Desenvolvimento de aplicação FTP e setup de uma rede

RCOM TP2

by Diogo Babo, Gustavo Costa, João Oliveira, LEIC02

Índice

Sumário

O TP2 remete ao download de ficheiros da web através de um cliente FTP, numa máquina pertencente a uma rede configurada por nós. Esta rede conta com conceitos como: bridges, switches, routers, IP address, MAC address, routes, servidores DNS, NAT, entre outros.

Introdução

O relatório terá a seguinte estrutura, com o intuito de esclarecer o objetivo de cada um dos tópicos, assim como revelar a forma e funcionamento do trabalho que foi feito até agora:

- **Parte 1**:
  - Arquitetura da aplicação de download.
  - Resultados.
- **Parte 2** (Para cada experiência, da 1 à 6):
  - Objetivos da experiência.
  - Arquitetura da rede.
  - Comandos principais à configuração.
  - Análises das logs capturadas via wireshark.
- **Conclusões** - síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.
- **Referências**
- **Anexos** - código da aplicação de download, comandos de configuração e logs wireshark.

Parte 1

Arquitetura da aplicação de download

A aplicação de download consta com 10 funções cujas funcionalidades se encontram, aqui detalhadas.

int main(int argc, char *argv[]);

Função main, responsável por chamar as restantes funções sequencialmente, garantindo assim o fluxo normal de um programa FTP, dependendo das respostas dadas pelo servidor FTP.

int argParser(char* input, struct urlData* data);

Função responsável por dar *parse* ao input do utilizador, decompondo o endereço do servidor FTP nas suas componentes essenciais, como o *user*, *password*, *host*, *ip, path e nome do file, guardando-os na struct urlData.*

int startSocket(char* ip, int port);

Inicia a conexão *socket* TCP na porta recebida nos argumentos e liga-se ao servidor dado por *ip.*

int login(char* username, char* password, int fd);

Recorrendo às funções sendMessage & readMessage, esta função é responsável por fazer o login onde são enviados os comandos USER username e PASS password.

int sendMessage(int fd, char* msg);

Envia comandos FTP para o servidor.

int readMessage(int fd);

Lê as respostas enviadas pelo servidor FTP.

int readPassive(int fd);

Lê as respostas enviadas pelo servidor, contudo é usada após ser enviado o comando PASV. A função vai calcular a nova porta, e também guardar o IP para abrir a conexão na nova socket.

int passiveMode(int fd);

Envia o comando PASV para o servidor.

int fileMessage(int fd);

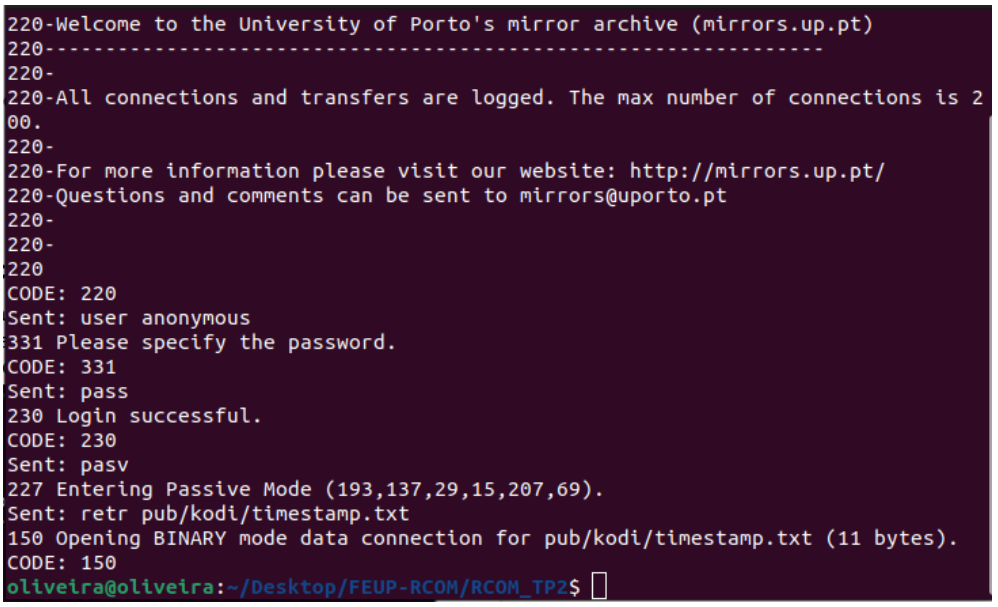
Envia o comando RETR *filename* para a socket.

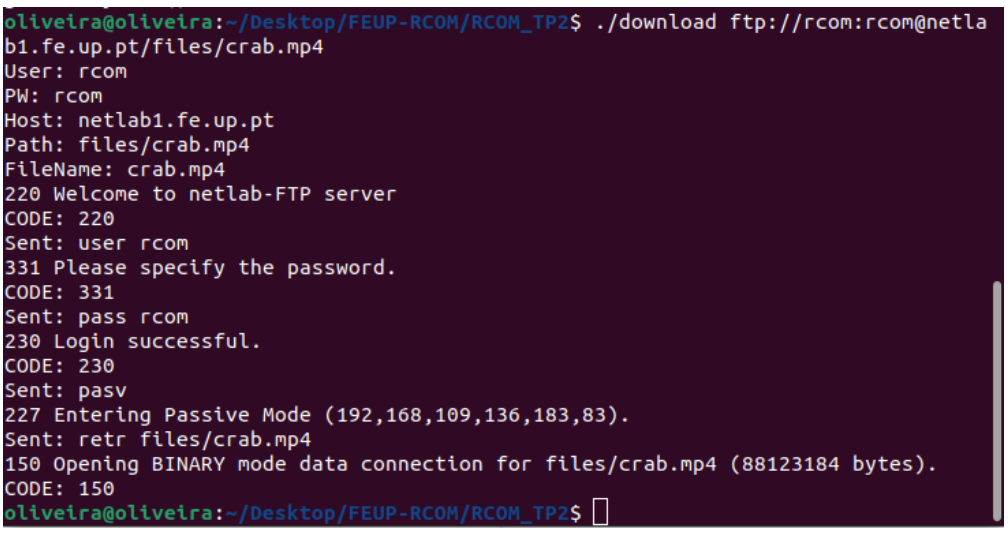
int writeFile(int fd);

Lê os bytes transmitidos pelo servidor FTP, e escreve-os para um ficheiro novo.

Resultados

A aplicação funcionou como esperado, para tanto URL's com username e password como sem nenhum dos dois (caso em que o username é anonymous). Além disso, são verificados possíveis erros que possam ocorrer no decorrer do programa.





Parte 2

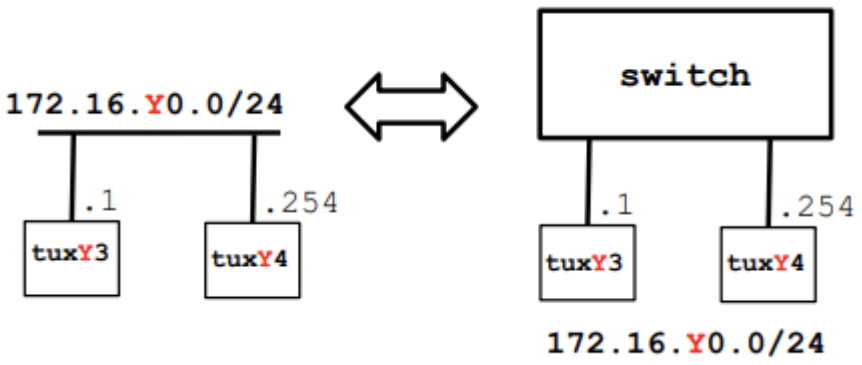
Experiência 1: Configurar uma rede IP

Objetivos da experiência

Esta experiência tem como objetivo configurar uma rede IP entre duas máquinas (Tux3 e Tux4) permitindo assim a comunicação entre elas.

Arquitetura da rede

A arquitetura desejada pode ser representada pelo seguinte diagrama:



Legenda: Y - nr da bancada

É possível observar que a máquina Tux3 (.1) e a máquina Tux4(.254) conseguem comunicar entre si pois encontram-se na mesma rede, Y0.

Fisicamente, estas máquinas encontram-se ligadas a uma switch.

Comandos principais à configuração

Em cada Tux é necessário correr apenas um comando na sua consola: ifconfig <interface> <endereço ip>/<netmask>

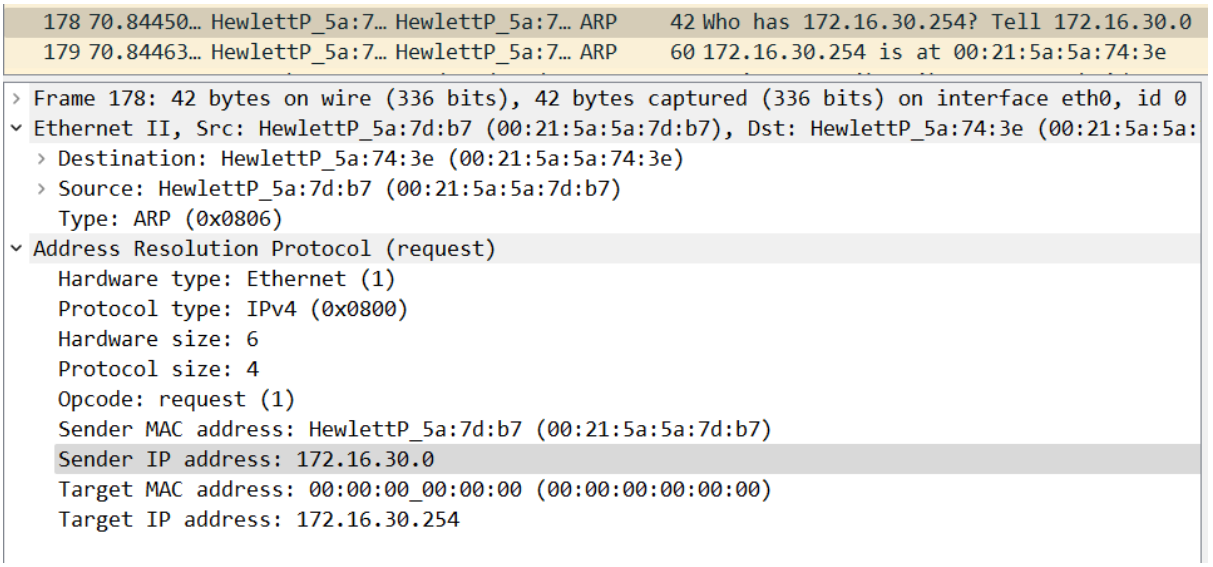
Exemplo, no Tux3: ifconfig eth0 172.16.Y0.1/24

Análises das logs capturadas via wireshark

P: “What are the ARP packets and what are they used for? What are the MAC and IP addresses of ARP packets and why?”

R: ARP (Address Resolution Protocol) é um protocolo que serve para mapear endereços IP (camada 3 ou camada lógica do modelo OSI) em endereços MAC (camada 2 ou camada física do modelo OSI). Ao tentar enviar um ping do Tux3 para o Tux4, este verifica se a MAC address do Tux4 se encontra mapeada na sua ARP table. Caso não se

encontre, este envia um pacote ARP para a sua rede local a perguntar qual a MAC address da máquina que tem o endereço IP 172.16.30.254, à qual o Tux4 responde com um pacote ARP a dizer que esse endereço se encontra na MAC 00:21:5a:5a:74:3e.

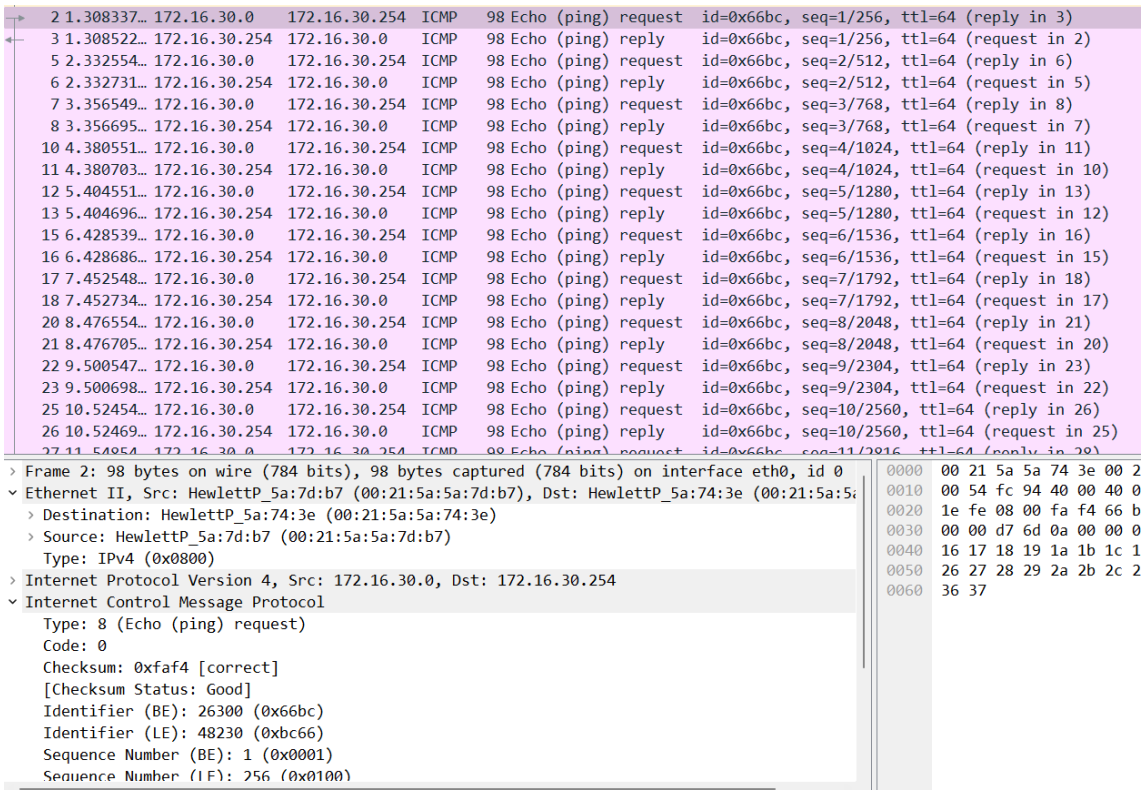


A MAC address alvo do pacote enviado pelo Tux3 é 00:00:00:00:00:00 pois este não sabe a MAC do Tux4, daí a necessidade do pacote ARP em si.

Caso o Tux4 mude de endereço de IP, este automaticamente enviará um pacote ARP para a sua rede local a avisar desta mudança, garantindo assim integridade nas ARP tables de todas as máquinas dessa rede.

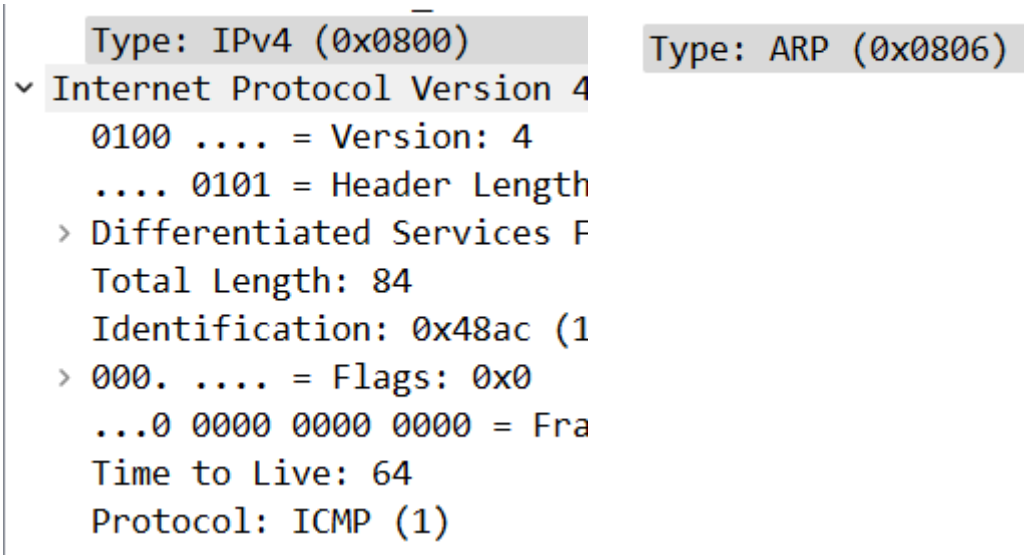
P: “What packets does the ping command generate? What are the MAC and IP addresses of the ping packets?”

R: Os comandos ping geram pacotes ICMP (Internet Control Message Protocol), caso os endereços IP estejam mapeados na ARP table. Quando o Tux3 envia um ping para o Tux4, este responde com um ping ao Tux3, confirmando a receção do ping anterior. As MAC e IP addresses destes pings correspondem às MAC e IP addresses destas duas máquinas.



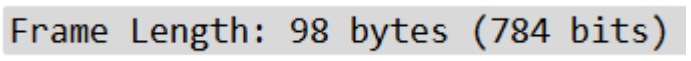
P: “How to determine if a receiving Ethernet frame is ARP, IP, ICMP? “

R: É possível distinguir entre tramas Ethernet ARP e IP/ICMP com base no seu header. Se este tomar o valor 0x0806, trata-se de uma trama ARP, se este tomar o valor 0x0800 trata-se de uma trama IP/ICMP. Considerando que o protocolo ICMP é derivado do protocolo IP, para distinguir entre tramas destes dois protocolos é necessário verificar o valor do IP header. Caso este esteja a 1, trata-se do protocolo ICMP.



P:”How to determine the length of a receiving frame?”

R: Pode ser obtido via Wireshark, no campo chamado “Frame Length”



P:”What is the loopback interface and why is it important?”

R: A interface de loopback é uma interface virtual sempre ativa cuja funcionalidade principal é testar as configurações da rede. Devido ao seu modo de funcionamento, todas as mensagens que saem por esta interface, voltam a entrar por esta interface.

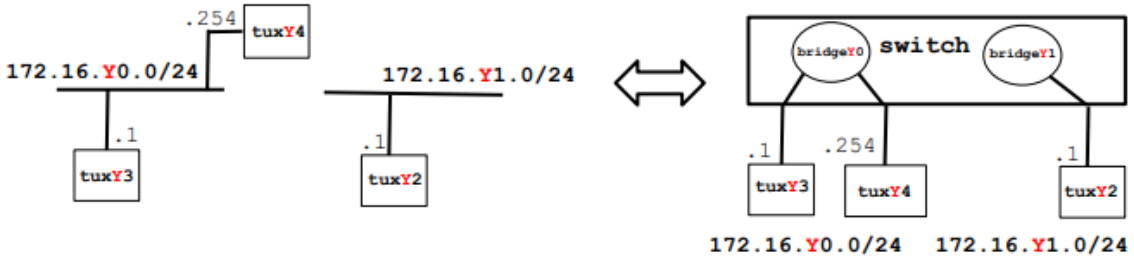
Experiência 2: Implementar duas bridges numa switch

Objetivos da experiência

Esta experiência tem como objetivo configurar duas bridges, uma por cada rede local, introduzindo uma terceira máquina, Tux2.

Arquitetura da rede

Após a realização da experiência, a rede deverá ter a seguinte estrutura:



Como é possível verificar, relativamente à experiência anterior, introduziu-se uma terceira máquina, Tux2, numa rede distinta da já existente.

Não havendo ligação física ou lógica entre as redes, o Tux3 e o Tux4 conseguem comunicar entre si, mas não com o Tux2.

Mais uma vez, relembramos que Y é o nº da bancada onde se encontram as máquinas.

Comandos principais à configuração

Desta vez é necessário configurar as bridges usando a consola da switch.

Para tal, criam-se as duas bridges (Z = nº da bridge, 0 ou 1):

/interface bridge add name=bridgeYZ

Removem-se as antigas configurações das portas ethernet onde as máquinas estão ligadas (XX = nº da porta ethernet):

/interface bridge port remove [find interface=etherXX]

E adicionam-se as mesmas portas às respectivas bridges:

/interface bridge port add interface=etherXX bridge=bridgeYZ

Análises das logs capturadas via wireshark

P:"How to configure bridgeY0?"

R:Para configurar a bridgeY0, seguem-se os passos anteriormente descritos, tendo em consideração que as interfaces ethernet de cada Tux se encontram ligadas a qualquer porta da switch, exceto a console e a nº 1.

P:"How many broadcast domains are there? How can you conclude it from the logs?"

R: Um domínio de broadcast é uma divisão lógica numa rede de computadores na qual todos os nós conseguem comunicar entre si via broadcast. Neste caso, como o Tux 3 e 4 conseguem comunicar entre si, mas não com o Tux2, por não haver ligação entre as redes, existem 2 domínios de broadcast. Isto é possível observar nas logs, pois quando o Tux3 ou Tux4 estão a fazer ping -b <ip> o Tux2 não deteta nada, e vice-versa.

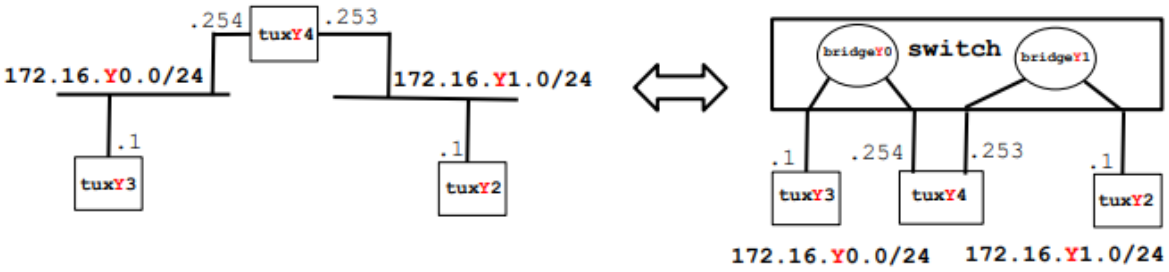
Experiência 3: Configurar um Router em Linux

Objetivos da experiência

A experiência 3 tem como objetivo configurar um Router em Linux, permitindo a comunicação entre PCs localizados em duas redes distintas.

Arquitetura da rede

Após a realização da experiência, a rede deverá ter a seguinte estrutura:



Partindo da experiência anterior, introduziu-se uma ligação do Tux4 à bridgeY1, de modo a este funcionar como um router, permitindo assim a comunicação entre as duas redes.

Comandos principais à configuração

Para configurar o Tux4 como um router, é necessário ligar outra das suas interfaces ethernet (eth1) à bridge à qual este não se encontra conectado.

Assim sendo, após conectar o eth1 do Tux4 a qualquer porta da switch e seguir as configurações anteriormente apresentadas para configurar o seu IP e introduzi-lo na bridgeY1, é preciso habilitar o IP forwarding e desabilitar o ICMP echo-ignore-broadcast com os seguintes comandos:

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

echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

É também necessário reconfigurar o Tux2 e Tux3 para que estes considerem o Tux4 como o seu router default:

route add default gw 172.16.Y1.253 (Tux2)

route add default gw 172.16.Y0.254 (Tux3)

Análises das logs capturadas via wireshark

P:"What routes are there in the tuxes? What are their meaning?"

R: Quer o Tux2 quer o Tux3 têm rotas default para o Tux4. Uma rota default é um caminho usado por um pacote quando não existe outro caminho para este seguir. Assim sendo, quando um pacote é enviado de um dos Tux, este primeiro verifica se o pacote tem como destino um IP da sua rede local. Como o Tux2 e Tux3 se encontram em redes distintas, esta verificação falha e então o Tux enviará esse pacote para a rota default, que é o Tux4. Considerando que o Tux4 funciona como um router, a existência das rotas default para o Tux4 no Tux2 e Tux3 permite a comunicação entre estas duas máquinas.

P:"What information does an entry of the forwarding table contain?"

R: Na forwarding table cada entrada contém as seguintes informações:

Destination - o IP do computador de destino;

Genmask - a netmask para a rede de destino;

Gateway - o IP do computador para o qual a mensagem vai ser enviada, sendo que é este que depois vai dar route da mensagem para o destino;

Interface - placa de rede utilizada para enviar a mensagem. (e.g eth0,eth1);

Metric - custo da rota;

Flags - informações sobre a rota;

Ref - número de referência para a rota;

Use - contador do número de consultas à rota.

P:"What ARP messages, and associated MAC addresses, are observed and why?"

R: Como explicado anteriormente, o protocolo ARP trata de mapear endereços IP em endereços MAC. Visto que temos implementado rotas default no Tux2 e Tux3, a MAC address observada será a da gateway, e não a do destino final do pacote.

P:"What ICMP packets are observed and why?"

R: Os pacotes observados são do tipo ICMP **Request** e ICMP **Reply**, visto que todas as rotas estão adicionadas, então cada Tux consegue comunicar com os outros todos.

P:"What are the IP and MAC addresses associated to ICMP packets and why?"

R: Os endereços IP e MAC associados a pacotes ICMP são os das máquinas de origem e destino. Caso o Tux2 queira interagir com o Tux4, o pacote ICMP terá como IP e MAC de origem os respetivos endereços do Tux2, e como IP e MAC de destino os do Tux4. A situação torna-se menos linear quando tentamos fazer a análise de uma comunicação entre máquinas presentes em redes distintas. Assim sendo, como o Tux4 está presente nas duas e serve como router, tentando uma comunicação entre Tux3 e Tux2, serão enviados dois pacotes, em vez de um. O primeiro pacote será enviado do Tux3 para o Tux4, com os IPs e MACs respetivos. O segundo pacote será enviado do Tux4 para o Tux2, com os IPs e MACs respectivos.

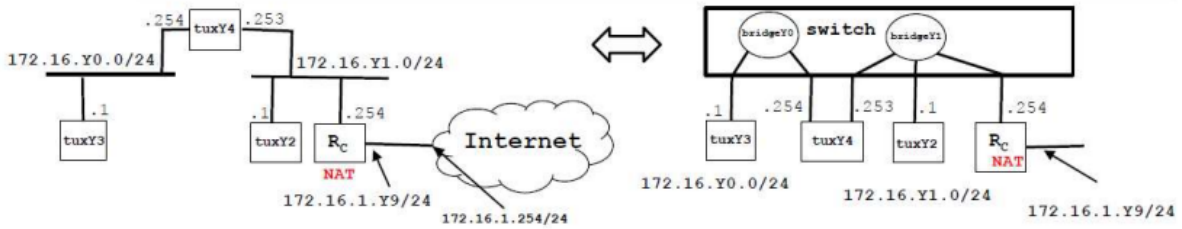
Experiência 4: Configurar um Router comercial e implementar NAT

Objetivos da experiência

O objetivo desta experiência é configurar um Router comercial e implementar NAT no mesmo.

Arquitetura da rede

Após a realização da experiência 4, a rede deverá ter a seguinte estrutura:



Como é possível observar, a grande diferença entre a configuração desta experiência para a anterior é a introdução de um Router comercial. Este Router comercial vai permitir a comunicação com a rede externa (Internet) no futuro, utilizando o mecanismo NAT e o mecanismo de DNS.

Comandos principais à configuração

Primeiramente, é necessário dar reset às configurações anteriores, a modos de garantir o correto funcionamento: /system reset-configuration

Depois, configuramos os endereços de IP associados às interfaces de rede do Router. A interface 1 tem de estar ligada ao PY.1, para haver acesso à internet:

/ip address add address=172.16.2.Y9/24 interface=ether1

/ip address add address=172.16.Y1.254/24 interface=ether2

Adicionar a interface 2 do router à bridgeY1, na switch:

/interface bridge port remove [find interface=etherXX]

/interface bridge port add interface=etherXX bridge=bridgeY1

Adicionar o Router como gateway default no Tux2 e Tux4:

route add default gw 172.16.Y1.254

Adicionar uma rota para 172.16.Y0.0/24 no Tux2:

ip route add 172.16.Y0.0/24 via 172.16.Y1.253

E por fim, adicionar rotas para 172.16.Y0.0/24 no Router:

/ip route add dst-address=0.0.0.0/0 gateway=172.16.2.254

/ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253

Análises das logs capturadas via wireshark

P: “How to configure a static route in a commercial router?”

R: Para configurar uma rota estática é necessário executar o seguinte comando:

ip route add <**IP**>/<**M**> via <**G**>

Sendo que o **IP** corresponde ao endereço de origem dos pacotes, o **M** à mask da sub-rede e o **G** corresponde ao gateway dos pacotes.

P: “What are the paths followed by the packets in the experiments carried out and why? “

R: Os caminhos podem ser observados através do comando traceroute <ip>. Assim é possível perceber o funcionamento da rede. Um pacote que saia do Tux3 com destino ao IP 8.8.8.8, terá o seguinte percurso:

Tux3 → Tux4 → RC → Servidor do Lab → Servidor da Feup → … (o resto do percurso incluí domínios a nós desconhecidos) → 8.8.8.8

Este percurso deve-se ao facto de existirem rotas em cada uma das máquinas e o IP forwarding estar ativado no Tux4.

P: “How to configure NAT in a commercial router?”

R: Após o reset das configurações do router, o NAT vem por default ativado. Podemos verificar isto correndo o comando /ip firewall nat print. Caso não se encontre ativado, podemos ativá-lo com o comando /ip firewall nat enable 0

P: “What does NAT do?”

R: NAT (Network address translation) é uma técnica que consiste em reescrever, utilizando uma tabela hash, os endereços IP de origem de um pacote, para que seja possível com apenas um endereço público servir vários endereços privados. Assim é possível poupar o espaço de endereçamento público, recorrendo a IPs privados. Faz um mapeamento baseado no IP interno e na porta local do computador. Com esses dois dados o NAT gera um número de 16 bits usando a tabela hash, e esse número é então escrito no campo da porta de origem.

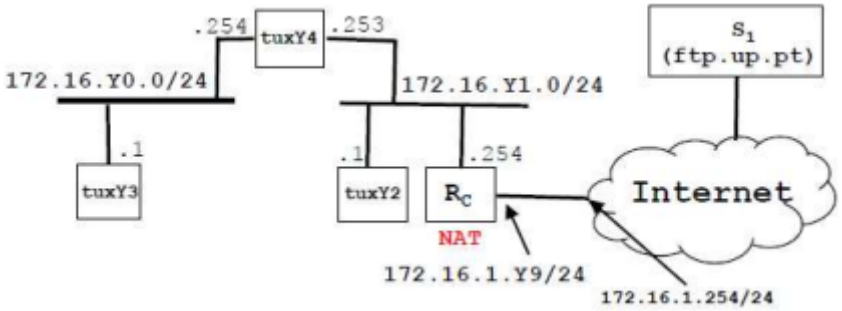
Experiência 5: DNS

Objetivos da experiência

O objetivo desta experiência consiste em adicionar um DNS (Domain Name System), que efetua a tradução de hostnames para endereços IP, e verificar como isso influencia a forma como as máquinas se conectam à Internet.

Arquitetura da rede

Após a realização da experiência 5, a rede deverá ter a seguinte estrutura:



A grande diferença entre esta e a última experiência, é o facto de o DNS estar implementado, permitindo comunicar com websites usando o seu url, em vez do seu IP. O grande benefício disto é que, além de não ser necessário mapear manualmente URLs para endereços IP, existe uma garantia de que um servidor mapeado num servidor DNS (ex: google.com → 8.8.8.8), mantém sempre o mesmo IP.

Comandos principais à configuração

Para esta experiência apenas é necessário editar um ficheiro de configuração do DNS. Assim sendo, correndo o comando sudo nano /etc/resolv.conf e introduzindo a linha nameserver 172.16.2.1, rapidamente se conclui a experiência.

Análises das logs capturadas via wireshark

P: “How to configure the DNS service at an host?”

R: É necessário editar o ficheiro resolv.conf e introduzir o ip correspondente, através do comando:

sudo nano /etc/resolv.conf
nameserver 172.16.2.1

P:"What packets are exchanged by DNS and what information is transported"

R: O host envia para o server um pacote com o hostname, esperando que seja retornado o seu respectivo endereço IP. O servidor responde por outro lado, com o IP correspondente a esse hostname.

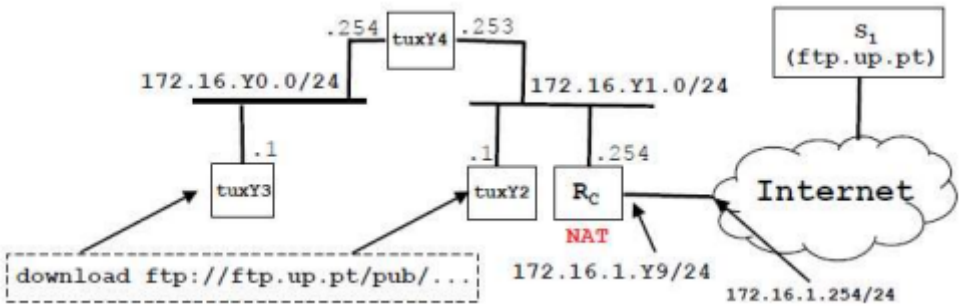
Experiência 6: Conexões TCP

Objetivos da experiência

O objetivo foi a observação do funcionamento e do comportamento da rede configurada por nós, e além disso testar a aplicação cliente FTP referida na Parte 1.

Arquitetura da rede

Após a realização da experiência 6, a rede deverá ter a seguinte estrutura:



Não existe diferença entre a rede desta experiência e a rede da experiência anterior, apenas introduzimos o nosso cliente FTP no Tux3.

Comandos principais à configuração

Não existe configuração necessária, apenas correr o cliente FTP feito por nós. Para tal, basta compilar o código com o comando gcc -o -Wall download download.c e executá-lo com o comando ./download <url ftp>

Análises das logs capturadas via wireshark

P: "How many TCP connections are opened by your ftp application? "

R: A nossa aplicação FTP abre 2 conexões TCP. Inicialmente uma para o envio de mensagens e respostas entre o cliente e o servidor (na porta 21), e a segunda para o envio dos dados do ficheiro pretendido (na porta calculada com os dados fornecidos após executar o comando pasv).

P: "In what connection is transported the FTP control information? "

R: A informação de controlo FTP é transportada na primeira ligação, indicada anteriormente (envio e receção de respostas).

P: “What are the phases of a TCP connection? “

R: Há 3 fases numa conexão TCP: início da ligação, transferência de dados e término da ligação.

P:” How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs? “

R: É um mecanismo para detectar e resolver possíveis erros que possam acontecer durante a transferência de dados. Neste mecanismo, o emissor envia os pacotes de dados e o receptor envia um ACK para confirmar que o pacote foi recebido com sucesso. Caso haja um erro, não haverá o envio do ACK e o emissor vai reenviar o pacote. Os “fields” relevantes que podem ser observados a partir dos logs são: o sequence number, acknowledge number, window size e flags.

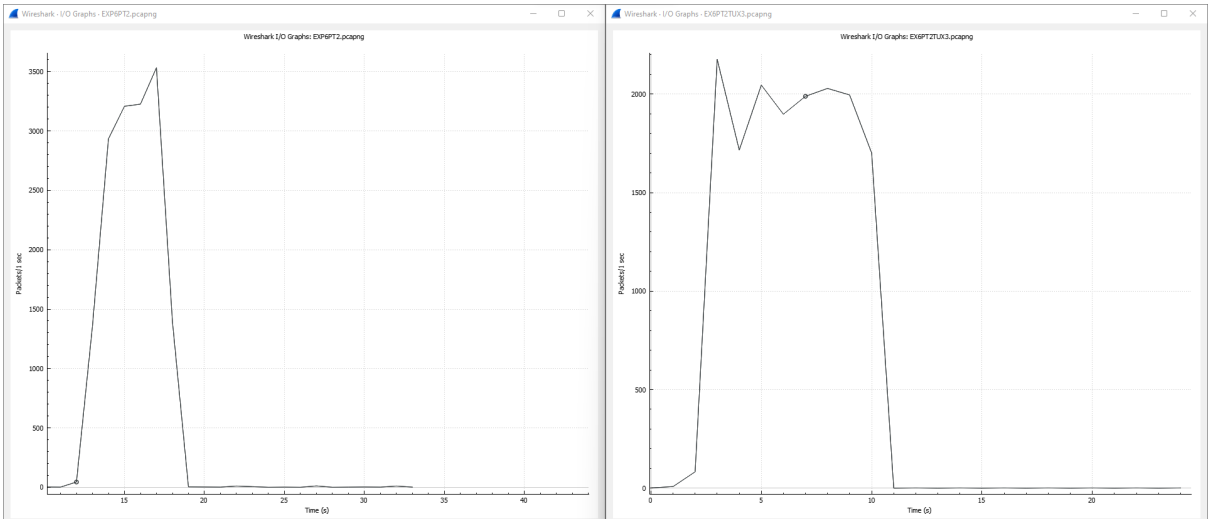
P:”How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according the TCP congestion control mechanism? “

R: O mecanismo de controlo de congestão TCP tem o objetivo de controlar o fluxo de dados numa conexão de modo a evitar o congestionamento da rede. Baseia-se no número de ACK recebidas por unidade de tempo calculada com os tempos de ida e volta, RTT (Round Trip Travel). Através deste mecanismo, o TCP consegue então prever o congestionamento da rede.

Por cada conexão há a Congestion Window, com o intuito de regular o tamanho da janela deslizante de transmissão de pacotes dependendo da congestão da conexão. Caso a congestão da rede diminua este valor é incrementado, caso contrário ele é decrementado. Por norma, o incremento é de uma 1 unidade a cada RTT. Quando existe um erro com o pacote, a congestion window reduz para metade, de forma a garantir o controlo do tráfego.

P: “Is the throughput of a TCP data connection disturbed by the appearance of a second TCP connection? How?”

R: Sim, a taxa de transferência de pacotes da primeira conexão TCP diminuí, visto que a segunda conexão também é atribuída uma capacidade de transferência. Eventualmente, estas acabam por estabilizar, ficando distribuídas de uma forma semelhante.



Isto é visível no gráfico anterior, onde à esquerda é mostrado o número de pacotes em trânsito por segundo quando existe apenas uma conexão TCP e à direita quando há duas conexões TCP. É também possível verificar na parte da direita que quando é aberta a segunda conexão TCP, o número de pacotes por segundo diminui temporariamente até que estabiliza.

Conclusão

Os objetivos para este projeto foram atingidos: a implementação da aplicação de download, a configuração de uma rede de computadores e a abordagem de conceitos como bridges, switches, routers etc.

Este projeto permitiu aos elementos do grupo ter uma melhor perceção e aprofundar conceitos sobre redes e sobre o protocolo FTP.

Referências

O trabalho foi realizado com recurso aos slides teóricos, ao guião fornecido, à ajuda do Exmo. Professor Orangel Contreras e a websites alheios da rede global (Internet).

Anexos

Download.h

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

#define SERVER_PORT 6000
#define SERVER_ADDR "192.168.28.96"
#define SOCK_READY 220



struct urlData
{
    char user[200];
    char password[200];
    char host_name[200];
    char file_path[200];
    char file_name[200];
    char ip[25];
    char ipNovo[25];
    int porta;
};


int argParser(char* input, struct urlData* data);

int startSocket(char* ip,int port);

int login(char* username, char* password, int fd);
```

```
int sendMessage(int fd, char* msg);

int readMessage(int fd);

int readPassive(int fd);

int passiveMode(int fd);

int fileMessage(int fd);

int writeFile(int fd);

char *strrev(char *str);
```

Download.c

```
#include "download.h"

struct urlData data;
int fd;
int newfd;


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

    if(argc < 2){
       fprintf(stderr, "Incorrect usage! Try: ftp://<host>/<url-path> or
ftp://<user>:<password>@<host>/<url-path>\n");
       exit(-1);
    }

    char* str = strdup(argv[1]);
    argParser(str, &data);

    fd = startSocket(data.ip,21);

    if(readMessage(fd) != SOCK_READY) {
        printf("Error opening Socket");
        return -1;
    }

    if(login(data.user,data.password,fd) != 0) {
```

```
        printf("Error Login");
        return -1;
    }

     if(passiveMode(fd) != 0) {
        printf("Error Passive");
        return -1;
    }

    newfd = startSocket(data.ip,data.porta);

    if(fileMessage(fd) != 0) {
        printf("Error getting file");
        return -1;
    }

    if(writeFile(newfd) != 0) {
        printf("Error saving file");
        return -1;
    }


    close(fd);
    close(newfd);
    return 0;
}

//ftp://<user>:<password>@<host>/<url-path> or ftp://<host>/<url-path>
int argParser(char* input, struct urlData* data) {
    struct hostent *h;
    char* type;
    type = strtok(input, ":");

    if(strcmp(type,"ftp") != 0 || type == NULL) {
        printf("Protocol isn't FTP!");
        return -1;
    }

    char* res = strtok(NULL,"\0");

    // means there is no username/pw
    if (strchr(res, '@') == NULL)
    {
        char* host_name = strtok(res,"/");
        strcpy(data->host_name,host_name);
        printf("Host: %s\n",host_name);
```

```
        strcpy(data->user,"anonymous");

        strcpy(data->password,"");

        char* path = strtok(NULL,"");
        strcpy(data->file_path,path);
        printf("Path: %s\n",path);

        char* name = strrev(strtok(strrev(path),"/"));

        printf("FileName: %s\n",name);
        strcpy(data->file_name,name);

        data->porta = 21;
    }
    else {
        char* user = strtok(res,":") + 2;
        strcpy(data->user,user);
        printf("User: %s\n",user);

        char* pw = strtok(NULL,"@");
        strcpy(data->password,pw);
        printf("PW: %s\n",pw);

        char* host_name = strtok(NULL,"/");
        strcpy(data->host_name,host_name);
        printf("Host: %s\n",host_name);

        char* path = strtok(NULL,"");
        strcpy(data->file_path,path);
        printf("Path: %s\n",path);

        char* name = strrev(strtok(strrev(path),"/"));
        printf("FileName: %s\n",name);
        strcpy(data->file_name,name);

        data->porta = 21;

    }

    if ((h = gethostbyname(data->host_name)) == NULL) {
        herror("gethostbyname()");
        exit(-1);
    }
    strcpy(data->ip,inet_ntoa(*((struct in_addr *) h->h_addr)));
```

```
    //printf("%s\n",data->ip);
    return 0;
}

int startSocket(char* ip,int port) {
    int sockfd;
    struct sockaddr_in server_addr;
    char buf[] = "Mensagem de teste na travessia da pilha TCP/IP\n";
    size_t bytes;

    /*server address handling*/
    bzero((char *) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ip);
    server_addr.sin_port = htons(port);

    /*open a TCP socket*/
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        return -1;
    }

    /*connect to the server*/
    if (connect(sockfd,(struct sockaddr *)
&server_addr,sizeof(server_addr)) < 0) {
        perror("connect()");
        return -1;
    }

    return sockfd;
}

int readMessage(int fd) {
    int code;
    FILE * sock = fdopen(fd,"r");

    char * buf;
    size_t currBytes = 0;


    while(getline(&buf,&currBytes,sock) > 0) {
        printf("%s",buf);
        if(buf[3] == ' ') {
            sscanf(buf, "%d", &code);
          break;
        }
```

```
    }

    printf("CODE: %d\n",code);

    return code;
}

int sendMessage(int fd, char* msg) {
    int st = send(fd,msg,strlen(msg),0);
    printf("Sent: %s",msg);
    if(st <= 0) {
        printf("Error Sending Message");
        return -1;
    }
    return 0;
}

int login(char* username, char* password, int fd) {
    char msg[200];
    sprintf(msg,"user %s\r\n", username);
    if(sendMessage(fd,msg) != 0) {
        printf("Error in Login Username");
        return -1;
    }
    if(readMessage(fd) != 331) {
        //printf("Error in Login Username");
        //return -1;
    };
    memset(&msg,0,sizeof(msg));
    sprintf(msg,"pass %s\r\n", password);
    if(sendMessage(fd,msg) != 0) {
        printf("Error in Login Password");
        return -1;
    }
    if(readMessage(fd) != 230) {
        printf("BAD PASSWORD");
        return -1;
    };
    return 0;
}

int readPassive(int fd) {
    int code;
    FILE * sock = fdopen(fd,"r");

    char * buf;
```

```
    size_t currBytes = 0;


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

    char* total = strtok(buf,"(");
    char* rest = strtok(NULL,""); // ip todo
    char* pt1,*pt2,*pt3,*pt4;
    char* porta1,*porta2;

    pt1 = strtok(rest,",");
    pt2 = strtok(NULL,",");
    pt3 = strtok(NULL,",");
    pt4 = strtok(NULL,",");

    porta1 = strtok(NULL,",");
    porta2 = strtok(NULL,")");

    data.porta = atoi(porta1)*256 + atoi(porta2);
    sprintf(data.ipNovo, "%s.%s.%s.%s", pt1, pt2, pt3, pt4);

    return 0;

}

int passiveMode(int fd) {
    char msg[200];
    sprintf(msg,"pasv\r\n");
    if(sendMessage(fd,msg) != 0) {
        printf("Error in passive mode");
        return -1;
    }

    readPassive(fd);
    return 0;
}

int fileMessage(int fd) {
    char msg[250];
    sprintf(msg,"retr %s\r\n",data.file_path);
```

```
    if(sendMessage(fd,msg) != 0) {
        printf("Error in file");
        return -1;
    }

    if(readMessage(fd) != 150) {
        printf("Error in File");
        return -1;
    }

    return 0;
}

int writeFile(int fd) {
    int fp;
    if ((fp = open(&data.file_name, O_WRONLY | O_CREAT, 0777)) < 0) {
        printf("Error opening file\n");
        return -1;
    }

    char buf[256];
    int numBytesRead;

    while((numBytesRead = read(newfd, buf, 256)) > 0) {
        if (write(fp, buf, numBytesRead) < 0) {
            printf("Error writing to file!\n");
            return -1;
        }
    }

    if (close(fp) < 0) {
        printf("Error closing file\n");
        return -1;
    }

    return 0;
}

char *strrev(char *str)
{
      char *p1, *p2;

      if (! str || ! *str)
            return str;
      for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
      {
```

```
            *p1 ^= *p2;
            *p2 ^= *p1;
            *p1 ^= *p2;
      }
      return str;
}
```

Protocolo Configuração Rede

```
Desligar TODOS os cabos

Ligar eth0 do Tux2, Tux3, Tux4 a qualquer porta da switch exceto a
CONSOLE e a 1ª

Ligar S0 do Tux2 ao RS232->Cisco e o Cisco->RS232 à console da switch

Dar reset à configuração de ethernet em TODOS os computadores:
systemctl restart networking

Tux2:
      ifconfig eth0 172.16.Y1.1/24 (ou ifconfig eth0 172.16.Y1.1 netmask
255.255.255.0)

Tux3:
      ifconfig eth0 172.16.Y0.1/24

Tux4:
      ifconfig eth0 172.16.Y0.254/24

No Tux2, abrir o GTKterm com baudrate a 115200, carregar no ENTER e dar
reset à configuração:
      /system reset-configuration
User: admin
Pass: (blank)

Criar 2 bridges:
      /interface bridge add name=bridgeY0
      /interface bridge add name=bridgeY1

Remover as ports ethernet onde os PCs estão ligados:
      /interface bridge port remove [find interface=etherXX] (Tux2)
      /interface bridge port remove [find interface=etherXX] (Tux3)
```

```
      /interface bridge port remove [find interface=etherXX] (Tux4)

Adicionar as ports ethernet às bridges criadas:
      /interface bridge port add interface=etherXX bridge=bridgeY1
(Tux2)
      /interface bridge port add interface=etherXX bridge=bridgeY0
(Tux3)
      /interface bridge port add interface=etherXX bridge=bridgeY0
(Tux4)

Verificar que as bridges foram criadas:
      /interface bridge port print brief

Em TODOS os PCs, desabilitar o ignore broadcasts:
      echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

Ligar eth1 do Tux4 a qualquer porta da switch exceto a CONSOLE e a 1ª

Configurar o ip na Tux4:
      ifconfig eth1 172.16.Y1.253/24

Adicionar Tux4 à bridge1:
      /interface bridge port remove [find interface=etherXX] (Tux4)
      /interface bridge port add interface=etherXX bridge=bridgeY1
(Tux4)

Habilitar IP forwarding no Tux4:
      echo 1 > /proc/sys/net/ipv4/ip_forward

Adicionar routes no Tux2 e Tux3:
      route add default gw 172.16.Y1.253 (Tux2)
      route add default gw 172.16.Y0.254 (Tux3)

----------------- TASK 4--------------------

Configurar um router comercial e implementar NAT:
Desligar Cisco->RS232 da consola da switch e ligar ao MTIK do router

Re-abrir o GTKterm com baudrate 115200

Reset router config:
      /system reset-configuration

Ligar a ether1 do Router ao PY.1

Configurar IP address:
```

```
      /ip address add address=172.16.2.Y9/24 interface=ether1
      /ip address add address=172.16.Y1.254/24 interface=ether2

Ligar ether2 do Router à switch e colocar essa porta na bridgeY1:
      /interface bridge port remove [find interface=etherXX] (switch)
      /interface bridge port add interface=etherXX bridge=bridgeY1
(switch)

No Tux4 e Tux2 adicionar o router como default gateway
      route add default gw 172.16.Y1.254

Adicionar routes para 172.16.Y0.0/24 no Tux2:
      ip route add 172.16.Y0.0/24 via 172.16.Y1.253

Adicionar routes para 172.16.Y0.0/24 no Router:
      /ip route add dst-address=0.0.0.0/0 gateway=172.16.2.254
      /ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253

Se NAT estiver disabled (/ip firewall nat print) dar enable no router:
/ip firewall nat enable 0

----------------- TASK 5--------------------

Configurar DNS no Tux2, Tux3, Tux4:
      sudo nano /etc/resolv.conf
      nameserver 172.16.2.1

Testar com:
      ping www.google.com
      ping www.ojogo.pt
      Abrir browser em www.fcporto.pt
```