Universidade do Minho - Dep.to Informática

Licenciatura em Engenharia Informática - 1º Semestre, 2022/2023

COMUNICAÇÕES POR COMPUTADOR

Ficha Trabalho Prático Nº2 – Implementação de Sistema DNS

Versão 5.0

Objetivos:

- Consolidação dos conhecimentos sobre o serviço DNS da arquitetura TCP/IP e sobre os protocolos de transporte UDP e TCP.
- Saber especificar um PDU e as primitivas de serviço dum protocolo aplicacional utilizando um protocolo de transporte orientado à conexão e um protocolo de transporte não orientado à conexão.
- Consolidar competências de programação de aplicações distribuídas utilizando o paradigma dos *sockets*.

Observações:

- O trabalho deverá ser realizado em cerca de 45 horas efetivas de trabalho individual.
- O trabalho é para ser realizado em grupos de dois ou três alunos. Se for realizado apenas por um aluno os objetivos serão adaptados adequadamente pela equipa docente.

AVISOS:

- Não serão tolerados atropelos aos direitos de autor de qualquer tipo de *software*...

Relatório & Código

Elabore o relatório do trabalho para ser entregue eletronicamente, juntamente com todas as especificações criadas, código produzido e todo o material que ache relevante entregar, antes da defesa do trabalho nas datas acordadas com os docentes e a direção de curso. Os ficheiros de código e respetivos ficheiros de configuração, *logs*, etc. devem ser submetidos num único ficheiro comprimido **CC-TP2-G<*Turno*>.<*Grupo*>-MATERIAL.zip** enquanto todos os outros documentos devem ser entregues em formato PDF e reunidos noutro ficheiro **CC-TP2-G<*Turno*>.<*Grupo*>-DOCUMENTOS.zip**

A primeira página do texto do relatório deve conter apenas, bem visível, a identificação do grupo e a identificação individual do(s) aluno(s), identificação do trabalho em questão, data da entrega, nome do curso e da unidade curricular.

Na secção introdutória devem ser abordados assuntos genéricos sobre o trabalho, discutidas decisões relevantes e estratégias genéricas que tenham guiado ou influenciado a realização posterior. Normalmente não se deve referir nesta secção aspetos de conceção mais detalhados ou de implementação.

Na segunda secção do relatório deve explicar-se a arquitetura do sistema, as funcionalidades de cada elemento e uma explicação introdutória do modelo de comunicação entre eles.

Na terceira secção do relatório deve detalhar-se o modelo comunicativo, explicando os PDU criados (incluindo a sua codificação precisa) e todas as formas de interação possíveis entre os elementos da arquitetura. Além disso, deve indicar-se todas as situações de erro e como o sistema lida com elas, incluindo ameaças de segurança.

Na quarta secção deve explicar-se as estratégias de implementação de cada elemento, as opções de execução e as funcionalidades dos seus componentes/processos, i.e., quais os requisitos funcionais

enumerados neste enunciado é que são implementados e como. Adicionalmente, devem indicar-se com detalhe adequado (incluindo a sintaxe) quais as atividades que são reportadas nos ficheiros de *log*.

A quinta secção 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 exaustiva do próprio ambiente de teste, devem ser incluídos num anexo próprio.

O texto do relatório deve conter ainda uma secção, antes das conclusões, que inclua uma tabela identificando as atividades desenvolvidas e o nível de participação de cada elemento do grupo nessas atividades.

Depois das conclusões deve aparecer a lista de referências e/ou bibliografia usada.

Por fim, noutro anexo devem ser incluídos os manuais de utilização dos componentes de software desenvolvidos.

O relatório não deve conter excertos do código. Por outro lado, o código deve estar bem documentado e comentado com relevância, incluindo a identificação da autoria, data de criação e data da última atualização, de todos os ficheiros.

Componentes do Sistema

Tendo em consideração as normas que especificam o sistema DNS da Internet, podemos identificar quatro tipos de elementos fundamentais que podem interagir: **Servidor Primário** (SP), **Servidor Secundário** (SS), **Servidor de Resolução** (SR) e **Cliente** (CL). Durante o projeto, os grupos terão de especificar e implementar componentes de software que implementem estes quatro elementos. Um componente de software pode executar em simultâneo mais do que um tipo de servidor. Por exemplo, um componente pode executar um ou mais SP, um ou mais SS e um ou mais SR. Por exemplo, um SP dum determinado domínio DNS também costuma ser o SP para o domínio reverso respetivo. Todos os tipos de servidores devem possuir a funcionalidade de *cache*.

- **SP** – Servidor DNS que responde a, e efetua, *queries* DNS e que tem acesso direto à base de dados dum domínio DNS, sendo a autoridade que o gere. Qualquer atualização necessária à informação dum domínio DNS tem de ser feita diretamente na base de dados do SP. Para além dos dados respeitantes diretamente ao domínio DNS, um SP tem de ter acesso a informação de configuração específica (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, endereços dos servidores de topo, etc.). Neste projeto, um SP tem 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; como *output* tem um ficheiro de *log*.

- **SS** – Servidor DNS que responde a, e efetua, *queries* DNS além de ter autorização e autoridade para possuir (e tentar manter atualizada) uma réplica da base de dados original do SP autoritativo dum domínio DNS. Um SS 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, endereços dos servidores de topo, etc.). Neste projeto, um SS tem como *input* um ficheiro de configuração e um ficheiro com a lista de servidores de topo; como *output* tem um ficheiro de *log*. Além disso,

por razões de segurança, foi decidido que a informação replicada do SP é armazenada apenas em memória volátil no SS.

- **SR** – Servidor DNS que responde a, e efetua, *queries* DNS sobre qualquer domínio, mas que não tem autoridade sobre nenhum pois serve apenas de intermediário. Um SR também é conhecido como *local DNS server, DNS resolver, DNS cache-only server, DNS forwarder*, etc. Um SR pode ser implementado a muitos níveis da rede, desde um processo em cada aplicação cliente, até um servidor DNS que responde aos clientes duma rede IP local, aos clientes dos provedores de serviços ou aos clientes duma instituição. É comum, hoje em dia, os próprios sistemas operativos incluírem um SR que pode ser contactado diretamente pelos programadores através de API específicas. Os SR podem funcionar em cascata, dependendo da gestão da configuração dos clientes, dos provedores de serviços e das instituições. Um SR tem de ter acesso a informação de configuração específica (eventuais domínios por defeito e lista dos servidores DNS que deve contactar, porta de atendimento, identificação do ficheiro de *log*, endereços dos servidores de topo, etc.). Neste projeto, um SR tem como *input* um ficheiro de configuração e um ficheiro com a lista de servidores de topo; como *output* tem um ficheiro de *log*.

- **CL** – Uma aplicação cliente de DNS é o processo que precisa da informação da base de dados de DNS dum determinado domínio (por exemplo, uma aplicação de *browser*, um cliente de e-mail, um cliente de ftp, etc.). Obtém essa informação realizando *queries* DNS a um SR (duma lista de SR predefinida). Normalmente, um CL tem uma lista de SR num ficheiro de configuração (endereços IP e portas de atendimento) ou usa o SR do próprio sistema operativo. Neste projeto será desenvolvido um CL específico para se consultar o DNS diretamente (à imagem do nslookup e outras aplicações semelhantes). O *input* e *output* deste CL será através da linha de comando sem necessidade dum ficheiro de configuração.

Para além destes tipos fundamentais de elementos, existem duas variantes especiais de servidores: os **Servidores de Topo** (ST, ou *DNS root servers*) e os **Servidores de Domínios de Topo** (SDT, ou *DNS Top-Level Domain servers*). Para o contexto deste projeto, o comportamento dos SDT é igual aos SP ou aos SS (ou seja, um SP ou SS autoritativos para um domínio de topo é um SDT) ainda que não tenham domínios hierarquicamente acima na árvore DNS. Por outro lado, os ST são como SP, mas têm apenas uma base de dados em que, para cada domínio de topo, inclui informação dos SDT respetivos (i.e. os nomes e os endereços IP dos seus SS e do seu SP). Portanto, só podem responder com estes dois tipos de informação. No contexto deste projeto, tanto os ST como os SDT devem ser implementados com o mesmo componente que implementa um SP ou um SS.

Qualquer componente de software desenvolvido pode ter parâmetros de funcionamento que sejam passados por argumentos na altura de arranque. No mínimo, os componentes devem suportar três parâmetros de funcionamento: um para indicar a porta de atendimento dos servidores quando esta for diferente da porta normalizada (53), outro que indica o valor de *timeout* quando se espera pela resposta a uma *query* e um terceiro que especifica se funciona em modo *debug* ou não. Neste modo, toda a atividade registada nos *logs* também deve ser enviada para o *standard output*.

Ficheiros do Sistema

Para este projeto são definidos alguns ficheiros de configuração, de dados e de *log* com uma sintaxe predefinida e que os grupos devem seguir com precisão. Os ficheiros de configuração são apenas lidos e processados no arranque do componente de software a que dizem respeito e moldam o seu comportamento. Os ficheiros de dados também são consultados apenas no arranque e a sua informação deve ser armazenada em memória. Se for necessário atualizar o comportamento dos servidores com informação modificada nos ficheiros de configuração ou de dados que lhes dizem respeito a única alternativa é reiniciar esses servidores.

Ficheiro de configuração dos servidores SP, SS e SR – Este ficheiro tem uma sintaxe com as seguintes regras (o componente deve saber reportar as situações de incoerência de configuração que possam resultar dum ficheiro de configuração ou sintaxes que não entenda; nesses casos o componente deve registar a informação nos *logs* respetivos e encerrar imediatamente a execução):

- As linhas começadas por '#' são consideradas comentários e são ignoradas;
- As linhas em branco também devem ser ignoradas;
- Deve existir uma definição de parâmetro de configuração por cada linha seguindo esta sintaxe: {*parâmetro*} {*tipo do valor*} {*valor associado ao parâmetro*}

Tipos de valores aceites (todas as referências a domínios, quer nos parâmetros quer nos valores, são considerado nomes completos):

- DB – o valor indica o ficheiro da base de dados com a informação do domínio indicado no parâmetro (o servidor assume o papel de SP para este domínio);
- SP – o valor indica o endereço IP[:porta] do SP do domínio indicado no parâmetro (o servidor assume o papel de SS para este domínio);
- SS – o valor indica o endereço IP[:porta] dum SS do domínio indicado no parâmetro (o servidor assume o papel de SP para este domínio) e que passa a ter autorização para pedir a transmissão da informação da base de dados (transferência de zona); podem existir várias entradas para o mesmo parâmetro (uma por cada SS do domínio);
- DD – o valor indica o endereço IP[:porta] dum SR, dum SS ou dum SP do domínio por defeito indicado no parâmetro; quando os servidores que assumem o papel de SR usam este parâmetro é para indicar quais os domínios para os quais devem contactar diretamente os servidores indicados se receberem *queries* sobre estes domínios (quando a resposta não está em *cache*), em vez de contactarem um dos ST; podem existir várias entradas para o mesmo parâmetro (uma por cada servidor do domínio por defeito); quando os servidores que assumem o papel de SP ou SS usam este parâmetro é para indicar os únicos domínios para os quais respondem (quer a resposta esteja em *cache* ou não), i.e., nestes casos, o parâmetro serve para restringir o funcionamento dos SP ou SS a responderem apenas a *queries* sobre os domínios indicados neste parâmetro;
- ST – o valor indica o ficheiro com a lista dos ST (o parâmetro deve ser igual a "*root*");
- LG – o valor indica o ficheiro de *log* que o servidor deve utilizar para registar a atividade do servidor associada ao domínio indicado no parâmetro; só podem ser indicados domínios para o qual o servidor é SP ou SS; tem de existir pelo menos uma entrada a referir um ficheiro de *log* para toda a atividade que não seja diretamente referente aos domínios especificados noutras entradas LG (neste caso o parâmetro deve ser igual a "*all*").

Segue-se um exemplo dum ficheiro de configuração dum SP dum domínio `example.com`:

```
# Configuration file for primary server for example.com
example.com DB /var/dns/example-com.db
example.com SS 193.137.100.250
example.com SS 193.136.130.251:5353
example.com DD 127.0.0.1
example.com LG /var/dns/example-com.log
all LG /var/dns/all.log
root ST /var/dns/rootservers.db
```

Exemplo 1: Ficheiro de configuração dum SP dum domínio `example.com`.

Ficheiro com a lista de ST – Este ficheiro tem a lista de ST que devem ser contactados sempre que necessário (quando se quer saber informação sobre domínios acima do domínio atual e a resposta não

está em *cache*); deve existir um endereço IP[:porta] ST por cada linha do ficheiro (as linhas começadas por '#' ou em branco devem ser ignoradas).

Ficheiros de *log* – Estes ficheiros registam toda a atividade relevante do componente; deve existir uma entrada de log por cada linha do ficheiro; sempre que um componente arranca este deve verificar a existência dos ficheiros de log indicados no seu ficheiro de configuração; se não existir algum deve ser criado e se já existirem as novas entradas devem ser registadas a partir da última entrada já existente no ficheiro; a sintaxe de cada entrada é a seguinte (a etiqueta temporal é a data e hora completa do sistema operativo na altura em que aconteceu a atividade registada e não a data e hora em que foi registada):
{*etiqueta temporal*} {*tipo de entrada*} {*endereço IP[:porta]*} {*dados da entrada*}

Tipos de entradas aceites:

- QR/QE – foi recebida/enviada uma *query* do/para o endereço indicado; os dados da entrada devem ser os dados relevantes incluídos na *query*; 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 – foi enviada/recebida uma resposta a uma *query* para o/do endereço indicado; os dados da entrada devem ser os dados relevantes incluídos na resposta à *query*; 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 – foi iniciado e concluído corretamente um processo de transferência de zona; o endereço deve indicar o servidor na outra ponta da transferência; os dados da entrada devem indicar qual o papel do servidor local na transferência (SP ou SS) e, opcionalmente, a duração em milissegundos da transferência e o total de bytes transferidos;
- 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 descodificar corretamente; opcionalmente, os dados da entrada podem ser usados para indicar informação adicional (como, por exemplo, o que foi possível descodificar corretamente e em que parte/byte aconteceu o erro);
- EZ – foi detetado um erro num processo de transferência de zona que não foi concluída corretamente; o endereço deve indicar o servidor na outra ponta da transferência; os dados da entrada devem indicar qual o papel do servidor local na transferência (SP ou SS);
- FL – foi 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 descodificação 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*).

Segue-se um excerto exemplificativo dum ficheiro genérico de *log* dum componente servidor:

```
# Log file for DNS server/resolver
19:10:2022.11:20:50:020 ST 127.0.0.1 53 20000 shy
19:10:2022.11:20:50:210 EV @ conf-file-read /var/dns/SP.config
19:10:2022.11:20:50:915 EV @ log-file-create /var/dns/all.log
19:10:2022.11:20:51:783 EV @ db-file-read example-com.db
19:10:2022.11:22:38:201 QR 192.144.99.250 [...]
```

Exemplo 2: Excerto dum ficheiro genérico de *log* dum componente servidor.

Ficheiro de dados do SP – Este ficheiro tem uma sintaxe com as seguintes regras (o SP deve saber reportar as situações de incoerência nos valores dos parâmetros ou de sintaxe não compreendida):

- As linhas começadas por '#' são consideradas comentários e são ignoradas;
- As linhas em branco também devem ser ignoradas;
- Deve existir uma definição de parâmetro de dados por cada linha seguindo esta sintaxe: {*parâmetro*} {*tipo do valor*} {*valor*} {*tempo de validade*} {*prioridade*}*

O tempo de validade (TTL) é o tempo máximo em segundos que os dados podem existir numa cache dum servidor (tanto serve para cache normal como para cache negativa, se for suportada). Quando o TTL não é suportado num determinado tipo, o seu valor deve ser igual a zero, mas é de indicação obrigatória. O valor da prioridade é o único campo de indicação opcional.

O campo da prioridade é um valor inteiro menor que 256 e que define uma ordem de prioridade de vários valores associados ao mesmo parâmetro. Quanto menor o valor maior a prioridade. Para parâmetros com um único valor ou para parâmetros em que todos os valores têm a mesma prioridade, o campo não deve existir. Este campo ajuda a implementar sistema de *backup* de serviços/hosts quando algum está em baixo (números de prioridade entre 0 e 127) ou balanceamento de carga de serviços/*hosts* que tenham vários servidores/*hosts* aplicacionais disponíveis (números de prioridade entre 128 e 255).

Os nomes completos de e-mail, domínios, servidores e hosts devem terminar com um '.' (exemplo de nome completo de domínio: `example.com.`). Quando os nomes não terminam com '.' subentende-se que são concatenados com um prefixo por defeito definido através do parâmetro @ do tipo DEFAULT.

Tipos de valores a suportar (os tipos marcados com '*' são de implementação opcional e os tipos de valores que devem suportar o campo da prioridade são indicados explicitamente):

- 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 valor indica o nome completo do SP do domínio (ou zona) indicado no parâmetro;
- SOAADMIN – o valor indica o endereço de e-mail completo do administrador do domínio (ou zona) indicado no parâmetro; o símbolo '@' deve ser substituído por '.' e '.' no lado esquerdo do '@' devem ser antecedidos de '\';
- SOASERIAL – o valor indica o número de série da base de dados do SP do domínio (ou zona) indicado no parâmetro; sempres que a base de dados é alterada este número deve ser incrementado;
- SOAREFRESH – o valor indica o intervalo temporal em segundos para um SS perguntar ao SP do domínio indicado no parâmetro qual o número de série da base de dados dessa zona;
- 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 valor indica o intervalo temporal para um SS deixar de considerar a sua réplica da base de dados da zona indicada no parâmetro como válida, deixando de responder a perguntas sobre a zona em causa, mesmo que continue a tentar contactar o SP respetivo;
- NS – o valor indica o nome dum servidor que é autoritativo para o domínio indicado no parâmetro, ou seja, o nome do SP ou dum dos SS do domínio; este tipo de parâmetro suporta prioridades;
- A – o valor indica o endereço IPv4 dum *host*/servidor indicado no parâmetro como nome; este tipo de parâmetro suporta prioridades;
- CNAME – o valor indica um nome canónico (ou *alias*) associado ao nome indicado no parâmetro; um nome canónico não deve apontar para um outro nome canónico nem podem existir outros parâmetros com o mesmo valor do nome canónico;
- MX – o valor indica o nome dum servidor de e-mail para o domínio indicado no parâmetro; este tipo de parâmetro suporta prioridades;
- PTR – o valor indica o nome dum servidor/*host* que usa o endereço IPv4 indicado no parâmetro; a indicação do IPv4 é feita como nos domínios de DNS reverso (rDNS) quando se implementa *reverse-mapping*;

Segue-se um exemplo dum ficheiro de base de dados DNS dum SP do domínio `example.com`:

```
# DNS database file for domain example.com
# It also includes a pointer to the primary server
# of the smaller.example.com subdomain

@ DEFAULT example.com.
TTL DEFAULT 86400

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

@ NS ns1.example.com. TTL
@ NS ns2.example.com. TTL
@ NS ns3.example.com. TTL

smaller.@ NS sp.smaller.example.com. TTL

@ MX mx1.example.com TTL 10
@ MX mx2.example.com TTL 20

ns1 A 193.136.130.250 TTL
ns2 A 193.137.100.250 TTL
ns3 A 193.136.130.251 TTL
sp.smaller A 193.140.90.11 TTL
mx1 A 193.136.130.200 TTL
mx2 A 193.136.130.201 TTL
www A 193.136.130.80 TTL 200
www A 193.136.130.81 TTL 200
ftp A 193.136.130.20 TTL

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

Exemplo 3: Ficheiro de base de dados DNS dum SP do domínio `example.com`.

Modelo Comunicacional do Sistema

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). 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.

Ver a Figura 1 com a representação lógica duma mensagem DNS usada neste sistema.

DNS Message

HEADER	DATA

HEADER Fields

MESSAGE ID*	FLAGS*	RESPONSE CODE	NUMBER OF VALUES	NUMBER OF AUTHORITIES	NUMBER OF EXTRA VALUES

DATA Fields

QUERY INFO*	RESPONSE VALUES	AUTHORITIES VALUES	EXTRA VALUES

QUERY INFO Fields

NAME*	TYPE OF VALUE*

Figura 1: A representação lógica da Mensagem de DNS usada no sistema.

Todas as interações começam com o envio duma *query* DNS para um servidor. Essa *query* vem dum CL ou dum outro qualquer servidor DNS. A *query* é transportada numa mensagem DNS. Na mensagem da *query* só são usados os campos assinalados com '*'. Os restantes campos do cabeçalho são ignorados (campos devem ser colocados a zero) e os restantes campos dos dados são nulos (não são sequer incluídos) na mensagem.

Um servidor deve processar a *query* recebida e se a descodificação da informação da *query* for correta o servidor deve tentar encontrar informação direta que responda à query (i) na sua *cache* ou (ii) na sua base de dados (se for um servidor autoritativo para o domínio do `NAME`). No caso (i) e o servidor não encontra resposta direta à *query* (i.e., não encontrou na *cache* uma entrada com `NAME` e valores do tipo `TYPE OF VALUE`) então deve reenviar a *query* para um SDT que seja o servidor do domínio de topo incluído no `NAME` (ou tem a informação desse SDT em cache ou tem de a obter enviando uma *query* a um ST). O processo continua duma forma iterativa ou recursiva (seguindo os mesmos dois tipos de operação da norma DNS) até o servidor obter uma resposta final, i.e., uma resposta em que já obteve informação direta à *query* ou uma resposta em que, a partir dali, não é possível obter mais informações diretas sobre a *query*.

Os campos do cabeçalho devem ser implementados de tal forma que:

- `MESSAGE ID` – identificador de mensagem (número inteiro entre 1 e 65535, gerado aleatoriamente pelo CL ou servidor que faz a *query* original) que irá ser usado para relacionar as respostas recebidas com a *query* original;
- `FLAGS` – devem ser suportadas as *flags* Q, R e A; a flag Q ativa indica que a mensagem é uma *query*, senão é uma resposta a uma *query*; se a *flag* R estiver ativa na *query* indica que se deseja que o processo opere de forma recursiva e não iterativa (que é o modo por defeito); se a *flag* R estiver ativa na resposta indica que o servidor que respondeu suporta o modo recursivo;

se a *flag* A estiver ativa na resposta indica que a resposta é autoritativa (o valor da *flag* A é ignorada nas queries originais);

- `RESPONSE CODE` – indica o código de erro na resposta a uma *query*; se o valor for zero então não existe qualquer tipo de erro e a resposta contém informação que responde diretamente à *query*; a resposta deve ser guardada em *cache*; se houver erros, o sistema deve suportar os seguintes códigos de erro: 1, o domínio incluído em `NAME` existe mas não foi encontrada qualquer informação direta com um tipo de valor igual a `TYPE OF VALUE` (o campo de resultados vem vazio mas o campo com a lista de valores de autoridades válidas para o domínio e o campo com a lista de valores com informação extra podem ser incluídos); este caso é identificado como resposta negativa e pode ser guardada em *cache* para que a um pedido semelhante, num horizonte temporal curto, o servidor possa responder mais rapidamente; 2, o domínio incluído em `NAME` não existe (o campo de resultados vem vazio mas o campo com a lista de valores de autoridades válidas onde a resposta foi obtida e o campo com a lista de valores com informação extra podem ser incluídos); este caso também é identificado como resposta negativa e pode ser guardada em *cache*; 3, a mensagem DNS não foi descodificada corretamente;
- `NUMBER OF VALUES` – número de entradas relevantes (num máximo de 255) que respondem diretamente à *query* (i.e., as entradas na *cache* ou na base de dados do servidor autoritativo e que têm um parâmetro igual a `NAME` e um tipo de valor igual a `TYPE OF VALUE`) e que fazem parte da lista de entradas incluídas no campo `RESPONSE VALUES`;
- `NUMBER OF AUTHORITIES` – número de entradas (num máximo de 255) 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;
- `QUERY INFO` – informação do parâmetro da query (`NAME`) e o tipo de valor associado ao parâmetro (`TYPE OF VALUE`); os tipos suportados são os mesmos suportados na sintaxe dos ficheiros de base de dados dos SP; na resposta a queries, os servidores devem copiar a informação do `QUERY INFO` e incluí-la na mensagem de resposta;
- `RESPONSE VALUES` – lista das entradas que fazem *match* no `NAME` e `TYPE OF VALUE` incluídos na *cache* ou na base de dados do servidor autoritativo; cada entrada deve ter a informação completa tal como é definida na base de dados DNS do SP do domínio referente ao `NAME`;
- `AUTHORITIES VALUES` – lista das entradas que fazem *match* com o `NAME` e com o tipo de valor igual a NS incluídos na *cache* ou na base de dados do servidor autoritativo; cada entrada deve ter a informação completa tal como é definida na base de dados DNS do SP do domínio referente ao `NAME`;
- `EXTRA VALUES` – lista das entradas do tipo A (incluídos na *cache* ou na base de dados do servidor autoritativo) e que fazem *match* no parâmetro com todos os valores no campo `RESPONSE VALUES` e no campo `AUTHORITIES VALUES` de forma a que o elemento que o CL ou servidor que recebe a resposta não tenha que fazer novas *queries* para saber os endereços IP dos parâmetros que vêm como valores nos outros dois campos; cada entrada deve ter a informação completa tal como é definida na base de dados DNS do SP do domínio referente ao `NAME`.

Segue-se um exemplo de interação assíncrona entre um CL e o SP do exemplo 3, que é autoridade do domínio/zona `example.com`. Mais concretamente, o CL quer saber quais são os servidores de email do domínio `example.com` e envia a seguinte *query* para o servidor SP (que por acaso é o servidor DNS por defeito na configuração do CL):

```
# Header
MESSAGE-ID = 3874, FLAGS = Q+R, RESPONSE-CODE = 0,
N-VALUES = 0, N-AUTHORITIES = 0, N-EXTRA-VALUES = 0,;
# Data: Query Info
QUERY-INFO.NAME = example.com., QUERY-INFO.TYPE = MX,;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = (Null)
AUTHORITIES-VALUES = (Null)
EXTRA-VALUES = (Null)
```

Exemplo 4: *Query* enviada do CL para o SP.

Ou, num formato mais conciso (que deve ser também utilizado nas entradas dos ficheiros de *log* e no modo comunicacional de *debug*):

```
3874,Q+R,0,0,0,0;example.com.,MX;
```

Exemplo 5: *Query* enviada do CL para o SP (formato conciso).

O servidor processa a informação da *query*, verifica se a descodificação é correta, consulta a sua base de dados da zona/domínio `example.com` e seleciona a informação para construir a seguinte resposta:

```
# Header
MESSAGE-ID = 3874, FLAGS = R+A, RESPONSE-CODE = 0,
N-VALUES = 2, N-AUTHORITIES = 3, N-EXTRA-VALUES = 5;
# Data: Query Info
QUERY-INFO.NAME = example.com., QUERY-INFO.TYPE = MX;
# Data: List of Response, Authorities and Extra Values
RESPONSE-VALUES = example.com. MX mx1.example.com 86400 10,
RESPONSE-VALUES = example.com. MX mx2.example.com 86400 20;
AUTHORITIES-VALUES = example.com. NS ns1.example.com. 86400,
AUTHORITIES-VALUES = example.com. NS ns2.example.com. 86400,
AUTHORITIES-VALUES = example.com. NS ns3.example.com. 86400;
EXTRA-VALUES = mx1.example.com. A 193.136.130.200 86400,
EXTRA-VALUES = mx2.example.com. A 193.136.130.201 86400,
EXTRA-VALUES = ns1.example.com. A 193.136.130.250 86400,
EXTRA-VALUES = ns2.example.com. A 193.137.100.250 86400,
EXTRA-VALUES = ns3.example.com. A 193.136.130.251 86400;
```

Exemplo 6: Resposta enviada do SP para o CL.

Ou, num formato mais conciso (nas entradas dos ficheiros de *log* e na informação de *debug* não deve haver separação da informação em várias linhas; aqui foi feito para facilitar a leitura):

```
3874,R+A,0,2,3,5;example.com.,MX;
example.com. MX mx1.example.com 86400 10,
example.com. MX mx2.example.com 86400 20;
example.com. NS ns1.example.com. 86400,
example.com. NS ns2.example.com. 86400,
example.com. NS ns3.example.com. 86400;
mx1.example.com. A 193.136.130.200 86400,
mx2.example.com. A 193.136.130.201 86400,
ns1.example.com. A 193.136.130.250 86400,
ns2.example.com. A 193.137.100.250 86400,
ns3.example.com. A 193.136.130.251 86400;
```

Exemplo 7: Resposta enviada do SP para o CL (formato conciso).

Os elementos do sistema devem ser capazes de comunicar utilizando estas mensagens DNS e devem seguir o comportamento esperado pelos mesmos tipos de elementos dum sistema DNS real.

Relembra-se que, no modo de funcionamento *debug* dos componentes do sistema, as mensagens DNS a serem transmitidas são simplesmente uma sequência de carateres (*string*) tal como representado nos exemplos 5 e 7. Estas sequências devem ser também usadas nas entradas dos ficheiros de *log* sempre que for necessário registar o envio ou a receção de *queries* e de respostas.

No modo normal de funcionamento dos componentes as mensagens DNS têm que ser codificadas em binário duma forma eficiente para uma mensagem ocupar o menor espaço possível. As entradas nos ficheiros de *log* devem continuar a utilizar o mesmo formato do modo *debug*.

Cabe aos grupos definir (e explicar adequadamente no relatório) uma sintaxe de codificação eficiente.

O modo *debug* é obrigatório implementar e o modo normal é de implementação opcional.

No contexto deste trabalho, a implementação do modo recursivo (correspondente à ativação da *flag* R) também não é de implementação obrigatória.

Segue-se um exemplo de resposta dum ST a uma *query* semelhante à do exemplo 5, mas feita em modo iterativo. O ST, neste caso, indica que não tem a resposta direta à *query,* mas refere os servidores autoritativos do domínio da *query* que devem ser contactados para prosseguir o processo iterativo de obtenção da resposta à *query*.

```
# Header
MESSAGE-ID = 3874, FLAGS = A, RESPONSE-CODE = 1,
N-VALUES = 0, N-AUTHORITIES = 3, N-EXTRA-VALUES = 3;
# Data: Query Info
QUERY-INFO.NAME = example.com., QUERY-INFO.TYPE = MX;
# Data: List of Response, Authorities and Extra Values
# No direct response for this domain on this server
;
AUTHORITIES-VALUES = example.com. NS ns1.example.com. 86400,
AUTHORITIES-VALUES = example.com. NS ns2.example.com. 86400,
AUTHORITIES-VALUES = example.com. NS ns3.example.com. 86400;
EXTRA-VALUES = ns1.example.com. A 193.136.130.250 86400,
EXTRA-VALUES = ns2.example.com. A 193.137.100.250 86400,
EXTRA-VALUES = ns3.example.com. A 193.136.130.251 86400;
```

Exemplo 8: Resposta enviada pelo ST a um servidor, modo iterativo.

Transferência de Zona

Todas as interações numa operação de transferência de zona devem ser feitas utilizando uma conexão TCP. Um SS utiliza *queries* normais para saber se a versão da base de dados do respetivo SP é mais atual que a sua. Se for, o SS 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 verifica a validade do domínio e que aquele 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 (valor máximo de 65535; as linhas de comentário e linhas em branco não são contadas nem transmitidas), Depois do SS aceitar receber essa quantidade de entradas (respondendo 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 (acrescenta um número de ordem da entrada antes da entrada propriamente dita). O SS vai 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. Um SP também não deve aceitar pedidos consecutivos de transferência de zona do mesmo SS com intervalo menor de SOARETRY segundos.

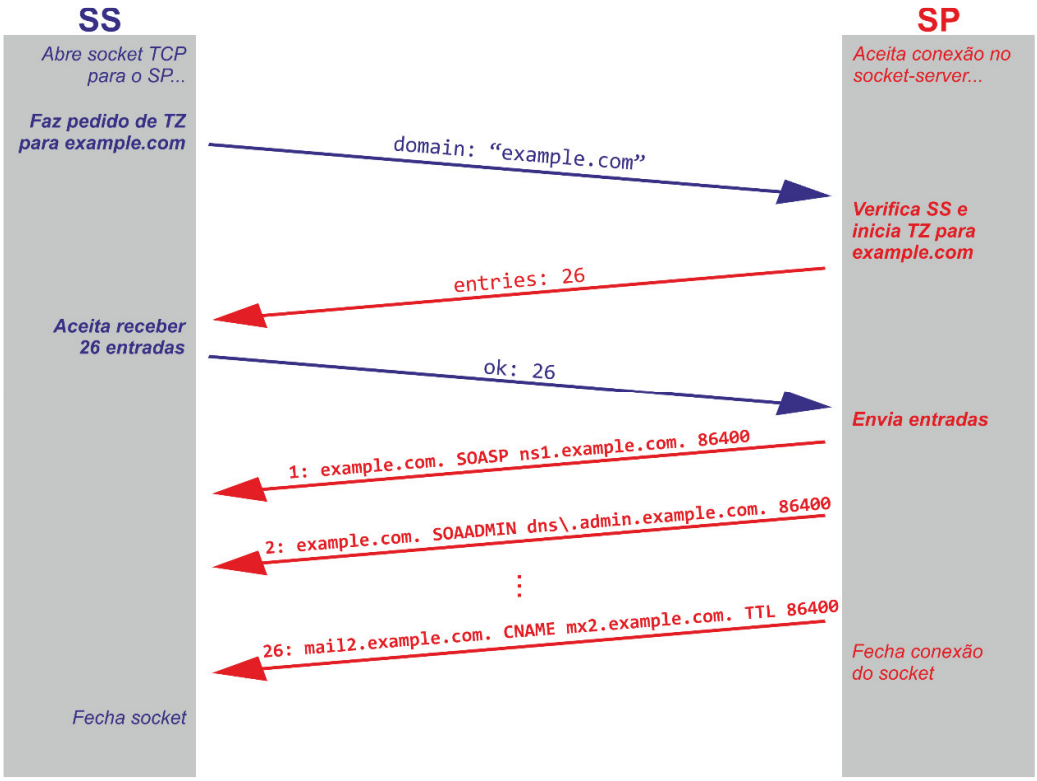

Figura 2: Exemplo de transferência de zona.

Não é preciso definir qualquer tipo de codificação mais eficiente para as transferências de zona. Cabe aos grupos explicar os detalhes da implementação do processo de transferência de zona, tanto nos SS como nos SP.

A Figura 2 apresenta um exemplo de interação entre um SS e um SP numa transferência de zona.

Quando um servidor se inicializa e tenta imediatamente uma transferência de zona para os domínios em que é SS, ainda não sabe o valor temporal SOARETRY para esses domínios até a transferência de zona terminar com sucesso. Assim, quando a primeira transferência de zona não termina com sucesso, o SS deve tentar novamente, em ciclo infinito, ao fim dum tempo definido pelo próprio *software* do servidor (fica a cargo dos grupos definirem este tempo por defeito ou por argumento do *software*). Depois da primeira transferência de zona com sucesso, o SS deve assumir o comportamento normal, respeitando os valores temporais SOA* desse domínio.

Funcionamento do CL

Todos os elementos (ou componentes) do sistema precisam implementar processos idênticos de leitura de ficheiros de configuração (ainda que cada um tenha o seu tipo de ficheiro de configuração, a sintaxe dos ficheiros é muito semelhante) e de processos comunicacionais assíncronos utilizando *queries* e respostas a *queries*. Todos os elementos limitam o seu *input* a ficheiros de configuração ou de dados e o seu *output* a ficheiros de *log* (em modo *debug* também é feita uma cópia das entradas dos ficheiros de log para a *interface* de *standard output*).

No entanto, no caso da aplicação CL, não são 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 é o único que suporta 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 servirá, assim, para indagar um serviço DNS. Para isso, e em primeiro lugar, 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` duma query DNS, i.e., o nome completo do parâmetro `NAME` e o tipo de valor esperado `TYPE OF VALUE`. Opcionalmente, o utilizador pode indicar se pretende tentar o tipo de *query* recursiva (correspondente à ativação da *flag* R); por defeito, o CL utiliza *queries* no modo iterativo e não ativa a *flag* R.

Com as informações recolhidas pelo utilizador, 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*. No mínimo, os resultados da *query* devem ser mostrados ao utilizador utilizando uma sintaxe/formatação semelhante à utilizada nas entradas dos ficheiros de *log*, i.e., no formato conciso dos exemplos 5 e 7. Opcionalmente, pode ser utilizado um formato mais amigo do utilizador como o usado nos exemplos 4 e 6.

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 carateres conciso como o dos exemplos 5 e 7. Numa implementação opcional, o CL pode funcionar em modo normal e utilizar a codificação binária das mensagens DNS que tenha sido definida para os outros elementos/componentes do sistema. De qualquer forma, o modo normal ou *debug* não afeta o formato com que a informação é disponibilizada ao utilizador.

Segue-se um exemplo dum comando de execução dum CL que tenta obter informações do SP (com uma base de dados definida como a do exemplo 3) sobre os servidores de e-mail do domínio `example.com`:

```
bash> dnscl 193.136.130.250 example.com. MX R
```

Exemplo 8: Programa a implementar um CL no sistema.

De notar que, no exemplo 8, o nome de domínio está completamente indicado pois o argumento será passado como `NAME` na *query* DNS, pelo que o uso do ponto final é necessário.

O resultado do programa `dnscl` que implementa um CL no exemplo 8, deve ser, no mínimo, igual ao do exemplo 7, ou, melhor ainda, igual ao do exemplo 6, desde que o CL implementado suporte esse formato mais amigo do utilizador.

Sistemas de *cache*

Os sistemas de *cache* (ou de *caching*) são uma funcionalidade indispensável para manter o desempenho do serviço de DNS atual. Praticamente todos os componentes dum sistema DNS fazem *caching* positivo (quando as respostas disponibilizam os resultados pedidos) e negativo (quando as respostas informam que uma determinada informação não existe). É importante responder rapidamente a um pedido, mesmo quando a resposta é negativa!

Os sistemas de cache são mais necessários em servidores não autoritativos. Os SR são servidores que não são autoritativos para nenhum domínio pelo que nesses tipos de servidores é essencial a implementação de sistemas eficientes de *caching*. É relativamente simples construir sistemas eficientes de *caching* em memória volátil, um pouco mais complexo se for para implementação em memória não volátil.
No contexto deste trabalho, espera-se apenas a implementação dum sistema simples de *caching* positivo em memória volátil nos servidores (a implementação é a mesma para qualquer tipo de servidor). De notar que o registo de informação em *cache* deve ser efetuado sempre que se receber uma resposta recebida a uma *query* feita anteriormente. O envio de queries não está relacionado com nenhuma atividade de *caching*.

Os CL deste trabalho não precisam de qualquer tipo de *cache*.

Os grupos devem explicar o seu processo de implementação de *caching* positivo, incluindo as estruturas de dados usadas.

Especificação base dum sistema de *cache*

A funcionalidade de *cache* deve ser especificada e implementada num módulo independente que possa ser adicionada a todos os servidores. Esta *cache* serve para todos os tipos de servidores implementarem a sua base de dados de informação sobre entradas DNS, não sendo necessária mais nenhuma estrutura de dados adicional.

Seguem-se algumas considerações que podem ajudar na sua implementação:

- A *cache* pode ser implementada à custa duma estrutura de dados equivalente a uma tabela com nove colunas/campos, em que cada linha/posição representa uma entrada completa (ou *DNS Resource Record*). Os nove campos/colunas são: (a) parâmetro/*Name*, (b) tipo do valor/*Type*, (c) valor/Value, (d) tempo de validade/TTL , (e) prioridade/*Order*, (f) origem da entrada/*Origin* (FILE, SP, OTHERS), (g) tempo em segundos que passou desde o arranque do servidor até ao momento em que a entrada foi registada na *cache*/*TimeStamp*, (h) número da entrada/*Index* (as entradas válidas são numeradas/indexadas de 1 até N, em que N é o número total de entradas na *cache*, incluindo as entradas livres/FREE) e (i) estado da entrada/*Status* (FREE, VALID).
- Quando o servidor arranca a *cache* deve ser inicializada com, pelo menos, uma entrada/posição FREE (se o tamanho da cache for gerido dinamicamente, o que não é relevante no contexto do trabalho) ou então com N entradas FREE (se o tamanho é fixo a N entradas).
- Deve ser implementada uma função para procurar uma entrada VALID a partir de três valores *Index*, *Name* e *Type*. A função deve devolver o índice da primeira entrada encontrada que faça o *match* com *Name* e *Type*. A procura deve começar a partir da entrada com o índice *Index*. Sempre que é necessário encontrar todas as entradas que façam *match* ao tuplo [*Name*,*Type*] deve começar-se com *Index*=1 e repetir-se até o índice devolvido ser igual a N+1. Esta função deve também verificar se as entradas visitadas com *Origin* igual a OTHERS estão expiradas (i.e., se o tempo que já passou desde o momento do registo da entrada guardado em *TimeStamp* for superior a TTL). Nesse caso deve colocar-se essas entradas como FREE. Assim, a função de procura serve também para atualizar as entradas da *cache* e desativar as entradas expiradas.
- Deve ser implementada uma função para registar/atualizar uma entrada na *cache*. Se o campo *Origin* da entrada for igual a FILE ou SP, então deve ser registada a entrada na primeira posição FREE da *cache*, com o *TimeStamp* calculado no momento do registo e com *Status* igual a VALID. Se o campo *Origin* da entrada a registar for igual a OTHERS, primeiro é necessário procurar se a entrada completa já existe ou não na cache (i.e., se faz o *match* dos campos *Name*, *Type*, *Value, TTL,* e *Order*): (i) se já existir e o campo *Origin* da entrada existente for igual a FILE ou SP, ignora-se o pedido de registo; (ii) se já existir e o campo *Origin* da entrada existente for igual a

OTHERS atualiza-se o campo *TimeStamp* com o valor calculado no momento e atualiza-se o campo *Status* para VALID; (iii) se não existir, então regista-se a nova entrada na última posição FREE, com o *TimeStamp* calculado no momento e com *Status* igual a VALID.

- Quando um SP arranca deve registar na *cache* todas as entradas dos seus ficheiros de bases de dados dos domínios para o qual é primário, utilizando repetidamente a função anterior com o campo *Origin* igual a FILE.
- Quando um SS arranca deve fazer uma transferência de zona dos SP respetivos e deve registar na sua *cache* todas as entradas recebidas do SP, utilizando repetidamente a função anterior com o campo *Origin* igual a SP.
- Quando uma entrada é registada ou atualizada na *cache* deve ser gerada uma entrada correspondente no ficheiro de *logs* (se o modo de funcionamento for de *debug* essa entrada deve também ser impressa no *standard output*).
- Deve ser implementada uma função que atualize, em todas as entradas da *cache* com *Name* igual ao domínio passado como argumento, o campo *Status* para FREE. Quando o temporizador associado à idade da base de dados dum SS relativo a um domínio atinge o valor de SOAEXPIRE, então o SS deve executar esta função para esse domínio. Esta função é exclusiva dos servidores do tipo secundário.
- Não é importante os grupos terem preocupações de otimização destas funções de procura, registo e atualização da *cache*, quer em termos de velocidade de execução quer em termos de ocupação de memória.
- Todas as respostas a *queries* devem ser procuradas na *cache*, independentemente do tipo de servidor, uma vez que a cache armazena todas os *resource records* disponíveis num servidor.

Esta especificação só implementa *cache* positiva, tal como requerido no enunciado.

Ambiente de Teste

Para testar os componentes do sistema desenvolvido, os grupos terão de instalar um ambiente de teste em que possam definir servidores de topo, domínios de topo e subdomínios dos domínios de topo de forma experimental, i.e., sem qualquer relação com endereços e nomes do mundo real. Para isso devem instalar e configurar os seus componentes (SP, SS, SR e CL) no *setup* de configuração do Core disponibilizado pelos docentes de tal forma que, no mínimo, tenham:

- Dois ST (estes servidores só precisam de incluir no ficheiro de base de dados entradas para todos os SDT do sistema);
- Os SDT para dois domínios de topo (os nomes de domínio devem ser inventados e não podem corresponder a qualquer nome de domínio existente na realidade nem podem ser iguais a nomes de domínios usados por outros grupos de trabalho – os docentes verificarão os nomes propostos pelos grupos para garantir a criatividade e originalidade dos nomes);
- Um domínio de topo nomeado de `.reverse` onde estarão pendurados os domínios de DNS reverso;
- Um subdomínio em cada domínio de topo (os nomes de subdomínios devem ser inventados e não podem corresponder a qualquer nome de subdomínio existente na realidade nem podem ser iguais a nomes de subdomínios usados por outros grupos de trabalho – os docentes verificarão os nomes propostos pelos grupos para garantir a criatividade e originalidade dos nomes);
- Para cada domínio de topo e seus subdomínios devem instalar e configurar um SP e dois SS;
- Num dos subdomínios devem instalar um SR que, pelo menos, implemente *caching* positivo;
- Numa host qualquer instalar o programa que implementa o CL;
- Para um dos domínios de topo devem instalar-se e configurar-se os respetivos servidores SP e SS para implementar o respetivo *reverse mapping*;

- Para cada domínio e subdomínio também devem ser incluídas nos ficheiros de bases de dados DNS informações referentes a servidores de email e endereços e nomes canónicos para vários *hosts*;
- Deve ser utlizada uma porta privada como porta de atendimento de todos os servidores do sistema e que; utilizar a mesma porta *standard* do DNS pode trazer problemas de execução dos componentes no Core.

Através da aplicação/programa que implementa o CL deverá ser possível obter informações do sistema DNS a partir dum qualquer servidor instalado, em particular a partir do SR.

Por defeito, todos os componentes devem funcionar em modo *debug* por forma a ser mais fácil verificar e corrigir o funcionamento do sistema, incluindo a monitorização das comunicações entre os elementos através do *wireshark*.

Num anexo próprio do relatório, os alunos devem explicar (com a ajuda eventual dum mapa de rede) o ambiente de teste criado. Devem incluir integralmente os ficheiros de configuração de todos os servidores instalados (ficheiros de configuração, ficheiros de bases de dados DNS, ficheiro com a lista de ST) e alguns excertos de ficheiros de *log* dos servidores depois de terem sido executados vários exemplos por forma a serem geradas entradas relevantes nesses ficheiros. Ou seja, neste anexo devem também incluir exemplos de comandos executados no CL bem como os resultados obtidos.

Planeamento das tarefas

Os grupos devem fazer um planeamento prévio das principais tarefas a realizar durante trabalho (sobretudo as tarefas mais prementes) e a forma como se vão organizar como grupo em termos da sua execução. Sugere-se a criação duma tabela com essa lista de tarefas, o plano de ação para a realizar, a sua ordenação temporal, a sua classificação como obrigatória ou opcional, etc.

Antes de iniciar a implementação de qualquer módulo ou componente, os grupos devem especificar o modelo de informação e o modelo comunicacional a utilizar no sistema (definição formal do formato das mensagens, dos mecanismos/regras de codificação, de todas as interações possíveis entre os elementos do sistema, incluindo o protocolo para transferência de zona, etc.).
Devem também definir as estruturas de dados duma forma abstrata e a organização dos componentes de software a desenvolver, indicando de antemão os módulos que os constituem e que podem ser reaproveitados de componente para componente. Devem definir a organização dos componentes em termos de processos em execução, usando *multithreading* sempre que necessário.

Para além disso, é necessário especificar logo o ambiente de teste, a hierarquia de nomes escolhidos para os domínios e a forma como são mapeados os servidores e os domínios no ambiente do Core. Apesar de não conseguirem implementar no início o ambiente de teste especificado, devem criar de imediato todos os ficheiros de configuração e todos os ficheiros de dados do ambiente que virão depois a ser usados no ambiente de teste. Isto ajuda na consolidação dos conhecimentos sobre o serviço DNS antes de começarem a implementação dos componentes do sistema.

Estas definições todas anteriormente referidas devem ser plasmadas numa primeira versão do relatório para irem sendo discutidas entre os elementos do grupo e com os docentes, atualizadas e corrigidas sempre que necessário.

Os grupos podem então, depois da validação de todas as especificações anteriores, começar a implementar e testar os módulos principais e que são comuns à maior parte dos componentes. Devem construir e testar os primeiros componentes completos tendo em consideração os critérios de avaliação

da primeira fase. Relembra-se que é essencial implementar bem o modo *debug* para que o teste dos módulos e dos componentes possa ser bem feito logo desde o início.

Critérios de avaliação para duas fases distintas

Nesta secção serão definidos os critérios de avaliação para serem tomados em conta na avaliação do trabalho desenvolvido. O trabalho global será avaliado em duas fases distintas. Para cada fase serão indicadas tarefas distintas, ainda que algumas sejam definidas como uma evolução/correção/otimização de tarefas da fase anterior.

A definição dos objetivos e a forma como serão avaliados pode fornecer pistas úteis aos grupos sobre a forma de organizar o trabalho e a maneira de modularizar os componentes a especificar e implementar.

Claramente, na primeira fase deve dar-se primazia à definição de requisitos funcionais a atingir, de modelos de informação, de modelos de comunicação, de eventuais modelos de codificação, de estratégias de implementação, etc. Uma boa especificação e um bom planeamento prévio é um requisito essencial para o sucesso na realização deste trabalho.

Na primeira fase também é importante definir claramente, e com detalhe, uma primeira versão do ambiente de teste, incluindo todas as configurações dos servidores dos domínios (ficheiros de configuração, ficheiro de bases de dados, etc.).

Ainda durante a primeira fase devem desenvolver-se os componentes básicos de comunicação e processamento da informação duma forma modular por forma a facilitar o reaproveitamento do software em todos os elementos do sistema.

No final da primeira fase já devem existir, no mínimo, os primeiros protótipos de um SP, de um SS e de um CL a suportarem a comunicação básica por *query* e resposta direta. Deve também ser implementada a comunicação necessária para executar uma transferência de zona entre um SP e um SS.

Também é importante apresentar um bom relatório logo no final da primeira fase. O relatório final a apresentar no final da segunda fase terá este primeiro relatório como base.

Os trabalhos na segunda fase centrar-se-ão em aspetos evolutivos dos protótipos já desenvolvidos, acrescentando funcionalidades e, se possível, implementando extras opcionais. Devem ser implementados o resto dos elementos do sistema à custa dos módulos já desenvolvidos.

Duas tarefas muitos importantes desta fase final são a realização e análise de testes no ambiente de trabalho definido na fase anterior e a escrita do relatório final.

Por fim, faz-se notar que será possível, na avaliação da fase final, melhorar a avaliação de alguns critérios da fase anterior se esses critérios puderem refletir uma evolução positiva relevante dos aspetos que são importantes para a quantificação desses critérios. Ou seja, é possível melhorar a nota da avaliação da primeira fase durante a avaliação final.

Tipos de avaliação

Os critérios têm associado um tipo que define a forma como a valoração do critério será adicionada à classificação total da fase. Se o tipo é "Peso" então o critério é valorado numa nota de número inteiro entre 0 (zero) e 5 (cinco) e adicionado ao total da avaliação da fase respetiva através do peso relativo definido na tabela dos critérios de avaliação. Se o tipo é "Extra" quer dizer que o critério é valorado numa nota de número inteiro entre 0 (zero) e 5 (cinco) e adicionado diretamente à classificação dum

critério já avaliado anteriormente (indicado na coluna de "Peso") e, neste caso, o critério não tem valoração para a fase em que ele é definido, mas sim para a fase referenciada por "Peso".

É possível que na avaliação da segunda fase os grupos consigam melhorar a avaliação de alguns critérios da primeira fase e, assim, melhorar a classificação final da primeira fase.

Classificação final de cada fase

As classificações finais obtidas em cada fase são depois multiplicadas por 4 (quatro) para obter a nota final de cada fase na escala 0 (zero) a 20 (vinte).

O peso da classificação de cada fase para a nota final da componente prática da UC já foi definido pela equipa docente na apresentação da UC, sendo que as notas individuais dos alunos dum grupo podem ser diferentes da nota final alcançada pelo grupo nas duas fases do TP2 pois a notas individuais também têm em conta a prestação individual de cada aluno na defesa/apresentação do trabalho e com a participação nas aulas PL.

	Tarefa/Objetivo 1ª Fase	**Tipo de Avaliação**	**Peso**
A.1	Arquitetura do Sistema (descrição do sistema e especificação completa dos requisitos funcionais esperados para cada elemento do sistema; descrição dos componentes de software a utilizar e os módulos que os compõem)	Peso	5%
A.2	Modelo de Informação (especificação completa da sintaxe e da semântica de todos os ficheiros e do comportamento dos elementos em situação de erro de leitura, do PDU/mensagens DNS e eventual mecanismo de codificação binária)	Peso	10%
A.3	Modelo de Comunicação (especificação completa de todas as interações possíveis e do comportamento dos elementos em situação de erro)	Peso	10%
A.4	Planeamento do Ambiente de Teste (descrição e especificação completa do ambiente de testes a utilizar na fase 2, incluindo o conteúdo dos ficheiros de configuração e de dados)	Peso	15%
A.5	Protótipo SP e SS em modo *debug* (implementação dum componente que implemente um SP e um SS em modo *debug* conciso que suporte a receção de *queries* e responda, no mínimo, com a inclusão simples de apenas o campo de resposta direta; inclusão do mecanismo de transferência de zona simples sem verificação das versões das bases de dados)	Peso	20%
A.6	Protótipo CL em modo *debug* (implementação dum componente que implemente um CL em modo *debug* conciso que suporte o envio de *queries* e permita, no mínimo, visualizar o campo de resposta direta)	Peso	10%
A.7	Relatório (avaliação da qualidade da primeira versão do relatório com a informação necessária para ajudar a avaliar os critérios A.1 a A-6)	Peso	15%
A.8	Apresentação/Defesa (avaliação da qualidade da apresentação e defesa do trabalho e que ajuda a avaliar os critérios A.1 a A.6)	Peso	15%

Tabela 1: Critérios de avaliação da 1ª Fase do TP2.

	Tarefa/Objetivo 2ª Fase	Tipo de Avaliação	Peso
B.1	Atualização da Arquitetura do Sistema (correção e adição de novos aspetos na especificação da arquitetura)	Extra	A.1
B.2	Atualização do Modelo de Informação (correção e adição de novos aspetos na especificação respetiva)	Extra	A.2
B.3	Modelo de Comunicação (correção e adição de novos aspetos na especificação respetiva)	Extra	A.3
B.4	Planeamento do Ambiente de Teste (correção e adição de novos aspetos da descrição e especificação completa do ambiente de testes)	Extra	A.4
B.5	Implementação de SP, SS, ST, SDT e SR em modo *debug* e normal (implementação completa do componente que implemente um SP, tendo em consideração que um ST, um SDT e um SR são apenas casos particulares de implementações de SP ou SS configurados de forma adequada e incluindo um mecanismo de *cache*)	Peso	40%
B.6	CL em modo *debug* e modo normal (implementação completa dum componente que implemente um CL)	Peso	15%
B.7	Implementação do Ambiente de Teste (implementação e teste da especificação avaliada em A.4/B.4)	Peso	15%
B.8	Relatório (avaliação da qualidade da versão final do relatório com a informação necessária para ajudar a avaliar os critérios B.1 a B-7)	Peso	15%
B.9	Apresentação/Defesa (avaliação da qualidade da apresentação e defesa do trabalho e que ajuda a avaliar os critérios B.1 a B.7)	Peso	15%

Tabela 2: Critérios de avaliação da 2ª Fase do TP2.

Datas relevantes

- Data limite de entrega do relatório e material da 1ª fase: 23:59 do dia 16, 17 ou 18 de novembro de 2022, conforme se trate de grupos de turnos de quarta-feira, quinta-feira ou sexta-feira, respetivamente.

- Apresentação/Defesa do trabalho desenvolvido na 1ª fase: semana de 21 a 25 de novembro de 2022, em sessões individuais a marcar com os docentes. Por razões de disponibilidade de agenda e de salas este período pode ser alargado por mais uns dias.

- Data limite de entrega do relatório final e material completo: 23:59 do dia 2 de janeiro de 2023 para todos os grupos de todos os turnos.

- Apresentação/Defesa final do trabalho desenvolvido: semana de 16 a 20 de janeiro de 2023, em sessões individuais a marcar com os docentes. Por razões de disponibilidade de agenda e de salas este período pode ser alargado por mais uns dias.

FAQ

1. Arquitetura e Modelos: O modelo comunicacional do sistema já está especificado no enunciado. No entanto, é referido que devemos explicar o modelo das mensagens enviadas entre os diversos componentes do sistema. Devemos então criar um novo modelo de comunicação ou utilizar o que está no enunciado?

Os grupos devem utilizar a arquitetura do sistema, o modelo comunicacional e o modelo de informação descrito no enunciado. No entanto, devem usar a sua própria especificação, incluindo a eventual utilização de formalismos linguísticos mais precisos. Podem também adicionar alguns aspetos que no enunciado não são detalhados.

2. Servidores de Topo: O enunciado refere a necessidade de implementar, pelo menos dois ST. As suas bases de dados terão de ser iguais, certo? Existe alguma relação do tipo SP <-> SS?

No DNS real, os ST não implementam uma relação hierárquica do tipo SP <-> SS. São todos iguais e mantêm entre eles a informação atualizada. No contexto do trabalho, pode resolver-se a questão de duas formas:

A. Ficam todos como SP do domínio especial "root" (que não aparecerá no nome dos domínios de topo) sem servidores secundários e copia-se manualmente o ficheiro de dados (com a informação dos domínios de topo e respetivos servidores SDT) para todos os ST do ambiente de teste.

B. Fica um dos ST como SP do domínio especial "root" (que não aparecerá no nome dos domínios de topo) e os outros como SS para que a atualização seja automática.

Relembra-se que, no caso dos SDT, este são domínios "normais", com um SP e um ou mais SS instalados.

3. Zona de DNS reverso: Como deve esta zona ser implementada?

Da mesma forma que o *reverse mapping* é implementado no DNS real por baixo de .arpa. Pode consultar-se um breve tutorial sobre *reverse mapping* aqui:
`http://www.tcpipguide.com/free/t_DNSReverseNameResolutionUsingtheINADDRARPADomain-2.htm`

Ou seja, os grupos têm de configurar um domínio de topo denominado de .reverse com um único subdomínio in-addr (ficando então in-addr.reverse) e, por baixo desse aparece a hierarquia dos números que compõem os endereços IPv4. Atendendo aos endereços IPv4 usados na configuração do ambiente de teste, os grupos têm de preparar o *reverse mapping* para se poder obter o nome dum serviço (ou email ou www, por exemplo) ou do SP, a partir do endereço IPv4. Para simplificar, nestes domínios de *reverse mapping* pode usar-se apenas um SP por cada domínio numérico necessário, sem qualquer SS instalado.

4. Formato das mensagens DNS: O que se entende por "mecanismo de codificação binária"?

Na versão obrigatória, criada na 1ª fase, as mensagens são formadas por *strings* literais com os campos. No final, uma mensagem é um *string* que é a concatenação de todos os campos no formato que é designado por "debug conciso". Parecido com que o HTTP/1.* faz...

Opcionalmente, os alunos podem implementar uma codificação binária, i.e., cada campo é codificado em valores de base 2, ou seja, binário, utilizando um método eficiente de codificação (como fazem o TCP ou o UDP). Por exemplo o primeiro campo duma mensagem DNS, o campo da identificação, é um

inteiro que vai de 1 até 65535. Ora, uma forma eficiente de codificar um valor inteiro deste género é utilizar dois bytes. Imaginemos que o valor da identificação é 1023. No modo de codificação em *string* literal (que no enunciado é o formato "debug conciso"), este campo é representado pela *string* "1023", que ocupa, no mínimo, 4 bytes (cada um com a representação ASCII do respetivo dígito). Numa codificação binária, este valor é codificado em apenas 2 bytes (16 bits) como 0000001111111111. Além disso, numa codificação binária não são usados separadores dos campos.

5. Modo *debug*: O modo *debug* de funcionamento dos componentes/elementos é simplesmente apresentar as mensagens de Log no terminal, para além de as registar no ficheiro de log?

Sim.

6. Entrada do tipo DD: Nos ficheiros de configuração dos servidores SP, SS e SR para que servem as entradas do tipo DD?

Um servidor SR pode pertencer a (estar dentro) um determinado domínio e, para perguntas feitas sobre esse domínio, podemos preferir que ele contacte diretamente logo o SP e os SS desse domínio. É o endereço desses SP e SS que deve ser indicado no DD. Por exemplo, se um SR estiver dentro do departamento de informática, o DD pode indicar que para *queries* sobre o domínio di.uminho.pt, quando a resposta já não está em *cache*, o SR deve contactar os servidores indicados pelas entradas DD deste domínio em vez de contactar os ST.

Se for um SP ou um SS, esta entrada indica que o servidor só responde a *queries* sobre esses domínios (geralmente sobre os domínios de quem são responsáveis). Isto permite aumentar a segurança e o desempenho dos servidores porque não têm de perder tempo a responder a *queries* sobre domínios de que não são responsáveis/autoritativos.

7. Lista de ST: Porque é que um servidor dum subdomínio (ex: alunos.pl5.lei-cc.) tem de ter um ficheiro com a lista dos ST?

Se um servidor dum domínio receber uma *query* sobre um qualquer subdomínio desse domínio, não precisa contactar um ST, basta contactar um servidor desse subdomínio para iterativamente obter a resposta. Mas para *queries* sobre todos os outros domínios, e a resposta não está em *cache*, a única forma do servidor obter uma resposta é perguntar a um ST e começar iterativamente a obter os servidores de domínios hierarquicamente superiores ao domínio da *query* (primeiro dum domínio de topo e depois dum seu subdomínio e assim por diante) até obter os endereços dos servidores do domínio da *query*. Sem a lista dos ST o servidor não conseguiria arrancar o processo de obter uma resposta indireta por iteração ou recursividade.

8. Nomes de Domínios: No enunciado é definido que não se podem replicar nomes de domínios que existam na realidade. Isto também é válido também para os domínios de topo (*Top Level Domains*)?

A regra também se aplica para domínios de topo (como .com, .pt, etc.). Os grupos terão de inventar nomes de domínios de topo não utilizados em lado nenhum.

9. Ambiente de teste: Pode usar-se a topologia do Core fornecida para o TP1?

Sim, os grupos devem, preferencialmente, usar a topologia do TP1 para implementar o ambiente de teste. Não é necessário acrescentar mais *routers*, *hosts* ou redes. Podem usar os nomes dos *hosts* e *routers* que estão já definidos ou podem trocá-los por outros à escolha.

Devem executar/correr todos os elementos (ST, SDT, SP, SS, SR e CL) nos *hosts* dessa topologia, de preferência um elemento em cada *host* (mas também podem executar/correr mais do que um elemento no mesmo *host* se quiserem, desde que usem portas diferentes de atendimento).

10. Arquitetura dos elementos: Pode criar-se um único programa que implemente todos os elementos do sistema?

Sim, podem desenvolver um único programa que implemente todos os tipos de elementos e a diferenciação é feita no arranque através de argumentos de execução e dos ficheiros de configuração/dados.

Mas também podem criar um programa diferente para cada tipo de elemento, por isso foi aconselhado que a arquitetura dos elementos seja modular por forma a reutilizarem-se módulos nos elementos.

É aconselhável que, no mínimo, sejam criados dois programas bem diferenciados, um que implemente os elementos servidores e outro que implemente o cliente.

11. Sistema de *caching*: O que acontece se um servidor não tem a informação em *cache* de todos os *resource records* para completar os campos *Authorities Values* e *Extra Values* da resposta a uma *query*?

Quando um servidor tem apenas a informação em *cache* suficiente para preencher o campo *Response Values* da resposta à query, deve deixar os campos *Authorities Values* e *Extra Values* em branco.

Nunca é possível a um servidor saber se a informação não autoritativa que tem em *cache* está completa ou não. O primeiro passo é verificar se tem na *cache* uma ou mais entradas válidas que respondam diretamente à *query* (e que são para incluir no campo *Response Values* na mensagem de resposta). Depois tenta encontrar informação válida em *cache* sobre os servidores primário e secundários do domínio referido na *query* para incluir no campo *Authorities Values*. Se não encontrar deve deixar esse campo em branco. Além disso, deve tentar encontrar informação em *cache* sobre os endereços IP de todos os servidores e *hosts* referidos nos campos *Response Values* e *Authorities Values* para incluir no campo *Extra Values*. Mais uma vez, se não encontrar essa informação deve deixar esse campo em branco. Se encontrar informação incompleta deve incluir a que tiver. Estas situações raramente acontecem pois quando um servidor guarda em *cache* respostas vindas doutros servidores, costuma vir sempre completa, com todas as entradas necessárias para incluir nos campos *Authorities Values* e *Extra Values*.

12. Organização dos Domínios de DNS: Existe alguma relação entre a estrutura organizativa dos domínios DNS e a topologia das redes IP?

Não existe nenhuma relação entre a organização dos domínios e subdomínios do DNS e a topologia das redes IP. Como tal, não faz sentido colocar o SP e os SS dum domínio na mesma rede IP local. Os SS existem para aumentar a fiabilidade do serviço, logo, devem estar todos "afastados" uns dos outros. Pela mesma razão, os ST do ambiente de teste não devem estar na mesma rede local.

13. Protótipos da Fase A: Qual o nível de complexidade da implementação da interação entre os servidores e o cliente na Fase A?

A implementação do envio duma *query* e a composição e envio da resposta a essa *query* é simplificada na Fase A. Se for necessário, a mensagem da *query* e da resposta podem ser construídas manualmente e definas *hard-coded* no código, podendo ser utilizada a query e a resposta já incluídas no exemplo do enunciado.

14. Codificação binária das mensagens DNS: Que tipos de dados devem ser suportados?

Na codificação binária a especificar na Fase A (e implementar opcionalmente na Fase B) só é necessário codificar três tipos de campos: números inteiros positivos (identificadores, TTL, prioridades, etc.), *strings* (nomes de domínios, máquinas, serviços, etc.) e *flags*. Os inteiros são facilmente codificáveis num número certo de bytes, dependendo do intervalo de valores possíveis. As *strings* são codificadas diretamente numa sequência de símbolos ASCII (todos os nomes que é possível usar nas bases de dados do DNS têm de ser representáveis por códigos ASCII). As *flags* podem ser facilmente codificadas em bits específicos dum byte. Os separadores dos campos na codificação binária são desnecessários. Também não é preciso qualquer mecanismo de compressão, encriptação ou verificação da integridade das mensagens de DNS.

15. Entradas da base de dados dum domínio e a *cache*: Pode usar-se uma única estrutura de dados para armazenar tudo num servidor?

Tal como definido na especificação incluída do enunciado, pode usar-se a *cache* para armazenar os dados da base de dados do domínio em memória (para os SP e SS) bem como os dados recebidos por fontes externas (para todos os servidores), sem necessidade de ter uma estrutura de dados separada.

16. Acesso aos dados da *cache*: É necessário especificar a *cache* para suportar procedimentos otimizados para registo e procura da informação?

Não é necessário haver preocupações de eficiência/otimização das funções de registo e procura de *resource records* na *cache*. O registo simples na primeira ou última entrada FREE da cache e a função simples de procura sequencial a partir dum determinado ponto, tal como especificado no enunciado, é suficiente. Não é necessário implementar mecanismos de *hash* ou algoritmos de *sorting*. Um simples *array* de estruturas (define-se uma estrutura para representar a informação de cada linha da tabela) de tamanho fixo é suficiente.

17. Implementação de tipos de servidores: É possível um servidor ter o papel de SP e SS simultaneamente?

Sim, desde que assuma o papel de SP e SS para domínios diferentes. Um servidor pode ser um SP e/ou SS para vários domínios em simultâneo, desde que não assuma o papel de SP e SS para o mesmo domínio.

Note-se que um servidor assumir o papel de SP e SS em simultâneo não é o mesmo que ter dois servidores distintos no mesmo endereço IP:porta (ou uma solução equivalente em que um componente arranca "threads" diferentes e independentes para dois servidores). Este caso é totalmente desaconselhável.

18. Zona de DNS reverso: Pode ter-se uma configuração simplificada?

Sim, pode simplificar-se a configuração necessária para o DNS reverso do enunciado. Mais concretamente:

A - Se for utilizada a topologia do TP1 só existe uma rede classe A (10.0.0.0) em toda a topologia, podendo ignorar-se o sub-endereçamento utilizado;

B - Só é necessário configurar o DNS reverso para os três domínios da árvore "10.in-addr.reverse";

C - No domínio "10." só é necessário incluir duas ou três entradas PTR (IPv4) na base de dados do domínio, para além das entradas SOA* e NS;

D - Não é preciso configurar qualquer servidor como SS para os domínios de DNS reverso;

E - Os SP dos três domínios podem ser executados no mesmo servidor, i.e., cria-se um ficheiro de dados para cada um dos três domínios, mas usa-se um único ficheiro de configuração para um único módulo de *software* de servidor a executar.