

Comunicações por Computador

TP2.b Grupo 7.04



Lídia Sousa
a93205



Henrique Alvelos
a93316



Gonçalo Moreira
a73591

Conteúdo

1	Introdução	2
1.1	Contextualização	2
1.2	Objetivos	2
1.3	Estratégias genéricas	3
1.3.1	Primeira fase	3
1.3.2	Segunda fase	3
1.4	Glossário	3
2	Arquitetura do sistema	4
2.1	Componentes software	4
2.1.1	Servidor	4
2.1.2	Cliente	5
2.2	Módulos	5
2.3	Interação entre componentes	7
2.3.1	Primeiro caso de estudo	7
2.3.2	Segundo caso de estudo	8
3	Modelo de Comunicação	10
3.1	Modelo de comunicação	10
3.1.1	Representação lógica de mensagem DNS	10
3.1.2	Transferência de zona	11
3.2	Modelo de informativo	13
3.2.1	Situações de erro	13
4	Implementação	16
4.1	Funcionalidades dos componentes	16
4.1.1	Cliente	16
4.1.2	Servidor	17
4.2	Outras funcionalidades	18
4.2.1	Modo debug	18
4.2.2	Parsers	18
5	Análise de Testes	19
5.1	Ambiente de teste	19
5.2	Testes	20
5.2.1	Query efetuada acerca de ST	20
5.2.2	Query efetuada a SR do domínio da query	20
5.2.3	Query efetuada a SP do domínio da query	21
5.2.4	Query efetuada a SS do domínio da query	21
5.2.5	Query efetuada a um servidor de um domínio acerca de um subdomínio	22
6	Participação do grupo	23
7	Conclusão	24

Capítulo 1

Introdução

Keywords: TCP · UDP · DNS · CORE

Neste relatório é apresentada a explicação detalhada da implementação de um Sistema DNS bem como o conjunto de decisões tomadas em torno do mesmo como por exemplo, a especificação do PDU, os tipos de comunicação, entre outros. As alterações realizadas no contexto da segunda fase do trabalho vão ser enunciadas ao longo deste trabalho.

De forma a facilitar a leitura deste relatório e identificar as alterações de forma mais simples, o conteúdo que coincidir com o relatório da primeira fase vão estar a cinzento e as adições vão estar a preto.

1.1 Contextualização

O DNS — do inglês *Domain Name System* - consiste num registo de nomes de domínios e respetivos IP's associados. Assim, um serviço DNS é globalmente distribuído e converte nomes legíveis e intuitivos para os humanos, como `www.comunicacoescomputador.com`, em endereços IP como `192.0.0.1`, usados pelos computadores. Para que isto aconteça, há um mapeamento de nomes e endereços através de servidores DNS.

Tipos de DNS

DNS autoritativo - Os servidores autoritativos são os servidores que têm autoridade para fornecer informações de um domínio. No caso, são os servidores que são configurados quando se atribuem os DNSs a um domínio.

DNS recursivo e iterativo - O DNS recursivo consiste num servidor DNS comunicar com vários outros para encontrar o endereço IP e retornar o mesmo ao cliente. Por outro lado, o DNS iterativo consiste numa consulta em que o cliente comunica diretamente com cada servidor DNS envolvido.

DNS reverso - Se o *resolver* – o servidor de resolução de nomes de domínio – é responsável por determinar o IP correspondente a cada domínio, o reverso faz o inverso, ou seja, determina o domínio ou nameserver (domínio do servidor) correspondente a um IP.

1.2 Objetivos

Com a realização deste trabalho pretende-se a implementação de um sistema DNS que mantenha *cache* consultas feitas e resoluções DNS, ou seja, dos endereços IP correspondentes a cada domínio utilizado. Além disso pretende-se que a implementação siga um conjunto de especificações definidas no enunciado e ao longo do relatório.

1.3 Estratégias genéricas

1.3.1 Primeira fase

Na primeira fase deste trabalho implementamos os componentes básicos que consistem numa comunicação servidor-cliente em UDP com suporte a uma query e modo *debug*. O sistema DNS vai funcionar de forma iterativa. Inicialmente vamos também definir também um ambiente de teste bem como os respetivos ficheiros de configuração e base de dados.

1.3.2 Segunda fase

Na segunda fase do trabalho implementamos as funcionalidades restantes, como a cache, a resposta a *queries* em modo recursivo bem como a correção de alguns problemas identificados na solução entregue na primeira fase, como as *queries hard-coded*, problemas com portas alocadas, modo debug desenvolvido de forma incorreta, a escrita dos logs nos devidos ficheiros, entre outros.

De forma resumida as estratégias seguidas para corrigir os erros da primeira fase consistem em:

- Remover as porções de código *hard-coded*;
- Alterar o modo debug do formato das mensagens de log para o exemplificado do enunciado;
- Passar a escrita de logs para um ficheiro ao invés de para o terminal;
- Implementar o modo iterativo e o servidor *resolver*;

1.4 Glossário

DNS - *Domain Name System*

ST - Servidor de Topo

SP - Servidor Primário

SS - Servidor Secundário

SR - Servidor *Resolver*

IP - *Internet Protocol*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

PDU - *Protocol Data Unit*

Capítulo 2

Arquitetura do sistema

O serviço *DNS* a implementar consiste, essencialmente, num sistema servidor-cliente.

2.1 Componentes software

2.1.1 Servidor

Um dos componentes deste sistema são os **servidores**. Há três tipos de servidores: **Servidor Primário**, **Servidor Secundário** e **Servidor de Resolução**. O servidor recebe um ficheiro de configuração, que determina todas as informações e, consequentemente, o seu tipo e retorna um ficheiro de *log*, que será devidamente explicitado ao longo do relatório.

Servidor Primário:

O Servidor Primário de um domínio DNS é um **servidor de DNS reverso**. Uma consulta de DNS reversa é uma consulta de DNS para o nome de domínio associado a um determinado endereço de IP. O SP responde e efetua *queries* DNS e é autoridade que gere um determinado domínio DNS, tendo acesso direto à base de dados do mesmo. Ou seja, os servidores primários são **autoritativos**, são os servidores que têm autoridade para fornecer informações de um domínio.

Para além dos dados que dizem respeito a um domínio DNS, cada SP vai ter acesso a:

- Domínios para os quais é SP;
- Portas de atendimento;
- Identificação dos ficheiros das bases de dados - o SP tem um ficheiro de base de dados por cada domínio gerido;
- Identificação do ficheiro de *log*;
- Informação de segurança para acesso às bases de dados;
- Identificação dos Servidores Secundários respetivos e dos Servidores Primários dos subdomínios;
- Endereços dos servidores de topo;

Servidor Secundário:

O SS responde e efetua *queries* DNS. Além disso deve ter uma réplica da base de dados original do Servidor Primário autoritativo. A atualização da réplica desta base de dados vai ser feita através de **transferência de zona**, que será abordado noutra fase deste relatório.

Para além da base de dados, o SS vai ter acesso a:

- Domínios para os quais é SS;

- Portas de atendimento;
- Identificação do SP do domínio para o qual é SS;
- Identificação do ficheiro de *log*;
- Informação de segurança para acesso aos SP;
- Endereços dos servidores de topo;

Servidor de Resolução:

O Servidor de Resolução, ou *resolver*, é o que responde e efetua *queries* DNS sobre qualquer domínio, no entanto não tem autoridade sobre nenhum.

Um SR tem de ter acesso a informação de configuração específica:

- Domínios por defeito e lista dos servidores DNS que deve contactar;
- Portas de atendimento;
- Identificação do ficheiro de *log*;
- Endereços dos servidores de topo;

2.1.2 Cliente

O outro componente chave neste sistema é o **cliente**. O cliente DNS funciona através de um processo na linha de comando que questiona a informação de um determinado domínio, efetuando as *queries* DNS a um SR. Estas *queries* são portanto efetuadas através da linha de comandos e, consequentemente, o *output* é através do mesmo meio.

2.2 Módulos

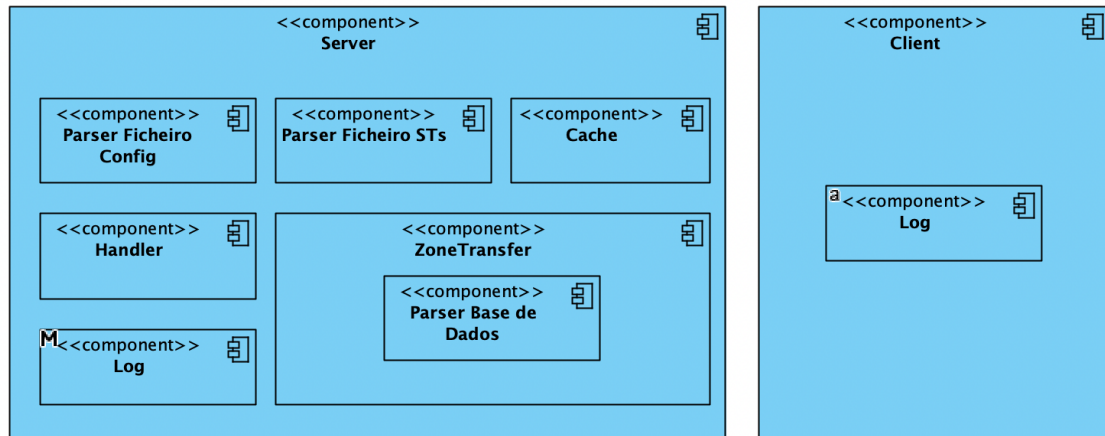


Figura 2.1: Diagrama de módulos

Na implementação final do projeto, a divisão e estrutura do código está desenvolvida de forma a estruturar o armazenamento de dados de forma eficiente bem como os serviços de forma independente. Ou seja, a *Zone Transfer*, por exemplo, efetuada por Servidores quer primários quer secundários está definida numa classe independente, que será depois utilizada pela classe do servidor. O funcionamento de cada um destes módulos será explicada em detalhe no capítulo de implementação. De forma a ser mais intuitivo, apresentamos na figura 2.2 um modelo que permite visualizar a estrutura e divisão do código:

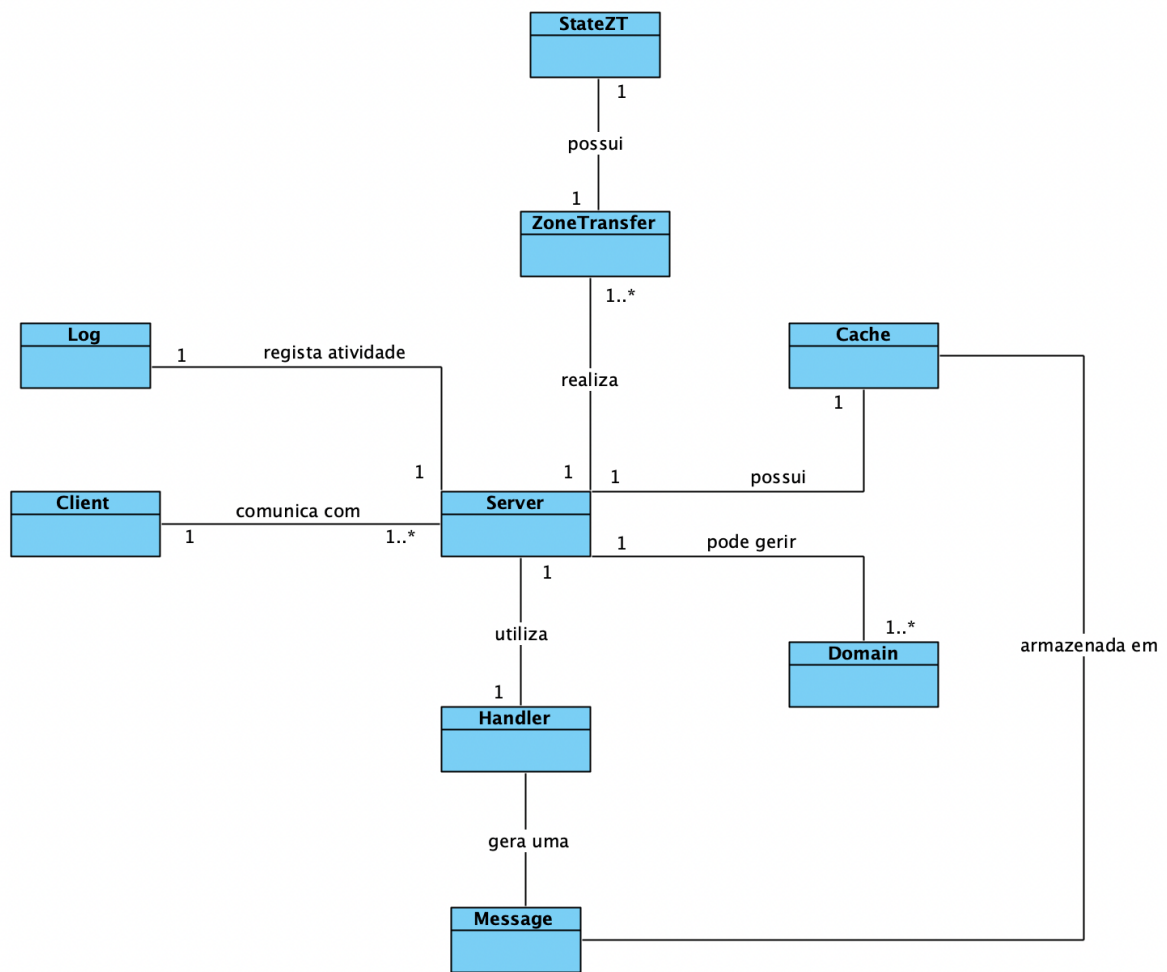


Figura 2.2: Modelo domínio

Relativamente a cada classe da figura 2.2, faremos agora uma explicação mais detalhada daquilo em que consiste a interligação entre cada uma:

- **Server** - O servidor recebe pedidos dos clientes e recorre a *threads* de forma a conseguir atender diversos pedidos em simultâneo. As *threads* usam o Handler como forma de tratar um pedido e saber de que forma devem responder.
- **Handler** - O Handler recebe uma query, e dado o funcionamento do programa percebe se o servidor de que se trata deve enviar a query aos ST ou responder à query.
- **ZoneTransfer** - A ZoneTransfer trata o processo de sincronização de bases de dados entre servidores usando o protocolo TCP. A sincronização acontece em intervalos de valor SOA-REFRESH ou, caso esta falhe, em intervalo de tempo SOARETRY.
- **StateZT** - O StateZT é o estado que a ZoneTransfer usa para verificar as linhas que envia comparando com as linhas que tem que enviar.
- **Cache** - A cache implementa um sistema que todos os servidores vão ter que, de forma resumida, tem uma capacidade. Os elementos têm prioridade e são removidos conforme a prioridade. No caso da prioridade ser a mesma, é usado o método LFU (Least Frequently Used).
- **Log** - O Log trata dos métodos responsáveis por escrever as mensagens nos ficheiros devidos.

- **Domain** - A classe Domain guarda as informações relativas a cada domínio como o seu nome, ficheiro de base de dados e IP de servidor primário.
- **Message** - A classe Message consiste na estrutura que vai ser usada para os clientes e servidores trocarem informações (queries e respetivas respostas).
- **Client** - O cliente é o componente de software que introduz a query que pretende efetuar. O nosso cliente tem um resolver agnóstico de domínios que, no fundo, até receber a resposta à query efetuada ou erro, vai controlar a query.

O funcionamento deste sistema DNS vai basear-se então, de forma resumida, num cliente enviar uma *query* acerca de um domínio NAME para um servidor. Este servidor procede à descodificação da *query* recebida e caso esta esteja correta tenta encontrar informação ou na *cache* ou na sua base de dados (caso seja um servidor autoritativo para o domínio NAME).

Para o caso em que a resposta à *query* não se encontra na *cache*, reenvia a *query* para um servidor de topo (ST) que seja o servidor do domínio de topo incluído no NAME. O ST procede da mesma forma, verificando se tem a informação em *cache* e caso isto não se verifique reenvia a *query*. Caso o SR tenha a informação em *cache* dispõe a mesma ao cliente automaticamente.

- **Query efetuada a um SR:** Se o cliente enviar uma query ao resolver do domínio do qual quer informações, caso este não tenha na sua cache a resposta, responde com a entrada DD (caso exista). Caso contrário, procede como os restantes, respondendo com a lista de STs.
- **Query efetuada a um SP:** Caso a resposta não exista na cache, e caso o SP não tenha informações de quais os servidores a ser interrogados para responder à query, deve enviar a lista de STs.

2.3 Interação entre componentes

Vamos apresentar dois diagramas de sequência com dois casos de uso diferentes. O primeiro caso de uso vai ser uma query efetuada diretamente a um servidor com informação acerca do domínio de forma a ser mais fácil entender os principais sub-componentes do servidor e cliente e interação entre eles.

O outro caso de uso vai ser para representar o modo iterativo e a forma como os componentes interagem entre si para enviar uma resposta ao cliente.

2.3.1 Primeiro caso de estudo

Este caso de estudo refere-se a uma query efetuada a um domínio (LadoA-G704.) sobre esse mesmo domínio. O servidor vai procurar a resposta na cache, caso esta não exista vai verificar se o domínio pertence aos seus e vai construir a resposta enviando a mesma ao cliente.

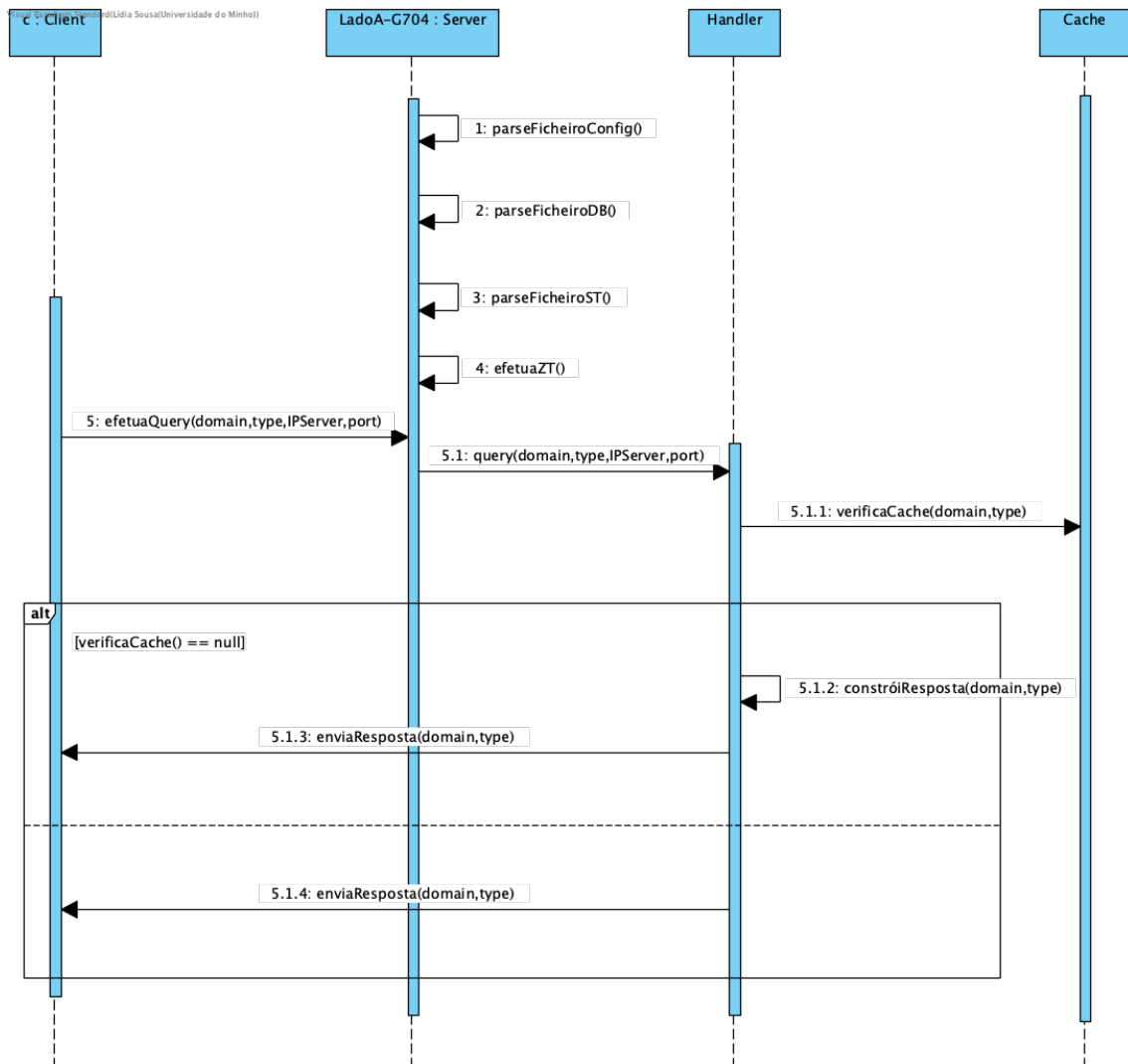


Figura 2.3: Exemplo de interação entre componentes cliente e servidor

2.3.2 Segundo caso de estudo

Neste caso, vamos representar uma query efetuada a um domínio (LadoA-G704.) relativamente a um subdomínio (um-LadoA-G704.). Logo, inicialmente o servidor verifica a cache. Se a resposta não estiver na cache deve construir uma resposta com os valores que deve enviar (neste caso informações sobre os servidores do subdomínio) e enviar a mesma ao cliente.

O cliente, que tem um resolver agnóstico, verifica se esta resposta é final através da *response code* e respetivo significado. Se a resposta for final esta é apresentada ao cliente, senão é reencaminhada para os valores indicados pelo domínio.

Neste caso, a query é enviada então a um servidor do domínio um-LadoA-G704, que novamente repete o processo de procurar na cache a resposta da query e caso não esteja, construir a mesma para enviar a resposta final ao cliente. De seguida, mostramos esta explicação no diagrama de sequência 2.4 que descreve uma query efetuada a um domínio (LadoA) acerca do seu subdomínio (um-LadoA):

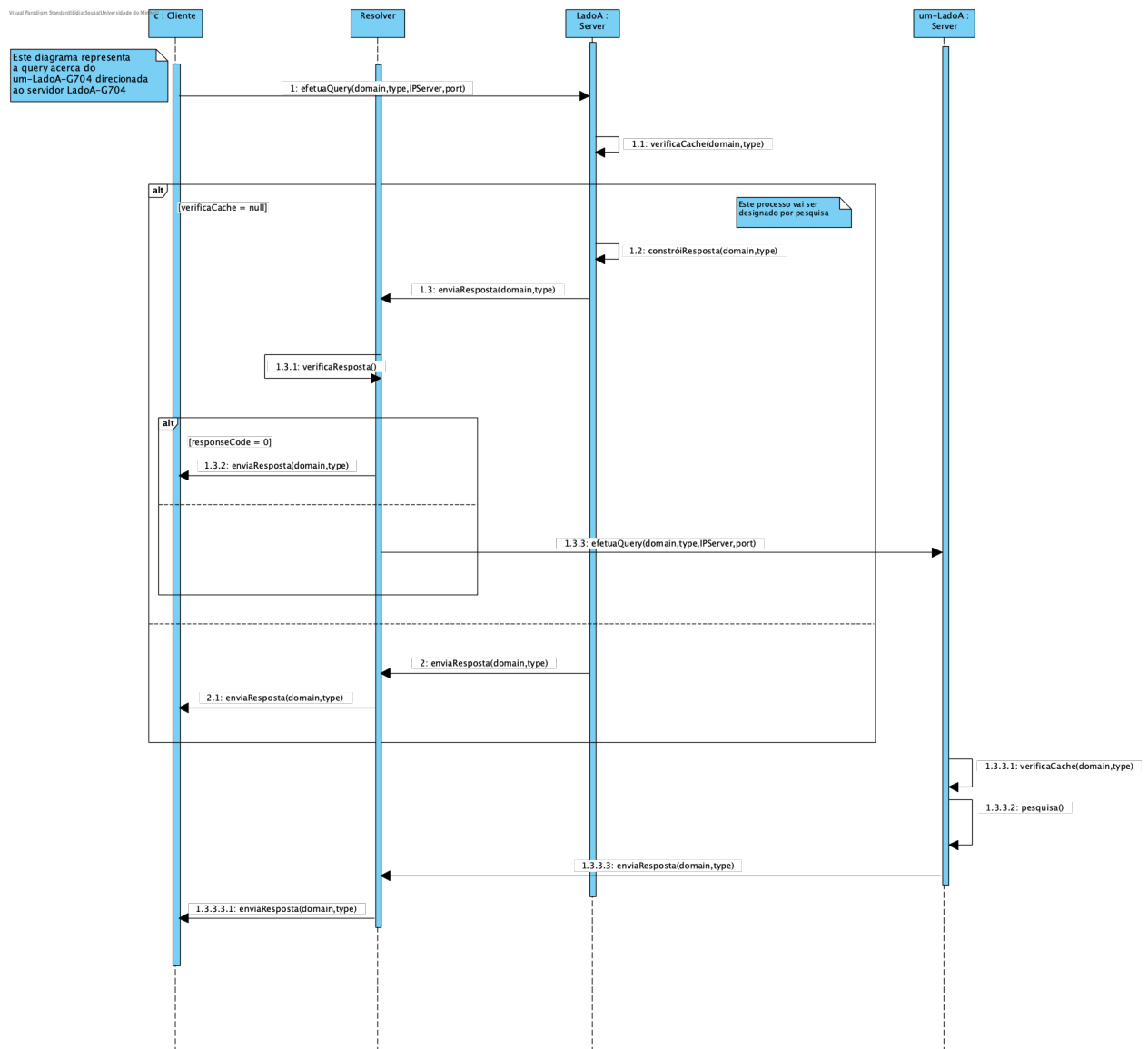


Figura 2.4: Exemplo de interação entre componentes cliente e servidor

Capítulo 3

Modelo de Comunicação

3.1 Modelo de comunicação

Os tipos de comunicação que existem são entre o cliente e os servidores, que entre si não têm diferenças. Esta comunicação assenta num **protocolo UDP**. Existe também uma comunicação entre servidor primário e servidor secundário no contexto da transferência de zona. Contrariamente à anterior, esta comunicação assenta no **protocolo TCP**.

3.1.1 Representação lógica de mensagem DNS

Numa primeira fase, a representação das mensagens DNS é implementada no contexto de uma *struct* denominada Message com campos do tipo *int*, *String* e *ArrayList*. Os campos da *struct Message* foram definidos conforme o enunciado no sentido em que cada um dos campos da DNS Message é uma variável da *struct*. No entanto, inicialmente, não haverá uma divisão daquilo que se refere ao campo *Header* e ao campo *Data*. A mensagem passa *serialized* num array de *bytes*. Numa segunda fase, a ideia será que o PDU tenha sintaxe de codificação eficiente de forma a uma mensagem ocupar o menor espaço possível. O objetivo seria efetuar uma codificação binária que não use separadores de campos

Header

Message ID - Corresponde a um inteiro de 16-bits (2 bytes).

Flags - Corresponde a 3 bits que representam a ordem de {Query}{Modo}{Autoritativo} das flags Q,R e A. A flag Query indica que a mensagem é uma *query* ou uma resposta a uma *query*. A flag Modo indica se pretendemos que o processo seja recursivo (1) ou iterativo (por default, 0). A flag A indica que a resposta é autoritativa. Por default o valor deve ser 0 e ignorado nas *queries* efetuadas pelo cliente;

Response Code - Representado por 2 bits de forma a reportar os 3 tipos de erros apresentados na mensagem DNS;

Number of values - Máximo de 8 bits para representar o número de entradas relevantes de no máximo 255 relativos a *queries*;

Number of authorities - Máximo de 8 bits;

Number of extra values - Máximo de 8 bits;

Data

O único campo que pretendemos codificar no que toca a Data é no campo **Query Info**, mais precisamente o **Type Of Value**. Isto deve-se ao facto de conseguirmos representar, por exemplo, 'MX' com menos do que 2 bytes caso este seja representado numa string. Portanto, a codificação

deste campo seria com 3 bits porque existem apenas as possibilidades definidas no ficheiro de dados do SP.

3.1.2 Transferência de zona

A transferência de zona é feita utilizando uma conexão TCP, tal como referido anteriormente. O SS efetua *queries* em intervalos de tempo no valor de **SOAREFRESH** para saber se a versão da base de dados que tem é a mais recente e recebe uma mensagem com a versão da base de dados do SP. Caso o SP esteja mais atualizado que o SS inicia-se uma tentativa de transferência de zona em que o SS envia ao SP o nome do domínio para o qual quer uma cópia da base de dados.

Verifica-se se o SS tem autorização para aceder a estes dados e verificando-se o SP informa o SS de quantas linhas (n) pretende enviar com a base de dados atualizada. A atualização da base de dados vai ser feita totalmente, não há comparação do que há em ambos os servidores, é enviado na totalidade o conteúdo. Após obter confirmação de que o SS está pronto para receber as n linhas (a confirmação é feita quando o SS envia ao SP o mesmo n), o SP envia as entradas.

A transferência de zona tem um tempo definido para decorrer e caso este seja ultrapassado (*timeout*) o SS termina a conexão TCP e efetua a *query* após um intervalo de tempo de **SOARETRY**. As mensagens relativas à transferência de zona foram definidas sem recurso à representação lógica da Mensagem de DNS presente no enunciado. Para isto, definimos apenas alguns campos importantes, descritos a seguir:

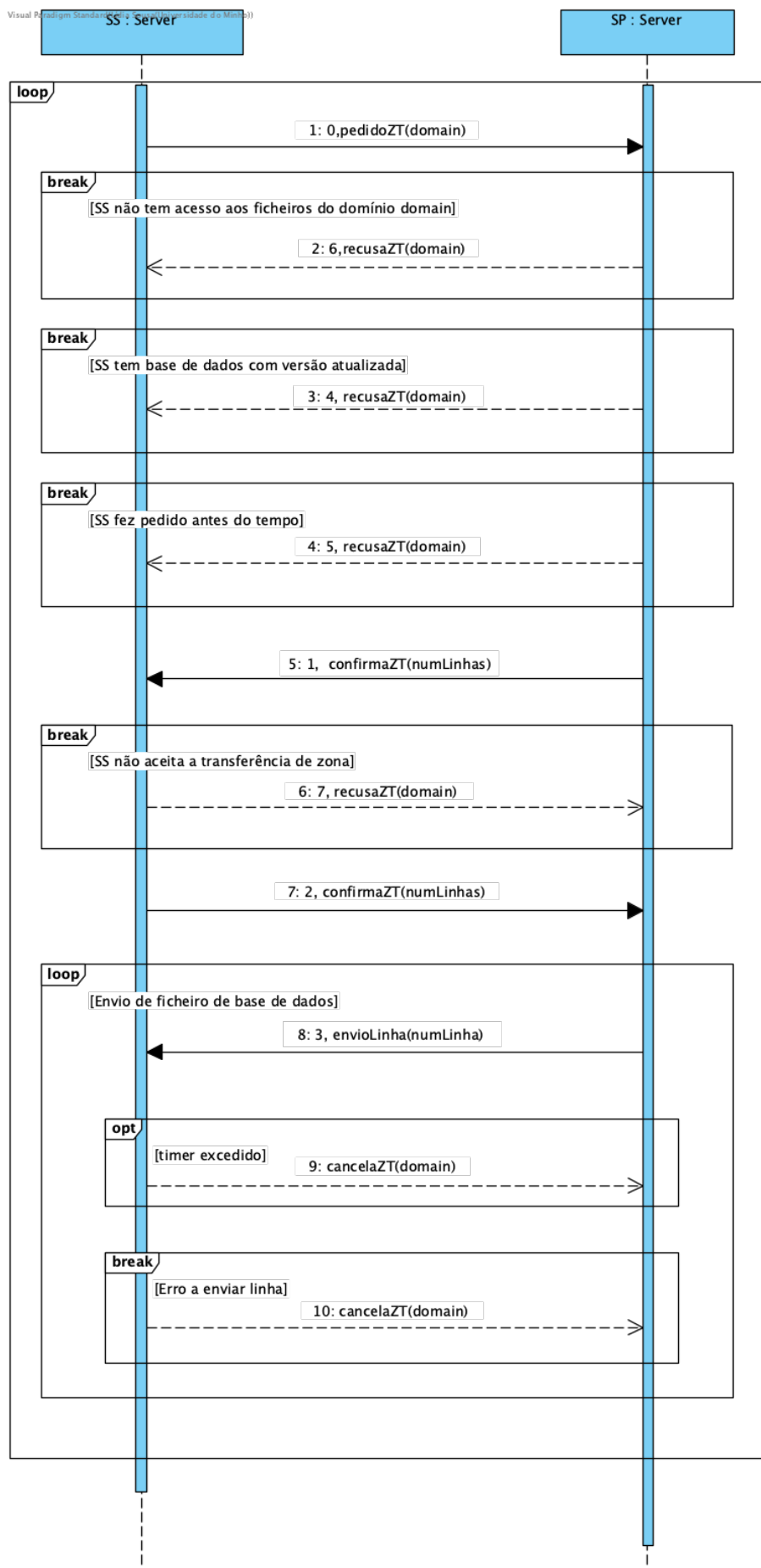
- **Tipo:** código da mensagem
- **Domínio:** nome do domínio
- **Extra:** campo híbrido onde, dependendo do tipo, pode ter várias funções

Segue-se o significado das variações dos valores do setor **Tipo**:

- **0:** Pedido do Servidor Secundário sobre o ficheiro *.db* de um determinado domínio.
- **1:** Confirmação do Servidor Primário e envio do número de linhas do ficheiro.
- **2:** Confirmação do Servidor Secundário, afirmando que quer receber o ficheiro *.db*
- **3:** Envio de uma linha do ficheiro *.db* por parte do Servidor Primário, com indicação do número da linha.
- **4:** Indica que o SS já tem a versão mais recente dos dados, pelo que não é necessária realizar a transferência de zona
- **5:** Servidor primário não aceita o pedido do servidor secundário porque não esperou o valor de **SOARETRY** em milissegundos
- **6:** Servidor primário não aceita o pedido visto que o servidor que pediu não é um secundário de determinado domínio.
- **7:** Servidor secundário não pretende receber o ficheiro *.db*

Na segunda fase do trabalho implementou-se os valores de timers **SOARETRY**, bem como **SOAEXPIRE** e a verificação de versões da base de dados.

Tendo por base o servidor primário do domínio LadoA e respetivos servidores secundários, apresentamos de seguida um diagrama de sequência que represente o processo de transferência de zona. Este diagrama de sequência é baseado no apresentado no enunciado do trabalho.



3.2 Modelo de informativo

3.2.1 Situações de erro

Parsers

Relativamente aos parsers, estes não foram alterados de forma significativa e como tal, as situações de erro a ser reportadas são exatamente as mesmas que na fase anterior.

Comunicação cliente-servidor

Na comunicação cliente-servidor podem ocorrer alguns erros, nomeadamente erros a criar o servidor (além dos erros referidos relativos a parsers). Pode haver erros no que toca à criação de socket, pelo que se apresenta no terminal uma referência ao erro, a atividade é colocada no ficheiro de log e o programa relativo ao servidor termina.

Por outro lado, pode haver um erro diretamente relacionado com a comunicação que consiste em problemas com os pacotes enviados e recebidos. No cliente, a mensagem a ser serializada e posteriormente enviada pode estar mal construída pelo que o erro é reportado e nenhum pacote é enviado. Pode também acontecer de o cliente estar a enviar o pacote para endereço IP e/ou portas inadequadas pelo que o pacote não é enviado. E, por outro lado, no servidor, aquando da passagem para o handler pode haver um erro no que toca aos pacotes recebidos que pode surgir de uma desserialização incorreta da mensagem recebida.

No caso de pacotes com erros a *thread* que está a tratar do envio/receção desses pacotes termina.

De seguida, apresentamos tabelas com uma descrição do erro, a classe em que este acontece e se este erro termina com o programa ou não.

Tabela 3.1: Erros do Cliente

Descrição do erro	Classe	Método	Término
Erro no envio do pedido se houver um problema com a ligação à rede ou se o servidor não estiver a funcionar. (IOException)	Cliente	sendEcho	Não
Erro na receção da resposta a um pedido se houver um problema com a ligação de rede ou se o servidor tiver deixado de responder (IOException)		receiveEcho	Não
Quando se chama o construtor de mensagens pode dar erro se a classe do objecto tiver sido alterada ou removida, ou se o objeto tiver sido serializado de forma incorreta. (ClassNotFoundException)		resolver	Sim
Quando ocorre um timeout enquanto se espera por um pacote. Isto pode acontecer se o servidor demorar demasiado tempo a responder à consulta. (SocketTimeoutException)		resolver, receiveEcho	Sim
Número de argumentos de linha de comando for inferior a 3 ou superior a 4.		main	Sim
Se o endereço do servidor e a porta do primeiro argumento de linha de comando. (NumberFormatException, UnknownHostException)		main	Sim

Tabela 3.2: Erros do Servidor

Descrição do erro	Classe	Método	Término
Erro a criar o DatagramSocket no construtor. Isto pode acontecer se a porta especificada já estiver em uso, ou se houver um problema com a rede. (SocketException)	Server	Construtor	Não
Host name não pode ser resolvido para um endereço IP. Isto pode acontecer se o host name for inválido, ou se houver um problema com a rede. (UnknownHostException)		Construtor	Não
O ficheiro especificado no parâmetro configFile não pode ser encontrado. Isto pode acontecer se o ficheiro não existir ou se o path para o ficheiro estiver incorrecto. (FileNotFoundException)		parseConfigFile	Não
Ocorre um erro durante o parse do ficheiro de configuração. Isto pode acontecer se o ficheiro não estiver no formato correcto, ou se houver um erro nos dados contidos no ficheiro. (IllegalStateException)		parseConfigFile	Não

Tabela 3.3: Erros do Handler

Descrição do erro	Classe	Método	Término
Erro na desserialização de pacote. (ClassNotFoundException)	Handler	run	Não
Ocorre um erro de input/output. Ao ler ou escrever no ficheiro de logs, ou ao enviar uma mensagem através do DatagramSocket. (IOException)		run	Não
Ocorre um timeout numa socket. Se o método que recebe o DatagramSocket fecha ou dá timeout enquanto espera por uma resposta. (SocketTimeoutException)		run	Não
Se o campo SrvData não contiver o domínio ou tipo especificado. (NullPointerException)		queryResponse	Não
Argumento ilegal ou inapropriado passado. Se o campo de parâmetro de um objecto de Dados na lista de valores extra não for um nome de domínio válido. (IllegalArgumentException)		queryResponse	Não

Transferência de zona

Pode haver erros do tipo IOException se ocorrer um erro durante a leitura ou escrita dos sockets streams. Isto pode ocorrer se a ligação ao servidor for perdida, ou se houver um problema com a rede.

Pode ocorrer um erro enquanto se tenta analisar um array de caracteres como um número. Isto pode ocorrer se não for um número válido. (NumberFormatException) Ao analisar a String recebida do DataInputStream para um número inteiro usando Integer.parseInt(), uma NumberFormatException pode ser lançada se a string não contiver um número inteiro.

Quando uma aplicação tenta utilizar uma referência de objeto que tem o valor nulo. Isto pode ocorrer se o lastUpdate for nulo, se o map SOAData for nulo, entre outros. (NullPointerException)

Sistema de *cache*

Tabela 3.4: Erros da Cache

Descrição do erro	Classe	Método	Término
Se a thread que chama o método lock no objeto ReentrantLock não possuir o lock. (IllegalMonitorStateException)	Cache	put	Não
Se a key for nula, há uma tentativa de utilizar uma referência de objeto com valor nulo. (NullPointerException)		get	Não
Se o campo prev ou o next do objeto Node for nulo. (NullPointerException)		removeNode	Não
Se o campo prev do objeto Node for nulo. (NullPointerException)		addToFront	Não
Se o campo prev ou o next do objeto Node for nulo. (NullPointerException)		moveToFront	Não

Capítulo 4

Implementação

4.1 Funcionalidades dos componentes

4.1.1 Cliente

A implementação do cliente assenta numa estratégia simples que consiste em iniciar um cliente com qualquer porta disponível na máquina local. De seguida o cliente envia uma mensagem a um servidor.

A implementação que desenvolvemos pressupõe um *resolver* agnóstico de qualquer domínio associado diretamente ao cliente. Este resolver, enquanto não obtenha o *response code* de 0, ou erro, vai tratar de reenviar a query conforme as informações que recebe e, por fim, enviar a resposta obtida ao cliente. O cliente pode definir o servidor para o qual pretende enviar a informação. O método **resolver** processa a resposta DNS. Se o código de resposta for zero, indicando que a informação está disponível, devolve a mensagem de resposta. Se o código de resposta for um ou dois, indica que o servidor é incapaz de resolver a consulta e o cliente deve enviar a consulta para outro servidor. Se o cliente não obter resposta em 10s, termina o pedido.

1. Cliente insere na linha de comandos a *query*;
2. Proceda-se à verificação do número de argumentos da linha de comando. Se houver menos de 3 ou mais de 4 argumentos, imprime uma mensagem de erro e retorna;
3. Tenta então analisar o endereço e a porta do servidor a partir do primeiro argumento. Se isto falhar imprime uma mensagem de erro e retorna.
4. Define a variável do nome para o segundo argumento de linha de comando, anexando o "." ao final, se ainda não o tiver;
5. Cria uma lista de strings e atribui-lhe as flags;
6. Cria um novo objeto Cliente e atribui-o à variável *c*;
7. Chama o método `sendEcho` com o Cliente *c*, passando no nome, tipo, flags, endereço e porta como argumentos.
8. Inicializa uma resposta da *query* a nula;
9. Introduz um loop que continua até que a resposta deixe de ser nula. Dentro do loop, é usado o método `resolver` para obter a resposta.
10. Imprime a resposta com valores da resposta ou, caso falhe, as mensagens de erro.

4.1.2 Servidor

A implementação do servidor assenta na estratégia de, após proceder ao *parse* devido dos ficheiros e criação do servidor aguardar a receção de um pacote. Quando é recebido um pacote, vindo de determinado cliente, é atribuída a responsabilidade a um *handler* de tratar de obter a resposta à mensagem. A implementação do *handler* surgiu como uma forma de manter o código mais organizado, de fácil compreensão. Cada pedido é associado a uma *thread*. O handler desconstrói o pacote recebido, construindo uma mensagem. Esta mensagem é interpretada de forma a obter as informações pretendidas pelo cliente e é construída uma nova mensagem com estas mesmas informações que será enviada num pacote para o cliente. Quanto à resposta das *query*, no caso do servidor gerir o domínio do qual é questionado envia no campo de Response Values os valores de resposta. Caso contrário, neste campo envia os IPs dos servidores de topo. Aquando da criação do servidor e após o devido *parse* de todos os ficheiros ser concretizado com sucesso é realizado o processo de transferência de zona criando uma *thread* para que o mesmo aconteça.

Transferência de zona

O método do handler da ZT é um método que se destina a lidar com ligações entre dois servidores. Lê em strings a partir de um *DataStream*, processa-as, e escreve strings para um *DataOutputStream*. Espera-se que as strings estejam no formato "type;domain;data" onde type é um número que representa o tipo de mensagem a ser enviada, domain é o domínio a ser referido, e data são dados adicionais relevantes para o tipo de mensagem.

A mensagem do tipo 0 representa um pedido de um servidor secundário (SS) para receber a base de dados (db) para um determinado domínio a partir do servidor primário (SP). O SP verifica se o SS está autorizado a receber a db e responde com uma mensagem indicando se o pedido foi aceite ou rejeitado. Se for aceite, o SP envia uma mensagem com o número de linhas na db para o domínio.

É enviada uma mensagem de tipo 1 que representa uma resposta do SS indicando que está pronto para receber a db. O SP envia as linhas de db para o SS uma de cada vez. De seguida, o tipo 2 representa uma resposta do SS, indicando que recebeu uma linha db do SP. O SP continua a enviar as linhas db até que todas as linhas tenham sido enviadas.

O controlo de intervalos SOARETRY, SOAREFRESH é efetuado e caso seja enviado um pedido do SS para o SP é enviada uma mensagem do tipo 5 (O SP recebeu demasiados pedidos de actualização do SS dentro de um determinado período de tempo. O SP responde com uma mensagem indicando que o pedido foi rejeitado).

Modo iterativo

Neste trabalho implementamos o modo iterativo, explicado ao longo deste trabalho. No entanto, de forma a ficar mais explícito vamos passo-a-passo explicar de que forma fizemos esta funcionalidade.

O método de execução da classe Handler processa uma mensagem DNS recebida. Começa por extrair as flags, o nome do domínio e o tipo da mensagem.

De seguida, verifica a cache para ver se já foi armazenada uma resposta a esta consulta. Se tiver sido, devolve a resposta em cache e escreve um evento no registo. Se não for encontrada resposta na cache, chama o método *queryResponse* para gerar uma nova resposta com base nos dados armazenados. Finalmente, envia a resposta de volta para o remetente da consulta original.

O método *queryResponse* gera uma resposta a uma consulta DNS para um domínio e tipo. Faz isto, primeiro recuperando uma lista de valores que correspondem ao domínio e tipo de consulta, e depois uma lista de entradas de "autoridade" que correspondem ao domínio e têm o tipo "NS". Obtém também uma lista de valores "extra" que correspondem ao domínio e que têm o tipo "A". Finalmente, constrói e devolve um objeto Message que inclui as flags da consulta original, um código de resposta de 0 (indicando que não há erro e que a resposta contém informação que responde diretamente à consulta), o número de valores, autoridades, e valores extra, o domínio e tipo da consulta, e as listas de valores, autoridades, e valores extra.

O remetente de consulta original é o cliente, no entanto, tal como explicado anteriormente, este cliente tem uma espécie de resolver que vai controlar se a resposta é a final ou não e caso não seja,

vai tratar de reenviar a query para o servidor que tem mais informações acerca da query.

Cache

No contexto deste trabalho vai tratar-se a implementação de um sistema de *cache* positiva simples em memória volátil nos servidores. Serão guardadas em *cache* as respostas positivas a uma *query* feita anteriormente por um tempo designado TTL (tempo de validade - *time to live*). Ou seja, o TTL consiste no tempo que os dados existem na *cache* de um servidor. O sistema de *cache* que pretendemos implementar funcionará à partida de forma intuitiva, isto é, quando o cliente efetua uma *query*, antes de consultar a base de dados, é consultada a *cache*. Caso a resposta a esta *query* não se encontre na *cache*, é feito o *load* com o devido tempo e a *query* é efetuada a um ST. Caso contrário, a resposta é obtida na *cache* e enviada ao cliente.

A *cache* tem um HashMap que mapeia uma chave para um objeto Node. A classe Node armazena a chave, o valor (um objeto Message), e o TTL da entrada da *cache*. A *cache* tem uma capacidade fixa e utiliza uma lista duplamente ligada para seguir a ordem das entradas. Pelo facto de se tratar de uma lista duplamente ligada a *cache* pressupõe sempre uma capacidade igual ou superior a 3 nodos.

A adição de uma entrada, ou atualização de entradas funciona da seguinte forma: Se a entrada a ser adicionada/atualizada não estiver na *cache*, adicionamos a nova entrada da *cache* à parte da frente da lista duplamente ligada e estabelecemos a sua prioridade. Se a *cache* estiver cheia, removemos a entrada menos usada recentemente com a prioridade mais baixa até encontrarmos uma entrada com prioridade inferior à que está a ser inserida, ou até ficarmos sem entradas. Se ficarmos sem entradas, não removemos nenhuma entrada e simplesmente inserimos a nova entrada.

Caso a entrada já esteja na *cache* atualizamos o valor e a prioridade da entrada da *cache* e movemo-la para a frente da lista duplamente ligada.

Ou seja, os aspetos principais que tornam a *cache* mais funcional são:

- Todos os nodos da *cache* têm prioridade e como tal, quando a capacidade é excedida, são removidos primeiramente nodos com prioridade inferior.
- Caso os nodos tenham a mesma prioridade, o método de expulsão é LFU (Least Frequently Used).
- Cada nodo tem também um TTL e caso este seja excedido, o nodo é removido.
- Tal como definido no enunciado, quando um SP é iniciado são colocadas todas as entradas que este tenha na sua DB na *cache*.
- Da mesma forma, o SS coloca na *cache* todas as entradas que recebe do SP.

4.2 Outras funcionalidades

4.2.1 Modo debug

O modo debug, relativamente ao modo normal, apresenta como diferença o facto do cliente ter acesso aos logs. Como tal, caso o cliente introduza esta flag, para além da resposta imprimida em *formato prettify* são imprimidos os logs.

4.2.2 Parsers

Quando é inicializado um servidor, é dado como argumento a diretoria do ficheiro config associado. É inicializado o config *parser* que irá ler o ficheiro config associado e irá guardar os valores segundo o dominio apresentado numa estrutura de dados, assim como a diretoria do ficheiro de logs e STlist em variáveis estáticas. Após isto, será inicializado um data *parser*, que verificará se o servidor inicializado é um servidor primário e caso seja, irá ler por dominios os ficheiros de dados associados e guarda os valores em 3 estruturas de dados. Uma para macros, outra para flags "SOA" e a última para o resto dos valores, normalmente pedidos por queries. No final é lido o ficheiro STlist e guardado os seus valores na estrutura de dados associada. A diretoria já foi guardada anteriormente pelo config *parser*.

Capítulo 5

Análise de Testes

5.1 Ambiente de teste

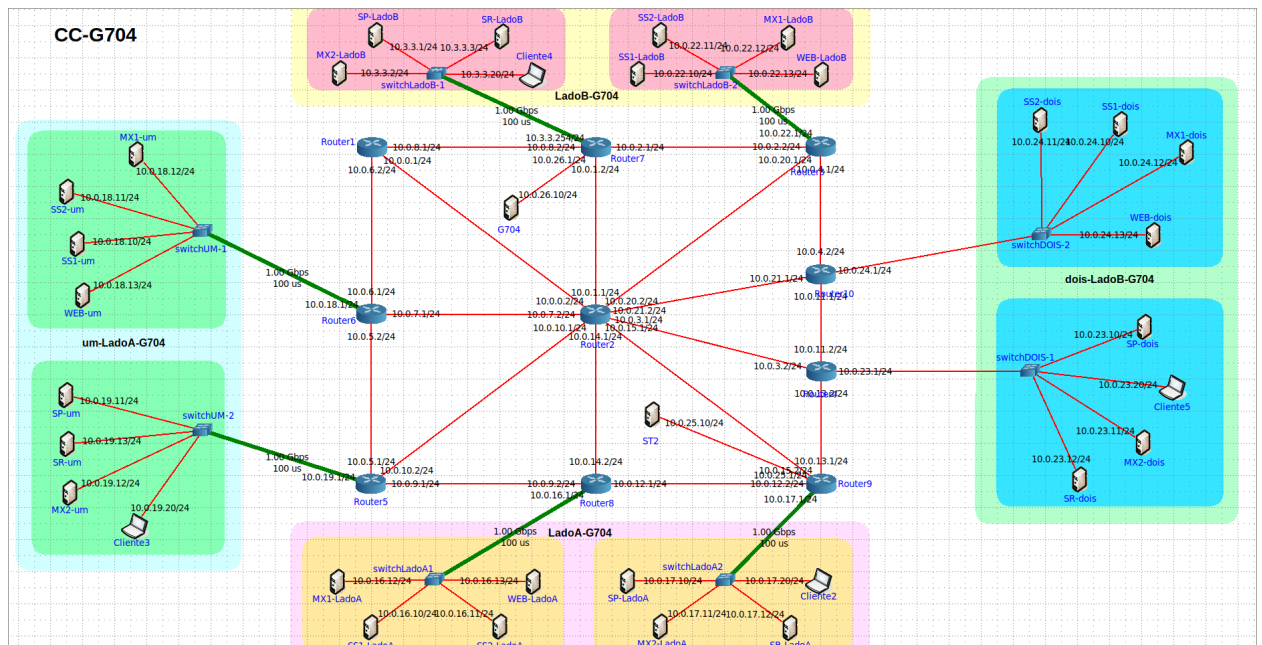


Figura 5.1: Esquema Base - Topologia

A topologia CORE construída foi desenvolvida de forma a ser intuitivo visualmente. De forma sucinta, temos 2 ST e os domínios G704, LadoA, LadoB, um e dois. O Lado A e Lado B são subdomínios do G704 e os subdomínios um e dois são subdomínios do Lado A e Lado B, respetivamente.

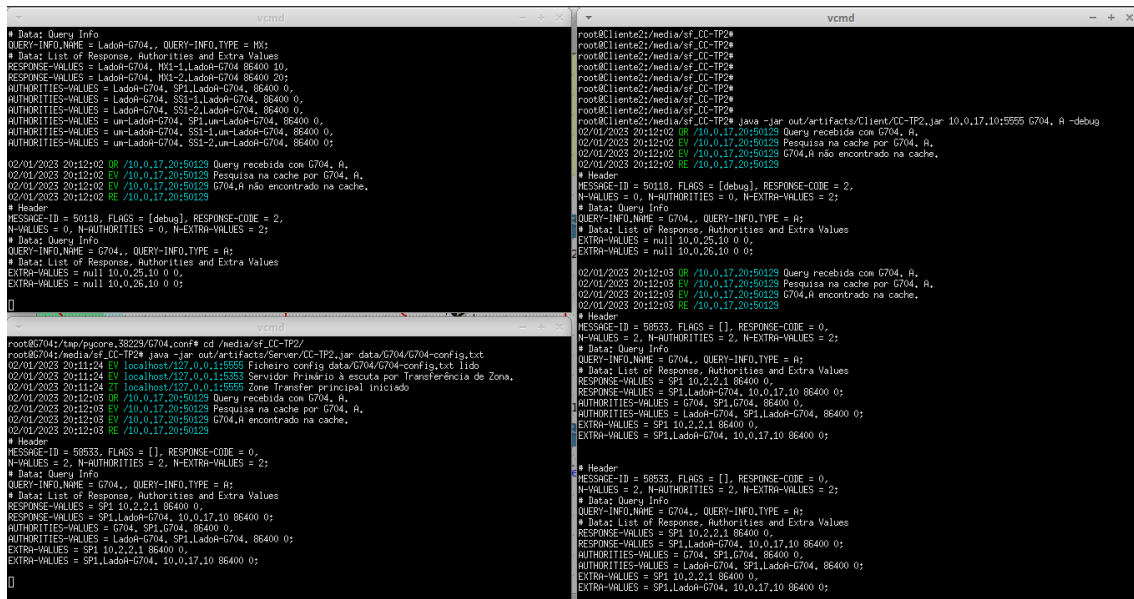
As máquinas MX e Web foram colocadas de forma a ser mais fácil construir os ficheiros de bases de dados com os IPs corretos.

5.2 Testes

Os testes efetuados no CORE consistiram em quatro *queries* efetuadas a diferentes servidores:

5.2.1 Query efetuada acerca de ST

Esta query foi feita ao SP do LadoA relativamente às entradas A do domínio G704. A query é recebida no terminal superior à direita (LadoA), este não encontra na sua cache informações e como tal mostra a mensagem que vai enviar ao cliente com informação dos servidores que deve contactar (servidores de topo). De seguida, o servidor de G704. recebe a query e faz o mesmo processo, enviando por fim ao cliente a resposta acerca do domínio G704. do tipo A.



```
# Data: Query Info
QUERY-INFO.NAME = LadoA-G704., QUERY-INFO.TYPE = MX;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = LadoA-G704. MX1-LadoA-G704 86400 10;
RESPONSE-VALUES = LadoA-G704. MX2-LadoA-G704 86400 20;
AUTHORITIES-VALUES = LadoA-G704. SP1-LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704. SSI-1-LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704. SSI-2-LadoA-G704 86400 0;
AUTHORITIES-VALUES = un-LadoA-G704. SP1-un-LadoA-G704 86400 0;
AUTHORITIES-VALUES = un-LadoA-G704. SSI-1-un-LadoA-G704 86400 0;
AUTHORITIES-VALUES = un-LadoA-G704. SSI-2-un-LadoA-G704 86400 0;
EXTRA-VALUES = null 10.0.25.10 0 0;

02/01/2023 20:12:02 WR /10.0.17.20:50129 Query recebida com G704. A.
02/01/2023 20:12:02 EV /10.0.17.20:50129 Pesquisa na cache por G704. A.
02/01/2023 20:12:02 EV /10.0.17.20:50129 G704.A não encontrado na cache.
02/01/2023 20:12:02 RE /10.0.17.20:50129

# Header
MESSAGE-ID = 50118, FLAGS = [debug], RESPONSE-CODE = 2,
N-VALUES = 0, N-AUTHORITIES = 0, N-EXTRA-VALUES = 2;
# Data: Query Info
QUERY-INFO.NAME = G704., QUERY-INFO.TYPE = A;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = null 10.0.25.10 0 0;
EXTRA-VALUES = null 10.0.25.10 0 0;

02/01/2023 20:12:03 WR /10.0.17.20:50129 Query recebida com G704. A.
02/01/2023 20:12:03 EV /10.0.17.20:50129 Pesquisa na cache por G704. A.
02/01/2023 20:12:03 EV /10.0.17.20:50129 G704.A encontrado na cache.
02/01/2023 20:12:03 RE /10.0.17.20:50129

# Header
MESSAGE-ID = 50533, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 2, N-AUTHORITIES = 2, N-EXTRA-VALUES = 2;
# Data: Query Info
QUERY-INFO.NAME = G704., QUERY-INFO.TYPE = A;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = SP1-LadoA-G704 10.0.17.10 86400 0;
AUTHORITIES-VALUES = LadoA-G704. SP1-LadoA-G704 86400 0;
EXTRA-VALUES = SP1 10.2.2.1 86400 0;
EXTRA-VALUES = SP1-LadoA-G704 10.0.17.10 86400 0;

# Header
MESSAGE-ID = 50533, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 2, N-AUTHORITIES = 2, N-EXTRA-VALUES = 2;
# Data: Query Info
QUERY-INFO.NAME = G704., QUERY-INFO.TYPE = A;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = SP1-LadoA-G704 10.0.17.10 86400 0;
AUTHORITIES-VALUES = LadoA-G704. SP1-LadoA-G704 86400 0;
EXTRA-VALUES = SP1 10.2.2.1 86400 0;
EXTRA-VALUES = SP1-LadoA-G704 10.0.17.10 86400 0;
```

Figura 5.2: Query acerca G704. ao LadoA-G704.

5.2.2 Query efetuada a SR do domínio da query

Esta query foi feita ao resolver do domínio LadoA-G704. acerca das entradas MX. O resolver envia ao cliente informação da sua entrada DD de forma a que este reencaminhe a query. De seguida, o servidor do LadoA-G704. responde com as informações da query, de forma autoritativa.


```

root@sp1-ladoa1:/media/ef/CC-TP2: java -jar out/artifacts/Server/CC-TP2.jar data/LadoA/LadoA-config.txt
02/01/2023 20:06:20 EV localhost/127.0.0.1:5555 Fichero config data/LadoA/LadoA
02/01/2023 20:06:20 EV localhost/127.0.0.1:5555 Fichero de ST lido
02/01/2023 20:06:20 EV localhost/127.0.0.1:5555 Servidor Principal a escuta por T
02/01/2023 20:06:21 TI localhost/127.0.0.1:5555 Zone Transfer principal iniciado
02/01/2023 20:06:23 EV /10.0.17.20:49666 Query recebida com LadoA-G704, MX,
02/01/2023 20:06:23 EV /10.0.17.20:49666 Pesquisa na cache por LadoA-G704, MX,
02/01/2023 20:06:23 EV /10.0.17.20:49666 LadoA-G704_MX encontrado na cache.
# Header
MESSAGE-ID = 42483, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 2, N-AUTHORITIES = 6, N-EXTRA-VALUES = 0;
# Data: Query Info
QUERY-INFO.NAME = LadoA-G704., QUERY-INFO.TYPE = MX;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = LadoA-G704, MX-1,LadoA-G704 86400 10;
RESPONSE-VALUES = LadoA-G704, MX-2,LadoA-G704 86400 20;
AUTHORITIES-VALUES = LadoA-G704, SP1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-2,LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
02/01/2023 20:08:18 TI /10.0.16.10:5353 Transferência de zona iniciada do domínio LadoA-G704..
# Header
MESSAGE-ID = 40012, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 2, N-AUTHORITIES = 6, N-EXTRA-VALUES = 0;
# Data: Query Info
QUERY-INFO.NAME = LadoA-G704., QUERY-INFO.TYPE = MX;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = LadoA-G704, MX-1,LadoA-G704 86400 10;
RESPONSE-VALUES = LadoA-G704, MX-2,LadoA-G704 86400 20;
AUTHORITIES-VALUES = LadoA-G704, SP1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-2,LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
02/01/2023 20:08:17 EV localhost/127.0.0.1:5555 Fichero config data/LadoA/SSI-LadoA-config.txt lido
02/01/2023 20:08:17 EV localhost/127.0.0.1:5555 Fichero de ST lido
02/01/2023 20:08:17 TI localhost/127.0.0.1:5555 Zone Transfer para o servidor 10.0.17.10 iniciado
02/01/2023 20:08:17 EV /10.0.17.10:50874 SS aceite na ZT por SP no domínio LadoA-G704..
02/01/2023 20:08:18 EV /10.0.17.10:50874 Transferência de zona concluída.
02/01/2023 20:08:22 EV /10.0.17.10:5353 SP1 Dominio (LadoA-G704.) atualizado.
02/01/2023 20:08:26 EV /10.0.17.20:54273 Query recebida com LadoA-G704, MX,
02/01/2023 20:08:26 EV /10.0.17.20:54273 Pesquisa na cache por LadoA-G704, MX,
02/01/2023 20:08:26 EV /10.0.17.20:54273 LadoA-G704_MX encontrado na cache.
# Header
MESSAGE-ID = 40012, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 2, N-AUTHORITIES = 6, N-EXTRA-VALUES = 0;
# Data: Query Info
QUERY-INFO.NAME = LadoA-G704., QUERY-INFO.TYPE = MX;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = LadoA-G704, MX-1,LadoA-G704 86400 10;
RESPONSE-VALUES = LadoA-G704, MX-2,LadoA-G704 86400 20;
AUTHORITIES-VALUES = LadoA-G704, SP1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-2,LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
02/01/2023 20:08:36 EV /10.0.16.10:5353 Transferência de zona iniciada do domínio LadoA-G704..
# Header
MESSAGE-ID = 40012, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 2, N-AUTHORITIES = 6, N-EXTRA-VALUES = 0;
# Data: Query Info
QUERY-INFO.NAME = LadoA-G704., QUERY-INFO.TYPE = MX;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = LadoA-G704, MX-1,LadoA-G704 86400 10;
RESPONSE-VALUES = LadoA-G704, MX-2,LadoA-G704 86400 20;
AUTHORITIES-VALUES = LadoA-G704, SP1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-1,LadoA-G704 86400 0;
AUTHORITIES-VALUES = LadoA-G704, SS1-2,LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;

```

Figura 5.5: Query ao SS1 do LadoA-G704. sobre o LadoA-G704.

5.2.5 Query efetuada a um servidor de um domínio acerca de um sub-domínio

Esta query foi feita ao SP do LadoA relativamente às entradas A do domínio um-LadoA-G704. O servidor do LadoA-G704. envia ao cliente as informações dos servidores do um-LadoA-G704 (que corresponde ao melhor sufixo que tem na sua base de dados comparado com o domínio da query). Por fim, o servidor do um-LadoA-G704. envia ao cliente as respostas à sua query.

```

02/01/2023 20:13:27 EV /10.0.17.20:53767 Pesquisa na cache por um-LadoA-G704, A,
02/01/2023 20:13:27 EV /10.0.17.20:53767 um-LadoA-G704,A não encontrado na cache.
Sufixo: LadoA-G704,
Sufixo: LadoA-G704,
Sufixo: um-LadoA-G704,
Sufixo: um-LadoA-G704,
Sufixo: um-LadoA-G704,
um-LadoA-G704, SP1,um-LadoA-G704 86400 0 SP1,um-LadoA-G704 10.0.19.11 86400 0
um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0 SS1-1,um-LadoA-G704 10.0.18.10 86400 0
um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0 SS1-2,um-LadoA-G704 10.0.18.11 86400 0
02/01/2023 20:13:27 EV /10.0.17.20:53767
# Header
MESSAGE-ID = 11239, FLAGS = [debug], RESPONSE-CODE = 1,
N-VALUES = 6, N-AUTHORITIES = 3, N-EXTRA-VALUES = 3;
# Data: Query Info
QUERY-INFO.NAME = um-LadoA-G704., QUERY-INFO.TYPE = A;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
EXTRA-VALUES = SP1,um-LadoA-G704 10.0.19.11 86400 0;
EXTRA-VALUES = SS1-1,um-LadoA-G704 10.0.18.10 86400 0;
EXTRA-VALUES = SS1-2,um-LadoA-G704 10.0.18.11 86400 0;
02/01/2023 20:13:27 EV /10.0.17.20:53767 Pesquisa na cache por um-LadoA-G704, A,
02/01/2023 20:13:27 EV /10.0.17.20:53767 um-LadoA-G704,A encontrado na cache.
# Header
MESSAGE-ID = 23688, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 6, N-AUTHORITIES = 3, N-EXTRA-VALUES = 6;
# Data: Query Info
QUERY-INFO.NAME = um-LadoA-G704., QUERY-INFO.TYPE = A;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = SP1 10.0.19.11 86400 0;
RESPONSE-VALUES = SS1-1 10.0.18.10 86400 0;
RESPONSE-VALUES = SS1-2 10.0.18.11 86400 0;
RESPONSE-VALUES = MX-1 10.0.18.12 86400 0;
RESPONSE-VALUES = MX-2 10.0.19.12 86400 0;
RESPONSE-VALUES = MEX1 10.0.18.13 86400 200;
AUTHORITIES-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
EXTRA-VALUES = SP1 10.0.19.11 86400 0;
EXTRA-VALUES = SS1-1 10.0.18.10 86400 0;
EXTRA-VALUES = SS1-2 10.0.18.11 86400 0;
EXTRA-VALUES = MX-1 10.0.18.12 86400 0;
EXTRA-VALUES = MX-2 10.0.19.12 86400 0;
EXTRA-VALUES = MEX1 10.0.18.13 86400 200;
02/01/2023 20:13:27 EV /10.0.17.20:53767 Pesquisa na cache por um-LadoA-G704, A,
02/01/2023 20:13:27 EV /10.0.17.20:53767 um-LadoA-G704,A não encontrado na cache.
# Header
MESSAGE-ID = 23688, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 6, N-AUTHORITIES = 3, N-EXTRA-VALUES = 6;
# Data: Query Info
QUERY-INFO.NAME = um-LadoA-G704., QUERY-INFO.TYPE = A;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = SP1 10.0.19.11 86400 0;
RESPONSE-VALUES = SS1-1 10.0.18.10 86400 0;
RESPONSE-VALUES = SS1-2 10.0.18.11 86400 0;
RESPONSE-VALUES = MX-1 10.0.18.12 86400 0;
RESPONSE-VALUES = MX-2 10.0.19.12 86400 0;
RESPONSE-VALUES = MEX1 10.0.18.13 86400 200;
AUTHORITIES-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
EXTRA-VALUES = SP1 10.0.19.11 86400 0;
EXTRA-VALUES = SS1-1 10.0.18.10 86400 0;
EXTRA-VALUES = SS1-2 10.0.18.11 86400 0;
EXTRA-VALUES = MX-1 10.0.18.12 86400 0;
EXTRA-VALUES = MX-2 10.0.19.12 86400 0;
EXTRA-VALUES = MEX1 10.0.18.13 86400 200;
02/01/2023 20:13:27 EV /10.0.17.20:53767 Pesquisa na cache por um-LadoA-G704, A,
02/01/2023 20:13:27 EV /10.0.17.20:53767 um-LadoA-G704,A encontrado na cache.
# Header
MESSAGE-ID = 23688, FLAGS = [], RESPONSE-CODE = 0,
N-VALUES = 6, N-AUTHORITIES = 3, N-EXTRA-VALUES = 6;
# Data: Query Info
QUERY-INFO.NAME = um-LadoA-G704., QUERY-INFO.TYPE = A;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = SP1 10.0.19.11 86400 0;
RESPONSE-VALUES = SS1-1 10.0.18.10 86400 0;
RESPONSE-VALUES = SS1-2 10.0.18.11 86400 0;
RESPONSE-VALUES = MX-1 10.0.18.12 86400 0;
RESPONSE-VALUES = MX-2 10.0.19.12 86400 0;
RESPONSE-VALUES = MEX1 10.0.18.13 86400 200;
AUTHORITIES-VALUES = um-LadoA-G704, SP1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-1,um-LadoA-G704 86400 0;
AUTHORITIES-VALUES = um-LadoA-G704, SS1-2,um-LadoA-G704 86400 0;
EXTRA-VALUES = SP1 10.0.19.11 86400 0;
EXTRA-VALUES = SS1-1 10.0.18.10 86400 0;
EXTRA-VALUES = SS1-2 10.0.18.11 86400 0;
EXTRA-VALUES = MX-1 10.0.18.12 86400 0;
EXTRA-VALUES = MX-2 10.0.19.12 86400 0;
EXTRA-VALUES = MEX1 10.0.18.13 86400 200;

```

Figura 5.6: Query a servidor do LadoA-G704. sobre o um-LadoA-G704.

Capítulo 6

Participação do grupo

De seguida apresentamos a tabela que representa o esforço e dedicação de cada elemento do grupo a cada um dos elementos de avaliação deste trabalho.

Tarefa	A93205	A93316	A73591
B1.	0.5	0.5	0
B2.	0.7	0.3	0
B3.	0.7	0.3	0
B4.	0.2	0.8	0
B5.	0.45	0.45	0.1
B6.	0.4	0.4	0.2
B7.	0	1	0
B8.	0.5	0.5	0

Tabela 6.1: Tabela de participação

Capítulo 7

Conclusão

A realização deste trabalho permitiu o desenvolvimento de um sistema de DNS com as funcionalidades descritas no enunciado do trabalho. Este trabalho também permitiu consolidar o conhecimento acerca dos sistemas DNS e do funcionamento do mesmo.

Reconhecemos melhorias na implementação bem como na legibilidade do código, no entanto consideramos que desenvolvemos com sucesso o trabalho.