.724403470	172.16.1.1	172.16.60.1	DNS	334	Standard query response 0xc652 A google.com A 216.58.211.46 NS ns2.google.com NS ns4.google.com NS ns3.google.com NS ns1.google.com A 216.239.32.10 AAAA 2001:4860:4802:32::a A 216.239.34…
56	33.756708221	172.16.1.1	172.16.60.1	DNS	346	Standard query response 0x455b AAAA google.com AAAA 2a00:1450:4003:802::200e NS ns4.google.com NS ns2.google.com NS ns1.google.com NS ns3.google.com A 216.239.32.10 AAAA 2001:4860:4802:3…
57	33.756991775	172.16.60.1	216.58.211.46	ICMP	98	Echo (ping) request id=0x1dd3, seq=1/256, ttl=64 (reply in 58)
58	33.770033775	216.58.211.46	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dd3, seq=1/256, ttl=112 (request in 57)
59	33.770929806	172.16.60.1	172.16.1.1	DNS	86	Standard query 0xb7de PTR 46.211.58.216.in-addr.arpa
60	33.772137008	172.16.1.1	172.16.60.1	DNS	469	Standard query response 0xb7de PTR 46.211.58.216.in-addr.arpa PTR muc03s14-in-f14.1e100.net PTR muc03s14-in-f46.1e100.net PTR mad08s05-in-f14.1e100.net NS ns4.google.com NS ns3.google.co…
62	34.758217237	172.16.60.1	216.58.211.46	ICMP	98	Echo (ping) request id=0x1dd3, seq=2/512, ttl=64 (reply in 63)
63	34.771458883	216.58.211.46	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dd3, seq=2/512, ttl=112 (request in 62)
64	35.759552349	172.16.60.1	216.58.211.46	ICMP	98	Echo (ping) request id=0x1dd3, seq=3/768, ttl=64 (reply in 65)
65	35.772697335	216.58.211.46	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dd3, seq=3/768, ttl=112 (request in 64)
67	36.760778090	172.16.60.1	216.58.211.46	ICMP	98	Echo (ping) request id=0x1dd3, seq=4/1024, ttl=64 (reply in 68)
68	36.773967565	216.58.211.46	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dd3, seq=4/1024, ttl=112 (request in 67)
69	37.762052021	172.16.60.1	216.58.211.46	ICMP	98	Echo (ping) request id=0x1dd3, seq=5/1280, ttl=64 (reply in 70)
70	37.775198125	216.58.211.46	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dd3, seq=5/1280, ttl=112 (request in 69)
72	38.763279057	172.16.60.1	216.58.211.46	ICMP	98	Echo (ping) request id=0x1dd3, seq=6/1536, ttl=64 (reply in 73)
73	38.776449148	216.58.211.46	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dd3, seq=6/1536, ttl=112 (request in 72)
74	39.764530950	172.16.60.1	216.58.211.46	ICMP	98	Echo (ping) request id=0x1dd3, seq=7/1792, ttl=64 (reply in 75)
75	39.777651492	216.58.211.46	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1dd3, seq=7/1792, ttl=112 (request in 74)

Figura 8 - pacotes enviados pelo DNS (para google.com) e resposta recebida

Experiência 6 - Ligações TCP

O objetivo desta experiência foi observar o comportamento do protocolo TCP utilizando a aplicação *download* desenvolvida.

A arquitetura da rede pode ser encontrada na figura 9 do anexo II.

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

A aplicação FTP abre 2 conexões TCP: um para enviar comandos ao servidor (e receber a resposta do mesmo) e outra para receber os dados do servidor (e enviar as respostas ao cliente) .

» ***In what connection is transported the FTP control information?***

O controlo de informação é transportado na primeira conexão TCP, a responsável pela troca de comandos.

» ***What are the phases of a TCP connection?***

São 3 as fases duma conexão TCP:

- Estabelecimento da conexão;
- Troca de dados;
- Terminação da conexão.

» ***How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?***

O protocolo TCP (*Transmission Control Protocol*) utiliza o ARQ (*Automatic Repeat Request*) em conjunto com o SWP (*Sliding Window Protocol*), um método de controlo de erros de transmissão de dados que se baseia na leitura e confirmação de mensagens enviadas pelo recetor a indicar que recebeu a trama corretamente (*acknowledgement*). Ao atribuir a cada *byte* enviado um *sequence number*, permite que o recetor os ordene, descarte os duplicados e identifique os que faltam, sendo este *sequence number* restringido por um teto máximo fixo para evitar uma congestão na rede (*window size*).

» ***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 TCP mantém uma estimativa do número de octetos que a rede consegue encaminhar, não enviando mais do que o essa estimativa nem mais do que o máximo definido pelo recetor.

» ***Is the throughput of a TCP data connections disturbed by the appearance of a second TCP connection? How?***

Com o aparecimento de uma segunda conexão TCP, a existência de uma transferência de dados em simultâneo poderá levar a um decréscimo da taxa de transmissão (pois esta é distribuída de igual forma para cada ligação).

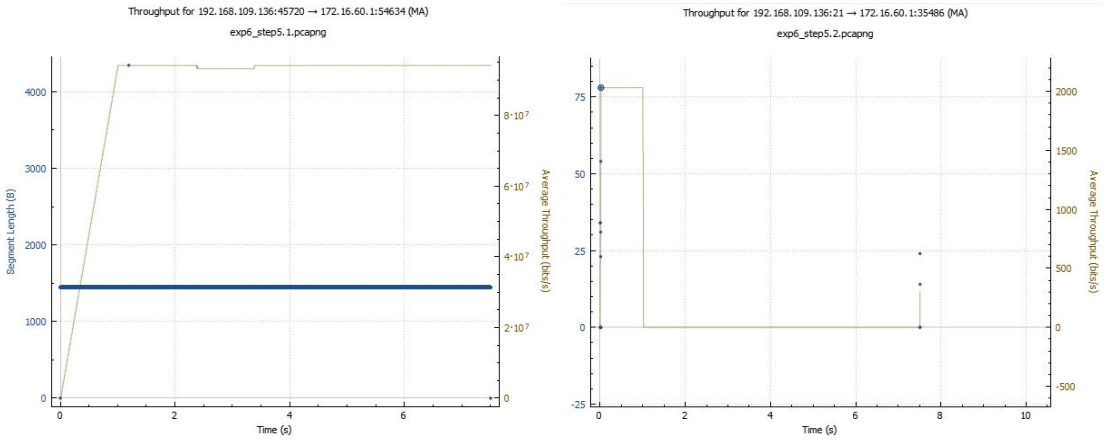

Figura 10 e 11 - diferença de bitrate de download de um ficheiro quando se começa a transferência do segundo

Conclusões

Este trabalho fez-nos pôr em prática a matéria lecionada na Unidade Curricular, permitindo-nos assimilar melhor as temáticas nela abordadas. O desenvolvimento da aplicação *download* permitiu-nos perceber melhor como funciona o protocolo FTP e ao fazer a configuração conseguimos ganhar algum conhecimento base sobre a forma como as redes de computadores operam.

Apesar das limitações ao acesso dos laboratórios impostas pela pandemia COVID-19 conseguimos concluir as experiências necessárias para a realização do trabalho na sua totalidade.

Anexos

Anexo I - código fonte da aplicação *download*

macros.h

```c
#pragma once

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

#define PORT_FTP 21
#define SERV_READY 220
#define USER_LOGIN 331
#define PASS_LOGIN 230
#define BIN_READY 200
#define PASV_READY 227
#define RETRV_READY 150
#define FILE_READY 226
```

download.h

```c
#include "macros.h"



typedef struct {
    char user[256];
    char password[256];
    char url_host[256];
    char url_path[256];
    char host_name[256];  // Host Name from gethostbyname()
    char ip[256];
} urlData;

int url_parser(char *url, urlData *url_object);
```

```
int getIP(char host[], urlData *url_object);
```

download.c

```
#include "download.h"


int url_parser(char *url, urlData *url_object){
    char *ftp = strtok(url, "/");
    char *args = strtok(NULL, "/");
    char *url_path = strtok(NULL, "");

    if (ftp == NULL) {
        perror("url should start with ftp");
        return -1;
    }
    if (args == NULL) {
        perror("No <host> or [<user>:<password>@]<host> declared");
        return -1;
    }
    if (url_path == NULL){
        perror("<url-path> is null");
        return -1;
    }

    char *user = strtok(strdup(args), ":");
    char *password  = strtok(NULL, "@");
    char *host = strtok(NULL, " ");

    if ((user != NULL) && (password != NULL)){
        strcpy(url_object->user, user);
        strcpy(url_object->password, password);
        strcpy(url_object->url_host, host);
    }
    else {
        user = "anonymous";
        password = "password";
        host = args;

        strcpy(url_object->user, user);
        strcpy(url_object->password, password);
        strcpy(url_object->url_host, host);
    }
    strcpy(url_object->url_path, url_path);
```

```
    printf("\n -- URL OBJECT -- \n");
    printf("User: %s\n", url_object->user);
    printf("Password: %s\n", url_object->password);
    printf("Host: %s\n", url_object->url_host);
    printf("URL path: %s\n", url_object->url_path);

    return 0;
}


int getIP(char host[], urlData *url_object) {
    struct hostent *h;

    /*
    struct hostent {
        char    *h_name;    Official name of the host.
        char    **h_aliases;    A NULL-terminated array of alternate
names for the host.
        int     h_addrtype; The type of address being returned; usually
AF_INET.
        int     h_length;   The length of the address in bytes.
        char    **h_addr_list;  A zero-terminated array of network
addresses for the host.
        Host addresses are in Network Byte Order.
    };
    #define h_addr h_addr_list[0]   The first address in h_addr_list.
    */

    if ((h = gethostbyname(host)) == NULL) {
        herror("gethostbyname");
        exit(1);
    }

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

    strcpy(url_object->host_name,  h->h_name);
    strcpy(url_object->ip, inet_ntoa(*((struct in_addr *)h->h_addr)));

    return 0;
}
```

```c
int main(int argc, char *argv[])
{
    //checks correct usage
    if (argc != 2) {
        perror("usage: download
ftp://[<user>:<password>@]<host>/<url-path>");
        return 1;
    }


    //gets url data
    urlData url_object;
    url_parser(argv[1], &url_object);
    getIP(url_object.url_host, &url_object);


    // inits
    int socketfd;
    if (init(url_object.ip, 21, &socketfd) != 0){
        perror("Error: init()");
        return 1;
    }
    int response = ftp_rcv_response(socketfd);
    if(response != SERV_READY){
        perror("Received Bad Response");
        socket_close(socketfd);
        return 1;
    }


    // logs in
    if(ftp_login(socketfd, url_object.user, url_object.password) != 0){
        perror("Error logging in");
        return 1;
    }


    //downloads file
    if(ftp_download(socketfd, url_object.url_path) != 0){
        perror("Error downloading file");
        return 1;
```

```
    }


    return 0;
}
```

connection.h

```
#include "macros.h"

int init(char *ip, int port, int *socketfd);
int ftp_rcv_command(int socketfd);
int ftp_send_command(int sockfd, char *msg);
int ftp_login(int socketfd, char *username, char *password);
static int ftp_enter_passive(int socketfd, char *ip, int *port);
int ftp_download(int socketfd, char *url_path);
int socket_close(int sockfd);
```

connection.c

```
#include "connection.h"



//Inits connection
int init(char *ip, int port, int *socketfd){
    struct sockaddr_in servaddr;

    bzero(&servaddr, sizeof(servaddr));

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

    // socket create and varification
    if ((*socketfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
        perror("Error creating socket");
        return 1;
    }

    // connects to the server
```

```
    if(connect(*socketfd, (struct sockaddr *)&servaddr,
sizeof(servaddr)) < 0){
        perror("Connection with the server failed");
        return 1;
    }

    return 0;
}


//Receives a response
int ftp_rcv_response(int socketfd){
    FILE* f = fdopen(socketfd,"r");
    char *buf;
    size_t bytes_read = 0;
    int response;

    while(getline(&buf,&bytes_read,f)>0){
        printf("%s",buf);
        if(buf[3]==' '){
            sscanf(buf,"%d",&response);
            break;
        }
    }

    return response;
}


//Sends a command
int ftp_send_command(int sockfd, char *msg) {
    int b, len=strlen(msg);

    if((b=write(sockfd,msg,len))!=len){
        printf("Couldn't write to socket");
        return 1;
    }

    return 0;
}


//Logs on Server
```

```c
int ftp_login(int socketfd, char *username, char *password) {
    int ret;
    char cmd_send[1024];


    //Send USER
    sprintf(cmd_send, "USER %s\r\n", username);
    ret = ftp_send_command(socketfd, cmd_send);
    if(ret != 0)
    {
        printf("ret is %d\n", ret);
        perror("Couldn't send USER command");
        socket_close(socketfd);
        return 1;
    }
    ret = ftp_rcv_response(socketfd);
    if(ret != USER_LOGIN && ret != PASS_LOGIN)
    {
        perror("Couldn't log in user");
        socket_close(socketfd);
        return 1;
    }


    //Send PASS
    sprintf(cmd_send, "PASS %s\r\n", password);
    ret = ftp_send_command(socketfd, cmd_send);
    if(ret != 0)
    {
        perror("Couldn't send PASS command");
        socket_close(socketfd);
        return 1;
    }
    ret = ftp_rcv_response(socketfd);
    if(ret != PASS_LOGIN)
    {
        perror("Couldn't send log in with password");
        socket_close(socketfd);
        return 1;
    }


    //Set to binary mode
```

```
    ret = ftp_send_command(socketfd, "TYPE I\r\n");
    if(ret != 0)
    {
        perror("Couldn't send binary mode command");
        socket_close(socketfd);
        return 1;
    }
    ret = ftp_rcv_response(socketfd);
    if(ret != BIN_READY)
    {
        perror("Couldn't set binary mode");
        socket_close(socketfd);
        return 1;
    }


    return 0;
}


//Sets FTP server to passive mode
static int ftp_enter_passive(int socketfd, char *ip, int *port)
{
    int ret, a,b,c,d,pa,pb;
    char *find, *buf;

    ret = ftp_send_command(socketfd, "PASV\r\n");
    if(ret != 0)
    {
        perror("Couldn't send PASV command");
        socket_close(socketfd);
        return 1;
    }

    //Receives response
    FILE* f = fdopen(socketfd,"r");
    size_t bytes_read = 0;
    while(getline(&buf,&bytes_read,f)>0){
        printf("%s",buf);
        if(buf[3]==' '){
            sscanf(buf,"%d",&ret);
            break;
        }
```

```
    }
    if(ret != PASV_READY)
    {
        perror("Couldn't set passive mode");
        socket_close(socketfd);
        return 1;
    }

    //Calculates port
    find = strrchr(buf, '(');
    sscanf(find, "(%d,%d,%d,%d,%d,%d)", &a, &b, &c, &d, &pa, &pb);
    sprintf(ip, "%d.%d.%d.%d", a, b, c, d);
    *port = pa * 256 + pb;

    return 0;
}


//Download Files
int ftp_download(int socketfd, char *url_path) {
    int ret, port, data_socketfd;
    char ip[64], cmd_send[1024];


    //Queries data address
    ret = ftp_enter_passive(socketfd, ip, &port);
    if(ret != 0)
    {
        perror("Could not enter passive mode");
        socket_close(socketfd);
        return 1;
    }


    //Inits data socket connection
    ret = init(ip, port, &data_socketfd);
    if(ret != 0)
    {
        perror("failed to init data port\r\n");
        socket_close(socketfd);
        return 1;
    }
```

```
//Ready to download
sprintf(cmd_send, "RETR %s\r\n", url_path);
ret = ftp_send_command(socketfd, cmd_send);
if(ret != 0)
{
    perror("failed to send RETR command");
    socket_close(data_socketfd);
    socket_close(socketfd);
    return 1;
}
ret = ftp_rcv_response(socketfd);
if(ret != RETRV_READY)
{
    perror("failed to open binary mode connection");
    socket_close(data_socketfd);
    socket_close(socketfd);
    return 1;
}


//Downloads
int file_fd, bytes_read;
char url_path_copy[1024], buf[1024], *filename;
strcpy(url_path_copy, url_path);

char *token = strtok(url_path_copy, "/");
while (token != NULL){
    filename = token;
    token = strtok(NULL, "/");
}

if ((file_fd = open(filename, O_WRONLY | O_CREAT, 0777)) < 0) {
    perror("failed to open file");
    socket_close(data_socketfd);
    socket_close(socketfd);
    return 1;
}
while((bytes_read = read(data_socketfd, buf, 1024)) > 0) {
    if (write(file_fd, buf, bytes_read) < 0){
        perror("Error while writing downloaded data to file");
        return 1;
    }
```

```
    }
    if (close(file_fd) < 0) {
        perror("Error closing file");
        return 1;
    }

    socket_close(data_socketfd);

    ret = ftp_rcv_response(socketfd);
    if (ret != FILE_READY) {
        perror("Couldn't retrieve file");
        socket_close(socketfd);
        return 1;
    }


    //QUIT
    sprintf(cmd_send, "QUIT \r\n");
    ret = ftp_send_command(socketfd, cmd_send);
    if (ret != 0) {
        perror("Couldn't send QUIT command");
        socket_close(socketfd);
        return 1;
    }
    socket_close(socketfd);


    return 0;
}


int socket_close(int sockfd) {
    close(sockfd);
    return 0;
}
```

Anexo II - Figuras relevantes

```
 -- URL OBJECT --
User: anonymous
Password: password
Host: ftp.up.pt
URL path: pub/linuxmint/stable/10/linuxmint-10-gnome-oem-i386.iso
Host name  : mirrors.up.pt
IP Address : 193.137.29.15

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
331 Please specify the password.
230 Login successful.
200 Switching to Binary mode.
227 Entering Passive Mode (193,137,29,15,205,209).
150 Opening BINARY mode data connection for pub/linuxmint/stable/10/linuxmint-10-gnome-oem-i386.iso (714858496 bytes).
226 Transfer complete.
```

Figura 1 - Output da aplicação de download ao transferir com sucesso um ficheiro .iso

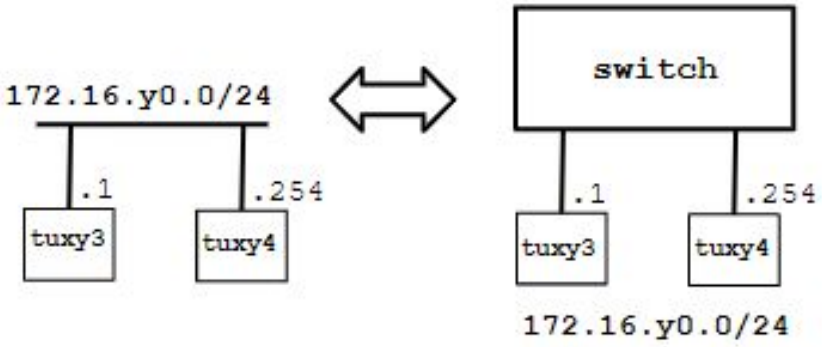

Figura 2 - Arquitetura da rede na experiência 1

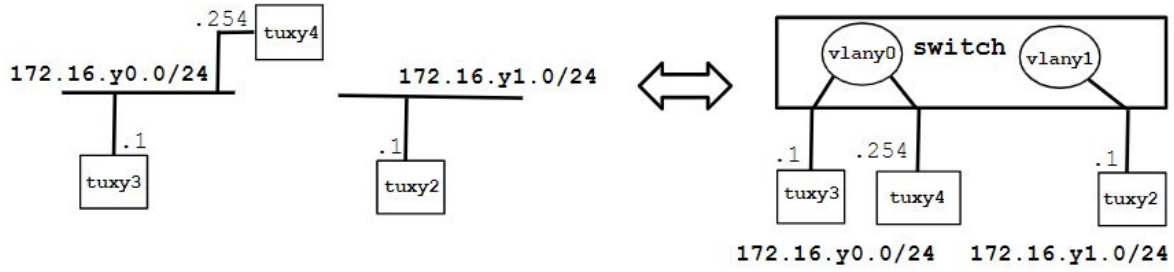

Figura 3 - Arquitetura da rede na experiência 2

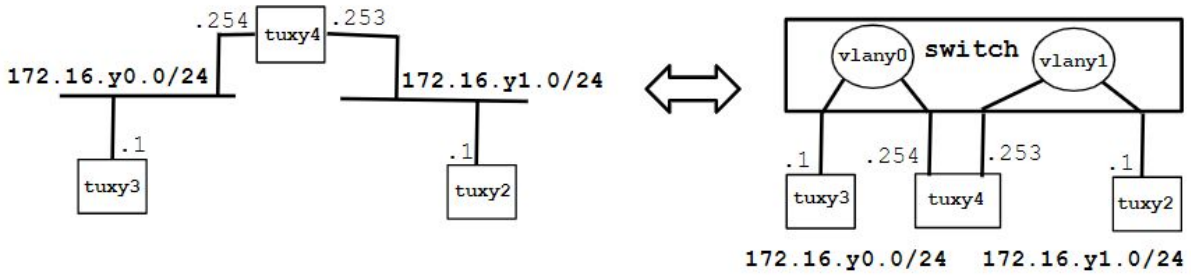

Figura 4 - Arquitetura da rede na experiência 3

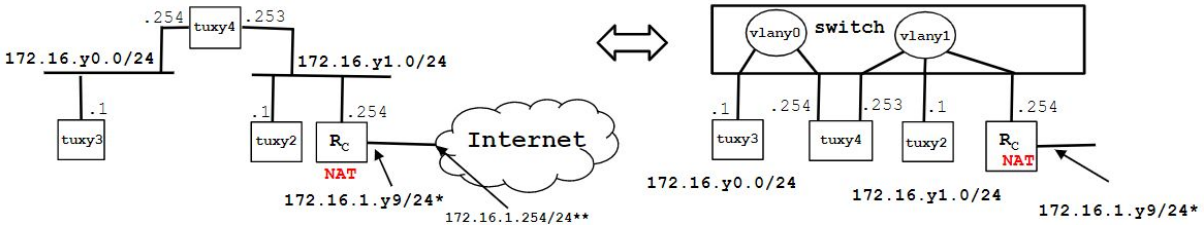

Figura 6 - Arquitetura da rede na experiência 4

tuxy3$ nano /etc/resolv.conf
search netlab.fe.up.pt
*nameserver 172.16.1.1**

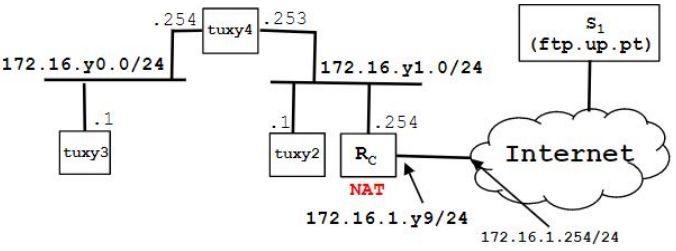

Figura 7 - Arquitetura da rede na experiência 5

tuxy3$ nano /etc/resolv.conf
search netlab.fe.up.pt
nameserver 172.16.1.1

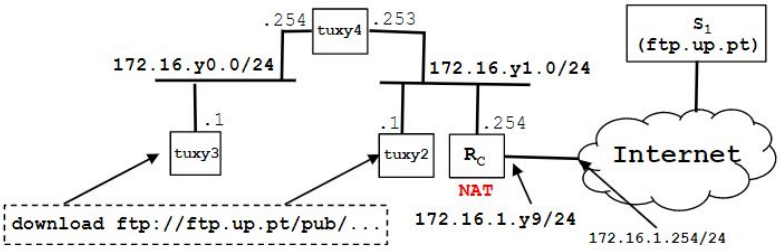

Figura 9 - Arquitetura da rede na experiência 6

Anexo III - Comandos de configuração

Experiência 1 - Configuração de um endereço IP de rede

tux3:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.60.1/24
> ifconfig eth0
```

tux4:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.60.254/24
> ifconfig eth0
```

tux3:

```
> ping 172.16.60.254
```

tux4:

```
> ping 172.16.60.1
```

tux3:

```
> route -n
> arp -a
```

Experiência 2 - Implementar duas LANs virtuais num switch

tux3:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.20.1/24
> ifconfig eth0
```

tux4:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.60.254/24
> ifconfig eth0
```

tux2:

```
»configure terminal
»vlan 60
»end
»show vlan id 60

»configure terminal
»interface fastethernet 0/1
»switchport mode access
»switchport access vlan 60
»end
»show running-config interface fastethernet 0/1
»show interfaces fastethernet 0/1 switchport
```

```
»configure terminal
»interface fastethernet 0/2
»switchport mode access
```

```
»switchport access vlan 60
»end
```

```
»configure terminal
»vlan 61
»end
»show vlan id 61

»configure terminal
»interface fastethernet 0/3
»switchport mode access
»switchport access vlan 61
»end
```

Experiência 3 - Configuração de um router em Linux

tux3:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.60.1/24
> ifconfig eth0
```

tux4:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.60.254/24
> ifconfig eth0

> ifconfig eth1 up
> ifconfig eth1 172.16.61.253/24
> ifconfig eth1
```

tux2:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.61.1/24
> ifconfig eth0
```

```
»configure terminal
»vlan 60
»end
»show vlan id 60
```

```
»configure terminal
»interface fastethernet 0/1
»switchport mode access
»switchport access vlan 60
»end
```

```
»configure terminal
»interface fastethernet 0/3
»switchport mode access
»switchport access vlan 60
»end
```

```
»configure terminal
»vlan 61
»end
»show vlan id 61
```

```
»configure terminal
»interface fastethernet 0/2
»switchport mode access
»switchport access vlan 61
»end
```

```
»configure terminal
»interface fastethernet 0/4
»switchport mode access
»switchport access vlan 61
»end
```

tux4:

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

tux3:

```
route add -net 172.16.61.0/24 gw 172.16.60.254
```

tux2:

```
route add -net 172.16.60.0/24 gw 172.16.61.253
```

tux3:

```
ping 172.16.61.1
```

tux2:

```
ping 172.16.60.1
```

```
route -n
```

Experiência 4 - Configuração de um router comercial e implementação de NAT

```
»interface gigabitethernet 0/0
»ip address 172.16.61.254    255.255.255.0
»no shutdown
»exit
»show interface gigabitethernet 0/0
```

```
#Configurar Ips
#Criar Vlans e adicionar portas
#ip forwarding e disable ICMP echo-ignore-broadcast no tux24
```

```
#adicionar routes
```

```
> route -n
```

```
conf -t

interface gigabitethernet 0/0 *
ip address 172.16.61.254 255.255.255.0
no shutdown
ip nat inside
exit
```

```
interface gigabitethernet 0/1 *
ip address 172.16.1.69 255.255.255.0
no shutdown
ip nat outside
exit
```

```
ip nat pool ovrld 172.16.1.69 172.16.1.69 prefix 24
ip nat inside source list 1 pool ovrld overload
access-list 1 permit 172.16.60.0 0.0.0.7
access-list 1 permit 172.16.61.0 0.0.0.7
ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.60.0 255.255.255.0 172.16.61.253
end
```

Experiência 5 - DNS

tux2, tux3 e tux4:

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

```
ping ftp.up.pt
```