

Fundamentos de Redes de Computadores

Ilustrados com Base na Internet e nos
Protocolos TCP/IP
(Versão 0.95)

José Legatheaux Martins

Departamento de Informática
da Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

Março de 2017

Endereço do autor:

José Legatheaux Martins
Departamento de Informática
da Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa
Quinta da Torre
2829-516 Monte da Caparica,
PORTUGAL

Email do autor: `jose.legatheaux at fct.unl.pt`

Site do autor: `http://jose.legatheaux.info`

Site do livro: `http://book.legatheaux.info`

Copyright © 2017 - José Legatheaux Martins
Todos os direitos reservados.

Dedicado à Edite e à Inês.

Prefácio

Este livro nasceu da ambição de fornecer um suporte adequado ao ensino e estudo do tema de redes de computadores por estudantes universitários de Engenharia Informática, ou outras pessoas que procuram perceber como as redes funcionam e suportam os sistemas informáticos. Ele destina-se essencialmente àqueles que estudam e praticam a Engenharia Informática e que necessitam de compreender profundamente como funciona uma rede de computadores.

As aplicações informáticas são hoje em dia construídas predominantemente como um sistema formado por componentes distribuídas, que se coordenam entre si para providenciar o serviço final aos utilizadores. O suporte desses sistemas são redes de computadores que permitem a comunicação e coordenação dessas componentes.

A forma como a rede funciona tem um impacto decisivo sobre a arquitectura e o desempenho dos sistemas e aplicações informáticas e influencia a sua arquitectura. Por outro lado, as características dos sistemas informáticos influenciaram decisivamente a forma como a rede é construída, está organizada e funciona. Assim, as necessidades e características dos sistemas e aplicações estão no centro da ação. Entre as necessidades dos sistemas informáticos desempenham um papel primordial o desempenho e a flexibilidade. Por isso a rede tem ela própria de ter um bom desempenho e uma arquitectura flexível e extensível. Entre as características dos dispositivos computacionais, a sua elevada capacidade computacional e a possibilidade de executarem inúmeros algoritmos, influenciaram também decisivamente as redes de computadores, conduzindo àquilo que alguns chamaram a “rede estúpida”.

Durante muitos anos o ponto de vista usado no ensino das redes de computadores foi o de que os sistemas de telecomunicações eram o ponto de partida e influenciavam directamente o que era ou não possível fazer a nível das aplicações. Por isso o funcionamento das aplicações vinha em último lugar, e usava-se predominantemente uma abordagem de “baixo para cima” (*bottom-up*), com uma grande ênfase nos suportes e tecnologias de comunicação que, de alguma forma, acabavam por tudo condicionar. Os protocolos TCP/IP eram introduzidos nessas abordagens apenas como um exemplo entre muitas outros, não como actores principais.

Com o progresso das redes e das suas aplicações, este ponto de vista, que também usei durante vários anos de ensino, tornou-se pouco interessante para os estudantes de Engenharia Informática do Séc. XXI e está desfasado dos objectivos da sua formação. A evolução das próprias redes tornaram obsoletas várias tecnologias de comunicações que preencheram (demasiado ?) espaço nessa formação.

Por volta do ano 2000 James Kurose e Keith Ross introduziram uma outra forma de ensinar redes de computadores Kurose and Ross [2013], baseada numa abordagem que estudava primeiro os níveis superiores das redes de computadores e as aplicações mais conhecidas, numa forma que passou a designar-se por de “cima para baixo” (*top-*

down), que também adoptei no ensino de licenciatura.

A minha reflexão e experiência de mais de 25 anos de ensino do tema levaram-me, no entanto, a considerar que uma abordagem que seguisse estritamente, de “baixo para cima” ou vice-versa, aquilo que se convencionou chamar os níveis OSI (*Open Systems Interconnection*), não era assim tão importante, tanto mais que esses níveis se vão cada vez mais influenciando mutuamente e interpenetrando, e já não são assim tão estanques e determinantes para se perceberem muitas das aplicações modernas das redes (*e.g.*, a distribuição adaptativa de vídeo a pedido sobre HTTP, ...). Por isso, tentar perceber como a rede é usada, não pode hoje em dia deixar de considerar como central a forma como os sistemas distribuídos se organizam para tirarem todo o partido das redes.

Por isso preferi relaxar esse tipo de percurso rígido e seguir uma aproximação mais do tipo “saltar para dentro da piscina”. Mas que obviamente continua a respeitar de alguma forma certas camadas essenciais da organização da rede, mas mais relacionadas com o local onde residem o software e os algoritmos fundamentais para resolverem os problemas. É o que de mais parecido consigo fazer com algo que poderia ser descrito como uma aproximação “dirigida aos problemas”. Por outro lado, as facetas mais ligadas a telecomunicações vão sendo introduzidas na justa medida das necessidades da compreensão do impacto das suas características nos níveis superiores das redes, e não são sequer objecto de estudo per si.

Foi a visão desta necessidade que foi decisiva para começar, e me deu coragem para continuar, a longa tarefa de escrever este livro. Até então, pensava que não valia a pena escrever um livro sobre um tema versado de forma tão brilhante por vários autores de grande qualidade, como por exemplo: [Tanenbaum and Wetherall, 2011; Peterson and Davies, 2012; Kurose and Ross, 2013; Marsic, 2013] e [Stallings, 2013], só para citar os suportes bibliográficos que mais frequentemente usei.

Esta tarefa é um alvo em movimento e que, acredito, não terminou ainda. Por um lado não tive ainda tempo de introduzir o tratamento de questões fundamentais como a segurança e a mobilidade (só parcialmente abordadas de forma transversal em alguns capítulos). Por outro lado seria desejável aprofundar alguns aspetos suplementares em algumas das partes do livro. Finalmente, é inevitável que espero ter a oportunidade de corrigir os erros e as gralhas que ainda possam estar presentes nesta versão.

Utilizações esperadas

A maioria dos capítulos deste livro dão suporte ao ensino e aprendizagem sobre redes de computadores ao nível de licenciatura. Não são usados formalismos ou teorias que possam constituir uma barreira à sua utilização ao nível dos primeiros anos de diferentes contextos de ensino universitário ou politécnico.

De qualquer forma dois factores impedem provavelmente a sua utilização logo no primeiro ano. Por um lado é dada uma ênfase algorítmica ao tratamento dos temas, são apresentados muitos algoritmos e sugerida a realização de programas logo desde a primeira parte, pelo que é necessário algum domínio da programação.

No entanto, o aspecto principal que recomenda a sua utilização numa fase mais avançada da formação, tem a ver com a necessidade de o estudante ser capaz de raciocinar em termos de abstrações, interfaces e de organização de sistemas. Uma aptidão que só se adquire após algum treino e relega o estudo das redes de computadores mais para o final do segundo, ou para o início do terceiro ano de estudos superiores.

Vários capítulos ou partes de capítulos destinam-se a um estudo mais avançado, no final da licenciatura ou no início de um mestrado. No entanto, desse ponto de vista, faltam tratar vários tópicos que, não sendo de investigação, são temas importantes que devem ser abordados a esse nível e que estão ausentes desta versão do livro, nomeadamente: redes baseadas em túneis (*e.g.*, MPLS, LISP e outros), desempenho e optimização da rede, qualidade de serviço, redes de centros de dados, *software defined networking, etc.*

O livro também é adequado para auto-estudo ou para suporte da formação de técnicos de redes de computadores, pois está escrito com bastantes preocupações pedagógicas e de clareza. No entanto, se é bom para a compreensão dos fundamentos e do que “está por detrás da superfície visível”, tem de ser complementado, quando o objectivo é o treino na utilização das tecnologias, com outras referências que contenham exemplos de como na prática as redes são construídas e os equipamentos parametrizados. Sempre que relevante, essas referências bibliográficas complementares também são fornecidas.

Estrutura do livro

Como referido acima, este livro está estruturado em torno de grandes temas ou partes.

Parte I - Introdução – O todo e as partes

Esta parte do livro apresenta uma panorâmica da forma como estão organizadas, quais são as componentes fundamentais e os princípios e principais modelos que estão na base das redes de computadores. O domínio desta parte é imprescindível a qualquer estudante, não obstante incluir alguns aspectos no Capítulo 4 que podem ser dispensados numa primeira abordagem. Ela é constituída pelos seguintes capítulos:

Capítulo 1. Como funciona uma rede de computadores. Este capítulo apresenta uma panorâmica geral do que é uma rede de computadores, como funciona e como permite o funcionamento dos diferentes tipos de aplicações.

Capítulo 2. Canais de dados. Este capítulo apresenta uma introdução aos canais de dados, como podem ser caracterizados qualitativa e quantitativamente, como são construídos, e apresenta ainda um exemplo preliminar do funcionamento concreto de um dos mais populares canais de dados, os canais Ethernet.

Capítulo 3. Comutação de circuitos e de pacotes. Este capítulo mostra porque as primeiras redes de telecomunicações se baseavam na comutação de circuitos e evoluíram posteriormente para a comutação de pacotes. Discute as características das redes de pacotes, como funcionam e como podem ser caracterizadas através de propriedades quantitativas.

Capítulo 4. Princípios, modelos e ferramentas. Este capítulo discute os princípios fundamentais que presidiram ao desenvolvimento das redes de computadores modernas, os principais modelos utilizados para as compreender e estudar, e finalmente discute quais são as ferramentas usadas para desenhar redes concretas, para medir e estudar as suas propriedades, ou ainda para simulá-las ou emulá-las.

Capítulo 5. Programação com Sockets em Java. Apresenta, através de exemplos, uma introdução à interface de programação sistema (API) para aceder aos serviços da rede num sistema de operação moderno em Java.

Parte II - Transferência de dados

Esta parte do livro discute os métodos usados para transferir dados entre dispositivos computacionais ligados através de uma rede, e põe em evidência como esses métodos se devem adaptar ao contexto e às necessidades das aplicações. Trata-se de um capítulo que noutras livros se poderia chamar “Transporte” mas que inclui também facetas aplicacionais que interferem directamente com a transferência de dados. Esta parte é constituída pelos seguintes capítulos:

Capítulo 6. Fiabilidade com base em retransmissão. Este capítulo introduz os métodos e protocolos fundamentais usados para transferir dados com base na retransmissão dos pacotes que se perderam, ou que foram corrompidos pela rede.

Capítulo 7. O protocolo TCP. É o principal protocolo de transferência de dados usado na Internet, e que usa alguns dos métodos e algoritmos apresentados no capítulo anterior.

Capítulo 8. Controlo da saturação da rede. Quando a quantidade de pacotes que entram na rede é superior à que esta consegue encaminhar, a qualidade de serviço extremo a extremo degrada-se. No entanto, se o software dos sistemas em comunicação cooperar, é possível encontrar uma solução do tipo “*win-win*”.

Capítulo 9. Transporte de dados multimédia. Os dados multimédia podem ser transportados com perda de resolução, envolvendo mais intimamente as aplicações no transporte dos mesmos. Por outro lado, são às vezes usados em cenários (*e.g., multicasting*) em que é preferível usar métodos de correção das faltas através do envio de informação redundante.

Capítulo 10. Outros protocolos de transferência de dados. Este capítulo introduz diversas alternativas de protocolos de transporte. Trata-se de um capítulo complementar que usa um estilo mais adequado para um estudante com autonomia para realizar estudos complementares.

Parte III - Aplicações – Protocolos e sistemas de suporte

Esta parte do livro apresenta um conjunto de mecanismos e princípios em que as redes e os sistemas que a utilizam se cruzam e influenciam mutuamente. Ela é constituída pelos seguintes capítulos:

Capítulo 11. Nomes e endereços. Para além de introduzir estes conceitos e as suas variantes, discute os mecanismos usados para realizar o mapeamento de entidades dos diferentes níveis de designação e introduz o DNS (*Domain Name System*).

Capítulo 12. O protocolo HTTP. O protocolo HTTP (*Hyper Text Control Protocol*) é um protocolo genérico de transferência de objectos, do tipo cliente / servidor, que inclui um conjunto de funcionalidades extensíveis para adaptação do mesmo às necessidades dos diferentes tipos de aplicações. É dada bastante ênfase à problemática do *caching*, extensibilidade e implicações da segurança do protocolo.

Capítulo 13. Redes de distribuição de conteúdos. As aplicações de maior escala usadas hoje em dia são suportadas no protocolo HTTP mas também num conjunto de servidores e princípios arquitecturais que conduziram à construção de um novo tipo de redes, ditas redes sobrepostas (*overlay networks*) ou lógicas. O capítulo discute a organização dos principais protocolos e redes de distribuição de conteúdos, quer os baseados em infraestruturas de centros de dados e *reverse caches*, quer os baseados em sistemas cooperativos (P2P). Boa parte deste capítulo não contém material obrigatório numa primeira introdução.

Parte IV - Redes de pacotes

Esta parte do livro discute de forma detalhada o funcionamento interno de uma rede de pacotes. Segue-se o princípio de tentar resolver os problemas de forma o mais simples que possível, passando apenas a soluções mais complexas quando as anteriores deixaram de ser possíveis. Ela é constituída pelos seguintes capítulos:

Capítulo 14. Redes baseadas em canais de difusão. Um canal baseado em difusão permite construir imediatamente a mais simples de todas as redes. O capítulo discute também em detalhe as tecnologias IEEE 802.3 (Ethernet com fios) e 802.11 (Wi-Fi).

Capítulo 15. Encaminhamento com base em inundação. É relativamente fácil fazer uma rede capaz de encaminhar pacotes baseada no algoritmo de inundação. O mesmo também pode ser usado para difundir dados de forma fiável. Depois o capítulo introduz as redes comutadas baseadas em *switches* Ethernet e os algoritmos e protocolos complementares por estas usados.

Capítulo 16. Encaminhamento pelo caminho mais curto. Quando as soluções anteriores não são aplicáveis, é necessário passar a soluções mais complexas. Uma solução de compromisso consiste em usar encaminhamento pelo caminho mais curto, cujos algoritmos são apresentados em detalhe. É feita uma breve referência aos protocolos que usam este tipo de algoritmos.

Capítulo 17. Interligação de redes - protocolo IP. O protocolo IP proporciona uma interface e uma camada de abstração para os diferentes tipos de redes e é a “cola” da Internet. Por cima dele tudo pode mudar, e por baixo dele a diversidade também é a regra geral. Neste capítulo estuda-se este protocolo nas versões 4 e 6, como é organizado o endereçamento numa rede IP e como esta funciona no concreto.

Capítulo 18. Encaminhamento na Internet global. Quando uma rede interliga bilhões de sistemas terminais através de milhões de comutadores de pacotes e envolve milhares de operadores independentes e em competição, mas unidos pelo objectivo de proporcionar conectividade completa, é necessário passar a um nível superior de organização. O capítulo introduz também o protocolo BGP e uma parte razoável da sua complexidade.

Todos os capítulos terminam sempre com duas secções. A primeira chama-se **Resumo e referências** e contém um resumo do capítulo, muitas vezes baseado na repetição das caixas de resumos parciais das diferentes secções do capítulo, a lista dos termos introduzidos, incluindo a sua definição, um conjunto de referências bibliográficas complementares, assim como apontadores para *sites* com informação relevante para o tema do capítulo.

A segunda chama-se **Questões para revisão e estudo** e contém um conjunto de questões para ajuda à revisão da matéria do capítulo e treino da sua compreensão.

Como o livro pode ser usado para ensino e estudo

Numa primeira disciplina de licenciatura de redes de computadores habitualmente uso os materiais correspondentes aos seguintes capítulos:

Parte I - Introdução. Capítulo 1, Capítulo 2, Capítulo 3 e as primeiras quatro secções do Capítulo 4. O Capítulo 5 é introduzido em função das necessidades, durante as aulas laboratoriais.

Parte II - Transferência de dados. Capítulo 6, Capítulo 7, as primeiras três secções do Capítulo 8, e o Capítulo 9 sem a Secção 9.3.

Parte III - Protocolos e sistemas de suporte das aplicações. Capítulo 11, Capítulo 12 e o início do Capítulo 13.

Parte IV - Redes de pacotes. Capítulo 14, as duas primeiras secções do Capítulo 15, Capítulo 16 e Capítulo 17. É feita uma breve referência ao tema do Capítulo 18.

Este conjunto de matérias deve ser complementado com um capítulo sobre segurança caso a mesma não seja coberta noutra disciplina do mesmo curso.

Alguns capítulos contém tópicos que habitualmente apenas trato em disciplinas de mestrado. Nomeadamente os seguintes: as últimas três secções do Capítulo 4, as duas últimas secções do Capítulo 8, a Secção 4 do Capítulo 9. O Capítulo 10. Boa parte do Capítulo 13. A terceira e quarta secções do Capítulo 15 e praticamente todo o Capítulo 18.

Do uso da língua portuguesa e de termos em inglês

Os profissionais que trabalham em redes de computadores têm por hábito, mesmo quando a sua língua de trabalho não é a língua inglesa, utilizarem muitos termos em inglês. Em contrapartida, os académicos portugueses têm por dever fomentar a correcta utilização da língua portuguesa num contexto técnico e científico quando esta é a língua de trabalho.

Este livro está escrito em português e por isso existe a preocupação de utilizar os termos adequados dessa língua para designar os conceitos, mecanismos e dispositivos usados na área de redes de computadores. No entanto, procura-se referir em todos os casos os termos tradicionalmente usados em língua inglesa, para ajudar o leitor a situar-se num contexto em que sejam utilizados predominantemente os termos na língua inglesa.

No entanto, esta opção não é aplicada de forma radical e por isso, quando se descreve um mecanismo muito ligado a uma norma tecnológica que só existe escrita na língua inglesa, ou se refere um conceito para o qual é difícil introduzir um termo em português, optei por usar o termo na língua inglesa impresso em itálico. Em cada caso a opção está justificada.

Por outro lado, sempre que são apresentados extractos de programas ou algoritmos em pseudo-código, dada a utilização de palavras chave em inglês, os comentários e nomes de variáveis também estão em inglês para manter a uniformidade.

O livro utiliza a ortografia do português na sua versão de antes do último acordo ortográfico.

Agradecimentos

Este livro não teria sido possível sem a colaboração de diversos colegas e das instituições universitárias nas quais desempenhei ou desempenho funções.

Uma primeira palavra de agradecimento é devida ao Departamento de Informática e à Faculdade de Ciências e Tecnologia por me terem libertado de lecionar aulas durante dois semestres, o que me permitiu dispor do tempo necessário para escrever a sua primeira versão.

Um grande agradecimento também é devido aos colegas: Henrique João Domingos, João Magalhães, Nuno Preguiça, Paulo Lopes, Pedro Medeiros, Sérgio Marco Duarte e Vitor Duarte, com quem partilhei durante vários anos o desafio de ensinar a estudantes dos cursos de licenciatura e mestrado em Engenharia Informática as matérias de redes de computadores, em diversas disciplinas, primeiro na Faculdade de Ciências da Universidade de Lisboa, e depois na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

Agradeço igualmente a todas as pessoas que contribuíram com observações e revisões de partes deste texto, nomeadamente: Carmen Morgado, Cecília Gomes, Henrique João Domingos, João Leitão, Mário de Almeida, Nuno Preguiça, Paulo Lopes, Ricardo Martins, Sérgio Marco Duarte e Vitor Duarte.

Agradeço igualmente aos autores dos inúmeros clips que utilizei nas figuras e que os colocaram à disposição do público para utilização sem constrangimentos via o site <https://openclipart.org>.

Monte da Caparica, Fevereiro de 2017

José Legatheaux Martins
<http://jose.legatheaux.info>

Conteúdo

I Introdução – O todo e as partes	1
1 Como funciona uma rede de computadores	5
1.1 O que é uma rede de computadores	5
1.2 Interligação de redes - a rede Internet	11
1.3 Divisão de responsabilidades na rede	14
1.4 Aplicações suportadas no protocolo TCP	19
1.5 Aplicações suportadas no protocolo UDP	23
1.6 Interfaces de rede	29
1.7 Organismos de normalização e governação	33
1.8 Resumo e referências	36
1.9 Questões para revisão e estudo	38
2 Canais de dados	43
2.1 Definição de canal de comunicação	44
2.2 Caracterização quantitativa dos canais	48
2.3 Exemplos de meios de comunicação	53
2.4 Detecção de erros	59
2.5 Exemplo – canais Ethernet	65
2.6 Resumo e referências	69
2.7 Questões para revisão e estudo	71
3 Comutação de circuitos e de pacotes	75
3.1 Comutação de circuitos	77
3.2 Comutação de pacotes	81
3.3 Tempo de trânsito extremo a extremo	85
3.4 Como viver com o <i>jitter</i>	91
3.5 Comparação entre os dois modelos	94
3.6 Indicadores de desempenho	95
3.7 Resumo e referências	96
3.8 Questões para revisão e estudo	99
4 Princípios, modelos e ferramentas	105
4.1 Princípios	106
4.2 Segurança e controlo de acesso e de recursos	112
4.3 Neutralidade da rede	115
4.4 Modelos	117

4.5	Análise e avaliação do desempenho	127
4.6	Resumo e referências	132
4.7	Questões para revisão e estudo	135
5	Programação com Sockets em Java	139
5.1	Utilização de sockets UDP em Java	140
5.2	Comunicação multi-ponto – Sockets Multicast	151
5.3	Sockets Java para canais TCP	156
5.4	Resumo e referências	164
5.5	Questões para revisão e estudo	166
II	Transferência de dados	171
6	Fiabilidade com base em retransmissão	175
6.1	Mecanismos de base	176
6.2	Protocolo <i>stop & wait</i>	181
6.3	Protocolos de janela deslizante	184
6.4	Protocolos e máquinas de estado com acções	198
6.5	Resumo e referências	199
6.6	Questões para revisão e estudo	202
7	O protocolo TCP	209
7.1	A interface do protocolo TCP	210
7.2	Descrição do protocolo	212
7.3	Abertura e fecho das conexões	223
7.4	Resumo e referências	231
7.5	Questões para revisão e estudo	235
8	Controlo da saturação da rede	239
8.1	Em que consiste a saturação	241
8.2	Como controlar a saturação	246
8.3	Controlo da saturação no protocolo TCP	250
8.4	Controlo com <i>feedback</i> explícito	259
8.5	Equidade e desempenho do TCP	265
8.6	Resumo e referências	273
8.7	Questões para revisão e estudo	277
9	Transporte de dados multimédia	281
9.1	Codificação de informação multimédia	282
9.2	Transporte sobre TCP e sobre UDP	290
9.3	Tratamento de erros em fluxos multimédia	295
9.4	RTP – <i>Real-time Transport Protocol</i>	300
9.5	Resumo e referências	303
9.6	Questões para revisão e estudo	307
10	Outros protocolos de transferência de dados	309
10.1	Datagram Congestion Control Protocol	310
10.2	Stream Control Transmission Protocol	311
10.3	Multi-Path TCP	313
10.4	<i>Middleboxes</i> e “ossificação” da Internet	314
10.5	Propostas para aprofundamento	316

III Aplicações – Protocolos e sistemas de suporte	317
11 Nomes e endereços	321
11.1 Nomes, endereços e identificadores	322
11.2 O DNS (<i>Domain Name System</i>)	325
11.3 Organização e funcionamento	330
11.4 Resumo e referências	343
11.5 Questões para revisão e estudo	346
12 O protocolo HTTP	353
12.1 Funcionamento	355
12.2 Desempenho	365
12.3 O protocolo HTTP e a Web actual	371
12.4 Resumo e referências	378
12.5 Questões para revisão e estudo	381
13 Redes de distribuição de conteúdos	385
13.1 Servidores <i>proxies</i> de HTTP	386
13.2 Distribuição de carga de acessos HTTP	389
13.3 CDNs com infra-estrutura dedicada	395
13.4 Distribuição P2P de conteúdos	401
13.5 Resumo e referências	411
13.6 Questões para revisão e estudo	414
IV Redes de pacotes	419
14 Redes baseadas em canais de difusão	425
14.1 Requisitos e possíveis soluções	428
14.2 Canais de dados Ethernet (IEEE 802.3)	434
14.3 Canais de dados Wi-Fi (IEEE 802.11)	440
14.4 Resumo e referências	451
14.5 Questões para revisão e estudo	453
15 Encaminhamento com base em inundação	457
15.1 Encaminhamento com base em inundação	458
15.2 Comutação Ethernet (<i>Ethernet switching</i>)	464
15.3 Árvores de cobertura e STP	466
15.4 Virtual Local Area Networks (VLANs)	473
15.5 Resumo e referências	477
15.6 Questões para revisão e estudo	480
16 Encaminhamento pelo caminho mais curto	489
16.1 Encaminhamento: o problema e uma solução	490
16.2 Determinação de caminhos mais curtos	492
16.3 Encaminhamento pelo estado dos canais	497
16.4 Encaminhamento pelo algoritmo Bellman-Ford	507
16.5 Resumo e referências	517
16.6 Questões para revisão e estudo	521

17 Interligação de redes - protocolo IP	531
17.1 A Internet e o endereçamento IP	532
17.2 IP versão 4 e IP versão 6	544
17.3 Encaminhamento de pacotes IP	549
17.4 Protocolos auxiliares do IP	555
17.5 Resumo e referências	565
17.6 Questões para revisão e estudo	568
18 Encaminhamento na Internet global	577
18.1 Os problemas da escala	577
18.2 Sistemas autónomos	579
18.3 Protocolo BGP	589
18.4 Resumo e referências	602
18.5 Questões para revisão e estudo	605
Bibliografia	613

Parte I

Introdução – O todo e as partes

Para se compreender qualquer sistema é necessário começar por perceber qual a função que o mesmo desempenha. Uma rede de computadores serve para que os computadores, ou melhor, os seus utilizadores, possam comunicar, partilhar informação e para se coordenarem.

Mas quando o objectivo é perceber um sistema e ficar a perceber *como* o mesmo funciona, não basta saber *para que serve*: é também necessário conhecer a sua organização e funcionamento interno. Esta é a perspectiva sobre o estudo de redes de computadores que este livro pretende suportar.

Uma rede de computadores, e em particular a rede Internet, é um sistema complexo formado por várias partes que interagem. Antes de podermos começar a estudar cada parte em detalhe, é necessário ter uma primeira visão de conjunto, e elaborar um ou mais modelos mentais, que permitam compreender as peças que formam a rede: para que servem, que forma têm, como se encaixam umas nas outras e como interagem entre si.

Esta primeira parte do livro tem exactamente o objectivo de permitir ao leitor ter esta primeira perspectiva global do que é uma rede de computadores. Os capítulos que a constituem apresentam os conceitos iniciais das redes de computadores, proporcionando um vislumbre do quadro completo.

Para proporcionar uma abordagem introdutória à terminologia e principais conceitos das redes de computadores, começamos por apresentar no Capítulo 1 uma panorâmica geral do que é uma rede de computadores e quais as suas componentes hardware e software. Essa panorâmica inclui igualmente uma breve discussão da organização e funcionamento das aplicações que a utilizam.

Internamente, depois de reduzidas à sua expressão mais simples, as redes são formadas por dois componentes principais: os canais de comunicação e os comutadores de pacotes. O Capítulo 2 é dedicado à apresentação dos canais de comunicação, sua caracterização, propriedades essenciais e apresentação sumária de vários exemplos de tecnologias de suporte dos mesmos.

O Capítulo 3 discute a arquitectura interna de uma rede de computadores e as principais implicações da mesma. Ele começa com uma retrospectiva sobre a forma como as redes de computadores se diferenciaram da solução tradicional em redes de telecomunicações (redes organizadas em torno da noção de circuito de voz) para se tornarem redes de pacotes de dados.

O Capítulo caracteriza em detalhe o modelo de rede de comutadores de pacotes, assim como as implicações do modelo para o comportamento global da rede e a forma como esta suporta os protocolos e aplicações que correm nos computadores que lhe estão ligados.

Como as redes de computadores são sistemas complexos, ter uma visão de conjunto do sistema é fundamental. A essas visões de conjunto de um sistema, naturalmente simplificadas, costumamos chamar modelos do sistema. No capítulo 4 são introduzidos vários princípios e modelos úteis para essa visão de conjunto de uma rede de computadores e da Internet em particular. O capítulo inclui também uma sensibilização à necessidade da segurança, e refere igualmente aspectos não técnicos, com especial realce para os organismos que têm um papel importante na governação das redes e na elaboração de normas tecnológicas sobre redes de computadores.

Finalmente, esta parte termina com o Capítulo 5, que apresenta uma breve introdução à programação de aplicações distribuídas, através do estudo de parte das interfaces disponíveis na linguagem de programação Java, complementando a descrição no Capítulo 1 de como a rede é usada pelas aplicações.

Capítulo 1

Como funciona uma rede de computadores

“Felix, qui potuit rerum cognoscere causas.”
(Feliz aquele que conhece a causa das coisas)

– Autor: *Virgílio, historiador romano*

O objectivo deste capítulo é proporcionar uma visão panorâmica do que é uma rede de computadores e de como a mesma funciona internamente, mostrando a sua estrutura e organização, a divisão de responsabilidades entre componentes e como é que estas interagem. Um segundo objectivo consiste em mostrar como são desenvolvidas aplicações distribuídas que utilizam os serviços disponibilizados pela rede. A Internet e os protocolos TCP/IP são usados como ilustração. O capítulo apresenta uma visão introdutória, mas global e completa, que vai permitir ao leitor situar-se mais facilmente nos capítulos e partes seguintes.

1.1 O que é uma rede de computadores

Uma **rede de computadores** (*computer network*) é um conjunto de computadores que comunicam através da troca de mensagens e se coordenam entre si para proporcionarem serviços distribuídos aos seus utilizadores.

Em tempos que já lá vão, tempos que em termos da evolução das redes poderia ser caracterizado como o tempo “em que os animais falavam”, a maioria dos computadores estavam isolados e trabalhavam apenas para utilizadores que estavam praticamente ao seu lado. Esta situação alterou-se radicalmente. Não só o próprio conceito de computador evoluiu, como quase todos eles se encontram actualmente interligados através de redes.

Com efeito, a evolução da electrónica e a massificação dos circuitos integrados permitiram que o conceito de computador evoluísse e a sua utilização se generalizasse. Hoje em dia existem computadores que se chamam *smartphones*, *tablets*, computadores pessoais, computadores servidores (que podem estar agregados aos milhares em centros de dados acessíveis na “nuvem”), sensores e controladores computerizados de toda a espécie, televisões e rádios com computadores de controlo e comunicação embbebidos, automóveis cheios de computadores e com ligação à Internet, relógios, peças de vestuário e óculos com computadores integrados, *etc.* Quase todos os exemplos de

computadores apresentados estão hoje em dia ligados a redes, pelo menos em parte do tempo. Os computadores e as redes de computadores generalizaram-se a todos os domínios de actividade: económica, militar, artística e de lazer.

Aplicações distribuídas

Uma primeira forma de definir um sistema é indicar para o que ele serve. Um automóvel serve para deslocar pessoas e mercadorias entre dois pontos distintos interligados por uma rede viária. Uma máquina de lavar roupa serve para lavar roupa (mas não loiça por exemplo), etc. Uma rede de computadores também pode ser definida pelos serviços que presta: acesso a informação e conteúdos, suporte de colaboração e discussão, execução de cálculos à distância, entrega de informação a terceiros (como por exemplo quando entregamos uma declaração de impostos a um servidor do Portal das Finanças/Autoridade Tributária e Aduaneira), suporte de vídeo-chamadas telefónicas, acerto do relógio através da rede, aquisição de dados do ambiente, actuação sobre esse ambiente, *etc.*

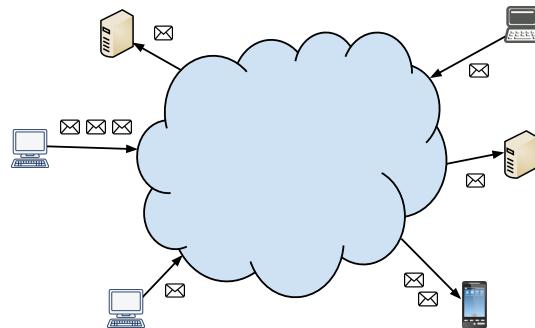


Figura 1.1: Uma rede de computadores

Esta definição funcional não é suficiente em muitos contextos. Para os estudantes de engenharia é necessário ir um pouco mais à essência das coisas. Os computadores executam programas e a rede permite construir uma nova categoria de aplicações, as **aplicações distribuídas** (*distributed applications*), formadas por um conjunto de programas em execução em computadores distintos, e que comunicam e se coordenam entre si através da troca de mensagens. Por exemplo, quando entregamos a declaração de impostos através do nosso computador pessoal, esse computador executa um programa que envia uma mensagem para o servidor do serviço de colecta de impostos, onde é executado um programa que vai analisar a nossa mensagem, e responder com uma mensagem de confirmação de recepção.

Poderíamos discutir se os programas fazem ou não parte da rede de computadores. Eles fazem certamente parte do sistema distribuído que fornece serviços aos utilizadores finais. O software que executa no sistema de operação dos computadores e dá suporte às aplicações distribuídas também faz parte da rede. No entanto, para simplificar, vamos assumir por agora uma definição mais simples, ilustrada na Figura 1.1.

Uma rede de computadores permite que os programas executados pelos diferentes computadores que lhe estão ligados troquem mensagens, tanto para a troca de informação como para a coordenação da sua execução.

A coordenação requer a ordenação dos eventos ligados ao envio e recepção de mensagens. Por exemplo, eu só posso saber se a minha mensagem com a declaração de impostos foi bem recebida depois de o programa no meu computador a ter enviado ao servidor.

Canais e comutadores de pacotes de dados

Os computadores estão ligados à rede através de interfaces para dispositivos, que designaremos por **canais de comunicação** (*communication links*, *data links*, ou **simplesmente links**), os quais permitem a circulação das mensagens. Podemos imaginar os canais como se fossem estradas e os pacotes como veículos que nela circulam.¹

A maioria dos canais de comunicação suporta a circulação de mensagens nos dois sentidos: do computador para a rede e da rede para o computador (como as estradas com dois sentidos). No entanto, como é muito caro e trabalhoso introduzir canais dedicados entre todos os pares de computadores quando o seu número é muito elevado (numa rede com n computadores seriam necessários $n \times (n - 1)$ canais), é necessário algo mais do que canais.

Se fizermos *zoom* sobre a nuvem, ver Figura 1.2, verificamos que os canais estão interligados por equipamentos, ditos equipamentos de comutação, que permitem interligar muitos computadores com um número mais baixo de canais. Recuperando a analogia anterior, podemos imaginar os equipamentos de comutação como os cruzamentos nas redes viárias.

A Figura 1.2 dá uma ideia mais detalhada da rede. Agora já são visíveis os canais e os comutadores. Os comutadores são assim chamados porque executam o encaminhamento de mensagens entre canais. Quando uma mensagem chega, o comutador decide qual o canal pelo qual ela deve continuar até ao seu destino. Esta operação é muitas vezes designada por **comutação de mensagens entre canais** (*message switching*).¹

Por razões que ficarão mais claras no capítulo 2, e que se destinam, no essencial, a tornar mais fácil e eficiente o funcionamento dos canais e dos comutadores de pacotes, a uma mensagem trocada entre os programas que executam nos computadores não corresponde directamente uma mensagem que vai transitar na rede através dos canais e dos comutadores. Na verdade, a nossa declaração de impostos pode conter 300 KBytes mas não transita na rede como uma mensagem única com essa dimensão. Ela transita decomposta em pequenas mensagens, de por exemplo 1 KByte cada uma, enviadas umas a seguir às outras.

A essas pequenas mensagens, que podem transitar internamente pelos canais e os comutadores, chamamos geralmente **pacotes de dados** (*data packets*), e por essa razão os comutadores chamam-se **comutadores de pacotes de dados** (*data packet switches*). Há uma parte do software que executa nos computadores, cujo papel é decompor as mensagens dos programas em pacotes, enviá-los uns a seguir aos outros, agregá-los no destino e recompor a mensagem original antes de a entregar ao programa de destino. É assim como pegar num móvel, decompô-lo em peças, despachar as peças uma a uma, e montar o móvel de novo no destino.

Começamos agora a ter uma ideia mais clara da rede. Nela existem programas distribuídos pelos diferentes computadores que comunicam entre si para executarem aplicações distribuídas.

¹Os termos usados em inglês são *switching* e às vezes *forwarding*. Na língua portuguesa poderíamos usar os termos encaminhar, dirigir ou comutar. Por analogia com o termo *switching* que foi inicialmente o mais usado, continuaremos a usar o termo comutar.

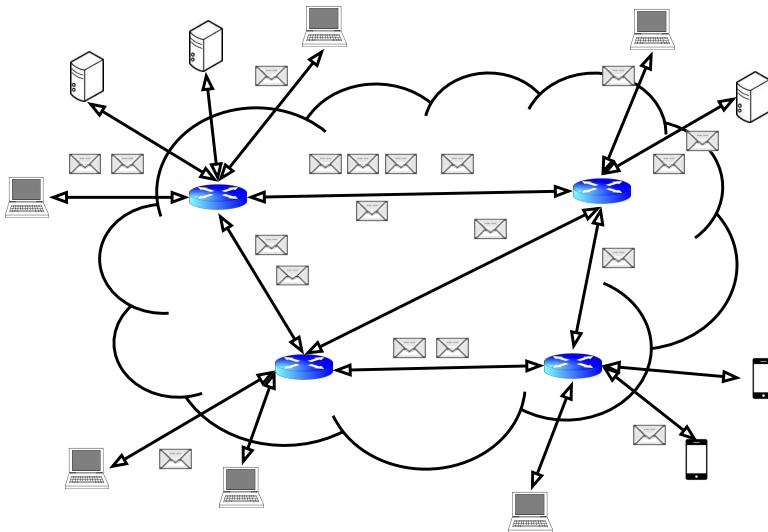


Figura 1.2: A rede de computadores contém canais e comutadores de pacotes

Para suportar essa comunicação, existe um conjunto de software que executa nos computadores e que se apoia nos serviços de uma infra-estrutura de mais baixo nível (no sentido em que os serviços que providencia são mais elementares). Esse conjunto de software permite que os computadores comuniquem através da troca de pacotes de dados. A este nível da rede chamemos por agora o **nível da rede de encaminhamento de pacotes de dados (packet routing network)** ou simplesmente **rede de pacotes (packet network)**.

O protocolo IP

Para que um conjunto de entidades possa comunicar e coordenar-se, é necessário que “falem a mesma língua” e saibam exactamente do que estão a falar. Como os computadores são muito diferentes uns dos outros, foram fabricados por diferentes fabricantes e usam sistemas de operação distintos (por exemplo, uns usam Windows, outros Linux ou Android, outros Mac OS X ou iOS, etc.), é necessário que se ponham de acordo sobre o formato dos pacotes de dados e sobre o significado dos mesmos. A esse conjunto de regras, que envolvem o formato e o significado das mensagens trocadas, chama-se um **protocolo (protocol)**. Para que uma rede de computadores funcione correctamente estabelecem-se protocolos para diferentes funcionalidades e níveis.

Um **protocolo** é um conjunto de mensagens, regras sintácticas e regras semânticas que regulam a comunicação e coordenação de um conjunto de entidades para atingirem um objectivo comum.

Um exemplo é o protocolo que regula a troca de informações entre um *browser* e um servidor de dados da Web. Outro é um protocolo que permite decompor uma mensagem de grande dimensão num conjunto de pacotes de dados e vice-versa.

O sistema de protocolos que suporta o funcionamento da rede Internet designa-se genericamente TCP/IP (mais tarde veremos porquê). Por agora, vamos abordar o

protocolo que lhe serve de suporte, um dos protocolos mais importantes do conjunto utilizado na Internet: o **protocolo IP (Internet Protocol)**, ver o RFC 791².

Este protocolo regula o formato dos pacotes de dados trocados pelos computadores ligados através da rede Internet e por isso esses pacotes chamam-se **pacotes de dados IP** ou simplesmente **pacotes IP (IP packets)**. A Figura 1.3 mostra o formato de um pacote IP.

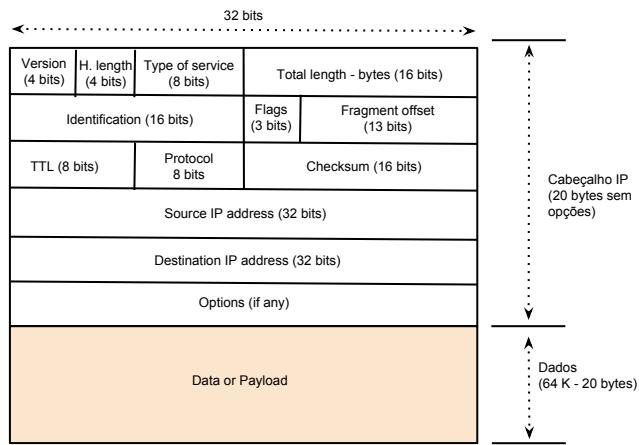


Figura 1.3: Pacote IP - campos do cabeçalho do pacote e parte de dados

Praticamente todas as mensagens trocadas numa rede de computadores estão organizadas em duas partes. Uma parte que se chama o cabeçalho da mensagem e outra que se chama os dados da mensagem. Em inglês é habitual usar os termos *message header*, para designar o cabeçalho da mensagem, e *message payload*, para designar a parte de dados. O cabeçalho contém informação de controlo que permite saber, por exemplo, a quem se destina a mensagem. A parte de dados contém a parte da mensagem com dados para o destinatário. Uma analogia usada frequentemente é a que existe entre o cabeçalho da mensagem e as informações escritas no envelope de correio que transporta uma carta (o nome e endereço de quem envia a mensagem e o nome e endereço do destinatário da mesma, carimbos, *etc.*). O conteúdo do envelope corresponde aos dados da mensagem, isto é, ao *message payload*, já que geralmente os correios facturaram o serviço de encaminhamento de cartas ou encomendas de forma proporcional ao seu peso.

Os pacotes IP, tal como as mensagens de outros protocolos, também estão divididos em cabeçalho e parte de dados. O cabeçalho dos pacotes IP contém várias informações que permitem aos computadores e aos comutadores de pacotes saberem como devem processá-los. Por exemplo, dois dos mais importantes campos do cabeçalho desses pacotes são o endereço origem e o endereço de destino. O endereço origem contém um endereço do computador que envia o pacote IP, e o endereço de destino contém um endereço do computador a quem o pacote IP se destina.

Tal como o formato e o significado dos pacotes IP, também o formato e os significado dos endereços também está normalizado. Na Internet, de acordo com o protocolo IP, os endereços origem e destino dos pacotes dizem-se **endereços IP (IP addresses)**, e correspondem, na versão mais comum do protocolo, a um campo com 32 bits, que especifica a origem, ou o destino, dos pacotes IP. Mais tarde veremos que esses en-

² Na secção 1.7 é apresentada a forma de consultar a série de normas tecnológicas conhecidas pela sigla RFC.

dereços têm uma estrutura que facilita o trabalho dos comutadores de pacotes quando tomam decisões sobre como fazer chegar os pacotes ao destino.

As mensagens usadas pelos protocolos são normalmente estruturadas em duas partes. A primeira diz-se cabeçalho e contém essencialmente informação de controlo. A segunda é a parte de dados e contém os dados transportados pela mensagem.

Assim, daqui para a frente o nosso objecto de estudo serão as redes a funcionar ao nível da troca de pacotes de dados segundo o protocolo IP, redes essas que permitem aos computadores que lhes estão ligados trocarem pacotes IP entre si, com origem e destino indicados através de endereços IP.

Qualidade de serviço do protocolo IP

O protocolo IP especifica que a rede formada pelos comutadores de pacotes, os canais e os computadores origem e destino dos pacotes IP, funciona segundo o **princípio do melhor esforço** (*best-effort principle*). Quer isso dizer que a rede ao nível do protocolo IP (*i.e.*, ao nível da troca de pacotes IP entre computadores) vai fazer o melhor possível para encaminhar os pacotes da origem até ao destino.

No entanto, se no interior da rede ocorrerem problemas que tornem mais difícil fazer chegar o pacote ao seu destino, os equipamentos do interior da rede podem estar programados para não tentarem resolver todos os problemas.

Os problemas que podem surgir chamam-se normalmente erros ou falhas. Quando um sistema está preparado para “corrigir” essas falhas, diz-se que é **tolerante a falhas, mascara as falhas ou compensa as falhas** (*fault tolerant*).

O protocolo IP está definido de tal forma que considera que o nível interno da rede pode não conseguir mascarar integralmente todas as falhas e que, quando recebe um pacote para entregar ao destino, é apenas obrigado a fazer o melhor possível para que o pacote chegue lá, sem oferecer garantia de entrega. A definição do protocolo não prevê sequer que a origem ou destino do pacote sejam notificados da decisão (de entrega ou não).

Poderíamos considerar que nesses casos a rede não cumpriu as suas promessas. Mas isso não é verdade, pois a definição do protocolo IP, que é um contrato entre as partes, não faz promessas, exigindo apenas que a rede faça o melhor possível para entregar o pacote ao destino. Apesar de tudo, trata-se de um contrato aceitável pois, na maioria dos casos, uma rede bem desenhada e dimensionada, ou seja com capacidade para lidar com o tráfego expectável, acaba por entregar a grande maioria dos pacotes ao destino. É como uma rede de estradas onde circulam os automóveis: na ausência de catástrofes, anomalias, e fora das horas de ponta, não existem razões para acreditar que um automóvel não pode chegar atempadamente ao destino por razões imputáveis à rede de estradas. Trata-se de um contrato baseado na experiência prévia e na confiança entre as partes, mas sem cláusulas que implicariam um custo demasiado elevado em situações limite ou anómalias.

Uma rede de pacotes baseada na filosofia do protocolo IP assegura o transporte de pacotes IP entre os computadores que lhe estão ligados, através do encaminhamento dos mesmos pelos canais e comutadores. No entanto, a rede IP não faz promessas que pode não conseguir cumprir, e não garante o transporte dos pacotes em situações limite ou até simplesmente anómalias. É uma rede que se baseia na **filosofia do melhor esforço**. Se a rede estiver bem dimensionada e não existirem situações anómalias, a maioria dos pacotes são entregues ao destino.

Bom, está tudo muito certo, mas será que nós queremos usar uma rede que não faz promessas e até poderia fazer com que os nossos ficheiros ficasse corrompidos durante uma transferência? É claro que não e veremos a seguir como isso pode ser evitado. Mas antes vamos de novo ajustar o grau de *zoom* sobre o interior da rede de pacotes.

1.2 Interligação de redes - a rede Internet

Na secção anterior apresentámos a rede que está dentro da infra-estrutura que interliga os computadores como um conjunto de canais e de comutadores de pacotes, sem mais estrutura. Sugerimos que dentro desse conjunto não existe nenhuma outra estrutura. Esta visão foi propositadamente simplificada. De facto, a maioria das redes não são assim tão horizontais, elas encontram-se também organizadas internamente.

No mundo da Internet e dos protocolos IP essa divisão em redes internas é muitas vezes escondida e a mesma pode ser ignorada em muitas discussões. Para um estudante de redes isso não é aconselhável, pois essa estrutura interna irá revelar-se mais tarde importante para a compreensão real de como funciona a Internet. A seguir vamos esclarecer melhor qual é de facto a estrutura interna dessa “rede” e que sub-redes a formam. A Figura 1.4 dá uma visão idealizada de parte dessa estrutura interna.

Redes de acesso, redes residenciais e redes institucionais

Existem diversos tipos de redes que se distinguem pelo seu objectivo e configuração. Desde logo as redes dos operadores especializados em fazer chegar canais aos particulares e às empresas. Geralmente essas redes são conhecidas como **redes de acesso (access networks)** e são especialmente concebidas para servir o maior número possível de utilizadores residenciais (*i.e.*, que accedem à rede a partir de casa ou de empresas) da forma mais económica e eficiente possível.

Existem outras redes de acesso que são especializadas em fornecer conectividade a computadores móveis. São as redes de acesso dos operadores móveis, que hoje em dia fornecem igualmente canais que ligam os computadores portáteis (incluindo verdadeiros computadores portáteis, *smartphones* e *tablets*) à Internet. Estas redes usam canais que usam o ar como meio de propagação, ou seja, canais sem fios ou *wireless*.

Vulgarizaram-se igualmente as situações em que um utilizador, ou uma empresa, dispõem igualmente de uma rede própria nas suas instalações. Finalmente, as grandes instituições (universidades, empresas, *etc.*) dispõem igualmente de redes próprias, às vezes cobrindo vários edifícios, ou até várias cidades. Essas redes particulares, geralmente designadas **redes institucionais**, são muitas vezes autênticas redes de acesso privadas. A maioria das redes de acesso têm uma predominância de canais que servem para ligar mais e mais computadores à Internet.

Redes de trânsito

Os operadores de redes de acesso são de várias dimensões. Alguns deles fornecem serviços numa região alargada, como por exemplo um país ou um continente. Naturalmente, as diferentes redes de acesso de um operador estão interligadas numa rede que se chama normalmente um **backbone de operador**³. As redes de acesso, mesmo que estejam interligadas com outras redes de acesso do mesmo operador, necessitam também de estar interligadas com as redes de acesso dos outros operadores, assim como com as redes dos operadores de aplicações e conteúdos de que falaremos a seguir.

³ *Backbone* pode ser traduzido por espinha dorsal mas este termo não é usado no campo das redes de computadores no mundo da língua portuguesa.

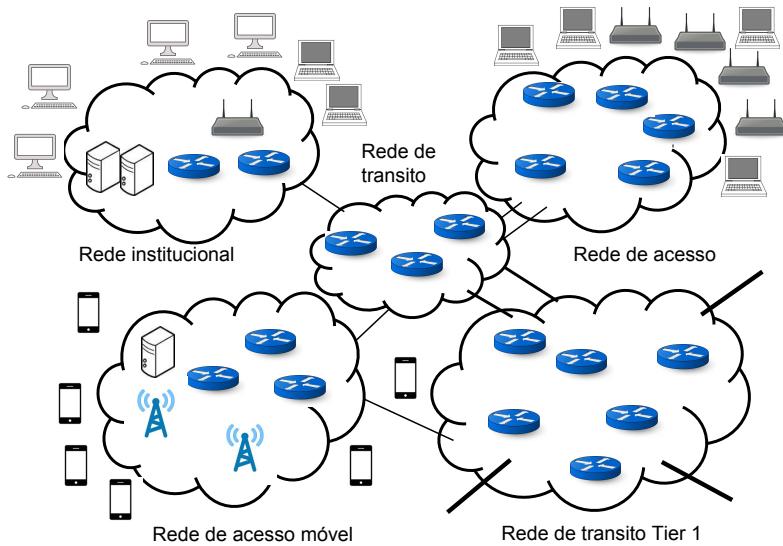


Figura 1.4: A Internet é uma rede de redes

O argumento que serviu para justificar que uma rede, por razões económicas e técnicas, tem comutadores de pacotes, serve também para justificar a necessidade de existirem *backbones* de interligação de outros *backbones* e de redes de acesso. Trata-se de uma espécie de *backbones* de *backbones* que cobrem geralmente uma grande região, como por exemplo um país ou um continente. Assim, alguns operadores especializaram-se em fornecer serviços de interligação a outros operadores através de redes designadas como **redes de trânsito** (*transit networks*). Geralmente, as redes de trânsito caracterizam-se por terem apenas como clientes outros operadores, e os pacotes que as atravessam não têm origem nelas, nem se destinam a elas, estão em trânsito, como sugerido pelo nome.

A existência de redes de trânsito não impede que vários operadores estabeleçam canais directos entre os seus *backbones*. No essencial, a decisão sobre qual a melhor forma de interligar redes distintas segue uma lógica económica.

No topo desta hierarquia estão **redes de trânsito transcontinentais** que interligam *backbones* continentais ou regionais. Na Internet as redes de trânsito intercontinentais, que apenas interligam outras redes, são designadas por **redes Tier 1** pois em inglês “*tier*” quer dizer o nível ou camada e “*Tier 1*” será a camada do topo. As redes *Tier 1* encontram-se ligadas directamente uma às outras já que não faria sentido recorrerem a redes de trânsito de maior gabarito para se interligarem.

Redes de centros de dados

Finalmente, é necessário referir que na Internet se vulgarizaram os chamados centros de dados. Tratam-se de infra-estruturas constituídas por vários milhares de computadores, com elevada capacidade computacional, e instalados de forma compacta em edifícios especializados para optimizar a segurança e as condições energéticas internas ou de arrefecimento, ver a Figura 1.5. A interligação destes aglomerados de computadores é realizada através de redes especializadas de elevada capacidade, que se designam por **redes de centros de dados** (*data center networks*).

Os centros de dados estão ligados, directa ou indirectamente, às redes de acesso, para que os clientes finais possam aceder aos serviços neles providenciados. Com



Figura 1.5: Um centro de dados contém vários milhares de servidores interligados por uma rede especialmente dedicada a esse fim

efeito, os centros de dados são usados para computação de alto desempenho, como por exemplo a análise de grandes volumes de dados, mas também são usados para albergarem serviços fornecidos aos clientes finais ligados às redes de acesso.

Os exemplos mais conhecidos desses serviços são as redes sociais, os serviços de armazenamento de ficheiros e fotografias, o correio electrónico, ou ainda os servidores de difusão de vídeos. Alguns operadores especializados em fornecer serviços acessíveis através da Internet dispõem de centros de dados dedicados privados e de redes privadas que interligam esses centros de dados. Quando tal é o caso, costuma-se chamar a essas redes **redes de operadores de conteúdos**.

A Internet é uma rede de redes

Finalmente, é agora claro que a Internet não é uma rede mas deveria ser antes caracterizada como um conjunto de redes interligadas, como esquematizado na Figura 1.4. Na verdade, se considerarmos apenas as redes geridas pelos operadores e as redes empresariais de dimensão razoável, o número de redes interligadas é superior a uma centena de milhar. Se também considerarmos as redes residenciais, então a Internet é constituída por muitos milhões de redes interligadas.

No entanto, vistas de fora, muitas vezes essas redes internas, ou sub-redes, podem ser ignoradas pelos computadores que estão ligados às mesmas. Quando o computador emissor emite um pacote IP destinado a um computador de destino, existem boas razões para acreditar que esse pacote acabará por ser entregue ao computador de destino, mesmo que tenha de atravessar várias sub-redes distintas, geridas por operadores distintos. É o protocolo IP e a obediência de todos os computadores, comutadores e redes de diferentes operadores ao mesmo, que permite que o conjunto se apresente como uma única rede lógica a que chamamos a Internet.

A Internet pode ser definida como uma rede constituída por muitas redes interligadas, que comunicam através da família de protocolos TCP/IP. Essas redes são muito variadas e incluem redes residenciais, redes institucionais, redes de acesso, redes de provedores móveis, redes governamentais, redes de trânsito locais, regionais e mundiais, redes de centros de dados, redes de fornecedores de conteúdos, *etc.* Essas redes interligam biliões de dispositivos equipados a computadores e usam uma grande variedade de canais de comunicação e comutadores de pacotes.

A tabela 1.1 sintetiza os diferentes tipos de redes a que nos referimos nesta secção.

Tabela 1.1: Tipos mais comuns de redes

Tipo de rede	Caracterização
Rede residencial	Rede que serve para interligar vários computadores numa residência, geralmente dispõe de um canal que a liga a uma rede de acesso
Rede empresarial	Rede que serve para interligar vários computadores numa instituição de dimensão apreciável, geralmente dispõe de um ou mais canais que a liga a uma ou mais redes de acesso
Rede de acesso	Rede que serve para interligar muitos utilizadores, residenciais ou móveis, à rede mais geral
<i>Backbone</i>	Parte da rede de uma instituição ou de um operador que serve para interligar outras redes, geralmente da mesma instituição
Rede de um centro de dados	Rede especializada que serve para interligar (centenas ou milhares) de computadores do mesmo centro de dados
Rede de um operador de conteúdos	Rede que interliga diferentes centro de dados do operador de conteúdos, e directamente ligada a redes de trânsito e de acesso
Rede de trânsito	Rede de interligação de outras redes e que, geralmente, não origina nem é o destino final de pacotes

1.3 Divisão de responsabilidades na rede

Pretende-se fazer o *download* de uma aplicação para executar num computador portátil. A aplicação que assegura esta funcionalidade, geralmente um browser Web, foi programada usando directamente a interface de rede IP? Isso seria o mesmo que programar o acesso a um ficheiro local usando uma interface que permitiria apenas ler blocos de dados gravados no disco do computador. Tal seria muito difícil e até muito ineficaz do ponto de vista do desenvolvimento dos programas. Seria como se tivéssemos regressado ao tempo em que os computadores não dispunham de sistemas de operação com sistemas de gestão de ficheiros, nem houvesse a possibilidade de desenvolver aplicações usando linguagens de alto nível. Todos os sistemas de operação modernos oferecem interfaces de mais alto nível que facilitam a leitura de ficheiros, e essas interfaces estão acessíveis na maioria das linguagens de programação para facilitar a vida dos programadores.

Da mesma forma, todos os sistemas de operação dispõem de interfaces que dão acesso a serviços de rede, de mais alto nível que o do protocolo IP. No entanto, como esses serviços exigem o diálogo com outros computadores e sistemas de operação diferentes, ligados através da rede, essas interfaces de mais alto nível têm de estar normalizadas no que diz respeito ao formato das mensagens que usam e no significado das mensagens trocadas, ou seja, essas interfaces dão acesso a serviços proporcionados por protocolos normalizados de mais alto nível. Só que esses protocolos não são executados pelos equipamentos da rede de pacotes, mas sim pelos sistemas de operação dos sistemas que usam essa rede para comunicar.

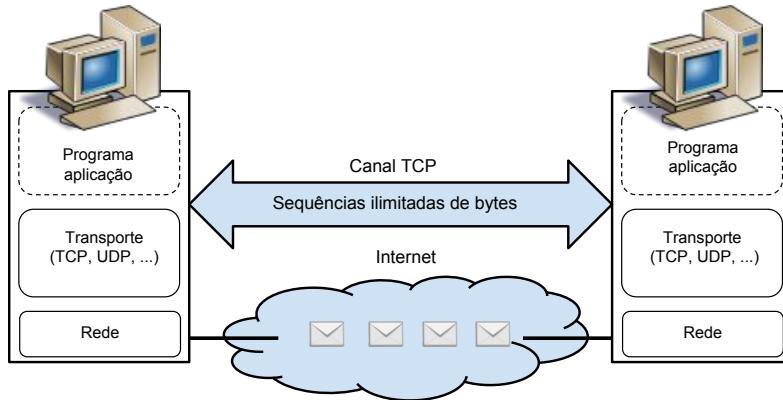


Figura 1.6: Serviços de transporte no sistema de operação - um canal TCP

Deparamo-nos aqui com várias ideias novas importantes, ilustradas na Figura 1.6. Por um lado, sabemos agora que há protocolos de rede que proporcionam serviços de mais alto nível, e com melhor qualidade de serviço, cuja implementação é da responsabilidade dos sistemas de operação dos computadores, e não da rede de pacotes propriamente dita. Ou seja, parte das funcionalidades da rede, tomada no sentido mais geral, são asseguradas pelos próprios computadores. Veremos a seguir que os protocolos deste tipo, que mais próximos estão dos serviços gerais de comunicação de base, chamam-se **protocolos de transporte** (*transport protocols*).

Por outro lado, como para comunicação na rede de pacotes propriamente dita só está disponível o protocolo IP, quer isto dizer que as mensagens trocadas pelos protocolos de mais alto nível têm de viajar obrigatoriamente na parte de dados dos pacotes IP. Chama-se a esta técnica o **encapsulamento** (*encapsulation*) de mensagens de um nível em mensagens do nível mais abaixo. Veremos já a seguir um exemplo concreto.

Exemplo - o protocolo TCP

Um dos protocolos de transporte mais usados nas redes baseadas no protocolo IP é o protocolo TCP (Transport Control Protocol), ver o RFC 793, que permite que dois programas, a executarem no mesmo ou em computadores distintos, estejam ligados por aquilo a que poderíamos chamar um canal TCP. Este canal não é um canal físico entre os dois programas, não passa de uma abstração materializada pelos sistemas de operação, que usam os serviços proporcionados pelo protocolo IP para a realizar, ver a Figura 1.6.

Este conceito não é novo para estudantes de engenharia. Os ficheiros, tal como os sistemas de operação os disponibilizam, não passam de uma abstração construída sobre uma funcionalidade implementada usando discos, ou outros equipamentos similares. Essa funcionalidade, de mais baixo nível, é equivalente a um serviço de acesso a blocos de dados registados numa memória não volátil. Torna-se agora mais claro porque as redes que estamos a usar para ilustrar os conceitos se chamam redes baseadas nos protocolos TCP/IP.

O protocolo TCP disponibiliza canais de dados fiáveis e bidireccionais, que permitem enviar sequências de bytes nos dois sentidos, ver a Figura 1.7. Estes canais permitem a cada uma das extremidades escreverem sequências de bytes no canal que, ao invés de serem escritas num ficheiro, são enviadas para a outra extremidade, que as pode ler como se se tratasse da leitura de um ficheiro. O protocolo trata de assegurar

a fiabilidade desta transmissão de bytes pela rede e não impõe nenhuma limitação à quantidade de informação que pode ser enviada ou lida de cada vez.

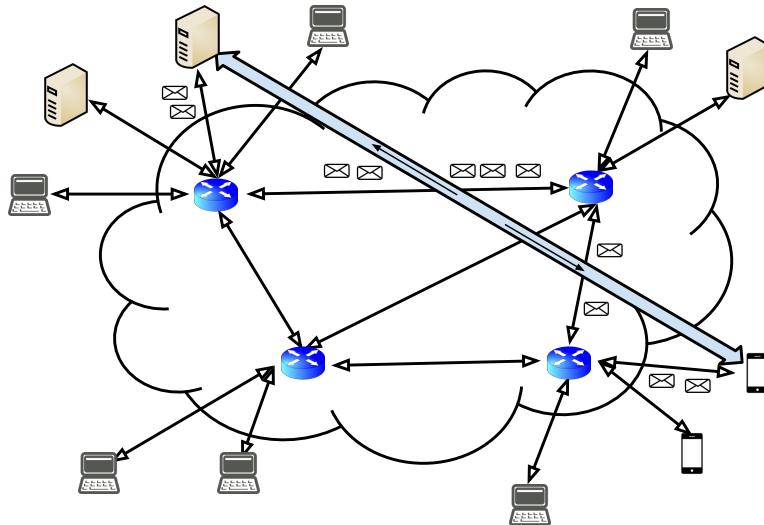


Figura 1.7: Um canal TCP é uma canal bidireccional virtual que liga directamente dois computadores

A interface que dá acesso aos serviços proporcionados pelo protocolo TCP contém primitivas (ou seja métodos ou chamadas) que permitem estabelecer o canal entre as partes. Para mais detalhes ver a Secção 1.6 assim como o capítulo 5. Depois deste canal estabelecido, os programas na sua extremidade podem usar as primitivas *read* e *write* para enviar e receber dados.

O protocolo permite também que existam vários canais lógicos TCP estabelecidos entre programas no mesmo computador. Ou seja, não existem limitações ao número de programas distintos que no mesmo computador têm canais TCP estabelecidos com programas a executarem outros computadores ligados à rede. Os computadores que proporcionam serviços a muitos outros computadores podem ter centenas ou mesmo milhares de canais TCP simultaneamente activos. Algumas vezes, cada um desses canais tem na extremidade um *thread* distinto, do mesmo ou de vários programas activos.

Desenvolver programas distribuídos com base em canais TCP é geralmente mais fácil do que desenvolver programas com base na troca de pacotes IP. Neste último caso, seria a aplicação que assumiria a responsabilidade de verificar se não faltavam dados, e se estes estavam a ser lidos por ordem. As aplicações que usam canais TCP não têm este problema pois os canais TCP são fiáveis e o protocolo assegura que não se perdem dados, e que estes chegam pela ordem com que foram emitidos.

Para garantir esta fiabilidade, o protocolo TCP usa mensagens mais ricas que os pacotes IP. Os dois computadores situados na extremidade do canal TCP trocam mensagens que se chamam *segmentos TCP*. Um segmento TCP tem o formato indicado na Figura 1.8.

Ao analisarmos a Figura vários aspectos podem desde logo ser postos em evidência. O primeiro está relacionado com o facto de que um segmento TCP viaja na rede encapsulado num pacote IP e no cabeçalho do pacote IP encontram-se os endereços IP dos computadores situados na extremidade do canal. A seguir vemos aparecer um

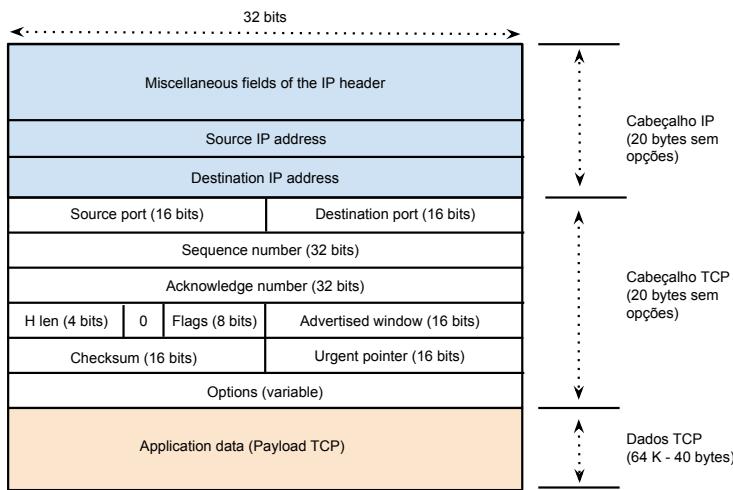


Figura 1.8: Segmento TCP e campos do seu cabeçalho

padrão que já conhecemos: o segmento TCP tem um cabeçalho e uma parte de dados. No cabeçalho do segmento encontram-se as informações de controlo do segmento, na parte de dados viajam uma fracção dos bytes que transitam pelo canal. Ou seja, os dados que circulam no canal são transportados em vários segmentos diferentes, enviados uns a seguir aos outros.

É ao receptor que compete juntar os diferentes segmentos para obter os dados que circulam pelo canal e desta forma não existem limites à quantidade de dados transmitidos. Como os canais TCP são bidireccionais, existem segmentos a circular nos dois sentidos, entre ambas as extremidades do canal.

O protocolo TCP será analisado em detalhe no capítulo 7. Vamos apenas referir aqui alguns campos do cabeçalho que permitem ilustrar alguns aspectos de como o protocolo funciona. Os dois primeiros campos interessantes são as portas origem e destino. Estes campos permitem distinguir os canais TCP num computador, o qual pode ter vários activos em simultâneo.

A distinção dos canais faz-se com base nas portas TCP. Dois canais TCP distintos no mesmo computador, são em geral caracterizados por portas distintas. Assim, um canal é caracterizado não só pelos endereços das extremidades, como também pelas portas. Ou seja, um canal TCP é caracterizado por dois pares, cada um constituído por um endereço IP e uma porta TCP.

A Figura 1.9 ilustra este aspecto. A mesma máquina cliente tem dois canais TCP estabelecidos com uma mesma máquina servidor. Ou seja, existem dois canais a terminarem nos mesmos dois computadores. Um dos canais no servidor está ligado a um servidor Web, cuja porta específica é, por normalização, a porta 80, *i.e.*, a porta reservada para o protocolo HTTP (Hyper Text Transfer Protocol - RFC 2616) que discutiremos a seguir. O outro canal no mesmo servidor está ligado a um servidor de correio electrónico cuja porta específica é, por normalização, a porta 25, *i.e.*, a porta reservada para o protocolo SMTP (Simple Mail Transfer Protocol), ver o RFC 876.

Um outro campo do cabeçalho a que nos vamos referir desde já é o campo número de sequência. Para que serve este campo? A resposta é simples: para assegurar a fiabilidade do canal. Quando os segmentos são enviados, cada um deles recebe um número de sequência crescente.

Para simplificar podemos imaginar que o primeiro segmento tem o número de

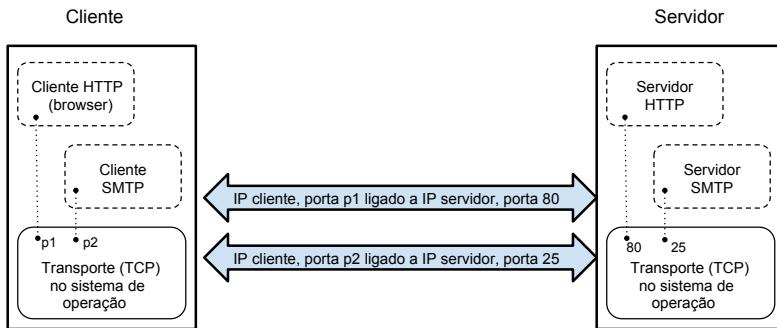


Figura 1.9: Diferentes canais TCP nos mesmos computadores distinguem-se por portas distintas

sequência 1 e contém 1000 bytes, que o segundo tem o número de sequência 1001 e contém 20 bytes, que o terceiro tem o número de sequência 1021 e contém 500 bytes. Que ilustra o exemplo? Que os diferentes segmentos podem ser de diferentes dimensões e levam a indicação da posição dos bytes transmitidos na sequência lógica de bytes transferidos pelo canal (1, 1001, 1021, etc.). Desta forma o receptor do outro lado do canal pode controlar se se perderam dados ou se estes estão fora de ordem. Caso assim seja, pode solicitar à outra extremidade que reenvie os dados em falta, ou pode colocá-los pela ordem certa antes de os entregar à aplicação ligada a essa extremidade do canal.

Este diálogo entre os módulos que implementam o canal TCP nos sistemas de operação dos computadores ligados pelo canal, faz-se trocando segmentos TCP diretamente entre esses computadores. Os segmentos viajam encapsulados em pacotes IP e a rede dos comutadores de mais baixo nível não sabe que esses pacotes contêm segmentos TCP, no sentido em que não precisaria de o saber e isso lhe é indiferente. Ou seja, o protocolo TCP é um protocolo da exclusiva responsabilidade das extremidades. Um protocolo deste tipo diz-se um **protocolo extremo-a-extremo (end-to-end protocol)**. O protocolo IP não é extremo-a-extremo, ao contrário do protocolo TCP.

Um protocolo extremo-a-extremo é um protocolo exclusivamente implementado pelos computadores ligados à rede e que usa a troca directa de pacotes disponibilizada pela infra-estrutura dos canais e comutadores.

Esta arquitectura de protocolos permite pôr em evidência diversos aspectos. O primeiro é que na rede existe uma divisão de responsabilidades entre o centro e a periferia da rede, e que parte das funcionalidades são implementadas exclusivamente nos extremos (nos computadores ligados à rede). No sistema de protocolos TCP/IP usou-se um princípio em que se tenta, sempre que possível, colocar as funcionalidades mais complexas nos computadores na periferia da rede, simplificando as funções desempenhadas pelos comutadores de pacotes. Esta faceta tem-se revelado muito importante para a evolução das redes IP no seu conjunto pois mudar a periferia é mais fácil e mais flexível. Por exemplo, se for introduzido um novo protocolo de transporte, o mesmo tem obrigatoriamente que estar disponível nos sistemas de operação dos dois computadores que querem comunicar usando esse novo protocolo. No entanto, não é necessário modificar o software dos outros computadores ou dos comutadores de pacotes IP da rede.

O outro aspecto ilustrado por este exemplo tem a ver com a organização da rede por níveis. No que toca à transmissão e comutação dos pacotes IP, tudo o que não está nos respectivos cabeçalhos pode ser ignorado.

Os problemas da fiabilidade da transmissão de dados, por exemplo, são da responsabilidade dos protocolos de transporte, ou seja, de um nível superior. Mas o nível de transporte depende dos serviços do nível inferior, o nível da comutação dos pacotes IP, para poder ser implementado. No capítulo 4 serão discutidos diversos princípios e modelos usados para estruturar as redes de computadores.

O software e o hardware das redes de computadores está organizado por níveis ou camadas independentes, em que as camadas dos níveis superiores estão dependentes das funcionalidades providenciadas pelas dos níveis inferiores.

1.4 Aplicações suportadas no protocolo TCP

Começa agora a ser mais claro como é que a rede funciona. No entanto ainda não falámos de como é que a mesma é de facto útil para os utilizadores finais, ou seja, como funcionam as aplicações distribuídas que a utilizam. Já vimos que o protocolo TCP disponibiliza uma interface que permite criar canais lógicos entre programas em execução em computadores distintos, graças à funcionalidade de troca de pacotes IP proporcionada pela infra-estrutura de canais e comutadores de pacotes e ao software TCP executado nos sistemas de operação.

Vamos agora ver como esses canais podem ser usados para criar aplicações distribuídas.

Aplicações cliente / servidor

É frequente estruturar as aplicações distribuídas mais simples em duas partes: o programa cliente e o programa servidor. Essas aplicações dizem-se **aplicações cliente / servidor (client / server)** pois usam o **padrão cliente / servidor** que está ilustrado na Figura 1.10.

Tipicamente, na sequência de uma acção do utilizador, o programa cliente envia uma mensagem para o programa servidor. Essa mensagem designa-se por **mensagem de pedido (request message)**. Quando a mensagem chega ao servidor, o programa servidor analisa-a e deduz qual o serviço solicitado, calcula a resposta e envia uma **mensagem de resposta (reply message)** ao programa cliente. Este interpreta a mensagem de resposta e apresenta o resultado ao utilizador. A Figura 1.11 apresenta um diagrama temporal da execução do padrão.

Exemplo - o protocolo HTTP

O exemplo mais popular de aplicação cliente / servidor é a que proporciona acesso a páginas Web através do protocolo HTTP (Hyper Text Transfer Protocol), ver o RFC 2616, que funciona sobre o protocolo TCP. O cliente é geralmente uma aplicação conhecida como *browser*, que espera que o cliente indique o URL (grosso modo o endereço) do conteúdo a que o utilizador pretende aceder. Quando o utilizador dá o URL ao *browser*, este encontra o endereço do servidor (veremos a seguir como), estabelece um canal TCP com o mesmo (usando como porta do servidor a porta 80 que é a porta normalizada do protocolo HTTP), e finalmente envia-lhe a mensagem com o pedido. Depois de analisar a mensagem de pedido, o servidor responde com a mensagem de resposta, que contém a informação ou o conteúdo solicitado pelo utilizador (que pode ser um texto, uma imagem, um filme, *etc.*). Finalmente, o *browser* mostra ao utilizador a informação que recebeu do servidor.

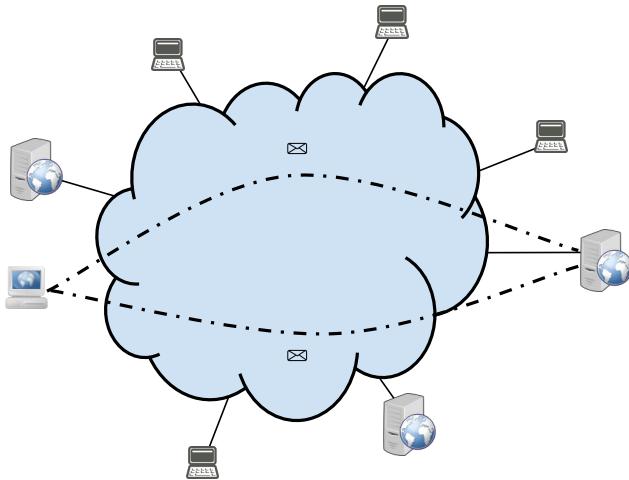


Figura 1.10: O padrão cliente / servidor

A Figura 1.12 ilustra uma hipotética troca de mensagens que obedece ao protocolo HTTP, o qual especifica o formato das mesmas. Ao contrário dos dois outros exemplos vistos atrás (pacotes IP e segmentos TCP), as mensagens do protocolo HTTP estão estruturadas como sequências de bytes, com informação geralmente codificada em código ASCII e facilmente legível. Os diferentes campos das mensagens são muito extensíveis e flexíveis (*i.e.*, existem inúmeras opções), mas essas mensagens não deixam de ter a mesma organização que as dos protocolos anteriores: uma parte de cabeçalho e outra de dados.

O cabeçalho da mensagem de pedido contém na primeira linha o código de operação a executar (`GET` neste caso, para obter um conteúdo alojado no servidor), a indicação do nome do conteúdo pretendido (`/index`) e a versão do protocolo que o cliente pretende executar.

Seguem-se várias linhas com indicações opcionais que podem ser úteis ao servidor. O cliente começa por indicar o nome do servidor específico a que pretende aceder, pois no endereço IP do servidor pode estar um servidor que aloja vários servidores virtuais e é necessário discriminar aquele em que o cliente está interessado (`Host: www.wikipedia.org`). Adicionalmente o cliente indica o seu tipo (`User-Agent:...`).

Cada linha é terminada com dois caracteres de controlo que na figura estão indicados pela sequência `<cr lf>` como abreviatura de `<carriage return line feed>`. Nas mensagens do protocolo HTTP uma linha vazia separa o cabeçalho dos dados. Como neste pedido não são enviados mais dados para o servidor, a mensagem de pedido termina nesta linha vazia, que na figura corresponde à sequência `<cr lf> <cr lf>`.

A mensagem de resposta também contém uma primeira linha especial com o código do resultado da operação (`HTTP/1.1 200 OK` - indicando neste caso que o conteúdo foi encontrado), seguida de várias outras linhas de cabeçalho. As primeiras dessas linhas indicam a hora do ponto de vista do servidor, o tipo do servidor e a data em que o conteúdo foi modificado pela última vez. Após estas indicações seguem-se outras que são imprescindíveis ao cliente, como por exemplo a indicação da dimensão do conteúdo enviado na resposta (`Content-Length: 4768`), e a forma como este está codificado (`Content-Type: text/html`). Finalmente, após a linha em branco `<cr lf> <cr lf>`, que separa a parte do cabeçalho da parte de dados, segue-se o objecto retornado (que

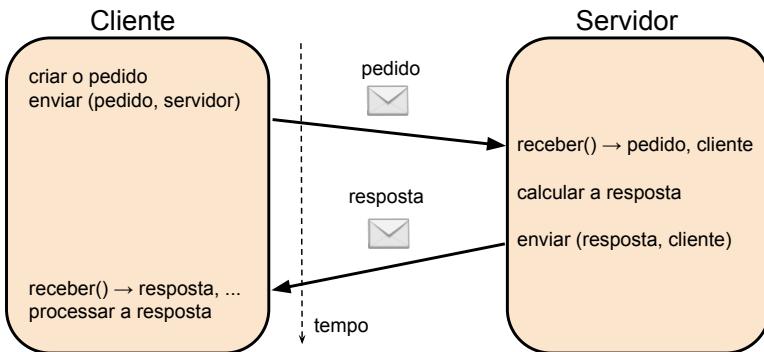


Figura 1.11: Diagrama temporal de execução do padrão cliente servidor

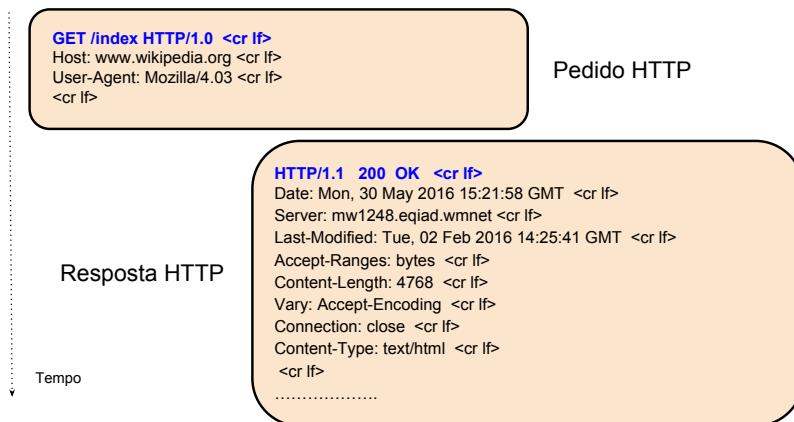


Figura 1.12: Exemplo de mensagens de pedido e resposta do protocolo HTTP

no exemplo está omitido e é representado por vários pontos).

O protocolo HTTP é muito simples e dado que o cliente e o servidor podem trocar entre si qualquer tipo de conteúdos (textos, ficheiros, filmes, código executável pelo *browser*, etc.), é muito utilizado como protocolo de base para construir aplicações distribuídas cliente / servidor em que um *browser* Web é utilizado como cliente genérico.

Aplicações P2P

As aplicações distribuídas do tipo cliente / servidor, como a ilustrada, seguem um padrão simples e fácil de realizar. No entanto, na prática, nem sempre as coisas são assim tão simples, pois muitas vezes existem muitos clientes a tentar obter serviços do mesmo servidor. Nesses casos o servidor pode não conseguir servir atempadamente todos os clientes e o serviço torna-se muito lento e desconfortável.

Uma solução possível consiste em arranjar mais servidores, e uma vez resolvido o problema de distribuir os clientes pelos diferentes servidores, continua-se a usar o

padrão cliente / servidor. É a solução adoptada na maioria dos serviços mais populares na Internet. Para suportarem milhões de utilizadores, geralmente esses serviços são disponibilizados através de vários milhares de servidores. Este tipo de solução só é acessível a instituições muito grandes e com grande poder económico. Quando um conteúdo é popular e não é possível, directa ou indirectamente⁴, cobrar pelo acesso ao mesmo, este é dificilmente acessível de forma confortável, pois será disponibilizado apenas por poucos servidores visto que não há capacidade económica para suportar um melhor serviço.

A seguir vamos ilustrar outro padrão utilizado por algumas aplicações distribuídas e que é particularmente adequado a situações em que há necessidade de acesso massificado a conteúdos imutáveis (*i.e.*, conteúdos contidos em ficheiros que não sofrem alterações, como por exemplo músicas, livros ou filmes).

Nestes casos é comum utilizar outro padrão de estruturação de aplicações distribuídas, conhecido como o **padrão parceiro a parceiro ou P2P** (em inglês diz-se **peer-to-peer**, o que está na origem da abreviatura P2P). Uma motivação comum para a utilização do padrão P2P é a aceleração do acesso a ficheiros imutáveis, transformando os clientes simultaneamente em servidores. Esta faceta, caracterizada por os clientes serem simultaneamente servidores, é a característica mais relevante do padrão P2P.

Exemplo - o protocolo BitTorrent

As aplicações mais conhecidas que o utilizam o padrão P2P usam o protocolo BitTorrent para implementarem o *download* colectivo de ficheiros. De acordo com este protocolo os ficheiros são obtidos por blocos. Um cliente conseguirá aceder a uma cópia válida do ficheiro depois de obter todos os blocos que o formam, o que pode validar, pois antes de começar a tentar obter o ficheiro deverá saber qual o número de blocos total. Como cada bloco contém o seu número, cada participante consegue saber se já tem uma cópia completa do ficheiro mesmo que receba os blocos de forma desordenada.

Assim, as aplicações baseadas no protocolo BitTorrent usam um servidor para obter um ficheiro que descreve as características dos ficheiros a trocar (dimensão, número de blocos, *etc.*) e que servidores usar para se coordenarem com outros clientes⁵. Cada cliente tenta conhecer outros clientes que lhe fornecem blocos que ele ainda não tenha e, à medida que vai obtendo blocos, funciona simultaneamente como servidor fornecendo os blocos que já arranjou a outros parceiros que deles precisem para completar o *download*. A Figura 1.13 ilustra o protocolo em execução.

Não vamos entrar nos detalhes do protocolo BitTorrent, mas o mesmo caracteriza-se por cada participante funcionar simultaneamente como cliente e servidor neste processo de troca de blocos. O protocolo (e as aplicações e serviços que o usam) inclui mecanismos engenhosos para incentivar que cada cliente funcione igualmente como servidor, ou até que a partir do momento em que já conseguiu obter uma cópia integral do ficheiro, passe a funcionar apenas como servidor.

Nas aplicações cliente / servidor em que os clientes apenas fazem o *download* de conteúdos, os canais que ligam esses clientes à rede estão a ser usados sobretudo no sentido da rede para o cliente. A capacidade disponível do cliente para a rede, geralmente designada por capacidade de *upload*, não está a ser usada. O que o protocolo BitTorrent faz com eficácia é conseguir uma utilização mais completa e eficaz dos recursos da rede que estão disponíveis pois, cada cliente, ao funcionar simultaneamente como servidor, contribui para a transmissão mais rápida do ficheiro ao conjunto de

⁴Muitos serviços disponibilizados “gratuitamente” na Internet são afinal pagos por publicidade ou por cedência de privacidade para tornar a publicidade mais valiosa.

⁵ Além desta forma, também é possível utilizar os chamados “magnet links”, da forma `magnet:hash`, e obter a restante informação por comunicação com outros clientes já conhecidos.

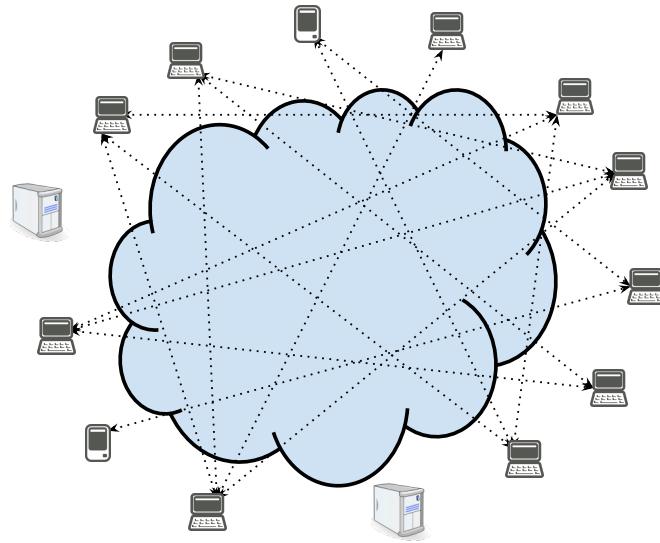


Figura 1.13: O padrão P2P em execução - parceiros a usarem o protocolo Bit-Torrent

parceiros nele interessados. Esse ganho vem do facto de que cada cliente utiliza de forma mais efectiva a capacidade de *upload* do canal que o liga à rede.

Como o protocolo BitTorrent também utiliza o protocolo TCP para a troca dos blocos do ficheiro entre parceiros, a pergunta a fazer é a seguinte: para que dois programas possam comunicar através da rede, a única funcionalidade de nível transporte de dados que está disponível são os canais lógicos TCP? A resposta é não. Existem vários outros protocolos de transporte que seguem lógicas diferentes. Entre esses outros protocolos, o mais conhecido é um protocolo chamado UDP (User Datagram Protocol).

1.5 Aplicações suportadas no protocolo UDP

No protocolo TCP os dois programas que comunicam estão ligados por um canal lógico de dados, fiável e bidireccional, que lhes permite trocarem sequências de bytes, potencialmente ilimitadas, entre eles. Não existe a noção de mensagem pois com TCP, ao nível dos programas utilizadores, os bytes são enviados e recebidos pura e simplesmente em sequência, sem nenhum marcador relacionado com blocos, ou mensagens, ou mesmo de grupos de bytes enviados de uma só vez. Compete aos programas que usam os canais TCP arranjarem os seus mecanismos próprios para definirem onde começam e acabam as mensagens.

Por exemplo, no protocolo HTTP, as mensagens têm um formato especial que permite a ambas as partes reconhecerem onde acaba o cabeçalho e começa a parte de dados, ou ainda qual a dimensão da parte de dados. Nas mensagens de resposta, ver o exemplo na Figura 1.12, existe uma linha em branco que indica onde acaba o cabeçalho e existe um campo no cabeçalho que indica a dimensão da parte de dados.

O protocolo UDP, ver o RFC 768, disponibiliza um serviço diferente do dos canais

lógicos TCP. Ele permite a troca de mensagens bem definidas entre os computadores ligados à rede. Ou seja, se um programa emissor emite uma mensagem composta por um certo número de bytes, o programa receptor recebe de uma só vez uma mensagem exactamente idêntica e com os mesmos bytes.

Assim, à primeira vista, poderia parecer que esta filosofia não justificaria a necessidade de introduzir um protocolo diferente. Afinal o protocolo HTTP é um protocolo aplicacional que usa mensagens de pedido e resposta implementadas pela aplicação sobre canais TCP.

No entanto, surpreendentemente, a funcionalidade que mais distingue o UDP do TCP tem a ver com a ordem e a fiabilidade da transmissão dos dados e não com a delimitação das mensagens. Como o protocolo TCP garante a chegada dos dados por ordem, quando um dado se atrasa, todos os que foram emitidos a seguir também se atrasam pois têm de ser entregues depois. Na verdade, o protocolo TCP é um protocolo complexo que diminui o ritmo de emissão sempre que constata que a rede poderá ter dificuldade em suportar o ritmo actual. O protocolo TCP negoceia e equilibra a fiabilidade com a velocidade com que os dados chegam à outra extremidade da conexão.

No protocolo UDP cada mensagem é independente, e se uma mensagem se atrasar, ou desaparecer, as outras emitidas a seguir não sofrem com isso e podem até chegar primeiro que as anteriores.

Por outro lado, para garantir as suas funcionalidades, o protocolo TCP leva um certo tempo e gasta alguns pacotes de controlo só para estabelecer o canal. O protocolo UDP é mais leve e não requer que ambas as partes estabeleçam qualquer forma de relação prévia: ao emissor basta conhecer o endereço (e a porta) do receptor para poder enviar-lhe mensagens. Finalmente, o protocolo UDP não garante a entrega fiável das mensagens e não avisa sequer se as mesmas não chegarem ao destino.

A Figura 1.14 mostra uma mensagem UDP. Estas designam-se por ***datagrama UDP (UDP datagram)***, por analogia com a terminologia usada num dos primeiros serviços de envio electrónico de mensagens à distância, que usava o termo telegrama para designar uma mensagem.⁶

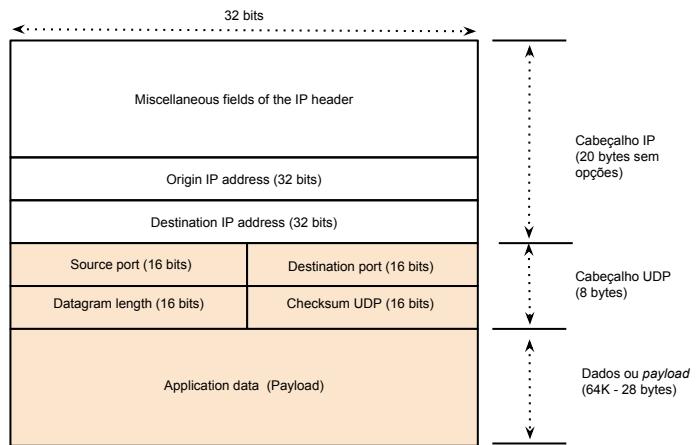
Ao analisarmos os datagramas UDP verificamos de novo que estes viajam na parte de dados de um pacote IP, cujos campos endereços origem e destino indicam a origem e o destino do *datagrama*, que existe um cabeçalho de *datagrama* e que o *datagrama* também tem uma parte de dados. No cabeçalho encontramos de novo dois campos chamados porta origem e porta destino. Na verdade o serviço fornecido pelo protocolo UDP não é mais do que o serviço fornecido pelo protocolo IP, mas os pacotes são agora acessíveis directamente aos programas através das portas, que identificam dentro de cada computador, diferentes origens e destinos dos *datagramas*.

Exemplo - transmissão de som e vídeo em tempo real

O leitor poderia fazer a seguinte pergunta: mas se desenvolver aplicações distribuídas com base na troca directa de pacotes IP é difícil e pode até revelar-se penoso e pouco eficaz, é interessante fazer aplicações distribuídas com base nas funcionalidades do protocolo UDP? A resposta é afirmativa em pelo menos dois tipos de situações.

A primeira situação é a das aplicações que se baseiam na troca de grandes sequências de mensagens mas que preferem perder algumas das mensagens (desde que não seja uma grande maioria delas) a atrasarem todas as que forem enviadas a seguir. As aplicações mais comuns deste tipo são as aplicações de transmissão de informação multimédia (som ou vídeo) em tempo real.

⁶O serviço do telégrafo, e mais tarde o de telegramas, foi ultrapassado pelo serviço de FAX, e mais recentemente pelo serviço de correio electrónico. Assim já não existem telegramas mas continuam a existir *datagramas*. A palavra é usada abusivamente neste livro como um neologismo, que não é reconhecido pelos dicionários portugueses normais, e por isso quando isolada é escrita em itálico.

Figura 1.14: Formato de um *datagrama* UDP

Estas aplicações toleram alguma perda de pedaços de informação intermédia (cuja perda provoca deficiências, às vezes imperceptíveis, no som ou na imagem do receptor) do que receberem a informação com um atraso que a tornasse imperceptível (o som ou a imagem ficam deformados, ou os interlocutores não se entendem).

É esta razão que justifica que as aplicações com requisitos estritos sobre o tempo que as mensagens levam a circular pela rede possam ser concebidas recorrendo ao protocolo UDP. Dado não tolerarem os potenciais atrasos introduzidos pelo protocolo TCP, alguns jogos de computador distribuídos também usam o protocolo UDP para transmitirem as jogadas dos jogadores.

O segundo exemplo de aplicações distribuídas em que se pode recorrer ao protocolo UDP são aplicações do tipo cliente / servidor, em que o cliente e o servidor fazem uma única interacção na qual trocam poucas mensagens de reduzida dimensão (contendo algumas centenas de bytes por exemplo). Nestas aplicações, o problema da fiabilidade pode ser resolvido se as **operações forem idempotentes** (*idempotent operations*), isto é, se as mensagens do cliente para o servidor podem ser reenviadas, de novo interpretadas e executadas sem problema. Se um cliente não receber a resposta a uma pergunta, não é suficiente repeti-la? E se recebeu a resposta, então também tem a certeza que o servidor recebeu o seu pedido.

Se a interacção entre o cliente e o servidor se resumir a esta breve troca de mensagens, talvez não compense pagar o preço de estabelecer um canal TCP, sobretudo se o cliente tem de fazer muitas interacções deste tipo com muitos servidores diferentes. O serviço DNS, apresentado a seguir, baseia-se nestas hipóteses e utiliza um protocolo cliente / servidor geralmente executado directamente sobre o protocolo UDP.

Exemplo - o serviço DNS

Até ao momento quando precisámos de designar um programa a executar num servidor foi sugerido que os protocolos UDP e TCP se baseiam nos endereços IP para designar as partes em comunicação. De facto assim é, mas como todos os utilizadores sabem, geralmente os endereços usados pelos utilizadores têm a forma de mnemónicas⁷, i.e., nomes com algum significado.

⁷Uma mnemónica é uma técnica de codificação, ou uma simples associação a símbolos ou ideias, que permite memorizar mais facilmente algo mais complexo.

Por exemplo, supondo que existia um banco designado “Banco de Depósitos Muito Seguros”, com a sigla **bdms**, seria natural que o seu site Web estivesse acessível por um URL como por exemplo <http://bdms.com>. Existem inúmeros outros exemplos bem conhecidos do dia a dia. Esta técnica é muito rica pois é mais fácil memorizar a mnemônica do que um número inteiro com muitos dígitos (o endereço IP). Adicionalmente, também permite uma maior flexibilidade. De facto, o nome mnemônico de um serviço pode ser sempre o mesmo, mas o endereço pode mudar (pois arranjou-se um servidor novo), ou ser diferente para diferentes clientes (poder-se-ia associar um endereço IP diferente conforme o país do cliente por exemplo), *etc.*

Mas como é possível usar estes nomes mnemônicos quando as comunicações pela rede vão ter lugar entre interlocutores designados por endereços IP e portas? É o serviço DNS (Domain Name System), ver o RFC 1034, que permite, entre outras funcionalidades, traduzir nomes mnemônicos em endereços IP.

O serviço DNS é uma base de dados distribuída, contendo entradas para vários milhões de nomes mnemônicos hierárquicos, que permite estabelecer relações entre nomes e atributos ou propriedades. O DNS pode, logicamente, ser assimilado a uma gigantesca tabela, com um nome por linha, e tantas colunas quantos os atributos que é possível associar-lhe. O principal atributo associado à grande maioria dos nomes é constituído por um ou mais endereços IP.

A base dados está organizada em domínios hierárquicos, e distribuída por muitos servidores. Cada servidor apenas conhece uma parte da base dados, geralmente um conjunto de linhas associadas a nomes com o mesmo sufixo, que se chama um domínio DNS.

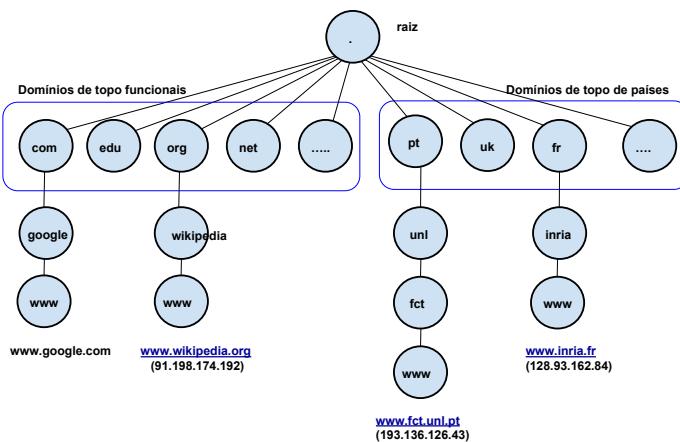


Figura 1.15: Uma parte da árvore de domínios do DNS e exemplo de alguns nomes

O serviços DNS é outro exemplo em que a complexidade está relegada para a periferia pois a rede dos canais e dos comutadores não conhece nomes mnemônicos, apenas conhece endereços IP, tal como os protocolos de transporte.

Os nomes mnemônicos usados pelo DNS estão organizados por domínios, ver a Figura 1.15. Estes são hierárquicos, com uma raiz da hierarquia por baixo da qual estão nomes bem conhecidos (*e.g.*, **com**, **net**, **org**, **edu**, **info**, **uk**, **eu**, **pt**,

...) associados a propriedades do domínio (*e.g.*, `edu` para universidades, `com` para empresas) ou a países (*e.g.*, `pt` é o domínio de Portugal e `fr` o da França). Continuando a descer na hierarquia aparecem domínios subordinados (*e.g.*, `un1.pt` é o domínio da Universidade Nova de Lisboa, `fct.unl.pt` é o domínio da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa). Os nomes lêem-se da esquerda para a direita como é habitual, e usando esse sentido de leitura vai-se subindo na hierarquia. A Figura 1.15 ilustra um pequeno subconjunto da hierarquia do DNS.

Associado a cada domínio da hierarquia existem vários servidores que conhecem os nomes subordinados ao domínio, *i.e.*, terminados no nome do domínio, como por exemplo o nome `www.wikipedia.org` que é formado pelo nome `www`, concatenado com o nome do domínio `wikipedia.org`. Adicionalmente, se um domínio não é o fim do seu ramo da hierarquia, no domínio também estão registados os sub-domínios e os endereços IP dos respectivos servidores. Por exemplo, no domínio `.org` estaria registado o domínio `wikipedia.org` e o endereço IP dos servidores DNS desse domínio.

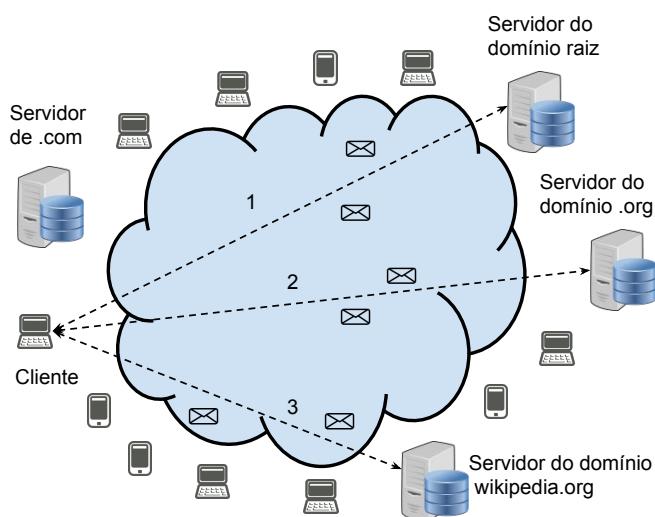


Figura 1.16: Pesquisa do endereço IP de `www.wikipedia.org`

Quando se pretende conhecer o endereço IP associado a um nome, é necessário dirigir a questão para um servidor do respectivo domínio. Mas como saber o endereço IP dos servidores dos muitos milhões de domínios existentes? Admitindo que se conhecem os endereços IP dos servidores do topo da hierarquia (*i.e.*, os chamados servidores da raiz, ou servidores de *root*), é possível descer na hierarquia até um servidor que conhece o domínio do nome que procuramos. Como já referimos, para cada domínio, incluindo a raiz, existe sempre mais do que um servidor, pelo que o DNS pode continuar a ser consultado mesmo que alguns servidores estejam momentaneamente inacessíveis, o que confere uma grande robustez ao serviço.

Por exemplo (ver a Figura 1.16), admitindo que se procura o endereço IP do servidor designado por `www.wikipedia.org`, é possível perguntar a um dos servidores da raiz qual é esse endereço (a troca de mensagens 1 na figura). Este responderá que não o conhece, mas indicará os endereços IP dos servidores do domínio `.com`. Repare-se que os servidores da raiz conhecem necessariamente o endereço dos servidores de `.org` pois `.org` é um domínio subordinado da raiz. Escolhendo um dos servidores de `.org`, é possível perguntar-lhe de novo qual é o endereço de `www.wikipedia.org`. A

troca de mensagens 2 na figura. O servidor responderá novamente que não o conhece, mas conhece os endereços IP dos servidores do domínio `wikipedia.org`. Tal permite finalmente chegar a um servidor que poderá responder à questão: qual é o endereço IP de `www.wikipedia.org`? A troca de mensagens 3 na figura.

Para que os programas que executam nos computadores ligados à Internet possam consultar o DNS, o sistema de operação fornece uma interface especial, designada por *resolver interface* (de resolução de nomes), por detrás da qual é executada uma troca de mensagens obedecendo ao protocolo DNS. Nada impede um computador de executar o protocolo do DNS directamente, executando a sucessão de questões que acabámos de ilustrar. Por exemplo, a aplicação `dig` dos sistemas Unix ou semelhantes (Linux, Android, Mac OS/X, iOS, ...) executa o protocolo descrito. No entanto, geralmente os computadores limitam-se a consultar um servidor DNS, designado *caching only server* ou *local name server*, como ilustrado na Figura 1.17. Estes servidores não são responsáveis por nenhum domínio, mas sabem executar o protocolo que descrevemos no exemplo anterior.

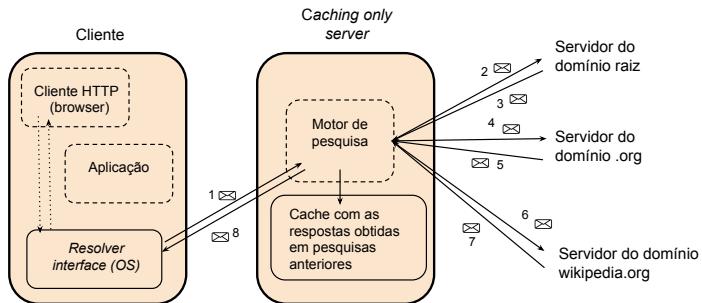


Figura 1.17: Pesquisa do endereço IP de `www.wikipedia.org` através de um *caching only server* - a numeração das mensagens segue a ordem de execução do protocolo

A vantagem de se utilizarem servidores *caching only* está ligada ao facto de que eles podem guardar as respostas que vão obtendo numa *cache*. Desta forma podem responder a questões futuras idênticas muito mais depressa pois já conhecem a resposta. Geralmente os diferentes operadores de rede e as redes institucionais têm servidores DNS *caching only* cujos endereços IP são fornecidos aos computadores que lhes estão ligados.

O protocolo do DNS é na grande maioria das situações executado pelos computadores dos utilizadores como um simples protocolo cliente / servidor dirigido aos *caching only servers*. Estes executam a seguir um protocolo do tipo cliente / servidor iterativo até obterem as informações solicitadas, e no fim respondem aos clientes iniciais. Se a resposta às questões recebidas estiverem na sua *cache*, eles respondem imediatamente, sem necessidade de executarem o processo iterativo de consulta de outros servidores DNS.

O protocolo pode ser executado sobre TCP ou UDP, mas dada a sua natureza, e a quantidade de informação trocada em cada mensagem, geralmente é executado sobre UDP. Caso um cliente não obtenha resposta a uma questão, pode enviá-la de novo. Pode também enviá-la a um servidor distinto que é suposto também saber a resposta, ou pelo menos parte dela, para poder continuar a iterar até obter a informação pretendida. No limite, um cliente até poderia dirigir a questão a vários servidores simultaneamente, para tentar obter a resposta do mais rápido a responder.

Para fecharmos esta discussão sobre os protocolos ditos de transporte e a sua

relação com o tipo de aplicações que melhor suportam, apresenta-se a seguir uma caracterização sintética do protocolo TCP.

O protocolo TCP é um protocolo *end-to-end* de transporte orientado à conexão, pois requer o estabelecimento prévio de um canal lógico entre as partes.

O canal TCP é bidireccional e transporta sequencialmente bytes nos dois sentidos.

O canal TCP é fiável e ajusta a velocidade de transmissão à capacidade do receptor, o que se designa por **controlo de fluxo**, ver o Capítulo 7, e da rede, o que se designa por **controlo de saturação**, ver o Capítulo 8.

O TCP não garante a capacidade do canal nem o limite superior do tempo necessário para a entrega dos dados ao receptor.

O TCP é um protocolo adequado para o envio fiável de quantidades de dados de dimensão apreciável sem constrangimentos temporais fortes.

seguida da caracterização sintética do protocolo UDP.

O protocolo UDP é um protocolo *end-to-end* de transporte que permite trocar *datagramas* (mensagens) entre programas.

Não requer o estabelecimento prévio de qualquer ligação entre as partes que vão comunicar.

Não garante a fiabilidade nem a ordem de entrega das mensagens e não faz controlo de fluxo ou de saturação.

Oferece a mesma qualidade de serviço que a própria rede (melhor esforço) e portanto exibe os defeitos e qualidades desta em termos de fiabilidade e de tempo de trânsito das mensagens.

O UDP é um protocolo adequado para interações curtas do tipo das do DNS ou quando não se requer fiabilidade, nem se toleram atrasos na transferência dos dados.

Ambos os protocolos designam as partes em comunicação através de endereços IP e portas, mas o serviço DNS permite associar nomes mnemónicos aos endereços IP para evitar que os utilizadores os tenham de memorizar.

Para fecharmos esta primeira visita guiada ao mundo das redes vamos ver a seguir que aspecto têm as interfaces dos computadores e dos programas com a rede.

1.6 Interfaces de rede

Geralmente os computadores estão ligados através interfaces de rede e canais aos primeiros comutadores de pacotes que lhes dão acesso ao resto da rede. Essas interfaces de rede são baseadas em componentes electrónicos (e às vezes também ópticos) e contêm dispositivos de ligação aos meios de transmissão que suportam os canais (antenas nos canais sem fios e fichas para fios ou fibras ópticas nos restantes casos). A maioria

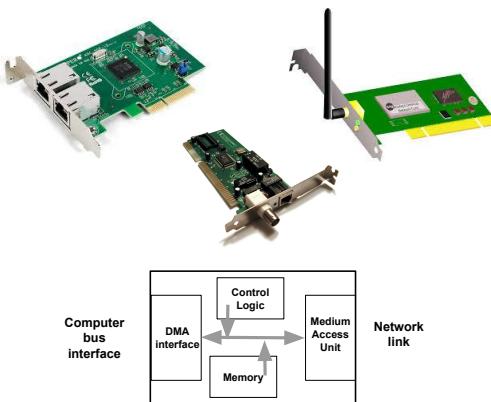


Figura 1.18: Interfaces de rede para computadores (separadas das placas mãe) e um diagrama esquemático das suas componentes

dos dispositivos móveis actuais têm as interface e antenas integradas na caixa, como os *smartphones* por exemplo.

Muitas vezes usamos o termo placas de rede (*networking cards*) como nome informal dessas interfaces, ver a Figura 1.18, mas estas cada vez mais se encontram directamente embbebidas nas placas mãe (*motherboards*) dos computadores, como é o caso corrente em todos os computadores pessoais, *smartphones*, *tablets*, etc. Na transmissão, estes dispositivos transformam sequências de bytes (pacotes de dados) no formato e representação adequados à sua transmissão pelo canal, e efectuam a transformação inversa na recepção, ver o Capítulo 2.

Os programas não acedem directamente aos pacotes de dados, nem às interfaces físicas dos canais, pois vêm os serviços da rede mediados pelo sistemas de operação, através de interfaces programáticas que dão acesso aos serviços proporcionados pelos protocolos de transporte (e.g., TCP e UDP).

Uma das primeiras interfaces deste tipo para o mundo TCP/IP foi desenvolvida para o sistema Unix na Universidade de Berkeley, na Califórnia, sob contrato do governo dos Estados Unidos. O contrato estipulava que a interface, assim como o código fonte da sua implementação, seriam públicos e poderiam ser integrados nouros sistemas. Por esta razão, esta interface, disponibilizada pela primeira vez em 1983, acabou por ser transportada ou adoptada por quase todos os sistemas de operação e é, hoje em dia, praticamente universal. A interface ficou conhecida pelo nome *sockets interface*⁸.

Actualmente a interface dos sockets está normalizada e integrada na norma tecnológica POSIX⁹, a norma de sistemas “a la Unix”, implementada pela maioria dos sistemas UNIX ou semelhantes (Linux, Mac OS/X, Android, iOS, ...). Mas a interface dos sockets também está disponível nos sistemas Windows. No conjunto de todos estes sistemas o comando `netstat` permite conhecer os sockets activos.

A interface de sockets utiliza vários conceitos nossos conhecidos como: endereços IP, portas, *datagramas* UDP, canais TCP, etc. mas o conceito central é o de “socket”, que foi a inspiração para o nome da interface. Um socket é um meio de ligação à rede, como uma tomada eléctrica é um meio de ligação à rede eléctrica. Trata-se de um ponto

⁸Em inglês socket pode ser traduzido por “tomada” e o termo aplica-se normalmente a tomadas eléctricas de parede e aos suportes de encaixe de circuitos.

⁹A norma POSIX corresponde ao Standard IEEE 1003.1, desenvolvido em conjunto pelo IEEE e o The Open Group, e pode ser consultada *on-line* a partir dos endereços das duas instituições: <http://standards.ieee.org> ou <http://pubs.opengroup.org>.

através do qual um programa acede aos serviços da rede para comunicar e constitui aquilo que se designa como uma “extremidade de comunicação” (*communication endpoint*). No entanto, ao contrário dos aparelhos eléctricos, um programa pode usar vários sockets simultaneamente.

Os sockets podem ser de dois tipos: UDP ou TCP. Um socket UDP é caracterizado por um endereço IP (do computador que executa o programa que possui o socket) e por uma porta UDP, e não precisa de estar ligado a nenhum interlocutor. Um socket deste tipo pode receber e enviar *datagramas* UDP para qualquer outro socket UDP no mesmo ou outro computador, bastando para isso conhecer o endereço IP e porta do socket de destino.

No fundo, um endereço IP e uma porta UDP activos (que podem receber e enviar *datagramas*) estão associados necessariamente a um socket UDP de um programa em execução num computador ligado à rede naquele endereço IP. Como o *datagrama* recebido tem a porta UDP de origem no seu cabeçalho e vem encapsulado num pacote IP com o endereço IP de origem no respectivo cabeçalho, o receptor pode responder ao emissor do *datagrama*.

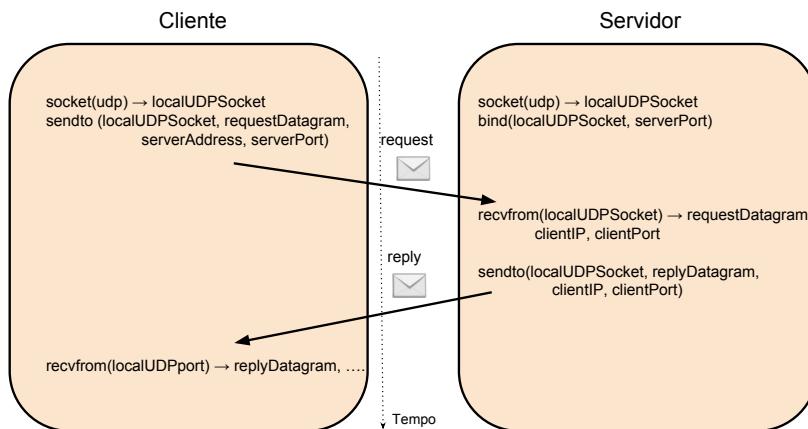


Figura 1.19: Comunicação com sockets UDP

A Figura 1.19 mostra dois programas, um cliente e um servidor, a comunicarem através de sockets UDP para trocarem *datagramas* entre si. A seguir vamos dar uma primeira ideia em pseudo código das entradas na interface de sockets, isto é de alguns dos seus *system calls*, tal como ela foi definida e é acessível (tipicamente usando a linguagem C) nas camadas mais baixas dos sistemas de operação (*i.e.*, ao nível de *system calls*). Veremos depois que existem muitas outras interfaces, de mais alto nível e disponíveis em muitas linguagens diferentes, para aceder às funcionalidades desta interface, como por exemplo em Java, ver o Capítulo 5.

Ambos os programas começam por criar um socket UDP. Na interface dos sockets esta chamada chama-se exactamente `socket()` e permite criar diferentes tipo de sockets. Um socket UDP usa geralmente uma porta seleccionada dinamicamente pelo sistema, mas o programa também pode decidir associar-lhe uma porta à sua escolha através da chamada `bind(socket, port)`. Um servidor é obrigado a usar esta funcionalidade, senão os seus clientes não saberiam qual era a sua porta pois a mesma teria um valor qualquer escolhido pelo sistema. O cliente pode usar o DNS para conhecer o endereço IP, mas (infelizmente?) o DNS não contém a porta dos serviços, pois estas estão normalizadas para os serviços principais. No exemplo da figura o servidor

executa as chamadas:

```
socket(udp) → localUDPSocket
bind(localUDPSocket, serverPort)
```

Depois de criado o socket, um cliente geralmente envia um *datagrama* com um pedido para o servidor e para tal utilizará a chamada **sendto()**:

```
sendto(localUDPSocket, requestDatagram, serverIPAddress, serverPort)
```

para enviar, através de um seu socket local, um *datagrama*, para um socket remoto, identificado por um endereço IP e uma porta. Para receber um *datagrama*, um programa deve usar a chamada **recvfrom()**. Por exemplo, para receber o pedido do cliente, o servidor executa:

```
recvfrom(localUDPSocket) → requestDatagram, clientIP, clientPort
```

que devolve o *datagrama* recebido e a identificação do emissor do mesmo (na verdade essa identificação vem com o *datagrama*). Esta chamada bloqueia o programa até que um *datagrama* esteja disponível, pois associada a cada socket UDP existe uma fila de espera de *datagramas* recebidos e à espera de serem consumidos através de chamadas a **recvfrom()**.

Os sockets TCP também são criados através da chamada **socket()** e podem ser associados a uma porta TCP específica usando também a chamada **bind()**. No entanto, a comunicação através de dois sockets TCP só pode ter lugar depois de estes terem sido ligados através de um canal TCP. A Figura 1.20 mostra um exemplo de comunicação entre um cliente e um servidor com base em sockets TCP. O servidor fica à espera que um cliente estabeleça uma comunicação pois é aos clientes que compete tomar a iniciativa.

```
socket(tcp) → localTCPsocket
bind(localTCPsocket, port)
accept(localTCPsocket) → newTCPsocket
```

No exemplo, o servidor criou um socket, associou-lhe a porta acordada com os clientes e ficou à espera que um deles estabeleça uma conexão invocando:

```
accept(localTCPsocket)
```

Esta chamada bloqueia o servidor até que um cliente desencadeie uma conexão. Quando essa conexão se estabelecer, um novo socket é criado (**accept()** retorna o socket **newTCPsocket**) exclusivamente dedicado a representar a extremidade do novo canal de comunicação TCP que liga o servidor ao seu novo cliente. Efectivamente, repare-se que **accept()**, após a conexão se estabelecer, retorna um novo socket (**newTCPsocket**) que passa a representar a extremidade do novo canal no servidor.

Por seu lado, o cliente cria um socket local e liga-o ao servidor através da chamada **connect()** que lhe permite ligar o seu socket ao do servidor (identificado pelo endereço IP e uma porta TCP que são passados em parâmetro da chamada **connect()**).

```
socket(tcp) → localTCPsocket
connect(localTCPsocket, ipAddress, port)
```

A partir daqui ambos podem usar as chamadas **read(socket)** e **write(socket)** para trocar dados, como se o socket que representa a extremidade local do canal TCP fosse um ficheiro.

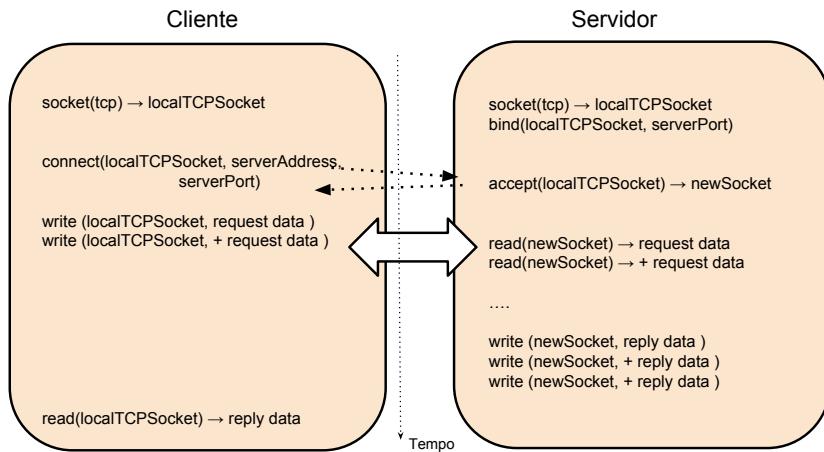


Figura 1.20: Comunicação com sockets TCP

Os dados escritos pelo cliente podem vir a ser lidos pelo servidor e vice versa. Com já referimos acima, os canais TCP não preservam nenhuma noção de mensagem, *i.e.*, o servidor pode escrever 100 bytes de cada vez e o cliente pode ler todos esses bytes lendo 1000 bytes de uma só vez caso estes já estejam disponíveis.

O leitor interessado no desenvolvimento de aplicações que usem directamente a interface de sockets ao nível sistema, e a linguagem C, pode consultar por exemplo [Stevens et al., 2004; Comer and Stevens, 1997] e muitos outros livros, ou ainda a numerosa documentação disponível *on-line*. Por exemplo, pesquisando por “Berkeley sockets” na Wikipedia. No capítulo 5 serão apresentados exemplos da utilização da interface de sockets para realizar clientes e servidores utilizando bibliotecas de mais alto nível.

1.7 Organismos de normalização e governação

Finalmente, para fechar o capítulo, vamos fazer referência a um aspecto crítico das redes de computadores: a normalização. Ao longo deste capítulo, em diversas ocasiões, foram feitas referência a normas tecnológicas, nomeadamente de protocolos e de interfaces. Sendo as redes de computadores realizações sofisticadas, com inúmeras componentes distintas, compostas por milhões de dispositivos hardware e software, cobrindo todos os continentes e com inúmeras interfaces, a normalização de interfaces e protocolos tem um papel primordial para permitir a inter-operação dessas várias componentes, em ambiente de concorrência e de multiplicidade de fornecedores de hardware e software. Só assim se poderá tentar assegurar liberdade de fabrico, construção e aquisição dos equipamentos e do software.

Uma norma tecnológica é o equivalente a uma especificação técnica de um mecanismo. O seu papel é garantir que qualquer implementação desse mecanismo, que obedeça à norma, funciona de acordo com a sua especificação e é capaz de interagir com mecanismos semelhantes que implementem a mesma norma. Adicionalmente, permite que os fabricantes e utilizadores do mecanismo tenham uma base comum para estabelecerem os seus contratos. Assim, uma norma funciona como uma base de entendimento comum entre os implementadores e os diferentes utilizadores.

Por outro lado, a gestão e inter-ligação de redes de acesso público, envolve facetas de governação e coordenação que também são relevantes para a inter-operação de redes da responsabilidade de diferentes autoridades administrativas.

Ao longo da história do desenvolvimento das redes de computadores diversos organismos foram desempenhando um papel importante na elaboração dessas normas e práticas de governação, promovendo grupos de trabalho que, através de discussão e consenso, vão elaborando as normas necessárias para garantir a inter-operação e a inter-conexão das redes.

Entre esses organismos figuram associações profissionais, organismos oficiais dos estados, consórcios empresariais e organizações não governamentais sem fins lucrativos. Para permitir ao leitor situar-se quando forem feitas referências a esses organismos, a seguir apresenta-se uma lista, não necessariamente completa, das mais relevantes na actualidade. Noutros capítulos serão referenciadas outras organizações que já tiveram maior impacto ou que actuam em facetas particulares, e cuja inclusão na lista abaixo torná-la ia demasiado longa.

A vida tem mostrado que os processos de normalização são complexos, muito lentos e pouco inovadores e estão, quase sempre, contaminados por facetas comerciais e de concorrência. No que diz respeito às redes de computadores, acresce que as mesmas estão na confluência de várias aproximações e de pontos de vista diferentes: os da indústria de computadores e os da indústria de telecomunicações tradicional (e até mais recentemente de outras indústrias e sectores empresariais onde se incluem os media).

Na indústria de computadores predominam as normas estabelecidas através de consórcios de fabricantes (ou impostas na prática por alguns deles). Devido à agilidade do desenvolvimento do software, e à constante evolução da capacidade dos dispositivos hardware, as normas evoluem muito depressa e os consórcios de construtores (e às vezes também de utilizadores) têm um peso significativo.

No que diz respeito à indústria de telecomunicações tradicional, que actua sobretudo nas comunicações à distância e em ambientes que requerem regulação (de frequências ou de direitos de instalação de infra-estruturas), os organismos de normalização tendem sempre para envolver organismos com representação dos Estados e Governos e um conjunto, geralmente reduzido, de operadores e de fabricantes de equipamentos.

A actividade académica, e mais tarde também empresarial, que envolveu o desenvolvimento da Internet, sempre esteve na confluência do mundo dos computadores com o mundo das comunicações. Dada a sua génese académica e o multiculturalismo tecnológico e industrial, a comunidade envolvida acabou por desenvolver métodos inovadores de estabelecimento de normas. Nesta comunidade, pelo menos numa primeira fase, foi dada uma grande importância ao mérito técnico e científico e a soluções pragmáticas testadas através de implementações reais, por isso a velocidade e impacto das normas foi, pelo menos inicialmente, bastante rápida.

É esta constatação que justifica a ordem pela qual são listadas as organizações que se seguem.

Internet Society (ISOC)

A Internet nasceu como um projecto de investigação académico, subsidiado pelo Governo dos EUA. Quando se tornou claro que era necessário tornar a rede pública, e acessível sem reservas em todos os países, foi necessário libertar a Internet do enquadramento governamental e nacional inicial. A solução encontrada consistiu na formação da Internet Society (ISOC), <http://isoc.org>, uma organização não governamental, aberta, enquadrando uma multiplicidade de pontos de vista e grupos de interesse, destinada a promover e enquadrar a governação da Internet.

No seu site *on-line* a sua missão é definida da seguinte forma: "To promote the

open development, evolution, and use of the Internet for the benefit of all people throughout the world.”

A ISOC não desenvolve normas técnicas mas a sua intervenção nas facetas técnicas da Internet, e com impacto técnico nas redes de computadores, faz-se via a sua colaboração e enquadramento do IETF (Internet Engineering Task Force).

Internet Engineering Task Force (IETF)

A IETF, <https://www.ietf.org>, é uma organização não lucrativa de voluntários, que nasceu em 1986 no âmbito do projecto de desenvolvimento inicial da Internet, e que é responsável por promover, de forma aberta, as soluções tecnológicas e de gestão, e as correspondentes normas, da rede Internet. No seu *site on-line* a sua missão é definida da seguinte forma: “The mission of the IETF is to make the Internet work better by producing high quality, relevant technical documents that influence the way people design, use, and manage the Internet.”

A IETF promove o desenvolvimento e é responsável pela edição e publicação da série de documentos conhecida como “Request for Comments” (RFC), ver <https://www.ietf.org/rfc.html>, que registam a maioria das normas e outra documentação técnica sobre a Internet e os protocolos e aplicações TCP/IP.

Actualmente a ISOC funciona como organização “guarda chuva” da IETF e publica o IETF Journal.

Institute of Electrical and Electronics Engineers (IEEE)

A IEEE, <http://www.ieee.org>, é uma associação de engenheiros e cientistas, com cerca de 400.000 membros no mundo inteiro, que tem como objectivo promover a evolução técnica e a educação nas áreas da engenharia electrotécnica, electrónica, de telecomunicações e de computadores.

No seu *site on-line* a sua missão é definida da seguinte forma: “IEEE’s core purpose is to foster technological innovation and excellence for the benefit of humanity”.

A IEEE desenvolve uma actividade muito importante de normalização em diversas áreas da sua actividade. Entre as normas mais conhecidas da IEEE, encontra-se o grupo de normas de redes locais e metropolitanas conhecidas pelo prefixo IEEE 802, que incluem as normas da rede Ethernet (802.3) e as normas de redes sem fios (802.11). Este conjunto de documentos normalizam os canais e redes mais populares a nível empresarial e residencial.

Tratam-se de normas, essencialmente promovidas pela indústria de computadores, por contraponto às normas desenvolvidas pela indústria e os operadores de telecomunicações, que são sobretudo editadas pela ITU.

International Telecommunication Union (ITU)

A ITU, <http://www.itu.int>, é uma agência especializada das Nações Unidas para as tecnologias de comunicação e informação. Trata-se portanto de uma organização inter-governamental, estruturada por países, com origem nas necessidades de normalização e da regulação da inter-operação dos sistemas de telecomunicações internacionais. Naturalmente, dado o enquadramento, os reguladores nacionais e os operadores de telecomunicações de cada país têm um peso significativo na sua actividade. Tem como membros 193 países, cerca de 700 empresas de telecomunicações e algumas universidades.

No seu *site on-line* a sua missão é definida da seguinte forma: “We allocate global radio spectrum and satellite orbits, develop the technical standards that ensure networks and technologies seamlessly interconnect, and strive to improve access to ICTs to underserved communities worldwide (...”).

A ITU tem uma actividade de normalização muito activa, sobretudo relevante a nível das tecnologias de comunicação mais importantes para os operadores de

telecomunicações. O seu envolvimento nas redes de computadores baseadas na comutação de pacotes desenvolveu-se em paralelo com as soluções desenvolvidas pela indústria de computadores ou no seio da IETF. Só com a adopção dos protocolos TCP/IP pelos operadores de telecomunicações no final do séc. XX os dois mundos se aproximaram em termos de tecnologia e normas.

1.8 Resumo e referências

Resumo

Uma rede de computadores tem na sua base uma infra-estrutura constituída por canais e comutadores de pacotes. Essa infra-estrutura permite aos computadores trocarem entre si mensagens de dimensão relativamente reduzida, designadas por pacotes de dados. No tipo de redes dominantes actualmente, baseadas nos protocolos TCP/IP, essas mensagens chamam-se pacotes de dados IP, e são encaminhados do computador origem ao computador destino numa base dita de melhor esforço, *i.e.*, sem garantias de entrega ou de ordenação.

Desenvolver aplicações com base na troca de pacotes IP é um processo complexo e pouco eficiente do ponto de vista da programação, pelo que os sistemas de operação dos computadores implementam protocolos de transporte extremo a extremo, (no sentido que são apenas executados nos computadores e não nos comutadores de pacotes), que oferecem serviços de nível superior que tornam mais fácil o desenvolvimento das aplicações distribuídas.

Vimos também que a infra-estrutura constituída pelos canais e os comutadores de pacotes é constituída geralmente por sub-redes, especializadas em funções específicas, de carácter operacional ou comercial (redes de acesso, redes residenciais, redes institucionais, redes de trânsito, redes de centros de dados, *etc.*). A Internet é um gigantesco agregado de milhões de redes de diferentes tipos.

Dada a dimensão e diversidade dos comutadores, dos computadores e dos sistemas de operação, os protocolos que especificam o significado e formato das mensagens trocadas pelos participantes na rede estão normalizados. Na secção 1.7 apresentou-se uma lista de organismos com relevância na normalização e governação das redes de computadores.

Os serviços de comunicação providenciados pelos protocolos de transporte são usados, por programas aplicação, para implementar aplicações distribuídas, geralmente baseadas em protocolos ditos aplicacionais. Essas aplicações distribuídas são constituídas por vários programas que colaboram usando um pequeno número de padrões de comunicação e coordenação. Alguns dos padrões mais populares são os que se designam por cliente / servidor e P2P.

Os protocolos de transporte mais populares no mundo TCP/IP são o TCP e o UDP. O protocolo TCP implementa uma abstracção de canal de comunicação lógico, bidireccional e fiável, que liga directamente dois programas. Este tipo de canais são especialmente úteis para transportar grandes quantidades de dados entre dois programas, com fiabilidade, mas sem garantias de tempo real. Por exemplo, o protocolo HTTP é um protocolo cliente / servidor aplicacional que usa canais TCP para comunicar.

Por outro lado, aplicações que apenas trocam pequenas quantidades de dados entre vários interlocutores, e cujas interacções materializam predominantemente operações idempotentes, preferem usar o protocolo UDP. Um exemplo muito popular desta opção é o serviço DNS, que é um sistema que permite, entre outras funcionalidades, estabelecer associações entre nomes mnemónicos e endereços IP. Outras aplicações que também usam UDP são aquelas que têm necessidades de tempo real, mas admitem perder alguns dos dados transferidos, como as aplicações multimédia em tempo real.

Finalmente, o capítulo apresenta uma primeira visita guiada à interface de sockets, que é a interface dominante para aceder nos sistemas de operação aos serviços providenciados pelos protocolos TCP e UDP.

Os principais termos introduzidos neste capítulo são a seguir passados em revista. Entre parêntesis figura a tradução mais comum em língua inglesa.

Canal de comunicação (*communication link, data link, link*) Dispositivo que permite transmitir dados entre equipamentos computacionais na rede.

Pacote de dados (*data packet*) Pequena mensagem unitária, geralmente constituída por algumas centenas de bytes, que é transmitida de uma só vez pela rede usando os comutadores de pacotes e os canais de comunicação.

Comutador de pacotes (*packet switch*) Equipamento que encaminha pacotes de dados entre os vários canais que lhe estão directamente ligados.

Rede de computadores (*computer network*) Infra-estrutura de canais e comutadores que liga vários computadores e lhes permite trocarem pacotes de dados.

Protocolo (*protocol*) Conjunto de mensagens, regras sintácticas e regras semânticas que regulam a comunicação e coordenação de um conjunto de entidades para atingirem um objectivo comum.

Protocolo IP (*IP protocol*) Protocolo de encaminhamento de pacotes entre computadores e comutadores da família TCP/IP. Os pacotes de dados, ditos pacotes IP à luz deste protocolo, são encaminhados segundo uma política de melhor esforço, sem garantias de fiabilidade.

Protocolo extremo a extremo (*end-to-end protocol*) Protocolo implementado apenas pelos computadores ligados à rede e que usa um protocolo de transporte ou a troca directa de pacotes disponibilizada pela infra-estrutura dos canais e comutadores.

Protocolo de transporte (*transport protocol*) Protocolo extremo a extremo especializado em funcionalidades de transporte de dados.

Protocolo TCP (*TCP protocol*) Protocolo de transporte que implementa canais lógicos fiáveis, bidireccionais, que interligam directamente dois programas.

Protocolo UDP (*UDP protocol*) Protocolo de transporte que permite aos programas trocarem mensagens, chamadas *datagramas* UDP, sem garantias de fiabilidade.

Cliente / servidor (*client / server*) Protocolo ou aplicação estruturados em torno de um padrão de interacção em que uma parte, o cliente, toma a iniciativa de solicitar serviços à outra, o servidor.

P2P (*peer-to-peer*) Protocolo ou aplicação estruturados em torno de um padrão de interacção em que os participantes actuam simultaneamente como clientes e servidores entre si.

Protocolo HTTP (*HTTP protocol*) Protocolo aplicacional que permite a um cliente Web pedir e receber objectos de um servidor Web. Para esse efeito o cliente usa um canal TCP para se ligar ao servidor.

Protocolo DNS (*DNS protocol*) Protocolo aplicacional que permite a um cliente interrogar um servidor de nomes. Geralmente, o protocolo é executado usando directamente a troca de *datagramas* UDP entre o cliente e servidores DNS.

Interface de sockets (*sockets interface*) interface do sistema de operação que dá acesso directo aos serviços providenciados pelos protocolos de transporte.

Referências

Este capítulo apresentou uma panorâmica de assuntos que vão ser tratados mais em detalhe nos capítulos seguintes. No entanto, caso o leitor prefira optar por outras leituras, existem vários livros que constituem referências bem conhecidas para o estudo das redes de computadores. Damos especial realce ao seguinte conjunto [Tanenbaum and Wetherall, 2011; Peterson and Davies, 2012; Kurose and Ross, 2013; Stallings, 2013].

Apontadores para informação na Web

- <https://www.wikipedia.org> – A Wikipedia, sobretudo na sua versão inglesa, tem inúmeros artigos sobre aspectos de rede de computadores que podem servir como forma de tirar dúvidas, por exemplo quando o leitor encontrar um termo que não conhece.
- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- <http://isoc.org/wp/ietfjournal/> – Site do IETF Journal.
- <http://standards.ieee.org/about/get/> – Site das normas IEEE 802.
- <http://www.itu.int/en/ITU-T/Pages/default.aspx> – Site de normas da ITU.
- <http://www.sigcomm.org> – Site do Special Interest Group on Communications (SIGCOMM), i.e., a divisão da Association for Computing Machinery (ACM) dedicada a redes de computadores e sistemas distribuídos. É responsável pela edição de numerosas publicações e organização de conferências sobre redes de computadores.
- <http://www.comsoc.org> – Site da IEEE Communications Society, a divisão da IEEE dedicada a comunicações, redes de computadores e sistemas distribuídos. É responsável pela edição de numerosas publicações e organização de conferências sobre redes de computadores.
- <http://www.internethalloffame.org> – Este Site, em parte ligado à ISOC, contém muita informação sobre os pioneiros do desenvolvimento inicial das redes de computadores e a história da Internet, nomeadamente no artigo disponível em <http://www.internethalloffame.org/brief-history-internet>.
- <http://internetpt.legatheaux.info> – Este Site, mantido pelo autor, contém informação diversa sobre o aparecimento da Internet em Portugal e os seus primeiros passos no nosso país.

1.9 Questões para revisão e estudo

1. Verdade ou mentira?
 - (a) Um protocolo especifica rigorosamente a sequência de mensagens que um conjunto de interlocutores tem de trocar para atingir um objectivo comum. No entanto, o formato das mensagens trocadas é livre para possibilitar evoluções futuras.
 - (b) As mensagens de um protocolo contêm duas partes: um cabeçalho e uma parte de dados. O cabeçalho contém informações de controlo do protocolo, a parte de dados transporta geralmente dados arbitrários.
 - (c) Para decidirem sobre a forma como encaminham os pacotes de dados, os comutadores de pacotes analisam a parte dos dados dos mesmos.
 - (d) As mensagens de um protocolo de extremo-a-extremo viajam normalmente na parte de dados dos pacotes transportados pela rede de comutadores.

2. Verdade ou mentira?

- (a) O protocolo IP foi pensado para a comunicação aplicação a aplicação.
- (b) Os pacotes IP emitidos por computador chegam sempre ao destino e pela ordem com que foram enviados. De qualquer forma, caso não seja esse o caso, o receptor é avisado pelos comutadores de pacotes.
- (c) Numa primeira análise, os pacotes de dados podem ser vistos como mensagens de dimensão arbitrária, ou pelo menos suficiente grande para a maioria das trocas de informação entre os computadores se fazerem num único pacote.
- (d) Os comutadores de pacotes IP no interior da rede estão todos sob a mesma autoridade administrativa, pois caso contrário seria impossível garantir que os pacotes cheguem ao destino.
- (e) O protocolo IP é um exemplo de um protocolo de transporte que assegura uma comunicação fiável de extremo a extremo
- (f) O protocolo IP é responsável por encaminhar de forma fiável pacotes de dados de um computador origem até um computador de destino
- (g) O protocolo IP não garante a ordem de entrega dos pacotes de dados de um computador origem até um computador de destino nem sequer se os entrega de facto.

3. Verdade ou mentira?

- (a) A rede Internet é formada por diversas sub-redes, nomeadamente uma por país.
- (b) Os utilizadores domésticos com ligações à Internet estão ligados por canais que os ligam directamente às redes de conteúdos.
- (c) Uma rede de trânsito é geralmente utilizada para a interligação de redes de acesso.
- (d) Uma rede de trânsito é uma rede que transporta pacotes que não tiveram origem nela e que não se destinam a computadores que lhes estejam directamente ligados.

4. Verdade ou mentira?

- (a) Os canais TCP não garantem a entrega ao receptor dos dados emitidos pelo emissor, nem sequer na ordem pela qual os mesmos foram emitidos.
- (b) Os canais TCP implementam um canal lógico de transmissão entre o emissor e o receptor cujo débito (número de bytes transmitidos pelo emissor e entregues ao receptor por unidade de tempo) é constante.
- (c) Uma conexão TCP é fiável e de extremo a extremo, isto é, não se perdem dados, porque os comutadores de pacotes da rede garantem que todos os pacotes que lhes chegam são entregues intactos ao comutador seguinte.
- (d) Uma conexão TCP é assegurada pelo sistema de operação dos computadores em diálogo através de um protocolo que pressupõe que a rede pode perder e trocar a ordem dos pacotes IP.

5. Verdade ou mentira?

- (a) As mensagens do protocolo HTTP são transmitidas na parte de dados de um *datagrama* UDP.
- (b) O número e tipo dos parâmetros de uma mensagem de pedido HTTP são fixos e não variam de pedido para pedido.

- (c) A dimensão de uma mensagem de resposta do protocolo HTTP é variável.
 - (d) O tipo dos dados de uma resposta HTTP é variável e está definido na mensagem.
 - (e) As mensagens de pedidos HTTP não estão divididas em cabeçalho e parte de dados.
6. Verdade ou mentira?
- (a) As funções da biblioteca sistema *resolver* abrem sempre um canal TCP para um servidor DNS para obterem respostas aos pedidos das aplicações.
 - (b) Quando um servidor DNS de um domínio não está disponível, deixa de ser possível conhecer os endereços IP dos computadores do domínio.
 - (c) O protocolo de consulta do DNS na sua versão iterativa faz do DNS um sistema P2P visto que todos os servidores DNS se comportam como clientes e servidores.
 - (d) As aplicações consultam o DNS usando o protocolo UDP porque, apesar de os pacotes UDP se poderem perder, a recepção de uma resposta assinala que o servidor DNS recebeu o pedido, e a aplicação a resposta do servidor.
7. Suponha que o tempo médio para executar uma interacção cliente / servidor (com um pedido e uma resposta) com um dado servidor DNS é desprezável quando o servidor de DNS está na rede interna da sua faculdade (é um *caching only server*), e é de 100 ms (milissegundos) quando o servidor DNS está fora dessa rede. Os computadores da rede da faculdade estão a usar o servidor local para obterem respostas a consultas ao DNS. Nas questões abaixo escolha a opção que mais se aproxima da resposta certa.
- (a) Qual o tempo necessário para um computador ligado à rede da sua faculdade obter o endereço associado ao nome **streaming.cnn.com**, admitindo que o servidor local não faz *caching*, só conhece o endereço IP dos servidores da raiz do DNS, e que o nome existe?
desprezável, 20, 50, 75, 100, 200, 300, 400, 500, 600, 700, 800 ms
 - (b) Os servidores DNS locais podem, ou não, ter a resposta a um pedido de um cliente disponível na sua *cache*. Tendo isso em consideração, qual é o tempo mínimo que pode ser necessário para um cliente dentro da rede da sua faculdade obter o endereço associado ao nome **streaming.cnn.com**?
desprezável, 20, 50, 75, 100, 200, 300, 400, 500, 600, 700, 800 ms
 - (c) Idem alínea anterior, mas agora determine qual é o tempo máximo que pode ser necessário para um computador ligado à rede da sua faculdade obter o endereço associado ao nome **streaming.cnn.com**?
desprezável, 20, 50, 75, 100, 200, 300, 400, 500, 600, 700, 800 ms
 - (d) Admita que a taxa de sucesso de o servidor DNS local ter a resposta a um pedido de um cliente disponível na sua cache é de 30%. Qual é o tempo médio que pode ser necessário a um computador ligado à rede da sua faculdade para obter o endereço associado ao nome **streaming.cnn.com**? Admita que quando o nome não está na *cache* o servidor demora sempre o tempo máximo determinado na alínea anterior.
desprezável, 20, 50, 75, 100, 200, 300, 400, 500, 600, 700, 800 ms
8. Verdade ou mentira?
- (a) Muitas aplicações desenvolvidas para usarem a Internet são estruturadas como um conjunto de programas que comunicam através de canais bidireccionais lógicos, capazes de encaminharem de forma fiável sequências de bytes entre 2 programas distribuídos.

- (b) As aplicações desenvolvidas para usarem a Internet utilizam a interface de sockets do sistema de operação para aceder aos serviços da rede. Essa interface não permite que entre dois computadores distintos exista mais do que um socket TCP para comunicação.
- (c) Os sockets UDP são os mais adequados para aplicações de transferência de ficheiros de grande dimensão.
- (d) Um socket UDP só pode ser usado para enviar *datagramas* depois de estar ligado ao socket do receptor.
- (e) Um computador envolvido numa transferência de um ficheiro a usar uma aplicação P2P só troca dados com um outro computador de cada vez.
9. Dê exemplos de aplicações para as quais é preferível utilizar um transporte em modo *datagrama*, *e.g.*, usando o protocolo UDP, e exemplos de aplicações para as quais é preferível utilizar um transporte em modo ligação da dados, *e.g.*, usando o protocolo TCP.

Capítulo 2

Canais de dados

*I'm gonna wrap myself in paper,
I'm gonna dab myself in glue,
Stick some stamps on top of my head!
I'm gonna mail myself to you.*

– Autor: Woody Guthrie, folk song writer, *The Mail Song*

Numa rede de computadores os pacotes de dados são transportados pela rede desde o emissor até ao receptor. Durante esta viagem, o pacote pode atravessar inúmeros canais, viajar milhares de quilómetros e os canais atravessados podem ser de diferentes naturezas e suportados em tecnologias muito diversas. Por exemplo, nas grandes distâncias é comum os canais basearem-se em fibra óptica ou serem suportados por satélites. Nas distâncias mais curtas é frequente serem suportados em fios de cobre que foram instalados nos últimos 100 anos para suportar a rede telefónica ou, mais recentemente, para suportar as redes de televisão por cabo. No interior dos edifícios é comum serem suportados em fios de cobre semelhantes aos fios telefónicos, mas começa também a ser popular a instalação de fibra óptica. Finalmente, nos últimos anos, os canais que usam a atmosfera como meio de propagação são cada vez mais populares, quer nas redes para o interior dos edifícios (conhecidas por redes Wi-Fi), quer nas redes celulares dos operadores de telecomunicações móveis.

Compreender em profundidade como funcionam os diferentes tipos de canais exigiria o estudo de vários livros. De facto, o seu conhecimento profundo requer o tratamento de temas do âmbito da teoria da informação, do processamento de sinal, da propagação de sinais, electrónica, óptica, lasers, etc. No entanto, se o objectivo principal é compreender como funcionam as redes de computadores, é possível trabalhar com um modelo de canal de dados que abstrai as propriedades essenciais mais relevantes para esse efeito. É esse o ponto de vista que será usado neste capítulo – permitir ao leitor ter uma ideia suficiente para perceber o que é um canal de dados, quais as suas propriedades essenciais e como estas têm impacto nos níveis superiores da rede. O nível de detalhe a que chegaremos é o estritamente essencial para se perceber o impacto que os diferentes tipos de canais têm no comportamento mais geral da rede e das aplicações que a utilizam.

Por outro lado, a evolução tecnológica e as dimensões industriais e comerciais das redes, tomadas no seu sentido mais lato, incluindo portanto as redes telefónicas e de entretenimento, tiveram um impacto profundo sobre os canais que são usados quer para nos ligarmos a redes, quer para construir os seus *backbones*. Hoje em dia, em muitos países mais desenvolvidos, a disponibilidade de canais de dados em redes sem

fios e em redes móveis é generalizada e as casas começam também a ser ligadas por canais de fibra óptica aos operadores de redes.

Antigamente os canais usados para construir redes de computadores eram, na maior parte das vezes, como que subprodutos de canais de outras redes (telefónicas, de televisão, *etc.*). Por esta razão, os livros sobre redes de computadores dedicavam muitos capítulos a explicar como é que esses canais eram construídos reutilizando funcionalidades disponibilizadas por essas redes especializadas. Em contrapartida, hoje em dia a situação alterou-se bastante e as posições inverteram-se, e um livro sobre redes de computadores deve explicar agora como o serviço telefónico e a distribuição de canais televisivos (um conceito também em grande mutação) passaram a ser oferecidos pelas redes de computadores.

Por todas estas razões, este capítulo não dá muita ênfase a explicar como funcionam as diversas variantes de canais de acesso à Internet (*e.g.*, linhas telefónicas, ISDN, ADSL, redes híbridas de fibra e cabos coaxiais, redes por cabo, redes ópticas, *etc.*) pois provavelmente pelo menos algumas delas deixarão de ser populares dentro de alguns anos. Com efeito, os canais de dados só podem ser compreendidos em profundidade abordando aspectos tecnológicos, económicos e de organização dos mercados que ultrapassam o âmbito deste livro.

O capítulo começa por introduzir uma definição genérica e abstracta de canal de dados que destaca as componentes que o formam, assim como o seu papel, e descreve como estas funcionam. Depois apresenta as facetas mais relevantes que caracterizam os diferentes tipos de canais. Em seguida analisa que grandezas condicionam o desempenho de um canal, quer do ponto de vista da quantidade de informação que este é capaz de transmitir por unidade de tempo, quer do ponto de vista do tempo de trânsito imposto pelo canal. Aqui chegados veremos de forma muito sucinta quais os principais suportes de transmissão à distância que são utilizados para construir canais de dados. Um aspecto determinante para o desempenho global da rede são os erros de transmissão que podem ocorrer nos canais. De facto, quando frequentes, os erros têm um impacto bastante negativo no desempenho da rede e das aplicações. Por isso este capítulo dedica uma secção ao problema dos erros de transmissão e sua detecção. Finalmente, o capítulo apresenta um exemplo de canal que é bastante comum e também usa o formato das mensagens que este transmite como exemplo representativo.

2.1 Definição de canal de comunicação

Um **canal de comunicação** (*communication link*) é um dispositivo que permite a computadores e comutadores de pacotes trocarem pacotes de dados de forma directa. Geralmente utiliza-se o termo **nó de comunicação** (*communication node*) para designar, sem necessidade de distinções suplementares, os dispositivos que comunicam através dos canais.¹

Como já foi referido, os pacotes de dados podem ser vistos como mensagens, *i.e.*, simples sequência de bits. Para pôr em evidência o facto de que os canais não interpretam os pacotes de dados, apenas os transmitem, e que os protocolos usados nos canais requerem cabeçalhos específicos, a este nível os pacotes de dados são designados por um termo distinto. Na literatura em língua inglesa sobre canais de dados estas sequências são designadas normalmente por *bit frames* pois, como veremos a seguir, elas têm um formato especial que inclui um prefixo e um sufixo. Neste capítulo e noutras vamos frequentemente usar o termo em inglês *frame*² para designarmos as mensagens transmitidas ao nível canal.

¹ Esta nomenclatura tem origem na teoria dos grafos, pois uma rede é muitas vezes modelada como um grafo, o qual é constituído por um conjunto de nós e um conjunto de arcos (os canais).

² Um *bit frame* poderia ser traduzido quase literalmente por “enquadramento de bits”,

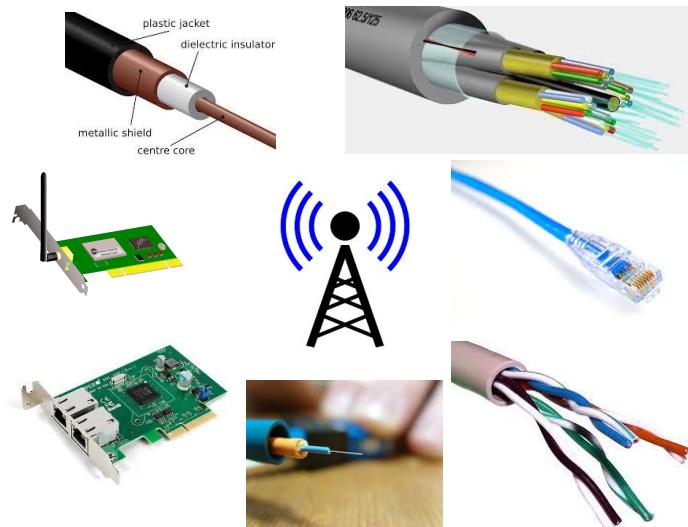


Figura 2.1: Exemplos de interfaces de comunicação e de meios de propagação do sinal

Grosso modo, um canal de dados é constituído por **interfaces de comunicação** (*communication interfaces, network adapters, network interface cards (NIC)*), e um **meio de propagação do sinal** (eléctrico, electromagnético, luminoso ou sonoro) (*propagation media*) que transporta a informação a transmitir. A Figura 2.1 mostra exemplos de interfaces e de meios de propagação do sinal. Convém, no entanto, não esquecer que hoje em dia a maioria dos computadores e outros dispositivos equiparáveis têm as interfaces integradas na chamada “placa mãe” (*motherboard*). O meio de propagação do sinal é às vezes designado como meio de transmissão ou de transporte da informação, e pode assumir formas muito variadas como por exemplo fios eléctricos, fibras ópticas, a atmosfera, *etc*. A Figura 2.2 mostra as componentes de um canal que liga dois nós através de um meio de transmissão constituído por 4 fios eléctricos: 2 transmitem sinais (um em cada sentido) e 2 transmitem o sinal de referência ou de terra (também um em cada sentido).

As interfaces de comunicação dispõem de registos que contêm os pacotes a transmitir pelo meio de comunicação. Quando é possível realizar a transmissão de um pacote, a unidade de controlo da interface acrescenta-lhe os campos requeridos pelo protocolo do canal para construir um *frame* e desencadeia a codificação da sequência de bits na forma adequada à sua transmissão pelo meio, ou seja, realiza a transformação da sequência de bits numa sequência de sinais eléctricos, electromagnéticos, sonoros ou luminosos que permitem a sua transmissão.

A forma como esta codificação é realizada é muito variada e depende do tipo de canal, do seu débito e do meio de transmissão. A Figura 2.2 sugere, meramente a título de exemplo, uma forma de codificação muito simples: os bits a 0 são codificados num nível de tensão eléctrica, por exemplo -5 volts, e os a 1 num nível de tensão eléctrica diferente, por exemplo +5 volts. Desta forma de codificação resultaria um sinal digital alternando entre esses dois valores de tensão eléctrica. Compete à interface do lado do receptor reconhecer o início da mensagem e realizar a transformação inversa, *i.e.*, transformar a sequência de sinais detectados no meio de comunicação na sequência de

“quadro de bits” ou “trama de bits”. Apesar de esta última tradução ser a mais natural, não utilizaremos nenhuma destas traduções pois as mesmas podem resultar confusas.

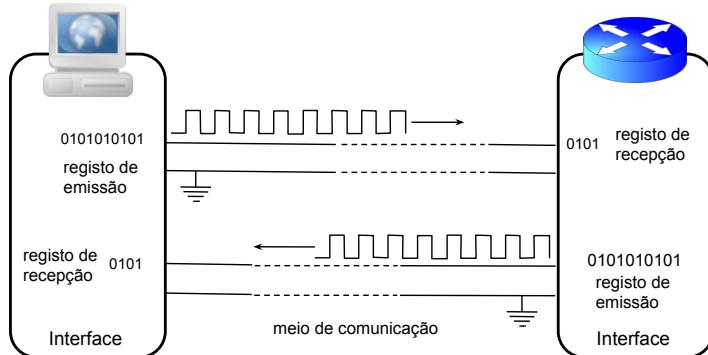


Figura 2.2: Um canal de comunicação

bits que constituí o *frame* recebido.

Um **canal de comunicação de dados (data link)** é um dispositivo tecnológico que permite a um conjunto de nós de comunicação trocarem mensagens diretamente entre si. O canal é formado por interfaces e um meio de propagação de sinal. As interfaces codificam a informação a transmitir da forma mais adequada para ser transmitida pelo meio. As mensagens transmitidas pelo meio de propagação chamam-se *frames*.

Esta visão da forma como funciona um canal de dados é lógica e modeliza de forma genérica a enorme diversidade de canais existentes. Para se penetrar um pouco mais na operação de um canal é necessário analisar um conjunto alargado de questões, parte das quais são referidas brevemente a seguir. O leitor interessado numa visão mais profunda terá de mergulhar no mundo das telecomunicações e da transmissão de dados e consultar obras especializadas, como por exemplo [Hsu, 2003; Couch, 2000; Forouzan, 2007; Moussavi, 2012].

Canais ponto-a-ponto e multi-ponto

Um canal diz-se **ponto-a-ponto (point-to-point)** quando liga apenas dois nós e diz-se **multi-ponto (multipoint)** quando liga mais do que dois nós. Os canais baseados em fios de cobre, cabos coaxiais ou fibras ópticas dizem-se canais baseados **em meios de comunicação guiados (guided media)** (cujo meio de transmissão é baseado num “fio”). É comum, mas não obrigatório, que este tipo de canais liguem apenas dois nós e nesse caso são canais ponto-a-ponto. Os canais baseados num **meio de comunicação não guiado (unguided media)** (*i.e.*, geralmente a atmosfera ou a água) difundem o sinal através de um meio de comunicação sem fios (*wireless*). Geralmente este tipo de canais suportam a comunicação entre vários (> 2) nós e são portanto multi-ponto. É importante, no entanto, realçar que existem canais multi-ponto baseados em meios de transmissão guiados e canais ponto-a-ponto baseados em antenas direcionadas e que usam a atmosfera como meio de propagação.

Canais *simplex*, *half-duplex* e *ful-duplex*

Alguns canais apenas permitem transmitir informação num só sentido e dizem-se canais **simplex**. Os canais mais interessantes e populares permitem transmitir informação nos vários sentidos. Quando um canal ponto-a-ponto é deste último tipo, e permite que

a comunicação nos dois sentidos tenha lugar em paralelo (*i.e.*, simultaneamente), o canal diz-se *full-duplex*. Na verdade o canal é equivalente a dois canais *simplex*, um em cada sentido, como ilustrado na Figura 2.2. Caso o canal permita a comunicação em vários sentidos mas apenas suporte que esta tenha lugar num só sentido de cada vez (*i.e.*, em alternância), o canal diz-se *half-duplex*.

A maioria dos canais ponto-a-ponto actuais são *full-duplex* e são logicamente equivalentes a dois canais *simplex*, um em cada sentido, que trabalham em paralelo e de forma independente.

Codificação da informação

A forma como a codificação da sequência de bits é realizada está dependente do meio de transmissão, da qualidade das interfaces e do ambiente em que se pretende usar o canal. A transmissão e codificação de informação digital para transmissão remota tem sido objecto de intensa investigação e inúmeros desenvolvimentos industriais e militares nas últimas dezenas de anos.

Detecção de erros

Alguns meios de propagação deformam os sinais ao longo da sua extensão e outros são especialmente vulneráveis ao ruído. Assim, o sinal que muitas vezes chega ao receptor é bastante diferente do que foi produzido pelo emissor. Se o receptor não for particularmente eficaz a reconhecer a sequência de bits codificada no sinal originalmente emitido, o *frame* recebido pode ser diferente do emitido. Se esta alteração não fosse detectada, as aplicações poderiam receber mais facilmente dados errados.

É possível introduzir nas sequências de bits emitidas códigos com bits redundantes que permitem que o receptor detecte os erros ou até que os consiga corrigir. Alguns destes códigos são discutidos brevemente na secção 2.4. O leitor interessado num tratamento mais profundo, poderá consultar [Hamming, 1980; Zaragoza, 2002; Rorabaugh, 1996].

Enquadramento ou *framing* e sincronização

À primeira vista parece simples o receptor reconhecer o início de um *frame* ao nível canal pois tal evento deverá corresponder a uma transição bem definida das características do sinal detectado. Na verdade, dependendo das formas de codificação utilizadas, o problema pode revelar-se delicado. Adicionalmente, o receptor, para interpretar o sinal recebido, tem de retirar amostras periódicas do mesmo. Para este efeito utilizará um dispositivo (geralmente um relógio baseado num oscilador) que marca o momento em que cada amostra deve ser medida. É necessário sincronizar estes momentos de amostragem de forma adequada ao sinal recebido. Uma das formas de resolver estes problemas é explicada na secção 2.5.

Controlo de fluxo e controlo de erros

Um receptor lento pode não conseguir tratar atempadamente os *frames* recebidos pela sua interface. Isto poderá conduzir a uma situação em que um novo *frame* é recebido mas não há nenhum registo livre para o memorizar. Só resta ao receptor “esquecê-lo” (também se diz descartá-lo). Por outro lado, quando a probabilidade de um *frame* chegar com erros é elevada, muitos terão de ser igualmente recusados pela interface do receptor.

Alguns canais incluem um protocolo executado entre as interfaces do emissor e do receptor que adapta o ritmo de emissão de *frames* à capacidade de o receptor não os perder, ou que leva à reemissão dos *frames* que chegaram com erros.

As redes sem noção de conexão rede, como são as redes TCP/IP, ver o Capítulo 3, dispensam controlo de fluxos e de erros (*flow and error control*) ao nível canal. No entanto, em canais com elevadas taxas de erro, o desempenho global da rede pode ser favorecido quando estes problemas são atacados logo “à nascença”, *i.e.*, ao nível dos canais que os exibem para além do razoável. As técnicas usadas para introduzir controlo de fluxos e controlo de erros serão discutidas detalhadamente no capítulo 6.

Após esta visão qualitativa, passamos agora a completar o modelo de um canal com a caracterização quantitativa do mesmo.

2.2 Caracterização quantitativa dos canais

Os canais de dados têm um impacto relevante no desempenho da rede de computadores, *i.e.*, as características dos canais determinam se a rede está bem dimensionada e responde às necessidades, ou se pelo contrário está saturada e com dificuldade em proporcionar um nível de serviço adequado. Por essa razão, nesta secção abordaremos as principais propriedades quantitativas dos canais pois são estas que mais facilmente permitem caracterizar indirectamente o desempenho da rede.

Débito, capacidade ou velocidade de transmissão

Dado que a função de um canal é transmitir informação entre os nós de uma rede, a sua mais importante caracterização quantitativa diz respeito à quantidade de informação por unidade de tempo que é capaz de transmitir. Esta grandeza diz-se o débito ou capacidade do canal (*link bit rate or bandwidth*) e mede-se em bits por segundo (bps). Dadas as ordens de grandeza envolvidas, utilizam-se geralmente as abreviaturas K (kilo), M (mega), G (giga) e T (tera) como abreviaturas respectivamente de 10^3 , 10^6 , 10^9 , 10^{12} .³

O débito, capacidade ou velocidade de transmissão (*link bit rate or bandwidth*) de um canal é a quantidade de informação, medida em bits por segundo, que o canal é capaz de transmitir por unidade de tempo.

O débito dos canais é função de vários factores que são fixos e constantes, como o esquema de codificação usado e as características do meio de transmissão. No entanto, alguns canais envolvem mecanismos suplementares como por exemplo bits redundantes de sincronização do receptor com o emissor, códigos de controlo de erros, mecanismos de compensação de erros, cabeçalhos com endereços e até, nos canais multi-ponto, mecanismos de coordenação dos diferentes emissores. Todos estes mecanismos conduzem a desperdícios (*overheads*) variáveis da capacidade máxima de transferência que são importantes para caracterizar um canal.

No entanto, quando se caracteriza de forma sintética o débito de um canal, é comum indicar o débito máximo teórico permitido pelo método de codificação e de transmissão. A diferença entre o débito máximo e o débito real é geralmente pouco significativa nos canais ponto-a-ponto com taxas de erro insignificantes.

Se o objectivo fosse transferir de forma contínua quantidades infindáveis de informação, a caracterização do canal pelo seu débito poderia ser suficiente. No entanto, em muitos casos estamos interessado em saber também quanto tempo leva um dado pacote a ser transferido da memória do emissor para a memória do receptor. Ignorando o tempo necessário para copiar o pacote dos registo das interfaces para a memória

³Como se trata de uma grandeza que se exprime por unidade de tempo, as abreviaturas indicadas, por convenção, não representam 2^{10} , 2^{20} , 2^{30} , ..., como é comum quando se medem em Informática quantidades de memória.

central dos nós, e vice-versa, o tempo total de transferência envolve a transmissão do *frame*, o progresso dos seus bits através do meio de comunicação e finalmente a recepção dos mesmos.

Para se calcular este tempo é necessário calcular o tempo para transmitir todo o *frame* do primeiro ao último bit e somar-lhe o tempo que cada bit demora a ser recebido pela interface do destinatário, pois um *frame* só estará integralmente recebido quando for recebido o seu último bit.

Tempo de transmissão

O tempo de transmissão de um *frame* (*frame transmission time*) depende do débito do canal pois, como é necessário transmitir os bits uns após os outros, se o *frame* tem D bits (a dimensão do *frame*) e o débito do canal é C bits / s (a capacidade ou débito do canal), então o *frame* leva D/C segundos a ser transmitido, *i.e.*, medeiam D/C segundos desde que começa a ser emitido o primeiro bit até que acaba de ser emitido o último bit. A Figura 2.3 ilustra o impacto do débito de um canal (ou o seu *bitrate*) sobre o tempo de transmissão dos *frames*.

A relação entre tempo de transmissão e débito é bem ilustrada por um exemplo do dia-a-dia: introduzir uma dada quantidade de líquido dentro de um tubo leva tanto mais tempo quanto menor for o débito (*i.e.*, a largura) do tubo.

O tempo de transmissão (*transmission time*) de um *frame* com D bits por um canal com o débito ou capacidade de C bits por segundo, *i.e.*, o tempo desde que começa a ser emitido o primeiro bit até que acabe de ser emitido o último bit, é D / C segundos.

$$\text{Tempo de transmissão} = \text{Dimensão do } frame / \text{Débito do canal}$$

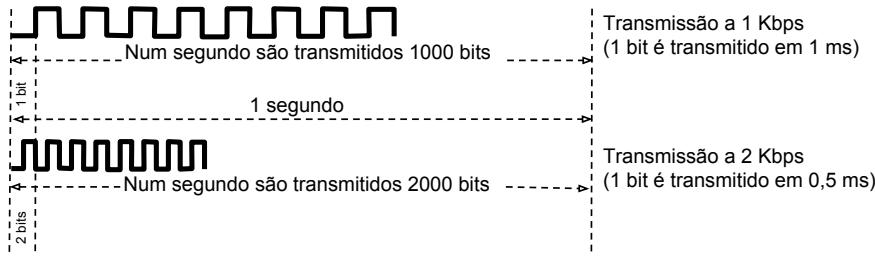


Figura 2.3: Aumentando o débito para o dobro (de 1 Kbps para 2 Kbps) o tempo de transmissão diminui para metade (1000 bits passam a ser transmitidos em 0,5 segundo ao invés de em 1 segundo)

Na literatura de língua inglesa, o débito de um canal pode ser indicado como sendo a sua *bandwidth* ou *bit rate*. O primeiro termo tem a ver com o facto de que existe uma relação indirecta entre o débito do canal e a banda passante, *i.e.*, a dimensão do intervalo das frequências que o canal utiliza no seu meio de suporte.

Um aspecto que importa realçar é que existe uma relação directa entre o *bit rate*, ou débito de um canal, e o tempo que leva a transmitir um bit por esse canal: num canal com o débito de 1 Kbps, cada bit leva 1 ms a transmitir ($1/10^3 = 10^{-3}s = 1\text{ ms}$), se o débito é 1 Mbps, cada bit leva 1 micro segundo a transmitir ($1/10^6 = 10^{-6}s = 1\text{ }\mu\text{s}$), se

o débito é 1 Gbps, cada bit leva 1 nano segundo a transmitir ($1/10^9 = 10^{-9}s = 1\text{ ns}$), etc.

Tempo de propagação

À primeira vista poderia parecer que o tempo que leva um bit a transitar pelo meio de propagação é desprezável. No entanto, se um canal tiver vários quilómetros, tal não será o caso. Esse tempo de trânsito depende essencialmente do tempo de propagação (*propagation time*) do sinal no meio de comunicação. Medidas efectuadas indicam que o valor de 200.000 Km / s é uma boa estimativa para a velocidade de propagação média do sinal na maioria dos meios guiados, nomeadamente nas fibras ópticas.

Com efeito, na fibra essa velocidade de propagação é $\approx 2 \times 10^8\text{ m/s}$ e num fio de cobre é ligeiramente diferente mas, como a fibra é o meio dominante a grande distância, a sua velocidade de propagação é usada como referência para o cálculo do tempo de propagação. Se um canal tiver 1 Km de comprimento, o tempo de propagação é de $1/200000 = 5 \times 10^{-6}\text{ s} = 5\text{ }\mu\text{s}$. No entanto, se o canal tiver 10.000 Km, como por exemplo um canal do centro da Europa ao centro dos EUA, esse tempo sobe para $10000/200000 = 50$ milissegundos (50 ms), o que pode revelar-se mais significativo.

Daqui resulta que o tempo de trânsito de um *frame* do emissor até ao receptor é igual à soma do tempo de transmissão com o tempo de propagação.

O tempo de trânsito de um *frame* por um canal é, no caso geral, dado por:

$$\text{Tempo de trânsito} = \text{Tempo de transmissão} + \text{Tempo de propagação}$$

com:

$$\text{Tempo de transmissão} = \text{Dimensão do } frame / \text{Débito do canal}$$

$$\text{Tempo de propagação} = \text{Dimensão do canal} / \text{Velocidade de propagação no meio}$$

A Figura 2.4 ilustra a problemática do tempo de propagação num canal com um comprimento significativo. O sinal que codifica cada um dos bits leva o dobro do tempo a propagar-se até ao fim de um canal com 2.000 Km do que num canal com 1.000 Km. Logo, o último bit emitido leva o dobro do tempo até ser recebido pelo receptor.

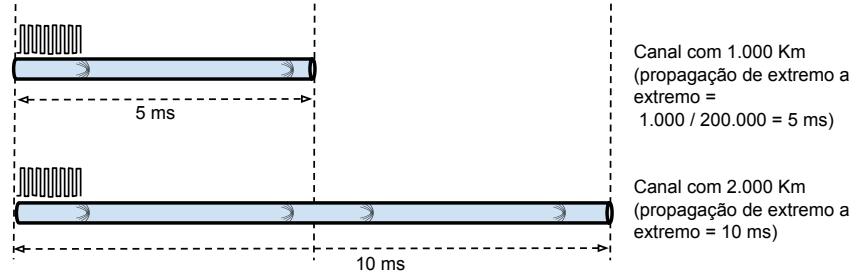


Figura 2.4: Num canal com 1.000 Km um bit chega à outra extremidade em 5 ms mas se o canal tiver 2.000 Km chega em 10 ms

O impacto do tempo de propagação sobre um protocolo pode ser medido pelo seguinte exemplo: admita que é necessário transferir uma mensagem M entre dois nós

ligados por um canal. A mensagem M tem de ser partida em 100 pacotes e o protocolo de transferência usado exige que seja transferido um pacote de cada vez. Admita ainda que o tempo de transmissão de cada pacote é 1 ms. Se o tempo de propagação for desprezável, M é transferida em 100 ms. Mas se o tempo de propagação for de 50 ms, o tempo total de transferida é de 5,1 segundos.

Em muitos protocolos, como por exemplo num protocolo cliente / servidor, é necessário atravessar o canal em ambos os sentidos. Admitindo que a dimensão da mensagem é idêntica nos dois sentidos, o tempo total de ida e volta é o dobro do tempo de trânsito num só sentido. Na literatura em língua inglesa usa-se o termo *Round Trip Time - RTT* para designar este tempo de trânsito de ida e volta. Finalmente, os termos usados na mesma língua para o tempo de trânsito são *latency* e *delay*.

Tempo de trânsito de ida e volta (RTT) (*Round Trip Time*) é o tempo necessário para que uma mensagem vá da origem ao destino, e uma mensagem idêntica faça o percurso inverso.

O impacto do tempo de transmissão e do tempo de propagação nas aplicações depende da forma de funcionamento das mesmas. Numa aplicação cliente / servidor que envia quantidades pequenas de dados num sentido e no outro, o tempo de propagação é dominante em quase todos os canais modernos (que têm débitos superiores a 10 Mbps). Por exemplo, se uma pergunta a um servidor DNS envolver mensagens com cerca de 125 bytes em cada sentido, o tempo de transmissão a 10 Mbps é de cerca de $125 \times 8/10^7 = 10^3/10^7 = 10^{-4} = 0,1\text{ ms}$. Se o canal a atravessar tiver 10.000 Km, o RTT é aproximadamente 100 ms e não é muito importante por em evidência que ele é de facto 100,2 ms. No entanto, se a aplicação consiste em transferir o conteúdo de um filme pelo mesmo canal, admitindo que esse filme exige a transferência em contínuo de centenas de M bytes, então o que interessa é mesmo o débito do canal pois o tempo de trânsito não terá assim tanta relevância desde que seja limitado e constante.

Volume do canal

O impacto do tempo de propagação sobre a caracterização de um canal pode ser captado pela noção de volume de um canal (*link bandwidth delay product*), grandeza que mede a quantidade de bits total que pode estar em trânsito simultaneamente no canal, e que é dada pelo produto do débito pelo tempo de propagação.

$$\text{Volume do canal} = \text{Débito} \times \text{Tempo de propagação}$$

O volume do canal corresponde à quantidade de bits que é necessário emitir continuamente para o encher completamente (*i.e.*, até que o primeiro bit que foi emitido chegue ao receptor). Uma outra forma de visualizar o volume do canal é imaginar uma transmissão de bits (ou berlindes) contínua e olhar para os bits que estão “pendurados”, *i.e.*, em trânsito, entre os dois extremos do canal.

Se um protocolo exige que os dois parceiros na extremidade do canal dialoguem entre si, então o protocolo só aproveita bem a disponibilidade do canal se cada uma das partes poder emitir continuamente mensagens de dimensão semelhante ao volume do canal. Caso contrário, ficarão à espera da outra parte e desperdiçarão a capacidade de comunicação disponível (se só elas o estiverem a usar naquele momento).

O volume dos canais é particularmente relevante em canais de grande débito e grande distância (a qual afecta o tempo de propagação do sinal). Por exemplo, um canal com o débito de 1 Gbps e um tempo de propagação de 50 ms tem um volume de

$$10^9 \times 0,050 = 10^9 \times 5 \times 10^{-2} = 5 \times 10^7 = 50 \text{ Mbits} \approx 5 \text{ MBytes}$$

que seria uma mensagem de dimensão bastante apreciável. Na prática estes canais são geralmente utilizados por várias aplicações simultaneamente, como veremos no capítulo 3, e o problema é atenuado.

Cálculos com aproximações

Este exemplo permite-nos voltar a chamar a atenção para que nas grandezas envolvendo tempos, como por exemplo velocidades, as abreviaturas K(ilo), M(ega), G(iga), ... representam potências de 10 (dez) e não potências de 2 (dois)⁴. No entanto, quando falamos da quantidade de informação que está registada numa memória, a convenção é que K representa 2^{10} , M representa 2^{20} , G representa 2^{30} , etc.

Por outro lado, e como é bem sabido, 1 B (1 byte) contém 8 bits, pelo que o resultado do cálculo acima deveria ser $50 \text{ Mbits} = 50.000.000/8 \text{ bytes}$ que é diferente de 50 MB. No entanto, como muitas vezes apenas estamos interessados em estimar a ordem de grandeza, usamos bytes com 10 bits o que representa um erro de cerca de 20% e consideramos que $2^{10} \approx 10^3$, que $2^{20} \approx 10^6$, que $2^{30} \approx 10^9$, etc. Adicionalmente, e pela mesma razão, se pretendermos calcular o tempo que leva a transmitir um pacote com 1 KB por um canal com o débito de 1 Mbps, podemos considerar que esse tempo é sensivelmente

$$1 \times 8 \times 10^3 / 10^6 = 8 \times 10^{-3} = 8 \text{ ms}$$

que é diferente de $8 \times 2^{10} / 10^6$ mas é uma aproximação aceitável em situações em que só se está a estimar a ordem de grandeza.

Em situações em que um valor aproximado é aceitável podemos considerar que $2^{10} = 1024 \approx 10^3$, que $2^{20} \approx 10^6$ etc. pois o erro cometido é inferior a 3%.

Em situações em que pretendemos apenas saber qual é a ordem de grandeza de um resultado podemos considerar que um byte tem 10 bits e que por exemplo $1 \text{ KB} = 8 \times 1024 \approx 10 \times 10^3 = 10^4 \text{ bits}$. O erro introduzido é aproximadamente 18%.

Taxa de erros

Para além do débito, tempo de propagação, RTT e volume, um canal também é caracterizado pela sua taxa de erros. Esta grandeza será referida de forma mais rigorosa a seguir. Por agora vamos considerar que a taxa de erros é o rácio entre os bits recebidos erradamente e a totalidade dos bits transmitidos.

Taxa de erros de um canal (*error bit rate*) é o quociente entre os bits recebidos com erros e a totalidade dos bits transmitidos por um canal.

⁴Os prefixos Kibi (Ki), Mebi (Mi), Gibi (Gi), etc., introduzidos pela norma SI (Sistema Internacional), podem ser usados quando se pretende representar potências de 2. No entanto, como a norma é recente e a larga maioria do material bibliográfico ainda não a adaptou, a nova e a antiga normas coexistem e é necessário usar o contexto e as unidades na interpretação de qual o significado dos prefixos K, M, G, etc.

2.3 Exemplos de meios de comunicação

Nesta nossa viagem pelo mundo dos canais chegou agora a altura de fazer uma rápida visita guiada aos meios de propagação do sinal usados nos canais de dados. Desta forma será mais fácil estabelecer uma ponte entre o modelo abstracto de um canal de dados e aspectos concretos do seu funcionamento que analisaremos posteriormente.

Meios guiados baseados em fios de cobre

A primeira rede de comunicações directamente acessível nas habitações foi a rede telefónica. Por esta razão a grande maioria das zonas urbanas foi dotada de pares de fios de cobre que ligam as casas às centrais telefónicas e generalizou-se a instalação de cabos telefónicos no interior dos edifícios. Com o passar do tempo ambas as infraestruturas foram aproveitadas ou reconvertidas para suportarem canais de dados.

Os exemplos mais comuns de suporte de canais de comunicação baseados em pares de fios de cobre são os chamados cabos UTP/STP (*Unshielded / Shielded Twisted Pairs*).⁵ Estes cabos são usados dentro dos edifícios e suportam comunicações a distâncias curtas (inferiores a uma ou duas centenas de metros) com baixa taxa de erros e velocidades de transmissão elevadas: de centenas de Mbps até Gbps.



Figura 2.5: Cabos de pares retorcidos ou entrançados, cabo UTP com fichas normalizadas (RJ45) e fichas RJ45 ligadas a interfaces

A qualidade final permitida por estes suportes depende do comprimento máximo, da grossura dos fios de cobre e da utilização ou não de isolamento metálico externo. Com isolamento externo metálico o cabo diz-se cabo blindado (*shielded*). Ao contrário, se o cabo só tem isolamento plástico diz-se não blindado (*unshielded*), sendo esta a versão mais generalizada. A Figura 2.5 mostra um par de fios de cobre entrançados, cabos UTP contendo 4 desses pares e as fichas normalizadas usadas para ligar os cabos às interfaces dos nós.

Os cabos de pares entrançados de fios de cobre são muito populares por serem económicos, cómodos de instalar, leves e flexíveis, e permitirem boa qualidade de comunicação a distâncias curtas. As velocidades de transmissão vão de vários Mbps (com cabos de centenas de metros) a alguns Gbps (com cabos mais curtos, de vários metros).

Quando se pretendem usar cabos de fios de cobre expostos a intempéries e transmitir maiores quantidades de informação a distâncias mais significativas, recorre-se

⁵A designação *Unshielded*, respectivamente *Shielded*, *Twisted Pair* pode ser traduzida por cabo de pares retorcidos ou entrançados sem blindagem, respectivamente com blindagem.

aos chamados cabos coaxiais. Estes cabos são a forma mais comum de distribuir o sinal de televisão nas redes de televisão por cabo e permitem obviar aos dois problemas citados. Um cabo coaxial, ver a Figura 2.6, dispõe de um condutor interno em cobre cujo diâmetro é superior ao dos cabos UTP. Este condutor está protegido por uma manga plástica que o separa de uma rede metálica que funciona como “terra” e blindagem contra o ruído. Finalmente, todo o conjunto é envolvido numa protecção externa que permite uma maior resistência a intempéries e outras agressões. Por todas estas razões estes cabos são menos económicos, são mais difíceis de instalar e menos flexíveis na utilização. A tendência actual é para serem substituídos por fibras ópticas.

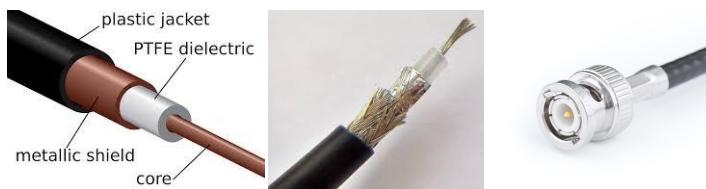


Figura 2.6: Cabo coaxial e ficha de terminação

Meios guiados baseados em fibras ópticas

As fibras ópticas foram inventadas para suportarem comunicação digital de alta qualidade, *i.e.*, de grandes quantidades de informação com baixa taxa de erros. Nestas, a informação a ser transportada à distância é codificada em impulsos luminosos, que se propagam através de um “fio” especialmente concebido para esse fim. As fibras podem ser construídas em plástico ou em sílica – neste caso têm uma composição semelhante à do vidro. O plástico tem a vantagem de ser mais flexível mas tem menos pureza e portanto oferece uma maior resistência à propagação da luz. A sílica é mais cara mas permite fabricar canais baseados em fibras ópticas com várias centenas de quilómetros de comprimento.

Os impulsos luminosos são gerados pelas interfaces usando LEDs (*light emitting diodes*) ou lasers. A tradução no sentido inverso é realizada por sensores de luz. Devido a fenómenos de refracção, a luz que circula na fibra não sai da mesma e a luz externa não penetra no interior. Em resultado dos dois fenómenos, obtém-se um meio praticamente imune ao ruído e que, caso fibra seja de pureza adequada, mantém a qualidade do sinal mesmo a grandes distâncias. Na verdade, o ponto fraco deste condutor são as junções, pelo que a fusão de duas fibras requer equipamento e cuidados especiais. As ligações através de fichas nos canais de fibra também são uma fonte de degradação da qualidade do sinal luminoso.

Cada fibra tem alguns micrões de espessura (1 mícron é mil vezes mais pequeno que um milímetro). A Figura 2.7 mostra exemplos de fibras agrupadas em cabos que lhes dão resistência mecânica e às intempéries (e também às agressões de roedores!), um chicote de fibra e fichas de ligação a interfaces. Na prática, como o custo de instalação é uma fracção muito significativa do custo total, os cabos de fibra que se instalaram têm vários pares de fibras agrupados, especialmente os que cobrem grandes distâncias, que podem chegar a conter mais de uma centena de fibras.

Os cabos de fibra óptica são sinónimo de comunicação de alta capacidade, com baixa taxa de erros e a grande distância. A maioria destes canais funcionam a 1, 10, 40 ou mesmo centenas de Gbps.

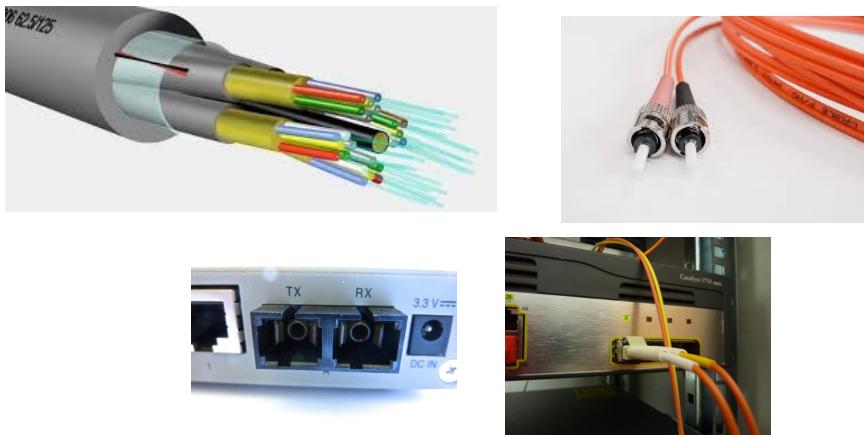


Figura 2.7: Cabo com várias fibras ópticas, “chicote” de fibra óptica e fichas de ligação a interfaces. Cada canal *full-duplex* usa um par de fibras (TX - Transmitir, RX - Receber)

Ao contrário dos fios de cobre, o limite superior do débito de um canal baseado em fibra é sobretudo limitado pela qualidade das interfaces nas extremidades do que pelo meio de propagação propriamente dito. Os *backbones* nacionais e internacionais, assim como os cabos submarinos, são todos baseados em fibras ópticas. Com a generalização da sua utilização, o preço de instalação nas ruas das cidades, e mesmo em casas, tem descido e pode vir a substituir completamente os cabos coaxiais e os cabos telefónicos existentes fora dos edifícios.

Meios não guiados – atmosfera

Como é fácil de reconhecer, os canais guiados têm o defeito de obrigar a instalar e arrastar cabos. Isso é particularmente incômodo e inadequado quando os nós se movem. Mesmo nos casos em que os nós estão fixos, é também fácil perceber que não usar cabos é muito mais cômodo e menos confuso, como ilustra a Figura 2.8.

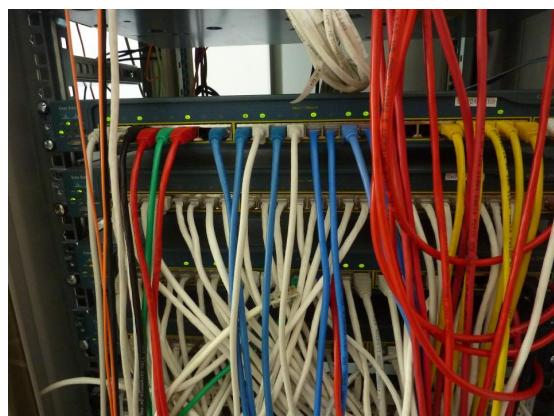


Figura 2.8: Interligação de muitos computadores através de vários comutadores pondo em evidência a complexidade da gestão dos cabos UTP nesse cenário

Por estas razões existem inúmeros canais de comunicação que usam a atmosfera como meio de propagação do sinal. Estes canais requerem que as interfaces estejam ligadas a antenas (que às vezes estão escondidas dentro do próprio nó). A Figura 2.9 mostra antenas usadas em canais sem fios (*wireless*) e permite pôr em evidência a grande diversidade de âmbito e características deste tipo de canais.



Figura 2.9: Antenas de canais sem fios – Wi-Fi e operador de telecomunicações

Como não há “bela sem senão”, a flexibilidade no manuseamento do meio (a atmosfera) é acompanhada de desafios delicados à propagação dos sinais (rádio). Isto é particularmente significativo dado que o meio não é guiado e suporta em simultâneo vários canais na mesma região, que podem interferir uns com os outros.

A qualidade do sinal depende da sua intensidade e da qualidade da antena emissora, da sensibilidade e qualidade da antena do receptor, e também da frequência usada. Dependendo desta, as ondas rádio podem ou não propagar-se mais longe e podem ou não ser capazes de atravessar certos obstáculos (paredes, edifícios, montanhas, ...). Alguns dos obstáculos provocam reflexões que fazem com que o receptor receba várias versões do sinal emitido e se “confunda” (fenómeno designado por *fading*). Todos estes factores conspiram para diminuir a qualidade do sinal e condicionam a distância e as antenas que é necessário usar. Nos canais sem fios, especialmente naqueles que usam antenas simples, a qualidade do sinal degrada-se muito rapidamente com o quadrado da distância e a taxa de erros é frequentemente muito significativa. Estes canais estão, desse ponto de vista, nos antípodas das fibras ópticas.

Acresce que quando existem vários emissores que partilham o mesmo meio, é necessário coordená-los para não interferirem uns com os outros. Algumas vezes isso pode ser resolvido reservando frequências para os diferentes operadores – um bem escasso e portanto caro – e usando sistemas de coordenação complexos – o que torna ainda mais cara a operação desses canais. Esta é a solução adoptada na generalidade das redes móveis operadas por operadores de telecomunicações.

A alternativa consiste em permitir uma espécie de “caos organizado”, que só é realista se a potência do sinal emitido for muito baixa, o que implica canais de curto alcance. Com efeito, o aumento do âmbito e da escala tornaria a gestão do caos impossível e a liberdade de aumentar a potência do sinal desencadearia uma competição sem fim. Esta é a solução adoptada nas redes baseadas em canais sem fios de curto alcance (dezenas de metros no máximo) cujo nome popular é redes Wi-Fi. Estes canais só podem emitir sinais de baixa potência e estão sujeitos às interferências de outros utilizadores e outros dispositivos que usem exactamente as mesmas frequências, ou muito próximas.

Os canais sem fios têm uma elevada taxa de erros que é compensada com retransmissões ou com códigos de correcção de erros ao nível do próprio canal. Adicionalmente, os diferentes emissores interferem uns com os outros na competição para acesso ao meio. O resultado final é que a velocidade de transferência real nô a nó (extremo-a-extremo) é sempre muito mais baixa que a velocidade de transmissão de pico anunciada.

Meios não guiados – canais por satélite

Grosso modo, e de forma algo simplista, um satélite de comunicações pode ser visto como um conjunto de antenas e amplificadores de sinal em órbita. As antenas do satélite reflectem para a terra os sinais que recebem vindos dos emissores terrestres. Assim, um emissor de sinal com uma antena adequada emite um sinal para o satélite, para que este o ecoe na sua “área de cobertura”. Esta área de cobertura é uma superfície terrestre com dezenas a milhares de quilómetros quadrados. Um receptor na área de cobertura e munido de uma antena adequada, recebe o sinal ecoado pelo satélite. A Figura 2.10 ilustra de forma simplificada o funcionamento de um canal suportado num satélite.

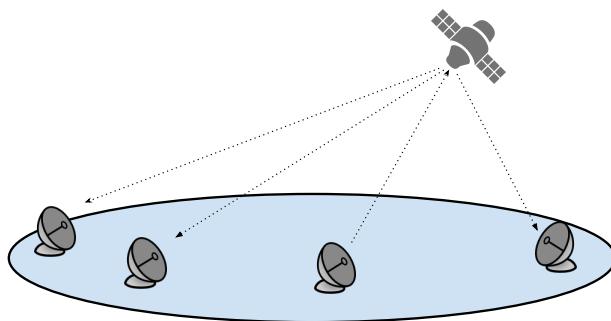


Figura 2.10: Canal de dados baseado em satélite

Pela natureza do dispositivo, os canais por satélite são muito adequados para realizar canais de difusão de sinal de televisão, suportando a transmissão de sinal de um emissor para milhares de receptores. Um só destes satélites pode transmitir milhares de canais de televisão para áreas cobrindo vários países. No entanto, também podem servir para realizar canais de comunicação de dados ponto-a-ponto ou multi-ponto com vários nós emissores. Neste último caso é necessário coordenar os emissores para que só um emita de cada vez em cada frequência.

Independentemente da forma de funcionamento, estes canais têm por característica terem uma dimensão muito apreciável. Com efeito, a forma mais simples de evitar que as antenas terrestres tenham de “seguir” satélites, o que as encareceria, consiste em usar satélites de comunicações em órbitas geoestacionárias. Estas órbitas estão a cerca de 37.000 Km de altura, fora da atmosfera, o que implica que os canais satélite geoestacionários tenham cerca de 70.000 Km de comprimento. Como a luz se propaga no espaço a cerca de 330.000 Km / s, o sinal leva cerca de 250 milissegundos a percorrer o canal do emissor ao receptor, o que corresponde a um RTT de cerca de meio segundo.

Os canais por satélite que suportam canais de dados usam esquemas sofisticados para suportarem simultaneamente muitos canais ponto-a-ponto, assim como canais multi-ponto de menor velocidade de transmissão. Os canais ponto-a-ponto podem usar velocidades de transmissão até várias dezenas de Mbps, com tempos de propagação de cerca de 250 milissegundos (RTT de cerca de 500 milissegundos), e usam antenas de dimensão significativa (de 1 a vários metros de diâmetro). Os canais multi-ponto têm geralmente velocidades de transmissão inferiores a 10 Mbps, usam antenas de menor dimensão e o mecanismo de coordenação pode implicar que o tempo de propagação duplique para 500 milissegundos (RTT de cerca de 1 segundo).

Transmissão em banda de base e em banda do canal

Para fecharmos esta breve visita aos suportes dos canais de dados, vamos abordar uma forma de partilhar o mesmo meio físico de propagação por vários canais distintos.

Para codificar a informação a transmitir pelo canal, a forma mais simples consiste em emitir um sinal eléctrico para o meio de transmissão, em que a tensão eléctrica varia segundo um padrão digital, directamente relacionado com a sequência de bits a emitir. Esta forma de transmissão diz-se **transmissão na banda de base** (*base-band transmission*), é usada com suportes baseados em fios de cobre com dimensão reduzida e está ilustrada na onda superior da Figura 2.11.

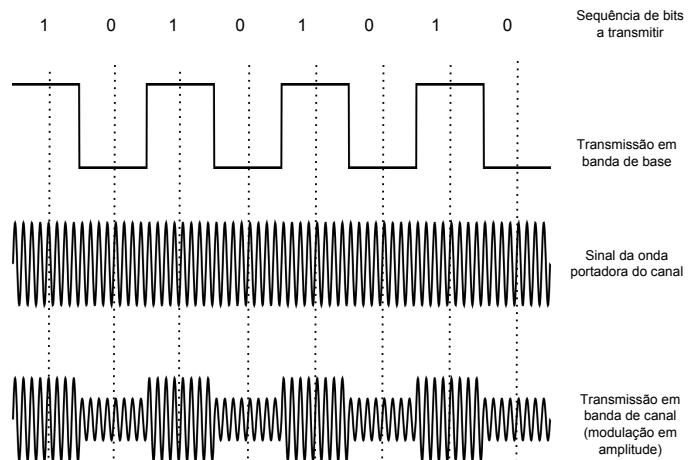


Figura 2.11: Codificação de uma sequência alternada de 0s e 1s em banda de base e em banda de canal com modulação em amplitude da portadora

Em canais de maior dimensão, e obrigatoriamente nos canais sem fios, a frequência de emissão está constrangida pelo tipo de canal e pela gama de frequências que lhe foi afectada. Por exemplo, alguns canais Wi-Fi têm de transmitir em torno dos 2,4 GHz.⁶ Nestes canais, a informação digital a transmitir é codificada em variações das características de uma onda, geralmente sinusoidal, com a frequência afectada ao canal. A esta onda de base chama-se a **portadora do canal** pois é a mesma que “transporta”

⁶Nas ondas de rádio e televisão é habitual usar a unidade Hertz (Hz) como sinónimo do número de vezes por segundo que a onda muda de sinal.

a informação. Pequenas variações da frequência, da amplitude ou da fase da portadora codificam a informação digital a transmitir. A Figura 2.11 ilustra na parte inferior a modulação em amplitude da portadora. Este modo de transmissão chama-se **transmissão na banda do canal** ou na **frequência da portadora** (*carrier modulated transmission*).

Usando transmissão em banda de canal e utilizando ondas portadoras distintas, vários canais diferentes podem partilhar o mesmo meio físico de propagação. Por exemplo, nas redes Wi-Fi, diferentes canais podem partilhar o mesmo espaço sem interferirem uns com os outros, usando frequências portadoras distintas. Nas fibras ópticas pode-se usar o mesmo princípio usando diferentes cores de impulsos luminosos associados a cada canal (as variações de frequência nas portadoras desses canais correspondem a variações da cor da luz).

A utilização no mesmo meio de propagação físico de várias portadoras distintas com frequências diferentes permite dividir um canal em sub-canais distintos e chama-se **multiplexagem por frequência** (*frequency division multiplexing - FDM*) do meio de transmissão.

2.4 Detecção de erros

Existem inúmeras razões que podem levar a que um *frame* seja corrompido entre a emissão e a recepção. O número total de bits errados na mensagem recebida diz-se o comprimento do erro.

Os erros costumam ser classificados em **erros de um bit** e **erros em cadeia** (*bit errors* e *burst errors*). Os erros de um bit são geralmente provocados pela presença de ruído de fundo, também conhecido como ruído branco, que aleatoriamente degrada a relação do sinal útil com o ruído de fundo aleatório, e leva a interface receptora a “confundir-se” e a trocar esporadicamente um bit na recepção. Este tipo de erros são raros nas fibras ópticas, mas são mais frequentes noutros canais mais vulneráveis a interferências. A atenuação do sinal devida à distância percorrida também degrada a relação sinal / ruído e aumenta a probabilidade do aparecimento destes erros. A Figura 2.12 ilustra como a degradação do sinal tem lugar e pode conduzir ao aparecimento de erros de recepção.

Os erros em cadeia afectam conjuntos de bits próximos e têm a ver com eventos aleatórios que afectam de forma incisiva o sinal durante a sua progressão no canal: radiações esporádicas que interferem na gama frequências usada pelo canal, contactos eléctricos com deficiências mecânicas intermitentes, sujidade em contactos ópticos, ecos de sinais, *etc.*

Os progressos conseguidos nos últimos anos, sobretudo com a generalização da utilização da fibra óptica, diminuíram muito a probabilidade de ocorrerem erros (excepto nos canais sem fios). Outro progresso tem a ver com a introdução de algoritmos e técnicas de detecção de erros (*error detection*) eficientes, que diminuíram de forma significativa a probabilidade de as interfaces dos canais não detectarem os erros que ocorram. De qualquer forma, os protocolos de mais alto nível, assim como algumas aplicações, usam técnicas suplementares de detecção de erros extremo-a-extremo, ou seja, implementadas nos computadores finais em comunicação. Apesar de às vezes algumas delas poderem parecer inúteis, ou até ingénugas, quando comparadas com as usadas nos canais, nesta secção estudaremos também algumas dessas técnicas extremo-a-extremo usadas nas redes TCP/IP.

As aplicações com requisitos importantes de segurança usam canais lógicos criptograficamente protegidos e aplicam também técnicas de verificação criptográfica das

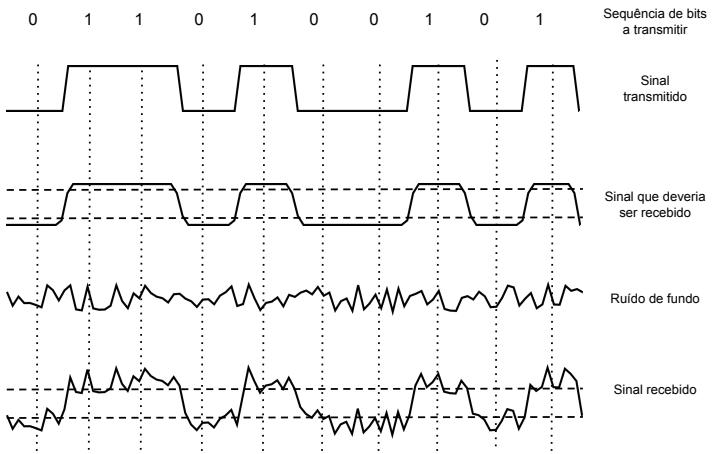


Figura 2.12: O sinal digital original emitido é degradado, quer pela propagação do mesmo pelo canal, quer pela presença de ruído no canal

transferências de dados, que são muito robustas na detecção de erros provocados por atacantes sofisticados e, por maioria de razão, dão níveis de garantia que ultrapassam em muito aqueles que as técnicas de detecção de erros ao nível canal podem proporcionar, pois estes últimos devem-se apenas a fenômenos aleatórios e não a atacantes que exploram fragilidades dos códigos de detecção de erros.

De qualquer forma, a detecção dos erros ao nível dos canais, não sendo decisiva para garantir segurança de extremo-a-extremo, é fundamental para melhorar o desempenho da rede em geral, como veremos nos capítulos dedicados ao problema da troca fiável de dados, na segunda parte deste livro.

Erros e suas probabilidades

A qualidade de um canal do ponto de vista da sua resistência a erros de transmissão (detectados ou não) é caracterizada pela probabilidade de um bit chegar errado, também conhecida como taxa de erros ao nível do bit ou *link bit error rate*. Esta probabilidade deve ser medida em intervalos, por exemplo, uma vez em cada segundo.

A **taxa de erros de um canal (*link bit error rate*)**, em termos de bits, é o número de bits erradamente recebidos por unidade de tempo. Esta média, medida em intervalos de, por exemplo, uma vez em cada segundo, constitui a probabilidade de um bit chegar errado ao receptor.

Admitindo que a ocorrência de um erro num bit é um acontecimento independente de ocorrer outro bit (o que nem sempre é verdade), e que uma mensagem tem n bits, então a probabilidade dessa mensagem chegar sem erros é a probabilidade de o primeiro bit não chegar errado e o segundo não chegar errado, até ao enésimo bit não chegar errado, e a probabilidade de a mensagem chegar com erros é 1 menos essa probabilidade.

Dados:

p a probabilidade de um bit chegar errado (*Bit Error Rate ou BER*)

n a dimensão de uma mensagem M a transferir

A probabilidade de M chegar SEM erros é $(1 - p)^n$
e a probabilidade de M chegar COM erros é $1 - (1 - p)^n$

Por exemplo, admitindo que $p = 10^{-7}$ e que $n = 10^4$, a probabilidade da mensagem chegar SEM erros é $(1 - 10^{-7})^{10.000}$ e a probabilidade de chegar COM erros é o complemento $1 - (1 - 10^{-7})^{10.000} = 0,0009995$.

No entanto, este resultado pode ser facilmente aproximado considerando, erradamente, que a probabilidade de ocorrência de um qualquer evento entre n eventos, cada um dos quais de muito baixa probabilidade de ocorrência (p), é $n \times p$. Nesse caso o resultado anterior seria facilmente calculado como sendo $p = 10^4 \times 10^{-7} = 10^{-3} = 0,001$.

Com efeito, se $n = 2$ e $p = 10^{-3}$, a probabilidade de a mensagem chegar sem erros é $(1 - 10^{-3}) \times (1 - 10^{-3}) = 1 - 2 \times 10^{-3} + 10^{-6}$ e a probabilidade de a mensagem chegar com erros é $1 - (1 - 10^{-3} - 10^{-3} + 10^{-6}) = 2 \times 10^{-3} - 10^{-6}$. Um resultado final aproximado não consideraria a parcela -10^{-6} tanto mais que na prática $p \ll 10^{-7}$ hoje em dia.

Técnicas de detecção de erros

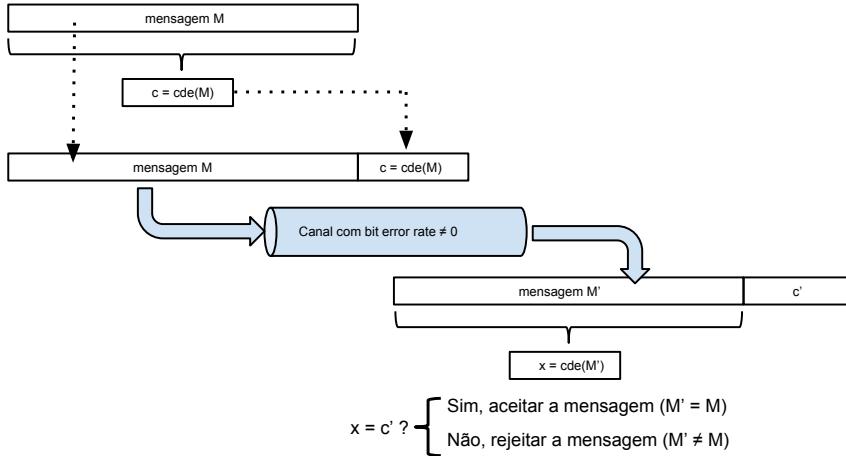


Figura 2.13: Utilização de um código de detecção de erros para controlar se uma mensagem é ou não alterada durante a transmissão pelo canal

A detecção de erros baseia-se num princípio fácil de enunciar e ilustrado na Figura 2.13. Dada uma mensagem M a transmitir, aplica-se a M a função cde , dita de cálculo do código de detecção de erros ($c = cde(M)$), e transmite-se a mensagem original concatenada com o código calculado ($M||c$). Seja $M'||c'$ a mensagem recebida. Calcula-se $x = cde(M')$ e considera-se que $M' = M$ caso $x = c'$. Ou seja, considera-se que a mensagem foi recebida sem erros, se após se recalcular o código de detecção de erros, o mesmo coincide com o recebido com a mensagem.

A mais simples função de detecção de erros é a função identidade, que corresponderia na prática a enviar duas cópias de cada frame. Por um lado, o custo deste tipo

de código seria muito elevado, pois em cada dois bits transmitidos apenas um corresponderia a informação útil, e por outro lado existem alguns erros, provavelmente raríssimos, que a função não detectaria (quais?). O grande desafio é detectar o máximo de erros com um número mínimo de bits redundantes.

Três exemplos de como se pode tentar resolver este desafio são apresentados de seguida.

Bits de paridade

A referência a esta técnica de detecção de erros tem apenas um papel ilustrativo e histórico.

Os bits de paridade foram introduzidos em canais cuja unidade base de informação transferida era uma pequena sequência de bits, geralmente 7 bits, aos quais se juntava um bit extra, geralmente o oitavo bit, dito bit de paridade. O valor do bit de paridade garantia que o número total de bits com valor 1 nos 8 bits transmitidos era par (esta opção é designada por paridade par) ou ímpar (esta opção é designada por paridade ímpar).

Esta técnica apenas permite detectar erros de comprimento 1 e alguns erros de comprimento maior. Com 1 bit extra de paridade por cada 7 bits transmitidos, o custo desta redundância é de $1/8 = 12,5\%$.

Soma de controlo – *Checksum*

Um outro tipo de código de controlo de erros é o *Internet checksum*, assim designado por ser usado no protocolo IP para proteger o cabeçalho, e nos protocolos TCP e UDP para proteger toda a mensagem, ver o RFC 1071.

A ideia do *Internet checksum* é simples e consiste em ver o pacote como uma sequência de palavras de 16 bits (a unidade base da memória dos computadores mais simples da época), fazer a soma dessas palavras, e colocar o resultado final, *i.e.*, o código de controlo de erros, também numa palavra de 16 bits. O método de soma usado é uma soma de complemento a 1. Quando a soma intermédia dá *overflow* (*i.e.*, há um *carry bit*) o valor do resultado é incrementado de 1, mas tomam-se sempre os 16 bits menos significativos para continuar.

O método Java abaixo calcula o *Internet checksum* de um pacote contido no vector de bytes `data`, que se assume ter um número par, e maior de zero, de bytes (se antes era ímpar, pressupõe-se que foi acrescentado um último byte com o valor zero). O algoritmo normalizado requer que o emissor envie, como código de detecção de erros, o complemento a 1 da soma, e por isso o resultado é invertido antes de ser retornado pelo método.

Listing 2.1: Algoritmo de cálculo do *Internet Checksum*

```
int checksum (byte[] data) {
    int sum = 0;
    int i = 0;
    for (;;) {
        sum = sum + byte[i]<<8 + byte[i+1];
        if ( sum & 0xFFFF000 > 0 ) { // a carry bit occurred
            sum &= 0xFFFF;
            sum++;
        }
        i += 2;
        // when finished return the ones complement of the sum
        if (i > data.length) return ~ (sum & 0xFFFF);
    }
}
```

O custo desta soma de controlo é uma palavra de 16 bits por pacote enviado. Num pacote de 1000 bytes o desperdício é de 0,2%.

Teste cíclico redundante – *Cyclic Redundancy Check (CRC)*

O método que vamos introduzir a seguir é o mais interessante pois, quando utilizado com os parâmetros adequados, detecta um conjunto muito alargado de erros. Está normalizado, é implementado hoje em dia por hardware banalizado, e é utilizado em muitos canais actuais com um código de 32 bits que conduz, num *frame* de 1000 bytes, a um desperdício de apenas 0,4%.

Os códigos de detecção de erros cíclicos redundantes, de comprimento n , são calculado como sendo o resto da divisão em aritmética módulo 2 de um número equivalente à mensagem m a ser transmitida, deslocado para a esquerda de n bits (esta operação é equivalente ao resultado de $m \times 2^n$), por um número especial de $n + 1$ bits de comprimento, chamado o *divisor* ou *gerador* do código.

A teoria destes códigos ultrapassa os objectivos deste livro, mas de forma simplificada podemos considerar:

m – a mensagem original que se pretende transmitir (m tem k bits),

d – o divisor (demonstra-se que este tem de ter $n + 1$ bits),

q – o resultado de $(m \times 2^n) / d$ (quociente),

r – o resto da divisão de $(m \times 2^n) / d$ (resto), e

crc – o código de controlo de erros a enviar (crc tem $n < k$ bits).

O que se pretende é calcular um valor de crc , o código de controlo de erros, tal que a mensagem a ser transmitida, $m \times 2^n + crc$, dividida pelo divisor d em aritmética módulo 2 tem resto zero. O emissor calcula esse valor como sendo o resto da divisão de $m \times 2^n$ por d , o qual usará como valor de crc , enviando a mensagem $m \times 2^n + crc$. Em aritmética módulo 2, o resto da divisão módulo 2 por um número com $n+1$ bits tem no máximo n bits. Ao receptor cabe verificar se o resto da divisão da mensagem recebida pelo divisor d tem resto 0. Se sim, a mensagem recebida tem elevada probabilidade de não ter erros, senão tem erros com probabilidade 1. A Figura 2.14 ilustra a utilização do método.

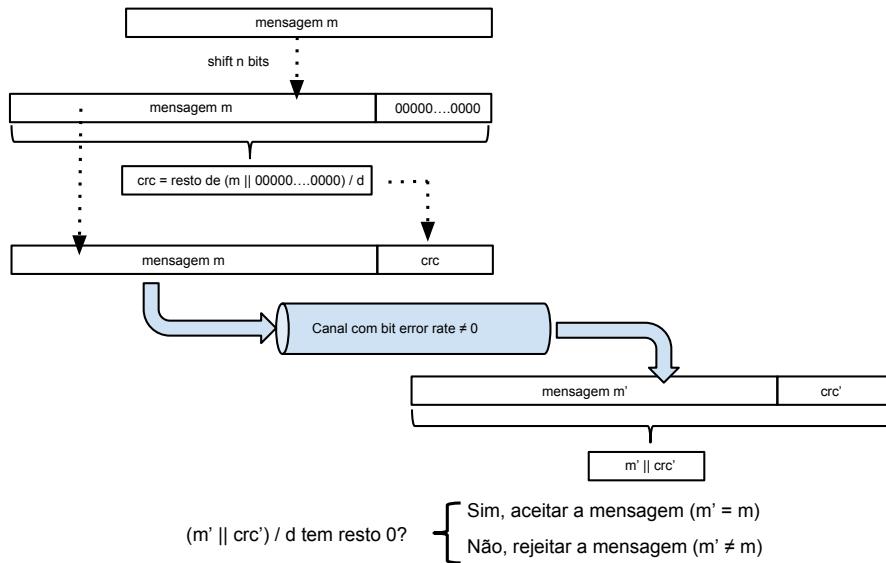


Figura 2.14: Cálculo do código de controlo de erros com o método *Cyclic Redundancy Check (CRC)* – ' \parallel ' denota concatenação

Para garantir que este método de cálculo do *crc* conduz ao objectivo pretendido é preciso garantir que com o *crc* calculado como o resto da divisão $(m \times 2^k)/d$, então $(m \times 2^k + crc)/d$ tem resto zero. Mas também, que se existirem erros à saída do canal, repetindo a mesma operação, esse resto é diferente de 0.

Comecemos pela primeira questão. Dividindo $m \times 2^n + r$ por d , obtemos:

$$\frac{m \times 2^n + r}{d} = \frac{m \times 2^n}{d} + \frac{r}{d} \quad (2.1)$$

como

$$\frac{m \times 2^n}{d} = q + \frac{r}{d} \quad (2.2)$$

resulta que

$$\frac{m \times 2^n + r}{d} = \frac{m \times 2^n}{d} + \frac{r}{d} = q + \frac{r}{d} + \frac{r}{d} \quad (2.3)$$

mas como em aritmética módulo 2 a soma de qualquer número com ele próprio tem como resultado 0, temos

$$\frac{m \times 2^n + r}{d} = q + \frac{r}{d} + \frac{r}{d} = q \quad (2.4)$$

como não há resto, daí resulta que a mensagem a transmitir é divisível pelo divisor, *i.e.*, a divisão tem resto 0.

Assim, o emissor calcula o código *crc* a transmitir como sendo o resto da divisão por d , da mensagem a transmitir *shifted* de n bits para a esquerda. O receptor aceita a mensagem recebida como não tendo erros se o resto da divisão da mesma pelo divisor d tiver resto 0.

Demonstra-se que com uma escolha adequada do divisor d , o método permite detectar todos os erros de comprimento 1 e 2, todos os erros com um número par de bits errados, todos os erros em cadeia de comprimento inferior ao número de bits do *crc* (ou seja n) e mais um conjunto significativo de erros em cadeia de comprimento superior.

Estão normalizados códigos cíclicos redundantes com comprimento 12, 16 e 32 bits e os respectivos divisores (d). O uso de 32 bits de código de controlo de erros do tipo *Cyclic Redundancy Check* é considerado adequado nos canais que usam *frames* de comprimentos significativos (por exemplo 1500 bytes). Como a implementação em hardware está banalizada e exibe boas capacidades de detecção de erros, a sua utilização é muito frequente nos canais actuais.

É fácil de aceitar que um código de controlo de erros como o *Internet checksum* não é totalmente seguro e que o mesmo representa um compromisso apenas válido numa dada época, como se pode verificar pela data de edição (1988) do respectivo RFC. Com efeito, o *Internet checksum* foi concebida numa época em que os canais usavam geralmente sistemas ainda mais deficientes de detecção de erros (como os bits de paridade ilustrados acima), com o objectivo de dar mais segurança, extremo-a-extremo, de que não haveria erros nos pacotes recebidos. Nessa altura só era realizável que estes códigos de controlo de erros fossem calculados por software, e por isso a soma de controlo baseia-se num conjunto de somas cuja implementação em software é eficiente. A opção tomada teve também como repercução não impor demasiadas restrições, do ponto de vista da detecção de erros, aos canais que se podiam usar para transportar pacotes IP. Mais tarde foram inventados outros códigos baseados em somas de controlo mais seguros também facilmente implementáveis em software [Feldmeier, 1995], mas nessa altura já era muito tarde para mudar as normas dos protocolos IP, TCP e UDP.

Modernamente, os códigos de detecção de erros do tipo CRC são os preferidos, mas a sua implementação por software é mais pesada pois o seu cálculo implica divisões de números representados num número muito elevado de bits. Como quase todos os canais modernos usam códigos de detecção de erros sofisticados do tipo CRC, implementados

por hardware, a problemática da segurança do *Internet checksum* deixou de ser tão premente, tanto mais que as aplicações com elevados requisitos de segurança usam elas próprias mecanismos criptográficos de detecção de erros muito mais seguros que qualquer dos códigos acima discutidos e cuja implementação se apoia em instruções especiais dos processadores modernos.

Na versão mais recente do protocolo IP, a versão 6 do protocolo, o cabeçalho dos pacotes IP deixou de conter um código de protecção de erros do cabeçalho por se acreditar que o controlo de erros exercido ao nível dos canais é suficiente para esse efeito.

Códigos de correcção de erros

Existem códigos que permitem, para além da detecção de erros, determinar quais os bits que estão errados. Uma forma ingénua de implementar este tipo de códigos poderia consistir em enviar 3 cópias de cada *frame* e depois decidir, quando se detectava que 1 bit aparecia com um valor distinto em alguma das cópias, pelo valor do bit que estivesse em maioria nas três cópias. Para além de esta técnica não conseguir anular completamente a possibilidade de uma decisão errada, apesar de tomada por maioria, de cada 3 bits enviados apenas um corresponderia a informação útil. Estariam perante um desperdício de cerca de 66% da capacidade do canal.

Os **códigos de correcção de erros** (*error correction codes*) são mais complexos que os códigos de detecção de erros, e também menos eficientes, pois exigem mais bits redundantes. Como a probabilidade de existirem erros nos canais tem caído significativamente, é preferível utilizar apenas códigos de detecção de erros pois é menos oneroso retransmitir esporadicamente um *frame*, do que penalizar em excesso todos os *frames* transmitidos com um número significativo de bits redundantes adicional.

No entanto, esta linha de raciocínio não se aplica nos canais sem fios onde são usados em alguns casos códigos de correcção de erros. De facto, nestes canais a probabilidade de existirem erros continua a ser significativa, pelo que continuam a ser usadas técnicas especiais de detecção e correcção de erros cada vez mais sofisticadas. O leitor interessado poderá consultar [Stallings, 2011; Zaragoza, 2002] por exemplo.

No capítulo 9, secção 9.3, são introduzidas técnicas de correcção de erros, usadas ao nível aplicacional, em aplicações que toleram alguma degradação da informação recebida. É o caso de algumas aplicações multimédia, em que os erros podem ser parcialmente compensados por informação redundante, mas de resolução inferior à errada ou em falta. Na mesma secção são também referidos códigos de correcção de erros sem degradação da qualidade, especialmente adequados para aplicações de difusão de informação de 1 para n .

2.5 Exemplo – canais Ethernet

Durante os anos 70 do século passado nasceu no Xerox Palo Alto Research Center, no Silicon Valley nos EUA, a primeira rede de alta velocidade para o interior de edifícios, baptizada pelo seu inventor, Bob Metcalfe, com o nome de rede Ethernet. Este trabalho teve várias evoluções, que se traduzem hoje em dia na existência de vários canais de dados (e até redes completas) que herdaram o nome e algumas das características da proposta inicial. Actualmente, os canais Ethernet usam cabos UTP e fibras ópticas, e as suas capacidades variam de 10 Mbps até 100 Gbps ou mais. Esta variedade de canais usa diferentes meios de transmissão e diferentes técnicas de

codificação, alguns são ponto-a-ponto, outros multi-ponto, e variam ainda em mais alguns detalhes, mas todos partilham um formato de *frame* grosso modo semelhante. Para mostrar um exemplo concreto de canal de dados, vamos detalhar a seguir algumas facetas dos canais Ethernet, na versão normalizada segundo a norma IEEE 820.3, que usa geralmente cabos UTP para transmitir os dados. Designaremos esses canais genericamente como canais Ethernet.

Os canais Ethernet norma IEEE 802.3

Os *frames* Ethernet têm o formato indicado na Figura 2.15. O preâmbulo contém 8 bytes, dos quais, os primeiros 7 têm o valor 10101010, e o último o valor 10101011. Este preâmbulo tem por papel alertar o receptor de que vai começar a receber um novo *frame*, dá-lhe a oportunidade de (voltar a) sincronizar o seu oscilador (relógio) com o do emissor, para realizar correctamente a amostragem do sinal, e tem um sinal no último byte que assinala onde começa o cabeçalho do *frame*.

A técnica de codificação usada permite ao receptor distinguir entre a ausência de transmissão (canal em repouso) e o início de um *frame*, e permite-lhe também reconhecer o seu fim.



Figura 2.15: Estrutura do *frame* Ethernet na norma IEEE 802.3

O cabeçalho tem três campos: dois com 6 bytes (48 bits) cada um, que são os endereços origem e destino, e o terceiro campo, designado como tipo, com 2 bytes. Os canais Ethernet 802.3 podem funcionar em modo ponto-a-ponto ou multi-ponto. Neste último caso, ao enviar um *frame*, o nó emissor tem de discriminar o destinatário entre todos os que estão ligados ao mesmo canal, e o nó receptor necessita de saber o endereço do nó que lhe enviou o *frame*. O formato e a forma de afectação dos endereços está também normalizada. O campo tipo tem por papel facilitar a vida ao receptor indicando-lhe a que protocolo de nível superior pertencem os dados. Assim, o receptor pode imediatamente saber que módulos software deve usar para interpretar os dados contidos no *frame*.

A seguir vêm os dados, os quais podem ocupar desde a dimensão mínima de 46 bytes até à dimensão máxima de 1500 bytes. O facto de a dimensão máxima ser 1500 bytes será discutida mais abaixo. A imposição de uma dimensão mínima tem a ver com o modo de funcionamento multi-ponto. Se o emissor pretende enviar dados que ocupem menos que a dimensão mínima, terá de completar o campo de dados com informação irrelevante. Cabe ao software de tratamento do protocolo a que pertencem os pacotes contidos no campo de dados distinguir qual a parte que é válida. Por exemplo, se na parte de dados for um pacote IP, o cabeçalho do pacote IP contém um campo com a dimensão do pacote.

Finalmente, o *frame* termina com um campo com um código de controlo de erros com 32 bits (4 bytes) que utiliza um código do tipo *Cyclic Redundancy Check* (CRC), como apresentado na secção anterior. Quando a interface do receptor reconhece a presença de um *frame*, após a sua recepção completa testa o código CRC. Se o teste falhar, o *frame* é ignorado e espera-se pelo próximo, pois estes canais não incluem

nenhum mecanismo de recuperação de erros nem de controlo de fluxo. Esta opção tem a ver com o facto de que se pretende um canal simples e flexível, e as taxas de erro das diferentes variantes do mesmo serem tão baixas que não se justifica a complexidade adicional. Assim, foi deixada aos níveis superiores a incumbência de tratar da ausência de *frames*.

Dimensão máxima dos *frames* nos canais

Em muitos canais ponto-a-ponto não são impostas dimensões mínimas à parte de dados dos *frames*, mas todos os canais impõem uma dimensão máxima. A pergunta que se coloca é a seguinte: que factores condicionam este valor? À primeira vista seria desejável não ter limitações pois isso facilitaria a vida ao nível rede e ao nível aplicacional que poderiam usar qualquer dimensão de *frame*. Demonstra-se que isso permitiria melhorar o desempenho dos canais lógicos TCP e diminuir a utilização da CPU dos computadores, sobretudo com canais de muito alta velocidade [Murray and Dixon, 2012]. Mas na verdade essa ausência de limites não é possível pelas seguintes razões:

Partilha equitativa Um canal é partilhado por vários fluxos de comunicação pertencentes a vários computadores e aplicações. Em geral, um *frame* contém um único pacote pertencente a um único fluxo e, enquanto este pacote está a ser transmitido, esse fluxo monopoliza o canal. É indesejável que essa afectação se prolongue no tempo. Por exemplo, num canal de baixa velocidade, com a capacidade de 10 Kbps, um *frame* com 10.000 bits monopolizaria o canal durante pelo menos 1 segundo. É um problema da mesma natureza que o da partilha de um processador por vários processos num sistema de operação. Nos canais de alta velocidade este problema está minorado.

Sincronização emissor / receptor Para reconhecer os bits, o receptor tem de analisar periodicamente amostras do sinal recebido, a intervalos de tempo muito precisos e idênticos aos usados pelo emissor para produzir o sinal. Apesar de ambas as interfaces funcionarem ao mesmo ritmo, os osciladores dos relógios não oscilam exactamente à mesma frequência e as interfaces necessitam de voltar a ser sincronizadas de tempos a tempos. Geralmente isso acontece no início de cada *frame*. Ora quanto maior for este, maior a probabilidade de os dois relógios se afastarem, o que aumenta a possibilidade de o receptor interpretar erradamente o sinal recebido.

Taxa de erros Como vimos atrás, a maioria dos canais com baixa taxa de erros não usa códigos de correcção de erros, apenas códigos de detecção de erros. Se um erro é detectado na recepção, todo o *frame* é ignorado e tem de ser retransmitido mais tarde quando a sua falta for detectada pelos níveis superiores. Naturalmente, quanto maiores forem os *frames*, maior a probabilidade de os mesmos chegarem com erros e o desperdício ser superior.

Assim, durante a fase de concepção de um novo canal, tendo em consideração factores como os acima listados, é escolhida uma dimensão razoável para o maior campo de dados dos *frames* que podem ser transmitidos, e esse valor é fixado na norma (do *standard*) do canal. Esse valor máximo designa-se por MTU (*Maximum Transfer Unit*) do canal.

Com o passar do tempo, a maioria dos canais aumentaram o débito e baixaram a taxa de erros, pelo que aquele limite pode ser cada vez maior. No entanto, na prática, reconhece-se que nos canais Ethernet actuais não seria uma boa política “liberalizar” muito o uso de MTUs superiores aos 1500 bytes da rede Ethernet inicial.

Com efeito, a maioria dos canais da periferia da rede, ou seja os que a ligam directamente aos computadores, são quase todos variantes de canais Ethernet (com e sem fios). No interior da rede os comutadores de pacotes recebem pacotes por uns canais

e transmitem-nos por outros. Caso os canais admitam valores de MTU diferentes, um comutador de pacotes poderia ter de partir (fragmentar) um pacote em vários para o encaminhar para o destino. Tal constitui uma complicação extra indesejável porque muitos comutadores têm de processar milhões de pacotes por segundo. Para evitar esta complicação suplementar aos comutadores de pacotes usados na Internet, os operadores optam, por segurança, por não permitirem a entrada na rede de pacotes com dimensão superior à referência estabelecida nos primórdios dos canais de alta velocidade. No entanto, no interior da rede, é frequente optarem por usar MTUs maiores.

Quando uma rede está confinada, todos os comutadores de pacotes e computadores estão sob a mesma autoridade, e as interfaces funcionam a velocidades de pelo menos 1 Gbps, é possível parametrizar as interfaces dos nós para usarem *frames* de maior dimensão. Esses *frames* chamam-se *Jumbo Frames* e têm um MTU com 9000 bytes ou mais, em vez dos 1500 correspondentes ao valor por omissão.

Encapsulamento de pacotes em *frames*

Os canais de dados usam protocolos para que as interfaces dos seus extremos se coordenem para enviar os *frames*. Essas interfaces e os algoritmos (software ou hardware) que as controlam não interpretam a informação que vai no campo de dados, apenas interpretam a informação que vai no cabeçalho e no sufixo (*e.g.*, o campo CRC). No entanto, no campo de dados geralmente é transmitido um pacote de dados, uma mensagem do nível rede, que também tem um cabeçalho próprio.

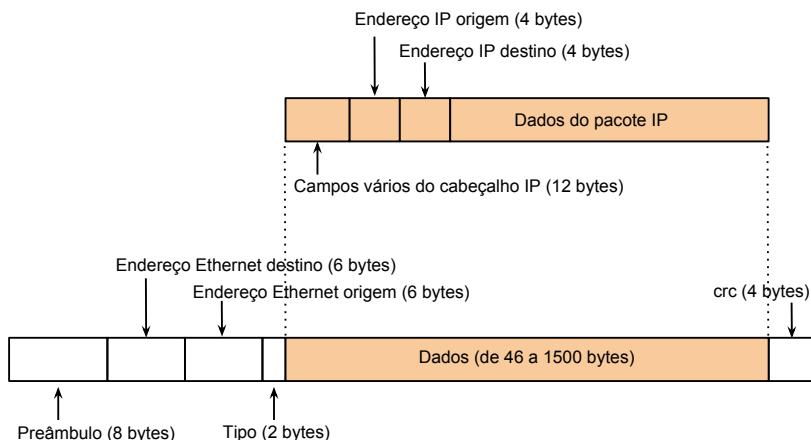


Figura 2.16: Encapsulamento de um pacote IP num *frame* Ethernet

Por exemplo na Figura 2.16 um pacote IP é transportado no campo de dados de um *frame* Ethernet. Esta técnica é designada por **encapsulamento** de mensagens umas nas outras. A mesma conduz a que o campo de dados dos *frames* do nível canal comecem geralmente por um ou mais cabeçalhos dos protocolos dos níveis superiores. Este assunto será retomado no capítulo 4.

À técnica de transportar na parte de dados de um nível (canal, rede, transporte, ...) uma mensagem (pacote, segmento, ...) de um nível superior, chama-se **encapsular** uma mensagem de um nível numa mensagem de um nível inferior.

2.6 Resumo e referências

Resumo

Uma rede de computadores é formada internamente por nós e permite-lhes trocar informação entre si, em unidades que chamamos genericamente mensagens. Ao nível dos canais de dados, que interligam os nós, essas mensagens designam-se *frames*.

Os canais são dispositivos físicos, constituídos por interfaces e meios de propagação dos sinais (eléctricos, electromagnéticos, ópticos, ou sonoros), que ligam directamente vários nós via as interfaces e o meio de propagação do sinal. Os canais podem ser ponto-a-ponto, quando só ligam dois nós, ou multi-ponto, se ligam mais do que dois nós. O meio de propagação pode ser guiado (*i.e.*, baseado em fios) ou não guiado (*i.e.*, usa a atmosfera ou o vazio).

As propriedades mensuráveis dos canais, que maior relevância têm para a sua caracterização, são a capacidade ou débito, o tempo de trânsito e a taxa de erros. O débito mede-se em bits por segundo ou bps e é determinante para calcular o tempo de transmissão de um *frame* pelo emissor.

O tempo de trânsito de um canal está directamente ligado ao tempo que leva o sinal a propagar-se no meio e depende da dimensão do canal. Em todos os canais mais comuns, excepto aqueles cujo meio de propagação é o vazio, usa-se como valor aproximado da velocidade de propagação o valor de referência de 200.000 Km/s. Nos canais cujo meio de propagação é o vazio, usa-se a velocidade de propagação da luz no vazio como referência.

A taxa de erros é o rácio dos bits que chegam errados pelo número total de bits transferidos pelo canal. Este rácio é medido em intervalos de tempo, como por exemplo de segundo a segundo. A taxa de erros do canal corresponde à probabilidade de um bit chegar errado ao destino.

Os canais actualmente existentes usam predominantemente fios de cobre ou fibras ópticas, designados meios guiados, e a atmosfera, designada meio não guiado. Neste último caso os canais dizem-se sem fios. Existem canais cujo meio de transmissão é predominantemente o espaço existente para além da atmosfera. Tratam-se de canais suportados por satélites, que se caracterizam por terem um tempo de trânsito muito elevado, de cerca de 250 ms ou mais.

Para lidar com os erros existentes nos canais e tentar evitar que estes se propaguem aos níveis superiores, as interfaces dos canais incluem mecanismos baseados em códigos de detecção e de correcção de erros. Estes mecanismos usam bits redundantes, cujo valor é calculado na emissão, e verificado na recepção. Quando se detectam discrepâncias, o *frame* recebido é simplesmente rejeitado (quando se usam códigos de detecção de erros) ou corrigido (quando se usam códigos de correcção de erros).

Os códigos de correcção de erros são mais complexos que os códigos de detecção de erros, e também menos eficientes, pois exigem mais bits redundantes. Como a probabilidade de existirem erros nos canais tem caído significativamente, é preferível utilizar apenas códigos de detecção de erros pois é menos oneroso retransmitir esporadicamente um *frame*, do que penalizar em excesso todos os *frames* transmitidos com um número significativo de bits redundantes adicionais. Este raciocínio não é aplicável quando o canal tem uma taxa de erros significativa. Por esta razão, os canais sem fios usam frequentemente códigos de correcção de erros.

Os *frames* que atravessam um canal têm um formato que obedece ao protocolo do canal. De forma geral, o seu formato compreende um preâmbulo, um cabeçalho, uma parte de dados e um sufixo, constituído geralmente por um código de correcção de erros. O preâmbulo serve para sinalizar a presença e o início do *frame*. O cabeçalho contém informação de controlo como por exemplo o tipo do *frame*, endereços (obrigatórios quando o canal é multi-ponto), e campos de controlo (obrigatórios quando o protocolo do canal inclui mecanismos de reemissão ou de controlo de fluxo).

Na parte de dados viajam as mensagens dos protocolos do nível superior (geralmente os pacotes do protocolo IP) ou de outros protocolos do nível rede. Diz-se então que os pacotes do nível superior estão encapsulados no campo de dados dos *frames* que passam nos canais.

O capítulo acaba apresentando, a título de exemplo, o formato dos *frames* dos canais Ethernet da norma IEEE 802.3 e discute porque razão todos os canais usam *frames* cuja parte de dados está limitada a uma dimensão máxima, o chamado MTU (*Maximum Transfer Unit*) do canal.

Os principais termos introduzidos ou referidos neste capítulo são a seguir passados em revista. Entre parêntesis figuram as traduções mais comuns em língua inglesa.

Canal de comunicação (*communication link, data link, link*) Um canal de comunicação de dados é um dispositivo que permite a um conjunto de nós de comunicação trocarem directamente entre si mensagens. O canal é formado por interfaces e um meio de propagação de sinal. As interfaces codificam a informação a transmitir da forma mais adequada para ser transmitida pelo meio de transmissão ou propagação. As mensagens transmitidas pelo meio de propagação chamam-se *frames*.

Canais ponto-a-ponto e multi-ponto (*point-to-point, multipoint links*) Um canal ponto-a-ponto liga apenas dois nós, um em cada extremidade. Um canal multi-ponto liga mais do que dois nós.

Canais simplex, half- e full-duplex (*simplex, half- and full-duplex links*) Alguns canais apenas permitem transmitir informação num só sentido e dizem-se canais *simplex*. Os canais mais interessantes e populares permitem transmitir informação nos vários sentidos. Quando um canal ponto-a-ponto é deste último tipo e permite que a comunicação nos dois sentidos tenha lugar em paralelo (*i.e.*, simultaneamente), o canal diz-se *full-duplex*. Caso o canal permita a comunicação em vários sentidos mas apenas suporte que esta tenha lugar num só sentido de cada vez (*i.e.*, em alternância), o canal diz-se *half-duplex*.

Débito, capacidade ou velocidade de transmissão (*bit rate, bandwidth*) O débito, capacidade ou velocidade de transmissão de um canal é a quantidade de informação (medida em bits) que o canal é capaz de transmitir por unidade de tempo.

Tempo de transmissão (*transmission time*) O tempo de transmissão de um *frame* com D bits por um canal com o débito ou capacidade de C bits por segundo, ou seja o tempo que medeia desde que começa a ser emitido o primeiro bit até que acabe de ser emitido o último bit, é D / C segundos.

Tempo de propagação (*propagation time, delay, latency*) É o tempo necessário para que o sinal se propague de uma extremidade à outra do canal.

Tempo de trânsito de um frame num canal (*end-to-end frame delay*)

Tempo de trânsito = Tempo de transmissão + Tempo de propagação.

Volume do canal (*delay bandwidth product*)

Volume do canal = Débito × Tempo de propagação.

Transmissão em banda de base (*baseband transmission*) Forma de codificação da informação a transmitir pelo canal segundo uma forma de onda digital directamente relacionada com a sequência de bits a emitir.

Transmissão em banda de canal (*carrier modulated transmission*) Forma de codificação da informação a transmitir pelo canal usando variações de características (amplitude, frequência, fase, ...) de um sinal emitido numa frequência chamada a frequência da onda portadora do canal.

Multiplexagem em frequência (*frequency-based multiplexing*) Consiste na utilização do mesmo meio de propagação físico por várias portadoras distintas, com frequências diferentes, que permitem que o meio seja partilhado por diferentes canais. Por outras palavras, a multiplexagem como que *divide um canal em sub-canais distintos*.

Taxa de erros de um canal (*bit error rate*) A taxa de erros de um canal, em termos de bits, é o número de bits errados recebidos por unidade de tempo. Esta média, medida por intervalos, constitui a probabilidade de um bit chegar errado ao receptor.

Detecção de erros (*error detection*) Técnica que consiste em acrescentar um conjunto de bits redundantes a cada *frame* emitido, cujo valor é função da sequência de bits a transmitir. O valor dos bits redundantes é recalculado e verificado na recepção. Em caso de falha do teste, considera-se que o *frame* recebido contém erros.

Correcção de erros (*error correction*) Semelhante à técnica de controlo de erros, mas em caso de falha do teste, o valor dos bits redundantes recebidos permite saber quais os bits errados. Os códigos de correcção de erros exigem maior redundância, pelo que só são utilizados quando a taxa de erros é significativa.

Código de controlo de erros (*Cyclic Redundancy Check (CRC)*) Código de controlo de erros cujo cálculo é baseado em divisões módulo 2 do *frame* a transmitir deslocado para a esquerda de n bits, por um valor com $n + 1$ bits, chamado divisor ou gerador do código. A execução destes cálculos com elevada velocidade utiliza geralmente suporte de hardware específico.

Encapsulamento de pacotes em frames (*packet encapsulation in frames*) Na parte de dados dos *frames* viajam as mensagens dos protocolos do nível superior, geralmente os pacotes do protocolo IP, ou de outros protocolos do nível rede. Diz-se então que os pacotes do nível superior estão encapsulados no campo de dados dos *frames* que passam nos canais.

Referências

Este capítulo apresenta o conceito de canal e alguns aspectos essenciais de como os canais são implementados. A ênfase está na caracterização das funcionalidades e nas propriedades com maior impacto nos restantes níveis da rede.

O leitor interessado em aspectos mais concretos sobre a forma como os canais funcionam e são implementados deverá recorrer a outras referências bibliográficas, como por exemplo os capítulos 3, 4 e 5 de [Stallings, 2013], ou o capítulo 2 de [Tanenbaum and Wetherall, 2011].

Apontadores para informação na Web

- <http://www.ccs-labs.org/teaching/rn/animations/propagation/index.htm> – Contém um programa que permite visualizar o funcionamento de um canal do ponto de vista dos tempos de transmissão e de propagação.
- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.

2.7 Questões para revisão e estudo

1. Defina o que é um canal de dados.
2. Como se distingue um canal ponto-a-ponto de um canal multi-ponto?

3. Como se distingue um canal *full-duplex* de um canal *half-duplex*?
4. Indique pelo menos três grandezas que permitem caracterizar globalmente um canal?
5. Defina o que é o tempo de transmissão de um canal.
6. Defina o que é o tempo de propagação de um canal e explique como o mesmo se calcula.
7. Calcule quanto tempo leva um *frame* de 1000 bytes a ser transferido entre dois nós directamente ligados por um canal ponto-a-ponto com a capacidade de 100 Kbps e com 1 Km de comprimento. Considere que a velocidade de propagação no canal é de 200.000 Km/s.
8. Calcule quanto tempo leva um *frame* de 1000 bytes a ser transferido entre dois nós directamente ligados por um canal ponto-a-ponto com a capacidade de 100 Kbps e com 5.000 Kms de comprimento. Considere que a velocidade de propagação do canal é de 200.000 Km/s.
9. Nas condições do exercício anterior, qual o volume do canal (*i.e.*, quantos bits pode transmitir o nó origem até que o primeiro bit transmitido chegue ao nó de destino)?
10. Calcule quanto tempo leva um *frame* de 1 KB a ser transferido entre dois nós directamente ligados por um canal ponto-a-ponto com a capacidade de 100 Kbps e com 5.000 Kms de comprimento. Considere que a velocidade de propagação do canal é de 200.000 Km/s.
11. Calcule quanto tempo leva um *frame* de 1000 bytes a ser transferido entre dois nós terrestres, directamente ligados por um canal satélite com a capacidade de 100 Kbps. O tempo de propagação da Terra até ao satélite é de 125 ms.
12. Nas condições do exercício anterior, qual o volume do canal (*i.e.*, quantos bits pode transmitir o nó origem até que o primeiro bit transmitido chegue ao nó de destino)?
13. Defina o que é a taxa de erros de um canal.
14. Em princípio, mesmo que um canal tenha uma taxa de erros pouco significativa, utilizam-se técnicas que evitam que o canal entregue *frames* com erros ao nó de destino. Em que consistem essas técnicas?
15. Uma das características importantes que caracteriza cada tipo de canal é designada por MTU (*Maximum Transfer Unit*). Defina em que consiste e indique várias factores que são usados na sua fixação.
16. Considere um canal de dados com a taxa de erros de 10^{-5} . Neste canal é realista usar um MTU de 2.000 bytes?
17. Pretende-se que a probabilidade de um canal ter de desprezar *frames* por os mesmos conterem erros seja inferior a 10^{-4} . Qual o MTU que deve ser adoptado sabendo que a taxa de erros característica do meio de propagação deste canal é 10^{-8} .
18. Pretende-se que a probabilidade de um canal ter de desprezar *frames* por os mesmos conterem erros seja inferior a 10^{-4} . O canal tem um MTU de 1500 bytes. Qual a taxa de erros máxima que o meio de comunicação e a tecnologia de transmissão a serem adoptados pode ter?
19. A norma IEEE 802.3 (norma Ethernet) especifica que cada *frame* deve ser separado do seguinte por pelo menos 12 bytes. Qual o menor desperdício (relação entre a dimensão dos dados transportados e a dimensão total do *frame*) do protocolo do canal Ethernet? Tenha em consideração o formato e as dimensões dos campos dos seus *frames*.

20. Porque razão no cabeçalho dos *frames* de um canal multi-ponto aparecem campos designados por endereço origem e destino?

Capítulo 3

Comutação de circuitos e de pacotes

“For every complex problem there is an answer that is clear, simple, and wrong.”

– Autor: *M.L. Mencken*

O problema fundamental de qualquer rede, seja ela viária, de água ou de informação, é permitir o transporte eficiente, desde a origem até ao destino, dos objectos que encaminha.

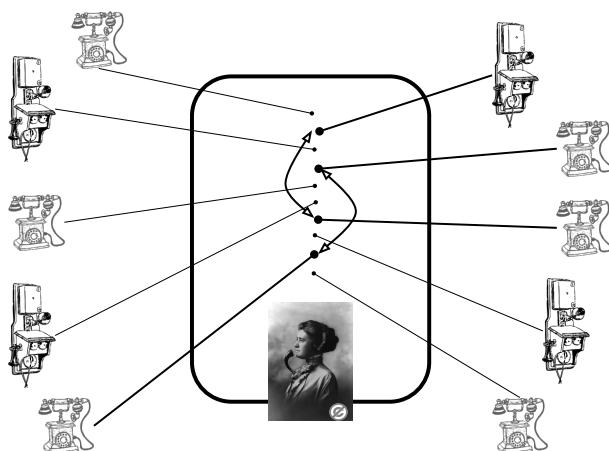


Figura 3.1: Estrutura de uma pequena rede telefónica primitiva

As redes antecessoras das redes de computadores foram as redes telefónicas. Nos seus primórdios o serviço telefónico não era universal, nem abrangia mais do que uma cidade. As primeiras empresas de telefones começaram por instalar fios de cobre entre as casas dos clientes e o seu escritório, e para se realizar uma chamada telefónica, era necessário solicitar à operadora que ligasse ao destinatário. A operadora, que até conhecia pelo nome cada um dos clientes, usava um sistema mecânico com fios eléctricos e fichas e estabelecia a ligação directa entre os fios de cobre que ligavam aos dois telefones. A Figura 3.1 apresenta um esquema lógico destas primeiras “centrais

telefónicas” manuais. Estas primeiras redes telefónicas, muito simples, tinham uma configuração em estrela com o sistema de fichas de ligação e a telefonista no centro.

Com o crescimento do número de aderentes nas grandes cidades começaram-se a estabelecer escritórios locais, regionais e nacionais, mas o princípio de base era o mesmo: para se realizar uma chamada telefónica, era necessário ligar em cadeia uma sequência de fios que iam desde o telefone que pediu a chamada até ao telefone de destino. Todo o processo podia envolver várias operadoras, amplificadores intermédios, levar bastante tempo, e até, para as chamadas de mais longa distância, implicar marcação prévia (e indirectamente reserva dos fios disponíveis entre centrais).

Dada a popularidade crescente do serviço, as cidades foram sendo cheias de fios de cobre que ligavam as centrais aos aderentes, as centrais foram-se ligando uma às outras por tantos fios quanto o número máximo de chamadas que as podiam atravessar simultaneamente, e foi estabelecida a prática de facturar cada chamada em função da duração e do número de centrais que atravessava. Esta indústria empregava então quantidades muito elevadas de mão de obra, quer para manter os fios, quer para o estabelecimento das ligações para as chamadas. A gravura reproduzida na Figura 3.2 data do início do século XX e mostra uma central telefónica da época.

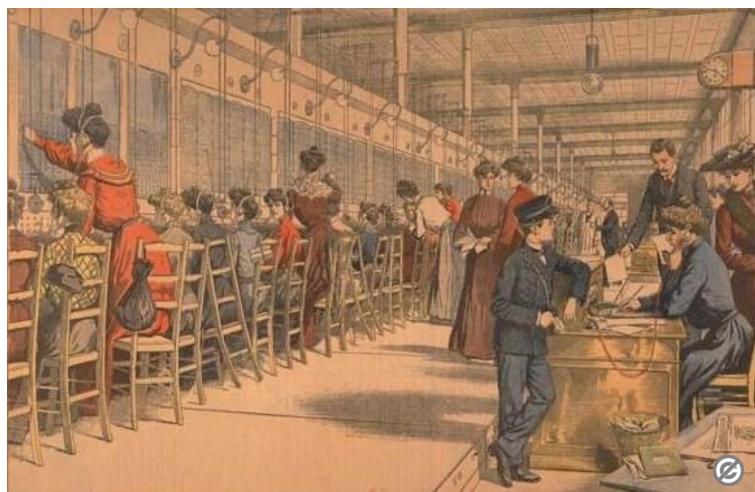


Figura 3.2: Uma central telefónica no início do séc. XX

Ao longo do século XX o processo foi-se automatizando. As centrais passaram a usar mecanismos electro-mecânicos para interligar automaticamente os fios, e foram desenvolvidas formas de usar um único suporte de transmissão para enviar simultaneamente várias chamadas, técnica designada por multiplexagem. Durante a segunda metade de século XX a voz passou a ser digitalizada pelas centrais telefónicas e mais tarde as próprias centrais passaram a ser digitais. No entanto, o princípio lógico de funcionamento da rede era do mesmo tipo: para se estabelecer uma chamada telefónica, era necessário “arranjar um fio” (virtual), chamado circuito telefónico, entre os dois interlocutores.

O sistema telefónico e a noção de circuito deu origem às chamadas redes de comutação de circuitos, um princípio estabelecido há bastante mais de um século e que teve uma grande influência no pensamento sobre redes de telecomunicações.

Na mesma altura em que o sistema telefónico iniciou o processo que levou à digitalização integral da rede telefónica, outro tipo de rede, baseada na comutação de

pacotes, começou a emergir. Na forma mais pura do conceito, a rede não tem nenhuma noção de conexão entre as partes em comunicação e por isso se diz rede de comutação de pacotes sem conexão. As redes IP e a Internet, ver o capítulo 4, são desta natureza.

Entre os dois extremos, rede de circuitos e rede de pacotes sem conexão, existem também soluções híbridas, mas os extremos mostram com clareza as vantagens e os defeitos das duas abordagens e o seu estudo e comparação são o objectivo deste capítulo. Nele ficará claro que uma rede de circuitos é mais inflexível e pode desaproveitar recursos, mas é a mais adequada perante uma situação de escassez dos mesmos pois, nessa situação, ainda consegue proporcionar algum trabalho útil, ainda que reduzido, enquanto que uma rede de pacotes colapsa. Num quadro de abundância de recursos, ou pelo menos com um dimensionamento adequado dos recursos às solicitações feitas à rede, a opção pela rede de pacotes é mais eficiente, flexível e escalável.

Como vimos no capítulo 2, os canais têm aumentado exponencialmente de capacidade e o seu preço também tem descido, o que, aliado com o aumento da sofisticação e capacidade dos equipamentos que usam ou controlam a rede explica a presente predominância das redes de pacotes.

No entanto, alguns elementos das redes de circuitos continuam presentes, apesar de sob outras formas, o que mostra que é importante conhecer bem os dois conceitos. Neste capítulo vamos analisar de forma informal e breve como funcionam as rede de circuitos e depois, mais em detalhe, como funcionam as redes de pacotes.

3.1 Comutação de circuitos

Numa rede baseada neste princípio, para que seja possível estabelecer um circuito entre dois interlocutores, é necessário dispor de uma forma de ter vários (de preferência muitos) canais paralelos que liguem os equipamentos de interligação de canais. Estes equipamentos, que inicialmente correspondiam às centrais telefónicas, chamam-se **comutadores de circuitos** (*circuit switches*).

Para construir muitos canais paralelos entre comutadores de pacotes são utilizadas técnicas de **multiplexagem**, que consistem, no essencial, em sub-dividir um canal em vários sub-canais autónomos, que possam ser dedicados independentemente ao transporte de sinais, dados, etc. Os dispositivos capazes de realizar esta multiplexagem chamam-se MUX e DEMUX, que são abreviaturas de *multiplexer / demultiplexer*, ver a Figura 3.3.

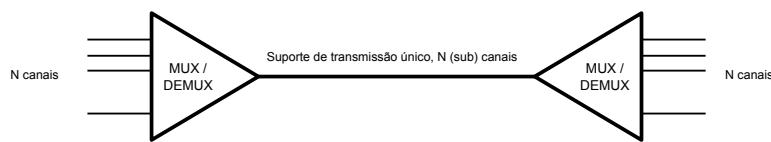


Figura 3.3: Multiplexagem de um suporte de transmissão

A multiplexagem de um canal consiste em dividi-lo em sub-canais que podem ser utilizados de forma independente. Esta operação é implementada por dispositivos, situados nas extremidades do meio de transporte do sinal, que se chamam MUX e DEMUX (por abreviatura de *multiplexer / demultiplexer*).

Multiplexagem por frequência e multiplexagem por tempo

Como foi apresentado na secção 2.3, é possível no mesmo suporte de transmissão usar diferentes frequências portadoras para enviar diferentes sinais em paralelo. Esta forma de multiplexagem diz-se **multiplexagem por frequência (frequency division multiplexing (FDM))** e permite transmitir pelo mesmo suporte vários canais distintos em paralelo.

A outra forma de multiplexagem chama-se **multiplexagem por tempo (time division multiplexing (TDM))** e baseia-se em afectar a cada sub-canal um intervalo de tempo distinto, chamado *time slot*. Do lado do emissor, os diferentes sub-canais vão sendo servidos e transmitidos pelo canal multiplexado, em sucessão, durante os respectivos *time-slots*. Após servir todos os sub-canais volta-se ao princípio e recomeça-se o processo. Com n *time-slots*, cada um com uma duração de t segundos, o processo periódico repete-se em cada $n \times t$ segundos.

O exemplo a seguir é fictício mas serve para ilustrar o princípio. Através de um canal a transmitir 1 Mbps é possível implementar 10 sub-canais a transmitirem a 100 Kbps cada um. Durante o primeiro *time-slot* de 100 ms transmitem-se 100.000 bits do sub-canal 1, a seguir transmitem-se 100.000 bits do sub-canal 2, etc. até se servir o sub-canal 10. Ao fim de 1 segundo o processo recomeça. Em cada segundo transmitem-se 100.000 bits de cada um dos 10 sub-canais o que corresponde um ritmo de transmissão de 100 Kbps.

O exemplo não passa de uma ilustração pois a multiplexagem por tempo usa geralmente *time-slots* de muito curta duração, durante os quais se transmite um número reduzido de bits de cada sub-canal, por exemplo 8. Por outro lado, com o objectivo de sincronizar as duas extremidades do canal multiplexado, um ou mais *time-slots* são reservados para transmitir informação de sincronização. A implementação da multiplexagem por tempo requer a existência de registos (*buffers*) nos MUXs pois, enquanto um sub-canal está a receber bits, estão a ser transmitidos os que recebeu no ciclo anterior. Estes registos são responsáveis pela introdução de um atraso (geralmente desprezável) que se acrescenta ao tempo de propagação dos canais. A Figura 3.4 mostra o funcionamento de um MUX a usar multiplexagem por tempo.

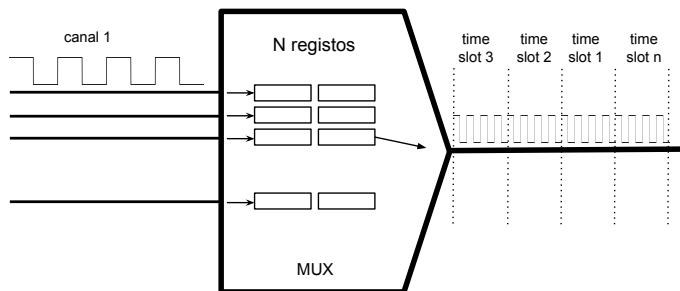


Figura 3.4: Multiplexagem por tempo

Dispondo de muitos sub-canais entre cada comutador de circuitos (e.g., central telefónica) é agora possível sugerir como é implementado o circuito (Figura 3.5). O dispositivo que pretende comunicar (e.g., fazer a chamada telefónica), solicita à rede que estabeleça o circuito. Para tal indica ao comutador a que está ligado o endereço do destinatário (e.g., o número de telefone), esse comutador determina o comutador seguinte no caminho para o destino e afecta um sub-canal ao circuito. O comutador seguinte repete o processo até se chegar ao comutador a que está ligado o destinatário. Nessa altura dispõe-se de um circuito, constituído pela “concatenação” dos sub-canais seleccionados, e que fica afectado aos dois interlocutores enquanto estiverem a

comunicar (e.g., enquanto a chamada durar).

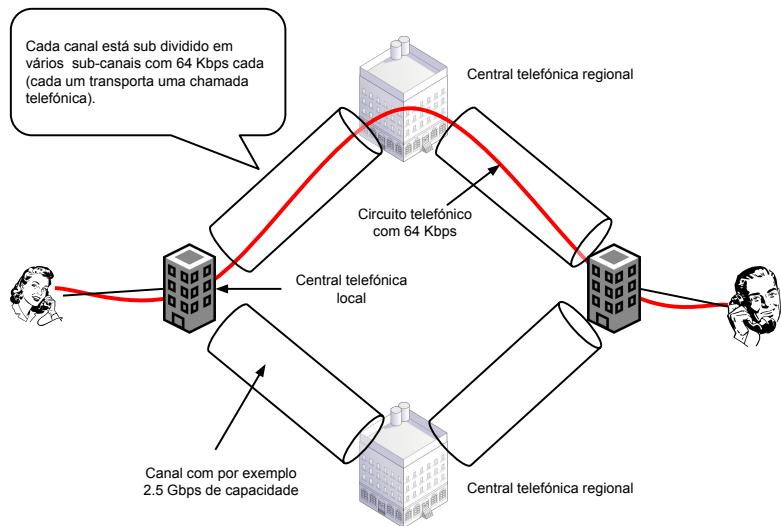


Figura 3.5: Um circuito telefónico típico suportava uma chamada telefónica transmitindo o som digitalizado a 64 Kbps

Se no momento do estabelecimento do circuito não houverem sub-canais disponíveis, o seu estabelecimento falha e não é possível comunicar. Trata-se de uma situação em que a rede está saturada e não aceita estabelecer mais circuitos.

O modo de funcionamento de uma rede de circuitos pode ser assim caracterizado.

Caracterização das redes de comutação de circuitos:

1. Antes de poder comunicar, é necessário solicitar à rede que estabeleça um circuito entre as extremidades.
2. Se o circuito for estabelecido, a rede reservou capacidade que lhe fica afectada enquanto este estiver activo.
3. Um circuito só pode usar a capacidade que lhe foi afectada.
4. Mesmo que os interlocutores não tenham nenhuma informação a transmitir, a reserva da capacidade na rede mantém-se e essa capacidade de comunicação não pode ser usada por outros circuitos.

As redes de circuitos foram inicialmente concebidas para suportarem chamadas telefónicas e por isso foram muito influenciadas pelas suas características: 1) geralmente um equipamento só faz uma chamada de cada vez e esta dura um tempo apreciável, pelo que se amortiza bem o tempo necessário para estabelecer o circuito; 2) o débito requerido é constante e conhecido *a priori* (nas redes telefónicas digitais tradicionais o som emitido pelos telefones analógicos era digitalizado pelas centrais, sem compressão, a 64 Kbps, ver o Capítulo 9); e 3) o circuito afectado à chamada tem de estar disponível em permanência pois não se sabe quando alguém vai falar a seguir. Apesar de o débito requerido para a transmissão de imagens digitalizadas ser variável, mais tarde o mesmo modelo foi também adoptado para o suporte de vídeo-chamadas.

As primeiras redes de computadores começaram por usar como canais de longa distância circuitos permanentes disponibilizados pelas redes dos operadores de telecomunicações. No entanto, quando se começaram a instalar redes de computadores dentro dos edifícios, começaram a ser usados canais de muito maior velocidade de transmissão como os canais Ethernet, ver a Secção 2.5.

Desde sempre foi claro para os investigadores que as aplicações das redes de computadores (*e.g.*, correio electrónico, transferência de ficheiros, acesso ao WWW, ...) têm padrões de comunicação muito diferentes dos característicos das chamadas telefónicas.

Com efeito, um computador pode comunicar directamente com vários outros computadores simultaneamente (frequentemente dezenas, ou mesmo milhares, no caso dos servidores). As aplicações de dados têm um ritmo de progresso muito irregular e podem ter momentos de inactividade prolongados. Mesmo quando requerem transferência de grandes quantidades de informação, muitas delas adaptam-se ao débito extremo-a-extremo disponível, levando mais ou menos tempo a terminar conforme o débito disponível no momento. Algumas aplicações (*e.g.*, DNS) não necessitam de comunicar por períodos prolongados pois apenas necessitam de trocar uma ou duas mensagens, e por isso o estabelecimento do circuito revela-se desproporcionalmente pesado.

Quando a rede se destina a todo o tipo de aplicações (dados, voz, vídeo) o modelo da rede de circuitos não é adequado visto que leva a desperdícios (pois requer reservas) e não é suficientemente flexível (pois o padrão de utilização previsto é único e adapta-se mal à diversidade de padrões do tráfego de dados).

Perante este tipo de constatações foi desenvolvido um modelo alternativo de rede. Mas antes de o apresentarmos vamos ver a técnica de multiplexagem usada no seu suporte, designada multiplexagem estatística.

Multiplexagem estatística

A **multiplexagem estatística** (*statistical multiplexing*) pode ser vista como uma generalização da multiplexagem por tempo: ao invés de os *time slots* serem fixos e iguais, os mesmos têm uma duração máxima, mas podem ser mais curtos se o sub-canal não tiver tráfego para enviar. Para a implementação são usados *frames* (ver a Secção 2.5) com um cabeçalho que indica a que sub-canal os dados que estes contém pertencem. Quando um canal transmite um destes *frames*, está integralmente afectado ao sub-canal respectivo. Ver a Figura 3.7.

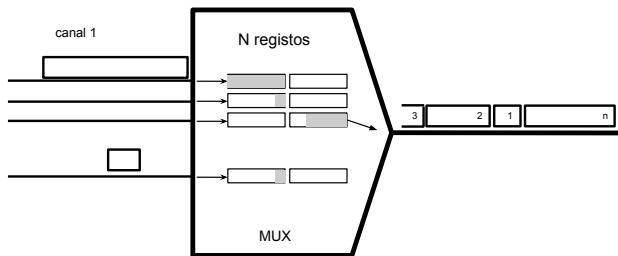


Figura 3.6: Multiplexagem estatística

O MUX dispõe de registos associados aos diferentes sub-canais e vai servindo-os em carrossel (*round robin*), enviando sucessivamente um *frame* por conta de cada sub-canal. Para permitir equidade e um grau adequado de multiplexagem, a dimensão

máxima de um *frame* é limitada (ver a secção 2.5), e se todos os sub-canais estiverem a usar a capacidade máxima que podem, são servidos equitativamente, pois o MUX transmite o mesmo número de bits (o *frame* máximo) por sub-canal. Comparativamente, apenas se introduz um pequeno desperdício devido à existência de cabeçalhos (com o identificador do sub-canal, dimensão do *frame*, etc.).

No entanto, caso um dos sub-canais não tenha informação para transmitir, o MUX passa ao canal seguinte e não perde tempo. O mesmo se passa se um canal tem menos tráfego, os *frames* correspondentes são mais curtos. No limite, se só um sub-canal estiver activo num dado momento, ele pode consumir toda a capacidade disponível no canal. Este modo de funcionamento diz-se estatístico porque, perante tráfego irregular, a afectação do canal a cada sub-canal tem um carácter aleatório, dependente dos diferentes padrões de tráfego.

A multiplexagem estatística contempla uma afectação flexível da capacidade do canal a cada sub-canal, não havendo desperdício da capacidade disponível. A capacidade não usada por um sub-canal será usada pelos outros. Caso todos os sub-canais requeiram o débito máximo, o canal é subdividido entre eles de forma equitativa e caso um só sub-canal esteja activo, ele usará a totalidade da capacidade disponível.

3.2 Comutação de pacotes

Uma rede baseada na **comutação de pacotes** (*packet switching*), funciona segundo o modelo já introduzido na secção 1. Os emissores emitem pacotes de dados, sendo um pacote IP um exemplo popular de pacote de dados. O pacote tem um cabeçalho com a indicação, entre outras informações, do endereço do emissor e do receptor. No caso dos pacotes IP, os endereços são globais à rede e não se referem a nenhuma noção de (sub-)canal. Noutras redes, esses endereços podem ter um significado local ao comutador de pacotes, mas esta diferença não é relevante a este nível.

Um **comutador de pacotes** (*packet switch*), o equipamento inteligente dentro da rede, recebe um pacote por um dos seus canais de entrada, memoriza-o e verifica se este tem erros. Se tiver, ignora-o, mas se não, analisa o seu cabeçalho e determina para que canal de saída o deve enviar. Depois coloca-o na fila do espera associada ao canal de saída, onde o pacote fica à espera de chegar a sua vez de ser transmitido. Existem comutadores de pacotes com políticas mais sofisticadas, mas por agora podemos considerar que essas filas de espera são únicas por canal de entrada e saída, e geridas segundo uma política do tipo “primeiro chegado, primeiro servido” (FCFS) ou “primeiro *in*, primeiro *out*” (FIFO). Assim, a interface de saída do canal vai transmitindo à velocidade máxima, uns a seguir aos outros, todos os pacotes que devem seguir para o seu destino pelo canal.

A este modo de funcionamento, caracterizado por cada comutador receber, memorizar, analizar e transmitir os pacotes, chama-se o modo de funcionamento ***Store & Forward***. Ao modo como os canais são partilhados pelos diferentes fluxos de comunicação existentes na rede chama-se, como já vimos, multiplexagem estatística.

Um comutador de pacotes é mais sofisticado e versátil que um simples MUX estatístico: no comutador de pacotes os *frames* têm um código de controlo de erros e associado a cada canal existem filas de espera de entrada e saída de pacotes e não simples registos. O comutador tem maiores possibilidades de adaptação do tráfego à capacidade disponível.

As noções de sub-canal e de circuito deixam de ser necessárias. Aparecem em sua substituição as noções de **pacote de dados** (*data packet*) e de **fluxo de pacotes**



Figura 3.7: Fotografia de vários tipos de comutadores de pacotes de diferentes dimensões e sua interligação através de canais sem fios e de fibras ópticas (figuras superiores). As figuras inferiores apresentam representações convencionadas de comutadores de pacotes através de diagramas.

(packet flow). Um fluxo de pacotes é um conjunto de pacotes correlacionados que têm em comum o emissor e o receptor.

Uma rede a funcionar segundo o princípio da comutação de pacotes é uma rede que encaminha pacotes de dados através de comutadores de pacotes que funcionam segundo o modelo *Store & Forward*, e que gerem a afectação dos canais aos diferentes fluxos de pacotes usando multiplexagem estatística e filas de espera.

Esta forma de funcionamento de um comutador de pacotes está ilustrado na Figura 3.8 e conduz a redes que podem ser assim caracterizadas.

Caracterização da comutação de pacotes:

1. O consumo da capacidade de um canal pelos diferentes fluxos de dados, é de carácter estocástico, e depende do conjunto do tráfego que atravessa o canal.
2. Se um emissor não emitir momentaneamente pacotes, não consome recursos da rede durante esse período.
3. A capacidade dos diferentes canais que um fluxo de pacotes pode usar está dependente da capacidade consumida pelos outros fluxos.

Um aspecto que o leitor deve ter em atenção para melhor perceber a figura é que as interfaces de um comutador trabalham em paralelo umas com as outras. Portanto, um comutador pode estar a receber pacotes em paralelo por diferentes interfaces e pode, simultaneamente, estar a enviar pacotes em paralelo por várias outras interfaces.

O custo do *Store & Forward* e da multiplexagem estatística

O funcionamento *Store & Forward* tem como repercussão a introdução de um compasso de espera em cada equipamento, que é no mínimo equivalente ao tempo de transmissão dos pacotes.

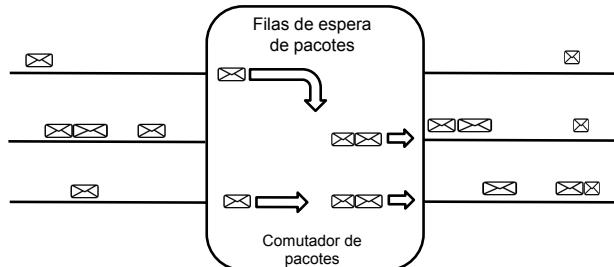


Figura 3.8: Comutação de pacotes com multiplexagem estatística

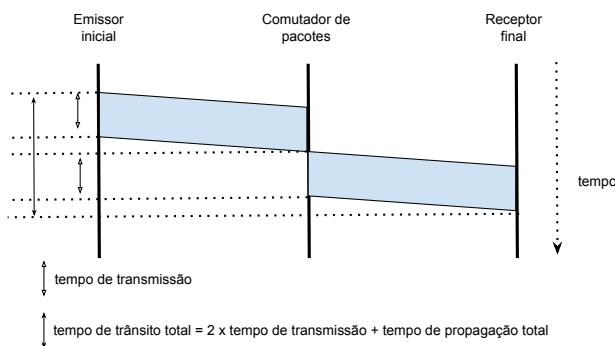


Figura 3.9: Atraso introduzido devido ao tempo de transmissão

Com efeito, o tempo de trânsito de um pacote por um canal é a soma do tempo de transmissão com o tempo de propagação, ver a Secção 2.2. Se um canal for separado em dois canais e for introduzido um comutador de pacotes no meio, o tempo de propagação é o mesmo, pois, por hipótese, os dois canais têm a dimensão do antigo canal, mas o comutador introduz um tempo de transmissão extra.

Assim, desprezando o tempo que um comutador leva a processar um pacote (análise do cabeçalho e cópia, em memória, entre filas de espera) e admitindo que não existe outro tráfego, *i.e.*, isolando o custo do funcionamento *Store & Forward*, verifica-se que cada comutador de pacotes a funcionar de acordo com o mecanismo *Store & Forward* introduz no mínimo um tempo de transmissão extra no tempo de trânsito de cada pacote, ver a Figura 3.9.

No entanto, o atraso pode ser maior pois, quando um pacote chega à fila de espera de saída do canal por que vai ser transmitido, pode encontrar outros pacotes à sua frente que têm de ser transmitidos antes dele, ver a Figura 3.10.

A formação de filas de espera tem lugar quando um canal recebe, num dado momento, mais pacotes do que aqueles que consegue transmitir. Este tipo de situação verifica-se frequentemente quando um comutador concentra vários canais de entrada / saída ou os canais de saída têm menos capacidade que os de entrada.

O seguinte exemplo ilustra esta questão e põe em evidência a evolução no tempo da dimensão da fila de espera. Dois computadores A e B estão ligados ao resto da rede por um comutador de pacotes (Figura 3.11). Os computadores estão ligados por canais a 10 Mbps (A) e a 100 Mbps (B) ao comutador de pacotes. O comutador está ligado ao exterior por um canal a 10 Mbps.

Se só o computador A enviar pacotes para o exterior, continuamente e sem in-

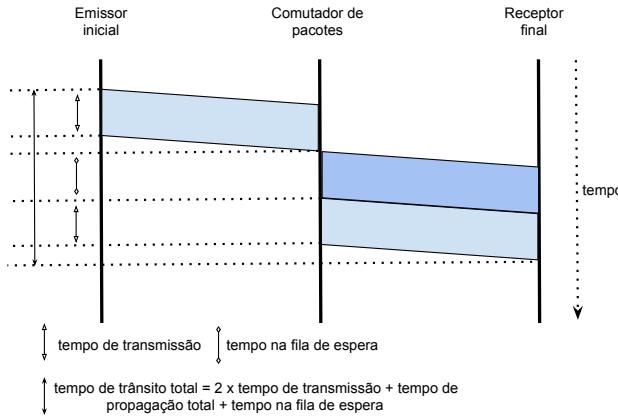


Figura 3.10: Atraso introduzido devido ao tempo de transmissão dos pacotes que já estavam na fila de espera (a mais escuro) e atraso introduzido devido ao tempo de transmissão do pacote

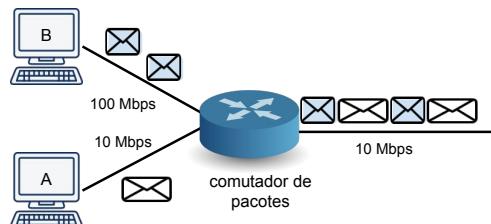


Figura 3.11: Os computadores A e B estão ligados via o comutador de pacotes em cuja interface de saída se forma uma fila de espera

terrupção, a fila de espera no comutador só tem o pacote a ser transmitido. De facto, visto que os dois canais (o que liga o computador A ao comutador e o que liga o comutador ao exterior) têm a mesma velocidade de transmissão (10 Mbps), no momento em que um novo pacote acaba de ser recebido pelo comutador, o último acabou de ser transmitido para o exterior, ver a Figura 3.12 (a).

A dimensão da fila de espera de saída do comutador está representada na Figura 3.12 (b), e apenas contém o pacote a ser emitido em cada momento. Neste caso apenas é introduzido um tempo de transmissão extra. Admitindo que os pacotes têm um total de 10.000 bits, esse tempo de transmissão seria de $10^4 \times 10^{-7} = 10^{-3}$ s = 1 ms.

No entanto, se ambos os computadores emitirem pacotes de vez em quando, de forma descoordenada, os pacotes de A podem entrar às vezes em competição com os de B, ver a Figura 3.12 (c), a fila de espera de saída cresce, como ilustrado na Figura 3.12 (d)), e o atraso introduzido pelo comutador vai crescer e variar. Por exemplo, admitindo que num dado momento um pacote emitido pelo computador A chega ao comutador, e na fila de espera de saída já estão 3 pacotes com a mesma dimensão à sua frente, o tempo tomado pelo pacote para atravessar o comutador cresce para 4 ms.

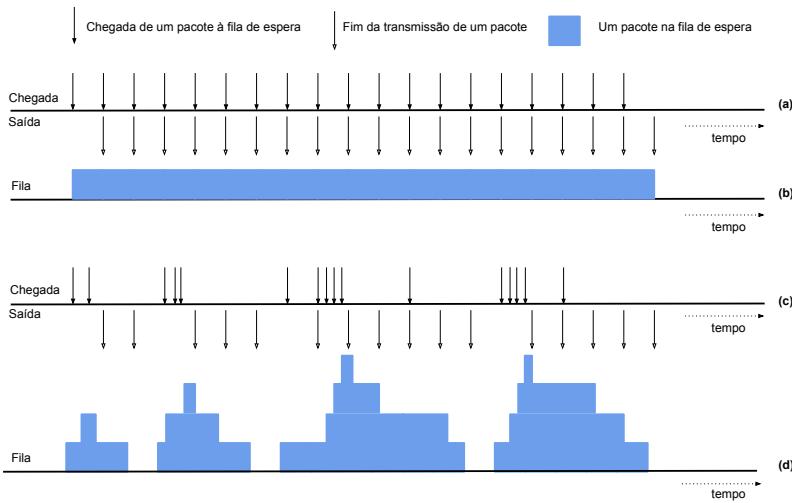


Figura 3.12: Evolução do estado da fila de espera de saída do comutador (Figura 3.11) nas situações: (a) e (b) apenas o computador A emite continuamente pacotes; em (c) e (d) pacotes dos computadores A e B chegam de forma irregular e descoordenada à fila de saída do comutador

A flexibilidade introduzida pelo funcionamento *Store & Forward* e pela multiplexagem estatística de pacotes tem como contrapartida um aumento do tempo de trânsito dos mesmos: tempo de transmissão extra dos pacotes que os antecedem nas diferentes filas de espera, e tempo de transmissão extra do pacote por cada comutador cruzado.

3.3 Tempo de trânsito extremo a extremo

Uma rede é formada por vários canais e vários comutadores de pacotes e o tempo total que leva um pacote a chegar ao destinatário designa-se por **tempo de trânsito extremo a extremo (end-to-end delay)**.

Por hipótese, admitamos que se conhecem os comutadores e os canais atravessados, e que estes são identificados pelo índice $i = 0..k$, incluindo o emissor inicial ($i = 0$) e todos os comutadores atravessados (de $i = 1$ até $i = k$).

Também por hipótese, consideremos que é possível desprezar o tempo de processamento do pacote, quer pelo emissor inicial, quer pelos comutadores intermédios. Finalmente, seja $T_t(i)$ o tempo de transmissão, $T_p(i)$ o tempo de propagação e $T_{fe}(i)$ o tempo que o pacote tem de esperar na fila de espera. O tempo de trânsito extremo a extremo (T) é dado por:

$$T = \sum_{i=0}^k (T_{fe}(i) + T_t(i) + T_p(i)) \quad (3.1)$$

Admitindo que existia um único canal (ou um circuito) entre a origem e o destino, o tempo de trânsito seria

$$T = T_{fe}(0) + T_t(0) + \sum_{i=0}^k T_p(i)$$

i.e., o tempo gasto no emissor inicial mais o tempo de propagação. Logo, a contribuição da comutação de pacotes para o tempo de trânsito é

$$T = \sum_{i=1}^k (T_{fe}(i) + T_t(i)) \quad (3.2)$$

i.e., como seria de esperar, o tempo de permanência nas filas de espera e o tempo de transmissão de cada comutador de pacotes atravessado. A este tempo teríamos de acrescentar o tempo de processamento em cada equipamento mas, por hipótese, desprezámos este factor.

Usando sempre o mesmo caminho e continuando a admitir que o tempo de processamento é desprezável, todos os termos das equações 3.1 e 3.2 são constantes excepto os termos $T_{fe}(i)$, porque o tempo de espera nas filas de espera depende dos pacotes emitidos no conjunto da rede. No caso geral da Internet, ou de qualquer rede com muitos utilizadores e muitas aplicações de dados, os ritmos de emissão de pacotes pelos diferentes emissores são muito variáveis e só se podem caracterizar através de variáveis aleatórias.

Se um comutador receber pacotes a um ritmo tal que excede a sua capacidade de transmissão e, consequentemente, que alguma das suas filas de espera comece a ser significativo, o tempo de trânsito também cresce. Se a totalidade dos pacotes injectados na rede for tal que os seus comutadores fiquem saturados, as suas filas de espera tendem para ter uma “infinitade” de pacotes à espera de serem transmitidos, e o tempo de trânsito tenderia para infinito!

Essa rede poderia produzir algum trabalho útil, pois inicialmente chegavam ao destino pacotes com pouco atraso. Mas rapidamente o atraso com que os pacotes chegavam ao destino aumentaria muito (no limite até ao “infinito”) e os pacotes deixariam de ser úteis pois estariam a chegar “tarde demais”. Assim, os comutadores de pacotes limitam a dimensão máxima das suas filas de espera e quando um pacote chega a um comutador, é suprimido caso não haja espaço nas ditas, como se fosse recebido em erro.

O aumento do ritmo de emissão de pacotes para além do que a rede pode suportar, leva ao aumento do tempo de trânsito e, no limite, à perda de pacotes no interior da rede.

Com filas de espera limitadas, os termos $T_{fe}(i)$ não deixam de ser variáveis, mas pode-se conhecer o intervalo de variação: de 0 (no caso em que um pacote encontra a fila vazia) até ao máximo que corresponde ao tempo de transmitir a totalidade da fila de espera (quando o pacote chegou, ele era o último que ainda coube na fila de espera). Nesse caso, o tempo de trânsito de um pacote que chega ao destino varia entre

$$T_1 = \sum_{i=0}^k T_t(i) + T_p(i) \quad e \quad T_2 = T_1 + T_{fe-max}(i) \quad (3.3)$$

Na prática, o tempo de trânsito varia entre um mínimo e valores dependentes da actividade existente no conjunto da rede e é proporcional ao dimensionamento da mesma. Quanto melhor for o dimensionamento da mesma, *i.e.*, mais longe da saturação e do crescimento das filas de espera dos comutadores ela estiver, menor será a variação.

Quanto pior for esse dimensionamento, ou seja quanto mais próximos de encherem as filas de espera estiverem os comutadores, maior será a variação.

Em redes de computadores é comum adoptar-se o termo em língua inglesa *jitter*, como sinónimo da variação do tempo de trânsito dos pacotes que transitam pela rede, de um emissor para um receptor. A tradução mais directa para *jitter* neste contexto é instabilidade, e portanto o seu uso no campo das redes é sinónimo de instabilidade do tempo de trânsito.

Na verdade o que acabámos de descrever é fácil de perceber. Basta transpô-la para a experiência quotidiana que consiste em comparar o tempo que leva a atravessar de automóvel uma zona congestionada da cidade à hora de ponta, com o tempo que leva a mesma travessia de madrugada.

Repare-se que, em contraste, numa rede de circuitos não só não há atrasos variáveis, como também não há saturação ou perda de pacotes, pois se não há sub-canais disponíveis para activar o circuito, a comunicação não é possível. É como se numa cidade, uma carreira de autocarros só pudesse iniciar um percurso depois de reservar um conjunto contíguo de faixas de rodagem entre o início e o fim do percurso. Seria uma rede viária muito cara e inflexível, mas em que as poucas carreiras possíveis fariam o percurso à velocidade máxima.

Assim, conhecendo o caminho, as características dos canais e a dimensão máxima das filas de espera dos comutadores de pacotes atravessados, é possível calcular o tempo de trânsito mínimo e máximo de um pacote. No entanto, na grande maioria dos casos esses dados não são conhecidos pelo que a alternativa possível é tentar medir o tempo de trânsito, usando para esse efeito o programa `ping`, disponível em todos os sistemas de operação.

Medir o tempo de trânsito extremo a extremo

O programa `ping` envia sucessivos pacotes (pacotes sonda) entre o computador que o executa e o computador alvo. Os pacotes são enviados utilizando um protocolo (o protocolo ICMP, ver o RFC 792 e o Capítulo 17) que leva o computador alvo a responder ao emissor original com um pacote do mesmo tamanho. Assim é possível medir o tempo de trânsito de ida e volta ou *Round Trip Time* (RTT).

Admitindo que a) o caminho de ida e volta atravessa exactamente os mesmos comutadores, b) todos os canais são ponto-a-ponto, *full-duplex* e simétricos, e c) a situação das filas de espera é equivalente, o RTT é igual ao dobro do tempo de trânsito. O programa envia vários pacotes sonda e, quando termina, apresenta os valores mínimo, médio, máximo e o desvio padrão das medidas feitas.

Anteriormente, quando estabelecemos as fórmulas de cálculo do tempo de trânsito extremo a extremo, usámos certas hipóteses e considerámos que tínhamos a informação completa sobre o caminho, os comutadores e os canais. Na prática nem todas as hipóteses se verificam (*e.g.*, poder desprezar o tempo de processamento), nem conhecemos todos os dados sobre o percurso dos pacotes. Assim, o programa `ping` fornece-nos às vezes resultados que revelam algumas surpresas, mas esses resultados não deixam de estar em linha com o modelo usado para deduzir teoricamente o tempo de trânsito.

A seguir apresentam-se algumas medidas obtidas com o programa `ping` para medir o RTT entre um computador situado na região de Lisboa e computadores situados nas regiões de Paris (França), de Los Angeles (EUA) e de Auckland (Nova Zelândia, nos antípodas de Portugal). Em todos os casos fizeram-se 6 medidas (parâmetro `c = 6`).

Quer o computador origem, quer os computadores alvo, estão ligados à Internet através de canais guiados, *i.e.*, baseados em fios. Portanto, os testes não sofrem do impacto de fenómenos relacionados com características particulares dos canais sem fios

(taxas de erros muito elevadas e contenção no acesso devido à partilha entre vários emissores, como foi apresentado na secção 2.3).

```
$ ping -c 6 www.inria.fr
PING ezp3.inria.fr (128.93.162.84): 56 data bytes
64 bytes from 128.93.162.84: icmp_seq=0 ttl=51 time=39.508 ms
64 bytes from 128.93.162.84: icmp_seq=1 ttl=51 time=38.203 ms
64 bytes from 128.93.162.84: icmp_seq=2 ttl=51 time=38.367 ms
64 bytes from 128.93.162.84: icmp_seq=3 ttl=51 time=37.888 ms
64 bytes from 128.93.162.84: icmp_seq=4 ttl=51 time=38.262 ms
64 bytes from 128.93.162.84: icmp_seq=5 ttl=51 time=37.895 ms

--- ezp3.inria.fr ping statistics ---
6 packets transmitted, 6 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 37.888/38.354/39.508/0.546 ms

$ ping -c 6 ucla.edu
PING ucla.edu (128.97.27.37): 56 data bytes
64 bytes from 128.97.27.37: icmp_seq=0 ttl=50 time=177.351 ms
64 bytes from 128.97.27.37: icmp_seq=1 ttl=50 time=176.195 ms
64 bytes from 128.97.27.37: icmp_seq=2 ttl=50 time=175.938 ms
64 bytes from 128.97.27.37: icmp_seq=3 ttl=50 time=176.594 ms
64 bytes from 128.97.27.37: icmp_seq=4 ttl=50 time=175.894 ms
64 bytes from 128.97.27.37: icmp_seq=5 ttl=50 time=175.830 ms

--- ucla.edu ping statistics ---
6 packets transmitted, 6 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 175.830/176.300/177.351/0.535 ms

$ ping -c 6 www.auckland.ac.nz
PING www.auckland.ac.nz (130.216.159.127): 56 data bytes
64 bytes from 130.216.159.127: icmp_seq=0 ttl=231 time=294.663 ms
64 bytes from 130.216.159.127: icmp_seq=1 ttl=231 time=293.014 ms
64 bytes from 130.216.159.127: icmp_seq=2 ttl=231 time=293.077 ms
64 bytes from 130.216.159.127: icmp_seq=3 ttl=231 time=292.939 ms
64 bytes from 130.216.159.127: icmp_seq=4 ttl=231 time=293.009 ms
64 bytes from 130.216.159.127: icmp_seq=5 ttl=231 time=292.044 ms

--- www.auckland.ac.nz ping statistics ---
6 packets transmitted, 6 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 292.044/293.124/294.663/0.774 ms
```

Os exemplos mostram que não se perdem pacotes e que o tempo de trânsito não varia muito, o que permite tirar duas conclusões. Primeiro, durante os testes, as zonas da Internet atravessadas pelos pacotes estavam bem dimensionadas e não se perderam pacotes. Segundo, a variação do tempo devido a filas de espera foi baixo. Com efeito, no momento em que os testes foram realizados nas redes atravessadas pelos pacotes a grande maioria dos canais internacionais funcionam a 10 Gbps. Como os pacotes de sonda eram pequenos (56 bytes sem considerar cabeçalhos), e portanto com um comprimento total inferior a 1.000 bits, o seu tempo de transmissão a 10 Gbps é de aproximadamente $10^3/10^{10} = 10^{-7} = 0.1 \mu\text{s}$ (microsegundo). Ou seja, em canais de elevada capacidade, o impacto das filas de espera é relativamente pequeno no tempo de trânsito quando o tempo de propagação é elevado.

As distâncias em quilómetros entre o centro das cidades, medidas em linha recta sobre a superfície da terra, e os tempos de propagação numa fibra estendida em linha recta entre as mesmas cidades, são os indicados na tabela 3.1.

Como o dobro do tempo de propagação na fibra (o RTT representa a ida e a volta) é sempre inferior ao RTT medido, isso mostra dois factos. Primeiro, o caminho seguido afasta-se bastante do que seria um canal directo entre as duas cidades, que teria de atravessar continentes e oceanos em linha recta. Segundo, sobretudo no caso das cidades mais próximas (Lisboa e Paris), o tempo de processamento não deve ser desprezado. Este é geralmente bastante relevante nos computadores nos dois extremos dado que, quando um pacote chega, este pode não ser imediatamente processado, sobretudo não se tratando de um protocolo crítico e os computadores alvo serem

Tabela 3.1: Tabela de distâncias, tempos de propagação numa fibra directa entre as duas cidades, RTT e TTL medidos

Origem / destino	Distância em linha recta	Tempo de propagação	RTT médio medido	TTL
Lisboa / Paris	1454 km	7 ms	39.5 ms	51
Lisboa / Los Angeles	9145 km	43 ms	176.3 ms	50
Lisboa / Auckland	19 680 km	92 ms	294.7 ms	231

servidores envolvidos em muitas outras actividades. Para além disso, neste tipo de testes, envolvendo servidores públicos como alvo, estão presentes cada vez mais outros dispositivos de rede, como por exemplo equipamentos de segurança, que introduzem atrasos suplementares. Por outro lado, como será posto em evidência a seguir, as redes de acesso dos operadores envolvem um conjunto de equipamentos e canais que introduzem igualmente atrasos significativos devido a tempos de processamento extra.

Quantos e quais comutadores foram atravessados?

O programa `ping` fornece-nos também, indirectamente, informação sobre o número de comutadores atravessados pelos pacotes. Em cada linha com resultados aparecem parâmetros suplementares (*e.g.*, `icmp_seq=1 ttl=231`) entre os quais um designado por TTL, que é a abreviatura de *Time To Live* e que também figura na tabela 3.1.

No cabeçalho dos pacotes IP existe um campo de 8 bits, designado TTL (ver a Figura 1.3 no capítulo 1), cujo papel é evitar que um pacote que entre em ciclo entre comutadores fique eternamente a ser transmitido, consumindo recursos indefinidamente. Sempre que um comutador recebe um pacote IP, para além de controlar os erros do pacote usando os campos de controlo de erros, também decrementa o valor encontrado no campo TTL, e se o resultado for 0, o pacote é considerado em erro e é suprimido. Assim, quando o emissor inicial envia o pacote inicializa o campo TTL com um valor que ache adequado para que este chegue ao destino, mas que evite que o pacote possa entrar num ciclo sem fim.

Como o computador que envia inicialmente o pacote não conhece o comprimento do caminho que este vai percorrer, o valor inicial do TTL é arbitrado, e toma o valor inicial 255 ou 64 nos sistemas de operação actuais. Assim, é possível estimar que o pacote de resposta vindo de Paris atravessou provavelmente 13 comutadores ($64 - 51 = 13$), que o pacote de Los Angeles até Lisboa atravessou provavelmente 14 comutadores ($64 - 50 = 14$) e que o pacote vindo de Auckland até Lisboa atravessou provavelmente 24 comutadores ($255 - 231 = 24$).

Com a técnica explicada é possível ter uma ideia do número de comutadores que um pacote IP atravessa. No entanto, se quisermos saber quais é necessário usar outro programa. O protocolo IP, ver o RFC 791, que regula o percurso dos pacotes IP pela rede, estipula que um comutador de pacotes IP deve, quando suprime um pacote por o seu TTL atingir o valor 0, notificar (opcionalmente) o emissor original de que o pacote foi suprimido, usando para tal o protocolo ICMP, ver o RFC 1122 e o Capítulo 17. Utilizando este mecanismo, é possível tentar saber, como se explica a seguir, que comutadores um pacote atravessa na sua viagem pela Internet.

Se um emissor E emitir um pacote sonda para o destino D, com TTL inicial igual a 1, o primeiro comutador de pacotes que o recebe, decrementa o TTL do pacote (de 1 para 0), suprime-o, e envia uma notificação a E. E pode então anotar o endereço do comutador que lhe enviou a notificação. Se E emitir um novo pacote dirigido a D com o TTL igual a 2, quando o pacote chegar ao segundo comutador, o seu TTL tem o valor 1, o comutador suprime-o e envia uma notificação a E. O processo continua até

um pacote que chegue ao destino D, e permite ao emissor E ficar a conhecer o caminho seguido pelos seus pacotes sonda até ao destino como está ilustrado na Figura 3.13.

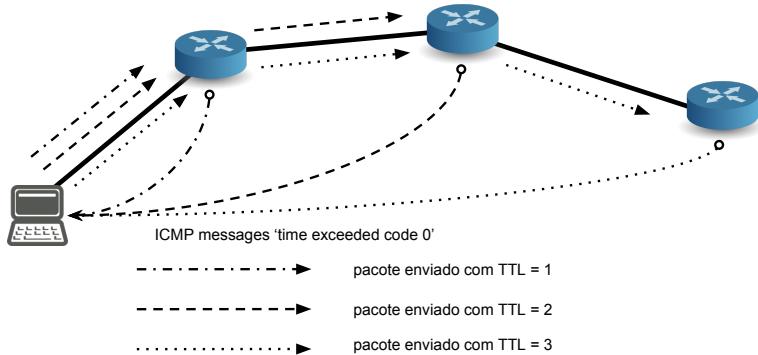


Figura 3.13: Execução do programa **traceroute**

O programa **traceroute** (**tracert** no sistema Windows) realiza a sequência de operações descritas e mostra o caminho seguido pelos pacotes até ao destino. Abaixo mostramos dois exemplos da sua utilização para descobrir o caminho entre um computador em Lisboa e dois outros computadores, um em Paris e outro em Los Angeles.

```
$ traceroute www.inria.fr
traceroute to ezp3.inria.fr (128.93.162.84), 64 hops max, 52 byte packets
 1  10.0.1.1 (10.0.1.1)  1.050 ms  0.774 ms  0.723 ms
 2  dsldevice.lan (192.168.1.254)  2.246 ms  2.056 ms  3.533 ms
 3  2.128.189.46.rev.vodafone.pt (46.189.128.2)  10.643 ms  8.584 ms  6.554 ms
 4  21.93.30.213.rev.vodafone.pt (213.30.93.21)  9.572 ms  9.281 ms  9.552 ms
 5  ae3-100-ucr1.lis.cw.net (195.10.57.1)  7.137 ms  7.392 ms  9.698 ms
 6  ae5-xcr1.mal.cw.net (195.2.30.230)  40.438 ms  40.977 ms  42.671 ms
 7  ae9-xcr1.plt.cw.net (195.2.30.181)  37.798 ms  39.233 ms  38.668 ms
 8  ae5-xcr1.prp.cw.net (195.2.10.89)  40.193 ms  40.881 ms  37.214 ms
 9  giprenater-gw.par.cw.net (195.10.54.66)  39.583 ms  44.874 ms  39.926 ms
10  te2-1-paris1-rtr-021.noc.renater.fr (193.51.177.27)  40.805 ms  43.202 ms ...
11  * * *
12  * * *
13  * * *
14  ezp3.inria.fr (128.93.162.84)  41.312 ms  38.729 ms  39.435 ms
```

No primeiro caso o programa mostra o resultado da descoberta do caminho entre o computador em Lisboa e um em Paris. O programa **traceroute** mostra a sequência de comutadores descobertos. Para cada um são apresentados o endereço IP e o tempo medido até receber a notificação de que o pacote de sonda foi suprimido por o seu TTL ter sido ultrapassado. Como o programa realiza 3 testes com o mesmo valor de TTL, são apresentadas 3 medidas. Os casos em que o resultado da medida foi substituído por “*”, assinalam que não foi recebida uma notificação, provavelmente por o comutador não a ter enviado visto que, segundo o protocolo ICMP, não é obrigado a responder ao teste. Isso é cada vez mais frequente sobretudo quando se tratam de equipamentos envolvidos na segurança da instituição.

O teste põe em evidência que o caminho está longe de ser directo, atravessa vários operadores e que o tempo necessário para os pacotes saírem da rede de acesso na região de Lisboa (que se dá entre os comutadores 4 e 5) é já muito elevado, o que mostra que o tempo de processamento pode ser significativo. Verifica-se também que os RTTs para cada um dos comutadores intermédios e o destino são variáveis.

```
$ traceroute ucla.edu
traceroute to ucla.edu (128.97.27.37), 64 hops max, 52 byte packets
 1 10.0.1.1 (10.0.1.1) 2.046 ms 1.879 ms 1.517 ms
 2 ds1device.lan (192.168.1.254) 5.250 ms 3.066 ms 1.972 ms
 3 2.96.54.77.rev.vodafone.pt (77.54.96.2) 9.896 ms 9.579 ms 9.998 ms
 4 21.93.30.213.rev.vodafone.pt (213.30.93.21) 9.525 ms 9.862 ms 9.677 ms
 5 ae5-100-ucr1.lis.cw.net (195.10.57.9) 15.140 ms 12.979 ms 12.717 ms
 6 ae5-xcr1.mal.cw.net (195.2.30.230) 23.257 ms 26.014 ms 23.604 ms
 7 et-1-3-0-xcr2.prp.cw.net (195.2.24.189) 36.238 ms 38.611 ms 36.070 ms
 8 lag-26.ear1.paris1.level3.net (212.73.242.237) 39.280 ms 39.765 ms 37.218 ms
 9 * * *
10 * * *
11 cenic.ear1.losangeles1.level3.net (4.35.156.66) 179.252 ms 179.903 ms 179.225 ms
12 * * *
13 bd11f1.anderson--cr01f1.anderson.ucla.net (169.232.4.6) 181.289 ms
  bd11f1.anderson--cr01f2.cs1.ucla.net (169.232.4.4) 180.015 ms
  bd11f1.anderson--cr01f1.anderson.ucla.net (169.232.4.6) 180.748 ms
14 cr01f2.cs1--sr02f2.cs1.ucla.net (169.232.8.7) 179.717 ms
  cr01f1.anderson--sr02fb.jsei.ucla.net (169.232.8.53) 178.705 ms 180.013 ms
15 128.97.27.37 (128.97.27.37) 179.559 ms !Z 177.996 ms !Z 178.304 ms !
```

No segundo caso é particularmente interessante observar que os testes na linha 8 mostram um comutador em Paris, com um tempo de recepção da notificação de 39 ms, enquanto que o comutador da linha 11 está em Los Angeles, e o tempo para obter a notificação passou para cerca de 180 ms.

No caso 13 verifica-se também que há mais do que um comutador a responder com o mesmo TTL. Isso mostra que há canais alternativos que estão a ser usados para distribuir a carga.

O conjunto de testes apresentados, realizados com os programas `ping` e `traceroute`, mostraram situações em que a variação do tempo de trânsito não era muito visível por se estarem a usar redes bem dimensionadas e com velocidades muito elevadas. Diz-se então que estamos perante redes com baixo *jitter*. No entanto, as redes de pacotes com menores capacidades podem introduzir variações significativas do tempo de trânsito. Dependendo das aplicações, estas variações podem ser, ou não, melhor toleradas.

As aplicações de transferência de dados como ficheiros, correio electrónico, acesso a Web, etc. adaptam-se com facilidade a tempos de trânsito variáveis, como até a taxas de transmissão de extremo a extremo também variáveis. No entanto, certas aplicações, como por exemplo as aplicações multimédia, ou os jogos, podem não conseguir lidar bem com essas variações. As aplicações de acesso a vídeo, telefonemas, vídeo-conferências, etc. quando executadas sobre uma rede de pacotes com *jitter* significativo, perdem qualidade e tornam-se por vezes inviáveis de todo, deixando de poder ser utilizadas. Também os jogos interactivos podem sofrer uma grande perda de qualidade se existir *jitter*.

A seguir é introduzida uma técnica que pode ser usada para compensar as variações do tempo de trânsito introduzidas pelas redes de pacotes.

3.4 Como viver com o jitter

As aplicações que não toleram *jitter* elevado usam uma técnica de compensação que é comum a todos as aplicações multimédia (de transmissão de som e vídeo) que se baseiam nos seguintes mecanismos:

- associar marcas temporais aos blocos de dados emitidos;
- usar uma fila de espera de blocos de dados no receptor, designada *playback buffer*, para introduzir compassos de espera variáveis;
- processar os blocos de dados recebidos apenas quando chega o momento certo.

A técnica aplica-se quer a blocos de dados genéricos, quer a pacotes, mas para a sua explicação a seguir pressupomos que são pacotes. A ideia de base consiste em usar

do lado do receptor uma fila de espera, chamada *playback buffer*, dos pacotes recebidos e só ir processando (*e.g.*, “tocando”) esses pacotes quando chega o momento certo. A mostra do estado destes *playback buffers* de recepção é aliás comum nas aplicações de visualização de vídeo, como está ilustrado na Figura 3.14.

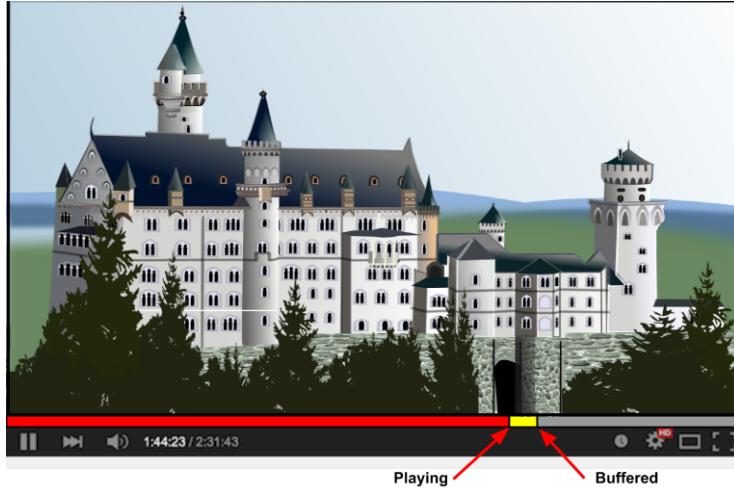


Figura 3.14: Cliente de visualização de vídeo pondo em evidência que o *playback buffer* contém informação multimédia ainda não visualizada

Vamos agora descrever o procedimento genérico usado (o leitor pode seguir a explição com auxílio da Figura 3.15). Cada pacote recebe uma marca temporal indicando o momento em que o pacote deveria ser processado (*e.g.*, transformado em som ou numa imagem a mostrar ao utilizador) admitindo que os pacotes chegam ao receptor instantaneamente, ou seja, como se fossem processados localmente. Designemos essas marcas por tempo de emissão ou t_e e a sua sucessão por $t_e(0), t_e(1), t_e(2), \dots, t_e(i), \dots$

O método requer que cada pacote tenha uma marca temporal distinta pois os diferentes pacotes podem ser de tamanhos distintos devido à utilização de técnicas de compressão. Por exemplo, caso as marcas fossem em milissegundos, poderiam corresponder a uma sucessão de valores como por exemplo 0, 22, 55, 83, 100, 126, 155, 187, 210, 240, 264,

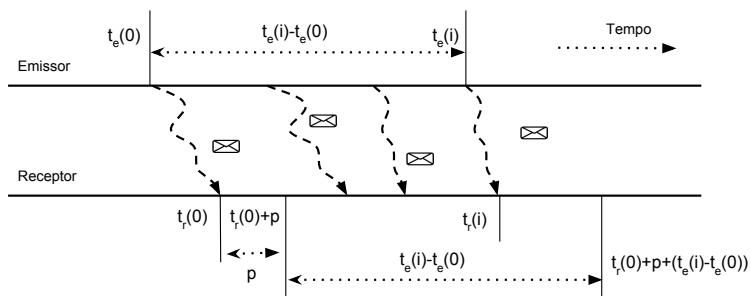


Figura 3.15: Utilização de marcas temporais, de um *playback buffer* e de um *playout delay* (P) para compensar o *jitter*

Seja $t_r(i)$ o tempo em que o receptor recebe o pacote $t_e(i)$. O receptor recebe os pacotes (com as marcas $t_e(i)$) e coloca-os no *playback buffer*. Seja p um compasso de espera usado pelo receptor, chamado na literatura em língua inglesa o *playout delay*. Quer isto dizer que se o pacote com a marca temporal $t_e(0)$ chegou ao receptor no momento $t_r(0)$, então ele deve ser processado no momento $t_r(0) + p$. O problema agora consiste em saber quando deve ser processado cada um dos pacotes seguintes, nomeadamente o pacote i . A resposta é no momento $t_r(0) + p + (t_e(i) - t_e(0))$ pois o tempo entre o momento em que foi processado pelo receptor o primeiro pacote, e o momento em que deve ser processado o pacote i , deve ser o mesmo que medeia entre a marca temporal do pacote 0 e a marca temporal do pacote i . O algoritmo 3.1 permite ao receptor processar os pacotes recebidos com os intervalos adequados.

Listing 3.1: Algoritmo de compensação do *jitter* através de um *playout delay* de duração p segundos

```
// Given packet p, p.getTimeStamp() returns its timeStamp, clock() returns
// the local time and wait(t) blocks the execution for t seconds
//
// Wait for the first packet using method getPacket
packet = getPacket()
te0 = packet.getTimeStamp()
tr0 = clock()
wait(p)
packet.play()
do {
    packet = getPacket() // wait for next packet
    timeToWait = clock() - tr0 + p + (packet.getTimeStamp() - te0 )
    wait( max (0, timeToWait)
    packet.play()
} while (there are more packets to receive)
```

O método é perfeito caso a aplicação tolere um valor muito elevado de p , o *playout delay*, pois existe grande margem de variação do tempo de trânsito e, quando os pacotes devem ser processados, estão sempre disponíveis. No entanto, tempos de espera muito elevados causam desconforto ao utilizador (o vídeo só é mostrado muitos segundos depois de o seu *download* começar) e certas aplicações, como as interactivas, são inviáveis quando é necessário usar um compasso de espera muito elevado.

Por isso, idealmente, o valor do compasso de espera deve ser o mais curto que possível. Nesse caso, se a rede tiver grandes variações de tempo de trânsito, alguns pacotes podem só estar disponíveis já depois do momento em que deveriam ter sido processados pois o *playback buffer* ficou vazio e deixou de haver margem de manobra suficiente. Nessa situação é necessário alargar o *playout delay*, voltar a criar uma margem de segurança acumulando pacotes antes de os visualizar e diz-se que ocorreu *rebuffering*. Nessas situações o utilizador apercebe-se da incapacidade da rede para responder atempadamente às necessidades da aplicação.

Quando as aplicações não toleram a variação do tempo de trânsito de extremo a extremo dos dados (*jitter*), é possível introduzir, através de marcas temporais nos pacotes e *playback buffers* no receptor, técnicas de compensação que compensam o efeito dessa variação. A possibilidade de realizar adequadamente essa compensação está limitada pelo compasso de espera (*playout delay*) que é necessário introduzir do lado do receptor.

Dependendo das aplicações, existem formas diferentes de lidar com estes problemas e de tentar estimar o *playout delay* mais adequado. Voltaremos a discutir este assunto no capítulo 9.

3.5 Comparação entre os dois modelos

As redes de comutação de circuitos são caracterizadas por:

1. Antes de poder comunicar, é necessário solicitar à rede que estabeleça um circuito.
2. Se o circuito for estabelecido, a rede reservou capacidade que lhe fica afectada enquanto este estiver activo.
3. Um circuito só pode usar a capacidade que lhe foi afectada.
4. Mesmo que os interlocutores não tenham nenhuma informação a transmitir, a reserva de capacidade na rede mantém-se e essa capacidade de comunicação não pode ser usada por outros circuitos.

Estas redes são especialmente adequadas a situações onde são necessárias garantias, quer para o tempo de trânsito, quer para a disponibilidade de capacidade de comunicação, quer ainda de ausência de *jitter*. Numa rede com recursos relativamente escassos, os circuitos permitem que aplicações com aqueles requisitos funcionem, mas à custa de uma rigidez elevada na reserva de recursos, do desperdício que essa reserva implica, e de impedir a utilização da rede quando não há recursos suficientes disponíveis.

Na secção 1.3 introduzimos a noção de rede IP (Internet Protocol). Essa rede é uma rede de comutação de pacotes que funciona segundo o modelo do “melhor esforço”, sem reserva *a priori* de recursos (capacidades dos canais, caminhos, *etc.*) para os diferentes fluxos de dados que a atravessam. Como vimos, no limite, perante anomalias ou em caso de saturação da rede, esta não é obrigada a encaminhar todos os pacotes e pode descartar alguns, sem ser sequer obrigada a avisar os respectivos emissores. Uma rede IP é um dos exemplos mais simples de rede a funcionar segundo o princípio da comutação de pacotes, que pode, nesse caso, ser chamada de **rede de comutação de pacotes sem conexão**, e é assim caracterizada:

1. Antes de poder comunicar, não é necessário solicitar à rede que estabeleça nenhuma ligação entre a origem e o destino dos pacotes a emitir.
2. O consumo da capacidade de um canal pelos diferentes fluxos de dados é de carácter estatístico, e depende do conjunto do tráfego que atravessa o canal.
3. Não existe *a priori* garantia de capacidade entre interlocutores, nem garantias de tempo de trânsito mínimo ou de *jitter* máximo.
4. Se um emissor não emitir momentaneamente pacotes, não consome nenhum recurso da rede.
5. A capacidade dos diferentes canais que um fluxo de pacotes pode usar está dependente da capacidade consumida pelos outros fluxos.
6. Em caso de saturação das suas filas de espera, um comutador pode suprimir pacotes sem ser obrigado a avisar o emissor ou os outros comutadores.

Uma rede de pacotes é mais simples de operar globalmente, mais flexível e mais adequada a aproveitar os recursos, pois não os desperdiça devido às reservas. Em contrapartida, não oferece nenhuma garantia aos seus utilizadores e nem sequer é obrigada, no caso das redes IP, a notificá-los das dificuldades ou das anomalias. Perante recursos escassos, não suporta senão aplicações de transferência de dados, sem necessidade de garantias de capacidade, tempo de trânsito ou *jitter*.

Nos primórdios das redes digitais, redes como a rede Internet apenas podiam ser usadas por aplicações de dados, enquanto que as aplicações tradicionais das redes de telecomunicações (chamadas telefónicas, vídeo, *etc.*) apenas eram suportadas por redes de circuitos, ou por redes especiais, como por exemplo as dedicadas à difusão de sinal de televisão.

Com a subida exponencial da capacidade dos canais de dados, a descida do seu preço, e a generalização de canais de longa distância baseados em fibra óptica, com baixas taxas de erro e velocidades muito elevadas de transmissão (maiores ou iguais à dezena de Gbps), as redes de pacotes, e a Internet em particular, oferecem uma qualidade de serviço suficiente para satisfazer as necessidades da maioria das aplicações com requisitos especiais como os acima referidos.

Adicionalmente, com o desenvolvimento de técnicas de compensação do *jitter*, mesmo aplicações interactivas com requisitos mais severos no que diz respeito ao tempo de trânsito e à sua variação, passaram a poder ser suportadas por redes de pacotes, e estas tornaram-se completamente dominantes.

Quer isto dizer que as redes de circuitos estão completamente abandonadas? A resposta não é simples pois as soluções tecnológicas estão sempre dependentes de um contexto e de factores que tornam algumas soluções superiores a outras. Se os contextos mudam, soluções consideradas inadequadas podem voltar a ter um lugar.

As redes de pacotes começaram por ser adoptadas exclusivamente para situações onde as aplicações que se adaptam facilmente a débitos variáveis (transferência de ficheiros, correio electrónico, Web, etc.) eram predominantes. Mas actualmente, com a subida generalizada da capacidade dos canais, a grande maioria das aplicações sem aquela capacidade (e.g., telefone, televisão, etc.), que estavam reservados às redes de circuitos antigas, passaram também a ser fornecidas através de redes de pacotes como a Internet. Os ganhos em flexibilidade e uniformização são duas vantagens evidentes.

No entanto, existem situações em que redes baseadas na comutação de circuitos continuam a ser usadas, mas já não segundo a filosofia inicial. Nessa redes, os circuitos são estabelecidos por períodos longos e proporcionam a optimização da transferência contínua de grandes quantidades de informação. Este tipo de soluções estão presentes, por exemplo, no coração da rede dos maiores operadores, ou em redes com requisitos muito especiais, mas a sua discussão aqui seria deslocada.

3.6 Indicadores de desempenho

Dada a forma como funcionam as redes de pacotes, a caracterização do seu desempenho é mais difícil e requer que se tomem em consideração mais indicadores do que os usados para caracterizar os canais (e.g., débito, tempo de propagação, taxa de erros). A seguir listam-se alguns dos mais relevantes.

Taxa de utilização de um canal (*link utilization*) A multiplexagem estatística introduz a possibilidade de um canal estar inactivo durante alguns períodos. A taxa de utilização de um canal representa a fracção de tempo durante a qual o canal está activo a transmitir, e deve ser medida periodicamente (de minuto a minuto por exemplo).

Tempo de trânsito de extremo a extremo (*end-to-end delay*) Este indicador foi extensivamente discutido neste capítulo e, como é evidente, só pode ser caracterizado estatisticamente, através de médias, mínimos, máximos, etc. ou de forma mais completa através de um conjunto alargado de amostras ou de uma distribuição estatística. Na literatura de língua inglesa por vezes usa-se o termo *latency* para designar o valor mínimo do tempo de trânsito.

Variabilidade do tempo de trânsito (*jitter*) Este indicador foi extensivamente discutido e pode ser definido formalmente como a variância do tempo de trânsito de extremo a extremo.

Taxa de perda de pacotes de extremo a extremo (*end-to-end packet drop rate*) Fracção de pacotes que não chegam ao destino na totalidade dos pacotes emitidos. Esta taxa deve ser medida periodicamente e, em geral, só pode ser caracterizada estatisticamente.

Taxa ou débito de transferência de extremo a extremo (*end-to-end throughput*)

Quantidade de bits que são transferidos por unidade de tempo entre dois computadores ligados à rede. Esta quantidade deve ser medida periodicamente e, em geral, só pode ser caracterizada estatisticamente.

Sempre que é apresentado um único valor sobre um dos indicadores que têm de ser caracterizados estatisticamente, esse valor refere-se à média (sem especificar o intervalo específico em que a mesma foi avaliada). Trata-se de uma caracterização incompleta mas que pode revelar-se suficiente para uma estimativa.

Por exemplo, quando se diz que a taxa de perda de pacotes de extremo a extremo é sempre inferior a 0,2%, a afirmação pode ser suficiente em certos contextos, mas é incompleta pois não especifica se esta taxa foi avaliada de segundo a segundo, de minuto a minuto, de hora a hora, de mês a mês ou de ano a ano.

Se a taxa de perda de pacotes for de 50% durante cinco minutos, a taxa calculada durante um dia pode ser inferior a 0,2% ($2,5/(60 * 24) = 2,5/1440 = 0,00173$). No entanto, durante aqueles 5 minutos, a rede praticamente não podia ser usada.

O desempenho de certas aplicações apenas está dependente da taxa de transferência de dados, pois estas adaptam-se à capacidade de transferência que estiver disponível. Os exemplos clássicos são a transferência de ficheiros, o correio electrónico e, até em larga medida, o acesso à Web. Estas aplicações dizem-se **aplicações elásticas**.

Ao contrário, outras aplicações têm requisitos temporais no que diz respeito aos indicadores de débito e tempo de trânsito extremo a extremo e *jitter*, e só conseguem funcionar adequadamente se estes dois indicadores se mantiverem dentro de um intervalo de valores específico. Alguns exemplos de aplicações desta categoria são os jogos, as aplicações multimédia e as aplicações interactivas. Estas aplicações dizem-se **aplicações não elásticas** ou **aplicações com requisitos de qualidade de serviço**.

Aplicações elásticas são aplicações que se adaptam facilmente a redes de pacotes, mesmo sub-dimensionadas, pois apenas estão dependentes do indicador taxa de transferência de dados, e conseguem progredir mesmo com valores baixos do mesmo.

Exemplos: transferência de ficheiros, acesso a correio electrónico, acesso a páginas Web, etc.

Aplicações não elásticas ou com requisitos de qualidade de serviço são aplicações que só conseguem funcionar adequadamente se os indicadores débito e tempo de trânsito extremo a extremo e *jitter* tiverem valores contidos em intervalos específicos. Numa rede de pacotes sem políticas especiais de gestão das filas de espera, estas aplicações só funcionam se a rede estiver adequadamente dimensionada.

Exemplos: aplicações interactivas, aplicações multimédia, jogos, acesso a serviços críticos, etc.

3.7 Resumo e referências

Resumo

As primeiras redes de informação da era moderna foram as redes telefónicas, que eram redes baseadas no conceito de circuito telefónico. Grosso modo, um circuito telefónico consiste numa ligação dinâmica, estabelecida a pedido, entre dois aparelhos telefónicos.

Inicialmente, os circuitos telefónicos eram materializados por uma concatenação de fios de cobre, mas com o evoluir da rede, passaram a ser materializados pela concatenação de um conjunto de sub-canais digitais dedicados a cada chamada telefónica.

As redes que suportam este tipo de funcionalidades usam multiplexagem em frequência ou temporal de canais, chamam-se redes de comutação de circuitos e podem ser caracterizadas por garantirem, através de reservas, a capacidade necessária ao funcionamento dos circuitos que aceitam estabelecer.

Com o aparecimento das primeiras aplicações baseadas na troca de dados digitais, como por exemplo a transferência de ficheiros, ficou claro que as redes de circuitos eram pouco flexíveis e eficazes. Introduziram-se então diversas técnicas que permitiam responder de forma mais eficaz a este novo contexto: multiplexagem estatística, troca de pacotes de dados, funcionamento *store & forward* e introdução de filas de espera de pacotes de dados nos equipamentos de comutação. Estas novas redes, de que a Internet é um exemplo popular, chamam-se redes de comutação de pacotes.

Apesar de mais flexíveis, as redes de comutação de pacotes introduzem atrasos suplementares e variáveis no tempo de trânsito dos pacotes de dados desde o emissor até ao receptor. Esses atrasos e as suas variações podem ter um impacto tal, em redes inadequadamente dimensionadas, que torna impossível o funcionamento de aplicações com requisitos especiais de qualidade de serviço (jogos, multimédia, etc.). No entanto, quando mantidos dentro de valores adequados, podem ser compensados pelas aplicações usadas nos extremos da rede, através de marcas temporais nos pacotes e compassos de espera suplementares.

O tempo de trânsito dos pacotes e o caminho que estes seguem numa rede de pacotes a usar os protocolos TCP/IP, como a Internet, podem ser obtidos através dos programas **ping** e **traceroute**, disponíveis em todos os sistemas de operação modernos.

Com o progressivo aumento do débito dos canais disponíveis, tornou-se possível o dimensionamento mais adequado (e às vezes até o sobre dimensionamento) das redes de pacotes. Este facto, aliado ao desenvolvimento de técnicas cada vez mais sofisticadas para esconder as variações de débito e de tempo de trânsito dos fluxos de pacotes, tornou realista a utilização de redes de pacotes para suportar todas as aplicações, independentemente dos seus requisitos de qualidade de serviço. Por isso, as redes de comutação de pacotes são hoje em dia dominantes.

Apesar disso, existem situações em que se continuam a usar redes de comutação de circuitos para fornecer canais dedicados de muito alta capacidade e duração significativa. Estas soluções são sobretudo usadas no interior das redes dos operadores ou em redes com requisitos muito especiais.

Os principais termos introduzidos ou referidos neste capítulo são a seguir passados em revista. Entre parêntesis figuram as traduções mais comuns em língua inglesa.

Multiplexagem (*multiplexing*) Sub-divisão de um canal em sub-canais independentes. Geralmente é realizada através de uma divisão fixa da capacidade do canal entre os diferentes sub-canais.

Comutadores de circuitos (*circuit switches*) Equipamentos de rede capazes de estabelecerem a concatenação de um conjunto de sub-canais de igual capacidade para materializar um circuito de dados.

Rede de comutação de circuitos (*circuit switching network*) Rede baseada na filosofia dos circuitos, em que só é possível duas entidades comunicarem depois de a rede ter disponibilizado um circuito entre ambas.

Multiplexagem estatística (*statistical multiplexing*) A multiplexagem estatística contempla uma afectação flexível da capacidade do canal a cada sub-canal, não havendo desperdício da capacidade disponível.

Comutador de pacotes (*packet switch*) Equipamento de rede capaz de processar pacotes de dados e encaminhá-los entre os seus diferentes canais de entrada e saída.

Rede de comutação de pacotes (*packet switching network*) Uma rede a funcionar segundo o princípio da comutação de pacotes, é uma rede que encaminha pacotes de dados através de comutadores de pacotes que funcionam segundo o modelo *Store & Forward*, e que gerem a afectação dos canais aos diferentes fluxos de pacotes usando multiplexagem estatística e filas de espera.

Taxa de utilização de um canal (*link utilization / usage*) A multiplexagem estatística introduz a possibilidade de um canal estar inactivo durante alguns períodos. A taxa de utilização de um canal representa a fração de tempo durante a qual o canal está activo a transmitir.

Tempo de trânsito de extremo a extremo (*end-to-end delay*) O tempo total que leva um pacote a transitar entre o emissor e o receptor. As redes de pacotes introduzem atrasos variáveis suplementares no tempo de trânsito. Na literatura de língua inglesa por vezes usa-se o termo *latency* para designar o valor mínimo do tempo de trânsito.

Variância do tempo de trânsito (*jitter*) A variância do tempo de trânsito de extremo a extremo.

Supressão de pacotes pelos comutadores (*packet dropping*) Como as redes de pacotes limitam a dimensão máxima das filas de espera dos comutadores, suprimem pacotes para manter aqueles limites.

Taxa de perda de pacotes de extremo a extremo (*end-to-end packet drop rate*) A fração de pacotes que não chegam ao destino na totalidade dos pacotes emitidos.

Débito ou taxa de transferência de extremo a extremo (*end-to-end throughput*) A quantidade (média) de bits que são transferidos por unidade de tempo entre dois computadores ligados à rede.

Compensação da variância do tempo de trânsito (*jitter compensation*) Quando as aplicações não toleram *jitter* significativo, é possível introduzir, através de marcas temporais e *buffers* no receptor, técnicas de compensação que cancelam o efeito dessa variação. A possibilidade de realizar adequadamente essa compensação está limitada pelo compasso de espera (*playout delay*) suplementar que é necessário introduzir do lado do receptor

Qualidade de serviço (*quality of service*) Conjunto de indicadores (e.g., taxa de transferência de extremo a extremo, tempo de trânsito de extremo a extremo e *jitter etc.*), cuja definição permite caracterizar os requisitos para o funcionamento de uma aplicação

Aplicações elásticas (*elastic applications*) As aplicações que se adaptam facilmente a redes de pacotes, mesmo sub-dimensionadas, pois apenas estão dependentes da taxa de transferência de dados, e conseguem progredir mesmo com valores baixos da mesma, dizem-se aplicações elásticas. Exemplos: transferência de ficheiros, acesso a correio electrónico, acesso a páginas Web, etc.

Aplicações não elásticas (*non-elastic applications*) As aplicações que só conseguem funcionar adequadamente se certos parâmetros de qualidade de serviço forem satisfeitos. Exemplos: aplicações interactivas, aplicações multimédia, jogos, acesso a serviços críticos, etc.

Referências

Este capítulo apresenta o conceito de comutação de pacotes e explica a arquitectura e o funcionamento de uma rede baseada no mesmo. A explicação inclui a apresentação da filosofia da arquitectura de rede alternativa, baseada na comutação de circuitos, assim como a explicação dos diversos factores, ligados à evolução das tecnologias de

comunicação e dos padrões de comunicação, que justificam a adopção progressiva da comutação de pacotes.

Apesar desses factores, no interior da rede, existem canais que continuam a ser disponibilizados como circuitos (permanentes) de uma rede de circuitos e, por razões de optimização das infra-estruturas, o conceito de circuito continua a ter relevância. O leitor que pretenda aprofundar a problemática das redes de circuitos poderá estudar os capítulos 8 e 9 de [Stallings, 2013] e parte do capítulo 2 de [Tanenbaum and Wetherall, 2011].

Em [Pierce, 1984] J. Pierce apresenta, de um ponto de vista histórico, o sistema telefónico, mostrando que o mesmo correspondeu ao primeiro sistema que permitiu a comunicação directa pessoa a pessoa em tempo real. A sua realização envolveu ultrapassar desafios tecnológicos e de escala que nunca tinham sido até então afrontados pelos engenheiros.

O conceito de comunicação através da comutação de pacotes foi inventado simultaneamente por diversos investigadores, em diversos países, entre os anos de 1960 e 1970. Leonard Kleinrock, então estudante de doutoramento no M.I.T, foi dos primeiros a interessar-se pelo novo conceito e a demonstrar que o seu desempenho era mais eficaz para o suporte da comunicação de dados. Por essa razão, Kleinrock é o autor de alguns dos primeiros estudos de carácter probabilístico sobre o desempenho das redes de pacotes e publicou um livro [Kleinrock, 1972] onde esses estudos são apresentados de forma sistemática.

Apontadores para informação na Web

- <http://www.ccs-labs.org/teaching/rn/animations/queue/index.htm> – Contém um programa que permite visualizar o funcionamento de um comutador de pacotes do ponto de vista da gestão de uma fila de espera de saída.
- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- <http://www.iplocation.net> – É o endereço de um serviço que permite localizar geograficamente um endereço IP. Apresenta o resultado fornecido por diferentes operadores deste tipo de serviços.
- <http://ping.eu> – É o endereço de um dos muitos serviços que permitem realizar testes na linha de comando dos programas ping e traceroute na Internet.

3.8 Questões para revisão e estudo

1. Caracterize uma rede baseada na comutação de circuitos.
2. Defina o que é *time division multiplexing* e o que é *frequency division multiplexing*.
3. Numa rede baseada na comutação de circuitos caracterize o atraso introduzido por cada comutador.
4. Um canal com o débito de 1 Gbps foi subdividido em sub-canais através de multiplexagem temporal (TDM – *time division multiplexing*). Os 75 *time slots* usados são de dimensões diferentes: 25 de classe A com a duração t segundos e 50 de classe B com a duração $t/2$ segundos. Por hipótese, o mecanismo TDM usado não introduz nenhum desperdício da capacidade do canal. Qual a capacidade afectada a cada um dos circuitos que usam sub-canais de classe B?
5. Caracterize uma rede baseada na comutação de pacotes.
6. Que aspectos são importantes para distinguir um comutador de pacotes de um *multiplexer / demultiplexer* estatístico?

7. Numa rede baseada na comutação de pacotes caracterize o atraso introduzido por cada comutador.
8. Considere uma rede que interliga dois computadores através de 4 comutadores. Todos os canais dessa rede são ponto-a-ponto, bidireccionais *full-duplex*, com 1 Mbps de débito em cada sentido e têm um tempo de propagação desprezável. Calcule o tempo de trânsito entre os dois computadores de um pacote com 10.000 bits em cada um dos seguintes casos:
 - (a) A rede é de comutação de circuitos e cada comutador introduz 0,1 ms de atraso.
 - (b) A rede é de comutação de pacotes e não há outro tráfego na rede.
9. Um pacote chegou ao comutador de pacotes A, foi processado, transmitido por um canal que liga A a B e finalmente chegou ao comutador B. Indique o conjunto de factores que contribuem para o tempo total que decorreu desde que o pacote chegou a A até que foi recebido por B.
10. Escolha das seguintes, as opções válidas, tendo em consideração que pode haver mais do que uma. A multiplexagem estatística é superior à afectação fixa do tráfego através de multiplexagem temporal porque:
 - (a) o tráfego de dados é irregular e pode variar de fluxo para fluxo;
 - (b) a multiplexagem estatística garante a capacidade usada por cada fluxo;
 - (c) a multiplexagem estatística permite uma melhor utilização de um canal, *i.e.*, evita o desperdício da capacidade disponível na rede;
 - (d) a multiplexagem estatística garante um menor *jitter*;
 - (e) a multiplexagem estatística garante um tempo de transferência de extremo a extremo menor.
11. Escolha das seguintes, as opções válidas, tendo em consideração que pode haver mais do que uma.
 - (a) Com a comutação de pacotes a distribuição da capacidade dos canais pelos diferentes fluxos de pacotes é mais flexível pois não há reserva *a priori* de capacidade para cada fluxo.
 - (b) Com a comutação de pacotes a garantia de capacidade e tempo de trânsito extremo a extremo dos pacotes dos diferentes fluxos está garantida.
 - (c) Com a comutação de pacotes o tempo de trânsito extremo a extremo dos pacotes é constante.
 - (d) Com a comutação de circuitos o caminho seguido pelos pacotes da origem até ao destino é sempre o mesmo.
12. Quais destes factores justificam a existência de *jitter* no tráfego extremo a extremo de uma rede de pacotes?
 - (a) dimensão das filas de espera;
 - (b) débito dos canais;
 - (c) dimensão dos pacotes;
 - (d) dimensão dos canais;
 - (e) tempo de processamento pelos comutadores;
 - (f) volume dos canais.

13. Considere uma rede de pacotes em que o RTT mínimo entre Lisboa e Moscovo é 90 ms. Todos os canais da rede têm o débito de 100 Mbps. Quantos bits cabem dentro do *canal lógico* que vai de Lisboa a Moscovo e volta a Lisboa (excluindo os bits nas filas de espera dos comutadores)? Escolha uma das seguintes opções: 900, 9000, 90.000, 900.000, 9.000.000, 90.000.000.
14. Uma aplicação necessita de transmitir informação ininterruptamente durante uma hora com o débito constante de 100 Kbps. Admitindo que pode tomar uma decisão sem considerar quaisquer outros factores, seria preferível suportar essa aplicação numa rede de pacotes, ou numa rede de circuitos? Justifique a sua resposta.
15. Dois computadores A e B enviam pacotes para o computador C através de uma rede de pacotes. A envia 100 pacotes por segundo de 5.000 bits cada. B envia 200 pacotes por segundo também de 5.000 bits cada. Os pacotes de A e B chegam ao comutador S que liga directamente a C através de um canal com o débito de 1 Mbps e uma fila de espera, por hipótese, infinita. C só está a receber pacotes de A e B. Qual o débito extremo a extremo do tráfego de A para C?
16. Considere, salvo indicação em contrário, que todos os canais da rede são ponto-a-ponto, bidirecionais, *full-duplex*, têm o débito de 1 Mbps e não têm erros. Pretende-se transmitir do computador A para o computador B um ficheiro com 1×10^6 bits em diversos cenários. Indique, em cada um deles, quanto tempo decorre desde que a transmissão do ficheiro por A tem início, até que B tenha uma cópia integral do ficheiro. Considere ainda que o tempo de processamento pelos comutadores é desprezável, que a dimensão dos cabeçalhos dos pacotes também e que a velocidade de propagação do sinal nos canais é de 200.000 Km por segundo.
 - (a) A e B estão directamente ligados por um canal, com 1.000 Km, que suporta a transmissão do ficheiro num único pacote. Calcule também a taxa de transferência de extremo a extremo conseguida.
 - (b) A e B estão directamente ligados ao mesmo comutador de pacotes por canais com 500 Km cada um. Os canais e o comutador suportam a transmissão do ficheiro num único pacote. Nenhum outro tráfego atravessa o comutador. Calcule também a taxa de transferência de extremo a extremo conseguida.
 - (c) A e B estão directamente ligados ao mesmo comutador de pacotes por canais com 500 Km cada um. Os canais e o comutador suportam pacotes com no máximo 10.000 bits de dimensão. Nenhum outro tráfego atravessa o comutador. Calcule também a taxa de transferência de extremo a extremo conseguida.
 - (d) A e B estão directamente ligados ao mesmo comutador de pacotes por canais com 500 Km cada um. Os canais e o comutador suportam pacotes com no máximo 10.000 bits de dimensão. Em média, cada pacote que transita de A para B via o comutador encontra na fila de espera à sua frente 2 pacotes com 5.000 bits cada um. Calcule também a taxa de transferência de extremo a extremo conseguida.
 - (e) A e B estão directamente ligados ao mesmo comutador de pacotes por canais com 500 Km cada um mas, ao contrário dos cenários anteriores, o canal que liga A ao comutador apenas tem o débito de 500 Kbps ao invés de 1 Mbps. Os canais e o comutador suportam pacotes com no máximo 10.000 bits de dimensão. Nenhum outro tráfego atravessa o comutador. Calcule também a taxa de transferência de extremo a extremo conseguida.

17. Dois computadores A e B estão ligados cada um a um comutador de pacotes distinto através de canais com o débito de 1 Gbps e 100 metros de comprimento. Os dois comutadores estão ligados por um canal com o débito de 1 Mbps e com um tempo de propagação de 10 ms. Fizeram-se testes com o programa `ping` para obter dados sobre o RTT entre A e B.

- (a) O resultado final foi o seguinte:

```
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 4.04 / 5.19 / 6.81 / 0.4 ms
```

este resultado é credível?

- (b) Fizeram-se testes em duas ocasiões distintas, obtendo-se os resultados finais seguintes:

```
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 21.04 / 22.19 / 23.81 / 0.4 ms
```

```
10 packets transmitted, 8 packets received, 20% packet loss
round-trip min/avg/max/stddev = 21.04 / 200.19 / 1030.81 / 450 ms
```

Os resultados são ambos possíveis? Que poderá explicar a diferença entre os dois resultados?

18. O computador A está ligado directamente ao computador B através de uma canal C, ponto-a-ponto, dedicado e *full-duplex*, com cerca de 100 metros de comprimento, mas cujo débito é desconhecido. Em A usou-se o programa `ping` para medir o RTT. Os pacotes usados pelo programa `ping` para fazer o teste têm cerca de 1000 bits. O resultado foi o seguinte:

```
--- ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 4.004 / 4.019 / 4.051 / 0.001 ms
```

Qual dos seguintes pode ser o débito do canal C: 1 Kbps, 10 Kbps, 100 Kbps, 1 Mbps, 1 Gbps ou nenhum destes?

19. Mediú-se repetidamente o tempo de trânsito de pacotes de 100 bytes entre dois computadores ligados através de uma rede de pacotes e verificou-se que os valores medidos variavam entre 10 ms e 300 ms, com um valor médio de 50 ms e uma variância elevada. Justifique o que está na origem deste comportamento? O mesmo seria possível numa rede de circuitos?

20. Dois computadores A e B estão ligados a uma rede de pacotes cuja configuração é desconhecida. Apenas se sabe que todos os canais são ponto-a-ponto, *full-duplex* e têm uma taxa de erros desprezável. Fizeram-se testes com o programa `ping` para obter dados sobre o RTT entre A e B e o resultado final foi o seguinte:

```
--- B ping statistics ---
10 packets transmitted, 8 packets received, 20% packet loss
round-trip min/avg/max/stddev = 21.04 / 200.19 / 1030.81 / 450 ms
```

Sabendo que o tráfego gerado pelo teste era o único tráfego que passava na rede entre A e B, os dois computadores estão ligados directamente por um canal ou o tráfego entre eles atravessa comutadores de pacotes? Justifique a sua resposta.

21. Dois computadores A e B estão ligados através de uma rede de pacotes. Os pacotes que transitam de A para B têm n bits de comprimento, atravessam r comutadores e $r + 1$ canais ponto-a-ponto. Todos os canais têm 100 metros de comprimento e o débito de 1 Mbps.

- (a) Qual o tempo de trânsito de extremo a extremo de pacotes entre A e B em função de r e n ?

- (b) Mesma questão mas neste caso os canais que ligam os comutadores de pacotes têm 1.000 Km de comprimento. O sinal propaga-se a 200.000 Km por segundo.
22. Dois computadores A e B estão ligados através de um conjunto de canais e comutadores de pacotes. Pretende-se transferir de A para B um ficheiro que pode ser transferido usando F pacotes, todos de igual dimensão. Entre A e B existem r comutadores e $r+1$ canais iguais. Esses canais são ponto-a-ponto, *full-duplex*, têm uma taxa de erros desprezável, os pacotes usados são transmitidos nos mesmos em t ms e o tempo de propagação de cada canal é $2 \times t$ ms. Calcule o tempo de transferência do ficheiro de A para B. Considere, por hipótese, que o tráfego correspondente à transmissão do ficheiro é o único existente nos canais e comutadores atravessados.
23. Pretende-se transmitir um ficheiro com x Mbytes através de uma rede de pacotes em que o mesmo tem de atravessar y canais, cada um dos quais com z metros de comprimento e com a velocidade de transmissão de q Mbps. Quanto tempo leva a transmitir o ficheiro caso o mesmo seja transmitido numa sequência de pacotes com r bits cada um?
- Considere apenas o tempo necessário para transmitir de forma contínua pacotes, sem perdas nem retransmissões. Assuma que a velocidade de propagação do sinal nos canais é $c = 200.000.000$ metros por segundo. Assuma também que só há este tráfego na rede e que portanto não há filas de espera. Para todos os efeitos tem de considerar o tempo de transmissão e o tempo de propagação de todos os bits que formam o ficheiro pelos y canais e o resultado tem de ser válido para casos limites quando por exemplo $y = 1$ ou $x \times 2^{20} \times 8 = r$.
- Qual das seguintes é a resposta certa?
- $y \times ((r/q \times 10^{-6}) \times (8 \times x \times 2^{20}/r))$
 - $(x \times 2^{20} \times 8)/q \times 10^{-6} + y \times z/c + (y - 1) \times (r/q \times 10^{-6})$
 - $(x \times 2^{20} \times 8)/q \times 10^{-6} + y \times z/c + y \times (r/q \times 10^{-6})$
24. Um computador emitiu um pacote IP e inicializou o campo TTL (*time to live*) do pacote com o valor n . Qual o comprimento máximo do caminho que esse pacote pode percorrer dentro da rede?
25. Descreva como o protocolo ICMP é usado pelo programa **traceroute** para determinar a rota de um pacote da origem até ao destino. Caso existam rotas assimétricas (o caminho de ida é diferente do de volta), qual delas o programa determina?
26. Três computadores A, B, C estão ligados através de canais a um mesmo comutador de pacotes, formando uma rede de pacotes em estrela com o comutador ao centro. O computador A está a enviar, em média, para o computador C, 200 pacotes por segundo (pps), em média com 512 bytes cada um. O computador B está a enviar, em média, para o computador C, 50 pps, em média com 1024 bytes cada um. A interface de saída do comutador que liga ao computador C tem uma fila de espera de pacotes, gerida segundo uma política FIFO, raramente vazia. Indique a percentagem média do débito do canal que liga o comutador a C que está ocupada pelo tráfego entre A e C sabendo que o tráfego entre A, B e C é o tráfego dominante nesse canal (indique dos valores abaixo qual o que se aproxima mais da resposta certa):
- 0% 10% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 100%
27. O computador A em Lisboa usou o programa **ping** para testar o tempo de trânsito ida e volta (RTT) para um computador B. O resultado final do teste foi:

```
--- ping statistics ---
100 packets transmitted, 100 packets received, 0% packet loss
round-trip min/avg/max/stddev = 8.04 / 8.19 / 8.51 / 0.01 ms
```

- (a) Em qual das seguintes cidades de destino pode estar o computador B?
 Porto, PT (cerca de 700 Km ida e volta), Londres, UK (cerca de 4.500 Km ida e volta) ou New York, EUA (cerca de 12.000 Km ida e volta).
- (b) A está a enviar para B tráfego multimédia colocando marcas temporais nos pacotes emitidos. Qual o menor dos seguintes valores de *playout delay* B deve usar para compensar o *jitter*?
 0 10 20 30 40 50 60 70 80 90 100 ms
28. Num serviço de transferência (*streaming*) de vídeo, o tempo de trânsito dos pacotes entre o emissor para o receptor varia entre 12 ms e 128 ms. Quais das seguintes afirmações são verdade nesse cenário?
- (a) Tendo em consideração o tempo de trânsito extremo a extremo, o receptor não precisa de usar um *playback buffer* de recepção, nem um compasso de espera (*playout delay*) antes de mostrar o conteúdo dos pacotes recebidos.
 - (b) Para assegurar que os pacotes estão disponíveis quando são necessários, o cliente necessita de um *playback buffer* que contenha pelo menos $12 + 128 = 140$ ms de vídeo.
 - (c) Para assegurar que os pacotes estão disponíveis quando são necessários, o cliente necessita de um *playback buffer* que contenha pelo menos 128 ms de vídeo.
 - (d) Para assegurar que os pacotes estão disponíveis quando são necessários, o cliente necessita de um *playback buffer* que contenha pelo menos $128 - 12 = 126$ ms de vídeo.
29. Você está a desenvolver uma aplicação para suportar corretores a negociarem remotamente na bolsa de Lisboa. Por causa da negociação, o corretor “Compra e Vende Depressa” disse-lhe que o tempo máximo que uma mensagem da aplicação poderia levar a chegar ao servidor da bolsa era de 10 milissegundos. Os canais que ligam os corretores ao servidor da bolsa têm uma capacidade que permite considerar que o tempo de transmissão dos pacotes é desprezável e cada comutador introduz um atraso máximo de 1 ms. Atendendo a que o número de comutadores entre o corretor “Compra e Vende Depressa” e o servidor da bolsa é 5, indique entre as seguintes opções qual a distância máxima em quilómetros entre o escritório do corretor e o servidor da bolsa. Considere que a velocidade de propagação do sinal é 200.000 Km por segundo.
 0,1 1 10 100 200 300 500 1000 1500 2000 10.000 Km
30. Dê exemplos de aplicações referidas nas questões anteriores para as quais a necessidade de qualidade de serviço é importante.
31. Considere o serviço de acesso ao DNS (Domain Name System, ver a Secção 1.5). Trata-se de um serviço elástico ou não elástico?

Capítulo 4

Princípios, modelos e ferramentas

“All models are wrong, but some are useful.”

– Autor: *George E. P. Box*

Os capítulos precedentes apresentaram diversas facetas das redes de computadores e, através dos mesmos, foi possível perceber: que a rede tem uma parte interna constituída por canais e nós de comutação de pacotes; que esses nós de comutação de pacotes funcionam segundo uma filosofia de multiplexagem estatística e gerem de forma dinâmica a afectação dos canais aos diferentes fluxos de pacotes; que existe uma divisão de responsabilidades entre, por um lado, a infra-estrutura interna constituída pelos canais e os nós de comutação e, por outro, a periferia da rede, ou seja, o software que reside nos computadores; que os nós de comutação não asseguram serviços fiáveis, nem são obrigados a notificar a periferia de eventuais anomalias que ocorram; que a periferia é constituída por uma parte, geralmente integrada nos sistemas de operação, que materializa os protocolos de transporte, correspondentes aos serviços de rede vistos pelas aplicações; que as aplicações têm estruturas e requisitos de rede muito diversos.

Com exceção da razão da opção pela utilização da comutação de pacotes, como modo de funcionamento do núcleo da rede, as outras facetas e opções de estruturação foram apresentadas sem uma justificação clara da razão de ser da sua escolha. Chegou a altura de percebermos que escolhas e decisões presidiram a estas opções. Este capítulo começa por apresentar um conjunto de princípios que conduziram às soluções apresentadas, nomeadamente o princípio de “privilegiar os extremos” (*end-to-end arguments*), o princípio do “funcionamento sem estado e entrega de pacotes com base no melhor esforço” (*stateless and best-effort delivery network*) e a forma como os princípios da modularidade (*modularity*) e da independência das partes (*separation of concerns*) foram usados nas redes de computadores.

Veremos igualmente que na arquitectura inicial da Internet os problemas da segurança, autenticação e contabilização de recursos foram protelados em favor da simplicidade e flexibilidade, e que isso tem repercussões delicadas, nomeadamente na grande popularidade de ataques que se costumam designar por “ataques de negação de serviço”, que consomem inutilmente grandes quantidades de recursos.

Uma vez abordados os princípios que presidiram ao desenho de um sistema, importa tentar encontrar formas de dimensioná-lo, estudar o seu comportamento, analisar alternativas e prever a sua evolução.

Para esse efeito usam-se modelos. Como as redes são sistemas complexos, com múltiplas componentes que interagem, é necessário recorrer a modelos simplificados,

que abstraem a maioria dos detalhes, mas permitem pôr em evidência propriedades essenciais. Com efeito, qualquer modelo abstrai um conjunto de facetas e ignora outras. Por isso, todos os modelos são incompletos (ou mesmo errados), mas alguns deles são úteis apesar disso.

No que diz respeito às redes de computadores, existem diversos modelos que se têm revelado muito úteis. Em particular os modelos baseados na teoria de grafos, para analisar a arquitectura interna da rede e a sua estruturação em sub-redes, e os modelos de camadas, que permitem separar as diferentes funções e problemas das redes em partes independentes. Faremos igualmente uma breve referência aos modelos da teoria das filas de espera, que são muito importantes para tentar prever o desempenho dos diferentes protocolos.

Finalmente, o capítulo termina fazendo uma breve referência a um conjunto de ferramentas que são usadas para o estudo do desempenho das redes e dos sistemas distribuídos. A razão de ser destas ferramentas tem a ver com o facto de existirem dois grandes tipos de modelos: os modelos analíticos e os modelos usados em simulação e emulação computacionais.

Os modelos analíticos, que se baseiam em fórmulas matemáticas, geralmente só abrangem alguns aspectos essenciais, e portanto correspondem a uma abstracção simplificada do sistema. Por vezes, é também difícil modelizar todas as facetas de um sistema complexo. Quando os modelos analíticos são insuficientes, ou difíceis de conceber, é habitual recorrer a simulações computacionais ou a maquetas (*maquettes, sketches*) dos sistemas, para tentar estudar o seu comportamento de forma indirecta. Na parte final do capítulo faremos uma breve referência a algumas ferramentas deste tipo.

4.1 Princípios

Privilegiar os extremos (*end-to-end arguments*)

Algumas aplicações têm necessidade de que os dados sejam transportados de forma fiável mas, como vimos nos capítulos anteriores, a rede internamente não assegura um serviço de transporte fiável de dados. No caso das redes TCP/IP, como a Internet, essa fiabilidade é assegurada, no essencial, pelo protocolo TCP, implementado nos extremos da rede, ou seja, nos computadores que lhe estão ligados. No entanto a fiabilidade do transporte dos dados pode ser implementada a vários níveis, como ilustra a Figura 4.1.

Porque não se optou por colocar o essencial da responsabilidade pela fiabilidade da transferência dos dados nos nós internos da rede? Existem várias razões para isso [Saltzer et al., 1984].

1. Primeiro, porque nem todas as aplicações requerem fiabilidade e caso se optasse por assegurar a fiabilidade do transporte dos dados no núcleo da rede, as aplicações que não precisassem dela teriam de pagar um processamento e uma complexidade de que não necessitariam. É como ter de pagar por algo que não se consome.
2. Segundo, porque os nós de comutação teriam dificuldade em assegurar fiabilidade em todas as situações. Por exemplo, que aconteceria se houvesse uma avaria grave num nó ou num canal? Seria necessário re-encaminhar os pacotes contendo os dados em trânsito por outro caminho. Como assegurá-lo garantindo que não se perderiam dados, nem que os mesmos não seriam entregues em duplicado? A forma mais simples de o fazer é recorrendo a alguma funcionalidade nos extremos!
3. Terceiro, mesmo que os nós assegurassem fiabilidade, como seria possível prever os extremos contra erros não detectados no interior da rede? Por exemplo,

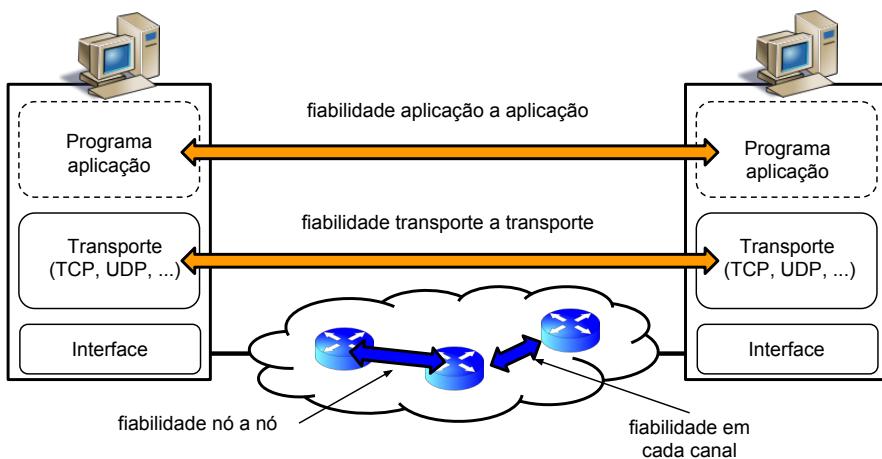


Figura 4.1: Diversas opções para garantir a fiabilidade do transporte dos dados: canal a canal, nó a nó, no protocolo de transporte, nas aplicações

como detectar, antes do cálculo de um código de controlo de erros, uma corrupção da memória de um nó de comutação que contém um pacote de dados? Para o detectar, a intervenção dos extremos é de novo necessária!

Estes casos mostram que não é possível assegurar completa fiabilidade sem intervenção dos extremos da rede. No limite, se quisermos tomar em consideração todos os cenários potenciais de erros, será mesmo necessário recorrer à intervenção das aplicações. Por isso, certas aplicações críticas verificam criptograficamente os dados que transmitem, e algumas aplicações de transferência de ficheiros verificam criptograficamente os ficheiros transferidos.

Esta forma de analisar o problema poderia levar à conclusão de que os nós de comutação e os canais devem ser libertados de todas as preocupações com a fiabilidade. Aliviar o interior da rede de toda a complexidade supérflua só a tornaria mais simples, e portanto mais escalável. Logo, parece que o ideal é concentrar o máximo de funcionalidades na periferia, e só deixar no interior aquelas que não podem deixar de lá estar.

A resposta não é assim tão simples, porque às vezes alguma intervenção das camadas inferiores torna o sistema globalmente mais eficaz. Por exemplo, se os erros de troca de bits nos canais fossem completamente ignorados pela rede, competiria às interfaces dos computadores, aos sistemas de operação, ou às aplicações, a responsabilidade pela sua detecção.

Se uma aplicação de transferência de um ficheiro verificar, no fim da transferência, que o ficheiro tem erros, o ficheiro tem de ser reemitido desde o início. Se o erro não for detectado mais cedo, tal conduz a um grande desperdício. Se for o sistema de operação a verificar os erros, pacote a pacote, devido ao tempo de trânsito de extremo a extremo, o resultado dessa opção também pode ser um débito de extremo a extremo mais baixo nos casos em que a taxa de erros é elevada.

No entanto, como o custo de detectar os erros ao nível de cada canal é bastante reduzido, ver a Secção 2.4, é preferível dividir a responsabilidade pelo controlo de erros entre o interior da rede e a periferia. Os pacotes com erros são logo descartados ao nível do canal e, caso a taxa de erros do canal seja muito elevada, o protocolo do canal deve ele próprio assegurar a reemissão dos pacotes com erros.

De qualquer forma, mesmo quando a taxa de erros é pequena e existe controlo

e re-emissão de pacotes ao nível dos canais, como o nível transporte não sabe como funcionam os diferentes canais, deve ele próprio optar por continuar a usar um mecanismo de controlo de erros suplementar, como o fazem os protocolos UDP e TCP, ver a Secção 7.2. Também, se a informação transferida sobre um canal TCP é crítica e muito valiosa, a aplicação deve ela própria aplicar controlos de extremo a extremo mais sofisticados do que um simples *Internet Checksum*.

Desenhar as interfaces e especificar as funcionalidades que devem ser implementadas a cada nível de um sistema, é uma arte que exige experiência, e é um processo sujeito a tentativas e erros. Raramente os arquitectos conseguem a melhor solução sem sucessivas iterações do desenho. Este problema é partilhado pelas redes de computadores e pelos sistemas informáticos complexos, como por exemplo os sistemas de operação [Lampson and Sproull, 1979; Lampson, 1983].

Por exemplo, dependendo de diferentes aplicações, qual o melhor modelo para os dados persistentes? Ficheiros sequenciais? Ficheiros de acesso directo? Ficheiros indexados? Repositórios chave / valor? Bases de dados? Alguns sistemas de operação mais antigos optavam por implementar vários modelos de gestão de dados persistentes directamente no núcleo. No entanto, a maioria dos sistemas modernos deixam às bibliotecas das linguagens de programação, às bases de dados, ou a outros subsistemas independentes, a tarefa de lidarem com dados estruturados e usam um modelo único mais simples.

No entanto, é também necessário considerar a economia proveniente da factorização das funcionalidades. A maioria das funcionalidades do protocolo TCP poderiam ser implementadas pelas aplicações, usando apenas UDP para transporte. Portanto, aparentemente, o único protocolo de transporte a fornecer deveria ser o UDP. As aplicações que necessitassem de canais fiáveis deveriam implementá-los usando o UDP e bibliotecas de outros sub-sistemas, usando as técnicas descritas nos capítulos 7 e 8.

Mas a funcionalidade implementada pelo protocolo TCP é tão popular que o melhor é disponibilizá-la imediatamente no núcleo do sistema de operação. Acresce, que o sistema de operação pode, eventualmente, conseguir uma implementação mais eficiente do que aquela que seria possível usando uma biblioteca ligada à aplicação.

De qualquer forma, o desenho de um sistema flexível e adaptável deve sempre permitir introduzir facilmente novas funcionalidades, cuja necessidade não tenha sido antecipada inicialmente. Por esta razão, o sistema de protocolos TCP/IP permite a introdução de outros protocolos de transporte e a sua introdução não implica nenhuma alteração do núcleo da rede.

Aqui chegados, em que ficamos? Que funcionalidades devem ser colocadas na periferia, nas camadas mais elevadas do sistema, e que funcionalidades devem ser colocadas mais abaixo, no núcleo dos sistemas?

O princípio de privilegiar os extremos (*end-to-end principle*) indica: idealmente, toda a complexidade deve ser relegada para a periferia, *i.e.*, para os extremos ou camadas superiores do sistema; no entanto, quando uma funcionalidade comum e popular pode ser implementada com menor custo e maior eficiência nas camadas mais baixas, é preferível implementá-la directamente a esse nível.

Adicionalmente, um sistema bem desenhado deve permitir flexibilidade de introdução de novas funcionalidades nas camadas superiores, com um impacto mínimo nas camadas inferiores e sem exigir a sua modificação, no limite, proporcionando maior extensibilidade.

Passamos a seguir à discussão de outro dos princípios que presidiram ao desenho da arquitectura da Internet.

Rede sem estado e de melhor esforço (*stateless best-effort delivery network*)

O desenho da arquitectura da Internet adoptou, tal como outras redes desenhadas na mesma época (*e.g.*, a rede Cyclades [Pouzin, 1975]), uma filosofia diferente das redes de telecomunicações tradicionais. Essa filosofia consistiu, para além da adopção da filosofia de rede de comutação de pacotes, em adoptar adicionalmente o princípio de que cada nó de comutação de pacotes não deve conter nenhum estado sobre os pacotes que o atravessam.

Muitas outras redes adoptaram uma versão intermédia entre uma rede de comutação de circuitos e uma rede de comutação de pacotes sem estado. Essa versão intermédia consiste em continuar a exigir aos computadores que vão comunicar que solicitem ao núcleo da rede autorização para esse efeito, através da abertura daquilo que se convencionou chamar um “circuito virtual”.

Durante o estabelecimento do circuito virtual, o caminho que os pacotes que os dois interlocutores vão trocar é estabelecido, e os nós de comutação de pacotes que esse caminho usará receberão informação (estado) sobre o mesmo. É como se num labirinto fossem colocadas marcas que guiam os futuros transeuntes para a saída.

As vantagens da adopção dos circuitos virtuais são várias. Se os mesmos forem adoptados, o processamento dos pacotes que circulam num circuito virtual é mais simples, pois cada nó de comutação, ao reconhecer o circuito a que o pacote pertence, já sabe qual a interface pela qual ele deve ser transmitido. A dimensão dos cabeçalhos pode ser menor, pois os pacotes que circulam no circuito não precisam de ter um endereço global à rede. Os nós de comutação podem, mais facilmente, controlar se os pacotes do circuito estão a fluir correctamente; podem, por exemplo, introduzir controlo de fluxo e controlo de erros só a nível de cada circuito, de forma a darem mais segurança às extremidades do circuito sobre o seu funcionamento. E, por último, é possível realizar uma mais simples contabilização da utilização de recursos pelo circuito. Nas redes que usavam circuitos virtuais, era comum facturar pelo tempo de utilização de cada circuito e pelo tráfego transferido pelo mesmo, tal como as chamadas telefónicas.

Chegou agora a altura de ver os defeitos. Um dos primeiros defeitos que foi logo identificado estava relacionado com o que aconteceria caso houvesse uma avaria num canal, ou num nó, usado pelo circuito virtual. Para esconder este facto às extremidades do circuito, o núcleo da rede tem de ser capaz de fazer re-encaminhamento do circuito, o que complica muito os nós de comutação. Também é difícil explorar simultaneamente diversos caminhos entre as duas partes, caso existam, ou mudar dinamicamente entre os mesmos, para optimizar a utilização dos recursos da rede.

E, finalmente, existiria um defeito ainda mais grave, mas que não foi logo evidente inicialmente. Quantos circuitos virtuais seria necessário estabelecer em cada momento e quanto tempo duraria cada um? Actualmente é evidente que o número de diferentes fluxos de pacotes (entre dois sockets distintos) que atravessam a rede por unidade de tempo é da ordem de grandeza dos milhões por segundo em qualquer rede de média dimensão. Os nós de comutação, nos cruzamentos principais, teriam de ter uma capacidade gigantesca para processar o estabelecimento de novos circuitos virtuais e, numa fração significativa dos casos, estes durariam escassos segundos. Com efeito, para além da grande quantidade de computadores existentes, muitos executam aplicações que necessitam de comunicar simultaneamente com vários computadores distintos (Web, DNS, *etc.*). As comunicações actuais na rede não são como as chamadas telefónicas (cada telefone só faz uma chamada de cada vez e a mesma dura muitos segundos). Abandonar a noção de circuito virtual entre computadores foi uma boa opção para a escalabilidade da Internet.

Assim, foi estabelecido o princípio de que cada nó de comutação de pacotes só conhece informações sobre o estado da rede e não tem qualquer informação sobre os fluxos particulares que a atravessam. Quando um nó recebe um pacote, analisa-o como

se fosse a primeira vez que recebesse um pacote daquela origem e para aquele destino e, em função do estado corrente da rede, toma a opção de encaminhamento que for mais adequada no momento. Por outro lado, caso apareçam anomalias, a rede não assume a responsabilidade de mascarar as mesmas à periferia, ver o Capítulo 1. É como já vimos, a noção de “rede sem estado” e a funcionar segundo o princípio do “melhor esforço”, *i.e.*, sem garantias.

Este conjunto de opções, a saber, rede sem estado e a funcionar segundo o princípio do melhor esforço, permite uma grande flexibilidade de implementação do núcleo da rede, permite adoptar diferentes tecnologias mais facilmente, e conferiu à arquitectura da Internet a grande escalabilidade que esta conhece actualmente.

De alguma forma é uma continuação do princípio da responsabilização dos extremos que vimos atrás. Obviamente, todas as optimizações de implementação são possíveis, mas as mesmas não passam disso mesmo, optimizações que não colocam em causa o princípio da rede sem estado e de melhor esforço.

O princípio da rede sem estado e de melhor esforço (*stateless best-effort delivery network*) indica que o estado conhecido pelos nós da rede deve ser o estritamente necessário para fazer chegar cada pacote ao seu destino. Ou seja, os nós da rede não necessitam de memorizar estado sobre os fluxos de pacotes que a atravessam. Por outro lado, em caso de anomalias (*e.g.*, incapacidade de suportarem o ritmo com que os pacotes chegam, ou avarias graves), os nós não são obrigados a recuperar os pacotes em circulação, ou a avisar a periferia das falhas.

Este princípio confere uma grande flexibilidade de implementação, favorece a escalabilidade e é coerente com a responsabilização dos extremos e o seu envolvimento no correcto funcionamento da rede.

Finalmente, e para terminarmos esta discussão sobre alguns princípios das redes de computadores, vamos discutir brevemente os princípios da modularidade e independência das partes aplicados às redes.

Modularidade e independência da partes (*separation of concerns*)

Em informática existe um conjunto de princípios gerais que são fundamentais para a análise e desenho dos sistemas em geral, e dos programas em particular.

Um desses princípios fundamentais tem a ver com a necessidade de decompor um sistema em partes ou módulos e, em cada momento, ser capaz de caracterizar cada uma dessa partes, não pelos seus detalhes internos, mas pelas suas propriedades essenciais vistas do exterior, usando abstracções.

Essas propriedades e funcionalidade essenciais são consubstanciadas numa especificação e podem, depois, conduzir à definição de uma interface que permite aceder à funcionalidade do módulo. Os detalhes da sua implementação são ignorados e devem permanecer escondidos.

Esta decomposição em módulos e a especificação e autonomia dos mesmos, que se traduz na relação entre os módulos através de interfaces, permite alterar a implementação dos módulos de forma independente, sem alterar os restantes. A mesma é também essencial para a resolução de um problema complexo, decompondo-o em sub-problemas resolvidos de forma independente.

Em redes de computadores estes princípios também são usados a diversos níveis. Por exemplo, as aplicações utilizam a rede através das interfaces que dão acesso aos serviços de transporte. Os serviços de transporte usam os serviços da rede através dos serviços prestados pelo protocolo IP. O protocolo IP implementa os seus serviços recorrendo aos canais e aos nós de comutação.

No interior da rede existem diversas sub-redes. Cada uma delas tem autonomia no que diz respeito à forma como realiza o encaminhamento dos pacotes. Adicionalmente, cada uma dessas sub-redes é gerida de forma autónoma, adoptando relações bilaterais apenas com aquelas a que está ligada directamente. Existe depois um protocolo global, que permite que esses compromissos bilaterais sejam propagados a todas as redes ligadas, de forma a que as mesmas, por transitividade, conheçam as alternativas de ligação a cada destino.

A interface entre o núcleo da rede e os computadores que lhe estão ligados assume um papel fundamental, ao isolar a infra-estrutura dos canais e dos comutadores, dos computadores que lhe estão ligados. Esta decomposição e separação em sub-problemas, realizada pelo protocolo IP (caracterizado semanticamente também pela noção de qualidade de serviço “melhor esforço”) tem-se revelado fundamental para a evolução e adaptabilidade da Internet.

Com efeito, esta interface esconde os detalhes do núcleo interno da rede e tem permitido que a tecnologia dos nós de comutação e dos canais de comunicação tenha evoluído continuamente e aumentado em várias ordens de grandeza em débito e qualidade. No entanto, os computadores e os protocolos de transporte têm, no essencial, sido isolados dessas alterações, o que testemunha o êxito do princípio. Por outro lado, o transporte e as aplicações têm também sido capazes de evoluir e adaptarem-se continuamente, sem necessidade de alterações fundamentais directamente relacionadas com a evolução dos canais ou dos nós de comutação.

O carácter central do protocolo IP, e a relevância do seu papel, é muitas vezes resumido naquilo que é habitual designar como o modelo da ampulheta dos protocolos TCP/IP, ver a Figura 4.2. Esta representação põe em evidência a grande diversidade de soluções e protocolos aplicacionais e de transporte, isolados da grande diversidade dos canais e dos nós de comutação, por uma única interface, a do protocolo IP.

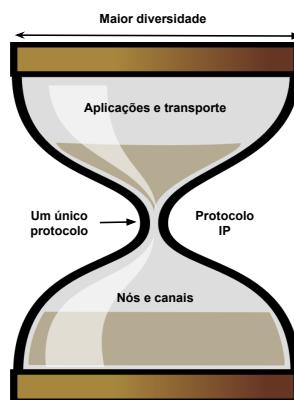


Figura 4.2: O protocolo IP é o gargalo da ampulheta; ele separa a diversidade das aplicações e transportes, da diversidade dos nós de comutação e canais

As redes de computadores incluem igualmente mecanismos de designação e associação tardia (*late binding*) que são importantes para facilitar a independência das partes e a modularidade. Estes são constituídos pela separação entre diversos mecanismos de designação, nomeadamente: **nomes** e **endereços** (*names and addresses*).

Os endereços são mecanismos de designação que associam entidades à sua localização na rede e permitem o acesso às mesmas. Nas redes actuais, os endereços das entidades podem necessitar de ser alterados porque estas se movem, ou têm novas materializações (*e.g.*, diferentes servidores).

Para além dos endereços, as entidades têm nomes. Estes são mapeados nas entidades e nos respectivos endereços, tão tarde quanto possível, pois isso aumenta a flexibilidade. Por exemplo, dependendo do contexto ou da localização, uma funcionalidade pode estar associada a servidores diferentes.

Este tipo de associações tardias, de alguma forma semelhantes às que se utilizam na programação e execução de aplicações, são muito importantes para o suporte da modularidade e a flexibilidade da organização de um sistema distribuído.

Existe apenas um aspecto que esta modularidade e separação entre módulos não acomoda adequadamente: o desempenho. Por vezes é desejável que alguns módulos tenham indicações sobre quais as opções que devem adoptar para melhorarem o seu desempenho, tirando o melhor partido possível do estado ou de características de outros módulos.

Este aspecto tem sido posto em evidência por diversos sistemas, como por exemplo os sistemas de distribuição de conteúdos e os sistemas móveis. Em diversas ocasiões, tem sido demonstrado que se as aplicações ou o transporte tivessem informações sobre o estado e a configuração da rede, poderiam optimizar o seu funcionamento.

Por exemplo, se um conteúdo está disponível em diversos servidores, qual deve o cliente escolher? A resposta é aquele para o qual a rede proporciona melhor débito extremo a extremo. No entanto, com os protocolos actuais não é possível aceder a essa informação. Os sistemas móveis podem usar diversos canais pois existem diversas redes acessíveis (*e.g.*, Wi-Fi, rede celular, *etc.*), mas qual é preferível usar para aceder a um certo serviço? A forma como os protocolos de transporte funcionam numa rede com canais sem fios poderia ser melhorada se o transporte tivesse acesso a informação sobre os erros que acontecem nos canais. Existem inúmeros exemplos destas situações e o leitor interessado pode consultar, por exemplo [Dai et al., 2010; Xie et al., 2008; Balakrishnan et al., 1995], onde são discutidas algumas delas. Em alguns dos casos analisados, chegou-se à conclusão que seria interessante aceder a informação que está escondida na implementação de outras componentes da rede. Em alternativa, seria necessário alterar as interfaces para darem acesso a essas informações, ou mesmo alterar a decomposição em partes como é proposto em [Feldmann, 2007; Day, 2008].

A filosofia e os princípios do desenho dos protocolos TCP/IP, que acabámos de rever, está apresentada no seguinte artigo de David Clark [Clark, 1988] cuja leitura recomendamos. No mesmo é também posto em evidência que no desenho dos protocolos e outras componentes da Internet foram ignoradas duas facetas: a segurança e o controlo da utilização dos recursos. Discutiremos a seguir as implicações destas opções.

4.2 Segurança e controlo de acesso e de recursos

Nas redes de circuitos é mais fácil introduzir segurança, controlo de acessos e controlo do consumo dos recursos. Com efeito, no momento da abertura do circuito, é possível solicitar alguma forma de autenticação da entidade que requer a sua abertura, esperar que a outra extremidade do circuito aceite a ligação e autenticá-la também, limitar nos nós de comutação envolvidos os recursos que o circuito pode consumir e, mais tarde, imputar os custos da sua utilização. Numa rede de comutação de pacotes sem estado, um computador pode, *a priori*, enviar pacotes para muitos destinos diferentes e estes controlos são mais difíceis de implementar.

Quando as primeiras redes de pacotes foram desenhadas, como por exemplo a Internet, o objectivo principal era conseguir realizar a interligação dos computadores. Adicionalmente, tratavam-se de projectos realizados em ambiente académico, com contratos governamentais, com o objectivo fundamental de apoiar a investigação. Os problemas de segurança e de controlo da utilização dos recursos, assim como a sua facturação, não eram objectivos importantes.

Quer por esses problemas não serem considerados fundamentais, quer porque a sua solução num quadro de rede de pacotes, sem estado, e com entrega de pacotes com base no melhor esforço ser mais difícil, os protocolos e a arquitectura da Internet não tratavam inicialmente, e continuam a não tratar de forma obrigatória, dos problemas de segurança, controlo de acessos, ou do controlo do consumo de recursos.

Na arquitectura da Internet o problema da segurança é considerado um problema dos extremos e de solução opcional. Se ambas as partes em comunicação requerem segurança e autenticação, devem implementá-las extremo a extremo, por exemplo, usando uma versão segura de canais TCP, ver o RFC 5246. Se não utilizarem mecanismos deste tipo, os pacotes podem ser copiados ou alterados por atacantes que se consigam colocar no seu caminho. Como a segurança é sempre um problema de avaliação de risco, compete aos interlocutores, nos extremos da rede, adoptarem o nível de segurança que achem desejável. Por omissão, na Internet nada impede que os pacotes sejam copiados ou alterados por atacantes suficientemente poderosos e interessados, como ilustra a Figura 4.3.

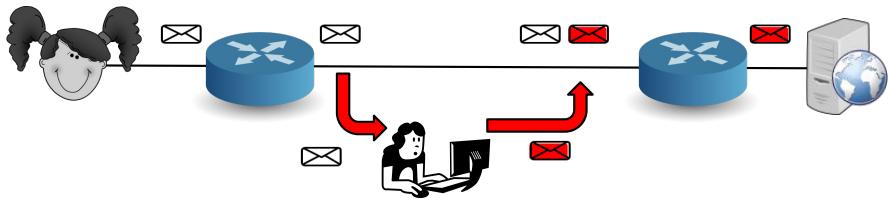


Figura 4.3: Por omissão, na Internet nada impede que os pacotes sejam copiados ou alterados por atacantes suficientemente poderosos e interessados

Sem mecanismos extremo-a-extremo de autenticação e segurança (*end-to-end authentication and security*) dos fluxos de pacotes, por omissão, a Internet não impede que os pacotes sejam copiados ou alterados. Compete aos utilizadores finais avaliarem os riscos envolvidos em cada contexto e adoptarem os mecanismos de autenticação e segurança extremo a extremo mais adequados em cada caso.

O problema da contabilização da utilização de recursos foi deixado para ser resolvido pelas diferentes sub-redes, quer através de contratos com os seus clientes finais (nas redes de acesso), quer através de mecanismos de compensação entre as diferentes sub-redes interligadas (e.g., com as redes de trânsito ou outras). No entanto, sem um mecanismo de autenticação global dos computadores e das diferentes sub-redes, este problema tem uma solução geral mais difícil.

Ora o problema da autenticação acabou por ser quase completamente ignorado à partida pois, *a priori*, caso a sub-rede de acesso a que está ligado não o controle, nada impede um computador de enviar pacotes com endereços origem falsos. O pacote será, mesmo assim, provavelmente encaminhado até ao destino. Para além disso, na arquitectura interna da Internet não está previsto nenhum mecanismo de autenticação entre sub-redes que não estejam directamente ligadas.

É possível introduzir nas redes mecanismos de autenticação e de controlo da utilização de endereços origem legais. Mas, na arquitectura da Internet, baseada em controlo distribuído e autonomia das sub-redes, esses mecanismos não estão previstos

por omissão e, na verdade, não existe actualmente um modelo económico que incentive a sua utilização, pelo que, no essencial, não são implementados.

Infelizmente, esta ausência de autenticação e controlo é a razão principal que torna mais fáceis os ataques designados por **ataques de negação de serviço**.

Um **ataque de negação de serviço** (*denial of service attack*) consiste em encontrar formas de provocar a impossibilidade de a vítima do ataque ser capaz de usar ou fornecer serviços através da rede. Esta impossibilidade deve-se à exaustão dos recursos da vítima (*e.g.*, capacidade da ligação à rede, CPU, memória ou espaço em disco). Em geral, o ataque terá tanto mais êxito quanto menos recursos exigir ao atacante e mais recursos consumir à vitima.

A maioria dos ataques de negação de serviço pretende atacar infra-estruturas críticas, como por exemplo o DNS, serviços de grande visibilidade, ou serviços com elevado valor económico. As motivações são várias, como por exemplo vandalismo, crime económico ou prejudicar um concorrente. Para ilustrar o problema, descrevemos a seguir uma forma de ataque particularmente eficaz, designado ataque distribuído por reflexão, ver a Figura 4.4.

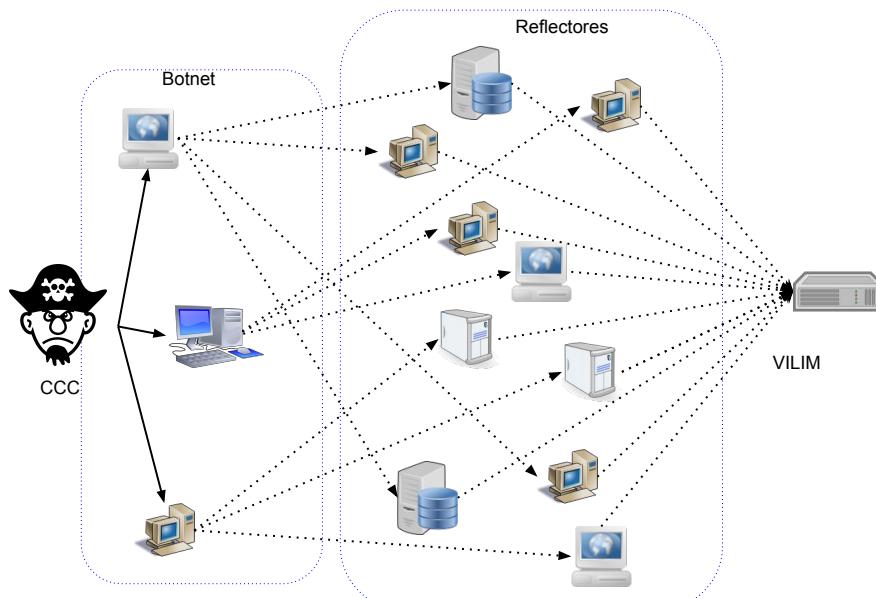


Figura 4.4: Ataque de negação de serviço distribuído por reflexão: CCC, através da sua *botnet*, ataca o computador VILIM

Uma organização criminosa, chamemos-lhe CCC (*Ciber Criminosos Competentes*), vende um serviço para realizar ataques de negação de serviço que impedem os computadores atacados de fornecer serviços. A organização CCC dispõe de uma *botnet* própria, *i.e.*, um conjunto de computadores de terceiros que controla. A organização CCC está sempre a renovar os membros da sua *botnet*, procurando computadores ligados à Internet com vulnerabilidades de segurança, e nos quais instala um programa especial, controlado por ela remotamente, geralmente designado por *Cavalo de Tróia*.

Quando decide atacar um servidor, chamemos-lhe VILIM (*Vítimas ILIMITadas*), CCC usa a consola de controlo da sua *botnet*, e ordena aos seus membros que enviem pacotes ICMP (os pacotes usados pelo programa *ping*) para conjuntos aleatórios de computadores, designados reflectores. Todos esses pacotes ICMP têm como endereço origem o endereço VILIM. Resultado, VILIM é bombardeado com um volume gigantesco de respostas a pacotes de sonda que nunca enviou e o seu canal de ligação à rede fica saturado.

A utilização de uma quantidade muito elevada de computadores atacantes, endereços origem falsos e muitos reflectores, seleccionados aleatoriamente, dificultam a contenção e detecção da origem do ataque. Se, pelo contrário, o atacante usasse um só computador, todos os pacotes, mesmo com endereços origem falsos, viriam de um único ponto, o que poderia facilitar a identificação do atacante e a sua contenção. Por outro lado, como não há contabilização de recursos, é mais difícil aos responsáveis pelos computadores da *botnet* aperceberem-se da utilização anormal que deles é feita.

Os ataques de negação de serviço são comuns. No artigo [Peng et al., 2007] são apresentadas as principais técnicas usadas para o seu combate. De qualquer forma, o leitor não deverá deixar de analisar este problema sob vários ângulos. A ausência de controlo de acessos e de contabilização tem as suas virtudes. Em particular, ela incentivou um modelo de contabilização simples, baseado principalmente na capacidade dos canais que ligam os clientes à rede¹. Por outro lado, esta ausência de controlo não requer nenhum mecanismo normalizado de autenticação, e não impõe procedimentos complexos de compensação entre operadores.

Globalmente, ela torna a operação das redes muito menos onerosa e não impõe barreiras, nem à entrada de novos operadores, nem à cedência do serviço (*e.g.*, uma universidade pode ceder acesso aos seus alunos, um café pode dar acesso aos seus clientes, ...). Um modelo global de autenticação e compensação entre redes, pela sua utilização por clientes de outras, como é usado actualmente pelas redes celulares de operadores móveis, tornaria a rede menos flexível e imporia mais barreiras à entrada de novas redes no conjunto. Provavelmente a realidade da Internet seria bastante diferente da actual, pelo menos do ponto de vista dos custos de acesso e da flexibilidade da sua expansão.

Um outro aspecto ligado à forma como os operadores gerem as suas redes e as relações com os seus clientes é referido brevemente a seguir.

4.3 Neutralidade da rede

Uma outra faceta que foi também, no essencial, ignorada na fase inicial do desenvolvimento da Internet e dos protocolos TCP/IP, e que não encontrou reflexo nos princípios que presidiram aos seu desenvolvimento, foi a problemática da qualidade de serviço prestada aos diferentes fluxos de pacotes que atravessam a rede.

Nos primeiros tempos da Internet as aplicações dominantes consistiam na transferência de ficheiros e no acesso a computadores remotos, as quais são aplicações elásticas, ver a Secção 3.6. As aplicações com requisitos especiais de qualidade de serviço (telefone, som e imagem) usavam redes especiais dedicadas (*e.g.*, redes telefónicas e de televisão). Requeria-se apenas, então, que os diferentes fluxos fossem tratados de forma equitativa, o que é razoavelmente compatível com um modelo de facturação simples: cada cliente paga um preço de acesso proporcional à capacidade da sua ligação à rede, e portanto o preço pago era proporcional ao ritmo máximo com que o cliente

¹É preciso ter em atenção que os esquemas de contabilização detalhada, por exemplo de chamadas telefónicas, representam por si só uma fracção importante dos custos de operação da rede e implicam a necessidade de regulação para evitar práticas anti-competitivas.

poderia transferir dados. Contabilizar a quantidade de dados transferidos, por exemplo, é mais complexo e dispendioso e, em muitos cenários, revelou-se comercialmente contraproducente.

À medida que as capacidades de ligação às redes de acesso foram crescendo (e a capacidade dos *backbones* também), surgiram as aplicações de troca intensiva de dados, como por exemplo as aplicações P2P, e a heterogeneidade entre as solicitações feitas à rede pelos diferentes clientes cresceu muito. Nesse quadro, os utilizadores intensivos, em redes sub-dimensionadas para os mesmos, prejudicavam a qualidade de serviço dos utilizadores menos intensivos, mas ambas as categorias de utilizadores pagavam o mesmo. A solução adoptada consistiu, frequentemente, em limitar artificialmente, e às vezes até extra-contratualmente, as transferências dos utilizadores intensivos.

Com a generalização da utilização das redes de pacotes em geral, e da Internet em particular, para suporte de aplicações com necessidades de qualidade de serviço especiais (*e.g.*, chamadas telefónicas e aplicações multimédia em geral), foi preciso começar a introduzir mecanismos de diferenciação dos diferentes fluxos de tráfego, de forma a garantir que essas novas aplicações das redes de pacotes pudessem funcionar adequadamente. Dado que as aplicações elásticas toleram melhor eventuais deficiências da qualidade de serviço, é aceitável que estas sejam tratadas com menor prioridade dentro da rede.

Neste novo quadro, e tendo em consideração que a indústria clássica de telecomunicações absorveu a indústria dos operadores de acesso à Internet, os chamados ISPs (*Internet Service Providers*), dentro da mesma rede coexistem serviços com requisitos de qualidade de serviço especiais de dois tipos: os fornecidos directamente pelos operadores da rede de acesso aos seus clientes finais (*e.g.*, telefone, canais TV, filmes), e os serviços prestados por operadores de novo tipo, chamados operadores de conteúdos, que aproveitam o carácter extremo a extremo da rede, para fornecerem serviços do tipo multimédia (*e.g.*, chamadas telefónicas, acesso a filmes, séries e aplicações multimédia em geral), em competição com os fornecidos pelos operadores das redes de acesso.

Quando estes serviços são prestados pelos operadores de acesso, os fluxos de pacotes que os suportam transitam entre os clientes finais e servidores dedicados do operador de acesso. Quando estes serviços são prestados por operadores externos, os fluxos de pacotes que os suportam transitam, como é comum, entre os clientes finais e servidores ligados a outras redes, muitas vezes atravessando redes de trânsito. Por razões de competição comercial, os operadores de acesso estão interessados em introduzir uma nova forma de discriminação dos fluxos de tráfego, que dê prioridade aos fluxos que terminam nos seus servidores, ou aos fluxos de pacotes que terminam em servidores de operadores de conteúdos que aceitem pagar-lhes (aos operadores de acesso) esse tratamento diferenciado.

A neutralidade da rede (*network neutrality*) é um quadro de gestão da qualidade de serviço dentro da rede, em que a diferenciação do serviço prestado a diferentes fluxos de pacotes não toma em consideração critérios de competição comercial entre serviços com requisitos de qualidade de serviço semelhantes.

As formas de discriminação de fluxos acima referidas estão relacionadas com práticas de concorrência comercial entre diferentes tipos de operadores. O termo **neutralidade da rede** foi introduzido para caracterizar um quadro de gestão da rede em que a diferenciação de qualidade de serviço entre fluxos não é aplicada a fluxos com o mesmo tipo de necessidades de qualidade de serviço, mas com origem ou destino em computadores pertencentes a entidades com diferentes relações comerciais com os operadores da rede².

² A neutralidade da rede não impede a discriminação entre clientes finais que contratam diferentes classes de serviço.

A arquitectura e os princípios de funcionamento das redes de pacotes, e da Internet em geral, nada estabelecem sobre a problemática da qualidade de serviço, a contabilização da utilização dos recursos e a autenticação dos utilizadores. Como já referimos, uma rede pura de pacotes, não orientada à conexão, não fornece um quadro simples e conveniente para a solução destes problemas. Ao contrário, as redes de circuitos facilitam a solução destas questões. No entanto, e como disse o cómico M. L. Mencken, “Todos os problemas complexos têm uma solução clara e simples, mas errada”.

A facturação e o controlo da utilização da rede, realizada numa rede de circuitos, circuito a circuito, conduziria a uma rede pouco escalável, rígida, e provavelmente sem capacidade de evoluir e suportar o nível de inovação que conhecemos na Internet. Estas limitações à inovação teriam conduzido certamente a um ecossistema Internet muito diferente do que conhecemos.

Depois da análise dos princípios que presidem à concepção e funcionamento das redes TCP/IP, vamos a seguir analisar alguns dos modelos que permitem analisar e desenhar as redes de computadores.

4.4 Modelos

Entre os modelos mais relevantes em redes de computadores, começamos por discutir os modelos topológicos baseados em grafos.

Modelos topológicos baseados em grafos

A infra-estrutura de rede constituída pelos canais e nós de comutação pode ser modelizada através de um grafo. Um grafo, $G = (V, E)$, é um objecto matemático constituído por um conjunto V de vértices e um conjunto E de arcos (*edges*). Alguns dos conceitos fundamentais da teoria de grafos [Gross and Yellen, 2005] são particularmente relevantes neste contexto.

Os arcos são denotados por um par de vértices (*e.g.*, (v_1, v_2)) e dizem-se *não orientados* (onde (v_1, v_2) e (v_2, v_1) denotam o mesmo arco) quando modelizam um canal *full-duplex*, ou *orientados*, no caso contrário, e neste caso modelizam um canal *simplex*. Um caminho é uma sequência não vazia de arcos e modeliza a sequência de nós e canais que ligam os nós origem e destino do caminho. Um *lacete* é um arco com início e destino no mesmo vértice.

Um grafo diz-se *conexo* quando existe pelo menos um caminho entre dois quaisquer nós, e *particionado* no caso contrário. O número de arcos que terminam num nó diz-se o seu *grau*.

Um grafo diz-se *pesado* quando existe um função *peso* que associa cada arco a um valor real, sendo o peso de cada arco positivo. O *custo* de um caminho p é a soma dos pesos dos arcos de p , e o *comprimento* de p é o número de arcos de p .

A modelização da uma rede através de um grafo faz corresponder cada nó de comutação de pacotes a um vértice, e cada canal a um arco. Por isso, a seguir usaremos os pares de termos rede e grafo, vértice e nó, e arco e canal, como sinónimos. A existência de pelo menos um caminho, com custo finito, entre os nós v_1 e v_2 , é condição necessária e suficiente para que exista conectividade entre v_1 e v_2 .

Vários aspectos são particularmente relevantes na análise das redes através de grafos. A existência de caminhos disjuntos³ entre nós garante conectividade mesmo

³ Um par de caminhos diz-se disjunto quando estes não têm nenhum arco, nem nenhum nó (excepto a origem e o destino) em comum.

quando algum canal avaria e fica inoperacional. Uma rede bem desenhada deve minimizar o impacto dessas avarias, garantindo a manutenção da conectividade mesmo em presença das falhas previsíveis (*i.e.*, as que fazem parte do modelo de falhas da rede). A existência de caminhos redundantes pode também proporcionar uma melhor distribuição de carga. No entanto, esta distribuição só proporciona capacidade máxima de extremo a extremo se o grau de partilha de cada canal pelos diferentes caminhos, usados pelos nós, for bem distribuído.

O grau dos nós é muito relevante, pois o custo monetário dos nós de comutação de pacotes cresce com o seu número de interfaces (o seu grau). Este número máximo é limitado, sobretudo em equipamentos de muito alto débito. O custo monetário da rede é constituído pelo custo monetário dos nós, adicionado ao custo dos canais. Estes últimos, quando instalados fora de edifícios, têm um custo monetário grosso modo proporcional à distância e, em menor grau, ao seu débito.

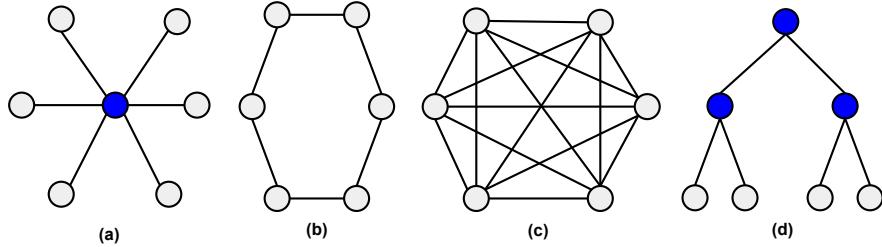


Figura 4.5: Redes (a) estrela (*hub-spoke*), (b) anel (*ring*), (c) malha completa (*full-mesh*), (d) hierárquica (*hierarchical*)

A seguir discutem-se alguns grafos bem conhecidos cujas propriedades são interessantes. Comecemos pelos da Figura 4.5.

Uma rede em estrela (4.5a) tem um nó central (o *hub*) que assegura a conectividade entre todos os outros, que são todos de grau 1 (sistemas finais). A avaria do nó central, cujo grau é necessariamente elevado, impede a rede de fornecer qualquer serviço, constituindo este nó um ponto central de falha. Este tipo de configuração é muito usada em redes no interior de edifícios com nós centrais robustos e bem protegidos, e canais de curta distância, portanto pouco onerosos. Redes em estrela também são frequentemente usadas em redes de interligação de vários edifícios (*e.g.*, redes de *campus*) mas nesse caso é frequente robustecer o nó central e diminuir o seu grau através de configurações como a ilustrada na Figura 4.6 (e).

Uma rede em anel (4.5b) só tem nós de grau 2 e mantém conectividade completa mesmo que um canal se avarie. É usada em *backbones* em que se pretende minimizar o custo dos canais sem deixar de garantir algum grau de redundância. Este tipo de rede é frequentemente usada como *backbone* de redes de acesso em áreas metropolitanas, com configurações como a ilustrada na Figura 4.6 (f).

Uma rede *full-mesh* (4.5c) de n nós sem lacetos assegura que existem $n - 1$ caminhos entre quaisquer dois nós: o caminho directo e $n - 2$ outros caminhos de comprimento 2, cada um dos quais passando por um nó intermédio. São redes caras em termos de grau dos nós e do número de canais, mas são especialmente adaptadas à distribuição de carga e resistem a várias avarias dos canais. Estas redes são usadas para a interligação de agregados de computadores (*computer clusters*) quando o critério custo é secundário face à necessidade de maximizar a capacidade de interligação.

Uma rede hierárquica (4.5 d) garante a conectividade entre os nós folha da árvore (os nós de grau 1) através de nós internos à árvore. Aumentando o número de níveis internos, este tipo de rede permite interligar muitos nós folha minimizando o grau dos

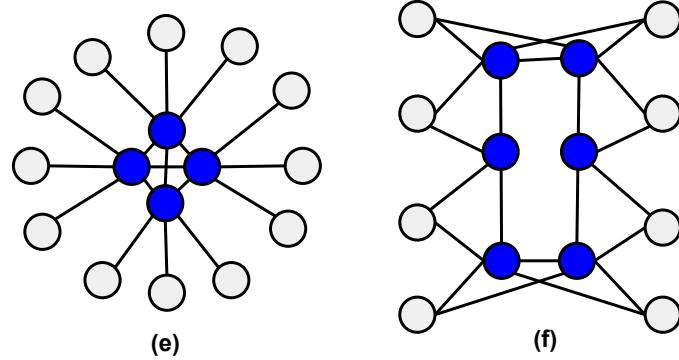


Figura 4.6: Redes (e) estrela com centro reforçado (*hub-spoke*), (f) centro em anel com ligações redundantes

nós de interconexão. Como contrapartidas, apresenta uma grande heterogeneidade de comprimento dos diferentes caminhos, e resulta frágil perante avarias de canais ou nós dos níveis mais elevados. São redes populares em centros de dados de grande dimensão, nos quais são usadas configurações alternativas, como a ilustrada na Figura 4.7 (g), que permitem assegurar alguma redundância e resistência a avarias, assim como caminhos alternativos. No entanto, muitos desses caminhos, sobretudo os que atravessam os níveis superiores da hierarquia, partilham os mesmos canais, pelo que a distribuição de carga é deficiente. Por outro lado, para garantir que quaisquer dois nós podem comunicar usando a capacidade máxima que os liga à rede, os nós do topo da hierarquia têm de ter elevada capacidade, o que os torna mais caros.

Quando a distribuição de carga e a maximização da capacidade de transferência simultânea entre todos os nós são objectivos prioritários, podem ser usadas redes designadas por redes *Clos*, do nome do seu inventor, ver a Figura 4.7 (h). Uma destas redes, com n nós em cada nível, é uma rede que assegura que existem sempre n caminhos distintos entre quaisquer dois nós do nível de baixo, o que permite uma redundância e distribuição de carga óptimas, quando só os nós desse nível são origem ou destino de pacotes.

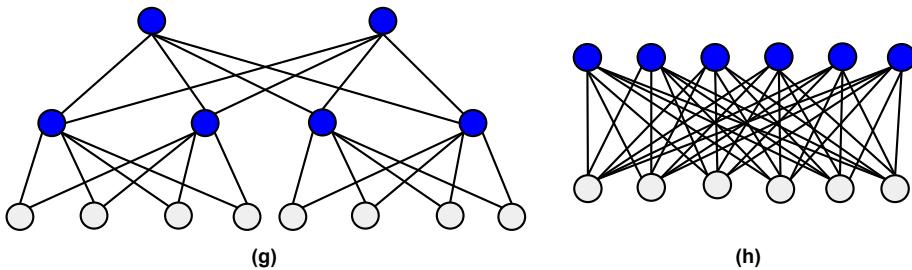


Figura 4.7: Redes (g) Hierárquica duplicada, também conhecida por *Fat tree*, e (h) *Clos network*

As redes até aqui apresentadas são usadas em situações em que a geografia ou a concentração do tráfego não são muito relevantes. As redes dos operadores que

cobrem grandes áreas geográficas, conhecidas por WAN (*Wide Area Networks*), por oposição às LAN (*Local Area Networks*) e MAN (*Metropolitan Area Networks*), são redes configuradas em função das concentrações geográficas do tráfego, e de forma a minimizarem o custo dos canais, dadas as distâncias geográficas envolvidas.

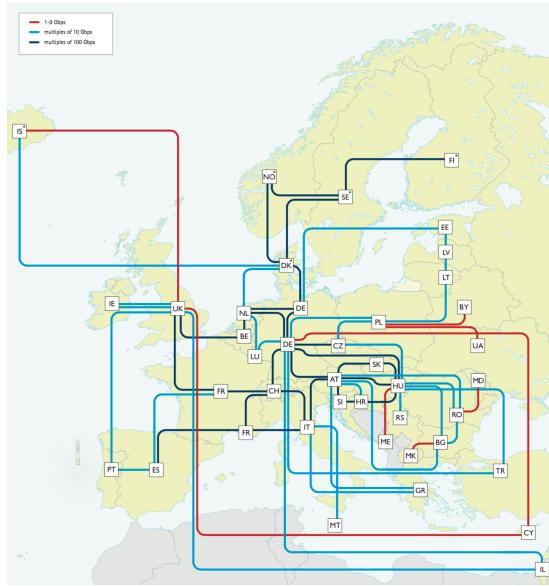


Figura 4.8: Grafo correspondente à rede Géant em 2015, um *backbone* de interligação das redes de investigação europeias. A figura foi extraída da figura semelhante disponível em <http://www.geant.net>.

Nos modelos deste tipo de redes, os nós simples de comutação de pacotes são substituídos pela noção de POP (*Point of Presence*). Um POP é um agregado de comutadores, concentrado num mesmo edifício, que faz a interface entre a WAN e as redes de acesso regionais. Trata-se de uma situação em que para manter o modelo manejável, se aumenta o nível de abstração usado. A Figura 4.8 apresenta um exemplo típico de uma rede WAN de interligação de POPs em diferentes países, neste caso é o *backbone* Géant de interligação de um conjunto de redes de investigação europeias. O leitor interessado em representações gráficas dos grafos que modelizam diversas redes de POPs de operadores poderá consultar [Knight et al., 2011] ou visitar o *site* indicado na secção 4.6.

Visões da Internet como um grafo

Existem vários biliões de computadores ligados à Internet e esta é formada internamente por milhões de comutadores de pacotes. Mesmo que a informação necessária estivesse disponível num só ponto, a representação desta rede como um grafo seria uma tarefa ciclopica.

Como os endereços IP estão organizados hierarquicamente, o número de prefixos IP distintos visíveis na zona de interligação das diferentes sub-redes que formam a Internet, dá uma ideia da complexidade organizacional da rede, ou pelo menos permite apreciar a evolução da mesma.

Esta zona de interligação chama-se a DFZ (*Default-Free Zone*) porque os comutadores da mesma têm de conhecer todos os prefixos existentes, não podendo usar outros comutadores como opção de encaminhamento por omissão, na esperança de que os mesmos tenham uma visão mais completa da rede. A evolução do número desses prefixos

ao longo dos anos é monitorada, ver por exemplo o site <http://bgp.potaroo.net/>, que mostra que durante o ano de 2016 se ultrapassaram os 600,000 prefixos IP distintos na DFZ.

Não sendo possível representar a Internet num grafo, e atendendo a que tal representação em detalhe é pouco útil, tenta-se caracterizar indirectamente o seu grafo por propriedades que permitam deduzir alguma das suas propriedades globais, *e.g.*, qual a distribuição do comprimento dos caminhos existentes, quais os nós que são críticos para o funcionamento da rede, *etc.*

Dadas as dimensões envolvidas, é necessário elevar ainda mais o grau de abstração. Para o estudo da estrutura da Internet usam-se modelos de grafos em que um nó representa um sistema autónomo (AS – *Autonomous System*), os quais correspondem, grosso modo, às sub-redes que formam a Internet. O nome sistema autónomo foi escolhido para pôr em evidência que a política de gestão de cada uma dessas sub-redes é autónoma.

A investigação da estrutura do grafo dos ASs permitiu algumas conclusões interessantes sobre a forma como estes se encontram interligados. O grau de um AS indica o número de ligações que este tem a outros ASs. Um caminho entre dois ASs é um caminho formado por ASs, incluindo a origem e o destino, e os ASs intermédios atravessados pelo caminho. Os caminhos entre redes de acesso são ASs intermédios que são geralmente redes de trânsito, ver a Secção 1.3.

A distribuição do grau dos ASs mostra que, das várias dezenas de milhar de ASs existentes, a grande maioria tem grau 1. Esta maioria de AS representam redes institucionais ligadas na periferia da Internet ligadas a um único operador. No meio existem muitos sistemas autónomos com um grau significativo, que correspondem às redes de trânsito de segundo nível. No outro extremo, existe menos de uma centena de ASs cujo grau é da ordem de grandeza de milhares. Esses ASs correspondem a redes de trânsito *Tier 1*, ver a Secção 1.3, e a alguns operadores de redes conteúdos gigantes, cujos nomes são bem conhecidos do grande público. Quer as redes *Tier 1*, quer as redes de conteúdos equiparadas em grau, ligam a uma fração muito significativa dos outros ASs. Por outro lado, os ASs *Tier 1* estão completamente interligados uns com os outros.

Daqui resulta que os caminhos entre quaisquer dois ASs não são muito longos. Com efeito, cada um dos ASs origem ou destino não estão muito longe de um AS *Tier 1*, e como estes estão interligados entre si, verifica-se que o caminho médio entre quaisquer dois ASs é relativamente curto. Na verdade, esse comprimento médio é da ordem de grandeza de 4. Por outro lado, quase todos os ASs estão ligados aos ASs dos grandes operadores de conteúdos por caminhos muito curtos, geralmente com comprimento médio da ordem de grandeza de 2.

A interligação entre os ASs que formam a Internet está em constante evolução devido à evolução comercial da mesma. Por isso, a sua estrutura não é desenhada a régua e esquadro, mas é comandada por forças e relações económicas. No entanto, a sua estrutura é vagamente hierarquizada, como as redes da Figura 4.7 (h), mas com elevado grau de redundância nos níveis superiores da hierarquia, a zona das redes de trânsito *Tier 1* e dos operadores das redes de conteúdos, ver a Figura 4.9.

Os modelos das redes baseados em grafos são particularmente úteis para perceber a sua estrutura, decidir sobre formas de realizar o encaminhamento dos pacotes, optimizar a rede, estudar a sua caracterização global em termos de relações entre nós, modelizar o comportamento dos protocolos de encaminhamento, *etc.*

A seguir vamos estudar outro tipo de modelos também bastante comuns, os modelos de camadas.

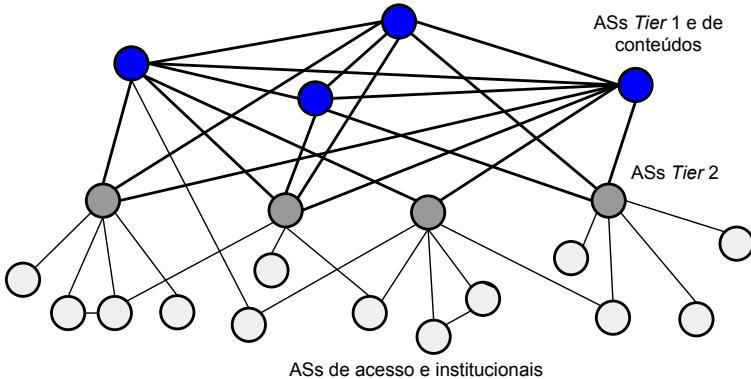


Figura 4.9: A estrutura vagamente hierárquica da Internet

Modelos baseados em camadas

Um outro tipo de modelo, sem carácter topológico e frequentemente utilizado em redes de computadores, é o chamado modelo de camadas ou níveis, que organiza as redes em camadas horizontalmente sobrepostas. A ideia destes modelos é porem em evidência os sub-problemas que são resolvidos pelos diferentes níveis de abstracção.

Trata-se de uma espécie de divisão da rede em módulos, correspondendo cada camada a um conjunto de módulos semelhantes, do mesmo nível de abstração. Tal como os modelos baseados em módulos, cada camada utiliza os serviços de outra, via as interfaces que os módulos dessa camada exportam. A diferença para outros modelos baseados em módulos, consiste em que os modelos das camadas em redes são estritamente hierárquicos. Uma camada usa exclusivamente os serviços providenciados pelos módulos da camada logo abaixo, e assim sucessivamente.

Um modelo de camadas ou níveis é assim caracterizado:

- Cada nível introduz uma abstração bem definida, isolada e diferente da dos restantes níveis. Cada camada introduz um conjunto bem definido de serviços, acessíveis pela interface do nível.
- As entidades de um nível só comunicam com entidades do mesmo nível.
- Para realizarem os seus protocolos, com excepção do nível mais baixo, as entidades de um nível usam os serviços oferecidos pela interface do nível imediatamente abaixo.
- O conjunto dos níveis forma uma hierarquia estrita organizada em pilha.

O primeiro modelo de camadas que foi formalizado para redes de computadores foi o chamado modelo OSI (*Open Systems Interconnection Model*) [Zimmermann, 1980] da ISO (*International Standards Organization*)⁴. Este modelo organiza uma rede num conjunto de 7 níveis de abstração (*abstraction layers*) sendo cada nível constituído por um conjunto de entidades do mesmo nível, que comunicam e colaboram, usando os serviços da interface da camada imediatamente abaixo.

⁴A ISO é uma organização internacional de normalização, formada por organismos nacionais oficiais de normalização, um por cada um dos 163 países constituintes. Trata-se de uma organização para a normalização, que intervém em todos os domínios de actividade empresarial.

O modelo usado nas redes TCP/IP é semelhante mas mais simples, e só possui 4 camadas, que são apresentadas na Figura 4.10 e na tabela 4.1



Figura 4.10: Pilha dos protocolos TCP/IP

Tabela 4.1: Modelo de camadas dos protocolos TCP/IP

Nível	Caracterização
Aplicação	Inclui as entidades aplicacionais (<i>e.g.</i> , clientes, servidores, ...) que utilizam a rede para implementar as aplicações e os serviços distribuídos. É um nível extremo a extremo
Transporte	Permite que as entidades aplicacionais comuniquem directamente uma com as outras. É também um nível extremo a extremo
Rede	Assegura o transporte e encaminhamento de pacotes entre computadores com recurso aos comutadores de pacotes
Canal	Assegura a ligação directa entre dois ou mais computadores ou comutadores para permitir que os mesmos troquem mensagens entre si

As camadas são de alguma forma já nossas conhecidas, e correspondem aos conceitos ilustrados que a seguir são passados em revista.

Nível aplicação O nível aplicação contém entidades que fornecem serviços directamente aos utilizadores finais, ou a outros sistemas aplicacionais. Exemplos de entidades do nível aplicacional já nossas conhecidas são, por exemplo, os clientes e servidores HTTP, os clientes e servidores DNS, *etc.* As entidades do nível aplicacional colaboram e coordenam-se entre si para providenciar o serviço correspondente. Elas usam protocolos do nível aplicacional, como por exemplo os protocolos HTTP e DNS, completamente contidos dentro desta camada. Estes protocolos são implementados usando os serviços providenciados pelo nível abaixo, o nível de transporte.

Nível transporte O nível de transporte é responsável por fornecer e implementar os serviços que permitem às entidades do nível aplicacional comunicar. Tanto o nível de transporte como o nível aplicacional são níveis de extremo a extremo, pois

as suas entidades residem exclusivamente nos sistemas finais que estão ligados à rede, ou seja nos computadores. As entidades do nível de transporte comunicam e coordenam-se entre si para implementar os serviços de transporte. Para este efeito, usam protocolos de transporte, como por exemplo os protocolos UDP e TCP. As interfaces do nível de transporte são materializadas por interfaces nos sistemas de operação, como por exemplo a interface de sockets, ver a Secção 1.6. Para a implementação dos serviços de transporte são usados os serviços prestados pela camada de rede.

Nível rede O nível rede é a primeira camada que não é de extremo a extremo. As entidades deste nível residem quer nos computadores ligados à rede, quer em todos os comutadores de pacotes da rede. Como é fácil de inferir, esta camada assegura o serviço de encaminhamento de pacotes desde a origem até ao destino. O protocolo IP é o exemplo mais conhecido deste nível. Mas como veremos mais tarde, o nível tem de resolver toda a problemática do encaminhamento de pacotes e da gestão da qualidade de serviço, para o que recorre a numerosos outros protocolos que serão introduzidos a seu tempo.

Nível canal O nível canal é uma abstração que incorpora a noção de canal e permite a quaisquer entidades físicas da rede (computadores ou comutadores de pacotes) directamente ligados entre si comunicarem através da troca directa de *frames*. O nível canal abrange todos os problemas e protocolos cuja descrição já foi introduzida no capítulo 2. Este nível assume formas específicas correspondentes a cada tipo de canal e é localizado e não global.

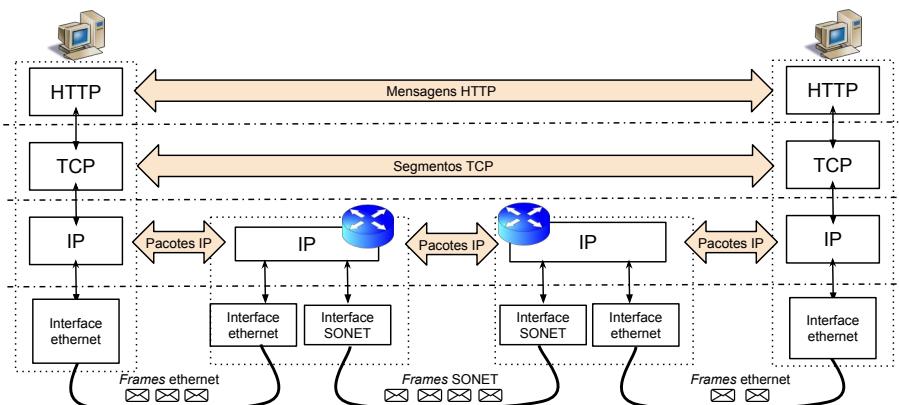


Figura 4.11: O modelo de camadas e a sua concretização em módulos nos computadores e comutadores

A Figura 4.11 apresenta um exemplo da concretização do modelo num cenário em que entidades aplicacionais, residentes em sistemas finais, comunicam directamente entre si usando o protocolo HTTP, para permitir a um utilizador aceder a uma página Web.

O *browser* HTTP e o servidor HTTP comunicam através de um canal TCP, providenciado pelo nível de transporte. O canal é implementado no sistema de operação dos dois sistemas finais, nos quais os módulos TCP comunicam e coordenam-se entre si, trocando segmentos TCP, que viajam encapsulados em pacotes IP. Estes pacotes são transportados entre os sistemas finais com recurso à interface e aos serviços do nível rede.

No exemplo é possível ver que o nível rede corresponde a um conjunto de serviços, residentes em todos comutadores de pacotes IP e nos dois sistemas finais, que asseguram que os pacotes IP são encaminhados desde a origem até ao destino. Para transitarem entre comutadores IP, dos comutadores IP até aos sistemas finais, e vice versa, os pacotes transitam por canais. Em cada canal pode ser usado um protocolo diferente. No exemplo, entre os sistemas finais e os comutadores são usados canais Ethernet sobre cabos UTP. Entre comutadores de pacotes são usados canais SONET sobre fibra óptica.

O exemplo concretiza o modelo num cenário em que se está pôr em evidência um só protocolo em cada um dos níveis aplicação, transporte e rede. A pilha de protocolos TCP/IP é mais diversa e, como já referimos, existem vários, na verdade inúmeros, protocolos do nível aplicacional, e vários protocolos do nível transporte, como ilustrado na Figura 4.12. Destes, já referimos dois dos mais conhecidos (TCP e UDP), mas existem outros, ver o Capítulo 10. O nível rede comporta um único protocolo, global a toda a Internet, que garante a inter-operação entre os computadores, os comutadores e todas as sub-redes e, como referimos na secção 4.1, constitui um ponto de estreitamento do grau de diversidade existente. Já no que diz respeito ao nível canal, existem tantos protocolos desse nível quantos tipos de canais diferentes estão disponíveis. Apesar de toda a diversidade existente, dado o papel central do protocolo IP, e a popularidade do protocolo TCP, e também a forma como historicamente estes protocolos foram desenvolvidos, esta pilha de protocolos designa-se por pilha TCP/IP.

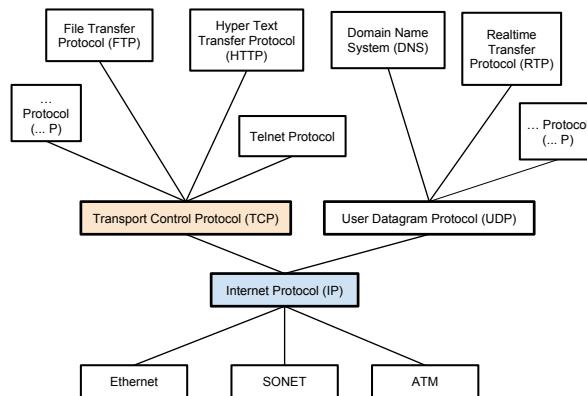


Figura 4.12: Exemplos de protocolos dos diferentes níveis da pilha de protocolos TCP/IP

A discussão sobre o modelo de camadas permite tornar agora mais claros diversos aspectos já parcialmente abordados anteriormente. O primeiro tem a ver com o facto de que todos os protocolos estruturam as mensagens que trocam numa parte de cabeçalho e outra de dados, chamada *payload*. No cabeçalho são colocadas as informações de controlo correspondentes ao protocolo do nível. No *payload* das mensagens de um nível são colocadas as mensagens do nível superior, ver a Figura 4.13. Por exemplo, no *payload* de um pacote IP é encapsulado um segmento TCP. No *payload* de um segmento TCP é encapsulada parte, ou a totalidade, de uma mensagem HTTP.

Como as mensagens do nível superior também são constituídas por uma parte de cabeçalho e outra de dados, as mensagens dos níveis mais abaixo contêm uma sucessão de cabeçalhos. Com efeito, qualquer mensagem, de qualquer nível, contém sempre a sucessão de cabeçalhos dos níveis superiores. Por exemplo, no nível canal, a seguir ao cabeçalho correspondente ao nível, encontra-se o cabeçalho do pacote IP, ou seja, o

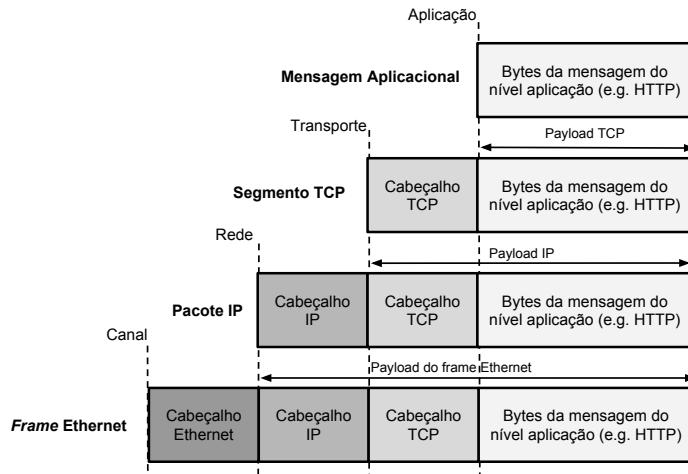


Figura 4.13: Cabeçalhos e encapsulamento nos diferentes níveis

cabeçalho do nível rede, a seguir vem o cabeçalho do nível transporte, e a seguir vem parte do cabeçalho ou dos dados da mensagem HTTP.

A noção de encapsulamento é muito rica e pode ser usada de forma recursiva. Em certas situações é desejável construir redes em que os canais são lógicos, ao invés de corresponderem a dispositivos físicos. Por exemplo, é possível colocar pacotes IP como dados de um pacote UDP, ou mesmo de outro pacote IP. Nesses casos, diz-se que estamos a encapsular IP sobre UDP, ou IP sobre IP, respectivamente. Na verdade, o canal lógico é implementado como um túnel, por exemplo entre um computador numa rede e um computador situado noutra rede, que está a actuar como comutador de pacotes. Este tipo de funcionamento diz-se uma rede lógica ou sobreposta (*overlay network*) e permite aos dois computadores levarem os pacotes trocados a seguirem um caminho escolhido por eles, e não o escolhido pela rede.

Finalmente, é agora também possível clarificar um aspecto relacionado com a terminologia sobre as mensagens dos diferentes níveis. Com efeito, o modelo de camadas especifica que as entidades de um nível só comunicam com entidades do mesmo nível, trocando mensagens do protocolo correspondente. Seria possível inventar uma forma de designar essas mensagens compatível com o modelo, e passar a usar a terminologia: mensagens do nível aplicação, mensagens do nível transporte, mensagens do nível rede e mensagens do nível canal. No entanto, por ser mais fácil manter as designações tradicionais, vamos passar a usar sistematicamente a seguinte terminologia daqui para a frente.

Terminologia adoptada para designar as mensagens aos diferentes níveis:

Mensagens são as mensagens do nível **aplicação**.

Segmentos (e às vezes *datagramas*) são as mensagens do nível **transporte**.

Pacotes são as mensagens do nível **rede**.

Frames são as mensagens do nível **canal**.

Alguns autores criticam o modelo da pilha TCP/IP por este ser demasiado simples quando comparado com o modelo OSI da ISO (OSI/ISO). Algumas das críticas mais

frequentes têm a ver com o facto de que o modelo TCP/IP não comporta camadas para tratamento da segurança, da coordenação, nem da representação de dados. No modelo TCP/IP consideram-se que estas funcionalidades estão todas incluídas no nível aplicação. Outra crítica tem a ver com o facto de que o nível canal não está subdividido nos sub-níveis de controlo de acesso ao meio, nível canal propriamente dito, e nível físico.

Uma resposta possível à primeira crítica tem a ver com o facto de que os problemas citados (segurança, coordenação e representação de dados) eram pouco claros na altura do desenho do modelo, têm muitas soluções possíveis dependentes do contexto aplicacional, e nem sempre se podem organizar em pilha, em particular a segurança. Tentar normalizar o que não se percebe bem acaba por ser inútil e contraproducente. A resposta à segunda crítica, relacionada com a falta de detalhe ao nível do nível canal, é simplesmente que o modelo de camadas não tem uma utilidade muito clara a este nível.

O modelo OSI/ISO teve bastante influência no ensino e análise das redes de computadores. Um sinal claro dessa influência tem a ver com a frequente utilização das designações numéricas dos níveis no modelo OSI/OSI para designar as camadas, mesmo quando as mesmas são referidas no âmbito do modelo TCP/IP. Assim, quando se diz o nível 4, estamos a referirmo-nos ao nível transporte do modelo OSI/ISO e ao nível transporte do modelo TCP/IP. O nível 3 é o nível rede nos dois modelos e o nível 2 é o nível canal nos modelos OSI/ISO e TCP/IP.

4.5 Análise e avaliação do desempenho

Uma rede de computadores é um conjunto complexo de componentes hardware e software, tendo no centro uma malha de canais e comutadores de pacotes. Como na realidade não existem recursos infinitos, e na rede muitas componentes são partilhadas e a sua capacidade dividida entre os diversos sistemas e utilizadores, colocam-se várias questões no que diz respeito ao seu desempenho (*performance*):

- Como especificar o desempenho deste sistema? Na secção 3.6 vimos alguns dos indicadores e unidades de medida em que é expresso esse desempenho.
- Como medir esses indicadores? Na secção 3.3 vimos um primeiro exemplo de como um desses indicadores pode ser medido.
- Como avaliar o resultado de diversas alternativas de concepção ou parametrização? Onde estão os estrangulamentos da rede?
- Como prever o comportamento do sistema em função das solicitações projectadas para o futuro? *etc.*

Para caracterizar o desempenho de uma rede, analisar alternativas de desenho e tentar prever o seu comportamento futuro, diversos métodos podem ser usados. Caso a rede esteja em funcionamento, é possível **obter medidas e estabelecer estatísticas** sobre o seu desempenho. Outra técnica consiste em modelizá-la usando um programa especial, chamado simulador, e **simular o seu funcionamento**. Finalmente, é possível fazer um **modelo analítico da rede** e usá-lo para obter resposta às questões colocadas.

A maioria dos modelos analíticos usados em redes de computadores são baseados na teoria de filas de espera [Jain, 1991]. Em geral, o tratamento analítico de um sistema só é possível se o mesmo for modelizado de forma sintética, tentando captar algum aspecto essencial que caracterize globalmente o seu comportamento. Tentar modelizar

analiticamente de forma completa um sistema complexo pode revelar-se uma tarefa impossível.

A outra alternativa de avaliar uma rede, para estudar o seu comportamento em função de diversas alternativas de concepção ou parametrização, ou de diversas projeções de carga, consiste em fazer um estudo baseado em simulação. Um simulador é um programa informático que executa modelos simplificados da rede e permite inferir o comportamento de facetas do sistema real.

Finalmente, para se realizarem experiências, também é possível recorrer ao sistema real. Para esse efeito podemos montar a rede num laboratório, introduzir-lhe as solicitações previstas (*e.g.*, o tráfego, as aplicações, as avarias, *etc.*) e medir o seu desempenho. Por exemplo, para medir alguns parâmetros simples, podemos usar o programa `ping` para medir o tempo de trânsito, ou o programa `iperf`⁵ para medir taxas de transferência de extremo a extremo. O software dos comutadores de pacotes dispõe de inúmeras alternativas de recolha de dados sobre o comportamento dos mesmos, e existem sistemas de colecta centralizada e processamento dessas informações que permitem obter dados sobre o funcionamento real da rede (*e.g.*, qual a taxa de utilização dos diferentes canais, quantos pacotes por segundo foram processados, *etc.*).

No entanto, muitas vezes não é realista montar e estudar o sistema real para se fazerem as experiências. Por exemplo, porque tal não é economicamente viável, ou porque não é pura e simplesmente possível, como são quase todas as experiências que envolvem introduzir alterações em redes operacionais de grande dimensão, ou na própria Internet real. Nestes casos também é possível recorrer a sistemas mistos que associam partes do sistema real a partes simuladas. Esta alternativa baseia-se na utilização de laboratórios distribuídos e emuladores.

A seguir faremos referência a algumas ferramentas que são usadas para fazer estudos, quer por simulação, quer através de métodos mistos, que envolvem simulação e componentes de sistemas reais.

Simuladores de redes

Um simulador é um programa de computador que executa um modelo do sistema real. Um simulador de redes deve, pelo menos, permitir que o seu utilizador defina a topologia da rede, incluindo os nós de comutação e os canais, defina as propriedades dos canais (*e.g.*, capacidade, tempo de propagação, *etc.*) e dos nós (*e.g.*, dimensão das filas de espera, *etc.*) e os protocolos implementados (*e.g.*, de transporte, encaminhamento, *etc.*).

É também necessário que o simulador permita definir a carga da rede (*e.g.*, o modelo de geração de pacotes ou de execução de aplicações) e, finalmente, que permita obter traços de execução (*logs*) e estatísticas sobre diversos parâmetros medidos, em diferentes momentos. Alguns simuladores dispõem de interfaces gráficas que permitem especificar a rede em estudo e visualizar os resultados da simulação. Outros baseiam-se em linguagens de *scripting* para o mesmo efeito.

Os simuladores mais comuns para análise do desempenho em redes são programas baseados no processamento de eventos discretos (*discrete event driven*). De forma simplificada, o simulador pode ser visto como um processador genérico e sequencial de uma fila de espera global de eventos. O processamento de um evento pode levar à mudança de estado dos objectos sobre os quais este actua, e pode desencadear novos eventos a serem processados no futuro. O processamento de um evento pode também levar à criação de novos objectos (novos eventos ou outros objectos).

Associado a cada evento na fila de espera existe uma marca temporal, que indica em que momento o evento deve ser tratado. Essa marca temporal é num referencial de tempo simulado, que não corresponde ao tempo físico de execução do simulador. Por

⁵O programa `iperf` é um programa constituído por um cliente e um servidor que permite medir as taxas de transferência entre dois computadores pelos protocolos TCP e UDP. Para obter acesso ao programa consultar a secção 4.6.

esta razão, apesar de os eventos serem processados sequencialmente pelo simulador, os eventos são processados e os objectos transitam de estado num espaço temporal simulado, onde eventos independentes são processados em paralelo, no momento exacto do tempo simulado. Na verdade, em 5 minutos de execução real, o simulador pode fazer avançar o tempo simulado várias horas.

Por exemplo, o simulador dispõe de geradores de tráfego que criam pacotes. Esses pacotes são marcados com a marca temporal correspondente ao momento simulado em que são emitidos pela primeira vez, e os eventos correspondentes são colocados na fila de espera global do simulador, para serem processados quando o tempo simulado for igual à sua marca temporal.

Assim, para um pacote que foi emitido por um computador, é possível calcular o momento em que ele chega a um comutador, a forma como ele deve ser encaminhado, e em que fila de espera deve ser colocado, ou se pode ser transmitido imediatamente para o comutador seguinte. Em qualquer dos casos, o tratamento consiste em determinar o estado seguinte para que passa e em que momento será de novo processado. Uma vez realizado o processamento simulado do pacote e calculado o seu novo estado e nova marca temporal, o evento correspondente é colocado de novo na fila geral de eventos do simulador, para ser tratado quando chegar a sua altura, e o simulador passa ao tratamento do evento seguinte, avançando o tempo simulado.

Os simuladores mais comuns dispõem de módulos software que implementam o modelo das componentes da rede e os protocolos mais comuns, e fornecem um *framework* de extensibilidade que permite introduzir novos protocolos, ou especializar as componentes disponíveis.

Existem inúmeros simuladores de redes. Alguns são comerciais, outros têm objectivos pedagógicos e destinam-se essencialmente a apoiar o ensino e são do domínio público, ou pelo menos de utilização livre para efeitos pedagógicos. Finalmente, existem simuladores do domínio público que foram desenvolvidos essencialmente para suporte da investigação. Na secção 4.6 serão apresentados vários exemplos de simuladores e referências para a sua obtenção.

Emuladores de redes

Quando se pretende simular de forma muito completa um sistema, é necessário que o simulador disponha de implementações também muito completas não só de todas as facetas da rede, mas também dos protocolos de transporte, das aplicações, e até de partes do sistema de operação dos computadores. A maioria dos simuladores não são assim tão completos e, mesmo se o fossem, a montagem de cada experiência tornar-se-ia demasiado complexa.

A alternativa consiste em fazer coexistir sistemas reais com dispositivos, chamados emuladores, que processam os verdadeiros pacotes, emulando o seu encaminhamento pela verdadeira rede. Por exemplo, como pode ser visto na figura 4.14, poder-se-iam ligar diversos computadores reais a um nó de comutação especial, que assegura que todos os pacotes, antes de serem entregues ao computador de destino, recebem um tratamento por um servidor especial, o emulador, que os atrasa de acordo com o modelo da rede. Ou seja, o emulador e o comutador ao qual estão ligados os computadores emulam o nível rede, mas os sistemas reais executam os verdadeiros níveis transporte e aplicação.

O nó central é responsável por emular o tratamento que a rede daria aos verdadeiros pacotes: quanto tempo levam a transitar na rede, qual a ordem porque transitam, e se se perdem ou não. Os computadores ligados ao sistema central não “se apercebem” da diferença entre a comunicação através do emulador e a comunicação através da rede real.

Caso se pretenda avaliar uma rede grande, à qual estão ligados muitos computadores, através de uma infra-estrutura como a apresentada na Figura 4.14, pode ser necessário dispor de muitos computadores. O passo seguinte é substituir os computadores

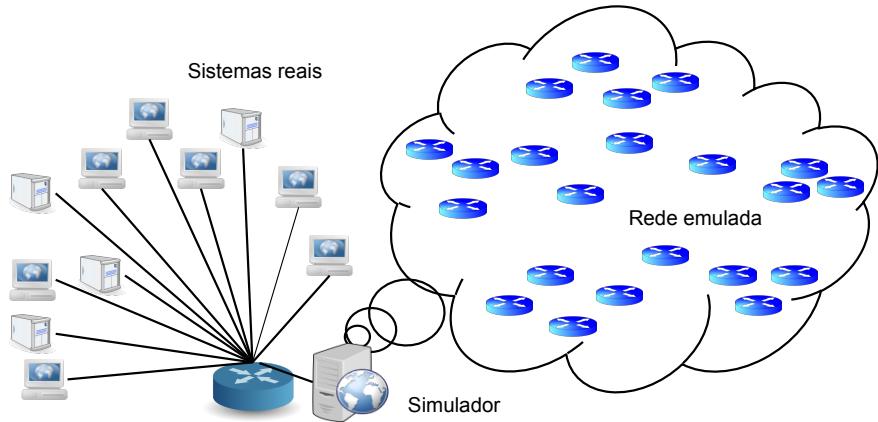


Figura 4.14: Estudo de uma rede de computadores através de um emulador

reais por computadores a executar em máquinas virtuais num ou mais computadores reais.

Assim, recorrendo a mais software, é possível diminuir os requisitos necessários em termos de hardware. No limite, todo o laboratório de teste pode basear-se na utilização de máquinas virtuais. Por exemplo, máquinas virtuais que emulam os computadores tradicionais e executam aplicações e sistemas de operação reais, máquinas virtuais que emulam comutadores de pacotes, máquinas virtuais que emulam a rede, ou partes dela, *etc.* Os *hypervisors* de gestão das máquinas virtuais incluem uma componente software que assegura a troca dos verdadeiros pacotes de dados entre o conjunto.

Este tipo de solução começa a ser popular, e muitos “simuladores” actuais passaram a incluir componentes baseadas no software real, executado em máquinas virtuais, e componentes da rede que são emuladas noutras máquinas virtuais, no sistema de operação, ou no *hypervisor*. Este tipo de sistemas usa um referencial de tempo real, e não simulado, e nisso se distinguem claramente dos simuladores puros. O realismo das medidas que permitem obter depende das características das componentes emuladas e da capacidade computacional da infra-estrutura usada. Esta infra-estrutura pode ser um simples computador, um servidor potente, ou um conjunto de servidores. Se a mesma conseguir fazer progredir o conjunto sem atrasar a execução das diferentes componentes relativamente a uma situação em que estas teriam hardware real dedicado, as medidas obtidas serão semelhantes às que se obteriam no sistema real [Handigol et al., 2012].

Não cabe aqui uma análise detalhada da oferta actual de emuladores e sistemas mistos simulação / emulação, mas na secção 4.6 serão apresentados várias referências para sistemas deste tipo.

Laboratórios distribuídos sobre a Internet

Todos os emuladores procuram emular o funcionamento do nível rede real. Para redes simples é possível emular parte das suas componentes com algum realismo (*e.g.*, do tempo de trânsito, da taxa de perda de pacotes, ...). No entanto, se o objectivo é emular o comportamento de uma rede real, da dimensão da Internet, isso é praticamente impossível, pois o emulador é sempre baseado num modelo, e todos os modelos são uma simplificação da realidade [Floyd and Paxson, 2001].

A Internet é muito complexa e, para além disso, é um sistema em contínua expansão e alteração, sobre o qual não se dispõe de informação fiável e completa. O

leitor interessado em conhecer alguns dos esforços, desenvolvidos na actualidade, para conhecer e medir a Internet com algum detalhe e rigor, poderá consultar as indicações incluídas na secção 4.6.

Factores como o aumento constante do débito médio de acesso dos utilizadores e das instituições, a evolução do número e capacidade dos dispositivos móveis ligados à Internet, a evolução da capacidade dos *backbones*, a modificação das aplicações mais populares em que a predominância actual das aplicações multimédia é cada vez mais relevante, e o aparecimento dos centros de dados e das redes de conteúdos, são outras tantas razões que mostram que a Internet está em constante evolução e transformação.

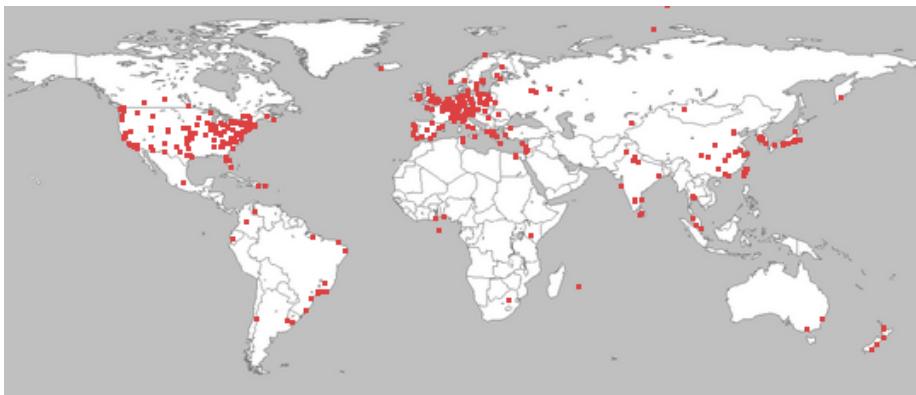


Figura 4.15: Distribuição dos nós do Planet-Lab pelo mundo (retirada de <http://www.planet-lab.org> em Julho de 2016)

Para colmatar este problema, foram desenvolvidos laboratórios colaborativos que colocam à disposição dos investigadores centenas de servidores com máquinas virtuais simplificadas, espalhadas pelos diversos continentes da Internet real. O caso mais conhecido é o sistema Planet-Lab (ver as referências apresentadas na secção 4.6), cujo nome deriva da abreviatura do acrónimo “laboratório planetário”. Os investigadores com acesso ao Planet-Lab podem usar máquinas virtuais em diferentes servidores, espalhados pelo mundo (ver a Figura 4.15), e assim terem acesso a uma infra-estrutura cujo nível rede não é emulado, mas corresponde ao nível rede da Internet real. Mesmo assim, como os nós do sistema estão instalados em universidades e laboratórios de investigação, os mesmos podem não ser representativos do nível rede visto pelos utilizadores residenciais da Internet.

Em resumo, com o objectivo de estudar de forma exacta o desempenho de uma rede, é necessário poder testar e medir o objecto real, *i.e.*, a verdadeira rede. Em geral, excepto em casos muito simples, tal não é possível ou é demasiado caro. Em alternativa, é possível recorrer a simuladores, emuladores e partes da verdadeira rede. Em todos estes casos, os resultados obtidos são necessariamente diferentes dos que se obteriam medindo o sistema real. Só uma análise e crítica sérias poderão avaliar a utilidade real dos resultados obtidos por esses métodos.

4.6 Resumo e referências

Resumo

Neste capítulo foram introduzidos alguns princípios e modelos conceptuais que são frequentemente utilizados em redes em geral, e nas redes TCP/IP em particular.

Para começar, o princípio de privilegiar os extremos (*end-to-end principle*), que indica que, idealmente, toda a complexidade deve ser relegada para a periferia, *i.e.*, para os extremos ou camadas superiores do sistema, a menos de constrangimentos de desempenho sérios que recomendem a implementação de funcionalidades específicas a níveis inferiores. Adicionalmente, um sistema bem desenhado, deve permitir flexibilidade de introdução de novas funcionalidades nas camadas superiores, com um impacto mínimo nas camadas inferiores e maximizando a extensibilidade.

Em seguida analisámos porque razão este princípio, quando aplicado às redes TCP/IP, favoreceu a adopção da noção de rede sem estado e de melhor esforço (*stateless best-effort delivery network*). Este princípio arquitectural confere uma grande flexibilidade de implementação às redes TCP/IP, favorece a sua escalabilidade e é coerente com a responsabilização dos extremos e o seu envolvimento no correcto funcionamento da rede.

A procura da escalabilidade, flexibilidade e extensibilidade tem as suas vantagens. No entanto, quer o ambiente em que os protocolos TCP/IP foram desenvolvidos, quer as opções arquitecturais em que os mesmos se baseiam, relegaram para segundo plano problemas críticos, como a segurança e a contabilização da utilização dos recursos. Estas facetas estão na base de algumas fragilidades, em particular aquelas que permitem os ataques de negação de serviço, que são hoje frequentes.

Alguns modelos matemáticos têm sido muito úteis para a análise e modelação de redes. Entre estes avultam os modelos da teoria de grafos. Um outro modelo, aparentado com os modelos usados na organização dos sistemas de software, é o modelo das camadas, um modelo que se tem revelado importante para a compreensão e organização dos diferentes “módulos” que formam uma rede.

Um modelo de camadas ou níveis é assim caracterizado:

- Cada nível introduz uma abstração bem definida, isolada e diferente da dos restantes níveis. Cada camada introduz um conjunto bem definido de serviços, acessíveis pela interface do nível.
- As entidades de um nível só comunicam com entidades do mesmo nível.
- As entidades de um nível usam os serviços oferecidos pela interface do nível imediatamente abaixo.
- O conjunto dos níveis formam uma hierarquia estrita organizada em pilha.

A discussão do modelo das camadas revelou-se igualmente importante para clarificar a terminologia usada em redes de computadores e para designar as mensagens a diferentes níveis do sistema. Assim,

- mensagens são as mensagens do nível aplicação,
- segmentos (e às vezes *datagramas*) são as mensagens do nível transporte,
- pacotes são as mensagens do nível rede, e
- *frames* são as mensagens do nível canal.

Finalmente, para caracterizar o desempenho de uma rede, analisar alternativas de desenho ou parametrização e tentar prever o seu comportamento futuro, diversos métodos podem ser usados. Caso a rede esteja em funcionamento, é possível obter medidas e estabelecer estatísticas sobre o seu desempenho. Outra técnica consiste em modelizá-la usando um programa especial, chamado simulador, e simular o seu

funcionamento. Vimos então o que são simuladores e emuladores de rede e discutimos brevemente os seus princípios de funcionamento e as suas limitações.

Uma outra alternativa usada para estudar o desempenho de uma rede é utilizar modelos analíticos fundados na teoria das filas de espera.

Quaisquer que sejam as ferramentas baseadas em modelos usadas para estudar uma rede de computadores, os resultados obtidos são necessariamente diferentes dos que se obteriam medindo o sistema real. Só uma análise e crítica sérias poderão avaliar a utilidade real dos resultados obtidos a partir de simulação, emulação ou de modelos analíticos.

Os principais termos introduzidos ou referidos neste capítulo são a seguir passados em revista. Entre parêntesis figuram as traduções mais comuns em língua inglesa.

Privilegar os extremos (*end-to-end arguments*) Princípio segundo o qual, idealmente, toda a complexidade deve ser relegada para a periferia, *i.e.*, para os extremos ou camadas superiores do sistema, a menos de constrangimentos de desempenho sérios que recomendem a implementação de funcionalidades específicas a níveis inferiores.

Rede sem estado e melhor esforço (*stateless best-effort delivery network*) Princípio segundo o qual os nós da rede apenas necessitam de memorizar estado sobre a mesma, e não necessitam de memorizar estado sobre cada fluxo de pacotes particular que a atravessa. Por outro lado, em caso de anomalias, *e.g.*, incapacidade de suportar o ritmo com que os pacotes chegam, ou avarias graves, os nós não são obrigados a recuperar os pacotes em circulação, ou a avisar a periferia das falhas.

Este princípio confere uma grande flexibilidade de implementação, favorece a escalabilidade e é coerente com a responsabilização dos extremos e o seu envolvimento no correcto funcionamento da rede.

Modularidade (*separation of concerns*) Princípio segundo o qual as diferentes componentes da arquitectura de uma rede, dos canais às aplicações, deve seguir uma organização modular em que as diferentes componentes têm interfaces bem definidas e escondem umas das outras os detalhes de implementação.

Ataque de negação de serviço (*denial of service attack*) Ataque que consiste em encontrar formas de impedir a vítima do mesmo de usar ou fornecer serviços através da rede.

Neutralidade da rede (*network neutrality*) Princípio segundo o qual a gestão da qualidade de serviço dentro da rede, para a diferenciação do serviço prestado a diferentes fluxos de pacotes, não toma em consideração critérios de concorrência comercial entre serviços com requisitos de qualidade de serviço semelhantes.

Modelo de rede em camadas (*network layers model*) Modelo de organização da estrutura lógica de uma rede em termos de camadas. Nas redes TCP/IP foram identificadas, numa visão de cima para baixo, as camadas: aplicação, transporte, rede e canais.

Simulador de redes (*network simulator*) Programa de computador que executa um modelo do sistema real, simulando todas as suas componentes e respectivas características (canais, comutadores, protocolos, aplicações, *etc.*).

Os simuladores permitem definir a carga da rede e obter traços de execução (*logs*) e estatísticas sobre diversos parâmetros medidos, em diferentes momentos.

Emulador de redes (*network emulator*) Sistema com os mesmos objectivos que os simuladores, composto por várias componentes (computadores, comutadores, máquinas virtuais, *etc.*) em que os computadores executam o software convencional e enviam pacotes como se estivessem ligados a uma rede, enquanto que outras componentes encaminham esses pacotes simulando o funcionamento de uma rede e dos seus comutadores.

Referências

Neste capítulo começámos por referir alguns princípios fundamentais que presidem à concepção e desenho de sistemas informáticos complexos, como sistemas de operação ou redes de computadores. Os artigos [Lamson, 1983] e [Saltzer et al., 1984] são artigos célebres sobre esta problemática, de leitura obrigatória para qualquer estudante de sistemas.

A aplicação desses princípios no caso de uma rede complexa, é ilustrada paradigmaticamente pela forma como o conjunto da arquitectura do sistema de protocolos TCP/IP foi concebida. No artigo [Clark, 1988] procura-se sintetizar de forma abrangente essa arquitectura e explicar os seus êxitos e fraquezas. Os estudantes de redes devem procurar ler este artigo.

Os modelos matemáticos que mais têm sido aplicados ao estudo das redes são a teoria dos grafos e a teoria das filas de espera. Os livros [Gross and Yellen, 2005] e [Jain, 1991] são reputados como suportes do estudo de ambas as teorias.

Os modelos de camadas foram usados desde muito cedo na análise e organização das redes. O artigo [Zimmermann, 1980] contém uma das primeiras apresentações formais e sistemáticas de um modelo de camadas. O livro [Day, 2008] revisita o tema das camadas usadas na descrição das redes TCP/IP, põe em evidência as suas lacunas, e propõe alternativas. Vários artigos apresentam alguns esforços em curso para revisitá-los os princípios e modelos arquitecturais que presidiram ao desenho da Internet. Um bom exemplo é o da autoria de Anja Feldmann [Feldmann, 2007].

Para se fazerem estudos sobre o desempenho de redes usam-se redes reais, métodos analíticos, simuladores, emuladores, e laboratórios distribuídos. Quase todas essas soluções baseiam-se em modelos simplificados, cuja validade está sempre sujeita a escrutínio. No artigo [Floyd and Paxson, 2001] Floyd e Paxson mostram porque razões é muito difícil simular adequadamente a Internet.

Apontadores para informação na Web

- Existem muitos esforços para fazer levantamentos da topologia e realizar medições com o objectivo de caracterizar a Internet.

O site <http://www.topology-zoo.org> – The Internet Topology Zoo, contém uma galeria com a topologia de diversas sub-redes da Internet. Os sites da Caida – Center for Applied Internet Data, <http://www.caida.org>, e do RIPE - European IP Networks, <https://www.ripe.net>, contém imensa informação sobre medidas, estatísticas e representações de partes da estrutura da Internet.

- A ISO – International Standards Organizations, <http://www.iso.org>, é uma entidade que agrupa organismos nacionais de normalização e que teve um papel importante na difusão do modelo de camadas, conhecido como modelo OSI (Open Systems Interconnection), norma ISO 7498. A ISO coordena a sua actividade de normalização com a ITU, ver a Secção 1.7, e a mesma norma do modelo é também designado por norma X.200 da ITU.
- O programa `iperf` é um programa constituído por um cliente e um servidor que permite medir as taxas de transferência entre dois computadores pelos protocolos TCP e UDP e está disponível a partir de <https://github.com/esnet/iperf>.
- O site <http://bgp.potaroo.net>, mantido por Geoff Huston, contém na secção “BGP Table” imensa informação sobre o funcionamento da Internet na zona de interligação das suas diferentes sub-redes.
- OPNET é um simulador de redes comercial, acessível gratuitamente para efeitos de ensino e investigação, integrado num conjunto de ferramentas para análise e optimização de redes. Recentemente a empresa que o desenvolveu foi adquirida pela Riverbed (<http://www.riverbed.com>).

- OMNeT++ é um simulador do domínio público, integrado com o IDE Eclipse, e acessível a partir <http://omnetpp.org/intro>.
- ns-2 é um simulador baseado em eventos discretos, do domínio público, desenvolvido em conjunto por várias universidades desde 1998. O simulador foi especialmente concebido para o suporte da investigação em redes TCP/IP. Continua acessível a partir de <http://nsnam.isi.edu/nsnam>.
- ns-3 é um novo simulador, de nova geração, desenvolvido a partir da experiência do ns-2, e com os mesmos objectivos educacionais e académicos. Está acessível em <http://www.nsnam.org>.
- Emulab é uma infra-estrutura de emulação de redes, permitindo realizar experiências para efeitos de investigação, em que os computadores são nós reais e a rede tem a configuração seleccionada em cada experiência. Consultar <https://www.emulab.net>.
- CORE (Common Open Research Emulator) é um projecto de código aberto (*open-source project*) com objectivos educacionais e de suporte à investigação, que permite realizar simulações / emulações com recurso a máquinas virtuais simplificadas, e que corre numa máquina pessoal ou em vários servidores, e está disponível a partir de <http://www.nrl.navy.mil/itd/ncs/products/core>.
- Mininet é um projecto semelhante ao anterior, mas especializado no estudo de redes controladas segundo o paradigma SDN – Software Defined Networks. Está acessível a partir de <http://www.mininet.org>.
- O Planet-Lab é constituído por um conjunto de servidores, instalados em diversas universidades e laboratórios de investigação, que disponibiliza aos seus utilizadores conjuntos de máquinas virtuais espalhadas pela Internet, que podem ser usadas para fazer experiências de novos sistemas e protocolos. O site do projecto está acessível em <http://www.planet-lab.org>. O ramo europeu do projecto é acessível em <http://www.planet-lab.eu>.
- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.

4.7 Questões para revisão e estudo

Dada a natureza deste capítulo, muitas das propostas para revisão e estudo que se seguem não são problemas, mas sim propostas de elaboração de estudos sintéticos.

1. Elabore uma síntese do artigo [Saltzer et al., 1984] em 5 páginas no máximo.
2. Elabore uma síntese do artigo [Clark, 1988] em 5 páginas no máximo.
3. Elabore uma síntese do artigo [Saltzer et al., 1984] em 2 páginas no máximo.
4. Elabore uma síntese do artigo [Clark, 1988] em 2 páginas no máximo.
5. Elabore um ensaio sobre as semelhanças e diferenças entre os artigos [Lampson, 1983] e [Saltzer et al., 1984].
6. Quais das seguintes afirmações estão integralmente de acordo com o princípio “privilegiar os extremos”?
 - (a) Só devem ser tratadas no interior da rede as funcionalidades que não podem ser tratadas nos seus extremos.
 - (b) Só devem ser tratadas no interior da rede as funcionalidades que não podem ser tratadas nos seus extremos, ou cuja implementação nos extremos tem um impacto negativo sobre o desempenho.

- (c) As funcionalidades cuja implementação no interior da rede têm um custo não menosprezável e que nem todos os computadores nos extremos necessitam, devem ser, se possível, relegadas para uma implementação nos extremos.
7. Verdade ou mentira?
- Um sistema extensível permite a introdução de novas funcionalidades através da composição e extensão de funcionalidades já existentes. O seu desenho deve facilitar este processo, disponibilizando funcionalidades de mais baixo nível que suportem o desenvolvimento futuro de necessidades ainda não antecipadas.
 - Uma funcionalidade equivalente à do protocolo TCP, se não estivesse disponível nos sistemas de operação, não poderia ser implementada noutra nível da rede.
 - O protocolo TCP tem de ser necessariamente implementado sobre o protocolo UDP pois existe uma hierarquia estrita entre os diferentes protocolos de transporte.
8. Verdade ou mentira: o princípio “rede sem estado e baseada na qualidade de serviço melhor esforço” implica que o nível rede não pode basear-se na noção de circuito virtual? Um circuito virtual de pacotes é um fluxo de pacotes, da mesma origem e para o mesmo destino, que segue um caminho pré-definido dentro da rede. Desenvolva os seus argumentos e analise as seguintes afirmações no quadro de uma rede de circuitos virtuais.
- Se um canal ficar inoperacional, como podem os circuitos que o atravessam continuar operacionais?
 - Se um comutador de pacotes ficar inoperacional, como podem os circuitos que o atravessam continuar operacionais?
 - Numa rede de circuitos os pacotes podem chegar fora de ordem?
 - Numa rede de circuitos o estado nos comutadores de pacotes é proporcional ao número de circuitos que os atravessam e não à topologia da rede.
 - Numa rede de circuitos é mais fácil controlar os recursos usados por cada fluxo de pacotes da mesma origem para o mesmo destino.
9. Considere uma rede de pacotes baseada na noção de circuito virtual. Invente uma forma de nessa rede distribuir a carga do circuito por vários caminhos com igual custo.
10. Verdade ou mentira?
- O protocolo IP disponibiliza informação aos computadores que o usam sobre o funcionamento interno da rede, dando acesso a informação sobre a taxa de erros extremo a extremo, ou a velocidade de transferência extremo a extremo.
 - O protocolo TCP dá indicações às aplicações sobre a estrutura interna da rede de forma a que estas se possam adaptar e retirar o melhor desempenho possível das conexões TCP.
 - Os protocolos IP e TCP não dão nenhuma indicação sobre como se está a comportar a rede do ponto de vista da qualidade de serviço das comunicações entre dois computadores.
11. Elabore um ensaio, incluindo exemplos, sobre como a segurança pode ser violada numa rede baseada nos protocolos TCP/IP.

12. Qual a principal característica do funcionamento interno do protocolo IP que facilita os ataques de negação de serviço?
13. Elabore um ensaio sobre o que é a neutralidade da rede dando exemplos de como a mesma pode ser violada.
14. Elabore um ensaio sobre todas as facetas relevantes de uma rede de computadores, que foram relegadas para segundo plano na concepção inicial dos protocolos TCP/IP. Enumere algumas vantagens e defeitos dessas opções.
15. Elabore um ensaio sobre o que é um modelo de camadas. Pode inspirar-se no artigo [Zimmermann, 1980].
16. Verdade ou mentira? No quadro do modelo designado como “arquitectura em camadas da rede”?
 - (a) O modelo de camadas consiste em dividir por diferentes computadores as funcionalidades da rede.
 - (b) O modelo de camadas permite aos engenheiros separar os problemas em sub-problemas e introduzir interfaces bem definidas entre componentes da rede.
 - (c) O modelo de camadas permite evitar que agentes estranhos modifiquem o código executável dos equipamentos da rede.
 - (d) O modelo de camadas consiste em estruturar a rede num conjunto de sub-redes especializadas e interligadas entre si.
 - (e) Um nível deve fornecer um serviço bem definido e sem funcionalidades que, na maioria dos casos, os níveis superiores dispensam.
17. Verdade ou mentira? No quadro do modelo designado como “arquitectura em camadas da rede”?
 - (a) O cabeçalho de um protocolo deve incluir sempre dados sobre a forma como os canais funcionam.
 - (b) Todos os protocolos definem claramente a estrutura e a semântica dos dados que passam na parte de dados das mensagens usadas na sua implementação.
 - (c) Todos os protocolos definem claramente a estrutura e a semântica dos campos dos cabeçalhos das mensagens usadas na sua implementação.
18. Verdade ou mentira?
 - (a) Um *frame* é uma mensagem ao nível aplicacional.
 - (b) Um segmento é uma mensagem ao nível canal.
 - (c) Um pacote é uma mensagem ao nível rede.
19. Quais as características essenciais que permitem distinguir um simulador de um emulador?
20. Considere um emulador simples constituído por um computador que simula a rede e diversos outros computadores equivalentes aos computadores que a usam. Durante uma experiência, qual a propriedade do funcionamento do computador que simula a rede que permite concluir que os resultados obtidos estão de acordo com o modelo de rede usado?

Capítulo 5

Programação com Sockets em Java

“You think you know when you learn, are more sure when you can write, even

more when you can teach, but certain when you can program.”

(Pode pensar que percebeu quando estudou, estará mais seguro quando puder escrevê-lo, ainda mais seguro quando conseguir ensiná-lo, mas ficará absolutamente seguro quando conseguir programá-lo.)

– Autor: Alan J. Perlis

Na secção 1.6 do Capítulo 1, foi introduzida a Interface de Sockets, interface disponibilizada na generalidade dos sistemas de operação, que permite utilizar os serviços do nível transporte de uma rede baseada nos protocolos TCP/IP. Essa interface baseia-se num conjunto de chamadas sistema, geralmente directamente acessíveis numa biblioteca sistema, com uma interface expressa na linguagem C.

A utilização da Interface de Sockets em C conduz a um código fonte extenso e cheio de detalhes do sistema. Para dar acesso à mesma interface noutras linguagens de programação, a Interface de Sockets é adaptada. No caso da linguagem Java, existe um package, designado `java.net`¹, que também dá acesso aos serviços de transporte dos protocolos UDP e TCP e, aproveitando a natureza orientada aos objectos da linguagem Java, introduz um conjunto de funcionalidades que permitem um desenvolvimento mais rápido e a escrita de programas com menos linhas de código fonte. Em resumo, o `package java.net` dá acesso à Interface de Sockets com base num modelo de objectos, de mais alto nível que a verdadeira interface disponibilizada pelos sistemas de operação.

O `package java.net` providencia muito mais classes e interfaces do que as necessárias para aceder aos protocolos de transporte. Em particular, ele disponibiliza um conjunto de interfaces e classes que permitem desenvolver aplicações cliente / servidor com base no protocolo HTTP, que é um protocolo de nível aplicacional muito utilizado. O `package java.net` também permite usar mecanismos de segurança que complementam os canais TCP com propriedades de segurança. No entanto, esses aspectos não serão aqui tratados.

Para concluir, neste capítulo apresenta-se uma breve introdução à programação em Java de aplicações distribuídas que usam a rede directamente ao nível dos protocolos de transporte, isto é, muito próximo dos conceitos que foram introduzidos sobre o funcionamento interno das redes de computadores.

¹ Package, no contexto de linguagens de programação, pode ser traduzido por pacote. Neste capítulo referimo-nos a um dos *packages* da linguagem Java, que é uma facilidade específica desta linguagem, e por isso continuaremos a usar `package java.net` para o designar.

O ponto de vista adoptado será o de mostrar como a interface do nível transporte pode ser usada para desenvolver aplicações directamente a esse nível da rede. O leitor deverá ter presente que existem outros métodos mais expeditos e eficazes de desenvolvimento de aplicações de distribuídas, que usam *frameworks* de mais alto nível e com mais funcionalidades disponíveis *a priori*, mas apenas aplicáveis a contextos específicos. Procurar-se-á também, sempre que possível, relacionar os conceitos da interface de transporte com o que se passa na rede e no sistema para permitir perceber o que está por detrás da “mágica” e como esta é implementada.

5.1 Utilização de sockets UDP em Java

Um socket UDP é uma “tomada de rede para comunicação” (*communication end-point*) pela qual é possível enviar e receber datagramas UDP. Para que um socket UDP possa ser usado, ele tem de estar necessariamente ligado a um programa em execução no computador, e é caracterizado por um número de porta UDP e um endereço IP desse computador, ver a Figura 5.1.

Um socket não necessita de estar ligado a nenhum outro socket para poder enviar e receber datagramas. No entanto, para poder enviar um datagrama para outro socket, é necessário conhecer o endereço IP e porta a que o socket de destino está ligado.

Os sockets UDP são *tomadas de rede para comunicação* dos programas, que permitem a comunicação através de datagramas UDP. Essa comunicação tem lugar entre programas no mesmo computador, ou em computadores distintos ligados via a rede.

Um socket UDP é caracterizado por um endereço IP e uma porta e tem de estar ligado a um programa activo no computador a que pertence. Para poder enviar um datagrama para outro socket, é necessário conhecer o respectivo endereço IP e porta.

Quando um programa recebe um datagrama, tem acesso ao endereço IP e porta do socket que o emitiu.

O package `java.net` disponibiliza 3 classes relevantes para os nossos objectivos: endereços IP, datagramas UDP e sockets UDP ou sockets datagrama.

Endereços IP – classe `InetAddress`

Um aspecto que o leitor deve ter em atenção daqui para a frente está relacionado com o facto de que um computador com uma só interface de rede, tem um único endereço IP, mas um computador com mais do que uma interface rede, tem um endereço IP distinto por cada uma das interfaces de rede activas. Por exemplo, a maioria dos computadores portáteis têm uma interface Ethernet com fios e uma interface Wi-Fi. Se ambas as interfaces estiverem activas, o computador tem dois endereços IP distintos.

Ao nível do sistema e da rede um endereço IP é representado por um número binário de 32 bits (na versão 4 do protocolo IP ou IPv4) ou de 128 bits (na versão 6 ou IPv6). Para facilitar a escrita de endereços IP, os da versão 4 são geralmente escritos indicando o valor de cada um dos 4 bytes que os formam separados por pontos (*e.g.*, 193.136.12.1, 200.10.78.9, ...). Esta forma de representação diz-se *dotted-quad notation*. Para facilitar a utilização de endereços IP, como vimos na secção 1.5, o sistema DNS permite obter o endereço IP associado a um nome mnemónico como por exemplo www.wikipedia.org. Adicionalmente, na maioria dos sistemas existem nomes

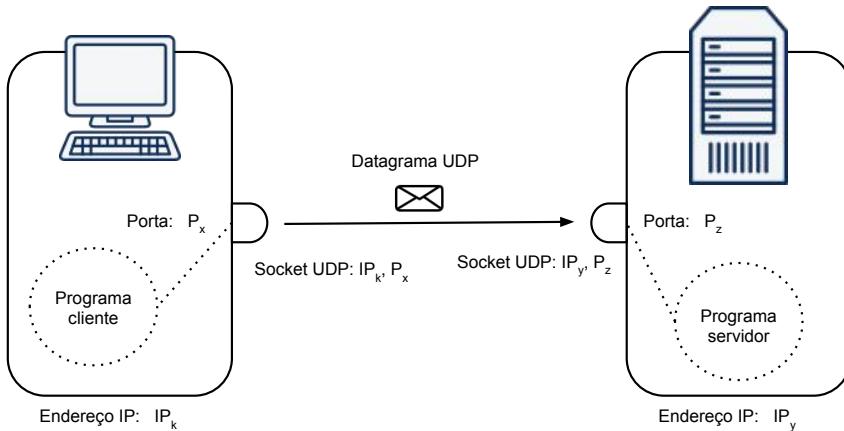


Figura 5.1: Computadores, endereços IP, portas e sockets UDP

por omissão, como por exemplo "localhost", e estes também podem ser usados ao invés de nomes mnemónicos.

A classe `InetAddress` permite criar objectos que representam, no programa Java, um endereço IP a partir de qualquer uma destas formas de representação (IPv4 ou IPv6), ou consultando automaticamente o DNS. Por exemplo, através do método estático `getByName` da classe, é possível criar objectos que representam os endereços IP em que estamos interessados, ver a listagem 5.1.

Listing 5.1: Exemplos de criação de objectos da classe InetAddress

```

1 InetAddress myself = InetAddress.getByName("localhost");
2 InetAddress myself = InetAddress.getByName("127.0.0.1");
3 InetAddress myself = InetAddress.getLocalHost();
4 InetAddress server = InetAddress.getByName("www.wikipedia.org");
5 InetAddress server = InetAddress.getByName("200.10.78.9");
6 InetAddress [ ] servers = InetAddress.getAllByName("google.com");

```

Os objectos assim criados podem depois ser passados em parâmetro dos métodos ou construtores que necessitam de endereços IP. Os dois primeiros exemplos, nas linhas 1 e 2 da listagem 5.1, retornam objectos equivalentes pois "localhost" é o nome convencionado do endereço IP 127.0.0.1, que representa o próprio computador local em IPv4. É também possível obter um endereço de uma das interfaces de rede do computador através do método estático `getLocalHost` da classe `InetAddress`, como ilustrado na linha 3.

O exemplo da linha 4 pode retornar uma exceção no caso em que não tenha sido possível obter o endereço IP associado ao nome `www.wikipedia.org` pois o mesmo pode não existir no DNS. O exemplo da linha 5 cria um objecto que representa o endereço IP 200.10.78.9, mas pode não existir nenhum computador com aquele endereço ou, tal como no exemplo anterior, o mesmo pode ser inacessível ao computador. A criação do objecto endereço IP é uma acção meramente local ao computador que a realiza.

Finalmente, o exemplo da linha 6 mostra que associado a um nome no DNS pode haver mais do que um endereço IP. O método estático `getAllByName` permite obtê-los a todos.

A classe `InetAddress` dispõe do método `String getHostAddress()` que retorna o valor do endereço escrito em *dotted-quad notation*.

A definição completa da classe deve ser consultada através da documentação oficial do package `java.net` que está disponível no *site* indicado na secção 5.4.

Datagramas UDP – classe `DatagramPacket`

Um *datagrama* UDP tem o formato indicado na Figura 5.2, ou seja, a nível da rede trata-se uma mensagem, encapsulada num pacote IP, formada por um cabeçalho UDP e um corpo ou *payload*. A norma do protocolo IP permite que um pacote IP tenha no máximo 64 KBytes incluindo cabeçalhos, e portanto um datagrama UDP poderá ter no máximo 64 K - 28 bytes, *i.e.*, 65508 bytes no *payload* (a parte “útil” do datagrama).

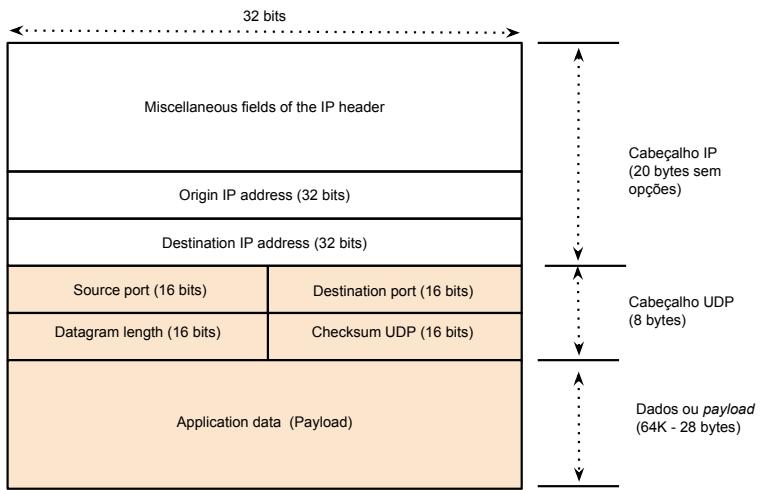


Figura 5.2: Formato de um datagrama UDP encapsulado num pacote IP

Na verdade, este valor é um pouco teórico e, apesar de ser possível, não é aconselhável. De facto, como a grande maioria dos canais não suportam *frames* dessa dimensão, o datagrama seria fragmentado e enviado usando vários *frames*, o que potencia alguma sobrecarga do nível rede e pode revelar-se delicado quando a taxa de erros de algum dos canais atravessados é elevada.

Caso não existam razões significativas que recomendem o contrário, a maioria dos pacotes IP devem poder ser enviados num único *frame*. Uma solução possível é tomar como referência o tamanho máximo do *payload* de um *frame* Ethernet normal, isto é, 1500 bytes, pelo que se retirarmos os 20 bytes do cabeçalho IP e os 8 bytes do cabeçalho UDP, ficamos com um *payload* máximo de 1472 bytes se quisermos que o nosso datagrama caiba num único *frame* Ethernet.

Do ponto de vista da rede, o *payload* é uma mera sequência de bytes, que é materializado por um vector de bytes, traduzido num `buffer` ao nível dos objectos datagramas UDP em Java. Este vector tem de ser explicitamente criado e passado em parâmetro do construtor do datagrama em Java. A seguir são apresentados dois exemplos de criação de datagramas UDP usando a classe `DatagramPacket` do package `java.net`.

Neste primeiro exemplo pediu-se ao utilizador que escrevesse uma mensagem na consola e a mesma foi lida para o `String msg`, linhas 1 a 3, depois foi criado um vector de bytes com os bytes correspondentes à mensagem lida, linha 4, foi criado um datagrama, linhas 5 e 6, e finalmente o datagrama é transmitido, linha 7.

Os parâmetros deste construtor da classe `DatagramPacket` são o vector de bytes e a sua dimensão, *i.e.*, o *payload* do datagrama a enviar, um endereço IP e uma porta (um inteiro) que correspondem ao endereço IP e porta de destino do datagrama. O endereço

Listing 5.2: Exemplo de criação de um objecto da classe `DatagramPacket` para ser enviado

```

1 Scanner in = new Scanner( System.in );
2 System.out.printf("Type\u00a0a\u00a0message:\u00a0");
3 String msg = in.nextLine();
4 byte[] msgData = msg.getBytes();
5 DatagramPacket request =
6     new DatagramPacket(msgData, msgData.length, server, port);
7 socket.send(request);

```

IP e porta origem do datagrama emitido serão os do socket usado para a emissão do datagrama, ver a seguir. A dimensão do *payload* foi explicitamente indicada e neste caso indicou-se que a totalidade dos bytes do vector `msgData` devem ser considerados como *payload* do datagrama. Mas, se quiséssemos, podíam-se indicar menos (mas não mais).

No exemplo que se segue, um objecto da classe `DatagramPacket` vai ser criado para receber um datagrama que será recebido da rede por um socket local. O `buffer` tem a dimensão máxima de `MAXMSG` bytes.

Listing 5.3: Criação de um objecto da classe `DatagramPacket` pronto a receber um datagrama vindo da rede

```

byte[] buffer = new byte[MAXMSG];
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
socket.receive(reply);

```

O objecto da classe `DatagramPacket` que é criado está pronto a receber um datagrama UDP vindo da rede, como se se tratasse de um receptáculo no qual vai ser colocado o datagrama recebido. Na criação, o seu *payload* (o vector `buffer`) não contém nada de útil, e o endereço IP e porta também não estão definidos.

Repare-se, no entanto, que também foi explicitamente indicado qual a dimensão máxima do `buffer` que poderá ser usado pelo sistema para colocar os dados recebidos da rede. Quando este objecto for usado para lá colocar um datagrama recebido da rede, mesmo que o datagrama recebido tenha mais dados, apenas os bytes de 0 a `buffer.length - 1` do seu *payload* serão copiados para o vector `buffer` associado ao objecto `reply`, sendo os restantes bytes ignorados.

A semântica de ligação do `buffer` dos objectos da classe `DatagramPacket` ao *payload* dos datagramas que viajam na rede é uma semântica de cópia: do objecto para o *payload* do datagrama, e do *payload* do datagrama recebido para o `buffer` do objecto. O membro `length` da classe `DatagramPacket` permite controlar a quantidade de bytes copiados.

Um objecto da classe `DatagramPacket` tem associadas parte das informações dos cabeçalhos IP e UDP, *i.e.*, um endereço IP e uma porta, assim como um vector de bytes de qualquer dimensão, *i.e.*, um `buffer`, destinado ao *payload*.

O endereço IP e a porta do objecto devem identificar o socket de destino antes da emissão. Mas contém o endereço IP e porta do emissor, *i.e.*, identificam o socket emissor (ou origem) no momento da recepção. Na emissão, o endereço IP e porta origem do datagrama na rede serão os do socket usado na emissão e, na recepção, o endereço IP e porta de destino do datagrama que passou na rede são os do socket usado para a recepção.

Assim, um `DatagramPacket` acabado de receber está pronto a ser devolvido ao emissor sem mais nenhum processamento (nesse caso o *payload* será o mesmo), invocando `socket.send()` logo após `socket.receive()`.

No objecto datagrama, existem dois membros da classe associados ao vector **buffer**, um designado **offset**, por omissão com o valor 0, e outro, designado **length**, que indica quantos bytes do **buffer** contém, ou pode conter, o *payload*. O significado do membro **length** depende da situação do **DatagramPacket**.

No envio, o *payload* do datagrama a enviar pela rede será construído copiando **length** bytes do vector do objecto, a começar na posição **offset**. Na recepção, **length** bytes do *payload* do datagrama recebido serão copiados para o vector de bytes do objecto datagrama, começando a colocá-los na posição **offset**. Mas que acontece se no datagrama recebido não existem **length** bytes? Nesse caso apenas serão copiados os presentes no *payload* recebido da rede, e o membro **length** será ajustado para essa quantidade.

Ou seja, o vector de bytes associado ao objecto **DatagramPacket** é um receptáculo para o qual serão copiados os bytes do datagrama recebido (até à dimensão indicada, mesmo que o vector seja maior, ou até se esgotarem os dados do datagrama recebido, caso em que o parâmetro **length** será ajustado) e é o receptáculo ao qual, no momento do envio, o sistema de operação vai buscar o número de bytes indicados para construir o datagrama a enviar para a rede.

A classe **DatagramPacket** tem mais construtores e métodos que dão muita flexibilidade à manipulação dos objectos datagrama. Voltaremos a este assunto mais adiante.

Para enviar e receber os datagramas é necessário usar os sockets UDP.

Sockets UDP – classe **DatagramSocket**

Um objecto da classe **DatagramSocket** corresponde a um socket UDP. Ele dispõe de métodos, **send** e **receive**, para enviar e receber datagramas UDP, *i.e.*, objectos da classe **DatagramPacket**.

Um dos seus construtores não tem nenhum parâmetro, e o socket criado recebe implicitamente como endereço IP um endereço IP do computador onde executa o programa que o criar, e uma porta cujo valor será automaticamente escolhido pelo sistema. Este é o exemplo típico de um programa cliente, pois a porta usada para comunicar é indiferente, como é ilustrado na primeira linha da listagem 5.4. No caso de um servidor, ver a segunda linha na mesma listagem, para criar o seu socket usa-se um parâmetro suplementar no construtor para indicar a porta a que o socket deve ficar associado. Esta porta deve ter sido acordada com os clientes para que estes possam contactar o servidor. No exemplo, o endereço IP é um endereço escolhido pelo sistema do computador onde o programa executa. No entanto, existe um construtor que permite passá-lo em parâmetro, para o caso em que o computador tenha mais do que uma interface rede.

Listing 5.4: Criação sockets UDP usando a classe **DatagramSocket**

```
DatagramSocket clientSocket = new DatagramSocket();
DatagramSocket serverSocket = new DatagramSocket(PORT);
```

Antes de prosseguirmos convém ainda referir alguns aspectos suplementares. Nos sistemas da família Unix (Linux, Mac OS X, Android, ...), as portas com números inferiores a 1024 correspondem geralmente a protocolos bem definidos, e estão reservadas a programas que executam com direitos especiais, correspondentes a servidores bem definidos que só podem ser lançados pelo administrador do sistema.

Por exemplo, a porta 53 corresponde ao protocolo DNS, e só deve ser usada em sockets locais por verdadeiras servidores DNS. Outro aspecto a ter em atenção é que não podem haver sockets UDP distintos associados ao mesmo endereço IP e porta, pois nesse caso o sistema não saberia a que socket deveria entregar o datagrama recebido².

²Um socket pode ficar associado a um endereço IP virtual especial (0.0.0.0 no caso do IPv4, :: no IPv6), com o significado de “qualquer dos endereços IP do computador”.

Para perceber melhor a sincronização que tem lugar com os sockets, é necessário ter presente que a cada socket UDP estão associadas duas filas de espera, como ilustrado na Figura 5.3. A fila de espera contendo os datagramas enviados pelo socket, mas ainda à espera de serem transmitidos para a rede, e a fila de espera dos datagramas recebidos da rede, e à espera que seja executado o método `receive` sobre esse socket. Ou seja, associado a cada socket UDP existem duas filas de espera (de envio e recepção) geridas com um mecanismo de sincronização do tipo produtor / consumidor.

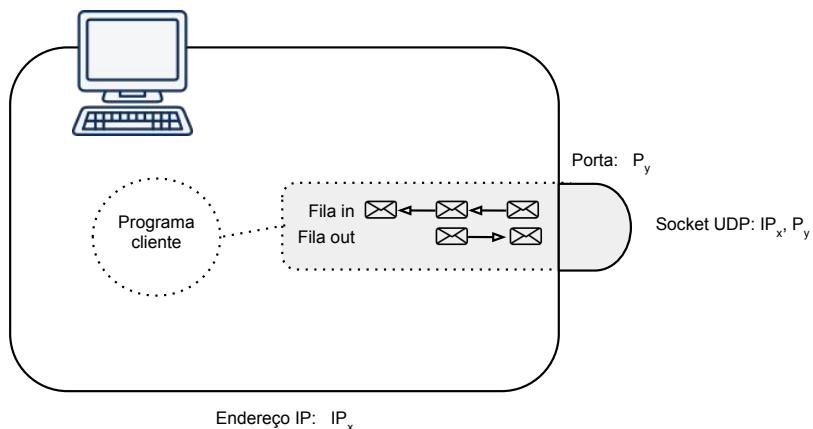


Figura 5.3: Filas de espera associadas a um socket UDP

Dependendo dos sistemas de operação, existem limites concretos para estas filas de espera, e a classe `DatagramSocket` permite conhecê-los e alterá-los (através dos métodos `get/set Receive/SendBufferSize`). A forma como os valores são contabilizados varia de sistema para sistema, mas o mais importante é ter em atenção que os datagramas recebidos vão para uma fila de espera, aqui designada `ReceiveBuffer` e, se não forem consumidos por um programa ligado ao socket, acabarão por enchê-la. Quando o limite máximo da fila de espera for atingido, os datagramas que chegarem a seguir serão ignorados de forma silenciosa, dado que a semântica do protocolo UDP é equivalente à semântica dita “melhor esforço” (*best-effort*) do protocolo IP.

Uma vez aqui chegados resta-nos ver com mais detalhe como podem ser usados os objectos da classe `DatagramSocket` para enviar e receber datagramas UDP através dos métodos `send` e `receive`. Como mostra a Figura 5.4, um cliente e um servidor com sockets UDP criam cada um o seu socket, e depois utilizam-no para trocar mensagens encapsuladas em datagramas UDP.

Nas listagens 5.5 e 5.6 são apresentados o exemplo de um cliente que lê uma mensagem da consola e envia-a para o servidor, o qual devolve à origem exactamente a mesma mensagem. O servidor atende os pedidos num socket com o endereço IP do seu computador e na porta 8000. O cliente recebe o endereço IP (ou o nome mnemónico) do servidor em parâmetro.

Nada obsta a que ambos os programas sejam executados no mesmo computador pois os sockets são necessariamente distintos. Em sistemas da família Unix, o utilizador pode usar o comando `netstat -p udp` para ver os sockets UDP activos no computador.

Os exemplos apresentados destinam-se apenas a pôr em evidência a utilização dos sockets pelos dois programas. Os mesmos estão incompletos de outros pontos de vista. Por exemplo, não é feito qualquer tratamento das excepções e quaisquer problemas que surgirem traduzir-se-ão em excepções não tratadas, que enviarão mensagens estranhas

Listing 5.5: Um cliente Eco em UDP

```

import java.net.*;
import java.util.*;

public class UDPEchoClient {
    private static final int MAXMSG = 255;
    private static final int PORT = 8000;

    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("usage: java UDPEchoClient servidor");
            System.exit(0);
        }
        InetAddress serverAddress = InetAddress.getByName(args[0]);
        DatagramSocket socket = new DatagramSocket();
        Scanner in = new Scanner(System.in);
        System.out.printf("Type\u00a0\u00a0message:\u00a0");
        String msg = in.nextLine();
        byte[] msgData = msg.getBytes();
        DatagramPacket request =
            new DatagramPacket(msgData, msgData.length,
                               serverAddress, PORT);
        socket.send(request);
        byte[] buffer = new byte[MAXMSG];
        DatagramPacket reply = new DatagramPacket(buffer, MAXMSG);
        socket.receive(reply);
        String answer =
            new String(reply.getData(), 0, reply.getLength());
        System.out.printf("The\u00a0answer\u00a0is:\u00a0\"%s\"\n", answer);
        socket.close();
    }
}

```

Listing 5.6: Um servidor Eco em UDP

```

import java.net.*;

public class UDPEchoServer {

    private static final int MAXMSG = 1024; // 1 Kilobyte
    private static final int PORT = 8000;

    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(PORT);
        for (;;) { // loop for ever
            byte[] buffer = new byte[MAXMSG];
            DatagramPacket request = new DatagramPacket(buffer, MAXMSG);
            // wait for the next datagram
            socket.receive(request);
            byte[] msg = request.getData();
            int msgLength = request.getLength();
            System.out.println("Recebi:\u00a0" + new String(msg, 0, msgLength));
            // reply directly to the client socket with the same payload
            socket.send(request);
        }
        // never reached, leaves by exception
        // socket.close();
    }
}

```

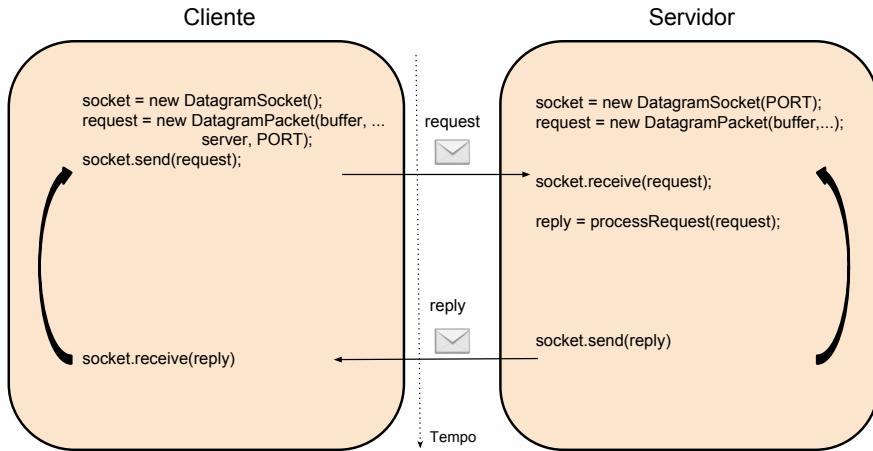


Figura 5.4: Cliente e servidor em comunicação usando sockets UDP em Java

a um utilizador menos preparado.

A classe `DatagramSocket` dispõe de um outro construtor que permite fixar explicitamente o endereço IP do computador em que o socket é criado. Esta opção só tem sentido num computador com várias interfaces de rede, e com diferentes endereços IP associados a cada uma delas. Mesmo nas situações em que o computador tem mais do que uma interface activa (*e.g.*, a interface Ethernet com fios e a interface Wi-Fi), o sistema, se não for indicado o endereço a utilizar, escolherá o endereço de uma das interfaces (*e.g.*, o endereço IP da interface Ethernet com fios).

Se num computador um socket UDP receber pacotes a um ritmo superior àquele com que os mesmos são consumidos através execução do método `receive`, a capacidade máxima de recepção do socket pode ser ultrapassada, e alguns datagramas perderem-se. Pode passar-se algo semelhante na emissão? Sim, os datagramas UDP podem perder-se mesmo antes de chegarem à rede. Se um programa usar um socket associado a uma interface lenta, e se enviar datagramas a um ritmo muito elevado, quando comparado com o ritmo de transmissão de *frames* pela interface, a fila de espera de datagramas que estão à espera de serem transmitidos pelo socket pode esgotar-se, e não ter espaço para receber mais datagramas para enviar. Nesse caso, o sistema também suprimirá os datagramas enviados a seguir através da invocação do método `send`, sem notificar o emissor, *i.e.*, o socket não produzirá nenhuma exceção.

Caso tal possa estar a ocorrer, o emissor pode usar um pequeno compasso de espera antes de emitir o próximo datagrama, por exemplo através do método:

```
Thread.sleep(milliSecondsToWait)
```

Note que este é um método ingênuo, como veremos em detalhe no capítulo 8.

Quando um socket já não tem utilidade, o programa deve fechá-lo usando o método `close()`. Desta forma libertará um recurso do sistema de que não necessita mais. Em certas circunstâncias, o compilador Java chamará a atenção do programador que se está a esquecer de fechar o socket e indicará o erro.

Quando um datagrama é enviado, pode-se saber se este chegou ao destino? Na verdade não se pode com o protocolo UDP, e o sistema nada dirá, mesmo que o socket de destino nem sequer exista. A única forma de saber, é usar um protocolo aplicacional em que o receptor (o servidor por exemplo) envie uma mensagem de

resposta, por exemplo, a assinalar a recepção. Caso não receba essa confirmação, o cliente pode reemitir o pacote com o mesmo pedido. Isso implica que o cliente deve poder fazer uma espera limitada da recepção, *i.e.*, se um datagrama não chegar até um certo limite de tempo, é necessário tentar qualquer outra acção, como por exemplo reemitir o datagrama com o pedido.

Na ausência desta hipótese, o cliente ficaria eternamente bloqueado à espera de uma resposta. É o que acontecerá com o cliente `UDPEchoClient`, caso não exista nenhum servidor `UDPEchoServer` no computador alvo.

Espera temporalmente limitada pelo próximo datagrama

Os sockets UDP têm um membro associado designado `soTimeout` (de *socket timeout*). O valor do membro indica o tempo máximo, em milissegundos, que o programa aceita ficar bloqueado à espera que esteja disponível um datagrama para ser devolvido depois de executar `receive` sobre o socket. Se o valor for 0, o valor por omissão, a espera será eterna. Se o valor de `soTimeout` for $\neq 0$, e for ultrapassado, a chamada de `receive` desencadeará a excepção `InterruptedException`.

O exemplo na listagem 5.7 mostra a utilização do mecanismo numa nova versão do cliente `UDPEchoClient`. O cliente indica que esperará no máximo 2 segundos ao fixar o valor do `soTimeout` em 2000 milissegundos na linha 20 e, caso não receba resposta do servidor, tenta até um máximo de 3 vezes (fixado pela constante `MAXATTEMPTS`). O número actual de tentativas ainda possíveis é registado pela variável `attempts`, inicializada na linha 29. A invocação de `receive` é envolvida num bloco `try - catch`, linhas 33 - 45. Caso a excepção seja desencadeada, linhas 39 - 45, o número de tentativas é decrementada e o programa termina abruptamente caso se ultrapasse o limite de tentativas. Antes de o programa se suicidar, o socket é fechado. Caso seja recebido um datagrama, o ciclo de tentativas de emissão é abandonado, na linha 39, e segue-se com o processamento normal. O método `receive` também admite um parâmetro explícito de `TimeOut` em cada invocação.

Para terminarmos esta introdução à utilização de sockets UDP na linguagem Java, vamos discutir um último aspecto que só é relevante em contextos onde os problemas de eficiência sejam significativos. A questão que se coloca é a seguinte:

É necessário um objecto do tipo `DatagramPacket` diferente por cada datagrama?

A resposta é negativa pois o objecto `DatagramPacket` é um receptáculo que pode ser reutilizado para receber tantos datagramas quanto se necessitar. Imaginando que se está perante um programa que processa milhares e milhares de pacotes e tem pouca memória, criar um objecto por cada um dos pacotes pode revelar-se um problema, pois poderia implicar uma intervenção mais significativa do gestor de memória (*garbage collector*).

Quando se pretende, é possível reutilizar um objecto do tipo `DatagramPacket` para conter, sucessivamente, diferentes datagramas. Tal implica reutilizar o `buffer` e outros membros do mesmo objecto, o que significa que quando se realiza uma nova recepção, os valores anteriores já foram tomados em consideração.

A listagem 5.8 ilustra um ciclo de recepção e processamento de datagramas que usa sempre o mesmo objecto para os colocar. Em princípio, o objecto só é reutilizado após ter sido usado e estar de novo disponível. A novidade está na linha 6, onde se invoca o método `setLength` sobre o objecto `request`, fixando de novo o número máximo de bytes contidos no *payload* do datagrama recebido que poderão ser copiados para o `buffer`. Este reajuste da dimensão máxima é necessário pois, após a execução com sucesso de `receive`, este membro do objecto foi ajustado para a dimensão do *payload*. Ora se um pacote tivesse um *payload* com 10 bytes, sem este ajuste, apenas 10 bytes do *payload* do próximo datagrama recebido seriam copiados. Para evitar isso é necessário ajustar o número máximo de bytes a copiar antes de cada recepção. O

Listing 5.7: UDPEchoClient com espera limitada e reemissão até um certo número de vezes

```

1 import java.io.*;
2 import java.net.*;
3 import java.util.*;
4
5 public class UDPEchoClient {
6
7     private static final int MAXMSG = 255;
8     private static final int PORT = 8000;
9     private static final int MAXATTEMPTS = 3;
10    private static final int TIMEOUT = 2000;
11
12    public static void main(String[] args) throws Exception {
13        if( args.length != 1 ) {
14            System.err.println("usage: java UDPEchoClient servidor");
15            System.exit(0);
16        }
17
18        InetAddress server = InetAddress.getByName(args[0]);
19        DatagramSocket socket = new DatagramSocket();
20        socket.setSoTimeout(TIMEOUT);
21        Scanner in = new Scanner(System.in);
22        System.out.printf("Type a message:");
23        String msg = in.nextLine();
24        byte[] msgData = msg.getBytes();
25        DatagramPacket request =
26            new DatagramPacket(msgData, msgData.length, server, PORT);
27        byte[] buffer = new byte[MAXMSG];
28        DatagramPacket reply = new DatagramPacket(buffer, MAXMSG);
29        int attempts = MAXATTEMPTS;
30        while (true) {
31            System.out.printf("Sendig request: \"%s\"\n", msg);
32            socket.send(request);
33            try {
34                socket.receive(reply);
35                String answer =
36                    new String(reply.getData(), 0, reply.getLength());
37                System.out.printf("The answer was: \"%s\"\n", answer);
38                break; // got a datagram and processed it
39            } catch (InterruptedException e) {
40                attempts--;
41                if (attempts == 0) {
42                    System.out.println("Too many tries, I give up!");
43                    socket.close();
44                    System.exit(0);
45                }
46            }
47        }
48        socket.close();
49    }
50 }
```

Listing 5.8: Reutilização da mesma instância de `DatagramPacket` para diferentes recepções

```

1 private static final int MAXMSG = 1024; // 1 Kilobyte
2 byte[] buffer = new byte[MAXMSG];
3 DatagramPacket request = new DatagramPacket(buffer, MAXMSG);
4 while (true) { // loop for ever
5     // wait for the next datagram
6     request.setLength(MAXMSG);
7     socket.receive(request);
8     System.out.println("Recebi:" + 
9         new String(buffer, 0, request.getLength()));
10    // .....
11 }
```

exemplo também ilustra como processar directamente o *payload* recebido através do acesso directo ao vector `buffer`.

A classe `DatagramPacket` dispõe dos construtores indicados a seguir. Os dois primeiros destinam-se geralmente a criar objectos receptáculo de datagramas. O segundo permite posicionar o índice inicial de `buffer` para onde se começará a copiar os bytes recebidos por `receive`. O terceiro e o quarto são construtores que contém já os valores do endereço IP e porta para onde o datagrama deve ser enviado. Tipicamente, serão usados para criar datagramas a enviar.

```

DatagramPacket (byte[] buffer, int length)
DatagramPacket (byte[] buffer, int offset, int length)
DatagramPacket (byte[] buffer, int length,
               InetAddress remoteAddr, int remotePort)
DatagramPacket (byte[] buffer, int offset, int length,
               InetAddress remoteAddr, int remotePort)
```

No entanto, a classe é muito flexível e dispõe de métodos que permitem aceder e modificar todos os membros do objecto. A seguir referem-se os mais significativos.

```

InetAddress getAddress()
void setAddress(InetAddress address)
int getPort()
void setPort(int port)
```

Os dois primeiros métodos retornam e modificam o endereço IP associado ao objecto. Para além do método `setAddress` e do construtor, o método `receive` também modifica o membro endereço IP associado ao objecto afectando-lhe o endereço IP do emissor do datagrama. Os dois métodos seguintes permitem, de forma análoga, manipular a porta associada ao objecto datagrama, com exactamente o mesmo significado. Os dois métodos abaixo

```

int getLength()
int setLength(int length)
```

retornam e modificam o membro `length` do objecto datagrama, respectivamente. Para além do construtor e do método `setLength`, o método `receive` também modifica este membro e usa-o da seguinte forma: aquando da chamada `receive` ele indica o número máximo de bytes que vão poder ser copiados do datagrama recebido para o objecto, e no retorno de `receive`, o membro toma o valor do número de bytes que foram efectivamente copiados. Finalmente, os métodos abaixo permitem manipular o membro `buf` da classe

```
byte[] getData()
```

```
void setData(byte[] buffer)
void setData(byte[] buffer, int offset, int length)
```

`getData` retorna uma referência para o último objecto `buffer` que foi associado ao objecto datagrama, pelo construtor ou pelo método `setData`. É preciso ter em atenção que o vector de bytes retornado tem uma dimensão que pode não coincidir com o membro `length` do objecto datagrama e pode portanto conter dados que não correspondem ao último datagrama recebido. `setData` associa um novo `buffer` ao objecto datagrama, sem modificar `offset` ou `length`, a não ser que estes tenham de ser ajustados em função da dimensão do novo `buffer`. O terceiro método actualiza todos os três membros do objecto datagrama.

5.2 Comunicação multi-ponto – Sockets Multicast

Até ao momento todas as formas de comunicação que temos abordado são do tipo ponto-a-ponto (*i.e.*, de um para um) ou ponto-multiponto (*i.e.*, em difusão ou de um para todos). A primeira é frequentemente designada em inglês por comunicação *unicasting* enquanto que a segunda é designada por comunicação *broadcasting*. Alguns canais suportam directamente comunicação em difusão para todas as interfaces que lhe estão directamente ligadas, mas na Internet esta forma de comunicação não está disponível de forma generalizada ao nível rede.

Existe uma terceira forma de comunicação, designada por comunicação em grupo, ou de um para um grupo, que em inglês se costuma designar por *multicasting*. Esta forma de comunicação foi definida e normalizada no nível rede das redes de computadores TCP/IP e é designada por “IP Multicasting”, por contraste com a comunicação tradicional que se designa por “IP Unicasting”. Dado que o transporte UDP é uma elevação da semântica do protocolo IP para o nível transporte, acrescentando-lhe a noção de porta, o protocolo UDP também dá acesso aos serviços IP Multicasting das redes IP. Vamos ver a seguir como.

Grupos IP Multicasting

Um grupo IP Multicasting é um grupo de computadores ligados à rede ou, para ser mais preciso, um grupo de interfaces com os respectivos endereços IP, que pode ser endereçada colectivamente através de um endereço comum, dito o endereço IP Multicasting do grupo (um subconjunto especial de endereços IPv4 pois os endereços IP Multicasting estão no intervalo 224.0.0.0 a 239.255.255.255). A noção de grupo é interessante porque se for enviado um pacote IP para o grupo, uma cópia do pacote é entregue a cada um dos seus membros, ver a Figura 5.5.

Existe uma outra diferença fundamental com a comunicação IP Unicasting e que é a seguinte: um grupo tem um endereço IP único, e pode receber pacotes, mas um grupo não pode emitir pacotes. Se um membro do grupo necessitar de emitir um pacote, o endereço origem do pacote é o desse membro, e não o do grupo. Ou seja, nenhum membro do grupo pode falar em nome dos outros membros.

A filiação num grupo IP Multicasting é dinâmica, isto é, pode-se entrar e sair de um grupo, pelo que a lista dos membros do grupo é variável. A operação de entrada num grupo diz-se a subscrição ou adesão ao grupo e, em IP Multicasting, só pode ser executada pelo próprio, *i.e.*, um computador pode decidir entrar no grupo, e mais tarde sair do grupo, mas um computador não pode fazer outro computador entrar no grupo, ou expulsá-lo do grupo. Entrar e sair do grupo é um acto voluntário, executado a partir do próprio computador que entra ou sai do grupo.

Tal como para outras operações ao nível do protocolo IP, um grupo não tem controlo de acessos. Em princípio, um computador pode aderir ou emitir para qualquer

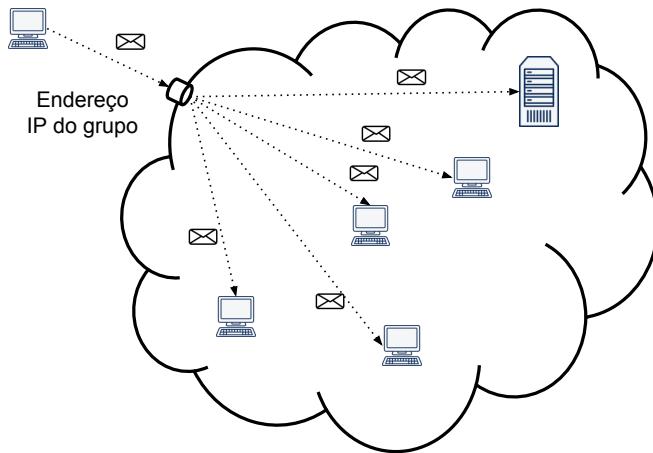


Figura 5.5: Difusão através de IP Multicasting de um datagrama para um grupo de computadores

grupo, desde que conheça o respectivo endereço IP. No entanto, como é habitual, não é possível aderir a todos os grupos que se pretenda pois, não só é necessário conhecer o seu endereço, como a implementação concreta do IP Multicasting impõe algumas restrições na adesão a certos grupos. Existem alguns grupos pré-definidos a que se pertence, ou não, por omissão, como por exemplo o grupo de todos os comutadores de pacotes IP ligados ao mesmo canal. Não é necessário (nem possível!) aderir ou sair desses grupos.

Um **grupo IP Multicasting** (*IP Multicasting group*) é um grupo de interfaces rede (e implicitamente dos seus computadores) que têm um endereço IP comum, dito endereço IP Multicasting do grupo. Se um pacote IP for enviado para o endereço do grupo, uma cópia do pacote é entregue a todos os computadores acessíveis que fazem parte do grupo. A semântica é equivalente à do nível rede, *i.e.*, de melhor esforço.

Os endereços IPv4 Multicasting estão no intervalo 224.0.0.0 a 239.255.255.255.

Para aderir, ou enviar um datagrama para um grupo, basta conhecer o seu endereço.

Aderir e sair de um grupo são ações que só podem ser executadas por programas com acesso aos sockets que entram ou saem do grupo.

A classe `InetAddress` também pode ser usada para criar objectos que representam no programa um endereço IP Multicasting. Adicionalmente, a mesma classe disponibiliza o método `boolean isMulticastAddress()` que permite testar se um endereço é ou não um endereço IP Multicasting. A listagem 5.9 ilustra a utilização deste método para testar se o parâmetro do programa corresponde de facto a um endereço IP

Multicasting.

Listing 5.9: Teste se um endereço IP é um endereço IP Multicasting

```
InetAddress group = InetAddress.getByName(args[0]);
if(!group.isMulticastAddress()) {
    System.err.println("Multicast_address_required...");
    System.exit(0) ;
}
```

A comunicação em IP Multicasting e a gestão dos grupos IP Multicasting tem muitas outras facetas que não iremos aqui tratar, pois as mesmas ultrapassam o âmbito da utilização de *multicasting* ao nível IP. Por agora, o conjunto de conceitos introduzidos é suficiente pois o objectivo é simplesmente introduzir a interface de sockets em Java para usar IP Multicasting.

Comunicação *multicasting* com datagramas UDP

O IP Multicasting está acessível às aplicações via o protocolo UDP, visto que a semântica do UDP é a do envio de datagramas UDP, *i.e.*, pacotes IP aos quais foram acrescentados uma porta origem, uma porta de destino, e um controlo suplementar de erros, através de um *checksum*, ver a Secção 2.4.

A principal diferença com o que vimos até aqui é que para trocar datagramas através de *multicasting* é necessário utilizar a classe `MulticastSocket` mas os objectos transmitidos e recebidos também são da classe `DatagramPacket`. A classe `MulticastSocket` estende a classe `DatagramSocket`, pelo que herda os mesmos membros, construtores e métodos. Um programa que deseja aderir a um grupo deverá criar um socket do tipo `MulticastSocket` mas, adicionalmente, terá de subscrever o grupo, isto é, entrar no grupo. Para esse efeito deve usar o método `void joinGroup(group)` desta classe. Repare-se que ao subscrever o grupo, a operação é aplicada a um socket IP Multicast, ao qual têm de estar necessariamente associados uma porta e um endereço IP locais. Tal como qualquer receptor em UDP, um grupo de recepção ao nível UDP deve ter uma porta associada, cujo número tem de ser conhecido pelo(s) emissor(es).

Em seguida apresenta-se um exemplo simples de utilização de *multicasting*. O emissor é uma espécie de “relógio falante”, *i.e.*, um programa que emite periodicamente (*e.g.*, uma vez em cada 10 segundos) um datagrama contendo a hora do seu relógio. O `SpeakingClock` é apresentado na listagem 5.10. O mesmo não tem grandes novidades do ponto de vista de um emissor de datagramas. As principais diferenças são o endereço IP de destino.

A listagem 5.11 apresenta um subscriptor que gosta de estar sempre atento à hora. Na verdade, enquanto não for parado, continuará eternamente a receber a hora do relógio.

Para sair de um grupo e deixar de receber os datagramas que lhe sejam dirigidos, usa-se o método `void leaveGroup()` da classe `MulticastSocket`.

A classe `MulticastSocket` tem outros métodos que permitem, em computadores com várias interfaces, indicar por que interface os datagramas são emitidos. Para além disso, é possível também controlar o alcance da difusão, isto é, quantos comutadores de pacotes IP o datagrama pode atravessar. Contudo, esses tópicos ultrapassam o âmbito desta breve introdução à comunicação multi-ponto com a interface de sockets.

Depois desta breve passagem em revista das diferentes formas como se podem usar sockets UDP, a secção seguinte fará o mesmo para os sockets baseados nos canais TCP.

Listing 5.10: Utilização de IP Multicasting para emitir a hora

```
import java.net.*;
import java.util.*;

public class SpeakingClock {

    private static final int PORT = 8000;
    private static final int INTERVAL = 10*1000; // 10 seconds

    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("usage: java SpeakingClock group");
            System.exit(0);
        }
        InetAddress group = InetAddress.getByName(args[0]);
        if (!group.isMulticastAddress()) {
            System.err.println("Multicast address required...");
            System.exit(0);
        }

        DatagramSocket socket = new DatagramSocket();
        for (;;) { // loop for ever
            String msg = "Current time is: " + new Date().toString() + "\n";
            socket.send(new DatagramPacket(msg.getBytes(),
                msg.getBytes().length, group, PORT));
            try { Thread.sleep(INTERVAL); }
            catch (InterruptedException e) { }
            System.out.print(".");
        }
        // never reached, leaves by exception
        // msocket.close()
    }
}
```

Listing 5.11: Subscrição de um grupo *Multicasting* para receber periodicamente a hora

```

import java.net.*;

public class ClockListener {

    private static final int MAXMSG = 256;
    private static final int PORT = 8000;

    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("usage:java ClockListener group");
            System.exit(0);
        }
        InetAddress group = InetAddress.getByName(args[0]);
        if (!group.isMulticastAddress()) {
            System.err.println("Multicast address required...");
            System.exit(0);
        }
        MulticastSocket msocket = new MulticastSocket(PORT);
        // subscribe the group of the speaking clock
        msocket.joinGroup(group);
        DatagramPacket msg =
            new DatagramPacket(new byte[MAXMSG], MAXMSG);
        for (;;) { // loop for ever
            msg.setLength(MAXMSG); // resize with max size
            msocket.receive(msg);
            String payload = new String(msg.getData(),0,msg.getLength());
            System.out.println(payload);
        }
        // never reached, leaves by exception
        // msocket.close();
    }
}

```

5.3 Sockets Java para canais TCP

A comunicação com canais TCP tem uma natureza diferente da que vimos acima, pois um canal TCP é um canal lógico, fiável, bidireccional, que transmite sequências de bytes entre dois interlocutores, sem noção de início e fim de mensagens. Os sockets usados em TCP são distintos dos sockets UDP, e portanto pertencem a outras classes da linguagem Java. Com efeito, em TCP, a comunicação só é possível depois de estabelecida, com sucesso, uma conexão entre dois sockets, designados em Java “stream sockets”. Assim, depois de criados os stream sockets, é necessário ligá-los antes de poderem comunicar.

O estabelecimento da conexão é assimétrico, pois um dos interlocutores tem de tomar a iniciativa de a estabelecer, e o outro é passivo, e terá de a aceitar. A seguir veremos que existem duas classes de stream sockets. Os sockets da classe **Socket** são stream sockets que vão estar na extremidade dos canais TCP e que podem ser usados para solicitar o pedido de estabelecimento do canal. Os sockets da classe **ServerSocket** são stream sockets, criados do lado de quem aceita conexões, *i.e.*, do lado do servidor, e que servem apenas para aceitar pedidos de conexão.

A classe **Socket** tem vários construtores, alguns dos quais estão listados a seguir:

```
Socket(InetAddress address, int port);
Socket(String name, int port);
```

Ambos os construtores criam um stream socket local, ao qual fica associado um endereço IP local (se o computador tiver vários endereços, é escolhido um por omissão, tal como no caso dos sockets UDP) e uma porta escolhida automaticamente pelo sistema (tal como nos sockets UDP é possível também impor, quer o endereço IP local, quer a porta local). No entanto, ao contrário dos sockets UDP, estes construtores desencadeiam imediatamente uma conexão dirigida à outra extremidade.

No caso do primeiro construtor, a outra extremidade do canal TCP é indicada através de um endereço IP e uma porta. No segundo construtor, o computador remoto é identificado pelo seu nome mnemónico (*e.g.*, www.wikipedia.org) ou por um endereço IP (*e.g.*, 192.145.156.12).

Se o construtor retornar com êxito, o stream socket encontra-se ligado à outra extremidade, geralmente um servidor, e pode ser usado imediatamente para comunicar. Existem outros construtores que não desencadeiam imediatamente a conexão, podendo esta ser mais tarde desencadeada através do método **connect()** da classe **Socket**.

Quando um stream socket está conectado ou ligado à outra extremidade, pode imediatamente ser usado para enviar e receber dados. Em Java, estes sockets implementam a interface **Stream** e associados a cada stream socket conectado existe um **InputStream** e um **OutputStream**, que são acessíveis através dos métodos da classe **getInputStream** e **getOutputStream**. Desta forma é possível ler e escrever dados *i.e.*, sequências de bytes, nos dois **streams**.

Os sockets TCP, também designados stream sockets, são *communication endpoints* que, quando conectados a outro stream socket, permitem a comunicação através de um canal TCP.

Este canal é virtual e caracterizado por permitir comunicação fiável, bidireccional, que transmite sequências de bytes, sem noção de mensagens ou outros delimitadores, entre os dois stream sockets da extremidade. O canal TCP é caracterizado pelo endereço IP e porta desses stream sockets.

Sem estar conectado, um stream socket só pode ser usado para estabelecer conexões TCP e não para comunicar dados.

Listing 5.12: Um cliente Eco em TCP

```

1 import java.io.*;
2 import java.net.*;
3 import java.util.*;
4
5 public class TCPEchoClient {
6
7     private static final int PORT = 8000;
8
9     public static void main(String[] args) throws Exception {
10         if (args.length != 1) {
11             System.err.println("usage: java TCPEchoClient servidor");
12             System.exit(0);
13         }
14
15         for (;;) { // for ever
16             Socket socket = new Socket(args[0], PORT);
17             InputStream inStream = socket.getInputStream();
18             OutputStream outStream = socket.getOutputStream();
19             Scanner in = new Scanner(System.in);
20             System.out.printf("Type a message: ");
21             String msg = in.nextLine();
22             msg += "\n"; // adds an end of line to the message
23             outStream.write(msg.getBytes());
24             String answer = new Scanner(inStream).nextLine();
25             System.out.println("The answer was: " + answer + "\n");
26             socket.close();
27         }
28         // never reached, leaves by exception
29     }
30 }
```

A listagem 5.12 mostra o código de um cliente “Eco” em TCP, semelhante na funcionalidade ao mesmo cliente que foi ilustrado em UDP.

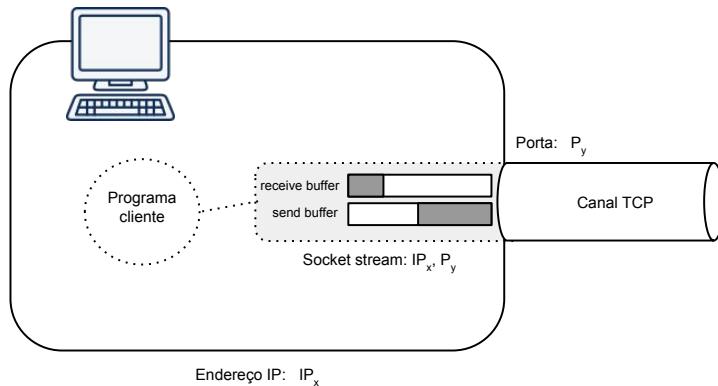
O exemplo merece alguns comentários suplementares. A mensagem a enviar ao servidor é uma sequência de bytes, correspondentes a uma linha de texto, obtida a partir da consola (na linha 21). Repare-se que o `Scanner` usado para ler a linha (criado na linha 19) devolve uma sequência de caracteres sem nenhum marca de fim de linha. Esta marca é acrescentada à linha lida (na linha 22). O método usado para transmitir dados pelo canal é, como em qualquer `Stream`, o método `write`, que admite vários parâmetros, nomeadamente um vector de bytes como no caso da linha 23, ou um inteiro, que denota o código inteiro sem sinal do byte a transmitir.

Num `Stream` ligado a um canal TCP não existe nenhuma noção de mensagem. Os bytes passados em parâmetro de `write` são colocados num `buffer` do `Stream` e depois transmitidos para um `send buffer` do socket, ver a Figura 5.6.

O protocolo TCP irá posteriormente transmitir o conteúdo do `send buffer` do socket para a outra extremidade do canal usando segmentos TCP, preenchidos segundo uma lógica independente da noção de mensagem aplicação. Geralmente, o protocolo procurará enviar segmentos TCP (ver o Capítulo 7) tão cheios quanto possível, mas evitando também atrasar demais o envio dos dados escritos. Neste exemplo concreto acabará por, mais cedo do que mais tarde, ser enviado um segmento com todos os bytes escritos de uma só vez na linha 22 da listagem. Mas um programa pode escrever dados byte a byte e serem enviados de cada vez segmentos com centenas de bytes.

Já no que diz respeito à leitura, um `InputStream` é lido através do método `read`, que geralmente tem um vector de bytes em parâmetro, como ilustrado na listagem 5.13.

Através da invocação da linha 2 acima são devolvidos os bytes disponíveis no `receive buffer`, e que caibam no vector `buf`. A quantidade de bytes de facto lidos é

Figura 5.6: Canal TCP e *buffers* associados ao stream socket

Listing 5.13: Leitura de bytes recebidos por um stream socket

```

1 byte[] buf = new byte[1024];
2 n = is.read(buf);
3 if (n == -1) ..... // this TCP connection is closed

```

devolvida pelo método. Se o valor lido for `-1`, isso quer dizer que o canal TCP está fechado. Se não existirem bytes no `receive buffer` para devolver, o método bloqueia até que hajam. Mas quantos bytes são devolvidos de cada vez? Isso não está definido e depende do ritmo com que o protocolo TCP os vai colocando no `receive buffer`. Este ritmo depende da capacidade de transmissão extremo a extremo da rede e até dos erros ocorridos.

É por isso que na listagem 5.12 do cliente se usa, na linha 24, a invocação `new Scanner(inStream).nextLine()`. Esta garante que serão lidos pelo `Scanner` tantos bytes quantos os necessários para formar uma linha inteira. Isto é, neste exemplo, a marca de fim de linha está a funcionar como delimitador de mensagens ao nível aplicacional. Trata-se de uma prática comum quando as mensagens trocadas correspondem a sequências de caracteres. O protocolo HTTP, ver a Secção 1.4, usa uma solução semelhante.

No entanto, um dos erros mais frequentes cometidos com os stream sockets, é achar que as sequências de bytes escritas ou lidas de cada vez, podem ter alguma relação com as mensagens de nível aplicacional, o que é falso.

Supondo que por qualquer motivo se sabem quantos bytes formam uma mensagem (o delimitador de mensagens agora é o seu tamanho), o seguinte extracto de código permite lê-los integralmente mesmo que os mesmos cheguem em segmentos separados e disponíveis de forma espaçada.

```

1 byte[] buffer = new byte[4096]; // max size of a record
2 int recordSize = ....; // the actual size of this record
3 int bytesReceived = 0;
4 int n;
5 while (bytesReceived < recordSize) {
6     n = is.read(buffer, bytesReceived, recordSize - bytesReceived);
7     if (n == -1) throw new SocketException("Connection closed?");
8 }

```

Neste exemplo é usada, na linha 6, uma outra variante da leitura, nomeadamente `read(buffer, offset, limit)`, que permite indicar o número máximo de bytes a ler (`limit`), assim como a posição (`offset`) a partir da qual eles devem ser colocados no `buffer`.

Vejamos agora o servidor Echo. Do lado do servidor é necessário aceitar conexões TCP para que seja estabelecido um canal TCP. Para esse efeito, o servidor tem de usar um socket especial, da classe `ServerSocket`, que lhe permite aceitar conexões.

Esta classe tem vários construtores mas os mais comuns são os seguintes:

```
ServerSocket(int port);
ServerSocket(int port, int backlog);
```

Este socket especial serve para o servidor aceitar conexões na porta local passada em parâmetro. Para aceitar uma conexão, é necessário usar o método `Socket accept()` que, quando retorna com êxito, devolve um stream socket que representa a extremidade de um canal TCP ligado a um cliente. Só a partir daí o servidor poderá dialogar com o cliente através desse canal.

Que acontece se no entretanto outro cliente tentar ligar-se? O mesmo só será atendido se o servidor invocar outra vez `accept` sobre o seu `ServerSocket` e, enquanto o servidor não o fizer, o cliente fica em fila de espera e a criação do stream socket no cliente não retorna. O parâmetro `backlog` permite especificar o tamanho dessa fila de espera (para o caso em que o valor por omissão se revelar inadequado). Se um cliente se tentar ligar a um servidor que já tem `backlog` clientes à espera do estabelecimento de conexões, a criação do stream socket do lado do cliente falha com uma exceção `IOException`.

A listagem 5.14 mostra o código de um servidor “Eco” em TCP. Este servidor aceita um cliente de cada vez. Quando um cliente se liga, o servidor entra num ciclo em que lê o que o cliente lhe envia, e responde-lhe com exactamente a mesma sequência de bytes, até que o cliente decide parar e o canal é fechado. Um canal TCP fechado não pode ser usado para ler ou escrever dados e `read(buf)` devolve -1. Nessa altura o servidor abandona este cliente e passa ao seguinte. Apesar de o cliente acima só enviar uma linha de texto ao servidor e depois fechar o socket após receber a resposta, este servidor aceitaria uma quantidade arbitrária de dados enviada sucessivamente pelo cliente.

Neste caso, o controlo da dimensão dos dados transmitidos e recebidos e o seu fim está integralmente do lado do cliente. O servidor limita-se a enviar os dados que for recebendo pelo canal TCP no sentido inverso, e só termina esse envio quando o cliente “ficar farto” e fechar o socket. O servidor também não controla a quantidade de dados lidos e enviados de cada vez, isso compete à implementação do protocolo TCP. Apenas fixa a dimensão máxima do que aceita ler de cada vez (`MAXRECORD`).

A Figura 5.7 mostra a sequência de operações e o diálogo entre um cliente a enviar múltiplos dados e este servidor no tempo.

Para além dos métodos que controlam os endereços e portas locais, a classe `Socket` tem muitos outros métodos que permitem controlar o comportamento do canal TCP e dos programas que o usam. A seguir apresentam-se os mais relevantes:

```
int getReceiveBufferSize()
int getSendBufferSize()
void setReceiveBufferSize(int size)
void setSendBufferSize(int size)
```

Estes métodos permitem conhecer e modificar a dimensão dos `buffers` internos do socket. Tal pode ser útil quando é necessário fazer afinações do desempenho do canal TCP.

```
int getSoTimeout()
void setSoTimeout(int timeout)
```

Listing 5.14: Um servidor Eco em TCP

```

import java.net.*;
import java.io.*;

public class TCPEchoServer {

    private static final int PORT = 8000;
    private static final int MAXRECORD = 1024;

    public static void main(String[] args) throws Exception {
        ServerSocket serverSo = new ServerSocket(PORT);
        for (;;) { // loop for ever
            Socket socket = serverSo.accept(); // a client connected
            InputStream is = socket.getInputStream();
            OutputStream os = socket.getOutputStream();
            int n;
            byte[] buf = new byte[MAXRECORD];
            for (;;) {
                n = is.read(buf);
                if (n == -1) break; // this TCP connection is closed
                os.write(buf, 0, n);
            }
            socket.close();
        }
        // never reached, leaves by exception
        // serverSo.close();
    }
}

```

Estes métodos permitem manipular o membro `timeout` associado ao `socket`. Este membro controla o tempo máximo que uma chamada a `read` poderá bloquear o programa. O valor por omissão é 0, que significa que programa esperará indefinidamente por dados. Se o valor for diferente de 0, e não chegarem dados até se passarem `timeout` milissegundos, a exceção `InterruptedException` será desencadeada. Estes métodos também se podem aplicar aos `ServerSockets` e controlam o tempo máximo que se espera pelo retorno da chamada `accept()`.

Outros métodos como `close()`, `getInputStream()` e `getOutputStream()`, já foram referidos. Os métodos `getPort()` e `getInetAddress` permitem conhecer a porta e o endereço IP remotos associados a um stream socket ligado à outra extremidade.

Antes de terminarmos esta parte, recordam-se a seguir várias variantes do método `write`, e outros métodos interessantes da classe `OutputStream`, que podem ser úteis no contexto da comunicação por canais TCP.

```

void write(byte[] buf)
void write(byte[] buf, int offset, int length)
void write(int data) // the low-order 8 bits
    // are written to the stream
void flush() // flushes any buffered data
void close() // terminates the stream

```

Seguem-se também alguns métodos da classe `InputStream` apresentados com o mesmo objectivo.

```

int read(byte[] buf)
int read(byte[] buf, int offset, int length)
int read() // returns next byte in the stream
    // to the least significant byte of the returned integer
void close() // terminates the stream
int available() // returns the number of bytes available

```

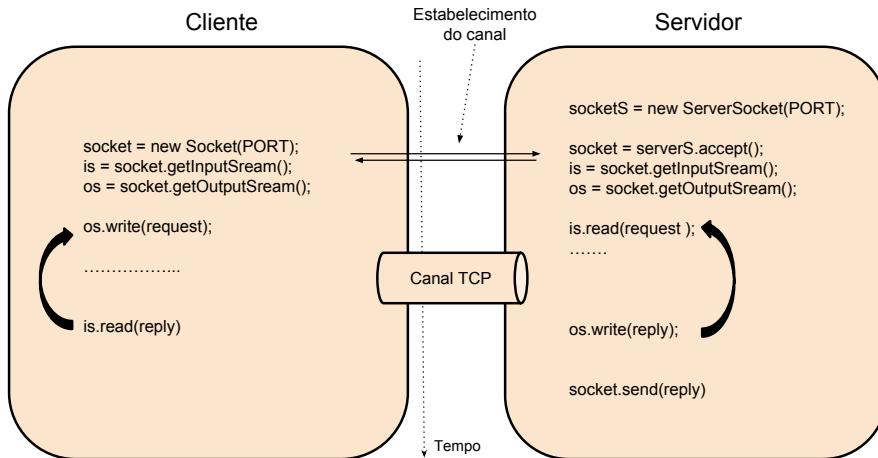


Figura 5.7: Comunicação entre um cliente e um servidor através de um canal TCP

As duas primeiras formas do método `read` colocam os bytes lidos no parâmetro `buf` e retornam o número de bytes lidos. Se esse valor for `-1`, significa que o canal está fechado. O mesmo se aplica no caso do método `read` sem parâmetros, o último caso deste método. O método `available` permite saber o número de bytes disponíveis localmente para leitura imediata, pois o valor retornado não toma em consideração os bytes que ainda não foram entregues pelo protocolo TCP.

Servidores concorrentes

O servidor `TCPEchoServer` da listagem 5.14 serve um cliente de cada vez, e só passa ao seguinte depois deste ter fechado o seu socket. Pelo facto de só servir um cliente de cada vez, um servidor deste tipo diz-se iterativo. Seria fácil fazer um servidor `TCPEchoServer` que depois de aceitar a conexão de cada cliente, ecoasse uma única mensagem, e a fechasse imediatamente, para poder servir o cliente seguinte. Apesar de iterativo, esse servidor não faria os diferentes clientes esperarem muito para obterem as respostas.

No entanto, quando um servidor demora um tempo mais significativo a executar o serviço solicitado, antes de poder responder ao cliente (por exemplo, porque depende de outros servidores ou da leitura de dados do disco) a qualidade de serviço prestada será má, pois o servidor passará muito tempo bloqueado, sem realizar nenhum trabalho útil, e sem poder atender outros clientes.

É possível melhorar esse servidor passando-o a concorrente, *i.e.*, um servidor que serve vários clientes simultaneamente. Para realizar um servidor concorrente em Java, usam-se `threads`. A ideia de base é simples: sempre que o servidor aceita um pedido novo de um cliente, lança um `thread` para o tratar. Desta forma, diferentes clientes serão atendidos simultaneamente. Se o servidor executar numa máquina multiprocessadora, a execução de cada pedido será executada realmente em paralelo. Se não, o paralelismo será “virtual”, situação designada por execução concorrente com *interleaving*, mas os ganhos de tempo de execução serão reais em ambos os casos.

Um **servidor iterativo (iterative server)** é um servidor que aceita um pedido de um cliente de cada vez, e que só aceita o pedido seguinte, depois de ter respondido ao pedido anterior. Ao contrário, um **servidor concorrente (concurrent server)**, aceita vários pedidos simultaneamente, respondendo a cada um conforme vai acabando o seu tratamento.

Listing 5.15: Servidor Eco concorrente em TCP

```

1 import java.net.*;
2 import java.io.*;
3 import java.util.*;
4
5 public class TCPEchoServer {
6
7     private static final int PORT = 8000;
8
9     public static void main(String[] args) throws Exception {
10         Logger log = new Logger("log.txt");
11         ServerSocket serverSo = new ServerSocket(PORT);
12         for (;;) { // loop for ever
13             Socket socket = serverSo.accept(); // a client connected
14             ClientHandler thread = new ClientHandler(socket,log);
15             (new Thread(thread)).start();
16         }
17         // never reached, leaves by exception
18     }
19 }
```

A utilização de servidores iterativos não penaliza o desempenho quando a computação do serviço é muito rápida (eco, consulta da hora, consulta do DNS, ...). Caso contrário, sobretudo em casos em que a execução de um pedido envolve espera por respostas suplementares, é necessário utilizar concorrência no servidor.

Em Java, um servidor concorrente pode utilizar diferentes **threads** para tratar os diferentes pedidos e, se necessário, os mesmos têm de se sincronizar entre si, para evitar resultados inconsistentes.

Em Java um **thread** pode ser construído de diversas maneiras. No exemplo ilustrado na listagem 5.16 é usada a forma que consiste em criar uma classe que implementa a interface **Runnable**, a qual tem um método público com a assinatura **void run()**, cuja execução corresponde à actividade do **thread** propriamente dito. Após a sua criação (através de **new**), o objecto **thread** permanece inactivo. A sua execução concorrente só começa após a invocação do método herdado **start()**, a qual desencadeia a execução do código correspondente ao método **run()**.

Dado que o servidor concorrente TCP é razoavelmente genérico, ver a listagem 5.15, vamos começar por apresentá-lo.

O servidor começa por criar um objecto **Logger** na linha 10. Este objecto, apresentado na listagem 5.17, vai permitir aos diferentes **threads** registarem no mesmo ficheiro a sua actividade, sem correrem o risco, como será explicado a seguir, que as suas escritas concorrentes possam resultar num ficheiro de registo de actividade confuso. A seguir, na linha 9, o servidor cria o socket de atendimento dos pedidos de conexão dos diferentes clientes, e entra no ciclo de atendimento dos mesmos. Este ciclo vai desde a linha 13 à 16.

O ciclo de atendimento dos clientes começa pela execução do método **accept** sobre o socket do servidor, na linha 13. Sempre que tem lugar uma nova conexão, é criado um novo **thread** através da criação de um objecto **ClientHandler**³, na linha 14, que recebe em parâmetro o socket do novo cliente e o objecto **Logger**. Em seguida, na linha 15, o novo **thread** inicia a sua actividade e vai servir o seu cliente. É pressuposto que no fim do tratamento do cliente o seu socket seja fechado, o que fica a cargo do seu **handler**. O servidor irá então atender o cliente seguinte.

³ClientHandler poderia ser traduzido por criado, empregado ou tratador do cliente.

Listing 5.16: Thread de processamento de um pedido de um cliente

```

1 import java.net.*;
2 import java.util.*;
3 import java.io.*;
4
5 public class ClientHandler implements Runnable {
6
7     private Socket socket; // client socket
8     private Logger log; // for logging purposes
9     private String myName;
10
11    public ClientHandler(Socket s, Logger log) {
12        this.socket = s;
13        this.log = log;
14    }
15
16    public void run() {
17        myName = Thread.currentThread().getName() + " ";
18        log.writeEntry(myName + new Date().toString() + " starting");
19        try {
20            String request =
21                new Scanner(socket.getInputStream()).nextLine();
22            log.writeEntry(myName + new Date().toString() +
23                " received request: " + request);
24            // waits 2 seconds to simulate a complex request
25            try { Thread.sleep(2000); } catch (InterruptedException e) { }
26            String reply = request + "\n";
27            OutputStream out = socket.getOutputStream();
28            out.write(reply.getBytes());
29            log.writeEntry(myName + new Date().toString()
30                +" reply sent\n");
31            socket.close();
32        } catch (Exception e) {}
33    }
34 }
35

```

Desta forma, o servidor TCP concorrente fica razoavelmente genérico e poderá ser adaptado a outros serviços, através da modificação do método `run` da classe `ClientHandler`.

No exemplo de `ClientHandler` apresentado na listagem 5.16, a classe implementa a interface `Runnable`, para pôr em evidência que pode ser lançada como um `thread`, invocando `start()` ao invés de `run()`. O construtor da classe, nas linhas 11 a 14, serve para inicializar os membros que permitem ao método `run` ter acesso ao socket do cliente e ao objecto `Logger`. O método `run` começa por inicializar o seu nome na linha 17 com o nome do `thread` que acabou de nascer. Desta forma, no ficheiro de registo de actividade (o `log`), será possível reconhecer a actividade de cada `thread` distinto. A hora de início do `thread` é registada na linha 18. Em seguida, através do socket do cliente, é lida uma sequência de bytes terminada na marca de fim de linha, pois o serviço usado para efeitos de ilustração é o serviço TCPEcho, *i.e.*, o delimitador de mensagens entre o cliente e o servidor é a marca de fim de linha. A linha enviada pelo cliente é colocada na variável `request`, nas linhas 20 e 21, e tal facto é registado no ficheiro de registo de actividade, nas linhas 22 e 23. Em seguida, para se simular que o serviço tem um tempo de execução elevado, a execução é suspensa por algum tempo, na linha 25. Finalmente, a resposta é enviada ao cliente, linha 28, tal facto é registado, linha 29, e o socket do cliente é fechado na linha 31.

A classe `Logger` põe à disposição de todos os `threads` um mecanismo de *logging*, permitindo-lhes registarem a sua actividade num ficheiro, através da escrita de mensagens no mesmo. O construtor da classe abre o ficheiro de *logging* (criando-o caso este não exista), na linha 10, e todos os `threads` podem posteriormente, de forma

Listing 5.17: Classe Logger

```

1 import java.io.*;
2 import java.util.*;
3
4 class Logger {
5
6     private PrintStream f;
7
8     Logger(String logFileName) {
9         try {
10         f = new PrintStream(logFileName);
11         String message = "Logging started at " + new Date().toString();
12         f.println(message + "\n");
13     } catch (IOException e) {
14         System.err.println("Can't open logfile " + logFileName);
15         System.exit(0);
16     }
17 }
18
19 public synchronized void writeEntry(String s) {
20     f.println(s);
21     f.flush();
22 }
23 }
```

concorrente, escrever as suas mensagens, usando para tal o método `writeEntry`. Este método `writeEntry` é sincronizado, ver a linha 19, o que tem por resultado garantir a realização de cada escrita sem atropelos. Trata-se de um mero exemplo pois na prática a escrita de linhas de texto de pequena dimensão num ficheiro é feita de uma só vez e sem interrupções pela maioria dos sistemas de operação.

Com este exemplo termina esta breve introdução aos servidores concorrentes em TCP. É relativamente fácil adaptar o exemplo para o caso em que o protocolo entre o cliente e o servidor se baseasse em UDP. Por outro lado, como foi posto em evidência com a classe `Logger`, a concorrência entre os diferentes `threads`, que representam cada um dos clientes, pode introduzir problemas de concorrência sobre objectos partilhados, que em Java se podem resolver através dos mecanismos de sincronização disponíveis.

Este tópico termina esta breve introdução à comunicação com sockets em Java.

5.4 Resumo e referências

Resumo

Neste capítulo foi apresentada uma introdução à comunicação com sockets em Java, ao nível da interface de transporte. Os principais assuntos tratados foram a comunicação com sockets UDP, a comunicação com IP Multicasting, os sockets TCP e os servidores concorrentes.

Os sockets UDP são *tomadas de rede para comunicação* dos programas, que permitem a comunicação através de datagramas UDP. Essa comunicação tem lugar entre programas no mesmo computador, ou em computadores distintos ligados via a rede. Um socket UDP é caracterizado por um endereço IP e uma porta e pertence necessariamente a um programa em execução no computador a que o socket pertence. Para poder enviar um datagrama para outro socket, é necessário conhecer o seu endereço IP e porta. Quando um programa recebe um datagrama, tem acesso ao endereço IP e porta do socket emissor do mesmo.

Um grupo IP Multicasting é um grupo de interfaces rede (e implicitamente dos seus computadores) que têm um endereço IP comum, dito endereço IP Multicasting

do grupo. Se um pacote IP for enviado para o endereço do grupo, uma cópia do pacote é entregue a todos os computadores acessíveis que fazem parte do grupo. A semântica é equivalente à do nível rede, *i.e.*, de melhor esforço.

Para aderir, ou enviar um datagrama para um grupo, basta conhecer o seu endereço. Aderir e sair de um grupo são acções que só podem ser executadas por programas com acesso aos sockets que entram ou saem do grupo.

Os sockets TCP, também designados stream sockets, são *tomadas de rede para comunicação* que, quando conectados a outro stream socket, permitem a comunicação através de um canal TCP. Este canal é virtual e caracterizado por permitir comunicação fiável, bidireccional, transmitindo sequências de bytes, sem noção de mensagens ou outros delimitadores, entre os dois stream sockets da extremidade. Os endereços e portas destes dois stream sockets caracterizam o canal TCP. Sem estar conectado, um stream socket só pode ser usado para estabelecer conexões TCP e não para comunicar dados.

Um servidor iterativo é um servidor que aceita um pedido de um cliente de cada vez, e que só aceita o pedido seguinte depois de ter respondido ao pedido anterior. Em contraste, um servidor concorrente aceita vários pedidos simultaneamente, respondendo a cada um conforme vai acabando o seu tratamento. A utilização de servidores iterativos não penaliza o desempenho quando a computação do serviço é muito rápida (eco, consulta da hora, consulta do DNS, ...). Caso contrário, sobretudo em casos em que a execução de um pedido envolve espera por respostas suplementares, é necessário utilizar concorrência no servidor.

Em Java, um servidor concorrente pode utilizar diferentes **threads** para tratar os diferentes pedidos e, se necessário, os mesmos têm de se sincronizar entre si, para evitar resultados inconsistentes.

Os principais termos introduzidos ou referidos neste capítulo são a seguir passados em revista. Entre parêntesis figuram as traduções mais comuns em língua inglesa.

Tomada de comunicação (*communication endpoints*) Ligação lógica à rede, identificada por um endereço IP e uma porta, pela qual um programa num computador pode receber e enviar dados para a rede.

Socket UDP (*UDP socket*) Tomada de comunicação orientada ao envio e recepção de *datagramas*/mensagens UDP.

Endereço IP (*IP address*) Sequência de 32 ou 128 bits (IPv4 ou IPv6, respectivamente), identificada ao nível dos programas como símbolos, que identifica na rede uma interface de comunicação de um computador.

Grupo IP Multicast (*IP Multicast group*) Grupo de interfaces rede (e implicitamente dos seus computadores) que têm um endereço IP comum, dito endereço IP Multicasting do grupo. Se um pacote IP for enviado para o endereço do grupo, uma cópia do pacote é entregue a todos os computadores acessíveis que fazem parte do grupo.

Socket TCP (*TCP socket*) Tomada de comunicação orientada ao envio e recepção de um fluxo de bytes de forma ordenada e fiável através de uma conexão TCP.

Servidor iterativo (*iterative server*) Servidor que aceita um pedido de um cliente de cada vez, e que só aceita o pedido seguinte, depois de ter respondido ao pedido anterior.

Servidor concorrente (*concurrent server*) Servidor que aceita vários pedidos simultaneamente, respondendo a cada um conforme vai acabando o seu tratamento.

Processo leve (*thread*) Programa composto por vários processos leves internos, geralmente chamados “fios de execução” autónomos, que executam em concorrência real ou simulada, e que partilham os recursos (memória, sockets, ficheiros, etc.) do seu programa. A utilização de processos leves é a forma mais simples de implementar um servidor concorrente.

Referências

Existem inúmeros livros sobre a programação de aplicações usando a interface de sockets em geral, e em Java em particular. O livro [Harold, 2013] é uma referência clássica, muito completa, que cobre todos os detalhes sobre o assunto. Interessa sobretudo a um profissional que tenha necessidade de trabalhar com sockets em Java. O livro [Calvert and Donahoo, 2011] é um guia prático para uma primeira abordagem ao tema, cobrindo apenas os aspectos essenciais. A sua cobertura da parte de representação de dados está um pouco desactualizada, mas todo o restante livro responde bem às necessidades de um estudante da comunicação com sockets em Java.

Apontadores para informação na Web

- <http://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html> – Contém a documentação oficial do package `java.net` na sua versão mais recente no início de 2017.
- <https://docs.oracle.com/javase/tutorial/networking/sockets/> – Contém um tutorial oficial sobre a programação em Java de aplicações que usam a rede.
- <http://www.oracle.com/technetwork/java/javase/java-tutorial-downloads-2005894.html> – Contém vários eBooks sobre Java, entre os quais alguns que cobrem o tema da programação de aplicações distribuídas com Java.

5.5 Questões para revisão e estudo

1. Modifique o servidor Eco da listagem 5.6 para não responder ao cliente. O que acontece ao cliente da listagem 5.5 na versão sem *timeouts*?
2. Modifique o servidor Eco da listagem 5.6 para só aceitar pequenas mensagens, por exemplo de no máximo 10 bytes. Verifique o que se passa com o cliente e o servidor quando são enviadas mensagens contendo mais do que 10 bytes.
3. Modifique o cliente Eco da listagem 5.5 para verificar se a resposta vem do servidor Eco.
4. Modifique o servidor Eco da listagem 5.6 para mostrar a porta e o endereço IP do cliente.
5. Modifique o cliente Eco da listagem 5.7 de forma a que ele passe a usar uma porta de origem diferente para cada pedido, pois dessa forma garante-se que um datagrama atrasado não é confundido com a resposta ao pedido seguinte.
6. Sugira uma solução mais eficaz para o problema evocado na questão precedente.
7. A listagem 5.18 apresenta um programa (`FileCp`) que realiza a cópia bloco a bloco de um ficheiro origem para um ficheiro destino.

Use-a para realizar um cliente e um servidor que transferem ficheiros por UDP. O protocolo entre o cliente e o servidor pode ser ingênuo e não corrigir a perda de datagramas, nem a chegada de datagramas UDP fora de ordem. O cliente deve emitir em sequência, sem esperar por qualquer resposta, os blocos do ficheiro em datagramas, dirigidos a uma socket UDP do servidor. Verifique se o ficheiro recebido tem a mesma dimensão do ficheiro emitido; se não tiver, é provável que se tenham perdido datagramas entre o cliente e o servidor. Pode tentar melhorar a fiabilidade da transferência fazendo o emissor esperar um pouco entre cada envio de um datagrama. No entanto, mesmo que o resultado seja melhor, o tempo de transferência aumenta muito.

Listing 5.18: Programa de cópia de ficheiros

```

import java.io.*;

public class FileCp {

    static final int BLOCKSIZE = 1024; // block buffer size

    public static void main(String[] args) {
        // reading arguments
        if (args.length != 2) {
            System.err.println("usage:java FileCp fromfile tofile");
            System.exit(0);
        }
        String fromFile = args[0];
        String toFile = args[1];
        // does file exist and is readable ?
        File f = new File(fromFile);
        if (f.exists() && f.canRead()) {
            System.out.println("file:" + fromFile + " ok to copy");
        } else {
            System.err.println("Can't open fromfile" + fromFile);
            System.exit(0);
        }
        try {
            long time = System.currentTimeMillis();
            FileInputStream in = new FileInputStream (fromFile);
            FileOutputStream out = new FileOutputStream (toFile);
            byte[] fileBuffer = new byte[BLOCKSIZE];
            boolean finished = false;
            int byteCount = 0;
            int blockCount = 0;
            int n;
            do {
                n = in.read(fileBuffer);
                if (n == -1) n = 0;
                if (n < BLOCKSIZE) finished=true; // last block
                // if (seconds > 0) Thread.sleep(ms);
                if (n > 0) out.write(fileBuffer, 0, n);
                byteCount += n;
                blockCount += 1;
            } while (!finished);
            in.close();
            out.close();
            // compute time spent copying bytes
            time = System.currentTimeMillis() - time;
            long speed = 1000 * 8 * Math.round(byteCount / time);
            System.out.printf(blockCount + " blocks," +
                byteCount + " bytes copied in " +
                time + " ms, at " + speed + " bps");
        } catch (Exception e) {
            System.err.printf("Can't copy file\n");
            System.exit(0);
        }
    }
}

```

8. Use sockets multicasting para fazer um programa elementar de *Chatting*. O programa que suporta um utilizador do sistema tem dois **threads**. Um deles subscreve o grupo Multicast acordado, numa porta acordada, e mostra as mensagens recebidas dos outros participantes. O outro lê da consola as mensagens a enviar pelo utilizador aos outros participantes e envia-as para o grupo. Cada mensagem só tem uma linha. A identificação de cada participante pode ser passada em parâmetro do programa. Este programa pode ser melhorado e expandido com novas funcionalidades quase sem limite, mas nesta versão pretende-se apenas exercitar a comunicação com sockets multicast.
9. Que acontece ao cliente da listagem 5.12 se o servidor da listagem 5.14 deixar de chamar `accept`?
10. Que acontece ao cliente `TCPEchoClient` da listagem 5.12 se o mesmo enviar dados para o socket stream, mas este ainda não estiver ligado ao servidor, pois este ainda não chamou `accept`?
11. Que acontece ao cliente da listagem 5.12, caso o servidor da listagem 5.14 ainda não tenha sido lançado?
12. Modifique o servidor da listagem 5.14 de forma que o mesmo não sirva cada cliente mais do que $N \geq 1$ mensagens seguidas, isto é, após processar as N mensagens de um cliente, fecha a conexão e passa ao cliente seguinte.
13. Modifique o cliente da listagem 5.12 para caso o servidor feche a conexão, o cliente abra uma outra e recomece.
14. Faça uma versão concorrente do servidor da listagem 5.14.
15. A listagem 5.19 apresenta um cliente (`TCPFileClient`) que recebe um ficheiro de um servidor remoto por TCP. O cliente envia para o servidor uma linha contendo o nome do ficheiro que pretende. Faça um servidor que implemente o envio de ficheiros remotos, tendo em consideração que o primeiro byte enviado pelo servidor contém um código de operação. Se o valor enviado pelo servidor nesse byte for diferente de '1', isso significa que o servidor não consegue enviar o ficheiro pedido.
16. Faça uma variante do cliente e do servidor que quando o servidor indica que não consegue enviar o ficheiro (*i.e.*, quando o primeiro byte enviado é diferente de 1), ao invés de enviar o ficheiro, o servidor envia uma mensagem numa linha a explicar o problema.
17. Faça variar o valor de `BLOCKSIZE` no cliente e no servidor e verifique se o mesmo influencia a velocidade de transferência.
18. Faça uma versão concorrente do servidor. O servidor regista no ficheiro de registos os ficheiros transferidos, a sua dimensão e o tempo que demorou a enviá-los.

Listing 5.19: Cliente para cópia de um ficheiro por TCP

```

import java.net.*;
import java.io.*;

public class TCPFileClient {

    static final int BLOCKSIZE = 2048; // block transfer size
    static final int PORT = 8000;

    public static void main(String[] args) throws Exception {
        // reading arguments
        if (args.length != 2) {
            System.err.println("usage: java FileCopyServer file");
            System.exit(0);
        }
        try {
            Socket socket = new Socket(args[0], PORT);
            InputStream in = socket.getInputStream();
            OutputStream out = socket.getOutputStream();
            FileOutputStream fout = new FileOutputStream(args[1]);
            String request = args[1] + "\n";
            out.write(request.getBytes());
            int diag = in.read();
            if (diag != 1) {
                System.err.printf("Server doesn't know file\n");
                System.exit(0);
            }
            long time = System.currentTimeMillis();
            byte[] fileBuffer = new byte[BLOCKSIZE];
            int byteCount = 0;
            int n;
            for (;;) {
                n = in.read(fileBuffer);
                if (n == -1) break; // no more bytes
                fout.write(fileBuffer, 0, n);
                byteCount += n;
            }
            // compute time spent copying bytes
            time = System.currentTimeMillis() - time;
            long speed = 1000 * 8 * Math.round(byteCount/time);
            System.out.printf(byteCount + "bytes copied in " +
                time +"ms, at " + speed + "bps");
            fout.close();
            socket.close();
        } catch (Exception e) {
            System.err.printf("Can't copy file\n");
            System.exit(0);
        }
    }
}

```


Parte II

Transferência de dados

As aplicações distribuídas envolvem geralmente a transmissão de dados entre emissores e receptores, e por isso esta funcionalidade é fundamental nas redes de computadores. Na verdade, no modelo de camadas existe um nível inteiro dedicado a este objectivo, o nível de transporte. A importância de se discutirem diversas técnicas de transferência de dados tem por origem o facto de que quer os canais, quer a rede em geral, funcionarem com base na transferência de pacotes e, pelas mais diversas razões, estes podem não chegar ao seu destino, ou chegarem com erros, ou fora de ordem.

Apesar de ser bastante mais raro, os dados de uma mensagem podem ser corrompidos para além do seu transporte pela rede, como por exemplo na memória de um computador. As aplicações que não suportam este tipo de erros utilizam técnicas especiais para os detectar e corrigir mas a sua discussão ultrapassaria o âmbito deste capítulo. Neste capítulo vamos ver como é possível lidar com os erros com origem na rede.

Através das técnicas de detecção de erros discutidas na secção 2.4 é possível detectar se existem ou não diferenças de conteúdo entre a emissão e a recepção de um pacote. Apesar de existirem aplicações que podem conviver com pacotes com alguns erros, a prática mais frequente é suprimi-los antes de estes serem entregues ao destino. Por esta razão, raramente as aplicações recebem dados com erros ou, para efeitos práticos, a grande maioria das aplicações considera que a probabilidade de isso acontecer é, na prática, desprezável.

Assim, os erros considerados no contexto do transporte de pacotes de dados são as falhas de omissão, caracterizadas por os pacotes não chegarem ao destino, e ainda os pacotes entregues fora de ordem. Se estamos perante aplicações que não suportam essas falhas, é necessário compensá-las obtendo os pacotes que faltam e colocando-os na ordem adequada.

A necessidade de mascarar estas falhas pode aparecer a quase todos os níveis. Se um dado canal tem uma taxa de erros muito elevada, pode ser importante tentar corrigir logo a esse nível esses erros. Como ao nível de transporte existem protocolos de transferência fiável de dados, é obrigatória a correcção destas falhas por parte desses protocolos. Finalmente, quando uma aplicação se baseia em protocolos de transporte não fiável, e necessita de compensar estas falhas, o problema também se coloca. Teoricamente, o nível rede poderia também tentar corrigir este tipo de erros mas, devido ao princípio de privilegiar os extremos, essa opção não é geralmente utilizada.

Assim, nesta parte, logo no Capítulo 6, começaremos por discutir técnicas de compensação da ausência de pacotes baseadas na retransmissão dos pacotes em falta e técnicas de reordenação dos pacotes recebidos fora de ordem. No capítulo a seguir, o Capítulo 7, é discutida a utilização dessas técnicas no quadro do protocolo TCP.

As omissões de pacotes têm duas origens distintas: erros de transmissão nos canais e ritmos de chegada de novos pacotes que os comutadores não comportam. Sempre que pacotes são desprezados pelos comutadores por falta de espaço nas filas de espera, a rede está a desperdiçar recursos, *i.e.*, a capacidade de transmissão que esses pacotes já consumiram. Por outro lado, filas de espera muito grandes aumentam o tempo de trânsito, podendo levar a retransmissões inúteis, outro desperdício. O Capítulo 8 analisa formas de lidar com estes problemas e diminuir a probabilidade de se perderem demasiados pacotes por saturação da rede, *i.e.*, por se estar a utilizá-la acima das sua capacidade. Boa parte deste capítulo é dedicada ao estudo da forma como o protocolo TCP tenta encontrar um ritmo de emissão adequado, *i.e.*, um ritmo que minimize a perda de pacotes por saturação dos comutadores.

Em certos contextos é possível lidar com informação incompleta ou aproximada, tentando inferir a parte que falta. Os seres humanos são particularmente hábeis a recompor o significado de textos, sons ou imagens com pequenos erros por exemplo. Este facto pode ser aproveitado em certas aplicações que usam transferência de informação parcialmente errada, mas que recompõem, de forma aproximada, a parte

que falta. O Capítulo 9 apresenta algumas das técnicas usadas para compensação parcial de erros de omissão de pacotes. Estas técnicas são frequentemente usadas por algumas aplicações multimédia que conseguem lidar com informação incompleta ou de resolução inferior.

Finalmente, o Capítulo 10 apresenta uma breve referência a um conjunto de protocolos de transporte menos populares, ou emergentes, que apresentam formas alternativas de transferência de dados de forma fiável.

Capítulo 6

Fiabilidade com base em retransmissão

All things are difficult before they are easy.

– Autor: *Thomas Fuller*

Um fluxo de pacotes entre dois nós da rede é uma sequência de pacotes que está a ser transmitida do nó emissor para o nó receptor. O fluxo é fiável se todos os pacotes emitidos chegarem ao receptor, sem alterações e na ordem adequada. As técnicas de detecção de erros permitem rejeitar os pacotes que, para efeitos práticos, foram alterados entre a emissão e a recepção. Por outro lado, admite-se que a rede possa, por várias razões, perder pacotes, ou trocar a ordem da sua entrega. Assim, para garantir a fiabilidade do fluxo, é necessário que o receptor receba uma cópia sem erros de todos os pacotes emitidos e que consiga colocá-los na ordem adequada.

Neste capítulo vamos estudar formas de implementar a fiabilidade de um fluxo com base na detecção de quais os pacotes em falta, e de formas de levar o emissor a retransmitir uma nova cópia dos mesmos.

As hipóteses de trabalho são as seguintes. A rede é uma rede de comutação de pacotes sem estado, ver a Secção 4.1, que pode perder pacotes, e entregar pacotes ao destinatário por ordem diferente da sua emissão. O receptor recebe pacotes sem erros porque a interface dos canais rejeita os pacotes alterados no seu trajecto pelos mesmos. Os nós da rede não corrompem os pacotes entre a recepção e a emissão ou, se o fizerem, existem mecanismos de detecção, implementados entre o emissor e o receptor, que permitem identificar e rejeitar os pacotes alterados. Os canais e os nós da rede podem perder pacotes, mas não os perdem a todos, *i.e.*, mais tarde ou mais cedo alguns pacotes chegarão ao receptor. Sem esta última hipótese o problema não tem solução. Os pacotes com informação fluem do emissor para o receptor, mas pacotes com informação de controlo podem fluir do receptor para o emissor. Finalmente, os métodos usados devem tentar maximizar o débito extremo a extremo, ver 3.6.

Os protocolos a desenvolver podem ser usados no nível canal, para compensar os erros num canal com taxa de erros muito elevada, ou nos níveis transporte ou aplicação, para implementar fiabilidade na transmissão de dados. No primeiro caso não é necessário considerar a hipótese de que os pacotes podem chegar fora de ordem, pois, por hipótese, um canal não troca a ordem dos pacotes pois não contém *buffers* nem proporciona caminhos alternativos. Em ambos os casos é necessário considerar que é possível a comunicação nos dois sentidos, mesmo que sujeita a perda de pacotes.

A terminologia a usar para designar as unidades de informação transmitidas pelos protocolos, ver 4.4, poderia ser *frames*, pacotes, segmentos, blocos de dados ou

mensagens, dependendo do nível a que o protocolo é usado. Por comodidade, a seguir usaremos o termo pacotes, excepto quando o mesmo pode provocar confusão.

Para evitar introduzir complexidade inútil, vamos também começar por tentar encontrar soluções simples para o problema, e só adoptar soluções mais complexas se tal se revelar estritamente necessário.

6.1 Mecanismos de base

A solução mais simples consiste em o emissor emitir os pacotes em sequência à velocidade máxima que o canal o permitir. Essa solução tem dois problemas, ambos ilustrados na Figura 6.1. Por um lado não garante que todos os pacotes cheguem ao destino, e por outro, os pacotes podem chegar com um tal ritmo que o receptor não tenha tempo de os tratar em tempo útil. Nesta última situação, o receptor ficará numa situação equivalente à da rejeição de pacotes com erros pois não poderá processar todos os pacotes recebidos.

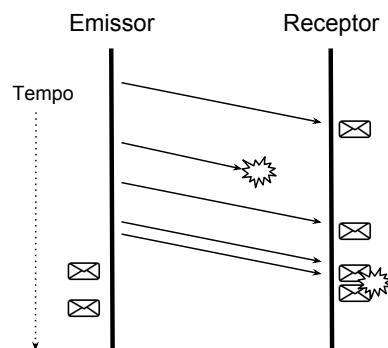


Figura 6.1: Diagrama temporal da transmissão de pacotes com informação do emissor para o receptor, com ilustração da perda de um pacote e da emissão de outro demasiado cedo

A segunda situação requer um mecanismo para adaptar a velocidade de emissão ao ritmo de processamento dos pacotes pelo receptor. Estes mecanismos designam-se mecanismos de controlo do fluxo.

Um **mecanismo de controlo de fluxo (flow control)** é um mecanismo de extremo a extremo que adapta o ritmo de emissão de pacotes pelo emissor à capacidade do receptor de tratar os pacotes em tempo útil.

A solução que podemos tentar a seguir consiste em obrigar o receptor a enviar ao emissor, após ter tratado cada pacote, um pequeno pacote de controlo indicando-lhe que pode emitir o pacote seguinte. A Figura 6.2 ilustra o funcionamento do mecanismo através de um diagrama temporal.

A Figura 6.3 mostra que esta solução tem pelo menos dois problemas, ambos relacionados com a perda de pacotes. Se se perderem pacotes do emissor para o receptor, ou do receptor para o emissor, o fluxo bloqueia.

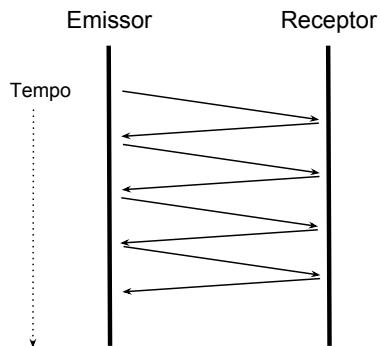


Figura 6.2: Diagrama temporal da transmissão de pacotes com informação do emissor para o receptor e de pacotes de controlo no sentido contrário

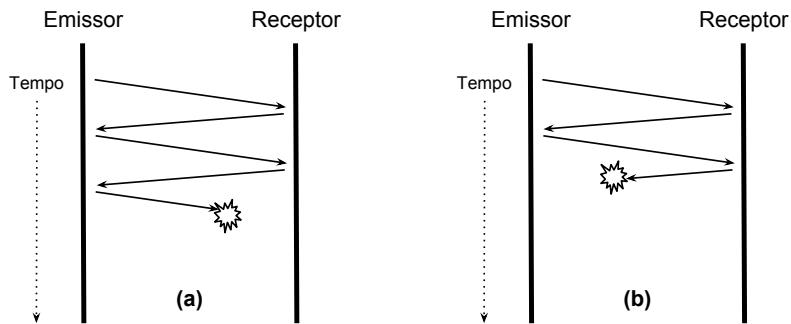


Figura 6.3: Bloqueio da transferência por (a) perda de um pacote com informação ou por (b) perda de um pacote de controlo

A solução para um problema deste tipo pode ser conseguida introduzindo um mecanismo que consiste na utilização de um **alarme temporizado (timer)**. O temporizador é inicializado com uma duração (d em segundos) e, caso não seja desarmado antes, depois de passarem d segundos sobre a sua inicialização, dispara um alarme. O momento do disparo do alarme é designado em inglês por *timeout time*, ou simplesmente *timeout*. Como o termo *timeout* é muito popular, vamos utilizá-lo frequentemente.

Assim, sempre que um pacote de dados é emitido, o emissor inicializa o temporizador com um valor adequado. Se um pacote de controlo chegar antes de o temporizador disparar um alarme, o temporizador é anulado, senão o alarme dispara e o último pacote emitido é retransmitido pelo emissor. O temporizador pode ser visto como uma espécie de relógio despertador, que é desligado pela chegada de um pacote e que toca caso nenhum chegue. A Figura 6.4 mostra como a utilização do alarme temporizado permite desbloquear o processo e fazê-lo recomeçar.

Infelizmente, ainda não temos uma solução correcta para o problema pois, como

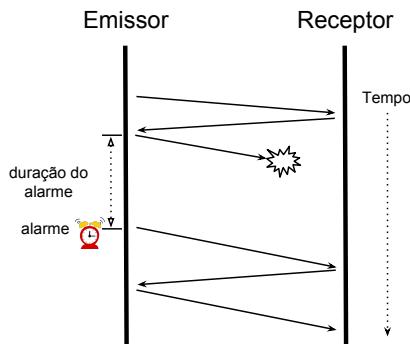


Figura 6.4: Utilização de um temporizador para desbloquear uma situação de bloqueamento por perda de pacotes

mostra a Figura 6.5 (a), a solução apresentada não trata correctamente a situação em que, ao invés de se perder o pacote com informação, o que se perde é o pacote de controlo. Nesta situação, a reemissão de pacotes de informação introduz duplicados no receptor, o que viola a correcção da solução.

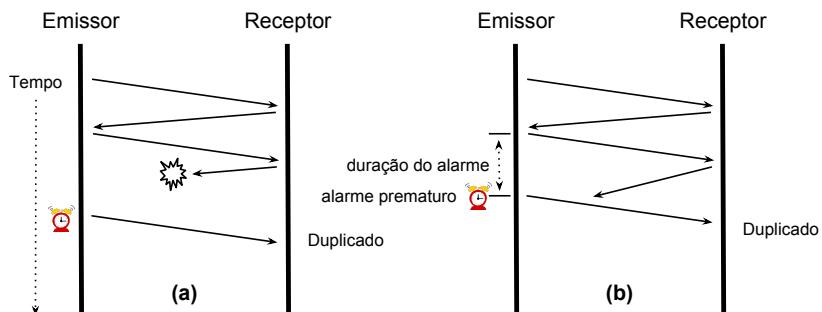


Figura 6.5: A perda de um pacote de controlo (a) ou um alarme prematuro (b) conduzem à recepção de um pacote em duplicado

Por outro lado, a introdução de um temporizador com alarme introduz um problema novo que consiste na regulação da sua duração. Com efeito, se o alarme ocorrer demasiado cedo, um pacote também pode ser emitido em duplicado, e nesse caso, o objectivo final também foi violado, ver a Figura 6.5 (b). Estas anomalias podem conduzir a situações como a ilustrada na Figura 6.6, em que o número de pacotes recebidos pelo receptor é igual ao número de pacotes emitidos, mas um foi duplicado e o outro está em falta.

A solução passa pela introdução de mais um mecanismo, que consiste na utilização de um número de sequência dos pacotes. A ideia consiste em numerar em sequência todos os pacotes, pois desta forma é possível detectar duplicados no receptor, e indicar exactamente ao emissor qual o pacote que foi de facto recebido pelo receptor. Assim,

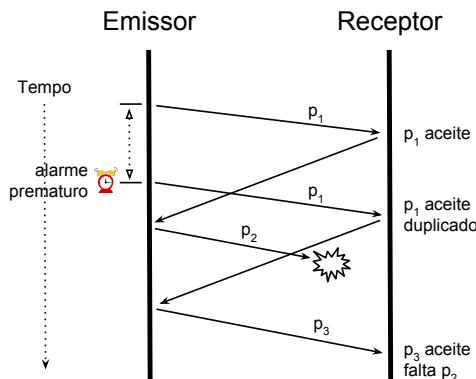


Figura 6.6: Duplicado de um pacote por o alarme ocorrer demasiado cedo, seguido da perda não detectada de um pacote

todos os pacotes emitidos contêm um cabeçalho com o número de sequência do pacote (por exemplo 1, 2, 3, ...), e os pacotes de controlo contêm o número de sequência do pacote cuja correcta recepção anunciam. Acresce, como se verá a seguir, que os números de sequência vão permitir resolver o problema de a rede poder entregar os pacotes por uma ordem diferente da com que foram emitidos.

O conjunto dos mecanismos até agora introduzidos, a saber, pacotes de dados, pacotes de controlo, temporizadores e números de sequência, são a base de um conjunto de protocolos, dos mais simples, aos mais sofisticados, que serão introduzidos a seguir. Esses protocolos baseiam-se na ideia da reemissão dos pacotes em falta e são por vezes designados por protocolos ARQ (ARQ – *Automatic Repeat ReQuest*).

Os protocolos de transferência fiável de dados em redes de pacotes, ou em canais, que se baseiam no método de retransmissão dos pacotes em falta, usam os seguintes mecanismos de base: pacotes de dados, pacotes de controlo, alarmes temporizados (*timeouts*) e números de sequência.

Para a realização destes protocolos são necessárias mensagens em que na parte de dados viajam os dados a transmitir e o cabeçalho contém a informação de controlo. A informação mínima necessária é a que consta da Figura 6.6, *i.e.*, um código de operação e um número de sequência.

O código de operação toma um dos seguintes valores: mensagem com dados, mensagem de confirmação, e também podem ser usadas mensagens de notificação de erro. A seguir usaremos as abreviaturas usadas frequentemente na literatura em língua inglesa para código de operação e que são muito populares nos protocolos normalizados.

DATA nas mensagens de dados

ACK nas mensagens de confirmação de recepção (**ACKnowledgement**)

NACK nas mensagens notificação de erro (**Negative ACKnowledgement**)

A Figura 6.8 mostra como a utilização de números de sequência nas mensagens de dados e de ACK resolve os problemas ilustrados na Figura 6.6, garantindo que o receptor recebe todos os pacotes sem faltas, e pela ordem correcta.

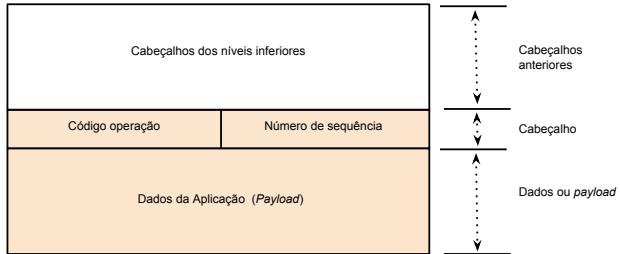


Figura 6.7: Formato das mensagens usadas pelos protocolos de transferência fiável de dados com base em retransmissão

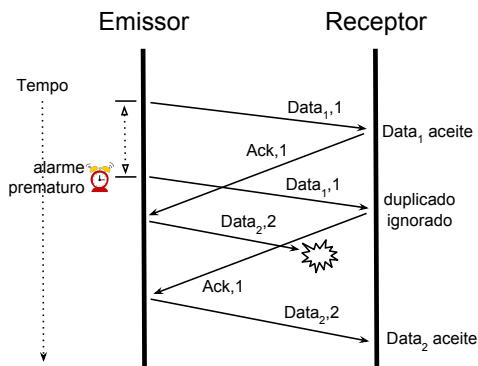


Figura 6.8: Utilização de números de sequência para garantir a correcta recepção dos pacotes

O campo com o código de operação pode ter um pequeno número de bits, um byte por exemplo parece ser suficiente. Mas quantos bits necessitamos para o campo com o número de sequência? À primeira vista poderia parecer que a dimensão em bits limitaria o número máximo de pacotes do fluxo. Por exemplo, se for usado um byte, então não podemos ter mais do 256 mensagens no fluxo. Na verdade, isso é falso, pois nada impede que se dê a volta, e se recomece a numerar as mensagens de novo com os números iniciais. Quanto maior for esse campo, maior será o desperdício, e por isso há todo o interesse em ter um campo relativamente pequeno.

No entanto, é preciso garantir que uma mensagem antiga não seja confundida com uma nova. Numa rede com comutadores que possam trocar a ordem dos pacotes, é necessário garantir que os números de sequência levam mais tempo a ser reutilizados que o tempo máximo de vida de um pacote atrasado dentro da rede. Num canal em que os pacotes são entregues por ordem, é necessário garantir que o pacote anterior não seja confundido com o que acabou de ser emitido. Por agora vamos admitir que os números de sequência usados têm o número suficiente de bits para garantir que essas condições são satisfeitas e portanto a associação entre pacotes e números de sequência é inequívoca.

Nos protocolos apresentados a seguir, por hipótese, é sempre possível fazer progresso, *i.e.*, o fluxo de dados nunca bloqueia num dado pacote que nunca é entregue ao destino com sucesso. Na prática, podem existir situações em que tal não é possí-

vel pelas mais diversas razões. Essas situações são detectadas impondo um limite ao número máximo de vezes que um pacote é retransmitido. Quando esse número é ultrapassado, o emissor desiste de tentar transmitir qualquer informação para o receptor. Esta condição excepcional de terminação não é explicitamente apresentada.

6.2 Protocolo *stop & wait*

O protocolo mais simples que assegura fiabilidade com retransmissão é designado na literatura de redes por protocolo *stop & wait*, e tem uma filosofia relativamente simples: sempre que é emitido um pacote de dados, só se pode passar ao seguinte depois de receber a confirmação da sua recepção. Caso essa confirmação não chegue, ou chegue uma indicação de que o pacote está em falta, reemite-se o mesmo pacote. O protocolo chama-se assim pois progride normalmente por um ciclo composto pela emissão de um pacote de dados, seguida de espera pela confirmação da recepção (ACK) do mesmo.

O protocolo *stop & wait* é um protocolo em que o emissor envia um pacote e só passa a enviar o pacote seguinte quando receber a confirmação da sua recepção (ACK) vinda do receptor.

A Listagem 6.1 apresenta a parte cíclica da actividade do emissor e do receptor durante a transferência dos dados.

Correcção do protocolo *stop & wait*

O protocolo é muito simples e por isso é relativamente fácil discutir a correcção do mesmo. Por um lado, o emissor repete a emissão do mesmo pacote até receber a confirmação de que o último pacote emitido foi recebido pelo receptor. Isso garante a condição: um pacote só se considera processado pelo emissor depois de este ter a certeza de que o receptor o recebeu. Por outro lado, a existência do temporizador e a ocorrência de *timeouts* quando não há recepção atempada garantem que o emissor não fica bloqueado e vai continuar a enviar o pacote corrente. Qualquer evento inesperado é ignorado. Como, por hipótese, mais tarde ou mais cedo os pacotes serão entregues, isso garante que o emissor não entra em ciclo eterno desde que o receptor retransmita os ACKs do último pacote bem recebido.

O receptor só guarda um pacote se o mesmo corresponder ao pacote que está à espera. Se tal for o caso, envia o ACK respectivo e passa a esperar pelo pacote seguinte. Senão, envia um ACK do último pacote bem recebido e continua à espera do mesmo pacote. Portanto, o receptor só confirma a recepção de um pacote quando este foi de facto recebido. O envio do ACK repetido é necessário para compensar a eventual perda do ACK mais recente. Como mais tarde ou mais cedo os dados acabarão por chegar ao receptor e os ACKs ao emissor, mesmo que seja necessária a sua retransmissão, é fácil aceitar que o protocolo vai fazendo progresso.

Ficam assim respondidas questões como: o protocolo resiste à perda de pacotes ou à sua chegada fora de ordem? A resposta é sim. Garante-se também que, mesmo nas condições mais adversas, existe uma relação inequívoca entre os números de sequência e os pacotes. De facto, por hipótese, os números de sequência têm um número de bits que garante que um número de sequência só é reutilizado depois de se garantir que o pacote que o utilizou mais recentemente já não existe.

Uma última questão diz respeito ao valor do *timeout*. Qual deve ser o seu valor? Que acontece se este está mal calculado? O valor do *timeout* tem de acomodar o RTT (tempo de trânsito entre o emissor e o receptor e vice-versa), assim como o tempo de processamento pelo receptor, que pode estar mais ou menos carregado com outras

Listing 6.1: Pseudo código da parte cíclica do protocolo *stop & wait* sem limitação do número de retransmissões.

```

Sender:
    seq = 1
    connection.create()

    while ( more application packets to send ) {
        packet = getNextPacketToSend()
        packet.opCode = DATA
        packet.sequence = seq
        // loop until the packet is acked
        while ( true ) {
            send(packet);
            startTimer( duration )

            On event < ACK received and ACK.sequence() == seq >:
                stopTimer()
                seq++
                break // leave loop

            On event < TIMEOUT >:
                // loop again
        }
    }

    connection.close()

Receiver:
    seq = 1

    while ( connection.isActive() ) {

        On event < DATA packet received and packet.sequence == seq >:
            // received the expected packet
            deliverPacketToApplication(packet)
            ACK.sequence = seq
            send(ACK)
            seq++

        On event < DATA packet received and packet.sequence != seq >:
            // ignore packet and resend the previous ACK
            ACK.sequence = seq - 1
            send(ACK)
    }
}

```

tarefas. Se o *timeout* for demasiado longo, a recuperação de erros será mais lenta mas a correcção do protocolo não será colocada em causa. Se o *timeout* for demasiado curto, serão retransmitidos pacotes inutilmente mas a correcção do protocolo também não será violada.

Acima foram introduzidos os pacotes NACK. No protocolo *stop & wait* um NACK pode ser enviado quando for recebido um pacote com erros. No entanto, nessa situação tanto um NACK como um ACK do último pacote bem recebido desarmam o temporizador no emissor e desencadeiam uma retransmissão, pelo que ambas as soluções têm o mesmo efeito.

Como já referimos, esta discussão ignora os problemas de inicializar e terminar a transferência. Um exemplo de como essas questões podem ser abordadas será apresentado aquando da discussão do protocolo TCP, ver o Capítulo 7.

Dado parecer claro que estamos perante um protocolo simples que cumpre o objectivo, o próximo passo consiste em avaliar o desempenho do mesmo.

Desempenho do protocolo *stop & wait* sem erros

O melhor desempenho do protocolo coincide com o seu funcionamento sem erros, pois a recuperação dos erros introduz desperdícios que prejudicam o desempenho. Como, no caso geral, o tempo de trânsito dos pacotes não é constante e o tempo de processamento pelo receptor também não, é habitual fazer as seguintes simplificações: 1) admite-se que o RTT médio é estável e toma-se o seu valor como uma constante; quando entre o emissor e receptor está um único canal, admite-se que RTT é igual a duas vezes o tempo de propagação; 2) admite-se que o receptor está bem dimensionado e despreza-se o tempo de processamento do mesmo; 3) admite-se que o tempo de transmissão dos pacotes de dados é constante pois os mesmos têm dimensão constante e suficientemente grande para se desprezar o tempo de transmissão dos cabeçalhos; e finalmente, 4) admite-se que os pacotes de ACK só têm cabeçalho e portanto despreza-se o seu tempo de transmissão.

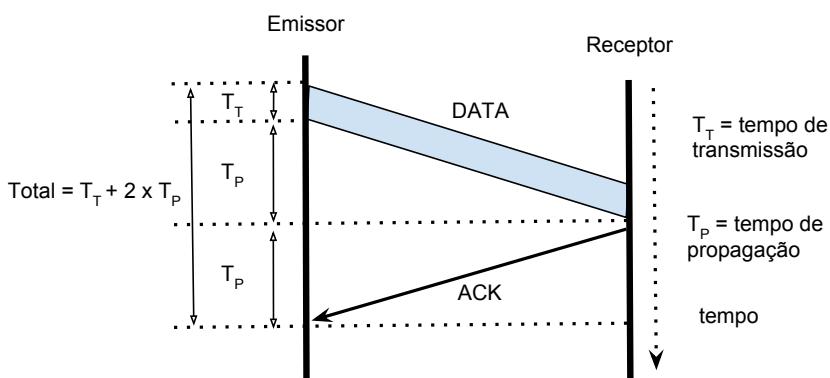


Figura 6.9: Determinação do desempenho do protocolo *stop & wait*

Com estas hipóteses, ver a Figura 6.9, o tratamento de cada pacote é repetido ciclicamente, cada iteração tem por duração $T_{total} = T_T + 2.T_P$ e durante cada uma transmite-se a quantidade de bits correspondentes a um pacote. Portanto, o débito médio extremo a extremo (V) será a dimensão em bits dos pacotes de dados (D) a

dividir por T_{total} .

$$V = \frac{D}{T_T + 2.T_P} = \frac{D}{T_T + RTT} \quad (6.1)$$

Quando o protocolo é aplicado num contexto em que existe um único canal *full-duplex* entre o emissor e o receptor, define-se a **taxa de utilização (usage ratio)** do canal como sendo a fração do tempo em que o canal se encontra a transmitir dados do emissor para o receptor. Denotando a taxa de utilização por T_U , a mesma é dada por

$$T_U = \frac{T_T}{T_T + 2.T_P} = \frac{T_T}{T_T + RTT} \quad (6.2)$$

Olhando para a Figura 6.9 é fácil reconhecer a validade da equação 6.2 e ainda como a relação entre o RTT e o T_T a influenciam. Se o valor do RTT é desprezável face ao T_T , a taxa tende para 100%, se o valor do RTT é muito maior que o T_T , a taxa aproxima-se de 0.

Assim, sempre que estamos numa situação em que o T_P é desprezável, a taxa de utilização aproxima-se de 1 e o débito extremo a extremo (V) aproxima-se do débito do canal. Sempre que o RTT é significativo e vai crescendo, a taxa tende para 0, e isso que dizer que o débito disponível do canal é cada vez menos utilizado.

Por exemplo, admitindo que os pacotes têm 10^4 bits e que o débito do canal (V_T - Velocidade de Transmissão) é 1 Mbps, $T_T = 10^4/10^6 = 10^{-2} = 10\ ms$. Admitindo adicionalmente que o RTT é de 0,1 ms, $T_U = 10/(10 + 0,1) = 99\%$. A Tabela 6.1 a seguir mostra as diversas taxas de utilização conseguidas em diversos contextos.

Tabela 6.1: Taxas de utilização obtidas pelo protocolo *stop & wait* com pacotes de dados de 10.000 bits, canal sem erros, desprezando o tempo de transmissão dos ACKs e variando o débito (V_T) e o RTT do canal

T_U	V_T (Mbps)	T_T (ms)	RTT (ms)	Caracterização
99%	1	10	0,1	V_T baixa, RTT desprezável
50%	100	0,1	0,1	V_T razoável, RTT desprezável
33%	1	10	20	V_T baixa, RTT baixo
0,5%	100	0,1	20	V_T razoável, RTT baixo
4,8%	1	10	200	V_T baixa, RTT alto
0,05%	100	0,1	200	V_T razoável, RTT alto
0,005%	1000	0,01	200	V_T alta, RTT alto

Os valores de T_U na tabela confirmam o que é evidente. O protocolo *stop & wait* não transfere mais do que um pacote por RTT. Perante um elevado RTT, caso ilustrado na parte direita da Figura 6.10, a taxa de utilização, assim como o débito médio, são muito baixos. Para se aumentar a taxa de utilização, é necessário arriscar e enviar mais pacotes, *i.e.*, enviar pacotes de avanço mesmo que não se tenha ainda a certeza de que alguns dos anteriores foram bem recebidos, como está ilustrado na parte esquerda da mesma figura.

6.3 Protocolos de janela deslizante

Existe um conjunto de protocolos do tipo ARQ, *i.e.*, com base em retransmissão dos pacotes pelo emissor, que se designam por protocolos de janela deslizante ou *pipelining*

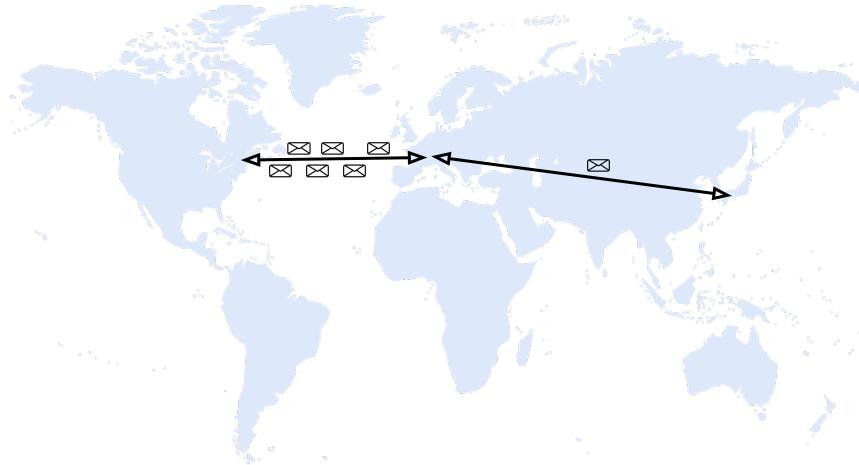


Figura 6.10: À direita dois interlocutores usam o protocolo *stop & wait*, que envia no máximo um pacote por RTT; à esquerda dois interlocutores usam um protocolo que envia mais do que um pacote por RTT

protocols. A ideia comum a estes protocolos consiste em ir enviando pacotes para a frente, mesmo sem receber ACKs dos pacotes emitidos anteriormente, na esperança de que os erros não sejam muito frequentes e com isso se aumente a velocidade de transferência extremo a extremo.

Os pacotes que já foram emitidos mas de que o emissor ainda não recebeu a confirmação de recepção, *i.e.*, que não foram ainda “acked”, dizem-se pacotes em trânsito (*in-transit or in-flight packets*). Todos estes protocolos limitam o número máximo de pacotes em trânsito em cada momento (N) por diversas razões. Primeiro por razões de controlo de fluxo, *i.e.*, de forma a que um emissor muito rápido não “afogue” um receptor lento a processar os pacotes. Outra razão tem a ver com a necessidade de não enviar mais pacotes do que aqueles que a rede consegue, no momento, encaminhar entre o emissor e o receptor sem demasiadas supressões de pacotes. Enviar pacotes a um ritmo superior ao que a rede aguenta diminui o desempenho e desperdiça recursos. Finalmente, pode ser inútil usar um valor de N demasiado grande, dependendo da taxa de erros da rede, porque o custo de corrigir um erro pode implicar a retransmissão de pacotes já em trânsito.

A Figura 6.11 mostra o caso em que $N = 3$, ou seja, o número de pacotes *in-flight* é no máximo 3. Se não ocorrerem erros, em cada ciclo de emissão de pacotes (que continua a durar $T_{total} = T_T + 2.T_P$), ao invés de se transferir um pacote para o receptor, foram transferidos $N = 3$ pacotes. O débito extremo a extremo aumentou N vezes.

Como ainda não foi confirmada a recepção dos pacotes emitidos, é necessário que o emissor mantenha uma cópia dos mesmos para os poder vir a retransmitir se necessário. Assim, o emissor tem de dispor de um *buffer* de emissão com espaço para N pacotes, o qual se designa **por janela de emissão de pacotes**. Portanto, a janela de emissão de pacotes é um *buffer* de dimensão N que contém os pacotes em trânsito. Que acontece quando a janela está cheia, *i.e.*, quando existem N pacotes em trânsito? Mais nenhum pacote pode ser emitido até chegarem ACKs dos emitidos. Se tudo estiver a correr bem, o número de sequência do primeiro ACK a chegar será o do pacote mais antigo na janela, que assim abandona a janela e esta diminui de uma unidade. Portanto, um

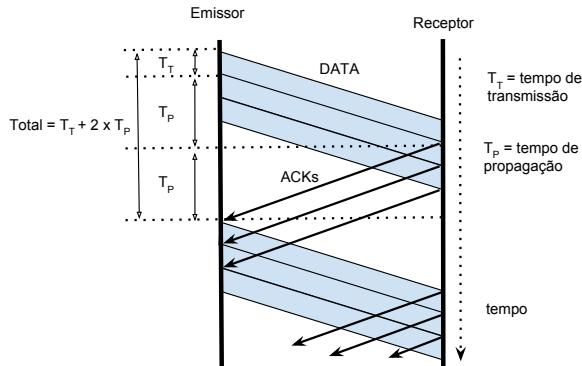


Figura 6.11: Aumento da velocidade de transferência extremo a extremo pelo aumento do número de pacotes *in-flight*

novo pacote pode ser emitido sem violar a condição: a dimensão da janela de emissão é $\leq N$, e assim sucessivamente.

Protocolos de janela deslizante (*sliding window protocols*) são protocolos em que o emissor envia vários pacotes de avanço, mesmo sem receber os ACKs referentes aos mais antigos. Os pacotes em trânsito são mantidos num *buffer* de emissão chamado a janela do emissor. A dimensão desta janela é limitada por razões de controlo de fluxo e de controlo da saturação da rede. Adicionalmente, a sua dimensão pode também ser limitada para se evitar desperdício em caso de perda de pacotes.

Torna-se agora mais claro porque este *buffer* do emissor se chama janela do emissor. Considerando os números de sequência do fluxo de pacotes que o emissor quer enviar ao receptor, podemos assimilar o *buffer* a uma janela sobre esse fluxo, ver a Figura 6.12, que contém um subconjunto contíguo de números de sequência (pacotes), correspondente aos pacotes em trânsito, ou que podem ser emitidos mantendo a condição dimensão da janela $\leq N$. Os números de sequência à esquerda da janela correspondem a pacotes de que o receptor já recebeu a confirmação de recepção (já foram “acked”). À direita da janela temos os números de sequência dos pacotes que irão ser transmitidos no “futuro”.

Qual a dimensão do *buffer* do receptor? A solução mais simples consiste em o receptor só ter espaço para um pacote. Quer isto dizer que se o pacote que o receptor receber não for exactamente o pacote de que ele estava à espera, o mesmo é desprezado. Esta primeira alternativa corresponde a um protocolo designado GBN.

Protocolo voltar atrás N (GBN - Go-back-N)

O protocolo GBN é um protocolo de janela deslizante em que a janela do emissor tem dimensão N e a janela do receptor tem (pelo menos) dimensão 1. O emissor, desde que tenha pacotes para enviar, e enquanto a condição sobre a dimensão da janela de emissão ser $\leq N$ for respeitada, emite pacotes uns atrás dos outros. O receptor, desde que receba um pacote com o número de sequência de que está à espera, aceita-o, processa-o (e.g., entrega-o à aplicação) e envia um ACK, ver a Figura 6.13.

No entanto, se o receptor receber um pacote com o número de sequência errado (i.e., um duplicado ou um pacote adiantado porque foi alterada a ordem de entrega dos

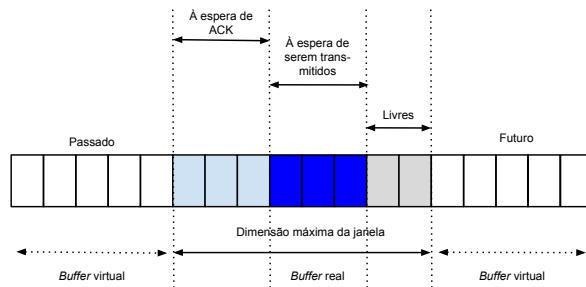


Figura 6.12: Janela do emissor nos protocolos de janela deslizante

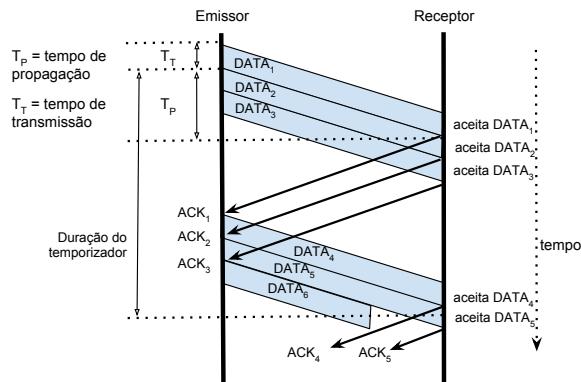


Figura 6.13: O protocolo GBN evoluindo sem erros

pacotes, ou o anterior perdeu-se) despreza-o, e retransmite um ACK correspondente ao último pacote que recebeu correctamente, ver a Figura 6.14.

Quando dispara um alarme no emissor (*timeout*), este volta a reemitir os pacotes que estão na janela, começando pelo que tem o número de sequência mais baixo, pois este é o pacote em trânsito mais antigo, e para poder voltar a colocar os pacotes pela ordem é necessário recomeçar por este e reemitir a janela. É por esta razão que o protocolo se chama GBN, porque em caso de anomalia, o emissor regressa ao pacote mais antigo (*i.e.*, recua N pacotes) e recomeça a emissão da janela.

As acções executadas pelo emissor do protocolo GBN são apresentadas a seguir na Listagem 6.2. O receptor é idêntico ao receptor do protocolo *stop & wait*.

A janela é uma fila de pacotes em que os novos pacotes a transmitir são acrescentados no fim, e o primeiro pacote é o pacote em trânsito mais antigo. O *timeout* é colocado quando é enviado o primeiro pacote da janela. Sempre que se recebe um ACK do pacote mais antigo na janela, *i.e.*, na primeira posição da fila, este deixa a janela e o valor do temporizador é ajustado. Se a janela ficou vazia, pára-se o temporizador, senão inicializa-se o temporizador com o valor *duration* - *t* em que *t* corresponde ao tempo que já decorreu desde que o novo pacote em trânsito mais antigo foi emitido. O procedimento *GoBackN* consiste em posicionar o pacote mais antigo da janela como o próximo pacote a emitir.

É possível introduzir uma optimização aquando da recepção de um ACK com

Listing 6.2: Pseudo código da parte cíclica do emissor no protocolo GBN com receptor com janela de recepção de dimensão 1 e sem limitação do número de retransmissões.

```
// "window" is a queue with maximum length N and
// with item positions from 0 to window.size()-1 if not empty
seq = 1
connection.create()

while ( more application packets to send or window.size() > 0 ) {

    On event < end of last packet transmission and window.size() == 0 >:
        // ignore it and loop again

    On event < end of last packet transmission and window.size() != 0 >:
        position = window.getNextPacketPosition() // next packet to send
        window.sendPacket(position)
        if ( position == 0 ) startTimer(duration)

    On event < ACK and ACK.sequence == window.getSequenceNumber(0) >:
        window.removeFirst()
        adjustTimer( window.getExpirationTime(0) )

    On event < ACK and ACK.sequence != window.getSequenceNumber(0) >:
        // ignore it and loop again

    On event < applicationPacketAvailable and window.size() < N >:
        packet = getNextApplicationPacket()
        packet.opCode = DATA
        packet.sequence = seq
        window.addLast(packet)
        seq++

    On event < TIMEOUT >:
        // The next packet to transmit is the one at position 0 of window
        window.GoBackN()
        next event = end of last packet transmission
}

connection.close()
```

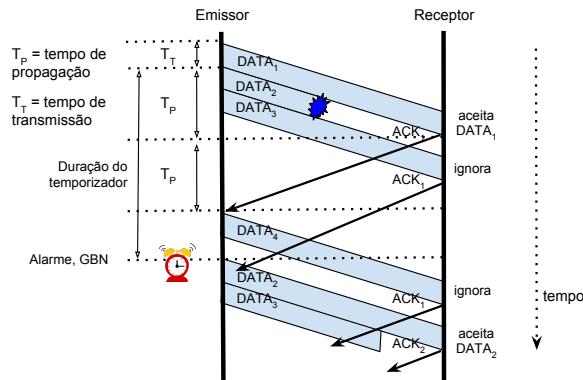


Figura 6.14: Ilustração do mecanismo GBN na sequência do disparo de um alarme

número de sequência repetido. A recepção de um ACK com o número de sequência repetido, para além de permitir compensar a perda de um ACK, permite também ao emissor aperceber-se mais cedo que houve uma anomalia e o pacote se perdeu sem necessitar de esperar pelo eventual disparo do *timeout*. Ver a Figura 6.15.

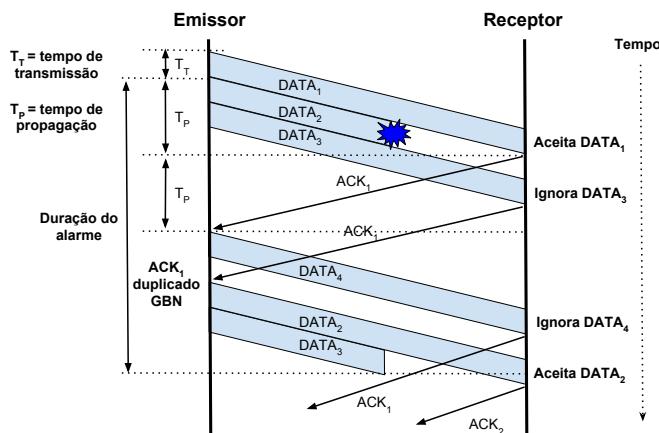


Figura 6.15: Ilustração da optimização possível na sequência de um ACK duplicado

A Figura 6.12 mostra a janela do emissor durante a evolução do protocolo GBN e a Figura 6.13 a evolução normal do protocolo sem erros.

O protocolo GBN (*Go-back-N*) é um protocolo de janela deslizante que usa uma janela de emissão maior que 1. Quando dispara um alarme (*timeout*), o emissor recomeça a enviar todos os pacotes que ainda estão em trânsito, a começar pelo mais antigo.

Protocolo Go-back-N com janela do receptor maior que 1

É relativamente fácil de imaginar uma forma de melhorar o protocolo GBN através do aumento da dimensão da janela do receptor. Nessa realização do protocolo os números de sequência dos ACKs são cumulativos, *i.e.*, um ACK com o número de sequência n significa que todos os pacotes até ao de número de sequência n foram correctamente recebidos.

Assim, quando um pacote chega fora de ordem, o receptor envia o ACK com o número de sequência do último pacote bem recebido, mas guarda o pacote se tiver lugar na janela de recepção. Assim que chegar um pacote que colmate a falta, o novo ACK, como é cumulativo, considera todos os pacotes correctamente recebidos até aí.

Por exemplo, na Figura 6.16 quando chega o pacote 1, o receptor envia um ACK 1, depois chegam os pacotes 3 e 4 e o receptor guarda-os mas envia sempre um NACK. O emissor quando recebe o NACK, decide iniciar imediatamente o processo GBN e volta a reemitir o pacote 2. Finalmente, quando o receptor recebe o pacote 2, envia um ACK 4 pois até ao momento já recebeu correctamente os pacotes 1, 2, 3 e 4.

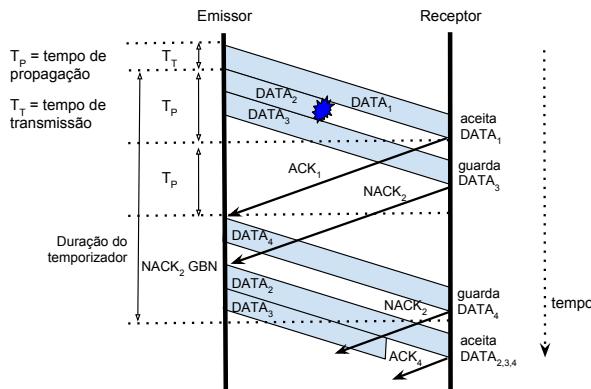


Figura 6.16: Protocolo GBN com o receptor com janela maior que 1 e a guardar os pacotes recebidos fora de ordem

A utilização de um ACK cumulativo, a utilização de uma janela do receptor maior que 1 e o desencadeamento do processo GBN sempre que é recebido um NACK ou ACKs duplicados, permite uma recuperação potencialmente mais rápida dos erros ocorridos.

Constata-se que nos casos em que o RTT é bastante significativo e a capacidade dos canais elevada, é necessário usar janelas de dimensão bastante elevadas, para que seja possível explorar adequadamente o débito extremo a extremo permitido pela rede. Nestes casos, a perda de um pacote, dado o elevado RTT e a dimensão da janela, só será recuperada depois de terem sido transmitidos muitos pacotes para a frente. O funcionamento Go-Back-N leva então à retransmissão inútil de muitos pacotes como mostra a Figura 6.17.

Existe uma outra versão do protocolo de janela deslizante em que a janela do receptor também é maior que 1, mas que permite recuperar mais rapidamente da perda de pacotes e tenta evitar o desperdício devido a retransmissões inúteis. Essa versão é designada repetição selectiva.

Antes de a apresentarmos, convém chamar a atenção para o facto de que se entre o emissor e o receptor está um único canal, este não troca a ordem dos pacotes, a

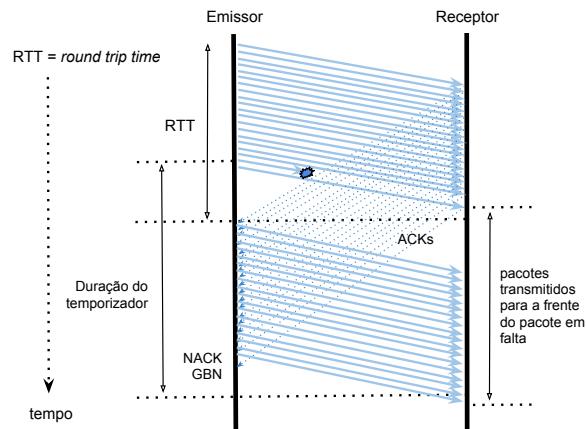


Figura 6.17: Quando a janela é muito grande, o funcionamento GBN leva à potencial retransmissão inútil de muitos pacotes que já foram transmitidos e recebidos para a frente do pacote em falta

recepção de um ACK cumulativo repetido (ou um NACK) é sinónimo da perda de um pacote e o seu reenvio imediato justifica-se. No caso de uma rede que pode trocar a ordem dos pacotes, a chegada de ACKs cumulativos (ou de NACKs) não deve ser interpretada imediatamente como equivalente a um evento *timeout*. Por exemplo, o protocolo TCP usa este tipo de mecanismos e só ao fim da recepção de 3 ACKs repetidos é que desencadeia a nova emissão.

Protocolo repetição selectiva (*Selective Repeat – SR*)

O protocolo de repetição selectiva (*Selective Repeat – SR*) usa janelas de dimensão maior que 1 no emissor e no receptor. No entanto, a diferença para o GBN está no processamento dos ACKs, pois estes deixam de ser cumulativos e passam a dizer respeito apenas a um único pacote. Um ACK do pacote *seq* significa apenas que o pacote *seq* foi bem recebido, não acrescentando nenhuma informação sobre outros pacotes. O mecanismo GBN deixa de ser usado, visto que o emissor nunca recua *N* pacotes. O emissor associa um temporizador separado a cada pacote emitido. Quando o respectivo alarme dispara, apenas o pacote ao qual está associado é retransmitido e retoma-se o processamento normal dos outros pacotes após essa retransmissão.

O objectivo do protocolo SR é evitar, tanto quanto possível, a retransmissão de pacotes que tenham sido bem recebidos. Assim, só são retransmitidos os pacotes para os quais disparou o *timeout*, ou foi recebido um NACK, e mais nenhum outro. Os outros pacotes, caso sejam recebidos os respectivos ACKs antes de o respectivo alarme disparar, não serão retransmitidos.

Existe apenas uma excepção ao caso da retransmissão, que é quando o valor do temporizador se revela curto para um dado pacote. Neste caso pode ocorrer um alarme prematuro e ser retransmitido um pacote ainda em trânsito.

Para acelerar o protocolo, sempre que o receptor recebe um pacote que cabe na janela mas está fora de sequência e abre um “buraco” na sequência de pacotes porque existem um ou mais pacotes em falta, para além de enviar um ACK do pacote corretamente recebido, o receptor pode enviar um NACK do pacote em falta, na esperança de se adiantar ao temporizador associado ao pacote no emissor e levá-lo a retransmitir o pacote tão cedo quanto possível.

O protocolo SR (*Selective Repeat protocol*) é um protocolo que usa janelas de emissão e recepção maiores que 1. O protocolo trata cada pacote em trânsito de forma independente. Quando dispara um alarme (*timeout*), o emissor apenas reenvia o pacote ao qual o alarme está associado.

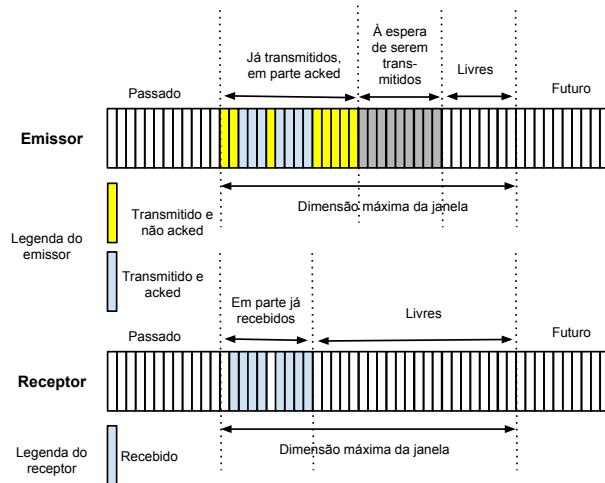


Figura 6.18: Janelas do emissor e do receptor no protocolo SR

A Figura 6.18 mostra as janelas do emissor e do receptor durante a evolução do protocolo. Em ambas as janelas, a condição de a dimensão ser $\leq N$ tem de ser respeitada. Assim, se a janela do emissor estiver cheia, este só pode aceitar mais pacotes da aplicação para transmitir quando receber o ACK do pacote em trânsito mais antigo. Por outro lado, o receptor só pode entregar à aplicação dados sem “buracos”. Se o receptor tem a janela cheia, mas com falta de pacotes “à esquerda”, pode optar por transmitir NACKs dos mesmos, para tentar desencadear a sua retransmissão tão cedo quanto possível.

A Figura 6.19 mostra um exemplo do funcionamento do protocolo que permite ver que de facto o protocolo evita retransmissões inúteis. Quando for recebido o ACK do pacote mais antigo, a janela pode deslocar-se N pacotes para a direita, visto que os pacotes seguintes estão todos acked. É, no entanto, necessário ter em atenção que o receptor pode receber pacotes antigos repetidos fora da janela, porque se perderam os respetivos ACKs. Nesse caso o receptor terá também de enviar um novo ACK dos mesmos.

O protocolo de repetição selectiva é mais complexo no seu conjunto, sobretudo no que diz respeito aos temporizadores, razão pela qual nem sempre é adoptado. Resumidamente, o seu funcionamento é o seguinte:

Apesar de o protocolo SR ser mais complexo que os protocolos anteriores, sobre tudo no que diz respeito à gestão dos alarmes, é relativamente fácil convencermos-nos da sua correcção pois cada pacote é tratado de forma isolada. Cada pacote particular só é considerado transferido pelo emissor quando este recebe o ACK respectivo. Dada a utilização de alarmes associados a cada pacote, enquanto o ACK não for recebido, esse pacote é retransmitido. Por outro lado, se algum ACK se perder, o protocolo assegura que serão enviadas cópias dos ACKs quando o receptor receber duplicados.

Listing 6.3: Pseudo código da parte cíclica do emissor do protocolo SR sem limitação do número de retransmissões.

```
// "window" is a queue with maximum length N and
// with item positions from 0 to window.size()-1 if not empty
seq = 1
connection.create()

while ( more application packets to send or window.size() > 0 ) {

    On event < end of last packet transmission and window.size() == 0 >:
        // ignore it and loop again

    On event < end of last packet transmission and window.size() != 0 >:
        position = window.getNextPacketToSend()
        window.sendPacket(position)
        // Start a timer for this packet
        startTimer( duration, window.getSequenceNumber(position) )

    On event < ACK and ACK.sequence in window >:
        window.markAsAcked(ACK.sequence)
        deleteTimer(ACK.sequence)
        // Remove acknowledged packets at the start of window if any
        while ( window.size() > 0 && window.getFirst().isAcked() )
            window.removeFirst()

    On event < ACK and ACK.sequence not in window >:
        // ignore it and loop again

    On event < TIMEOUT or NACK >: // both events have a sequence number
        position = window.getPacketPosition(event.sequence)
        if ( the packet is still in the window ) {
            window.sendPacket(position)
            startTimer( duration, window.getSequenceNumber(position) )
        }

    On event < applicationPacketAvailable and window.size() < N >:
        packet = getNextApplicationPacket()
        packet.opCode = DATA
        packet.sequence = seq
        window.addLast(packet)
        seq++
}

connection.close()
```

Listing 6.4: Pseudo código da parte cíclica do receptor do protocolo SR

```
seq = 1
while ( connection.isActive() ) {

    On event < DATA and DATA.sequence in window >:
        send(ACK, DATA.sequence);
        position = window.getPacketPosition(DATA.sequence);
        window.put(DATA, position);
        if (position != 0) send(NACK, window.getFirst().sequence())
        // Try to deliver as many packets as possible to the application
        while (window.size() > 0 and window.getFirst().isAvailable() ) {
            window.deliverFirstToApplication();
            window.deleteFirst();
        }

    On event < DATA and DATA.sequence not in window >:
        send(ACK, DATA.sequence); // resend ack
}
```

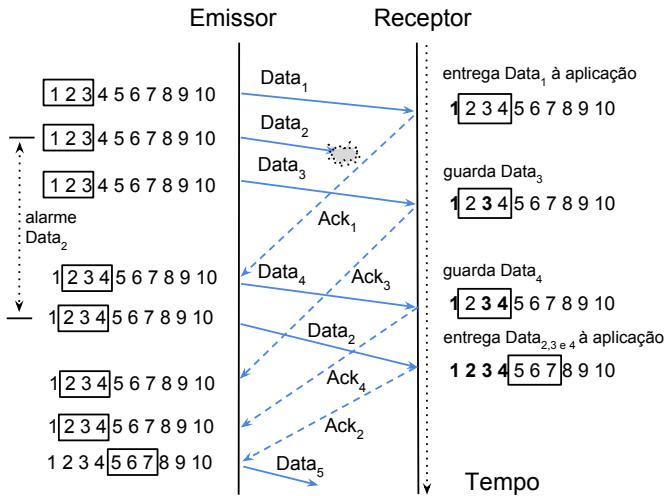


Figura 6.19: O protocolo SR em ação, com recuperação mais rápida do erro e sem retransmissão de pacotes já recebidos pelo receptor

Como por hipótese, mais tarde ou mais cedo, os pacotes enviados do emissor para o receptor, e vice versa, acabam por chegar ao destino, o protocolo acabará por fazer progresso. Adicionalmente, por hipótese, existe uma associação inequívoca entre números de sequência e pacotes, pelo que não pode resultar alguma confusão entre os números de sequência dos pacotes e dos respectivos ACKs.

Finalmente, as janelas só avançam de forma que asseguram que as condições anteriores se verificam. A janela do emissor só avança quando o emissor tem a certeza de que o receptor recebeu os pacotes. A janela do receptor só avança quando contém pacotes em sequência e que portanto podem ser entregues à aplicação sem violar as condições de fiabilidade do protocolo.

Tal como já vimos a propósito do protocolo *stop & wait*, a regulação da duração do alarme é um problema delicado com impacto apenas sobre o desempenho, mas sem implicações na correcção. Uma duração de *timeout* demasiado baixa provoca a emissão prematura de pacotes ainda em trânsito. Um valor de *timeout* demasiado elevado, atrasa a recuperação de erros. Por outro lado, a utilização de NACKs apenas serve para levar o emissor a retransmitir um pacote em falta o mais cedo que possível, mesmo sem a duração do respectivo *timeout* se esgotar completamente.

Comparação entre SR e GBN

Um dos aspectos mais complexos da implementação do protocolo SR tem a ver com a gestão dos *timeouts*. Naturalmente, a existência de tantos alarmes activos quantos os pacotes na janela (a janela de emissão pode em certas circunstâncias necessitar de conter várias centenas de pacotes) seria complexo e pouco realista. No entanto, é possível associar a cada pacote na janela a hora a que este deve ser reemitido se no entretanto não tiver sido *ack'd*, e usar apenas um alarme. Quando este dispara, é necessário determinar que pacote(s) já deveriam ter sido *ack'd*(s), retransmitir esse(s) pacote(s), calcular de novo quanto tempo falta para a hora de retransmissão mais próxima entre os pacotes presentes na janela, e activar de novo apenas um alarme com esse valor.

De resto os dois protocolos só diferem na forma como tratam os erros e desenca-deiam as retransmissões. O protocolo SR tem tendência a transmitir menos duplicados

que o GBN na medida em que o SR só retransmite inutilmente um pacote quando o respectivo ACK se perdeu, enquanto que o GBN tem tendência a retransmitir mais duplicados, pois ao voltar atrás recomeça a transmissão de todos os pacotes que se seguem ao que foi responsável pelo *timeout*, mas alguns desses podem já ter chegado. Sobretudo com grandes janelas, uma perda esporádica de um pacote implica a transmissão de menos pacotes em duplicado com SR do que com GBN. No entanto, sem NACKs, o GBN ao detectar ACKs cumulativos repetidos recupera das perdas mais depressa pois não tem de esperar pelo desencadear do alarme (*timeout*), que é o único meio à disposição do SR para detectar perdas nesse cenário.

Provavelmente, como veremos mais tarde, uma solução mais interessante passaria por associar as vantagens dos dois protocolos, *i.e.*, juntar os ACKs cumulativos do GBN com um mecanismo de sinalização do que foi e não foi recebido correctamente para a frente do ACK cumulativo. Esta é solução adoptada em opção pelo protocolo TCP, ver a Secção 7.2.

De qualquer forma, uma observação atenta dos diferentes protocolos de janela deslizante permite concluir que estes apenas diferem no método de recuperação dos erros. Se não ocorrerem erros, a velocidade de transferência de dados do emissor para o receptor só depende da dimensão da janela do emissor e dos mesmos parâmetros que o protocolo *stop & wait*. Por esta razão torna-se fácil analisar o desempenho dos protocolos de janela deslizante num cenário em que não ocorram erros.

Desempenho do protocolo de janela deslizante sem erros

A Figura 6.20 mostra o progresso de uma transferência do emissor para o receptor quando o emissor tem uma janela de emissão de dimensão $N > 1$.

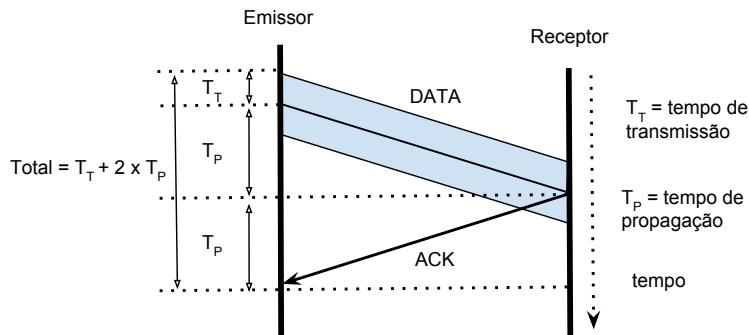


Figura 6.20: Progresso de um protocolo de janela deslizante sem erros e com $N = 2$

Com as mesmas hipóteses que as usadas aquando da avaliação do desempenho do protocolo *stop & wait*, cada ciclo tem uma duração $T_{total} = T_T + 2.T_P$ e durante o mesmo transmite-se a quantidade de bits correspondentes aos pacotes da janela. Portanto, o débito extremo a extremo (V) será a dimensão em bits dos N pacotes de dados (D) a dividir por T_{total} quando $N.T_T < T_T + 2.T_P$.

$$V = \frac{N.D}{T_T + 2.T_P} = N.V_{stop \& wait} \quad (6.3)$$

ou a velocidade máxima do emissor quando a janela não se esgota em $T_T + 2.T_P$. A

taxa de utilização (*usage ratio*), nas mesmas condições, é dada por

$$T_U = \frac{N \cdot T_T}{T_T + 2 \cdot T_P} = N \cdot T_{U_{stop \& wait}} \quad (6.4)$$

ou 100% quando a janela também não se esgota em $T_T + 2 \cdot T_P$.

Nas situações em que o tempo de transmissão seja desprezável face ao RTT ($T_T \approx 0$ e $RTT \approx 2 \cdot T_P$), a velocidade de transmissão extremo a extremo, se não ocorrerem erros, aproxima-se do número de bits que cabem na janela, a dividir pelo RTT

$$V = \frac{N \cdot D}{RTT} \quad (6.5)$$

Sem ocorrência de erros, os desempenhos dos protocolos GBN e SR são iguais vistos que ambos transferem ao ritmo máximo permitido pelo protocolo de janela deslizante. No entanto, a recuperação de erros é geralmente mais lenta no GBN como já se discutiu.

A ocorrência de erros implica que a transferência de alguns pacotes dura mais do que $T_T + 2 \cdot T_P$, com eventual impacto no atraso da transferência de outros pacotes, sobretudo no caso do GBN. Dado que a ocorrência de erros tem um carácter estocástico, a dedução do desempenho em função da distribuição dos erros é mais elaborada, ver as referências na secção 6.5.

Nomenclatura e parâmetros dos protocolos de janela deslizante

Todos os protocolos vistos neste capítulo são caracterizáveis, no essencial, pelo tamanho das janelas que usam. Na verdade, o protocolo *stop & wait* é um protocolo de janela deslizante com as janelas de emissão e recepção iguais a 1. A tabela a seguir apresenta essa visão.

Tabela 6.2: Dimensão das janelas usadas pelos protocolos de janela deslizante

Protocolo	Emissor	Receptor
<i>Stop & Wait</i>	1	1
<i>Go-Back-N</i>	N	1
<i>Selective Repeat</i>	N	N

Os protocolos de janela deslizante podem ser usados ao nível dos canais, ou aos níveis transporte ou aplicação. Quando os canais dominantes tinham uma taxa de erros significativa, eram usados protocolos deste tipo entre as extremidades de cada canal para compensar imediatamente os erros detectados. Hoje em dia, a maioria dos canais exibem taxas de erro muito baixas, especialmente os baseados em fibra óptica, e assim deixa-se aos níveis transporte ou aplicação a responsabilidade de recuperar os erros extremo a extremo. A exceção a esta regra são os canais sem fios, que continuam a exibir taxas de erros significativas. Esperar pela recuperação extremo a extremo depende de um valor de *timeout* necessariamente mais elevado que o RTT do canal, e até que um pacote seja retransmitido, o canal esteve subaproveitado.

Por essa razão, quase todos os canais sem fios, em particular os canais Wi-Fi, usam um protocolo de retransmissão de pacotes ao nível canal. Os canais Wi-Fi transmitem no máximo a uma centena de metros, como a velocidade de propagação do sinal no ar é cerca de $300.000.000$ m/s, o tempo de propagação máximo é de $0,33 \cdot 10^{-6}$ segundos ou $0,33 \mu s$. Por esta razão esses canais usam o protocolo *stop & wait*.

No início desta secção já discutimos os constrangimentos a que tem de obedecer a escolha da dimensão da janela do emissor. Em protocolos usados directamente em canais essa dimensão é fixa. No entanto, nos protocolos de janela deslizante usados nos

níveis superiores essa dimensão é geralmente variável. A mesma depende do estado da rede e da capacidade do receptor e das aplicações consumirem atempadamente os pacotes novos chegados, *i.e.*, por razões de controlo da saturação da rede, ver o Capítulo 8, e de controlo de fluxo.

Como também referimos atrás, os números de sequência, dado darem a volta, precisam de evoluir num intervalo suficientemente grande para que a sua reutilização não provoque confusão entre um pacote novo e um pacote muito atrasado.

Se um canal ou uma rede não trocarem a ordem de entrega dos pacotes, os protocolos *stop & wait* e GBN, com emissor com janela de dimensão N e receptor com janela de dimensão 1, necessitam de tantos números diferentes quanto a dimensão da janela do emissor + 1. Assim, o *stop & wait* precisa de 2 números de sequência diferentes antes da reutilização, e o GBN necessita de $N+1$ números diferentes se a janela do receptor tiver dimensão 1. No fundo, ambos os protocolos necessitam do número máximo de pacotes possíveis em trânsito mais um. Para justificar essa necessidade de mais um, pode-se pensar no caso limite em que o emissor envia toda a janela, portanto N pacotes, e o receptor envia todos os N ACKs, mas todos eles se perdem. O emissor reemitiria a janela integralmente, mas para o receptor seriam os N pacotes seguintes pois com N números de sequência distintos, estes têm de ser reutilizados. O protocolo SR com janelas de dimensão N necessita de $2.N+1$, pois quando o receptor já recebeu todos os pacotes da janela do emissor, mas entretanto perdeu-se o ACK do mais antigo, ele ainda pode receber pacotes duplicados com número de sequência igual ao do próximo novo pacote a receber menos N , e nesse caso não saberia distinguir o novo pacote daquele que se perdeu.

Tabela 6.3: Número mínimo de números de sequência distintos requeridos pelos protocolos quando o canal ou a rede não trocam a ordem dos pacotes (N é a dimensão das janelas)

Protocolo	Número mínimo
<i>Stop & Wait</i>	2
<i>Go-Back-N</i>	$N+1$
<i>Selective Repeat</i>	$2.N+1$

No caso em que a rede possa trocar a ordem de entrega dos pacotes, o problema torna-se mais delicado, tanto mais que é também necessário assegurar que pacotes de uma conexão antiga não são confundidos com os pacotes de uma conexão mais recente entre os mesmos extremos. Por esta razão, estes protocolos usam um grande intervalo de número de sequência distintos, representado em muitos bits, para que haja uma grande folga antes de ser necessário dar a volta e reutilizar números de sequência. Por exemplo, o protocolo TCP usa números de sequência representados em 32 bits, ver o Capítulo 7.

Uma outra faceta dos protocolos de janela deslizante tem a ver com a escolha do valor a usar no temporizador que guarda a recepção dos ACKs dos pacotes emitidos, que geralmente se designa por *timeout duration* ou simplesmente *timeout*. Como se referiu várias vezes, o valor do *timeout* deve ser suficiente para acomodar o RTT com alguma folga.

Quando os protocolos de janela deslizante estão a ser usados num quadro estável, por exemplo nas extremidades de um canal ponto-a-ponto, o valor do *timeout* pode ser estimado em função das características do canal e ser constante. Quando estes protocolos estão a ser usados extremo a extremo, o valor do RTT pode ser variável e, quando a rede está próxima da saturação e as conexões são de muito longa distância,

o valor do *jitter* pode ser significativo. Nestes casos os protocolos devem estimar dinamicamente os valores do RTT e do *jitter*, e ajustar o valor do *timeout* de forma a acomodar as variações. Este assunto será de novo retomado a propósito da discussão do protocolo TCP, ver o Capítulo 7.

Para terminar a discussão das diversas facetas dos protocolos acima descritos vamos analisar o que se passa quando uma conexão é bidireccional. Nesse caso poderíamos usar duas instâncias independentes do protocolo, uma em cada sentido. No entanto, a fusão das duas instâncias de um protocolo numa implementação conjunta permite algumas optimizações. A principal dessas optimizações consiste em transportar informação de controlo nos pacotes de dados.

Assim, sempre que uma das partes envia dados para a outra, o pacote é do tipo DATA e contém um número de sequência. No entanto, esse pacote pode também enviar informação de controlo, como por exemplo um ACK do último pacote recebido no sentido contrário. Esta abordagem necessita de uma expansão do cabeçalho, introduzindo um código de operação e um número de sequência suplementares, como é ilustrado na Figura 6.21. Se não é enviada informação ou controlo na outra direcção, o código de operação NOP (ausência de informação no sentido contrário) desfaz a ambiguidade. O termo na língua inglesa para designar esta técnica é *information piggybacking* (“andar às cavalitas”) pois a informação de controlo (no sentido contrário) vai às “cavalitas” dos dados.

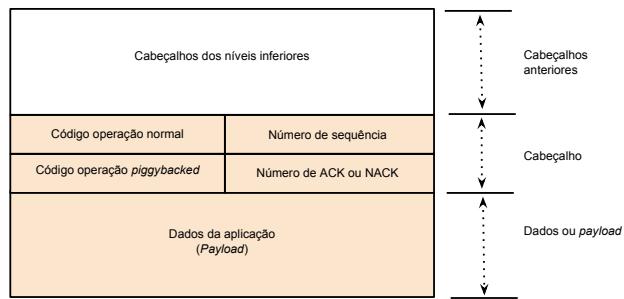


Figura 6.21: Formato das mensagens usadas pelos protocolos em conexões bidireccionais com informação *piggybacked*

6.4 Protocolos e máquinas de estado com acções

Acima descrevemos os protocolos introduzidos através de descrições informais e outras vezes algoritmos. A maioria desses algoritmos estavam estruturados como sequências de acções executadas em resposta a eventos, como a recepção de uma mensagem da rede, o disparo de um alarme, a recepção de uma mensagem da aplicação, *etc.*

Os sistemas deste tipo podem ser facilmente descritos usando máquinas de estado com acções, tratando-se basicamente de autómatos de estados. Partindo de um estado inicial, o autómato espera por um evento. Quando este ocorre, o autómato executa uma ou mais acções e transita para outro estado, onde ficará à espera do evento seguinte, e assim sucessivamente. Os eventos pertencem ao alfabeto aceite pelo autómato. Para melhor definir o seu comportamento podemos associar condições (*i.e.*, predicados) ao evento.

Uma **máquina de estados** (*state machine*) é um autómato com transições de estado associadas a eventos e predicados, que executa acções na transição entre estados. Um autómato deste tipo é uma forma sintética e semi-formal de descrever um protocolo e a sua evolução no tempo em função dos eventos que vão ocorrendo.

Um evento pode ser, por exemplo, receber uma mensagem ACK da rede. A condição suplementar pode ser testar se o número de sequência associado ao ACK é o que se está à espera.

O autómato é representado por um grafo em que os nós são os estados e os arcos são os eventos que ligam o estado em que o evento ocorre ao estado seguinte. Cada arco tem uma etiqueta com a indicação do evento a que está associado e as condições suplementares impostas, e também as acções a executar antes da transição de estado, separadas do evento por uma barra. Ver a Figura 6.22.

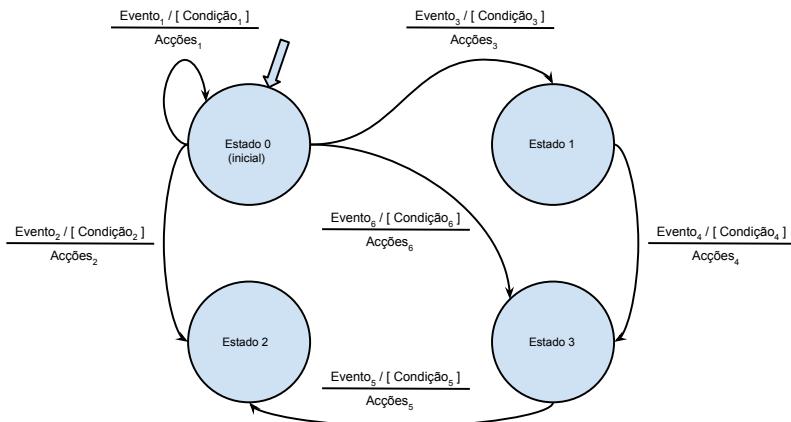


Figura 6.22: Representação gráfica de uma máquina de estados com acções

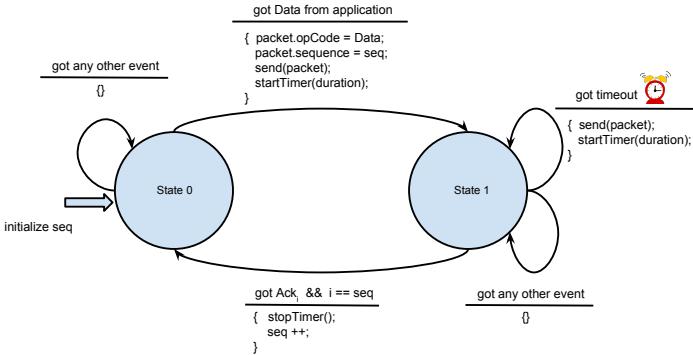
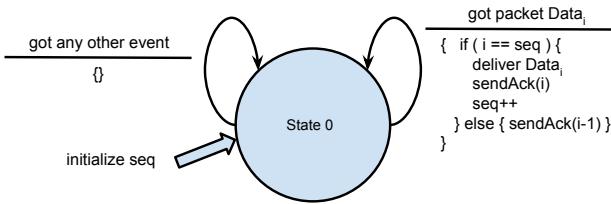
As Figuras 6.23 e 6.24 apresentam o protocolo *stop & wait* através de duas máquinas de estado, uma do emissor, e outra do receptor, respectivamente. A representação do protocolo através das duas máquinas de estados é equivalente ao algoritmo 6.1. No entanto, as máquinas de estados permitem uma visão gráfica e sintética, uma compreensão mais simples do protocolo, e a sua passagem a algoritmo com relativa facilidade.

A seguir apresenta-se um breve resumo e uma revisão dos principais conceitos introduzidos neste capítulo.

6.5 Resumo e referências

Resumo

Sempre que é necessário transferir informação de forma fiável através de uma rede ou de canais, que podem perder pacotes, é necessário utilizar mecanismos de compensação dessas falhas. Por hipótese, considera-se que os pacotes que chegam com erros são detectados e são rejeitados. Assim, um erro num pacote é equivalente à sua perda. Com o objectivo de introduzir fiabilidade num processo de transferência de dados,

Figura 6.23: Máquina de estados do emissor *stop & wait*Figura 6.24: Máquina de estados do receptor *stop & wait*

neste capítulo analisámos uma solução, correspondente a um conjunto de protocolos, que consiste em retransmitir os pacotes que não chegaram ao receptor. Este grupo de protocolos costuma ser designado por protocolos ARQ (*Automatic Repeat ReQuest*).

Adicionalmente, estes protocolos incluem um mecanismo de controlo de fluxo (*flow control*), o qual consiste num mecanismo de extremo a extremo que adapta o ritmo de emissão de pacotes pelo emissor à capacidade de o receptor os tratar em tempo útil.

Os protocolos de transferência fiável de dados em redes de pacotes, ou em canais, que se baseiam no método de retransmissão dos pacotes perdidos, usam os seguintes mecanismos de base: pacotes de dados, pacotes de controlo, alarmes temporizados (*timeouts*) e números de sequência. Os pacotes de controlo servem para o receptor assinalar ao emissor a recepção dos pacotes enviados (ou também a sua ausência). Os números de sequência permitem que o emissor e o receptor identifiquem univocamente os pacotes.

O capítulo apresentou três protocolos que se distinguem entre si pela quantidade máxima de pacotes simultaneamente em trânsito entre o emissor e o receptor. Com o protocolo *stop & wait* o emissor envia apenas um pacote de cada vez, e só passa ao seguinte quando tem a certeza que o receptor recebeu o anterior, *i.e.*, após ter recebido uma mensagem de controlo de confirmação de recepção (ACK) com o mesmo número de sequência que a do último pacote enviado.

Os protocolos *Go-Back-N* e *Selective Repeat* mantêm vários pacotes em trânsito simultaneamente. Diferem pela forma como tratam os erros, assinalados pelos alarmes (*timeouts*) que marcam a não recepção atempada das mensagens de confirmação de recepção (ACKs). Em ambos os protocolos, os pacotes em trânsito mantêm-se num *buffer*, chamado janela do emissor, que mantém uma cópia dos mesmos, pois podem ter de ser retransmitidos.

O capítulo discute também a velocidade de transferência extremo a extremo que os diferentes protocolos permitem. Essa discussão circunscreveu-se ao caso mais simples que é quando o RTT é constante e não existem erros. O desempenho destes protocolos é muito influenciado por diversos parâmetros, nomeadamente o número máximo de pacotes em trânsito (janela de emissão) e o valor do *timeout*. A forma de fixar esses parâmetros foi também discutida.

Finalmente, foi exemplificado como os protocolos podem ser especificados de forma semi formal através de máquinas de estado, usando eventos e predicados para desencadearem as transições de estado, e a execução de ações associadas às transições de estado.

Os principais termos introduzidos neste capítulo são a seguir passados em revista. Entre parêntesis figura a tradução mais comum em língua inglesa.

Controlo de fluxo (*flow control*) Um mecanismo de controlo de fluxo é um mecanismo de extremo a extremo que adapta o ritmo de emissão de pacotes pelo emissor à capacidade de o receptor os tratar em tempo útil.

Protocolos ARQ (*Automatic Repeat ReQuest protocols*) Protocolos de transferência fiável de dados, baseados na retransmissão dos pacotes cuja confirmação de recepção (ACK) pelo receptor ainda não foi recebida pelo emissor. Estes protocolos usam os seguintes mecanismos de base: pacotes de dados, pacotes de controlo, alarmes temporizados (*timeouts*) e números de sequência.

Protocolo stop & wait (*stop & wait protocol*) Protocolo em que o emissor envia um pacote e só passa a enviar o pacote seguinte quando receber a confirmação da sua recepção (ACK) vinda do receptor.

Protocolos de janela deslizante (*sliding window protocols*) Protocolos em que o emissor envia vários pacotes de avanço, mesmo sem receber os ACKs referentes aos mais antigos. Os pacotes em trânsito são mantidos num *buffer* de emissão chamado a janela do emissor. A dimensão desta janela é limitada por razões de controlo de fluxo e de controlo da saturação da rede.

Protocolo GBN (*Go-back-N protocol*) Protocolo que usa uma janela de emissão maior que 1. Quando dispara um alarme (*timeout*), o emissor recomeça a enviar por ordem todos os pacotes que ainda estão em trânsito, a começar pelo mais antigo.

Protocolo SR (*Selective Repeat protocol*) Protocolo que usa janelas de emissão e recepção maiores que 1. O protocolo trata cada pacote em trânsito de forma independente. Quando dispara um alarme (*timeout*), o emissor reenvia apenas o pacote ao qual o alarme está associado.

Máquina de estados (*state machine*) Autómato com transições de estado desencadeadas por eventos e predicados, que executa ações na transição entre estados. Um autómato deste tipo é uma forma sintética e semi-formal de descrever um protocolo e a sua evolução em função dos eventos que vão ocorrendo no tempo.

Referências

W. Stallings apresenta em [Stallings, 2013], no capítulo 7, a dedução da utilização de um canal com erros pelos protocolos *stop & wait*, GBN e SR.

Apontadores para informação na Web

- http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/index.htm – Aponta para um programa que permite visualizar o funcionamento dos protocolos *stop & wait* e GBN.

6.6 Questões para revisão e estudo

Salvo indicação em contrário, nas questões seguintes em que é necessário avaliar o desempenho dos protocolos, considere que todos os canais são ponto-a-ponto, bidimensionais, *full-duplex* (isto é, transmite nos dois sentidos simultaneamente) e considere que o tempo de processamento e o tempo de transmissão de eventuais mensagens de ACK são desprezáveis. Assuma igualmente que a velocidade de propagação do sinal nos canais é de 200.000 Km por segundo. Finalmente, despreze nos pacotes ou nas mensagens o espaço ocupado pelos cabeçalhos e considere que, por hipótese, e se não for indicado nada em contrário, não existem erros no canal.

1. A regulação do valor do *timeout* usado pelo emissor num protocolo de janela deslizante tem impacto no desempenho do protocolo em situações de erro.
 - (a) Indique os inconvenientes de um valor de *timeout* demasiado curto.
 - (b) Indique os inconvenientes de um valor de *timeout* demasiado longo.
 - (c) Indique como deve ser calculado o valor do *timeout* numa situação em que há um único canal entre o emissor e o receptor.
2. Um canal de dados tem a velocidade de transmissão de 100 Kbps e o tempo de propagação de uma extremidade à outra de 10 ms. Para mensagens de que comprimento é possível atingir uma taxa de utilização de pelo menos 50% com o protocolo *stop & wait*?
3. Considere um emissor a usar o protocolo GBN cujo código executável mantém actualizadas as seguintes variáveis:

LastPacketSent – contém o número de sequência do último pacote de dados enviado;

LastAckReceived – contém o número de sequência do ultimo ACK recebido e aceite;

MaxWindowSize – contém o tamanho máximo da janela (na verdade é uma constante).

Quais das seguintes relações são invariantes e a implementação do protocolo tem de assegurar que não são violadas? Selecione a resposta certa:

- (a) $\text{LastPacketSent} + \text{LastAckReceived} > \text{MaxWindowSize}$
- (b) $\text{LastPacketSent} + \text{LastAckReceived} \geq \text{MaxWindowSize}$
- (c) $\text{LastPacketSent} - \text{LastAckReceived} + 1 \leq \text{MaxWindowSize}$
- (d) $\text{LastPacketSent} - \text{LastAckReceived} \leq \text{MaxWindowSize}$
4. Verdadeiro ou Falso? Justifique.
 - (a) No protocolo SR é possível o emissor receber um ACK de um pacote que cai fora da sua janela de emissão.
 - (b) No protocolo GBN é possível o emissor receber um ACK de um pacote que cai fora da sua janela de emissão.
 - (c) O protocolo *stop & wait* é equivalente ao protocolo SR com o emissor e o receptor com janelas de dimensão 1.
 - (d) O protocolo *stop & wait* é equivalente ao protocolo GBN com o emissor e o receptor com janelas de dimensão 1.
 - (e) No protocolo GBN o receptor com janela igual a 1 nunca confirma a boa recepção de pacotes cujo número de sequência não seja o do pacote de que está à espera.

- (f) Com o protocolo GBN não vale a pena utilizar NACKs pois os mesmos só são úteis no protocolo SR.
 - (g) A utilização de valores de *timeout* demasiado curtos no protocolo SR impede a transferência fiável dos dados do emissor para o receptor.
 - (h) Seja qual for a situação, a utilização de uma grande janela de emissão com o protocolo GBN é sempre benéfica.
 - (i) Seja qual for a situação, a utilização de uma grande janela de emissão com o protocolo SR é sempre benéfica.
5. Um canal de dados entre dois computadores tem o débito de 1 Mbps e o tempo de propagação de 95 milissegundos. Qual a taxa de utilização do canal usando o protocolo *stop & wait* com pacotes de 10.000 bits?
 6. Um canal de dados entre dois computadores tem o tempo de propagação de 95 milissegundos. Qual o débito do canal a partir do qual a taxa de utilização do canal usando o protocolo *stop & wait* é superior a 90% com pacotes de 10.000 bits?
 7. Dois computadores A e B estão ligados diretamente por um canal com o débito de 1 Mbps e o tempo de propagação de 25 milissegundos. Escolha o tamanho mínimo do pacote (em bits) que permite uma taxa de utilização do canal de pelo menos 33,3% usando um protocolo *stop & wait*.
 8. Os computadores A e B estão a usar o protocolo GBN com uma janela de 10 pacotes de dados de 20.000 bits. O tempo de transmissão de cada pacote é de 20 milissegundos e o tempo de propagação entre A e B é de 30 milissegundos. Qual o tempo total que leva a transmitir um ficheiro com 6.000.000 bits através desse protocolo?
 9. Um canal de dados com o débito de 1,5 Mbps é utilizado para transmitir pacotes com 10.000 bits. O tempo total de propagação de uma extremidade à outra do canal é de 250 milissegundos. Os números de sequência são representados em 4 bits. Indique qual a taxa de utilização máxima deste canal com os protocolos *stop & wait* e GBN.
 10. Considere uma situação em que dois computadores estão ligados directamente por um canal ponto-a-ponto dedicado. Os computadores estão a usar o protocolo *stop & wait* para transferir dados de um para o outro. Caso o canal seja *half-duplex*, ao invés de *full-duplex*, o desempenho do protocolo é diferente?
 11. Existe uma versão do protocolo *stop & wait* que se chama “*alternating-bit protocol*”. Esta versão utiliza um único bit para codificar o número de sequência dos pacotes transmitidos do emissor para o receptor. Diga se seria possível realizar este protocolo sobre uma rede como a Internet.
 12. Considere os protocolos de janela deslizante e responda às seguintes questões de forma justificada.
 - (a) Indique pelo menos duas vantagens de num protocolo deste tipo ter o receptor com uma janela maior que 1.
 - (b) Indique se existe alguma vantagem de num protocolo deste tipo ter o receptor com uma janela maior que a do emissor.
 13. Dois computadores A e B estão ligados diretamente por um canal com o débito de 200 Kbps e o tempo de propagação de uma extremidade à outra de 75 milissegundos. A está a enviar para B pacotes com 10.000 bits de comprimento.
 - (a) Qual o número máximo de pacotes por segundo (pps) que A consegue transmitir para B transmitindo pacotes continuamente?

- (b) Qual o número máximo de pacotes por segundo (pps) que A consegue transmitir para B usando o protocolo *stop & wait*?
- (c) Qual a taxa de utilização do canal nas condições da alínea anterior?
14. Calcule o tempo total de transferência (extremo a extremo) entre dois computadores de um ficheiro com 1.000.000 bytes. Os computadores estão ligados diretamente por um canal com 10.000 Km de comprimento e portanto o tempo de propagação é de 50 ms (RTT = 100 ms). Na transferência são sempre usados pacotes com 500 bytes de dados.
- (a) O débito do canal é de 1 Mbps e os pacotes podem ser enviados continuamente. O ficheiro considera-se transferido quando o receptor acabar de receber o último pacote. Qual a taxa de utilização do canal?
 - (b) O débito do canal é de 1 Mbps mas depois de enviar cada pacote é necessário esperar um RTT pois está-se a usar o protocolo *stop & wait*. O ficheiro considera-se transferido quando o receptor acabar de receber o último pacote e o emissor receber o ACK dessa recepção. Qual a taxa de utilização do canal?
 - (c) O débito do canal é de 1 Mbps mas em cada RTT só se podem enviar 10 pacotes pois está-se a usar um protocolo de janela deslizante e a janela do emissor pode ter 10 pacotes. O ficheiro considera-se transferido quando o receptor acabar de receber o último pacote e o emissor receber o ACK dessa recepção. Qual a taxa de utilização do canal?
 - (d) O débito do canal é de 1 Mbps mas em cada RTT só se podem enviar 100 pacotes pois está-se a usar um protocolo de janela deslizante e a janela do emissor pode ter 100 pacotes. O ficheiro considera-se transferido quando o receptor acabar de receber o último pacote e o emissor receber o ACK dessa recepção. Qual a taxa de utilização do canal?
 - (e) O débito do canal é “infinito”, isto é, o tempo de transmissão pode ser desprezado, mas em cada RTT só se podem enviar 10 pacotes pois está-se a usar um protocolo de janela deslizante e a janela do emissor pode ter 10 pacotes. O ficheiro considera-se transferido quando o receptor acabar de receber o último pacote e o emissor receber o ACK dessa recepção. Tem sentido falar em taxa de utilização de um canal com débito infinito?
 - (f) O débito do canal é “infinito”, isto é, o tempo de transmissão pode ser desprezado, mas a janela vai aumentando dinamicamente e no 1º RTT pode-se enviar 1 pacote (isto é 2^{1-1} pacotes), durante o 2º podem ser transmitidos 2 pacotes (2^{2-1} pacotes), durante o 3º podem ser transmitidos 4 pacotes (2^{3-1} pacotes), durante o enésimo podem ser transmitidos 2^{N-1} pacotes. O ficheiro considera-se transferido quando o receptor acabar de receber o último pacote e o emissor receber o ACK dessa recepção. Tem sentido falar em taxa de utilização de um canal com débito infinito?
15. Um canal de dados tem o débito de 100 Kbps e tem 50.000 Km de comprimento. Calcule a sua taxa de utilização por um protocolo de janela deslizante com janelas com 1, 8 e 1000 mensagens, cada uma com 10.000 bits.
16. Um canal de dados tem o débito de 1 Mbps e tempo de propagação de uma extremidade à outra de 100 milissegundos.
- (a) Suponha que se utiliza o protocolo *stop & wait* e mensagens com 10000 bits. Qual é a taxa de utilização do canal?
 - (b) Usando o mesmo protocolo pretende-se aumentar a taxa de utilização para próximo de 100%. O que pode ser feito? A ou as soluções escolhidas têm inconvenientes? Se sim, quais são?

- (c) Suponha que se substitui o protocolo *stop & wait* por um protocolo de janela deslizante e que se continuam a usar mensagens com 10.000 bits. Para conseguir uma taxa de utilização de 90% do canal de dados, qual deveria ser a dimensão da janela do emissor?
17. Considere uma situação em que dois computadores A e B estão ligados através de uma rede. Os pacotes que transitam de A para B atravessam 2 comutadores de pacotes, C1 e C2, directamente ligados um ao outro. O canal entre C1 e C2 tem a dimensão de 1 Km e tem a capacidade de 1 Mbps. O computador A está ligado a C1 por um canal e o computador B está ligado a C2 por outro canal. Esses canais também têm a capacidade de 1 Mbps mas um tempo de propagação de 35 ms.
- Diga qual é o tempo de trânsito de A para B de pacotes com 10.000 bits de comprimento quando a rede só transmite esses pacotes.
 - Calcule a taxa de utilização da ligação entre A e B pelo protocolo *stop & wait* a utilizar pacotes com 10.000 bits de comprimento.
 - Calcule o tempo de transferência de A para B de um ficheiro com 1.000.000 bits nas condições da alínea anterior.
18. Considere uma situação em que dois computadores A e B estão ligados directamente a um comutador por canais com o débito de 1 Mbps e 5 ms de tempo de propagação. O computador A está a utilizar o protocolo *stop & wait* para enviar dados para o computador B usando pacotes com 10.000 bits. Qual a taxa de utilização pelo protocolo do canal que liga A ao comutador?
19. Considere uma situação em que dois computadores A e B estão ligados através de um canal. Os pacotes transmitidos de A para B têm dimensão constante e o tempo de transmissão T_T . O canal tem como tempo de propagação $N \cdot T_T$.
- Qual a taxa de utilização do canal usando o protocolo *stop & wait*?
 - Usando o protocolo *stop & wait* como evoluí da taxa de utilização nos seguintes casos: $N \ll 1$, $N = 1$ e $N \gg 1$?
 - Qual a taxa de utilização do canal com o protocolo GBN usando uma janela de emissão de K pacotes?
20. Os computadores A e B estão a usar o protocolo SR com uma janela de 5 pacotes com 20.000 bits cada para transmitir dados de A para B. O tempo de transmissão de cada pacote é de 20 milissegundos e o tempo de propagação entre A e B é de 30 milissegundos. Qual o tempo total aproximado em segundos que leva a transmitir um ficheiro com 6.000.000 bits através desse protocolo?
21. Um canal de dados com a velocidade de transmissão de 1 Mbps é utilizado para transmitir pacotes com 10.000 bits. O tempo total de propagação de uma extremidade à outra do canal é de 250 ms. Indique qual a taxa de utilização deste canal com um protocolo:
- stop & wait*.
 - Janela deslizante com uma janela do emissor de 10 pacotes.
 - Janela deslizante com uma janela do emissor de 100 pacotes.
22. Dois computadores A e B estão ligados através de um conjunto de canais e comutadores. Os pacotes que transitam de A para B têm todos N bits de comprimento, atravessam C comutadores e $C + 1$ canais ponto-a-ponto *full-duplex* intermédios. Todos os canais têm o tempo de propagação T_P e o tempo de transmissão dos pacotes T_T . Considere que, por hipótese, só existe na ligação entre A e B o tráfego correspondente à transmissão do ficheiro. Diga qual o tempo de transferência de um ficheiro de dimensão F pacotes de N bits:

- (a) Usando um protocolo optimista que envia todos os pacotes em sequência.
- (b) Usando o protocolo *stop & wait*.
23. Um canal com o débito de 1 Mbps liga o computador A ao computador B. O canal tem um tempo de propagação de extremo a extremo de 20 ms. Entre os dois computadores é executado um dos protocolos de transferência de dados de A para B do tipo janela deslizante que usa pacotes de dados de 10.000 bits. Responda às seguintes questões exprimindo o resultado em percentagem.
- Qual a taxa de utilização do canal entre A e B quando as janelas do emissor e do receptor são ambas iguais a 1 pacote?
 - Qual a taxa de utilização do canal entre A e B quando a janela do emissor é igual a 1 pacote e a do receptor é igual a 2 pacotes?
 - Qual a taxa de utilização do canal entre A e B quando a janela do emissor é igual a dois pacotes e a do receptor é igual a 1 pacote?
 - Quanto tempo leva a transmitir de A para B um ficheiro com a dimensão de 10 pacotes usando a versão do protocolo indicado na alínea anterior? A transferência só termina quando o emissor receber o último ACK. O resultado deve ser expresso em milissegundos.
24. Uma instituição tem uma sede e uma sucursal cada uma com um comutador de pacotes. Na sucursal estão 10 computadores ligados directamente ao comutador da sucursal por canais de 1 Gbps e 100 metros de comprimento. Na sede existe um servidor central ligado ao comutador da sede por um canal com 10 Gbps e com também 100 metros de comprimento. O comutador da sucursal está ligado ao comutador da sede por um canal com o débito de 1 Mbps e 90 ms de tempo de propagação. Pretende-se que cada um dos computadores da sucursal envie diariamente um ficheiro de 50 Mbytes para o servidor da sede através do protocolo *stop & wait* a usar pacotes de dados com 20.000 bits. Alguém sugeriu que o melhor seria transferir um ficheiro de cada vez pois desta forma seria mais rápido, e para além disso essa opção seria suficiente para garantir a transferência diária dos 10 ficheiros. Concorda com esta solução ou há uma alternativa melhor mesmo continuando a usar o protocolo *stop & wait*?
25. Responda às seguintes questões e justifique a sua resposta.
- Num protocolo de janela deslizante para que servem as mensagens de NACK enviadas pelo receptor quando os pacotes chegam fora de ordem ou estão em falta?
 - Conceba um protocolo de transferência fiável de A para B baseado apenas no envio de NACKS do emissor para o receptor e indique que condições tornariam realista esse protocolo.
 - As comunicações do emissor para o receptor correspondem a uma transmissão contínua que o receptor pode absorver facilmente e o canal que os liga tem uma taxa de erros desprezável. É preferível usar um protocolo baseado em ACKs ou um baseado só em NACKs para fazer as transferências?

Propostas de projecto de programação

Abaixo encontram-se a proposta de alguns projectos de programação relacionados com o protocolo *stop & wait* e o protocolo GBN.

- Realize, por exemplo na linguagem Java, um cliente do protocolo TFTP. Este protocolo é usado para transferir ficheiros entre um cliente e o servidor através de um protocolo semelhante ao *stop & wait*. O protocolo TFTP está definido

nos RFC 1350 e RFC 2348. O cliente deverá ser capaz de transferir um ficheiro do servidor para a máquina local, onde é executado. O servidor funciona como emissor e o cliente como receptor no protocolo. O cliente será invocado através do comando:

```
java TftpGet host port filename [-s blksize]
```

<filename> é o nome do ficheiro a enviar pelo servidor

<host> é o computador do servidor

<port> porta do servidor

[**-s blksize**] corresponde à opção **blksize**, conforme os RFCs do TFTP

No caso de o cliente terminar a transferência com sucesso, deve afixar a seguinte informação:

- Total de bytes transferidos (dimensão do ficheiro)
 - Número total de pacotes recebidos com dados (incluindo repetições)
 - Número total de pacotes enviados com ACKs
 - Tempo total que durou a transferência
 - Número total de pacotes de dados recebidos que eram duplicados
 - Tempo médio entre o envio de um ACK e a recepção do pacote de dados seguinte sempre que foi enviado o ACK N e o bloco a seguir recebido tem o número de série N+1
2. Complete o cliente anterior de forma a que este possa também funcionar como emissor do protocolo TFTP e seja capaz de transferir um ficheiro local para o servidor. O cliente será invocado através do comando:

```
java TftpPut host port filename [-s blksize]
```

em que os parâmetros têm o significado óbvio.

3. Modifique o cliente **TftpPut** para usar o protocolo GBN. Tenha em consideração que o servidor pode ser um servidor TFTP normalizado.

Capítulo 7

O protocolo TCP

There's an old maxim that says, "Things that work persist," which is why there's still Cobol floating around.

– Autor: Vinton G. Cerf

O protocolo TCP é o protocolo de transporte actualmente mais usado na Internet. Com efeito, o mesmo é o principal suporte do protocolo HTTP e portanto sustenta a maioria das aplicações que os utilizadores da rede utilizam. Mesmo o transporte de sinal multimédia em redes TCP/IP, usado para a visualização de filmes e canais de televisão, que era até há pouco tempo predominantemente feito sobre o protocolo UDP, começou recentemente também a utilizar TCP.

O protocolo TCP foi desenvolvido na primeira metade da década de 1970, a partir de muitas experiências realizadas anteriormente, e o artigo em que o mesmo foi cientificamente divulgado data de 1974[Cerf and Kahn, 1974]. Os seus autores, Vinton G. Cerf e Robert E. Kahn, receberam em 2004 o “Turing Award”, por muitos considerado o Prémio Nobel da Informática, como reconhecimento do trabalho que desenvolveram e que envolveu também o desenvolvimento do TCP.

Como diz a citação no início do capítulo, todas as coisas que funcionam e desempenham o seu papel têm tendência a ser usadas e não são substituídas. No entanto, ao contrário da linguagem Cobol, o protocolo TCP foi sendo melhorado, de tal forma que nunca foi substituído e, apesar de haver protocolos alternativos para os mesmos objectivos, a verdade é que o TCP continua a ser o protocolo de transporte fiável mais utilizado.

Trata-se de mais um exemplo notável da virtude da separação entre a interface de uma componente e a sua implementação, um princípio tão querido da Informática. Uma outra faceta notável do protocolo consiste também na definição minimalista dessa interface, de tal forma que a mesma foi resistindo a diferentes necessidades das aplicações e a diferenças importantes na forma como essa funcionalidade é implementada, sem que a interface tenha sido modificada.

Neste capítulo são apresentadas as funcionalidades e as características de base do protocolo que têm sido constantes desde a sua introdução. Veremos também quais os mecanismos que estão na base da sua implementação. A grande maioria dos mesmos foi introduzida logo nos primórdios da sua vida. Mais tarde alguns desses mecanismos foram modificados e introduzidas extensões e alternativas. No entanto, todos esses novos mecanismos e alterações foram introduzidos sempre com a intenção de melhorar o desempenho do protocolo e são, no essencial, transparentes às aplicações que o usam,

pois não alteram a funcionalidade e filosofia essenciais do mesmo. Ou seja, são apenas melhoramentos da implementação.

A maioria desses mecanismos mais recentes, assim como outras alternativas correntes, serão estudados nos capítulos que se seguem. Os mesmos estão intimamente relacionados com a própria evolução da Internet, que foi significativa desde 1974, o que teve como impacto a necessidade de adaptar o funcionamento do protocolo a essa evolução.

7.1 A interface do protocolo TCP

Como já foi referido nos capítulos 1 e 5, o protocolo TCP providencia a possibilidade de dois processos, no mesmo, ou em computadores distintos, comunicarem através de um canal lógico, designado conexão TCP. A conexão TCP permite a comunicação bidireccional e simultânea entre os dois processos. Duas versões da interface programática do protocolo já foram apresentadas nas secções 1.6 e 5.3.

O canal não disponibiliza nenhuma noção de mensagem, pois as unidades de informação transmitidas são sequências de um ou mais bytes cuja dimensão a aplicação não pode controlar. O emissor pode emitir 1000 bytes de uma só vez, mas o receptor pode ler a mesma sequência byte a byte. Não é possível ao receptor saber, através da interface do protocolo, como é que os dados que recebe foram originalmente particionados em sequências pelo emissor.

O canal é fiável no sentido em que os dados enviados pelos dois emissores, em qualquer uma das duas extremidades, chegarão ao receptor na outra extremidade pela mesma ordem e sem falhas. Caso estas duas propriedades não possam ser garantidas por qualquer motivo, a conexão não será estabelecida, ou se essa impossibilidade tiver lugar posteriormente, cada uma das partes acabará por receber uma exceção a comunicar que a conexão foi quebrada e deixou de ser possível.

O protocolo TCP incorpora dois mecanismos de controlo da velocidade de emissão: controlo de fluxo e controlo de saturação. Com efeito, ver a Figura 7.1, o protocolo TCP pressupõe que o emissor, em cada extremidade da conexão, tem um *buffer* do tipo produtor / consumidor partilhado com o nível rede do computador emissor, um *buffer* lógico constituído pelos pacotes de dados em trânsito pela rede entre os computadores emissor e receptor e, na outra extremidade, um *buffer* do tipo produtor / consumidor, partilhado entre o nível rede do computador e o programa receptor.

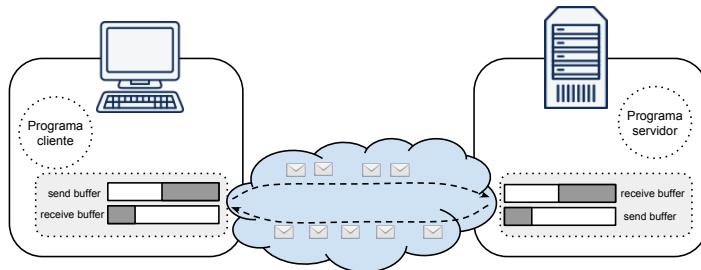


Figura 7.1: Os *buffers* envolvidos numa conexão TCP

O protocolo garante que os dados só deixam o *buffer* do emissor se (com alta probabilidade) existir espaço para os mesmos no *buffer* do receptor, usando para tal um mecanismo de controlo de fluxo que impede que um emissor rápido “afogue” um receptor lento. Como é sabido, o nível rede das redes TCP/IP não garante a entrega

de todos os pacotes pois, quer devido a erros, quer devido a saturação das filas de espera dos comutadores de pacotes, estes podem não chegar ao destino. O protocolo TCP incorpora também um mecanismo de controlo de saturação que tenta adaptar a velocidade do emissor à capacidade disponível na rede até ao destino. O objectivo é minorar a perda de pacotes devido a um débito de emissão de pacotes incompatível pela rede.

O protocolo não garante a velocidade de transferência extremo a extremo, nem sequer uma velocidade de transferência mínima. O TCP, deste ponto de vista, assume também um ponto de vista do tipo “melhor esforço”, *i.e.*, ambas as extremidades procurarão maximizar a velocidade de transferência extrema a extrema em cada sentido, mas a velocidade final será a que a rede poder suportar em cada momento.

Veremos mais adiante que a interface do protocolo TCP permite afinar alguns parâmetros do seu funcionamento que podem influenciar o desempenho do mesmo, nomeadamente a dimensão dos *buffers* do emissor e do receptor, assim como alguns dos parâmetros que condicionam a gestão destes *buffers*.

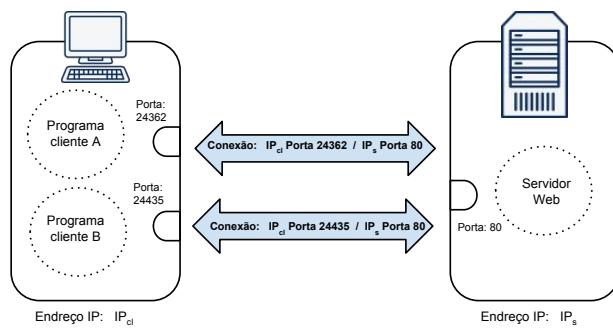


Figura 7.2: Endereços IP e portas envolvidos em duas conexões TCP entre um mesmo computador cliente e o mesmo servidor HTTP

Finalmente, importa referir que cada conexão tem de ser estabelecida antes de poder ser usada pelas duas partes, e que a mesma, como vimos no capítulo 5.3, é estabelecida entre dois sockets, caracterizados cada um por um endereço IP e uma porta. Assim, cada conexão TCP é inequivocavelmente identificada por quatro dados: o endereço IP do computador a partir do qual a conexão foi estabelecida, a porta usada na abertura, o endereço IP do computador da outra extremidade da conexão e a respectiva porta.

Por exemplo, considere-se o cenário da Figura 7.2, onde dois processos distintos no mesmo computador estabelecerem conexões TCP com um servidor Web. Do lado do servidor, ambos os sockets partilham o endereço do servidor e a porta 80, normalizada pelo protocolo HTTP. Do lado da máquina onde executa o cliente que abriu as conexões, ambos os sockets partilham o endereço IP do computador cliente, mas as portas têm de ser necessariamente distintas para que as duas conexões sejam também distintas.

O protocolo TCP disponibiliza canais lógicos fiáveis e bidireccionais entre dois processos, no mesmo ou em computadores distintos, designados conexões TCP. As conexões TCP transferem sequências de um ou mais bytes, sem qualquer relação entre a dimensão das sequências emitidas e a das recebidas, *i.e.*, o protocolo não tem a noção de mensagem.

O protocolo inclui um mecanismo de controlo de fluxo e um mecanismo de controlo de saturação da rede. Os programas utilizadores não podem solicitar ou impor os valores do débito extremo a extremo pois, deste ponto de vista, o protocolo também é baseado na noção de “melhor esforço”.

Uma conexão TCP é caracterizada pelos endereços IP e portas associadas aos sockets em ambas as extremidades. Duas conexões distintas têm necessariamente, um ou mais desses identificadores distintos.

Para perceber de forma mais completa em que consiste o protocolo teremos de penetrar na forma como está definido o comportamento das duas extremidades do canal TCP, isto é, na especificação do protocolo TCP.

7.2 Descrição do protocolo

O protocolo TCP é um protocolo de janela deslizante que pode funcionar no modo GBN (*Go-Back-N*), o modo por omissão, complementado opcionalmente com um modo semelhante ao SR (*Selective Repeat*). As unidades de informação transferidas entre as duas extremidades contêm dados e, opcionalmente, informação de controlo, e chamam-se **segmentos TCP**. O seu formato é o indicado na Figura 7.3.

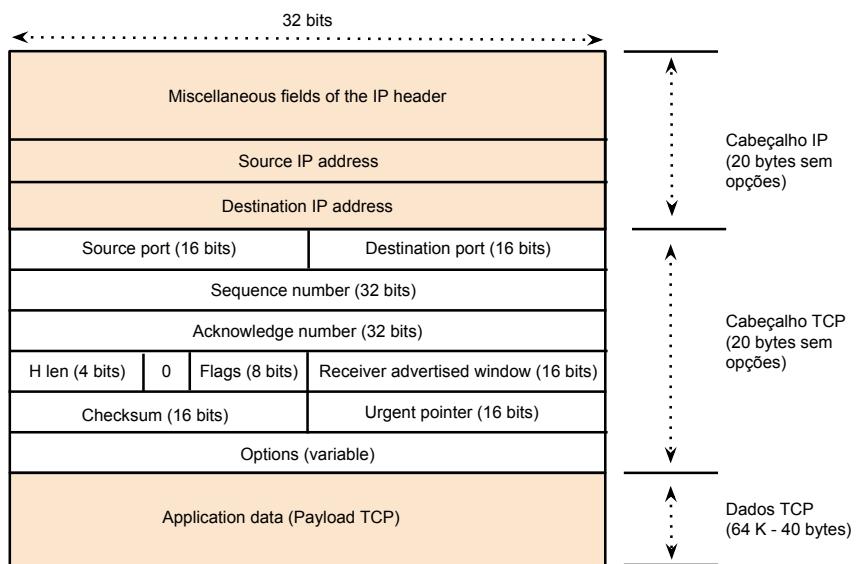


Figura 7.3: Formato de um segmento TCP

Os campos porta origem e destino, em conjunto com os campos endereço IP origem e destino do pacote IP, que encapsula o segmento TCP, permitem identificar inequivocavelmente a conexão a que o segmento pertence. Os campos número de sequência e número de ACK (do termo inglês *acknowledgement*) têm cada um 32 bits, e suportam o número de sequência dos dados quando o segmento viaja do emissor para o receptor e, opcionalmente, o número de sequência de ACK encavalitado (*piggybacked*) no sentido contrário. O mecanismo permite a cada uma das extremidades, quando envia dados, enviar também o número de sequência dos dados até aí correctamente recebidos no sentido contrário (ACK cumulativo).

O campo *flags* é constituído por um conjunto de 8 bits, em posições bem definidas, que são designados por *flags*. Cada um desses bits, se tiver o valor ‘1’, transporta informação de controlo. O posicionamento da flag ACK assinala que o receptor do segmento deve considerar o conteúdo do campo número de sequência ACK como válido,

e ignorá-lo no caso contrário. Assim, os segmentos TCP, dependendo dos valores do campo *flags*, podem assumir o papel de uma unidade de informação apenas com dados, apenas de controlo, ou mista, transportando dados e controlo. Várias das outras *flags* são usadas durante a abertura e fecho da conexão e ainda em circunstâncias que serão detalhadas a seguir.

O campo *checksum* contém um código de controlo de erros, calculado pelo algoritmo apresentado na secção 2.4, que protege os cabeçalhos IP e TCP e o *payload* do segmento. Naturalmente, o valor do campo *checksum* é calculado usando 0 como o seu valor anterior. Adicionalmente, alguns valores dos campos do cabeçalho IP que variam durante a viagem do pacote também têm de ter valores especiais para o cálculo do *checksum*. Para efeito deste cálculo, é habitual designar estes cabeçalhos especiais por pseudo cabeçalhos (*pseudo headers*). O formato dos pseudo cabeçalhos varia nas versões 4 e 6 do protocolo IP.

Os campos *receiver advertised window* e *urgent pointer* serão explicados posteriormente e referem-se também à informação de controlo.

O campo *header length* contém a dimensão do cabeçalho em múltiplos de 4 bytes. Assim, a maior dimensão possível do cabeçalho TCP são $15 \times 4 = 60$ bytes. Sem opções, caso de muitos segmentos, o cabeçalho tem 20 bytes, e os restantes apenas são usados quando o campo de opções existe. Várias das opções são usadas durante a inicialização da conexão e permitem a ambas as partes acordarem os diversos parâmetros que a vão posteriormente caracterizar. Um desses parâmetros, transmitido nesse momento, é o tamanho máximo dos segmentos usados na conexão. Começaremos a discussão mais detalhada do protocolo por este aspecto.

Escolha do MSS (*Maximum Segment Size*)

Apesar de teoricamente cada segmento poder ter o comprimento total de 64 Kbytes, muitos dos canais não suportam *frames* com dados dessa dimensão. Sempre que são transmitidos pacotes que não cabem no *frame* máximo possível num canal, na versão 4 do protocolo IP o pacote é decomposto em vários fragmentos, *i.e.*, o pacote é fragmentado. Assim, o segmento tem de ser enviado em diversos fragmentos, alguns dos quais poderiam não chegar ao destino, ou chegarem por uma ordem diferente da com que foram emitidos. Apesar de o protocolo IP ter mecanismos, designados por **mecanismos de fragmentação**, que realizam a fragmentação automaticamente, a mesma é hoje em dia considerada uma má prática, e a versão 6 do protocolo nem a suporta na base.

Por isso, o TCP procura encontrar o valor máximo do comprimento dos segmentos a usar durante a conexão para que não seja necessário usar fragmentação. Esse parâmetro da conexão é designado por MSS, de *Maximum Segment Size*. O MSS corresponde à dimensão máxima do *payload* e não corresponde portanto à dimensão máxima do pacote que pode atravessar um canal. Por exemplo, num canal Ethernet, que comporta *frames* com no máximo 1500 bytes de dados, o MSS é de $1500 - 40 = 1460$ bytes (20 bytes para o cabeçalho IP e 20 bytes para o cabeçalho TCP, ambos sem opções). O valor na versão 6 do protocolo IP é diferente pois a dimensão do cabeçalho IP é nesse caso maior.

O valor do MSS depende das características do conjunto dos canais atravessados pelos pacotes enviados pelas duas extremidades, que não são conhecidos pelo TCP no momento do estabelecimento da conexão. No entanto, hoje em dia, os canais que impõem maiores limitações à dimensão máxima dos *frames* concentram-se geralmente na periferia e ligam os próprios computadores à rede, ou estão perto deles. Este facto permite a cada uma das partes questionar o seu nível rede sobre o melhor valor de MSS a usar, e comunicar à outra o valor que acha mais adequado no momento da abertura da conexão. O valor final retido para o MSS da conexão será o menor dos valores comunicados, ver a Figura 7.4.

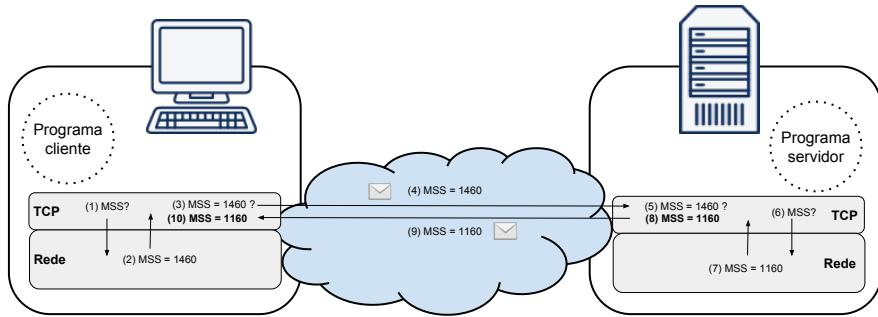


Figura 7.4: Na abertura da conexão, as partes acordam o valor do MSS (os números das legendas denotam a ordem das operações)

Como o valor do MSS escolhido na abertura da conexão não pode garantir que não exista necessidade de fragmentação, o protocolo TCP usa uma opção do protocolo IP (a única hipótese possível na versão 6 do protocolo IP) que indica que se for necessária fragmentação, a mesma deve ser rejeitada, e o emissor original do pacote avisado através de um pacote de notificação de erro. Este mecanismo, descrito no RFC 1191, para a versão 4 do protocolo IP e no RFC 1981, para a versão 6 do protocolo IP, permite afinar dinamicamente o valor do MSS. No entanto, o mesmo nem sempre funciona, pois em diversas circunstâncias os pacotes com notificações de erro são bloqueados¹. As versões mais modernas do protocolo TCP procuram afinar dinamicamente o valor do MSS quando detectam problemas, usando as técnicas descritas no RFC 4821.

Funcionamento do protocolo de janela deslizante

Por omissão, o protocolo TCP funciona segundo a estratégia GBN com valores do número de ACK cumulativos, que permitem janelas do emissor e do receptor maiores que o MSS e, opcionalmente, permitem também que o receptor guarde segmentos recebidos fora de ordem, como foi descrito na secção 6.3. No entanto, no protocolo TCP os números de sequência não identificam segmentos mas sim bytes.

Cada uma das partes fixa o seu valor inicial do número de sequência, designado ISN (de *Initial Sequence Number*), ver a Secção 7.3. O primeiro byte transmitido terá por número de sequência ISN+1, e o enésimo byte transmitido terá por número de sequência ISN+1+n. No exemplo da Figura 7.5 o número de sequência inicial é ISN+1 e o número de sequência lógico de cada byte i transmitido será ISN+1+i.

Quando é enviado um segmento, o número de sequência do segmento corresponde ao número de sequência do 1º byte contido no segmento. Por exemplo, seja 10 esse número; se o segmento contiver 5 bytes, o número de sequência do segmento seguinte será $10 + 5 + 1 = 16$ (ver a Figura 7.6). Por outro lado, o número de ACK não é o número de sequência dos bytes recebidos, mas o número de sequência do próximo byte a receber, *i.e.*, o número de sequência do último byte bem recebido + 1. No exemplo anterior, o número de ACK será 16 *i.e.*, o número de sequência do próximo byte esperado.

Cada segmento com dados pode ter de 1 a MSS bytes. O melhor rendimento é obtido com segmentos tão grandes quanto possível. No entanto, a aplicação escreve

¹O protocolo “auxiliar” do protocolo IP que permite reportar erros é o protocolo ICMP (*Internet Control Message Protocol*), já referido na Secção 3.3. O ICMP é muitas vezes bloqueado por razões de desempenho dos comutadores que detectam os erros, mas também de segurança, pois pode ser usado por um potencial atacante para obter informações sobre a configuração da rede e realizar ataques de negação de serviço aos comutadores.

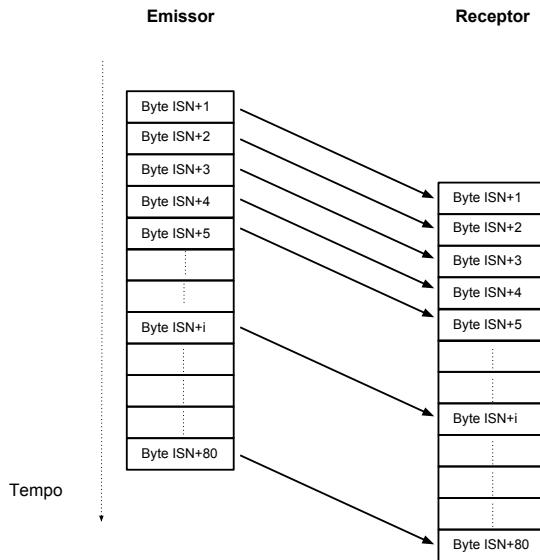


Figura 7.5: Números de sequência dos bytes no TCP

dados no canal conforme lhe é possível. Por exemplo, se a aplicação for uma aplicação do tipo sessão remota (*remote login*, *e.g.*, Telnet, *chat*), uma das mais populares aplicações usada nos primórdios da Internet, a aplicação vai obtendo bytes conforme o utilizador os escreve no teclado e o seu eco no monitor está dependente do computador remoto. Para obter o melhor desempenho possível, o TCP do lado do emissor teria de esperar até ter um número de bytes próximo do MSS. Contudo, isso poderá levar demasiado tempo, sobretudo quando a aplicação é interactiva. Como deve o TCP decidir quando deve emitir o próximo segmento?

O mecanismo usado pelo TCP é tentar esperar algum tempo para obter o máximo de bytes possível. Todavia, esse período de espera não pode exceder um valor máximo, geralmente 200 ms e, se o mesmo for ultrapassado, será enviado um segmento com os bytes disponíveis, independentemente da quantidade. A decisão de esperar ou não toma também em consideração se há ou não dados em trânsito e ainda não *acked*. O conjunto de estratégias usadas na emissão tem o nome de algoritmo de Nagle e o compasso de espera máximo foi fixado tendo em consideração vários factores, entre os quais a facilidade de implementação, que é condicionada pela granularidade dos temporizadores que é realista usar.

O socket do emissor admite que o programa, através da opção `SO_NODELAY`, force que cada escrita do programa, mesmo de um pequeno número de bytes, desencadeie o envio imediato de um segmento. A chamada `flush()` também permite ao programa solicitar o envio imediato de todos os bytes disponíveis no *buffer* de emissão. Este tipo de mecanismos são especialmente relevantes em programas interactivos. Quando os canais TCP mais não fazem que transferir dados continuamente, as opções por omissão desses mecanismos são adequadas e não necessitam de ser alteradas.

Uma questão simétrica, mas intimamente relacionada com esta, consiste em o receptor decidir quando envia um ACK. Na verdade, idealmente todos os ACKs deveriam ser enviados encavalitados nos dados enviados no sentido contrário, para evitar, tanto quanto possível, o envio de pequenos segmentos exclusivamente de controlo. Por isso, o TCP tenta esperar após a recepção de um segmento, até um máximo de 500 ms para ver se consegue transmitir o ACK encavalitado com dados enviados no sentido

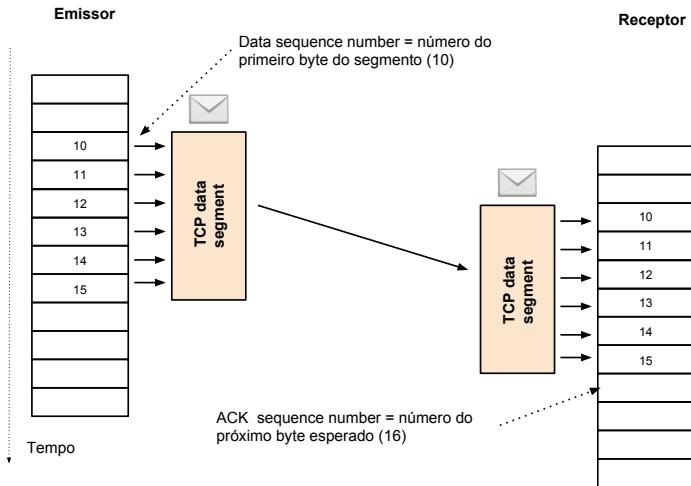


Figura 7.6: Números de sequência e de ACK dos segmentos

contrário. No entanto, se durante este compasso de espera for recebido um segundo segmento, um ACK é imediatamente transmitido mesmo que não hajam dados para transmitir no sentido contrário.

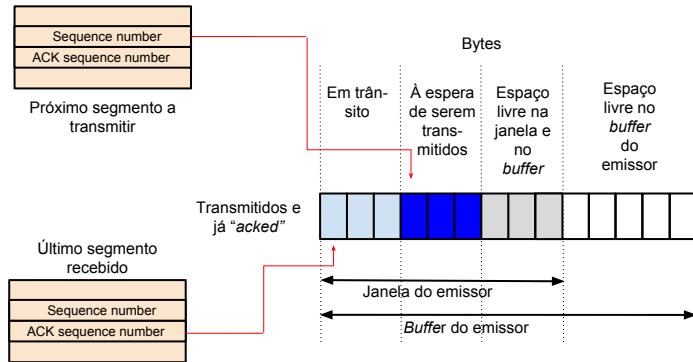
Estes mecanismos são importantes para melhorar o desempenho do protocolo, no entanto eles também pode ter um impacto negativo no desempenho de programas interactivos. As aplicações actuais (Web e nos sistemas móveis) procuram privilegiar, tanto quanto possível, o tratamento fino dos dados dos utilizadores localmente ao cliente. Esse tipo de estratégia minora este tipo de problemas, pois aumenta a dimensão das mensagens trocadas na conexão.

Como já foi dito, por omissão o protocolo funciona em modo GBN com ACKs cumulativos e o receptor pode guardar dados fora de ordem, apesar de a norma do protocolo não o impor. Em caso de alarme por ausência de ACK, são retransmitidos todos os bytes de que ainda não se receberam ACKs. Neste caso, os segmentos terão a dimensão tão próxima de MSS quanto possível, não interessando como os dados a transmitir foram escritos pelo programa emissor, ou originalmente segmentados para a transmissão. Este facto, assim como a discussão acima, mostra algumas das vantagens de o TCP não ter a noção de mensagem.

Como num protocolo desta natureza, o emissor e o receptor usam *buffers* de emissão e recepção, a dimensão dos mesmos é independente da janela de emissão para permitir o máximo de flexibilidade ao emissor e ao receptor, assim como aos respectivos programas. Um grande *buffer* de emissão permite ao TCP dar liberdade de escrita ao programa emissor e regular a dimensão dos segmentos como for mais adequado. Um grande *buffer* de recepção permite receber dados fora de ordem e acomodar alguma eventual lentidão do receptor. Ambas as dimensões têm valores por omissão que os programas podem modificar através de opções dos respectivos sockets.

É preciso, no entanto, ter em atenção que caso a rede tenha uma perda significativa de pacotes, o aumento dos *buffers*, em especial o de recepção, pode revelar-se negativo, pois pode aumentar o número de segmentos a retransmitir sempre que um alarme dispara. Este aspecto está relacionado com o mecanismo de controlo de fluxo do TCP que será discutido a seguir, e que também condiciona a dimensão da janela de emissão, *i.e.*, da quantidade de dados em trânsito.

A Figura 7.7 mostra o *buffer* do emissor e põe em evidência que a dimensão da janela de emissão não está necessariamente relacionada com a dimensão do *buffer*

Figura 7.7: Janela de emissão e *buffer* de emissão no TCP

de emissão. No TCP, como veremos no Capítulo 8, a janela de emissão é ajustada dinamicamente, enquanto que a dimensão do *buffer* de emissão é geralmente fixa.

Naturalmente, quando o emissor envia dados tem de instalar um alarme e executar o procedimento GBN quando o alarme dispara (*timeout event*). No protocolo não existe, neste modo de funcionamento, a noção de NACK. No entanto, quando recebe dados fora de ordem, o receptor envia sem demora um ACK cumulativo. Se o emissor continuar a enviar segmentos para a frente sem executar o procedimento GBN, esse ACK será provavelmente repetido. O emissor, utiliza um mecanismo, designado FAST RETRANSMIT, que consiste em desencadear imediatamente o mecanismo GBN caso receba 3 ACKs repetidos com o mesmo número de sequência.

A Tabela 7.1, apresenta de forma mais completa, uma panorâmica da gestão do envio dos ACKs pelo receptor, evidenciando as diferentes formas de este proceder.

Tabela 7.1: Gestão do envio dos ACKs pelo receptor TCP

Estado e evento	Acção
Segmentos recebidos na ordem e já <i>acked</i> e chegou um novo segmento	Esperar até 500 ms por outro segmento ou pela oportunidade de enviar um ACK encavalitado; se o tempo de espera expirar sem ser possível enviar um ACK encavalitado, enviar um ACK sozinho
Chegou mais um segmento durante o estado anterior, ou Chegou um segmento fora de ordem, ou Chegou um segmento que fecha total ou parcialmente um “buraco”	Enviar ACK cumulativo imediatamente

O funcionamento do mecanismo FAST RETRANSMIT está ilustrado na Figura 7.8. A eficácia do mecanismo é máxima com muitos dados transmitidos para a frente, janelas de emissão grandes, janelas de recepção pequenas e transferências de grandes quantidades de dados, pois tenta acelerar a recuperação de erros e evitar retransmissões inúteis. Com pequenas quantidades de dados, ou em aplicações interactivas, o mecanismo não consegue melhorar muito a situação e com RTT elevado não consegue

evitar a retransmissão, potencialmente inútil, de segmentos recebidos fora de ordem mas potencialmente guardados pelo receptor.

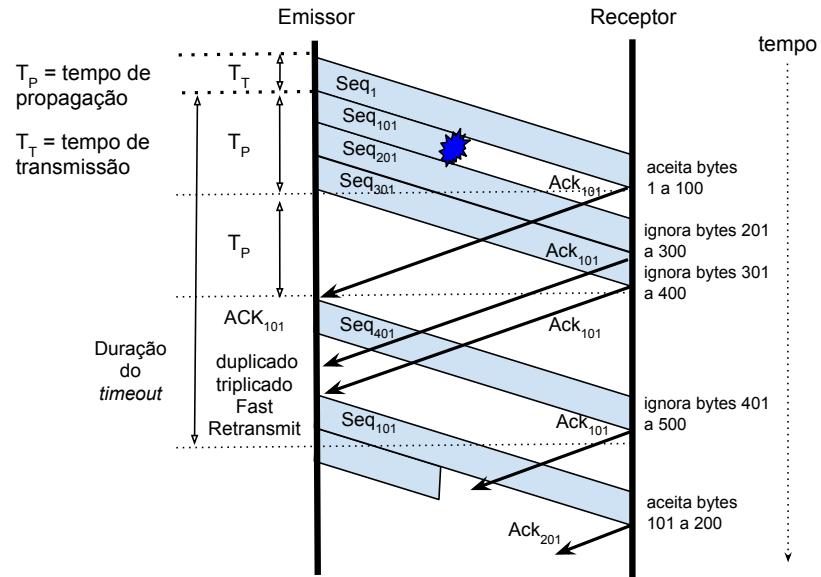


Figura 7.8: Janela de emissão e *buffer* de emissão no TCP

No protocolo TCP os números de sequência e de ACK são os números de ordem dos bytes no fluxo entre o emissor e o receptor. O número de sequência dos dados corresponde ao número de ordem do primeiro byte contido num segmento, e o número de sequência do ACK corresponde ao número de ordem do próximo byte a receber. O protocolo não impõe *a priori* nenhuma dimensão mínima, e em grande medida nem a máxima, aos segmentos que envia.

Por omissão, o protocolo é do tipo GBN com um funcionamento em que os números de ACK são cumulativos e o receptor pode guardar segmentos recebidos fora de ordem. Neste modo não existem NACKs, mas a recepção de 3 ACKs repetidos desencadeia imediatamente o procedimento GBN (FAST RETRANSMIT).

Vários mecanismos são usados para evitar enviar segmentos muito pequenos ou só com ACKs. Estes mecanismos passam pela introdução de compassos de espera limitados no tempo. Em programas interactivos os mesmos podem atrasar o progresso do protocolo, pelo que alguns podem ser desactivados. Adicionalmente, a interface também permite a manipulação da dimensão dos *buffers* de emissão e recepção.

Pelas razões atrás apresentadas, as versões mais modernas do protocolo admitem o funcionamento num modo semelhante ao *Selective Repeat*. Esta opção é comunicada à outra parte no momento da abertura da conexão. Caso ambas as partes aceitem esta forma de funcionamento, dependendo de eventuais perdas de segmentos, podem

ser enviados no campo de opções *Selective Acknowledgements*, também designados por SACKs.

Ou seja, para além de transmitir o ACK cumulativo normal, esta opção do protocolo usa o campo de opções para transmitir SACKs, *i.e.*, indicações sobre blocos de bytes correctamente recebidos para a frente, com a forma do número de sequência inicial e final de cada bloco recebido fora de ordem. Quando o receptor TCP recebe um ACK, retransmite imediatamente os dados em falta. Esta opção do protocolo está definida nos RFCs 2018 e 2883 e a sua disponibilidade é hoje em dia generalizada.

Opcionalmente, para melhorar o desempenho, o protocolo pode funcionar num modo semelhante ao *Selective Repeat*. A utilização desta opção, em conjunto com os números de sequência cumulativos normais, permite uma recuperação mais eficaz e rápida do que o mecanismo FAST RETRANSMIT, pois minora as retransmissões inúteis e repara mais rapidamente os erros.

Controlo de fluxo

O protocolo TCP inclui um mecanismo de controlo de fluxo baseado no campo *receiver advertised window*, ver a Figura 7.3. Este campo permite ao receptor TCP indicar, sempre que envia segmentos para o outro lado da conexão, qual o espaço que tem livre no seu *buffer* de recepção.

O protocolo impõe que cada emissor não deve ter dados *in-flight*, *i.e.*, transmitidos mas ainda não *acked*, em dimensão superior ao valor do campo *receiver advertised window* recebido da outra parte. Assim, o emissor tem de garantir que

$$\text{lastByteSentSequence} - \text{lastACKSequence} \leq \text{lastSeenAdvertisedWindow}$$

A Figura 7.9 mostra o cálculo deste parâmetro do lado do receptor. Caso a aplicação consuma poucos dados de cada vez, e lentamente, isso pode desencadear um fenômeno chamado *silly window syndrome*, que consiste em o emissor usar apenas segmentos de pequena dimensão, o que é mau para o desempenho do protocolo.

O algoritmo de Nagle, assim como o envio atrasado de ACKs, também foram introduzidos para combater o aparecimento deste fenômeno em situações em que o emissor transmitia, sem esperar, segmentos limitados por pequenos valores do campo *receiver advertised window*. Com efeito, sem esses mecanismos, emissor e receptor poderiam sincronizar-se e passar a utilizar apenas pequenos segmentos, mesmo com aplicações do lado do receptor que consumiam atempadamente todos os dados disponíveis no *buffer* de recepção.

Quando o valor do campo *receiver advertised window* tem o valor 0, o emissor tem de parar de enviar dados para o receptor. Se a situação persistir, poder-se-á ficar numa situação de bloqueio. Por isso o TCP do emissor usa um alarme para detectar essa situação e envia um segmento com poucos bytes quando o mesmo dispara, na esperança de que no entretanto a janela de recepção tenha alargado.

Numa situação em que o RTT é muito elevado, por exemplo de 100 ms, um emissor TCP só consegue tirar o rendimento máximo da conexão se transmitir continuamente até encher o canal lógico entre ele e o receptor. Daqui resulta que a janela de emissão tem de ter pelo menos a capacidade correspondente ao volume desse canal lógico. Isto é, deve poder transmitir continuamente até chegar o ACK do byte mais antigo transmitido, o que equivale a encher duas vezes o canal lógico entre as partes. A Tabela 7.2 ilustra o valor em bytes dessa janela, com um RTT de 100 ms, para alguns débitos de transmissão extremo e extremo significativos.

Como o campo *receiver advertised window* tem 16 bits, isso implicaria que a janela máxima de emissão do TCP fosse 64 Kbytes. Como este valor é muito baixo sempre

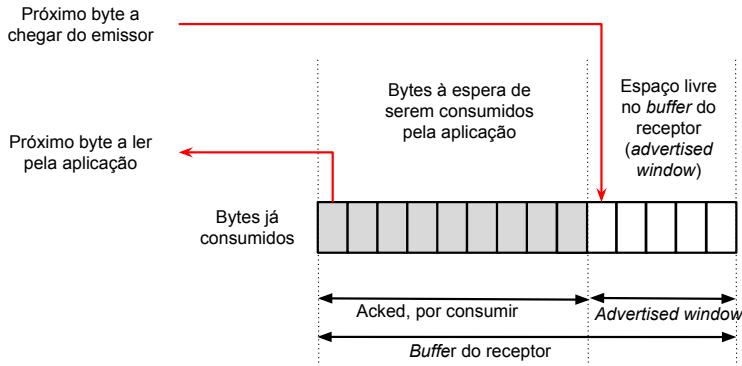


Figura 7.9: Janela de receptor e cálculo do valor de *receiver advertised window* no TCP

Tabela 7.2: Dimensão mínima aproximada da janela de emissão do TCP para aproveitar a capacidade de extremo a extremo quando o RTT é de 100 ms

Débito médio extremo a extremo	Dimensão mínima aproximada da janela TCP
100 Kbps	1250 bytes
1 Mbps	12,5 Kbytes
10 Mbps	125 Kbytes
100 Mbps	1,25 Mbytes
1000 Mbps	12,5 Mbytes

que o débito extremo seja igual ou superior a 5 Mbps (ver a Tabel 7.2), o TCP prevê uma opção chamada *window scaling option*. Durante a abertura da conexão, cada parte pode enviar à outra um valor de 0 a 14 bits no campo dessa opção que a outra parte deve usar para fazer um *shift* à esquerda dos valores do campo *receiver advertised window* que receber. Assim, se o valor desta opção para a outra extremidade for n , e o segmento recebido contiver no campo *receiver advertised window* o valor w , o receptor deve considerar que o valor do espaço livre no *buffer* de recepção da outra extremidade é $w \cdot 2^n$.

Determinação do valor do *timeout*

Como foi referido na secção 6.3, a utilização de um valor adequado do *timeout* é fundamental para o desempenho extremo a extremo dos protocolos de janela deslizante. O TCP estima o valor do RTT continuamente e usa as amostras colhidas para deduzir o valor do *timeout* a usar. Adicionalmente, o comportamento da conexão, do ponto de vista da perda de pacotes, também é usado para definir em cada momento o valor deste parâmetro.

Se a rede fosse uma rede que garantisse o tempo de trânsito de extremo a extremo, o valor do *timeout* poderia ser estimado de forma estática. Por exemplo, durante a abertura da conexão cada parte mediria o valor do RTT, e utilizaria daí para diante o dobro do valor medido como valor do *timeout* a usar.

Na verdade, o valor do RTT é variável pois o *jitter* pode ser significativo em

muitos momentos. Por outro lado, caso a conexão dure um tempo significativo, o valor médio pode ser bastante diferente se medido em períodos distintos. Tudo depende da competição pelos canais que os pacotes que transportam os segmentos da conexão encontrarem. Por exemplo, é frequente vários computadores partilharem o mesmo canal de ligação ao resto da rede. Se vários computadores abrirem simultaneamente conexões, o canal terá de ser partilhado por todas. Se o canal for limitado, em função da variação do número de conexões simultaneamente activas, a competição pelo mesmo irá também variando, assim como o estado da sua fila de espera.

Convém portanto usar um método não simplista para estimar o valor médio do RTT. Adicionalmente, para evitar retransmissões prematuras, é necessário tomar em consideração a sua variação a curto prazo.

Para estimar o valor médio do RTT, o TCP marca a hora de emissão de um segmento e mede o tempo que levou a chegar o respectivo ACK. Neste processo é necessário descartar as medidas em que o ACK foi recebido após uma retransmissão, assim como os ACKs que chegaram após ter sido aplicado algum atraso na sua emissão do lado do receptor. Por outro lado, não é possível ir simplesmente acumulando amostras e usar como RTT o seu valor médio. Por exemplo, se houvesse uma alteração sensível do RTT após as primeiras 1000 amostras, a alteração só teria repercussão na média depois de mais algumas centenas de amostras suplementares.

Para resolver este primeiro problema o TCP usa uma média ponderada entre o passado e o presente. Seja $nextRTTSample$ a última amostra obtida, e $avgRTT$ a média estimada até aí; então é possível usar o seguinte cálculo para actualizar o valor da média:

$$avgRTT = \alpha \cdot avgRTT + (1 - \alpha) \cdot nextRTTSample \quad (7.1)$$

A variável $avgRTT$ da equação acima chama-se no TCP *Smoothed RTT* ou SRTT. Este tipo de média chama-se uma **média móvel com ponderação exponencial, ou EWMA (Exponential Weighted Moving Average)**.

Para o peso do passado, o valor de α , toma-se $7/8 = 0,875$, e portanto $1 - \alpha = 0,125 = 1/8$. Estes valores foram escolhidos pois caso os valores do RTT sejam representados em inteiros, a afectação pode usar operações de *bit shift* de 3 posições, pois $8 = 2^3$. Este tipo de optimizações pode ser mais significativo em dispositivos computacionais sem muita capacidade de cálculo ou de energia. O valor de $avgRTT$ é inicializado a 3 segundos, e assim que é obtida a primeira medida, toma o valor do primeiro valor medido.

Adicionalmente, como o valor do SRTT é uma média e, por isso, muitas vezes inferior ao verdadeiro RTT, o *timeout* tem de ser calculado usando uma margem de segurança. Inicialmente o RTT medido era multiplicado por dois, mas verificou-se posteriormente que o valor escolhido era muito elevado se o RTT fosse estável e muito pequeno se o RTT variasse muito, pois quando os canais são de baixa capacidade e a sua utilização se aproxima da saturação, as filas de espera podem crescer muito e a variação do RTT ser muito maior que o próprio RTT.

Para lidar com estas situações, Van Jacobson introduziu o seguinte método para calcular o valor do *timeout* usando SRTT e a sua variação, também alisada (*Smoothed*), que é designado na fórmula abaixo SRTTVar (*Smoothed RTT Variation*):

$$timeout = SRTT + 4 \cdot SRTTVar \quad (7.2)$$

Para o cálculo de SRTTVar é usado um método semelhante ao usado para calcular SRTT:

$$SRTTVar = \beta \cdot SRTTVar + (1 - \beta) \cdot |SRTT - nextRTTSample| \quad (7.3)$$

com $\beta = 0,75$, $1 - \beta = 0,25$. O valor inicial de SRTTVar é $1/2$ do primeiro RTT medido.

Adicionalmente, sempre que se perdem pacotes, para cada uma das suas retransmissões é usado um valor de *timeout* igual ao valor anterior multiplicado por 2, até um limite superior. Assim, para evitar tanto quanto possível o envio de segmentos que provavelmente não chegarão ao destino devido a problemas na rede, o TCP retransmite os segmentos após o disparo de alarmes correspondentes a períodos de espera cada vez mais alargados.

Este tipo de gestão dos temporizadores é designada por “recurso exponencial” (*Exponential Backoff*). O RFC 6298 discute em detalhe a gestão do valor deste temporizador no TCP.

Dois aspectos suplementares merecem uma breve referência. O primeiro tem a ver com o facto de que para medir o RTT, o TCP não usa todos os segmentos e ACKs recebidos, mas apenas um subconjunto que garante que é realizada pelo menos uma medida em cada RTT. Se necessário, para realizar essa medida com precisão, é usada uma opção especial, chamada *timestamp option*, que permite colocar o valor de um relógio local ao emissor no campo de opções e receber um ACK que contém a mesma informação, transmitido assim que o segmento original chegou ao receptor. Assim, um emissor que use essa opção pode medir com rigor o RTT sem necessidade de sincronização de relógios, pois o receptor devolve imediatamente um ACK com o valor da *timestamp* enviada sem alterações, ou seja, a medição do RTT depende apenas do relógio do emissor.

O segundo aspecto tem a ver com a gestão de alarmes (*timeout*). Na secção 6.3 quando foi introduzido o protocolo SR, foi sugerido que o mesmo usa um alarme para cada pacote enviado pelo emissor. Numa conexão com um elevado RTT existem potencialmente muitos segmentos em trânsito, o que conduziria a uma grande quantidade de alarmes distintos, tanto mais que um computador pode ter várias conexões TCP activas simultaneamente. Os RFCs sugerem que cada conexão tenha apenas um alarme associado à necessidade de retransmissão. Esse alarme, tal como no protocolo GBN, está ligado ao segmento em trânsito mais antigo e ainda não *acked*, e é actualizado sempre que se recebe um ACK que faz avançar a janela “à esquerda”.

Dados urgentes

No cabeçalho dos segmentos TCP está presente o campo *urgent pointer*. O mesmo destina-se a assinalar que num segmento estão presentes dados considerados *out-of-band*, i.e., que a aplicação receptora deve tratar como dados urgentes e cujo tratamento deve ser tão rápido quanto possível. O campo *urgent pointer* é válido se a flag URG (de *URGENT*) estiver posicionada. Trata-se de um mecanismo que foi introduzido para enviar sinais urgentes à aplicação na outra extremidade da conexão, motivado pela necessidade de implementar a possibilidade de interromper a actividade normal em sessões remotas baseadas, por exemplo, no protocolo Telnet, ver o RFC 854, ou outros equivalentes (ssh, etc.).

Entre as *flags* presentes no cabeçalho, o emissor pode também posicionar uma designada PSH (de *PUSH*), que significa que os dados transmitidos no segmento devem ser transmitidos à aplicação com urgência assim que chegarem. Na interface de sockets do TCP este mecanismo não é acessível e não há nenhuma forma de um processo aceder a dados recebidos sem consumir todos os que os antecederam. Provavelmente esta opção está ligada ao facto de que a interface de sockets TCP, introduzida posteriormente ao protocolo, é orientada a *streams* e não comporta a noção de mensagens urgentes.

Ambos os mecanismos caíram em desuso e quando uma aplicação baseada em TCP necessita de usar formas de controlo deste tipo, geralmente usa duas conexões: uma para transferências normais de dados, e outra para transmissão de comandos e sinais. O protocolo FTP, ver o RFC 959, é um exemplo desta filosofia.

O protocolo TCP incorpora diversos mecanismos sofisticados para adequação dinâmica da velocidade de emissão à capacidade do receptor consumir os dados enviados (controlo de fluxo) e adaptação dinâmica do valor do *timeout* usado ao estado da rede.

Torna-se assim claro que se trata de um protocolo que tenta maximizar o desempenho através de mecanismos sofisticados de adaptação ao estado do receptor e da rede. Adicionalmente, como veremos no capítulo a seguir, incorpora ainda mecanismos de controlo da saturação da rede. Existem dezenas e dezenas de RFCs directamente ligados às diversas facetas do protocolo e à sua evolução desde que foi introduzido.

7.3 Abertura e fecho das conexões

Quando entre as partes em comunicação existe um só canal, por exemplo um canal ponto-a-ponto e *half-duplex*, é possível saber o tempo máximo durante o qual o canal pode “memorizar” pacotes (isto é, o tempo em que o sinal está em trânsito entre as extremidades do canal). Esse tempo está limitado pelo tempo de propagação. Adicionalmente, se a outra parte respondeu ao pacote p , então sabe-se que ela recebeu todos os pacotes enviados antes de p , ou nunca os receberá. Um canal não troca a ordem de entrega dos pacotes, só pode perdê-los.

No entanto, numa rede como a Internet, a rede pode memorizar pacotes durante algum tempo e trocar a sua ordem de entrega. Basta para isso que um comutador introduza atrasos extraordinários num pacote, ou que o caminho que estes seguem tenha sido alterado recentemente. Sendo assim, quando o emissor recebe a resposta ao pacote p , isso não garante que o receptor não receba posteriormente pacotes que tenham sido enviados antes de p , mas que este interpreta como tendo sido enviados depois.

Uma questão interessante que se pode colocar é: qual o tempo de vida máximo de um pacote dentro da Internet? Não é possível determinar esse valor, apenas é possível definir um valor a partir do qual a probabilidade de um pacote “perdido” ser mesmo assim entregue ser na prática nula. A esse valor chama-se MPL (*Maximum Packet Lifetime*) e no contexto do protocolo TCP, MSL (*Maximum Segment Lifetime*). Esse valor foi fixado na definição inicial do TCP no RFC 793 em 120 segundos para garantir uma segurança máxima.

Adicionalmente, por exemplo no fecho de uma conexão, é necessário a uma das partes (A), saber se a outra parte (B) está de acordo em fechar a conexão. Mas se B só puder fechar a conexão caso tenha a certeza que A recebeu a informação de que está de acordo em fechá-la, é necessário que A comunique a B que recebeu a sua resposta e tenha a certeza que B recebeu essa informação. Tal exige que B responda de novo a A e o problema não tem solução caso se possam perder pacotes.

Porque é que estas questões têm importância no protocolo TCP? Porque é necessário garantir que segmentos de uma conexão antiga não sejam confundidos com segmentos de uma conexão futura. O problema pode parecer bizarro, mas um protocolo só é robusto se resistir a todas as situações que possam suceder, por mais improváveis que pareçam. A questão prende-se intimamente com a escolha dos números de sequência iniciais, mas também com a velocidade com que os números de sequência são consumidos.

Valores iniciais dos números de sequência (ISN)

Os números iniciais de sequência, designados por ISN, de *Initial Sequence Numbers*, são estabelecidos por ambas as partes no momento do estabelecimento da conexão. Numa primeira aproximação, ambas as partes poderiam usar 0 como ISN. Na verdade isso criaria dois problemas: a entrada, por acidente, de segmentos de uma conexão antiga muito curta, numa conexão futura; e o facilitar a tentativa deliberada de um atacante de introduzir dados estranhos numa conexão. Comecemos por discutir o primeiro problema.

Existem duas formas de evitar esse problema. A primeira consiste em não reutilizar os endereços IP e as portas da conexão anterior, pelo menos até terem passado MSL segundos. Dos quatro valores que caracterizam uma conexão, apenas a porta inicial proposta pela parte que abre a conexão pode facilmente não ser reutilizada numa situação em que o mesmo computador abre sucessivas conexões para o mesmo servidor, para aceder ao mesmo serviço. Por exemplo, um cliente Web faz sucessivos pedidos HTTP (todos para a porta 80) ao mesmo servidor, usando sucessivas conexões, o que constitui uma situação frequente.

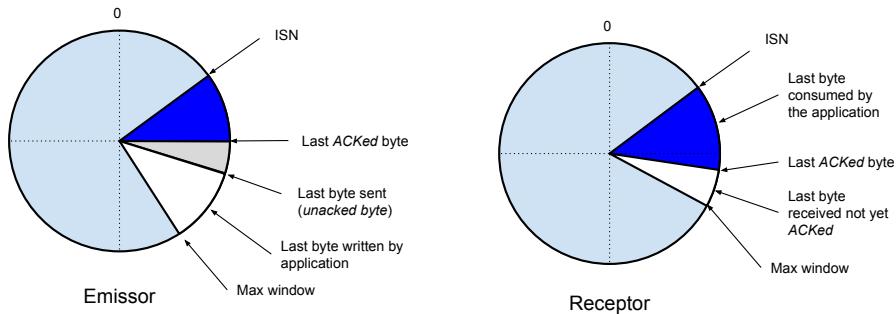


Figura 7.10: Visão da utilização dos números de sequência por cada um dos lados da conexão TCP como um relógio

A outra maneira de evitar confusão consiste em usar como ISN na conexão seguinte um valor superior ao número de sequência de qualquer dos números de sequência usados pelos bytes da conexão anterior. Este cálculo teria de ser realizado tendo em consideração que os números de sequência podem dar a volta. Na verdade os mesmos podem ser vistos como evoluindo numa circunferência, no sentido dos ponteiros do relógio, ver Figura 7.10. O ISN determina a posição inicial. Ao longo da vida da conexão os números de sequência vão crescendo no sentido dos ponteiros do relógio, até darem a volta.

Ambas as soluções (não reutilização de portas ou de números de sequência) podem ser aplicadas se as partes memorizarem alguma informação sobre as conexões antigas durante MSL segundos, e se não for possível a uma conexão consumir números de sequência a uma tal velocidade que os mesmos voltem ao início da janela em menos de MSL segundos. Infelizmente, dar a volta completa em menos de MSL segundos é “fácil” com grandes débitos, pois com um débito de extremo a extremo de 1 Gbps, os números de sequência dão uma volta completa ao “relógio” em 34 segundos, ver o RFC 1323.

Tudo indicaria ser fácil memorizar uma porta e não a reutilizar durante MSL segundos para resolver a confusão entre duas conexões distintas. Mas mesmo isso pode não ser suficiente e uma conexão suficientemente rápida pode ser induzida em erro por ela própria.

No entanto, mesmo que uma conexão leve mais do 120 segundos a dar a volta, é

sempre possível memorizar as portas ou os números de sequência das conexões fechadas nos últimos MSL segundos? A resposta é não, pois o cliente ou o servidor podem “morrer” repentinamente, perder a memória e reincarnar imediatamente a seguir. Ou outro computador poderá tomar o seu lugar e usar o mesmo endereço IP. Em ambos os casos a confusão entre duas conexões distintas pode acontecer.

Admitindo que memorizar informação sobre as conexões passadas chegaria para resolver o problema, existe uma solução fácil para o problema da “morte súbita”, que consiste em obrigar todos os computadores a levarem mais do que MSL segundos, após o seu arranque, para iniciarem a primeira conexão TCP. Dado que o MSL corresponde, por segurança, a várias dezenas de segundos, esta solução também não é em geral aceitável.

No RFC 1323 foram introduzidos um conjunto de mecanismos, baseados num relógio que se admite que não pára, mesmo quando um computador está em baixo, e na opção *timestamp* que permitem às partes trocar o valor desse relógio no início da conexão, no campo de opções. A título de parêntesis, interessa referir que o mesmo mecanismo permite também estender o número de sequência com mais 32 bits e resolver a ambiguidade com os números de sequência quando estes dão a volta, o que pode suceder facilmente em conexões de alto débito.

No entanto, nenhuma destas soluções resolvia o segundo problema, pois tornava mais fácil a um atacante adivinhar os números de sequência iniciais das conexões e enviar segmentos falsos, capazes de violar a fiabilidade das transferências.

Para tentar resolver os vários problemas enunciados, o TCP usa actualmente uma técnica de geração de números pseudo aleatórios para os valores do ISN, procura não reutilizar os números das portas usadas nas últimas conexões durante MSL segundos após o seu fecho, e usa o mecanismo das *timestamps* para troca dos valores do relógio e para proteger conexões de alta velocidade e medir mais rigorosamente o RTT.

Veremos a seguir que o TCP pode sofrer um ataque de negação de serviço durante a abertura da conexão, cuja prevenção complementa este tipo de precauções do lado da parte que aceita a conexão, usando técnicas criptográficas para calcular o ISN dessa parte.

A determinação dos ISNs (*Initial Sequence Numbers*) de uma conexão é um problema delicado quando se pretende assegurar, em todas as situações, a fiabilidade dos dados por esta transferidos. Para colmatar este problema, o TCP usa actualmente um método de determinação do ISN baseado em números pseudo aleatórios e um mecanismo de troca de valores do relógio na abertura das conexões e durante o funcionamento das mesmas.

Abertura de uma conexão

Para o estabelecimento das conexões, o TCP usa um mecanismo, chamado um “aperto de mão em 3 fases” (*three-way handshake*), baseado na troca de 3 segmentos. Para além de estes segmentos serem usados para trocar os valores dos ISNs, o MSS, *etc.* são também usadas as opções de passagem dos valores do relógio para evitar confusões com segmentos antigos. A utilização do *three-way handshake* é necessária para garantir que ambas as partes viram todas as informações uma da outra. Com um *two-way handshake*, a parte que aceita a conexão não teria a certeza de que duplicados de pedidos de conexão, ou duplicados de ACK, não conduziriam ao estabelecimento de mais do que uma conexão. Com efeito, dado que no protocolo TCP a parte que aceita o estabelecimento da conexão a esquece após o seu fecho, caso uma conexão durasse muito pouco tempo, um duplicado do pedido de abertura muito antigo poderia conduzir a parte que aceita a abertura da conexão a julgar que estaria perante uma conexão nova. Com o *three-way handshake*, essa parte só reconhece a conexão como aberta no fim do processo completo.

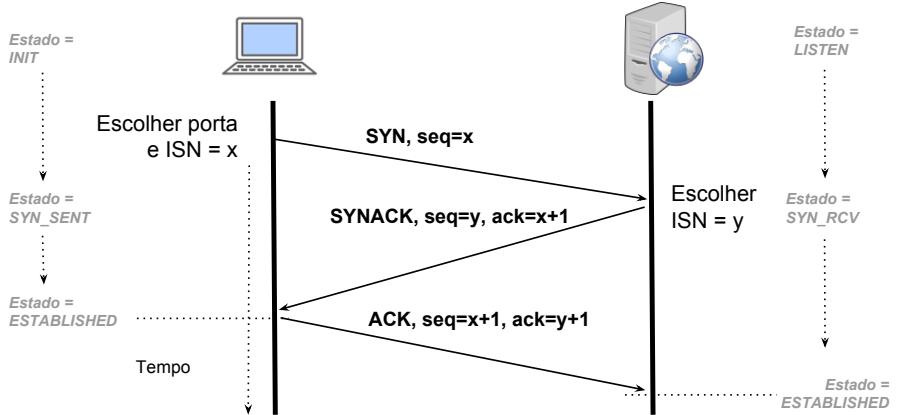


Figura 7.11: Abertura de uma conexão TCP. À esquerda a parte que solicita a abertura, à direita a que aceita a conexão

A parte que desencadeia o pedido de conexão é geralmente o cliente. A parte que atende pedidos de conexão é geralmente o servidor. Essa segunda parte criou um socket para atendimento de pedidos de conexão que está num estado, chamado LISTEN, em que aceita pedidos para novas conexões, ver a Secção 5.3. Uma conexão nessa situação costuma dizer-se que está “meio aberta”, num estado chamado “abertura passiva” (*passive open*). A parte que desencadeia a abertura da conexão diz-se que executa uma abertura activa (*active open*).

Para o estabelecimento da conexão são usadas as *flags* SYN (de *synchronize*) e ACK, como ilustrado na Figura 7.11. Inicialmente a parte que realiza a abertura activa da conexão envia um segmento com a *flag* SYN assinalada e propondo o seu número de sequência inicial (*ISN = x*). Este segmento chama-se um segmento SYN. O segmento SYN não pode transportar dados, apenas opções (MSS, window scale, timestamp, SACK suportado).

Caso a outra parte não tenha um socket no estado LISTEN associado à mesma porta, deve responder imediatamente com um segmento de erro, com a *flag* RST (RESET) posicionada. Esse segmento chama-se um segmento RST. O segmento RST também é enviado se o limite de pedidos de conexão à espera de serem atendidos pelo servidor, no mesmo socket, ultrapassou o seu limite. Este limite tem um valor por omissão que pode ser modificado através de opções do socket do servidor.

Logo que for possível atender o pedido, a parte que realizou a abertura passiva deve responder com um segmento com as *flags* SYN e ACK posicionadas, propondo o seu número de sequência inicial (*ISN = y*) e enviando um ACK do SYN recebido (com o valor $x + 1$). Este segmento chama-se um segmento SYNACK. O segmento SYNACK também não tem dados, apenas opções.

Quando recebe o segmento SYNACK, a parte que realizou a abertura activa responde com um segmento com a *flag* ACK posicionada e com o valor de ACK adequado ($y + 1$), e considera a conexão estabelecida. Este último segmento já poderia conter dados, mas a maioria das implementações enviam um segmento imediato só com o ACK. Se por acaso este último segmento ACK se perder, o próximo segmento que esta parte enviar com dados conterá necessariamente o ACK do segmento SYNACK.

Como o significado do número de sequência de um ACK é o número de ordem do próximo byte esperado, ambas as partes incrementam nos ACKs os ISNs recebidos e o número de ordem do primeiro byte transmitido por cada parte é sempre o seu ISN + 1.

O processo de abertura de uma conexão leva as partes a passarem por vários estados: CLOSED, LISTEN, SYN RECEIVED, SYN SENT e ESTABLISHED. O RFC do TCP contém uma máquina de estados que descreve o processo de abertura. A Figura 7.12 contém o subconjunto mais significativo dos eventos / acções e transições de estado executadas. O diagrama é, no essencial, auto-explicativo.

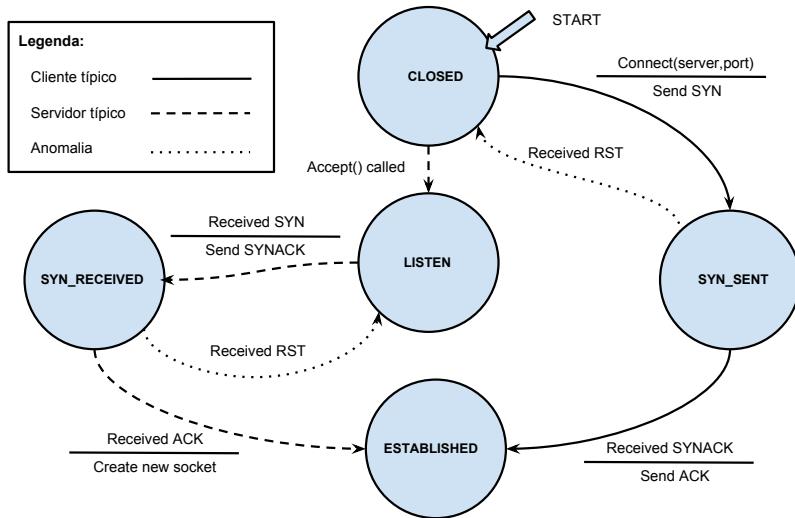


Figura 7.12: Máquina de estados simplificada correspondente à abertura de uma conexão TCP

Quando recebe o segmento SYN, a parte que está com a conexão no estado meio aberta (LISTEN), cria o estado possível para a conexão, regista as opções e afecta um *buffer* à mesma. Quando recebe o segmento SYNACK, a parte que está com a conexão no estado meio aberto (SYN_SENT), cria o estado para a conexão, regista as opções, afecta um *buffer* à conexão e considera-a aberta (ESTABLISHED).

Se o segmento SYN se perder, a parte que realiza a abertura activa reenvia-o após um *timeout*, geralmente de 3 segundos, pois no início nada se sabe sobre o RTT. Caso este primeiro segmento se perda, a abertura da conexão leva mais do que 3 segundos. Por isso, recentemente este *timeout* foi baixado para 1 segundo ou mesmo menos. Ao fim de um certo número de tentativas sem resposta, a conexão é considerada como impossível de estabelecer. Caso o segmento SYNACK se perda, a parte que realiza a abertura activa reenvia o segmento SYN, e o servidor reconhece uma conexão no estado SYN RECEIVED e não uma nova. O temporizador para o servidor desistir e esquecer uma conexão no estado SYN RECEIVED dura geralmente mais de 10 segundos.

Se for um *browser* Web que tenta abrir uma conexão e esta leva muito tempo a ser estabelecida, o utilizador tenta novamente aceder à página, obrigando o *browser* a abandonar a antiga conexão e a tentar abrir uma nova. A antiga tentativa é abandonada, e o cliente terá de usar necessariamente uma nova porta e outro número de sequência, pelo que o servidor verá sempre um novo pedido, e o estado da antiga será, mais tarde ou mais cedo, limpo através do temporizador que limita o tempo que uma conexão pode estar no estado SYN RECEIVED sem se completar.

Infelizmente, se o cliente enviar uma rajada de segmentos SYN diferentes, o servidor esgotará rapidamente a sua tabela de conexões (a maioria das conexões estarão no estado SYN RECEIVED). Trata-se de um ataque de negação de serviço, que se chama *SYN Flood Attack*, e que será discutido a seguir.

Se o último ACK se perder, como já foi referido, o cliente enviará mais tarde ou mais cedo dados que conterão esse ACK. Se não o fizer a breve trecho, o servidor esquecerá a conexão, e responderá posteriormente com um segmento RST quando receber dados referentes à mesma.

Compete à parte que abre activamente a conexão refazer os pedidos de abertura de uma conexão até conseguir uma resposta, ou desistir. A outra parte, passivamente, apenas responde a esses pedidos e espera que obtenha um ACK à sua resposta. Atendendo ao conjunto de mecanismos de discriminação de uma conexão (endereços IP, portas, ISNs e *time stamps*), não pode resultar confusão entre conexões. No máximo, existirão conexões no estado meio abertas que serão expurgadas pelo temporizador associado. Sem o *three way handshake*, o número de conexões no estado ESTABLISHED, mas inactivas, poderia ser muito maior.

Fecho de uma conexão

Quando se pretende fechar uma conexão, coloca-se o problema de saber se ainda há dados que a outra parte possa ainda querer enviar. Pretender que ambas as partes se ponham de acordo sobre um fecho simultâneo da conexão, porque já não têm nada a dizer uma à outra, cria um problema delicado de consenso que, como já referimos, é difícil de resolver quando a rede pode perder pacotes. Assim, é comum usarem-se protocolos de terminação abrupta de conexões, com eventual perda de dados já escritos por uma das partes, mas ainda não recebidos pela outra. Este efeito é possível no TCP usando segmentos RST que terminam abruptamente conexões.

No entanto, nos casos normais, para evitar terminações abruptas de conexões, o TCP usa um protocolo em que as duas partes de uma conexão fecham de forma independente cada metade da mesma. Assim, se uma das partes já não tem nada a transmitir, e tem a certeza que a outra parte já recebeu os seus dados, fecha a conexão do seu lado, mas pode continuar a ler os dados que a outra parte lhe continuar a enviar. Quando esta tiver a certeza que todos os seus dados foram bem recebidos, fecha por sua vez a conexão. Os sockets TCP dispõem da chamada `shutdown()`, que permite fechar um socket só para escrita, só para leitura, ou para ambas as funcionalidades. A chamada `close()` impede qualquer futuro acesso ao socket pelo processo que a invocar, e implicitamente invoca a funcionalidade de `shutdown()` se este for o único processo com acesso ao socket.

Para fechar uma conexão, cada uma das partes envia um segmento com a *flag FIN* (de *Finalize*) assinalada. Este segmento tem de ser *acked* como normalmente. O mesmo pode ser reenviado um certo números de vezes se necessário. A outra parte, quando desejar, deve fechar a conexão executando o protocolo simétrico. Ver a Figura 7.13.

Associado ao fecho de uma conexão existe igualmente uma máquina de estados que está ilustrada de forma simplificada na Figura 7.14. Dado que ambas as partes podem decidir fechar a conexão de forma independente, em momentos diferentes, existem três possibilidades diferentes de fechar a conexão:

Um lado fecha primeiro ESTABLISHED → FIN_WAIT1 → FIN_WAIT2 → TIME_WAIT → CLOSED

O outro lado fecha primeiro ESTABLISHED → CLOSE_WAIT → LAST_ACK → CLOSED

Ambos fecham simultaneamente ESTABLISHED → FIN_WAIT1 → CLOSING → TIME_WAIT → CLOSED

Como se pode verificar pelo diagrama de estados (Figura 7.14) e pelas transições acima, a parte que desencadeia o fecho da conexão (geralmente o cliente) passa pelo estado TIME_WAIT, onde permanece $2 \times MSL$ segundos, um período de tempo significativo (com dezenas de segundos). Este mecanismo garante que a conexão que

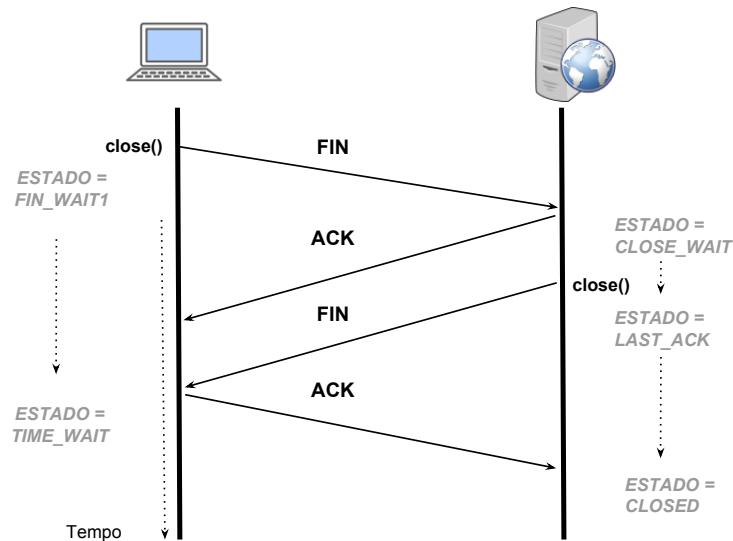


Figura 7.13: Protocolo de fecho de uma conexão TCP

foi fechada é memorizada durante $2 \times \text{MSL}$ segundos suplementares, o que assegura que não pode haver confusão entre conexões devido a segmentos atrasados.

Todas estas precauções com o fecho das conexões levam a um fecho relativamente “longo” e que guarda estado sobre uma conexão, para todos os efeitos já inativa, durante muito tempo. Isso pode tornar-se problemático num servidor Web que aceita vários milhares de conexões curtas por minuto. Por isso, alguns desses servidores fecham as conexões dos clientes usando segmentos com a flag RST posicionada para fechar abruptamente as conexões. Isso pode não implicar necessariamente a perda de dados dado que o protocolo HTTP, ao nível aplicacional, permite controlar se há ou não mais dados para receber.

O processo de abertura e fecho de uma conexão é relativamente pesado, envolvendo a troca de pelo menos 3 segmentos para a abertura e 4 para o fecho. A abertura é também impossível num período inferior a um RTT, durante o qual não é possível as partes trocarem dados. Se uma conexão for usada para transferência de pequenas quantidades de dados, o desperdício pode ser elevado.

Por isso, protocolos cliente / servidor como o do DNS realizam as suas transações sobre UDP, pois a resposta funciona como ACK do pedido. Atendendo a que os pacotes se podem perder, este tipo de solução pode resultar na repetição das operações, o que não é grave se estas não alterarem o estado do servidor (*e.g.*, leituras ou outras operações idempotentes).

O protocolo TCP é fiável, evita esses problemas e tem grandes vantagens quando ambas as partes pretendem transferir quantias significativas de dados. É claro que não se pode deixar de referir, que se uma conexão terminar abruptamente, o emissor não tem a certeza sobre se os dados chegaram ou não ao receptor.

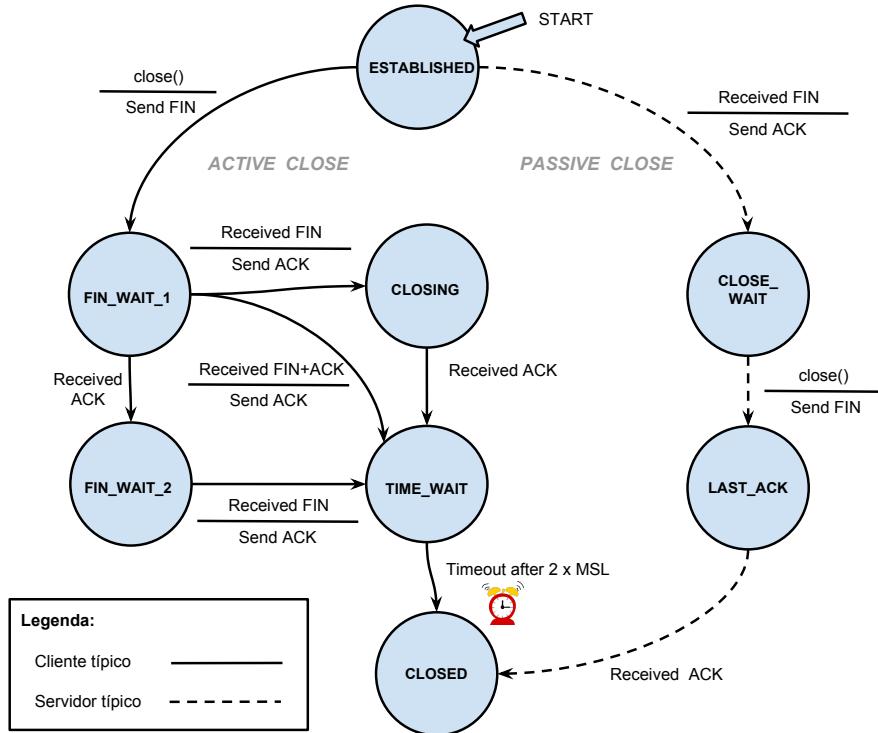


Figura 7.14: Máquina de estados correspondente ao fecho de uma conexão TCP

Ataque SYN Flood

Na implementação inicial do protocolo TCP, assim que a parte com conexões em estado passivo recebia o segmento SYN, criava todo o estado local necessário à futura conexão TCP. Isso incluía não só criar uma entrada na tabela de conexões, como afetar-lhe os dois *buffers*, de recepção e emissão.

Como na Internet é possível enviar pacotes com endereço e porta origem falsos sem que isso impeça muitas vezes o pacote de chegar ao destino, a partir de certa altura foi descoberta uma forma de atacar servidores TCP bem conhecidos, enviando-lhes uma grande quantidade de segmentos SYN com origem (do ponto de vista do servidor) em muitos clientes diferentes pois os diferentes endereços origem eram falsos.

O resultado era que o servidor esgotava rapidamente as tabelas de conexões, todas no estado SYN_RCV. Naturalmente, o atacante nem sequer recebia os segmentos SYNACK, mas mesmo que recebesse não estaria interessado em responder.

Trata-se de um ataque de negação de serviço, designado ataque por inundação de SYNs (*SYN Flood Attack*), que impede rápida e facilmente o servidor de servir clientes normais, sem custos significativos para o atacante.

Mais uma vez se revelou o inconveniente de ser possível usar endereços origem falsos para realizar ataques de negação de serviço, porque os ISPs não são obrigados a controlá-los. Com efeito, um ISP que realize esse controlo, não se está a defender, apenas está a contribuir para a defesa dos clientes dos seus concorrentes. Este tipo de situações costuma chamar-se de “tragédia dos comuns”.

Para confrinhar este ataque, foi desenvolvida uma técnica, designada *SYN Cookies*, descrita no apêndice do RFC 4987, que consiste em o servidor não memorizar senão a informação mínima sobre o cliente e devolver-lhe o seu ISN calculado criptografica-

mente, através da seguinte função:

$$\text{server_ISN} = F(\text{connection parameters}, \text{Secret_Key})$$

em que F é uma função de *hash* criptográfica e o parâmetro *Secret_Key* é um segredo que só o servidor conhece. O ISN enviado é assim computacionalmente impossível de reproduzir por alguém que desconheça a chave secreta do servidor.

Quando o cliente responder ao segmento SYNACK com um segmento ACK, se responder, o servidor decrementa o valor do campo ACK recebido, procura na sua tabela especial o pedido de conexão enviado com o SYN, e completa então o processo de estabelecimento da conexão pois sabe que o cliente existe.

Muitos ataques de negação de serviço são combatidos decrementando o ritmo por unidade de tempo dos novos eventos suspeitos que são aceites. Os que ultrapassarem esse ritmo são ignorados. O problema fundamental deste tipo de estratégia é que é necessário minorar os pedidos de clientes honestos que são suprimidos, por serem confundidos com pedidos de atacantes (chamados falsos positivos). A estratégia descrita acima permite identificar, com elevada probabilidade de não errar, os pedidos do atacante.

O RFC 4987 discute este ataque e o conjunto de técnicas que podem ser usadas para o combater.

Análise simplificada do desempenho do TCP

O desempenho do TCP é muito influenciado pelos mecanismos de recuperação de erros e de controlo da saturação da rede, que serão discutidos no capítulo 8. No entanto, a menos da influência desses mecanismos, é desde já possível estimar qual o desempenho máximo teórico.

Com efeito, admitindo que é possível desprezar o tempo de transmissão dos segmentos e que o RTT é significativo, *i.e.*, num cenário que os canais têm débitos elevados (*e.g.*, superiores a dezenas de Mbps) e o RTT é algumas dezenas de milissegundos, e admitindo também que não há perda de segmentos nem de ACKs, a velocidade média de transferência, ou débito médio, extremo a extremo, é dado por:

$$\bar{V} = \frac{\bar{W}}{\bar{RTT}}$$

ou seja, o débito médio é a dimensão média em bits da janela a dividir pelo RTT médio. Naturalmente, este débito também não pode ser superior ao débito do canal que liga o emissor à rede.

Assim, numa situação ideal, sem erros, sem perda de segmentos e sem interferência dos mecanismos de controlo de fluxo, a janela do emissor, assim como o RTT, têm uma influência decisiva sobre o desempenho do protocolo. Nestes cenários são necessárias janelas de emissão de dimensão elevada se os débitos e o RTT forem significativos, é necessário que o receptor as permita utilizar (o mecanismo de controlo de fluxo limita a janela do emissor ao espaço livre na do receptor) e é também necessário que a implementação de todos os mecanismos do TCP consiga funcionar atempada e sustentadamente.

7.4 Resumo e referências

Resumo

O protocolo TCP disponibiliza canais lógicos fiáveis e bidireccionais, entre dois processos, no mesmo ou em computadores distintos, designados conexões TCP. As conexões

TCP transferem sequências de um ou mais bytes, sem qualquer relação entre a dimensão das sequências emitidas e a das recebidas, *i.e.*, o protocolo não tem a noção de mensagem.

O protocolo inclui um mecanismo de controlo de fluxo e um mecanismo de controlo de saturação da rede. Os programas utilizadores não podem solicitar ou impor os valores do débito extremo a extremo pois, deste ponto de vista, o protocolo também é baseado na noção de “melhor esforço”.

Uma conexão TCP é caracterizada pelos endereços IP e portas associadas aos sockets em ambas as extremidades. Duas conexões distintas têm necessariamente um ou mais desses identificadores distintos.

No protocolo TCP os números de sequência e de ACK são os números de ordem dos bytes no fluxo entre o emissor e o receptor. O número de sequência dos dados corresponde ao número de ordem do primeiro byte contido num segmento, e o número de sequência do ACK corresponde ao número de ordem do próximo byte a receber. O protocolo não impõe *a priori* nenhuma dimensão mínima, e em grande medida nem a máxima, aos segmentos que envia.

Por omissão, o protocolo é do tipo *Go Back N* com um funcionamento em que os números de ACK são cumulativos, e o receptor pode guardar segmentos recebidos fora de ordem. Neste modo não existem NACKs, mas a recepção de 3 ACKs repetidos desencadeia imediatamente o procedimento GBN (FAST RETRANSMIT).

Vários mecanismos são usados para evitar enviar segmentos muito pequenos ou só com ACKs. Estes mecanismos passam pela introdução de compassos de espera limitados no tempo. Em programas interactivos os mesmos podem atrasar o progresso do protocolo pelo que alguns podem ser desactivados. Adicionalmente, a interface também permite a manipulação da dimensão dos *buffers* de emissão e recepção.

Opcionalmente, para melhorar o desempenho, o protocolo pode funcionar num modo semelhante ao *Selective Repeat*. A utilização desta opção em conjunto com os números de sequência cumulativos normais permite uma recuperação mais eficaz e rápida do que o mecanismo FAST RETRANSMIT, pois minora as retransmissões inúteis e repara mais rapidamente os erros.

O protocolo TCP incorpora diversos mecanismos sofisticados para adequação dinâmica da velocidade de emissão à capacidade do receptor consumir os dados enviados (controlo de fluxo) e adaptação dinâmica do valor do *timeout* usado ao estado da rede.

Torna-se assim claro que se trata de um protocolo que tenta maximizar o desempenho através de mecanismos sofisticados de adaptação ao estado do receptor e da rede. Adicionalmente, como veremos no capítulo a seguir, incorpora ainda mecanismos de controlo da saturação da rede. Existem dezenas e dezenas de RFCs directamente ligados às diversas facetas do protocolo e à sua evolução desde que foi introduzido.

A determinação dos ISNs (*Initial Sequence Numbers*) de uma conexão é um problema delicado, quando se pretende assegurar, em todas as situações, a fiabilidade dos dados por esta transferidos. Para colmatar este problema, o TCP usa actualmente um método de determinação do ISN baseado em números pseudo aleatórios e um mecanismo de troca de valores do relógio na abertura das conexões e durante o funcionamento das mesmas. Adicionalmente, no fecho de uma conexão procura-se assegurar, durante um lapso de tempo significativo, que a mesma não é esquecida, e as suas portas reutilizadas.

O processo de abertura e fecho de uma conexão é relativamente pesado, envolvendo a troca de pelo menos 3 segmentos para a abertura, e 4 para o fecho. A abertura dura pelo menos um RTT, durante o qual não é possível as partes trocarem dados. Se uma conexão for usada para transferência de pequenas quantidades de dados, o desperdício pode ser elevado.

Por isso, protocolos cliente / servidor como o do DNS realizam as suas transacções sobre UDP, pois a resposta funciona como ACK do pedido. Atendendo a que os pacotes se podem perder, este tipo de solução pode resultar na repetição das operações, o

que não é grave se estas não alterarem o estado do servidor (*e.g.*, leituras ou outras operações idempotentes).

O protocolo TCP é fiável, evita esses problemas e tem grandes vantagens quando ambas as partes pretendem transferir quantias significativas de dados. É claro que não se pode deixar de referir que se uma conexão terminar abruptamente, o emissor não sabe se os dados chegaram ou não ao receptor.

Os segmentos TCP, ver a Figura 7.3, podem transportar simultaneamente dados e informação de controlo. A codificação da informação de controlo usa vários campos, o conjunto de *flags* que figura na Tabela 7.3 e o conjunto de opções que figura na Tabela 7.4.

Tabela 7.3: Tabela de *flags* do cabeçalho TCP

<i>flag</i>	Valor	Descrição
URG	0x20	Urgent data present
ACK	0x10	ACK field is valid
PUSH	0x08	Push data to the receiver
RST	0x04	Reset connection
SYN	0x02	Synchronise connection
FIN	0x01	Finalize connection
CWR	0x80	Reduce cong. window (congestion related, RFC 3168)
ECE	0x40	ECN Echo (congestion related, RFC 3168)

Tabela 7.4: Tabela de opções do cabeçalho TCP

Opção	Descrição
1	Maximum segment size
2	Window scale
3	Selective ACK permitted
4	Timestamp

Alguns dos termos introduzidos com um significado específico no contexto do TCP são a seguir passados em revista.

Receiver Advertised Window Campo do cabeçalho TCP que permite ao emissor de um segmento indicar à outra parte qual o espaço livre no seu *buffer* de recepção, para que desta forma limite a quantidade máxima de bytes em trânsito que lhe são dirigidos.

FAST RETRANSMIT Depois de emitir vários segmentos, o emissor quando recebe três ACKs seguidos com o mesmo número de sequência, conclui que o receptor perdeu algum segmento e inicia o processo de retransmissão GBN.

ISN *Initial Sequence Number*. Número de sequência do primeiro segmento enviado por cada extremo da conexão TCP. O seu valor é calculado de forma aleatória de maneira a minimizar a probabilidade de um segmento atrasado, de uma antiga conexão, ser confundido com um segmento da conexão que se inicia. Adicionalmente, esta forma de escolha do valor do ISN tenta também minimizar a probabilidade de um atacante acertar em quais são os números de sequência usados por uma conexão.

MSL *Maximum Segment Lifetime*. Tempo de vida máximo de um segmento dentro da rede. Este conceito tem relevância para segmentos anormais cuja entrega é atrasada pela rede. É usado para garantir que um segmento atrasado não pode ser confundido com um segmento emitido posteriormente, mas com o mesmo número de sequência.

MSS *Maximum Segment Size*. Dimensão máxima dos dados (*payload*) de um segmento de uma conexão TCP. O seu valor é fixado de tal forma que se procura que um segmento caiba sempre dentro de um *frame* em todos os canais que tem de atravessar.

SACK *Selective ACK*. Opção do cabeçalho TCP que permite ao receptor sinalizar a correcta recepção de um ou mais segmentos fora de ordem.

Syn-flood Attack Forma de ataque em que o atacante inunda a vítima com segmentos SYN com endereço de origem falso e não responde aos SYNACKs, obrigando-a a esgotar o limite de conexões meio abertas.

Three Way Handshake Protocolo envolvendo o envio de 3 segmentos usado pelo TCP para abrir uma conexão.

Urgent Pointer Campo do cabeçalho TCP que permite ao emissor assinalar ao receptor a presença de dados *Out-of-Band* no segmento. Trata-se de um mecanismo usado predominantemente por aplicações interactivas como Telnet ou ssh.

Referências e apontadores

Vinton Gray Cerf e Robert E. Kahn receberam em 2004 o Prémio Turing Award da ACM (*Association for Computer Machinery*) e passamos a citar: “for pioneering work on internetworking, including the design and implementation of the Internet’s basic communications protocols, TCP/IP, and for inspired leadership in networking.” Ambos são os autores de um artigo, [Cerf and Kahn, 1974], que introduziu formalmente na literatura científica o protocolo TCP.

As publicações sobre o protocolo TCP e a sua evolução são inúmeras. No entanto, o livro [Stevens and Wright, 1994] continua a ser considerado a obra mais completa sobre os mecanismos do protocolo TCP, pelo menos até à publicação da sua última edição, pois o seu primeiro autor faleceu em 1999. O livro não só apresenta os conceitos fundamentais como os ilustra através de inúmeras experiências.

Existem imensas ferramentas que permitem medir o desempenho do protocolo TCP, nomeadamente os programas **iperf**, já referido, e **TTCP** (*Test TCP*). Richard Stevens desenvolveu um programa especial, o programa **sock**, para teste e experimentação com as diferentes opções dos protocolos UDP e TCP acessíveis via os respectivos sockets. O *packet sniffer* **wireshark** tem imensas opções para análise dos segmentos TCP e estudo do comportamento de uma conexão TCP. O programa **netstat**, disponível em praticamente todos os sistemas de operação, permite obter informações sobre as conexões TCP do computador onde o mesmo executa. **Web10G** é um módulo para o núcleo dos sistemas Linux que permite aceder directamente ao valor de todas as variáveis que caracterizam as conexões TCP activas. É especialmente adequado para o estudo dos factores que podem estar a limitar o desempenho das conexões TCP de um sistema Linux.

Apontadores para informação na Web

- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- <http://en.wikipedia.org/wiki/Ttcp> – Contém informação sobre o programa TTCP.

- <http://en.wikipedia.org/wiki/Iperf> – Contém informação sobre o programa iperf.
- <http://www.icir.org/christian/sock.html> – Contém informação sobre o programa sock de Richard Stevens.
- <https://www.wireshark.org> – Contém informação sobre o wireshark.
- <http://en.wikipedia.org/wiki/Netstat> – Contém informação sobre o programa netstat para diferentes sistemas de operação.
- <http://www.web10g.org> – Contém informação sobre o módulo Web10G.

7.5 Questões para revisão e estudo

1. Nos sistemas de operação, os processos em execução têm um número, geralmente chamado o PID (*Process Identifier*) do processo. Alguém sugeriu que esse número era mais interessante para discriminar as conexões TCP do que as portas. Acha a sugestão boa? Justifique a sua resposta.
2. Verdade ou mentira? Justifique a resposta.
 - (a) Para abrir e fechar as conexões, o protocolo TCP usa segmentos de controlo com cabeçalhos de formato diferente dos cabeçalhos dos segmentos de dados.
 - (b) Todas as conexões TCP usam 0 como número de sequência inicial.
 - (c) No protocolo TCP o número de sequência de um segmento é igual ao número inicial (ISN) mais o número de segmentos já emitidos.
 - (d) O protocolo TCP garante que a dimensão da janela do emissor é constante.
 - (e) O protocolo TCP exige que a dimensão da janela do receptor seja sempre constante.
 - (f) Quando executa o procedimento GBN, o protocolo TCP reemite exactamente os mesmos segmentos, com os mesmos números de sequência, que os que já tinha emitido previamente.
3. Verdade ou mentira? Justifique a resposta.
 - (a) Os canais TCP não garantem a entrega ao receptor dos pacotes transmitidos pelo emissor, nem sequer na ordem pela qual os mesmos foram transmitidos.
 - (b) Os canais TCP implementam um canal lógico de transmissão entre o emissor e o receptor cujo débito é constante.
 - (c) Uma conexão TCP é fiável de extremo a extremo, isto é, não se perdem dados, porque os comutadores de pacotes da rede garantem que todos os pacotes que lhes chegam são entregues intactos.
 - (d) Uma conexão TCP é assegurada pela rede através de uma concatenação de sub-canais físicos, chamada um circuito, que liga de forma dedicada o emissor ao receptor, e desta forma garante que todos os segmentos TCP chegam sempre ao destino.
 - (e) Uma conexão TCP é assegurada pelo sistema de operação dos computadores em diálogo, através de um protocolo que pressupõe que a rede pode perder e trocar a ordem dos pacotes IP.
4. Verdade ou mentira? Justifique a resposta.

- (a) Um computador A está a enviar um ficheiro para o computador B através de uma conexão TCP. Supondo que o computador B também tem dados para enviar para A, então B envia necessariamente para A dois tipos de segmentos: segmentos com dados e segmentos com ACKs.
- (b) O valor do campo *Receiver Advertised Window* do cabeçalho TCP nunca se altera nos diferentes segmentos usados durante toda a duração de uma conexão TCP.
- (c) Um computador A está a enviar um ficheiro para o computador B através de uma conexão TCP. Supondo que o número de sequência de um segmento enviado de A para B é N, então o número de sequência do próximo segmento enviado por A é necessariamente N+1.
- (d) Um computador A está a enviar um ficheiro muito grande para o computador B através de uma conexão TCP. Supondo que o computador B não tem dados para enviar para A, então B não envia ACKs para A pois não pode enviar ACKs encavalitados em segmentos com dados.
5. Verdade ou mentira? Justifique a resposta.
- (a) Numa conexão TCP, o *timeout* do emissor é apenas activado no fim da transmissão de toda a janela.
- (b) Num pacote IP são encapsulados um ou mais segmentos TCP.
- (c) Numa conexão TCP, o valor do *timeout* do emissor é quase sempre constante.
- (d) Numa conexão TCP, quando não se está a usar a opção SACK do protocolo, a janela do receptor tem sempre a dimensão do MSS.
- (e) Numa conexão TCP, o mecanismo FAST RETRANSMIT foi introduzido porque se pressupõe que a janela do receptor tem sempre a dimensão do MSS.
6. Indique quais das seguintes razões justificam que o número de sequência inicial de cada um dos lados de uma conexão TCP seja um número pseudo-aleatório. Justifique a sua resposta.
- (a) Por segurança, na medida em que assim os atacantes não conseguem adivinhar esse número de sequência.
- (b) Para evitar que algum pacote antigo possa ser confundido como um segmento da nova conexão.
- (c) Para identificar o momento em que a conexão foi aberta.
- (d) Porque a lógica dos protocolos de janela deslizante assim o exige.
- (e) Para diminuir a eficácia de um ataque de negação de serviço do tipo *SYN Flood Attack*.
7. O emissor TCP emite segmentos logo que recebe dados escritos pela aplicação? Se não é o caso, indique porque razão não é assim, e como é que este procede.
8. Diga para que serve a opção SO_NODELAY aplicada a um socket TCP.
9. O receptor TCP emite segmentos só com ACKs logo que recebe segmentos vindos da rede? Se não é o caso, indique porque razão não é assim, e como é que este procede.
10. Diga em que consiste a anomalia *silly window* do TCP.
11. O protocolo TCP não suporta mensagens de controlo NACK. No entanto, se considerarmos todos os mecanismos obrigatórios e opcionais, o TCP dispõe de dois mecanismos que substituem os NACKs. Quais são e como funcionam?

12. Um computador A abriu uma conexão para o computador B e fez o *download* de um ficheiro com 1024 bytes. Quantos segmentos trocaram A e B para realizar esta operação sabendo que o MSS da conexão era de 1460 bytes e que não se perderam segmentos?
13. Na situação da questão anterior, sabendo que o tempo de propagação medido de A para B era de 100 ms, e era idêntico ao que B mediou para A, indique quanto tempo levou a transferência (desde ao início da abertura da conexão até à recepção do último byte do ficheiro).
14. A regulação do valor dos alarmes (*timeouts*) usados pelo emissor de segmentos TCP é particularmente delicada.
 - (a) Indique os inconvenientes de um alarme cuja duração é demasiado curta.
 - (b) Indique os inconvenientes de um alarme cuja duração é demasiado longa.
 - (c) Indique resumidamente como é que no protocolo TCP a duração do alarme usado pelo emissor é calculado.
15. Considere uma conexão TCP a funcionar num contexto em que o RTT é de 100 ms e o débito máximo possível entre os extremos é de 100 Mbps. Admita que não pode usar a opção *Window Scaling Factor*.
 - (a) Qual o débito máximo teórico que essa conexão TCP pode atingir?
 - (b) O que faria para tentar melhorar o seu desempenho?
16. A implementação do protocolo TCP no servidor C é tal que uma conexão é fechada se esta não receber ou emitir segmentos durante 5 minutos. Invente uma forma de o TCP de um cliente de C evitar esse fecho, mesmo que os programas que estão a usar a conexão não necessitem de trocar dados durante 5 minutos. A sua solução não pode violar os requisitos de fiabilidade do protocolo TCP e deve ser transparente para as aplicações nos extremos da conexão.
17. Durante o funcionamento de uma conexão TCP todos os segmentos contêm um campo de nome *Receiver Advertised Window* que é sempre válido (pois não há nenhuma *flag* e indicar a sua validade ou não). É possível esse campo tomar o valor 0? Como é que o emissor que recebe um segmento com esse valor a 0 quebra a potencial situação de bloqueio em que pode ficar?
18. Considere uma conexão TCP entre dois computadores A e B com o MSS = 1024 bytes. O computador A transmite um ficheiro de 1.000 Kbytes para B. Admitindo que não há perda de pacotes na rede, estime a quantidade de segmentos transmitidos de B para A só com ACKs, e com ACKs e dados. Considere também os segmentos necessários para a abertura e fecho da conexão.
19. Na abertura de uma conexão TCP é fixado um valor de MSS.
 - (a) Admitindo que o MSS recomendado pelo nível rede das duas partes é maior que a dimensão do maior *frame* que cabe num dos canais atravessado pelos segmentos da conexão, o *Three-Way Handshake* inicial provoca obrigatoriamente fragmentação?
 - (b) Depois da conexão estabelecida, o MSS pode variar mais tarde quando se usa a versão 4 do protocolo IP?
 - (c) Depois da conexão estabelecida, o MSS pode variar mais tarde quando se usa a versão 6 do protocolo IP?
20. Durante a abertura de uma conexão TCP pretende-se avaliar o MSS a usar. Para isso, o lado que abre a conexão poderia começar por enviar um segmento TCP SYN em que a dimensão do segmento é artificialmente fixada como sendo a maior dimensão de *frame* da sua interface de ligação ao resto da rede. Descreva

como esse lado poderia usar o protocolo ICMP para descobrir o MSS correcto que deveria usar. Descreva também o que deve fazer a parte que aceita a conexão, com o mesmo objectivo, quando envia o segmento SYNACK de resposta.

21. Considere um ficheiro de dimensão D bytes a transferir de A para B através de TCP. Assuma que o MSS da conexão TCP a utilizar é 1000 bytes.
 - (a) Qual é a menor valor de D que obriga a que os números de sequência do TCP sejam reutilizados? Para efeito dos seus cálculos admita que $2^{10} \approx 10^3$, $2^{20} \approx 10^6$ e $2^{30} \approx 10^9$.
 - (b) Quantos segundos leva a transmitir o ficheiro com a dimensão calculada na alínea anterior, admitindo que o emissor e o receptor têm *buffers* infinitos, que não há perda de pacotes, que o RTT é 10 ms, que o débito máximo de extremo a extremo é de 100 Mbps, e que não pode usar a opção *Window Scaling*.
22. Indique porque razão o protocolo TCP, em particular a usar a opção SACK, não pode ter uma janela de emissão maior que o maior intervalo de números de sequência a dividir por 2, ou seja, 2^{31} .
23. Indique porque razão o mecanismo FAST RETRANSMIT do protocolo TCP desencadeia o procedimento GBN após três ACKs repetidos e não dois.
24. Considere uma conexão TCP entre os processos A e B ligados através da Internet. O módulo TCP do lado do processo A, que abriu a conexão e foi o primeiro a fechá-la, está no estado FIN_WAIT_2 e a seguir passou ao estado TIME_WAIT, onde se manteve por mais algumas dezenas de segundos. Porquê?

Capítulo 8

Controlo da saturação da rede

The scientist described what is: the engineer creates what never was.

– Autor: *Theodor von Karman – the father of the supersonic flight*

Como todos os sistemas construídos pelo homem, uma rede de computadores tem um capacidade limitada e um nível de utilização em que o seu rendimento é máximo. Muitos sistemas dispõem de mecanismos de *feedback* que permitem aos utilizadores procurarem o nível de solicitação ideal para a sua capacidade e para maximizarem o seu rendimento. Por exemplo, a maioria dos automóveis modernos dispõem de indicadores da velocidade e de consumo instantâneo que permitem ao condutor ajustar a sua condução a diferentes relações velocidade / consumo da viatura. Será fácil fazer algo de semelhante numa rede?

Numa rede de computadores em que a soma das capacidades dos canais de acesso dos computadores seja superior à capacidade de encaminhamento efectivo de pacotes pela rede, que é a situação mais frequente, é possível ao conjunto dos computadores injectarem pacotes a um débito superior ao que a rede é capaz de encaminhar. Numa rede IP essa situação irá traduzir-se na supressão de pacotes pelos comutadores. No entanto, a verdade é que nada impede um computador de aumentar o ritmo com que envia pacotes até atingir o débito máximo do canal que o liga à rede. Mas, se o fizer, está a melhorar o seu desempenho? A resposta depende do caminho tomado pelos seus pacotes e da actividade dos outros computadores com os quais está em competição.

Na grande maioria das situações, o aumento do débito de emissão dos computadores ligados a uma rede IP para além do que esta suporta, torna-se negativo para todos os computadores, incluindo os mais “gulosos”, e o respectivo débito de extremo a extremo até pode decrescer.

Idealmente, os computadores deveriam tentar não fazer solicitações para além do nível em que o rendimento da rede atinge o seu máximo. Infelizmente, em quase todas as redes, os comutadores no interior da rede ao detectarem a situação de saturação não têm capacidade de refrear os computadores.

É possível estabelecer uma analogia directa com o tráfego viário. Quando o ritmo dos automóveis que entram numa rede viária é superior ao que esta é capaz de acomodar, o resultado são engarrafamentos como os da Figura 8.1. Actualmente os sistemas de controlo do tráfego urbano até são capazes de detectar essas situações (*e.g.*, através de câmaras) mas de facto não têm muita capacidade para impedir os automóveis de

tentarem entrar nas zonas engarrafadas e agravarem a situação.¹



Figura 8.1: Um engarrafamento de tráfego automóvel devido à chegada de mais veículos por unidade de tempo do que aquele que essa zona da rede viária é capaz de acomodar

Nas primeiras redes de circuitos cada cliente só podia utilizar a capacidade reservada para o seu circuito, e um circuito só era estabelecido se a rede tivesse capacidade disponível para o acomodar. O problema da saturação² era resolvido negando a entrada de mais clientes, mesmo que houvesse capacidade para os acomodar tendo em consideração o consumo real dos circuitos activos. No outro extremo, as redes IP iniciais não impunham nenhum limite, voluntário ou involuntário, ao tráfego injectado na rede. O resultado foi catastrófico, e com a subida da sua popularidade, a Internet em geral exibia níveis de saturação muito elevados e detectavam-se quebras bruscas de débito extremo a extremo de 100 para 1. O comportamento era frequentemente o de uma rede viária em colapso pois nada progredia significativamente.

Esse problema foi resolvido através da introdução de mecanismos de controlo da saturação.

Os **mecanismos de controlo da saturação** (*congestion control mechanisms*) são mecanismos que procuram controlar **dinamicamente** as solicitações feitas à rede, de forma a maximizar o rendimento da mesma, tendo em consideração a utilização real e a capacidade disponível.

Os mecanismos de controlo da saturação serão tanto mais adequados quanto menos realizarem reservas *a priori*, quanto mais promoverem a equidade, e quanto mais permitirem um melhor aproveitamento de toda a capacidade disponível.

¹ Existem, no entanto, planos para conseguir alterar automaticamente os trajectos de automóveis sem condutor em função do estado da rede viária.

²O termo usado em inglês é *congestion*, que pode ser traduzido por “saturação”, “congestão” ou “engarrafamento”. Geralmente, “congestão” é usado como sinónimo de afluência anormal de sangue a um órgão e “engarrafamento” como afluência anormal de veículos. O termo saturação é o mais transversal a diversos campos e por isso preferimos usá-lo.

8.1 Em que consiste a saturação

O desempenho das redes de computadores é estudado formalmente usando a teoria das filas de espera. À luz desta teoria o desempenho de um serviço é caracterizado pela taxa de trabalho produzido por unidade de tempo. A letra que se costuma usar para denotar essa grandeza é a letra grega λ (lambda). Aplicada a uma rede de computadores, o trabalho produzido é medido em unidades de informação entregues ao destino por unidade de tempo. Se todos os pacotes fossem da mesma dimensão, pode-se usar a unidade pacotes por segundo (pps). No caso mais geral deve-se usar a unidade bits por segundo (bps).

Se olharmos para a rede como uma caixa preta, ver a Figura 8.2, num determinado período de observação os computadores injectam na rede uma certa quantidade de pacotes, da qual resulta o tráfego médio injectado na rede $T_{in} = \sum \lambda_{in}(i)$, onde $\lambda_{in}(i)$ denota a informação injectada por unidade de tempo em cada canal de entrada. No mesmo período a rede entrega aos computadores que lhe estão ligados o tráfego médio $T_{out} = \sum \lambda_{out}(i)$, onde $\lambda_{out}(i)$ denota a informação entregue ao destino por unidade de tempo através de cada canal de saída.

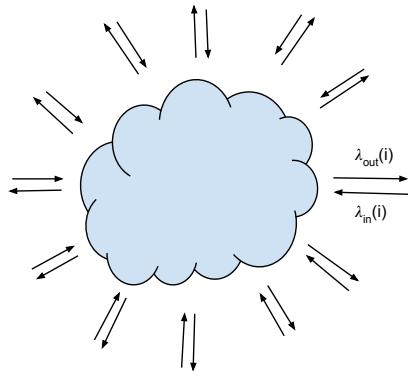
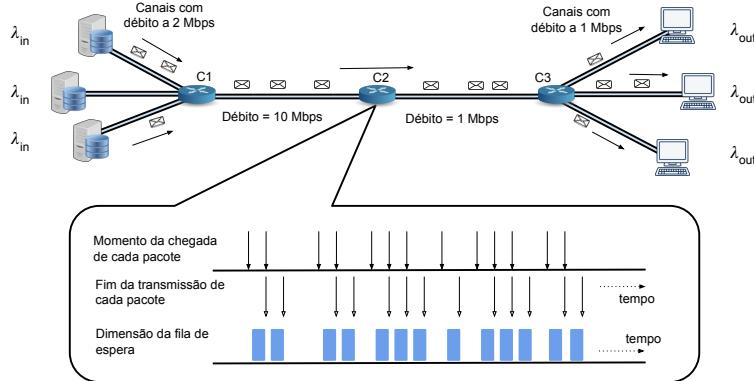


Figura 8.2: Tráfego injectado e tráfego entregue num determinado período

Em nenhum momento $T_{out} > T_{in}$ pois, por hipótese³, a rede não gera ela própria pacotes. Se tudo estiver a funcionar bem $T_{out} \approx T_{in}$ pois alguns pacotes perdem-se por erros nos canais. Quando a rede se aproxima da saturação T_{out} começa a ser significativamente inferior a T_{in} . Numa rede completamente saturada, existe um grande desperdício e $T_{out} \ll T_{in}$. Vamos a seguir ver porquê.

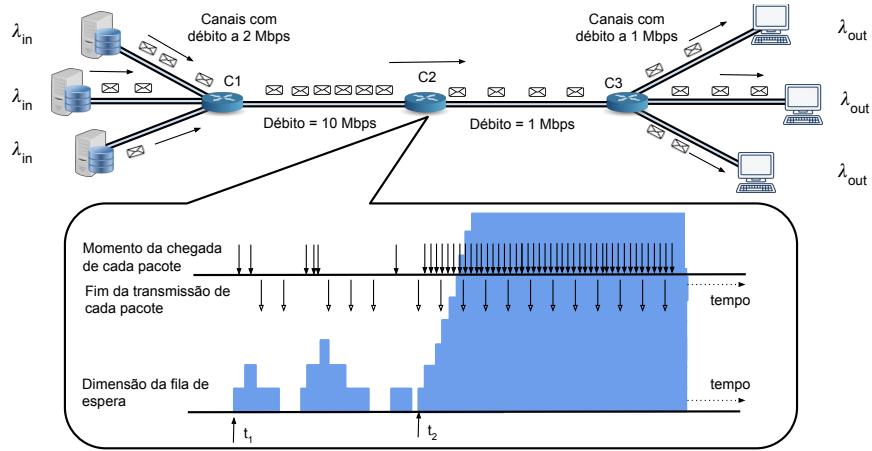
Para esse efeito vamos usar um modelo de rede simples como o da Figura 8.3. Os computadores à esquerda, os servidores, só injectam tráfego na rede, e este é sempre dirigido aos computadores da direita, os clientes. Todos os canais são ponto-a-ponto, bidirecionais e *full-duplex*. Os dois canais que ligam os três comutadores entre si, modelizam uma rede de trânsito simplificada. Os canais que ligam os computadores a um comutador modelizam duas redes de acesso, uma à esquerda e outra à direita. O débito agregado dos canais de acesso dos servidores (3×2 Mbps) é inferior ao do canal que liga os comutadores C1 a C2, que tem o débito de 10 Mbps. O débito agregado dos canais de acesso dos clientes (3×1 Mbps) é superior ao do canal de trânsito que liga os comutadores C2 a C3, com o débito de 1 Mbps e que é o canal de gargalo (*bottleneck link*) da rede. Admite-se que não existem erros nos canais

³ O tráfego *multicast* e *broadcast* viola esta hipótese pois a injeção de um único pacote, pode conduzir à geração de muitos outros.

Figura 8.3: Filas de espera no interior da rede com $T_{out} = T_{in}$

pelo que estes não perdem pacotes. Por hipótese também, caso os emissores usem um protocolo que emita ACKs, o tempo de transmissão desses pacotes é desprezável e a rede encaminha-os sem problemas dos clientes para os servidores. Finalmente, considera-se igualmente que é realista desprezar o espaço ocupado pelos cabeçalhos e considera-se que os pacotes de dados só transportam informação útil.

A Figura 8.3 mostra igualmente a chegada de pacotes ao comutador C2 e o estado da fila de espera do seu canal de saída. Na situação ilustrada na figura, quando chega um novo pacote, este encontra sempre a fila de espera vazia, isto é, todos os pacotes chegados anteriormente já foram transmitidos. Nesta situação a fila de espera só tem no máximo um pacote, exactamente o pacote que está a ser transmitido, e o atraso máximo introduzido pelo comutador C2 é o tempo de transmissão do pacote em trânsito. A rede está muito folgada, com pouca carga e encaminha sem problemas até ao destino os pacotes que nela entram e portanto $T_{out} = T_{in}$.

Figura 8.4: Filas de espera no interior da rede com $T_{out} = T_{in}$ de t_1 a t_2 e $T_{out} > T_{in}$ a partir de t_2

A Figura 8.4 mostra uma situação em que os servidores começaram a aumentar o ritmo com que injectam pacotes dirigidos aos clientes. Entre os momentos t_1 e t_2 alguns pacotes chegam em grupos ao comutador C2, e a fila de espera de espera cresce momentaneamente, mas é sempre possível transmitir cada grupo de pacotes antes que chegue o grupo seguinte. Nesse período continua a verificar-se $T_{out} = T_{in}$ mas alguns pacotes começam a ser encaminhados com um tempo de trânsito extremo a extremo variável e maior que anteriormente (*i.e.*, aparece *jitter*).

No entanto, a partir do momento t_2 , chega um grande grupo de pacotes contíguos e a fila de espera começa a crescer significativamente. Admitindo que essa fila de espera é infinita, mesmo que a chegada ininterrupta de pacotes continue, a rede vai continuar a entregar todos pacotes, mas estará limitada ao débito do canal gargalo (*bottleneck link*). Seja qual for a taxa com que chegam pacotes ao comutador, este só entrega pacotes com o débito de 1 Mbps e nessa altura $T_{in} > T_{out} = 1 \text{ Mbps}$. Naturalmente, a fila de espera vai crescendo sem limite. Admite-se igualmente que o protocolo usado pelos emissores é ingênuo e continua a emitir pacotes seja qual for a situação na fila de espera e o tempo de trânsito extremo a extremo experimentado pelos pacotes.

Define-se débito útil de informação ou *goodput* como sendo o débito útil de informação entregue pela rede aos destinatários, isto é, descontando todo o desperdício introduzido pelos protocolos, quer em termos de cabeçalhos dos pacotes, quer em termos de pacotes reemittidos e recebidos em duplicado (neste modelo, como já referimos, desprezam-se os cabeçalhos dos pacotes).

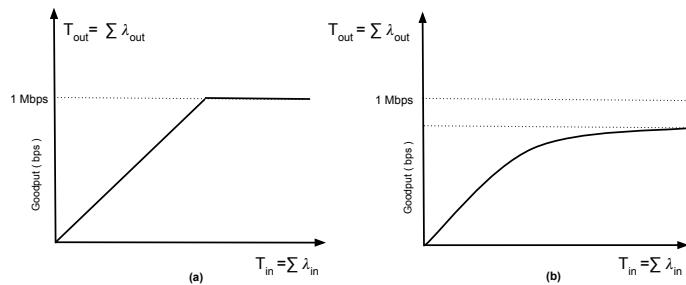


Figura 8.5: Débito útil (*goodput*) com filas de espera infinitas: (a) sem retransmissão de pacotes pelos emissores, (b) com retransmissão prematura de pacotes

A Figura 8.5 mostra a relação do *goodput* com T_{in} em duas situações diferentes, correspondentes a dois comportamentos distintos dos emissores. Em ambas as situações, sempre que $T_{in} < 1\text{Mbps}$, a rede está pouco carregada e verifica-se que $T_{out} = goodput = T_{in}$. No entanto, no caso do gráfico (a), sempre que $T_{in} > 1\text{Mbps}$, *goodput* = 1Mbps pois esse é o limite da capacidade de encaminhamento da rede de trânsito, mas a fila de espera de C2 vai crescendo e o tempo de trânsito extremo a extremo também. O gráfico (a) corresponde à situação acima referida, em que a fila de espera do comutador é infinita e os emissores usam um protocolo simplista e transmitem logo que podem os pacotes das aplicações.

No entanto, os protocolos de transmissão fiável de dados normais usam ACKs e são sensíveis ao tempo de trânsito dos pacotes. Mas como a fila de espera é infinita e os canais não perdem pacotes, todos os alarmes que dispararem são necessariamente prematuros. Assim, quanto maior for a fila de espera de C2, mais os pacotes tenderão a ficar na fila de espera, mais alarmes prematuros irão disparar nos emissores e mais pacotes duplicados serão injectados na rede. Resultado, a verdadeira relação entre T_{in} e o *goodput* é a que é mostrada pela Figura 8.5 através do gráfico (b) pois todos os pacotes emitidos em duplicado consomem inutilmente uma parte da capacidade

disponível.

De qualquer forma, o gráfico (b) assume que o valor dos *timeouts* vai aumentando dinamicamente para acomodar o aumentado do tempo de trânsito extremo a extremo, pois pressupõe-se, tal como no caso do TCP, que o valor dos *timeouts* é ajustado em função do RTT estimado. Se o valor do *timeout* fosse constante, como a fila de espera vai crescendo infinitamente, cada vez disparariam mais alarmes, cada vez mais pacotes seriam retransmitidos prematuramente, e a fracção de novos pacotes úteis recebidos pelos receptores sobre o número total de pacotes que cruzariam a rede iria descer, e o *goodput* também, dado que todos os pacotes, incluindo os duplicados, partilham a capacidade do canal gargalo ou *bottleneck link*.

Isto mostra mais uma vez que entregar pacotes muito atrasados é contraproducente e que portanto mais vale suprimir os pacotes que de qualquer forma vão chegar muito para lá do razoável num determinado contexto. Um modelo de rede mais realista é aquele em que as filas de espera são limitadas, e os pacotes que não têm lugar nas filas de espera são suprimidos como mostra a Figura 8.6. Isso também assegura que o tempo de trânsito extremo a extremo passará a ser limitado pelo somatório do tempo de transmissão das filas de espera máximas de cada um dos comutadores atravessados.

Adicionalmente, a rede da Figura 8.6 tem uma configuração mais complexa, pois no comutador C3, ligado directamente aos clientes, convergem agora canais vindos de outros comutadores que injectam outros pacotes em competição com os que os servidores enviam. Isso quer dizer que alguns dos pacotes que consumiram a capacidade do canal que liga C2 a C3 vão ser suprimidos devido a essa competição, e que uma cada vez maior fracção da capacidade desse canal não contribuirá para o débito útil final.

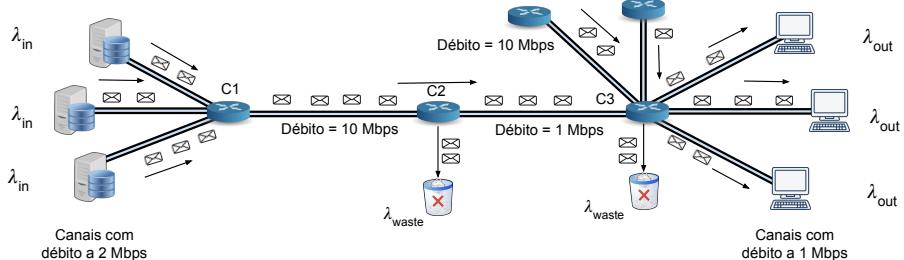


Figura 8.6: Modelo de rede com filas de espera finitas e supressão de pacotes

Como mostra a Figura 8.7, ao aumentar o tráfego injectado na rede (T_{in}), a fracção desse tráfego que atinge o destino e contribui para o débito útil da rede vai decaindo devido ao efeito combinado da existência de canais que estão saturados e limitam a capacidade da rede, da cada vez maior retransmissão prematura de pacotes atrasados e da supressão de pacotes que já consumiram capacidade na rede.

Assim, torna-se claro que se o conjunto do tráfego injectado na rede for tal que a rede funcione abaixo da sua zona de *stress*, o seu rendimento será máximo, quer pelo número reduzido de pacotes que se perdem, quer pelo facto de não haver pacotes que consomem recursos e aumentam o tempo de trânsito mas não chegam ao destino. A Figura 8.8 põe em evidência que se a rede funcionar abaixo da zona sombreada, a sua zona de *stress* ou de saturação, os pacotes chegam ao destino e o tempo de trânsito de extremo a extremo é próximo do seu mínimo, sem dilatações artificiais devidas a longas filas de espera.

O rendimento da rede tem sido “medido” informalmente nos parágrafos anteriores em termos da quantidade de pacotes entregues no destino, desprezando a faceta tempo de trânsito. Para captar as duas facetas foi introduzida a noção de **poder da rede**

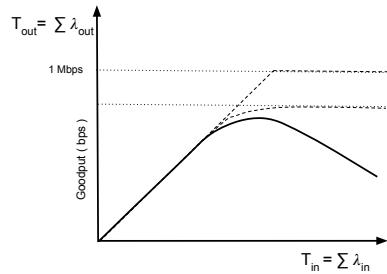


Figura 8.7: *Goodput* com filas de espera finitas, retransmissão prematura de pacotes atrasados e supressão de pacotes que consumiram recursos no *bottleneck link* antes de serem suprimidos

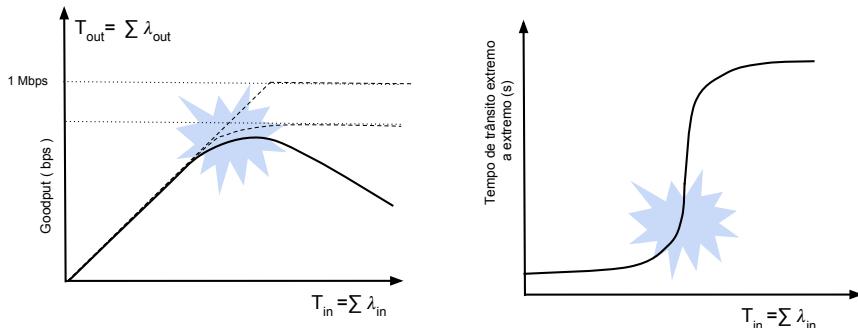


Figura 8.8: *Goodput* e tempo de trânsito numa rede saturada

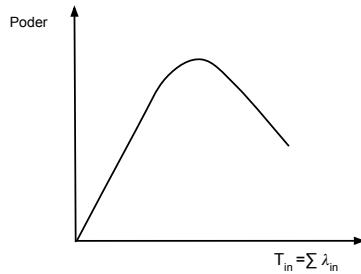
(*network power*) [Kleinrock, 1979] que é definido pela equação:

$$\text{Power} = \frac{\text{goodput}}{\text{delay}} \quad (8.1)$$

O gráfico da Figura 8.9 representa o poder de uma rede como o modelo usado na discussão. À medida que o tráfego injectado na rede e o débito útil sobem, o poder da rede vai crescendo pois o tempo de trânsito extremo a extremo é sensivelmente constante. O poder atinge o seu ponto máximo quando a rede começar a ficar saturada e o *goodput* começar a decrescer. Ultrapassado esse ponto, o *goodput* decresce e o tempo de trânsito extremo a extremo aumenta, provocando a queda significativa do poder da rede, visto que a mesma está agora completamente saturada.

Por essa razão, é importante encontrar formas de conseguir que o tráfego injectado na rede nunca ultrapasse a zona em que o poder atinge o seu máximo. Na verdade é preferível até trabalhar sempre numa zona com alguma folga pois, na prática, os fluxos de tráfego são irregulares e contém picos de débito repentinos que podem momentaneamente induzir saturação. Para se conseguir atingir este objectivo são usadas arquitecturas e protocolos que implementam algoritmos de controlo da saturação.

Uma rede entra em saturação quando o conjunto das solicitações de encaminhamento de pacotes que recebe conduzem a uma situação de colapso, e a quantidade real

Figura 8.9: Poder da rede em função de T_{in}

de pacotes encaminhados é inferior à capacidade efectiva da rede.

Define-se **débito útil (goodput)** como sendo o débito útil de informação entregue pela rede aos destinatários, isto é, descontando todo o desperdício introduzido pelos protocolos, quer em termos de cabeçalhos dos pacotes, quer em termos de pacotes reemitidos e recebidos em duplicado.

Define-se **poder da rede (network power)** como sendo o quociente do débito útil pelo tempo de trânsito de extremo a extremo. Numa rede saturada, o aumento do débito de injecção de pacotes na rede conduz à diminuição do seu poder.

O conjunto de arquitecturas, protocolos e algoritmos usados para controlar as solicitações à rede, de forma a evitar que esta entre em saturação, chamam-se **arquitecturas, protocolos e algoritmos de controlo da saturação (congestion control architectures, protocols and algorithms)**.

8.2 Como controlar a saturação

Um computador ligado à rede injecta tráfego dirigido a diferentes destinos que, naturalmente, atravessa zonas distintas da rede. Por isso o controlo do débito não pode ser feito por computador, mas pode ser feito por fluxo de pacotes, por exemplo por fluxo de transporte, *i.e.*, um conjunto de pacotes de dados com origem e destino nos mesmos sockets (*e.g.*, por conexão TCP, por sequência de datagramas UDP com a mesma origem e dirigidos ao mesmo destino, *etc.*).

No entanto, procurar tirar o melhor rendimento possível da rede para maximizar o seu poder não é fácil, pois existem diversos factores que tornam o problema complicado, nomeadamente os seguintes:

1. Os fluxos que atravessam a rede são dinâmicos e, no caso geral, alteram-se continuamente. A duração, os requisitos e a origem e destino dos fluxos existentes numa rede geral exibem uma grande variedade e uma grande escala. O controlo da saturação **tem de ser dinâmico** e adaptar-se continuamente ao conjunto dos fluxos que atravessam a rede em cada momento. A **convergência** dos algoritmos e protocolos de controlo da saturação mede o tempo que é necessário para adaptar a gestão dos fluxos na sequência de uma alteração das solicitações

à rede. Se a convergência é lenta, a gestão dos fluxos leva demasiado tempo a adaptar-se às mudanças, o que é sub-óptimo. Se for demasiado rápida, a situação pode oscilar e mesmo assim aparecer saturação.

2. Na grande maioria das situações, os computadores são autónomos e independentes da rede, pois pertencem a entidades distintas e com objectivos parcialmente contraditórios (*e.g.*, clientes e fornecedores). Por esta razão **não é possível usar arquitecturas de controlo da saturação baseadas em integração e coordenação** elevadas entre a gestão da rede e os algoritmos e protocolos que se executam nos computadores. A excepção a esta regra são as redes internas aos centros de dados em que a gestão da rede e dos servidores do centro pode ser feita de forma integrada.
3. Os diferentes fluxos que atravessam a rede têm origens e destinos distintos e seguem por diversos caminhos, por isso encontram **diferentes canais gargalos**. Encontrar o débito adequado a cada fluxo depende também dos fluxos que com ele se cruzam e competem.
4. Numa rede sem diferenciação da qualidade de serviço fornecida aos diferentes clientes, os fluxos têm de obter uma fracção equitativa da capacidade da rede. No entanto, essa equidade não é simples de estabelecer e não basta dividir igualmente a capacidade da rede pelos diferentes fluxos. Na verdade, **a noção de divisão equitativa só é fácil de aplicar localmente**, *i.e.*, ao conjunto dos fluxos que partilham o mesmo canal gargalo (*bottleneck link*).

Os dois primeiros pontos acima introduzem as dificuldades ligadas ao dinamismo do problema, e à dificuldade de integrar a gestão da rede e a gestão dos fluxos gerados pelos computadores na maioria dos casos. Os dois últimos pontos estão relacionados com o problema da afectação da capacidade disponível aos diferentes fluxos e com a necessidade de realizá-lo de forma equitativa.

Para pôr em evidência os contornos das duas últimas questões (ignorando os problemas do dinamismo e de encontrar uma implementação viável), vamos usar um algoritmo centralizado para afectar estaticamente o débito máximo a cada um dos fluxos presentes na rede.

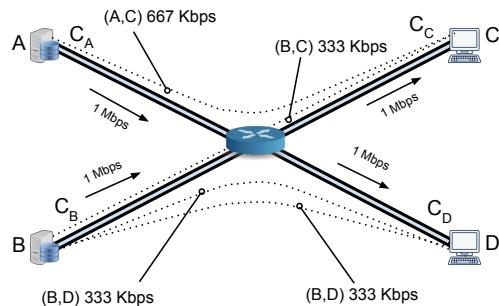


Figura 8.10: Determinação do débito máximo dos fluxos (A,C), (B,C) e 2×(B,D) numa rede em estrela

O algoritmo requer o conhecimento prévio da configuração da rede, do conjunto dos fluxos (cada fluxo é caracterizado pelo débito solicitado e pelo par de nós origem e destino) e do caminho que cada um deles segue na rede. O resultado do algoritmo é o débito máximo de cada um desses fluxos que maximiza o *goodput* e minimiza a perda de pacotes e o tempo de trânsito extremo a extremo (*i.e.*, maximiza o poder da rede).

O algoritmo usa uma pequena quantidade de tráfego designado *quantum*, denotado λ_δ , e consiste em ir incrementando de forma rotativa cada fluxo com esse *quantum*⁴. Logo que um fluxo esgota a capacidade do respectivo canal gargalo, deixa de ser incrementado. O algoritmo termina quando não for possível incrementar o débito de mais nenhum fluxo e o resultado final é o débito que foi afectado a cada fluxo.

Por exemplo, dada a rede da Figura 8.10, constituída por 4 computadores A, B, C e D interligados por um comutador através de canais com o débito de 1000 Kbps e um tempo de propagação desprezável, pretende-se encaminhar os quatro fluxos (A,C), (B,C) e 2×(B,D) afectando a cada um o máximo que for possível. Aplicando o algoritmo usando $\lambda_\delta = 1$ Kbps, chega-se à seguinte afectação: 667 Kbps a (A,C) e 333 Kbps a todos os outros, pois os restantes 3 fluxos têm o canal C_B como canal gargalo. Repare-se também que os canais C_A e C_D não são saturados e que o canal C_C é partilhado em 2/3 pelo fluxo (A,C) e 1/3 pelo fluxo (B,C).

Cada fluxo só obtém a capacidade correspondente à divisão, de forma aproximadamente equitativa, da capacidade do seu canal gargalo como acontece no caso dos três fluxos com origem em B. No entanto, no caso do fluxo com origem em A, o seu canal gargalo é o canal C_C , partilhado com outro fluxo que, como tem um canal gargalo diferente, deixa cerca de 2/3 da capacidade livre para o seu competidor.

Este tipo de afectação equitativa chama-se *max-min fairness* [Boudec, 2014] e é caracterizada por não ser possível incrementar a capacidade afectada a nenhum fluxo, sem decrementar a capacidade afectada a outros fluxos que já estão a receber uma fração igual ou inferior à que esse já recebe.

Definir uma arquitectura e um conjunto de protocolos que usasse um algoritmo centralizado para determinar a capacidade máxima de cada fluxo é bastante difícil no caso geral, pois isso exige conhecer todos os fluxos existentes na rede, qual o caminho que estes seguem, qual o estado exacto da rede, correr o algoritmo em tempo útil e ainda conseguir comunicar aos computadores as afectações determinadas para cada um dos seus fluxos. Tudo isto realizado dinamicamente e a uma escala de milhares ou mesmo milhões de fluxos.⁵

Tirar o melhor rendimento possível da rede, sem a saturar e assegurando equidade entre os diferentes fluxos de pacotes é um problema de optimização.

Calcular a solução óptima implica dispor de uma visão global do estado da rede, dinamicamente actualizada em função da evolução das solicitações e, adicionalmente, é necessário conseguir controlar os computadores ligados à rede para controlar o ritmo de emissão de pacotes por cada um dos seus fluxos.

Este tipo de arquitectura não é realista senão numa situação em que os gestores da rede tivessem controlo completo sobre os computadores que lhe estão ligados. No caso geral, o máximo que a rede pode esperar é alguma colaboração dos protocolos de transporte que esses computadores usam.

Para introduzir controlo da saturação, existem diversas alternativas de solução que se traduzem em diferentes algoritmos e protocolos assim como em diferentes arquitecturas de coordenação da rede e dos computadores. A figura 8.11 ilustra três grupos de alternativas diferentes.

⁴ Como em muitos algoritmos deste tipo usa-se o termo *quantum* para designar uma pequena quantidade cujo valor determina o erro cometido nos cálculos. O valor usado na prática é o maior valor que conduz a um erro menosprezável num dado contexto.

⁵ Acresce, no caso mais geral, que a Internet é um conjunto de redes interligadas com gestão autónoma e esta afectação só poderia ser realizada por um mecanismo que coordenasse o conjunto de todas as redes.

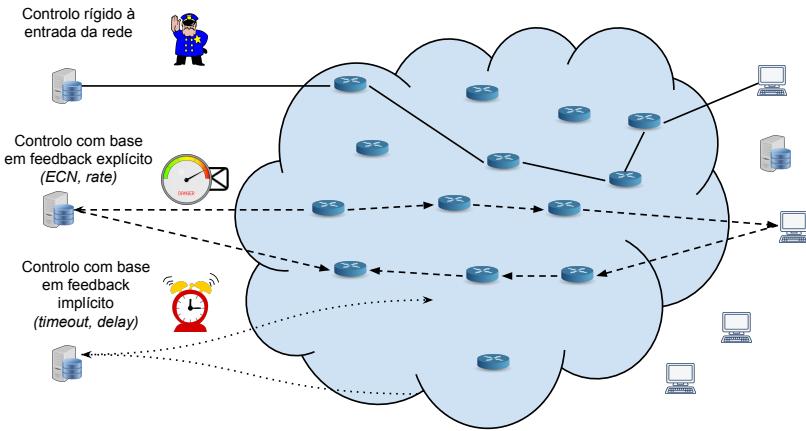


Figura 8.11: Alternativas para lidar com a saturação da rede

Começando de cima para baixo, o primeiro grupo de soluções baseia-se numa integração muito grande entre os computadores e a rede. Por exemplo, numa rede baseada na noção de circuito, um fluxo e um circuito podem coincidir e um computador só pode dar início a um fluxo depois de a rede ter aberto o respectivo circuito e lhe ter afectado a capacidade solicitada. Se o circuito não “couber” na rede, a rede recusa-o. Mesmo sem usar necessariamente a noção de circuito, este tipo de arquitectura de controlo da saturação inclui sempre mecanismos de reserva de recursos e de limitação do débito máximo que os computadores podem injectar na rede. Se esses limites forem conservadores anulam a possibilidade de haver saturação à custa de uma arquitectura rígida e que subaproveita a capacidade da rede, pois desperdiça a capacidade que não estiver a ser usada no momento. Adicionalmente, esta solução é pouco escalável e adapta-se com dificuldade a um grande dinamismo.

Os outros dois grupos de soluções de controlo da saturação não impõem limites *a priori*, nem ao número, nem ao débito dos fluxos. Adoptam um ponto de vista optimista, baseado na noção de “melhor esforço”, que não anula *a priori* a saturação, pois esta pode sempre ter lugar. Em contrapartida, usam mecanismos de *feedback* para indicarem ao nível de transporte dos computadores que deve reduzir o débito dos fluxos, sob pena de introduzir saturação. Baseiam-se portanto na colaboração e na disponibilidade voluntária do nível transporte para reduzir o débito de emissão de pacotes.

No primeiro grupo deste tipo de soluções esse *feedback* é explícito e pode assumir diversas formas: a rede sinaliza um fluxo de que o mesmo está em risco de provocar saturação e que portanto é preferível reduzir o seu débito (solução ECN – *Explicit Congestion Notification*) ou então é comunicado o débito explícito a adoptar pelo fluxo (*Explicit Rate Notification*).

No segundo grupo utiliza-se *feedback* implícito, o qual se traduz em o nível de transporte dos computadores medir o comportamento dos seus fluxos para detectar a presença de saturação (perda de pacotes e variação do débito e do tempo de trânsito extremo a extremo). Neste caso a rede adopta uma posição do tipo *laissez faire laissez passer* e espera que os seus utilizadores adoptem uma melhor postura e, ao aperceberem-se de que esta está saturada, reduzam voluntariamente o débito dos seus

fluxos. Em qualquer das situações, se as filas de espera estiverem demasiado cheias, são suprimidos pacotes.

O controlo da saturação necessita de regulação do débito dos fluxos de pacotes emitidos pelos computadores ligados à rede. Este controlo pode basear-se em imposição pela rede do débito máximo ou em *feedback* da rede aos computadores para que estes se auto-regulem.

As arquitecturas baseadas em *feedback* da rede dividem-se ainda em *feedback* explícito, quando a rede dá indicações directas aos computadores para que estes se auto regulem, e *feedback* implícito, quando são os computadores que detectam os indícios de saturação.

O protocolo TCP incorpora algoritmos de controlo da saturação baseados em *feedback* implícito e, opcionalmente, também explicito (a solução ECN). Assim vamos estudar implementações possíveis destas duas alternativas através do estudo do controlo da saturação no protocolo TCP.

8.3 Controlo da saturação no protocolo TCP

Os protocolos de janela deslizante proporcionam uma forma simples de controlo do ritmo de emissão: a dimensão da janela de emissão. De facto, uma conexão com janela de emissão de dimensão W transmite dados com o débito médio de W/RTT , ver a Secção 6.3.

O protocolo TCP usa a dimensão da janela de emissão para concretizar quer um mecanismo de controlo de fluxo, que adapta o ritmo de transmissão à capacidade do receptor consumir atempadamente os dados recebidos, quer um mecanismo de controlo da saturação, que adapta o ritmo de transmissão à capacidade disponível na rede para a conexão TCP.

Repare-se que a dimensão da janela de emissão do TCP limita o número máximo de segmentos que podem ser emitidos em sequência mas, uma vez todos os segmentos da janela transmitidos, os seguintes são transmitidos exactamente ao ritmo da chegada dos ACKs. Ora os ACKs vêm ao ritmo com que o receptor vai recebendo de facto segmentos, o qual é condicionado pela capacidade do canal com menos capacidade, o canal gargalo ou *bottleneck link*.

Por exemplo, ver a Figura 8.12, considere-se um emissor e um receptor ligados ao mesmo comutador. O canal do emissor tem um débito de 1 Gbps, e o do receptor tem um débito de 100 Kbps. Se a janela de emissão for igual a 4 segmentos de 1250 bytes cada (≈ 10.000 bits em cada segmento), o tempo de transmissão de cada segmento a 1 Gbps é negligenciável ($10^4/10^9 = 10 \mu s$). No entanto, o tempo de transmissão de cada segmento a 100 Kbps é 100 ms. Inicialmente, o emissor pode enviar os 4 segmentos em sequência em tempo negligenciável, mas os segmentos vão ser transmitidos pelo comutador através do canal do receptor, e este vai receber cada um deles espaçado do tempo de transmissão nesse canal (100 ms). Portanto, os ACKs são recebidos pelo emissor separados de pelo menos 100 ms e, após a sequência inicial, o emissor vai transmitir os outros segmentos separados de 100 ms cada, o ritmo com que recebe os ACKs.

O TCP é **auto regulado** (*self-clocking*) pois o ritmo de chegada de ACKs depende do ritmo com que os segmentos TCP são entregues ao receptor. Com efeito, sempre que um ACK é recebido, isso quer dizer que um segmento deixou a rede e outro pode ser transmitido.

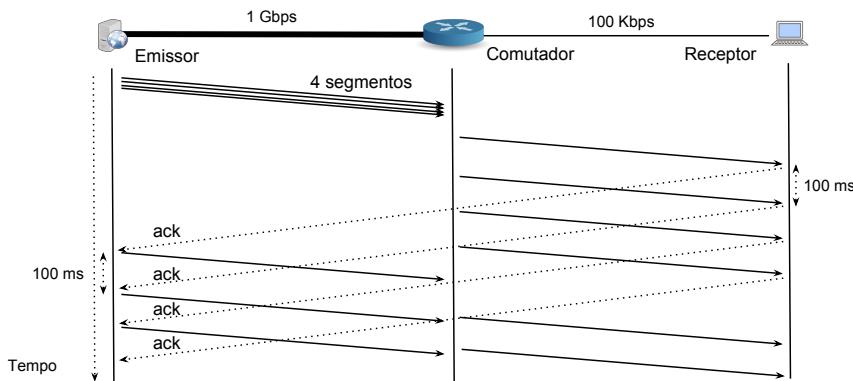


Figura 8.12: O débito do canal gargalo impõe o ritmo de chegada de ACKs e também o ritmo de envio de segmentos pelo emissor

Teoricamente, se apenas for limitada pelo débito disponível na rede, *i.e.*, não existem limitações aplicacionais nem de controlo de fluxos, uma conexão TCP poderia ajustar a dimensão da sua janela ao estritamente necessário para manter o débito permitido pelo canal gargalo e manter-se numa situação estável e ótima. Esta situação particular é muito difícil de obter porque a capacidade disponível no canal gargalo varia dinamicamente devido ao restante tráfego em competição e o TCP tem dificuldade em estimá-la. Uma forma de conseguir fazer esta estimativa consiste em ir aumentando progressivamente a janela, diminuindo-a apenas quando se apercebe que se passou o limite adequado.

De que sinais dispõe o TCP para se aperceber de que o fluxo de pacotes enviados tem um ritmo que não é suportado pelo canal gargalo? Existem dois tipos de sinal: a eventual perda de pacotes, sinalizada pelo disparo de alarmes (*timeouts*), e o tipo de variação experimentado pelo débito extremo a extremo e pelo tempo de trânsito dos pacotes.

Com efeito, se não existirem mecanismos de sinalização explícita pela rede da existência de saturação, o receptor não se apercebe da mesma pois desconhece o ritmo de emissão do emissor. Só o emissor, ao não receber atempadamente os ACKs, ou através da análise da variação RTT, poderá tentar inferir qual o estado da rede na zona que os seus pacotes atravessam, e em particular, qual o estado do canal gargalo que estes atravessam. O receptor também se apercebe de anomalias quando recebe pacotes fora de ordem, mas esse evento não está necessariamente ligado a saturação da rede.

O sinal transmitido pelo disparo de *timeouts* é não só o mais fiável, como também o mais simples de interpretar. Nas redes modernas, a perda de pacotes devido a erros nos canais é desprezável, excepto nos canais sem fios. Por outro lado, a forma como o TCP calcula a duração dos alarmes (ver a Secção 7.2), faz com que a retransmissão prematura de segmentos seja menos provável.

Como um emissor TCP não tem noção de qual a capacidade que tem disponível na rede terá de fazer experiências. Por exemplo, poderá começar com uma janela de dimensão igual à do MSS (*Maximum Segment Size*) e depois ir subindo ou descendo a sua dimensão. Se a variar lentamente, a convergência pode ser lenta e o emissor pode não aproveitar a capacidade disponível ou, em caso de saturação, pode levar demasiado tempo a corrigir o débito e a saturação persistir para além do desejável. Se a variação for brusca, o emissor poderá oscilar entre momentos de sub-aproveitamento

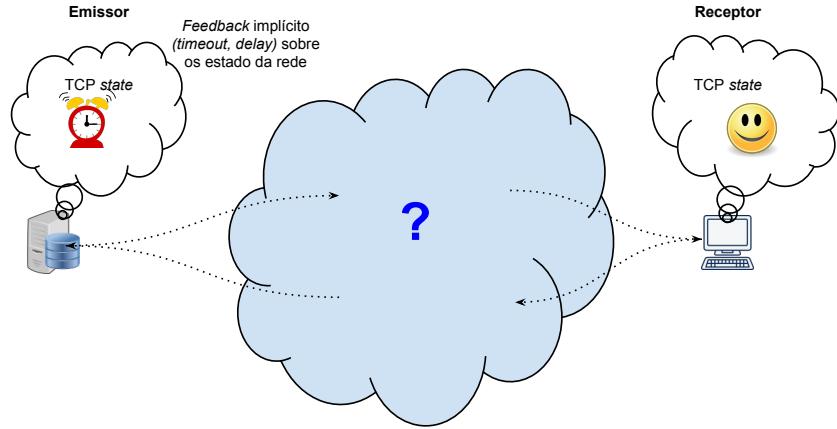


Figura 8.13: Controlo da saturação com base em *feedback* implícito

e momentos de saturação, sem encontrar um ponto de equilíbrio.

A variação é rápida se a janela de emissão mudar de dimensão de forma multiplicativa (*e.g.*, sobe para o dobro ou diminui para metade em cada RTT) e será relativamente lenta (linear) se a mesma dimensão variar de forma aditiva (*e.g.*, sobe ou desce de 1 MSS em cada RTT). Resta saber, das quatro possíveis combinações de ritmos de variação (subida e descida aditivas, subida e descida multiplicativas, subida aditiva e descida multiplicativa, e vice-versa) qual será a mais adequada.

Como um emissor TCP não tem noção de qual o estado da rede, terá de experimentar ir aumentando o ritmo de emissão e, quando se aperceber que há problemas, deverá diminui-lo. O TCP varia o seu ritmo de emissão máximo usando a fórmula **subida aditiva, descida multiplicativa (AIMD – Additive Increase, Multiplicative Decrease)**.

A técnica de ir incrementando o ritmo de transmissão para tentar aproximar a capacidade máxima da rede sem a saturar, chama-se **sondar os recursos da rede (network resource probing)**.

A razão de ser desta opção, ver a Figura 8.14, pode ser justificada de forma intuitiva: a prudência manda não se ser demasiado abrupto a exigir recursos da rede, e quando se detectam anomalias é preferível recuar rapidamente para uma situação em que a saturação e a perda de pacotes sejam rapidamente ultrapassadas, tanto mais que nessa situação vários segmentos estarão ainda em trânsito. Com efeito, se o RTT de uma conexão for elevado e a saturação ocorrer perto do destino, existe uma elevada probabilidade de estarem muitos pacotes em trânsito. Veremos a seguir que é possível justificar mais rigorosamente que a estratégia AIMD é de facto a mais adequada, ver a Secção 8.5.

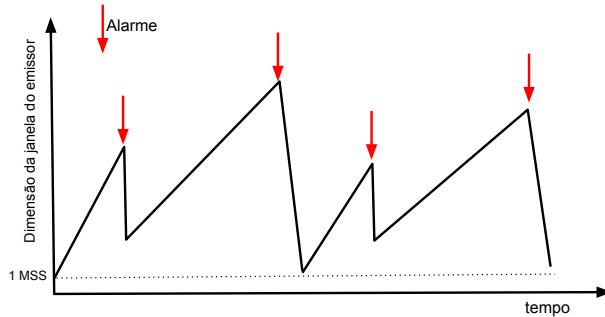


Figura 8.14: *Additive increase multiplicative decrease*

O protocolo TCP usa a dimensão da janela de emissão para concretizar quer um mecanismo de **controlo de fluxo** (*flow control*), que adapta o ritmo de transmissão à capacidade do receptor consumir atempadamente os dados recebidos, quer um mecanismo de **controlo da saturação** (*congestion control*), que adapta o ritmo de transmissão à capacidade disponível na rede para a conexão TCP.

O TCP ajusta dinamicamente a dimensão da janela de emissão de tal forma que essa dimensão nunca pode ser superior a dois limites. O primeiro limite designa-se *receiver advertised window* (*recAdvertisedWnd*), e corresponde ao limite imposto pelo receptor através do mecanismo de controlo de fluxo. O segundo designa-se *congestion window* (*congWnd*), e corresponde ao limite imposto pelo algoritmo de controlo da saturação.

O algoritmo de controlo da saturação do TCP tem início com $\text{congWnd} = \text{mss}$, incrementa o ritmo de transmissão subindo *congWnd*, usa o disparo do *timeout* como sinal da presença de saturação, da qual tenta fugir tão rapidamente quanto possível, e portanto reduz de novo o valor de *congWnd* ao valor do MSS na sequência de um *timeout*. Ou seja, a descida multiplicativa é a mais abrupta que é possível.

A subida aditiva é materializada através do incremento do valor de *congWnd* de MSS bytes em cada RTT. Para esse efeito, o TCP pode usar a seguinte estratégia: no início da conexão *congWnd* recebe o valor do MSS, após receber o primeiro ACK, i.e., após um RTT, *congWnd* sobe para $2 \times \text{MSS}$, após mais 2 ACKs, outro RTT, sobe para $3 \times \text{MSS}$, após mais 3 ACKs, outro RTT, sobe para $4 \times \text{MSS}$, e assim sucessivamente. Pode-se incrementar o valor de *congWnd* de MSS em cada RTT ou incrementar ligeiramente o valor com cada ACK recebido. Com efeito, se o TCP estiver a usar uma janela limitada apenas pelo valor de *congWnd* e segmentos com a dimensão MSS, e se todos os segmentos forem *acked*, em cada RTT os ACKs recebidos serão tantos quanto o número de segmentos transmitidos, ou seja $\text{congWnd} / \text{MSS}$. A subida aditiva pode então ser implementada incrementando *congWnd* de $\text{MSS} \times \text{MSS} / \text{congWnd}$ cada vez que o TCP recebe um ACK, ver a Listagem 8.1, pois continua a admitir-se que em cada RTT se transferem *congWnd* bytes.

A estratégia de começar com uma janela igual a MSS foi chamada *slow start*, porque a janela do TCP começa sempre com um valor baixo, por oposição à solução usada

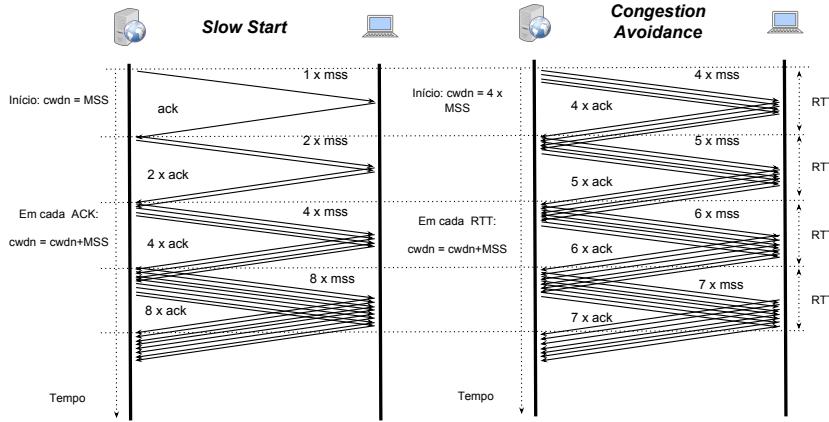
Listing 8.1: Pseudo código simplificado da actualização de `congWnd` com AIMD

```

start :      congWnd = MSS
alarm :      congWnd = MSS
ACK:        congWnd += MSS * MSS / congWnd

```

antes da introdução de controlo de saturação⁶. Por exemplo, supondo que se está numa situação em que o RTT é de 100 ms, o MSS de 1250 bytes (\approx 10.000 bits) e a capacidade disponível para a conexão de 1 Mbps, o débito inicial é $W/RTT = 10.000/0,1 = 100$ Kbps no 1º RTT, 200 Kbps no 2º RTT, 300 Kbps no 3º RTT, etc. e só no 10º RTT, ou seja ao fim de 1 segundo, é que se atinge o débito máximo disponível. Se o débito disponível for 10 Mbps, só ao fim de quase 10 segundos é que se atinge esse débito. A alternativa de usar um valor mais alto para a janela inicial é complicada pois o emissor não sabe com que situação real da rede está de facto a lidar.

Figura 8.15: Subida da janela de emissão durante a fase *slow start* e durante a fase *congestion avoidance* do TCP

Devido a este problema, o TCP durante a fase *slow start* usa uma estratégia de subida multiplicativa durante a primeira fase da subida, passando mais tarde a subida aditiva. A fase de subida multiplicativa continua a chamar-se *slow start* e a de subida aditiva chama-se *congestion avoidance* ou *additive increase*. A subida multiplicativa traduz-se em incrementar a dimensão da janela de emissão de 1 MSS cada vez que se recebe um ACK. Esta estratégia conduz à duplicação da dimensão da janela de emissão em cada RTT como está ilustrado na parte esquerda da Figura 8.15.

A subida durante a fase *congestion avoidance* traduz-se, como já foi referido, em aumentar `congWnd` de $MSS \times MSS / congWnd$ por cada ACK recebido e está ilustrada na parte direita da figura 8.15. O valor de `congWnd` que provoca a transição da fase *slow start* para a fase *congestion avoidance* chama-se *slow start threshold*, e está contido numa variável chamada `sst` daqui para a frente.

⁶ A versão inicial de TCP, sem controlo de saturação, só limitava a janela de emissão pelo mecanismo de controlo de fluxo. Por isso usava o valor do campo `recAdvertisedWnd` como dimensão máxima da janela o que, na altura, conduzia rapidamente à saturação da rede.

O algoritmo concreto utilizado no protocolo TCP para realizar o controlo de saturação passou por várias fases. Inicialmente o TCP não tinha controlo da saturação mas, quando esta apareceu, Van Jacobson [Jacobson, 1988] liderou um notável trabalho de engenharia e experimentação que conduziu a duas versões iniciais do algoritmo, designadas respectivamente TCP Tahoe e TCP Reno.

Desde essa altura a Internet evoluiu imenso em termos de escala e capacidade, e durante esse período de quase 30 anos os algoritmos de controlo da saturação evoluíram muito e tiveram várias versões, mas muitas dessas novas versões correspondem a refinamentos e afinações cada vez mais sofisticadas destas duas versões iniciais. Por isso vamos dedicar-lhes alguma atenção a seguir.

A discussão que se segue assume que o débito do emissor é limitado apenas pelo algoritmo de controlo da saturação. A realidade é mais subtil pois em cada momento o TCP é limitado por vários factores: a aplicação ter dados para enviar ao ritmo adequado, o canal gargalo não ser o canal que liga o computador à rede e o receptor ter espaço nos *buffers* de recepção. De facto, a verdadeira janela de emissão é assim calculada:

Listing 8.2: Cálculo da dimensão da janela de emissão do TCP

```
maxWnd = min(congWnd, recAdvertisedWnd)
effectiveWnd = maxWnd - (lastByteSent - lastByteAcked)
if (effectiveWnd > 0)
    // it is possible to send more data ...
```

e em cada momento só é possível transmitir dados se `effectiveWnd > 0`. Ou seja, quando `maxWnd = MSS` na sequência de um *timeout*, não podem ser transmitidos novos dados enquanto não forem retransmitidos e/ou *acked* os segmentos em trânsito.

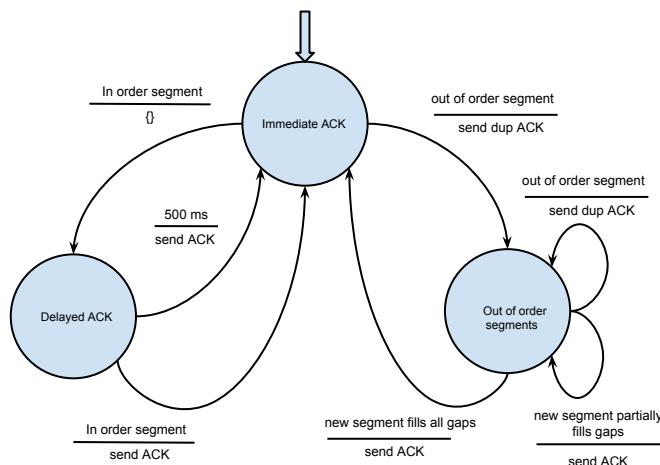


Figura 8.16: Funcionamento do receptor TCP

Na Listagem 8.1, apresentada acima, e nas listagens 8.3 e 8.4, apresentadas a seguir, aquando da actualização do valor de `congWnd` na sequência de um ACK, usa-se sempre o valor de MSS para actualizar aquela variável. Na realidade o valor usado é o número de bytes que o ACK cobre pois nem sempre o TCP envia segmentos da dimensão de MSS e, por outro lado, nem todos os segmentos recebidos desencadeiam o envio de um ACK pelo receptor. Para que o leitor os possa ter presentes, a Figura

8.16 recorda os aspectos essenciais do funcionamento do receptor TCP através da sua máquina de estados.

A seguir apresenta-se uma visão resumida das duas versões iniciais do algoritmo de controlo da saturação do TCP.

Controlo da saturação – TCP Tahoe

O primeiro algoritmo de controlo da saturação usado pelo protocolo TCP foi designado *TCP Tahoe congestion control* pois foi introduzido em 1988 com o sistema de operação Unix 4.2 BSD ou *release Tahoe*.

Inicialmente o TCP não conhece o valor de **sst** (*slow-start threshold*, o qual controla o ponto a partir do qual se passa da fase de incremento multiplicativo da janela para a fase de incremento aditivo) e usa um valor de **sst** alto (e.g., 64 KBytes) pelo que geralmente a janela sobe de forma multiplicativa até ao primeiro *timeout*. É como se o TCP começasse por tentar saber à partida qual o débito máximo possível ou, posto de forma irónica, para evitar a saturação é necessário começar por provocá-la. Na sequência do primeiro *timeout*, o valor de **sst** passa a 1/2 do valor de **congWnd** no momento do *timeout*, **congWnd** toma o valor inicial, i.e., **MSS**, e entra-se na fase *slow start* até **congWnd** atingir o valor de **sst**, altura em que se transita para a fase *congestion avoidance* (esta fase também se poderia ter chamado *network probing*) até disparar um novo *timeout* ou ter lugar o evento *triple duplicateACK* e o processo repete-se como está ilustrado na Figura 8.17. A Listagem 8.3 apresenta de forma mais completa a gestão dos valores de **congWnd** e de **sst**.

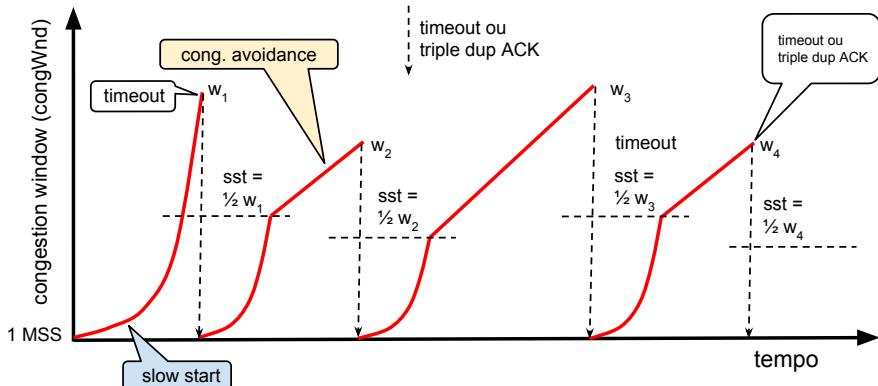


Figura 8.17: Evolução do valor do limite **congWnd** ao longo de uma conexão TCP na versão TCP Tahoe

Controlo de saturação – TCP Reno

Como vimos na secção 7.2, o TCP usa um método de cálculo do *timeout* que pretende prevenir, tanto quanto possível, as retransmissões prematuras, e é conservador na estimação do RTT e dos alarmes. Em particular, o valor do alarme duplica sempre que se retransmite um segmento. Isso quer dizer que em caso de perda de pacotes, muitos podem ter sido enviados antes da detecção da perda através do alarme. Por outro lado, para evitar tanto quanto possível retransmissões inúteis, o emissor usa também o mecanismo *FAST retransmission* quando recebe 3 ACKs duplicados (ou seja, depois de receber 4 ACKs seguidos com o mesmo número de sequência), o que é outra forma de detectar a perda de um segmento. Só que neste caso a perda foi detectada mais cedo e tudo indica que se trata de uma perda isolada, e não da perda

Listing 8.3: Pseudo código simplificado da actualização de `congWnd` na versão TCP Tahoe

```

On event < START >:
    congWnd = MSS
    sst = 64 KBytes
    dupAckCount = 0

On event < TIMEOUT or ( dup_ACK and dupAckCount == 3 ) >:
    sst = max (congWnd / 2, 2*MSS)
    congWnd = MSS
    dupAckCount = 0

On event < DUP_ACK and dupAckCount < 3 >:
    dupAckCount++

On event < NEW_ACK and status == slow start >:
    congWnd += MSS
    dupAckCount = 0

On event < NEW_ACK and status == congestion avoidance >:
    congWnd += MSS * MSS / congWnd
    dupAckCount = 0

```

de uma grande quantidade de segmentos, pois o emissor continua a receber ACKs o que mostra que os segmentos seguintes foram entregues ao receptor.

Por isso o algoritmo de controlo de saturação foi alterado e, quando é detectado um evento *triple duplicate ACK*, ao invés de se reduzir o valor de `congWnd` a um MSS, actualiza-se o valor de `sst` como anteriormente, mas entra-se num novo estado, chamado *fast recovery*, afecta-se a `congWnd` o valor `sst + 3 * MSS` e não o valor do MSS, e continua-se a manipular aditivamente o valor da `congWnd`.

Esta nova versão do algoritmo de controlo da saturação foi introduzida com a *release* do sistema de operação Unix 4.3 BSD Reno em 1990. Este algoritmo conduz à gestão de `congWnd` como ilustrado de forma aproximada na Figura 8.18 e descrito com mais detalhe na Listagem 8.4. Quando a conexão é estável e a grande maioria dos eventos de perda de segmentos são do tipo *triple duplicate ACK*, esta gestão conduz a um comportamento em “dente de serra”, compatível com a filosofia AIMD apresentada inicialmente.

No início do estado *fast recovery*, a variável `congWnd` toma o valor `congWnd = sst+3*MSS` porque continuaram a chegar dados ao receptor, e enquanto se permanecer nesse estado é porque se continuam a receber ACKs duplicados e a janela pode aumentar. Quando o processo de recuperação terminar, `congWnd` toma de novo o valor `congWnd = sst`. A máquina de estados com acções da Figura 8.19 mostra o comportamento do emissor TCP Reno.

O protocolo TCP recebeu diversos algoritmos de controlo da saturação com o objectivo de maximizar a capacidade da rede utilizada por cada conexão, sem deslealdade para as restantes com que compete, e evitando a entrada da rede em saturação. As versões iniciais desses algoritmos chamaram-se TCP Tahoe e TCP Reno e usam a perda de pacotes como sinónimo de *feedback* implícito da rede.

Estas versões constituem uma primeira solução para um problema complexo dada a diversidade das situações em que o controlo de saturação tem de actuar: quer a conexão atravesse uma pequena área da rede, homogénea, bem dimensionada, e com um pequeno RTT; quer a conexão atravesse uma grande quantidade de canais, hete-

Listing 8.4: Pseudo código simplificado da actualização de `congWnd` na versão TCP Reno

```

On event < START >:
    congWnd = MSS
    sst = 64 KBytes
    dupAckCount = 0
    status = slow start

On event < TIMEOUT >:
    sst = max (congWnd / 2, 2*MSS)
    congWnd = MSS
    dupAckCount = 0
    status = slow start

On event < NEW_ACK and status == slow start >:
    congWnd += MSS
    dupAckCount = 0
    if ( congWnd >= sst ) status = congestion avoidance

On event < NEW_ACK and status == congestion avoidance >:
    congWnd += MSS * MSS / congWnd
    dupAckCount = 0

On event < NEW_ACK and status == fast recovery >:
    congWnd = sst
    dupAckCount = 0
    status == congestion avoidance

On event < DUP_ACK and dupAckCount == 3 >: // triple dup ack
    sst = max (congWnd / 2, 2*MSS)
    congWnd = sst+3*MSS
    dupAckCount = 0
    status == fast recovery

On event < DUP_ACK and dupAckCount < 3 and status != fast recovery >:
    dupAckCount++

On event < DUP_ACK and status == fast recovery >:
    congWnd += MSS
    dupAckCount++

```

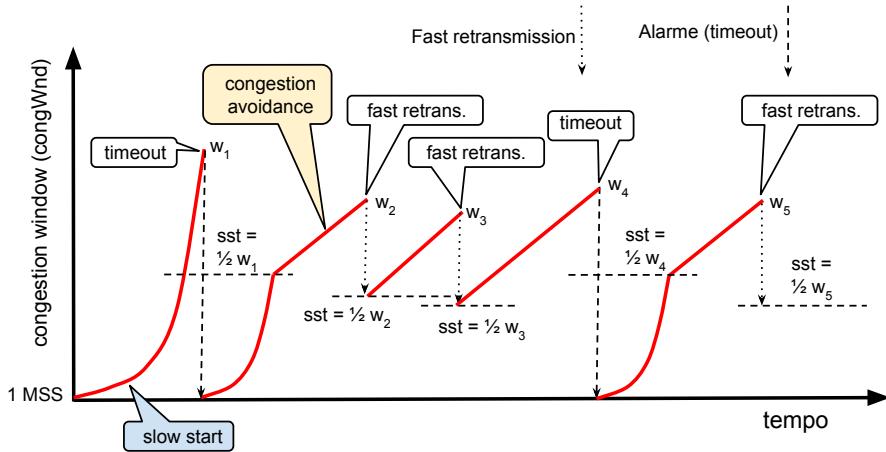


Figura 8.18: Evolução do valor do limite $cwnd$ ao longo de uma conexão TCP na versão TCP Reno (fase *fast recovery* omitida)

rogéneos, com níveis de utilização muito diversos e em presença de um elevado valor do RTT.

Acresce que, ao basearem-se exclusivamente em mecanismos de controlo com *feedback* implícito, os algoritmos de controlo da saturação requerem que a conexão TCP “tente adivinhar” o que se passa na rede a partir dos sintomas que pode inferir e não de avisos explícitos.

Os algoritmos de controlo da saturação usados com o protocolo TCP continuaram a evoluir desde a versão Reno, como a seguir veremos na secção 8.5, mas por agora vamos analisar outras soluções de suporte do controlo da saturação com base na colaboração da rede através da transmissão de notificações explícitas sobre o estado da mesma.

8.4 Controlo com *feedback* explícito

A saturação da rede é detectada pelos comutadores através da análise do estado das sua filas de espera. Quando a saturação se aproxima, os comutadores podem tentar avisar os originadores dos fluxos que os atravessam para que estes diminuam o ritmo de emissão de pacotes. Os inventores do protocolo IP imaginaram que seria possível, nessa situação, um comutador enviar uma notificação explícita para o emissor de cada pacote que fosse suprimido. Assim, quando um comutador suprimisse um pacote IP por não ter lugar para ele numa fila de espera, deveria enviar um pacote, chamado *choke packet*, para o endereço origem do pacote suprimido, e foi proposta a utilização do protocolo ICMP para esse efeito.

Se um comutador for atravessado por inúmeros fluxos de pacotes distintos, com diversas origens, em caso de congestionamento teria de enviar inúmeros *choke packets*. Se não o fizer, ou se estes sinais se perderem, a saturação persiste. Como é fácil de reconhecer, numa rede de grande escala, o envio de *choke packets* faz “parte do problema e não da solução”, na medida em que aumenta o tráfego numa rede já saturada e sobrecarrega comutadores já em apuros. Por outro lado, quando o mecanismo foi introduzido, não se sabia lá muito bem o que o receptor deveria fazer quando recebesse *choke packets*. Por isso este mecanismo nunca foi de facto implementado.

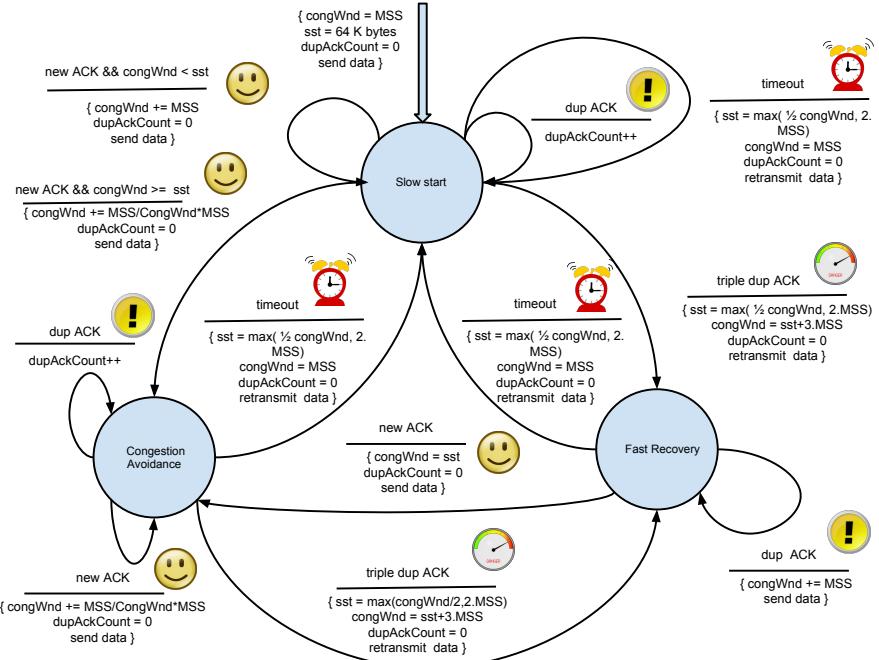


Figura 8.19: Descrição do algoritmo de controlo da saturação TCP Reno

No entanto, como a maioria dos fluxos (ou dos circuitos) estão associados a tráfego nos dois sentidos como o TCP, o *feedback* da rede pode chegar ao emissor original encavalitado no tráfego enviado no sentido contrário pelo receptor. Se possível de aplicar, esta técnica resolve o problema mesmo quando os caminhos num e noutro sentido são diferentes e não implica a introdução de pacotes extra na rede.

A Figura 8.20 ilustra a ideia. Se um comutador quiser transmitir alguma informação sobre o estado da rede ao emissor dos pacotes, anota essa informação nos cabeçalhos dos pacotes que este enviou, e o receptor, quando enviar pacotes no sentido contrário, copia a informação da rede e coloca-a no cabeçalho dos pacotes que envia ao emissor original. Por exemplo, se um comutador quiser enviar informação a um emissor TCP, anota essa informação no cabeçalho dos segmentos que encaminha, e o receptor ecoa-a para o emissor original no cabeçalho dos segmentos TCP que envia no sentido contrário. Naturalmente, esta técnica só pode ser usada com tráfego bidireccional e com a colaboração da outra extremidade da conexão.

Este mecanismo foi introduzido inicialmente em redes de circuitos e usava pacotes de controlo especiais para transportar a informação de controlo de volta ao emissor original. Quando recebiam os pacotes de controlo, os comutadores podiam analisar o seu conteúdo, eventualmente actualizá-lo e, quando o pacote de controlo chegasse à extremidade do circuito era ecoado para a extremidade contrária, levando a informação recolhida no sentido inverso para a extremidade a quem a mesma interessava.

ECN - Notificação explícita da saturação

Inspirado nesta ideia foi introduzido no protocolo TCP um mecanismo de sinalização explícita, da existência de saturação, ao emissor de um fluxo TCP. O mecanismo chama-se ECN - *Explicit Congestion Notification*, ver o RFC 3168, e utiliza duas flags do cabeçalho IP e outras duas do cabeçalho TCP. A implementação pressupõe a colaboração de diversas componentes da rede. Os comutadores usam as flags do

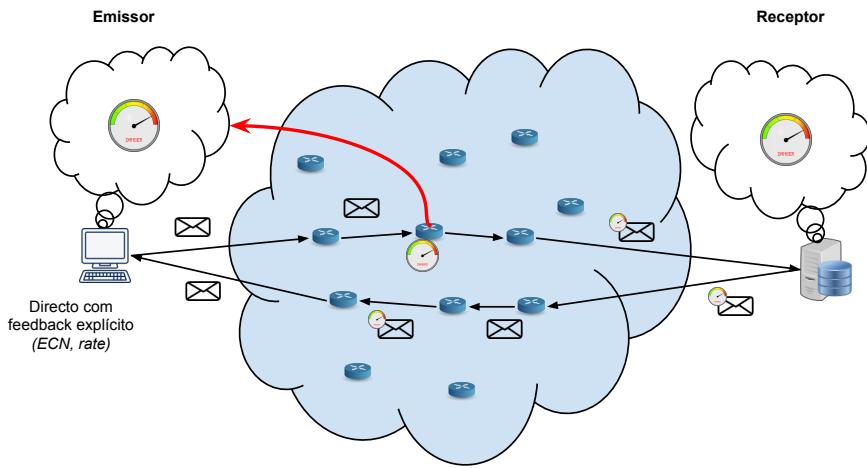


Figura 8.20: Encaminhamento de *feedback* explícito encavalitado no tráfego enviado no sentido contrário

cabeçalho IP para, no caso de a saturação se aproximar, marcarem um pacote com essa informação. Quando o pacote chega ao destino e o nível TCP envia um segmento no sentido contrário, usa *flags* do cabeçalho TCP para indicar ao emissor que um segmento que recebeu passou por um comutador em *stress*. Trata-se portanto de um mecanismo que envolve alguma colaboração explícita entre o nível rede e o nível transporte.

Assim, na abertura da conexão, ambas as extremidades indicam se suportam ou não ECN, pois não vale a pena uma perder tempo com o mecanismo se a outra não o suportar. Caso ambas o suportem, o emissor quando envia segmentos para a outra extremidade, indica no cabeçalho do pacote IP, no qual o segmento é encapsulado, que esse pacote transporta dados de uma conexão de transporte que suporta ECN.

No seu percurso para o receptor, o pacote com o segmento TCP passa por diversos comutadores. Se algum deles estiver em vias de ficar saturado e suportar a opção ECN (nem todos os comutadores a suportam), marca esse facto no cabeçalho do pacote IP. Se mais do que um comutador estiver nessa situação, basta que o primeiro marque. Quando o pacote chega ao receptor e é entregue ao nível TCP, como este suporta ECN, verifica se o pacote que contém o segmento que acabou de receber tem alguma indicação sobre saturação e, se for o caso, quando enviar tráfego no sentido contrário, por exemplo ao enviar um ACK, assinala no cabeçalho TCP desse ACK que o segmento com dados recebido continha uma notificação de saturação.

Quando este sinal chegar ao emissor original, este realiza exactamente as mesmas operações, do ponto de vista do controlo da saturação, que as que executa quando recebe um triplo ACK duplicado (mas sem retransmitir os dados pois estes foram recebidos).

A Figura 8.21 ilustra o mecanismo. Quando o emissor TCP envia dados novos⁷ marca os pacotes IP com esses segmentos com a indicação de que gostaria de ser notificado, se possível, através do mecanismo ECN (legenda (1) na figura). Caso o pacote chegue ao receptor com a indicação de que um comutador intermédio está saturado (legendas (2) e (3) na figura), o receptor envia segmentos TCP para o emissor (com os ACKs por exemplo) com a *flag* TCP ECE (ECE - ECN Echo) posicionada, a

⁷Com retransmissões ou ACKs o mecanismo não é usado.

indicar que foi avisado de que há risco de saturação no sentido contrário (legenda (4) na figura). O emissor, depois de receber a notificação ECN, nos novos segmentos que enviar ao receptor, posiciona a *flag* TCP CWR (CWR - *Congestion Window Reduced*), que funciona como uma espécie de confirmação de recepção da notificação, ou ACK, de saturação para o receptor, que a deixará de enviar nos segmentos seguintes.

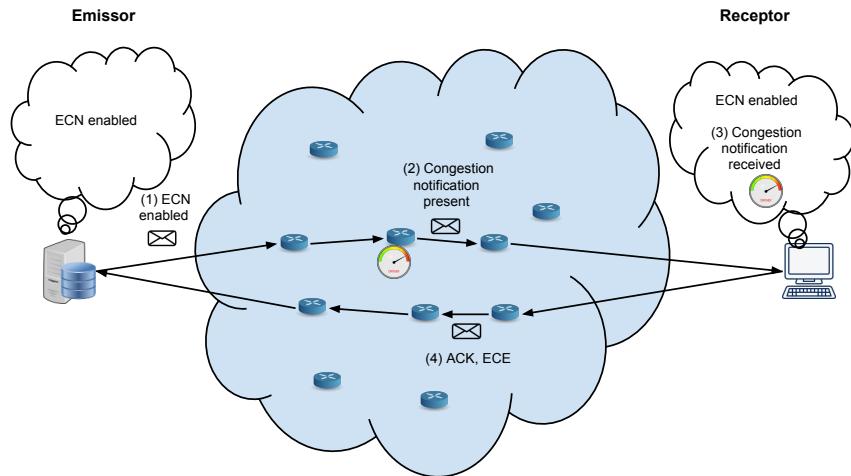


Figura 8.21: Funcionamento do mecanismo TCP ECN

O mecanismo é um exemplo de *feedback* explícito da rede aos computadores sobre a sua situação do ponto de vista da saturação. Este mecanismo é binário, mas existem outros mecanismos, não presentes nas redes IP, em que esse *feedback* é mais completo, e contém recomendações com indicação de qual o débito a usar pelo emissor.

Um mecanismo desse tipo poderia ser introduzido utilizando o campo de opções do TCP. O problema é que essa solução teria dois problemas delicados de implementação. O primeiro seria a obrigatoriedade de os comutadores conhecerem os detalhes do protocolo TCP, o que seria uma violação grosseira da *separation of concerns* implementada pela separação da rede em níveis. De facto, com essa opção, qualquer alteração de um protocolo de transporte poderia ter impacto ao nível rede. O outro problema, também delicado, é o de determinar qual o débito a oferecer a cada um dos fluxos. Isso só é fácil de determinar quando os fluxos são poucos e todos da mesma natureza, ou quando se reserva capacidade *a priori* como é possível nas redes de circuitos.

O mecanismo ECN Notificação Explícita da Saturação (ECN – *Explicit Congestion Notification*) permite indicar explicitamente aos emissores que devem refrear o seu débito de emissão para não saturarem a rede. Infelizmente, requer a cooperação entre os comutadores e o nível transporte nos computadores e a sua adopção tem-se revelado lenta. Trata-se de uma implementação pragmática do *feedback* explícito, mais simples de implementar do que um mecanismo que indicasse créditos concretos.

Vamos a seguir ver um mecanismo de *feedback* implícito chamado RED, uma alternativa a ECN.

Supressão prematura de pacotes

O mecanismo ECN é interessante mas exige que os comutadores e os protocolos de transporte o suportem. Quase duas dezenas de anos depois de o mecanismo ter sido proposto para as redes TCP/IP, estima-se que apenas uma minoria dos comutadores IP o suportam, ver por exemplo a análise apresentada em [Grosvenor et al., 2015]. O mais fácil é mesmo suprimir pacotes quando há saturação (na verdade nessa altura não existem muitas outras alternativas!).

No entanto, não será melhor começar a suprimir pacotes antes de a situação ser dramática? Se quando a carga das filas de espera de um comutador começar a passar para lá do razoável, este suprimir prematuramente alguns pacotes, tomados ao acaso e distribuídos por diferentes fluxos, isso funcionaria como um aviso prévio para as extremidades dos fluxos que o melhor era começarem já a reduzir o débito, pois a carga da rede parece estar a aumentar para lá do razoável. Como quando perde um segmento isolado, o TCP executa o procedimento “*fast recovery*” (do controlo de saturação Reno e seus sucessores) e abrande o ritmo de emissão, e dado que o TCP é um dos protocolos mais usados, essa estratégia pode contribuir para diminuir a carga (e a potencial saturação) de toda a rede.

A mesma linha de raciocínio recomenda que a probabilidade de um pacote ser suprimido seja proporcional ao nível de saturação: muito elevada se esse nível for elevado, pequena, ou mesmo nula, no caso contrário. Por outro lado, é igualmente importante distribuir as supressões prematuras de pacotes pelos diferentes fluxos, com maior incidência nos de maior débito.

Esta ideia foi proposta por Sally Floyd e Van Jacobson [Floyd and Jacobson, 1993], foi baptizada RED (*Random Early Detection* ou *Random Early Drop*) e é implementada da seguinte forma. Quando um pacote chega a um comutador, este determina o estado da fila de espera em que o pacote vai ser colocado à espera de ser transmitido. Nessa altura é tomada a decisão de supressão ou aceitação do pacote usando um gerador de números pseudo-aleatórios. Suprimir pacotes aleatoriamente aumenta a probabilidade de distribuir os avisos pelos diferentes fluxos, com maior incidência naqueles que enviam mais pacotes, ou seja nos com maior débito.

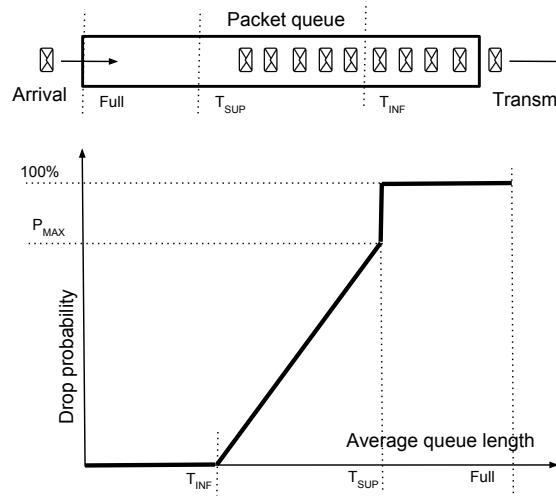


Figura 8.22: Função probabilidade de supressão de pacotes com RED

A supressão ou não de um pacote depende da dimensão da fila de espera e da função probabilidade que é usada para tomar a decisão. A dimensão da fila de espera usada

pelo RED não é a dimensão exacta, mas uma média calculada através de uma EWMA (Exponential Weighted Moving Average) semelhante à usada pelo TCP para calcular o RTT, ver a Secção 7.2. O objectivo é evitar que numa situação em que o tráfego médio é baixo, um pico momentâneo, que pode ser absorvido, seja excessivamente penalizado.

A função probabilidade do algoritmo RED usa diversos parâmetros, ver a Figura 8.22, nomeadamente a ocupação média da fila de espera abaixo da qual aos pacotes não devem ser suprimidos ($Threshold_{INF} = T_{INF}$), a ocupação média acima da qual todos os pacotes devem ser sempre suprimidos (T_{SUP}), e o andamento intermédio dessa probabilidade (dependente do valor de P_{max}) que determina a inclinação da recta intermédia. Se o valor de P_{MAX} for baixo, a probabilidade de suprimir pacotes quando a ocupação é média, i.e., entre T_{INF} e T_{SUP} nunca é muito alta. Quando $P_{MAX} = 1$, a distribuição das supressões é uniforme entre T_{INF} e T_{SUP} . A análise experimental permitiu concluir também que a probabilidade de suprimir um pacote deveria ir aumentando conforme aumentasse o número de pacotes aceites desde a última supressão, ou seja, a probabilidade deveria aumentar conforme foi passando mais tempo sem haver nenhuma supressão prematura. A listagem 8.5 apresenta de forma mais detalhada a forma de cálculo dos diferentes parâmetros.

Listing 8.5: Pseudo código da actualização dos parâmetros do algoritmo RED na sequência da recepção pelo comutador de mais um pacote

```

thresholdDiff = thresholdMax - thresholdInf
count = 0 // number of sent packets in sequence

// "gamma" defines the weight of the historic average when calculating
// the new average, i.e., it adjusts the sensibility to sudden changes
averageQueue = (1 - gamma) * averageQueue + gamma * currentMeasuredQueue

if (averageQueue < thresholdInf) dropProbability = 0
else if (averageQueue > thresholdMax) dropProbability = 1
else {
    p = pMax * (averageQueue - thresholdInf) / thresholdDiff
    dropProbability = p / (1 - count * p)
}

random = getRandom() // between 0 and 1
if ( random < dropProbability ) {
    drop packet
    count = 0
} else { count++ }

```

RED é uma técnica necessariamente menos interessante do que ECN porque suprime pacotes, mas é mais geral e pode ser usada em qualquer comutador e com qualquer transporte. O problema principal consiste em afinar os seus parâmetros, em função da quantidade de fluxos, da sua heterogeneidade, etc. de tal forma que no fim do dia se ganhe mais em evitar a saturação por supressão prematura de pacotes, do que sem usar o mecanismo. O estudo desses parâmetros é muito difícil sobre um protótipo onde se procure recriar as condições reais que ocorrem na Internet. A sua realização por simulação também é bastante difícil [Floyd and Paxson, 2001].

Em alternativa à notificação explícita da saturação, é possível suprimir prematuramente pacotes, na esperança que os respectivos emissores reduzam de forma equitativa o seu débito, como almeja o mecanismo de **Supressão prematura aleatória de pacotes (RED – Random Early Detection)**. Infelizmente, este mecanismo, simples de concretizar, é difícil de parametrizar de forma a que o mesmo se adapte dinamicamente à situação da rede e da diversidade dos fluxos e tenha mais proveito do que custos.

8.5 Equidade e desempenho do TCP

Caracterizar o desempenho do protocolo TCP tendo em consideração o seu algoritmo de controlo da saturação envolve várias facetas. Por um lado, é necessário analisar se a estratégia seguida pelo algoritmo conduz a uma distribuição equitativa da capacidade da rede entre os diferentes fluxos que a atravessam. Por outro lado é necessário analisar também o impacto do algoritmo sobre o desempenho de um fluxo tendo em consideração a capacidade disponível para o mesmo no *bottleneck link*. Começaremos por discutir o primeiro aspecto.

Equidade

Como vimos na secção 8.2, no contexto do controlo de saturação do TCP só faz sentido discutir equidade face à divisão da capacidade do *bottleneck link*. A pergunta que se coloca então é saber se perante várias conexões TCP que competem pela capacidade do mesmo *bottleneck link*, a estratégia AIMD (*Additive Increase Multiplicative Decrease*) conduz ou não a uma divisão equitativa dessa capacidade.

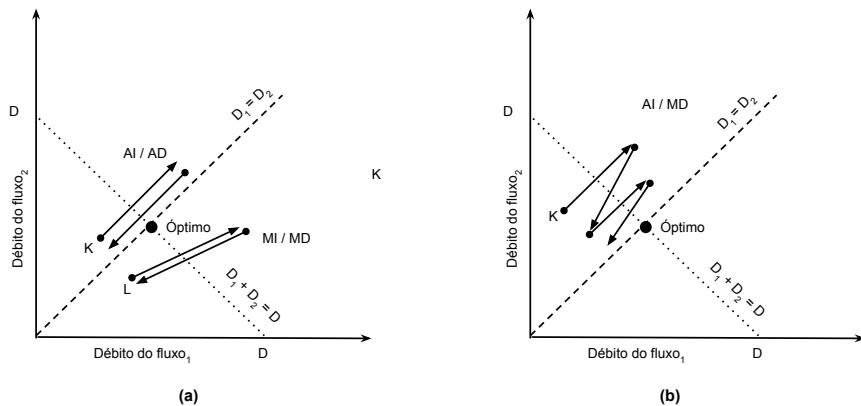


Figura 8.23: O mecanismo AIMD faz com que dois fluxos tendam para o equilíbrio

A resposta à questão no caso geral é difícil, mas no caso particular em que duas conexões TCP, com o mesmo RTT, partilham um *bottleneck link* com o débito D, a resposta é afirmativa. Chiu e Jain [Chiu and Jain, 1989] desenvolveram um esquema particularmente elegante para o mostrar, que se apresenta na Figura 8.23. Na discussão que se segue admite-se igualmente que ambas as conexões detectam os eventos *timeout* ou *triple duplicate ACK* simultaneamente.

Comecemos por perceber o gráfico (a), à esquerda. Qualquer ponto sobre esse gráfico representa um caso particular de partilha do débito D entre as duas conexões. A linha a ponteado, com a etiqueta $D_1 + D_2 = D$, representa as alternativas de partilha que consomem todo o débito do canal. A linha oblíqua a tracejado, com a etiqueta $D_1 = D_2$, representa os pontos em que a divisão do débito do canal é equitativa. O ponto de interceção das duas linhas corresponde à afectação óptima pois esta é equitativa e toda a capacidade do canal é afectada. Os pontos abaixo da linha a ponteado são afectações possíveis da capacidade pois representam pontos tais

que $D_1 + D_2 \leq D$, os pontos acima dessa linha conduzem a saturação visto que são afectações impossíveis pois $D_1 + D_2 > D$.

Dado o ponto K , que representa uma afectação não óptima visto que não é equitativa nem consome toda a capacidade disponível, a estratégia incremento e decremento aditivos (AIAD) aplicada a esse ponto, levaria as suas coordenadas (correspondentes aos débitos afectados aos fluxos 1 e 2) a serem ambas incrementadas ou decrementadas do mesmo valor, o que levaria K a dar lugar a um novo ponto em que ambas as coordenadas tinhamb sido incrementadas ou decrementadas do mesmo valor. Logo, esse novo ponto estará necessariamente sobre uma linha paralela à linha a tracejado e não se aproxima desta. Assim, o incremento ou o decremento aditivos não aproxima a solução da equidade, apenas afecta ou retira a mesma capacidade a cada uma das conexões e a ausência de equidade da afectação mantém-se.

Em contrapartida, o incremento e recuo multiplicativos (MIMD), como por exemplo o aplicado ao ponto L , mantém a proporção entre as duas coordenadas, pelo que a sua aplicação faz deslocar L ao longo de um segmento de recta com origem no ponto $(0, 0)$, correspondente à afectação (nulo, nulo) a cada uma das conexões. O incremento e o decremento de capacidade de forma multiplicativa só aproxima a afectação da equidade quando as capacidades afectadas decrescem, aproximando-se de 0, o que não é lá muito interessante. Assim, a estratégia AIAD (incremento e decremento aditivos) não aproxima a solução da equidade. A estratégia MIMD (incremento e decrecimento multiplicativos) só aproxima a partilha da equidade quando a afectação total se aproxima de 0. É a equidade do cemitério, onde todos estão igualmente mortos.

Se atentarmos agora no gráfico (b), à direita, partindo do ponto K aplica-se a estratégia AIMD, verifica-se que sempre que se sobe apenas se afecta mais capacidade a ambas as conexões sem corrigir a equidade, mas quando se desce o ponto aproxima-se da semi recta a tracejado dos pontos caracterizados por $D_1 = D_2$. Assim, no limite, a aplicação sucessiva desta estratégia coloca o ponto a oscilar sobre a linha a tracejado em que a afectação é equitativa e converge para o ponto da afectação óptima. A estratégia MIAD conduz ao afastamento da linha a tracejado e é portanto não equitativa.

A argumentação desenvolvida pelos autores Chin e Jain põe em evidência que, ao contrário das outras, a estratégia AIMD conduz à estabilidade e à equidade. De forma intuitiva também é fácil reconhecer que se trata de uma estratégia que trata a rede com prudência (AI) e que foge rapidamente da saturação (MD).

O argumento desenvolvido aplica-se a mais do que duas conexões mas deixa de poder aplicar-se sempre que o RTT das conexões é diferente ou as conexões TCP entram em competição com outros fluxos de pacotes que não aplicam de todo estratégias de controlo da saturação. A velocidade com que uma conexão TCP aumenta o seu débito durante a fase AI é tanto mais rápida quanto menor for o seu RTT, portanto as conexões com um menor RTT consomem uma fração superior da capacidade do canal quando estão em competição com outras com um RTT mais elevado. Mais: dado que estas serão mais frequentemente responsáveis pela saturação do canal, empurram as outras para baixo, visto que ambas recuam proporcionalmente ao ponto onde estavam, e quando recomeçarem a subir, as de menor RTT sobem de novo mais depressa do que as outras.

Por outro lado, se as conexões TCP entrarem em competição pela capacidade de um canal com fluxos de pacotes de débito fixo, que não aplicam nenhuma estratégia adaptativa, necessariamente perdem “a guerra”, pois os fluxos TCP diminuem o seu débito, enquanto os outros fluxos não. Por exemplo, se vários fluxos TCP competirem por um canal de débito D com fluxos UDP que consumem $1/2D$, os fluxos UDP consumirão sempre o débito que usam ($1/2D$) enquanto que os fluxos TCP apenas competirão pela capacidade que o UDP deixa livre. No limite, se os fluxos UDP consumirem toda a capacidade D , as conexões TCP praticamente “morrem”, pois só conseguirão tentar enviar um único segmento por RTT, e como este frequentemente se

perde, só tentam de novo após disparar o *timeout*, mas como de cada vez duplicam o valor do temporizador usado, a situação só poderá agravar-se até que só tentam enviar (melhor dizendo reenviar) um segmento de vários em vários segundos.

Finalmente, para terminarmos esta discussão sobre a equidade, é necessário ter em atenção que a divisão do débito do *bottleneck link* é feita por conexão TCP e não por computador. Assim, uma maneira de obter uma fração superior do débito partilhado é abrir várias conexões em paralelo ao invés de uma só. Esta estratégia é usada com frequência por aplicações P2P de partilha de conteúdos.

É possível demonstrar que a estratégia AIMD tende para a estabilidade e equidade na partilha de um *bottleneck link* entre várias conexões TCP com o mesmo RTT. No entanto, se as conexões TCP tiverem RTTs distintos a equidade não é garantida.

Por outro lado, o TCP não pode competir com fluxos de pacotes que não aplicem a mesma estratégia de controlo de saturação, pois reduz sempre o débito de emissão independentemente da estratégia seguida pelos seus competidores.

Isto mostra igualmente que quando se introduzem alterações no protocolo TCP, é necessário garantir que estas mantêm a equidade para com as versões anteriores do protocolo.

Após termos olhado de perto a problemática da equidade, vamos a seguir discutir o problema do desempenho efectivo de uma conexão TCP que está a usar o algoritmo de controlo da saturação Reno.

Desempenho

No final da secção 7.3 o débito médio extremo a extremo de uma conexão TCP foi grosseiramente caracterizado como sendo proporcional a W/RTT , em que W representa a dimensão média da janela em bits e RTT tem o significado habitual. O problema é que, como sabemos agora, W varia todo o tempo. Por outro lado, varia de forma diferente conforme a duração da conexão e o comportamento dos competidores.

Para reanalisarmos a questão vamos usar de novo um modelo relativamente simples. Admitamos que a conexão tem uma grande duração e que está continuamente a transferir dados. Admitimos também que o *bottleneck link* está no interior da rede. Com efeito, se o *bottleneck link* for o canal que liga o emissor à rede, a velocidade de transferência extremo a extremo será a desse canal, pois os mecanismos de gestão dos buffers do computador emissor levarão a que o ritmo de emissão seja o permitido por esse canal e, nessa situação, em princípio, não ocorre perda de pacotes por saturação. Admitimos igualmente que a dimensão da janela do receptor não limita a dimensão da janela do emissor através de controlo de fluxos. Podemos também considerar que a maioria das perdas de pacotes são tratadas por *fast retransmit* e desprezamos a ocorrência de *timeouts*.

Com estas hipóteses, o débito de extremo a extremo medido em termos do quociente W/RTT terá um andamento semelhante ao sugerido pelo gráfico da Figura 8.24 e estará compreendido no intervalo $1/2D$ e D , em que D representa a fração do débito do *bottleneck link* que cabe à conexão. Uma estimativa grosseira indica que em média, estimada a longo prazo, o débito médio extremo a extremo deverá ser $\approx 3/4D$.

O modelo usado é muito simples e usa hipóteses muito fortes: D é constante e os competidores são igualmente estáveis, a conexão é muito longa e tem sempre dados

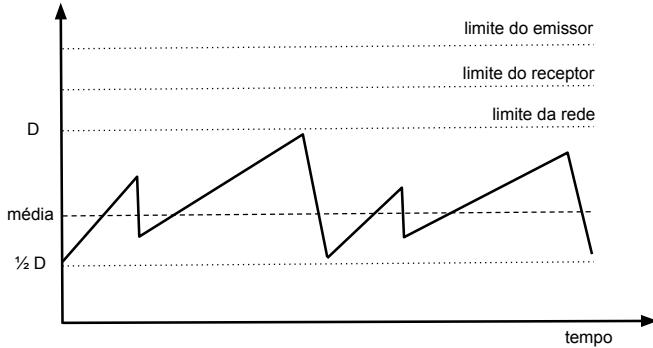


Figura 8.24: Débito de uma conexão TCP idealizada

para transmitir, não surgem novas conexões (que durante a fase *slow start* inicial provocam necessariamente saturação), não há perdas de pacotes a não ser esporádicas, a aplicação e o receptor não impõem limitações mais estritas do que a rede ao débito, e a capacidade do canal que liga o emissor à rede não refreia o seu débito de emissão. Naturalmente, o modelo não se aplica quando essas hipóteses não são verdadeiras.

É interessante discutir várias situações em que o modelo acima não se aplica pois algumas são frequentes e relevantes.

Conexões muito curtas

O acesso à Web é feito sobre TCP e frequentemente são abertas conexões para servidores para obter objectos de pequena dimensão (20 KBytes por exemplo). Logo que o cliente obtém o objecto fecha a conexão. Com um MSS de 1250 bytes, 20 KBytes correspondem a 16 segmentos. Devido à abertura de conexão e ao mecanismo *slow start*, no fim do 2º RTT, o cliente enviou o pedido e recebeu 1 segmento, no fim do 3º, recebeu 3 segmentos, no fim do 4º, recebeu 7 segmentos, no fim do 5º, recebeu 15 segmentos e no fim do 6º, já recebeu a totalidade dos 16 segmentos.

Em muitas situações a janela inicial poderia ser superior a 1 segmento. Por exemplo, se a janela inicial começasse com 4 segmentos, no fim do 2º RTT, o cliente já teria recebido 4 segmentos, no fim do 3º, já teria recebido 12 segmentos e no fim do 4º já teria recebido até 28 segmentos. O tempo para obtenção do objecto baixaria para 66% do anterior apenas porque a janela começou com 4 segmentos. Se o RTT for 100 ms, o que pode acontecer no acesso a servidores remotos, o objecto seria obtido em 400 ms ao invés de em 600 ms. A conexão TCP normal colocou a hipótese inicial de dispor apenas da capacidade de $1250 \times 8/0,1 \approx 100$ Kbps no *bottleneck link*, tendo chegado depois a usar 800 Kbps, enquanto que no segundo caso colocaria a hipótese de dispor inicialmente de 400 Kbps e depois de 800 Kbps.

Sempre que for possível começar com janelas maiores, tal é muito interessante para conexões de pequena duração. Na Internet actual a disponibilidade das capacidades indicadas é vulgar. No entanto, o que aconteceria em situações em que o *bottleneck link* não tivesse essas capacidades disponíveis? A simples abertura da conexão provocaria saturação. Idealmente deveria ser possível durante a abertura da conexão ter uma melhor estimativa da capacidade disponível para a mesma. De qualquer forma, as versões actuais de TCP usam janelas iniciais com a dimensão de vários segmentos.

O RFC 3390 recomenda que a janela inicial seja assim dimensionada

$$\min(4 \times \text{MSS}, \max(2 \times \text{MSS}, 4380 \text{ bytes}))$$

o que conduz com frequência a janelas de 4380 bytes, que correspondem geralmente a 3 segmentos pois 1460 bytes é um valor comum para o MSS. No entanto, o mesmo RFC abre a hipótese de se usarem janelas ainda maiores, o que se tornou prática corrente sobretudo quando se detectam RTTs elevados durante a abertura da conexão.

Perdas consecutivas

O algoritmo Reno baseia-se na hipótese de que quando a saturação se aproxima, têm lugar algumas perdas esporádicas de pacotes detectadas pelo evento *triple duplicate ACK*. No entanto, verificou-se que existem situações em que este tipo de perdas são consecutivas.

Nesta situação o algoritmo *fast recovery* do Reno tem como repercussão a descida exponencial da janela de saturação. Com efeito, com perdas sucessivas, o valor da variável `sst` diminui para próximo de alguns MSSs, o valor do *timeout* aumenta muito, e a conexão, mesmo se deixasse de perder pacotes a seguir, permaneceria muito tempo no estado *congestion avoidance* em que a janela é incrementada de forma linear.

Para lidar com este problema foram introduzidos dois algoritmos novos, um integrado com o tratamento dos ACKs selectivos chamado TCP SACK, e outro chamado New Reno.

Canais sem fios

O algoritmo de controlo da saturação põe a hipótese de que a maioria das perdas de pacotes deve a saturação e não a erros. Esta hipótese verifica-se nos canais com fios e sobretudo nos canais baseados em fibra óptica. No entanto, os canais sem fios têm uma elevada taxa de erros, o que frequentemente implica perda de pacotes por erros nos canais.

O TCP não consegue distinguir os diferentes tipos de perda de pacotes e interpreta-os a todos da mesma forma. Muitas vezes isso implica o disparo de um alarme e a redução da dimensão da janela a um segmento, ou pelo menos a sua redução a metade se o erro for mais esporádico.

É difícil encontrar uma versão do algoritmo que lide bem com este tipo de situações. Acresce que a prática mostrou que as redes celulares sem fios introduzem RTTs artificialmente elevados, o que prejudica a recuperação do TCP numa situação em que a janela foi inutilmente reduzida. Para uma discussão preliminar destes problemas em redes sem fios o leitor poderá consultar [Mascolo et al., 2001], onde os autores propõem um método de estimar a capacidade efectivamente disponível para a conexão TCP o que permite controlar de forma mais rigorosa e suave as variações da janela do emissor.

Soluções nesta linha voltarão a ser discutidas no fim da secção onde são apresentados algoritmos de estimativa da capacidade extremo a extremo realmente disponível. Essa estimativa, se rigorosa, permite evitar a congestão e o recuo excessivo da dimensão da janela quando há perdas esporádicas de pacotes.

Não se têm encontrado soluções totalmente satisfatórias usando controlo da saturação tradicional, baseado em *feedback* implícito, para situações em que ocorrem erros esporádicos. As alternativas de maior sucesso e independentes do TCP consistem em tentar mascarar os erros dos canais sem fios através de mecanismos de recuperação dos erros directamente na extremidade desses canais, e portanto transparentes para os níveis superiores. Infelizmente, essas soluções aumentam ainda mais o tempo de trânsito. A outra alternativa é colocar *caches* dentro das redes móveis celulares para que seja desnecessário, se possível, ter conexões com elevado RTT e atravessando muitos canais distintos. Outra alternativa consiste em “partir” a conexão em duas, uma correspondente ao canal sem fios, e outra correspondente aos restantes canais.

Trata-se de mais um caso que ilustra que os *end-to-end arguments* não se podem aplicar de forma simplista.

Reordenação de pacotes

Apesar de a qualidade de serviço do tipo “melhor esforço”, que caracteriza o protocolo IP, comportar a reordenação de pacotes, os algoritmos Tahoe e Reno interpretam a reordenação como uma perda de pacotes que desencadeia eventualmente um evento *triple duplicate ACK*. Existem situações de alteração do encaminhamento, que podem conduzir a reordenação de pacotes que são de facto esporádicas. No entanto, podem também suceder situações de distribuição de carga no interior da rede que provoquem reordenação de pacotes. O comportamento do TCP nessas situações conduz a uma redução dramática do desempenho da conexão.

Foram feitas diversas alterações dos algoritmos para lidar com este problema e também se tomaram medidas para que a distribuição de carga no interior da rede fosse feita de tal forma que diminuisse drasticamente a probabilidade de ocorrer reordenação de pacotes.

Ausência de equidade com diferentes RTTs – *RTT unfairness*

Diversos algoritmos atacam este problema e incluem alterações para tentar minorar os seus efeitos. A estratégia comum a esses algoritmos consiste em acelerar a subida da janela de saturação durante a fase *congestion avoidance*, sem no entanto introduzir uma subida multiplicativa. Desta forma as conexões com RTTs grandes são menos penalizadas face a conexões com RTTs curtos, as quais sobem mais rapidamente a sua janela de saturação durante a fase *congestion avoidance*.

Esta estratégia é também aplicada para atacar o problema a seguir apresentado.

Canais com grande volume

O TCP é usado para transferir dados para *smartphones* ligados por canais com baixa capacidade, mas também para transferir objectos de grande dimensão entre centros de dados. As redes que interligam esses centros de dados dispõem de canais dedicados com débitos muito elevados (*e.g.*, 10 Gbps) e um tempo de trânsito de centenas de milissegundos pois os centros de dados estão em diferentes continentes. Como se porta o TCP nesses canais?

Com um MSS de 1250 bytes (10^4 bits) e um RTT de 100 ms, para encher um canal com a capacidade de 10 Gbps (10^{10} bits por segundo), ou seja para encher o seu volume, são necessárias janelas com 10^9 bits, logo com 100.000 segmentos. Se a dimensão da janela a certa altura for metade, 50.000 segmentos, o algoritmo de incremento aditivo junta à janela mais 10 segmentos em cada segundo (1 segmento por RTT), logo seriam necessários 5.000 segundos sem erros (uma hora tem 3.600 segundos) para que a janela subisse até próximo de uma dimensão que permitisse explorar a capacidade disponível. Claramente, a parte que lida com o controlo da saturação do TCP Reno não chega para lidar com canais de grande capacidade e um elevado RTT.

Existe um resultado teórico que indica que a velocidade média de transferência extremo a extremo do TCP (débito extremo a extremo ou $D_{end-to-end}$) pode ser estimado pela equação

$$D_{end-to-end} = \frac{1.22 \times MSS}{RTT \times \sqrt{\rho}}$$

onde ρ representa a taxa de pacotes recebidos com erros. Se o RTT for da ordem de grandeza de 100 ms, o MSS da ordem de grandeza de 12.000 bits, para se atingirem velocidades de transmissão da ordem de grandeza de 10 Gbps é necessário que haja, no máximo, um erro em cada 5.000.000.000 segmentos o que é uma taxa praticamente impossível mesmo com fibra óptica.

Mesmo que os canais tenham capacidades bastante inferiores disponíveis para a conexão TCP, torna-se claro que os algoritmos de controlo da saturação com base em *feedback* implícito e na perda de segmentos devem ser melhorados em situações onde são necessárias janelas de grande dimensão para obter um bom rendimento. Por essa

razão existem inúmeros trabalhos para melhorar o desempenho do TCP em geral e nestas situações em particular.

Evitar a saturação

Num quadro em que a rede continue a não dar *feedback* explícito sobre o débito recomendado para uma conexão (o que evitaria que a conexão tenha que provocar saturação para o estimar), a alternativa consiste em tentar estimar esse débito usando o RTT e o débito real, assim como as respectivas variações. Adicionalmente, isso resolveria o problema do funcionamento do TCP sobre canais com grande volume. Dual e Vegas são dois algoritmos baseados nesta ideia.

O TCP Dual estima o RTT mínimo e máximo da conexão. O mínimo é o RTT mais curto observado, por exemplo na abertura da conexão, e o máximo é o RTT observado durante os períodos de saturação. Naturalmente, a diferença entre os dois dá uma indicação sobre o tempo perdido pelos pacotes em filas de espera. Finalmente, um determinado factor α , entre 0 e 1, é usado para estimar um valor limite da diferença entre o RTT observado e o RTT mínimo a partir do qual a saturação está próxima. Os autores do TCP Dual usavam $\alpha = 1/2$.

Quando a diferença entre o RTT corrente e o RTT mínimo ultrapassa o limite, a janela de saturação é descida de $1/8$. Senão, a janela pode crescer aditivamente. O algoritmo melhora claramente o desempenho de uma conexão única, mas revelou-se incapaz de assegurar equidade pois as conexões são incapazes de estimar correctamente o RTT mínimo correspondente a uma rede sem carga.

Com efeito, a primeira conexão que encontrasse a rede sem carga estimaria correctamente o RTT mínimo, mas uma conexão que tivesse início com a rede carregada, não conseguia estimar tal RTT. Por outro lado, o método de estimar o melhor desempenho só a partir do RTT revelou-se insuficiente. O TCP Vegas tentou melhorar diversos desses aspectos.

Tal como o TCP Dual, o TCP Vegas tenta igualmente evitar a saturação (*congestion avoidance*) estimando um valor de janela de saturação correspondente ao valor máximo que não sature a rede.

Com este objectivo, Vegas estima periodicamente o valor do RTT base, que é um valor de RTT mínimo mas que é ajustado periodicamente. A partir desse RTT base, o valor de `congWnd` usado em cada momento permite calcular

$$\text{expectedRate} = \text{congWnd}/\text{RTT}_{\text{base}}$$

que corresponde ao débito potencial quando a janela tem o valor `congWnd`, ver a Figura 8.25.

No entanto, contando o número de bytes transmitidos desde que um segmento é emitido até que o seu ACK chegue, numa situação sem perda de pacotes, é possível calcular o `actualRate` e o seu valor estabiliza a partir do momento em que a conexão está a usar a sua fração do *bottleneck link*, o que permite calcular a janela correspondente a esse débito.

A seguir a diferença $\text{diff} = \text{expectedRate} - \text{actualRate}$ é calculada. O valor de `diff` é sempre ≥ 0 porque se $\text{actualRate} > \text{congWnd}/\text{RTT}_{\text{base}}$ isso significa que RTT_{base} deveria ser alterado para o último RTT medido. São também definidos dois limites para `diff`, respectivamente o limite inferior α , e o limite superior β . Quando `diff` está abaixo de α , é praticado o incremento aditivo de `congWnd`. Quando `diff` está acima de β , `congWnd` é reduzida de $1/8$. Quando está no meio, não se altera a dimensão da janela de saturação.

A ideia base é a seguinte: quando `diff` está acima de β a saturação está próxima pois o débito real começa a ser inferior ao que a janela de saturação deveria permitir se o RTT estivesse ao nível de RTT_{base} , e a janela de saturação é reduzida de $1/8$. Quando `diff` está abaixo de α , existe provavelmente a oportunidade de aumentar a

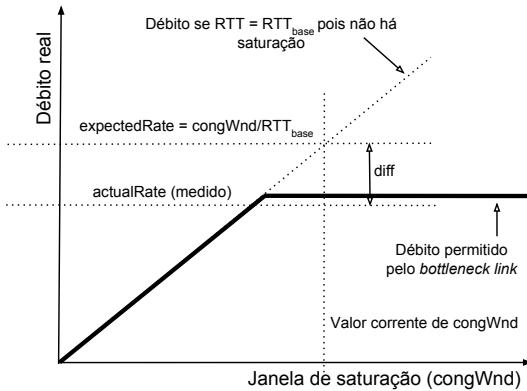


Figura 8.25: TCP Vegas: `expectedRate` e `actualRate` de uma conexão

janela de emissão e os dados em trânsito. Senão, não se altera o valor da janela pois é provável que a conexão esteja a funcionar à “velocidade de cruzeiro”.

TCP Vegas comporta-se relativamente bem mas é incapaz de competir com os derivados mais recentes de TCP Reno, obtendo uma fração inferior do *bottleneck link* pois Vegas é menos “atrevido” que Reno a aumentar a janela de saturação. Tal mostra que existe hoje em dia um problema suplementar para quem inventar um novo algoritmo de controlo da saturação e que consiste em tomar em consideração a sua coexistência com os já existentes. Algumas das ideias de TCP Vegas foram de qualquer forma retidas noutras propostas.

O débito extremo a extremo permitido pelo algoritmo de controlo da saturação Reno situa-se, em condições ideais, entre o débito disponível no *bottleneck link* para a conexão e metade do mesmo. Este resultado está relacionado com a necessidade que esse algoritmo tem de provocar saturação, para a evitar em seguida, e com o funcionamento AIMD, que usa para regular o débito.

O desempenho do algoritmo revela-se insuficiente quando as condições da rede, ou as características das conexões, estão afastadas do modelo de Internet para que foi concebido: conexões de longa duração, sobre canais com fios, com pequeno RTT e com relativamente baixo débito (alguns Mbps no máximo), e rede relativamente homogénea e com taxa de erros residual.

Muitas destas condições não se verificam hoje em dia e os algoritmos de controlo da saturação do TCP têm sofrido inúmeras melhorias para os adaptar a novos requisitos. Adicionalmente, têm sido introduzidos inúmeros refinamentos das propostas iniciais com o objectivo de melhorar o desempenho das conexões *etc.*

A dificuldade do problema advém também da utilização do mesmo protocolo em condições de utilização e de rede muito heterogéneas, abrangendo redes sem fios com taxas de erros elevadas, redes que reordenam pacotes, redes tradicionais mas com elevado RTT, conexões de pequena dimensão, redes com canais de elevado volume (débito muito elevado e RTT muito elevado).

Se o emissor conhecesse a capacidade que a rede lhe disponibiliza em cada momento, seria possível ter um desempenho superior. É o que tentou a versão TCP Vegas, que se baseia na ideia de evitar a saturação e não de a controlar. Infelizmente, não é fácil introduzir um algoritmo com uma nova filosofia caso o mesmo se revele incapaz de competir com as versões dominantes, pois isso põe em causa a equidade entre fluxos.

8.6 Resumo e referências

Resumo

Uma rede de computadores entra em saturação quando o conjunto das solicitações de encaminhamento de pacotes que recebe a conduzem a uma situação de colapso, e a quantidade real de pacotes encaminhados é inferior à sua capacidade efectiva. É uma situação análoga à que sucede nas redes viárias quando estas não suportam condições de tráfego automóvel anormais e têm lugar engarrafamentos.

Os mecanismos de controlo da saturação são mecanismos que procuram controlar dinamicamente o débito com que os pacotes são injectados na rede, tendo em consideração a situação real e a capacidade disponível, de forma a maximizarem o rendimento da rede e a entrega de pacotes aos seus destinos. As arquitecturas, protocolos e algoritmos usados para controlar as solicitações à rede, de forma a evitar que esta entre em saturação, chamam-se arquitecturas, protocolos e algoritmos de controlo da saturação.

Tirar o melhor rendimento possível da rede, sem a saturar e assegurando equidade entre os diferentes fluxos de pacotes, é um problema de optimização. Calcular a solução óptima implica dispor de uma visão global do estado da rede, dinamicamente actualizada em função da evolução das solicitações e, adicionalmente, é necessário conseguir controlar os computadores ligados à rede para controlar o seu ritmo de emissão de pacotes. Este tipo de arquitectura não é realista senão numa situação em que os gestores da rede tivessem controlo completo sobre os computadores que lhe estão ligados. No caso geral, o máximo que a rede pode esperar é alguma colaboração dos protocolos de transporte que esses computadores usam.

Geralmente o débito máximo que um fluxo de pacotes pode usar é condicionado pela capacidade disponível para o fluxo no canal mais carregado entre aqueles que o fluxo atravessa. Esse canal chama-se o canal gargalo (*bottleneck link*).

O controlo da saturação necessita de regulação do débito dos fluxos de pacotes emitidos pelos computadores ligados à rede. Este controlo pode basear-se em imposição pela rede do débito máximo, ou em *feedback* da rede aos computadores para que estes se auto-regulem. As arquitecturas baseadas em *feedback* da rede dividem-se ainda em *feedback* explícito, quando a rede dá indicações directas aos computadores para que estes se auto-regulem, e *feedback* implícito, quando são os computadores que detectam os indícios de saturação e voluntariamente se auto-regulam.

As redes TCP/IP foram definidas para utilizarem por omissão *feedback* implícito e o protocolo de transporte alvo é o TCP. Como um emissor TCP não tem noção de qual o estado da rede, nem por omissão recebe *feedback* desta, tem de experimentar ir aumentando o ritmo de emissão e, quando se aperceber que há problemas, deverá diminui-lo. A técnica de ir incrementando o ritmo de transmissão para tentar aproximar a capacidade máxima da rede sem a saturar, chama-se sondar os recursos da rede (*network resource probing*).

O protocolo TCP varia o seu ritmo de emissão máximo usando a fórmula “subida aditiva, descida multiplicativa” (*AIMD – Additive Increase, Multiplicative Decrease*). O protocolo usa a dimensão da janela de emissão para concretizar quer um mecanismo de controlo de fluxo (*flow control*), que adapta o ritmo de transmissão à capacidade do receptor consumir atempadamente os dados recebidos, quer um mecanismo de controlo

da saturação (*congestion control*), que adapta o ritmo de transmissão à capacidade disponível na rede para a conexão TCP.

O algoritmo de controlo da saturação do TCP utiliza uma variável, designada janela de saturação (*congWnd*), que limita a dimensão máxima da janela de emissão. Em cada momento, o protocolo tenta estimar a capacidade máxima disponível para a conexão no seu *bottleneck link*, subindo aditivamente o ritmo de emissão e usando os *timeouts* e os *triple duplicate ACKs* como indícios de saturação. Quando estes eventos têm lugar, a capacidade disponível é então estimada como sendo a do momento da detecção dessas ocorrências e aplica-se uma descida multiplicativa do ritmo de emissão.

Inicialmente, a janela de saturação tem a dimensão do MSS (*Maximum Segment Size*) e vai sendo duplicada em cada RTT, durante uma fase do algoritmo designada *slow start*. Quando a janela de saturação atinge um valor equivalente a metade da capacidade máxima estimada, entra numa fase chamada *congestion avoidance*, durante a qual a janela é incrementada aditivamente do valor do MSS (em bytes) em cada RTT. Na sequência de um *timeout*, a janela de saturação é reduzida de novo à dimensão do MSS e o emissor volta a entrar na fase *slow start*. Na sequência de um *triple duplicate ACK*, a janela de saturação é reduzida a metade da capacidade máxima estimada e o emissor continua na fase *congestion avoidance* até surgir de novo um *timeout* ou um evento *triple duplicate ACK*. Este algoritmo recebeu o nome de TCP Reno.

O algoritmo TCP Reno constituiu uma primeira solução para um problema complexo dada a diversidade das situações em que o controlo de saturação tem de actuar. Num caso extremo, a conexão pode atravessar uma área de rede pequena, homogénea, bem dimensionada e com RTT reduzido. No outro extremo, a conexão atravessa uma grande quantidade de canais heterogéneos, com níveis de utilização muito diversos e apresentando um RTT elevado ou até erros de transmissão, erroneamente interpretados como sinais de saturação, uma situação frequente em canais sem fios.

Em alternativa a soluções de controlo da saturação baseadas em *feedback* implícito, é possível utilizar soluções baseadas em *feedback* explícito. O mecanismo ECN – Notificação Explícita da Saturação (*Explicit Congestion Notification*) permite indicar explicitamente aos emissores TCP que devem refrear o seu débito de emissão para não saturarem a rede. Infelizmente, o mecanismo requer a cooperação entre os comutadores e o nível transporte nos computadores e a sua adopção tem-se revelado lenta. Trata-se de uma implementação pragmática de *feedback* explícito, mais simples de implementar do que um mecanismo que indicasse ritmos explícitos a serem utilizados pelas diferentes conexões.

Em alternativa à notificação explícita da saturação, é possível suprimir prematuramente pacotes, na esperança que os respectivos emissores reduzam de forma equitativa o seu débito, como almeja o mecanismo de supressão prematura, aleatória, de pacotes (RED – *Random Early Detection*). Infelizmente este mecanismo, simples de concretizar, é difícil de parametrizar de forma a que o mesmo se adapte dinamicamente à situação da rede e da diversidade dos fluxos e conduza a mais proveitos que custos.

É possível demonstrar que a estratégia AIMD tende para a estabilidade e equidade na partilha de um *bottleneck link* entre várias conexões TCP. No entanto, se as conexões TCP tiverem RTTs distintos a equidade não é garantida. Por outro lado, o TCP não pode competir com fluxos de pacotes que não apliquem uma estratégia de controlo de saturação, pois reduz sempre o débito de emissão independentemente da estratégia seguida pelos seus competidores.

O débito extremo a extremo permitido pelo algoritmo de controlo da saturação Reno situa-se, em condições ideais, entre o débito disponível no *bottleneck link* para a conexão e metade do mesmo. Este resultado está relacionado com a necessidade que esse algoritmo tem de provocar saturação, para a evitar em seguida, e com o funcionamento AIMD que usa para regular o débito.

O desempenho do algoritmo revela-se insuficiente quando as condições da rede, ou as características das conexões, estão afastadas do modelo de Internet para que foi

concebido: conexões de longa duração, sobre canais com fios, com pequeno RTT e com débito relativamente baixo (alguns Mbps no máximo), e rede relativamente homogénea com taxa de erros residual.

Muitas destas condições não se verificam hoje em dia e os algoritmos de controlo da saturação do TCP têm sofrido inúmeras melhorias para os adaptar a novos requisitos. Adicionalmente, têm sido introduzidos inúmeros refinamentos da proposta inicial com o objectivo de melhorar o desempenho das conexões e o rendimento da rede. A dificuldade do problema advém também da utilização do mesmo algoritmo em condições de utilização e de rede muito heterogéneas, que vão desde redes sem fios, com taxas de erros elevadas, redes que reordenam pacotes, redes tradicionais mas com elevado RTT, conexões de pequena dimensão, redes com canais de elevado volume (débito muito elevado e RTT muito elevado), etc.

Se o emissor conhecesse a capacidade que a rede lhe disponibiliza em cada momento, seria possível ter um desempenho superior. É o que tentou a versão TCP Vegas que se baseia na ideia de evitar a saturação e não de a controlar. Infelizmente, não é fácil introduzir um algoritmo com uma nova filosofia caso o mesmo se revele incapaz de competir em pé de igualdade com as versões dominantes pois isso põe em causa a equidade entre fluxos.

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

Controlo da saturação (*congestion control*) Mecanismos que procuram controlar dinamicamente o débito com que os pacotes são injectados na rede, tendo em consideração a situação real e a capacidade disponível, de forma a maximizar o rendimento da rede e a entrega de pacotes aos seus destinos.

Débito útil (*goodput*) Débito real de informação entregue pela rede aos destinatários, isto é, descontando todo o desperdício introduzido pelos protocolos, quer em termos de cabeçalhos dos pacotes, quer em termos de pacotes reemitidos e recebidos em duplicado.

Poder da rede (*network power*) Quociente do débito útil (*goodput*) pelo tempo de trânsito de extremo a extremo. Numa rede saturada, o aumento do débito de injecção de pacotes na rede conduz à diminuição do seu poder pois o *goodput* diminui e o RTT aumenta.

Canal gargalo (*bottleneck link*) Canal que impõe o débito máximo disponível para um fluxo.

Sondar a rede (*Network resource probing*) Técnica que consiste em ir incrementando o ritmo de transmissão para tentar aproximar a capacidade máxima da rede, sem a saturar.

AIMD (*Additive Increase, Multiplicative Decrease*) Forma de controlo do débito de um emissor que conduz a uma utilização equitativa do canal gargalo e a uma solução com estabilidade para o problema do controlo da saturação.

Controlo da saturação com base em feedback O controlo da saturação necessita de regulação do débito dos fluxos de pacotes emitidos pelos computadores ligados à rede. Este controlo pode basear-se em imposição pela rede do débito máximo ou em *feedback* da rede aos computadores para que estes se auto-regulem.

Feedback explícito Técnica de controlo da saturação que se baseia em a rede dar indicações directas aos computadores para que estes se auto-regulem.

Feedback implícito Técnica de controlo da saturação que se baseia em os computadores detectarem os indícios de saturação. Os indícios usados são a perda de pacotes e as variações do débito útil e do RTT.

Notificação Explícita da Saturação (*ECN – Explicit Congestion Notification*) Mecanismo que permite indicar explicitamente aos emissores que devem refrear o seu débito de emissão para não saturarem a rede.

Equidade (*Fairness*) Propriedade da regulação do débito de um conjunto de fluxos que é máxima quando nenhum dos fluxos recebe melhor serviço em detrimento do serviço prestado aos fluxos com que compete. A equidade é mínima quando só um fluxo recebe serviço e os restantes nenhum.

Evitar a saturação (*Congestion avoidance*) Técnica de controlo da saturação que se baseia em evitar a saturação sem a provocar. Os indícios usados para detectar a proximidade da saturação são as variações do débito útil e do RTT.

Referências

A primeira proposta para introdução de controlo de saturação no TCP foi feita por V. Jacobson [Jacobson, 1988]. O RFC 6582 começa com um resumo onde se pode ler: “RFC 5681 documents the following four intertwined TCP congestion control algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery. RFC 5681 explicitly allows certain modifications of these algorithms, including modifications that use the TCP Selective Acknowledgment (SACK) option (RFC 2883), and modifications that respond to “partial acknowledgments” (ACKs that cover new data, but not all the data outstanding when loss was detected) in the absence of SACK. This document describes a specific algorithm for responding to partial acknowledgments, referred to as “NewReno””. Este resumo testemunha que os algoritmos de controlo da saturação introduzidos inicialmente foram sendo sucessivamente melhorados de forma a incorporarem o tratamento de novas situações, não totalmente bem cobertas pelas versões anteriores.

Numa outra linha de abordagem do problema da saturação no protocolo TCP, a primeira proposta de *congestion avoidance* para o TCP foi o algoritmo Dual [Wang and Crowcroft, 1992]. O algoritmo TCP Vegas está descrito em [Brakmo et al., 1994]. Apesar de inicialmente esta abordagem não ter sido muito adoptada nas implementações usadas em produção do TCP, a investigação em torno destas propostas continuou com alguma intensidade e influenciou versões recentes dos algoritmos. O controlo da saturação com base em *feedback* explícito da rede, o mecanismo ECN, está descrito no RFC 3168. O mecanismo RED está descrito em [Floyd and Jacobson, 1993].

Com a subida da capacidade dos canais da Internet e a generalização de canais sem fios, acontecimentos que tiveram maior expressividade já no presente século, houve um renovado interesse em melhorar os algoritmos de controlo e para evitar a saturação, com o objectivo de melhorar o desempenho do protocolo TCP. Esses esforços traduziram-se em várias dezenas de propostas, algumas das quais foram introduzidas como melhoramentos dos algoritmos usados até então. As sínteses (*surveys*) [Afanasyev et al., 2010; Abed et al., 2011] recenseiam muitos desses esforços. O *survey* [Chen et al., 2005] é especialmente focado nos problemas relacionados com o desempenho do TCP nas redes sem fios. Le Boudec [Boudec, 2014] apresenta uma síntese de índole mais teórica sobre o problema do controlo da saturação.

Como resultado de todos estes esforços, e tendo em consideração a impossibilidade de dispor de um único algoritmo, por muitas variantes que incorpore para lidar com todas as diferentes situações, os sistemas de operação modernos passaram a incorporar diversos algoritmos de controlo da, ou para evitar a saturação, na implementação do protocolo TCP. Por omissão, o sistema usa uma versão, mas os utilizadores, através de módulos do núcleo do sistema, de opções dos *sockets* TCP ou através de outros mecanismos de controlo do núcleo, podem alterar a versão usada. Os algoritmos disponíveis vão variando no tempo, e o leitor interessado terá de analisar o seu caso, recorrendo à documentação específica aplicável ao seu sistema.

O sistema Mac OS X, dado usar um núcleo baseado na versão de sistema Unix FreeBSD, utilizava por omissão a versão New Reno, mas é provável que incorpore actualmente diversas outras versões. No sistema de operação Linux uma versão muito usada por omissão é designada TCP Cubic [Ha et al., 2008], mas o sistema permite

hoje em dia optar entre mais de 10 versões distintas. O sistema de operação Windows incluia uma versão chamada Compound TCP [TAN et al., 2006] mas também permite a opção por outras versões. Os sistemas iOS e Android, para pequenos dispositivos móveis, usam versões especiais do núcleo FreeBSD e Linux, e provavelmente usam versões de controlo da saturação no TCP especialmente adequadas ao ambiente móvel.

Apontadores para informação na Web

- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- <http://en.wikipedia.org/wiki/Iperf> – Contém informação sobre o programa **iperf**. Este programa pode ser usado para testar o desempenho de versões dos algoritmos de controlo de saturação disponíveis. Em complemento com um emulador ou um laboratório real de redes, é particularmente útil para realizar estudos comparativos.
- <https://www.wireshark.org> – Contém informação sobre o **wireshark**.
Este programa é um monitor dos pacotes que atravessam uma interface de rede de um computador. Como pode registar todos os dados contidos nos cabeçalhos TCP, permite a partir dos mesmos representar graficamente a evolução da janela e outros dados do cabeçalho.
- <http://www.web10g.org> – Contém informação sobre o módulo **Web10G**.
Este módulo, para o núcleo do sistema Linux, permite aceder a todas as variáveis internas usadas pela implementação do TCP.
- <http://www.linuxfoundation.org> – Permite aceder à documentação do módulo **tcpprobe** para Linux. É um módulo que fornece dados como o módulo **Web10G**, mas de forma mais sintética.

8.7 Questões para revisão e estudo

1. Verdade ou mentira? Justifique a sua resposta.
 - (a) Durante a fase *congestion avoidance* a janela do emissor é aumentada de 1 MSS sempre que é recebido um ACK.
 - (b) Durante a fase *congestion avoidance* a janela do emissor é aumentada de 1 MSS sempre que são recebidos todos os ACKs correspondentes aos segmentos da janela.
 - (c) O protocolo TCP evita a saturação da rede não variando o tamanho da janela de emissão.
 - (d) Durante a fase *slow start* a janela é aumentada de 1 MSS sempre que são recebidos todos os ACKs correspondentes aos segmentos da janela.
 - (e) Durante a fase *slow start* a janela é aumentada de 1 MSS sempre que é recebido um ACK.
 - (f) Um emissor TCP só usa segmentos de dimensão MSS. O receptor tem um *buffer* de recepção de dimensão MSS bytes. Logo, a janela do emissor nunca ultrapassa o valor de MSS bytes.
 - (g) Um emissor TCP só usa segmentos de dimensão MSS. O receptor tem um *buffer* de recepção de dimensão MSS bytes. O algoritmo Reno comportar-se neste caso da mesma forma que o algoritmo Tahoe.
 - (h) O algoritmo de controlo de saturação usado por um emissor TCP depende do algoritmo de controlo de saturação usado pelo receptor.

- (i) Admitindo por hipótese que uma conexão TCP apenas atravessa canais de muito alto débito que não perdem pacotes por erros de transmissão, a versão Reno do algoritmo de controlo da saturação é adequada mesmo que o RTT seja de 100 ms.
 - (j) O controlo da saturação tem uma solução completamente adequada através do algoritmo Reno sejam quais forem as condições de funcionamento da rede.
2. n computadores, cada um com uma única conexão TCP, partilham o mesmo *bottleneck link*. Não existe outro tráfego no mesmo canal. Qual é o resultado de um *browser* HTTP num dos n computadores pessoais abrir k conexões em paralelo para vários servidores distintos? Exprima o resultado em termos de n e k .
3. A Figura 8.26 representa a evolução da janela do emissor de uma conexão TCP ao longo do tempo. O eixo das ordenadas representa o tamanho da *congWnd* (a unidade é o MSS) e o das abcissas o tempo (a unidade é o RTT). O emissor está a usar o algoritmo de controlo da saturação Tahoe ou Reno? Justifique.

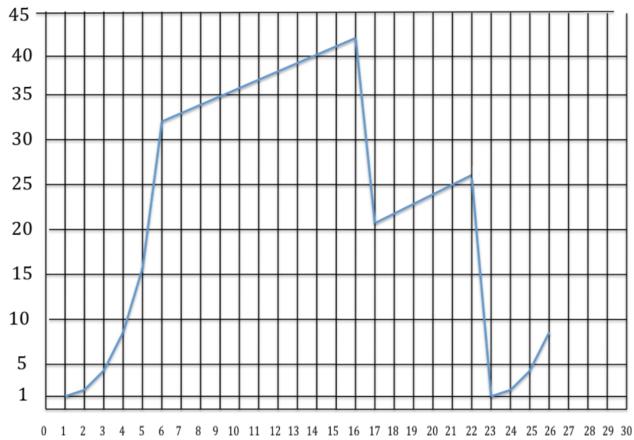


Figura 8.26: Evolução da dimensão da janela do emissor de uma conexão TCP

4. Um *browser* HTTP necessita de obter um objecto com 4.000 bytes de um servidor através de uma conexão TCP. Os canais que ligam o cliente e o servidor HTTP têm uma capacidade superior a 100 Mbps. O RTT entre os dois computadores é de 200 ms e a conexão TCP usa um MSS de 1250 bytes. A janela inicial de uma conexão é da dimensão de MSS bytes. Qual é aproximadamente o tempo necessário ao *browser* para obter o objecto sabendo que tem de abrir a conexão TCP e enviar uma primeira mensagem com o pedido?
5. Explique porque razão a capacidade do *bottleneck link* não é dividida equitativamente entre várias conexões TCP com RTTs distintos. Em particular, explique porque razão as conexões com menor RTT obtêm uma fração superior da capacidade do canal.
6. Qual o débito máximo de uma única conexão TCP que atravessa um conjunto de canais com o débito de 200 Mbps, apenas carregados a 10% com outro tráfego, com um RTT de extremo a extremo de 100 ms, mas em que ambas as partes não podem usar a opção “*receiver window scaling*”?
7. Dentro da rede dois comutadores estão interligados por dois canais *full-duplex* distintos, um com o tempo de propagação extrema de 50 ms e o outro

- com o tempo de 100 ms. Para dividir a carga, os segmentos de uma conexão TCP que passa por esses dois comutadores são enviados alternativamente, ora por um, ora pelo outro canal. Qual o resultado final deste melhoramento em termos do desempenho da conexão TCP?
8. Um canal com a capacidade de 100 Mbps é atravessado por 10 fluxos de pacotes e é o *bottleneck link* desses fluxos. Cinco dos fluxos correspondem a conexões TCP usadas para transferir ficheiros e podem, cada uma delas, ocupar todo o canal, isto é, potencialmente usar os 100 Mbps. Por hipótese, as conexões TCP têm RTTs semelhantes. Os outros 5 fluxos são de pacotes UDP pertencentes a 5 fluxos de vídeo independentes, cada um dos quais requer 2 Mbps de capacidade. Como é dividida aproximadamente a capacidade do canal pelos 10 fluxos?
 - (a) Todos os 10 fluxos ficam aproximadamente com 2 Mbps cada um;
 - (b) Todos os 10 fluxos ficam aproximadamente com 10 Mbps cada um;
 - (c) Os fluxos UDP ficam com aproximadamente 1 Mbps cada um e os fluxos TCP ficam com 10 Mbps cada um;
 - (d) Os fluxos UDP ficam com aproximadamente 2 Mbps cada um e os fluxos TCP ficam com 18 Mbps cada um;
 - (e) Os fluxos UDP ficam com aproximadamente 2 Mbps cada um e os fluxos TCP ficam com 10 Mbps cada um;
 - (f) Todos os 10 fluxos ficam aproximadamente com 100 Mbps cada um;
 - (g) Nenhuma destas opções.
 9. Um computador A está a transmitir um ficheiro de grande dimensão para um computador B através de uma conexão TCP. Ambos os computadores usam a versão Reno do algoritmo de controlo da saturação. Indique, de forma justificada, qual débito médio aproximado entre A e B nas condições indicadas a seguir.
 - (a) A e B estão ligados através de um canal directo *full-duplex*, dedicado exclusivamente à transferência, com uma velocidade de transmissão de V_t bps e um tempo de propagação desprezável.
 - (b) A e B estão ligados à Internet através de canais com capacidade de V_t bps. A janela de recepção máxima do *socket* de B tem J bits, com $J \gg MSS$ da conexão, o tempo de ida e volta entre A e B são RTT segundos, $V_t \gg J/RTT$ e verifica-se que nunca se perdem segmentos entre A e B.
 - (c) A e B estão ligados à Internet através de canais com capacidade de V_t bps. No interior da rede o *bottleneck link* só disponibiliza em média $V_{bottleneck}$ bps para a conexão, $V_t \gg V_{bottleneck}$, não existem limites devidos à dimensão da janela do receptor e os únicos segmentos que se perdem são retransmitidos por *Fast Retransmit*.
 10. Um programa criou uma conexão TCP entre os computadores A e B com o MSS de 10.000 bits. Essa conexão é a única actividade de rede em ambos os computadores. A funciona como emissor e B como receptor de um ficheiro de grande dimensão. Os canais que ligam A e B à rede têm o débito de 5 Mbps. O RTT entre A e B é de 100 ms. Não existem limites ao débito da conexão devidos a factores aplicacionais ou de dimensão dos *buffers* de A ou B.
 - (a) Qual o menor valor, em segmentos, da janela de emissão que maximiza a taxa de utilização do canal que liga A à rede?

- (b) Admita que o *bottleneck link* entre A e B apenas disponibiliza 1 Mbps para a conexão. Qual o valor médio aproximado, em MSSs, da janela de emissão no computador A, admitindo que os únicos segmentos que se perdem são retransmitidos por *Fast Retransmit*. Ambos os computadores usam a versão Reno do algoritmo de controlo da saturação.
11. O computador A abriu uma conexão TCP para o computador B com um MSS de 10.000 bits (1250 bytes aproximadamente). Os dois computadores estão ligados numa rede com o RTT de 98 ms. A rede tem uma grande capacidade interna e não perde pacotes. O computador A está a transmitir continuamente dados para B e a transferência não é refreada por factores ligados à aplicação. Os computadores A e B estão ligados à rede por canais que funcionam a 10 Mbps. O computador B tem uma janela máxima de recepção (*receiving window* máxima) de 100.000 bits (cerca de 12500 bytes). A janela inicial de uma conexão é da dimensão de MSS bytes.
- Qual a dimensão máxima atingida pela janela de emissão de A em múltiplos do MSS?
 - Quantos RTTs são necessários para que A atinja essa janela de emissão quando arranca na fase *slow start* do protocolo?
 - Quando a janela de emissão de A estabilizar, qual a velocidade média a que a transferência do ficheiro de A para B se processa?
12. Um cliente TCP necessita de obter 5 objectos de um servidor. Cada objecto tem 5 K Bytes. O cliente e o servidor estão separados por um tempo de trânsito de cerca de 50 ms pelo que o RTT entre ambos é de cerca de 100 ms. As conexões TCP não se encontram limitadas nem pela velocidade de transmissão dos canais que ligam os computadores à rede, nem pelo espaço livre nas janelas de recepção dos receptores, nem pela capacidade da rede. O MSS é de 1250 bytes e todas as conexões iniciam a gestão da janela de emissão pela fase *slow start*. A janela inicial de uma conexão é da dimensão de MSS bytes. Calcule o tempo necessário para obter os 5 objetos quando o cliente usa uma só conexão TCP e quando usa 5 conexões TCP abertas em paralelo. Justifique a sua resposta.

Capítulo 9

Transporte de dados multimédia

There are no facts, only interpretations.

– Autor: *Friedrich Nietzsche (1844-1900)*

A informação transmitida através de uma rede digital de computadores corresponde a sequências de símbolos, codificados como sequências de bits, que vão ser posteriormente processados pelos destinatários, *i.e.*, por máquinas ou por humanos (pelo menos por agora são poucos os outros animais que usam redes de comunicação). Quando o destinatário final são os humanos, a sequência de símbolos é transformada num conjunto de impulsos sensoriais de forma a poder ser absorvida e interpretada pelo destinatário. Por exemplo, os símbolos contidos numa mensagem são visualizados num ecrã de computador, ou são traduzidos por um altifalante num conjunto de sons, ou num conjunto de imagens animadas que reproduzem um filme. Os sons são percepcionados através do ouvido e as imagens através da vista.

No caso particular dos sons e das imagens, os humanos têm, em muitas situações, a capacidade de obter, grosso modo, a mesma quantidade de informação independentemente de os símbolos recebidos serem diferentes dos emitidos, pois quer o ouvido, quer a vista, têm capacidade de reconhecer o significado de sons e imagens degradadas face à sua versão incial. Apenas a qualidade e o conforto do destinatário ao processar essa informação é que vai variar. Por outro lado, no caso dos filmes, mas também no caso dos sons, a interpretação da informação recebida está dependente da que foi recebida anteriormente, pois essa informação só adquire o seu significado inserida num fluxo de informação. Por exemplo, uma imagem de um filme só se comprehende completamente quando inserida na sequência de imagens que a precederam.

Existem portanto situações em que a informação transmitida pela rede corresponde a fluxos de informação que codificam sequências de sons e imagens, ou de forma mais rigorosa, em que a informação transmitida é multi-formato (imagens, sons, legendas, *etc.*) e organizada em fluxos sincronizados (*e.g.*, os sons associados a imagens têm de estar sincronizados com estas). O termo popular para este tipo de informação é **informação multimédia em fluxos** ou simplesmente **informação multimédia** (*multimedia streaming* ou simplesmente *multimedia*). Os termos populares são um pouco enganadores porque qualquer ficheiro pode ser um ficheiro multimédia por ter palavras, frases, parágrafos e imagens, mas nós vamos utilizá-los também daqui para a frente.

Enquanto que a transmissão indiscriminada de informação tem de ser feita de forma fiável, pois não se sabe qual o seu tipo e a utilização que dela vai ser feita

pelo destinatário, a informação multimédia pode ser transmitida de forma não fiável. A solução não coloca problemas desde que o resultado seja de “qualidade aceitável” para o destinatário, sendo o adjetivo “aceitável” algo que depende do contexto e das exigências do utilizador.

Postas as coisas assim, parece que é suficiente enviar a informação multimédia através de um protocolo sem recuperação de erros, como o UDP por exemplo. Bom, as coisas não são assim tão simples, pois para que o resultado final seja de “qualidade aceitável” é necessário impor limites à taxa de erros mas também ao tempo de transferência de extremo a extremo e ao *jitter*. Por outro lado, é também possível tentar adaptar a resolução da informação transmitida à capacidade disponível na rede, o que dá uma nova perspectiva sobre o controlo de saturação.

No caso particular da transmissão de fluxos de informação multimédia podem ser usadas técnicas de transmissão de dados com perda de resolução, que tomam em consideração as características particulares da aplicação final a que os mesmos se destinam. Trata-se de um caso particular em que existe uma interligação entre o transporte e as aplicações. De alguma forma é mais um caso em que parece que a separação entre camadas não é absoluta e por essa razão este capítulo envolve aspectos que ultrapassam meramente a problemática do transporte de dados.

O capítulo começa exactamente por apresentar uma panorâmica breve sobre como são codificados de forma digital o som e a imagem, e de como essas informações são organizadas em fluxos sincronizados. A seguir apresentam-se os requisitos típicos das aplicações multimédia, e de que forma estes influenciam os protocolos e as soluções aplicacionais usadas pelas mesmas. Segue-se uma secção sobre as técnicas de correção de erros com e sem perda de resolução, usadas pelos canais lógicos extremo a extremo que suportam essas aplicações. Finalmente, é apresentado o protocolo *RTP – Real Time Transport Protocol*, que complementa o protocolo UDP com os mecanismos necessários para transmitir fluxos de informação multimédia em tempo real.

9.1 Codificação de informação multimédia

Codificação do som

Os objectos que vibram emitem vibrações, *i.e.*, ondas mecânicas, que se propagam pela atmosfera e que quando atingem as membranas e os ossos do ouvido dos animais são percepcionados como sons. O ouvido humano só consegue detectar sons compostos por ondas mecânicas cujas frequências de oscilação estejam entre 20 e 20.000 Hz¹ mas outros animais conseguem percepcionar frequências numa gama diferente. Aliás, a gama audível dos humanos varia de pessoa para pessoa e raramente é tão alargada como a indicada. A voz humana é composta de ondas mecânicas emitidas pelas cordas vocais numa gama de frequências grosso modo entre 300 e 8.000 Hz.

Um microfone é um dispositivo que transforma uma onda sonora num sinal eléctrico cuja variação é semelhante à da onda sonora, e um altifalante é um dispositivo que transforma um sinal eléctrico que segue o andamento de uma onda sonora numa onda sonora semelhante.

O sistema telefónico analógico tradicional realizava chamadas telefónicas à distância através da transmissão das ondas sonoras na forma de sinais eléctricos cuja variação era análoga à da voz dos interlocutores. Posteriormente, o sistema telefónico analógico deu lugar ao sistema telefónico digital que passou a transmitir o som digitalizado, pelo menos entre as centrais telefónicas, ver a Figura 9.1. Esta forma de codificação do

¹ A frequência dos fenómenos oscilatórios periódicos mede-se em Hertz (Hz) e 1 Hz corresponde à frequência de uma oscilação por segundo. Uma onda sinusoidal de 1 Hz corresponde a uma sinusoidal que completa o seu período em 1 segundo.

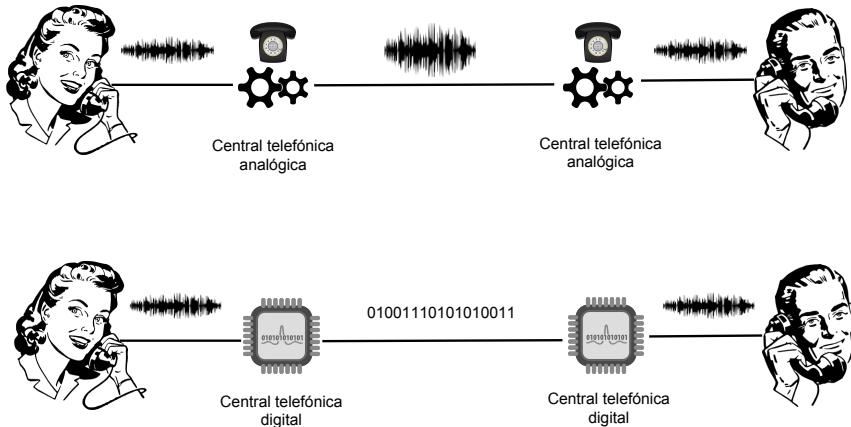


Figura 9.1: Transmissão do som à distância com base num sinal eléctrico analógico e com base no sinal digitalizado

som é também a adequada à sua transmissão por uma rede de pacotes de dados, visto que esta é capaz de transmitir sequências de valores.

A digitalização de uma onda (sonora, eléctrica, *etc.*) consiste em produzir uma sequência de valores numéricos que têm o mesmo andamento da onda. Os dispositivos que realizam esta transformação chamam-se **conversores analógico-digital (ADC - Analog to Digital Converter)**. A conversão inversa é feita por **conversores digital-analógico (DAC - Digital to Analog Converter)**.

Cada valor numérico diz-se uma **amostra (sample)**, e a sequência de valores corresponde a uma sequência de amostras tomadas a intervalos regulares. A periodicidade, ou seja o intervalo de tempo que separa cada amostra, fica caracterizada pela **frequência de amostragem (sampling rate)**.

A fidelidade entre a sequência de valores numéricos e a onda depende de dois factores: a frequência de amostragem e a resolução com que são expressos os valores numéricos. Quanto maior a frequência e quanto maior a resolução mais fidedigna (mais fiel) é a digitalização com respeito à onda original analógica e maior será também a quantidade de informação contida na sequência de valores numéricos.

Qual deve ser então a frequência de amostragem e a resolução que deve ser usada? A resposta a esta questão é dada pela teoria do processamento dos sinais digitais e depende do contexto de utilização. Segundo esta teoria, um sinal (sonoro, eléctrico, *etc.*) é equivalente à soma de um conjunto de sinais sinusoidais de diferentes frequências e amplitudes. Segundo a mesma teoria, se a frequência da componente de mais alta frequência for f , uma frequência de amostragem superior a $2f$ não acrescenta mais informação e portanto não aumenta a fidelidade.

Assim, qualquer som audível fica correctamente digitalizado do ponto de vista das frequências nele presentes desde que a frequência de amostragem seja superior a 40 KHz (40.000 vezes por segundo) pois o ouvido humano não percepciona sons cuja frequência seja superior a 20.000 Hz. A voz humana usa um conjunto de frequências

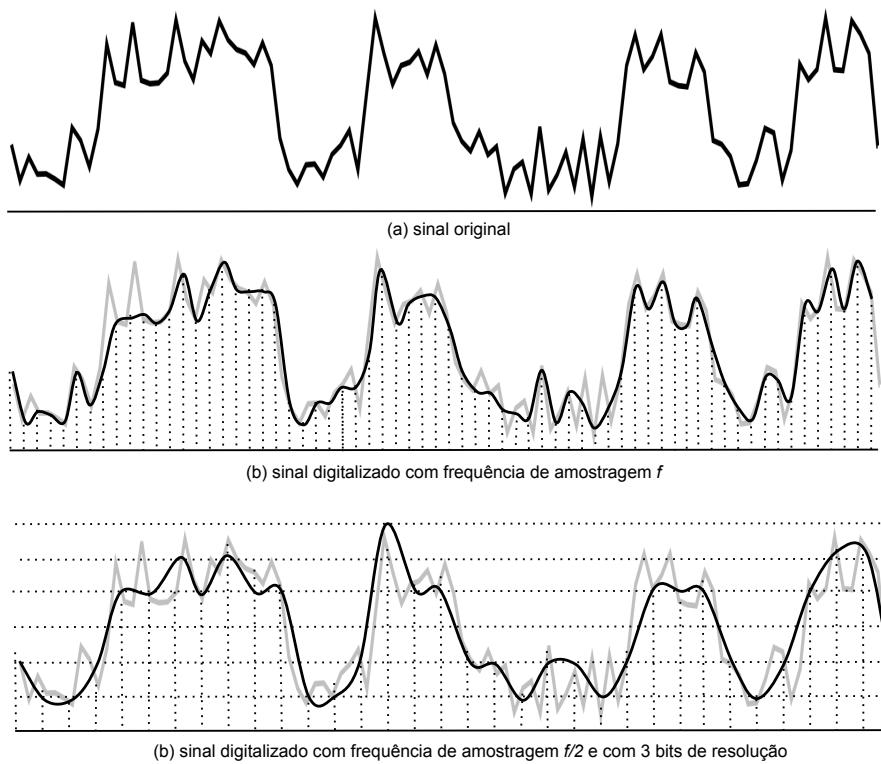


Figura 9.2: Impacto da frequência de amostragem e da resolução no sinal digitalizado

mais restritas que o conjunto dos sons audíveis e a experiência mostra que para efeitos de uma chamada telefónica, basta considerar frequências até 4 KHz, do que resulta uma frequência de amostragem de 8 KHz. Para a qualidade da voz característica dos telemóveis primitivos pode até usar-se uma frequência mais baixa, pois quanto menor for a frequência de amostragem, menor será a quantidade de informação produzida por unidade de tempo.

A resolução dos valores digitalizados está dependente do número de bits usados para representar cada valor da amostra. A deformação do sinal digitalizado em função da frequência de amostragem e da resolução é ilustrada pela figura 9.2.

Os valores normalizados inicialmente pelas indústrias de telecomunicações e da música foram respectivamente 8 KHz de frequência de amostragem e 8 bits de resolução para a voz no sistema telefónico digital, e 44,1 KHz de frequência de amostragem e 16 bits de resolução para o som nos CDs áudio. A opção no que diz respeito ao sistema telefónico introduz distorção em frequência na voz, pelo que a qualidade final do som pode não ser suficiente para transmitir uma ária de ópera pelo telefone convencional, visto que a forma de digitalização usada filtra as frequências superiores a 4 KHz.

Ambas as formas de digitalização introduzem distorção quantitativa pois o ouvido humano pode, no limite, distinguir intensidades distintas cuja gama de intensidade relativa pode ir de 1 a 1.000.000, enquanto que para a voz telefónica apenas são usados 256 níveis de intensidade distintos, e nos CDs musicais apenas 65.536 níveis. De qualquer forma essas resoluções têm sido consideradas suficientes para as conversas telefónicas e para a reprodução de música em alta fidelidade (excepto por alguns audiófilos mais exigentes).

Assim, a quantidade de informação por unidade de tempo que é produzida por uma chamada telefónica é de 64 Kbps (8 bits \times 8 KHz) e cada canal estéreo de música em alta fidelidade produz 705,6 Kbps (16 bits \times 44,1 KHz) ou 1,4112 Mbps para os dois canais. Estes valores correspondem aos débitos mínimos extremo a extremo necessários para transmitir essa informação por uma rede de computadores se não for usada compressão. Na realidade os débitos necessários são superiores, quer devido ao espaço ocupado pelos cabeçalhos, quer devido à necessidade de compensar o *jitter*.

Dados os débitos envolvidos, existem motivações significativas para usar compressão do som de forma a poupar capacidade na rede. Os algoritmos usados para comprimir o som são bastante diferentes dos algoritmos gerais de compressão, pois no caso do som os algoritmos de compressão usados podem conduzir a perda de informação. Um algoritmo de compressão que conserva a quantidade de informação após a descompressão chama-se um algoritmo sem perdas (*lossless compression*), enquanto que um algoritmo que conduz a perdas após a descompressão, chama-se naturalmente um algoritmo com perdas (*lossy compression*).

Os algoritmos de compressão do som mais eficientes são algoritmos com perdas, mas têm sido estudados intensivamente de modo a tornar essas perdas tão imperceptíveis quanto possível pelos humanos e por isso dizem-se psico-acústicos. Com efeito, o ouvido humano tem uma capacidade muito elevada de reconhecer frequências e intensidades distintas dos sons, mas não ao mesmo tempo. Assim, a presença num som de certas frequências audíveis cancela a hipótese de o ouvinte distinguir outras que lhe estejam próximas, e a capacidade de distinguir certas frequências do som depende também da intensidade de outros sons que lhes estejam próximos e presentes em simultâneo.

Os algoritmos de compressão usados exploram estas propriedades do ouvido humano para obterem factores de compressão muito significativos. Por exemplo, a voz telefónica pode ser transmitida com qualidade aceitável a 64 Kbps usando a forma de digitalização apresentada acima (8 KHz de frequência de amostragem e 8 bits de resolução). Esta forma de codificação digital costuma designar-se por PCM (*pulse code modulation*) e foi normalizada pela ITU através da norma G.711 em 1972. A norma G.711 produz uma amostra de som digitalizado de 125 em 125 microssegundos. No entanto, é também possível transmitir voz com um nível de qualidade aceitável para conversas telefónicas usando a norma G.729, que usa um algoritmo de compressão com perda de resolução mas que requer apenas o débito de 8 Kbps.

A música é transmitida usando algoritmos de compressão que se têm tornado populares entre o grande público, nomeadamente o MP3 (*MPEG-1, 2, 2.5 audio layer III*) e o AAC (*MPEG-2, 2.5, 4 Advanced Audio Coding*) e muitos outros, alguns normalizados e públicos e outros proprietários.

Por exemplo, os formatos MP3 e AAC conseguem qualidade de som variável e podem usar uma gama também variável de débito por canal: de 8 a 320 Kbps para o MP3 e de 8 a 529 Kbps para o AAC. Nos débitos maiores a compressão pode ser sem perdas. Em geral é possível obter uma qualidade aceitável com 96 Kbps por canal, mas para obter uma qualidade já próxima do CD original, dependendo do formato, é necessário usar resoluções maiores.

Um outro aspecto importante a reter é que estes algoritmos de compressão necessitam de actuar sobre quantidades mínimas de som para conseguir realizar a compressão e por isso introduzem atrasos suplementares. Assim, o MP3 actua sobre um mínimo de 100 ms de som digitalizado e o AAC em certas resoluções ainda pode necessitar de períodos de tempo maiores. Estes atrasos são independentes da capacidade de processamento e dos atrasos de extremo a extremo da rede, somando-se aos mesmos.

O som digitalizado consiste numa sequência de valores numéricos que representam a intensidade da onda sonora amostrada (medida) a intervalos regulares. A resolução da digitalização corresponde ao número de bits usados para representar os valores medidos.

A frequência de amostragem determina as frequências presentes na representação digital. É normal usar 8 bits de resolução e 8 KHz de frequência de amostragem para a digitalização da voz nos sistemas telefónicos, ou 16 bits / 44,1 KHz para a música, ou ainda maior resolução e frequência de amostragem quando existem requisitos de muito alta fidelidade.

Devido às características percepcionais do sentido da audição é possível aplicar algoritmos de compressão do sinal sonoro digitalizado. Estes algoritmos podem implicar perda de informação (são *lossy*) mas não perda de significado nos sistemas telefónicos, nem diferenças demasiado perceptíveis da qualidade musical. Os ganhos assim obtidos podem reduzir o débito do fluxo digital correspondente ao som digitalizado até uma ordem de grandeza, sem um impacto significativo na utilidade final da informação sonora transmitida.

Codificação das imagens

As imagens digitais, mesmo aquelas que parecem contínuas, são apresentadas pelos ecrãs digitais como um conjunto de pixels, *i.e.*, um conjunto de pontos discretos, ou unidades de informação elementares sobre a cor e intensidade da imagem². A vista humana é capaz de reconhecer a presença dos diferentes pixels em imagens digitais de baixa qualidade, ou seja, com pouca densidade de pixels por unidade de superfície. A partir do momento em que a densidade de pixels por unidade de superfície vai aumentando, a vista humana vai sendo incapaz de distinguir os pixels uns dos outros e a qualidade da imagem vai aumentando, passando o utilizador a vê-la como um contínuo.

Para representar imagens a preto e branco basta associar a cada pixel a intensidade da luz branca. Dependendo da resolução de cada pixel, pode usar-se 1 bit por pixel (o pixel está a preto ou a branco) ou vários bits onde se representam várias intensidades de luz branca. Já para representar imagens a cores é necessário associar também uma cor a cada pixel. Um forma comum de obter as diferentes gamas de cores mistura nas quantidades adequadas as cores vermelho, verde e azul (forma de codificação designada como *RGB – Red Green Blue*), ver a Figura 9.3.

Assim, numa imagem a cores, a cada pixel está associada uma quantidade de informação que codifica a intensidade de cada uma destas três cores. Em geral usam-se dois bytes (*High Color*) ou três bytes (*True Color*) para representar cada pixel. Quando se usam 2 bytes (16 bits) são usados 5 bits para o vermelho, 5 bits para o azul e 6 bits para o verde pois a vista é mais sensível ao verde. No caso da representação com 3 bytes usa-se 1 byte por cor para representar a intensidade da presença da mesma. Esta forma de codificação é conhecida por RGB888. É também frequente encontrar a forma de representação ARGB888 que acrescenta 1 byte à representação RGB888 para codificar o valor da transparência (*alpha value*), resultando em 4 bytes por pixel (32 bits).

As imagens de muito alta qualidade nos ecrãs de computador actuais são conseguidas com a codificação ARGB888 e uma densidade de pixels superior a 8.000 pixels / cm². Uma imagem de televisão 4K contém 4.096 × 2160 pixels (cerca de 8.000.000 milhões de pixels no ecrã) codificados em RGB888.

²A palavra pixel é um neologismo derivado do mesmo termo em inglês, que é formado a partir da união de dois elementos: “pix” (de *picture*) e “el” (de *element*).

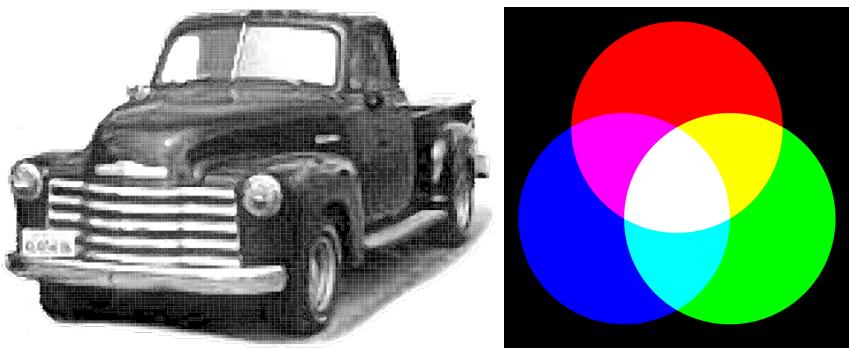


Figura 9.3: Representação de uma imagem usando pixels e o sistema de cores RGB

Um filme é apresentado fazendo desfilar a um ritmo adequado um conjunto de imagens em sucessão. O termo em português é uma sucessão de fotogramas. O termo em inglês para o número de fotogramas por unidade de tempo é *frame rate* (cada imagem é designada por *image frame* ou simplesmente *frame*). A vista humana é incapaz de distinguir umas imagens das outras a partir de um certo *frame rate*, nomeadamente cerca de 30 imagens por segundo. Actualmente a indústria cinematográfica usa 24 ou 48 imagens por segundo e a televisão usa 25 ou 50 (Europa) e 30 ou 60 (EUA). A tendência é para os valores mais altos serem os adoptados pois permitem melhor qualidade.

Daqui resulta que a quantidade de informação necessária para transmitir de forma digital um filme, sem compressão, mas com qualidade, é muitíssimo elevada. Por exemplo, com 50 imagens por segundo de 8.000.000 pixels cada, usando 3 bytes para representar cada pixel, a quantidade de informação por unidade de tempo seria: $50 \times 8 \cdot 10^6 \times 24 \text{ bps} = 9,6 \times 10^9 \text{ bps}$ ou 9,6 Gbps. Mesmo usando uma resolução inferior, a quantidade de informação atinge muito facilmente a gama das centenas de mega (ou mesmo dos giga) bits por segundo.

Felizmente que, tal como o som, também as imagens podem ser comprimidas de forma muito significativa, com perda de resolução mas eventualmente sem que o observador se aperceba disso. Assim, as imagens fixas podem ser comprimidas usando várias técnicas relativamente simples de descrever (mas não necessariamente de implementar).

A primeira é alterando a resolução e fundindo um conjunto de pixels vizinhos num só e suprimindo detalhes que a vista não distingue normalmente (sem lupa!). Na Figura 9.4 a técnica é ilustrada mas com perda de resolução perceptível ao utilizador. Frequentemente esta técnica de compressão é imperceptível à vista humana desde que a dimensão da imagem seja a adequada. No entanto, caso a imagem comprimida veja a sua dimensão ampliada, a densidade de pixels por unidade de superfície altera-se e as imperfeições aparecem, pois a ampliação é feita por interpolação. É por esta razão que as fotografias de qualidade média não resistem a serem impressas como *posters*. Uma segunda técnica consiste em fundir pixels contíguos com cores e intensidade semelhantes, como por exemplo a zona de uma imagem que contém um fundo de cor constante, ou seja explorando a redundância espacial.

O formato normalizado de representação de imagens fotográficas mais comum é o JPEG (Joint Photographic Experts Group) que especifica um formato comprimido de representação de imagens fixas. A norma recorre a um algoritmo de compressão que



Figura 9.4: Compressão de uma imagem alterando a resolução (logo do IDE Eclipse)

incorpora várias técnicas de compressão, com e sem perda de resolução.

É possível representar um filme mostrando em sucessão várias imagens independentes representadas no formato JPEG. Como é evidente, é possível fazer muito melhor que isso pois na grande maioria dos filmes muitos fotogramas (*image frames*) estão correlacionados e só apresentam pequenas modificações em relação aos precedentes, ver um exemplo na Figura 9.5.

O formato mais popular para representação com compressão de filmes é o formato MPEG (Motion Picture Experts Group). A norma MPEG tem vários formatos possíveis que contemplam para além das imagens, um ou mais canais de audio e canais com as legendas, todos sincronizados com as imagens.



Figura 9.5: Compressão de um filme explorando a redundância entre fotogramas – o fundo é comum aos diferentes *frames*, só o corredor difere de fotograma para fotograma

A variante MPEG-1 corresponde à qualidade da televisão analógica tradicional e produz um débito médio entre 1 e 3 Mbps, dependendo da qualidade seleccionada. Os formatos de vídeo do MPEG-1 produzem um fluxo de tipo CBR (*Constant Bit Rate*) pois a forma de compressão usada não depende das características da imagem e do movimento. O formato MPEG-2 corresponde à qualidade dos DVDs e, dado o tipo de compressão usada, conduz a um fluxo de tipo VBR (*Variable Bit Rate*) dependendo do movimento e das características das cenas com um débito médio de 4 Mbps. A variante MPEG-4 permite vários formatos e resoluções entre as quais a resolução da televisão de alta definição e a dos discos Blu-ray e o débito médio depende do formato usado. Na difusão satélite digital são usados os formatos MPEG-2 e -4.

Se todos os fotogramas MPEG representassem apenas as modificações com respeito

ao precedente, não era possível ver o filme senão começando pelo início e indo até ao fim, e não seria possível avançar nem recuar a imagem por exemplo. Por outro lado, também não seria possível perder um único *frame* caso estes fossem transmitidos pela rede. Finalmente, verifica-se também que a maioria dos filmes mudam de cena com um intervalo de alguns segundos (3 a 10 segundos por exemplo).

Por estas diferentes razões o formato MPEG especifica diferentes tipos de *frames*: os I-*frames* que contém imagens completas e capazes de serem processadas individual e isoladamente, e os P-*frames* e os B-*frames* que são *frames* que apenas contém diferenças relativamente aos precedentes e aos vizinhos (do passado ou do futuro). A descodificação dos I-*frames* é semelhante à descodificação de imagens JPEG, mas a descodificação dos outros tipos de *frames* exige a utilização de um *buffer* com vários *frames*. Todas estas técnicas mostram que a descodificação MPEG requer um *playout delay* suplementar.

Uma imagem digitalizada contém uma sequência de valores que codificam a cor e a intensidade de um conjunto de pixels, *i.e.*, de unidades elementares da imagem. O número de bits usados para codificar o valor de cada pixel e a densidade de pixels da imagem condicionam a sua qualidade. Um filme é uma sequência de imagens justapostas (*image frames*), com frequências entre 20 e 60 imagens por segundo.

O débito de informação por unidade de tempo requerido para a transmissão de imagens fixas e de filmes é muito elevado se não se utilizarem algoritmos de compressão. Dadas as características de redundância das imagens e dos filmes, assim como as características da vista humana, é possível utilizar algoritmos de compressão de imagem e de filmes que reduzem esse débito até 3 ordens de grandeza através de perda de informação, mas muitas vezes com resultados imperceptíveis para a vista.

Noção de codec

Quando informação multimédia é transmitida através de uma rede, é necessário que os emissores e os receptores estejam de acordo sobre o formato e a resolução usados para interpretar o conteúdo recebido.

Um codec é um dispositivo software, hardware ou misto, capaz de codificar e descodificar (COder / DECoder) informação multimédia de acordo com um formato específico. Emissores e receptores do mesmo fluxo multimédia têm de usar codecs equivalentes.

Para melhorar a modularidade dos sistemas hardware / software usados para transmissão e interpretação de informação multimédia (emissores e receptores satélite, emissores e receptores de televisão digital terrestre, aparelhos de leitura de DVD ou outros discos, software multimédia, *etc.*) estes podem usar ou ser estendidos com diferentes codecs. Estas operações consistem simplesmente em permitir que os diferentes dispositivos usem diferentes normas e formatos de informação multimédia. Assim, muitas vezes confunde-se o dispositivo codec com o próprio formato.

Os codecs que utilizam formatos normalizados obedecem a normas bem conhecidas e cuja especificação é pública. Os codecs que utilizam formatos não normalizados podem usar formatos proprietários e secretos (como por exemplo os codecs usados pela aplicação Skype) ou formatos públicos não normalizados.

Alguns dos codecs mais populares são da família MPEG, estão implementados em hardware e estão integrados em todas as televisões digitais modernas, DVDs, ou ainda na maioria dos softwares multimédia actuais.

A informação multimédia tem características particulares que permitem que a “mesma” informação seja transmitida com resoluções distintas, ou mesmo com erros, sem que a sua utilidade seja afectada na mesma proporção.

Esta característica, a par de outros requisitos das aplicações que usam informação multimédia, tem um impacto significativo sobre as técnicas de transmissão de informação multimédia usadas sobre uma rede de pacotes.

No resto do capítulo utilizaremos informalmente o termo codec para designar quer o dispositivo, quer a norma e os formatos que este implementa.

9.2 Transporte sobre TCP e sobre UDP

Existem diversos tipos de aplicações multimédia que usam redes de computadores. No primeiro grupo podemos incluir as aplicações que reproduzem ou visualizam³ de forma diferida informação multimédia a partir de um ficheiro. Estas aplicações não impõem nenhum requisito novo ao transporte de informação pois os ficheiros multimédia podem ser transmitidos usando o protocolo TCP por exemplo. Os exemplos mais comuns deste tipo de aplicações são aquelas que permitem visualizar filmes ou tocar música a partir de ficheiros registados no disco local dos computadores.

A novidade é introduzida pelas aplicações que visualizam informação multimédia em tempo real. Os termos usados na língua inglesa para designar este tipo de transmissão é *live streaming* e as aplicações de vizualização chamam-se *stream players*. Estas aplicações permitem que um cliente de visualização apresente ao utilizador a informação que está a ser emitida por um emissor multimédia “no momento”.

As aplicações de tempo real dividem-se nas unidireccionais e nas interactivas, ver a Figura 9.6. No caso das primeiras, existe um só emissor e um ou mais receptores passivos da informação emitida. Os exemplos clássicos são os canais de televisão ou de rádio difundidos por redes de pacotes IP (IPTV ou *live streaming*) e os filmes ou a música difundidos pelo mesmo meio (*on demand streaming*). Os exemplos mais comuns de aplicações multimédia interactivas são as chamadas telefónicas sobre IP (VoIP), os jogos multimédia e as vídeo-chamadas, todas suportadas em redes de pacotes IP.

A informação multimédia transmitida e recebida em tempo real tem requisitos temporais e de débito relacionados com a resolução e a frequência de amostragem da informação sonora e visual, mas também tem requisitos temporais especiais devido à necessidade de sincronização dos interlocutores. Assim, estas aplicações têm exigências específicas de qualidade de serviço da rede, em particular no que diz respeito ao débito e tempo de trânsito extremo a extremo e ao *jitter*.

Débito extremo a extremo mínimo

O codec usado pelo emissor requer um dado débito mínimo extremo a extremo. Se a rede não o tiver disponível, é necessário “mudar de rede”, ou usar uma resolução

³ Os termos normalmente usados na língua inglesa para designar estas aplicações são *players* e *viewers* e as mesmas permitem a a reprodução de som e a visualização de imagens com sonorização. Em português usaremos o termo “visualizador” como sinónimo de um programa ou dispositivo hardware / software que descodifica informação sonora ou visual e a reproduz, codificador àquele que apenas codifica, e emissor àquele que a transmite.

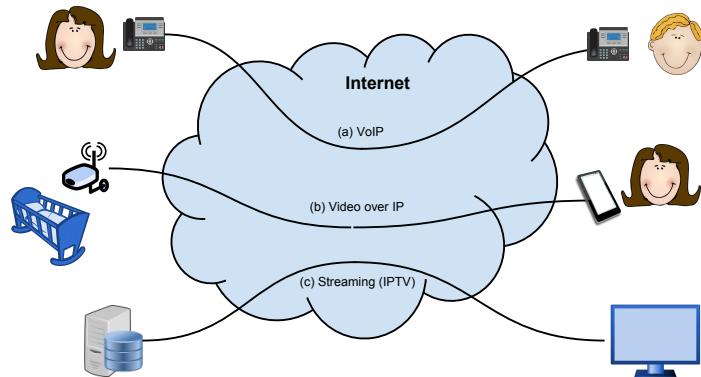


Figura 9.6: Tipos de aplicações multimédia em tempo real

mais baixa na informação multimédia transmitida. No limite, se o débito for variando com o tempo, a aplicação deveria adaptar-se dinamicamente à capacidade disponível e ajustar a resolução e a amostragem da informação multimédia transmitida.

Tempo de trânsito extremo a extremo máximo

Dado que a transmissão é em tempo real, a informação tem de ser visualizada até um limite de tempo máximo depois de ter sido produzida e emitida. Como todas as aplicações em tempo real ou ao vivo (*live*) têm de recorrer a um *playback buffer* para compensar o *jitter*, ver a Secção 3.4, ao tempo de trânsito extremo a extremo somam-se os atrasos com origem neste mecanismo de compensação das variações do tempo de trânsito, mas também os atrasos introduzidos pelas necessidades específicas do algoritmo de compressão e da transmissão em blocos de dados digitalizados característicos de cada codec. Assim, o atraso extremo a extremo visível pelo utilizador pode ser significativo.

O valor máximo do tempo de trânsito final aceitável é normalmente limitado, em particular numa aplicação interactiva, como uma aplicação de chamada telefónica (VoIP) podendo ser um pouco mais alargado em aplicações unidireccionais, como as aplicações de visualização de canais de televisão (IPTV), música ou de visualização de filmes.

No entanto, mesmo nesses casos, a paciência do utilizador tem limites pois este não está disposto a esperar vários minutos para começar a visionar um canal de televisão para o qual acabou de mudar. Mesmo com as aplicações não interactivas, o utilizador tolera no máximo alguns segundos até a informação começar (ou recomeçar) a ser visualizada e considera má qualidade de serviço esperar tempos superiores.

Jitter limitado

A rede, como sabemos, introduz *jitter*. Adicionalmente, os protocolos de transporte como o TCP podem acentuar significativamente o *jitter* extremo a extremo sempre que ocorrem erros, ou o ritmo de transmissão se altera devido aos mecanismos de controlo da saturação. Uma forma de combater estes dois factores é alongar o valor do *playout delay* a utilizar mas, como já referimos, esse alongamento tem limites.

Os estudos psicológicos demonstram que se um utilizador de uma aplicação interactiva de VoIP experimentar um intervalo de propagação maior que 200 ms, os utilizadores finais começam a aperceber-se desse atraso, e o mesmo torna-se incomodativo a partir de 400 ms. O fenómeno é perceptível nas conversas observadas nos noticiários envolvendo reportagens com diálogo interativo entre

o locutor local e um correspondente remoto noutro país distante através de um canal suportado em satélite, com atrasos claramente perceptíveis. O ritmo com que é possível fazer perguntas e obter respostas fica condicionado ao tempo de propagação extremo a extremo e um utilizador pouco experiente não consegue manter o diálogo.

Em conclusão, devido a fenómenos psicológicos dos utilizadores, as aplicações multimédia interactivas, para serem utilizadas de forma confortável, têm requisitos que implicam limites estreitos e rígidos ao *jitter*.

Todas as aplicações multimédia em tempo real têm requisitos de qualidade de serviço específicos que estão dependentes do tipo da aplicação, das expectativas dos utilizadores e do seu contexto de utilização. Esses requisitos dizem respeito ao débito, ao tempo de trânsito e ao *jitter*.

Os primeiros estão ligados às exigências dos codecs usados, os quais podem ser adaptados à capacidade disponível. Os requisitos mais difíceis de satisfazer são os ligados com o tempo de trânsito e o *jitter* e são particularmente importantes para as aplicações interactivas.

A questão que se coloca a seguir é saber que opções existem para transportar pela rede a informação multimédia destas aplicações.

Transmissão multimédia sobre TCP

O protocolo TCP é muito simples de utilizar e adapta-se à transmissão de todos os tipos de dados. Adicionalmente, devido a um fenómeno que será discutido na secção 10.4 e conhecido por “ossificação da Internet”, é o único protocolo de transporte que pode ser usado em certos contextos, pois os outros protocolos são bloqueados por equipamentos de segurança.

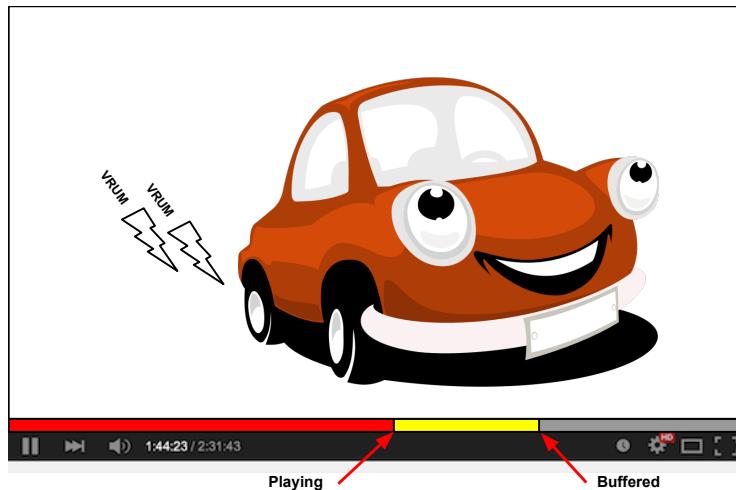


Figura 9.7: Utilização de um grande *playback buffer* em aplicações de *streaming a pedido*

Infelizmente, devido ao controlo de erros e ao controlo da saturação, o TCP tem um débito variável e, pior ainda, essas variações vêm acompanhadas de *jitter* suplementar, sobretudo em caso de erros. Para compensar esses problemas pode usar-se um *playback buffer* de grande dimensão, capaz de acomodar dezenas de segundos de visualização

mesmo sem chegarem novos dados, e um *playout delay* capaz de absorver variações pronunciadas da capacidade e do tempo de trânsito. É a solução mais comum nas aplicações de *streaming* de filmes ou de música como é ilustrado na Figura 9.7. No caso da música, como o débito necessário é mais baixo, o *playback buffer* pode ser mais pequeno.

Utilizar um *playback buffer* de grande dimensão conduziria a um compasso de espera significativo, quer no início da visualização, quer quando o utilizador resolve mudar a cena que pretende visualizar ou a música que pretende ouvir.

Mais recentemente começou-se a usar uma solução mais sofisticada, baseada em servidores contendo os filmes codificados em segmentos de alguns segundos de duração cada (*e.g.*, segmentos de 2 segundos), cada um dos quais disponível em diferentes resoluções. Compete ao visualizador fazer um pedido ao servidor a solicitar o envio de um segmento, indicando o seu índice (posição no filme) e a resolução escolhida.

Assim, o visualizador quando solicita o envio dos primeiros segundos do filme, pode optar por pedir segmentos codificados numa resolução mais baixa, e portanto requerendo um débito menor. Isso permite-lhe obter rapidamente segmentos do filme que pode ir mostrando ao utilizador. Se o utilizador mantiver uma visualização sequencial, o visualizador pode solicitar os segmentos seguintes com resolução mais elevada. Se a transição entre segmentos estiver alinhada com os *I-frames*, o utilizador acaba por não se aperceber das alterações de resolução.

Este mecanismo, para além de permitir um início de visualização mais rápido, permite também ao visualizador adaptar a resolução com que solicita os segmentos à capacidade da rede. Se os segmentos chegam mais depressa que a duração do filme neles codificado, o visualizador pode aumentar a resolução do próximo segmento a pedir, senão tem de diminuir a resolução dos segmentos seguintes. Trata-se de um controlo dinâmico e adaptativo da resolução, *i.e.*, da quantidade de informação pedida, à capacidade disponível na rede.

Esta solução torna-se ainda mais realista se os filmes forem replicados por diferentes servidores e os mecanismos que serão descritos no Capítulo 13 forem usados para dirigir os clientes para a réplica mais próxima. É a solução adoptada por serviços como o YouTube ou o Netflix, que usam conteúdos replicados em vários centros de dados espalhados por diferentes países. Como nesta situação os clientes estão relativamente mais próximos dos conteúdos a visualizar, é possível obter uma solução de qualidade razoável, baseada na transferência por TCP e na utilização de resolução variável. Uma solução semelhante também pode ser adoptada para a transmissão de canais de televisão em tempo real e para a transmissão de música. Neste último caso, na maioria das situações, as redes bem dimensionadas suportam a resolução mais alta.

É também possível utilizar TCP para transferir informação multimédia em aplicações interactivas. No entanto, neste caso, é habitual utilizar também resoluções variáveis mas é frequente o utilizador obter baixa qualidade final caso a capacidade da rede não seja suficiente, ou durante momentos de perda significativa de pacotes. É o exemplo típico de aplicações de vídeo-conferência interactivas gratuitas (*e.g.*, Skype, Google Hangouts, *etc.*).

O protocolo TCP é simples, omnipresente, compensa os erros e adapta-se à capacidade disponível na rede. Por estas razões constitui também um suporte adequado à transferência de informação multimédia se o débito médio extremo a extremo for o adequado ao codec usado.

Infelizmente, as sua virtudes têm como repercussão a amplificação do *jitter* extremo a extremo, em particular na presença de erros e também devido à actuação dos mecanismos de controlo da saturação. Este defeito pode ser combatido usando um *playout delay* mais longo.

Apesar de essa solução ser compatível com aplicações unidireccionais, a mesma introduz atrasos suplementares e incómodos para os utilizadores finais, particularmente em aplicações multimédia interactivas, pelo que muitas vezes as aplicações recorrem à redução da resolução, para diminuírem os requisitos de capacidade de rede e minorarem os efeitos negativos sobre o *jitter*.

A introdução de mecanismos de adaptação dinâmica da resolução dos fluxos multimédia, à qualidade da rede, é uma solução comum que tenta minorar os efeitos do controlo de saturação do TCP. Quando as aplicações são interactivas, a necessidade de encurtar tanto quanto possível o *playout delay*, implica a redução da qualidade dos fluxos multimédia através do uso de codecs de pior qualidade.

No entanto, existem diversas situações que recomendam o uso do protocolo UDP. A primeira é quando se pretende enviar canais de televisão em simultâneo para uma grande quantidade de receptores e optimizar a utilização da capacidade da rede através da utilização de IP multicasting, ver a Secção 5.1. A segunda é quando se deseja satisfazer requisitos de interactividade estritos para suporte de chamadas telefónicas.

Transmissão multimédia sobre UDP

O TCP só suporta canais ponto-a-ponto. Quando se pretende servir muitos clientes simultaneamente, a solução TCP com um único servidor não escala. Nesse caso é possível recorrer a muitos servidores em simultâneo, solução só acessível caso exista um modelo de negócio viável e capacidade de investimento significativa. Também é possível recorrer a um modelo de distribuição do conteúdo através de uma solução P2P (ver a Secção 1.4), mas nesse caso é mais difícil manter um nível de qualidade adequado.

Quando os conteúdos são difundidos a pedido, estas são provavelmente as únicas soluções possíveis pois cada cliente visualiza um conteúdo diferente em cada momento. No entanto, quando se trata de canais de televisão, muitos utilizadores pretendem ver simultaneamente o mesmo conteúdo e uma solução de difusão de um para vários parece ser a mais adequada e eficiente. Esta solução pode ser realizada usando IP Multicasting, mas este só é acessível às aplicações recorrendo ao transporte UDP.

Para além de suportar *multicasting* ao nível transporte, o UDP tem a vantagem de não amplificar o *jitter* da rede, mas tem o defeito de não compensar os erros, pelo que quando se perdem datagramas UDP, o receptor vai receber informação multimédia com lacunas. Se a origem das perdas de datagramas UDP for só os erros nos canais, a sua taxa nos canais de fibra actuais é relativamente baixa, mas se a perda de pacotes se dever também a canais mal dimensionados, a taxa de erros é geralmente muito elevada, independentemente da qualidade dos canais.

Na secção seguinte serão analisadas várias técnicas de compensação dos erros em canais que transportam informação multimédia. No entanto, essas técnicas só se podem aplicar quando as perdas não são significativas nem sucedem em cascata. Por isso, não é muito realista usar UDP para transferir informação multimédia em redes mal dimensionadas ou com taxas de erro significativas. Por outro lado, devido aos modelos de negócio dominantes na Internet, a utilização de *multicasting* não é realista quando os pacotes difundidos têm de atravessar redes controladas por mais do que um operador. Assim, a utilização de UDP e de IP Multicasting para transferir a informação multimédia de canais de televisão está restringida ao contexto de um operador e respectivos clientes ligados directamente por canais de qualidade.

Esta solução é hoje em dia popular nos serviços de difusão de canais de televisão sobre IP (IPTV), em substituição das soluções tradicionais da televisão por cabo. O operador com serviço IPTV reserva capacidade no seu *backbone* para transmitir para

todos os comutadores regionais e urbanos fluxos de pacotes UDP por *multicasting* contendo todos os canais de televisão acessíveis aos clientes. Cada cliente, através de um canal rede dedicado, apenas subscreve os canais de televisão que pretende visualizar no momento e só recebe os fluxos de pacotes UDP correspondentes a esses canais.

Com canais rede de qualidade, uma rede bem dimensionada e reserva de capacidade para os fluxo de pacote UDP em *multicasting* correspondentes aos canais de televisão, a qualidade final é bastante boa e é possível usar um *playout delay* bastante curto. A utilização de *multicasting* favorece a escalabilidade do serviço, sendo possível servir centenas de milhar de clientes a partir de poucos emissores.

O facto de o protocolo UDP não dilatar o *jitter* é muito importante em aplicações interactivas e por isso o transporte de informação multimédia via UDP é também frequentemente usado nas aplicações de telefone e de vídeo conferência. Nestes casos também se utilizam técnicas de compensação dos erros tendo em consideração a natureza multimédia da informação trocada pelos interlocutores.

A transmissão de conteúdos multimédia sobre UDP tem a vantagem de ser compatível com a utilização de um *playout delay* curto, pois o protocolo não introduz dilatações suplementares do *jitter*. Adicionalmente, o protocolo é compatível com a difusão (*multicasting*), pelo que aumenta a escalabilidade do serviço quando existem muitos clientes a subscrever simultaneamente o mesmo conteúdo.

No entanto, como o UDP não compensa os erros da rede, o conteúdo pode apresentar lacunas. Quando a taxa de erros se deve apenas a erros nos canais e é pouco significativa, existem técnicas especiais de compensação dos erros de omissão de informação multimédia que são usadas para minorar o seu impacto sobre a qualidade final percepcionada pelos utilizadores.

9.3 Tratamento de erros em fluxos multimédia

Quando usamos um protocolo como o UDP para transferir informação multimédia, alguns datagramas UDP podem não chegar ao destino. Nesta secção vamos analisar algumas técnicas de compensação desses erros, algumas das quais exploram facetas específicas do tipo de informação transmitida. Com efeito, como introduzimos no início do capítulo, mesmo quando reduzimos a quantidade de informação multimédia transmitida, a utilidade da mesma para os utilizadores finais pode até continuar a ser idêntica (*e.g.*, numa conversa telefónica, apesar de ser mais difícil de perceber o som com erros, o destinatário pode continuar a perceber a frase do interlocutor).

A informação multimédia é sempre organizada em *frames*. Para ilustrar as técnicas usadas para compensar os erros vamos admitir, por hipótese, e sem perda de generalidade, que cada *frame* corresponde a um datagrama UDP, contido num pacote IP, e que o mesmo contém som digitalizado correspondente a um certo período de tempo.

Usando o codec G.711, que usa 8 bits para codificar cada amostra, e uma frequência de amostragem de 8 KHz, a cada milissegundo de som correspondem 8 bytes de informação. Para encher um *payload* com 1024 bytes de informação, seria necessário esperar 125 ms. Como a estes 125 ms é necessário juntar o tempo de propagação extremo a extremo do pacote de dados, e ainda um *playout delay* para acomodar o *jitter*, é fácil ultrapassar o limite a partir do qual os utilizadores se aperceberiam do atraso da transmissão numa conversa telefónica (atrasos > 150 ms). Por esta razão,

o período contido em cada pacote é mais pequeno. Com 20 ms cada pacote conterá apenas 160 bytes de som digitalizado.

Assim, durante a transmissão a perda de um pacote inutilizará 20 ms de som. Para evitar que o utilizador se aperceba desta falta, é necessário encontrar formas de a compensar.

FEC – Forward Error Correction

Como a informação é transmitida por pacotes e existem mecanismos de detecção de erros que rejeitam pacotes com erros, o tipo de erros que importa corrigir são designados erros de omissão. Assumindo que os pacotes têm um número de sequência, o receptor pode detectar qual o pacote em falta.

Existem códigos de correção de erros que consistem em informação redundante, acrescentada à informação transmitida, e que permite ao receptor detectar e corrigir eventuais erros que tiveram lugar durante a transmissão. Este tipo de técnicas chamam-se FEC (*Forward Error Correction*), por as mesmas se basearem em informação transmitida para a frente, sem recurso a informação retransmitida (voltando atrás). As mesmas podem também ser usadas a nível aplicacional para resolver o nosso problema pois existe uma classe de códigos de correção de erros de omissão, designados por *erasure codes*, que foram desenvolvidos para lidar com esses erros.

Uma das formas mais simples deste tipo de códigos consiste em transmitir periodicamente um pacote que contém um sumário, calculado pelo emissor, de um conjunto (de por exemplo 4) dos últimos pacotes transmitidos. Quando o receptor detecta a falta de um pacote, pode usar o sumário para calcular o valor do pacote em falta. Por exemplo, seja $s = x_1 + x_2 + x_3 + x_4$ o sumário. Caso falte o pacote x_2 , o seu valor pode ser assim calculado: $x_2 = s - (x_1 + x_3 + x_4)$.

Para se perceber uma forma concreta de aplicar a técnica na prática sem usar aritmética de números representados em centenas ou milhares de bits, vamos assumir que cada pacote consiste num único bit. O tipo de sumário mais simples consiste num sumário correspondente ao resultado do XOR (*eXclusive OR*, operação a seguir denotada por \oplus) do conteúdo dos últimos 4 pacotes, *i.e.*, dos últimos 4 bits, transmitidos. Seja o sumário $s = x_1 \oplus x_2 \oplus x_3 \oplus x_4$, caso falte o bit x_2 , então $x_2 = s \oplus x_1 \oplus x_3 \oplus x_4$ visto que com a operação *eXclusive OR*, a soma e a subtração são equivalentes. Por exemplo, se os últimos 4 bits transmitidos foram 1, 1, 0, 0, o seu sumário é 0. Se faltar o segundo bit, é fácil deduzir que o seu valor era 1, pois $x_2 = 0 \oplus 1 \oplus 0 \oplus 0$. Dado que a operação realizada é equivalente ao cálculo da paridade ímpar, estes códigos também se chamam códigos de paridade.

Um pacote não contém um bit mas sim uma grande sequência deles. O raciocínio feito para um bit é agora feito para cada um dos bits de cada pacote como se estes estivessem alinhados coluna a coluna como mostra a Figura 9.8.

Assim, o receptor vai recebendo os pacotes, mas sempre que lhe falta um, espera pelo próximo pacote de sumário e calcula o valor do que está em falta, ver a figura 9.9. É claro que se o pacote em falta for apenas o do sumário não há problema, mas se entre dois sumários faltar mais do que um pacote, não é possível reconstitui-los pois faltam valores.

Esta técnica tem o defeito de introduzir uma percentagem significativa de bits redundantes. Por exemplo, com sumários de 4 pacotes introduz 25% de redundância. Para além disso, em caso de erro, multiplica por 4 o tempo necessário para processar cada pacote. Tal pode não ser muito grave caso o *layout delay* acomode o tempo necessário para encontrar o pacote com o sumário a tempo. Para diminuir a probabilidade de faltar mais do que um pacote entre cada sumário, é possível reduzir cada sumário ao sumário de três, dois ou mesmo de um único pacote. Neste último caso cada pacote é idêntico ao seu sumário, mas necessita-se do dobro do débito requerido pelo codec. No caso do G.711, ao invés de 64 Kbps, seriam necessários 128 Kbps.

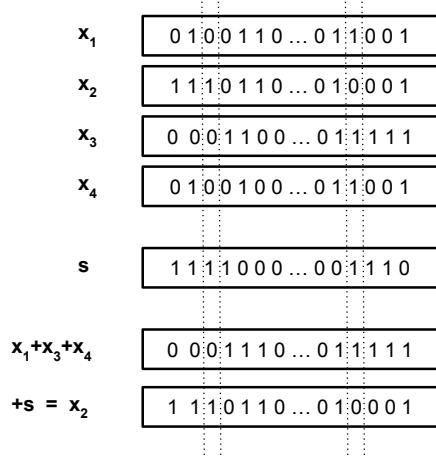


Figura 9.8: Cálculo do pacote sumário e sua utilização para calcular o valor de um pacote em falta

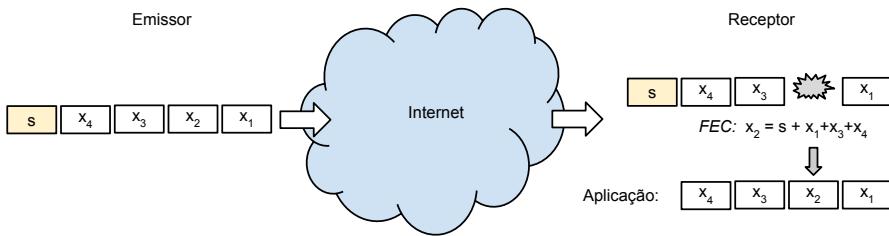


Figura 9.9: Correcção de erros de omissão usando *Forward Error Correction* com base em informação redundante em sumários

A técnica que acabámos de ilustrar é particularmente útil caso se esteja a utilizar IP Multicasting, pois as técnicas de retransmissão de pacotes em falta não são aplicáveis à difusão de um para muitos. De facto, as perdas de pacotes neste cenário são de diferentes pacotes pois cada receptor é servido por diferentes canais. No entanto, o mecanismo permite recuperar de diferentes erros de omissão, ver a Figura 9.10.

Existem outras técnicas de FEC que não introduzem informação redundante mas que se baseiam em tentar reconstruir a informação omitida a partir daquela que a antecede ou daquela que a segue. Os pacotes com 20 ms de amostragem de som contém 160 amostras. Reconstruir os 160 valores em falta é mais difícil, mas é possível usar uma técnica que torna a tarefa mais simples.

Emparelhamento (*Interleaving*)

Se tomarmos os 160 valores de um pacote e os 160 valores do pacote seguinte, é possível enviar no primeiro pacote os bytes pares de ambos os pacotes e no seguinte os bytes ímpares, ver a Figura 9.11. Se um dos pacotes se perder, o receptor, apesar de tudo, dispõe da informação dos dois pacotes mas com uma frequência de amostragem de metade da verdadeira e poderá, por interpolação, obter os valores perdidos.

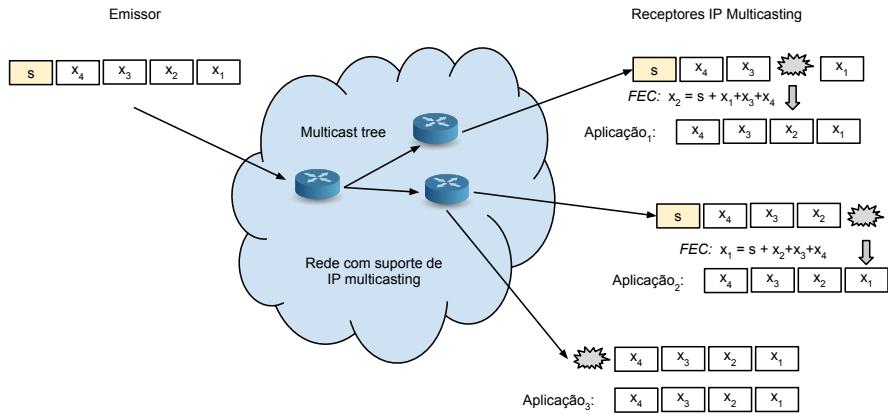


Figura 9.10: Correcção de erros de omissão com *multicasting* usando *Forward Error Correction*

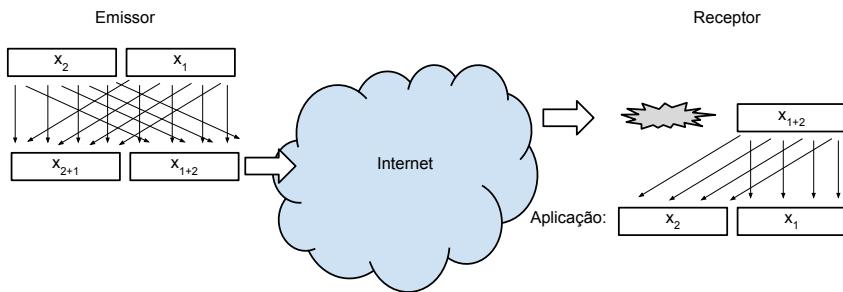


Figura 9.11: *Forward Error Correction* com *interleaving byte a byte*

A técnica ilustrada tem o mérito de não requerer o aumento do débito pois não introduz informação redundante. A mesma baseia-se em obter os bytes em falta por interpolação a partir de amostras com pior amostragem dos valores originais. No entanto, a mesma duplica o tempo necessário para processar cada pacote perdido pois o mesmo só pode ser reconstruído quando chega o seguinte.

Adicionalmente, é impossível usá-la com *frames* que contenham informação multimédia comprimida pois não é possível realizar interpolação sobre a mesma. No entanto, é possível aplicá-la usando outra alternativa que consiste em decompor cada pacote num número limitado de sub-intervalos independentes, por exemplo 4, e transmitir cada sub-intervalo em pacotes distintos, ver a Figura 9.12

Utilizando 4 sub-intervalos de 5 ms cada, caso um pacote seja omitido, é possível reconstruir cada um de 4 pacotes com 5 ms de som em falta em cada um, o que diminui o impacto da falha para apenas 40 bytes. Fazendo uma decomposição num número superior de pacotes diminuiria ainda mais o impacto das falhas, mas aumenta a pressão sobre o *playout delay* na medida em que só é possível recompor cada pacote depois de terem chegado todos os seus sub-intervalos.

Para terminar esta secção vamos ainda ilustrar outra técnica de compensação de erros que consiste em enviar simultaneamente dois fluxos multimédia, com resoluções distintas, e deslocados um do outro.

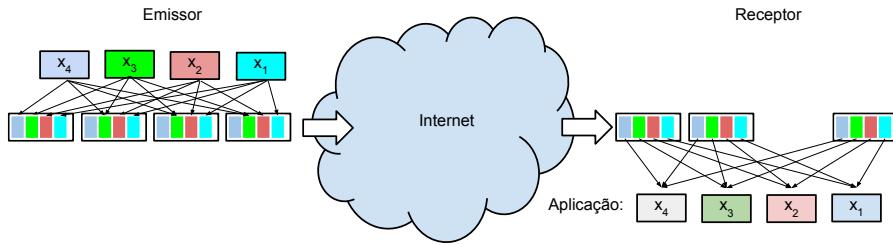


Figura 9.12: *FEC – Forward Error Correction* com *Interleaving* de sub-intervalos

Fluxos emparelhados

Supondo que o fluxo base é o correspondente ao codec G.711 e o fluxo de menor resolução é o correspondente ao codec G.709, que requer apenas cerca de 1/8 do débito do G.711, cada 20 ms de som ocupa 160 bytes no primeiro e cerca de 20 bytes no segundo. Assim, cada pacote contém o *frame* correspondente ao respectivo fluxo G.711, assim como o *frame* correspondente ao pacote anterior codificado segundo o codec G.729. No total, ao invés de 160 bytes, cada pacote tem 180 bytes de informação.

Se um pacote se perder, o pacote seguinte contém o som correspondente, só que com uma resolução inferior. Tal permite ao *player* recompor o som em falta apesar de com pior qualidade. A Figura 9.13 ilustra a técnica.

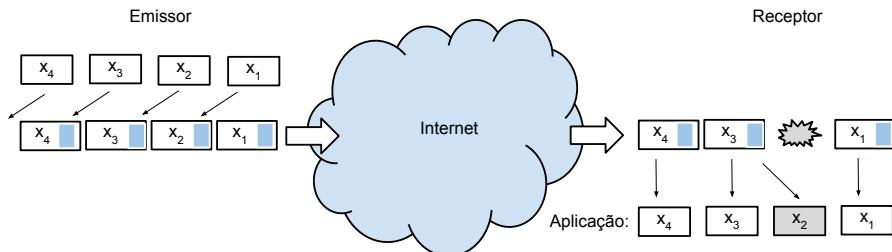


Figura 9.13: *Forward Error Correction* com base num codec de resolução inferior

Esta técnica permite recuperar a perda de um pacote. É possível generalizá-la para mascarar a perda de n pacotes incluindo em cada pacote uma versão comprimida dos n pacotes anteriores. Assim é possível mascarar perdas de maior dimensão à custa de mais fluxos redundantes, *i.e.*, maior débito afectado ao fluxo multimédia, e aumentando o tempo necessário para processar a informação em caso de perdas. Novamente, caso o *playback buffer* e o *playout delay* acomodem este tempo extra, não haverá problemas.

Quando se utiliza um protocolo como o UDP para transporte de informação multimédia, podem surgir erros devido à omissão de pacotes. Para mascarar o seu efeito é possível usar técnicas gerais de FEC (*Forward Error Correction*) com base em códigos de correcção de erros do tipo *erasure*.

No entanto, explorando as propriedades da informação multimédia, é também possível reconstruir a informação em falta usando interpolação de forma tal que a mesma

seja realista e o utilizador não se aperceba da perda de resolução. No limite, é sempre possível reutilizar o último *frame* recebido para cancelar uma perda, particularmente quando se trata de imagens.

Finalmente, para terminarmos esta panorâmica do transporte de informação multimédia, na secção seguinte é introduzido o protocolo RTP – *Real-time Transport Protocol*, que foi definido para suportar a transmissão de informação multimédia em tempo real.

9.4 RTP – *Real-time Transport Protocol*

Para que uma aplicação multimédia funcione, é necessário que as várias partes em comunicação se entendam sobre o número, o formato e o sincronismo dos fluxos multimédia que circulam pela rede. Quando a aplicação e o software envolvido são proprietários (*e.g.*, como no caso da aplicação Skype) não é preciso nenhum investimento suplementar em normalização. No entanto, quando se pretende atingir uma situação em que várias aplicações, desenvolvidas por equipas ou fabricantes distintos, possam funcionar em conjunto, é necessário tentar normalizar os formatos e os protocolos usados para garantir essa inter-operação.

As aplicações multimédia que usam redes são predominantemente aplicações em tempo real, uni ou multi-direcionais, em que os problemas do tempo de trânsito extremo a extremo são críticos, e por isso investiu-se na normalização do transporte de segmentos de informação multimédia (e não de fluxos contínuos sem delimitações de mensagens como o TCP).

A abordagem seguida foi definir um protocolo que completasse o protocolo UDP com funcionalidades comuns à maioria das aplicações multimédia do tipo *live streaming* (*e.g.*, IPTV), ou para suporte de conversas telefónicas e video-conferências (*e.g.*, VoIP e *Conference Call over IP*). Dessa forma conseguiam-se dois objectivos: aplicações desenvolvidas de forma independente poderiam inter-operar, e poupava-se o trabalho de reinventar a roda em cada aplicação.

Apesar da motivação original, nada impede os segmentos RTP de serem transportados sobre TCP, apesar de nesse caso existirem informações inúteis no cabeçalho e faltarem algumas como por exemplo a dimensão de cada segmento.

O resultado final foi um protocolo designado RTP (*Real-time Transport Protocol*). Como mostra a Figura 9.14, o RTP é uma especialização (no sentido da derivação de classes na orientação por objectos) do UDP para transporte de *frames* multimédia pertencentes a um ou mais fluxos multimédia. A Figura 9.14 mostra a inserção do RTP na pilha de protocolos TCP/IP. As mensagens do RTP contêm elementos do nível aplicação e por esse motivo deveríamos chamar-lhes mensagens ou *frames* multimédia e não segmentos ou pacotes. No entanto, a tradição manda que se use o termo pacote RTP, que usaremos também.

A Figura 9.15 mostra o cabeçalho do protocolo RTP. Os primeiros 12 bytes do cabeçalho, *i.e.*, as primeiras três linhas do cabeçalho RTP, são obrigatórias, enquanto que os campos CSRC e de extensão são opcionais, e são específicos daquilo que a norma designa como perfis (*profiles*), que são específicos de categorias de aplicações ou de tipos de informação multimédia. Procurou-se que o cabeçalho se restringisse aos campos comuns a todos os perfis, deixando para cabeçalhos opcionais suplementares o suporte de funcionalidades específicas. Os campos comuns são os mais importantes para se perceber a filosofia do protocolo e são os únicos a seguir explicados.

O cabeçalho tem inicialmente um conjunto de campos de pequena dimensão, quase todos campos do tipo *flags*, que servem para indicar a versão do protocolo, se o *payload* contém ou não informação de *padding*, se está presente ou não um cabeçalho de extensão, *etc.* Um desses campos, o campo CC, será referido a seguir quando for explicado o papel do campo CSRC.

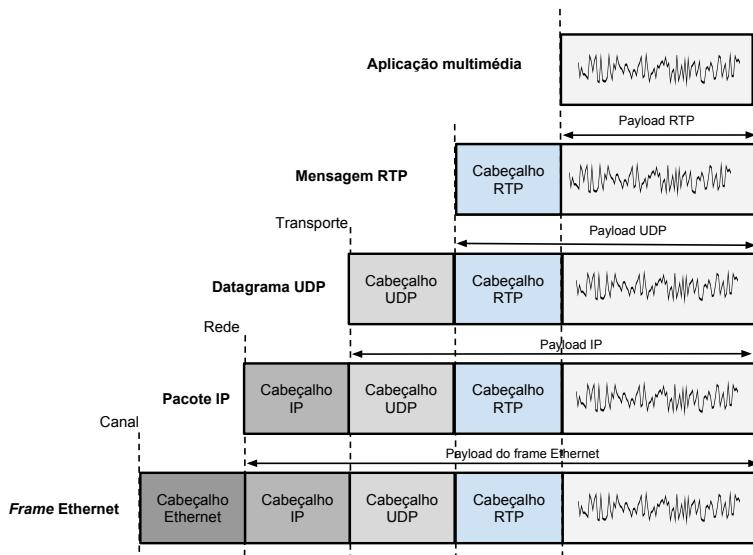


Figura 9.14: Inserção do protocolo RTP na pilha de protocolos TCP/IP

O campo **Payload Type** tem 7 bits e permite codificar o formato do fluxo, nomeadamente o codec usado. A tabela dos codecs possíveis depende do perfil usado pela aplicação e em função do mesmo estão definidas tabelas de codecs. O RFC 3551 define o perfil usado para conferências multimédia. A Tabela 9.1 lista um subconjunto dos valores de **Payload Type** definidos no mesmo. A indicação do tipo de codificação em cada pacote RTP permite alterar a qualquer momento o tipo usado pelo emissor sem necessidade de negociar o mesmo com o ou os receptores.

O campo **Sequence number** permite numerar os pacotes para que o receptor possa detectar as falhas e colocar os pacotes na ordem de emissão. Nenhum mecanismo de ACK e retransmissão está previsto no protocolo. Ao invés disso, os esquemas de FEC ou similares, anteriormente descritos, são usados para lidar com as falhas.

O campo **Timestamp** tem 32 bits e caracteriza a posição no fluxo do início do *payload* do pacote. Este valor não representa um tempo absoluto, nem a norma do RTP define as unidades em que se exprime. O valor inicial é fixado pelo emissor quando inicia o fluxo, e a unidade de progresso é definida pelo tipo do fluxo. Por exemplo, se o codec for o G.711, e um pacote contiver 20 ms de som, isso quer dizer que o valor do campo **Timestamp** do pacote seguinte deverá ser normalmente incrementado de 160 face ao deste pacote. O mecanismo de **Timestamp** é também usado para sincronizar a visualização de vários fluxos, mesmo que os seus valores iniciais e as suas escalas sejam específicas a cada um deles. A indicação de uma **Timestamp** em cada pacote permite substituir o silêncio num fluxo sonoro pela ausência de envio de *frames*.

O campo **Synchronisation source identifier (SSRC)** também tem 32 bits e é um número aleatório que identifica univocamente a origem de um fluxo. Por exemplo, numa conferência multimédia, cada participante usa um número destes diferente. Desta forma a aplicação não está dependente dos endereços IP ou das portas para distinguir as diferentes fontes de informação multimédia existentes. Por exemplo, o mesmo computador, quer tenha um só endereço IP ou vários diferentes e a mudarem dinamicamente, pode ser a origem de uma ou mais fontes de informação multimédia.

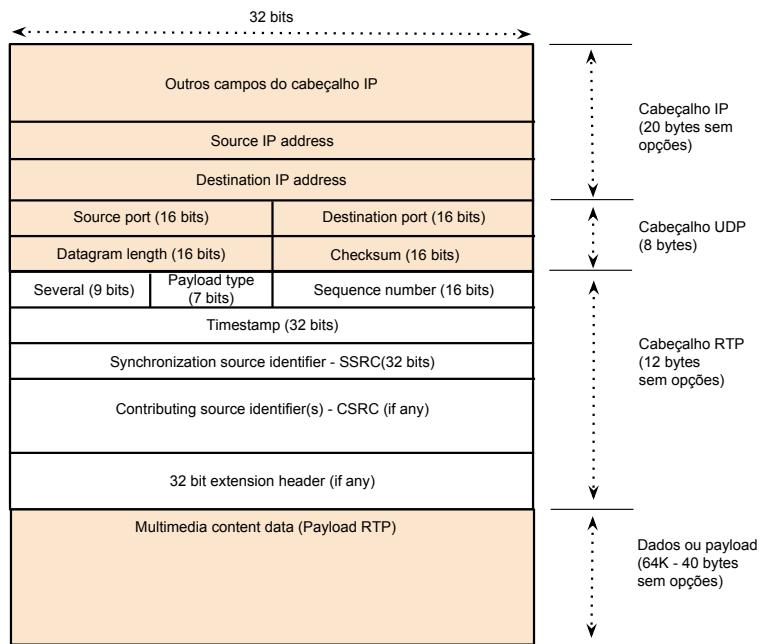


Figura 9.15: Cabeçalho do protocolos RTP

cuja identificação é independente de endereços IP e portas.

Existe um caso particular em que o SSRC não representa um utilizador, mas um misturador, que junta num só fluxo sonoro o som de vários participantes. Neste caso o misturador tem um identificador próprio, mas o campo *Contributing source identifiers (CSRC)* contém a lista das fontes misturadas. A flag CC é posicionada para assinalar que o pacote foi criado por um misturador.

O protocolo RTP é definido pelo RFC 3550. Associado ao mesmo existe outro protocolo, designado RTCP (*Real-time Transport Control Protocol*) que permite enviar periodicamente informação de controlo associada a um fluxo RTP (e.g., *feedback* sobre a qualidade da recepção, formas de sincronizar diferentes fluxos vindos do mesmo utilizador, etc.).

Para fomentar a inter-operação entre aplicações multimédia em tempo real que enviem dados multimédia encapsulados em UDP, foram normalizados os protocolos RTP (*Real-time Transport Protocol*) e RTCP (*Real-time Transport Control Protocol*).

O primeiro destes protocolos normaliza um cabeçalho contendo um conjunto de informações úteis sobre os dados multimédia enviados em cada pacote, entre os quais a identificação dos emissores da informação, o tipo de codificação usado (codec), um número de sequência que permite detectar perdas e uma indicação temporal para permitir a sincronização dos pacotes do mesmo ou de vários fluxos. O segundo normaliza o envio de informações sobre a qualidade da recepção, formas de sincronizar diferentes fluxos vindos do mesmo utilizador, etc.

Tabela 9.1: Codificação de alguns valores do campo do Payload Type segundo o RFC 3551 (lista parcial)

Código	Nome	Tipo	F. amostragem (Hz)	Débito
0	PCMU (G711)	Audio	8.000	64 Kbps
3	GSM	Audio	8.000	13 Kbps
4	G723	Audio	8.000	
6	DVI4	Audio	16.000	
7	LPC	Audio	8.000	2,4 Kbps
11	L16	Audio	44.100	
18	G729	Audio	8.000	
14	MPEG Audio	Audio	90.000	
26	Motion JPEG	Video	90.000	
31	H.261	Video	90.000	
32	MPEG 1	Video	90.000	
33	MPEG 2	Video	90.000	

9.5 Resumo e referências

Resumo

O som digitalizado consiste numa sequência de valores numéricos que representam a intensidade da onda sonora medida a intervalos regulares. A resolução da digitalização corresponde ao número de bits usados para representar os valores medidos. A frequência de amostragem determina as frequências presentes na representação digital. É normal usar 8 bits de resolução e 8 KHz de frequência de amostragem para a digitalização da voz nos sistemas telefónicos, ou 16 bits / 44,1 KHz para a música, ou ainda maior resolução e frequência de amostragem quando existem requisitos de muito alta fidelidade.

Devido às características percepcionais do sentido da audição é possível aplicar algoritmos de compressão do sinal sonoro digitalizado. Estes algoritmos podem implicar perda de informação (são *lossy*) mas não perda de significado nos sistemas telefónicos, nem diferenças demasiado perceptíveis da qualidade musical. Os ganhos assim obtidos podem reduzir o débito do fluxo digital correspondente ao som digitalizado até uma ordem de grandeza, sem um impacto significativo na utilidade final da informação sonora transmitida.

Uma imagem digitalizada contém uma sequência de valores que codificam a cor e a intensidade de um conjunto de pixels, *i.e.*, de unidades elementares da imagem. O número de bits usados para codificar o valor de cada pixel e a densidade de pixels da imagem condicionam a sua qualidade. Um filme é uma sequência de imagens justapostas (*image frames*), com frequências entre 20 e 60 imagens por segundo.

O débito de informação por unidade de tempo requerido para a transmissão de imagens fixas e de filmes é muito elevado se não se utilizarem algoritmos de compressão. Dadas as características de redundância das imagens e dos filmes, assim como as características da vista humana, é possível utilizar algoritmos de compressão de imagem e de filmes que reduzem esse débito até 3 ordens de grandeza, através de perda de informação, mas muitas vezes com resultados imperceptíveis para a vista.

A informação multimédia tem características particulares que permitem que a “mesma” informação seja transmitida com resoluções distintas, ou mesmo com erros, sem que a sua utilidade seja afectada na mesma proporção.

Esta característica, a par de outros requisitos das aplicações que usam informa-

ção multimédia, tem um impacto significativo sobre as técnicas de transmissão de informação usadas sobre uma rede de pacotes.

Todas as aplicações multimédia em tempo real têm requisitos de qualidade de serviço específicos que estão dependentes do tipo da aplicação, das expectativas dos utilizadores e do seu contexto de utilização. Esses requisitos dizem respeito ao débito, ao tempo de trânsito e ao *jitter*.

Os primeiros estão ligados às exigências dos codecs usados, os quais podem ser adaptados à capacidade disponível. Os requisitos mais difíceis de satisfazer são os ligados ao tempo de trânsito e *jitter*, e são particularmente importantes para as aplicações interactivas.

O protocolo TCP é simples, omnipresente, compensa os erros e adapta-se à capacidade disponível na rede. Por estas razões constitui também um suporte adequado à transferência de informação multimédia se o débito médio extremo a extremo for o adequado ao codec usado.

Infelizmente, as suas virtudes têm como repercussão a amplificação do *jitter* extremo a extremo, em particular na presença de erros e também devido à actuação dos mecanismos de controlo da saturação. Este defeito pode ser combatido usando um *playout delay* mais longo.

Apesar de essa solução ser compatível com aplicações unidireccionais, a mesma introduz atrasos suplementares e incómodos para os utilizadores finais, pelo que muitas vezes as aplicações recorrem à redução da resolução para diminuírem os requisitos de capacidade de rede e minorarem os efeitos negativos sobre o *jitter*.

A introdução de mecanismos de adaptação dinâmica da resolução dos fluxos multimédia à qualidade da rede é uma solução comum que tenta minorar os efeitos do controlo de saturação do TCP. Quando as aplicações são interactivas, a necessidade de encurtar tanto quanto possível o *playout delay*, implica a redução da qualidade dos fluxos multimédia através do uso de codecs de pior qualidade ou com parâmetros ajustados.

A transmissão de conteúdos multimédia sobre UDP tem a vantagem de ser compatível com a utilização de um *playout delay* curto, pois o protocolo não introduz dilatações suplementares do *jitter*. Adicionalmente, o protocolo é compatível com a difusão (*multicasting*), pelo que aumenta a escalabilidade do serviço quando existem muitos clientes a subscrever simultaneamente o mesmo conteúdo.

No entanto, quando se utiliza um protocolo como o UDP para transporte de informação multimédia, podem surgir erros devido à omissão de pacotes. Para mascarar o seu efeito é possível usar técnicas gerais de FEC (*Forward Error Correction*) com base em códigos de correcção de erros do tipo *erasure*.

No entanto, explorando as propriedades da informação multimédia, é também possível reconstruir a informação em falta usando interpolação de forma tal que a mesma seja realista e o utilizador não se aperceba da perda de resolução. No limite, é sempre possível reutilizar o último *frame* recebido para cancelar uma perda, particularmente quando se trata de imagens.

Para fomentar a inter-operação entre aplicações multimédia em tempo real que enviem dados multimédia encapsulados em UDP, foram normalizados os protocolos RTP – *Real-time Transport Protocol* e RTCP – *Real-time Transport Control Protocol*.

O primeiro destes protocolos normaliza um cabeçalho contendo um conjunto de informações úteis sobre os dados multimédia enviados em cada pacote, entre os quais a identificação dos emissores da informação, o tipo de codificação usado (codec), um número de sequência que permite detectar perdas e uma indicação temporal para permitir a sincronização dos pacotes do mesmo e de vários fluxos.

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

Conversor analógico - digital (*ADC – Analog to digital converter*) Dispositivo que converte uma onda eléctrica analógica numa sequência de valores numéricos.

Conversor digital - analógico (*DAC - Digital to Analog Converter*) Dispositivo que converte uma sequência de valores numéricos numa onda elétrica analógica.

Freqüência de amostragem (*Sampling rate*) Freqüência com que um conversor analógico - digital produz amostras do sinal analógico.

Resolução das amostras (*Sample quantization*) Número de bits com que o valor de cada amostra é expresso por um conversor analógico - digital.

Compressão (*Compression*) Os fluxos multimédia contêm uma grande quantidade de informação e por isso os mesmos são comprimidos para serem transmitidos. As técnicas de compressão usadas podem ser sem perda de informação (*lossless*) ou com perda de informação (*lossy*).

Fidelidade (*Fidelity*) A informação multimédia depois de digitalizada e comprimida, é memorizada ou transmitida e depois descomprimida e finalmente restituída aos utilizadores. A fidelidade é uma propriedade subjectiva e qualitativa que capta a maior ou menor percepção pelos utilizadores finais da equivalência entre o sinal original e o restituído. Os utilizadores podem considerar que existe alta fidelidade mesmo que no processo se tenha perdido informação devido às características dos sentidos auditivo e visual. As técnicas usadas dizem-se psico-acústicas e psico-visuais.

Pixel (*Pixel*) Unidade de informação elementar e não decomponível sobre a cor e a intensidade de uma imagem. Uma imagem é formada por uma grande quantidade de pixels e a sua fidelidade exprime-se em termos da densidade dos pixels por unidade de superfície, assim como da quantidade de bits usados para exprimir a cor e a transparência.

RGB (*Red Green Blue*) Forma de codificação de qualquer cor como uma composição das cores vermelho, verde e azul em diferentes proporções.

Frame rate Um filme é composto por uma sequência de imagens cuja quantidade por unidade de tempo condiciona a sua qualidade. A vista humana é incapaz de distinguir as diferentes imagens desde que o seu número por unidade de tempo seja superior a 20 ou 30 por segundo. Para garantir conforto da visualização dos filmes é frequente usar 50 imagens por segundo. Essas imagens também se podem chamar fotogramas.

Codec (*CODer / DECoder*) Dispositivo hardware, software ou misto, capaz de codificar e descodificar informação multimédia codificada de acordo com um formato específico. Muitas vezes, informalmente, o termo é usado para designar quer o dispositivo, quer o formato.

Qualidade de serviço em multimédia (*Multimedia QoS*) A transmissão de um fluxo multimédia em tempo real (com “simultaneidade” entre a sua transmissão e consumo) requer da rede um conjunto de garantias de qualidade de serviço, nomeadamente débito mínimo e tempo de trânsito e *jitter* limitados. As exigências temporais são particularmente importantes e as mais difíceis de garantir.

Correcção de erros sem retransmissão (*FEC – Forward Error Correction*) Técnicas usadas para mascarar os erros que se produzem na transmissão de dados, sem retransmissão dos dados chegados com erros, baseada na transmissão de informação suplementar que permite compensar a informação errada.

Erros de omissão (*Erasure errors*) Erros caracterizados por a informação recebida conter omissões face à informação transmitida, mas a informação que é de facto recebida não contém erros. Os códigos FEC especializados em compensação de erros de omissão dizem-se *erasure codes*.

Correcção de erros por emparelhamento (*Interleaving FEC*) Técnica de correcção de erros baseada em decompor cada unidade de informação a transmitir em

subconjuntos e na transmissão de pacotes contendo um emparelhamento de subconjuntos de diferentes unidades. Caso se perdam pacotes, esta técnica permite compensar os erros de omissão por interpolação sobre a informação parcial recebida.

RTP – Real-time Transport Protocol Protocolo que enriquece a noção de *frame* multimédia com a transmissão de informações particularmente adequadas ao transporte de fluxos multimédia, nomeadamente: tipo do codec dos dados transmitidos na parte de dados, números de sequência para detecção de erros de omissão, informação de sincronização temporal, *etc.* O protocolo é complementado com o protocolo RTCP – *Real-time Transport Control Protocol* que permite enviar periodicamente informação de controlo associada a um fluxo RTP (*e.g.*, *feedback* sobre a qualidade da recepção, formas de sincronizar diferentes fluxos vindos do mesmo utilizador, *etc.*).

Referências

Este capítulo apresenta uma breve introdução aos conceitos base sobre a informação multimédia. O leitor interessado em aprofundar o tema poderá consultar um dos livros recomendados pelo SIGMM (*Special Interest Group on MultiMedia*) da ACM (ver mais abaixo a referência para a página Web do SIGMM).

As técnicas de correcção de erros de omissão independentes do tipo de informação transmitida são uma área muito activa, com aplicações à difusão de informação multimédia, aos sistemas móveis e aos sistemas de arquivo de informação. O leitor interessado poderá consultar os RFCs 5053 e 5510 assim como os artigos [Rizzo, 1997; MacKay, 2005]. O artigo [Perkins et al., 1998] apresenta uma síntese das técnicas de recuperação de erros de omissão usando as propriedades da informação multimédia.

O protocolo RTP é um protocolo com a ambição de suportar uma família de aplicações muito variada e cujos requisitos estão em constante evolução. O argumento “end-to-end” recomenda um grande controlo das aplicações sobre as funcionalidades disponibilizados pelas camadas inferiores, para evitar duplicação de funcionalidades. O sucesso do arquitecto de protocolos está intimamente ligado à sua capacidade de definir o que fica em cada nível. O artigo [Clark and Tennenhouse, 1990] propõe formas de navegar neste labirinto com especial ênfase nas aplicações multimédia.

Apontadores para informação na Web

- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- <http://sigmm.org/Education> – Página oficial do SIGMM com recursos educacionais como cursos, vídeos educacionais, livros e software.
- <http://www.itu.int/rec/T-REC-G.114-200305-I/en> – Dá acesso à recomendação G.114 da ITU “One-way Transmission Time” de 2003, que discute o papel do tempo de trânsito extremo a extremo no telefone.
- <http://www.jpeg.org> – Site oficial dos membros do comité que desenvolve a norma JPEG.
- <https://en.wikipedia.org/wiki/JPEG> – Entrada na Wikipedia sobre as normas JPEG e contém uma interessante descrição dos aspectos técnicos do conjunto das normas JPEG.
- <http://mpeg.chiariglione.org> – Site oficial dos membros do comité que desenvolve a norma MPEG.

9.6 Questões para revisão e estudo

1. Numa empresa existe um serviço de atendimento aos clientes via telefone, suportado em VoIP, em que todas as chamadas telefónicas são gravadas. A chamada média dura 320 segundos. O codec usado é o G.711 com frequência de amostragem de 8 KHz e um byte de resolução por amostra, produzindo um débito de 64 Kbps. Supondo que se pretende gravar as chamadas, quanto espaço em disco é necessário para gravar, sem compressão, uma chamada de duração média nas seguintes situações.
 - (a) A gravação contém todo o som digitalizado dos dois interlocutores sem mistura. A gravação inclui todos os períodos de silêncio, em que nenhum dos interlocutores fala.
 - (b) A gravação contém todos os pacotes RTP trocados entre os dois interlocutores admitindo que o codec transmite em cada pacote 20 ms de som. A gravação inclui todos os períodos de silêncio, em que nenhum dos interlocutores fala, e todos os cabeçalhos (IP, UDP e RTP).
 - (c) Questão idêntica à alínea anterior mas a gravação contém pacotes RTP contendo todos os sons correspondentes à conversa mas misturados num único fluxo sonoro.
 - (d) Sugira, continuando a usar o formato RTP, formas de diminuir a quantidade de informação guardada continuando a usar o mesmo formato para o som.
2. Pretende-se gravar em disco um CD de música com a duração de uma hora, sem compressão. Quanto espaço em disco é necessário?
3. Uma chamada telefónica é suportada no protocolo RTP e o codec usado é o G.711 com frequência de amostragem de 8 KHz e um byte de resolução por amostra, produzindo um débito de 64 Kbps. Os pacotes contêm 20 ms de som e têm a dimensão 160 bytes mais cabeçalhos. O tempo de trânsito entre o emissor e o receptor varia entre 50 e 200 ms. O receptor usa um *playout delay* fixo de 150 ms. Na tabela abaixo estão indicados os momentos de chegada dos pacotes ao receptor.
 - (a) Complete a tabela indicando o momento em que cada pacote começa a ser reproduzido.
 - (b) Qual a dimensão do *playback buffer* necessário para acomodar o *playout delay* no receptor.
4. Pretende-se gravar em disco um filme com a duração de uma hora, sem compressão. O filme está gravado usando 50 *frames* por segundo, cada *frame* tem 600×400 pixels a cores, e cada pixel é codificado em 2 bytes. Quanto espaço em disco é necessário?
5. Um servidor de *on-demand streaming* envia um fluxo contínuo com o débito constante de 2 Mbps de informação para um cliente por TCP. O tempo de trânsito entre o servidor e o cliente é de 120 ms. O cliente tem um *playback buffer* com 4 Mbytes que momentaneamente está cheio. Admitindo que o TCP do emissor pára de enviar dados devido a um episódio de saturação momentâneo, quanto tempo máximo pode durar essa paragem? Despreze os factores de segurança necessários para acomodar as necessidades de informação anterior ou posterior para a restituição do som e da imagem.
6. Um *frame* MPEG tem um erro, quantos *frames* de um filme podem ser afectados por esse erro: um ou vários? Justifique a sua resposta.

Tabela 9.2: Pacotes e momento da sua chegada

Número de seq. do pacote	Momento da chegada (ms)	Momento para ser “tocado”
1	85	
2	125	
4	145	
5	160	
3	180	
8	220	
6	245	
7	285	
9	290	
10	310	

7. Verdade ou mentira? Os códigos de correcção de erros do tipo *erasure* conseguem por si sós corrigir erros com origem no ruído nos canais que alteram alguns bits nos pacotes. Justifique a sua resposta.
8. Verdade ou mentira? O código de correcção de erros do tipo FEC conhecido por código de paridade podia ser aplicado à transmissão de ficheiros, evitando toda a retransmissão de pacotes. Justifique a sua resposta.
9. Qual a percentagem de informação de controlo de erros suplementar introduzida por um código de correcção de erros do tipo FEC usando paridades quando cada sumário cobre os 3 pacotes anteriores? Justifique a sua resposta.
10. Um código de correcção de erros do tipo FEC usando paridades em que cada sumário cobre exactamente um pacote anterior, que erros de omissão consegue compensar e quais os que não consegue? Justifique a sua resposta.
11. Verdade ou mentira? A técnica de correcção de erros do tipo FEC usando *interleaving* podia ser aplicada à transmissão de ficheiros. Justifique a sua resposta.
12. Qual a percentagem de informação de controlo de erros suplementar introduzida por um código de correcção de erros do tipo FEC usando *interleaving* quando cada pacote contém informação de 5 pacotes? Justifique a sua resposta.
13. Verdade ou mentira? O RTP poderia usar o campo *Timestamp* para detectar a perda de pacotes. Justifique a sua resposta.
14. Os valores do campo *Timestamp* dos pacotes RTP podem ser colocados pelo software de implementação do próprio RTP no momento em que vai enviar o datagrama UDP? Justifique a sua resposta.
15. Uma aplicação multimédia pode enviar em momentos diferentes vários pacotes RTP diferentes com o mesmo valor de *Timestamp*? Justifique a sua resposta.
16. Uma aplicação multimédia pode enviar praticamente no mesmo momento vários pacotes RTP diferentes com diferentes valores de *Timestamp*? Justifique a sua resposta.

Capítulo 10

Outros protocolos de transferência de dados

Generality in the network increases the chance that a new application can be added without having to change the core of the network.

– Autores: *M. Blumenthal e David Clark*

Os protocolos UDP e TCP foram definidos durante a década de 1980. O protocolo UDP manteve-se desde então inalterado. Em contrapartida, o protocolo TCP teve imensas evoluções, com especial realce das ligadas à problemática do controlo da saturação, um mecanismo completamente ausente na definição inicial dos protocolos TCP/IP. Durante todos estes anos a escala e a capacidade da Internet evoluíram de forma explosiva, tal como as aplicações e o tipo de tráfego dominante, com repercuções profundas nas solicitações exercidas pelas aplicações sobre os protocolos de transporte. Qual foi o impacto dessa nova realidade?

Os anos de 1990 foram dominados pelo início da comercialização da Internet e pela subida vertiginosa da sua escala e número de utilizadores. A aplicação dominante passou a ser o acesso à Web, suportada no protocolo TCP. Por isso este protocolo foi sujeito a grande pressão e o seu mecanismo de controlo da saturação continuou a ser melhorado. No advento do século XXI esta tendência manteve-se e o número de utilizadores e a capacidade dos canais continuou a subir vertiginosamente, mas juntaram-se-lhe dois factos novos.

Por um lado todas as redes convergiram para redes IP e a imagem de televisão, o vídeo e as chamadas telefónicas, primeiro de forma temerosa e mais recentemente de forma vigorosa, convergiram para a Internet. O vídeo, a televisão e as chamadas telefónicas usam, ou pelo menos usavam, principalmente UDP. A problemática da subida do tráfego UDP e da sua coexistência com o TCP subiu para primeiro plano e exerceu pressão sobre a problemática do controlo da saturação e da qualidade de serviço da rede. A qualidade de serviço da rede não é algo que se resolva exclusivamente ao nível dos protocolos de transporte, mas estes participam na solução deste problema.

A necessidade de fazer convergir as redes tradicionais com as redes IP e a Internet em geral, e a cada vez maior utilização da Internet para transportar tráfego multimédia em tempo real, levaram ao desenvolvimento de dois novos protocolos de transporte que serão a seguir introduzidos: os protocolos DCCP e SCTP.

Mais recentemente, a subida da escala da Internet foi acompanhada pela vulgarização dos dispositivos móveis. Estes dispositivos, para além de usarem canais sem fios, bastante diferentes dos canais com fios tradicionais, são igualmente caracterizados por terem várias interfaces que podem estar activas simultaneamente.

Na secção 8.5 já nos referimos ao facto do TCP ter dificuldade em lidar com a taxa de erros dos canais sem fios dado o seu impacto no controlo da saturação. Adicionalmente, tal como concebido inicialmente, o protocolo não era capaz de explorar adequadamente as novas interfaces disponíveis, em particular, o facto de que o mesmo computador poder ter várias interfaces activas simultaneamente.

O protocolo SCTP já tinha introduzido a possibilidade de cada um dos dois extremos de uma conexão terem simultaneamente diferentes endereços IP. De forma mais oportuna ainda, uma nova versão do protocolo TCP, o Multi-Path TCP, introduziu o suporte de várias interfaces e vários endereços IP de cada lado da conexão como a regra e não como exceção. Esta versão do protocolo TCP é compatível com o TCP tradicional e foi definida de tal forma que a rede, e sobretudo os equipamentos de segurança, não se apercebesse da diferença.

Com efeito, a Internet é hoje em dia sujeita a uma pressão muito significativa do ponto de vista da segurança e da escala. Essa pressão tem levado à introdução na rede de uma nova classe de equipamentos, lado a lado com os comutadores de pacotes, que analisam, filtram e bloqueiam os pacotes, na ansia de bloquearem, se possível, os potenciais atacantes. Estes equipamentos, genericamente designados por *middleboxes* (*e.g., firewalls, IDS - intrusion detection systems, etc.*), são cada vez mais numerosos e estão parametrizados por omissão para bloquearem todos os protocolos de transporte distintos do UDP e do TCP e a maioria das portas não conhecidas e normalizadas. Esta evolução traduziu-se na chamada “ossificação da Internet” que consiste, no essencial, em cada rede bloquear todos as novidades vindas das outras redes, nomeadamente por “medo do desconhecido”. A regra passou a ser “tudo o que não é explicitamente permitido, é proibido”.

Esta situação tem sido responsável pela impossibilidade de introduzir novos protocolos de transporte (e não só) na Internet, e tem sido um factor que explica o insucesso em geral de tentativas de alteração significativa dos mesmos.

Neste capítulo passamos em revista de forma muito breve algumas das propostas que têm sido feitas para alargar o número de protocolos de transporte usados na Internet global. A situação exposta acima é responsável por a maioria dos esforços de inovação a este nível estarem concentrados em propostas que mantenham a compatibilidade com o UDP e o TCP, salvo em redes particulares, controladas por uma entidade, ou por um pequeno número de entidades. Por esta razão, a grande maioria dos esforços actuais de evolução dos métodos de transferência de dados concentra-se essencialmente ao nível aplicação, como será explicado no fim do capítulo.

Começamos por apresentar o protocolo DCCP, uma evolução do UDP no sentido da introdução de controlo da saturação.

10.1 Datagram Congestion Control Protocol

Com a evolução da escala, o controlo de saturação tornou-se crítico para a Internet. O protocolo UDP não incorpora controlo da saturação, mas tornou-se a certa altura popular na Internet para o suporte do transporte de informação multimédia em tempo real, como por exemplo canais de televisão, filmes interactivos e chamadas telefónicas, ver a Secção 9.4. Quando o UDP compete pela capacidade de canais gargalo com o protocolo TCP, o resultado é muito desfavorável para o TCP. É possível introduzir controlo da saturação ao nível das aplicações, mas isso revela-se complexo e não é suficientemente geral. Procurou-se então definir um protocolo semelhante ao UDP

mas que fosse *TCP friendly*, i.e., que não monopolizasse a utilização da capacidade disponível, através da utilização de controlo da saturação.

O protocolo DCCP (*Datagram Congestion Control Protocol*) associa a capacidade de enviar datagramas, que é semelhante à do UDP, com uma noção de conexão e com controlo da saturação, incluindo suporte de ECN. O RFC 4336 e o artigo [Kohler et al., 2006] apresentam a motivação e a filosofia do protocolo, o RFC 4340 é a proposta de norma.

Apesar de à primeira vista o DCCP parecer relativamente simples, pois a sua semântica e os serviços que presta são próximos dos do UDP e, ao contrário do TCP, não assegura fiabilidade, a verdade é que o protocolo revelou-se mais complexo do que o seu objectivo inicial poderia sugerir.

Essa complexidade está relacionada com a gestão da conexão e a robustez contra ataques de negação de serviço durante a sua abertura, a robustez contra ataques ao funcionamento da conexão usando números de sequência falsos, a gestão dos números de sequência que, dada a necessidade de aceitar a perda de pacotes como fazendo parte do funcionamento normal, tornam a interpretação dos ACKs mais complexa e delicada, a gestão da mobilidade e da utilização simultânea de vários endereços distintos, e o suporte de um conjunto de diferentes algoritmos de controlo da saturação parametrizáveis pelas aplicações.

O DCCP está apenas implementado para os sistemas Linux e FreeBSD e a sua utilização fora de ambientes experimentais é praticamente nula. Provavelmente, esta situação é explicada pela resistência à introdução de novos protocolos, mas também pela generalização da difusão de canais de televisão recorrendo a UDP, serviço designado *TV-over-IP (IPTV)*, e do transporte de chamadas telefónica recorrendo a UDP, designado *Voice-over-IP (VoIP)*, mas recorrendo a uma espécie de sub-rede dedicada, controlada por um operador de acesso e com capacidade reservada para servir os seus clientes directos destes serviços. Adicionalmente, mais recentemente, a difusão de informação multimédia na Internet tornou-se mais realista usando TCP e recurso a adaptação dinâmica da resolução, ver a Secção 9.2.

10.2 Stream Control Transmission Protocol

O protocolo SCTP (Stream Control Transmission Protocol) foi definido pelo grupo de trabalho SIGTRAN (*Signalling Transport*) do IETF com o objectivo inicial de permitir a implementação da interligação de centrais telefónicas digitais sobre redes IP e para controlo e suporte de chamadas telefónicas transportadas na forma de *voice-over-IP (VoIP)*.

O RFC 3286 apresenta uma descrição introdutória e a motivação e o RFC 4960 define o protocolo. Trata-se de um protocolo muito completo que reteve do TCP o carácter orientado conexão, a fiabilidade (em opção) e o controlo da saturação, mas acrescentou-lhe novas funcionalidades entre as quais as seguintes:

4-way handshake A motivação desta nova forma de estabelecer a conexão é impedir os ataques de negação de serviço do tipo SYN-Flood, ver a Secção 7.3. As quatro mensagens necessárias para completar a abertura da conexão chamam-se INIT, INIT-ACK, COOKIE-ECHO e COOKIE-ACK. A parte que abre passivamente a conexão só a considera activa depois de receber a mensagem COOKIE-ECHO e enviar COOKIE-ACK; a parte que abre activamente a conexão só a considera activa depois de receber a mensagem INIT-ACK e enviar a resposta COOKIE-ECHO.

Mensagens (framing) O protocolo TCP é orientado a uma sequência de bytes, o SCTP, tal como o UDP, mantém a noção de mensagem o que facilita a definição

de protocolos complexos do nível superior sem recorrer a métodos especiais de delimitação das mensagens.

Multi-fluxo (*multi-streaming*) Com o protocolo TCP, para transferir diversos fluxos entre os mesmos dois interlocutores, com prioridades e papéis distintos, e sem que uns atrasem os outros, é necessário usar simultaneamente mais do que uma conexão. Se algumas delas estiverem reservadas para controlo aplicacional e com pouco tráfego, o controlo de saturação impede-as de usarem uma fracção adequada da capacidade disponível. O SCTP resolve este problema introduzindo a noção de *sub-stream* na conexão. Assim, por uma única conexão, com um mecanismo unificado de controlo da saturação, circulam diversos fluxos de mensagens, com prioridades distintas. Como resultado desta opção, a transferência de um objecto de grande dimensão também não bloqueia o envio, recepção e tratamento imediato de mensagens de controlo urgentes por exemplo.

Ordenação das mensagens (*multiple delivery mode*) Para cada fluxo é possível seleccionar um de vários modos de entrega das mensagens, nomeadamente: ordem total por fluxo, mas parcial entre fluxos, ou com entrega imediata das mensagens mal cheguem, ou seja sem ordem, como no UDP. Para emular o TCP é necessário usar um único fluxo com ordem total. Adicionalmente, o SCTP permite que um emissor avise o receptor para ignorar dados atrasados, *i.e.*, dados chegados já depois de serem úteis.

Estas diferentes formas de ordenação permitem corrigir um problema chamado “*head-of-line blocking*” que aparece quer quando um segmento TCP recebido fora de ordem não pode ser entregue devido à falta de alguns dos dados que o antecedem, quer quando dados urgentes só chegam ao receptor depois de chegarem todos os dados emitidos antes deles.

Multi-endereço (*multi-homing*) As interligações entre centrais telefónicas têm de ser mantidas com fiabilidade durante longos períodos. Por outro lado, as redes com preocupações de fiabilidade estão interligadas por múltiplos canais com o exterior. Associar diversos endereços IP ao mesmo extremo da conexão permite explorar activamente essa diversidade para efeitos de distribuição de carga e fiabilidade, sem necessidade de quebrar a conexão, e sem necessidade de estabelecer mais do que uma conexão entre as mesmas entidades. O uso desta funcionalidade para suporte directo de mobilidade tem sido também explorada.

O SCTP foi definido mais do que 20 anos depois do TCP e do UDP e por isso, para além de fornecer serviços mais ricos e diversificados do que o TCP e o UDP juntos, incorpora igualmente opções de implementação que reflectem a rica experiência que o precedeu. Um exemplo disso é a abertura da conexão que assume uma forma que permite combater ataques do tipo SYN-Flood, mas também o facto de que durante a abertura da conexão os dois interlocutores estabelecem uma *verification tag*, um número aleatório que será futuramente transmitido em todos os segmentos da conexão para combater ataques que visam introduzir segmentos falsos na mesma.

Um outro exemplo consiste na divisão de um segmento em pedaços, chamados *chunks*, que permitem o envio no mesmo segmento de dados de vários sub-fluxos mas também, os chamados *control-chunks*, que permitem bastante maior flexibilidade do que um cabeçalho com formato único, um conjunto fixo de *flags*, e um campo de opções com a dimensão máxima de 40 bytes, como os usados pelo TCP. Tal tem-se revelado insuficiente em muitas situações, sobretudo para suporte de SACKs e da opção *timestamp* simultaneamente. Existem inúmeros outros exemplos desta maturidade como por exemplo a utilização de CRCs de 32 bits, que podem ser calculados em software, ou o facto da utilização de SACKs ser obrigatória e não uma opção.

O protocolo SCTP está disponível em todos os sistemas de operação e existe bastante experiência operacional da sua utilização [Dreibholz et al., 2011]. Apesar disso, a sua utilização pelas aplicações mais populares não é generalizada, em particular pelos

clientes e servidores Web que poderiam beneficiar muito das suas funcionalidades para o acesso a serviços Web. Isso está, provavelmente, ligado ao problema de todos os protocolos distintos de TCP e UDP serem na prática bloqueados, mas também ligado ao facto de que a adopção de SCTP implica a utilização de uma nova interface de sockets SCTP, o que implica que muitas aplicações teriam de ser alteradas.

Recentemente, o SCTP foi introduzido na chamada “*WebRTC – Real-Time Communications for the Web*” [Jennings et al., 2013], uma iniciativa para dotar os diferentes *browsers Web* de um conjunto de interfaces, mecanismos e protocolos que facilitam a utilização de comunicações interactivas, com suporte de tráfego multimédia directamente entre *browsers Web*. No entanto, para assegurar que o tráfego SCTP não é bloqueado, a proposta WebRTC prevê que os segmentos SCTP sejam encapsulados em datagramas UDP, ver o RFC 6951.

A seguir vamos ver outro protocolo que propõe uma forma alternativa de lidar com alguns dos problemas que o SCTP já tenta resolver, nomeadamente a utilização de diversos canais.

10.3 Multi-Path TCP

Como foi referido no início do capítulo, a multiplicidade de novos dispositivos móveis como *smart-phones*, *pads*, etc. introduziu uma nova categoria de computadores que podem ligar-se à rede através de diversos canais (*e.g.*, interfaces de rede Wi-Fi, interfaces de redes celulares, interfaces de rede *ethernet* com fios, ...). Cada uma dessas interfaces tem um endereço IP distinto. Adicionalmente, esses dispositivos são móveis e a mobilidade implica frequentemente a alteração do endereço IP.

As conexões TCP tradicionais estão associadas aos endereços IP e às portas dos extremos. Qualquer alteração de um endereço implica a quebra da conexão. O protocolo Multi-Path TCP (MPTCP) procura responder a este problema permitindo que uma conexão TCP possa explorar diversos canais e diversos endereços IP entre os dois extremos da conexão. Desta forma é possível explorar vários canais e caminhos alternativos entre os dois extremos da conexão para distribuição de carga. Adicionalmente, é possível ter diversas associações activas entre diferentes pares de endereços IP, e um dispositivo móvel manter uma conexão TCP activa mesmo que o dispositivo altere o endereço IP e o ponto de ligação à rede de uma das suas interfaces. Tudo o que é necessário é que durante a transição pelo menos alguma associação entre um par de endereços continue disponível para comunicar.

O MPTCP foi desenhado tendo em consideração algumas das razões que levaram à adopção residual do DCCP e em parte do SCTP: serem protocolos diferentes de TCP e do UDP e implicarem alterações à interface de sockets e às aplicações [Paasch and Bonaventure, 2014]. Assim, a sua concepção teve por objectivos: ser capaz de usar simultaneamente diversas interfaces e caminhos dentro da rede, coexistir facilmente com TCP (ser *TCP friendly*) e usar os mesmos tipos de controlo de saturação que o TCP, não necessitar de alterações das aplicações e, ao competir com TCP, continuar a existir equidade entre conexões.

O MPTCP aparece para o TCP convencional como uma nova opção. Se ambas as extremidades da conexão suportarem MPTCP, passam a usar uma noção de sub-fluxo, cada um dos quais associado a um par distinto de endereços, mas com as mesmas portas [Raiciu et al., 2012]. O estabelecimento da conexão inicial estabelece o primeiro sub-fluxo mas, posteriormente, outros podem ser acrescentados ou suprimidos. No estabelecimento da conexão inicial as duas extremidades também trocam chaves que lhes permitem depois identificar inequivocamente a conexão independentemente dos sub-fluxos. Cada um dos sub-fluxos é acrescentado por um novo *three-way handshake* suplementar, cujo estabelecimento, devido a usar endereços distintos, é sujeito a autenticação usando as chaves trocadas no estabelecimento da conexão inicial.

Para manter a equidade com as conexões TCP, o MPTCP executa controlo de saturação independente em cada um dos seus sub-fluxos, mas globalmente coordenado, e altera o débito usado em cada um deles para explorar da melhor forma possível a capacidade disponível. Para manter a equidade, o protocolo usa um controlo da saturação que se traduz numa subida mais lenta da dimensão da janela de emissão em cada um dos sub-fluxos do que o TCP tradicional no seu único fluxo, mas que é globalmente equitativa para o MPTCP. Adicionalmente, o MPTCP privilegia o uso dos sub-fluxos menos saturados [Wischik et al., 2011]. Esta é uma área onde o MPTCP abriu, provavelmente, novas avenidas para a investigação no controlo da saturação multi-caminho.

Seguindo uma tendência moderna, e a exemplo do protocolo DCCP, os algoritmos de controlo da saturação são vários e podem ser alterados pelas aplicações.

A possibilidade de usar simultaneamente diversos sub-fluxos através de interfaces distintas permite usar a conectividade de uma interface para compensar os erros noutra, ou activar dinamicamente novas interfaces durante os períodos de transição entre pontos de acesso à rede. Por exemplo, a ligação via uma rede celular pode compensar momentaneamente as falhas de conectividade durante a transição da ligação via Wi-Fi entre diferentes pontos de acesso [Paasch et al., 2012]. A utilização simultânea de várias interfaces foi também usada num centro de dados para demonstrar que o MPTCP pode ser usado para que uma conexão MPTCP única possa explorar simultaneamente todos os caminhos disponíveis na rede do centro de dados entre dois servidores interligados simultaneamente por múltiplos canais [Raiciu et al., 2011].

O MPTCP introduz mecanismos sofisticados e inovadores de melhoramento do desempenho e da longevidade das conexões TCP, através da utilização simultânea de vários sub-fluxos que terminam em interfaces distintas, e introduz novas formas de gerir o controlo da saturação. Para além destes aspectos, a definição e implementação do protocolo inclui um notável trabalho de engenharia para assegurar que os seus segmentos não são bloqueados pelas *middleboxes* hoje mais comuns na Internet. O MPTCP está disponível para Linux, Android, FreeBSD e iOS desde a versão 7. O RFC 6824 introduz a definição do protocolo.

Para terminarmos esta breve discussão de outros protocolos de transporte alternativos ao UDP e ao TCP, vamos a seguir voltar a analisar os factores que têm contribuído para o seu relativo insucesso, assim como algumas aproximações alternativas ao problema.

10.4 *Middleboxes* e “ossificação” da Internet

O nível rede em geral e a Internet em particular, tal como os temos apresentado até aqui, são infra-estruturas que transportam pacotes de dados entre computadores, sem os modificarem ou bloquearem. A verdade é hoje em dia mais complexa pois no interior da rede, para além dos comutadores de pacotes, existem numerosos equipamentos que intervêm no encaminhamento dos pacotes, podendo bloqueá-los ou modificá-los.

Esses equipamentos podem ser de vários tipos, nomeadamente: *firewalls*, *stateful firewalls*, *deep-packet inspection firewalls*, *network address translation boxes*, *proxies*, ... etc. Os *firewalls* bloqueiam pacotes que não pertencem a origens, destinos, protocolos e portas permitidos dentro da rede que protegem. São equipamentos que no essencial bloqueiam fluxos de pacotes com base nos endereços IP e portas origem / destino.

Os outros tipos de *firewalls* fazem uma análise mais fina dos pacotes e bloqueiam pacotes que não pertencem a conexões previamente estabelecidas, ou que contêm números de sequência impossíveis, ou bloqueiam opções não desejadas dos protocolos, ou ainda que procuram no conteúdo dos pacotes dados que revelam a presença de potenciais vírus ou outras formas de ataque implementadas a nível aplicacional.

As *network address translation (NAT) boxes* são equipamentos, muito comuns nas redes domésticas e nas redes dos operadores celulares, que transformam os endereços origem dos pacotes, permitindo que um ou poucos endereços IP sejam partilhados por um número muito maior de computadores para acesso à Internet. A forma como estes equipamentos funcionam exige o reconhecimento dos protocolos de transporte usados no *payload* dos pacotes IP, o que restringe a utilização de protocolos de transporte aos nossos bem conhecidos TCP e UDP.

Os *proxies* são equipamentos que terminam as conexões noutros computadores diferentes dos originalmente requeridos, e que partem as conexões TCP em diferentes conexões para melhorarem o desempenho.

Este equipamento são hoje em dia muitíssimo numerosos. De acordo com algumas estatísticas recenseadas em [Raiciu et al., 2013] a partir de diferentes estudos, cerca de 90% dos utilizadores domésticos e cerca de 80% dos operadores celulares utilizam equipamentos do tipo NAT. A maioria dos operadores celulares utilizam *firewalls*, assim como muitos computadores individuais e servidores. Todas as redes empresariais utilizam *middleboxes* em número quase tão significativo como o dos comutadores de pacotes, e um número relevante de redes institucionais e de redes de acesso usam *proxies*.

Como resultado desta situação, o protocolo IP já não é o gargalo da pilha de protocolos da Internet, ver a Secção 4.4. Este gargalo compreende actualmente também os protocolos UDP e TCP como está ilustrado na Figura 10.1.

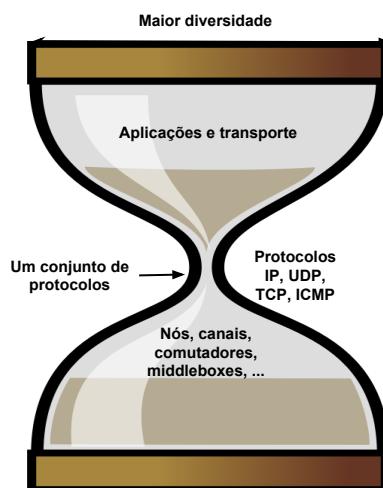


Figura 10.1: Os protocolos IP, UDP e TCP são o novo gargalo da ampulheta; eles separam as aplicações da “diversidade mitigada” da rede

Este fenómeno é em geral designado por “ossificação” da Internet e, como vimos através das funcionalidades que se tentaram introduzir no transporte de dados (múltiplos fluxos, múltiplas semânticas de fiabilidade e ordenação, múltiplas formas de realizar o controlo de saturação, exploração de múltiplas interfaces, *etc.*) e do insucesso da sua adopção, constatamos que estamos numa situação contraditória e que tende para uma grande rigidez de alternativas de transporte e de formas diferentes de suportar as necessidades do desenvolvimento das aplicações.

A visão inicial existente na comunidade que desenvolvia aplicações para a Internet, e que foi transmitida na primeira parte deste livro, é que as aplicações eram desenvol-

vidas usando um conjunto de protocolos de transporte, implementados directamente pelos sistemas de operação. Na verdade, a noção de transporte evoluiu e tornou-se mais rica. Hoje em dia a maioria das aplicações não usa directamente a interface de sockets, mas antes um conjunto de bibliotecas e APIs que fornecem um conjunto de serviços especialmente concebidos para um dado conjunto de aplicações.

Esta nova visão das funcionalidades de transporte permite que as mesmas mascrem a forma como os serviços de transporte são de facto disponibilizados. Muitos serviços de “transporte de dados” são hoje em dia implementados sobre protocolos, inclusive de nível aplicacional [Raiciu et al., 2013]. Para alguns, o protocolo HTTP é praticamente o único meio de transporte de informação universal actualmente disponível.

10.5 Propostas para aprofundamento

Ao longo deste capítulo foram indicadas várias referências bibliográficas que permitem o aprofundamento do estudo das diversas facetas nele tratadas. Segue-se um conjunto de propostas de trabalhos de aprofundamento dessas facetas.

1. Realize, sob a forma de um relatório, um estudo mais aprofundado do protocolo DCCP utilizando as referências que foram indicadas acima.
2. Realize, sob a forma de um relatório, um estudo mais aprofundado do protocolo SCTP. Utilize a referência [Budzisz et al., 2012] como um ponto de partida suplementar para além das que foram indicadas acima.
3. Realize, sob a forma de um relatório, um estudo mais aprofundado do protocolo MPTCP utilizando as referências que foram indicadas acima.
4. Realize, sob a forma de um relatório, um estudo mais aprofundado sobre a evolução recente dos protocolos de transporte fiável de informação partindo da referência [Raiciu et al., 2013].

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 infindável 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-lo 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 humanos, 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, *etc.*). 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 presidente joaninha	Nome de uma pessoa Nome de uma função (indireção) Alcunha (alias) de uma pessoa na agenda de outra pessoa
<code>www.fct.unl.pt</code> <code>/sources/search.java</code>	Nome hierárquico de um servidor Nome hierárquico de um ficheiro
IBM	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.

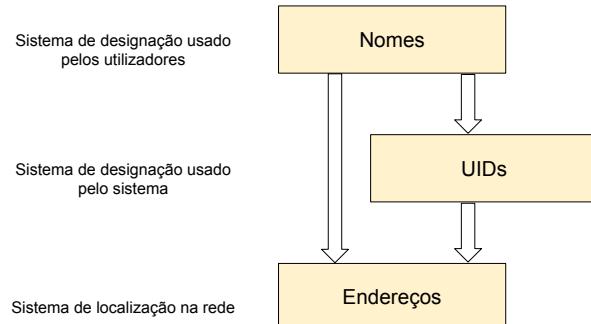


Figura 11.1: Hierarquia de sistemas de designação

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 Stanford, 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

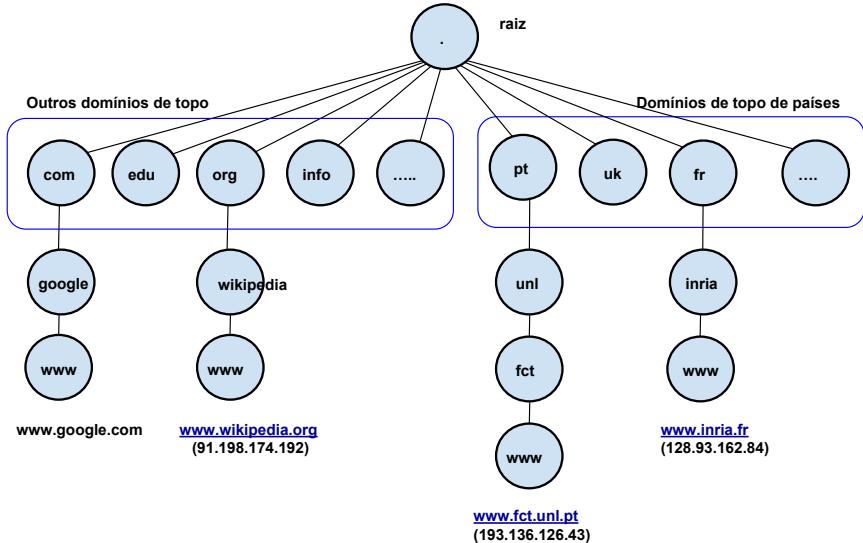


Figura 11.2: Hierarquia de designação do sistema DNS

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

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

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 gere 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 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 registo 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 registo 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.

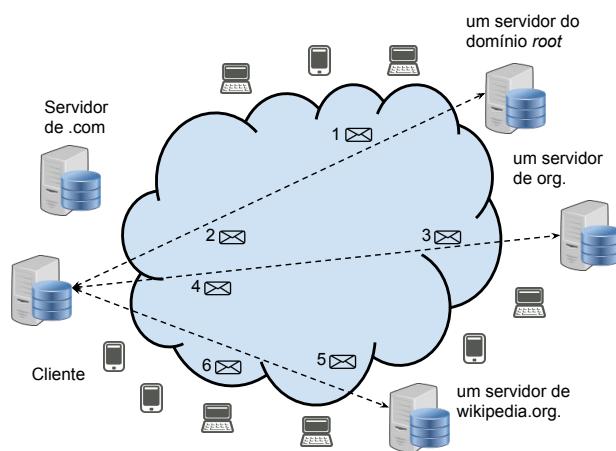


Figura 11.3: Progresso da consulta `[name: www.wikipedia.org, type: A]` iterativamente, começando num servidor da zona `root`. As mensagens ímpares são pedidos e as pares as respectivas respostas. Todas as mensagens de consulta são idênticas.

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].

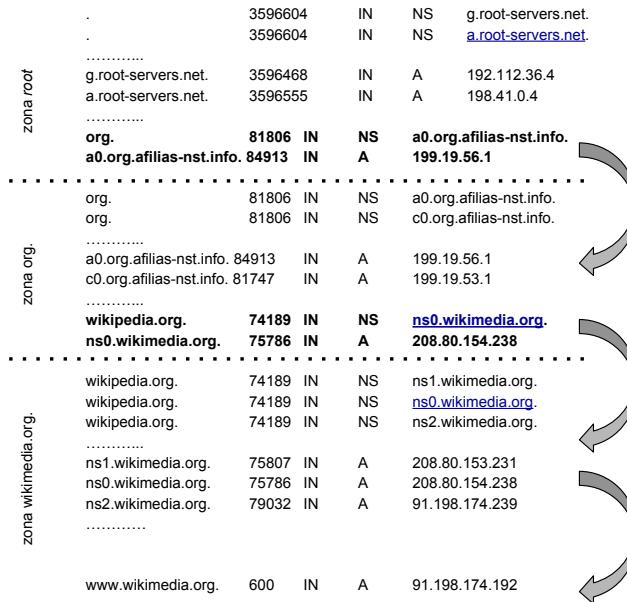


Figura 11.4: Progresso da consulta iterativa [name: www.wikipedia.org, type: A] através das zonas da base de dados global do DNS (a negrito um dos *glue records* de ligação entre zonas)

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 registos 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.

²De 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.

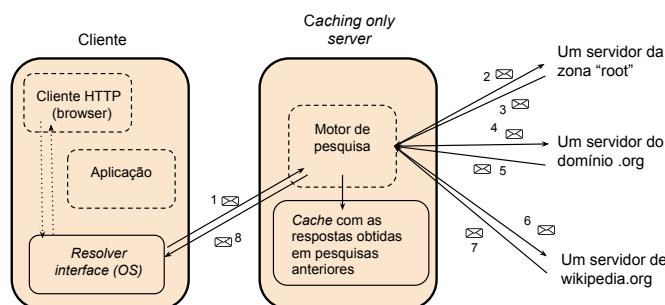


Figura 11.6: Progresso da consulta [name: www.wikipedia.org, type: A] recursiva enviada a um servidor *caching only*. O servidor desconhece a resposta e executa uma consulta iterativa.

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.

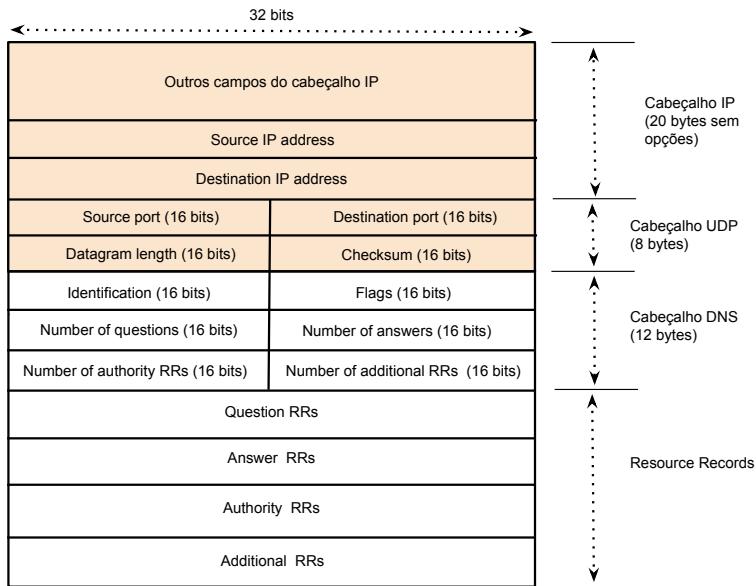


Figura 11.7: Mensagem de pedido e resposta ao DNS transportada pelo protocolo UDP

```
$ 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: ...](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 lerditão 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 registrar 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, diretas 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 registos 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 os 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 W20NS.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 propriedade de vários fabricantes de software.

Às 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.

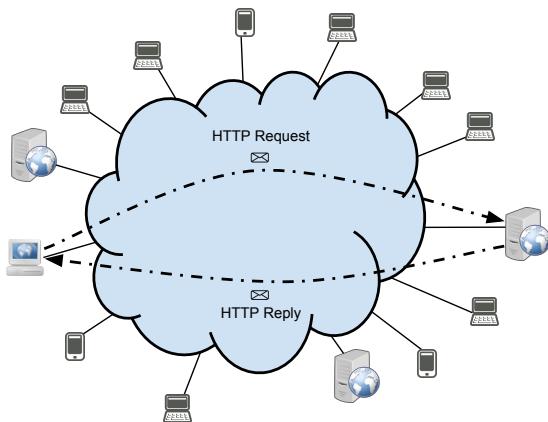


Figura 12.2: Obtenção pelo cliente de um objecto do servidor através do protocolo HTTP

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*

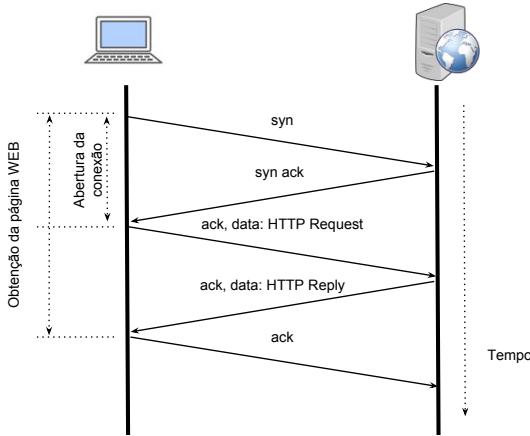


Figura 12.3: O padrão [abertura da conexão, envio do pedido, obtenção da resposta e fecho da conexão] para obtenção de cada objecto via HTTP

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

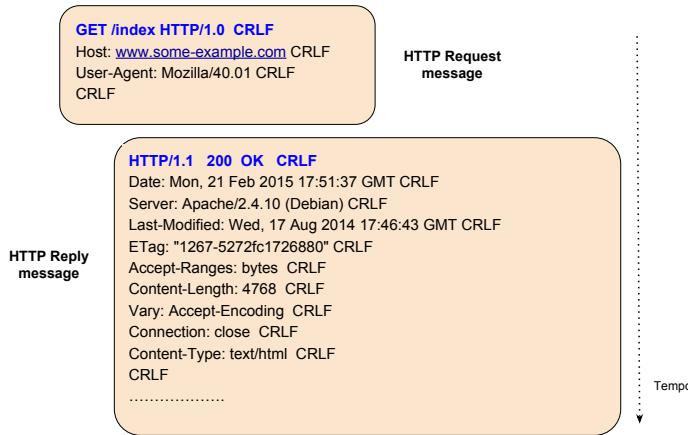


Figura 12.4: Exemplo de mensagens de pedido e resposta do protocolo HTTP. Na figura a abreviatura CRLF denota o par ordenado de caracteres: **carriage return, linefeed**.

de vista da elaboração de estatísticas. O *header field* `Host` permite ao cliente indicar a que servidor exato 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 breves 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/4.0
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

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

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.

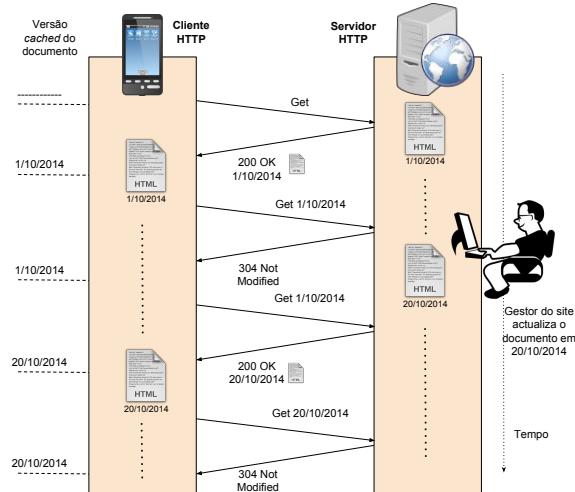


Figura 12.5: Gestão de uma *cache* num cliente HTTP com recurso ao *header field* **If-Modified-Since**

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, poupan-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*.

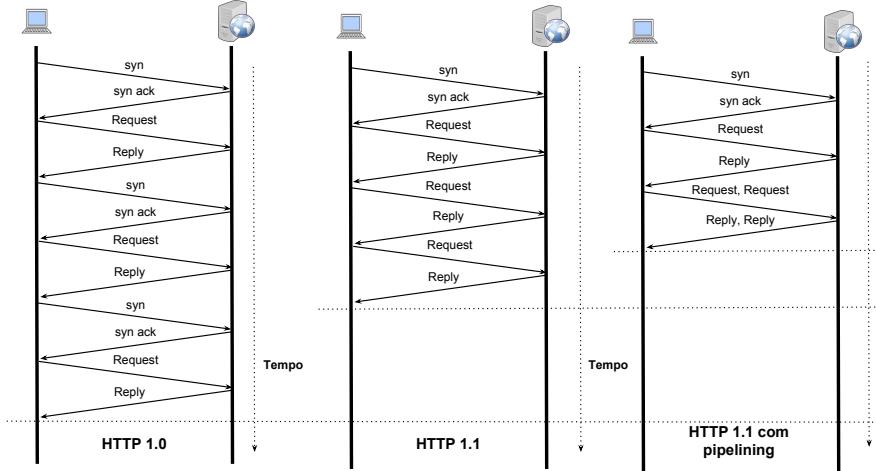


Figura 12.6: Tempo para obter uma página Web e dois objectos nela referenciados com diferentes versões do protocolo HTTP. O diagrama ignora o ganho suplementar devido a só existir, potencialmente, uma única fase *slow start* da conexão e considera, por hipótese, que cada pedido e resposta são executados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

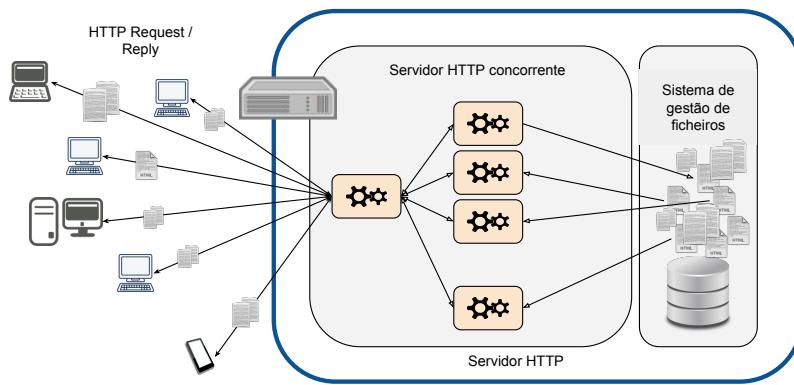


Figura 12.7: Servidor HTTP processando múltiplas conexões HTTP em paralelo através de diversos *threads* de execução

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

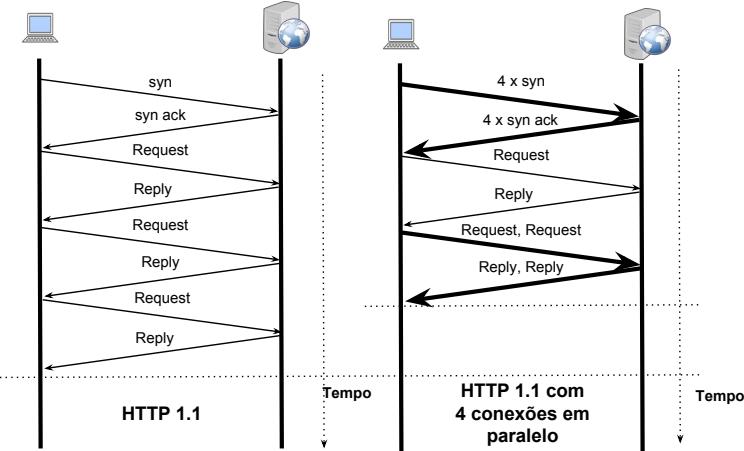


Figura 12.8: Tempo para obter uma página Web e dois objectos nela referenciados com uma conexão TCP e sem *pipelining*, e com 4 conexões TCP paralelas. O diagrama considera, por hipótese, que cada pedido e resposta é executado num único RTT.

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

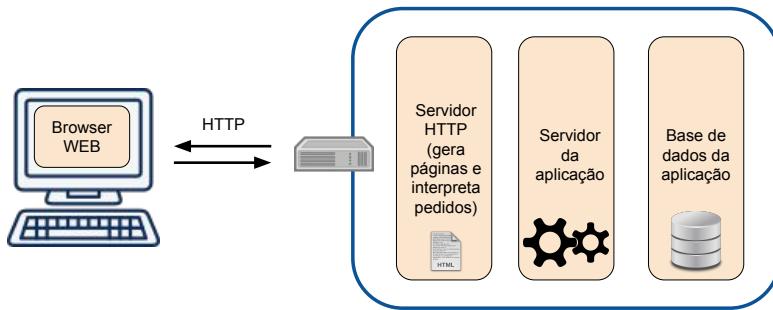


Figura 12.9: Arquitectura genérica de uma aplicação acessível via a Web (parte servidor)

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.only4dummyspies.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.only4dummyspies.com
Authorization: Basic aHR0cHdhGNoOmY=
```

onde “`aHR0cHdhGNoOmY=`” 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 embbebido 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 interacções 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

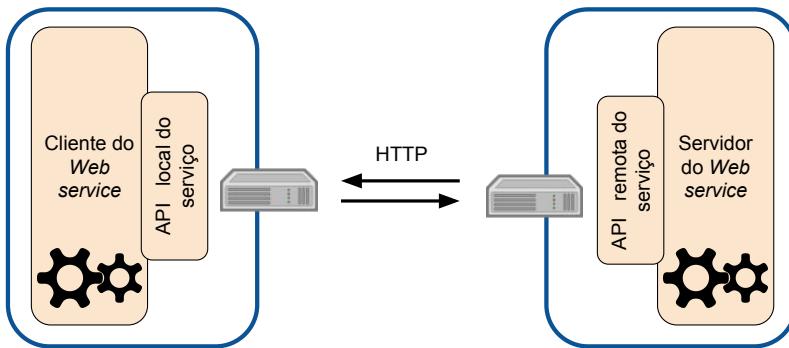


Figura 12.10: Um cliente usa os *Web services* exportados por um servidor através de uma API local que representa para o código do cliente a interface do servidor.

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 tutoriais 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.

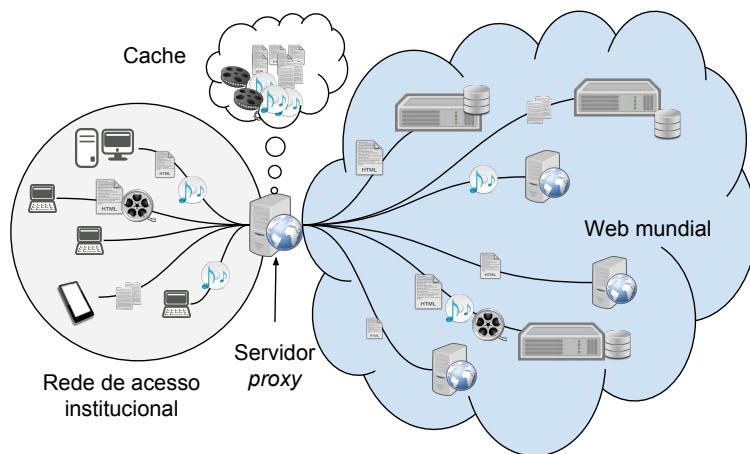


Figura 13.1: Acesso à web via um servidor *proxy* da instituição

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 páginas 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 às aquelas 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.

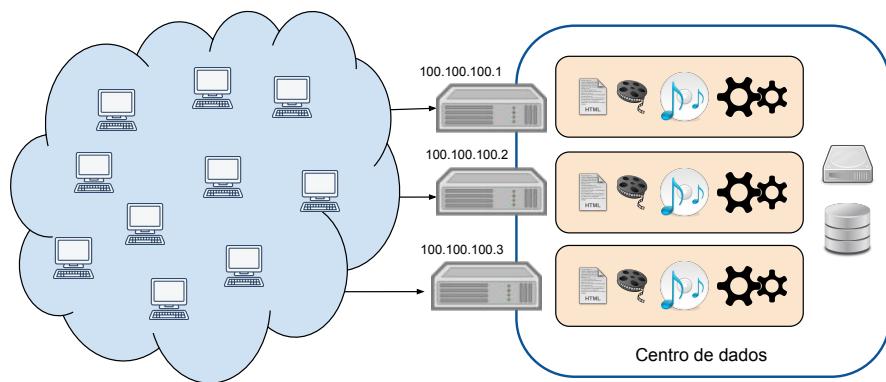


Figura 13.2: Distribuição de carga usando vários servidores com endereços IP distintos

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. Nesses 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 equilibrem 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.

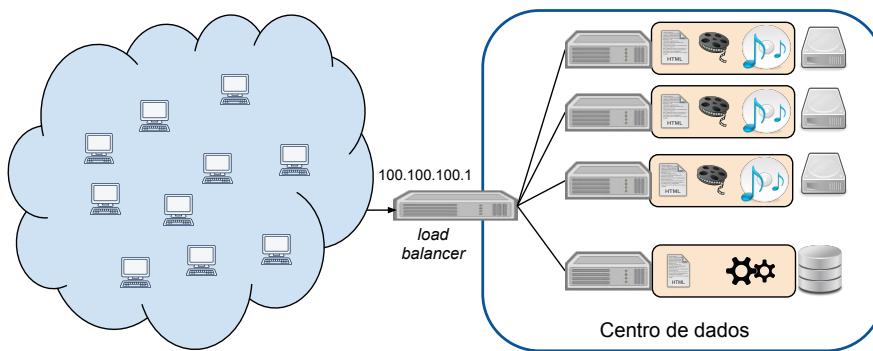


Figura 13.3: Distribuição de carga com um único endereço e um *load balancer*

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 servir 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.

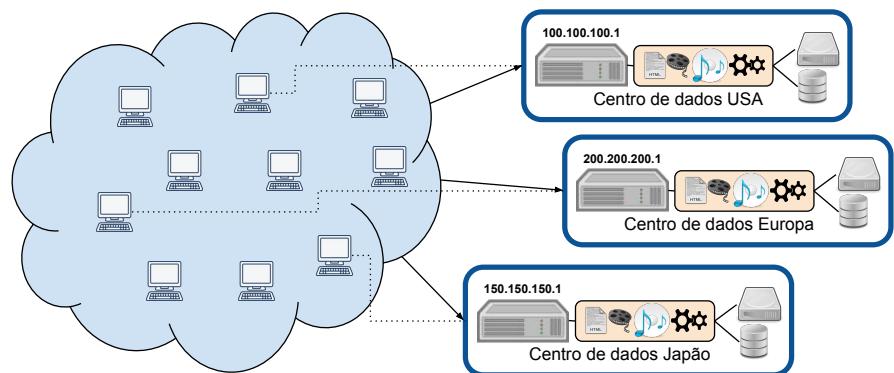


Figura 13.4: Distribuição de carga com diferentes centros de dados em diferentes regiões geográficas

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, selecção 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 contenham 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.

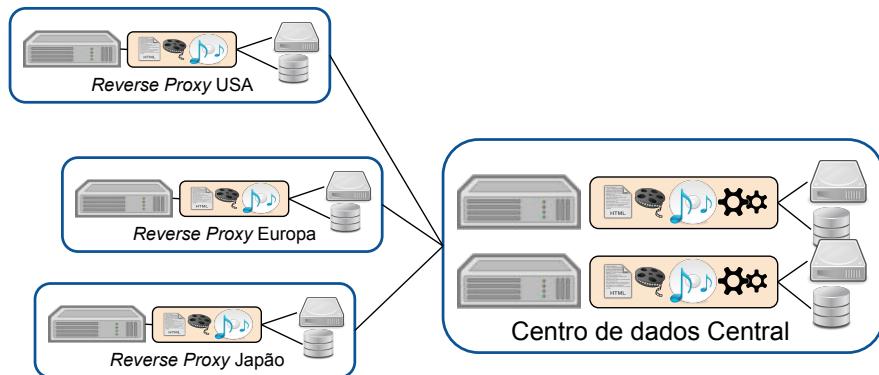


Figura 13.5: Distribuição de carga com recurso a *proxies* inversos em diferentes regiões geográficas e um centro de dados central

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 (proteção 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

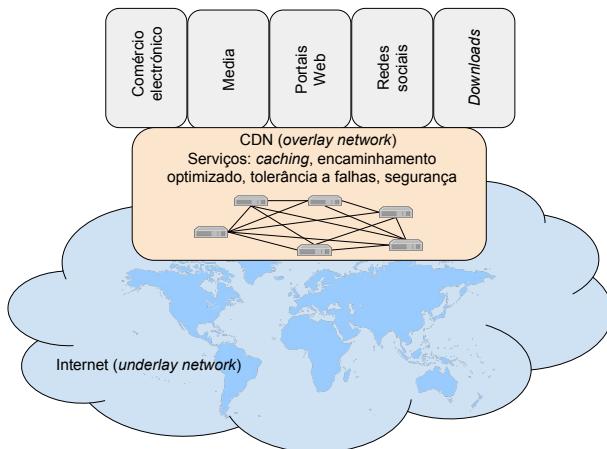


Figura 13.6: Posicionamento das CDNs entre a Internet e os serviços oferecidos aos clientes

(<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.

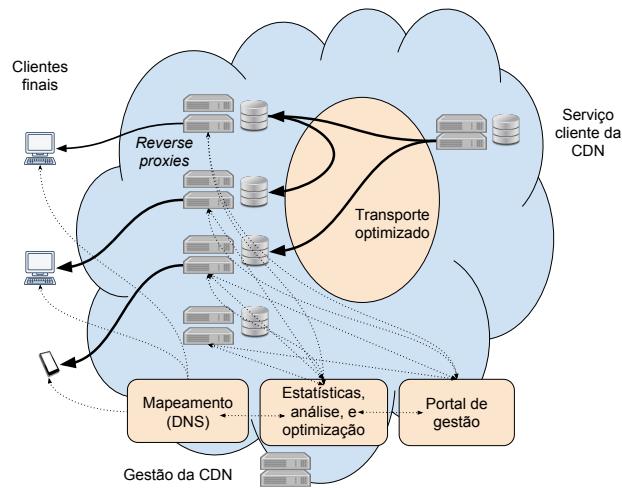


Figura 13.7: Estrutura interna de uma CDN com os *proxies* inversos em diferentes regiões geográficas e os servidores centrais de controlo e gestão. As linhas a negrito representam os fluxos de dados. As linhas a tracejado os fluxos de controlo. Adaptado de [Nygren et al., 2010].

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.

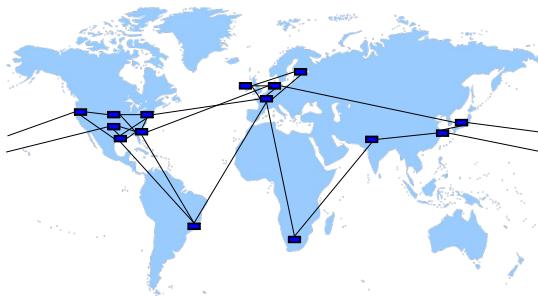


Figura 13.8: CDN hierárquica constituída por um conjunto de centros de dados regionais, complementados com POP-CDNs mais próximos dos clientes

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 Gulosoos 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 Gulosoos Associados.

Assim o URL dos objectos do serviço da empresa Gulosoos 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 à 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

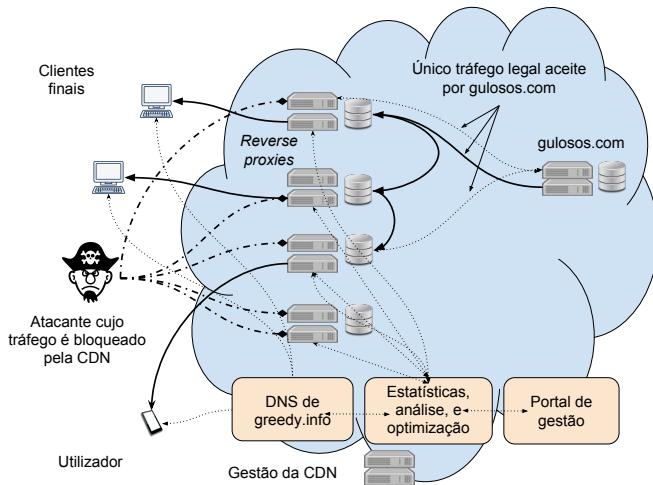


Figura 13.9: Utilização da CDN para providenciar serviços de segurança

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*.

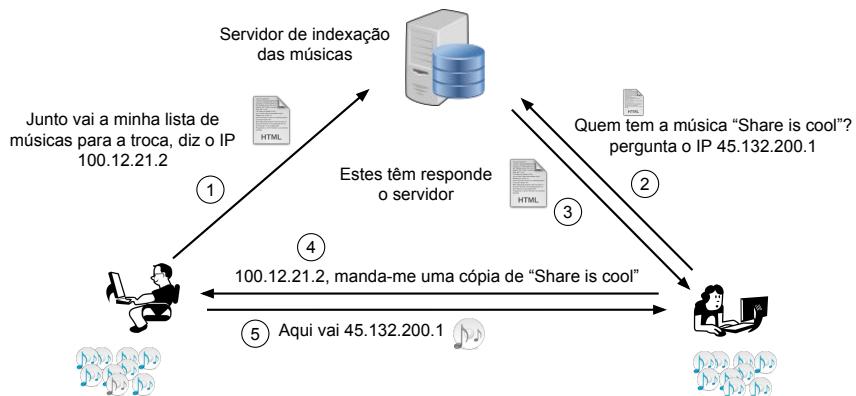


Figura 13.10: Um servidor de indexação de conteúdos proporciona a troca directa dos mesmos entre os utilizadores

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 $4.10^{15}/10^8 = 4.10^5 = 400.000$ 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.

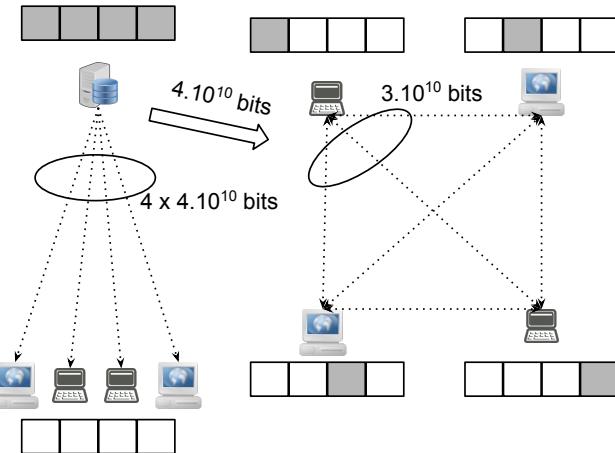


Figura 13.11: Duas alternativas para distribuir um ficheiro por 4 utilizadores

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 colaboraram 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 registrar 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.

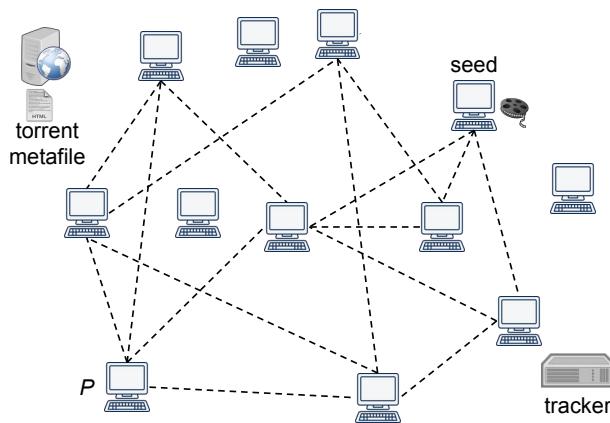


Figura 13.12: Uma torrente em funcionamento. As linhas representam parceiros a trocarem *pieces*.

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 atractivo 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 aí 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 *chok / 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 Bit-Torrent. 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 registado 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 $k = digest(\text{.torrent})$, 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.*, $n = digest(\text{endereço IP} + \#ram + \dots)$). 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 $\text{sucessor}(k)$, 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, $\text{sucessor}(k)$ é o identificador do mais próximo nó que no anel está colocado a seguir à chave k . Esta noção tem importância pois, por convenção, $\text{sucessor}(k)$ é o identificador do nó responsável por memorizar o valor associado à chave k . Por exemplo, na Figura 13.13, $702 = \text{sucessor}(640)$ e $702 = \text{succ}(580)$.

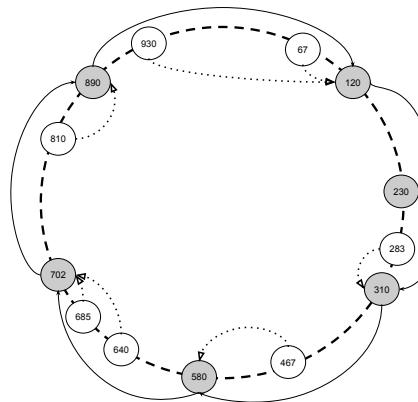


Figura 13.13: Organização lógica de um conjunto de identificadores em anel. Os nós a cinzento representam nós físicos da rede. Os restantes representam chaves de conteúdo. As setas a negro representam as ligações entre nós físicos. As setas a tracejado representam a ligação da chave k ao nó $\text{sucessor}(k)$, 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 $\text{succ}(n)$ e obtém o endereço IP do nó do anel com esse identificador, assim como o endereço IP do nó com identificador $\text{pred}(\text{succ}(n))$. 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 ($\text{succ}(\text{succ}(n))$). Periodicamente o nó n testa a conectividade até $\text{succ}(n)$ e se esta se quebrar, liga-se a $\text{succ}(\text{succ}(n))$.

Como cada nó de identificador n memoriza os valores associados às chaves k tal que $k \in [\text{pred}(n), n]$, 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 $\text{sucessor}(k)$ 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 $\text{successor}(k)$.

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 $\text{successor}(k)$ (e.g., $\text{pred}(\text{successor}(k))$ e $\text{succ}(\text{successor}(k))$), ou usando outras funções de *digest*, i.e., diferentes instâncias da função $\text{sucessor}(k)$, 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

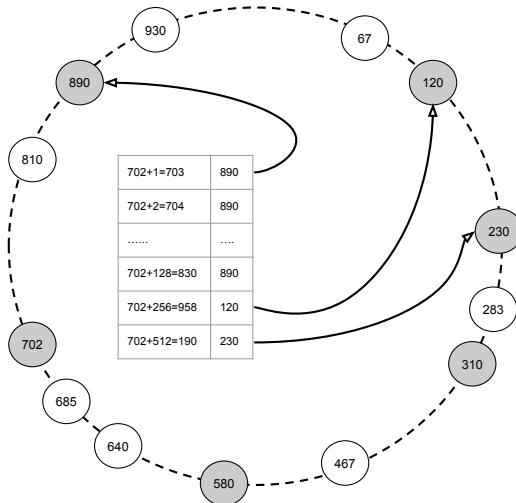


Figura 13.14: Um anel Chord e a *finger table* do nó 702. O domínio das chaves é de 0 a 1023 e as somas são realizadas módulo 1024.

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 redireção 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

Proxie 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.

Proxie 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 geo-localização 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 fração 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 necesita 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 repercussão 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 capacidade dos diferentes participantes para: *download* 20 Mbps e *upload* para 1 Mbps
 - Modificar as capacidade dos diferentes participantes para: *download* 10 Mbps e *upload* para 2 Mbps
 - Modificar as capacidade dos diferentes participantes para: *download* 1 Mbps e *upload* para 10 Mbps
 - Modificar as capacidade dos diferentes participantes para: *download* 5 Mbps e *upload* para 5 Mbps
 - Modificar as capacidade 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.

Parte IV

Redes de pacotes

Nos vários capítulos que antecedem esta parte referimo-nos várias vezes a uma infra-estrutura, que designámos como a “rede propriamente dita”, que tem por função receber pacotes de dados dos emissores, encaminhá-los até ao seu destino e finalmente entregá-los aos destinatários finais. Tradicionalmente, esta parte de uma rede de computadores é designado por **nível rede** nos modelos de camadas, ver o Capítulo 4.

Esta infra-estrutura é formada por um conjunto de nós de comutação de pacotes (comutadores de pacotes), ver o Capítulo 3, interligados por canais de dados, ver o Capítulo 2. Os computadores ligam-se entre si ou aos comutadores de pacotes, também através de canais de dados, e trocam pacotes directamente entre si (*end-to-end*) usando os serviços da rede. O coração da Internet é uma grande rede de pacotes, formado por inúmeras redes interligadas, e actuando em coordenação, de tal forma que essa nuvem de redes interligadas actua como uma rede logicamente única.

Compete ao software residente nos computadores aos níveis transporte e aplicação implementar os serviços distribuídos que os utilizadores utilizam, com base nos serviços do nível rede, como está ilustrado na Figura 13.15. Quando um computador cliente interage com um computador servidor para lhe pedir uma página Web, os dois dialogam “directamente” usando de forma transparente os serviços do nível rede.

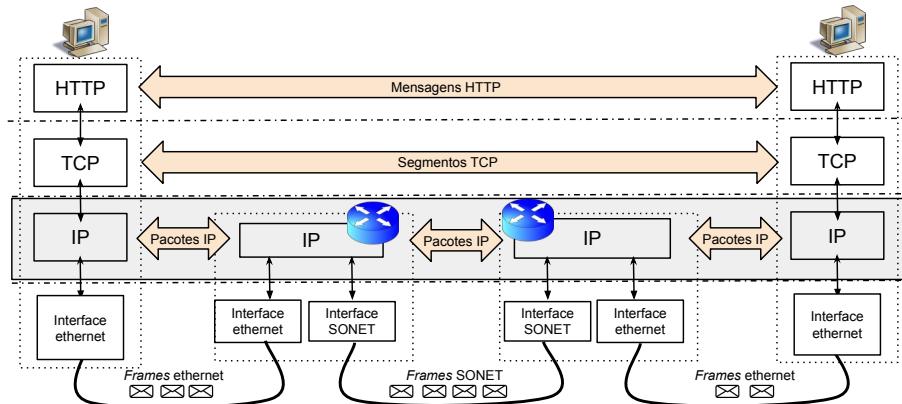


Figura 13.15: O nível rede (a sombra) utiliza canais de dados e comutadores de pacotes para providenciar os seus serviços aos níveis superiores

Nesta parte do livro, o termo rede de pacotes, ou simplesmente rede, vai ser usado para designar essa infra-estrutura de encaminhamento de pacotes da origem até ao destino. Uma rede pode ser muito simples, baseada num único canal multi-ponto que interliga vários computadores, pode ser mais complexa, sendo constituída por uma malha de canais e nós de comutação de pacotes, ou pode ser ainda mais complexa, sendo formada por uma interligação de redes diferentes, uma rede de redes ou um *internetwork* na terminologia em língua inglesa, e assim sucessivamente. O termo é usado de forma recursiva em diferentes situações, que o leitor terá de distinguir pelo contexto da discussão.

Desenhar uma rede envolve a solução de vários problemas. Primeiro é necessário definir um esquema de endereçamento que permita indicar a origem e o destino dos pacotes. No caso das redes muito simples, isso é simples de resolver pois o único requisito é que os endereços sejam diferentes uns dos outros. Se a rede é mais complexa, o seu funcionamento interno pode impor a necessidade de alguma abstração de detalhes através da utilização de uma hierarquia de endereços. Diferentes partes da rede olharão para diferentes partes do endereço, o que é sempre uma boa medida para promover a escalabilidade. Quando a rede é uma interligação de redes, com uma escala muito

grande como a Internet, o problema do endereçamento pode tornar-se ainda mais complexo.

Um segundo problema que a rede tem de resolver é determinar caminhos para encaminhar pacotes com sucesso até ao destino. Este problema também pode ter soluções simples, mas no caso geral, é bastante mais complexo pois a escolha do caminho tem de satisfazer critérios de correção; *i.e.*, os pacotes chegam aos destino previsto, de eficiência; *i.e.*, de alguma forma tenta-se optimizar o funcionamento do sistema; e tem-se também de encontrar formas de lidar com as falhas das componentes da rede (canais e nós de comutação) e eventualmente também com as perdas de pacotes. Finalmente, a rede tem de ser monitorada, gerida, e continuamente adaptada a novos utilizadores e a novos requisitos.

Uma rede viária é uma boa analogia para o nível rede. No mesmo continente, é possível iniciar uma viagem de automóvel numa rua de uma cidade num dado país, e terminá-la numa aldeia de outro país. Pelo meio o automóvel atravessou ruas numa cidade, entrou em estradas nacionais, passou para auto-estradas, atravessou fronteiras, passou por estradas municipais e finalmente chegou à aldeia. Para chegar com êxito ao destino é necessário conhecer o seu endereço. Para isso usam-se convenções comuns nos diferentes países. Todas as estradas usadas permitiram a circulação daquele automóvel, e finalmente, nos cruzamentos de estradas, houve a capacidade de tomar as decisões correctas.

Do ponto de vista do condutor do automóvel o processo é relativamente simples. No entanto, do ponto de vista da construção e gestão da rede viária o problema é muito mais complexo. Numa pequena cidade ou numa vila, a rede viária é relativamente fácil de construir e gerir. Trata-se de uma pequena rede. Do ponto de vista das grandes cidades e suas interligações, o problema é mais complexo. Por um lado temos redes viárias muito densas e carregadas, e por outro temos interligações dessas redes, que são elas próprias outras redes, ou seja redes de trânsito, pois as mesmas não têm edifícios, origem ou destino de trajectos, apenas existem para que se circule nelas.

Por isso, o estudo do nível rede também é muito multi-facetado pois existem diferentes tipos de redes com diferentes tipos de requisitos, e existe também uma interligação de todas essas redes. Os capítulos que formam esta parte do livro são vários, nomeadamente:

Capítulo 14 introduz os canais baseados em difusão, ou multi-ponto, e como estes podem ser usados para construir redes simples que permitem a muitos computadores trocarem pacotes directamente entre si usando um único canal. Neste tipo de redes não é necessário fazer o encaminhamento de pacotes recorrendo a nós de comutação de pacotes pois um único canal é suficiente para assegurar a comunicação extremo-a-extremo. Na analogia com as redes viárias estes canais são como ruas de cidades.

Capítulo 15 discute as formas mais simples de fazer encaminhamento, nomeadamente recorrendo a técnicas de inundação (*flooding*). Apesar de estas técnicas parecerem demasiado ingénugas é possível aproveitar todas as suas vantagens em contextos particulares, onde também é fácil contornar os seus inconvenientes. Em redes com um âmbito geográfico limitado e sem grandes constrangimentos de capacidade, está técnica é popular e revela-se bastante adequada. Na analogia com a rede viária estas redes são como localidades de pequena ou média dimensão.

Capítulo 16 introduz técnicas, algoritmos e protocolos de encaminhamento para redes mais complexas, mas baseados numa solução de compromisso relativamente simples: os pacote são encaminhados pelo caminho mais curto. Esta maneira de conceber o encaminhamento é um bom compromisso para qualquer tipo de rede, desde que a mesma esteja bem dimensionada para o tráfego injectado e

não existam canais saturados. Na analogia com a rede viária estas redes são como regiões de média dimensão ou como pequenos países.

Capítulo 17 discute as formas de endereçamento mais adequadas a redes simples, sem necessidade de administração de endereços, assim como as formas de endereçamento que são requeridas em redes de grande dimensão e hierárquicas. Discute também diversas facetas da relação entre o endereçamento e a liberdade de movimentos dos clientes entre redes e fornecedores de serviço. Depois o capítulo introduz o protocolo IP e a forma como este é usado para realizar o encaminhamento dentro e entre as redes da periferia da Internet.

Capítulo 18 introduz como é realizado o encaminhamento entre as diferentes redes que formam a Internet, *i.e.*, como são escolhidos os caminhos nas auto-estradas de interligação das diferentes redes que a formam. Por um lado, a este nível, os requisitos do encaminhamento são complexos pois é necessário satisfazer critérios de optimalidade mas também critérios relacionados com questões comerciais (quem paga a quem os serviços de transporte de pacotes) e políticos (será que estes pacotes podem atravessar esta rede?). Estes factores requerem algoritmos e um protocolo de encaminhamento especial, o protocolo BGP (*Border Gateway Protocol*). Na analogia com a rede viária estas redes são como as redes de auto-estradas, que eventualmente cruzam diversos países, e interligam regiões densamente povoadas.

Capítulo 14

Redes baseadas em canais de difusão

Metcalfe's Law: “The value of a network grows as the square of the number of its users.”

– Autor: Robert Metcalfe, inventor of the Ethernet network

Os canais baseados em difusão (*broadcasting*), ou multi-ponto, permitem que muitos computadores partilhando um meio de comunicação comum possam comunicar directamente uns com os outros; por essa razão permitem realizar redes muito simples como ilustra a Figura 14.1. Com efeito, se $n > 2$ computadores estiverem directamente ligados por um canal deste tipo, n computadores podem comunicar directamente sem necessidade de comutadores de pacotes, ver o Capítulo 3. Quando um computador transmite uma mensagem, todos os outros ligados ao canal vêm o seu conteúdo. No entanto, só o destinatário, i.e., o computador com o endereço do destino da mensagem, a guarda. Adicionalmente, estes canais suportam directamente a comunicação de um para todos, isto é, quando um emissor transmite uma mensagem para o endereço de difusão (*broadcasting*), esta é recebida por todos os outros.

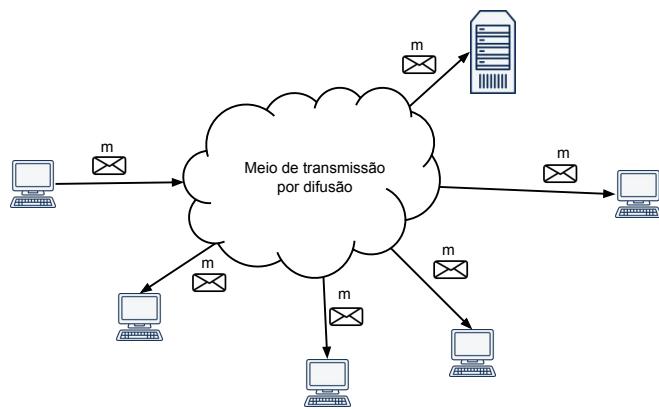


Figura 14.1: Transmissão de uma mensagem por um canal baseado em difusão através de um meio de comunicação partilhado

Como foi referido no Capítulo 2, um canal diz-se *simplex*, se só permite transmitir

informação num só sentido e diz-se *full-duplex* se permite a transmissão em simultâneo em todos os sentidos; um canal ponto-a-ponto *full-duplex* é equivalente a dois canais *simplex*, um em cada sentido. Um canal *half-duplex* é um canal que liga vários interlocutores mas em que só um pode emitir em cada momento. Os canais que iremos estudar neste capítulo são canais *half-duplex*.

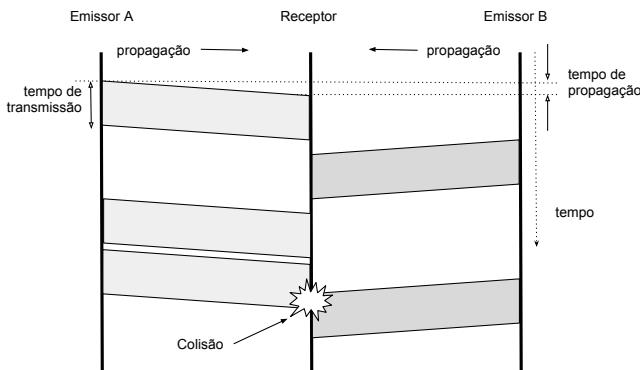


Figura 14.2: Emissores ligados ao mesmo meio de comunicação - se as emissões tiverem lugar em momentos diferentes, não há colisão; senão os *frames* colidem e o receptor recebe-os com erros

Com efeito, ver a Figura 14.2, como há vários emissores que partilham o mesmo meio de comunicação (o *transmission media*), estes têm de transmitir *frames* em momentos diferentes, sob pena de os sinais emitidos por cada um se cruzarem num potencial receptor, que será incapaz de reconhecê-los. Esta mistura dos sinais que suportam a codificação de cada um dos *frames* chama-se uma **colisão** (*collision*) e inutiliza todos os *frames* que colidam.

Assim, estes canais necessitam de mecanismos e protocolos que impeçam acessos simultâneos ao meio de comunicação. Estes são designados protocolos de **controlo de acesso ao meio** e são abreviados pela sigla em língua inglesa **MAC** – *Medium Access Control*. No essencial, é algo semelhante ao que é necessário usar durante a realização de uma reunião para evitar que os presentes falem todos ao mesmo tempo, i.e., trata-se de um mecanismo de regulação do direito a usar a palavra.

Os meios de transmissão usados pelos canais de difusão são vários. O mais comum é a atmosfera pois, quando não se usam antenas direcionadas, este meio permite que um sinal emitido seja recebido simultaneamente por muitos receptores. Alguns exemplos de canais deste tipo, já referidos no Capítulo 2, são canais de dados via satélite, assim como os banais canais Wi-Fi. No entanto, também existem canais baseados em difusão cujo meio de comunicação é baseado em meios guiados como cabos de cobre ou fibras ópticas.

A probabilidade de existirem colisões é muito dependente de vários factores. Estamos a admitir que os canais que nos interessam são canais de dados e portanto os emissores geram pacotes de forma muito irregular e não necessariamente coordenada, podendo alternar entre períodos sem actividade e períodos com muita actividade. Um *frame* com 1 Kbyte tem cerca de 8.000 bits e é transmitido em aproximadamente $80 \mu\text{s}$ a 100 Mbps e em aproximadamente $8 \mu\text{s}$ a 1 Gbps. Logo, se o tráfego for pouco intenso e todas as interfaces transmitirem de forma independente e não sincronizada (o que nem sempre é verdade), a probabilidade de colisões é baixa. No entanto, se o tráfego for intenso a probabilidade de colisões já é mais elevada.

Mesmo não havendo colisões, qual a probabilidade de os *frames* serem recebidos

com erros? No Capítulo 2 vimos que nos canais guiados, e sobretudo nas fibras ópticas, essa probabilidade é pequena e por isso deixa-se aos níveis superiores a sua correcção (*e.g.*, ao protocolo TCP). No entanto, nos canais sem fios essa probabilidade é muito elevada e provavelmente é melhor tentar corrigi-los logo ao nível de cada canal, quer através de códigos de correcção de erros, quer através de retransmissão. A razão é que a deteção dos erros ao nível extremo a extremo leva muitas vezes centenas de milissegundos enquanto que ao nível do canal pode levar apenas alguns micro segundos.

Em todos estes canais os receptores apercebem-se através dos códigos de controlo de erros da presença de colisões pois recebem sinais de mais do que um emissor misturados. A mesma questão se pode colocar com respeito ao emissor: será que este detecta a colisão? Para ser capaz de o fazer tem de ser capaz de comparar o sinal que emite com o recebido. Se houver diferenças significativas, então houve uma colisão. Nos canais sem fios, o sinal degrada-se exponencialmente com a distância percorrida e os obstáculos atravessados. Nestes canais os emissores não detectam as colisões pois só se “ouvem a eles próprios”. Com efeito, o sinal emitido pode ser centenas de milhares de vezes mais potente que o recebido.

Nos canais de difusão baseados em fios, os emissores conseguem detectar as colisões pois o sinal não se degrada de forma significativa a distâncias curtas (ou mesmo nas longas no caso das fibras ópticas). Assim, nestes canais é possível ao emissor detectar as colisões. No entanto, essa detecção está dependente do tempo de propagação do sinal pois, se o tempo de emissão for inferior ao tempo de propagação, será mais difícil detectar colisões.

Com efeito, se um canal de difusão com fios tiver 100 metros de comprimento, o tempo de propagação é de cerca de $100 \times 1/2.10^{-8} \approx 0,5 \mu\text{s}$ (micro segundos) visto que o sinal se propaga a cerca de 200.000 Km por segundo num meio de comunicação guiado, ver o Capítulo 2. No entanto, 10 bytes são transmitidos em $80 \times 10^{-8} \approx 0,08 \mu\text{s}$ a 1 Gbps, o que queria dizer que neste caso a colisão poderia não ser detectada pelo emissor visto que os dois sinais não se cruzariam simultaneamente na sua interface.

Os canais baseados em difusão pela atmosfera são muitos e variados pois são usados para difusão da rádio e televisão, nas redes de telemóveis e para comunicação de dados entre *smartphones*, nas redes Wi-Fi, nas redes de sensores, nos automóveis, *etc.* Existem também canais baseados em difusão que usam cabos coaxiais, fibras ópticas e satélites. A gama de soluções usadas para regular o acesso ao meio de comunicação é muito variada e ultrapassa claramente o âmbito deste livro. Neste capítulo vamos apenas analisar uma classe de canais deste tipo que usam mecanismos baseados em aleatoriedade, e que foram desenvolvidos com o objectivo principal de permitirem a comunicação de dados entre computadores.

O capítulo começa por discutir os requisitos que os protocolos MAC para comunicação de dados têm de satisfazer, delimita o tipo de canais que vamos analisar, e analisa os mecanismos que podem ser usados para coordenar o acesso ao meio de comunicação. Depois desta introdução, são apresentadas duas tecnologias usadas na maioria dos computadores e *smartphones* actuais, as tecnologias Ethernet (normas IEEE 802.3) e Wi-Fi (normas IEEE 802.11).

Os canais baseados em difusão funcionam necessariamente no modo *half-duplex* pois se várias interfaces emitem simultaneamente, sobre um mesmo meio de comunicação partilhado, dá-se uma **colisão** dos sinais que provoca a recepção de *frames* com erros e que por isso ficam inutilizados e têm de ser postos de lado, *i.e.*, ignorados. Para resolver este problema é necessário um mecanismo de **controlo de acesso ao meio**, que é abreviado pela sigla em língua inglesa **MAC** – *Medium Access Control*.

Existem muitas tecnologias de canais *half-duplex* baseados em difusão que utilizam vários tipos de meios de comunicação como a atmosfera ou meios guiados (cabos de

cobre ou de fibra óptica). Nos canais sem fios, ao contrário dos canais baseados em meios guiados, a probabilidade de erros é muito elevada, e pode não ser possível um emissor detectar se o canal está ou não ocupado, ou se houve ou não colisão.

14.1 Requisitos e possíveis soluções

Os protocolos MAC devem satisfazer requisitos comuns aos mecanismos flexíveis de multiplexagem de canais de dados de forma flexível. Assim, dado um canal multi-ponto com a capacidade ou débito D , ao qual estão ligadas várias interfaces de comunicação, então:

1. se só uma interface emissora pretende emitir mensagens, então esta deve poder utilizar a totalidade da capacidade do canal, *i.e.*, o débito D ;
2. se n interfaces pretendem emitir mensagens, então cada uma deve poder utilizar em média a fracção D/n da capacidade do canal;
3. o protocolo MAC deve ter um *overhead* baixo, e não consumir ele próprio uma fracção significativa da capacidade D ;
4. o protocolo MAC deve ser simples e escalável, ou seja, adaptar-se facilmente a um número variável de interfaces; e por fim,
5. opcionalmente, o protocolo MAC deve suportar um sistema de prioridades entre as interfaces.

Classes de mecanismos e de protocolos MAC

Existem duas grandes famílias de mecanismos e protocolos MAC: os centralizados e os distribuídos. Os protocolos centralizados são caracterizados por se basearem num árbitro central que “dá a palavra aos diferentes participantes” (como a mesa de uma assembleia). Os distribuídos baseiam-se em coordenação distribuída, isto é, o canal é ele próprio usado para coordenar (“dar a vez”) aos diferentes participantes, que nesse caso estão numa posição semelhante uns aos outros.

De forma geral, a indústria de computadores, ao contrário da indústria de telecomunicações, optou sobretudo por soluções distribuídas, em que as funções de arbitragem existem em todas as interfaces ligadas ao canal. Inicialmente foram comercializadas duas abordagens distribuídas distintas por esta industria. Uma mais complexa, baptizada “Token Ring”, baseada num mecanismo de coordenação distribuída que afecta o direito de emitir “à vez” em carrossel (*round robin*), mas que foi comercialmente abandonada. E outra, baseada na selecção aleatória da próxima interface a emitir, que foi e continua a ser muito utilizada.

No resto deste capítulo vamos estudar dois exemplos de canais baseados em protocolos MAC distribuídos com abordagens de afectação aleatória do canal: Ethernet clássica e Wi-Fi. Começaremos, no entanto, por abordar primeiro diversos mecanismos que são usados nesses protocolos. O tratamento apresentado é simplificado e serve apenas para perceber de forma qualitativa os mecanismos que vão ser utilizados nas duas tecnologias apresentadas a seguir. Na Secção 14.4 serão indicadas referências que apresentam tratamentos mais aprofundados e completos do tema.

Mecanismos usados pelos protocolos MAC aleatórios distribuídos

Os protocolos MAC aleatórios distribuídos residem exclusivamente nas interfaces de ligação ao canal, e implementam um sistema de coordenação dos emissores em competição pelo canal sem recurso a árbitros externos.

Em todas essas abordagens cada interface emissora processa um *frame* de cada vez. Assim, até que cada *frame* seja recebido pelo destinatário, ou abandonado por

sucessivos erros, a interface emissora fará várias tentativas até o processar, e novos *frames* que cheguem à mesma interface para serem emitidos ficarão à espera de vez numa fila de espera.

Numa primeira fase vamos assumir as condições mais desfavoráveis: a taxa de erros é elevada, e o emissor não consegue detectar se o meio está ocupado, nem se houveram colisões. A solução mais simples consiste em logo que a interface recebe um *frame* para transmitir, começar a emiti-lo imediatamente sem esperar (excepto se ainda estiver a processar o *frame* anterior). Coloca-se então a questão de saber se o *frame* chegou ao destinatário, ou se foi destruído por erros ou colisões.

Uma solução simples para ambas as dúvidas é adoptar o mesmo procedimento que o protocolo *stop & wait*, i.e., o destinatário, caso tenha recebido bem o *frame*, envia um ACK ao emissor. Os *frames de ACK*, sendo muito curtos, terão uma probabilidade inferior aos de dados de entrarem em colisão. No entanto, em geral, também se poderão perder por erros ou colisões.

Este funcionamento requer um sistema de numeração das interfaces ligadas ao canal. O único requisito desses endereços é serem diferentes uns dos outros, mantendo-se a unicidade mesmo quando diversas interfaces podem ligar-se ou desligar-se do canal dinamicamente e sem aviso. Os *frames* têm endereços de destino e de origem no cabeçalho, ver a Secção 2.5. Um endereço de destino especial, chamado o endereço de *broadcast*, assinala que o *frame* se destina a todas as interfaces.

Quando uma interface recebe um *frame*, só se o endereço de destino é o seu (ou o de *broadcast*) é que esta o memoriza e processa. Se o endereço de destino for o endereço específico da interface, esta envia um ACK. Os *frames de broadcast* não são *acked* pois isso desencadearia a emissão de uma “tempestade” de ACKs.

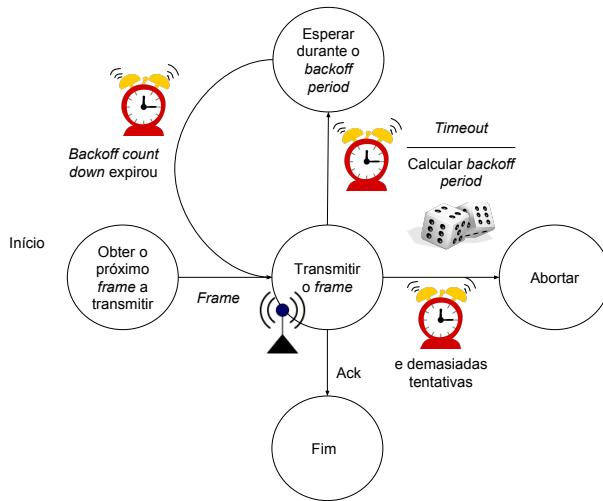


Figura 14.3: Máquina de estados usada no protocolo MAC ALOHA para a emissão de cada *frame*

Assim, tal como no protocolo *stop & wait*, o emissor instala um alarme temporizado (*timeout*) e espera pelo ACK. Se este não vier a tempo, reemite o *frame* até um certo número de tentativas. Se o *frame* se tiver perdido, o reemiti-lo imediatamente seria a receita para o desastre se os vários emissores usassem o mesmo valor de duração do alarme (*timeout*), pois continuariam eternamente a emitir simultaneamente, de forma sincronizada, e provocando novas colisões.

A solução para este desastre foi uma espécie de “ovo de Colombo” e consistiu

em tentar usar compassos de espera diferentes pelos diferentes emissores. Isso pode ser obtido usando um gerador de números aleatórios e adicionando ao *timeout* um **compasso de espera aleatório** suplementar que se chama o *backoff period*. A interface que obtiver o valor de *backoff period* mais baixo, emitirá primeiro que as outras. A Figura 14.3 mostra o diagrama da máquina de estados, ou simplesmente diagrama de estados, do processamento de cada *frame* por uma interface emissora. Este método foi baptizado de “ALOHA puro”¹.

Este protocolo MAC satisfaz de forma variável os critérios enunciados mais acima. Nomeadamente, se só um emissor pretender emitir, ele tem o canal à disposição e, se por hipótese os erros se devessem apenas a colisões, o *overhead* será baixo. O esquema é simples e não requer nenhum árbitro centralizado caro. No entanto, é óbvio que quando existem muitos emissores a tentarem emitir, ou bem que o intervalo dos compassos de espera é muito alargado, o que aumenta muito o desperdício da capacidade do canal (*overhead*), ou então a probabilidade de novas colisões será elevada.

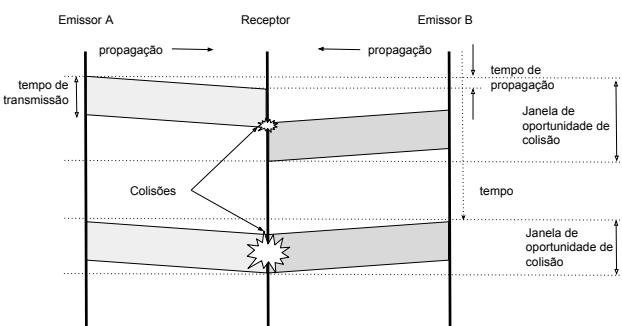


Figura 14.4: Com a versão Slotted ALOHA a oportunidade de colisões é menor

Uma primeira tentativa de melhoramento do MAC ALOHA consistiu em introduzir a noção de ***slots de emissão (intervalos fixos de emissão)***: uma interface só pode emitir no início de cada *slot*; desta forma diminui-se a probabilidade de as mensagens poderem colidir como ilustra a Figura 14.4. Em contrapartida, todas as interfaces têm de ser sincronizadas para reconhecerem o início de cada novo *slot* de forma sincronizada. Isso constituiu uma complicação suplementar. Este novo MAC chama-se MAC Slotted ALOHA e a sua máquina de estados está representada na Figura 14.5.

Para que o mecanismo de resolução de colisões seja efectivo, o intervalo temporal no qual é calculado o valor aleatório do *backoff period* é proporcional a um número significativo de vezes do tempo de emissão das mensagens, ou da duração do *slot*. Apesar de o MAC Slotted ALOHA ter melhor comportamento, o resultado em qualquer caso é que o *overhead* de ambas as variantes do MAC ALOHA é claramente superior a 50% da capacidade do canal sempre que há vários emissores, mesmo no caso mais favorável, e tende para próximo de 100% quando o número de emissores simultâneos vai aumentando, pois os emissores passam cada vez mais tempo à espera de poderem emitir ou a provocar colisões. Adicionalmente, a versão Slotted ALOHA introduz desperdícios suplementares se os *frames* tiverem uma dimensão inferior ao *slot*. Por estas razões é necessário ver se é possível melhorar estes protocolos.

¹ O método foi introduzido na primeira rede de dados sem fios publicamente acessível, desenvolvida na Universidade do Hawaii em 1971, que foi baptizada ALOHA (*Additive Links On-line Hawaii Area*).

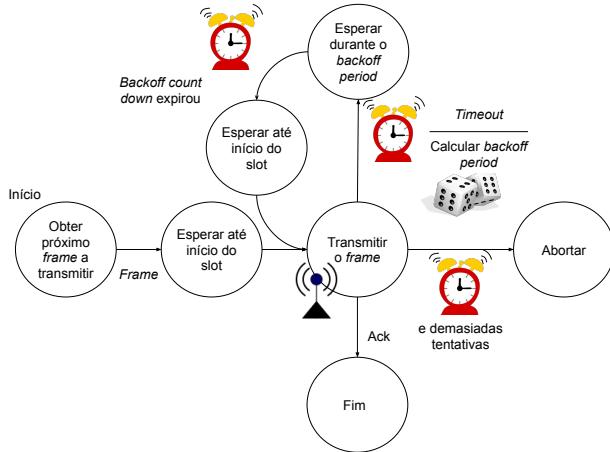


Figura 14.5: Máquina de estados usada no protocolo MAC Slotted ALOHA para a emissão de cada *frame*

Carrier Sense Multiple Access (CSMA)

Este melhoramento consiste em pôr os emissores permanentemente à escuta de actividade no meio de comunicação (*carrier sense*) e impedi-los de usarem o canal se este estiver ocupado. Assim, se o sinal da onda portadora (*carrier*) estiver presente, nenhum outro *frame* deve poder ser emitido, o que diminui a probabilidade de colisões.

Mas isso levanta o seguinte problema: que fazer quando a outra interface terminar a sua emissão e o meio de comunicação ficar livre? Há várias hipóteses, a primeira consiste em, assim que o canal ficar livre, todas as interfaces que têm *frames* à espera de serem emitidos, começam imediatamente a fazê-lo. Os *frames* que já foram transmitidos com colisão pelo menos uma vez não interferem obrigatoriamente nesta fase pois podem ainda estar à espera que o seu compasso de espera (*backoff period*) acabe, mas os que já podem ser transmitidos, assim como os novos, já podem colidir.

Assim, quando o meio fica livre, existem duas razões que podem estar na origem de que vários emissores se sincronizem para começarem a emitir ao mesmo tempo. A primeira diz respeito aos novos *frames*, que vão ser emitidos pela primeira vez, e a probabilidade de colisão é grande se houverem vários nessa situação. A segunda diz respeito aos *frames* que já foram emitidos pelo menos uma vez, mas já estão à espera de serem emitidos de novo, pois ainda não foi recebido o seu ACK e o *backoff period* para nova transmissão já se esgotou. Esta segunda origem pode ser menos frequente visto que esse período de espera é aleatório e potencialmente diferente em cada caso.

Existem várias formas de tratar estes problemas, nomeadamente: 1. ignorá-lo e emitir assim que o meio fica livre (designada por *1-persistent*); 2. baseada em testar periodicamente se o meio está livre e transmitir se este está livre, ou voltar a testar daí a um período aleatório senão estiver (designada por *nonpersistent*); e 3. a solução aplicada à versão Slotted ALOHA (designada *p-persistent*) que consiste em transmitir com probabilidade *p* quando, no início de um *slot*, o meio fica livre. A seguir vamos apresentar uma variante concreta que inspirou a usada nos canais Wi-Fi.

MAC CSMA com Collision Avoidance (CSMA/CA)

Este protocolo MAC adopta a solução que consiste em introduzir necessariamente um novo compasso de espera aleatório antes da primeira emissão de cada *frame*. Este MAC foi criado para situações em que o potencial emissor não tem a certeza se o

meio está livre ou não, como é o caso dos canais Wi-Fi por exemplo, como veremos na secção seguinte.

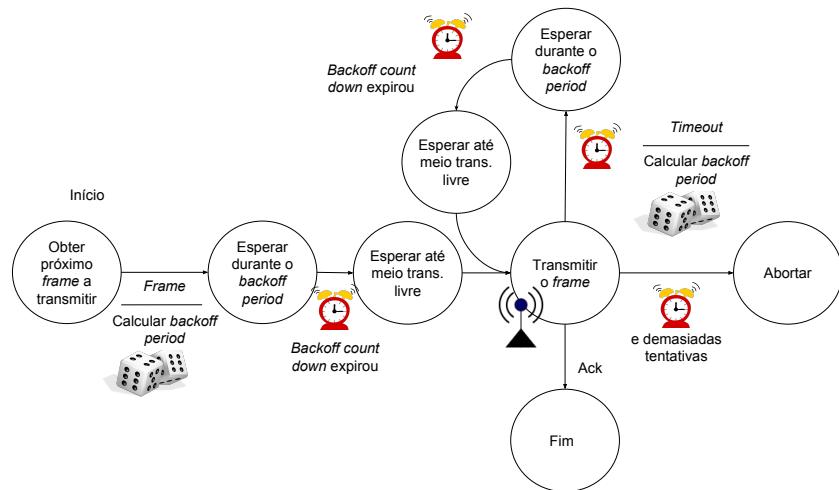


Figura 14.6: Máquina de estados usada no protocolo MAC CSMA/CA para a emissão de cada *frame*

Este período inicial suplementar de *backoff* é introduzido para aumentar a probabilidade de não haverem colisões iniciais, daí a designação *CA* - *Collision Avoidance*. A Figura 14.6 apresenta a respectiva máquina de estados.

Deteção de colisões pelos emissores

Em todos os MACs até agora referidos sempre que uma interface começa a transmitir, não pára a transmissão até ter transmitido integralmente o *frame*. Isso traduz-se num desperdício significativo da capacidade do canal quando ocorrem colisões. A pergunta a fazer é: não seria possível à interface emissora aperceber-se mais cedo da colisão e desta forma libertar o canal imediatamente? Como já foi referido, a grande maioria dos canais sem fios não permitem a deteção de colisões pois frequentemente o sinal emitido é vários milhares de vezes mais forte que o recebido. Adicionalmente várias características dos canais sem fios impedem essa deteção.

No entanto, o mesmo não se passa com os canais com fios. Nestes, dado que o sinal se propaga com pouca resistência pelo meio de comunicação, o emissor, durante a emissão, recebe o sinal emitido eventualmente misturado com o seu, e pode verificar se o sinal emitido e o sinal por ele recebido coincidem ou não. Se a resposta for negativa, é porque houve uma colisão.

CSMA/CD - CSMA com Collision Detection

Esta variante de protocolo MAC é especialmente adequada a canais com fios onde é fácil detectar as colisões de forma segura. Por outro lado, neste tipo de canais, se não houver colisão até ao fim da transmissão de um *frame*, então a probabilidade de não existirem erros e o mesmo ser bem recebido é muito elevada. Assim, pode-se igualmente abdicar do uso do ACK e deixar aos níveis superiores a eventual recuperação de erros. A Figura 14.7 apresenta a máquina de estados do protocolo CSMA/CD.

Uma questão suplementar interessante é saber durante quanto tempo (δt) tem de durar a emissão de uma interface para que essa emissão evite que outras interfaces possam começar a emitir. Ou seja, se uma interface *I* conseguir emitir sozinha durante

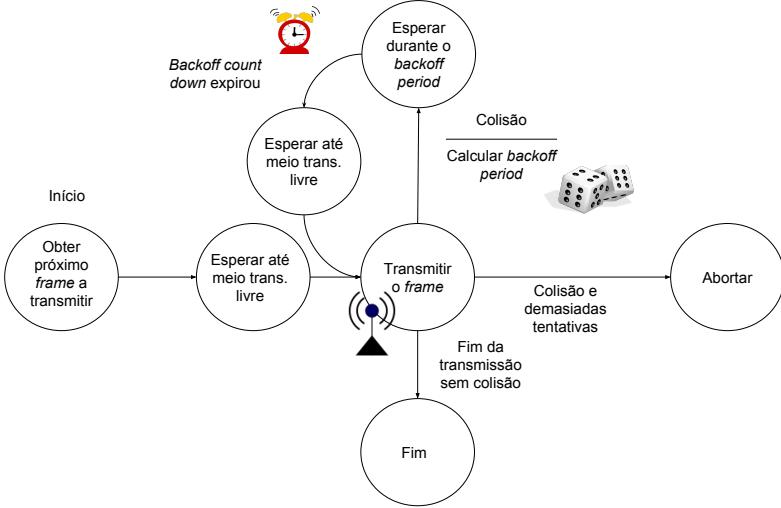


Figura 14.7: Máquina de estados usada no protocolo MAC CSMA/CD para a emissão de cada *frame*

δt , então todas as outras interfaces detectam essa emissão e abstêm-se de emitir, pelo que *I* como que reservou o canal. O valor de δt está relacionado com o tempo de propagação, como veremos a seguir, e condiciona a dimensão mínima dos *frames*.

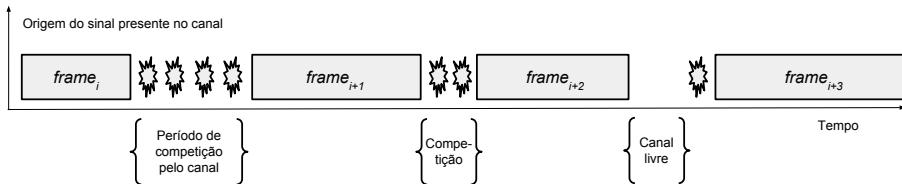


Figura 14.8: Evolução no tempo da utilização de um canal de difusão partilhado pelo protocolo CSMA/CD

Com o protocolo CSMA/CD podemos representar um diagrama temporal do estado da utilização do canal como o da Figura 14.8. Inicialmente o canal está ocupado a transmitir o *frame_i*. No fim, como existem diversas interfaces com *frames* para transmitir, segue-se um período de competição pelo canal com várias colisões. No final desse período de competição, uma das interfaces ganha e o *frame_{i+1}* é transmitido. Segue-se um novo período de competição, que termina com o início da transmissão do *frame_{i+2}*. No final da transmissão deste *frame*, nenhuma interface tem *frames* para transmitir, até que surge de novo uma situação em mais do que uma tem, e estas entram de novo em competição com um novo período de contenção. Este é seguido pela transmissão do *frame_{i+3}* pela interface que ganhou a competição.

Naturalmente, dadas as suas características, as variantes CSMA/CA e CSMA/CD desperdiçam uma menor capacidade do canal com a coordenação dos emissores que as variantes ALOHA. Em particular, a variante CSMA/CD sobre canais guiados satisfaz a maioria dos requisitos apresentados no início desta secção com *overhead* inferior ao das outras alternativas, particularmente devido às condições mais favoráveis em que é

utilizada, *i.e.*, deteção da ocupação do meio com maior fiabilidade, detecção de colisões e baixa taxa de erros do canal.

Os diferentes protocolos MAC acima apresentados foram caracterizados apenas de forma qualitativa com o objectivo de por em evidência alguns dos diferentes mecanismos a que os mesmos recorrem. A seguir vamos estudar de forma mais detalhada dois destes protocolos e a forma como esses mecanismos são concretizados em canais específicos.

Os canais baseados em difusão funcionam em modo *half-duplex* e por isso utilizam um protocolo MAC para regular e coordenar o acesso ao meio de comunicação.

Estes protocolos são vários e utilizam diferentes tipos de mecanismos, nomeadamente: ***frames* de ACK** quando a probabilidade de erro e de colisões é mais elevada, compassos de espera aleatórios (***Random Backoff Periods***) em caso de deteção de colisões ou antes de transmitirem um *frame* pela primeira vez, sondagem do meio de comunicação para verificar se este está livre ou ocupado (***CS - Carrier Sense***), e mecanismos de deteção de colisões (***CD - Collision Detection***).

Uma associação específica de diferentes mecanismos é usada em função do meio de comunicação e de outras características necessárias em diferentes contextos. Os protocolos MAC, quando têm por objectivo a interligação de computadores, foram normalizados pela IEEE, na família de protocolos com o prefixo **IEEE 802**.

14.2 Canais de dados Ethernet (IEEE 802.3)

Tendo como inspiração o protocolo ALOHA, Robert Metcalfe and David Boggs, na altura a trabalharem no Xerox Parc Research Center², inventaram uma rede, a que chamaram Ethernet [Metcalfe and Boggs, 1976]. Esta rede consistia num único canal de difusão baseado num cabo coaxial partilhado por (até) várias centenas de interfaces. O MAC usado foi o protocolo CSMA/CD.

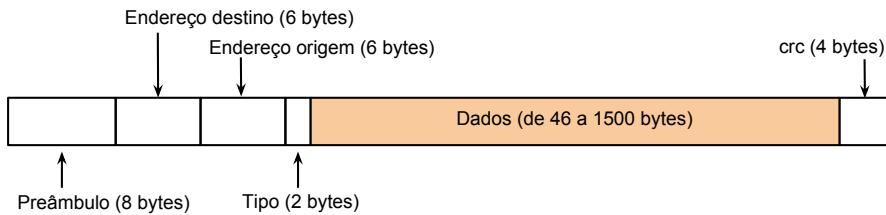
Estes canais já foram brevemente apresentados na Secção 2.5, onde o leitor poderá consultar o formato dos endereços e dos *frames* usados, e onde também é discutida a dimensão máxima dos mesmos. Resumidamente, um *frame* Ethernet tem o formato indicado na Figura 14.9. Uma das primeiras versões normalizadas e comercializadas destes canais e interfaces foi desenvolvida no seio da IEEE através de um conjunto de normas com o prefixo 802.3, daí esses canais também serem designados canais IEEE 802.3.

Os endereços de nível canal, ou nível MAC (*MAC addresses*), das interfaces Ethernet têm 48 bits e são afectados pelos seus fabricantes de forma que garantem que os mesmos são únicos no mundo pois cada fabricante tem de adquirir intervalos privados de endereços junto da IEEE. Desta forma está garantido que quando quaisquer duas interfaces deste tipo são ligadas ao mesmo meio de difusão, as mesmas terão necessariamente endereços canal distintos³.

A concretização do MAC CSMA/CD nas interfaces Ethernet passa por garantir que se o menor *frame* possível possa ser transmitido sem que o seu emissor tenha

²O Xerox Parc era um laboratório de investigação da Xerox na Califórnia, em Palo Alto, no Silicon Valley.

³A maioria dos *drivers* das interfaces Ethernet nos sistemas de operação actuais permitem alterar esses endereços pelo que nesse caso a unicidade não está garantida, o que pode levantar um problema de segurança.

Figura 14.9: Formato dos *frames* Ethernet

detectado uma colisão, então é porque todas as outras interfaces ligadas ao mesmo canal detectaram esta emissão e abstêm-se de interferir. Interessa que a duração deste *frame* seja a menor que possível dado que um *frame* pode ter uma parte de dados muito pequena. Na Ethernet 802.3 essa dimensão mínima é de 64 bytes ou 512 bits. A seguir mostra-se como este valor foi fixado.

Repare-se que é preciso garantir que a emissão com sucesso (sem colisão) de um *frame* de dimensão mínima adquiriu o canal só para ele através de uma espécie de *lock*. A fixação desta dimensão está relacionada com o tempo de transmissão e o tempo de propagação, *i.e.*, o tempo de transmissão do *frame* mínimo (T_t) tem de ser maior ou igual a duas vezes o tempo de propagação (T_p). A Figura 14.10 mostra porquê.

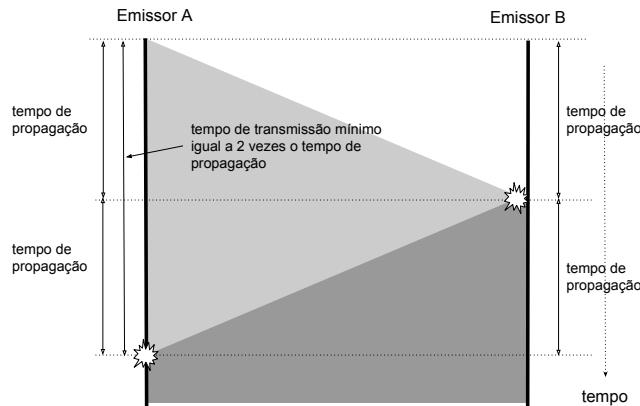


Figura 14.10: O emissor A começa a emitir, mas mesmo antes que o seu sinal chegue a B, este, julgando o meio livre, começa a emitir. B detecta imediatamente a colisão, mas A só detecta a colisão quando o sinal de B se propagar até ele, *i.e.*, ao fim de $2 \times T_p$.

A rede Ethernet inicial fixou a dimensão do *frame* mínimo em 64 bytes pois a norma admitia que o meio de propagação do sinal poderia ter no máximo 2.500 metros. Duas vezes o tempo de propagação extremo a extremo máximo corresponde a $51\mu s$ ($5 \cdot 10^3 / 2 \cdot 10^8$ s). A 10 Mbps, que era o débito dos canais Ethernet iniciais, neste tempo pode-se transmitir 511 bits ou aproximadamente 64 bytes (8×64 bits / 10^7 bps = $51\mu s$). Ao tempo de transmissão mínimo chamou-se o ***Collision Slot Time*** ou ***Contention Slot Time (CST)*** e corresponde, como já vimos, a $51,2\mu s$ a 10 Mbps.

Quando uma interface detecta uma colisão, emite um curto sinal especial chamado *jam signal*, que se destina a assinalar a todas as outras interfaces que aquele *frame*

não é válido.

Com a evolução da Ethernet foi possível aumentar o débito do canal para 100 Mbps. Neste caso o *frame* mínimo passaria a ter 640 bytes o que seria demasiado grande e levaria a um grande desperdício pois muitos *frames* são inferiores àquela dimensão. A alternativa consistiu em impor uma dimensão máxima do meio de comunicação de 250 metros. Assim, o *Collision Slot Time* (CST) a 100 Mbps passou a ser de 5,12 μ s e nesse período é possível emitir 512 bits ou 64 bytes a 100 Mbps. Desta forma a dimensão mínima dos *frames* continuou a ser independente do débito dos canais Ethernet.

Um outro aspecto muito relevante na implementação do MAC CSMA/CD tem a ver com a forma como o *backoff period* é calculado. Este período é proporcional a $n \times$ CST, onde n é calculado de forma aleatória através de um método que consiste em ir alargando o intervalo em que se calcula cada número aleatório, conforme o número de tentativas de resolução da colisão já realizadas anteriormente sem êxito. Após a primeira colisão o intervalo é $[0, 1]$, após a segunda colisão passa para $[0, 3], \dots$, após k colisões passa para $[0, 2^k - 1]$. Este método designa-se um *binary exponential backoff* e destina-se a acomodar uma forma mais inteligente de resolver as colisões.

Inicialmente, há a esperança de que só duas interfaces estejam em competição e por isso tenta-se resolver a colisão usando compassos de espera de 0 ou $1 \times$ CST (uma espécie de “moeda ao ar”). Caso suceda uma nova colisão, alarga-se o intervalo em que se tenta resolver a colisão usando compassos de espera de 0, 1, 2 ou $3 \times$ CST, e assim sucessivamente. Com $k = 10$ o intervalo é agora de 0 a $1023 \times$ CST, o maior intervalo possível indicado pela norma. A norma também especifica que não se devem tentar mais do que 15 vezes (ou 16 se contarmos a transmissão inicial). Após este limite considera-se impossível transmitir qualquer *frame* e este é abandonado. Ao intervalo no qual se calcula o compasso de espera chama-se a *collision window*. A Listagem 14.1 apresenta o pseudo código do protocolo CSMA/CD.

Listing 14.1: Pseudo-código do algoritmo CSMA/CD executado pelas interfaces IEEE 802.3

```

1  CST = 0.000051; // 10 Mbps
2
3  boolean CSMA_CD_TransmitFrame (frame) {
4      k = 1; // transmission attempts
5      while ( true ) {
6          wait_for_free_media(); // carrier sense
7          transmit ( frame ); // try transmission
8          if ( success ) return true; // end with success
9          transmit ( jam_signal ); // else collision
10         if ( k == 16 ) return false // unsuccess, abort transmission
11         else k++; // one more try
12         cw = min( 2^k - 1, 1023); // collision window <= 1023
13         n = random ( 0 .. cw );
14         wait ( n*CST );
15     }
16 }
```

Na primeira versão da Ethernet o meio de comunicação através do qual a difusão era realizada era um cabo coaxial, semelhante aos cabos coaxiais (ver o Capítulo 2) utilizados ainda recentemente pelas televisões por cabo, ao qual eram ligados (através de “baixadas”) os diferentes computadores. Quando uma interface transmitia, o sinal espalhava-se pelo cabo coaxial e atingia todas as outras interfaces. A Figura 14.11 ilustra este tipo de instalação da rede Ethernet tal como esta foi usada até ao início dos anos de 1990.

Mais tarde passaram-se a usar cabos mais flexíveis, designados UTP, e com fichas RJ45 (ver o Capítulo 2), que interligavam todos os equipamentos a um dispositivo central, chamado *hub ethernet*, que funcionava como um repetidor do sinal. Quando

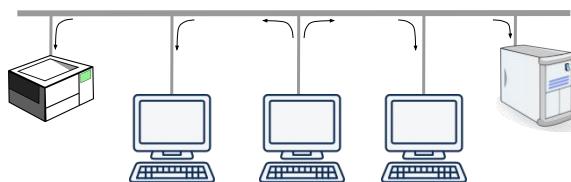


Figura 14.11: Um conjunto de computadores interligados a 10 Mbps por uma rede Ethernet 802.3 baseada num cabo coaxial partilhado entre os diferentes computadores

uma porta do *hub* recebia sinal, este amplificava-o e emitia-o para todas as outras portas. Ou seja, o meio de comunicação passou a ser constituído por uma estrela de cabos UTP com um amplificador no meio. A Figura 14.12 ilustra este tipo de instalação da rede Ethernet que passou a ser popular a partir dos anos de 1990, tendo substituído completamente as ligações baseadas em cabos coaxiais.

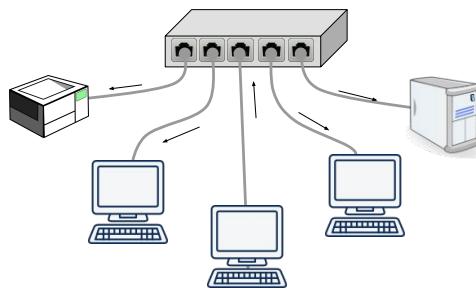


Figura 14.12: Um conjunto de computadores interligados por uma rede Ethernet 802.3 baseada em cabos UTP e um repetidor de sinal (*hub* Ethernet)

Todas estas configurações funcionavam em modo *half-duplex* através de um meio de comunicação em difusão partilhado, e realizavam o controlo de acesso ao meios através do protocolo MAC CSMA/CD, tal como acabámos de descrever.

O problema da deteção das colisões levantou questões delicadas quando foi possível construir interfaces de comunicação Ethernet com débitos de 1, 10 e 100 Gbps, como é corrente hoje em dia. Para a transmissão do sinal nos débitos mais elevados é necessário recorrer a fibras ópticas. Mas o mecanismo de detecção de colisões levaria a que a 1 Gbps o menor *frame* passasse para 640 bytes ou 4096 bits de comprimento, ou a distância entre os dois equipamentos mais afastados não poderia exceder 25 metros. A 10 Gbps, o mesmo raciocínio levaria a que se adoptasse a distância máxima de 2,5 metros *etc.*

A Tabela 14.1 mostra alguns parâmetros característicos do protocolo MAC Ethernet em função dos diferentes débitos. A tabela omite que nos débitos mais elevados podem usar-se *jumbo frames*, ver a Secção 2.5, uma opção que tem de ser explicitamente parametrizada em todos os equipamentos envolvidos.

Felizmente foi possível encontrar uma alternativa para o problema da dimensão mínima dos *frames*. Como os cabos UTP e as fibras ópticas suportam 2 canais *simplex* a transmitirem simultaneamente em cada sentido, foi possível passar a usar canais baseados nesses meios de comunicação em modo *full-duplex*, mas só ponto-a-ponto. Uma rede Ethernet passou então a poder usar simultaneamente canais *half-duplex*

Tabela 14.1: Parâmetros característicos do protocolo MAC Ethernet CSMA/CD IEEE 802.3 em função do débito das interfaces

Parâmetro	10 / 100 Mbps	1 Gbps	10 / 100 Gbps
Backoff slot time	512 bit times	4096 bit times	não se aplica
Inter Frame Space (IFS)	96 bits	96 bits	96 bits
# tentativas máximas	16	16	não se aplica
Jam size	32 bits	32 bits	não se aplica
Minimum frame size	64 bytes	640 bytes	64 bytes
Maximum frame size	1518 bytes	1518 bytes	1518 bytes

multi-ponto e *full-duplex* ponto-a-ponto. Os primeiros só podem transmitir um *frame* de cada vez entre $n \geq 2$ interfaces. Os segundos podem transmitir simultaneamente nos dois sentidos mas apenas entre duas interfaces, *i.e.*, ponto-a-ponto. Para fazer uma rede com canais ponto-a-ponto, os *hubs* Ethernet tiveram de ser substituídos por verdadeiros comutadores de pacotes a funcionarem no modo *store & forward*, que estudaremos no Capítulo 15, e que se chamam *switches* Ethernet.

Nos débitos superiores a 1 Gbps, os canais só podem funcionar em modo ponto-a-ponto, sem protocolo MAC CSMA/CD. No débito 1 Gbps, quase sempre os canais também só funcionam em modo ponto-a-ponto, mas a norma ainda admite uma forma especial de funcionamento com o MAC CSMA/CD que é pouco usada, mas que exige *frames* de comprimento mínimo de 4096 bits ou 512 bytes, contendo eventualmente vários pacotes IP, ver Tabela 14.1.

Com a descida de preço dos equipamentos, hoje em dia a grande maioria das redes Ethernet funcionam predominantemente com canais *full-duplex* e os equipamentos com essas interfaces são geralmente interligados por *switches* Ethernet. Os canais *half-duplex* podem funcionar a 10 e a 100 Mbps (ou até 1 Gbps em casos particulares). Os canais com débitos maiores que 1 Gbps são todos ponto-a-ponto e não estão dependentes de nenhum protocolo MAC.

A Figura 14.13 mostra uma configuração típica, caracterizada pela coexistência de canais *half* e *full-duplex*. Duas sub-redes, baseadas cada uma num canal *half-duplex* materializado por um *hub* Ethernet, são interligadas por dois *switches* Ethernet ligados por um canal *full-duplex*. Este canal, é formado por dois canais *simplex*, cada um dedicado a uma direcção de transmissão, e por isso não necessitam de qualquer protocolo MAC. Sempre que a interface necessitar de transmitir por um desses canais, pode fazê-lo imediatamente pois não existem competidores pelo meio de comunicação. Por essa razão, os canais Ethernet ponto-a-ponto *full-duplex* não têm limites de dimensão, a não ser os relacionados com a propagação do sinal. Usando fibras ópticas podem ter até centenas de Kms.

Esta interligação e coexistência é possível pois as interfaces modernas IEEE 802.3 baseadas em cabos de cobre conseguem funcionar nos dois modos. Em modo *half-duplex* executam o protocolo MAC CSMA/CD e partilham o meio de comunicação com outras interfaces a funcionarem no mesmo modo. Por isso se diz que esse meio assim partilhado é um domínio de colisão único.

Quando a interface está a funcionar em modo *full-duplex*, o canal só pode ligar a outra interface e não é executado nenhum protocolo MAC. Esses canais não têm colisões, são *collision free*. Os *switches* Ethernet têm geralmente interfaces que podem funcionar nos dois modos. As interfaces Ethernet actuais (*e.g.*, com conectores RJ45) nos computadores também podem funcionar nos dois modos. Na figura, os dois *switches* Ethernet têm portas *half-duplex* ligadas aos *hubs* e portas *full-duplex* que os ligam um ao outro. A configuração apresentada mostra aquilo que se costuma designar por

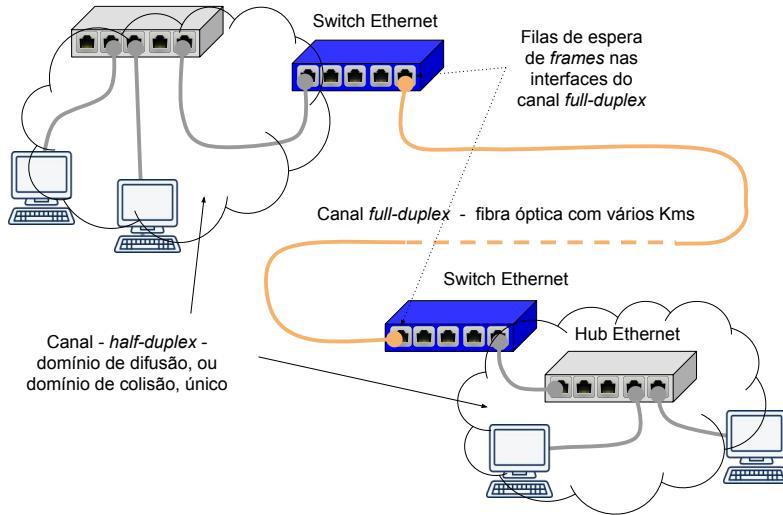


Figura 14.13: Coexistência na mesma rede Ethernet 802.3 de *hubs* e *switches* Ethernet e de canais *half* e *full-duplex*

dois domínios de colisão independentes.

Um outro aspecto interessante é analisar onde existem filas de espera. As portas dos *hubs* não têm filas de espera de *frames* porque estes são apenas meios de comunicação do sinal. Num *hub* só pode circular um *frame* de cada vez. Já as interfaces dos computadores e as interfaces dos *switches* têm associadas filas de espera para ser possível o funcionamento característico da multiplexagem estatística.

As tecnologias designadas como Ethernet nasceram nos anos de 70 do século passado, e funcionaram pela primeira vez em laboratório a 3 Mbps. Como foi apresentado acima, hoje em dia esses canais funcionam tipicamente de 100 Mbps a 100 Gbps e é provável que o débito possa vir a aumentar nos próximos anos. Todas as variantes partilham, grosso modo, o formato de *frame* e o sistema de endereçamento, os quais suportam a total interoperabilidade entre todas as versões desses canais. No que diz respeito ao protocolo MAC CSMA/CD, o mesmo é usado apenas nos canais de menor débito e a curta distância, dadas as características do protocolo MAC. Finalmente, como suportes de comunicação do sinal, podem ser usados cabos coaxiais (menos frequente hoje em dia), cabos *twisted pair* ou fibras ópticas.

As tecnologias colectivamente designadas como “Ethernet” são um exemplo notável do êxito de uma família de canais que evoluiu, e continua a evoluir, durante mais de 40 anos. O seu êxito está provavelmente relacionado com a simplicidade (KISS - *Keep it Simple, Stupid*), a compatibilidade mantida durante a evolução, a sua perfeita integração com o mundo IP, e o preço. Este último tem descido de forma consistente porque a simplicidade, os meios de comunicação usados, e um volume de produção de muitos milhões, constituem uma combinação imbatível.

Veremos a seguir que existem canais compatíveis com os canais 802.3 mas que usam como meio de propagação do sinal a atmosfera.

As normas IEEE 802.3 destinam-se a canais guiados em que é possível detectar as colisões, a fiabilidade da comunicação é elevada quando não existem colisões, e é possível reservar o meio através de um pequeno *frame* desde que a distância máxima entre interfaces seja relativamente pequena. Este conjunto de razões permitem a utilização de um **protocolo MAC do tipo CSMA/CD sem utilização de ACK**.

Uma outra faceta interessante da implementação na Ethernet do protocolo MAC CSMA/CD consistiu na adopção do método *binary exponential backoff*, que adapta a duração dos compassos de espera aleatórios em função da maior ou menor probabilidade de colisão.

Modernamente, esta família de normas abrange um conjunto de tecnologias que comportam funcionamento em *half-* e *full-duplex*. Neste último caso, não existe necessidade de utilizar um protocolo MAC, nem limitações de distância, porque o canal funciona em modo ponto-a-ponto entre apenas duas interfaces.

14.3 Canais de dados Wi-Fi (IEEE 802.11)

A utilização de canais sem fios é muito interessante pela comodidade que proporcionam e tornam-se imprescindíveis quando se usam equipamentos móveis. Dado o êxito da Ethernet nos escritórios e nas casas particulares, a utilização de um canal semelhante mas sem fios sempre foi muito atractiva.

Todos os canais sem fios utilizam ondas portadoras electromagnéticas (ver a Secção 2.3) que se propagam na atmosfera. Dependendo das frequências usadas e da potência da emissão, estas ondas podem atingir de alguns metros a milhares de quilómetros. Como a propagação se realiza geralmente de forma não guiada (com antenas omnidireccionais), é fácil os diferentes emissores interferirem uns com os outros. Para evitar essa situação, os canais sem fios usados pela rádio, a televisão, as redes de telemóveis e outras redes de voz móveis utilizam frequências portadoras reservadas ou licenciadas. Essas licenças são concedidas, contra pagamento, por uma autoridade de gestão do espaço radio-eléctrico⁴.

Para evitar as complicações e os custos do licenciamento, as normas IEEE 802.11 foram concebidas com base na utilização de intervalos de frequência deixadas livres pela regulação do espaço radio-eléctrico. Estas gamas de frequência têm a designação genérica de bandas ISM (*Industrial, Scientific and Medical bands*). Os intervalos de frequências mais usados são de 2,4 a 2,5 GHz e de 5 a 6 GHz⁵, que constituem uma fracção reduzida do espaço radio-eléctrico disponível e que se caracterizam por serem muito atenuadas por obstáculos. Só são permitidas baixas potências de emissão para evitar interferência entre diferentes canais e de forma a tentar confinar o respectivo âmbito a espaços privados (escritórios, casas, etc.).

As interfaces IEEE 802.11 e o software que as controlam, nesta secção designadas por equipamentos terminais, ou simplesmente **terminais**, exploram automaticamente a gama de frequências disponíveis e permitem a coexistência de vários canais de dados no mesmo espaço, através de multiplexagem em frequência (ver a Secção 3.1). Dependendo das larguras de banda de frequências usadas na multiplexagem em frequência,

⁴ Cada país tem a sua autoridade de regulação específica. Em Portugal esse papel cabe à ANACOM, a Autoridade Nacional das COMunicações.

⁵ As gamas reais de frequências disponíveis variam de país para país e é provável que mais gamas de frequências sejam libertadas no futuro, à medida que a pressão para a utilização de Wi-Fi suba, e que a televisão tradicional liberte frequências.

varia o débito e o número de sub-canais utilizáveis em cada uma das gamas de frequências. Para diminuir a interferência e também poupar energia nos terminais móveis, as potências de emissão são muito reduzidas, de tal forma que um canal IEEE 802.11 tem um âmbito de 20 ou 30 metros, ou menos, em espaço urbano, e de algumas centenas de metros ao ar livre (ou mais mas só se se utilizarem antenas especiais direcionadas).

Vários factores contribuem para uma elevada taxa de erros nestes canais, nomeadamente: a dificuldade das ondas com as frequências usadas atravessarem obstáculos (*e.g.*, paredes), fenómenos de reflexão que fazem com que o receptor receba várias ondas ligeiramente desfasadas, a baixa potência de emissão, a elevada resistência da atmosfera à propagação do sinal que provoca uma queda exponencial da sua potência durante a propagação, as interferências de outros equipamentos (*e.g.*, os fornos micro-ondas domésticos usam as mesmas frequências com alta potência) e a coexistência de múltiplos canais na mesma ou em frequências próximas que interferem uns com os outros.

A primeira versão da norma IEEE 802.11 data de 1997 e introduziu um canal multi-ponto sem fios a funcionar a 1 ou 2 Mbps. Mais tarde constitui-se uma aliança de fabricantes de interfaces IEEE 802.11 que tomou o nome Wi-Fi⁶ e que está na origem do nome corrente desta tecnologia.

Apesar do baixo débito inicial, o êxito foi significativo e seguiram-se versões a 11 e 54 Mbps (IEEE 802.11a/b e IEEE 802.11g). Hoje em dia os canais Wi-Fi banalizaram-se completamente, o que tem motivado sucessivas evoluções da norma, existindo versões actuais (*e.g.*, IEEE 802.11n introduzida em 2009 e IEEE 802.11ac introduzida em 2014) que admitem débitos de centenas de Mbps ou mesmo mais em casos particulares. Veremos a seguir que dadas as características do meio de propagação, dos protocolos MAC usados e do engarrafamento do espaço urbano com vários canais Wi-Fi em competição directa, o débito extremo a extremo experimentado pelos terminais é muito mais baixo.

Características do canal e do protocolo MAC adoptado

Devido às suas características, os canais IEEE 802.11 usam um protocolo MAC do tipo CSMA/CA. Com efeito, dadas as características do meio de transmissão, não é possível detectar colisões. A principal razão consiste no facto de que no meio usado a potência do sinal decresce rapidamente com a distância e, quando uma interface emite sinal, um sinal concorrente seria recebido com uma potência comparativamente tão baixa, que este não interferiria a ponto de ser interpretado como uma colisão. A Figura 14.14 ilustra a situação. Assim, a interface de um terminal só pode estar em dois modos alternativos: emissão ou recepção.

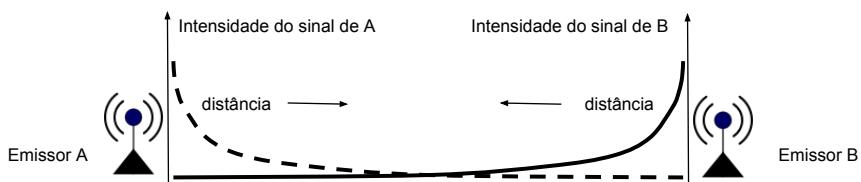


Figura 14.14: Os terminais A e B estão suficientemente afastados para não receberem os respectivos sinais com intensidade suficiente para detectarem a colisão

Por outro lado, o emissor não consegue detectar com exatidão se o meio está livre ou não, ou se houve ou não colisões. A Figura 14.15 ilustra uma situação designada

⁶ Com origem na mote de marketing *Wireless Fidelity* TM.

por “terminal escondido” (*hidden terminal*), em que um obstáculo esconde um terminal de outro (A e B não recebem as respectivas emissões) mas um terceiro (C) recebe de ambos. Este fenómeno impede a deteção da ocupação do meio com fiabilidade e também esconde colisões.

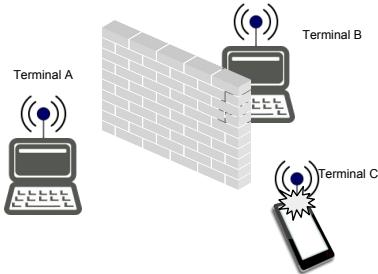


Figura 14.15: Terminal escondido. A não detecta a emissão de B e vice-versa. No entanto, C recebe ambos os sinais.

Estas não são as únicas razões que impedem a deteção de colisões ou da ocupação do canal com fiabilidade. Existem outras situações em que um terminal não detecta outro que está activo, e situações em que um terminal “acha” que outro está activo, no entanto isso não o impediria de emitir. Esta última situação é designada por “terminal exposto” (*exposed terminal*). A figura Figura 14.16 ilustra ambas as situações. Na mesma, os círculos representam o alcance do sinal emitido por cada terminal.

Devido a este conjunto de razões o protocolo MAC tem de usar ACKs para ter a certeza de uma correcta recepção e tentar compensar as eventuais colisões visto que não as consegue detectar. Por outro lado, se uma estação detectar que o meio está ocupado, não tem outra hipótese senão diferir a emissão, mesmo na situação de terminal exposto. Finalmente, quando uma estação começa a emitir, só termina quando completar a transmissão do *frame*.

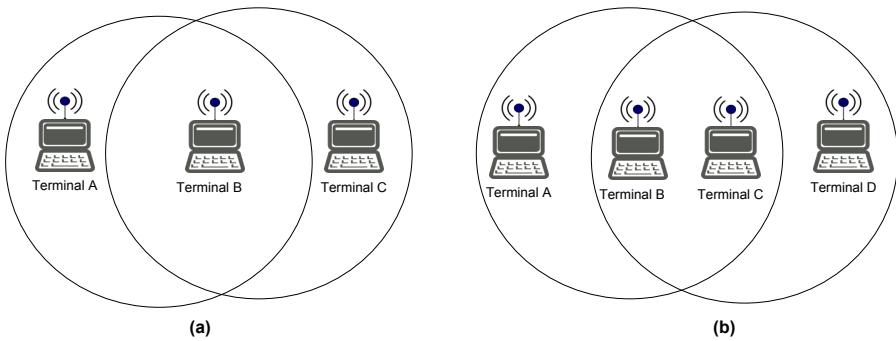


Figura 14.16: Em (a) os terminais A e C estão suficientemente afastados para não receberem os respectivos sinais com intensidade suficiente para detectarem a colisão que ocorre em B. Em (b) apesar de B e C estarem expostos aos respectivos sinais, B poderia emitir para A ao mesmo tempo que C emitiria para D.

Funcionamento de uma rede Wi-Fi

A norma IEEE 802.11 especifica que o canal pode funcionar em vários modos distintos. No primeiro, designado **modo ad hoc**, os terminais comunicam directamente uns com os outros, ver a Figura 14.17 parte (a). No segundo, designado **modo infra-estrutura**, os terminais comunicam indirectamente e um terminal especial, designado **AP - Access Point**, que funciona como comutador de pacotes (por *store & forward* de frames) intermediário de todas as comunicações entre os terminais que lhe estão ligados, ver a Figura 14.17 parte (b).

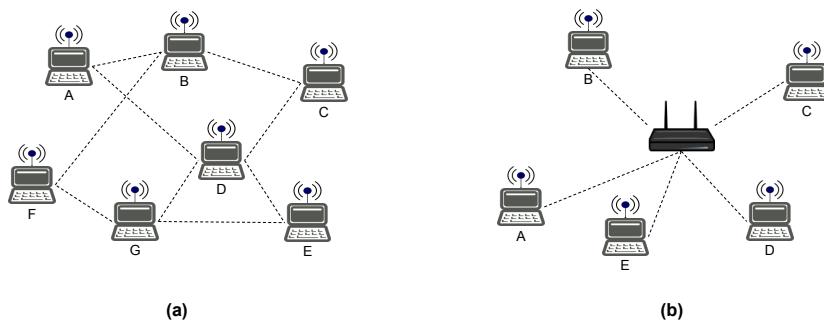


Figura 14.17: Modo ad hoc (a) e modo infra-estrutura (b). Neste último um ponto de acesso serve como intermediário das comunicações entre as diversas estações

Dois ou mais APs podem também ser interligados por *switches Ethernet* que asseguram a comunicação entre os terminais ligados a diferentes APs, assim como eventualmente com outras redes, ver a Figura 14.18.

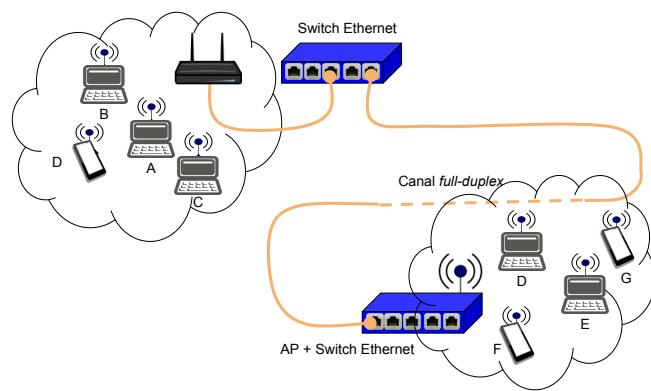


Figura 14.18: Sistema com vários APs interligados por uma rede baseada em *switches Ethernet*. Os APs funcionam também como *switches* e um dos *switches* é simultaneamente também um AP.

Na comunicação em modo ad hoc, quando a origem e os destino têm de passar por terminais intermédios, os diferentes terminais têm de funcionar como comutadores de pacotes. Por exemplo, na Figura 14.17, parte (a), o terminal A para enviar frames a E terá de passar por D. Sem essa hipótese, o âmbito nesse modo seria muito mais

restrito. Este tipo de redes, que também se costumam chamar redes sem fios em malha (*mesh*), exigem funcionalidades especiais de encaminhamento nos terminais e só são usadas quando o modo infra-estrutura, baseado em APs, não pode ser usado (*e.g.*, redes cooperativas, em zonas rurais, em zonas de desastre, *etc.*).

O modo infra-estrutura, com APs, é o mais popular e fácil de montar pois não exige nenhuma funcionalidade ou parametrização adicional dos terminais. Para que diversos APs possam coexistir no mesmo espaço físico, cada AP e os terminais que lhe estão associados pertencem a um **SSID - Service Set Identifier** ou identificador de rede diferente.

O AP difunde periodicamente o seu SSID, o seu endereço MAC e diversas características do canal através de *frames* especiais chamados **beacons frames**. São os diferentes SSIDs captados que os terminais mostram aos utilizadores quando assinalam que há várias redes Wi-Fi disponíveis.

O terminal para poder comunicar através de um canal identificado por um SSID, tem de se associar ao AP do mesmo através de um protocolo especial de associação. Quando a rede é fechada, é também executado um protocolo de autenticação e de estabelecimento de chaves criptográficas para tornar as comunicações seguras. Se um terminal detecta vários APs do mesmo SSID, escolhe automaticamente associar-se àquele com melhor qualidade de sinal. Se a qualidade do sinal se alterar (*e.g.*, por o terminal se deslocar), este pode decidir mudar dinamicamente o AP a que está associado.

O AP também informa periodicamente os terminais que lhe estão associados sobre qual a melhor potência de emissão que devem adoptar. Adicionalmente, os APs actuais escolhem automaticamente a gama de frequências em que operam de forma a diminuir a probabilidade de colisão com outros APs na proximidade. Os terminais, antes de se associarem, pesquisam os **beacons frames** presentes em diferentes frequências até se ligarem ao AP seleccionado.

N numa configuração baseada em infra-estrutura, os APs assumem diversos papéis. Assim, apesar de alguns terminais conseguirem comunicar directamente uns com os outros, um AP estende mais facilmente âmbito do canal. O AP também permite aos terminais libertarem-se mais facilmente dos *frames* que querem emitir e podem por isso poupar mais energia. Esta poupança é vital pois os terminais são geralmente móveis e alimentados a bateria. Quando se usam diversos APs distintos no mesmo espaço físico, é possível fazer coexistir mais facilmente vários canais distintos. Finalmente, os APs desempenham um papel importante na segurança do canal.

Na ausência dos APs os terminais têm de funcionar como comutadores de pacotes e fazerem encaminhamento de pacotes. Para além disso, têm de usar protocolos mais complexos para garantirem a coordenação das frequências, a autenticação e a segurança das comunicações. A faceta semi-centralizada do modo infra-estrutura apresenta diversas vantagens e é geralmente a preferida sempre que é possível utilizar este modo.

Funcionamento do protocolo MAC

Um terminal que pretende transmitir um *frame* começa por esperar durante um período designado IFS (**IFS - Inter-Frame Space**) seguido de um *random backoff period* inicial (excepto se o terminal não usou recentemente o canal e este está livre durante todo o IFS). Quando esse período se esgota e o meio está livre, inicia a transmissão até ao fim do *frame*. Em seguida espera pelo ACK e caso não o receba, faz nova tentativa após novo IFS seguido de um novo *random backoff period*. Por omissão, *i.e.*, sem prioridades, ver a seguir, o valor do IFS é igual a DIFS (**DIFS - Distributed Inter-Frame Space**). De forma mais sistemática:

1. Caso o terminal não tenha tido recentemente actividade e o meio está livre pelo menos durante DIFS, passar imediatamente à etapa 3.

2. Calcular um *random backoff period* multiplicando por um número aleatório o período *slot time* e esperar que o contador venha a 0. O contador só é decrementado quando o meio está livre e depois de passar DIFS a seguir ao fim da última ocupação por um outro *frame* de um competidor.
3. Emitir o *frame* sem parar até ao fim (visto que não consegue detectar colisões).
4. Esperar pelo ACK.
5. Caso receba o ACK no período previsto, a transmissão foi concluída com sucesso. Senão recomeça o processo na etapa 2, mas usando um compasso de espera calculado de forma aleatória num intervalo multiplicado por dois (*binary exponential backoff*). O número máximo de repetições é variável mas está normalmente fixado em 4.

Uma das maneiras de introduzir prioridades no acesso ao meio consiste em usar diferentes compassos de espera, *i.e.*, diferentes valores de IFS. Assim, o receptor, se recebeu com êxito o *frame*, espera o intervalo de tempo SIFS (**SIFS** - *Short Inter-Frame Space*), menor que DIFS, e transmite imediatamente o ACK. Esta forma de proceder, ilustrada na Figura 14.19, assegura que um ACK é sempre transmitido antes de qualquer outro *frame* a menos de problemas relacionados com um “terminal escondido”.

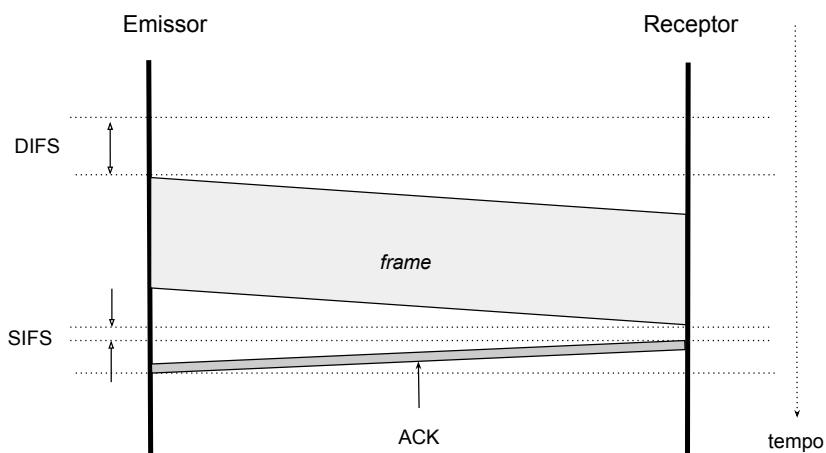


Figura 14.19: Funcionamento do MAC IEEE 802.11 e os respectivos compassos de espera DIFS e SIFS

O MAC IEEE 802.11 introduz um mecanismo suplementar que se chama **NAV** (*Network Allocation Vector*). Este mecanismo comprehende dois aspectos. O primeiro consiste em introduzir em todos os *frames* de dados um campo com a indicação do tempo necessário para a sua transmissão e recepção do respectivo ACK. Este campo está no cabeçalho e permite a qualquer outro terminal saber quanto tempo o canal vai estar ocupado por aquela transmissão concorrente. Se o *frame* não lhe for destinado, o terminal “pode dormir” (desligar a interface e o processamento de sinal do rádio) até ao fim do período NAV.

Adicionalmente, o NAV pode ser complementado por um mecanismo cujo efeito é a tentativa de reserva do canal durante esse período. O mecanismo funciona de tal forma que diminui a probabilidade de colisões em casos de “terminais escondidos”. Quando um terminal pretende enviar um *frame* extenso, emite preliminarmente um

frame especial designado RTS (*Request To Send*) em que especifica o NAV de que necessita, isto é o tempo para transmitir o *frame* e receber o seu ACK. O terminal de destino do futuro *frame*, se receber o RTS, deve responder com um *frame* CTS (*Clear To Send*) que inclui também um NAV com o tempo que levará a transmissão solicitada a terminar e a ser *acked*.

Os *frames* RTS e CTS são muito pequenos e portanto a probabilidade de provocarem colisões é menor. No entanto, caso tenham sido trocados com êxito, avisam todos os terminais que os receberem que o meio vai estar ocupado pelo emissor, pois este acabou, com a anuência do receptor, de reservar o canal. Este tipo de mecanismo só deve ser usado com *frames* a partir de uma dimensão significativa, pois com *frames* muito pequenos o *overhead* extra que introduz pode não compensar os ganhos obtidos. A dimensão do menor *frame* cuja transmissão já requer a utilização do mecanismo RTS / CTS é um parâmetro com um valor por omissão de poucas centenas de bytes (e.g., 400 bytes). A Figura 14.20 ilustra o funcionamento do mecanismo.

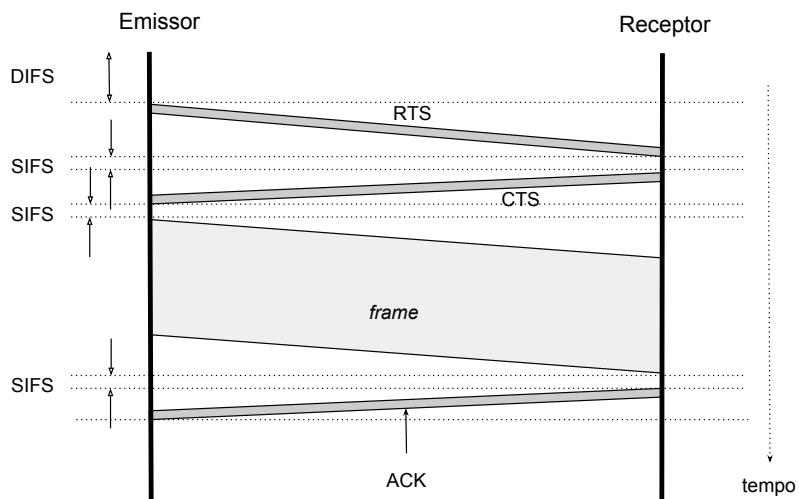


Figura 14.20: Funcionamento do MAC IEEE 802.11 utilizando o mecanismo RTS / CTS

É fácil reconhecer que num quadro de “terminal escondido”, como o da Figura 14.15, este mecanismo ajuda a resolver o problema. Com efeito, se o terminal A enviar um RTS a C, e C responder com um CTS, B não recebe o RTS mas recebe o CTS. Infelizmente, o mesmo não se passa no quadro de um “terminal exposto”. Devido ao *overhead* suplementar e a não ser uma “panaceia para todos os males”, o mecanismo só é usado para *frames* maiores que uma certa dimensão mínima, fixada como um parâmetro do terminal.

O funcionamento distribuído que acabámos de descrever designa-se por modo **DCF** - (*Distributed Coordination Function*). O MAC IEEE 802.11 introduziu também um modo de funcionamento, designado **PCF** - (*Point Coordination Function*), em que o AP funciona como árbitro centralizado da afectação do canal. No entanto, este modo não é utilizado pois é incapaz de funcionar correctamente quando vários APs interferem uns com os outros, o que mostra que a centralização tem os seus limites quando o coordenador não é único.

As normas IEEE 802.11a/b/g incluem vários mecanismos suplementares. Um deles é o da redução automática do débito para compensar uma situação em que o canal tem

muitos erros, ou em que um terminal está muito afastado do AP. Assim, ao invés de transmitir a 54 Mbps (característicos da versão 802.11g), o terminal pode transmitir a 1, 2, 5,5 ou 11 Mbps (característicos da versão 802.11b). A versão 802.11a difere das outras duas por usar a gama de frequências na banda dos 5 GHz e não pode coexistir com as mesmas.

Curiosamente, com adaptação do débito, se cada terminal só poder enviar um *frame* de cada vez, o terminal de maior débito passa a poder usar apenas uma fração minúscula da capacidade do canal. Com efeito, o tempo de transmissão de *frames* a baixo débito ocupa muito mais tempo o canal. Por exemplo, se dois terminais emitirem *frames* de 1 Kbyte, um a 1 Mbps ($T_t \approx 8 \text{ ms}$) e outro a 54 Mbps ($T_t \approx 150 \mu\text{s}$), o débito médio do terminal rápido (excluindo *overheads*) é pouco superior a 1 Mbps também. Por este motivo a norma evoluiu para permitir aos terminais mais rápidos enviarem mais do que um *frame* de cada vez para dividir mais equitativamente o tempo entre os diferentes competidores. Este aspecto será de novo retomado a seguir.

Existem ainda outros mecanismos complementares entre os quais a possibilidade de o AP memorizar durante algum tempo os *frames* que tem para enviar para os diferentes terminais, e avisá-los quando envia os *beacon frames* (que são transmitidos de 100 em 100 ms) se tem *frames* para lhes enviar ou não. Isso permite aos terminais “dormirem” até à emissão do próximo *beacon* caso não tenham tráfego para enviar ou receber no momento.

Em 2005 foi também introduzida uma extensão, designada IEEE 802.11e, que permite utilizar um sistema de prioridades. Esse mecanismo é implementado usando um conjunto alargado de intervalos IFS, que um terminal tem de esperar antes de poder transmitir quando o meio fica livre. Esse conjunto de intervalos de tempo alargado é apresentado na Tabela 14.2

Tabela 14.2: Intervalos de separação entre *frames* da norma IEEE 802.11e, listados por ordem crescente de duração.

Sigla	Nome	Antecede um
SIFS	Short Inter-frame Space	<i>frame</i> de controlo
AIFS1	Arbitration Inter-frame Space 1	<i>frame</i> de alta prioridade
DIFS	Distributed Inter-frame Space	<i>frame</i> de prioridade normal
AIFS4	Arbitration Inter-frame Space 4	<i>frame</i> de baixa prioridade
EIFS	Extended Inter-frame Space	<i>frame</i> emitido após um recebido com erros e NAV distorcido

Formato dos *frames* IEEE 802.11a/b/g

O formato dos *frames* é o apresentado na Figura 14.21. Olhando para o mesmo é fácil de identificar algumas diferenças com respeito aos *frames* no formato IEEE 802.3. Primeiro, o protocolo MAC é mais complexo, e por isso existe um primeiro campo com informação de controlo que permite indicar se o *frame* é de controlo (ACK, RTS, CTS, *beacon*, *association request*, *association response*, etc.) ou simplesmente um *frame* de dados. A este campo segue-se o campo **Duração** que contém a informação NAV, isto é a duração em micro segundos em que emissor vai (ou pretende no caso de um RTS) ocupar o canal.

O campo dos **Dados** (o *payload*) contém os dados propriamente ditos. Apesar de a norma admitir que os *frames* possam ser maiores, o limite imposto pela Ethernet (1500 bytes) é o dominante. Este campo inclui também alguma informação referente ao tipo de informação transportada (*pacote IP*, *outros pacotes*, etc.). A seguir ao campo de Dados vem o campo **CRC**, idêntico ao usado na norma IEEE 802.3.

O campo **Sequência** é necessário para realizar de forma completa o protocolo *stop & wait*. Com efeito, se o ACK se perder, correr-se-ia o risco de introduzir duplicados. Este campo serve para detectar os mesmos e para tornar fiável o protocolo com várias transmissões.

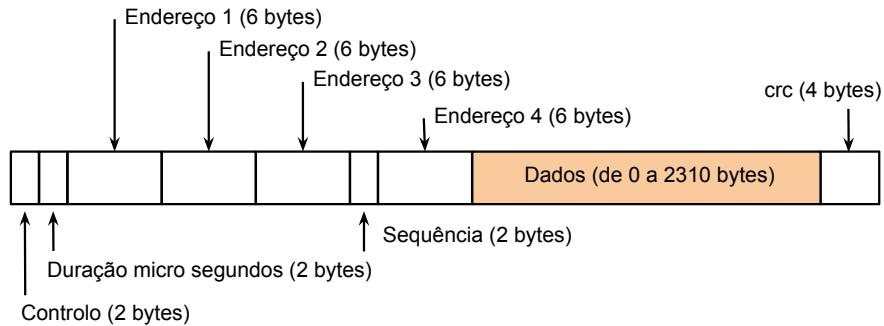


Figura 14.21: Formato dos *frames* IEEE 802.11

A parte menos comum do formato do *frame* tem a ver com a utilização de quatro endereços canal. Um primeiro aspecto a ter presente é que todos os equipamentos ligados a um canal IEEE 802.11 têm de ter endereços MAC, quer os terminais, quer os APs. De facto, só desta forma um AP, ou um terminal, pode decidir, comparando com o seu endereço, se o *frame* lhe é destinado ou não. Os endereços são em tudo compatíveis e similares aos endereços IEEE 802.3.

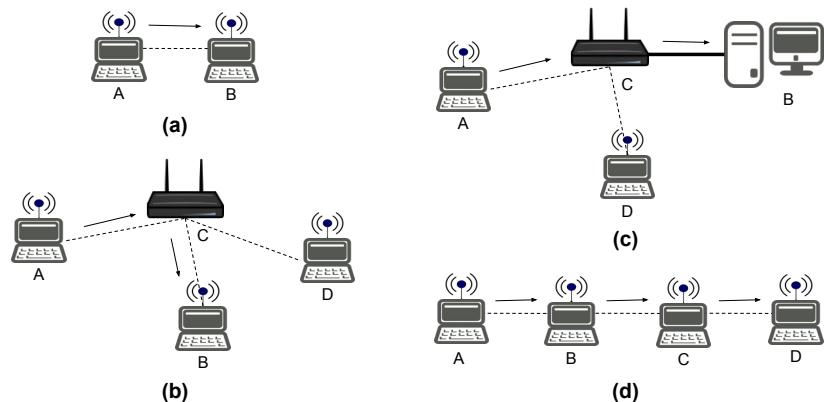


Figura 14.22: Vários cenários de comunicação entre terminais que ilustram a necessidade de usar mais do que dois endereços no cabeçalho dos *frames* IEEE 802.11 (situações (b), (c) e (d))

Quando duas estações estão directamente ligadas ao mesmo meio em modo ad-hoc, só são necessários os endereços origem e destino do *frame*, como é normal. Ver a Figura 14.22 (a). No entanto, quando dois terminais estão ligados através de um AP, o endereço origem é o do emissor, o endereço de destino é o do AP, e o endereço de destino final (um terceiro endereço) é o do outro terminal. Ver a Figura 14.22 (b). A

mesma situação tem lugar se o destino final é um outro equipamento ligado ao AP. Ver a Figura 14.22 (c). Finalmente, em modo ad hoc, como é ilustrado na Figura 14.22 (d), um *frame* enviado de A para D através de B e C necessita de 4 endereços quando transita entre B e C: os origem e destino final e os necessários para transitar de B para C.

Assim, o **Endereço 1** corresponde ao receptor real do *frame*, o **Endereço 2** corresponde ao emissor real do *frame*, o **Endereço 3** corresponde ao receptor final do *frame*, e o **Endereço 4** corresponde ao emissor original do *frame*. Logo, no exemplo (d) acima, quando o *frame* transita de B para C, **Endereço 1 = C**, **Endereço 2 = B**, **Endereço 3 = D** e **Endereço 4 = A**.

Os *frames* IEEE 802.11b/g numa rede que use as duas versões simultaneamente têm um preâmbulo e informação de controlo comuns, codificados sempre com o débito de 1 Mbps, que permite sincronizar o emissor e o receptor, e tem informação que permite interpretar o resto do *frame* (o qual é codificado com o débito próprio do emissor). Este preâmbulo tem a dimensão fixa de 192 bits e a duração fixa de $192 \mu s$, independentemente do débito do canal. No funcionamento em modo infraestrutura, o AP assegura a adaptação do débito do emissor ao usado pelo receptor. Na versão IEEE 802.11g e nas versões mais recentes da norma (IEEE 802.11n e ac), este preâmbulo é substituído por versões mais curtas quando não se pretende garantir a interoperabilidade com as versões anteriores.

Tabela 14.3: Valores de alguns parâmetros das normas IEEE 802.11b/g com garantia de coexistência entre estações a usarem diferentes versões do protocolo MAC.

Parâmetro	Valor / Descrição
Preâmbulo + CTL	192 μ segundos, 192 bits transmitidos a 1 Mbps
<i>Slot time</i>	20 μ segundos
SIFS	10 μ segundos
DIFS	50 μ segundos ($SIFS + 2 \times Slot\ time$)
EIFS	364 μ segundos
CW_{min}	15 <i>Slot times</i> (minimum contention window size)
CW_{max}	1024 <i>Slot times</i> (maximum contention window size)
Cabeçalho MAC	28 bytes
ACK	14 bytes
RTS	20 bytes
CTS	14 bytes

A Tabela 14.3 apresenta alguns valores característicos das normas IEEE 802.11b/g com garantia de interoperabilidade. Recorde-se que o *backoff period* é calculado como sendo $Random(CW)$ onde CW representa a *Congestion Window*. Quando se pretende a coexistência no mesmo canal das diferentes versões, uma parte dos valores da tabela acima é fixo e independente do débito em que o emissor está a funcionar. O preâmbulo do *frame*, acrescentado do DIFS e de um *backoff period* de por exemplo 8 *slots*, totaliza $192 + 50 + 8 \times 20 \approx 400 \mu s$. Se o emissor estivesse a funcionar a 54 Mbps, 1000 bytes seriam transmitidos em $8000/54.10^6 \approx 150 \mu s$ o que dá a dimensão do *overhead* potencial (e.g., superior a 200%) introduzido pelo protocolo MAC sobretudo se o terminal fosse obrigado a emitir um só *frame* de cada vez. Claro que se o emissor emitir com o débito de 1 Mbps, os 1000 bytes são transmitidos em 8 ms, e nesse caso o *overhead* não ultrapassa 6%.

Para compensar o *overhead* significativo a débitos mais elevados, é possível ao emissor transmitir vários *frames* seguidos desde que tenha obtido o acesso ao meio

de comunicação. Esta opção é designada por TXOP (*Transmit Opportunity*) e foi introduzida na extensão 802.11e aos MAC 802.11b/g. A mesma suporta igualmente um mecanismo de ACK conjunto dos *frames*. O valor mínimo de CW também pode ser 7 ou 3 para os *frames* de mais alta prioridade e o mecanismo TXOP é definido de tal forma que o emissor pode ocupar o canal durante 1,5 ou 3 milissegundos seguidos.

Para aumentar o rendimento da rede, sempre que possível, é preferível fixar o funcionamento exclusivo na norma 802.11g, a de maior desempenho. Nesse caso serão usados cabeçalhos mais curtos, que só são legíveis pelas estações a usarem o débito de 54 Mbps. O mesmo se aplica com as versões mais actuais, nomeadamente as 802.11.n e 802.11.ac.

As normas IEEE 802.11 também suportam a emissão de *frames* em *broadcast*, i.e., dirigidos a todos os terminais. Estes *frames* são caracterizados por serem dirigidos ao endereço especial FF:FF:FF:FF:FF:FF e por não existir ACK de resposta, nem ser possível usar o mecanismo RTS/CTS. Por outro lado, em modo compatível com várias versões, são todos emitidos com o débito de 1 Mbps para que todas as estações os possam receber.

Evolução da norma IEEE 802.11

Em 2009 foi introduzida uma nova evolução da norma IEEE 802.11, a versão IEEE 802.11n, também conhecida por MIMO Wi-Fi (*Multiple Input Multiple Output Wi-Fi*), ver a Secção 14.4. Uma das grandes alterações introduzidas consistiu na melhoria do nível físico que passou a poder utilizar vários feixes de ondas em simultâneo, codificações mais sofisticadas, maiores gamas de frequências por canal, assim como várias antenas em conjunto, o que desceu significativamente a taxa de erros e aumentou o débito. Com uma só antena, este sistema pode transmitir com o débito de 70 Mbps. No entanto, usando 4 antenas em cada terminal, canais ocupando uma maior gama de frequências, e se não houver competidores, é possível chegar a 600 Mbps. Este desempenho é, na prática, impossível num ambiente urbano engarrafado por diferentes redes Wi-Fi.

O simples aumento do débito não é suficiente para aumentar significativamente o débito útil real do canal pois o *overhead* do protocolo MAC pode ser significativo como vimos acima. Por esta razão, a norma introduz igualmente mecanismos TXOP, de *frame aggregation*, que permitem enviar vários *frames* de uma só vez, assim como um mecanismo de *block acknowledgement* que permite transmitir de uma só vez ACKs referentes a diferentes *frames*. A dimensão do *frame* máximo aumentou para 8000 bytes, transmitidos em vários fragmentos, e podem ser transmitidos até 64 Kbytes de uma só vez. No entanto, aproveitar todas estas vantagens no nível transporte é um desafio pois a interface tem de ter toda a informação disponível para a transmitir de uma só vez.

A coexistência com terminais a funcionarem nas versões anteriores (e.g., b, g) é possível, mas coloca problemas de desempenho. Por essa razão, o débito máximo teórico só é utilizável num modo em que só existam terminais 802.11/n, a utilizarem um cabeçalho específico deste modo, e a competirem pelo meio em exclusividade.

Em 2014 foi introduzida a versão IEEE 802.11ac, que utiliza a banda entre os 5 e os 6 GHz, melhores técnicas de codificação, mais antenas e mais feixes de ondas. Em condições teóricas óptimas esta versão pode atingir o débito de transmissão de 1,4 Gbps. Para os próximos anos estão previstas novas versões da norma com a forma geral IEEE 802.11a *x*. Estas utilizarão outras bandas de frequências IMS e terão âmbitos, técnicas de codificação e débitos diferentes.

Os canais sem fios são muito atractivos por serem cónmodos de instalar e suportarem directamente sistemas móveis. Por estas razões tem sido realizado um grande investimento em canais sem fios baseados em difusão, *half-duplex*, de grande débito, fáceis de instalar e pouco dispendiosos. Os canais da família de **normas IEEE 802.11**, também conhecidos por **canais Wi-Fi**, satisfazem estes requisitos, utilizam **bandas de frequências livres (IMS)** e, devido à baixa potência de emissão, têm o âmbito de algumas dezenas de metros em meio urbano.

Devido às características do sinal e do meio de propagação que utilizam, estes canais utilizam um **protocolo MAC do tipo CSMA/CA, ACKs dos frames, retransmissão** em caso de falha para compensar os erros, e mecanismos complementares de **alocação e reserva do canal (e.g., NAV e RTS/CTS)**.

Tem havido um grande progresso no débito nominal permitido pelas interfaces IEEE 802.11. No entanto, mesmo depois da introdução de inúmeros melhoramentos e optimizações no protocolo MAC, este continua a **apresentar um overhead significativo**. Adicionalmente, dado que existe uma grande **competição em meio urbano pela utilização das frequências disponíveis**, o débito real extremo a extremo é frequentemente uma fração relativamente pequena do débito nominal teórico.

Uma faceta relevante destes canais é o facto de utilizarem endereços MAC compatíveis com as normas IEEE 802, e ser relativamente fácil integrar vários canais sem fios IEEE 802.11 numa rede a funcionar com *switches* Ethernet como veremos no próximo capítulo. Este facto tem contribuído de forma significativa para o seu sucesso.

14.4 Resumo e referências

Resumo

Os canais baseados em difusão funcionam necessariamente no modo *half-duplex* pois se várias interfaces emitem simultaneamente, sobre um mesmo meio de comunicação partilhado, dá-se uma **colisão** dos sinais que provoca a recepção de *frames* com erros e que por isso ficam inutilizados e têm de ser postos de lado, *i.e.*, ignorados. Para resolver este problema é necessário um mecanismo de **controlo de acesso ao meio**, que é abreviado pela sigla em língua inglesa **MAC – Medium Access Control**.

Existem muitas tecnologias de canais *half-duplex* baseados em difusão que utilizam vários tipos de meios de comunicação como a atmosfera ou meios guiados (cabos de cobre ou de fibra óptica). Nos canais sem fios, ao contrário dos canais baseados em meios guiados, a probabilidade de erros é muito elevada, e pode não ser possível um emissor detectar se o canal está ou não ocupado ou se houve ou não colisão.

Estes protocolos são vários e utilizam diferentes tipos de mecanismos, nomeadamente: **frames de ACK** quando a probabilidade de erro e de colisões é mais elevada, compassos de espera aleatórios (**Random Backoff Periods**) em caso de deteção de colisões ou antes de transmitirem um *frame* pela primeira vez, sondagem do meio de comunicação para verificar se este está livre ou ocupado (**CS - Carrier Sense**), e mecanismos de deteção de colisões (**CD - Collision Detection**). Uma associação específica de diferentes mecanismos é usada em função do meio de comunicação e de outras características necessárias em diferentes contextos. Os protocolos MAC foram normalizados pela IEEE, na família de protocolos com o prefixo **IEEE 802**.

As normas IEEE 802.3 destinam-se a canais guiados em que é possível detectar as colisões, a fiabilidade da transmissão é elevada quando não existem colisões, e é possível reservar o meio através de um pequeno *frame* desde que a distância máxima entre interfaces seja relativamente pequena. Este conjunto de razões permite a utilização de um protocolo MAC do tipo CSMA/CD sem utilização de ACKs.

Uma outra faceta interessante da implementação na Ethernet do protocolo MAC CSMA/CD consistiu na adopção do método *binary exponential backoff* que adapta a duração dos compassos de espera aleatórios em função da maior ou menor probabilidade de colisão.

Modernamente, esta família de normas abrange um conjunto de tecnologias que comportam funcionamento em *half-* e *full-duplex*. Neste último caso, não existe necessidade de utilizar um protocolo MAC, nem limitações de distância, porque o canal funciona em modo ponto-a-ponto entre apenas duas interfaces.

Os canais sem fios são muito atractivos por serem cómodos de instalar e suportarem directamente sistemas móveis. Por estas razões tem sido realizado um grande investimento em canais sem fios baseados em difusão, *half-duplex*, de grande débito, fáceis de instalar e pouco dispendiosos. Os canais da família de **normas IEEE 802.11**, também conhecidos por canais Wi-Fi, satisfazem estes requisitos, utilizam **bandas de frequências livres (IMS)** e, apesar da baixa potência de emissão, têm o âmbito de algumas dezenas de metros em meio urbano.

Devido às características do sinal e do meio de propagação que utilizam, estes canais utilizam um **protocolo MAC do tipo CSMA/CA, ACKs dos frames, retransmissão** em caso de falha para compensar os erros, e mecanismos complementares de **reserva do canal (e.g., NAV e RTS/CTS)**.

Tem havido um grande progresso no débito nominal permitido pelas interfaces IEEE 802.11. No entanto, mesmo depois da introdução de inúmeros melhoramentos e optimizações no protocolo MAC, este continua a apresentar um *overhead* significativo. Adicionalmente, dado que existe uma grande competição em meio urbano pela utilização das frequências disponíveis, o débito real extremo a extremo é frequentemente uma fração relativamente pequena do débito nominal teórico.

Uma faceta relevante destes canais é o facto de utilizarem endereços MAC compatíveis com os das normas IEEE 802.3, e ser relativamente fácil integrar vários canais sem fios IEEE 802.11 numa rede a funcionar com *switches* Ethernet como veremos no próximo capítulo. Este facto tem contribuído de forma significativa para o seu sucesso.

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

Principais conceitos

Acesso Múltiplo (MA - Multiple Access) Canal baseado em difusão com meio de comunicação de acesso múltiplo.

Colisão (collision) Num canal baseado em difusão se dois ou mais emissores emitem sinal simultaneamente, os respectivos sinais entram em colisão e não é possível a um receptor destrinçar e obter os *frames* emitidos pelos emissores.

Controlo de acesso ao meio (MAC – Medium Access Control) Protocolo que permite coordenar os diferentes emissores ligados ao mesmo canal de forma a evitar colisões.

Frames de ACK *Frames* enviadas pelo receptor ao emissor num canal baseado em difusão para permitir correção de erros por retransmissão ao nível do canal. Estes *frames* permitem implementar um protocolo *stop & wait* ao nível canal.

Compassos de espera aleatórios (Random Backoff Periods) Períodos de espera aleatórios usados pelos diferentes emissores antes de emitirem, para tentarem decidir aleatoriamente qual deles tem acesso ao meio de comunicação em primeiro lugar.

Janela de contenção (CW - congestion window) O compasso de espera de cada emissor é dado por: $\text{Random}(0..CW)$.

Janel de contenção crescente exponencial binária (binary exponential backoff)

Janelas de contenção exponencialmente crescentes, sendo multiplicadas por 2 na sequência de tentativas anteriores de resolução das colisões falhadas.

Detecção da portadora (CS - Carrier Sense) Mecanismo que permite detectar no meio de comunicação o sinal de outro emissor.

Detecção da colisão (CD - Collision Detection) Mecanismo que permite ao emissor detectar que a sua emissão colidiu com a de outro emissor.

Evitar a colisão (CA - Collision Avoidance) Mecanismo baseado em compassos de espera aleatórios, antes da primeira tentativa de emissão, para diminuir a probabilidade de que dois ou mais emissores entrem em colisão.

CSMA/CD Protocolo MAC que se baseia na detecção da portadora e de colisões.

CSMA/CA Protocolo MAC que se baseia na detecção da portadora e de tentar evitar colisões pois as estações uma vez iniciada a emissão não conseguem detectar colisões.

Bandas de frequências livres (IMS) (Industrial, Medical, Scientific bands) Gamas de frequências públicas não sujeitas a licenciamento prévio. Por essa razão podem ser utilizadas simultaneamente por diferentes canais, os quais podem entrar em competição pelo acesso ao meio de comunicação.

Referências

A apresentação dos protocolos MAC nesta capítulo não inclui nenhuma tentativa de caracterizar de forma analítica o desempenho dos mesmos. Uma versão clássica deste tratamento, baseada num modelo simplificado do comportamento dos emissores e do canal, é apresentada no Capítulo 1 de [Marsic, 2013]. Esses tratamentos permitem comparar diferentes MACs e ordená-los por eficiência (menor *overhead*). No entanto, na prática, revelam-se demasiado optimistas pois o comportamento real do meio de comunicação e dos terminais são inadequadamente modelizados.

A maioria dos livros sobre redes de computadores contemplam um capítulo sobre o nível físico, onde os canais multi-ponto baseados em difusão, algumas vezes apresentados com o sub-título Redes Locais (LAN - Local Area Networks), são tratados dando um especial realce à problemática do controlo do acesso ao meio de comunicação. Por exemplo, [Tanenbaum and Wetherall, 2011] no Capítulo 4, [Kurose and Ross, 2013] nos Capítulos 5 e 6, [Peterson and Davies, 2012] no Capítulo 2, etc.

O artigo [Metcalfe and Boggs, 1976] apresenta a rede Ethernet. As normas IEEE 802.3 e IEEE 802.11 estão disponíveis no *site* da IEEE. No entanto, existem excelentes livros exclusivamente dedicados ao tema da Ethernet, e.g., [Spurgeon, 2000; Held, 2003], assim como ao tema das redes sem fios em geral, incluindo as normas IEEE 802.11 em particular, como por exemplo [Stallings, 2011]. O livro [Beard and Stallings, 2015] cobre as versões mais recentes dessas normas.

Para o leitor interessado numa primeira abordagem das técnicas de transmissão MIMO (*Multiple Input Multiple Output*), o artigo [Halperin et al., 2010] é um bom ponto de partida.

14.5 Questões para revisão e estudo

1. Verdade ou mentira?

- (a) Num canal ponto-a-ponto 802.3 e *full-duplex*, se o emissor quiser emitir um *frame* pode sempre fazê-lo.

- (b) Num canal partilhado 802.3 baseado em difusão e *half-duplex*, se o emissor quiser emitir um *frame* pode sempre fazê-lo.
 - (c) Num canal 802.11, a funcionar em modo distribuído, as colisões só podem ocorrer durante um período de tempo limitado designado *collision window*.
 - (d) Num canal partilhado 802.3, baseado em difusão e *half-duplex*, o emissor sabe que as colisões só podem ocorrer durante um período de tempo limitado dependente do tempo de propagação de extremo a extremo. Por essa razão, a interface começa geralmente por medir o RTT do canal.
 - (e) Num canal 802.11, a funcionar em modo distribuído, um emissor de um *frame* tem a certeza que o mesmo foi bem recebido sempre que recebe um ACK do receptor.
 - (f) Num canal partilhado 802.3 baseado em difusão e *half-duplex*, o emissor de um *frame*, na ausência de colisão, tem sempre a certeza de que o receptor o recebeu bem.
2. Quais dos seguintes protocolos MAC usam *timeouts* e retransmissão de *frames*: ALOHA, Slotted ALOHA, CSMA, CSMA/CA 802.11 e CSMA/CD 802.3 em modo *half-duplex*?
 3. Qual seria a repercussão sobre o protocolo TCP se o MAC 802.11 deixar de fazer qualquer retransmissão de *frames*?
 4. Verdade ou mentira?
 - (a) Os endereços MAC são hierárquicos e reflectem informação sobre a topologia da rede, i.e., informação sobre a região da rede em que a interface com o endereço se situa.
 - (b) Os endereços MAC são hierárquicos e reflectem o fabricante da interface.
 - (c) Os endereços MAC estão registados no DNS para que seja possível aos clientes encontrarem os endereços dos servidores.
 - (d) À saída da fábrica os endereços MAC são únicos no mundo inteiro.
 - (e) O cabeçalho dos *frames* nos canais multi-ponto não necessitam de conter endereços destino ou origem dos *frames*.
 - (f) Num canal IEEE 802.11 os *frames* só necessitam de dois endereços: destino e origem.
 - (g) Num canal IEEE 802.3 os *frames* só necessitam de dois endereços: destino e origem.
 5. Um protocolo MAC com mecanismos aleatórios realiza a multiplexagem do canal entre n interfaces distintas. Como se pode caracterizar essa multiplexagem: TDM (*Time Division Multiplexing*), FDM (*Frequency Division Multiplexing*) ou multiplexagem estatística?
 6. Um protocolo MAC do tipo CSMA/CA usado num canal sem fios apresenta um *overhead* superior ao de um do tipo CSMA/CD usado num canal com fios, ambos com o mesmo débito de transmissão. Quais dos seguintes factores contribuem mais significativamente para essa diferença?
 - (a) O mecanismo de CS (*Carrier Sense*).
 - (b) Os mecanismos de reserva do canal sem fios (NAV, RTS, CTS).
 - (c) Os endereços nos *frames*.
 - (d) O protocolo de ACK dos *frames*.
 - (e) O algoritmo *binary exponential backoff*.

- (f) O tempo de transmissão dos *frames*.
(g) O mecanismo de arbitragem do acesso inicial ao meio de comunicação.
7. Um conjunto de computadores com interfaces IEEE 802.3 estão interligados através de um equipamento central numa configuração em estrela. Algumas das interfaces têm o débito de 10 Mbps e outras de 100 Mbps, mas todas comunicam perfeitamente. O equipamento central é um *hub* ou um *switch* Ethernet? Justifique.
 8. Duas estações *a* e *b* estão ligadas via um AP numa rede IEEE 802.11 a funcionarem ambas com o débito de 11 Mbps e numa situação em que tem de ser assegurada a compatibilidade com estações a funcionarem com débitos mais baixos (*i.e.*, 802.11b/g em modo compatível). A estação *a* envia um *frame* com 11.000 bits à estação *b* usando o mecanismo RTS/CTS. Qual o menor tempo necessário para que o *frame* chegue ao destino admitindo que em todas as comunicações realizadas não houve contenção no acesso ao meio de comunicação (colisões / repetições). Calcule igualmente o débito extremo a extremo conseguido nessa situação optimista. Para efeitos dos cálculos utilize os dados fornecidos pela Tabela 14.3.
 9. Repita o problema anterior mas numa situação em que as duas estações estão ambas a funcionarem em modo exclusivo 802.11g. Neste caso todo o cabeçalho a restante parte do *frame* são transmitidos a 54 Mbps e, por hipótese, os tempos SIFS e DIFS tomam os valores 9 e 28 μ segundos.
 10. Duas estações *a* e *b* estão ligadas via um AP numa rede IEEE 802.11 a funcionarem ambas com o débito de 54 Mbps. Não existem nem outras estações ligadas à mesma rede, nem outras redes na proximidade. Através do programa **ping** fizeram-se medidas do tempo de trânsito entre *a* e *b*, tendo-se verificado existirem constantemente variações do RTT. Liste todos os factores podem justificar essas variações sabendo que só *a* e *b* estão a usar este canal?
 11. Porque é que o mecanismo RTS/CTS do protocolo MAC IEEE 802.11 é de utilização opcional?
 12. Um conjunto de computadores com interfaces de rede IEEE 802.3 estão interligados através de um equipamento central numa configuração em estrela e o tempo de propagação é desprezável. O tempo de transmissão de um *frame* é T_t . Quanto tempo leva o *frame* a ser recebido pela interface de destino quando o equipamento central é um *hub* Ethernet e quando é um *switch* Ethernet? Justifique.
 13. O protocolo MAC do tipo CSMA/CA da norma IEEE 802.3 não tem a noção de diferentes prioridades dos *frames*. Indique uma forma de introduzir um esquema de prioridades no mesmo. Indique as vantagens e desvantagens da sua proposta.
 14. Porque razão os protocolos MAC do tipo CSMA/CD para canais com fios (*e.g.*, 802.3) não compensam os erros do canal usando ACKs e retransmissões, enquanto que os protocolos MAC do tipo CSMA/CA para canais sem fios (*e.g.*, 802.11) o fazem?
 15. Pretende-se utilizar o protocolo MAC CSMA/CD para controlar o acesso a um canal *half-duplex* baseado em difusão do tipo IEEE 802.3 com o débito de 10 Mbps e um tempo máximo de propagação de extremo a extremo de 50 μ segundos. Indique qual a dimensão em bits do menor *frame* Ethernet que tem de ser emitido nesse canal
 16. Pretende-se utilizar um protocolo MAC do tipo CSMA/CD num canal *half-duplex* do tipo Ethernet com o débito de 10 Mbps e um tempo de propagação de extremo a extremo de 75 micro segundos. Indique quantos bits uma estação

emissora tem de transmitir no mínimo para que o MAC CSMA/CD possa ser usado adequadamente. Comece por calcular a dimensão do *collision slot* para chegar à sua resposta.

17. É possível usar um protocolo MAC do tipo CSMA/CD num canal satélite sem fios com o débito de 1 Mbps e o tempo de propagação extremo a extremo de 250 milissegundos? Justifique.
18. Em geral o nível rede transmite por um canal um pacote de cada vez, fazendo corresponder cada pacote a um *frame* distinto. Existem situações em que seria preferível colocar mais do que um pacote no mesmo *frame*? No entanto, tendo em consideração o nível transporte, em que casos seria isso realista? Justifique as suas respostas.
19. Segundo a norma IEEE 802.3, num canal multi-ponto partilhado *half-duplex* com o débito de 10 Mbps, e com o comprimento máximo de 2500 metros, o *collision slot* tem a duração de $51,2 \mu$ segundos, equivalente à transmissão de 512 bits. Pretende-se introduzir uma variante da norma para canais a funcionarem a 20 Mbps mas de tal forma que o *frame* mínimo continue a ter 512 bits. Com quais das formas a seguir indicadas é isso possível? Repare que pode haver mais do que uma ou várias equivalentes.
 - (a) Aumenta-se o comprimento máximo para 5.000 metros.
 - (b) Diminui-se o comprimento máximo para 100 metros.
 - (c) Diminui-se o comprimento máximo para 1.250 metros
 - (d) Muda-se o *collision slot* para ter a duração de $25,6 \mu$ segundos
 - (e) Muda-se o *collision slot* para ter a duração de $5,2 \mu$ segundos
20. Para interligar 12 computadores um repetidor (*hub*) foi substituído por um *switch* Ethernet com 12 portas com o mesmo débito das portas do *hub*. Internamente o *switch* Ethernet só consegue transferir um *frame* de cada vez entre cada duas portas. Associadas a cada uma das suas portas existem duas filas de espera, uma para os *frames* recebidos e outra para os *frames* a enviar. Indique se há ou não, potencialmente, alguma melhoria do débito extremo a extremo entre os computadores com esta substituição. Justifique a sua resposta.
21. Verdade ou mentira?
 - (a) O protocolo MAC IEEE 802.11 suporta o envio de *frames* em *broadcast*. O emissor recebe tantos ACKs quantos os receptores.
 - (b) Quando um emissor IEEE 802.11b/g (modo misto) envia um *frames* em *broadcast*, usa sempre na emissão o débito máximo da sua interface.
 - (c) Quando um emissor IEEE 802.11 envia *frames* de grande dimensão em *broadcast*, usa sempre o mecanismo RTS/CTS.
 - (d) A introdução de um mecanismo de NACK (*Negative ACK*) (e.g., 802.3) num protocolo MAC do tipo CSMA/CA permitiria ganhos de eficiência.

Capítulo 15

Encaminhamento com base em inundação

Increasingly, people seem to misinterpret complexity as sophistication, which is baffling – the incomprehensible should cause suspicion rather than admiration.

– Autor: *Niklaus Wirth*

No capítulo anterior vimos como fazer uma rede com um único canal multi-ponto baseado em difusão. Naturalmente, este tipo de rede tem necessariamente um âmbito limitado. No caso de redes Wi-Fi, este restringe-se a algumas dezenas de metros “sob telha” (*indoor*), um pouco mais ao ar livre dado existirem menos obstáculos à propagação. Neste capítulo continuaremos a ver soluções tão simples quanto possível para o encaminhamento de pacotes em redes de computadores, mas de forma a alargarmos o seu âmbito.

A solução terá de ser viável para uma rede arbitrária que interliga um conjunto de computadores através de canais e comutadores de pacotes. Alguns dos canais são ponto-a-ponto, enquanto que outros são multi-ponto. Essa rede deverá envolver caminhos alternativos e ter a configuração de uma malha arbitrária. Estes caminhos alternativos permitem que a rede continue a fornecer serviço mesmo quando alguns dos caminhos se tornam momentaneamente indisponíveis devido a avarias.

Os computadores comunicam através da troca de pacotes. Os pacotes têm um cabeçalho com pelo menos os endereços destino e origem. Os endereços dos computadores são todos diferentes uns dos outros e cada comutador tem ele próprio um endereço. Cada comutador que recebe um pacote consulta o endereço de destino, e caso este não o indique a ele próprio, decide o que fazer ao pacote e encaminha-o para o destino, geralmente via outro comutador.

Neste capítulo vamos explorar o mais simples dos métodos de encaminhamento de pacotes, que é conhecido pela designação de encaminhamento por inundação: quando um comutador recebe um pacote, se este não lhe é destinado, envia uma cópia do mesmo por todos os canais que conhece, excepto por aquele pelo qual o recebeu. Como é evidente, este método assegura que uma cópia do pacote chegue ao destino, mas eventualmente mais do que uma. Infelizmente também sobrecarrega a rede com um conjunto de cópias inúteis do mesmo pacote. Parece não ser uma muito boa ideia. No entanto, como veremos a seguir, o método de base pode ser complementado com um conjunto de mecanismos que o tornam mais realista e até de utilização comum.

A secção que se segue apresenta o método de encaminhamento com base em inundaçãO e estuda os mecanismos necessários para tornar este método de encaminhamento realista e adequado em certos contextos e para algumas aplicações específicas. A seguir veremos como este método de encaminhamento pode ser usado pelos *switches* Ethernet para implementar uma rede simples de interfaces IEEE 802 (.3, .11, etc.) a comunicarem directamente. Finalmente, analisaremos como estas redes são utilizadas na prática hoje em dia, e como podem mascarar as falhas que ocorreram e adaptarem-se aos computadores que se ligam e desligam dinamicamente da rede.

15.1 Encaminhamento com base em inundaçãO

O algoritmo é, como já se disse, muito simples: quando um nó da rede (o termo genérico para um computador ou um comutador) recebe um pacote, analisa o seu endereço de destino, e se este for diferente do seu, envia uma cópia do mesmo por cada uma das suas interfaces, excepto por aquela pela qual o pacote foi recebido. Se o pacote é dirigido ao nó, este processa-o sem continuar a encaminhá-lo (excepto se o endereço de destino for o endereço de *broadcasting*, caso em que a inundaçãO deve prosseguir). Se um nó só tem um canal, e recebe um pacote que não lhe é dirigido, o algoritmo determina que o nó destrua esse pacote pois não pode voltar a enviá-lo pelo canal pelo qual o recebeu.

Esta forma de encaminhamento chama-se **encaminhamento por inundaçãO** (*flooding*) pela analogia sugerida: a rede é inundada pelo pacote.

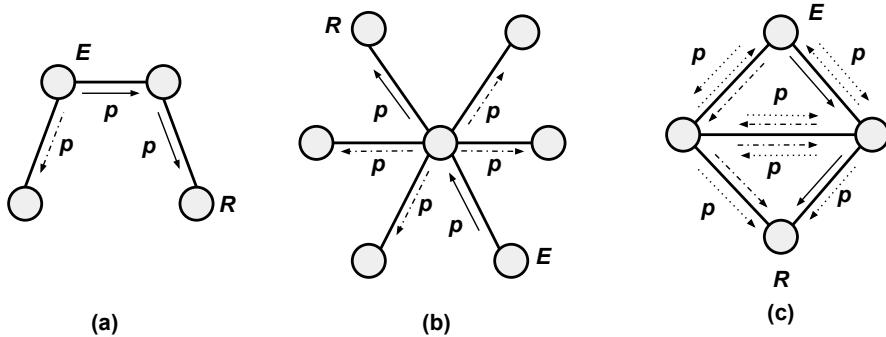


Figura 15.1: Encaminhamento por inundaçãO do pacote p do emissor E para o receptor R . As setas correspondem a cópias do pacote: as a negro são as necessárias; as a ponto e traço são inúteis; e as a ponteado são algumas das cópias duplicadas.

A Figura 15.1 mostra a utilização do encaminhamento por inundaçãO de um pacote (p) emitido pelo nó origem ou emissor (E), para o nó destino ou receptor (R), em três redes diferentes. Nos primeiros dois casos ((a) e (b)), a inundaçãO encaminha o pacote p de E para R correctamente, mas introduz desperdício pois há cópias inúteis do pacote que são encaminhadas para outros nós que acabam por as descartar por não terem mais alternativas de emissão. No caso (c), não só são introduzidas cópias inúteis, mas também pacotes duplicados (que os nós não identificam como tal) pelo que continuam a encaminhá-los sem fim, conduzindo a mais cópias inúteis e a outros duplicados, e o processo continua eternamente. Com efeito, o pacote p acaba por dar a volta em ciclo e retorna ao emissor inicial E , este não o reconhecendo como sendo duplicado do anterior, volta a enviá-lo como da primeira vez, e todo o processo recomeça sem

nunca acabar. Desde que a rede tenha ciclos, *i.e.*, caminhos que começam e acabam no mesmo nó, o processo não tem fim e não só o algoritmo não termina, como introduz uma **tempestade de difusão** (*broadcast storm*) que satura e inutiliza a rede.

A diferença entre os dois primeiros casos e o terceiro reside no facto de que nos primeiros as redes não têm ciclos, pelo que a inundação introduz cópias inúteis, mas não duplicados. No terceiro caso a rede tem ciclos e a inundação introduz cópias inúteis, duplicados, e uma inundação sem fim que provoca o colapso da rede.

O algoritmo de **encaminhamento por inundação** é muito simples: quando um nó recebe um pacote, analisa o seu endereço de destino, e se este for diferente do seu, envia uma cópia do mesmo por cada uma das suas interfaces, excepto por aquela pela qual o pacote foi recebido.

Este algoritmo consegue encaminhar um pacote até ao seu destino mas infelizmente, conduz ao envio de pacotes inúteis nas redes estruturadas em árvore, e a um fenómeno de colapso da rede designado por **tempestade de difusão** (*broadcast storm*) em redes com ciclos.

Veremos a seguir que existem métodos para obviar os problemas da inundação. Comecemos pelo mais grave, a tempestade de difusão no caso em que a rede tem ciclos.

Deteção e supressão de pacotes duplicados

Uma primeira forma de acabar com os pacotes duplicados, ou de pelo menos minorar a sua presença, consiste em introduzir um TTL (*Time To Live*) como o dos pacotes IP. Este campo do cabeçalho do pacote contém um contador decrescente dos nós já atravessados pelo mesmo.

Quando o nó inicial emite pela primeira vez p , inicializa o seu **ttl** com um valor positivo, por exemplo 10. Sempre que um pacote chega a um nó, este decrementa o seu **ttl** e, se este valer 0, descarta o pacote e não o processa. Senão, processa o pacote com o novo valor do **ttl**. Se houver necessidade de enviar mais do que uma cópia do pacote, o valor do **ttl** de cada uma dessas cópias é sempre o mesmo. Ou seja, o **ttl** de p só pode decrescer. Naturalmente, o método baseado num TTL permite limitar a tempestade, mas não a estanca à nascença e, adicionalmente, o emissor não conhecendo a configuração da rede, usa um valor de TTL suficiente para não impedir nenhum pacote de chegar ao destino. Esse valor tem de ser necessariamente muito alto para a maioria dos casos pois o emissor não consegue saber qual seria o valor adequado. Dada uma rede, o valor por omissão do maior TTL necessário é o tamanho do maior caminho na mesma, também chamado o diâmetro da rede. A utilização de um TTL é uma segurança contra algo que funcione mal e acaba por descartar, mais ou tarde ou mais cedo, pacotes em ciclo pois esses pacotes constituem um perigo fatal para a rede se não forem bloqueados.

Existe, no entanto, um método mais complexo, mas também mais eficaz, de estancar os duplicados logo à nascença. Trata-se de um método que permite aos nós detectarem que um pacote é um duplicado.

Esse método consiste em colocar uma marca em cada um dos pacotes emitidos. Sempre que um nó recebe um pacote, começa por verificar se já viu um pacote com uma marca igual. Se sim, o pacote é um duplicado, senão, regista a sua marca e processa-o. Mas como arranjar uma marca distinta para cada pacote? Um método relativamente simples consiste em usar como marca dos pacotes o par (endereço do emissor original do pacote, número de sequência). O número de sequência é colocado no cabeçalho pelo emissor original sempre que emite um novo pacote. Nenhum outro nó altera esse

número de sequência. Desta forma, à entrada da rede, cada pacote recebe uma marca distinta de todos os outros, uma espécie de “número fiscal do pacote”.

Se o número de sequência usado em diferentes pacotes por cada emissor for sempre crescente, cada nó só tem que memorizar, para cada nó da rede, o último número de sequência recebido num pacote emitido originalmente por esse nó. Se os números de sequência não forem sempre crescentes, cada nó tem de memorizar tantos números distintos quantos os pacotes. Ou seja, com números de sequência crescentes, a tabela de números de pacotes já recebidos é $O(\text{numero de nós da rede})$,¹ no caso contrário é $O(\text{número de pacotes recebidos})$.

No parágrafo que se segue vamos discutir uma variante do algoritmo de deteção de duplicados quando o objectivo é receber as mensagens por uma ordem crescente das marcas. Dado que esse algoritmo é utilizado a diferentes níveis da arquitectura das redes, utilizaremos o termo mensagem e não pacote para designar as unidades de informação trocadas pelos nós da rede.

Difusão fiável de mensagens

O algoritmo de inundação implementa directamente a comunicação de um para muitos ou difusão (*broadcasting*), *i.e.*, a possibilidade de uma mensagem ser enviada a todos os nós da rede. Basta que nesse caso o endereço de destino seja especial, *i.e.*, o endereço de *broadcasting*. É claro que os problemas dos duplicados e dos pacotes inúteis se continuam a colocar e é necessário usar um mecanismo que os permita resolver. Pelo menos o primeiro para evitar uma tempestade de difusão. A utilização de números de sequência sempre crescentes para suprimir duplicados pode também ser usada para implementar uma versão do protocolo de difusão designada **algoritmo de difusão fiável**.

Este algoritmo permite que as mensagens enviadas em difusão por cada um dos nós sejam recebidas por todos os outros pela ordem com que cada uma delas foi emitida. No entanto, dependente da versão do algoritmo, existem várias variantes de relação de ordem. Na versão aqui apresentada, essa relação de ordem só garante a ordenação das mensagens emitidas pelo mesmo nó, não a das mensagens emitidas por diferentes nós. Do ponto de vista das garantias de recepção de todas as mensagens, existem também várias versões deste algoritmo, cada uma com diferentes garantias. Aquela que vamos apresentar é das mais simples e implementa apenas algumas das garantias possíveis, nomeadamente que um nó só considera as mensagens mais recentes, pela ordem de emissão, mas pode saltar por cima de alguma mensagem ainda não chegada se no entretanto recebeu outra mais recente.

O algoritmo é um algoritmo de inundação em que cada nó emite uma sequência de mensagens numeradas com números de sequência sempre crescentes e, desde que a rede seja conexa (*i.e.*, existe pelo menos um caminho entre quaisquer dois nós), as mensagens emitidas por um nó são recebidas pelos diferentes nós por uma ordem compatível com a ordem com que foram emitidas.

Para implementar esta versão da difusão fiável, a troca de mensagens entre nós é feita usando uma variante do protocolo *stop & wait*. A diferença relativamente a este é que quando o nó receptor recebe uma mensagem que já possui, porque o emissor está atrasado nas mensagens que já recebeu, ao invés de lhe enviar um ACK, envia-lhe uma cópia da mensagem mais recente que possui. O algoritmo garante que os diferentes nós recebem as mensagens por uma ordem compatível com a ordem com que foram emitidas por cada emissor, mas não garante que o nó tenha acesso à sequência integral das mensagens enviadas. O algoritmo é apresentado na Listagem 15.1 para uma rede só com canais ponto-a-ponto. Repare-se que o nó não analisa o endereço de destino pois pressupõe-se que este é o endereço de *broadcasting*.

¹ No caso de a rede poder trocar a ordem de entrega dos pacotes, ou perder alguns, é preferível memorizar intervalos de números de sequência já recebidos de cada nó da rede.

Listing 15.1: Pseudo-código do algoritmo de difusão fiável - tratamento local da mensagem m

```

1 // Message database with the most recent message from each participant
2 // node in the network, that has been received by this node. Usage:
3 //
4 //     msgDatabase.put(message)
5 //
6 //     message = msgDatabase.getMessage(node)
7 //
8 // returns the most recent message from node or null if none
9 // has been received from that node
10
11 processMessage ( message msg, interface in ) {
12     seq = msg.getSequence()
13     node = msg.getNode()
14     lastSeen = msgDatabase.getMessage(node)
15     if ( lastSeen != null ) lastSeq = lastSeen.getSequence()
16     if ( lastSeen == null ) { // first message from this node
17         msgDatabase.put ( msg )
18         send ( in, ACK )
19         flood ( in, msg )
20     }
21     else if ( lastSeq < seq ) { // more recent message
22         msgDatabase.put ( msg )
23         send ( in, ACK )
24         flood ( in, msg )
25     }
26     else if ( lastSeq == seq ) { // duplicate
27         send ( in, ACK )
28     }
29     else if ( lastSeq > seq ) { // late message
30         send ( in, lastSeen )
31     }
32 }
```

O algoritmo apresentado é um dos mais simples de difusão fiável. Este complica-se mais se se pretender que cada nó receba exactamente todas as mensagens emitidas por cada outro nó. No caso de se pretender também uma ordem total de todas as mensagens (comum a todos os nós), o algoritmo necessário é também mais complicado.

Como veremos no Capítulo 16 este algoritmo é usado para replicar a configuração de uma rede de forma fiável entre um conjunto de comutadores de pacotes. Para esse efeito, o algoritmo utiliza um conjunto de medidas complementares, nomeadamente as seguintes. A base de dados de mensagens tem um TTL associado a cada nó e se este não origina mensagens durante um período alargado (*e.g.*, algumas dezenas de minutos), o mesmo é “esquecido”. Desta forma, é mais fácil de garantir que os identificadores dos nós são sempre necessariamente diferentes uns dos outros. Também, quando um nó teve uma avaria e desaparece da rede mas volta a integrá-la antes de ser esquecido, assim que se manifestar, os seus vizinhos têm oportunidade de o actualizar sobre o último número de sequência que utilizou, sob pena de as suas mensagens, por terem um número de sequência muito atrasado, serem ignoradas até voltarem a ter um número de sequência maior que o último que utilizou antes de ter “morrido e ressuscitado”.

O algoritmo de encaminhamento por inundaçāo pode ser complementado com um mecanismo baseado em números de sequência que permite detectar as mensagens duplicadas e parar o processo de inundaçāo. Apesar de este mecanismo não evitar o envio de pacotes inúteis, permite implementar **algoritmos de difusão fiável baseados em inundaçāo**.

Supressão de pacotes inúteis - aprendizagem pelo caminho inverso

Os métodos acima apresentados permitem minorar ou acabar de todo com os pacotes e mensagens duplicadas, mas não permitem minorar ou acabar com os pacotes inúteis que a inundaçāo introduz, que são bem evidentes nos casos (a) e (b) da Figura 15.1.

Nos contextos em que é possível acabar com os duplicados de forma segura, existe um algoritmo que permite evitar a introdução de pacotes inúteis. Esse algoritmo é designado por **aprendizagem pelo caminho inverso** (*backward learning*), ou filtragem pelo caminho inverso. Quando a rede não tem ciclos é mais fácil perceber o algoritmo e é também mais fácil ficar convicto da sua eficácia. Uma rede sem ciclos é designada em teoria de grafos uma árvore. Numa tal rede, entre quaisquer dois nós *E* e *R* só existe um caminho, e por isso a inundaçāo não introduz duplicados, apenas pacotes inúteis.

O algoritmo de aprendizagem pelo caminho inverso baseia-se na existência em cada nó de uma tabela de nós conhecidos. Nessa tabela, associados a cada nó conhecido, existem dois valores: o número da interface que permite chegar a esse nó e um TTL.

Sempre que um nó recebe um pacote *p*, analisa nessa tabela a entrada correspondente ao endereço origem desse pacote: *p.origem()*. Se essa entrada estiver vazia, quer dizer que o nó nunca recebeu um pacote que tenha sido originalmente emitido por *p.origem()* e não o conhece. Por isso cria uma nova entrada na tabela e coloca na mesma o número da interface pela qual *p* foi recebido. Adicionalmente o TTL associado a cada entrada é inicializado com um valor normalizado, por exemplo 120 segundos. Caso o nó já seja conhecido, a entrada na tabela é refrescada com o número de interface e o TTL é de novo inicializado.

Mas para que serve a tabela? Quando um nó recebe um pacote *p* dirigido ao nó *p.destino()*, o receptor vai ver se já conhece esse nó consultando a sua tabela. Se sim, então já sabe porque interface o pode alcançar, senão só lhe resta fazer inundaçāo. Com efeito, se todos os canais forem bidireccionais, numa rede em que só existe um caminho entre cada dois nós, o caminho inverso permite chegar ao nó origem.

O processo é ilustrado na Figura 15.2. Inicialmente o nó de comutação de pacotes não conhece nada sobre os computadores que lhe estão ligados. Mais tarde, o com-

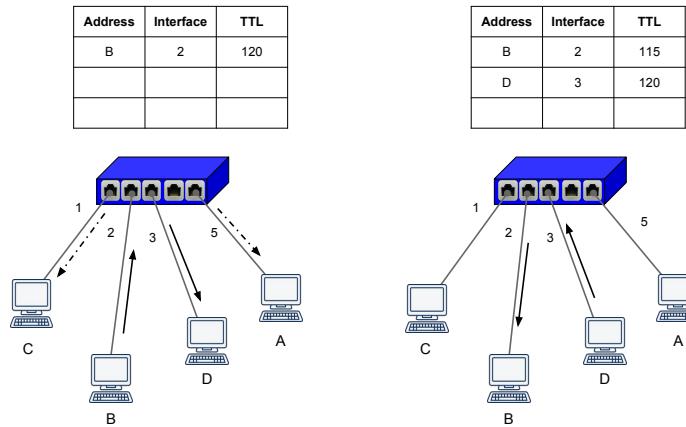


Figura 15.2: Encaminhamento por inundação e aprendizagem pelo caminho inverso: B envia um pacote a D, depois D responde a B. As setas a ponto e traço representam mensagens inúteis.

putador com o endereço B envia um pacote para o computador com o endereço D. O nó de comutação, não conhece D pelo que só lhe resta fazer a inundação, mas antes ficou a saber que B está ligado à sua interface 2 e regista esse facto na tabela com o TTL 120. Passados 5 segundos, D responde a B, mas desta vez o nó de comutação já conhece a localização de B pelo que não necessita de fazer inundação. Aproveita no entanto para registar a interface que dá acesso a D.

Este algoritmo de inundação e aprendizagem pelo caminho inverso é usado pelos nós de comutação (*switches*) Ethernet para encaminharem *frames* entre os computadores que lhe estão ligados. O algoritmo que executam figura na Listagem 15.2 que apresenta o seu pseudo código. A tabela dos *switches* toma então o nome de *MAC Address Table* ou simplesmente `macTable` na listagem.

Listing 15.2: Pseudo-código do algoritmo de tratamento por um nó de um pacote *p* recebido pela interface *in* através do algoritmo de inundação com aprendizagem pelo caminho inverso.

```

1  TTL = 120
2  processPacket ( packet p, interface incoming ) {
3      macTable.put(p.getOrigin(), incoming, TTL)
4      // is p locally addressed ?
5      if ( p.getDestination == self.getID() ) {
6          // packet p got to its destination
7          locally process packet p
8          return // done
9      }
10     interface outgoing = macTable.get(p.getDestination)
11     if ( outgoing == null ) flood(p) // not in macTable
12     else if ( outgoing != incoming ) outgoing.send(p)
13     // else ignore p
14 }
```

Repare-se que o endereço de *broadcasting* nunca é endereço origem pelo que nunca estará presente na tabela, logo todos os pacotes dirigidos a esse endereço são necessariamente *flooded*, o que implementa naturalmente a difusão.

15.2 Comutação Ethernet (Ethernet *switching*)

Quando os *switches*² Ethernet³ foram introduzidos, foi necessário decidir como seria possível introduzir o conceito de comutação de pacotes para encaminhamento de *frames* Ethernet entre vários canais de difusão. Por diversas razões, provavelmente ligadas a não se pretender alterar o cabeçalho dos *frames* Ethernet, nem introduzir complexidade suplementar nos comutadores, optou-se por não usar nenhum algoritmo de deteção de duplicados (com números de sequência) ou mesmo de simples limitação do seu tempo de vida (com TTLs) e por essa razão optou-se por só admitir a introdução destes equipamentos em redes com a configuração de uma árvore.

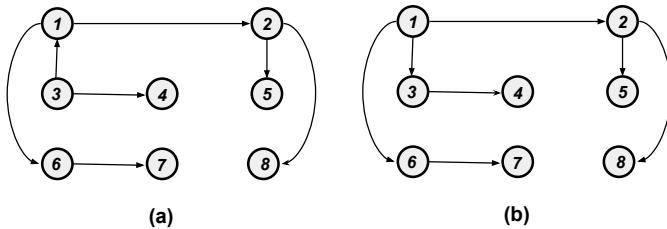


Figura 15.3: A difusão não introduz duplicados numa rede estruturada em árvore. No caso (a) o emissor é o nó 3 e no caso (b) é o nó 1.

Numa rede estruturada em árvore, o algoritmo de inundação permite o encaminhamento de um nó para outro nó, assim como a difusão, sem necessidade de recorrer à deteção de duplicados porque a inundação não os introduz numa rede sem ciclos.

Por outro lado, o mecanismo de **aprendizagem ou filtragem pelo caminho inverso (backward learning)** com base nos endereços MAC das interfaces Ethernet dos computadores, permite restringir significativamente o envio de *frames* inúteis.

Como os *frames* IEEE 802.11 podem ser facilmente transformados em *frames* IEEE 802.3 pelos APs, a rede pode conter comutadores de pacotes (*switches* Ethernet e APs), canais ponto-a-ponto, canais multi-ponto (IEEE 802.3 ou 802.11) e interfaces *full* e *half-duplex*.

A Figura 15.4 mostra um exemplo de uma rede Ethernet, organizada em árvore, composta por vários tipos de canais, *switches*, um AP e um *hub*, que formam uma rede em que todos os computadores podem endereçar sem problemas *frames* uns aos outros, incluindo de um para todos. Uma rede deste tipo costuma chamar-se uma **rede Ethernet comutada** ou **switched Ethernet network** na terminologia em língua inglesa.

O algoritmo de aprendizagem pelo caminho inverso tem por objectivo implementar uma optimização da inundação, transformando-a num encaminhamento por um

² Como já foi várias vezes referido, um *switch* Ethernet pode ser designado na língua portuguesa por comutador Ethernet ou nó de comutação de *frames* Ethernet. No entanto, por esses termos não serem utilizados pelos profissionais de redes, continuaremos a utilizar termos em língua inglesa.

³ Na verdade os *switches* Ethernet foram antecedidos por outros equipamentos chamados *Bridges* que tinham grosso modo as mesmas funcionalidades que os *switches* mas que usavam outro tipo de canais de interligação ponto-a-ponto.

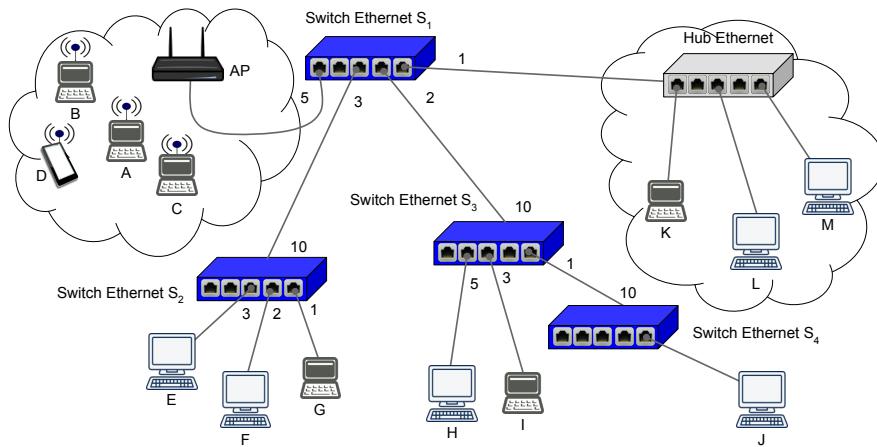


Figura 15.4: Rede com diferentes tipos de comutadores e de canais com encaminhamento por inundaçao e aprendizagem pelo caminho inverso.

caminho directo. Com efeito, na sua ausência os *frames* chegariam ao destino mas a rede conteria muito tráfego inútil. Com este algoritmo, é feita uma primeira inundaçao quando um endereço MAC de destino é desconhecido mas, a partir daí, deixam de ser enviados *frames* inúteis para a origem desse pacote.

O teste na linha 12 da Listagem 15.2 assegura que se um *frame* chegou ao *switch* pela interface por detrás da qual já se sabe estar o seu destino, então este já o deve ter recebido e é inútil continuar a encaminhá-lo. O próprio conceito de inundaçao determina que um pacote nunca deve ser enviado pelo canal pelo qual chegou ao nó de comutação. Por outro lado, se a essa interface estivesse ligado um canal multi-ponto, sem esse teste o algoritmo provocaria a recepção de duplicados pelo destino.

Desde que a rede seja organizada em árvore, o algoritmo de aprendizagem funciona perfeitamente. Por exemplo, se o computador E enviar um *frame* ao computador M, como o destino não é conhecido, o *frame* inunda a rede e todos os *switches* e o AP (o AP também executa o algoritmo), passarão a saber porque porta poderão aceder a E. Por exemplo, o *switch* S₄ passará a saber que E está por detrás da sua porta 10. Após todos os computadores enviarem *frames* uns aos outros, todos serão conhecidos por todos os nós de comutação. Por exemplo, o *switch* S₁ saberá que os computadores H, I e J estarão por detrás da sua porta 2. A maneira mais simples de um computador se dar a conhecer a todos os equipamentos de comutação (*switching*) é enviar um *frame* dirigido ao endereço de *broadcast*.

Um aspecto cuja discussão é interessante tem a ver com o dinamismo dos computadores e a gestão dos TTLs. Para que servem os TTLs das entradas das tabelas usadas pelo algoritmo de aprendizagem pelo caminho inverso? Existem várias razões para a sua presença. A mais óbvia tem a ver com o facto de que os computadores se podem desligar e, nesse caso, é inútil ter entradas nas tabelas ocupadas com os seus endereços (apesar de isso poder poupar algumas inundações). Assim, se o computador deixar de emitir pacotes, as entradas que lhe dizem respeito acabarão por expirar e deixarão de ocupar espaço.

A outra razão está ligada com o facto de que pode haver mais computadores do que entradas disponíveis nas tabelas. Quando houver falta de espaço para introduzir mais um endereço MAC na tabela, é necessário retirar uma das entradas para criar espaço. Uma solução possível consiste em retirar da tabela a entrada há mais tempo sem ser usada, *i.e.*, aquela cujo TTL é o mais baixo. Esta razão tem hoje em dia

bastante menos razão de ser em redes pequenas pois a maioria dos *switches* Ethernet têm tabelas com pelo menos centenas de entradas.

No entanto, é preciso ter em atenção que o processamento de um *frame* deve ser tão rápido quanto possível quando comparado com o tempo da sua transmissão. Ora um *frame* com 1.000 bits é transmitido em $1\ \mu$ segundo a 1 Gbps ($1000/10^9 = 10^6$). Por esta razão, os *switches* usam circuitos integrados especiais⁴ com memórias muito rápidas para implementar as tabelas de endereços MAC e os algoritmos que as manipulam. Essas memórias rápidas são caras e consomem muita energia. Os *switches* de gama mais alta têm tabelas destas com 64 K entradas o que aumenta bastante o seu custo.

Finalmente, o TTL está também ligado ao facto de que os computadores hoje em dia são na sua maioria portáteis e portanto podem mudar de AP ou de porta a que estão ligados. Quando isso acontece, a tabela fica desactualizada e poderia deixar de ser possível enviar *frames* ao computador que migrou de localização na rede. Até o TTL expirar, os *frames* dirigidos a esse endereço MAC seriam dirigidos para o local errado.

Na verdade, isto só seria verdade se esse computador só recebesse tráfego e não enviasse nenhum. Na prática, o computador que mudou de localização acabará por enviar *frames*, e assim que o fizer, começa a actualizar as entradas que lhe dizem respeito nas diferentes tabelas. Tudo depende do destino dos *frames* que enviar. A maneira mais eficaz de assegurar a actualização total e instantânea consiste em o computador que migrou enviar um *frame* em difusão assim que se voltar a ligar à rede. Desta forma todas as entradas são imediatamente actualizadas. Actualmente, a maioria dos sistemas de operação dos computadores e os APs, sempre que uma interface muda de localização, fazem automaticamente esse *broadcast*.

As razões que acabámos de apresentar mostram que continua a ser importante usar um TTL associado a cada entrada das tabelas de endereços, mas deixou de ser muito importante discutir qual o valor com que este deve ser inicializado. Basta assegurar que este não induz inundações inúteis com demasiada frequência.

15.3 Árvores de cobertura e STP

Na prática, quando uma rede tem um pequeno âmbito e está sobredimensionada para o tráfego que a utiliza, é relativamente fácil organizá-la como uma árvore. Nessa rede o tempo de propagação de extremo a extremo é desprezável, a probabilidade de avarias é baixa, e o tráfego não seguir pelo caminho mais directo não é grave. A Figura 15.5 representa uma rede em que dois canais multi-ponto, a distâncias geográficas de vários quilómetros, estão interligados por dois canais ponto-a-ponto. O objectivo é garantir que em caso de avaria de um desses canais é possível continuar a comunicar usando o outro, pois a reparação de canais de longa distância pode levar bastante tempo. Infelizmente a configuração tem um ciclo e essa rede de *switches* Ethernet entraria em colapso com uma tempestade de difusão (*broadcast storm*).

Para resolver este problema foi introduzido um protocolo que a todo o momento calcula de forma distribuída uma árvore de cobertura da rede e desliga os canais que não fazem parte dessa árvore. No entanto, se um dos canais que faz parte da árvore se avariar, é recalculada uma nova árvore e a rede é reconfigurada automaticamente.

Dado um grafo conexo, *i.e.*, em que existe pelo menos um caminho entre quaisquer dois nós, uma **árvore de cobertura** do mesmo é um seu sub-grafo contendo todos os nós e que é uma árvore, *i.e.*, no qual só existe um caminho entre quaisquer dois nós.

⁴ Estes circuitos designam-se por CAMs - *Content Addressable Memories*.

O protocolo que permite a um conjunto de *switches* Ethernet calcular uma árvore de cobertura e apenas operar de acordo com a mesma, chama-se **SPT - Spanning Tree Protocol** e foi normalizado pela norma IEEE 802.1D.

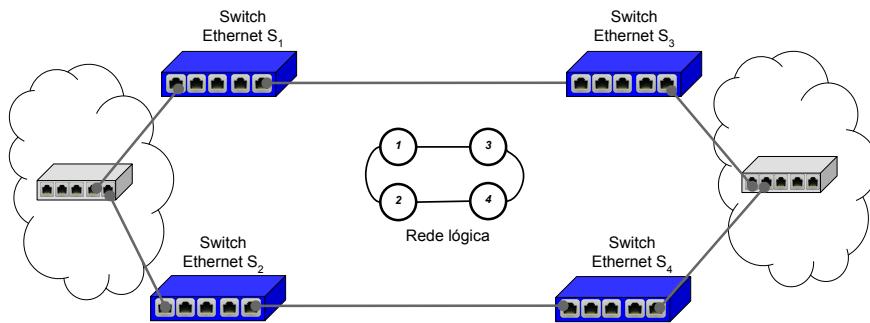


Figura 15.5: Rede com uma configuração necessária para efeitos de tolerância a falhas mas que introduz ciclos.

O protocolo STP

O protocolo seleciona na rede os canais que fazem parte da árvore e coloca em inactividade os que não fazem, ver a Figura 15.6. Nela, a parte (a) mostra a rede completa e a (b) uma árvore de cobertura calculada pelo STP. Após a avaria do canal que liga o nó 1 ao 2, o STP reconfigura a rede e adopta a árvore de cobertura (c), desactivando uns canais e activando outros, de forma a manter de novo uma configuração em árvore que cobre todos os nós. Nesta secção usaremos com frequência os termos originais em língua inglesa introduzidos com a descrição do protocolo.

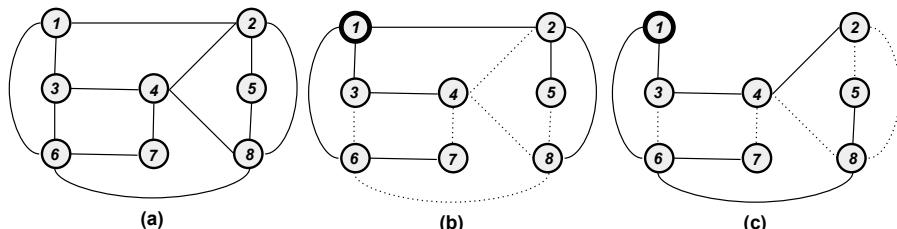


Figura 15.6: Papel do protocolo STP: (a) rede completa, (b) uma sub-rede calculada pelo STP, (c) outra sub-rede calculada pelo STP após a avaria do canal que liga os nós 1 e 2.

O STP calcula uma árvore começando por eleger um *switch* com o papel de raiz da árvore. No caso das redes das figuras 15.6 e 15.7 a raiz é o nó 1. Todos os outros *switches* se ligam a essa árvore. A inundação vai depois funcionar enviando *frames* na direção da raiz, no sentido ascendente, e *frames* no sentido das folhas, no sentido descendente, ver a Figura 15.7. Uma forma de visualizar este tipo de encaminhamento é imaginar que o *switch* no qual cada *frame* emitido tem origem, funciona como raiz

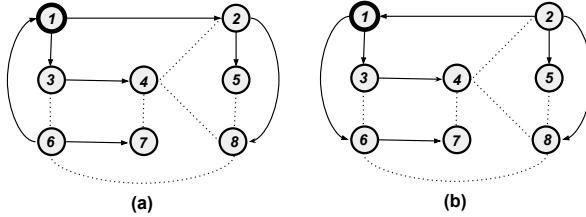


Figura 15.7: Uso da árvore STP para inundaçāo. A raiz da árvore STP é o *switch* 1. Encaminhamento de um *frame* emitido pelo *switch* 6 (a) e pelo *switch* 2 (b) usando a árvore calculada pelo STP.

de uma árvore para efeitos da inundaçāo. Repare-se, no entanto, que estas árvores, pelo menos nos exemplos apresentados, são bastante desequilibradas.

O protocolo funciona com base nos conceitos a seguir introduzidos.

Switch ID é o identificador de cada *switch*. É composto pela concatenação de 12 bits de prioridade com os 48 bits do endereço MAC do *switch*.

Root switch ou *switch* raiz é o *switch* cujo identificador é o mais baixo em toda a rede. O administrador da rede pode forçar um certo *switch* a ser a raiz atribuindo-lhe uma prioridade mais baixa que a dos outros, senāo a raiz é simplesmente escolhida com base nos endereços MAC, de forma mais ou menos aleatória. O *root switch* será a raiz da árvore calculada pelo STP.

Port ID é o identificador de cada porta de um *switch*. Todas as portas de um *switch* têm identificadores distintos.

Link cost é o custo de cada canal. O custo varia de 1 a 200.000.000. O custo dos diferentes canais é definido pela norma e é inversamente proporcional ao seu débito (quanto menor o débito, maior o custo). Por exemplo, o custo de um canal a 1 Gbps é 4, a 100 Mbps é 19, a 10 Mbps é 100, etc.

Root path cost é o custo do caminho (o somatório dos custos dos canais que este atravessa) que liga um dado *switch* ao *root switch*. Em cada *switch*, a porta que o liga ao canal onde comea o caminho para a raiz é designada a **Root port**. Só pode haver uma destas portas por *switch*.

Designated port é a porta de um canal responsável por encaminhar *frames* até à raiz.

Este conceito é especialmente importante em portas ligadas a canais multi-ponto ao qual estão ligados diversos *switches*. Este aspecto será retomado a seguir.

A árvore seleccionada pelo STP é caracterizada por ser uma árvore de caminhos mais curtos (*i.e.*, mais “baratos”) desde a raiz até cada uma das folhas. Nos casos em que existirem diversas alternativas de caminhos mais curtos, o STP dispõe de regras para optar de forma unívoca por um deles.

O protocolo desenvolve-se em várias fases, nomeadamente: 1. eleição da raiz; 2. calcular os caminhos mais curtos da raiz para cada *switch*; 3. bloquear os canais que não fazem parte desses caminhos; e 4. activar posteriormente os restantes canais.

O STP baseia-se num único tipo de mensagens, designadas *frames* BPDU⁵ e descritas na Figura 15.8. Através de um BPDU o emissor comunica aos seus vizinhos

⁵ Os primeiros equipamentos que usaram este protocolo eram designados por pontes (*bridges*). Esta designação tinha por origem a ideia de que o seu principal papel era a interligação

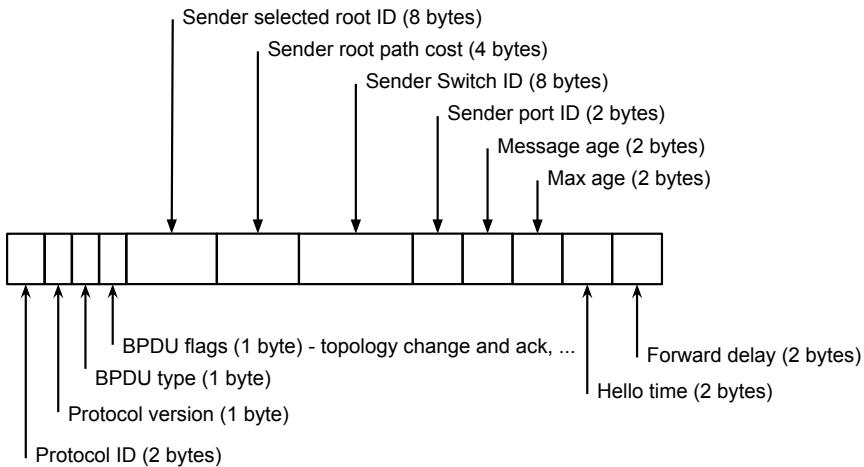


Figura 15.8: STP BPDUs (Bridge Protocol Data Unit)

qual é, na sua opinião, a raiz, a que distância (custo) esta está, qual o seu ID e a porta porque enviou este BPDU, assim como diversos parâmetros de funcionamento do protocolo. Alguns dos campos já foram referidos ou serão referidos a seguir mas de qualquer forma seguem-se algumas breves observações sobre alguns deles. Os campos ID da raiz segundo o emissor do BPDU e ID do emissor são expressos 8 bytes cada: 2 bytes com a prioridade (0x8000 por omissão mas este valor pode ser alterado pelo gestor da rede) mais 6 bytes com o MAC Address. O campo com o ID da porta do emissor é expresso em 1 byte para a prioridade (pode ser alterada pelo gestor) mais 1 byte para o número de porta. O campo *maximum age* é o valor do temporizador que dá o alarme caso não sejam recebidos BPDUs. O campo *hello time* é o tempo que separa a emissão de cada BPDU pela raiz (geralmente 2 segundos) e o campo *forward delay* é o tempo que leva uma porta a transitar para o estado *forwarding*. Todos os tempos são codificados em múltiplos de 8 ms (1/256 segundos).

Eleição da raiz e seleção do caminho mais curto para a mesma

Numa primeira fase, após a inicialização de um *switch*, ou na sequência da deteção de um evento que obriga a uma reconfiguração, a rede elege o *root switch*. Para esse efeito, todos os *switches* se candidatam a ser raiz e começam a difundir periodicamente (*e.g.*, a cada 2 segundos), por inundação, BPDUs em que se consideram raiz da rede. Estes BPDUs iniciais contêm como raiz o ID do emissor, visto que este se considera a si próprio raiz, e por isso também anuncia que o seu custo até à raiz é 0.

Quando um *switch* recebe BPDUs de outros, compara o identificador da raiz recebida com o identificador do *switch* que considera raiz (ou seja ele próprio no início). Se o ID da raiz recebido for maior, ignora a mensagem recebida, se for menor, passa a considerar o ID anunciado como sendo o ID da nova raiz, e o canal pelo qual recebeu este BPDU, o canal da *root port*, *i.e.*, o canal do caminho mais curto para a raiz. Se for igual, mas recebeu este BPDU de uma porta diferente da que considera como *root port*, muda de *root port* caso o custo para chegar à raiz seja menor pela nova porta, ou sendo igual, se o ID do vizinho, ou o da porta pelo qual este lhe enviou o BPDU, são inferiores.

de vários canais multi-ponto entre si como ilustrado pela Figura 15.5. As pontes foram completamente substituídas pelos *switches* Ethernet e o termo caiu em desuso, no entanto, o termo BPDU (Bridge Protocol Data Unit) está na norma STP e mantém-se em uso.

Quando um *switch* recebe um BPDU enviado pelo *switch* que eleger (ou acaba de eleger) como raiz e que lhe chegou pela *root port*, actualiza o custo até à raiz e a idade do BPDU, e difunde aos vizinhos por inundação um novo BPDU com a raiz e o seu custo para lá chegar. Senão, o BPDU recebido é ignorado pois não acrescentaria nenhuma informação relevante aos outros *switches*. Só o *switch* raiz é que continua a enviar periodicamente BPDUs, os restantes apenas difundem por inundação os BPDUs recebidos do *switch* que consideram raiz.

Este procedimento elege uma raiz única e permite igualmente a cada *switch* seleccionar o caminho mais curto para a mesma. Nos casos em que há diversas alternativas de caminho mais curto, as regras sobre os identificadores dos *switches* e das portas resolvem a ambiguidade.

A Figura 15.9 ilustra algumas dessas situações. No caso (a) o *switch* 1 é eleito para raiz (menor identificador) e o *switch* 4 escolhe o 2 para chegar à raiz visto que o 3, que oferece um caminho de igual custo, mas tem um identificador maior que o 2. No caso (b) existem dois canais entre os *switches* 1 e 3, mas aquele que do lado do *switch* 1 está ligado ao número de porta mais baixo é seleccionado visto que os dois canais têm o mesmo custo. No caso (c) entre os *switches* 1 e 3 é escolhido o canal de mais baixo custo visto que o outro tem um custo superior.

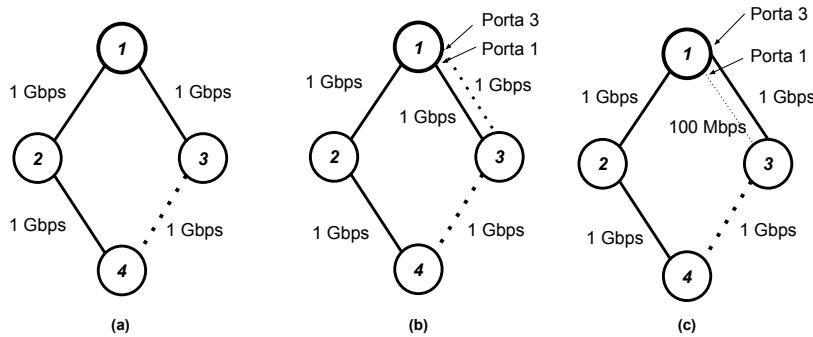


Figura 15.9: Escolha do caminho mais curto em casos aparentemente ambíguos. Cada *switch* é representado por um círculo com o seu ID no centro.

O protocolo STP baseia o seu funcionamento na inundação periódica da rede com *frames* de controlo chamados BPDUs, emitidos inicialmente por todos os *switches*, mas mais tarde apenas pelo *switch* eleito como raiz.

Cada participante na inundação, altera os BPDUs de forma que torna possível realizar a eleição da raiz e determinar o caminho mais curto até à mesma. Este caminho também é o mais curto desde a raiz até cada *switch* se todos os canais tiverem o mesmo débito nos dois sentidos, o que é o caso mais comum.

Escolha das *designated ports*

Em todos os canais que fazem parte de caminhos mais curtos para a raiz (os únicos canais que ficarão activos) é necessário escolher uma *designated port*. Caso o canal seja ponto-a-ponto e *full-duplex* só existem duas portas ligadas ao mesmo e a *designated port* é a que está no extremo oposto à da *root port*.

No entanto, no caso de canais multi-ponto, o problema é mais delicado pois ao mesmo canal podem estar ligados mais do que um *switch* e eventualmente cada um terá caminhos mais curtos com o mesmo custo para a raiz.

Nesse caso é necessário eleger apenas uma *designated port* entre os diferentes *switches* e portas ligados ao mesmo canal. Esta porta será a porta do *switch* com a menor distância até à raiz, ou com o menor ID se existirem vários à mesma distância da raiz, ou com o menor ID de porta se existirem várias do mesmo *switch* ligadas a esse canal, e que nenhuma é *root port* desse *switch* pois uma porta não pode ter os dois papéis. A eleição segue o mesmo processo que o usado na eleição da raiz. Cada *switch* considera cada porta candidata como *designated port* mas à medida que vai recebendo por ela BPDUs dos outros, ou se não receber nenhum, pode tomar uma decisão de acordo com as regras indicadas acima.

Uma *designated port* assume a responsabilidade de enviar para o canal os *frames* recebidos da raiz e de enviar para a raiz os *frames* recebidos do canal. As outras portas não eleitas devem ficar bloqueadas se forem de *switches* que executam o protocolo STP. De facto, qualquer nó ligado à mesma rede (computador, *switch* sem suporte de STP, *hub*, etc.) apenas passa os BPDUs sem os interpretar e não gera BPDUs. Como os BPDUs são sempre dirigidos ao endereço de *broadcast* nunca são interceptados por *switches* sem suporte de STP.

A Figura 15.10 ilustra o processo e põe em evidência que sem esta precaução seria introduzida uma tempestade de difusão. Sem a noção de *designated port* um *frame* emitido por um dos computadores chegaria em duplicado aos diferentes *switches* pois o canal multi-ponto, necessário para chegar aos computadores, introduziria igualmente um canal entre os *switches* 2 e 3.

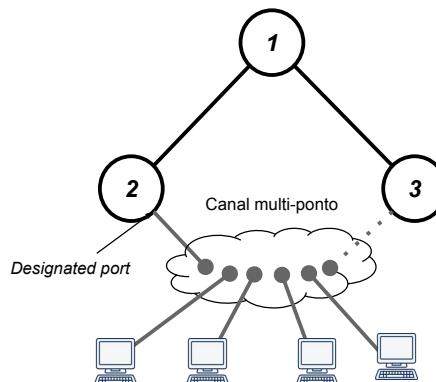


Figura 15.10: Eleição da *designated port* num canal multi-ponto

Estabilização do protocolo e início do encaminhamento

Enquanto o STP não estabiliza as suas decisões, não podem ser encaminhados *frames* de dados pois pode existir o risco de os encaminhar através de ciclos. Assim, só depois de o STP estabilizar é que os *switches* passam a encaminhar os *frames* que recebem via as portas eleitas como *root ports* ou *designated ports*. Até lá as portas estão num estado que se chama *blocked* (bloqueado) em que só encaminham BPDUs. Quando estão activas, passam ao estado *forwarding* (a encaminhar *frames* de dados).

Mas como saber se o STP já estabilizou? Para mais esta decisão tem de ser tomada por cada *switch* de forma independente. Na versão IEEE 802.1D a decisão é tomada tendo em consideração a “ausência de alterações nos últimos tempos”. Durante a execução do STP as portas estão todas no estado bloqueado, i.e., sem receberem ou

enviarem *frames* diferentes dos BPDUs. Após passar um período de tempo desde que teve lugar a última modificação do estado da raiz ou de uma porta, o *switch* considera que a situação está estável e passa o estado das portas eleitas como *root* ou *designated* de *blocked* para *forwarding*. Esse período de tempo tem normalmente a duração de várias dezenas de segundos pelo que em caso de necessidade de reconfiguração, a rede fica inoperacional durante muito tempo, o que é mau.

A implementação consiste simplesmente em inicializar todas as portas no estado bloqueado, e passar todas as portas ao estado bloqueado quando se dá uma reconfiguração, uma alteração da raiz ou do caminho para a mesma, e só passar uma porta ao estado *forwarding* após esta ter estado nesse estado sem alterações durante um período de tempo significativo. Actualmente um valor típico para este compasso de espera é de 15 segundos.

Existem diversas situações que desencadeiam a execução de uma reconfiguração da rede, a qual passa por todos os *switches* se considerarem de novo raiz e começarem uma nova eleição da raiz e uma nova escolha dos caminhos mais curtos para chegar à mesma.

Se a raiz corrente tiver uma avaria ou se for desligada da rede, os outros *switches* deixam de receber os seus anúncios periódicos. Como conhecem o limite de tempo sem receberam BPDUs periódicos (através do parâmetro **Maximum Age** dos BPDUs da raiz), mudam de estado e uma nova eleição é desencadeada. A mesma começa com o primeiro *switch* que detectou a situação a enviar um BPDU pelo qual se considera raiz e com a *flag Topology Change* posicionada.

A reconfiguração da rede também pode ser desencadeada por um *switch* que deteta uma avaria do canal ligado à sua *root port*. Eventualmente, a partir de informação memorizada sobre outros anúncios recebidos, ou quando os receber, o *switch* pode mudar para outro caminho para a raiz e escolher uma nova *root port*. Se isso não for possível, o *switch* considera-se raiz e inicia o processo de eleição. Se a avaria tiver lugar numa *designated port* serão os outros *switches* que desencadearão o processo.

Um *switch* STP também só activa uma sua porta ligada a um computador, ou a toda uma zona sem STP, depois do compasso de espera que permite passar uma *designated port* ao estado de *forwarding*. Nesse caso, a ligação de um computador a uma dessas portas implica esperar vários segundos até a porta estar activa. Isto tem de ser assim pois não sabe se por detrás dessa porta está ou não outro equipamento com outras ligações que poderiam introduzir ciclos. Muitos *switches* admitem a parametrização de portas reservadas a computadores como estando fora do processo STP e portanto não sujeitas a este tipo de compassos de espera aborrecidos.

Chama-se convergência de um protocolo de encaminhamento ao tempo necessário para, na sequência de uma qualquer alteração de estado dos nós ou canais, todos os nós de comutação tomarem em consideração essas alterações, reconfigurarem a rede, e voltarem a colocá-la em funcionamento de acordo com a nova configuração.

O protocolo STP converge muito lentamente pois faz depender a convergência de, na sequência da deteção de uma alteração, passar um tempo de espera longo sem se produzir nenhuma nova alteração, nem as inundações periódicas, que se processam sempre ao mesmo ritmo, implicarem novas alterações.

Durante a convergência, o encaminhamento tem de ser bloqueado, porque não existem nenhum outros mecanismos (*e.g.*, TTLs ou deteção de duplicados) que estancassem uma eventual tempestade de difusão (*broadcast storm*).

Observações finais sobre o STP

A capacidade de uma rede Ethernet gerida por STP é em geral subaproveitada pois muitos canais disponíveis têm de ficar bloqueados para que a rede tenha a configuração de uma árvore. Não é possível aproveitá-los para fornecerem um melhor caminho entre a origem e o destino, nem fazer distribuição de carga.

De facto os *frames* só seguem por “caminhos bons” enquanto esses caminhos coincidirem com os caminhos mais curtos entre a origem e o destino. Isso só se verifica em caminhos que “subam” ou “desçam” a árvore, mas não nos que incluem os dois tipos de trajectos. Por exemplo, se a maioria do tráfego for gerado de e para um servidor, idealmente este deve estar ligado à raiz da rede. O parâmetro prioridade dos *switches* permite ao gestor da rede forçar que um dado *switch* seja a raiz da rede independentemente do seu endereço MAC.

Por outro lado, se a rede tiver uma grande quantidade de computadores, serão realizadas inundações com bastante frequência, o que também não é um bom aproveitamento da sua capacidade.

No entanto, estas redes têm muitas vantagens pois são simples e geridas de forma automática, *i.e.*, com pouca ou nenhuma intervenção de um gestor. Por outro lado, quando utilizadas em áreas confinadas (dentro de um edifício ou de um grupo de edifícios) e com capacidade abundante para as necessidades, os inconvenientes assinalados têm um menor impacto.

O grande inconveniente nessa situação são os compassos de espera em caso de necessidade de reconfiguração, que levam a que a rede esteja inoperacional durante muitos segundos. Vários serviços críticos (voz, vídeo, *etc.*) não suportam essa espera. Por esta razão foi introduzida uma versão de STP, chamada Rapid STP - RSTP (norma IEEE 802.1w de 2001) que consegue reconfigurar a rede em menos de 1 segundo quando a avaria é de um canal. A norma RSTP foi incorporada na norma IEEE 802.1D na versão de 2004 que continua a ser compatível com a versão do STP descrita nesta secção.

Todos os algoritmos e protocolos que apresentámos para suporte do funcionamento de uma rede Ethernet comutada (*switched*), *i.e.*, baseada em inundação com aprendizagem pelo caminho inverso e no protocolo STP, pressupõem que todos os equipamentos funcionam de acordo com o esperado pelos protocolos e não violam ou modificam os seus endereços MAC.

Caso contrário é possível bloquear a rede enviando falsos BPDUs, ou até desviar o tráfego, por exemplo enviando periodicamente *frames* em *broadcasting* com endereços MAC origem falsos de forma a atrair para si o tráfego que lhes era dirigido. Deixada entregue a si própria, sem outros mecanismos de segurança, uma tal rede torna-se muito frágil.

15.4 Virtual Local Area Networks (VLANs)

O êxito das redes Ethernet comutadas permitiu montar redes destas com dezenas e dezenas de nós de comutação, abrangendo vários edifícios e servindo para a interligação de milhares de computadores. No entanto, esta abrangência levanta pelo menos dois problemas.

O primeiro tem a ver com segurança pois todos esses computadores podem endereçar-se mutuamente e, como foi referido acima, é possível, por exemplo, atacar a rede enviando *frames* em difusão com endereços MAC origem falsos para desviar o tráfego. Por outro lado, sempre que há uma inundação, as interfaces recebem *frames* que lhes

não são destinados e *frames* inúteis invadem toda a rede o que prejudica o seu funcionamento. Por isso é também possível provocar um ataque de negação de serviço enviando muitos *frames* em difusão.

É portanto mais prudente seccionar a rede em vários segmentos distintos e protegidos uns dos outros, organizando a rede em sub-redes disjuntas e independentes, uma para cada fim específico. Por exemplo, numa universidade poder-se-iam criar redes distintas para os estudantes usarem livremente, outras para os laboratórios de aulas, outras para os docentes, outras para a administração, etc.

Isso pode ser feito usando cabos UTP e equipamentos distintos, mas só funcionaria bem se houvesse a possibilidade de facilmente ligar computadores às diferentes redes nos mesmos locais, o que implicaria a multiplicação de cabos e equipamentos pelas diferentes áreas da universidade. O mesmo aconteceria numa empresa de grande dimensão: qualquer reorganização da distribuição dos funcionários pelos diferentes gabinetes, implicaria mexidas nos cabos e nos equipamentos da rede.

O segundo problema tem a ver com o facto de estas redes se basearem no algoritmo de inundação. Quanto maior a rede maior o número e âmbito das inundações e maior o desperdício da maioria delas. Idealmente é também desejável confinar as inundações a diferentes sub-redes.

Os *switches* Ethernet mais sofisticados incluem mecanismos de seccionamento da rede em áreas disjuntas, cada uma das quais se chama uma **rede lógica ou virtual ou VLAN (Virtual Local Area Network)**⁶. As VLANs são independentes entre si e não podem comunicar usando os *switches*. No entanto, para montar várias VLANs não é necessário dispor de equipamentos afectados a cada uma delas. Tudo é feito por software e pode ser reconfigurado dinamicamente em função das necessidades.

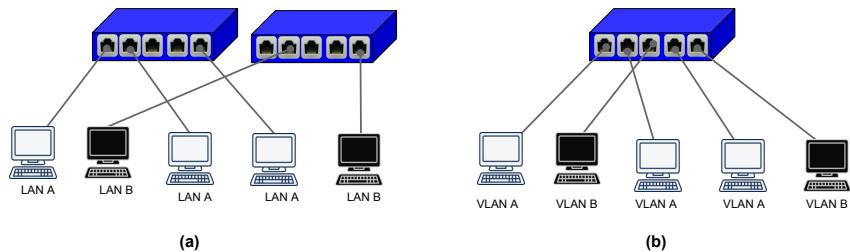


Figura 15.11: Duas redes distintas fisicamente (a) e uma implementação equivalente através de um único *switch* e duas VLANs distintas (b).

A Figura 15.11 ilustra a solução. À esquerda, parte (a), são implementadas duas redes distintas usando dois *switches*. À direita, parte (b), são implementadas duas VLANs usando um *switch* com suporte de VLANs. Cada uma das portas desse *switch* pertence a uma VLAN distinta. Isso quer dizer que qualquer *frame* recebido numa dessas portas só pode ser encaminhado (por inundação ou usando as tabelas de comunicação com endereços MAC) para portas pertencentes à mesma VLAN. O mesmo se aplica a *frames* enviados para o endereço de broadcast pois o âmbito de uma inundação está sempre restringida às portas que pertencem à VLAN do emissor.

É fácil de imaginar como implementar VLANs com um só *switch* pois este conhece de que porta um *frame* foi originado, qual a VLAN a que esta pertence, e pode restringir as portas em que é realizada a inundação às portas associadas à mesma VLAN. Mas como generalizá-lo ao conjunto da rede de tal forma que a mesma VLAN possa abranger computadores ligados a diferentes *switches*? A Figura 15.12 mostra uma

⁶ O termo VLAN tem por origem o facto de que o nome tradicional das redes baseadas em canais de difusão se designarem por LANs - Local Area Networks.

rede física em que há computadores pertencentes à mesma VLAN ligados a diferentes *switches*. Como pode um *switch* que recebe um *frame* de outro *switch* saber a que VLAN o mesmo se destina?

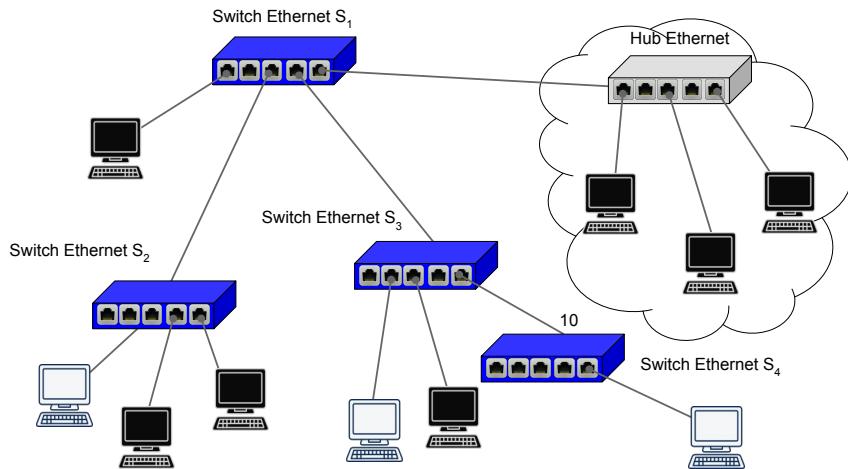


Figura 15.12: Rede com vários *switches*. Cada um deles liga computadores de duas VLANs distintas. Computadores a claro pertencem a uma VLAN, e os outros pertencem a outra VLAN. Os computadores não necessitam de “saber” o que é uma VLAN e os computadores ligados ao mesmo canal multi-ponto pertencem necessariamente à mesma VLAN.

A maneira mais simples de resolver esse problema consiste em expandir o cabeçalho IEEE 802.3 com um campo que indica a que VLAN o *frame* pertence. Esse novo cabeçalho, foi definido na norma IEEE 802.1Q de 1998 e está representado na Figura 15.13.

Esse novo campo, também designado por **marca de VLAN ou VLAN tag**, é colocado no cabeçalho de um *frame* pelo *switch* com suporte de VLANs que o recebeu pela primeira vez, e é suprimido pelos *switches* que o enviam por uma porta ligada a equipamentos que não suportam VLANs. Desta forma não é necessário inutilizar as interfaces e os equipamentos que não suportam nem conhecem VLANs. O mesmo problema já surgiu na interligação de APs IEEE 802.11 com redes IEEE 802.3. Os equipamentos de interligação têm de modificar o cabeçalho para o adaptar ao domínio para que os *frames* transitam.

Assim, as portas dos *switches* passam a ser de dois tipos, dependentes do canal que está ligado às mesmas. Se a porta apenas pode receber *frames* sem VLAN tag, então a porta está geralmente associada ela própria a uma VLAN e o *switch* coloca a tag respectiva em cada *frame* que recebe pela mesma. Por outro lado, só *frames* dirigidos a essa VLAN podem ser enviados por essa porta depois de a tag ter sido retirada. As outras portas, tipicamente as que interligam os *switches*, só podem ser atravessadas por *frames* com VLAN tags para que o destino saiba como os processar. Esses canais e portas são geralmente designados por canais de tipo *trunk* ou de interligação.

O mecanismo de aprendizagem pelo caminho inverso também pode ser expandido e, quando um *switch* recebe um *frame*, para além de colocar na tabela de endereços MAC a porta de acesso ao equipamento com esse endereço, também pode anotar a

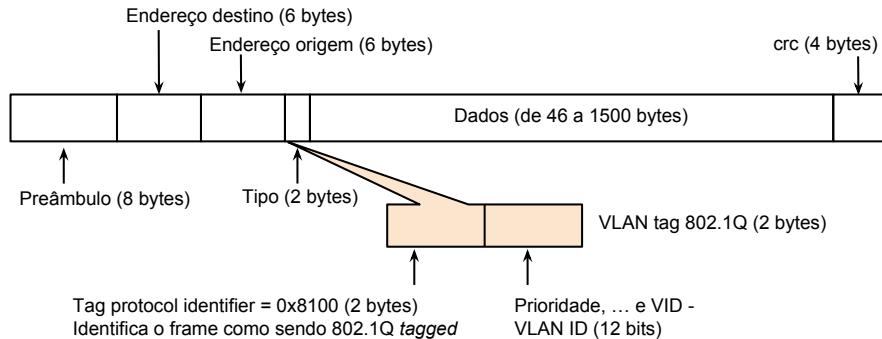


Figura 15.13: Formato dos *frames* Ethernet que incluem o cabeçalho IEEE 802.1Q. O valor que indica a presença deste foi escolhido de tal forma que permite distinguir um *frame* 802.1Q de um convencional pois é diferente de todos os valores legais do campo *Tipo*.

que VLAN o mesmo pertence. Assim, um *frame* enviado a partir de uma VLAN a que o destino não pertence, pode imediatamente ser suprimido.

As VLANs são um poderoso instrumento para o administrador de uma rede a seccionar em sub-redes distintas, isoladas por motivações de segurança, mas também para reduzir o âmbito das inundações. Adicionalmente, os *switches* mais sofisticados, para além de disporem de mecanismos de gestão de VLANs por software, também permitem parametrizar o STP para construir, ou não, árvores de cobertura distintas por VLAN. É também possível “podar” as árvores de cobertura das diferentes VLANs de forma a evitar que *switches* sem ligações a uma VLAN, recebam BPDUs referentes a essa instância de STP, ou inundações com *frames* destinados a essa VLAN.

As redes Ethernet equipadas com *switches* com suporte de VLANs ficam dotadas de uma mecanismo muito flexível que permite seccionar a rede em sub-redes distintas e isoladas, cada uma das quais associada a uma VLAN distinta.

Este mecanismo é fundamental para organizar uma rede complexa em sub-redes distintas por razões de segurança, isolamento e desempenho. Os APs com suporte de 802.11 também podem ter suporte de VLANs e separarem os utilizadores e os SSIDs por VLANs distintas.

Um domínio com VLANs inclui sempre uma VLAN especial, a VLAN 1, à qual pertencem todos os *switches* para efeitos de gestão dos mesmos. Por omissão, sem indicações contrárias, todas as portas e todos os equipamentos com suporte de VLANs pertencem à VLAN 1. Ou seja, mesmo que os *switches* suportem STP e VLANs, o administrador que não necessite das VLANs pode continuar a montar e gerir uma rede completamente auto configurável visto que o STP para a VLAN 1 está sempre activo por omissão.

15.5 Resumo e referências

Resumo

O algoritmo de **encaminhamento por inundação** é muito simples: quando um nó recebe um pacote, analisa o seu endereço de destino, e se este for diferente do seu, envia uma cópia do mesmo por cada uma das suas interfaces, excepto por aquela pela qual o pacote foi recebido. Este algoritmo consegue encaminhar um pacote até ao seu destino mas, infelizmente, conduz ao envio de pacotes inúteis nas redes estruturadas em árvore, e a um fenómeno de colapso da rede designado **tempestade de difusão (broadcast storm)** em redes com ciclos.

O algoritmo pode ser complementado com um mecanismo baseado em números de sequência que permite detectar os pacotes duplicados e parar o processo de inundação. Apesar de este mecanismo não evitar o envio de pacotes inúteis, permite implementar **algoritmos de difusão fiável baseados em inundação**.

Numa rede estruturada em árvore, o algoritmo da inundação permite o encaminhamento de um nó para outro, assim como a sua difusão, sem necessidade de recorrer à deteção de duplicados porque estes não aparecem neste tipo de redes. Apenas serão enviados pacotes inúteis. No entanto, o mecanismo usado nas redes Ethernet comutadas conhecido como **aprendizagem ou filtragem pelo caminho inverso (backward learning)** com base nos endereços MAC das interfaces Ethernet dos emissores, permite restringir significativamente o envio de *frames* inúteis.

Dado um grafo arbitrário, uma árvore de cobertura do mesmo é um seu sub-grafo contendo todos os nós e que é uma árvore, *i.e.*, no qual só existe em caminho entre quaisquer dois nós. O protocolo que permite a um conjunto de *switches* Ethernet calcular uma árvore de cobertura e apenas operar de acordo com a mesma, chama-se **SPT - Spanning Tree Protocol**, norma IEEE 802.1D.

O protocolo STP baseia o seu funcionamento na inundação periódica da rede com *frames* de controlo chamados BPDUs, emitidos inicialmente por todos os *switches*, mas mais tarde apenas pelo *switch* eleito como raiz. Cada participante na inundação altera os BPDUs de forma que torna possível realizar a eleição da raiz e determinar o caminho mais curto até à mesma.

Chama-se **convergência de um protocolo de encaminhamento** ao tempo necessário para, na sequência de uma qualquer alteração de estado dos nós e canais, todos os nós de comutação tomarem em consideração essas alterações, reconfigurarem a rede, e voltarem a colocá-la em funcionamento de acordo com a nova configuração.

O protocolo STP converge muito lentamente pois faz depender a convergência de, na sequência da deteção de uma alteração, passar um tempo de espera longo sem se produzir nenhuma nova alteração, nem as inundações periódicas implicarem novas alterações. Durante a convergência o encaminhamento tem de ser bloqueado porque não existem nenhum outros mecanismos (*e.g.*, TTLs ou deteção de duplicados) que estancassem uma eventual tempestade de difusão (*broadcast storm*).

Todos os algoritmos e protocolos apresentados para suporte do funcionamento de uma rede Ethernet comutada (*switched*), *i.e.*, baseada em inundação com aprendizagem pelo caminho inverso e no protocolo STP, pressupõem que todos os equipamentos funcionam de acordo com o esperado pelos protocolos e não violam ou modificam os seus endereços MAC.

Caso contrário é possível bloquear a rede enviando falsos BPDUs, ou até desviar o tráfego, por exemplo enviando periodicamente *frames* em *broadcasting* com endereços MAC origem falsos de forma a atrair o tráfego que lhes era dirigido. Deixada entregue a si própria, sem outros mecanismos de segurança, uma tal rede torna-se muito frágil.

As redes Ethernet equipadas com *switches* com suporte de VLANs ficam dotadas de uma mecanismo muito flexível que permite seccionar a rede em sub-redes distintas e isoladas, cada uma das quais associada a uma VLAN distinta.

Este mecanismo é fundamental para organizar uma rede complexa em sub-redes distintas por razões de segurança, isolamento e desempenho. Os APs com suporte de 802.11 também podem ter suporte de VLANs e separarem os utilizadores e os SSIDs por VLANs distintas.

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

Principais conceitos

Inundação (*flooding*) Algoritmo muito simples de encaminhamento numa rede: ao receber um pacote, um nó analisa o seu endereço de destino, e se este for diferente do seu, envia uma cópia do mesmo por cada uma das suas interfaces, excepto por aquela pela qual o pacote foi recebido. Numa rede normal este algoritmo introduz duplicados e pacote inúteis. Numa rede estruturada em árvore, este algoritmo apenas introduz pacotes inúteis que podem ser confinados usando aprendizagem pelo caminho inverso.

Tempestade de difusão (*broadcast storm*) Fenómeno que tem lugar numa rede a funcionar com encaminhamento por inundação caso não exista um mecanismo de detecção de duplicados, ou a rede não esteja estruturada como uma árvore. Uma tal tempestade satura e inutiliza a rede.

Aprendizagem pelo caminho inverso (*backward learning*) Técnica que permite minorar o tráfego inútil numa rede estruturada em árvore: se foi recebido um pacote por uma interface, com um dado endereço origem, então o nó com esse endereço está necessariamente por detrás dessa interface.

Protocolo STP (*Spanning Tree Protocol*) Protocolo (normalizado inicialmente pela norma IEEE 802.1D) que permite a um conjunto de *switches* Ethernet calcular uma árvore de cobertura e apenas encaminhar *frames* de acordo com a mesma. Caso haja uma avaria na rede, o STP reconfigura a árvore de cobertura.

BPDU (*Bridge Protocol Data Unit*) *Frames* de controlo usados pelo STP para calcular uma árvore de cobertura.

Convergência de um protocolo de encaminhamento é o tempo necessário para, na sequência de uma qualquer alteração de estado dos nós ou canais, todos os nós de comutação tomarem em consideração essas alterações, reconfigurarem a rede, e voltarem a colocá-la em funcionamento de acordo com a nova configuração. O STP convencional converge muito lentamente, *i.e.*, em dezenas de segundos.

VLAN (*Virtual Local Area Network*) Mecanismo muito flexível que permite seccionar uma rede Ethernet comutada em várias sub-redes distintas e isoladas, cada uma das quais associada a uma VLAN distinta. Os *frames* Ethernet que atravessam domínios com noção de VLAN passam a incluir um campo especial no cabeçalho (normalizado inicialmente pela norma IEEE 802.1Q) e designado por VLAN *tag*.

Referências e notas históricas

Os algoritmos e tecnologias descritos neste capítulo foram introduzidos fundamentalmente pelos fabricantes de equipamentos informáticos nas então designadas redes locais (LAN - *Local Area Networks*), *i.e.*, redes confinadas a edifícios ou grupos de edifícios. O termo era usado por oposição a WAN (WAN - *Wide Area Network*), uma área tradicionalmente dominada ao nível do fornecimento de canais pelos operadores de telecomunicações. Tratou-se, no essencial, de um processo lento de resposta às necessidades das instituições que ia sendo tornado possível pela própria evolução da possibilidade de fabricar equipamentos cada vez mais sofisticados, mas de custo acessível. A procura da simplicidade foi uma constante.

O artigo original que introduziu o algoritmo mais tarde adoptado pelo protocolo STP foi publicado em 1985 por Radia Perlman [Perlman, 1985], na altura a trabalhar

num grande fabricante de computadores, a DEC - Digital Equipment Corporation. Todas as tecnologias descritas evoluíram nos laboratórios dos fabricantes e foram depois normalizadas no seio da IEEE na série de normas com o prefixo IEEE 802.

A quantidade de normas desenvolvidas para as redes Ethernet comutadas é imensa e continua a crescer. Essa evolução inclui igualmente algumas revisões e fusões de normas. Por exemplo, na descrição da norma IEEE 802.1Q-2014 no site www.ieee802.org pode ler-se: “This standard incorporates IEEE Std 802.1QTM-2011, IEEE Std 802.1QbeTM-2011, IEEE Std 802.1QbcTM-2011, IEEE Std 802.1QbbTM-2011, IEEE Std 802.1QazTM-2011, IEEE Std 802.1QbfTM-2011, IEEE Std 802.1QbgTM-2012, IEEE Std 802.1aqTM-2012, IEEE Std 802.1QTM-2011/Cor 2-2012, and IEEE Std 802.1QbpTM-2014. The standard includes much functionality previously specified in 802.1D.” Ou seja, a versão de 2014 da norma IEEE 802.1Q incorpora todas as funcionalidades do STP, Rapid STP, Multiple Spanning Trees, VLANs, Per VLAN Spanning Trees, etc.

Com o aparecimento de canais Ethernet *full-duplex* de longa distância baseados em fibras ópticas, e *switches* de alta capacidade, estas redes deixaram de se confinhar ao espaço das LANs (termo que começou por isso a cair em desuso), e passaram a ser usadas também pelos operadores de longa distância, por exemplo para fornecerem serviços de interconexão de redes Ethernet com base em tecnologias Ethernet. Quando estas encaminham *frames* Ethernet de clientes que já usam VLANs, necessitam de preservar as VLANs dos clientes. Isso levou à introdução de um campo de VLAN *tagging* que inclui duas VLANs, a do cliente e a do operador. Por outro lado, o campo tradicional com o número de VLAN apenas permite cerca de 4000 VLANs distintas, um número insuficiente para um operador. Por isso este campo também teve necessidade de crescer.

As VLANs também são usadas nas redes de centros de dados para isolar os diferentes clientes que utilizam o centro de dados. O número de diferentes VLANs disponíveis também se tornou demasiado curto.

A entrada da Ethernet na longa distância e nas redes de operadores, assim como nos centros de dados de grande dimensão, tornou impossível continuar a adoptar tecnologias de encaminhamento sem preocupações de optimização da rede e dos caminhos seguidos pelos pacotes. Nesta sequência, começam a ser adoptadas para as redes Ethernet os mesmos algoritmos e técnicas de encaminhamento desenvolvidos para as redes gerais, que começaremos a estudar no próximo capítulo.

Esta linha de evolução está a materializar-se em torno de duas propostas, uma designada TRILL (Transparent Interconnection of Lots of Links) que foi desenvolvida no seio do IETF, ver [Perlman and Eastlake, 2011] e o RFC 6362, e a outra designada Shortest Path Bridging [Luo and Suh, 2011], ou norma IEEE 802.1aq. Ambas visam substituir as árvores de cobertura por alternativas mais sofisticadas. Estas propostas estão igualmente a ser desafiadas pelo aparecimento das Software Defined Networks, que são uma forma alternativa de melhor gerir redes de comutadores de pacotes e que será também apresentada nos capítulos seguintes.

Como diria um conhecido autor (A. S. Tanenbaum): “a beleza das normas é que há sempre muitas por onde escolher”.

Apontadores para informação na Web

- <http://www.ieee802.org> – É o site oficial das normas IEEE 802.
- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.

15.6 Questões para revisão e estudo

1. Uma rede Ethernet comutada (*switched*) organizada em árvore usa o algoritmo de encaminhamento por inundaçāo com filtragem baseada em aprendizagem pelo caminho inverso.
 - (a) Nesta rede é necessário existir um mecanismo de detecção de duplicados?
 - (b) Admitindo que o endereço MAC de destino *Addr* é conhecido de todos os *switches*, o encaminhamento de um *frame* dirigido a *Addr* é realizado usando cópias inúteis?
2. Considere o algoritmo de encaminhamento de pacotes por inundaçāo (*flooding*). Qual ou quais das seguintes afirmações são verdadeiras no contexto desse algoritmo de encaminhamento?
 - (a) O algoritmo de inundaçāo implementa naturalmente a difusão de mensagens para todos os computadores ligados à rede.
 - (b) O algoritmo de encaminhamento por inundaçāo é muito ineficiente e por isso nunca é usado na prática
 - (c) Uma maneira de suprimir a introdução de todos os pacotes duplicados através de inundaçāo numa rede estruturada em malha (com ciclos) é usar um TTL adequado.
 - (d) Numa rede estruturada em árvore, a inundaçāo encaminha sempre pelo melhor caminho a primeira cópia a chegar ao destino.
 - (e) Numa rede totalmente conexa (cada nó tem pelo menos um caminho para qualquer um dos outros) o algoritmo de inundaçāo assegura que todas as mensagens chegam a todos os nós desde que não existam erros de transmissão.
 - (f) Caso a topologia da rede não tenha ciclos, a inundaçāo é um processo razoavelmente eficiente de implementar o encaminhamento porque nunca enviará mensagens em duplicado, nem mensagens inúteis.
3. Indique uma ou mais razões para associar um TTL a cada endereço MAC registado na tabelas de endereços dos *switches* que realizam aprendizagem pelo caminho inverso.
4. Num dado *switch* constatou-se que a porta pela qual este recebeu um *frame* com endereço origem A foi a porta P2, mas 2 minutos atrás tinha sido a porta P1. Que poderá explicar esta situação?
5. Num dado *switch* constatou-se que a porta pela qual este recebeu um *frame* com endereço origem A foi a porta P2, mas 100 μ segundos atrás tinha sido a porta P1. Que poderá explicar esta situação?
6. Numa rede Ethernet comutada (*switched*) não podem existir ciclos pois a presença dos mesmos introduziria uma situação designada por tempestade de difusões (*broadcast storm*). Explique em que consiste esta situação e dê uma sugestão de como a mesma poderia ser detectada pelos *switches*.
7. Numa rede Ethernet comutada os *switches* dispõem de tabelas de endereços MAC.
 - (a) Para programar essas tabelas qual a estrutura de dados que escolheria?
 - (b) Porque razão, ou razões, é possível a dimensão das tabelas ser inferior ao número de diferentes endereços vistos pelo *switch*?

- (c) Indique um mecanismo particularmente eficaz na ajuda da convergência das tabelas dos *switches* que um computador pode usar sempre que muda de porta, *i.e.*, sempre que a sua interface é ligada e depois se volta a ligar.
- (d) Indique uma forma de um computador conduzir um ataque de negação de serviço a uma rede deste tipo (mais eficaz no caso em que há mais interfaces do que entradas nas tabelas de comutação). O ataque tem por consequência que o tráfego dos computadores normais passa a ser encaminhado por inundação pois o mecanismo de aprendizagem deixa de ser eficaz.
- (e) Idealize uma ou mais formas de combater este tipo de ataques.
8. 1000 computadores estão ligados a uma rede comutada (*switched*) Ethernet com 3 comutadores (*switches*) Ethernet. Estes equipamentos são idênticos a menos do seguinte aspecto: os comutadores A e B têm uma tabela de endereços MAC com 32 entradas e o comutador C tem uma tabela de endereços MAC com 1024 entradas. Após fazer medidas, você verificou que os comutadores A e B realizavam mais inundações que o comutador C. Que justifica a diferença observada?
9. Considere a rede Ethernet comutada (*switched*) da Figura 15.14. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. O computador com o endereço MAC_1 enviou um *frame* em difusão, seguido de um *frame* dirigido ao computador com o endereço MAC_6 e este enviou um *frame* em difusão seguido de um *frame* dirigido ao computador com o endereço MAC_1 .

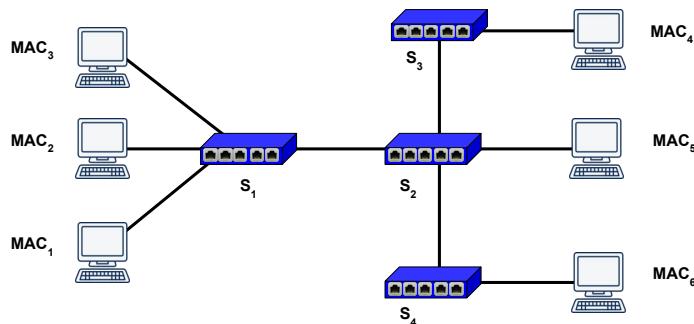


Figura 15.14: Rede de comutadores Ethernet do exercício 9.

- (a) No final desta troca de *frames* quais os endereços MAC que o comutador S_2 passou a conhecer?
- (b) No final desta troca de *frames* quais os endereços MAC que o comutador S_3 passou a conhecer?
10. Considere a rede Ethernet comutada (*switched*) da Figura 15.15. Cada computador é designado pelo seu endereço MAC. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. O seguinte conjunto de comunicações teve lugar: primeiro o computador MAC_1 enviou o *frame* f_1 ao computador MAC_8 ; depois o MAC_9 enviou o *frame* f_9 ao

computador MAC_3 ; depois o computador MAC_8 enviou o frame f_8 ao computador MAC_1 ; e finalmente o computador MAC_3 enviou o frame f_3 ao computador MAC_9 . Quais dos diferentes frames enviados chegaram à interface do computador MAC_4 ?

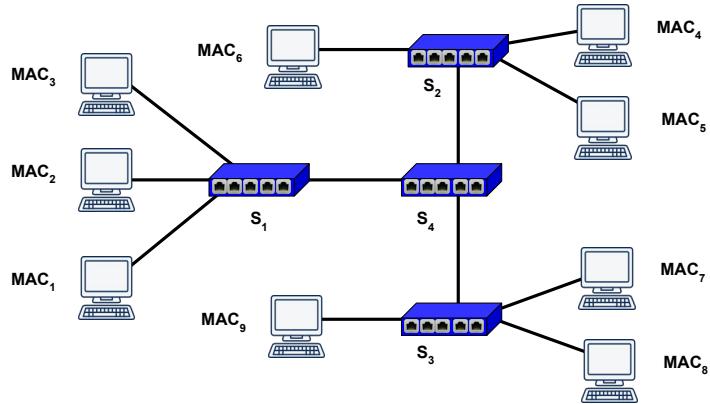


Figura 15.15: Rede de comutadores Ethernet do exercício 10.

11. Considere uma rede Ethernet comutada (*switched*) com a configuração da Figura 15.16, em que os canais têm o débito indicado em cada um. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. Comece por indicar os custos de cada canal, desenhe a árvore calculada pelo protocolo STP e indique o custo com que cada *switch* vê a raiz escolhida.

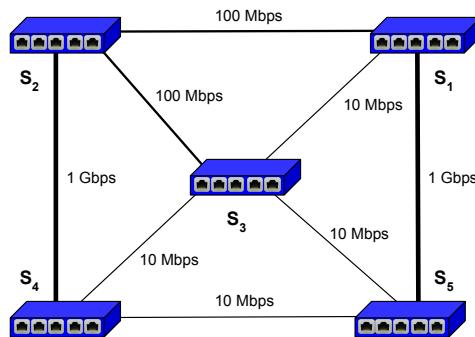


Figura 15.16: Rede de comutadores Ethernet do exercício 11.

12. Considere a rede Ethernet comutada (*switched*) da Figura 15.17. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*.
 - (a) Comece por indicar o custo de cada canal, desenhe a árvore calculada pelo protocolo STP e indique o custo com que cada *switch* vê a raiz escolhida.

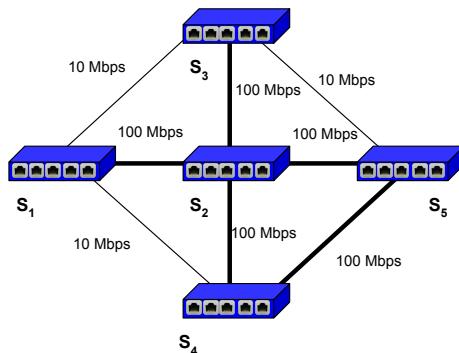


Figura 15.17: Rede de comutadores Ethernet do exercício 12.

- (b) Dada a árvore calculada pelo protocolo STP dê o exemplo de 2 caminhos ótimos (do ponto de vista do custo extremo a extremo) e o exemplo de um caminho não ótimo seguido pelos *frames* na rede através da árvore calculada. Indique cada caminho como uma sequência de identificadores de *switches*.
- (c) Por hipótese, o *switch* S_1 passou a não suportar o protocolo STP e ignora os seus *frames*, tratando-os como todos os outros, isto é, propagando-os normalmente por todas as interfaces visto que o seu endereço de destino é o endereço de *broadcast*. Calcule a nova árvore calculada pelo STP nesse caso e indique o custo com que cada *switch* vê a nova raiz escolhida.
- (d) Qual é a *Designated Port* do canal multi-ponto que passou a existir na rede?
13. Considere uma rede Ethernet comutada (*switched*) com a configuração da Figura 15.18, em que os canais têm todos o débito de 100 Mbps e os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. As portas dos *switches* estão numeradas de tal forma que as portas dos canais verticais têm sempre um número inferior ao número das portas a que estão ligados os canais horizontais. Nessa rede, existe um único servidor que está ligado ao *switch* S_5 . Associado a cada um dos outros *switches* existe um cliente.
- Comece por indicar o custo de cada canal, desenhe a árvore calculada pelo protocolo STP e indique o custo com que cada *switch* vê a raiz escolhida.
 - Admita que a rede é reconfigurada e o *switch* S_5 e o *switch* S_1 trocam de posição, mas mantendo-se o servidor no centro (agora ligado ao *switch* S_1). Calcule a nova árvore.
 - O servidor tem de transferir um ficheiro de 1 Gbyte para cada um dos clientes por TCP, abrindo uma conexão para cada um deles. Em qual das configurações a transferência acabaria mais depressa? (isto é, todos os clientes recebiam a sua cópia). Justifique a sua resposta apresentando argumentos rigorosos e baseados em factos, ou seja, mostrando que num caso, algumas conexões TCP terminariam necessariamente mais tarde a transferência do que no outro.
14. Considere uma rede com a configuração da Figura 15.19, em que os canais têm todos o mesmo débito. Os endereços MAC dos *switches* estão ordenados de

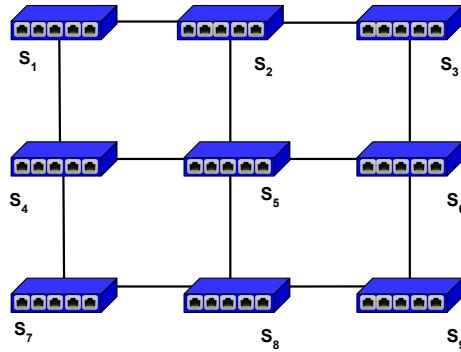


Figura 15.18: Rede de comutadores Ethernet do exercício 13.

forma compatível com a ordem dos números de cada *switch*. Por baixo de cada *switch* estão indicados os números (IDs) das portas. Diga qual dos *switches* era seleccionado pelo STP como raiz e quais os canais activos e quais os bloqueados.

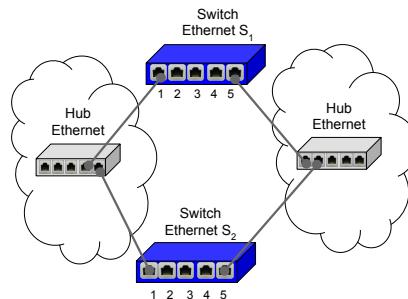


Figura 15.19: Rede de comutadores Ethernet do exercício 14.

15. Considere uma rede Ethernet comutada (*switched*) com a configuração da Figura 15.20, em que os canais têm o mesmo débito. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. Qual a árvore de cobertura escolhida pelo protocolo STP?
16. Considere uma rede Ethernet comutada (*switched*) com a configuração da Figura 15.21 em que os canais têm os débitos indicados. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*.
 - (a) Comece por indicar o custo de cada canal, desenhe a árvore calculada pelo protocolo STP e indique o custo com que cada *switch* vê a raiz escolhida.
 - (b) Na mesma rede são usadas várias VLANs distintas, cada uma das quais espalhada pela maioria dos *switches*. Existe alguma desvantagem em não usar uma única STP comum a todas as VLANs?
 - (c) Na mesma rede existe uma VLAN confinada aos *switches* S_1 e S_3 . Existe alguma desvantagem em usar uma única STP que cubra apenas esses dois *switches*?

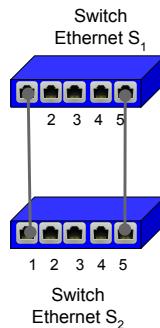


Figura 15.20: Rede de comutadores Ethernet do exercício 15.

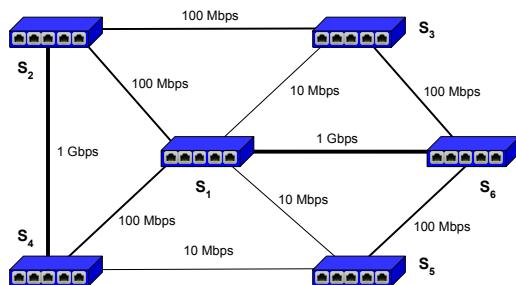


Figura 15.21: Rede de comutadores Ethernet do exercício 16.

17. Considere uma rede Ethernet comutada (*switched*) com a configuração da Figura 15.22, em que os canais a cheio têm o débito de 100 Mbps ou os a tracejado de 10 Mbps. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. Os *switches* estão parametrizados para admitirem várias VLANs e com uma versão do protocolo STP que suporta árvores distintas por cada VLAN. Nas portas de acesso dos *switches* para ligação de computadores estão presentes as seguintes VLANs: S_1 a S_3 — VLANs 1 e 100; S_4 a S_6 — VLANs 1 e 200 e S_7 a S_9 — VLANs 1, 100 e 200.

Os *switches* aplicam filtros às portas de *trunking* de forma a que os *frames* (incluindo os do protocolo STP) permitidos em entrada ou saída pela porta só podem pertencer às VLANs a que os *switches* dão acesso e os *switches* desconhecem completamente as VLANs para os quais não têm portas.

Comece por indicar o custo de cada canal, desenhe a árvore calculada pelo protocolo STP e indique o custo com que cada *switch* vê a raiz escolhida.

18. Considere uma rede Ethernet comutada (*switched*) com a configuração da Figura 15.23, em que os canais mais grossos têm o débito de 100 Mbps e os mais finos 10 Mbps. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. O tráfego principal na rede é o que circula entre os *switches* 7, 8 e 9, sendo o tráfego entre os restantes *switches* muito inferior. A rede tem um erro na sua concepção. Proponha uma solução para o mesmo tendo em consideração que só pode alterar a parametrização do STP.

19. Considere a rede da Figura 15.24 baseada num conjunto de troços Ethernet *full-*

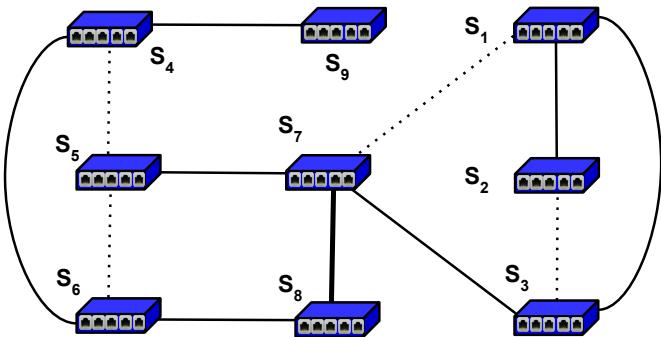


Figura 15.22: Rede de comutadores Ethernet do exercício 17.

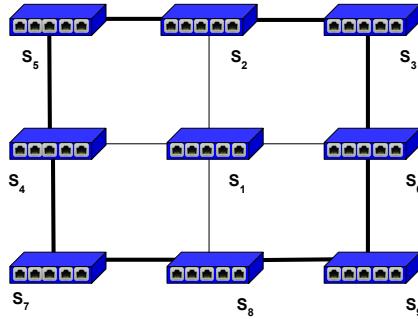


Figura 15.23: Rede de comutadores Ethernet do exercício 18.

duplex (a 100 Mbps os a traço forte e a 10 Mbps os a traço fino) interligando um conjunto de *switches* que executam o protocolo STP. Os endereços MAC dos *switches* estão ordenados de forma compatível com a ordem dos números de cada *switch*. A rede é complementada por um *hub* cujas portas interligam os *switches* indicados através das ligações ponteadas com o débito de 10 Mbps. Desenhe a árvore de cobertura determinada pelo protocolo STP.

20. As mensagens do protocolo STP têm os campos indicados na Figura 15.8.
- Indique quais desses campos são importantes para um *switch* calcular qual a sua *Root Port*.
 - O protocolo STP usa um temporizador chamado *Forward Delay*. Diga para que serve e qual a influência do seu valor sobre a qualidade do protocolo quando um canal ou um *switch* têm uma avaria.
 - Descreva como poderia um gestor da rede tentar que a convergência do STP fosse melhorada em caso de avaria de um canal, baixando as temporizações dos diferentes alarmes usados pelo protocolo, cujos valores por omissão são elevados. Por exemplo: *Forward Delay* = 15 segundos, *Hello Time* = 2 segundos e *Max age* = 6 segundos. Suponha que o maior caminho na rede tem $K \leq 20$ hops, que cada canal tem um tempo de propagação inferior a 1 ms e uma capacidade de pelo menos 10 Mbps. Repare que mesmo

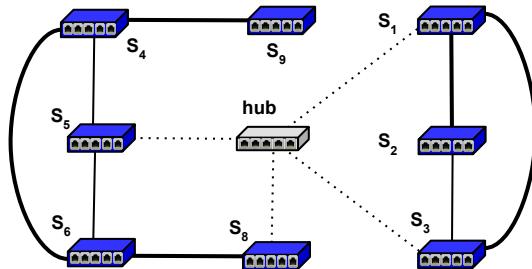


Figura 15.24: Rede de comutadores Ethernet do exercício 19.

havendo a possibilidade de se perderem BPDUs por erros nos canais, a probabilidade de tal suceder é muito baixa.

21. Quando um *switch* Ethernet recebe um *frame* instancia-o num objecto da classe **EtherFrame**, a qual tem os seguintes métodos:

```
void send(Port port)    // envia o frame pela porta port
void flood(Port port)   // inunda todas as portas com o frame excepto port
MACaddress getOrigin()  // devolve o endereço origem
MACaddress getDestination() // devolve o endereço destino
```

O *switch* tem também uma tabela (**macTable**) de endereços MAC:

```
Map< MACaddress, Port > macTable =
    new HashMap < MACaddress, Port > ();
```

com os métodos convencionais **put** e **get**.

O *switch* acabou de receber um *frame* **f** pela porta **p**. Escreva o pseudo-código necessário para o processar:

```
processFrame ( EtherFrame f, Port p ) { }
```

22. Numa rede Ethernet comutada (*switched*) especial são usados *frames* em cujo cabeçalho figuram também um TTL e um número de sequência colocado pelo emissor. A ideia é desta forma detectar duplicados e poder usar o algoritmo de inundação numa rede arbitrária, com eventuais ciclos. Quando um *switch* Ethernet recebe um *frame* instancia um objecto da classe **SpecialEtherFrame**, a qual tem os seguintes métodos:

```
void send(Port port)    // envia o frame pela porta port
void flood(Port port)   // inunda todas as portas com o frame excepto port
MACaddress getOrigin()  // devolve o endereço origem
MACaddress getDestination() // devolve o endereço destino
int getSequence() // devolve o número de sequência
void decreaseTTL() // decrementa o TTL
boolean expiredTTL() // true se TTL <= 0
```

```
int getTTL() // devolve o TTL do frame
void setTTL(int t) // fixa o TTL do frame
```

O *switch* tem uma tabela de endereços MAC, a tabela `macTable`:

```
Map< MACaddress, Port > macTable =
    new HashMap < MACaddress, Port > ();
```

com os métodos `put` e `get` como é convencional. O *switch* tem também uma tabela (`frameTable`) de endereços MAC origem e números de sequência com os métodos:

```
void register( SpecialEtherFrame f);
// regista o frame f na tabela
boolean isDuplicate( SpecialEtherFrame f);
// verifica se f já foi registado
```

O *switch* acabou de receber um *frame* *f* pela porta *p*. Escreva o pseudo-código necessário para o processar:

```
processFrame ( SpecialEtherFrame f, Port p ) { }
```

23. Numa hipotética rede Ethernet comutada (*switched*), organizada em malha, *i.e.*, com vários ciclos e caminhos redundantes, utiliza-se um protocolo inspirado do STP para calcular tantas árvores de cobertura quantos os *switches* existentes. Cada uma das árvores tem raiz num *switch* diferente.
 - (a) Sugira as modificações a introduzir no protocolo STP para que isso seja possível.
 - (b) Para encaminhar os *frames* é também usada uma variante do algoritmo normal. Essa variante consiste em um frame *frame f* ser sempre encaminhado pela árvore com raiz no *switch* em que *f* entrou na rede. Para que isto seja possível, teriam de ser realizadas modificações no cabeçalho dos *frames* Ethernet? Se sim sugira uma hipótese.
 - (c) Qual o custo do encaminhamento de um *frame* Ethernet nesta variante? Compare-o sucintamente com os custos de encaminhamento numa Ethernet comutada com uma única árvore de cobertura.

Capítulo 16

Encaminhamento pelo caminho mais curto

Make everything as simple as possible, but not simpler.

– Autor: *Albert Einstein (1879-1955)*

O encaminhamento com base em inundação é uma aplicação directa do princípio KISS (*Keep It Simple, Stupid*), i.e., trata-se de uma solução tão simples quanto possível. Essa solução é adequada quando os requisitos de uma solução de encaminhamento não são demasiado exigentes, nomeadamente: a rede tem uma elevada capacidade face às necessidades, o custo dos canais é suficientemente baixo para se poderem ter canais de reserva sem serem utilizados, o âmbito geográfico da rede não é elevado pelo que o impacto do tempo de trânsito de extremo a extremo também pode ser ignorado, e é aceitável realizar a inundação de toda a rede com pacotes desde que o rácio entre inundações e encaminhamento ponto-a-ponto seja baixo.

No entanto, caso a rede tenha um elevado âmbito e o número de computadores ligados à mesma seja muito elevado, esta solução deixa de ser realista pois os canais de longa distância são dispendiosos e deixa de ser viável sobre-dimensioná-los. Adicionalmente, caso a rede tenha de cobrir muitas cidades, deixa de ser aceitável realizar frequentemente inundações e também seria um desperdício inaceitável não usar os melhores caminhos para atingir o destino. Cobrir um grande número de cidades com uma rede estruturada em árvore só é realista com distâncias geográficas curtas. Finalmente, com várias dezenas ou centenas de milhar de interfaces distintas, também seria catastrófico para o desempenho provocar uma inundação cada vez que cada uma fosse activada ou endereçada pela primeira vez. Em resumo, a inundação, apesar de simples, só é uma solução viável em casos particulares.

No caso mais geral, é necessário usar soluções mais sofisticadas que permitam responder a vários constrangimentos e que tornem possível satisfazer o maior número possível de clientes, com o menor custo possível. Ou seja, no caso geral, o encaminhamento envolve facetas de correção (os pacotes têm de chegar ao destino) mas também de optimização: quer no que diz respeito a cada fluxo de pacotes em particular, quer no que diz respeito à operação global da rede, minimizando os seus custos, ao mesmo tempo que se maximiza a qualidade e quantidade do serviço prestado. Por outro lado, como a rede é formada por muitos nós e canais, em caso de avaria de alguns é necessário mantê-la operacional, reconfigurando-a rápida e dinamicamente se necessário for.

O problema do encaminhamento de pacotes é um dos problemas centrais das redes de computadores e também um dos mais complexos, pois envolve várias facetas e requisitos:

- teóricas** para modelização e optimização da rede;
- algorítmicas** para concretizar essas soluções;
- de engenharia** para implementar um sistema complexo que responda aos requisitos e minimizar o seu custo;
- de monitorização** para obter dados sobre o seu funcionamento real;
- operacionais e de planeamento** para responder às necessidades do dia a dia e adaptar a rede à continua evolução dos requisitos que lhe são colocados.

Assim, existem inúmeras soluções de gestão do encaminhamento que são função de requisitos específicos e das condições concretas da rede. Neste capítulo vamos estudar uma solução relativamente geral e bem conhecida que é designada por encaminhamento pelo “caminho mais curto” (*shortest path routing*). Neste quadro, usa-se o conceito de distância como forma de medir a adequação de um caminho.

O capítulo começa por enquadrar o problema do encaminhamento e explicar por que razão encaminhar pelo caminho mais curto é uma solução adequada em muitos contextos. Em seguida apresenta um algoritmo para determinar o caminho mais curto entre quaisquer dois nós de uma rede. Finalmente, apresenta várias soluções de engenharia para concretização desta forma de encaminhamento e faz referência às suas concretizações em termos de protocolos de encaminhamento normalizados.

16.1 Encaminhamento: o problema e uma solução

À primeira vista, o problema do encaminhamento de pacotes é simples de enunciar: quando se pretende encaminhar um pacote de x para y , ou quando um nó de comutação x pretende enviar um pacote para o destino y , que caminho escolher? No entanto, esta simplicidade esconde alguma complexidade e questões suplementares que também são importantes.

Se o caminho variar de pacote para pacote, pode ser necessário manter a estabilidade do encaminhamento do conjunto dos pacotes relacionados com a mesma conexão, variando potencialmente o caminho apenas de conexão para conexão. Se na rede se pretende dar diferentes qualidades de serviço a diferentes aplicações ou a diferentes clientes, pode ser necessário tomar em consideração não só a origem e o destino, mas também quais são as aplicações e os clientes envolvidos. Dependendo de diversos factores, o melhor caminho pode ser o com o maior débito de extremo a extremo, o com o menor tempo de trânsito de extremo a extremo, o com o menor custo monetário, ou ainda o que for mais fiável (*i.e.*, que perder menos pacotes).

Se se pretender optimizar a utilização dos canais disponíveis para aumentar a quantidade de tráfego por unidade de tempo que a rede encaminha, os caminhos podem ter de ser adaptados em função da carga da rede para evitar a saturação de alguns canais, ao mesmo tempo que outros estão sub-aproveitados. Isto é, caso a rede disponha de caminhos alternativos, será também desejável poder explorá-los para realizar **distribuição de carga**.

Assim, para além dos critérios relacionados com a finalidade propriamente dita da rede, que impõem que o encaminhamento seja **correcto**, *i.e.*, que os pacotes chegam

ao seu destino, é também necessário considerar critérios de **qualidade do serviço** mas também de **optimização da rede**.

Optimização

Com efeito, a solução deste problema vai condicionar a qualidade de serviço oferecida pela rede aos seus utilizadores. Por exemplo: como diminuir o tempo de trânsito? Como maximizar a utilização da rede de forma a servir o melhor que possível o maior número possível de utilizadores? Como manter a equidade entre os diferentes utilizadores? Como dar garantias de serviço aos utilizadores (em termos do tempo de trânsito, da taxa de perca de pacotes, do débito mínimo de extremo a extremo)?

Problemas operacionais

Adicionalmente, é também necessário tomar em consideração requisitos de natureza operacional. Assim, é necessário encontrar soluções que permitam **responder automaticamente e dinamicamente a avarias**. Ou seja, caso alguns caminhos deixem de ser possíveis porque alguns nós ou canais avariaram repentinamente, deve ser possível escolher caminhos alternativos para manter o encaminhamento dos pacotes que atravessavam anteriormente as zonas com avarias.

É possível os arquitectos da rede fazerem uma análise detalhada da mesma, escolherem os diferentes caminhos e parametrizarem os nós de comutação de pacotes para realizarem essa comutação de forma **estática**, no entanto, se a rede for de dimensão apreciável, é preferível dispor de um sistema que automaticamente se adapte à evolução da mesma, quer de curto prazo, porque ocorreram avarias, quer de médio prazo porque a rede cresceu para servir mais nós que se juntaram à mesma. Assim, a solução para o encaminhamento deve envolver a possibilidade de se adoptarem soluções de **encaminhamento dinâmico**.

Uma solução de compromisso: encaminhamento pelo caminho mais curto

Existe uma solução de compromisso cuja realização permite responder a um conjunto alargado dos requisitos anteriores. Essa solução consiste em encaminhar os pacotes pelo caminho mais curto entre a origem e o destino.

Com efeito, o **encaminhamento pelo caminho mais curto** (*shortest path routing*)¹ tenta optimizar o serviço prestado a cada pacote em particular, e daí a cada fluxo e a cada cliente. Enquanto a rede não tiver canais saturados, tudo indica que esta solução é aceitável.

No fundo, ela é equivalente à que é praticada numa rede viária fora das horas de ponta: o caminho escolhido pelos automobilistas é o caminho mais curto desde a origem até ao destino. Caso não existam engarrafamentos, não é necessário recorrer a caminhos alternativos mais longos, por os mais directos estarem engarrafados.

Se conseguirmos implementar esta solução de tal forma que os outros requisitos operacionais sejam satisfeitos, dispomos de uma solução razoável que apenas coloca de parte as questões de optimização global da rede e as de diferenciação da qualidade de serviço.

Mas em que consiste um caminho mais curto? Numa rede viária o caminho mais curto é o que encurta a distância geográfica. Numa rede de computadores, por omissão, o caminho mais curto é o que encurta o tempo de trânsito de extremo a extremo dos pacotes. Esse tempo é função do débito dos canais atravessados, do tempo de propagação dos mesmos e até da sua carga, visto que esta influencia a dimensão das filas de espera. Voltaremos a este ponto mais tarde, mas por agora podemos abstrair esta característica de um canal através de uma função, chamada custo, que devolve

¹ A noção de “mais curto” depende da forma como se calcula a distância, pelo que esta noção pode acomodar as várias variantes de melhor caminho acima enunciadas.

um valor estritamente maior que zero. O custo de um caminho será a soma do custo dos canais que este atravessa. Quanto “pior” o canal, maior deve ser o seu custo.

Por isso, rigorosamente, este encaminhamento deveria chamar-se encaminhamento pelo caminho de menor custo. No entanto, a tradição dita que se use a terminologia característica de se considerar o custo equivalente à distância pois foi dessa forma que o problema foi originalmente enunciado em teoria dos grafos.

O problema do encaminhamento numa rede de comutação de pacotes é um problema fundamental das redes de computadores que envolve facetas de modelização e optimização, algorítmicas, de engenharia para sua concretização, operacionais, e de planeamento e gestão.

Uma primeira solução de compromisso relativamente comum consiste em usar encaminhamento pelo caminho mais curto. **Esta solução é aceitável quando os caminhos mais curtos estão bem dimensionados para o tráfego existente.**

16.2 Determinação de caminhos mais curtos

Num grafo, um algoritmo de selecção de um caminho mais curto do nó a para o nó b seleciona um caminho cujo custo é mínimo entre todos os caminhos simples (sem ciclos) de a para b . Com efeito, pode existir mais do que um caminho mais curto. O algoritmo que vamos apresentar deve-se a E. W. Dijkstra. Dado um nó origem $orig$, este algoritmo calcula caminhos mais curtos para todos os outros nós do grafo. Esse conjunto de caminhos forma uma árvore de cobertura de caminhos mais curtos com raiz em $orig$.

Neste contexto, a rede é definida por um grafo $G = (N, E)$, onde N é o conjunto dos vértices ou nós e E é o conjunto dos arcos (*edges*). Seja n o número de nós em N , i.e., $n = \#N$. O grafo G é *simples* (um grafo sem lacetes nem arcos paralelos), *orientado* (onde (c, d) e (d, c) denotam arcos distintos) e *pesado*, sendo o peso ou custo de cada arco estritamente positivo, i.e., $\forall e \in E, cost(e) > 0$. Todos os caminhos em que estamos interessados são *simples*, i.e., todos os nós de um caminho são diferentes, o que é equivalente a o caminho não ter ciclos.

Neste modelo, os nós do grafo modelizam os computadores e os nós de comutação de pacotes. Os arcos modelizam os canais. O custo de um arco representa o custo do canal. Para não perder generalidade, considera-se que cada computador e cada nó de comutação podem ser origem ou destino de pacotes.

Seja $orig \in N$ o nó origem, a raiz de uma árvore de cobertura de caminhos mais curtos que se pretende calcular. No contexto do algoritmo, o custo de um nó x para outro nó y é $cost(x, y)$ se existe o arco $(x, y) \in E$ e ∞ no caso contrário. Dado um nó x , a distância de $orig$ a x ($distance(x)$) é definida como sendo a soma dos custos dos arcos de um caminho mais curto entre $orig$ e x . Esta distância corresponde ao custo de encaminhar tráfego de $orig$ até x por esse caminho. Por definição, a distância do nó $orig$ a si próprio é nula, i.e., $distance(orig) = 0$.

O algoritmo usa o conjunto de nós S (S de *Stable*) para os quais já se conhece um caminho mais curto desde $orig$. Inicialmente $S = \{orig\}$. Em cada iteração do algoritmo, um novo nó x é seleccionado, de tal forma que o caminho mais curto entre $orig$ e x foi determinado. x é acrescentado a S e passará a ser a base para determinar o novo nó a incluir em S na próxima iteração. Por esta razão, chamaremos a esse nó o nó *pivot* em cada iteração. A chave do algoritmo consiste em seleccionar o *pivot* de tal forma que é garantido que não existe nenhum outro caminho mais curto para

Listing 16.1: Pseudo código simplificado do algoritmo de Dijkstra para cálculo de caminhos mais curtos

```

1 // Input:
2 //      N - set of network nodes, n = #N
3 //      E - set of network edges (links) with costs C
4 //      orig - origin node
5 // Local variables:
6 //      S - set of nodes for which a shortest path is already known
7 //      pivot - a pivotal node for which a shortest path is known
8 // Output:
9 //      distance int[1 .. n] - distance to orig
10 //      predecessor node[1 .. n] - predecessor in the
11 //                                selected shortest path to orig
12
13 S = {orig}
14 for each (node x in N) {
15     // if (x,y) not in E, cost(x,y) = infinity
16     distance[x]=cost(orig,x)
17     if ( (orig,x) in E ) predecessor[x] = orig
18     else predecessor[x] = null
19 }
20 distance[orig] = 0
21 predecessor[orig] = orig
22 pivot = orig
23 while ( S != N ) {
24     find node x in N : x not in S and distance(x) is minimal
25     pivot = x
26     S += pivot // since there is no shorter path to pivot
27     // then update the distance for all nodes y not in S
28     // that are directly connected to pivot
29     for each (node y in N : y not in S and (pivot,y) in E ) {
30         d = min ( distance(y), distance(pivot)+cost(pivot,y))
31         if ( d < distance(y) ) {
32             // we found a better alternative
33             distance(y) = d
34             predecessor(y) = pivot
35         }
36     }
37 }
```

esse nó. O algoritmo termina quando $S = N$. O seu pseudo código é apresentado na Listagem 16.1. Ele usa o vector `distance[]` para representar o resultado do cálculo de $distancia(i)$, que vai actualizando conforme vai progredindo. Inicialmente, este vector é inicializado da seguinte forma:

$\forall x \in N \text{ distance}[x] = \text{cost(orig}, x)$

e `distance(orig)=0`, ver as linhas 13 a 22. O vector `predecessor[]` é usado para representar o nó que é o predecessor de cada nó x no caminho mais curto que foi escolhido de *orig* até x . No fim, este vector permitirá ao nó *orig* saber que caminho mais curto tomar para chegar a x . Inicialmente, $\forall x \in N$ `predecessor[x]=null` com excepção dos casos particulares em que existe um arco do nó para *orig* (predecessor provisório) e do caso `predecessor[orig]=orig`, ver também as linhas 13 a 22.

Para clarificar a forma como o algoritmo opera e como os seus resultados podem ser usados para implementar um protocolo de encaminhamento, vamos aplicá-lo à rede apresentada na Figura 16.1 (a), que é modelizada pelo grafo apresentado na Figura 16.1 (b). Nas figuras onde se apresenta a configuração de uma rede, os símbolos genéricos que são mais frequentemente usados para representar comutadores de pacotes são os que figuram na Figura 16.1 (a), *i.e.*, um cilindro com um conjunto de setas que simbolizam uma encruzilhada à qual chegam e da qual partem pacotes de dados. Os custos dos canais usados no modelo são inversamente proporcionais ao respectivo

débito. Para simplificar consideram-se todos os canais bidireccionais com custos iguais e equivalentes a dois arcos, um em cada sentido. Como os custos são iguais, apenas são indicados uma vez em cada arco.

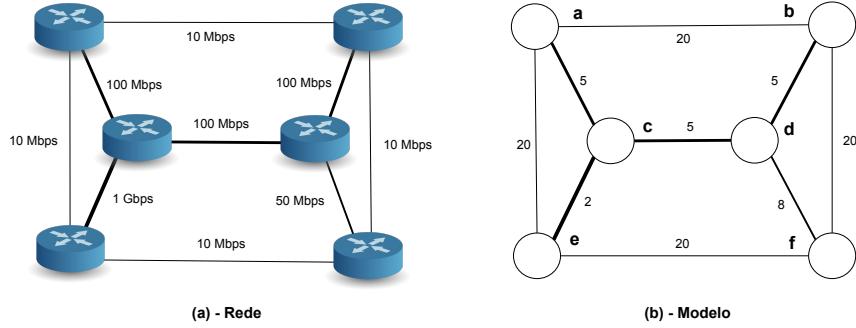


Figura 16.1: Rede (a) e sua modelização via um grafo (b)

A Figura 16.2 (a) mostra o estado do algoritmo ao grafo após a sua inicialização. O nó origem é o nó **a** que é o único elemento do conjunto **S**. Nessa figura e seguintes, os membros do conjunto **S** estão assinalados através de um círculo forte e o seu interior é sombreado. O nó **pivot** inicial é o próprio nó **a**. Em cada iteração o nó **pivot** está assinalado por uma seta nas figuras.

Adicionalmente, durante a inicialização, as entradas correspondentes aos nós **b**, **c**, **e** nos vectores **distance** e **predecessor** recebem o custo do arco que os liga directamente ao nó **a** e este torna-se o nó predecessor destes nós. Esta informação está apresentada no interior dos círculos correspondentes a esses nós. Em cada iteração, a distância e o nó predecessor são actualizados nos nós directamente ligados ao nó **pivot** (nos restantes não se podem alterar), e esse facto é assinalado sombreando o interior dos nós não pertencentes a **S** em que isso tem lugar. Por analogia, os nós directamente ligados a **a** estão sombreados no fim da inicialização.

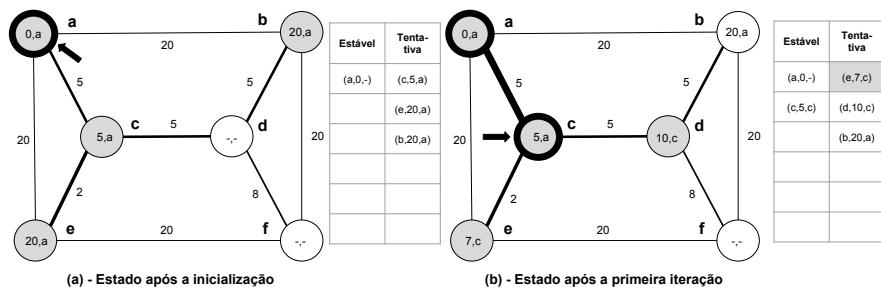


Figura 16.2: Estado do grafo após a inicialização (a) e o mesmo estado após a primeira iteração (b). À direita de cada grafo figura a visão do encaminhamento pelo nó **a**.

Para se ver como o algoritmo pode ser usado por um nó de comutação de pacotes, é necessário ter presente que estes necessitam de uma tabela que lhes permita, dado o destino de um pacote, saber por que interface transmitir o mesmo. Neste exemplo, ilustra-se essa tabela no nó **a**, pois estamos a usar o algoritmo para calcular os caminhos

mais curtos de a para os outros nós, ou seja, para todos os destinos possíveis. Essas tabelas chamam-se tabelas de comutação ou de encaminhamento e serão discutidas na secção seguinte.

Ao lado direito de cada grafo, é apresentada uma tabela que é uma primeira aproximação da tabela de encaminhamento do nó a , com a indicação da interface que o nó a tem de usar para atingir outro. Esta informação está codificada num triplo ordenado assim constituído (**destino**, **custo para lá chegar**, **vizinho para que enviar o pacote**). Na coluna *Estável* consideram-se apenas caminhos mais curtos. Na coluna *Tentativa* consideram-se formas conhecidas de chegar a outros nós, não necessariamente por um caminho mais curto.

Na Figura 16.2 (b) é apresentado o estado após a primeira iteração. Primeiro é seleccionado o novo **pivot** que será o nó não pertencente a S com a distância mínima. O nó seleccionado é o nó c . Esse nó é inserido em S e passou a ser conhecido um caminho mais curto para o mesmo, passando a figurar o mesmo na coluna *Estável*².

Em seguida, é actualizada a distância a todos os nós directamente ligados ao novo **pivot** e que não pertencem a S . Os nós em questão são os nós $\{e, d\}$. Para o nó d passou a ser conhecido um caminho e por isso este entra na coluna *Tentativa*. No caso do nó e descobriu-se um caminho mais curto, via o nó c , e por isso a sua entrada na coluna *Tentativa* é alterada e está marcada a sombreado.

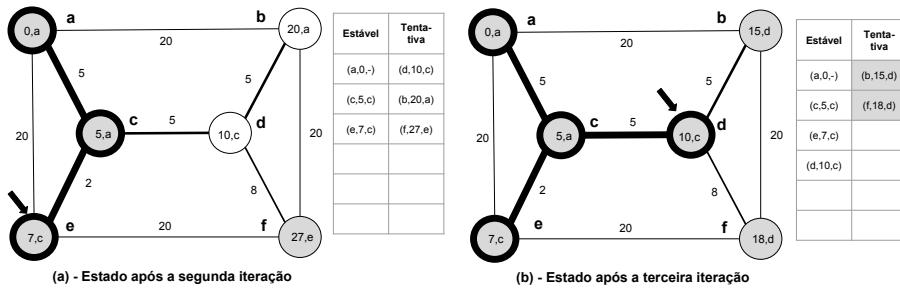


Figura 16.3: Estado do grafo após a segunda iteração (a) e após a terceira iteração (b). À direita de cada grafo figura a visão do encaminhamento pelo nó a .

Na Figura 16.3 (a) é apresentado o desenrolar da segunda iteração. Primeiro é seleccionado o novo **pivot**: o nó e passa a pertencer a S e entra na coluna *Estável* pois é o nó com menor distância. Depois é actualizada a distância e o nó sucessor dos nós que não pertencem a S mas que estão directamente ligados a e . O único nessas condições é o nó f que entra na coluna *Tentativa*.

A terceira iteração consta da Figura 16.3 (b). O nó d é o novo **pivot**, é inserido em S e na coluna *Estável*. Depois a distância e predecessor aos nós $\{b, f\}$ são actualizadas e como em ambos os casos se descobrem novos caminhos mais curtos, as informações que lhes estão associadas na coluna *Tentativa* são alteradas em ambos os casos.

Na Figura 16.4 (a) é apresentado o estado no fim da quarta iteração. O nó b é o novo **pivot**, é inserido em S e na coluna *Estável*. Em seguida tenta-se actualizar a informação sobre os nós não pertencentes a S directamente ligados a b . Trata-se apenas do nó f , mas como não existe via b um melhor caminho para f tudo continua

² Visto que se chega a cada nó em S por um caminho mais curto, o caminho conhecido para se chegar ao nó não pertencente a S com função distância mínima, i.e., o novo **pivot**, é necessariamente um caminho mais curto porque qualquer outro implicaria que os caminhos conhecidos para os membros de S não seriam mais curtos. A demonstração realiza-se por absurdo.

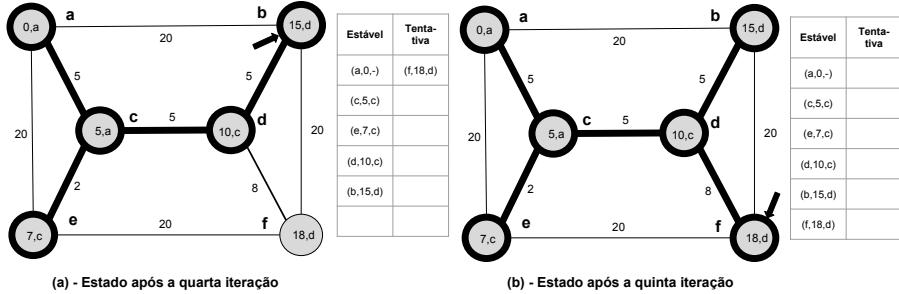


Figura 16.4: Estado do grafo após a quarta iteração (a) e após a quinta e última iteração (b). À direita de cada grafo figura a visão do encaminhamento pelo nó *a*.

na mesma. Finalmente, a Figura 16.4 (b) apresenta o estado final, após a quinta iteração. O novo nó **pivot** é inserido em *S* e na coluna *Estável* e mais nada se altera.

No fim, a coluna *Tentativa* está vazia, pois conhecem-se caminhos mais curtos para todos os nós, e a coluna *Estável* contém uma primeira versão da tabela de encaminhamento do nó *a*, em que figura, para cada destino existente na rede, a indicação do custo para lá chegar através de um caminho mais curto e o vizinho a usar para esse efeito.

Ao longo da evolução do algoritmos foi sendo igualmente marcada a árvore de cobertura de caminhos mais curtos que o algoritmo determinou. É interessante assinalar neste momento que este algoritmo acaba por calcular uma árvore de cobertura de caminhos mais curtos de forma centralizada, enquanto que o STP, se elegesse como raiz o nó *a*, calcularia de forma distribuída a mesma árvore (ou uma equivalente devido a poder existir mais do que um caminho mais curto).

Convém, no entanto, não perder de vista que o algoritmo distribuído usado pelo STP apenas calcula uma árvore de cobertura única para a rede inteira, que serve para encaminhar por inundação, e que com aprendizagem pelo caminho inverso, esse encaminhamento é óptimo apenas no caso em que o emissor é a raiz da árvore³. Enquanto que o algoritmo de Dijkstra pode ser aplicado a cada um dos nós e determinará nesse caso tantas árvores de encaminhamento óptimas quanto o número de nós existentes na rede, uma para cada origem.

Uma outra observação relevante tem a ver com o facto de que o STP usa um mecanismo para seleccionar, entre as diferentes opções de caminhos mais curtos, um caminho específico. Para a sua seleção usa os identificadores dos *switches* e das interfaces. O algoritmo de Dijkstra tem o mesmo problema quando o número de potenciais **pivots** é maior que um, ou quando ao actualizar a distância e o predecessor de um nó, se constata que existem diversas alternativas. Nesses casos o algoritmo tem também de tomar uma decisão arbitrária. As implementações reais podem optar por registrar as diversas alternativas de caminhos mais curtos na tabela *Estável*.

Para terminar, assinalamos que este algoritmo requer um visão global do grafo, incluindo os nós, arcos e seus custos, e que, tal como foi apresentado, a sua complexidade temporal é $O(n \times n)$. É no entanto possível implementar uma versão cuja complexidade temporal é $O(n \log n)$.

³ Rigorosamente, neste caso o encaminhamento é óptimo apenas do ponto de vista em que só usa caminhos mais curtos.

O algoritmo de Dijkstra é um algoritmo centralizado que, dado um grafo com arcos pesados e um nó origem, calcula uma árvore de cobertura de caminhos mais curtos com raiz no nó origem.

Este algoritmo, aplicado repetidamente a cada um dos nós de comutação de pacotes de uma rede, permite determinar caminhos mais curtos de cada nó para todos os outros, ou seja permite determinar tantas árvores de cobertura quantos os nós do grafo. Seja qual for o nó que toma uma decisão de encaminhamento de um pacote, encaminhá-lo-á por um caminho mais curto.

16.3 Encaminhamento pelo estado dos canais

Nesta secção e na seguinte vamos ver duas concretizações possíveis do **encaminhamento pelo menor caminho**, muitas vezes também designado *shortest path routing*. Antes vamos ver alguns aspectos comuns às duas formas de encaminhamento.

Routers ou comutadores inteligentes de pacotes

Os comutadores de pacotes com inteligência para executarem um protocolo genérico de encaminhamento de pacotes são muitas vezes designados em língua inglesa por *routers*, pois conhecem a noção de rota ou caminho (*route* ou *path*). O termo *router* pode ser traduzido por roteador ou encaminhador. No entanto, no resto do capítulo, continuaremos a usar o termo comutador de pacotes com inteligência, ou simplesmente comutador de pacotes, por ser essa a tradição no mundo dos profissionais de redes. A outra alternativa consiste em usar a terminologia inglesa que é de facto a que é usada mais frequentemente. Voltaremos a discutir a problemática da terminologia a usar no capítulo seguinte pois o problema merece maior aprofundamento e não é simplesmente uma questão da língua utilizada, ver o Capítulo 17.

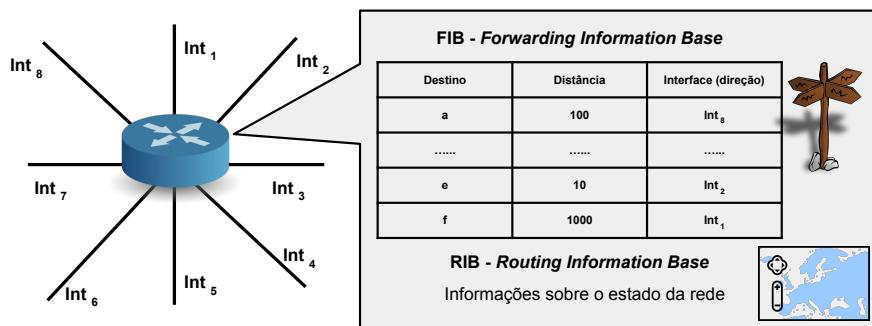


Figura 16.5: Comutador de pacotes e as duas bases de informação: FIB - *Forwarding Information Base* e RIB - *Routing Information Base*

Um comutador de pacotes contém em geral duas bases de informação, ver a Figura 16.5. A primeira chama-se **tabela de comutação ou de encaminhamento** e a terminologia em língua inglesa para a designar é **Forwarding Information Base (FIB)**, que será o termo que também usaremos. A FIB contém a informação que permite tomar rapidamente a decisão de comutação e é concretizada através de uma tabela de triplos (*destino, distância, interface*). Assim, quando recebe um novo

pacote, o comutador começa por extraír o endereço de destino do mesmo, depois consulta a FIB para saber qual a interface pela qual o pacote deve continuar a sua viagem até ao destino e, caso exista uma, o pacote é transmitido pela mesma. Senão, o pacote é ignorado pois o comutador não sabe que lhe fazer.

Caso um mesmo destino seja alcançável por várias interfaces, a que der acesso ao caminho mais curto será a escolhida. Naturalmente, se várias interfaces diferentes derem acesso a caminhos diferentes mas com o mesmo custo, é possível escolher uma delas de forma a tentar distribuir a carga pelos diferentes caminhos.

O comutador é, deste ponto de vista, semelhante a um cruzamento onde é necessário decidir qual a direção a seguir até ao destino. Para tal é usado o equivalente a um conjunto de “tabuletas a apontar para as diferentes direções”. No entanto, existe uma diferença, se um destino não está na tabela, o comutador não sabe como alcançá-lo e não consegue encaminhar o pacote. Por isso, é um cruzamento cheio de tabuletas, com milhares delas em redes de grande dimensão.

Mas como é que a FIB é preenchida? Uma das formas mais simples consiste em o administrador do comutador definir o conteúdo da FIB. Este tipo de encaminhamento chama-se encaminhamento estático e, como já foi referido, não é uma solução muito razoável, salvo em casos particulares. No caso geral, é o software que executa no comutador, que através de um protocolo de encaminhamento, preenche e actualiza a FIB de forma automática. Esse protocolo é executado em colaboração com outros comutadores, ou com servidores de controlo da rede. A informação que vai sendo adquirida pelo comutador através desses protocolos designa-se genericamente por ***Routing Information Base (RIB)***. A RIB permite ao comutador actualizar a FIB.

Por oposição à FIB, que pode ser assimilada a um “conjunto de tabuletas com distâncias e direções”, a RIB é como se fosse um “mapa das estradas” que permite conhecer os diferentes caminhos entre quaisquer duas cidades. Através de algoritmos é possível saber as rotas existentes, seleccionar a ou as melhores e preencher a FIB.

Métricas de encaminhamento

Para comparar caminhos e canais para efeitos de encaminhamento é necessário conhecer o seu custo. O custo de um caminho é a soma dos custos dos canais que o formam. Cada protocolo de encaminhamento usa uma função, também chamada métrica do custo ou simplesmente métrica, que associa um custo a cada canal.

A métrica mais simples consiste em considerar que todos os canais têm o mesmo custo, por exemplo o custo 1. Nesse caso, o custo de um caminho é equivalente ao seu número de canais, o que se designa por comprimento do caminho. Este tipo de métrica é usada por alguns protocolos de encaminhamento mais simples, mas a mesma só é realista se todos os canais tiverem débitos, cargas e tempos de propagação semelhantes.

Em alternativa, poder-se-ia medir periodicamente o tempo necessário para atravessar um canal até ao próximo comutador. Para esse efeito poder-se-ia usar a mesma técnica que a empregada pelo programa ping, ver a Secção 3.3. No entanto, esta forma de determinar o custo teria alguns problemas.

Por um lado, a noção de custo está associada à direção do canal e, se as duas direções tiverem débitos diferentes, o custo associado a cada uma deve ser diferente. Mesmo que as duas direções de comunicação fossem equivalentes em débito e distância, o custo pode variar com a direção devido à intensidade do tráfego e devido ao estado das filas de espera. Ora o programa ping só mede o tempo de ida e volta e a sua divisão por dois pode não ser a mais adequada. É possível medir mais rigorosamente o tempo de trânsito em cada direção, mas isso exige relógios sincronizados nas duas extremidades, o que é uma exigência muito forte, apesar de mais realista hoje em dia usando GPS.

Por outro lado, a prática mostrou que o custo de um canal deve ser relativamente estável e não variar num curto espaço de tempo, como será discutido no final desta

secção. Por isso, no contexto do encaminhamento pelo estado dos canais, o custo deve ser apenas proporcional ao débito, combinado com outros factores como o tempo de propagação por exemplo. O custo dos canais pode ser fixado pelos administradores da rede, mas uma solução mais simples e mais comum é usar um custo fixado automaticamente e baseado apenas no débito. Por exemplo, usando a seguinte fórmula:

$$\text{Custo do canal} = \text{débito de referência}/\text{débito do canal}$$

O débito de referência é o débito correspondente ao custo 1. Se este débito for 1 Gbps (10^9), então um canal com 100 Mbps terá o custo $10 (10^9 \times 10^{-8})$ e um de 1 Mbps o custo $1000 (10^9 \times 10^{-6})$. Todos os comutadores de uma rede que usem uma métrica baseada em débito devem usar o mesmo débito de referência e este deve tomar o valor do débito dos canais de maior débito da rede.

Esta métrica é designada **métrica por omissão** nas implementações concretas.

Os routers ou comutadores de pacotes inteligentes dispõem de uma **tabela de comutação ou de encaminhamento** (*Forwarding Information Base (FIB)*), que lhes permitem comutar rapidamente os pacotes, e uma **Routing Information Base (RIB)**. A RIB é preenchida pelos protocolos de encaminhamento e permite ao comutador actualizar a FIB.

Para computar a FIB, os protocolos de encaminhamento necessitam de associar um custo ou métrica a cada canal da rede. Existem diversas métricas de encaminhamento. Uma das mais simples e comum, designa-se **métrica por omissão**, e consiste em calcular um custo inversamente proporcional ao débito do canal.

Encaminhamento pelo estado dos canais

O encaminhamento conhecido como **encaminhamento pelo estado dos canais** (*link state routing*) baseia-se em encontrar uma forma de replicar em todos os comutadores um grafo com o estado global da rede. Como este estado global replicado é idêntico, cada comutador usa o algoritmo de Dijkstra para determinar uma árvore de cobertura de caminhos mais curtos para cada destino existente na rede. Depois, através destes caminhos, os quais constituem a totalidade das rotas baseadas em caminhos mais curtos existentes na rede, preenche a FIB com a informação que vai ser usada para encaminhar os pacotes.

Se todos os comutadores usarem da mesma descrição da rede, as diferentes FIBs serão compatíveis e encaminharão os pacotes pelas mesmas árvores de caminhos mais curtos desde a sua origem.

Caso o estado dos canais se altere, essa informação é de novo actualizada em todos os comutadores da rede e as FIBs são também de novo actualizadas em função do novo estado da rede.

Assim, cada comutador executa as seguintes operações:

1. Inicialmente, ou na sequência de uma alteração, cada comutador difunde aos outros o custo e o estado (*up / down*) dos canais que lhe estão ligados, assim como a identificação do nó na extremidade. Normalmente, estas informações chamam-se **Informações de Adjacência** (*Adjacency Announcements*).
2. Depois de uma alteração local, ou remota mas recebida dos outros comutadores, cada comutador executa de novo o algoritmo de Dijkstra e determina quais são as entradas da sua FIB que se alteraram.

3. Por fim, o comutador repercute as alterações na sua FIB.

É por cada comutador adquirir por difusão o estado replicado da rede, que esta técnica de encaminhamento se pode também caracterizar por encaminhamento centralizado, pois baseia-se na execução de um algoritmo centralizado por cada comutador para preencher a FIB. A parte distribuída do protocolo consiste na difusão do estado entre comutadores.

Com este tipo de encaminhamento, a RIB dos comutadores designa-se ***Link State Database (LSDB)*** ou ***Adjacencies Database*** e contém o estado de todos os canais da rede, cada um dos quais caracterizado pelos identificadores (dos comutadores) de origem e destino, pelo seu custo e pelo seu estado (*up / down*). Na verdade, se se considerar que um canal no estado *down* (inactivo) tem o custo infinito, então basta o custo para caracterizar o estado de cada canal. Repare-se que cada canal *full-duplex* é equiparado a dois canais *simplex* e corresponde a duas adjacências.

Como é que a LSDB é replicada por todos os comutadores? Cada comutador envia periodicamente (com um período de vários minutos), ou sempre que se produz uma alteração (incluindo na sua inicialização), **anúncios de estado dos canais (LSAs - Link State Announcements)** ou **anúncios de adjacência (adjacency announcements)** dirigidos a todos os outros comutadores. Cada comutador só origina a informação que lhe é local, *i.e.*, o conjunto das suas adjacências. A LSDB global é formada pela união de todas as informações locais.

Um LSA tem um identificador origem, *i.e.*, o identificador do comutador que o gera, um número de sequência e uma lista das suas adjacências. Os LSAs são depois propagados a toda a rede através do algoritmo de difusão fiável descrito na Secção 15.1. Recorde-se que esse algoritmo permite replicar em todos os membros da rede (os comutadores) mensagens (os LSAs) ordenadas por um número de sequência. Desta forma, sempre que se produz uma alteração, e periodicamente com um período alargado (de vários minutos), todos os comutadores enviam aos outros a informação necessária para estes preencherem as suas RIBs e, salvo durante a propagação das alterações de estado, todas as LSDBs dos diferentes comutadores terão os mesmos valores.

Associado a cada LSA existe também um parâmetro, designado “Idade” que denota o tempo desde que o LSA foi emitido pela última vez e é usado para evitar ter na LSDB informações que não são actualizadas há muito tempo. Assim, se um comutador for retirado da rede, ou se um canal for suprimido, ao fim de alguns minutos os seus LSAs deixarão de ser refreshados, a sua idade cresce até à idade máxima e será suprimido da LSDB.

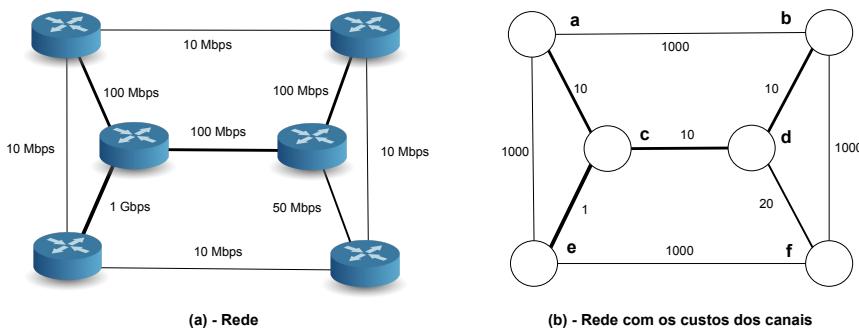


Figura 16.6: Rede e informações necessárias para preencher a LSDB da mesma

A Figura 16.6 mostra uma rede com as informações necessárias para preencher a LSDB. Esta base de dados está parcialmente ilustrada na Tabela 16.1

Tabela 16.1: Visão parcial da LSDB da rede da Figura 16.6. As entradas são a informação de adjacência e mostram o comutador que a originou, o número de sequência, a idade em minutos e a informação de adjacência.

Nó origem	# sequência	Idade	Adjacência
a	9	45	a - b, custo 1000
a	9	45	a - c, custo 10
a	9	45	a - e, custo 1000
b	12	16	b - a, custo 1000
b	12	16	b - d, custo 10
b	12	16	b - f, custo 1000
.....
f	3	31	f - b, custo 1000
f	3	31	f - d, custo 20
f	3	31	f - e, custo 1000

Depois desta caracterização a alto nível deste tipo de protocolos de encaminhamento, vamos a seguir analisar mais em detalhe algumas das facetas mais importantes dos mesmos.

Determinação do estado de um canal ou de uma adjacência

Para determinarem o estado dos seus canais, os comutadores enviam periodicamente para a outra extremidade de cada canal uma mensagem `hello`. O comutador da extremidade deve responder com outra mensagem `hello`. Desta forma os comutadores ficam a saber a identificação da outra extremidade e o estado de cada um dos seus canais. Se este for o único meio que permite ao comutador saber se o canal está operacional ou não, a velocidade com que uma avaria é detectada depende do período com que estas mensagens são enviadas. Uma detecção muito rápida requer uma frequência mais alta, por exemplo de 100 em 100 milissegundos, visto que só depois de várias tentativas falhadas o canal é dado como inoperacional.

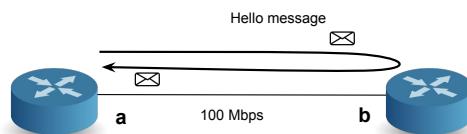


Figura 16.7: Execução do protocolo *Hello* entre dois comutadores interligados directamente por um canal ponto-a-ponto

Modernamente, a maioria dos canais ponto-a-ponto dispõem de mecanismos que permitem detectar a operacionalidade da outra extremidade através de sistemas de sincronização de baixo nível (detecção da onda portadora ou de sinais de sincronização). Nesse caso a interface lança alarmes rápidos caso o canal deixe de funcionar (*e.g.*, em cerca de 10 ms). De qualquer forma, mesmo que este tipo de mecanismos existam, por segurança continua a usar-se o protocolo `hello`, pois este também permite conhecer os identificadores de ambas as extremidades e reconhecer que, para além do canal estar

operacional, a outra extremidade também participa no protocolo. Para este efeito, basta enviar mensagens `hello` espaçadas de um ou dois minutos.

Finalmente, os comutadores conseguem saber através das suas interfaces qual o débito dos canais e assim calculam o custo dos mesmos.

Como o algoritmo de Dijkstra só “sabe” lidar com canais ponto-a-ponto, os canais multi-ponto de interligação de vários comutadores levantam um problema suplementar. Uma forma de resolver esse problema consistiria em transformar um canal multi-ponto de interligação de n comutadores em $n \times (n - 1)/2$ canais ponto-a-ponto ou o dobro de adjacências. Para evitar esta explosão de adjacências, a solução consiste em um dos comutadores ser eleito, por um processo semelhante ao usado pelo protocolo STP, e tornar-se o *designated router* do canal. Este comutador introduz um espécie de comutador virtual suplementar na rede e associa um identificador virtual de comutador à sua interface para o canal. Assim, os n comutadores interligados passam a corresponder a $2 \times (n - 1)$ adjacências, $n - 1$ de interligação do *designated router* com os outros $n - 1$ comutadores, e outros $n - 1$ no sentido contrário. Todas as comunicações entre estes $n - 1$ comutadores passam necessariamente pelo *designated router*, ver a Figura 16.8.

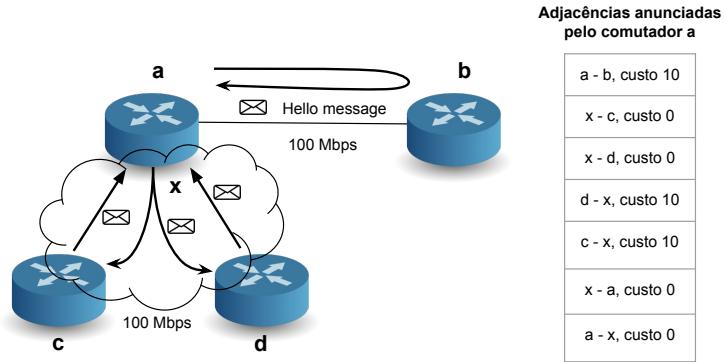


Figura 16.8: Entre os comutadores directamente ligados a um canal multi-ponto, um é eleito *designated router* e passa a ser responsável por um pseudo comutador, com o objectivo de evitar a explosão do número de canais ponto-a-ponto da rede.

Disseminação fiável do estado e partições da rede

Diz-se que se forma uma partição da rede quando a mesma fica dividida em duas ou mais partes incapazes de comunicarem entre si. Por exemplo, se na rede da Figura 16.9 (a) o canal entre os comutadores *c* e *d* avariar, o seu custo torna-se infinito e as duas partições resultantes, ver a Figura 16.9 (b), ficam incomunicáveis entre si.

Na sequência da avaria, o comutador *c* dissemina aos comutadores *a* e *e* que o canal passou a ter o custo infinito. A aplicação do algoritmo de Dijkstra nos outros comutadores conduz a uma LSDB e uma FIB nos comutadores *a*, *c* e *e* em que os destinos *b*, *d* e *f* estão à distância infinito, i.e., inacessíveis. Simetricamente, nas FIBs dos comutadores *b*, *d* e *f* os destinos *a*, *c* e *e* passaram a estar à distância infinito. Se a partição durar mais que o TTL máximo, de cada lado da rede, os LSAs emitidos pelos comutadores da outra partição acabarão por ser esquecidos por falta de refreshamento, visto que os LSAs de um lado não passam para o outro e vice-versa.

No entanto, que acontece se algum tempo depois da avaria esta recupera e a comunicação se estabelece de novo? Até que novos LSAs sejam emitidos por todos

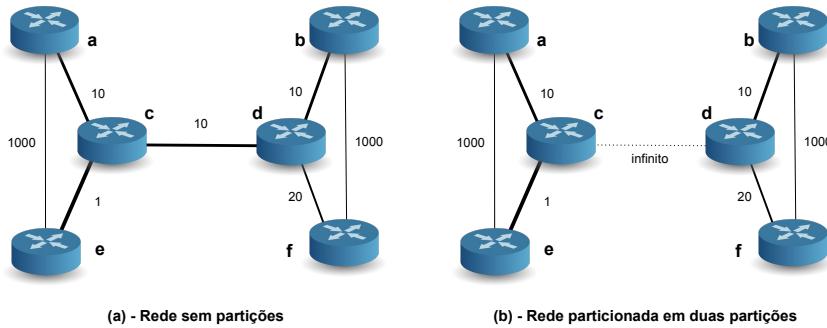


Figura 16.9: Rede completa com todos os canais operacionais (a). Na sequência da avaria do canal entre os comutadores c e d, a rede fica partitionada em duas incomunicáveis entre si (b).

os comutadores de cada lado, a outra parte continuaria com uma visão eventualmente desfasada do estado da rede. Para recuperar o mais rapidamente que possível de uma partição, quando um comutador se apercebe que um evento acabou de reparar uma partição com outro, ambos os comutadores cuja comunicação foi restabelecida trocam mutuamente todos os LSAs que conhecem. Ao executarem o protocolo de disseminação fiável da Listagem 15.1 para cada um dos LSAs, sincronizam e actualizam as respectivas adjacências.

Este processo designa-se nas implementações concretas *to bring up adjacencies* e é optimizado através da troca de mensagens designadas por *database descriptive packets*, trocadas entre os comutadores, e que contém os identificadores dos LSAs conhecidos e os respectivos números de sequência.

O mesmo mecanismo permite que um comutador que acabou de se ligar à rede passe a conhecer toda a LSDB da rede e possa computar uma FIB que lhe possibilitará o envio de pacotes para todos os destinos acessíveis na mesma.

Adicionalmente, o mecanismo que permite a um comutador, quando recebe um LSA atrasado, enviar ao outro a versão actual do mesmo LSA, permite que um comutador que se desligou da rede, e se liga mais tarde antes que o TTL não leve ao seu esquecimento pela rede, não seja ignorado por todos os outros comutadores. Com efeito, esse comutador inicializa o número de sequência dos seus LSAs a 1 mas os outros conhecem-no por números de sequência mais elevados. O mecanismo permite-lhe ficar a conhecer o último número de sequência que usou.

Convergência

Quando um ou mais eventos que alteraram a configuração da rede têm lugar, geram-se LSAs que são propagados pelos diferentes comutadores. Quando os outros comutadores recebem esses LSAs, alteram as suas LSDBs e na sequência disso alteram as respectivas FIBs.

Chama-se **tempo de convergência** (*convergence time*) de um protocolo de encaminhamento ao tempo que passa desde que se produz uma alteração da rede (e.g., um canal avariou-se), até ao momento em que todos os comutadores actualizaram as RIBs e FIBs e toda a rede voltou a estar num estado estável.

Este processo tem lugar de forma assíncrona nos diferentes comutadores e por isso todas as LSDBs e as FIBs dos comutadores não são alteradas instantaneamente e de forma sincronizada. Enquanto essas alterações não estabilizarem, os diferentes comutadores têm visões distintas da rede e podem encaminhar os pacotes em contradição

uns com os outros. Como resultado dessas incoerências, alguns pacotes podem entrar em ciclo, como é ilustrado na Figura 16.10.

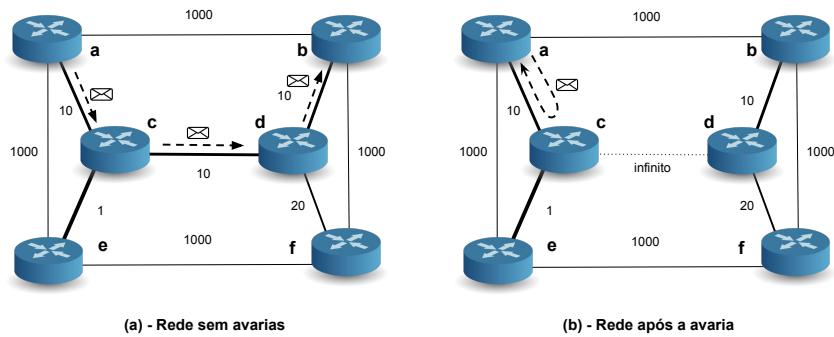


Figura 16.10: Antes da avaria do canal entre os comutadores c e d , parte (a), os pacotes que vão de a para b seguem via o caminho a , c , d , b por este ser o mais curto. Depois da avaria, parte (b), o comutador c acha que o melhor caminho é via a , b mas o comutador a ainda não actualizou a sua FIB e os pacotes entram em ciclo.

Para que estas incoerências momentâneas não desencadeiem uma catástrofe, todos os pacotes de dados assim encaminhados devem dispor de um mecanismo de segurança baseado num TTL. Se algum pacote entrar em ciclo durante a incoerência, acabará, mais tarde ou mais cedo, por ser suprimido.

Idealmente, o tempo de convergência deve ser o mais breve que possível, quer para evitar gastar recursos inutilmente durante a instabilidade, quer porque durante a mesma se perdem pacotes, o que pode afectar as aplicações, nomeadamente as que envolvem facetas de tempo real e que são não elásticas, como a generalidade das aplicações multimédia.

À primeira vista, a maneira mais simples de diminuir o tempo de convergência consiste em diminuir todos os tempos de reação aos diferentes eventos: deteção da alteração, início da difusão do LSA, recepção do LSA, computação do novo caminho mais curto e alteração da FIB. Infelizmente, a prática mostrou que isso pode conduzir à instabilidade da rede.

Com efeito, uma alteração, por pequena que seja, conduz ao envio de mais do que um LSA. Pelo menos dois, *i.e.*, um por cada um dos comutadores envolvidos se se tratar de um canal ponto-a-ponto. Se a avaria for de um comutador, serão enviados tantos LSAs quantos os canais ao qual ele estava ligado. Em certas situações isso pode envolver dezenas de LSAs.

Se a reação for muito rápida, isso pode conduzir a que cada comutador da rede desencadeie vários cálculos e alterações das FIB e RIB sucessivamente, cada uma das quais envolvendo cada um dos LSAs recebidos. Por outro lado, se a rede for complexa, o algoritmo de Dijkstra envolve muitos nós e canais e é computacionalmente pesado. Por isso, é preferível não o executar senão um certo tempo depois de o ter executado pela última vez, e também deixar passar algum tempo desde que chega um LSA antes de o desencadear, na esperança que cheguem todos os LSAs relacionados com o mesmo evento.

Por outro lado, é frequente produzirem-se avarias intermitentes. Por exemplo, um canal cujo estado alterna entre estar operacional e deixar de o estar. Resultado, uma instabilidade de uma componente da rede pode desencadear uma instabilidade generalizada da rede. Uma maneira simples de realizar um ataque de negação de

serviço consiste em enviar periodicamente LSAs falsos que conduzam a reconfigurações constantes de todos os comutadores.

Na prática, os temporizadores associados ao desencadear da reação aos diferentes eventos têm de ser elevados. Quando os protocolos de encaminhamento com base no estado dos canais foram introduzidos, o valor por omissão dos alarmes usados levavam a que o tempo de convergência fosse da ordem de grandeza de 10 segundos. Isso era reforçado pelo facto de que os comutadores usados na época tinham pouco poder computacional e pouca memória, e considerava-se que era preferível uma convergência mais lenta, do que uma grande instabilidade em toda a rede induzida por constantes reconfigurações, ou comutadores com a CPU saturada e incapazes de responderem aos diferentes eventos que iam tendo lugar.

Na verdade, na gestão das redes, é necessário fugir “como o diabo da cruz” de toda a instabilidade, seja qual for a sua origem: *bugs* de software nos comutadores, comutadores com a CPU saturada e incapazes de reagirem atempadamente aos eventos, instabilidades no hardware, canais com problemas, erros dos gestores por precipitação em casos de emergência, *etc.* Actualmente fizeram-se bastantes progressos na engenharia dos comutadores e na implementação concreta destes protocolos e é possível obter tempos de convergência inferiores a 1 segundo em redes complexas [Francois et al., 2005].

Métricas de custo

O problema da convergência permite igualmente ilustrar porque razão as métricas de custo usadas na prática não tomam em consideração alterações relacionadas com modificações da intensidade do tráfego e da dimensão das filas de espera. A Figura 16.11 mostra porquê. Na mesma, as etiquetas de cada canal mostram a intensidade do tráfego que o atravessa. Por hipótese, neste exemplo, o custo de um canal considera-se proporcional a essa intensidade de tráfego, dado que os débitos e tempos de trânsito dos canais são idênticos. Atendendo a que o custo é função da intensidade do tráfego, o custo varia com a direcção do tráfego e um canal tem custos distintos nas duas direções. Quando o custo não está assinalado na figura numa dada direcção, isso quer dizer que o mesmo custo é desprezável pois não há tráfego nessa direcção.

As setas junto a cada comutador mostram o tráfego que cada um deles injecta na rede. Por hipótese, os comutadores **a** e **c** injectam a mesma quantidade x de tráfego na rede, em ambos os casos dirigido ao comutador **b**. O comutador **d** injecta a quantidade δ (muito pequena em face de x) também dirigida ao comutador **b**.

A Figura 16.11 (a) mostra as decisões de encaminhamento tomadas pelos comutadores logo após o aparecimento do tráfego δ . O comutador **d** toma a decisão de enviar o tráfego δ pela direita, ora essa decisão faz com que o custo visto por **c** para enviar tráfego para **b** seja superior ao de enviá-lo pela outra alternativa, e a rede reconfigura-se para usar o encaminhamento ilustrado na Figura 16.11 (b). Só que agora o comutador **a** vê o custo com que envia tráfego para **b** aumentar e a rede altera-se para a configuração de encaminhamento da Figura 16.11 (c). O mesmo raciocínio leva a uma nova reconfiguração para a situação da Figura 16.11 (d), e assim sucessivamente.

Por conseguinte, fazer a intensidade de tráfego ser considerada de forma muito “apertada” na métrica do custo pode conduzir a situações de instabilidade suplementares.

Em conclusão, a técnica de encaminhamento designada como encaminhamento pelo estado dos canais requer cuidados na sua concretização, relacionados com a não atomicidade com que a alteração da LSDB nos diferentes comutadores tem lugar. Isso complica a implementação concreta desta forma de fazer o encaminhamento, nomeadamente no que diz respeito à coordenação da evolução do estado replicado nos diferentes comutadores. Estas dificuldades estão relacionadas com alguns dos problemas fundamentais dos sistemas distribuídos, nomeadamente a gestão de estado replicado e a coordenação das diferentes componentes do sistema distribuído. Estas dificuldades

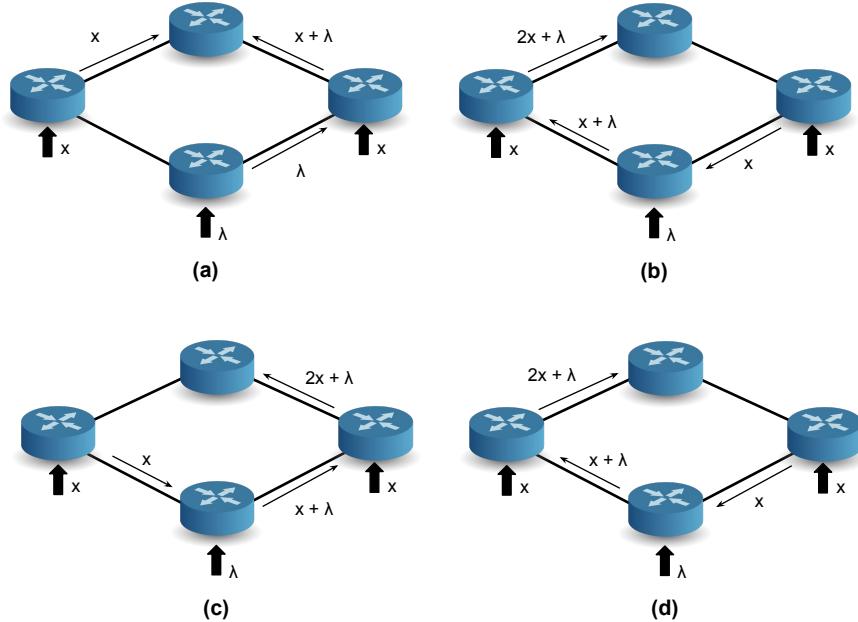


Figura 16.11: Considerar a intensidade do tráfego ou a dimensão das filas de espera de forma significativa na métrica do custo de um canal pode conduzir a instabilidade no encaminhamento.

estão relacionadas com a ausência de um relógio global que sincronize as diferentes componentes do sistema.

A técnica de encaminhamento pelo caminho mais curto com base no **estado dos canais** (*link-state routing*) baseia-se na replicação por todos os comutadores, por difusão fiável, de todas as adjacências existentes na rede. Desta forma, cada comutador adquire a visão do grafo que caracteriza a rede e, usando o algoritmo de Dijkstra, pode calcular os melhores caminhos para todos os destinos existentes e assim construir a sua FIB.

Sempre que têm lugar modificações do estado de qualquer componente da rede, os comutadores que as detectam difundem actualizações, também designadas SLAs (*Link State Announcements* ou *Adjacency Announcements*) e, logo que a difusão fiável se completar, as actualizações são repercutidas nas FIBs de todos os comutadores e a rede converge voltando a um estado estável.

Existem dois protocolos concretos baseados nesta filosofia de encaminhamento: o protocolo IS-IS (*Interior System to Interior System*), definido pela norma ISO/IEC (ISO/IEC 10589:2002, Second Edition), e o protocolo OSPF (*Open Shortest Path First*), ver os RFCs 2328 (IPv4) e 5340 (IPv6). O primeiro foi definido, inicialmente, como sendo mais genérico e mais flexível pois permitia a extensão do tipo de informações transportadas pelos LSAs, um mecanismo conhecido como *type-length-value* (TLV), que o tornou independente do protocolo de encaminhamento concreto que suportava pois é capaz de replicar diversos tipos de informações.

O segundo foi introduzido como uma especialização do primeiro para a comutação de pacotes IPv4. Esta especialização teve repercuções posteriores. Por exemplo, a utilização de IS-IS para IPv6 não requereu nenhuma nova versão do protocolo, enquanto que no caso do OSPF foi necessário introduzir uma nova versão. No entanto, a quantidade de opções do OSPF para a gestão concreta de uma rede IP era mais completa e a extensão do tipo da informação transportada pelos LSAs também foi posteriormente introduzida no OSPF.

A opção por um ou por outro protocolo depende muitas vezes da maturidade da implementação nos equipamentos e das opções de gestão dos mesmos. Geralmente, o IS-IS é mais usado pelos ISPs, enquanto que o OSPF é preferido nas redes empresariais. No entanto, esta regra não é obrigatória.

O protocolo IS-IS é também usado para replicar estado genérico entre um grupo de comutadores fora do mundo da comutação de pacotes IP. Por exemplo, o IS-IS é usado para replicar estado nas novas propostas de encaminhamento para *switches* Ethernet com base em endereços MAC, nomeadamente TRILL (Transparent Interconnection of Lots of Links) e Shortest Path Bridging, ambas referidas no final do capítulo anterior.

16.4 Encaminhamento pelo algoritmo Bellman-Ford

Antes da introdução do método de encaminhamento com base no estado dos canais foram usadas outras formas de encaminhamento. Nas redes mais antigas e simples, era comum usar exclusivamente encaminhamento estático, com todos os defeitos deste método já assinalados. Foram também usados métodos dinâmicos baseados num controlador central que alterava as FIBs dos comutadores se o estado da rede se alterasse. Em ambos os casos os comutadores não necessitavam de uma RIB, apenas dispunham de uma FIB que era remotamente alterada.

Mais tarde, surgiu a ideia de dar inteligência e autonomia aos comutadores de forma que estes se adaptassem automaticamente às alterações do estado da rede. No entanto, antes da introdução do método visto na secção anterior, foi usada uma outra forma de encaminhamento distribuído, autónomo e dinâmico, baseado no algoritmo de Bellman-Ford, do nome dos seus dois inventores. Estes introduziram o mesmo algoritmo de forma independente e quase simultânea, pelo que é tradicional dar a ambos o crédito pela sua invenção. No final da secção compararemos as duas abordagens: encaminhamento com base nos estados dos canais e encaminhamento com base no algoritmo de Bellman-Ford.

Esta última forma também costuma ser designada por encaminhamento com base em vectores de distâncias (a distância é, como vimos atrás, a nomenclatura convencional para o custo de um caminho), pois uma das formas mais populares da sua concretização passa por os comutadores trocarem entre si vectores contendo a lista dos destinos que cada um conhece e a que distâncias os consegue alcançar.

Esta abordagem parte de uma ideia muito simples: se um comutador x conhece um conjunto de destinos, pode comunicar esse facto aos seus vizinhos directos. Quando um deles recebe essa informação, dado que conhece a distância (custo) que o separa de x , fica a conhecer uma (nova) forma de alcançar esses destinos somando o custo de chegar ao vizinho x , que lhe fez o anúncio, às distâncias por este anunciadas. Se essa forma de alcançar um dado destino k for melhor que a que o comutador conhecia previamente, então pode passar a encaminhar os pacotes dirigidos a k via o vizinho x . Os anúncios enviados pelos comutadores aos seus vizinhos chamam-se **vectores de distâncias** (*distance vectors*) ou **anúncios de visibilidade** (*reachability announcements*), ver a Figura 16.12.

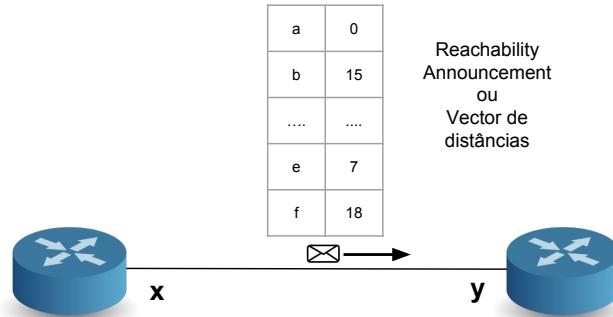


Figura 16.12: O comutador x passa a um seu vizinho y o conjunto de destinos que conhece e a que distância os consegue alcançar, através de um vector de distâncias ou *reachability announcement*.

Cada comutador guarda na sua RIB os anúncios de visibilidade que recebeu dos seus vizinhos directos. Através destes pode determinar qual o caminho mais curto para chegar a cada destino. Antes de prosseguirmos, convém ter presente que um comutador só passa anúncios aos seus vizinhos directos. Estes, processam os anúncios recebidos, preenchem a sua FIB e, periodicamente, ou sempre que tiver lugar uma alteração, constroem de novo um anúncio de visibilidade, e passam-no a todos os seus vizinhos directos. Não se trata portanto de um algoritmo de difusão fiável pois cada comutador envia aos seus vizinhos a sua visão, que é diferente da dos outros. Os comutadores só guardam na RIB os anúncios dos seus vizinhos directos, não os anúncios de todos os comutadores da rede.

Quando recebe um anúncio de um vizinho, um comutador pode actualizar imediatamente a sua FIB pois o processamento do anúncio é fácil. A listagem 16.2 indica uma versão preliminar desse processamento.

Listing 16.2: Processamento de um anúncio pelo algoritmo de Bellman-Ford

```

1 Input:
2   x - node that sent the announcement
3   d - distance or cost to reach x
4   reachability[1..k] - received reachability announcement
5   distance[1..k] - FIB - distance to each node (1..k)
6   nexthop[1..k] - FIB - neighbor selected to reach each node (1..k)
7
8   for i in 1..k do {
9     if ( reachability[i] + d < distance[i] ) {
10       distance[i] = reachability[i] + d
11       nexthop[i] = x
12   }

```

Isto é, “se x anunciar um destino i , a uma distância tal que a soma dessa distância com o custo para alcançar x , for menor que o caminho conhecido actualmente para chegar a i , então a melhor maneira que se tem de alcançar o destino i é via x , e i passará a esta à distância de chegar a x mais a distância com que x consegue chegar a i ”.

Para se perceber como o algoritmo permite construir dinamicamente as FIBs de todos os comutadores de uma rede vamos admitir, por hipótese, que a rede está integralmente sincronizada por um hipotético relógio global, e que cada nó executa de forma sincronizada um passo do algoritmo sempre que o relógio emite um sinal. Em cada comutador este passo é assim constituído:

Compor um vector de distâncias ou anúncio de visibilidade
 Enviá-lo a cada um dos vizinhos
 Receber os vectores enviados pelos vizinhos
 Processá-los em sequência de acordo com o algoritmo 16.2 e actualizar a FIB

Após a inicialização, cada comutador só conhece o seu próprio destino (por hipótese o seu identificador como na secção anterior) assim como o dos seus vizinhos directos e os custos para os alcançar. Esta informação é fácil de adquirir através, por exemplo, de uma mensagem `hello`. Depois, a cada sinal do relógio ou iteração, todos os nós começam por enviar o seu vector de distâncias aos seus vizinhos directos, lêem os vectores distâncias recebidos, processam-nos e actualizam a sua FIB. O processo acaba quando a rede convergir. Por outras palavras, quando todos os nós ao receberem vectores distâncias, já não “aprenderem” mais nada com os seus vizinhos (ou seja, qualquer novo vector de distâncias recebido não altera nenhuma entrada de nenhuma FIB). A rede usada para exemplificar será a mesma que já usámos na ilustração do funcionamento do algoritmo de Dijkstra.

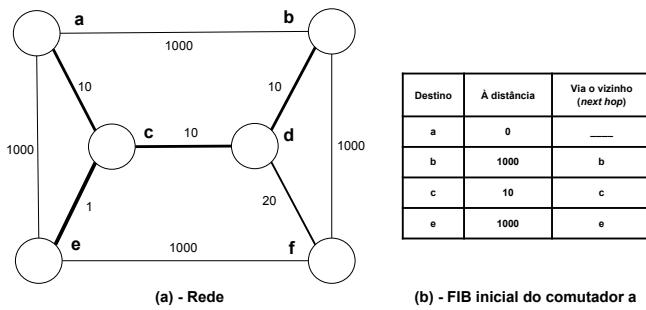


Figura 16.13: Rede usada no exemplo e FIB inicial do comutador a

A Figura 16.13 (a) mostra a rede com as informações necessárias para preencher a FIB de cada comutador. A Figura 16.13 (b) mostra a FIB inicial do comutador a. A Figura 16.14 mostra a execução de um passo do algoritmo do ponto de vista do comutador a. A FIB inicial, Figura 16.14 (a), é a FIB do comutador após a sua inicialização. A Figura 16.14 (b) mostra os vectores de distâncias que lhe são enviados pelos vizinhos, e a Figura 16.14 (c) mostra a mesma FIB depois de esta incorporar o processamento de cada um dos vectores recebidos, pela ordem indicada. As entradas que foram modificadas na FIB do comutador a estão sombreadas se foram alteradas pelo processamento dos anúncios recebidos dos vizinhos.

Com o vector do comutador b, o comutador a passa a conhecer o comutador d à distância 1010 (1000 para b mais 10 deste para d) e o destino f à distância 2000 (1000 para b mais 1000 deste para f). Com o vector do comutador c, o comutador a passa a conhecer o comutador d à distância 20 (10 para c mais 10 deste para d e melhor que os 1010 anteriores) e o destino e passou a estar à distância 11 (10 para c mais 1 deste para e). Com o vector do comutador e, o comutador a não obtém nenhuma informação que o leve a alterar o encaminhamento escolhido.

Neste último caso, poder-se-ia perguntar: mas se a já sabe agora que existe um caminho para e com custo 11, porque não usar essa informação para ir para f via esse caminho à distância de 11 mais o custo com que e conhece um caminho para f, ao invés de considerar a sua distância a e nos cálculos? De facto seria possível fazer uma outra versão do algoritmo que analisasse mais profundamente o conteúdo

da RIB, no entanto, o facto é que se existir uma forma melhor de chegar a um destino, ele aparecerá necessariamente num anúncio posterior do comutador que dá início ao mesmo, como se verá a seguir.

1 - vector distância enviado por b

Destino	a	b	d	f
Distância	1000	0	10	1000

2 - vector distância enviado por c

Destino	a	c	d	e
Distância	10	0	10	1

3 - vector distância enviado por e

Destino	a	c	e	f
Distância	1000	1	0	1000

(a) - FIB do comutador a antes da primeira iteração (b) - Vectores distância recebidos pelo comutador a (c) - FIB após o processamento dos anúncios

Figura 16.14: (a) FIB inicial do comutador a. (b) vectores distância recebidos durante a primeira iteração. (c) FIB após processamento pela ordem dos vectores distância recebidos.

A Figura 16.15 mostra a evolução da FIB dos restantes comutadores no fim da primeira iteração. Por falta de espaço não se mostra a FIB inicial, nem os vectores de distâncias recebidos por cada comutador dos seus vizinhos directos.

Dst	d.	next hop
a	1000	a
b	0	—
c	20	d
d	10	d
e	2000	f
f	30	d

FIB do comutador b

Dst	d.	next hop
a	10	a
b	20	d
c	0	—
d	10	d
e	1	e
f	30	d

FIB do comutador c

Dst	d.	next hop
a	20	c
b	10	b
c	10	c
d	0	—
e	11	c
f	20	f

FIB do comutador d

Dst	d.	next hop
a	11	c
b	2000	a
c	1	c
d	11	c
e	0	—
f	1000	f

FIB do comutador e

Dst	d.	next hop
a	2000	b
b	30	d
c	30	d
d	20	d
e	1000	e
f	0	—

FIB do comutador f

Figura 16.15: FIB dos restantes comutadores no fim da primeira iteração

A Figura 16.16 mostra a evolução da FIB do comutador a durante a segunda iteração. Os vectores recebidos por a já incorporam o que os seus vizinhos aprenderam com os vizinhos deles no final do passo anterior. Com o vector enviado por b, a apenas aprende que existe um caminho para f via b e este ficará à distância 1030. No entanto, com o vector enviado por c é possível concluir que existe um melhor caminho via este comutador e f ficará à distância 40. Com esse mesmo vector é possível concluir que via o mesmo comutador existe um caminho para b e este ficará à distância 30.

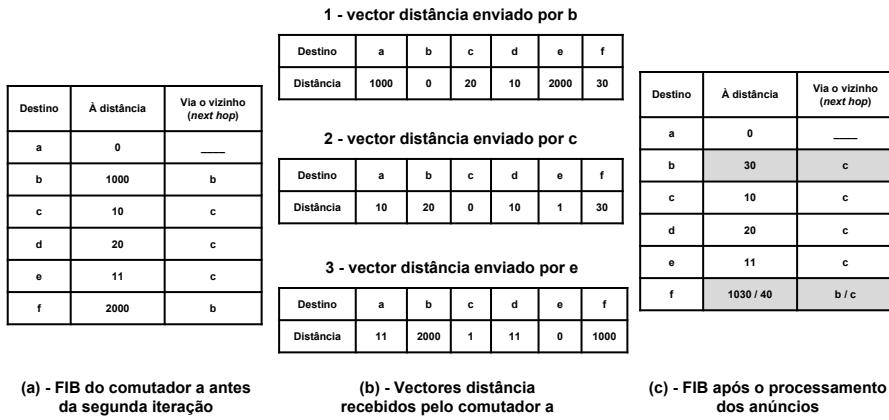


Figura 16.16: (a) FIB do comutador a antes da segunda iteração. (b) vectores distância recebidos dos vizinhos de a. (c) FIB do comutador a após a segunda iteração.

Novas iterações não conduzirão à descoberta de alternativas melhores para chegar aos diferentes destinos e o processo estabilizará. É possível concluir isso executando mais iterações. No entanto, é também possível observar o seguinte: após a inicialização, todos os comutadores conhecem caminhos de comprimento 1 para os seus vizinhos. No fim da primeira iteração passam a conhecer caminhos de comprimento 2 e destes já selecionaram os melhores. Finalmente, no fim da segunda, todos os comutadores já conhecem os melhores caminhos de comprimento 3. Ora como não existem caminhos mais curtos de comprimento maior, os mesmos não podem ser descobertos através de iterações suplementares.

Se considerarmos que durante a inicialização cada comutador apenas se descobre a si próprio, e só passa aos seus vizinhos directos um primeiro vector com a sua própria identificação, então o algoritmo de Bellman-Ford consegue que os comutadores descubram os melhores caminhos para todos os outros comutadores no máximo em tantos passos como o diâmetro da rede, i.e., o comprimento do maior caminho mais curto existente na mesma.

Numa implementação concreta do algoritmo, i.e., num protocolo de encaminhamento, é necessário tomar em consideração que não é fácil introduzir um relógio global que sincronize a execução em todos os comutadores. Também não é possível garantir que todos os anúncios são recebidos num tempo bem definido e que não há atrasos nem perda de mensagens. Assim, as implementações concretas são assíncronas e têm mecanismos para combater a perda de mensagens, e.g., reenviando os anúncios periodicamente. Por outro lado é necessário ter em consideração que a rede muda de estado dinamicamente. Por exemplo, se um canal se avariar, no momento seguinte todos os caminhos que o usavam deixam de ser viáveis e o algoritmo tem também de tomar isso em consideração. Por isso, o vector de distâncias de cada comutador altera-se também sempre que a distância aos seus vizinhos se modifica. Os mais conhecidos protocolos de encaminhamento baseados no algoritmo de Bellman-Ford enviam novos vectores distâncias aos vizinhos após a inicialização, sempre que esses vectores se alteram e ainda periodicamente para compensar eventuais perdas de mensagens e controlar longas avarias de comutadores.

A listagem 16.3 introduz o pseudo-código de uma versão mais completa, mas ainda assim simples, do protocolo. Este funciona de forma assíncrona pois as acções têm lugar na sequência de eventos exclusivamente locais a cada comutador. Adicionalmente,

Listing 16.3: Pseudo código de um protocolo de encaminhamento com base no algoritmo de Bellman-Ford

```

1 Local data including the FIB:
2   distance[1..k] - distance to reach node i
3   nexthop[1..k] - the first hop in the shortest path to reach node i
4   boolean neighbours[1..k] - node i is a direct neighbour
5
6 On event < START >:
7   for i in 1..k do {
8     distance[i] = infinity
9     nexthop[i] = null
10    }
11   distance[myself] = 0
12   nexthop[myself] = myself
13   // probe links, discover neighbours, initialize neighbours[1..k]
14   // and update their distance and nexthop
15   sendReachability(true)
16
17 On event < link to neighbour i becomes INOPERATIONAL >:
18   neighbours[i] = false
19   // update path to all destinations using i as next hop
20   for j in 1..k do if ( nexthop[j] == i ) {
21     nexthop[j] = null
22     distance[j] = infinity
23   }
24   sendReachability(false)
25
26 On event < link to neighbour i returns to OPERATIONAL >:
27   neighbours[i] = true
28   // if we previously knew no other path to i
29   if ( nexthop[i] == null ) {
30     nexthop[i] = i
31     distance[i] = its value
32     sendReachability(true)
33   }
34
35
36 On event < PERIODIC TIMER triggered >:
37   sendReachability(true)
38
39 On event < receive reachability ANNOUNCEMENT (reachability[1..k]) from
40 neighbour x > : // at distance d
41   for i in 1..k do {
42     // if we route to i by x, always update distance
43     if ( nexthop[i] == x ) distance[i] = reachability[i] + d
44     // else try to learn about better alternatives
45     else if ( reachability[i] + d < distance[i] ) {
46       distance[i] = reachability[i] + d
47       nexthop[i] = x
48     }
49   }
50   sendReachability(false)
51
52 void sendReachability(boolean always) {
53   if ( always or FIB has been changed ) {
54     send reachability announcements
55     (distance[1..k]) to direct neighbours
56   }

```

os vectores de distâncias são enviados periodicamente para combater eventuais perdas de mensagens. Por outro lado, se algum evento provocar uma alteração da FIB, o novo vector de distâncias é enviado a todos os vizinhos. Nas implementações concretas do algoritmo os anúncios enviados na sequência de alterações das FIBs são designados *triggered updates*. Estes anúncios devem conter necessariamente todos os destinos que foram alterados, mas também podem conter por omissão o vector de distâncias completo. Eles não são enviados imediatamente, mas só depois de um pequeno compasso de espera aleatório, que evita que se forme uma cascata sincronizada de actualizações em todos os comutadores. A listagem também omite o tratamento das mensagens *hello* que permitem a deteção do estado e do custo de atingir os vizinhos.

Um outro aspecto a ter em atenção está relacionado com o facto de que esta versão do algoritmo não usa a memorização dos anúncios recebidos dos vizinhos. Por isso, quando um canal para um vizinho se avaria, todos os destinos que o usavam são colocados à distância infinito, sem recorrer a eventuais alternativas via outros vizinhos. Esta opção é possível, na medida em que mais tarde ou mais cedo aparecerão anúncios de outros vizinhos que restabelecerão a conectividade para esses destinos, caso existam alternativas.

A mesma opção é usada quando o comutador recebe um anúncio de um comutador i e esse comutador é usado como vizinho (*next hop*) através de qual se atinge um dado destino x . Qualquer alteração da distância a que o vizinho v_i é imediatamente repercutida na FIB sem a preocupação, no caso da subida da distância, de procurar imediatamente outras alternativas via outros vizinhos. Tal é impossível pois os seus vectores de distância não foram guardados e pressupõe-se igualmente que essas melhores alternativas serão anunciadas mais tarde ou mais cedo, caso existam.

A seguir ficará mais claro que esta opção, para além de ser mais simples, contribui para não desencadear potenciais efeitos laterais indesejáveis.

O algoritmo de Bellman-Ford baseia-se na troca de informação entre nós vizinhos sobre os destinos que conhecem e as distâncias a que os alcançam.

É fácil de perceber e simples de incorporar num protocolo de encaminhamento, apenas obriga à existência de uma FIB que incorpore a noção de distância pelo que a informação local obrigatória é mínima. O processamento de cada mensagem é simples e linear com a dimensão dos anúncios, os quais são proporcionais ao número de destinos existentes na rede.

Infelizmente, como veremos a seguir, nem sempre o algoritmo tem um comportamento tão benigno.

Convergência do protocolo

À primeira vista a convergência de protocolos baseados no algoritmo Bellman-Ford é relativamente rápida. Na sequência de uma alteração da FIB de um comutador, este envia novos vectores de distâncias aos vizinhos, estes processam-nos com baixo custo e, caso as suas FIBs se alterem, propagam novos vectores. O número máximo de iterações é, à primeira vista, limitado pelo diâmetro da rede. Com efeito, sempre que a alteração consiste na descida do custo de um canal (*e.g.*, passou de infinito a finito) ou no aparecimento de um novo destino (*e.g.*, foi ligado um novo comutador), o algoritmo converge para a determinação dos caminhos mais curtos, o que foi provado matematicamente pelos seus inventores.

Infelizmente, como veremos a seguir, no caso de a alteração ser no sentido contrário (*e.g.*, um canal passou de um custo finito para infinito porque avariou), a convergência pode ser muito lenta pois os comutadores podem enviar inadvertidamente vectores

de distâncias com informações falsas (*e.g.*, considerar destinos acessíveis quando na verdade já não o estão, só que o comutador em questão ainda não foi disso notificado).

A figura 16.17 permite ilustrar o problema. Na Figura 16.17 (a), à esquerda, o comutador (x) é introduzido na rede. Assim que o canal x - c ficar operacional e passar de inexistente a existente, o comutador c divulga imediatamente um vector de distâncias contendo o destino x à distância 1 (admitamos que esse é o custo de todos os canais); logo a seguir o comutador b introduz na sua FIB que o destino x é acessível via c à distância 2 e anuncia a a que o destino x está acessível à distância 2; finalmente, o comutador a actualiza a sua FIB para passar a conter a indicação de que o destino x é acessível via b à distância 3. Quaisquer anúncios futuros, num sentido ou outro, não podem introduzir alterações enquanto a configuração da rede se mantiver estável. A convergência deu-se rapidamente num número de passos limitado pelo diâmetro da rede.

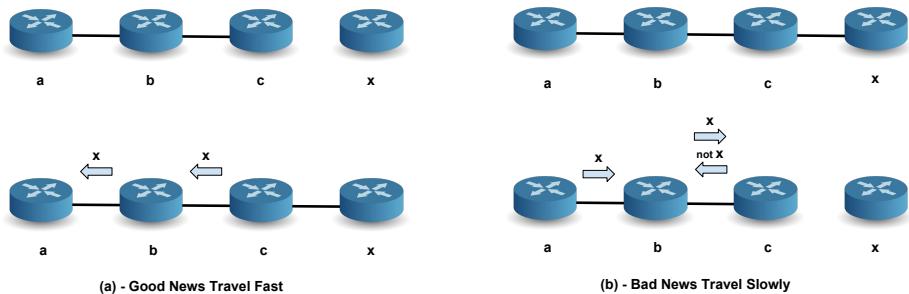


Figura 16.17: Ilustração do fenómeno “as boas notícias correm velozes, mas as más devagar” (“*good news travel fast, while bad news travel slowly*”).

Na Figura 16.17 (b), à direita, ilustra-se a situação inversa: o canal que liga c a x avariou-se e passou de um custo finito a infinito. O processo que se passou no caso (a) pode ter agora lugar no caso (b) e todos os comutadores passam a considerar que o destino x deixou de estar acessível. Infelizmente as coisas não são assim tão simples pois os anúncios não fluem só da direita para a esquerda, mas também no sentido contrário. Por outro lado, o momento em que cada comutador envia anúncios não está sincronizado com os outros. Que acontece se b anuncia a c que conhece um caminho para x com custo 2 antes de receber de c a indicação de que x deixou de estar acessível?

Não há nenhuma razão para que c não passe a acreditar que existe um outro caminho para x com custo 3 via b visto que com o algoritmo Bellman-Ford os anúncios são opacos sobre os caminhos usados para os destinos anunciados. A mesma conclusão poderia ser tomada por c caso usasse uma versão do algoritmo em que guardasse os anúncios recebidos dos outros nós e resolvesse, sempre que um custo de um caminho que está a usar subisse, ir ver se não haveria alternativa melhor via outro vizinho. A partir do momento em que c acreditasse que existe um caminho para x via b, iria anunciarlo a b, b aumentaria o custo com que conseguia chegar a x e voltava a anunciarlo a c mas com custo maior, e assim sucessivamente. Nada pode parar este ping pong de anúncios excepto quando o custo anunciado for tão alto que os comutadores o equiparem a infinito.

Este fenómeno pode surgir quando um anúncio falso se introduz na rede simplesmente porque o “mentiroso” ainda não se apercebeu de que uma informação que recebeu deixou de ser válida. Quando o problema aparece, aparecem ciclos de pacotes e alguns canais podem deixar de ser utilizáveis pois ficam saturados pelo tráfego, quer de anúncios, quer de eventuais pacotes em ciclo. O fenómeno foi designado por “contagem para o infinito” ou “as boas notícias correm velozes, mas as más

devagar” (“good news travel fast, while bad news travel slowly”).

Foram propostas e são usadas diversas abordagens que mitigam a possibilidade de esta anomalia ter lugar. A primeira destas abordagens consiste em tentar limitar a duração do ping pong. Por exemplo, com o protocolo RIP (Routing Information Protocol), ver o RFC 2453, todos os canais, seja qual for o seu débito, só podem ter custo 1, logo a métrica do custo de um caminho é equivalente ao seu comprimento. No protocolo RIP considera-se que o maior custo de um caminho é 15 pois infinito toma o valor 16. Esta solução limita o número máximo de anúncios em ping pong mas também limita o diâmetro da rede e a métrica do custo.

Uma segunda solução consiste em tentar diminuir a probabilidade de se introduzirem mentiras (ou afirmações falsas mas que um dia foram verdadeiras) e consiste numa abordagem designada *split horizon with poison reverse*. A mesma consiste em o comutador **b** anunciar com custo infinito a um vizinho **c**, todos os destinos **x** tais que **b** usa **c** para lá chegar. Na verdade, a informação contrária é redundante e até meio falsa pois não vale a pena estar a anunciar a **c** que se conhece um caminho para **x** que passa pelo próprio **c**. Esta solução resolve o problema na rede da Figura 16.17. No entanto, é fácil reconhecer que não o resolve definitivamente em qualquer rede que tenha um ciclo, como por exemplo a rede ilustrada na Figura 16.18. A sua explicação deixa-se como exercício para o leitor, ver a Secção 16.6 no final do capítulo.

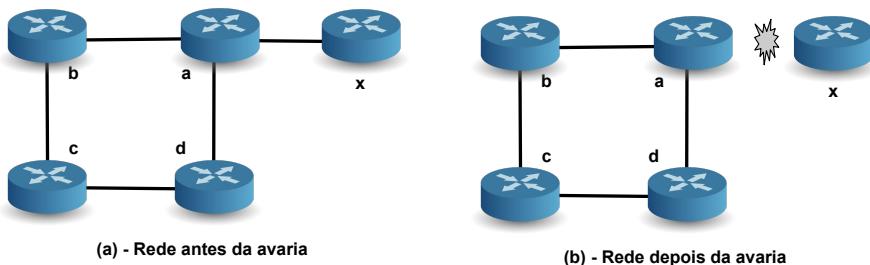


Figura 16.18: Rede onde a solução “*split horizon with poison reverse*” pode não resolver o problema da contagem para o infinito.

A terceira solução consiste em introduzir um mecanismo que tem como efeito ignorar as “boas notícias” durante algum tempo depois de se receberem “máx notícias”. De forma mais concreta, se um comutador recebe um anúncio de que o destino **x** passou a estar à distância infinito, ignora novos anúncios sobre o destino **x** durante algum tempo. O alarme que regula este compasso de espera chama-se *hold-down-timer*. Esta forma de proceder dá tempo a todos os comutadores de se aperceberem que **x** deixou de estar acessível e evita que os outros acreditem em falsidades sobre a sua acessibilidade. Para ser efectivo, a duração deste alarme tem de ser maior que o tempo de convergência da rede.

A solução anterior tem contudo um problema, não sendo o tempo de convergência à partida conhecido, e porque se podem perder mensagens, a duração do alarme tem que ser estimada por excesso, podendo resultar num valor demasiado elevado.

O problema suplementar é que o facto de **x** deixar de ser acessível por um certo caminho, não quer dizer que não continue acessível por outro. Ora esta solução tem o efeito indesejável de nesse caso não permitir que o outro caminho seja usado durante o período de espera. Neste caso, a “prudência” tem um elevado custo pois, apesar de se evitar a existência de ciclos e instabilidade na rede, a convergência é lenta, ficando sempre igual a pelo menos o tempo de espera introduzido pelo *hold-down timer*. A contagem para o infinito pode também surgir quando os comutadores guardam os anúncios dos seus vizinhos e, no caso de um destino deixar de estar acessível, usem

imediatamente outro que tenham memorizado pois este pode corresponder apenas a algo que foi verdadeiro mas que já não é.

Saindo do quadro estrito da utilização de vectores de distâncias, é possível adoptar outras soluções para o problema. Uma das mais comuns é a usada pelo protocolo BGP, ver o Capítulo 17, que consiste em complementar os vectores de distâncias com a indicação, para cada destino, do caminho usado para lá chegar. Os novos vectores dizem-se então vectores de caminhos. Assim, quando um comutador recebe um anúncio de visibilidade, pode analisar o caminho usado pelo vizinho para chegar ao destino. Se o comutador receptor figurar nesse caminho, o caminho deve ser rejeitado pois introduziria um ciclo. Neste caso, o facto de se guardarem os anúncios dos diversos vizinhos não é perigoso pois estes não podem conter ciclos.

Curiosamente, só neste quadro é possível de forma segura passar a usar um protocolo que não envie periodicamente os anúncios, mas apenas quando houver alterações. Repare-se que se os anúncios forem transmitidos de forma fiável apenas quando há alterações, caso um caminho deixe de estar disponível, outro caminho alternativo não será necessariamente anunciado senão quando houver alterações que o afectem. Isso quer dizer que se se trocarem anúncios de forma fiável, então é obrigatório memorizar os anúncios recebidos dos diferentes vizinhos.

Por isso, o problema da contagem para o infinito, a transmissão de anúncios de forma fiável e a memorização dos anúncios recebidos de todos os vizinhos são opções de implementação que estão todas interligadas.

O algoritmo Bellman-Ford serviu de base a quase todos os protocolos de encaminhamento que foram desenvolvidos inicialmente para as redes de pacotes experimentais e mesmo de produção. A sua simplicidade e adequação às capacidades computacionais dos *routers* e canais, então disponíveis, tornaram-no a opção preferida pelos implementadores.

Com a experiência foi-se constatando que o fenómeno designado “**contagem para o infinito**” ou “**as boas notícias correm velozes, mas as más devagar**” (“*good news travel, fast while bad news travel slowly*”) introduz problemas delicados de convergência.

Este problema pode ser minorado introduzindo diversos mecanismos, nomeadamente: limitar o número de *hops* máximo dos caminhos, usar anúncios do tipo *split horizon with poison reverse*, e introduzir *hold-down-timers*. Estes mecanismos minoram o problema sem o resolver definitivamente.

Infelizmente, as soluções definitivas conhecidas anulam a simplicidade do algoritmo. Assim, na prática, sempre que as redes têm alguma complexidade e a sua convergência é mais crítica, os protocolos baseados no estado dos canais são os preferidos.

Notas históricas

Protocolos baseados em vectores de distâncias foram usados pelas primeiras redes de pacotes experimentais que introduziram encaminhamento dinâmico, como a Arpanet e a Cyclades. Mais tarde a mesma filosofia foi adoptada numa rede experimental desenvolvida no Xerox Parc, onde foi usado pela primeira vez o acrônimo RIP (XNS-RIP).

O XNS-RIP serviu de inspiração para o desenvolvimento da primeira versão de RIP incluída na implementação de TCP/IP do sistema de operação Berkeley UNIX, ver a Secção 1.6. Como o código fonte deste sistema era público e podia ser reutilizado, a sua implementação de TCP/IP, incluindo a implementação de RIP, serviu de base às

implementações de TCP/IP em muitos outros sistemas, e o RIP tornou-se o protocolo de encaminhamento mais popular na Internet da época. Mesmo para ser implementado de novo, necessitava apenas de uma tabela e do processamento de duas mensagens e alguns temporizadores.

A versão inicial, ver o RFC 1058, introduziu uma versão muito simples do protocolo. Mais tarde os diferentes mecanismos descritos acima para combater o problema da contagem para infinito foram introduzidos, e as versões mais recentes, RIPv2, ver os RFCs 2453, e RIPng ou RIP para IPv6, ver o RFC 2080, passaram a inclui-los, nomeadamente *triggered updates, split horizon with poison reverse e hold-down timers*. No entanto, o protocolo continua a basear-se em UDP e anúncios periódicos *best effort* sem memorização dos anúncios recebidos.

O RIP tem a vantagem adicional de poder ser usado praticamente sem recurso a parametrização nenhuma dos *routers*, o que é mais difícil com outros protocolos. No entanto, as suas características confinam-no a redes simples, de pequena dimensão. Em redes de maior dimensão, geralmente usam-se outros protocolos.

Existem outros protocolos baseados em vectores de anúncios (de visibilidade ou de caminhos) em que o problema da contagem para o infinito foi adequadamente resolvida, como por exemplo os protocolos: EIGRP (Enhanced Interior Gateway Routing Protocol), um protocolo proprietário de um fabricante de comutadores mas transformado em aberto recentemente, ver o RFC 7868 de 2016, e o BGP (Border Gateway Protocol) que será discutido no Capítulo 17.

Observações finais

De um ponto de vista meramente abstracto, *i.e.*, independentemente das opções de implementação concretas, é difícil comparar o encaminhamento baseado em estado dos canais, com o encaminhamento baseado em vectores de distâncias ou caminhos, porque as qualidades finais estão muito dependentes quer das opções concretas de implementação, quer de mecanismos complementares que são importantes em redes reais.

Em redes de pequena dimensão e sem constrangimentos significativos de convergência, não existem razões fortes contra a utilização de RIP. Em redes médias e com constrangimentos de convergência são geralmente utilizados os protocolos baseados em estado dos canais, como o OSPF ou o IS-IS. Em redes muito grandes, geralmente utilizam-se os dois tipos de protocolos mas com base em implementações sofisticadas de protocolos baseados em anúncios de caminhos, nomeadamente o protocolo BGP.

16.5 Resumo e referências

Resumo

O encaminhamento de pacotes é um dos problemas centrais das redes de computadores e também um dos mais complexos. Envolve várias facetas e requisitos: teóricas, para modelização e optimização da rede; algorítmicas, para concretizar essas soluções; de engenharia, para implementar um sistema complexo que responda aos requisitos e minimizar o seu custo; de monitorização, para obter dados sobre o seu funcionamento real. Adicionalmente problemas operacionais e de planeamento, para responder às necessidades do dia a dia e adaptar a rede à continua evolução dos requisitos que lhe são colocados.

Uma primeira solução de compromisso relativamente comum consiste em usar encaminhamento pelo caminho mais curto. Esta solução é aceitável quando os caminhos mais curtos estão bem dimensionados para o tráfego existente. O algoritmo de Dijkstra é um algoritmo centralizado que, dado um grafo com arcos pesados e um nó origem, calcula uma árvore de cobertura de caminhos mais curtos com raiz no nó origem.

Este algoritmo, aplicado repetidamente a cada um dos nós de comutação de pacotes de uma rede, permite determinar caminhos mais curtos de cada nó para todos os outros, ou seja permite determinar tantas árvores de cobertura quantos os nós do grafo. Seja qual for o nó que toma uma decisão de encaminhamento de um pacote, encaminhá-lo-á para o destino por um caminho mais curto.

Para o encaminhamento pelo caminho mais curto foram introduzidas diversas soluções concretas, entre as quais se destacam duas: o encaminhamento com base no **estado dos canais** (*link-state routing*), e a conhecida como **vectores de distâncias** (*distance vectors*) ou **anúncios de visibilidade** (*reachability announcements*).

A técnica de encaminhamento pelo caminho mais curto com base no estado dos canais (*link-state routing*) baseia-se na replicação por todos os comutadores, por difusão fiável, de todas as adjacências existentes na rede. Desta forma, cada comutador adquire a visão do grafo que caracteriza a rede e, usando o algoritmo de Dijkstra, pode calcular os melhores caminhos para todos os destinos existentes e assim construir a sua FIB.

Sempre que têm lugar modificações do estado de qualquer componente da rede, os comutadores que as detectam difundem actualizações, também designadas SLAs (*Link State Announcements* ou *Adjacency Announcements*) e, logo que a difusão fiável se completar, as actualizações são repercutidas nas FIBs de todos os comutadores e a rede converge e volta a um estado estável.

O encaminhamento por vectores de distâncias, ou usando o algoritmo de Bellman-Ford, consiste na troca entre comutadores vizinhos de vectores de distâncias ou anúncios de visibilidade. Estas mensagens indicam aos vizinhos, os destinos e as distâncias que cada comutador conhece. Ao receber anúncios dos vizinhos, cada comutador pode escolher entre as diversas alternativas aquelas que correspondem aos caminhos mais curtos.

O algoritmo é fácil de perceber e simples de incorporar num protocolo de encaminhamento, apenas obriga à existência de uma FIB que inclua a noção de distância pelo que a informação local obrigatória é mínima. O processamento de cada mensagem é simples e linear com a dimensão dos anúncios, os quais são proporcionais ao número de destinos diferentes existentes na rede.

Com a experiência foi-se constatando que o fenómeno designado “**contagem para o infinito**” ou “**as boas notícias correm velozes, mas as más devagar**” (“*good news travel fast while bad news travel slowly*”) introduz problemas delicados de convergência no algoritmo de Bellman-Ford. Este fenómeno pode ser minorado introduzindo diversos mecanismos, nomeadamente: limitar o número de *hops* máximo dos caminhos, usar anúncios do tipo *split horizon with poison reverse*, e introduzir *hold-down-timers*. Estes mecanismos minoram o problema sem o resolver definitivamente.

Soluções mais completas anulam a simplicidade do algoritmo. Assim, na prática, sempre que as redes têm alguma complexidade e a sua convergência é mais crítica, os protocolos baseados no estado dos canais são os preferidos em redes de média dimensão.

Em redes de pequena dimensão e sem constrangimentos significativos de convergência, não existem razões fortes contra a utilização de RIP, um protocolo simples baseado no algoritmo de Bellman-Ford. Em redes médias e com constrangimentos de convergência são geralmente utilizados os protocolos baseados em estado dos canais, como o OSPF ou o IS-IS. Em redes muito grandes, geralmente utilizam-se os dois tipos de protocolos mas com base em implementações sofisticadas de protocolos baseados em anúncios de caminhos, como o protocolo BGP, ver o Capítulo 17.

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

Principais conceitos

Custo de um caminho (*path cost*) Define-se custo de um caminho como sendo a soma dos custos dos diferentes canais atravessados por esse caminho.

Distância (*distance*) Define-se distância entre dois nós como sendo o custo do um caminho de menor custo entre esses nós. A distância é relativa a um conjunto de caminhos conhecidos.

Caminho mais curto (*shortest path*) Define-se caminho mais curto entre dois nós como sendo um caminho mais curto entre todos os caminhos que os ligam.

Algoritmo de Dijkstra (*Dijkstra algorithm*) é um algoritmo centralizado que, dado um grafo com arcos pesados e um nó origem, calcula uma árvore de cobertura de caminhos mais curtos com raiz no nó origem.

Comutador inteligente de pacotes (*packet router*) É um comutador de pacotes que executa um protocolo de encaminhamento, *i.e.*, um algoritmo distribuído de troca de informação de encaminhamento entre os comutadores, que permite aos mesmos adaptarem-se dinamicamente ao estado da rede e realizarem o encaminhamento dos pacotes em função de uma estratégia de optimização como por exemplo encaminhar pelo caminho mais curto.

Métrica de um protocolo (*routing protocol metric*) Numa rede real cada canal tem um custo. A métrica de um protocolo é a forma concreta que este usa para determinar o custo dos canais. Uma métrica simples é aquela que associa um custo constante a cada canal, por exemplo 1. As métricas mais sofisticadas tentam associar a um canal um custo função do tempo de trânsito dos pacotes que o atravessam. Neste caso uma métrica pode ser inversamente proporcional ao débito, proporcional ao tempo de propagação, proporcional à intensidade do tráfego ou ainda proporcional ao custo monetário do canal.

Encaminhamento pelo caminho mais curto (*shortest path routing*) Forma de encaminhamento de pacotes numa rede que consiste em escolher sempre caminhos mais curtos para os pacotes. Esta solução é aceitável quando os caminhos mais curtos estão bem dimensionados para o tráfego existente na rede.

FIB (*Forwarding Information Base*) Estrutura de dados que contém a informação necessária para um comutador de pacotes escolher qual a interface pela qual tem de transmitir um pacote a caminho do seu destino. Geralmente, a FIB implementa um simples mapeamento de endereços de destino em interfaces e distâncias e é independente dos protocolos de encaminhamento usados.

RIB (*Routing Information Base*) Estrutura de dados de um comutador associada a um protocolo de encaminhamento específico, que contém a informação, necessária para a computação de uma FIB, obtida por esse protocolo.

Encaminhamento com base no estado dos canais (*link state routing*) Estratégia de encaminhamento que consiste em os comutadores trocarem informação, através de um protocolo de difusão fiável, que lhes permite adquirirem todas as adjacências de cada comutador. Desta forma, cada comutador adquire uma visão completa do estado global da rede. Esse estado é idêntico em todos os comutadores e permite a cada um deles, usando por exemplo o algoritmo de Dijkstra, computar a sua FIB.

LSA (*Link State Announcement*) São as mensagens com a informação de adjacência transmitidas por cada comutador, que são replicadas por difusão fiável e que servem de base aos protocolos de encaminhamento com base no estado dos canais. Os LSAs podem ser de vários tipos e servirem para transportar vários tipos de informações entre os comutadores da rede.

LSDB (*Link State Data Base*) Estrutura de dados de um comutador que contém o conjunto dos LSAs recebidos por cada comutador a participar em encaminhamento com base no estado dos canais.

Encaminhamento com vectores de distâncias (*distance vector routing*) Estratégia de encaminhamento que consiste em pressupor que cada comutador conhece

uma forma de encaminhar pacotes dirigidos a alguns destinos e que divulga a mesma aos seus vizinhos directos através de vectores ou anúncios de distâncias. Estes anúncios consistem num conjunto de pares de (destino, distância), que permitem ao receptor, sabendo a que distância está o vizinho emissor do anúncio, saber se o deve seleccionar para encaminhar pacotes para aquele destino por um caminho mais curto.

Algoritmo de Bellman-Ford Algoritmo que está na base do encaminhamento com vectores de distâncias, designado com base no nome dos seus inventores.

Anúncios de visibilidade (*reachability announcements*) É outra designação dada aos vectores de distâncias na forma de encaminhamento designada como encaminhamento com base em vectores de distâncias.

Convergência (*convergence*) É o tempo necessário para, na sequência de uma qualquer alteração de estado dos nós ou dos canais de uma rede, todos os nós de comutação tomarem em consideração essas alterações, reconfigurarem a rede, e voltarem a colocá-la em funcionamento de acordo com a nova configuração. Os protocolos com base no estado dos canais convergem rapidamente. Os protocolos com base em vectores de distância convergem também rapidamente se a alteração de estado consistir numa diminuição de distância. No entanto, quando se dá um aumento da distância, em particular quando um destino deixa de ser acessível, pode produzir-se uma situação anómala em que alguns comutadores introduzem na rede uma informação que já não é válida, mas que outros tomam como certa. Em casos particulares, isso pode dar uma situação oscilatória e transitória em que se formam ciclos de encaminhamento e que é designada por contagem para o infinito (*counting to infinity*).

Split horizon with poison reverse Num protocolo de encaminhamento com base em vectores de distâncias, os comutadores anunciam aos seus vizinhos um destino à distância infinito caso usem esse vizinho para atingir o destino. Esta técnica permite minorar o aparecimento da anomalia contagem para o infinito. No entanto, não a resolve definitivamente.

Hold-down-timers Num protocolo de encaminhamento com base em vectores de distâncias, os comutadores colocam em quarentena os destinos que passaram a estar à distância infinito e não aceitam anúncios sobre esses destinos durante o período de tempo *hold-down timer*. O objectivo é permitir a todos os comutadores receberem o anúncio com as “máis notícias” e não introduzirem, por inadvertência, um anúncio falso sobre a possibilidade de enviarem pacotes para o destino que de facto se tornou inacessível. Este mecanismo resolve a anomalia contagem para o infinito à custa de cortar o acesso durante o período de tempo *hold-down*.

RIP (*Routing Information Protocol*) Um dos primeiros protocolos de encaminhamento introduzidos foi o protocolo RIP. Trata-se de uma protocolo do tipo encaminhamento com base em vectores de distâncias que implementa a forma mais simples de realizar esta estratégia de encaminhamento. Tem como vantagem a simplicidade mas só é realista em redes de pequena dimensão e onde se tolerem tempos de convergência elevados.

IS-IS (*Intermediate System - Intermediate System*) Primeiro protocolo de encaminhamento que usou o método de encaminhamento designado como encaminhamento com base no estado dos canais. Pode ser usado em inúmeros contextos e diferentes redes pois dispõe de um sistema extensível de LSAs que pode adaptar-se à difusão de vários tipos de informações em diversos tipos de redes.

OSPF (*Open Shortest Path First*) Protocolo de encaminhamento inspirado do IS-IS mas especializado no encaminhamento em redes TCP/IP.

EIGRP (*Enhanced Interior Gateway Routing Protocol*) Protocolo de encaminhamento com base em vectores de distâncias que inclui métodos sofisticados de evitar a anomalia contagem para o infinito, assim como suporta várias métricas de encaminhamento. Trata-se de um protocolo de encaminhamento proprietário de um fabricante de comutadores que só recentemente foi tornado público.

BGP (*Border Gateway Protocol*) Protocolo de encaminhamento com base em vectores de caminhos que transmite anúncios de visibilidade que incluem o caminho completo para chegar ao destino. Por essa razão, previnem a introdução de ciclos de encaminhamento e anulam a contagem para o infinito. Para além disso, os anúncios são memorizados pelos comutadores e transmitidos de forma fiável sobre TCP, pelo que só são transmitidas as alterações quando estas têm lugar. Trata-se de um protocolo complexo que só é usado em redes de grande dimensão de operadores ou em redes de acesso ligadas a diferentes operadores.

Referências

Os livros [Bellman, 1957; Ford and Fulkerson, 1962] são hoje em dia dois clássicos e ambos apresentam o algoritmo Bellman-Ford. O primeiro numa versão centralizada e o segundo numa versão distribuída. O artigo [Dijkstra, 1959] apresenta o algoritmo de Dijkstra para cálculo de caminhos mais curtos.

O livro [Huitema, 1995] é uma referência clássica que apresenta uma introdução aos protocolos RIP e OSPF e discute as suas bases algorítmicas. Todos os livros bem conhecidos de introdução às redes de computadores cobrem os algoritmos cobertos neste capítulo, nomeadamente [Tanenbaum and Wetherall, 2011; Peterson and Davies, 2012; Kurose and Ross, 2013]. O problema da convergência do encaminhamento com base no estado dos caminhos em redes complexas é analisado em detalhe em [Francois et al., 2005].

Existem inúmeros livros com descrições dos protocolos citados neste capítulo. O livro [Doyle and Carroll, 2006] é considerado um dos melhores, que inclui também indicações concretas sobre como os parametrizar em comutadores de pacotes populares.

Apontadores para informação na Web

- <http://en.wikipedia.org> – Os artigos sobre RIP, OSPF, IS-IS e BGP disponíveis na wikipedia na versão inglesa constituem boas introduções, relativamente sintéticas, a estes protocolos. Têm a vantagem adicional de listar todos os RFCs relevantes.
- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo: RFC 1058 (RIPv1), RFC 2080 (RIPng - IPv6), RFC 2328 (OSPF IPv4), RFC 2453 (RIPv2), RFC 5340 (OSPF IPv6) e RFC 7868 (EIGRP).

16.6 Questões para revisão e estudo

1. Defina encaminhamento estático e encaminhamento dinâmico.
2. Indique que factores se devem tomar em consideração para conceber um protocolo ideal de encaminhamento dinâmico.
3. Defina encaminhamento pelo melhor caminho.
4. Diga se é verdade ou mentira ou comente:
 - (a) Um protocolo de encaminhamento do tipo vector de distâncias pode tratar os canais ponto-a-ponto e os canais multi-ponto exactamente da mesma forma.

- (b) Um protocolo de encaminhamento do tipo estado dos canais trata os canais ponto-a-ponto e os multi-ponto exactamente da mesma forma.
 - (c) Um protocolo de encaminhamento do tipo estado dos canais trata a reparação de uma partição de uma rede como um caso normal de anúncio do estado do canal que acabou de ser reparado.
 - (d) Um protocolo de encaminhamento do tipo estado dos canais suporta bem redes de grande dimensão, com muitos canais e com muitos destinos possíveis.
 - (e) Um protocolo de encaminhamento do tipo estado dos canais converge rapidamente quando há canais que se avariam.
 - (f) Um protocolo de encaminhamento do tipo estado dos canais converge rapidamente quando há canais avariados que são reparados.
 - (g) Um protocolo de encaminhamento do tipo estado dos canais assegura que um anúncio de um novo destino é propagado a todos os comutadores num período de tempo proporcional ao número de comutadores da rede.
 - (h) Um protocolo de encaminhamento do tipo estado dos canais assegura que um anúncio de um novo destino é propagado a todos os comutadores num período de tempo proporcional ao número de canais da rede.
 - (i) Um protocolo de encaminhamento do tipo estado dos canais assegura que um anúncio de um novo destino é propagado a todos os comutadores num período de tempo proporcional ao diâmetro da rede.
 - (j) Se vários canais ficarem inoperacionais, a rede pode decompor-se em várias partições. Quando uma partição de rede se repara, pode surgir o problema contagem para o infinito associado a um protocolo de encaminhamento de tipo vector de distâncias.
 - (k) Um protocolo de encaminhamento do tipo vector de distâncias suporta bem redes de grande dimensão e com muitos destinos possíveis.
 - (l) Um protocolo de encaminhamento do tipo vector de distâncias converge rapidamente quando há canais que se avariam.
 - (m) Um protocolo de encaminhamento do tipo vector de distâncias converge rapidamente quando há canais avariados que são reparados.
 - (n) Um protocolo de encaminhamento do tipo vector de distâncias assegura que um anúncio de um novo destino é propagado a todos os comutadores num período de tempo proporcional ao número de comutadores da rede.
 - (o) Um protocolo de encaminhamento do tipo vector de distâncias assegura que um anúncio de um novo destino é propagado a todos os comutadores num período de tempo proporcional ao número de canais da rede.
 - (p) Um protocolo de encaminhamento do tipo vector de distâncias assegura que um anúncio de um novo destino é propagado a todos os comutadores num período de tempo proporcional ao diâmetro da rede.
 - (q) Se vários canais ficarem inoperacionais, a rede pode decompor-se em várias partições. Quando uma partição de rede se repara, pode surgir o problema contagem para o infinito associado a um protocolo de encaminhamento de tipo vector de distâncias.
5. Diga se são verdade ou mentira as seguintes afirmações produzidas no contexto de uma rede que está a usar um protocolo de encaminhamento do tipo estado dos canais. Justifique a sua resposta.
- (a) Se juntarmos à rede mais nós, o tempo de execução do algoritmo de Dijkstra aumentará apenas nos nós vizinhos dos novos nós.

- (b) Cada comutador inunda a rede para enviar para todos os outros a topologia de toda a rede que conhece.
- (c) Cada comutador inunda a rede para enviar para todos os outros o estado dos canais a que está ligado directamente.
- (d) Cada comutador calcula árvores de menores custos entre si e todos os outros comutadores.
- (e) O algoritmo de Dijkstra pode introduzir o problema designado contagem para o infinito.
- (f) Quando há uma alteração do estado de um canal, o comutador que lhe está ligado espera 120 segundos antes de decidir enviar um anúncio de mudança de estado pois desta forma agrupa anúncios, poupa mensagens e não inunda a rede com alarmes inúteis.
6. O encaminhamento diz-se simétrico se os canais atravessados de A para B são os mesmos, mas por ordem inversa, que os canais usados de B para A. Suponha que numa dada rede só existe um caminho mais curto entre cada dois comutadores. Diga se é verdade ou mentira:
- (a) Nessa rede, caso a métrica do protocolo de encaminhamento seja o número de *hops*, então todo o encaminhamento é simétrico.
- (b) Nessa rede o encaminhamento é simétrico independentemente da métrica do protocolo de encaminhamento.
- (c) Nessa rede, caso a métrica do protocolo de encaminhamento seja o inverso do débito dos canais, existe uma situação particular em que o encaminhamento é simétrico.
7. Considere uma rede baseada em canais ponto-a-ponto que interligam vários comutadores. Imagine que um dos canais tem um problema que faz com que esteja constantemente a transitar do estado operacional (capaz de transmitir dados) para o estado inoperacional (sem poder transmitir dados) e vice-versa, a um ritmo relativamente rápido (*e.g.*, a cada 5 segundos).
- (a) De que forma uma transição de estado do canal é tomada em consideração por um protocolo de encaminhamento do tipo estado dos canais?
- (b) De que forma uma transição de estado do canal é tomada em consideração por um protocolo de encaminhamento do tipo vector de distâncias?
- (c) Imagine uma forma de minorar os custos induzidos por esta instabilidade em ambos os protocolos.
8. Um ciclo no encaminhamento (*routing loop*) ocorre, por exemplo, quando um comutador A ligado directamente a um comutador B julga que o melhor caminho para atingir o destino D é via B e B julga que o melhor caminho para atingir o mesmo destino é via A.
- (a) Quais os inconvenientes de um ciclo no encaminhamento?
- (b) A situação indicada acima é a única em que um ciclo no encaminhamento pode ocorrer?
- (c) Que mecanismo existe por omissão nos cabeçalhos dos pacotes para evitar os inconvenientes desta anomalia quando ela ocorre?
- (d) Em que situação pode ocorrer um ciclo no encaminhamento quando se usa um protocolo de encaminhamento do tipo estado dos canais?

- (e) Em que situação pode ocorrer um ciclo no encaminhamento quando se usa um protocolo de encaminhamento do tipo vector de distâncias?
9. Considere a rede da Figura 16.18 e indique uma sequência de eventos que pode levar ao aparecimento de uma anomalia do tipo contagem para o infinito se o canal que liga os nós **x** e **a** se avariar.
10. Existe uma situação particular em que dois comutadores a usarem um protocolo baseado no estado dos canais têm de sincronizar as respectivas LSA Data Bases. Qual é esse caso?
11. Considere a rede da Figura 16.19. O encaminhamento é do tipo vector de distâncias com anúncios periódicos.

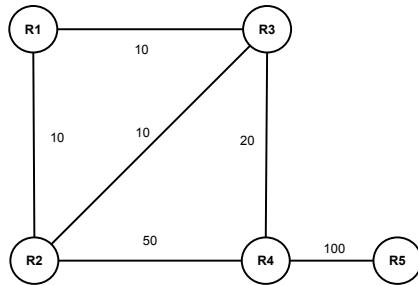


Figura 16.19: Rede do exercício 11

- (a) Indique as FIBs (tabelas de encaminhamento) dos comutadores R2, R3, R4 e R5.
- (b) A rede já convergiu e todos os canais estão operacionais. Descreva os anúncios que R2, R3, R4 e R5 enviam periodicamente sabendo que o protocolo não usa *split horizon with poison reverse*.
- (c) A rede já convergiu e todos os canais estão operacionais. Descreva os anúncios que R4 e R5 enviam periodicamente sabendo que o protocolo usa agora *split horizon with poison reverse*.
- (d) A rede não usa *hold down timers*. Que anúncios fará R4 quando o canal para R5 falhar se os comutadores estiverem a usar *split horizon with poison reverse*.
- (e) A rede não usa *hold down timers*. Descreva uma situação em que pode surgir uma anomalia de contagem para o infinito quando o canal para R5 falhar e os comutadores estiverem a usar *split horizon with poison reverse*.
- (f) Se a rede usar *hold down timers* que acontece quando o canal de R4 para R5 falhar?
12. Considere a rede da Figura 16.20. O encaminhamento é do tipo vector de distâncias com anúncios periódicos. O custo de um canal com a capacidade de 10 Mbps é 100, e o de um com a capacidade de 100 Mbps é 10. O tempo de propagação de extremo a extremo de cada canal é de 100 ms.
- (a) O protocolo já convergiu. Qual o custo de enviar pacotes de R1 para R5 do ponto de vista do algoritmo de encaminhamento?
- (b) Como é o vector de distâncias que contém um anúncio de R4 dirigido a R5 quando todos os canais estão operacionais ?

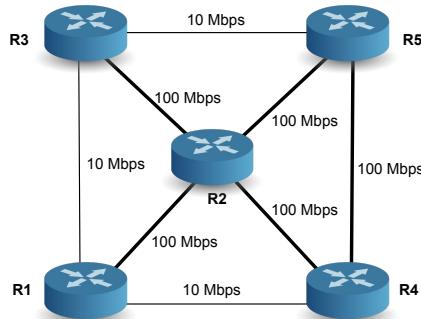


Figura 16.20: Rede do exercício 12

- (c) Admitindo que os anúncios são transmitidos instantaneamente sem demora ou tempo de transmissão relevante, e que o tempo de processamento dos mesmos é desprezável, qual o tempo máximo que levam todos os routers a ficar a conhecer o novo encaminhamento quando o canal que liga R5 a R4 for abaixo?
- (d) Idem quando o canal avariado for o o canal que liga R1 a R4.
13. Considere a rede da Figura 16.21 que interliga os comutadores indicados com os custos assinalados nos canais. A rede usa um protocolo de encaminhamento baseado no algoritmo Bellman-Ford com anúncios periódicos e sem memorização dos anúncios.

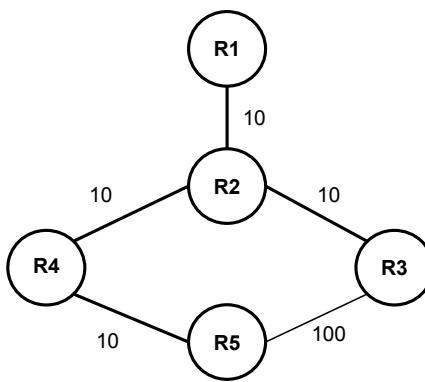


Figura 16.21: Rede do exercício 13

- (a) A rede já convergiu e todos os canais estão operacionais. Descreva os anúncios que R3 e R5 enviam aos seus vizinhos sabendo que o protocolo não usa *split horizon with poison reverse*.
- (b) A rede já convergiu e todos os canais estão operacionais. Descreva os anúncios que R3 e R5 enviam aos seus vizinhos sabendo que o protocolo usa *split horizon with poison reverse*.
- (c) A rede já convergiu mas mais tarde o canal entre R1 e R2 avaria e R2 faz uma anúncio de que a distância a R1 passou infinito. Sabendo que

- o protocolo não usa *split horizon with poison reverse*, é possível entrar-se num ciclo de contagem para o infinito nesta rede? Justifique.
- (d) A rede já convergiu mas mais tarde o canal entre R1 e R2 avaria e R2 faz um anúncio de que a distância a R1 passou infinito. Sabendo que o protocolo usa *split horizon with poison reverse*, é possível entrar-se num ciclo de contagem para o infinito nesta rede? Justifique.
- (e) Se a rede usar *hold down timers* que acontece quando o canal de R1 para R2 falha?
- (f) Se a rede usasse um protocolo do tipo estado dos canais seria possível acontecer a anomalia contagem para o infinito? Seria necessário usar *hold down timers*?
14. Considere uma rede baseada num protocolo de encaminhamento do tipo vector de distâncias.
- (a) Como é que um comutador pode tentar atrair para si todos os pacotes da rede?
- (b) De que forma um comutador pode tentar obrigar a rede a parar de encaminhar pacotes para os diferentes destinos?
- (c) Quais as implicações dos ataques anteriores do ponto de vista da segurança?
15. Considere uma rede baseada num protocolo de encaminhamento do tipo estado dos canais.
- (a) Como é que um comutador pode tentar atrair para si todos os pacotes da rede?
- (b) De que forma um comutador pode tentar obrigar a rede a parar de encaminhar pacotes para os diferentes destinos?
- (c) Quais as implicações dos ataques anteriores do ponto de vista da segurança?
16. Considere um protocolo de encaminhamento que usa uma métrica exclusivamente baseada no tempo de trânsito. No momento de tomar a decisão de encaminhamento, o comutador considera o tempo de trânsito assinalado na FIB mas também o tamanho da fila de espera dos canais pelos quais pode transmitir os pacotes.
- (a) Descreva o processo de tomada de decisão de encaminhamento pelo comutador.
- (b) Numa dada rede dois comutadores estão ligados directamente por dois canais paralelos com o mesmo tempo de trânsito. Como realiza o comutador o encaminhamento quando o *next hop* é esse vizinho e as filas de espera envolvidas têm comprimentos distintos?
- (c) Questão idêntica à anterior mas as filas de espera envolvidas estão sempre vazias?
- (d) Que efeito tem esta forma de encaminhamento sobre a ordem com que os pacotes são entregues ao destino? Acha isso bom ou mau?
- (e) Existem outras razões para não se adoptar esta métrica?

17. Numa rede de computadores, organizada como uma malha com ciclos, existem vários caminhos possíveis de cada comutador até cada destino. Se a rede tiver muitos ciclos e um grau elevado de redundância, existem muitos desses caminhos e alguns terão custos idênticos. Um protocolo de encaminhamento capaz de utilizar simultaneamente vários caminhos para o mesmo destino com o mesmo custo, diz-se um protocolo de encaminhamento multi-caminho com custo igual (*equal cost multi-path*).
- Indique uma vantagem de no caso particular em que existem vários caminhos com o mesmo custo explorar alternativamente esses vários caminhos.
 - Algumas métricas consideram que cada salto (*hop*) entre dois comutadores tem sempre custo 1 seja qual for a capacidade da ligação e não reflectem de forma proporcional o tempo de trânsito de um caminho. Por exemplo, um caminho entre os computadores A e B tem tempo de trânsito de extremo a extremo de 10 milissegundos e outro, entre os mesmos computadores, tem tempo de trânsito de 100 milissegundos. No entanto, se ambos os caminhos tiverem o mesmo número de *hops*, a função de custo indicará que têm o mesmo custo e poderiam ser usados para encaminhamento multi-caminho. Qual o resultado dessa utilização?
 - Suponha que uma aplicação multimédia interactiva sobre UDP entre os computadores A e B está a utilizar uma rede com as características indicadas na alínea anterior e que os pacotes entre A e B estão a seguir caminhos alternativos: o 1º pacote vai por um caminho, o 2º por outro, o 3º pelo caminho do 1º, o 4º pelo caminho do 2º, etc. Que parâmetro dessa aplicação teria de ser regulado para que não houvesse uma repercussão demasiado negativa do encaminhamento multi-caminho neste cenário?
18. No protocolo OSPF, os LSAs são passados aos vizinhos de forma fiável (com um protocolo do tipo *stop & wait*) que implica que as relações entre vizinhos sejam sempre ponto-a-ponto, mesmo quando o canal que os liga é multi-ponto. No protocolo RIP, os anúncios são passados aos vizinhos usando mensagens multi-ponto quando o canal é multi-ponto. O protocolo RIP introduz fiabilidade na transmissão dos anúncios repetindo-os periodicamente. Analise se seria preferível o OSPF adoptar uma solução semelhante ao RIP, e o RIP adoptar uma solução semelhante ao OSPF, sempre que os vizinhos estão ligados por canais em difusão.
19. É possível modificar o algoritmo Bellman-Ford para passar a incluir nos anúncios o caminho usado para o destino (os vectores de distâncias passam a chamar-se vectores de caminhos). Por exemplo, o anúncio inclui destinos, distâncias e os caminhos (uma lista de identificadores de comutadores) usados para os alcançar. Inicialmente, um comutador apenas consegue anunciar os caminhos para os destinos a que dá acesso directamente. Mas à medida que vai recebendo anúncios, não só enriquece a sua RIB com mais destinos e caminhos, como os envia aos seus vizinhos. Existem na rede n destinos, c comutadores e k canais. O diâmetro máximo também é conhecido e é d_{max} .
- Indique uma estrutura de dados adequada para suportar a RIB.
 - Indique uma estrutura de dados adequada para suportar um anúncio.
 - Indique a complexidade espacial da RIB e compare-a com a complexidade espacial da FIB do algoritmo de Bellman-Ford.
 - Quais as vantagens e defeitos desta variante do algoritmo de encaminhamento?

20. Um ponto de partida para conceber um algoritmo de encaminhamento para uma rede com ciclos, que só tem canais ponto-a-ponto, poderia ser o seguinte: cada vez que um comutador recebe um pacote, encontra no seu cabeçalho o endereço do emissor inicial, assim como o número de *hops* percorridos desde a origem e sabe qual a interface pela qual o pacote chegou. Com estas informações o comutador pode ir enriquecendo a sua FIB.
- (a) Indique como se poderia chamar essa técnica por analogia com outro protocolo de encaminhamento.
 - (b) Como actualiza a FIB um comutador quando recebe um pacote?
 - (c) O comutador acabou de receber um pacote dirigido a um destino para o qual ainda não existe nenhuma entrada na FIB. Como deve processar o pacote? Que requisitos é necessário garantir para que a forma de proceder indicada seja realista?
 - (d) Esta técnica pressupõe ou não que a rede tem simetria, *i.e.*, que o melhor caminho de A para B é simétrico do melhor caminho de B para A?
 - (e) Admita que a carga ou a configuração da rede vai variando. As soluções indicadas nos parágrafos anteriores ainda são adequadas? Como se poderia o algoritmo adaptar a essas alterações?
21. No quadro do encaminhamento baseado no estado dos canais, um comutador recebeu de um vizinho um LSA que o vizinho está a difundir por inundação. Essa mensagem, da classe LSA, pode ser manipulada pelos seguintes métodos:
- `int getRouter()` – devolve o identificador do comutador que gerou originalmente este LSA
 - `int getSeq()` – devolve o número de sequência do LSA
 - `void flood()` – faz o *flooding* do LSA para os outros comutadores
 - `void sendACK()` – envia um ACK ao vizinho que enviou o LSA
 - `void sendNACK(LSA newerLSA)` – envia um NACK ao vizinho que enviou o LSA com a versão mais recente do mesmo LSA.

A tabela `lsadb`, da classe `LSADatabase`, contém em cada comutador o estado da rede *i.e.*, o conjunto dos LSAs mais recentes recebidos. Esta tabela pode ser manipulada pelos seguintes métodos:

- `LSA getLSA(int c)` – devolve o mais recente `lsa` gerado pelo comutador `c` ou `null` se não existe nenhum
- `void putLSA(LSA lsa)` – insere um `lsa` na tabela e assegurando que existe apenas o `lsa` mais recente associado a cada comutador.

Escreva o código do método `processReceivedLSA` executado pelos comutadores quando recebem um LSA de um vizinho para assegurarem a difusão fiável dos LSAs por toda a rede.

```
void processReceivedLSA ( LSA lsa, LSADatabase lsadb ) { ... }
```

22. Numa rede, os comutadores têm vários vizinhos, a cada um dos quais estão ligados directamente por um canal distinto. Por hipótese, todos os destinos existentes na rede e todos os comutadores são identificados por uma letra: A,

B, C, ..., Z. As interfaces que ligam um comutador aos vizinhos têm por identificador o identificador do vizinho.

Um comutador envia um pacote `p` para o vizinho `X` usando a função `sendTo` (`p, X`). Um comutador tem acesso ao seu identificador através do método `myID()`. Um pacote `p` tem os seguintes campos: `p.source`, `p.destination`, `p.ttl` e `p.data` com os significados óbvios.

Um comutador tem uma FIB `fib` que suporta vários métodos, entre os quais:

- `fib.getGateway (destination)` - devolve o vizinho que encaminha para `destination` ou `null` se `fib` desconhece `destination`
- `fib.getDistance (destination)` - devolve o custo do caminho deste comutador até `destination` ou `infinity` se `fib` desconhece `destination`
- `fib.setFIBEntry (destination, distance, gateway)` - actualiza a entrada de `fib` referente a `destination`.

- (a) Escreva o pseudo código usado pelo comutador para processar um pacote `p` que acabou de receber. Se o comutador não sabe ou não pode processar o pacote, deve ignorá-lo.
 - (b) Escreva o pseudo código usado pelo comutador para processar um pacote `p` que acabou de receber do vizinho `V`. Se o comutador não conseguir processar o pacote ignora-o, mas gera um pacote `error` dirigido à origem do pacote `p`. Como deveria este pacote de erro ser processado por `V`?
 - (c) Na rede usa-se o algoritmo Bellman-Ford para actualizar as tabelas de encaminhamento. O comutador usa a versão *split horizon with poison reverse* do algoritmo. Escreva o pseudo código usado pelo comutador para gerar o anúncio que vai enviar ao vizinho `V`.
 - (d) Na rede usa-se o algoritmo Bellman-Ford para actualizar as tabelas de encaminhamento. O comutador usa a versão *split horizon with poison reverse* do algoritmo. Escreva o pseudo código usado pelo comutador para processar um anúncio que acabou de receber do vizinho `V` à distância `Vcost`.
23. Considere uma rede com n comutadores e c canais ponto-a-ponto. Existem duas opções para protocolo de encaminhamento para essa rede: RIP ou OSPF. Utilize nas questões a seguir a notação $O(n)$ para indicar que uma grandeza é proporcional a n .
- (a) Compare em cada uma das opções a memória ocupada em cada comutador.
 - (b) Compare em cada uma das opções a carga da CPU em cada comutador sempre que se liga um novo comutador e quando um canal se avaria.
 - (c) Compare em cada uma das opções a carga da rede em número de mensagens quando a rede está estável.
 - (d) Compare em cada uma das opções a carga da rede em número de mensagens sempre que se liga um novo comutador e quando um canal se avaria.

Capítulo 17

Interligação de redes - protocolo IP

An expert is a man who made all the mistakes, which can be made, in a very narrow field.

– Autor: Niels Bohr

Nos capítulos anteriores vimos como funcionam internamente algumas redes particulares, nomeadamente redes baseadas em canais de difusão, redes que usam protocolos de encaminhamento baseados em inundação e algoritmos e protocolos para encaminhamento pelo melhor caminho. Nenhuma dessas soluções particulares tomada isoladamente é adequada à Internet como um todo. Com efeito, a Internet é formada pela interligação de um grande conjunto de redes particulares: redes domésticas, redes institucionais, redes de acesso e vários tipos de redes de trânsito, ver a Secção 1.3 e ver a Secção 4.4.

Os bons princípios de desenho de um sistema complexo, ver a Secção 4.1, recomendam a sub-divisão do mesmo em partes separadas e isoladas, e a definição de interfaces entre as mesmas que permitam a sua independência. Do ponto de vista do encaminhamento, a Internet é sub-dividida em diferentes redes, que formam uma partição da mesma, *i.e.*, essas diferentes redes são independentes e sem sobreposições. No entanto, essas redes estão interligadas através de múltiplas ligações, ver a Secção 4.4. O termo técnico usado para um conjunto de redes interligadas é um *internetwork*.

A independência das diferentes partes interligadas de um sistema depende em larga medida da correcta definição de interfaces. Na Internet, do ponto de vista do encaminhamento, as principais interfaces são o protocolo IP, que define o endereçamento, o formato do cabeçalho dos pacotes e o significado e processamento dos seus diferentes campos, assim como os protocolos de encaminhamento entre as diferentes redes. Adicionalmente, os bons princípios recomendam a independência das partes, o que implica que os protocolos de encaminhamento entre as diferentes redes deve deixar liberdade às mesmas para esconderem as soluções de encaminhamento que usam internamente¹.

Neste capítulo vamos ver como são definidas e funcionam as interfaces entre as redes que formam a Internet. Começaremos por analisar como é definido o sistema de endereçamento da Internet. Uma faceta fundamental que abordaremos é a interligação entre endereçamento e encaminhamento e como o endereçamento condiciona as

¹ Este princípio também recomendaria que o endereçamento e as interfaces usadas internamente pelas partes também fosse escondida do exterior, ver [Day, 2008] por exemplo. No entanto, esta faceta foi ignorada na definição do protocolo IP.

formas de encaminhamento usadas. Esta relação é delicada e merece uma explicação relativamente longa.

A seguir vamos estudar o protocolo IP. Este protocolo é a principal interface entre os computadores e as redes e entre as diferentes redes interligadas. No entanto, em muitas redes, o protocolo IP é também a principal interface entre as diferentes componentes internas das mesmas. Veremos então que o conceito de rede é na verdade recursivo.

Para que uma rede baseada no protocolo IP funcione, são necessários um conjunto de protocolos auxiliares. A sua apresentação será feita mais adiante.

Existem actualmente duas versões do protocolo IP. A versão 4 (IPv4), que é a versão ainda mais popular no momento da escrita deste capítulo, e a versão 6 (IPv6), que é a versão que começa a ser também usada e que tenderá a tornar-se dominante. Sempre que for relevante serão postos em evidência os aspectos em que a nova versão difere da antiga.

17.1 A Internet e o endereçamento IP

Os endereços das componentes de um sistema, e em particular os endereços das interfaces ligadas a uma rede, estão envolvidos em vários problemas, nomeadamente: como gerá-los e afectá-los, e como chegar às entidades a que estão associados.

A maneira mais simples de gerar e afectar endereços consiste em gerá-los de forma descentralizada e aleatória. Com um número suficiente de bits (*e.g.*, 128, 192, 256, ...) a probabilidade de colisão seria da mesma ordem de grandeza que a de descobrir por força bruta uma chave criptográfica com o mesmo número de bits. No entanto, este método também tem vários inconvenientes: a necessidade de usar um grande número de bits (seriam 128 suficientes?) e a necessidade de usar geradores pseudo-aleatórios com diferentes sementes e de qualidade suficiente para evitar colisões. No entanto, o grande inconveniente desta solução tem a ver com o encaminhamento: como localizar um destes endereços na rede global?

A Internet tem hoje em dia vários milhares de milhões de utilizadores. No primeiro trimestre de 2016 só a Akamai ([Akamai Technologies, 2016]) conseguiu recensear cerca de um milhar de milhões de endereços IP diferentes em pacotes que chegaram à sua infra-estrutura global. Como realizar então encaminhamento numa rede com esta dimensão se usássemos endereços aleatórios? Necessariamente, algumas entradas nas tabelas de encaminhamento ou comutação dos comutadores das redes de trânsito teriam de ter muitos milhões de entradas e a localização passaria por métodos sofisticados, envolvendo directórios de grande dimensão².

Uma solução mais simples passa por introduzir alguma forma de agregação ou hierarquia no endereçamento. Assim, os endereços IP estão agregados em conjuntos disjuntos, discriminados por um certo número de bits de prefixo, que se chamam *prefixos IP*. Os endereços de cada um desses conjuntos estão afectados e localizados numa das redes que forma a Internet e portanto há uma correspondência entre prefixos e redes que facilita a localização. Quando um sistema de endereçamento está dependente da localização, diz-se que o mesmo tem **carácter topológico**. Os endereços e os prefixos IP têm uma organização hierárquica e topológica.

Esta problemática da agregação de endereços já foi de alguma forma abordada quando discutimos o endereçamento IEEE 802 (originalmente concebido para as redes Ethernet). Como estas redes se baseiam em difusão, e o encaminhamento repousa em difusão e inundação, elas podem usar quaisquer endereços, desde que distintos uns dos

² Este problema tem de ser hoje em dia solucionado nas rede celulares de milhões de terminais móveis e existe também uma linha de investigação semelhante em redes de conteúdos, que são identificados com identificadores com um grande número de bits.

outros, pois a problemática do encaminhamento está *a priori* resolvida. A geração e gestão dos endereços de nível MAC ficou a cargo dos fabricantes das interfaces. Por isso estes endereços têm prefixos por fabricante, o que facilita a sua geração e gestão, mas esses prefixos não têm carácter topológico pois são independentes da localização.

Os endereços IP também têm prefixos, mas estes são topológicos e relacionados com a localização numa dada rede e por isso não podem ser afectados pelos fabricantes. Na verdade, esta dependência da localização implica que os mesmos sejam afectados pelos gestores das diferentes redes constituintes da Internet.

Aqui chegados, poderíamos colocar a seguinte questão: mas se as interfaces já têm de ter endereços IP, porque razão são necessários os endereços de nível MAC? A resposta passa por constatar que os endereços MAC são bastante mais simples de gerir e afectar, podem ser usados para comunicar antes de as interfaces terem endereços IP definidos e, adicionalmente, podem haver redes que não utilizam sequer endereços IP. De qualquer forma, como veremos a seguir, as redes IPv6 podem usar os endereços de nível MAC para facilitar a gestão e afectação de endereços IPv6.

Devido à escala da Internet, onde existem biliões de interfaces globalmente endereçáveis, e a necessidade de tornar o encaminhamento mais simples, os endereços IP são de carácter topológico e hierárquico, são afectados por prefixos relacionados com a localização das diversas redes constituintes da Internet e são depois geridos e afectados pelos gestores das mesmas.

Endereços IPv4

Os endereços IPv4 têm 32 bits. Esses 32 bits escritos como um número decimal inteiro sem sinal são difíceis de memorizar. Como a representação binária também é difícil de manipular pelos humanos, os endereços IPv4 são representados (para utilização por humanos) usando a designada ***quad notation***, que consiste em representar em notação decimal sem sinal cada um dos 4 (quad) bytes que formam o endereço, separados por um ponto, ver a Figura 17.1.

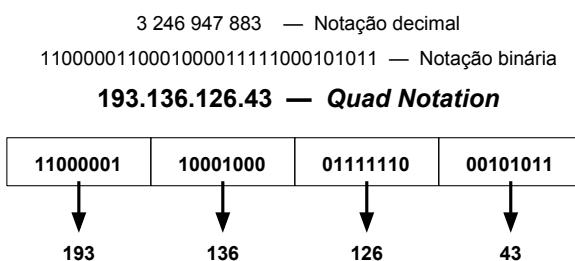


Figura 17.1: Representação do mesmo endereço IPv4 em decimal, binário e usando a *quad notation*.

Dado o carácter topológico do endereçamento IP, os endereços estão divididos em conjuntos, identificados por um conjunto de bits mais significativos, chamado o prefixo do endereço. O conjunto de endereços identificado por um prefixo tem uma cardinalidade que é uma potência de dois³. Daqui para a frente chamaremos a estes conjuntos de endereços **prefixos IP ou simplesmente prefixos**. Cada prefixo é

³ Como foram concebidos para serem manipulados por humanos, os números de telefone utilizam prefixos telefónicos cuja dimensão é uma potência de dez, *i.e.*, são codificados num dado número de algarismos decimais.

identificado pelo conjunto de bits mais significativo que o caracteriza. Repare-se, no entanto, que a dimensão em bits de diferentes prefixos não tem de ser a mesma.

Para denotar um prefixo, utiliza-se também a *quad notation*, complementada com a dimensão em bits do prefixo e separada do mesmo com uma barra. Com esta representação é relativamente mais fácil manipular os endereços e é também mais fácil reconhecer se vários endereços IP pertencem ou não ao mesmo prefixo.

Devido à escala da Internet, onde existem biliões de interfaces globalmente endereçáveis, e a necessidade de tornar o encaminhamento mais simples, os endereços IP são de carácter topológico e hierárquico, são afectados por **prefixos** relacionados com a localização das diversas redes constituintes da Internet e são depois geridos e afectados pelos gestores das mesmas.

Um prefixo IP é representado por um endereço escrito em *quad notation*, com zeros nos bits à direita do prefixo, seguido de uma barra e da dimensão do prefixo.

Admitindo que o prefixo de rede a que o endereço 193.136.124.43 pertence tem 24 bits de comprimento, esse prefixo é representado por 193.136.124.0/24. A notação usada indica que este prefixo tem 24 bits de comprimento e por isso é distinto do prefixo 193.136.124.0/28 que tem 28 bits de comprimento. O exemplo também põe em evidência que um prefixo pode estar contido noutro. Por outro lado, é fácil de reconhecer que os endereços 193.136.124.10 e 193.136.124.129 estão ambos contidos dentro do prefixo 193.136.124.0/24 pois os bits em que os endereços diferem estão todos no quarto byte, enquanto que o prefixo é caracterizado pelo valor dos primeiros três bytes. No entanto, enquanto que o endereço 193.136.124.10 está contido no prefixo 193.136.124.0/28 (que corresponde intervalo de endereços 193.136.126.0 – 193.136.126.15), o endereço 193.136.124.129 não está. O número a seguir à barra que se segue ao endereço em *quad notation* é o número de bits mais significativos que caracterizam o prefixo. Quanto maior for esse número, menor é a cardinalidade do conjunto de endereços que o prefixo denota.

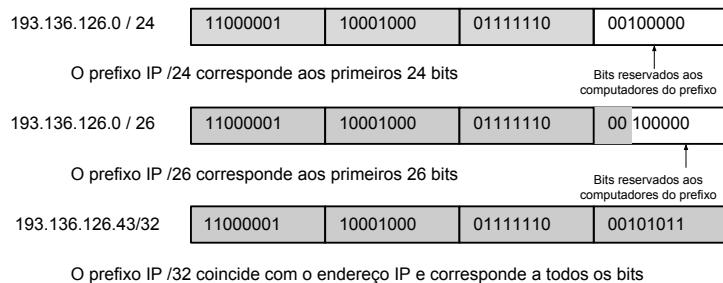


Figura 17.2: Prefixos (a sombreado) e endereços

A Figura 17.2 ilustra o papel dos prefixos dos endereços. Um prefixo IP caracteriza um conjunto de endereços contíguos, alinhado com intervalos da dimensão de uma potência de 2, e associado a uma (sub-)rede.

Cada prefixo IP activo está associado a uma (sub-)rede. Os bits do endereço à direita dos bits do prefixo, *i.e.*, os bits menos significativos, correspondem ao endereço IP da interface dentro do intervalo do prefixo, *i.e.*, dentro da (sub-)rede. Um endereço activo dentro do conjunto do prefixo está necessariamente associado a uma interface que está acessível dentro da (sub-)rede associada ao prefixo.

Um prefixo IP denota um conjunto de endereços IP distintos mas contíguos, e que têm em comum o conjunto de bits mais significativos. O prefixo identifica a (sub-)rede associada do ponto de vista do encaminhamento. Por outro lado, os prefixos têm entre si uma relação hierárquica, ou bem que um está completamente contido no outro, e nesse caso a sua intercepção corresponde ao prefixo mais longo, *i.e.*, o de menor cardinalidade, ou então são disjuntos.

Dado os prefixos serem formados por conjuntos associados a conjuntos mais significativos de bits, das duas uma: ou bem que as duas (sub-)redes são completamente disjuntas, ou bem que uma é um sub-conjunto da outra. A Figura 17.3 mostra vários prefixos IP e ilustra a relação entre os mesmos. Na mesma consideram-se dois prefixos disjuntos ($10.0.0.0/8$ e $193.0.0.0/8$), assim como vários sub-prefixos do segundo, contidos uns nos outros.

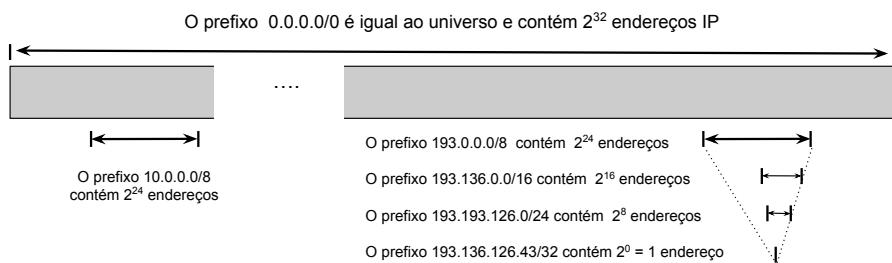


Figura 17.3: Prefixos IP e ilustração da relação entre diferentes prefixos (a dimensão dos prefixos na figura não está à escala).

Existe uma relação estreita entre os prefixos IP e os endereços. Ao nível global da Internet os endereços IP estão associados a interfaces e são opacos aos prefixos a que pertencem pois analisando isoladamente um endereço IP (*e.g.*, 193.136.124.10), não é possível saber a que prefixos activos pertence.

No entanto, o encaminhamento na Internet é realizado em direção a prefixos IP pois cada uma das (sub-)redes da Internet tem um ou mais prefixos associados. Por outro lado, numa primeira aproximação, os sub-prefixos de um dado prefixo estão associados a redes subordinadas à que está associada ao prefixo envolvente. Mesmo tratando-se de uma visão simplificada do processo, o facto é que se torna claro que os prefixos IP são essenciais para tornar o encaminhamento global na Internet mais simples e escalável. As entradas das FIBs (tabelas de encaminhamento ou comutação) dos comutadores IP são indexadas por prefixos IP.

Os diferentes prefixos IP existentes têm entre si relações hierárquicas. Todos os prefixos são sub-prefixos do universo, denotado $0.0.0.0/0$, o prefixo de comprimento nulo. Diferentes sub-prefixos são depois afectados às redes que formam a Internet e sub-prefixos desses são depois afectados a redes subordinadas às mesmas.

Esta hierarquia não é estrita mas desempenha um papel fundamental no encaminhamento pois o encaminhamento no mundo IP consiste em encaminhar os pacotes para os prefixos a que pertencem. É esta característica que está por detrás da escalabilidade do encaminhamento na Internet pois as tabelas de encaminhamento contém prefixos IP e não endereços IP isolados.

Notas históricas sobre os prefixos IP

Quando o protocolo IP foi introduzido, acreditou-se que os 32 bits seriam mais do que suficientes para todos os endereços que viessem a ser necessários, e também se arranjou um esquema de, dado um endereço IP, poder identificar imediatamente a que prefixo ele pertencia. Isso dispensava o uso de comprimento do prefixo pois este estava implícito.

Assim, se o primeiro bit do endereço valesse '0', *i.e.*, se o valor do primeiro byte estivesse entre 0 e 127 (50% do espaço de endereçamento), o prefixo era o /8 e considerava-se que a (sub-)rede a que o mesmo estava associado era uma rede classe A, que era identificada pelo valor do primeiro byte do endereço. Se os dois primeiros bits do endereço valessem '10', *i.e.*, se o valor do primeiro byte estivesse entre 128 e 191(25% do espaço de endereçamento), o prefixo era o /16 e considerava-se que a (sub-)rede a que o mesmo estava associado era uma rede classe B, que era identificada pelo valor dos primeiros dois bytes do endereço. Finalmente, Se os três primeiros bits do endereço valessem '110', *i.e.*, se o valor do primeiro byte estivesse entre 192 e 223, o prefixo era o /24 e considerava-se que a (sub-)rede a que o mesmo estava associado era uma rede classe C, que era identificada pelo valor dos primeiros três bytes do endereço. Este esquema revelou-se demasiado rígido e provocava um grande desperdício do espaço de endereçamento (*e.g.*, em 50% do espaço de endereçamento apenas podiam existir 127 (sub)-redes) e por isso foi abandonado.

O novo esquema, que é o introduzido acima e que continua em utilização, foi batizado por **endereçamento sem classes ou Classless Interdomain Routing (CIDR)**. Em muitos livros de texto continua-se a introduzir a forma antiga (por classes) e só depois se introduz o conceito de CIDR. No entanto, mais de 20 anos depois da sua introdução e do mesmo se ter tornado íntimo e imprescindível ao esquema de endereçamento IP, parece-nos inútil continuar a introduzir o conceito de prefixo IP com base na sua origem histórica, já caída em desuso e para mais inaplicável em IPv6. Esta nota histórica justifica-se apenas para o caso de o leitor tropeçar no termo CIDR e na noção de classe de um endereço IP.

Um outro conceito que caiu em desuso foi o de **máscara de rede IP (IP network mask)** que é uma outra forma de indicar a que prefixo um endereço IP pertence. Uma máscara IPv4 é uma sequência de 32 bits que começa com uma sequência de bits a um, seguida de uma sequência de bits a 0. A máscara correspondente aos endereços classe A, *i.e.*, prefixos da forma /8, é 255.0.0.0 (primeiro byte com o valor '1111111'), a máscara correspondente aos endereços classe B, *i.e.*, prefixos da forma /16, é 255.255.0.0, a máscara correspondente aos endereços classe C, *i.e.*, prefixos da forma /24, é 255.255.255.0, a máscara correspondente ao prefixo /26 é 255.255.255.192, etc.

Para todo os efeitos, a máscara é uma forma de indicar em 4 bytes o comprimento do prefixo, ao invés de em um único byte, mas com a vantagem de permitir calcular imediatamente o prefixo através de uma operação de *and* bit a bit entre a máscara e o valor do endereço. Modernamente a tendência é para deixar ao software o encargo de calcular a máscara e deixar os humanos usarem a notação '*/tamanho do prefixo*' que é mais confortável. No entanto, se o leitor cair sobre uma máscara IPv4 já sabe do que se trata. A mesma notação em IPv6 também é aplicável mas é raramente utilizada.

Antes de penetrarmos de forma mais detalhada na relação entre prefixos IP e encaminhamento, vamos ver a seguir como são afectados os prefixos IP às (sub-)redes.

Como são afectados os prefixos IP

A entidade responsável por gerir o espaço de endereçamento (Versão 4 e Versão 6) da Internet é a **Internet Corporation for Assigned Names and Numbers (ICANN)**, uma organização sem fins lucrativos. A ICANN gera o espaço de endereçamento IP desempenhando o papel de **Internet Assigned Numbers Authority**

(IANA). A ICANN também gera o domínio **root**, e delega a gestão dos seus sub-domínios directos (os TLDs ou *Top Level Domains*). Dado o carácter de monopólio supra-nacional que a ICANN exerce sobre a Internet, esta empresa é gerida segundo um princípio *multistakeholder*, i.e., uma gestão por consenso envolvendo todas as partes a quem os processos dizem respeito.

A gestão dos endereços também é hierarquizada. A IANA afecta prefixos IP globais a diversos usos. A maioria desses prefixos, são prefixos /8 de endereços IP públicos que são afectados às **Regional Internet Registries (RIRs)**. Estas organizações são 5 a nível mundial e repartem entre si a responsabilidade de gestão sobre as diversas regiões do mundo. Por exemplo, a associação **Réseaux IP Européens Network (RIPE)** é responsável pela Europa, Rússia, Médio Oriente e Ásia Central. As RIRs são associações sem fins lucrativos formadas pelos operadores de redes da Internet (ISPs)⁴ da sua região e afectam-lhes sub-prefixos dos prefixos que recebem da IANA.

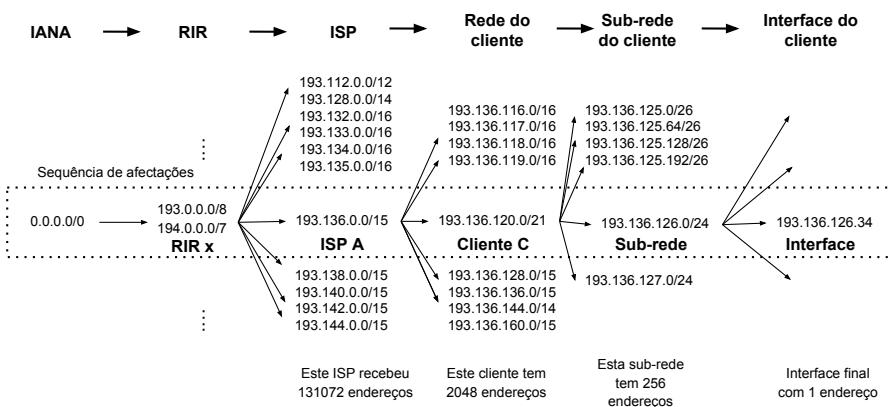


Figura 17.4: Sequência de alocações de prefixos IP: à RIR x, ao ISP A, ao cliente C etc. que permitem a afectação de um endereço a uma interface na rede do cliente C.

A dimensão do prefixo afectado a cada ISP depende da dimensão e importância da sua rede, e indirectamente do valor da sua quota anual para a RIR da sua região. Conforme vai evoluindo e necessita de mais espaço de endereçamento, um ISP pode ir recebendo mais prefixos IP, mas eventualmente disjuntos.

Os ISPs afectam espaço de endereçamento aos seus clientes, i.e., sub-prefixos dos prefixos IP que lhes foram afectados pela sua RIR. A Figura 17.4 ilustra este processo: um prefixo /8 foi afectado a uma RIR, esta afecta um prefixo /15 a um operador (ISP) e este afecta sub-prefixos aos seus clientes, que por sua vez afectam prefixos às suas sub-redes e endereços aos computadores ligados às mesmas.

Prefixos IP e encaminhamento

O encaminhamento na Internet é realizado movendo os pacotes em direção às redes associadas aos prefixos IP que a que pertencem os seus endereços de destino. No entanto, nem todos os prefixos têm de ser visíveis no sistema de encaminhamento global da Internet. Os prefixos afectados pela IANA aos RIRs são meramente administrativos pois não são visíveis pelo sistema de encaminhamento visto que não há “encaminhamento para África”, nem “encaminhamento para a América do Norte”, i.e., o encaminhamento IP não é baseado na localização geográfica. O primeiro nível

⁴ Internet Service Providers.

visível é o do encaminhamento para os prefixos dos operadores, pois estes têm de estar necessariamente visíveis para efeitos de encaminhamento. A seguir coloca-se a questão: e os prefixos dos clientes dos ISPs são visíveis no sistema de encaminhamento da Internet? A resposta é “depende”.

As redes domésticas estão ligadas às redes de acesso dos operadores. O comutador de pacotes que liga cada uma delas à Internet tem um endereço ou um prefixo que apenas é conhecido dentro da rede do operador. Isto é, na Internet estes destinos são alcançados entregando os pacotes que lhes são dirigidos aos comutadores da fronteira da rede do operador que os serve. Só dentro da rede do operador os prefixos mais longos das sub-redes dos clientes do operador, assim como os endereços finais dos clientes domésticos, são conhecidos.

Veremos na Secção 17.3 que muitas dessas redes domésticas têm mais do que um computador ligado, cada um com um endereço diferente, mas todos esses computadores usam um endereço IP partilhado, pelo que em IPv4, o prefixo que está associado a uma rede doméstica é geralmente de comprimento /32, *i.e.*, contém apenas um endereço. Os terminais móveis ligados às redes dos operadores celulares também têm um único endereço IP, contido num prefixo IP do operador celular.

Muitos clientes institucionais (universidades, empresas, *etc.*) recebem prefixos dos seus operadores e muitas dessas redes também não estão visíveis no sistema de encaminhamento global. Para que um pacote lhes chegue, primeiro determina-se o prefixo IP a que o endereço de destino pertence, depois faz-se chegar o pacote à rede do operador à qual o prefixo está associado. Posteriormente, dentro da rede do ISP é realizado o encaminhamento para o prefixo da rede institucional do cliente.

O que acontece se um cliente do ISP A pretender mudar de operador? Se tudo continuar na mesma do ponto de vista dos endereços e dos prefixos anunciados na Internet global, a nova ligação é inútil. De facto ela não será usada, visto que na Internet global a mesma não é conhecida porque o prefixo do cliente continua contido (e escondido) no prefixo do antigo operador. Para ser usada, o prefixo específico do cliente tem de ser anunciado pelo novo operador para que o encaminhamento para o cliente possa ser realizado via o novo operador (*e.g.*, ISP B), ao invés de via o antigo operador (ISP A).

Fazendo referência à Figura 17.4, o anúncio 193.136.0.0/15 **acessível via A** é anunciado pelo ISP A, o que é suficiente para enviar pacotes para o cliente C com o prefixo subordinado 193.136.120.0/21. Se o cliente C quiser mudar para o ISP B, das duas uma, ou muda de endereços, o que pode ser complexo e dispendioso, ou deixa de ser acessível na Internet pois os pacotes serão enviados para uma rede (a do ISP A) sem ligação à sua. A outra alternativa, sem mudar de endereços, é o ISP B passar a fazer um anúncio 193.136.120.0/21 **acessível via B**. Só que agora os pacotes dirigidos ao cliente C (por exemplo 193.136.126.43) pertencem a dois prefixos nas tabelas de encaminhamento: 193.136.0.0/15 **via A** e 193.136.120.0/21 **via B**. Se for escolhido o prefixo mais curto (/15) o pacote irá parar a um beco sem saída pois o ISP A deixou de ter ligação a C.

Para que o encaminhamento funcione correctamente é aplicada a regra **o encaminhamento pelo prefixo mais longo é o melhor (the longest prefix is the best)**. Com esta regra, o anúncio via B tem precedência e os pacotes dirigidos a 193.136.126.43 vão sempre via o ISP B, que é o que se pretendia.

Supondo agora que o desejo do cliente C era assegurar conectividade à Internet via os dois operadores (ISP A e ISP B), então teriam de ser feitos dois anúncios para além dos anúncios dos prefixos dos operadores A e B: prefixo 193.136.120.0/21 **acessível via A e acessível via B**. Se aparecesse apenas o anúncio do prefixo do cliente C via o ISP B, então a ligação ao ISP A seria inútil porque nunca seria usada⁵. Quando um cliente pretende estar ligado a mais do que um ISP, esse cliente diz-se *multi-homed*.

⁵ A outra alternativa seria não fazer nenhum anúncio específico do prefixo do cliente C. No entanto, como veremos no Capítulo 18, essa solução não seria muito adequada.

Devido à regra *the longest prefix is the best*, um cliente pode mudar de operador sem mudar de prefixo IP, mas os operadores, por razões comerciais e também técnicas, começaram a restringir a hipótese de os clientes usarem prefixos IP dos seus blocos sem lhes estarem ligados, ou seja sem estarem ligados ao “dono” dos endereços. Quando um prefixo de um cliente está contido no prefixo de um operador, esse prefixo diz-se pertencer ao **espaço de endereçamento dependente de provedores** (*provider dependent address space*).

Para os clientes com dimensão razoável que prevêem que mais tarde ou mais cedo desejem mudar de ISP e não queiram mudar de endereços nessa altura, ou desejem *a priori* estar ligados a vários ISPs, as RIRs dispõem também de blocos de endereços que não estão afectados a nenhum ISP e funcionam como um ISP virtual para efeitos de administração de endereços, pois afectam directamente prefixos específicos a clientes finais de dimensão razoável. Um prefixo desse tipo diz-se que pertence ao **espaço de endereçamento independente** (*provider independent address space*) e tem de ser anunciado directamente na Internet por um ou mais ISPs pois o prefixo global em que está contido não é anunciado por nenhum operador.

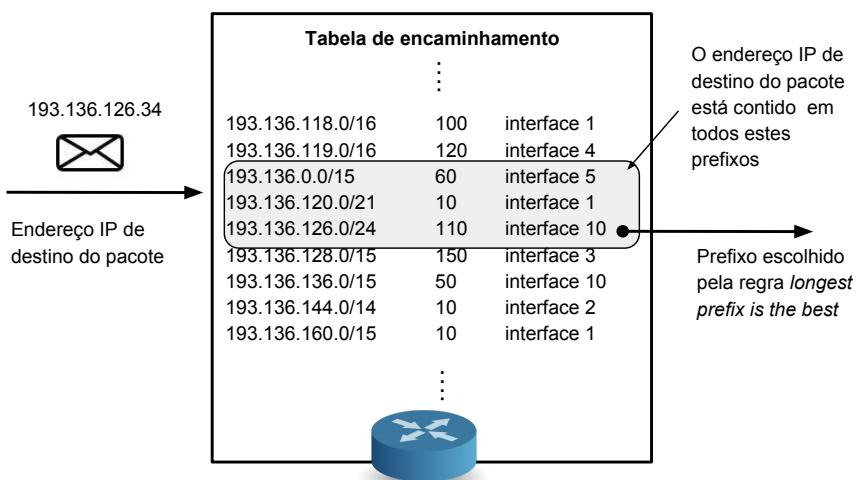


Figura 17.5: Tabela de encaminhamento IP e a regra *longest prefix is the best*

Figura 17.5 mostra um trecho de uma tabela de encaminhamento global (*i.e.*, da FIB de um comutador que conhece os prefixos IP encaminhados na Internet global⁶) e como a regra *longest prefix is the best* é usada para seleccionar a interface pela qual o pacote deve seguir. A flexibilidade permitida por esta regra, e o conjunto dos mecanismos de afectação de endereços IP que foram introduzidos, permite o máximo de concorrência do ponto de vista comercial entre ISPs e também todas as formas de interligação que se revelarem desejáveis por razões de desempenho ou segurança.

No entanto, os mesmos têm dois efeitos nefastos. O primeiro tem a ver com o facto de que a consulta das FIBs deixou de ser possível usando simplesmente uma estrutura de dados do tipo tabela de dispersão (*hash tables*) e passou a ser necessária um estrutura mais sofisticada, caracterizada como uma árvore de prefixos. A raiz dessa árvore está associada ao prefixo 0.0.0.0/0. Os filhos da raiz são os sub-prefixos de topo dos operadores e os correspondentes aos prefixos pertencentes aos *provider independent address spaces*. A seguir, qualquer outro prefixo conhecido pelo comutador, tem de ser

⁶ A Internet global é também designada por *Default-Free Zone* (DFZ) por razões que serão mais claras a seguir.

filho do maior prefixo que o contém. A gestão desta estrutura é muito mais pesada que a simples gestão de tabelas de dispersão, o que encarece a construção e a operação dos comutadores de alta gama.

O segundo efeito nefasto tem a ver com a explosão do número de prefixes IP específicos que têm de ser conhecidos na Internet global. No limite, se cada rede doméstica tivesse endereços no espaço *provider independent address space*, as tabelas de encaminhamento globais teriam de ter biliões de entradas com prefixes de comprimento /32. É por esta razão que existe pressão para só anunciar globalmente prefixes curtos (*i.e.*, com um pequeno número de bits), pois isso torna as tabelas de encaminhamento globais menos detalhadas e portanto de menor dimensão.

No início do ano de 2016 a tabela de prefixes IP visíveis globalmente, *i.e.*, a tabela de encaminhamento dos comutadores de pacotes do “coração” da Internet, continha mais de 600.000 entradas (prefixos distintos que exigem encaminhamento específico). Sempre que os comutadores se tornaram mais lentos, ou mesmo incapazes de lidarem com tabelas da dimensão e complexidade necessária, porque a tecnologia do momento não o permitia, muitos operadores Tier 1 e 2, ver o Capítulo 4, os que mais sofreram com o problema, começam a não aceitar os anúncios de prefixes muito longos (*i.e.*, com muitos bits), o que limita a flexibilidade do encaminhamento para os clientes finais e tem implicações comerciais de exercício de poder ligado à dimensão da rede.

A Figura 17.6 mostra um gráfico da evolução do número de prefixes visíveis na Internet global ao longo dos anos.

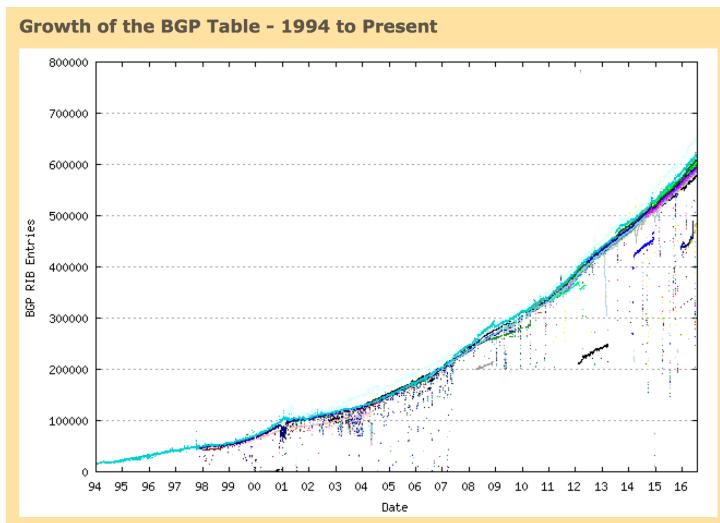


Figura 17.6: Evolução do número de prefixes IP visíveis nas tabelas de encaminhamento globais da Internet segundo o site <http://bgp.potaroo.net>. Consulta realizada em Julho de 2016.

O encaminhamento na Internet global, a chamada *Default-Free Zone*, é realizado com base nos prefixos IP anunciados pelas diferentes redes interligadas. Para evitar que uma hierarquização total dos prefixos redundasse num sistema demasiado rígido, e potencialmente com as características de um monopólio dos operadores Tier1, são anunciados prefixos correspondentes a todas as redes, mesmo de clientes finais, desde que estas tenham a necessidade de controlar de forma mais fina o encaminhamento dos pacotes que lhes são dirigidos.

Os prefixos afectados pelos operadores aos seus clientes fazem parte do espaço *provider dependent address space*. Esses clientes, se não forem *multi-homed*, estão num espaço de endereçamento que também se pode chamar *provider-based aggregation address space* porque poupa entradas nas tabelas de encaminhamento. Ao contrário, os prefixos das redes clientes mas independentes do fornecedor, têm prefixos no *provider independent address space* ou *without aggregation address space* e expandem necessariamente as tabelas de encaminhamento globais.

A regra *longest prefix matching is the best* na escolha das entradas das tabelas de encaminhamento dos comutadores funciona como garante desta flexibilidade. O controlo fino do encaminhamento para uma rede, e as necessidades de *multi-homing* levam à subida incessante do número de prefixos IP visíveis no “coração” da Internet.

Prefixos IPv4 especiais

As regras de afectação de endereços IPv4 comportam diversos prefixos IP especiais que não podem figurar nas tabelas de encaminhamento globais e que têm utilizações especiais, nomeadamente os da Tabela 17.1, onde figuram os principais blocos de endereços desse tipo. Os blocos de endereços IPv4 privados são discutidos no fim da Secção 17.3. O bloco de endereços IP multicasting são usados para grupos de IP multicasting, ver a Secção 5.2. O endereço 127.0.0.1 é o endereço de *loopback*, i.e., o endereço de uma interface virtual existente em todos os dispositivos que comunicam em IP, e que denota localmente o próprio dispositivo. Este endereço também é conhecido pelo nome *localhost*, ver a Secção 5.1. Os *link local IP addresses* são endereços IP que podem ser afectados localmente pelo software IP mas que só são válidos para comunicar com computadores ou outros dispositivos ligados ao mesmo canal. ver a Secção 17.3

A quem pertencem e onde estão os endereços IP

Dado um endereço IP é possível consultar as bases de dados das RIRs e obter informação sobre a sua afectação. Este serviço está disponível via o comando `whois`. O exemplo abaixo mostra parte do resultado da execução da interrogação `whois -h whois.ripe.net 193.136.126.43` à base de dados do RIPE. O resultado mostra que o endereço 193.136.126.43 pertence ao prefixo IP 193.136.120.0/21 que foi afectado à Faculdade de Ciencias e Tecnologia da Universidade Nova de Lisboa (FCT/UNL) em 30 de Novembro de 1993 pelo serviço IP da UNL e que se trata de um prefixo do PA - *Provider Aggregated address space*. Com efeito este prefixo faz parte de um prefixo maior afectado pela Rede das Universidades Portuguesas à UNL.

Tabela 17.1: Principais prefixos IPv4 reservados, ver o RFC 6890

Nome	Intervalo ou endereço	# endereços disponíveis	Prefixo IP	RFC
24-bit block private IP addresses	10.0.0.0 a 10.255.255.255	16 777 216	10.0.0.0/8	RFC 1918
20-bit block private IP addresses	172.16.0.0 a 172.31.255.255	1 048 576	172.16.0.0/12	RFC 1918
16-bit block private IP addresses	192.168.0.0 a 192.168.255.255	65 536	192.168.0.0/16	RFC 1918
multicast IP addresses	224.0.0.0 a 239.255.255.255	268 435 456	224.0.0.0/4	RFC 1122
loopback IP addresses	127.0.0.1 a 127.255.255.255	16 777 216	127.0.0.0/8	RFC 1122
loopback IP address	127.0.0.1	1	127.0.0.1/32	RFC 1122
link local IP address	169.254.0.0 a 169.254.255.255	65 536	169.254.0.0/16	RFC 3927

```
$ whois -h whois.ripe.net 193.136.126.43
% This is the RIPE Database query service.
...
193.136.120.0 - 193.136.127.255
PT-FCT-UNL-1
Faculdade de Ciencias e Tecnologia da Universidade Nova ... PT
RCUN1-RIPE
ASSIGNED PA
ORG-UNDL3-RIPE
SERVIP-UNL
created 19931130
```

Uma outra faceta interessante dos endereços IP tem a ver com o facto de que hoje em dia é possível obter informação sobre a localização geográfica dos endereços IP, como já foi referido na Secção 13.2.

Endereços IPv6

Os endereços da versão 6 do protocolo IP, ou IPv6, têm 128 bits, que acomodam $2^{128} = 340\ 282\ 366\ 920\ 938\ 463\ 463\ 374\ 607\ 431\ 768\ 211\ 456$ endereços distintos.

Para a representação legível destes endereços foi introduzida uma notação designada **hexadecimal colon notation**: cada endereço é escrito usando 8 grupos de 4 dígitos hexadecimais separados pelo símbolo dois pontos. Cada grupo de dígitos tem 4 símbolos e representa 16 bits. Um exemplo de utilização desta notação é o endereço: `2001:0:0:0:45bf:ff23:8b4f`.

A notação admite igualmente que qualquer sequência de grupos só com zeros possa ser omitida, sendo substituída por dois símbolos dois pontos seguidos. Em cada endereço só pode haver uma destas substituições, senão resultaria numa ambiguidade (excepto no caso particular do endereço "0::0" que pode ser abreviado para "::"). Aplicando esta regra ao exemplo anterior resulta numa representação mais compacta do mesmo endereço: `2001::45bf:ff23:8b4f`.

Os endereços IPv6 são afectados com a mesma lógica que os endereços IPv4, nomeadamente usando prefixos IPv6. Existem igualmente diversos prefixos IPv6 reservados. A Tabela 17.2 apresenta os principais.

Tabela 17.2: Principais prefixos IPv6 reservados, ver o RFC 6890

Nome	Intervalo ou endereço	# endereços disponíveis	Prefixo IP	RFC
reserved for special purposes	0:0:0:0:0:0:0 a 00ff:ffff:ffff:...ffff:ffff	2^{120}	::0/8	RFC 4291
unspecified IP address	0:0:0:0:0:0:0	1	::0/128	RFC 4291
loop-back IP address	0:0:0:0:0:0:1	1	::1/128	RFC 4291
multicast IP address	ff00:0::0 a ffff:ffff:...:ffff:ffff	2^{120}	ff00::/8	
link-local IP addresses	fe80:: a febf:ffff:...:ffff:ffff	2^{118}	fe80::/10	RFC 4291
site-local IP addresses	fec0:: a feff:ffff:...:ffff:ffff	2^{118}	fec0::/10	RFC 4193

Os endereços que começam com 8 bits a zero (o prefixo ::0/8) são reservados para funções especiais. O endereço "::0" é um endereço que significa ausência de endereço e que só pode ser usado por uma interface antes de ter um endereço próprio, para enviar um pacote em difusão à procura de quem lhe afecte um endereço. O endereço "::1" é o endereço *loopback* que corresponde ao nome `localhost` em IPv6 e é equivalente ao endereço 127.0.0.1 em IPv4. Nesta gama de endereços estão também previstas formas especiais de representar em IPv6 endereços IPv4. Esses endereços começam com 96 bits a zero seguidos dos 32 bits do endereço IPv4 ou com 80 bits a zero, seguidos de 16 bits a um, seguidos dos 32 bits do endereço IPv4.

Os endereços que começam com 8 bits a um são endereços IPv6 multicasting. Os endereços *link-local IP addresses* são endereços IP que podem ser afectados localmente pelo software IP mas que só são válidos para comunicar em IP com computadores ou outros dispositivos ligados ao mesmo canal. Os endereços *site-local IP addresses* têm em IPv6 o mesmo papel que os endereços privados em IPv4.

Actualmente a ICANN está a afectar blocos de endereços IPv6 às RIRs no prefixo 2000::/3 (com 001 nos primeiros 3 bits do endereço) para serem distribuídos como endereços unicast. As afectações de prefixos IPv6 estão também registadas pelos RIR. Abaixo mostra-se parte do resultado de executar o comando `whois -h whois.ripe.net` sobre o prefixo 2001:690::/32. Através do resultado constata-se que se trata de um prefixo afectado pela Fundação para a Ciéncia e a Tecnologia à rede Portuguese National Research & Education Network como sub-prefixo do seu prefixo 2001:690::/29.

```
$ whois -h whois.ripe.net 2001:690::/32
% Information related to '2001:690::/29'

inet6num:      2001:690::/29
netname:       PT-RCCN-20000623
country:        PT
org:           ORG-FpaC1-RIPE
admin-c:        JNF1-RIPE
status:         ALLOCATED-BY-RIR
.....
created:       2013-11-22T07:46:28Z
source:        RIPE

organisation:  ORG-FpaC1-RIPE
```

```

org-name: Fundacao para a Ciencia e a Tecnologia, I.P.
...
address: PORTUGAL

% Information related to '2001:690::/32AS1930'

route6: 2001:690::/32
descr: The Portuguese National Research & Education Network
origin: AS1930
...
created: 2005-04-06T14:05:57Z
source: RIPE

```

Depois desta discussão sobre o endereçamento IP e as sua relações com o encaminhamento e a estrutura interna da Internet, vamos de seguida estudar o protocolo IP propriamente dito.

17.2 IP versão 4 e IP versão 6

O protocolo IP especifica o formato dos pacotes trocados na rede, compreendendo um cabeçalho (os primeiros bytes transferidos) com informação para o funcionamento do protocolo, seguido do *payload*, ou seja, os bytes com os dados que se pretende transferir, ver a Figura 17.7. O protocolo especifica igualmente a forma como os campos do pacote devem ser interpretados e tratados pelos comutadores de pacotes IP durante o encaminhamento do mesmo para o destino. Tradicionalmente, os comutadores de pacotes capazes de processarem pacotes IP e realizarem todas as operações envolvidas nesse processamento chamam-se comutadores IP, e em língua inglesa *IP routers*. Devido ao facto de que os especialistas em redes IP usam o termo em inglês mesmo quando a sua língua materna não é o inglês, no resto deste capítulo utilizaremos o termo *router IP*, ou simplesmente *router*, para designar os equipamentos capazes de realizarem processamento inteligente de pacotes IP e outros protocolos que lhe estão associados.

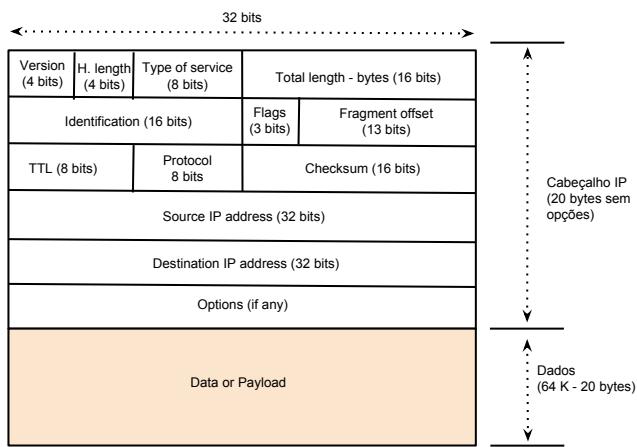


Figura 17.7: Cabeçalho de um pacote IP na versão 4

Começamos por discutir o papel dos diferentes campos do cabeçalho IPv4. O primeiro campo, com 4 bits, indica a versão do protocolo a que o pacote pertence.

Neste caso será necessariamente o valor 4 correspondente a esta versão. O campo seguinte é o campo **Header length** que especifica a dimensão do tamanho do cabeçalho em grupos de 4 bytes. Este campo é necessário pois o cabeçalho tem comprimento variável, dependente da presença de opções. Na ausência de opções, o cabeçalho ocupa $5 \times 4 = 20$ bytes pelo que este campo tem o valor 5 nesse caso. O maior cabeçalho pode ter até $15 \times 4 = 60$ bytes, pelo que as opções podem ocupar até 40 bytes.

O campo **Type of service** permite codificar uma indicação sobre o tipo de serviço que o emissor original do pacote pretende obter para o mesmo. Dependendo das funcionalidades implementadas pelos *routers* da rede ou redes atravessadas pelo pacote, essas indicações podem ou não ser respeitadas. No momento em que este capítulo foi escrito, na maioria das redes de trânsito este campo é ignorado. É neste campo que se podem registar igualmente informações sobre a saturação dos *routers* atravessados pelo pacote usando a funcionalidade ECN, ver a Secção 8.4.

O campo **Total length** indica a dimensão total do pacote em bytes. Com 16 bits o campo permite que um pacote IP tenha no máximo 64 Kbytes. Como já foi referido, a grande maioria dos canais não permitem transmitir num único *frame* um pacote desta dimensão. Nesse caso, o protocolo IPv4 prevê que o *router* que se confronte com esta situação fragmente o pacote. Os campos **Identification**, **Flags** e **Fragment Offset** destinam-se a suportar esta funcionalidade que será discutida a seguir.

O campo **Time-To-Live** ou **TTL** é já nosso conhecido e destina-se a prevenir que um pacote envolvido num ciclo de encaminhamento permaneça eternamente dentro da rede. Pode também ser usado para outros efeitos, como por exemplo para limitar o âmbito do percurso de um pacote na rede, ver a Secção 3.4. Quando um pacote chega a um *router* este tem de decrementar o valor do TTL, caso o valor que daí resulte seja 0, o pacote deve ser destruído (ignorado).

Inicialmente pensava-se que o campo limitaria o tempo de vida dos pacotes em segundos, mas nunca foi realista implementar esta visão pois é muito difícil a um *router* calcular exactamente quanto tempo leva a processar o pacote, acrescido do tempo necessário para este chegar ao *router* seguinte. As implementações iniciais limitavam-se a decrementar de uma unidade o valor do TTL. Assim, a implementação que acabou por ser dominante é equivalente a um simples limitador do número de *routers* atravessados (*hop-count*). O valor do campo é inicializado pelo emissor original do pacote e os valores típicos de inicialização por omissão são 64 e 255, ver a Secção 3.4.

O campo **Protocol** contém um código que especifica qual o protocolo a que pertence o *payload* do pacote (e.g., ICMP, UDP, TCP, ...).

O campo **Checksum** contém um código de deteção de erros que também já é nosso conhecido, ver a Secção 2.4. Ele apenas detecta erros no cabeçalho do pacote e é calculado com o próprio valor a 0, quer pelo emissor, quer pelo receptor (ver a noção de *pseudo-header* nos RFCs 791 e 1812).

Este mecanismo foi introduzido numa época em que as técnicas de deteção de erros em alguns canais eram deficientes e destinava-se a evitar que um *router* analisasse pacotes IP com o cabeçalho corrompido. Actualmente todos os canais incluem mecanismos de deteção de erros com maior precisão que este, e os protocolos de nível superior também já incluem um campo do tipo *checksum*. O mecanismo foi abandonado em IPv6 e muitas implementações actuais do software IP deixaram de analisar este campo, mas fazem o seu cálculo antes da emissão de um pacote para respeitarem a norma e evitarem problemas com *routers* que o calculem. Com efeito, dado que o cabeçalho é sempre alterado (pelo menos o TTL é sempre decrementado), esta operação, que no essencial é inútil, tem de ser executada por todos os *routers*. Trata-se de mais um exemplo do papel benéfico, mas também limitador da evolução, que as normas introduzem.

Os dois campos obrigatórios finais são os endereços origem e destino cujos significado e utilização são óbvios.

A problemática da fragmentação em IPv4

Quando o protocolo IP foi definido inicialmente, devido a elevadas taxas de erro e baixo débito, alguns canais tinham um MTU (*Maximum Transfer Unit*), ver a Secção 2.5, muito baixo, com apenas algumas centenas de bytes. Por esta razão o IPv4 tem um mecanismo que permite a um *router* confrontado com um pacote IP superior ao MTU do canal porque o deve transmitir, fragmentar esse pacote em vários fragmentos. No entanto, para prevenir o caso em que a operação tenha de ser repetida por diferentes *routers* para o mesmo pacote, foi adoptada a solução de que a recomposição (agrupamento dos fragmentos do pacote original) só é realizada no destino final do pacote, pois cada fragmento pode ser encaminhado de forma independente. Ou seja, adoptou-se a posição de que se houver necessidade de fragmentar um pacote, a probabilidade de este voltar a ter de ser fragmentado é elevada. Por outro lado, realizar a recomposição do pacote original apenas no destino também simplifica o trabalho e a gestão de memória dos *routers* intermediários visto que estes não têm de memorizar fragmentos à espera para posterior recomposição do pacote.

O campo **Identification** permite identificar os diferentes fragmentos como pertencentes ao mesmo pacote. O campo **Flags** permite distinguir o último fragmento dos restantes, e o campo **Fragment Offset** permite colocar cada fragmento na sua posição certa no pacote mesmo que os fragmentos cheguem fora de ordem. Para mais detalhes consultar os RFCs 791 e 815.

Adicionalmente, caso um fragmento se perca, como este só é enviado uma vez, o receptor será obrigado a destruir os restantes fragmentos, ao fim de um intervalo de tempo marcado por um alarme, pois não conseguirá recompor o pacote original. Dadas as características da Internet, este alarme tem de ser necessariamente de alguns segundos. Ver a noção de *Maximum Packet Life Time* na Secção 7.3.

Finalmente, como o campo **Identification** só admite 65535 identificações diferentes antes de obrigar à reutilização dos identificadores, com os débitos actuais, essa quantidade de pacotes pode ser emitida num período inferior à duração do alarme associado à recomposição dos fragmentos no receptor. Na eventualidade da perda de um fragmento, deixaria de estar garantida a fiabilidade do processo de recomposição de pacotes. Na Secção 17.6 é proposto um problema sobre esta possibilidade.

Actualmente quase todos os canais têm MTUs bastante mais elevados, com pelo menos 1,2 ou 1,5 Kbytes ou até bastante mais (e.g., *jumbo frames* de vários K bytes). Com efeito, mesmo os canais sem fios que usam às vezes *frames* menores, recompõem nas interfaces dos extremos do canal *frames* lógicos de maior dimensão. Assim, actualmente é preferível forçar o emissor original do pacote a emitir imediatamente pacotes IP de menor dimensão, do que envolver os *routers* e o destinatário do pacote na fragmentação.

Por todas estas razões realizar fragmentação dentro da rede é hoje em dia considerada uma má prática e o mecanismo foi abandonado em IPv6 ao nível dos *routers*. Mesmo em IPv4 recomenda-se igualmente a utilização de um protocolo designado *path MTU discovery*, ver o RFC 1981, e enviar apenas pacotes de dimensão compatível com esse MTU. É o que geralmente faz o protocolo TCP na abertura de uma sessão ao calcular o MSS (*Maximum Segment Size*), ver a Secção 7.3.

Opções IPv4

Quando o protocolo IP foi definido, o seu cabeçalho recebeu o campo **Options** que permite introduzir mecanismos opcionais de forma flexível. A maioria das opções definidas destinam-se a permitir realizar *debug* da rede, como por exemplo a opção de registar o endereço dos *routers* que o pacote atravessou. Outras opções destinam-se a permitir a implementação de formas diferentes de encaminhamento, nomeadamente diversas formas de **encaminhamento controlado pelo emissor (source routing)**. Este tipo de opções permite ao emissor de um pacote indicar o endereço de *routers* por onde este deve passar no seu caminho para o destino.

As opções revelaram-se “um pau de dois bicos”. Por um lado, pelo menos à primeira vista, este tipo de mecanismos parecem ser úteis e desejáveis. Mas infelizmente alguns inconvenientes parecem suplantar os benefícios.

Toda a filosofia e o desempenho das redes IP deve basear-se na ideia de que o trabalho dos *routers* deve ser o mais simples que possível, relegando a complexidade para a periferia. As opções introduzem complexidade adicional nos *routers* e por essa razão a minoria de pacotes que as contém são sempre processados à parte e sem apoio de hardware específico. Com efeito, as opções não são amigas nem do desempenho, nem da simplicidade.

Este defeito poderia não ser demasiado grave visto que as opções são opcionais e só “paga o seu preço quem as usar”, mas para infelicidade do mecanismo a maioria das opções que envolvem *source routing* revelaram-se fatais do ponto de vista da segurança. Por essa razão a maioria dos *routers* dos operadores são parametrizados para ignorarem as opções IPv4.

IP versão 6

A motivação principal para a introdução da versão 6 do protocolo IP veio da consciência de que um endereço IP com 32 bits iria tornar-se, mais tarde ou mais cedo, uma barreira intransponível à expansão da Internet. De facto, no momento em que este capítulo foi escrito todos os RIRs, com excepção do de África, já não podem afectar mais endereços IPv4 aos operadores e o *provider independent address space* também se encontra esgotado. Existem apenas pequenas reservas de endereços IPv4 que só são afectados em casos especiais.

Alterar a dimensão dos endereços passou necessariamente por uma modificação significativa do cabeçalho, por isso, aproveitou-se a experiência adquirida até então e introduziram-se outras alterações. O novo cabeçalho é maior mas mais simples que o anterior, ver a Figura 17.8.

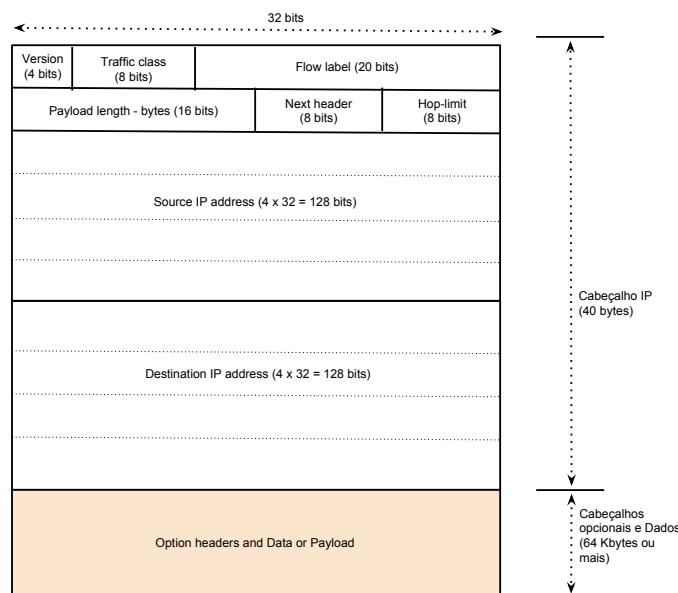


Figura 17.8: Cabeçalho de um pacote IP na versão 6

Na versão 6 do protocolo IP o cabeçalho é de dimensão fixa e não existem vários dos campos presentes em IPv4. Em particular foram suprimidos o campo *checksum*, o campo com a dimensão do cabeçalho pois este é de dimensão fixa, e não é suportada fragmentação dos pacotes pelos *routers*, nem opções. Apesar do cabeçalho mínimo IPv4 (com 20 bytes) ter sido expandido para 40 bytes em IPv6, o número de campos é inferior. A expansão da dimensão deve-se ao facto de que os endereços cresceram 4 vezes em dimensão (de 4 para 16 bytes cada um).

No que diz respeito à fragmentação a norma determina que todos os canais devem ter um MTU maior ou igual a 1280 bytes. Se necessário para garantir este limite inferior, as interfaces do canal devem implementar a fragmentação ao nível canal mas de forma transparente para os *routers*. Caso um *router* receba um pacote que necessite de fragmentação, o mesmo é suprimido e ocorre um erro.

O campo **Version** tem a mesma posição, função e dimensão que no cabeçalho IPv4. Naturalmente, neste caso o seu valor é sempre 6.

O campo **Traffic class** tem a mesma função e dimensão que o campo **Type of service** no cabeçalho IPv4. Tal como no caso do IPv4 este campo é na verdade constituído por dois sub-campos: 6 bits codificam o tipo de serviço e designam-se em IPv6 **Differentiated Services (DS)** e 2 bits dizem respeito à funcionalidade ECN. O campo **Flow label** não tem equivalente no cabeçalho IPv4. O seu papel é permitir identificar um conjunto de pacotes como pertencendo a um fluxo específico e que devem ser tratados de forma particular pelos *routers*. No entanto continua a ser considerado um mecanismo experimental que não é considerado em ambientes de produção.

O campo **Payload length** contém a dimensão do **payload** do pacote incluindo eventuais cabeçalhos opcionais. O campo **Next header** contém um código a indicar qual é cabeçalho seguinte. O mesmo pode ser um protocolo de nível superior como no IPv4 (*e.g.*, ICMP, UDP, TCP, ...) ou um cabeçalho opcional do IPv6. O campo **Hop limit** tem a mesma função e dimensão que o campo TTL no cabeçalho IPv4 e os campos **Source IP Address** e **Destination IP Address** têm os significados óbvios.

Cabeçalhos opcionais ou de extensão IPv6

O cabeçalho IPv6 descrito acima apenas permite a implementação de mecanismos imprescindíveis ao encaminhamento, assim como funcionalidades opcionais de qualidade de serviço. Provavelmente, a grande maioria dos pacotes não precisam de quaisquer outros tratamentos. As funcionalidades opcionais do IPv6 são introduzidas através de cabeçalhos opcionais ou de extensão que são colocados imediatamente a seguir ao cabeçalho IPv6. A norma admite que sejam colocados vários dos cabeçalhos opcionais seguidos no mesmo *datagrama* IPv6 desde que os mesmos não sejam incompatíveis entre si.

Estes cabeçalhos permitem a introdução de funcionalidades opcionais como por exemplo: *source routing* e outros processamentos ditos *hop-by-hop*, e fragmentação mas só extremo a extremo. A *Jumbo Payload Option* permite aos extremos enviar pacotes com mais de 64 Kbytes (65535 bytes), também chamados de *Jumbograms*, e os cabeçalhos de segurança introduzem autenticação e cifra dos pacotes. Com exceção dos cabeçalhos com as opções de encaminhamento e outras acções do tipo *hop-by-hop*, os mecanismos introduzidos por estes cabeçalhos de extensão são extremo a extremo e podem, no essencial, ser ignoradas pelos *routers* intermédios.

O protocolo IP é a “cola” ou “camada unificadora” da Internet pois permite que todos os equipamentos que o suportam possam trocar pacotes de dados entre si. O cabeçalho IP na versão 4, a versão mais comum actualmente, para além dos endereços IP origem e destino do pacote, contém um conjunto de campos que apoiam o encaminhamento dos mesmos, nomeadamente nas facetas relacionadas com a prevenção de ciclos de encaminhamento (TTL) e qualidade de serviço.

Devido à falta de experiência e às características tecnológicas das redes quando o protocolo foi definido, o cabeçalho IPv4 inclui igualmente o suporte de mecanismos que se vieram a revelar pouco amadurecidos, como por exemplo a fragmentação, e o facto de o cabeçalho ser de comprimento variável devido às presenças no mesmo de várias opções (*e.g., source routing, record route, etc.*). Na prática, as opções IPv4 pouco ou nada são suportadas e a fragmentação não é recomendada.

A versão 6 do protocolo IP introduz uma alteração fundamental ao nível da dimensão e formato dos endereços IP, e suprimiu alguns mecanismos que se revelaram inúteis ou contraproducentes em IPv4. Adicionalmente, substituiu as opções por uma eventual pilha de cabeçalhos de extensão, sempre que são necessárias funcionalidades opcionais semelhantes às opções do IPv4, ou ainda novas opções de segurança, *jumbograms*, *etc.* Essas opções são fundamentalmente *end-to-end* e pouco utilizadas tal como em IPv4.

17.3 Encaminhamento de pacotes IP

Nesta secção vamos penetrar nos detalhes concretos de como é realizado o encaminhamento de pacotes IP. Começaremos por analisar como os sistemas nos extremos da rede recebem endereços e colaboram no encaminhamento. Será então posto em evidência como eles são poupadados às principais complexidades do encaminhamento e em particular como se evita que tenham de conhecer todos os destinos possíveis, mesmo quando estão ligados a mais do que um *router*. Veremos também como é realizado o processamento dos pacotes pelos *routers*.

Planos de endereçamento IP

Todos os equipamentos que enviam e recebem pacotes IP dispõem de interfaces para canais. **Essas interfaces têm necessariamente um endereço IP específico.** Assim, para além do endereço *loopback*, qualquer um destes equipamentos tem tantos endereços IP diferentes quanto as suas interfaces activas. De facto, os endereços IP não estão associados a equipamentos mas sim a interfaces. O endereço IP associado à interface, é distinto e não substitui qualquer endereço de nível MAC (nível canal) que a interface também tenha.

Os endereços IP das interfaces pertencem aos prefixos IP do canal a que as mesmas estão ligados. Com efeito, **todos os canais usados no encaminhamento IP têm associado um prefixo IP específico.** Assim, se o canal é ponto-a-ponto, esse prefixo contém geralmente pelo menos 4 endereços: o primeiro endereço do prefixo, um endereço para cada extremidade, e um endereço dito endereço de *broadcast*. Quando o canal é um canal em difusão, *i.e.*, multi-ponto, o seu prefixo tem de acomodar tantos endereços quantos os sistemas acessíveis pelo canal mais os dois endereços dos extremos (o do prefixo e o de *broadcast*)⁷. O endereço de *broadcast* só existe em IPv4 e serve

⁷ Na verdade o endereço do prefixo também pode ser usado por uma das interfaces ligadas

para endereçar todos os sistemas ligados ao mesmo canal e com o mesmo prefixo. Em IPv6 não existem endereços deste tipo e utilizam-se endereços IP Multicasting para o mesmo efeito.

Do ponto de vista do IP a noção de canal multi-ponto abrange qualquer infra-estrutura que permita endereçar directamente todos os sistemas finais ligados à mesma, como por exemplo uma rede de *switches* Ethernet. O nível IP considera uma tal rede, mesmo constituído por dezenas de *switches* Ethernet ligando centenas de computadores, equivalente a um canal multi-ponto pois os pacotes podem ser endereçados directamente a outro sistema ligado à mesma infra-estrutura sem necessidade de operações de encaminhamento IP via algum *router*. Para isso basta encapsular o pacote num *frame* do nível canal. Ou seja, o nível IP equipara essa infra-estrutura a um canal simples. No *canão IP* este tipo de canal chama-se uma ***subnet IP*** ou sub-rede IP.

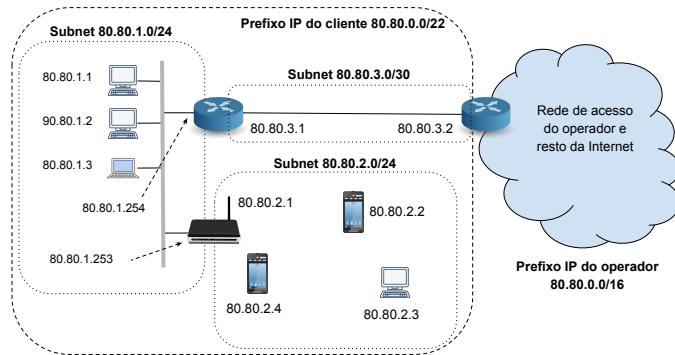


Figura 17.9: Plano de endereçamento de uma rede IP de um cliente ligado a um operador Internet

A Figura 17.9 mostra o plano de endereçamento de uma rede de uma instituição ligada à Internet via um operador (um ISP). O ISP recebeu da RIR da sua região o prefixo IP 80.80.0.0/16. Deste afetou ao cliente o sub-prefixo 80.80.0.0/22, que cobre os endereços de 80.80.0.0 a 80.80.3.255, para sua utilização interna. Como o cliente não está ligado a outro ISP, o seu espaço de endereçamento é *provider dependent address space* e só o prefixo do operador é anunciado na Internet global.

Internamente o cliente tem 3 *subnets*. Uma correspondente ao canal ponto-a-ponto que liga o seu *router* ao *router* do ISP, com o prefixo 80.80.3.0/30. O prefixo comporta 4 endereços mas só dois deles são usados para numerar as interfaces dos dois *routers*. A segunda *subnet* corresponde a um canal multi-ponto baseado numa infra-estrutura Ethernet que recebeu o prefixo 80.80.1.0/24. Todos os computadores e *routers* ligados a essa *subnet* têm endereços distintos nesse prefixo. Repare-se que os equipamentos com duas interfaces (o *router* e um *access point* de uma rede WiFi) têm mais do que um endereço IP distinto, cada um em *subnets* distintas e portanto em prefixos distintos. A terceira e última *subnet* corresponde à rede WiFi e tem o prefixo 80.80.2.0/24. O *access point* tem a interface rádio, que dá acesso ao canal WiFi, com um endereço neste prefixo. O cliente ficou com o prefixo 80.80.0.0/24 livre assim como a parte restante do prefixo 80.80.3.0/24, de 80.80.3.4 a 80.80.3.255.

Este plano de numeração vai repercutir-se nas tabelas de encaminhamento dos diferentes equipamentos ligados a esta rede. Por exemplo, a tabela de encaminhamento do computador com o endereço 80.80.1.3 na *subnet* da esquerda é apresentada na Figura 17.10.

ao canal.

Prefixo IP	Endereço do gateway (<i>next hop</i>)	Interface	Número de <i>hops</i>
80.80.1.3/32	local	eth0	0
80.80.1.0/24	directo	eth0	0
80.80.2.0/24	80.80.1.253	eth0	1
80.80.3.0/30	80.80.1.254	eth0	1
0.0.0.0/0	80.80.1.254	eth0	
...

Figura 17.10: Tabela de encaminhamento do computador com o endereço 80.80.1.3

A tabela de encaminhamento dos computadores pode ser consultada dando o comando `netstat -r` na consola ou na linha de comando da grande maioria dos sistemas de operação.

Esta tabela contém entradas que permitem enviar pacotes para o próprio computador via a sua interface Ethernet (a primeira entrada com o prefixo 80.80.1.3/32). A segunda entrada, indexada pelo prefixo do canal (80.80.1.0/24), permite enviar pacotes directamente para equipamentos ligados ao mesmo canal de difusão que o próprio computador. Esta forma de encaminhamento chama-se **encaminhamento directo** (*direct routing*).

As duas entradas seguintes, com os prefixos (80.80.2.0/24 e 80.80.3.0/30) permitem enviar pacotes para as outras *subnets* do cliente visto que são indexadas pelos respectivos prefixos. Estas entradas contém o endereço do *gateway / next-hop* que é o endereço de um *router*, acessível por um canal ligado a este computador, que “sabe” encaminhar para as *subnets* com aqueles prefixos. Esta forma de encaminhamento chama-se **encaminhamento indirecto** (*indirect routing*).

Finalmente, a última entrada (indexada pelo prefixo 0.0.0.0.0/0 ou universo) corresponde uma forma especial de encaminhamento indirecto que se chama **encaminhamento por defeito ou omissão** (*default routing*). Esta entrada permite encaminhar pacotes cujo endereço de destino é qualquer outro.

O processamento de um pacote por um computador ou um *router* IPv4 consiste, grosso modo, nas operações indicadas no algoritmo 17.1. Os métodos `packet.*_route()`, após seleccionarem a interface pela qual o pacote deve ser transmitido, deverão implementar a fragmentação se a mesma for necessária, decrementar o TTL e recalcular o *checksum*. O encaminhamento realizado por cada um é descrito a seguir.

Em IPv6 o processamento é grosso modo idêntico excepto que não há teste do *checksum*, o processamento das opções é transformado no processamento dos cabeçalhos opcionais, e se for necessária fragmentação, o pacote é ignorado e desencadeia-se um processamento de erro.

Encaminhamento local

O encaminhamento local é uma excepção pois o pacote é de novo entregue ao software IP local para processamento de um pacote a ele destinado. O mesmo acontece com os pacotes dirigidos ao endereço *loopback*.

Encaminhamento directo

Este tipo de encaminhamento tem lugar quando o endereço de destino do pacote está contido no prefixo IP associado a um canal ligado ao equipamento. Na rede que já foi introduzida mais acima, ver a Figura 17.11, é o caso quando o computador com o

Listing 17.1: Pseudo código a alto nível do algoritmo de encaminhamento de um pacote IP (*packet*)

```

1 if ( ! packet.checksum_ok() ) { // ignore packet
2     packet.process_error()
3     return
4 }
5 if ( packet.getTTL() - 1 == 0 ) { // end of packet life
6     packet.process_error()
7     return
8 }
9 if ( packet.header_length() > 5 ) { // packet has options
10    packet.process_options()
11    return
12 }
13 d = packet.getDestAddress()
14 r = routing_table.get_longest_prefix_match( d )
15 if ( r == null ) { // cannot route packet
16     packet.process_error()
17     return
18 }
19 if ( r.type() == local ) packet.locally_route(r, d)
20 else if ( entry.type() == direct ) packet.directly_route(r, d)
21 else if ( entry.type() == indirect ) packet.indirectly_route(r, d)

```

endereço IP 80.80.1.3 pretende enviar um pacote para o computador com o endereço 80.80.1.1.

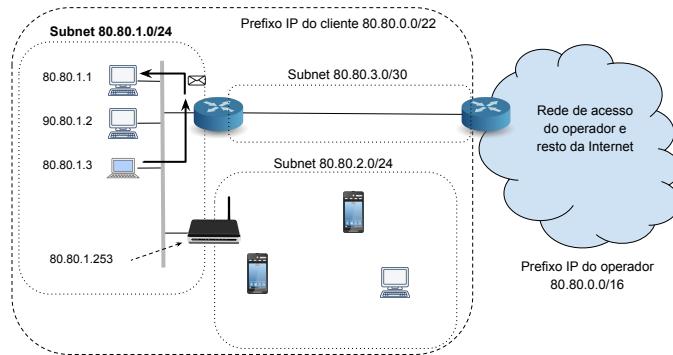


Figura 17.11: Envio de um pacote do computador com o endereço 80.80.1.3 para o com o endereço 80.80.1.1 através de encaminhamento directo.

Com o encaminhamento directo, se o canal é ponto-a-ponto basta transmitir o pacote para a outra extremidade do canal encapsulado num *frame* adequado. Se o canal é em difusão (e.g., Ethernet com fios ou WiFi) ou uma rede baseada em inundação, i.e., uma infra-estrutura de comunicação multi-ponto, (e.g., switched Ethernet) o pacote deve ser transmitido directamente encapsulado num *frame* dirigido ao endereço MAC do dispositivo ao qual está associado o endereço IP de destino. A Figura 17.12 mostra, a título de exemplo, a entrada (a) que é a seleccionada na tabela de encaminhamento para envio directo de um pacote do computador de endereço IP 80.80.1.3 para o computador com o endereço 80.80.1.1 usando o algoritmo *longest prefix matching* pois o endereço está contido em vários dos prefixes da tabela (nomeadamente o (a) e o (c)).

Para realizar este envio directo via um canal em difusão ou uma infra-estrutura

	Prefixo IP	Endereço do gateway (next hop)	Interface	Número de hops
	80.80.1.3/32	local	eth0	0
(a)	80.80.1.0/24	directo	eth0	0
(b)	80.80.2.0/24	80.80.1.253	eth0	1
	80.80.3.0/30	80.80.1.254	eth0	1
(c)	0.0.0.0/0	80.80.1.254	eth0	

Figura 17.12: Entrada na tabela de encaminhamento do computador de endereço 80.80.1.3 que é selecionada por *longest prefix matching* para envio de pacotes: para (a) 80.80.1.1 por envio directo; (b) para 80.80.2.3 por envio indirecto; e (c) para 193.136.126.43 por envio por *default*.

de comunicação multi-ponto, o software IP e de controlo do canal têm de conhecer qual é o endereço MAC da interface associada ao endereço IP do destino. É possível resolver este problema de tradução de endereços de várias formas, mas a implementação corrente para este tipo de canais é usar o protocolo *Address Resolution Protocol* (ARP) que será explicado a seguir.

Encaminhamento indirecto

Para envio de um pacote para o endereço 80.80.2.3 a entrada seleccionada pelo algoritmo *longest prefix matching* é a (b) da Figura 17.12, pois o endereço está contido em vários dos prefixes da tabela (nomeadamente o (b) e o (c)). Esta entrada indica como *next hop* o endereço da interface do equipamento onde começa o caminho para o destino. Este endereço tem de estar directamente acessível por encaminhamento directo, *i.e.*, acessível directamente por um canal ao qual o emissor do pacote esteja ligado. Tal é o caso pois o endereço 80.80.1.253 está no mesmo prefixo que um dos canais do emissor, logo é acessível por encaminhamento directo. Assim, o emissor encapsula o pacote num *frame* Ethernet e envia-o para o endereço MAC do *gateway*, *i.e.*, o endereço MAC associado ao endereço 80.80.1.253 do AP. Este endereço MAC pode-se obter via o protocolo ARP caso o mesmo não seja ainda conhecido pelo emissor.

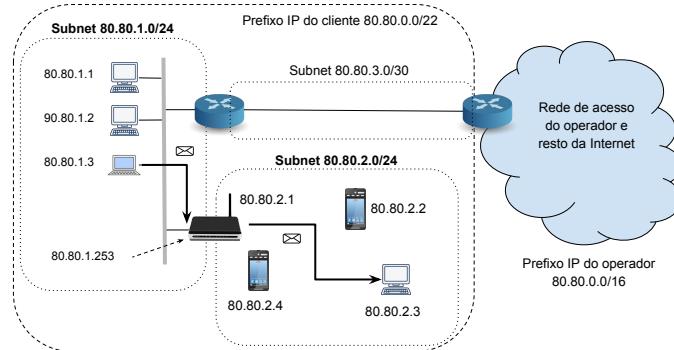


Figura 17.13: Envio de um pacote do endereço 80.80.1.3 para o endereço 80.80.2.3 através de encaminhamento indirecto.

Depois de o pacote chegar à interface com o endereço 80.80.1.253, o software IP do AP aplica o algoritmo 17.1 ao destino do pacote e concluirá que o destino é

acessível localmente por encaminhamento directo. Assim, retransmite-o encapsulado num *frame* Ethernet dirigido ao endereço MAC do computador com o endereço IP 80.80.2.3.

Encaminhamento por omissão ou *default*

Para envio de um pacote para o endereço 193.136.126.43, a entrada seleccionada pelo algoritmo *longest prefix matching* é a (c) na Figura 17.12 pois o endereço só está contido nesse prefixo. Este encaminhamento indirecto especial diz-se por omissão ou *default routing* pois este prefixo é uma espécie de encaminhamento pela via do “último recurso”. Na verdade o mesmo pressupõe que o *router* intermediário tem possibilidades de chegar a todos os destinos válidos.

Apesar de o prefixo envolvido ser especial, trata-se de encaminhamento indirecto convencional e a entrada tem de conter um endereço de *next-hop / gateway* directamente acessível. Tal é efectivamente o caso do endereço 80.80.1.254 associado ao *router* de fronteira do cliente. O pacote é encapsulado num *frame* Ethernet dirigido ao endereço MAC associado ao endereço IP 80.80.1.254 e o pacote é transmitido para esse *router*, ver a Figura 17.14.

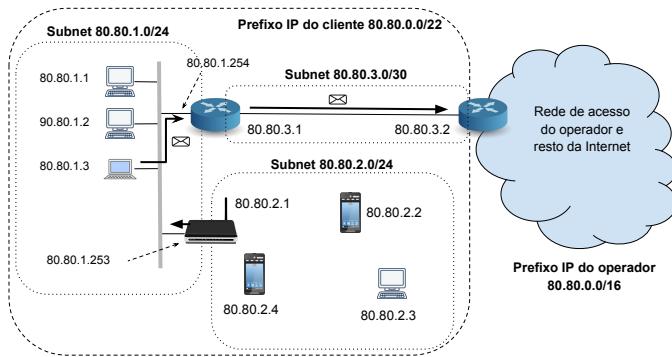


Figura 17.14: Envio de um pacote do endereço 80.80.1.3 para o endereço 193.136.126.43 através de encaminhamento por *default*.

Uma vez o pacote chegado ao *router* este consultará a sua tabela de encaminhamento, apresentada na Figura 17.15, e constatará que o seu endereço de destino apenas faz *matching* com a entrada correspondente à rota *default*, que indica que o *next-hop* está no endereço 80.80.3.2, acessível directamente pela interface do canal que liga o cliente ao ISP.

Prefixo IP	Endereço do gateway (<i>next hop</i>)	Interface	Número de hops
80.80.1.254/32	local	eth0	0
80.80.1.0/24	directo	eth0	0
80.80.2.0/24	80.80.1.253	eth0	1
80.80.3.1/32	local	eth1	0
80.80.3.0/30	directo	eth1	0
0.0.0.0/0	80.80.3.2	eth1	

Figura 17.15: Tabela de encaminhamento (FIB) do *router* de fronteira do cliente

Este exemplo mostra toda a potência do mecanismo dos prefixos IP e do algoritmo *longest prefix matching*. Em particular, fica evidente que o encaminhamento é profundamente hierárquico e que os diferentes equipamentos apenas necessitam de tabelas de encaminhamento com um número de entradas proporcional ao número de *subnets* distintas que têm de conhecer de forma discriminada. O prefixo 0.0.0.0/0, associado ao encaminhamento por omissão, representa de forma comprimida “o resto do mundo”.

Naturalmente, no interior da rede do ISP o número de prefixos conhecidos será muito maior pois as redes de acesso têm de conhecer os prefixos de todos os clientes a ela ligados. Adicionalmente, se o ISP for *multi-homed*, o seu *backbone* terá de conhecer todos os prefixos visíveis no *core* da Internet, pois nesse caso não é comum existirem rotas por omissão.

A região da Internet onde já não são usadas rotas por omissão, *i.e.*, na região de interligação dos operadores, é designada por esse motivo *Default-Free Zone* (DFZ). As tabelas de encaminhamento dos *routers* desta zona têm de contar todos os prefixos usados na Internet, e se o endereço de destino de um pacote não está contido em nenhum, então o pacote não pode ser encaminhado.

17.4 Protocolos auxiliares do IP

Para que uma rede IP funcione, este protocolo é complementado com vários protocolos auxiliares de descoberta de vizinhos e envio de informação de controlo e meta-information: ARP, ICMP e DHCP. No final da secção discute-se ainda o conceito de endereços privados e o seu papel.

Address Resolution Protocol (ARP)

O software IP dos sistemas de operação dos computadores e dos *routers* dispõe de uma tabela, chamada **tabela ARP (ARP table)**, que mapeia endereços IP em endereços MAC. Quando se procura o endereço MAC correspondente a um endereço IP e o mesmo não está presente nessa tabela, é usado o protocolo ARP para o obter. Este protocolo comprehende apenas duas mensagens: *ARP query* e *ARP reply*. A mensagem *ARP query* é enviada em difusão pelo canal (por isso o ARP só funciona em canais ou infra-estruturas com suporte de *broadcasting*). Essa interrogação é realizada enviando um *frame ARP query* com a pergunta: Who has IP address x, asks host with IP address y? Esta questão é difundida a todos os sistemas ligados ao canal, e se existir um com aquele endereço IP, este responde através do envio de um *frame ARP reply* dirigido ao endereço MAC origem da interrogação, ver a Figura 17.16. Ou seja, a interrogação é enviada em *broadcast* e a resposta em *unicast*.

Naturalmente, sempre que obtém o endereço MAC associado a um endereço IP via o protocolo ARP, o software IP coloca-o na tabela ARP para não a ter de pedir sempre que esta é necessária. Para evitar que a tabela tenha informação desactualizada, a cada entrada está associado um alarme de alguns segundos e, quando esse tempo se esgota, a entrada é suprimida da tabela para evitar conter informação que provavelmente já não seria válida. Isto é, o protocolo não comporta nenhuma mensagem para invalidar associações entre endereços IP e endereços MAC e a gestão da tabela apenas garante que a informação nela contida era válida quando lá foi colocada. A informação contida na tabela ARP diz-se então ser *soft state*⁸.

Na maioria dos sistemas de operação a tabela de ARP é acessível através de comandos na consola, como por exemplo `arp -a` nos sistemas derivados de Unix (Linux,

⁸ O que sugere que o termo *hard state* é reservado para situações em que a informação é permanentemente mantida de forma consistente, o que se revela geralmente mais complexo e com custos acrescidos.

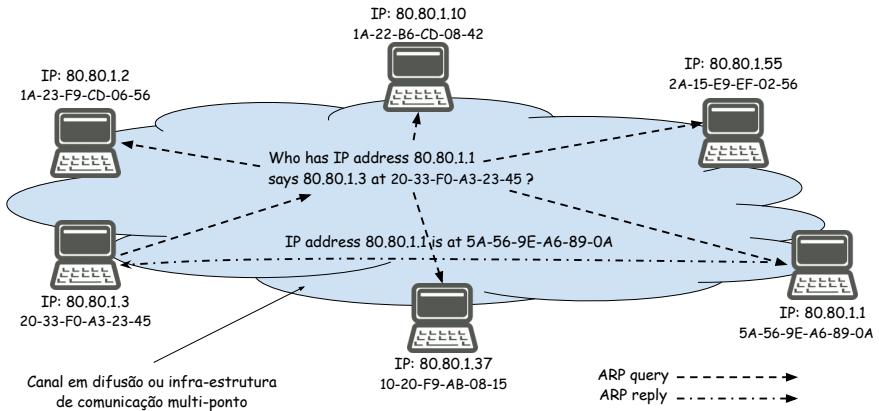


Figura 17.16: Funcionamento do protocolo ARP quando o computador com endereço 80.80.1.3 pretende descobrir o endereço MAC do computador com o endereço IP 80.80.1.1

Mac OS X, iOS, Android, ...). Em alguns destes sistemas as tabelas de encaminhamento e a tabela de ARP estão fundidas na mesma tabela. As implementações de ARP também utilizam as *ARP queries* para enriquecerem a tabela de ARP com os endereços IP e MAC do computador que emite a interrogação, visto que ambas as informações são acessíveis no *frame ARP query* que é recebido por todos os sistemas ligados ao canal multi-ponto.

O protocolo ARP não comporta nenhuma forma de autenticação, pode ser facilmente violado, e levar à propagação de informação errada. Basta, por exemplo, que um computador emita *frames ARP query* com endereços IP origem falsos. Por esta razão, este protocolo não é adequado a situações onde o aparecimento deste tipo de ataques seja frequente e possa resultar em consequências graves. Trata-se de mais um protocolo desenvolvido para *subnets* com baixa probabilidade de presença de intrusos.

Dynamic Host Configuration Protocol (DHCP)

Como vimos nos exemplos anteriores, um computador para poder comunicar em IP necessita de obter endereço(s) IP e preencher a sua tabela de encaminhamento. O mesmo problema se coloca quando se trata de parametrizar os *routers* da uma rede.

No início da Internet essa parametrização era realizada de forma manual. Quando se ligava mais um computador a uma rede IP era necessário saber qual o seu endereço IP, o prefixo IP do canal de ligação à rede, o endereço do *router* que representava a rota por omissão e os endereços de um ou mais servidores de DNS.

Mais tarde este processo de inicialização do software IP foi automatizado através do protocolo *Boot Protocol* que foi depois substituído pelo *Dynamic Host Configuration Protocol* ou DHCP, ver o RFC 1541. O mesmo é hoje em dia imprescindível pois, sempre que um computador móvel se liga a uma rede diferente, ou muda de rede, o software IP do mesmo tem de ser reinicializado.

Os *routers* que ligam as redes domésticas aos ISPs também utilizam o protocolo DHCP para obterem as informações necessárias ao seu correcto funcionamento. No entanto, muitos equipamentos de rede dos *backbones* continuam a ser parametrizados manualmente.

Quando um computador activa a interface de um canal que o liga à rede, esse canal é quase sempre um canal multi-ponto, ou um canal que o liga a uma infra-estrutura equiparada. Por essa via é pressuposto ser possível enviar pacotes para um

ou mais **servidores DHCP**. O protocolo comporta 4 mensagens: **Discovery**, **Offer**, **Request** e **Acknowledge**. De forma muito resumida, o cliente começa por tentar localizar um servidor DHCP enviando em difusão uma mensagem do tipo **Discovery**. Os servidores DHCP acessíveis deverão responder com mensagens **Offer**, propondo um endereço IP ao cliente e enviando outras informações relevantes. O cliente escolhe uma das ofertas de endereço recebidas e envia uma mensagem **Request** indicando o endereço escolhido. Por razões que ficarão mais claras a seguir, esta mensagem também é enviada em difusão. O servidor, caso confirme a oferta, deve enviar uma mensagem **Acknowledge** que confirma a afectação. A Figura 17.17 ilustra o processo.

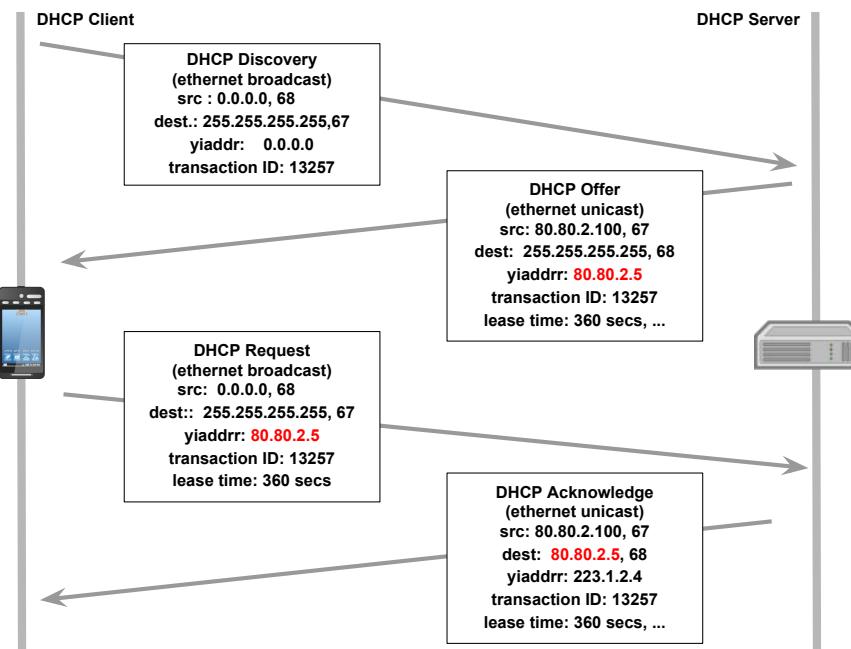


Figura 17.17: Funcionamento do protocolo DHCP

As mensagens DHCP são enviadas sobre UDP. A porta do cliente é a porta 68 e a porta do servidor é a 67. A mensagem **Discovery** tem por endereço origem o endereço indefinido ou 0 (0.0.0.0) pois o cliente ainda não conhece o seu endereço. O endereço de destino é o endereço de *broadcast* (255.255.255.255). A mensagem pode ser enviada mesmo sem o cliente conhecer o seu endereço IP porque a sua interface conhece o respectivo endereço MAC (afectado na fábrica) e o destino é o endereço de difusão, quer ao nível MAC, quer ao nível IP. O parâmetro **Transaction ID** é colocado pelo cliente para lhe permitir relacionar as respostas com os pedidos de forma mais segura. O parâmetro **yiaddr** (*Your IP Address*) é onde o servidor pode colocar o endereço oferecido ao cliente. No caso da mensagem inicial o seu valor é 0 (0.0.0.0).

O servidor responde com uma mensagem **Offer** propondo um endereço IP e enviando outros parâmetros úteis ao cliente (*e.g.*, prefixos IP, endereços de servidores DNS, rotas por omissão, *etc.*). Adicionalmente, o servidor envia um tempo de validade da oferta, este parâmetro é designado por *lease time*. A resposta é enviada em *unicast* Ethernet.

Como o cliente pode receber ofertas de vários servidores, como por exemplo um primário e outro de *backup*, o cliente deve escolher uma das ofertas e reafirmá-la envi-

ando uma mensagem **Request** em difusão, o que permitirá aos servidores não escolhidos cancelarem as suas ofertas. Tudo termina com uma mensagem **Acknowledgment** enviada em *unicast* pelo servidor escolhido ao cliente DHCP. Caso se percam mensagens, compete ao cliente reemitir os pedidos.

O protocolo DHCP comporta uma mensagem **Release** mas o seu envio pelo cliente é opcional pelas razões indicadas a seguir. O parâmetro *lease time* permite ao servidor reutilizar o endereço se este período expirar e o cliente não renovar o pedido, o que pode fazer enviando uma mensagem de **Request** a renovar o interesse na afectação do endereço.

Este simples mecanismo evita que o servidor se tenha de preocupar com o desaparecimento abrupto do cliente por este ter tido uma avaria ou pura e simplesmente ter sido desligado. Trata-se de outro exemplo de informação gerida por *soft state*, que é simples mas não dá garantias de validade. Basta pensar no caso em que o servidor DHCP falha e é reinicializado, ou substituído por outro, antes de os *lease time* das afectações anteriores se tenham esgotado. Uma solução para este problema é proposta em exercícios no fim do capítulo, ver a Secção 17.6.

O protocolo admite inúmeras opções introduzidas para permitir transmitir aos clientes as mais variadas informações. Em contrapartida, o DHCP não usa nenhuma mecanismo de segurança. Isso permite inúmeros ataques como por exemplo os baseados em servidores DHCP a enviarem informações falsas, clientes a fazerem ataques de negação de serviço para esgotarem os endereços disponíveis e clientes que usam endereços que não lhes foram afectados. Para combater estes problemas foram introduzidas variantes com mecanismos de segurança mas as mesmas não foram adoptadas de forma generalizada. Quando o DHCP é usado entre os ISPs e os seus clientes, são usados mecanismos complementares de autenticação dos clientes a outros níveis.

Para terminarmos esta discussão de protocolos de apoio ao funcionamento do IP, vamos a seguir analisar o protocolo ICMP.

Internet Control Message Protocol (ICMP)

Por opção conceptual, ver a Secção 4.1, o protocolo IP não obriga estritamente os *routers* IP a assinalarem aos seus parceiros e outros sistemas ligados à rede quando acontecem eventos que conduzem a erros ou outras situações equiparáveis (*e.g.*, TTLs esgotados, pacotes suprimidos por as filas de espera estarem cheias, destinos desconhecidos, pacotes mal formados, *etc.*).

No entanto, existe um protocolo chamado *Internet Control Message Protocol (ICMP)*, definido inicialmente pelo RFC 792 e referido no RFC 1122 e outros, que permite enviar notificações de erro. Adicionalmente, o ICMP permite igualmente troca de (meta-)informação entre equipamentos IP. As mensagens ICMP são transportadas directamente dentro de pacotes IP, são enviadas para o endereço IP do dispositivo que originou o erro, ou fez o pedido de informação, e não são endereçadas a portas pois devem ser tratadas directamente pelo software do nível IP. A Figura 17.18 mostra a parte comum do cabeçalho ICMP.

O campo **Type** organiza as mensagens ICMP em diversos tipos e o campo **Code** em variantes dentro dos tipos. Conforme o tipo das mensagens, o resto do cabeçalho pode variar e a parte de dados também. A tabela 17.3 lista alguns exemplos de tipos e códigos ICMP.

É fácil reconhecer o papel de alguns dos tipos de mensagens ICMP. Por exemplo, o tipo 8 (*Echo request*) e o tipo 1 (*Echo reply*) podem ser usados pelo programa **ping**, ver a Secção 3.3. O funcionamento do programa **traceroute**, ver a Secção 3.4, repousa nas mensagens ICMP de tipo 11 (*Time Exceeded - TTL expired in transit*). A mensagem de tipo 3, código 4 (*Fragmentation required, and DF flag set*) permite saber que era necessário realizar a fragmentação de um pacote mas que, dado o posicionamento de uma *flag* específica no cabeçalho do pacote, o *router* suprimiu-o e enviou uma

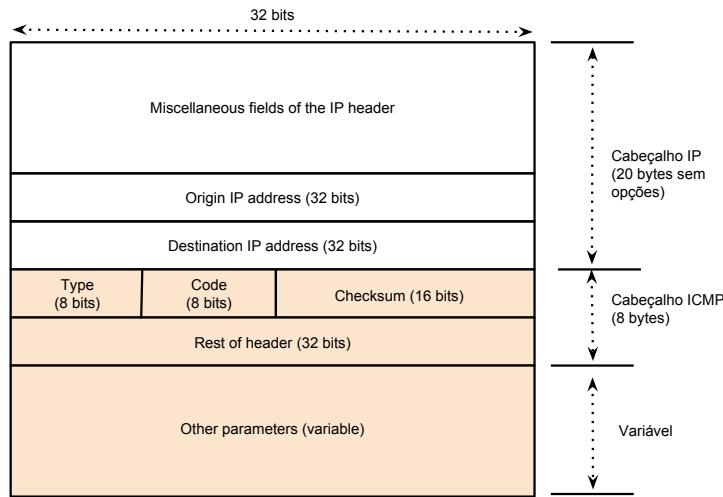


Figura 17.18: Formato das mensagens do protocolo ICMPv4

notificação ICMP. Este tipo e código estão na base do protocolo *Path MTU discovery*, ver o RFC 1191.

Algumas mensagens ICMP servem para controlo. Por exemplo, as mensagens com os tipos 13 e 14 (*Time request / Reply*) podem ser usadas para medir o tempo de trânsito com maior rigor entre dois equipamentos e os tipos 11 e 10 (*Router discovery / advertisement*) podem ser usados para um sistema terminal descobrir um *router* no canal a que está ligado e instalar uma rota por omissão.

A definição exacta do papel de cada mensagem e da forma como estas devem ser processadas pelo emissor e pelo receptor é objecto de descrição muito detalhada nos RFCs que lhes dizem respeito. O leitor deverá consultar esses documentos para conhecer os detalhes. Um aspecto comum a um grande conjunto de tipos de mensagens é o facto de que a sua emissão é opcional, *i.e.*, se um *router* suprime um pacote por o TTL ter expirado, o envio da mensagem ICMP 11 é opcional. Na realidade muitos *routers* estão preparados pelos seus administradores para ignorarem certas mensagens ICMP ou para não enviarem respostas às mesmas. Esta possibilidade é acomodada pelo carácter *best effort* do IP, ver a Secção 4.1. Muitas vezes a motivação para parametrizar os *routers* para não enviarem ou responderem a algumas mensagens ICMP tem a ver com razões de desempenho e segurança.

Muitas das mensagens ICMP têm cabeçalhos complementares e transportam informação que permite ao receptor relacionar uma resposta com um pedido. Por exemplo, as mensagens *Echo Request / Reply* têm campos para serem colocados números de sequência que suportam envios sucessivos e também é pressuposto que a parte de dados do *request* seja copiado para a parte de dados do *reply*. As mensagens ICMP com notificações de erros (*e.g., Host unreachable*) devem conter na parte de dados o cabeçalho do pacote IP que deu origem ao problema e os primeiros 8 bytes do respectivo *payload*.

ARP, ICMP e DHCP em IPv6

Os protocolos ARP, ICMP e DHCP em IPv6 foram introduzidos ao longo dos anos à medida que a experiência e as necessidades de funcionamento e parametrização do IPv6 iam crescendo. Por esta razão existem algumas sobreposições entre esses protocolos e

Tabela 17.3: Alguns exemplos de tipos e códigos do ICMP

Type	Code	Descrição
1	0	Echo reply
3	0	Destination unreachable - Network unreachable
3	1	Destination unreachable - Host unreachable
3	3	Destination unreachable - Port unreachable
3	4	Destination unreachable - Fragmentation required, and DF flag set
3	5	Destination unreachable - Source route failed
8	0	Echo request
9	0	Router advertisement
10	0	Router discovery/selection/solicitation
11	0	Time Exceeded - TTL expired in transit
11	1	Time Exceeded - Fragment reassembly time exceeded
13	0	Time request
14	0	Time reply

vários deles dependem da disponibilidade directa de *broadcasting* no canal. Em IPv6, usando a experiência anterior, foi possível passar a limpo a definição destes protocolos.

O suporte da comunicação multi-ponto em IPv6 foi restringido à funcionalidade de IPv6 *multicasting* (não existe *broadcasting*). A definição de IPv6 *multicasting* comporta a definição de grupos pré-definidos (tal como o IPv4 *multicasting*) como por exemplo todos os nós acessíveis por um canal (ou infra-estrutura) multi-ponto, todos os *routers* ligados a um canal (ou infra-estrutura) multi-ponto, destes, todos os com suporte de OSPF, etc.

Baseado nas funcionalidades do IPv6 *multicasting*, o IPv6 acrescentou ao protocolo ICMPv6, como descrito no RFC 2461, um subconjunto de mensagens que servem para *Neighbor Discovery* (ND) e que integram as funcionalidades dos protocolos ARP e parte das do DHCP. Parte das funcionalidades de ND implementadas pelo ICMPv6 são as seguintes: *Router Discovery*, *Prefix Discovery*, *Parameter Discovery*, *Address Resolution*, *Duplicate Address Resolution*, *Redirect*, etc..

Com estas funcionalidades, o ICMPv6 substitui integralmente o ARP e permite que uma interface se auto-configure correctamente sem recurso a nenhum servidor DHCP usando *stateless auto-configuration*: conhecendo o prefixo da *subnet* do canal é possível gerar um endereço IPv6 usando o endereço MAC da interface como sufixo (os últimos 48 bits dos 128 bits do endereço). Apesar disso, o IPv6 também dispõe do DHCPv6, ver o RFC3315 e outros, que permite afectar endereços e outros parâmetros fornecidos por um servidor.

Não existe um fosso conceptual entre a operação destes protocolos em IPv4 e em IPv6, mas existem bastantes diferenças pois em IPv6 pretendeu-se favorecer a parametrização dinâmica de redes sem recurso a servidores e com pouca intervenção dos gestores da mesma. Por essa razão é preferível o leitor consultar documentação especializada como por exemplo as referências incluídas na Secção 17.5. Para terminar esta secção introduzem-se a seguir os endereços privados e a tradução de endereços.

Endereços privados e tradução de endereços

Existem diversas razões que tornam desejável implementar uma rede cujo espaço de endereçamento seja privado, i.e., de alguma forma não visível do exterior da mesma. Historicamente esta solução teve como principal motivação poupar endereços IPv4. A ideia era permitir que vários computadores conseguissem partilhar entre si um único endereço IPv4 ligado à Internet. Definiu-se então o conceito de **espaço de endereçamento privado** já referido neste capítulo, (ver a Secção 17.1), ou seja, prefixos que não podem ser encaminhados na Internet global, e que podem ser reutilizados

entre diferentes redes privadas pois não são visíveis fora dessas redes. Uma tal solução requer alguma forma de **tradução entre endereços privados e públicos**, i.e., entre endereços privados e os endereços regulares, sob pena de os sistemas internos não poderem comunicar com o exterior da rede.

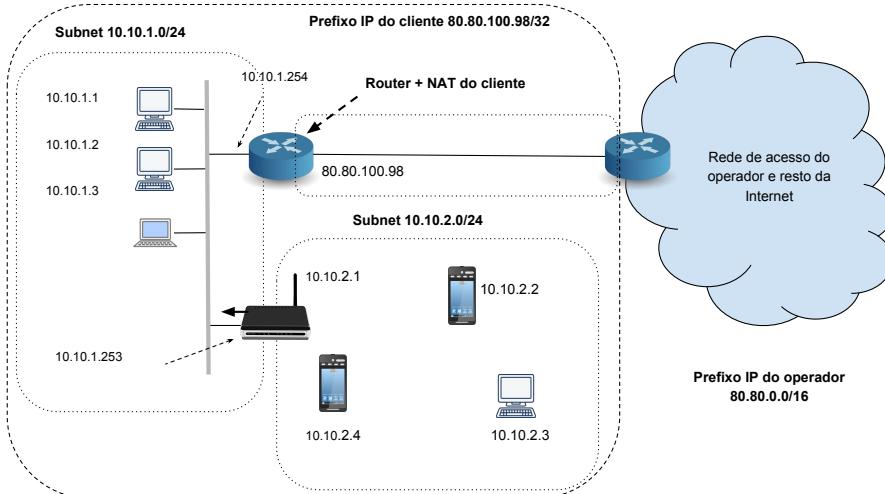


Figura 17.19: Rede de um cliente baseada em endereçamento privado

A Figura 17.19 apresenta a já nossa conhecida rede de um cliente mas que desta vez utiliza o prefixo privado $10.0.0.0/8$ para numerar as suas *subnets* internas. O *router* que liga o cliente ao ISP tem uma interface ligada à rede privada e outra interface ligada ao ISP. O ISP desta vez apenas atribui um único endereço público ao cliente, exactamente o endereço da interface do *router* do cliente que termina a ligação deste à rede de acesso do ISP.

Uma outra motivação para usar esta solução tem a ver com o facto de que ela facilita a mudança de ISP. Com efeito, se o cliente não tem espaço de endereçamento dependente de um ISP, se trocar de fornecedor, as alterações estão confinadas ao seu *router* de fronteira assinalado na figura com a sigla NAT de *Network Address Translation*. Toda a sua infra-estrutura interna está isolada e não requer alterações. A terceira motivação só fica mais clara depois de vermos como é que a tradução de endereços pode funcionar.

Parece relativamente simples o *router* do cliente trocar cada endereço interno por um externo. No entanto, esta solução exige tantos endereços públicos quantos os privados. O que é necessário é conseguir utilizar um único endereço IP público para representar vários endereços privados em simultâneo. Isso consegue-se recorrendo a uma porta extra por cada interação (transação) entre um computador da rede privada e um computador na rede pública. A função de tradução de endereços é designada pela sigla NAT (*Network Address Translation*) e o mesmo mecanismo recorrendo a portas, é designado pela sigla NATP (*Network Address Translation with Ports*).

Para percebermos como o mecanismo pode funcionar vamos imaginar que o computador com o endereço IP privado 10.10.1.3 pretende abrir uma conexão TCP para o servidor HTTP no endereço IP público 193.136.126.43. Para tal envia um pacote, ver a Figura 17.20 (a), para o servidor. Seguindo a rota por omissão este chega ao *router* do cliente que então reescreve o cabeçalho do pacote alterando o endereço origem para o seu, e a porta origem para uma que afecta a esta conexão. Finalmente regista esta informação (antigo endereço e porta / novo endereço e porta) na sua tabela de

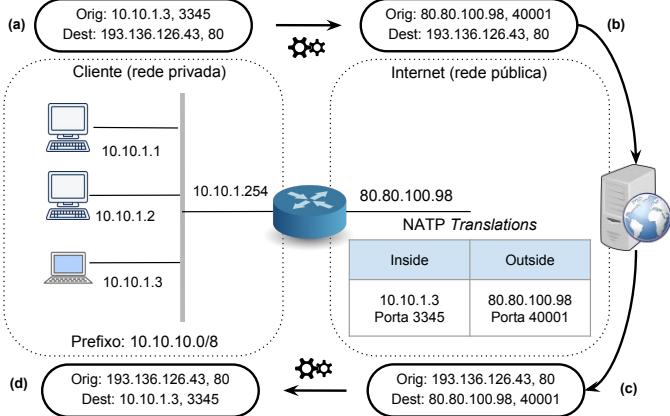


Figura 17.20: Tradução de endereços com base em portas (NATP)

NATP *translations* e transmite o pacote com endereçamento modificado, ver a Figura 17.20 (b).

O servidor processa o pacote e responde com um novo pacote, ver a Figura 17.20 (c), dirigido ao endereço e porta do *router* NATP. Ao receber o pacote do servidor, através da porta o *router* consegue saber qual o endereço privado e porta para que o pacote tem de ser enviado, reescreve o cabeçalho e este chega ao computador com o endereço IP privado 10.10.1.3, ver a Figura 17.20 (d). Os outros pacotes enviados por ambas as partes sofrem exactamente a mesma transformação enquanto a conexão durar e a associação na tabela se mantiver.

O mecanismo funciona correctamente afectando uma porta diferente do *router* NATP para cada interacção (em UDP, em TCP, em ICMP, ...) iniciada por qualquer computador com endereço IP privado. Para que as entradas na tabela de *NATP translations* não fiquem lá eternamente, é preciso verificar quando as conexões TCP são fechadas, ou usar um alarme (*timeout*) para as interacções UDP. Como os pacotes ICMP não usam portas, o mecanismo é mais complexo para os mesmos. No final do capítulo são propostos exercícios sobre ambos estes problemas, ver a Secção 17.6.

Analizando o mecanismo é possível verificar que o mesmo só permite a existência de transações iniciadas na rede interna. Qualquer pacote que chegue ao *router* NATP para uma porta sem associação não pode entrar na rede privada. Adicionalmente, os *routers* NATP só utilizam para traduções as portas altas (por exemplo entre 40.000 e 60.000), e por outro, estes podem também memorizar para que endereço público é que o pacote associado à entrada foi enviado (o endereço de destino do pacote (b) no exemplo). Qualquer pacote dirigido à porta reservada pelo *router* à conexão, que não venha do endereço público associado (193.136.126.43 no exemplo) é deitado fora⁹. Desta forma a probabilidade de se conseguir enviar da rede pública para os computadores da rede privada, pacotes estranhos a interacções iniciadas na rede privada, fica bastante mais reduzida.

Esta é exactamente a terceira vantagem de se usarem mecanismos NATP pois estes introduzem uma barreira suplementar de segurança (ainda que ténue e às vezes enganadora). Com efeito, o mecanismo descrito implementa algo que está de acordo com o princípio da “separation of concerns”, ver a Secção 4.1, entre a rede privada e a rede pública e introduz o conceito de *gateway* de acesso ao espaço privado.

No entanto, pode acontecer que o cliente também quisesse ter um servidor HTTP

⁹ Este controlo suplementar também pode criar alguns inconvenientes e restrições quando se pretende implementar mecanismos de NAT *traversal*, como veremos a seguir.

na sua rede interna acessível para clientes na Internet pública. Como exposto, isso não é possível a não ser que se complemente o mecanismo de tradução dinâmica e a pedido com um mecanismo inverso mas permanente. Essa tradução no sentido inverso designa-se **redirecção de portas (port forwarding)** e funciona como se explica a seguir.

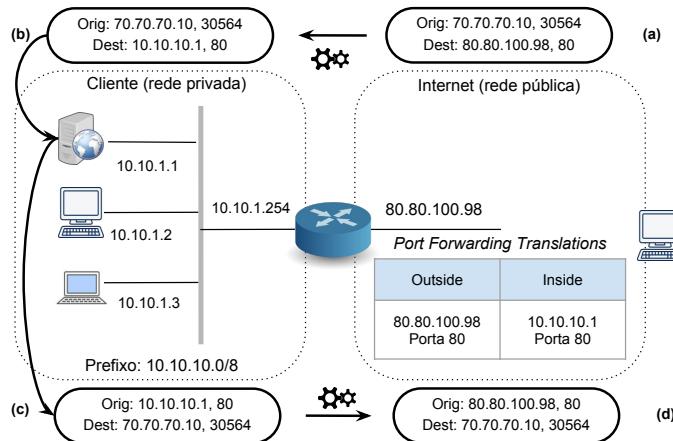


Figura 17.21: Acesso a um servidor interno com base em *port forwarding*

Numa tabela de traduções complementares pode-se associar um endereço privado interno de forma fixa a uma porta (e portanto um só endereço privado pode ser assim associado de forma fixa). Por exemplo, ver a Figura 17.21 (a), quando um cliente na Internet envia um pacote de abertura de uma conexão HTTP sobre TCP para a porta 80 do endereço público do cliente (endereço 80.80.100.98), o pacote chega ao *router* do cliente que reescreve o seu cabeçalho no endereço privado do servidor (endereço 10.10.1.1), ver a Figura 17.21 (b).

Quando o servidor responde ao pacote recebido, ver a Figura 17.21 (c), este é enviado através da rota por omissão e chega ao *router* NATP que reescreve de novo o cabeçalho, usando a regra fixa, e o envia ao cliente HTTP, ver a Figura 17.21 (d).

Desta forma, as portas associadas a serviços bem conhecidos podem ser associadas a servidores na rede privada que passam a receber os pacotes vindos do exterior e dirigidos às mesmas. Este mecanismo tem as vantagens indicadas e serve para servidores de serviços bem conhecidos, mas não pode ser usado para que um qualquer computador na rede privada possa receber conexões do exterior pois exige afectar portas a computadores de forma fixa.

O outro inconveniente da tradução de endereços tem a ver com o facto de que estes mecanismos não são transparentes para protocolos aplicacionais que necessitam de passar endereços privados no interior do *payload* dos pacotes. Esses endereços, ao serem recebidos pelos destinatários, tornam-se inúteis pois não estão associados a computadores acessíveis. Alguns dos protocolos populares que ficam assim inutilizados passaram a ser reconhecidos pelo software de alguns equipamentos de NAT mas, como é evidente, esta solução é frágil, particular e pouco flexível.

O NAT tornou-se tão popular em ambiente IPv4 que foram desenvolvidas e normalizadas várias soluções que permitem a computadores em redes públicas tomarem a iniciativa de comunicarem com outros em redes privadas, ou até permitir a comunicação quando ambos os computadores estão em redes privadas. Estes mecanismos designam-se genericamente por mecanismos de NAT *traversal*.

Essas soluções passam por dois tipos de mecanismos. Por um lado, um computador

tem de ser capaz de reconhecer se está ou não por detrás de um dispositivo de NAT. À primeira vista isso parece simples pois os endereços privados estão recenseados. No entanto, é mais geral arranjar um protocolo que permite a um cliente saber qual o endereço IP origem com que um servidor público recebe os seus pacotes. Se for diferente do seu, é porque está por detrás de NAT. Isto torna-se imprescindível quando o NAT está, por qualquer motivo, escondido do computador.

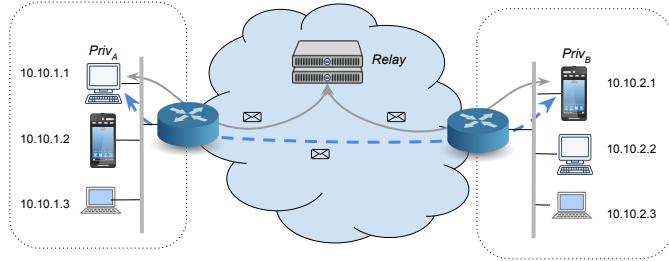


Figura 17.22: Comunicação entre dois dispositivos IP em redes privadas usando um servidor de *relaying*. A cheio o trajecto dos pacotes reais, a tracejado o trajecto lógico dos pacotes.

O outro mecanismo necessário é baseado num servidor de *relaying*, designado de agora em diante por *Relay*, ver a Figura 17.22. O computador na rede privada (*Priv_A*) inicia uma transação com *Relay* de forma a que este tenha um caminho aberto no dispositivo NAT para lhe chegar, *i.e.*, uma entrada na tabela de NATP *translations* associada ao seu endereço e com uma porta dedicada. Esta via entre *Priv_A* e *Relay* tem de ser periodicamente refreshada por iniciativa de *Priv_A* para se manter activa. O outro computador (*Priv_B*), fora da rede privada ou mesmo noutra rede privada, executa o mesmo protocolo com *Relay*. Finalmente, os dois computadores podem trocar pacotes via *Relay*, ver a Figura 17.22, que assegura uma via privada entre os dois e representa para cada um o outro interlocutor.

O mundo do NAT e do NAT *traversal* transformou-se num labirinto de soluções particulares pois as variantes dos controlos exercidos sobre a origem dos pacotes recebidos pelo dispositivo de NAT, e a necessidade de tradução de endereços nos pacotes e respectivo *payload* requerem inúmeras soluções particulares. O que é um claro indício de engenharia de má qualidade. Por outro lado, as soluções de NAT *traversal* são inúmeras e também dependem da solução NAT particular e do tipo das aplicações envolvidas. As soluções proprietárias para casos particulares são numerosas.

O IETF fez algum esforço na tentativa de normalizar soluções de NAT *traversal*, ver o RFC 5766.

Este estado de coisas teve origem no facto de que o NAT sempre foi visto como um subterfúgio para adiar a adopção de IPv6 por uma parte da comunidade Internet, o que fez com que o mesmo não fosse analisado, tratado e normalizado de forma sólida desde o início. Ao mesmo tempo a vida impôs a utilização maciça de NAT, incluindo pelos operadores com o chamado CGN (*Carrier Grade NAT*).

A generalização da utilização de IPv6 tornará a utilização de NAT inútil para vencer as restrições sobre a escassez de endereços. Quanto às outras vantagens indirectas do NAT, nomeadamente facilitar de forma simples a independência dos operadores sem aumentar o número de prefixos *provider independent* visíveis na Internet, e fornecer um primeiro mecanismo de isolamento e segurança, o mesmo já não é tão claro.

Todos os equipamentos com suporte de IP usam um conjunto de mecanismos obrigatórios e protocolos complementares (ARP, DHCP, ICMP, *etc.*) que permitem ao encaminho de pacotes IP funcionar na prática. Esses mecanismos têm no seu centro, como ideias construtoras, as noções de que a cada canal está associado um prefixo IP distinto, a ideia de que todas as interfaces IP para um canal têm um endereço IP no prefixo do canal, e a noção de tabela de encaminhamento como suporte do algoritmo *longest prefix match is the best*.

A tabela de encaminhamento de um computador, mesmo quando ligado a um só canal, discrimina as formas elementares do encaminhamento IP, nomeadamente: encaminhamento local, encaminhamento directo, encaminhamento indirecto e encaminhamento por omissão (por *default*). Estes mecanismos são suficientes para a ligação dos sistemas terminais e de redes simples ao mundo IP.

17.5 Resumo e referências

Resumo

Devido à escala da Internet, onde existem biliões de interfaces globalmente endereçáveis, e a necessidade de tornar o encaminhamento mais simples, os endereços IP são de carácter topológico e hierárquico, são afectados por prefixos relacionados com a localização das diversas redes constituintes da Internet e são depois geridos e afectados pelos gestores das mesmas.

Os diferentes prefixos IP existentes têm entre si relações hierárquicas. Todos os prefixos são sub-prefixos do universo, denotado 0.0.0.0/0, o prefixo de comprimento nulo. Diferentes sub-prefixos são depois afectados às redes que formam a Internet e sub-prefixos desses são depois afectados a redes subordinadas às mesmas.

Esta hierarquia não é estrita mas desempenha um papel fundamental no encaminhamento pois o encaminhamento no mundo IP consiste em encaminhar os pacotes para os prefixos a que pertencem. É esta característica que está por detrás da escalabilidade do encaminhamento na Internet pois as tabelas de encaminhamento (FIBs) contém prefixos IP e não endereços IP isolados.

O encaminhamento na Internet global, a chamada *Default-Free Zone*, é realizado com base nos prefixos IP anunciados pelas diferentes redes interligadas. Para evitar que uma hierarquização total dos prefixos redundasse num sistema demasiado rígido, e potencialmente com as características de um monopólio dos operadores Tier1, são anunciados prefixos correspondentes a todas as redes, mesmo de clientes finais, desde que estas tenham a necessidade de controlar de forma mais fina o encaminhamento dos pacotes que lhes são dirigidos.

Os prefixos afectados pelos operadores aos seus clientes fazem parte do espaço *provider dependent address space*. Esses clientes, se não forem *multi-homed*, estão num espaço de endereçamento que também se pode chamar *provider-based aggregation address space* porque poupa entradas nas tabelas de encaminhamento. Ao contrário, os prefixos das redes clientes mas independentes do fornecedor, têm prefixos no *provider independent address space* ou *without aggregation address space* e expandem necessariamente as tabelas de encaminhamento.

A regra *longest prefix matching is the best* na escolha das entradas das tabelas de encaminhamento dos comutadores funciona como garante de flexibilidade. O controlo fino do encaminhamento para uma rede, e as necessidades de *multi-homing* levam à subida incessante do número de prefixos IP visíveis no coração da Internet.

O protocolo IP é a “cola” ou “camada unificadora” da Internet pois permite que todos os equipamentos que o suportam possam trocar pacotes de dados entre si. O

cabeçalho IP na versão 4, a versão mais comum actualmente, para além dos endereços IP origem e destino do pacote, contém um conjunto de campos que apoiam o encaminhamento dos mesmos, nomeadamente nas facetas relacionadas com a prevenção de ciclos de encaminhamento (TTL) e qualidade de serviço.

Devido à falta de experiência e às características tecnológicas das redes quando o protocolo foi definido, o cabeçalho IPv4 inclui igualmente o suporte de mecanismos que se vieram a revelar pouco amadurecidos, como por exemplo a fragmentação, e o facto de o cabeçalho ser de comprimento variável devido à presença no mesmo de várias opções (*e.g., source routing, record route, etc.*). Na prática, as opções IPv4 pouco ou nada são suportadas e a fragmentação não é recomendada.

A versão 6 do protocolo IP introduz uma alteração fundamental ao nível da dimensão e formato dos endereços IP, e suprimiu alguns mecanismos que se revelaram inúteis ou contraproducentes em IPv4. Adicionalmente, substituiu as opções por uma eventual pilha de cabeçalhos de extensão, sempre que são necessárias funcionalidades opcionais semelhantes às opções do IPv4, ou ainda novas opções de segurança, *jumbograms, etc.* Essas opções são fundamentalmente *end-to-end* e pouco utilizadas, tal como em IPv4.

Todos os equipamentos com suporte de IP usam um conjunto de mecanismos obrigatórios e protocolos complementares (ARP, DHCP, ICMP, *etc.*) que permitem ao encaminhamento de pacotes IP funcionar na prática. Esses mecanismos têm no seu centro, como ideias construtoras, as noções de que a cada canal está associado um prefixo IP distinto, a ideia de que todas as interfaces IP para um canal têm um endereço IP no prefixo do canal, e a noção de tabela de encaminhamento como suporte do algoritmo *longest prefix match is the best*.

A tabela de encaminhamento de um computador, mesmo quando ligado a um só canal, discrimina as formas elementares do encaminhamento IP, nomeadamente: encaminhamento local, encaminhamento directo, encaminhamento indirecto e encaminhamento por omissão (por *default*). Estes mecanismos são suficientes para a ligação dos sistemas terminais e de redes simples ao mundo IP.

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

Principais conceitos

Endereço IP (*IP address*) Endereços afectados às interfaces dos dispositivos com suporte do protocolo IP que se caracterizam por serem afectados num espaço com carácter topológico e hierárquico.

Quad notation Notação usada para representar os endereços IP versão 4 usando 4 grupos de dígitos decimais separados por um ponto (Exemplo: 193.136.126.43).

Hexadecimal colon notation Notação usada para representar os endereços IP versão 6 usando 8 grupos de 4 dígitos hexadecimais separados por dois pontos (Exemplo: 2001:0:0:0:0:45bf:ff23:8b4f).

Prefixo IP (*IP prefix*) Conjunto de endereços IP contíguos que partilham entre si um conjunto de bits mais significativos. São representados pelo endereço IP correspondente aos bits do prefixo, seguidos de bits a zero, seguido de uma barra e do número do comprimento do prefixo (Exemplos: 193.136.126.0/24 e 2001:690::/29).

O prefixo mais longo é o melhor (*longest prefix matching is the best*) As tabelas de encaminhamento IP são indexadas por prefixes IP. Um endereço IP pode estar contido em mais do que um dos prefixes existentes na tabela. Se for o caso, o prefixo mais longo, *i.e.*, com mais bits, toma precedência na escolha.

Campos do cabeçalho IP (*IP header fields*) Os campos do cabeçalho dos pacotes IP condicionam a forma como estes são processados e encaminhados entre a

origem e o destino. Entre os principais encontram-se os endereços origem e destino, assim como o campo que impede que um pacote possa permanecer eternamente num ciclo de encaminhamento.

Encaminhamento directo (*direct routing*) Forma de encaminhamento de um pacote por um dispositivo IP r em que o destino do pacote é outro dispositivo IP ligado directamente a r por um canal.

Encaminhamento indirecto (*indirect routing*) Forma de encaminhamento de um pacote por um dispositivo IP r em que o destino do pacote se encontra por detrás de um *router* IP directamente ligado a r por um canal.

Encaminhamento por omissão (*default routing*) Forma de encaminhamento indirecto associada ao prefixo IP 0.0.0.0/0, também conhecido por rota por omissão (*default route*), que funciona como rota de último recurso quando mais nenhuma outra se aplica a um pacote.

Default-Free Zone (DFZ) A região da Internet em que os *routers* não usam rotas por omissão chama-se DFZ pois as suas tabelas de encaminhamento devem, em princípio, conter todos os prefixos IP válidos na Internet pública. Se o endereço de destino de um pacote não está contido em nenhum desses prefixos, então um *router* da DFZ não consegue encaminhá-lo. Tal só deve acontecer com endereços de prefixos não atribuídos ou inválidos.

Address Resolution Protocol (ARP) Protocolo baseado em difusão que permite a um dispositivo IP r obter o endereço MAC de outro dispositivo IP directamente ligado a r via um canal e de que apenas conhece o endereço IP. Este protocolo não existe em IPv6.

Internet Control Message Protocol (ICMP) Protocolo que permite enviar notificações de erro na sequência do tratamento de um pacote para o emissor inicial do mesmo. O protocolo também permite obter meta-informação sobre a rede. Na versão do ICMPv6 estão incluídas as funcionalidades do ARP para IPv6.

Dynamic Host Configuration Protocol (DHCP) Protocolo que permite a um sistema terminal obter a informação necessária para a sua parametrização (*e.g.*, endereço, prefixo, endereços de servidores DNS, *etc.*).

Network Address Translation with Ports (NATP ou simplesmente NAT) Mecanismo que permite a um conjunto de dispositivos de uma rede, partilharem um único endereço quando necessitam de comunicar com o exterior dessa rede. Este mecanismo só permite que a iniciativa da comunicação tenha por origem sistemas pertencentes ao reduto “privado”.

Port forwarding Mecanismo complementar do NAT que permite que um sistema no reduto “privado” receber conexões do exterior usando uma porta fixa.

NAT Traversal Mecanismos complementares do NAT que permitem que um computador no exterior do reduto “privado” tome a iniciativa de comunicar com um dispositivo arbitrário no interior do reduto “privado”.

Referências

O problema do crescimento da dimensão das tabelas de encaminhamento da DFZ tem sido uma preocupação constante pois se os *routers* não fossem capazes de lidar com a dimensão dessa tabela, ou de processarem os pacotes em tempo útil, seria necessário diminuir o número de prefixos visíveis. Este problema foi objecto de um *workshop* cujas conclusões, relatadas no RFC 4984, influenciaram uma investigação intensa quer na forma como conseguir melhorar os *routers*, quer na pesquisa de arquitecturas de encaminhamento que obviasssem esses problemas, mantendo a liberdade dos clientes e dos ISPs.

O aumento do desempenho da pesquisa em tabelas de encaminhamento tem sido dominada pelo desenvolvimento de novos algoritmos [Ruiz-Sanchez et al., 2001] e implementações em hardware. No entanto, têm sido também conseguidos progressos em termos de implementações em software, como por exemplo as relatadas em [Dobrescu et al., 2009; Zec et al., 2012] que permitem manter algum optimismo no que respeita à possibilidade de se utilizarem tabelas de encaminhamento cada vez maiores.

Existem vários livros sobre IPv6. Os seguintes são dos mais reputados [Graziani, 2013; Hagen, 2014]. O artigo [Czyz et al., 2014] constitui uma análise detalhada sobre a adopção de IPv6 por volta do ano de 2014.

WebRTC[Jennings et al., 2013] é um *framework* JavaScript para permitir a comunicação directa entre *browsers* Web mesmo quando os interlocutores estão ambos em redes privadas. O *framework* é um exemplo importante de um contexto em que a problemática NAT *traversal* tem muita relevância e requer APIs e soluções normalizadas como: *Interactive Connectivity Establishment* (ICE), ver os RFCs 4091 e 4092, *Traversal Using Relays around NAT* (TURN) e *Relay Extensions to Session Traversal Utilities for NAT* (STUN), ver o RFC 5766.

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, responsável pela afectação de endereços IP.
- <http://www.internetsociety.org/deploy360/ipv6> – É o *site* oficial da Internet Society sobre o IPv6.
- <http://bgp.potaroo.net> – É um *site* com informação sobre os prefixos IP presentes nas tabelas de encaminhamento globais da Internet, com especial realce para um conjunto de relatórios sobre a sua evolução, incluindo apontadores para outros relatórios como o indicado a seguir.
- <http://www.cidr-report.org/> – É um *site* que contém um relatório elaborado semanalmente, assim como uma análise da evolução histórica, do estado da tabela de encaminhamento da *Default Free Zone* da Internet em IPv4 e IPv6.
- <https://www.vyncke.org/ipv6status/> – É um *site* mantido por Eric Vyncke que apresenta relatórios sobre a evolução da utilização de IPv6.

17.6 Questões para revisão e estudo

1. Verdade ou mentira?
 - (a) O cabeçalho dos pacotes IP contém um campo designado TTL que permite a todo o momento saber quanto tempo passou desde que o pacote foi emitido.
 - (b) O cabeçalho dos pacotes IP contém um campo que assegura que se um pacote entrar num ciclo de encaminhamento dentro da rede, acabará por ser destruído.
 - (c) Os *routers* no interior da rede estão todos sob a mesma autoridade administrativa pois caso contrário seria impossível garantir que os pacotes chegam ao destino.
 - (d) O cabeçalho de um pacote IP contém um campo onde é registada a lista dos *routers* que este atravessou para que possa ser detectado se o pacote entrou em ciclo e deve ser destruído.

2. Verdade ou mentira?
 - (a) Os endereços do nível MAC são hierárquicos e reflectem informação sobre a topologia da rede, *i.e.*, informação sobre a região da rede em que a interface que tem o endereço se situa.
 - (b) Os endereços IP apesar de poderem ser utilizados numa rede com biliões de computadores têm uma estrutura interna que permite evitar que as tabelas de encaminhamento sejam da mesma dimensão.
 - (c) As interfaces Ethernet têm um endereço MAC fixo, potencialmente eterno. O ideal ao nível rede da Internet actual seria os utilizadores poderem ter endereços IP públicos fixos com a mesma propriedade e poderem-nos usar independentemente do ISP a que estivessem ligados fosse qual fosse o país ou região para que se deslocassem. Tal opção não é implementada apenas por conservadorismo da rede Internet.
 - (d) A razão pela qual os ISP não aceitam que os utilizadores Internet mantêm sempre o mesmo endereço IP fixo e independente do ISP (portabilidade de endereços) tem exclusivamente por origem estratégias comerciais de cativação dos clientes.
3. Quantos prefixos IP /24 existem dentro do prefixo 87.103.0.0/17?
4. Quais dos seguintes endereços IP: 192.168.0.129, 192.168.0.145, 192.168.1.129, 192.168.0.1 e 192.168.0.140 estão incluídos no prefixo 192.168.0.128/28?
5. Um *router* recebe por uma das suas interfaces um *frame* Ethernet contendo um pacote IP. Para o processar, o *router* executa várias operações, algumas das quais estão indicadas a seguir, mas as mesmas estão desordenadas. Coloque-as por ordem.
 - (a) se não há entrada na tabela de encaminhamento para o endereço de destino, descarta o pacote, e termina o processamento.
 - (b) processa o *frame* Ethernet, retirando o pacote e colocando-o numa variável.
 - (c) se descartou o pacote devido a um erro, envia um pacote ICMP para o emissor original e termina o processamento.
 - (d) encapsula pacote num *frame*, envia-o para *next-hop* encapsulado no *frame* adequado e termina o processamento.
 - (e) calcula o novo *checksum* do cabeçalho do pacote.
 - (f) decrementa o TTL de pacote e se este ficar com o valor 0 descarta-o.
6. Um pacote IPv4 com um cabeçalho IP com 20 bytes foi recebido e enviado de novo por um dado *router* que não suportava opções de qualidade de serviço, nem teve necessidade de fragmentar. Que campos do cabeçalho do pacote IP foram modificados pelo *router*? Escolha uma e uma só das opções.
 - (a) Nenhum
 - (b) Os campos TTL, Type of Service, Options
 - (c) Os campos TTL, Offset
 - (d) Os campos TTL, Checksum
 - (e) Os campos TTL, Origin Address, Options
 - (f) Os campos TTL, Destination Address
 - (g) Os campos TTL, Cheksum, Origin Address

- (h) Os campos TTL, Cheksum, Destination Address
7. O IPv4 suporta um mecanismo de fragmentação de pacotes pelos *routers*. Responda às seguintes questões:
 - (a) Se um pacote IP foi fragmentado, o receptor final do pacote só consegue recompor o pacote inicial se os diferentes segmentos chegarem pela ordem com que foram produzidos? Justifique a sua resposta.
 - (b) A recomposição de um pacote necessita de usar um alarme. Porquê?
 - (c) No IPv6 resolveu-se abandonar o mecanismo da fragmentação nos *routers*. Indique o porquê e qual ou quais as alternativas que passaram a ser usadas.
 8. Qual a maior dimensão possível do *payload* de um pacote IPv4 que é aconselhável enviar por uma interface Ethernet? Mesma questão com um pacote IPv6?
 9. O campo **Identification** do cabeçalho dos pacotes IPv4 tem 16 bits. Assim, após o envio por um computador de cerca de 64.000 pacotes para o mesmo destino, os números de identificação dos pacotes são necessariamente reutilizados. O computador está a enviar pacotes compatíveis com a dimensão da sua interface Ethernet mas não sabe se os pacotes vão ou não ser fragmentados pelos *routers*. Por hipótese o *Maximum Packet Life Time* (MPLT) é 120 segundos. Indique qual o débito máximo a que este computador pode enviar pacotes para o mesmo destino, para ter a certeza que em caso de fragmentação nenhum pacote pode ser corrompido pelo mecanismo de recomposição dos pacotes no receptor.
 10. Um comutador IP (*router*) tem uma tabela de encaminhamento (**table**) com entradas do tipo **route** (obtidas por `table.get-route(address)`). Uma rota tem os seguintes atributos: **IP prefix**, **type** (`local`, `direct`, `indirect`), **interface** (a **interface** ou `null` se o encaminhamento for `indirecto`), e **next-hop** (o endereço IP do **next-hop** se o encaminhamento é `indirecto` ou `null` nos outros casos). Os pacotes IP são do tipo **packet** com vários atributos entre os quais: **origin** (endereço IP), **destination** (endereço IP) e **TTL** (um inteiro). Os comutador pode processar os pacotes recebidos por um dos seguintes métodos: `packet.ignore()` - que destrói o pacote retirando-o dos *buffers* do comutador `packet.process()` - que processa um pacote dirigido a uma das interfaces do comutador e que necessariamente verifica a condição:
`tabela.get-rota(packet.destination).tipo == local`
`packet.send(interface, next-hop)` - para envio de um pacote e em que interface é necessariamente diferente de `null` e `next-hop` é um endereço IP diretamente alcançável pelo comutador.
 Apresente o pseudo-código de processamento pelo router de um pacote p.
 11. A tabela de encaminhamento de um computador tem as entradas a seguir indicadas. No entanto, uma delas é impossível e não serve para nada. Diga qual e justifique a sua resposta.

192.10.1.1/32	local	eth0
192.10.1.0/24	directo	eth0
192.10.2.0/24	indirecto	netx hop: 192.10.1.254
192.10.3.0/24	indirecto	netx hop: 192.10.6.254
 12. A tabela de encaminhamento de um computador tem as entradas a seguir indicadas.

190.30.16.1/32	local	eth0
190.30.16.0/24	directo	eth0
190.40.0.0/16	directo	eth1
190.30.45.0/24	indirecto	next hop: 190.30.16.10
190.30.48.0/24	indirecto	next hop: 190.30.32.10
0.0.0.0/0	indirecto	next hop: 190.30.16.10

- (a) Dado o endereço 190.30.16.100 qual das entradas da tabela é escolhida para decidir qual é o *next hop*?
 - (b) Dado o endereço 190.30.100.34 qual das entradas da tabela é escolhida para decidir qual é o *next hop*?

13. A tabela de encaminhamento de um computador tem as seguintes entradas.

```
192.10.1.0/24      directo      eth0
192.10.2.0/24      directo      eth1
192.10.3.0/24      indirecto    next hop: 192.10.1.254
192.10.4.0/24      indirecto    next hop: 192.10.2.254
0.0.0.0            indirecto    next hop: 192.10.1.254
```

- (a) Uma destas entradas é redundante e pode ser suprimida. Diga qual.
 - (b) Dado o endereço 192.10.3.15 qual das entradas da mesma é escolhida para decidir qual é o *next hop*?

14. Considere a seguinte tabela de encaminhamento.

```
190.30.16.0/20      directo      eth0
190.30.32.0/20      directo      eth1
190.30.0.0/20      indirecto    next hop: 190.30.16.1
190.30.48.0/20      indirecto    next hop: 190.30.32.1
0.0.0.0            indirecto    next hop: 190.30.16.1
```

- (a) Dado o endereço 190.30.64.34 qual das entradas da mesma é escolhida para decidir qual é o *next hop*?
 - (b) Dado o endereço 190.30.49.34 qual das entradas da mesma é escolhida para decidir qual é o *next hop*?

15. Considere a rede de um cliente da Figura 17.23 e indique o conteúdo da tabela de encaminhamento do *router* do cliente.

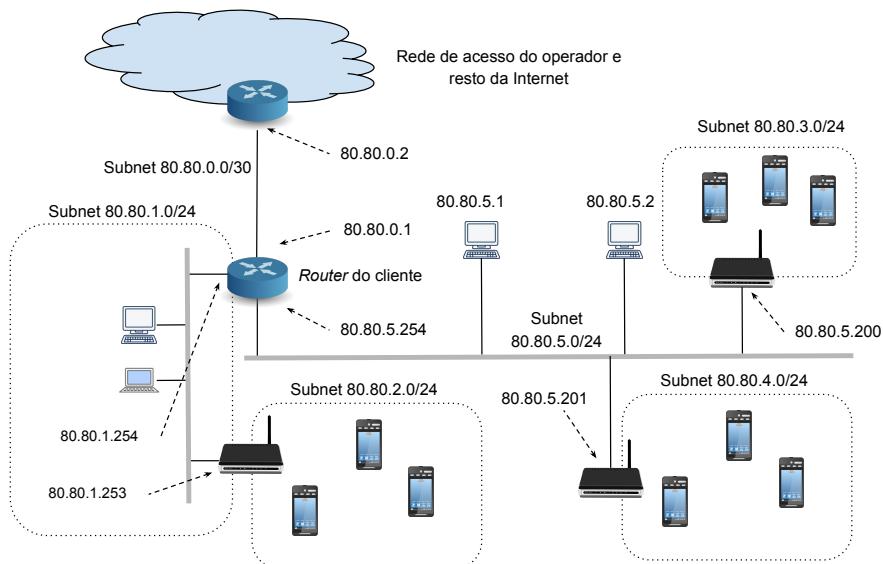


Figura 17.23: Rede de um cliente ligado à Internet através de um ISP

16. Considere a rede da Figura 17.24 e indique o conteúdo da tabela de encaminhamento do *router R1*. Indique também se o *router R2* deve ou não executar a função de NAT (*Network Address Translation*)?

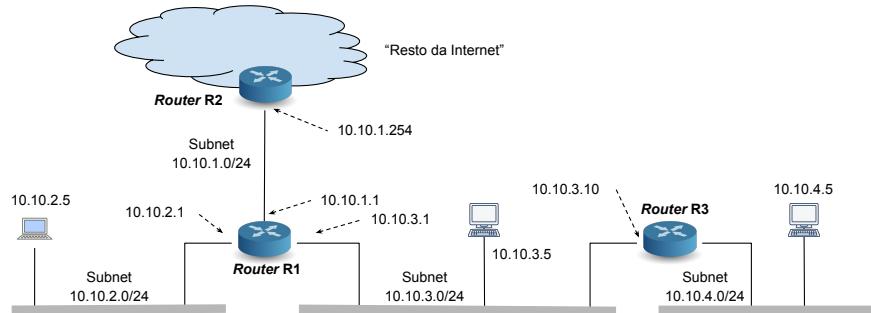


Figura 17.24: Rede ligada à Internet através de um ISP

17. Um *router r* tem duas interfaces: a interface *eth0* com IP 10.0.1.254 e *eth1* com IP 10.0.2.254. As tabelas de ARP e encaminhamento do *router* têm o seguinte conteúdo:

10.0.1.1	23:45:A0:4F:67:CD	(arp)
10.0.1.254	23:45:AB:2F:83:AD	(arp)
10.0.2.254	23:45:AB:E3:56:D2	(arp)
10.0.1.0/24	directo eth0	
10.0.2.0/24	directo eth1	
0.0.0.0/0	indirecto via 10.0.2.254	

Indique que *frames* é que o *router* tem de enviar (tipo do *frame* e endereços MAC origem e destino) para encaminhar um pacote:

- (a) Para o endereço 10.0.2.254?
 - (b) Para o endereço 10.0.2.100?
 - (c) Para o endereço 10.0.3.30?
 - (d) Caso a interface *eth1* receba um *frame* Ethernet do tipo ARP com a *query*: **Who has address 10.0.2.100?**, o *router* deve responder à mesma? Justifique a sua resposta.
18. Um *router r* tem duas interfaces: a interface *eth0* com IP 10.0.1.254 e *eth1* com IP 10.0.2.254. As tabelas de ARP e encaminhamento do *router* têm o seguinte conteúdo:
- | | | |
|-------------|-------------------|-------|
| 10.0.1.1 | 23:45:A0:4F:67:CD | (arp) |
| 10.0.1.254 | 23:45:AB:2F:83:AD | (arp) |
| 10.0.2.254 | 23:45:AB:E3:56:D2 | (arp) |
| 10.0.1.0/24 | directo eth0 | |
| 10.0.2.0/24 | directo eth1 | |
- Indique que *frames* é que o *router* tem de enviar (tipo do *frame* e endereços MAC origem e destino) para encaminhar um pacote para o endereço 10.0.3.30?
19. Um computador tem uma interface com o endereço 100.100.0.1/24 e recebe um pacote com endereço de destino 100.100.0.100. A tabela de ARP do computador tem o seguinte conteúdo:

100.100.0.1	00:00:AA:BB:00:10
100.100.0.4	00:00:AA:BB:00:02
100.100.0.10	00:00:AA:BB:00:01
100.100.0.67	00:00:AA:BB:00:55

Indique quais os *frames* Ethernet que o computador deve enviar para encaminhar o pacote até ao seu destino.

20. Descreva de que forma o programa **traceroute** utiliza o protocolo ICMP para funcionar.
21. Um servidor DHCP pretende assegurar-se, antes de afectar o endereço IP a um computador, que não existe nenhum outro computador a usar o mesmo endereço IP na sub-rede que controla. Como pode proceder?
22. Numa rede IP com suporte de difusão de *frames* Ethernet existe um servidor DHCP, o servidor DHCP1, que afeta endereços IP no prefixo 100.100.100.0/24 aos computadores que se ligam a essa rede, com um *lease time* de 1000 segundos. O servidor DHCP1 avariou-se e foi substituído imediatamente pelo servidor DHCP2 que não conhece, no momento do arranque, quais os endereços afetados pelo servidor DHCP1 que ainda estão a ser usados. DHCP2 afeta endereços no mesmo prefixo que DHCP1 e tem agora o problema de quando recebe um novo pedido de endereço, decidir que endereço afetar. DHCP2 poderia correr o risco de afetar endereços aleatoriamente na esperança de que não houvessem colisões com endereços afetados por DHCP1 que ainda estivessem a ser usados. Poderia também pedir a todos os computadores que devolvessem os endereços em uso e pedissem um novo, mas o protocolo DHCP não suporta essa hipótese.
Que pode fazer DHCP2 para diminuir a probabilidade de afetar endereços que ainda estejam a ser usados por computadores ligados à sua rede? Justifique a sua resposta.
23. Um computador recebeu um endereço IP de um servidor DHCP (único nessa rede) mas pouco depois esse servidor foi aberto antes da *lease* do cliente expirar. O servidor DHCP não recuperou da avaria senão 24 horas depois. Quais das seguintes afirmações são de certeza verdadeiras?
 - (a) O computador perdeu a conectividade com a Internet porque o servidor DHCP deixou de encaminhar os seus pacotes para a Internet.
 - (b) O computador percebeu-se da ausência do servidor DHCP quando a *lease* expirou.
 - (c) Outros computadores deixaram de conseguir obter endereços IP na mesma rede.
 - (d) O computador deixou de conseguir aceder a *sites* WEB porque deixou de resolver *queries* DNS.
 - (e) O computador deixou de ter actualizações da sua tabela de encaminhamento.
24. Os RFCs relacionados com o protocolo DHCP recomendam que um servidor DHCP envie uma mensagem ARP sobre o endereço que vai afectar antes de fazer uma *offer* do mesmo. Quais das seguintes afirmações podem justificar esta recomendação?
 - (a) O servidor DHCP não sabe que endereços IP afetou antes de um seu anterior *crash*.
 - (b) O utilizador de algum computador pode utilizar um endereço livre mesmo sem o mesmo ter sido afectado pelo servidor DHCP.

- (c) O servidor DHCP foi reinicializado.
- (d) De forma a que os *switches* dessa *subnet* passem a conhecer o endereço MAC do servidor DHCP, e optimizem a sua localização pelos clientes.
25. Um *router* recebeu um pacote IPv4 com endereço origem 130.45.3.3 e endereço de destino 201.23.4.6. Não existe nenhum prefixo de rede na tabela de encaminhamento que contenha esse endereço e não há nenhuma rota por omissão (*default route*). Como deve o *router* tratar esta situação? (Só uma das opções abaixo é válida)
- (a) Devolver o pacote ao endereço 130.45.3.3
 - (b) Destruir o pacote e avisar por ICMP o *router* anterior
 - (c) Destruir o pacote e enviar uma mensagem ICMP do tipo “Host Unreachable” para o endereço 201.23.4.6
 - (d) Avisar o computador no endereço 130.45.3.3 de que não sabe encaminhar o pacote
 - (e) Destruir o pacote e enviar uma mensagem ICMP do tipo “Host Unreachable” para o endereço 201.23.4.6
 - (f) Destruir o pacote e enviar uma mensagem ICMP do tipo “Host Unreachable” para o endereço 130.45.3.3
26. Descreva como o protocolo ICMP pode ser usado para descobrir qual o valor do MTU que deve ser usado para o envio de pacotes sem fragmentação entre dois computadores.
27. NAT neste contexto designa *Network Address Translation with Ports*. Qual ou quais das seguintes afirmações são verdadeiras?
- (a) O NAT permite que um único endereço IP privado seja partilhado por vários computadores da rede interna.
 - (b) O NAT permite que um único endereço IP público seja partilhado por vários computadores da rede interna.
 - (c) A técnica de NAT permite que o TCP funcione mais rapidamente pois faz *caching* de conexões TCP e permite que o estabelecimento da conexão (*three way hand shaking*) funcione mais depressa.
28. O computador *A* tem uma ligação TCP a um servidor *S* pelo que lhe envia pacotes contendo segmentos TCP. *A* e *S* estão ligados através de vários comutadores e diversos canais Ethernet. *A* tem o endereço 10.0.1.12 no prefixo 10.0.1.0/24 e *S* tem o endereço 193.136.126.43 no prefixo 193.136.126.0/24. Quer na emissão, quer na receção, as mensagens trocadas entre *A* e *S* têm três cabeçalhos (cabeçalho do *frame* Ethernet, cabeçalho IP e cabeçalho TCP). Durante o transporte de uma dessas mensagens de *A* para *S* alguns cabeçalhos podem eventualmente ser alterados. Qual ou quais das seguintes afirmações são verdadeiras ?
- (a) Os três cabeçalhos da mensagem recebida por *S* contendo um segmento TCP são os mesmos que os da mensagem transmitida originalmente por *A*.
 - (b) Os cabeçalhos do *frame* Ethernet e o cabeçalho IP da mensagem recebida por *S* são exatamente os mesmos que os da mensagem transmitida por *A*, mas o cabeçalho TCP é diferente
 - (c) Os cabeçalhos do *frame* Ethernet e o cabeçalho IP da mensagem recebida por *S* são diferentes dos da mensagem transmitida por *A*, mas o cabeçalho TCP é igual.

- (d) O endereço IP origem dos pacotes recebidos pelo servidor é o endereço do computador *A*.
- (e) O cabeçalho do *frame* Ethernet da mensagem recebida por *S* é exatamente o mesmo que o da mensagem transmitida por *A*, mas os cabeçalhos IP e TCP são diferentes.
- (f) Os três cabeçalhos da mensagem recebida por *S* contendo um segmento TCP são todos diferentes dos da mensagem transmitida originalmente por *A*.
- (g) O endereço MAC origem dos *frames* Ethernet recebidos pelo servidor *S* é o endereço MAC da interface externa do *router* que liga a sua rede do computador *A* ao seu fornecedor de serviços Internet.
29. O computador *A*, numa rede local de uma universidade, está a enviar segmentos TCP para um computador *S*, numa rede local de uma universidade de outra cidade. *A* tem um endereço privado e *S* um endereço público. Diga qual ou quais dos seguintes campos dos cabeçalhos IP e TCP dos segmentos são alterados entre a emissão por *A* e a recepção por *S*:
- (a) Números de sequência do segmento TCP
 - (b) *advertised window size* do segmento TCP
 - (c) Endereço IP origem do pacote IP
 - (d) Endereço IP destino do pacote IP
 - (e) Número de porta origem do segmento TCP
 - (f) Número de porta destino do segmento TCP
 - (g) Campo TTL do pacote IP
 - (h) Campo qualidade de serviço do pacote IP
 - (i) Campo *checksum* do pacote IP
 - (j) Campo *checksum* do pacote TCP
30. Um *router* está a desempenhar igualmente funções de NAT (*Network Address Translation with Ports*). Quando se abre uma nova conexão TCP, ou quando passa um primeiro segmento UDP (com endereço origem, destino, porta origem ou porta destino diferentes) é criada uma nova associação na tabela de associações NAT. O problema reside em decidir quando é que a mesma associação é declarada “morta” e pode ser removida. Em TCP é possível detectar o fecho da conexão, mas em UDP essa noção não existe. Por outro lado, em TCP se uma das partes “morre”, a conexão não pode ser fechada regularmente. Como arranjar uma forma adequada de lidar com estes dois problemas?
31. Um *router* está a desempenhar igualmente funções de NAT (*Network Address Translation with Ports*). Associado aos pacotes ICMP não existem portas. Será mesmo assim possível encontrar uma forma adequada de lidar com NAT *traversal* de pacotes ICMP?

Capítulo 18

Encaminhamento na Internet global

"In theory, there is no difference between theory and practice. But in practice, there is."

– Autor: *Yogi Berra*

A Internet é uma rede de redes IP ou um *internetwork IP*, i.e., uma interligação de redes com interfaces IP. Entre essas redes interligadas encontramos redes domésticas, redes institucionais, redes de centros de dados, redes de acesso e redes de trânsito. Com exceção das redes domésticas e institucionais, todas as redes são operadas por operadores especializados em fornecerem serviços de rede, cuja actividade é desempenhada normalmente no quadro de uma empresa com fins lucrativos.

Os mecanismos que foram introduzidos no Capítulo 17 são suficientes para o funcionamento das redes domésticas e das redes institucionais de pequena dimensão. Em muitas redes IP de média dimensão, os protocolos apresentados no Capítulo 16 são suficientes para a parametrização e gestão do encaminhamento interno às mesmas.

No entanto, a interligação de redes de grande dimensão, em particular de redes dos operadores, tem requisitos especiais. A este nível a escala é muito significativa e tem crescido sem cessar. Por outro lado, os critérios de escolha das rotas não dependem apenas de constrangimentos de desempenho, mas dependem igualmente de constrangimentos relacionados com relações comerciais e políticas entre operadores. Para essa interligação de redes IP existe um único protocolo normalizado e omnipresente que tenta responder a esses requisitos especiais, trata-se do protocolo BGP (*Border Gateway Protocol*). A discussão desses requisitos e deste protocolo é o objectivo deste capítulo.

18.1 Os problemas da escala

Quando uma rede tem uma dimensão da ordem de grandezza dos milhares ou mesmo milhões de *routers*, os protocolos de encaminhamento pelo melhor caminho apresentados no Capítulo 16 têm problemas em suportarem esta escala. Com efeito, numa rede com essa dimensão, o número de actualizações a propagar por unidade de tempo é certamente muito grande, pois em cada momento há sempre algum evento num *router* com repercuções na maioria dos outros *routers*, por exemplo quando uma sub-rede

deixa de ser acessível. Por isso os protocolos OSPF e IS-IS, ver o Capítulo 16, suportam a partição da rede em várias zonas: uma zona chamada *backbone*, e outras subordinadas, interligadas pela primeira, ver a Figura 18.1.

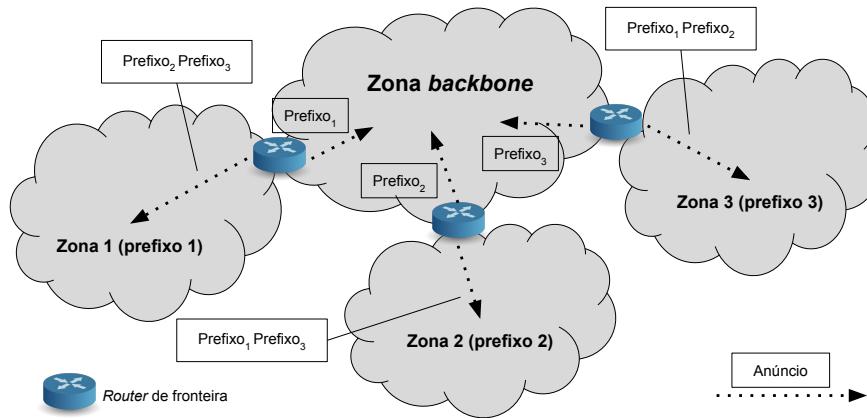


Figura 18.1: Organização de uma rede OSPF em várias zonas.

Com esta organização hierárquica as alterações de estado dos canais apenas são propagadas por inundação para os *routers* da zona onde a alteração teve lugar, e cada uma das zonas só conhece os detalhes da sua rede interna.

Compete aos *routers* que fazem a ligação de cada zona ao *backbone* fazerem a interface em termos de tráfego e de encaminhamento: o *backbone* deve conhecer os prefixos internos a cada zona e assegurar a ligação entre as mesmas, e cada zona deve conhecer todos os prefixos conhecidos pelo *backbone*, i.e., os prefixos das outras zonas. Estes *routers*, chamados de *fronteira*, isolam cada zona dos detalhes sobre o estado das outras zonas, mas propagam os prefixos conhecidos de cada lado da fronteira. Uma forma de aumentar drasticamente a escalabilidade desta estrutura consiste em organizar o endereçamento de tal forma que cada zona possa ser conhecida pelas outras apenas por um único prefixo IP, o qual tem como sub-prefixos todos os prefixos internos da zona.

Com uma gestão adequada dos prefixos e dos anúncios, também é possível fazer uma organização hierárquica do mesmo tipo com base em protocolos do tipo vector de distâncias.

A organização da rede em zonas permite aumentar a sua dimensão sem que os custos do seu *control plane* se tornem insuportáveis, quer do ponto de vista do tráfego de propagação das actualizações do estado da rede, quer do ponto de vista dos recursos usados nos *routers* para os processar. Em contrapartida, ela cria vários constrangimentos pois impõe uma organização estrita da rede que não se compadeca com a evolução dos requisitos, da dimensão relativa das zonas etc. Ao fim de algum tempo torna-se necessário reorganizar as zonas e o endereçamento. Ora quanto maior for a rede, maior será o custo destas reorganizações.

No início da Internet, quando esta não passava de um projecto académico e científico, a estrutura adoptada era a de uma rede organizada como uma árvore em que a raiz era o *backbone* da NFSNET, i.e., o *backbone* subsidiado pela National Science Foundation dos EUA. Este *backbone* assegurava a interligação das redes regionais, as quais asseguravam a interligação das redes locais, ver a Figura 18.2.

Com a comercialização da Internet, ocorrida durante a primeira metade dos anos 90, aparecerem vários grandes *backbones* comerciais, que se interligaram entre si e forneciam serviços às redes regionais e mesmo a algumas locais. O dinamismo da

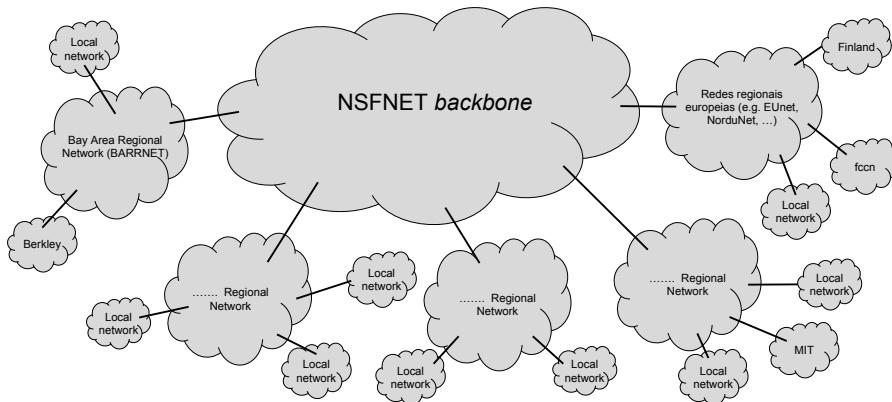


Figura 18.2: Estrutura da Internet pré-comercialização com exemplos de redes regionais e locais

Internet em termos de operadores a entrarem e a saírem do mercado, a evolução da dimensão relativa das diferentes redes nas diferentes regiões do mundo, a generalização da interligação directa de redes regionais e locais, *etc.* deixou de ser compatível com uma organização estritamente hierárquica e a hierarquização (mais ou menos estrita) dos prefixos IP.

Dada a dimensão gigantesca desta rede de redes, também não é realista usar os protocolos tradicionais para gerir o encaminhamento do conjunto dos milhões de *routers* existentes no seu conjunto. Os protocolos do tipo *link-state* não escalariam até à dimensão de um potencial *backbone* da Internet. Por outro lado, dada a heterogeneidade das diferentes redes interligadas, em dimensão, responsabilidade, forma de gestão, métricas do custo do encaminhamento, *etc.* é indesejável impor-lhes um funcionamento e práticas de gestão uniformizadas. Essa heterogeneidade requer descentralização e isolamento da gestão, das soluções técnicas adoptadas e também das métricas e outras técnicas usadas para gerir cada sub-rede da Internet.

Para responder a estes desafios foi necessário introduzir novos conceitos de organização do encaminhamento, nomeadamente a noção de sistema autónomo, e protocolos capazes de lidarem com a nova situação, nomeadamente o protocolo BGP.

18.2 Sistemas autónomos

Um **Sistema Autónomo (Autonomous System)**, que abreviaremos daqui para a frente por AS, é uma rede ou conjunto de redes sob a mesma administração e cujo encaminhamento interno é opaco para o exterior. A forma como o encaminhamento interno ao AS é realizado é um problema do AS que este deve resolver como achar melhor, usando os protocolos e métricas de custo que preferir. A fronteira dos ASs é constituída por *routers* especiais, os *routers de fronteira* ou *border routers*, que asseguram a ligação com os outros ASs.

Assim, o encaminhamento na Internet está estruturado a dois níveis: **Intra-AS**, o qual é da responsabilidade de cada AS, e **Inter-AS** que tem de ser uniformizado e normalizado. A Internet global é formada pela interligação entre si dos diferentes ASs, via o encaminhamento inter-AS que é suportado por um único protocolo de encaminhamento, o protocolo BGP ou *Border Gateway Protocol*, executado pelos *border*

*routers*¹. Naturalmente, a interface de troca real de pacotes entre ASs tem de ser compatível com o protocolo IP.

Os protocolos de encaminhamento são assim classificados em protocolos IGP (*Internal Gateway Protocols*) e protocolos EGP (*External Gateway Protocols*), que se resumem actualmente ao protocolo BGP.

Encaminhamento intra-AS

O encaminhamento dentro de um AS é assegurado usando encaminhamento estático e dinâmico. Neste último caso os protocolos mais populares são os protocolos RIP, EIGP, OSPF, IS-IS, etc. que são protocolos que se baseiam nos princípios apresentados no Capítulo 16. Nas redes dos operadores de grande dimensão, os chamados *carriers*, são usadas outras soluções mais complexas. Em alguns centros de dados também são usadas soluções especiais.

Os protocolos de encaminhamento usados constituem o chamado **control plane do AS**. Este tem por papel fazer chegar aos intra-*routers* a informação que eles usam para manterem as suas tabelas de encaminhamento (ou *Forward Information Bases* (FIBs)) actualizadas com informação da mesma natureza que as tabelas de encaminhamento IP apresentadas na Capítulo anterior, ver o Capítulo 17. Assim, a informação de acessibilidade que estes protocolos transmitem entre *routers* é sempre em termos de prefixos IP e custos. As métricas de custos desses protocolos são selecionadas de molde a atingirem-se os objectivos de gestão pretendidos pelos gestores do AS. A variedade de critérios e a diversidade das redes determina a liberdade de escolha das métricas e dos protocolos intra-AS. Seria impossível impor métricas uniformes e universais adequadas a todos os contextos pois os mesmos são muito diferentes uns dos outros.

O AS de um operador inclui também as redes dos seus clientes cujos prefixos estão escondidos dentro dos seus próprios prefixos, pois só a ele estão ligados e não têm AS próprio. Neste caso, os protocolos de encaminhamento usados entre os clientes e o operador também são escolhidos com bastante liberdade. Caso uma dessas redes tenha um único canal de ligação ao operador, uma simples rota por omissão (*default route*) é suficiente para que o caminho para todos os destinos da Internet seja conhecido em toda a rede do cliente.

Cada AS existente na Internet tem um número único a nível mundial que, tal como os prefixos IP, é afectado pela IANA via os RIRs. Os números de AS não têm significado geográfico e são globais. Inicialmente os números de AS foram afectados num espaço com 16 bits, mas actualmente são afectados num espaço com 32 bits pois o seu número tem crescido continuamente.

Um **Sistema Autónomo (Autonomous System)** é uma rede ou conjunto de redes sob a mesma administração e cujo encaminhamento interno é opaco para o exterior.

Os protocolos de encaminhamento são classificados em **protocolos IGP (Internal Gateway Protocols)** ou **Intra-AS**, ver o Capítulo ??, os protocolos usados dentro dos ASs, e **protocolos EGP (External Gateway Protocols)** ou **Inter-AS**. Neste capítulo introduziremos o protocolo deste tipo que é mais popular, o protocolo BGP.

Encaminhamento inter-AS

Um AS conhece os outros ASs pelos seus números de AS e pelos prefixos IP que lhes pertencem. Como veremos a seguir, para efeitos de encaminhamento, os ASs

¹ Antigamente os *routers* que asseguravam a interligação das redes eram designados como *gateways*, daí a designação do protocolo.

podem ser assimilados e relacionam-se entre si como *routers* virtuais e os diferentes ASs encontram-se ligados por múltiplos canais, todos logicamente do tipo ponto-a-ponto. Do ponto de vista do encaminhamento inter-AS, não existem restrições à forma como os diferentes ASs se ligam entre si.

Os sistemas computacionais com endereços num dos prefixos pertencentes a um AS estão necessariamente no seu interior. O encaminhamento inter-AS tem por objectivo fazer chegar os pacotes ao AS a que pertence o prefixo que contém os seus endereços de destino. O encaminhamento intra-AS assume a responsabilidade de fazer a entrega ao sistema final.

Os ASs estabelecem assim uma partição das redes interligadas na Internet e uma partição do espaço dos prefixos IP. A este nível um prefixo IP visível está confinado a um único AS e é autónomo (do ponto de vista do encaminhamento pela regra do *longest prefix matching rule*) pois não pode estar subordinado do ponto de vista do encaminhamento a nenhum outro prefixo (mesmo os que o contenham). A Figura 18.2 ilustra diversos ASs, as sua relações com os prefixos e as suas interconexões.

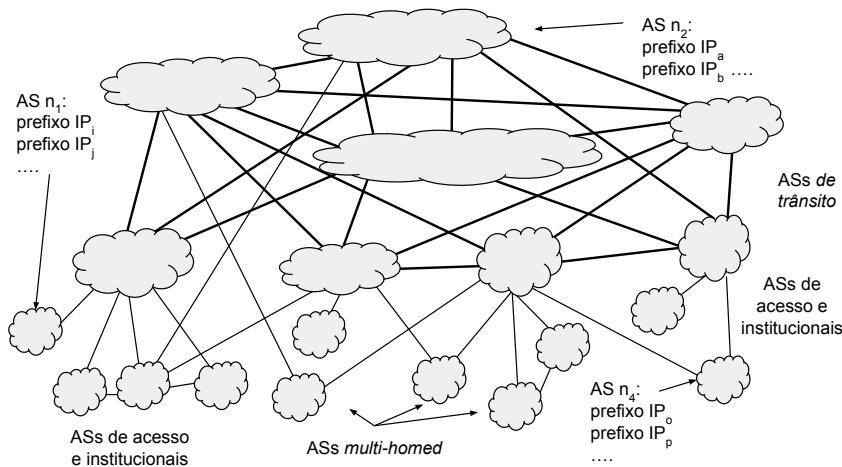


Figura 18.3: A Internet é formada por um conjunto de ASs interligados. Cada AS tem um número próprio e um conjunto autónomo de prefixos IP. Tipicamente, os ASs de maior dimensão são Tier-1, os intermédios são Tier-2 e os mais pequenos Tier-3 ou *Stub-ASs*

O encaminhamento inter-ASs é assegurado pelo protocolo **BGP (Border Gateway Protocol)** que é geralmente executado pelos *routers* de fronteira ou *border routers*, os quais são responsáveis pela interligação dos ASs.

Todos os prefixos IP existentes na Internet pública pertencem necessariamente a um AS, ou estão incluídos num que pertence. Quando uma rede está ligada a diferentes ASs, constitui necessariamente um AS específico.

Stub ASs e Transit ASs

A tipologia dos ASs e a caracterização das suas relações já foi brevemente introduzida na Secção 4.4. Uma primeira classificação dos ASs tem a ver com a sua dimensão. Os maiores ASs são designados por ASs *Tier-1* (nível 1), os intermédios por ASs *Tier-2*

(nível 2) e os mais pequenos por ASs *Tier-3* (nível 3). Neste capítulo a classificação que nos interessa é outra: os ASs também podem ser classificados em *Stub ASs* e *Transit ASs*, em função da origem e destino dos pacotes que neles circulam.

Os *Stub ASs* estão na periferia da Internet, *i.e.*, são os ASs dos “extremos”. Os pacotes que neles circulam têm um endereço de origem ou um endereço de destino contido nos seus prefixos próprios. Ou seja, o trajecto desses pacotes ou começa ou acaba nesses ASs.

Ao contrário, os *Transit ASs* asseguram o trânsito dos pacotes na Internet pois também transportam pacotes originados e com destino a outros ASs. Os ASs de trânsito transportam pacotes cujos endereços de origem e de destino não pertencem a sistemas finais existentes dentro da sua rede. Muitos *Transit ASs* também são *Stub ASs* para alguns dos seus prefixos.

Os *Stub ASs* que estão ligados a mais do que um AS mas que não fazem trânsito entre os mesmos, chamam-se *multi-homed ASs*. Tecnicamente não existem restrições a quem pode ou não ser um AS. Uma rede doméstica pode adquirir o estatuto (e a complexidade de interligação) de um AS.

Se considerarmos a analogia com *routers* virtuais, os *Stub ASs* são os sistemas finais e os *Transit ASs* são os sistemas intermediários (os *routers*). À primeira vista os *Stub ASs* e os *multi-homed ASs* deveriam ser todos de tipo *Tier-3*, *i.e.*, de pequena dimensão, mas essa correspondência não é directa. Para além de algumas multinacionais de grande dimensão, os ASs dos grandes fornecedores de conteúdos (*e.g.*, Google, Facebook, Amazon, *etc.*), que têm centros de dados directamente interligados entre si espalhados pelo mundo, mas que não fornecem serviços de trânsito de pacotes pois apenas recebem e enviam pacotes de outros ASs dirigidos aos seus servidores, são *multi-homed ASs*, mas do ponto de vista da dimensão seriam equiparáveis a ASs do tipo *Tier-1*.

Caderno de encargos do encaminhamento inter-ASs

Os ASs estão interligados entre si por múltiplos canais. Por exemplo, os ASs *Tier-1* são transcontinentais e interligam-se com outros ASs *Tier-1* em diferentes cidades dos diferentes continentes. Os AS *Tier-2* e *Tier-3* estão geralmente confinados a continentes ou a países, no entanto, para melhorarem a sua conectividade, estão também ligados a outros ASs em diferentes continentes, nomeadamente a diversos ASs *Tier-1*. Em geral, os *Stub ASs* apenas estão ligados a ASs da sua região geográfica, com exceção dos ASs dos grandes operadores de conteúdos que também são intercontinentais e estão ligados de forma generalizada aos outros operadores. Com efeito, para melhorarem a sua conectividade, é frequente estes ASs terem *reverse proxies*, ver a Secção 13.3, e *border routers* no interior das instalações de operadores.

Muitas cidades que concentram muita população e actividade económica e também muito tráfego Internet, dispõem de infra-estrutura de interligação de operadores, chamadas Internet *exchange points* ou Internet *peering points*. Tratam-se de edifícios onde se concentram muitos *border routers* dos operadores e onde é fácil estabelecer interconexões entre os mesmos.

O número de prefixos IP para os quais deve ser assegurado o encaminhamento na Internet era, no primeiro trimestre de 2016, cerca de 620.000, e o número de diferentes sistemas autónomos activos é de cerca de 55.000. Alguns dos maiores ASs têm milhares de ligações a outros ASs. A maioria dos ASs intermédios tem dezenas de ligações a outros ASs. Felizmente, a maioria dos *Stub ASs*, que são a larga maioria dos ASs, têm ligação a apenas um só AS. De qualquer forma, a escala do grafo dos ASs é muito grande e o grau dos seus nós centrais também é muito elevado.

O encaminhamento inter-ASs tem também requisitos especiais, relacionados com a estrutura da Internet e as relações, comerciais ou políticas, que são estabelecidas entre os operadores das redes de acesso, de centros de dados e de trânsito. Essas relações são geralmente do tipo cliente / fornecedor ou de cooperação. O protocolo não poderá

impõe constrangimentos a essa relações, nem influenciar o jogo da concorrência e das relações comerciais. Terá de ser neutro desses pontos de vista. Por exemplo, será essencial que admita a seleção de caminhos por critérios comerciais ou políticos.

Uma solução que implicasse uma hierarquização estrita da rede e dos prefixos resultaria muito difícil de gerir e pouco flexível, pois implicaria uma estrutura de relações entre ASs muito associada à dimensão e à geografia e a áreas de influência geo política. Adicionalmente, essa estrutura teria de ser pré-planeada e não seria fácil de alterar. Esta solução não é adequada pois essa estrutura dificilmente permitiria a flexibilidade e liberdade de escolha de caminhos acima enunciada.

Que tipo de protocolos de encaminhamento escolher?

Os protocolos baseados em anúncios de visibilidade permitem controlar os anúncios, fazendo-os de forma selectiva. Quando um anúncio é feito a um vizinho, quer dizer que se lhe está comunicar que se conhece um caminho para o destino, e também que se está disposto a encaminhar pacotes para esse destino. Uma maneira de aceitar encaminhar os pacotes enviados por um AS para um certo destino, e não aceitar para outro, passa por anunciar-lhe esse destino e esconder-lhe o outro. Assim, estes protocolos revelam maior flexibilidade para controlar a visibilidade e os caminhos possíveis.

No entanto, os protocolos do tipo “vector de distâncias” têm o problema de poderem introduzir ciclos de encaminhamento, por isso é necessário encontrar formas de o impedir. Para este efeito é possível incluir no anúncio o caminho usado para lá chegar. Desta forma o receptor do anúncio pode verificar se faz parte do mesmo e ignorar os anúncios que conduziram a ciclos de encaminhamento. Por outro lado, conhecer o caminho usado também dá mais informação para o processo de escolha dos caminhos. Este tipo de protocolos dizem-se protocolos baseados em “vectores de caminhos” ou *AS paths*.

AS paths

O protocolo BGP é um protocolo baseado em anúncios de caminhos, materializados numa lista de números de AS e designados por *AS paths*. Um anúncio BGP contém um ou mais prefixos IP, um *AS path*, i.e., o caminho em ASs para lá chegar, e ainda o endereço IP do *next hop*, i.e., o endereço IP do *border router* para que deve ser dirigido o tráfego destinado aos prefixos assim anunciados. Para cada prefixo, um *border router* pode receber dos seus vários vizinhos diversos *AS paths*, selecciona um deles e, se desejar, acrescenta o seu número de AS a esse *AS path* e propaga-o num novo anúncio com o *AS path* modificado.

O protocolo só propaga o *AS path* seleccionado pelo anunciante e não todos os que este conhece, senão o número de *AS paths* explodiria. A Figura 18.4 ilustra os anúncios BGP e como estes são propagados de AS em AS. Repare-se também que os anúncios fluem num sentido e o tráfego no sentido inverso. É como se os anúncios atraíssem o tráfego.

Um anúncio ou rota BGP é constituída por uma lista de prefixos e vários atributos comuns aos mesmos. Entre estes figura um caminho expresso como uma lista dos ASs, chamado *AS path*, que o anunciante usa para chegar ao AS onde residem os prefixos. Uma rota contém também um atributo com o endereço do *border router* de recepção do tráfego que é dirigido aos seus prefixos.

Os *AS paths* permitem saber exactamente por que ASs passarão os pacotes, o que facilita a tomada de decisão em função do caminho proposto e permite suprimir a introdução de ciclos de encaminhamento.

Dado os anúncios BGP conterem caminhos, o protocolo é caracterizado como sendo um protocolo com base em **prefixos e vectores de caminhos (prefix-based path-vector protocol)**.

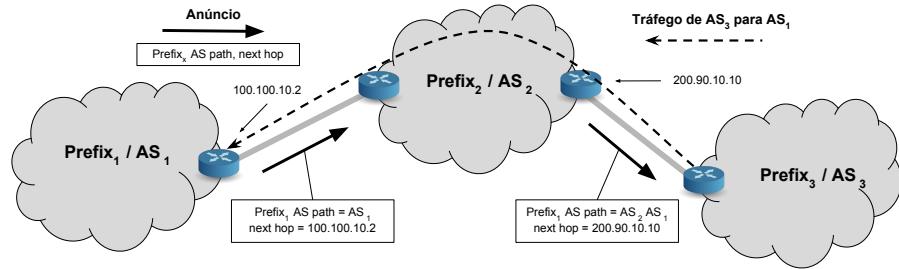


Figura 18.4: Forma e propagação dos anúncios BGP e como os mesmos atraem o tráfego dirigido ao prefixo anunciado.

Anúncios BGP e relações comerciais entre ASs

As relações comerciais entre ASs são de diferentes tipos, nomeadamente: **cliente / fornecedor** (e vice-versa) e **peer-to-peer**, ver a Figura 18.5.

As relações cliente / fornecedor são simples de perceber: o AS_a é cliente de trânsito pago do AS_b se AS_a paga ao AS_b para este fazer chegar os seus pacotes e os dos seus clientes ao destino final, e também para que o AS_b lhe entregue os pacotes vindos de outros ASs que se destinam a ele ou aos seus clientes.

Para que o AS_b seja fornecedor do AS_a , AS_b tem de anunciar aos outros ASs a que está ligado que tem um caminho para os prefixos do AS_a incluindo os dos seus clientes. Quando um AS fornece serviços de encaminhamento a outro AS diz-se que o primeiro fornece trânsito pago ao outro (*provides a paid transit service*).

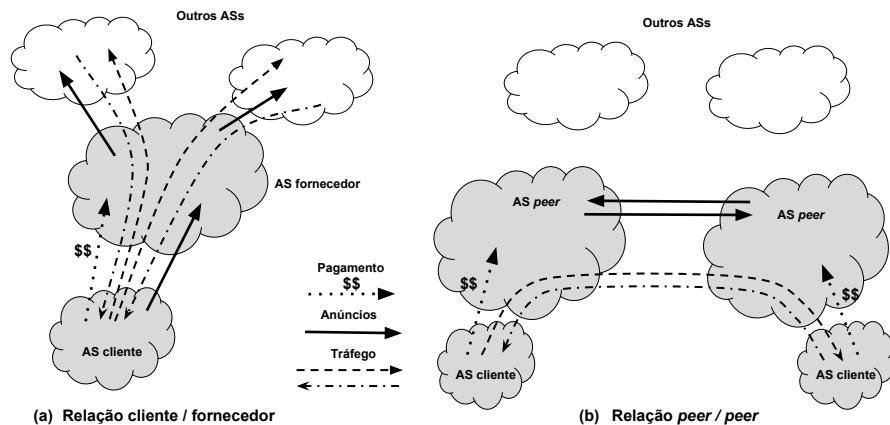


Figura 18.5: Relações entre ASs clientes e fornecedores (a) e relações entre ASs peers (b). Fluxos de anúncios dos prefixos, do tráfego e do dinheiro.

Quando os AS_a e AS_b têm uma relação do tipo *peer-to-peer*, quer dizer que AS_a está disposto a receber pela ligação a AS_b os pacotes originados em AS_b , ou nos seus clientes, e que se dirigem a AS_a , ou aos seus clientes, e vice-versa. Isso quer dizer que AS_a e AS_b anunciam mutuamente um ao outro os seus prefixos e os dos seus clientes. No entanto, quer AS_a quer AS_b não anunciam esse prefixos para terceiros ASs. Em geral, os serviços deste tipo não implicam pagamentos caso as quantidades de tráfego

enviadas em cada sentido sejam da mesma ordem de grandeza. No entanto, em caso de assimetria, é frequente os contratos de *peering* incluírem pagamentos.

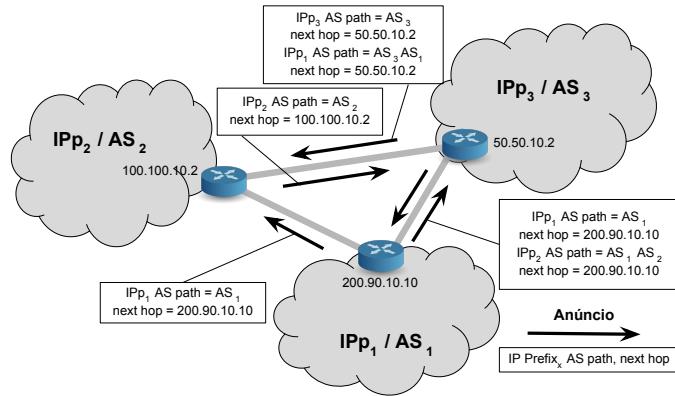


Figura 18.6: Utilização dos anúncios BGP para tomar decisões políticas sobre as escolhas de encaminhamento.

O controlo sobre os anúncios de prefixes através de *AS paths* BGP é muito flexível e permite implementar diversos tipos de políticas. A Figura 18.6 permite ilustrar esta faceta do protocolo. Por exemplo, o *AS₂* recebe anúncios da acessibilidade do prefixo *IPp₁* via o próprio *AS₁* e via o *AS₃*, no entanto, pode preferir alcançar esse prefixo passando pelo *AS₃* visto que este caminho é mais barato (em termos monetários) que o directo. Mas se este caminho deixar de estar disponível, pode optar pelo caminho directo para o *AS₁*. O protocolo BGP vai-lhe permitir fazer essas escolhas. Por outro lado, o *AS₁* prefere esconder ao *AS₂* que conhece um caminho para *IPp₃* e assim *AS₂* não pode usar essa via caso o seu canal para *AS₃* não esteja disponível, visto que não sabe que a mesma existe.

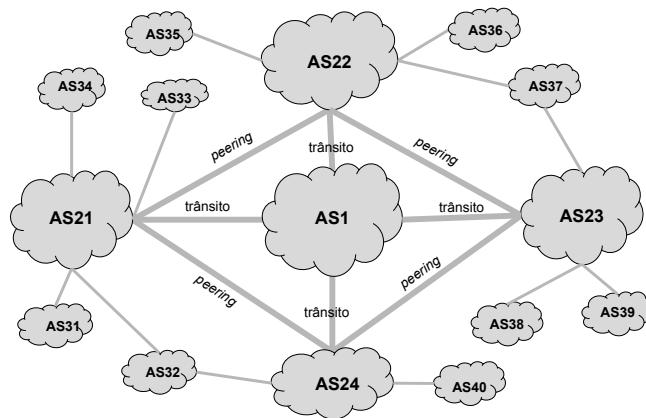


Figura 18.7: Um conjunto hipotético de ASs de diversos *tiers* com relações cliente / fornecedor e de *peering* entre eles.

Exemplo mais completo

O exemplo que se segue, ver a Figura 18.7, é também hipotético mas mais completo e corresponde a opções comuns no mundo dos operadores. O AS1 fornece trânsito pago aos AS21, AS22, AS23 e AS24. Estes três ASs fornecem trânsito pago aos seus clientes directos e têm relações de *peering* mútuas dois a dois. Os AS232 e AS237 são *multi-homed*. Os prefixos estão omitidos para maior clareza da figura, mas por hipótese cada AS tem um único prefixo, correspondente à sua rede e com o mesmo nome que o seu AS.

Discussão: anúncios realizados

A implementação destas políticas passa por os ASs AS31 até AS40 apenas anunciam o próprio prefixo para os seus fornecedores directos (lembre-se que cada AS apenas faz um anúncio sobre cada prefixo IP a cada um dos ASs a que está directamente ligado). Cada um dos *Tier 2* ASs (AS21 a AS23) anuncia os seus clientes directos para o AS1 (*Tier 1 AS*) e para os seus *peers*. O AS1 anuncia para os seus clientes de nível 2 todos os ASs que conhece, *i.e.*, todos os AS existentes na rede. Este AS representa a raiz de uma árvore que assegura a conectividade completa do conjunto. Os ASs do nível 2 têm caminhos para todos os ASs existentes, via os canais que os ligam aos seus clientes directos, ou via os canais que os ligam aos seus *peers*, e ainda para toda a restante rede via o AS1. Por isso anunciam todos os prefixos que conhecem aos seus clientes directos que assim passam a ter conectividade para toda a rede. Os clientes *multi-homed* necessitam destes anúncios para escolherem um AS *path* para cada destino. No entanto, os outros ASs clientes finais apenas precisam de uma rota por omissão (*default*) visto que não têm alternativas.

Discussão: políticas implementadas pelo anúncios

As políticas descritas conduzem à utilização de AS *paths* específicos. Por exemplo, o AS31 envia pacotes para o AS34 através do AS21, *i.e.*, pelo AS *path* [AS21, AS34]; envia pacotes para o AS35 via o AS *path* [AS21, AS22, AS35] visto que a ligação de *peering* é preferida à ligação a pagar via o AS1; e envia pacotes para o AS38 via o AS *path* [AS21, AS1, AS23, AS38] visto que essa é a única via de que dispõe para lá chegar (o *peering* não é transitivo).

Supondo que por algum motivo os canais que ligam o AS22 ao AS23 e ao AS1 ficarem inoperacionais, existem diversos caminhos que permitiriam ao AS36 chegar ao AS38, mas nenhum deles é utilizável. O AS *path* [AS36, AS22, AS37, AS23, AS38] não pode ser usado porque o AS37 não fornece trânsito aos seus dois fornecedores e o AS *path* correspondente não existe na rede. O AS *path* [AS36, AS22, AS21, AS1, AS23, AS38] não é utilizável porque o AS21 não fornece trânsito a AS22, apenas *peering*, e o AS38 não é cliente de AS21.

Discussão: alternativas de políticas

Em geral, os diferentes operadores são concorrentes e portanto, segundo a teoria de que entre empresas com fins lucrativos não “existem almoços grátis”, estas só fazem “favores” em troca de “pagamentos ou vantagens”. Por outro lado, se os clientes pagam pelos serviços de conectividade recebidos, só estão dispostos a fornecerem conectividade de trânsito, se os seus canais a suportassem, em troca também de “pagamentos ou vantagens”.

No entanto, os AS21 e o AS22 poderiam estabelecer um acordo mais completo em que, para além do *peering*, poderiam fornecer *backup* mútuo para o caso em que os canais que os ligam ao AS1 tivessem anomalias. Ou seja, caso os caminhos via AS1 não fossem acessíveis, passariam a usar o caminho via o parceiro. Para esse efeito, cada um deles deveria não só anunciar ao parceiro os seus clientes directos, mas também todos os ASs anunciados por AS1. Resta discutir como é que seria possível dar prioridades

aos diferentes caminhos, mas isso será mais claro quando discutirmos o protocolo BGP na Secção 18.3.

Com os protocolos convencionais, que só tomam decisões em termos de custos, só seria possível implementar uma política de conectividade total, usando sistematicamente todos os canais disponíveis em cada momento. Isso só seria adequado no caso em que o conjunto das redes envolvidas estivessem sob a mesma administração e autoridade. Ora no mundo dos operadores e dos clientes, esse tipo de relações não são adequadas dado o quadro de competição. O maior operador tem vantagens de escala que quer sempre aumentar para ter maior rentabilidade, ou até comprar os competidores.

Os ASs que só têm relações de *peering* com outros ASs e não são clientes de nenhum AS, dizem ASs *Transit-only*, visto que não dependem de anúncios “pagos”, isto é, recebidos ou propagados porque o AS paga a outro AS para esses efeitos. Geralmente, a maioria dos AS *Tier 1* são *Transit-only*.

Características do grafo dos ASs

O número de ASs presentes nas tabelas de encaminhamento do conjunto das ASs *Tier 1* subiu de 5.000 em 1999 para cerca de 35.000 em 2010. Uma subida de cerca de 700% em 10 anos. Esta subida está relacionada com alguma subida do número de operadores, mas sobretudo com a generalização de *multi-homing* e outras práticas que implicam dispor de um AS próprio e fazer encaminhamento BGP. No fim do primeiro trimestre de 2016, os ASs da DFZ são cerca de 55.000, uma subida inferior a 100% em 6 anos.

Como os *Internet peering points* existem nas cidades onde o tráfego é mais intenso, e se torna mais conveniente terminar os cabos de fibras ópticas de longa distância, nos respectivos edifícios existem numerosos *border routers* concentrados e estabelecem-se inúmeras ligações negociadas entre os mesmos.

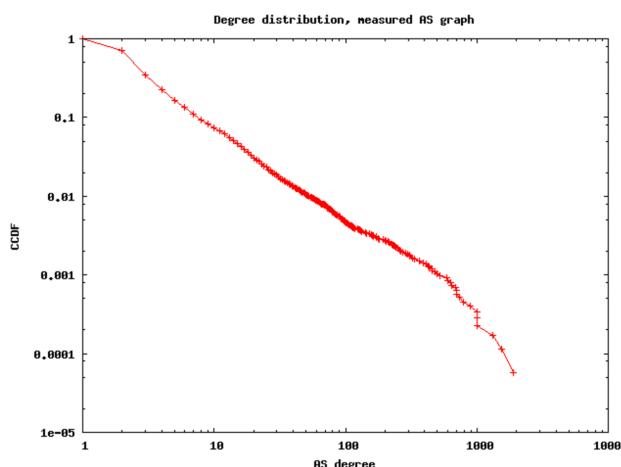


Figura 18.8: Distribuição cumulativa da probabilidade de um AS ter grau superior a um valor dado nas abcissas. Gráfico obtido no site <https://www.caida.org/research/topology/generator>.

Na prática verifica-se que o grau dos diferentes ASs, *i.e.*, o número de ligações de que cada um dispõe com os outros, é muito mais elevado nos ASs *Tier-1* do que nos outros. Por um lado estes ASs têm relações de *peering* entre eles e fornecem trânsito pago a muitos ASs de trânsito. Por outro, tal como na Web “*the winner takes it all*”, e

esses fornecedores concentram igualmente ligações de muitos clientes com *Stub ASs* de grande valor. A maioria dos ASs *Tier-2* têm várias centenas de ligações, quer a outros AS do mesmo ou de mais elevado *Tier*, quer também a muitos clientes. A grande maioria dos *Stub ASs* têm grau 1.

Existem diversos servidores de observação do funcionamento do BGP como por exemplo: <http://www.cidr-report.org> e <http://as-rank.caida.org>, que fornecem estatísticas publicamente acessíveis. Por exemplo, o gráfico da Figura 18.8, extraído do site da Caida no outono de 2016, mostra a distribuição cumulativa em percentagem do grau dos diferentes ASs. Assim, 100% dos ASs têm grau maior ou igual a 1 (um AS com grau 0 estaria isolado da Internet), menos de 10% têm grau superior a 10, enquanto que apenas uma percentagem inferior a 1% têm grau superior a 100.

A tabela 18.1 contém algumas informações sobre alguns ASs de grande dimensão, provavelmente todos do tipo *Tier-1*. Através da mesma é possível observar que os ASs mais importantes da Internet anunciam um elevado número de prefixos IP, e fornecem conectividade, directa ou indirectamente, a muitos outros ASs. A tabela inclui também vários ASs de menor *ranking* para comparação, e mostrar como a grande maioria dos 55.000 ASs são de facto *Stub ASs* que só dão acesso aos seus prefixos.

Tabela 18.1: Lista ordenada dos primeiros 10 ASs ordenados pelo número de ASs alcançados directa ou indirectamente, ou *AS customer cone* (o próprio, os clientes, os clientes dos clientes, ...) e prefixos IPv4 anunciados. A tabela foi obtida em 1 de Junho de 2016 a partir do site <http://as-rank.caida.org>. Incluem-se também alguns ASs de menor *rank* para comparação. O leitor não deve esquecer que o número total de ASs é superior a 50.000.

Rank	AS number	Nome da organização	AS customer cone	Número de prefixos IPv4
1	3356	Level 3 Communications, Inc.	29.494	224.970
2	174	Cogent Communications	23.299	172.963
3	1299	TeliaSonera AB	21.954	191.391
4	2914	NTT America, Inc.	18.991	174.304
5	3257	Tinet Spa	18.140	161.377
6	6762	TELECOM ITALIA SPARKLE	14.394	123.771
7	6453	TATA COMMS (AMERICA) Inc	12.300	135.127
8	6939	Hurricane Electric, Inc.	8.088	79.800
9	2828	XO Communications	6.251	60.271
10	1273	Cable and Wireless Worldwide	5.878	42.258
....
500	35819	Mobily-AS	49	636
501	10835	Visionary Communications, Inc.	49	473
....
999	13767	DataBank Holdings, Ltd.	21	134
10000	201952	Atel-Rybinsk LTD	1	22
....

Têm sido feitos inúmeros estudos sobre a estrutura profunda da Internet no que diz respeito à topologia do grafo dos ASs. No final deste capítulo serão fornecidas referências para alguns desses estudos e as fontes em que se baseiam.

Na Internet global o grafo de ligação entre os diferentes ASs é conhecido através dos prefixos IP e dos AS *paths* escolhidos e anunciado pelos ASs uns aos outros. A partir de *routers* da DFZ (*default free zone*) podem observar-se muitos desses AS *paths*, o que permite analisar as relações existentes entre os diferentes ASs e verificar que esses caminhos estão subordinados a relações comerciais ou de colaboração.

O grafo dos ASs é de grande dimensão pois no momento em que este livro foi escrito tem cerca de 55.000 nós e várias centenas de milhares de arcos. O número de ligações de cada AS, *i.e.*, o seu grau, é muito heterogéneo e com uma distribuição muito desigual.

Esta distribuição permite concluir que o conjunto dos ASs *Tier 1* é responsável directa ou indirectamente pela conectividade entre praticamente todos os ASs existentes.

18.3 Protocolo BGP

O protocolo BGP² é o protocolo que permite aos ASs directamente interligados trocarem entre si informações de encaminhamento. Um *router* que estabelece relações BGP com outros *routers* chama-se um BGP *speaker*. Geralmente os BGP *speakers* coincidem com os *border routers*, mas isso não é obrigatório.

As mensagens BGP são trocadas através de conexões TCP que se chamam sessões BGP (BGP *sessions*). Quando dois BGP *speakers* estabelecem uma sessão, trocam os anúncios que têm de comunicar um ao outro através de mensagens BGP. Depois, mantêm a conexão activa para, sempre que se produzem alterações relevantes nas suas BGP RIBs (BGP *Routing Information Bases*), enviarem actualizações um ao outro. A Figura 18.9 contém um gráfico da evolução da dimensão da BGP RIB.

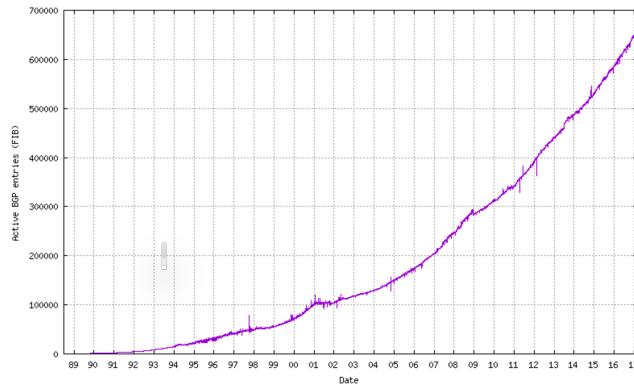


Figura 18.9: Evolução da dimensão da RIB BGP. Gráfico obtido no final de 2016 no site <http://bgp.potaroo.net/>.

Essas actualizações só contém alterações da informação transferida anteriormente. Ou seja, após a troca inicial de anúncios, a sessão só serve, enquanto durar, para enviar actualizações. Por isso os *routers* nas extremidades da conexão vigiam-se mutuamente através de mensagens de *keep alive*, que verificam se a conexão se mantém activa, o que garante que os anúncios enviados antes pelo vizinho são todos conhecidos.

Estas opções justificam-se porque a informação trocada pode ser muito volumosa e é pressuposto os *routers* memorizarem todas as rotas (identificadas por um prefixo e contendo um AS path e outros atributos) enviadas por cada vizinho, pelo que não é

² A descrição do protocolo que se segue baseia-se na sua versão 4, a versão em utilização no momento em que este livro foi escrito.

necessário refrescar toda a informação com origem num vizinho enquanto a sessão se mantiver operacional.

Se a sessão se quebrar, todos as rotas com origem no vizinho com que se perdeu o diálogo BGP são imediatamente suprimidas da BGP RIB. Nessa sequência, se o AS conhecer outras alternativas de rota para esses prefixos, a conectividade para os mesmos pode manter-se, senão é quebrada.

De cada vizinho, o BGP só recebe e só guarda uma única rota por ele anunciada. No entanto, os BGP *speakers* podem memorizar várias rotas diferentes para o mesmo prefixo, desde que recebidas de diferentes vizinhos. No entanto, como já foi referido, um AS só anuncia a um vizinho uma única rota com um AS *path* que deve ter origem no por si escolhido, modificado pelo concatenação do seu número de AS.

Sessões BGP externas e internas

Um AS pode ter mais do que um BGP *speaker*. Nesse caso, as rotas BGP seleccionadas por cada *speaker* têm de ser difundidas para que os diferentes *border routers* se coordenem e, na posse de informação mais completa, tomem as opções mais adequadas. Assim, ver a Figura 18.10, o BGP mantém dois tipos de sessões: As sessões eBGP, que são sessões externas mantidas com os BGP *speakers* dos outros ASs, e as sessões iBGP, ou sessões internas BGP, que são as sessões mantidas com os outros *routers* BGP do mesmo AS.

No iBGP os AS *paths* não introduzem ciclos internos pois cada *speaker* apenas propaga AS *paths* aprendidos do exterior. Assim, é necessário que cada *speaker* estabeleça uma sessão com cada um dos outros *speakers* do seu AS. Desta forma, as BGP RIBs de todos os *speakers* do AS contém as rotas seleccionadas por todos os outros *speakers* do AS.

Num AS com n destes *routers*, cada um deles tem de manter $n - 1$ sessões iBGP. Isso pode tornar-se muito pesado, pelo que foram introduzidos outros esquemas de fazer a difusão. Por exemplo, usando um único *router* especial como reflector de rotas (*route reflector*), ver o RFC 4456. Este *router* terá de manter $n - 1$ sessões iBGP, mas todos os outros apenas uma. Para além disso o reflector tem de evitar a introdução de ciclos internos ao AS.

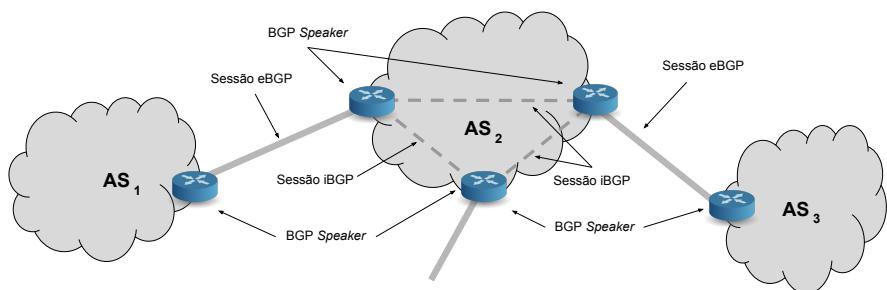


Figura 18.10: Sessões BGP estabelecidas pelos BGP *speakers* dentro do AS (sessões iBGP a tracejado) e os dos ASs vizinhos (sessões eBGP a traço grosso).

Através dos anúncios BGP os *routers* mantêm a informação das suas BGP RIBs actualizadas e tomam decisões de encaminhamento para os prefixos externos ao AS. Naturalmente, essa informação de encaminhamento tem também de ser injectada nos *routers* não BGP do AS através da colaboração entre os BGP *speakers* e os outros *routers* internos ao AS através da injeção de anúncios via os protocolos de encaminhamento do interior do AS (IGPs). Este aspecto será discutido no final da desta secção.

Estabelecimento das sessões BGP e mensagens BGP

Os *routers* BGP são parametrizados manualmente sobre as sessões que mantêm com os seus vizinhos. Uma sessão BGP passa pelas fases indicadas na máquina de estados da parte esquerda da Figura 18.11. Primeiro os *routers* estabelecem a conexão TCP e trocam diversos parâmetros da sessão e depois iniciam a troca de anúncios de rotas. As mensagens BGP são enviadas sobre TCP, como sequências de bytes, e são as indicadas a seguir.

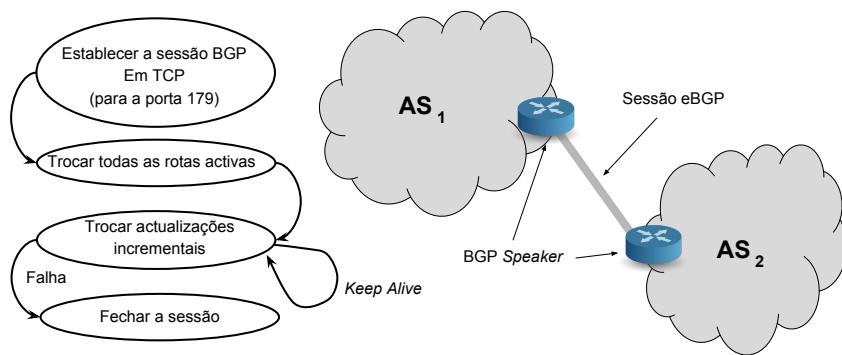


Figura 18.11: Estabelecimento das sessões BGP e fases da sessão.

Open Esta é a primeira mensagem enviada na sessão após o seu estabelecimento. Contém diversos parâmetros como a versão do protocolo (em 2016 a versão mais actual é a BGP 4, ver os RFCs 1771 e 4271), número do AS, identificador do *router*, duração do alarme para a deteção da interrupção da sessão e outros parâmetros opcionais.

Keepalive Esta mensagem serve para as partes detectarem se continuam activas e presentes na sessão.

Notification Serve para comunicar códigos de erro. Uma sessão que não é quebrada por um alarme é fechada por uma destas mensagens.

Update É através desta mensagem que os dois *speakers* trocam informações de encaminhamento. A mesma será discutida de seguida.

Um *router* BGP contém na sua RIB BGP uma tabela com todos os prefixos IP que conhece via os anúncios BGP recebidos e, para cada um deles, as rotas que pode usar para lá chegar. Destas, seleciona uma para utilização, e sempre que há uma alteração da rota seleccionada, comunica-a aos seus *peers* através de uma mensagem **Update**. A Figura 18.12 contém uma visão alto nível dos parâmetros desta mensagem. Esta permite o envio de várias actualizações de uma só vez pois tem vários campos de comprimento variável.

Na primeira parte da mensagem, correspondente às **withdrawn routes**, são indicados prefixos IP para os quais o emissor deixou de ter uma rota (um ou mais prefixos IP envolvendo uma ou mais rotas suprimidas). A seguir é indicada uma nova rota, para um ou mais prefixos IP, que o emissor instalou. A rota começa por um campo designado **path attributes field**, que indica um AS *path* e outros atributos sobre a rota e, finalmente, na parte **Network layer reachability info**, são enviados os prefixos que a mesma permite alcançar.

Alguns dos atributos de uma rota são os seguintes: **Origin** (opções: obtida via o IGP, via o EGP ou por outra forma, como por exemplo por encaminhamento estático), **AS path** (sequência de um ou mais ASn *numbers* terminado necessariamente pelo AS

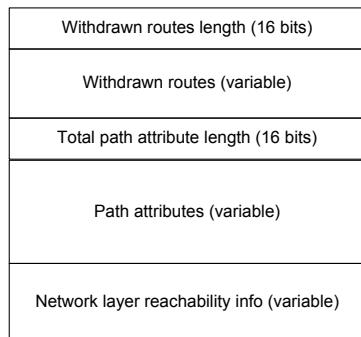


Figura 18.12: Mensagem BGP Update

number do router anunciante) e **next-hop** (o endereço do BGP *speaker* para o qual devem ser enviados os pacotes alvo da rota). Assim, um **Update** pode suprimir vários prefixes para os quais o emissor deixou de ter uma rota, e anunciar vários prefixes que partilham uma rota com um conjunto de atributos como o *AS path*, **next-hop**, *etc.*

O BGP é o protocolo que assegura a conectividade da Internet global e que tem como requisitos ser capaz de lidar com centenas de milhar de prefixos IP e várias dezenas de milhar de redes interligadas, cada uma das quais com políticas específicas de encaminhamento. Apesar desta escala, pretende-se que convirja rapidamente.

O protocolo é executado por *routers* especiais, ditos **BGP speakers**. Estes *routers* coincidem geralmente com *border routers*, os *routers* que asseguram a interligação dos AS. O diálogo entre os BGP *speakers* é assegurado através de **sessões BGP (BGP sessions)** que são ponto-a-ponto e utilizam conexões TCP. Os anúncios são incrementais e só comunicam actualizações (novas rotas ou supressão de rotas).

Funcionamento do protocolo e seleção dos caminhos

A Figura 18.13 mostra como a rota para o prefixo 193.136.0.0/15, propagada inicialmente pelo AS1 (à esquerda), se transforma em várias rotas alternativas conforme vai sendo propagada pelos diferentes ASs que aceitam fornecer trânsito para esse prefixo. Assim, quando o AS6 (à direita) recebe as mesmas, vai dispor de duas rotas para o prefixo. A questão que se coloca então é saber como é que esse AS se decide por uma delas. Um protocolo de encaminhamento IGP usaria para esse efeito a noção de custo. No entanto o BGP não tem, por construção, essa noção numa forma convencional, e precisa de satisfazer outros requisitos.

O BGP não tem nenhuma noção convencional de custo pois cada AS pode usar diferentes IGPs e diferentes funções de custo. Por outro lado, um AS pode cobrir um continente e pode-se atravessá-lo de 1 a 50 ms ou mais, dependendo da entrada e saída usadas. A única função de custo que o BGP pode usar é comparar o número de ASs atravessado pela rota. No exemplo da Figura 18.13 este critério conduziria à seleção pelo AS6 da rota via o AS5 visto que esta tem 3 ASs de comprimento, inferior à rota via o AS4 que atravessa 4 ASs. No entanto, o AS não sabe se isso conduz ou não a um menor tempo de trânsito, ou uma maior capacidade da rota, ou a uma rota de melhor qualidade pois a probabilidade de um pacote se perder é menor. Para isso ter-se-ia de conhecer as características dos ASs atravessados para os comparar.

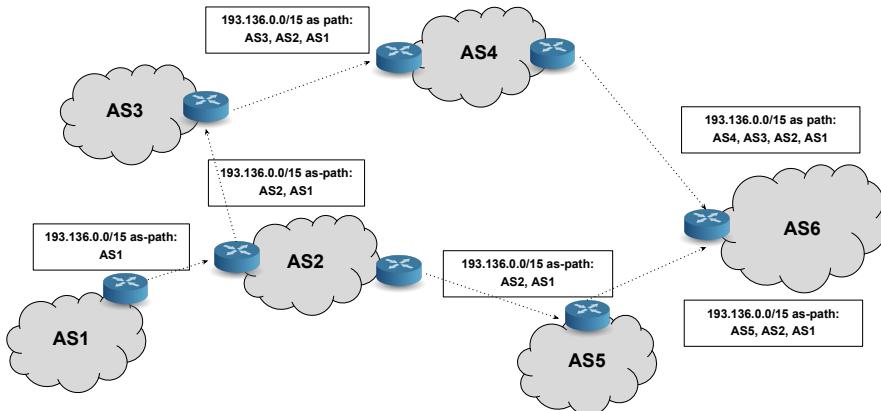


Figura 18.13: Propagação dos AS *paths* através dos ASs e recepção por um AS de mais do que uma rota para o mesmo prefixo.

No entanto, é aqui que entram em jogo outras preferências como por exemplo as ditadas por razões comerciais ou políticas. A norma do protocolo BGP fixa um conjunto de procedimentos a que qualquer *router* BGP deve obedecer para seleccionar rotas, mas deixa em aberto os fabricantes introduzirem mecanismos sofisticados em algumas delas, baseados por exemplo em linguagens de manipulação de *updates*.

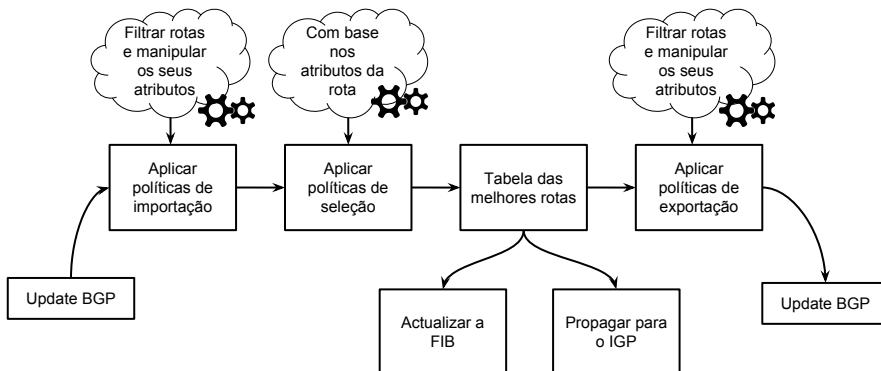


Figura 18.14: Passos seguidos desde que um BGP *update* é recebido até que o mesmo se repercuta na FIB do *router*, é propagado aos outros *routers* através do IGP e dá origem a outro(s) *update(s)* enviados para outros ASs.

A Figura 18.14 mostra os passos seguidos após a recepção de uma rota até que a mesma é tomada em consideração ou não, e eventualmente propagada aos vizinhos.

Inicialmente os atributos da rota são analisados e é verificado se a mesma é aceite ou não. Se for aceite, pode receber atributos suplementares. Existem várias razões que podem levar um *router* a ignorar certas rotas. Por exemplo, porque a mesma vem de um fornecedor mas contém o AS de um seu cliente, ou porque atravessa um AS que se pretende evitar. Uma política bastante comum consiste em filtrar todas as rotas enviadas por um cliente sobre cujos prefixos este não tem autoridade. O resultado

deste processo pode consistir em ignorar a rota ou, em alternativa, fixar-lhe o valor do atributo **LOCAL_PREF** (preferência local) afectando-lhe uma preferência mais baixa.

Por exemplo, as rotas dos *peers* devem ser preferidas às dos fornecedores por searem mais baratas, mas as rotas recebidas via os fornecedores devem manter-se como alternativa menos prioritária para suprirem a falha do canal para o *peer*. Estas regras de filtragem e modificação são expressas em linguagens que variam na sintaxe e sofisticação de fabricante para fabricante, visto que a norma não as define, apenas define os atributos das rotas.

Se a rota for aceite é memorizada e em seguida as diferentes alternativas recebidas dos diferentes vizinhos são comparadas para seleção da preferida pelo *router*. O processo de comparação das rotas usa diversos critérios aplicados sequencialmente, ver a Tabela 18.2. Passa-se ao critério seguinte quando a aplicação do corrente não permite discriminar as rotas em comparação e eleger a melhor delas.

Tabela 18.2: Critérios normalizados pelo BGP para comparar diferentes rotas para o mesmo grupo de prefixos IP

Prioridade	Atributo	Descrição, exemplo ou explicação
1	LOCAL_PREF	<i>Local preference value attribute: policy decision</i> <i>e.g., preferir as rotas dos peers às dos providers</i>
2	AS_PATH	<i>Shortest AS-PATH</i> Preferir as rotas com um número inferior de ASs
3	MED	<i>Lowest Multi-Exit-Discriminator</i> (MED) Se contratual, seguir as indicações do outro AS
4	IGP Path	<i>Closest Next-hop</i> Usar <i>Hot-potato routing</i>
5	eBGP > iBGP	<i>External route better than internal</i> Preferir a rota recebida de outro AS
6	Router ID	<i>Smallest Next-hop IP Address</i> À falta de melhor

O primeiro critério é exactamente o da preferência local. É dada preferência a rotas que no passo anterior foram definidas como mais interessantes. Caso este critério não permita a seleção de uma rota, usa-se o critério seguinte que consiste em comparar o comprimento do AS path. É claro que, tal como a Figura 18.15 ilustra, comparar o número de ASs não é o mesmo que comparar rotas do ponto de vista das métricas convencionais dos IGPs.

O critério seguinte tem a ver com o atributo *Lowest Multi-Exit-Desriminator* (MED) que é fixado pelo vizinho que comunicou a rota. Este critério permite a um AS indicar a um vizinho com o qual tem mais do que uma ligação, qual a que prefere que este use para lhe enviar tráfego por isso lhe ser mais conveniente. Caso o AS que recebe as rotas estiver obrigado contratualmente em seguir esta indicação, a mesma deverá ser respeitada.

A alternativa consiste em usar a rota tal que o *next-hop* indicado pelo vizinho está a menor distância do AS local do ponto de vista do IGP. Este tipo de critério conduz ao chamado *hot-potato routing*, um dos principais responsáveis por as rotas na Internet global não serem necessariamente simétricas.

Hot-potato routing

Quando um AS recebe um pacote em trânsito, deve tentar entregá-lo o mais depressa que possível ao próximo AS, como se de uma batata quente se tratasse, pois menos

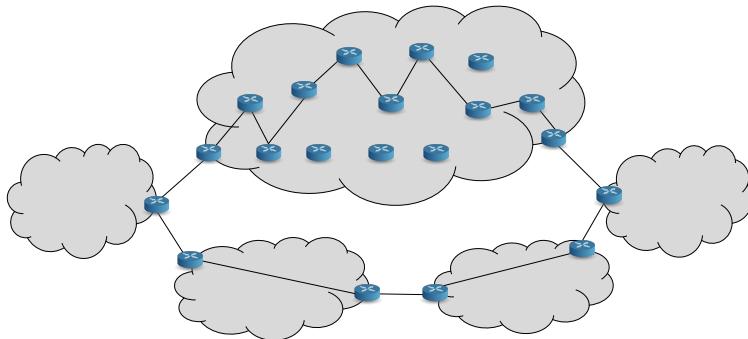


Figura 18.15: Comparar rotas pelo número de ASs pode revelar-se enganador.

recursos gasta com o mesmo. Quando a política seguida consiste em ignorar qualquer indicação enviada pelo vizinho e seguir a política do *closest next-hop*, quer dizer que a rota que é seleccionada é a que corresponde ao *router* mais próximo do vizinho. Se dois ASs *peers* de grande dimensão usarem esta política, as rotas seguidas pelos pacotes resultam assimétricas, ver a Figura 18.16.

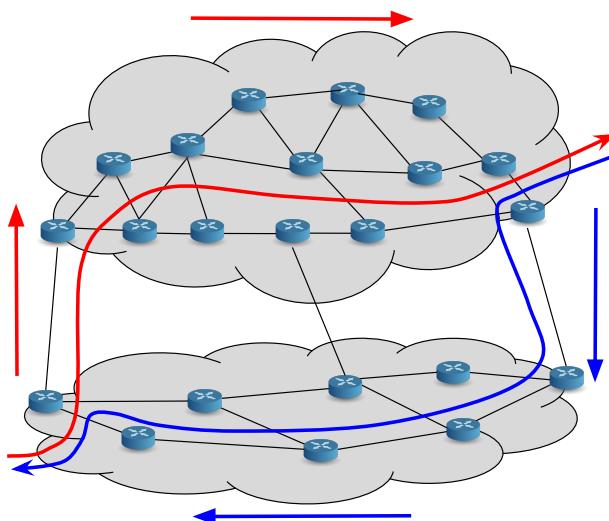


Figura 18.16: Política *hot-potato routing* e rotas assimétricas.

Finalmente, o *router* pode dar preferência a rotas recebidas dos vizinhos externos do que a rotas recebidas de outros BGP *speakers* do mesmo AS, e depois, à falta de melhor critério pode usar um factor arbitrário de desempate, que é grosso modo equivalente a tirar à sorte.

Uma vez a rota seleccionada, a mesma é repercutida nas FIBs e pode ser exportada para os outros ASs via eBGP. Em qualquer caso, deve ser propagada por iBGP aos outros BGP *speakers* do AS que eventualmente existam.

Políticas de exportação de rotas

Antes de uma rota seleccionada ser propagada para o exterior do AS é sujeita a filtragem e afinação dos seus atributos. Existem vários razões para não exportar rotas, nomeadamente, como já vimos, o não querer receber tráfego de um vizinho dirigido às mesmas. Por exemplo, não exportar rotas recebidas de um *peer*.

Mesmo quando uma rota é exportada, o seu AS *path* é modificado acrescentando o número do AS local. Uma transformação suplementar consiste em acrescentar várias vezes o número do AS local (*repeated AS prepending*) que conduz a um aumento artificial da dimensão desse AS *path*. Esta transformação é frequentemente usada pelos ASs *multi-homed* pois permite-lhes tentar influenciar qual o fornecedor que é escolhido pelo resto da Internet para lhes enviar pacotes ou fazer distribuição de carga entre as diversas ligações externas que possui.

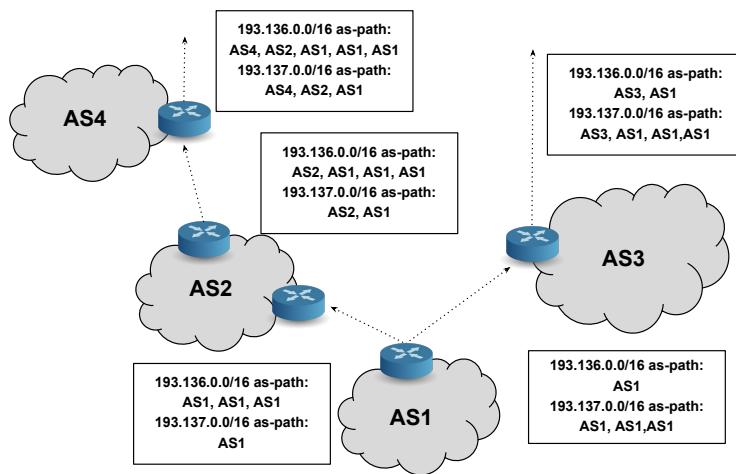


Figura 18.17: Utilização de *repeated AS prepending* para influenciar a distribuição do tráfego de entrada num AS *multi-homed*.

Na Figura 18.17 ilustra-se uma utilização desta técnica para fazer distribuição de carga. O prefixo do AS1 (193.136.0.0/15) foi dividido em dois (193.136.0.0/16 e 193.137.0.0/16) e cada um deles é anunciado de forma diferente a cada um dos fornecedores. Desta forma, à primeira metade dos endereços do prefixo é dada prioridade ao caminho via o AS3, enquanto que à segunda metade dos endereços do prefixo é dada prioridade via o AS2. Este tipo de procedimento leva ao aumento do número de prefixes distintos anunciados na Internet global mas o BGP não dispõe de outros mecanismos para controlar a distribuição de carga.

Convergência do BGP

A convergência do protocolo BGP é um processo que em certas circunstâncias é bastante demorado dado o volume de informação envolvido. Com efeito, quando um canal de interligação de dois ASs vai abaixo, os *border routers* têm de desencadear diversas operações envolvendo bastante processamento.

Em primeiro lugar têm de suprimir todas as rotas da sua BGP RIB que foram anunciadas pelo ex vizinho. Em seguida, caso as rotas suprimidas tivessem sido seleccionadas como as melhores, é necessário enviar supressões de rotas por eBGP e iBGP e actualizar os protocolos IGP usados visto que os *routers* convencionais do AS devem também ser notificados da ausência dessas rotas e, finalmente, é necessário seleccionar uma rota alternativa e, caso a mesma exista, eventualmente propagá-la aos ASs

vizinhos e ao IGP. Como alguns ASs partilham centenas de rotas, algumas avarias de canais, ou simples reinicializações de sessões ou eventuais problemas de software de BGP *speakers* podem desencadear muitas actualizações.

Ao contrário de um IGP do tipo vector de distâncias, ver a Secção 16.4, mesmo que a conectividade para um prefixo se mantenha e o número de ASs para lá chegar não se altere, a simples alteração do caminho tem de ser comunicada e é propagada como uma alteração.

Nos *site* <http://potaroo.net> é possível ver que o número de alterações de rotas recebidas por um *border router* da DFZ é muito grande e possui picos que podem atingir milhares de actualizações por segundo. A caixa abaixo mostra um extracto dessas estatísticas extraídas do *site* <http://bgpupdates.potaroo.net>.

```
7 Day BGP Profile: 8-February-2017 00:00 - 14-February-2017 23:59 (UTC+1000)
```

```
Number of BGP Update Messages: 1952059
Number of Prefix Updates: 4607549
Number of Prefix Withdrawals: 262619
Average Prefixes per BGP Update: 2.49
Average BGP Update Messages per second: 2.82
Average Prefix Updates per second: 7.05
Peak BGP Update Message Rate per second: 2030 (07:51:56 Mon, 13-Feb-2017)
Peak Prefix Update Rate per second: 14538 (07:50:24 Mon, 13-Feb-2017)
Peak Prefix Withdraw Rate per second: 8498 (07:51:56 Mon, 13-Feb-2017)
Prefix Count: 671654
Updated Prefix Count: 244064
Stable Prefix Count: 427590
Origin AS Count: 56554
Updated Origin AS Count: 26035
Stable Origin AS Count: 30519
Unique Path Count: 291667
Updated Path Count: 170597
Stable Path Count: 121070
```

Por esta razão os ASs de grande dimensão fazem um investimento muito significativo nas suas redes de forma a darem-lhes estabilidade e também usam *border routers* de muito alta qualidade e robustez. Felizmente, os padrões reais mostram que a maioria dos ASs apresentam uma grande estabilidade e que a maioria das anomalias se concentram em relativamente poucos ASs, como se pode verificar no mesmo *site*, que contém o *ranking* dos ASs que enviam mais actualizações.

Caso a quebra de uma sessão BGP seja exclusivamente baseada na deteção da ausência de mensagens *keep-alive*, o BGP exige que haja 3 falhas sucessivas. Logo, a deteção da quebra da sessão pode levar bastantes segundos.

Para evitar sucessivas reconfigurações em cascata, o BGP, tal como normalizado em 2006 pelo RFC 4271 (BGP), impunha um limite mínimo de tempo entre o envio de duas actualizações seguidas sobre a mesma rota. Este *timer* designa-se por MRAI (*Minimum Route Advertisement Interval*), devia valer 30 segundos para o eBGP e 5 segundos para o iBGP, e devia também aplicar-se às simples supressões de uma rota. O racional por detrás deste limite era tentar que o BGP *speaker* receba outras actualizações expectáveis da rota e evite enviar actualizações que teria de anular logo a seguir. Mais à frente é ilustrada uma situação em que a convergência só se dá depois de várias trocas de mensagens, apesar de os prefixos da rota estarem inacessíveis logo desde o início.

Apesar de mais recentemente se considerar que esses valores são muito altos e que devem ser adaptados ao contexto de cada AS, o facto é que o valor deste período mínimo tem um impacto decisivo na convergência. Por outro lado, quando existe instabilidade numa rota, o RFC 4271 recomendava também a técnica *Route Flap Damping* (RFD), que resultava em deferir ou ignorar as actualizações da mesma rota durante um período de tempo ainda mais alargado, para compensar a instabilidade.

No entanto, algumas observações mais recentes vão também no sentido de considerar que este algoritmo é demasiado agressivo e por vezes tem como resultado atrasar inutilmente a convergência que chega a levar vários minutos, ver a Secção 18.4.

O problema da convergência do BGP continua a ter actualidade e estar sujeito a controvérsia, mas hoje em dia procura-se privilegiar a rapidez da convergência, em eventual detrimento da instabilidade dos *routers* BGP. No final do capítulo são apresentadas algumas referências sobre este problema cuja discussão ultrapassa o âmbito deste livro.

No que se refere à convergência, vamos apenas chamar a atenção para o facto de que esta é diferente da dos protocolos de encaminhamento pelo melhor caminho que foram apresentados no Capítulo 16. Como o BGP memoriza obrigatoriamente rotas alternativas, isso pode ser usado para encontrar imediatamente uma alternativa e manter a conectividade.

No entanto, em certas circunstâncias isso apenas serve para alargar a convergência por um processo designado *path exploration*, como é ilustrado na Figura 18.18.

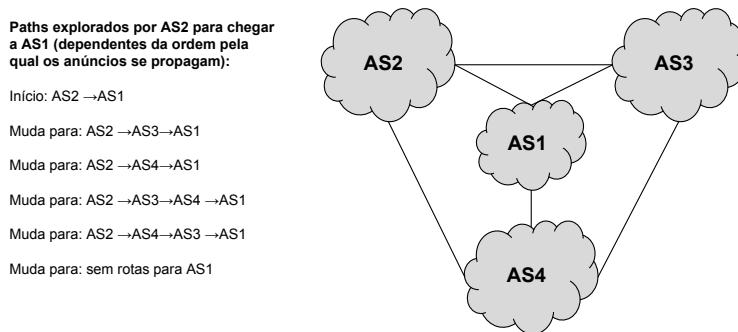


Figura 18.18: Instabilidade introduzida pela *path exploration* quando o AS1 deixou de estar acessível. Evolução dos caminhos usados por AS2 para chegar a AS1 até concluir que este deixou de estar acessível. Uma evolução semelhante passa-se também com o AS3 e o AS4.

O AS1 é cliente dos AS2, AS3 e AS4. Estes são *peers* de forma transitiva e anunciam AS *paths* para AS1 sem restrições. Inicialmente todos os ASs da figura têm conectividade para o AS1 e usam os canais directos que os ligam a AS1. No entanto, mais tarde o AS1 deixa de estar totalmente acessível e esses canais deixam de estar operacionais. Então todos os ASs mudam para caminhos alternativos. Como todos os *peers* fazem o mesmo, enviam actualizações das novas rotas usadas. No entanto, dependendo da ordem de recepção e tratamento das actualizações, a sequência de actualizações vistas pelo AS2 pode ser a que a figura ilustra visto que as relações de *peering* são transitivas. Este processo pode levar várias dezenas de segundos ou mesmo minutos.

Em conclusão, o encaminhamento com base em vectores de caminhos evita a anomalia designada *count to infinity*, ver o Capítulo 16, no entanto não deixa de prolongar a convergência de forma indesejável devido à exploração de diversas alternativas de caminhos até concluir que afinal nenhum está disponível.

Interacção entre o BGP e o encaminhamento IGP

Quando os pacotes dirigidos aos prefixos de um AS chegam aos seus *border routers*, estes usam o IGP para os fazer chegar ao destino final. Com efeito estes *routers* executam simultaneamente os protocolos BGP e IGP. A FIB é comum mas as RIBs são distintas, apesar de interagirem.

No entanto, como se passa o processo inverso? Como é que os *routers* internos ao AS fazem chegar os pacotes dirigidos ao exterior do AS aos *border routers*?

Num *Stub AS* com uma única ligação ao exterior, o *border router* pode injectar no IGP uma rota por omissão (*default*), que é suficiente para o objectivo pretendido. Num AS *multi-homed*, com vários *border routers*, que o ligam a diversos provedores de acesso à Internet global, cada um deles pode injectar no IGP uma rota por defeito. No entanto, este esquema não é suficiente para que os pacotes dirigidos ao exterior sejam encaminhados para o *border router* mais próximo do destino.

Para que fosse possível introduzir alguma racionalidade no caminho escolhido internamente para os pacotes dirigidos ao exterior, seria necessário introduzir no IGP a totalidade das rotas adquiridas pelo eBGP num processo de encaminhamento que se designa por *default free*. É claro que isso exige *routers* internos de grande capacidade visto que a tabela global de prefixos contém centenas de milhar de prefixos. Tal pode revelar-se contraproducente e até inutilmente caro.

Existem diversas alternativas possíveis. Uma primeira passa por injectar no IGP apenas algumas rotas mais importantes, responsáveis pelo maioria do tráfego para o exterior. Por exemplo todos os prefixos dos *peers* nacionais e dos fornecedores de conteúdos mais usados. Outra hipótese é usar um esquema hierárquico em que o conjunto dos *routers* do *backbone*, que inclui todos os *border routers*, são *default free*. Em contrapartida as diferentes zonas ligadas ao *backbone* usam rotas por omissão para chegarem ao mesmo. Naturalmente, os *backbones* dos AS de trânsito são todos necessariamente *default free*.

Esta discussão põe mais uma vez em evidência como a hierarquização do encaminhamento a todos os níveis é fundamental para a escalabilidade do mesmo. Por outro lado, ela também mostra que qualquer rede de dimensão razoável, e ligada à Internet por várias vias, usa simultaneamente vários métodos de encaminhamento interno e externo em cooperação.

O protocolo BGP possui diversos mecanismos que permitem a implementação de encaminhamento com base em políticas, nomeadamente: controlo da importação de rotas, manipulação de atributos das rotas, regras sofisticadas para comparação de rotas, controlo da exportação de rotas, *etc.*

Adicionalmente, o protocolo tem mecanismos para permitir aos diferentes *speakers* do mesmo AS conhecerem as rotas usadas pelos outros (*internal BGP* ou *iBGP*) e interage com os protocolos internos ao AS (os *Internal Gateways Protocols* ou *IGPs*).

Encaminhamento BGP e segurança

Todos os protocolos de encaminhamento têm problemas de segurança dado que um “*router*” intruso tem hipóteses de sabotar o encaminhamento e atrair para si tráfego alheio, ou fazer ataques de negação de serviço. Na maioria das redes sob controlo de uma só organização, estes ataques são possíveis mas mais difíceis, pois os gestores da rede controlam todos os equipamentos que podem originar anúncios, assim como as respectivas vizinhanças. Por outro lado, é relativamente fácil parametrizar os equipamentos para só aceitarem anúncios de vizinhos legítimos.

O BGP executa na Internet global e quase sempre um *border router* tem sessões eBGP com *border routers* sob controlo de outras organizações. Sem desobedecer ao RFC 4271, um *router* pode ser configurado para anunciar um número de AS qualquer, e originar quaisquer prefixos, seja qual for a sua dimensão (e.g., /30). Se o vizinho não estiver parametrizado para filtrar este anúncios, vai aceitá-los como bons. Por isso muitos ASs de trânsito tomam especiais cuidados em só aceitarem de *Stub ASs* clientes

anúncios com os seus prefixos e número de AS, o que é facilmente implementado usando filtros de aceitação de rotas externas. Geralmente as informações necessárias fazem parte dos anexos técnicos dos contratos.

No entanto, quer porque esses controlos são caros do ponto de vista da gestão, quer porque conforme se sobe na hierarquia dos ASs de trânsito as regras de filtragem são cada vez mais complexas, e não existe nenhuma forma segura de as estabelecer de forma automática, na prática é fácil de introduzir falhas no controlo.

A maioria das RIR (*Regional Internet Registries*) mantém bases de dados de acesso público sobre os ASs e as suas políticas. Estas bases de dados são acessíveis via o serviço **whois**. Infelizmente, muitas dessas bases de dados estão incompletas ou desactualizadas e não podem ser usadas para automatizar os filtros.

Esta situação proporciona a introdução de anúncios errados por acidente ou malícia. Por exemplo, em 1977 um AS de uma universidade introduziu milhares de anúncios de prefixos /24 errados por acidente, e atraiu para si o tráfego que lhes era dirigido provocando um “buraco negro” na sua conectividade que durou várias horas. Também é célebre o caso em que o AS da Telecom do Afeganistão introduziu um anúncio em que se apresentava como o originador do prefixo de servidores do YouTube, atraindo para si o tráfego que lhes era dirigido e impedindo o seu acesso normal.

A Figura 18.19 ilustra como tem lugar um ataque deste tipo. O atacante anuncia uma rota para um prefixo mais específico e, dada a regra *longest prefix matching*, atrai para si o tráfego correspondente a esse prefixo mais específico. No exemplo, um atacante no AS1 consegue atrair para o seu AS o tráfego do servidor de endereço IP 193.136.126.43 ligado ao AS2. O AS1 para esse efeito anuncia o prefixo 193.136.126.32/27, que por ser mais específico que o prefixo 193.136.0.0/15, atrai para si o tráfego dirigido ao servidor legítimo. Mesmo que não envie pacotes de resposta utilizando um endereço origem falso, conseguiria impedir desta forma o acesso ao servidor legítimo.

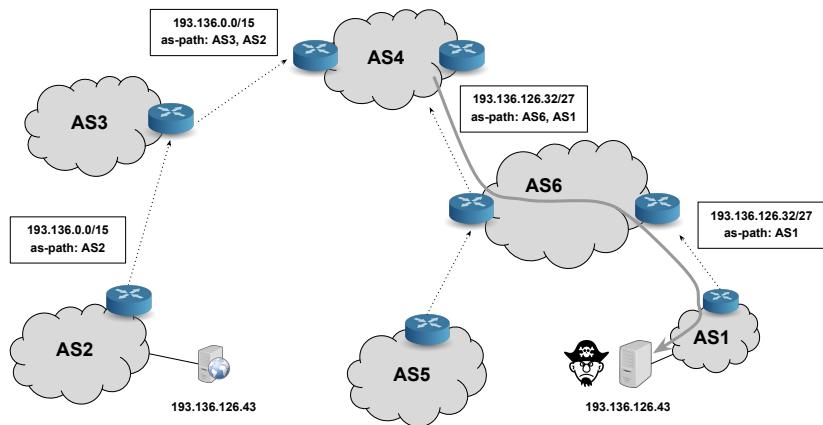


Figura 18.19: Ataque ao encaminhamento BGP através do anúncio pelo AS1 de uma rota para um prefixo mais estreito que não lhe pertence. A regra *longest prefix matching* desvia para o AS1 o tráfego dessa parte do prefixo legítimo anunciado pelo AS2.

Uma vez identificado que o ataque está a utilizar o protocolo BGP, é relativamente fácil perceber a origem do mesmo seguindo o AS *path* da rota. Esta faceta talvez explique porque razão estes ataques não são generalizados. Por outro lado, os operadores de trânsito que os permitissem de forma continuada, acabariam por ser

colocados em listas negras pelos outros operadores. Com efeito, se a generalidade dos operadores passassem a filtrar as rotas que contivessem o seu número de AS, isso levá-lo-ia à falência. Apesar da fragilidade óbvia do BGP, quando comparada com a sua responsabilidade em assegurar o funcionamento da Internet global, a verdade é que até ao momento em que este livro foi escrito não existem ataques ao BGP de forma continuada e generalizada.

Com efeito, a maioria dos *Stub ASs* são clientes de ASs que fornecem trânsito. Ora os prefixos IP anunciados por cada cliente são geralmente em pequeno número e a sua lista faz parte de contratos. Por isso a grande maioria dos ASs coloca filtros aos anúncios enviados pelos seus clientes, quer por razões de segurança, quer sobretudo por motivações comerciais. Por esta razão, a maioria dos ASs até não conseguem fazer anúncios falsos e existem motivações por parte dos seus fornecedores para o impedir. Adicionalmente, a percepção pelos ASs de *Tier-2* de que o risco de propagarem anúncios falsos, é superior aos potenciais lucros, é motivação suficiente para tentarem exercer controlo sobre os anúncios que recebem e propagam.

Por estas razões, os atacantes mais comuns preferem outros tipos de ataques mais subtils, focados em objectivos mais precisos e sem impacto generalizado sobre os operadores que os “apoiasssem”. De qualquer forma, o problema persiste, e num quadro de fragmentação da Internet, ou de ciber guerra generalizada, os ataques via BGP serão mais realistas, e com efeitos potencialmente devastadores, pela instabilidade no encaminhamento global que introduziriam.

O BGP é um protocolo que propaga prefixos IP e rotas para os alcançar. Por defeito dois BGP *speakers* autenticam-se um ao outro. No entanto, se não for exercido controlo sobre os anúncios recebidos e propagados, é relativamente fácil introduzir instabilidade no encaminhamento global e impedir o acesso a regiões do espaço de endereçamento. O protocolo apenas define os mecanismos que permitem a introdução desses controlos. No entanto, não existem mecanismos que os tornem obrigatórios.

Apesar destas fragilidades, o quadro em que o protocolo opera minora a probabilidade de os controlos não serem exercidos e, por isso, não existem de forma frequente, na realidade actual, ataques conduzidos por esta via. Só num quadro de agressividade entre operadores ou blocos políticos esses ataques seriam mais prováveis.

Existem várias propostas de tentar tornar o BGP em geral mais seguro (*e.g.*, [Kent et al., 2000; Charles, 2000]). Infelizmente, as mais eficazes exigem requisitos de gestão e de capacidade computacional que, na ausência de incentivos claros, retardam a generalização da sua adopção.

Implementação de IP Anycasting

Nos capítulos anteriores foi referida a utilização de IP Anycasting como uma forma de dirigir um pacote para um grupo de servidores, selecionando automaticamente o servidor mais próximo do emissor do pacote. Por exemplo, foi discutida esta utilização para endereçamento dos *root servers* do DNS, em que um único endereço IP representava um conjunto de diferentes servidores, localizados em diferentes ASs, ver a Secção 11.3. Foi também referida a utilização de IP Anycasting como forma de endereçar os *edge servers* de uma CDN mais próximos do emissor, ver a Secção 13.3.

A forma mais simples de implementar IP Anycasting consiste em associar um prefixo IP específico, e um AS do proprietário do serviço, a um conjunto dos endereços IP Anycasting, e anunciar esse prefixo e AS simultaneamente a partir de várias redes distintas. Cada um dos outros ASs vai seleccionar apenas um AS *path* para o prefixo

anunciado, naturalmente o que corresponder à sua política de encaminhamento. Desta forma, mediante a seleção realizada, a instância do serviço alcançada será a que estiver mais próxima (em termos das métricas de encaminhamento usadas) do emissor, em função das políticas seguidas pelos ASs atravessados. O mesmo esquema pode ser implementado dentro de um AS, usando um esquema equivalente com os protocolos de encaminhamento usados no interior do AS. No fundo a técnica é aplicável com qualquer protocolo de encaminhamento, ficando o âmbito do grupo anycasting subordinado ao âmbito dos anúncios realizados via o protocolo de encaminhamento usado.

Esta implementação do IP Anycast tem por repercussão o aumento da dimensão das tabelas de encaminhamento BGP mas não existem alternativas à mesma quando se pretende obter o seu efeito a nível global da Internet.

O anycasting funciona sem problemas em aplicações baseadas em UDP e com interações rápidas entre os clientes e os servidores, como é o caso do protocolo do DNS. No caso do TCP, se durante a conexão TCP variar o servidor em que a mesma termina, existirão problemas e a conexão não sobreviverá a essa alteração.

O anycasting é implementado realizando um conjunto de anúncios equivalentes, a partir de diferentes pontos da rede.

Esta técnica tem inúmeras vantagens, nomeadamente: 1) aumento da eficiência do encaminhamento, pois os pacotes dirigem-se para a instância do serviço mais próxima em termos das métricas de encaminhamento; 2) tolerância a falhas, pois se uma instância do serviço falha, desde que a instância do anúncio do prefixo IP deixe de existir, os novos clientes não se apercebem da falha; e 3) a técnica é muito útil para distribuir a carga.

Por estas razões, mesmo para aplicações baseadas no protocolo HTTP, o anycasting é usado em numerosas CDNs pois as suas vantagens são superiores aos seus potenciais defeitos. Com efeito, sendo o HTTP um protocolo predominantemente sem estado, tendo em consideração que muitas aplicações Web são implementadas usando inúmeras transações HTTP, geralmente de pequena duração, e finalmente usando técnicas de GETs HTTP parciais para a obtenção de objectos volumosos mas estáticos, é possível usar IP Anycasting com TCP de forma bastante eficaz. Esta prática está a generalizar-se entre os fornecedores de serviços de CDN para efeitos da aceleração a conteúdos estáticos e para fornecimento de serviços de segurança. A mesma é também usada pelos servidores *caching only* públicos (*e.g.*, Open DNS, Google DNS, ...).

18.4 Resumo e referências

Resumo

Para aumentar a escala de um sistema, é frequente o mesmo ser particionado em sub-componentes isoladas e introduzir alguma forma de hierarquia entre as mesmas. Este princípio, aplicado à Internet global, levou à sua subdivisão em sub-redes independentes, cada uma das quais é chamada um sistema autónomo.

Um **Sistema Autónomo (Autonomous System)** é uma rede ou conjunto de redes sob a mesma administração e cujo encaminhamento interno é opaco para o exterior. Todos os prefixes IP existentes na Internet pública pertencem necessariamente a um AS, ou estão incluídos num que pertence. Quando uma rede está ligada a diferentes ASs, constitui um AS específico e usa prefixes IP próprios.

De acordo com esta divisão da Internet em sub-redes, classificam-se os protocolos de encaminhamento em **protocolos IGP (Internal Gateway Protocols)**, os protocolos usados dentro dos ASs para encaminhamento intra-AS, e **protocolos EGP**

(*External Gateway Protocols*) para encaminhamento inter-AS. O encaminhamento inter-AS é assegurado pelo protocolo **BGP** (*Border Gateway Protocol*) que é geralmente executado pelos *routers* de fronteira ou *border routers*, os quais são responsáveis pela interligação dos ASs.

Um anúncio ou rota BGP é constituída por uma lista de prefixos e vários atributos comuns aos mesmos. Entre estes figura um caminho expresso como uma lista dos ASs, chamado *AS path*, que o anunciante usa para chegar ao AS onde residem os prefixos. Uma rota contém também um atributo com o endereço do *border router* de recepção do tráfego que é dirigido aos seus prefixos. Os *AS paths* permitem saber exactamente por que ASs passarão os pacotes, o que facilita a tomada de decisão em função do caminho proposto e permite suprimir a introdução de ciclos de encaminhamento. Dado os anúncios conterem caminhos, o BGP é caracterizado como um protocolo com base em **prefixos e vectores de caminhos** (*prefix-based path-vector protocol*).

Na Internet global o grafo de ligação entre os diferentes ASs é conhecido através dos prefixos IP e dos AS *paths* escolhidos e anunciado pelos ASs uns aos outros. A partir de *routers* da DFZ (*default free zone*) podem observar-se muitos desses AS *paths*, o que permite analisar as relações existentes entre os diferentes ASs e verificar que esses caminhos estão subordinados a relações comerciais ou de colaboração.

O grafo dos ASs é de grande dimensão pois tem cerca de 55.000 nós e muitos mais milhares de arcos. O número de ligações dos ASs é muito heterogénea e com uma distribuição muito desigual. Esta distribuição permite concluir que o conjunto dos ASs *Tier 1* é responsável directa ou indirectamente pela conectividade entre praticamente todos os ASs existentes.

O BGP é um protocolo complexo que assegura a conectividade da Internet global e que tem como requisitos ser capaz de lidar com centenas de milhar de prefixos IP e várias dezenas de milhar de redes interligadas, cada uma das quais com políticas específicas de encaminhamento. Apesar desta escala, pretende-se que convirja rapidamente.

O protocolo é executado por *routers* especiais, ditos **BGP speakers**. Estes *routers* coincidem geralmente com *border routers*, os *routers* que asseguram a interligação dos ASs. O diálogo entre os BGP *speakers* é assegurado através de **sessões BGP** (*BGP sessions*) que são ponto-a-ponto e utilizam conexões TCP. Os anúncios são incrementais e só comunicam actualizações (novas rotas ou supressão de rotas).

Como os caminhos em BGP são definidos como listas de ASs, é possível estabelecer regras para suporte de opções de encaminhamento com base em políticas e relações contratuais. O protocolo possui diversos mecanismos que permitem a implementação dessas políticas, nomeadamente: controlo da importação de rotas, manipulação de atributos das rotas e regras sofisticadas para comparação de rotas, controlo da exportação de rotas, etc. Adicionalmente o protocolo tem mecanismos para permitir aos diferentes *speakers* do mesmo AS conecerem as rotas usadas pelos outros (*internal BGP* ou *iBGP*) e interage com outros protocolos IGP do AS.

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

Principais conceitos

Sistema Autónomo - AS (*Autonomous System*) Um AS é uma rede ou conjunto de redes sob a mesma administração e cujo encaminhamento interno é opaco para o exterior. A Internet está particionada em alguns milhares de ASs. Cada AS corresponde grosso modo a um operador ou a uma organização.

AS border router Os *routers* responsáveis pela ligação dos ASs uns aos outros designam-se por *border routers*.

Protocolos IGP (*Internal Gateway Protocols*) Protocolos de encaminhamento usados internamente a um AS, ou protocolos intra-AS.

Protocolos EGP (*External Gateway Protocols*) Protocolos de encaminhamento usados externamente aos ASs, ou protocolos entre ASs ou inter-AS.

Protocolo BGP (*Border Gateway Protocol*) É o único protocolo especialmente concebido para encaminhamento inter-AS.

BGP speaker Os *routers* que executam o protocolo BGP designam-se BGP *speakers* e geralmente, mas não obrigatoriamente, coincidem com os *border routers*.

Caminho de ASs (*AS path*) Uma sequência de números de ASs que denota um caminho sem ciclos entre ASs na Internet global.

Rota BGP (*BGP route*) Um anúncio ou rota BGP é constituída por uma lista de prefixos IP e vários atributos entre os quais um AS *path*, que o anunciantre usa para chegar ao AS onde residem os prefixos. Uma rota contém também um atributo com o endereço do *border router* de recepção do tráfego que é dirigido aos seus prefixos.

Default Free Router Um BGP *speaker* ou outro *router* que não possui rotas por omissão (*default*) e conhece todos os prefixos IP reconhecidos na Internet global, diz-se um *Default Free Router*.

Sessão BGP (*BGP session*). Um canal virtual estabelecido entre dois BGP *speakers* chama-se uma BGP *session*. Estas sessões são ponto-a-ponto, e são mantidas sobre uma conexão TCP pelo que não exigem a adjacência por um canal físico entre os dois *speakers*. Os anúncios BGP que circulam nas sessões são incrementais e só comunicam actualizações (novas rotas ou supressão de rotas).

Protocolo iBGP (*Internal BGP*) Quando um AS tem mais do que um BGP *speaker* estes têm de conhecer as rotas externas seleccionadas uns pelos outros. A versão do protocolo executada entre os BGP *speakers* do mesmo AS chama-se iBGP.

Protocolo eBGP (*External BGP*) É a versão do BGP executada entre *speakers* de diferentes ASs.

Filtros de rotas BGP Para implementar as políticas de encaminhamento de um AS este usa filtros sobre as rotas importadas de outros ASs (*BGP import filters*) e filtros sobre as rotas exportadas para os outros ASs (*BGP export filters*). Os filtros mais comuns são só aceitar rotas de clientes para os prefixos próprios de cada cliente, e só exportar rotas para outros ASs sobre prefixos para os quais se aceita encaminhar tráfego.

Seleção de rotas Os *routers* BGP têm de selecionar a rota que vão usar para atingir cada prefixo. Para esse efeito compararam diversos atributos de todas as rotas para o prefixo que importam dos seus vizinhos. Uma das regras mais simples consiste em selecionar a rota com o AS *path* mais curto.

Referências

O livro [Huitema, 1995] é uma referência clássica que apresenta uma introdução ao BGP e discute as suas bases algorítmicas. Como o BGP é bastante complexo, normalmente raramente é tratado de forma muito completa em livros genéricos sobre redes de computadores. É necessário recorrer a livros especializados como por exemplo [Doyle and Carroll, 2017] que é considerado um dos melhores, e que inclui também indicações concretas sobre como o parametrizar em comutadores de pacotes populares.

No site da Caida, ver abaixo, encontram-se muitas referências bibliográficas e dados sobre a estrutura interna da Internet global recolhida de várias fontes entre as quais de tabelas de rotas BGP importadas de vários *routers* da DFZ. O survey [Motamedi et al., 2015] apresenta uma panorâmica das técnicas usadas para fazer levantamentos da topologia da Internet em termos de ASs.

O problema da convergência do BGP é um tema actual de investigação. O artigo [Gill et al., 2013] permite uma primeira incursão no mesmo. Tem sido conduzida uma

investigação intensiva sobre os problemas de segurança do BGP. [Butler et al., 2010; Huston et al., 2011] apresentam dois excelentes *surveys* sobre o tema, incluindo boas introduções ao BGP. No *site* da Internet Society, ver abaixo, encontram-se também muitas indicações sobre este tema.

Apontadores para informação na Web

- <http://www.icann.org> – É o *site* oficial da ICANN com informação como são atribuídos os números de AS.
- <http://bgp.potaroo.net> – É o *site* que mantém um relatório semanal actualizado do funcionamento do BGP na Internet global. Contém imensas estatísticas sobre a dinâmica do protocolo.
- <http://www.cidr-report.org> – É um *site* que mantém informação actualizada sobre os prefixos IP visíveis na Internet global e sobre a sua dinâmica de agregação e desagregação.
- <http://as-rank.caida.org> – É o *site* da Caida que mantém informação actualizada sobre todos os ASs existentes na Internet.
- <http://as-rank.caida.org/?mode0=as-ranking&n=10&ranksort=2> – É o *site* da Caida que mantém informação actualizada sobre o *ranking* dos ASs.
- <http://www.internetsociety.org/deploy360/securing-bgp/> – Este *site*, da responsabilidade da Internet Society, contém muita informação, artigos e recomendações sobre o problema da segurança do BGP.
- <http://ietf.org/rfc.html> – É o repositório oficial dos RFCs, nomeadamente dos citados neste capítulo.
- As diversas RIRs (*Regional Internet Registries*) mantêm um serviço *whois* sobre os diferentes ASs da sua região, assim como sobre os prefixos por estes exportados. Por exemplo, os comandos abaixo ilustram como obter informações do RIPE:

```
whois -h whois.ripe.net 1930
whois -h whois.ripe.net 193.136.0.0/15
```

18.5 Questões para revisão e estudo

1. Verdade ou mentira? Justifique a sua resposta.
 - (a) Quando no protocolo OSPF se introduzem zonas, estas são sub-redes que para todos os efeitos são equivalentes a um sistema autónomo (AS).
 - (b) O *backbone* NSFNET foi e é hoje em dia o AS número 1.
 - (c) O protocolo BGP é um protocolo de encaminhamento interno.
 - (d) O protocolo BGP é um protocolo de encaminhamento externo.
 - (e) O protocolo BGP é do tipo *link-state*.
 - (f) O protocolo BGP é do tipo *distance vector*.
 - (g) Um AS *multi-homed* assegura trânsito pago aos seus fornecedores de conectividade para a Internet global.
 - (h) Um *border router* de um AS escolhe sempre o caminho mais curto para alcançar um destino externo.

- (i) Um AS exporta via BGP uma rota para o prefixo IP p . Sempre que se produz uma alteração de estado na rede interna desse AS, que altera o custo com que p é alcançado internamente, essa alteração tem de desencadear uma actualização da rota para p via o BGP.
 - (j) Quando um AS exporta uma rota, nem sempre acrescenta o seu número de AS ao AS *path* da mesma.
 - (k) Um BGP *speaker* quando exporta um prefixo, exporta todas as rotas que conhece para o mesmo.
 - (l) Um BGP *speaker* só exporta uma rota para um prefixo IP se assegurar encaminhamento para o mesmo.
 - (m) Se um AS fizer encaminhamento interno para um dado prefixo, então necessariamente exporta uma rota para o mesmo.
 - (n) O protocolo OSPF é usado para fazer chegar automaticamente aos BGP *speakers* os prefixos IP que o AS deve exportar através de rotas.
 - (o) Como o encaminhamento interno de um AS deve ser opaco para o exterior, os prefixos IP públicos do AS não devem ser divulgados em rotas exportadas pelo BGP.
2. Verdadeiro ou falso? Justifique a sua resposta.
- (a) Dentro de um *Stub AS* apenas circulam pacotes com origem ou destino aos prefixos IP que lhe pertencem.
 - (b) Dentro de um *Stub AS* circulam pacotes com destino aos prefixos IP que lhe pertencem.
 - (c) De dentro para fora de um *Stub AS* circulam pacotes com origem nos prefixos do AS.
 - (d) De fora para dentro de um *Stub AS* circulam pacotes com origem nos prefixos do AS.
 - (e) Dentro de um AS de trânsito circulam pacotes com origem ou destino nos prefixos IP que lhe pertencem.
 - (f) Dentro de um AS de trânsito apenas circulam pacotes com origem ou destino nos prefixos IP para os quais o AS anuncia rotas.
 - (g) Dentro de um AS de trânsito circulam pacotes com destino aos prefixos IP para os quais o AS anuncia rotas.
 - (h) Dentro de um AS de trânsito circulam pacotes com origem nos prefixos IP para os quais o AS anuncia rotas.
 - (i) Dentro de um AS de trânsito circulam pacotes com origem nos prefixos IP dos seus clientes.
3. A maioria dos protocolos de encaminhamento IGP usam UDP ou equivalente para passarem informação de encaminhamento entre comutadores vizinhos. No entanto, o BGP usa TCP. Apresente algumas das razões que justificam esta opção.
4. Descreva o processo de decisão usado pelo BGP para comparar rotas alternativas para o mesmo destino.
5. Descreva os mecanismos usados pelo BGP para implementar decisões de encaminhamento de carácter político, *i.e.*, independentes de quaisquer métricas de custo.
6. Este exercício e o seguinte estão propostos em [Peterson and Davies, 2012]. Seja a o número de ASs na Internet e d (de diâmetro) o tamanho máximo de um AS *path*.

- (a) Descreva um modelo de conectividade entre ASs tal que $d = O(\log a)$.
 (b) Descreva um modelo de conectividade entre ASs tal que $d = O(\sqrt{a})$.
7. Admitindo que na Internet existem n prefixos, dê o valor aproximado da dimensão da informação recebida de um vizinho por um BGP *speaker* para que o mesmo preencha completamente a sua FIB nos seguintes modelos de conectividade.
- (a) Modelo de conectividade entre ASs tal que $d = O(\log a)$.
 (b) Modelo de conectividade entre ASs tal que $d = O(\sqrt{a})$.
8. A Figura 18.20 apresenta um conjunto de ASs com as ligações indicadas. Cada AS_i é simultaneamente o identificador de um AS e o identificador do prefixo por este anunciado. Os anúncios das rotas para os prefixos devem seguir a seguinte notação: [prefixo, [AS path]]. O AS_2 é um operador de trânsito, enquanto que todos os outros são apenas clientes. As ligações oblíquas são de *peering* directo entre cada um dos ASs clientes que lhes estão ligados e apenas para os respetivos prefixos IP.

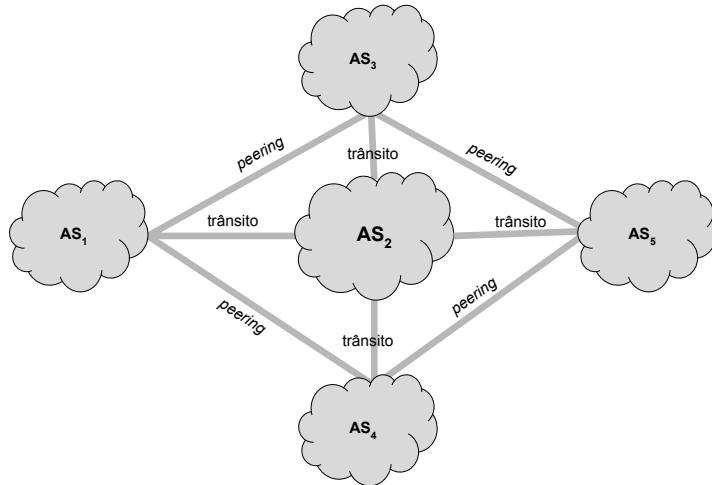


Figura 18.20: Rede do exercício 8

- (a) Quantas rotas pode o AS_1 explorar para chegar ao prefixo do AS_5 e qual a que prefere?
 (b) Quantas rotas pode o AS_1 explorar para chegar ao prefixo do AS_3 e qual a que prefere?
 (c) Caso o canal que liga o AS_2 ao AS_5 tivesse uma avaria e ficasse indisponível, como anunciaría o AS_2 tal evento ao AS_1 ? Quantas rotas poderia então o AS_1 explorar para chegar ao prefixos do AS_5 ?
 (d) Admita agora que o AS_3 pretende dar trânsito ao AS_1 mas só para o AS_5 . Como é que isso poderia ser implementado e que critérios deveria usar o AS_1 para escolher entre as duas possíveis rotas que teria para chegar ao AS_5 quando a rede estiver com todos os canais disponíveis?

9. A Figura 18.21 apresenta um conjunto de ASs interligados com as ligações indicadas. Cada AS_i é simultaneamente o identificador de um AS e o identificador do prefixo por este anunciado. Os anúncios das rotas para os prefixos devem seguir a seguinte notação: [prefixo, [AS path]]. Os sistemas autónomos AS_A a AS_D representam redes de trânsito que fazem *peering* transitivo entre si, i.e., , conhecem todas as redes clientes uns dos outros. Os AS_x a AS_k são clientes. Cada um dos clientes anuncia o seu prefixos para os seus fornecedores de trânsito e recebem destes o anúncio de todos os prefixos alcançáveis na rede. Os ASs AS_x e AS_y têm um canal directo entre si e estabeleceram um contrato entre os dois para que apenas o tráfego de e para cada um dos dois passe diretamente nesse canal (acordo de *peering* directo). Adicionalmente, o AS_x paga ao AS_y para que este lhe dê conectividade para o resto da rede apenas no caso em que o canal de AS_x para AS_B for abaixo.

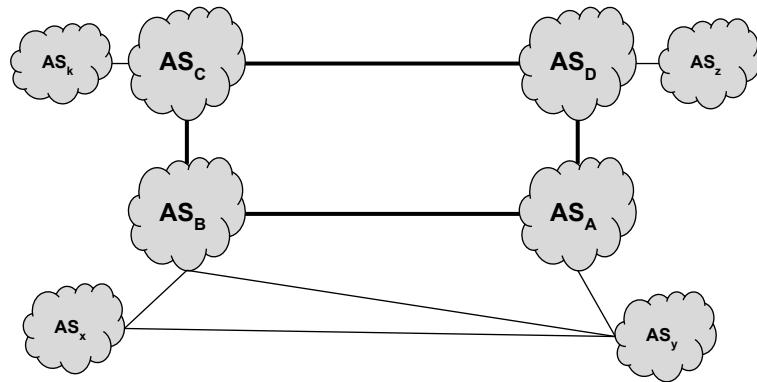


Figura 18.21: Rede do exercício 9

Indique os anúncios BGP enviados:

- (a) de AS_x para AS_B
 - (b) de AS_x para AS_y
 - (c) de AS_y para AS_A
 - (d) de AS_y para AS_B
 - (e) de AS_A para AS_y
 - (f) de AS_B para AS_y
 - (g) de AS_B para AS_x
 - (h) de AS_y para AS_x
10. A Figura 18.22 apresenta um conjunto de ASs interligados com as ligações indicadas. Cada AS_i é simultaneamente o identificador de um AS e o identificador do prefixo por este anunciado. Os anúncios das rotas para os prefixos devem seguir a seguinte notação: [prefixo, [AS path]]. Por exemplo, AS_3 anuncia ao AS_5 o caminho que usa para chegar a AS_8 através do seguinte anúncio: $[AS_8, [AS_3, AS_2, AS_8]]$. Não existem restrições de uso de caminhos devido a políticas comerciais e todos os ASs escolhem o caminho que atravessa um menor número de ASs. Caso um AS conheça duas rotas distintas para o mesmo prefixo e com o mesmo tamanho, prefere o caminho cujo primeiro AS tenha o identificador mais baixo (preferiria AS_8 a AS_9 por exemplo). Os anúncios BGP possíveis são: **update** e **withdraw**.

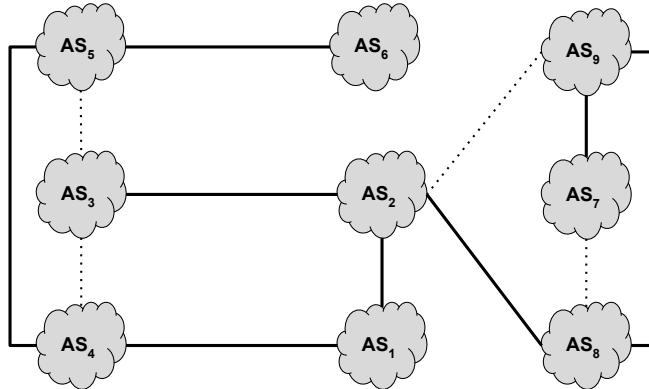


Figura 18.22: Rede do exercício 10

- (a) Indique para cada um dos ASs todas as rotas para o prefixo AS_9 que cada um deles conhece.
- (b) O canal que liga AS_2 a AS_9 foi aberto. Que mensagens BGP enviou AS_2 aos seus vizinhos.
- (c) O canal que liga AS_2 a AS_9 foi aberto no momento t e AS_2 detectou esse evento imediatamente. Durante quanto tempo AS_2 não conseguiu enviar pacotes destinados a AS_9 sabendo que o valor do timer MRAI é de 30 segundos? Justifique a sua resposta.
- (d) Considere agora que os diferentes ASs têm uma relação comercial entre eles de tal forma que os canais a negrito entre dois ASs constituem uma relação cliente / fornecedor em que o fornecedor é o AS com identificador inferior. Por exemplo, o canal entre o AS_1 e o AS_2 é um canal pelo qual o AS_2 paga a AS_1 para ter conectividade para AS_4 , AS_5 e AS_6 (note que AS_3 e AS_8 são clientes de AS_2). Os canais a tracejado são canais de *peering* que, por hipótese, só podem ser usados para comunicar directamente entre os dois ASs a eles ligados. Por exemplo, o canal entre AS_2 e AS_9 só pode ser usado para AS_2 comunicar com AS_9 e vice-versa e o canal entre AS_7 e AS_8 também só pode ser usado para AS_7 e AS_8 comunicarem directamente. Neste quadro indique todos os caminhos anunciados por AS_2 aos seus vizinhos.
11. Este exercício está proposto em [Marsic, 2013]. A Figura 18.21 apresenta um conjunto de ASs interligados com as ligações indicadas. Cada AS_i é simultaneamente o identificador de um AS e o identificador do prefixo por este anunciado. Os ASs AS_A e AS_B são *peers* e ambos compram trânsito a AS_C , o qual assegura a ligação ao resto da Internet. Todos os ASs usam *hot potato routing*. Os anúncios realizados são os compatíveis com as relações comerciais existentes entre os ASs. Os ASs que são *peers* não fornecem *backup* um ao outro. Todos os AS paths entre ASs vizinhos diretos não contém ocorrências repetidas de ASs. Os diferentes ASs usam todos o mesmo IGP e todos os canais internos a um AS têm custo idêntico nesse IGP.
- (a) De quantos e quais AS paths dispõem o computador y para chegar ao computador x ?
- (b) Porque routers passam os pacotes de y para x e vice-versa?

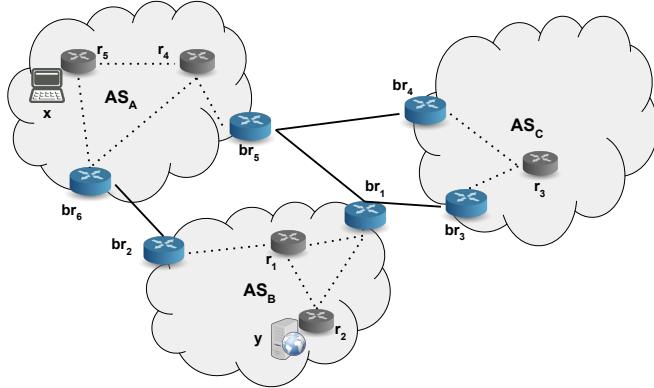


Figura 18.23: Rede do exercício 11

- (c) Em que situação esses dois caminhos poderiam coincidir?
- (d) Em que circunstâncias os pacotes de y para x atravessam o AS AS_C ?
- (e) Como se assegura que o tráfego de AS_A para o resto da Internet não vá via o AS_B caso a ligação de AS_A a AS_C ficar inoperacional?
- (f) Admita que se pretende que de facto AS_A e AS_B dêem backup mútuo na ligação à Internet um do outro. Como se pode atingir esse objectivo?
12. Na maioria dos *routers* é possível introduzir rotas estáticas manualmente e fazer com que os anúncios dos protocolos IGP os incluam. Este tipo de mecanismos é frequentemente usado para anunciar uma rota por omissão (*default route*). As perguntas a seguir devem ser respondidas no contexto da rede de uma instituição com um *backbone* IP próprio, com vários *routers*, em várias localidades, e uma ou mais ligações à Internet. O IGP usado é o OSPF. O EGP é o BGP mas a comunicação entre os dois só se faz usando rotas estáticas, *i.e.*, colocadas manualmente pelos gestores.
- (a) Suponha que a rede só tem uma ligação à Internet. O *router* que a assegura tem uma rota estática, associada à rota por omissão (*default*), que aponta para a interface de ligação ao exterior do *border router*. O OSPF permite dar manualmente um custo a esta rota. Essa rota é normalmente anunciada a todos os outros *routers* da rede interna. Quando é que os outros *routers* usam a rota por omissão assim anunciada?
- (b) Suponha que a interface de ligação à Internet referida na alínea anterior vai abaixar. Que alterações têm lugar nas FIBs da rede interna, como e com que velocidade?
- (c) Suponha agora que há várias ligações à Internet. Estando todas activas, como se passa a distribuição de carga entre elas?
- (d) Se no contexto da alínea anterior uma das interfaces de ligação externa for abaixar, que alterações têm lugar nas FIBs dos *routers* da rede interna e como?
13. A rede u está ligada à rede t (operador de trânsito de u) para ter conectividade Internet. A rede u tem o prefixo $100.100.100.0/24$ e a rede t tem o prefixo $100.100.0.0/16$. Como a rede u só tem um fornecedor externo, os seus gestores parametrizaram-na de tal forma que o *router* r_1 que a liga à rede t injecta uma

rota estática por omissão ($0.0.0.0/0$) no seu IGP. A rede t apenas anuncia por BGP na Internet global o seu prefixo ($100.100.0.0/16$). Com efeito, a rede u não tem outras ligações ao mundo e é vista como uma sub-rede da rede t sem AS próprio.

Mais tarde, devido a um projecto de colaboração, a rede u ligou-se também à rede isp , com o prefixo $200.200.200.0/24$, através do router r_2 . O objectivo é permitir que os utilizadores da rede u acedam via esse router à rede $200.200.200.0/24$, e vice versa, sem necessidade de passar pela Internet global. Os outros ASs não sabem desta “ligação directa”.

(a) Para fazer esta ligação a rede u e a rede isp têm de usar o mesmo protocolo de encaminhamento IGP nas suas redes internas (RIP, OSPF, ...) e têm de estabelecer uma ligação BGP entre si? Justifique.

(b) Que rota deve anunciar estaticamente o router r_2 para dentro da rede u e que rota deve anunciar estaticamente o router dessa ligação na rede isp ? Justifique.

(c) Mais tarde a rede u resolveu que a sua ligação à rede isp passaria a dar-lhe também trânsito para a Internet, mas ao mesmo tempo pretendia continuar com a ligação à Internet via a rede t . Para esse efeito arranjou um número de AS, o AS- u , e parametrizou os routers r_1 e r_2 para passarem a usar BGP.

Os routers r_1 e r_2 passaram a anunciar o AS Path [$100.100.100.0/24$, [AS- u]] por BGP para os outros operadores. A rede u arranjou também forma de os routers internos dividirem a carga do tráfego de saída entre si (dentro da rede u ambos os routers anunciam uma *default route*). A rede t continuou a anunciar apenas o seu prefixo via a rota [$100.100.0.0/16$, [AS- t]] e a rede isp passou a anunciar para a Internet as rotas [$100.100.100.0/24$, [AS- isp , AS- u]] e [$200.200.0.0/16$, [AS- isp]]. Por que rede ou redes passou a entrar o tráfego da Internet dirigido à rede u ? Justifique.

(d) Como poderia a rede u conseguir que o tráfego vindo da Internet para o seu prefixo passasse a usar os dois fornecedores? Justifique.

(e) Qual a forma mais simples que tem a rede u de impedir que o tráfego vindo da rede t e dirigido a clientes da rede isp atravessem a sua rede e vice-versa? Justifique.

(f) Que meios poderiam usar os gestores da rede u para afinar a distribuição de carga por esses dois fornecedores quer em entrada, quer em saída?

14. Inspire-se do protocolo BGP para recomendar uma nova versão do protocolo RIP que permita ao mesmo ser usado em redes de muito maior escala e anule o seu principal defeito ligado a um tempo de convergência demasiado elevado.

15. Execute os comandos:

```
whois -h whois.ripe.net AS1930
whois -h whois.ripe.net 193.136.0.0/15
```

e interprete os seus resultados, nomeadamente sob o ponto de vista das relações e políticas de importação e exportação de rotas dos ASs referidos.

16. Através do estudo da bibliografia sugerida, apresente um resumo das principais fragilidades do protocolo BGP do ponto de vista da segurança.

17. Através do estudo da bibliografia sugerida, apresente um resumo das principais propostas para colmatar as falhas de segurança do protocolo BGP.

Bibliografia

- Abboud, O., Pussep, K., Kovacevic, A., Mohr, K., Kaune, S., and Steinmetz, R. (2011). Enabling resilient p2p video streaming: survey and analysis. *Multimedia Systems*, 17(3):177–197.
- Abed, G. A., Ismail, M., and Jumari, K. (2011). A survey on performance of congestion control mechanisms for standard tcp versions. *Australian Journal of Basic and Applied Sciences*, 5(12).
- Afanasyev, A., Tilley, N., Reiher, P., and Kleinrock, L. (2010). Host-to-host congestion control for tcp. *Communications Surveys Tutorials, IEEE*, 12(3):304–342.
- Akamai Technologies, I. (2016). The akamai state of the internet report - first quarter of 2016. Available on-line from the company site.
- Allen, K. S. (2012). *What Every Web Developer Should Know About HTTP*. OdeToCode LLC, Kindle Edition, 1st edition.
- Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. (2001). Resilient overlay networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 131–145, New York, NY, USA. ACM.
- Balakrishnan, H., Seshan, S., Amir, E., and Katz, R. H. (1995). Improving tcp/ip performance over wireless networks. In *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking*, MobiCom '95, pages 2–11, New York, NY, USA. ACM.
- Beard, C. and Stallings, W. (2015). *Wireless Communications and Systems*. Pearson Education.
- Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.
- Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. (1994). The world-wide web. *Commun. ACM*, 37(8):76–82.
- Berners-Lee, T., Hall, W., Hendler, J., Shadbolt, N., and Weitzner, D. J. (2006). Creating a science of the web. *Science*, 313(5788):769–771.
- Birrell, A. D., Levin, R., Schroeder, M. D., and Needham, R. M. (1982). Grapevine: An exercise in distributed computing. *Commun. ACM*, 25(4):260–274.

- Birrell, A. D. and Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59.
- Board, I. A. (2000). IAB Technical Comment on the Unique DNS Root. RFC 2826 (Informational).
- Boudec, J.-Y. (2014). Rate adaptation, congestion control and fairness: A tutorial. Available on-line from Ecole Polytechnique Fédérale de Lausanne (EPFL).
- Brakmo, L. S., O’Malley, S. W., and Peterson, L. L. (1994). Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM ’94, pages 24–35, New York, NY, USA. ACM.
- Budzisz, L., Garcia, J., Brunstrom, A., and Ferrús, R. (2012). A taxonomy and survey of sctp research. *ACM Comput. Surv.*, 44(4):18:1–18:36.
- Butler, K., Farley, T. R., McDaniel, P., and Rexford, J. (2010). A survey of bgp security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122.
- Calvert, K. and Donahoo, M. (2011). *TCP/IP Sockets in Java: Practical Guide for Programmers*. The Practical Guides. Elsevier Science.
- Cerf, V. and Kahn, R. (1974). A protocol for packet network intercommunication. *Communications, IEEE Transactions on*, 22(5):637–648.
- Chadwick, D. (1994). *Understanding X.500: The Directory*. Chapman & Hall, Ltd., London, UK, UK.
- Charles, S. K. (2000). Secure border gateway protocol (s-bgp)—real world performance and deployment issues. *Proceedings of the NDSS2000*.
- Chen, F., Sitaraman, R. K., and Torres, M. (2015). End-user mapping: Next generation request routing for content delivery. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM ’15, pages 167–181, New York, NY, USA. ACM.
- Chen, X., Zhai, H., Wang, J., and Fang, Y. (2005). A survey on improving tcp performance over wireless networks. In *In Resource Management in Wireless Networking, Cardei M, Cardei I, Du D-Z (eds*, pages 657–695. Kluwer Academic Publishers.
- Chiu, D. M. and Jain, R. (1989). Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14.
- Chu, Y.-h., Rao, S. G., and Zhang, H. (2000). A case for end system multicast (keynote address). In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS ’00, pages 1–12, New York, NY, USA. ACM.
- Clark, D. (1988). The design philosophy of the darpa internet protocols. In *SIGCOMM ’88: Symposium proceedings on Communications architectures and protocols*, pages 106–114, New York, NY, USA. ACM.
- Clark, D. D. and Tennenhouse, D. L. (1990). Architectural considerations for a new generation of protocols. In *Proceedings of the ACM Symposium on Communications Architectures & Protocols*, SIGCOMM ’90, pages 200–208, New York, NY, USA. ACM.

- Cohen, B. (2003). Incentives build robustness in BitTorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*.
- Comer, D. E. and Stevens, D. L. (1997). *Internet With TCP/IP – Volume III – Client-Server Programming and Applications*. Prentice-Hall.
- Couch, L. (2000). *Digital and Analog Communication Systems*. Prentice-Hall.
- Czyz, J., Allman, M., Zhang, J., Iekel-Johnson, S., Osterweil, E., and Bailey, M. (2014). Measuring ipv6 adoption. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 87–98, New York, NY, USA. ACM.
- Dai, J., Liu, F., and Li, B. (2010). The disparity between p2p overlays and isp underlays: issues, existing solutions, and challenges. *Netwrk. Mag. of Global Internetwkg.*, 24:36–41.
- David, E. and Jon, K. (2010). *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA.
- Day, J. (2008). *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall.
- Dijkstra, E. (1959). A note on two problems in connection with graphs. *Numerische Mathematik*, 1(269-270):6.
- Dobrescu, M., Egi, N., Argyraki, K., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Manesh, M., and Ratnasamy, S. (2009). Routebricks: Exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 15–28, New York, NY, USA. ACM.
- Donnet, B., Gueye, B., and Kaafar, M. A. (2010). A survey on network coordinates systems, design, and security. *IEEE Communications Surveys Tutorials*, 12(4):488–503.
- Doyle, J. and Carroll, J. (2006). *Routing TCP/IP, Volume 1, 2nd edition*. Pearson Education and Cisco Press.
- Doyle, J. and Carroll, J. (2017). *Routing TCP/IP, Volume 2, 2nd edition*. Pearson Education and Cisco Press.
- Dreibholz, T., Rathgeb, E., Rungeler, I., Seggelmann, R., Tuxen, M., and Stewart, R. (2011). Stream control transmission protocol: Past, current, and future standardization activities. *Communications Magazine, IEEE*, 49(4):82–88.
- Feldmann, A. (2007). Internet clean-slate design: What and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64.
- Feldmeier, D. (1995). Fast software implementation of error detection codes. *Networking, IEEE/ACM Transactions on*, 3(6):640–651.
- Floyd, S. and Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413.
- Floyd, S. and Paxson, V. (2001). Difficulties in simulating the internet. *IEEE/ACM Trans. Netw.*, 9(4):392–403.
- Ford, D. R. and Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press, Princeton, NJ, USA.

- Forouzan, B. A. (2007). *Data Communications and Networking – Fourth Edition*. McGraw-Hill.
- Francois, P., Filsfils, C., Evans, J., and Bonaventure, O. (2005). Achieving sub-second igp convergence in large ip networks. *SIGCOMM Comput. Commun. Rev.*, 35(3):35–44.
- Gill, P., Schapira, M., and Goldberg, S. (2013). A survey of interdomain routing policies. *SIGCOMM Comput. Commun. Rev.*, 44(1):28–34.
- Gourley, D., Totty, B., Sayer, M., Aggarwal, A., and Reddy, S. (2002). *HTTP: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition.
- Graziani, R. (2013). *IPv6 Fundamentals: A Straightforward Approach to Understanding IPv6*. Cisco Press, Indianapolis, IN, USA.
- Grigorik, I. (2013). Making the web faster with http 2.0. *Queue*, 11(10):40:40–40:53.
- Gross, J. and Yellen, J. (2005). *Graph Theory and Its Applications, Second Edition*. Textbooks in Mathematics. Taylor & Francis.
- Grosvenor, M. P., Schwarzkopf, M., Gog, I., Watson, R. N. M., Moore, A. W., Hand, S., and Crowcroft, J. (2015). Queues don't matter when you can jump them! In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 1–14, Oakland, CA. USENIX Association.
- Ha, S., Rhee, I., and Xu, L. (2008). Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74.
- Hagen, S. (2014). *Praise for IPv6 Essentials, Third Edition - Integrating IPv6 into Your IPv4 Network*. "O'Reilly Media, Inc.".
- Hall, W. and Tiropanis, T. (2012). Web evolution and web science. *Comput. Netw.*, 56(18):3859–3865.
- Halperin, D., Hu, W., Sheth, A., and Wetherall, D. (2010). 802.11 with multiple antennas for dummies. *ACM SIGCOMM Computer Communication Review*, 40(1):19–25.
- Hamming, R. (1980). *Coding and Information Theory*. Prentice-Hall.
- Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. (2012). Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 253–264, New York, NY, USA. ACM.
- Harold, E. (2013). *Java Network Programming, 4th Edition*. O'Reilly Media.
- Held, G. (2003). *Ethernet Networks: Design, Implementation, Operation, and Management*. John Wiley & Sons.
- Hofmann, M. and Beaumont, L. R. (2005). *Content Networking: Architecture, Protocols, and Practice (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Howes, T. A., Smith, M. C., and Good, G. S. (2003). *Understanding and Deploying LDAP Directory Services, Second Edition*. Addison-Wesley Professional.
- Hsu, W. (2003). *Analog and Digital Communications*. McGraw-Hill.

- Huitema, C. (1995). *Routing in the Internet*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- Huston, G., Rossi, M., and Armitage, G. (2011). Securing bgp - a literature survey. *IEEE Communications Surveys Tutorials*, 13(2):199–222.
- Jacobson, V. (1988). Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA. ACM.
- Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley professional computing. Wiley.
- Jennings, C., Hardie, T., and Westerlund, M. (2013). Real-time communications for the web. *Communications Magazine, IEEE*, 51(4):20–26.
- Kamp, P.-H. (2015). Http/2.0: The ietf is phoning it in. *Commun. ACM*, 58(3):40–42.
- Kent, S., Lynn, C., and Seo, K. (2000). Secure border gateway protocol (s-bgp). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592.
- Kleinrock, L. (1972). *Communication Nets; Stochastic Message Flow and Delay*. Dover Publications, Incorporated.
- Kleinrock, L. (1979). Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Proceedings of the International Conference on Communications*, pages 1–10.
- Knight, S., Nguyen, H., Falkner, N., Bowden, R., and Roughan, M. (2011). The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765 –1775.
- Kohler, E., Handley, M., and Floyd, S. (2006). Designing dccp: Congestion control without reliability. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 27–38, New York, NY, USA. ACM.
- Kurose, J. and Ross, K. (2013). *Computer Networks – A Top-Down Approach – Sixth Edition*. Always learning. Pearson Education.
- Lampson, B. W. (1983). Hints for computer system design. *SIGOPS Oper. Syst. Rev.*, 17(5):33–48.
- Lampson, B. W. and Sproull, R. F. (1979). An open operating system for a single-user machine. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, pages 98–105, New York, NY, USA. ACM.
- Liu, C. (2002). *DNS and BIND Cookbook*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition.
- Luo, Z. and Suh, C. (2011). An improved shortest path bridging protocol for ethernet backbone network. In *The International Conference on Information Networking 2011 (ICOIN2011)*, pages 148–153.
- MacKay, D. (2005). Fountain codes. *Communications, IEEE Proceedings-*, 152(6):1062–1068.
- Marsic, I. (2013). *Computer Networks - Performance and Quality of Service*. Rutgers University, New Jersey, USA.

- Mascolo, S., Casetti, C., Gerla, M., Sanadidi, M. Y., and Wang, R. (2001). Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297. ACM.
- Metcalfe, R. M. and Boggs, D. R. (1976). Ethernet: distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404.
- Motamedi, R., Rejaie, R., and Willinger, W. (2015). A survey of techniques for internet topology discovery. *IEEE Communications Surveys Tutorials*, 17(2):1044–1065.
- Moussavi, M. (2012). *Data Communication and Networking: A Practical Approach*. Delmar.
- Müller, C. and Timmerer, C. (2011). A vlc media player plugin enabling dynamic adaptive streaming over http. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM ’11, pages 723–726, New York, NY, USA. ACM.
- Murray, David; Terry Koziniec, K. L. and Dixon, M. (2012). Large mtus and internet performance. In *Proceedings of the 13th IEEE Conference on High Performance Switching and Routing*, HPSR 2012. IEEE.
- Nygren, E., Sitaraman, R. K., and Sun, J. (2010). The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19.
- Paasch, C. and Bonaventure, O. (2014). Multipath tcp. *Commun. ACM*, 57(4):51–57.
- Paasch, C., Detal, G., Duchene, F., Raiciu, C., and Bonaventure, O. (2012). Exploring mobile/wifi handover with multipath tcp. In *Proceedings of the 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design*, CellNet ’12, pages 31–36, New York, NY, USA. ACM.
- Pathan, A.-M. K. and Buyya, R. (2007). A taxonomy and survey of content delivery networks. *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, page 4.
- Peng, T., Leckie, C., and Ramamohanarao, K. (2007). Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv.*, 39(1).
- Perkins, C., Hodson, O., and Hardman, V. (1998). A survey of packet loss recovery techniques for streaming audio. *Network, IEEE*, 12(5):40–48.
- Perlman, R. (1985). An algorithm for distributed computation of a spanning tree in an extended lan. *ACM SIGCOMM Computer Communication Review*, 15(4):44–53.
- Perlman, R. and Eastlake, D. (2011). Introduction to trill. *The Internet Protocol Journal*, 14(3).
- Peterson, L. and Davies, B. (2012). *Computer Networks – a Systems Approach – Fifth Edition*. Elsevier.
- Pierce, J. (1984). Telephony—a personal view. *Comm. Mag.*, 22(5):116–120.
- Pouzin, L. (1975). The cyclades network – present state and development trends. In *Proceedings of the Symposium on Computer Networks*, pages 8–13. IEEE Computer Society.

- Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley, M. (2011). Improving datacenter performance and robustness with multipath tcp. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 266–277, New York, NY, USA. ACM.
- Raiciu, C., Iyengar, J., and Bonaventure, O. (2013). Recent advances in reliable transport protocols. In H. Haddadi, O. Bonaventure (Eds.), *Recent Advances in Networking*, pages 59–106.
- Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and Handley, M. (2012). How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 29–29, Berkeley, CA, USA. USENIX Association.
- Rizzo, L. (1997). Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM Computer Communication Review*, 27(2):24–36.
- Rodrigues, R. and Druschel, P. (2010). Peer-to-peer systems. *Commun. ACM*, 53(10):72–82.
- Rorabaugh, C. (1996). *Error Coding Cookbook*. McGraw-Hill.
- Ruiz-Sanchez, M. A., Biersack, E. W., and Dabbous, W. (2001). Survey and taxonomy of ip address lookup algorithms. *IEEE Network*, 15(2):8–23.
- Saltzer, J. H., Reed, D. P., and Clark, D. D. (1984). End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2:277–288.
- Sermersheim, J. (2006). Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511 (Proposed Standard).
- Spurgeon, C. (2000). *Ethernet: the definitive guide*. "O'Reilly Media, Inc.".
- Stallings, W. (2011). *Wireless Communications & Networks*. Pearson Education.
- Stallings, W. (2013). *Data and Computer Communications – 10th Edition*. Pearson Education.
- Stevens, R., Fenner, B., and Rudoff, A. M. (2004). *Unix Network Programming – Volume 1*. Addison-Wesley.
- Stevens, W. and Wright, G. (1994). *TCP/IP Illustrated: The protocols*. Number v. 1 in Addison-Wesley professional computing series. Addison-Wesley Publishing Company.
- Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA. ACM.
- TAN, K., SONG, J., ZHANG, Q., and SRIDHARN, M. (2006). A compound tcp approach for high-speed and long distance networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, page 1–12.
- Tanenbaum, A. and Wetherall, D. (2011). *Computer Networks – Fifth Edition*. Pearson Education.

- Theotokis, S. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371.
- Vogels, W. (2009). Eventually consistent. *Commun. ACM*, 52(1):40–44.
- Wang, J. (1999). A survey of web caching schemes for the internet. *SIGCOMM Comput. Commun. Rev.*, 29(5):36–46.
- Wang, Z. and Crowcroft, J. (1992). Eliminating periodic packet losses in 4.3-tahoe bsd tcp congestion control algorithm. *ACM Computer Communication Review*, 22(2):9–16.
- Webber, J., Parastatidis, S., and Robinson, I. (2010). *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly Media, Inc., 1st edition.
- Wischik, D., Raiciu, C., Greenhalgh, A., and Handley, M. (2011). Design, implementation and evaluation of congestion control for multipath tcp. In *NSDI*, volume 11, pages 8–8.
- Xie, H., Yang, Y. R., Krishnamurthy, A., Liu, Y. G., and Silberschatz, A. (2008). P4p: provider portal for applications. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 351–362, New York, NY, USA. ACM.
- Zaragoza, R. (2002). *The Art of Error Correcting Coding*. Addison-Wesley.
- Zec, M., Rizzo, L., and Mikuc, M. (2012). Dxr: Towards a billion routing lookups per second in software. *SIGCOMM Comput. Commun. Rev.*, 42(5):29–36.
- Zhang, C., Dhungel, P., Wu, D., and Ross, K. W. (2011). Unraveling the bittorrent ecosystem. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1164–1177.
- Zimmermann, H. (1980). Osi reference model—the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432.