

Comunicações por Computador

Trabalho Prático 2 (TP2) - Parte B

Implementação de Sistema DNS

Grupo 1 - PL5

Guilherme Martins (A92847)

Millena Freitas (A97777)

Vasco Oliveira (A96361)

Data de entrega : 2 de Janeiro de 2023



3º ano - 1º semestre
Licenciatura em Engenharia Informática
Departamento de Informática
Universidade do Minho
2022/2023

Conteúdo

1	Introdução	4
2	Arquitetura do Sistema	5
2.1	Arquitetura da Aplicação	6
2.2	Requisitos funcionais - Todos os componentes	6
2.3	Requisitos funcionais - Servidor Primário	7
2.4	Requisitos funcionais - Servidor Secundário	7
2.5	Requisitos funcionais - Transferência de Zona (SP e SS)	7
2.6	Requisitos funcionais - Servidor de Resolução	8
2.7	Requisitos funcionais - Cliente	8
2.8	Implementação de multithreading	9
2.9	Implementação da Cache	9
2.10	Implementação do SR	10
3	Modelo de Informação	11
3.1	Ficheiro de configuração dos servidores SP, SS e SR	11
3.2	Ficheiro com a lista de ST (<i>Servidores de Topo</i>)	12
3.3	Ficheiros de <i>log</i>	12
3.4	Ficheiro de dados do SP (<i>Servidor Primário</i>)	13
3.5	PDU: Formato das mensagens de DNS	15
3.6	Mecanismo de Codificação Binária	15
4	Modelo de Comunicação	16
4.1	Representação lógica da Mensagem de DNS	16
4.2	Envio e Receção de queries	16
4.2.1	Exemplos usando um SR	17
4.3	Tipos de funcionamento	18
4.4	Transferência de Zona	18
4.5	Implementação da Transferência de Zona	18
4.6	Exemplo de transferência de zona	19
4.7	Comportamento dos elementos em situação de erro	20
4.8	Ameaças de segurança	20
5	Ambiente de Teste	21
5.1	Transferência de Zona	22
5.1.1	Procedimento de transferência de zona	22
5.1.2	Teste de queries para o SS após a transferência de zona	23
5.2	Queries	23
5.2.1	MX	23
5.2.2	NS	24
5.2.3	Subdomínio	24
5.2.4	A	24
5.2.5	CNAME	25
5.2.6	Erros	25
5.2.7	Querie SR com DD	26
5.2.8	Querie SR usando o ST	27
5.2.9	Prova de Resiliência	29
6	Tabela de Atividades	30
7	Conclusão	31
8	Bibliografia	33

9	Anexos	34
9.1	Manual de utilização	34
9.1.1	Cliente	34
9.1.2	Servidor	35

Lista de Figuras

1	Arquitetura hierárquica DNS	5
2	Arquitetura da aplicação a ser desenvolvida	6
3	Lógica implementada para o multithreading	9
4	Exemplo de ficheiro de configuração para o SP do domínio .greens.dance.	12
5	Ficheiro com os respetivos IPs dos servidores de topo	12
6	Exemplo de ficheiro de base de dados para o domínio .greens.dance.	14
7	Envio/Resposta de uma query entre um SP e um CL	16
8	Envio/Resposta de uma query entre um SR e um CL através de DD	17
9	Envio/Resposta de uma query entre um SP e um CL através do ST	17
10	Exemplo de transferência de zona dentro do domínio .blacks.	19
11	Topologia do Ambiente de Teste	21
12	Transferência de zona entre SP e SS1 do domínio greens	22
13	Testes no SS1	23
14	Query MX	23
15	Query NS	24
16	Teste para subdomínio como NAME	24
17	Query A	24
18	Query CNAME	25
19	Teste Erro 2	25
20	Teste Erro 3	25
21	Servidor Primário .greens.dance.	26
22	Servidor de Resolução no domínio .greens.dance.	26
23	Resultado final no Cliente	26
24	Cliente	27
25	ST .dance.	27
26	SP .blacks.dance.	27
27	SR .greens.dance.	28
28	SR .greens.dance / Cliente	29
29	SS .greens.dance.	29
30	Tabela de Atividades	30
31	Argumentos para executar o Cliente	34
32	Argumentos para executar o Servidor Primário	35
33	Argumentos para executar o Servidor Secundário	35
34	Argumentos para executar o Servidor de Resolução	35

1 Introdução

Este relatório foi realizado no âmbito da Unidade Curricular de *Comunicações por Computador*, inserido no 3º ano - 1º semestre da licenciatura em Engenharia Informática da Universidade do Minho. O respetivo trabalho prático (TP2) teve os seguintes objetivos:

- Consolidar conhecimentos acerca do serviço DNS da arquitetura TCP/IP e sobre os protocolos de transporte UDP e TCP.
- Especificar um PDU e as primitivas de serviço de um protocolo aplicacional utilizando um protocolo de transporte orientado à conexão e um protocolo de transporte não orientado à conexão.
- Aplicar e consolidar conhecimentos de programação de aplicações distribuídas utilizando o paradigma dos *sockets*.

O presente relatório está estruturado nos seguintes capítulos:

- **Arquitetura do Sistema:** onde trata de apresentar e descrever os componentes do sistema, com os respetivos requisitos funcionais e os módulos que os compõem.
- **Modelo de Informação:** apresenta-se a especificação completa da sintaxe e da semântica de todos os ficheiros, para além do comportamento dos elementos em situação de erro de leitura, do PDU/mensagens DNS e, por fim, do mecanismo de codificação binária que será implementado na próxima fase do trabalho.
- **Modelo de Comunicação:** é feita a especificação completa de todas as interações possíveis e do comportamento dos elementos em situação de erro.
- **Ambiente de Teste:** inclui uma descrição e especificação completa do ambiente de testes, incluindo o conteúdo dos ficheiros de configuração e de dados; bem como, a apresentação do conjunto de testes realizados para demonstrar o funcionamento dos programas desenvolvidos.
- **Tabela de Atividades:** secção que contém uma tabela que identifica as atividades desenvolvidas ao longo do trabalho prático e o nível de participação de cada elemento do grupo nessas atividades.

No final do relatório, apresenta-se uma conclusão do nosso trabalho na qual apontamos as nossas opiniões sobre este projeto e sobre as dificuldades que tivemos de ultrapassar, bem como uma bibliografia com todas as referências utilizadas para este relatório.

Para além disso, nos anexos do relatório,

2 Arquitetura do Sistema

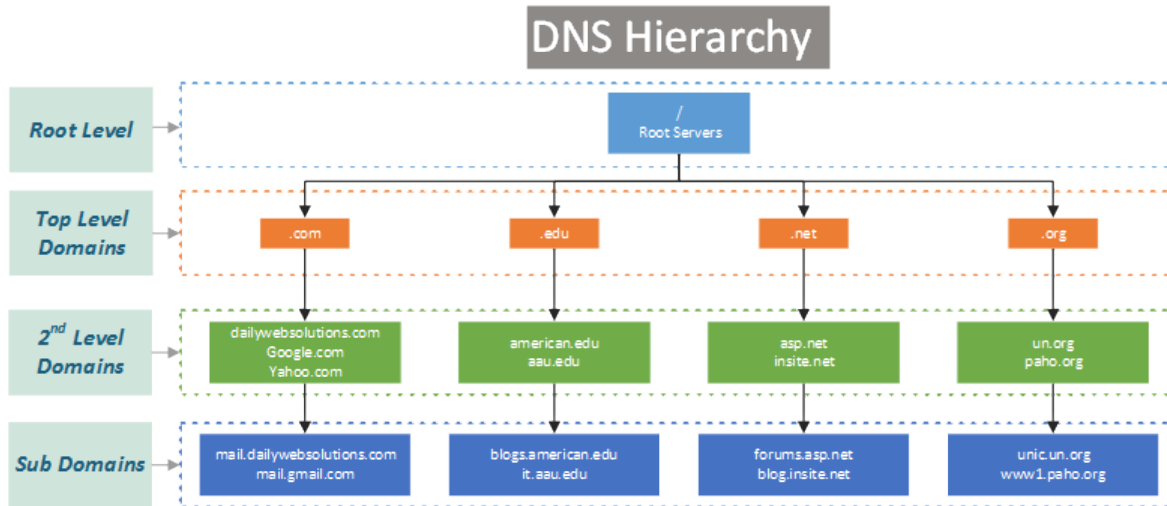


Figura 1: Arquitetura hierárquica DNS

Para a realização deste projeto é necessário identificar os componentes fundamentais para que este obtenha êxito e respeite as normas que especificam o sistema DNS, tal como se pode verificar na figura 6. Portanto, há quatro tipos de elementos essenciais que podem interagir: *Servidor Primário* (SP), *Servidor Secundário* (SS), *Servidor de Resolução* (SR) e *Cliente* (CL).

Como será retratado detalhadamente posteriormente, há diversos domínios e subdomínios constituídos por um ou mais destes servidores cada, portanto é de grande importância que estes componentes de software possam executar em simultâneo mais do que um tipo de servidor. Por exemplo, para cada domínio há um Servidor Primário e dois Servidores Secundários e, conseqüentemente, o componente pode executar um ou mais SP, um ou mais SS e um ou mais SR.

Para garantir o funcionamento e a fiabilidade do serviço DNS, há também mais duas variantes especiais de servidores: os Servidores de Topo (ST), também conhecidos por *DNS root servers*, e os Servidores de Domínios de Topo (SDT), também conhecidos por *DNS Top-Level Domain servers*.

Como foi abordado anteriormente, a administração do DNS é estruturada em uma hierarquia usando diferentes áreas ou zonas gerenciadas. Quando se trata dos ST, trata-se da zona raiz no topo desta hierarquia. Estes servidores possuem informação acerca de todos os SDT responsáveis por cada domínio de topo.

Para o contexto deste projeto, o comportamento dos SDT é igual aos SP ou aos SS. Portanto, um SP ou SS autoritativos para um domínio de topo é um SDT, ainda que não tenham domínios hierarquicamente acima na árvore DNS. Com relação aos ST, estes são como SP, mas possuem apenas uma base de dados em que, para cada domínio de topo, inclui informação dos SDT respetivos, ou seja, os nomes e os endereços IP dos seus SS e do seu SP. Devido a estes comportamentos explicados, tanto os ST quanto os SDT são implementados com o mesmo componente que implementa um SP ou um SS.

2.1 Arquitetura da Aplicação

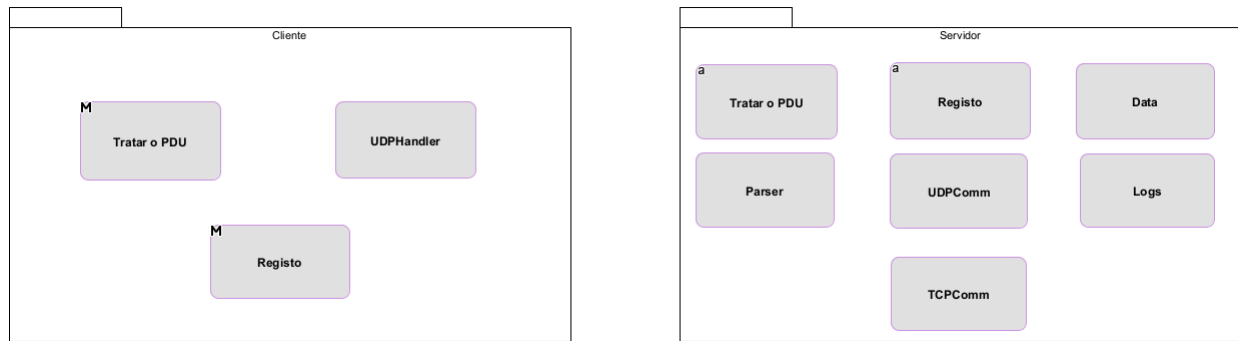


Figura 2: Arquitetura da aplicação a ser desenvolvida

Tal como se pode observar na figura 2, tanto o Cliente quanto o Servidor deve possuir um módulo capaz de tratar o PDU, pois este representa a estrutura utilizada no modelo de comunicação dos serviços DNS. Portanto, ambos devem ser capazes de ler, tratar e interpretar a informação contida nos PDUs. O PDU é composto por Registos, portanto deve haver também um módulo responsável por tratá-los. Cada Registo possui toda a informação contida numa linha da base de dados.

Também devem ser capazes de efetuar comunicações via UDP, visto que este é o protocolo de transporte utilizado pelo serviço DNS para envio/recepção de queries/respostas.

Com relação ao Servidor, deve haver um módulo capaz de tratar do parser dos ficheiros de configuração e dos ficheiros de base de dados quando se tratar de um Servidor Primário. Também deve ser capaz de efetuar operações de procuras nos dados armazenados até então de modo a processar as respostas para as queries recebidas.

Como toda a atividade é registada em formato de Logs nos seus respetivos ficheiros de logs e também no terminal quando em modo debug, também torna-se essencial um módulo capaz de criar estes logs no formato requisitado que será abordado posteriormente neste documento.

É necessário ter um módulo Data para administrar os dados. Como esta aplicação possua apenas o modo iterativo implementado, a noção de caching apenas se aplica ao SR. Portanto, há uma estrutura de dados compatível tanto para cache quanto para o modelo sem cache. Com a implementação e manipulação de caching, torna-se necessária a capacidade de armazenar e controlar os dados de acordo com seus respetivos TTL (time-to-live), como também desenvolver métodos de procura na cache. Este módulo também é o responsável por procurar respostas às queries, tanto para procura na cache quanto para procura na base de dados.

Por conta do mecanismo de transferência de zona entre os Servidores Secundários e os Servidores Primários, é necessário garantir a capacidade de efetuar comunicações via TCP de modo a garantir a fiabilidade e a transferência de todos os dados, desta forma o SS pode assumir o papel do SP caso ocorra alguma falha que torne este último inapto a responder queries. Da mesma forma, é necessário um mecanismo de controle sobre os pedidos e respostas acerca da transferência de zona.

2.2 Requisitos funcionais - Todos os componentes

Os componentes devem, no mínimo, funcionar em modo debug. Neste modo, toda a atividade referente a estes componentes registada nos logs também é enviada para o *standard output* de forma a facilitar a identificação de erros e incoerências.

Os sistemas de *caching* são indispensáveis de forma a garantir e manter o desempenho do serviço de DNS, visto que é importante responder rapidamente a um pedido. Portanto, todos os servidores devem possuir um sistema simples de *caching* positivo em memória volátil, ou seja, quando as respostas disponibilizam os resultados pedidos. O registo de informação em cache só deve ser efetuado depois de se processar uma resposta recebida a uma query feita anteriormente. O envio de queries não deve estar relacionado com nenhuma atividade de *caching*. Os CL não precisam de qualquer tipo de cache.

Para prevenir a propagação de registos DNS expirados, é necessário associar um TTL *time-to-live* aos registos no sistema de caching. Este TTL é o tempo máximo em segundos que os dados podem existir numa

cache dum servidor. Deste modo, é garantida a credibilidade e coerência dos registos. Caso o TTL não seja suportado num determinado tipo de registo, o seu valor deve ser igual a zero.

Todos os componentes de software devem implementar processos idênticos de leitura de ficheiros de configuração e de processos comunicacionais assíncronos utilizando queries e respostas a queries. Todas estes elementos devem limitar o seu *input* a ficheiros de configuração ou de dados e o seu *output* a ficheiros de log.

Para garantir a coerência e bom funcionamento dos componentes, é necessário passar parâmetros de funcionamento como argumentos na altura do arranque. Portanto, no mínimo, espera-se o suporte a três parâmetros essenciais:

- Quando a porta de atendimento dos servidores for diferente da porta normalizada (53) é necessário fornecer a porta de atendimento que será utilizada.
- O servidor DNS para ser devidamente configurado precisa do valor *timeout* como parâmetro, assim, caso não receba a resposta antes deste valor de tempo limite, encaminhará a query para o próximo servidor em sua lista. Evidencia-se que o servidor que recebe a query precisa ter um tempo razoável para responder a esta query. Caso seja fornecido um valor muito pequeno, este servidor pode não ter tempo o suficiente para responder, fazendo com que a query falhe. Por outro lado, caso seja um valor grande, o servidor DNS poderá aguardar muito tempo para que o outro servidor o responda. Isto pode causar atrasos, o que não é de todo viável de acordo com propósito do DNS.
- É necessário identificar se o componente será executado em modo debug ou não.

2.3 Requisitos funcionais - Servidor Primário

Este servidor DNS deve responder a, e efetuar queries DNS e tem acesso direto à base de dados dum domínio DNS, sendo a autoridade que os gere. Qualquer atualização necessária à informação dum domínio DNS tem de ser feita diretamente na base de dados do SP, de modo a garantir que esta mudança sera refletida corretamente.

De modo a garantir o seu devido funcionamento, deve ter acesso a informação de configuração específica, ou seja, domínios para os quais é SP, portas de atendimento, identificação dos ficheiros das bases de dados, identificação do ficheiro de log, informação de segurança para acesso às bases de dados, identificação dos SS respetivos e dos SP dos subdomínios e endereços dos servidores de topo, para além dos dados respeitantes diretamente ao domínio DNS.

Deve ter como *input* um ficheiro de configuração, um ficheiro de base de dados por cada domínio gerido e um ficheiro com a lista de servidores de topo; e como *output* um ficheiro de log.

Caso se trate de um *Servidor de Topo* (ST), o ficheiro da base de dados deve ser de forma que, para cada domínio de topo, inclui informação dos SDT respetivos.

2.4 Requisitos funcionais - Servidor Secundário

Servidor DNS que responde a, e efetua, queries DNS além de ter autorização e autoridade para possuir uma réplica da base de dados original do SP autoritativo dum domínio DNS. Este servidor tem de ter acesso a informação de configuração específica (domínios para os quais é SS, portas de atendimento, identificação dos SP dos domínios para os quais é SS, identificação do ficheiro de log, informação de segurança para acesso aos SP e endereços dos servidores de topo).

Um Servidor Secundário tem como input um ficheiro de configuração e um ficheiro com a lista de servidores de topo; como output tem um ficheiro de log.

Por razões de segurança, a informação replicada do SP deve ser armazenada apenas em memória volátil no SS.

De forma a garantir a coerência dos seus dados com os contidos no Servidor Primário, deve tentar manter a sua base de dados atualizada. Isto deve ser feito através de **Transferência de Zona**.

2.5 Requisitos funcionais - Transferência de Zona (SP e SS)

- Todas as interações numa operação de transferência de zona devem ser feitas utilizando uma conexão **TCP**, pois deve haver consistência e uma conexão segura e fiável estabelecida entre os servidores, de modo a prevenir o risco de pacotes perdidos ou falsificados.

- Deve utilizar queries normais para saber se a versão da base de dados do respetivo SP é mais atual que a sua. Se for, inicia uma tentativa de transferência de zona enviando o nome completo do domínio para a qual quer receber uma cópia da base de dados do SP.
- O SP deve verificar a validade do domínio e que o SS tem autorização para receber a cópia da sua base de dados. Se for esse o caso, o SP envia o número de entradas do ficheiro de base de dados. O valor máximo deve ser 65535 e as linhas de comentário e linhas em branco não devem ser contadas nem transmitidas.
- Caso o SS aceite receber essa quantidade de entradas deve responder ao SP com o mesmo valor.
- O SP pode enviar todas entradas em formato de texto, tal como estão no ficheiro de base de dados, mas numerando-as por ordem crescente.
- O SS deve ir verificando se recebeu todas as entradas esperadas até um tempo predefinido se esgotar. Quando esse tempo terminar, o SS termina a conexão TCP e desiste da transferência de zona. Deve tentar outra vez após um intervalo de tempo igual a *SOARETRY*.

2.6 Requisitos funcionais - Servidor de Resolução

Servidor DNS que responde a, e efetua, queries DNS sobre qualquer domínio, mas que não tem autoridade sobre nenhum pois serve apenas de intermediário.

Neste projeto devem ser implementados como um servidor DNS que responde aos clientes numa rede IP local.

Estes servidores podem funcionar em cascata, dependendo da gestão da configuração dos clientes, dos provedores de serviços e das instituições. Devem ter acesso a informação de configuração específica como eventuais domínios por defeito e lista dos servidores DNS que deve contactar, porta de atendimento, identificação do ficheiro de log e endereços dos servidores de topo.

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.

2.7 Requisitos funcionais - Cliente

Um Cliente deve obter informação realizando queries DNS a um SR numa lista de SR predefinida. Deve possuir uma lista de SR num ficheiro de configuração (endereços IP e portas de atendimento). O *input* e *output* deste CL será através da linha de comando sem necessidade dum ficheiro de configuração.

Deve ser desenvolvido um CL específico para se consultar o DNS diretamente (à imagem do nslookup e outras aplicações semelhantes).

Quando se trata da aplicação CL, não devem ser usados ficheiros de configuração, de dados ou de logs. O seu *input* deve vir totalmente da linha de comandos ou da interface da própria aplicação. Adicionalmente, o seu *output* deve ser feito totalmente para a interface de standard output ou para a interface da própria aplicação.

Este componente deve suportar uma interação normal com um utilizador humano, permitindo assim a esse utilizador aceder diretamente à informação do sistema DNS desenvolvido, instalado e configurado.

Uma aplicação CL deve ser capaz de indagar um serviço DNS. Para isso, o CL precisa de saber, assim que arranca e em todos os momentos, que servidor DNS usar como destino das suas queries. Esta informação é o primeiro parâmetro passado como argumento na forma dum endereço IP[:porta]. Como o CL usa queries DNS normais, tem de obter do utilizador a informação que tem de utilizar no campo QUERY INFO numa query DNS, i.e., o nome completo do parâmetro NAME e o tipo de valor esperado TYPE OF VALUE. O CL deve utilizar queries no modo iterativo e não ativar a flag R.

O CL deve executar uma query tal como se fosse uma aplicação cliente normal ou um SR. No entanto, o CL não deve implementar nenhum mecanismo de cache. Os resultados da query devem ser mostrados ao utilizador utilizando uma sintaxe/formatação clara e concisa.

O CL deve, por defeito, executar em modo debug, i.e., as mensagens de DNS não são codificadas em binário e são transmitidas no formato de sequência de caracteres conciso. Isto não deve afetar o formato com que a informação é disponibilizada ao utilizador.

2.8 Implementação de multithreading

De modo a garantir paralelismo e tornar o sistema mais eficiente, é necessário implementar multithreading. Para isto, na main thread há criação de uma thread que será responsável pela comunicação TCP, ou seja, tudo relacionado a transferência de zona. O módulo de comunicação TCP também cria uma thread para cada pedido que recebe. Note que apenas o Servidor Primário utiliza este módulo TCP com multithread, visto que o Servidor Secundário apenas envia e recebe de um único Servidor Primário.

A main thread portanto segue para a comunicação UDP. No módulo de comunicação através de UDP, é criada uma thread para atender a cada pedido. Portanto, como todas as threads irão aceder à estrutura de dados que contém a informação necessária para responder à queries, é necessário implementar um controlo de concorrência.

Para isto, o módulo Data possui um ReadWriteLock, no qual permite leituras simultâneas. Apenas uma thread pode executar uma operação de escrita por vez e, enquanto a executa não pode haver outras threads a executar operações de leitura. Isto garante a consistência dos dados. Este lock é utilizado em todos os métodos que busquem ou insiram informação na estrutura de dados.

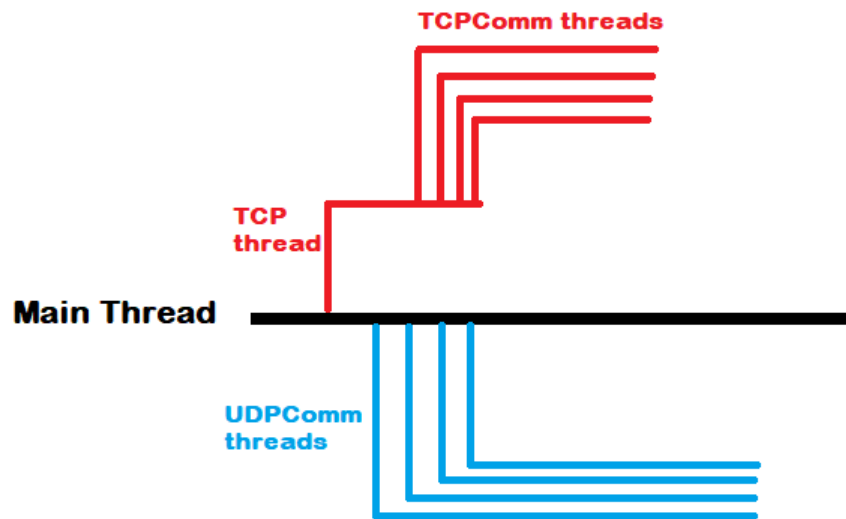


Figura 3: Lógica implementada para o multithreading

2.9 Implementação da Cache

A estrutura de dados utilizada para a cache é a mesma utilizada para guardar os dados de servidores que não suportam o serviço de caching. Esta estrutura é abordada no módulo Data e é um mapa no qual a chave corresponde ao type of value e o valor corresponde a lista de registos que contém este type of value.

É importante enfatizar que os mesmos métodos de pesquisa por type of value, utilizados na primeira fase para servidores que não possuem mecanismo de caching, foram utilizados para pesquisa por resposta na cache. Portanto, não foi necessário código adicional para buscas.

Portanto, a lógica utilizada para procura de respostas contidas na cache foi a seguinte:

- Primeiro procura-se na cache se há uma resposta direta à query recebida.
- Caso tenha obtido resposta, procura por informação sobre os authorities values e extra values referentes a resposta que possam estar contidos na cache. Preenche o PDU com todas estas informações.
- Caso não tenha obtido resposta, procura por uma resposta de referência. Ou seja, faz uma procura por longest-prefix match de modo a tentar chegar na resposta mais rapidamente com menor número

de saltos. Caso encontre uma referência, preenche o PDU com a informação e com rcode 1 e nValues 0, o que indica explicitamente que trata-se de uma referência.

- Caso não encontre uma referência, retorna o PDU nulo.

É necessário manter a cache atualizada, ou seja, percorrer e certificar que quando o TTL expira a linha já não é mais válida para obtenção de resposta. Para evitar o uso de uma thread em background apenas para ser acordada eventualmente e percorrer toda a estrutura, o que consumiria mais recursos e comprometeria a eficiência, decidiu-se implementar este mecanismo ao fazer buscas na cache. Portanto, os métodos de busca na cache por type of value verificam se o TTL expirou, e caso tenha expirado, muda o campo valid para false e este já não será válido para a resposta.

2.10 Implementação do SR

- O SR inicializa um PDU resposta como nulo e uma flag como false. Seu primeiro passo é procurar por uma resposta à query em sua cache.
- Após esta procura, caso o PDU ainda permaneça nulo, inicializa uma Queue. Caso o campo name da query contenha o domínio associado ao SR como domínio por defeito (DD), adiciona à esta queue todos os IP's dados como DD no ficheiro de configuração do SR.
- Começa então por remover da Queue um IP para reencaminhar a query. Caso a queue esteja vazia, significa que não se trata de uma query para o DD e, portanto, adiciona os IP's dos ST's à queue.
- Remove então o IP da queue e reencaminha a query. Caso obtenha um timeout, volta para o início do ciclo onde remove outro IP da queue para reencaminhar a query. Isto foi feito de forma a conferir resiliência para o SR. Pois, caso os DD's falhem, tenta para cada ST até obter uma resposta.
- Quando obtém uma resposta, força sua saída do ciclo.
- Em seguida, verifica se o PDU é nulo. Caso não seja nulo, verifica se trata de uma referência ou uma resposta final. Uma referência é caracterizada pela presença do rcode 1 e do nValues 0.
- Caso seja uma referência, extrai os IP's dos servidores do campo EXTRA VALUES do PDU e os adiciona na Queue. De seguida, tentará enviar para cada um destes até obter uma resposta. Caso tente para todos e não consiga obter uma resposta, tenta novamente para os ST's.
- Caso se trate de uma resposta final, esta é reencaminhada para o cliente.

Em cada passo onde o SR recebe uma resposta, seja final ou referência, extrai os campos RESPONSE VALUES, AUTHORITIES VALUES E EXTRA VALUES e adiciona cada linha à sua cache. Ao extrair o Registo que contém a informação da linha, atualiza o TTL do registo antes de o guardar na cache.

3 Modelo de Informação

Para este trabalho prático foram definidos ficheiros de configuração, de dados e de *log*, de forma a que o programa pudesse armazenar todos os dados relevantes de todos os componentes do sistema, bem como todas as atividades que estas realizam, segundo uma sintaxe predefinida, dependendo do tipo de ficheiro em causa. Assim, foram criados quatro tipos de ficheiros para a modelação da informação do sistema:

- Ficheiro de configuração dos servidores SP, SS e SR;
- Ficheiro com a lista de ST;
- Ficheiros de *log*;
- Ficheiro de dados do SP;

Todos os ficheiros seguem as seguintes regras de sintaxe (regras mais específicas para cada tipo de ficheiro serão enunciadas nos próximos subtópicos):

- As linhas começadas por '#' são **comentários**, pelo que são ignoradas;
- As linhas em branco também são ignoradas;

É importante realçar que, se for necessário atualizar o comportamento dos servidores com informação modificada nos ficheiros de configuração ou de dados que lhes dizem respeito, o programa simplesmente reinicia estes servidores.

3.1 Ficheiro de configuração dos servidores SP, SS e SR

Os *ficheiros de configuração* possuem a responsabilidade de conter toda a informação necessária para o funcionamento dos seus servidores, moldando assim o seu comportamento. Estes são apenas lidos e processados no arranque do programa.

Neste ficheiro, cada linha segue a seguinte sintaxe:

[parâmetro] [tipo do valor] [valor associado ao parâmetro]

Apresentam-se de seguida os vários tipos de dados aceites para este ficheiro, bem como o significado do campo "valor" quando se escreve o seu respetivo parâmetro (de notar que todas as referências a domínios, quer nos parâmetros quer nos valores, são considerado nomes completos):

- **DB** – o campo "valor" fornece a *diretoria* para aceder ao ficheiro de base de dados do domínio indicado no parâmetro;
- **SP** – o campo "valor" fornece o endereço IP[:porta] do SP do domínio indicado no parâmetro;
- **SS** – o valor indica o endereço IP[:porta] de um SS do domínio indicado no parâmetro. Com este tipo de valor, o SS passa a ter autorização para pedir a transmissão da informação da base de dados;
- **DD** – o valor indica o endereço IP[:porta] de um SR, SS ou SP do domínio indicado no parâmetro;
- **ST** – o campo "valor" fornece a *diretoria* para aceder ao ficheiro com a lista dos ST. Para este tipo de valor, o parâmetro é obrigatoriamente igual a '*root*';
- **LG** – o campo "valor" fornece a *diretoria* para aceder ao ficheiro de *log* que o servidor deve utilizar para registar a atividade do servidor associada ao domínio indicado no parâmetro. Para este tipo de valor, só podem ser indicados domínios para o qual o servidor é SP ou SS. Tem de sempre haver pelo menos uma entrada a referir a *diretoria* de um ficheiro de *log*, que servirá para ser usado registar todo o tipo de atividade que não esteja diretamente relacionado com domínios especificados noutras entradas LG (no campo "parâmetro" é escrito '*all*').

```
# Configuration file for primary server for .greens.dance.
.greens.dance. DB ../var/dns/greensdb.txt
.greens.dance. SS 10.0.9.10
.greens.dance. SS 10.0.11.10
.greens.dance. LG ../var/dns/greenslogsp.txt
root ST ../var/dns/rootservers.txt
```

Figura 4: Exemplo de ficheiro de configuração para o SP do domínio .greens.dance.

No caso de identificação de erros e/ou incoerências no ficheiro de configuração ou nas suas sintaxes que não entenda, o programa encerra imediatamente a sua execução.

3.2 Ficheiro com a lista de ST (*Servidores de Topo*)

```
10.0.17.10
10.0.18.10
```

Figura 5: Ficheiro com os respetivos IPs dos servidores de topo

Este ficheiro contém a lista de todos os ST que devem ser contactados sempre que um componente da topologia da rede quer obter informações sobre os domínios acima do domínio atual. Por cada linha do ficheiro, deve existir um endereço IP de um ST.

3.3 Ficheiros de *log*

Os ficheiros de *log* registam toda a atividade dos vários componentes existentes na topologia de rede (por exemplo: existe um ficheiro de *log* para cada um dos servidores da topologia). Para este tipo de ficheiros, existe sempre uma entrada de *log* por cada linha.

Aquando do arranque de um componente, este verifica se existem ficheiros de *log* indicados no seu ficheiro de configuração. Se existirem, as novas entradas de *logs* são registadas a partir da última entrada que existe no ficheiro. Caso contrário, os ficheiros de *logs* são imediatamente criados. Cada entrada possui a seguinte sintaxe:

[etiqueta temporal] [tipo de entrada] [endereço IP[:porta]] [dados da entrada]

A *etiqueta temporal* é a data e a hora completa do sistema operativo na altura em que aconteceu a atividade registada, e não a data e a hora em que foi registada. *Por exemplo: 19:10:2022.11:20:50:020* quer dizer que a atividade aconteceu no dia 19 de outubro de 2022, às 11 horas, 20 minutos, 50 segundos e 20 milésimas de segundo.

- **QR/QE** – indica que foi recebida/enviada uma *query* do/para o endereço IP indicado no respetivo campo. É necessário realçar que a sintaxe dos dados de entrada é a mesma que é usada no PDU de *query* no modo debug de comunicação entre os elementos.
- **RP/RR** – indica que foi enviada/recebida uma resposta a uma *query* para o/do endereço IP indicado no seu respetivo campo. É necessário realçar que a sintaxe dos dados de entrada é a mesma que é usada no PDU de resposta às *queries* no modo debug de comunicação entre os elementos;
- **ZT** – quer dizer que foi iniciado e concluído corretamente um processo de transferência de zona. O campo "endereço" indica o servidor do outro lado da transferência. Por outro lado, o campo "dados da entrada" indica o papel do servidor local na transferência (SP ou SS) e o tempo em milissegundos da respetiva transferência;

- **EV** – foi detetado um evento/atividade interna no componente; o endereço deve indicar 127.0.0.1 (ou localhost ou @); os dados da entrada devem incluir informação adicional sobre a atividade reportada (por exemplo, ficheiro de configuração/dados/ST lido, criado ficheiro de log, etc.);
- **ER** – foi recebido um PDU do endereço indicado que não foi possível decodificar corretamente; opcionalmente, os dados da entrada podem ser usados para indicar informação adicional (como, por exemplo, o que foi possível decodificar corretamente e em que parte/byte aconteceu o erro);
- **EZ** – detetado um erro no processo de transferência de zona, sendo que esta não foi concluída corretamente. O campo "endereço" mostra o servidor da outra ponta da transferência, enquanto que o campo "dados da entrada" indica qual o papel do servidor local na transferência (SP ou SS);
- **FL** – detetado um erro no funcionamento interno do componente; o endereço deve indicar 127.0.0.1; os dados da entrada devem incluir informação adicional sobre a situação de erro (por exemplo, um erro na decodificação ou incoerência dos parâmetros de algum ficheiro de configuração ou de base de dados);
- **TO** – foi detetado um timeout na interação com o servidor no endereço indicado; os dados da entrada devem especificar que tipo de timeout ocorreu (resposta a uma query ou tentativa de contato com um SP para saber informações sobre a versão da base de dados ou para iniciar uma transferência de zona);
- **SP** – a execução do componente foi parada; o endereço deve indicar 127.0.0.1; os dados da entrada devem incluir informação adicional sobre a razão da paragem se for possível obtê-la;
- **ST** – a execução do componente foi iniciada; o endereço deve indicar 127.0.0.1; os dados da entrada devem incluir informação sobre a porta de atendimento, sobre o valor do timeout usado (em milissegundos) e sobre o modo de funcionamento (modo "shy" ou modo debug).

3.4 Ficheiro de dados do SP (*Servidor Primário*)

Os *ficheiros de dados* contêm todos os dados que sustentam as bases de dados dos Servidores Primários, sendo que são consultados apenas no arranque e a sua informação é armazenada em memória. Para este ficheiro, cada linha segue a seguinte sintaxe (de notar que neste trabalho, não foi implementada a opção de prioridade):

[parâmetro] [tipo do valor] [valor] [tempo de validade - TTL]

Para todos os tipos de valores disponíveis, o tempo de validade (TTL) apresenta o tempo máximo (em segundos) que os dados podem existir numa dada cache de um servidor.

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

- **DEFAULT*** – define um nome (ou um conjunto de um ou mais símbolos) como uma macro que deve ser substituída pelo valor literal associado (não pode conter espaços nem o valor dum qualquer parâmetro DEFAULT); o parâmetro @ é reservado para identificar um prefixo por defeito que é acrescentado sempre que um nome não apareça na forma completa (i.e., terminado com '.'); o valor de TTL deve ser zero;
- **SOASP** – o campo "valor" indica o nome completo do SP do domínio (ou zona) indicado no parâmetro;
- **SOADMIN** – o campo "valor" fornece o endereço de e-mail completo do administrador do domínio indicado no parâmetro;
- **SOASERIAL** – o campo "valor" indica o número de série da base de dados do SP do domínio indicado no parâmetro. Sempre que a base de dados do SP é alterada este número é incrementado;
- **SOAREFRESH** – o campo "valor" apresenta o intervalo temporal (em segundos) para um SS perguntar ao SP do domínio indicado no parâmetro qual o número de série da sua base de dados;

- **SOARETRY** – o valor indica o intervalo temporal 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;
- **SOAEXPIRE** – o campo "valor" fornece o intervalo temporal (em segundos) que um SS pode considerar a réplica da base de dados da zona indicada no parâmetro como válida. Caso seja atingido o intervalo de tempo fornecido, o SS deve, então, deixar de responder a perguntas sobre a zona em causa, mesmo que continue a tentar contactar o respetivo SP;
- **NS** – o campo "valor" apresenta o nome de um servidor que é autoritativo para o domínio indicado no parâmetro, ou seja, o nome do SP ou de um dos SS do domínio. Assim, o tipo de valor NS é usado para identificar os *NameServers*
- **A** – fornece o endereço IPv4 do *host*/servidor indicado no parâmetro como nome;
- **CNAME** – o campo "valor" indica um nome canónico associado ao nome indicado no parâmetro;
- **MX** – o campo "valor" apresenta o nome do servidor de e-mail para o domínio indicado no campo 'parâmetro';

```
# DNS database file for domain .greens.dance.
# It also includes a pointer to the primary server
# of the alicent.greens.dance. subdomain

@ DEFAULT greens.dance.
TTL DEFAULT 86400

@ SOASP ns1.greens.dance. TTL
@ SOAADMIN dns\admin.greens.dance. TTL
@ SOASERIAL 0117102022 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS ns1.greens.dance. TTL
@ NS ns2.greens.dance. TTL
@ NS ns3.greens.dance. TTL

alicent.@ NS sp.alicent.greens.dance. TTL

@ MX mx1.greens.dance. TTL
@ MX mx2.greens.dance. TTL

ns1 A 10.0.0.10 TTL
ns2 A 10.0.9.10 TTL
ns3 A 10.0.11.10 TTL
sp.alicent A 10.0.13.10 TTL
mx1 A 10.0.0.12 TTL
mx2 A 10.0.0.13 TTL
www A 10.0.0.11 TTL

sp CNAME ns1 TTL
ss1 CNAME ns2 TTL
ss2 CNAME ns3 TTL
mail1 CNAME mx1 TTL
mail2 CNAME mx2 TTL
web CNAME www TTL
```

Figura 6: Exemplo de ficheiro de base de dados para o domínio .greens.dance.

3.5 PDU: Formato das mensagens de DNS

As mensagens de DNS estão separadas entre dois grandes campos, o *HEADER* e o *DATA*.

Na mensagem da query não é usado todos os campos, pois são campos que vão ser usados após a comunicação da query ou durante a resposta à query.

HEADER Fields:

- MESSAGE ID - Usado para identificar a mensagem, ao fim de relacionar as respostas recebidas com a query original;
- FLAGS - Existem 3 flags distintas (Q,R,A). Se a flag Q estiver ativa indica que a mensagem é uma query, caso contrário quer dizer que é uma resposta a uma query. Se a flag R estiver ativa então é desejado que o processo opere de forma recursiva. Se a flag R estiver ativa na resposta a uma query significa que o servidor que respondeu suporta o modo recursivo. Se a flag A estiver ativa na resposta a uma query indica que a resposta é autoritativa;
- RESPONSE CODE – Indica o código de erro na resposta a uma query. Se o valor for '0' quer dizer que não houve erro. 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, este caso é identificado como resposta negativa e poderá ser guardado em cache. Se o valor for '2' o domínio incluído em NAME não existe, e também é identificado como resposta negativa podendo ser guardado em cache. Se o valor for '3' o TYPE OF VALUE é inválido;
- NUMBER OF VALUES - Número de entradas, do servidor autoritativo, que têm um parâmetro igual a NAME e um tipo de valor igual a TYPE OF VALUE ou que façam parte da lista de entradas do campo RESPONSE VALUES;
- NUMBER OF AUTHORITIES - Número de entradas que identificam os servidores autoritativos para o domínio incluído no RESULT VALUES;
- NUMBER OF EXTRA VALUES - Número de entradas (num máximo de 255) com informação adicional relacionada com os resultados da query ou com os servidores da lista de autoridades

DATA Fields:

- QUERY INFO - Informação sobre os campos NAME, que indica o nome do domínio, e TYPE OF VALUE, que indica o tipo de valor do parâmetro. Na resposta a uma query, os servidores devem copiar a informação do QUERY INFO e incluí-la na mensagem de resposta;
- RESPONSE VALUES - lista das entradas que correspondem aos campos NAME e TYPE OF VALUE incluídos na cache ou na base de dados do servidor autoritativo;
- AUTHORITIES VALUES - Número de entradas (num máximo de 255) que identificam os servidores autoritativos para o domínio incluído no RESULT VALUES;
- EXTRA VALUES - lista das entradas com a flag A ativa e que coincidem com todos os valores no campo RESPONSE VALUES e no campo AUTHORITIES VALUES.

3.6 Mecanismo de Codificação Binária

Tal como referido no ponto 3, para que haja a comunicação entre servidores e entre cliente-servidor, é necessário enviar mensagens DNS com um formato específico, nomeadamente, segundo um formato Binário. O formato **Binário** de mensagens permite diminuir o espaço em memória ocupada para armazenar todos os dados, aumentando a rapidez com que estas mensagens são transmitidas.

No entanto, para a 1ª fase do projeto, o Cliente e os Servidores executarão, por defeito, em modo *debug*, onde, as mensagens de DNS são transmitidas no formato de sequência de caracteres conciso. Assim, o modo normal com o formato Binário será apenas apenas implementado na segunda fase do projeto.

Com o mecanismo de codificação binária, em vez de a query da mensagem DNS ser transmitida no formato de caracteres conciso, cada caracter ASCII da string é convertido no seu respetivo valor binário.

Só quando a mensagem (em formato Binário) chega ao servidor/cliente destino, é que esta é reconvertida na sua string original permitindo a sua análise.

4 Modelo de Comunicação

Antes da realização de qualquer módulo ou componentes também é preciso identificar todas as interações possíveis entre os elementos do sistema, incluindo o protocolo para transferência de zona.

4.1 Representação lógica da Mensagem de DNS

Todas as interações assíncronas (não orientadas à conexão) possíveis neste sistema são feitas através de mensagens DNS encapsuladas no protocolo UDP. Todas as mensagens DNS devem ser implementadas usando a sintaxe da mesma unidade de dados protocolar (PDU), já explicitado anteriormente. Uma mensagem DNS deve ter um cabeçalho de tamanho fixo e uma parte de dados que deve ocupar até 1 KByte. A parte de dados contém sempre quatro partes distintas: i) os dados duma query original; ii) os resultados diretos a essa query original; iii) informação sobre os servidores que têm informação autoritativa sobre os dados da resposta e iv) informação adicional indiretamente ligada aos resultados ou aos dados da query original.

4.2 Envio e Receção de queries

Segue-se um exemplo duma interação assíncrona entre um CL e o SP, que é autoridade do domínio *.blacks.*. CL quer saber qual é o nome canónico do servidor *ns1.blacks.* e envia então uma query para o servidor SP, recebendo uma query em resposta, representado na figura 7:

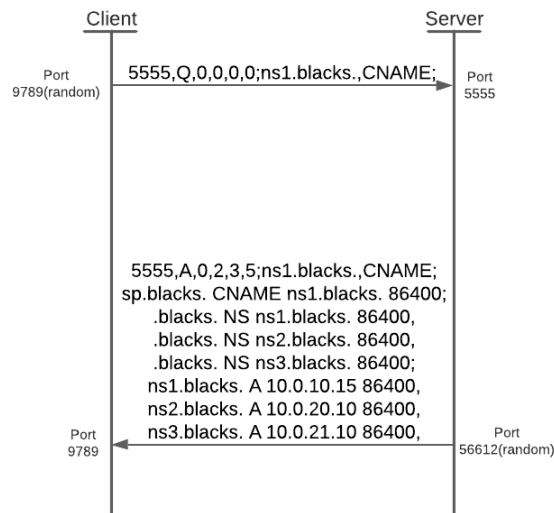


Figura 7: Envio/Resposta de uma query entre um SP e um CL

Nota: Por questões de eficiência optamos pelo formato mais conciso das queries.

Uma querie ao ser enviada deverá ter sempre a flag 'Q' ativa no campo FLAGS, por se tratar de um envio de uma querie, e os campos de QUERY INFO preenchido em relação ao que pretende receber, na figura 7 o campo NAME está como "ns1.blacks." e o campo TYPE OF VALUE como "CNAME". Os campos RESPONSE VALUES, AUTHORITIES VALUES e EXTRA VALUES não devem ser preenchidos e os restantes devem ter o valor 0.

Na resposta à query a flag 'Q' não pode estar ativa no campo FLAGS , na figura 7 a flag 'A' está ativa pois trata-se de uma resposta autoritativa. O campo MESSAGE ID deve estar preenchido igualmente como está no envio da query, e os restantes campos devem ser preenchidos conforme foram apresentados anteriormente e tendo em conta o que foi pedido na query enviada. Em caso de erro o campo RESPONSE CODE deverá ser preenchido tendo em conta o que causou o erro.

4.2.1 Exemplos usando um SR

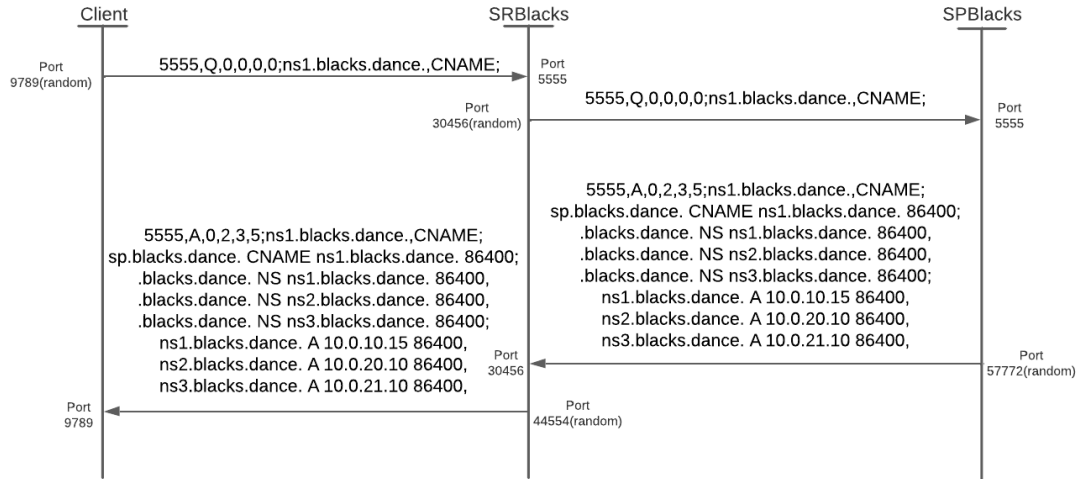


Figura 8: Envio/Resposta de uma query entre um SR e um CL através de DD

O Cliente envia uma query para o SR do domínio .blacks.dance. Como esta query é referente ao próprio domínio por defeito DD deste SR, ele reencaminha a query diretamente ao SP do .blacks.dance., que o responde e esta resposta é então reencaminhada de volta ao Cliente.

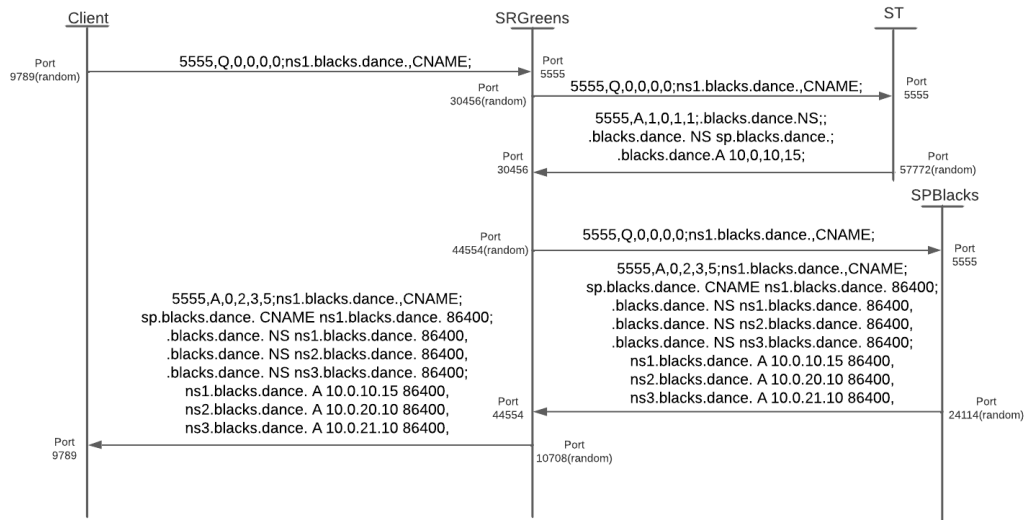


Figura 9: Envio/Resposta de uma query entre um SP e um CL através do ST

O Cliente envia uma query referente ao domínio .blacks.dance. ao SR localizado no domínio .greens.dance. Este não possui ainda informação na cache e, como não se trata do seu domínio por defeito DD, reencaminha então a query para o ST .dance. Este portanto através do longest prefix match responde com a referência para o SP do .blacks.dance. O SR então recebe esta referência, extrai o IP e reencaminha para este IP a query. O SP então responde com a resposta final e o SR a reencaminha para o Cliente.

4.3 Tipos de funcionamento

No modo normal de funcionamento dos componentes as mensagens DNS têm que ser codificadas em binário numa forma eficiente, já explicitado anteriormente, para uma mensagem ocupar o menor espaço possível.

Em modo debug as mensagens de DNS são transmitidas no formato de sequência de caracteres conciso, tal como mostrada na figura1(MUDAR O NOME DA FIGURA) não precisando de ser codificadas.

4.4 Transferência de Zona

A Transferência de Zona é uma conexão TCP de um SS com um SP com o objetivo de manter atualizada a base de dados dos SS.

Inicialmente o SS, através de queries normais, descobre se a sua versão da base de dados é a mesma que a do SP. Se não for dá-se início à Transferência de Zona, onde o SS envia o nome do domínio que quer fazer cópia da base de dados para o respetivo SP. O SP valida o domínio e a autorização que o SS tem para receber a cópia da base de dados, e se for válido então envia o número de entradas do ficheiro de base de dados para o SS. O SS aceita receber esse número de entradas, respondendo ao SP com o mesmo valor, e então o SP manda as entradas em formato de texto numerando-as por ordem crescente. De seguida o SS vai verificando se recebeu todas as entradas esperadas até um tempo predefinido se esgotar, escolhido no tipo de valor do ficheiro log SOAREFRESH. Ao fim desse tempo, se o SS não tiver todas as entradas esperadas, ele termina a conexão TCP e desiste da Transferência de Zona, tentando outra vez após um intervalo de tempo, escolhido no tipo de valor do ficheiro log SOAENTRY.

Por questões de segurança foi decidido que a informação replicada do SP é armazenada apenas em memória volátil no SS.

4.5 Implementação da Transferência de Zona

- Inicialmente o SOASERIAL do SS é -1 e só depois de realizar transferência de zona é que fica com o da base de dados
- O SS começa por pedir o SOASERIAL do SP e se for igual ao seu simplesmente muda-se a variável valid dos registos para true, caso contrário o SS manda o seu dominio
- O SP tem timeout no socket, se não chegar a receber mais nada do SS assume que é para o fechar
- De seguida o SP verifica se o domínio é o mesmo que o dele e que o IP do SS corresponde a um IP dos SS ao qual ele conhece
- Se o SP validar manda o número de linhas da sua base de dados
- O SS aceita respondendo com o mesmo número de linhas
- O SP manda a sua base de dados linha a linha e o SS acaba por copiar a base de dados para si mesmo

Todas estes pedidos/respostas entre o SS e o SP têm timeout no socket, passado por argumento. Como a transferência de zona tem que ocorrer dentro do timeout, o timeout do socket não iria funcionar para esta situação, pois para cada linha iria-se resetar o timeout e é suposto o timeout ser o tempo em que demoramos a mandar todas as linhas e não o tempo que demoramos a mandar uma linha. Por isso nós usamos um contador que compara o tempo atual com o tempo que o começou o timeout. A cada linha que o SS recebem corretamente, ele manda de volta o número da linha, se der timeout, ele interrompe e manda uma mensagem de timeout para o SP parar e fechar.

Quando o SOASERIAL é igual ou quando é terminada a transferência de zona damos sleep ao SOARE-FRESH, e quando ocorre um timeout também damos sleep ao SOARETRY.

O SOARETRY e o SOAREFRESH, tal como o timeout, são passados por argumento enquanto não estiver feita a transferência.

Quando o SS recebe uma query de um cliente, ele verifica se já atingiu o SOAEXPIRE, não precisando, assim, de ter uma thread em background a correr para verificar se expirou. O SS tem um counter que marca o tempo em que recebeu os registos da transferência de zona e também é atualizado quando o SOASERIAL for igual na transferência de zona, e se atingir o SOAEXPIRE ele marca tudo como não válido e não prossegue com a query, e como o Cliente tem timeout se não receber nesse período de tempo vai fechar.

4.6 Exemplo de transferência de zona

Segue-se um exemplo duma transferência de zona entre o SP e um SS pertencentes ao domínio *.blacks.*, representado pela seguinte Figura 10:

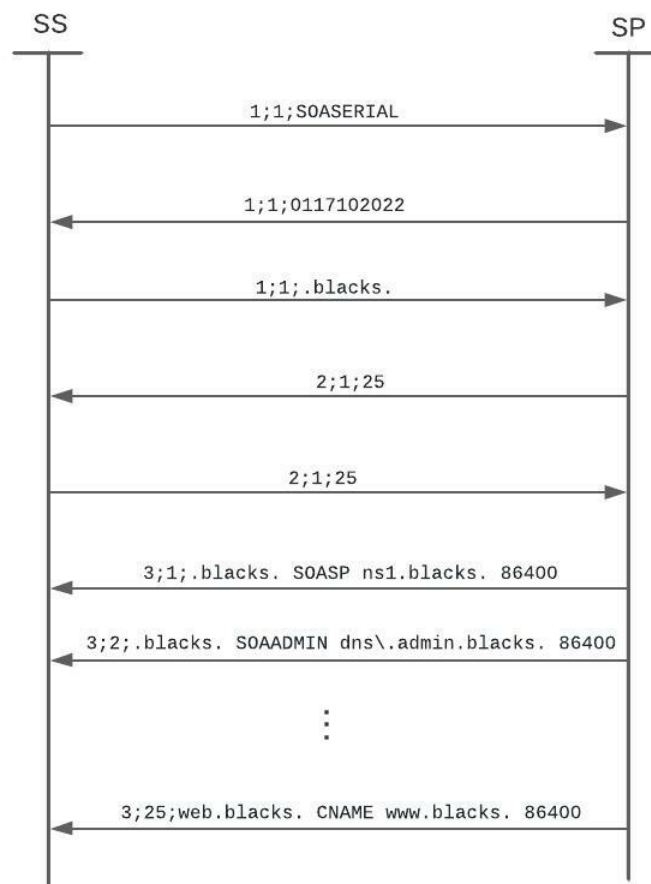


Figura 10: Exemplo de transferência de zona dentro do domínio *.blacks.*

No exemplo da Figura 10 o SS começa por mandar uma query SOASERIAL para o SP onde o SP responde com a sua versão da base de dados. Depois o SS verifica se a sua versão da base de dados (em que inicialmente é sempre -1) é diferente da recebida e, como é diferente, o SS faz o pedido de transferência de zona para o domínio *.blacks.* mandando o mesmo. De seguida, o SP verifica que o domínio dado é o mesmo dele e que o ip do SS corresponde a um dos ips dos SS que ele conhece e manda o número de entradas da sua base de dados, neste caso 25. O SS aceita receber 25 entradas respondendo com o mesmo número de entradas. Por fim o SP acaba por mandar as entradas da sua base de dados linha a linha numerando-as por ordem crescente seguida de um ";", tal como representado na Figura.

Também decidimos dar um tipo a cada envio/resposta, começando sempre pelo número do tipo seguido de um ";", estando divididos da seguinte forma:

- Envio/Resposta da query SOASERIAL e envio do nome do domínio - tipo 1
- Envio/Resposta do número de linhas da base de dados - tipo 2
- Receção das entradas da base de dados - tipo 3

4.7 Comportamento dos elementos em situação de erro

Situações de erro no envio e receção de queries:

- O domínio não existe: neste caso o servidor, na resposta à querie, deve pôr o valor '2' no campo RESPONSE CODE, e o campos RESPONSE VALUES deve ser vazio e os campos AUTHORITIES VALUES e EXTRA VALUES devem conter a respetiva informação contida no servidor.
- O formato do PDU está errado : neste caso o servidor, na resposta à querie, deve pôr o valor '3' no campo RESPONSE CODE e os campos RESPONSE VALUES, AUTHORITIES VALUES e EXTRA VALUES devem estar vazios.

Existe erro na Transferência de Zona quando o domínio do Servidor Secundário não corresponde ao domínio do Servidor Primário ou quando o IP do Servidor Secundário não é reconhecido pelo Primário.

4.8 Ameaças de segurança

Existem várias maneiras de direcionar e explorar servidores de DNS, e estes são dos ataques mais comuns:

- **Falsificação de DNS:** trata-se de um ataque no qual dados de DNS falsificados são introduzidos na cache de um resolvidor de DNS e, como resultado, o resolvidor retorna um endereço IP incorreto para um domínio e, conseqüentemente, invés de ir para o site correto, o tráfego pode ser desviado para uma máquina mal-intencionada ou para qualquer outro lugar que o invasor desejar;
- **Tunelamento de DNS:** este ataque usa outros protocolos para criar um túnel que atravessa as consultas e respostas de DNS. Os invasores podem usar SSH, TCP ou HTTP para transmitir malware(qualquer software intencionalmente feito para causar danos a um computador) ou informações roubadas nas consultas de DNS;
- **Sequestro de DNS:** no sequestro de DNS, o invasor redireciona as consultas para um nameserver de domínio diferente. Isso pode ser feito com malware ou por meio da modificação não autorizada de um servidor de DNS. Embora o resultado seja semelhante ao da falsificação de DNS, trata-se de um ataque fundamentalmente diferente, pois visa o registro de DNS do site no nameserver e não no cache de um resolvidor.

5 Ambiente de Teste

deve conter uma análise de testes do sistema que permita verificar o funcionamento de todos os seus elementos num ambiente de teste pré-definido conforme especificação a encontrar neste enunciado. Esta secção deve incluir uma explicação do funcionamento dos componentes do sistema baseado nos exemplos executados no ambiente de teste. Os resultados detalhados das experiências executadas no ambiente de teste, assim como a especificação exhaustiva do próprio ambiente de teste, devem ser incluídos num anexo próprio.

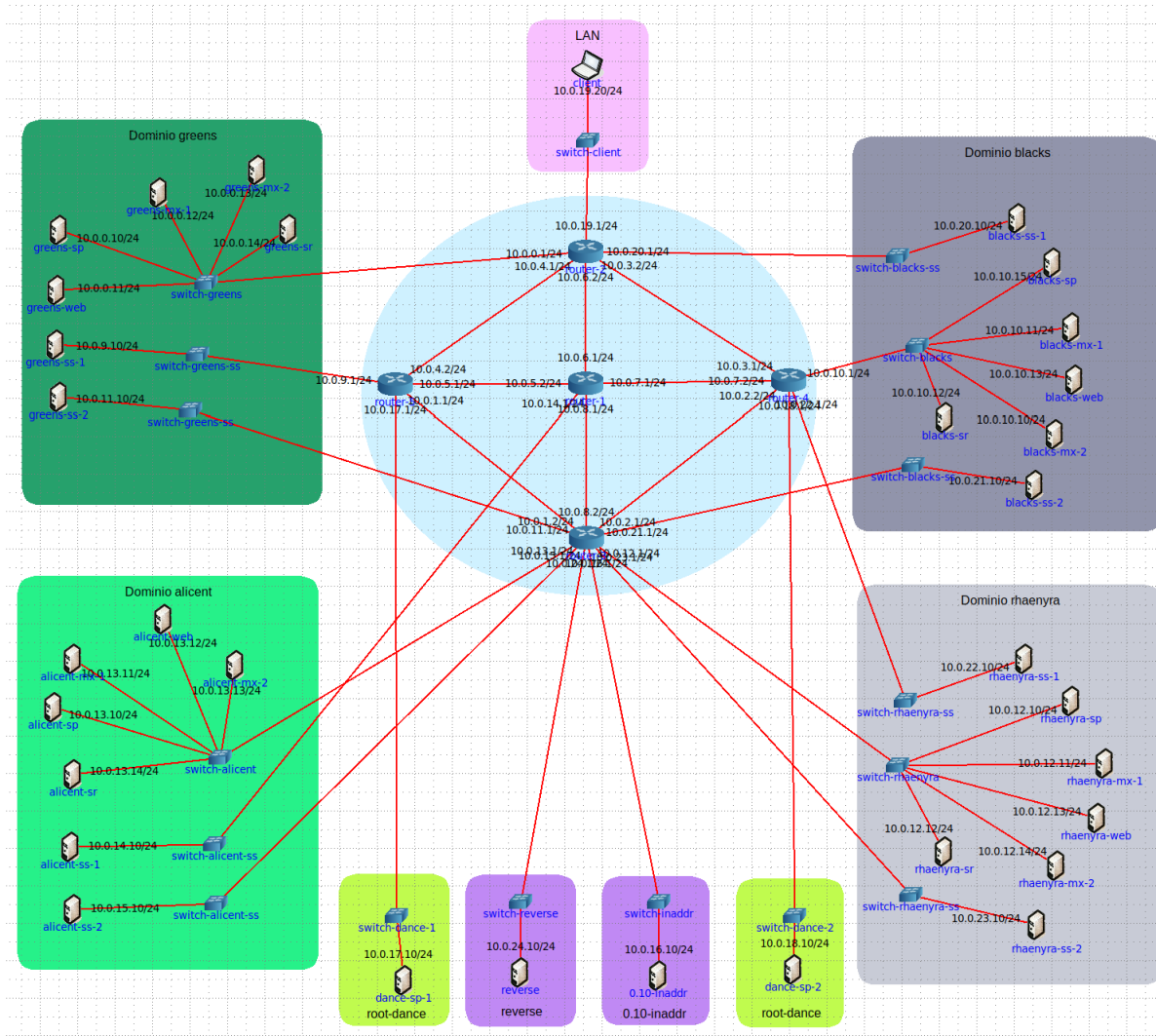


Figura 11: Topologia do Ambiente de Teste

A figura 11 mostra a arquitetura escolhida para o ambiente de teste do projeto. Há dois ST, 3 SDT: **.greens.**, **.blacks.** e **.reverse.** e 3 subdomínios: **.alitent**, **.rhaenyra** e **0.10-inaddr**. Cada um destes, com exceção dos referentes ao DNS reverso, possui um SP e dois SS. Foram analisadas duas situações que poderiam levar a não ser possível utilizar o SP: este Servidor Primário pode falhar por falha no servidor ou por uma falha na rede. Portanto, concluiu-se que a melhor decisão seria ter os SS alocados a redes diferentes do SP e umas das outras, de forma a sempre garantir o funcionamento do serviço.

Para melhor visualização e compreensão da hierarquia, decidiu-se nomear os servidores root por **.dance..** Estes possuem apenas um SP, visto que a redundância já é estabelecida através da existência de dois SP para o mesmo domínio.

Cada domínio é composto por um SDT representado pelo conjunto de um SP e dois SS, um servidor *web* (*www*), dois servidores de *mail* (*mx*), um servidor de resolução e um subdomínio. Cada subdomínio é composto por um SP e dois SS, um servidor *web* (*www*), dois servidores de *mail* (*mx*) e um servidor de

resolução.

Com relação aos Servidores de Topo, optou-se pela implementação de dois ST, cada um com o seu SP. Desta forma, é possível garantir que estes servidores sejam muito confiáveis.

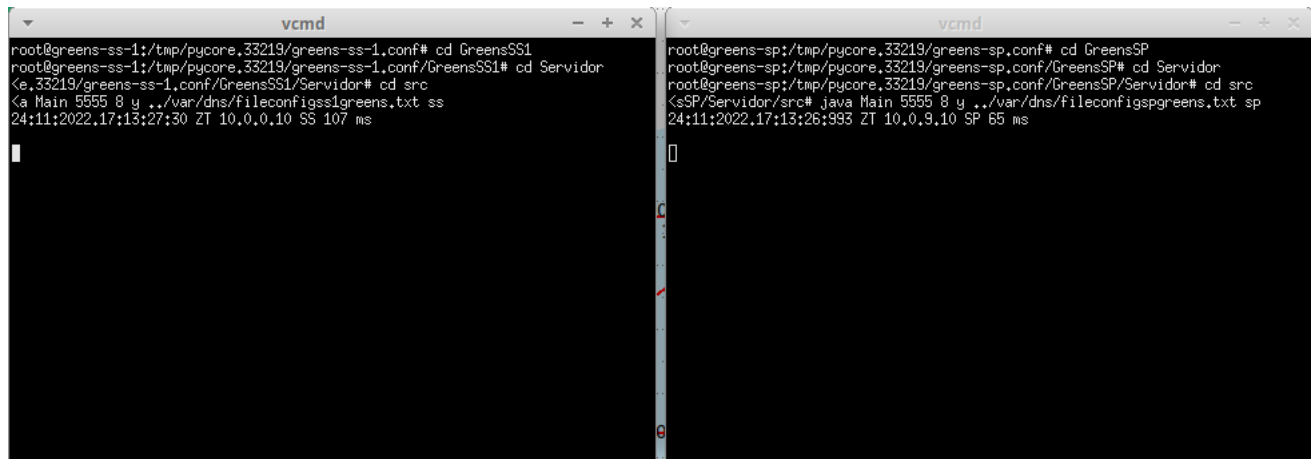
Cada rede é composta por um switch no qual os hosts encontram-se conectados. Cada switch está conectado a um router.

É necessário abordar os requisitos funcionais para cada componente mencionado de forma a garantir sua execução de maneira coerente, robusta e fiável, e também proporcionar uma melhor compreensão da arquitetura do sistema desenvolvido.

É importante enfatizar que as separações por cores não refletem uma separação geográfica, visto que, por exemplo, os SS encontram-se em redes diferentes dos SP. Portanto, o esquema de figuras e cores foi apenas utilizado de forma a ilustrar a posse do domínio sobre os seus elementos de forma a contribuir para uma percepção mais fácil da estrutura seguida.

5.1 Transferência de Zona

5.1.1 Procedimento de transferência de zona



```
vcmd
root@greens-ss-1:/tmp/pycore.33219/greens-ss-1.conf# cd GreensSS1
root@greens-ss-1:/tmp/pycore.33219/greens-ss-1.conf/GreensSS1# cd Servidor
root@greens-ss-1:/tmp/pycore.33219/greens-ss-1.conf/GreensSS1/Servidor# cd src
root@greens-ss-1:/tmp/pycore.33219/greens-ss-1.conf/GreensSS1/Servidor/src# java Main 5555 8 y ../var/dns/fileconfigsslgreens.txt ss
24:11:2022,17:13:27:30 ZT 10,0,0,10 SS 107 ms

vcmd
root@greens-sp:/tmp/pycore.33219/greens-sp.conf# cd GreensSP
root@greens-sp:/tmp/pycore.33219/greens-sp.conf/GreensSP# cd Servidor
root@greens-sp:/tmp/pycore.33219/greens-sp.conf/GreensSP/Servidor# cd src
root@greens-sp:/tmp/pycore.33219/greens-sp.conf/GreensSP/Servidor/src# java Main 5555 8 y ../var/dns/fileconfigspgreens.txt sp
24:11:2022,17:13:26:993 ZT 10,0,9,10 SP 65 ms
```

Figura 12: Transferência de zona entre SP e SS1 do domínio greens

A Figura 12 apresenta o procedimento de transferência de zona no domínio .greens. entre o Servidor Primário e o Servidor Secundário. Primeiramente, o SP é inicializado e fica a aguardar por um pedido de comunicação através do socket TCP. Quando o SS é inicializado, este começa por solicitar a comunicação com o SP e, quando estabelecida, requisita a transferência de zona.

É possível confirmar que este processo foi finalizado com sucesso de acordo com o log de etiqueta ZT impresso no terminal de ambos. Este log também indica o tempo em milissegundos desde o início até o fim da transferência dos dados.

5.1.2 Teste de queries para o SS após a transferência de zona

A fim de confirmar que a transferência de dados foi de facto bem sucedida, duas queries foram realizadas entre o Cliente e o Servidor Secundário. De acordo com a Figura 13, é possível concluir que a transferência de zona foi efetuada com sucesso visto que o Servidor Secundário foi capaz de responder corretamente às queries recebidas.

The image shows two terminal windows side-by-side, both titled 'vcmd'. The left window shows a series of DNS queries and responses from the server side. The right window shows the corresponding queries and responses from the client side. The queries include MX, NS, and A records for the domain .greens.

```

root@greens-ss-1:/tmp/pycore.33219/greens-ss-1.conf# cd GreensSS1
root@greens-ss-1:/tmp/pycore.33219/greens-ss-1.conf/GreensSS1# cd Servidor
<e.33219/greens-ss-1.conf/GreensSS1/Servidor# cd src
<a Main 5555 8 y ../var/dns/fileconfigss1greens.txt ss
24:11:2022.17:13:27:30 ZT 10,0,0,10 SS 107 ms

24:11:2022.17:15:52:112 QR 10,0,19,20 3114,Q,0,0,0,0;.greens.,MX;;;

24:11:2022.17:15:52:113 RP 10,0,19,20 3114,A,0,2,3,5;.greens.,MX;.greens. MX mx1
.greens. 86400 1000000;.greens. MX mx2.greens. 86400 1000000;.greens. NS ns1.gre
ens. 86400 1000000;.greens. NS ns2.greens. 86400 1000000;.greens. NS ns3.greens.
86400 1000000;.greens. NS ns1.greens. A 10,0,0,10 86400 1000000;.greens. NS ns2.greens. A 10,0,9,10 864
00 1000000;.greens. NS ns3.greens. A 10,0,11,10 86400 1000000;.greens. NS ns1.greens. A 10,0,0,12 86400
1000000;.greens. NS ns2.greens. A 10,0,0,13 86400 1000000;

24:11:2022.17:17:35:983 QR 10,0,19,20 5065,Q,0,0,0,0;.greens.,A;;;

24:11:2022.17:17:35:984 RP 10,0,19,20 5065,A,0,1,3,2;.greens.,A;.greens. A
10,0,0,10 86400 1000000;.greens. NS ns1.greens. 86400 1000000;.greens. NS ns2.g
reens. 86400 1000000;.greens. NS ns3.greens. 86400 1000000;.greens. NS ns1.greens. A 10,0,0,10 86400
1000000;.greens. NS ns2.greens. A 10,0,9,10 86400 1000000;.greens. NS ns3.greens. A 10,0,11,10 86400
1000000;

root@greens-ss-1:/tmp/pycore.33219/greens-ss-1.conf/GreensSS1/Servidor#

```

```

root@client:/tmp/pycore.33219/client.conf# cd Cliente
root@client:/tmp/pycore.33219/client.conf/Cliente# cd Cliente
<19/client.conf/Cliente/Cliente# java Cliente dnsc1 10,0,9,10 .greens. MX
3114,Q,0,0,0,0;.greens.,MX;;;
3114,A,0,2,3,5;.greens.,MX;
.greens. MX mx1.greens. 86400 1000000;.greens. MX mx2.greens. 86400 1000000;
.greens. NS ns1.greens. 86400 1000000;.greens. NS ns2.greens. 86400 1000000;.gre
ens. NS ns3.greens. 86400 1000000;
ns1.greens. A 10,0,0,10 86400 1000000;.greens. NS ns2.greens. A 10,0,9,10 86400 1000000;.gre
ens. NS ns3.greens. A 10,0,11,10 86400 1000000;.greens. NS ns1.greens. A 10,0,0,12 86400 1000000;.gre
ens. NS ns2.greens. A 10,0,0,13 86400 1000000;

<19/client.conf/Cliente/Cliente# java Cliente dnsc1 10,0,9,10 ns1.greens. A
5065,Q,0,0,0,0;.greens.,A;;;
5065,A,0,1,3,2;.greens.,A;
ns1.greens. A 10,0,0,10 86400 1000000;
.greens. NS ns1.greens. 86400 1000000;.greens. NS ns2.greens. 86400 1000000;.gre
ens. NS ns3.greens. 86400 1000000;
ns2.greens. A 10,0,9,10 86400 1000000;.greens. NS ns3.greens. A 10,0,11,10 86400 1000000;

root@client:/tmp/pycore.33219/client.conf/Cliente/Cliente#

```

Figura 13: Testes no SS1

5.2 Queries

De forma a testar se a aplicação é capaz de devolver as respostas corretas às queries, foram efetuados 8 testes diferentes, um para cada tipo de valor possível para a query. Destes 8, 3 foram para testar as respostas ao encontrar erros.

5.2.1 MX

Nesta query foi solicitada a informação de todos os servidores de mail do domínio .greens. É possível observar na Figura 14 que o servidor foi capaz de responder corretamente e que os logs foram impressos com o formato esperado.

The image shows two terminal windows side-by-side, both titled 'vcmd'. The left window shows a series of DNS queries and responses from the server side. The right window shows the corresponding queries and responses from the client side. The queries include MX, NS, and A records for the domain .greens.

```

root@greens-sp:/tmp/pycore.33219/greens-sp.conf# cd GreensSP
root@greens-sp:/tmp/pycore.33219/greens-sp.conf/GreensSP# cd Servidor
root@greens-sp:/tmp/pycore.33219/greens-sp.conf/GreensSP/Servidor# cd src
<sSP/Servidor/src# java Main 5555 8 y ../var/dns/fileconfigspgreens.txt sp
24:11:2022.17:13:26:993 ZT 10,0,9,10 SP 65 ms

24:11:2022.17:20:20:628 QR 10,0,19,20 23988,Q,0,0,0,0;.greens.,MX;;;

24:11:2022.17:20:20:628 RP 10,0,19,20 23988,A,0,2,3,5;.greens.,MX;.greens. MX mx1
.greens. 86400 1000000;.greens. MX mx2.greens. 86400 1000000;.greens. NS ns1.gre
ens. 86400 1000000;.greens. NS ns2.greens. 86400 1000000;.greens. NS ns3.greens.
86400 1000000;.greens. NS ns1.greens. A 10,0,0,10 86400 1000000;.greens. NS ns2.greens. A 10,0,9,10 864
00 1000000;.greens. NS ns3.greens. A 10,0,11,10 86400 1000000;.greens. NS ns1.greens. A 10,0,0,12 86400
1000000;.greens. NS ns2.greens. A 10,0,0,13 86400 1000000;

root@greens-sp:/tmp/pycore.33219/greens-sp.conf/GreensSP/Servidor#

```

```

root@client:/tmp/pycore.33219/client.conf# cd Cliente
root@client:/tmp/pycore.33219/client.conf/Cliente# cd Cliente
<19/client.conf/Cliente/Cliente# java Cliente dnsc1 10,0,9,10 .greens. MX
23988,Q,0,0,0,0;.greens.,MX;;;
23988,A,0,2,3,5;.greens.,MX;
.greens. MX mx1.greens. 86400 1000000;.greens. MX mx2.greens. 86400 1000000;
.greens. NS ns1.greens. 86400 1000000;.greens. NS ns2.greens. 86400 1000000;.gre
ens. NS ns3.greens. 86400 1000000;
ns1.greens. A 10,0,0,10 86400 1000000;.greens. NS ns2.greens. A 10,0,9,10 86400 1000000;.gre
ens. NS ns3.greens. A 10,0,11,10 86400 1000000;.greens. NS ns1.greens. A 10,0,0,12 86400 1000000;.gre
ens. NS ns2.greens. A 10,0,0,13 86400 1000000;

root@client:/tmp/pycore.33219/client.conf/Cliente/Cliente#

```

Figura 14: Query MX

5.2.2 NS

Nesta query foi solicitada a informação de todos os servidores autoritativos do domínio .greens. É possível observar na Figura 15 que o servidor foi capaz de responder corretamente e que os logs foram impressos com o formato esperado.

[illegible]

Figura 15: Query NS

5.2.3 Subdomínio

Nesta query foi solicitada a informação do endereço IP de um servidor mail mx1 do subdomínio alicent.greens. É possível observar na Figura 16 que o servidor foi capaz de responder corretamente. Neste caso, enviou a informação que continha sobre o servidor primário autoritativo do subdomínio alicent.greens. Os logs também foram impressos com o formato esperado.

```
24:11:2022.17:24:21:456 QR 10.0.19.20 61869,0.0.0.0;mx1.alicent.greens.,A;::: <ent# java Cliente dnsl 10.0.0.10 mx1.alicent.greens. A
61869,0.0.0.0;mx1.alicent.greens.,A;:::
24:11:2022.17:24:21:456 RP 10.0.19.20 61869,,0.1.0.1;mx1.alicent.greens.,A;alice
nt.greens. NS sp.alicent.greens. 86400 1000000;sp.alicent.greens. A 10.0.13.10
86400 1000000;
sp.alicent.greens. A 10.0.13.10 86400 1000000;
root@client:/tmp/bucone.33219/client_conf/Cliente/Cliente#
```

Figura 16: Teste para subdomínio como NAME

5.2.4 A

Nesta query foi solicitada a informação do endereço IP do servidor web do domínio .greens. É possível observar na Figura 17 que o servidor foi capaz de responder corretamente e que os logs foram impressos com o formato esperado.

```
24:11:2022.17:25:40:32 QR 10.0.19.20 17541,Q.0.0.0.0/www.greens.,A;:::
24:11:2022.17:25:40:36 RP 10.0.19.20 17541,A.0.1.3.3/www.greens.,A/www.greens.,A
10.0.0.11 86400 1000000;.greens.NS ns1.greens. 86400 1000000;.greens.NS ns2.g
reens. 86400 1000000;.greens.NS ns3.greens. 86400 1000000;ns1.greens.A 10.0.10.
10 86400 1000000;ns2.greens.A 10.0.9.10 86400 1000000;ns3.greens.A 10.0.11.10
86400 1000000;
root@client:/tmp/pucore.33219/client.conf/Cliente/Cliente#
```

Figura 17: Query A

5.2.5 CNAME

Nesta query foi solicitada a informação do nome canónico estabelecido para o servidor web do domínio .greens. É possível observar na Figura 18 que o servidor foi capaz de responder corretamente e que os logs foram impressos com o formato esperado.

```
24:11:2022,17:27:10:968 QR 10.0.19.20 53709,0.0.0.0:www.greens.,CNAME;;; <Cliente# java Cliente dnsc1 10.0.0.10 www.greens. CNAME
53709,0.0.0.0:www.greens.,CNAME;;;
53709,A,0.1.3.4:www.greens.,CNAME;
web.greens. CNAME www.greens. 86400 1000000;
ns. CNAME www.greens. 86400 1000000;.greens. NS ns1.greens. 86400 1000000;.gre
s. NS ns2.greens. 86400 1000000;.greens. NS ns3.greens. 86400 1000000;ns1.greens
. A 10.0.0.10 86400 1000000,ns2.greens. A 10.0.9.10 86400 1000000,ns3.greens. A
10.0.11.10 86400 1000000,www.greens. A 10.0.0.11 86400 1000000;
root@client:/tmp/pycore.33219/client.conf/Cliente/Cliente#
```

Figura 18: Query CNAME

5.2.6 Erros

Response code 2

Nesta query, foi passado um domínio desconhecido ao domínio .greens. Desta forma o servidor deve responder com o response code igual a 2, sem valor de resposta e sem fornecer informação de seus servidores autoritativos visto que a query supostamente não se destinava a seu domínio. Na Figura 19 é possível visualizar que esta foi a resposta do servidor e que os logs foram impressos com o formato esperado.

```
24:11:2022,17:30:16:133 QR 10.0.19.20 36484,0.0.0.0:greens.,NS;;; <Cliente# java Cliente dnsc1 10.0.0.10 greens. NS
36484,0.0.0.0:greens.,NS;;;
36484,A,2.0.0.0:greens.,NS;
root@client:/tmp/pycore.33219/client.conf/Cliente/Cliente#
```

Figura 19: Teste Erro 2

Response code 3 Nesta query, foi passado um type of value inválido dentre os permitidos. Desta forma o servidor deve responder com o response code igual a 3, sem valor de resposta e sem fornecer informação de seus servidores autoritativos visto que, neste caso, há um erro de formatação da query, consequentemente, tornando o PDU inválido e impossibilitando respostas contendo informação retirada do servidor. Na Figura 20 é possível visualizar que esta foi a resposta do servidor e que os logs foram impressos com o formato esperado.

```
24:11:2022,17:31:41:294 QR 10.0.19.20 12652,0.0.0.0:greens.,DRAGONS;;; <te/Cliente# java Cliente dnsc1 10.0.0.10 greens. DRAGONS
12652,0.0.0.0:greens.,DRAGONS;;;
12652,A,3.0.0.0:greens.,DRAGONS;
root@client:/tmp/pycore.33219/client.conf/Cliente/Cliente#
```

Figura 20: Teste Erro 3

5.2.7 Querie SR com DD

```
02:01:2023,18:31:07:714 QR 10.0.0.14 57355,Q,0,0,0,0;:greens.dance.,NS;:::
02:01:2023,18:31:07:719 RP 10.0.0.14 57355,A,0,3,3,3;:greens.dance.,NS;:greens.d
ance, NS ns1.greens.dance, 86400 1000000 true file;:greens.dance, NS ns2.greens.d
ance, 86400 1000000 true file;:greens.dance, NS ns3.greens.dance, 86400 1000000
true file;:greens.dance, NS ns1.greens.dance, 86400 1000000 true file;:greens.d
ance, NS ns2.greens.dance, 86400 1000000 true file;:greens.dance, NS ns3.greens.d
ance, 86400 1000000 true file;ns1.greens.dance, A 10.0.0.10 86400 1000000 true
file;ns2.greens.dance, A 10.0.9.10 86400 1000000 true file;ns3.greens.dance, A 1
0.0.11.10 86400 1000000 true file;
```

Figura 21: Servidor Primário .greens.dance.

```
Dominio: .greens.dance, DD: [10.0.0.10]
02:01:2023,18:31:07:719 QE 10.0.0.10 57355,Q,0,0,0,0;:greens.dance.,NS;:::
02:01:2023,18:31:07:721 RR 10.0.0.10 57355,A,0,3,3,3;:greens.dance.,NS;:greens.d
ance, NS ns1.greens.dance, 86400 1000000 true file;:greens.dance, NS ns2.greens.d
ance, 86400 1000000 true file;:greens.dance, NS ns3.greens.dance, 86400 1000000
true file;:greens.dance, NS ns1.greens.dance, 86400 1000000 true file;:greens.d
ance, NS ns2.greens.dance, 86400 1000000 true file;:greens.dance, NS ns3.greens.d
ance, 86400 1000000 true file;ns1.greens.dance, A 10.0.0.10 86400 1000000 true
file;ns2.greens.dance, A 10.0.9.10 86400 1000000 true file;ns3.greens.dance, A 1
0.0.11.10 86400 1000000 true file;

02:01:2023,18:31:07:733 RP 10.0.19.20 57355,A,0,3,3,3;:greens.dance.,NS;:greens.d
ance, NS ns1.greens.dance, 86400 1000000 true file;:greens.dance, NS ns2.greens.d
ance, 86400 1000000 true file;:greens.dance, NS ns3.greens.dance, 86400 1000000
0 true file;:greens.dance, NS ns1.greens.dance, 86400 1000000 true file;:greens.d
ance, NS ns2.greens.dance, 86400 1000000 true file;:greens.dance, NS ns3.greens.d
ance, 86400 1000000 true file;ns1.greens.dance, A 10.0.0.10 86400 1000000 true
file;ns2.greens.dance, A 10.0.9.10 86400 1000000 true file;ns3.greens.dance, A
10.0.11.10 86400 1000000 true file;
```

Figura 22: Servidor de Resolução no domínio .greens.dance.

```
vcmd
Response code: 0
Number of Values: 3
Number of Authorities: 3
Number of Extra Values: 3

Query Info:
Name: .greens.dance, Type of Value: NS;

Response Values:
.greens.dance, NS ns1.greens.dance, 86400 1000000
.greens.dance, NS ns2.greens.dance, 86400 1000000
.greens.dance, NS ns3.greens.dance, 86400 1000000`

Authorities Values:
.greens.dance, NS ns1.greens.dance, 86400 1000000
.greens.dance, NS ns2.greens.dance, 86400 1000000
.greens.dance, NS ns3.greens.dance, 86400 1000000`

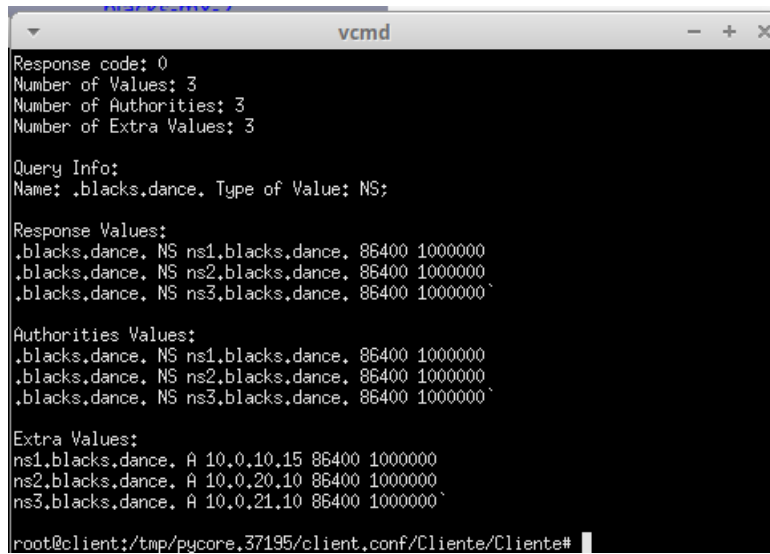
Extra Values:
ns1.greens.dance, A 10.0.0.10 86400 1000000
ns2.greens.dance, A 10.0.9.10 86400 1000000
ns3.greens.dance, A 10.0.11.10 86400 1000000`

root@client:/tmp/pycore.37195/client.conf/Cliente/Cliente#
```

Figura 23: Resultado final no Cliente

Nesta query, o cliente enviou como pedido .greens.dance. NS ao SR localizado no domínio .greens.dance. Como o domínio por defeito DD deste SR é o servidor primário do .greens.dance., o SR reencaminha a mesma query a este SP. O SP responde ao SR com o resultado da query e, por fim, o SR reencaminha esta resposta ao Cliente.

5.2.8 Querie SR usando o ST



```

vcmd
Response code: 0
Number of Values: 3
Number of Authorities: 3
Number of Extra Values: 3

Query Info:
Name: .blacks.dance. Type of Value: NS;

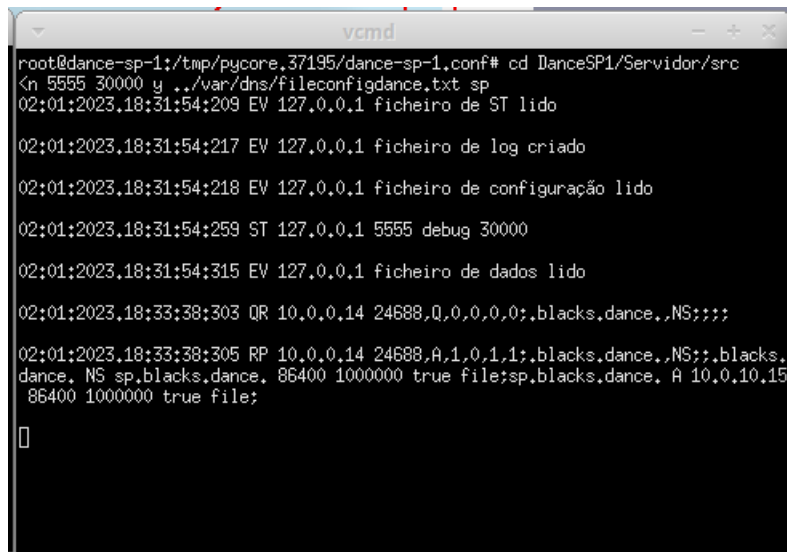
Response Values:
.blacks.dance. NS ns1.blacks.dance. 86400 1000000
.blacks.dance. NS ns2.blacks.dance. 86400 1000000
.blacks.dance. NS ns3.blacks.dance. 86400 1000000`

Authorities Values:
.blacks.dance. NS ns1.blacks.dance. 86400 1000000
.blacks.dance. NS ns2.blacks.dance. 86400 1000000
.blacks.dance. NS ns3.blacks.dance. 86400 1000000`

Extra Values:
ns1.blacks.dance. A 10.0.10.15 86400 1000000
ns2.blacks.dance. A 10.0.20.10 86400 1000000
ns3.blacks.dance. A 10.0.21.10 86400 1000000`
root@client:/tmp/pycore.37195/client.conf/Cliente/Cliente#

```

Figura 24: Cliente



```

vcmd
root@dance-sp-1:/tmp/pycore.37195/dance-sp-1.conf# cd DanceSP1/Servidor/src
<n 5555 30000 y ../var/dns/fileconfigdance.txt sp
02:01:2023,18:31:54:209 EV 127,0,0,1 ficheiro de ST lido

02:01:2023,18:31:54:217 EV 127,0,0,1 ficheiro de log criado

02:01:2023,18:31:54:218 EV 127,0,0,1 ficheiro de configuração lido

02:01:2023,18:31:54:259 ST 127,0,0,1 5555 debug 30000

02:01:2023,18:31:54:315 EV 127,0,0,1 ficheiro de dados lido

02:01:2023,18:33:38:303 QR 10,0,0,14 24688,Q,0,0,0,0;.blacks.dance.,NS;;;

02:01:2023,18:33:38:305 RP 10,0,0,14 24688,A,1,0,1,1;.blacks.dance.,NS;;.blacks.
dance. NS sp.blacks.dance. 86400 1000000 true file;sp.blacks.dance. A 10,0,10,15
86400 1000000 true file;

```

Figura 25: ST .dance.



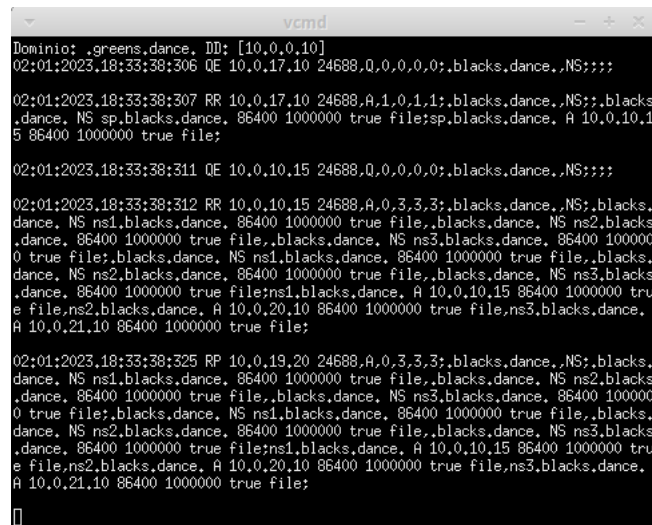
```

02:01:2023,18:33:38:308 QR 10,0,0,14 24688,Q,0,0,0,0;.blacks.dance.,NS;;;

02:01:2023,18:33:38:310 RP 10,0,0,14 24688,A,0,3,3,3;.blacks.dance.,NS;.blacks.d
ance. NS ns1.blacks.dance. 86400 1000000 true file;.blacks.dance. NS ns2.blacks.
dance. 86400 1000000 true file;.blacks.dance. NS ns3.blacks.dance. 86400 1000000
true file;.blacks.dance. NS ns1.blacks.dance. 86400 1000000 true file;.blacks,d
ance. NS ns2.blacks.dance. 86400 1000000 true file;.blacks.dance. NS ns3.blacks.
dance. 86400 1000000 true file;ns1.blacks.dance. A 10,0,10,15 86400 1000000 true
file;ns2.blacks.dance. A 10,0,20,10 86400 1000000 true file;ns3.blacks.dance. A
10,0,21,10 86400 1000000 true file;

```

Figura 26: SP .blacks.dance.



```
vcmd
Dominio: .greens.dance, ID: [10,0,0,10]
02:01:2023,18:33:38:306 QE 10,0,17,10 24688,Q,0,0,0,0;.blacks.dance,.NS;;;
02:01:2023,18:33:38:307 RR 10,0,17,10 24688,A,1,0,1,1;.blacks.dance,.NS;.blacks
.dance, NS sp.blacks.dance, 86400 1000000 true file;sp.blacks.dance, A 10,0,10,1
5 86400 1000000 true file;
02:01:2023,18:33:38:311 QE 10,0,10,15 24688,Q,0,0,0,0;.blacks.dance,.NS;;;
02:01:2023,18:33:38:312 RR 10,0,10,15 24688,A,0,3,3,3;.blacks.dance,.NS;.blacks
.dance, NS ns1.blacks.dance, 86400 1000000 true file;.blacks.dance, NS ns2.blacks
.dance, 86400 1000000 true file;.blacks.dance, NS ns3.blacks.dance, 86400 100000
0 true file;.blacks.dance, NS ns1.blacks.dance, 86400 1000000 true file;.blacks
.dance, NS ns2.blacks.dance, 86400 1000000 true file;.blacks.dance, NS ns3.blacks
.dance, 86400 1000000 true file;ns1.blacks.dance, A 10,0,10,15 86400 1000000 tru
e file;ns2.blacks.dance, A 10,0,20,10 86400 1000000 true file;ns3.blacks.dance,
A 10,0,21,10 86400 1000000 true file;
02:01:2023,18:33:38:325 RP 10,0,19,20 24688,A,0,3,3,3;.blacks.dance,.NS;.blacks
.dance, NS ns1.blacks.dance, 86400 1000000 true file;.blacks.dance, NS ns2.blacks
.dance, 86400 1000000 true file;.blacks.dance, NS ns3.blacks.dance, 86400 100000
0 true file;.blacks.dance, NS ns1.blacks.dance, 86400 1000000 true file;.blacks
.dance, NS ns2.blacks.dance, 86400 1000000 true file;.blacks.dance, NS ns3.blacks
.dance, 86400 1000000 true file;ns1.blacks.dance, A 10,0,10,15 86400 1000000 tru
e file;ns2.blacks.dance, A 10,0,20,10 86400 1000000 true file;ns3.blacks.dance,
A 10,0,21,10 86400 1000000 true file;
```

Figura 27: SR .greens.dance.

O Cliente enviou uma query .blacks.dance. NS para o SR localizado no domínio .greens.dance. Como o seu domínio por defeito DD não corresponde ao nome enviado na query, o SR então reencaminha a query ao ST. O ST possui o IP do servidor primário do .blacks.dance. e envia esta resposta referência de volta para o SR. Por fim, o SR reencaminha a query para o SP do blacks e, quando recebe a resposta final, reencaminha para o Cliente.

5.2.9 Prova de Resiliência

```

vcmd
Dominio: .greens.dance, ID: [10,0,0,10]
02:01:2023,17:50:17:433 QE 10,0,0,10 61521,Q,0,0,0,0;.greens.dance.,MX;;;

02:01:2023,17:50:47:675 TO 10,0,0,10 Timeout Query

02:01:2023,17:50:47:675 QE 10,0,9,10 61521,Q,0,0,0,0;.greens.dance.,MX;;;

02:01:2023,17:50:47:683 RR 10,0,9,10 61521,A,0,2,3,5;.greens.dance.,MX;.greens.dance.,MX mx1.greens.dance, 86400 1000000 true other;.greens.dance.,MX mx2.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns1.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns2.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns3.greens.dance, 86400 1000000 true other;.greens.dance.,A 10,0,0,10 86400 1000000 true other;.greens.dance.,A 10,0,9,10 86400 1000000 true other;.greens.dance.,A 10,0,11,10 86400 1000000 true other;.greens.dance.,A 10,0,12 86400 1000000 true other;.greens.dance.,A 10,0,13 86400 1000000 true other;

02:01:2023,17:50:47:693 RP 10,0,19,20 61521,A,0,2,3,5;.greens.dance.,MX;.greens.dance.,MX mx1.greens.dance, 86400 1000000 true other;.greens.dance.,MX mx2.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns1.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns2.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns3.greens.dance, 86400 1000000 true other;.greens.dance.,A 10,0,0,10 86400 1000000 true other;.greens.dance.,A 10,0,9,10 86400 1000000 true other;.greens.dance.,A 10,0,11,10 86400 1000000 true other;.greens.dance.,A 10,0,12 86400 1000000 true other;.greens.dance.,A 10,0,13 86400 1000000 true other;

[

vcmd
Message id: 61521
Flags: A
Response code: 0
Number of Values: 2
Number of Authorities: 3
Number of Extra Values: 5

Query Info:
Name: .greens.dance, Type of Value: MX;

Response Values:
.greens.dance, MX mx1.greens.dance, 86400 1000000
.greens.dance, MX mx2.greens.dance, 86400 1000000

Authorities Values:
.greens.dance, NS ns1.greens.dance, 86400 1000000
.greens.dance, NS ns2.greens.dance, 86400 1000000
.greens.dance, NS ns3.greens.dance, 86400 1000000

Extra Values:
ns1.greens.dance, A 10,0,0,10 86400 1000000
ns2.greens.dance, A 10,0,9,10 86400 1000000
ns3.greens.dance, A 10,0,11,10 86400 1000000
mx1.greens.dance, A 10,0,0,12 86400 1000000
mx2.greens.dance, A 10,0,0,13 86400 1000000

[

```

Figura 28: SR .greens.dance / Cliente

```

vcmd
<a Main 5555 30000 y ../var/dns/fileconfigsslgreens.txt ss #
02:01:2023,17:47:48:76 EV 127,0,0,1 arquivo de ST lido

02:01:2023,17:47:48:85 EV 127,0,0,1 arquivo de log criado

02:01:2023,17:47:48:86 EV 127,0,0,1 arquivo de configuração lido

02:01:2023,17:47:48:245 ZT 10,0,0,10 SS 49 ms 1496 bytes

02:01:2023,17:50:47:676 QR 10,0,0,14 61521,Q,0,0,0,0;.greens.dance.,MX;;;

02:01:2023,17:50:47:691 RP 10,0,0,14 61521,A,0,2,3,5;.greens.dance.,MX;.greens.dance.,MX mx1.greens.dance, 86400 1000000 true other;.greens.dance.,MX mx2.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns1.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns2.greens.dance, 86400 1000000 true other;.greens.dance.,NS ns3.greens.dance, 86400 1000000 true other;.greens.dance.,A 10,0,0,10 86400 1000000 true other;.greens.dance.,A 10,0,9,10 86400 1000000 true other;.greens.dance.,A 10,0,11,10 86400 1000000 true other;.greens.dance.,A 10,0,12 86400 1000000 true other;.greens.dance.,A 10,0,13 86400 1000000 true other;

[

```

Figura 29: SS .greens.dance.

Neste caso, o Cliente há havia feito uma query .greens.dance. NS ao SR, que guardou o resultado em cache. Ao fazer a query .greens.dance. MX, o servidor primário perdeu conexão. O SR ao receber a query, procurou por uma resposta direta e não encontrou, portanto, procurou por uma referência e encontrou para o SP e os dois SS de .greens.dance., pois foram guardados anteriormente em cache após a query previamente mencionada. O SR então tenta reencaminhar a query ao SP, e imprime um log de timeout, visto que o SP perdeu a conexão. Segue portanto para o próximo IP, que corresponde ao SS e, por fim, consegue obter a resposta que é reencaminhada ao Cliente.

6 Tabela de Atividades

Tarefa	Guilherme Martins (A92847)	Millena Freitas (A97777)	Vasco Oliveira (A96361)
B.1	5%	90%	5%
B.2	90%	5%	5%
B.3	5%	5%	90%
B.4	40%	20%	40%
B.5	20%	60%	20%
B.6	33%	33%	33%
B.7	30%	30%	40%
B.8	40%	30%	30%

Figura 30: Tabela de Atividades

7 Conclusão

É inegável a importância do DNS para o normal funcionamento da Internet. Sem este protocolo, não conheceríamos a Internet como ela é hoje em dia. Por isso, a sua compreensão é um tópico fundamental para a formação de qualquer Engenheiro Informático e/ou de Redes.

Assim, ao estruturar e implementar uma versão mais simplificada do DNS, mas que ainda reflete os pontos importantes na realidade, foi possível consolidar os nossos conhecimentos sobre o seu funcionamento, bem como compreender melhor as comunicações via UDP e TCP.

A primeira fase deste projeto consistiu na construção de protótipos de servidores primário e secundário que funcionam em modo debug conciso e são capazes de realizar o procedimento de transferência de zona simples sem a verificação da versão da base de dados e de receber e responder queries. Foi seguido o formato dos ficheiros, da estrutura do PDU e da estrutura requisitada para os Logs. Também foi implementado o Cliente em modo debug conciso que suporta o envio de queries e a visualização de suas respostas através do terminal.

Para a primeira fase do trabalho, a primeira dificuldade encontrada foi em perceber o funcionamento do DNS com relação a hierarquia dos servidores. Quando isto se tornou consolidado, a análise de como implementar cada passo necessário para o projeto em termos de código tornou-se muito mais eficaz. A dificuldade seguinte foi com relação aos detalhes necessários para o enriquecimento do relatório, porém estes detalhes contribuíram para o próprio entendimento do grupo ao reler excertos do relatório a fim de relembra-los ou confirmar alguns assuntos.

A implementação da comunicação via UDP e TCP por sockets também começou com uma certa dificuldade, porém após perceber o que era necessário e estruturar o passo a passo, tornou-se simples e a implementação no projeto seguiu de imediato.

Para esta segunda fase, foi implementado um grau de paralelismo para possibilitar que servidores respondam a diversos clientes ao mesmo tempo, visto que o serviço DNS é muito utilizado e precisa de ser rápido. Portanto, um servidor é capaz de responder a vários clientes concorrentemente.

O Parser da primeira fase era simples e não contava com a funcionalidade de concatenar o nome DEFAULT caso não terminasse com '.'. Nesta fase, esta funcionalidade foi implementada.

Com relação à transferência de zona, nesta fase ela se tornou mais completa e robusta. Antes os servidores secundários apenas submetiam o seu pedido ao arrancar, quando na realidade isto pode ser feito esporadicamente. Portanto, foi outro motivo que levou à necessidade de uma implementação não sequencial, pois assim o servidor primário pode responder ao servidor secundário sem interromper o seu serviço de receber e responder a queries. Dito isto, agora os servidores secundários enviam esporadicamente pedidos de transferência de zona, há verificações por parte do servidor primário para garantir que o servidor secundário possui permissão para receber o conteúdo da base de dados e há timeouts.

Também foi implementado um Servidor de Resolução que funciona em modo iterativo, bem como o serviço de caching para este servidor. Desta forma, o tempo de resposta para um cliente pode ser reduzido ao encontrar esta resposta na cache. E, por fim, foi implementado o DNS Reverso no qual através de um IP é possível encontrar o seu nome.

As dificuldades encontradas nesta fase foram relacionadas primeiramente com multithreading, no qual tivemos que implementar um controle de concorrência de modo a garantir consistência dos dados. Após isto, surgiu dificuldade com relação à implementação da lógica por trás do Servidor de Resolução, principalmente por conta dos possíveis timeouts e para garantir a sua resiliência. E, por fim, ao decorrer da implementação da cache também surgiram dificuldades em relação à lógica da procura, referências, atualização do TTL e validação das linhas.

Acreditamos que há diversos pontos que poderiam ter sido implementados, como por exemplo mais logs, controlo de erros mais robusto, outras funcionalidades opcionais, como o modo normal e o modo recursivo. E concordamos que seria ótimo proceder com estas implementações de maneira a solidificar este projeto e torná-lo apto para publicação. Porém, com relação ao projeto final, concluímos que corresponde ao mínimo esperado de uma implementação de um sistema DNS e que foi muito útil não só para aprender sobre DNS, mas também para consolidar e adquirir conhecimentos sobre comunicação através de sockets UDP e TCP, multithreading, caching, parser de ficheiros, entre outros assuntos que são de imensa importância para o percurso académico como um todo.

8 Bibliografia

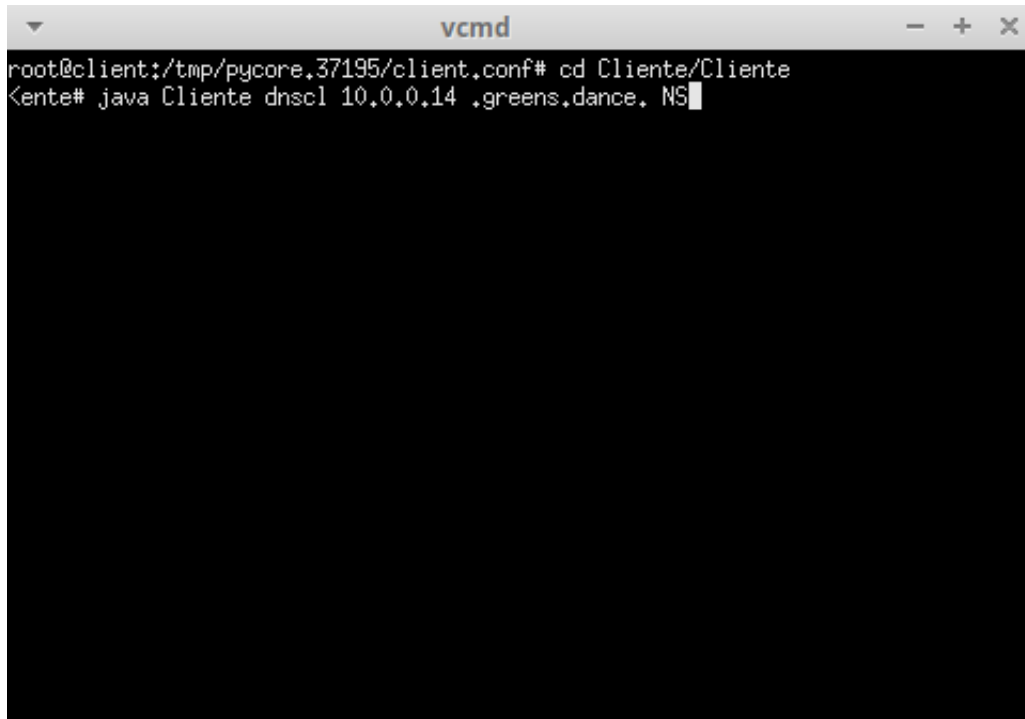
- How to increase dns timeout value. Marktugbo, 2019. Disponível em: <https://marktugbo.com/2019/04/29/how-to-increase-dns-timeout-value/>
- DNS Root Servers: What Are They and Are There Really Only 13?. Sara Jelen, Security Trails. Disponível em: <https://securitytrails.com/blog/dns-root-servers>
- O que é um servidor raiz de DNS? Cloudflare. Disponível em: <https://www.cloudflare.com/pt-br/learning/dns/glossary/dns-root-server/>
- Understanding DNS cache. Catchpoint, 2014. Disponível em: <https://www.catchpoint.com/blog/dns-cache>
- What is DNS and DNS hierarchy? WebdevelopersTuts. Disponível em: <http://www.webdevelopertuts.com/what-is-DNS-and-DNS-hierarchy>
- O que é segurança de DNS? Cloudflare. Disponível em: www.cloudflare.com/pt-br/learning/dns/dns-security/
- Convert String into Binary Sequence. GeeksforGeeks, 2021. Disponível em: <https://www.geeksforgeeks.org/convert-string-binary-sequence/>

9 Anexos

Apresenta-se nesta secção um manual de utilização dos programas desenvolvidos para esta aplicação.

9.1 Manual de utilização

9.1.1 Cliente



```
vcmd
root@client:/tmp/pycore.37195/client.conf# cd Cliente/Cliente
Kente# java Cliente dnsc 10.0.0.14 .greens.dance. NS
```

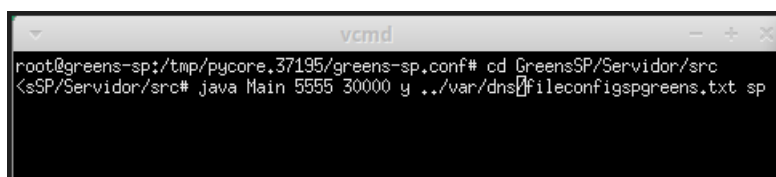
Figura 31: Argumentos para executar o Cliente

Para executar o Cliente, é necessário primeiro escrever "java Cliente" por conta da forma que os ficheiros foram compilados ao utilizar jar. Em seguida deve seguir este padrão:

dnsc <IP> <NAME> <TYPE OF VALUE>

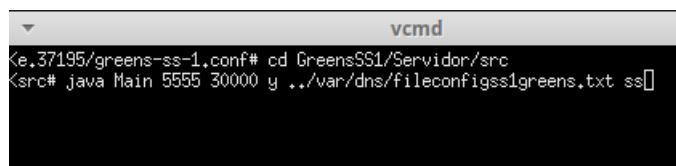
Onde o IP corresponde ao IP do servidor que deseja enviar a query. Para mais informações sobre os campos NAME e TYPE OF VALUE leia **aqui**.

9.1.2 Servidor



```
vcmd
root@greens-sp:/tmp/pycore.37195/greens-sp.conf# cd GreensSP/Servidor/src
<sSP/Servidor/src# java Main 5555 30000 y ../var/dns/fileconfigspgreens.txt sp
```

Figura 32: Argumentos para executar o Servidor Primário



```
vcmd
Ke,37195/greens-ss-1.conf# cd GreensSS1/Servidor/src
<src# java Main 5555 30000 y ../var/dns/fileconfigss1greens.txt ss
```

Figura 33: Argumentos para executar o Servidor Secundário



```
vcmd
root@greens-sr:/tmp/pycore.37195/greens-sr.conf# cd GreensSR/Servidor/src
<sSR/Servidor/src# java Main 5555 30000 y ../var/dns/fileconfigsrngreens.txt sr
```

Figura 34: Argumentos para executar o Servidor de Resolução

Para executar o Servidor , é necessário primeiro escrever "java Main" por conta da forma que os ficheiros foram compilados ao utilizar jar. Em seguida deve seguir este padrão:

<porta de atendimento> <timeout> <y/n> <caminho para o ficheiro de configuração> <tipo de servidor>

- O timeout deve ser dado em milissegundos.
- y significa que deseja em modo debug. Apenas o modo debug foi implementado nesta aplicação.
- O tipo de servidor pode ser: sp, ss ou sr.