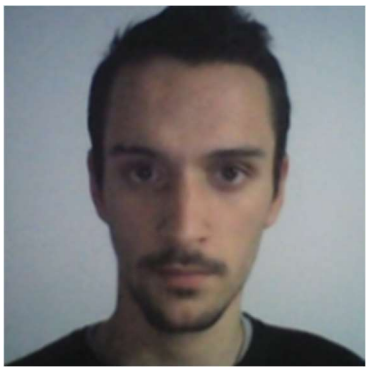
Universidade do Minho

Trabalho Prático 2A

Implementação de Sistema DNS

Comunicações por Computador

2022-2023



Artur Leite- A97027

Afonso Magalhães- A95250

Nuno Rodrigues- A90439

Grupo 4.01

17/11/2022

Índice

1. Introdução

Para este trabalho, foi-nos pedido uma implementação de um sistema de comunicação DNS, que implica um desenvolvimento de um conjunto de servidores que comunicam entre si. Para tal, começamos por rever os conceitos mais importantes na abordagem a este desafio, nomeadamente:

- Funcionamento dos serviços DNS da arquitetura TCP/IP;
- Características dos protocolos de transporte UDP e TCP;
- Capacidade de análise das unidades de dados de protocolo (PDU);
- Saber especificar as primitivas de serviço dum protocolo aplicacional;
- Saber aplicar o paradigma dos *sockets*;

Optamos ainda por utilizar a linguagem de programação *Python* devido à experiência que já tínhamos adquirido noutros projetos.

2. Arquitetura do Sistema

Após analisar o enunciado, identificamos 4 elementos de elevada importância para a nossa arquitetura:

- **SS:** Servidor que recebe e responde a *queries* e também possui autorização e autoridade de possuir uma cópia da base de dados dos SPs dos domínios dos quais são autoritativos;
- **SP:** Servidor que recebe e responde a *queries* e que tal como o SS é autoritativo de um determinado domínio, porém este já possui acesso direto à base de dados original do mesmo;
- **SR:** Servidor que recebe e responde a *queries* mas que não possui nenhuma autoridade sobre qualquer domínio, sendo que para responder às *queries* que recebe deve ser ele a questionar os servidores SP e/ou SS para que estes lhe forneçam as informações que necessita, ou pelo menos informações intermédias que ajudem na obtenção daquilo que realmente requer;
- **CL:** Aplicação cliente de DNS que necessita de obter informação presente na base de dados de um determinado domínio, questionando desta forma um determinado servidor, normalmente um SR, para obter esta informação;

Adicionalmente, existem ainda mais duas variantes de servidores que teremos de considerar, sendo estes os **Servidores de Topo (ST) e Servidores de Domínio de Topo (SDT).** Os STs possuem comportamento igual a um SP ou um SS, possuindo uma Base de Dados que contem a informação sobre todos os SDTs, desta forma, quando é necessário descobrir servidores DNS para os Dominios de Topo, é ao ST que é efetuado esse pedido. Por outro lado, os SDTs representam os *.com, .pt, .org, etc.*

3. Modelo de Informação

Para um bom funcionamento do sistema, existe a necessidade de manter a consistência e a semântica de elementos de configuração relativos aos componentes servidores, nomeadamente os **ficheiros de configuração**, **ficheiros de dados e ficheiros de log.**

3.1. Ficheiros de Configuração

Quando este ficheiro é lido e processado, os Servidores passam a possuir os parâmetros necessários para adaptar o seu comportamento. Possuem a sintaxe *{parâmetro} {tipo do valor} {valor associado ao parâmetro}.* Eis as opções comportamentais presentes nestes ficheiros:

- **DB-** indica o ficheiro da base de dados com a informação deste domínio;
- **SP-** indica o endereço IP do SP deste domínio;
- **SS-** indica o endereço IP de um SS deste domínio;
- **DD-** indica o endereço IP de um SR, SS ou SP do domínio por defeito indicado no parâmetro;
- **ST-** indica o ficheiro com a lista de ST's (o parâmetro deve ser igual a "root");
- **LG-** indica o ficheiro de log a utilizar em cada atividade associada a este domínio (o parâmetro deve ser igual a "all").

Situação de Erro de Configuração: O sistema terá de ser capaz de garantir que os domínios que o perfazem são compostos pelos elementos corretos. A título de exemplo, não poderá existir um domínio que possui mais que um elemento SP. A deteção deste tipo de erros dá-se no momento de leitura dos ficheiros de configuração, logo, existe a necessidade de implementar mecanismos de deteção nos nossos métodos de leitura.

Para além disso, o sistema terá ainda de garantir a coerência dos parâmetros definidos:

- O servidor não pode ser SS e SP ao mesmo tempo;
- Não pode receber um ficheiro log para um domínio ao qual não é SP nem SS;
- O servidor tem de receber um all log file obrigatoriamente;
- O servidor tem de receber um ficheiro DB para poder aceitar entradas de ficheiros SS;

3.2. Ficheiros de *Log*

Toda a atividade relevante efetuada pelo componente ao qual este ficheiro está associado deverá ser sempre registada neste. A existência deste ficheiro é sempre verificada no ficheiro de configuração. Caso contrário, este deve ser criado no arranque do servidor. Toda a informação adicionada deve ser registada após a última entrada já existente. As linhas de configuração devem possuir a sintaxe *{etiqueta temporal} {tipo de entrada} {endereço IP[:Porta]} {dados da entrada}.* Eis as configurações possíveis para estes ficheiros:

- **QR/QE-** indica que recebeu/enviou uma query de/para o endereço no parâmetro;
- **RR/RE-** indica que recebeu/enviou uma resposta à query para o/do endereço no parâmetro;
- **ZT-** indica o sucesso na execução de uma transferência de zona;
- **EV-** indica atividade interna no componente;
- **ER-** indica a presença de erro na descodificação;

- **EZ-** indica a presença de erro na transferência de zona;
- **FL-** indica a presença de erro interno ao componente;
- **TO-** indica *timeout* entre o servidor e o IP indicado;
- **SP-** indica a paragem de execução do componente; C
- **ST-** indica o início da execução do componente.

3.3. Ficheiros de Dados

Estes ficheiros estão presentes nos SPs e possuem a seguinte sintaxe *{parâmetro} {tipo do valor} {valor} {tempo de validade} {prioridade}.* Os tipos de valores possíveis em cada linha são os seguintes:

- **DEFAULT (opcional)-** define um nome que deverá ser substituído pelo valor literal associado no futuro;
- **SOASP-** indica o nome completo SP do domínio;
- **SOAADMIN-** indica o e-mail completo do administrador do domínio;
- **SOASERIAL-** indica o número de série da base de dados do SP do domínio;
- **SOAREFRESH-** indica o intervalo de tempo para um SS perguntar ao SP do domínio indicado no parâmetro qual o número de série da base de dados dessa zona;
- **SOARETRY-** indica o intervalo de tempo para um SS voltar a perguntar ao SP do domínio indicado qual o número de série da base de dados dessa zona, após um *timeout;*
- **SOAEXPIRE-** indica o tempo que leva para um SS invalidar a sua replica da base de dados da zona;
- **NS-** indica o nome de um servidor autoritativo para o domínio;
- **A-** Indica o IPv4 do *host* servidor evidenciado no parâmetro;
- **CNAME-** indica o nome canónico associado ao nome indicado no parâmetro. Não deve apontar para outro nome canónico nem podem existir outros parâmetros com o mesmo valor;
- **MX-** indica o nome de um servidor e-mail;
- **PTR-** indica o nome de um servidor/*host* que usa o IPv4 indicado no parâmetro.

Adicionalmente, o campo {*tempo de validade}* representa o tempo máximo em segundos que os dados podem existir numa cache de um servidor sem serem utilizados.

Por sua vez, o campo {*prioridade} define* uma ordem prioritária constituída por vários valores associados ao mesmo parâmetro (quanto menor o valor, maior a prioridade). Este assume ainda um valor inteiro menor que 256 e não deverá existir quando um parâmetro tem apenas um valor ou quando todos os valores do mesmo possuem um nível de prioridade igual.

4. Modelo de Comunicação

No nosso sistema, todas as interações entre componentes surgem das *queries* pedidas pelos clientes, que iniciam uma cadeia de troca de mensagens entre os componentes de forma a obter a resposta ao problema pedido pelo cliente. Numa situação normal, um componente servidor deverá ser capaz de processar e descodificar o pedido*,* seguido da procura pela resposta ao mesmo.

4.1. Comportamento de Resposta

Quando um componente servidor necessita de responder a uma *query,* a sua primeira ação envolve procurar na sua *cache* (ou na sua base de dados, caso seja um servidor autoritativo para o domínio NAME) pelo elemento pedido. Caso esta se encontrar aqui, o servidor obtém a resposta e envia para o cliente, sem problemas. Caso o servidor não encontrar a resposta na sua cache, ele procederá a reenviar a *query* para um SDT incluído no campo NAME. Para obter a informação deste SDT, o servidor deve procurá-la na sua cache, pedi-la a um Servidor de Topo (ST) através do envio de uma *query* ou então defini-lo no campo DD do ficheiro de configuração. Este processo irá continuar de forma iterativa até o servidor obter a resposta pretendida ou até não ser capaz de obter mais informações relativas à *query.*

4.2. Situações de Erro

Caso a execução de uma *query* der erro, o servidor devolve uma resposta onde o campo RESPONSE CODE da mensagem DNS assume o valor 1, 2 ou 3 dependendo do erro que sucedeu:

- **Valor 1-** caso o domínio incluído no campo NAME for encontrado, mas não foi encontrada qualquer informação relevante ao pedido efetuado;
- **Valor 2-** caso o domínio incluído no campo NAME não for encontrado;
- **Valor 3-** caso a mensagem DNS não foi descodificada corretamente.

4.3. Cabeçalho das Mensagens DNS

De forma a complementar as especificações necessárias para as interações entre os elementos do sistema, as mensagens DNS possuem um campo *HEADER* na sua mensagem que, por sua vez, é dividido em outros campos que detalham um conjunto de opções/informações referentes à mensagem. Eis esses possíveis campos:

- **MESSAGE ID-** corresponde ao identificador da mensagem, desta forma, é possível relacionar respostas com o pedido original;
- **FLAGS-** inclui três opções:
  - **Q-** indica que é uma *query,* caso contrário trata-se de uma resposta
  - **R-** caso esta opção estiver ativa numa *query,* vai indicar que o processo deverá ser recursivo. Por outro lado, se estiver ativa num *reply,* indica que o servidor que enviou a resposta suporta modo recursivo;
  - **A-** quando está ativa numa mensagem de *reply* indica que esta é autoritativa.

- **NUMBER OF VALUES-** indica o número de entradas que correspondem à resposta pretendida pela *query,* ou seja, indicam o número de entradas incluídas no campo *RESPONSE VALUES;*
- **NUMBER OF AUTHORITIES-** indica o número de servidores autoritativo para o domínio incluído no campo *RESULT VALUES;*
- **NUMBER OF EXTRA VALUES-** indica o número de entradas com informação adicional relacionada com os resultados da *query* ou com os servidores da lista de autoridades;
- **QUERY INFO-** indica a informação do nome da *query* que se pretende realizar e o valor associado ao parâmetro;
- **RESPONSE VALUES-** indica a lista de entradas que descrevem a correspondência entre o *NAME e o TYPE OF VALUE,* podendo estes estar na sua cache ou na Base de Dados do servidor autoritativo;

- **AUTHORITIES VALUES-** lista de entradas que descrevem a correspondência entre o *NAME* e o tipo de valor igual a *NS*, incluídos na cache ou no na Base de Dados de um servidor autoritativo;
- **EXTRA VALUES-** indica a lista de entradas do tipo A que fazem *match* no parâmetro com todos os valores no campo *RESPONSE VALUES* e no campo *AUTHORITIES VALUES.* Desta forma, o elemento que o cliente ou servidor que recebe a resposta não tenha que fazer novos processos para saber IPs dos parâmetros que vêm como valores nos outros dois campos;

4.4. Transferência de Zona

Neste sistema, as operações de transferência de zona, devem ser sempre feitas através de uma conexão TCP. Um SS vai efetuar um conjunto de *queries* constantemente de forma a verificar se a versão da base de dados do seu respetivo SP é mais recente que a cópia que possui. Caso isso se verifique, o servidor secundário vai efetuar uma Transferência de Zona, começando por enviar o nome do domínio para o qual deseja receber uma cópia da Base de Dados do SP. De seguida, o SP valida o domínio e autoriza a cópia da sua base de dados por parte do SS.

Terminada a fase de verificação, o SP envia ao SS o número de entradas do ficheiro e este responde com o mesmo número caso o aceite. Feita a confirmação por parte do SS, o SP inicia o processo de enviar as entradas da base de dados em formato de texto, numerando-as por ordem crescente. Durante isto, o SS executa um processo de verificação onde vai avaliar se recebeu todas as entradas que estava à espera até um certo tempo predefinido acabar. Caso isso acontecer, o SS termina a conexão TCP e desiste da transferência de zona. Voltará a tentar restabelecer a conexão após um intervalo de tempo que consta no parâmetro SOARETRY. Por sua vez, o SP não deverá aceitar pedidos de transferência de zona do mesmo SS com intervalo menor de SOARETRY segundos.

4.5. Sistema de Cache

A presença de um sistema de cache é fulcral para o bom desempenho do serviço de DNS. Através da cache, os elementos do serviço serão capazes de responder a *queries* de forma mais eficiente, na medida que a sua capacidade de obtenção de respostas é melhorada exponencialmente a longo prazo. Estas são mais necessárias em servidores não autoritativos, visto que estes não têm acesso à informação de outros domínios, surgindo a necessidade de perguntar a servidores autoritativo, retardando o processo. Algo colmatado pela existência da cache.

4.6. Codificação de Mensagens

Com o intuito de diminuir o espaço ocupado na memória por parte dos componentes das mensagens de DNS, será necessário implementar uma codificação binária. Desta forma, certos parâmetros serão lidos e considerados da mesma forma, mudando apenas a memória que ocupam. Tendo em conta que apenas a performance do sistema é que vai ser afetada de forma positiva, não sendo afetada de qualquer maneira o funcionamento do mesmo, torna esta implementação bastante importante por questões de otimização.

5. Métodos de Implementação

5.1. Componente Cliente e Componente Servidor

Na análise da arquitetura do sistema, começamos por identificar os 4 componentes principais: SP, SS, SR e CL. Com o intuito de integrar estes quatro elementos de forma eficiente, optamos por criar dois executáveis de grande relevância.

Primeiro, temos o componente **CL** (cliente) que será encarregue de realizar as funcionalidades requeridas pelo cliente, tais como enviar uma *query* relativamente a uma informação que necessita para um servidor, seja este intermédio ou final.

De seguida, temos o componente **Servidor**. Ora, analisando os elementos acima mencionados que representam servidores, chegamos à conclusão de que os SPs, SSs e SRs diferem essencialmente na autoridade que possuem relativamente às bases de dados dos domínios. Devido a tal, é neste componente que todos estes servidores estão definidos, ao contrário de os separar em executáveis diferentes. A distinção entre cada server é obtida pela informação que está presente nos ficheiros de configuração.

O componente **Servidor** está dividido em 3 grandes módulos. O primeiro baseia-se na configuração de um servidor, que será encarregue de dar parse a um ficheiro de configuração, devolvendo um objeto *ServerConfig* onde vai constar todos os parâmetros importantes da configuração, seja as bases de dados do servidor, os DDs, os ficheiros de log etc. O segundo modulo será o *Database* onde se irá basear maioritariamente em realizar o parse de forma a obter a configuração das bases de dados dos domínios, seja para um servidor SP através da leitura direta do ficheiro, seja para o SS, através da receção da copia do ficheiro pelo processo de transferência de zona entre o SS e o SP. O módulo de *Database* possui ainda os métodos referentes ao sistema de *caching.* Finalmente, identificamos ainda outro modulo que consideramos importante, que será relativo às mensagens de DNS, onde irá constar todas as funcionalidades importantes relativas ao protocolo de mensagens DNS, tais como a codificação das mensagens.

Um dos requisitos que consideramos muito importante relativo aos servidores seria a possibilidade de conseguir realizar as várias tarefas como receção de *queries*, envio e tratamento das mesmas e também lidar com transferências de zonas de forma concorrente, para que não se encontre bloqueado em alguma das mesmas, acreditando assim que o seu funcionamento seja mais eficiente. Desta forma, cremos que seja importante que um servidor tenha constantemente um *thread* em que irá estar a receber pedidos apenas através de um *socket*, em que a cada mensagem que este receber, o servidor deverá criar uma outra *thread* que irá tratar da descodificação e eventual resolução da *query* bem como o envio da resposta para o cliente correspondente. Na eventualidade de o servidor em questão também se tratar de um servidor SS, achamos por bem também que este possua uma *thread* por cada servidor SP do qual é SS onde irá constantemente perguntar ao respetivo servidor o número de Série da base de dados para que na eventual incompatibilidade de versões, este cria uma *thread* em que irá solicitar uma transferência de zona com o respetivo SP. Identificando assim 4 Principais *Threads*, UDPQueryReceiver, UDPQueryAnswer, UDPTransferSender, TCPZoneTransferSenderController. Esta última *thread* está a ser corrida constantemente e é ainda encarregue de criar mais uma, a TCPZoneTransferSender, quando recebe um pedido de transferência de zona.

5.2. Configuração de Servidores

Com o intuito de adquirir a configuração pretendida para um servidor, começamos por implementar um componente responsável por ler as entradas presentes no ficheiro .txt. Em cada entrada analisada, o componente afere quais os campos a ser configurados, juntamente com os parâmetros referentes a cada opção. Dependendo do campo a ser analisado, a informação será guardada de diferentes maneiras.

- Caso se trate do campo DB, a informação é adicionada a um dicionário relativo ao domínio que é acompanhado pelo nome do ficheiro de base de dados, procedendo ao parse do mesmo. A presença deste campo assume o papel de SP para o servidor que o possui, no respetivo domínio;
- Caso se trate do campo SP, a informação é adicionada a um dicionário relativo ao domínio que é acompanhado pelo endereço IP do servidor respetivo. A presença deste campo assume o papel de SS para o servidor que o possui, no respetivo domínio;
- Caso se trate do campo SS, a informação é adicionada a uma lista mapeada num dicionário pelo domínio do parâmetro. A presença deste campo assume o papel de SP para o servidor que o possui, no respetivo domínio;
- Caso se trate do campo ST, é efetuado o parse do ficheiro passado, adicionando cada servidor à lista de *root servers;*
- Caso se trate do campo DD, o servidor aí incluído é adicionado a uma lista que é mapeada pelo domínio do parâmetro. Sendo este servidor um SP ou SS, este deixa de responder a *queries* de domínios distintos do que está no parâmetro. Caso seja SR, recebendo uma *query* relativo ao domínio em questão, comunicará diretamente com esses servidores;
- Caso se trate do campo LG, a informação pode ser adicionada a dois locais distintos. Caso se trate do ficheiro *log "all",* o nome deste é adicionado a um campo único e exclusivo. Caso se trate de um ficheiro *log* de um domínio, o nome do ficheiro é adicionado a uma lista. Sendo que neste ficheiro serão registados os eventos relativos a esse mesmo domínio.

5.3. Sistema de Cache- Utilização de Locks

Na implementação do Sistema de Cache, incluímos um conjunto de métodos responsáveis pela obtenção e adição de dados na cache. Desta forma, permitimos que sejam executados os processos de resposta a *queries* que envolvam informações presentes nestas. Para além disso, como tínhamos mencionado acima, o nosso programa possui um conjunto de *threads* que estão a executar em simultâneo. Como muitas destas *threads* irão aceder à cache de forma regular para obter as informações relevantes às *queries* que querem responder, esta representa uma zona crítica no que toca ao funcionamento concorrente destes elementos. Para tal, os métodos que acedem à cache possuem *locks* que não permitem que uma *thread* lhe aceda enquanto está a ser lida ou manipulada por outro processo. Esta implementação era necessária pois a manutenção da execução concorrente das *threads* é fulcral para um bom funcionamento do sistema, acelerando o processo. No âmbito do nosso programa, incluímos os componentes de caching e base de dados no mesmo módulo.

5.4. *Timeouts*

Quanto aos *timeouts,* qualquer componente que emite uma *query*, irá ter um tempo de espera que, assim que chegar ao fim, reenvia a mesma. De seguida, caso o tempo de *timeout* seja atingido mais uma vez, o processo de reenvio é efetuado mais uma vez. No terceiro envio, caso o tempo seja atingido outra vez, já não será executado o reenvio, resultando na suspensão do processo na sua totalidade. Para tal, utilizamos a função *settimeout*, pré-definida em Python.

5.5. Implementação das *Zone Transfers*

Tal como mencionamos nos nossos componentes, existe a *thread* UDPTransferSender que irá estar a correr constantemente perguntando ao SS qual é o seu número de série. Quando se depara com um diferente do esperado, o SS envia um pedido de conexão TCP ao *socket* presente no TCPZoneTransferSenderController. Após isto, caso a conexão for estabelecida com sucesso, o SP cria uma *thread* TCPZoneTransferSender, focada principalmente nesta transferência, em que, recebendo uma mensagem do SS com o domínio do qual deseja receber a cópia da BD sendo este verificado pelo SP, seja da existência do domínio na sua configuração, seja pela permissão do SS receber essa mesma cópia. De seguida, caso estas verificações tenham sucesso, o servidor trata da emissão do ficheiro da base de dados, seja até este terminar ou até passar o tempo pré-definido na qual esta transferência deve ser realizada.

5.6. Implementação da Codificação Binária de Mensagens

De maneira a reduzir o espaço ocupado pelas mensagens, seja por informação inútil ou por excesso de bytes usados para cada campo, decidimos proceder a uma implementação de um conjunto de métodos com esse mesmo objetivo:

- No campo MESSAGE ID, visto que está compreendido entre 1 e 65535, achamos suficiente a utilização de apenas dois bytes, em vez do um byte ocupado por algarismo.
- Para os campos QUERY FLAG e RESPONSE CODE, tendo em conta que estes têm três (3 bits) e quatro (2 bits) opções respetivamente, optamos por incluir estes dois parâmetros no mesmo byte;
- Para os campos NUMBER OF VALUES, NUMBER OF EXTRA VALUES e NUMBER OF AUTHORITIES, sendo que estes estão compreendidos entre 0 e 255, achamos também necessário apenas utilizar um *byte* por cada um;
- O campo QUERY INFO NAME seria precedido por um *byte* que corresponderia ao tamanho do domínio presente no NAME e de seguida a própria string, visto que os domínios têm tamanho variável, assumindo também que não será maior que 255.
- No campo QUERY INFO TYPE, visto que as possibilidades para este parâmetro são apenas 12, podendo ser desta forma codificado com 4 *bits*, decidimos codificar este campo com um *byte.*
- Nos campos de resposta, visto que são maioritariamente constituídos por (domínio, tipo, valor, [TTL], [prioridade]) decidimos codificar de forma semelhante aos campos da QUERY INFO, sendo portanto o domínio precedido por um byte indicando o seu tamanho, o tipo será constituído por um *byte*, o valor será codificado de forma semelhante ao domínio, o campo TTL, sendo opcional, deverá ser precedido por uma flag (1 bit) indicando a sua existência e finalmente o campo prioridade vai ser precedido por uma flag (1 bit) por ser também opcional seguido de um *byte* para o codificar;

6. Ambiente de Teste

Com o intuito de realizar testes de verificação do funcionamento dos componentes que perfazem o nosso sistema, implementamos uma simulação de um sistema DNS no CORE. Este ambiente de teste possui os seguintes elementos:

- Dois servidores de topo, ST1 e ST2 que agem como SP e SS respetivamente. Estes estão situados muito distantes um do outro, o que faz com que pertençam a sub-redes diferentes. Isto é relevante para que caso um deles falhe, o acesso à *root* não é totalmente perdido;
- Dois servidores de domínio de topo, Tiroliroliro e o Tiroliroló. Ambos possuem um subdomínio, sendo estes o Esquina e o Concertina, respetivamente. Tanto os domínios como os subdomínios possuem um SP e dois SS, sendo que estão situados em sub-redes diferentes, devido às mesmas razões pelas quais os STs estão em locais distantes;
- Em seguida, possuímos ainda um domínio de topo nomeado de reverse, onde estarão implementados os domínios de DNS reverso, contendo como subdomínio o addr;
- Temos ainda um SR que está presente no subdomínio esquina;
- Um CL cuja posição poderia ser alterada livremente;

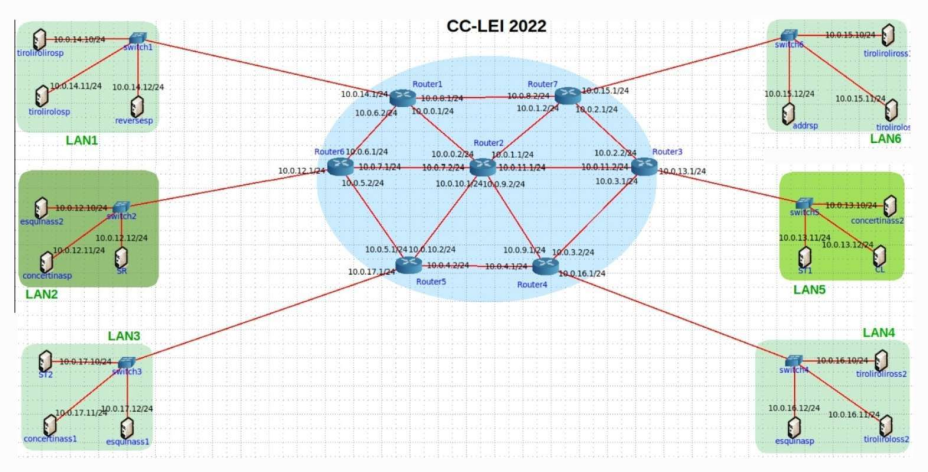


Figura 1- Ambiente de Teste em Core

7. Ficheiros de Configuração

Configuração do servidor primário do domínio reverse

```
reverse DB /var/dns/reverse.db

reverse LG /var/dns/reverse.log

all LG /var/dns/reverse.log

root ST /var/dns/root-servers.db
```

Configuração do servidor de topo ST1

```
. DB /var/dns/root.db
. SS 10.0.17.10
. LG /var/dns/root.log
all LG /var/dns/all.log
```

Configuração do servidor de topo ST2

```
. SP 10.0.13.11
. LG /var/dns/root.log
all LG /var/dns/all.log
```

Configuração do Servidor Primário addr

```
reverse.soledo DB /var/dns/reverse-addr.db
reverse.soledo LG /var/dns/reverse-addr.log
all LG /var/dns/reverse-addr.log
root ST /var/dns/root-servers.db
```

Configuração do Servidor Primário Esquinasp

```
tiroliroliro.esquina DB /var/dns/tiroliroliro-esquina.db
tiroliroliro.esquina SS 10.0.17.12
tiroliroliro.esquina SS 10.0.12.10
tiroliroliro.esquina LG /var/dns/tiroliroliro-esquina.log
all LG /var/dns/all.log
root ST /var/dns/root-servers.db
```

Configuração do Servidor Secundário Esquinass1

```
tiroliroliro.esquina SP 10.0.16.12
tiroliroliro.esquina LG /var/dns/tiroliroliro-esquina.log
all LG /var/dns/all.log
root ST /var/dns/root-servers.db
```

Configuração do Servidor Secundário Esquinass2

```
tiroliroliro.esquina SP 10.0.16.12
tiroliroliro.esquina DD
tiroliroliro.esquina LG /var/dns/tiroliroliro-esquina.log
all LG /var/dns/all.log
root ST /var/dns/root-servers.db
```

Configuração do Servidor Primário Tirolirolirosp

```
tiroliroliro DB /var/dns/tiroliroliro.db
tiroliroliro SS 10.0.15.10
tiroliroliro SS 10.0.16.10
tiroliroliro LG /var/dns/tiroliroliro.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Configuração do Servidor Secundário Tirolirolirоss1

```
tiroliroliro SP 10.0.14.10
tiroliroliro LG /var/dns/tiroliroliro.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Configuração do Servidor Secundário Tirolirolirоss2

```
tiroliroliro SP 10.0.14.10
tiroliroliro LG /var/dns/tiroliroliro.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Configuração do Servidor Primário Concertinasp

```
tirolirolo.concertina DB /var/dns/tirolirolo-concertina.db
tirolirolo.concertina SS 10.0.17.11
tirolirolo.concertina SS 10.0.13.10
tirolirolo.concertina LG /var/dns/tirolirolo-concertina.log
all LG /var/dns/all.log
root ST /var/dns/root-servers.db
```

Configuração do Servidor Secundário Concertinass1

```
tirolirolo.concertina SP 10.0.12.11
tirolirolo.concertina LG /var/dns/tirolirolo-concertina.log
all LG /var/dns/all.log
root ST /var/dns/root-servers.db
```

Configuração do Servidor Secundário Concertinass2

```
tirolirolo.concertina SP 10.0.12.11
tirolirolo.concertina LG /var/dns/tirolirolo-concertina.log
all LG /var/dns/all.log
root ST /var/dns/root-servers.db
```

Configuração do Servidor Primário Tirolirolosp

```
tirolirolo DB /var/dns/tirolirolo.db
tirolirolo SS 10.0.15.11
tirolirolo SS 10.0.16.11
tirolirolo LG /var/dns/tirolirolo.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Configuração do Servidor Secundário Tiroliroloss1

```
tirolirolo SP 10.0.14.11
tirolirolo LG /var/dns/tirolirolo.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Configuração do Servidor Secundário Tiroliroloss2

```
tirolirolo SP 10.0.14.11
tirolirolo LG /var/dns/tirolirolo.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Configuração do Servidor SR

```
esquina.tiroliroliro DD 10.0.12.10
esquina.tiroliroliro LG /var/dns/esquina-tiroliroliro.log
all LG /var/dns/all.log
root ST /var/dns/root-servers.db
```

8. Ficheiros de Configuração de Bases de Dados

Reverse.db

@ DEFAULT reverse.

TTL DEFAULT 86400

```
@ SOASP ns1.reverse. TTL
@ SOAADMIN dns\.admin.reverse. TTL
@ SOASERIAL 0000000000 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS ns1.reverse. TTL
addr.reverse. NS sp.addr.reverse. TTL

ns1 A 10.0.14.12 TTL
sp.addr.reverse. A 10.0.15.12 TTL
sp CNAME ns1 TTL
```

Reverse-addr.db

```
@ DEFAULT addr.reverse.
TTL DEFAULT 86400
@ SOASP ns1.addr.reverse. TTL
@ SOAADMIN dns\.admin.addr.reverse. TTL
@ SOASERIAL 0000000000 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS ns1.addr.reverse. TTL

ns1 A 10.0.15.12 TTL

10.0.15.10 PTR ns2.tiroliroliro.
10.0.14.10 PTR ns1.tiroliroliro.
10.0.16.10 PTR ns3.tiroliroliro.

10.0.14.11 PTR ns1.tirolirolo.
```

```
10.0.15.11 PTR ns2.tirolirolo.
10.0.16.11 PTR ns3.tirolirolo.

10.0.12.11 PTR ns1.concertina.tirolirolo.
10.0.17.11 PTR ns2.concertina.tirolirolo.
10.0.13.10 PTR ns3.concertina.tirolirolo.

10.0.16.12 PTR ns1.esquina.tiroliroliro.
10.0.17.12 PTR ns2.esquina.tiroliroliro.
10.0.12.10 PTR ns3.esquina.tiroliroliro.

sp CNAME ns1 TTL
```

Concertina-tirolirolo.db

```
@ DEFAULT concertina.tirolirolo.
TTL DEFAULT 86400
@ SOASP ns1.concertina.tirolirolo. TTL
@ SOAADMIN dns\.admin.concertina.tirolirolo. TTL
@ SOASERIAL 0000000000 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS ns1.concertina.tirolirolo. TTL
@ NS ns2.concertina.tirolirolo. TTL
@ NS ns3.concertina.tirolirolo. TTL

@ MX mx1.concertina.tirolirolo. TTL
@ MX mx2.concertina.tirolirolo. TTL

ns1 A 10.0.12.11 TTL
ns2 A 10.0.17.11 TTL
ns3 A 10.0.13.10 TTL
```

```
mx1 A 10.0.12.200 TTL
mx2 A 10.0.12.250 TTL

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

Esquina-tiroliroliro.db

```
@ DEFAULT esquina.tiroliroliro.
TTL DEFAULT 86400
@ SOASP ns1.esquina.tiroliroliro. TTL
@ SOAADMIN dns\.admin.esquina.tiroliroliro. TTL
@ SOASERIAL 0000000000 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS ns1.esquina.tiroliroliro. TTL
@ NS ns2.esquina.tiroliroliro. TTL
@ NS ns3.esquina.tiroliroliro. TTL

@ MX mx1.esquina.tiroliroliro. TTL
@ MX mx2.esquina.tiroliroliro. TTL

ns1 A 10.0.16.12 TTL
ns2 A 10.0.17.12 TTL
ns3 A 10.0.12.10 TTL

mx1 A 10.0.16.200 TTL
mx2 A 10.0.16.250 TTL
```

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

Root.db

```
@ DEFAULT .
TTL DEFAULT 86400
@ SOASP st1. TTL
@ SOAADMIN dns\.admin. TTL
@ SOASERIAL 0000000000 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS st1. TTL
@ NS st2. TTL

tirolliroliro. NS sp.tirolliroliro.
tirolliroliro. NS ss1.tirolliroliro.
tirolliroliro. NS ss2.tirolliroliro.

tirolirolo. NS sp.tirolirolo.
tirolirolo. NS ss1.tirolirolo.
tirolirolo. NS ss2.tirolirolo.

reverse. NS sp.reverse.

sp.tirolliroliro. A 10.0.14.10 TTL
ss1.tirolliroliro. A 10.0.15.10 TTL
ss2.tirolliroliro. A 10.0.16.10 TTL

sp.tirolirolo. A 10.0.14.11 TTL
```

```
ss1.tirolirolo. A 10.0.15.11 TTL
ss2.tirolirolo. A 10.0.16.11 TTL

reverse A 10.0.14.12 TTL

sp CNAME st1 TTL
ss CNAME st2 TTL
```

Tiroliroliro.db

```
@ DEFAULT tiroliroliro.
TTL DEFAULT 86400
@ SOASP ns1.tiroliroliro. TTL
@ SOAADMIN dns\.admin.tiroliroliro. TTL
@ SOASERIAL 0000000000 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS ns1.tiroliroliro. TTL
@ NS ns2.tiroliroliro. TTL
@ NS ns3.tiroliroliro. TTL

esquina.@ NS sp.esquina.tiroliroliro.
esquina.@ NS ss1.esquina.tiroliroliro.
esquina.@ NS ss2.esquina.tiroliroliro.

ns1 A 10.0.14.10 TTL
ns2 A 10.0.15.10 TTL
ns3 A 10.0.16.10 TTL

sp.esquina A 10.0.16.12 TTL
ss1.esquina A 10.0.17.12 TTL
ss2.esquina A 10.0.12.10 TTL
```

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

Tirolirolo.db

```
@ DEFAULT tirolirolo.
TTL DEFAULT 86400
@ SOASP ns1.tirolirolo. TTL
@ SOAADMIN dns\.admin.tirolirolo. TTL
@ SOASERIAL 0000000000 TTL
@ SOAREFRESH 14400 TTL
@ SOARETRY 3600 TTL
@ SOAEXPIRE 604800 TTL

@ NS ns1.tirolirolo. TTL
@ NS ns2.tirolirolo. TTL
@ NS ns3.tirolirolo. TTL

concertina.@ NS sp.concertina.tiroliroliro.
concertina.@ NS ss1.concertina.tiroliroliro.
concertina.@ NS ss2.concertina.tiroliroliro.

ns1 A 10.0.14.11 TTL
ns2 A 10.0.15.11 TTL
ns3 A 10.0.16.11 TTL

sp.concertina A 10.0.12.11 TTL
ss1.concertina A 10.0.17.11 TTL
ss2.concertina A 10.0.13.10 TTL

sp CNAME ns1 TTL
ss1 CNAME ns2 TTL
```

```
ss2 CNAME ns3 TTL
```

9. Conclusão

Após a realização desta primeira fase do projeto, acreditamos que fomos capazes de implementar um sistema DNS eficiente e capaz de lidar com situações de erro de forma concisa. Analisando as nossas implementações, chegamos à conclusão que a implementação de *multithreading* foi uma decisão fulcral para o bom funcionamento do nosso programa, especialmente no que toca à performance e ao tempo de execução. Apesar disso, encontramos muitas dificuldades em termos da compreensão inicial dos requisitos pedidos no enunciado. Adicionalmente, a nossa proposta inicial de um ambiente de teste era muito suscetível a uma paragem total de funcionamento causado por um erro num servidor, o que nos levou a uma mudança de arquitetura nesta vertente.