

Parte III

Aplicações – Protocolos e sistemas de suporte

Nos anos iniciais da Internet foram desenvolvidas várias aplicações: a transferência de ficheiros, suportada no protocolo FTP (*File Transfer Protocol*), o acesso remoto a outro computador para aí realizar uma sessão interactiva, suportado no protocolo TELNET, o correio electrónico, suportado no protocolo SMTP (*Simple Mail Transfer Protocol*), e os grupos de discussão através de mensagens (*e.g.*, Usenet newsgroups). Estes protocolos foram igualmente complementados com protocolos de suporte do funcionamento global da rede, como por exemplo o protocolo NTP (*Network Time Protocol*), e o sistema de designação da Internet, o DNS (*Domain Name System*).

Durante a primeira metade da década de 1990, foi desenvolvido o acesso a conteúdos, suportado no protocolo HTTP (*Hyper Text Transfer Protocol*) e foram introduzidos clientes HTTP (os designados navegadores ou *browsers* Web) nos computadores pessoais, o que permitiu a generalização do uso da Internet pelos “homens e mulheres comuns”. Adicionalmente, tal conduziu à introdução de muitos sistemas aplicacionais que se transformaram na, praticamente única, face visível da Internet: redes sociais, sistemas de correio electrónico, sistemas de pesquisa de informação, transmissão de voz e vídeo, televisão digital, sistemas de mensagens e de colaboração, *sites* Web de notícias, e mais um infinável rol de novas aplicações e sistemas.

Muitos destes sistemas já não dependem exclusivamente de um protocolo, um cliente e um servidor, mas sim de sistemas aplicacionais compostos por vários protocolos, muitos servidores e outros sistemas de suporte que são eles próprios sistemas distribuídos complexos. Entre os protocolos de suporte destas aplicações destaca-se o protocolo HTTP.

A descrição de todos esses sistemas e da forma como estão organizados ultrapassa o âmbito de um livro de fundamentos de redes de computadores. Por essa razão este capítulo apenas apresenta dois protocolos do nível aplicacional, o sistema e o protocolo DNS e o protocolo HTTP. Mas depois complementa os mesmos com uma breve introdução à forma como estão organizados os sistemas de suporte às aplicações mais populares na Internet actual. A esta panorâmica breve de protocolos de suporte directo de aplicações deve ser adicionado o protocolo RTP, de transporte de dados multimédia, já apresentado no Capítulo 9, ver a Secção 9.4.

Segue-se uma breve apresentação dos capítulos desta parte do livro.

Capítulo 11 Para se poder utilizar um serviço é necessário designá-lo e obter a sua localização na rede. Este papel é desempenhado pelos sistemas de designação ou nomeação (*Naming Systems*). Este capítulo apresenta o sistema de designação mais popular da Internet, o DNS.

Capítulo 12 Para se transferirem objectos entre clientes e servidores é possível usar o protocolo TCP mas este não permite indicar qual o nome do objecto a transferir, nem permite indicar qual o tipo do objecto ou outros atributos do mesmo. O protocolo HTTP, que é apresentado neste capítulo, é um protocolo genérico de transferência de conteúdos através de conexões TCP (ficheiros de todos os tipos, páginas Web, programas que são executados pelos *browsers* Web, fotografias, *etc.*).

Capítulo 13 Quando uma aplicação ou um *site* Web se torna muito popular e adquire uma grande escala, não é mais possível implementá-la com um único servidor para servir todos os utilizadores. A solução é construir um sistema distribuído, com muitos servidores, designado por CDN (*Content Distribution Network*). O objectivo deste capítulo é estudar as soluções usadas para aumentar a escalabilidade das aplicações distribuídas baseadas no protocolo HTTP e descrever como são organizadas as redes aplicacionais que as suportam.

Para além do DNS e do HTTP, existem muitos outros protocolos aplicacionais representativos e que suportam aplicações populares como o correio electrónico (SMTP, POP e IMAP), os sistemas de vídeo conferência e a telefonia sobre a Internet (SDP, SIP e H.323), a difusão de vídeo a pedido (RTSP), gestão de equipamentos de rede

(SNMP), *etc.* Algumas dessas aplicações e protocolos baseiam-se em aproximações semelhantes às que serão explicadas neste capítulo (*e.g.*, SMTP, POP e IMAP) ou que já foram referidas no Capítulo 9. Por essa razão, ou para não aumentar demais a dimensão deste livro, não os incluiremos aqui.

O leitor pode, por exemplo, consultar nos seguintes livros de texto: [Peterson and Davies, 2012], [Tanenbaum and Wetherall, 2011] e [Kurose and Ross, 2013] a descrição desses protocolos e aplicações, geralmente nos seus capítulos sobre aplicações.

Capítulo 11

Nomes e endereços

The Domain Name System (DNS) is the Achilles heel of the Web. The important thing is that it's managed responsibly.

– Autor: Sir Tim Berners-Lee, inventor da Web

Logo que os humanos começaram a ter capacidade de comunicar e formular pensamentos abstractos, tiveram necessidade de poder designar objectos, coisas, pessoas e outros seres. Para esse efeito recorremos a sequências de palavras, *i.e.*, sequências de símbolos sonoros, ou a sequências de símbolos gráficos quando se usa uma forma escrita. O objectivo é discriminar uma dada entidade, seja qual for a sua espécie, entre outras entidades. Por exemplo, o nome uma localidade discrimina-a entre as diferentes localidades, o nome de uma pessoa discrimina-a entre outras pessoas, *etc.* Os humanos também introduziram nomes para conjuntos de entidades, muitas vezes captando atributos ou propriedades do conjunto. Por exemplo, “legumes” designa o conjunto de todas as entidades que são do tipo legumes, e “vertebrado” designa um conjunto de animais com esqueleto ósseo, um dado atributo desses animais.

Os humanos também começaram a usar sequências de símbolos semelhantes a nomes para discriminar localizações. Inicialmente essas localizações estavam associadas a formas de chegar a uma dado local. Por exemplo, a horta que fica por detrás de um dado riacho, ou uma zona que fica para lá dos montes como “Trás-os-montes”. Modernamente, inventaram-se outros métodos mais sofisticados de indicar localizações, de forma não relativa, como por exemplo o sistema de endereços urbanos (por país, cidade, rua, número da porta, *etc.*). O sistema de coordenadas GPS é outra forma de indicar localizações. Este tipo especial de cadeias de símbolos designam-se por endereços, pois permitem indicar a localização de uma entidade. Às vezes isso pode ser suficiente para discriminar a entidade, sobretudo quando a entidade não se pode mover. Nestes casos o nome e o endereço da entidade como que coincidem. Mas, geralmente, um endereço não discrimina uma entidade e é diferente de um nome, pois um endereço discrimina uma localização e muitas entidades podem mudar de localização pois são móveis.

Os sistemas modernos de comunicação, mesmo antes da digitalização, fizeram uso dos nomes e endereços. Por exemplo, no sistema telefónico tradicional, as pessoas e as empresas que tinham telefone eram listados em listas telefónicas que associavam os seus nomes a endereços, *i.e.*, os seus números de telefone. Com efeito, no sistema telefónico clássico, um número de telefone estava associado a uma dada porta numa central telefónica e a um fio que ligava essa central telefónica às instalações onde estava

o telefone, portanto estava associado a uma localização específica. Quando as pessoas mudavam de habitação, tinham de mudar de número de telefone.

Com a digitalização característica da segunda metade do Século XX, e em particular com o aparecimento das redes de computadores e dos sistemas distribuídos, os nomes e os endereços passaram a ter uma utilização maciça que ultrapassa em muito as noções informais e do tipo senso comum que referimos acima. Este capítulo começa exactamente por definir mais rigorosamente o que são nomes, endereços, identificadores e sistemas de designação do ponto de vista das redes e dos sistemas distribuídos.

11.1 Nomes, endereços e identificadores

Num sistema informático, ou numa rede de computadores, um **nome** (*name*) é um símbolo associado a uma entidade. Geralmente o nome tem uma dada sintaxe, dado que é formado por uma sequência de símbolos elementares, como por exemplo uma sequência de caracteres. A sintaxe dos nomes indica quais os nomes legais num determinado contexto. O nome de uma entidade está associado, ou é mapeado, através de um sistema de designação, num conjunto de atributos que caracterizam a entidade nesse sistema.

Por exemplo, na maioria dos sistemas de operação existe um sistema de ficheiros que permite aceder a ficheiros designados por um nome mnemónico. Em que consiste o sistema de designação dos ficheiros? Um tal sistema é semelhante a um catálogo que mapeia nomes de ficheiros nos seus atributos (data de criação, dimensão, tipo, localização num disco, *etc.*). Geralmente, entre esses atributos existem atributos especiais de localização, localizadores ou **endereços** (*addresses*), que indicam onde o ficheiro reside num disco ou numa memória não volátil. Este exemplo põe em evidência que um sistema de gestão de ficheiros é composto por um sistema de designação de ficheiros, que associa nomes a atributos de ficheiros, e por um sistema de acesso aos ficheiros, que usa localizadores ou endereços de ficheiros para materializar o acesso ao conteúdo dos ficheiros propriamente ditos.

Assim, um sistema de designação é um catálogo que associa nomes de entidades aos seus atributos. Formalmente trata-se de uma função ou um mapa. Cada sistema de designação tem uma sintaxe própria, que indica que nomes são legais e como estes estão organizados. Por exemplo, a maioria dos sistemas de designação mais simples organiza os nomes de forma hierárquica, usando um separador de nível hierárquico: o carácter “/” nos sistemas a la Unix ou o carácter “\” nos sistemas a la Windows.

Os sistemas de designação mais comuns têm diversos mecanismos que introduzem diversos tipos de nomes como por exemplo: “sinónimos” (*alias*), caso em que uma entidade pode ter mais do que um nome, nomes “relativos”, nomes “absolutos” (ou canónicos), *i.e.*, com início numa raiz e discriminantes, *etc.*. Os nomes têm a propriedade de poderem ser facilmente memorizados pelo humano, e às vezes até incorporam associações directas a propriedades ou atributos das entidades. Por exemplo, o nome `programs/sources/search.java` está provavelmente associado a um ficheiro contendo o código fonte de um programa de pesquisa escrito na linguagem Java. A Tabela 11.1 apresenta exemplos comuns de nomes usados em diferentes contextos.

Nas redes de computadores e nos sistemas distribuídos existem sistemas de designação que associam nomes a entidades (*e.g.*, servidores, utilizadores, …). A cada uma dessas entidades estão associados atributos. Geralmente, entre esses atributos existem endereços de rede, que permitem comunicar directamente com as entidades.

Endereços são cadeias de símbolos especialmente adaptadas a indicar uma localização num sistema. Um endereço pode conter elementos que facilitam o encaminhamento para a entidade (“como lá se chega”) mas isso não é obrigatório. Por exemplo, um endereço IP (*e.g.*, 195.100.45.36) tem componentes que facilitam o encaminhamento para a interface rede a que o endereço está associado. No entanto, um endereço de

Tabela 11.1: Exemplos de nomes

Nome	Significado
João Filipe das Neves	Nome de uma pessoa
presidente	Nome de uma função (indireção)
joaninha	Alcunha (alias) de uma pessoa
	na agenda de outra pessoa
<code>www.fct.unl.pt</code>	Nome hierárquico de um servidor
<code>/sources/search.java</code>	Nome hierárquico de um ficheiro
<code>IBM</code>	Símbolo de uma empresa
	cotada em bolsa (não hierárquico)

nível canal, como um endereço Ethernet, não contém nenhum elemento que facilite a localização da interface a que o mesmo está associado pois os canais Ethernet foram inicialmente concebidos para funcionarem em difusão. A Tabela 11.2 apresenta exemplos de diversos tipos de endereços.

Tabela 11.2: Exemplos de endereços

Endereço	Significado
130.55.200.10	Endereço IP versão 4 de um computador
2001:690:a00:1036:1113::247	Idem versão 6
80	Endereço da porta IP do serviço HTTP
60:03:08:8d:a5:96	Endereço de nível canal (MAC Address)
0x040004404	Endereço de memória
38°39'39.01"N, 9°12'15.49"W	Coordenadas geográficas da FCT/UNL
Campus da Caparica, 2829-516	Endereço postal da FCT/UNL
CAPARICA, Portugal	

Para além dos nomes tradicionais e dos endereços, muitos sistemas usam igualmente um outro tipo de nomes, que designaremos por **identificadores únicos ou UIDs (Unique Identifiers)**. Ao contrário dos nomes, que são fáceis de decorar por serem formados por símbolos que codificam propriedades das entidades a que estes estão associados, o UIDs são uma espécie de nomes “puros”, *i.e.*, sem estarem relacionados com quaisquer atributos da entidade a que estão associados. Estes nomes são formados por uma (geralmente grande) sequência de símbolos que funciona como uma chave única da entidade.

Nos sistemas distribuídos, os UIDs dizem-se nomes sistema por oposição aos nomes usados pelos utilizadores. Veremos a seguir que também são usados pelos sistemas informáticos acessíveis ao grande público.

O objectivo de se usarem este tipo de nomes é dispor de um nome único, associado de forma unívoca a uma entidade, de maneira que o nome é único no espaço e no tempo. No espaço, porque o nome não muda mesmo que a entidade mude de localização. No tempo, porque nenhuma outra entidade poderá (re)utilizar esse nome mesmo que a entidade associada ao nome deixe de existir. Estas propriedades são úteis em redes de computadores e sistemas distribuídos quando é necessário facilitar a implementação de certos algoritmos, que passam a dispor de chaves únicas no espaço e no tempo para resolverem problemas complexos.

Os UIDs têm de ser afectados centralmente, ou usando um gerador de números aleatórios e um número tal de bits, que torna desprezável, para efeitos práticos, produzir uma colisão em tempo útil. A Tabela 11.3 apresenta exemplos de diversos tipos de identificadores. A mesma põe em evidência que muitos sistemas de designação foram definidos usando UIDs para permitir a designação das entidades a que estão associados (pessoas, empresas, livros, *etc.*) de forma unívoca.

Tabela 11.3: Exemplos de identificadores

Identificador	Significado	Reutilizável
0x02a6dd9df8c3ff6276c11df45	UID de um serviço	N
PT505256256	Número de contribuinte	N
PT8032156	Número de cartão de cidadão	N
+351984592634	Número de telefone móvel	S
PT50003500501042581007	Número de conta bancária	S/N
ISBN 978-0-12-385059-1	ISBN (International Standard Book Number)	N

Como já foi referido acima, um **sistema de designação** (*naming system*) é um sistema que permite obter os atributos associados à entidade que o nome designa, *i.e.*, que implementa um mapeamento entre nomes e atributos. Os nomes, os UIDs e os endereços estão geralmente interligados através de sistemas de designação na forma ilustrada na Figura 11.1. Assim, geralmente os sistemas de nomes são usados para obter os UIDs ou nomes sistema das entidades, ou directamente os seus endereços de rede. Por sua vez, os UIDs também podem ser usados para obter os endereços de rede. Estes sistemas de mapeamento também são muitas vezes designados por directorias. Nas redes e nos sistemas distribuídos estas directorias são geralmente sistemas distribuídos elas próprias. O DNS é o sistema de directoria mais conhecido da Internet.

Nomes (*names*) são sequências de símbolos que permitem designar entidades. Geralmente têm um significado mnemónico para facilitar a sua memorização pois codificam propriedades da entidade que designam.

Identificadores únicos (UIDs) (*Unique Identifiers*) são nomes “puros” que discriminam as entidades a que estão associados de forma única no tempo e no espaço. De forma geral não revelam propriedades das entidades a que estão associados.

Endereços (*addresses*) são sequências de símbolos que permitem indicar como aceder num certo momento à entidade a que estão associados.

Sistemas de designação (*naming systems*) são sistemas que implementam catálogos de tradução ou mapeamento de nomes em propriedades ou atributos das entidades a que os nomes estão associados.

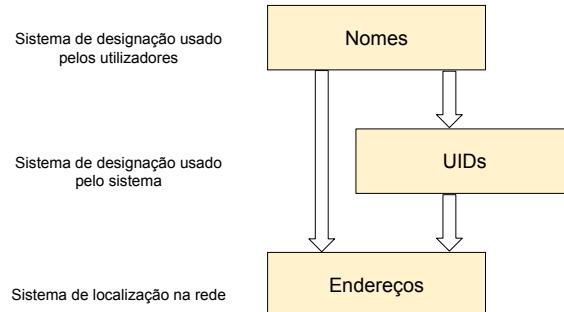


Figura 11.1: Hierarquia de sistemas de designação

11.2 O DNS (*Domain Name System*)

No início da Internet o número de computadores ligados era muito pequeno. O catálogo que associava o nome de cada computador ao seu endereço IP era mantido pelo SRI (Stanford Research Institute da Universidade de Stanfورد, nos E.U.A.) de forma centralizada, num único ficheiro, chamado `hosts.txt`. Com o crescimento do número de computadores ligados foi necessário passar para outro sistema mais prático, distribuído, com capacidade de crescimento em escala, que permitisse uma administração descentralizada (por domínios), e que fosse muito robusto e tolerante a falhas. Esse sistema veio a designar-se **DNS** (*Domain Name System*), foi normalizado inicialmente pelos RFC 1034 e RFC 1035, e implementa hoje em dia um catálogo com várias centenas de milhões de domínios, provavelmente mais de um bilião de entradas, e continua sempre a crescer.

Os nomes DNS são hierárquicos e escrevem-se da esquerda para a direita, ou seja, com a raiz à direita, como por exemplo em `www.fct.unl.pt`. Cada componente da hierarquia é designada por um domínio e o carácter que marca a distinção entre domínios é o carácter '.'. Um nome diz-se completo (*full qualified*) se termina no nome da raiz, cujo nome é '.', pois não se saberia se a seguir não viria mais uma parte do nome que foi omitida. Por exemplo `www.fct` é um nome relativo que só ficaria completo caso se soubesse o resto do nome até à raiz.

Na prática os nomes DNS quase sempre nunca levam o ponto final pois os domínios que estão logo por baixo da raiz são bem conhecidos e raramente daí resultam ambiguidades. Estes domínios são chamados domínios **TLD** (*Top Level Domains*). Quando um nome é dado sem o ponto final, é quase sempre interpretado como se o ponto lá estivesse. Por isso, a ambiguidade é quase sempre resolvida pressupondo que o domínio mais à direita é um TLD, o que conduziria a um erro caso não fosse.

A hierarquia dos domínios está organizada em árvore como ilustrado na Figura 11.2. Por baixo da raiz da árvore encontram-se os TLDs. Estes são de dois tipos, os associados a países ou CC TLDs (*Country Code TLDs*), *e.g.*, US (U.S.A.), UK (United Kingdom), FR (France), ES (Spain), BR (Brasil), PT (Portugal), *etc.* e os genéricos ou GTLDs (*Generic TLDs*), *e.g.*, ORG (Organizações não governamentais), COM (empresas), EDU (entidades educacionais), NET (organizações de gestão da rede), INFO, EU (entidades europeias), *etc.*

Para a designação de domínios, as regras iniciais do DNS só permitiam a utilização de letras do código ASCII, o carácter '_' e números, mas mais recentemente foram autorizados outros caracteres para permitir nomes de domínios com acentos e nomes de domínios usando outros alfabetos (*e.g.*, árabe, chinês, *etc.*). Um nome não pode ter mais do que 253 caracteres e a profundidade da árvore dos domínios também

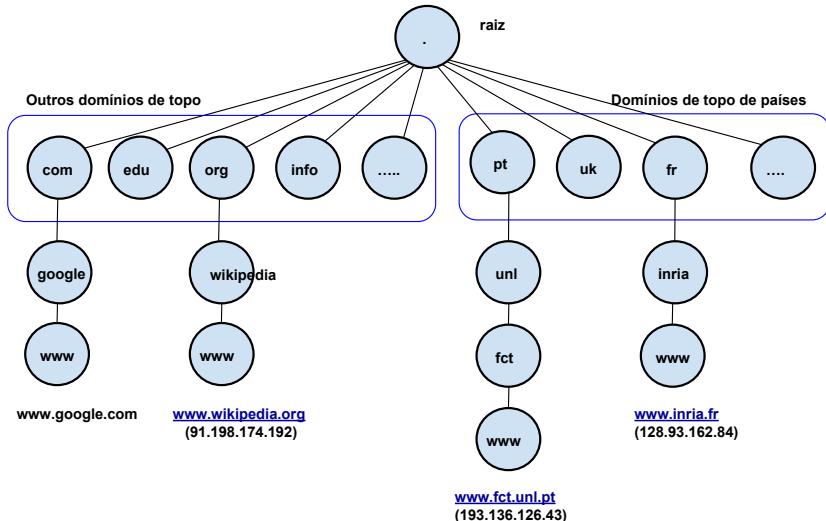


Figura 11.2: Hierarquia de designação do sistema DNS

é limitado a um pouco mais do que uma centena de níveis, o que se tem revelado largamente suficiente. No DNS os nomes não distinguem minúsculas de maiúsculas, ou seja `www.wikipedia.org` e `WWW.WIKIpedia.Org` são o mesmo nome.

De forma simplificada, o DNS pode ser visto logicamente como uma gigantesca base de dados com uma única tabela com várias colunas. A primeira coluna contém um nome, a seguinte um TTL, a seguir o tipo do atributo associado ao nome, e finalmente o seu valor. Ver a Tabela 11.4. Cada linha da tabela diz-se um registo do DNS.

A tabela apresenta um pequeno subconjunto ad-hoc de registos da tabela global do DNS. No entanto, este pequeno exemplo permite ilustrar diversas facetas da utilidade do DNS, assim como da sua própria organização interna.

Em primeiro lugar é possível verificar que todos os nomes existentes na tabela estão escritos terminando em '`.`' porque estão na forma absoluta, *i.e.*, a partir da raiz e incluindo esta. Essa é a forma como o DNS regista internamente os nomes.

Uma outra faceta comum a todos as entradas tem a ver com o facto de a cada uma delas estar associado um tipo, como por exemplo A, AAAA, NS, MX, *etc.* O nome `publico.pt` tem várias entradas na tabela, sendo uma delas do tipo A e duas do tipo NS. O exemplo mostra que o mesmo nome pode figurar diversas vezes, associado a diversas entradas de tipos distintos, mas que também podem figurar várias entradas com o mesmo nome e do mesmo tipo, mas com valores diferentes. Neste exemplo concreto é possível verificar que o nome `publico.pt` está associado a um endereço IP (a entrada do tipo A) e que tem pelo menos dois **Servidores de Nomes** (*name servers*), as entradas do tipo NS.

As entradas mais frequentemente consultadas pelos programas dos utilizadores são as dos tipos A e AAAA, que permitem conhecer os endereços IP (versão 4 ou 6 respectivamente) associados ao nome. É consultando o DNS que um *browser* Web obtém o endereço IP de um servidor Web, por exemplo do domínio `publico.pt`, e pode abrir uma conexão TCP para ir buscar uma página Web associada a esse nome. Quando estão disponíveis, as entradas do tipo AAAA indicam endereços IP na versão 6 associados a um nome.

Os exemplos também permitem verificar que o DNS é utilizado por outros servi-

Tabela 11.4: Visão lógica simplificada de um pequeno extracto do catálogo do DNS (nos tipos: NS = *Name Server*, A = *IPv4 Address*, AAAA = *IPv6 Address*, MX = *Mail eXchanger etc.*)

Nome	TTL	Tipo	Valor
.	254996	NS	a.root-servers.net.
a.root-servers.net.	254785	A	198.41.0.4
a.root-servers.net.	257226	AAAA	2001:503:ba3e::2:30
publico.pt.	3349	A	195.23.42.21
publico.pt.	1744	NS	ns1.novis.pt.
publico.pt.	1744	NS	dns.publico.pt.
dns.publico.pt.	864	A	193.126.13.202
www.unl.pt.	12215	A	193.137.110.30
ns.unl.pt.	86400	A	193.137.110.15
ns.unl.pt.	86400	A	193.137.110.9
fct.unl.pt.	9308	NS	dns1.fct.unl.pt.
fct.unl.pt.	9308	NS	ns3.unl.pt.
www.fct.unl.pt.	27041	A	193.136.126.43
fct.unl.pt.	10800	MX	30 ASPMX2.GOOGLEMAIL.COM.
fct.unl.pt.	10800	MX	30 ASPMX3.GOOGLEMAIL.COM.

ços. Por exemplo, as entradas do tipo MX, associadas ao nome `fct.unl.pt`, permitem ficar a conhecer os nomes dos servidores de correio electrónico desse domínio. Assim, quando um servidor de correio electrónico pretende entregar uma mensagem de correio electrónico dirigida a um endereço de destino da froma `user@fct.unl.pt`, ele consulta o DNS para obter os nomes dos servidores de correio electrónico do domínio `fct.unl.pt`, perguntando, para esse efeito, pelas entradas do tipo MX associadas ao nome `fct.unl.pt`.

Se não obtiver resposta, ele passa a saber que não existe nenhum servidor disponível. Se obtiver uma resposta como a sugerida pela tabela, ele pode escolher qualquer um dos servidores indicados, obedecendo às prioridades indicadas. No exemplo, a tabela permite verificar que existem dois servidores da mesma prioridade (30). Bastaria depois escolher um deles, perguntar pelo seu endereço IP e abrir uma conexão TCP para esse servidor para lhe entregar a mensagem. Curiosamente, a crer nos nomes dos servidores presentes na tabela, verifica-se que o correio electrónico da FCT/UNL é recebido por servidores da Google.

O exemplo também põe em evidência que o DNS tem a sua configuração em termos de domínios e os servidores que os servem codificado dentro dele próprio. Trata-se de um exemplo notável de utilização recursiva de um serviço.

No exemplo vemos que existe pelo menos um servidor (NS) DNS do domínio '.', a raiz. Na verdade existem muitos mais, mas o exemplo só mostra um deles, o servidor de nome `a.root-servers.net`. As entradas seguintes indicam que para consultar o domínio `root` é possível dirigir as consultas para um dos seus servidores, cujos endereços em IPv4 e IPv6 estão a seguir indicados. Outras entradas ilustram a mesma faceta pois a partir das mesmas é possível saber o nome de alguns dos servidores de nomes dos domínios `publico.pt` e `fct.unl.pt`. Isto também nos permite inferir que associada à hierarquia de domínios existe um hierarquia de servidores e que, associados a um domínio, existem vários servidores.

Torna-se desde logo claro que o DNS é muito útil para obter endereços IP, mas também para obter informações úteis para outros serviços, incluindo, recursivamente,

o próprio DNS. Existem várias dezenas de tipos de entradas na base de dados do DNS. Muitas deles são experimentais e são raramente usados.

A Tabela 11.5 contém uma lista com alguns dos mais importantes. A cada entrada do DNS ou linha da tabela do DNS, costuma-se chamar, na sua própria terminologia, um RR (*Ressource Record*) e aos diferentes tipos de registos chama-se os RRT (*Ressource Record Types*). Os principais RRTs omitidos e que são também muito importantes estão relacionados com a segurança e não são tratados neste capítulo.

Tabela 11.5: Lista com alguns dos principais tipos de RRs do DNS

Tipo	Abreviatura	Descrição
Endereço IPv4	A	O valor é um endereço IPv4
Endereço IPv6	AAAA	O valor é um endereço IPv6
Name Server	NS	O valor é um servidor de nomes do domínio nome
Servidor de correio	MX	O valor é um servidor <i>Mail eXchanger</i> do nome
Alias	CNAME	<i>Canonical Name</i>
Comentário	TXT	Texto livre
Localização	LOC	Coordenadas geográficas (é pouco usado)
Start of Authority	SOA	Início de uma partição horizontal ou zona autónoma de gestão do DNS

O tipo SOA tem a ver com a organização interna do DNS, nomeadamente no que diz respeito à descentralização da autoridade de gestão e dos diferentes servidores que estão associados a diferentes zonas de gestão do sistema. Voltaremos a este aspecto na secção seguinte.

Um último aspecto que ainda referiremos aqui é o conceito de TTL, presente nos exemplos apresentados na Tabela 11.4. Quando se faz uma consulta ao DNS, associado a cada RR da resposta aparece um valor de TTL expresso em segundos. Este valor indica ao cliente que se quiser consultar o mesmo RR nos próximos TTL segundos, provavelmente isso não vale a pena, pois o seu valor é pressuposto não mudar durante esse tempo. Posto por outras palavras, se o cliente quiser fazer *caching* da resposta, não deve fazê-lo para além de TTL segundos.

Como é fácil de reconhecer, o DNS contém informações cuja actualização tem lugar com um ritmo muito lento. Por exemplo, o endereço IP de um servidor não muda durante várias semanas seguidas. O mesmo se aplica aos servidores de correio electrónico de um domínio. Assim, é essencial para o desempenho e escala do DNS (lembrem-se que o mesmo gera milhões de domínios e existem biliões de computadores a fazerem consultas) que todos os clientes coloquem as resposta que obtêm em *cache* e evitem repeti-las enquanto não passarem TTL segundos desde que o valor foi obtido.

Nos exemplos ilustrados existem muitos valores diferentes de TTLs pois estes são fixados pelos gestores dos diferentes domínios em função da evolução esperada da informação. Os endereços dos servidores de *root* têm associado um TTL de vários dias. O TTL associado ao endereço IP de alguns servidores de nomes (e.g., unl.pt) é da ordem de grandeza de um dia. Outros RRs presentes na tabela têm TTLs mais baixos (e.g., algumas horas e até alguns minutos). Um TTL com o valor 0 indica que o valor não deve ser *cached*.

De que forma os administradores dos domínios optam pelos diferentes valores dos TTLs? Existe um documento (RFC 1912) com recomendações que indica que os TTLs

mínimos devem ser de algumas horas para os endereços IP, ou superiores para outras informações mais estáveis, como por exemplo o SOA. Nos casos em que se espera que um valor mude com frequência, como por exemplo o endereço IP de um computador móvel mas que presta serviços a outros, o TTL deve ser bastante mais curto.

Uma coisa é certa, se o valor de um RR for alterado, só se tem a garantia de que o antigo valor foi esquecido por todas as *caches* depois de passarem pelo menos TTL segundos. Assim, as alterações dos valores de RRs muito estáveis (com TTLs elevados), devem ser precedidas de um período em que o TTL das mesmas é tornado mais baixo (*e.g.*, alguns minutos).

Em geral a interface de acesso ao DNS pelos seus clientes só permite realizar consultas e todas as informações são públicas. As actualizações são feitas numa base de dados própria do domínio a que só os seus gestores têm acesso. A única excepção a esta regra tem a ver com a actualização do endereço IP de um computador com endereço IP dinâmico, mas que pretende fornecer serviços a outros computadores. Para esse efeito existe um mecanismo de actualização do endereço, sujeito a autenticação, que alguns domínios providenciam. O mecanismo e o protocolo de suporte chama-se DDNS (*Dynamic DNS*) e está descrito no RFC 2136.

Todos os sistemas de operação e todas as linguagens de programação incluem bibliotecas com chamadas ou métodos que permitem consultar o DNS. Por exemplo, o programa abaixo permite obter o endereço IP associado a um nome DNS passado como argumento.

Listing 11.1: Programa Java para consulta do DNS para obter um endereço IP

```

1 import java.net.*;
2
3 public class QueryDNS {
4
5     public static void main ( String[] argv ) throws Exception {
6         InetAddress addr = InetAddress.getByName(argv[0]);
7         System.out.println(addr.getHostAddress());
8     }
9 }
```

Os utilizadores que queiram consultar directamente a base de dados do DNS podem fazê-lo através de diversos utilitários. Um dos mais conhecidos é o **dig**, disponível em quase todos os sistemas. A seguir ilustra-se como se pode usar o **dig** na linha de comando do sistema para obter o endereço IP associado ao nome **www.fct.unl.pt**.

```

$ dig www.fct.unl.pt A
; <>> DiG 9.8.3-P1 <>> www.fct.unl.pt A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29205
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;
;; QUESTION SECTION:
;www.fct.unl.pt. IN A
;
;; ANSWER SECTION:
www.fct.unl.pt. 12160 IN A 193.136.126.43
;
;; Query time: 24 msec
;; SERVER: 10.0.1.1#53(10.0.1.1)
;; WHEN: Tue Mar  8 15:53:25 2016
;; MSG SIZE  rcvd: 48
```

O **dig** mostra qual foi a consulta, qual a resposta e diversas outras informações sobre a consulta como por exemplo o tempo que levou a obter a resposta e o endereço IP do servidor que respondeu. No exemplo acima esse endereço (10.0.1.1) é o endereço de um servidor que estava próximo do computador que fez a consulta. Com efeito, o

DNS está organizado de tal forma que existem vários servidores de *caching* intermédios que tornam o sistema mais eficiente. Foi um desses servidores que respondeu. Levantar este véu é o objectivo da secção seguinte.

O exemplo mostra que associado a cada registo existe ainda uma informação suplementar que aparece antes do tipo. Trata-se da informação sobre a classe ou espaço de nomes (**IN**) consultado. O DNS foi inventado para poder gerir diversos espaços de nomes distintos. No entanto, na prática, só o espaço da Internet (**IN**) tem de facto uma utilização alargada.

O DNS é uma gigantesca base de dados distribuída que associa nomes a atributos de vários tipos. Cada registo da base de dados diz-se um **RR** (*Resource Record*). Cada registo associa uma informação de um certo tipo, o chamado **RRT** (*Resource Record Type*), a um nome. Os RRTs mais comuns são os que permitem associar endereços IP aos nomes do DNS, mas existem muitos outros tipos incluindo aqueles que mantêm a informação interna de parametrização do DNS (NS, SOA, ...).

Os registos existentes no DNS são de consulta pública. A sua actualização só pode ser realizada pelos administradores dos domínios. A excepção são os endereços IP de computadores que fornecem serviços mas cujos endereços mudam com alguma frequência. Para essa actualização existe um extensão específica, que permite ao computador actualizar o seu endereço no DNS.

Associado a cada RR existe um **TTL** (*Time To Live*) o qual indica o limite máximo que o registo pode permanecer numa *cache* sem ser actualizado. Dado que a maioria da informação contida no DNS é muito estável, geralmente os TTLs que lhe estão associados correspondem a várias horas. Esses valores determinam o período máximo em que podem existir valores desactualizados *cached* em clientes do DNS. Assim, o TTL é determinante para controlar tempo máximo de convergência dos valores fornecidos aos clientes após alteração dos RRs.

11.3 Organização e funcionamento

Domínios e zonas

A tabela do DNS (provavelmente com mais de um bilião de entradas) está seccionada horizontalmente em **zonas** (*zones*) que contêm todos os nomes de um domínio, incluindo eventualmente também os nomes contidos nos seus domínios subordinados. Cada uma dessas zonas está sob a mesma autoridade e a cada uma delas está associado um gestor e uma base de dados com todos os RRs da zona.

Para simplificar, o leitor que não vá gerir o DNS de um domínio, pode imaginar que a cada domínio corresponde uma zona. No entanto, na verdade uma zona pode incluir um domínio e todos os seus domínios subordinados se estes estiverem por baixo da mesma autoridade.

Por exemplo, a zona `unl.pt` poderia incluir o domínio `unl.pt` assim como os domínios subordinados de todas as faculdades da Universidade Nova de Lisboa (UNL). Nesse caso, haveria um gestor único para todos esses domínios. No entanto, caso os gestores fossem diferentes, a cada faculdade deveria corresponder uma zona distinta, marcada por um novo registo do tipo SOA (*Start of Authority*). Com efeito, a partição da base de dados em zonas corresponde a uma partição por autoridades e não por domínios. No entanto, para facilitar a discussão, vamos admitir a seguir que a cada domínio corresponde uma zona distinta.

Assim, ao domínio `root`, corresponde a zona `root`, ao domínio `org`, corresponde a zona `org`, ao domínio `com`, corresponde a zona `com`, ao domínio `wikipedia`, corresponde a zona `wikipedia`, ao domínio `pt`, corresponde a zona `pt`, e assim sucessivamente. As zonas realizam uma partição horizontal da base de dados do DNS.

Todos os registos que pertencem ao mesmo domínio, *i.e.*, uma *fatia horizontal* da base de dados, são conhecidos por pelo menos dois servidores do domínio. Esses servidores designam-se por servidores com autoridade (*Authoritative Name Servers*) pois eles conhecem uma cópia actualizada de todos os registos do domínio, assim como o TTL oficial de cada um. A cada domínio está também associada a base de dados do domínio, que só pode ser alterada pelo seu gestor e que, sempre que há uma alteração, repercute-a nos servidores com autoridade sobre o domínio.

Consultas ao DNS

As bases de dados dos diferentes domínios estão organizadas numa hierarquia através dos designados GRs (*Glue Records*). Os GRs são registos de uma zona, que indicam os *name servers* e os seus endereços IP, dos domínios que lhe estão directamente por baixo. Assim, a zona `root` contém o nome de todos os servidores de nomes com autoridade sobre os TLDs, assim como os seus endereços IP. O mesmo se passa com o domínio `pt` que conhece os servidores de nomes e os seus endereços IP de todos os domínios da forma `qualquer-coisa.pt`, e assim sucessivamente.

Portanto, quando se faz uma pergunta a um servidor da zona `root`, mesmo que este não conheça a resposta, pode sempre indicar os nomes e os endereços IP dos servidores que “sabem mais do que ele” sobre essa consulta. O mesmo se aplica aos servidores dos TLDs.

Por exemplo, se a consulta `[name: www.wikipedia.org, type: A]` for recebida por um servidor da zona `root`, este pode sempre responder com os nomes e os endereços IP de servidores da zona `org`. Se um desses servidores receber a mesma consulta, como não conhece a resposta, vai responder com os nomes e os endereços IP dos servidores da zona `wikipedia.org`. Finalmente, se um desses servidores receber a mesma consulta, ele já sabe responder com autoridade à mesma.

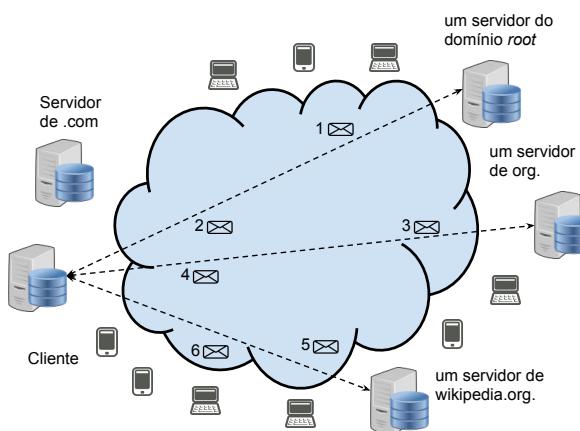


Figura 11.3: Progresso da consulta `[name: www.wikipedia.org, type: A]` iterativamente, começando num servidor da zona `root`. As mensagens ímpares são pedidos e as pares as respectivas respostas. Todas as mensagens de consulta são idênticas.

Quando um servidor não responde a uma consulta sobre um registo que não conhece, procura sempre, se possível, responder com informação, dita *referral information*, para ajudar o cliente a aproximar-se de um servidor que conheça a resposta. Na verdade, procura até antecipar outras perguntas do mesmo cliente e envia-lhe o máximo de informação possível. Por exemplo, se na resposta indica o nome de um servidor, procura também incluir o seu endereço IP pois o cliente vai provavelmente precisar dele.

Esta forma de resolução iterativa das consultas ao DNS é a forma que é executada por omissão pelos servidores DNS quando dialogam entre si. Repare-se que nessa situação esses servidores não fazem nenhuma análise iterativa dos nomes por que pesquisam, pois a consulta feita a cada servidor é sempre a mesma, *e.g.*, [name: www.wikipedia.org, type: A]. Isto é uma diferença importante com respeito ao que se passa normalmente na interpretação do nome de um ficheiro, em que o nome é decomposto nas suas componentes. A forma como tem lugar uma consulta iterativa está ilustrada na Figura 11.3.

A Figura 11.4 mostra o progresso da navegação da consulta através da base de dados das diferentes zonas do DNS. A mesma é dirigida inicialmente a um servidor da zona root. Como este não conhece a resposta, responde com os *glue records* sobre a zona org (um deles está a negrito na figura). A consulta é então dirigida a um dos servidores da zona org. Como este não conhece a resposta, responde com os *glue records* da zona wikipedia.org (um deles está também a negrito na figura). Finalmente, a consulta é dirigida a um desses servidores, que conhece a resposta e responde com o registo [name: www.wikipedia.org, type: A, value: 91.198.174.192].

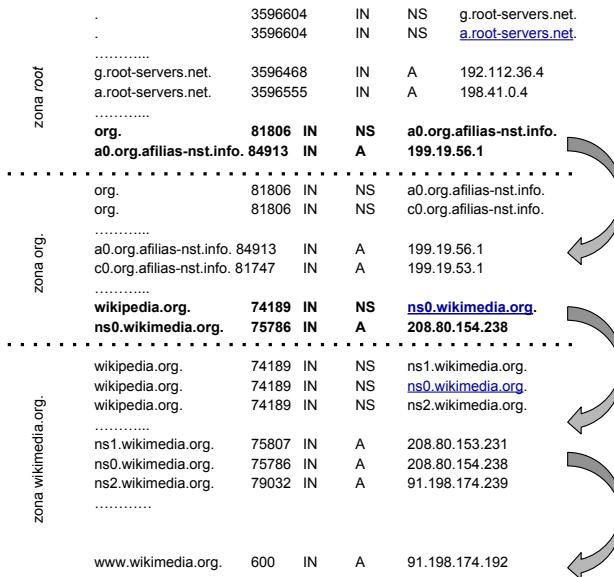


Figura 11.4: Progresso da consulta iterativa [name: www.wikipedia.org, type: A] através das zonas da base de dados global do DNS (a negrito um dos *glue records* de ligação entre zonas)

O protocolo de consulta aos servidores DNS é, por omissão, executado sobre UDP, sempre que a pergunta ou a resposta não ocupam mais do que um certo número de

bytes. Inicialmente este limite foi fixado em 512 bytes, mas hoje em dia os servidores podem responder com *datagramas* UDP de maior dimensão. De qualquer forma, um servidor pode sempre responder com uma resposta truncada e com uma *flag* posicionada a indicar que a resposta completa só pode ser obtida por TCP. Com efeito, o protocolo também pode ser executado sobre TCP, e nesse caso não existem limites teóricos para a dimensão das consultas e das respostas.

Como é evidente, a utilização de UDP é mais eficiente pois a resolução de um nome pode envolver consultas a muitos servidores e cada conexão TCP envolve uma troca preliminar e outra final suplementares de pacotes (para a abertura e fecho da conexão), cuja dimensão pode ser superior à dimensão dos dados envolvidos num pedido e numa resposta. Adicionalmente, servir consultas sobre UDP ocupa muito menos recursos no servidor do que uma consulta sobre TCP.

No entanto, quando as respostas a consultas enviadas sobre UDP são de maior dimensão, mesmo podendo ser dadas por UDP, muitas vezes os servidores optam por responder com respostas truncadas para evitar que sejam usados para ataques de negação de serviço por reflexão, ver o Capítulo 4, Secção 4.2.

Por omissão, o protocolo de pesquisa do DNS é um protocolo cliente/servidor iterativo, através do qual o cliente vai repetindo a pergunta a diferentes servidores, até chegar ao servidor que conhece a resposta. O protocolo funciona sobre UDP e TCP mas, por razões de eficiência, geralmente é executado sobre UDP.

Na ausência de informação adicional, o primeiro servidor interrogado é um servidor da zona **root**. Caso este não conheça a resposta, responde com *referral information*, i.e., o nome e endereço de servidores abaixo na hierarquia, mais próximos da zona a que pertence o registo consultado.

Servidores com autoridade sobre as zonas

Quando um cliente pretende fazer uma consulta ao DNS, e não tem nenhuma informação sobre quais são os servidores com autoridade sobre o domínio a que pertencem os registo que pretende conhecer, pode tentar fazer a consulta a um servidor de um domínio numa posição superior na hierarquia. Naturalmente, em muitas situações, os clientes só podem recorrer aos servidores de último recurso, i.e., a um dos servidores da zona **root**. Por esta razão, estes servidores têm um papel central no funcionamento do DNS e a sua base de dados é crítica.

Inicialmente havia um número pequeno destes servidores, localizados nos E.U.A. e no norte da Europa. Com o alargamento da Internet, passaram a existir 13 servidores espalhados pelos diferentes continentes. Mais tarde, o número de servidores da zona **root** passou a mais de uma centena, espalhados pelas diferentes regiões do mundo onde a concentração de clientes é superior.

Estes servidores, têm os nomes **x.root-servers.net**. (com x a variar de 'a' a 'm') e os seus endereços IPv4 e IPv6 de 13 estão disponíveis publicamente¹. Cada um desses endereços, através de um mecanismo designado por IP Anycasting (ver o RFC 4786), representa muitas instâncias diferentes do mesmo servidor². Qualquer um destes servidores contém uma cópia idêntica da zona **root** e a consulta a qualquer deles deve produzir sempre o mesmo resultado. A implementação deste mecanismo é discutida no final da Secção 18.3.

¹O número 13 foi escolhido pois mensagens UDP de resposta com 512 bytes só podem conter o nome e os endereço de até 13 servidores.

² De forma simplificada, o IP Anycasting consiste em dirigir um pacote para um endereço IP de destino que representa um grupo de servidores, dos quais é selecionado automaticamente o mais “próximo” (em termos da rede) do emissor do pacote.

Qualquer servidor DNS quando arranca, tem de conhecer pelo menos um endereço de um desses servidores. A seguir envia-lhe uma consulta para conhecer a lista completa e os respectivos endereços, e finalmente testa a velocidade de resposta de cada um. Daí em diante passa a dirigir as suas consultas para o servidor `root` mais rápido, ou para qualquer outro a seguir se este não responder.

É desta forma que o DNS garante a fiabilidade do sistema e a sua capacidade de resistir a quaisquer ataques, pois é praticamente impossível atacar simultaneamente os muitos servidores existentes. A Figura 11.5 dá uma imagem aproximada da densidade de distribuição dos servidores da zona `root` pelos diferentes continentes.



Figura 11.5: Distribuição aproximada da concentração de servidores da zona `root` pelas diferentes regiões do mundo

Os servidores da zona `root` apenas conhecem o nome dos servidores dos TLDs e os seus endereços. Algumas dessas zonas, como por exemplo a zona `.com`, tem milhões de sub-domínios, da forma `exemplo.com` e o número de servidores que gerem essa zona é também muito grande, e os mesmos são também acessíveis por *AnyCast*. No entanto, a maioria dos domínios TLD apenas têm algumas centenas de milhar de sub-domínios e o número dos seus servidores é menor. Abaixo mostra-se o resultado da consulta ao DNS sobre qual é a lista de servidores do TLD `.pt` no momento da escrita deste livro.

```
$ dig pt. NS
; <>> DiG 9.8.3-P1 <>> pt NS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54408
;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 0
;
;; QUESTION SECTION:
;pt. IN NS

;; ANSWER SECTION:
pt. 85641 IN NS c.dns.pt.
pt. 85641 IN NS ns.dns.pt.
pt. 85641 IN NS b.dns.pt.
pt. 85641 IN NS ns.dns.br.
pt. 85641 IN NS ns2.dns.pt.
pt. 85641 IN NS ns2.nic.fr.
pt. 85641 IN NS sns-pb.isc.org.

;; Query time: 1 msec
;; ...
```

Conforme se vai descendo na hierarquia, menor é o número de servidores de cada domínio. No entanto, para assegurar a fiabilidade e a tolerância a falhas do DNS, todos os domínios têm pelo menos 2 servidores com autoridade sobre o domínio.

Para permitir uma gestão descentralizada do DNS, a sua base de dados está secionada em **zonas** (*zones*), i.e., um conjunto de um ou mais domínios contíguos. A cada zona estão associados dois ou mais **servidores com autoridade** (*Authoritative Name Servers*) sobre a mesma. Cada zona contém informação de ligação às zonas subordinadas (os nomes e os endereços dos respectivos servidores). Essas informações são designadas por ***Glue Records*** e permitem realizar uma descida iterativa da árvore do DNS.

Servidores *caching only*, consultas recursivas e *resolvers*

A eficiência do DNS está intimamente dependente do sistema de *caching*. Por outro lado, sempre que um cliente pretende consultar o DNS arrisca-se a ter de fazer uma consulta iterativa, tanto mais longa quanto mais baixo na hierarquia estiver o registo que pretende conhecer. Adicionalmente, as *caches* dão melhores resultados quanto maior for a sua partilha por diferentes utilizadores.

Para responder a estes diversos constrangimentos e oportunidades, o DNS tem servidores especiais, designados servidores *caching only*, que são servidores que não gerem zonas, mas que aceitam quaisquer consultas, executam o protocolo iterativo de consulta dos servidores que forem necessários, obtêm a resposta, guardam-na na *cache*, e finalmente respondem ao cliente inicial. Existem muitos servidores deste tipo espalhados pela Internet, geralmente vários em cada ISP e nas instituições com muitos utilizadores da rede.

De forma geral, os computadores quando se ligam a uma rede, recebem o endereço de um ou mais servidores *caching only* da sua rede, e utilizam-nos por defeito para fazer as consultas. Desta forma é organizada uma *cache* partilhada junto dos clientes finais. Naturalmente, se os interesses dos diferentes clientes forem razoavelmente homogéneos, a grande maioria das consultas são respondidas imediatamente pelos servidores *caching only* partilhados. Desta forma o DNS consegue responder às consultas dos biliões de computadores que estão actualmente ligados à Internet.

O protocolo executado entre os clientes finais e estes servidores é idêntico ao protocolo de consulta dos outros servidores. A única diferença é que as consultas levam uma *flag*, chamada *recursion requested*, em que o cliente pede ao servidor que obtenha ele próprio a resposta final à consulta. Estas consultas dizem-se recursivas por oposição às consultas iterativas. A Figura 11.6 ilustra a forma como um cliente final dialoga com um servidor *caching only* para obter o registo ou registos que pretende.

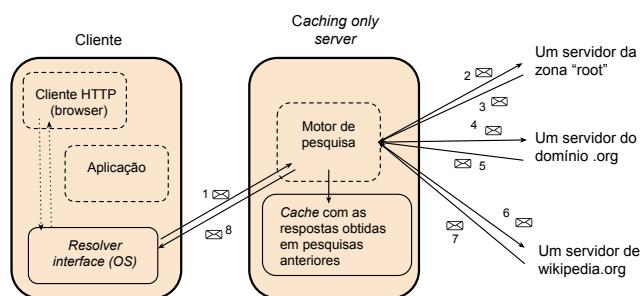


Figura 11.6: Progresso da consulta [name: www.wikipedia.org, type: A] recursiva enviada a um servidor *caching only*. O servidor desconhece a resposta e executa uma consulta iterativa.

Geralmente os clientes utilizam os servidores *caching only* indicados pela sua rede.

No entanto, também é possível utilizar directamente alguns servidores deste tipo espalhados pela Internet e de acesso público (*e.g.*, OpenDNS, GoogleDNS, Level3 Public DNS servers, OpenNIC, ...). Isso só se justifica se se desconfiar dos servidores indicados pela rede a que nos ligámos ou se, por qualquer motivo, se admitir que esses servidores públicos fornecem um melhor serviço que os da rede a que estamos ligados. Voltaremos a este ponto mais adiante.

A Figura 11.6 também ilustra a existência nos clientes finais de uma componente software designada genericamente por *resolver*. Trata-se de uma componente software existente na maioria dos sistemas de operação que fornece a interface do serviço de consulta ao DNS. A maioria dos *resolvers* também podem gerir uma pequena *cache* local ao computador. Apesar de o conceito de *resolver* abranger qualquer componente capaz de fornecer o serviço de consulta ao DNS, a designação aplica-se geralmente a uma componente do sistema de operação que apenas sabe dialogar com servidores *caching only*.

Hoje em dia todos os sistemas de operação dispõem de *resolvers* que são acessíveis programaticamente e que estão ligados a um primeiro servidor *caching only* existente à entrada da rede, como por exemplo num pequeno comutador de pacotes que assegura uma ligação doméstica à Internet. Este, por sua vez, consulta um servidor *caching only* do ISP. Ou seja, os servidores *caching only* podem estar organizados em cadeia.

Para simplificar as consultas ao DNS, e para melhorar a sua eficiência através de *caching*, existem servidores, designados por **servidores recursivos ou caching only**, que aceitam consultas, ditas recursivas, e que executam eles próprios o protocolo iterativo, guardam na sua *cache* a resposta obtida e enviam-na finalmente aos clientes finais.

Uma componente software ou software e hardware que implementa um *front-end* do DNS e permite obter a resposta às consultas pretendidas, diz-se um *resolver*. Todos os sistemas operativos actuais disponibilizam um. Geralmente, os *resolvers* interrogam recursivamente os servidores *caching only* e também implementam uma *cache* local.

Eficiência e coerência do *caching* no DNS

Os mecanismos de *caching* do DNS são tanto mais eficientes quanto maior for o grau de partilha dos servidores de *caching* pelos diferentes clientes, e ainda quanto maior for o TTL dos registos nas diferentes zonas.

No que diz respeito ao grau de partilha dos servidores de *caching*, faz sentido distribuí-los pelas sub-redes da Internet. Assim, a maioria das redes de acesso e institucionais têm um ou mais destes servidores para os seus utilizadores. Também é possível organizar o *caching* de forma mais independente da estrutura interna da rede disponibilizando servidores de *caching* públicos.

Para respeitar o valor do TTL fixado pelos administradores das zonas, é necessário que os registos não fiquem em quaisquer *caches* mais do que esse tempo, desde que foram obtidos de um servidor com autoridade. Caso contrário existe o risco de valores desactualizados dos registos poderem permanecer memorizados algures na Internet.

Quando um registo é passado entre servidores, os servidores sem autoridade sobre o registo, assim como os clientes, decrementam o valor do TTL para que a intenção inicial fixada neste parâmetro não seja violada. Procura-se assim que a intenção do gestor do domínio, ao associar um tempo de vida a cada registo, não seja violada. Ou seja, um registo uma vez obtido, seja qual for a sua proveniência, não pode ficar memorizado mais do que TTL segundos desde que foi recebido, e quando um registo é passado de servidor em servidor, o valor do TTL deve ser decrementado.

No entanto, é necessário ter em atenção que uma resposta a uma consulta obtida de um servidor de *caching* pode estar desactualizada (*stale*) mesmo que o seu TTL não tenha sido violado, pois pode já ter sido alterada num servidor com autoridade desde que foi obtida. Aliás, na sequência de uma actualização, isto será sempre verdade excepto quando o TTL tiver o valor nulo.

Com efeito, na terminologia dos sistemas de gestão de nomes, o valor obtido é designado por sugestão (*hint*). Com efeito, a única garantia que uma *cache* gerida através de TTL pode dar, é que a resposta fornecida foi válida algures no passado, mas que pode, no entretanto, ter sido actualizada. Dada a natureza da maioria da informação presente no DNS, este tipo de garantia é geralmente suficiente, pois na grande maioria das situações a informação fornecida é a correcta, ou se o não for, os clientes apercebem-se disso. Este tipo de mecanismos e ausência de garantias é, no entanto, crítico para a eficiência e adaptação à escala do DNS.

Existe um caso que o DNS trata de forma especial para aumentar a sua eficiência. Inicialmente, quando uma consulta se referia a um nome que não existia, nada era colocado na *cache*, o que autorizava uma consulta semelhante imediatamente a seguir. Posteriormente foi introduzida a noção de *Negative Caching* que permite colocar na cache um pseudo registo a indicar que esse nome não existe. O TTL associado a este tipo de registos é um valor por omissão associado ao registo SOA da zona que não contém o registo procurado.

Alguns servidores de *caching* também introduzem na *cache* informação negativa com um TTL arbitrado, quando nenhum servidores de nomes de uma zona está acessível. Desta forma evitam estar a repetir as consultas a intervalos demasiado curtos.

Para aumentar a eficiência, cada zona do DNS deve ter mais do que um servidor, pois desta forma aumenta-se a sua resistência a avarias. A mesma opção também permite realizar distribuição de carga entre os vários servidores. Como estes estão espalhados por diferentes regiões da Internet, os servidores de *caching* podem melhorar a eficiência ao tentarem usar os servidores que lhe estão mais próximos. Uma possível estratégia é usar uma política de rotação (*round robin*) e ir memorizando o tempo de resposta dos diferentes servidores, ordenando-os por tempo crescente de resposta.

Os mecanismos de *caching* no DNS são fundamentais para a sua eficiência, no entanto, a única garantia que uma *cache* gerida através de TTL pode dar, é que a resposta fornecida era válida algures no passado, mas que pode, no entretanto, já ter sido actualizada. Dada a natureza da maioria da informação presente no DNS, este tipo de garantia é geralmente suficiente, pois na grande maioria das situações a informação fornecida é válida, ou se o não for, os clientes apercebem-se disso.

Utilização do DNS para distribuição de carga

O DNS é um sistema distribuído muito eficiente pois, como vimos acima, implementa mecanismos de distribuição de carga e de *caching*. Adicionalmente, o DNS é ele próprio usado para distribuir carga.

O primeiro exemplo desta utilização consiste na associação de diversos registos de servidores de correio electrónico (RRs MX) a um mesmo domínio de correio. Por um lado podem ser associados diversos servidores ao mesmo domínio (diversos RRs do tipo MX associados ao mesmo nome) com prioridades diferentes, o que aumenta a resistência a avarias. Por outro lado, se existirem diversos servidores com a mesma prioridade, isso permite realizar distribuição de carga. A consulta abaixo ilustra este tipo de mecanismos pois mostra que associados ao domínio `fct.unl.pt` existem vários servidores, alguns com diferentes prioridades, mas outros com prioridades iguais.

```
$ dig fct.unl.pt MX
; <>> DiG 9.8.3-P1 <>> fct.unl.pt MX
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29827
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 0
;;
;; QUESTION SECTION:
;fct.unl.pt. IN MX
;;
;; ANSWER SECTION:
fct.unl.pt. 10800 IN MX 20 ALT2.ASPMX.L.GOOGLE.COM.
fct.unl.pt. 10800 IN MX 30 ASPMX2.GOOGLEMAIL.COM.
fct.unl.pt. 10800 IN MX 30 ASPMX3.GOOGLEMAIL.COM.
fct.unl.pt. 10800 IN MX 10 ASPMX.L.GOOGLE.COM.
fct.unl.pt. 10800 IN MX 20 ALT1.ASPMX.L.GOOGLE.COM.

;; Query time: 34 msec
;; ...
```

Com o objectivo de também facilitar a distribuição de carga, o DNS permite associar ao mesmo nome diferentes registos do tipo A ou AAAA, pondo em evidência que o mesmo serviço tem mais do que um servidor com diferentes endereços IP. Isto também facilita a distribuição dos clientes do serviço pelos diferentes servidores do mesmo. Para facilitar este objectivo, os servidores com autoridade, assim como os de *caching*, devem rodar a ordem pela qual fornecem na resposta os registos do tipo A e AAAA a diferentes clientes.

Tolerância a falhas e convergência do DNS

Quando existem diversos servidores com autoridade sobre a mesma informação, diz-se que esses servidores replicam a informação. Todos os servidores com autoridade sobre uma zona, replicam todos os registos da zona.

O DNS dispõe de um mecanismo para assegurar a replicação de uma zona. Trata-se de uma mecanismo de primário / secundários. Entre os servidores com autoridade sobre a zona, um é considerado o primário, e os restantes os secundários. As modificações da zona são repercutidas pelo seu administrador apenas no servidor primário. Depois, um sistema automático, sincroniza os secundários com o primário.

Existem duas formas diferentes de executar esta sincronização. Na primeira, os servidores secundários contactam periodicamente o primário a perguntar se houve alguma alteração que tenham que repercutir na sua cópia local. Caso tenha sido o caso, obtêm o ou os valores actualizados. Na segunda, compete ao servidor primário avisar os secundários para que venham buscar as alterações, ou eventualmente difundi-las imediatamente pelos secundários.

Na verdade, a segunda forma de realizar a sincronização tem de ser sempre complementada com mecanismos usados pela primeira forma, pois quando um servidor secundário está inacessível, não pode receber notificações do primário sobre actualizações. Assim, os secundários devem contactar o primário a indagar sobre se houve alterações logo que voltem a estar operacionais, ou que a conectividade com o primário tenha sido restaurada.

O DNS optou inicialmente pela primeira forma e a sincronização por omissão dos servidores com autoridade é realizada obrigando os servidores secundários a comunicarem periodicamente com o primário. Se existirem alterações desde o último contacto, fazem a sua transferência. Posteriormente foi introduzido o mecanismo complementar de notificação dos secundários pelo primário para que realizem a sincronização imediatamente.

Assim, as inconsistências no DNS podem não só ser devidas à gestão das *caches* através de um mecanismo de TTL, mas também por eventuais atrasos na sincronização dos servidores com autoridade. Com efeito, a inconsistência permanece igualmente

enquanto todos os secundários não actualizarem a zona. Até lá, respondem com autoridade mas fornecem informações desactualizadas.

Os parâmetros usados pelos servidores secundários para se sincronizarem com o primário são passados aos secundários pelo administrador através do RR SOA. Este contém diversas informações como: um número de versão dos registos, TTLs por defeito, periodicidade da consulta ao primário pelos secundários, *etc.* Um mecanismo de transferência integral de zonas entre servidores faz também parte do protocolo.

Nos domínios de grande dimensão, com milhões de registos, a sincronização entre servidores com autoridade recorre provavelmente a mecanismos mais sofisticados, mas estes não se encontram geralmente documentados publicamente.

A gestão da validade e actualização de informação replicada é um dos tópicos mais importantes, estudado há muitos anos, nos sistemas distribuídos, com especial realce para todos os sistemas distribuídos replicados modernos [Vogels, 2009], como bases de dados e outros sistemas equiparáveis.

O DNS tem uma escala avassaladora, com biliões de clientes, e responde de forma satisfatória às suas consultas, através da utilização intensiva de *caching* gerida através de TTLs, e de replicação dos servidores com autoridade através de um mecanismo primário/secundários. Os servidores secundários não estão só de reserva para o caso de o primário ter uma avaria, mas também respondem a pedidos, o que também pode introduzir inconsistências suplementares momentâneas. No entanto, só os servidores primários aceitam actualizações.

As garantias sobre a consistência das respostas obtidas às consultas ao DNS são classificadas em sistemas distribuídos por *Eventual Consistency* (que poderia ser traduzido por consistência a prazo). Estas garantias são suficientes no caso do DNS, dada a natureza da informação neste memorizada, e o baixo risco corrido pelos clientes se usarem informação desactualizada.

As mensagens do protocolo do DNS

As mensagens de pedido e resposta do protocolo do DNS podem ser transportadas em UDP ou TCP. Em ambos os casos, a porta do protocolo é a porta 53. A Figura 11.7 mostra o formato único dessa mensagem pois, quer as mensagens de pedido, quer as de resposta, têm o mesmo formato genérico. No exemplo, o transporte usado é o UDP.

O Campo **Identification** facilita que um servidor *caching only* faça corresponder as mensagens de pedido e de resposta. Este campo é copiado pelo servidor que responde. O Campo **Flags** permite passar informação de controlo. Alguns exemplos são: as *flags* REQUEST, REPLY e RECURSION REQUESTED com significados óbvios, a *flag* NOTIFY permite a um servidor primário avisar os secundários de que houve alterações na zona, a *flag* TRUNCATION permite a um servidor indicar que uma resposta enviada sobre UDP está incompleta, a *flag* RCODE permite a um servidor indicar códigos de resposta (No ERROR, Name ERROR, *etc.*), *etc.*

Os restantes campos especificam o número de perguntas ou respostas de cada tipo. A seguir ilustra-se uma utilização destas diferentes secções. Através do exemplo é fácil verificar para que servem as secções QUESTION e ANSWER. A secção AUTHORITY mostra o nome dos servidores autoritários para a informação da resposta. Finalmente, a secção ADDITIONAL apresenta informação que o servidor admite que possa vir a ser útil ao cliente para lhe evitar consultas suplementares. Esta informação também é designada de *hints* ou sugestões nos RFCs do DNS.

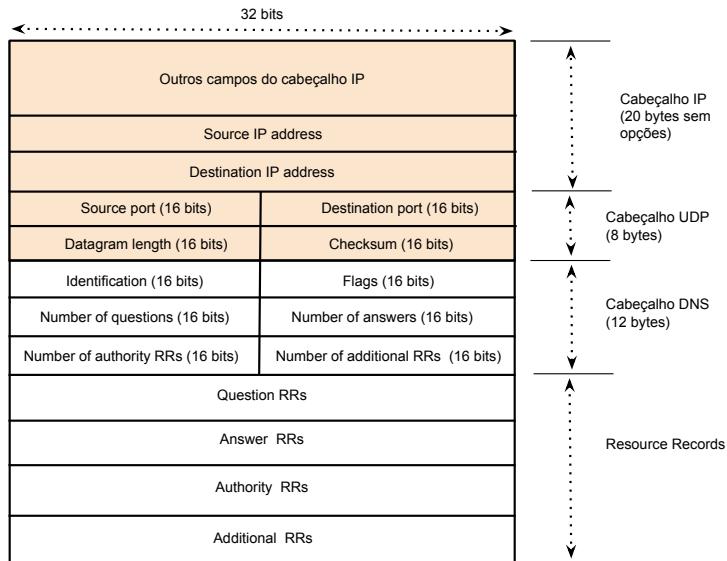


Figura 11.7: Mensagem de pedido e resposta ao DNS transportada pelo protocolo UDP

```
$ dig @dns1.fct.unl.pt fct.unl.pt MX

; <>> DiG 9.8.3-P1 <>> @dns1.fct.unl.pt fct.unl.pt MX
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19859
;; flags: qr aa rd; QUERY: 1, ANSWER: 5, AUTHORITY: 3, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;fct.unl.pt. IN MX

;; ANSWER SECTION:
fct.unl.pt. 10800 IN MX 10 ASPMX.L.GOOGLE.COM.
fct.unl.pt. 10800 IN MX 20 ALT1.ASPMX.L.GOOGLE.COM.
fct.unl.pt. 10800 IN MX 20 ALT2.ASPMX.L.GOOGLE.COM.
fct.unl.pt. 10800 IN MX 30 ASPMX2.GOOGLEMAIL.COM.
fct.unl.pt. 10800 IN MX 30 ASPMX3.GOOGLEMAIL.COM.

;; AUTHORITY SECTION:
fct.unl.pt. 86400 IN NS dns1.fct.unl.pt.
fct.unl.pt. 86400 IN NS dns2.fct.unl.pt.
fct.unl.pt. 43200 IN NS ns3.unl.pt.

;; ADDITIONAL SECTION:
dns1.fct.unl.pt. 86400 IN A 193.136.126.101
dns2.fct.unl.pt. 86400 IN A 193.136.126.102
```

A administração dos domínios e criação de um subdomínio

A Internet Corporation for Assigned Names and Numbers (ICANN) é o organismo responsável pela gestão da zona `root` e pela criação e aceitação dos domínios TLDs. Inicialmente, este organismo respondia perante o Governo dos EUA mas, com o passar dos anos, passou a ser gerido e responder perante um grande conjunto de organizações. Trata-se de uma forma cooperativa de gestão, designada *multistakeholder*, com

representantes de muitos governos, e tendo por observadores associações de estados, associações de operadores, associações de outras empresas relacionadas com a Internet, e diversos organismos de investigação de grande dimensão. Para além de ser responsável pelo DNS, a ICANN é também responsável, usando um modelo semelhante, pela gestão do espaço de endereçamento IP.

Inicialmente os GTLDs eram poucos: `edu`, `com`, `net`, `org`, *etc.* Em 2005 foram introduzidos os seguintes: `eu`, `asia`, `travel`, `mobi` e `cat`. Recentemente foi liberalizada a criação de mais GTLDs, utilizando para esse efeito um concurso aberto e exigindo um pagamento, relativamente avultado, aos proponentes do domínio. O número de GTLDs já ultrapassa o milhar.

A maioria dos CTLDs é gerida por uma entidade nomeada por um governo ou na base de associações de organismos interessados. Os domínios GTLDs iniciais são geridos pela ICANN (ou delegados por esta em associações ou empresas). Os mais recentes são geridos pelas organizações que adquiriram o domínio. Em qualquer caso, as organizações responsáveis por um domínio são designadas por *registries* do domínio, gerem a base de dados central do mesmo, e directa ou indirectamente, *e.g.*, em *outsourcing*, os servidores com autoridade sobre o mesmo.

Nos domínios pequenos ou institucionais, geralmente a instituição com autoridade sobre o domínio também gere os pedidos de registo de sub-domínios e os servidores com autoridade sobre o mesmo. Por exemplo, a Universidade Nova de Lisboa gera o domínio `unl.pt` e os seus servidores, mas delega os domínios das suas unidades orgânicas nas mesmas.

No entanto, quando os domínios são muito grandes (a maioria dos GTLDs e os CTLDs de países com muitos sub-domínios), outras organizações podem actuar como *registars*, *i.e.*, revendedores do serviço de registo de domínios. De forma geral, a maioria dos *registars* fornecem igualmente aos seus clientes (os donos dos domínios) serviços suplementares como a gestão dos servidores com autoridade sobre o domínio, correio electrónico, *etc.* A este modelo em que a gestão dos registos num domínio é subcontratada a outras entidades, chama-se o modelo *registry-registrar*.

A seguir ilustra-se como se processa a criação de um domínio fictício, exactamente o domínio `ficticio.pt`. A entidade que queira registar este domínio tem de verificar primeiro se este ainda não existe. Tem também de verificar se não está a entrar em colisão com alguma outra entidade que reclame o direito legal à utilização desse domínio³.

Em seguida, admitindo que é a entidade criadora que vai gerir os servidores do domínio, terá que criar uma base de dados inicial do domínio com um conjunto de registos. Por exemplo:

```
ficticio.pt. 86400 IN SOA dns1.ficticio.pt. admin.ficticio.pt.
                           2016031501 10800 1800 691200 3600
ficticio.pt. 3600 IN NS dns1.ficticio.pt.
ficticio.pt. 3600 IN NS dns2.ficticio.pt.
ficticio.pt. 3600 IN MX 10 mail1.ficticio.pt.
ficticio.pt. 3600 IN MX 20 mail2.ficticio.pt.
mail1.ficticio.pt. 3600 IN A 100.100.100.5
mail2.ficticio.pt. 3600 IN CNAME www.ficticio.pt.
www.ficticio.pt. 3600 IN A 100.100.100.10
dns1.ficticio.pt. 3600 IN A 100.100.100.1
dns2.ficticio.pt. 3600 IN A 100.100.100.12
.... etc.
```

O RR SOA tem um TTL de 24 horas (86400 segundos) e em seguida lista o nome do servidor primário, o endereço de correio electrónico do administrador do domínio e, na linha seguinte, o número de versão da base de dados, *i.e.*, um inteiro sempre crescente, que no exemplo é composto como uma data seguido de um número (2016 03 15 01). Seguem-se diversos parâmetros do mecanismo de replicação entre o servidor

³Trata-se de um problema legal que ultrapassa o âmbito desta discussão.

primário e os secundários, e finalmente o SOA termina com o TTL por omissão que vale 1 hora (3600 segundos).

A seguir vêm outros RRs. Primeiro são indicados os nomes dos dois servidores de nomes do domínio (RRs do tipo NS). Mais no fim são indicados os seus endereços IP (RRs do tipo A). Em seguida indica-se o nome de dois servidores de correio electrónico (RRs do tipo MX). Mais abaixo são indicados os endereços IP dos servidores `mail1.ficticio.pt` e `www.ficticio.pt`. Finalmente é indicado que o nome `mail2.ficticio.pt` é um *alias* para o nome `www.ficticio.pt` (RR CNAME) pois provavelmente o mesmo computador funciona como servidor de Web e servidor de *backup* para o correio electrónico. Normalmente seguir-se-iam outros RRs, geralmente indicando o endereço IP de outros computadores.

Depois, a base de dados de registo deveria ser colocada na base de dados do servidor primário, o servidor secundário deveria ser sincronizado com este e, finalmente, o servidor primário do domínio `pt` deveria passar a incluir os seguintes *glue records*:

```
ficticio.pt. 3600 IN NS dns1.ficticio.pt.
ficticio.pt. 3600 IN NS dns2.ficticio.pt.
dns1.ficticio.pt. 3600 IN A 100.100.100.1
dns2.ficticio.pt. 3600 IN A 100.100.100.12
```

para que a criação do domínio passasse a ter efeito. Esta última parte da criação do domínio teria de ser intermediada por um dos *registrars* do domínio `pt`. Na maioria dos casos, este poderia ele próprio fornecer os servidores do domínio e tratar das suas parametrizações mediante pagamento. Na prática, alguns servidores DNS são servidores com autoridade sobre centenas de zonas.

Segurança do DNS

Inicialmente o DNS não tinha mecanismos de segurança. Mais tarde começaram a aparecer diversos tipos de problemas. Destes, o mais grave consiste em tentar corromper a visão que um cliente tem dos registos de um domínio, com o objectivo de, por exemplo, o dirigir para um servidor falso. Por exemplo, admitindo que o servidor `www.ficticio.pt` teria o endereço 200.200.200.1, ao invés de 100.100.100.10.

Isto pode ser conseguido corrompendo um dos servidores de `pt`, modificando os seus *glue records* sobre o domínio `ficticio.pt`, e desviando os clientes para um servidor DNS alternativo, ou, o que é geralmente mais fácil, corrompendo a informação fornecida por algum servidor de *caching* quando questionado sobre nomes do domínio `ficticio.pt`. Uma das técnicas mais usadas consiste em tentar enviar a um servidor de *caching* respostas fictícias incluindo *hints* falsos. Genericamente, este tipo de práticas costuma-se designar por *DNS cache poisoning*.

Estes exemplos mostram que é fundamental apenas usar servidores de *caching* confiáveis e bem geridos. Esta pode ser uma das razões que leva alguns utilizadores a preferirem usar servidores *caching only* públicos, mantidos por organizações bem conhecidas e confiáveis. Como os computadores usam geralmente *resolvers* dependentes de servidores de *caching*, estes têm um grande poder sobre a forma como os computadores finais “vêm” à Internet, pois têm a possibilidade de filtrar ou modificar as respostas às consultas feitas. Às vezes estes mecanismos são usados para introduzir bloqueios abusivos, porque não autorizados judicialmente, de acesso a servidores controversos.

Para combater este tipo de problemas de segurança, o DNS tem actualmente um conjunto de extensões de segurança, que permitem que as respostas dos servidores com autoridade venham assinadas criptograficamente através das chaves da zona (RFC 4033). Os clientes, verificando a veracidade das chaves públicas, e a integridade das assinaturas, podem assim verificar a autenticidade das respostas recebidas.

Este tipo de mecanismos de segurança estão a ser progressivamente adoptados. No entanto, dado que introduzem uma complexidade suplementar, incluindo uma maior

complexidade de gestão, pois as chaves dos domínios têm de ser certificadas pelos domínios ascendentes até à `root` do sistema, a sua adopção tem sido mais lenta que o desejável.

A possibilidade de corrupção da *cache* DNS põe em evidência que o acesso a servidores Web contendo informação crítica deve ser feita usando canais seguros que verifiquem se o servidor em questão é capaz de provar que é o servidor que se espera (usando URLs da forma `https:// ...`). Este é mais um exemplo de que um cliente tem geralmente a possibilidade de verificar se a informação fornecida pelo DNS é consistente e verídica.

Um aspecto que também convém ter em atenção é que é frequente existirem inconsistências no DNS devido à existência de *glue records* errados. Com efeito, se os servidores de um domínio forem alterados, ou se mudarem de endereço IP, é necessário actualizar igualmente os *glue records* presentes no servidor primário do domínio pai. Este tipo de erros é mais frequente nos pequenos domínios, geridos manualmente, do que nos TLDs de grande dimensão, cuja gestão está automatizada e sujeita a procedimentos de segurança.

Finalmente, para fecharmos esta breve referência aos problemas de segurança do DNS, vamos-nos referir a um problema similar mas diferente, conhecido pela designação de *alternative roots*. Como já foi referido, a grande maioria das consultas iterativas começa num servidor de `root` e vai seguindo os *glue records* por este (e os seus descendentes) indicados, até se chegar a um servidor do domínio procurado.

Se este processo iterativo começar num servidor de `root` alternativo, é como se a pesquisa tivesse lugar num espaço de nomes alternativo. Por ativismo político, ou para contestação daquilo que é percepcionado pelos próprios como um monopólio sobre os nomes da Internet exercido pelo ICANN, têm sido realizadas várias tentativas de introdução de *roots* alternativas. O leitor interessado em seguir este tipo de polémicas pode, por exemplo, começar por ler a posição de princípio do IAB (*Internet Architecture Board*) [Board, 2000] sobre o assunto.

O DNS levanta importantes problemas de segurança que têm começado a ser endereçados através da extensão DNSSEC. Trata-se de uma extensão para introdução de assinaturas criptográficas nos registos devolvidos como resposta às consultas. Esta extensão baseia-se em assinaturas com chaves privadas, e verificadas com chaves públicas, que têm de ser certificadas até à raiz do sistema. Esta extensão introduz complexidades técnicas e de gestão suplementares que justificam a lentidão da sua adopção.

Outro aspecto a ter em atenção é que os clientes que usam um dado conjunto de servidores de *caching* dependem da confiabilidade destes para obterem respostas confiáveis. Alguns destes servidores podem filtrar ou modificar abusivamente a informação registada no conjunto do DNS.

11.4 Resumo e referências

Resumo

O DNS é uma gigantesca base de dados distribuída que associa nomes a atributos de vários tipos. Cada registo da base de dados diz-se um **RR** (*Resource Record*), e associa uma informação de um certo tipo, o chamado **RRT** (*Resource Record Type*), a um nome. Os RRTs mais comuns são os que permitem associar endereços IP aos nomes do DNS, mas existem muitos outros tipos, incluindo aqueles que mantêm a informação interna de parametrização do próprio DNS (NS, SOA, ...).

Os registos existentes no DNS são de consulta pública. A sua actualização só pode ser realizada pelos administradores dos domínios. A excepção são os endereços IP de computadores que fornecem serviços mas cujos endereços mudam com alguma frequência. Para essa actualização existe um extensão específica, que permite ao computador actualizar o seu endereço no DNS.

Associado a cada RR existe um **TTL** (*Time To Live*) o qual indica o limite máximo que o registo pode permanecer numa *cache* sem ser actualizado. Dado que a maioria da informação contida no DNS é muito estável, geralmente os TTLs que lhe estão associados correspondem a várias horas. Esses valores determinam o período máximo em que podem existir valores desactualizados *cached* em clientes do DNS. Assim, o TTL é determinante para controlar o tempo máximo de convergência dos valores fornecidos aos clientes após alguma alteração de RRs.

Por omissão, o protocolo de pesquisa do DNS é um protocolo cliente/servidor iterativo, através do qual o cliente vai repetindo a pergunta a diferentes servidores, até chegar ao servidor que conhece a resposta. O protocolo funciona sobre UDP e TCP mas, por razões de eficiência, geralmente é executado sobre UDP.

Na ausência de informação adicional, o primeiro servidor interrogado é um servidor da zona **root**. Caso este não conheça a resposta, responde com *referral information*, *i.e.*, o nome e endereço de servidores abaixo na hierarquia, mais próximos da zona a que pertence o registo consultado.

Para permitir uma gestão descentralizada do DNS, a sua base de dados está partitionada em **zonas** (*zones*), *i.e.*, um conjunto de um ou mais domínios contíguos. A cada zona estão associados dois ou mais **servidores com autoridade** (*Authoritative Name Servers*) sobre a mesma. Cada zona contém informação de ligação às zonas subordinadas (os nomes e os endereços dos respectivos servidores). Essas informações são designadas por ***Glue Records*** e permitem realizar uma descida iterativa da árvore do DNS.

Para simplificar as consultas ao DNS, e para melhorar a sua eficiência através de *caching*, existem servidores, designados por **servidores recursivos ou caching only**, que aceitam consultas, ditas recursivas, e que executam eles próprios o protocolo iterativo, guardam na sua *cache* a resposta obtida e enviam-na finalmente aos clientes finais.

Uma componente software ou hardware que implementa um *front-end* do DNS e permite obter a resposta às consultas pretendidas, diz-se um *resolver*. Todos os sistemas operativos actuais disponibilizam um. Geralmente, os *resolvers* interrogam recursivamente os servidores *caching only* e também implementam uma *cache* local.

Estes mecanismos de *caching* do DNS são fundamentais para a sua eficiência, no entanto, a única garantia que uma *cache* gerida através de TTLs pode dar, é que o registo obtido como resposta, era válido algures no passado, mas que pode posteriormente já ter sido actualizado. Dada a natureza da maioria da informação presente no DNS, este tipo de garantia é geralmente suficiente, pois na grande maioria das situações a informação fornecida é válida, ou se o não for, os clientes apercebem-se disso.

O DNS tem uma escala avassaladora, com biliões de clientes, e responde de forma satisfatória às suas consultas, através da utilização intensiva de *caching*, gerida através de TTLs, e de replicação dos servidores com autoridade através de um mecanismo primário/secundários.

As garantias sobre a consistência das respostas obtidas às consultas ao DNS são classificadas em sistemas distribuídos por *Eventual Consistency* (que poderia ser traduzido por consistência a prazo). Estas garantias são suficientes no caso do DNS, dada a natureza da informação neste memorizada, e o baixo risco corrido pelos clientes se usarem informação desactualizada.

O DNS levanta importantes problemas de segurança que têm começado a ser endereçados através da extensão DNSSEC. Trata-se de uma extensão para introdução

de assinaturas criptográficas nos registos devolvidos como resposta às consultas. Esta extensão introduz complexidades técnicas e de gestão suplementares que justificam a lentidão da sua adopção.

Outro aspecto a ter em atenção é que os clientes que usam um dado conjunto de servidores de *caching* dependem da confiabilidade destes para obterem respostas confiáveis. Alguns destes servidores podem filtrar ou modificar abusivamente a informação registada no conjunto do DNS.

Alguns dos termos introduzidos no capítulo são a seguir passados em revista. Entre parêntesis figura a tradução mais comum em língua inglesa.

Principais conceitos

Nomes (*names*) Sequências de símbolos que permitem designar entidades. Geralmente têm um significado mnemónico para facilitar a sua memorização pois codificam propriedades da entidade que designam.

Identificadores únicos (UIDs) (*Unique Identifiers*) Nomes “puros” que discriminam as entidades a que estão associados de forma única no tempo e no espaço e que geralmente não revelam propriedades das entidades a que estão associados.

Endereços (*addresses*) Sequências de símbolos que permitem indicar como aceder num certo momento à entidade a que estão associados.

Sistemas de designação (*naming systems*) Sistemas que implementam catálogos de tradução ou mapeamento de nomes em propriedades ou atributos das entidades a que os nomes estão associados.

DNS (*Domain Name System*) Base de dados distribuída que associa nomes a atributos de vários tipos e que é utilizado principalmente para registar o endereço IP dos servidores existentes Internet.

RR (*Resource Record*) designação dada a cada registo da base de dados do DNS.

TTL (*Time To Live*) Parâmetro associado a cada RR que indica o tempo máximo que este pode ficar *cached*.

Consulta iterativa (*iterative query*) consulta em que o cliente vai percorrendo diversos servidores DNS até chegar ao servidor que conhece o RR solicitado.

Zona (*zone*) Secção horizontal da base de dados, geralmente contendo toda a informação de um ou mais domínios contíguos e sob a mesma autoridade administrativa. A zona associada ao domínio da raiz do DNS chama-se a **root zone**.

Glue records Registos contidos em cada zona e que permitem dirigir um cliente para servidores de zonas subordinadas à do servidor que responde.

Servidores com autoridade (*authoritative servers*) Servidores que contêm uma cópia integral dos registos de uma zona. Esses registos eram válidos e oficiais quando o servidor secundário contactou pela última vez o servidor primário.

Servidores de caching (*caching only servers*) Servidores que aceitam consultas, ditas recursivas, e que executam eles próprios o protocolo iterativo, guardam na sua *cache* a resposta obtida, e enviam-na finalmente aos clientes finais.

Consulta recursiva (*recursive query*) Consulta dirigida a um servidor em que o cliente solicita ao servidor para obter a resposta final à consulta, executando uma consulta iterativa se necessário.

DNSSEC (*DNS SEcurity*) Acrónimo de um conjunto de extensões ao DNS, que permitem que as respostas dos servidores venham assinadas com as chaves do gestor do domínio da resposta. O objectivo da extensão é combater uma forma de ataque ao DNS conhecida como envenenamento da *cache*.

Referências

O primeiro sistema de designação para sistemas em rede de grande escala desenvolvido no mundo da investigação, mas que teve grande impacto teórico e prático, foi o sistema Grapevine [Birrell et al., 1982]. Este sistema e a experiência obtida com a sua operação foi uma fonte de inspiração para o desenho do DNS. O livro [Liu, 2002] é uma referência clássica para a compreensão e gestão operacional do mais popular software de servidores do DNS.

Depois do DNS foram desenhados muitos outros sistemas de designação para redes de computadores e sistemas distribuídos. Geralmente procurou-se generalizar o tipo de atributos e a sofisticação das consultas que esses sistemas permitiam. O objectivo foi sempre tentar aproximar esses sistemas de uma verdadeira base de dados distribuída de diversas entidades, como por exemplo: instituições, departamentos, pessoas, serviços, servidores, listas de correio electrónico, *etc.* com uma lista arbitrária e extensível de atributos, e permitir consultas de todo o tipo como por exemplo seleção de entidades com um dado atributo particular.

O exemplo mais conhecido é o sistema X.500 [Chadwick, 1994], o qual é descrito num conjunto extenso de normas. É habitual dizer que se o DNS é parecido com uma lista telefónica, o X.500 permite a implementação de “páginas amarelas”, na medida em que permite pesquisas por atributos, para além de pesquisas por nome: *e.g.*, qual é a lista das pessoas com a categoria de **coordenador departamental** da instituição **Banco Maravilha**?

Geralmente considera-se que a norma X.500 e as suas inúmeras normas associadas são demasiado complexas e que a utilização inicialmente prevista era irrealista. Por este motivo as mesmas não são muito utilizadas. No entanto, o trabalho realizado com o X.500 teve vários resultados muito importantes.

O sistema LDAP (*Lightweight Directory Access Protocol*) [Sermersheim, 2006; Howes et al., 2003] é uma versão simplificada de X.500 que está na base dos sistemas de directório distribuído implementados nas redes internas de inúmeras instituições. A norma X.509 é utilizada para codificação dos certificados de chaves públicas usados, por exemplo, com o protocolo TLS (*Transport Layer Security*), que permite estabelecer canais seguros sobre os protocolos de transporte e que necessita de certificados X.509 das chaves públicas dos extremos do canal.

Apontadores para informação na Web

- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- <http://www.icann.org> – É o *site* oficial da ICANN.
- <http://www.isc.org/downloads/bind> – É o *site* oficial do software de servidor de domínios bind, um dos mais utilizados na prática para implementar servidores DNS.
- <http://www.dnssec.net> – É um *site* com imenso material sobre as normas DNSSEC e muita documentação técnica sobre o DNS.

11.5 Questões para revisão e estudo

1. Verdade ou mentira?

- (a) Quando um computador se liga à rede, o DNS permite que este adquira o seu endereço IP.
- (b) O DNS é uma base de dados distribuída que associa nomes a atributos (como por exemplo endereços IP).

- (c) A interacção com os servidores com autoridade sobre as diferentes zonas DNS, para a realização de uma consulta iterativa, é equivalente a um protocolo do tipo P2P pois esses servidores comportam-se igualmente como clientes durante a execução da consulta.
- (d) Os servidores da zona `root` do DNS fazem *caching* dos registo DNS com os endereços IP dos servidores Web da Internet.
- (e) Os servidores da zona `root` do DNS estão necessariamente envolvidos em todas as consultas envolvendo um TLD (*e.g.*, `pt`, `fr`, `com`, `net`, `org`, `eu`, *etc.*).
- (f) Os servidores da zona `root` do DNS estão necessariamente envolvidos em todas as consultas envolvendo qualquer domínio.
- (g) O protocolo de consulta do DNS é um protocolo do nível transporte implementado por servidores especiais.
- (h) O protocolo de consulta do DNS é um protocolo do nível rede implementado por servidores especiais.
- (i) O protocolo de consulta do DNS é um protocolo de nível aplicacional, implementado ao nível de bibliotecas *user level* dos sistemas de operação e em colaboração com servidores especiais.
- (j) Quando se acede à página WWW com o URL `http://100.100.100.2` o *browser* Web não necessita de utilizar o DNS para descobrir o servidor para o qual tem de abrir uma conexão.
- (k) Quando o servidor primário de um domínio deixa de funcionar, deixa de ser possível conhecer os endereços IP dos computadores com nomes nesse domínio.
- (l) A biblioteca *resolver* disponível nos sistemas de operação nunca desencadeia o envio de pacotes UDP para servidores DNS porque só devolve informação já *cached* localmente.
- (m) A biblioteca *resolver* disponível nos sistemas de operação abre sempre uma conexão TCP para um servidor com autoridade para obter respostas aos pedidos das aplicações.
- (n) Os registos DNS *cached* pelos servidores *caching only* são descartados quando passa um período de tempo dependente de parâmetros fixados pelo administrador da zona à qual os registos estão associados.
- (o) Os registos DNS *cached* pelos servidores *caching only* são descartados quando passa um período de tempo dependente de parâmetros fixados pelo administrador da zona `root`.

2. Quais das seguintes questões são verdadeiras?

- (a) As aplicações consultam o DNS abrindo um canal TCP para o servidor que mantém a base de dados do DNS.
- (b) As aplicações costumam consultar o DNS abrindo canais TCP para vários servidores DNS até encontrarem o que conhece a resposta.
- (c) As aplicações consultam o DNS usando geralmente o transporte UDP porque, apesar de os pacotes UDP se poderem perder, a recepção de uma resposta assinala que o servidor DNS recebeu o pedido, e o cliente a resposta do servidor.
- (d) As aplicações consultam o DNS usando o transporte UDP porque este serviço não é compatível com a existência de *jitter* na rede.

- (e) As aplicações consultam o DNS usando o transporte UDP porque este serviço garante a fiabilidade através de um protocolo do tipo *stop & wait*.
3. Quais das seguintes questões são verdadeiras?
- O protocolo do DNS apenas envia um só pedido em cada mensagem de consulta.
 - Para obter o nome dos servidores de correio electrónico de um domínio *D*, assim como os respectivos endereços IP, é sempre necessário enviar mais do que uma mensagem de consulta a um servidor DNS da zona *D*.
 - Todas as registo contidos no DNS têm um TTL associado que determina o tempo máximo de *caching* indicado pelo gestor do domínio do registo. Esta forma de gestão por *soft state* da *cache* garante que os clientes do DNS obtém sempre a última versão de cada registo.
4. Suponha que o ISP *bigisp.net* recebe grandes quantidades de correio electrónico dirigido aos seus clientes, *i.e.*, os utilizadores com endereços de correio electrónico da forma *user@bigisp.net*. O ISP tem 3 servidores de correio electrónico: *big1/2/3.bigisp.net*. Explique que outros registo DNS o domínio *bigisp.net* necessita, além dos indicados a seguir, para que as mensagens de correio electrónico que são dirigidas aos seus utilizadores sejam distribuídas pelos seus 3 servidores de correio electrónico.

```
big1.bigisp.net. 1000 IN A 100.100.100.100
big2.bigisp.net. 1000 IN A 100.100.100.110
big3.bigisp.net. 1000 IN A 100.100.100.120
```

5. Admitindo que todos os CDs com músicas têm um código associado (por hipótese com 10 dígitos), invente um método eficiente para descobrir através do DNS os endereços IP de servidores com as músicas de um CD conhecido pelo seu código. Critique esta solução.
6. Quais as vantagens e as desvantagens de utilizar servidores DNS *caching only* públicos (*e.g.*, OpenDNS, GoogleDNS, Level3 Public DNS servers, OpenNIC, ...) para consultar o DNS? Consulte na Web a informação prestada pelos próprios.
7. O servidor DNS primário do domínio *paradise.com* tem na sua tabela o registo:

```
true.paradise.com. 1000 IN A 100.100.100.100
```

Num determinado momento foi feita uma actualização desse registo que passou a ter o valor:

```
true.paradise.com. 2000 IN A 100.100.100.200
```

Responda às seguintes questões:

- Quanto tempo (em segundos) é necessário para que uma consulta deste registo devolva sempre o novo endereço IP, seja qual for o local da Internet em que a mesma é realizada? Escolha apenas uma das seguintes opções:
 ≈ 10 ≈ 3000 exatamente 2000 exatamente 1000 mais de 3000 nunca ou imediatamente
 - Você é o gestor do domínio acima. Para diminuir a probabilidade de que na altura em que procede à actualização, se corra o risco de alguns clientes não conseguirem aceder ao serviço, na véspera resolve mudar um parâmetro associado ao registo. Que parâmetro é esse e que valor ele deve tomar?
8. O servidor primário do domínio *utopia.com* tem o seguinte registo:

```
verdade.utopia.com. 3600 IN A 100.100.100.100
```

No momento t_1 esse registo foi actualizado para:

```
verdade.utopia.com. 7200 IN A 100.100.100.200
```

Quanto tempo em segundos é necessário para que a consulta ao registo necessariamente devolva o novo endereço IP, seja qual for o local da Internet em que essa consulta é feita? Escolha apenas uma das seguintes opções: $\approx t_1 + 3600$
 $\approx t_1$ $\approx t_1 + 3600 + 7200$ $\approx t_1 + 7200$ $\approx t_1 + 7200 - 3600$ ou nunca

9. Suponha que o tempo médio para executar uma consulta ao servidor *caching only* da sua faculdade é desprezável, mas que esse tempo é de 100 milissegundos quando um servidor DNS está no exterior dessa rede. Quanto tempo leva um computador na rede da sua faculdade para obter o endereço IP associado a www.wikipedia.org quando, por hipótese, o servidor *caching only* apenas conhece os endereços IP dos servidores da zona `root`?
10. Suponha que o tempo médio para executar uma consulta ao servidor *caching only* da sua faculdade é desprezável, mas que esse tempo é de 100 milissegundos quando um servidor DNS está no exterior dessa rede. Considere em todas as questões a seguir que o cliente final que faz a consulta está na rede da sua faculdade e que em todos os casos a consulta diz respeito ao registo com o endereço IP associado ao nome `streaming.fun.com`.
 - (a) Qual o tempo necessário para que o cliente obtenha a resposta admitindo que todos os servidores consultados têm a sua *cache* vazia.
 - (b) Admitindo que os servidores consultados têm registos na sua *cache*, qual o tempo mínimo necessário para que o cliente obtenha a resposta?
 - (c) Tendo em consideração que a probabilidade de o servidor *caching only* ter a resposta a uma consulta disponível na sua *cache* é de 30%, qual o tempo médio necessário para que o cliente obtenha a resposta?
 - (d) Admitindo que o servidor *caching only* da sua faculdade tem na sua *cache* os seguintes registos:


```
fun.com NS dns1.fun.com
fun.com NS dns2.fun.com
dns1.fun.com A 100.100.100.24
dns2.fun.com A 100.100.100.56
```

 e mais nenhuma informação útil para a consulta, qual o tempo necessário para que o cliente obtenha a resposta?
11. Admita que todos os telefones móveis de um país (cujos números começam pelo dígito 9 e têm sempre 9 dígitos) passavam a ter um endereço IP associado.
 - (a) Foi-lhe proposto usar a seguinte solução para registar esses endereços IP no DNS: cada telefone móvel passaria a ser conhecido por um nome DNS da forma: `número.mobile.pt` (*e.g.*, `915678927.mobile.pt`). Critique esta solução.
 - (b) Proponha uma alternativa, continuando a usar o DNS, que permita uma gestão descentralizada do domínio `mobile.pt`.
12. O comando `dig @ns.di.fct.unl.pt di.fct.unl.pt mx` apresentou o resultado abaixo. Responda às questões apresentadas a seguir.

```
$ dig @ns.di.fct.unl.pt di.fct.unl.pt mx
; <>> DiG 8.2 <>> @ns.di.fct.unl.pt di.fct.unl.pt mx
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 4
;; QUERY SECTION:
;; di.fct.unl.pt, type = MX, class = IN

;; ANSWER SECTION:
di.fct.unl.pt.    1D IN MX 10 mail.di.fct.unl.pt.
di.fct.unl.pt.    1D IN MX 100 ftp.di.fct.unl.pt.

;; AUTHORITY SECTION:
di.fct.unl.pt. 1D IN NS ns.di.fct.unl.pt.
di.fct.unl.pt. 1D IN NS ftp.di.fct.unl.pt.
di.fct.unl.pt. 1D IN NS ciup1.ncc.up.pt.

;; ADDITIONAL SECTION:
mail.di.fct.unl.pt. 1D IN A 193.136.122.1
ftp.di.fct.unl.pt. 1D IN A 193.136.122.228
ns.di.fct.unl.pt. 1D IN A 193.136.122.1
ciup1.ncc.up.pt. 23h51m57s IN A 193.136.51.52

;; Total query time: 55 msec
;; FROM: spirou to SERVER: ns.di.fct.unl.pt 193.136.122.1
;; WHEN: Mon Feb 19 09:50:47 2001
;; MSG SIZE sent: 31 rcvd: 194
```

- (a) Indique o nome dos servidores com autoridade sobre o domínio.
- (b) Indique o nome dos servidores de correio electrónico do domínio.
- (c) Deixa de ser possível enviar correio para o utilizador `user@di.fct.unl.pt` se o servidor `ns.di.fct.unl.pt` estiver inacessível?
- (d) Um computador no Japão deixará de poder resolver nomes DNS do domínio `di.fct.unl.pt` caso os canais que ligam a instituição à Internet estiverem avariados e os computadores com endereços IP da forma `193.136.122.x` estejam inacessíveis?
- (e) Os registos apresentados têm associado um TTL com o valor 1D (um dia), mas o registo do nome `ciup1.ncc.up.pt` tem um valor menor (23h51m57s). Explique o que pode estar na origem desta diferença.
- (f) Qual o papel da secção `; ; ADDITIONAL SECTION:?`

13. O comando `dig @bitsy.mit.edu mit.edu any` deu como resultado:

```
$ dig @bitsy.mit.edu mit.edu any
; <>> DiG 9.4.1-P1 <>> @bitsy.mit.edu mit.edu any
; (1 server found)
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62895
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 12, AUTHORITY: 0, ADDITIONAL: 4

;; QUESTION SECTION:
;mit.edu. IN ANY

;; ANSWER SECTION:
mit.edu. 120 IN MX 100 M24-004-BARRACUDA-3.mit.edu.
mit.edu. 120 IN MX 100 W92-130-BARRACUDA-1.mit.edu.
mit.edu. 120 IN MX 100 W92-130-BARRACUDA-2.mit.edu.
mit.edu. 120 IN MX 100 W92-130-BARRACUDA-3.mit.edu.
mit.edu. 120 IN MX 100 M24-004-BARRACUDA-1.mit.edu.
mit.edu. 120 IN MX 100 M24-004-BARRACUDA-2.mit.edu.
mit.edu. 21600 IN SOA BITSY.mit.edu. NETWORK-REQUEST.mit.edu.4878 3600 .....
mit.edu. 21600 IN NS STRAWB.mit.edu.
mit.edu. 21600 IN NS W2ONS.mit.edu.
mit.edu. 21600 IN NS BITSY.mit.edu.
mit.edu. 60 IN A 18.7.22.69

;; ADDITIONAL SECTION:
W92-130-BARRACUDA-1.mit.edu. 21600 IN A 18.7.21.220
W92-130-BARRACUDA-2.mit.edu. 21600 IN A 18.7.21.223
W92-130-BARRACUDA-3.mit.edu. 21600 IN A 18.7.21.224
M24-004-BARRACUDA-1.mit.edu. 21600 IN A 18.7.7.111

;; Query time: 125 msec
;; SERVER: 18.72.0.3#53(18.72.0.3)
;; WHEN: Sat Apr 26 11:51:44 2008
;; MSG SIZE rcvd: 509
```

Responda às seguintes questões:

- (a) Quantos servidores DNS com autoridade tem o domínio?
 - (b) Quantos servidores primários DNS com autoridade tem o domínio?
 - (c) Quantos servidores de correio electrónico tem o domínio?
 - (d) O computador que fez a consulta obteve a resposta directamente de que servidor?
 - (e) Quantas consultas realizou o `dig` admitindo que não tinha *cached* o endereço IP do servidor que lhe enviou a resposta?
14. Escreva um programa na sua linguagem de programação preferida que implemente funcionalidades semelhantes ao programa `dig`.

Capítulo 12

O protocolo HTTP

Anyone who has lost track of time when using a computer knows the propensity to dream, the urge to make dreams come true and the tendency to miss lunch.

– Autor: Sir Tim Berners-Lee, inventor da Web

Até ao final da década de 1980 as aplicações dominantes na Internet eram o correio electrónico, a troca de ficheiros e as sessões remotas. A grande maioria dos utilizadores eram académicos ou investigadores ligados à Informática e os utilizadores com outro perfil eram raros. Tim Berners-Lee era nessa altura investigador no CERN (Centre Européen de Recherche Nucléaire) na Suiça e defrontava-se com o problema de conceber um sistema que tornasse o acesso a informação (documentos, ficheiros, bases de dados, ...) remota mais fácil aos milhares de investigadores da instituição.

Estes investigadores eram especialistas em diferentes áreas, tinham culturas científicas distintas e usavam diferentes tipos de laboratórios, ferramentas, computadores, etc. Com efeito, eles usavam diferentes sistemas de operação, com diferentes formas de designar os documentos, vários formatos de documentos, diferentes códigos de caracteres, diferentes mecanismos de controlo de acesso, diferentes aplicações, referências entre documentos heterogéneas e dependentes de cada tipo de repositório, etc.

Para resolver este problema, Tim Berners-Lee tinha de resolver vários subproblemas, nomeadamente inventar uma forma de:

1. designar um documento de forma normalizada e independente da sua localização;
2. estabelecer referências normalizadas num documento para outro documento de forma independente das respectivas localizações;
3. definir uma linguagem normalizada de descrição de documentos para codificar os diversos (tipos de) documentos;
4. definir um protocolo de acesso a documentos remotos genérico e normalizado; e
5. Implementar um demonstrador do sistema independente do sistema de operação.

Ele conseguiu resolver estes diferentes problemas e, com ajuda da equipa que então dirigiu, apresentou para cada um uma solução específica:

1. inventou os URLs (*Uniform Resource Locators*);
2. usou a noção de Hyper Texto (*Hyper Text*) e os URLs para estabelecer referências “clicáveis” que permitem a navegação entre documentos;

3. adoptou a linguagem de descrição e formatação de documentos HTML (*Hyper Text Markup Language*),
4. inventou o protocolo cliente / servidor HTTP (*Hyper Text Transfer Protocol*), e
5. implementou um cliente e um servidor de HTTP e um interpretador de HTML para vários sistemas de operação.

Um **browser Web** (**navegador Web**) é um cliente do protocolo HTTP, capaz de interpretar a linguagem HTML, os vários tipos de objectos recebidos, e permitir a navegação pelo conjunto de documentos acessíveis, a chamada **Web** (**World Wide Web**). Um **servidor HTTP** é um servidor capaz de enviar e receber entidades digitais (daqui para a frente chamados objectos) através do protocolo HTTP.

Uma linguagem de *markup* é uma linguagem que permite introduzir anotações (*tags*) num texto. Essas anotações não modificam o significado do texto, mas servem para indicar a sua relação com outras partes do texto, ou outros textos, ou ainda para controlar a forma como um texto deve ser formatado e visualizado¹. A linguagem HTML é uma das linguagens de *markup* mais conhecidas. Por exemplo, o seguinte extracto de código HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>HTML Hello World</h1>
<p>Isto parece excitante!</p>
<a href="http://www.w3schools.com/html/">Siga um tutorial sobre HTML</a>
</body>
</html>
```

é um exemplo de um pequeno documento descrito em HTML que um *browser Web* deve interpretar e visualizar de forma normalizada, ver a Figura 12.1.



Figura 12.1: Um pequeno documento descrito em HTML

Nos anos seguintes a versão 1.0 do protocolo HTTP foi desenvolvida e normalizada (mais tarde publicada pelo IETF através do RFC 1945 em 1996) e clientes e servidores foram desenvolvidos para vários sistemas de operação (e.g., Unix, Linux, Windows ...). Em 1992 foi desenvolvida a primeira versão de um *browser Web* para PC e a seguir foi fundada a empresa Netscape, que vulgarizou uma versão comercial do mesmo para Windows (chamado Mosaic). Nos anos seguintes a Web explodiu e tornou-se acessível ao grande público. A Internet, através deste impulso formidável, tornou-se *a rede*, assim como o meio privilegiado de acesso a conteúdos que hoje conhecemos.

¹ O termo linguagem de *markup* tem como origem as convenções de anotação pelos editores dos textos para sua correção ou adequada formatação pelos tipógrafos.

O protocolo HTTP foi definido de tal forma que se tornou um protocolo genérico de transferência entre servidores e *browsers* Web: não só de documentos HTML, mas também de ficheiros em qualquer formato, de todo o tipo de conteúdos digitais, incluindo conteúdos multimédia, e ainda de código de programas para serem interpretados e executados pelos *browsers* Web. A linguagem HTML evoluiu de forma a permitir o desenvolvimento de *sites* Web sofisticados graficamente, com interfaces muito ricas, e com inúmeras funcionalidades que fazem as delícias dos utilizadores.

O objectivo principal deste capítulo é a descrição do protocolo HTTP e tornar claro porque o mesmo se tornou um protocolo genérico de transferência de objectos entre servidores e clientes.

O protocolo **HTTP** (*Hyper Text Transfer Protocol*) é um protocolo cliente / servidor genérico, de transferência entre servidores e clientes: de documentos HTML, de ficheiros em qualquer formato, de todo o tipo de conteúdos digitais, incluindo conteúdos multimédia, e ainda de código de programas para serem interpretados e executados pelos *browsers* Web.

A linguagem **HTML** (*Hyper Text Markup Language*) é uma linguagem que permite introduzir anotações (*tags*) num texto. Essas anotações não modificam o significado do texto, mas servem para indicar a sua relação com outras partes do texto, ou outros textos, ou ainda para controlar a forma como um texto deve ser formatado e visualizado.

12.1 Funcionamento

Uma página Web é normalmente um texto escrito em HTML que descreve uma página a visualizar por um *browser* Web. Geralmente, esse primeiro documento contém igualmente um conjunto de referências para outros objectos, nomeadamente imagens e outros elementos que são automaticamente obtidos pelo *browser* Web de forma a mostrar ao utilizador um conjunto de elementos gráficos que, no seu conjunto, formam a verdadeira página visualizada pelo utilizador.

O protocolo HTTP é um protocolo cliente / servidor, executado sobre uma conexão TCP, e que em cada interacção entre o cliente e o servidor transfere um objecto de cada vez. No pedido o cliente pode indicar a designação (*e.g.*, nome) do objecto a transferir e indicar igualmente outros parâmetros significativos para a transferência. O servidor transmite o objecto para o cliente acompanhado de meta informação sobre o mesmo (*e.g.*, nome, tipo, dimensão, data de criação, *etc.*).

Os clientes mais populares do protocolo HTTP são os *browsers* Web (*e.g.*, Firefox, Internet Explorer, Safari, Chrome, Opera, *etc.*) e muitas outras aplicações como por exemplo as aplicações para *smartphones*, ou programas como o `wget`. É relativamente fácil programar um cliente HTTP simples que obtém objectos de um servidor HTTP. De facto, o que é bastante mais complexo é interpretar e visualizar os objectos recebidos.

Existem também inúmeros servidores HTTP. Um dos mais utilizados, cujo código é do domínio público, é o servidor Apache, distribuído pela Apache Foundation (<http://www.apache.org>), mas existem muitos outros proprietários de vários fabricantes de software.

As referências para uma página HTML, e genericamente para qualquer objecto com informação digital que o *browser* Web pretenda obter via o protocolo HTTP, chama-se um **URL** (*Uniform Resource Locator*)². Um URL tem uma sintaxe bem

² É também frequente usar-se a nomenclatura URI (*Uniform Resource Identifier*) ao invés

definida e é composto pelos seguintes elementos:

Sintaxe geral de um URL HTTP:

`http[s]://nome-do-servidor[:porta]/nome-local-do-objecto`

À parte inicial corresponde ao **protocolo** do URL e pode ter vários valores, nomeadamente **http** ou **https**. A variante HTTPS é usada para garantir que o acesso aos objectos é por HTTP mas suportado em canais seguros. A noção de URL suporta também outros protocolos, como por exemplo **file**, que indica ao *browser* Web que o objecto está no sistema de ficheiros local. No entanto, essas variantes não usam o protocolo HTTP para obter o objecto pelo que não as iremos referir mais.

O campo **//nome-do-servidor**, como o nome indica, corresponde ao nome DNS do servidor ao qual se pretende solicitar o objecto. O mesmo objecto designa-se por **/nome-local-do-objecto** relativamente a esse servidor. A porta é um parâmetro que toma valores por omissão caso não seja indicada (a porta por omissão do protocolo HTTP é a porta 80 e a porta por omissão do protocolo HTTPS é a porta 433). Quando a componente **/nome-local-do-objecto** do URL não está presente, esta também toma um valor por omissão (*e.g.*, `/index.html`).

Vamos agora ver de mais perto como se desenrola a interacção entre um cliente e um servidor HTTP. Por exemplo, se um *browser* Web receber indicação para aceder à página Web `http://en.wikipedia.org/wiki/Main_Page`, executa as seguintes operações:

1. Analisar o URL para decompô-lo nas suas componentes;
2. obter o endereço IP de `en.wikipedia.org` através do DNS;
3. abrir uma conexão TCP para esse endereço IP, na porta por omissão, a porta 80 dado o protocolo ser HTTP;
4. enviar uma mensagem HTTP de pedido do objecto `/wiki/Main_Page`;
5. receber a mensagem de resposta e analisar o seu cabeçalho;
6. se o resultado tiver êxito, mostrar ao utilizador o documento HTML recebido; e
7. caso o documento contenha referências para outros objectos que devem ser igualmente obtidos, obtê-los e processá-los de forma semelhante.

O protocolo HTTP é um protocolo cliente servidor, ver a Figura 12.2, com duas mensagens: HTTP Request e HTTP Reply. Estas mensagens têm de ser transmitidas sobre conexões TCP. Cada mensagem HTTP Request só pode solicitar um objecto e cada mensagem HTTP Reply só pode conter um objecto. Assim, para se obterem n objectos, é necessário realizar n trocas de mensagens semelhantes à ilustrada na figura.

Na sua forma mais simples o protocolo HTTP é executado de tal forma que a cada troca de mensagens pedido / resposta está associada uma conexão TCP diferente. Isto é, o padrão abertura da conexão, envio do pedido, obtenção da resposta e fecho da conexão, é repetido para cada objecto, como ilustrado na Figura 12.3. Esta põe em evidência que não é possível obter desta forma um objecto em menos de $2 \times RTT$ segundos, assumindo que o pedido e o objecto cabem cada um dentro de um único segmento TCP. Veremos mais adiante que é possível melhorar este padrão reutilizando a conexão.

de URL. No entanto, no contexto do protocolo HTTP não faz sentido, na maioria das situações, falar em URI, pois o protocolo HTTP necessita de facto do nome DNS de um servidor para abrir a conexão para o mesmo. Ora um nome DNS é traduzido num endereço IP e não tem geralmente as propriedades de um identificador, ver o Capítulo 11.

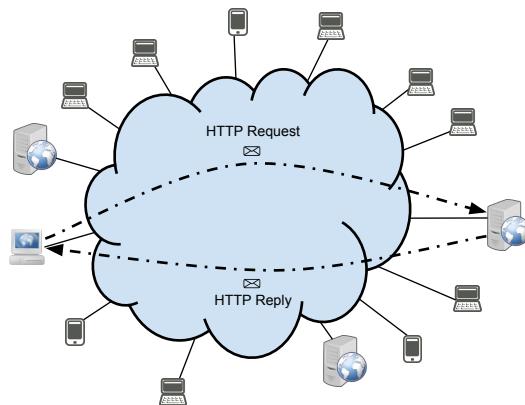


Figura 12.2: Obtenção pelo cliente de um objecto do servidor através do protocolo HTTP

As referências para objectos acessíveis pelo protocolo HTTP são materializadas nos **URLs (Uniform Resource Locators)**. Estes contém, entre outras informações, o nome do servidor que dá acesso ao objecto, a porta desse servidor, assim como o nome relativo do objecto nesse servidor.

O protocolo HTTP (*Hyper Text Transfer Protocol*) é um protocolo cliente / servidor que funciona sobre conexões TCP. O protocolo só tem duas mensagens: HTTP Request (pedido) e HTTP Reply (resposta). Cada mensagem de pedido só pode pedir um objecto ao servidor e cada mensagem de resposta só pode conter um objecto.

Na forma mais simples do protocolo, a cada pedido e resposta corresponde uma conexão TCP específica, pelo que a obtenção de um objecto neste caso nunca pode levar menos do que $2 \times RTT$ segundos.

Formato genérico das mensagens HTTP

Como em todos os outros protocolos, cada mensagem HTTP é também formada por um cabeçalho e um *payload*. No entanto, as mensagens HTTP têm uma diferença fundamental com respeito às dos outros protocolos que vimos até aqui. Nos protocolos como o IP, UDP, TCP, RTP, DNS, ... os cabeçalhos estão organizados numa sequência de campos representados em binário (*i.e.*, cada campo tem um certo número de bits que codificam inteiros por exemplo). No HTTP, os cabeçalhos são constituídos por uma sequência de linhas de texto codificadas no código US-ASCII (American Standard for Information Interchange) e por isso são facilmente legíveis. A Figura 12.4 ilustra uma troca de mensagens que utiliza o protocolo HTTP.

Uma mensagem de pedido, que se chama HTTP Request, tem um cabeçalho constituído por um conjunto de linhas texto. Cada uma dessas linhas termina exactamente com a sequência CRLF que corresponde ao carácter de controlo US-ASCII CR, carriage return (13) seguido do carácter de controlo US-ASCII LF, linefeed (10). Desta forma quebra-se a ambiguidade sobre qual a forma como uma linha deve terminar pois esta convenção é diferente no sistema Windows (que só usa CR) e nos sistemas a

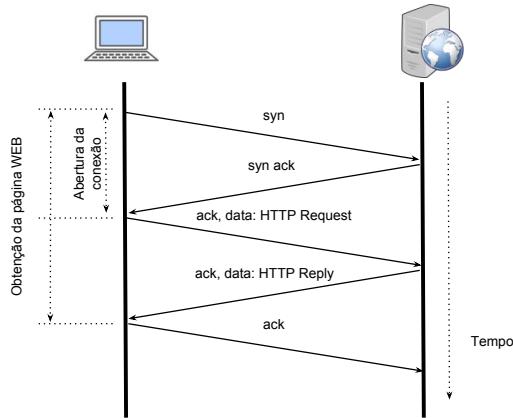


Figura 12.3: O padrão [abertura da conexão, envio do pedido, obtenção da resposta e fecho da conexão] para obtenção de cada objecto via HTTP

la Unix (que usam CRLF).

A primeira linha do pedido é especial e designa-se por *request line*. As restantes designam-se por *request header lines*, podem ir por uma ordem arbitrária e, com exceção da segunda linha do exemplo (*Host:*), são geralmente opcionais em muitas interacções. Muitos servidores tentam suprir os parâmetros ausentes nos pedidos com valores por omissão. No limite, um pedido HTTP poderia ter um cabeçalho que se resumisse à primeira linha. O cabeçalho termina por uma linha em branco e, finalmente, segue-se uma parte opcional, designada *entity body*, cujo papel será explicado mais tarde.

Formato genérico de uma mensagem de pedido:

```
método /nome-local-do-objecto versão-do-protocolo CRLF
header field name: field value CRLF
header field name: field value CRLF
.....
header field name: field value CRLF
CRLF
[ entity body ]
```

Exemplo:

```
GET /wiki/Main_Page HTTP/1.0 CRLF
Host: en.wikipedia.org CRLF
User-Agent: Mozilla/44.01 CRLF
CRLF
```

A *request line* contém o equivalente a um código operação (método), seguido do nome de um objecto e da versão do protocolo que o cliente pretende usar. As restantes linhas do cabeçalho, as *request header lines*, permitem ao cliente passar diversas informações ao servidor. Por exemplo, o *header field User-Agent* permite ao *browser* Web dizer qual o seu modelo, o que pode ser interessante para o servidor do ponto

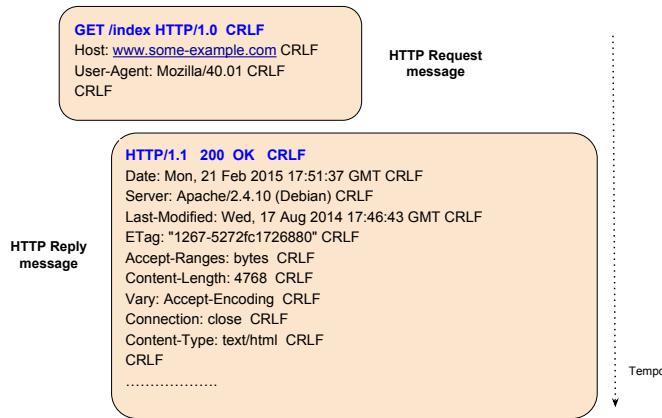


Figura 12.4: Exemplo de mensagens de pedido e resposta do protocolo HTTP. Na figura a abreviatura CRLF denota o par ordenado de caracteres: **carriage return, linefeed**.

de vista da elaboração de estatísticas. O *header field* `Host` permite ao cliente indicar a que servidor exacto pretende fazer o pedido. Este mecanismo foi introduzido para permitir que um servidor Web físico, com um único endereço IP, suporte vários servidores virtuais, cada um com um nome distinto. Em alternativa ao *header field* `Host` a *request line* pode conter o URL completo ao invés de `/nome-local-do-objecto`.

A mensagem de resposta, que se chama HTTP Response, é semelhante na sintaxe. A primeira linha da resposta é especial e designa-se por *status line*. As restantes designam-se por *response header lines*, podem estar por uma ordem qualquer e servem, no essencial, para o servidor transmitir informação e meta-dados sobre o objecto transmitido (*e.g.*, data de criação, dimensão, *etc.*). O cabeçalho da resposta termina por uma linha em branco e, finalmente, segue-se uma parte opcional, designada *entity body*, que geralmente contém o objecto solicitado pelo cliente.

Formato genérico de uma mensagem de resposta:

```

versão código-do resultado frase CRLF
header field name: field value CRLF
header field name: field value CRLF
.....
header field name: field value CRLF
CRLF
[ entity body ]

```

Exemplo:

```

HTTP/1.1 200 OK CRLF
Date: Mon, 21 Feb 2015 17:51:37 GMT CRLF
Server: Apache/2.4.10 (Debian) CRLF
Last-Modified: Wed, 17 Aug 2014 17:46:43 GMT CRLF
ETag: "1267-5272fc172688" CRLF
Accept-Ranges: bytes CRLF

```

```
Content-Length: 4768 CRLF
Vary: Accept-Encoding CRLF
Connection: close CRLF
Content-Type: text/html CRLF
CRLF
dados, dados, ...
```

A *status line* contém a versão do protocolo escolhida pelo servidor, seguida de um inteiro que é o código do resultado, e termina com o resultado escrito numa forma legível pelos humanos (a frase). As *header lines* contém várias informações. Por exemplo, a data no servidor no momento do envio da resposta (*header field Date*), o modelo do servidor (*Server*), meta dados que caracterizam o objecto da resposta (*Last-Modified*, *Content-Length*, *Content-Type*, ...), etc. O cabeçalho termina com uma linha em branco e depois, geralmente, seguem-se os dados com o objecto pedido pelo cliente.

Esta forma de conceber os cabeçalhos das mensagens HTTP foi inspirada pelo protocolo de correio electrónico SMTP (*Simple Mails Transfer Protocol*) e caracteriza-se, do lado das vantagens, por ser muito flexível e extensível, e auxiliar o desenvolvimento e o *debug*. Como defeito, poder-se-ia sublinhar a “grande” dimensão do cabeçalho. No entanto, este defeito tem hoje em dia um peso menor face à capacidade dos canais, à dimensão dos objectos transportados e, por outro lado, torna-se irrelevante quando se usa compressão dos cabeçalhos HTTP, uma opção cada vez mais comum.

Uma das vantagens destas opções é posta em evidência pelo facto de que é possível dialogar *manualmente* com um servidor HTTP na linha de comando através do utilitário *Telnet* como se ilustra a seguir.

```
$ telnet en.wikipedia.org 80
Trying 91.198.174.192...
Connected to en.wikipedia.org.
Escape character is '^].
GET / HTTP/1.0

HTTP/1.1 200 OK
Server: Apache
X-Powered-By: HHVM/3.3.0-static
Last-Modified: Tue, 02 Feb 2016 14:25:41 GMT
ETag: "3d2-52aca46b79fd9"
Content-Type: text/html
.....
```

Nos parágrafos seguintes são discutidos os diferentes campos dos cabeçalhos.

Métodos ou comandos do HTTP

A grande maioria das interacções entre os clientes e os servidores são para obter objectos do servidor. Estes pedidos utilizam o método GET. No entanto, existem outros comandos possíveis como é ilustrado na Tabela 12.1.

Os comandos PUT e DELETE modificam o estado do servidor pelo que, geralmente, são sujeitos a controlo de acesso. O mesmo se aplica aos outros comandos se o servidor o exigir. O controlo de acesso será discutido na Secção 12.3. Os comandos POST e PUT também transmitem um objecto para o servidor.

Header fields HTTP

O protocolo HTTP admite várias dezenas de *header fields* distintos, muitos normalizados, mas também suporta um mecanismo de extensão que permite a uma dada categoria de clientes e servidores introduzirem alguns de uso privado, específicos de um fabricante ou de uma aplicação. A Tabela 12.2 lista alguns *header fields* populares.

Tabela 12.1: Lista dos principais comandos HTTP

Nome	Descrição
GET	Pedido do objecto identificado no URL
HEAD	Pedido do objecto identificado no URL, mas o cliente apenas pretende obter o cabeçalho da resposta
TRACE	Ecoa o pedido do cliente para apoio ao <i>debug</i>
OPTIONS	Ecoa os comandos e opções que o servidor aceita para um certo URL (muitos servidores ignoram este comando por segurança)
POST	Permite ao cliente transmitir um conjunto de dados para o servidor, geralmente contendo parâmetros de uma operação (<i>e.g.</i> , envio de um formulário HTML, comentário numa discussão, valor a colocar numa base de dados, ...)
PUT	Introdução ou substituição do objecto identificado no URL
DELETE	Supressão do objecto identificado no URL

Seguem-se uns breve comentários sobre alguns dos exemplos presentes na tabela. O primeiro grupo são *header fields* que são enviados pelo cliente ao servidor para indicar que tipo de cliente é (*User-Agent*) assim como as suas opções (*e.g.*, *Accept-Charset*, *Accept-Encoding* e *Accept-Language*). Os *header fields*:

If-Modified-Since e If-None-Match

são formas de condicionar a resposta do servidor. No primeiro caso, o cliente só está interessado no objecto se este tiver sido modificado depois da data indicada, pois provavelmente tem uma versão em *cache*. No segundo caso, o cliente envia um código de segurança do tipo *digest* do objecto e pretende que o servidor só aceite o objecto enviado por um PUT se a versão por este recebida mantém o mesmo *digest*³. O *header field Range* ilustra um pedido parcial: o cliente só está interessado em receber um certo intervalo do conteúdo do objecto.

O segundo grupo são *header fields* que ilustram alguns dos dados enviadas pelos servidores. O primeiro mostra um exemplo que permite ao servidor indicar o seu modelo. Os mais comuns permitem ao servidor indicar atributos (meta dados) do objecto enviado (*e.g.*, *Last-Modified*, *Content-Length*, *Content-Encoding*, *Content-Type* e *ETag*). O *header field Accept-Ranges* é um exemplo em que o servidor indica ao cliente que suporta a funcionalidade pedidos parciais.

O *header field Content-Type* é um dos mais importantes pois permite ao *browser* Web saber como deve interpretar o objecto recebido. A tabela 12.3 ilustra um pequeno subconjunto. Um *browser* Web é uma peça de software muito complexa que para além de executar o protocolo HTTP tem de interpretar os conteúdos transmitidos pelo servidor HTTP (e vice-versa) e, em função do seu tipo, processá-los imediatamente e mostrá-los aos utilizadores (através de funcionalidades nativas ou de *plug-ins*), lançar dinamicamente uma aplicação externa que os execute (*e.g.*, Microsoft Excel), ou fazer o simples *download* dos mesmos.

³ Uma função de *digest* é uma função que a um conteúdo arbitrário faz corresponder uma sequência de bits única, ou *digest*, que se pode chamar um “resumo seguro” desse conteúdo. Por exemplo, a função SHA-1 calcula um resumo (*digest*) com 160 bits de comprimento. Os *digests*, se tiverem um número adequado de bits, permitem verificar a integridade, *i.e.*, a não modificação de uma informação por um intermediário, pois se a informação for alterada, o seu *digest* modifica-se necessariamente. Por outro lado, é computacionalmente impossível gerar um conteúdo com um *digest* pré-definido.

Tabela 12.2: Lista de vários exemplos de *header fields* HTTP

<i>Header fields</i>	Exemplo
User-Agent	User-Agent: Mozilla/40.0
Accept-Charset	Accept-Charset: utf-8
Accept-Encoding	Accept-Encoding: gzip
Accept-Language	Accept-Language: en-UK
If-Modified-Since	If-Modified-Since: Tue, 02 Feb 2016 14:25:41 GMT
If-Match	If-Match: "756154ad8d 3102f1349317f"
Range	Range: bytes=500-999
...
Server	Server: Apache
Last-Modified	Last-Modified: Tue, 02 Feb 2016 14:25:41 GMT
Content-Length	Content-Length: 348
Content-Encoding	Content-Encoding: gzip
Content-Type	Content-Type: text/html; charset=utf-8
ETag	ETag: "3d2-52aca46b79fd9"
Accept-Ranges	Accept-Ranges: bytes
...

Códigos de resposta HTTP

Os códigos de resposta do servidor têm como papel essencial transmitir um diagnóstico do servidor para o cliente. Estes aparecem na *status line* através de um número, o código de diagnóstico, e uma texto explicativo que é apenas uma ajuda para o cliente mostrar o diagnóstico ao utilizador. A Tabela 12.4 lista alguns códigos de diagnóstico comuns.

O leitor interessado em conhecer uma lista exaustiva dos métodos, códigos de diagnóstico e *header fields* HTTP poderá consultar os RFCs 7230 a 7235 que descrevem o protocolo, a sintaxe das mensagens e a utilização das linhas dos cabeçalhos para diversos fins.

O protocolo HTTP suporta diversos pedidos do cliente ao servidor, que se traduzem em outros tantos métodos (semelhantes a códigos de operação) na mensagem de pedido. Os de utilização mais frequentes são: o método GET para solicitar um objecto, o método POST para transmitir um formulário preenchido para o servidor, e o método PUT para transmitir um objecto arbitrário para o servidor.

As mensagens HTTP contém cabeçalhos estruturados como um conjunto de linhas de texto terminadas nos caracteres CRLF e o cabeçalho termina numa linha em branco. Com exceção da primeira linha dos cabeçalhos do pedido e do resultado, as outras linhas designam-se por *request / response header lines*, podem variar em número, e permitem ao cliente e ao servidor trocarem diversas informações (*e.g.*, os meta dados sobre os objectos transmitidos) ou condicionar a execução dos diferentes métodos. Esta forma de estruturar os cabeçalhos é muito extensível, flexível e também facilita o *debug*.

Onde acaba um objecto

O cabeçalho das mensagens HTTP é extensível mas, quer o cliente, quer o servidor, reconhecem o fim do cabeçalho quando encontram uma linha em branco. No entanto,

Tabela 12.3: Alguns exemplos de *Content-Types* HTTP

Tipo	Descrição
text/plain	Texto não formatado
text/html	Texto com código HTML
image/jpeg	Imagen codificada em JPEG
video/mpeg	Vídeo codificado em MPEG
application/octetstream	Objecto opaco (<i>e.g.</i> , código executável externo)
application/postscript	Documento codificado em PostScript
application/java-vm	Java Bytecode File
application/javascript	JavaScript File
application/vnd.ms-excel	Microsoft Excel File
application/json	JavaScript Object Notation (JSON)

Tabela 12.4: Exemplos de *Status codes* HTTP

Código 1xx	Exemplo Códigos informativos	Descrição
100	Continue	Um objecto vai ser devolvido mas ainda está a ser gerado
2xx	Códigos de sucesso	
200	OK	Sucesso, segue-se o objecto pedido
204	OK but no content	O objecto existe mas está vazio
3xx	Códigos de redirecção	
301	Moved permanently	O objecto tem um novo URL
4xx	Códigos de erro (cliente)	
400	Bad request	Pedido não reconhecido (sintaxe, ...)
404	Not found	O objecto pedido não existe
5xx	Códigos de erro (servidor)	
501	Not implemented	O servidor não suporta o método

reconhecer o fim de um objecto transmitido é mais complicado pois o conteúdo de um objecto é arbitrário e não pode ser facilmente reconhecido com uma marca.

Quando um cliente recebe uma resposta, caso esteja a usar a versão 1.0 do protocolo HTTP, que exige que a cada interacção cliente / servidor distinta esteja associada uma conexão TCP distinta, o objecto da resposta termina quando não houver mais nada a ler nessa conexão. Esta é a forma mais simples de um cliente reconhecer o fim de um objecto. No entanto, o protocolo HTTP admite, em versões posteriores, que a mesma conexão TCP seja partilhada por vários pedidos e respostas. Neste caso, o fim do objecto enviado pelo servidor tem de ser reconhecido recorrendo ao *header field Content-Length* que indica a sua dimensão. Adicionalmente, certos objectos, como por exemplo os formulários HTML, enviadas pelo cliente ao servidor, têm uma marca (*tag*) que permite reconhecer o seu fim.

Comandos HTTP idempotentes

Todos os protocolos cliente / servidor têm um problema delicado que consiste em saber se quando um cliente solicita uma operação ao servidor, a mesma foi executada zero, uma, ou mais vezes. À primeira vista a questão pode parecer deslocada pois a

resposta óvia: seria exactamente uma ou nenhuma vez. Ou seja, o comportamento deveria ser o que se designa como comportamento atómico ou transacional, *i.e.*, ou bem que a operação teve sucesso e foi executada ou, no caso contrário, o seu resultado é equivalente a nunca ter acontecido. Um servidor transaccional desfaz todos os efeitos de uma operação abortada, como se esta nunca tivesse existido.

Implementar o comportamento atómico não é assim tão simples, pois pode haver um problema na rede, e o cliente que não recebeu a resposta pode desencadear de novo o pedido. Tal pode acontecer simplesmente porque a operação levou muito tempo a executar ou o utilizador impacientou-se e carregou de novo num botão, o que desencadeou um novo pedido. Pode também acontecer que o cliente tenha mudado de ideias e resolva abortar a operação, mas esta já tinha sido executada pelo servidor apesar de o cliente ainda não o saber. Um servidor transaccional tem de distinguir operações com sucesso de operações repetidas e de operações abortadas, e garantir o comportamento transacional.

Perante as situações indicadas, o servidor pode receber pedidos repetidos sem que o cliente (ou o utilizador) se apercebam do sucedido. Em certos casos, isso não tem importância nenhuma, mas noutros pode ter um efeito totalmente insatisfatório. Por exemplo, ao invés de encomendar uma televisão, o utilizador encomendou duas, ou fez duas vezes o mesmo pagamento.

Este tipo de efeitos nocivos podem ser evitados se o servidor tomar as medidas para que nunca execute operações repetidas. Uma solução simples poderia passar por o cliente dar um número único a cada operação, e o servidor detectar os duplicados. No entanto, isso implicaria que o servidor passe a conhecer os clientes que o contactam, e guarde também alguma história sobre as operações que estes realizaram no passado. Em qualquer hipótese isso é uma complicação para o servidor que passa obrigatoriamente, mesmo que não o deseje, a ter de conhecer de forma unívoca os seus clientes e a sua história passada. Tudo isso complica o trabalho do servidor.

No entanto, há uma categoria especial de pedidos realizados pelo servidor que não têm problemas se forem executados em duplicado. O exemplo mais conhecido consiste na obtenção de uma cópia de um objecto imutável (*i.e.*, que não muda). Este tipo de operações dizem-se **operações idempotentes** (*idempotent operations*). Uma operação idempotente é uma operação cuja execução repetida, mesmo que devolva um valor diferente em cada caso, não conduz a erros, nem no servidor nem no cliente. Uma operação de substituição de um objecto por outro também pode ser considerada idempotente em certos contextos aplicacionais. Por exemplo, se o objecto só pode ser modificado por um único cliente, uma execução repetida desta operação conduz sempre ao mesmo resultado final.

Os servidores DNS só suportam operações idempotentes pelo que são mais simples e não precisam de manter informação sobre os seus clientes. Isto é, uma vez uma operação executada, eles podem-na esquecer completamente, assim como ao cliente que fez o pedido. Os servidores que só implementam operações idempotentes são mais simples que os outros, devido à possibilidade de não manterem informação sobre o passado.

O protocolo HTTP foi definido de forma a que quase todas as operações fossem idempotentes. Por isso várias operações seguidas, executadas pelo mesmo cliente, podem ser executadas usando uma única ou várias conexões TCP, e podem ser repetidas se o utilizador se impacientar.

Os métodos GET, PUT, HEAD, TRACE, OPTIONS e DELETE são idempotentes na maioria dos contextos aplicacionais. O método POST não é de certeza. É necessário ter em atenção que os métodos idempotentes podem não deixar o servidor no mesmo estado se forem executados uma ou mais vezes. Por exemplo, se associado ao método GET estiver um contador de vezes que os clientes o solicitam, cada execução repetida incrementa o contador mais do que uma vez. No entanto, isso pode não constituir um erro propriamente dito em diversos contextos onde alguma imprecisão do contador

pode ser tolerada.

Uma **operação idempotente** (*idempotent operation*) é uma operação cuja execução repetida, mesmo que devolva um valor diferente em cada caso, não conduz a erros aplicacionais, nem no servidor, nem no cliente.

Se um servidor tem uma interface exclusivamente baseada em operações idempotentes, diz-se um **servidor sem estado** (*stateless server*) e torna-se mais simples porque não tem de detectar operações executadas mais do que uma vez. Parte das operações executadas pelos servidores HTTP também são idempotentes, nomeadamente os métodos GET e PUT.

No seu essencial, o protocolo HTTP privilegia, através da sua definição, a idempotência das operações. No entanto, com o passar dos tempos, o protocolo passou a servir de base a inúmeras aplicações mais complexas do que a simples obtenção de objectos imutáveis de um servidor e, por essa razão, foi necessário introduzir mecanismos que permitissem ao servidor reconhecer os seus clientes (e no limite o utilizador por detrás do *browser* Web) e implementar mecanismos opcionais de conhecer a história dos mesmos. Voltaremos a este assunto na Secção 12.3.

12.2 Desempenho

O protocolo HTTP é hoje em dia responsável pelo transporte da quase totalidade da informação que circula na Web, é muito usado para suporte do diálogo com servidores pelas aplicações móveis (que são executadas pelos *smartphones*), e começou também a ser muito usado para o transporte de informação multimédia para a difusão de vídeo [Müller and Timmerer, 2011]. Por estas razões este protocolo tornou-se responsável por grande parte dos dados que circulam entre operadores na Internet. Dada esta prevalência e centralidade, é natural que os investimentos na melhoria da sua eficiência tenham um grande retorno, quer para as aplicações em particular, quer para a optimização do tráfego que atravessa a Internet.

Esses investimentos abrangem diversas facetas: a maneira como as aplicações usam o HTTP, a maneira como o HTTP se relaciona com o TCP e pode ele próprio ser melhorado, e a forma como se podem introduzir novos subsistemas na Internet para apoiarem o funcionamento da distribuição de conteúdos baseada em HTTP. Começaremos a discussão pela primeira destas facetas.

Suporte de *caching* nos clientes

Muitos dos objectos disponíveis na Web são imutáveis *i.e.*, não são alterados (*e.g.*, fotografias, documentos, livros, ...) ou pelo menos evoluem muito devagar, *i.e.*, a sua actualização para novas versões é feita com intervalos muito espaçados. Por esta razão, e dado que muitos desses objectos são volumosos (*e.g.*, têm dezenas, centenas ou mesmo milhares de Kbytes), torna-se interessante fazer *caching* dos mesmos. Com efeito, todos os *browsers* Web (e não só) fazem *caching* de alguns dos objectos que recebem.

Um dos grandes problemas de todos os sistemas de *caching* é decidir sobre se um objecto *cached* ainda corresponde à versão mais recente do mesmo, ou já está desactualizado (*staled*). Como é evidente, tornar os servidores responsáveis por avisar os clientes de que um objecto *O* foi actualizado, exigiria que os servidores mantivessem informação sobre todos os clientes que obtiveram uma cópia de *O*, e os mesmos fossem avisados se *O* for alterado. Exigiria também enviar potencialmente centenas, milhares,

ou mesmo milhões de notificações. As várias redes sociais existentes suportam o envio de notificações a milhares ou mesmo milhões de clientes (*e.g.*, Twitter). No entanto, isso exige uma infra-estrutura de grande complexidade que não pode ser implementada com um único servidor.

Assim, o protocolo HTTP apenas suporta por omissão *caching* do lado dos clientes sem notificações sobre actualizações pelos servidores. Compete aos clientes indagarem sobre se houve ou não actualização dos objectos *cached*. O DNS tem um problema semelhante que é resolvido usando o TTL, ver o Capítulo 11. Os mecanismos introduzidos no protocolo HTTP são diferentes.

Por um lado, cada objecto devolvido por um servidor é acompanhado pelo meta dado transmitido pelo *header field Last-Modified*, que permite ao cliente calcular posteriormente a idade de cada versão *cached* de um objecto. Existe também um *header field*, denominado *Expires*, que permite indicar a data prevista para a actualização de um objecto transmitido, e outro *header field* denominado *Cache-control* (ver RFC 7231), que permite ao servidor transmitir aos clientes indicações flexíveis sobre a política de *caching* a aplicar ao objecto.

O HTTP suporta igualmente um mecanismo designado pedido condicional (*conditional get*) que permite a um cliente fazer um pedido condicionado usando a data anotada anteriormente, *i.e.*, transmitida pelo *header field Last-Modified* quando o objecto foi recebido pela última vez. Assim, o cliente pode fazer um pedido condicional introduzindo na mensagem de pedido um *header field If-Modified-Since* em que a data indicada é a data obtida pelo *header field Last-Modified*. Se o mesmo objecto no servidor não tiver sido actualizado em relação à versão da data indicada, a mensagem de resposta terá o código 304 *Not Modified* e o objecto não é transmitido. Caso o objecto tenha sido actualizado, a resposta é 200 *OK* e a nova versão do objecto é transmitida. A Figura 12.5 ilustra o funcionamento deste mecanismo.

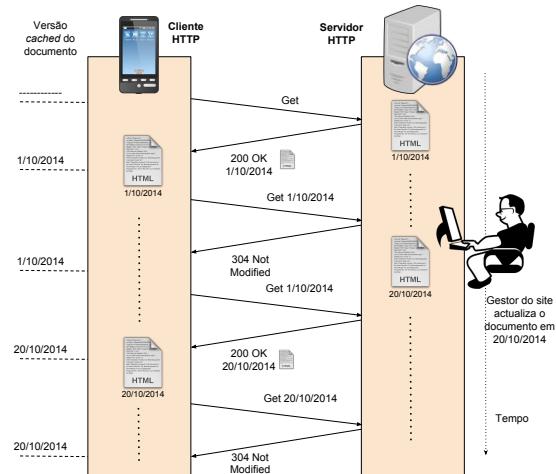


Figura 12.5: Gestão de uma *cache* num cliente HTTP com recurso ao *header field If-Modified-Since*

O mecanismo designado *HTTP Etag* também permite fazer um GET condicional se o cliente anotar a *Etag* transmitida pelo servidor através do *header field Etag* (*e.g.*, *ETag: "3d27f9"*) e usar o *header field If-None-Match*, (*e.g.*, *If-None-Match: 3d27f9*) transmitido pelo cliente ao servidor no cabeçalho da mensagem GET. A mensagem de resposta a este pedido terá o código 304 *Not Modified* se a *Etag* continuar

válida, ou a nova versão do objecto no caso contrário⁴.

Muitos dos objectos, nomeadamente os calculados dinamicamente, não devem ser *cached* pelos clientes pois são específicos e efémeros. Idealmente o servidor deve assinalar isso aos clientes, o que pode fazer usando o *header field Expires* com o valor `-1`. O *header field Cache-control* permite ainda maior flexibilidade e será discutido no Capítulo 13.

Reutilização das conexões TCP

A versão 1.0 do protocolo HTTP exige que o cliente abra uma nova conexão TCP por cada pedido que faz ao servidor. O objectivo era proporcionar o máximo de simplicidade: o servidor esquece o cliente no fim de cada pedido, o que associado ao carácter grosso modo idempotente da interface, torna o servidor tão simples quanto possível. Adicionalmente, o cliente sabe que quando a conexão for fechada pelo servidor já recebeu a totalidade da mensagem de resposta.

Esta solução adaptava-se bem a situações em que todos os objectos solicitados pelos clientes eram volumosos e o RTT relativamente baixo. Neste contexto, o tempo de abertura da conexão e a lentidão da fase *slow start* do TCP eram amortizados e tornavam-se pouco significativos.

No entanto, posteriormente, a maioria das páginas começou a ser formada por inúmeras componentes (separadores, menus, fotografias, publicidade, scripts, vários botões, ...) e hoje em dia é frequente encontrarem-se páginas que para serem visualizadas exigem que o cliente obtenha várias dezenas de objectos, muitos de pequena dimensão. Por outro lado, muitos servidores são acedidos a partir de outro continente e o RTT pode tomar valores superiores a 100 milissegundos. Neste quadro, o tempo da abertura de muitas conexões TCP, assim como o tempo necessário para vencer a fase *slow start* em cada uma, aumenta inutilmente o tempo necessário para obter de forma completa uma página.

A versão 1.1 do protocolo HTTP foi introduzida pelo RFC 2068 em 1997 e revista em 1999 pelo RFC 2616. Esta nova versão introduziu uma revisão extensa do protocolo incluindo a possibilidade de uma única conexão TCP ser partilhada por vários pedidos de um cliente ao mesmo servidor. Por um lado, para obter n objectos a partir do mesmo servidor, poupar-se $n - 1$ aberturas de conexão, e por outro, se o ritmo de transmissão dos objectos for adequado, a janela TCP pode alargar até um valor mais adequado pois o débito extremo a extremo de uma conexão TCP é proporcional à dimensão da janela sobre o RTT, ver o Capítulo 7.

Assim, para que um cliente HTTP obtenha de um servidor todos os n objectos necessários para mostrar um página Web ao utilizador, a versão HTTP 1.1 permite ao cliente poupar pelo menos $RTT \times (n - 1)$ segundos na operação, sem considerar o ganho potencial devido à utilização de uma janela TCP em média maior no servidor que transmite os objectos. A Figura 12.6 ilustra a diferença quando $n = 3$, isto é, quando após obter a página HTML inicial, o cliente constata que a mesma faz referência a dois objectos suplementares (*e.g.*, fotografias) que é necessário obter também para mostrar a página de forma completa ao utilizador.

Com o protocolo HTTP 1.0 o cliente executa três vezes o padrão [abertura de conexão, envio do pedido, obtenção da resposta, fecho da conexão]. A figura ilustra uma situação teórica em que o objecto pode ser transmitido num único segmento mas, geralmente, a transmissão do objecto requer vários RTTs. É por isso que se os objectos forem todos de grande dimensão, o tempo requerido para a sua recepção amortiza o tempo de abertura da conexão.

Com o protocolo HTTP 1.1, o cliente executa o padrão [abertura de conexão, envio do pedido, obtenção da resposta] para obter a página HTML inicial, mas a seguir apenas repete duas vezes o padrão [envio do pedido, obtenção da resposta] pois reutiliza a conexão.

⁴Para a geração de *Etags* o servidor poderá usar funções de cálculo de *digests*.

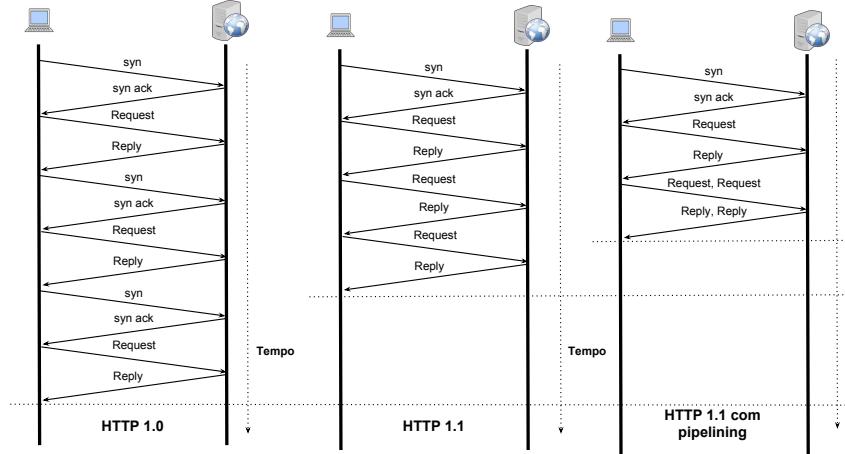


Figura 12.6: Tempo para obter uma página Web e dois objectos nela referenciados com diferentes versões do protocolo HTTP. O diagrama ignora o ganho suplementar devido a só existir, potencialmente, uma única fase *slow start* da conexão e considera, por hipótese, que cada pedido e resposta são executados num único RTT.

A versão 1.1 do protocolo HTTP permite também a utilização opcional de um mecanismo designado *pipelining*, que consiste em permitir ao cliente enviar vários pedidos imediatamente uns a seguir aos outros, e receber posteriormente as respostas tambémumas a seguir às outras. Este mecanismo, cuja utilização é também ilustrada no diagrama mais à direita da Figura 12.6, permite, nas hipóteses teóricas colocadas, diminuir ainda mais o tempo necessário para obter a totalidade dos objectos. Os diagramas não ilustram o fecho das conexões pois estas têm lugar em paralelo com a continuação da execução, *i.e.*, o mostrar ao utilizador os conteúdos obtidos.

A utilização de *pipelining* com métodos não idempotentes é de todo desaconselhada pois no caso de a conexão ser interrompida, o cliente não sabe qual ou quais dos seus pedidos foram de facto executados pelo servidor.

Utilização simultânea de múltiplas conexões TCP

Como é fácil de antever, para além do protocolo HTTP, mas com este intimamente relacionado, existem diversos factores com um impacto significativo no desempenho do acessos a conteúdos. Esses factores dizem respeito à forma como as páginas são concebidas, aos limites da rede que liga os clientes aos servidores, e à forma como os *browsers* Web utilizam o protocolo HTTP.

Os *designers* de páginas Web e de aplicações acessíveis por essa via necessitam de as tornar muito atractivas e cheias de funcionalidades. Essa tendência tem como repercução a explosão do número de componentes que formam a página, incluindo muitos elementos gráficos, botões, diversas bibliotecas de código executável (*e.g.*, JavaScript), *etc.* O HTTP e o HTML conduzem à necessidade, pelo menos no momento de carregamento de páginas recheadas de elementos gráficos, da execução de dezenas de interacções HTTP entre o cliente e o servidor.

Utilizando HTTP 1.0 isso traduz-se numa explosão do número de conexões, uma por objecto. Com HTTP 1.1 a conexão TCP pode ser reutilizada mas o número de pedidos / resposta, envolvendo cada um pelo menos um RTT adicional, resulta de novo

num impacto significativo sobre o tempo de transferência. Isso pode ser melhorado com *pipelining* mas o seu impacto é limitado e mesmo indesejável em certas circunstâncias.

Com efeito, a definição do *pipelining* e do HTTP implica que todos as respostas são serializadas (enviadas por ordem), *i.e.*, as respostas do servidor têm de vir exactamente pela ordem com que os pedidos foram feitos. O cliente quando faz os pedidos não sabe qual a melhor forma de os ordenar e, algumas das respostas têm de ser geradas dinamicamente a partir de bases de dados. O resultado é que não existe nenhuma paralelização das interacções entre o cliente e o servidor, do que resulta que uma resposta longa ou lenta de calcular atrasa todas as seguintes, um fenómeno designado por *head-of-line-blocking*. Como resultado, o cliente não pode apresentar a página, mesmo numa versão incompleta, com a celeridade desejada. Diversos estudos experimentais mostram que os utilizadores têm tendência a abandonar os *sites* lentos.

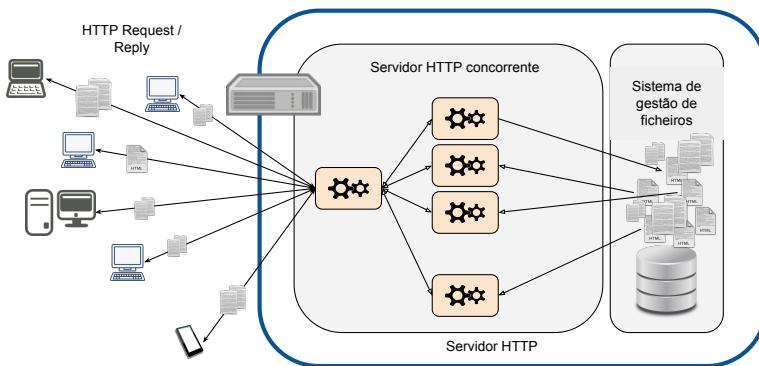


Figura 12.7: Servidor HTTP processando múltiplas conexões HTTP em paralelo através de diversos *threads* de execução

Este problema é comum nos sistemas de operação e nos sistemas distribuídos e levou à introdução de mecanismos de execução concorrente (*e.g.*, *threads*) que suportam paralelismo na execução dos pedidos recebidos. Todos os servidores HTTP em produção admitem pedidos em paralelo, ver a Figura 12.7, mas requerem, mesmo com HTTP 1.1, que os mesmos sejam suportados em outras tantas conexões TCP. Com *pipelining* isso é impossível. Por esta razão a quase totalidade dos *browsers* Web opta por abrir simultaneamente e *a priori* várias conexões TCP para o mesmo servidor, e não recorre ao *pipelining* para controlar melhor o progresso da execução dos pedidos HTTP. O número de conexões é parametrizável, mas os números 4 e 6 são as opções por omissão mais comuns.

A Figura 12.8, permite, nas mesmas hipóteses teóricas já colocadas, comparar a utilização de várias conexões TCP HTTP 1.1 em paralelo com a utilização de uma única sem *pipelining*. A solução permite obter, grosso modo, as vantagens do *pipelining* do ponto de vista de melhor tempo para obtenção da página completa, mas sem os riscos e os inconvenientes deste. Por outro lado, o impacto de eventuais perdas de pacotes é diluído nas várias conexões e o cliente poderá ainda obter uma fração superior da capacidade disponível na rede entre si e o servidor, visto que esta fração é proporcional ao número de conexões em paralelo. Esta solução tem, no entanto, o inconveniente de sobrecarregar o servidor com várias conexões.

Finalmente, para melhorar o desempenho é também possível modificar as páginas e o servidor. A principal modificação do lado dos servidores consiste em tentar diminuir o número de objectos computados dinamicamente (ou pelo menos fazer *caching* nos servidores de versões pré-calculadas das respostas aos pedidos mais comuns, como

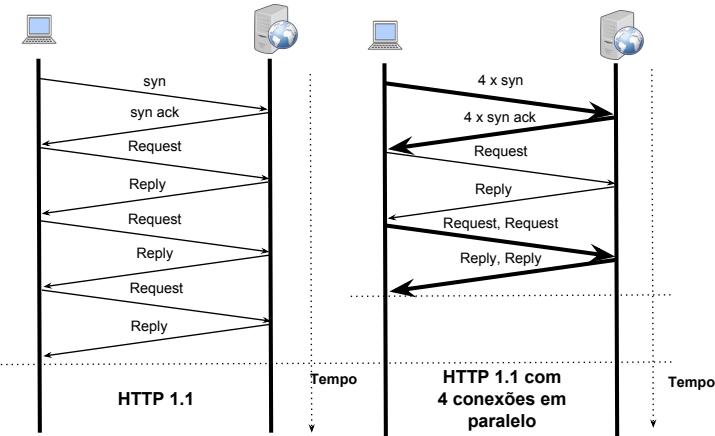


Figura 12.8: Tempo para obter uma página Web e dois objectos nela referenciados com uma conexão TCP e sem *pipelining*, e com 4 conexões TCP paralelas. O diagrama considera, por hipótese, que cada pedido e resposta é executado num único RTT.

fazem os motores de busca mais populares) e também aumentar a capacidade do servidor através da sua paralelização em servidores *multi core*.

HTTP 2

Naturalmente, o RTT e o débito extremo a extremo da rede entre o cliente e o servidor têm um grande impacto no desempenho do protocolo. No entanto, a partir de um débito razoável (*e.g.*, 10 Mbps), com páginas com muitas componentes de pequena dimensão, o factor determinante que influencia o desempenho é o RTT e não o débito. Ora, infelizmente, o RTT é mais elevado nas redes móveis celulares, o que agrava o problema nos sistemas móveis.

Estes estudos e conclusões levaram ao desenvolvimento de uma nova versão do protocolo HTTP, designada HTTP 2, ver [Grigorik, 2013] e o RFC 7540. Do ponto de vista desta discussão, o que é mais relevante no HTTP 2 é a introdução da noção de um mecanismo de multi-fluxo (*multi-streaming*) na conexão HTTP entre o cliente e o servidor, semelhante à mesma noção já presente no protocolo SCTP, ver a Secção 10.2.

Este mecanismo permite que o cliente e o servidor implementem vários fluxos independentes e paralelos sobre a mesma conexão TCP. O objectivo é obter as vantagens de várias conexões em paralelo, sem os inconvenientes assinalados quer para os servidores, quer para a partilha equitativa da rede. Uma outra funcionalidade introduzida consiste em o servidor poder tomar a iniciativa de enviar logo objectos para o cliente por antecipar que este os vai pedir a seguir. Trata-se de um mecanismo um pouco na mesma linha do usado pelo DNS (*Additional section*).

Mas, como não há bela sem senão, a solução torna esta versão do protocolo HTTP bastante mais complexa. Para alguns investigadores algumas outras opções da mesma são controversas [Kamp, 2015], nomeadamente o facto de o HTTP 2.0 ter suprimido o uso de simples conexões HTTP, directamente sobre TCP, e só admitir transferências de mensagens HTTP sobre conexões autenticadas e cifradas sobre TCP.

Em jeito de conclusão, para melhorar a eficiência do acesso a conteúdos é possível modificar as páginas disponibilizadas pelos servidores, melhorar o protocolo HTTP e

a forma como o cliente dialoga com o servidor. No entanto, é também possível melhorar o desempenho das aplicações introduzindo servidores suplementares, ou mesmo redes de servidores. Esse tipo de soluções são o objectivo específico do Capítulo 13. Antes, na secção que se segue, vamos analisar ainda alguns dos mecanismos presentes no protocolo HTTP para suportar diversas funcionalidades fundamentais para o desenvolvimento de aplicações Web.

O protocolo HTTP é responsável por uma fracção muito significativa do tráfego da Internet, pelo que o seu desempenho tem grandes implicações. Logo nos primeiros anos de utilização do protocolo foram introduzidos vários mecanismos para melhorar o seu funcionamento, nomeadamente, a utilização extensiva de **caching em clientes e a reutilização das conexões TCP**, introduzida na versão HTTP 1.1.

Para melhorar ainda mais o desempenho, a maioria dos *browsers* Web mais comuns optam por estabelecer *a priori* várias conexões TCP paralelas entre um cliente e um servidor. A necessidade de introduzir vários fluxos de pedidos e respostas em paralelo, levou à introdução desta funcionalidade na versão HTTP 2.

Cedo também se reconheceu que havia que apoiar simples clientes e servidores com um conjunto de servidores suplementares que aumentassem a escalabilidade da distribuição de conteúdos. Esta faceta será estudada no Capítulo 13.

12.3 O protocolo HTTP e a Web actual

Com a explosão da Web durante a década de 1990, a prática de suportar aplicações distribuídas com recurso a HTTP e HTML tornou-se uma prática dominante, e os *browsers* Web tornaram-se clientes genéricos de aplicações distribuídas.

Para enriquecer as interfaces e torná-las mais eficientes, os *browsers* Web passaram igualmente a executar (de forma interpretada) código escrito numa linguagem de programação, transmitido com as páginas Web via HTTP. Inicialmente usaram-se Java Applets interpretados por uma Java Virtual Machine integrada no *browser*, mas mais tarde generalizaram-se linguagens de *scripting*, entre as quais a linguagem JavaScript tornou-se das mais populares. Esta prática é coloquialmente designada por *client-side scripting*. Desta forma os *browsers* Web passaram a ter cada vez mais funcionalidades, a serem capazes de interpretar muitos tipos de dados (*e.g.*, textos, filmes, sons, gráficos, vídeos, animações, *etc.*), de estabelecerem diálogos com o utilizador, de apoiarem o preenchimento de *forms* complexos, de executarem verdadeiras aplicações interactivas, e as suas funcionalidades são constantemente actualizadas via a própria Web através do carregamento de código executável (*e.g.*, JavaScript, *plug-ins* específicos). Por outro lado, cada vez mais as páginas cresceram em complexidade e número de objectos distintos.

As aplicações acessíveis via a Web passaram a ser suportadas, do lado do servidor, numa arquitectura em camadas como ilustra a Figura 12.9. Para aumentar a flexibilidade de desenvolvimento de aplicações foram também desenvolvidos muitos *frameworks* de desenvolvimento de aplicações acessíveis via a Web (*e.g.*, Java Server Pages, PHP, *etc.*).

Estes requisitos implicaram uma grande evolução do HTML, e também a generalização do uso de *frameworks* de execução de aplicações dentro dos *browsers* Web (*e.g.*, HTML5, AJAX, *etc.*), mas levaram igualmente à introdução ou refinamento de mecanismos suplementares no HTTP. A seguir vamos ver alguns dos mecanismos existentes

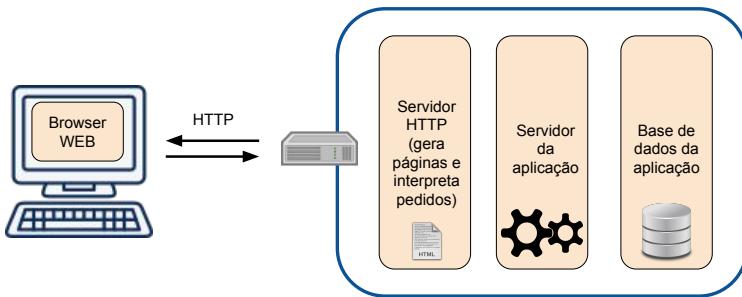


Figura 12.9: Arquitectura genérica de uma aplicação acessível via a Web (parte servidor)

no HTTP com o objectivo de facilitar o suporte de aplicações distribuídas.

Passagem de parâmetros no URL

O HTML suporta entidades designadas por *forms* que permitem apresentar um questionário ao utilizador, recolher as suas respostas, e enviá-las para o servidor via o comando **POST**. No entanto, quando se pretendem passar parâmetros numa operação sem para isso ter de os pedir ao utilizador através de um *form*, é necessário utilizar outros mecanismos.

Um dos mecanismos mais simples para passar parâmetros ao servidor, disponível logo nas versões iniciais do protocolo, é a passagem de parâmetros no URL. Para esse efeito o URL é complementado com um conjunto de pares (nome do parâmetro = valor).

Sintaxe geral de um URL com o nome de uma operação e parâmetros:

https ou http://servidor[:porta]/operação[?parâmetros]

em que a parte parâmetros, quando presente, toma a forma de um ou mais pares (nome = valor) separados pelo carácter '&' e com a forma genérica:

parâmetro=valor&parâmetro=valor*

Exemplos:

```
https://www.socialnet.com/profile.php?id=100014350098345
http://www.search.com/search?language=english&q=http+tutorial
```

Os servidores Web dispõem de mecanismos para recolher esses parâmetros e passá-los às componentes software encarregues de calcular a resposta a enviar ao cliente.

Autenticação

A utilização da Web para aceder a objectos com acesso controlado exige a utilização de autenticação. O mesmo se aplica a inúmeras aplicações distribuídas. A especificação actual deste mecanismo do HTTP consta do RFC 7235 de Junho de 2014, mas o esquema existe desde o HTTP 1.0.

Quando o servidor recebe um pedido HTTP que só pode satisfazer se o utilizador se autenticar, responde com um código de erro HTTP 401 **Unauthorized status** e envia igualmente um *header field* **WWW-Authenticate** como veremos no exemplo a seguir de pedido seguido de resposta HTTP:

```
GET /secretpage HTTP/1.1
Host: www.only4dummypies.com
```

```
HTTP/1.1 401 Access Denied
WWW-Authenticate: Basic realm="Info for spies"
Content-Length: 0
```

O valor a seguir a **realm** é opcional e é apenas usado pelos servidores que pretendem indicar um domínio de proteção (*protection space*) ao cliente. Quando recebe a resposta, o *browser* Web abre uma caixa pedindo ao utilizador um nome de utilizador e uma palavra passe, e deve depois reenviar o pedido mas incluindo um *header field* **Authorization** como a seguir ilustrado.

```
GET /secretpage HTTP/1.1
Host: www.only4dummypies.com
Authorization: Basic aHR0cHhdGNoOmY=
```

onde “aHR0cHhdGNoOmY=” representa “utilizador:palavra-passe” codificado num formato especial na base de uma sequência de caracteres US-ASCII e designado por Base64⁵ que, apesar de sugerir que se usa cifra, é enganador pois os valores são de facto passados em claro. Hoje em dia só tem sentido utilizar esta forma de autenticação com canais autenticados e cifrados sobre TCP (HTTPS).

O *browser* Web faz *caching* em memória do par (utilizador, palavra-passe) e reenvia o *header field* **Authorization** sempre que o utilizador fizer pedidos ao mesmo URL ou a URL subordinados. Desta forma o servidor não necessita de memorizar *a priori* dados sobre o cliente e as operações continuam a manter o seu carácter de idempotência.

Existe uma variante de codificação do par (utilizador: palavra passe) chamada **Digest** ao invés de **Basic**, em que a informação passada é o resultado da codificação de um *digest* do par: (utilizador: palavra passe). Esta opção esconde a palavra passe mas não resiste a um ataque do tipo *replaying*. Este consiste em copiar o *header field* **Authorization** e reenviá-lo posteriormente para o mesmo servidor a partir de outro cliente. Portanto, esta forma de autenticação só é segura usando igualmente HTTPS para impedir a cópia do *header field* **Authorization**. Modernamente esta forma de autenticação caiu em desuso, pois foram introduzidas outras alternativas, algumas das quais serão explicadas a seguir.

Cookies

O HTTP foi definido de forma a que a maioria das operações fosse idempotente. No entanto, muitas aplicações Web usam interfaces e operações que não o são, como por exemplo as de aquisição de bens via a Web. Torna-se assim claro que é necessária uma noção de sessão, *i.e.*, uma sequência de pedidos / respostas correlacionados, e identificar os pedidos feitos pelo mesmo utilizador durante a sessão para os associar entre si. Assim, cada sessão pode ter um identificador, que é gerado pelo servidor no seu início, e que o cliente deve enviar como um parâmetro suplementar em todos os pedidos seguintes.

Mais tarde veio-se a constatar que caso o cliente memorizasse de forma persistente o identificador, este poderia ser enviado ao servidor numa sessão posterior, mesmo

⁵A representação Base64 é conceptualmente equivalente a hexadecimal, mas é mais eficiente. A mesma é discutida num dos exercícios do fim do capítulo.

noutro dia, o que permitiria identificar a nova sessão como pertencendo ao mesmo utilizador.

Estes identificadores foram chamados *magic cookies* ou simplesmente *cookies*⁶. Os *cookies* são gerados pelos servidores e ficam associados a utilizadores para identificarem sessões ou os próprios utilizadores posteriormente. Neste último caso chamam-se *persistent cookies*. Os *persistent cookies*, quando utilizados com canais autenticados e cifrados sobre TCP (HTTPS), chamam-se *secure persistent cookies* e podem ser usados para autenticar os utilizadores. Os *cookies* foram introduzidos em 1994 nos *browsers* Web da Netscape e normalizados pelo IETF em 1997 pelo RFC 2109. A sua mais recente especificação é de 2011, no RFC 6265. O seu modo de funcionamento é explicado a seguir de forma breve.

Quando um servidor envia uma resposta HTTP, pode utilizar um ou mais *header fields* **Set-Cookie** para enviar *cookies* para o cliente. Cada um destes *header fields* transmite ao cliente um *cookie* diferente e um conjunto de atributos do mesmo. O único atributo obrigatório de um *cookie* é o nome, que permite definir um nome e um valor, como se fosse um par (atributo ou variável, valor). Outros atributos dizem respeito ao tipo de *cookie*. Por exemplo, a presença de um atributo **Expires**=*data* indica que o *cookie* é persistente e que deve ser memorizado de forma persistente pelo *browser* Web até àquela *data* (excepto quando *data*=0 o que indica que o *cookie* não deve ser memorizado). Na ausência deste atributo, o *cookie* é de sessão e deve ser suprimido pelo *browser* Web quando a janela onde foi adquirido for fechada. Outros atributos são o domínio e o *path* que serão explicados a seguir.

Para transmitir um *cookie* ao servidor é usado o *header field* **Cookie** que permite enviar um ou mais *cookies* ao servidor a que estes se aplicam. Seguem-se alguns exemplos.

Pedido inicial do cliente:

```
GET /compras HTTP/1.1
Host: www.paraiso.pt
```

Resposta do servidor:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: session=1274126492323
Set-Cookie: userToken=user1412610421; Expires=Wed, 02 Jun 2021 22:18:13 GMT
....
```

Pedido seguinte do cliente:

```
GET /compras/ HTTP/1.1
Host: www.paraiso.pt
Cookie: session=1274126492323; userToken=user1412610421
.....
```

Com os *cookies* e com os atributos do exemplo (ausência de atributos **Domain** e **Path**), os mesmos devem ser enviados sempre que o cliente usar URLs da forma `http[s]://www.paraiso.pt/ ...` i.e., dirigidos exactamente ao mesmo servidor e página, ou a páginas subordinadas a esta. O de sessão durante essa sessão, e o persistente, mesmo em sessões futuras, até este expirar.

No entanto, um *cookie* pode ser associado a um URL diferente. Tal é possível quando o mesmo inclui atributos **Domain** ou **Path** específicos, pois estes indicam explicitamente quais os URLs a que o *cookie* deve ser enviado pelo cliente sempre que as páginas indicadas por esses atributos forem visitadas. Se por um lado, isso põe em

⁶Um *magic cookie* é a designação popular nos EUA de um pequena bolacha com uma prenda ou um segredo escondido lá dentro.

evidência a flexibilidade do mecanismo, mostra também que o mesmo pode ser usado de forma perversa. Por exemplo, se o *browser* Web do utilizador aceitasse que este memorize um *cookie* associado a um domínio acima da página visitada *e.g.*, `.com`, a partir daí esse *cookie* seria enviado a todos os servidores com um nome por baixo de `.com`. Como um *cookie* pode ter até 4 Kbytes de informação associada, muitos dados poderiam ser codificado no mesmo.

Os *cookies* permitem de facto fazer o seguimento (*trace*) da actividade dos utilizadores da Web. Por exemplo, se um servidor quiser contar o número de vezes que um dado utilizador o visita, mesmo sem este se autenticar, basta enviar-lhe um *cookie* com um identificador único da primeira vez, que passará a detectar nas visitas seguintes, o que permite fazer a contagem. Também é possível fazer essas correlações mesmo que o utilizador não visite explicitamente a página do servidor, pois o *cookie* pode ser instalado por um simples anúncio (*e.g.*, um *banner*) presente na página por este visitada. Com um *cookie* no *banner*, específico para cada utilizador, seria possível contar quantas vezes cada utilizador vê o anúncio (ou pelo menos o mesmo está presente numa página que este visitou).

O mecanismo pode também ser usado para obter um perfil do utilizador. Suponha, por exemplo, que a empresa Publicidade Ilimitada coloca anúncios dos seus clientes em páginas, mediante pagamento a quem os aceita mostrar aos seus visitantes. Esta é uma prática comum para subsidiar os custos de serviços “prestado gratuitamente”. O servidor de distribuição de anúncios é o servidor `publicity.net`. Um dos distribuidores da publicidade de Publicidade Ilimitada é a empresa Serviços Grátis Limitada, cujo servidor é `free.net`. Os seus serviços são afinal pagos através da publicidade feita por conta de Publicidade Ilimitada.

Assim, anúncios de Publicidade Ilimitada aparecem na página `free.net` através de um grafismo embebido com um URL na sua página inicial, correspondente a um *banner* que aponta para o servidor `publicity.net`, e para mostrar a página e a publicidade, o *browser* Web do visitante tem de seguir esse URL. O URL tem um parâmetro que permite a `publicity.net` saber que o pedido vem de `free.net`. O servidor de `publicity.net` regista, a quando do pedido, que tem de pagar mais um centímo a Serviços Grátis Limitada e envia-lhe um *cookie*, gerado no momento, que o visitante de `free.net` vai registar no seu disco.

Quando o visitante visitar outra página com anúncios de `publicity.net`, este servidor vai receber o *cookie* que enviou ao visitante da última vez, e envia-lhe outro novo⁷. A pouco e pouco, o servidor `publicity.net` vai ficando com a história das visitas feitas por este utilizador, sabe que páginas este visita, e começa a tornar a sua publicidade cada vez mais cara. Com efeito, de facto Publicidade Ilimitada passa a poder enviar os anúncios apenas aos seus potenciais interessados, algo que é muito valorizado pelos seus clientes.

Como um *cookie* pode ter até cerca de 4 Kbytes, o servidor pode codificar no mesmo imensa informação que vai ser posteriormente reenviada pelo cliente. Como é evidente, com um pouco de imaginação, o céu é o limite na utilização de *cookies*. No limite, se o *banner* se resumir a um único pixel, imperceptível pelo utilizador, tudo funciona mesmo sem nenhum anúncio explícito na página. Geralmente, os utilizadores, sem se aperceberem, acumulam facilmente centenas de *cookies* persistentes nos seus *browser* Web.

A maioria dos *browsers* Web incluem hoje em dia várias alternativas para bloquear *cookies*, ou para bloqueio parcial dos mesmos. No entanto, muitas páginas não conseguem funcionar correctamente sem *cookies*, pelo que os utilizadores são forçados a não usar esses bloqueios.

Devido a estas controvérsias, e a alguma legislação de combate a abusos, foram introduzidas regras de conduta e outros mecanismos de alerta dos utilizadores. No

⁷ O novo *cookie* pode ser acompanhado da anulação do anterior. Basta para isso que o anterior seja de novo enviado com o atributo `Expires` com uma data no passado.

entanto, o assunto necessitaria que os utilizadores o compreendessem em profundidade, e tivessem formação jurídica e tempo para lerem dezenas de páginas com os termos e condições de utilização dos serviços. O resultado final continua a ser a troca da privacidade por serviços teoricamente grátis, pois o modelo do negócio dos gigantes do sector passa por quebrar, tanto quanto possível, essa mesma privacidade.

Os *cookies* persistentes providenciam também um método de autenticação, permitindo o reconhecimento posterior de um utilizador, quando este se autenticou antes perante a mesma página. *Cookies* de autenticação só podem ser enviados por conexões seguras sob pena de poderem ser copiados por atacantes. Os utilizadores também não devem usar essas aplicações em computadores de utilização pública, pois esses *cookies* podem ficar registados pelos *browsers* Web desses computadores. Também podem, se souberem como, apagar todos os *cookies* ao fecharem o *browser*.

De facto, quando os *cookies* são usados para facilitar a autenticação em futuras visitas a uma página, os mesmos adquirem as propriedades de uma capacidade, e podem tornar-se um perigo caso sejam roubados, e o servidor da página visitada não tomar medidas suplementares para verificar a autenticação do visitante. Essas precauções suplementares devem ser usadas pelo menos quando o utilizador executa acções que conduzem a pagamentos, a mudanças da palavra chave ou outras com futuras implicações de segurança. Nestes casos começa a ser cada vez mais frequente recorrer-se a mecanismos de autenticação suplementares (pedir de novo a palavra chave, desafios com perguntas, desafios com códigos via telemóveis associado ao utilizador, *etc.*).

O leitor interessado pode consultar os *cookies* memorizados pelo seu *browser* Web visitando as opções de privacidade do mesmo.

Os *cookies* podem ser usados para introduzir uma noção de sessão, para seguimento dos utilizadores e para autenticação.

A sua utilização para o desenvolvimento de aplicações complexas e com interações prolongadas via a Web, *i.e.*, com a noção de sessão (*e.g.*, aquisições e outros serviços envolvendo transações prolongadas via a Web) e suporte de autenticação tornou-se imprescindível.

No entanto, devido a vários **problemas de segurança e privacidade**, têm de ser utilizados de forma cuidadosa.

Web Services

Como o HTTP é um protocolo raramente bloqueado pelos equipamentos de controlo de segurança (*e.g.*, *firewalls*) tornou-se atraente usá-lo para suporte de interacções entre componentes de aplicações distribuídas por diferentes computadores ligados à Internet.

A noção de API (*Application Program Interface*) distribuída permite a um programa cliente obter serviços de um servidor remoto. Com a realização do conceito a seguir explicada, essas interfaces designam-se frequentemente por *Web Services*.

A ideia é relativamente simples, ver a Figura 12.10, um servidor disponibiliza serviços caracterizados por uma API remota. Esses serviços podem ser invocados enviando mensagens para o servidor, com a indicação da operação a executar e contendo os parâmetros de entrada da mesma. O servidor responderá com os parâmetros do resultado.

Como estar a codificar a mensagem e a colocar dentro da mesma os parâmetros (que têm de ser codificados da forma que o servidor os perceba) é um processo semelhante para todas as operações e portanto automatizável, é criada uma outra API, a API local, com a mesma interface que a operação remota, mas cujo papel é simplesmente preparar a mensagem com o pedido, enviá-la para o servidor, esperar pela resposta e

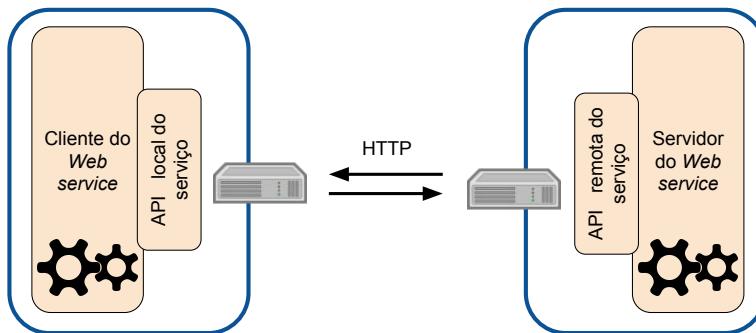


Figura 12.10: Um cliente usa os *Web services* exportados por um servidor através de uma API local que representa para o código do cliente a interface do servidor.

devolver os parâmetros da resposta ao cliente. Ou seja, a API remota representa para o código do servidor o cliente, e a API local ao cliente, representa o servidor no código do cliente. Ambas as APIs funcionam como representantes (*proxies* na terminologia em língua inglesa) da outra parte. Para que as duas APIs troquem mensagens, uma solução popular passa por usar o protocolo HTTP.

De facto, este proporciona a invocação remota e a passagem arbitrária de parâmetros de entrada através dos métodos POST e PUT que compreendem a possibilidade de envio de um objecto para o servidor. Esse objecto pode codificar, numa representação pré-acordada, um objecto de dados contendo os dados a transmitir ao servidor com o pedido, *i.e.*, os parâmetros de entrada da operação. Na resposta, o servidor pode igualmente devolver um objecto de dados codificados no mesmo formato, contendo a resposta à invocação remota da operação. A interface do servidor remoto assim acessível é endereçável através de um URL, e este pode também ser usado para codificar o nome da operação a invocar.

Esta forma de organização do diálogo entre um cliente e um servidor tem sido utilizada extensivamente para permitir a invocação remota de serviços através de uma API remota. Praticamente todas as aplicações sofisticadas que executam dentro de *browsers* Web e nos clientes móveis utilizam *Web Services* e o protocolo HTTP como transporte da invocação remota. Adicionalmente, o mecanismo tornou-se igualmente popular para dar acesso através da Internet a serviços remotos, mesmo quando o cliente é ele próprio um programa (*Machine-to-Machine Communication*).

Para que programas escritos em diversas linguagens de programação possam integrar de forma distribuída segundo este modelo, foram desenvolvidos *frameworks* de invocação remota de *Web services*, e normas para a representação dos dados, definição das interfaces, codificação e organização das mensagens *etc.* Estas agrupam-se hoje em dia em torno de duas opções: SOAP (Simple Object Access Protocol) e REST (REpresentational State Transfer). A descrição desses *frameworks* ultrapassa os objectivos deste livro. O leitor interessado pode consultar a bibliografia sugerida na Secção 12.4.

Finalmente, não podemos deixar de referir uma outra utilização do HTTP para facilitar o desenvolvimento de aplicações destinadas a serem utilizadas localmente a um computador, mas relativamente independentes do sistema de operação e da interface do computador. Essas aplicações incluem consigo um pequeno servidor WEB e a interface com o utilizador local é feita com recurso a um *browser* Web. Desta forma, a aplicação torna-se bastante independente do tipo de funcionalidades gráficas e das interfaces disponíveis em qualquer sistema em que seja instalada.

O HTTP é um protocolo que foi inicialmente concebido para acesso a documentos remotos, mas transformou-se a pouco e pouco num protocolo de suporte a serviços distribuídos, jogos, difusão de conteúdos, *streaming* de vídeo, *etc.*

Este tipo de utilizações foi possível pois encontraram-se formas de completar a sua definição, passando a incluir várias formas de passagem de parâmetros, autenticação, gestão do estado das sessões, execução local de código móvel obtido a partir de servidores (*client-side scripting*), *etc.*

Muita desta flexibilidade foi possível pois o protocolo foi cuidadosamente definido no que diz respeito à sua faceta cliente / servidor, à idempotência da sua interface, ao carácter predominantemente sem história prévia das invocações, à extensibilidade dos cabeçalhos recorrendo à introdução e extensão dos *header fields*, à uniformidade da interface, à flexibilidade da noção de URL e à possibilidade de carregar código a pedido.

12.4 Resumo e referências

O protocolo **HTTP** (*Hyper Text Transfer Protocol*) é um protocolo cliente / servidor genérico, de transferência entre servidores e clientes: de documentos HTML, de ficheiros em qualquer formato, de todo o tipo de conteúdos digitais, incluindo conteúdos multimédia, e ainda de código de programas para serem interpretados e executados pelos *browsers* Web.

A linguagem **HTML** (*Hyper Text Markup Language*) é uma linguagem que permite introduzir anotações (*tags*) num texto. Essas anotações não modificam o significado do texto mas servem para indicar a sua relação com outras partes do texto, ou outros textos, ou ainda para controlar a forma como um texto deve ser formatado e visualizado.

O HTTP funciona sobre conexões TCP e só tem duas mensagens: HTTP Request (pedido) e HTTP Reply (resposta). Cada mensagem de pedido só pode pedir um objecto ao servidor e cada mensagem de resposta só pode conter um objecto.

É possível realizar diversos pedidos do cliente ao servidor, que se traduzem em outros tantos métodos (semelhantes a códigos de operação) na mensagem de pedido. Os pedidos de utilização mais frequentes são: o método GET para solicitar um objecto, o método POST para transmitir um formulário preenchido para o servidor, e o método PUT para transmitir um objecto para o servidor.

As mensagens HTTP contém cabeçalhos estruturados como um conjunto de linhas de texto terminadas necessariamente no par ordenado de caracteres **carriage return**, **linefeed** e o cabeçalho termina numa linha em branco. Com excepção da primeira linha dos cabeçalhos do pedido e do resultado, as outras linhas designam-se por *request / response header lines*, podem variar em número, e permitem ao cliente e ao servidor trocarem diversas informações (*e.g.*, os meta dados sobre os objectos transmitidos) e condicionar a execução dos diferentes métodos. Esta forma de estruturar os cabeçalhos é muito extensível, flexível e também facilita o *debug*.

Uma **operação idempotente** (*idempotent operation*) é uma operação cuja execução repetida, mesmo que devolva um valor diferente em cada caso, não conduz a erros aplicacionais, nem no servidor nem no cliente.

Se um servidor tem uma interface exclusivamente baseada em operações idempotentes, diz-se um **servidor sem estado** (*stateless server*) e torna-se mais simples porque não tem de detectar operações executadas mais do que uma vez. Parte das ope-

rações executadas pelos servidores HTTP também são idempotentes, nomeadamente os métodos GET e PUT.

O protocolo HTTP é responsável por uma fracção muito significativa do tráfego da Internet, pelo que o seu desempenho tem grandes implicações. Logo nos primeiros anos de utilização do protocolo foram introduzidos vários mecanismos para melhorar o seu funcionamento, nomeadamente, a utilização extensiva de *caching* nos clientes e a reutilização das conexões TCP, introduzida na versão HTTP 1.1. Para melhorar ainda mais o desempenho, a maioria dos *browsers* Web mais comuns optam por estabelecer *a priori* várias conexões TCP paralelas entre um cliente e um servidor. A necessidade de introduzir vários fluxos de pedidos e respostas em paralelo, levou à introdução desta funcionalidade sobre uma única conexão TCP no HTTP 2.

O HTTP é um protocolo que foi inicialmente concebido para acesso a documentos remotos, mas transformou-se a pouco e pouco num protocolo genérico de suporte a serviços distribuídos, jogos, difusão de conteúdos, *streaming* de vídeo, *etc.* Este tipo de utilizações foi possível pois encontraram-se formas de completar a sua definição, passando a incluir várias formas de passagem de parâmetros, autenticação, gestão de estado das sessões, execução local de código obtido a partir de servidores (*client-side scripting*), *etc.*

Muita desta flexibilidade foi possível pois o protocolo foi cuidadosamente definido no que diz respeito à sua faceta cliente / servidor, à idempotência do essencial da sua interface, ao carácter predominantemente sem história prévia das invocações, à extensibilidade dos cabeçalhos recorrendo à introdução e extensão dos *header fields*, à uniformidade da interface, à flexibilidade da noção de URL e à possibilidade de carregar código a pedido.

Alguns dos termos introduzidos no capítulo são a seguir passados em revista. Entre parêntesis figura a tradução mais comum em língua inglesa.

Principais conceitos

HTTP (*Hyper Text Transfer Protocol*) protocolo do tipo cliente / servidor genérico, de transferência entre servidores e clientes: de documentos HTML, de ficheiros de qualquer tipo e codificados de forma arbitrária, de todo o tipo de conteúdos digitais incluindo conteúdos multimédia e ainda de código executável por interpretação pelos *browsers* Web.

HTML (*Hyper Text Markup Language*) Linguagem que define um conjunto de marcas (*tags*) associadas a um texto para indicar a sua relação com outras partes do texto, ou para controlar a forma gráfica como o mesmo deve ser visualizado.

URL (*Uniform Resource Locator*) Referência para um objecto que reside num servidor. Inclui a indicação do protocolo a executar com o servidor, o seu nome e porta, e a referência para o objecto no servidor.

GET Comando HTTP que permite solicitar ao servidor o objecto referenciado no URL.

PUT Comando HTTP que permite transmitir para o servidor um objecto que será referenciado pelo URL.

POST Comando HTTP que permite transmitir para o servidor um objecto contendo geralmente um conjunto de parâmetros (pares atributo / valor).

Operação idempotente (*idempotent operation*) Operação cuja execução repetida, mesmo que devolva um valor diferente em cada caso, não conduz a erros aplicacionais, nem no servidor, nem no cliente. As operações GET e PUT do protocolo HTTP são idempotentes.

Servidor sem estado (*stateless server*) Servidor com uma interface só com operações idempotentes. Estes servidores tornam-se mais simples porque não têm de detectar operações executadas mais do que uma vez, nem guardar estado sobre os seus clientes.

Pipelining Possibilidade de um cliente enviar vários pedidos seguidos para o servidor sem obter antes as respostas. As diferentes operações executadas usando *pipelining* têm de ser necessariamente serializadas. A maioria dos *browsers* Web não usa este mecanismo preferindo usar conexões em paralelo, ou HTTP 2 que já inclui um mecanismo de paralelização de pedidos sobre uma única conexão TCP.

Caching Mecanismo que permite acelerar o acesso a documentos remotos com base em cópias locais obtidas previamente.

Cookie Mecanismo que permite a um servidor gerar um identificador que será incluído nos futuros pedidos do mesmo cliente, e que serve para implementar a noção de sessão, autenticação e seguimento / correlação da actividade dos utilizadores. A sua utilização, especialmente com o último objectivo, tem sido sujeita a controvérsia e cuidados legais para proteção da privacidade dos utilizadores. A sua utilização para autenticação deve revestir-se de cuidados especiais.

Referências

Um dos artigos mais conhecidos do inventor da Web, Tim Berners-Lee, foi publicado na revista Publications of The ACM [Berners-Lee et al., 1994]. Muitas das suas opções sobre o protocolo HTTP inspiraram-se na noção de invocação remota (*Remote Procedure Call*) [Birrell and Nelson, 1984] e no seu trabalho prévio sobre este tópico.

No momento da escrita deste livro, as últimas versões mais relevantes das normas sobre o HTTP são as seguintes RFCs: RFC 7230 a 7235, que cobrem os seguintes tópicos respectivamente: *Message Syntax and Routing*, *Semantics and Content*, *Conditional Requests*, *Range Requests*, *Caching* e *Authentication*. O HTTP 2 é definido no RFC 7540.

Os livros [Gourley et al., 2002; Allen, 2012] contém uma boa descrição do protocolo e das implicações dos seus diferentes mecanismos. O primeiro, apesar de ter sido publicado em 2002, continua a ser uma boa referência para um estudo técnico profundo do protocolo HTTP. O segundo é mais simples, directo e actual. A versão HTTP 2 é apresentada em [Grigorik, 2013].

O leitor interessado em conhecer o tema dos *Web Services* poderá consultar a imensa literatura e os tutorials existentes sobre o assunto. O capítulo sobre aplicações do livro [Peterson and Davies, 2012] contém uma breve introdução, assim como uma comparação entre SOAP e REST. Para aqueles que pretendem fazer desenvolvimentos usando este tipo de *frameworks*, o livro [Webber et al., 2010] apresenta uma introdução ao tema com base em REST.

Apontadores para informação na Web

- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- <http://www.w3.org/Consortium> – É o site oficial do W3C, um consórcio com um papel preponderante na normalização da linguagem HTML e vários dos seus *frameworks* e linguagens complementares.

No seu site *on-line* a sua missão é definida da seguinte forma: “*The World Wide Web Consortium (W3C) is an international community where Member organizations, a full-time staff, and the public work together to develop Web standards. Led by Web inventor Tim Berners-Lee and CEO Jeffrey Jaffe, W3C’s mission is to lead the Web to its full potential. Contact W3C for more information.*”.

12.5 Questões para revisão e estudo

1. Verdade ou mentira?

Na pilha TCP/IP, o protocolo HTTP:

- (a) É um protocolo do nível aplicação suportado em TCP.
- (b) É um protocolo do nível transporte que está vocacionado para envio e recepção de mensagens entre clientes e servidores Web.
- (c) É um protocolo do nível aplicação suportado em UDP.
- (d) É um protocolo do nível transporte inspirado no protocolo TCP e que pressupõe que todas as mensagens são enviadas para o porto 80.

2. Qual destas afirmações é a que está certa?

Quando o browser no seu computador solicita uma página Web a um servidor está a usar:

- (a) HTTP sobre TCP.
- (b) HTTP sobre UDP.
- (c) HTTP sobre *stop & wait*.
- (d) FTP sobre UDP.

3. Verdade ou mentira?

- (a) As páginas `www.exemplo.pt` e `www.exemplo.pt/exemplo` podem ser transmitidas pela mesma conexão TCP usando o protocolo HTTP 1.1.
- (b) Com conexões TCP de suporte a HTTP 1.0 é impossível que um segmento TCP transporte dados de duas mensagens HTTP distintas.
- (c) Com conexões persistentes HTTP 1.1 é impossível que um segmento TCP transporte dados de duas mensagens HTTP distintas.
- (d) Em HTTP e usando conexões não persistentes, não é possível que um segmento TCP possa vir a transportar duas mensagens HTTP resultantes de dois pedidos.
- (e) As páginas `www.abc.pt`, `www.abc.pt:80` e `www.abc.pt:8080` têm de ser obtidas usando necessariamente pelo menos 2 conexões TCP distintas.
- (f) Uma mensagem HTTP *Reply* em HTTP 1.1 pode conter mais do que um objecto.

4. Um *browser* Web pretende obter a página `www.exemplo.pt/index.html`, já tem uma cópia da mesma na sua *cache* local, mas não sabe se a mesma corresponde à última versão. Qual ou quais os *header fields* que deve colocar na mensagem HTTP GET dessa página para a obter com a máxima eficiência?

5. A página `www.time.net/date.html` está sempre a mudar de 1 em 1 segundo. Qual ou quais os *header fields* que o servidor `www.time.net` deve colocar na mensagem HTTP de resposta quando a transmite?

6. Um cliente HTTP acede sucessivamente a várias páginas no mesmo servidor. Estas páginas têm uma dimensão significativa (*e.g.*, 1 Mbyte). Realizaram-se vários testes de conectividade entre o cliente e o servidor e chegou-se à conclusão que o RTT era significativo (cerca de 200 ms), mas que não havia perda de pacotes nem estrangulamentos no débito extremo a extremo.

- (a) Elabore um esquema temporal que permite pôr em evidência graficamente a superioridade da versão 1.1 sobre a versão 1.0 do protocolo HTTP neste quadro.

- (b) Suponha agora que as medidas feitas mostraram que o RTT era o mesmo mas que existiam períodos em que aparecia uma taxa de perda de pacotes significativa, alternando com períodos sem perda de pacotes. Neste novo quadro todas as vantagens do protocolo HTTP 1.1 ainda se mantêm?
- (c) No quadro da alínea anterior haveria alguma alternativa mais prudente a adoptar pelo cliente?
- (d) Faz sentido um servidor HTTP que não seja *multi-threaded* aceitar conexões persistentes HTTP 1.1?
7. Um cliente HTTP acede a uma página HTML num servidor. Depois de obter essa página, o cliente deduz que a mesma tem 2 imagens e que as mesmas devem ser obtidas igualmente, a partir desse mesmo servidor, para mostrar o conteúdo total ao utilizador. O tempo de trânsito ida e volta (RTT) entre o cliente e o servidor é de (cerca de) 100 mili segundos. O cliente não tem nenhuma conexão aberta para o servidor antes de aceder às páginas mas já conhece o seu endereço IP. O tempo necessário para transmitir os pacotes, com os cabeçalhos HTTP, a página ou as imagens são negligenciáveis e cada mensagem HTTP (pedido ou resposta) é transmitida de uma vez dado caber sempre num só pacote.
- (a) Qual o tempo necessário para o cliente obter a página com as imagens usando a versão 1.0 do protocolo HTTP?
- (b) Qual o tempo necessário para o cliente obter a página com as imagens usando a versão 1.1 do protocolo HTTP com *pipelining*?
- (c) Qual o tempo necessário para o cliente obter a página com as imagens usando a versão 1.1 do protocolo HTTP sem *pipelining*?
8. Entre um cliente e um servidor HTTP o RTT é de (cerca de) 20 mili segundos e este valor é estável. Para resolver as questões que se seguem despreze o tempo de processamento e o tempo de transmissão de todos os pacotes envolvidos. Admita também, por hipótese, que todas as mensagens de pedido HTTP cabem dentro de um único pacote IP e que não se perdem pacotes. As conexões TCP começam sempre a transmitir dados com o valor da *ConWnd* (*Congestion Window*) igual a um segmento.
- (a) Quanto tempo leva o cliente a obter uma página WWW usando o protocolo HTTP 1.0, sabendo que essa página cabe em 7 segmentos TCP? O cliente abre a conexão, pede a página e tem de obter a resposta completa.
- (b) Quanto tempo leva o cliente a obter duas páginas do mesmo servidor, idênticas em dimensão às da alínea anterior, mas usando uma conexão persistente HTTP 1.1? O cliente abre a conexão e envia um pedido de cada vez. Compare os tempos para obter cada uma das páginas e explique a diferença.
9. Um cliente está a interagir com 5 servidores distintos para obter, através do protocolo HTTP, 5 objectos de grande dimensão. Cada um dos objectos está num servidor distinto. Quais das seguintes alternativas correspondem ao menor tempo total para obter os 5 objectos por HTTP?
- (a) Utilizar HTTP sobre UDP para não ser limitado pelo controlo de saturação do TCP.
- (b) Abrir uma conexão persistente usando HTTP 1.1 para apenas um dos servidores e solicitar-lhe todos os objectos através de *pipelining*.
- (c) Obter cada um dos objectos sequencialmente através de conexões HTTP 1.0 sobre TCP para os diferentes servidores HTTP.

- (d) Obter cada um dos objectos sequencialmente através de conexões HTTP 1.1 sobre TCP para os diferentes servidores HTTP.
- (e) Obter cada um dos objectos em paralelo através de diferentes conexões HTTP 1.0 sobre TCP para os diferentes servidores HTTP.
- (f) Obter cada um dos objectos em paralelo através de diferentes conexões HTTP 1.0 sobre UDP para os diferentes servidores HTTP.
10. Um servidor Web pretende contar o número de vezes que é visitado por um dado utilizador U sem o autenticar, *i.e.*, contar número de vezes que recebe uma mensagem HTTP GET na sequência de um pedido do utilizador U. Como pode essa contabilização ser implementada nas seguintes alternativas:
- Sabendo que U não usa sempre o mesmo computador.
 - Sabendo que U usa sempre o mesmo computador.
11. A transferência de dados binários codificados em bytes (com 8 bits significativos) em mensagens contendo apenas caracteres US-ASCII, pode realizar-se codificando o valor de cada conjunto de bits através de caracteres alfanuméricos do código ASCII. Por exemplo, utilizando a codificação em hexadecimal. Esta poderia ser a forma de transmitir um nome de utilizador e a sua palavra passe no *header field* HTTP **Authorization**.
A codificação em hexadecimal é na base 16 (4 bits por símbolo) e representa cada grupo de 4 bits num dos seguintes caracteres: "0".."9", "A", "B", "C", "D", "E", "F".
O método designado Base64 baseia-se em tomar cada grupo de 24 bits (3 bytes), dividi-lo em grupos de 6 bits, e codificar cada um dos grupos de 6 bits num dos caracteres correspondentes da seguinte tabela ASCII (indexada de 0 a 63): "A".."Z", "a".."z", "0".."9", "+"e "/" , isto é, uma tabela com exactamente 64 caracteres distintos.
Porque razão este método foi adoptado pelo HTTP ao invés da codificação em hexadecimal?
12. Um cliente HTTP acede a uma página HTML num servidor. Depois de obter essa página, o cliente deduz que a mesma tem 6 imagens e que as mesmas devem ser obtidas igualmente, a partir do mesmo servidor, para mostrar o conteúdo ao utilizador.
O tempo de trânsito ida e volta (RTT) entre o cliente e o servidor é de (cerca de) 100 ms. O cliente não tem nenhuma conexão aberta para o servidor antes de começar a aceder à página, mas já conhece o endereço IP do servidor. O tempo necessário para transmitir os pacotes com o ou os comandos, a página ou as imagens são negligenciáveis, e cada mensagem HTTP (pedido ou resposta) é transmitida de uma vez dado caber sempre num só segmento TCP.
Qual o menor tempo necessário para o cliente obter a página e as imagens usando o protocolo HTTP 1.1 com *pipelining* sabendo que o cliente está parametrizado para abrir no máximo 4 conexões em paralelo para cada servidor?
13. Você é o arquitecto de um aplicação Web especial que exige que os seus clientes, para obterem o serviço, obtenham vários ficheiros de pequena dimensão. Todos os ficheiros são estáticos (isto é nunca são alterados) excepto um que tem de ser gerado por um programa que leva 10 segundos a gerá-lo, sempre que este é necessário para transmitir. Todos os ficheiros são servidos pelo mesmo servidor. Qual das seguintes soluções é a que assegura que a obtenção da totalidade dos ficheiros é a mais rápida?
- O cliente usa várias conexões HTTP 1.0 em sequência, *i.e.*, uma de cada vez.

- (b) O cliente usa uma única conexão TCP e o protocolo HTTP 1.1 com suporte de *pipelining*. O cliente só pede o ficheiro que tem de ser gerado no fim.
- (c) O cliente usa várias conexões HTTP 1.0 em paralelo.
- (d) O cliente usa uma única conexão TCP e o protocolo HTTP 1.1 com suporte de *pipelining*. O cliente começa logo por pedir o ficheiro que tem de ser gerado para ir adiantando trabalho.
- (e) O cliente usa uma conexão TCP e o protocolo HTTP 1.1 com suporte de *pipelining* para pedir todos os ficheiros menos o que tem de ser gerado. Utiliza em paralelo uma outra conexão para obter o ficheiro que tem de ser gerado e começa logo por pedir esse ficheiro.
14. Um servidor acessível via o serviço HTTP distribui segmentos de filmes com a dimensão, sempre constante, de 256×1400 bytes ≈ 360.000 bytes. O servidor aceita pedidos em HTTP 1.0 ou em HTTP 1.1. Em qualquer das hipóteses responde com uma mensagem contendo as seguintes *header fields* na resposta: *Accept-Ranges: none* e *Connection: close*.
 Todas as conexões TCP do servidor usam um MSS (*Maximum Segment Size*) de 1460 bytes pelo que pode considerar que em cada segmento TCP se transferem geralmente 1400 bytes de filme. As transferências nunca estão limitadas pelo débito de nenhum canal na rede pois os mesmos são todos de muito alto débito. A dimensão máxima dos *buffers* do TCP é sempre de 512 K bytes no cliente e no servidor. As conexões começam sempre a transmitir dados com o valor da *ConWnd* (*Congestion Window*) igual a um segmento.
- (a) Quanto tempo leva um cliente a obter cada um dos segmentos de filme usando o protocolo HTTP 1.0, sem conexões paralelas, sabendo que o RTT do cliente ao servidor é de (cerca de) 100 ms?
- (b) Idem nas condições da a) mas usando o protocolo HTTP 1.1?
15. Escreva um programa na sua linguagem de programação preferida, que recebe um URL em parâmetro e escreve no disco local o objecto que é obtido executando a operação GET sobre o URL dado. O programa deve também mostrar os *header fields* da resposta do servidor.
16. Desenvolva um servidor HTTP elementar na sua linguagem de programação preferida. O servidor só suporta o comando GET e serve ficheiros locais. Defina também com cuidado, o subconjunto de *header fields* suportados pelo seu programa.

Capítulo 13

Redes de distribuição de conteúdos

A priori, *finding similar laws governing Web page popularity, city populations, and gene copies is quite mysterious, but if one views all these as outcomes of processes exhibiting rich-get-richer effects, then the picture starts to become clearer.*

– Autores: *David Easley and John Kleinberg*

Inicialmente os utilizadores da Internet eram uma fracção diminuta da população mundial, confinada essencialmente a investigadores em países desenvolvidos. Com o aparecimento da Web, os utilizadores foram-se alargando, e o correio electrónico e o acesso a páginas Web e a conteúdos digitais tornaram-se as aplicações dominantes até ao final dos anos de 1990.

Por essa altura o meio privilegiado para acesso à Internet pelas pessoas comuns era baseado em canais suportados no sistema telefónico, com débitos máximos inferiores a 64 Kbps. Transferir um ficheiro de 1 Mbyte por estes canais levava mais de 2 minutos. Transferir uma canção com a duração de 3 minutos, codificada em MP3 num ficheiro com 4 Mbytes, levava pelo menos 8 minutos. Se o débito extremo a extremo fosse menor que o dos canais de acesso, *i.e.*, se as redes de trânsito estivessem congestionadas ou saturadas, os tempos necessários para acesso a páginas Web e outros recursos digitais volumosos tornavam-se penosos para os utilizadores. No início do Século XXI dizia-se, por brincadeira, que WWW era a abreviatura de *World Wide Wait*.

Na mesma época muitas empresas e universidades tinham ligações à Internet a velocidades bastante superiores (*e.g.*, alguns Mbps), que eram partilhadas pelos numerosos utilizadores das suas redes institucionais. Essas redes locais funcionavam internamente com débitos muito maiores (*e.g.*, 100 Mbps) e rentabilizar a capacidade de acesso externo à Internet tornava-se essencial. Era então popular usarem-se servidores auxiliares de acesso à Web, chamados *proxies* Web institucionais, que tentavam rentabilizar as conexões institucionais externas fazendo *caching* de páginas. Explicar o que são e como funcionam esses servidores é o primeiro objectivo deste capítulo.

A partir do ano 2000, as expectativas de explosão de negócios suportados na Web, levou a um grande investimento em infra-estruturas de comunicação e ao aparecimento de redes de acesso que permitiam débitos maiores para os utilizadores residenciais. O termo *banda larga* generalizou-se e foi aplicado inicialmente a ligações à Internet com débitos superiores a algumas centenas de Kbps. O acesso com débitos superiores, a descida de preços e uma maior generalização dos acessos, levou à explosão da oferta de conteúdos digitais na Web, e aparecerem os primeiros serviços com centenas de milhar

ou mesmo milhões de utilizadores. Entre estes, a pesquisa de conteúdos (*e.g.*, motores de busca) tornou-se um dos mais populares.

Isso criou um problema suplementar, pois um único servidor físico não conseguia satisfazer a procura, e foi necessário encontrar formas escaláveis de fornecer conteúdos a esse grande número de novos utilizadores. As páginas Web populares passaram a ser servidas, não por um servidor, mas por um agregado de servidores (*server cluster* ou *server farm*) capaz de servir de forma eficiente uma procura mais alargada. Explicar como funcionam os agregados de servidores que fornecem serviços simultâneos a muitos milhares de utilizadores é o segundo objectivo deste capítulo.

Quando se generalizou o acesso à Internet com ligações de débito de 1 Mbps ou mais, a transferência da canção codificada em MP3 e contida num ficheiro com 4 Mbytes, passou a ser possível em cerca de 32 segundos ou menos. Conteúdos volumosos, como fotografias com boa resolução, canções, livros, discos inteiros e até filmes, puderam passar a estar acessíveis em servidores Web, o que tornou a sua distribuição potencialmente mais barata, e tornou-os acessíveis a uma grande fracção da população mundial.

Apesar de não se saber neste novo ambiente como realizar o pagamento e a protecção contra cópia de conteúdos digitais sujeitos a direitos de autor (pagos), o facto é que se tornou evidente que uma revolução no mundo da distribuição de conteúdos digitais era inevitável. Inventaram-se então dois tipos de serviços para distribuição de conteúdos: a distribuição por agregados maciços de servidores, e a distribuição cooperativa com base nos interessados no conteúdo, que veio a generalizar-se sob a designação de sistemas P2P (*Peer-To-Peer*).

O contínuo progresso conseguido em técnicas de transmissão de sinal, levou a uma significativa subida do débito dos canais das redes de interligação, a uma grande subida da capacidade das redes de acesso, à generalização do acesso via terminais móveis (*smartphones*), e a uma descida de preços que permitiu a generalização do acesso à Internet em banda larga. No momento em que este livro foi escrito, banda larga é um termo que se aplica a débitos de acesso de várias dezenas de Mbps. Ao mesmo tempo iniciou-se a migração das redes telefónicas, de televisão e de vídeo para a Internet, ou pelo menos para redes baseadas na tecnologia TCP/IP. Neste momento, o débito dos canais utilizados nas redes de trânsito mede-se em dezenas ou centenas de Gbps.

A transmissão de música, vídeo, televisão, fotografias, documentos *etc.* assim como o suporte das comunicações interpessoais, as redes sociais, a colaboração em equipa, as relações com os fornecedores de serviços público e privados, tornaram-se dominantes e levaram a uma escala sem precedentes da utilização da Internet.

A distribuição de conteúdos volumosos, nomeadamente vídeo, é o principal responsável pelo tráfego que circula nas redes de acesso e nas redes de interligação, assim como é o principal motor da utilização dos agregados de servidores que providenciam a sua distribuição. O protocolo HTTP, como base do transporte de conteúdos e execução de aplicações, tornou-se omnipresente. Esta nova realidade tornou obrigatória a utilização de redes de servidores, organizadas em redes lógicas (chamadas redes *overlay*), que são chamadas redes de distribuição de conteúdos. O seu estudo é o objectivo das restantes secções deste capítulo.

13.1 Servidores *proxies* de HTTP

Como vimos no capítulo anterior, diversas componentes dos conteúdos obtidos na Web via HTTP são conteúdos estáticos (*i.e.*, imutáveis) de que os *browsers* Web fazem *ca-*

caching localmente. **HTTP proxies institucionais**¹ são servidores que fazem *caching* colectivo para um conjunto de utilizadores de uma rede de acesso institucional, com o objectivo de optimizar o tráfego HTTP, fazendo *caching* dos objectos que são modificados mais raramente.

O esquema é relativamente simples se os diferentes *browsers* Web dos utilizadores da rede de acesso, ao invés de obterem os objectos directamente dos servidores indicados nos URLs, dirigirem todos os pedidos para um mesmo servidor *proxy*. Desta forma, este pode fazer os pedidos aos servidores finais, obter os objectos das respostas, guardá-los em memória persistente local caso isso seja adequado (ver a seguir), e devolvê-los aos clientes. Sempre que um pedido de um cliente envolver um objecto disponível na *cache*, o *proxy* pode responder directamente ao cliente sem necessidade de contactar o servidor HTTP original. A Figura 12.9 ilustra o posicionamento lógico do servidor *proxy* institucional.

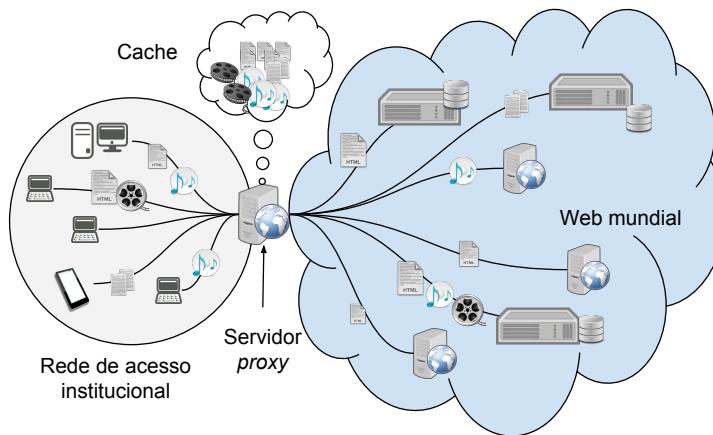


Figura 13.1: Acesso à web via um servidor *proxy* da instituição

A eficácia de um mecanismo de *caching* é medida pelo **rácio de sucesso (hit ratio)**, que é definido como sendo o quociente do total de pedidos servidos directamente a partir da memória *cache* pela totalidade dos pedidos realizados. Esse rácio, expresso geralmente em percentagem, permite-nos avaliar os ganhos potenciais da utilização do *proxy* institucional.

Por exemplo, supondo que em média um objecto pode ser obtido do *proxy* em cerca de 5 milissegundos, e directamente da Web em cerca de 100 milissegundos, então um rácio de sucesso de 30% permitiria concluir que o tempo médio de obtenção de objectos da Web pelos utilizadores da rede de acesso institucional é 100 milissegundos sem utilizar o *proxy*, e $0,30 \times 5 + 0,70 \times 105 = 75$ milissegundos se este for utilizado.

Claro que este resultado foi obtido com um modelo muito simplificado, que deixou escondidos diversos aspectos, nomeadamente como são estimados os tempos médios e que relevância estes têm, assim como o rácio de sucesso da *cache*. No entanto, ele permite, de forma grosseira, ter uma ideia do ganho potencial.

Por outro lado, o modelo também esconde que o tempo para obter uma página é não só dependente do RTT entre clientes e servidores, e da quantidade de bytes que a

¹ *Proxy* ou *Surrogate* poderiam ser traduzidos por *procurador*, *representante*, *substituto*, No entanto, dado não ser essa a tradição no mundo técnico e académico de língua portuguesa, utilizaremos a designação original na língua inglesa: *HTTP proxy*, ou simplesmente *proxy* dependente do contexto.

compõem, mas também do estado de ocupação dos canais que ligam a rede de acesso institucional ao resto da Internet. Se estes canais estiverem pouco ocupados (*e.g.*, carga < 50% da capacidade disponível), não se perdem muitos pacotes e o RTT e o tamanho das página são dominantes no tempo de acesso. No entanto, se o tráfego for tal que esses canais se aproximam da saturação, formam-se filas de espera, perdem-se pacotes, e o tempo para obter um objecto de um servidor Web torna-se instável e cresce significativamente.

A noção de rácio de sucesso também permite raciocinar sobre a capacidade do ou dos canais que devem ligar o *proxy* à Internet. Supondo que no momento de maior carga, o conjunto dos utilizadores necessita de aceder em média a 100 páginas por segundo, cada uma com 100 Kbytes em média, e se pretende que esse débito corresponda em média a 50% do débito máximo desses canais, como dimensionar a ligação à Internet do *proxy*?

Neste cenário a capacidade total necessária para o acesso com canais dedicados entre cada utilizador e cada servidor seria da ordem de grandeza de $100 \times 8 \times 10^5 = 80$ Mbps. No entanto, dado tratar-se de uma rede de pacotes em que por hipótese o *bottleneck link* é o canal que liga o *proxy* à Internet, e se pretende garantir que a carga desse canal não ultrapasse 50%, então teria de se dimensionar o mesmo a $1/0,5 \times 80 \approx 160$ Mbps sem *proxy*. Com *proxy* e uma taxa de sucesso 50%, a capacidade necessária seria apenas de cerca de 80 Mbps.

Apesar de estas contas corresponderem a uma simplificação grosseira, elas permitem ter alguma sensibilidade sobre a eficácia potencial da utilização de um *proxy* se todos os objectos fossem da mesma dimensão e se esta dimensão fosse elevada (*e.g.*, 100 Kbytes). Na prática, os objectos transferidos são de diferentes dimensões e muitos deles são de pequena dimensão.

Transparência dos *proxies* institucionais

Quando um *browser* Web delega num *proxy* o seu acesso via HTTP à Web, ele tem de ser explicitamente parametrizado para esse efeito (o que nem sempre é uma tarefa fácil sobretudo se os utilizadores mudam frequentemente de rede de acesso). Por isso são muitas vezes usados *proxies* especiais.

Este tipo especial de *proxies* estão geralmente integrados com o comutador de pacotes que liga a rede institucional à Internet e, quando detectam o início de uma conexão TCP, redirecionam os segmentos TCP para um módulo software que intercepta a conexão. Se esta corresponder a um pedido HTTP, executa acções equivalentes àquelas que seriam executadas por um servidor *proxy* convencional.

Os *proxies* cujo funcionamento não depende de parametrizações dos *browsers* Web dos utilizadores chamam-se *proxies* transparentes.

Problemas com o uso de *proxies* institucionais

Existem vários aspectos que tornam a utilização de um servidor *proxy* partilhado, como ilustrado acima, não tão eficaz quanto poderia parecer à primeira vista. Alguns destes têm a ver com tráfego HTTP gerado dinamicamente, ou contendo indicações específicas para um utilizador (*e.g.*, *cookies*), e que não podem ser *cached*. Os servidores quando enviam respostas dinâmicas devem tomar todas as precauções para que estas não sejam *cached*. Por exemplo, inserindo o *header field*:

```
cache-control: private, max-age=0, no-cache
```

na resposta.

Um outro aspecto tem a ver com a utilização de conexões seguras entre um *browser* Web e um servidor HTTP. Este tipo de canais são os correspondentes a URLs em que o campo protocolo assume a forma **https**: e que são designadas informalmente

por conexões HTTPS². Por razões de segurança e impossibilidade de o *proxy* servir de intermediário destas conexões, as mesmas são estabelecidas directamente com o servidor HTTP final e não via um *proxy*.

Finalmente, a distribuição da popularidade dos diferentes serviços influencia de forma determinante o rácio de sucesso de um memória *cache*. Geralmente, cada utilizador, quando tomado isoladamente, tem tendência a concentrar os seus pedidos num pequeno número de *sites Web*, pelo que as *caches* dos *browsers Web* podem apresentar um rácio de sucesso elevado. No entanto, na Web em geral, o número de utilizadores é muito elevado e apesar de um certo número de serviços concentrarem muitos pedidos, há uma imensidão de serviços que têm uma utilização que, quando tomada de forma agregada, é muito significativa. Assim, quanto maior é a população que partilha um *proxy* institucional, maior pode ser a diversidade dos acessos e menor a sua eficácia.

Todos estes aspectos concorrem para que a utilização de um *proxy* institucional não seja muito efectiva para melhorar a qualidade do acesso à Web de uma população alargada de utilizadores.

No entanto, em algumas instituições, por razões de controlo de acessos, ou mesmo de proibição do acesso, a certos conteúdos, ou outras razões de política da instituição, o acesso HTTP e HTTPS à rede exterior da instituição só pode ser realizado através de um servidor *proxy* de filtragem e registo de todos os acesso realizados pelos utilizadores (em *logs* de acessos).

Com efeito, a noção de servidor *proxy* é muito geral e tem muitas outras utilizações. Por exemplo, para ultrapassar certas barreiras políticas, ou para garantir anonimato, existem servidores públicos na Internet que aceitam intermediar o acesso a certos conteúdos, por boas (ultrapassar barreiras de censura política ou garantir anonimato) ou más razões (esconder práticas ilegais).

Proxy servers (servidores procuradores) são servidores que actuam como representantes de outros servidores. Os *proxies* são utilizados para melhorar ou modificar, através de intermediação, o acesso aos serviços que representam.

No caso do acesso à Web, podem ser usados *proxies* institucionais que actuam como intermediários entre os utilizadores de uma rede de acesso, e os servidores Web existentes na Internet. O seu papel é acelerarem o acesso através da realização de *caching* de objectos que são actualizados mais lentamente. Estes servidores podem desempenhar este papel porque os acessos dos clientes HTTP da rede de acesso são redirigidos para os mesmos.

A eficácia de um mecanismo de *caching* é medida pelo chamado **rácio de sucesso (hit ratio)**, que é definido como sendo o quociente do total de pedidos servidos directamente da memória *cache*, pela totalidade dos pedidos realizados. No caso do acesso à Web, o aumento da percentagem de objectos dinâmicos e a adopção cada vez mais frequente de conexões HTTP cifradas (HTTPS), entre outros aspectos, têm diminuído a eficácia real dos *proxies* institucionais.

13.2 Distribuição de carga de acessos HTTP

Quando o número de utilizadores de um serviço acessível via a Web aumenta, são tomadas medidas para evitar que os utilizadores sejam servidos cada vez mais lenta-

² Como já foi referido, estas conexões são estabelecidas entre um *browser Web* e o servidor usando técnicas de criptografia em complemento do estabelecimento da conexão TCP convencional.

mente. Para esse efeito pode-se tentar melhorar as características do computador físico que suporta o servidor HTTP do serviço (*e.g.*, aumentando a memória, melhorando os processadores, as interfaces e a memória não volátil). No entanto, a melhoria de qualidade assim conseguido é limitado. A alternativa consiste em aumentar o número de servidores que prestam o serviço.

Para evitar que os utilizadores tenham que memorizar diversos URLs do serviço, o que até se tornaria impraticável pois muitas páginas têm URLs embutidos, é necessário encontrar uma forma de distribuir os diferentes pedidos HTTP por diferentes servidores. Isso é possível de diversas maneiras.

A solução mais simples passa por replicar o servidor HTTP em vários servidores idênticos e distribuir os pedidos pelas diferentes réplicas.

Distribuição de carga com vários endereços e distribuidores de carga

O DNS permite associar mais do que um endereço IP ao mesmo nome *n*, através de RRs (*resource records*) A ou AAAA associados a *n*, como foi introduzido no Capítulo 11. Por exemplo:

```
exemplo_popular.pt      10      IN      A 10.100.100.1
exemplo_popular.pt      10      IN      A 10.100.100.2
exemplo_popular.pt      10      IN      A 10.100.100.3
```

Sempre que um servidor recebe o pedido [name: exemplo_popular.pt, type: A], devolve as linhas acima com os três endereços IP, mas de cada vez vai rodando a ordem pela qual envia os 3 registos. Se o *browser* Web usar a primeira resposta que recebe, de cada vez usará um servidor diferente. A Figura 13.2 mostra um exemplo em que o serviço é acessível através de três servidores distintos, cada um com um endereço IP distinto.

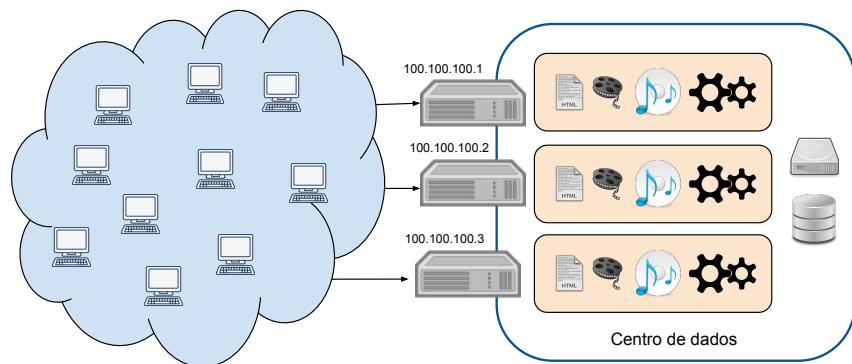


Figura 13.2: Distribuição de carga usando vários servidores com endereços IP distintos

Este método exige que o servidor usado seja indiferente. Para tal os servidores têm de conter uma cópia idêntica de todos os objectos e o seu estado deve ser completamente idêntico, mesmo quando este é modificado. Caso existam bases de dados que suportem o cálculo de respostas dinâmicas (*e.g.*, *cookies*) é também necessário continuar a assegurar que o servidor usado pelo cliente seja indiferente. Nestes casos é habitual usar sistemas mais complexos. Por exemplo, os objectos estáticos ou que evoluem lentamente podem estar replicados em cada servidor, mas para o estado dinâmico recorre-se a um servidor de base de dados partilhado entre os diferentes servidores HTTP.

O método ilustrado implementa uma primeira solução mas a mesma tem algumas insuficiências. Por um lado não é claro que assegure que todos os servidores tenham cargas equilibradas. Isso só poderia ser melhor aproximado caso todos os clientes usassem o mesmo DNS *caching only server* e todos os pedidos fossem, grosso modo, equivalentes do ponto de vista dos recursos exigidos para os satisfazer. Também, para obviar o facto de que alguns destes servidores não rodam as respostas DNS, usam-se TTLs muito baixos (10 segundos no exemplo acima) o que diminui a eficiência do *caching* no sistema DNS. Por outro lado, se um dos servidores tiver uma avaria, até o DNS ser actualizado continuam a ser enviado pedidos para o mesmo.

Para obviar estes defeitos pode-se adoptar uma solução baseada num servidor especial, com um único endereço IP, portanto independente do DNS, chamado um distribuidor de carga (*load balancer*). Estes servidores podem receber indicações sobre a carga recebida por cada servidor do serviço e irem distribuindo as conexões HTTP de forma a equilibriarem a mesma. Por outro lado, caso um servidor tenha uma avaria, o distribuidor deixará de enviar conexões para o mesmo. A Figura 13.3 ilustra este novo tipo de configuração.

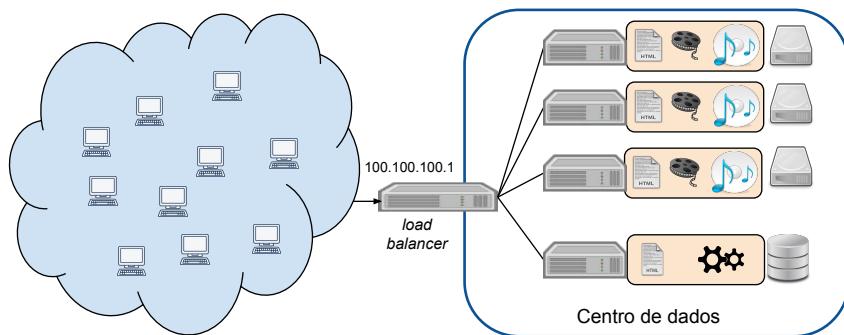


Figura 13.3: Distribuição de carga com um único endereço e um *load balancer*

Esta solução pode ser realizada com diversos níveis de sofisticação. No caso mais simples o *load balancer* recebe todos os pacotes e assegura apenas que as diferentes conexões TCP são distribuídas entre os servidores para equilibrar a carga, mas cada uma delas termina sempre no mesmo servidor. Este tipo de distribuidor apenas necessita de analisar os pacotes até aos níveis IP e TCP. No entanto, o distribuidor pode ir mais longe e analisar também o cabeçalho HTTP de cada pedido e, em função do objecto pedido, distribuir o pedido por servidores especializados em servirem conteúdos estáticos e servidores especializados em servirem conteúdos dinâmicos.

Distribuição de carga por geo-localização

Os métodos de distribuição de carga ilustrados acima funcionam bem se todos os servidores do serviço estiverem no mesmo local, isto é, no mesmo centro de dados. Caso o serviço seja muito popular e com clientes espalhados pelo mundo inteiro, importa diminuir, tanto quanto possível, o RTT entre os clientes e os servidores. Neste caso é necessário distribuir os servidores por centros de dados situados em diversos países e continentes e diz-se então que o serviço está replicado geograficamente ou geo-replicado.

Idealmente, cada cliente deve ser servido pelo servidor que estiver mais “próximo”. Como esses servidores têm necessariamente endereços IP distintos, não se pode usar a solução DNS em que o DNS responde com o conjunto dos endereços. Idealmente, o

DNS deverá ser mais inteligente e responder a cada cliente com apenas um endereço IP, de preferência o do servidor ou do *load balancer* que estiver mais próximo do cliente. Ora isso exigiria que o servidor DNS conhecesse qual o RTT entre cada cliente e cada servidor, o que não se afigura muito simples.

É no entanto possível tentar obter uma aproximação dessa distância se for possível saber quais as coordenadas geográficas dos clientes e dos servidores. Com efeito, a distância geográfica entre dois computadores na rede pode ser tomada como uma primeira aproximação, provavelmente grosseira, do RTT entre os mesmos. Mesmo que essa relação seja muito grosseira, a mesma deve ser suficiente para evitar que um cliente em Lisboa seja enviado para um servidor em Los Angeles quando existe um em Paris.

Veremos a seguir que é possível relacionar endereços IP com a sua localização geográfica, e admitindo que essa informação é conhecida, é possível construir um servidor DNS que, quando tem diversos endereços IP associados ao mesmo nome, responde com o endereço IP que está mais próximo geograficamente do endereço IP do cliente que enviou a interrogação ao servidor DNS. Desta forma, o servidor DNS responde com o endereço IP do servidor HTTP que provavelmente terá o menor RTT para o cliente. A Figura 13.4 ilustra esta nova configuração.

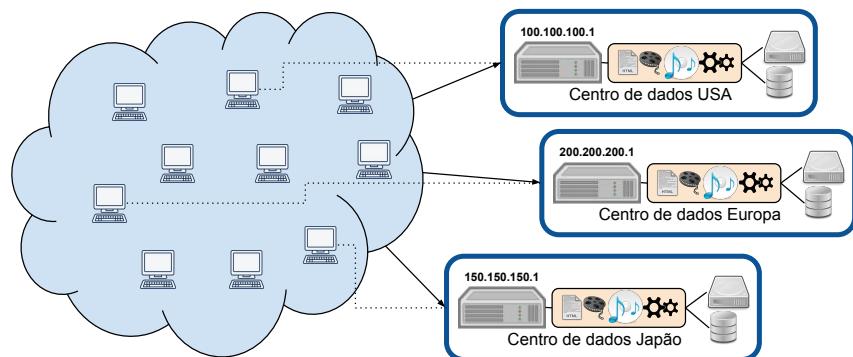


Figura 13.4: Distribuição de carga com diferentes centros de dados em diferentes regiões geográficas

Geo-localização de endereços IP

Durante alguns anos tentaram-se definir métodos de localização dos endereços dos computadores da Internet num espaço de coordenadas virtual, de tal forma que a distância entre essas coordenadas pudesse ser mapeada no RTT entre os endereços [Donnet et al., 2010]. Apesar de ter havido algum progresso, esses trabalhos não conduziram a nenhum serviço público que possa ser consultado para estimar dinamicamente o RTT entre quaisquer dois endereços IP arbitrários.

No entanto, existem diversos serviços públicos, ou a pagar, que disponibilizam o mapeamento de um endereço IP na sua localização geográfica aproximada. Esses serviços têm muito valor pois permitem saber a partir de que país é realizado cada acesso, o que pode ser importante para efeitos de segurança, ou para simples localização de clientes e afinação dos serviços propostos, seleção da língua usada no diálogo, etc.

De facto existem diversas formas que permitem associar os endereços IP à localização (país, cidade) dos mesmos, ou até mesmo às suas coordenadas GPS aproximadas. Com efeito, os endereços IP são afectados por blocos aos ISPs ou às instituições que deles necessitam, ver a Secção 17.1. Ora essa afectação é realizada pelas chamadas

Regional Internet Registries que mantêm bases de dados públicas das afectações que realizam e onde figura a morada da sede das instituições a quem a afectação foi feita. Estas bases de dados permitem uma geo-localização que, em função da dimensão do bloco de endereços e do âmbito geográfico em que os mesmos podem ser usados, pode ter uma maior ou menor exactidão.

No entanto, dado o valor intrínseco da geo-localização, diversas instituições com infra-estruturas de âmbito mundial utilizam outras informações complementares, que permitem também mapear a localização geográfica dos endereços IP, algumas vezes com grande exactidão. Finalmente, estes meios são complementados com análise de dados realizada por prestadores de serviços que pedem a morada aos clientes, serviços que pedem a localização aos clientes, serviços que obtém a localização dos clientes a partir das aplicações que correm nos seus telemóveis, *etc.* Através da utilização intensiva de análise de dados e cruzando informação de diversos clientes, obtida por diversos meios, é possível construir bases de dados com muito maior exactidão.

Com base nestas bases de dados, designadas *IP Geolocation Database Services*, os servidores DNS podem ser parametrizados para as usarem como acima foi descrito.

Replicação dos serviços

De qualquer forma, e como já foi referido, a distribuição de um serviço por vários servidores HTTP, exige que os servidores estejam sincronizados e respondam da mesma forma, seja qual for o servidor usado pelo cliente. Isso exige que os servidores conteham sempre o mesmo **estado geo-replicado**.

Manter cópias de estado sincronizado é relativamente fácil quando esse estado é estático (*e.g.*, conjuntos de fotografias, vídeos, *etc.*) ou evolui muito lentamente (*e.g.*, previsão do estado do tempo para as próximas 24 horas). O DNS é um exemplo paradigmático de uma aplicação distribuída com replicação do estado, em que este evolui lentamente. As técnicas usadas pelo DNS são um bom exemplo de algumas das soluções que podem ser adoptadas para realizar essa replicação em contextos similares.

No entanto, quando a aplicação envolve estado que evolui com frequência (*e.g.*, o carrinho de compras de um comprador, o sistema de autenticação e facturação dos utilizadores de um sistema de distribuição de filmes, *etc.*), manter a sincronização permanente do estado da aplicação entre vários servidores de forma a tornar indiferente qual deles é usado, é bastante mais complexo de implementar e tem conduzido a estudos sofisticados e desenvolvimentos inovadores [Vogels, 2009].

A seguir vamos apenas exemplificar uma das soluções que tem sido adoptada com alguma frequência nos cenários mais simples, nomeadamente serviços de distribuição de conteúdos estáticos populares (*e.g.*, vídeos, fotografias, filmes, *etc.*).

Distribuição de carga com recurso a *proxies* inversos

Dado que a geo-replicação se pode revelar bastante complexa em função da taxa de actualização dos objectos, e da repercussão mais ou menos grave de ausência de sincronização, muitos serviços Web geo-replicados adoptaram uma configuração mista, especialmente adequada à distribuição de vídeo, música e fotografias, que descrevemos em seguida.

Ao invés de se colocar uma réplica integral do serviço em cada centro de dados, cada um destes centros possui um (ou mais) servidor *proxy* especial designado **proxy inverso (reverse proxy)**. Os clientes são dirigidos para o *reverse proxy* mais próximo sem consciência de estarem a estabelecer uma conexão com um *proxy*. Com esse objectivo é usado um servidor de DNS especial que utiliza geo-localização para seleccionar o *proxy* mais próximo do cliente. Com esta solução, as conexões HTTP e HTTPS dos clientes terminam necessariamente nestes *proxies* dedicados ao serviço.

Cada *reverse proxy* conhece o conjunto de servidores centrais que têm a cópia oficial dos objectos do serviço, e quando recebem um pedido, dirigem-o para o servidor central, obtém a resposta e devolvem-a ao cliente. No entanto, como em muitos casos a resposta

contém um objecto estático, (vídeo, fotografia, *etc.*), faz *caching* do mesmo localmente como faria um *proxy* institucional. Desta forma vai-se convergindo para uma situação em que a grande maioria dos objectos de maior dimensão, geralmente estáticos, são servidos a partir das *caches* locais. A Figura 13.5 ilustra esta nova configuração.

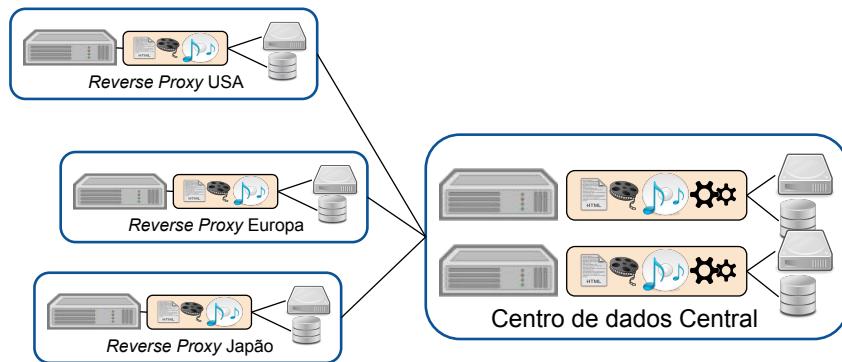


Figura 13.5: Distribuição de carga com recurso a *proxies* inversos em diferentes regiões geográficas e um centro de dados central

Esta solução consiste numa forma pragmática de tentar lidar com as dificuldades levantadas pela geo-replicação. Ela utiliza *caching* para resolver o problema da replicação de objectos que não são actualizados frequentemente, como por exemplo os conteúdos multimédia. No entanto, a parte dinâmica dos serviços, por exemplo a autenticação dos utilizadores, a sua facturação, e a análise das suas preferências, continua a ser gerida centralmente.

Os *proxies* inversos só são possíveis de utilizar quando a sua montagem está intimamente integrada com as aplicações distribuídas que estes suportam. De facto, os clientes quando obtêm o endereço IP de um servidor do serviço, obtêm o endereço do *proxy* que está mais próximo. Por isso, uma rede de acesso não pode montar um *proxy* deste tipo para servir o conjunto das aplicações em que os seus clientes estão interessados.

A arquitectura que acabámos de descrever é um exemplo particular e simples de um conceito chamado rede de distribuição de conteúdos (*CDN - Content Distribution Network*) que é o objecto da próxima secção.

As aplicações acessíveis via a Web com grande número de clientes, utilizam servidores replicados para fornecerem o serviço, com distribuição (da carga) dos clientes pelos diferentes servidores. Quando o conjunto dos clientes é muito grande, e estes estão espalhados por diferentes continentes, os servidores são geo-replicados para garantir uma melhor qualidade de serviço aos clientes, nomeadamente minimizando o RTT.

Geo-replicar um conjunto de serviços pode revelar-se complexo. Por isso foram desenvolvidas soluções baseadas em ***proxies inversos***, que permitem replicar de forma inteligente os conteúdos estáticos, ao mesmo tempo que se mantém a parte dinâmica da aplicação potencialmente centralizada.

13.3 CDNs com infra-estrutura dedicada

Muitos serviços acessíveis via a Web, com especial realce para os que necessitam de enviar aos utilizadores páginas muito complexas, com dezenas e dezenas de objectos, muitos dos quais estáticos (fotografias de produtos, ficheiro com descrições, *etc.*), têm páginas muito pesadas de obter sempre que o RTT é elevado.

Por exemplo, se o RTT for da ordem de grandeza de 100 ms, o RTT típico entre a Europa e a América do Norte, uma página com 100 objectos levaria pelo menos 10 segundos para ser obtida com HTTP 1.0. Mesmo com várias conexões paralelas, o utilizador teria de esperar alguns segundos para obter toda a página. No entanto, se o RTT fosse 10 ms, tudo se passaria muito rapidamente fosse qual fosse a solução de acesso usada. Isto é tanto mais verdadeiro, quanto mais pequenos forem os objectos e maior o seu número.

Quanto menor for o RTT, mais rápida é a subida da dimensão da janela do TCP e mais rápida é a recuperação quando se perdem pacotes. Pelos diversos motivos, quanto menor for o RTT, melhor é potencialmente a qualidade do serviço fornecido aos clientes.

Caso o tipo de conteúdo a enviar aos utilizadores seja muito volumoso, ou o número de utilizadores seja muito elevado, existem ganhos significativos se se usarem vários servidores próximos dos clientes pois desta forma cada conteúdo é servido mais depressa. Adicionalmente, usando vários servidores distribuídos é mais fácil absorver picos de carga nas horas de ponta, diminuindo assim o tráfego e a carga dos servidores originais dos conteúdos.

Também, se esses servidores forem partilhados entre vários serviços, quando há situações de procura anormal de um dado conteúdo (na sequência de eventos especiais como desastres ou notícias de última hora, saída de uma nova série de televisão, ou de uma nova versão de um sistema operativo, *etc.*) dá-se uma concentração de acessos (*flash crowd*). Como nem todos os serviços estão sujeitos a estes eventos anormais ao mesmo tempo, o conjunto dos servidores absorve de forma mais rentável essas procuras anormais.

Finalmente, se se realizar a monitorização constante da disponibilidade dos servidores de conteúdos que existem junto dos clientes, é possível dirigir os utilizadores apenas para os servidores operacionais e desta forma mascarar as falhas dos servidores inoperacionais.

Este conjunto de razões levou ao desenvolvimento das **redes de distribuição de conteúdos** (*CDN - Content Distribution Networks*), ver a Figura 13.6. Uma CDN é um conjunto de computadores coordenados entre si com o objectivo de providenciar de forma mais eficiente e conveniente um serviço de distribuição de conteúdos na Internet. Uma CDN pode ser baseada numa infra-estrutura dedicada ou na colaboração entre os próprios clientes dos serviços. Nesta secção vamos estudar o primeiro caso e na secção seguinte o segundo.

Os principais utilizadores de CDNs com infra-estrutura dedicada são as indústrias dos conteúdos e de difusão de multimédia (*e.g.*, jornais, vídeos, filmes, TV, música, *etc.*), os portais Web, os serviços de venda de produtos *on-line*, as redes sociais, *etc.*

Estas CDNs consistem num conjunto de *proxies* inversos, e numa rede de distribuição de conteúdos, optimizada para comunicação entre os *proxies* e outros servidores dedicados. Ambas as componentes têm como objectivo a melhoria da qualidade de serviço oferecido aos clientes finais. As CDNs com infra-estrutura dedicada fornecem igualmente serviços de valor acrescentado como por exemplo proteção contra falhas de funcionamento e, eventualmente, serviços de segurança (protecção contra *DDoS attacks*, ver a Secção 4.2).

Devido aos elevados custos operacionais de uma CDN são raros os serviços com CDNs dedicadas, apesar de existirem alguns (*e.g.*, serviço YouTube), mas a maioria das CDNs são partilhadas entre muitos clientes da CDN. A empresa Akamai

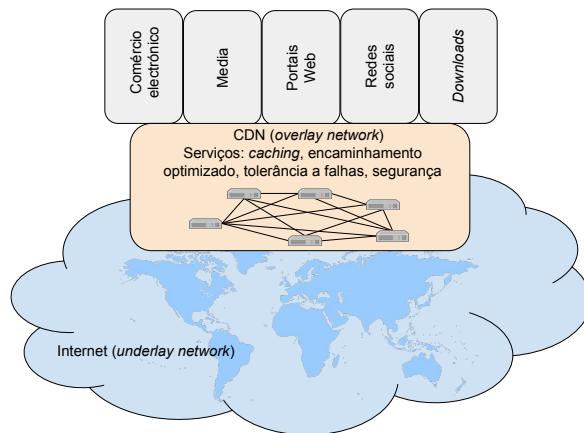


Figura 13.6: Posicionamento das CDNs entre a Internet e os serviços oferecidos aos clientes

(<https://akamai.com>) distingue-se por ter sido a primeira a oferecer este tipo de serviços através de uma CDN partilhada.

Estrutura interna e funcionamento da CDN

A CDN é constituída por um conjunto de servidores da CDN a que os clientes finais, *i.e.*, os utilizadores, se ligam directamente para a obtenção dos conteúdos servidos pela CDN. Por conveniência e analogia chamemos-lhe *proxies* inversos ou simplesmente *proxies* da CDN. A distribuição geográfica destes *proxies* determina a adequação da CDN a servir um dado conjunto de clientes. Uma CDN sem *proxies* na Europa não é adequada a servir utilizadores europeus.

Para além dos *proxies*, a CDN tem um serviço de monitorização. Este serviço mede periodicamente o débito extremo a extremo entre os diferentes *proxies*, e destes com os servidores dos seus clientes³, mede a popularidade dos serviços por regiões e mede o débito extremo a extremo entre os utilizadores e os *proxies* a que estes se ligam. Adicionalmente, uma CDN com milhares de *proxies* inversos, espalhados pelas principais redes de acesso mundiais, tem a possibilidade de ajudar a incrementar o rigor do mapeamento geográfico dos endereços IP dos utilizadores finais.

Por exemplo, a Akamai dispõe, aquando da escrita deste livro, de cerca de 150.000 servidores, espalhados por cerca de 1.500 redes de acesso (ISPs), em mais de 100 países diferentes. Cada ponto de presença da CDN (POP-CDN), contém várias dezenas de servidores (geralmente designados por *edge servers*) acessíveis por distribuidores de carga, está próximo de uma ou mais redes de acesso (por exemplo directamente ligada à infra-estrutura de um ISP) e dispõe de muitos *proxies* para atender simultaneamente milhares e milhares de clientes. Cada um desses aglomerados de servidores, para além da distribuição dos conteúdos, tem servidores de distribuição de carga e servidores de monitorização, sendo capaz de se auto reconfigurar para sobreviver a falhas [Nygren et al., 2010].

Os servidores de cada POP-CDN são complementados por um certo número de servidores centrais de controlo da CDN. Nomeadamente o serviço de colecta das estatísticas de monitorização, o qual permite a execução de algoritmos necessários à

³Estas medidas são realizadas medindo o tempo necessário para transferir entre os extremos ficheiros com alguns Kbytes.

operação e à optimização do funcionamento da CDN, os servidores de DNS especiais da CDN, etc. A Figura 13.7 ilustra a estrutura interna de uma CDN.

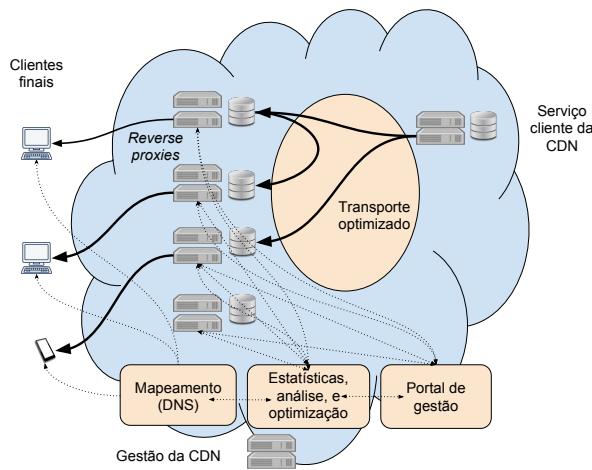


Figura 13.7: Estrutura interna de uma CDN com os *proxies* inversos em diferentes regiões geográficas e os servidores centrais de controlo e gestão. As linhas a negrito representam os fluxos de dados. As linhas a tracejado os fluxos de controlo. Adaptado de [Nygren et al., 2010].

Quando um utilizador final acede a um URL de um dos serviços fornecido com apoio da CDN, o serviço de DNS da CDN, através do endereço IP do cliente, determina qual o melhor *proxy* a que este se deve ligar, e devolve o endereço IP desse POP-CDN e não o endereço IP dos servidores do cliente da CDN. Assim, quando o cliente final solicita objectos a esse serviço, solicita-os ao *proxy* para que foi dirigido.

A escolha do *proxy* que serve os pedidos de um cliente final toma em consideração o endereço desse cliente, o estado da rede para minorar o RTT e aumentar o débito extremo a extremo, e evita os *proxies* inoperacionais. Com efeito, para além de utilizarem a já referida geo-localização dos endereços IP, algumas CDNs, devido à sua presença alargada junto de redes de acesso, utilizam também as estatísticas sobre o tempo de transferência extremo a extremo entre os clientes e os seus diferentes POP-CDNs. Uma outra técnica consiste em usar *anycasting*, como já foi referido a propósito dos servidores de *root* do DNS, ver a Secção 11.3, e que será melhor explicada na Secção 18.3.

A instalação de um POP-CDN junto a uma rede de acesso, constitui uma benção para o operador da rede de acesso (ISP), pois o tráfego daquele serviço para os clientes do ISP, passa a ser local ao invés de externo (internacional ou mesmo intercontinental), o qual é mais caro. Por esta razão, no caso das maiores CDNs, os operadores de rede e da CDN chegam geralmente a acordos que levam a que o operador da CDN aloje directamente o POP-CDN dentro das instalações do ISP.

Comunicação interna à CDN

O serviço de comunicação interno à CDN é um serviço optimizado que utiliza vários mecanismos, alguns dos quais são a seguir descritos.

Caching a priori Este mecanismo consiste em transferir previamente para diferentes POP-CDN os conteúdos mais populares (*e.g.*, o próximo episódio de uma série

muito popular). Desta forma, assim que chegarem os pedidos dos clientes finais os objectos já estão disponíveis nos *proxies*.

Transferências inteligentes Quando um cliente final solicita um objecto a um POP-CDN que não dispõe localmente de uma cópia do mesmo, este pode tentar localizar um POP-CDN que já o tenha em *cache*. Isso permite obter uma cópia do objecto a partir do POP-CDN que estiver “mais próximo”. Em alternativa, o cliente final pode ser dirigido para o mesmo via um HTTP “*redirect*”, o que pode ser menos eficiente. O *proxy* transferir o objecto a partir do servidor original do cliente da CDN é a solução de último recurso.

Difusão inteligente de conteúdos Os POP-CDNs podem obter os objectos directamente dos servidores do cliente da CDN. No entanto, o serviço de monitorização do estado da rede permite determinar caminhos optimizados para disseminação dos objectos. Por exemplo, se um dado filme tem de ser colocado num grande conjunto de POP-CDNs, este serviço determina uma forma óptima de os *proxies* irem copiando entre eles o filme desde a sua origem.

Split TCP Os diferentes *proxies* mantêm várias conexões TCP paralelas permanentemente abertas entre si. Essas conexões são partilhadas (multiplexadas) entre diferentes conexões TCP. Desta forma é possível optimizar o caminho seguido, poupar o tempo de estabelecer conexões entre *proxies*, e as suas janelas podem ser mantidas com dimensões razoáveis durante mais tempo (porque evitam constantemente a fase *slow start*).

Janelas TCP não padrão As conexões TCP entre os servidores da CDN, e entre estes e os clientes finais, podem usar janelas TCP iniciais (*congWnd*, ver a Secção 8.3) maiores que as padrão, de forma a optimizar a fase *slow start* da conexão TCP.

Redirecção com base no DNS (*DNS redirection*)

Para ilustrar de que forma uma CDN pode ser partilhada por diferentes clientes, vamos agora analisar um exemplo fictício. A empresa Gulosos Associados fornece receitas culinárias através do URL <http://greedy.info>. Como esta página se tornou muito popular, resolveu contratar os serviços da World Best CDN ([http://bestcdn.com/....](http://bestcdn.com/)) para distribuir as receitas.

A página HTML <http://greedy.info> é servida pelos servidores da Gulosos Associados, contém texto, algumas fotografias e o URL do serviço de pesquisa de receitas (<http://greedy.info/search>). Como as receitas propriamente ditas são conteúdos estáticos, são servidas pela CDN, e portanto os seus URLs têm ser transformados. Assim, o URL dos objectos no serviço original do cliente da CDN são:

```
http://greedy.info/search
http://greedy.info/receitas/salada
http://greedy.info/receitas/bacalhau
http://greedy.info/receitas/coelho
```

mas o URL dos mesmos objectos com a integração na CDN passará a ser:

```
http://greedy.info/search
http://bestcdn.com/greedy.info/receitas/salada
http://bestcdn.com/greedy.info/receitas/bacalhau
http://bestcdn.com/greedy.info/receitas/coelho
```

Quando o DNS é consultado sobre o endereço IP de bestcdn.com, o pedido vai para os servidores DNS da World Best CDN. Estes, a partir do endereço IP do pedido, devolvem o endereço IP do POP-CDN mais próximo do cliente final. A seguir o *browser* Web abre uma conexão e envia, por exemplo, um pedido HTTP da forma:

```
GET http://bestcdn.com/greedy.info/receitas/salada
```

O servidor do POP-CDN que recebe este pedido, analisa-o, verifica se algum dos *proxies* do POP já contém o conteúdo `greedy.info/receitas/salada` e serve-o se o encontrar. Senão, vai obtê-lo, ou pode devolver uma redireção HTTP para um POP-CDN que o contenha.

O endereço IP origem do pedido de resolução do endereço de `bestcdn.com` não é o verdadeiro endereço do *browser* Web do cliente final, mas sim o do servidor de *caching only* que este estiver a utilizar, ver a Secção 11.3. Geralmente este servidor é o do ISP do cliente, e portanto o erro pode não ser muito elevado.

No entanto, com a generalização de servidores DNS *caching only* genéricos, o erro pode ser maior e o cliente ser redirigido para um POP-CDN errado. Por esta razão foi introduzida uma pequena modificação no protocolo do DNS para que os servidores DNS possam conhecer o endereço IP origem de uma consulta recursiva ao DNS [Chen et al., 2015].

Dado que as condições (de carga e disponibilidade) da rede podem ir mudando, o TTL dos registo dos endereços IP dos *proxies* é relativamente baixo (*e.g.*, 10 ou 20 segundos).

Algumas CDNs aceitam providenciar serviços à medida a clientes muito grandes. Por exemplo, poderiam alojar o serviço de pesquisa de receitas em servidores dos seus POP-CDN. Desta forma, a pesquisa também poderia correr junto do cliente, mas esse tipo de serviços são específicos e mais caros e em geral só são disponibilizados pelas CDNs dedicadas como se ilustra a seguir.

CDNs dedicadas hierárquicas

Algumas empresas atingiram um número de clientes tão grande que se justifica montar uma CDN dedicada para os seus serviços. O caso mais conhecido é o da empresa Google. Dados os volumes envolvidos, e a quantidade de clientes espalhados pelos diferentes continentes, essas redes privadas têm geralmente uma configuração hierárquica. Nestes casos, o coração da CDN é constituído por um conjunto de centros de dados de grandes dimensões, espalhados pelo mundo, com uma rede de interligação privada que constitui a espinha dorsal dessa infra-estrutura, ver a Figura 13.8.

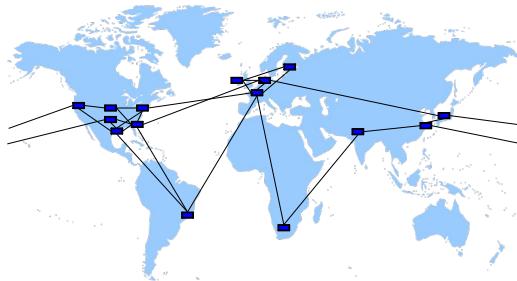


Figura 13.8: CDN hierárquica constituída por um conjunto de centros de dados regionais, complementados com POP-CDNs mais próximos dos clientes

Cada um desses centros de dados centrais tem centenas de milhar de servidores que prestam serviços aos utilizadores da sua área geográfica, mas também se coordenam, se necessário, entre si, de forma a assegurarem total tolerância a falhas, *backups*, reconfiguração de serviços entre continentes se necessário, *etc.*

Os diferentes centros de dados podem depois ser complementados com centenas de POP-CDNs, onde se concentram essencialmente serviços de *caching* e de aceleração local de certas aplicações.

Vários gigantes Internet têm também CDNs baseadas em infra-estruturas dedicadas, também constituídas por diversos centros de dados espalhados pelo mundo. Por exemplo, quer a Amazon, quer a Facebook, têm uma estrutura baseada em centros de dados espalhada pelo mundo. No entanto, no momento da escrita deste capítulo, têm poucas POP-CDNs com *proxies* inversos em complemento desses centros de dados. Isso é particularmente frequente em serviços com grande interactividade, pois nesse caso o ganho introduzido pela utilização de *proxies* inversos é menor.

Um outro exemplo de CDN dedicada é a usada pela empresa Netflix. Neste caso, como o único serviço é o da distribuição de filmes e séries, a Netflix tem uma infra-estrutura central onde gera a autenticação dos utilizadores e as recomendações, e utiliza servidores espalhados pelo mundo para distribuir os vídeos. Neste caso particular, a Netflix utiliza servidores alugados em empresas de *cloud services*. Trata-se de um outro exemplo de CDN dedicada, mas baseada em infra-estruturas alugadas a terceiros.

CDNs e serviços de segurança

Como já foi referido na Secção 4.2, o desenho inicial da arquitectura da Internet menosprezou a necessidade de mecanismos de segurança, e os sistemas de operação e muito do software com estes fornecido, foram também inicialmente desenhados antes da expansão que a Internet hoje conhece, e da explosão do seu valor social, económico e político. Por isso os servidores dos serviços acessíveis na Internet são sujeitos a ataques de negação de serviço (*DDoS attacks*) e a tentativas de intrusão, mas nem sempre estão bem preparados para lhes resistir.

Não cabe aqui discutir a forma como os servidores devem ser preparados para resistirem a esses ataques, mas é fácil de perceber que isso acaba por ser bastante dispendioso, dado que é necessário recorrer a peritos em segurança e fazer constantes actualizações do software sempre que são descobertas novas fragilidades. Ora quanto mais valioso for o serviço prestado, mais os seus servidores estão sujeitos a ataques de todo o tipo.

As CDNs baseadas em infra-estrutura têm um grande número de servidores expostos na Internet (e.g., *proxies* inversos) mas estes são cópias de meia dúzia de configurações de base. Manter actualizado e protegido o seu software é caro, mas esse custo é amortizado facilmente, pois vários milhares de servidores partilham o mesmo software e as respectivas configurações. Adicionalmente, esses servidores têm mesmo de estar preparados para ataques porque servem milhões de utilizadores.

Acontece que quando a resposta aos pedidos HTTP é realizada pelos servidores da CDN, os servidores do serviço original não estão expostos aos potenciais atacantes. Adicionalmente, se for possível redirigir todo o tráfego dirigido aos servidores do serviço para os servidores da CDN, os servidores do serviço ficam completamente escondidos por detrás dos servidores da CDN.

Analisemos então de novo o exemplo anterior mas num quadro em que a empresa Gulosos Associados contrata apenas a proteção dos seus servidores pela CDN. Neste quadro a CDN passa também a gerir os servidores DNS do domínio `greedy.info`, e resolve este nome de tal forma que os endereços IP devolvidos são os dos seus *proxies* e estes passam a fazer de intermediários para os servidores da Gulosos Associados.

Assim o URL dos objectos do serviço da empresa Gulosos Associados, que são recebidos pelos servidores de `http://bestcdn.com` são:

```
http://greedy.info/search
http://greedy.info/receitas/salada
http://greedy.info/receitas/bacalhau
http://greedy.info/receitas/coelho
```

mas o URL dos mesmos objectos, só conhecidos pelos servidores da CDN são:

```
http://gulosos.com/search
```

```
http://gulosos.com/receitas/salada
http://gulosos.com/receitas/bacalhau
http://gulosos.com/receitas/coelho
```

Adicionalmente, o servidor no endereço `gulosos.com` pode estar parametrizado para só aceitar pedidos dos servidores da CDN, e desta forma fica completamente protegido dos atacantes, pois é integralmente representado pelos *proxies* da mesma. Os servidores da CDN, recebem os pedidos, sujeitam-nos a análise, rejeitam os identificados como sendo de atacantes e redirigem os restantes. A Figura 13.9 mostra a configuração do serviço neste caso.

Se a Gulosos Associados apenas contratar serviços de proteção, os servidores da CDN apenas actuam como filtro e redirecção dos pedidos. No entanto, em muitos casos poderiam também fazer *caching* das respostas. Isso mostra que existe uma grande latitude de opções. Num extremo, a CDN só fornece serviços de DNS e de filtragem e redirecção de todos os pedidos legais. No outro extremo, a CDN fornece o serviço de DNS, faz *caching* dos objectos estáticos, e até poderia alojar em cada POP-CDN uma réplica da aplicação de pesquisa.

Uma outra solução possível para dirigir os pedidos dos clientes finais para os POP-CDNs pode consistir na utilização de IP Anycasting, ver a Secção 18.3.

Uma CDN baseada numa infra-estrutura dedicada é uma rede lógica justaposta (*overlay network*) de servidores com o objectivo de providenciar de forma mais eficiente e conveniente um serviço acessível via a Internet. Os servidores coordenam-se entre si pela Internet, que funciona como uma **rede de suporte (*underlay network*)**.

Esta infra-estrutura dispõe de servidores espalhados por toda a Internet, e recorre a servidores DNS com geo-localização ou outras técnicas (*e.g., anycasting*), a *proxies* inversos, monitorização e algoritmos de optimização, assim como a filtragem e redirecção de pedidos.

Estas CDNs podem ser dedicadas ou partilhadas entre muitos serviços e fornecedores, hierárquicas, *i.e.*, complementando os *proxies* com centros de dados regionais, e podem fornecer vários serviços de valor acrescentado como por exemplo: absorção de carga, aceleração de aplicações, proteção contra falhas de funcionamento, segurança e proteção contra ataques.

13.4 Distribuição P2P de conteúdos

Entre o final dos anos de 1990 e o início do século seguinte generalizou-se a partilha de ficheiros MP3 com músicas. Inicialmente essa partilha baseava-se na utilização de *flash pens*, um sistema semelhante à partilha de CDs entre amigos, mas com o aparecimento da banda larga, tornou-se viável transferi-los directamente entre computadores ligados à Internet. Qualquer música poderia passar a estar à distância de um clique desde que se conhecesse o seu URL para *download*.

Surgiu então um serviço de indexação das músicas existentes nos computadores pessoais dos seus utilizadores, que dada uma música, permitia obter URLs onde a mesma estava disponível para *download*. O serviço chamava-se Napster e tornou-se célebre pois atingiu rapidamente várias dezenas de milhões de utilizadores.

Os servidores centrais do Napster não distribuíam músicas, apenas funcionavam como uma espécie de intermediário entre os utilizadores que as queriam trocar entre

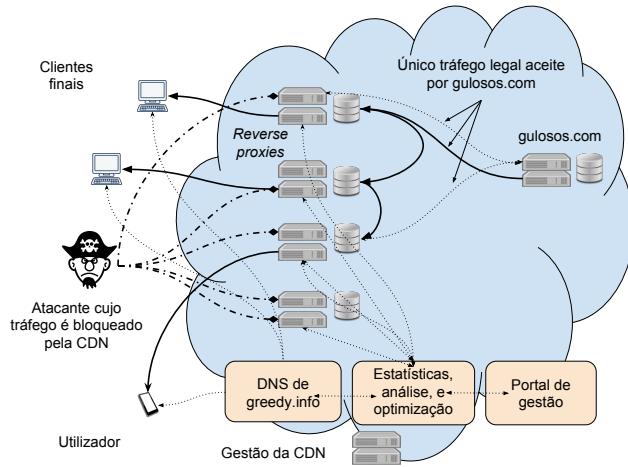


Figura 13.9: Utilização da CDN para providenciar serviços de segurança

si, mas como todo o ecossistema que o Napster alimentava violava os direitos de autor, o serviço foi fechado por ordem judicial.

A Figura 13.10 mostra a utilização de um servidor de indexação de conteúdos para a troca directa de conteúdos entre os utilizadores. O utilizador da esquerda, que quer partilhar músicas, contacta o servidor e indica-lhe, na forma de uma lista de URLs, as músicas que estão disponíveis no seu computador. Mais tarde, a utilizadora da direita, que procura uma dada música, contacta o servidor para obter uma lista de URLs. Se a lista devolvida pelo servidor for não vazia, escolhe uma das alternativas e usa-a para fazer o *download*.

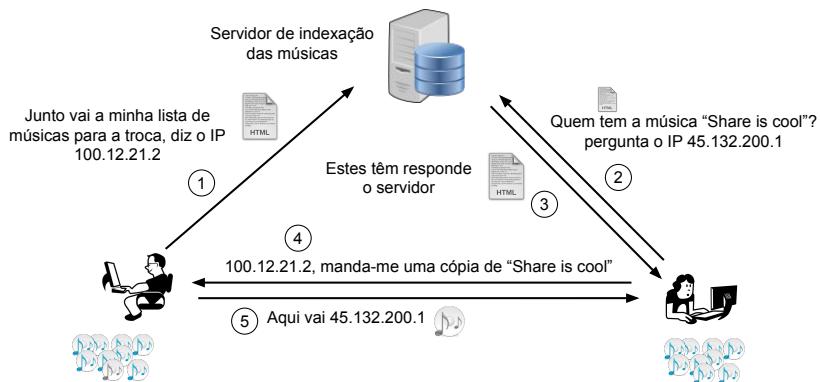


Figura 13.10: Um servidor de indexação de conteúdos proporciona a troca directa dos mesmos entre os utilizadores

Independentemente da polémica à volta de questões legais ou éticas e dos preços praticados pelas editoras musicais, o sistema mostrou que era tecnicamente possível fazer uma CDN com servidores de indexação de conteúdos, mas com a distribuição dos conteúdos assegurada pelos próprios utilizadores finais. Mais, fora o facto de os

servidores de indexação constituírem um ponto central de falha, o sistema exibia propriedades muito interessantes. Nomeadamente, a sua capacidade subia com a procura, pois cada participante obtinha músicas, mas também tinha de contribuir com músicas.

Os sistemas de distribuição de conteúdos baseados na colaboração entre os participantes, com ou sem coordenação central, passaram a chamar-se sistemas P2P (*Peer-to-Peer*) de distribuição de conteúdos pois baseiam-se na colaboração entre pares, sem distinção entre clientes e servidores. A seguir vamos ver duas categorias de sistemas deste tipo. Na primeira, os participantes agregam-se de forma aleatória e o sistema não exibe nenhuma estrutura aparente. São os sistemas P2P não estruturados de que o protocolo BitTorrent é o exemplo mais popular. Na segunda estão os sistemas P2P estruturados, de que os exemplos mais significativos são as chamadas DHT (*Distributed Hash Tables*).

Começaremos por um simples exemplo para motivar as vantagens dos sistemas P2P não estruturados. Por hipótese, pretende-se entregar um ficheiro com 5 Gbytes ($40 \text{ Gbits} \approx 4.10^{10} \text{ bits}$) a 10.000 utilizadores (10^5). A quantidade total de informação que deverá chegar ao conjunto dos utilizadores é 4.10^{15} bits . Com um servidor ligado com o débito de 1 Gbps = 10^9 bps , precisaríamos de $4.10^{15}/10^9 = 4.10^6 = 4 \text{ milhões}$ de segundos, ou seja mais de um mês, para o ficheiro chegar a todos os clientes. Dispondo de 10 servidores ligados a 1 Gbps, já só precisaríamos de $4.10^{15}/10^8 = 4.10^5 = 400.000 \text{ segundos}$, i.e., cerca de 110 horas (mais de 4 dias). Os clientes teriam de ter a capacidade de *download* de pelo menos 1 Mbps pois $10.000 \times 1 \text{ Mbps} = 10 \text{ Gbps}$. Se estes estivessem ligados por canais com débitos inferiores, os clientes e não o servidor é que retardariam a distribuição do ficheiro.

Admitamos que os clientes estão ligados à rede por canais *full duplex* com o débito de 1 Mbps em cada sentido⁴. Se arranjarmos alguma forma de $n = 10.000$ destes computadores irem trocando entre si partes do ficheiro continuamente, de tal forma que cada computador receba continuamente de outros as partes do ficheiro que lhe faltam, a troco de enviar-lhes as partes que já tem, essa rede teria exactamente a mesma capacidade agregada de *upload* que os 10 servidores. Ou seja, se os 10.000 clientes ligados a 1 Mbps conseguissem estar sempre a trocar entre si informação não duplicada, com o sistema sempre a fazer progresso colectivo, a sua capacidade agregada é a mesma que a de uma infra-estrutura, provavelmente mais cara e necessariamente gerida centralmente (i.e., 10 servidores ligados à Internet a 1 Gbps).

Em ambos os cenários estamos a admitir algumas hipóteses simplistas, nomeadamente, que seria possível estar a transmitir continuamente com os débitos indicados, o que pressupõe que a única limitação das transferências viria dos canais que ligam os computadores à rede, não havendo *bottleneck links* no interior da rede. Adicionalmente estamos a admitir que não se perdem pacotes devido a erros.

A infra-estrutura baseada em servidores não coloca desafios técnicos, apenas financeiros. No entanto, a versão baseada na distribuição cooperativa do ficheiro coloca desafios técnicos bastante grandes. Nomeadamente, como manter a troca sempre a fazer progresso, e fazer com que cada computador receba continuamente informação nova, a troco exactamente da informação que já tem.

Uma primeira solução poderia consistir em dividir o ficheiro em n partes iguais, tantas quanto o número de participantes, entregar cada uma delas a cada um dos participantes e depois, cada participante faria chegar a sua parte aos outros $n - 1$ participantes. A Figura 13.11 ilustra as duas alternativas com 4 participantes. À esquerda, o servidor envia o ficheiro para todos os clientes e os canais destes só são usados para *download*. À direita, os 4 participantes já têm cada um quarto diferente do ficheiro e trocam esses pedaços entre si aproveitando a capacidade dos seus canais em ambos sentidos.

⁴ O valor escolhido é hoje em dia baixo em certos cenários, mas foi escolhido para simplificar a discussão.

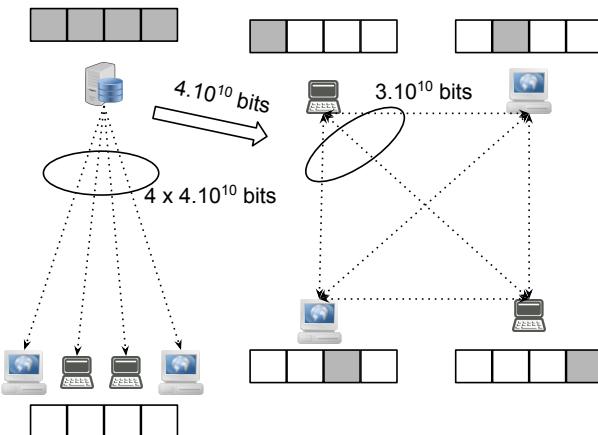


Figura 13.11: Duas alternativas para distribuir um ficheiro por 4 utilizadores

Transformar esta ideia num sistema realista que funcione apenas com base na colaboração voluntária dos participantes é um grande desafio, que passa por: dividir o ficheiro em tantos bocados quantos os participantes, distribuir os bocados pelos participantes, levar cada participante a entregar o seu bocado a todos os outros, *etc.* Infelizmente, como os participantes estão sempre a entrar e a sair e não se sabe o seu número, nem eles se conhecem *a priori* uns aos outros, a solução não se afigura muito realista. Mesmo que se conhecesse o número de clientes, em certas situações os pedaços dos ficheiros seriam tão pequenos que a solução seria impossível.

Resolver o problema como indicado não é possível, mas é possível fazer um sistema inspirado pelas mesmas ideias: 1) dividir o ficheiro em partes de dimensão fixa; 2) escolher aleatoriamente os parceiros com que se trocam pedaços do ficheiro; 3) cada parceiro deve tentar arranjar pedaços que faltem aos outros para distribuir os pedaços pelos diferentes participantes de forma o mais equilibrada que possível; e 4) cada participante deve recusar colaborar com os que não colaboram com ele.

Estes mecanismos foram inventados por Bram Cohen [Cohen, 2003] que os concretizou no protocolo BitTorrent (acessível em <http://bittorrent.org>) e num sistema de troca de ficheiros baseado nele. Este protocolo é hoje em dia utilizado por diversos programas que permitem a distribuição P2P de ficheiros.

Protocolo e sistema BitTorrent

Segundo a terminologia do protocolo BitTorrent⁵, o conjunto dos utilizadores que estão num determinado momento a tentar obter ficheiros em conjunto através do protocolo BitTorrent, chama-se um enxame (*swarm*). Destes, cada subconjunto que pretende obter um dado ficheiro concreto chama-se uma torrente (*torrent*).

O conjunto dos membros da torrente é variável e esse processo de rotatividade dos membros é geralmente designado por *user churn*. A qualquer momento novos membros podem chegar e outros podem partir. Os primeiros porque querem obter uma cópia do ficheiro. Os segundos porque já têm uma cópia e comportam-se de forma egoísta, ou porque desistiram. Alguns membros ficam a distribuir o ficheiro mesmo depois de já terem a sua própria cópia. Estes membros têm um comportamento altruísta e chamam-se, novamente na terminologia do protocolo, as sementes (*seeds* ou

⁵ O protocolo BitTorrent usa uma terminologia bastante metafórica que conservaremos no original, e geralmente na língua inglesa.

seeders). Uma torrente com poucos membros, ou sem a presença de nenhuma *seed*, provavelmente não consegue distribuir o ficheiro.

Cada ficheiro a distribuir pelo protocolo é dividido em segmentos, designados por *pieces*, cada um dos quais com de 64 a 512 Kbytes. Para cada *piece* é calculada uma chave criptográfica de validação que permite verificar a sua autenticidade. É também criado um ficheiro de meta dados com a extensão **.torrent**, que contém diversas propriedades do ficheiro a distribuir: nome, dimensão total, dimensão de cada *piece*, lista das chaves de autenticação das mesmas, e URL de um servidor especial, chamado *tracker* desta torrente. Este ficheiro é geralmente 2 a 3 ordens de grandeza mais pequeno que o ficheiro a distribuir, e é colocado num *site* público onde os interessados o podem obter.

A coordenação da torrente é realizada pelo *tracker*. Todos os novos participantes começam por registar-se nele e este registo tem de ser revalidado periodicamente sob pena de o *tracker* considerar que o cliente abandonou a torrente. O *tracker* fornece a cada cliente uma lista aleatória de outros participantes na mesma torrente (e.g., 50 ou mais participantes). Desta forma, qualquer participante activo na torrente é conhecido do *tracker* e pode conhecer um subconjunto aleatório de membros da mesma.

Um participante *P*, depois de se registar e obter um conjunto aleatório de outros participantes, tenta abrir uma conexão TCP para alguns deles, geralmente cerca de 40. A lista de participantes para os quais *P* tem ligações vai variando, quer porque alguns partem, quer porque novos estabelecem ligações com *P*. Quando constata que a sua lista de parceiros é pequena, *P* pode ir ao *tracker* obter mais candidatos.

Inicialmente, mas também sempre que adquire mais *pieces*, *P* envia para cada um dos parceiros a que está ligado a lista das *pieces* que possui. Desta forma todos ficam a conhecer a distribuição das *pieces*. Após obter estas informações, *P* pede a cada um dos seus parceiros uma *piece* que não tem. Naturalmente, o processo é simétrico e alguns dos seus parceiros vão fazer o mesmo, como ilustra a Figura 13.12.

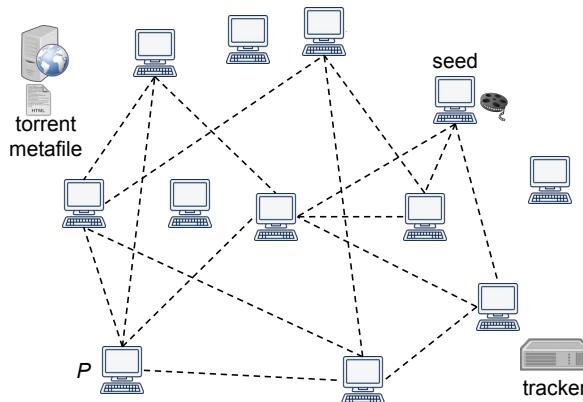


Figura 13.12: Uma torrente em funcionamento. As linhas representam parceiros a trocarem *pieces*.

Boa parte do êxito do protocolo BitTorrent está intimamente ligado à forma como várias decisões são tomadas: que *pieces* *P* solicita primeiro aos seus parceiros e para que parceiros aceita enviar *pieces*. A primeira destas decisões é baseada numa estratégia designada **rarest first**. A ideia é privilegiar as *pieces* mais raros para tentar, tanto quanto possível, distribuir o conjunto das *pieces* o mais equitativamente possível pelos

diferentes participantes, e também fazer com que cada participante seja mais atrativo para os outros pois, quanto mais raros são as *pieces* que tem, maior é a probabilidade de lhos pedirem, o que é essencial para o valorizar.

Inicialmente o protocolo só pode funcionar se na torrente estiver presente pelo menos uma *seed*. No entanto, a estratégia *rarest first* vai tentar fazer com que as *pieces* se distribuam pelos membros da torrente de forma a, logo que possível, mesmo que a *seed* desaparecesse, a união das *pieces* de todos os membros ainda activos contivesse a totalidade do ficheiro.

Depois de *P* seleccionar as *pieces* a pedir, envia os respectivos pedidos. Inicialmente *P* não tem *pieces* para enviar e fica dependente da boa vontade dos outros. Só depois de ter pelo menos uma *piece* é que *P* pode participar em trocas simétricas. O protocolo procura que cada parceiro esteja activamente a trocar *pieces* com 4 ou 5 dos seus parceiros, privilegiando sempre que pode a simetria das trocas. Como os parceiros são escolhidos e como variam é também realizado segundo uma estratégia que consiste em privilegiar os parceiros que fornecem blocos mais depressa. De facto, quanto mais depressa *P* receber *pieces*, mais depressa obtém o ficheiro.

Em cada momento cada parceiro está a enviar *pieces* a 4 ou 5 parceiros e a receber pedidos de todos os outros. Os parceiros a quem *P* está a enviar *pieces* dizem-se estar no estado *unchoked*. Para tomar decisões, *P* vai monitorando a velocidade extremo a extremo da troca de *pieces* com os seus parceiros activos. Periodicamente, *e.g.*, de 30 em 30 segundos, *P* aceita dar uma *piece* a um novo parceiro, mesmo que este não lhe esteja a dar nada. O feliz contemplado, diz o protocolo, é *optimistically unchoked*. Na verdade é este mecanismo que permite a um parceiro recém chegado começar a obter *pieces* porque ainda não tem nenhuma para troca.

Cada parceiro acaba por ficar geralmente com 5 parceiros activos mas, logo que obtém mais estatísticas sobre as velocidades com que recebe blocos, selecciona dos 5 aquele que lhe está a dar *pieces* mais devagar e coloca-o no estado *choked*, *i.e.*, deixa de lhe fornecer *pieces*.

Este processo leva a que *P* privilegie continuamente entre os seus parceiros aqueles que lhe dão *pieces* com maior débito e, como estes fazem o mesmo, as trocas são maximizadas. Esta estratégia foi designada pelo autor do protocolo como “toma lá dá cá” (*tit-for-tat*). A mesma conduziria a conjuntos fechados de parceiros, não fosse o mecanismo de recenseamento das *pieces* em falta e o mecanismo de ir tentando aceitar periodicamente fazer trocas com outros parceiros até af desconhecidos.

É fácil de perceber que existem algumas estratégias que favorecem o “bem comum” da rede, como por exemplo: 1) ficar na torrente mesmo depois de obter o ficheiro, *i.e.*, os chamados *seeders* (o que levou à constituição de redes P2P fechadas onde os parceiros recebem pontos por fornecerem blocos aos outros e são por isso privilegiados nas trocas); 2) tentar que o *tracker* forneça listas de subconjuntos de participantes que estejam potencialmente mais “próximos” uns dos outros (em termos de débito de transferência), para evitar tanto quanto possível tráfego de trânsito; 3) tentar agregar vários ficheiros mais pequenos num grande para aumentar a procura e o volume da torrente; *etc.*

Por outro lado existem também comportamentos que prejudicam a rede, como por exemplo tentar obter *pieces* sem dar nenhuma aos outros, os chamados *free riders* ou *leechers*, pois estes consomem recursos da rede sem darem nada em troca. Ora a capacidade da rede é maximizada quando existe simetria entre a informação recebida e fornecida. Este comportamento é combatido pelo algoritmo de *choke / unchoke* mas sem total êxito pois ele dá o benefício da dúvida aos principiantes. No entanto, resolver este problema não se revelou simples e o mesmo foi sujeito a intensa investigação.

Surpreendentemente, protocolos como o BitTorrent revelaram-se muito interessantes e a sua eficiência depende de os participantes se comportarem como *cidadãos exemplares*. Ora, nem sempre é fácil encontrar a forma de dar os bons incentivos pois, no fim do dia, a maioria dos participantes o que pretende é obter os ficheiros o mais

depressa possível e pouco se importar com os outros. De alguma forma o tema cruza a tecnologia com a economia e a sociologia.

Tal como apresentado, o protocolo exibe uma fraqueza que tem a ver com um seu ponto central de falha, o *tracker*, que se for atacado, pára o progresso do enxame. A solução para esse problema foi posteriormente introduzida distribuindo a função do *tracker*. Posteriormente, a maioria dos diversos softwares que implementam o protocolo BitTorrent implementam mecanismos alternativos ao *tracker* para realizar a descoberta de parceiros. Esses mecanismos baseiam-se na colaboração dos diferentes *peers* entre si de forma a implementarem aquilo que se designa por uma DHT - *Distributed Hash Table*. Estas estruturas de dados distribuídas serão discutidas a seguir.

Foram também introduzidas variantes do protocolo BitTorrent para distribuição P2P de vídeo em tempo real (*live video streaming*) [Abboud et al., 2011]. No entanto, neste caso, as próximas *pieces* a pedir aos parceiros são seleccionadas também por uma estratégia que toma também em consideração o momento em que o conteúdo das *pieces* é necessário ao utilizador final.

Durante os primeiros 15 anos do Século XXI foram desenvolvidas inúmeras redes cooperativas de distribuição de conteúdos, geralmente designadas como **sistemas P2P (Peer-to-Peer)**. Estes podem agrupar-se em **sistemas P2P estruturados e não estruturados**.

Os sistemas P2P não estruturados mais conhecidos baseiam-se no protocolo BitTorrent. Este protocolo revelou-se muito eficiente no suporte à distribuição de ficheiros de grande dimensão, quando o número de interessados simultâneo é muito grande. O protocolo permite aproveitar de forma eficiente a capacidade de *upload* dos diferentes participantes para distribuírem entre si partes do ficheiro. O problema principal do protocolo está relacionado com a gestão dos incentivos que levem os participantes a entrarem no sistema e a utilizarem-no de forma não egoísta.

DHT - Distributed Hash Tables

As tabelas de dispersão (*hash tables*) são estruturas de dados muito úteis que servem para realizar dicionários, *i.e.*, estruturas que associam chaves a valores. Tornar um dicionário (ou uma directória) acessível remotamente, é relativamente fácil. Basta integrá-lo num servidor e permitir a sua manipulação através de um protocolo cliente / servidor. Esta solução tem o inconveniente de centralizar a gestão do dicionário e de criar um ponto central de falha.

O DNS é um dicionário⁶ distribuído e hierárquico que tem o inconveniente de requerer um conjunto de servidores rigidamente organizado para a sua gestão. De facto o DNS não admite que os servidores partam e voltem sempre que lhes apetecer. Adicionalmente, tem a fragilidade associada à sua organização hierárquica pois qualquer ataque à totalidade dos servidores de um domínio, torna inacessíveis todos os seus subdomínios.

Os sistemas P2P como o BitTorrent, e outros sistemas P2P igualmente não estruturados, também requerem dicionários para a localização de parceiros ou dos próprios conteúdos. Esses sistemas são baseados num ponto de coordenação central (o *tracker* do BitTorrent ou os servidores centrais do Napster) ou então usam métodos muito inefficientes, baseados em inundação da rede com pedidos de localização de parceiros ou de conteúdos, como por exemplo o sistema Gnutella.

⁶ As chaves poderiam ser criadas a partir dos nomes DNS concatenados com o tipo do RR da consulta como por exemplo: *digest("www.wikipidea.org A")*.

Em paralelo com o desenvolvimento de sistemas P2P não estruturados, os investigadores desenvolveram um tipo de sistemas P2P ditos estruturados, que implementam dicionários distribuídos de forma descentralizada e com filiação variável. Ou seja, os participantes no sistema, que também se designam por nós da rede, podem juntar-se ou deixar a rede de livre vontade e a qualquer momento, sem que o dicionário deixe de funcionar.

O desafio a vencer era conceber um dicionário distribuído, implementado com base na colaboração de um conjunto variável de parceiros (P2P), sem coordenação central de qualquer tipo. Esses sistemas vieram a designar-se por DHT - *Distributed Hash Tables*. Os sistemas P2P mais populares hoje em dia usam o protocolo BitTorrent acrescido de uma DHT em substituição do *tracker*. Vamos então ver de forma introdutória o que é e como funciona uma DHT.

Ilustração do funcionamento de uma DHT

Cada item registrado na DHT tem de ter uma chave única e a DHT fornece uma interface com as seguintes operações:

```
dht.put(chave, valor)  
valor = dht.get(chave)
```

que permitem, respectivamente, registar um valor associado a uma chave, e obter o valor associado a uma chave. Os valores associados às chaves são arbitrários e dependem do contexto de aplicação. Por exemplo, para distribuição de ficheiros, cada ficheiro tem de ter uma chave própria, e o valor pode ser a lista de nós da sua torrente.

Para as chaves usam-se números únicos gerados, por exemplo, a partir de uma função *digest*. Esta, dado um conteúdo arbitrário, gera um número fixo de m bits (*e.g.*, $m = 160$), em princípio único⁷. Assim, uma chave k é tal que $k \in [0, 2^m - 1]$. Por exemplo, no caso do sistema BitTorrent, um ficheiro pode ter como chave $k = \text{digest}(\text{torrent})$, ou seja, o resultado da aplicação da função de *digest* ao conteúdo do ficheiro de meta dados (com extensão `.torrent`).

Cada nó físico do sistema tem também uma chave, por exemplo, o resultado da aplicação da função de *digest* ao seu endereço IP e a outros valores que o caracterizem (*e.g.*, $n = \text{digest}(\text{endereço IP} + \#ram + \dots)$). À chave de um nó físico chama-se o seu identificador. A probabilidade dos identificadores de nós distintos ser a mesma, pode ser tão reduzida quanto necessário, usando funções de *digest* que devolvem o número adequado de bits.

Existem diversos tipos de DHTs, mas as mais simples organizam logicamente todos os identificadores presentes na rede num anel (em círculo), ver Figura 13.13. Os identificadores estão colocados no anel por ordem crescente. Alguns dos identificadores representam nós físicos da rede. Outros correspondem às chaves de conteúdos. Os primeiros são responsáveis pela organização material do anel.

Consideremos apenas os nós responsáveis pela organização material do anel aos quais chamaremos, daqui para a frente, nós propriamente ditos ou simplesmente nós. Os outros membros do anel são designados por conteúdos e são identificados pelas suas chaves. Sobre os identificadores dos nós é possível definir trivialmente as funções: 1) *succ(i)* - que devolve o identificador do nó que no anel está colocado a seguir ao nó i e 2) *pred(i)* - que devolve o identificador do nó que no anel está colocado antes do nó i . Num anel com um só nó, este é sucessor e predecessor de si próprio.

⁷ Como já foi referido, as funções de *digest* asseguram a seguinte propriedade: dados dois conteúdos arbitrários distintos, a probabilidade de os respectivos *digest* serem iguais é desprezável na prática e pode ser tão pequena quanto desejado alterando a função e aumentando o espaço das chaves. Adicionalmente, é também praticamente impossível gerar um conteúdo com um dado *digest*.

No que diz respeito às chaves dos conteúdos, é também possível definir a função $sucessor(k)$, a qual devolve o identificador do nó físico cujo identificador é o menor dos identificadores de nós físicos superior ou igual a k . Por outras palavras, $sucessor(k)$ é o identificador do mais próximo nó que no anel está colocado a seguir à chave k . Esta noção tem importância pois, por convenção, $sucessor(k)$ é o identificador do nó responsável por memorizar o valor associado à chave k . Por exemplo, na Figura 13.13, $702 = sucessor(640)$ e $702 = succ(580)$.

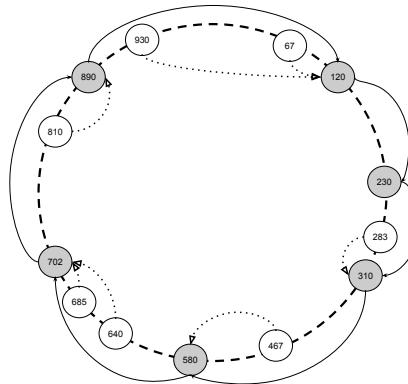


Figura 13.13: Organização lógica de um conjunto de identificadores em anel. Os nós a cinzento representam nós físicos da rede. Os restantes representam chaves de conteúdo. As setas a negro representam as ligações entre nós físicos. As setas a tracejado representam a ligação da chave k ao nó $sucessor(k)$, nó responsável por memorizar o valor associado a k . O domínio das chaves é de 0 a 1023.

Inicialmente a rede só tem um nó. Mais tarde já tem vários nós. Um nó com o identificador n , tem de conhecer o endereço IP de um nó qualquer da rede para poder entrar. Em seguida executa a função $succ(n)$ e obtém o endereço IP do nó do anel com esse identificador, assim como o endereço IP do nó com identificador $pred(succ(n))$. Nessa altura n comunica ao nó que vai ser o seu predecessor para se ligar a ele, e depois liga-se ao nó que era o sucessor desse nó. Quando um nó sai de forma planeada, comunica ao seu predecessor para se ligar ao seu sucessor. Para evitar que o anel se parta quando um nó sai sem avisar, cada nó tem também de conhecer o endereço IP e o identificador do sucessor do seu sucessor ($succ(succ(n))$). Periodicamente o nó n testa a conectividade até $succ(n)$ e se esta se quebrar, liga-se a $succ(succ(n))$.

Como cada nó de identificador n memoriza os valores associados às chaves k tal que $k \in [pred(n), n]$, a entrada de um novo nó só fica completa depois de este obter do seu sucessor a cópia dos valores porque passa a ser responsável.

Quando se pretende memorizar ou obter valores da DHT usando a chave k , é necessário obter o endereço IP do nó $sucessor(k)$ pois este nó é responsável pela entrada da DHT de chave k . A pesquisa no anel do nó responsável pela chave k pode ser linear: partindo de um nó qualquer, percorre-se o anel até chegar ao $successor(k)$.

Para evitar que desapareça o valor associado à chave k quando um nó sai da rede de forma não planeada, esse valor deve estar replicado no anel, usando os nós sucessores e antecessores do nó $successor(k)$ (e.g., $pred(successor(k))$ e $succ(successor(k))$), ou usando outras funções de *digest*, i.e., diferentes instâncias da função $successor(k)$, para determinar outras chaves e portanto outras posições para o valor associado a k no

anel.

O sistema Chord [Stoica et al., 2001] foi um dos primeiros sistemas a organizar uma DHT em anel e propôs também uma forma de organização do anel que suporta pesquisa binária no espaço das chaves. Neste sistema, cada nó tem uma tabela, chamada *finger table*, que contém os endereços IP de outros nós no anel que permitem chegar mais depressa ao *successor(k)*.

Resumidamente, dado um nó com o identificador k , a sua *finger table* contém um conjunto de entradas tal que a entrada com o índice i contém o endereço IP do nó $\text{succ}(k + 2^i)$, com soma módulo 2^m . Assim, a entrada 0 contém o endereço IP do sucessor deste nó, $\text{succ}(k + 1)$. A entrada 1 contém o endereço IP de $\text{succ}(k + 2)$, a entrada 2 contém o endereço IP de $\text{succ}(k + 4)$, a entrada 3 contém o endereço IP de $\text{succ}(k + 8)$, etc. A Figura 13.14 apresenta um exemplo. As *fingers tables* dos nós permitem percorrer o anel de forma mais eficiente, como numa pesquisa binária, pois suportam saltos a distâncias bem definidas no anel

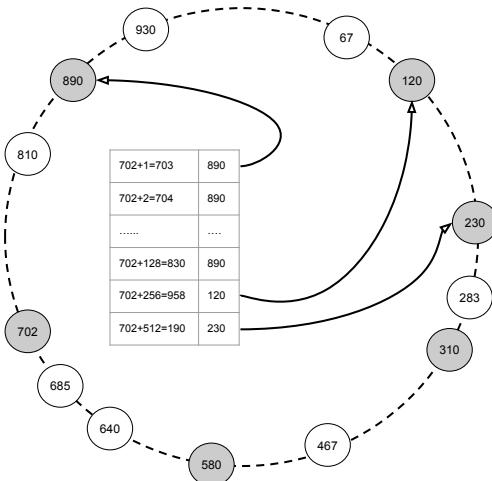


Figura 13.14: Um anel Chord e a *finger table* do nó 702. O domínio das chaves é de 0 a 1023 e as somas são realizadas módulo 1024.

Naturalmente, estas tabelas podem ficar desactualizadas com a entrada e saída de nós da rede. Por isso, um processo periódico é executado em cada nó para as manter actualizadas.

Os sistemas P2P estruturados mais comuns são conhecidos como **DHTs - Distributed Hash Tables**. Estas implementam um dicionário distribuído cooperativo, que continua em funcionamento mesmo quando o número de participantes varia.

Os sistemas P2P em geral, e as DHTs em particular, motivaram uma investigação muito extensa, *i.e.*, em largura, e intensa, *i.e.*, em profundidade. Foram desenvolvidos sistemas P2P para execução de cálculos em paralelo (*e.g.*, Seti@Home), para suporte de IP Multicasting na Internet global (*e.g.*, [Chu et al., 2000]), para melhoria do encaminhamento na Internet (*e.g.*, [Andersen et al., 2001]), assim como inúmeras variantes de sistemas P2P estruturados e não estruturados para distribuição de conteúdos ou outras aplicações distribuídas de gestão de informação. Existem igualmente utilizações conjuntas de sistemas P2P estruturados e não estruturados, como por exemplo

no quadro do suporte ao funcionamento do protocolo BitTorrent em que é utilizada uma DHT para substituição do *tracker* [Zhang et al., 2011]. Diversas panorâmicas dos sistemas P2P são apresentadas em numerosos artigos (*e.g.*, [Theotokis and Spinellis, 2004] e [Rodrigues and Druschel, 2010]).

Com a generalização de CDNs baseadas em infra-estrutura para distribuição de conteúdos multimédia (*e.g.*, música, filmes, séries, ...), com planos de preços relativamente acessíveis aos consumidores dos países mais desenvolvidos, a utilização de sistemas P2P para distribuição de conteúdos deixou de ser tão popular como já o foi, e a percentagem de tráfego na Internet que lhes é atribuído tem vindo a decrescer.

No entanto, os gigantes da distribuição de conteúdos, que operam geralmente através de CDNs, estão também, no momento de escrita deste livro, a tentar que os clientes dos seus serviços participem (sem saberem) no apoio ao funcionamento da CDN usando técnicas P2P, inspiradas do protocolo BitTorrent, para se apoiarem mutuamente na distribuição de conteúdos que estão a partilhar no momento. Adicionalmente, para além dos exemplos anteriores, mas fora do quadro puro da distribuição de conteúdos, formas especiais de DHTs com centenas de milhares de nós são utilizadas no interior de centros de dados para suporte de sistemas de computação em nuvem.

Os sistemas P2P representam mais de 15 anos de investigação intensiva, motivada por problemas reais, que levou à introdução de inúmeras invenções e tecnologias elegantes, sofisticadas e muito interessantes. Como com todas as tecnologias, a alteração de condições envolventes tem implicações sobre a sua utilização concreta.

13.5 Resumo e referências

Resumo

A distribuição de conteúdos via a Web tornou-se um problema muito relevante quando o número de utilizadores começou a explodir. Inicialmente, para melhorar a experiência dos utilizadores, foram introduzidos servidores *proxies* Web, partilhados entre os vários utilizadores da mesma rede de acesso, para complementar o *caching* realizado pelos *browsers* Web.

Adicionalmente, os serviços com muitos utilizadores passaram a usar vários servidores, primeiro para servirem a parte estática dos seus conteúdos, e depois mesmo para a parte dinâmica. Este tipo de configuração do serviço baseou-se em **mecanismos de distribuição de carga por agregados (clusters) de servidores** situados no mesmo centro de dados.

A evolução seguinte consistiu em instalar réplicas dos servidores de conteúdos junto dos clientes, de forma a diminuir o RTT entre os clientes e os conteúdos estáticos. O mecanismo de distribuição de carga neste caso tem de tomar em consideração a localização dos clientes o que levou a soluções baseadas em servidores de DNS que resolvem os nomes dos serviços em função da localização dos seus clientes. Os servidores de *caching*, agora colocados junto dos clientes, passaram a representar directamente o serviço e chamam-se *proxies inversos*.

A fase seguinte consistiu em desenvolver *Content Distribution Networks* ou **CDNs**. Uma CDN baseada numa infra-estrutura dedicada é uma **rede lógica justaposta (overlay network)** de servidores, com o objectivo de providenciar de forma mais eficiente e conveniente um serviço distribuído que usa a Internet. Os servidores da CDN coordenam-se entre si também pela Internet, que funciona neste caso como uma **rede de suporte (underlay network)**. Estas infra-estruturas dispõem de servidores espalhados por toda a Internet, e recorrem a servidores DNS com geo-localização, a *proxies* inversos (*in the edge*), monitorização e algoritmos de optimização, assim como a filtragem e redirecção de pedidos HTTP.

As CDNs podem ser dedicadas ou partilhadas, hierárquicas, *i.e.*, complementando os *proxies* com centros de dados regionais, e podem fornecer vários serviços de valor acrescentado como por exemplo: absorção de carga, aceleração de aplicações, proteção contra falhas de funcionamento, segurança e proteção contra ataques.

Ao mesmo tempo que foram desenvolvidas CDNs baseadas em infra-estrutura, foram igualmente desenvolvidos sistemas alternativos baseados na colaboração entre os utilizadores. Assim, durante os primeiros 10 anos do Século XXI foram desenvolvidas inúmeras redes cooperativas de distribuição de conteúdos, geralmente designadas como **sistemas P2P (*Peer-to-Peer*)**. Estes podem agrupar-se em **sistemas P2P não estruturados e estruturados**.

Os sistemas P2P não estruturados de distribuição de conteúdos mais populares baseiam-se no protocolo BitTorrent. Este protocolo revelou-se muito eficiente no suporte à distribuição de ficheiros de grande dimensão, quando o número de interessados simultâneo é muito grande. O protocolo permite aproveitar de forma eficiente a capacidade de *upload* dos diferentes participantes para distribuírem entre si partes do ficheiro. O problema principal destes sistemas está relacionado com a gestão dos incentivos que movem os participantes a entrarem no sistema e a utilizarem-no de forma não egoísta.

Os sistemas P2P estruturados mais conhecidos são conhecidos como **DHTs - Distributed Hash Tables**, e motivaram um grande entusiasmo nas academias e uma investigação muito intensa. No entanto, as DHTs tiveram um impacto pouco profundo na distribuição de conteúdos pois, com a generalização de CDNs baseadas em infra-estrutura para distribuição de conteúdos multimédia, a utilização de CDNs cooperativas ou P2P representa uma cada vez menor fracção do tráfego da Internet. É relativamente consensual entre vários observadores que o tráfego na Internet vai ser dominado pela distribuição de vídeo recorrendo a infra-estruturas dedicadas.

Alguns dos termos introduzidos no capítulo são a seguir passados em revista. Entre parêntesis, quando necessário, figura a tradução mais comum em língua inglesa.

Principais conceitos

Proxy HTTP (servidor procurador) Servidores que actuam como representantes de outros servidores. Os *proxies* são utilizados para melhorar ou modificar, através de intermediação, o acesso aos serviços que representam. O principal mecanismo que usam para acelerar o acesso à Web é o *caching* de objectos imutáveis ou cujo estado evolui muito lentamente.

Rácio de sucesso (*hit ratio*) Quociente do total de pedidos servidos directamente da memória *cache* sobre a totalidade dos pedidos recebidos.

Distribuição de carga Mecanismo que permite usar vários servidores para satisfazer um elevado conjunto de solicitações a um serviço. Este é o principal mecanismo usado para melhorar a qualidade de serviço de um serviço Web muito popular, distribuindo os pedidos dos clientes por vários servidores.

Proxy inverso (*reverse proxy*) Servidor controlado por um prestador de serviços, que representa o seu serviço junto dos clientes. Os pedidos dirigidos ao serviço são dirigidos aos *reverse proxies* através de servidores DNS, que utilizam geolocalização para dirigirem os clientes para o *proxy* mais próximo.

Geolocalização (*IP Geolocation*) Processo que permite associar um endereço IP a uma localização geográfica. Esta informação pode ser útil por razões de segurança ou para melhor servir os clientes. De facto, a distância geográfica entre dois endereços IP geo-localizados pode funcionar como uma indicação indirecta do tempo de trânsito aproximado entre os mesmos.

CDN (*Content Distribution Network*) Rede de distribuição de conteúdos, *i.e.*, uma rede lógica justaposta (*overlay network*) de servidores com o objectivo de pro-

videnciar de forma mais eficiente e conveniente um serviço distribuído que usa a Internet.

CDN P2P CDN especial, constituída pelos computadores dos utilizadores do serviço, que colaboram entre si, e que servem para distribuir conteúdos de forma cooperativa ou para implementar DHTs.

DHT (*Distributed Hash Table*) Sistema P2P especialmente adequado a implementar de forma distribuída dicionários que memorizam a associação entre chaves e valores. As DHTs usam um conjunto variável de participantes para implementar o dicionário e mantém o dicionário em funcionamento mesmo com essa variação dos membros do sistema.

Aspectos complementares

Em inúmeras situações verificou-se que a distribuição da popularidade dos conteúdos na Web, da importância dos nós de comutação de pacotes nas redes de computadores, da importância das redes de trânsito na Internet, da distribuição da riqueza entre as pessoas, da popularidade das publicações científicas em termos de referências bibliográficas mútuas, da popularidade das pessoas nas redes sociais, da população das cidades, *etc.* seguem distribuições em que um pequeno número de entidades são dominantes e absorvem uma fracção muito significativa da importância relativa.

O estudo destas distribuições tornou-se em anos recentes um tema conhecido como *Network Science*. O leitor com curiosidade por esse fascinante mundo das propriedades de objectos interligados, poderá estudar o livro [David and Jon, 2010], escrito por dois dos mais proeminentes cientistas estudiosos deste tema.

A Web, sendo constituída por objectos digitais interligados por hyper-ligações, também tem sido objecto dos mesmos estudos e verificou-se que a popularidade dos conteúdos, também segue o mesmo tipo de distribuições, com um pequeno conjunto de nós dominantes e muitos outros de menor importância. Estas propriedades são muito importantes e têm um grande impacto no estudo da popularidade e na distribuição de conteúdos na Internet. Nomeadamente na efectividade dos mecanismos de *caching* e no funcionamento das CDNs. Por exemplo, os conteúdos que fazem parte do conjunto mais significativo devem ser tratados de forma especial. No entanto, o estudo da popularidade e das interligações de objectos na Web também pode ser objecto de estudos que ultrapassam esse âmbito restrito e com perspectivas mais amplas [Hall and Tiropanis, 2012; Berners-Lee et al., 2006].

Referências

A utilização de *proxies* Web como mecanismo de aceleração do acesso aos conteúdos foi muito popular por volta do fim do Século XX e nos anos seguintes. O artigo [Wang, 1999] apresenta uma panorâmica do estado da arte por volta dessa altura.

As redes de distribuição de conteúdos têm um grande impacto no funcionamento da Internet e no acesso dos utilizadores aos conteúdos por esta distribuídos. O relatório [Pathan and Buyya, 2007] apresenta uma panorâmica dos diferentes tipos de CDNs existentes e dos mecanismos por estas usadas. O livro [Hofmann and Beaumont, 2005], apesar de já com alguns anos, apresenta um estudo das CDNs que põe em evidência os mecanismos fundamentais nos quais estas se baseiam. O artigo [Nygren et al., 2010] apresenta alguns detalhes sobre o funcionamento interno de uma das maiores CDNs partilhadas existentes.

Os sistemas P2P são muito interessantes, quer como desafio científico, quer pelos resultados concretos que proporcionam. Quando estes sistemas são meramente sustentados na cooperação entre utilizadores finais, eventualmente em oposição a fornecedores mais convencionais, despertaram polémicas e debates acalorados.

Foram desenvolvidos sistemas P2P para os mais diversos objectivos: distribuição de carga e de computações, distribuição de conteúdos, defesa do anonimato e privacidade, *etc.* O artigo [Theotokis and Spinellis, 2004] é muito citado pois apresenta uma

panorâmica global de vários sistemas P2P. O artigo [Rodrigues and Druschel, 2010], já referido, apresenta igualmente uma visão panorâmica, mas sintética e mais recente do mesmo tópico.

13.6 Questões para revisão e estudo

1. Defina o que é um servidor *proxy* Web.
2. Um utilizador está com dúvidas sobre se deve usar ou não um servidor *proxy* da rede da sua universidade para acesso a páginas Web.
 - (a) Indique duas vantagens de utilizar um *proxy* Web como intermediário de acesso a páginas http.
 - (b) Indique duas desvantagens desta forma de acesso.
3. Defina o que é um servidor *proxy* inverso (*reverse proxy*).
4. Defina o que é uma CDN.
5. Defina o que é um sistema P2P.
6. Verdade ou mentira?
 - (a) Na versão 1.1 do protocolo HTTP, um cliente HTTP que não está a usar um *proxy* pode obter a resposta a uma sua mensagem HTTP Request num tempo inferior ao RTT (tempo de ida e volta) entre si e o servidor Web.
 - (b) Na versão 1.1 do protocolo HTTP, um cliente HTTP que não está a usar um *proxy* pode obter a resposta a uma sua mensagem HTTP Request sem ter de abrir um conexão TCP para o servidor.
 - (c) A utilização de um *proxy* HTTP garante que qualquer acesso à WEB pelos clientes é sempre mais eficaz que sem *proxy*.
7. Um browser Web na rede da sua universidade acede a páginas Web de forma directa quando estas estão em servidores do domínio da sua universidade, e com recurso a um servidor *proxy* quando estas estão fora desse domínio. Fizeram-se medidas e verificou-se que o *cache hit ratio* do *proxy* era de 50%, que o tempo médio de transferência extremo a extremo dos objectos Web de servidores internos à sua universidade era desprezável, e que o tempo médio de transferência extremo a extremo de objectos Web fora da rede da sua universidade era em média de 100 ms. Qual o tempo médio que um browser Web levaria a obter os objectos directamente dos servidores externos usando o *proxy*?
8. Um browser Web na rede da sua universidade acede a páginas Web de forma directa quando estas estão em servidores do domínio da sua universidade, e com recurso a um servidor *proxy* Web quando estas estão fora desse domínio. Fizeram-se medidas e verificou-se que o *cache hit ratio* era de 33,3%, que o tempo médio de transferência extremo a extremo de objectos Web de servidores internos à rede da sua universidade era de 10 ms, e o mesmo tempo para objectos fora dessa rede, mas sem usar *proxy*, era de 100 ms. Qual o tempo médio que um browser Web leva a obter objectos de servidores externos usando o *proxy*?
9. A rede da sua universidade está ligada à Internet por um único canal com o débito de 100 Mbps e não usa nenhum servidor *proxy* Web. Na hora de ponta o tráfego de acesso à Web corresponde aproximadamente a um débito máximo médio de 90 Mbps. Se esse fosse o único tráfego no canal acima referido, qual o *cache hit ratio* de um servidor *proxy* Web a instalar que levasse a ocupação média do canal a não ultrapassar os 50%?

10. Um servidor Web pretende contar o número de pedidos recebido de um utilizador usando o *header field Authorization*. A presença de um servidor *proxy* Web entre o cliente e o servidor interfere com este objectivo?
11. Um servidor Web pretende contar o número de pedidos recebido de um utilizador usando um *header field Cookie*. A presença de um *proxy* entre o cliente e o servidor interfere com este objectivo?
12. Suponha que o ISP **bigisp.pt** recebe grandes quantidades de correio electrónico dirigido aos seus clientes. Os utilizadores do ISP têm endereços de correio electrónico da forma **utilizador@bigisp.pt**. O ISP tem 3 servidores de correio electrónico todos com o mesmo nome, **mail.bigisp.pt**, mas vários endereços IP diferentes, ver a seguir. Explique que outra(s) entrada(s) DNS o domínio **bigisp.pt** deveria ter para que as mensagens de correio electrónico que são dirigidas aos utilizadores do ISP sejam distribuídas pelos seus 3 servidores de correio electrónico. Justifique a sua resposta.

```

mail.bigisp.pt      300      IN      A      100.10.10.1
mail.bigisp.pt      300      IN      A      100.10.10.2
mail.bigisp.pt      300      IN      A      100.10.10.3
.....

```

13. Uma empresa com milhões de clientes, espalhados por todo o mundo, tem um serviço acessível através do URL: **https://omnipresente.com**. Dada a sua dimensão, a empresa tem servidores espalhados pelos diferentes continentes (África, Europa, América e Ásia) com 10 servidores em cada continente. Você foi contratado pela empresa para melhorar o desempenho do serviço. Ao testar as respostas dos servidores de DNS da empresa à pergunta: “qual o endereço IP do servidor com o nome **https://omnipresente.com**” deparou-se com a seguinte resposta:

```

omnipresente.com    36000    IN      A      193.10.10.1
omnipresente.com    36000    IN      A      193.10.10.2
.....
omnipresente.com    36000    IN      A      193.20.20.1
omnipresente.com    36000    IN      A      193.20.20.2
.....

```

em que a ordem pela qual os endereços eram listados variava em cada consulta. Existe uma ou mais coisas erradas nesta solução? Que sugestão, se alguma existe, daria para melhorar o serviço?

14. Você é gestor dos servidores de uma empresa que tem 400.000 clientes e necessita de distribuir um *update* com 1,2 G Bytes a cada um dos clientes. O seu servidor está ligado à Internet através de um canal de alta velocidade, capaz de fornecer o *update* a 1.000 clientes de cada vez em cerca de 20 minutos. Para fornecer o *update* a todos os clientes desta forma serão necessárias cerca de 400 vezes 20 minutos ou aproximadamente 5 dias. Uma alternativa seria contratar 400 servidores equivalentes em vários centros de dados e fornecer o *update* aos 400.000 clientes em aproximadamente 20+20 min. Uma terceira alternativa consiste em usar uma técnica P2P e fornecer o *update* aos primeiros 1.000 clientes e depois, enquanto se fornece o *update* a outros 1.000 clientes, cada um dos primeiros 1.000 clientes fornece o *update* a 1 cliente, ou seja, na segunda ronda, 2.000 clientes recebem o *update*, ficando no total 3.000 clientes com o *update*. Na terceira ronda, enquanto o servidor fornece o *update* a mais 1.000 clientes, cada um dos outros 3.000 clientes fornece o *update* a mais um cliente, assim, mais 4.000 novos clientes recebem o *update*, ficando ao fim da terceira ronda um

- total de 7.000 clientes com o *update*. Usando esta técnica P2P quantas rondas no total são necessárias para todos os 400.000 clientes receberem o *update*?
15. Vários operadores (e.g., Google e Open DNS) começaram a oferecer um serviço de *caching only servers* DNS (servidores que aceitam pedidos DNS recursivos). A adopção desta solução tem alguma repercução sobre o funcionamento das CDNs baseadas em *proxies* inversos?
 16. Um servidor de um serviço muito conhecido envia no cabeçalho da resposta HTTP o seguinte *header field* com um *cookie*. Qual o seu papel exacto?
Set-Cookie: GeoIP=PT:14:Lisbon:38.72:-9.13:v4; Path=/; secure; Domain=.
 17. Você está ligado a um serviço de distribuição de livros electrónicos através da Internet. O serviço vai enviar diariamente várias dezenas de milhar de livros, com vários Mbytes cada, para algumas centenas de milhar de clientes espalhados por vários continentes. O serviço pretende-se de grande qualidade, rápido e os clientes são autenticados antes de fazerem um *download* pois o serviço é pago. Pretende-se também diminuir a possibilidade de alguém obter cópias dos livros sem ser cliente e por isso os livros são distribuídos de forma cifrada. Quais ou qual das alternativas abaixo pode suportar todos os requisitos do serviço?
 - (a) Um conjunto de vários servidores Web hospedados num único centro de dados com muito boas ligações, servindo todos os *downloads* por HTTPS.
 - (b) Um conjunto de servidores Web hospedados num único centro de dados com muito boas ligações distribuindo os livros através do protocolo Bit-Torrent. Os servidores do centro de dados funcionavam como servidores de autenticação, sementes e *trackers* e os clientes usam clientes baseados no protocolo BitTorrent.
 - (c) Um conjunto de servidores Web hospedados num centro de dados com boas ligações, que realizavam a autenticação e têm uma cópia de todos os livros, e um sistema de *proxies* inversos baseado num operador de distribuição de conteúdos.
 - (d) Idem (c) mas com uma CDN dedicada.
 18. Considere um cenário em que 50.000 computadores estão a executar simultaneamente o protocolo BitTorrent a fazer o *download* de um ficheiro com 6 Gbytes. O cenário é o habitual em que as ligações à Internet dos diferentes participantes têm uma capacidade de *download* bastante superior à de *upload*. Indique dos factores seguintes quais os que sem qualquer espécie de dúvida aceleram o *download* de todos os participantes, isto é, indique todas as soluções que diminuem o tempo total necessário para garantir que todos os participantes têm uma cópia do ficheiro:
 - (a) Aumentar significativamente a capacidade de *download* dos canais que ligam os diferentes parceiros à Internet.
 - (b) Aumentar significativamente a capacidade de *upload* dos canais que ligam os diferentes parceiros à Internet.
 - (c) Diminuir o número de *seeds* (sementes).
 - (d) Quando um parceiro termina o *download* permanece na torrente durante pelo menos mais 2 horas.
 - (e) Quando um parceiro termina o *download* abandona imediatamente a torrente.
 - (f) Os participantes procuram fazer imediatamente o *download* dos blocos mais abundantes.

- (g) Os participantes só aceitam dialogar com os parceiros que não têm nenhum bloco em comum.
19. Milhares de computadores todos com capacidade de download de 10 Mbps e *upload* de 1 Mbps estão a fazer o *download* simultâneo de um ficheiro com 2 Gbytes, usando o software BitTorrent. Cada computador está ligado a 50 outros, mas em cada momento só mantém trocas ativas de blocos com 4 deles no máximo. Verifica-se sempre que todos os computadores que ainda não têm o ficheiro completo conseguem fazer progresso pois algum dos seus parceiros lhes dá blocos que lhes faltam. Indique, das alternativas seguintes, quais as que diminuem o tempo total necessário para garantir que todos os participantes têm uma cópia do ficheiro:
- Modificar as capacidades dos diferentes participantes para: *download* 20 Mbps e *upload* para 1 Mbps
 - Modificar as capacidades dos diferentes participantes para: *download* 10 Mbps e *upload* para 2 Mbps
 - Modificar as capacidades dos diferentes participantes para: *download* 1 Mbps e *upload* para 10 Mbps
 - Modificar as capacidades dos diferentes participantes para: *download* 5 Mbps e *upload* para 5 Mbps
 - Modificar as capacidades dos diferentes participantes para: *download* 20 Mbps e *upload* para 0,5 Mbps
20. Invente uma forma de usar uma DHT para substituir o sistema DNS. Por exemplo, a chave de pesquisa de um valor poderia ser a concatenação do nome a ser pesquisado com o tipo de pesquisa a realizar (A, TXT, MX, *etc.*), ou outra alternativa se preferir. Compare os prós e os contras da sua solução com a solução realmente usada pelo DNS.

