Universidade do Minho
Escola de Engenharia

Implementação de Sistema DNS

Trabalho Prático 2 - Fase 1

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

novembro, 2022

Conteúdo

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: SP (Servidor Primário), SS (Servidor Secundário), SR (Servidor de Resolução) - apenas será implementado na segunda fase - e CL (Cliente). 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*). Por outro lado, os ST têm o mesmo comportamento que o SP mas a sua base de dados apenas inclui informação dos SDT respetivos para cada entrada - domínio de topo -, portanto este tipo de servidor só pode responder com os nomes e os endereços IP dos seus SS e do seu SP.

Em suma, tanto os ST como os SDT serão implementados com o mesmo componente que implementa um SP ou um SS.

2.2 Requisitos funcionais

2.2.1 *Parsing*

- O sistema deve suportar a leitura de ficheiros.
- Utilizado para retirar as caraterísticas de um servidor, nomeadamente configuração e base de dados.
- As linhas vazias e os comentários 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*.

2.2.2 Cliente

- O sistema deve implementar um programa que recebe como argumentos o endereço do servidor DNS, o *NAME* e o *VALUE OF TYPE* de uma *query* e, possivelmente, o modo recursivo;
- 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 tem de apresentar uma opção de *debug*, passando também a registar a sua atividade no *stdout*;
- O registo de informação em *cache* deve ser efetuado sempre que se receber uma resposta recebida a uma *query* feita anteriormente;

- Caso não encontre resposta para a *query*, o servidor deverá saber se contacta o servidor de topo ou não responde.

2.2.4 *Cache*

- O sistema deve implementar um mecanismo de *cache*;
- A *cache* vai ser implementada nos servidores, de modo a aumentar tempos de acesso aos dados;
- Os servidores vão guardar, na *cache*, informação do seu ficheiro de base de dados, excetuando os servidores de resolução e informação proveniente de *queries*;
- 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 na base de dados do seu servidor primário. No contexto deste sistema, o servidor primário vai receber 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.

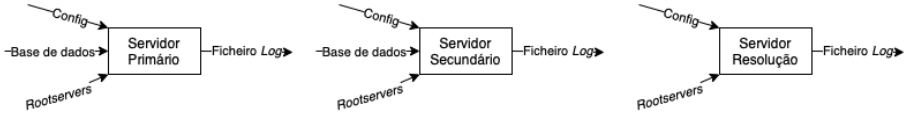

Figura 1: Componentes de *software*.

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

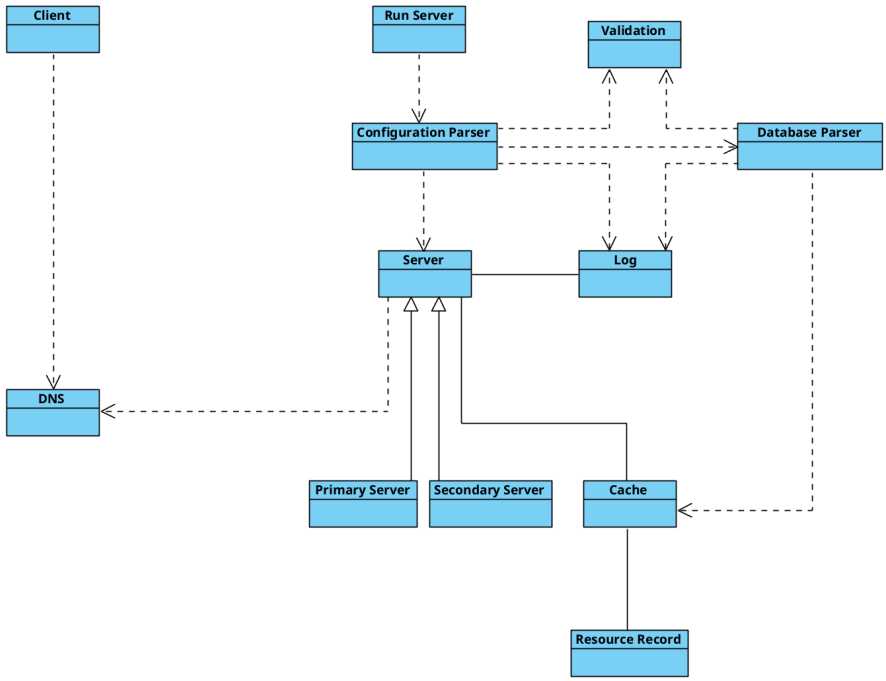

Figura 2: Arquitetura do sistema.

Para a leitura de ficheiros, criamos módulos de *parse* que recorrem a algumas funções de validação de parâmetros localizadas no módulo "Validation".

Tomamos a decisão de implementar o funcionamento dos servidores através de uma hierarquia de classes, uma vez que todos os servidores vão ter comportamentos semelhantes e nos permite organizar melhor o código. O cliente encontra-se no módulo "Client". Tanto os servidores como os clientes vão-se servir do módulo "DNS" 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".

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 descodificação de um PDU enviado do endereço indicado. Pode ser ainda reportado o que foi possível descodificar 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 numa cache de um servidor. Consoante o tipo, 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* aplicacionais 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;
- **SOAADMIN** - 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 deiar 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 PDU

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

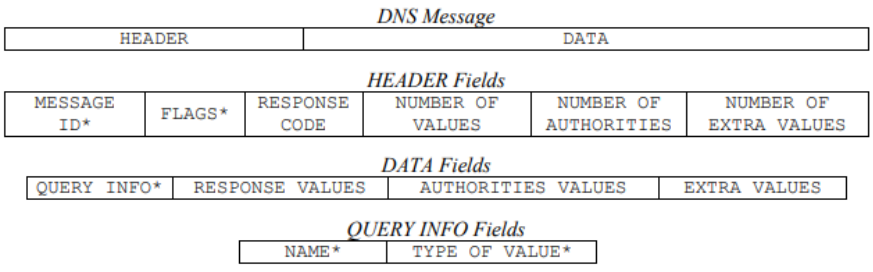

Figura 3: PDU da mensagem DNS.

Os vários campos que constituem a mensagem DNS podem-se dividir em 2 componentes: *header*, que contém metadados sobre o que é transmitido e *data* que 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 descodificada 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)*.

3.5 Mecanismo de codificação da unidade de dados

Numa primeira fase do trabalho e de modo a facilitar o *debug* dos vários programas, todos os campos foram representados como *strings*, 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, seja por TCP, o que reduz a performance do nosso sistema.

Por isso, um dos requisitos do nosso trabalho passa por implementar 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 2 *bytes* para representar todos estes valores. Por exemplo, 0x00 00 representa o valor 1 e assim sucessivamente.
- ***Flags*** - Neste trabalho temos a necessidade de representar 3 *flags*: Q, R e A. Seriam assim suficientes apenas 3 *bits* (um para cada *flag*) que poderiam estar ativos ou não. No entanto, decidimos usar 6 *bits* de modo

a poder suportar eventuais funcionalidades futuras e também para que o cabeçalho tenha um tamanho múltiplo de 8 *bits* (1 *byte*).

- ***Response code*** - Há a necessidade de representar 4 códigos de resposta (0 a 3), pelo que 2 *bits* são suficientes.
- ***Number of values, Number of authorities, Number of 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.
- ***Number of authorities*** - Este campo pode ter um valor entre 0 e 255. São por isso suficientes 8 *bits* (1 *byte*) para representar todos os valores possíveis.
- ***Number of extra values*** - Este campo pode 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**
Message ID	16 (2 *bytes*)
Flags	6
Response Code	2
Number of Values	8 (1 *byte*)
Number of Authorities	8 (1 *byte*)
Number of Extra Values	8 (1 *byte*)

Em suma, através do método utilizado, podemos codificar o cabeçalho de uma mensagem DNS com 6 *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. Como o campo *Name* é uma *string*, utilizaremos 1 *byte* para indicar o tamanho da *string*. É importante referir que, através do número de *response values*, *authorities values* e *extra values* conseguimos saber a quantidade de elementos a desserializar.

Campo	**Nº de *bits***
Name	variável
Type of Value	4
Response Values	variável
Authorities Values	variável
Extra Values	variável

Os campos *response values*, *authorities values* e *extra values* vão conter listas de registos da base de dados dos SPs. Assim, temos também de definir um esquema de codificação para os registos. O esquema da codificação é o apresentado na tabela.

Campo	**Nº de bits**
Name	variável
Type	4 *bits*
Value	variável
TTL	32 (4 *bytes*)
Prioridade	8 (1 *byte*)

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

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

Esta é a codificação que minimiza o número de *bits* usados na representação de um *ResourceRecord*.

Para além disso, por serem de comprimento variável, os campos *Name* e *Value* terão de ser precedidos por um valor inteiro (1 *byte*) que indica o tamanho da *string* que é enviada. Um tamanho de 0 indica que está a ser iniciada uma nova lista, pelo que o carater seguinte é o comprimento do campo *Name* da lista seguinte.

4 Modelo de Comunicação

4.1 Descrição introdutória do modelo comunicacional

Os 4 componentes especificados deverão poder comunicar entre si através de mensagens DNS, detalhadas na secção 3. Desta forma, o fio de execução principal de cada servidor é responsável pela criação de um *socket* UDP na porta especificada no ficheiro de configuração. Este *socket* acolherá as várias mensagens enviadas para esse componente do sistema.

De um modo geral, no modo iterativo, ao receber uma *query* de um dado endereço IP, o servidor verifica se tem na sua *cache* os dados pedidos. Se possuir uma resposta autoritativa a essa *query*, responde com os dados requeridos. Caso a *query* seja relativa a um subdomínio do servidor questionado, o servidor reencaminha a *query* para o servidor do subdomínio procurado. Num outro caso, se o servidor recebe uma *query* sobre um domínio que não é o seu, vai reencaminhar a mensagem para um *root server*. Excecionalmente, não havendo qualquer referência aos dados pedidos na *query*, o servidor simplesmente não responde (*timeout*).

Nesta sistema de DNS, os servidores secundários terão 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.

4.2 Interações

Nesta primeira fase do trabalho implementamos um primeiro protótipo do sistema DNS, dando a possibilidade do utilizador enviar *queries* para servidores primários e secundários de um dado domínio (nesta fase sem o auxílio de servidor de resolução). No entanto, o sistema real que implementa o DNS é muito mais complexo, havendo interações quer entre clientes e servidores, quer apenas entre servidores. Todas essas interações são explicadas em seguida, sendo algumas delas implementadas apenas na segunda fase.

4.2.1 Interações envolvendo *query* do cliente

O cliente começa por formular uma *query* com o domínio e o seu tipo, que vai ser enviada para um servidor,

Caso esta *query* seja enviada para um servidor que seja autoridade para esse domínio (servidor primário e secundários desse domínio), o servidor irá procurar a resposta à *query* na sua *cache*, de modo a ser enviada para o cliente. Excecionalmente, se não houver resposta na *cache*, estamos perante uma situação de *cache miss* e, consequente, *timeout*.

No caso de um servidor receber uma *query* para a qual não tem autoridade sobre o seu domínio, primeiramente, vai procurar a resposta à *query* na sua *cache*. Se ocorrer um *hit*, envia a resposta para o endereço que enviou a *query*. Caso contrário, vai aos seus domínios por defeito procurar o endereço IP correspondente ao domínio da *query*. Por conseguinte, se existir um domínio por defeito vai reencaminhar a *query* para esse servidor (Servidor Domínio de Topo). Caso contrário, vai contactar os *root servers*, enviando uma *query* a perguntar o endereço IP correspondente ao domínio.

Assumindo que todas as *caches* estão vazias, as interações acima podem ser descritas no seguinte exemplo:

Um cliente envia uma *query* a um servidor de resolução, procurando saber o endereço IP de um dado subdomínio (por exemplo "google.com"). Havendo uma *cache miss* neste primeiro servidor, o *DNS Resolver* envia uma *query* a um dos servidores de topo, perguntando qual o endereço IP do servidor primário (Servidor de Domínio de Topo) correspondente ao domínio ".com". A resposta é enviada ao servidor de resolução. Em seguida, este envia uma nova *query*, desta vez com destino ao servidor primário do domínio ".com", inquirindo qual o endereço IP correspondente ao subdomínio "google.com". Este tem acesso à base de dados com a informação correspondente, pelo que a resposta é enviada para o servidor de resolução, que envia ao cliente o endereço IP pretendido.

4.2.2 Interações entre servidores

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 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, indicando o nome completo do domínio a atualizar.

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:

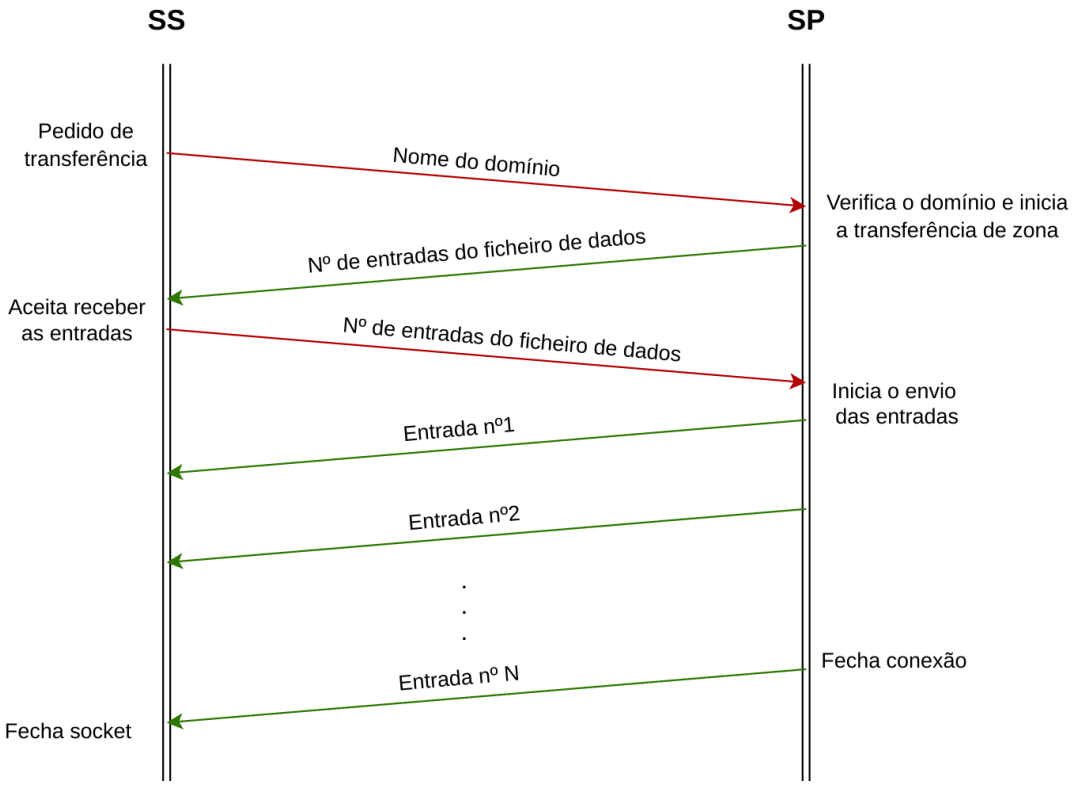

Figura 4: Representação do processo de transferência de zona.

5 Implementação

5.1 Cliente

Nesta fase do trabalho, 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. É importante referir que caso a porta não seja indicada no *input*, o valor *default* é 5353.

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*.

Um cliente deve ser executado da seguinte forma:

```
python dns/src/client.py 10.2.2.1:2000 mike.ggm. A
```

Figura 5: Execução do programa cliente.

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

5.2 Servidores

Nesta fase do trabalho, optamos por uma hierarquia de classes para a implementação dos servidores, uma vez que o comportamento dos servidores vai ser muito semelhante, apenas difere na transferência de zona. Como tal, o processo de receber *queries* por UDP é feito na superclasse e a transferência de zona é implementada nas subclasses (servidor primário e secundário).

O programa que corre o servidor recebe como argumentos a diretoria do ficheiro de configuração, a porta de atendimento, o valor de *timeout* e o modo de funcionamento. Com esta informação criamos o objeto correspondente ao tipo de servidor inferido no *parsing* do ficheiro. Também é feito o *parsing* do ficheiro de dados, cuja diretoria se encontra no ficheiro de configuração, e é criado o objeto *Cache* no servidor, objeto este que contém a informação necessária para responder às *queries*. Finalizada a configuração do servidor, são criadas duas *threads* que vão correr o método da transferência de zona (definido por cada subclasse) e o método que recebe *queries* a partir de um *socket* UDP.

5.2.1 Receber e responder a *queries*

Para o servidor responder a *queries*, é criado um *socket* UDP, na porta de atendimento passada como argumento na execução do componente, que vai receber as *queries* enviadas pelos clientes. Optamos por implementar o modelo *thread-per-connection*, o que nos permite criar uma *thread* por mensagem recebida. Deste modo, o servidor consegue responder a múltiplas mensagens em simultâneo.

Cada *thread* criada executará um método que vai interpretar a mensagem recebida. Este método verifica se a mensagem é efetivamente uma *query* ou uma resposta. No caso de ser uma *query*, o servidor responde ao pedido com base na informação que possui na sua *cache*, verificando inclusive os vários *alias* para o nome passado na *query*. No caso de ser uma resposta, por enquanto, é registado um *log*, uma vez que o comportamento perante esta situação apenas será implementado na próxima fase do trabalho.

5.2.2 Realizar transferência de zona

Na transferência de zona, ambos os servidores (primário e secundário) possuem comportamentos diferentes e, por isso, é definido o método nas respetivas subclasses. Toda esta operação vai ser realizada em TCP.

Servidor primário

No programa que corre um servidor, assim que é criada a *thread* responsável pela transferência de zona, cria-se o *socket* TCP que vai ser colocado à escuta (modo *listening*). Mais uma vez, sempre que uma ligação é estabelecida, ou seja, uma transferência de zona é pedida, cria-se uma *thread* que vai executar o processo em si. Isto permite-nos ter várias transferências de zona a ocorrer em simultâneo.

O servidor primário bloqueia à espera de mensagens na ligação estabelecida e vai realizar todo o processo de responder à *query* da versão da base de dados e enviar todas as entradas diretamente da *cache*.

Servidor secundário

Nesta primeira fase, optamos já por implementar o "SOAREFRESH". Deste modo, o método que é corrido na *thread* correspondente à transferência de zona, no programa principal, possui um *loop* infinito que realiza o processo da transferência e, após isso, a *thread* adormece "SOAREFRESH" segundos. Com isto, a transferência de zona vai ser efetuada no intervalo de tempo do "SOAREFRESH".

Neste tipo de servidor, o processo da transferência de zona consiste em formular a *query* da versão da base de dados, a *query* AXFR para iniciar a transferência e a receção de todas as entradas vindas do servidor primário.

5.3 *Cache*

Nesta fase do trabalho, decidimos implementar já a *cache* nos servidores, algo que nos facilitou a nível das estruturas usadas, uma vez que a base de dados é guardada na cache. Assim, temos só uma estrutura que guarda todos os dados, quer estes sejam relativos à base de dados ou a *caching*.

A *cache* é definida tal como está descrita no enunciado do trabalho prático. Utilizamos uma lista que vai guardar todos os registos sob a forma de "ResourceRecord", classe que é definida para representar um registo. Cada entrada da *cache* guarda um nome do domínio, tipo do valor, valor, tempo de validade, prioridade, origem da entrada, *TimeStamp* 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.

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, possuíndo também uma rede *backbone* de *routers* interligados.

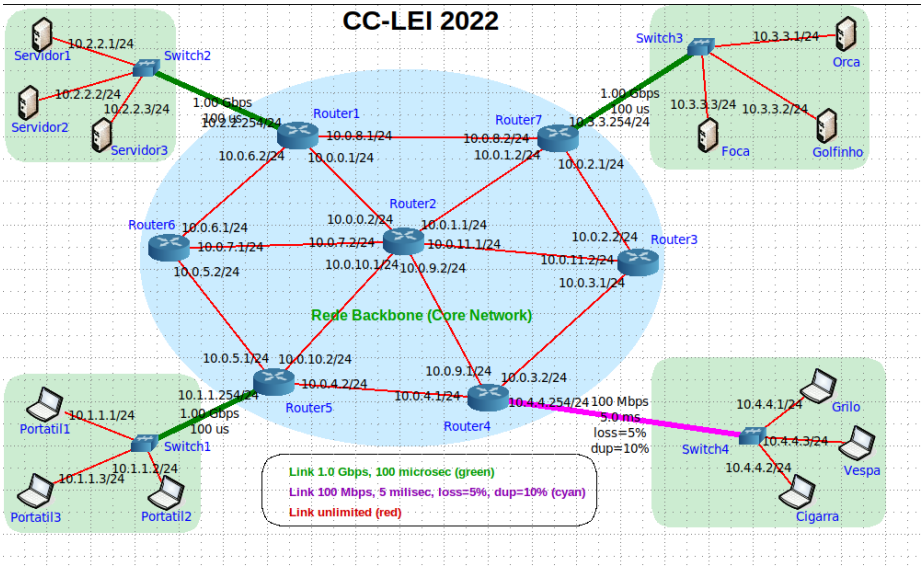

Figura 6: Topologia a utilizar

O nosso sistema de teste de DNS será constituído pelos seguintes componentes:

- **2 servidores de topo** : o 1º é implementado pelo "Servidor1" na porta 1998 e o 2º é implementado pelo servidor "Orca" na porta 1999. 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 1º é o domínio .ggm (SP - "Servidor1" na porta 2000, SS1 - "Orca" na porta 2001, SS2 - "Grilo" na porta 2002) e o 2º o domínio .fcp (SP - "Servidor2" na porta 2003, SS1 - "Golfinho" na porta 2004, SS2 - "Vespa" na port 2005).
- **O subdomínio cc.ggm** : o SP é o "Portátil 2" na porta 2006, o SS1 é a máquina "Cigarra" na porta 2007 e o SS2 é o servidor "Foca" na porta 2008.
- **O subdomínio taremi.fcp** : o SP é a máquina "Grilo" na porta 2009, o SS1 é a máquina "Portátil3" na porta 2010 e o SS2 é o "Servidor3" na porta 2011.
- **Um servidor de resolução** : incluído no subdomínio "taremi.fcp", sendo representado pela máquina "Servidor3". Este tem como domínio por defeito o domínio "taremi.fcp".
- **Um host** : representado pelo "Portátil1" e que tem associado o único servidor de resolução que foi definido.
- **Um domínio de topo .reverse**: representado pela máquina "Golfinho" na porta 2012.
- **Um subdomínio .in-addr**: representado pela máquina "Cigarra" na porta 2013.
- **Os subdomínios X.in-addr.reverse**: a máquina e a porta de atendimento encontram-se especificados na tabela seguinte:

Subdomínio	Máquina	Porta de atendimento
10.in-addr.reverse	Portátil1	2014
1.10.in-addr.reverse	Servidor1	2015
2.10.in-addr.reverse	Servidor2	2016
3.10.in-addr.reverse	Servidor3	2017
4.10.in-addr.reverse	Orca	2018
1.1.10.in-addr.reverse	Golfinho	2019
2.2.10.in-addr.reverse	Foca	2020
3.3.10.in-addr.reverse	Grilo	2021
4.4.10.in-addr.reverse	Vespa	2022

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

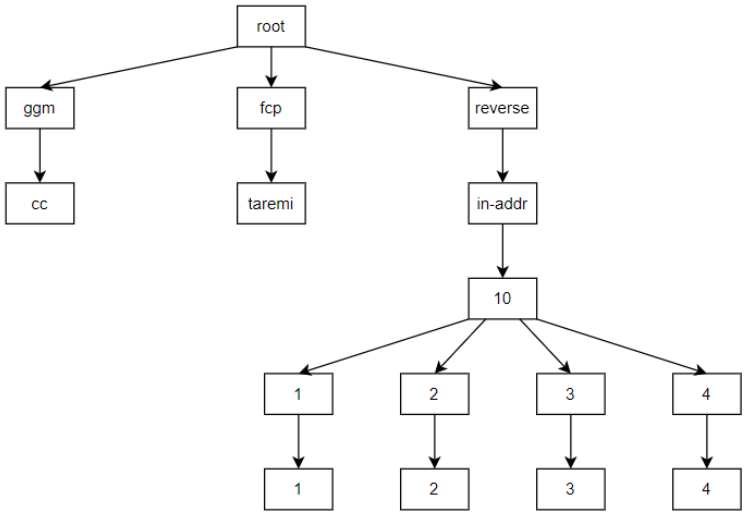

Figura 7: Hierarquia de domínios

Foram criados também os vários ficheiros de configuração (para servidores primários, secundários, de topo, de resolução), os ficheiros de base de dados dos SP, bem como o ficheiro com a lista dos servidores de topo. Estes encontram-se em anexo.

Nesta primeira fase, apenas foram configurados no *core* o servidor primário e um servidor secundário do domínio ".ggm" de modo a testar as funcionalidades básicas de envio/receção de *queries* e a operação de transferência de zona. Nesta fase, cada servidor possui sempre uma porta de atendimento específica. No entanto, na próxima fase, tencionamos ter apenas um servidor para cada máquina que representa diferentes papéis, podendo ser SP para vários domínios, por exemplo. Nesse caso, os diferentes SPs poderão receber *queries* na mesma porta.

7 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 relativo à primeira fase e o planeamento do ambiente de teste. Uma representação do trabalho realizado encontra-se na tabela a seguir.

Tarefa	Gabriela	Guilherme	Miguel
A.1 Arquitetura do Sistema	33%	33%	33%
A.2 Modelo de Informação	33%	33%	33%
A.3 Modelo de Comunicação	33%	33%	33%
A.4 Planeamento do Ambiente de Teste	30%	30%	40%
A.5 Protótipo SP e SS	35%	35%	30%
A.6 Protótipo Cliente	35%	35%	30%
A.7 Relatório	30%	30%	40%

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).

8 Próxima fase do trabalho

Para a próxima etapa do nosso trabalho, estipulamos os seguintes objetivos:

- Permitir que um servidor aceite vários componentes de domínios diferentes;
- Implementar o mecanismo de serialização de dados apresentado;
- Completar o modo iterativo de resposta, permitindo que caso não haja resposta, o processo continue em servidores de outros domínios;
- Terminar a configuração dos restantes domínios na topologia;
- Acrescentar ao mecanismo de transferência de zona os tempos de *SOARETRY* e *SOAEXPIRE*;
- Implementar o modo recursivo de resposta a *queries* (extra).

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 tem sido um bom complemento para aquilo que aprendemos nas aulas teóricas.

Nesta primeira fase, 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*.

O balanço desta fase é, por isso, bastante positivo pelo que consideramos estar aptos para a segunda fase deste projeto, onde iremos implementar funcionalidades mais avançadas do sistema DNS.

10 Lista de Siglas e Acrónimos

DNS - *Domain Name System*

SP - Servidor Primário

SS - Servidor Secundário

SR - Servidor de Resolução

ST - Servidor de topo

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 Ficheiros de configuração

Servidor de topo

```
# Configuration file for root server

. DB dns/files/config_files/root.db
. LG dns/files/log_files/root.log
all LG dns/files/log_files/all.log
```

Figura 8: Ficheiro de configuração do servidor "root"

Servidor primário: ggm

```
# Configuration file for primary server for ggm domain

ggm DB dns/files/db_files/ggm.db
ggm SS 10.3.3.1:2001
ggm SS 10.4.1.1:2002
ggm LG dns/files/log_files/ggm.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 9: Ficheiro de configuração do servidor primário do domínio ggm

Servidor secundário: ggm

```
# Configuration file for secondary server 1 for ggm domain

ggm SP 10.2.2.1:2000
ggm LG dns/files/log_files/ggm.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 10: Ficheiro de configuração do servidor secundário do domínio ggm

Servidor primário: fcp

```
# Configuration file for primary server for fcp domain

fcp DB dns/files/db_files/fcp.db
fcp SS 10.3.3.2:2004
fcp SS 10.4.4.3:2005
fcp LG dns/files/log_files/fcp.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 11: Ficheiro de configuração do servidor primário do domínio fcp

Servidor secundário: fcp

```
# Configuration file for secondary server 1 for fcp domain

fcp SP 10.2.2.2:2003
fcp LG dns/files/log_files/fcp.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 12: Ficheiro de configuração do servidor secundário do domínio fcp

Servidor primário: cc.ggm

```
# Configuration file for primary server for cc.ggm domain

cc.ggm SS 10.4.4.2:2007
cc.ggm SS 10.3.3.3:2008
cc.ggm LG dns/files/log_files/cc_ggm.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 13: Ficheiro de configuração do servidor primário do domínio cc.ggm

Servidor secundário: cc.ggm

```
# Configuration file for secondary server 1 for cc.ggm domain

cc.ggm SP 10.1.1.2:2006
cc.ggm LG dns/files/log_files/cc_ggm.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 14: Ficheiro de configuração do servidor secundário do domínio cc.ggm

Servidor primário: taremi.fcp

```
# Configuration file for primary server for taremi.fcp domain

taremi.fcp SS 10.1.1.3:2010
taremi.fcp SS 10.2.2.3:2011
taremi.fcp LG dns/files/log_files/taremi_fcp.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 15: Ficheiro de configuração do servidor primário do domínio taremi.fcp

Servidor secundário: taremi.fcp

```
# Configuration file for secondary server 1 for taremi.fcp domain

taremi.fcp SP 10.4.4.1:2009
taremi.fcp LG dns/files/log_files/taremi_fcp.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 16: Ficheiro de configuração do servidor secundário do domínio taremi.fcp

Servidor de resolução: taremi.fcp

```
# Configuration file for name resolver for taremi.fcp domain

taremi.fcp DD 10.4.4.1:2009
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 17: Ficheiro de configuração do servidor de resolução para o domínio taremi.fcp

Servidor primário: reverse

```
# Configuration file for primary server for .reverse domain

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

Figura 18: Ficheiro de configuração do servidor primário do domínio reverse

Servidor primário: in-addr.reverse

```
# Configuration file for primary server for .in-addr.reverse domain

in-addr.reverse LG dns/files/log_files/in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 19: Ficheiro de configuração do servidor primário do domínio in-addr.reverse

Servidor primário: 10.in-addr.reverse

```
# Configuration file for primary server for 10.in-addr.reverse domain

10.in-addr.reverse. LG dns/files/log_files/10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 20: Ficheiro de configuração do servidor primário do domínio 10.in-addr.reverse

Servidor primário: 1.10.in-addr.reverse

```
# Configuration file for primary server for 1.10.in-addr.reverse domain

1.10.in-addr.reverse. LG dns/files/log_files/1_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 21: Ficheiro de configuração do servidor primário do domínio 1.10.in-addr.reverse

Servidor primário: 2.10.in-addr.reverse

```
# Configuration file for 2.10 domain

2.10.in-addr.reverse. LG dns/files/log_files/2_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 22: Ficheiro de configuração do servidor primário do domínio 2.10.in-addr.reverse

Servidor primário: 3.10.in-addr.reverse

```
# Configuration file for 3.10 domain

3.10.in-addr.reverse. LG dns/files/log_files/3_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 23: Ficheiro de configuração do servidor primário do domínio 3.10.in-addr.reverse

Servidor primário: 4.10.in-addr.reverse

```
# Configuration file for 4.10 domain

4.10.in-addr.reverse. LG dns/files/log_files/4_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 24: Ficheiro de configuração do servidor primário do domínio 4.10.in-addr.reverse

Servidor primário: 1.1.10.in-addr.reverse

```
# Configuration file for 1.1.10 domain

1.1.10.in-addr.reverse LG dns/files/log_files/1_1_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 25: Ficheiro de configuração do servidor primário do domínio 1.1.10.in-addr.reverse

Servidor primário: 2.2.10.in-addr.reverse

```
# Configuration file for 2.1.10 domain

2.1.10.in-addr.reverse LG dns/files/log_files/2_1_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 26: Ficheiro de configuração do servidor primário do domínio 2.2.10.in-addr.reverse

Servidor primário: 3.3.10.in-addr.reverse

```
# Configuration file for 3.3.10 domain

3.3.10.in-addr.reverse LG dns/files/log_files/3_3_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 27: Ficheiro de configuração do servidor primário do domínio 3.3.10.in-addr.reverse

Servidor primário: 4.4.10.in-addr.reverse

```
# Configuration file for 4.4.10 domain

4.4.10.in-addr.reverse LG dns/files/log_files/4_4_10_in-addr_reverse.log
all LG dns/files/log_files/all.log
root ST dns/files/rootservers.db
```

Figura 28: Ficheiro de configuração do servidor primário do domínio 4.4.10.in-addr.reverse

12.2 Ficheiros de base de dados

Root server

```
# DNS database file for root server

@ DEFAULT root.
TTL DEFAULT 86400

# Config data
@ SOADMIN mike\.ggm. TTL
@ SOASERIAL 1 TTL


# Name servers
root. NS s1. TTL
root. NS s2. TTL
ggm. NS ns1.ggm. TTL
fcp. NS ns1.fcp. TTL
reverse. NS ns1.reverse. TTL

# Addresses

sp.ggm. A 10.2.2.1:2000 TTL
ss1.ggm. A 10.3.3.1:2001 TTL
ss2.ggm. A 10.4.4.1:2002 TTL
sp.fcp. A 10.2.2.2:2003 TTL
ss1.fcp. A 10.3.3.2:2004 TTL
ss2.fcp. A 10.4.4.3:2005 TTL
sp.reverse. A 10.3.3.2:2012 TTL

s1. A 10.2.2.1:1998 TTL
s2. A 10.3.3.1:1999 TTL
```

Figura 29: Ficheiro de base de dados do servidor “root“

Domínio ggm

```
# DNS database file domain .ggm

@ DEFAULT ggm.
TTL DEFAULT 86400

@ SOASP s1.ggm TTL
@ SOAADMIN admin\.ggm. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 60 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS sp.ggm. TTL
@ NS ss1.ggm. TTL 1
@ NS ss2.ggm. TTL 2

# Primary server of cc.ggm subdomain
cc.@ NS sp.cc.@ TTL

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

sp A 10.2.2.1:2000 TTL
ss1 A 10.3.3.1:2001 TTL
ss2 A 10.4.4.1:2002 TTL
sp.cc A 10.1.1.2:2006 TTL
mike A 10.1.1.3 TTL
gui A 10.4.4.1 TTL
gabs A 10.3.3.2 TTL
ms1 A 10.3.3.1 TTL
ms2 A 10.3.3.3 TTL

miguel CNAME mike TTL
guilherme CNAME gui TTL
gabriela CNAME gabs TTL
```

Figura 30: Ficheiro de base de dados do domínio ggm

Domínio fcp

```
# DNS database file for domain .fcp

@ DEFAULT fcp.
TTL DEFAULT 86400

@ SOASP ns1.fcp. TTL
@ SOADMIN admin.\fcp. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.fcp. TTL
@ NS ns2.fcp. TTL
@ NS ns3.fcp. TTL

taremi.@ NS ns1.taremi.fcp.

# MAIL SERVERS
@ MX ms1.fcp. TTL

# Addresses
ns1 A 10.2.2.2:2003 TTL
ns2 A 10.3.3.2:2004 TTL
ns3 A 10.4.4.3:2005 TTL
ns1.taremi A 10.4.4.1:2009 TTL
ms1 A 10.2.2.1 TTL
pdc A 10.4.4.1 TTL
avb A 10.4.4.2 TTL

sp CNAME ns1 TTL
ss1 CNAME ns2 TTL
ss2 CNAME ns3 TTL
mail1 CNAME ms1 TTL
```

Figura 31: Ficheiro de base de dados do domínio fcp

Domínio cc.ggm

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

@ DEFAULT cc.ggm.
TTL DEFAULT 86400

@ SOASP ns1.cc.ggm. TTL
@ SOADMIN admin\.cc.ggm. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.cc.ggm. TTL
@ NS ns2.cc.ggm. TTL
@ NS ns3.cc.ggm. TTL

# MAIL SERVERS
@ MX ms1.@ TTL

# Addresses
ns1 A 10.1.1.2:2006 TTL
ns2 A 10.4.4.2:2007 TTL
ns3 A 10.3.3.3:2008 TTL
ms1 A 10.4.4.2 TTL
tone A 10.1.1.1 TTL
tunes A 10.1.1.3 TTL

sp CNAME ns1 TTL
ss1 CNAME ns2 TTL
ss2 CNAME ns3 TTL
```

Figura 32: Ficheiro de base de dados do domínio cc.ggm

Domínio taremi.fcp

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

@ DEFAULT .taremi.fcp
TTL DEFAULT 86400

@ SOASP ns1.taremi.fcp. TTL
@ SOADMIN admin\.taremin.fcp. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.taremi.fcp. TTL
@ NS ns2.taremi.fcp. TTL
@ NS ns3.taremi.fcp. TTL

# MAIL SERVERS
@ MX ms1.@ TTL

# Addresses
ns1 A 10.4.4.1:2009 TTL
ns2 A 10.1.1.3:2010 TTL
ns3 A 10.2.2.3:2011 TTL
taremi A 10.3.3.3 TTL
pepe A 10.4.4.3 TTL
zaidu A 10.3.3.2 TTL
ms1 A 10.3.3.1 TTL
sr 10.2.2.3 TTL

sp CNAME ns1 TTL
ss1 CNAME ns2 TTL
ss2 CNAME ns3 TTL
```

Figura 33: Ficheiro de base de dados do domínio taremi.fcp

Domínio reverse

```
# DNS database file for reverse domain.

@ DEFAULT .reverse.
TTL DEFAULT 86400

# Config data

@ SOASP ns1.reverse. TTL
@ SOADMIN gabs\.ggm. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers

@ NS ns1.reverse. TTL
in-addr.reverse. NS ns1.in-addr.reverse. TTL

# Addresses

ns1 A 10.3.3.2:2012 TTL
ns1.in-addr A 10.4.4.2:2013 TTL
```

Figura 34: Ficheiro de base de dados do domínio reverse

Domínio in-addr.reverse

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

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

# Config data

@ SOASP ns1.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm. TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.in-addr.reverse. TTL
10.in-addr.reverse. NS ns1.10.in-addr.reverse. TTL

# Addresses

ns1 A 10.4.4.2:2013
ns1.10.in-addr A 10.1.1.1:2014 TTL
```

Figura 35: 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 ns1.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers

@ NS ns1.10.in-addr.reverse. TTL
1.@ NS ns1.1.1.10.in-addr.reverse. TTL
2.@ NS ns1.2.10.in-addr.reverse. TTL
3.@ NS ns1.3.10.in-addr.reverse. TTL
4.@ NS ns1.4.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.1.1.1:2014 TTL
ns1.1 A 10.2.2.1:2015 TTL
ns1.2 A 10.2.2.2:2016 TTL
ns1.3 A 10.2.2.3:2017 TTL
ns1.4 A 10.3.3.1:2018 TTL
```

Figura 36: Ficheiro de base de dados do domínio 10.in-addr.reverse

Domínio 1.10.in-addr.reverse

```
# DNS database file for domain 1.10.in-addr.reverse

@ DEFAULT 1.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.1.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers

@ NS ns1.1.10.in-addr.reverse. TTL
1.@ NS ns1.1.1.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.2.2.1:2015 TTL
ns1.1 A 10.3.3.2:2019 TTL
```

Figura 37: Ficheiro de base de dados do domínio 1.10.in-addr.reverse

Domínio 2.10.in-addr.reverse

```
# DNS database file for domain 2.10.in-addr.reverse

@ DEFAULT 2.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.2.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.2.10.in-addr.reverse. TTL
2.@ NS ns1.2.2.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.2.2.2:2016 TTL
ns1.2 A 10.3.3.3:2020 TTL
```

Figura 38: Ficheiro de base de dados do domínio 2.10.in-addr.reverse

Domínio 3.10.in-addr.reverse

```
# DNS database file for domain 3.10.in-addr.reverse

@ DEFAULT 3.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.3.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers

@ NS ns1.3.10.in-addr.reverse. TTL
3.@ NS ns1.3.3.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.2.2.3:2017 TTL
ns1.3 A 10.4.4.1:2021 TTL
```

Figura 39: Ficheiro de base de dados do domínio 3.10.in-addr.reverse

Domínio 4.10.in-addr.reverse

```
# DNS database file for domain 4.10.in-addr.reverse

@ DEFAULT 4.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.4.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers

@ NS ns1.4.10.in-addr.reverse. TTL
4.@ NS ns1.4.1.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.3.3.1:2018 TTL
ns1.4 A 10.4.4.3:2022 TTL
```

Figura 40: Ficheiro de base de dados do domínio 4.10.in-addr.reverse

Domínio 1.1.10.in-addr.reverse

```
# DNS database file for domain 1.1.10.in-addr.reverse

@ DEFAULT 1.1.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.1.1.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.1.1.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.3.3.2 TTL
10.1.1.1 PTR [sp.10.in-addr.reverse , tone.cc.ggm.] TTL
10.1.1.3 PTR [sp.cc.ggm , tunes.cc.ggm.] TTL
10.1.1.2 PTR [ss1.taremi.fcp , mike.ggm] TTL
```

Figura 41: Ficheiro de base de dados do domínio 1.1.10.in-addr.reverse

Domínio 2.2.10.in-addr.reverse

```
# DNS database file for domain 2.2.10.in-addr.reverse

@ DEFAULT 2.2.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.2.2.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.2.2.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.3.3.3 TTL
10.2.2.1 PTR [s1. , sp.ggm. , sp.1.10.in-addr.reverse. , ms1.fcp.] TTL
10.2.2.2 PTR [sp.fcp. , sp.2.10.in-addr.reverse.] TTL
10.2.2.3 PTR [ss2.taremi.fcp. , sr.taremi.fcp. , sp.3.10.in-addr.reverse.] TTL
```

Figura 42: Ficheiro de base de dados do domínio 2.2.10.in-addr.reverse

Domínio 3.3.10.in-addr.reverse

```
# DNS database file for domain 3.3.10.in-addr.reverse

@ DEFAULT 3.3.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.3.3.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.3.3.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.4.4.1 TTL
10.3.3.1 PTR [s1. , ss1.ggm. , sp.4.10.in-addr.reverse , ms1.ggm.] TTL
10.3.3.2 PTR [ss1.fcp. , sp.reverse. , sp.1.1.19.in-addr.reverse. , gabs.ggm] TTL
10.3.3.3 PTR [ss2.cc.ggm. , sp.2.2.10.in-addr.reverse , ms2.ggm] TTL
```

Figura 43: Ficheiro de base de dados do domínio 3.3.10.in-addr.reverse

Domínio 4.4.10.in-addr.reverse

```
# DNS database file for domain 4.4.10.in-addr.reverse

@ DEFAULT 4.4.10.in-addr.reverse.
TTL DEFAULT 86400

@ SOASP ns1.4.4.10.in-addr.reverse. TTL
@ SOADMIN gabs\.ggm TTL
@ SOASERIAL 1 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600
@ SOAEXPIRE 604800 TTL

# Name servers
@ NS ns1.4.4.10.in-addr.reverse. TTL

# Addresses
ns1 A 10.4.4.3 TTL
10.4.4.1 PTR [ss2.ggm. , sp.taremi.fcp. , sp.3.3.10.in-addr.reverse. , pdc.fcp. , gui.ggm.] TTL
10.4.4.2 PTR [ss2.fcp. , sp.4.4.10.in-addr.reverse, ms1.cc.ggm. , avb.fcp.] TTL
10.4.4.3 PTR [ss1.cc.ggm. , sp.in-addr.reverse.] TTL
```

Figura 44: Ficheiro de base de dados do domínio 4.4.10.in-addr.reverse