



Implementação de um Sistema DNS

Trabalho Prático 2

Comunicações por Computador
Licenciatura em Engenharia Informática

PL1 - Grupo 03

Gabriela Santos Ferreira da Cunha - a97393

Miguel de Sousa Braga - a97698

Nuno Guilherme Cruz Varela - a96455

janeiro, 2023

Conteúdo

| | | |
|----------|--|-----------|
| 1 | Introdução | 4 |
| 2 | Arquitetura do Sistema | 4 |
| 2.1 | Descrição do sistema | 4 |
| 2.2 | Requisitos funcionais | 5 |
| 2.2.1 | <i>Parsing</i> de ficheiros | 5 |
| 2.2.2 | Cliente | 5 |
| 2.2.3 | Servidores | 5 |
| 2.2.4 | <i>Cache</i> | 6 |
| 2.2.5 | <i>Logger</i> | 6 |
| 2.3 | Módulos e componentes de software | 7 |
| 2.3.1 | Cliente | 7 |
| 2.3.2 | Servidores | 7 |
| 3 | Modelo de Informação | 9 |
| 3.1 | Ficheiros de configuração | 9 |
| 3.2 | Ficheiros de <i>log</i> | 10 |
| 3.3 | Ficheiros de dados | 11 |
| 3.4 | Tratamento de erros | 12 |
| 3.5 | PDU | 13 |
| 3.6 | Mecanismo de codificação da unidade de dados | 15 |
| 4 | Modelo de Comunicação | 18 |
| 4.1 | Descrição introdutória do modelo comunicacional | 18 |
| 4.2 | Interações | 18 |
| 4.2.1 | Resolução de <i>queries</i> | 18 |
| 4.2.2 | Transferência de zona | 21 |
| 4.3 | Comportamento dos elementos em situações de erro | 22 |
| 5 | Implementação | 23 |
| 5.1 | Cliente | 23 |
| 5.2 | Servidores | 24 |
| 5.2.1 | Receber e responder a <i>queries</i> | 24 |
| 5.2.2 | Realizar transferência de zona | 25 |
| 5.3 | <i>Cache</i> | 26 |
| 5.4 | Codificação binária | 27 |
| 6 | Planeamento do Ambiente de Teste | 27 |
| 7 | Elaboração de testes na topologia criada | 30 |
| 7.1 | Transferência de zona | 30 |
| 7.2 | Modo iterativo | 31 |
| 7.3 | Modo recursivo | 33 |
| 7.4 | Domínios por defeito | 35 |
| 7.4.1 | Servidor de resolução | 35 |

| | | |
|-----------|--|-----------|
| 7.4.2 | Servidor primário/secundário | 36 |
| 7.5 | Prioridades | 37 |
| 7.6 | DNS reverso | 37 |
| 8 | Trabalho desenvolvido | 38 |
| 9 | Conclusão | 39 |
| 10 | Lista de Siglas e Acrónimos | 40 |
| 11 | Referências | 40 |
| 12 | Anexos | 40 |
| 12.1 | Ficheiro com lista de servidores de topo | 40 |
| 12.2 | Ficheiros de configuração | 41 |
| 12.3 | Ficheiros de base de dados | 45 |

1 Introdução

Para este trabalho, foi-nos proposta a implementação de um sistema de resolução de nomes (DNS), testando-o numa topologia configurada de raiz.

O DNS, abreviação de *Domain Name System*, é um protocolo da camada de aplicação que consiste numa base de dados distribuída implementada numa hierarquia de servidores. Este protocolo ajuda a direcionar o tráfego na internet, associando nomes de domínios, mais facilmente memorizáveis, a endereços IP, necessários à localização e identificação de serviços e dispositivos, processo denominado por *name resolution*.

Este trabalho foi desenvolvido em Python, de modo a construir um sistema genérico que funcione em qualquer contexto de redes de computadores. Para além disso, um dos objetivos deste trabalho passa por garantir a modularidade do sistema que iremos construir, evitando a duplicação de código e aumentando a capacidade de manutenção e eventual expansão do código numa reutilização futura.

2 Arquitetura do Sistema

2.1 Descrição do sistema

Para a implementação deste sistema de DNS, existem 4 elementos fundamentais que podem interagir: CL (Cliente), SP (Servidor Primário), SS (Servidor Secundário) e SR (Servidor de Resolução). Todos estes tipos de servidores devem possuir a funcionalidade de *cache*, sendo essencial principalmente para servidores não autoritativos.

Para além destes tipos fundamentais de elementos, existem 2 variantes especiais de servidores: os ST ou *DNS Root Servers* (Servidores de Topo) e os SDT ou *DNS Top-Level Domain Servers* (Servidores de Domínios de Topo).

No contexto deste projeto, o comportamento dos SDT é similar ao dos SP e dos SS, ainda que não tenham domínios hierarquicamente acima na árvore DNS (para além do domínio *root*). Deste modo, um SP ou SS autoritativo para um domínio de topo é um SDT.

Os ST têm, também, o mesmo comportamento que um SP mas, por outro lado, a sua base de dados apenas inclui, para cada domínio de topo, os nomes e endereços IP dos respetivos SS e SP - SDTs desse domínio. Com isto, os servidores de topo apenas podem responder com estes dois tipos de informação.

Em suma, todos estes tipos de servidores são implementados no mesmo componente.

2.2 Requisitos funcionais

2.2.1 *Parsing* de ficheiros

- O sistema deve suportar a leitura de ficheiros;
- Utilizado para retirar as características de um servidor, nomeadamente configuração e base de dados;
- As linhas vazias e os comentários de ficheiros devem ser ignorados;
- Se existir algum erro no ficheiro de configuração, o servidor não deve ser inicializado;
- Se existir algum erro ou incoerência no ficheiro de dados, a linha deve ser ignorada e o erro deve ser reportado no respetivo ficheiro de *log* e, se o servidor estiver em modo *debug*, no *stdout*.

2.2.2 Cliente

- O sistema deve implementar um programa que recebe como argumentos o endereço do servidor DNS, o domínio e o tipo de uma *query* e, opcionalmente, as respetivas *flags* ativas, o respetivo *timeout* e o modo de execução do cliente;
- O sistema deve atribuir valores *default* para a porta de atendimento, *flags* e *timeout* de uma *query* e modo de execução caso algum dos valores não seja fornecido pelo utilizador;
- O cliente deve construir uma *query* com os parâmetros dados;
- O cliente deve enviar a *query* construída através de um *socket* UDP para o servidor destino;
- O cliente espera pela resposta do servidor e imprime-a no *stdout*.

2.2.3 Servidores

- O servidor deve arrancar com uma configuração válida proveniente de um ficheiro de configuração;
- O servidor tem de saber interpretar *queries*;
- O sistema deve garantir que um servidor pode receber *queries* e enviar respostas através de um *socket* UDP;
- O sistema deve suportar o mecanismo de transferência de zona entre servidores primários e secundários, recorrendo a *sockets* TCP;
- O servidor tem de registar toda a atividade que nele acontece através de *logs* nos devidos ficheiros;

- O servidor deve apresentar modo *shy* e *debug*, passando também a registar a sua atividade no *stdout* neste último;
- O registo de informação em *cache* deve ser efetuado sempre que se receber uma mensagem DNS respondida com sucesso (*response code* igual a 0);
- Caso receba uma mensagem com a bandeira recursiva ativa, o servidor vai tentar resolver a *query* iniciando o modo iterativo/recursivo, dependendo do modo que este suporta;
- Caso receba uma mensagem sem a *flag* recursiva ativa, o servidor vai responder à mensagem apenas com recurso à informação que tem na sua *cache* e envia-a de volta para o emissor da mensagem recebida.

2.2.4 *Cache*

- O sistema deve implementar um mecanismo de *cache*;
- A *cache* vai ser implementada nos servidores, de modo a diminuir tempos de acesso aos dados;
- Os servidores vão guardar informação dos seus ficheiros de base de dados na *cache*, excetuando os servidores de resolução.
- Os servidores vão também guardar na *cache* informação sobre mensagens respondidas com sucesso.
- Quando o servidor arranca, a *cache* deve ser inicializada com, pelo menos, uma entrada/posição FREE (se o tamanho for gerido dinamicamente);
- Deverá ser possível adicionar um elemento à *cache*, atualizando o *timestamp* de uma entrada que eventualmente já exista;
- A *cache* deverá permitir saber se o TTL de uma dada entrada já expirou ou não;
- Deve ser possível procurar uma entrada válida com um dado nome e um dado tipo.

2.2.5 *Logger*

- Deverá permitir associar um ficheiro de *output* e o *stdout* para onde a informação de *logging* deve ser enviada;
- Deverá indicar o *timestamp* e os dados associados à mensagem de *logging*.

2.3 Módulos e componentes de software

2.3.1 Cliente

Uma aplicação cliente de DNS é um processo que precisa de informação de uma base de dados, como por exemplo uma aplicação de *browser*. Esta informação é obtida a partir de *queries* DNS enviadas a servidores. Neste sistema, vai ser desenvolvido um cliente com interação a partir da linha de comandos, como referimos na descrição do mesmo.

2.3.2 Servidores

Servidor primário

O servidor primário é um servidor DNS que responde a, e efetua, *queries* DNS e tem acesso direto à base de dados de um domínio, sendo este que o gere. Sempre que se quer atualizar informação de um domínio DNS, tem que se aceder diretamente à base de dados do seu servidor primário. No contexto deste sistema, o servidor primário recebe um ficheiro de configuração, um ficheiro de base de dados por cada domínio e um ficheiro com a lista de servidores de topo.

Servidor secundário

O servidor secundário é um servidor DNS que, tal como o servidor primário, responde a, e efetua, *queries* DNS. Tem autorização para possuir uma réplica da base de dados original do servidor primário dum domínio, através de um processo denominado de transferência de zona. No contexto de sistema, o servidor secundário vai receber um ficheiro de configuração e um ficheiro com a lista de servidores de topo.

Servidor de resolução

O servidor de resolução responde a, e efetua, *queries* DNS sobre qualquer domínio, assim como os outros servidores, mas não tem autoridade sobre nenhum pois serve apenas de intermediário. Os SR podem funcionar em cascata, dependendo da gestão da sua configuração. Neste projeto, um SR tem como *input* um ficheiro de configuração e um ficheiro com a lista de servidores de topo; como *output* tem um ficheiro de *log*.

As interfaces de entrada e saída dos programas que implementam os servidores encontram-se evidenciadas na figura 1.

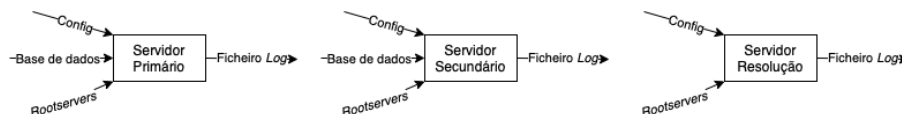


Figura 1: Componentes de *software*.

De modo a cumprir com o objetivo de ter uma solução bem estruturada e modular, desenvolvemos uma arquitetura baseada no seguinte diagrama.

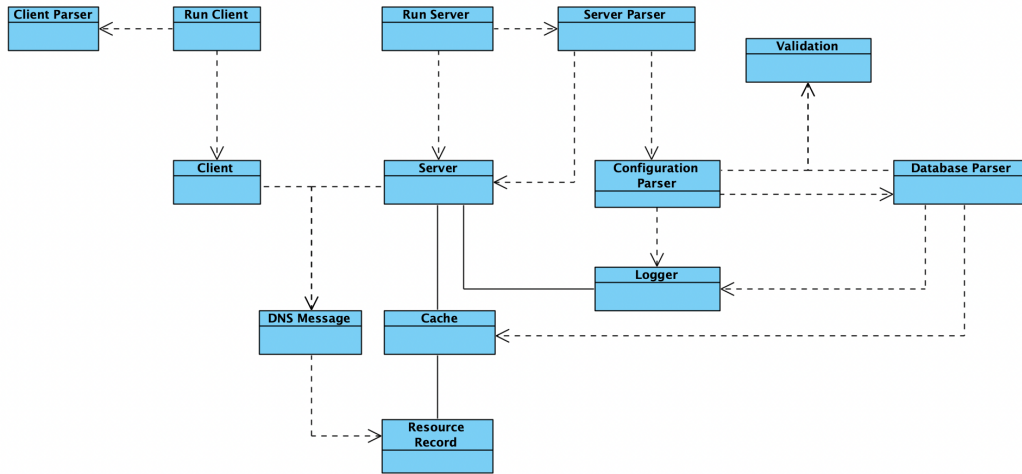


Figura 2: Arquitetura do sistema.

Anteriormente, a arquitetura do nosso sistema contemplava uma hierarquia de classes na implementação dos diferentes tipos de servidores. No entanto, uma vez que, nesta fase, o servidor pode assumir vários papéis em simultâneo para diversos domínios, a nossa abordagem já não faria sentido, pelo que passamos a implementar tudo no mesmo módulo.

A execução dos componentes cliente e servidor é tratada pelos módulos “Run Client” e “Run Server”, respetivamente, e serve-se das bibliotecas “Client” e “Server”, com o objetivo de aumentar o nível de encapsulamento e abstração. O *parse* dos argumentos inseridos pelo utilizador para esta execução é feito nos módulos “Client Parser” e “Server Parser”, sendo este último responsável também pela leitura dos ficheiros de configuração e dados de um servidor que, por sua vez, recorre a funções auxiliares de validação de parâmetros como portas de atendimento e IP’s contidas no módulo “Validation”.

A biblioteca do cliente disponibiliza métodos para enviar e receber mensagens DNS do servidor. Já na biblioteca do servidor, encontram-se todos os métodos relativos à resolução de mensagens DNS, realização de transferências de zona e todo o tipo de interações entre servidores. Estas duas bibliotecas servem-se do módulo “DNS Message” que representa a estrutura de um PDU.

Já para a implementação da *cache* de um servidor, recorremos ao módulo “Cache”. A *cache* é constituída por vários *records*, por sua vez representados no módulo “Resource Record”.

Toda a atividade relevante de um servidor deve ser registada nos respetivos ficheiros de *log* e no *stdout*, dependendo do modo de funcionamento do componente. Assim, cada servidor tem o seu próprio *logger*, representado pela classe “Logger”, que efetua estes registos.

3 Modelo de Informação

3.1 Ficheiros de configuração

Os ficheiros de configuração são apenas lidos e processados no arranque do componente de *software* a que dizem respeito e moldam o seu comportamento. Se for necessário atualizar o comportamento dos servidores com informação modificada nos ficheiros de configuração ou de dados que lhes dizem respeito a única alternativa é reiniciar esses servidores.

Este ficheiro tem uma sintaxe específica e o componente deve saber reportar as situações de incoerência de configuração que possam resultar do ficheiro ou sintaxes que não entenda; nesses casos o componente deve registar a informação nos *logs* respetivos e encerrar imediatamente a execução. As linhas vazias e os comentários devem ser ignorados e uma linha do ficheiro considerada válida deve seguir a seguinte sintaxe:

«parâmetro» «tipo de valor» «valor associado ao parâmetro»

Os tipos de valor suportados são os seguintes:

- **DB** - nos SPs, o valor indica o *PATH* do ficheiro da base de dados do domínio indicado no parâmetro;
- **SP** - nos SSs, o valor indica o endereço (IP:[porta]) do SP do domínio indicado no parâmetro;
- **SS** - nos SPs, o valor indica o endereço (IP:[porta]) de um dos SSs do domínio indicado no parâmetro;
- **DD** - nos SRs, o valor indica o endereço (IP:[porta]) que este servidor deve contactar diretamente, quando recebem queries sobre o domínio especificado no parâmetro. Nos SPs e SSs, o parâmetro indica um dos domínios a que os servidores podem responder;
- **ST** - para todos os servidores, o valor indica o *PATH* do ficheiro com os servidores de topo;

- **LG** - em todos os servidores, o valor indica o *path* do ficheiro de *log* associado ao domínio do parâmetro, que deve ser um domínio para o qual o servidor é SP ou SS. O parâmetro pode ser “all”, nesse o ficheiro recebe entradas de *log* referentes a todos os domínios.

3.2 Ficheiros de *log*

Os ficheiros de *log* registam toda a atividade relevante do componente. Sempre que um componente arranca, o mesmo deve verificar a existência dos ficheiros de *log* indicados no seu ficheiro de configuração; Caso um ficheiro não exista, este deve ser criado e, se já existir, as novas entradas devem ser adicionadas após a última entrada do ficheiro, sendo que deve existir uma entrada de *log* por cada linha.

Cada entrada deve ter a seguinte sintaxe:

«etiqueta temporal» «tipo» «endereço IP[:porta]» «dados da entrada»,

onde a etiqueta temporal é a data e a hora completa do sistema operativo na altura em que aconteceu a atividade registada.

Os tipos de entradas aceites são os seguintes:

- **QR/QE** - receção/envio de uma *query* de/para o endereço indicado. Os dados da entrada são os dados da *query* em questão, representados em modo *debug*;
- **RP/RR** - envio/receção de uma resposta a uma *query* para o/do endereço indicado. Os dados da entrada são os dados da *query* em questão, representados em modo *debug*;
- **ZT** - iniciado e concluído com sucesso um processo de transferência de zona. O endereço refere o servidor no outro lado da transferência. Podem ser ainda indicados o papel do servidor na transferência (SS ou SP), a duração da transferência em ms e o número de *bytes* transferidos;
- **EV** - detetado um evento diverso no componente, devendo o endereço fazer referência ao *localhost*. Podem ser adicionados detalhes que especifiquem o tipo de evento ocorrido;
- **ER** - detetado um erro na decodificação de um PDU enviado do endereço indicado. Pode ser ainda reportado o que foi possível decodificar ou onde ocorreram erros;
- **EZ** - detetado um erro no processo de transferência de zona. O endereço refere o servidor no outro lado da transferência. Pode ser ainda indicado o papel do servidor na transferência (SS ou SP);

- **FL** - detetado um erro de funcionamento interno do componente, devendo o endereço fazer referência ao *localhost*. Deve ser também reportado o tipo de erro ocorrido;
- **TO** - detetado um *timeout* na interação com o servidor com o endereço indicado. Deve ser também indicado o processo em que este *timeout* ocorreu (por exemplo, transferência de zona ou resposta a uma *query*);
- **SP** - parada a execução de um componente, devendo o endereço fazer referência ao *localhost*. Deve ser possível indicar a razão da paragem.
- **ST** - iniciada a execução de um componente, devendo o endereço fazer referência ao *localhost*. Deve ser ainda indicada a porta de atendimento, o valor do *timeout* (em ms) e o modo de funcionamento (“*shy*” ou *debug*);

3.3 Ficheiros de dados

No ficheiro de configuração de cada servidor primário, é indicado o *path* para o ficheiro que contém os dados que deverão ser inseridos na base de dados de cada SP de modo a obter a informação para responder às *queries* recebidas. Assim como no ficheiro de configuração, este ficheiro também tem uma sintaxe específica e qualquer situação de incoerência ou sintaxe que não esteja em conformidade deve ser registada nos logs respetivos. As linhas vazias e os comentários devem também ser ignorados. Uma linha válida do ficheiro de dados deve obedecer à seguinte sintaxe:

«parâmetro» «tipo do valor» «valor» «TTL» «prioridade»

O tempo de validade (TTL) é o tempo máximo em segundos que os dados podem existir na *cache* de um servidor. Consoante o tipo de entrada, o TTL pode ou não ser suportado, sendo o seu valor 0 quando este não é suportado. Quando o 4º campo da linha, isto é, o campo 'TTL' é igual a "TTL", o valor do TTL deverá ser o valor *default*.

A prioridade define uma ordem de prioridade de vários valores associados ao mesmo parâmetro. Quanto menor o valor maior a prioridade. Para parâmetros com um único valor ou para parâmetros em que todos os valores têm a mesma prioridade, o campo não deve existir. Este campo ajuda a implementar sistema de *backup* de serviços/*hosts* quando algum está em baixo (números de prioridade entre 0 e 127) ou balanceamento de carga de serviços/*hosts* que tenham vários servidores/*hosts* aplicativos disponíveis (números de prioridade entre 128 e 255).

Os nomes completos de e-mail, domínios, servidores e hosts devem terminar com um '.'. Quando os nomes não terminam com '.' subentende-se que são concatenados com um sufixo por defeito definido através do parâmetro @ do tipo DEFAULT.

Os tipos de valor suportados são os seguintes:

- **DEFAULT** - o campo valor representa uma macro para o valor indicado no parâmetro. No caso do parâmetro ser '@', o valor indica o texto que deve ser acrescentado sempre que um nome não termina com um '.'; No caso do parâmetro ser 'TTL' o valor indica o novo valor default que deverá tomar o TTL.
- **SOASP** - o valor indica o nome completo do SP do domínio indicado no parâmetro;
- **SOADMIN** - o valor indica o endereço de e-mail do administrador do domínio;
- **SOASERIAL** - o valor indica o número de série da base de dados do domínio indicado no parâmetro;
- **SOAREFRESH** - o valor indica o tempo em segundos para um SS perguntar novamente ao SP do domínio indicado no parâmetro qual o número de série da base de dados dessa zona;
- **SOARETRY** - o valor indica o tempo em segundos para um SS voltar a perguntar ao SP do domínio indicado no parâmetro qual o número de série da base de dados dessa zona, após um timeout nessa pergunta;
- **SOAEXPIRE** - o valor indica o tempo em segundos que o SS deve deixar de considerar a sua réplica da base de dados da zona indicada indicado no parâmetro. Quando o tempo expira, deve tentar realizar transferência de zona;
- **NS** - o valor indica o nome dum servidor autoritativo para o domínio indicado no parâmetro (nome dos SSs ou dos SPs);
- **A** - o valor indica o endereço IPV4 duma máquina dentro deste domínio com o nome indicado no parâmetro;
- **CNAME** - o valor representa um nome alternativo para o nome representado no parâmetro;
- **MX** - o valor indica o nome dum servidor e-mail do domínio indicado no parâmetro;
- **PTR** - o valor indica o nome duma máquina que usa o endereço IPV4 (domínio) indicado no parâmetro. A sintaxe é a simétrica da entrada A;

3.4 Tratamento de erros

Ao longo dos ficheiros de configuração e de dados dos servidores, podem ocorrer vários erros.

Relativamente aos ficheiros de configuração, quando existem endereços IP inválidos nas entradas do tipo “SP“, “SS“ e “DD“ é levantada uma exceção e a inicialização do servidor é imediatamente abortada.

Quanto aos ficheiros de dados, são levantadas exceções e emitidos logs a cada erro, sendo ignorada a linha onde o mesmo ocorreu e procedendo-se à leitura das restantes linhas.

Neste tipo de ficheiro são detetados erros quando, por exemplo, existem menos argumentos do que os esperados numa dada entrada - menos de 3 argumentos para entradas do tipo “DEFAULT“ e menos de 4 para outras entradas - ou mais argumentos do que os esperados - mais de 3 argumentos para entradas do tipo “DEFAULT“ e mais de 5 para as restantes. Em cada entrada, é também verificado o TTL introduzido, ocorrendo um erro caso seja introduzido algum valor que não possa ser convertido em um inteiro, à exceção de “TTL“ que representa o valor *default*. O mesmo ocorre para entradas do tipo “SOASERIAL“, “SOAREFRESH“, “SOAREFRESH“ e “SOAEXPIRE“, onde os valores introduzidos devem, também, ser inteiros. Quanto ao valor de prioridade, para além de ter de ser um inteiro, tem ainda de ser um valor entre 0 e 256 para que esta seja considerada válida. Para as entradas do tipo “A“ é verificado se o IP é válido e para entradas do tipo “CNAME“ se o nome ao qual se pretende adicionar um CNAME existe.

3.5 PDU

De acordo com o enunciado, a estrutura da unidade de dados protocolar do DNS é a apresentada na figura 3.

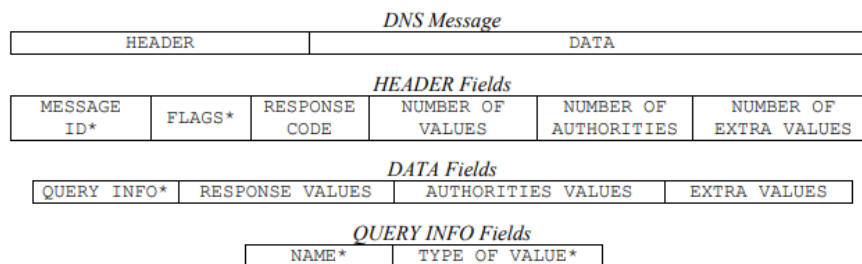


Figura 3: PDU da mensagem DNS.

Os vários campos que constituem a mensagem DNS podem-se dividir em 2 componentes: *header* - contém metadados sobre o que é transmitido - e *data* - contém os dados efetivamente transmitidos.

O cabeçalho pode, ainda, ser dividido nos seguintes campos:

- **Message ID** - identificador da interação entre duas máquinas que implementam o protocolo DNS. Permite relacionar as respostas recebidas com a *query* original;
- **Flags** - lista de *flags* que indicam o tipo de mensagem que está a ser transmitida. Em função das *flags* ativas, a semântica de outros campos pode ser alterada. As *flags* suportadas são as *flags* Q, R e A. Uma mensagem com a *flag* Q ativa é uma *query*; caso a *flag* não esteja ativa é uma resposta. Em relação à *flag* R, caso esta esteja ativa numa *query* indica-se que se pretende que o processo opere de forma recursiva; se estiver ativa numa resposta indica-se que o servidor que respondeu suporta o modo recursivo. Por último, a *flag* A ativada indica que a resposta é autoritativa; o valor nas *queries* originais é simplesmente ignorado;
- **Response Code** - indica as várias situações de erro ou sucesso que podem acontecer na resposta a uma *query*. Existem 4 *response codes* possíveis. Se o valor for 0, não existe qualquer tipo de erro e a resposta contém informação que responde diretamente à *query*, deve ser guardada em *cache*. Se o valor for 1, o domínio incluído em NAME existe mas não foi encontrada qualquer informação direta com um tipo de valor igual a TYPE OF VALUE - caso identificado como resposta negativa. Caso o valor seja 2, o domínio incluído em NAME não existe e é também uma resposta negativa. Por fim, caso o valor seja 3, a mensagem DNS não foi decodificada corretamente;
- **Number of values** - comprimento da lista *Response values*;
- **Number of authorities** - comprimento da lista *Authorities values*;
- **Number of extra values** - comprimento da lista *Extra values*;

O componente de dados do PDU é constituído pelos seguintes campos:

- **Query Info (Name)** - Representa o parâmetro pedido na *query* (p.e domínios, nome da máquina que se pretende saber o endereço). Tem a mesma semântica do parâmetro da base de dados do SP;
- **Query Info (Type of Value)** - Representa o tipo de valor pedido na *query* (p.e NS, MX, A, etc). Tem a mesma semântica do tipo do valor da base de dados do SP;
- **Response values** - lista dos *Resource Records* que fazem *match* no parâmetro e no tipo indicados no campo *Query Info*. Por exemplo, todos os MX referentes ao endereço “example.com.”;

- **Authorities values** - lista dos servidores autoritativos para o domínio especificado no campo *Query Info (Name)*. Na prática, é a lista das entradas com tipo NS e que fazem *match* com o domínio indicado;
- **Extra values** - lista das entradas da *cache*/base de dados referentes a endereços do domínio especificado no campo *Query Info (Name)*.

3.6 Mecanismo de codificação da unidade de dados

Numa primeira fase do trabalho, e de modo a facilitar o *debug* dos vários programas, toda a codificação de uma mensagem DNS era feita em modo texto, tal como explicitado no modo *debug* conciso. Contudo, essa é uma má abordagem do ponto de vista da eficiência do sistema que pretendemos construir. Tem duas desvantagens óbvias:

- Aumenta consideravelmente a complexidade espacial do nosso processo, uma vez que dados guardados em modo texto ocupam maior espaço em memória, comparativamente com dados codificados noutros tipos de codificações;
- Aumenta o *overhead* transmitido através da rede, seja por UDP ou TCP, o que reduz a performance do nosso sistema.

Por isso, implementamos um esquema de codificação mais eficiente, procurando resolver os problemas referidos acima. A codificação é então realizada da seguinte forma:

- **Message ID** - Temos de representar inteiros entre 1 e 65535 (65535 valores). Podemos, por isso, recorrer a 2 *bytes* para representar todos estes valores. Por exemplo, 0x0000 representa o valor 1 e assim sucessivamente.
- **Flags** - Neste trabalho temos a necessidade de representar 3 *flags*: Q, R e A. Para cada mensagem, consideramos as 6 seguintes combinações de *flags*: nenhuma *flag* ativa, Q e R ativas, R e A ativas e apenas uma das 3 *flags* ativas. São assim suficientes apenas 3 *bits* para representar estas combinações, encontrando-se a respetiva codificação na tabela abaixo.

| Flags ativas | Codificação |
|---------------------|--------------------|
| “” | 0000 |
| “Q” | 0001 |
| “R” | 0010 |
| “A” | 0011 |
| “Q+R” | 0100 |
| “R+A” | 0101 |

Tabela 1: Codificação das *flags*.

- **Response code** - Há a necessidade de representar 4 códigos de resposta (0 a 3), pelo que 2 *bits* são suficientes.
- **Number of response values, authorities values e extra values** - Estes campos podem ter um valor entre 0 e 255. São por isso suficientes 8 *bits* (1 *byte*) para representar todos os valores possíveis.

| Campo | Nº de bits |
|-------------------------------------|----------------------|
| <i>Message ID</i> | 16 (2 <i>bytes</i>) |
| <i>Flags</i> | 3 |
| <i>Response Code</i> | 2 |
| <i>Number of response values</i> | 8 (1 <i>byte</i>) |
| <i>Number of authorities values</i> | 8 (1 <i>byte</i>) |
| <i>Number of extra values</i> | 8 (1 <i>byte</i>) |

Tabela 2: Codificação dos campos do *header*.

Em suma, através da codificação binária utilizada, podemos codificar o cabeçalho de uma mensagem DNS com 45 *bits*, isto é, pouco mais de 5 bytes.

Quanto aos campos de dados, apenas o campo *Type of Value* terá um comprimento fixo, uma vez que os outros campos ou representam *strings* ou representam listas de registos. O campo *Name* é uma *string* que representa um domínio e, como tal, utilizaremos 1 *byte* para indicar o seu tamanho. À partida um domínio não vai ter mais do que 255 caracteres, pelo que decidimos usar apenas 1 *byte* para indicar o seu tamanho. É importante referir que, através do número de *response values*, *authorities values* e *extra values* conseguimos saber a quantidade de elementos de cada lista a desserializar.

| Campo | Nº de bits |
|---------------------------|-------------------------------|
| <i>Name</i> | 8 (1 <i>byte</i>) + variável |
| <i>Type of Value</i> | 4 |
| <i>Response Values</i> | variável |
| <i>Authorities Values</i> | variável |
| <i>Extra Values</i> | variável |

Tabela 3: Codificação dos campos de dados.

Os campos *response values*, *authorities values* e *extra values* vão conter listas de registos das base de dados dos servidores. Assim, temos também de definir um esquema de codificação para os registos. Um *record* tem um *Name* e um *Value* que são *string* e, por isso, utilizamos a codificação referida acima. A prioridade é um campo opcional, pelo que é necessário 1 *bit* para identificar a presença ou ausência deste. Se tiver prioridade, serão necessários mais 8 bits (1 *byte*) para o envio do valor. Como o TTL é um inteiro, utilizamos 4 *bytes* para a sua codificação. Na tabela seguinte encontra-se a codificação descrita.

| Campo | Nº de bits |
|--------------|-------------------------------|
| <i>Name</i> | 8 (1 <i>byte</i>) + variavel |
| <i>Type</i> | 4 |
| <i>Value</i> | 8 (1 <i>byte</i>) + variavel |
| TTL | 32 (4 <i>bytes</i>) |
| Prioridade | 1 ou 9 |

Tabela 4: Codificação de um *record*.

Os diferentes valores para o campo *Type* são os que se encontram na tabela abaixo.

| Tipo | Codificação |
|------------|-------------|
| SOASP | 0000 |
| SOAADMIN | 0001 |
| SOASERIAL | 0010 |
| SOAREFRESH | 0011 |
| SOARETRY | 0100 |
| SOAEXPIRE | 0101 |
| NS | 0110 |
| A | 0111 |
| CNAME | 1000 |
| MX | 1001 |
| PTR | 1010 |
| AXFR | 1011 |

Tabela 5: Codificação do tipo de um *record*.

Deste modo, optamos por uma codificação a nível de *bits*, recorrendo à biblioteca “BitString”. Caso não utilizássemos *bits*, apenas conseguiríamos codificar o cabeçalho com, pelo menos, 6 *bytes*, assumindo que as *flags* necessitariam de 6 bits para as representar que juntamente com os 2 *bits* do *response code* formariam 1 *byte*, tal como definido na primeira fase. Isto permite-nos aumentar a eficiência e diminuir o *overhead* da nossa codificação.

4 Modelo de Comunicação

4.1 Descrição introdutória do modelo comunicacional

A comunicação entre os diversos componentes do sistema é algo imprescindível para a resolução das *queries* DNS propostas por um dado cliente. Desta forma, o fio de execução principal de cada componente passa por enviar e receber mensagens DNS por *sockets*, de modo a possibilitar a comunicação entre os servidores e o cliente.

O cliente tem a possibilidade de formular uma *query* que irá ser enviada a um servidor e, em seguida, vai esperar por uma resposta vinda deste. Este servidor pode reencaminhar a *query* para outros servidores em algumas situações explicadas em seguida.

Os servidores suportam dois modos de resolução de *queries*: o iterativo e o recursivo. O modo como vão operar sobre a mensagem depende se a *flag* de recursividade está presente na mensagem DNS.

Neste sistema DNS, os servidores secundários terão ainda a possibilidade de realizar transferência de zona, isto é, obter uma cópia da base de dados de um domínio DNS. Para tal ocorrer, o servidor secundário vai estabelecer uma conexão TCP com o servidor primário do domínio para o qual quer uma cópia dos dados.

4.2 Interações

Num sistema DNS, existem 2 tipos de interações - síncronas e assíncronas. Estes distinguem-se consoante o protocolo da camada de transporte utilizado, sendo uma interação síncrona a que usa o protocolo TCP e assíncrona a que usa o protocolo UDP. A única interação síncrona que ocorre, neste caso, dá-se entre um servidor primário e um servidor secundário no processo de transferência de zona. As restantes interações são assíncronas e resultam de uma *query* enviada, seja de um cliente para um servidor ou mesmo reencaminhada de um servidor para um servidor do mesmo ou outro tipo.

4.2.1 Resolução de *queries*

Primeiramente, um cliente deve começar por formular a *query* a enviar para um servidor. Esta *query* é uma mensagem DNS que não deverá possuir conteúdo nos campos, à exceção dos que indicam o domínio, tipo e as *flags* ativas. Uma mensagem proveniente do cliente deve sempre conter a *flag* “Q”, indicando que se trata de uma *query*. Opcionalmente, o cliente pode também ativar a *flag* “R”, se quiser que o servidor resolva a *query* por ele em caso de falta de informação em *cache* para responder à *query* com sucesso.

Ao receber uma *query*, um servidor terá diferentes comportamentos consoante a sua configuração e as *flags* ativas na mensagem. Deste modo, caso a *flag* “R” não esteja presente, o servidor dedicar-se-á apenas a responder com base na informação em *cache*, podendo devolver uma resposta sem sucesso (*response code* 1 ou 2). Por outro lado, se a *flag* referida anteriormente estiver presente, o “fardo” da resolução é passado ao servidor e este deve proceder ao reencaminhamento da mensagem, caso não encontre informação suficiente em *cache* para responder à mensagem recebida (*response code* 1).

Consideremos, portanto, que a mensagem enviada contém a *flag* “R”.

Caso o servidor seja primário ou secundário, este irá verificar se pode responder à mensagem, tendo em conta os seus domínios por defeito. Caso o servidor não tenha domínios por defeito, tomará o papel de servidor de resolução, o qual será explicado posteriormente. Caso contrário, e averiguando se nos seus domínios por defeito está presente um sufixo do domínio da mensagem, o servidor tem autorização para proceder à formulação da resposta.

Se este servidor for primário/secundário para o domínio da mensagem, este irá responder com sucesso com recurso à informação em *cache* e, se for primário desse domínio, ativará a *flag* autoritativa (“A”) na resposta. Se o servidor não for SP/SS do domínio da mensagem, este irá assumir o papel de servidor de resolução para responder à mensagem.

Desta forma, se o servidor for de resolução e tiver a resposta em *cache*, envia de volta a mensagem respondida sem a *flag* autoritativa. Caso não tenha informação suficiente em *cache* para responder, irá proceder ao contacto com outros servidores, iniciando o modo iterativo ou recursivo. Partindo do princípio acima referido (*flag* “R” ativa) e que o servidor suporta modo recursivo, vai proceder à resolução de forma recursiva. De outra forma, irá iniciar o modo iterativo. O próximo servidor a ser contactado é calculado tendo em conta os domínios por defeito do SR e a informação proveniente da *cache*. No caso de encontrar um sufixo nos domínios por defeito, contacta diretamente esse servidor. Caso contrário, contacta diretamente os servidores de topo, se a *cache* não contiver endereços de servidores com o domínio da mensagem.

Um dos passos importantes para a compreensão das interações entre servidores, em geral, é perceber como funciona a *flag* recursiva (ou “R”) das mensagens DNS juntamente com o facto de o servidor suportar ou não o modo recursivo.

Caso a mensagem seja proveniente de um servidor e tenha a *flag* “R” ativa, a mesma indica que o servidor de onde veio a mensagem suporta o modo recursivo e, portanto, o servidor atual deve operar de um modo recursivo, se o suportar, ou dar início ao modo iterativo, em caso contrário.

Modo iterativo

Neste modo de resolução, o servidor irá recorrer aos diversos servidores para obter respostas do próximo passo a tomar, isto é, próximo servidor a contactar até encontrar a resposta, ou seja, mensagem com *response code* diferente de 1.

Inicialmente, para entrar em contacto com outro servidor, terá que determinar o próximo passo. Se em *cache* não possuir nada acerca do domínio, então contactará diretamente um servidor de topo. Caso encontre informação na *cache*, vai contactar o *name server* presente na *cache*. Ao enviar a mensagem, o servidor irá remover a *flag* recursiva da mesma, uma vez que não suporta o modo recursivo.

Esta operação resume-se no seguinte diagrama.

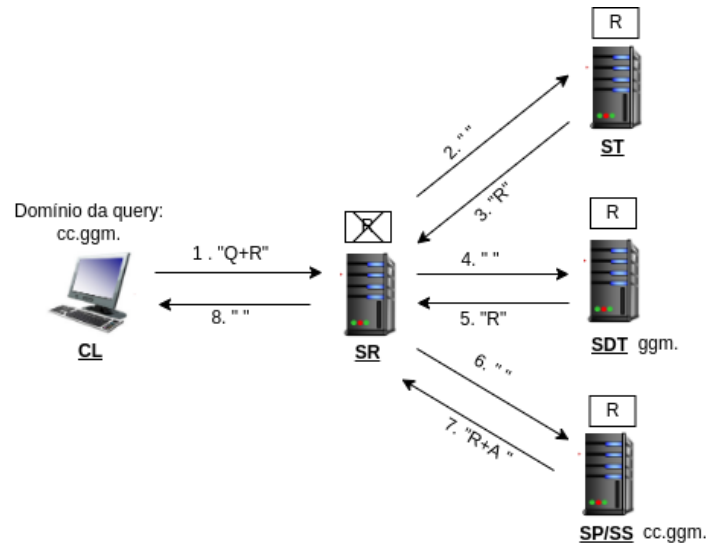


Figura 4: Representação do modo iterativo.

Modo recursivo

Em contrapartida, no modo recursivo, o servidor irá enviar a mensagem DNS com o "R" nas *flags* e, portanto, passará o "fardo" da resolução ao próximo servidor a contactar. Se este próximo servidor não suportar o modo recursivo, irá proceder de forma iterativa. Desta forma, a mensagem quando chegar ao servidor que iniciou a recursividade, irá ter *response code* 0 ou 2.

O modo recursivo pode ser representado pelo seguinte diagrama.

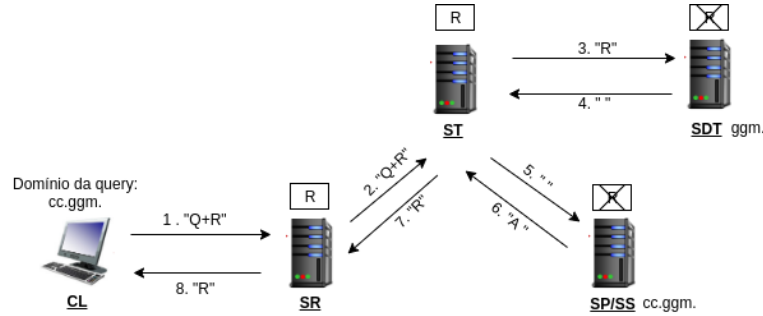


Figura 5: Representação do modo recursivo.

4.2.2 Transferência de zona

A transferência de zona é o processo de um servidor secundário pedir uma réplica da base de dados a um servidor primário para um dado domínio. Esta implica conexões TCP e pode ocorrer em quatro situações:

- Quando o servidor secundário arranca;
- Quando o intervalo temporal do "SOAREFRESH" termina;
- Quando o intervalo temporal do "SOARETRY" termina;
- Quando o intervalo temporal do "SOAEXPIRE" termina.

Inicialmente, o servidor secundário envia uma *query* ao servidor primário para saber a sua versão da base de dados ("SOASERIAL"). A resposta irá incluir a versão da base de dados do servidor primário para o domínio indicado na *query*. Caso as versões dos dois servidores não sejam iguais, ou seja, o servidor secundário possuir uma cópia desatualizada da base de dados do primário, o servidor secundário deve iniciar uma tentativa de transferência de zona, enviando uma *query* que indica que pretende realizar transferência de zona para um certo domínio.

Caso o domínio seja válido, o servidor primário envia o número de entradas a enviar. Se o servidor secundário aceitar receber todas essas entradas, responde ao servidor primário com o número de linhas. Desta forma, o servidor primário inicia o envio de todas as entradas da sua base de dados numeradas para o servidor secundário verificar se recebeu todas as entradas enviadas dentro de um intervalo de tempo predefinido. Após esse tempo terminar, o servidor secundário espera um tempo igual a "SOARETRY" antes de voltar a tentar a transferência de zona. No caso de o tempo predefinido não se esgotar, o servidor secundário verifica se já recebeu todas as entradas comparando com o número de linhas recebido inicialmente e termina a conexão, se já tiver recebido tudo.

A transferência de zona pode ser resumida pelo seguinte diagrama:

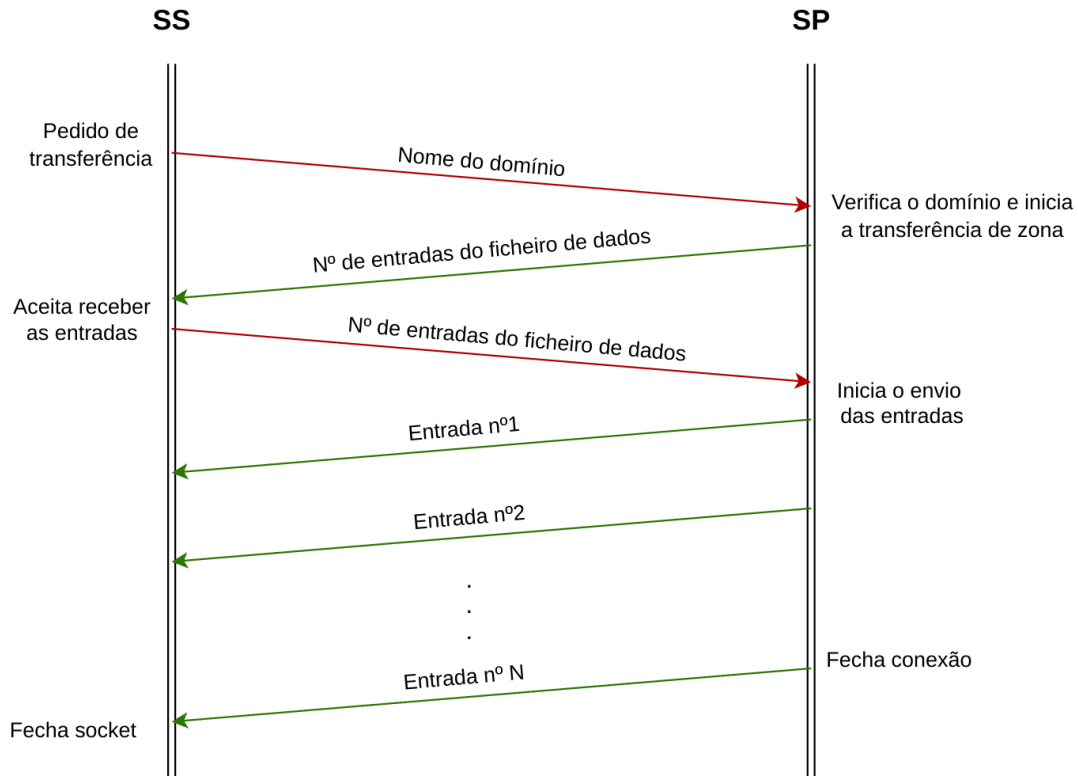


Figura 6: Representação do processo de transferência de zona.

4.3 Comportamento dos elementos em situações de erro

Face às situações de erro, os componentes devem estar preparados para lidar com as mesmas. Desta forma, nas diversas interações, a resposta pode não chegar em tempo útil ou mesmo não chegar efetivamente ao emissor da mensagem, sendo, por isso, necessário definir um valor de *timeout* para os componentes, de modo a que não fiquem bloqueados à espera de uma resposta.

Uma das situações de erro, dá-se quando um SP/SS possui domínios por defeito e nessa lista não se encontra nenhum sufixo do domínio da mensagem. Desta forma, o servidor não responde e ocorre um *timeout* no cliente. Assim, em todo o tipo de interações, se o emissor não receber uma resposta num tempo determinado, deve ocorrer um *timeout*.

Quando um cliente ou um servidor procura o próximo servidor a contactar, por vezes a escolha é múltipla, isto é, quando existe mais do que um servidor do domínio da *query* a ser enviada/reencaminhada. Na implementação deste sistema, o servidor atual deve ter em consideração todos estes servidores, contactando o primeiro da lista e, se necessário, caso ocorra um *timeout* nesta tentativa, o próximo, e assim sucessivamente. É importante referir que esta lista está ordenada pela prioridade das entradas de *cache* que indicam o endereço e o nome dos servidores a contactar.

Já em relação à transferência de zona ocorre um *timeout*, por exemplo, quando o servidor secundário não recebe as linhas de entrada do servidor primário dentro do tempo predefinido, sucedendo-se o término da ligação, uma vez que a conexão entre os SP e SS neste processo decorre sob o protocolo TCP. Quando o servidor secundário não recebe do servidor primário o número de entradas da sua DB, de modo a obter um número esperado de entradas a receber, ou esse número esperado não coincide com o número de linhas efetivamente recebidas, a conexão é terminada e o SS deve voltar a tentar efetuar o processo após SOARETRY segundos.

5 Implementação

5.1 Cliente

O nosso cliente recebe os parâmetros da *query* como argumentos da linha de comandos - endereço IP destino, domínio da *query*, tipo da *query* e, opcionalmente, *flag* recursiva, valor de *timeout* e o modo de funcionamento do componente. É importante referir que caso a porta não seja indicada no *input*, o valor *default* é 5353. Também os parâmetros opcionais têm um valor por predefinição, sendo o valor de *timeout* de 10 segundos e o modo de funcionamento o modo *debug*. Caso o utilizador não indique o requerimento da *flag* “R”, então a *query* terá unicamente a *flag* “Q” ativa.

Na figura 7 encontram-se vários exemplos de como um cliente pode ser executado.

```
python3 run_client.py 10.3.3.3:2000 ggm.root. MX R 20 shy
python3 run_client.py 10.3.3.3:2000 ggm.root. MX R 20
python3 run_client.py 10.3.3.3:2000 ggm.root. MX 20 shy
python3 run_client.py 10.3.3.3:2000 ggm.root. MX R shy
python3 run_client.py 10.3.3.3:2000 ggm.root. MX shy
python3 run_client.py 10.3.3.3:2000 ggm.root. MX
```

Figura 7: Execução do programa cliente.

Neste exemplo, o cliente cria uma mensagem DNS com domínio “ggm.root.” e tipo de entrada “MX”. Está, por isso, interessado em saber se existe na *cache*/base de dados do servidor inquirido alguma entrada do tipo “MX” com o domínio indicado.

Após criar a mensagem DNS em modo texto, o cliente cria o *socket* UDP, enviando para o endereço (IP:porta) destino especificado a mensagem correspondente. Por fim, espera pela resposta do servidor e apresenta-a no *stdout*, fechando o *socket*. Contudo, caso a resposta recebida não seja uma resposta com sucesso, isto é, com *response code* 0 ou 2, o mesmo vai iniciar o modo iterativo e contactar os servidores vindos na resposta. Com isto, optámos por criar um cliente que vai tentar encontrar resposta, caso o servidor não resolva a *query* por ele.

5.2 Servidores

Na primeira fase do trabalho, os servidores foram implementados numa hierarquia de classes, algo que não faria tanto sentido para esta segunda fase. Portanto, a sua implementação passa agora a ser feita num só módulo.

O programa que corre o servidor recebe como argumentos a diretoria do ficheiro de configuração, a porta de atendimento, o valor de *timeout*, o modo de resolução do servidor e o modo de funcionamento. Com esta informação criamos o objeto servidor através de um *parser* que vai ler toda a configuração. Se possuir ficheiros de dados, vai ser invocado o *parsing* destes ficheiros e é criado o objeto *Cache*, objeto este que vai conter toda a informação necessária para responder a *queries*.

Finalizada a configuração do servidor, vão ser criadas múltiplas *threads* com o seguinte funcionamento:

- *threads* para realizar transferências de zona, ao longo da sua execução, para os domínios para os quais é servidor secundário;
- uma *thread* para receber pedidos de transferência de zona;
- uma *thread* para receber pedidos de *queries*.

5.2.1 Receber e responder a *queries*

Na *thread* que trata das *queries*, o servidor inicialmente cria um *socket* UDP, na porta de atendimento passada, pelo qual o servidor vai receber mensagens DNS. Sempre que recebe um pedido, cria uma *thread* que vai ser responsável por responder a este (*thread-per-request*), o que permite ao servidor responder a múltiplas mensagens em simultâneo.

Cada *thread* criada executará o método “interpret_message” que vai interpretar a mensagem recebida. Este método começa por criar um *socket* UDP, pelo qual o servidor vai responder ao pedido, quer seja para responder ao cliente ou para contactar novos servidores. De seguida, vai verificar se o servidor é efetivamente um servidor de resolução ou primário/secundário para algum domínio. Esta distinção tem de ser feita, uma vez que ambos os casos apresentam diferentes comportamentos perante os domínios por defeito. Perante a mensagem, o servidor vai à sua *cache* com o objetivo de encontrar a resposta. Se na ida à *cache* conseguir formular uma resposta com sucesso, então devolverá a resposta para trás. Caso contrário, vai iniciar o processo de resolução da mensagem, se a mensagem contiver a *flag* recursiva ativa.

A resolução de uma mensagem, por parte do servidor, passa por encontrar o próximo servidor a contactar (método “find_next_step”) e repetir o processo até que mensagem contenha *response code* 0 ou 2. O próximo passo a dar é determinado tendo em conta as prioridades dos *records*, os *name servers* contidos na mensagem, os domínios por defeito e os servidores de topo. É importante referir que caso ocorra um *timeout* à espera de resposta do servidor, irá ser contactado o próximo servidor possível. Por exemplo, assumindo que a mensagem pertence ao domínio “cc.ggm.root.” e tem uma lista de 3 *name servers* do domínio “ggm.root.”, o servidor irá escolher o *record* com maior prioridade e caso falhe no contacto do escolhido, irá prosseguir na lista contactando o próximo.

Todo este processo acima descrito é acompanhado pelo registo de *logs* nos respetivos ficheiros presentes na configuração do servidor.

5.2.2 Realizar transferência de zona

Servidor primário

No programa que corre um servidor, é criada uma *thread*, caso seja primário de algum domínio, para aceitar todas as conexões de pedidos de transferência de zona vindas de servidores secundários dos seus domínios. Nesta vai ser criado um *socket* TCP que vai ser colocado à escuta (modo *listening*), de maneira a aceitar as várias conexões que receber. Mais uma vez, para permitir que o servidor execute múltiplas transferências de zona em simultâneo, é criada uma *thread* para resolver o pedido.

Relativamente ao processo de transferência de zona de um servidor primário, este vai ficar à espera que o SS envie mensagens pela conexão, mensagens estas que podem ser um pedido da versão da sua base de dados ou o início da transferência de zona (*query* AXFR). É importante referir que a receção no *socket* foi protegida com a colocação de um *timeout*, de modo a que o servidor não fique bloqueado infinitamente, caso algo corra mal da parte do servidor secundário.

Finalmente, todos os logs são registados, incluindo o tempo que demorou a efetuar a transferência de zona, e é fechada a conexão com o servidor secundário.

Servidor secundário

No arranque do servidor, para cada domínio para o qual o servidor é secundário é criada uma *thread* que visa tratar da transferência de zona para esse domínio ao longo da execução do componente. Deste jeito, é possível que um servidor suporte vários domínios para o qual é secundário, algo que era um dos objetivos do grupo.

A nossa implementação deste processo passa por termos um ciclo infinito em que a *thread* vai realizar efetivamente a transferência de zona e adormecer uma certa quantidade de tempo. Este tempo de *sleep* é determinado consoante o sucesso da transferência de zona. Caso tenha ocorrido algum *timeout*, o tempo de espera vai ser igual a SOARETRY segundos. Caso contrário, vai esperar SOAREFRESH segundos até tentar a próxima transferência. Tudo isto é implementado com o auxílio de exceções que são lançadas nas diversas funções da transferência de zona e posteriormente tratadas na *thread* que contém o ciclo infinito.

Relativamente à transferência, mais concretamente, nós optamos por dividi-la em 3 secções:

- **pedir versão:** o servidor envia uma *query* ao seu servidor primário pedindo o SOASERIAL da sua base de dados para o domínio em causa;
- **pedido da transferência de zona:** caso tenha a base de dados desatualizada, envia um pedido (*query* AXFR) para o servidor primário que lhe vai responder com o número de linhas da sua base de dados;
- **receção das entradas:** o servidor recebe todas as entradas do servidor primário, confirmando se recebeu o número de entradas expectável.

Finalmente, o valor de SOAEXPIRE obtido é registado na *cache* para o domínio da transferência de zona.

5.3 Cache

Os servidores têm a necessidade de guardar a informação relativa aos seus domínios. Para tal, necessitam de uma estrutura onde o possam fazer.

A *cache* foi então implementada de modo a poder ser guardada toda a informação relacionada a bases de dados de domínios e informação proveniente de efeitos de *caching*.

A *cache* é definida tal como está descrita no enunciado do trabalho prático. Utilizamos um dicionário que guardará, para cada domínio, uma lista que vai guardar todos os registos desse domínio sob a forma de “ResourceRecord”, classe que é definida para representar um registo. Cada entrada da lista guarda um nome do domínio, tipo do valor, valor, tempo de validade, prioridade, origem da entrada, *timestamp* de quando foi adicionada/atualizada e o estado da entrada.

Sempre que acedemos à *cache* para ir buscar entradas, o *timestamp* dos registos com origem “OTHERS” é atualizado, de forma a manter a *cache* sempre atualizada.

A *cache* é uma estrutura que pode ser acedida por várias *threads* concorrentemente. Portanto, vimos a necessidade de recorrer a primitivas de controlo de concorrência para garantir a exclusividade mútua neste objeto.

5.4 Codificação binária

Como referido anteriormente, decidimos fazer a serialização dos dados com recurso à menor unidade de informação, o *bit*. Inicialmente, isto traria alguns problemas, visto que sendo o *python* uma linguagem de muito alto nível, poderia ser algo problemático. No entanto, encontramos a biblioteca “BitString” que disponibiliza *arrays* de *bits* e fizemos uso dela. Desta forma, conseguimos serializar e desserializar, com sucesso, estruturas como as mensagens DNS e os *records* das bases de dados.

6 Planeamento do Ambiente de Teste

De modo a poder testar e validar o código concebido, foi-nos proposta a criação de um ambiente de teste, tendo por base uma topologia do CORE.

A topologia que iremos utilizar é a mesma que foi utilizada para o TP1. Esta é constituída por 4 subredes de hosts/servidores, possuindo também uma rede *backbone* de *routers* interligados.

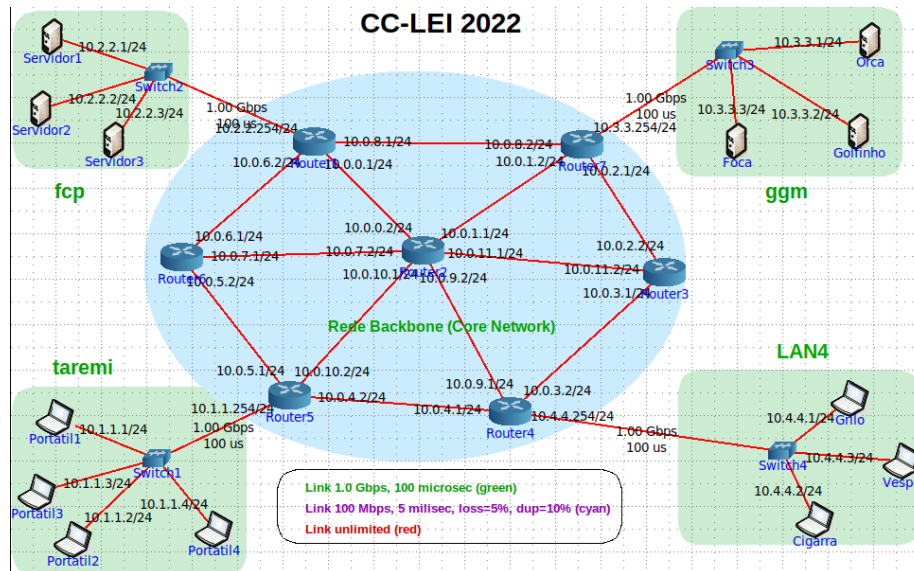


Figura 8: Topologia a utilizar.

Observações:

- Optamos por adicionar, numa fase avançada do projeto, uma nova máquina - “Portátil4”, pois seria necessário criar um servidor que servisse apenas como servidor de resolução;
- Os domínios não irão ser referidos com a raiz “root.”, tendo em vista a simplificação.

O nosso sistema de teste será constituído pelos seguintes componentes:

- **2 servidores de topo** : o primeiro é implementado pelo “Servidor1” e o segundo pelo servidor “Orca”. Faz sentido que estejam em subredes diferentes para que a falha numa subrede não impeça o sistema de continuar a funcionar.
- **2 domínios de topo** : o primeiro é o domínio “ggm” (SP - “Servidor2”, SS1 - “Portátil1”, SS2 - “Grilo”) e o segundo é o domínio “fcp” (SP - “Servidor2”, SS1 - “Golfinho”, SS2 - “Vespa”).
- **O subdomínio “cc.ggm”** : o SP é o “Portátil2”, o SS1 é representado pela máquina “Cigarra” e o SS2 pelo servidor “Golfinho”.
- **O subdomínio “taremi.fcp”** : o SP encontra-se na máquina “Grilo”, o SS1 na máquina “Portátil3” e o SS2 no “Servidor3”.
- **2 servidores de resolução** : sendo representados pelas máquinas “Foca” e “Portátil4”.
- **Um domínio de topo “reverse”**: representado pela máquina “Golfinho”.
- **Um subdomínio “in-addr.reverse”**: representado pela máquina “Cigarra”.
- **Um subdomínio “10.in-addr.reverse”**: na máquina “Portátil1”.

Em resumo, a árvore que representa a hierarquia dos vários domínios definidos é a seguinte:

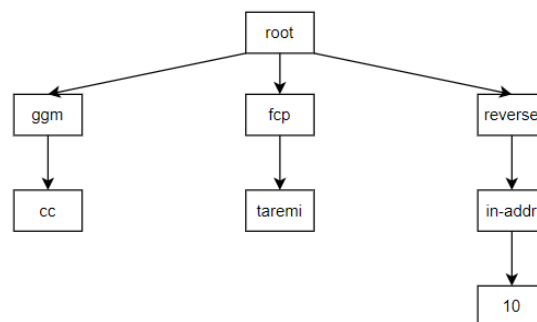


Figura 9: Hierarquia de domínios.

Quanto ao ramo do DNS reverso na árvore, optamos por colocar todas as entradas PTR na máquina que representa o subdomínio “10.in-addr.reverse“. Temos a noção de que na realidade, estas entradas são divididas pelas diferentes redes IP. Por exemplo, a entrada PTR para um endereço IP do tipo X.Y.Z.W teria de estar armazenada no domínio Z.Y.X.in-addr.reverse. Preferimos, por isso, limitar o número de servidores e a complexidade do sistema, armazenando todas as entradas PTR num mesmo servidor, pois estas pertencem todas à mesma rede IP.

De acordo com o traçado para a segunda fase, a mesma máquina poderá servir, na mesma porta de atendimento, diferentes tipos de servidores, por exemplo, servidor primário para um domínio e servidor secundário para outro domínio. Assim, para cada máquina, apresentam-se, em seguida, os componentes que estas representam:

| Nome da Máquina | Componentes |
|-----------------|----------------------------------|
| Servidor 1 | ST1 |
| Servidor 2 | SP(fcp), SP(ggm) |
| Servidor 3 | SS(taremi.fcp) |
| Orca | ST2 |
| Golfinho | SS(fcp), SP(reverse), SS(cc.ggm) |
| Foca | SR |
| Grilo | SP(taremi.fcp), SS(ggm) |
| Vespa | SS(fcp) |
| Cigarra | SS(cc.ggm), SP(in-addr.reverse) |
| Portátil 1 | SP(10.in-addr.reverse), SS(ggm) |
| Portátil 2 | SP(cc.ggm) |
| Portátil 3 | SS(taremi.fcp) |
| Portátil 4 | SR |

Tabela 6: Máquinas e respetivos componentes que representam.

7 Elaboração de testes na topologia criada

7.1 Transferência de zona

Neste teste pretende-se mostrar como o sistema lida com servidores que não estão em funcionamento, nomeadamente na transferência de zona. O servidor secundário vai fazer diversas tentativas até conseguir realizar a transferência de zona. O intervalo de tempo entre cada tentativa é dado pelo SOARETRY default que o grupo definiu. O teste foi realizado na máquina “Grilo” que é servidor secundário do domínio “ggm”. O servidor primário é representado pelo “Servidor2”.

```
vcmd
root@Grilo:/tmp/pycore.35733/Grilo.conf# bash run.sh
31:12:2022.13:06:52:321735 EV localhost conf-file-read dns/files/Grilo/config.txt
31:12:2022.13:06:52:322201 ST 127.0.0.1 2000 100 debug
31:12:2022.13:06:52:322331 EV localhost log-file-create dns/files/log_files/taremi_fcp.log
31:12:2022.13:06:52:322576 EV localhost log-file-create dns/files/log_files/ggm.log
31:12:2022.13:06:52:322757 EV localhost log-file-create dns/files/log_files/all.log
31:12:2022.13:06:52:322955 EV localhost db-file-read dns/files/db_files/taremi_fcp.db
31:12:2022.13:06:52:323440 EV localhost db-file-read dns/files/db_files/taremi_fcp.db
31:12:2022.13:06:52:326913 EZ 10.2.2.2:2000 SS: Failed to connect to primary server
31:12:2022.13:07:02:338851 EZ 10.2.2.2:2000 SS: Failed to connect to primary server
31:12:2022.13:07:12:342355 EZ 10.2.2.2:2000 SS: Failed to connect to primary server
31:12:2022.13:07:22:353806 EZ 10.2.2.2:2000 SS: Failed to connect to primary server
31:12:2022.13:07:32:366469 EZ 10.2.2.2:2000 SS: Failed to connect to primary server
31:12:2022.13:07:42:379485 OE ('10.2.2.2', 2000) 8096,0,0,0,0:ggm.root.,SOASERIAL;
31:12:2022.13:07:42:387332 RR None 8096,0,1,3,3;ggm.root.,SOASERIAL;
ggm.root. SOASERIAL 1 86400 -1;
ggm.root. NS servidor2.ggm.root. 86400 -1,
ggm.root. NS portatill.ggm.root. 86400 1,
ggm.root. NS grilo.ggm.root. 86400 2;
servidor2.ggm.root. A 10.2.2.2:2000 86400 -1,
portatill.ggm.root. A 10.1.1.1:2000 86400 -1,
grilo.ggm.root. A 10.4.4.1:2000 86400 -1;
31:12:2022.13:07:42:387412 ZT ('10.2.2.2', 2000) SS : Zone Transfer started
31:12:2022.13:07:42:387940 OE ('10.2.2.2', 2000) 37735,0,0,0,0:ggm.root.,AXFR;
31:12:2022.13:07:42:391325 RR None 37735,A,0,1,0,0;ggm.root.,AXFR;
ggm.root. AXFR 27 0 -1;
31:12:2022.13:07:42:392131 RP None 37735,A,0,1,0,0;ggm.root.,AXFR;
ggm.root. AXFR 27 0 -1;
31:12:2022.13:07:42:393976 ZT ('10.2.2.2', 2000) SS : Zone Transfer concluded successfully 0.00656s
```

Figura 10: Teste sem conexão.

Em seguida, desligou-se novamente o servidor primário, de modo a alterar a versão da base de dados. Mais uma vez, o servidor secundário foi capaz de esperar pelo servidor primário, realizando em seguida transferência de zona, pois o valor da versão não era aquele que ele tinha em memória.

```
31:12:2022.13:42:54:572671 EZ 10.2.2.2:2000 SS: Database is up to date
31:12:2022.13:43:54:635435 OE ('10.2.2.2', 2000) 33650,0,0,0,0:ggm.root.,SOASERIAL;
31:12:2022.13:43:54:643636 RR None 33650,A,0,1,3,3;ggm.root.,SOASERIAL;
ggm.root. SOASERIAL 2 86400 -1;
ggm.root. NS servidor2.ggm.root. 86400 -1,
ggm.root. NS portatill.ggm.root. 86400 1,
ggm.root. NS grilo.ggm.root. 86400 2;
servidor2.ggm.root. A 10.2.2.2:2000 86400 -1,
portatill.ggm.root. A 10.1.1.1:2000 86400 -1,
grilo.ggm.root. A 10.4.4.1:2000 86400 -1;
31:12:2022.13:43:54:643879 ZT ('10.2.2.2', 2000) SS : Zone Transfer started
31:12:2022.13:43:54:644384 OE ('10.2.2.2', 2000) 12381,0,0,0,0:ggm.root.,AXFR;
31:12:2022.13:43:54:649039 RR None 12381,A,0,1,0,0;ggm.root.,AXFR;
ggm.root. AXFR 27 0 -1;
31:12:2022.13:43:54:650008 RP None 12381,A,0,1,0,0;ggm.root.,AXFR;
ggm.root. AXFR 27 0 -1;
31:12:2022.13:43:54:651973 ZT ('10.2.2.2', 2000) SS : Zone Transfer concluded successfully 0.00809s
```

Figura 11: Teste alteração da versão.

7.2 Modo iterativo

Para testar o modo iterativo, ligamos o servidor de resolução “Foca” (não suporta modo recursivo), um servidor de topo e o “Servidor2” (para o domínio “fcp”). O cliente foi executado com o seguinte comando:

```
python3 run_client.py 10.3.3.3:2000 fcp.root. MX R 20 shy
```

Figura 12: Comando executado pelo cliente.

Para responder a esta *query*, o servidor de resolução terá, primeiramente, de enviar a *query* ao servidor de topo disponível, recebendo as entradas NS do domínio em questão. De seguida, terá de perguntar, iterativamente, ao servidor primário do domínio “fcp” que, por sua vez, lhe irá devolver uma resposta autoritativa. Por fim, a resposta é enviada de volta para o cliente.

```
02:01:2023,20:59:29:815391 EV localhost conf-file-read dns/files/config_files/foca.txt
02:01:2023,20:59:29:816068 ST 127.0.0.1 2000 10 debug
02:01:2023,20:59:29:816548 EV localhost log-file-create dns/files/log_files/all.log
02:01:2023,21:02:16:254930 QR ('10.4.4.2', 50651) 60367,Q+R,0,0,0,0:fcp.root.,MX;
02:01:2023,21:02:16:255204 QE ('10.2.2.1', 2000) 60367,Q,0,0,0,0:fcp.root.,MX;
02:01:2023,21:02:16:257250 RR ('10.2.2.1', 39342) 60367,,1,0,3,3:fcp.root.,MX;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
02:01:2023,21:02:16:257332 RP ('10.2.2.2', 2000) 60367,,1,0,3,3:fcp.root.,MX;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
02:01:2023,21:02:16:258997 RR ('10.2.2.2', 43080) 60367,A,0,1,3,4:fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
02:01:2023,21:02:16:259136 RP ('10.4.4.2', 50651) 60367,,0,1,3,4:fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
```

Figura 13: Resposta à primeira *query*.

Foi testada também a funcionalidade da *cache* nos servidores de resolução. Como podemos ver nas imagens seguintes, a *query* do cliente é imediatamente respondida pelo SR, uma vez que o resultado já foi colocado em *cache*, após a primeira *query* ser respondida.

```
02:01:2023.21:02:16:259136 RP ('10.4.4.2', 50651) 60367,,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
02:01:2023.21:02:18:598308 QR ('10.4.4.2', 32981) 47712,Q+R,0,0,0,0;fcp.root.,MX;
02:01:2023.21:02:18:598597 RP ('10.4.4.2', 32981) 47712,,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
```

Figura 14: Resposta à mesma *query*, com consulta em *cache*.

Verifica-se, portanto, que apenas uma *query* é enviada para o servidor autoritativo do domínio “fcp”.

```
02:01:2023.20:59:21:407231 EV localhost log-file-create dns/files/log_files/ggm.
log
02:01:2023.20:59:21:407732 EV localhost log-file-create dns/files/log_files/all.
log
02:01:2023.20:59:21:408539 EV localhost db-file-read dns/files/db_files/fcp.db
02:01:2023.20:59:21:408833 EV localhost db-file-read dns/files/db_files/ggm.db
02:01:2023.21:02:16:258011 RR ('10.3.3.3', 56397) 60367,,1,0,3,3;fcp.root.,MX;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
02:01:2023.21:02:16:258382 RP ('10.3.3.3', 56397) 60367,A,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
```

Figura 15: *Queries* recebidas pelo servidor autoritativo.

7.3 Modo recursivo

O modo recursivo total assume que todos os servidores aceitam responder de forma recursiva às *queries* enviadas. Assim, para suportar o modo recursivo um servidor deverá ser executado da seguinte forma:

```
python3 dns/src/run_server.py dns/files/Servidor2/config.txt 2000 100 1
```

Figura 16: Execução de um servidor com a *flag* recursiva a 1.

Analizando o comportamento do servidor de resolução (neste caso a máquina “Foca” com o IP 10.3.3.3), verificamos que este recebe a *query* do cliente e envia-a para o servidor “Root”, recebendo, mais tarde, a resposta deste e enviando-a de volta para o cliente.

```
01:01:2023.11:27:52:612712 QR ('10.4.4.2', 60438) 2369,Q+R,0,0,0,0;fcp.root.,MX;
01:01:2023.11:27:52:613646 QE ('10.2.2.1', 2000) 2369,Q+R,0,0,0,0;fcp.root.,MX;
01:01:2023.11:27:52:645198 RR ('10.2.2.1', 32855) 2369,R,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1;
fcp.root. NS golfinho.fcp.root. 86400 -1;
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1;
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1;
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1;
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
01:01:2023.11:27:52:658045 RP ('10.4.4.2', 60438) 2369,R,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1;
fcp.root. NS golfinho.fcp.root. 86400 -1;
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1;
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1;
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1;
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
```

Figura 17: Fluxo de *queries* e respostas no SR.

Quanto ao ST, este é responsável por grande parte do trabalho de resolução desta *query*. Temos noção que no mundo real esse não é o comportamento habitual, pois isso resultaria num *bottleneck* no nosso sistema. No entanto, decidimos incluir a possibilidade de resolução recursiva em todos os servidores, possibilitando uma maior customização de cada componente. Posto isto, o comportamento obtido é o desejado, visto que este servidor começa por receber a *query* recursiva vinda do SR, enviando-a de seguida para o servidor seguinte, que é o SP do domínio “fcp”. O ST espera pela resposta do SP, enviando-a depois para o SR, cumprindo a sua parte no processo recursivo.

```

01:01:2023.11:27:52:624447 RP ('10.2.2.2', 2000) 2369,R,1,0,3,3;fcp.root.,MX;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
01:01:2023.11:27:52:633172 RR ('10.2.2.2', 59396) 2369,R+A,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
01:01:2023.11:27:52:649773 RP ('10.3.3.3', 51865) 2369,R,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;

```

Figura 18: Fluxo de *queries* e respostas no ST.

Quanto ao SP, este apenas tem de processar e responder à *query* enviada pelo Servidor de Topo.

```

01:01:2023.11:27:01:792631 EV localhost db-file-read dns/files/db_files/fcp.db
01:01:2023.11:27:01:792973 EV localhost db-file-read dns/files/db_files/ggm.db
01:01:2023.11:27:52:621672 RR ('10.2.2.1', 32855) 2369,R,1,0,3,3;fcp.root.,MX;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;
01:01:2023.11:27:52:636443 RP ('10.2.2.1', 32855) 2369,R+A,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;

```

Figura 19: Fluxo de *queries* e respostas no SR.

Por fim, no cliente observamos a resposta desejada e a criação da *query* com a *flag* recursiva (R) ativa.

```

root@Cigarra:/tmp/pycore.40139/Cigarra.conf# cd dns/src/
<python3 run_client.py 10.3.3.3:2000 fcp.root. MX R 20 shy
2369,R,0,1,3,4;fcp.root.,MX;
fcp.root. MX ms1.fcp.root. 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1,
fcp.root. NS golfinho.fcp.root. 86400 -1,
fcp.root. NS vespa.fcp.root. 86400 -1;
ms1.fcp.root. A 10.4.4.2:2000 86400 -1,
servidor2.fcp.root. A 10.2.2.2:2000 86400 -1,
golfinho.fcp.root. A 10.3.3.2:2000 86400 -1,
vespa.fcp.root. A 10.4.4.3:2000 86400 -1;root@Cigarra:/tmp/p

```

Figura 20: Resposta obtida.

7.4 Domínios por defeito

7.4.1 Servidor de resolução

Para testar o funcionamento dos domínios por defeito nos servidores de resolução, elaboramos a seguinte *query* num cliente. O servidor alvo é um servidor de resolução (endereço 10.1.1.4), cujos domínios por defeito contêm uma referência para o servidor primário do domínio “cc.ggm“. Assim, é de esperar que não haja qualquer troca de mensagens entre o servidor de resolução e o ST.

```
python3 run_client.py 10.1.1.4:2000 cc.ggm.root. MX R 20 shy
```

Figura 21: *Query* enviada pelo cliente.

De facto, podemos ver que o servidor de resolução envia a *query* diretamente para o servidor autoritativo do domínio da *query*.

```
root@Portatil4:/tmp/pycore.40879/Portatil4.conf# bash run.sh
01:01:2023,14:55:31:132874 EV localhost conf-file-read dns/files/config_files/portatil4.txt
01:01:2023,14:55:31:133660 ST 127.0.0.1 2000 10 debug
01:01:2023,14:55:31:134128 EV localhost log-file-create dns/files/log_files/all.log
01:01:2023,14:56:03:228588 QR ('10.1.1.1', 40780) 65369,Q+R,0,0,0;cc.ggm.root.,MX;
01:01:2023,14:56:03:232560 QE ('10.1.1.2', 2000) 65369,Q+R,0,0,0;cc.ggm.root.,MX;
01:01:2023,14:56:03:240795 RR ('10.1.1.2', 60414) 65369,A,0,1,3,4;cc.ggm.root.,MX;
cc.ggm.root. MX ms1.cc.ggm.root. 86400 -1;
cc.ggm.root. NS portatil2.cc.ggm.root. 86400 -1,
cc.ggm.root. NS golfinho.cc.ggm.root. 86400 -1,
cc.ggm.root. NS cigarra.cc.ggm.root. 86400 -1;
ms1.cc.ggm.root. A 10.4.4.2:2000 86400 -1,
portatil2.cc.ggm.root. A 10.1.1.2:2000 86400 -1,
golfinho.cc.ggm.root. A 10.3.3.2:2000 86400 -1,
cigarra.cc.ggm.root. A 10.4.4.2:2000 86400 -1;
01:01:2023,14:56:03:257613 RP ('10.1.1.1', 40780) 65369,R,0,1,3,4;cc.ggm.root.,MX;
cc.ggm.root. MX ms1.cc.ggm.root. 86400 -1;
cc.ggm.root. NS portatil2.cc.ggm.root. 86400 -1,
cc.ggm.root. NS golfinho.cc.ggm.root. 86400 -1,
cc.ggm.root. NS cigarra.cc.ggm.root. 86400 -1;
ms1.cc.ggm.root. A 10.4.4.2:2000 86400 -1,
portatil2.cc.ggm.root. A 10.1.1.2:2000 86400 -1,
golfinho.cc.ggm.root. A 10.3.3.2:2000 86400 -1,
cigarra.cc.ggm.root. A 10.4.4.2:2000 86400 -1;
```

Figura 22: Fluxo *queries* e respostas no SR.

Como podemos observar, o servidor de topo ainda continua à espera de *queries*, pelo que não houve qualquer tipo de ligação entre este servidor e o servidor de resolução.

```
root@Servidor1:/tmp/pycore.40879/Servidor1.conf# bash run.sh
01:01:2023.14:55:49:794202 EV localhost conf-file-read dns/files/config_files/se
rvidor1.txt
01:01:2023.14:55:49:794743 ST 127.0.0.1 2000 10 debug
01:01:2023.14:55:49:795065 EV localhost log-file-create dns/files/log_files/root
.log
01:01:2023.14:55:49:795372 EV localhost log-file-create dns/files/log_files/all.
log
01:01:2023.14:55:49:796260 EV localhost db-file-read dns/files/db_files/root.db
```

Figura 23: Comportamento do ST.

7.4.2 Servidor primário/secundário

Para testar o comportamento de um servidor primário/secundário face aos domínios por defeito, foi enviada uma *query* de domínio “ggm” para o servidor “Vespa”, servidor secundário do domínio “fcp”, que contém domínios por defeito no seu ficheiro de configuração.

```
02:01:2023.21:28:02:972695 EV localhost log-file-create dns/files/log_files/all.
log
02:01:2023.21:28:02:977222 QE ('10.2.2.2', 2000) 31295,Q,0,0,0,0;fcp.root.,SOASE
RIAL:
02:01:2023.21:28:02:979187 RR None 31295,A,0,1,3,3;fcp.root.,SOASERIAL;
fcp.root. SOASERIAL 1 86400 -1;
fcp.root. NS servidor2.fcp.root. 86400 -1;
fcp.root. NS golfinho.fcp.root. 86400 -1;
fcp.root. NS vespa.fcp.root. 86400 -1;
servidor2.fcp.root. A 10.2.2.2;2000 86400 -1;
golfinho.fcp.root. A 10.3.3.2;2000 86400 -1;
vespa.fcp.root. A 10.4.4.3;2000 86400 -1;
02:01:2023.21:28:02:981634 ZT ('10.2.2.2', 2000) SS : Zone Transfer started
02:01:2023.21:28:02:982013 QE ('10.2.2.2', 2000) 31491,Q,0,0,0,0;fcp.root.,AXFR;
02:01:2023.21:28:02:982907 RR None 31491,A,0,1,0,0;fcp.root.,AXFR;
fcp.root. AXFR 24 0 -1;
02:01:2023.21:28:02:983613 RP None 31491,A,0,1,0,0;fcp.root.,AXFR;
fcp.root. AXFR 24 0 -1;
02:01:2023.21:28:02:985371 ZT ('10.2.2.2', 2000) SS : Zone Transfer concluded su
ccessfully 0.00373s
02:01:2023.21:28:35:060703 QR ('10.1.1.2', 33009) 38651,Q+R,0,0,0,0;ggm.root.,MX;
02:01:2023.21:28:35:060914 TO Server has no permission to attend the query domain!
```

Figura 24: Fluxo *queries* e respostas no SS.

Como esperado, o cliente é informado do *timeout* que ocorreu, visto que o servidor não tem autorização para responder à *query* com o domínio pretendido.

7.5 Prioridades

De modo a testar a funcionalidade das prioridades que foi implementada, enviamos uma *query* com domínio “ggm” a um servidor de resolução. Este SR recorre ao servidor de topo, visto que não tem informação na *cache* e, posteriormente, vai contactar o servidor de domínio “ggm” com maior prioridade (10.1.1.1), que neste caso é o que tem menor valor.

```
02:01:2023,21:51:58:289941 OE ('10.2.2.1', 2000) 41559,0,0,0,0;ggm.root.,MX;
02:01:2023,21:51:58:291541 RR ('10.2.2.1', 33658) 41559,,1,0,3,3;ggm.root.,MX;
ggm.root. NS servidor2.ggm.root. 86400 10,
ggm.root. NS portatil1.ggm.root. 86400 5,
ggm.root. NS grilo.ggm.root. 86400 15;
servidor2.ggm.root. A 10.2.2.2:2000 86400 -1,
portatil1.ggm.root. A 10.1.1.1:2000 86400 -1,
grilo.ggm.root. A 10.4.4.1:2000 86400 -1;
02:01:2023,21:51:58:291620 RP ('10.1.1.1', 2000) 41559,,1,0,3,3;ggm.root.,MX;
ggm.root. NS servidor2.ggm.root. 86400 10,
ggm.root. NS portatil1.ggm.root. 86400 5,
ggm.root. NS grilo.ggm.root. 86400 15;
servidor2.ggm.root. A 10.2.2.2:2000 86400 -1,
portatil1.ggm.root. A 10.1.1.1:2000 86400 -1,
grilo.ggm.root. A 10.4.4.1:2000 86400 -1;
```

Figura 25: Fluxo de *queries* do SR.

7.6 DNS reverso

É também possível testar a funcionalidade do *reverse* DNS, invocando a seguinte *query* no cliente.

```
<nf/dns/src# python3 run_client.py 10.3.3.3:2000 1.1.1.10.in-addr.reverse.root. PTR R 20 shy
38008,,0.2.0.1;1.1.1.10.in-addr.reverse.root.,PTR;
1.1.1.10.in-addr.reverse.root. PTR portatil1.10.in-addr.reverse.root. 86400 -1,
1.1.1.10.in-addr.reverse.root. PTR portatil1.ggm.root. 86400 -1;
portatil1.10.in-addr.reverse.root. A 10.1.1.1:2000 86400 -1;root@Servidor2:/tmp/pycore.41139/Servidor2.conf/dns/src#
```

Figura 26: Cliente no teste do DNS reverso.

A *query* é um pedido recursivo ao servidor de resolução, que irá percorrer a árvore do DNS até chegar ao servidor autoritativo do domínio “10.in-addr.reverse”, que irá responder da seguinte forma:

```
01:01:2023,23:45:07:532939 RR ('10.2.2.2', 60984) 23735,,1,0,1,1;1.1.1.10.in-addr.reverse.root.,PTR;
10.in-addr.reverse.root. NS portatil1.10.in-addr.reverse.root. 86400 -1;
portatil1.10.in-addr.reverse.root. A 10.1.1.1:2000 86400 -1;
01:01:2023,23:45:07:541008 RP ('10.2.2.2', 60984) 23735,,0.2.0.1;1.1.1.10.in-addr.reverse.root.,PTR;
1.1.1.10.in-addr.reverse.root. PTR portatil1.10.in-addr.reverse.root. 86400 -1,
1.1.1.10.in-addr.reverse.root. PTR portatil1.ggm.root. 86400 -1;
portatil1.10.in-addr.reverse.root. A 10.1.1.1:2000 86400 -1;
```

Figura 27: Resposta autoritativa no DNS reverso.

8 Trabalho desenvolvido

O trabalho desenvolvido foi construído à base de módulos bem definidos e independentes. Como tal, optamos por, a cada etapa do trabalho, distribuir tarefas de forma equilibrada por todos os elementos do grupo.

Numa primeira fase, o nosso grupo analisou, de forma cuidada, o âmbito e os requisitos do sistema que nos foi proposto implementar. Esta fase incluiu também o esboço da arquitetura do sistema e dos seus vários componentes.

Numa segunda fase, o grupo passou para a implementação da especificação consensualizada. Em paralelo, foi elaborado o relatório e o planeamento do ambiente de teste.

Uma representação do trabalho realizado encontra-se na tabela a seguir.

| Tarefa | Gabriela | Guilherme | Miguel |
|--|----------|-----------|--------|
| B.1 Arquitetura do Sistema | 33% | 33% | 33% |
| B.2 Modelo de Informação | 33% | 33% | 33% |
| B.3 Modelo de Comunicação | 33% | 33% | 33% |
| B.4 Planeamento do Ambiente de Teste | 33% | 33% | 33% |
| B.5 Implementação dos componentes | 33% | 33% | 33% |
| B.6 CL em modo debug e modo normal | 33% | 33% | 33% |
| B.7 Implementação do Ambiente de Teste | 33% | 33% | 33% |
| B.8 Relatório | 33% | 33% | 33% |

Tabela 7: Tabela de atividades e contribuições.

Consideramos, por isso, que o trabalho foi desenvolvido de forma bastante equilibrada por todos os elementos do grupo. Fazemos por isso um balanço positivo desta abordagem modular, pois contribuiu para uma boa distribuição das tarefas a realizar (para além de contribuir para uma boa manutenibilidade do código).

9 Conclusão

Em jeito de conclusão, este trabalho teve como propósito a implementação de um sistema de DNS próximo do que é encontrado na realidade. Esta parte prática proporcionou-nos uma oportunidade de entender, em detalhe, as funcionalidades e os detalhes de um sistema deste tipo. Consideramos, por isso, que este foi um excelente complemento para aquilo que aprendemos nas aulas teóricas.

Numa primeira fase do trabalho, dedicamo-nos sobretudo a aspetos mais estruturais do trabalho, realizando a especificação do sistema e implementando protótipos de clientes e servidores DNS. Foi também uma ótima forma de aprofundar o nosso conhecimento em mecanismos de comunicação inter-processos, em particular, *sockets*.

Na segunda fase, melhoramos os aspetos implementados na primeira fase, com especial destaque para os modos iterativo e recursivo de resolução de *queries*. Foi também implementada a comunicação normal (em binário) entre servidores. Na fase final, pudemos realizar alguns testes com o intuito de validar as funcionalidades desenvolvidas, tendo como ambiente de teste uma topologia Core.

O balanço final deste projeto é, por isso, bastante positivo, pois consideramos que atingimos todos os objetivos essenciais estipulados. Apesar de terem ficado por implementar funcionalidades mais específicas do DNS como a *cache* negativa, consideramos que o nosso projeto cumpre os requisitos necessários para servir de suporte a um sistema de DNS distribuído.

10 Lista de Siglas e Acrónimos

DNS - *Domain Name System*

SP - Servidor Primário

SS - Servidor Secundário

SR - Servidor de Resolução

ST - Servidor de topo

DD - Domínio por defeito

SDT - Servidor de domínio de topo

UDP - *User Datagram Protocol*

TCP - *Transmission Control Protocol*

TTL - *Time to Live*

PDU - *Protocol Data Unit*

11 Referências

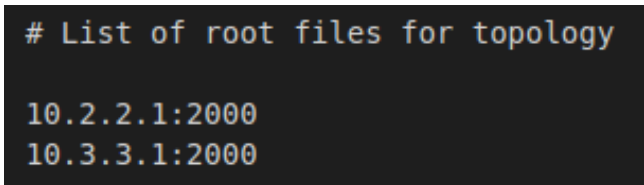
RFC 1034: Domain Names - Concepts and Facilities.

RFC 1035 : Domain Names - Implementation and Specification.

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

12 Anexos

12.1 Ficheiro com lista de servidores de topo



```
# List of root files for topology  
  
10.2.2.1:2000  
10.3.3.1:2000
```

Figura 28: Ficheiro com a lista de servidores de topo

12.2 Ficheiros de configuração

Servidor1

```
# Root server
root DB dns/files/db_files/root.db
root LG dns/files/log_files/root.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 29: Ficheiro de configuração da máquina “Servidor1”.

Servidor2

```
# Primary server for fcp domain
fcp.root DB dns/files/db_files/fcp.db
fcp.root SS 10.3.3.2:2000
fcp.root SS 10.4.4.3:2000
fcp.root LG dns/files/log_files/fcp.log

# Primary server for ggm domain
ggm.root DB dns/files/db_files/ggm.db
ggm.root SS 10.1.1.1:2000
ggm.root SS 10.4.1.1:2000
ggm.root LG dns/files/log_files/ggm.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 30: Ficheiro de configuração da máquina “Servidor2”.

Servidor3

```
# Secondary server for taremi.fcp domain
taremi.fcp.root SP 10.4.4.1:2000
taremi.fcp.root LG dns/files/log_files/taremi_fcp.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 31: Ficheiro de configuração da máquina “Servidor3”.

Orca

```
# Root server
root DB dns/files/db_files/root.db
root LG dns/files/log_files/root.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 32: Ficheiro de configuração da máquina “Orca”.

Golfinho

```
# Primary server for fcp domain
fcp.root DB dns/files/db_files/fcp.db
fcp.root SS 10.3.3.2:2000
fcp.root SS 10.4.4.3:2000
fcp.root LG dns/files/log_files/fcp.log

# Primary server for reverse domain
reverse.root DB dns/files/db_files/reverse_dns/reverse.db
reverse.root LG dns/files/log_files/reverse.log

# Secondary server for cc.ggm domain
cc.ggm.root SP 10.1.1.2:2000
cc.ggm.root LG dns/files/log_files/cc_ggm.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 33: Ficheiro de configuração da máquina “Golfinho”.

Grilo

```
# Primary server for taremi.fcp domain
taremi.fcp.root DB dns/files/db_files/taremi_fcp.db
taremi.fcp.root SS 10.1.1.3:2000
taremi.fcp.root SS 10.2.2.3:2000
taremi.fcp.root LG dns/files/log_files/taremi_fcp.log

# Secondary server for ggm domain
ggm.root SP 10.2.2.2:2000
ggm.root LG dns/files/log_files/ggm.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 34: Ficheiro de configuração da máquina “Grilo”.

Foca

```
# DNS Resolver
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 35: Ficheiro de configuração da máquina “Foca”.

Vespa

```
# Secondary server for fcp domain
fcp.root SP 10.2.2.2:2000
fcp.root DD 127.0.0.1
fcp.root DD 10.2.2.2:2000
fcp.root DD 10.3.3.2:2000
taremi.fcp.root DD 10.2.2.3:2000
taremi.fcp.root DD 10.1.1.3:2000
taremi.fcp.root DD 10.4.4.1:2000
fcp.root LG dns/files/log_files/fcp.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 36: Ficheiro de configuração da máquina “Vespa”.

Cigarra

```
# Secondary server for cc.ggm domain
cc.ggm.root SP 10.1.1.2:2000
cc.ggm.root LG dns/files/log_files/cc_ggm.log

# Primary server for in-addr.reverse domain
in-addr.reverse.root DB dns/files/db_files/reverse_dns/in-addr_reverse.db
in-addr.reverse.root LG dns/files/log_files/in-addr_reverse.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 37: Ficheiro de configuração da máquina “Cigarra”.

Portatil1

```
# Primary server for 10.in-addr.reverse domain
10.in-addr.reverse.root DB dns/files/db_files/reverse_dns/10_in-addr_reverse.db
10.in-addr.reverse.root LG dns/files/log_files/10_in-addr_reverse.log

# Secondary server for ggm domain
ggm.root SP 10.2.2.2:2000
ggm.root LG dns/files/log_files/ggm.log

root ST dns/files/rootservers.db
all LG dns/files/log_files/all.log
```

Figura 38: Ficheiro de configuração da máquina “Portatil1”.

Portatil2

```
# Primary server for cc.ggm domain
cc.ggm.root DB dns/files/db_files/cc_ggm.db
cc.ggm.root SS 10.4.4.2:2000
cc.ggm.root SS 10.3.3.2:2000
cc.ggm.root LG dns/files/log_files/cc_ggm.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 39: Ficheiro de configuração da máquina “Portatil2”.

Portatil3

```
# Secondary server for fcp domain
taremi.fcp.root SP 10.4.4.1:2000
taremi.fcp.root LG dns/files/log_files/taremi_fcp.log

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 40: Ficheiro de configuração da máquina “Portatil3”.

Portatil4

```
# DNS Resolver
taremi.fcp.root DD 10.4.4.1:2000
taremi.fcp.root DD 10.1.1.3:2000
taremi.fcp.root DD 10.2.2.3:2000
cc.ggm.root DD 10.1.1.2:2000
cc.ggm.root DD 10.4.4.2:2000
cc.ggm.root DD 10.3.3.2:2000

all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 41: Ficheiro de configuração da máquina “Portatil4”.

12.3 Ficheiros de base de dados

Root server

```
# DNS database file for root server

@ DEFAULT root.
TTL DEFAULT 86400

# Config data
@ SOAADMIN mike\.ggm.root. TTL
@ SOASERIAL 1 TTL

# Name servers
@ NS servidor1 TTL
@ NS orca TTL

ggm NS servidor2.ggm TTL 10
ggm NS portatil1.ggm TTL 5
ggm NS grilo.ggm TTL 15

fcp NS servidor2.fcp TTL
fcp NS golfinho.fcp TTL
fcp NS vespa.fcp TTL

reverse NS golfinho.reverse TTL

# Addresses
servidor2.ggm A 10.2.2.2:2000 TTL
portatil1.ggm A 10.1.1.1:2000 TTL
grilo.ggm A 10.4.4.1:2000 TTL

servidor2.fcp A 10.2.2.2:2000 TTL
golfinho.fcp A 10.3.3.2:2000 TTL
vespa.fcp A 10.4.4.3:2000 TTL

golfinho.reverse A 10.3.3.2:2000 TTL

servidor1 A 10.2.2.1:2000 TTL
orca A 10.3.3.1:2000 TTL
```

Figura 42: Ficheiro de base de dados do servidor “root”.

Domínio ggm

```
# DNS database file domain .ggm

@ DEFAULT ggm.root.
TTL DEFAULT 86400

@ SOASP servidor2.ggm.root. TTL
@ SOAADMIN admin\ggm.root. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 60 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

# Name Servers
@ NS servidor2 TTL
@ NS portatil1 TTL 1
@ NS grilo TTL 2

# cc.ggm subdomain
cc.@ NS portatil2.cc.@ TTL
cc NS golfinho.cc TTL
cc NS cigarra.cc TTL

@ MX ms1.@ TTL
@ MX ms2.@ TTL

# Addresses
servidor2 A 10.2.2.2:2000 TTL
portatil1 A 10.1.1.1:2000 TTL
grilo A 10.4.4.1:2000 TTL

portatil2.cc A 10.1.1.2:2000 TTL
golfinho.cc A 10.3.3.2:2000 TTL
cigarra.cc A 10.4.4.2:2000 TTL

ms1 A 10.3.3.1:2000 TTL
ms2 A 10.2.2.3:2000 TTL

maquina1 CNAME servidor2 TTL
maquina2 CNAME portatil1 TTL
maquina3 CNAME grilo TTL
mail1 CNAME ms1 TTL
mail2 CNAME ms2 TTL
```

Figura 43: Ficheiro de base de dados do domínio “ggm”.

Domínio fcp

```
# DNS database file for domain .fcp

@ DEFAULT fcp.root.
TTL DEFAULT 86400

@ SOASP servidor2.fcp.root. TTL
@ SOADMIN admin.\fcp.root. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS servidor2 TTL
@ NS golfinho TTL
@ NS vespa TTL

taremi.@ NS grilo.taremi TTL
taremi NS servidor3.taremi TTL
taremi NS portatil3.taremi TTL

# MAIL SERVERS
@ MX ms1 TTL

# Addresses
servidor2 A 10.2.2.2:2000 TTL
golfinho A 10.3.3.2:2000 TTL
vespa A 10.4.4.3:2000 TTL

grilo.taremi A 10.4.4.1:2000 TTL
servidor3.taremi A 10.2.2.3:2000 TTL
portatil3.taremi A 10.1.1.3:2000 TTL

ms1 A 10.4.4.2:2000 TTL

maquina1 CNAME servidor2 TTL
maquina4 CNAME golfinho TTL
maquina5 CNAME vespa TTL
mail1 CNAME ms1 TTL
```

Figura 44: Ficheiro de base de dados do domínio “fcp”.

Domínio cc.ggm

```
# DNS database file for domain cc.ggm

@ DEFAULT cc.ggm.root.
TTL DEFAULT 86400

@ SOASP portatil2.cc.ggm.root. TTL
@ SOADMIN admin\cc.ggm.root. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS portatil2 TTL
@ NS golfinho TTL
@ NS cigarra TTL

# MAIL SERVERS
@ MX ms1.@ TTL

# Addresses
portatil2 A 10.1.1.2:2000 TTL
cigarra A 10.4.4.2:2000 TTL
golfinho A 10.3.3.2:2000 TTL

ms1 A 10.4.4.2:2000 TTL

sr2. A 10.1.1.4:2000 TTL

resolver2. CNAME sr2. TTL
maquina8 CNAME portatil2 TTL
maquina4 CNAME golfinho TTL
maquina9 CNAME cigarra TTL
```

Figura 45: Ficheiro de base de dados do domínio “cc.ggm”.

Domínio taremi.fcp

```
# DNS database file for domain taremi.fcp

@ DEFAULT taremi.fcp.root.
TTL DEFAULT 86400

@ SOASP grilo.taremi.fcp.root. TTL
@ SOADMIN admin\.taremi.fcp.root. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS grilo TTL
@ NS portatil3 TTL
@ NS servidor3 TTL

# MAIL SERVERS
@ MX ms1.@ TTL

# Addresses
grilo A 10.4.4.1:2000 TTL
portatil3 A 10.1.1.3:2000 TTL
servidor3 A 10.2.2.3:2000 TTL

ms1 A 10.3.3.1:2000 TTL
sr1. A 10.3.3.3:2000 TTL

maquina3 CNAME grilo TTL
maquina6 CNAME portatil3 TTL
maquina7 CNAME servidor3 TTL
mail1 CNAME ms1 TTL
resolver1. CNAME sr1. TTL
```

Figura 46: Ficheiro de base de dados do domínio “taremi.fcp”.

Domínio reverse

```
# DNS database file for reverse domain.

@ DEFAULT reverse.root.
TTL DEFAULT 86400

# Config data
@ SOASP golfinho.reverse.root. TTL
@ SOADMIN gabs\.\ggm.root. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS golfinho TTL
in-addr NS cigarra.in-addr TTL

# Addresses
golfinho A 10.3.3.2:2000 TTL
cigarra.in-addr A 10.4.4.2:2000 TTL
```

Figura 47: Ficheiro de base de dados do domínio “reverse”.

Domínio in-addr.reverse

```
# DNS database file for .in-addr domain.

@ DEFAULT in-addr.reverse.root.
TTL DEFAULT 86400

# Config data
@ SOASP cigarra.in-addr.reverse.root. TTL
@ SOADMIN gabs\.\ggm.root. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS cigarra TTL
10 NS portatill.10 TTL

# Addresses
cigarra A 10.4.4.2:2000 TTL
portatill.10 A 10.1.1.1:2000 TTL
```

Figura 48: Ficheiro de base de dados do domínio “in-addr.reverse”.

Domínio 10.in-addr.reverse

```
# DNS database file for domain 10.in-addr.reverse

@ DEFAULT 10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP portatil1.10.in-addr.reverse.root. TTL
@ SOAADMIN gabs\ggm.root. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS portatil1 TTL

# Addresses
portatil1 A 10.1.1.1:2000 TTL

# PTR Records
# 10.1.1
1.1.1 PTR portatil1.10.in-addr.reverse.root. TTL
1.1.1 PTR portatil1.ggm.root. TTL

2.1.1 PTR portatil2.cc.ggm.root. TTL

3.1.1 PTR portatil3.taremi.fcp.root. TTL

4.1.1 PTR sr2. TTL

# 10.2.2
1.2.2 PTR servidor1.root. TTL

2.2.2 PTR servidor2.fcp.root. TTL
2.2.2 PTR servidor2.ggm.root. TTL

3.2.2 PTR servidor3.taremi.fcp.root. TTL
3.2.2 PTR ms2.ggm.root. TTL

# 10.3.3
1.3.3 PTR orca.root. TTL
1.3.3 PTR ms1.taremi.fcp.root. TTL
1.3.3 PTR ms1.ggm.root. TTL

2.3.3 PTR golfinho.fcp.root. TTL
2.3.3 PTR golfinho.reverse.root. TTL
2.3.3 PTR golfinho.cc.ggm.root. TTL
```

Figura 49: Ficheiro de base de dados do domínio “10.in-addr.reverse”.