U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

– ***MIEIC Redes de Computadores*** –

Rede de Computadores

(2º Trabalho Laboratorial)

24 de Dezembro de 2015

Sumário:

O objectivo do presente trabalho em análise consiste na transferência de ficheiros de um servidor FTP e da configuração de uma rede local de computadores. A implementação da rede foi feita ao longo das aulas práticas através de 6 experiências, sendo que cada uma destas experiências foi seguida por uma análise e discussão dos resultados.

Grupo:
João Ramos - ei12062@fe.up.pt
Maria Miranda – ei12046@fe.up.pt
Rui Gonçalves - ei12185@fe.up.pt

Introdução

O presente relatório refere-se ao trabalho desenvolvido no âmbito da unidade curricular de Redes de Computadores (RCOM), do 3º ano do Mestrado Integrado em Engenharia Informática e Computação (MIEIC) da Faculdade de Engenharia da Universidade do Porto (FEUP). O trabalho em análise teve como objectivos:

- Criação de uma aplicação de download de ficheiros;
- Configuração e análise de uma rede local de computadores.

O nosso relatório está dividido em duas secções. A primeira secção inside na aplicação desenvolvida para efectuar o download de um ficheiro de um servidor FTP, através da implementação de um protocolo FTP, como especifica a norma RFC959, sendo que os dados introduzidos pelo utilizador estão também especificados na norma RFC1738. A segunda secção do nosso relatório aborda o estudo e análise da rede local de computadores, inclui portanto, a análise das 6 experiências realizadas nas aulas práticas. Esta parte do nosso projecto incluiu as seguintes etapas:

- Configuração de uma rede de IP;
- Implementação de virtual LANs;
- Configuração do router em Linux;
- Configuração de um router comercial e implementação de NAT;
- Configuração de DNS
- Conexões TCP.

Por último, referimos as principais conclusões retiradas deste projecto e em anexo mostramos o código da aplicação de download, no anexo 1, e os comandos de configuração de cada experiência.

Parte 1 – Aplicação de download

Estrutura do Código

A aplicação está dividida em várias funções:

- *int receive_answer(int sockfd, char * buf);*
  Esta função trata de receber uma resposta do servidor FTP e guardá-la em *buf.* Se a resposta contiver um código inicial de 500 ou acima, ou seja, código de erro, a aplicação termina.

- *int send_answer(int sockfd, char * msg);*
  Função que trata de enviar a resposta guardada em *msg* ao servidor.

- *int argvHandler(char* argv);*
  O objetivo desta função é de validar o input do utilizador ao começar o programa. Se o input estiver incorreto a aplicação termina com uma mensagem de erro. O input correto será: ftp://[<user>:<password>@]<host>/<url_path>.

- *int save_into_file(int fsockfd, char* fname);*
  Esta função trata de guardar todo o conteúdo das mensagens recebidas do servidor num ficheiro com o nome *fname*.

- *int main(int argc, char** argv);*
  Principal função do programa. O seu funcionamento e, consequentemente, o funcionamento da aplicação são explicadas a seguir.

Foram também criadas duas structs para guardar os dados a utilizar pela aplicação:

- *struct ftpStruct {*

  char user[64];
  char password[64];
  char host[256];
  char url_path[256];
  };
  Esta é a estrutura usada para guardar os dados que foram introduzidos pelo utilizador como argumento ao iniciar o programa.

- *struct ipStruct {*

  int ip1;
  int ip2;

```
    int ip3;
    int ip4;
    int port1;
    int port2;
    int fport;
};
```

Esta estrutura guarda os dados de resposta do servidor, separadamente, após o envio da mensagem "PASV". É também guardada, em fport, o resultado da fórmula para obter a porta final entre port1 e port2.

Funcionamento da aplicação

Ao iniciar a aplicação, esta chama a função *argvHandler(char* argv)*, que guarda a informação de *argv* na estrutura *ftpStruct ftps*. Se o link que tiver sido introduzido for válido em termos de sintaxe, é memorizado o nome do ficheiro a ser recebido, extraído através de *ftps.url_path* e da função *strtok*.

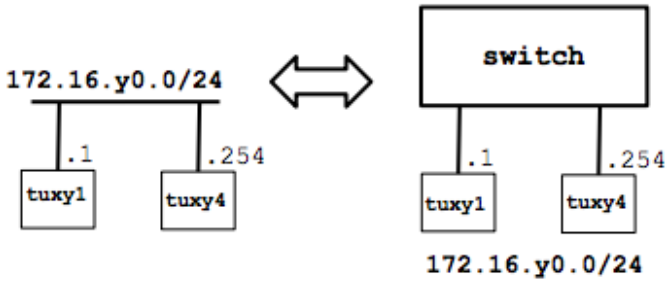
Com a função *getaddrinfo* vamos buscar a informação do servidor. Então, começa a tentativa de conexão com o servidor. É inicializado um socket e é feita uma tentativa de conectar com o servidor através do mesmo socket.

Se a conexão anterior for bem sucedida, começa então o envio das mensagens para o servidor e a receção das respostas do servidor, através das funções *send_answer e receive_answer*. Nesta parte da ligação são enviadas 3 mensagens para a ligação ao servidor: "USER *ftps.user"*; "PASS *ftps.password"*; "PASV".
A resposta do servidor à última mensagem irá dar-nos o ip e a porta final, guardada na estrutura *ipStruct ips*, e repete-se os passos de conetar com o servidor no novo ip e na nova porta.

Em seguida, é enviada a mensagem "RETR *ftps.url_path*" e, após chamar a função *receive_answer*, é finalmente chamada a função *save_into_file,* que vai criar e guardar a informação que o servidor enviar no ficheiro com o nome que foi memorizado no início da aplicação.

Parte 2 – Configuração da Rede

2.1 – Configuração de uma rede IP:



Na primeira experiência, o objetivo principal foi conectar dois computadores, no nosso caso tux21 e tux24, com endereços de forma a possibilitar a comunicação entre eles. Inicialmente desconectamos o switch da rede laboratorial e depois usando os comandos ifconfig e route configuramos os dois computadores. Após a configuração inicial usamos o comando ping para testar a conexão entre as duas máquinas. Este comando envia pacotes ICMP de pedido e resposta contendo os endereços da origem e do destino. Depois de visualizarmos as tabelas forwarding e ARP, apagamos os registos da ARP e, recorrendo novamente ao comando ping, analisamos o processo que desta vez foi guardado pelo programa Wireshark.

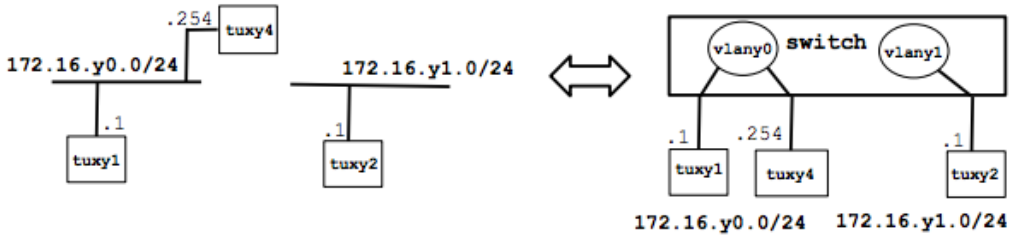
Os pacotes ARP (Address Resolution Protocol) fazem parte de um protocolo de dados usado para converter endereços IP em endereços físicos Ethernet que são reconhecidos na rede local, ou seja, servem para obter o endereço MAC de um computador a partir do seu IP. Este protocolo é usado quando um computador pretende enviar uma trama ethernet, sabendo já qual é o endereço MAC que corresponde ao IP pretendido. Desta forma, cada trama ARP contém o IP da origem e do destino, sendo ignorado o endereço MAC do destino.

Quando executamos o comando ping, no caso do IP ainda não estar na sua tabela ARP, é enviada uma trama ICMP echo request para o computador de destino. Este computador, que recebe a trama, reenvia a trama ICMP echo reply. Os endereços MAC e IP estarão então de acordo com a ARP table.

Para distinguir estes endereços, necessitamos de analisar os 2 bytes correspondentes ao header do pacote ethernet. Se tiver o valor 0x8000, corresponde ao ARP, no caso de ter o valor 0x0806 corresponde ao IP.

Por fim, uma interface loopback é uma interface virtual que não está associada a qualquer componente física num computador, sendo usada para fazer testes e diagnósticos à rede. Esta interface deve estar sempre ativa permitindo assim ter um endereço IP no router sempre ativo que não depende de qualquer interface física como referimos anteriormente.

2.2 – Implementação de virtual LANs:



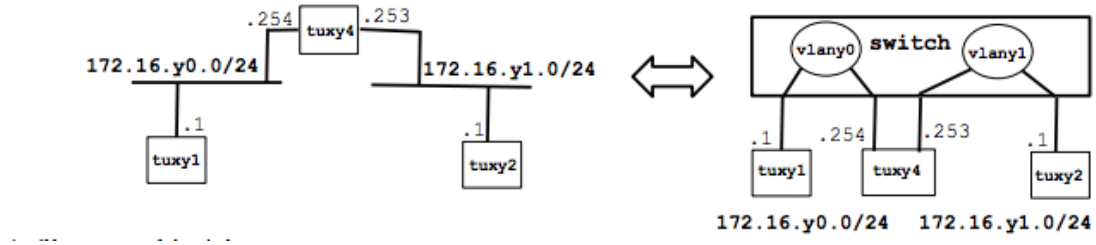
O objetivo desta experiência foi a criação de duas redes LANs, através de um switch. Numa das VLAN foram ligados dois computadores e na outra o restante.

Para a criação destas duas subredes, foi necessário aceder ao switch e recorrer à aplicação gkterm para esse efeito. Usaram-se os seguintes comandos para a criação e configuração de cada VLAN (só estão os comandos principais):

- Criação da VLAN:
  - vlan Y0
- Associação das portas à VLAN:
  - Interface fastethernet 0/X
  - Switchport access vlan Y0

Através dos logs obtidos podemos comprovar que nesta experiência existem dois domínios broadcast, um para a primeira VLAN e outro para a segundo, sendo que ambos se encontram em .255 em cada uma das redes.

2.3 – Configuração do router em Linux:



Para esta experiência, o objetivo é configurar o tuxY4 para funcionar como um router entre as duas redes criadas anteriormente.

De modo ao tuxY1 conseguir enviar pacotes para o tuxY2, este tem de conhecer um caminho pelo qual vai fazer essa ligação. Para criar essa ligação é configurada a interface eth1 no tuxY4 e permite-se *ip forwarding* no mesmo. Após isso, são criadas routes do tuxY1 para a rede VLANY1 através do tuxY4 e do tuxX2 para a rede VLANY0 através do tuxY1.

Após isto, foi verificada a conetividade através de um ping entre as máquinas 1 e 2, concluindo, com sucesso, a experiência.

2.4 – Configuração de um router comercial e implementação de NAT:

O objectivo desta experiência era configurar um router comercial e a implementação de NAT posteriormente, nesse mesmo router.

Primeiramente, adicionámos uma rota default no tuxY2 para o router comercial e seguidamente configurámos esse mesmo router, adicionando-o à subrede local em que estávamos a trabalhar nas experiências anteriores (172.16.Y1.0/24). Tendo isto feito, verificamos que o tuxY4 servia como router ao tuxY1 e que o router comercial servia como router ao tuxY2 e ao tuxY4.

Posteriormente, verificámos que a rede estava bem configurada, uma vez que, a partir do tuxY1, conseguimos fazer ping para o tuxY4, tuxY2 e o Rc.

De seguida foi retirada a rota do tuxY2 para a sub-rede 172.16.Y0.0/24, através do tuxY4. Verificámos, utilizando o comando traceroute que, com a remoção desta rota, os pacotes que iam do tuxY2 para o tux2Y1 seguiam o caminho tuxY2 – Rc – tuxY4 – tuxY1, como era de esperar.

Por fim, no tux21 fizemos ping para o router do laboratório (172.16.1.254), não sendo possível uma vez que ainda não tinha sido implementado o NAT no Rc.

Tendo isto, passámos para a implementação do NAT. Depois, ao fazer ping para o router do laboratório, a partir do tux21, já se obteve resposta. Isto deve-se ao facto de o NAT implementado possibilitar o acesso a redes externas à rede local implementada.

2.5 – Configuração de DNS:

Para esta experiência, o objetivo é fazer ping a um servidor da Internet usando o seu hostname, em vez do endereço de ip. Para isso, verificamos se existe duas linhas no ficheiro *resolv.conf* localizado na pasta *etc*:

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

Verificada a existência do ficheiro contendo o conteúdo acima referido, verificamos que conseguimos alcançar o objetivo principal da experiência.

Ao executar o comando ping para um hostname, vai ser enviado um pacote DNS para o servidor configurado no ficheiro acima. O servidor responde com um pacote DNS, que contém a informação pretendida. Após saber o IP do servidor, é realizado o ping.

2.6 – Conexões TCP:

Realizando a primeira parte da experiência com a nossa aplicação e completando o download do ficheiro pretendido com sucesso, verificamos que a nossa aplicação abre duas conexões TCP, a primeira para enviar pacotes de controlo e a segunda para receber o ficheiro.

Ao realizar a segunda parte da experiência, verificamos que a velocidade do tuxY1 diminuiu e que as velocidades de download dos dois tuxs se estabilizam.

As conexões TCP estão divididas em três fases: estabelecimento de ligação, transferência de dados e encerramento da ligação:

- Na primeira fase, é usado um *three-way-handshake*:
    - O cliente envia um SYN para o servidor e altera o número da sequência da trama para um valor aleatório A.
    - Em resposta, o servidor envia um SYN-ACK, em que o número de ACK é mais um do que o que foi recebido, ou seja A+1, e o número de sequência toma outro valor aleatório B.
    - Finalmente, o cliente responde com um ACK, em que o número de sequência tem como valor A+1 e ACK tem como valor B+1.

- Na fase de transferência de dados, existem mecanismos que garantem que os dados foram transmitidos com sucesso. Esses mecanismos garantem, por exemplo, que as sequências de dados estejam ordenados, que existe retransmissão de pacotes, etc.

- Por último, a fase de encerramento de ligação executa um *four-way-handshake:*
    - O cliente envia uma trama FIN para o servidor.
    - O servidor responde com uma trama ACK.
    - Quando o servidor estiver ponto a terminar envia uma trama FIN para o cliente.
    - Depois de receber a trama, o cliente envia uma trama ACK e é terminada a ligação.

Conclusões

Concluido o trabalho, pensamos que os objetivos propostos para o mesmo foram alcançados. Após a realização das experiências e estudo das questões colocadas durante as mesmas, sentimos que houve uma boa aprendizagem em relação ao funcionamento de redes de computadores. No entanto houve alguns desafios que tiveram de ser ultrapassados.

Em primeiro, a estrutura da aplicação de download. Apesar do código da aplicação em si ser relativamente simples e de termos feito a experiência na primeira aula prática deste trabalho, a parte de conexão ao servidor através de sockets e, principalmente, de arranjar uma solução de mostrar aquilo que estávamos a fazer no ecrã, ou seja, *debug* ao nosso código, mostrou-se algo que requeriu alguma pesquisa.

Mas, principalmente, o grupo teve problemas com a bancada que estávamos a, como foi visto na apresentação deste trabalho. O facto deste problema estar fora do nosso controlo, atrasou um pouco o trabalho, e houveram experiências que podiam ser concluidas em menos tempo.

Anexo 1

```
//dowload.h
#ifndef DOWNLOAD_H
#define DOWNLOAD_H

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

#define SERVER_PORT "21"
#define SIZE 1024

struct ftpStruct {

	char user[64];
	char password[64];
	char host[256];
	char url_path[256];
};

struct ipStruct {

	int ip1;
	int ip2;
	int ip3;
	int ip4;
	int port1;
	int port2;
	int fport;
};


struct ftpStruct ftps;
struct ipStruct ips;

int receive_answer(int sockfd, char * buf);
int send_answer(int sockfd, char * msg);
```

```
int argvHandler(char* argv);
int main(int argc, char** argv);
int save_into_file(int fsockfd, char* fname);

#endif

//download.c

#include "download.h"

//function to create and save contents received in a new file
int save_into_file(int fsockfd, char* fname){

        char buf[SIZE];
        FILE *my_file;
        int received_b = 0;
        unsigned long totalReceived = 0;

        my_file = fopen (fname, "w");

  if(my_file == NULL) {
                printf("Failed to open and write in file %s\n", fname);
                exit(1);
  }
  else
        printf("\nCreated or erased file %s!\nStarted retrieving file, please
wait...\n", fname);

        while((received_b = recv(fsockfd, buf,SIZE,0)) != 0){
                totalReceived += received_b;
                fwrite(buf,sizeof(char),received_b, my_file);
        }

        printf("\n\nSuccess! Received %lu bytes and saved into file %s!\n\n",
totalReceived, fname);
        fclose(my_file);

        return 0;
}

//function to receive answers from the socket
int receive_answer(int sockfd, char * buf){

        int code;
        memset(buf, 0, SIZE);
        sleep(1);
        strcpy(buf, "");
        recv(sockfd,buf,SIZE, 0);
        printf("\n\nReceived message:\n\n%s\n\n",buf);
```

```
        if(sscanf(buf, "%d %*s", &code) == 1){

                if(code >= 500)
                {
                        printf("Failed with code %d!\n", code);
                        exit(1);
                }
        }
        return 0;
}

//function to send an answer through the socket
int send_answer(int sockfd, char* msg){

        printf("\n\nSending message:\n\n%s\n\n",msg);
        strcat(msg,"\n");
        send(sockfd,msg, strlen(msg),0);

        return 0;
}

int argvHandler(char* argv){

        //ftp://[<user>:<password>@]<host>/<url->
   if (sscanf(argv, "ftp://%[^:]:%[^@]@%[^/]/%s\n", ftps.user, ftps.password,
ftps.host, ftps.url_path) == 4){

        printf("Copied to user: %s\n",ftps.user);
        printf("Copied to password: %s\n",ftps.password);
        printf("Copied to host: %s\n",ftps.host);
        printf("Copied to url_path: %s\n",ftps.url_path);
        printf("Argv successfully decripted!\n\n");

        return 0;

   }
   if (sscanf(argv, "ftp://%[^/]/%s\n", ftps.host, ftps.url_path) == 2){

        strcpy(ftps.user, "anonymous");
        strcpy(ftps.password, "123");
        printf("Copied to user: %s\n",ftps.user);
        printf("Copied to password: %s\n",ftps.password);
        printf("Copied to host: %s\n",ftps.host);
        printf("Copied to url_path: %s\n",ftps.url_path);
        printf("Argv successfully decripted!\n\n");

        return 0;
```

```
    }

    return -1;

}

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


            char buf[SIZE];
            char msg[512];
            char IP[128];
            char port[8];
            char IP_str[INET_ADDRSTRLEN];

            char fname[128];
            char *split;

            int sockfd, fsockfd;
            struct addrinfo host_info, *server_info, *aux, *new_server;
            int getaddr;

    if (argc != 2) {
        fprintf(stderr,"usage: ./app
ftp://[<user>:<password>@]<host>/<url_path>\n");
        exit(1);
    }

    if(argvHandler(argv[1]) != 0){
            fprintf(stderr,"Argv format is wrong!\n Usage:
ftp://[<user>:<password>@]<host>/<url_path>\n");
            exit(1);
    }

   split = strtok(ftps.url_path, "/");

   while( split != NULL )
   {
     strcpy(fname,split);
     split = strtok(NULL, "/");
   }

   printf("\n\nDestination filename is: %s\n\n", fname);
            memset(&host_info, 0, sizeof(host_info));
            host_info.ai_family = AF_INET;
            host_info.ai_socktype = SOCK_STREAM;
            host_info.ai_protocol = 0;
```

```
        if ((getaddr = getaddrinfo(ftps.host, SERVER_PORT, &host_info,
&server_info)) != 0) {
            fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(getaddr));
            exit(1);
        }

        for(aux = server_info; aux != NULL; aux = aux->ai_next) {
            if ((sockfd = socket(aux->ai_family, aux->ai_socktype, aux-
>ai_protocol)) == -1) {
                        perror("Failed to create sockfd!");
                        continue;
            }

            if (connect(sockfd, aux->ai_addr, aux->ai_addrlen) == -1) {
                        close(sockfd);
                        perror("Failed to connect to host!");
                        continue;
            }


                printf("Host name is:%s\n", ftps.host);

                struct in_addr  *addr;
            if (aux->ai_family == AF_INET) {
               struct sockaddr_in *ipv = (struct sockaddr_in *)aux->ai_addr;
               addr = &(ipv->sin_addr);
            }
            else {
               struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)aux->ai_addr;
               addr = (struct in_addr *) &(ipv6->sin6_addr);
            }
      inet_ntop(aux->ai_family, addr, IP_str, sizeof IP_str);

      printf("Server address is:          %s\n", IP_str);
                /*
            if(aux->ai_family == AF_INET){
                inet_ntop(aux->ai_family, &(((struct sockaddr_in *)addr)-
>sin_addr),
              IP_str, sizeof(IP_str));

            }
            else{
                inet_ntop(aux->ai_family, &(((struct sockaddr_in6 *)addr)-
>sin6_addr),
              IP6_str, sizeof(IP6_str));
                printf("Server address is:      %s\n", IP6_str);
            }*/

            break;
```

```
        }

        if (aux == NULL) {
           fprintf(stderr, "Failed to connect!\n");
           exit(2);
        }



   //receive answer from server
   receive_answer(sockfd, buf);
   //send user
   memset(msg,0,512);
   strcpy(msg, "USER ");
   strcat(msg, ftps.user);
   send_answer(sockfd, msg);

   //receive
   receive_answer(sockfd, buf);
   //send pass
   memset(msg, 0,512);
   strcpy(msg, "PASS ");
   strcat(msg, ftps.password);
   send_answer(sockfd, msg);

   //receive
        receive_answer(sockfd, buf);
        //send pasv
        memset(msg, 0, 512);
   strcpy(msg, "PASV");
   send_answer(sockfd, msg);
   receive_answer(sockfd, buf);

        //receive answer as 193.231.321.31.2312.3 // check page from guião 1ªa
aula
        // res is the buffer where the answer is saved
        if(sscanf(buf, "%*[^(](%d,%d,%d,%d,%d,%d)\n", &ips.ip1, &ips.ip2,
&ips.ip3, &ips.ip4, &ips.port1, &ips.port2) != 6)
        {
                printf("Failed to read ip from server response! %s\n", buf);
                exit(1);
        }

        memset(IP, 0, 128);
        sprintf(IP,"%d.%d.%d.%d", ips.ip1, ips.ip2, ips.ip3, ips.ip4);
        printf("\nCopied ip: %s\n", IP);
        memset(port, 0, 8);
        ips.fport = 256 * ips.port1 + ips.port2;
        sprintf(port,"%d",ips.fport);
```

```
        printf("\nCopied port: %s\n", port);

        if ((getaddr = getaddrinfo(IP, port, &host_info, &new_server)) != 0) {
           fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(getaddr));
           exit(1);
        }

        for(aux = new_server; aux != NULL; aux = aux->ai_next) {
           if ((fsockfd = socket(aux->ai_family, aux->ai_socktype, aux-
>ai_protocol)) == -1) {
                        perror("Failed to create sockfd!");
                        continue;
           }

           if (connect(fsockfd, aux->ai_addr, aux->ai_addrlen) == -1) {
                        close(fsockfd);
                        perror("Failed to connect to host!");
                        continue;
           }

           break;
        }

        if (aux == NULL) {
           fprintf(stderr, "Failed to connect!\n");
           exit(2);
        }

        memset(msg, 0, 512);
   strcpy(msg, "RETR ");
   strcat(msg, ftps.url_path);
   send_answer(sockfd, msg);

   receive_answer(sockfd,buf);

   save_into_file(fsockfd,fname);

        close(fsockfd);
        close(sockfd);
        freeaddrinfo(server_info);
        freeaddrinfo(new_server);
        exit(0);
}
```

Anexo 2

EXP1:

---- CONFIGURAÇÕES-----

-No tuxY1:
> /etc/init.d/networking restart
> ifconfig eth0 up //Iniciar eth0
> ifconfig
> ifconfig eth0 172.16.Y0.1/24
> route add default gw 172.16.Y0.254
> route -n //ver se a route foi criada corretamente

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

IP - 172.16.20.1 //tux 21
MAC - 00:0f:fe:8c:af:9d

-No tux24:

> /etc/init.d/networking restart
> ifconfig eth0 up //Iniciar eth0
> ifconfig
> ifconfig eth0 172.16.Y0.254/24
> route add default gw 172.16.Y0.254
> route -n //ver se a route foi criada corretamente

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

IP - 172.16.20.254 //tux 24
MAC - 00:22:64:a6:a4:f1

------ PING -------

> ping 172.16.Y0.1 // tuxY4
> ping 172.16.Y0.254 // tuxY1

> arp -a

> arp -d 172.16.Y0.254 //tuxY1

EXP2:

- No tuxY2:
> /etc/init.d/networking restart
> ifconfig eth0 172.16.Y1.1/24 up
> ifconfig //verificar que eth0 está lá com o IP certo
> echo 1 > /proc/sys/net/ipv4/ip_forward
> echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

IP: 172.16.21.1 //tux 22
MAC: 00:21:5a:61:2b:72

Restart switch:
> enable
> 8nortel
> conf t
> no vlan 2-4094
> exit
> copy flash:tux2-clean startup-config [Enter]
> reload [Enter]

> enable
> 8nortel
> conf t //
> vlan Y0 // criar vlan20 se nao existir ainda
> exit //

> interface fastethernet 0/1 //
> switchport mode access // adicionar port 1
> switchport access vlan Y0 //
> exit //

> interface fastethernet 0/4 //
> switchport mode access // adicionar port 4
> switchport access vlan Y0 //
> end //

> show vlan brief // verificar que existe VLAN 20 e tem port Fa0/1 e Fa0/4

> configure terminal //
> vlan Y1 // criar vlan21 se nao existir ainda
> exit //

> interface fastethernet 0/2 //
> switchport mode access // adicionar port 2
> switchport access vlan Y1 //
> end //

> show vlan brief // verificar que existe VLAN Y1 e tem port Fa0/2

EXP 3:

No tuxY4:

> ifconfig eth1 172.16.Y1.253/24

---eth1--- //tux 24
IP:172.16.21.253
MAC: 00:08:54:50:3f:2c

---eth0---//tux 24
IP - 172.16.20.254
MAC - 00:22:64:a6:a4:f1

No tuxY2:

> route add -net 172.16.Y0.0/24 gw 172.16.Y1.253 /adicionar rota para tux Y1

No tuxY1:

> route add -net 172.16.Y1.0/24 gw 172.16.Y0.254 //adicionar route para tuxY2

No switch:

> conf t
> interface fastethernet 0/5
> switchport mode access
> switchport access vlan Y1
> end

----PING----

No tuxY1:

>ping 172.16.Y0.254
>ping 172.16.Y1.253
>ping 172.16.Y1.1

EXP 4:

No tuxY2:

> route add default gw 172.16.Y1.254

No router:

> configure terminal
> interface gigabitethernet 0/0
> ip address 172.16.Y1.254 255.255.255.0
> no shutdown
> exit

> interface gigabitethernet 0/1
> ip address 172.16.1.Y9 255.255.255.0
> no shutdown
> exit

>ip route 172.16.Y0.0 255.255.255.0 172.16.Y1.253

No tuxY4:

>route del default gw 172.16.Y0.254

No switch:

>conf t
>interface fastethernet 0/3
>switchport mode access
>switchport access vlan Y1
>end

No tuxY2:
> echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects //rejeitar redirects
> echo 0 > /proc/sys/net/ipv4/conf/eth0/accept_redirects //rejeitar redirects

>route del -net 172.16.Y0.0/24 gw 172.16.Y1.253
>traceroute 172.16.Y0.1

1 -> 172.16.Y1.254 // router
2 -> 172.16.Y1.253 // tuxY4
3 -> 172.16.Y0.1 // tuxY1

>route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
>traceroute 172.16.Y0.1

1 -> 172.16.Y1.253 // tuxY4

2 -> 172.16.Y0.1 // tuxY1

>route del -net 172.16.Y0.0/24 gw 172.16.Y1.253
> echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects
> echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects

//apos o primeiro redirect ele guarda o endereço de ip em cache e usa-o para os próximos pings

No tuxY1:
>ping 172.16.1.254 // Não funciona porque não há NAT no router

No router:

>copy flash:tuxY-clean startup-config
>reload

> configure terminal

> 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 //from 172.16.1.Y9 to 172.16.1.Y9
> 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.Y1.254
>ip route 172.16.Y0.0 255.255.255.0 172.16.Y1.253

No tuxY1:

> ping 172.16.1.254

1 -> 172.16.Y0.254
2 -> 172.16.Y1.254
3 -> firetux.netlab.fe.up.pt (172.16.1.254)