U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

2º Trabalho Laboratorial

FTP + Lab Network Configuration

Redes de Computadores

Grupo 4, Turma 2

Realizado por:

- Diogo Miguel Chaves dos Santos Antunes Pereira – up201906422
- Joana Teixeira Mesquita – up201907878

Janeiro de 2022

Sumário

Este projeto visa a implementação de um aplicação de download que utiliza o protocolo de dados FTP, bem como a configuração da rede no local onde esta é executada.

Introdução

Este projeto está divido em duas partes. A primeira parte diz respeito ao desenvolvimento de uma aplicação de download utilizando o protocolo de dados FTP e consiste na explicação detalhada do funcionamento desta, bem como os resultados obtidos. A segunda parte consistiu num conjunto de experiências com o objetivo de configurar o router para ter acesso à rede e assim poder testar a aplicação de download criada. Nesta descrevemos os objetivos e procedimentos das experiências bem como conclusões e explicações teóricas de alguns dos termos do procedimento.

Parte 1 – Aplicação de download

O objetivo da primeira parte do trabalho laboratorial é o desenvolvimento de uma aplicação de download FTP que recebe um ficheiro usando o protocolo de dados FTP.

Assim sendo, a nossa aplicação recebe um argumento em sintax URL com o seguinte formato:

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

Arquitetura da aplicação de download

A aplicação de download começa por, ao receber o argumento, verificar se este começa com *ftp://* e posteriormente guarda o resto do argumento numa variável, para ser usada mais tarde.

De seguida é chamada a função *void parseinput(char* input,int size, char* password, char* user, char* address, char* port, char* path)* para retirar a informação de que precisamos do formato URL em que a recebemos e para colocar os comandos que terão de ser enviados para socket antes desta.

Com estas informações começamos por encontrar o endereço IP do host ao qual nos queremos conectar através da função *void getAddress(char* address)* e com este podemos configurar e estabelecer a conexão ao host. É necessário ainda criar um file descriptor para leitura através de *fdread*.

Agora com a ligação estabelecida é apenas necessário ler e escrever para o servidor alternadamente, até esta ligação ser colocada em modo passivo com o comando *pasv*, tendo em atenção possíveis erros na password e username inseridos.

Assim que a nossa primeira conexão entra em modo passivo é necessário calcular a porta para a criação de uma segunda conexão, para este cálculo utilizamos os últimos dois números enviados pelo servidor, quando entramos em modo passivo, e multiplicamos o primeiro por 256 somando o segundo na função *int get_termB_port(FILE* fd, char* message)*. Com este port calculado é aberta com ele nova conexão para o host e a partir da nossa primeira conexão é enviado o path do ficheiro a receber levando a segunda conexão a dar print deste antes de terminar.

Prova de um download efetuado com sucesso

Utilizando o programa criado conseguimos fazer pedidos de download ao servidor do netlab1, como se pode ver nas figuras 1 e 2.

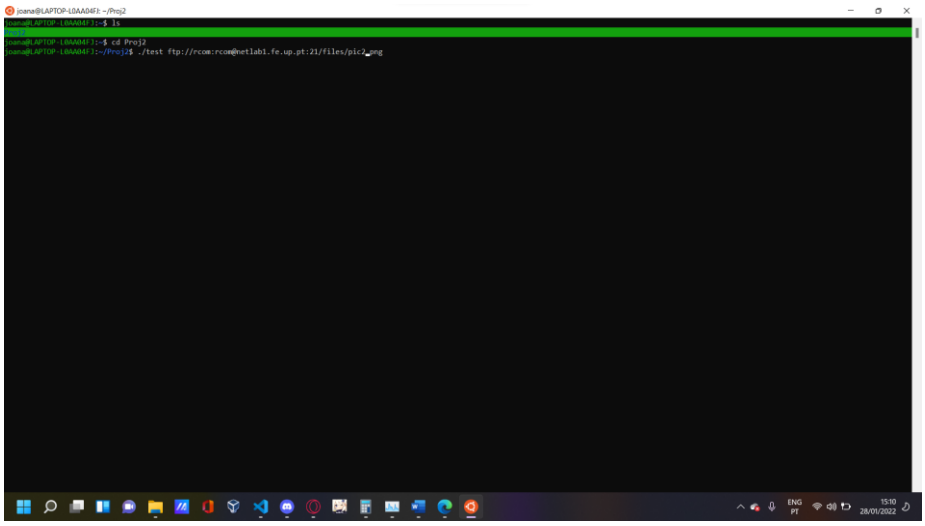

Figura 1 – Pedido ao servidor netlab1 de download da imagem pic2.png presente no diretório files.

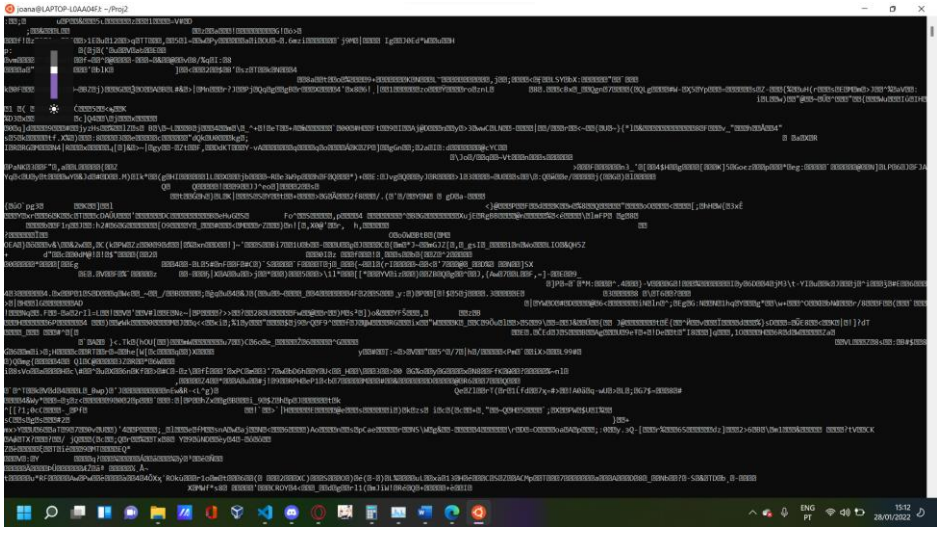

Figura 2 – Print do conteúdo da imagem pic2.png recebida.

Parte 2 – Análise e configuração da rede

Experiment 1/Guide 1 - IP Config

<u>Arquitetura da Rede</u>

No final desta experiência o tux63 e o tux64 (em que o Y é 6 devido a encontrarmo-nos na bancada 6) devem estar ligados ao switch e ter os IPs configurados conforme o que está exemplificado na figura 3.

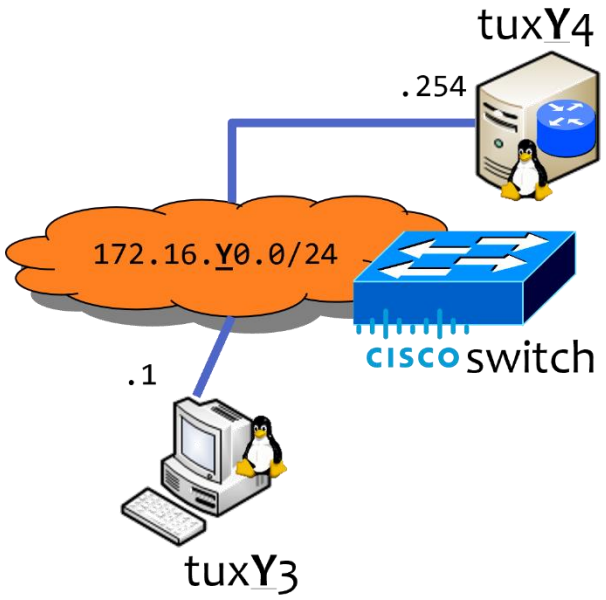

Figura 3 - Objetivo final do laboratório 1

Objetivos

Os objetivos desta experiência são:

- Definir os endereços IP do tux63 e do tux64 e fazer com que estes tenham a rede 172.16.60.0/24 definida de maneira a que possam comunicar entre si.

Procedimento

Começamos por conectar a entrada eth0 do tux63 e do tux64 ao switch e conectar a entrada S0 do tux63 à consola do switch. A partir da consola do switch fizemos a limpeza das configurações já presentes, para começarmos do zero.

Após isto tanto no tux63 como no tux64 utilizamos os comandos:

- `ifconfig eth0 down` - para retirarmos qualquer IP que já pudesse estar configurado na entrada eth0;
- `ifconfig eth0 172.16.60.1/24 no tux63 e ifconfig eth0 172.16.60.254/24 no tux64` - para configurarmos o IP address e a net mask do eth0 de cada tux;
- `ifconfig` - para vermos os ips configurados em cada tux
-

No final, ao usar `ifconfig` obtivemos os seguintes resultados:

- tux63
  - `MAC: ether 00:22:64:a7:32:ab txqueuelen 1000 (Ethernet)`
  - `IP: inet 172.16.60.1 netmask 255.255.255.0 broadcast 172.16.60.255`

- tux64
  - `MAC: ether 00:21:5a:5a:75:bb txqueuelen 1000 (Ethernet)`
  - `IP: inet 172.16.60.254 netmask 255.255.255.0 broadcast 172.16.60.255`

De seguida utilizámos o comando `ping` para verificar a conexão entre o tux63 e o tux64 e verificámos que esta estava estabelecida.

Por fim verificámos com o comando `route -n` o conteúdo da forwarding table, com o comando `arp -a` o conteúdo da tabela ARP de cada tux, e com o comando `arp -d <ipaddress>` apagámos todas essas entradas antes de iniciar o wireshark no tux63 e dar ping deste para o tux64.

Análise

O ARP ou Address Resolution Protocol é um protocolo que conecta o IP (Internet Protocol), que se altera constantemente, a uma endereço de posição física da máquina conhecido como o endereço MAC ou Media Access Control. Este processo é importante pois o tamanho dos endereços do IP e do MAC variam, pelo que é necessário fazer uma tradução destes para que os sistemas se possam reconhecer.

Assim, quando um packet com informação é enviado para uma máquina específica, a gateway pede ao ARP para encontrar o endereço MAC que corresponde ao IP de maneira a que este possa ser enviado para a máquina correta. Isto é feito através dos ARP packets que são mensagens básicas utilizadas para transportar address resolution requests e responses utilizados pelo ARP. Estes contêm os endereços MAC e IP das máquinas que recebem e enviam o packet.

Por outro lado, quando utilizamos o comando ping, estamos a enviar ICMP (Internet Control Message Protocol) packets, com o objetivo de gerar uma resposta da máquina para a qual são enviados e descobrir assim se existe uma ligação establecida. Estes packets contêm os endereços MAC e IP tanto da máquina que envia o ping como da que o recebe .

Tanto os ARP como os IP packets podem ser identificados no cabeçalho das Ether Frames. Se no campo EtherType deste o valor for 0x0806 então refere-se a um protocolo ARP, se pelo contrário tiver o valor de 0x0800 podemos concluir que tem um protocolo ipv4 ou seja, contém um IP. Por fim, para verificar se esta é um ICMP packet é necessário que a frame seja do tipo IP e que o valor da header do IP seja 1. Estas frames não precisam necessariamente de conter o seu tamanho pois o seu fim é sinalizado pela perda do carrier ou por um símbolo especial ou sequência.

Por fim, a loopback interface é um método de identificação de dispositivos com várias vantagens, uma vez que enquanto que o endereço de qualquer interface pode ser utilizado para ver se a máquina está online estes podem ser alterados ou apagados, mas o endereço de loopback nunca muda.

Quando apagamos as entradas da tabela arp na experiência, estas são recriadas assim que o comando ping é chamado novamente, nunca impedindo a conexão entre os dois computadores já que elas não estarem em cache apenas significa que têm de ser calculadas novamente.

Experiment 2/Guide 2 - Virtual LANs

Arquitetura da Rede

No final desta experiência as VLANS vlan60 e vlan61 devem estar configuradas no switch e o tux62 deve estar configurado e ligado ao switch como se pode observar na figura 4.

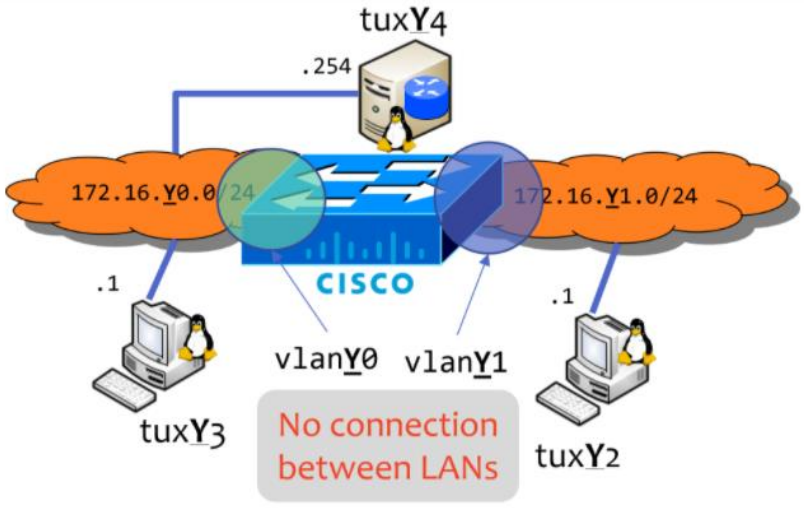

Figura 4 - Objetivo final do laboratório 2

Objetivos

Os objetivos desta experiência são:

- Definir o endereço ip do tux62 e fazer com que este tenha a rede 172.16.61.0/24 definida;
- Criar as vlans 60 e 61 no switch e adicionar-lhes as portas correspondentes.

Procedimento

Começamos por conectar o tux62 ao switch através da sua entrada eth0 e posteriormente configuramos o endereço IP do tux62 com os seguintes comandos:

- `ifconfig eth0 down` - para retirarmos qualquer ip que já pudesse estar configurado na entrada eth0;
- `ifconfig eth0 172.16.61.1/24` - para configurarmos o IP address e a net mask do eth0;
- `ifconfig` - para vermos o ip configurado.

Após usar o comando ifconfig para verificar se o ip tinha sido bem configurado obtivemos o seguinte resultado:

- tux62
  - `MAC: ether 00:21:5a:61:2d:df txqueuelen 1000 (Ethernet)`
  - `IP: inet 172.16.61.1 netmask 255.255.255.0 broadcast 172.16.61.255`

Por último, através do tux63, utilizamos o comando `ping` para o tux64 e para o tux62 individualmente. Após obtermos os resultados, fizemos um `ping broadcast` a partir do tux63 e, de seguida, a partir do tux62.

Análise

Configurar a vlan **Y**0 consiste na sua criação e em adicionar portas do switch a esta. Para isto é necessário:

- Inserir na consola da switch o comando `configure terminal`;
- Criar a vlan com os comandos:
  - `vlan Y0`;
  - `end.`
- Adicionar à vlan as portas corretas do switch com os comandos:
  - `configure terminal;`
  - `interface fastethernet 0/X` em que X é a porta a adicionar;
  - `switchport mode access;`
  - `switchport access vlan Y0` para adicionarmos a porta X à vlan **Y**0;
  - `end.`

No final desta experiência passam a existir dois domínios de broadcast. Chegamos a esta conclusão porque o `ping broadcast` do tux63 foi apenas detetado no tux64 e o `ping broadcast` do tux62 não foi detetado por nenhum outro tux. Como dentro de um domínio de broadcast todos os nodes têm acesso uns aos outros, estes dois broadcasts não terem atingido todos os tux significa que estão a ocorrer em dois dominios de broadcast diferentes.

Experiment 3/Guide 3 - Router Configuration

Cisco Router Configuration

Em primeiro lugar, analisámos o ficheiro de configurações para o cisco router que está a fazer NAT e routing nas interfaces como se pode ver na figura 5.

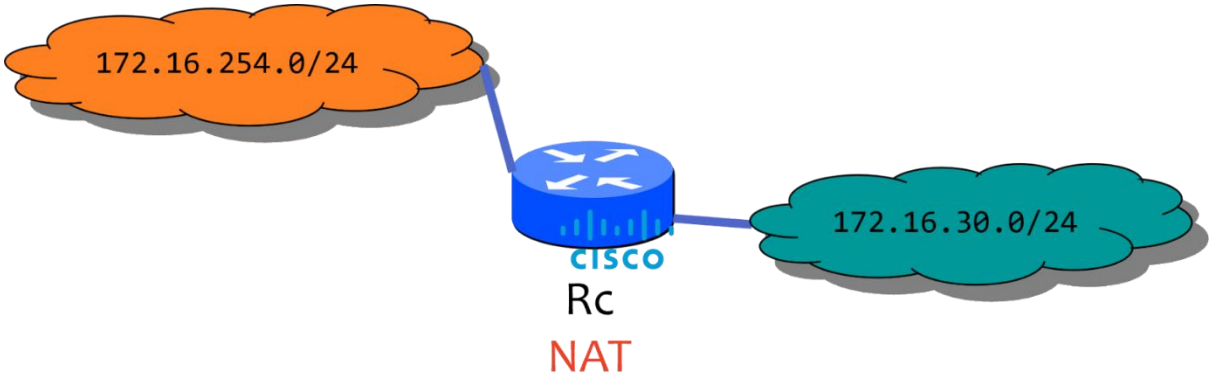

Figura 5 – Diagrama de um cisco router.

A partir do ficheiro chegamos à conclusão que:

- O nome do router é gnu-rtr1;
- Temos dois Ethernet ports disponíveis e são do tipo fast-ethernet;
- Temos os endereços de IP e netmasks seguintes configuradas:
  - ip address 172.16.30.1 255.255.255.0;
  - ip address 172.16.254.45 255.255.255.0;
- Temos as seguintes routes configuradas:
  - ip route 0.0.0.0 0.0.0.0 172.16.254.1;
  - ip route 172.16.40.0 255.255.255.0 172.16.30.2;

Quanto às configurações NAT do router podemos observar que:

- A interface interface FastEthernet0/1 é a interface conectada à internet;
- Só o endereço 172.16.254.45 é que se encontra disponível para NATing;
- O router está a usar overload como podemos ver nas linhas:
  - ip nat pool ovrld 172.16.254.45 172.16.254.45 prefix-length 24
  - ip nat inside source list 1 pool ovrld overload

Análise

NAT ou network address translation é uma forma de mapear vários endereços privados para um endereço público, antes de transferir informação. Por exemplo, se um computador quiser enviar um pedido de informação para a internet, este envia um packet ao qual o router muda o endereço privado local para um endereço público, caso contrário o servidor que recebe o pedido de informação não saberia para onde a enviar.

Uma static route dá-nos acesso a um conjunto de caminhos de routing fixos na rede. Para a configurar num router comercial seria necessário correr os seguintes comandos num modo privilegiado de execução:

- `ip route prefix mask {ip-address | interface-type interface-number [ip-address]}`
- `end`

Para configurar NAT num router comercial seria necessário correr a seguinte sequência de comandos:

- `conf t`
- `interface gigabitethernet 0/0 *`
- `ip address 172.16.y1.254 255.255.255.0`
- `no shutdown`
- `ip nat inside`
- `exit`

- `interface gigabitethernet 0/1*`
- `ip address 172.16.1.y9 255.255.255.0`
- `no shutdown`
- `ip nat outside`
- `exit`
- `ip nat pool ovrld 172.16.1.y9 172.16.1.y9 prefix 24`
- `ip nat inside source list 1 pool ovrld overload`
- `access-list 1 permit 172.16.y0.0 0.0.0.7`
- `access-list 1 permit 172.16.y1.0 0.0.0.7`
- `ip route 0.0.0.0 0.0.0.0 172.16.1.254`
- `ip route 172.16.y0.0 255.255.255.0 172.16.y1.253`
- `end`

DNS configs

Em primeiro lugar foi adicionada ao ficheiro /etc/hosts a entrada `142.250.200.142 youtubas` e de seguida foi enviado um ping para o endereço enquanto os packets eram capturados com o wireshark.

Posteriormente fizemos uma captura no wireshark enquanto fazíamos ping a `enisa.europa.eu` .

Por fim adicionámos a entrada `nameserver 9.9.9.9` ao ficheiro `/etc/resolv.conf` e iniciámos nova captura no wireshark enquanto dávamos ping ao `parlamento.pt`.

Análise

DNS ou Domain Name System é um processo que converte nomes de hosts para endereços IP que o computador consegue compreender. Para configurar um serviço DNS num host específico é necessário fazer alterações ao ficheiro /etc/hosts da máquina inserindo uma linha com o IP e o domínio do host a configurar. O DNS possui ainda dois tipos distintos de pacotes DNS Query que contém o hostname do qual queremos saber o IP e DNS Reply que contém o IP pedido na query.

Linux Routing

Primeiro corremos o comando `route -n` para vermos a forwarding table e dessa retirámos a default gateway:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	10.0.2.1	0.0.0.0	UG	100	0	0	Enp0s3

De seguida, eliminámos a default gateway da tabela e usámos o comando `traceroute -n 104.17.113.188` para verificar a conectividade, no entanto este não foi capaz de alcançar a rede.

Através do comando `sudo route add 104.17.113.188 gw 10.0.2.1 enp0s3` adicionámos uma nova route à nossa forwarding table e, voltámos a chamar traceroute para o IP, agora com resultados.

Por fim usámos o comando `traceroute -n who.int` que não nos premitiu obter uma conexão, isto porque a máquina não tem uma route definida para o servidor DNS. Para a definir adicionámos uma entrada ao ficheiro /etc/resolv.conf.

Análise

Uma route é o caminho que um determinado packet tem de percorrer para chegar ao seu destino. Se estas routes estiverem bem definidas para o destino a que queremos chegar, os pacotes ICMP enviados e recebidos são todos ou do tipo request ou do tipo reply, caso contrário estes packets são do tipo Destination Unreachable. Estes packets têm os endereços IP e MAC das máquinas que os recebem e enviam o que indica de onde vieram e qual é o seu destino.

Arquitetura do Linux Router

No final desta experiência todos os tuxs devem estar ligados ao switch e ao tux64 assim como se pode ver na figura 6.

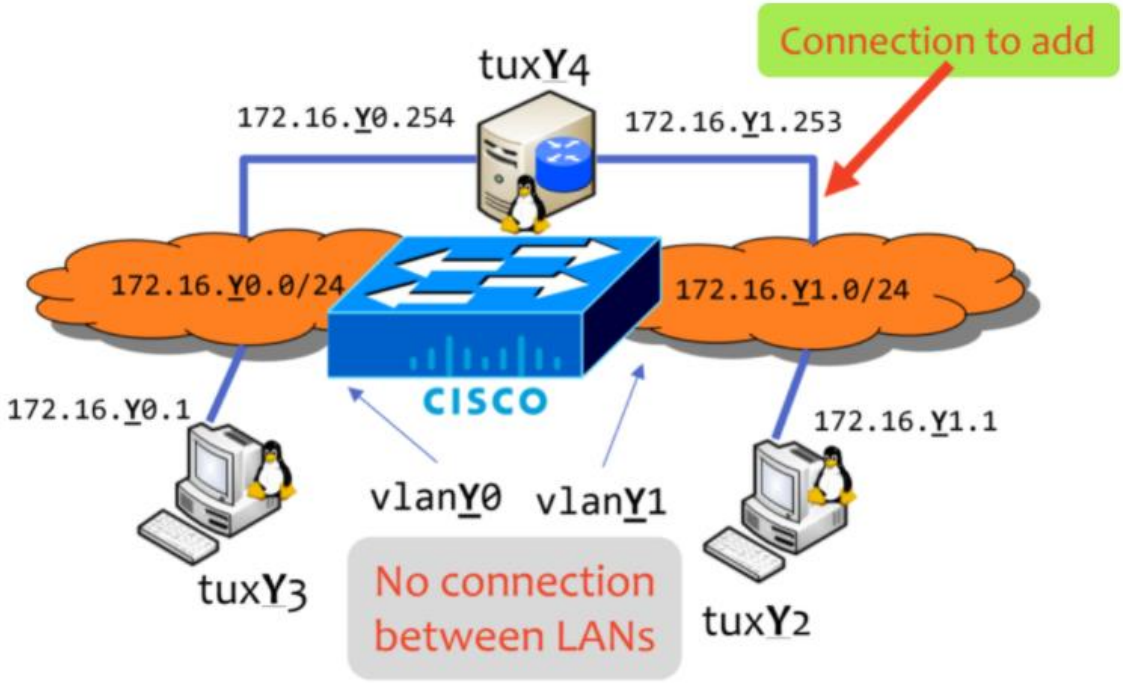

Figura 6 – Objetivo da primeira parte da experiência 4.

Objetivos

Os objetivos desta experiência são:

- fazer uma ligação do tux64 ao switch e colocá-la na vlan 61;
- configurar os tux62 e tux63 para que possam estabelecer uma conexão.

Procedimento

Primeiro, conectámos a entrada eth1 do tux64 ao switch e adicionámo-la à vlan61 utilizando os comandos da switch da experiencia 2. De seguida configuramos o ip da porta eth1 do tax64 com o comando:

- `ifconfig eth1 172.16.61.253/24`;

Ativámos ainda o IP Forwarfing e desativamos o ICMP echo ignore broadcast no tux64 com os seguintes comandos:

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

De seguida, verificámos os IPs e os Macs do tux64 para eth0 e eth1, obtendo os seguintes resultados:

```
eth0:
      MAC: ether 00:21:5a:5a:75:bb txqueuelen 1000 (Ethernet)
      IP: inet 172.16.60.254 netmask 255.255.255.0 broadcast 172.16.60.255
eth1:
      MAC: ether 00:08:54:71:73:ed txqueuelen 1000 (Ethernet)
      IP: inet 172.16.61.253 netmask 255.255.255.0 broadcast 172.16.61.255
```

Para o tux62 e o tux63 se conseguirem alcançar utilizámos o tux64 como intermediário configurando routes em ambos, com os comandos:

- no tux63:
  - `ip route add 172.16.61.0/24 via 172.16.60.254`
- no tux62:
  - `ip route add 172.16.60.0/24 via 172.16.61.253`

Após configurármos as routes, enviámos um ping através do tux63 aos outros tuxes para verificar a conectividade e começámos uma captura no wireshark do tux64 enquanto enviávamos pings do tux63 ao tux62.

Análise

Os routes nos tuxes são os seguintes:

tux4:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
172.16.60.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.61.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1

tux3:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.60.1	0.0.0.0	UG	0	0	0	eth0
172.16.2.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
172.16.60.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
172.16.61.0	172.16.60.254	255.255.255.0	UG	0	0	0	eth0

tux2:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.61.254	0.0.0.0	UG	0	0	0	eth0
172.16.1.0	172.16.2.254	255.255.255.0	UG	0	0	0	eth1
172.16.2.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
172.16.60.0	172.16.61.253	255.255.255.0	UG	0	0	0	eth0
172.16.61.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

Após as configurações efetuadas, podemos ver os routes que inserimos nas tabelas de routes de cada tux (as com Destination 172.16.6X.0), significando que existe contacto entre as máquinas. As routes que contêm gateway 0.0.0.0 foram criadas quando defenimos os IPs das máquinas uma vez que estes as colocam na mesma rede e os restantes foram criados durante este laboratório e visam indicar uma forma de passagem entre o tux62 e o tux63 (que previamente não conseguiam establecer qualquer conexão) através do tux64 que se encontra ligado a ambos.

Cada forwarding table contém informação do destino, da gateway, da genmask, as flags, a metric, o ref, o use e o interface

- tux63
  - MAC: ether 00:22:64:a7:32:ab txqueuelen 1000 (Ethernet)
  - IP: inet 172.16.60.1 netmask 255.255.255.0 broadcast 172.16.60.255
- tux64
  - MAC: ether 00:21:5a:5a:75:bb txqueuelen 1000 (Ethernet)
  - IP: inet 172.16.60.254 netmask 255.255.255.0 broadcast 172.16.60.255
  - MAC: ether 00:08:54:71:73:ed txqueuelen 1000 (Ethernet)
  - IP: inet 172.16.61.253 netmask 255.255.255.0 broadcast 172.16.61.255
- tux62
  - MAC: ether 00:21:5a:61:2d:df txqueuelen 1000 (Ethernet)
  - IP: inet 172.16.61.1 netmask 255.255.255.0 broadcast 172.16.61.255.

Através das ARP messages e MACs observados podemos concluir que os tuxes estão todos conectados.

Os pacotes ICMP contêm informação dos endereços IP e MAC das máquinas que os recebem e enviam, o que indica de onde vieram e qual é o seu destino. Assim, caso os routes estejam bem definidos para o seu destino, os pacotes ICMP enviados e recebidos são ou do tipo request ou do tipo reply, caso contrário são do tipo Destination Unreachable.

Arquitetura do Cisco Router

No final desta experiência o cisco deve estar ligado à vlan correta e ligado ao router como exemplificado na figura 7.

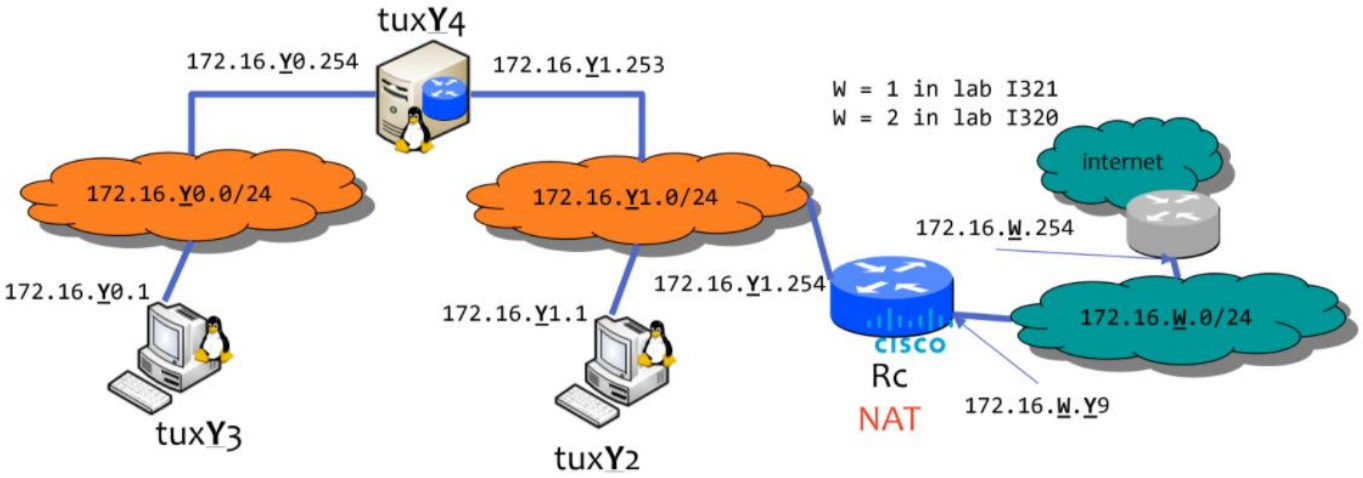

Figura 7 – Objetivo da segunda parte da experiência 4.

Objetivos

Os objetivos desta experiência são:

- conectar o cisco à netword 172.16.61.0/24;
- conectar o cisco ao router;
- colocar o router na vlan correta.

Procedimento

Obtendo os comandos do powerpoint fornecido, com Y = 6 e W = 2, entramos no config mode do terminal do router e executamos a seguinte sequência de comandos:

- `conf t`
- `interface fastethernet 0/0`
- `ip address 172.16.61.254 255.255.255.0`
- `no shutdown`
- `ip nat inside`
- `exit`

- `interface fastethernet 0/1`
- `ip address 172.16.2.69 255.255.255.0`
- `no shutdown`
- `ip nat outside`
- `exit`

- `ip nat pool oveld 172.16.2.69 172.16.2.69 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 0.0.0.0 172.16.2.254`
- `ip route 172.16.60.0 255.255.255.0 172.16.61.253`
- `end`

Para verificar conectividade, fizemos `ping` do Cisco Router para todos os tuxes, para 172.16.2.254 e para 104.17.113.188.

Para configurármos o tux62 e o tux64, adicionámos-les gateways para o Cisco Router através do comando `ip route add default via 172.16.61.254`.

Por fim, fizemos um ping do tux63 para 172.16.2.254 e do tux63 para 104.17.113.188.

Análise

Quando se realiza o ping ao IP 104.17.113.188 e ao IP 172.16.2.254 a partir do tux63 estes passam do tux63 para o tux64 e só depois para o tux62 (visto que este e o tux63 não estão ligados diretamente) e depois do tux62 é que estes packets chegam aos IP 104.17.113.188 e 172.16.2.254.

Conclusão

Este projeto fornece bases sobre o funcionamento de ciscos e routers bem como a forma de alterar as suas configurações e criar ligações entre computadores a partir destes. Esta é uma área de estudo essencial visto que nos dias que correm a transmição rápida e fiel de informação tanto dentro de sistemas de computadores como para o exterior é de extrema importância para os vários setores da sociedade.

Tal como no primeiro projeto uma das dificuldades que tivemos foi o acesso ao laboratório que foi ainda mais agravado no desenvolvimento deste, devido ao facto de a segunda parte do projeto necessitar quase na totalidade de ser desenvolvido no laboratório.

Anexo

Anexo I – Código Fonte

```
/**      (C)2000-2021 FEUP
 *       tidy up some includes and parameters
 * */

#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <sys/select.h>

#include <string.h>

#define SERVER_PORT 21
#define SERVER_ADDR "192.168.28.96"

void getAddress(char* address){

    struct hostent *h;

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

    strcpy(address, inet_ntoa(*((struct in_addr *) h->h_addr)));
}

char* read_from_socket(FILE* fd, char* r){
    char* i = fgets(r,250,fd);
    printf("%s",r);
    return i;
}

int get_termB_port(FILE* fd, char* message){
    int num = 0, count=0;
    char* num2 = (char *) malloc(3);
    char* num1 = (char *) malloc(3);
    int e;
    for (int i = 0; i <strlen(message);i++){
        if (message[i] == '('){
            num = 1;
        }
        if (num == 1){
```

```
            if (message[i] == ')'){
                break;
            }
            if (message[i] == ','){
                count ++;
                i++;
                e= 0;
            }
            if (count == 4){
                num1[e] = message[i];

                e++;
            }
            else if (count == 5){
                num2[e] = message[i];
                e++;
            }

        }
    }

    int port = atoi(num1) * 256 + atoi(num2);
    free(num1);
    free(num2);

    return port;
}

//ftp://anonymous:qualquer-password@ftp.up.pt:21/pub/kodi/timestamp.txt

void parseinput(char* input,int size, char* password, char* user, char* address, char*
port, char* path){
    int i = 0;

    user[0] ='u';
    user[1] ='s';
    user[2] ='e';
    user[3] ='r';
    user[4] =' ';

    for (int e = 5; i < size; i++, e++){
        if (input[i] == ':'){
            user[e] = '\n';
            i++;
            break;
        }
        user[e] = input[i];
    }

    password[0] ='p';
    password[1] ='a';
    password[2] ='s';
```

```
    password[3] ='s';
    password[4] =' ';

    for (int e = 5; i < size; i++, e++){
        if (input[i] == '@'){
            password[e] = '\n';
            i++;
            break;
        }
        password[e] = input[i];
    }


    for (int e = 0; i < size; i++, e++){
        if (input[i] == ':'){
            i++;
            break;
        }
        address[e] = input[i];
    }

    for (int e = 0; i < size; i++, e++){
        if (input[i] == '/'){
            i++;
            break;
        }
        port[e] = input[i];
    }

    path[0] ='r';
    path[1] ='e';
    path[2] ='t';
    path[3] ='r';
    path[4] =' ';

    for (int e = 5; i <= size; i++, e++){
        if (i == size){
            path[e] = '\n';
        }
        else path[e] = input[i];
    }


    /*printf("user : %s\n",user);
    printf("password: %s\n",password);
    printf("address: %s\n",address);
    printf("port: %s\n",port);
    printf("path: %s\n",path);*/

}

int write_to_socket(int sockfd, char* str){
```

```
    int bytes;
    int count = strlen(str);

    bytes = write(sockfd, str, count);

    if (bytes > 0)
        printf("%s",str);
    else {
        perror("write()");
        return(-1);
    }

    return 0;
}

int close_sockets(int sockfd,int sockfd2){

    if (close(sockfd2)<0) {
        perror("close()");
        return(-1);
    }

    if (close(sockfd)<0) {
        perror("close()");
        return(-1);
    }
}

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

    char big[] = "ftp://";
    char input[255];
    char read[250];

    if (argc != 2 || strncmp(big, argv[1], 6) != 0) {
        fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>:<port>/<url-path>\n",
argv[0]);
        exit(-1);
    }

    strcpy(input,argv[1]+6);

    int size = (int)strlen(argv[1]);

    int sockfd, sockfd2;
    struct sockaddr_in server_addr;
    char address[250] = {}, passive[] = "pasv\n", user[250] = {},password[250] = {},
path[255]={}, port[255] = {};


    parseinput(input,size-6,password,user,address,port,path);
```

```
    getAddress(address);

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

    //open a TCP socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
    }
    //connect to the server
    if (connect(sockfd,
                (struct sockaddr *) &server_addr,
                sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
    }

    FILE* fdread = fdopen(sockfd,"r");

    for (int i = 0; i <1;i++){
        read_from_socket(fdread,read);
    }


    write_to_socket(sockfd, user);
    read_from_socket(fdread,read);

    if (strncmp (read, "530 Permission denied.",22) == 0){
        if(close_sockets(sockfd,sockfd2) == -1){
            exit(-1);
        }
        return 0;
    }

    write_to_socket(sockfd, password);
    read_from_socket(fdread,read);

    if (strncmp (read, "530 Login incorrect.",20) == 0){
        if(close_sockets(sockfd,sockfd2) == -1){
            exit(-1);
        }
        return 0;
    }

    write_to_socket(sockfd, passive);
    read_from_socket(fdread,read);
```

```
    int port2 = get_termB_port(fdread, read);

    bzero((char *) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(address);    //32 bit Internet address network
byte ordered
    server_addr.sin_port = htons(port2);        //server TCP port must be network byte
ordered

    //open a TCP socket
    if ((sockfd2 = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        exit(-1);
    }
    //connect to the server
    if (connect(sockfd2,
                (struct sockaddr *) &server_addr,
                sizeof(server_addr)) < 0) {
        perror("connect()");
        exit(-1);
    }

    FILE* fdread2 = fdopen(sockfd2,"r");

    write_to_socket(sockfd, path);
    read_from_socket(fdread,read);

    if (strncmp (read, "550 Failed to open file.",24) == 0){
        if(close_sockets(sockfd,sockfd2) == -1){
            exit(-1);
        }
        return 0;
    }

    printf("\ndata:\n");
    while (1){
        char* i = read_from_socket(fdread2,read);
        if (i == NULL) break;
    }
    printf("\n");

    if(close_sockets(sockfd,sockfd2) == -1){
        exit(-1);
    }
    return 0;
}
```

Anexo II – Configuration Commands

link up/activate eth0	# ifconfig eth0 up
list current net itfs' configurations	# ifconfig
configure eth0 with IP address 192.168.0.1 and netmask of 16 bits	# ifconfig eth0 192.168.0.1/16
add a route to the subnet	# route add -net 192.168.1.0/24 gw 172.16.4.254 eth0
add a default route	# route add default gw 192.168.1.1 eth0
list current routes	# route -n
list ARP entries	# arp -a
delete an ARP entry	# arp -d <ipaddress>
Creating an Ethernet VLAN	Switch# **configure terminal** Switch(config)# **vlan Y0** Switch(config)# **end** Switch# **show vlan id Y0**
Deleting VLAN	Switch# **configure terminal** Enter configuration commands, one per line. End with CNTL/Z. Switch(config)# **no vlan Y0** Switch(config)# **end** Switch# **show vlan brief**
Add port 1 to vlan Y0	Switch# **configure terminal** Enter configuration commands, one per line. End with CNTL/Z. Switch(config)# **interface fastethernet 0/1** Switch(config-if)# **switchport mode access** Switch(config-if)# **switchport access vlan Y0** Switch(config-if)# **end** Switch# **show running-config interface fastethernet 0/1** Switch# **show interfaces fastethernet 0/1 switchport**
Resetting the switch	Switch# **configure terminal** Enter configuration commands, one per line. End with CNTL/Z. Switch(config)# **no vlan 2-4094** Switch(config)# **exit** Switch(config)# **copy flash:tuxy-clean startup config** Switch(config)# **reload**
Entering config mode	router> **enable** router# **configure terminal** Enter configuration commands, one per line. End with CNTL/Z. *// do what is needed....* router (config)# **exit**

Resetting the router	router> **enable** router# **configure terminal** Enter configuration commands, one per line. End with CNTL/Z. router(config)# **copy flash:tuxy-clean startup-config** router(config)# **reload**

Anexo III – Logs Captured

Devido a não encontrarmos uma maneira de as colocar no documento Word estas encontram-se na pasta wireshark.