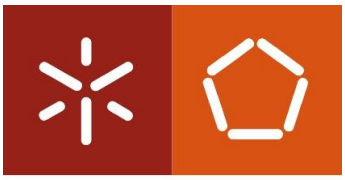

Universidade do Minho

Licenciatura em Engenharia Informática

Comunicações por Computador
TP2 - Implementação de um sistema DNS

Grupo 6.14

Gonçalo Pereira (A96849) José Moreira (A95522)
Santiago Domingues (A96886)

02/01/2023 - Ano Letivo 2022/2023

Índice

Lista de Figuras

Capítulo 1

Introdução

Em proposta pelos docentes da UC de **Comunicações por Computador**, desenvolveu-se e implementou-se um **sistema DNS**. Um *Domain Name System* é um serviço hierárquico responsável por associar domínios aos seus respetivos endereços, facilitando o acesso dos diversos utilizadores da rede. De uma forma mais fácil de compreender, um utilizador, quando digita "www.uminho.pt"no seu browser, está a forçar o mesmo a requisitar ao **DNS** que lhe retorne o **endereço IP** dessa mesma página à qual o utilizador quer ter acesso. O DNS recebe esse nome e, pelos seus métodos e, acedendo aos nomes guardados em bases de dados, obtém o endereço 193.137.9.114, que é devolvido ao "utilizador". Todo este processo permite que os utilizadores de uma rede e, de um modo geral, da internet, consigam aceder aos vários servidores sem que tenham que possuir, integralmente, os endereços IP dos mesmos. Apesar de parecer algo simples, o serviço DNS possui uma metodologia e um algoritmo complexos. De forma a conseguir concretizar todo o processo referido no guião, procurou-se, primeiramente, perceber integralmente todo o trabalho realizado pelo serviço DNS. Com isto, aprofundou-se o estudo em função de vários termos que foram mostrando importância na família do DNS: **domínio, servidor, endereço, autoritarivo, recursivo, root, iterativo, browser, caching**.

Em discussão, decidiu-se explicar os vários componentes do sistema e as várias decisões escolhidas em função dos vários temas apresentados no guião do projeto. Assim, em modo introdutório, serão abordados esses mesmos temas e todas as ponderações que levaram a que a arquitetura e o desenvolvimento do nosso projeto nascessem. Em primeiro lugar, toda a arquitetura dos componentes do nosso projeto foi esboçada de forma a que se conseguisse perceber os modos de interação entre eles e que função iriam exercer no serviço. Com isto, refere-se os **servidores primário, secundário e de resolução** e o **Cliente**, parâmetros de funcionamento e todos os ficheiros respetivos: log, configuração, entre outros. De uma forma óbvia, o esboço e desenvolvimento dos servidores também levou a que se discutisse a utilidade e modos de funcionamento dos **servidores de topo** e dos **servidores de domínio de topo**. Por outro lado e, sempre com o processo de desenvolvimento dos componentes em mente, desenvolveu-se os ficheiros usados pelo sistema (ficheiros já referidos: log, configuração, ou seja, ficheiros de input, etc.). Toda a escrita desses ficheiros teve por base os ficheiros exemplo apresentados no guião e, toda a leitura desses mesmos ficheiros interagiu diretamente com o desenvolvimento de funções em todo o nosso software. É também importante referir que os métodos aplicados nos parâmetros de comunicação do sistema obtiveram as devidas ponderação e análise e serão explicados de forma mais detalhada num momento futuro do nosso relatório, realçando sempre os diferentes pontos de abordagem, tais como: **protocolos, interações servidor-servidor e servidor-cliente, envio e receção de queries, modelo comunicacional do sistema, transferência de zona**, entre outros. De outro ponto

de vista, os sistemas de armazenamento de informação, ou seja, a **Cache**, mostra-se uma implementação importante no serviço, uma vez que garante o bom desempenho do mesmo. A resposta a queries, apesar de poder ser encontrada por meios recursivos ou iterativos entre servidores de DNS, pode-se encontrar em memória volátil e levar a que uma grande parte da complexidade da "pesquisa"por um endereço ou por informações relativas ao mesmo seja descartada. Como foi referido, todas as discussões em grupo que levaram a certas escolhas terão o seu devido fundamento e todo o projeto desenvolvido possuirá as devidas justificações, não só no relatório como também na documentação do código.

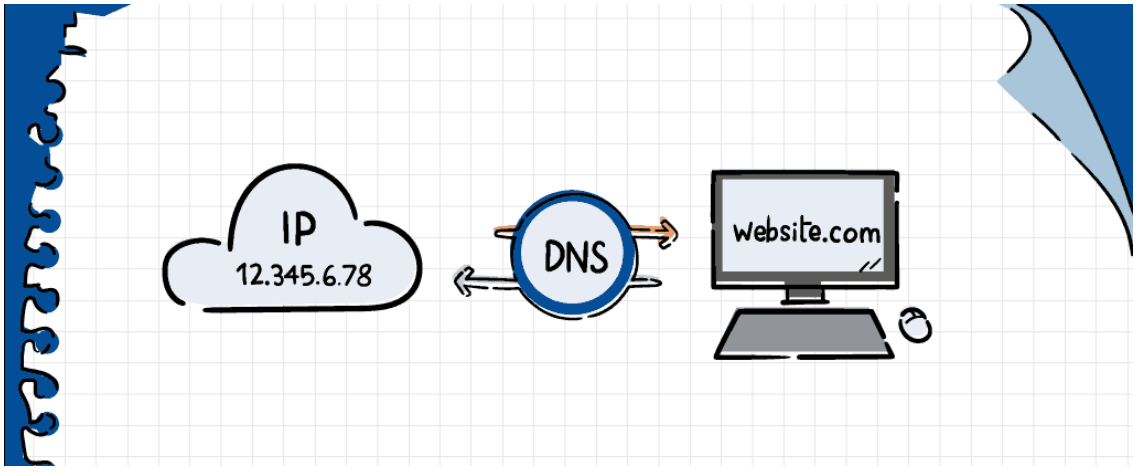

Figura 1.1: Exemplo ilustrativo do funcionamento de um DNS

Capítulo 2

Arquitetura do Sistema

2.1 Descrição do Sistema

O sistema desenvolvido, de forma a corresponder a todos os requisitos apresentados no guião do projeto, é composto, essencialmente, por dois elementos fundamentais que interagem entre si: **Cliente** e **Servidor**. De forma a modular o sistema por um algoritmo mais simples, cada um destes elementos é implementado por um componente de software diferente, ou seja, foram desenvolvidos dois componentes de software principais, na linguagem de programação **Java**, responsáveis, cada um deles, por executar, singularmente, cada um dos elementos referidos. Como é esperado, ambos os elementos apresentados contactam e interagem com diversos outros módulos, de forma a que o trabalho útil do sistema seja realizado corretamente.

Resumidamente, desenvolveu-se peças de software que permitem ao **Cliente** comunicar com todo o tipo de servidores e vice-versa. Foi também implementada a possibilidade dos servidores comunicarem entre si. Como sabemos, o objetivo fulcral do projeto é desenvolver um sistema **DNS** que funcione a partir de **queries** realizadas pelo **Cliente**, sendo as mesmas introduzidas no mesmo através do stdin. Ou seja, o objetivo deste componente do sistema é enviar uma **query** sobre um domínio ao qual quer ter acesso, de forma a obter a informação pretendida sobre o mesmo, ao mesmo tempo que toda essa informação se encontra disponível em componentes modulares presentes nos servidores (partindo do princípio que esse domínio é contemplado pela informação presente nos servidores e que a **query** enviada está estruturalmente correta).

Inicialmente, é do conhecimento que os servidores possuem um input que permite aos mesmos funcionarem de forma correta. Deste modo, o **Servidor Primário** possui, precisamente como input, um **ficheiro de configuração**, um **ficheiro de base de dados** relativo ao domínio abrangido pelo mesmo e um **ficheiro com a lista de servidores de topo**, enquanto que o **Servidor Secundário** e o **Servidor de Resolução** são inicializados com um **ficheiro de configuração** e um **ficheiro de servidores de topo**. Com isto, podemos perceber que a autoridade do **Servidor Primário** é superior à do **Servidor Secundário**, uma vez que o mesmo possui um ficheiro de base de dados, ao mesmo tempo que o outro não. De forma a colmatar este défice de informação por parte do **Servidor Secundário**, o mesmo possuirá uma cópia, em memória volátil, dessa mesma base de dados, que será fornecida pelo **Servidor Primário** a partir de um processo denominado **Transferência de Zona**, que será explicado numa fase posterior do presente relatório. Em semelhança entre os servidores, todos terão como output um **ficheiro de log**, onde será escrita toda a atividade dos mesmos. Por outro lado, todos estes mesmos servidores possuem um **ficheiro de log** comum, responsável por guardar a totalidade das ações ocorridas em todos os

módulos. Do ponto de vista dos **sistemas de Cache**, o **Servidor de Resolução** é o único que possui a capacidade de **caching**, mesmo que os vários servidores possuam a propriedade comum de guardar informação, através de estruturas idênticas, em memória volátil. Todo este sistema de armazenamento de memória permite ao mesmo responder de uma forma mais rápida e eficaz a todas as queries que o mesmo recebe. Como sugerido no guião, o grupo decidiu implementar os **servidores de topo** e os **servidores de domínio de topo** com o mesmo módulo que implementa todos os outros servidores. Este mesmo módulo, a partir dos **ficheiros de configuração** e dos domínios abrangidos, consegue distinguir o tipo de servidor que o mesmo está a representar.

2.2 Software

Foram desenvolvidas peças de software que permitem que o nosso projeto satisfaça os requisitos funcionais pedidos. Vários ficheiros **.java** interagem para que os dois elementos base do programa consigam corresponder às expectativas planeadas. Inicialmente, os ficheiros **"Client.java"** e **"Server.java"** representam, algoritmicamente, o trabalho executado pelo **Cliente** e pelos diversos tipos de **Servidor**, respetivamente. No entanto, estes ficheiros recorrem a outros para que as várias tarefas necessárias sejam realizadas, ou seja, **parsing**, **procura de queries** em base de dados, **comunicação TCP**, entre outros. Porém, o ficheiro responsável pelo papel dos servidores, de forma a adaptar o trabalho desenvolvido a situações de **Multithreading**, recprre a outro ficheiro denominado **"MultiThreadedServer.java"**.

Respetivamente ao **Cliente**, o mesmo possui código que permite transformar o input em mensagens estruturalmente bem definidas e, com isto, enviá-las para os respeitvos servidores a contactar, através do **protocolo UDP**, a explicar numa fase tardia do trabalho. Por outro lado, os servidores recorrem aos ficheiros **"Parsing.java"** e **"ConvertVar.java"** de forma a conseguirem ler os ficheiros de input já descritos, guardando-os em variáveis e, com isto, manipulando-os da forma pretendida. Em todos os servidores, os ficheiros de configuração são guardados em **Map<String,Map<String,List<String>>>**, enquanto que os ficheiros relativos a servidores de topo são guardados em **List<String>**. De uma forma parecida, o conteúdo presente na base de dados à qual o **Servidor Primário** tem acesso é guardado no mesmo e no **Servidor Secundário** em variáveis **List<String[]>**, denominadas de **Cache**, apesar de ser obtido de maneira diferente : o **Servidor Primário** recebe o mesmo a partir da leitura de um ficheiro, enquanto que o **Servidor Secundário** o recebe a partir de uma **comunicação TCP** entre o mesmo e o **Servidor Primário**, denominada **Transferência de Zona**, recorrendo aos ficheiros **"TCPsp.java"** e **"TCPss.java"**. De uma forma idêntica, os **servidores de topo** guardam a informação necessária em **List<String[]>**, ou seja, todos os servidores usam a mesma estrutura para guardarem o conteúdo importante ao seu funcionamento, desde resposta a queries a endereços IP. Toda esta capacidade de armazenamento de **Cache** e a propriedade de **caching** (neste caso, apenas disponível no **Servidor de Resolução**) são suportadas pelo módulo **"Cache.java"**. De modo a permitir que a **transferência de zona** funcione num ambiente **multithreading**, foi desenvolvido um módulo, **"TCPspMT"**, que é responsável pela criação de uma **thread** onde será executada a **transferência de zona** por parte do **Servidor Primário**.

Como referido, nos vários tipos de servidor, há a necessidade de se estabelecer ligação com o **Cliente**, de forma a que as **queries** sejam devidamente transportadas, lidas e relacionadas com a informação relevante presente nas estruturas responsáveis por guardar a informação das bases de dados. Esta mesma ligação é efetuada através de um **protocolo UDP** e recorre ao ficheiro **"UDPConnection.java"**. De uma forma simples, todas as **queries** transportadas entre os diversos módulos do sistema são criadas com a ajuda de **"Converter.java"**, uma vez que o mesmo tem a capacidade de criar uma mensagem com uma sintaxe pré-definida. Em simultâneo, todos os acontecimentos são registados devidamente nos ficheiros de output dos servidores, ou seja, os **ficheiros log**. Todas estas ocorrências recorrem a **"LogRegister.java"**, que conta com algorit-

mos de controlo de concorrência, uma vez que existe a escrita simultânea dos vários servidores num ficheiro comum de log.

Uma vez que há a necessidade do projeto possuir um caráter contínuo, isto é, ambos os servidores possuírem uma execução contínua até que sejam forçados a parar, o código dos mesmos possui um **ciclo while**, onde acontece toda a interação com os clientes. Ou seja, toda o processo de leitura de ficheiros e criação de estruturas de armazenamento é efetuado antes da inserção deste mesmo ciclo. Com isto, é possível garantir que as estruturas são criadas apenas uma vez (podendo, numa fase posterior, ser modificadas), mas que, por outro lado, há a satisfação contínua dos vários pedidos efetuados pelos clientes, ou seja, a capacidade de receberem e responderem a queries até que os servidores sejam forçados a parar. Da mesma forma, para que os servidores não bloqueiem aquando a espera por processos de **Transferência de Zona**, foram desenvolvidos os módulos **"TransferZoneMTP.java"** e **"TransferZoneMTS.java"** que sáo responsáveis por realizar os processos referidos em **"background"**, ou seja, recorrendo a outras **threads**.

Capítulo 3

Modelo de Informação

3.1 Ficheiros

Todos os ficheiros, tanto de input como de output, possuem sintaxe e semântica previamente definidas. Deste modo, toda a composição estrutural e lógica dos mesmos possui uma explicação devidamente justificada. Assim, surge a necessidade de mostrar, detalhadamente, a composição de cada um dos ficheiros e, também, a forma como os mesmo são lidos, no caso dos ficheiros de input, ou a forma como os mesmos são criados e preenchidos, no caso dos ficheiros de output.

3.1.1 Ficheiro de Configuração

Este ficheiro, comum a todos os servidores, possui uma estrutura específica, caracterizada pela seguinte sequência estrutural: **parâmetro**, **tipo do valor** e o **valor associado ao parâmetro**. Os tipos de valores acima referidos podem ser:

DB: indica o **ficheiro da base de dados** que contém a informação do domínio indicado no parâmetro;

SP: indica o endereço IP[:porta] do **Servidor Primário** do domínio indicado no parâmetro;

SS: indica o endereço IP[:porta] de um **Servidor Secundário** do domínio indicado no parâmetro, que passa a ter autorização para pedir a transmissão da informação da base de dados (**transferência de zona**);

DD: indica o endereço IP[:porta] de um **Servidor de Resolução** (ainda não implementado), de um **Servidor Secundário** ou de um **Servidor Primário** do domínio, por defeito, indicado no parâmetro. Se o servidor em questão se tratar de um **Servidor de Resolução**, o mesmo usa este parâmetro para indicar quais os domínios para os quais devem contactar diretamente os servidores indicados se receberem queries sobre os mesmos, caso a resposta não se encontre em cache, em vez de contactarem um dos **Servidores de Topo**. Se o servidor assumir a função de **Servidor Primário** ou **Servidor Secundário**, o mesmo usa este parâmetro para indicar os únicos domínios para os quais podem responder e, neste caso, o parâmetro irá servir como método restritivo do funcionamento dos **Servidores Primário e Secundário**, de modo a responderem apenas a queries sobre os domínios indicados no parâmetro referido;

ST: indica o ficheiro com a lista dos **Servidores de Topo**. De notar que o parâmetro deve ser igual a "root";

LG: indica o **ficheiro de log** que o servidor vai criar e utilizar para registar a atividade do servidor associada ao domínio indicado no parâmetro. É importante referir que só podem ser indicados domínios para o qual o servidor é **Servidor Primário** ou **Servidor Secundário** e tem de existir, pelo menos, uma entrada a referir um **ficheiro de log** para toda a atividade que não seja diretamente referente aos domínios especificados noutras entradas "LG", sendo que neste caso o parâmetro deve ser igual a "all".

3.1.2 Ficheiro de Base de Dados

Este ficheiro apenas diz respeito aos **Servidores Primários**, visto que são aqueles que possuem uma base de dados e não uma cópia da mesma, como acontece com os **Servidores Secundários**. São delimitados pela seguinte sintaxe: **parâmetro**, **tipo do valor**, **valor**, **tempo de validade** (**TTL**), isto é, o tempo máximo em segundos que os dados podem existir numa cache de um servidor, e **prioridade**, representada por um valor inteiro menor que 256, que define uma ordem de prioridade de vários valores associados ao mesmo parâmetro. Analogamente ao **ficheiro de configuração**, existe uma panóplia de possíveis resultados para os **tipos de valores**, sendo estes:

SOASP: indica o nome completo do **Servidor Primário** do domínio indicado no parâmetro;

SOAADMIN: indica o endereço de e-mail completo do administrador do domínio indicado no parâmetro;

SOASERIAL: indica o número de série da base de dados do **Servidor Primário** do domínio indicado no parâmetro;

SOAREFRESH: indica o intervalo temporal, em segundos, para um **Servidor Secundário** perguntar ao **Servidor Primário** do domínio indicado no parâmetro qual o número de série da base de dados dessa zona;

SOARETRY: indica o intervalo temporal para um **Servidor Secundário** voltar a perguntar ao **Servidor Primário** do domínio indicado no parâmetro qual o número de série da base de dados dessa zona, após um timeout;

SOAEXPIRE: indica o intervalo temporal para um **Servidor Secundário** deixar de considerar a sua réplica da base de dados da zona indicada no parâmetro como válida, deixando de responder a perguntas sobre a zona em causa, mesmo que continue a tentar contactar o **Servidor Primário** respetivo;

NS: indica o nome de um servidor que é autoritativo para o domínio indicado no parâmetro, ou seja, o nome do **Servidor Primário** ou de um dos **Servidores Secundários** do domínio;

A: indica o endereço IPv4 de um host/servidor indicado no parâmetro como nome;

CNAME: indica um nome canónico associado ao nome indicado no parâmetro;

MX: indica o nome de um servidor de e-mail para o domínio indicado no parâmetro;

PTR: indica o nome de um servidor ou host que usa o endereço IPv4 indicado no parâmetro.

3.1.3 Ficheiro de Log

É o único ficheiro de output dos servidores implementados. Este ficheiro contém todos os registos da atividade dos servidores em questão. No caso de não existir um **ficheiro de log**, este deve ser criado e nele registado todas as informações resultantes da execução dos servidores. Tal como os ficheiros anteriores, o **ficheiro de log** segue uma estrutura predefinida e específica, sendo reconhecida pelos seguintes campos: **etiqueta temporal**, **tipo de entrada**, **endereço IP**[**:porta**] e **dados da entrada**. No campo **tipo de entrada**, estes são os tipos previstos:

QR/QE: indica que foi recebida/enviada uma **query** do/para o endereço indicado;

RP/RR: indica que foi enviada/recebida uma resposta a uma **query** para o/do endereço indicado;

ZT: indica que foi iniciado e concluído, corretamente, um processo de transferência de zona;

EV: indica que foi detetado um evento/atividade interna no componente;

ER: indica que foi recebido um **PDU** do endereço indicado que não foi possível descodificar corretamente;

EZ: indica que foi detetado um erro num processo de transferência de zona que não foi concluída corretamente;

FL: indica que foi detetado um erro no funcionamento interno do componente;

TO: indica que foi detetado um timeout na interação com o servidor no endereço indicado;

SP: indica que a execução do componente foi parada;

ST: indica que a execução do componente foi iniciada.

3.2 Parsing e escrita

Uma vez já referido, o **Parsing** dos ficheiros revelou-se uma fase muito importante do projeto, uma vez que é o mesmo que define como é que a informação vai ser guardada e, posteriormente, acedida. De acordo com a sintaxe explicada na secção anterior, todos os ficheiros de input padecem de um processo de leitura efetuado pelas várias funções do ficheiro **"Parsing.java"**. Por outro lado, tratando-se de output, todos os ficheiros de log respetivos a cada servidor são devidamente criados (caso não existam) e, posteriormente, preenchidos.

3.2.1 Parsing do Ficheiro de Configuração

O parsing do **ficheiro de configuração** dos servidores é efetuado numa fase de vida precoce dos mesmos, sendo este o primeiro a ser lido. A leitura do mesmo é efetuada através da função **configsParse(String file_path)**, do módulo **"Parsing.java"**. Esta mesma função, tendo como argumento a path do ficheiro, tem a capacidade de guardar toda a informação armazenada no ficheiro de configuração numa variável **Map<String,Map<String,List<String>>>**. Com isto, há uma facilidade de acesso à informação, sendo, a primeira key, o **parâmetro** e, a segunda key, o **tipo do valor**.

3.2.2 Parsing do Ficheiro de Base de Dados

O **ficheiro de base de dados** representa parte do input apenas dos **Servidores Primários**, ou seja, todos os outros servidores não possuem o processo de **parsing** deste mesmo ficheiro. A leitura do mesmo é realizada na totalidade pela função **dbParse(List<String>stringList)**, presente em **"Parsing.java"**. Esta função recebe, como argumento, as várias linhas do ficheiro de base de dados, sendo cada uma representada por uma **String** e transforma-as numa variável do tipo **List<String[]>**, como já referido, preenchendo assim o conteúdo da cache do mesmo. Inicialmente, esta mesma função tinha como argumento a path do ficheiro, porém, uma vez que os **Servidores Secundários** têm a necessidade de transformar o conteúdo que recebem dos **Servidores Primários** numa variável idêntica à que os mesmos usam, houve uma pequena alteração de forma a que a mesma função possa ser utilizada em ambos os casos, havendo, portanto, reutilização útil do código desenvolvido.

3.2.3 Parsing do Ficheiro de Servidores de Topo

Como input dos vários servidores (à exceção, como é óbvio, dos **Servidores de Topo**), o **ficheiro de servidores de topo** é o de mais fácil leitura e armazenamento. O **parsing** deste ficheiro é realizado pela função **topParsing(String file_path)** do objeto **"Parsing.java**. Esta mesma função recebe, como argumento, a path do ficheiro e retorna uma **List<String>**.

3.2.4 Criação/Escrita no Ficheiro de Log

Diferente dos ficheiros anteriores, o **ficheiro de log** representa o output de todos os servidores em atividade no serviço. Com isto, não há a necessidade de ler um ficheiro, mas de escrever num ficheiro, caso exista, ou criar um ficheiro e preenchê-lo, caso contrário. Este mesmo ficheiro, sendo responsável pelo registo das várias atividades do sistema, é escrito no fragmento modular **"LogRegister.java"** através da função **addRegist(String type, String address, String message, String allPath)**. Todos os argumentos que a mesma recebe referem-se a campos constituintes dos registos de acordo com a estrutura dos **ficheiros de log** já referida. Por outro lado, sabemos que todos os ficheiros possuem um **ficheiro de log** comum. Com isto, há uma necessidade óbvia de controlo de concorrência, uma vez que há a possibilidade dos vários servidores agirem simultaneamente e registarem as suas atividades nesse mesmo ficheiro, também de uma forma simultânea. Este controlo simples de concorrência é realizado através do uso da interface **Lock**.

3.3 Situações de erro

Todas as situações de inicialização dos servidores necessitam de grande atenção e precisão por parte do administrador dos mesmos, ou seja, o arranque dos servidores tem que ser feito segundo parâmetros corretos escritos no terminal, mesmo que o software esteja pronto para reportar situações em que o número de argumentos atribuídos é incorreto. Por outro lado, o conteúdo dos **ficheiros de input** é, maioritariamente, da responsabilidade de quem os desenvolve, uma vez que o serviço não possui uma capacidade muito desenvolvida de os avaliar. Porém, todos os servidores possuem métodos simples de verificação dos campos, uma vez que a estrutura dos ficheiros encontra-se previamente definida.

3.4 PDU/mensagens DNS

Todas as queries/respostas a queries são mensagens DNS que envolvem uma estrutura bem definida, a qual é seguida e respeitada ao longo de todo o código. Estas mesmas mensagens, através do encapsulamento pelo protocolo UDP, permitem que o **Cliente** e o **Servidor Primário/Servidor Secundário** comuniquem de uma forma simples e eficaz. Apesar de ser sugerido, nenhuma destas mensagens apresenta codificação binária, uma vez que o projeto apenas sustenta o modo **debug**. Porém, numa fase futura do trabalho prático, poderá haver a necessidade de implementação de um sistema eficaz de codificação, o que levará a que todos os métodos e processos necessários possuam a devida ponderação e discussão coletiva.

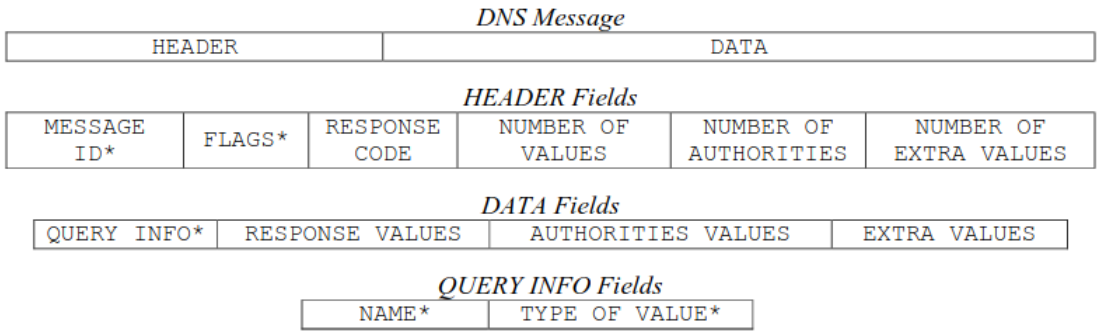

Figura 3.1: Estrutura de uma PDU/mensagem DNS

Como podemos observar através da **Figura 3.1**, as mensagens DNS utilizadas possuem uma sintaxe muito objetiva. Desta forma, podemos dividir a mensagem em dois campos principais: **Header** e **Data**. Resumidamente, o **Cliente**, ao enviar uma query a um servidor, faz o respetivo envio com o campo **Data** praticamente vazio, uma vez que é precisamente essa informação que o mesmo pretende obter. Apenas os campos do parâmetro **Query Info** são preenchidos, de forma a que o servidor recetor possa perceber sob que tipo de domínio (ou sob que tipo de parâmetro) atuar e de que forma. Por outro lado, o servidor, ao receber e processar a query, faz o respetivo preenchimento desse mesmo campo (caso possua a devida informação para tal) e altera os campos do **Header** de forma a fornecer informações básicas sobre a informação presente no resto da mensagem.

3.4.1 Header

O cabeçalho, sendo a fração informativa da mensagem, possui uma composição muito simples:

Message ID: identificador da mensagem, a ser usado para relacionar as respostas recebidas com a query original (este campo é gerado aleatoriamente pelo **Cliente** ou pelo servidor que recebe a query, sendo um número inteiro entre 1 e 65535);

Flags: este campo suporta as flags Q, que indica que a mensagem é uma query e A, que indica que a resposta já sofreu transformação por parte de um servidor;

Response Code: este campo indica o código de erro em resposta a uma query. Este mesmo campo suporta os valores 0, que indica que a query acabou de ser enviada pelo **Cliente** ou que o

Servidor encontrou conteúdo que responde diretamente à query; 1, que indica que o **Servidor** recetor ainda não encontrou informação de resposta, mas sabe que essa informação poderá existir num outro servidor; 2, que mostra que não há informação no serviço que responda à **query** e 3, que assinala algum erro na composição da mensagem;

Number of Values: número inteiro que indica o número de entradas relevantes que respondem diretamente à query e que fazem parte da lista de entradas incluídas no campo **Response Values**. Mais uma vez, na query enviada pelo **Cliente**, este campo possui o valor 0;

Number of Authorities: indica o número de entradas que identificam os servidores autoritarivos para o domínio incluído no **Result Values**. À semelhança dos campos anteriores, este campo possui, inicialmente, o valor 0;

Number of Extra Values: este campo fornece o número de entradas com informação adicional relacionada com os resultados da query ou com os servidores da lista de autoridades tendo, inicialmente, o valor 0;

3.4.2 Data

Por outro lado, o campo **Data** das mensagens DNS possui toda a informação possível relativa a um domínio. Todos os parâmetros deste campo devem ser devidamente preenchidos pelo fragmento de software responsável pela resposta às queries, ou seja, por um servidor. À semelhança do **Header**, há vários campos que constituem esta porção da mensagem:

Query Info: este parâmetro possui dois campos: **Name** e **Type of Value**. O primeiro é responsável pela informação da query, entre os quais o nome do domínio, enquanto que o segundo diz respeito ao tipo de valor associado ao parâmetro;

Response Values: campo preenchido pela lista das entradas que fazem match no **Name** e no **Type of Value** incluídos na cache ou na base de dados do servidor autoritarivo. Cada uma destas entradas deve ter a informação completa tal como é definida da base de dados do **Servidor Primário** do domínio referente ao **Name**;

Authorities Values: lista das entradas que fazem match com o **Name** e com o tipo de valor igual a **NS** incluídos na cache ou na base de dados do servidor autoritarivo. Cada uma das entradas deve conter a informação da mesma forma como é definida na base de dados do **Servidor Primário** do domínio indicado por **Name**;

Extra Values: lista de entradas do tipo A, mais uma vez, presentes na cache ou na base de dados do servidor autoritarivo, que fazem match no parâmetro com todos os valores no campo **Response Values** e no campo **Authorities Values**. Assim, é facilitado o serviço, uma vez que, com isto, o **Cliente** ou o servidor que recebe a resposta não tem que fazer novas queries para saber os endereços IP dos parâmetros que vêm como valores nos outros dois campos. À semelhança dos campos anteriores, cada entrada deve ter a informação completa tal como é definida na base de dados DNS do **Servidor Primário** do domínio referente ao **Name**.

Capítulo 4

Modelo Comunicacional do Sistema

4.1 Comunicação Cliente-Servidor / Servidor-Servidor : UDP

O protocolo de comunicação **UDP** tem como principal objetivo permitir a comunicação entre um **Cliente** e um **Servidor**, visto ser um protocolo indicado para fluxos de dados em tempo real e tendo a vantagem de não perder tempo nem na criação nem no encerramento de conexões.

O **Cliente** inicia a comunicação, convertendo a informação da estrutura da mensagem **DNS**, especificada previamente neste documento, numa **String**, sendo, numa fase posterior, convertida num conjunto de bytes e armazenada num array específico (**byte**[]). Do lado do **Cliente**, há a abertura de um **DatagramSocket** que irá servir como porta de comunicação entre o mesmo e o **Servidor**. De seguida, toda a informação da mensagem é empacotada num **DatagramPacket**, constituído pelo **buffer** que contém a mensagem convertida em bytes, o seu tamanho, o endereço do servidor e a respetiva porta. Finda a concepção da mensagem, esta é enviada através do **DatagramSocket** criado anteriormente, ficando o **Cliente** à espera da resposta do servidor. Do outro lado, no servidor, que se deve encontrar já em execução e com os **ficheiros de input** devidamente lidos e armazenados, é inicializado um **DatagramSocket** que permite ao servidor comunicar com o **Cliente**. Na criação deste elemento de comunicação, é fornecida uma porta, predefinida pelo grupo, consoante o tipo de servidor com o qual se quer estabelecer ligação. Similarmente ao funcionamento do lado do **Cliente**, no servidor, é criado um **DatagramPacket** que armazena os dados enviados pelo **Cliente** através do **DatagramSocket**. Após esse processo, a informação contida no **DatagramPacket** é convertida numa **String**, de modo a ser possível executar o passo seguinte, a procura da informação na **Cache** dos servidores. A pesquisa na **Cache** retorna uma **String** com a informação pretendida ou com o reencaminhamento para outro **Servidor** de forma a encontrar a resolução da **query** do **Cliente**, sendo este processo repetido até que o valor do **"Responde Code"** seja diferente de 1. Posto isto, é criado um **DatagramPacket** para armazenar a resposta do servidor e, consequentemente, ser enviada através do **DatagramSocket**. Logo após o envio da resposta, o servidor encerra o **DatagramSocket** destinado ao envio da mensagem. Antes da finalização da comunicação, o **Cliente** recebe a resposta do servidor, armazena-a num **DatagramPacket** e faz a sua conversão para uma **String**, fechando o **DatagramSocket** e dando como encerrada a comunicação com o servidor. O resultado é então impresso no **stdout** do **Cliente**.

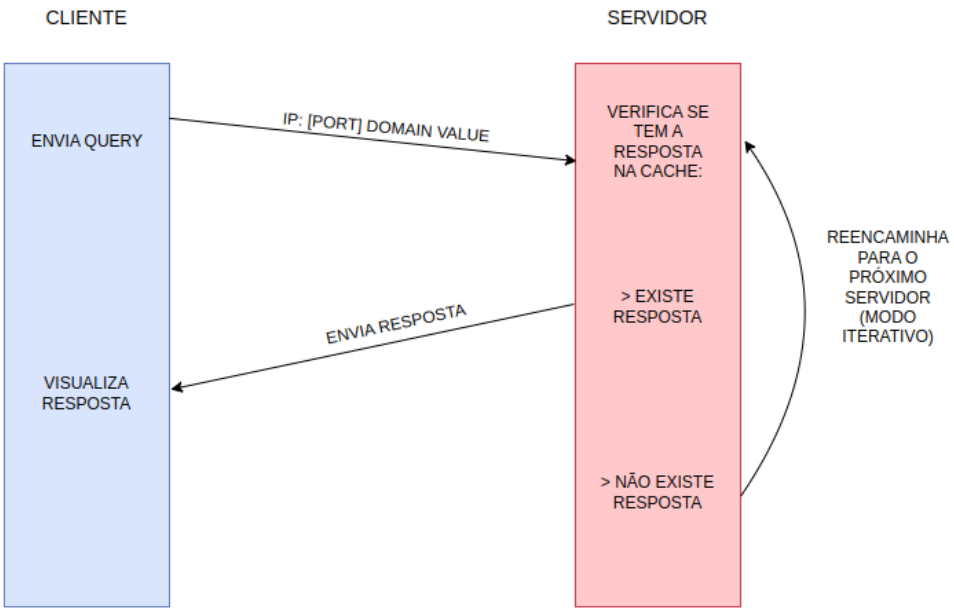

Figura 4.1: Comunicação UDP

4.2 Comunicação Servidor-Servidor : TCP

O protocolo de comunicação **TCP** tem como principal objetivo, neste projeto, conectar o **Servidor Primário** ao **Servidor Secundário**. Toda esta conexão permite que uma das tarefas mais importantes do projeto seja realizada: a **Transferência de Zona**. O processo de **Transferência de Zona** é fundamental, uma vez que permite ao **Servidor Secundário** possuir informação atualizada sobre o domínio sob o qual opera. Do lado do **Servidor Primário**, a **Transferência de Zona** é efetuada em processo **multithreading**, enquanto que, no **Servidor Secundário**, este processo ocorre na inicialização do mesmo e não volta a acontecer até que o mesmo seja reiniciado.

Inicialmente, o **Servidor Primário** inicializa uma nova **thread** apenas para a realização de **transferências de zona**. Neste **thread**, será inicializado um **ServerSocket** de forma a que consiga aceitar todas as tentativas de conexão por parte dos **Servidores Secundários**, executando-as individualmente em **threads** distintas. Após esta fase, o **Servidor Secundário** cria um **Socket** com os parâmetros correspondentes ao **IP** do **Servidor Primário** e à sua porta de atendimento. Posto isto, o **Servidor Secundário** inicia o processo de **transferência de zona**, isto é, o **Servidor Secundário** envia uma mensagem ao Servidor Primário com o nome do domínio sobre o qual quer receber dados. De seguida, o **Servidor Primário** verifica a autoridade do servidor que lhe enviou a mensagem, ou seja, averigua se o mesmo possui caráter de **Servidor Secundário** para o domínio pedido e, em caso positivo, envia-lhe o número de entradas do ficheiro de base de dados. Em caso negativo, é enviada uma mensagem "end"para que o **Servidor Secundário** termine a sua tentativa de transferência. Com isto, é garantida uma parte da segurança relativa a este tipo de comunicação, o que se mostra um aspeto positivo da implementação. Após esta fase e, partindo do princípio que o **Servidor Secundário** possui autoridade para o domínio requirido, o mesmo, ao receber o número de entradas do ficheiro de base de dados, responde, enviando exatamente a

mesma mensagem, dizendo, portanto, que aceita receber esse número de entradas. O **Servidor Primário**, ao ser informado com a confirmação, traduz o conteúdo da base de dados para formato de texto, ou seja, **String**. Posteriormente, envia o total da informação, enquanto que o **Servidor Secundário** se responsabiliza por recebê-la e convertê-la, estruturalmente, às variáveis definidas. Todas as linhas são enviadas apenas numa **String**, ou seja, toda a informação é enviada apenas numa mensagem. Após a receção da informação pretendida, o **Servidor Secundário** termina o processo de **transferência de zona** conseguindo com sucesso atualizar a sua cópia da base de dados do **Servidor Primário**.

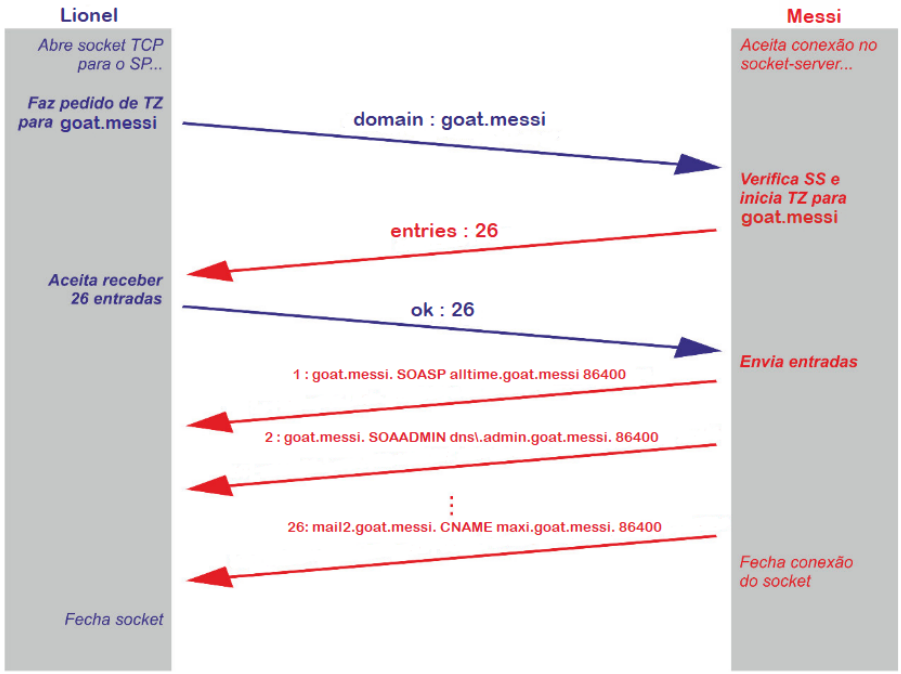

Figura 4.2: Transferência de Zona

Capítulo 5

Execução do código

Este capítulo serve como uma espécie de manual à utilização do nosso programa, ou seja, os argumentos de arranque dos vários executáveis serão detalhadamente apresentados e justificados.

De forma a que o **Cliente** seja executado corretamente, há a necessidade do mesmo receber três argumentos diferentes, ou seja, a inicialização do mesmo, num terminal, exige a escrita de um comando do estilo **"java Client IP:port name type_of_value"**. O primeiro argumento, **IP:port** refere-se ao IP e, opcionalmente, à porta que o mesmo entrará em contacto de forma a conseguir conectar-se ao respetivo servidor. Por outro lado, o argumento **name** representa a informação da query a enviar, podendo ser o nome do domínio sobre o qual o **Cliente** quer obter informação. Em último lugar, o **type_of_value** expõe o tipo de valor associado ao parâmetro. Qualquer valor não contemplado pela informação presente nos servidores é aceite como input, nos último dois argumentos, apesar de, como é esperado, apenas resultará numa mensagem DNS de erro. Por outro lado, a escrita incorreta do IP proporcionará uma situação de erro não controlável, em que o **Cliente** não será inicializado, uma vez que estará a tentar aceder a um **IP** não contemplado pelos servidores.

Do lado dos servidores, há a necessidade de escrita dos mesmos argumentos, ou seja, todo o tipo de servidores é inicializado exatamente da mesma forma, tendo o software a capacidade de distinguir a função dos mesmos numa fase posterior à leitura dos ficheiros. O comando a escrever no terminal terá o aspeto **"java Server config_path domain"**. O primeiro argumento refere-se, como o nome sugere, à path para o **ficheiro de configuração** do respetivo servidor e o segundo argumento explana o nome do domínio para o qual o respetivo servidor apresenta papel de **Servidor Primário** ou **Servidor Secundário**. Por outro lado, este campo, em relação aos **Servidores de Topo**, é introduzido com o valor ".". A introdução errada de um destes argumentos provoca a interrupção involuntária do programa que diz respeito ao **Servidor**, uma vez que o mesmo terá problemas de acesso às estruturas, na fase de **parsing**. Com isto, é importante que o usuário, ou seja, o administrador dos servidores, não cometa algum tipo de erro na inicialização dos mesmos.

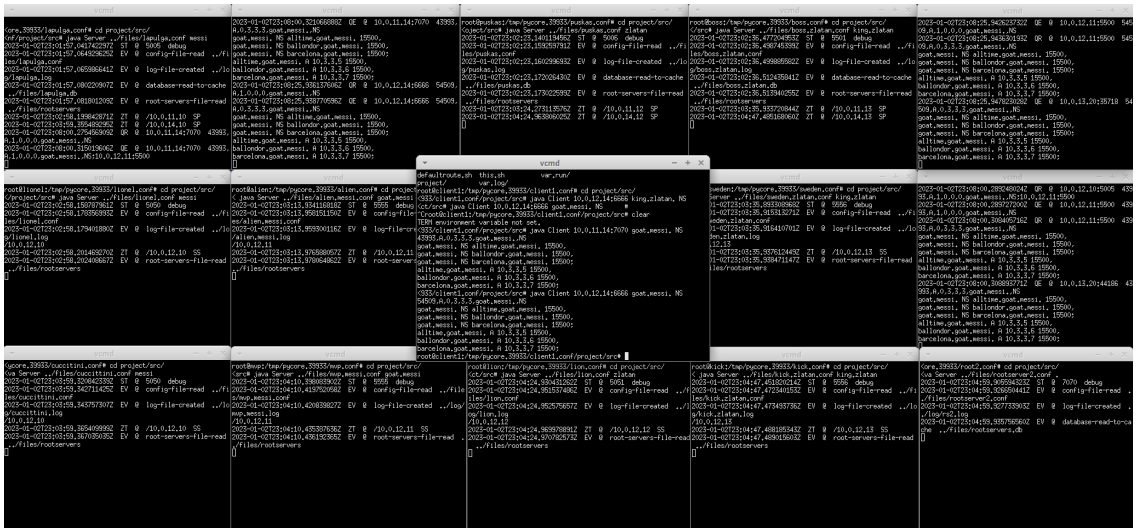

Figura 5.1: Execução de todos os elementos da Topologia

Capítulo 6

Planeamento do Ambiente de Teste

6.1 Topologia

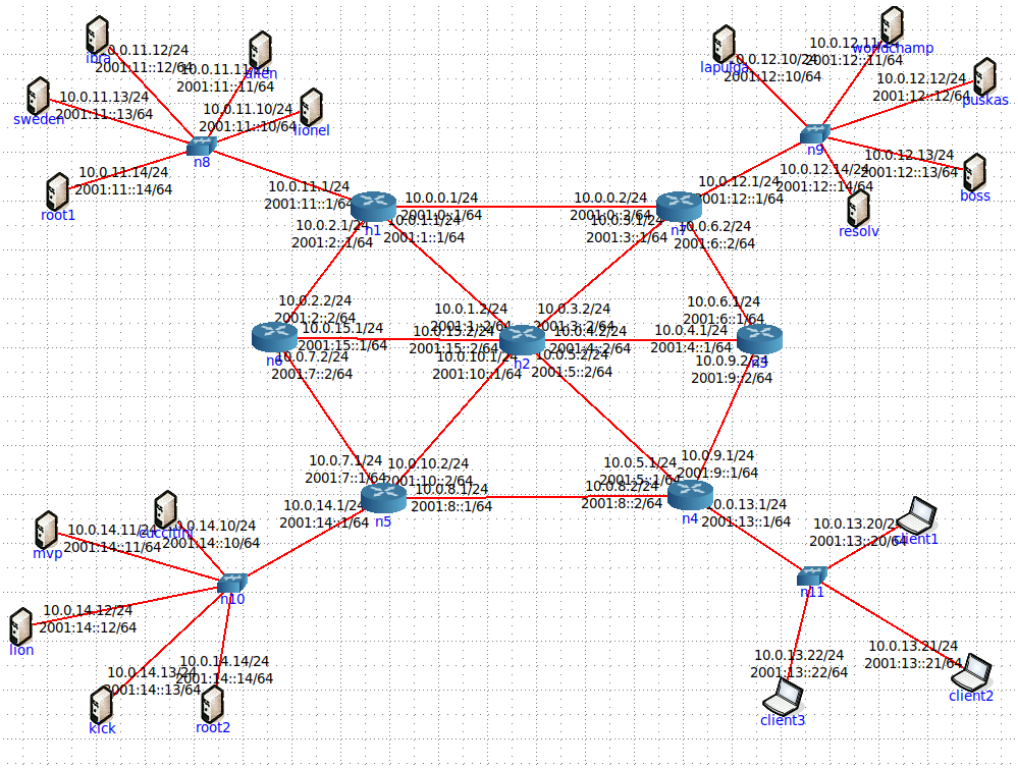

Figura 6.1: Topologia Final de Teste

Como pode ser observado na **Figura 6.1**, a topologia concebida torna-se completa com a presen 16 elementos fulcrais ao correto funcionamento do programa, tendo em destaque dois **Servidores de Topo**, como por exemplo **root1**, um **Servidor Primário** e dois **Servidores Secundários** por domínio, por exemplo, referente ao domínio **"goat.messi"**, temos, respetivamente os servidores **"worldChamp"**, **"Alien** e **"Mvp"**, um **Servidor de Resolução**, presente no domínio **"goat.messi"**, designado **"resolv"**, e um ou mais clientes, que serão responsáveis pelo envio de **queries**.

6.2 Ficheiros

De modo a respeitar as configurações de todos os tipos de servidores, bem como a sintaxe e a semântica de todos os ficheiros de **input**, foram criados os seguintes ficheiros:

```
# Configuration file for primary server for goat.messi
goat.messi DB ../files/worldchamp.messi.db
goat.messi SS 10.0.11.11
goat.messi SS 10.0.14.11
goat.messi LG ../log/worldchamp.messi.log
all LG ../log/all.log
root ST ../files/rootservers
```

Figura 6.2: Ficheiro de Configuração do Servidor Primário **worldChamp**

```
# DNS database file for domain king.zlatan
king.zlatan. SOASP milan.king.zlatan. 86400
king.zlatan. SOAADMIN dns\.admin.king.zlatan. 86400
king.zlatan. SOASERIAL 0117102022 86400
king.zlatan. SOAREFRESH 14400 86400
king.zlatan. SOARETRY 3600 86400
king.zlatan. SOAEXPIRE 604800 86400
king.zlatan. NS milan.king.zlatan. 86400
king.zlatan. NS goldenboot.king.zlatan. 86400
king.zlatan. NS psg.king.zlatan. 86400
king.zlatan. MX striker.king.zlatan. 86400
king.zlatan. MX goal.king.zlatan. 86400
milan.king.zlatan. A 10.3.3.5 86400
goldenboot.king.zlatan. A 10.3.3.6 86400
psg.king.zlatan. A 10.3.3.7 86400
striker.king.zlatan. A 193.136.130.200 86400
goal.king.zlatan. A 193.136.130.201 86400
www.king.zlatan. A 193.136.130.80 86400
www.king.zlatan. A 193.136.130.81 86400
ftp.king.zlatan. A 193.136.130.20 86400
sp.king.zlatan. CNAME milan.king.zlatan. 86400
ss1.king.zlatan. CNAME goldenboot.king.zlatan. 86400
ss2.king.zlatan. CNAME psg.king.zlatan. 86400
mail1.king.zlatan. CNAME striker.king.zlatan. 86400
mail2.king.zlatan. CNAME goal.king.zlatan. 86400
```

Figura 6.3: Ficheiro de Base de Dados do Servidor Primário **boss**

```
# Configuration file for secondary server for king.zlatan
king.zlatan SP 10.0.12.13
king.zlatan LG ../log/kick.zlatan.log
all LG ../log/all.log
root ST ../files/rootservers
```

Figura 6.4: Ficheiro de Configuração do Servidor Secundário **kick**

```
# Configuration file for resolution server for goat.messi
goat.messi DD 10.0.12.11:5500
goat.messi LG ../log/resolv.log
all LG ../log/all.log
root ST ../files/rootservers
```

Figura 6.5: Ficheiro de Configuração do Servidor de Resolução **resolv**

```
# Configuration file for root server 1
. DB ../files/rootservers.db
. LG ../log/rs1.log
all LG ../log/all.log
```

Figura 6.6: Ficheiro de Configuração do Servidor de Topo **root1**

```
# Root servers database
messi lapulga 10.0.12.10:5005
messi lionel 10.0.11.10:5005
messi cuccittini 10.0.14.10:5005
zlatan puskas 10.3.3.3:5005
zlatan ibra 10.2.2.3:5005
zlatan lion 10.1.1.3:5005
```

Figura 6.7: Ficheiro de Base de Dados do Servidor de Topo **root1**

```
10.0.11.14:7070
10.0.14.14:7777
```

Figura 6.8: Ficheiro de Servidores de Topo

Capítulo 7

Avaliação do grupo

A heteroavaliação realizada pelos membros do grupo, teve por base o trabalho e o empenho mostrado pelos mesmos durante todo o projeto. Todas as notas apresentadas encontram-se numa escala de 0-20.

	B.1	B.2	B.3	B.4	B.5	B.6	B.7	B.8
Gonçalo Pereira	14	14	14	16	14	20	14	15
José Moreira	20	18	18	19	20	20	19	19
Santiago Domingues	16	18	18	18	17	20	18	19

Figura 7.1: Avaliação dos membros do grupo

Capítulo 8

Conclusão

Como foi apresentado, o objetivo deste projeto passa por desenvolver um sistema **DNS**. Deste modo, seria fulcral criar peças de software que, interagindo através dos protocolos estabelecidos, tivessem a capacidade de comunicar entre si de forma a satisfazerem determinadas **queries**. Nesta segunda fase, pretendia-se o desenvolvimento de código capaz de reproduzir, na totalidade, o funcionamento de um sistema de **DNS**, realizando todas as **queries** pretendidas de um modo iterativo, com a total comunicação entre todos os servidores, que formam o ambiente de testes.

Com isto, acreditamos ter desenvolvido as peças de código necessário para cumprir com as funcionalidades do trabalho prático, desde documentação do código a definição precisa dos vários ficheiros, . Deste modo, encerra-se o relatório final referente ao projeto projeto: **TP2 - Implementação de um sistema DNS**.

Capítulo 9

Bibliografia

https://elearning.uminho.pt

https://www.cloudflare.com/learning/dns/what-is-dns/

https://www.scientific.net/paper-keyword/dns

Application Layer, Computer Networking: A Top-Down Approach 8th Ed., J. Kurose, K. Ross, Pearson, 2020

Transport Layer, Computer Networking: A Top-Down Approach 8th Ed., J. Kurose, K. Ross, Pearson, 2020

Computer Networking - A Top-Down Approach, 7th Edition, Kurose, Ross