

Rede de Computadores

(2º Trabalho Laboratorial)

Redes de Computadores ▪ L.EIC025

2022/2023 ▪ 1º Semestre

Turma: 3LEIC04

Carlos Sousa

up202005954

Daniela Tomás

up202004946

Fábio Rocha

up202005478

1- Sumário

No âmbito da unidade curricular de Redes de Computadores, foi-nos proposta a realização de um projeto composto por duas partes. A primeira parte consiste no desenvolvimento de uma aplicação de *download* de um ficheiro utilizando o protocolo FTP (*File Transfer Protocol*). Na segunda parte, foram efetuadas diversas experiências para a configuração de uma rede.

O objetivo do projeto foi concluído com sucesso, pois conseguimos criar uma aplicação capaz de transferir ficheiros e também configurar a rede corretamente.

2- Introdução

Com este relatório, pretendemos explicar de uma forma detalhada e organizada o funcionamento da aplicação de *download* de um ficheiro ao ser fornecido um URL da forma `ftp://[<user>:<password>@]<host>/<url-path>` e também o funcionamento da configuração da rede de computadores capaz de se ligar à internet e testar a aplicação de *download*. Assim, estruturámos o relatório da seguinte forma:

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

3- Aplicação *download*

3.1- Arquitetura

A primeira parte do trabalho consiste em desenvolver uma aplicação de *download* de ficheiros de acordo com o protocolo FTP. Para executarmos a aplicação, inicialmente é necessário compilar o programa e passar-lhe como argumento o seguinte link, que está de acordo com a sintaxe URL descrita no RFC173:

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

A aplicação inicialmente faz *parse* do URL, ou seja, cada componente fica armazenada num array. Caso o *user* e a *password* não sejam especificados, estes são assumidos como “anonymous” e “password”. Depois de interpretados os dados do URL

necessários para a execução da transferência é necessário comunicar com o servidor através de um *socket* TCP. De seguida, o programa irá tentar aceder ao servidor com as credenciais indicadas e, depois do login, o programa solicita que o servidor responda de modo passivo (PASV), pois é o servidor que decide para que porta o ficheiro será enviado. Abre-se outra conexão num novo *socket* para a transferência do ficheiro. Após ser transferido o ficheiro, os dois *sockets* são fechados.

3.2- Caso de sucesso

A aplicação foi testada com dois ficheiros distintos, *pic1.jpg* e *timestamp.txt*, com e sem autenticação no servidor (figura 3.2-A e 3.2-B).

```
danielatomas@LAPTOP-EUBEC4F1:/mnt/c/Users/35191/Desktop/UNI/3ºano/RCOM/FEUP-RCOM/proj2/code$ make
gcc -Wall download.c -o download
danielatomas@LAPTOP-EUBEC4F1:/mnt/c/Users/35191/Desktop/UNI/3ºano/RCOM/FEUP-RCOM/proj2/code$ ./download ftp://rcom:rcom@netlab1
.fe.up.pt/files/pic1.jpg

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

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

331 Please specify the password.

230 Login successful.
227 Entering Passive Mode (192,168,109,136,158,18).
Written 22 bytes
150 Opening BINARY mode data connection for /files/pic1.jpg (340603 bytes).

Receiving file pic1.jpg...
226 Transfer complete.
221 Goodbye.
danielatomas@LAPTOP-EUBEC4F1:/mnt/c/Users/35191/Desktop/UNI/3ºano/RCOM/FEUP-RCOM/proj2/code$
```

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

```
danielatomas@LAPTOP-EUBEC4F1:/mnt/c/Users/35191/Desktop/UNI/3ºano/RCOM/FEUP-RCOM/proj2/code$ ./download ftp://ftp.up.pt/pub/kodi/timestamp.txt

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

Host name : mirrors.up.pt
IP Address : 193.137.29.15
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220-----
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220

331 Please specify the password.

230 Login successful.
227 Entering Passive Mode (193,137,29,15,202,124).
Written 29 bytes
150 Opening BINARY mode data connection for pub/kodi/timestamp.txt (11 bytes).

Receiving file timestamp.txt...
226 Transfer complete.
221 Goodbye.
danielatomas@LAPTOP-EUBEC4F1:/mnt/c/Users/35191/Desktop/UNI/3ºano/RCOM/FEUP-RCOM/proj2/code$
```

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

4- Configuração e análise de redes

4.1- Experiência 1 - Configurar uma rede IP

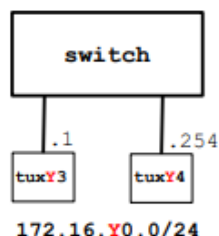


Figura 4.1: Experiência 1

O objetivo desta experiência é configurar duas máquinas, tux33 e tux34, sendo necessário a atribuição de um endereço IP para cada uma delas.

4.1.1- Análise de Logs

O protocolo ARP (*Address Resolution Protocol*) é um procedimento para mapear endereços de IP dinâmicos para um endereço de máquina físico numa rede local. Assim, associa o endereço IP com o endereço MAC, traduzindo endereços 32-bit para 48-bit e vice-versa, trabalhando entre a *data link layer* e a *network layer* do OSI model. Depois de apagar a tabela ARP dos computadores, o protocolo ARP foi executado.

O endereço MAC é o endereço da máquina física e localiza-se na *data link layer*. Já endereço IP é um endereço exclusivo que identifica um dispositivo na Internet ou numa rede local, localizando-se na *network layer*. Uma máquina pode possuir vários endereços IP, mas apenas um endereço MAC. Como o ARP é um protocolo de requisição e resposta, a requisição é feita via *broadcast*, solicitando um endereço MAC de uma máquina através do endereço IP. Por sua vez, a resposta é fornecida de forma *Unicast* pela máquina com o endereço lógico requisitado, contendo o endereço físico da mesma.

O comando *ping* gera ICMPs *requests* para o *host* desejado e espera pela resposta ICMP, assim, são reportados erros, perda de pacotes e um sumário estatístico de resultados. Nesta experiência, o comando *ping* serve para descobrir se existe conectividade entre o tux33, que tem como endereço de IP 172.16.30.1 e endereço MAC 00:21:5a:61:24:92, e o tux34, com endereço IP de 172.16.30.254 e endereço MAC 00:21:5a:5a:7d:74.

Para conseguirmos determinar o tipo de trama recebida, deve-se analisar o cabeçalho da trama. Se o valor for 0x0806, então é do tipo ARP. No caso de ter valor 0x0800, a trama é do tipo IP. Se depois se verificar que o cabeçalho do IP é 1, então é do tipo ICMP. Se for uma trama IP, essa informação encontra-se no seu cabeçalho. Para identificarmos o tamanho da trama recebida, usamos o Wireshark.

Ainda analisando o comando *ping* através do Wireshark, podemos verificar o envio de tramas *loopback* pelo emissor. A *loopback interface* é uma interface virtual que está sempre ativa e acessível desde que pelo menos uma das interfaces IP no *switch* esteja operacional. Assim, uma *loopback interface* é útil para tarefas de *debugging*, pois seu endereço IP pode sempre receber *ping* se qualquer outra interface do *switch* estiver ativa.

4.1.2- Principais comandos

O Configuração do endereço IP de cada computador:

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

Para verificar se conexão entre os dois computadores foi feita corretamente:

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

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

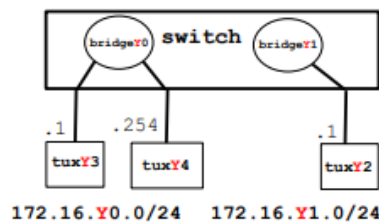


Figura 4.2: Experiência 2

O objetivo desta experiência é criar duas *bridges* no *switch*, e perceber a conectividade entre os computadores e como estas influenciam a troca de informação entre as máquinas.

4.2.1- Análise de Logs

Para configurarmos as *bridges*, inicialmente criamos a bridge30 e bridge31 através do *GTKterm*, e associamos à primeira o tux33 e o tux34 e à segunda o tux32, tendo assim a arquitetura pretendida. Depois, podemos verificar a conectividade entre os computadores, fazendo um *ping* do tux33 até ao tux34, que foi executado com sucesso, pois estes encontram-se na mesma sub-rede. Contudo, fazendo um *ping* do tux33 até ao tux32, não obtivemos resposta, pois não existe nenhuma rota entre as duas *bridges*, tornando impossível o tux33 chegar à interface de rede do tux32. O mesmo acontece se fizermos um *ping* entre o tux34 e o tux33. Portanto, podemos concluir que existem dois domínios de *broadcast* correspondentes às sub-redes bridge30 e bridge31.

4.2.2- Principais comandos

Por forma a configurar as *bridges* no *GTKTerm*, para criar as *bridges*:

- `/interface bridge add name=bridgeY0`

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

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

Por fim utilizamos o comando *ping* pela mesma lógica já referida anteriormente.

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

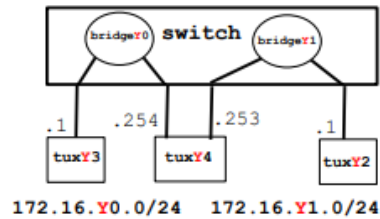


Figura 4.3: Experiência 3

O objetivo desta experiência é configurar o tux34 de modo a este funcionar como um *router*, para possibilitar a comunicação entre o tux33 e o tux32 através das *bridges* configuradas na experiência anterior. Para permitir a comunicação é necessário configurar os endereços IP's das portas e *ethernet* dos computadores e as rotas que serão usadas.

4.3.1- Análise de Logs

Para conseguirmos ter ligação entre o tux33 e o tux32, é necessário adicionar uma rota ao tux33 para aceder aos endereços 172.16.31.0/24 a partir do IP 172.16.30.254. Assim, quando o tux33 quer enviar um *ping* para a bridge31 este vai utilizar o router, que neste caso é o tux34(172.16.30.254), como *gateway* e uma rota ao tux32 para aceder aos endereços 172.16.30.0/24 a partir do IP 172.16.31.253. Estas rotas podem ser vistas na *forwarding table* utilizando o comando *route -n*. A *forwarding table* é uma *table* onde cada entrada possui informação do tipo *Destino-Gateway-Interface*, em que o destino é o IP do computador de destino. O *gateway* é o IP do computador para o qual se vai enviar a mensagem e a interface é a placa de rede usada para enviar a mensagem. Assim, com estas rotas definidas, é possível fazer *ping*, a partir do tux33, a todas as interfaces dos outros computadores.

Com isto, ao fazer *ping* do tux33 para o tux32 são observados os seguintes pacotes ARP. Primeiro é observado um pacote que pede que o endereço MAC do IP 172.16.30.254 seja enviado para 172.16.30.1, o endereço MAC observado é o 00:21:5a:5a:7d:74. Isto acontece porque o tux33 está a encontrar a interface do tux34 que encaminha o pacote enviado pelo *ping* para o tux32, o endereço MAC de resposta corresponde à interface eth0 de tux34. Em seguida, na mesma interface é observado o pacote ARP que pede que o endereço MAC de 172.16.30.1 seja enviado para 172.16.30.254. Isto acontece porque o tux34 necessita do endereço MAC da interface do tux33 para encaminhar a resposta do tux32 ao *ping*. O endereço MAC observado é o 00:21:5a:61:24:92. Já na eth1 do tux34, o tux34 pede qual o endereço MAC do tux32

para lhe poder encaminhar o pacote de *ping* e este dá o endereço MAC de eth0 do tux32. Depois, o tux32 pede o endereço MAC de eth1 do tux34 para lhe poder enviar a resposta a *ping* para que esta seja encaminhada a tux33. O tux34 responde com o endereço MAC de eth1 do tux34. Esta troca de mensagens ARP ocorre sempre que uma mensagem é enviada de uma máquina para outra sendo que os endereços MAC não são conhecidos.

Observando os pacotes ICMP, podemos verificar pacotes do tipo *request* e *reply*, pois todas as rotas estão configuradas. Se estas não se encontrassem configuradas seriam ver pacotes ICMP do tipo *Host Unreachable*. Os endereços de IP de destino associados com os pacotes ICMP correspondem sempre aos IP's da nossa máquina, enquanto os endereços de IP associados com os pacotes ICMP de origem correspondem aos IP's pelos quais os pacotes viajam para alcançarem o destino desejado. O endereço MAC de destino associado com os pacotes ICMP corresponde à interface virtual, enquanto o endereço MAC de origem associado com os pacotes ICMP é o endereço MAC da interface virtual do computador *host*.

4.3.2- Principais comandos

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

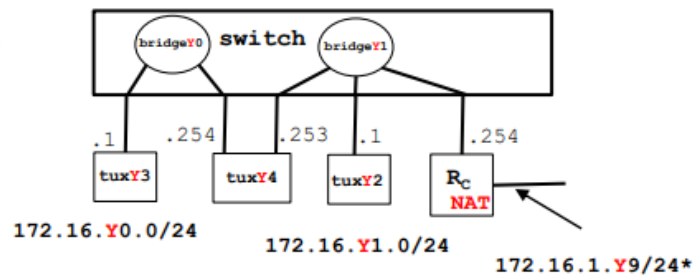


Figura 4.4: Experiência 4

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

4.4.1- Análise de Logs

Inicialmente necessitamos de configurar o *router*, para isso iniciamos a sessão no *router* a partir do *GTKTerm*, depois ligamos o cabo S0 de qualquer tux à entrada de configuração do *router*. Com isto temos acesso ao *router* através do *GTKTerm*, e podemos configurar o IP do *router* e também as rotas, utilizando o comando *ip route*.

Existem dois cenários possíveis para o que o computador possa ter acesso a internet, sendo o primeiro em que existe uma rota configurada entre as duas máquinas e assim a informação segue essa rota, e no segundo caso em que essa rota não existe e os pacotes são enviados pela rota *default* e a máquina para qual a rota *default* está definida irá enviar os pacotes para o destino. Como nesta experiência a rota do tux32 para o

tux33 foi apagada, os pacotes foram redirecionados para o *router* e de seguida foram passados para o tux34.

O NAT é um mecanismo implementado em *routers* que substitui os endereços IP locais nos pacotes por um endereço IP público de forma a se conseguir estabelecer uma ligação para fora da rede. Sendo assim, o *router* que implementa o NAT torna-se responsável por encaminhar todos os pacotes para o endereço correto, dentro ou fora da rede local. Por exemplo, nesta experiência supondo que o tux33 quer enviar um pacote para um endereço em uma rede pública, o pacote primeiramente é enviado para o *router* que irá modificar o endereço *source* do pacote, para o seu endereço exterior assim assegurando a privacidade e a segurança do seu remetente. O pacote é enviado, e obtêm como resposta um pacote. O *router*, ao receber esse pacote, reenvia-o para tux33, alterando o destinatário do pacote para o seu endereço, possibilitando a comunicação entre a rede privada local e a rede pública.

4.4.2- Principais comandos

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

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

No tuxy2 definir RC como *default router*:

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

No tuxy3 definir tuxy4 como *default router*:

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

No tuxy4 definir RC como *default router*:

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

4.5- Experiência 5 – DNS

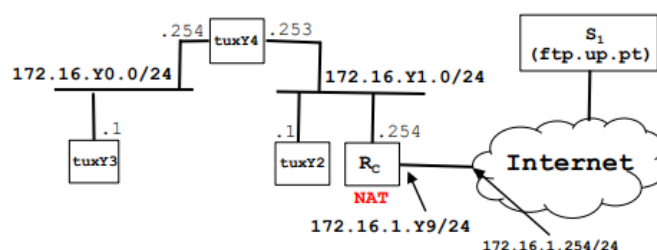


Figura 4.5: Experiência 5

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

4.5.1- Análise de Logs

O serviço DNS é configurado no ficheiro *resolv.conf*, que se localiza no diretório */etc/* do determinado tux, bastando adicionar no ficheiro:

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

O *host* envia para o server um pacote com o *hostname*, esperando que seja retornado o seu endereço IP. O servidor responde com um pacote que contém o endereço IP do *hostname* em causa.

4.5.2- Principais comandos

Inicialmente é necessário em cada tux, editar o ficheiro *resolv.conf*, de modo a configurarmos o DNS. Para isso, executamos o seguinte comando no terminal:

- echo '\$search netlab.fe.up.pt\nnameserver 172.16.1.1' > /etc/resolv.conf

4.6- Experiência 6 – Conexões TCP

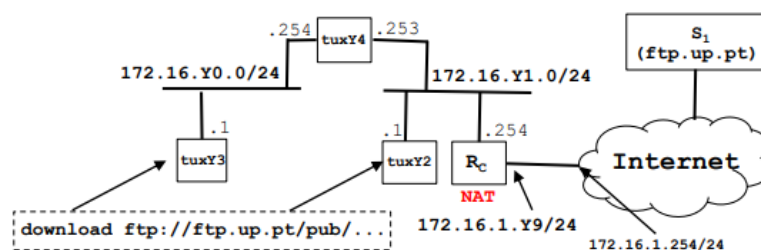


Figura 4.6: Experiência 6

O objetivo desta experiência é observar o comportamento e funcionamento do protocolo TCP, usando a aplicação que desenvolvemos.

4.6.1- Análise de Logs

A aplicação que elaboramos foi compilada e executada. Durante a execução desta são abertas duas conexões TCP, uma quando se entra em contacto com o servidor, para o controlo da informação tratando do envio e receção de comandos, e outra para fazer a transferência do ficheiro. Numa conexão TCP encontra-se dividida em três fases: inicialmente estabelece-se conexão, de seguida ocorre a troca de dados e finalmente a conexão é encerrada.

O ARQ TCP é um método de controlo de erros para transmissão de dados que usa confirmações (mensagens enviadas pelo recetor indicando que recebeu corretamente o pacote) e *timeouts* (períodos de tempo especificados antes que uma confirmação seja recebida) para obter uma transmissão de dados confiável através de um canal de comunicação não confiável. Se o remetente não receber uma confirmação antes do tempo limite, ele retransmite o pacote até receber uma confirmação ou exceder um número predefinido de retransmissões.

Relativamente ao mecanismo de controlo de congestão do TCP, este tem como base os ACKs recebidos na transmissão dos pacotes. É utilizada uma nova variável por conexão, *CongestionWindow*, de modo a regular o tamanho da janela deslizante de transmissão de pacotes tendo em conta a congestão da conexão. O valor desta é incrementando se a congestão da rede diminuir e decrementado se a congestão da rede aumentar. Quando se deteta que um pacote é perdido, o valor da *CongestionWindow* passa para metade.

Quando iniciamos uma segunda conexão TCP a taxa de transmissão de pacotes da conexão TCP diminui. Isto deve-se ao facto de quando aumentamos o número de conexões TCP a largura da banda disponível para cada uma das conexões diminui. Com isto, o *download* do ficheiro do servidor TCP leva mais tempo.

4.6.2- Principais comandos

Nesta experiência apenas compilamos e executamos a nossa aplicação download.

5- Conclusões

Podemos concluir que tanto o objetivo do desenvolvimento de uma aplicação que permite o *download* de um ficheiro através de conexões utilizando os protocolos FTP e TCP, com a configuração de uma rede IP, foram atingidos com sucesso. Desta forma foi possível perceber como funciona o *router* e a *switch*, as diversas técnicas (NAT, DNS, etc), os protocolos (ICMP, ARP, etc) e as estruturas de dados que são utilizados para efetuar a comunicação entre as máquinas, exemplos disso são as tabelas ARP e as *forwarding tables*. Além disso, a análise dos vários logs das experiências capturados das diferentes máquinas tux através do Wireshark permitiu um bom entendimento de como as diferentes camadas de comunicação participam dentro de uma rede de computadores. Em suma, podemos afirmar com toda a convicção que adquirimos conhecimentos de extrema relevância ao longo do desenvolvimento projeto.

6- Anexo I – Aplicação *download*

6.1- *download.c*

```
7.  #include <stdio.h>
8.  #include <stdlib.h>
9.  #include <netdb.h>
10. #include <netinet/in.h>
11. #include <arpa/inet.h>
12. #include <sys/socket.h>
13. #include <arpa/inet.h>
14. #include <unistd.h>
15. #include <string.h>
16. #define MAX_SIZE 512
17. #define FTP_PORT 21
18.
19. int main(int argc, char **argv) {
20.
```

```

21.     struct hostent* h;
22.     char buf[MAX_SIZE], user[50], password[50], host[100],
urlPath[MAX_SIZE];
23.
24.     if (argc != 2) {
25.         fprintf(stderr, "Usage: %s
ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
26.         exit(-1);
27.     }
28.
29.     sscanf(argv[1], "ftp://%511s", buf);
30.
31.     size_t i = strcspn(buf, ":"), size = strlen(buf);
32.
33.     if(i < size) {
34.         size_t j = strcspn(buf, "@");
35.
36.         if(j == size) {
37.             fprintf(stderr, "Usage: %s
ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
38.             exit(-1);
39.         }
40.
41.         strncpy(user, buf, i);
42.         user[i] = '\0';
43.         strncpy(password, buf+i+1, j-i-1);
44.         password[j-i-1] = '\0';
45.         i = strcspn(buf+j+1, "/");
46.         strncpy(host, buf+j+1, i);
47.         host[i] = '\0';
48.         strcpy(urlPath, buf+j+i+1);
49.     }
50.     else {
51.         i = strcspn(buf, "/");
52.         strncpy(host, buf, i);
53.         host[i] = '\0';
54.         strcpy(urlPath, buf+i+1);
55.         strcpy(user, "anonymous");
56.         strcpy(password, "password");
57.     }
58.
59.     printf("\nuser: %s\npassword: %s\nhost: %s\nurl-path:
%s\n\n", user, password, host, urlPath);
60.
61.     if ((h = gethostbyname(host)) == NULL) {
62.         perror("gethostbyname()");
63.         printf("%s\n", host);
64.         exit(-1);
65.     }

```

```

66.
67.     printf("Host name  : %s\n", h->h_name);
68.     printf("IP Address : %s\n", inet_ntoa(*((struct in_addr*) h-
>h_addr_list[0])));
69.
70.     //server address handling
71.     struct sockaddr_in server_addr;
72.     bzero((char*) &server_addr,sizeof(server_addr));
73.     server_addr.sin_family = AF_INET;
74.     server_addr.sin_addr.s_addr = inet_addr(inet_ntoa(*((struct
in_addr*) h->h_addr_list[0]))); //32 bit Internet address network
byte ordered
75.     server_addr.sin_port = htons(FTP_PORT); //server TCP port must be
network byte ordered
76.
77.     //open a TCP socket
78.     int sockfd;
79.     if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
80.         perror("socketfd\n");
81.         exit(-1);
82.     }
83.
84.     //connect to the server
85.     if (connect(sockfd,(struct sockaddr*)
&server_addr,sizeof(server_addr)) < 0) {
86.         printf("%s\n",h->h_addr_list[0]);
87.         perror("connect\n");
88.         exit(-1);
89.     }
90.
91.     //send a string to the server
92.     usleep(100000);
93.     int bytesRead = read(sockfd,buf,MAX_SIZE);
94.     buf[bytesRead] = '\0';
95.     printf("%s\n",buf);
96.
97.     if(strncmp(buf,"220",3) != 0) {
98.         perror("220\n");
99.         exit(-1);
100.    }
101.
102.    char buf2[519];
103.    sprintf(buf2,"user %s\r\n",user);
104.    size_t bytes = write(sockfd, buf2, strlen(buf2));
105.    bytesRead = read(sockfd,buf,MAX_SIZE);
106.    buf[bytesRead] = '\0';
107.    printf("%s\n",buf);
108.
109.    if(strncmp(buf,"331 Please specify the password.",32) != 0) {

```

```

110.     perror("331 Please specify the password.\n");
111.     exit(-1);
112. }
113.
114.
115.     sprintf(buf2,"pass %s\r\n",password);
116.     write(sockfd,buf2,strlen(buf2));
117.     bytesRead = read(sockfd,buf,MAX_SIZE);
118.     buf[bytesRead] = '\0';
119.     printf("%s",buf);
120.
121.     if(strncmp(buf,"230",3) != 0) {
122.         perror("230\n");
123.         exit(-1);
124.     }
125.
126.     strcpy(buf2,"pasv\r\n");
127.     write(sockfd,buf2,strlen(buf2));
128.     bytesRead = read(sockfd,buf,MAX_SIZE);
129.     buf[bytesRead] = '\0';
130.     printf("%s",buf);
131.
132.     if(strncmp(buf,"227",3) != 0) {
133.         perror("227\n");
134.         exit(-1);
135.     }
136.
137.     int h1 = 0, h2 = 0, h3 = 0, h4 = 0, p1 = 0, p2 = 0, pasvport;
138.     sscanf(buf,"227 Entering Passive Mode
139. (%i,%i,%i,%i,%i,%i).",&h1,&h2,&h3,&h4,&p1,&p2);
140.     pasvport = p1*256+p2;
141.     //server address handling
142.     struct sockaddr_in server_addrC;
143.     sprintf(buf,"%i.%i.%i.%i",h1,h2,h3,h4);
144.     bzero((char*) &server_addrC,sizeof(server_addrC));
145.     server_addrC.sin_family = AF_INET;
146.     server_addrC.sin_addr.s_addr = inet_addr(buf); //32 bit Internet
147.     server_addrC.sin_port = htons(pasvport); //server TCP port must
148.     //be network byte ordered
149.
150.     int sockfdC;
151.     if ((sockfdC = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
152.         perror("socketfdC\n");
153.         exit(-1);
154.     }
155.     //connect to the server

```

```

156.     if (connect(sockfdC,(struct sockaddr*)
    &server_addrC,sizeof(server_addrC)) < 0) {
157.         perror("connect\n");
158.         exit(-1);
159.     }
160.
161.     sprintf(buf2,"retr %s\r\n",urlPath);
162.     bytes = write(sockfd,buf2,strlen(buf2));
163.
164.     if (bytes > 0) {
165.         printf("Written %ld bytes\n", bytes);
166.     }
167.     else {
168.         perror("write\n");
169.         exit(-1);
170.     }
171.
172.     bytesRead = read(sockfd,buf,MAX_SIZE);
173.     buf[bytesRead] = '\0';
174.     printf("%s\n", buf);
175.
176.     if(strncmp(buf,"150",3) != 0) {
177.         perror("150\n");
178.         exit(-1);
179.     }
180.
181.     char* filename = strrchr(urlPath, '/');
182.
183.     printf("Receiving file %s...\n",filename+1);
184.
185.     FILE* fd = fopen(filename+1,"w");
186.
187.     if(!fd) {
188.         perror("fopen\n");
189.         exit(-1);
190.     }
191.
192.     while((bytesRead = read(sockfdC,buf,MAX_SIZE)) > 0){
193.         fwrite(buf,1,bytesRead,fd);
194.         //printf("%s",buf);
195.     }
196.
197.     fclose(fd);
198.
199.     do {
200.         memset(buf, 0, MAX_SIZE);
201.         read(sockfd,buf,MAX_SIZE);
202.     } while (buf[0] < '1' || buf[0] > '5' || buf[3] != ' ');
203.     printf("%s", buf);

```

```

204.     buf[3] = '\0';
205.
206.     if(strncmp(buf,"226",3) != 0) {
207.         perror("226\n");
208.         exit(-1);
209.     }
210.
211.     sprintf(buf2,"QUIT\r\n");
212.     write(sockfd,buf2,strlen(buf2));
213.     bytesRead = read(sockfd,buf,MAX_SIZE);
214.     buf[bytesRead] = '\0';
215.     printf("%s",buf);
216.
217.     if(strncmp(buf,"221",3) != 0) {
218.         perror("221\n");
219.         exit(-1);
220.     }
221.
222.     if (close(sockfd) < 0) {
223.         perror("close sockfd\n");
224.         exit(-1);
225.     }
226.
227.     if (close(sockfdC) < 0) {
228.         perror("close sockfdC\n");
229.         exit(-1);
230.     }
231.
232.     return 0;
233. }
234.

```

6.2- Makefile

```

1.  CC = gcc
2.  CFLAGS = -Wall
3.
4.  .PHONY: all
5.  all:
6.      $(CC) $(CFLAGS) download.c -o download
7.
8.  .PHONY: clean
9.  clean:
10.      rm -f download
11.
12.  #////////////////////////////////////
13.  .PHONY: pic1
14.  pic1:

```

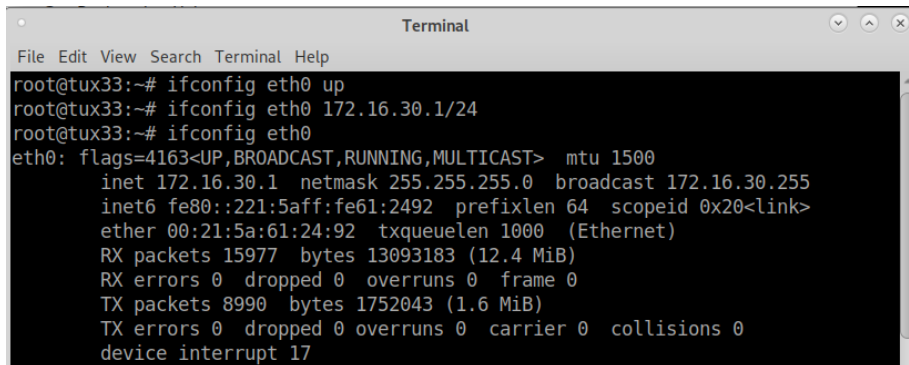
```

15.      ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/pic1.jpg
16.
17. .PHONY: pipe
18. pipe:
19.      ./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt
20.

```

7- Anexo II – Configuração de comandos e logs capturados

7.1- Experiência 1 - Configurar uma rede IP

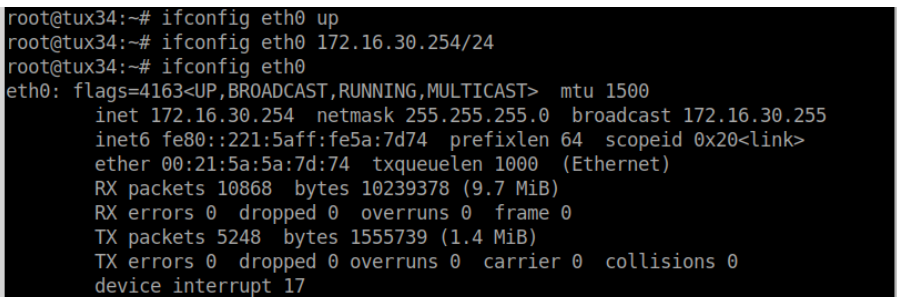


```

Terminal
File Edit View Search Terminal Help
root@tux33:~# ifconfig eth0 up
root@tux33:~# ifconfig eth0 172.16.30.1/24
root@tux33:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.30.1 netmask 255.255.255.0 broadcast 172.16.30.255
    inet6 fe80::221:5aff:fe61:2492 prefixlen 64 scopeid 0x20<link>
    ether 00:21:5a:61:24:92 txqueuelen 1000 (Ethernet)
    RX packets 15977 bytes 13093183 (12.4 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8990 bytes 1752043 (1.6 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 17

```

Figura 7.1-A: ifconfig no tux33

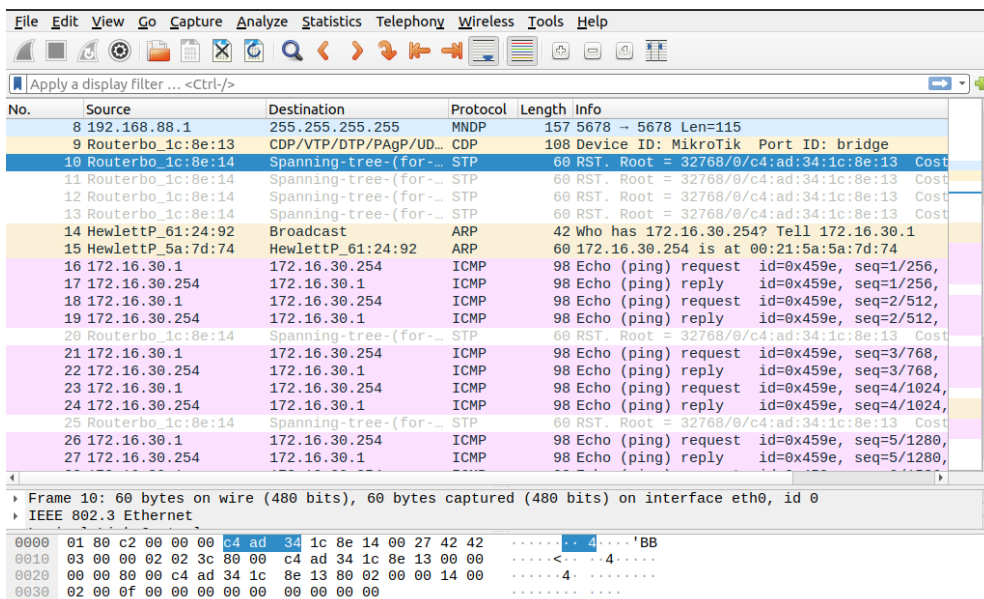


```

root@tux34:~# ifconfig eth0 up
root@tux34:~# ifconfig eth0 172.16.30.254/24
root@tux34:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.30.254 netmask 255.255.255.0 broadcast 172.16.30.255
    inet6 fe80::221:5aff:fe5a:7d74 prefixlen 64 scopeid 0x20<link>
    ether 00:21:5a:5a:7d:74 txqueuelen 1000 (Ethernet)
    RX packets 10868 bytes 10239378 (9.7 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 5248 bytes 1555739 (1.4 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 17

```

Figura 7.1-B: ifconfig no tux34



No.	Source	Destination	Protocol	Length	Info
8	192.168.88.1	255.255.255.255	MNDP	157	5678 → 5678 Len=115
9	Routerbo_1c:8e:13	CDP/VTP/DTP/PAGP/UD...	CDP	108	Device ID: Mikrotik Port ID: bridge
10	Routerbo_1c:8e:14	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost
11	Routerbo_1c:8e:14	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost
12	Routerbo_1c:8e:14	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost
13	Routerbo_1c:8e:14	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost
14	HewlettP_61:24:92	Broadcast	ARP	42	Who has 172.16.30.254? Tell 172.16.30.1
15	HewlettP_5a:7d:74	HewlettP_61:24:92	ARP	60	172.16.30.254 is at 00:21:5a:5a:7d:74
16	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x459e, seq=1/256,
17	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x459e, seq=1/256,
18	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x459e, seq=2/512,
19	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x459e, seq=2/512,
20	Routerbo_1c:8e:14	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost
21	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x459e, seq=3/768,
22	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x459e, seq=3/768,
23	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x459e, seq=4/1024,
24	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x459e, seq=4/1024,
25	Routerbo_1c:8e:14	Spanning-tree-(for...	STP	60	RST. Root = 32768/0/c4:ad:34:1c:8e:13 Cost
26	172.16.30.1	172.16.30.254	ICMP	98	Echo (ping) request id=0x459e, seq=5/1280,
27	172.16.30.254	172.16.30.1	ICMP	98	Echo (ping) reply id=0x459e, seq=5/1280,

Frame 10: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0

IEEE 802.3 Ethernet

```

0000 01 80 c2 00 00 00 00 c4 ad 34 1c 8e 14 00 27 42 42 .....4.....BB
0010 03 00 00 02 02 3c 80 00 c4 ad 34 1c 8e 13 00 00 .....4.....
0020 00 00 80 00 c4 ad 34 1c 8e 13 80 02 00 00 14 00 .....4.....
0030 02 00 0f 00 00 00 00 00 00 00 00 00 00 00 00 .....

```


Figura 7.1-C: Log experiência 1

7.2- Experiência 2 – Implementação de duas bridges no switch

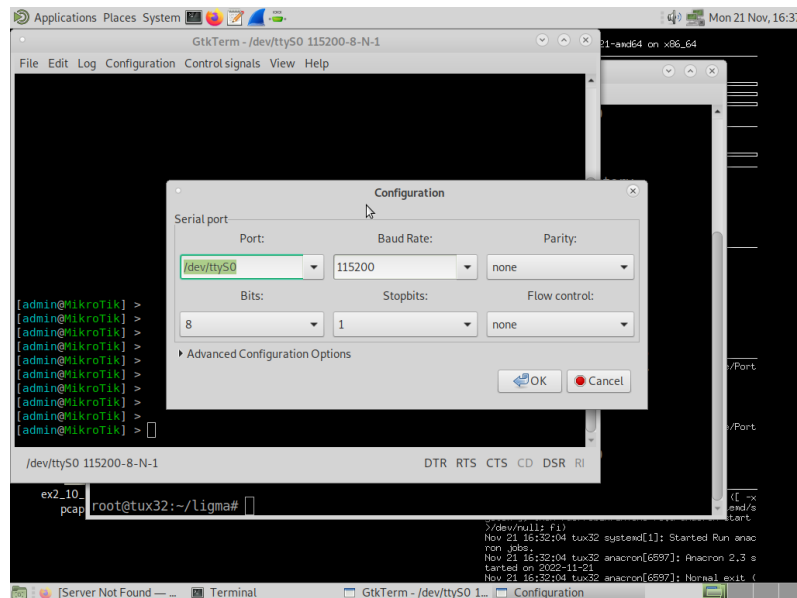


Figura 7.2-A: GtkTerm – Configuração do baudrate

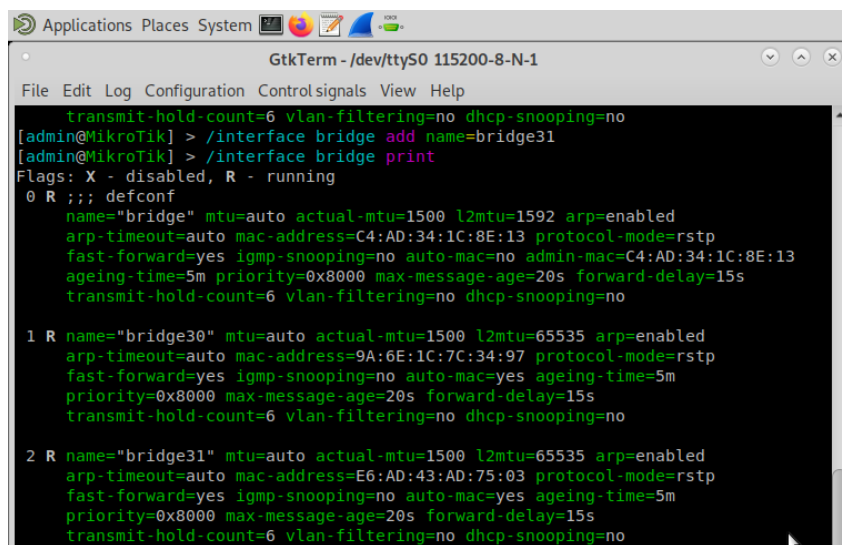


Figura 7.2-B: Criar bridges

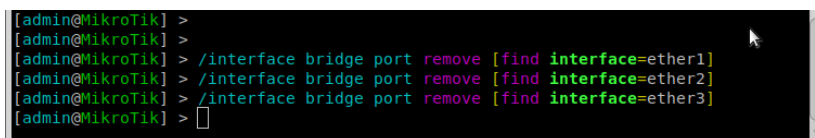


Figura 7.2-B: Remover portas das bridges

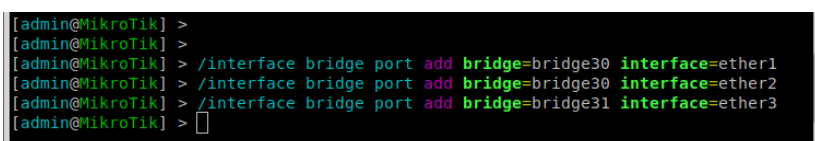


Figura 7.2-C: Adicionar portas às bridges

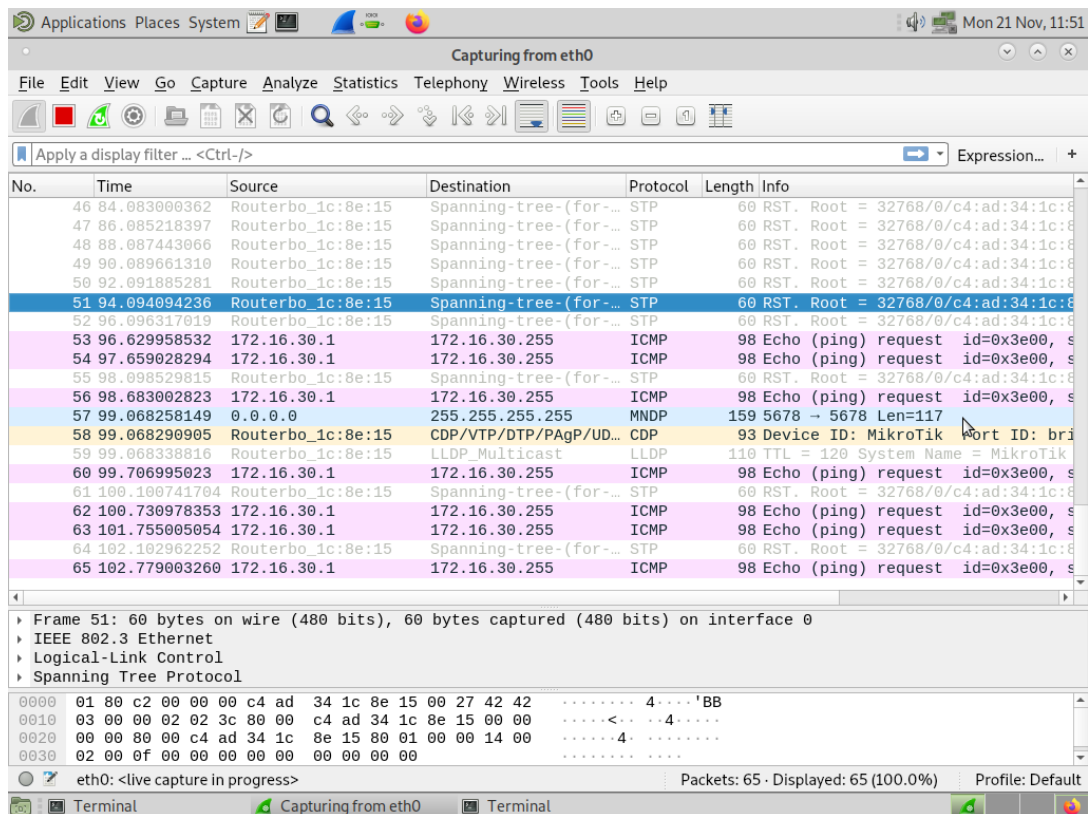


Figura 7.2-D: Ping 172.16.30.255 no tux33

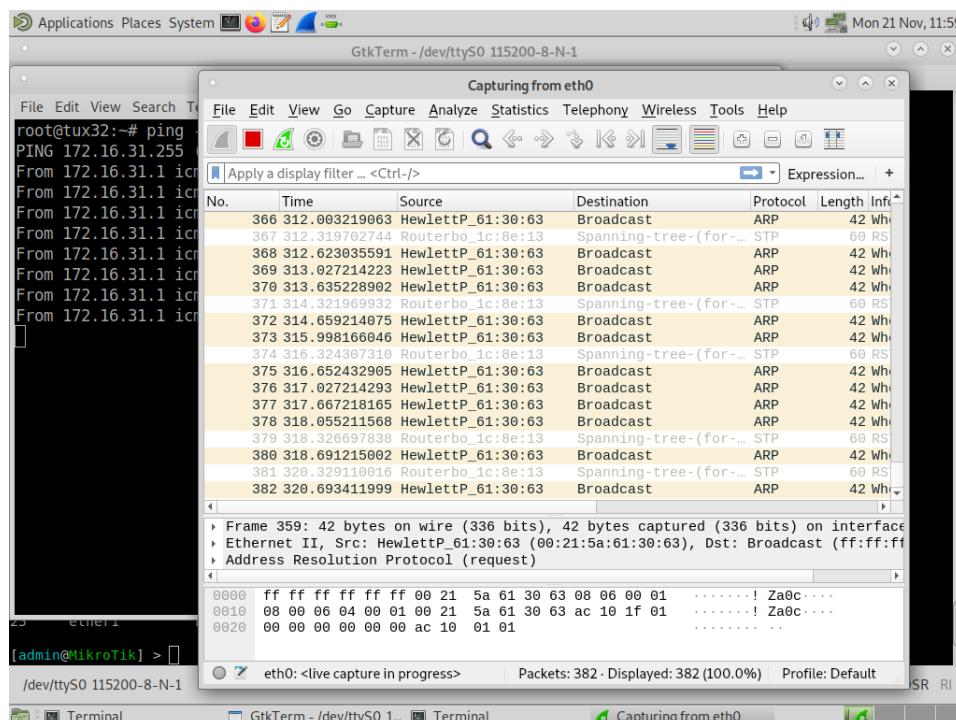


Figura 7.2-D: Ping broadcast tux32

7.3- Experiência 3 – Configuração de um router em Linux

```

Applications Places System
Terminal
File Edit View Search Terminal Help
root@tux34:~# ifconfig eth1 up
root@tux34:~# ifconfig eth1 172.16.31.253/24
root@tux34:~# ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.31.253 netmask 255.255.255.0 broadcast 172.16.31.255
    inet6 fe80::2c0:dfff:fe25:260a prefixlen 64 scopeid 0x20<link>
    ether 00:c0:df:25:26:0a txqueuelen 1000 (Ethernet)
    RX packets 59022 bytes 4856695 (4.6 MiB)
    RX errors 0 dropped 8116 overruns 0 frame 0
    TX packets 2401 bytes 238226 (232.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@tux34:~# echo 1 > /proc/sys/
abi/    crypto/ debug/  kdev/   fs/      kernel/ net/     user/   vm/
root@tux34:~# echo 1 > /proc/sys/net/ipv4/
Display all 112 possibilities? (y or n)
root@tux34:~# echo 1 > /proc/sys/net/ipv4/ip_forward
root@tux34:~# echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

```

Figura 7.3-A: Enable IP forwarding, disable ICMP echo-ignore-broadcast

```

root@tux34:~# arp -a
? (172.16.31.1) at 00:21:5a:61:30:63 [ether] on eth1
? (172.16.30.1) at 00:21:5a:61:24:92 [ether] on eth0
root@tux34:~#

```

Figura 7.3-B: Endereço MAC e IP no tux34.eth0 e tux34.eth1

24	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request	id=0x4c5a, seq=1/256,
25	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4c5a, seq=1/256,
26	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request	id=0x4c5a, seq=2/512,
27	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4c5a, seq=2/512,
28	Routerbo_1c:8e:14	Spanning-tree-(for-...	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8e:13	Cost
29	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request	id=0x4c5a, seq=3/768,
30	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4c5a, seq=3/768,
31	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request	id=0x4c5a, seq=4/1024,
32	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4c5a, seq=4/1024,
33	Routerbo_1c:8e:14	Spanning-tree-(for-...	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8e:13	Cost
34	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request	id=0x4c5a, seq=5/1280,
35	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4c5a, seq=5/1280,
36	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request	id=0x4c5a, seq=6/1536,
37	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4c5a, seq=6/1536,

Figura 7.3-C: Ping 172.16.30.254 no tux33

56	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request	id=0x4cec, seq=1/256,
57	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4cec, seq=1/256,
58	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request	id=0x4cec, seq=2/512,
59	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4cec, seq=2/512,
60	Routerbo_1c:8e:13	Spanning-tree-(for-...	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8e:13	Cost
61	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request	id=0x4cec, seq=3/768,
62	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4cec, seq=3/768,
63	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request	id=0x4cec, seq=4/1024,
64	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4cec, seq=4/1024,
65	Routerbo_1c:8e:13	Spanning-tree-(for-...	STP	60 RST. Root = 32768/0/c4:ad:34:1c:8e:13	Cost
66	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request	id=0x4cec, seq=5/1280,
67	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply	id=0x4cec, seq=5/1280,

Figura 7.3-D: Ping 172.16.31.1 no tux33

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

1	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
2	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
3	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=1/256,
4	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=1/256,
5	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=2/512,
6	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=2/512,
7	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
8	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=3/768,
9	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=3/768,
10	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=4/1024,
11	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=4/1024,
12	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
13	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=5/1280,
14	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=5/1280,
15	Routerbo_eb:24:12	HewlettP_61:24:01	ARP	60	Who has 172.16.31.1? Tell 172.16.31.254
16	HewlettP_61:24:01	Routerbo_eb:24:12	ARP	42	172.16.31.1 is at 00:21:5a:61:24:01
17	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=6/1536,
18	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=6/1536,
19	HewlettP_61:24:01	Routerbo_eb:24:12	ARP	42	Who has 172.16.31.254? Tell 172.16.31.1
20	Routerbo_eb:24:12	HewlettP_61:24:01	ARP	60	172.16.31.254 is at 74:4d:28:eb:24:12
21	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
22	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=7/1792,
23	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=7/1792,
24	172.16.31.1	172.16.31.254	ICMP	98	Echo (ping) request id=0x312d, seq=8/2048,
25	172.16.31.254	172.16.31.1	ICMP	98	Echo (ping) reply id=0x312d, seq=8/2048,
26	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost

Figura 7.4: Experiência 4

7.5- Experiência 5 – DNS

6	142.250.200.67	172.16.31.1	TCP	66	[TCP ACKed unseen segment] 80 → 39810 [ACK
7	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
8	172.16.31.1	172.16.2.1	DNS	87	Standard query 0x31bd PTR 67.200.250.142.ir
9	172.16.2.1	172.16.31.1	DNS	125	Standard query response 0x31bd PTR 67.200.2
10	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
11	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
12	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
13	172.16.31.1	172.16.2.1	DNS	87	Standard query 0x79f0 PTR 67.200.250.142.ir
14	172.16.2.1	172.16.31.1	DNS	125	Standard query response 0x79f0 PTR 67.200.2
15	HewlettP_61:24:01	Routerbo_eb:24:12	ARP	42	Who has 172.16.31.254? Tell 172.16.31.1
16	Routerbo_eb:24:12	HewlettP_61:24:01	ARP	60	172.16.31.254 is at 74:4d:28:eb:24:12
17	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
18	172.16.31.1	142.250.200.67	TCP	66	[TCP Dup ACK 5#1] 39810 → 80 [ACK] Seq=3302
19	142.250.200.67	172.16.31.1	TCP	66	[TCP Dup ACK 6#1] [TCP ACKed unseen segment]
20	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
21	172.16.31.1	172.16.2.1	DNS	87	Standard query 0xcba8 PTR 67.200.250.142.ir
22	172.16.2.1	172.16.31.1	DNS	125	Standard query response 0xcba8 PTR 67.200.2
23	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
24	172.16.31.1	172.16.2.1	DNS	71	Standard query 0x5551 A www.fnac.pt
25	172.16.31.1	172.16.2.1	DNS	71	Standard query 0x4b5b AAAA www.fnac.pt
26	172.16.2.1	172.16.31.1	DNS	178	Standard query response 0x5551 A www.fnac.p
27	172.16.2.1	172.16.31.1	DNS	221	Standard query response 0x4b5b AAAA www.fna
28	172.16.31.1	184.28.198.184	ICMP	98	Echo (ping) request id=0x41fc, seq=1/256,
29	184.28.198.184	172.16.31.1	ICMP	98	Echo (ping) reply id=0x41fc, seq=1/256,
30	172.16.31.1	172.16.2.1	DNS	87	Standard query 0x063f PTR 184.198.28.184.ir
31	172.16.2.1	172.16.31.1	DNS	153	Standard query response 0x063f PTR 184.198.
32	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
33	172.16.31.1	184.28.198.184	ICMP	98	Echo (ping) request id=0x41fc, seq=2/512,
34	184.28.198.184	172.16.31.1	ICMP	98	Echo (ping) reply id=0x41fc, seq=2/512,
35	172.16.31.1	184.28.198.184	ICMP	98	Echo (ping) request id=0x41fc, seq=3/768,
36	184.28.198.184	172.16.31.1	ICMP	98	Echo (ping) reply id=0x41fc, seq=3/768,
37	Routerbo_1c:a3:2b	Spanning-tree-(for-...	STP	60	RST. Root = 32768/0/74:4d:28:eb:24:12 Cost
38	172.16.31.1	184.28.198.184	ICMP	98	Echo (ping) request id=0x41fc, seq=4/1024,

Figura 7.5: Experiência 5