Configuração de uma rede e desenvolvimento de uma aplicação de download

Redes de Computadores - 2º Trabalho Laboratorial

Turma 5a1530- Grupo G

up201904814@edu.fe.up.pt José Castro Baptista
up201905296@edu.fe.up.pt Tomás Vasconcelos Estácio

Junho, 2022

Índice

1 Introdução

Este relatório é realizado no âmbito do segundo trabalho laboratorial de Redes de Computadores que se desenvolveu ao longo de diversas aulas laboratoriais, e tem como principal objetivo o estudo de uma rede de computadores, da sua configuração e posterior ligação a uma aplicação desenvolvida pelo grupo. Para tal, além de seguir as recomendações e instruções fornecidas no guião, o grupo teve de fazer pesquisas acerca do funcionamento do protocolo em questão e da respetiva ligação ao servidor em uso. O protocolo usado no trabalho foi o FTP com auxílio de um servidor da Universidade do Porto (ftp.up.pt) e outro servidor do Técnico Lisboa (ftp.rnl.tecnico.ulisboa.pt).

O trabalho foi concluído com sucesso, com a ajuda e o apoio de documentos RFC (Request for Comments) que descrevem padrões de protocolos. Relativamente à configuração de uma rede, o seu objetivo é permitir a execução de uma aplicação, a partir de duas VLAN's dentro de um *switch*. Posteriormente, foi desenvolvida uma aplicação de download de acordo com o protocolo FTP e com a ajuda de ligações TCP/IP (*Transmission Control Protocol/Internet Protocol*) a partir de *sockets*. Numa das VLAN, realizou-se uma ligação para um router, no qual foi implementado a NAT, estando este ativo, e na outra não, tendo esta última ligação à Internet para a aplicação de download funcionar corretamente.

Precisamos de entender o que é um cliente, um servidor e as suas especificidades em TCP/IP, saber como se caraterizam protocolos em aplicações no geral, como definir um URL e descrever o comportamento de um servidor FTP. Para além desses conhecimentos, foi necessário perceber a importância do DNS como um banco de dados distribuído que faz o mapeamento de nome de host em endereço IP e vice-versa, permitindo a sua localização num host com domínio determinado.

Quanto ao relatório, o seu objetivo é expor e explicar toda a componente teórica presente neste segundo trabalho, tendo a seguinte estrutura:

- **Introdução**
  Descrição dos objetivos do trabalho;
- **Parte 1: Aplicação de Download FTP**
  Arquitetura da mesma, sendo também apresentados os resultados de um download bem-sucedido e a sua análise;
- **Parte 2: Configuração de Rede e Análise**
  Arquitetura de cada experiência, objetivos experimentais, comandos de configuração principais e análise dos logs feitos durante a sua realização, tendo por base as perguntas previamente fornecidas pelo docente da unidade curricular.
- **Conclusões**
  Síntese da informação apresentada nos tópicos anteriores e uma opinião final do grupo acerca do projeto;
- **Referências**
  Documentos/sites consultados pelo grupo;
- **Anexos**
  Código da aplicação de download, comandos de configuração e os logs/ capturas feitas usando o Wireshark.

2 Parte 1 – Aplicação de download

Uma das componentes do segundo trabalho de Redes de Computadores era o desenvolvimento de uma aplicação de download na linguagem de programação C, que aceita um URL como argumento do tipo:

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

O campo deverá conter o username com que o utilizador deseja entrar no servidor, e a respetiva password, o campo [host] indicará o endereço do servidor FTP ao qual se deseja conectar, e [url-path] o caminho para o ficheiro que se pretende transferir.

No caso de desejar entrar de forma anónima, o utilizador pode introduzir o username "anonymous"e qualquer password ou pode omitir os campos do <user>:<password> ficando o argumento da forma:

ftp://:@<host>/<url-path>

Para a sua implementação o grupo teve de estudar vários documentos, especificamente o RFC959 que aborda o protocolo de transferência de ficheiros (FTP).

De seguida iremos descrever resumidamente o plano de implementação do programa e quais as suas funcionalidades, assim como a apresentação de resultados e a sua análise.

2.1 Arquitetura

A aplicação de Download está dividida em três partes: argumentos(args), conexão(connectClient) e download(main):

2.1.1 Args

Neste módulo é feito o processamento dos argumentos da aplicação download. A partir da função parseArgs() é guardada a informação da URL dos argumentos do download para um elemento da struct args. No início da função são guardados os seguintes campos:

- **Utilizador**
- **Palavra passe**
- **Host**
- **Path**

```
typedef struct args{
    char user[128];
    char password[128];
    char host[128];
    char url_path[128];
    char file_name[128];
    char host_name[128];
    char ip[128];
} args;
```

No final são usadas as funções getIP() e getFileName() para, respetivamente, a partir do URL, usando a função gethostbyname(), saber o endereço IP do Host e o nome do Host, e para extrair, do último elemento do path, o nome do ficheiro que se vai descarregar, para no final do programa, ao criar o ficheiro transferido, saber que nome lhe atribuir. A porta TCP utilizada para controlo da conexão é a 21.

2.1.2 connectClient

Neste módulo temos como primeira função client_init(), que é encarregada de, ao receber um endereço IP e uma porta, criar um socket e iniciar a ligação a esse socket.

A função clientCommand recebe um file descriptor de um socket e uma string com o comando a ser enviado. Com recurso à função write, o comando é enviado para o servidor, escrevendo-o no socket.

A função readResponde executa a abertura do ficheiro do socket de ligação de controlo de forma a conseguir ler as respostas recebidas pelo servidor aos comandos enviados. Utiliza a função fgets para executar a leitura de cada linha da resposta até que o quarto caracter de uma linha a ser lida seja igual a ' ' (espaço). A função pasvMode usa a função readResponse já criada para ler a resposta do servidor ao comando "pasv" enviado.

Esta resposta contém o IP separado em quatro valores decimais (ipv4) e dois números que permitem fazer o cálculo da porta necessária para nos conectarmos ao servidor e conseguirmos descarregar o ficheiro de lá.

A função writeFile está encarregue de criar o ficheiro que irá transferir para a máquina e preencher o ficheiro com os dados que recebe do servidor. Enquanto é feita a leitura dos dados, estes são adicionados ao ficheiro sequencialmente, até o buffer de leitura estar vazio.

2.1.3 Download

O módulo de download apenas contém a função main com os passos necessários para fazer o download de um ficheiro. O programa começa por verificar se apenas foi passado um argumento a partir da linha de comandos, caso isto não se verifique, é apresentada uma mensagem de erro e o programa termina.

Se o programa receber um URL, este é passado por argumento para a função parseArgs que preenche a struct args com os componentes do URL. Uma lista de detalhes sobre a ligação é imprimida na consola e o programa continua a sua execução, chamando o client_init, que cria um socket para conseguir comunicar com o servidor.

O programa procede com uma invocação de readResponse para ler a mensagem de ligação e esvaziar o buffer de leitura. Nesta altura inicia-se a sequência de login, na qual o programa envia o comando "user" com um nome de utilizador, lê a resposta com readResponse() e executa as mesmas duas funções com o comando "pass .". Caso não sejam especificadas as credenciais de login, o nome de utilizador usado é "anonymous" e a palavra passe usada é "pass".

O programa executa o envio da palavra passe mesmo que não seja necessário, isto não afeta a troca de mensagens e simplifica o código. o próximo comando a enviar é "pasv", para pedir ao servidor que transfira dados em modo passivo, o que faz com que seja necessário abrir outra ligação. A resposta a este comando é lida na função pasvMode e contém o endereço IP e a porta à qual se realizará a nova ligação. Esta é feita pelo client_init e devolve um novo file descriptor do socket de onde se retirará o ficheiro pretendido.

Para finalizar é enviado o comando "retr [path]"com o caminho para o ficheiro e o programa chama a função writeFile para guardar num ficheiro os dados lidos a partir do novo socket. Se não tiver ocorrido nenhum erro durante o processo, é apresentada uma mensagem de sucesso, que indica que o ficheiro foi transferido. Por último, o programa fecha os dois sockets abertos. (nos anexos temos a captura 22 da aplicação de download bem-sucedida).

```
150 Opening BINARY mode data connection for raspbian/snapshotindex.txt (485839 bytes).
226 Transfer complete.
< quit

221 Goodbye.
```

2.2 Resultados

O nosso programa foi testado para diversos casos, nomeadamente a utilização de diferentes logins (introdução de username e password e entrada em modo anónimo), e a utilização de diferentes tipos e tamanhos, nos ficheiros transferidos.

Para maior compreensão do processo, são imprimidos na consola todos os passos efetuados, assim como as respetivas respostas do servidor. Concluímos todos os requisitos desta primeira parte com sucesso, pelo que a aplicação se encontra totalmente funcional e de acordo com o especificado.

3 Parte 2 – Configuração de rede e análise

3.0 Atribuição dos tuxs e bancada

- Tux1 do guião é o nosso tux2
- Tux2 do guião é o nosso tux3
- Tux4 do guião é o nosso tux4

A bancada que nos foi atribuída inicialmente foi a 6 nas experiências 1, 2 e 3, e nas experiências 4, 5 e 6 usamos a bancada 4.

Bancada 6

Tux	Mac Adress
Tux62	00:21:5a:61:2d:df
Tux63	00:22:64:a7:32:cb
Tux64	00:21:5a:5a:75:bb

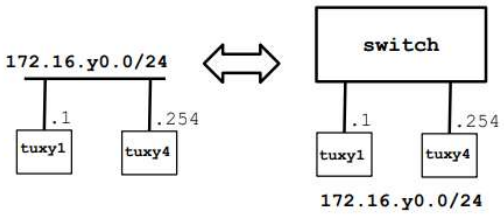
Bancada 4

Tux	Mac Adress
Tux42	00:22:64:19:02:ba
Tux43	00:21:5a:61:2f:13
Tux44	00:21:5a:c3:78:76

Router	Mac Adress
Rc	00:1e:7a:9c:83:3c

3.1 Experiência 1

Nesta experiência foram configurados endereços IP de duas máquinas ligadas através de um switch para entender melhor como funciona a comunicação entre máquinas e o significado e funcionamento de pacotes ARP.



3.1.1 Questões

1. O que são pacotes ARP?

O protocolo *Address Resolution Protocol* (ARP) é um protocolo de comunicação utilizado para um determinado computador descobrir o endereço da camada de ligação associado ao endereço de IP (ipv4). Serve para mapear o endereço de rede a um endereço físico, por exemplo um endereço MAC.

Para enviar uma trama para um dispositivo na rede, o dispositivo que assim o pretende tem de descobrir o endereço MAC correspondente ao endereço IP do dispositivo com o qual quer comunicar. Para tal, envia em broadcast um pacote ARP que contém o endereço de IP desse mesmo dispositivo e espera uma resposta com o endereço MAC que lhe corresponde.

2. Quais são os endereços MAC e IP dos pacotes ARP e porquê?

Ao usar o comando ***ifconfig*** no *tux62*, pudemos verificar que o seu IP tinha sido bem configurado, tendo ele sido configurado como *172.16.60.1*, e que o seu endereço MAC era *00:21:5a:61:2d:df*. No *tux64* o procedimento foi o mesmo, e o seu IP foi configurado como

172.16.60.254 e o seu endereço MAC era *00:21:5a:5a:75:bb.* Posteriormente o *tux62* responde, dizendo que é ele que tem aquele *IP* enviando o seu endereço MAC.

Inicialmente (imagem da esquerda), o pacote ARP enviado pelo tux62 quando fazemos ping 172.16.60.254 tem como endereço MAC e IP do sender os do tux62 (emissor) e como endereço IP do target o do tux64 (recetor), sendo que o endereço MAC do target se encontra a 00:00:00:00:00:00 uma vez que é essa a informação que queremos obter. De facto, na imagem da direita vemos o pacote ARP enviado como resposta ao inicial e vemos que este já contem o endereço MAC do sender e do target.

```
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: HewlettP_61:2d:df (00:21:5a:61:2d:df)
    Sender IP address: 172.16.60.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 172.16.60.254
```

```
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: HewlettP_5a:75:bb (00:21:5a:5a:75:bb)
    Sender IP address: 172.16.60.254
    Target MAC address: HewlettP_61:2d:df (00:21:5a:61:2d:df)
    Target IP address: 172.16.60.1
```

3. Quais pacotes são gerados pelo comando ping?

O comando *ping* é utlizado para testar a conectividade entre o tux62 e o tux64 e gera tanto pacotes ARP (onde se obtém o endereço MAC), como pacotes ICMP (*Internet Control Message Protocol)*. Estes pacotes são usados para transferir mensagens de controlo entre endereços IP.

O facto de nesta experiência o envio de pacotes ARP ter acontecido antes do envio de pacotes ICMP, deve-se ao facto de o tux62 não conter na sua tabela ARP informação acerca do endereço MAC do tux64. Assim sendo, se repetíssemos o ping 172.16.60.254 no tux62 apenas iríamos capturar pacotes ICMP, visto que já obtivemos anteriormente o endereço MAC correspondente ao IP de destino.

4. Quais são os endereços MAC e IP dos pacotes ping?

Os endereços de origem e destino dos pacotes vão ser os designados na tabela seguinte:

	MAC		**IP**	
	Origem	Destino	Origem	Destino
Pacote Pedido	00:21:5a:61:2d:df	00:21:5a:5a:75:bb	172.16.60.1	172.16.60.254
Pacote Resposta	00:21:5a:5a:75:bb	00:21:5a:61:2d:df	172.16.60.254	172.16.60.1

Tabela 1: endereços de origem e destino dos pacotes ARP no ping

5. Como determinar se a trama Ethernet recebida é do tipo ARP, IP ou ICMP?

É possível obter esta informação inspecionando o cabeçalho de uma trama Ethernet. Se o valor for 0x0800, representa um pacote do tipo IP, sendo que neste caso também é possível analisar o IP header. Se este tomar o valor de 1, quer dizer que se trata de um protocolo do tipo ICMP. Contudo, se o valor for 0x806, representa um pacote do tipo ARP.

6. Como determinar o comprimento de uma trama recebida?

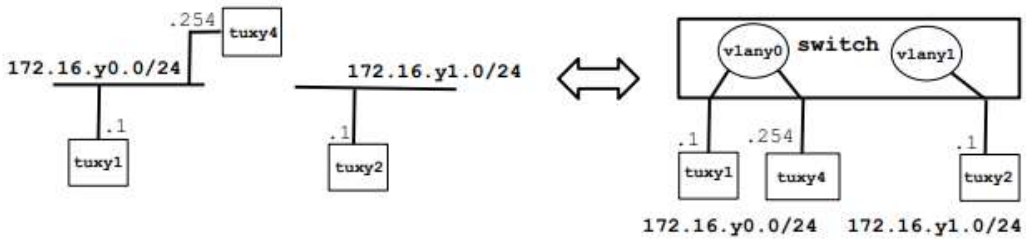
Através da utilização do Wireshark é possível inspecionar a trama e observar o seu comprimento, tal como a captura 1 do anexo que representa uma trama do tipo ARP com 60 bytes de comprimento.

7. O que é a interface loopback e qual a sua importância?

A interface loopback é uma interface de rede virtual que permite que o computador comunique com ele mesmo, utilizada para realizar testes de diagnóstico de 10 em 10 segundos. É um mecanismo utilizado para testar a correta configuração da rede, permitindo a existência de um IP sempre ativo no router, o que descarta a dependência numa interface física.

3.2 Experiência 2

Esta experiência tem como objetivo a criação de duas VLAN's (vlan60 e vlan 61) no switch e perceber a conectividade entre os tuxs, depois de os configurar em cada uma das sub-redes. Os computadores *tux62* e *tux64* foram adicionados à primeira, enquanto que o computador *tux63* foi adicionado à segunda.



3.2.1 Questões

1. Como configurar *vlan*?

A configuração física da *vlan60* passa por realizar as ligações corretas. Na régua 1 a porta *cisco->RS232* terá que ser ligada à porta do *switch* na régua 2. O *tux* que se pretende que esteja ligado ao *switch* tem de ter a sua porta S0 ligada à porta *RS232->cisco* da régua 1. De seguida são introduzidos os seguintes comandos no ***GTKTerm*** do *tux* a configurar:

- configure terminal
- vlan 60
- end

Adicionar portas (porta do tux62 e do tux64):

- configure terminal
- interface fastethernet 0/[nº da porta]
- switchport mode access
- switchport access vlan 60
- end

2. Quantos domínios de transmissão existem? O que se pode concluir das capturas do Wireshark?

Nesta configuração existem dois domínios de transmissão. Quando os *tux62 e tux64* fazem *ping broadcast* apenas recebem resposta um do outro e não do *tux63*. O *tux62* recebe do *tux64* e vice-versa. Por outro lado, quando o *tux63* faz *ping broadcast* não recebe qualquer resposta. Assim é possível afirmar que existem dois domínios de transmissão e ainda que os *tux62 e tux64* pertencem a um e o *tux63* pertence a outro.

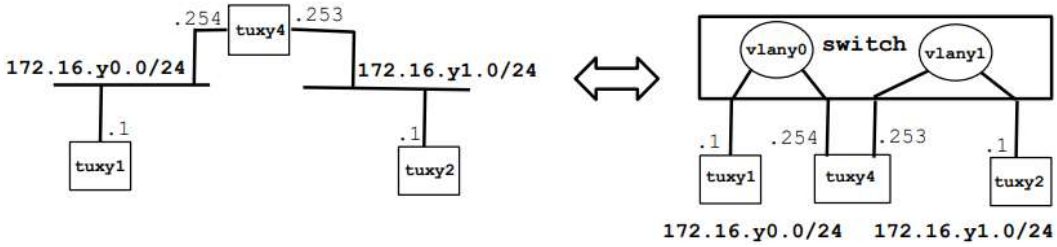
Em específico, vemos que quando fazemos um ping broadcast (ping -b 172.16.60.255) no tux62

este recebe uma resposta apenas do tux64 (172.16.60.254) e, quando fazemos um ping broadcast no tu63 (ping -b 172.16.61.255) este não recebe qualquer tipo de resposta.

Assim sendo, os tux62 e tux64 estão num domínio de Broadcast (vlan 60) diferente do domínio de Broadcast do tux63 (vlan61) o que explica a existência de dois domínios de broadcast identificados.

3.3 Experiência 3

Esta experiência tem como objetivo tornar o tux64 num router para possibilitar a comunicação entre o tux63 e o tux62, através das VLAN's 0 e 1. Para haver comunicação terá de se configurar os endereços IP's das portas ethernet dos tux's e as routes que serão usadas.



3.3.1 Questões

1. **Que rotas há nos tuxs? Qual o seu significado?**

As rotas para as vlans associadas:

a. Tux62 tem uma rota direta para a vlan60 associada à rede 172.16.60.0/24.
b. Tux64 tem uma rota direta para a vlan60 associada à rede 172.16.60.0/24 e uma rota direta para a vlan61 associada à rede 172.16.61.0/24.
c. Tux63 tem uma rota direta para a vlan61 associada à rede 172.16.61.0/24.

As rotas que foram criadas durante a experiência:

a. Tux62 tem uma rota para a vlan61 associada à rede 172.16.61.0/24 pela gateway 172.16.60.254.
b. Tux63 tem uma rota pata a vlan60 associada à rede 172.16.60.0/24 pela gateway 172.16.61.253.

O destino das rotas é até onde o tux que está na origem da rota consegue chegar.

2. **Que informação é que uma entrada da tabela de *forwarding* contém?**

- **Destination***:* o destino da rota.
- **Gateway**: o ip do próximo ponto por onde passará a rota.
- **Netmask**: usado para determinar o ID da rede a partir do endereço IP do destino.
- **Flags**: dá-nos informações sobre a rota.
- **Metric**: o custo de cada 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 isto vai ser o número de falhas da cache (-F) ou o número de sucessos (-C).
- **Interface:** qual a placa de rede responsável pela *gateway* (eth0/eth1).

3. Que mensagens ARP e endereços MAC associados são observados e porquê?

Quando um tux dá *ping* a outro e o tux que recebeu o *ping* não conhece o MAC *address* do que enviou o *ping*, pergunta qual o MAC *address* do tux com aquele IP. E faz isso enviando uma mensagem ARP.

Essa mensagem vai ter o MAC do tux de origem associado e 00:00:00:00:00:00 (mensagem enviada em modo de broadcast) pois ainda não sabe qual o tux de destino. De seguida, o tux de destino responde uma mensagem ARP a dizer o seu MAC *address*.

Esta mensagem vai ter associado tanto o MAC *address* do tux de destino como o de origem.

4. Que ICMP packets são observados e porquê?

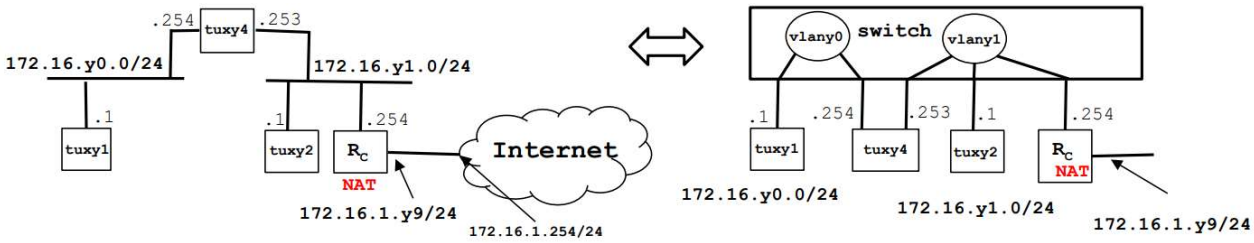
São observados pacotes ICMP de *request* e *reply*, pois depois de serem adicionadas as rotas todos os tux's se conseguem ver uns aos outros. Se não se conseguissem ver, seriam enviados os pacotes ICMP de *Host Unreachable*.

5. Quais são os endereços IP e MAC associados a ICMP packet e porquê?

Os endereços IP e MAC associados com os pacotes ICMP são os endereços IP e MAC dos tux's de origem e destino. Por exemplo, quando se faz *ping* do tux62 para o tux64 (.253) os endereços de origem vão ser 172.16.60.1 (IP) e 00:21:5a:61:2d:df (MAC) e o de destino 172.16.61.253 (IP) e 00:21:5a:5a:75:bb (MAC).

3.4 Experiência 4

Esta experiência tem como objetivo estabelecer uma ligação com a rede dos laboratórios e implementar rotas num router comercial, adicionar-lhe funcionalidade NAT, e perceber qual a sua função.



3.4.1 Questões

1. Como se configura um router estático num router comercial?

De forma a configurar o *router*, foi necessário ligar a porta T4, da régua 1, à porta do *router*, da régua 2. Relativamente à porta T3, da régua 1, esta vai estar ligada à porta S0 do TUX que se pretende que esteja ligado ao *router*. De seguida, invocam-se os seguintes comandos no **GTKTerm** do TUX escolhido:

- configure terminal
- interface fastethernet 0/[nº da porta]
- ip address [ip do router] [mascara]
- no shutdown
- ip route [ip rota de destino] [máscara] [ip gateway]
- exit

2. Quais são as rotas seguidas pelos pacotes durante a experiência e porquê?

No caso de a rota existir, os pacotes usam essa mesma rota. Caso contrário, os pacotes vão ao *router* (rota *default*), o *router* informa que o TUX64 existe, e deverá ser enviado pelo mesmo.

3. Como se configura o NAT num router comercial?

De forma a configurar o *router*, foi necessário configurar as interfaces interna e externa no processo de NAT, que foi feito seguindo o guião fornecido para a dada experiência. A partir do **GTKTerm**, foram inseridos os comandos presentes nos anexos.

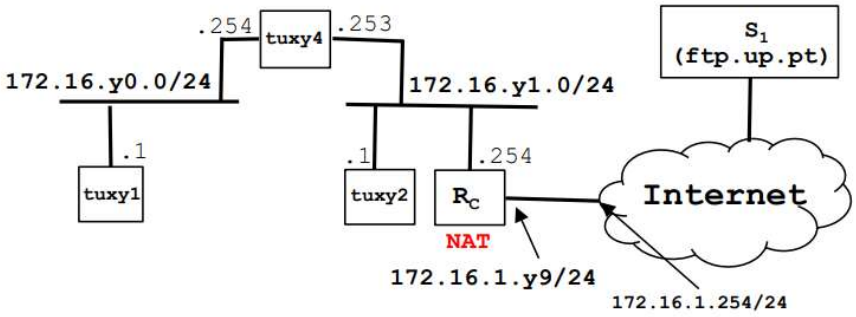
4. O que faz o NAT?

O NAT (*Network Address Translation*) tem como objetivo a conservação de endereços IP. Assim, permite que as redes IP privadas que usem endereços IP não registados se conectem à Internet ou uma rede pública. O NAT opera num *router*, onde conecta duas redes e traduz os endereços privados, na rede interna, para endereços legais, antes que os pacotes sejam encaminhados para outra rede.

Resumindo, permite que os computadores de uma rede interna, como a que foi criada, tenham acesso ao exterior, sendo só necessário um único endereço IP para representar um grupo de computadores fora da sua própria rede.

3.5 Experiência 5

Esta experiência tem como objetivo saber como configurar um serviço DNS numa máquina e descobrir que pacotes são trocados pelo serviço de DNS, para além da informação contida nos mesmos.



3.5.1 Questões

1. Como configurar o serviço DNS num *host*?

De forma a configurar o serviço DNS, é necessário mudar o ficheiro **resolv.conf** pelo uso do comando **nano** que se localiza em **/etc** no *host* tux. Esse ficheiro tem de conter a seguinte informação:

- search netlab.fe.up.pt
- nameserver 172.16.2.1

Onde **netlab.fe.up.pt** é o nome do servidor DNS e 172.16.2.1 o seu endereço de IP. Após esta experiência, é possível aceder à internet nos tux's.

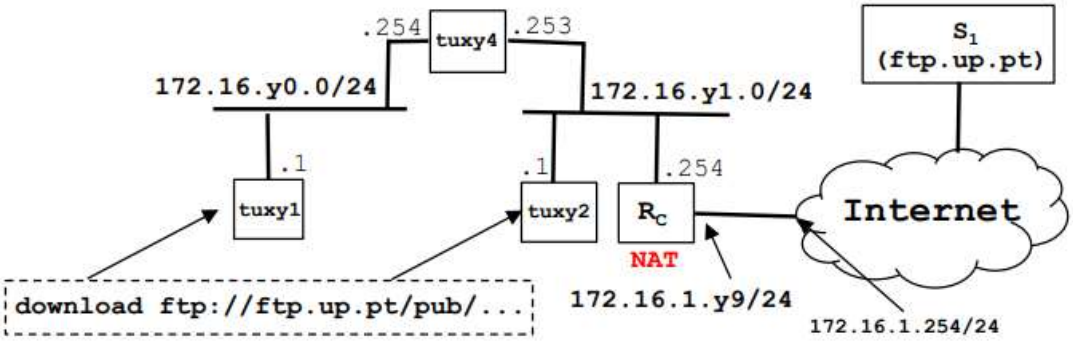
2. **Que pacotes são trocados pelo DNS e que informações são transportadas?**

Em primeiro lugar, temos um pacote enviado do *Host* para o *Server* que contém o *hostname* desejado, pedindo o seu endereço de IP.

O servidor responde com um pacote que contem o endereço IP do *hostname*.

3.6 Experiência 6

Por fim, na experiência 6, compilou-se e executou-se a aplicação desenvolvida e descrita na primeira parte do relatório. Para testar a aplicação, foi usado um servidor FTP e efetuado o download de um ficheiro. O download efetuou-se corretamente, o que demonstrou que a rede estava bem configurada, não trazendo qualquer problema no acesso por protocolo FTP, assim como à utilização de um servidor exterior à rede.



3.6.1 Questões

1. **Quantas conexões TCP foram abertas pela aplicação FTP?**

São abertas 2 conexões TCP pela aplicação FTP, uma de controlo e outra de dados.

2. **Em que conexão é transportado o controlo de informação FTP?**

A informação de controlo FTP é transportada pela conexão de controlo, pela porta TCP 21.

3. **Quais são as fases da conexão TCP?**

Existem 3 fases: o estabelecer da conexão (three-way handshake), a transferência de dados e o fim da conexão.

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

O ARQ TCP assenta num mecanismo de janela deslizante com controlo de erros na transmissão de dados. Para isso o emissor coloca diversas informações em diferentes campos das mensagens. **Acknowledgment Numbers**, que indicam que a trama foi recebida corretamente; **Window size**, que especifica o número de pacotes que o emissor pode enviar sem receber confirmação e o **Sequence number,** que informa o número do pacote a ser enviado.

5. **Como é que o mecanismo de controlo de congestão TCP funciona? Como é que o fluxo de dados da conexão evoluiu ao longo do tempo? Está de acordo com o mecanismo de controlo de congestão TCP?**

O mecanismo de controlo de congestão mantém na rede um número de pacotes estimado para essa rede. Não são enviados mais pacotes do que o mínimo da janela definida pelo recetor e pela janela de congestão. Quando são testados dois *downloads* ao mesmo tempo, no início da conexão verifica-se um aumento da taxa de transferência. Esta acaba por estabilizar após alguns segundos, no entanto é possível visualizar algumas variações, onde as descidas mais significativas representam erros detetados. Esta informação está de acordo com o mecanismo referido.

6. De que forma é afetada a conexão de dados TCP pelo aparecimento de uma segunda conexão TCP? Como?

Com o aparecimento de uma segunda conexão TCP, uma transferência de dados pré-existente sofre uma queda na taxa de transmissão, uma vez que a taxa de transferência passa a ser distribuída de igual forma para cada ligação.

3.5.1 Download em tux1

É possível verificar nas capturas que a nossa aplicação de facto abre 2 ligações TCP para o mesmo servidor, uma para a porta 21 (ligação de controlo) e outra para a porta 35754 (ligação de dados). A informação de controlo é transportada na ligação que foi criada em primeiro lugar para a porta 21, pode até ver-se nos logs os comandos que são enviados e as respostas recebidas. Só depois é que é aberta a ligação de dados.

Cada uma das ligações tem as 3 fases de uma ligação TCP: abertura da ligação, troca de dados e finalmente o encerramento da ligação. O mecanismo de ARQ do TCP é uma variação do Go-Back-N com janela deslizante em que os ACK's contêm apenas um número de sequência que confirma a receção de todos os bytes com um número de sequência mais baixo, ou seja, é o número de sequência do byte que quem envia o ACK pretende receber.

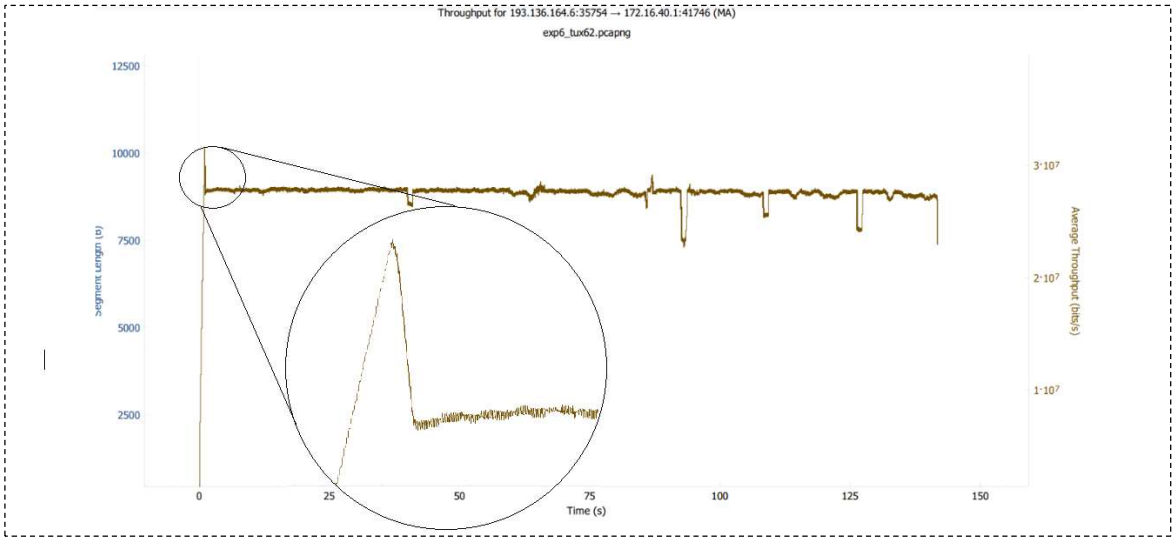
Os campos do cabeçalho TCP mais relevantes para o mecanismo ARQ são o número de sequência e o número de acknowledgement.

O controlo de congestionamento do TCP é o mecanismo que garante que as ligações TCP não geram mais tráfego do que aquele que a rede consegue lidar. Ele funciona variando o tráfego gerado em função de uma medição da congestão atual, que é feita partindo do princípio de que a perda de um pacote é causada muito provavelmente por congestão da rede.

Ou seja, vai-se aumentando o tráfego gerado até haver perda de pacotes, momento em que se pode concluir que se atingiu a capacidade máxima da rede.

O mecanismo tem 2 fases importantes: primeiro aumenta o tráfego exponencialmente de forma a determinar rapidamente a capacidade da rede e depois entra numa fase de evasão de congestão em que vai aumentando o tráfego mais lentamente a partir de um valor calculado a partir do máximo atingido na primeira fase e o reduz quando se perdem pacotes.

O campo relevante para este mecanismo é o **tamanho da janela** que regula quantos dados podem ser enviados antes de se exigir um ACK de retorno.



Neste gráfico que mostra a evolução do throughput, podemos observar o mecanismo de controlo de congestionamento em ação: primeiro uma fase que o throughput aumenta exponencialmente (Slow Start) e de seguida a fase de evasão de congestionamento em que o tráfego aumenta e diminui ligeiramente ao longo do tempo.

3.5.2 Download em tux1 e tux2 simultaneamente

No gráfico podemos observar que a evolução do throughput, a partir da queda aos 40 segundos, foi diminuindo ao longo do tempo com algumas quedas registadas devido ao congestionamento do tráfico das duas ligações TCP/IP formadas.

Também podemos verificar que o mecanismo de controlo ARQ funcionou corretamente havendo várias mudanças de tamanho da janela ao longo da transmissão dos dados, no caso de serem detetados vários NACKs seguidos ou erro no envio ou receção de ACKs seguidos.

3.7 Ligações físicas

Inicialmente, para escrever comandos no switch foi necessário fazer uma ligação através da serial port, entre a porta s0 do tux62 e o switch. Se quiséssemos comunicar com o router Rc em vez do switch esta comunicação teria de ser também através da serial port, logo era estabelecida entre a porta s0 do tux62 e o router Rc.

De seguida, ligamos da porta eth0 do tux2 à entrada fastethernet 0/1 do switch e da porta eth0 do tux64 à entrada fastethernet 0/2 do switch. Desta forma, já temos a nossa vlan60 configurada.

Para configurar a vlan61 ligamos da porta eth0 do tux3 à entrada fastethernet 0/3 do switch, ligamos da porta eth1 do tux4 à entrada fastethernet 0/4 do switch e por fim ligamos da porta fastethernet 0/0 do router Rc à entrada fastethernet 0/5 do switch. De forma a estabelecermos uma ligação ao router do laboratório ligamos a entrada 6.1 à entrada fastethernet 0/1 do router.

4 Conclusões

Este segundo projeto da unidade curricular Redes de Computadores teve como objetivo, numa primeira parte, a implementação de uma aplicação de download de ficheiros recorrendo ao protocolo FTP, e, numa segunda parte, à configuração de uma rede de computadores.

Apesar de ter sido dificultado pelo reduzido tempo que tivemos para aproveitar o laboratório e também por alguns problemas nos PC's, este projeto foi terminado com sucesso, sendo que conseguimos desenvolver todos os aspetos pedidos. Ao mesmo tempo, conseguimos interiorizar conceitos importantes nesta área, e compreendemos os diversos protocolos que foram abordados.

Concluindo, conseguimos alcançar todos os nossos objetivos, e conhecemos conceitos dos quais nunca antes tínhamos ouvido falar, mas que utilizávamos com frequência no nosso dia-a-dia.

5 Referências

- Active FTP vs. Passive FTP, http://slacksite.com/other/ftp.html#passive
- RFC 959 - File Transfer Procol (FTP), https://www.w3.org/Protocols/rfc959/
- Cisco Systems Inc. Cisco 3900 Series, 2900 Series, and 1900 Series Integrated Services Routers Generation 2 Software Configuration
- DNS Types - https://ns1.com/resources/dns-types-records-servers-and-queries
- TCP Congestion Control - https://www.noction.com/blog/tcp-transmission-controlprotocol-congestion-control

6 Anexo 1 – Código da aplicação FTP

6.1 args.c

```
#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "args.h"

//FIRST PROGRAM: obtain passing arguments for the TCP/IP connection
// ftp://<user>:<password>@<host>/<url-path>

int parseArgs(char *url, args *args){
   //returns -1 if error, 0 if success
   char ftp[] = "ftp://";
   int length = strlen(url);
   int state = 0;
   int i = 0;
   int j = 0;

   while(i < length){
      switch(state){
         case 0:
         if(url[i] == ftp[i] && i < 5) break;
         else if(i == 5 && url[i] == ftp[i]) state = 1;
         else printf("Error: not passing ftp\n");
         break;

         case 1:
         if(url[i] == ':'){
            state = 2;
            j = 0;
         }
         else{
            args->user[j] = url[i];
            j++;
         }
         break;

         case 2:
         if(url[i] == '@'){
            state = 3;
            j = 0;
         }
         else{
            args->password[j] = url[i];
            j++;
```

```
            }
            break;

            case 3:
            if(url[i] == '/'){
               state = 4;
               j = 0;
            }
            else{
               args->host[j] = url[i];
               j++;
            }
            break;

            case 4:
            args->url_path[j] = url[i];
            j++;
            break;
         }
         i++;
      }
      //case of anonymous user - define user and pass with something random
      if(strcmp(args->password, "\0") == 0){
         strcpy(args->user, "anonymous");
         strcpy(args->password, "pass");
      }

      if(getIp(args->host, args) != 0){
         printf("Error getIp()\n");
         return -1;
      }

      if(getFileName(args) != 0){
         printf("Error getFileName()\n");
         return -1;
      }

      return 0;

   }

   int getIp(char *host, args *args){
      //returns -1 if error, 0 if success
      struct hostent *h;

      if((h = gethostbyname(host)) == NULL){
         herror("Error getting host name\n");
         return -1;
      }
```

```
    strcpy(args->host_name, h->h_name);
    strcpy(args->ip, inet_ntoa( *( ( struct in_addr *)h->h_addr) )); //32 bits Internet addr with net
ordenation for dotted decimal notation

    return 0;
}

int getFileName(args *args){
    //returns 0 if success
    char fullpath[256];

    strcpy(fullpath, args->url_path);

    char* aux = strtok(fullpath, "/");

    while(aux != NULL){
        strcpy(args->file_name, aux);
        aux = strtok(NULL, "/");
    }

    return 0;
}
```

6.2 args.h

```
#ifndef ARGS_H
#define ARGS_H

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

typedef struct args{
    char user[128];
    char password[128];
    char host[128];
    char url_path[128];
    char file_name[128];
    char host_name[128];
    char ip[128];
} args;

//gets url-path, username, password, Host_name, Ip from Host_name and File_name
int parseArgs(char *url, args *args);
//gets Ip address from hostname and gets hostname from function gethostname()
int getIp(char *host, args *args);
//gets file_name from url-path
```

```
int getFileName(args *args);

#endif
```

6.3 connectClient.c

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <strings.h>
#include <arpa/inet.h>
#include <stdio.h>
#include "connectClient.h"

//starting with initiating the connection in the client side - create a connection socket in server

int client_init(char *ip, int port, int *socketfd){
   //returns -1 if error, 0 if success;
   struct sockaddr_in server_addr;

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

   //open TCP connection
   if((*socketfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
      perror("socket()");
      return -1;
   }

   //connect to server
   if(connect(*socketfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
      perror("connect()");
      return -1;
   }

   return 0;
}

int clientCommand(int socketfd, char * command){
   //sends command, returns -1 if error, 0 if closing connection, 1 if command other than closing the
connection and well sent
   int size;
```

```
    if((size = write(socketfd, command, strlen(command))) <= 0){
        printf("Error: command not sent\n");
        return -1;
    }

    printf("< %s\n", command);
    return 0;
}

int pasvMode(int socketfd, char *ip, int *port){

    char buf[1024];

    if(readResponse(socketfd, buf, sizeof(buf)) == -1){
      printf("Error: cannot enter passive mode\n");
      return -1;
    }

    //< 227 Entering Passive Mode (193,136,28,12,19,91), example

    strtok(buf, "(");
    char* ip1 = strtok(NULL, ",");
    char* ip2 = strtok(NULL, ",");
    char* ip3 = strtok(NULL, ",");
    char* ip4 = strtok(NULL, ",");

    sprintf(ip, "%s.%s.%s.%s", ip1, ip2, ip3, ip4);

    char* aux1 = strtok(NULL, ",");
    char* aux2 = strtok(NULL, ")");

    *port = (256 * atoi(aux1)) + atoi(aux2);

    return 0;
}

int readResponse(int socketfd, char* rd, size_t size){

    char aux[3];
    long num;
    FILE* fd;

    if(!(fd = fdopen(socketfd, "r"))){
        printf("Error opening file to read\n");
        return -1;
    }

    do{
        memset(rd, 0, size);
        rd = fgets(rd, size, fd);
```

```
        printf("%s", rd);
    } while(rd[3] != ' ');

    strncpy(aux, rd, 3);
    num = atoi(aux);
    if(num > 420 && num < 554){
        printf("Command error - «400» (command not accepted and requested action did not take
place) or «500» (syntax error, command unrecognized and the requested action did not take
place)\n");
        return -1;
    }

    return 0;
}

int writeFile(int socketfd, char* filename){
    //returns -1 if error, 0 if success
    int bytes;
    char buf[1024];

    FILE* fd;

    if(!(fd = fopen(filename, "w"))){
        printf("Error opening file to write\n");
        return -1;
    }

    while((bytes = read(socketfd, buf, sizeof(bytes))) > 0){
        if(bytes < 0){
            printf("Error: Nothing received from data socket\n");
            return -1;
        }
        if((fwrite(buf, bytes, 1, fd)) < 0){
            printf("Error writting file in socketfile\n");
            return -1;
        }
    }

    fclose(fd);

    return 0;
}
```

6.4 connectClient.h

```
#ifndef CONNEC_H
#define CONNEC_H

#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <strings.h>
#include <arpa/inet.h>
#include "args.h"

//initiate the connection between client and server with the creation of a connection socket
int client_init(char *ip, int port, int *socketfd);
//client sends commands to socket to be received and read in server
int clientCommand(int socketfd, char * command);
//pasvMode command reception (saves IP and Port number)
int pasvMode(int socketfd, char *ip, int *port);
//reading the different commands responses from server to client
int readResponse(int socketfd, char* rd, size_t size);
//writing on socketfile the communication between client and server
int writeFile(int socketfd, char* filename);

#endif
```

6.5 main.c

```
#include "main.h"

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

  if(argc != 2){
    fprintf(stderr, "usage: download ftp://[<user>:<password>@]<host>/<url-path>\n");
    exit(0);
  }

  args arguments;
  int socketfd_data, socketfd_control;
  char urlcpy[256];
  char command[256];
  char rd[1024];

  strcpy(urlcpy, argv[1]);

  memset(arguments.file_name, 0, 128);
```

```
    memset(arguments.host, 0, 128);
    memset(arguments.host_name, 0, 128);
    memset(arguments.ip, 0, 128);
    memset(arguments.password, 0, 128);
    memset(arguments.url_path, 0, 128);
    memset(arguments.user, 0, 128);

    if(parseArgs(urlcpy, &arguments) == -1){
        printf("usage: %s ftp://<user>:<password>@<host>/<url-path>\n", argv[0]);
        return -1;
    }

    printf("host: %s\nurl path: %s\nuser: %s\npassword: %s\nfile name: %s\nhost name: %s\nip
addr: %s\n", arguments.host, arguments.url_path, arguments.user, arguments.password,
arguments.file_name, arguments.host_name, arguments.ip);

    //TCP port = 21
    if(client_init(arguments.ip, 21, &socketfd_control) == -1){
        printf("Error: init() control\n");
        return -1;
    }

    if(readResponse(socketfd_control, rd, sizeof(rd)) == -1) return -1;

    sprintf(command, "user %s\r\n", arguments.user);

    if(clientCommand(socketfd_control, command) == -1) return -1;
    if(readResponse(socketfd_control, rd, sizeof(rd)) == -1) return -1;

    sprintf(command, "pass %s\r\n", arguments.password);

    if(clientCommand(socketfd_control, command) == -1) return -1;
    if(readResponse(socketfd_control, rd, sizeof(rd)) == -1) return -1;

    char ip[32];
    int port;

    sprintf(command, "pasv\r\n");
    if(clientCommand(socketfd_control, command) == -1) return -1;
    if(pasvMode(socketfd_control, ip, &port)) return -1;

    printf("IP: %s\nPort Number: %d\n", ip, port);

    if((client_init(ip, port, &socketfd_data)) == -1){
        printf("Error: init() data\n");
        return -1;
    }

    sprintf(command, "retr %s\r\n", arguments.url_path);
    if(clientCommand(socketfd_control, command) == -1) return -1;
```

```
    if(readResponse(socketfd_control, rd, sizeof(rd)) == -1) return -1; //response of opening mode
DATA

    if(writeFile(socketfd_data, arguments.file_name) == -1) return -1;
    if(readResponse(socketfd_control, rd, sizeof(rd)) == -1) return -1; //response of finishing the
transfer

    sprintf(command, "quit\r\n");
    if(clientCommand(socketfd_control, command) == -1) return -1;
    if(readResponse(socketfd_control, rd, sizeof(rd)) == -1) return -1;

    close(socketfd_control);
    close(socketfd_data);

    return 0;
}
```

6.5 main.h

```
#ifndef MAIN_H
#define MAIN_H

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#include "connectClient.h"

#endif
```

7 Comandos de configuração

7.1 Configurar vlan60

```
configure terminal
vlan 60
end
show vlan id 60
configure terminal
interface fastethernet 0/1
switchport mode access
switchport access vlan 60
end
show running-config fastethernet 0/1
configure terminal
interface fastethernet 0/2
switchport mode access
switchport access vlan 60
end
show running-config fastethernet 0/2
```

7.2 Configurar vlan61
configure terminal
interface fastethernet 0/3
switchport mode access
switchport access vlan 61
end
show running-config fastethernet 0/3
configure terminal
interface fastethernet 0/4
switchport mode access
switchport access vlan 61
end
show running-config fastethernet 0/4
configure terminal
interface fastethernet 0/5
switchport mode access
switchport access vlan 61
end
show running-config fastethernet 0/5

7.3 Configurar tux2
updateimage
ifconfig eth0 up 172.16.60.1/24
route add -net 172.16.61.0/24 gw 172.16.60.254
route add default gw 172.16.60.254
route -n

7.4 Configurar tux3
updateimage
ifconfig eth0 up 172.16.61.1/24
route add -net 172.16.60.0/24 gw 172.16.61.253
route add -net default gw 172.16.61.254
route -n

route del -net 172.16.60.0/24 gw 172.16.61.253 // comando usado na experiência 4 para mostrar que esta rota não era necessária para se estabelecer uma ligação deste tux à rede 172.16.60.0 desde que a rota default estivesse definida para o router

7.5 Configurar tux4
updateimage
sysctl -w net.ipv4.ip_forward=1
sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
ifconfig eth0 up 172.16.60.254/24
ifconfig eth1 up 172.16.61.253/24
route add -net default gw 172.16.61.254
route -n

7.6 Configurar router (sem NAT)

```
configure terminal
interface fastethernet 0/0
ip address 172.16.61.254 255.255.255.0
no shutdown
exit
show interface fastethernet 0/0
configure terminal
interface fastethernet 0/1
ip address 172.16.2.69 255.255.255.0
no shutdown
exit
show interface fastethernet 0/1
ip route 0.0.0.0 0.0.0.0 172.16.2.254
ip route 172.16.60.0 255.255.255.0 172.16.61.253
```

7.7 Configurar router (com NAT)

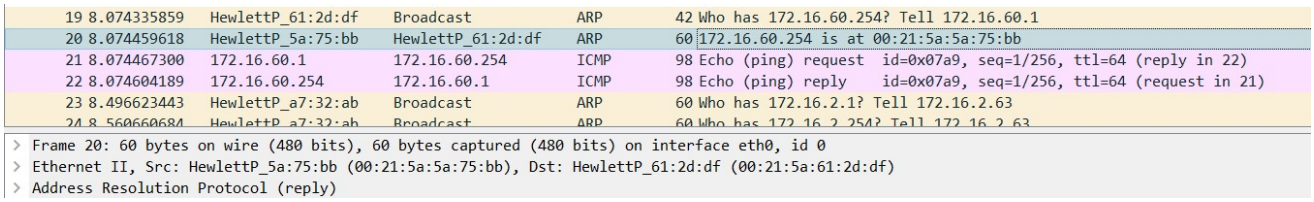
```
configure terminal
interface fastethernet 0/0
ip address 172.16.61.254 255.255.255.0
no shutdown
ip nat inside
exit
configure terminal
interface fastethernet 0/1
ip address 172.16.2.69 255.255.255.0
no shutdown
ip nat outside
exit
ip nat pool ovrld 172.16.2.69 172.16.2.69 prefix 24
ip nat inside source list 1 pool ovrld overload
access-list 1 permit 172.16.60.0 0.0.0.7
access-list 1 permit 172.16.61.0 0.0.0.7

ip route 0.0.0.0 0.0.0.0 172.16.2.254
ip route 172.16.60.0 255.255.255.0 172.16.61.253
end
```

7.8 Configurar servidor DNS

```
nano /etc/resolv.conf
search netlab.fe.up.pt
nameserver 172.16.2.1
```

8 Wireshark logs



```
    19 8.074335859   HewlettP_61:2d:df   Broadcast           ARP      42 Who has 172.16.60.254? Tell 172.16.60.1
    20 8.074459618   HewlettP_5a:75:bb   HewlettP_61:2d:df   ARP      60 172.16.60.254 is at 00:21:5a:5a:75:bb
    21 8.074467300   172.16.60.1         172.16.60.254       ICMP     98 Echo (ping) request  id=0x07a9, seq=1/256, ttl=64 (reply in 22)
    22 8.074604189   172.16.60.254       172.16.60.1         ICMP     98 Echo (ping) reply    id=0x07a9, seq=1/256, ttl=64 (request in 21)
    23 8.496623443   HewlettP_a7:32:ab   Broadcast           ARP      60 Who has 172.16.2.1? Tell 172.16.2.63
    24 8.560660684   HewlettP_a7:32:ab   Broadcast           ARP      60 Who has 172.16.2.254? Tell 172.16.2.63
> Frame 20: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0
> Ethernet II, Src: HewlettP_5a:75:bb (00:21:5a:5a:75:bb), Dst: HewlettP_61:2d:df (00:21:5a:61:2d:df)
> Address Resolution Protocol (reply)
```

Captura 1- Experiência 1. Pacotes ARP e ICMP do tux 62 para o tux 64 capturado no tux62

```
 8 10.874540884  172.16.60.1      172.16.60.254      ICMP     98 Echo (ping) request  id=0x1351, seq=1/256, ttl=64 (reply in 9)
 9 10.874703754  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x1351, seq=1/256, ttl=64 (request in 8)
10 11.900931926  172.16.60.1      172.16.60.254      ICMP     98 Echo (ping) request  id=0x1351, seq=2/512, ttl=64 (reply in 11)
11 11.901069234  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x1351, seq=2/512, ttl=64 (request in 10)
```

Captura 2-Experiência 2. Pacotes ICMP do tux62 para tux64 capturado no tux62

```
17 24.706652106  172.16.60.1      172.16.60.254      ICMP     98 Echo (ping) request  id=0x0c26, seq=1/256, ttl=64 (reply in 18)
18 24.706820842  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0c26, seq=1/256, ttl=64 (request in 17)
19 25.708780810  172.16.60.1      172.16.60.254      ICMP     98 Echo (ping) request  id=0x0c26, seq=2/512, ttl=64 (reply in 20)
20 25.708916092  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0c26, seq=2/512, ttl=64 (request in 19)
```

Captura 3-Experiência 2. Pacotes ICMP do tux62 para Broadcast (172.16.60.255) capturado no tux64

```
12 18.038530569  Cisco_7b:ce:82   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8002
13 19.390453876  172.16.60.1      172.16.60.255      ICMP     98 Echo (ping) request  id=0x0ca2, seq=1/256, ttl=64 (no response found!)
14 19.390493825  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0ca2, seq=1/256, ttl=64
15 20.048976064  Cisco_7b:ce:82   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8002
16 20.408905254  172.16.60.1      172.16.60.255      ICMP     98 Echo (ping) request  id=0x0ca2, seq=2/512, ttl=64 (no response found!)
17 20.408943108  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0ca2, seq=2/512, ttl=64
18 21.432897489  172.16.60.1      172.16.60.255      ICMP     98 Echo (ping) request  id=0x0ca2, seq=3/768, ttl=64 (no response found!)
19 21.432934086  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0ca2, seq=3/768, ttl=64
20 22.049739955  Cisco_7b:ce:82   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8002
21 22.456922409  172.16.60.1      172.16.60.255      ICMP     98 Echo (ping) request  id=0x0ca2, seq=4/1024, ttl=64 (no response found!)
22 22.456963476  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0ca2, seq=4/1024, ttl=64
23 23.480892574  172.16.60.1      172.16.60.255      ICMP     98 Echo (ping) request  id=0x0ca2, seq=5/1280, ttl=64 (no response found!)
```

Captura 4-Experiência 2. Pacotes ICMP do tux62 para Broadcast (172.16.60.255) capturado no tux62

```
 7 8.015401285   Cisco_7b:ce:81   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8001
 8 10.024493728  Cisco_7b:ce:81   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8001
 9 12.025179039  Cisco_7b:ce:81   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8001
10 14.030091278  Cisco_7b:ce:81   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8001
11 16.035054711  Cisco_7b:ce:81   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8001
12 17.763242567  Cisco_7b:ce:81   Cisco_7b:ce:81     LOOP     60 Reply
13 18.040007458  Cisco_7b:ce:81   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8001
14 20.049117641  Cisco_7b:ce:81   Spanning-tree-(for-… STP      60 Conf. Root = 32768/60/00:1e:14:7b:ce:80  Cost = 0  Port = 0x8001
```

Captura 5-Experiência 2. Pacotes ICMP do tux62 para Broadcast (172.16.60.255) capturado no tux63

```
17 24.706652106  172.16.60.1      172.16.60.254      ICMP     98 Echo (ping) request  id=0x0c26, seq=1/256, ttl=64 (reply in 18)
18 24.706820842  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0c26, seq=1/256, ttl=64 (request in 17)
19 25.708780810  172.16.60.1      172.16.60.254      ICMP     98 Echo (ping) request  id=0x0c26, seq=2/512, ttl=64 (reply in 20)
20 25.708916092  172.16.60.254    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x0c26, seq=2/512, ttl=64 (request in 19)
```

Captura 6-Experiência 3. Pacotes ICMP do tux62 para o tux64 capturado no tux62

```
51 38.428905244  172.16.60.1      172.16.61.253      ICMP     98 Echo (ping) request  id=0x1361, seq=4/1024, ttl=64 (reply in 52)
52 38.429037593  172.16.61.253    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x1361, seq=4/1024, ttl=64 (request in 51)
53 39.452903807  172.16.60.1      172.16.61.253      ICMP     98 Echo (ping) request  id=0x1361, seq=5/1280, ttl=64 (reply in 54)
54 39.453032803  172.16.61.253    172.16.60.1        ICMP     98 Echo (ping) reply    id=0x1361, seq=5/1280, ttl=64 (request in 53)
```

Captura 7-Experiência 3. Pacotes ICMP do tux62 para o tux64 capturado no tux62

75 50.122621036	Cisco_7b:ce:81	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/60/00:1e:14:7b:ce:80 Cost = 0 Port = 0x8001
76 50.946462244	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x136b, seq=1/256, ttl=64 (reply in 77)
77 50.946771081	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x136b, seq=1/256, ttl=63 (request in 76)
78 50.978572565	Cisco_7b:ce:81	CDP/VTP/DTP/PAgP/UD…	CDP	601	Device ID: gnu-sw6 Port ID: FastEthernet0/1
79 51.964932457	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x136b, seq=2/512, ttl=64 (reply in 80)
80 51.965178508	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x136b, seq=2/512, ttl=63 (request in 79)

Captura 8-Experiência 3. Pacotes ICMP do tux62 para o tux63 capturado no tux62

46 73.269283185	Netronix_71:73:ed	Broadcast	ARP	42	Who has 172.16.61.1? Tell 172.16.61.253
47 73.269403032	HewlettP_a7:32:ab	Netronix_71:73:ed	ARP	60	172.16.61.1 is at 00:22:64:a7:32:ab
48 73.269421261	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x13f3, seq=1/256, ttl=63 (reply in 49)
49 73.269539013	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x13f3, seq=1/256, ttl=64 (request in 48)

Captura 9-Experiência 3. Pacotes ARP e ICMP do tux62 para o tux63 capturado no tux64 na interface eth1

24 34.706914172	HewlettP_61:2d:df	Broadcast	ARP	60	Who has 172.16.60.254? Tell 172.16.60.1
25 34.706944832	HewlettP_5a:79:97	HewlettP_61:2d:df	ARP	42	172.16.60.254 is at 00:21:5a:5a:79:97
26 34.707071454	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x1b04, seq=1/256, ttl=64 (reply in 27)
27 34.707357663	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1b04, seq=1/256, ttl=63 (request in 26)
28 35.739012231	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x1b04, seq=2/512, ttl=64 (reply in 29)
29 35.739188580	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x1b04, seq=2/512, ttl=63 (request in 28)

Captura 10-Experiência 3. Pacotes ARP e ICMP do tux62 para o tux63 capturado no tux64 na interface eth0

8 7.987607833	172.16.60.1	172.16.61.253	ICMP	98	Echo (ping) request id=0x159f, seq=2/512, ttl=64 (reply in 9)
9 7.987735435	172.16.61.253	172.16.60.1	ICMP	98	Echo (ping) reply id=0x159f, seq=2/512, ttl=64 (request in 8)
10 9.011603453	172.16.60.1	172.16.61.253	ICMP	98	Echo (ping) request id=0x159f, seq=3/768, ttl=64 (reply in 11)
11 9.011764790	172.16.61.253	172.16.60.1	ICMP	98	Echo (ping) reply id=0x159f, seq=3/768, ttl=64 (request in 10)

Captura 11-Experiência 4. Pacotes ICMP do tux62 para o tux64 capturado no tux62

88 55.036904398	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x136b, seq=5/1280, ttl=64 (reply in 89)
89 55.037187744	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x136b, seq=5/1280, ttl=63 (request in 88)
90 56.060912249	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) request id=0x136b, seq=6/1536, ttl=64 (reply in 91)
91 56.061151246	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) reply id=0x136b, seq=6/1536, ttl=63 (request in 90)

Captura 12-Experiência 4. Pacotes ICMP do tux62 para o tux63 capturado no tux62

41 21.619624046	172.16.60.1	172.16.61.254	ICMP	98	Echo (ping) request id=0x15a7, seq=2/512, ttl=64 (reply in 42)
42 21.620402022	172.16.61.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x15a7, seq=2/512, ttl=254 (request in 41)
43 22.643623578	172.16.60.1	172.16.61.254	ICMP	98	Echo (ping) request id=0x15a7, seq=3/768, ttl=64 (reply in 44)
44 22.644413986	172.16.61.254	172.16.60.1	ICMP	98	Echo (ping) reply id=0x15a7, seq=3/768, ttl=254 (request in 43)

Captura 13-Experiência 4. Pacotes ICMP do tux62 para o router Rc capturado no tux62

8 6.840566039	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x0f84, seq=1/256, ttl=64 (reply in 10)
9 6.841251531	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
10 6.841609817	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x0f84, seq=1/256, ttl=63 (request in 8)
11 7.841697957	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x0f84, seq=2/512, ttl=64 (reply in 13)
12 7.842339519	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
13 7.842632852	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x0f84, seq=2/512, ttl=63 (request in 11)
14 8.019409173	Cisco_7c:8f:83	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/61/00:1e:14:7c:8f:80 Cost = 0 Port = 0x8003
15 8.654841723	172.16.61.1	172.16.2.1	DNS	81	Standard query 0xf21c A 1.debian.pool.ntp.org
16 8.654851571	172.16.61.1	172.16.2.1	DNS	81	Standard query 0xb829 AAAA 1.debian.pool.ntp.org
17 8.842733005	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x0f84, seq=3/768, ttl=64 (reply in 19)
18 8.843378757	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
19 8.843702611	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x0f84, seq=3/768, ttl=63 (request in 17)
20 9.843782370	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x0f84, seq=4/1024, ttl=64 (reply in 22)
21 9.844479176	172.16.61.254	172.16.61.1	ICMP	70	Redirect (Redirect for host)
22 9.844770275	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x0f84, seq=4/1024, ttl=63 (request in 20)
23 10.028290260	Cisco_7c:8f:83	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/61/00:1e:14:7c:8f:80 Cost = 0 Port = 0x8003

Captura 14-Experiência 4. Pacotes ICMP do tux63 para o tux62 capturado no tux63 sem rota e com redirecionamento

8 2.103958890	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x1147, seq=2/512, ttl=64 (reply in 9)
9 2.104224706	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x1147, seq=2/512, ttl=63 (request in 8)
10 3.127940361	172.16.61.1	172.16.60.1	ICMP	98	Echo (ping) request id=0x1147, seq=3/768, ttl=64 (reply in 11)
11 3.128186901	172.16.60.1	172.16.61.1	ICMP	98	Echo (ping) reply id=0x1147, seq=3/768, ttl=63 (request in 10)

Captura 15-Experiência 4. Pacotes ICMP do tux63 para o tux62 capturado no tux63 sem rota

7 4.259610500	172.16.60.1	172.16.2.254	ICMP	98	Echo (ping) request id=0x1844, seq=4/1024, ttl=64 (no response found!)
8 5.283615619	172.16.60.1	172.16.2.254	ICMP	98	Echo (ping) request id=0x1844, seq=5/1280, ttl=64 (no response found!)
9 6.010537603	Cisco_7c:8f:81	Spanning-tree-(for-…	STP	60	Conf. Root = 32768/60/00:1e:14:7c:8f:80 Cost = 0 Port = 0x8001
10 6.211569601	HewlettP_19:02:ba	HewlettP_c3:78:76	ARP	42	Who has 172.16.60.254? Tell 172.16.60.1
11 6.211684003	HewlettP_c3:78:76	HewlettP_19:02:ba	ARP	60	172.16.60.254 is at 00:21:5a:c3:78:76
12 6.307611519	172.16.60.1	172.16.2.254	ICMP	98	Echo (ping) request id=0x1844, seq=6/1536, ttl=64 (no response found!)
13 7.331607070	172.16.60.1	172.16.2.254	ICMP	98	Echo (ping) request id=0x1844, seq=7/1792, ttl=64 (no response found!)

Captura 16-Experiência 4. Pacotes ICMP do tux62 para o 172.16.2.254 capturado no tux62 (router sem NAT)

8 4.007728021	172.16.61.1	172.16.2.254	ICMP	98	Echo (ping) request id=0x142b, seq=3/768, ttl=64 (reply in 9)
9 4.008273492	172.16.2.254	172.16.61.1	ICMP	98	Echo (ping) reply id=0x142b, seq=3/768, ttl=63 (request in 8)
10 5.031735507	172.16.61.1	172.16.2.254	ICMP	98	Echo (ping) request id=0x142b, seq=4/1024, ttl=64 (reply in 11)
11 5.032281886	172.16.2.254	172.16.61.1	ICMP	98	Echo (ping) reply id=0x142b, seq=4/1024, ttl=63 (request in 10)

Captura 17-Experiência 4. Pacotes ICMP do tux62 para o 172.16.2.254 capturado no tux62 (router com NAT)

12 12.494413224	172.16.2.1	172.16.60.1	DNS	322	Standard query response 0x7981 A netlab2.fe.up.pt A 192.168.109.137 NS magoo.fe.up.pt NS cns1.fe.up.pt NS cns2.fe.up.pt NS ns2…
13 12.494607523	172.16.2.1	172.16.60.1	DNS	119	Standard query response 0xc28b AAAA netlab2.fe.up.pt SOA ns1.fe.up.pt
14 12.494832622	172.16.60.1	192.168.109.137	ICMP	98	Echo (ping) request id=0x4577, seq=1/256, ttl=64 (reply in 15)
15 12.496406080	192.168.109.137	172.16.60.1	ICMP	98	Echo (ping) reply id=0x4577, seq=1/256, ttl=61 (request in 14)
16 12.496525649	172.16.60.1	172.16.2.1	DNS	88	Standard query 0xf065 PTR 137.109.168.192.in-addr.arpa
17 12.498441610	172.16.2.1	172.16.60.1	DNS	88	Standard query response 0xf065 Server failure PTR 137.109.168.192.in-addr.arpa
18 12.498476530	172.16.60.1	193.136.28.10	DNS	88	Standard query 0xf065 PTR 137.109.168.192.in-addr.arpa
19 12.500777737	193.136.28.10	172.16.60.1	DNS	242	Standard query response 0xf065 PTR 137.109.168.192.in-addr.arpa PTR netlab2.fe.up.pt NS ns2.fe.up.pt NS ns1.fe.up.pt A 193.136…
20 13.495864961	172.16.60.1	192.168.109.137	ICMP	98	Echo (ping) request id=0x4577, seq=2/512, ttl=64 (reply in 21)
21 13.496926342	192.168.109.137	172.16.60.1	ICMP	98	Echo (ping) reply id=0x4577, seq=2/512, ttl=61 (request in 20)

Captura 18-Experiência 5. Pacotes DNS e ICMP do tux62 para o netlab2.fe.up.pt capturado no tux62

4 5.337127232	172.16.60.1	172.16.2.1	DNS	69	Standard query 0xba20 A ftp.up.pt
5 5.339275466	172.16.2.1	172.16.60.1	DNS	107	Standard query response 0xba20 A ftp.up.pt CNAME mirrors.up.pt A 193.137.29.15
6 5.339405998	172.16.60.1	193.137.29.15	TCP	74	45926 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2403284158 TSecr=0 WS=128
7 5.344461845	193.137.29.15	172.16.60.1	TCP	74	21 → 45926 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM=1 TSval=823528810 TSecr=2403284158 WS=128
8 5.344482449	172.16.60.1	193.137.29.15	TCP	66	45926 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2403284163 TSecr=823528810
9 5.349408741	193.137.29.15	172.16.60.1	FTP	139	Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
10 5.349418728	172.16.60.1	193.137.29.15	TCP	66	45926 → 21 [ACK] Seq=1 Ack=74 Win=64256 Len=0 TSval=2403284168 TSecr=823528814
11 5.350046597	193.137.29.15	172.16.60.1	FTP	141	Response: 220----------
12 5.350052394	172.16.60.1	193.137.29.15	TCP	66	45926 → 21 [ACK] Seq=1 Ack=149 Win=64256 Len=0 TSval=2403284169 TSecr=823528814
13 5.350753316	193.137.29.15	172.16.60.1	FTP	310	Response: 220-All connections and transfers are logged. The max number of connections is 200.
14 5.350759043	172.16.60.1	193.137.29.15	TCP	66	45926 → 21 [ACK] Seq=1 Ack=393 Win=64128 Len=0 TSval=2403284169 TSecr=823528814
15 5.350816941	172.16.60.1	193.137.29.15	FTP	82	Request: user anonymous
16 5.355228437	193.137.29.15	172.16.60.1	TCP	66	21 → 45926 [ACK] Seq=393 Ack=17 Win=65280 Len=0 TSval=823528820 TSecr=2403284169
17 5.355568911	193.137.29.15	172.16.60.1	FTP	100	Response: 331 Please specify the password.
18 5.355594543	172.16.60.1	193.137.29.15	FTP	77	Request: pass pass
19 5.358324201	193.137.29.15	172.16.60.1	TCP	66	21 → 45926 [ACK] Seq=427 Ack=28 Win=65280 Len=0 TSval=823528824 TSecr=2403284174
20 5.359702301	193.137.29.15	172.16.60.1	FTP	89	Response: 230 Login successful.
21 5.359728491	172.16.60.1	193.137.29.15	FTP	72	Request: pasv
22 5.362226208	193.137.29.15	172.16.60.1	TCP	66	21 → 45926 [ACK] Seq=450 Ack=34 Win=65280 Len=0 TSval=823528828 TSecr=2403284178
23 5.362858407	193.137.29.15	172.16.60.1	FTP	118	Response: 227 Entering Passive Mode (193,137,29,15,224,245).
24 5.362906039	172.16.60.1	193.137.29.15	TCP	74	42850 → 57589 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2403284182 TSecr=0 WS=128
25 5.365550352	193.137.29.15	172.16.60.1	TCP	74	57589 → 42850 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1380 SACK_PERM=1 TSval=823528831 TSecr=2403284182 WS=128
26 5.365563133	172.16.60.1	193.137.29.15	TCP	66	42850 → 57589 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2403284184 TSecr=823528831

Captura 19-Experiência 6. Ligação TCP/IP - "three-way handshake"

156 5.385523780	193.137.29.15	172.16.60.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr raspbian/snapshotindex.txt)
157 5.385530834	172.16.60.1	193.137.29.15	TCP	66	42850 → 57589 [ACK] Seq=1 Ack=108073 Win=2304 Len=0 TSval=2403284204 TSecr=823528850
158 5.385653265	193.137.29.15	172.16.60.1	FTP-DA…	1506	FTP Data: 1440 bytes (PASV) (retr raspbian/snapshotindex.txt)
159 5.385659760	172.16.60.1	193.137.29.15	TCP	66	42850 → 57589 [ACK] Seq=1 Ack=109513 Win=896 Len=0 TSval=2403284204 TSecr=823528851
160 5.388635329	172.16.60.1	193.137.29.15	TCP	66	[TCP Window Update] 42850 → 57589 [ACK] Seq=1 Ack=109513 Win=19200 Len=0 TSval=2403284207 TSecr=823528851
161 5.391275940	193.137.29.15	172.16.60.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr raspbian/snapshotindex.txt)
162 5.391391667	193.137.29.15	172.16.60.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr raspbian/snapshotindex.txt)

Captura 20-Experiência 6. Ligação TCP/IP - transferência de dados

604 5.496163309	193.137.29.15	172.16.60.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr raspbian/snapshotindex.txt)
605 5.496168198	172.16.60.1	193.137.29.15	TCP	66	42850 → 57589 [ACK] Seq=1 Ack=488409 Win=11520 Len=0 TSval=2403284315 TSecr=823528961
606 5.496329460	193.137.29.15	172.16.60.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr raspbian/snapshotindex.txt)
607 5.496447561	193.137.29.15	172.16.60.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr raspbian/snapshotindex.txt)
608 5.496452310	172.16.60.1	193.137.29.15	TCP	66	42850 → 57589 [ACK] Seq=1 Ack=491145 Win=9472 Len=0 TSval=2403284315 TSecr=823528961
609 5.496564614	193.137.29.15	172.16.60.1	FTP-DA…	1434	FTP Data: 1368 bytes (PASV) (retr raspbian/snapshotindex.txt)
610 5.496626912	193.137.29.15	172.16.60.1	FTP-DA…	795	FTP Data: 729 bytes (PASV) (retr raspbian/snapshotindex.txt)
611 5.496633757	172.16.60.1	193.137.29.15	TCP	66	42850 → 57589 [ACK] Seq=1 Ack=493243 Win=7424 Len=0 TSval=2403284315 TSecr=823528961
612 5.498730029	193.137.29.15	172.16.60.1	FTP	90	Response: 226 Transfer complete.
613 5.498736105	172.16.60.1	193.137.29.15	TCP	66	45926 → 21 [ACK] Seq=67 Ack=614 Win=64128 Len=0 TSval=2403284317 TSecr=823528964
614 5.514748092	172.16.60.1	193.137.29.15	FTP	72	Request: quit
615 5.517697609	193.137.29.15	172.16.60.1	FTP	80	Response: 221 Goodbye.
616 5.517730295	172.16.60.1	193.137.29.15	TCP	66	45926 → 21 [FIN, ACK] Seq=73 Ack=628 Win=64128 Len=0 TSval=2403284336 TSecr=823528983
617 5.517745171	172.16.60.1	193.137.29.15	TCP	66	42850 → 57589 [FIN, ACK] Seq=1 Ack=493243 Win=64128 Len=0 TSval=2403284336 TSecr=823528961
618 5.518494493	193.137.29.15	172.16.60.1	TCP	66	21 → 45926 [FIN, ACK] Seq=628 Ack=73 Win=65280 Len=0 TSval=823528983 TSecr=2403284333
619 5.518504620	172.16.60.1	193.137.29.15	TCP	66	45926 → 21 [ACK] Seq=74 Ack=629 Win=64128 Len=0 TSval=2403284337 TSecr=823528983
620 5.519311002	193.137.29.15	172.16.60.1	TCP	66	57589 → 42850 [ACK] Seq=493243 Ack=2 Win=65280 Len=0 TSval=823528985 TSecr=2403284336
621 5.520895900	193.137.29.15	172.16.60.1	TCP	66	21 → 45926 [ACK] Seq=629 Ack=74 Win=65280 Len=0 TSval=823528986 TSecr=2403284336
622 5.722409795	fe80::221:5aff:fe5a…	ff02::2	ICMPv6	70	Router Solicitation from 00:21:5a:5a:75:bb

Captura 21-Experiência 6. Ligação TCP/IP – fim da ligação

```
tomestacio12@DESKTOP-GSM03VJ:/mnt/c/Users/Tomas/Desktop/FEUP/RC/lab2/code$ gcc *.c -o download
tomestacio12@DESKTOP-GSM03VJ:/mnt/c/Users/Tomas/Desktop/FEUP/RC/lab2/code$ ./download ftp://:@ftp.up.pt/raspbian/snapshotindex.txt
host: ftp.up.pt
url path: raspbian/snapshotindex.txt
user: anonymous
password: pass
file name: snapshotindex.txt
host name: mirrors.up.pt
ip addr: 193.137.29.15
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
< user anonymous

331 Please specify the password.
< pass pass

230 Login successful.
< pasv

227 Entering Passive Mode (193,137,29,15,229,121).
IP: 193.137.29.15
Port Number: 58745
< retr raspbian/snapshotindex.txt

150 Opening BINARY mode data connection for raspbian/snapshotindex.txt (494843 bytes).
226 Transfer complete.
< quit

221 Goodbye.
```

Captura 22- Aplicação de download bem-sucedida.