U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Rede de Computadores

(2º Trabalho Laboratorial)

Redes de Computadores · L.EIC025

2023/2024 · 1º Semestre

Turma: 3LEIC08

João Correia up202005015

Miguel Charchalis up201506074

1- Sumário

No âmbito da unidade curricular Redes de Computadores, realizou-se o 2º projeto que nos foi proposto. O projeto "Computer Networks" está dividido em duas partes. A primeira parte consiste em desenvolver uma aplicação de *download* de um ficheiro utilizando o protocolo FTP (*File Transfer Protocol*). A segunda parte consiste em realizar várias experiências para configurar uma rede.

O projeto foi realizado com sucesso, pois criamos uma aplicação capaz de transferir ficheiros e configuramos a rede como foi proposto.

2- Introdução

Através deste relatório, pretendemos apresentar de maneira organizada o funcionamento da aplicação de *download* de um ficheiro, quando fornecido um URL no formato ftp://[<user>:<password>@]<host>/<url-path> e também o funcionamento da configuração da rede de computadores capaz de se conectar à internet e testar a aplicação.
Organizamos o relatório com a seguinte estrutura:

- **Aplicação de *download***
  Descrição da arquitetura e do caso de sucesso.
- **Configuração e análise da rede**
  Descrição da arquitetura da rede, objetivos, principais comandos de configuração e análise dos principais *logs* capturados para as seis experiências realizadas.
  - Experiência 1 – Configurar uma rede IP
  - Experiência 2 – Implementação de duas *bridges* no *switch*
  - Experiência 3 – Configuração de um *router* em Linux
  - Experiência 4 – Configurar um *router* comercial e implementar NAT
  - Experiência 5 – DNS
  - Experiência 6 – Conexões TCP
- **Conclusões**
  Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

3- Aplicação *download*

3.1- Arquitetura

Para realizar a execução da aplicação, foram consultadas as normas RFC959 (funcionamento e arquitetura do protocolo FTP) e RFC1738 (tratamento do URL).

Inicialmente compila-se o programa e passamos-lhe como argumento o seguinte link:

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

A aplicação faz um *parse* do URL, armazenando cada componente num array. Se o *user* e a *password* não forem especificados, são assumidos como “anonymous” e “password”.

Depois de interpretados os dados do URL, é feita a comunicação com o servidor através de um *socket* TCP. O passo seguinte é o programa tentar aceder ao servidor com as credenciais indicadas e, depois o login, o programa solicita ao servidor a resposta em modo passivo (PASV), pois o servidor é quem vai escolher para que porta o ficheiro será enviado. Outro *socket* é aberto para a transferência do ficheiro. Após concluída, os dois *sockets* são fechados.

3.2- Caso de sucesso

Para testar a aplicação, realizamos os testes com dois ficheiros distintos, pic1.jpg e timestamp.txt, com e sem autenticação no servidor (Figura 3.2a e 3.2b).

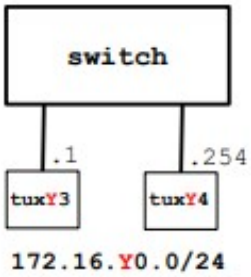
```
(base) joaocorreia@MacBook-Pro-de-Joao projeto2 % ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/pic1.jpg

user: rcom
password: rcom
host: netlab1.fe.up.pt
url-path: /files/pic1.jpg

Host name  : netlab1.fe.up.pt
IP Address : 192.168.109.136
220 Welcome to netlab-FTP server

331 Please specify the password.

230 Login successful.
227 Entering Passive Mode (192,168,109,136,170,131).
Written 22 bytes
150 Opening BINARY mode data connection for /files/pic1.jpg (340603 bytes).

Receiving file pic1.jpg...
226 Transfer complete.
221 Goodbye.
```

Figura 3.2a - *Transferência do ficheiro pic1.jpg (340603 bytes) com autenticação*

```
(base) joaocorreia@MacBook-Pro-de-Joao projeto2 % ./download ftp://ftp.up.pt/pub/kodi/timestamp.txt

user: anonymous
password: password
host: ftp.up.pt
url-path: pub/kodi/timestamp.txt

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.
227 Entering Passive Mode (193,137,29,15,210,127).
Written 29 bytes
150 Opening BINARY mode data connection for pub/kodi/timestamp.txt (11 bytes).

Receiving file timestamp.txt...
226 Transfer complete.
221 Goodbye.
```

Figura 3.2b - *Transferência do ficheiro timestamp.txt (11 bytes) sem autenticação*

4- Configuração e análise de redes



4.1- Experiência 1 - Configurar uma rede IP

O objetivo desta experiência é configurar duas máquinas, Tux53 e Tux54, sendo necessário a atribuição de um endereço IP para cada uma delas.

Foi ligado a entrada E0 de cada Tux ao switch e com o comando ifconfig, configurou-se os IPs de cada um. O Tux 53 foi configurado com o IP 172.16.50.1/24, enquanto o Tux 54 foi configurado com o IP 172.16.50.254/24.

O protocolo ARP (*Address Resolution Protocol*) é um procedimento para mapear endereços de IP dinâmicos para um endereço de máquina físico numa rede local. Assim, associa o endereço IP com o endereço MAC. Depois de apagar a tabela ARP dos computadores, o protocolo ARP foi executado.

O protocolo ICMP (*Internet Control Message Protocol*) é utilizado para trocar mensagens de controlo, indicando sucesso ou erros que possam existir durante a comunicação com outro endereço.

Principais comandos:

Configuração de ip:

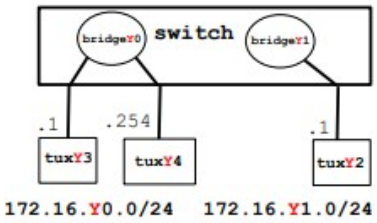
- *ifconfig ethX*
- *ifconfig ethX <ip-do-computador>*

Verificar conexão entre os dois Tuxs:

- *ping <ip-do-computador-que-pretende-verifcar>*

4.2- Experiência 2 – Implementação de duas *bridges* no *switch*

O objetivo desta experiência é criar duas *bridges* no *switch*, e perceber a conectividade



entre os computadores e como estas influenciam a troca de informação entre as máquinas.

Inicialmente repetiu-se o processo de configuração de IP para o Tux 52 (172.16.51.1).

Para configurarmos as *bridges*, criamos a bridge50 e bridge51 através do *GTkterm*, e associamos à primeira o tux53 e o tux54 e à segunda o tux52, tendo assim a arquitetura pretendida. Depois, podemos verificar a conectividade entre os computadores, fazendo um *ping* do tux53 até ao tux54, que foi executado com sucesso, pois estes encontram-se na mesma sub-rede. Fazendo um *ping* do tux53 até ao tux52, não obtivemos resposta, pois não existe nenhuma rota entre as duas *bridges*.

Principais comandos:

Para criar as *bridges*, no *GTkTerm:*

- */interface bridge add name=bridgeY0*

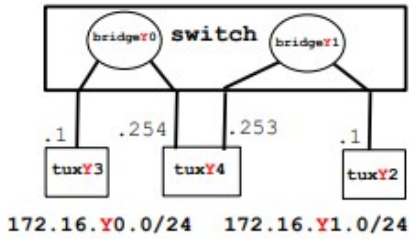
Depois para adicionar as correspondentes *ports* às *bridges*:

- */interface bridge port add bridge=bridgeY0 interface=etherX*

Por fim utilizamos o comando *ping* pela mesma lógica referida na Experiência 1.

4.3- Experiência 3 – Configuração de um *router* em Linux

O propósito desta experiência é ajustar o tux54 de forma a que funcione como um router, facilitando a comunicação entre o tux53 e o tux52 através das bridges configuradas na experiência anterior. Para possibilitar a comunicação, é necessário configurar os endereços IP das portas e interfaces Ethernet dos computadores, bem como as rotas que serão utilizadas.



Para estabelecer uma conexão entre o tux53 e o tux52, é necessário adicionar uma rota ao tux53 para acessar os endereços 172.16.51.0/24 a partir do IP 172.16.50.254. Dessa forma, quando o tux53 deseja enviar um ping para a bridge51, ele utilizará o router, que neste caso é o tux54 (172.16.50.254), como gateway. Além disso, é necessário adicionar uma rota ao tux52 para acessar os endereços 172.16.50.0/24 a partir do IP 172.16.51.253.

Essas rotas podem ser visualizadas na tabela de encaminhamento usando o comando **route -n**. A tabela de encaminhamento é um conjunto de entradas onde cada entrada contém informações do tipo Destino-Gateway-Interface. O destino é o IP do computador de destino, o gateway é o IP do computador para o qual a mensagem será enviada e a interface é a placa de rede usada para enviar a mensagem.

Assim, com essas rotas definidas, torna-se possível realizar ping, a partir do tux53, para todas as interfaces dos outros computadores.

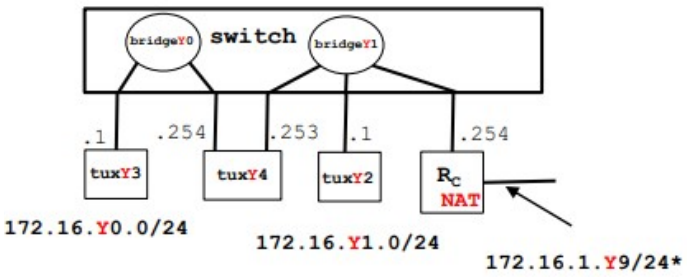
Dessa forma, ao realizar um ping do tux53 para o tux52, são registrados os seguintes pacotes ARP. Inicialmente, é identificado um pacote que solicita o envio do endereço MAC associado ao IP 172.16.50.254 para o endereço 172.16.50.1. Isso ocorre porque o tux53 está identificando a interface do tux54, que encaminha o pacote enviado pelo ping para o tux52. O endereço MAC de resposta corresponde à interface eth0 do tux54.
Em seguida, na mesma interface, é observado o pacote ARP que requisita o envio do endereço MAC de 172.16.50.1 para 172.16.50.254. Essa solicitação ocorre porque o tux54 precisa do endereço MAC da interface do tux53 para encaminhar a resposta do tux52 ao ping. Na eth1 do tux54, há uma interação em que o tux54 pergunta pelo endereço MAC do tux52 para encaminhar o pacote de ping, e o tux52 fornece o endereço MAC de eth0 do tux52.

Posteriormente, o tux52 solicita o endereço MAC de eth1 do tux54 para enviar a resposta do ping, e o tux54 responde com o endereço MAC de eth1 do tux54. Essa troca de mensagens ARP ocorre sempre que uma mensagem é enviada de uma máquina para outra, quando os endereços MAC não são conhecidos.

Ao analisar os pacotes ICMP, podemos identificar pacotes do tipo request e reply, indicando que todas as rotas estão devidamente configuradas. Caso não estivessem configuradas, veríamos pacotes ICMP do tipo Host Unreachable. Os endereços IP de destino nos pacotes

ICMP sempre correspondem aos IPs de nossa máquina, enquanto os endereços IP associados aos pacotes ICMP de origem representam os IPs pelos quais os pacotes viajam para atingir o destino desejado. O endereço MAC de destino nos pacotes ICMP está associado à interface virtual, ao passo que o endereço MAC de origem nos pacotes ICMP é o endereço MAC da interface virtual do computador host.

4.4- Experiência 4 – Configurar um *router* comercial e implementar NAT



O objetivo desta experiência é estabelecer uma ligação com a rede dos laboratórios e implementar rotas num *router* comercial, configurando-lhe funcionalidade NAT para garantir a conexão entre as máquinas e a internet.

Inicialmente, é necessário configurar o router. Para realizar esta configuração, iniciamos a sessão no router por meio do GTkTerm e conectamos o cabo S0 de qualquer Tux à entrada de configuração do router. Essa abordagem permite o acesso ao router via GTkTerm, possibilitando a configuração do IP do router e das rotas usando o comando **ip route**.

Existem dois cenários possíveis para possibilitar o acesso à internet a partir do computador. No primeiro cenário, uma rota é configurada entre as duas máquinas, e a informação segue essa rota. No segundo cenário, caso essa rota não exista, os pacotes são enviados pela rota padrão, e a máquina para a qual a rota padrão está definida encaminha os pacotes para o destino. Na experiência em questão, como a rota do Tux52 para o Tux53 foi apagada, os pacotes foram redirecionados para o router e, posteriormente, encaminhados para o Tux54.

O NAT (*Network Address Translation*) é um mecanismo implementado em routers que substitui os endereços IP locais nos pacotes por um endereço IP público, permitindo estabelecer conexões para fora da rede. Dessa forma, o router que implementa o NAT assume a responsabilidade de encaminhar todos os pacotes para os endereços corretos, seja dentro ou fora da rede local.
Nesta experiência, considerando que o Tux53 deseja enviar um pacote para um endereço em uma rede pública, o pacote é inicialmente enviado para o router. Este router modifica o endereço de origem do pacote para o seu endereço externo, garantindo assim a privacidade e a segurança do remetente. O pacote é então enviado, e, ao receber uma resposta, o router a encaminha de volta para o Tux53, ajustando o destinatário do pacote para o seu próprio endereço.

Esse processo possibilita a comunicação entre a rede privada local e a rede pública, enquanto mantém a segurança e a privacidade dos dispositivos locais ao ocultar os seus endereços IP privados na rede externa.

4.4.2- Principais comandos

Configurar *ip* e as rotas do *router*:

- */ip address add address=172.16.1.Y9/24 interface=ether1*
- */ip address add address=172.16.Y1.254/24 interface=ether2*
- */ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253*
- */ip route add address=0.0.0.0/0 gateway=172.16.2.254*

No tuxy2 definir RC como *default router*:

- *route add default gw 172.16.y1.254*

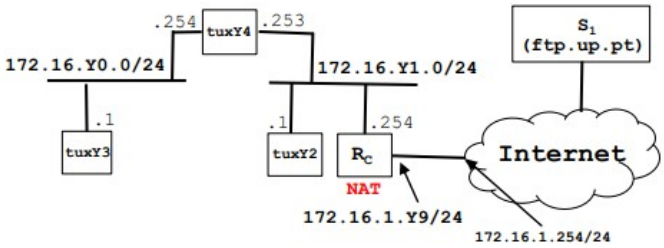
No tuxy3 definir tuxy4 como *default router*:

- *route add default gw 172.16.y0.254*

No tuxy4 definir RC como *default router*:

- *route add default gw 172.16.y1.254*

4.5- Experiência 5 – DNS

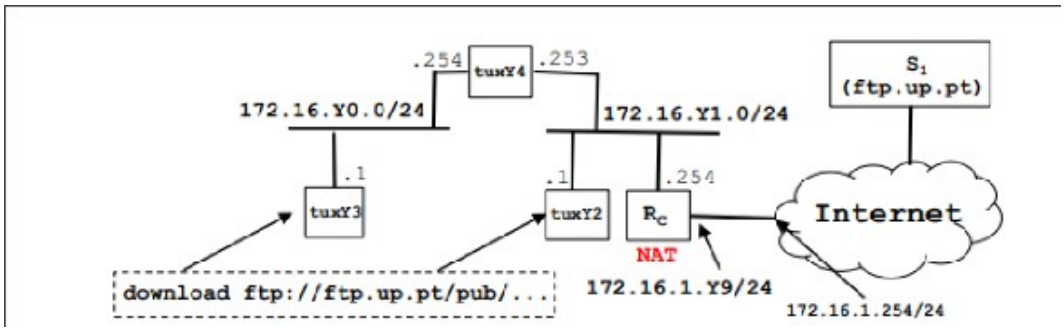


O objetivo desta experiência é configurar o DNS responsável por traduzir endereços URL em endereços IP.

O serviço DNS é configurado no ficheiro *resolv.conf de cada Tux* bastando adicionar no ficheiro:

- search netlab.fe.up.pt
- nameserver 172.16.1.1

4.6- Experiência 6 – Conexões TCP



O objetivo desta experiência é compilar e correr o programa download que fizemos.

Ao executar o codigo não foi possível extrair os ficheiros usando a rede que configuramos. Depois de revermos a configuração da rede e consultarmos com o professor do Laboratório, chegamos à conclusão que o problema não era da nossa configuração da rede mas sim com algo fora da nossa responsabilidade.

Testamos o programa download noutro computador e conseguimos extrair os dois ficheiros com sucesso.

5 - Conclusão

Podemos concluir que os objetivos de desenvolver uma aplicação que permite o download de um arquivo por meio de conexões utilizando os protocolos FTP e TCP, juntamente com a configuração de uma rede IP, foram alcançados com sucesso. Desta forma, foi possível compreender o funcionamento do router e do switch, assim como diversas técnicas (NAT, DNS, etc.), protocolos (ICMP, ARP, etc.) e estruturas de dados utilizados para realizar a comunicação entre as máquinas, como as tabelas ARP e as forwarding tables.

Além disso, a análise dos vários logs das experiências capturados nas diferentes máquinas Tux por meio do Wireshark proporcionou um entendimento profundo de como as diferentes camadas de comunicação interagem dentro de uma rede de computadores. Em resumo, podemos afirmar com convicção que adquirimos conhecimentos de extrema relevância ao longo do desenvolvimento deste projeto.

6 – Anexo – Aplicação download

```c
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define MAX_SIZE 512
#define FTP_PORT 21

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

    struct hostent* h;
    char buf[MAX_SIZE], user[50], password[50], host[100], urlPath[MAX_SIZE];

    if (argc != 2) {
        fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
        exit(-1);
    }

    sscanf(argv[1],"ftp://%511s",buf);

    size_t i = strcspn(buf,":"), size = strlen(buf);
```

```c
if(i < size) {
    size_t j = strcspn(buf,"@");

    if(j == size) {
        fprintf(stderr,"Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
        exit(-1);
    }

    strncpy(user,buf,i);
    user[i] = '\0';
    strncpy(password,buf+i+1,j-i-1);
    password[j-i-1] = '\0';
    i = strcspn(buf+j+1,"/");
    strncpy(host,buf+j+1,i);
    host[i] = '\0';
    strcpy(urlPath,buf+j+i+1);
}
else {
    i = strcspn(buf,"/");
    strncpy(host,buf,i);
    host[i] = '\0';
    strcpy(urlPath,buf+i+1);
    strcpy(user,"anonymous");
    strcpy(password,"password");
}

printf("\nuser: %s\npassword: %s\nhost: %s\nurl-path: %s\n\n",user,password,host,urlPath);

if ((h = gethostbyname(host)) == NULL) {
    herror("gethostbyname()");
    printf("%s\n",host);
    exit(-1);
}

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

```c
//server address handling
struct sockaddr_in server_addr;
bzero((char*) &server_addr,sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(inet_ntoa(*((struct in_addr*) h->h_addr_list[0])));
server_addr.sin_port = htons(FTP_PORT); //server TCP port must be network byte ordered

//open a TCP socket
int sockfd;
if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
    perror("socketfd\n");
    exit(-1);
}

//connect to the server
if (connect(sockfd,(struct sockaddr*) &server_addr,sizeof(server_addr)) < 0) {
    printf("%s\n",h->h_addr_list[0]);
    perror("connect\n");
    exit(-1);
}

//send a string to the server
usleep(100000);
int bytesRead = read(sockfd,buf,MAX_SIZE);
buf[bytesRead] = '\0';
printf("%s\n",buf);

if(strncmp(buf,"220",3) != 0) {
    perror("220\n");
    exit(-1);
}
```

```
char buf2[519];
sprintf(buf2,"user %s\r\n",user);
size_t bytes = write(sockfd, buf2, strlen(buf2));
bytesRead = read(sockfd,buf,MAX_SIZE);
buf[bytesRead] = '\0';
printf("%s\n",buf);

if(strncmp(buf,"331 Please type the password.",32) != 0) {
    perror("331 Please type the password.\n");
    exit(-1);
}


sprintf(buf2,"pass %s\r\n",password);
write(sockfd,buf2,strlen(buf2));
bytesRead = read(sockfd,buf,MAX_SIZE);
buf[bytesRead] = '\0';
printf("%s",buf);

if(strncmp(buf,"230",3) != 0) {
    perror("230\n");
    exit(-1);
}


strcpy(buf2,"pasv\r\n");
write(sockfd,buf2,strlen(buf2));
bytesRead = read(sockfd,buf,MAX_SIZE);
buf[bytesRead] = '\0';
printf("%s",buf);
```

```c
if(strncmp(buf,"227",3) != 0) {
    perror("227\n");
    exit(-1);
}

int ip1 = 0, ip2 = 0, ip3 = 0, ip4 = 0;
int p1 = 0, p2 = 0;
int pasvport;

sscanf(buf,"227 Entering Passive Mode (%i,%i,%i,%i,%i,%i).",&ip1,&ip2,&ip3,&ip4,&p1,&p2);
pasvport = p1*256+p2;


//server address handling
struct sockaddr_in server_addrC;
sprintf(buf,"%i.%i.%i.%i",ip1,ip2,ip3,ip4);
bzero((char*) &server_addrC,sizeof(server_addrC));
server_addrC.sin_family = AF_INET;
server_addrC.sin_addr.s_addr = inet_addr(buf); //32 bit Internet address network byte ordered
server_addrC.sin_port = htons(pasvport); //server TCP port must be network byte ordered

int sock;
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("sock\n");
    exit(-1);
}

//connect to the server
if (connect(sock,(struct sockaddr*) &server_addrC,sizeof(server_addrC)) < 0) {
    perror("connect\n");
    exit(-1);
}
```

```
fclose(fd);

do {
    memset(buf, 0, MAX_SIZE);
    read(sockfd,buf,MAX_SIZE);
} while (buf[0] < '1' || buf[0] > '5' || buf[3] != ' ');
printf("%s", buf);
buf[3] = '\0';

if(strncmp(buf,"226",3) != 0) {
    perror("226\n");
    exit(-1);
}

sprintf(buf2,"QUIT\r\n");
write(sockfd,buf2,strlen(buf2));
bytesRead = read(sockfd,buf,MAX_SIZE);
buf[bytesRead] = '\0';
printf("%s",buf);

if(strncmp(buf,"221",3) != 0) {
    perror("221\n");
    exit(-1);
}

if (close(sockfd) < 0) {
    perror("close sockfd\n");
    exit(-1);
}

if (close(sock) < 0) {
    perror("close sockfdC\n");
    exit(-1);
}

return 0;
```