

# RCOM - 2º Trabalho Laboratorial

## Relatório Final



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

### Turma 1

Diogo Machado - up201706832

Eduardo Ribeiro - up201705421

José Guerra - up201706421

Faculdade de Engenharia da Universidade do Porto  
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

23 de Dezembro de 2019

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>4</b>
<b>2</b>	<b>Aplicação <i>download</i></b>	<b>4</b>
2.1	Arquitetura . . . . .	4
2.2	Resultados . . . . .	5
<b>3</b>	<b>Configuração e análise de redes</b>	<b>5</b>
3.1	Experiência 1 - Configurar uma rede IP . . . . .	5
3.2	Experiência 2 - Implementação de duas virtual LANs num switch . . . . .	7
3.3	Experiência 3 - Configuração de um router em Linux . . . . .	8
3.4	Experiência 4 - Configurar um router comercial e implementar NAT . . . . .	9
3.5	Experiência 5 - DNS . . . . .	11
3.6	Experiência 6 - Conexões TCP . . . . .	11
3.7	Experiência 7 - Implementação de NAT em Linux . . . . .	13
<b>4</b>	<b>Conclusões</b>	<b>13</b>

## **Sumário**

Este relatório ilustra e descreve o trabalho desenvolvido no 2º trabalho da cadeira de Redes de Computadores (RCOM), do 3º ano do curso MIEIC, na FEUP. As experiências foram realizadas maioritariamente nas aulas práticas da cadeira, com o equipamento fornecido pelos laboratórios da faculdade. Nas secções seguintes todo o trabalho é documentado e explicado, tanto a nível do processo como dos resultados obtidos.

# 1 Introdução

No âmbito da Unidade Curricular de Redes de Computadores, foi proposto o desenvolvimento de um projeto com duas partes. A primeira incide sobre o desenvolvimento de uma aplicação de *download* de ficheiros por FTP, e a segunda incide sobre uma série de experiências com as várias etapas da configuração de uma rede.

Quanto à primeira parte do projeto, a aplicação foi desenvolvida na linguagem C, e é capaz de transferir ficheiros do modo pretendido, não descurando a necessária robustez à ocorrência de erros ou a inputs inválidos do utilizador.

Em relação à segunda parte, as experiências foram realizadas com sucesso, tendo os objetivos sido alcançados, e tendo as principais dúvidas/questões relativas a estas sido respondidas.

## 2 Aplicação *download*

A primeira parte deste trabalho consistiu no desenvolvimento de uma aplicação de *download* (transferência) de ficheiros de acordo com o protocolo FTP (file transfer protocol) descrito no RFC959 e com ligações TCP (Transmission Control Protocol) a partir de sockets, sendo assim a aplicação é capaz de fazer o *download* de qualquer tipo de ficheiros de um servidor FTP. A aplicação aceita como argumento um link com a seguinte configuração:

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

Este link está de acordo com a sintaxe url descrita no RFC1738. No <url-path> foi colocado como meio de teste o servidor FTP ftp.fe.up.pt seguido do path do ficheiro que se pretende transferir (ftp.fe.up.pt/path-do-ficheiro). Foi necessário estudar com cuidado o RFC959 “Internet message protocol” para compreender a parte do FTP e o RFC1738 “Uniform Resource Locators (URL)” para compreender o tratamento de informação provenientes de URL’s.

### 2.1 Arquitetura

```
int parseArguments(struct arguments *args, char *commandLineArg);
int getAddress(char *ipAddress, char *hostName);
int createAndConnectSocket(char *address, int port);
int sendToControlSocket(struct ftp *ftp, char *cmdHeader, char *cmdBody);
int receiveFromControlSocket(struct ftp *ftp, char *string, size_t size);
int sendCommandInterpretResponse(struct ftp *ftp, char *cmdHeader, char *cmdBody,
                                  char *response, size_t responseLength, bool readingFile);
int login(struct ftp *ftp, char *username, char *password);
int getServerPortForFile(struct ftp *ftp);
int changeWorkingDirectory(struct ftp *ftp, char *path);
int downloadFile(struct ftp *ftp, char *fileName);
int retr(struct ftp *ftp, char *fileName);
int disconnectFromSocket(struct ftp *ftp);
```

Em primeiro lugar é necessário fazer o parsing (processamento) do url que é passado através da consola quando se corre a aplicação. Para esta funcionalidade temos uma função chamada **parseArguments** que guarda os valores resultantes do parsing (username, password, nome do host, path do ficheiro, nome do ficheiro) numa struct *arguments*.

Em seguida utiliza-se uma função chamada **getIPAddress** (fornecida) para a qual passamos o nome do host e que retorna o endereço IP correspondente. Com este endereço IP utilizamos a função **createAndConnectSocket** para conectar com o socket, sendo utilizada sempre a porta 21.

Para enviar comandos e receber respostas do socket utiliza-se a função **sendCommandInterpretResponse** que por sua vez utiliza a função **sendToControlSocket** para efetivamente enviar comandos e a função **receiveFromControlSocket** para receber respostas. Esta função também faz o processamento da resposta baseando-se no primeiro dígito da resposta. Caso o primeiro dígito seja 1,2,3 a resposta é de natureza positiva, caso seja 3 e 4 a resposta é de natureza negativa. Caso o primeiro dígito seja 1, então se se estiver a receber um ficheiro retorna, senão a função **receiveFromControlSocket** é chamada outra vez e a resposta volta a ser interpretada. Caso seja 2 ou 3, a função retorna, e, caso seja 4, tenta mandar outra vez o comando e ler a resposta. Caso seja 5, é detetado que ocorreu um erro ao mandar o comando, o socket é fechado e a aplicação termina.

Depois de aberto o socket é agora necessário fazer o login onde serão enviados os comandos USER user e Pass pass, para este feito temos a função **login**.

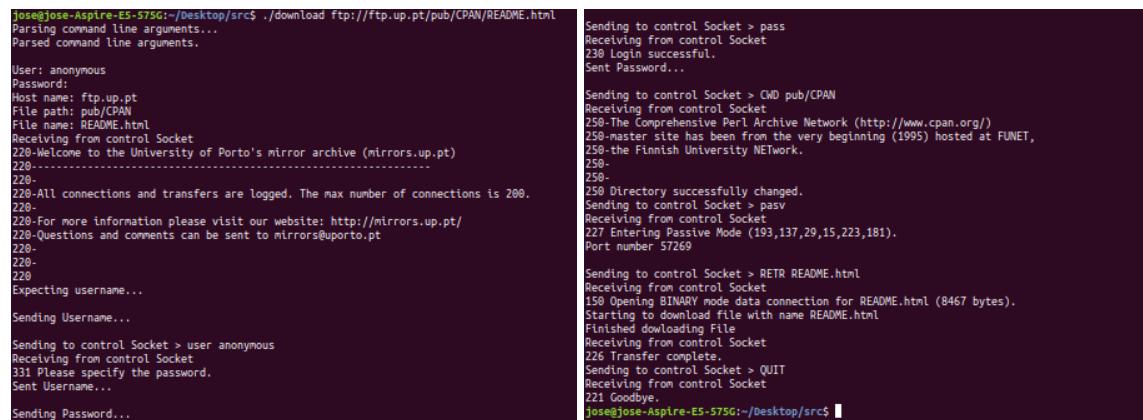
Depois é necessário mudar o diretório atual no servidor para onde se encontra o ficheiro que se pretende fazer o *download*, para isso temos a função **changeWorkingDirectory**.

Em seguida, é feita a entrada em modo passivo pelo envio para o servidor do comando PASV, que retorna a porta para abrir o socket que vai servir para receber o ficheiro. Tanto o envio do comando PASV bem como a abertura deste novo socket são feitos pela função **getServerPortForFile**.

Para pedir o ficheiro é necessário agora enviar o comando RETR filename para o novo socket, feito através da função **retr**, e depois fazer efetivamente o *download* do ficheiro, função **downloadFile**. Por fim é utilizada a função **disconnectFromSocket** que faz o fecho da ligação com o servidor através do envio do comando QUIT.

## 2.2 Resultados

A nossa aplicação funcionou como esperado, quer no caso de serem passadas as credenciais de login, quer no caso em que não (login é feito com o username *anonymous* e password vazia). Para além disso foi possível fazer o download de vários tipos de ficheiros e de diferentes tamanhos.



```
jose@jose-Aspire-E5-575G:~/Desktop/src$ ./download ftp://ftp.up.pt/pub/CPAN/README.html
Parsing command line arguments...
Parsed command line arguments.

User: anonymous
Password:
Host name: ftp.up.pt
File path: pub/CPAN
File name: README.html
Receiving from control Socket
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@porto.pt
220-
220-
220-
Expecting username...
Sending Username...
Sending to control Socket > user anonymous
Receiving from control Socket
331 Please specify the password.
Sent Username...
Sending Password...

Sending to control Socket > pass
Receiving from control Socket
230 Login successful.
Sent Password...

Sending to control Socket > CWD pub/CPAN
Receiving from control Socket
250-The Comprehensive Perl Archive Network (http://www.cpan.org/)
250-master site has been from the very beginning (1995) hosted at FUNET,
250-the Finnish University NETwork.
250-
250-
250-
250 Directory successfully changed.
Sending to control Socket > pasv
Receiving from control Socket
227 Entering Passive Mode (193,137,29,15,223,181).
Port number 57269

Sending to control Socket > RETR README.html
Receiving from control Socket
150 Opening BINARY mode data connection for README.html (8467 bytes).
Starting to download file with name README.html
Finished downloading file
Receiving from control Socket
226 Transfer complete.
Sending to control Socket > QUIT
Receiving from control Socket
221 Goodbye
jose@jose-Aspire-E5-575G:~/Desktop/src$
```

## 3 Configuração e análise de redes

*Nota: imagens representativas de cada uma das experiências no anexo do relatório.*

### 3.1 Experiência 1 - Configurar uma rede IP

O objetivo da experiência foi a configuração de duas máquinas, o *tuxy1* e o *tuxy4*, sendo que foi necessária a atribuição de um endereço IP a cada uma delas. Foi também implementada e observada a transmissão de mensagens entre elas.

Principais comandos:

- Para a atribuição de um endereço IP a uma máquina: *ifconfig <cartaDeRede> <endIP>/<mascara>*  
(ex: ifconfig eth0 172.16.30.1/24)
- Para a configuração de rotas: *route add -net <destino>/<mask> gw <endGateway>*
- Para o ping de uma máquina para outra: *ping <endDestino>;*

- O que são os pacotes ARP, e para o que são usados?

O protocolo ARP (Address Resolution Protocol) tem como função o mapeamento de um endereço IP de uma máquina a um endereço MAC da máquina na rede local. Um computador, ao tentar enviar um pacote a outro, na mesma rede local (admitindo que a tabela ARP não tem entradas para o IP da máquina que recebe), irá enviar um pacote ARP, por broadcast, para todas as máquinas da rede local perguntando qual das máquinas tem um endereço MAC que corresponde ao endereço IP do destinatário. O destinatário irá enviar outro pacote ARP que indica à máquina que envia qual o seu endereço MAC. Deste modo, a transferência de pacotes pode ser efetuada.

- Quais são os endereços MAC e IP dos pacotes ARP, e porquê?

Quando o *tuxy1* tenta enviar um pacote ao *tuxy4*, como a entrada da tabela ARP referente ao *tuxy4* foi apagada, o *tuxy1* não sabe qual é o endereço MAC associado ao endereço IP do *tuxy4* (172.16.30.254). Deste modo, irá enviar um pacote ARP em broadcast (para toda a rede local), sendo que este pacote contém o endereço IP (172.16.30.1) e o endereço MAC (00:0f:fe:8b:e4:4d) do *tuxy1*. O endereço MAC do destinatário, como não se conhece, tem o valor de 00:00:00:00:00:00.

De seguida, o *tuxy4* irá enviar um pacote ARP para o *tuxy1*, com o endereço MAC dele (00:21:5a:5a:7d:74), e o seu endereço IP (172.16.30.254).

Portanto, pode-se concluir que cada pacote ARP contém campos para os endereços MAC e IP da máquina que envia, e para os endereços MAC e IP da máquina que recebe.

- Que pacotes gera o comando *ping*?

O comando ping gera primeiro pacotes ARP para saber qual o endereço MAC do destinatário, tal como foi dito anteriormente; de seguida, gera pacotes ICMP (Internet Control Message Protocol).

- Quais são os endereços MAC e IP dos pacotes *ping*?

Quando é efetuado um ping do *tuxy1* para o *tuxy4*, os endereços dos pacotes enviados são os seguintes:

- Pacote request (de *tuxy1* para *tuxy4*):

- IP address da source: 172.16.30.1 (*tuxy1*)
- MAC address da source: 00:0f:fe:8b:e4:4d (*tuxy1*)
- IP address do destinatário: 172.16.30.254 (*tuxy4*)
- MAC address do destinatário: 00:21:5a:5a:7d:74 (*tuxy4*)

- Pacote reply (de *tuxy4* para *tuxy1*):

- IP address da source: 172.16.30.254 (*tuxy4*)
- MAC address da source: 00:21:5a:5a:7d:74 (*tuxy4*)
- IP address do destinatário: 172.16.30.1 (*tuxy1*)
- MAC address do destinatário: 00:0f:fe:8b:e4:4d (*tuxy1*)

- Como determinar se uma *frame* Ethernet é ARP, IP, ICMP?

Conseguimos determinar o tipo da trama recetora analisando o Ethernet header da trama:

- Caso este tenha o valor 0x0800, significa que a trama é do tipo IP. Já se tiver o valor 0x0806, então a trama é do tipo ARP.
- Caso a trama seja do tipo IP, podemos analisar o seu IP header. Se este header tomar o valor 1, então o tipo de protocolo é ICMP.

- Como determinar o tamanho de uma *frame* recebida?

O comprimento de uma trama pode ser visualizado através do software *Wireshark*.

- O que é a interface *loopback*, e qual a sua importância?

A interface loopback é uma interface virtual da rede que permite ao computador receber respostas de si próprio, para testar a correta configuração da carta de rede.

### 3.2 Experiência 2 - Implementação de duas virtual LANs num switch

Neste experiência, foram criadas duas *VLANs* no switch, sendo que duas máquinas (*tuxy1* e *tuxy4*) foram ligadas à primeira *VLAN*, e uma outra máquina (*tuxy2*) à segunda. O objetivo foi entender como é que se efetua a configuração destas *VLANs* e como é que estas permitem e influenciam a troca de informação entre as máquinas.

Principais comandos: Comandos para configurar as *VLANs* (detalhados mais à frente)

- Como configurar a *vlan y0*?

Estando ligado ao switch pela porta de série */dev/ttyS0*, de modo a ser possível executar os comandos no *GTKTerm*, e correspondendo *y* ao número da bancada:

- Criar as *VLANs*:

```
* configure terminal
* vlan y0
* vlan y1
* end
```

- Adicionar as portas às respetivas *VLANs*:

```
* configure terminal
* interface fastethernet 0/X (sendo X o número da porta, no switch, à qual o computador em questão está ligado)
* switchport mode access
* switchport access vlan yY (sendo Y o número da VLAN à qual se deve ligar o computador em questão)
* end
```

- Eliminar as *VLANs*, no fim do procedimento:

```
* configure terminal
* no vlan y0
* no vlan y1
* end
```

- Quantos domínios de transmissão existem? Como se pode concluir isso a partir dos *logs*?

Existem 2 dominios de broadcast já que ao fazer broadcast apenas abrange as portas pertencentes a essa virtual lan, tal como foi visto na experiência e reiterado pelos seguintes logs

### 3.3 Experiência 3 - Configuração de um router em Linux

Nesta experiência, o *tuxy4* foi configurado de modo a funcionar como um router, que está ligado a ambas as *VLANs*, o que permite a conexão e transmissão de informação entre máquinas em *VLANs* diferentes (*tuxy1* e *tuxy2*).

Principais comandos: *ifconfig*, *route*, comandos para configuração do router:

- Enabling IP forwarding: *echo 1 > /proc/sys/net/ipv4/ip\_forward*
- Disabling ignore broadcast: *echo 1 > /proc/sys/net/ipv4/icmp\_echo\_ignore\_broadcasts*

- Que rotas existem nos *tuxes*? Qual é o seu significado?

Algumas rotas são geradas automaticamente, “ligando” as máquinas às *VLANs* a que pertencem: o *tux31* tem uma rota para a *vlan30*, o *tux32* tem uma rota para a *vlan31*, e o *tux34* tem rota para as duas. O gateway para estas rotas é 0.0.0.0.

Para além disso, no *tux31* adicionou-se a rota - **route add -net 172.16.31.0/24 gw 172.16.30.254**, isto é, quando o *tux31* (172.16.30.1) quer enviar um ping ou mensagem para a *vlan31* (172.16.31.0) ele vai utilizar como gateway o router que é o *tux34* (172.16.30.254). Este *tux* vai ter uma forwarding table que torna isto possível.

No *tux34* fez-se algo similar ao que se fez no *tux31* só que no sentido inverso, criou-se uma rota - **route add -net 172.16.30.1/24 gw 172.16.31.253**, isto é, quando queremos enviar um ping/mensagem para a *vlan30* envia-se primeiro para o router (172.16.31.253).

- Que informação contém uma entrada da *forwarding table*?

Na tabela de forwarding cada entrada possui informação do tipo **[Destination - Gateway - Interface]**, em que a *destination* é o IP do computador/rede de destino, o *gateway* é o IP do computador para o qual vai ser mandada a mensagem e este é que vai dar routing da mensagem para o destino, e a *interface* é a placa de rede usada para enviar a mensagem, exemplo: *eth0*, *eth1*, etc.

Outras informações que também estão presentes para cada entrada na tabela são:

- *Netmask* - utilizado para determinar o ID da rede a partir do IP address do destino
- *Flags* - informações sobre a rota
- *Metric* - custo da rota
- *Ref* - número de referências para esta rota (não usado no kernel do linux).
- *Use* - contador de pesquisas pela rota, dependendo do uso de -F ou -C (número de falhas da cache e número de sucessos, respetivamente).

- Que mensagens ARP, e endereços MAC associados, são observados, e porquê?

Ocorre uma troca de mensagens ARP quando um *tux* (*tux1*) envia um ping a outro *tux* (*tux2*), e o *tux1* não conhece o MAC address de *tux2*. O *tux1* envia uma mensagem ARP, na qual pede o MAC address de *tux2*, através do seu IP. Quando uma mensagem ARP é enviada, o *tux* que envia ( neste caso, *tux1* ) associa o seu MAC address à mensagem, para que o receptor esperado da mensagem (*tux2*) saiba a que *tux* responder (“Who has [IP de *tux2*]? Tell [próprio IP]”). Esta mensagem será enviada no modo *broadcast* (MAC address do receptor tem o valor 00:00:00:00:00:00), porque o MAC address de *tux2* ainda é desconhecido. Quando recebe este *broadcast*, *tux2* responde com uma mensagem ARP, na qual fornece o seu MAC address (“[IP de *tux2*] is at [MAC address de *tux2*]”). Esta mensagem é enviada apenas para o MAC address de *tux1*, e não em modo *broadcast*.

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 (por exemplo, da interface *eth1* do *tux4* para o *tux2*).

- Que pacotes ICMP são observados, e porquê?

Podemos observar pacotes ICMP do tipo request e reply, já que, estando todas as routes adicionadas, todos os tuxes reconhecem a presença uns dos outros. Se não reconhecessem, os pacotes ICMP enviados seriam do tipo Host Unreachable.

- Quais são os endereços IP e MAC associados aos pacotes ICMP, e porquê?

Os endereços IP e MAC de origem e destino associados com os pacotes ICMP são os endereços MAC e IP das máquinas/interfaces que recebem/enviam os pacotes. Por exemplo, quando um pacote ICMP é enviado do *tuxy1* para a interface do *tuxy4* conectada à mesma subrede (no nosso caso foi utilizada a interface eth0), os endereços MAC e IP de origem serão os do *tuxy1* (IP: 172.16.30.1 e MAC: 00:0f:fe:8b:e4:4d) e os endereços IP e MAC de destino serão os associados à interface eth0 do *tuxy4* (IP: 172.16.30.254 e MAC: 00:21:5a:5a:7d:74).

Se as duas máquinas não estiverem conectadas à mesma virtual network (como é o caso do *tuxy1* e *tuxy2*), não é possível efetuar o envio de informação entre as duas máquinas com apenas um pacote ICMP sem este ser modificado. Neste caso, o que acontece é: o *tuxy1* irá enviar um pacote ICMP para a interface eth0 do *tuxy4*, uma vez que o *tuxy4* está ligado a ambas as *VLANs* e consegue interagir diretamente com o *tuxy2*; os endereços IP do pacote serão os IPs do *tuxy1* e *tuxy2* (origem e destino, respectivamente), e os endereços MAC serão os do *tuxy1* e o da interface eth0 do *tuxy4*. Ao receber este pacote, o *tuxy4* irá reencaminha-lo para o *tuxy2*, mantendo os endereços IP do pacote, mas os endereços MAC associados a este serão diferentes: o de origem será o da interface eth1 do *tuxy4* (esta interface está conectada à vlan onde se encontra o *tuxy2*), e o de destino será o endereço MAC do *tuxy2*.

### 3.4 Experiência 4 - Configurar um router comercial e implementar NAT

Esta experiência teve como objetivo a configuração de um router comercial, efetuando a ligação deste à rede do laboratório (172.16.1.0/24) e também a *vlany1*. Foi também efetuada a configuração do router de modo a este utilizar a técnica NAT para garantir a conexão entre as máquinas rede IP e a internet. O objetivo da experiência foi não só perceber de maneira é que o NAT influencia a conexão entre as máquinas e a Internet, mas também analisar e perceber o mecanismo de redirecionamento de pacotes ICMP.

Principais comandos: Configuração do router (e rotas do mesmo) e da NAT (slides 45 e 46 do guião)

- Como configurar uma rota estática num *router* comercial?

É necessário iniciar sessão no router através do gtkterm, para tal finalidade é necessário ligar através do cabo de série S0 de um TUX da bancada à entrada de configuração do router. Para configurar as routes (rotas) temos que executar o comando "ip route" dentro do gtkTerm. O comando ip route segue a seguinte estrutura:

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

Exemplo de adicionar uma route ao router:

- *configure terminal*
- *ip route [destino] [máscara] [gateway]*
- *exit*

- Quais são os caminhos seguidos pelos pacotes nas experiências efetuadas, e porquê?

No caso de a rota existir, os pacotes seguem essa mesma rota. Caso contrário, os pacotes são dirigidos ao router (pela rota default).

No passo 4 da experiência 4, ao desativar os redirecionamentos através dos comandos:

- echo 0 > /proc/sys/net/ipv4/conf/eth0/accept\_redirects
- echo 0 > /proc/sys/net/ipv4/conf/all/accept\_redirects

está-se a fazer com que, caso haja redirects na mesma interface de rede, o tux não guarde na sua lista de forwarding uma entrada resultante do redirect de um outro tux. No *tux32* foram feitos os dois comandos em cima, desativando os redirects, para além disso foi definido tanto para o *tux34* como para o *tux32* como default route o router Rc.

O *tux34* é único que tem comunicação com o *tux31* através da *vlan30*. O *tux34*, o *tux32* e o router estão todos ligados à *vlan31*. O router é o único que sabe que para chegar ao *tux31* é necessário utilizar como gateway o *tux34*, para isso adicionou-se a seguinte route ao router "ip route 172.16.30.1 255.255.255.0 172.16.31.253". Quando se envia um ping para o *tux31* a partir do *tux32* como este não tem conhecimento da rota para o *tux31* e eles não estão na mesma interface (rede), o *tux32* vai, em primeiro lugar, enviar o ping para o router, que é a sua rota default, e depois este como tem conhecimento da rota para o *tux31* envia para o *tux34* e este reencaminha finalmente para o *tux31* o ping. Nos seguintes pings o caminho vai ser sempre o mesmo. Caso os redirects estejam ativos apenas no primeiro ping é que vai haver um reencaminhamento do ping para o router, depois vai haver um redirect do router para o *tux32* e este vai guardar na sua tabela de fowarding que para chegar ao *tux31* pode utilizar o *tux34* como gateway, depois do primeiro ping não existem mais redirects já que o *tux32* envia logo para o *tux34* o ping invés de passar pelo router.

- Como configurar o NAT num *router* comercial?

De forma a configurar o router, foi configurada a interface interna do processo de NAT, segundo os passos indicados no slide 46 do guião fornecido. A partir do GTKTerm, os seguintes comandos foram inseridos:

- *conf t*
- *interface gigabitethernet 0/0*
- *ip address 172.16.31.254 255.255.255.0* (configuração de uma interface do router, atribuindo-lhe um IP Address a uma dada porta, neste caso a 0)
- *no shutdown*
- *ip nat inside* (especificar que o IP dado encontra-se ligado à subrede local)
- *exit*
- *interface gigabitethernet 0/1*
- *ip address 172.16.1.39 255.255.255.0* (configuração de uma interface do router, atribuindo-lhe um IP Address a uma dada porta, neste caso a 1)
- *no shutdown*
- *ip nat outside* (especificar que o IP dado é exterior à subrede local)
- *exit*
- *ip nat pool ovrlid 172.16.1.39 172.16.1.39 prefix 24*
- *ip nat inside source list 1 pool ovrlid overload*
- *access-list 1 permit 172.16.30.0 0.0.0.7*
- *access-list 1 permit 172.16.31.0 0.0.0.7*
- *ip route 0.0.0.0 0.0.0.0 172.16.1.254* (conf. da rota default para 172.16.1.254)
- *ip route 172.16.30.0 255.255.255.0 172.16.31.253* (conf. da rota para a subrede 172.16.30.0 c/gateway 172.16.31.253)
- *end*

- O que faz o NAT?

O NAT (Network Address Translation) é um protocolo que tem como função a associação e transformação de um IP Address noutro IP Address, de forma a “mascarar” o remetente/destinatário de pacotes enviados, podendo ter vários fins como assegurar a privacidade e segurança de máquinas numa rede privada local que estão a comunicar com máquinas “externas” (é implementado com frequência em ambientes de acesso remoto), conservando os seus IPs, mas também pode permitir que essa comunicação seja possível: permite que redes IP privadas que usem endereços não registados se conectem e comuniquem com a Internet ou redes públicas. As máquinas dessa rede, para as máquinas exteriores, são reconhecidas através de um IP único, que representa todos os dispositivos da mesma.

O NAT opera num router, como o que foi configurado e utilizado na experiência 4. Esse router estabelece o ponto de ligação entre a rede local e a Internet/rede pública. No caso de uma máquina da rede local (suponhamos o *tuxy1*) querer enviar um pacote para um endereço numa rede pública (172.16.1.254, como foi feito na experiência), o pacote é enviado para o router e o router modifica o endereço “source” do pacote para o seu endereço exterior (172.16.1.39), “mascarando” assim o verdadeiro remetente do pacote (*tuxy1*) e assegurando a sua privacidade e segurança. O pacote é enviado para 172.16.1.254, que responde com um pacote que tem como destinatário 172.16.1.39. O router, ao receber esse pacote, reenvia-o para *tuxy1*, mudando o destinatário do pacote para o seu endereço, possibilitando então a comunicação entre a rede privada local e a rede pública.

Caso o router não estivesse configurado com o protocolo NAT, a máquina 172.16.1.254, ao receber o pacote, que iria ter como “source” o verdadeiro remetente (o *tuxy1*), não saberia como responder para esse endereço IP.

### 3.5 Experiência 5 - DNS

O objetivo desta experiência foi a conexão das máquinas da rede IP a um servidor DNS, que efetua a tradução de hostnames para endereços IP, e verificar como isso muda a maneira de as máquinas se poderem conectar com a Internet.

Principais comandos: Comandos inseridos no ficheiro *resolv.conf* (detalhados mais à frente)

- Como configurar o serviço DNS num *host*?

O serviço DNS é configurado no ficheiro *resolv.conf*, localizado no diretório */etc/* do anfitrião *tux* em questão. A configuração é feita através de dois comandos, um que representa o nome do servidor DNS, e um com o respetivo endereço IP:

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

- Que pacotes são trocados por DNS, e que informação é transportada?

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.

### 3.6 Experiência 6 - Conexões TCP

Nesta experiência, o objetivo foi a observação do funcionamento e comportamento do protocolo TCP, sendo para isso utilizada a aplicação desenvolvida anteriormente.

Principais comandos: Compilação e execução da aplicação criada anteriormente.

- Quantas conexões TCP são abertas pela aplicação FTP?

São abertas duas conexões TCP pela aplicação, uma quando se entra em contacto com o servidor e através da qual se envia e recebe comandos para preparar a transferência do ficheiro, e outra para fazer a transferência do ficheiro em si.

- Em qual conexão é transportado o controlo de informação?

O controlo de informação é transportado na primeira conexão TCP (na que trata do envio e receção de comandos).

- Quais são as fases de uma conexão TCP?

Numa conexão TCP, primeiro estabelece-se a conexão, depois ocorre a troca de dados, e depois a conexão é encerrada, havendo assim três fases.

- Como funciona o mecanismo *ARQ TCP*? Quais são os campos TCP relevantes? Que informação relevante pode ser observada nos *logs*?

O mecanismo *ARQ TCP* funciona através do método da janela deslizante, que consiste no controlo de erros na transmissão de dados. Para este efeito, são utilizados *acknowledgement numbers*, que indicam o correto recebimento da trama, *window size*, que indica a gama de pacotes recebidos, e *sequence number*, que é o número do pacote a ser enviado.

- Como funciona o mecanismo de congestionamento de controlo TCP? Quais são os campos relevantes? Como é que o *throughput* da conexão de dados evolui ao longo do tempo? Está de acordo com o mecanismo de congestionamento de controlo TCP?

O mecanismo de controlo de congestão do TCP tem como base os ACKs recebidos na transmissão de pacotes. Estes são o *source clock* da transmissão. É utilizada uma nova variável/valor por conexão, denominada *CongestionWindow*, de modo a regular o tamanho da janela deslizante de transmissão de pacotes tendo em conta a congestão da conexão. Este valor é regulado, incrementando se a congestão da rede diminui e decrementando se a congestão da rede aumenta. Isto é normalmente feito incrementando *CongestionWindow* por 1, a cada RTT (“round trip time”). Quando se deteta que um pacote é perdido (normalmente isso é detetado quando ocorrer um timeout, ou quando se recebe 3 ACKs duplicados), o valor de *CongestionWindow* passa para metade. O bitrate da conexão irá ser aproximadamente igual a *CongestionWindow/RTT*.

No início da conexão, pode também haver uma fase de *slow start*, que serve para, de modo a delimitar um *threshold* que é depois utilizado numa fase posterior de *congestion avoidance*.

Foi registado que, quando o primeiro download, no *tuxy1*, começou, a taxa de transmissão aumentou rapidamente, tendo depois chegado a um pico alguns segundos depois. Após o início do segundo download, no *tuxy2*, a taxa de transmissão no *tuxy1* diminuiu rapidamente e a do *tuxy2* aumentou também rapidamente, e passado alguns “altos e baixos”, o throughput estabilizou relativamente num nível mais baixo do que era anteriormente, antes do segundo download ter começado.

Estas mudanças fazem sentido e estão de acordo com o mecanismo de controlo de congestão do TCP, uma vez que quando circulavam na rede menos pacotes (apenas um download a ser feito), o bitrate para a conexão do *tuxy1* é mais alto do que quando os dois downloads estavam a ser feitos ao mesmo tempo, o que aumenta o congestionamento da rede.

É possível observar um gráfico que demonstra o que foi explicado, nos anexos.

- O *throughput* de uma conexão de dados TCP é perturbado pelo aparecimento de uma segunda conexão TCP? Como?

Sim, é. A taxa de transmissão de pacotes da conexão TCP que já estava iniciada diminui, uma vez que à outra conexão foi atribuída uma capacidade para a transmissão de pacotes na mesma, de modo a que a taxa de transferência seja distribuída de igual forma para cada ligação.

### 3.7 Experiência 7 - Implementação de NAT em Linux

Esta experiência tem como objetivo a implementação da funcionalidade NAT no *tuxy4*, e a geração de tráfico do *tuxy1* para a Internet, de modo a observar a tradução de endereços IP e portas por parte do *tuxy4*.

Principais comandos:

- *wget*
- *traceroute*
- Comandos para configurar NAT no *tuxy4*:
  - *iptables -t nat -A POSTROUTING -o eth1 -s 10.10.0.0/24 -j SNAT -to-source 172.16.21.253*  
(pacotes que tenham como source endereços IP na rede 10.10.0.0/24:  
a sua source é modificada para 172.16.21.253)
  - *iptables -t nat -A PREROUTING -i eth1 -d 172.16.21.253 -j DNAT -to-destination 10.10.0.1*  
(pacotes que tenham como destino o endereço 172.16.21.253:  
o destino é modificado para 10.10.0.1)

Na experiência 7 procedeu-se, como já foi dito, à implementação de funcionalidades NAT no *tuxy4*. De seguida, gerou-se tráfico do *tuxy1* para a Internet (passando por *tuxy4*), utilizando diferentes comandos que utilizam diferentes protocolos (*TCP* - *wget*; *UDP* - *traceroute*, *ICMP* - *ping*).

Nos logs gerados em ambas as interfaces do *tuxy4*, é possível observar que a *source* de pacotes enviados do *tuxy1* foi alterada do *tuxy1* (10.10.0.1) para a interface *eth1* de *tuxy4* (172.16.21.253), quando eles saem do *tuxy4*. Nos pacotes de resposta o destino dos mesmos foi alterado do *tuxy4.eth1* para o *tuxy1*, ao chegarem ao *tuxy4*.

## 4 Conclusões

Este segundo trabalho de RCOM teve como objetivo o desenvolvimento de uma aplicação que permitisse o *download* de um ficheiro através de conexões utilizando os protocolos FTP e TCP, e também a configuração, em vários passos, de uma rede IP, de modo a perceber o funcionamento de várias máquinas e dispositivos, como o switch e o router, e também de modo a reconhecer as diversas técnicas, (NAT, DNS, etc), protocolos (ICMP, ARP, etc) e estruturas de dados (forwarding tables, tabelas ARP, etc) utilizadas na comunicação entre essas máquinas.

Somos da opinião que todos os objetivos deste trabalho foram cumpridos, sendo que o nosso conhecimento sobre os temas abordados no mesmo foi desenvolvido e aprofundado.

## Referências

Este trabalho foi desenvolvido com recurso à consulta dos slides teóricos da unidade curricular e ao guia fornecido.

## Anexos

Código da aplicação:

```
#include <stdio.h>
#include <string.h>
#include "macros.h"
#include "functions.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s %s\n", argv[0], "ftp://[<user>:<password>@]<host>/<url-path>");
        return -1;
    }
    // parse command line arguments
    struct arguments args;
    if (parseArguments(&args, argv[1]) != 0) {
        return -1;
    }
    printf("User: %s\n", args.user);
    printf("Password: %s\n", args.password);
    printf("Host name: %s\n", args.host_name);
    printf("File path: %s\n", args.file_path);
    printf("File name: %s\n", args.file_name);

    struct ftp ftp;
    char command[MAX_LENGTH];           // buffer to send commands
    char responseBuffer[MAX_LENGTH];   // buffer to read commands

    // get IP Address
    char ipAddress[MAX_LENGTH];
    if (getIPAddress(ipAddress, args.host_name) < 0) {
        return -1;
    }
    // create and connect socket to server
    if ((ftp.control_socket_fd = createAndConnectSocket(ipAddress, FTP_PORT_NUMBER)) < 0) {
        printf("Error creating new socket\n");
        return -1;
    }
    // receive confirmation from server
    receiveFromControlSocket(&ftp, responseBuffer, MAX_LENGTH);
    // checking confirmation from server
    if (responseBuffer[0] == '2') {
        printf("Expecting username...\n\n");
    }
    else
    {
        printf("Error in connection...\n\n");
        return -1;
    }
    // do login in server
    if (login(&ftp, args.user, args.password)<0) {
        printf("Login failed...\n\n");
        return -1;
    }
}
```

Figura 1: Código da função *main* - pt.1

```

// change working directory in server
if (strlen(args.file_path) > 0) {
    if (changeWorkingDirectory(&ftp, args.file_path) < 0)
    {
        printf("Error changing directory\n");
        return -1;
    }
}
// sends pasv command to get ip address and port for data socket
if (getServerPortForFile(&ftp) < 0){
    printf("Error getting server Port for file\n");
    return -1;
}
// sends retr command to begin file transfer through data socket
if(retr(&ftp, args.file_name) < 0){
    printf("Error sending comand retr\n");
    return -1;
}
// download of file
if(downloadFile(&ftp, args.file_name) < 0){
    printf("Error downloading file\n");
    return -1;
}
// disconnects from server
if(disconnectFromSocket(&ftp) < 0){
    printf("Error disconnecting from server\n");
    return -1;
}
return 0;
}

```

Figura 2: Código da função *main* - pt.2

```

#pragma once

#define MAX_LENGTH 200
#define FTP_PORT_NUMBER 21

```

Figura 3: Macros utilizadas

```

/**
 * Function that, having the host name, retrieves the IP address
 *
 * @param idAddress Variable that is going to point to the IP Address
 * @param hostName The host's name
 * @return int 0 if success; -1 otherwise
 */
int getIPAddress(char *ipAddress, char *hostName) {
    struct hostent *h;
    if ((h = gethostbyname(hostName)) == NULL) {
        perror("gethostbyname");
        return -1;
    }
    strcpy(ipAddress, inet_ntoa(((struct in_addr *)h->h_addr)));
    return 0;
}

```

Figura 4: Funções auxiliares desenvolvidas: *getIPAddress()*

```

/**
 * Function that parses the command line arguments, retrieving a struct with all the individual fields
 *
 * @param args Pointer to the structure that is going to have the individual fields
 * @param commandLineArg Argument from the command line, that is going to be parsed
 * @return int 0 if success; -1 otherwise
 */
int parseArguments(struct arguments *args, char *commandLineArg) {

    printf("Parsing command line arguments...\n");

    // verifying FTP protocol
    char *token = strtok(commandLineArg, ":");
    if ((token == NULL) || (strcmp(token, "ftp") != 0)) {
        printf("-> Error in the protocol name (should be 'ftp')\n");
        return -1;
    }

    token = strtok(NULL, "\0");
    char rest_of_string[MAX_LENGTH];
    strcpy(rest_of_string, token);

    // parsing user name
    char aux[MAX_LENGTH];
    strcpy(aux, rest_of_string);
    token = strtok(aux, ":");

    if (token == NULL || (strlen(token) < 3) || (token[0] != '/') || (token[1] != '/')) {
        printf("-> Error parsing the user name\n");
        return -1;
    }
    else if (strcmp(token, rest_of_string) == 0) {
        memset(args->user, 0, sizeof(args->user));
        strcpy(args->user, "anonymous");
        memset(args->password, 0, sizeof(args->password));
        strcpy(args->password, "");

        char aux2[MAX_LENGTH];
        strcpy(aux2, &rest_of_string[2]);
        strcpy(rest_of_string, aux2);
    }
    else {
        memset(args->user, 0, sizeof(args->user));
        strcpy(args->user, &token[2]);
        // parsing password
        token = strtok(NULL, "@");
        if (token == NULL || (strlen(token) == 0)) {
            printf("-> Error parsing the password\n");
            return -1;
        }
        memset(args->password, 0, sizeof(args->password));
        strcpy(args->password, token);

        token = strtok(NULL, "\0");
        strcpy(rest_of_string, token);
    }
}

```

Figura 5: Funções auxiliares desenvolvidas: *parseArguments()* - pt.1

```

    // parsing hostname
    token = strtok(rest_of_string, "/");
    if (token == NULL) {
        printf("-> Error parsing the hostname\n");
        return -1;
    }
    memset(args->host_name, 0, sizeof(args->host_name));
    strcpy(args->host_name, token);

    // parsing path and name
    token = strtok(NULL, "\0");
    if (token == NULL) {
        printf("-> Error parsing the file path\n");
        return -1;
    }
    char* last = strrchr(token, '/');
    if (last != NULL) {
        memset(args->file_path, 0, sizeof(args->file_path));
        strncpy(args->file_path, token, last - token);
        memset(args->file_name, 0, sizeof(args->file_name));
        strcpy(args->file_name, last + 1);
    }
    else {
        memset(args->file_path, 0, sizeof(args->file_path));
        strcpy(args->file_path, "");
        memset(args->file_name, 0, sizeof(args->file_name));
        strcpy(args->file_name, token);
    }

    printf("Parsed command line arguments.\n\n");
    return 0;
}

```

Figura 6: Funções auxiliares desenvolvidas: *parseArguments()* - pt.2

```

/**
 * Function that creates a new TCP socket, and connects it to the address and port specified
 *
 * @param address The IP address of the server
 * @param port The number of the port to be used
 * @return int Socket descriptor if success; -1 otherwise
 */
int createAndConnectSocket(char *address, int port) {
    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(address);
    server_addr.sin_port = htons(port);
    // open a TCP socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket()");
        return -1;
    }
    // connect to the server
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("connect()");
        return -1;
    }
    return sockfd;
}

```

Figura 7: Funções auxiliares desenvolvidas: *createAndConnectSocket()*

```

/**
 * Function that allows a command to be sent through a socket
 *
 * @param sockfd The socket descriptor
 * @param command The command to be sent
 * @param commandLength The length of the command
 * @return 0 if success; -1 otherwise
 */
int sendToControlSocket(struct ftp *ftp, char *cmdHeader, char *cmdBody) {
    printf("Sending to control Socket > %s %s\n", cmdHeader, cmdBody);
    int bytes = write(ftp->control_socket_fd, cmdHeader, strlen(cmdHeader));
    if (bytes != strlen(cmdHeader))
        return -1;
    bytes = write(ftp->control_socket_fd, " ", 1);
    if (bytes != 1)
        return -1;
    bytes = write(ftp->control_socket_fd, cmdBody, strlen(cmdBody));
    if (bytes != strlen(cmdBody))
        return -1;
    bytes = write(ftp->control_socket_fd, "\n", 1);
    if (bytes != 1)
        return -1;
    return 0;
}

```

Figura 8: Funções auxiliares desenvolvidas: *sendToControlSocket()*

```

/**
 * Function that allows the reading of a command through a socket
 *
 * @param ftp Struct with the socket descriptors
 * @param string Buffer that is going to store the 3 digit number code received from the server
 * @param size Number of digits to be received
 * @return int 0 if success; -1 otherwise
 */
int receiveFromControlSocket(struct ftp *ftp, char *string, size_t size) {
    printf("Receiving from control Socket \n");
    FILE *fp = fopen(ftp->control_socket_fd, "r");
    do {
        memset(string, 0, size);
        string = fgets(string, size, fp);
        printf("%s", string);
    } while (!(('1' <= string[0] && string[0] <= '5') || string[3] != ' '));
    return 0;
}

```

Figura 9: Funções auxiliares desenvolvidas: *receiveFromControlSocket()*

```

/**
 * Function that sends a command to the control socket and interprets the response received
 *
 * @param ftp Struct with the socket descriptors
 * @param cmdHeader Header of the command to be sent
 * @param cmdBody Body of the command to be sent
 * @param response Buffer that is going to store the number code received from the server
 * @param responseLength Number of digits to be received on the response
 * @param readingFile Indicates if the file is about to be read from the data socket
 * @return int Positive (depending on response) if success; -1 otherwise
 */
int sendCommandInterpretResponse(struct ftp *ftp, char *cmdHeader, char *cmdBody, char *response,
size_t responseLength, bool readingFile) {
    if (sendToControlSocket(ftp, cmdHeader, cmdBody) < 0) {
        printf("Error Sending Command %s %s\n", cmdHeader, cmdBody);
        return -1;
    }
    int code;
    while (1) {
        receiveFromControlSocket(ftp, response, responseLength);
        code = response[0] - '0';
        switch (code) {
            case 1:
                // expecting another reply
                if (readingFile) return 2;
                else break;
            case 2:
                // request action success
                return 2;
            case 3:
                // needs additional information
                return 3;
            case 4:
                // try again
                if (sendToControlSocket(ftp, cmdHeader, cmdBody) < 0) {
                    printf("Error Sending Command %s %s\n", cmdHeader, cmdBody);
                    return -1;
                }
                break;
            case 5:
                // error in sending command, closing control socket , exiting application
                printf("> Command wasn't accepted... \n");
                close(ftp->control_socket_fd);
                exit(-1);
                break;
            default:
                break;
        }
    }
}

```

Figura 10: Funções auxiliares desenvolvidas: *sendCommandInterpretResponse()*

```

/**
 * Function that obtains a server port for the transfer of a file.
 *
 * @param ftp Struct with the socket descriptors
 * @return int 0 if success; -1 otherwise
 */
int getServerPortForFile(struct ftp *ftp) {
    char firstByte[4];
    char secondByte[4];
    memset(firstByte, 0, 4);
    memset(secondByte, 0, 4);
    char response[MAX_LENGTH];
    int rtr = sendCommandInterpretResponse(ftp, "pasv", "", response, MAX_LENGTH, false);
    int ipPart1, ipPart2, ipPart3, ipPart4;
    int port1, port2;
    if (rtr == 2) {
        // starting to process information
        if ((sscanf(response, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)",
                    &ipPart1, &ipPart2, &ipPart3, &ipPart4, &port1, &port2)) < 0) {
            printf("ERROR: Cannot process information to calculating port.\n");
            return -1;
        }
    } else {
        printf("Error receiving pasv command response from server...\n\n");
        return -1;
    }
    char ip[MAX_LENGTH];
    sprintf(ip, "%d.%d.%d.%d", ipPart1, ipPart2, ipPart3, ipPart4);
    int port = port1 * 256 + port2;
    printf("Port number %d\n\n", port);
    if ((ftp->data_socket_fd = createAndConnectSocket(ip, port)) < 0) {
        printf("Error creating new socket\n");
        return -1;
    }
    return 0;
}

```

Figura 11: Funções auxiliares desenvolvidas: *getServerPortForFile()*

```

/**
 * Function that sends the "retr" command to the control socket,
 * so the file starts being transmitted in the file socket
 *
 * @param ftp Struct containing the socket descriptors
 * @param fileName The name of the file to be transferred
 * @return int 0 if successful; -1 otherwise
 */
int retr(struct ftp* ftp, char* fileName){
    char response[MAX_LENGTH];
    if(sendCommandInterpretResponse(ftp, "RETR", fileName, response, MAX_LENGTH, true) < 0){
        printf("Error sending Command Retr\n\n");
        return -1;
    }
    return 0;
}

```

Figura 12: Funções auxiliares desenvolvidas: *retr()*

```

/**
 * Function that sends the login information to the socket for authentication
 *
 * @param ftp Struct that contains the socket descriptors
 * @param username User name to be sent to the socket
 * @param password Password to be sent to the socket
 * @return int 0 if success; -1 otherwise
 */
int login(struct ftp *ftp, char *username, char *password) {
    printf("Sending Username...\n\n");
    char response[MAX_LENGTH];
    int rtr = sendCommandInterpretResponse(ftp, "user", username, response, MAX_LENGTH, false);
    if (rtr == 3) {
        printf("Sent Username...\n\n");
    }
    else {
        printf("Error sending Username...\n\n");
        return -1;
    }
    printf("Sending Password...\n\n");
    rtr = sendCommandInterpretResponse(ftp, "pass", password, response, MAX_LENGTH, false);
    if (rtr == 2) {
        printf("Sent Password...\n\n");
    }
    else {
        printf("Error sending Password...\n\n");
        return -1;
    }
    return 0;
}

```

Figura 13: Funções auxiliares desenvolvidas: *login()*

```

/**
 * Function to change the working directory of the FTP, using the CWD command
 *
 * @param ftp Struct containing the socket descriptors
 * @param path Path of the new working directory to be changed to
 * @return int 0 if successful; -1 otherwise
 */
int changeWorkingDirectory(struct ftp* ftp, char* path) {
    char response[MAX_LENGTH];
    if(sendCommandInterpretResponse(ftp, "CWD", path, response, MAX_LENGTH, false) < 0){
        printf("Error sending Comand CWD\n\n");
        return -1;
    }
    return 0;
}

```

Figura 14: Funções auxiliares desenvolvidas: *changeWorkingDirectory()*

```

/**
 * Function that downloads a file sent from the server through a socket, and saves it in a local file
 *
 * @param ftp Struct containing the socket descriptors
 * @param fileName The name of the file to be transferred
 * @return int 0 if successful; -1 otherwise
 */
int downloadFile(struct ftp* ftp, char * fileName){
    FILE *fp = openFile(fileName, "w");
    if (fp == NULL){
        printf("Error opening or creating file\n");
        return -1;
    }
    char buf[1024];
    int bytes;
    printf("Starting to download file with name %s\n", fileName);
    while((bytes = read(ftp->data_socket_fd, buf, sizeof(buf)))){
        if(bytes < 0){
            printf("Error reading from data socket\n");
            return -1;
        }
        if((bytes = fwrite(buf, bytes, 1, fp)) < 0){
            printf("Error writing data to file\n");
            return -1;
        }
    }
    printf("Finished dowloading File\n");
    if(closeFile(fp) < 0){
        printf("Error closing file\n");
        return -1;
    }
    close(ftp->data_socket_fd);
    char response[MAX_LENGTH];
    receiveFromControlSocket(ftp, response, MAX_LENGTH);
    if (response[0] != '2')
        return -1;
    return 0;
}

```

Figura 15: Funções auxiliares desenvolvidas: *downloadFile()*

```

/**
 * Function that ends the connection with the control socket
 *
 * @param ftp Struct containing the socket descriptors
 * @return int 0 if successful; -1 if error
 */
int disconnectFromSocket(struct ftp* ftp) {
    char response[MAX_LENGTH];
    if(sendCommandInterpretResponse(ftp, "QUIT", "", response, MAX_LENGTH, false) != 2){
        printf("Error sending Command Quit\n\n");
        return -1;
    }
    return 0;
}

```

Figura 16: Funções auxiliares desenvolvidas: *disconnectFromSocket()*

## Experiências laboratoriais

Experiência 1:

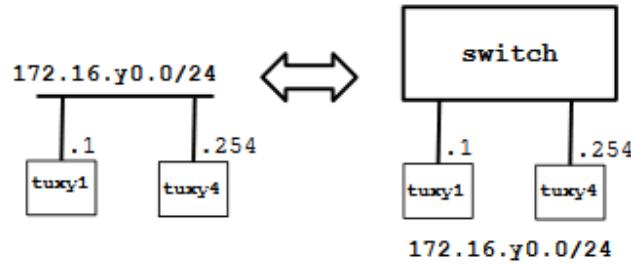


Figura 17: Exp.1: Topologia de rede

30 39.352980467 G-ProCom_8b:e4:4d	Broadcast	ARP	42 Who has 172.16.30.254? Tell 172.16.30.1
31 39.353184564 HewlettP_5a:7d:74	G-ProCom_8b:e4:4d	ARP	60 172.16.30.254 is at 00:21:5a:5a:7d:74
32 39.353196244 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xicfb, seq=1/256, ttl=64 (reply in 33)
33 39.353447759 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xicfb, seq=1/256, ttl=64 (request in 32)
34 39.579814714 172.16.30.1	172.16.2.1	DNS	81 Standard query 0x3dc0 AAAA tux31.netlab.fe.up.pt
35 39.581112697 172.16.30.1	172.16.2.1	DNS	81 Standard query 0xb2f5 AAAA tux31.netlab.fe.up.pt
36 40.352465282 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xicfb, seq=2/512, ttl=64 (reply in 37)
37 40.352621268 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xicfb, seq=2/512, ttl=64 (request in 36)
38 41.351229446 Cisco_3a:fa:86	Spanning-tree-(for...)	STP	60 Cont. Root = 32768/0/00:24:50:92:b9:86 Cost = 4 Port = 0x8006
39 41.352453767 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xicfb, seq=3/768, ttl=64 (reply in 40)
40 41.352705001 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xicfb, seq=3/768, ttl=64 (request in 39)
41 42.352499685 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xicfb, seq=4/1024, ttl=64 (reply in 42)
42 42.352666274 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xicfb, seq=4/1024, ttl=64 (request in 41)
43 43.042318291 Cisco_3a:fa:86	Cisco_3a:fa:86	LOOP	60 Reply
44 43.352450138 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xicfb, seq=5/1280, ttl=64 (reply in 45)
45 43.352789003 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xicfb, seq=5/1280, ttl=64 (request in 44)
46 43.380966614 Cisco_3a:fa:86	Spanning-tree-(for...)	STP	60 Cont. Root = 32768/0/00:24:50:92:b9:86 Cost = 4 Port = 0x8006
47 43.536438037 Cisco_3a:fa:86	CDP/VT/P/DTP/PAgP/UD...	CDP	453 Device ID: tux-sw3 Port ID: FastEthernet0/4
48 44.352461137 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xicfb, seq=6/1536, ttl=64 (reply in 49)
49 44.352614757 172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xicfb, seq=6/1536, ttl=64 (request in 48)
50 44.570354610 HewlettP_5a:7d:74	G-ProCom_8b:e4:4d	ARP	60 Who has 172.16.30.1? Tell 172.16.30.254
51 44.570378159 G-ProCom_8b:e4:4d	HewlettP_5a:7d:74	ARP	42 172.16.30.1 is at 00:0f:fe:8b:e4:d
52 44.584934037 172.16.30.1	193.136.28.10	DNS	81 Standard query 0x3dc0 AAAA tux31.netlab.fe.up.pt
53 44.586193249 172.16.30.1	193.136.28.10	DNS	81 Standard query 0xb2f5 AAAA tux31.netlab.fe.up.pt
54 45.352464337 172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xicfb, seq=7/1792, ttl=64 (reply in 55)

Figura 18: Exp.1: Mensagens entre o tuxy1 e o tuxy4 - ARP e ICMP

Experiência 2:

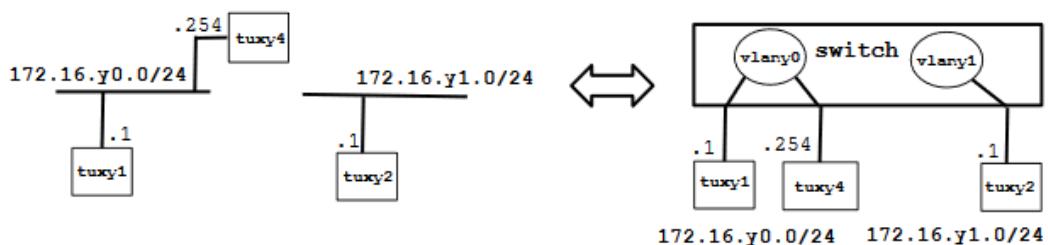


Figura 19: Exp.2: Topologia de rede

19	13.004987227	HewlettP_61:30:63	HewlettP_61:24:92	ARP	42 Who has 172.16.30.254? Tell 172.16.30.1
20	13.005106376	HewlettP_61:24:92	HewlettP_61:30:63	ARP	60 172.16.30.254 is at 00:21:5a:61:24:92
21	13.277192582	172.16.30.254	193.136.28.10	DNS	81 Standard query 0x0385 A 2.debian.pool.ntp.org
22	13.277262290	172.16.30.254	193.136.28.10	DNS	81 Standard query 0xd38e AAAA 2.debian.pool.ntp.org
23	13.39796257	Cisco_3a:fa:84	Cisco_3a:fa:84	LOOP	88 Reply
24	14.034565436	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
25	14.491610994	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x080d, seq=1/256, ttl=64 (reply in 26)
26	14.491177338	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x080d, seq=1/256, ttl=64 (request in 25)
27	15.501026632	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x080d, seq=2/512, ttl=64 (reply in 28)
28	15.501161007	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x080d, seq=2/512, ttl=64 (request in 27)
29	16.039433926	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
30	16.525027031	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x080d, seq=3/768, ttl=64 (reply in 31)
31	16.525161475	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x080d, seq=3/768, ttl=64 (request in 30)
32	17.105089342	Cisco_3a:fa:84	CDP/VTP/DTP/PoP/UD..	CDP	453 Device ID: tux-sw3 Port ID: FastEthernet0/2
33	17.549026521	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x080d, seq=4/1924, ttl=64 (reply in 34)
34	17.549155699	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x080d, seq=4/1924, ttl=64 (request in 33)
35	17.977992942	172.16.30.1	193.136.28.10	DNS	81 Standard query 0xb926 A 3.debian.pool.ntp.org
36	17.978082022	172.16.30.1	193.136.28.10	DNS	81 Standard query 0xd4d31 AAAA 3.debian.pool.ntp.org
37	18.044458308	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
38	18.282480196	172.16.30.254	172.16.1.1	DNS	97 Standard query 0xba35 A 2.debian.pool.ntp.org.netlab.fe.up.pt
39	18.282480198	172.16.30.254	172.16.1.1	DNS	97 Standard query 0xe0e45 AAAA 2.debian.pool.ntp.org.netlab.fe.up.pt
40	18.573024894	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x080d, seq=5/1280, ttl=64 (reply in 41)
41	18.573153262	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x080d, seq=5/1280, ttl=64 (request in 40)

Figura 20: Exp.2: Ping de *tuxy1* para *tuxy4*

116	172.485452100	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=1/256, ttl=64 (no response from 172.16.30.1)
117	173.495451558	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=2/512, ttl=64 (no response from 172.16.30.1)
118	174.454976259	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
119	174.519451398	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=3/768, ttl=64 (no response from 172.16.30.1)
120	175.543453121	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=4/1924, ttl=64 (no response from 172.16.30.1)
121	176.459002438	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
122	176.567447094	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=5/1280, ttl=64 (no response from 172.16.30.1)
123	177.105646819	Cisco_3a:fa:84	LOOP	60 Reply	98 Echo (ping) request id=0x091c, seq=6/1536, ttl=64 (no response from 172.16.30.1)
124	177.591457130	172.16.30.1	172.16.30.255	ICMP	60 Echo (ping) request id=0x091c, seq=7/1792, ttl=64 (no response from 172.16.30.1)
125	178.463985842	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
126	178.615449078	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=8/2048, ttl=64 (no response from 172.16.30.1)
127	179.639446334	172.16.30.1	172.16.30.255	ICMP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
128	180.468793260	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	98 Echo (ping) request id=0x091c, seq=9/2304, ttl=64 (no response from 172.16.30.1)
129	180.663443869	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=10/2560, ttl=64 (no response from 172.16.30.1)
130	180.687449296	172.16.30.1	172.16.30.255	ICMP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
131	182.473765769	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	98 Echo (ping) request id=0x091c, seq=11/2816, ttl=64 (no response from 172.16.30.1)
132	182.711454024	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=12/3072, ttl=64 (no response from 172.16.30.1)
133	183.735453934	172.16.30.1	172.16.30.255	ICMP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
134	184.759445740	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	98 Echo (ping) request id=0x091c, seq=13/3228, ttl=64 (no response from 172.16.30.1)
135	184.176.783447746	172.16.30.1	172.16.30.255	ICMP	98 Echo (ping) request id=0x091c, seq=14/3584, ttl=64 (no response from 172.16.30.1)
136	186.483558982	Cisco_3a:fa:84	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/30/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004

Figura 21: Exp.2: Ping broadcast a partir de *tuxy1*

Nota: apesar de o Wireshark não mostrar os pacotes ICMP reply do *tuxy4*, no terminal era possível verificar a sua resposta ao ping broadcast.

20	52.095312000	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=1/256, ttl=64 (no response from 172.16.31.1)
24	33.723506221	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=2/512, ttl=64 (no response from 172.16.31.1)
25	34.534929971	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
26	34.747491321	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=3/768, ttl=64 (no response from 172.16.31.1)
27	35.771503119	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=4/1924, ttl=64 (no response from 172.16.31.1)
28	36.539849466	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
29	36.795487851	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=5/1280, ttl=64 (no response from 172.16.31.1)
30	37.891498560	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=6/1536, ttl=64 (no response from 172.16.31.1)
31	38.5544764372	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
32	38.884349102	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=7/1792, ttl=64 (no response from 172.16.31.1)
33	39.886743544	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=8/2048, ttl=64 (no response from 172.16.31.1)
34	40.025496670	Cisco_3a:fa:86	LOOP	60 Reply	98 Echo (ping) request id=0x0c5c, seq=9/2304, ttl=64 (no response from 172.16.31.1)
35	40.549642322	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
36	40.891500791	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=10/2560, ttl=64 (no response from 172.16.31.1)
37	41.915492195	172.16.31.1	172.16.31.255	ICMP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
38	42.554486049	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	98 Echo (ping) request id=0x0c5c, seq=11/2816, ttl=64 (no response from 172.16.31.1)
39	42.9393490374	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=12/3072, ttl=64 (no response from 172.16.31.1)
40	43.963485480	172.16.31.1	172.16.31.255	ICMP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
41	44.559364555	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	98 Echo (ping) request id=0x0c5c, seq=13/3328, ttl=64 (no response from 172.16.31.1)
42	44.987474650	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=14/3584, ttl=64 (no response from 172.16.31.1)
43	46.011496785	172.16.31.1	172.16.31.255	ICMP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
44	46.564393365	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	98 Echo (ping) request id=0x0c5c, seq=15/3840, ttl=64 (no response from 172.16.31.1)
45	47.639474730	172.16.31.1	172.16.31.255	ICMP	98 Echo (ping) request id=0x0c5c, seq=16/4096, ttl=64 (no response from 172.16.31.1)
46	48.659473300	172.16.31.1	172.16.31.255	ICMP	60 Conf. Root = 32768/31/fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
47	48.574198616	Cisco_3a:fa:86	Spanning-tree-(for-..)	STP	98 Echo (ping) request id=0x0c5c, seq=17/4352, ttl=64 (no response from 172.16.31.1)
48	49.083472605	172.16.31.1	172.16.31.255	ICMP	60 Reply

Figura 22: Exp.2: Ping broadcast a partir de *tuxy2*

### Experiência 3:

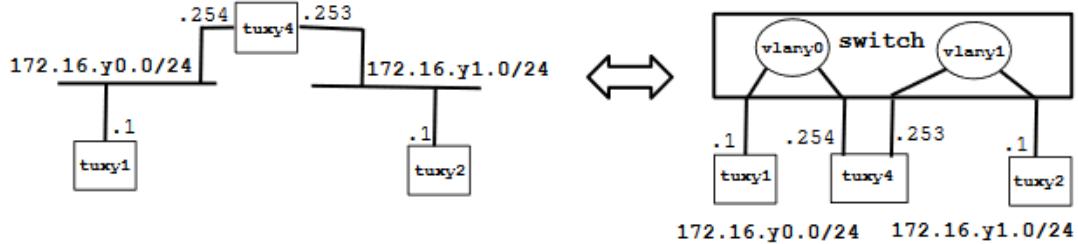


Figura 23: Exp.3: Topologia de rede

4 5.744700284	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=1/256, ttl=64 (reply in 5)
5 5.744984284	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=1/256, ttl=63 (request in 4)
6 6.014390602	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
7 6.015852447	Cisco_3a:fa:83	Cisco_3a:fa:83	LOOP	60 Reply
8 6.743703986	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=2/512, ttl=64 (reply in 9)
9 6.744155244	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=2/512, ttl=63 (request in 8)
10 7.742705953	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=3/768, ttl=64 (reply in 11)
11 7.743175763	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=3/768, ttl=63 (request in 10)
12 8.014138607	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
13 8.741709264	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=4/1024, ttl=64 (reply in 14)
14 8.742160672	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=4/1024, ttl=63 (request in 13)
15 9.741301291	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=5/1280, ttl=64 (reply in 16)
16 9.741578800	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=5/1280, ttl=63 (request in 15)
17 10.018973516	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
18 10.741288900	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=6/1536, ttl=64 (reply in 19)
19 10.741741910	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=6/1536, ttl=63 (request in 18)
20 10.745244568	G-ProCom_8b:e4:4d	HewlettP_5a:7d:74	ARP	42 Who has 172.16.30.1? Tell 172.16.30.1
21 10.74597691	HewlettP_5a:7d:74	G-ProCom_8b:e4:4d	ARP	60 172.16.30.254 is at 00:21:5a:5a:7d:74
22 10.890137928	G-ProCom_8b:e4:4d	HewlettP_5a:7d:74	ARP	60 Who has 172.16.30.1? Tell 172.16.30.254
23 10.890157297	G-ProCom_8b:e4:4d	HewlettP_5a:7d:74	ARP	42 172.16.30.1 is at 00:0f:fe:b8:e4:4d
24 10.890137929	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=7/1792, ttl=64 (reply in 25)
25 11.741757627	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=7/1792, ttl=63 (request in 24)
26 12.028907089	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
27 12.741294522	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1538, seq=8/2048, ttl=64 (reply in 28)
28 12.741764161	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1538, seq=8/2048, ttl=63 (request in 27)

Figura 24: Exp.3: Ping de tuxy1 para tuxy2 - pt.1

55 35.512239745	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
66 36.232837829	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x154b, seq=1/256, ttl=64 (reply in 67)
67 36.233957343	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x154b, seq=1/256, ttl=64 (request in 66)
68 37.231843565	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x154b, seq=2/512, ttl=64 (reply in 69)
69 37.232962648	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x154b, seq=2/512, ttl=64 (request in 68)
70 38.111618452	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
71 38.230844871	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x154b, seq=3/768, ttl=64 (reply in 72)
72 38.231847392	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x154b, seq=3/768, ttl=64 (request in 71)
73 39.229846898	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x154b, seq=4/1024, ttl=64 (reply in 74)
74 39.230095069	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x154b, seq=4/1024, ttl=64 (request in 73)
75 40.116404854	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
76 40.229298349	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0x154b, seq=5/1280, ttl=64 (reply in 77)
77 40.229502069	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0x154b, seq=5/1280, ttl=64 (request in 76)
78 41.498654979	Cisco_3a:fa:83	CDP/VTTP/DTP/PagP/UD..	CDP	435 Device ID: tux-sw3 Port ID: FastEtherne0/1
79 42.12.113032004	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
80 43.241257213	G-ProCom_8b:e4:4d	HewlettP_5a:7d:74	ARP	42 Who has 172.16.30.254? Tell 172.16.30.1
81 43.241607484	HewlettP_5a:7d:74	G-ProCom_8b:e4:4d	ARP	60 172.16.30.254 is at 00:21:5a:5a:7d:74
82 44.131110052	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003

Figura 25: Exp.3: Ping de tuxy1 para tuxy4.eth0

97 58.890192415	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1559, seq=1/256, ttl=64 (request in 96)
98 59.889294186	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x1559, seq=2/512, ttl=64 (reply in 99)
99 59.889543936	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1559, seq=2/512, ttl=64 (request in 98)
100 60.164742481	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
101 60.552656581	HewlettP_5a:7d:74	G-ProCom_8b:e4:4d	ARP	60 Who has 172.16.30.1? Tell 172.16.30.254
102 60.552681960	G-Procom_8b:e4:4d	HewlettP_5a:7d:74	ARP	42 172.16.30.1 is at 00:0f:fe:b8:e4:4d
103 60.889311372	172.16.30.1	172.16.31.253	ICMP	98 Echo (ping) request id=0x1559, seq=3/768, ttl=64 (reply in 104)
104 60.889665989	172.16.31.253	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1559, seq=3/768, ttl=64 (request in 103)
105 62.174450606	Cisco_3a:fa:83	Spanning-tree-(for-~ STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003

Figura 26: Exp.3: Ping de tuxy1 para tuxy4.eth1

239 381.686437024 G-ProCom_8b:e4:4d	Broadcast	ARP	60 Who has 172.16.30.254? Tell 172.16.30.1
240 381.686465240 HewlettP_5a:7d:74	G-ProCom_8b:e4:4d	ARP	42 172.16.30.254 is at 00:21:5a:5a:7d:74
241 381.686798941 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=1/256, ttl=64 (reply in 242)
242 381.687056236 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=1/256, ttl=63 (request in 243)
243 382.687579360 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=2/512, ttl=64 (reply in 244)
244 382.6877707947 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=2/512, ttl=63 (request in 243)
245 382.686615791 Cisco_3a:fa:86	Spanning-tree-(for-) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:86 Cost = 0 Port = 0x8006
246 383.687302750 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=3/768, ttl=64 (reply in 247)
247 383.687442014 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=3/768, ttl=63 (request in 246)
248 384.687335039 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=4/1024, ttl=64 (reply in 249)
249 384.687590426 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=4/1024, ttl=63 (request in 24...
250 384.970973998 Cisco_3a:fa:86	Spanning-tree-(for-) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
251 385.687375368 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=5/1280, ttl=64 (reply in 252)
252 385.687515112 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=5/1280, ttl=63 (request in 25...
253 386.687399128 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=6/1536, ttl=64 (reply in 254)
254 386.687537274 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=6/1536, ttl=63 (request in 25...
255 386.827277148 HewlettP_5a:7d:74	G-ProCom_8b:e4:4d	ARP	42 Who has 172.16.30.17 Tell 172.16.30.254
256 386.827559795 G-ProCom_8b:e4:4d	HewlettP_5a:7d:74	ARP	60 172.16.30.1 is at 00:fe:fe:8b:e4:4d
257 386.982844574 Cisco_3a:fa:86	Spanning-tree-(for-) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8006
258 387.687427855 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=7/1792, ttl=64 (reply in 259)
259 387.687564464 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=7/1792, ttl=63 (request in 25...
260 388.687458957 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=8/2048, ttl=64 (reply in 261)
261 388.687596893 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=8/2048, ttl=63 (request in 26...

Figura 27: Exp.3: Ping de *tuxy1* para *tuxy2* - pt.2

240 382.361328229 3Com_21:83:0e	Broadcast	ARP	42 Who has 172.16.31.1? Tell 172.16.31.253
241 382.361451149 HewlettP_61:30:63	3Com_21:83:0e	ARP	60 172.16.31.1 is at 00:21:5a:61:30:63
242 382.361458263 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=1/256, ttl=63 (reply in 243)
243 382.361563883 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=1/256, ttl=64 (request in 242)
244 382.944466661 Cisco_3a:fa:85	Spanning-tree-(for-) STP		60 Conf. Root = 32768/31:fc:fb:fb:3a:fa:86 Cost = 0 Port = 0x8005
245 383.362192720 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=2/512, ttl=63 (reply in 246)
246 383.362213209 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=2/512, ttl=64 (request in 245)
247 384.361836787 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=3/768, ttl=63 (reply in 248)
248 384.361946994 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=3/768, ttl=64 (request in 247)
249 384.949164938 Cisco_3a:fa:85	Spanning-tree-(for-) STP		60 Conf. Root = 32768/31:fc:fb:fb:3a:fa:89 Cost = 0 Port = 0x8005
250 385.361867679 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=4/1024, ttl=63 (reply in 251)
251 385.362099661 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=4/1024, ttl=64 (request in 25...
252 386.361908341 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=5/1280, ttl=63 (reply in 253)
253 386.362020793 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=5/1280, ttl=64 (request in 25...
254 386.954041361 Cisco_3a:fa:85	Spanning-tree-(for-) STP		60 Conf. Root = 32768/31:fc:fb:3a:fa:80 Cost = 0 Port = 0x8005
255 387.361931768 172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0x1810, seq=6/1536, ttl=63 (reply in 256)
256 387.362042746 172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0x1810, seq=6/1536, ttl=64 (request in 25...

Figura 28: Exp.3: Ping de *tuxy1* para *tuxy2* - pt.3

#### Experiênci 4:

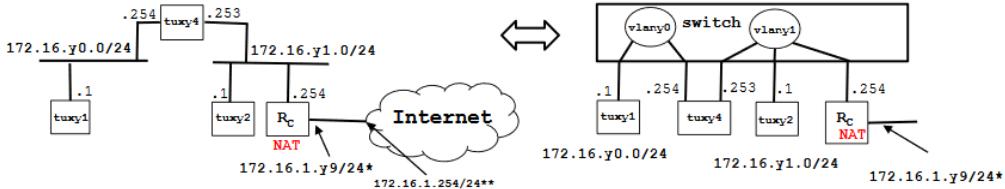


Figura 29: Exp.4: Topologia de rede

```

root@tux32:~# traceroute -4 172.16.30.1
traceroute to 172.16.30.1 (172.16.30.1), 30 hops max, 60 byte packets
 1  172.16.31.254 (172.16.31.254)  0.483 ms  0.498 ms  0.527 ms
 2  172.16.31.253 (172.16.31.253)  0.634 ms  0.346 ms  0.340 ms
 3  172.16.30.1 (172.16.30.1)  0.710 ms  0.692 ms  0.670 ms
root@tux32:~# route add -net 172.16.30.0/24 gw 172.16.31.253
root@tux32:~# traceroute -4 172.16.30.1
traceroute to 172.16.30.1 (172.16.30.1), 30 hops max, 60 byte packets
 1  172.16.31.253 (172.16.31.253)  0.187 ms  0.168 ms  1.143 ms
 2  172.16.30.1 (172.16.30.1)  1.128 ms  2.098 ms  3.070 ms
root@tux32:~#

```

Figura 30: Exp.4: Comando *traceroute* de *tuxy2* para *tuxy1*, antes e depois da configuração da rota de *tuxy2* para a *vlan y0*, através de *tuxy4.eth1*

#	Processo	Origem	Destino	Tipo	Comentário
4	5.50630327	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2, seq=1/256, ttl=64 (reply in 5)
5	5.506501937	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=1/256, ttl=64 (request in 4)
6	6.014729023	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
7	6.505309684	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2, seq=2/512, ttl=64 (reply in 8)
8	6.505560169	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=2/512, ttl=64 (request in 7)
9	7.504436100	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2, seq=3/768, ttl=64 (reply in 10)
10	7.5044645937	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=3/768, ttl=64 (request in 9)
11	7.868693146	Cisco_3a:fa:83	LOOP		60 Reply
12	8.019447451	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
13	8.504442043	172.16.30.1	172.16.30.254	ICMP	98 Echo (ping) request id=0xb2b2, seq=4/1024, ttl=64 (reply in 14)
14	8.504784029	172.16.30.254	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=4/1024, ttl=64 (request in 13)
15	10.602429368	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
16	12.634442953	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
17	12.618458957	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=1/256, ttl=64 (reply in 18)
18	12.618959512	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=1/256, ttl=63 (request in 17)
19	13.617461355	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=2/512, ttl=64 (reply in 20)
20	13.617718982	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=2/512, ttl=63 (request in 19)
21	13.631563494	Cisco_3a:fa:83	CDP/VT/DP/DTP/PagP/UDL CDP		435 Device ID: tux-sw3 Port ID: FastEthernet0/1
22	14.634151979	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
23	14.616465185	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=3/768, ttl=64 (reply in 24)
24	14.616933788	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=4/1024, ttl=64 (reply in 25)
25	15.616443454	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=4/1024, ttl=63 (request in 23)
26	15.616701723	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=4/1024, ttl=63 (request in 25)
27	16.638957470	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8003
28	16.616462022	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=5/1280, ttl=64 (reply in 29)

Figura 31: Exp.4: Ping de *tuxy1* para *tuxy4.eth0*

#	Processo	Origem	Destino	Tipo	Comentário
29	14.616465185	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=5/1280, ttl=63 (request in 23)
30	14.616933788	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=4/1024, ttl=64 (reply in 26)
31	15.616443454	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=4/1024, ttl=63 (request in 25)
32	15.616701723	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=4/1024, ttl=63 (request in 26)
33	16.638957470	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
34	16.616462022	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) request id=0xb2b2, seq=5/1280, ttl=64 (reply in 28)
35	16.616914638	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) reply id=0xb2b2, seq=5/1280, ttl=63 (request in 27)
36	17.617461355	172.16.30.1	Cisco_3a:fa:83	LOOP	60 Reply
37	17.617718982	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
38	17.631563494	172.16.30.1	172.16.31.1	ICMP	60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
39	17.6492770149	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
40	17.6492888319	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
41	17.6496785341	172.16.30.1	172.16.31.1	ICMP	60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
42	17.6497775712	Cisco_3a:fa:83	Spanning-tree-(forw.) STP		60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
43	17.6498450117	172.16.30.1	172.16.31.1	ICMP	60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
44	17.6498450767	172.16.31.1	172.16.30.1	ICMP	60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
45	17.6498454007	172.16.30.1	172.16.31.1	ICMP	60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
46	17.6498457664	172.16.31.1	172.16.30.1	ICMP	60 Conf. Root = 32768/30:fc:fb:3a:fa:80 Cost = 0 Port = 0x8003
47	17.6498481053	G-ProCom_B8:e4:4d	HewlettP_5a:7d:74	ARP	42 Who has 172.16.30.254 Tell 172.16.30.1
48	17.6498664199	HewlettP_5a:7d:74	G-ProCom_B8:e4:4d	ARP	60 172.16.30.254 is at 00:21:5a:5a:7d:74
49	17.6501417217	HewlettP_5a:7d:74	G-ProCom_B8:e4:4d	ARP	60 Who has 172.16.30.1 Tell 172.16.30.254

Figura 32: Exp.4: Ping de *tuxy1* para o *router*

4 2.566940543	172.16.31.1	172.16.31.255	NBNS	92 Name query NB WORKGROUP<1d>
5 2.787940298	fe80::221:5aff:fe61...	ff02::fb	MDNS	180 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs...
6 2.788027947	172.16.31.1	224.0.0.251	MDNS	160 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs...
7 2.857215476	Cisco_3a:fa:84	Cisco_3a:fa:84	LOOP	60 Reply
8 3.235193086	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) request id=0x01fe, seq=1/256, ttl=64 (reply in 10)
9 3.235506596	172.16.31.254	172.16.31.1	ICMP	70 Redirect (Redirect for host)
10 3.235775759	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) reply id=0x01fe, seq=1/256, ttl=63 (request in 8)
11 3.568225596	172.16.31.1	172.16.31.255	NBNS	92 Name query NB WORKGROUP<1b>
12 3.788989593	fe80::221:5aff:fe61...	ff02::fb	MDNS	180 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs...
13 3.789072961	172.16.31.1	224.0.0.251	MDNS	160 Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR _nfs...
14 4.089445066	Cisco_3a:fa:84	Spanning-tree-(for... STP	Spanning-tree-(for... STP	60 Conf. Root = 32768/31:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
15 4.264483255	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) request id=0x01fe, seq=2/512, ttl=64 (reply in 17)
16 4.264785182	172.16.31.254	172.16.31.1	ICMP	70 Redirect (Redirect for host)
17 4.265152739	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) reply id=0x01fe, seq=2/512, ttl=63 (request in 15)
18 4.568725860	172.16.31.1	172.16.1.1	DNS	85 Standard query 0x5bdb A WORKGROUP.netlab.fe.up.pt
19 4.568726646	172.16.31.1	172.16.1.1	DNS	95 Standard query 0x5bdb A WORKGROUP.netlab.fe.up.pt

Frame 9: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0  
 ▶ Ethernet II, Src: Cisco\_d6:aab9 (68:ef:bd:d6:aa:b9), Dst: HewlettP\_61:30:63 (00:21:5a:61:30:63)  
 ▶ Internet Protocol Version 4, Src: 172.16.31.254, Dst: 172.16.31.1  
 ▶ Internet Control Message Protocol  
 Type: 5 (Redirect)  
 Code: 1 (Redirect for host)  
 Checksum: 0x3001 [correct]  
 [Checksum Status: Good]  
 Gateway address: 172.16.31.253  
 ▶ Internet Protocol Version 4, Src: 172.16.31.1, Dst: 172.16.30.1  
 ▶ Internet Control Message Protocol

Figura 33: Exp.4: Ping de *tuxy2* para *tuxy1*, redirecionado pelo router através de *tuxy4.eth1*

3 3.08022/8/3	1/2.16.31.1	1/2.16.30.1	ICMP	98 Echo (ping) request id=0x06/d, seq=1/256, ttl=64 (reply in 5)
4 3.080621420	172.16.31.254	172.16.31.1	ICMP	70 Redirect (Redirect for host)
5 3.080925293	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) reply id=0x067d, seq=1/256, ttl=63 (request in 3)
6 3.162248388	Cisco_3a:fa:84	Spanning-tree-(for... STP	Spanning-tree-(for... STP	60 Conf. Root = 32768/31:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
7 4.110305835	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) request id=0x067d, seq=2/512, ttl=64 (reply in 8)
8 4.110690791	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) reply id=0x067d, seq=2/512, ttl=63 (request in 7)
9 4.932419904	Cisco_3a:fa:84	Cisco_3a:fa:84	LOOP	60 Reply
10 5.134299217	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) request id=0x067d, seq=3/768, ttl=64 (reply in 11)
11 5.134665254	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) reply id=0x067d, seq=3/768, ttl=63 (request in 10)
12 5.134665346	Cisco_3a:fa:84	Spanning-tree-(for... STP	Spanning-tree-(for... STP	60 Conf. Root = 32768/31:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
13 6.158306211	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) request id=0x067d, seq=4/1024, ttl=64 (reply in 14)
14 6.158685587	172.16.30.1	172.16.31.1	ICMP	98 Echo (ping) reply id=0x067d, seq=4/1024, ttl=63 (request in 13)
15 7.172830710	Cisco_3a:fa:84	Spanning-tree-(for... STP	Spanning-tree-(for... STP	60 Conf. Root = 32768/31:fc:fb:fb:3a:fa:80 Cost = 0 Port = 0x8004
16 7.1802000250	172.16.31.1	172.16.30.1	ICMP	98 Echo (ping) request id=0x067d, seq=5/1020, ttl=64 (reply in 17)

Figura 34: Exp.4: Ping de *tuxy2* para *tuxy1*, redirecionado pelo router através de *tuxy4.eth1*, utilizando NAT

Experiência 5:

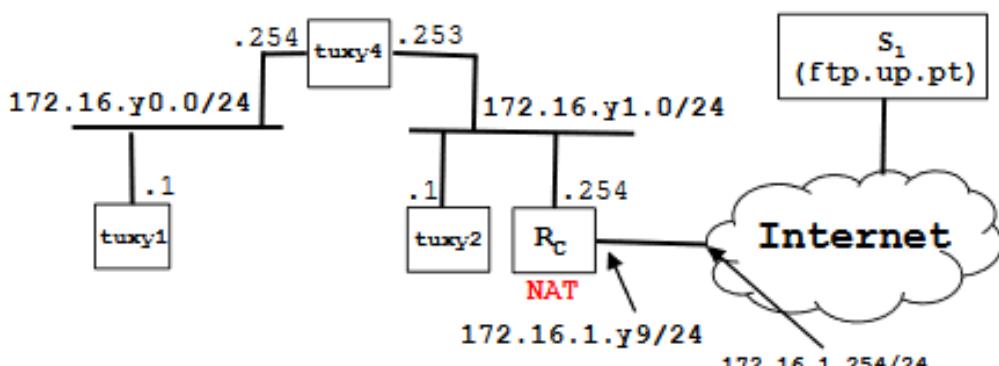


Figura 35: Exp.5: Topologia de rede

3 3.472981044	172.16.31.1	172.16.1.1	DNS	69 Standard query 0xe8d4 A ftp.up.pt
4 3.472992009	172.16.31.1	172.16.1.1	DNS	69 Standard query 0x39de AAAA ftp.up.pt
5 3.474396882	172.16.1.1	172.16.31.1	DNS	355 Standard query response 0xe8d4 A ftp.up.pt CNAME mirrors.up.pt A ...
6 3.474375185	172.16.31.1	172.16.31.1	DNS	367 Standard query response 0x39de AAAA ftp.up.pt CNAME mirrors.up.pt...
7 3.474653357	172.16.31.1	193.137.29.15	ICMP	98 Echo (ping) request id=0x1067, seq=1/256, ttl=64 (reply in 8)
8 3.476878941	193.137.29.15	172.16.31.1	ICMP	98 Echo (ping) reply id=0x1067, seq=1/256, ttl=68 (request in 7)
9 3.476968336	172.16.31.1	172.16.1.1	DNS	86 Standard query 0x2756 PTR 15.29.137.193.in-addr.arpa PTR...
10 3.478127397	172.16.1.1	172.16.31.1	DNS	361 Standard query response 0x2756 PTR 19.29.137.193.in-addr.arpa PTR...
11 4.084644438	Cisco_3a:fa:84	Spanning-tree-(for-)	STP	60 Conf. Root = 32768/31:fc:fb:fb:3a:fe:80 Cost = 0 Port = 0x8004
12 4.476215032	172.16.31.1	193.137.29.15	ICMP	98 Echo (ping) request id=0x1067, seq=2/512, ttl=64 (reply in 13)
13 4.478250233	193.137.29.15	172.16.31.1	ICMP	98 Echo (ping) reply id=0x1067, seq=2/512, ttl=58 (request in 12)
14 5.477334272	172.16.31.1	193.137.29.15	ICMP	98 Echo (ping) request id=0x1067, seq=3/768, ttl=64 (reply in 15)
15 5.479069790	193.137.29.15	172.16.31.1	ICMP	98 Echo (ping) reply id=0x1067, seq=3/768, ttl=58 (request in 14)

Figura 36: Exp.5: Funcionamento do DNS, com o comando *ping*

Experiência 6:

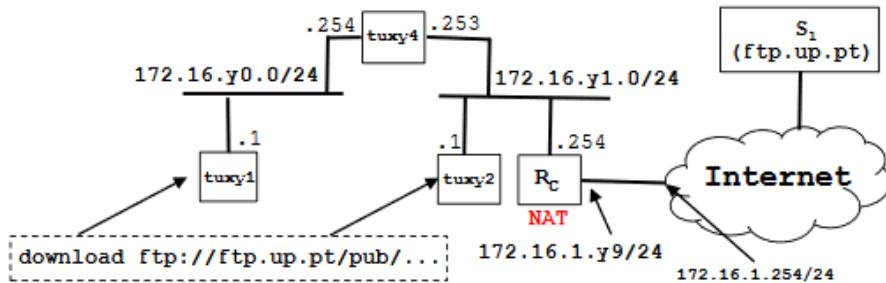


Figura 37: Exp.6: Topologia de rede

40 2.324305273	193.137.29.15	172.16.30.1	TCP	12 Response: 200
47 2.324371333	172.16.30.1	193.137.29.15	TCP	66 36545 - 21 [ACK] Seq=35 Ack=628 Win=29312
48 2.324381043	172.16.30.1	193.137.29.15	TCP	66 36545 - 21 [ACK] Seq=35 Ack=640 Win=29312
49 2.324608588	193.137.29.15	172.16.30.1	FTP	103 Response: 250 Directory successfully changed
50 2.324687513	172.16.30.1	193.137.29.15	FTP	70 Request: pasv
51 2.368670894	193.137.29.15	172.16.30.1	FTP	66 21 - 36545 [ACK] Seq=677 Ack=39 Win=29056
52 2.368701071	172.16.30.1	193.137.29.15	FTP	68 Request:
53 2.370967930	193.137.29.15	172.16.30.1	FTP	66 21 - 36545 [ACK] Seq=677 Ack=41 Win=29056
54 2.371327755	193.137.29.15	172.16.30.1	FTP	118 Response: 227 Entering Passive Mode (193,137,29,15)
55 2.371530754	172.16.30.1	193.137.29.15	TCP	74 38922 - 52461 [SYN] Seq=0 Win=29200 Len=0
56 2.373856868	193.137.29.15	172.16.30.1	TCP	74 52461 - 38922 [SYN, ACK] Seq=0 Ack=1 Win=29056
57 2.373883269	172.16.30.1	193.137.29.15	TCP	66 38922 - 52461 [ACK] Seq=1 Ack=1 Win=29312
58 2.373930614	172.16.30.1	193.137.29.15	FTP	70 Request: RETR
59 2.417673511	193.137.29.15	172.16.30.1	TCP	66 21 - 36545 [ACK] Seq=729 Ack=45 Win=29056
60 2.417701888	172.16.30.1	193.137.29.15	FTP	81 Request: RECENT-Z.json
61 2.419841807	193.137.29.15	172.16.30.1	TCP	66 21 - 36545 [ACK] Seq=729 Ack=60 Win=29056
62 2.420312813	193.137.29.15	172.16.30.1	FTP	143 Response: 150 Opening BINARY mode data conn
63 2.420693862	193.137.29.15	172.16.30.1	FTP-DA..	2802 FTP Data: 2736 bytes (PASV) (RETR)
64 2.420710159	172.16.30.1	193.137.29.15	TCP	66 38922 - 52461 [ACK] Seq=1 Ack=2737 Win=3468

Figura 38: Exp.6: Transferência de um ficheiro, utilizando a aplicação desenvolvida

70 2.424491010	172.16.30.1	193.137.29.15	TCP	99 38922 - 52461 [ACK] Seq=1 ACK=1 Win=29056
77 2.424998788	193.137.29.15	172.16.30.1	FTP-DA..	2802 FTP Data: 2736 bytes (PASV) (RETR)
78 2.425009294	172.16.30.1	193.137.29.15	TCP	66 38922 - 52461 [ACK] Seq=1 Ack=21889 Win=730
79 2.425250600	193.137.29.15	172.16.30.1	FTP-DA..	2802 FTP Data: 2736 bytes (PASV) (RETR)
80 2.425261888	172.16.30.1	193.137.29.15	TCP	66 38922 - 52461 [ACK] Seq=1 Ack=24625 Win=784
81 2.425499264	193.137.29.15	172.16.30.1	FTP-DA..	2802 [TCP Previous segment not captured] FTP Data
82 2.425510317	172.16.30.1	193.137.29.15	TCP	78 [TCP Window Update] 38922 - 52461 [ACK] Seq=1 ACK=1 Win=29056
83 2.425748800	193.137.29.15	172.16.30.1	FTP-DA..	2802 FTP Data: 2736 bytes (PASV) (RETR)
84 2.425760219	172.16.30.1	193.137.29.15	TCP	78 [TCP Window Update] 38922 - 52461 [ACK] Seq=1 ACK=1 Win=29056
85 2.425996757	193.137.29.15	172.16.30.1	FTP-DA..	1434 FTP Data: 1368 bytes (PASV) (RETR)
86 2.426006903	172.16.30.1	193.137.29.15	TCP	78 [TCP Window Update] 38922 - 52461 [ACK] Seq=1 ACK=1 Win=29056
87 2.426010858	193.137.29.15	172.16.30.1	FTP-DA..	1434 [TCP Previous segment not captured] FTP Data
88 2.426017044	172.16.30.1	193.137.29.15	TCP	86 [TCP Window Update] 38922 - 52461 [ACK] Seq=1 ACK=1 Win=29056
89 2.426568960	193.137.29.15	172.16.30.1	FTP-DA..	2802 FTP Data: 2736 bytes (PASV) (RETR)
90 2.426580274	172.16.30.1	193.137.29.15	TCP	86 [TCP Window Update] 38922 - 52461 [ACK] Seq=1 ACK=1 Win=29056
91 2.427113452	193.137.29.15	172.16.30.1	FTP-DA..	1434 FTP Data: 1368 bytes (PASV) (RETR)
92 2.427125102	172.16.30.1	193.137.29.15	TCP	86 [TCP Window Update] 38922 - 52461 [ACK] Seq=1 ACK=1 Win=29056
93 2.427367826	193.137.29.15	172.16.30.1	FTP-DA..	4170 FTP Data: 4104 bytes (PASV) (RETR)

Figura 39: Exp.6: Transferência simultânea de ficheiros em *tuxy1* e *tuxy2* (Captura de *tuxy1*)

95	2.427615773	193.137.29.15	172.16.30.1	FTP-DA..	2802 [TCP Data: 2736 bytes (PASV) (RETR)]
96	2.427626686	172.16.30.1	193.137.29.15	TCP	86 [TCP Window Update] 38922 → 52461 [ACK] Seq=86 38922 → 52461 [ACK] Seq=1 Ack=25993 Win=128
97	2.427863981	193.137.29.15	172.16.30.1	TCP	1434 [TCP Out-of-Order] 52461 → 38922 [ACK] Seq=1 Ack=25993 Win=128
98	2.427874949	172.16.30.1	193.137.29.15	TCP	86 38922 → 52461 [ACK] Seq=1 Ack=25993 Win=128
99	2.427878774	193.137.29.15	172.16.30.1	FTP-DA..	1434 FTP Data: 1368 bytes (PASV) (RETR)
100	2.427885400	172.16.30.1	193.137.29.15	TCP	86 [TCP Window Update] 38922 → 52461 [ACK] Seq=86 38922 → 52461 [ACK] Seq=1 Ack=25993 Win=128
101	2.428114821	193.137.29.15	172.16.30.1	TCP	1434 [TCP Out-of-Order] 52461 → 38922 [ACK] Seq=1 Ack=25993 Win=128
102	2.428125372	172.16.30.1	193.137.29.15	TCP	78 38922 → 52461 [ACK] Seq=1 Ack=34201 Win=128
103	2.428365700	193.137.29.15	172.16.30.1	FTP-DA..	1434 FTP Data: 1368 bytes (PASV) (RETR)
104	2.428375946	172.16.30.1	193.137.29.15	TCP	78 [TCP Window Update] 38922 → 52461 [ACK] Seq=78 38922 → 52461 [ACK] Seq=1 Ack=34201 Win=128
105	2.428878232	193.137.29.15	172.16.30.1	TCP	2802 [TCP Out-of-Order] 52461 → 38922 [ACK] Seq=1 Ack=34201 Win=128
106	2.428890232	172.16.30.1	193.137.29.15	TCP	78 38922 → 52461 [ACK] Seq=1 Ack=36937 Win=134
107	2.429139924	193.137.29.15	172.16.30.1	TCP	1434 [TCP Out-of-Order] 52461 → 38922 [ACK] Seq=1 Ack=36937 Win=134
108	2.429150691	172.16.30.1	193.137.29.15	TCP	78 38922 → 52461 [ACK] Seq=1 Ack=38305 Win=135
109	2.429390047	193.137.29.15	172.16.30.1	TCP	1434 [TCP Out-of-Order] 52461 → 38922 [ACK] Seq=1 Ack=38305 Win=135
110	2.429401095	172.16.30.1	193.137.29.15	TCP	66 38922 → 52461 [ACK] Seq=1 Ack=54721 Win=134
111	2.429404914	193.137.29.15	172.16.30.1	FTP-DA..	1434 FTP Data: 1368 bytes (PASV) (RETR)

Figura 40: Exp.6: Transferência simultânea de ficheiros em *tuxy1* e *tuxy2* (Capterura de *tuxy2*)

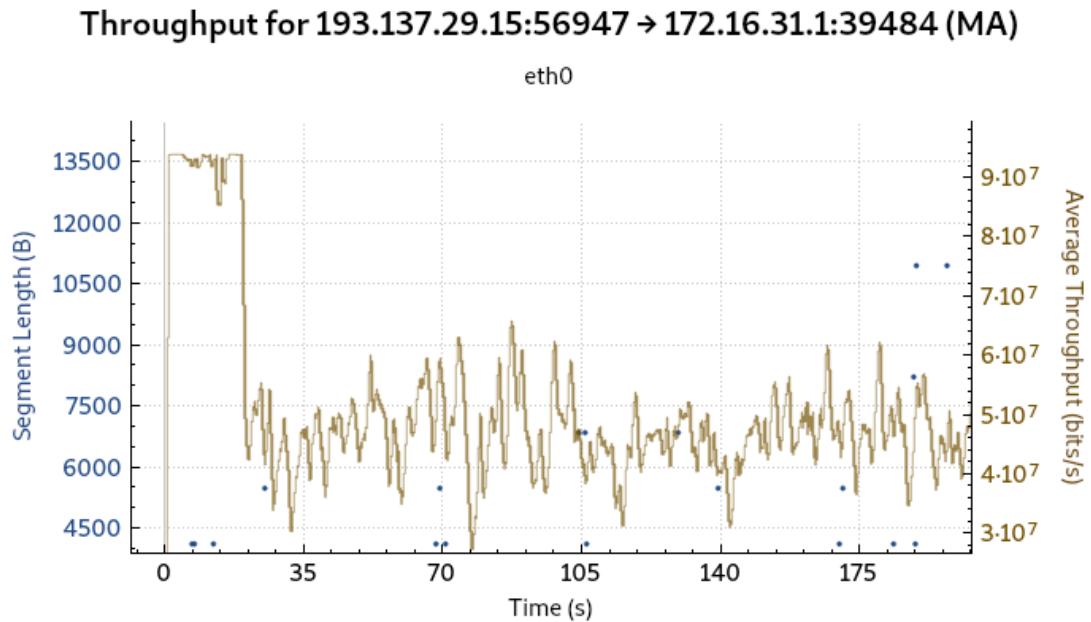


Figura 41: Exp.6: Gráfico da evolução do *throughput* da conexão TCP, para o *tuxy1*, ao longo do tempo, para a transferência simultânea de ficheiros em *tuxy1* e *tuxy2*

Nota: A secção no início do gráfico, em que o *throughput* é máximo, corresponde ao período de tempo antes de ser iniciada a transferência em *tuxy2*, e *tuxy1* tem toda a largura de banda à sua disposição

Experiência 7: (*notar a modificação da source e do destino nos pacotes, ao passarem por tuxy4*)

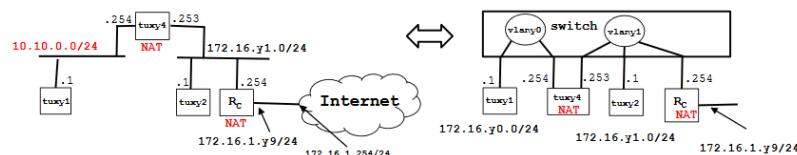


Figura 42: Exp.7: Topologia de rede

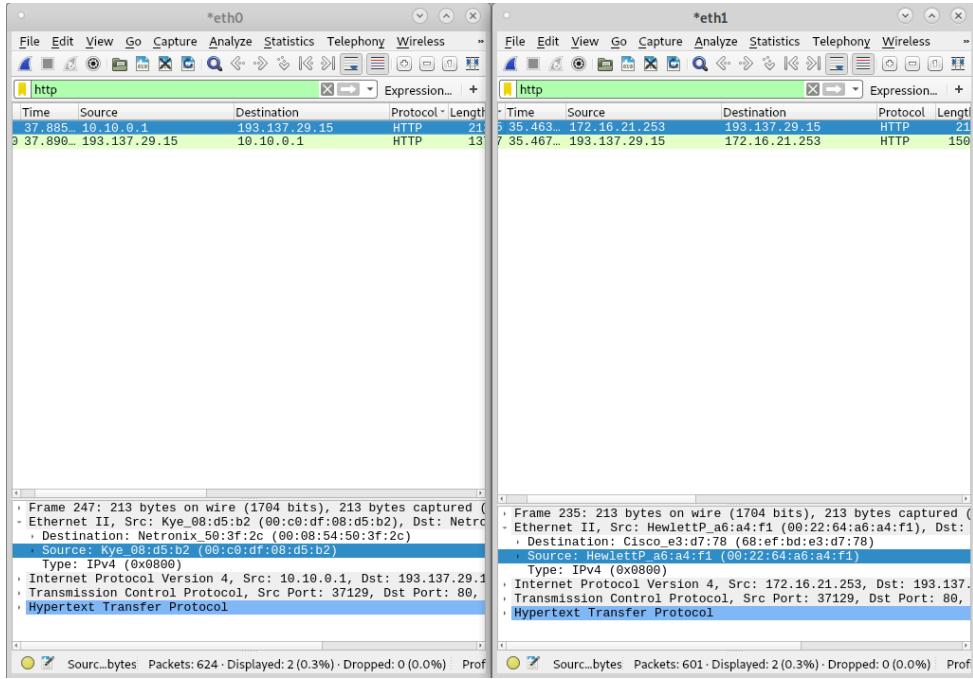


Figura 43: Exp.7: Tráfego HTTP gerado em ambas as interfaces de *tuxy4*, de *tuxy1* para *tuxy2*

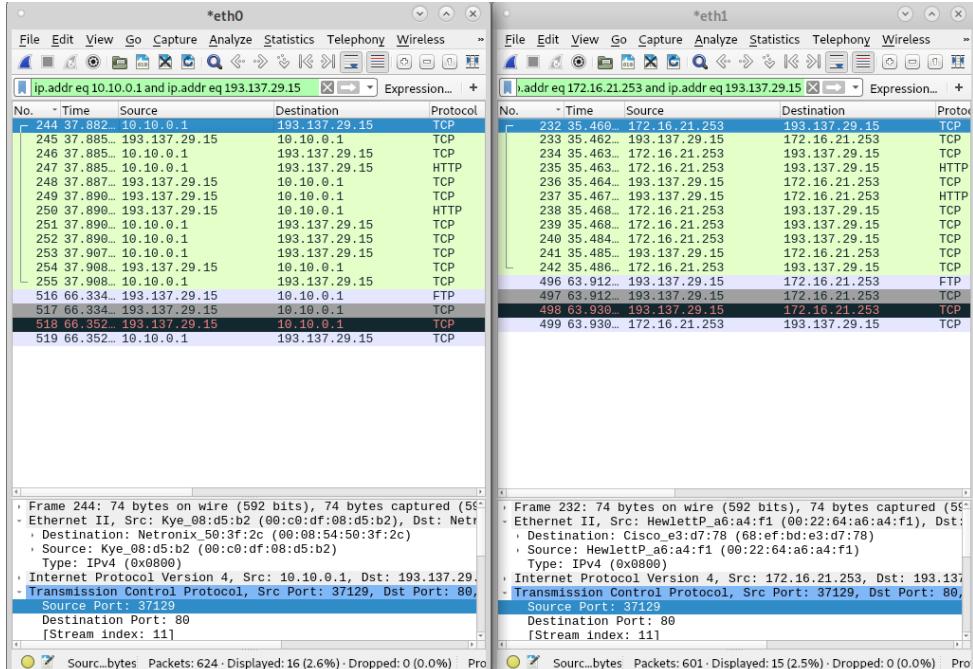


Figura 44: Exp.7: Tráfego TCP gerado em ambas as interfaces de *tuxy4*, de *tuxy1* para *tuxy2*, pela execução do comando *wget*

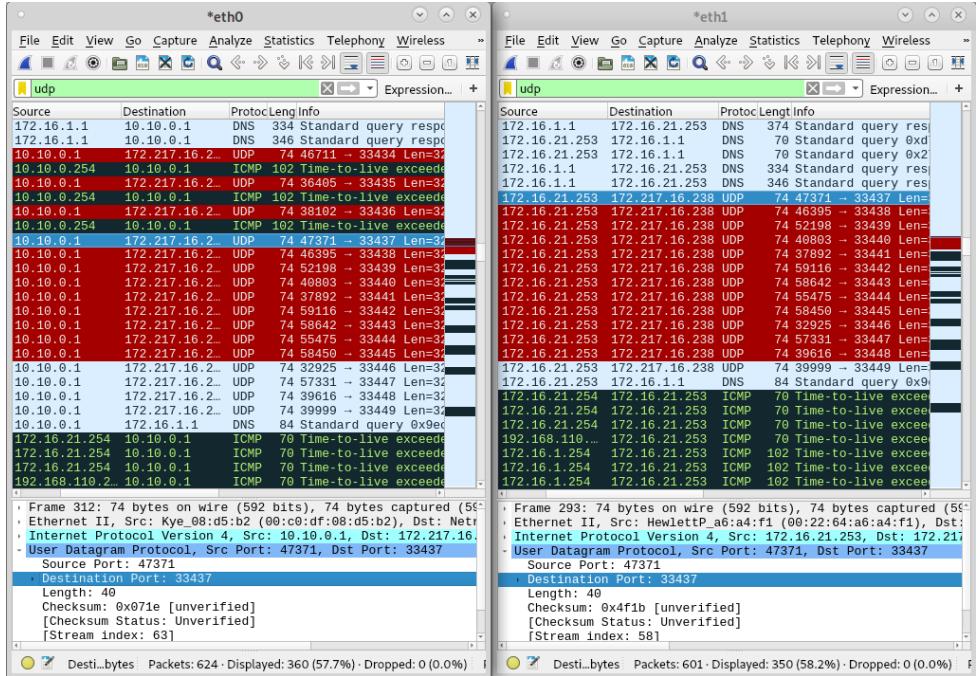


Figura 45: Exp.7: Tráfego UDP gerado em ambas as interfaces de *tuxy4*, de *tuxy1* para *tuxy2*, pela execução do comando *traceroute*

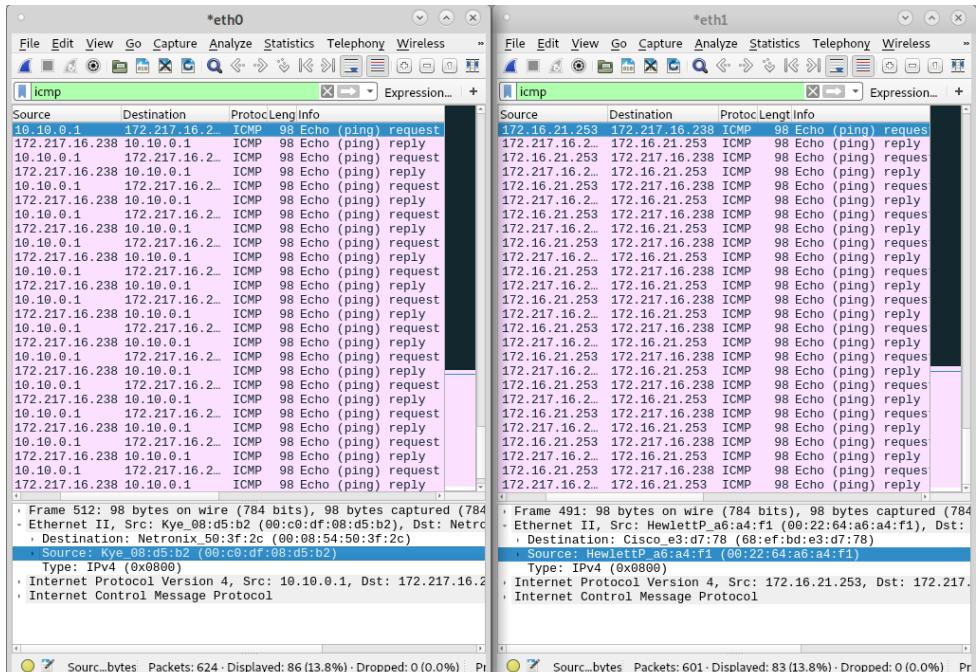


Figura 46: Exp.7: Tráfego ICMP gerado em ambas as interfaces de *tuxy4*, de *tuxy1* para *tuxy2*, pela execução do comando *ping*