

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 computorizados de toda a espécie, televisões e rádios com computadores de controlo e comunicação embalados, 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.*

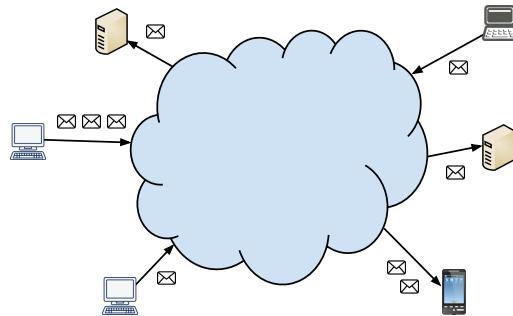


Figura 1.1: Uma rede de computadores

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 uma 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.

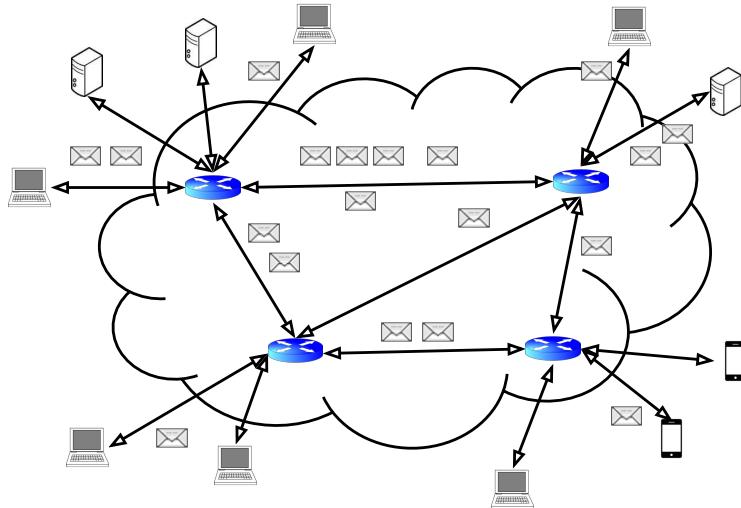


Figura 1.2: A rede de computadores contém canais e comutadores de pacotes

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.

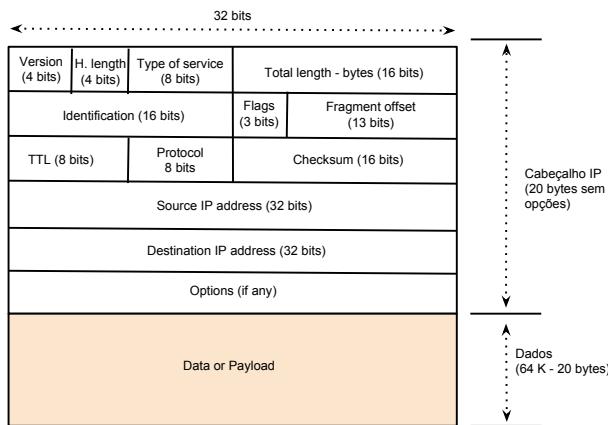


Figura 1.3: Pacote IP - campos do cabeçalho do pacote e parte de dados

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 facturam 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 ficasssem 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 acedem 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.

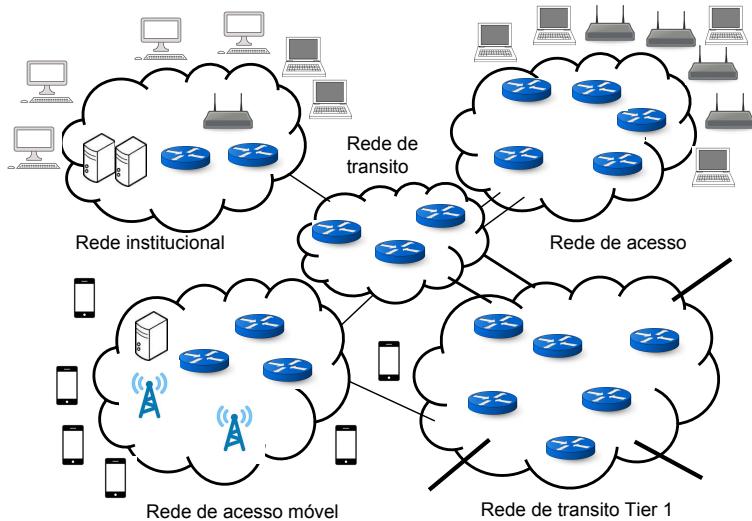


Figura 1.4: A Internet é uma rede de redes

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.

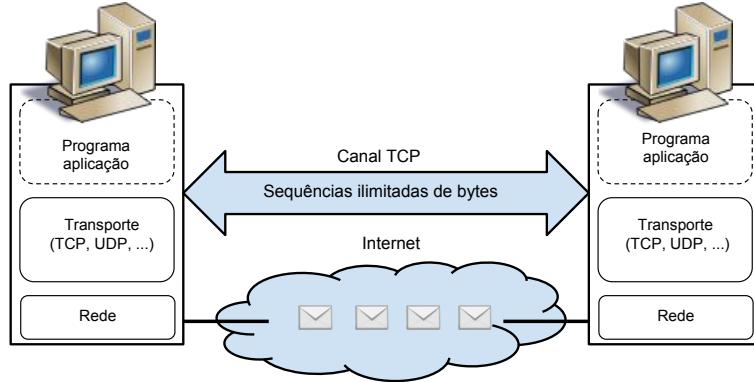


Figura 1.6: Serviços de transporte no sistema de operação - um canal TCP

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.

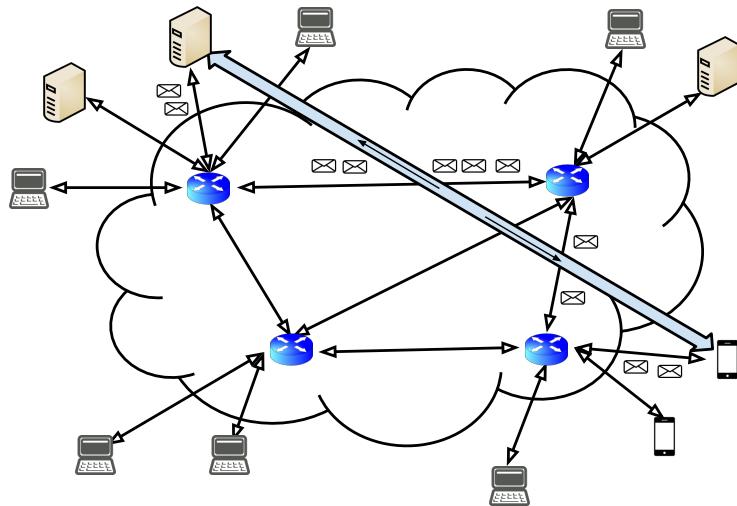


Figura 1.7: Um canal TCP é uma canal bidireccional virtual que liga directamente dois computadores

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

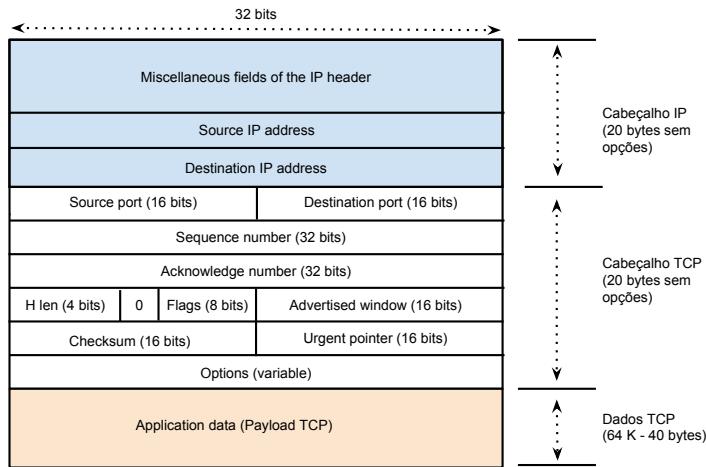


Figura 1.8: Segmento TCP e campos do seu cabeçalho

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

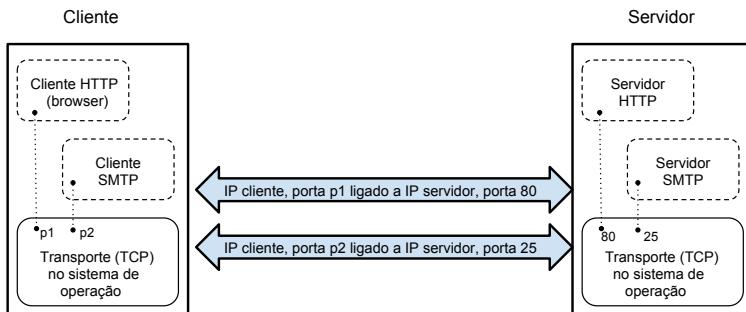


Figura 1.9: Diferentes canais TCP nos mesmos computadores distinguem-se por portas distintas

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 directamente 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.

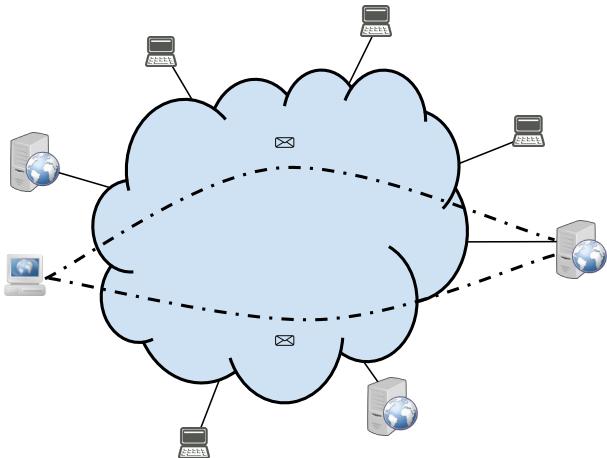


Figura 1.10: O padrão cliente / servidor

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

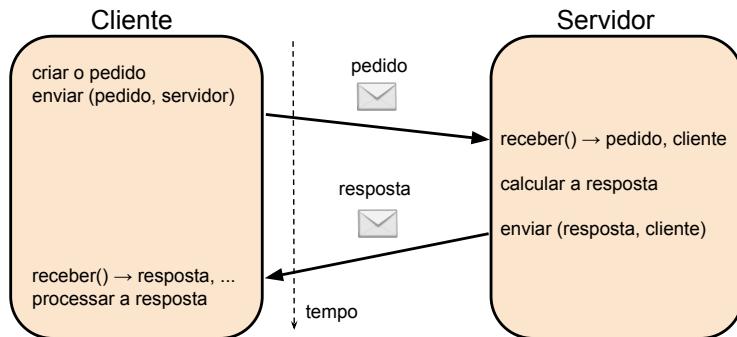


Figura 1.11: Diagrama temporal de execução do padrão cliente servidor

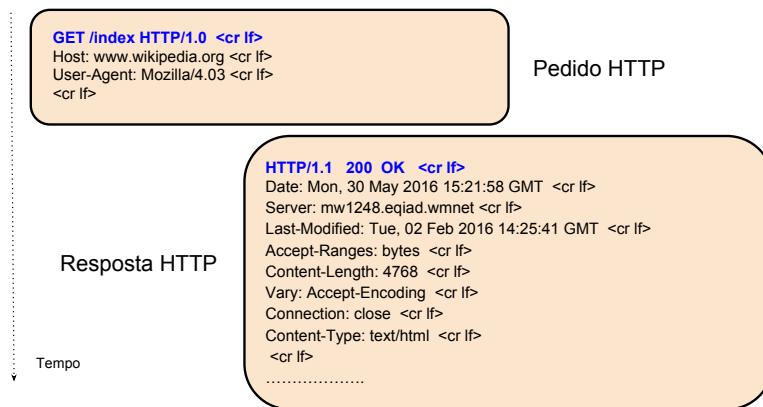


Figura 1.12: Exemplo de mensagens de pedido e resposta do protocolo HTTP

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 forneçam 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.

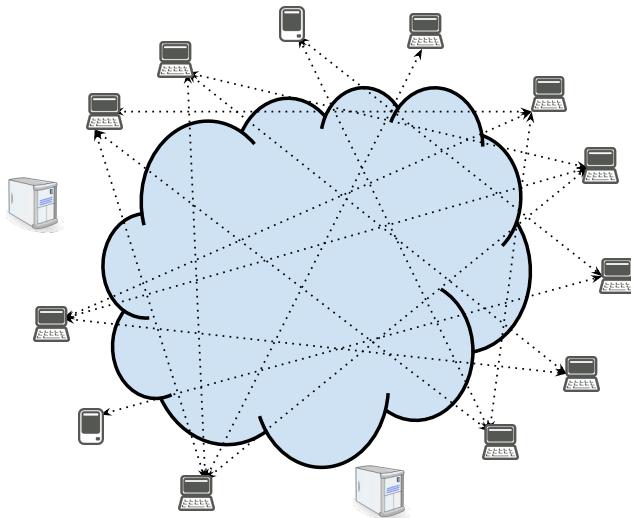


Figura 1.13: O padrão P2P em execução - parceiros a usarem o protocolo Bit-Torrent

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

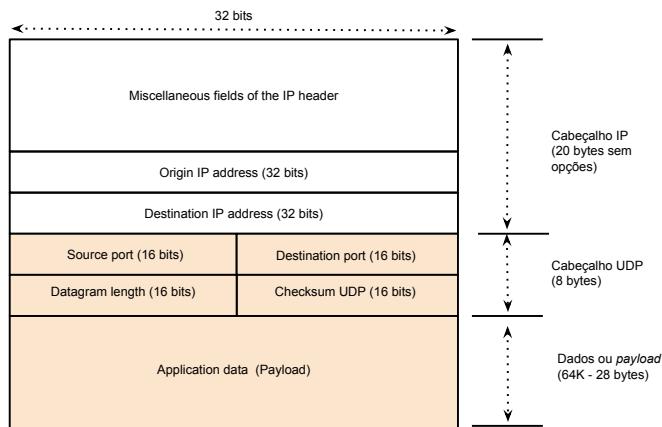
Ao analisarmos os datagramas UDP verificamos de novo que estes viajam na parte de dados de um pacote IP, cujos campos origem e destino indicam a origem e o destino do *datagrama*, que existe um cabeçalho de *datagrama* e que o *datagrama* também tem uma parte de dados. No cabeçalho encontramos de novo 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 mnemónica do que um número inteiro com muitos dígitos (o endereço IP). Adicionalmente, também permite uma maior flexibilidade. De facto, o nome mnemónico 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 mnemónicos 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 mnemónicos em endereços IP.

O serviço DNS é uma base de dados distribuída, contendo entradas para vários milhões de nomes mnemónicos 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.

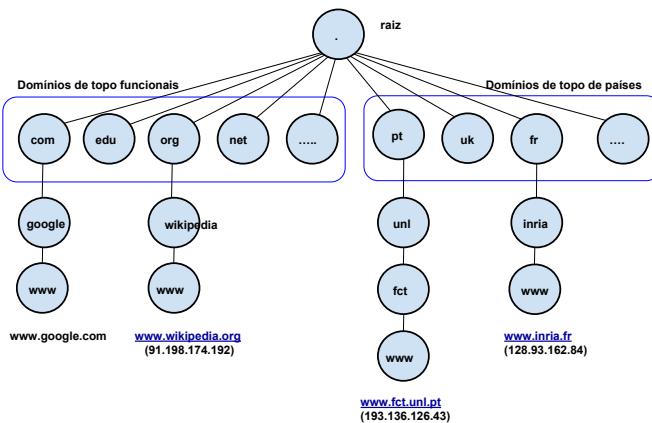


Figura 1.15: Uma parte da árvore de domínios do DNS e exemplo de alguns nomes

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 mnemónicos, apenas conhece endereços IP, tal como os protocolos de transporte.

Os nomes mnemónicos 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**, **fr**,

...) 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.*, `unl.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 registrado o domínio `wikipedia.org` e o endereço IP dos servidores DNS desse domínio.

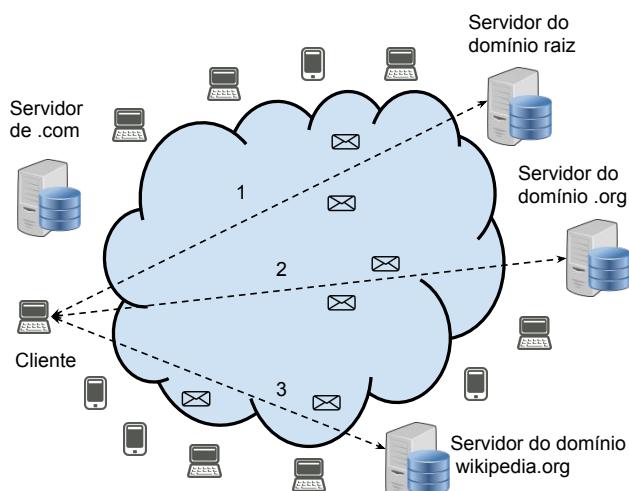


Figura 1.16: Pesquisa do endereço IP de `www.wikipedia.org`

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.

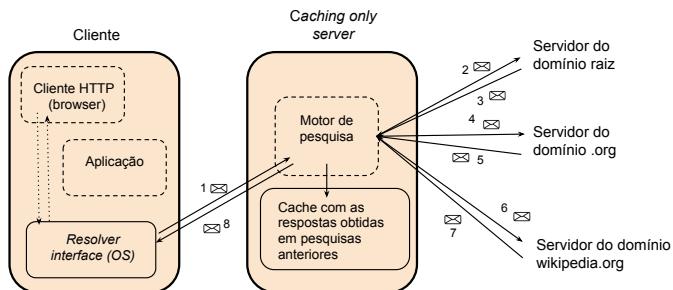


Figura 1.17: Pesquisa do endereço IP de `www.wikipedia.org` através de um *caching only server* - a numeração das mensagens segue a ordem de execução do protocolo

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

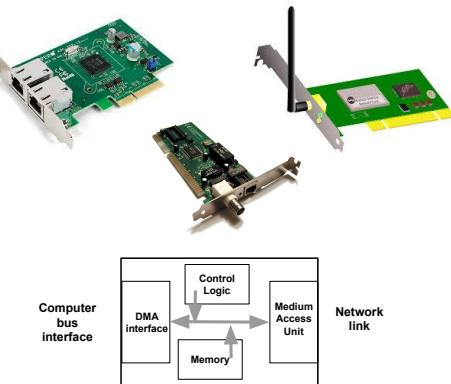


Figura 1.18: Interfaces de rede para computadores (separadas das placas mãe) e um diagrama esquemático das suas componentes

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 embebidas 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 accedem 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 noutras 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 noutra 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*.

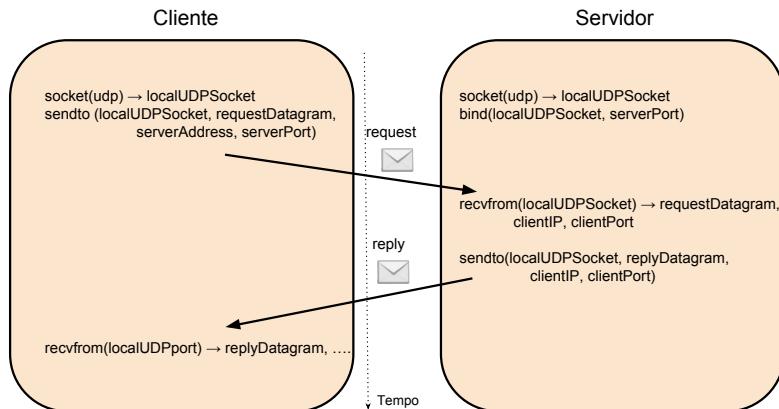


Figura 1.19: Comunicação com sockets UDP

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.

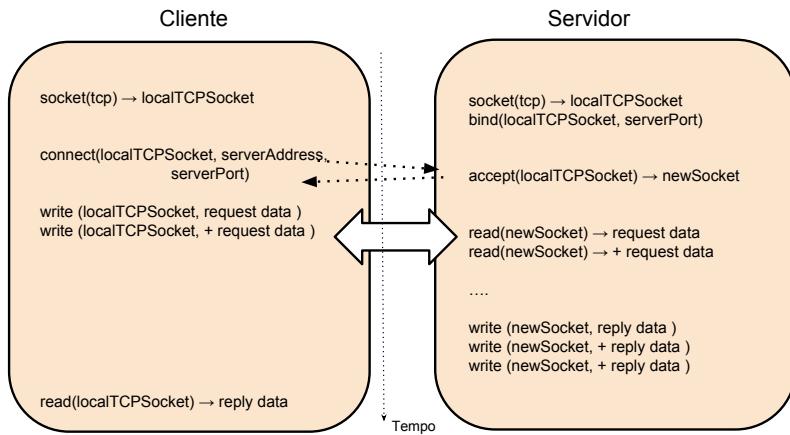


Figura 1.20: Comunicação com sockets TCP

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?
- As funções da biblioteca sistema *resolver* abrem sempre um canal TCP para um servidor DNS para obterem respostas aos pedidos das aplicações.
 - Quando um servidor DNS de um domínio não está disponível, deixa de ser possível conhecer os endereços IP dos computadores do domínio.
 - 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.
 - 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.
- 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
 - 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
 - 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
 - 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?
- 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”,

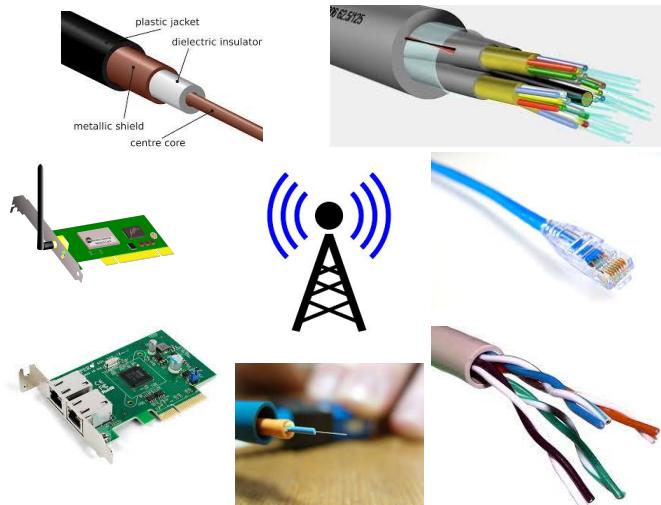


Figura 2.1: Exemplos de interfaces de comunicação e de meios de propagação do sinal

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.

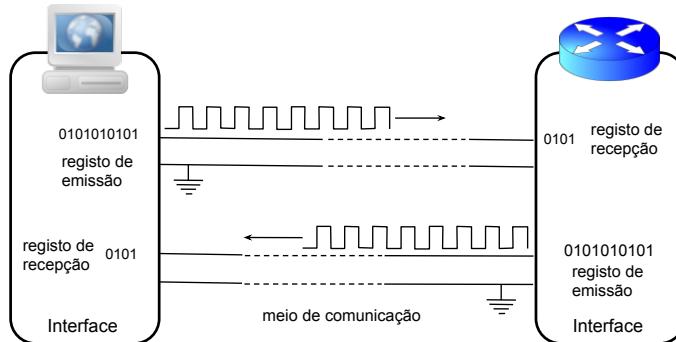


Figura 2.2: Um canal de comunicação

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 forma 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 registos 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 apóis 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.

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

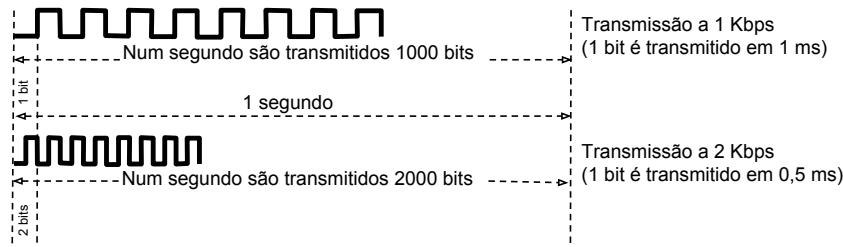


Figura 2.3: Aumentando o débito para o dobro (de 1 Kbps para 2 Kbps) o tempo de transmissão diminui para metade (1000 bits passam a ser transmitidos em 0,5 segundo ao invés de em 1 segundo)

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 ($1/10^9 = 10^{-9} s = 1 ns$), 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 $\approx 2 \times 10^8 m/s$ 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 $1/200000 = 5 \times 10^{-6} s = 5 \mu s$. 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.

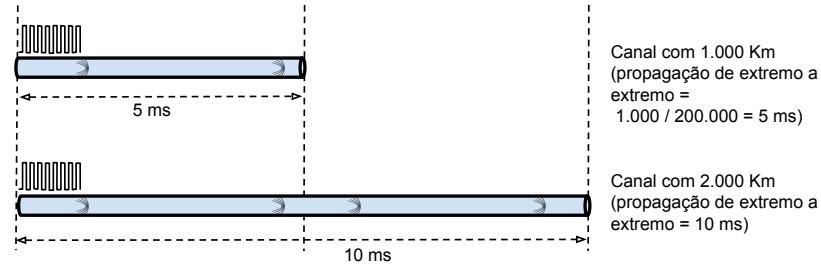


Figura 2.4: Num canal com 1.000 Km um bit chega à outra extremidade em 5 ms mas se o canal tiver 2.000 Km chega em 10 ms

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(ilô), 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 $50 \text{ Mbits} = 50.000.000/8$ 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 $1 \text{ KB} = 8 \times 1024 \approx 10 \times 10^3 = 10^4$ 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 micróns de espessura (1 micrón 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 instalam 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.

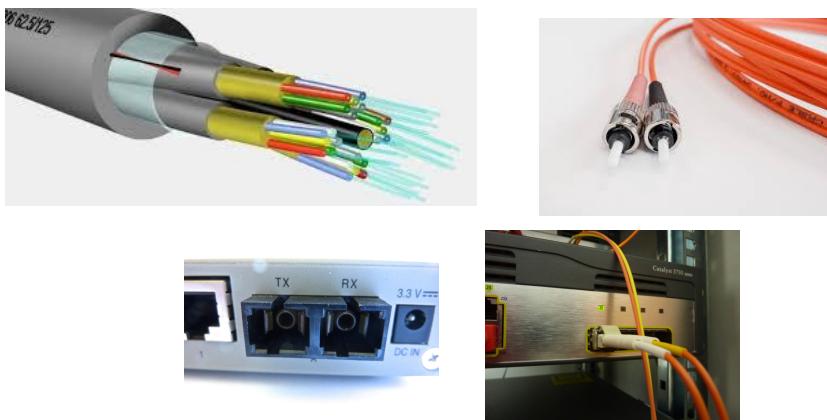


Figura 2.7: Cabo com várias fibras ópticas, “chicote” de fibra óptica e fichas de ligação a interfaces. Cada canal *full-duplex* usa um par de fibras (TX - Transmitir, RX - Receber)

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.

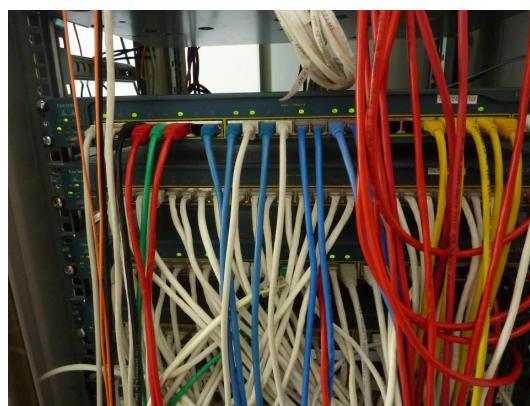


Figura 2.8: Interligação de muitos computadores através de vários comutadores pondo em evidência a complexidade da gestão dos cabos UTP nesse cenário

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 nó 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.

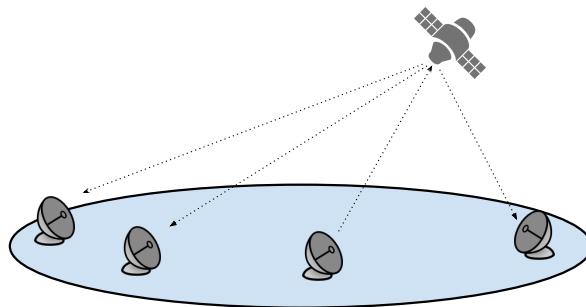


Figura 2.10: Canal de dados baseado em satélite

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 só um emitir 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.

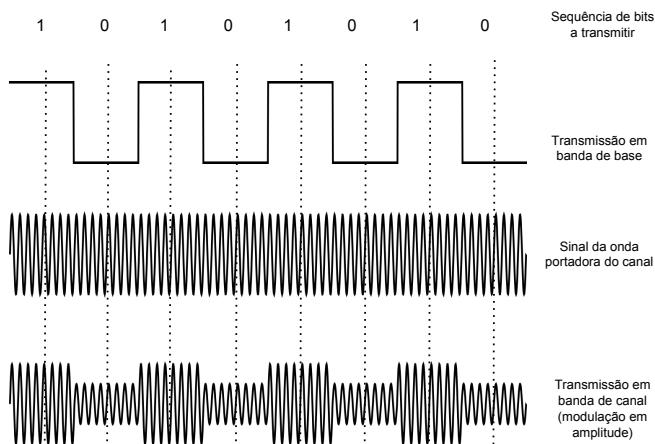


Figura 2.11: Codificação de uma sequência alternada de 0s e 1s em banda de base e em banda de canal com modulação em amplitude da portadora

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

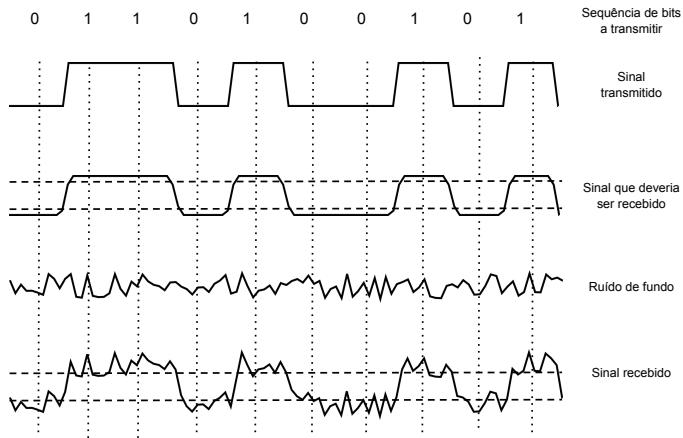


Figura 2.12: O sinal digital original emitido é degradado, quer pela propagação do mesmo pelo canal, quer pela presença de ruído no canal

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 fenómenos 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 noutro 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

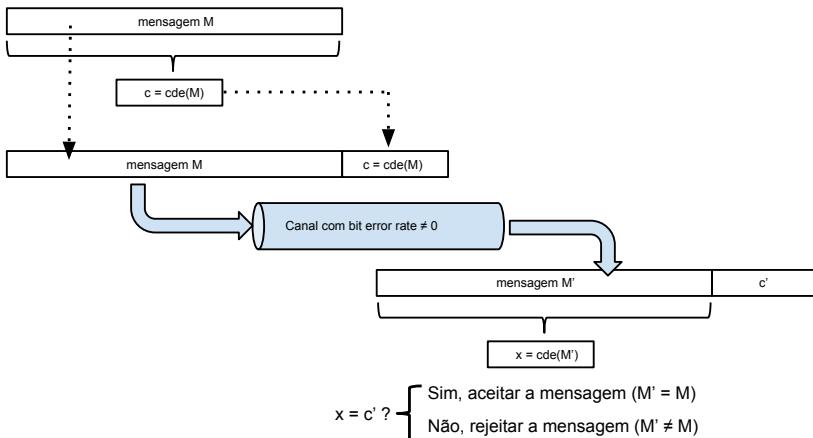


Figura 2.13: Utilização de um código de detecção de erros para controlar se uma mensagem é ou não alterada durante a transmissão pelo canal

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 cílicos 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.

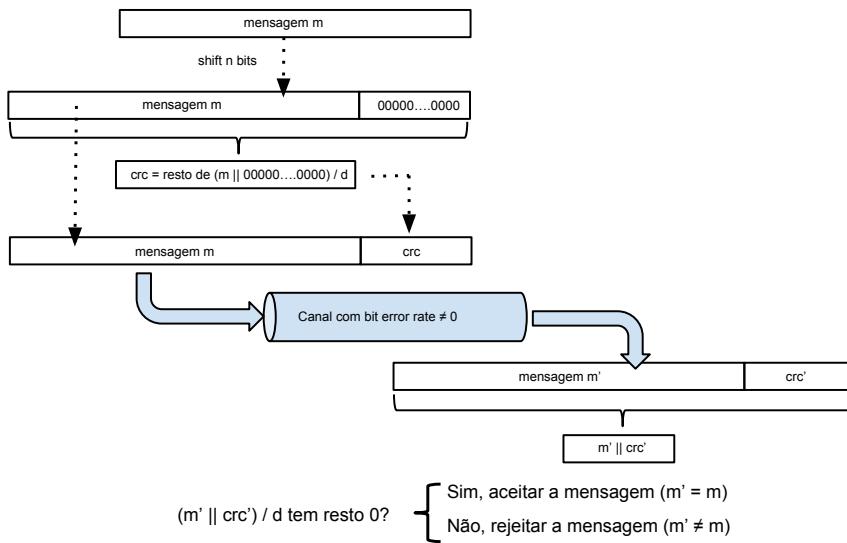


Figura 2.14: Cálculo do código de controlo de erros com o método *Cyclic Redundancy Check (CRC)* – ' \parallel ' denota concatenação

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-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. Estaríamos 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 tratarem 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.

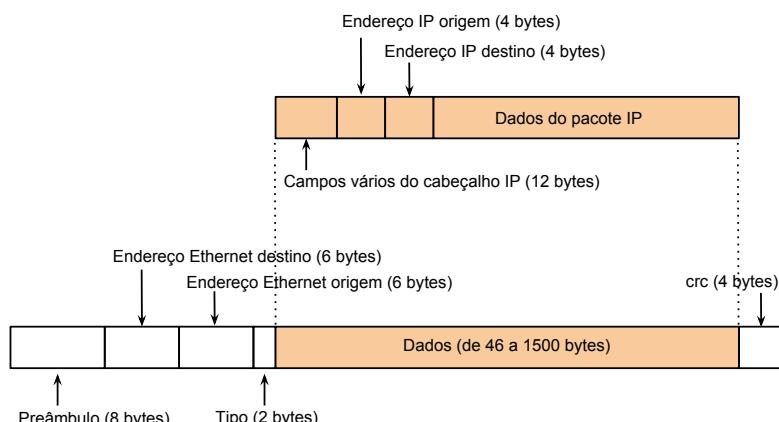


Figura 2.16: Encapsulamento de um pacote IP num *frame* Ethernet

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 comprehende 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.

Detectção 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.

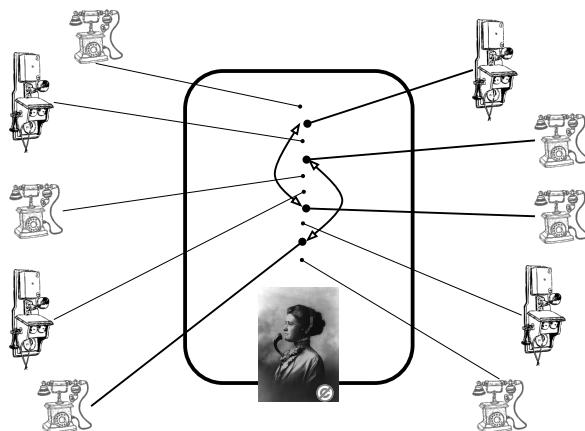


Figura 3.1: Estrutura de uma pequena rede telefónica primitiva

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.

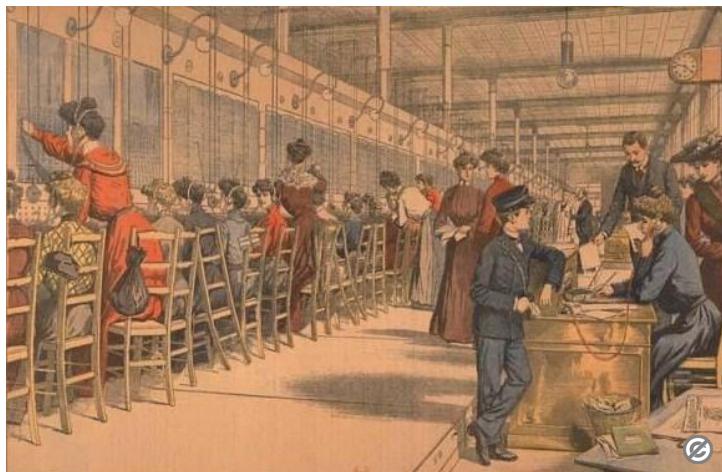


Figura 3.2: Uma central telefónica no início do séc. XX

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 do 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.

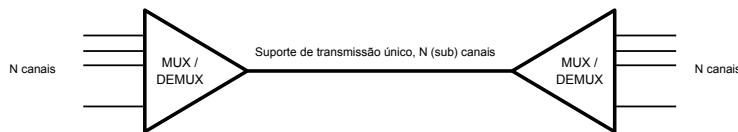


Figura 3.3: Multiplexagem de um suporte de transmissão

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 transmite-se 100.000 bits do sub-canal 1, a seguir transmite-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 transmite-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.

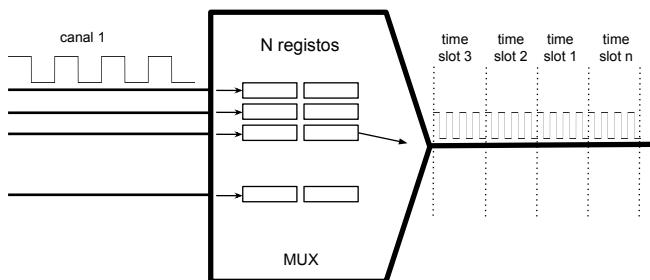


Figura 3.4: Multiplexagem por tempo

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).

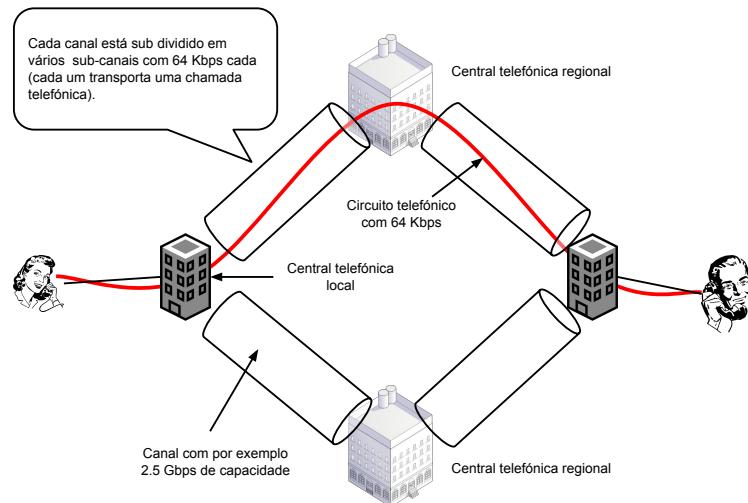


Figura 3.5: Um circuito telefônico típico suportava uma chamada telefônica transmitindo o som digitalizado a 64 Kbps

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.

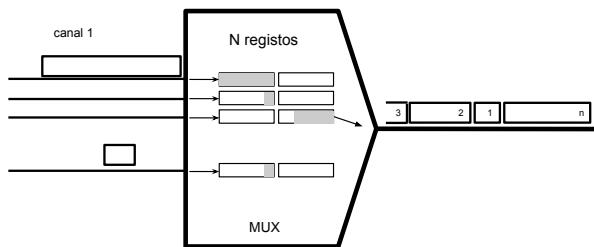


Figura 3.6: Multiplexagem estatística

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-canais 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.

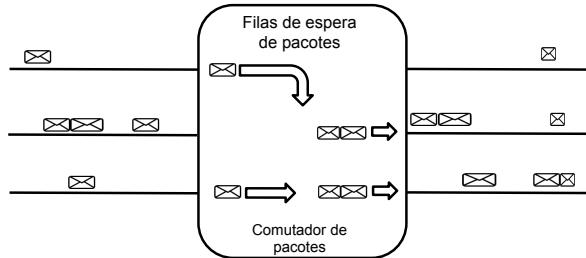


Figura 3.8: Comutação de pacotes com multiplexagem estatística

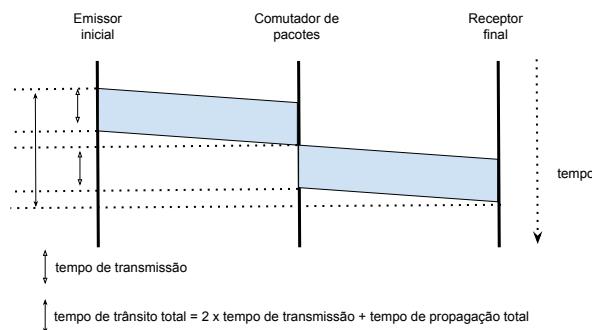


Figura 3.9: Atraso introduzido devido ao tempo de transmissão

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-

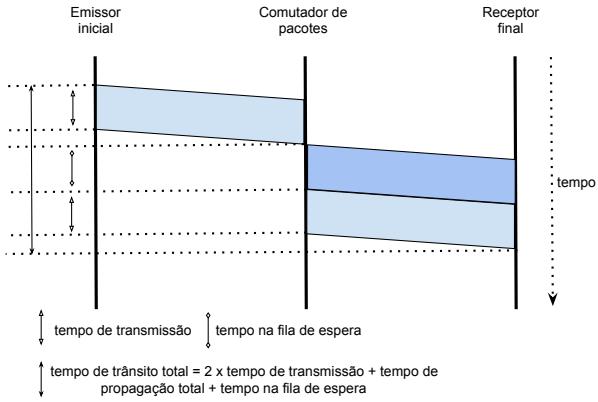


Figura 3.10: Atraso introduzido devido ao tempo de transmissão dos pacotes que já estavam na fila de espera (a mais escuro) e atraso introduzido devido ao tempo de transmissão do pacote

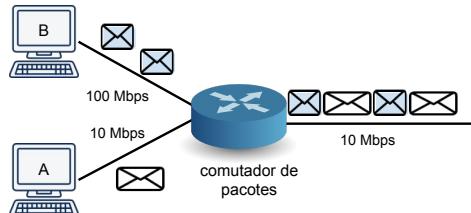


Figura 3.11: Os computadores A e B estão ligados via o comutador de pacotes em cuja interface de saída se forma uma fila de espera

terrupção, a fila de espera de saída 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.

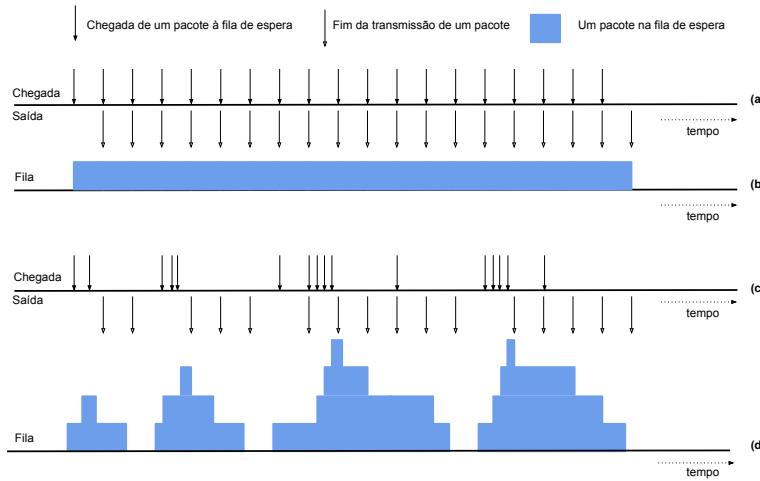


Figura 3.12: Evolução do estado da fila de espera de saída do comutador (Figura 3.11) nas situações: (a) e (b) apenas o computador A emite continuamente pacotes; em (c) e (d) pacotes dos computadores A e B chegam de forma irregular e descoordenada à fila de saída do comutador

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 ($i = 1$ 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, desprezamos 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 chegam 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.

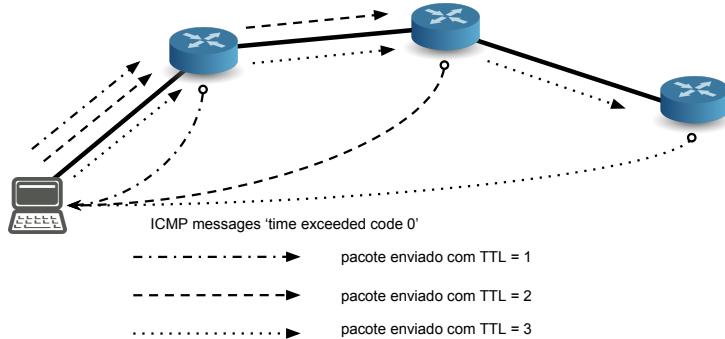


Figura 3.13: Execução do programa **traceroute**

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 dsldevice.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.csb1.ucla.net (169.232.4.4) 180.015 ms
  bd11f1.anderson--cr01f1.anderson.ucla.net (169.232.4.6) 180.748 ms
14 cr01f2.csb1--sr02f2.csb1.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.

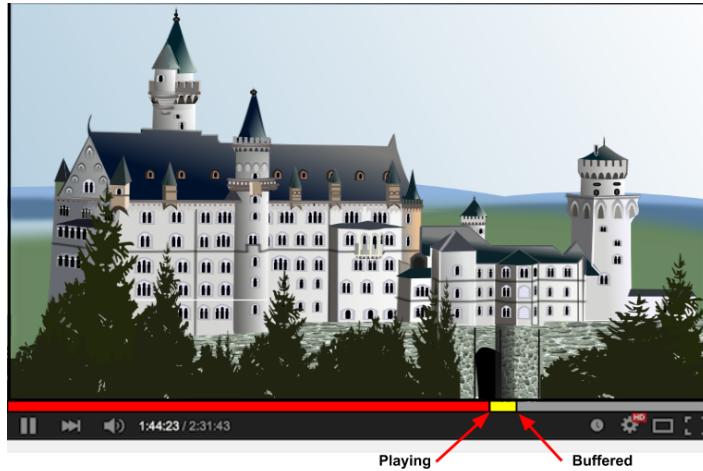


Figura 3.14: Cliente de visualização de vídeo pondo em evidência que o *playback buffer* contém informação multimédia ainda não visualizada

Vamos agora descrever o procedimento genérico usado (o leitor pode seguir a explicação 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,

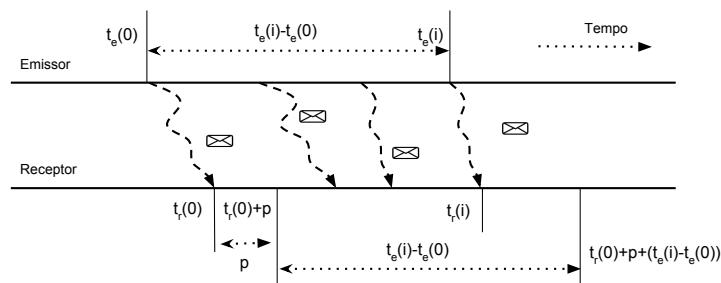


Figura 3.15: Utilização de marcas temporais, de um *playback buffer* e de um *layout delay* (P) para compensar o *jitter*

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 fracção 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 fracção 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, travessam 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?
- (a) $y \times ((r/q \times 10^{-6}) \times (8 \times x \times 2^{20}/r))$
 (b) $(x \times 2^{20} \times 8)/q \times 10^{-6} + y \times z/c + (y - 1) \times (r/q \times 10^{-6})$
 (c) $(x \times 2^{20} \times 8)/q \times 10^{-6} + y \times z/c + y \times (r/q \times 10^{-6})$
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 12.0000 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 “Compre 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 “Compre 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 excepção 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,

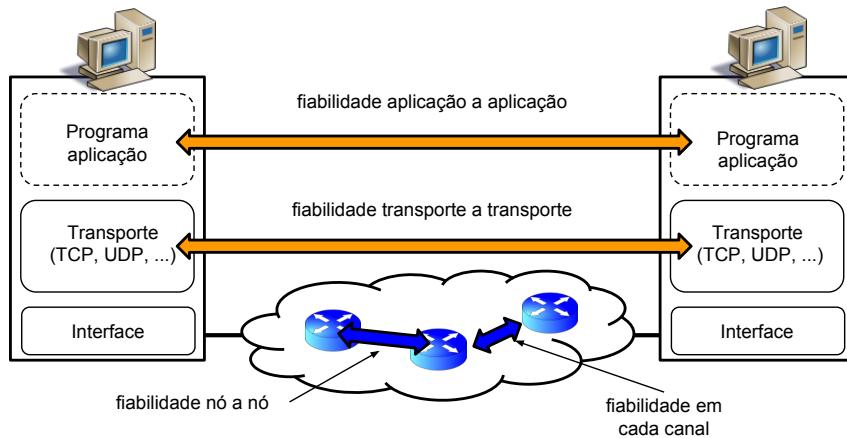


Figura 4.1: Diversas opções para garantir a fiabilidade do transporte dos dados: canal a canal, nó a nó, no protocolo de transporte, nas aplicações

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 “círculo virtual”.

Durante o estabelecimento do círculo 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 círculo 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 círculo 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 círculo 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 funcionalidades 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.

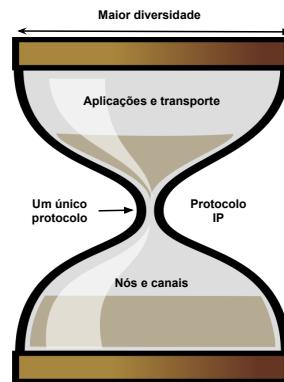


Figura 4.2: O protocolo IP é o gargalo da ampulheta; ele separa a diversidade das aplicações e transportes, da diversidade dos nós de comutação e canais

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 deseável 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.

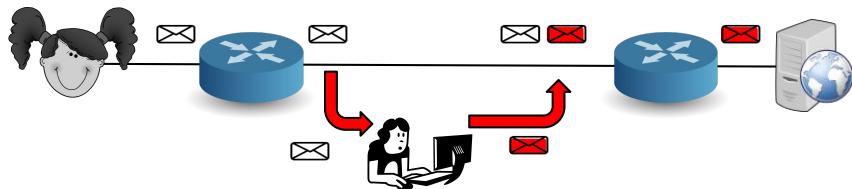


Figura 4.3: Por omissão, na Internet nada impede que os pacotes sejam copiados ou alterados por atacantes suficientemente poderosos e interessados

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 vítima.

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.

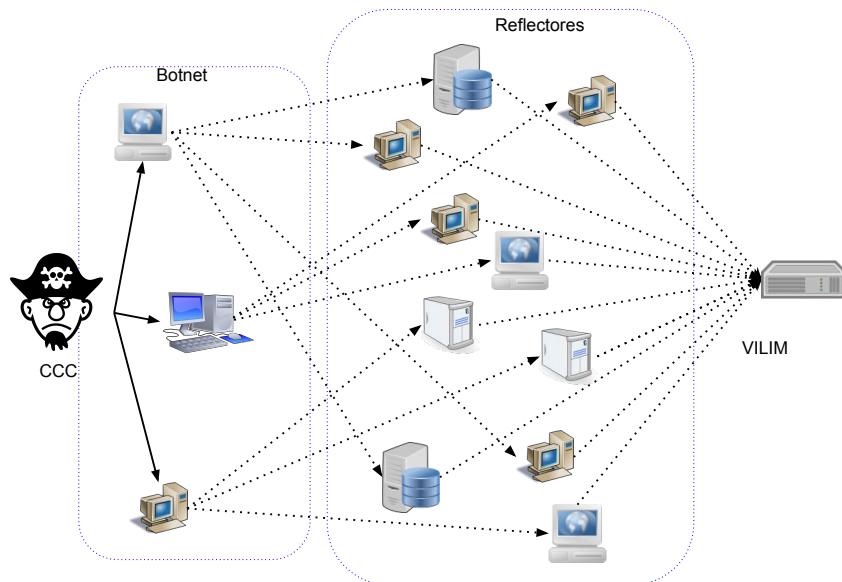


Figura 4.4: Ataque de negação de serviço distribuído por reflexão: CCC, através da sua *botnet*, ataca o computador VILIM

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 contraprodutivo.

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 contratem 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 de 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 do 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.

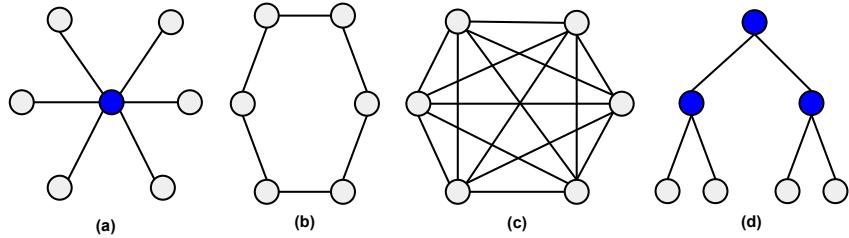


Figura 4.5: Redes (a) estrela (*hub-spoke*), (b) anel (*ring*), (c) malha completa (*full-mesh*), (d) hierárquica (*hierarchical*)

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

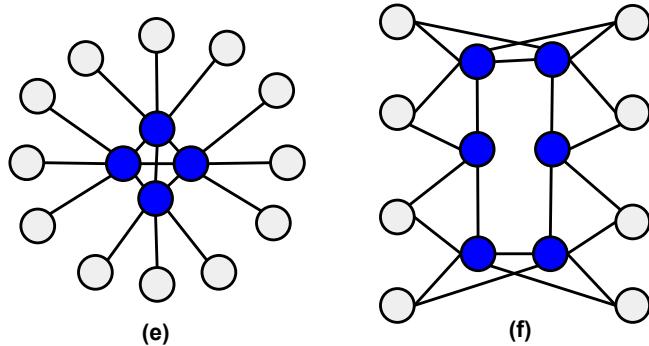


Figura 4.6: Redes (e) estrela com centro reforçado (*hub-spoke*), (f) centro em anel com ligações redundantes

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.

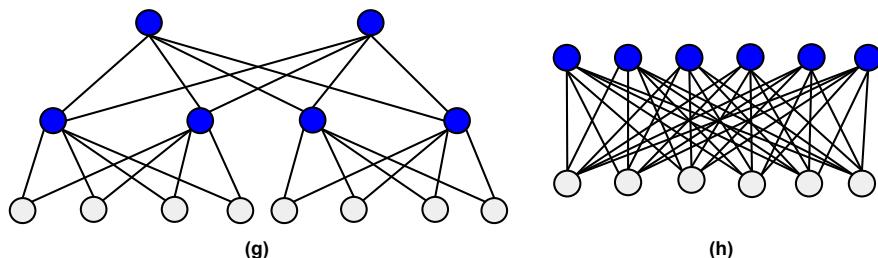


Figura 4.7: Redes (g) Hierárquica duplicada, também conhecida por *Fat tree*, e (h) *Clos network*

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.

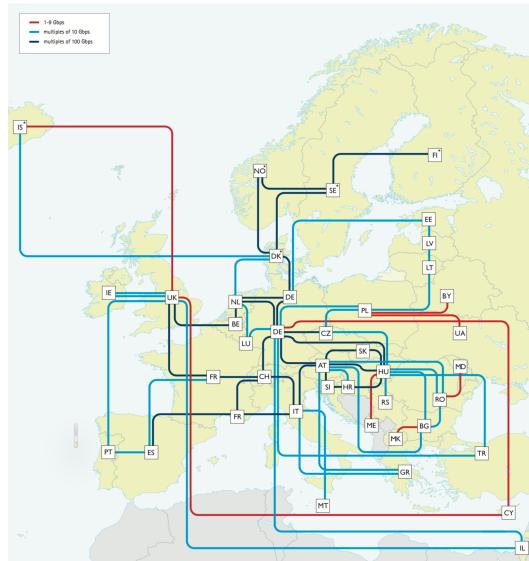


Figura 4.8: Grafo correspondente à rede Géant em 2015, um *backbone* de interligação das redes de investigação europeias. A figura foi extraída da figura semelhante disponível em <http://www.geant.net>.

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 ciclópica.

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 atraídos 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.

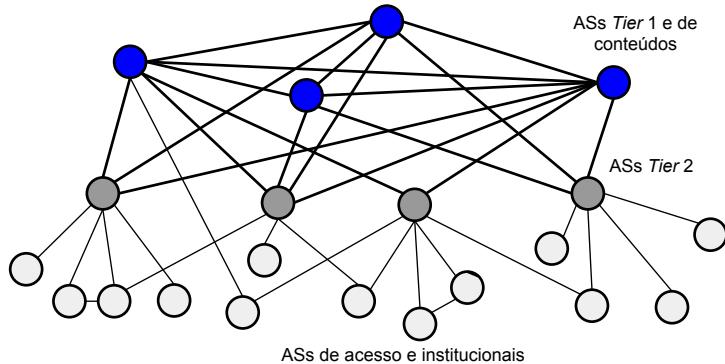


Figura 4.9: A estrutura vagamente hierárquica da Internet

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 porém 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.

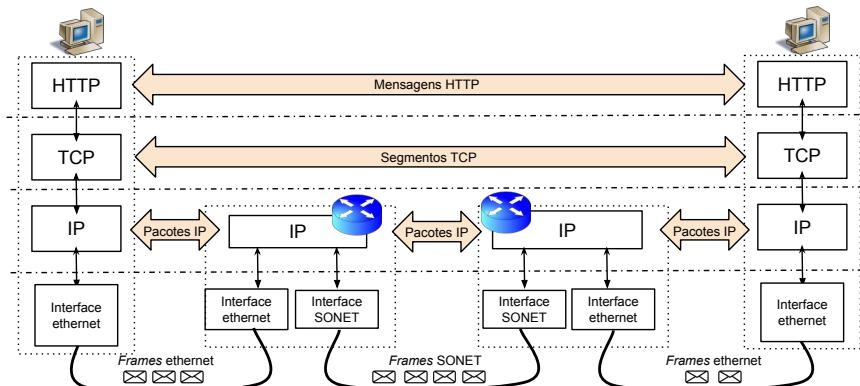


Figura 4.11: O modelo de camadas e a sua concretização em módulos nos computadores e comutadores

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.

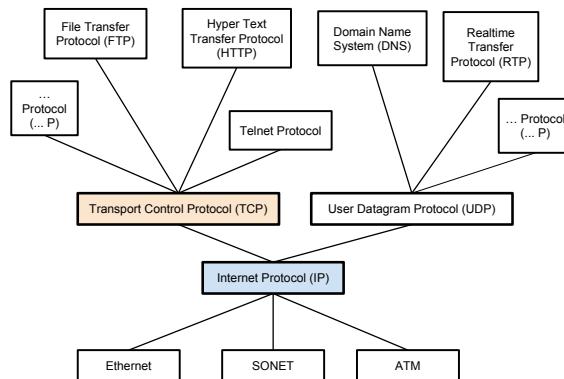


Figura 4.12: Exemplos de protocolos dos diferentes níveis da pilha de protocolos TCP/IP

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

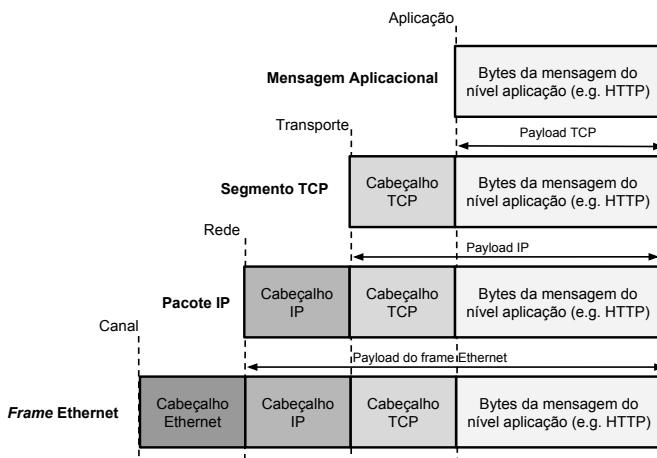


Figura 4.13: Cabeçalhos e encapsulamento nos diferentes níveis

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 tornaria-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

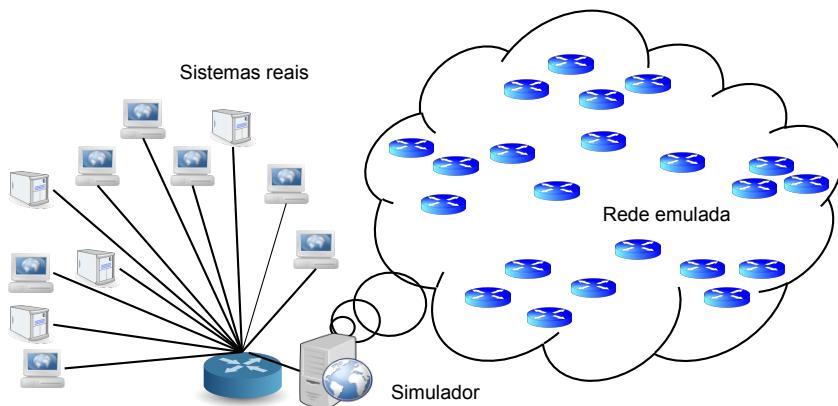


Figura 4.14: Estudo de uma rede de computadores através de um emulador

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.

Privilegiar 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 [Lampson, 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 revisitar 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 medidas 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 desprezável e que nem todos os computadores nos extremos necessitam, devem ser, se possível, relegadas para uma implementação nos extremos.

7. Verdade ou mentira?

- (a) 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.
- (b) Uma funcionalidade equivalente à do protocolo TCP, se não estivesse disponível nos sistemas de operação, não poderia ser implementada noutra nível da rede.
- (c) 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.

- (a) Se um canal ficar inoperacional, como podem os circuitos que o atravessam continuar operacionais?
- (b) Se um comutador de pacotes ficar inoperacional, como podem os circuitos que o atravessam continuar operacionais?
- (c) Numa rede de circuitos os pacotes podem chegar fora de ordem?
- (d) 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.
- (e) 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?

- (a) 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.
- (b) 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.
- (c) 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 `package 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.

¹Package, 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

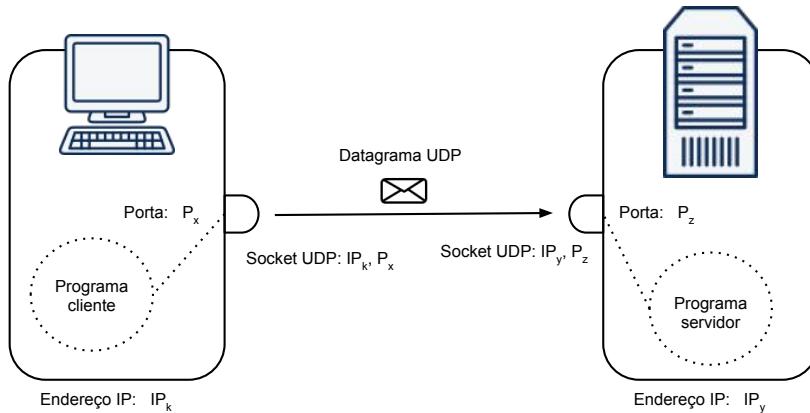


Figura 5.1: Computadores, endereços IP, portas e sockets UDP

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).

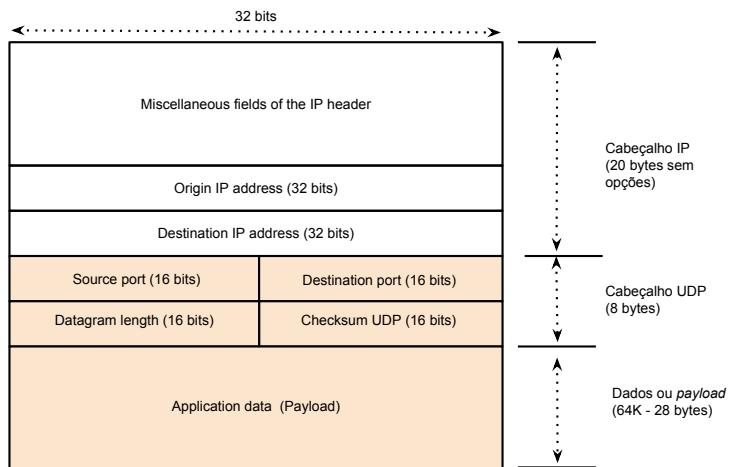


Figura 5.2: Formato de um datagrama UDP encapsulado num pacote IP

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\u0026amp;message:\u0026");
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, podiam-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 datagram 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.

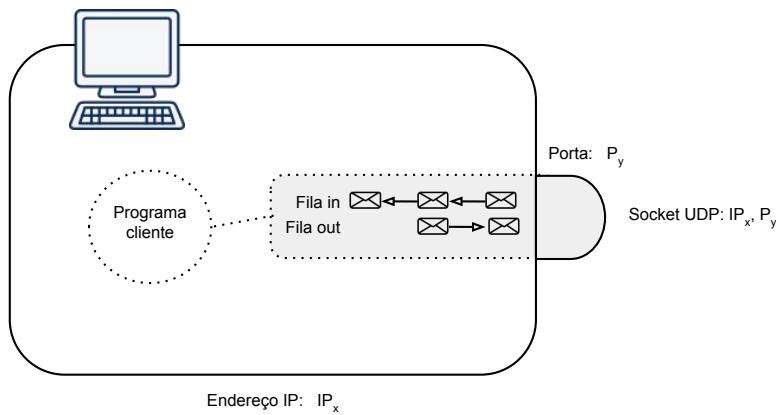


Figura 5.3: Filas de espera associadas a um socket UDP

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 encher-lá. 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 exceções e quaisquer problemas que surgiem traduzir-se-ão em exceções 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 a message: ");
        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 answer is: \"%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: " + 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();
    }
}

```

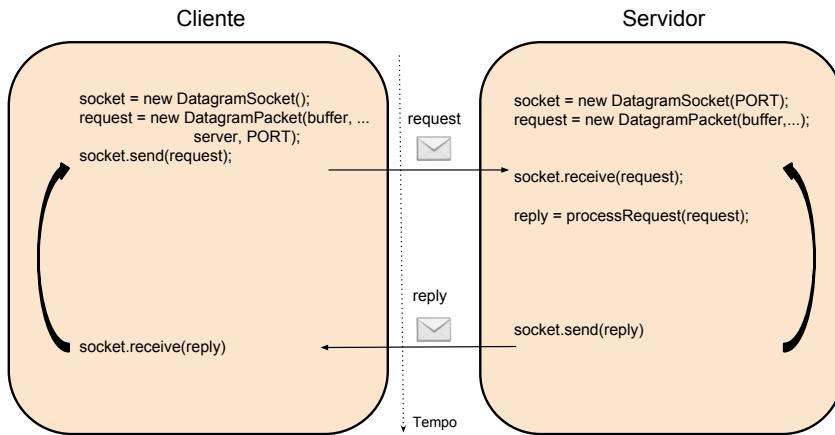


Figura 5.4: Cliente e servidor em comunicação usando sockets UDP em Java

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 começarão 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

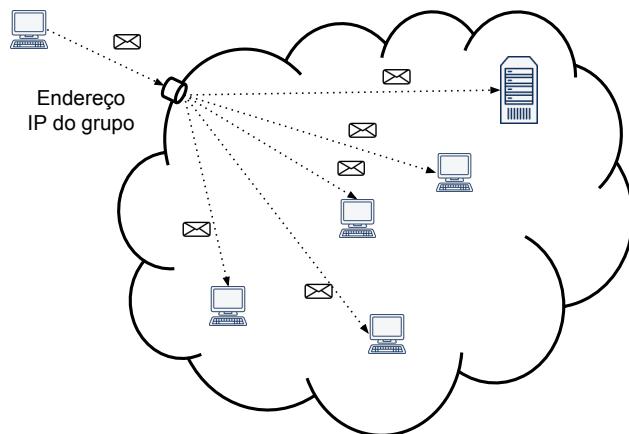


Figura 5.5: Difusão através de IP Multicasting de um datagrama para um grupo de computadores

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 acções 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\u00e1address\u00e1 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 deseje 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:javaSpeakingClockgroup");
            System.exit(0);
        }
        InetAddress group = InetAddress.getByName(args[0]);
        if (!group.isMulticastAddress()) {
            System.err.println("Multicastaddressrequired...");
            System.exit(0);
        }

        DatagramSocket socket = new DatagramSocket();
        for (;;) { // loop for ever
            String msg = "Currenttimeis:" + 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 estabelecer, e o outro é passivo, e terá de 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.

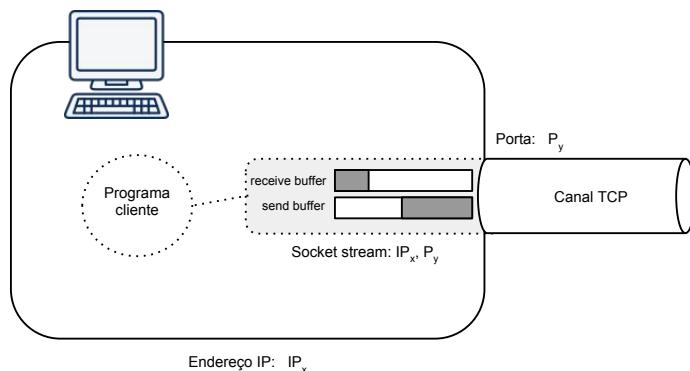
O exemplo merece alguns comentários suplementares. A mensagem a enviar ao servidor é uma sequência de bytes, correspondentes a uma linha de texto, obtida a partir da consola (na linha 21). Repare-se que o `Scanner` usado para ler a linha (criado na linha 19) devolve uma sequência de caracteres sem nenhum 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 mais 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

```

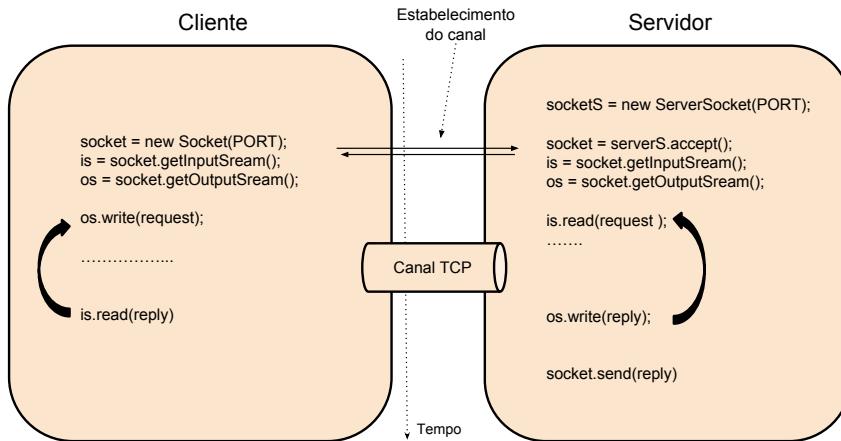


Figura 5.7: Comunicação entre um cliente e um servidor através de um canal TCP

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 `TCP Echo Server` 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 `TCP Echo Server` 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 multi-processadora, 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}

```

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 registrado, 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 log file " + 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 ações 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.out.println("usage:java FileCp fromFileToFile");
            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.out.println("Can't open file" + 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.out.println("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 FileCopy server 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());
            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);
        }
    }
}

```

