Segundo Trabalho Laboratorial de Redes de Computadores

Rui Moreira **up201906355**

José Silva **up201904775**

28 de janeiro de 2022

1 Aplicação de Download

1.1 Arquitetura

A aplicação de Download desenvolvida na primeira parte do segundo trabalho de Redes de Computadores, foi dividida nas seguintes parte: **parser, connection, download**. O código encontra-se devidamente comentado no Anexo A.

1.1.1 Parser

Neste módulo é feito o tratamento dos dados recebidos da bash quando a aplicação é executada. A partir de uma função parseConsoleInput() são guardadas as informação necessárias, extraídas do URL. Os seguintes campos são extraidos e guardados:

- Username
- Password
- Hostname
- File Path

Conjuntamente com esta função são usadas as funções **parseFileName()**, **getHostHelper()**, para respetivamente, a partir do URL, saber qual o nome do host e o path do file que o utilizador quer extrair do server FTP.

1.1.2 Connection

Como função inicial deste modulo temos a função **connectSocket()**. Que está encarregue de, ao receber um endereço de IP e uma port, que são respetivamente fornecidos pelas funções **getTip()** e **serverPort()**, criar uma socket e iniciar a ligação com essa socket.

A função **sendRequest()** recebe o file descriptor criado pela função acima mencionada com a socket aberta, e uma string com o comando a enviar para o servidor.

A função **getReply()** recorre a uma state machine que para receber a resposta enviada pelo servidor ao nosso request, processando os diferentes tipos de respostas possiveis.

Para o tratamento do envio e receção destas mensagens usamos uma função generica, **handleCommunication()**, que aceita um comando, um argumento caso o comando necessite, e consegue guardar a resposta para ser interpretada por outras funções. Esta função analiza as respostas do servidor ftp e trata cada uma das possibilidades de resposta

De seguida temos a função **userLogin()**, que permite o envio dos dados de login do utilizador para o servidor ftp.

A função **downloadFile()** está encarregue do download do ficheiro que irá transferir para a maquina e a função **saveFile** está encarregue de preencher o ficheiro na máquina com os dados que recebe o servidor. O dados são adicionados ao ficheiro sequencialmente, até o buffer de leitura estar vazio.

1.1.3 Download

Este módulo apenas contém a função main() que faz a chamada das funções necessárias para executar o download de um ficheiro do servidor ftp.

O programa começa por verificar o input da consola, caso este seja diferente do esperado, uma mensagem de erro é apresentada e terminamos a sua execução. Se o programa receber um URL, este é tratado pela função **parseConsoleInput()** que preenche os argumentos passados com os componentes do URL.

São impresso na consola, detalhes sobre a ligação que está a ser estabelecida e seguidamente é chamada a função **main()**, que cria um socket para proceder à comunicalção com o servidor ftp. Consequentemente o programa chama a função **getReply()** para ler a mensagem com os detalhes da ligação estabelecida.

Seguidamente inicia-se a tentativa de login, na qual é enviado o comando 'user' seguido do username, lê a resposta e executa o mesmo procedimento com o comando 'pass' para envio da password. Caso não sejam especificadas credenciais de login o username utilizado é anonymous e a password anonymous.

O proximo passo é enviar o comando pasv para pedir ao servidor que transfira através da porta atribuida dados em modo passivo, que faz com que seja necessário abrir outra ligação. A resposta dada pelo servidor é analisada, e consequentemente, caso esteja correta é aberta uma nova ligação, com o endereço de ip e porta fornecidos por essa mesma resposta.

Concluindo, o comando 'retr' é enviado levando o path do ficheiro como argumento, e o programa chama as funções **downloadFile()** e **SaveFile()**, que respetivamente fazem o download, e guardam o ficheiro.

2 Parte 2: Experiências em Laboratório

2.1 Experiência nº1

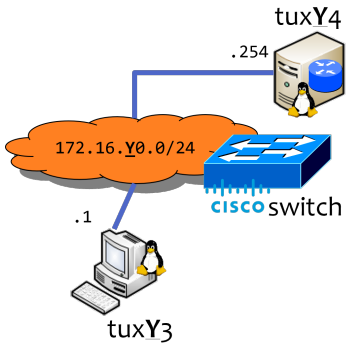

Fig 1 - Configuração de IP

O fundamento desta experiência é entender o que são pacotes ARP e qual a sua finalidade, que tipo de pacotes é que o comando ping gera e o que são, respetivamente endereços MAC e endereços IP.

Os comandos que foram usados durante esa experiência estão presentes no anexo B.1.

2.1.1 Análise dos Logs

O que são pacotes ARP e qual a sua utilização? Os pacotes ARP são usados para pedir o endereço MAC, sabendo o endereço IP de uma máquina. Os endereços MAC indentificam a placa de rede, enquanto que os endereços IP servem como identificadores públicos para permitir que uma máquina possa comunicar com outras máquinas através de uma rede. Uma máquina pode apenas possuir um endereço MAC, mas vários endereços de IP.

Quais são os pacotes gerados pelo comando ping Os pacotes ICMP são gerados através do comando ping, e estes são normalmente usados por hosts ou routers para mandar *Third Layer Errors* ou mensagens de controlo para outros hosts ou routers. Para o propósito desta experiência este comando apenas serve para testar a conectividade entre computadores.

Quais são os endereços MAC e IP dos ping packets? Os pacotes gerados pelo comando ping contêm o MAC do target e da source e contém também o IP do target e da source, com algumas flags incluidas.

Como distinguir se o trama recebido é ARP, IP ou ICMP? Através da análise das capturas do wireshark é possivel verificar o formato dos pacotes ARP que são enviados quando um ping é feito. O Wireshark atribui cores diferentes a tipos de pacotes diferentes, mas é possivel ver qual o tipo de pacote através dos detalhes dos mesmos.

Como determinar o tamanho de um trama recebido? Para analisar o tamanho das tramas recebidas, é verificado o valor "on wire"que representa o valor de bytes passados, ou então, verificar o campo *Total Length* nos tramas IPV4.

O que é *loopback interface* e o porquê da sua importância? O loopback interface é um interface virtual que está sempre ativa, e é sempre atingível desde que uma das interfaces de IP definidas no switch esteja operacional. Como resultado, e tendo em conta que o endereço do loopback nunca muda, é a melhor forma de indentificar um dispositivo numa rede. É particularmente útil para tarefas de debugging tendo em conta o facto de que os endereços de IP podem sempre ser pingados desde que qualquer outro switch interface esteja ativo tambem.

2.2 Experiência nº2

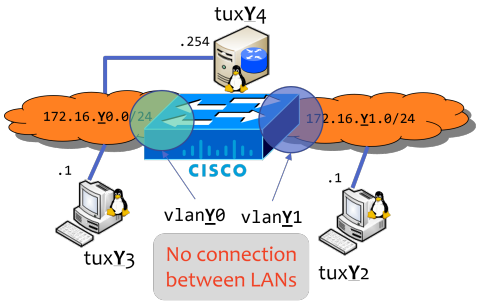

Fig 2 - Configuração de LAN virtual

O fundamento desta experiência é a criação de duas VLAN's no switch, e assim perceber a conectividade entre os computadores, depois de configurar cada um deles em cada uma das sub-redes.

Os comandos que foram usados durante esa experiência estão presentes no anexo B.2.

2.2.1 Análise dos Logs

Como configurar uma VLAN? Para configurar as VLAN's criámos a VLAN 40 e 41 e associamos à primeira, os tux's 43 e 44 e à segunda o tux 42. Tendo em vista com a obtenção da arquitetura desejada na figura. Para testar a conectividade entre os tux's, foi realizado um ping do tux43 até o tux44 que tal como seria de esperar teve sucesso, uma vez que se encontram na mesma sub-rede.

Quanto à conexão entre o tux43 e o tux42, o ping obteve a resposta *Network unreachable* devido ao simples facto de não haver nenhuma rota defenida entre as VLAN's, tornado impossível a comunicação entres os tux43 e tux42.

Quantos broadcasts existem, e como é que podemos concluir através dos logs? Também no tux43 foi efetuado um ping em broadcast, ping -b 172.16.40.255, que não obteve no tux 43. Seria expectável uma resposta do tux44 uma vez que se estão na mesma subrede, mas isto não acontece devido ao mecanismo *echo-ignore-broadcast* estar ativado por pré-definição por forma a evitar grandes amplificações de tráfego. No entanto, os logs realizados no tux44 provam que este recebeu um pedido do tux43 por broadcast.

Repetimos o processo anteriormente descrito mas agora a partir do tux42. Como seria de esperar nenhuma resposta foi obtida, mas por motivos diferentes aos apresentados anteriormente. Neste caso não foi obtida nenhuma resposta pois apenas o tux42 se encontra configurado na VLAN41.

Concluindo que exitem dois domínios diferentes de broadcast correspondentes às VLAN 40 e VLAN 41 com os endereços *172.16.40.255* e *172.16.41.255*, respetivamente.

2.3 Experiência nº3

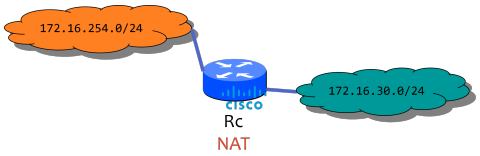

Fig 3 - Configuração de Router Cisco em Casa

2.3.1 Análise dos Logs

Como configurar uma rota estática num router comercial? Executando o seguinte comando: **ip route 172.16.40.0 255.255.255.0 172.16.30.2**. O primeiro endereço está associado à origem dos pacotes, o segundo endereço é a máscara de sub-rede, e o último endereço é o gateway dos pacotes.

Como configurar NAT num router comercial?

- A configuração da NAT segue os seguintes comandos:
  - Identificar a interface de rede: interface FastEthernet 0/0.
  - Associar à NAT o endereço de IP que da interface: ip address 172.16.30.1 255.255.255.0
  - Especificar o tipo de NAT: ip nat inside
- Postumamente deve ser configurada a pool de endereços exteriores disponiveis com: ip nat pool ovrld 172.16.254.45 172.16.254.45 prefix-length 24
- Por fim, basta configurar a pool IP's internos:
  - ip nat inside source list 1 pool ovrld overload
  - access-list 1 permit 172.16.40.0 0.0.0.7
  - access-list 1 permit 172.16.30.0 0.0.0.7

O que faz a NAT? NAT é uma técnica que permite mapear uma gama de endereços IP para outra gama modificando o IP dos packets enviados enquanto estão em trânsito. É na grande maioria das vezes utilizada para mapear o endereço IP público fornecido por um provedor de internet (ISP) para o endereço privado da máquina do utilizador.

Como configurar o serviço de DNS no host? Podem ser configuradas traduções especificas em /etc/hosts como fizemos para o mapeamento do endereço de youtubas. Um servidor central pode ser configurado de forma a realizar a tradução no ficheiro /etc/resolv.conf adicionando: nameserver IP_DNS.

Que pacotes são trocados pelo DNS e que informação é transportada? Podem ser reconhecidos dois tipos de pacotes. Ambos contém o hostname sobre contendo o IP que procuramos. A resposta aos pacotes do tipo 'A' é um endereço IPv4, já a resposta ao pacotes AAAA é um endereço IPv6. Os pacotes do tipo 'A' e os pacotes do tipo 'AAAA' têm como resposta, um endereço **IPv4** e **IPv6**, respetivamente.

Que pacotes ICMP são observados e porquê? O traceroute tenta encontrar o *shortest path* para alcançar um determinado destino. Para isso cria, com *tempo de vida (TTL)* crescente partindo de 1, pacotes UDP. Quando um pacote ICPM informa que TTL foi excedido o traceroute sabe que necessita de pelo menos mais um valor de TTL para alcançar o destino. Para esta experiência como o destino não aceita pedidos é ainda enviado um pacote ICPM que informa que a porta target é *unreachable*.

Quais são os endereços IP e MAC associados a pacotes ICMP e porquê? - Os endereço de origem estão relacionados com os vários IP's intermédios por onde os pacotes passam até alcançarem o target, e os endereços de destino correspondem às nossas máquinas. Sempre que o tempo de vida é aumentado pelo traceroute, o endereço de origem muda, sabendo assim que um nó foi atingido pelo pacote. O endereço MAC de origem permanece inalterado, e está associado à interface virtual do host, e o endereço MAC de destino corresponde à interface virtual do Guest OS.

Quais são as rotas na sua máquina? Qual o seu significado? A rota com origem em 0.0.0.0 e destino em 172.24.64.1 é a default gateway é que explicita por onde devem ser enviados os pacotes caso não possam ser encaminhados para a rede local. A rota com origem no endereço 172.24.64.0 e gateway 0.0.0.0 é uma rota inválida, destinada para os pacotes com destino inválido.

2.4 Experiência nª4

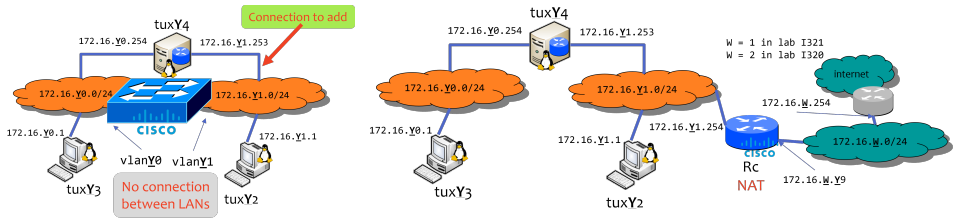

Fig 4 - Linux Router Configuration
Fig 5 - Cisco Router Configuration

O fundamento desta experiência é primeiramente transformar o tuxy4 num router para possibilitar a comunicação entre o tuxy3 e o tuxy2, através das VLAN's 0 e 1. Para haver comunicação terá de se configurar os endereços IP's das portas ethernet dos tuxy's e as routes que serão usadas. E seguidamente tem como objetivo estabelecer uma ligação com a rede dos laboratórios e implementar rotas num router comercial, adicionar-lhe funcionalidade NAT, e perceber qual a sua função.

Os comandos usados para esta experiência podem ser encontrados no Anexo B.3.

2.4.1 Análise dos Logs

Que rotas estão nos tuxes, e qual o seu significado? Para adicionar uma rota nova a um dos tuxes são necessários, o IP da rede que desejamos aceder, a *bit mask* desse IP e o IP da gateway a usar. De forma a existir uma ligação entre o tux43 e o tux42, foi adicionada uma rota ao tux43 de forma a aceder aos endereços 172.16.41.0/24 a partir do IP 172.16.40.254 (tux44 eth0) e uma rota ao tux42 para aceder aos endereços 172.16.40.0/24 a partir do IP 172.16.41.253 (tux44 eth1).

Que informações podem ser encontradas numa entrada da *forwarding table*? A *forwading table* contém as informações necessárias para enviar um pacote, no melhor trajeto possivel, até ao seu destino. Cada pacote contém informação sobre a sua origem e destino. A *forwading table* dá ao dispositivo instruções para o envio do pacote para o proximo nó na rota da rede.

Cada entrada contém:

- **ID Rede** - O ID da rede ou o destino corresponde à rota.
- **Máscara da Sub-rede** - A máscara é usada para para dar *match* ao IP de destino com o IP da rede.
- **Próximo nó** - O endereço de IP para onde o pacote é enviado.
- **Próxima Interface** - A próxima interface para onde pacote deve ir para chegar à rede de destino.
- **Metric** - Uma medida usada para saber aproximadamene o número de routers que o packet atravessa até o seu destino.

Quais são as mensagens ARP observadas, e os seus respetivos endereços MAC, e o porquê? ://www.overleaf.com/project/61f1698b4827eaeda98068e4par Com estas rotas definidas, é possivel realizar um ping, a partir do tux43, para todas as interfaces dos outros tux's. Também se verifica que a interface eth0 do tux44 enviou 2 pedidos ARP para conseguir determinar o endereço MAC da interface eth0 to tux43, enquanto que o tux23 mandou um pedido para saber o endereço MAC da interface eth0 do tux44.

Continuando, é possível verificar que existe comunicação entre os tux's 43 e 42, visto que os pings realizados pelo tux 43 obtêm uma resposta do tux 42 e vice-versa. Nota-se uma troca de mensagens ARP para o tux43 tomar conhecimento do endereço MAC da interface eth0 to tux44 e vice-versa, enquanto que no segundo caso existe uma troca de mensagens ARP para o tux42 tomar conhecimento do endereço MAC da interface eth1 do tux44 e vice-versa.

Quais são os caminhos seguidos pelos pacotes na experiência realizada e porquê? O mecanismo implementado, NAT, que tem como principio substituir os endereços de IP Locais nos pacotes enviados por um endereço de IP público possibilitando assim o estabelecimento de uma ligação para fora da rede. Um router que implemento o mecanismo NAT, é responsável pelo encaminhamento de todos os pacotes que lhe são chegados, para o endereço correto, que pode ou não encontrar-se dentro da rede local.

Começámos por configurar a interface GE 0/0 do router, atribuída à VLAN 41. Para a interface GE 0/1 do router, atribuiu-se o IP 172.16.1.49 para que fosse feita a ligação com a rede estabelicida dos laboratórios.

Através da analise da imagem disponiblizada, defenimos que o tux44 serviria de router para o tux43 e o router RC para o tux42 e tux44. Além disso, foram adicionadas as devidas rotas estáticas no router RC.

Após estas configurações foi possível realizar o ping do tux43 para todos os outros pontos da nossa rede. A única diferença de conexão do tux 43 para as experiências anteriores é que, agora também é possível aceder às interfaces GE 0/0 e GE 0/1 do router RC, sendo isto possível devido à adição de duas rotas no router RC.

- default gateway com o IP 172.16.1.29
- reencaminhamento de pacotes para a rede com IP 172.16.40.0/24 (VLAN 40 onde se encontra o tux43) através da interface eth1 do tux44 com IP 172.16.41.253.

A conexão do tux42 à interface eth0 do tux43 é efetuada através da rota implementada no tux42 que foi realizada na experiência anterior.

- Para verificar a nova implementação foi removida a rota do tux42,e foi executado o traceroute comprovando, que como não havia nenhuma rota definida até à VLAN 40, o router com o IP 172.16.41.254, definido como default gateway do tux42 ficou responsável por redirecionar os pacotes ICMP até ao destino.
- Voltando a adicionar a rota anteriormente definida e fazendo um novo traceroute verificamos que os pacotes deixam de passar pelo router e passam a seguir o *shortest path* via tux44.

Sendo assim, tentámos desde o tux43 realizar ping do router do laboratório com o IP 172.16.1.254, que não teve qualquer respostas visto que o mecanismo NAT ainda não tinha sido definido no router RC.

Após a adição do mesmo, voltamos a realizar o ping, tendo sucesso visto que, o NAT permite que os dispositivos conectados à rede local, 172.16.44.0/24 (interface 0/0), comuniquem com a rede externa, 172.16.1.29 (interface 0/1).

3 Anexo A Código Download

Todo o código está documentado com doxygen, explicando sucintamente cada função

3.1 Connection.h

```c
#ifndef CONNECTION_H
#define CONNECTION_H

#include "parser.h"

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

#define FTP_SERVER_PORT 21
#define MULTI_LINE_SYMBOL '-'
#define LAST_LINE_SYMBOL ' '
#define CARRIAGE_RETURN 13

#define PSV_RESPONSE_MAXSIZE 1000
#define FILE_BUFFER_SIZE 256

typedef enum serverResponse
{
  STATUS_CODE,
  IS_MULTI_LINE,
  READ_MULTI_LINE,
  READ_LINE
} serverResponse;

/**
 * The struct hostent (host entry) with its terms documented

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

    #define h_addr h_addr_list[0] The first address in h_addr_list.
*/

/**
 * @brief Gettip function provided by code examples in class, using the function gethostname to fill the hostent struct with the correct information
 *
 * @param hostname              - Hostname
 * @return struct hostent*      Returns the hostent struct with the data correctly filled upon success, exits with code 1 upon failure.
 */
struct hostent *getip(char hostname[]);

/**
 * @brief Connect the ftp server socket, open the ftp server file descriptor
 *
 * @param addr      - The ip address of the client
 * @param port      - The port oppened by the ftp server where data will be sent from
 * @return int      The socket file descriptor upon SUCCESS, ERROR otherwise.
 */
int connectSocket(char *addr, int port);

/**
 * @brief Get the Response from the ftp server, after a request has been sent
```

```
 *
 * @param socketFd  - File descriptor of the ftp server
 * @param code      - Response status code
 * @param text      - Arguments given by the ftp server
 * @return int      The socket file descriptor upon SUCCESS, ERROR otherwise.
 */
int getReply(int socketFd, char *code, char *text);

/**
 * @brief Send commands to the ftp server, commands can be sent with or without arguments, depending on its nature
 *
 * @param sockfd    - File descriptor of the ftp server
 * @param cmd       - Command to be sent to the ftp server
 * @param argument  - Argument that goes after the command, can be null if the command takes no arguments
 * @return int      Returns SUCCESS upon Success, ERROR otherwise.
 */
int sendRequest(int sockfd, char *cmd, char *argument);

/**
 * @brief Function to deal with userLogin fraction of the ftp server connection, handling the server responses for the username and password, respectively
 *
 * @param sockfd    - File descriptor of the ftp server
 * @param user      - Username parsed from the command line
 * @param pass      - Password parsed from the command line
 * @return int      Returns SUCCESS upon Success, ERROR otherwise.
 */
int userLogin(int sockfd, char *user, char *pass);

/**
 * @brief Get the Port object from the ftp server, if the server acknowledges the request without any errors, parse the port value
 *
 * @param sockfd    - File descriptor of the ftp server
 * @param port      - FTP port to send the files
 * @return int      Returns SUCCESS upon Success, ERROR otherwise.
 */
int serverPort(int sockfd, int *port);

/**
 * @brief Download the file from the ftp server, sending rtrv request to ackownledge the download request
 *
 * @param sockfd    - File descriptor of the ftp server
 * @param downloadFd - File descriptor of the file to be downloaded
 * @param path      - File download internal ftp path
 * @return int      Returns SUCCESS upon Success, ERROR otherwise.
 */
int downloadFile(int sockfd, int downloadFd, char *path);

/**
 * @brief Save the file being sent by the ftp server through the port
 *
 * @param downloadFd - File descriptor of the file to be downloaded
 * @param filename   - Filename parsed by the console input
 * @param fileSize   - Filesize received by the server response acknowledgment after the SIZE request
 * @return int       Returns SUCCESS upon Success, ERROR otherwise.
 */
int saveFile(int downloadFd, char *filename, size_t fileSize);

/**
 * @brief Handle the client server communication, send the request interpret the reply
 *
 * @param sockfd    - File descriptor of the ftp server
 * @param cmd       - Command to be sent to the ftp server
 * @param argument  - Argument to be sent to the ftp server can be null if the cmd takes no arguments
 * @param text      - Server reply
```

```
 * @return int      Returns SUCCESS upon Success, ERROR otherwise.
 */
int handleCommunication(int sockfd, char *cmd, char *argument, char *text);

/**
 * @brief Show the download progress bar
 *
 * @param percentage  - Download percentage
 */
void percentagePrint(size_t percentage);
#endif
```

3.2 Connection.c

```

#include "connection.h"

struct hostent *getip(char hostname[])
{
    printf("Getting IP Address from Host Name...\n");
    struct hostent *h;

    if ((h = gethostbyname(hostname)) == NULL)
    {
        herror("Failed to get host by name");
        exit(1);
    }

    return h;
}

int connectSocket(char *addr, int port)
{
    printf("Connecting to Server Socket...\n");

    int sockfd;
    struct sockaddr_in server_addr;

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

    /* open a TCP socket */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        fprintf(stderr,"socket()");
        return ERROR;
    }

    /* connect to the server */
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0)
    {
        fprintf(stderr,"connect()");
        return ERROR;
    }

    fprintf(stdout,"Server Socket Connected\n");

    return sockfd;
}

int getReply(int socketFd, char *code, char *text)
{
    char c;
    int i = 0;
    serverResponse state = STATUS_CODE;
```

```
55
56      /*State Machine to get the status code and the rest of the arguments provided by
        the server side*/
57      while (1){
58          switch (state){
59              case STATUS_CODE:
60                  if (read(socketFd, code, 3) < 0){
61                      fprintf(stderr, "Error reading status code\n");
62                      return ERROR;
63                  }
64
65                  fprintf(stdout,"Status Code -> %s", code);
66                  state = IS_MULTI_LINE;
67                  break;
68
69              case IS_MULTI_LINE:
70                  if (read(socketFd, &c, 1) < 0){
71                      fprintf(stderr, "Error reading response\n");
72                      return ERROR;
73                  }
74
75                  fprintf(stdout,"%c", c);
76
77                  if (c == MULTI_LINE_SYMBOL) state = READ_MULTI_LINE;
78                  else state = READ_LINE;
79                  break;
80
81              case READ_MULTI_LINE:
82                  int idxCounter = 0;
83                  char str[4];
84                  str[3] = '\0';
85
86                  while (1){
87                      if (read(socketFd, &c, 1) < 0){
88                          fprintf(stderr, "Error reading response\n");
89                          return -1;
90                      }
91
92                      fprintf(stdout,"%c", c);
93                      if (c == '\n') break;
94
95                      if (idxCounter <= 2)str[idxCounter] = c;
96
97                      if (idxCounter == 3 && strcmp(str, code) == 0 && c ==
    LAST_LINE_SYMBOL){
98                          state = READ_LINE;
99                          break;
100                     }
101
102                     /*if the symbol given by the server is different from carriadge or
     the text is null insert the char read to the server response*/
103                     if (idxCounter > 3 && text != NULL && c != CARRIAGE_RETURN) text[i
    ++] = c;
104
105                     idxCounter++;
106             }
107
108             break;
109
110         case READ_LINE:
111             while (1){
112                 if (read(socketFd, &c, 1) < 0){
113                     fprintf(stderr, "Error reading response\n");
114                     return ERROR;
115                 }
116
117                 fprintf(stdout,"%c", c);
118                 if (c == '\n') break;
119
120                 /*if the symbol given by the server is different from carriadge or
     the text is null insert the char read to the server response*/
```

```
                    if (text != NULL && c != CARRIAGE_RETURN) text[i++] = c;
                }

                if (text != NULL) text[i++] = '\0';
                return 0;

            default:
                break;
        }
    }
}

int sendRequest(int sockfd, char *cmd, char *argument){
    size_t cmd_len = strlen(cmd);

    if (write(sockfd, cmd, cmd_len) != cmd_len){
        fprintf(stderr, "Error while sending command\n");
        return ERROR;
    }

    /* Before sending the argument to the server, use a ' ' to help distinguish the
    blocks status ' ' argument*/
    if (argument != NULL){
        size_t arg_len = strlen(argument);
        char c = ' ';

        if (write(sockfd, &c, 1) != 1){
            fprintf(stderr, "Error while sending command\n");
            return ERROR;
        }

        if (write(sockfd, argument, arg_len) != arg_len){
            fprintf(stderr, "Error while sending command argument\n");
            return ERROR;
        }
    }

    /*Write a new line to the end of the command to end the request*/
    char c = '\n';
    if (write(sockfd, &c, 1) != 1){
        fprintf(stderr, "Error while sending command\n");
        return ERROR;
    }

    return SUCCESS;
}

int userLogin(int sockfd, char *user, char *pass){
    fprintf(stdout,"Sending username to the ftp server %s...\n",user);
    int res = handleCommunication(sockfd, "user", user, NULL); /* Sending the username
     to the ftp server, receiving a response */
    if (res < 0){
        fprintf(stderr, "Error while sending username to the ftp server!\n");
        return ERROR;
    }

    if (res == 1){
        fprintf(stderr,"Sending password to the ftp server...\n");

        if (handleCommunication(sockfd, "pass", pass, NULL) < 0){
            fprintf(stderr, "Error while sending password to the ftp server!\n");
            return ERROR;
        }
    }

    return SUCCESS;
}

int serverPort(int sockfd, int *port){
    char responseCode[4];
    memset(responseCode, 0, 4);
```

```
190
191     char response[PSV_RESPONSE_MAXSIZE];
192
193     fprintf(stdout,"Sending pasv...\n");
194
195     if (handleCommunication(sockfd, "pasv", NULL, response) < 0){
196         fprintf(stderr, "Error while sending pasv\n");
197         return ERROR;
198     }
199
200     if (parsePort(response, port) < 0){
201         fprintf(stderr, "Error parsing port\n");
202         return ERROR;
203     }
204
205     return SUCCESS;
206 }
207
208
209 int downloadFile(int sockfd, int downloadFd, char *path)
210 {
211     char responseCode[4];
212     memset(responseCode, 0, 4);
213     char fileName[MAX_PATH_SIZE];
214
215     char cmdResponse[MAX_RESPONSE_SIZE];
216
217     int res = handleCommunication(sockfd, "retr", path, cmdResponse);
218
219     size_t fileSize = parseFileSize(cmdResponse);
220
221     if (res < 0){
222         fprintf(stderr, "Error while sending retr\n");
223         return ERROR;
224     }
225     else if (res != 2){
226         fprintf(stderr, "Server refused to transfer file\n");
227         return ERROR;
228     }
229
230     parseFileName(path, fileName);
231     if (saveFile(downloadFd, fileName, fileSize) < 0) return ERROR;
232
233     if (getReply(sockfd, responseCode, NULL) < 0){
234         fprintf(stderr, "Failed to confirm file transfer\n");
235         return ERROR;
236     }
237
238     return SUCCESS;
239 }
240
241 int saveFile(int downloadFd, char *fileName, size_t fileSize)
242 {
243     FILE *file = fopen(fileName, "wb");
244
245     uint8_t buf[FILE_BUFFER_SIZE];
246     int bytes;
247     size_t readBytes = 0;
248     size_t percentage = -1;
249     while ((bytes = read(downloadFd, buf, FILE_BUFFER_SIZE)) > 0)
250     {
251         if (bytes < 0)
252         {
253             fprintf(stderr, "Error while reading file\n");
254             return ERROR;
255         }
256         fwrite(buf, bytes, 1, file);
257
258         if (fileSize > 0)
259         {
260             readBytes += bytes;
```

```
261             size_t newPercentage = (readBytes * 100) / fileSize;
262
263             fflush(stdout);
264             if (newPercentage != percentage)
265             {
266                 printf("\33[2K\r");
267                 percentagePrint(newPercentage);
268             }
269             percentage = newPercentage;
270         }
271     }
272     fprintf(stdout,"\n");
273
274     fclose(file); /*Close the file descriptor after the file being succesfully
    downloaded*/
275     return SUCCESS;
276 }
277
278 int handleCommunication(int sockfd, char *cmd, char *argument, char *text)
279 {
280
281     char responseCode[4];
282     memset(responseCode, 0, 4);
283     /*Send a request to the ftp server*/
284     if (sendRequest(sockfd, cmd, argument)){
285         fprintf(stderr, "Error while sending username\n");
286         return ERROR;
287     }
288
289     /*Acknowledge the reply from the server*/
290     int code;
291     if (getReply(sockfd, responseCode, text) < 0) return ERROR;
292     /*Get the status code from the reply*/
293     code = responseCode[0] - '0';
294
295     switch (code){
296         /* Expecting another reply  */
297         case 1:
298             return 2;
299
300         /* positive completion reply */
301         case 2:
302             break;
303
304         /* waiting for more information */
305         case 3:
306             return 1;
307
308         /* resend the command */
309         case 4:
310             if (handleCommunication(sockfd, cmd, argument, text) < 0)
311                 return ERROR;
312             break;
313
314         /* permanent negative completion reply */
315         case 5:
316             fprintf(stderr, "Command wasn't accepted\n");
317             return ERROR;
318     }
319
320     return SUCCESS;
321 }
322
323 void percentagePrint(size_t percentage){
324     if(percentage < 5) fprintf(stdout, "Progress: [###>                        ] (%lu%%)",
     percentage);
325     else if(percentage < 10) fprintf(stdout, "Progress: [#######>                    ]
    (%lu%%)", percentage);
326     else if(percentage < 15) fprintf(stdout, "Progress: [#########>                   ]
    (%lu%%)", percentage);
327     else if(percentage < 20) fprintf(stdout, "Progress: [##########>                  ]
```

```
    (%lu%%)", percentage);
328     else if(percentage < 25) fprintf(stdout, "Progress: [##########>                ]
    (%lu%%)", percentage);
329     else if(percentage < 30) fprintf(stdout, "Progress: [###########>                ]
    (%lu%%)", percentage);
330     else if(percentage < 35) fprintf(stdout, "Progress: [############>               ]
    (%lu%%)", percentage);
331     else if(percentage < 40) fprintf(stdout, "Progress: [#############>              ]
    (%lu%%)", percentage);
332     else if(percentage < 45) fprintf(stdout, "Progress: [###############>           ]
    (%lu%%)", percentage);
333     else if(percentage < 50) fprintf(stdout, "Progress: [################>          ]
    (%lu%%)", percentage);
334     else if(percentage < 55) fprintf(stdout, "Progress: [##################>       ]
    (%lu%%)", percentage);
335     else if(percentage < 60) fprintf(stdout, "Progress: [###################>      ]
    (%lu%%)", percentage);
336     else if(percentage < 65) fprintf(stdout, "Progress: [####################>     ]
    (%lu%%)", percentage);
337     else if(percentage < 70) fprintf(stdout, "Progress: [######################>   ]
    (%lu%%)", percentage);
338     else if(percentage < 75) fprintf(stdout, "Progress: [#######################>  ]
    (%lu%%)", percentage);
339     else if(percentage < 80) fprintf(stdout, "Progress: [########################>  ]
    (%lu%%)", percentage);
340     else if(percentage < 85) fprintf(stdout, "Progress: [##########################> ]
    (%lu%%)", percentage);
341     else if(percentage < 90) fprintf(stdout, "Progress: [###########################>]
    (%lu%%)", percentage);
342     else if(percentage < 95) fprintf(stdout, "Progress: [############################>]
     (%lu%%)", percentage);
343     else fprintf(stdout, "Progress: [##############################] (%lu%%)",
    percentage);
344 }
```

3.3 parser.h

```
1  #ifndef PARSE_H
2  #define PARSE_H
3
4  #include <string.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  #define PROTOCOL_SIZE 6
9  #define MAX_USER_SIZE 256
10 #define MAX_PWD_SIZE 256
11 #define MAX_HOST_SIZE 512
12 #define MAX_PATH_SIZE 1024
13 #define MAX_RESPONSE_SIZE 4096
14
15 #define SUCCESS 0
16 #define ERROR -1
17
18 typedef enum parserState
19 {
20   PROTOCOL,
21   USER,
22   PASS,
23   HOST,
24   PATH
25 } parserState;
26
27 /**
28  * @brief Function to parse input given from the console
29  *
30  * @param consoleInput - Input given in the console for download application command
31  * @param username     - Username provided to establish the FTP connection
32  * @param password     - Password provided to establish the FTP connection associated
     to the Username
```

```
 * @param hostname      - FTB hostname
 * @param path          - Path inside the FTP server specifing the file we want to
    download
 * @return int          Return SUCCESS upon success, ERROR otherwise
 */
int parseConsoleInput(char *consoleInput, char *username, char *password, char *
    hostname, char *path);

/**
 * @brief Get the port number to access ftp server (par1 *256) + par2
 *
 * @param response      - Passive mode data being received with the port information
    last two fields, with port information
 * @param port          - Passive mode port after being converted to real port number
 * @return int          Return SUCCESS upon success, ERROR otherwise
 */
int parsePort(char *response, int *port);

/**
 * @brief Get where index of the console command where the hostname and path starts,
    right after @ identifier
 *
 * @param input         - Console Input data
 * @return int          Return -1 in case no user and no password is provided,
    otherwise the index where the hostname starts
 */
int getHostHelper(char *input);

/**
 * @brief Auxiliary function to reverse a string, and calculate the console port
    number
 *
 * @param str           - String to be reversed to calculate port number
 * @return char*        Returns the port number in a string correctly reversed
 */
char *reverseString(char *str);

/**
 * @brief Given the file path, retrieve the filename wanted from the ftp server
 *
 * @param path          - Console trimmed string containing the path to ftb server file
 * @param fileName      - Filename wanted from the ftp server correctly trimmed
 */
void parseFileName(char *path, char *fileName);

/**
 * @brief After getting a 213 success response from ftp server get the filesize of the
     file we want to download
 *
 * @param text
 * @return size_t = 0 if error
 */
size_t parseFileSize(char *text);

#endif
```

3.4 parser.c

```

#include "parser.h"

int parseConsoleInput(char *consoleInput, char *username, char *password, char *
    hostname, char *path)
{
    int i = 0, j = 0;
    char protocol[PROTOCOL_SIZE + 1];
    memset(protocol, 0, PROTOCOL_SIZE + 1);

    parserState state = PROTOCOL;

```

```
12      int hostIdx = getHostHelper(consoleInput);
13      if (hostIdx == -1){ // Anonymous user
14          strcpy(username, "anonymous");
15          strcpy(password, "anonymous");
16      }
17
18      while (1)
19      {
20          char c = consoleInput[i++];
21
22          if (c == '\0' && state != PATH){
23              fprintf(stderr, "Wrong format:\nftp://[<user>:<password>@]<host>/<url-path
        >\n");
24              return ERROR;
25          }
26
27          switch (state){
28              case PROTOCOL:
29                  protocol[j++] = c;
30
31                  if (j == PROTOCOL_SIZE){
32                      if (strcmp(protocol, "ftp://") != 0){
33                          fprintf(stderr, "Please use ftp protocol://\n");
34                          return ERROR;
35                      }
36
37                      if (hostIdx > -1) state = USER;
38                      else state = HOST;
39                      j = 0;
40                  }
41
42                  break;
43              case USER:
44                  if (c == ':'){
45                      state = PASS;
46                      j = 0;
47                      break;
48                  }
49
50                  username[j++] = c;
51
52                  if (j > MAX_USER_SIZE){
53                      fprintf(stderr, "Username is too large! Max: %d\n", MAX_USER_SIZE)
        ;
54                      return ERROR;
55                  }
56
57                  break;
58              case PASS:
59                  if (i == hostIdx + 1){
60                      state = HOST;
61                      j = 0;
62                      break;
63                  }
64
65                  password[j++] = c;
66
67                  if (j > MAX_PWD_SIZE){
68                      fprintf(stderr, "Password is too large! Max: %d\n", MAX_PWD_SIZE);
69                      return ERROR;
70                  }
71
72                  break;
73              case HOST:
74                  if (c == '/'){
75                      state = PATH;
76                      j = 0;
77                      break;
78                  }
79
80                  hostname[j++] = c;
```

```
81
82                  if (j > MAX_HOST_SIZE){
83                      fprintf(stderr, "Hostname is too large! Max: %d\n", MAX_HOST_SIZE)
    ;
84                      return ERROR;
85                  }
86
87                  break;
88              case PATH:
89                  if (c == '\0')
90                      return 0;
91
92                  path[j++] = c;
93
94                  if (j > MAX_PATH_SIZE){
95                      fprintf(stderr, "File path is too large! Max: %d\n", MAX_PATH_SIZE
    );
96                      return ERROR;
97                  }
98                  break;
99              default:
100                 fprintf(stderr,"Args badly parsed from the console!");
101                 break;
102         }
103     }
104
105     return SUCCESS;
106 }
107
108
109 char *reverseString(char *str)
110 {
111     if (!str || !*str)
112         return str;
113
114     int i = strlen(str) - 1, j = 0;
115
116     while (i > j)
117     {
118         char c = str[i];
119         str[i--] = str[j];
120         str[j++] = c;
121     }
122
123     return str;
124 }
125
126 int parsePort(char *response, int *port)
127 {
128     size_t i = strlen(response) - 1, currIdx = 0;
129     char first[4], second[4];
130     memset(first, 0, 4);
131     memset(second, 0, 4);
132
133     if (response[i--] != ')')
134     {
135         if (response[i + 1] != '.' || response[i--] != ')')
136         {
137             fprintf(stderr, "Wrong response format: %s\n", response);
138             return ERROR;
139         }
140     }
141
142     while (response[i] != ',')
143         second[currIdx++] = response[i--];
144     i--;
145     currIdx = 0;
146     while (response[i] != ',')
147         first[currIdx++] = response[i--];
148
149     int p1 = atoi(reverseString(first));
```

```
150     int p2 = atoi(reverseString(second));
151 
152     *port = (p1 * 256) + p2;
153     return SUCCESS;
154 }
155 
156 int getHostHelper(char *input)
157 {
158     size_t len = strlen(input);
159 
160     int hostIdx = -1;
161 
162     for (int i = 0; i < len; i++)
163     {
164         if (input[i] == '@')
165         {
166             hostIdx = i;
167         }
168     }
169 
170     return hostIdx;
171 }
172 
173 void parseFileName(char *path, char *fileName)
174 {
175     size_t pathSize = strlen(path);
176 
177     int idx = 0;
178     for (int i = pathSize - 1; i >= 0; i--)
179     {
180         char c = path[i];
181         if (c == '/')
182             break;
183         fileName[idx] = c;
184         idx++;
185     }
186 
187     fileName[idx] = '\0';
188     reverseString(fileName);
189 }
190 
191 size\_t parseFileSize(char *text)
192 {
193     size_t textLen = strlen(text);
194 
195     size_t fileSize = 0;
196     int i = textLen - 9;
197     int counter = 1;
198     while (text[i] != '(')
199     {
200         int digit = text[i] - '0';
201 
202         if (digit < 0 || digit > 9)
203             return 0;
204 
205         fileSize += digit * counter;
206         counter *= 10;
207         i--;
208     }
209 
210     return fileSize;
211 }
```

3.5 download.c

```
1 
2 #include "connection.h"
3 
4 int main(int argc, char **argv){
5     if (argc != 2){
```

```
6           fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n",
    argv[0]);
7           exit(ERROR);
8       }
9
10      char responseCode[4];
11      memset(responseCode, 0, 4);
12
13      char user[MAX_USER_SIZE];
14      memset(user, 0, MAX_USER_SIZE);
15      char pass[MAX_PWD_SIZE];
16      memset(pass, 0, MAX_PWD_SIZE);
17      char host[MAX_HOST_SIZE];
18      memset(host, 0, MAX_HOST_SIZE);
19      char path[MAX_PATH_SIZE];
20      memset(path, 0, MAX_PATH_SIZE);
21
22      if (parseConsoleInput(argv[1], user, pass, host, path)) exit(ERROR);
23
24      printf("User: %s\nPass: %s\nHost: %s\nPath: %s\n", user, pass, host, path);
25
26      struct hostent *h = getip(host);
27      char *address = inet_ntoa(*((struct in_addr *)h->h_addr));
28
29      printf("Host name  : %s\n", h->h_name);
30      printf("IP Address : %s\n", address);
31
32      int sockfd = connectSocket(address, FTP_SERVER_PORT);
33      if (sockfd < 0){
34          fprintf(stderr, "Error while connecting to socket\n");
35          return ERROR;
36      }
37
38      printf("Getting connection response:\n");
39      getReply(sockfd, responseCode, NULL);
40
41      if (userLogin(sockfd, user, pass) < 0) exit(ERROR);
42
43      printf("Getting port from server...\n");
44      int port;
45      if (serverPort(sockfd, &port) < 0){
46          fprintf(stderr, "Error while getting port from server\n");
47          return ERROR;
48      }
49
50      printf("NEW PORT: %d\n", port);
51
52      int downloadFd = connectSocket(address, port);
53      if (downloadFd < 0){
54          fprintf(stderr, "Error while connecting to socket\n");
55          return ERROR;
56      }
57
58      printf("Downloading file...\n");
59
60      if (downloadFile(sockfd, downloadFd, path) < 0){
61          fprintf(stderr, "Error while downloading file\n");
62          return ERROR;
63      }
64
65      printf("Closing connection...\n");
66      if (close(sockfd) < 0){
67          fprintf(stderr, "Failed to close socket\n");
68          exit(ERROR);
69      }
70
71      if (close(downloadFd) < 0){
72          fprintf(stderr, "Failed to close socket\n");
73          exit(ERROR);
74      }
75
```

```
76      printf("Connection closed\n");
77
78      return SUCCESS;
79  }
```

4 Anexo B

4.1 Anexo B.1 - Descrição Procedimentos da 1ª Experiência

Steps

1. Disconnect the switch from netlab and connect tux computers to it.
2. Configure tuxy3 and tuxy4 to have an IP address and have the network 172.16.Y0.0/24 defined
   - using ip addr and ip route commands
   - you can also use respectively ifconfig and route
3. Take note of the IP and MAC addresses of the network interfaces on both tuxes.
4. Use ping command to verify connectivity between these computers.
   - Note that there is no name resolution for the IP addresses defined, so you need to use the IP addresses.
   - You can have a look a /etc/hosts.
5. Inspect the forwarding table (ip route show or route –n) and ARP (ip neigh or arp –a) tables
6. Delete ARP table entries in tuxy3 (ip neigh del ¡ipaddress¿ or arp –d ¡ipaddress¿)
7. Start Wireshark in eth0 of tuxy3 and start capturing packets
8. In tuxy3, ping tuxy4 for a few seconds
9. Stop capturing packets
10. Save log and study it at home to answer the questions below.

4.1.1 Steps 2 e 3

Listing 1: **Ligar os cabos ao switch e configurar os endereços de IP do tux3 e tux4**

```
#Tux43
> ifconfig eth0 up
> ifconfig eth0 172.16.20.1/24
> ifconfig eth0
#Tux44
> ifconfig eth0 up
> ifconfig eth0 172.16.20.254/24
> ifconfig eth0
```

172.16.40.1	00:21:5a:61:2c:54	tux43 eth0
172.16.40.254	00:22:64:19:09:5c	tux44 eth0

4.1.2 Step 4

Listing 2: **Pingar o tux43 do tux44 e vice-versa**

```
#Tux43
> ping 172.16.40.254  # recebe pacotes de 64 bytes desse endere o
#Tux44
> ping 172.16.40.1 # recebe pacotes de 64 bytes desse endere o
```

4.1.3 Step 5

Listing 3: **Inspecionar a arp table**

```
#Tux43
> route -n
#Criar a rota para 172.16.40.0 atrav s da eth0 e verificar a arp table
> arp -a
#Associa  o do endere o IP a um endere o MAC
> (172.16.40.254) at 00:22:64:19:09:5c [ether] on eth0
```

4.1.4 Step6

Listing 4: **Apagar a entrada na arp table**

```
> arp -d 172.16.40.254
> arp -a  #N o da output a nada pois n o tem mais a entry na arp table
```

4.2 Anexo B.2 - Descrição Procedimentos da 2ª Experiência

Steps

1. Configure tuxy2's network and register its IP and MAC addresses
2. Create vlanY0 in the switch and add corresponding ports
3. Create vlanY1 and add corresponding port
4. Start a Wireshark capture at eth0 of tuxy3.
5. In tuxy3, ping tuxy4 and then ping tuxy2. Recall that unless you setup /etc/hosts you need to use the IP addresses.
6. Stop the capture and save a log.
7. Start new captures in eth0 of tuxy3, eth0 of tuxy4 and eth0 of tuxy2.
8. In tuxy3, do ping broadcast (ping -b 172.16.y0.255) for a few seconds.
9. Observe the results, stop the captures and save the logs.
10. Repeat steps 7, 8 and 9, but now do ping broadcast in tux2 (ping -b 172.16.y1.255).

4.2.1 Cable Setup

TUX43Eth0	Tux42Eth0	TUX44Eth0
Switch Port 1	Switch Port 3	Switch Port 2

4.2.2 Step 1

Listing 5: **Configurar os IPs dos tuxes**

```
#Tux42 Config
> ifconfig eth0 up
> ifconfig eth0 172.16.41.1/24
> ifconfig eth0
#Tux43 Config
> ifconfig eth0 up
> ifconfig eth0 172.16.40.1/24
> ifconfig eth0
#Tux44 Config
```

```
> ifconfig eth0 up
> ifconfig eth0 172.16.40.254/24
> ifconfig eth0
```

172.16.41.1	00:1f:29:d7:45:c4	Tux42 eth0
172.16.40.1	00:21:5a:61:2f:d4	Tux43 eth0
172.16.40.254	00:21:5a:5a:7b:ea	Tux44 eth0

4.2.3 Step 2

Agora o próximo passo é configuar o switch através do tux43.

Listing 6: **Criar VLAN (vlan40)**

```
>configure terminal
>vlan 40
>end
>show vlan id 40
```

Listing 7: **Adicionar as portas 1 e 2 à VLAN criada**

```
#Porta 1
>configure terminal
>interface fastethernet 0/1
>switchport mode access
>switchport access vlan 40
>end
>show running-config interface fastethernet 0/1
>show interfaces fastethernet 0/1 switchport
#Porta 2
>configure terminal
>interface fastethernet 0/2
>switchport mode access
>switchport access vlan 40
>end
```

4.2.4 Step 3

Listing 8: **Criar a vlan 41**

```
>configure terminal
>vlan 41
>end
>show vlan id 41
```

Listing 9: **Adicionar a vlan 41 a porta 3**

```
>configure terminal
>interface fastethernet 0/3
>switchport mode access
>switchport access vlan 41
>end
```

4.2.5 Step 5

Listing 10: **Pingar os tuxes a partir do tux43**

```
# Pingar o tux42 a partir do tux43
$ ping 172.16.41.1
#Pingar o tux44 a partir do tux 43
$ ping 172.16.40.254
```

4.2.6 Step 8 e 10

Listing 11: **Pingar o broadcast dos tuxes a partir do tux43**

```
# Pingar o tux42 a partir do tux43
$ ping -b 172.16.40.255
#Pingar o tux44 a partir do tux 43
$ ping -b 172.16.41.255
```

4.3 Anexo B.3 - Descrição Procedimentos da 4ª Experiência

4.3.1 Steps

- As per the figure above connect a cable from tuxy4/eth1 to the switch and place it in vlanY1 (see Exp2 if in doubt).
- Verify the VLANs on the switch (show vlan brief).
- Configure tuxy4/eth1's IP address as per the figure (172.16.Y1.253/24).
- On tuxy4 (see helpers below) -¿ Enable IP forwarding; Disable ICMP echo ignore broadcast.
- Check the MAC addresses and IP addresses in tuxy4 for eth0 and eth1.
- Configure the routes in tuxy3 and tuxy2 so that they can reach each other. In tuxy3: ip route add 172.16.Y1.0/24 via 172.16.Y0.254 or route add -net 172.16.Y1.0/24 gw 172.16.Y0.254 In tuxy2: ip route add 172.16.Y0.0/24 via 172.16.Y1.253 or route add -net 172.16.Y0.0/24 gw 172.16.Y1.253
- Observe the routes available at the 3 tuxes.
- Start a capture at tuxy3.
- From tuxy3, ping the other network interfaces (172.16.Y0.254, 172.16.Y1.253, 172.16.Y1.1) and verify if there is connectivity.
- Stop the capture and save logs.
- Start a capture in tuxy4 on both interfaces (in Wireshark select with Ctrl+Click the connections to listen to).
- Clean the ARP tables in the 3 tuxes.
- In tuxy3, ping tuxy2 for a few seconds.
- Stop the capture in tuxy4 and save logs.

Step 1

TUX42	E0	SwitchPort 2
TUX43	E0	SwitchPort 1
TUX44	E0	SwitchPort 3
TUX44	E1	SwitchPort 4

Step 2

TUX43S0 –> T3
T4 –> Switch Console

Setup dos cabos
VLAN 0:

- tux43 eth0 –> port 1
- tux44 eth0 –> port 3

VLAN 1:

- tux42 eth0 –> port 2
- tux44 eth1 –> port 4

Configuração de todos os tuxes a partir do GKterm usando o switch
Login

```
>enable
>password: 8nortel
```

Criar VLAN 40:

```
>configure terminal
>vlan 40
>end
>show vlan id 40
```

Adicionar porta 1 a vlan 40:

```
>configure terminal
>interface fastethernet 0/1
>switchport mode access
>switchport access vlan 40
>end
```

Adicionar porta 3 a vlan 40:

```
>configure terminal
>interface fastethernet 0/3
>switchport mode access
>switchport access vlan 40
>end
```

Criar VLAN 41:

```
>configure terminal
>vlan 41
>end
>show vlan id 41
```

Adicionar porta 2 a vlan 41:

```
>configure terminal
>interface fastethernet 0/2
>switchport mode access
>switchport access vlan 41
>end
```

Adicionar porta 4 a vlan 41:

```
>configure terminal
>interface fastethernet 0/4
>switchport mode access
>switchport access vlan 41
>end
```

Verificação final para ver se as portas foram corretamente adicionadas às VLAN's

```
>show vlan brief
```

Step 3
Tux42:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.41.1/24
> ifconfig eth0
```

Tux43:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.40.1/24
> ifconfig eth0
```

Tux44:

```
> ifconfig eth0 up
> ifconfig eth0 172.16.40.254/24
> ifconfig eth0

> ifconfig eth1 up
> ifconfig eth1 172.16.41.253/24
> ifconfig eth1
```

eth0	00:1f:29:d7:45:c4	Tux42
eth0	00:21:5a:61:2f:d4	Tux43
eth0	00:21:5a:5a:7b:ea	Tux44
eth1	00:c0:df:25:la:f4	Tux44

Step 4:
Trocar para o tux44 e e inserir os seguintes comandos no terminal:

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

Step 5

eth0	00:1f:29:d7:45:c4	Tux42
eth0	00:21:5a:61:2f:d4	Tux43
eth0	00:21:5a:5a:7b:ea	Tux44
eth1	00:c0:df:25:la:f4	Tux44

Step 6
Usar os comandos descritos no enunciado.
Step 7
Fazer 'route -n' em cada 1 dos 3 tuxs para observar as routes.
tux22:

Destination	Gateway	Genmask	Interface
172.16.40.0	172.16.41.253	255.255.255.0	eth0
172.16.41.0	0.0.0.0	255.255.255.0	eth0

Tabela 1: Tabela das rotas do tux42.

tux23:
tux24:
Step 9

Destination	Gateway	Genmask	Interface
172.16.40.0	0.0.0.0	255.255.255.0	eth0
172.16.41.0	172.16.41.254	255.255.255.0	eth0

Tabela 2: Tabela das rotas do tux43.

Destination	Gateway	Genmask	Interface
172.16.40.0	0.0.0.0	255.255.255.0	eth0
172.16.41.0	0.0.0.0	255.255.255.0	eth1

Tabela 3: Tabela das rotas do tux44.

1. pingar a interface eth0 do tux44 - ping 172.16.40.254
2. pingar a interface eth1 do tux44 - ping 172.16.41.253
3. pingar a interface eth0 do tux42 - ping 172.16.41.1

Step 12

```
> arp -a (verificar quais os IPs que se podem apagar)
> arp -d endereco (limpar cada entrada da arp table com o endereco)
> arp -a (tem de retornar nada)
```

Repetir o processo em todos os tuxes

Step 13

A partir do tux43, pingar o tux22 *ping 172.16.41.1.*

4.3.2 Experiência 4 Parte 2

Ligar os cabos

```
Tux42 E0    -> Switch Porta 2
Tux43 E0    -> Switch Porta 1
Tux44 E0    -> Switch Porta 3
Tux44 E1    -> Switch Porta 4
Router GE0  -> Switch Porta 5
Router GE1  -> Router Lab Network Porta 1
```

TUX43	eth0	SwitchPort 1
TUX44	eth0	SwitchPort 3
GE0	-	SwitchPort 5
TUX42	eth0	SwitchPort 2
TUX44	eth1	SwitchPort 4

Tabela 4: Ligações no Switch.

Configure commercial router RC and connect it (no NAT) to the lab network (172.16.1.0/24)

```
>interface gigabitethernet 0/0
>ip address 172.16.41.254    255.255.255.0
>no shutdown
>exit
>show interface gigabitethernet 0/0

>interface gigabitethernet 0/1
>ip address 172.16.1.29    255.255.255.0
>no shutdown
>exit
>show interface gigabitethernet 0/1
```

```
>ip route 0.0.0.0 0.0.0.0 172.16.1.254
>ip route 172.16.20.0 255.255.255.0 172.16.41.253
```

Modificar o ficheiro de configuração para NAT

```
conf t
interface gigabitethernet 0/0 *
ip address 172.16.41.254 255.255.255.0
no shutdown
ip nat inside
exit

# Defines Ethernet 1 with an IP address and as a NAT outside interface.
interface gigabitethernet 0/1 *
ip address 172.16.1.29 255.255.255.0
no shutdown
ip nat outside
exit

# Defines a NAT pool named ovrld with a range of a single IP  address, 172.16.1.29.
ip nat pool ovrld 172.16.1.29 172.16.1.29 prefix 24
ip nat inside source list 1 pool ovrld overload

# Access-list 1 permits packets with source addresses ranging from 172.16.40.0 through 1
access-list 1 permit 172.16.40.0 0.0.0.7
access-list 1 permit 172.16.41.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.1.254
ip route 172.16.40.0 255.255.255.0 172.16.41.253
end
```