Capítulo 1: Fundação

Problema: Construindo uma Rede

Suponha que você queira construir uma rede de computadores com potencial para crescer em proporções globais e suportar aplicações tão diversas quanto teleconferência, vídeo sob demanda, comércio eletrônico, computação distribuída e bibliotecas digitais. Quais tecnologias disponíveis serviriam como blocos de construção subjacentes e que tipo de arquitetura de software você projetaria para integrá-los a um serviço de comunicação eficaz? Responder a essa pergunta é o objetivo principal deste livro: descrever os materiais de construção disponíveis e, em seguida, mostrar como eles podem ser usados para construir uma rede do zero.

Antes de entendermos como projetar uma rede de computadores, precisamos primeiro entender o que exatamente é uma rede de computadores. Antigamente, o termo " *rede"* significava o conjunto de linhas seriais usadas para conectar terminais burros a computadores mainframe. Outras redes importantes incluem a rede telefônica de voz e a rede de TV a cabo, usada para disseminar sinais de vídeo. O principal ponto em comum entre essas redes é que elas são especializadas para lidar com um tipo específico de dados (teclas, voz ou vídeo) e normalmente se conectam a dispositivos com finalidades específicas (terminais, receptores portáteis e aparelhos de televisão).

O que distingue uma rede de computadores desses outros tipos de rede? Provavelmente, a característica mais importante de uma rede de computadores é sua generalidade. Redes de computadores são construídas principalmente com hardware programável de uso geral e não são otimizadas para uma aplicação específica, como fazer chamadas telefônicas ou transmitir sinais de televisão. Em vez disso, elas são capazes de transportar muitos tipos diferentes de dados e suportam uma ampla e crescente gama de aplicações. As redes de computadores atuais praticamente

substituíram as funções anteriormente desempenhadas por redes de uso único. Este capítulo analisa algumas aplicações típicas de redes de computadores e discute os requisitos que um projetista de rede que deseja oferecer suporte a tais aplicações deve conhecer.

Uma vez compreendidos os requisitos, como procederemos? Felizmente, não construiremos a primeira rede. Outros, principalmente a comunidade de pesquisadores responsáveis pela internet, já nos antecederam. Usaremos a vasta experiência gerada pela internet para orientar nosso projeto. Essa experiência está incorporada em uma *arquitetura de rede* que identifica os componentes de hardware e software disponíveis e mostra como eles podem ser organizados para formar um sistema de rede completo.

Além de entender como as redes são construídas, é cada vez mais importante entender como elas são operadas ou gerenciadas e como as aplicações de rede são desenvolvidas. Quase todos nós agora temos redes de computadores em nossas casas, escritórios e, em alguns casos, em nossos carros, então operar redes não é mais uma questão exclusiva de alguns especialistas. E com a proliferação de smartphones, muito mais pessoas desta geração estão desenvolvendo aplicações em rede do que no passado. Portanto, precisamos considerar as redes sob estas múltiplas perspectivas: construtores, operadores, desenvolvedores de aplicações.

Para nos iniciar no caminho rumo à compreensão de como construir, operar e programar uma rede, este capítulo aborda quatro aspectos. Primeiro, explora os requisitos que diferentes aplicações e diferentes comunidades de pessoas impõem à rede. Segundo, apresenta o conceito de arquitetura de rede, que estabelece a base para o restante do livro. Terceiro, apresenta alguns dos elementos-chave na implementação de redes de computadores. Por fim, identifica as principais métricas utilizadas para avaliar o desempenho de redes de computadores.

1.1 Aplicações

A maioria das pessoas conhece a internet por meio de suas aplicações: a World Wide Web, e-mail, mídias sociais, streaming de música ou filmes, videoconferência, mensagens instantâneas, compartilhamento de arquivos, para citar apenas alguns exemplos. Ou seja, interagimos com a internet como *usuários* da rede. Os usuários da internet representam a maior classe de pessoas que interagem com a internet de alguma forma, mas existem vários outros grupos importantes.

Há o grupo de pessoas que *cria* os aplicativos, um grupo que se expandiu muito nos últimos anos, à medida que plataformas de programação poderosas e novos dispositivos, como smartphones, criaram novas oportunidades para desenvolver aplicativos rapidamente e levá-los a um grande mercado.

Há também aqueles que *operam* ou *gerenciam* redes — um trabalho geralmente realizado nos bastidores, mas crítico e frequentemente muito complexo. Com a prevalência das redes domésticas, cada vez mais pessoas também estão se tornando, ainda que em pequena escala, operadores de rede.

Por fim, há aqueles que *projetam* e *constroem* os dispositivos e protocolos que coletivamente compõem a internet. Esse público final é o alvo tradicional de livros didáticos de redes como este e continuará sendo nosso foco principal. No entanto, ao longo deste livro, também consideraremos as perspectivas de desenvolvedores de aplicações e operadores de rede.

Considerar essas perspectivas nos permitirá compreender melhor os diversos requisitos que uma rede deve atender. Os desenvolvedores de aplicativos também serão capazes de criar aplicativos que funcionem melhor se entenderem como a tecnologia subjacente funciona e interage com os aplicativos. Portanto, antes de

começarmos a descobrir como construir uma rede, vamos analisar mais detalhadamente os tipos de aplicativos que as redes atuais suportam.

1.1.1 Classes de Aplicações

A World Wide Web é a aplicação da internet que catapultou a internet de uma ferramenta um tanto obscura, usada principalmente por cientistas e engenheiros, para o fenômeno popular que é hoje. A própria web se tornou uma plataforma tão poderosa que muitas pessoas a confundem com a internet, e seria um pouco exagerado dizer que a web é uma aplicação única.

Em sua forma básica, a Web apresenta uma interface intuitivamente simples. Os usuários visualizam páginas repletas de objetos textuais e gráficos e clicam nos objetos sobre os quais desejam aprender mais, e uma nova página correspondente é exibida. A maioria das pessoas também sabe que, por baixo dos panos, cada objeto selecionável em uma página está vinculado a um identificador para a próxima página ou objeto a ser visualizado. Esse identificador, chamado de Localizador Uniforme de Recursos (URL), fornece uma maneira de identificar todos os objetos possíveis que podem ser visualizados a partir do seu navegador web. Por exemplo,

http://www.cs.princeton.edu/~llp/index.html

é o URL de uma página que fornece informações sobre um dos autores deste livro: a sequência `http`indica que o Protocolo de Transferência de Hipertexto (HTTP) deve ser usado para baixar a página, `www.cs.princeton.edu`é o nome da máquina que serve a página e `/~llp/index.html`identifica exclusivamente a página inicial de Larry neste site.

O que a maioria dos usuários da web não sabe, no entanto, é que clicando em apenas uma dessas URLs, mais de uma dúzia de mensagens podem ser trocadas pela Internet, e muito mais do que isso se a página da web for complicada com muitos objetos incorporados. Essa troca de mensagens inclui até seis mensagens para traduzir o nome do servidor (`www.cs.princeton.edu`) em seu endereço de Protocolo de

Internet (IP) (`128.112.136.35`), três mensagens para configurar uma conexão TCP (Transmission Control Protocol) entre seu navegador e este servidor, quatro mensagens para seu navegador enviar a solicitação HTTP "GET" e o servidor responder com a página solicitada (e para cada lado confirmar o recebimento dessa mensagem) e quatro mensagens para interromper a conexão TCP. Claro, isso não inclui os milhões de mensagens trocadas pelos nós da Internet ao longo do dia, apenas para informar uns aos outros que eles existem e estão prontos para servir páginas da web, traduzir nomes em endereços e encaminhar mensagens para seu destino final.

Outra classe de aplicação amplamente difundida da Internet é a entrega de áudio e vídeo por streaming. Serviços como vídeo sob demanda e rádio na Internet utilizam essa tecnologia. Embora frequentemente acessamos um site para iniciar uma sessão de streaming, a entrega de áudio e vídeo apresenta algumas diferenças importantes em relação à busca de uma simples página web com texto e imagens. Por exemplo, muitas vezes não é desejável baixar um arquivo de vídeo inteiro — um processo que pode levar alguns minutos — antes de assistir à primeira cena. O streaming de áudio e vídeo implica uma transferência mais rápida de mensagens do remetente para o destinatário, e o destinatário exibe o vídeo ou reproduz o áudio praticamente no momento em que ele chega.

Observe que a diferença entre aplicativos de streaming e a entrega mais tradicional de texto, gráficos e imagens é que os humanos consomem fluxos de áudio e vídeo de forma contínua, e descontinuidades — na forma de sons pulados ou vídeo parado — não são aceitáveis. Em contraste, uma página normal (não streaming) pode ser entregue e lida em partes. Essa diferença afeta a forma como a rede suporta essas diferentes classes de aplicativos.

Uma classe de aplicação sutilmente diferente é a de áudio e vídeo *em tempo real* . Essas aplicações têm restrições de tempo consideravelmente mais rígidas do que as de streaming. Ao usar um aplicativo de voz sobre IP, como o Skype, ou um aplicativo de videoconferência, as interações entre os participantes devem ser oportunas.

Quando uma pessoa em uma extremidade gesticula, essa ação deve ser exibida na outra extremidade o mais rápido possível. 1

1

Não exatamente "o mais rápido possível"... Pesquisas sobre fatores humanos indicam que 300 ms é um limite superior razoável para o quanto de atraso de ida e volta pode ser tolerado em uma chamada telefônica antes que os humanos reclamem, e um atraso de 100 ms parece muito bom.

Quando uma pessoa tenta interromper outra, a pessoa interrompida precisa ouvir isso o mais rápido possível e decidir se permite a interrupção ou continua falando sobre quem a interrompe. Muito atraso nesse tipo de ambiente torna o sistema inutilizável. Compare isso com o vídeo sob demanda, em que, se levar vários segundos entre o momento em que o usuário inicia o vídeo e a primeira imagem é exibida, o serviço ainda é considerado satisfatório. Além disso, aplicativos interativos geralmente envolvem fluxos de áudio e/ou vídeo em ambas as direções, enquanto um aplicativo de streaming provavelmente envia vídeo ou áudio em apenas uma direção.

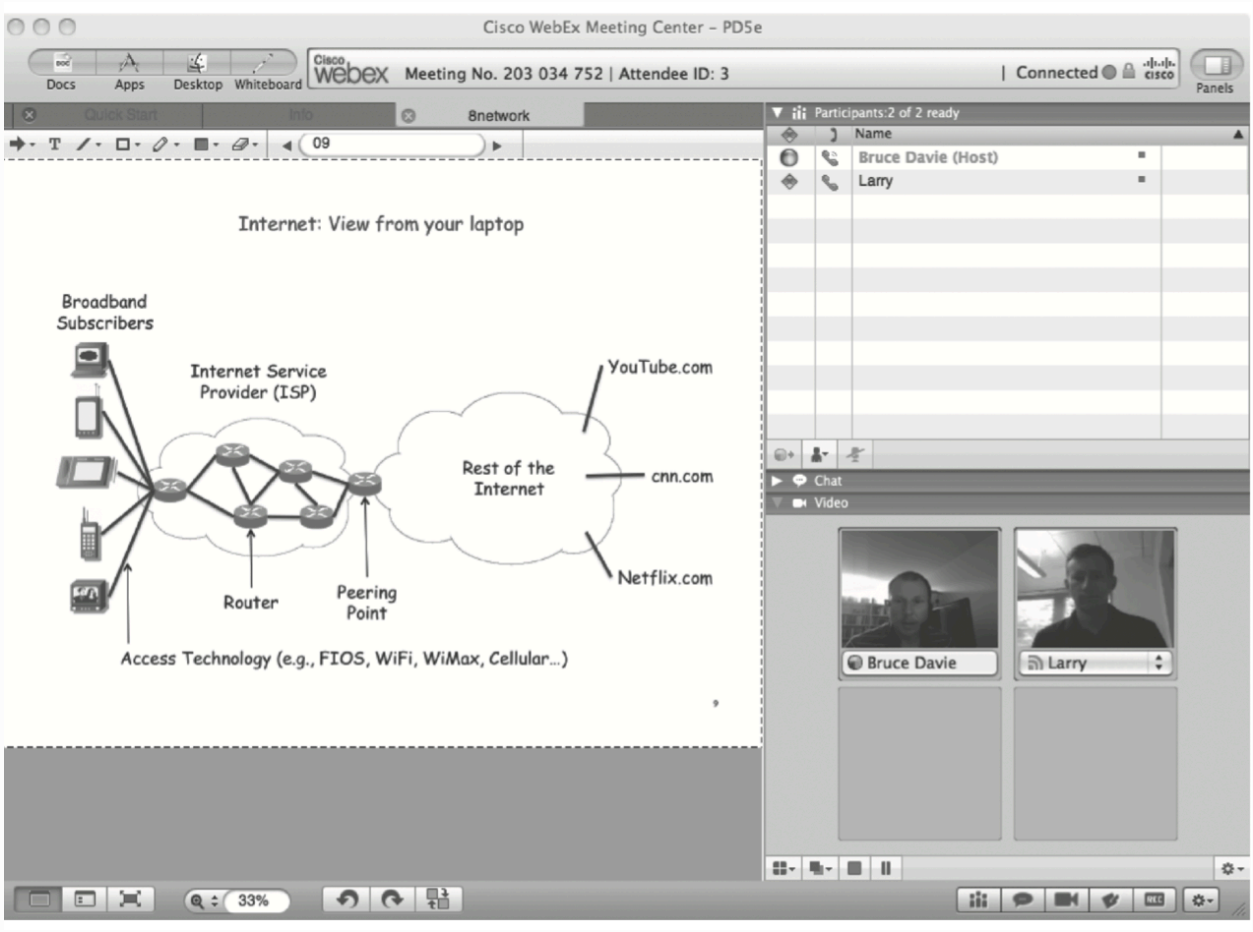

Figura 1. *Um aplicativo multimídia incluindo videoconferência.*

Ferramentas de videoconferência que rodam pela internet existem desde o início da década de 1990, mas alcançaram amplo uso nos últimos anos, com diversos produtos comerciais no mercado. Um exemplo de um desses sistemas é mostrado na Figura 1. Assim como baixar uma página da web envolve um pouco mais do que aparenta, o mesmo ocorre com aplicativos de vídeo. Ajustar o conteúdo de vídeo a uma rede de largura de banda relativamente baixa, por exemplo, ou garantir que o vídeo e o áudio permaneçam sincronizados e cheguem a tempo para uma boa experiência do usuário são problemas com os quais os projetistas de redes e protocolos precisam se preocupar. Analisaremos essas e muitas outras questões relacionadas a aplicativos multimídia mais adiante neste livro.

Embora sejam apenas dois exemplos, baixar páginas da web e participar de uma videoconferência demonstram a diversidade de aplicações que podem ser construídas

sobre a internet e sugerem a complexidade do design da internet. Mais adiante neste livro, desenvolveremos uma taxonomia mais completa dos tipos de aplicações para ajudar a orientar nossa discussão sobre as principais decisões de design à medida que buscamos construir, operar e usar redes. O livro conclui revisitando essas duas aplicações específicas, bem como várias outras que ilustram a amplitude do que é possível na internet atual.

Por enquanto, esta rápida olhada em algumas aplicações típicas será suficiente para nos permitir começar a analisar os problemas que devem ser abordados se quisermos construir uma rede que suporte tal diversidade de aplicações.

1.2 Requisitos

Estabelecemos uma meta ambiciosa para nós mesmos: entender como construir uma rede de computadores do zero. Nossa abordagem para atingir essa meta será partir dos princípios básicos e, em seguida, fazer os tipos de perguntas que naturalmente faríamos se estivéssemos construindo uma rede real. Em cada etapa, usaremos os protocolos atuais para ilustrar as diversas opções de projeto disponíveis, mas não aceitaremos esses artefatos existentes como verdade absoluta. Em vez disso, perguntaremos (e responderemos) por *que* as redes são projetadas da maneira como são. Embora seja tentador se contentar em apenas entender como as coisas são feitas hoje, é importante reconhecer os conceitos subjacentes, pois as redes estão em constante mudança à medida que a tecnologia evolui e novas aplicações são inventadas. Em nossa experiência, uma vez que você entenda as ideias fundamentais, qualquer novo protocolo com o qual se depare será relativamente fácil de assimilar.

1.2.1 Partes interessadas

Como observamos acima, um estudante de redes pode adotar diversas perspectivas. Quando escrevemos a primeira edição deste livro, a maioria da população não tinha

acesso à internet, e aqueles que tinham, o tinham no trabalho, na universidade ou por meio de um modem discado em casa. O conjunto de aplicativos populares podia ser contado nos dedos. Assim, como a maioria dos livros da época, o nosso se concentrava na perspectiva de alguém que projetaria equipamentos e protocolos de rede. Continuamos a nos concentrar nessa perspectiva e nossa esperança é que, após a leitura deste livro, você saiba como projetar os equipamentos e protocolos de rede do futuro.

No entanto, também queremos abordar as perspectivas de duas outras partes interessadas: aquelas que desenvolvem aplicações em rede e aquelas que gerenciam ou operam redes. Vamos considerar como essas três partes interessadas podem listar seus requisitos para uma rede:

- Um *programador de aplicativos* listaria os serviços que seu aplicativo precisa: por exemplo, uma garantia de que cada mensagem enviada pelo aplicativo será entregue sem erros dentro de um determinado período de tempo ou a capacidade de alternar suavemente entre diferentes conexões de rede conforme o usuário se movimenta.
- Um *operador de rede* listaria as características de um sistema que é fácil de administrar e gerenciar: por exemplo, no qual falhas podem ser facilmente isoladas, novos dispositivos podem ser adicionados à rede e configurados corretamente, e é fácil contabilizar o uso.
- Um *projetista de rede* listaria as propriedades de um projeto com boa relação custo-benefício: por exemplo, que os recursos da rede sejam utilizados de forma eficiente e alocados de forma justa para diferentes usuários. Questões de desempenho também provavelmente serão importantes.

Esta seção tenta destilar os requisitos de diferentes partes interessadas em uma introdução de alto nível às principais considerações que orientam o design da rede e, ao fazer isso, identificar os desafios abordados no restante deste livro.

1.2.2 Conectividade Escalável

Começando pelo óbvio, uma rede deve fornecer conectividade entre um conjunto de computadores. Às vezes, basta construir uma rede limitada que conecte apenas algumas máquinas selecionadas. De fato, por razões de privacidade e segurança, muitas redes privadas (corporativas) têm o objetivo explícito de limitar o conjunto de máquinas conectadas. Em contraste, outras redes (das quais a Internet é o principal exemplo) são projetadas para crescer de uma forma que lhes permita o potencial de conectar todos os computadores do mundo. Um sistema projetado para suportar o crescimento a um tamanho arbitrariamente grande é considerado *escalável* . Usando a Internet como modelo, este livro aborda o desafio da escalabilidade.

Para entender melhor os requisitos de conectividade, precisamos analisar mais detalhadamente como os computadores são conectados em uma rede. A conectividade ocorre em muitos níveis diferentes. No nível mais baixo, uma rede pode consistir em dois ou mais computadores conectados diretamente por algum meio físico, como um cabo coaxial ou uma fibra óptica. Chamamos esse meio físico de *link* e, frequentemente, nos referimos aos computadores que ele conecta como *nós* . (Às vezes, um nó é uma peça de hardware mais especializada do que um computador, mas ignoramos essa distinção para os propósitos desta discussão.) Conforme ilustrado na Figura 2 , os links físicos às vezes são limitados a um par de nós (tal link é chamado de *ponto a ponto*), enquanto em outros casos mais de dois nós podem compartilhar um único link físico (tal link é chamado de *acesso múltiplo*). Links sem fio, como aqueles fornecidos por redes celulares e redes Wi-Fi, são uma classe importante de links de acesso múltiplo. É sempre o caso de que os links de acesso múltiplo são limitados em tamanho, em termos tanto da distância geográfica que podem cobrir quanto do número de nós que podem conectar. Por esse motivo, muitas vezes implementam a chamada *última milha* , conectando os usuários finais ao restante da rede.

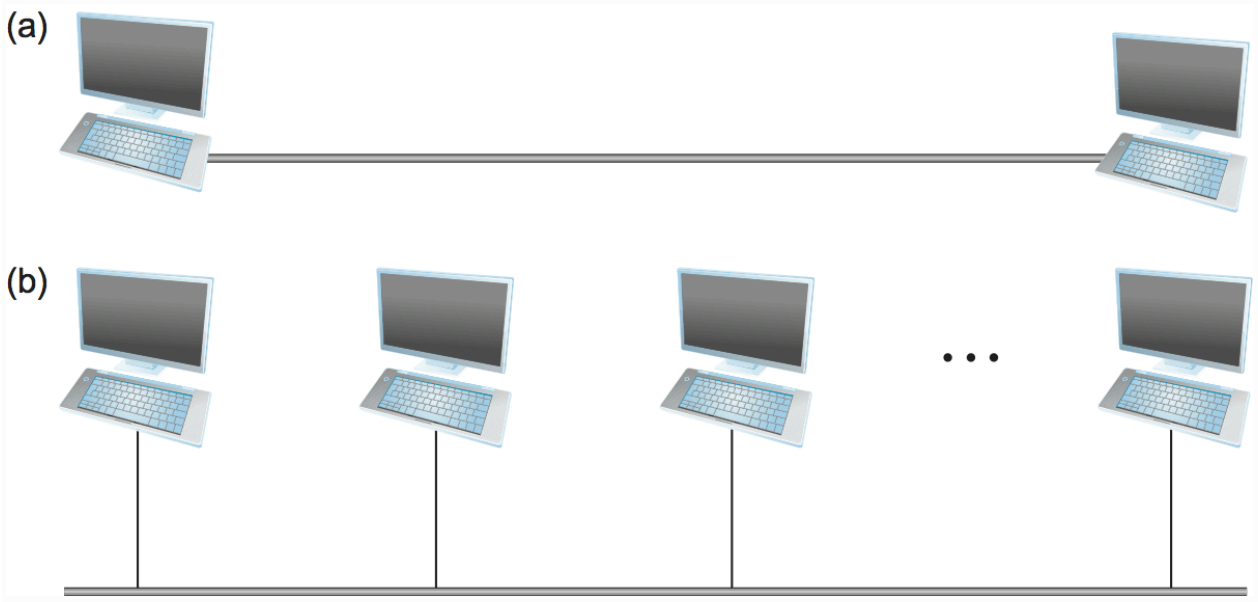

Figura 2. *Links diretos: (a) ponto a ponto; (b) acesso múltiplo.*

Se as redes de computadores se limitassem a situações em que todos os nós estivessem diretamente conectados entre si por um meio físico comum, as redes seriam muito limitadas no número de computadores que poderiam conectar, ou o número de fios que saíam da parte traseira de cada nó se tornaria rapidamente incontrolável e muito caro. Felizmente, a conectividade entre dois nós não implica necessariamente uma conexão física direta entre eles — a conectividade indireta pode ser alcançada entre um conjunto de nós cooperantes. Considere os dois exemplos a seguir de como um conjunto de computadores pode ser conectado indiretamente.

A Figura 3 mostra um conjunto de nós, cada um dos quais está conectado a um ou mais links ponto a ponto. Esses nós que estão conectados a pelo menos dois links executam software que encaminha os dados recebidos em um link para outro. Se organizados de forma sistemática, esses nós de encaminhamento formam uma *rede comutada* . Existem vários tipos de redes comutadas, das quais as duas mais comuns são *a comutação de circuitos* e *a comutação de pacotes* . A primeira é mais notavelmente empregada pelo sistema telefônico, enquanto a segunda é usada para a esmagadora maioria das redes de computadores e será o foco deste livro. (A comutação de circuitos está, no entanto, fazendo um certo retorno no reino das redes

ópticas, o que se torna importante à medida que a demanda por capacidade de rede cresce constantemente.) A característica importante das redes comutadas por pacotes é que os nós em tal rede enviam blocos discretos de dados uns aos outros. Pense nesses blocos de dados como correspondentes a algum pedaço de dados de aplicativo, como um arquivo, um pedaço de e-mail ou uma imagem. Chamamos cada bloco de dados de um *pacote* ou uma *mensagem* e, por enquanto, usamos esses termos de forma intercambiável.

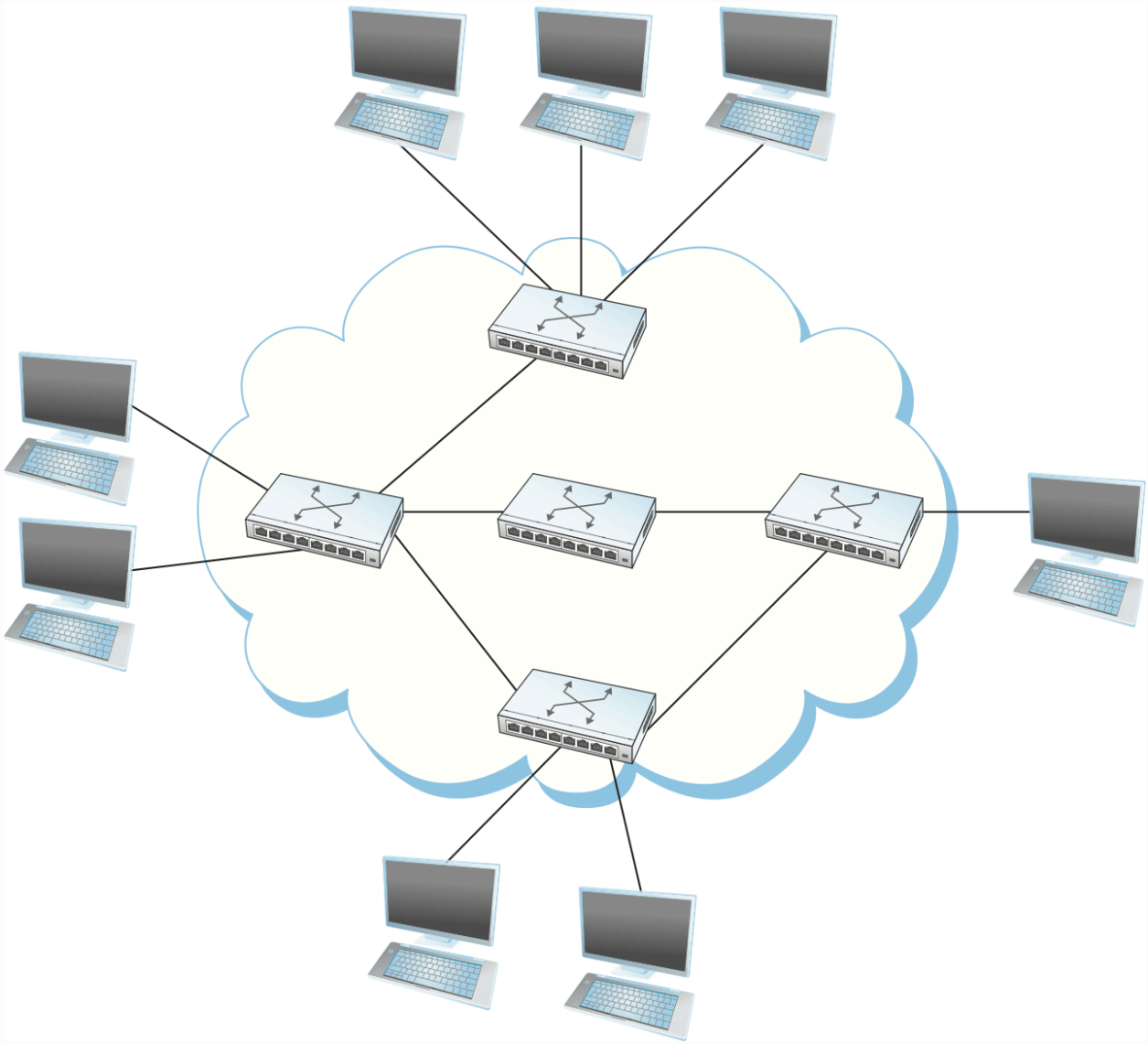

Figura 3. *Rede comutada.*

Redes comutadas por pacotes normalmente utilizam uma estratégia chamada *armazenar e encaminhar* . Como o nome sugere, cada nó em uma rede armazenar e encaminhar primeiro recebe um pacote completo por meio de algum enlace, armazena o pacote em sua memória interna e, em seguida, encaminha o pacote completo para o próximo nó. Em contraste, uma rede comutada por circuitos primeiro estabelece um circuito dedicado em uma sequência de enlaces e, em seguida, permite que o nó de origem envie um fluxo de bits por esse circuito para um nó de destino. O principal motivo para usar a comutação de pacotes em vez da comutação de circuitos em uma rede de computadores é a eficiência, discutida na próxima subseção.

A nuvem na Figura 3 distingue entre os nós internos que *implementam* a rede (eles são comumente chamados de *switches* , e sua função principal é armazenar e encaminhar pacotes) e os nós externos que *usam* a rede (eles são tradicionalmente chamados de *hosts* , e eles dão suporte aos usuários e executam programas de aplicativos). Observe também que a nuvem é um dos ícones mais importantes das redes de computadores. Em geral, usamos uma nuvem para denotar qualquer tipo de rede, seja um único link ponto a ponto, um link de acesso múltiplo ou uma rede comutada. Portanto, sempre que você vir uma nuvem usada em uma figura, você pode pensar nela como um espaço reservado para qualquer uma das tecnologias de rede abordadas neste livro. 1

1

O uso de nuvens para representar redes antecede o termo *computação em nuvem* em pelo menos algumas décadas, mas há uma conexão cada vez mais rica entre esses dois usos, que exploramos na discussão *sobre Perspectiva* no final de cada capítulo.

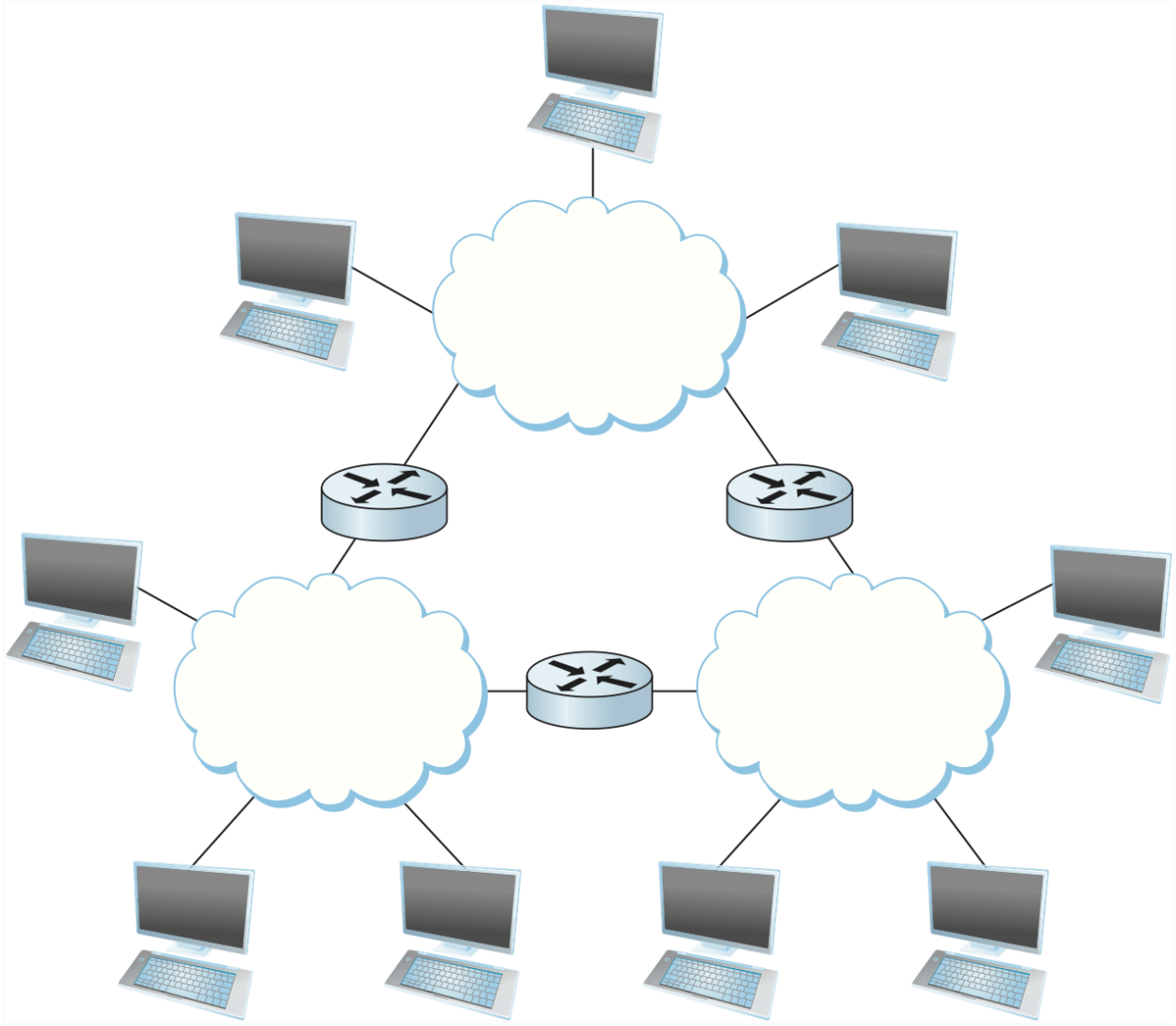

Figura 4. *Interconexão de redes.*

Uma segunda maneira pela qual um conjunto de computadores pode ser indiretamente conectado é mostrada na Figura 4. Nessa situação, um conjunto de redes independentes (nuvens) são interconectadas para formar uma *interconexão de redes* , ou internet para abreviar. Adotamos a convenção da Internet de nos referir a uma interconexão de redes genérica como uma internet *com i* minúsculo , e a Internet TCP/IP que todos usamos todos os dias como a Internet com *I* maiúsculo . Um nó conectado a duas ou mais redes é comumente chamado de *roteador* ou *gateway* , e desempenha praticamente o mesmo papel de um switch — ele encaminha mensagens

de uma rede para outra. Observe que uma internet pode ser vista como outro tipo de rede, o que significa que uma internet pode ser construída a partir de um conjunto de internets. Assim, podemos construir recursivamente redes arbitrariamente grandes interconectando nuvens para formar nuvens maiores. Pode-se argumentar razoavelmente que essa ideia de interconectar redes amplamente diferentes foi a inovação fundamental da Internet e que o crescimento bem-sucedido da Internet para um tamanho global e bilhões de nós foi o resultado de algumas decisões de design muito boas dos primeiros arquitetos da Internet, que discutiremos mais tarde.

Só porque um conjunto de hosts está direta ou indiretamente conectado entre si não significa que conseguimos fornecer conectividade host-a-host. O requisito final é que cada nó deve ser capaz de dizer com qual dos outros nós da rede ele deseja se comunicar. Isso é feito atribuindo um *endereço* a cada nó. Um endereço é uma sequência de bytes que identifica um nó; ou seja, a rede pode usar o endereço de um nó para distingui-lo dos outros nós conectados à rede. Quando um nó de origem deseja que a rede entregue uma mensagem a um determinado nó de destino, ele especifica o endereço do nó de destino. Se os nós de envio e recebimento não estiverem conectados diretamente, os switches e roteadores da rede usam esse endereço para decidir como encaminhar a mensagem para o destino. O processo de determinar sistematicamente como encaminhar mensagens para o nó de destino com base em seu endereço é chamado de *roteamento* .

Esta breve introdução ao endereçamento e roteamento pressupõe que o nó de origem deseja enviar uma mensagem para um único nó de destino (*unicast*). Embora este seja o cenário mais comum, também é possível que o nó de origem queira *transmitir* uma mensagem para todos os nós da rede. Ou, um nó de origem pode querer enviar uma mensagem para um subconjunto dos outros nós, mas não para todos eles, uma situação chamada *multicast* . Portanto, além dos endereços específicos de cada nó, outro requisito de uma rede é que ela suporte endereços multicast e broadcast.

Conclusão principal

A ideia principal desta discussão é que podemos definir uma *rede* recursivamente como composta por dois ou mais nós conectados por um link físico, ou como duas ou mais redes conectadas por um nó. Em outras palavras, uma rede pode ser construída a partir de um aninhamento de redes, onde, no nível mais baixo, a rede é implementada por algum meio físico. Entre os principais desafios para fornecer conectividade de rede estão a definição de um endereço para cada nó acessível na rede (seja lógico ou físico) e o uso desses endereços para encaminhar mensagens ao(s) nó(s) de destino apropriado(s). [Próximo]

1.2.3 Compartilhamento de recursos com boa relação custo-benefício

Como mencionado acima, este livro se concentra em redes de comutação de pacotes. Esta seção explica o principal requisito das redes de computadores — eficiência — que nos leva à comutação de pacotes como a estratégia de escolha.

Dado um conjunto de nós indiretamente conectados por um aninhamento de redes, é possível que qualquer par de hosts envie mensagens entre si através de uma sequência de enlaces e nós. É claro que queremos fazer mais do que suportar apenas um par de hosts comunicantes — queremos fornecer a todos os pares de hosts a capacidade de trocar mensagens. A questão, então, é como todos os hosts que desejam se comunicar compartilham a rede, especialmente se desejam usá-la ao mesmo tempo? E, como se esse problema não fosse suficientemente difícil, como vários hosts compartilham o mesmo *enlace* quando todos desejam usá-lo ao mesmo tempo?

Para entender como os hosts compartilham uma rede, precisamos introduzir um conceito fundamental: *multiplexação* , que significa que um recurso do sistema é compartilhado entre vários usuários. Intuitivamente, a multiplexação pode ser explicada por analogia a um sistema de computador de tempo compartilhado, onde um único processador físico é compartilhado (multiplexado) entre várias tarefas, cada uma das quais acredita ter seu próprio processador privado. Da mesma forma, os dados

enviados por vários usuários podem ser multiplexados pelos links físicos que compõem uma rede.

Para ver como isso pode funcionar, considere a rede simples ilustrada na Figura 5 , onde os três hosts no lado esquerdo da rede (remetentes S1-S3) estão enviando dados para os três hosts à direita (receptores R1-R3) compartilhando uma rede comutada que contém apenas um link físico. (Para simplificar, suponha que o host S1 esteja enviando dados para o host R1, e assim por diante.) Nessa situação, três fluxos de dados — correspondentes aos três pares de hosts — são multiplexados em um único link físico pelo switch 1 e, em seguida, *desmultiplexados* novamente em fluxos separados pelo switch 2. Observe que estamos sendo intencionalmente vagos sobre exatamente o que um "fluxo de dados" corresponde. Para os propósitos desta discussão, suponha que cada host à esquerda tenha um grande suprimento de dados que deseja enviar para sua contraparte à direita.

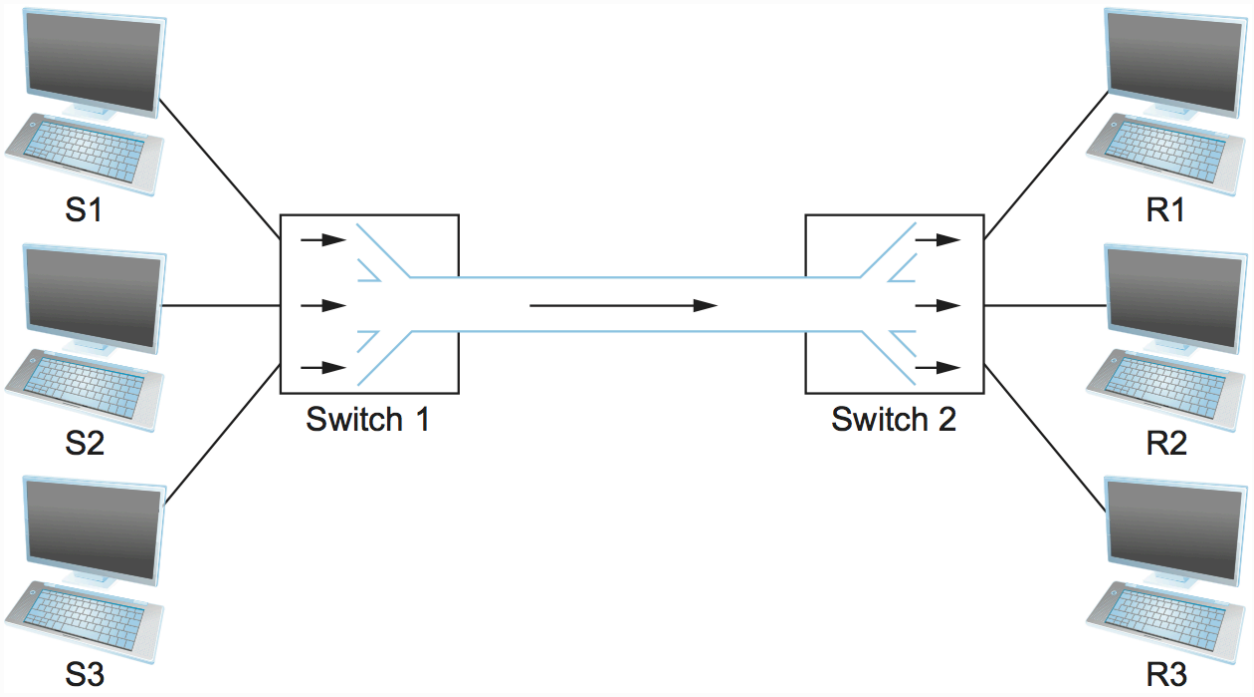

Figura 5. *Multiplexação de múltiplos fluxos lógicos em um único link físico.*

Existem vários métodos diferentes para multiplexar múltiplos fluxos em um link físico. Um método comum é *a multiplexação síncrona por divisão de tempo* (STDM). A ideia da STDM é dividir o tempo em quanta de tamanhos iguais e, em um sistema round-robin, dar a cada fluxo a chance de enviar seus dados pelo link físico. Em outras palavras, durante o quantum de tempo 1, os dados de S1 para R1 são transmitidos; durante o quantum de tempo 2, os dados de S2 para R2 são transmitidos; no quantum 3, S3 envia dados para R3. Nesse ponto, o primeiro fluxo (S1 para R1) volta a funcionar, e o processo se repete. Outro método é *a multiplexação por divisão de frequência* (FDM). A ideia da FDM é transmitir cada fluxo pelo link físico em uma frequência diferente, da mesma forma que os sinais de diferentes emissoras de TV são transmitidos em uma frequência diferente pelas ondas de rádio ou em um link de TV a cabo coaxial.

Embora simples de entender, tanto o STDM quanto o FDM são limitados de duas maneiras. Primeiro, se um dos fluxos (pares de hosts) não tiver dados para enviar, sua parcela do enlace físico — ou seja, seu quantum de tempo ou sua frequência — permanece ociosa, mesmo que um dos outros fluxos tenha dados para transmitir. Por exemplo, S3 teve que esperar sua vez atrás de S1 e S2 no parágrafo anterior, mesmo que S1 e S2 não tivessem nada para enviar. Para comunicação entre computadores, o tempo que um enlace fica ocioso pode ser muito grande — por exemplo, considere o tempo que você gasta lendo uma página da web (deixando o enlace ocioso) em comparação com o tempo que você gasta buscando a página. Segundo, tanto o STDM quanto o FDM são limitados a situações em que o número máximo de fluxos é fixo e conhecido antecipadamente. Não é prático redimensionar o quantum ou adicionar quanta adicionais no caso do STDM ou adicionar novas frequências no caso do FDM.

A forma de multiplexação que aborda essas deficiências, e da qual fazemos mais uso neste livro, é chamada de *multiplexação estatística* . Embora o nome não seja muito útil para entender o conceito, a multiplexação estatística é realmente muito simples, com duas ideias principais. Primeiro, é como STDM em que o link físico é compartilhado ao longo do tempo — primeiro os dados de um fluxo são transmitidos pelo link físico,

então os dados de outro fluxo são transmitidos, e assim por diante. Ao contrário do STDM, no entanto, os dados são transmitidos de cada fluxo sob demanda, em vez de durante um intervalo de tempo predeterminado. Assim, se apenas um fluxo tiver dados para enviar, ele consegue transmitir esses dados sem esperar que seu quantum seja revertido e, portanto, sem ter que assistir os quanta atribuídos aos outros fluxos passarem sem uso. É essa prevenção do tempo ocioso que dá à comutação de pacotes sua eficiência.

Conforme definido até agora, no entanto, a multiplexação estatística não tem nenhum mecanismo para garantir que todos os fluxos eventualmente tenham sua vez de transmitir pelo link físico. Ou seja, uma vez que um fluxo começa a enviar dados, precisamos de alguma forma de limitar a transmissão, para que os outros fluxos possam ter sua vez. Para atender a essa necessidade, a multiplexação estatística define um limite superior para o tamanho do bloco de dados que cada fluxo tem permissão para transmitir em um determinado momento. Esse bloco de dados de tamanho limitado é normalmente chamado de *pacote* , para distingui-lo da *mensagem* arbitrariamente grande que um programa de aplicação pode querer transmitir. Como uma rede comutada por pacotes limita o tamanho máximo dos pacotes, um host pode não ser capaz de enviar uma mensagem completa em um pacote. A origem pode precisar fragmentar a mensagem em vários pacotes, com o receptor remontando os pacotes de volta à mensagem original.

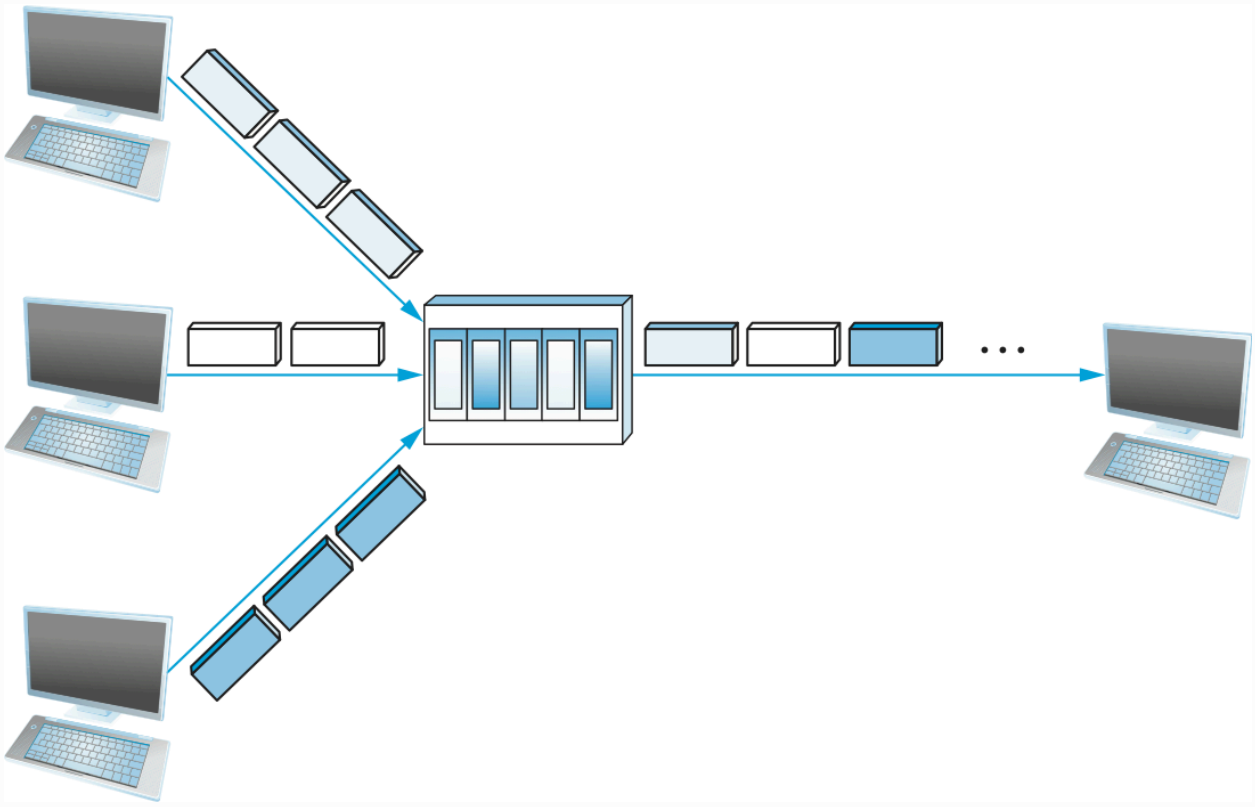

Figura 6. *Um switch multiplexando pacotes de várias fontes em um link compartilhado.*

Em outras palavras, cada fluxo envia uma sequência de pacotes pelo enlace físico, com uma decisão tomada pacote por pacote sobre qual pacote do fluxo enviar em seguida. Observe que, se apenas um fluxo tiver dados para enviar, ele poderá enviar uma sequência de pacotes consecutivamente; no entanto, se mais de um fluxo tiver dados para enviar, seus pacotes serão intercalados no enlace. A Figura 6 mostra um switch multiplexando pacotes de várias fontes em um único enlace compartilhado.

A decisão sobre qual pacote enviar em seguida em um link compartilhado pode ser feita de várias maneiras diferentes. Por exemplo, em uma rede composta por switches interconectados por links como o da Figura 5 , a decisão seria tomada pelo switch que transmite os pacotes para o link compartilhado. (Como veremos mais tarde, nem todas as redes comutadas por pacotes realmente envolvem switches, e elas podem usar outros mecanismos para determinar qual pacote vai para o próximo link.) Cada switch em uma rede comutada por pacotes toma essa decisão de forma independente, pacote

por pacote. Uma das questões que um projetista de rede enfrenta é como tomar essa decisão de forma justa. Por exemplo, um switch pode ser projetado para atender pacotes em uma base de primeiro a entrar, primeiro a sair (FIFO). Outra abordagem seria transmitir os pacotes de cada um dos diferentes fluxos que estão atualmente enviando dados através do switch de forma round-robin. Isso pode ser feito para garantir que determinados fluxos recebam uma parcela específica da largura de banda do link ou que seus pacotes nunca sejam atrasados no switch por mais de um determinado período. Às vezes, diz-se que uma rede que tenta alocar largura de banda para fluxos específicos oferece suporte *à qualidade de serviço* (QoS).

Observe também na Figura 6 que, como o switch precisa multiplexar três fluxos de pacotes de entrada em um link de saída, é possível que o switch receba pacotes mais rápido do que o link compartilhado pode acomodar. Nesse caso, o switch é forçado a armazenar esses pacotes em buffer em sua memória. Se um switch receber pacotes mais rápido do que pode enviá-los por um longo período de tempo, ele eventualmente ficará sem espaço no buffer e alguns pacotes terão que ser descartados. Quando um switch está operando nesse estado, diz-se que está *congestionado* .

Conclusão principal

Em resumo, a multiplexação estatística define uma maneira econômica para múltiplos usuários (por exemplo, fluxos de dados entre hosts) compartilharem recursos de rede (links e nós) de forma refinada. Ela define o pacote como a granularidade com que os links da rede são alocados a diferentes fluxos, com cada switch capaz de agendar o uso dos links físicos aos quais está conectado por pacote. A alocação justa da capacidade dos links para diferentes fluxos e o tratamento do congestionamento quando ele ocorre são os principais desafios da multiplexação estatística. [Próximo]

1.2.4 Suporte para Serviços Comuns

A discussão anterior se concentrou nos desafios envolvidos no fornecimento de conectividade econômica entre um grupo de hosts, mas é excessivamente simplista imaginar uma rede de computadores como a simples entrega de pacotes entre um conjunto de computadores. É mais preciso pensar em uma rede como o meio para a comunicação entre um conjunto de processos de aplicativos distribuídos por esses computadores. Em outras palavras, o próximo requisito de uma rede de computadores é que os programas aplicativos em execução nos hosts conectados à rede sejam capazes de se comunicar de forma significativa. Da perspectiva do desenvolvedor de aplicativos, a rede precisa facilitar sua vida.

Quando dois programas ou aplicativos precisam se comunicar, muitas coisas complexas precisam acontecer além do simples envio de uma mensagem de um host para outro. Uma opção seria os projetistas de aplicativos incorporarem toda essa funcionalidade complexa em cada programa aplicativo. No entanto, como muitos aplicativos precisam de serviços comuns, é muito mais lógico implementar esses serviços comuns uma vez e, em seguida, deixar que o projetista desenvolva o aplicativo usando esses serviços. O desafio para um projetista de rede é identificar o conjunto correto de serviços comuns. O objetivo é ocultar a complexidade da rede do aplicativo sem restringir excessivamente o projetista.

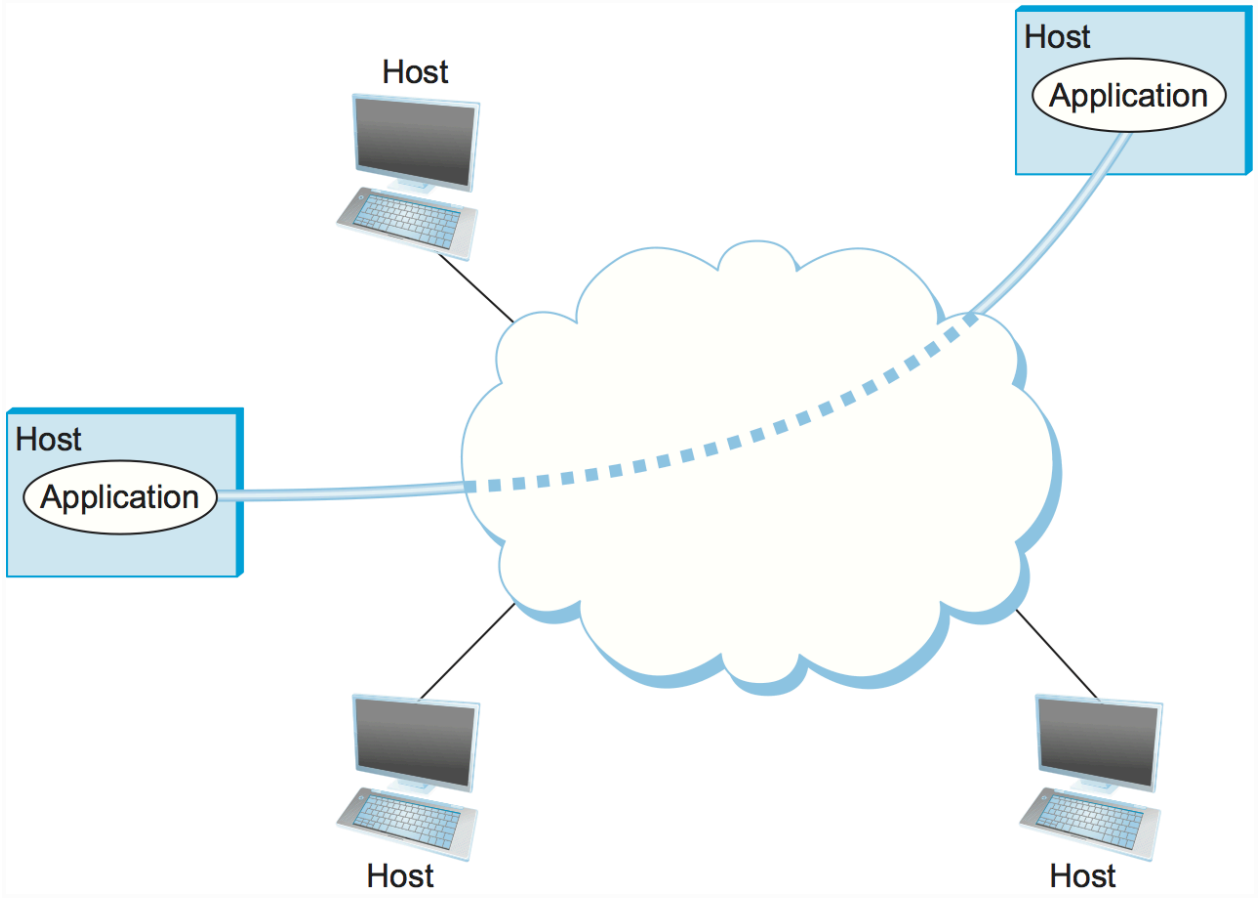

Figura 7. *Processos se comunicando por um canal abstrato.*

Intuitivamente, vemos a rede como um provedor *de canais* lógicos pelos quais processos em nível de aplicação podem se comunicar; cada canal fornece o conjunto de serviços requerido por essa aplicação. Em outras palavras, assim como usamos uma nuvem para representar abstratamente a conectividade entre um conjunto de computadores, agora pensamos em um canal como a conexão de um processo a outro. A Figura 7 mostra um par de processos em nível de aplicação se comunicando por um canal lógico que, por sua vez, é implementado sobre uma nuvem que conecta um conjunto de hosts. Podemos pensar no canal como um canal conectando duas aplicações, de modo que uma aplicação emissora pode inserir dados em uma extremidade e esperar que esses dados sejam entregues pela rede à aplicação na outra extremidade do canal.

Como qualquer abstração, canais lógicos entre processos são implementados sobre um conjunto de canais físicos entre hosts. Essa é a essência da estratificação, a base das arquiteturas de rede discutidas na próxima seção.

O desafio é reconhecer qual funcionalidade os canais devem fornecer aos programas aplicativos. Por exemplo, o aplicativo exige uma garantia de que as mensagens enviadas pelo canal sejam entregues ou é aceitável que algumas mensagens não cheguem? É necessário que as mensagens cheguem ao processo destinatário na mesma ordem em que são enviadas ou o destinatário não se importa com a ordem em que as mensagens chegam? A rede precisa garantir que terceiros não consigam espionar o canal ou a privacidade não é uma preocupação? Em geral, uma rede fornece uma variedade de tipos diferentes de canais, com cada aplicativo selecionando o tipo que melhor atende às suas necessidades. O restante desta seção ilustra o raciocínio envolvido na definição de canais úteis.

Identificar padrões comuns de comunicação

Projetar canais abstratos envolve primeiro entender as necessidades de comunicação de um conjunto representativo de aplicativos, depois extrair seus requisitos comuns de comunicação e, finalmente, incorporar a funcionalidade que atende a esses requisitos na rede.

Uma das primeiras aplicações suportadas em qualquer rede é um programa de acesso a arquivos, como o Protocolo de Transferência de Arquivos (FTP) ou o Sistema de Arquivos de Rede (NFS). Embora muitos detalhes variem — por exemplo, se arquivos inteiros são transferidos pela rede ou se apenas blocos individuais do arquivo são lidos/gravados em um determinado momento — o componente de comunicação do acesso remoto a arquivos é caracterizado por um par de processos: um que solicita a leitura ou gravação de um arquivo e um segundo processo que atende a essa solicitação. O processo que solicita acesso ao arquivo é chamado de *cliente* , e o processo que oferece suporte ao acesso ao arquivo é chamado de *servidor* .

A leitura de um arquivo envolve o cliente enviando uma pequena mensagem de solicitação a um servidor, e o servidor responde com uma mensagem grande contendo os dados do arquivo. A escrita funciona de forma oposta: o cliente envia uma mensagem grande contendo os dados a serem gravados no servidor, e o servidor responde com uma pequena mensagem confirmando que a gravação no disco ocorreu.

Uma biblioteca digital é uma aplicação mais sofisticada do que a transferência de arquivos, mas requer serviços de comunicação semelhantes. Por exemplo, a *Association for Computing Machinery* (ACM) opera uma grande biblioteca digital de literatura de ciência da computação em

`http://portal.acm.org/dl.cfm`

Esta biblioteca tem uma ampla variedade de recursos de pesquisa e navegação para ajudar os usuários a encontrar os artigos que desejam, mas, em última análise, muito do que ela faz é responder às solicitações de arquivos dos usuários, como cópias eletrônicas de artigos de periódicos.

Utilizando o acesso a arquivos, uma biblioteca digital e os dois aplicativos de vídeo descritos na introdução (videoconferência e vídeo sob demanda) como exemplo representativo, poderíamos optar por fornecer os dois tipos de canais a seguir: canais *de solicitação/resposta e canais de fluxo de mensagens* . O canal de solicitação/resposta seria usado pelos aplicativos de transferência de arquivos e biblioteca digital. Ele garantiria que cada mensagem enviada por uma parte fosse recebida pela outra e que apenas uma cópia de cada mensagem fosse entregue. O canal de solicitação/resposta também poderia proteger a privacidade e a integridade dos dados que fluem por ele, de forma que terceiros não autorizados não pudessem ler ou modificar os dados trocados entre os processos cliente e servidor.

O canal de fluxo de mensagens pode ser usado tanto por aplicativos de vídeo sob demanda quanto por videoconferência, desde que seja parametrizado para suportar tráfego unidirecional e bidirecional e para suportar diferentes propriedades de atraso. O

canal de fluxo de mensagens pode não precisar garantir que todas as mensagens sejam entregues, uma vez que um aplicativo de vídeo pode operar adequadamente mesmo que alguns quadros de vídeo não sejam recebidos. No entanto, ele precisaria garantir que as mensagens entregues cheguem na mesma ordem em que foram enviadas, para evitar a exibição de quadros fora de sequência. Assim como o canal de solicitação/resposta, o canal de fluxo de mensagens pode querer garantir a privacidade e a integridade dos dados de vídeo. Por fim, o canal de fluxo de mensagens pode precisar suportar multicast, para que várias partes possam participar da teleconferência ou assistir ao vídeo.

Embora seja comum que um projetista de redes busque o menor número possível de tipos de canais abstratos que possam atender ao maior número de aplicações, existe o perigo de se contentar com poucas abstrações de canais. Em termos simples, se você tem um martelo, tudo parece um prego. Por exemplo, se você só tem canais de fluxo de mensagens e de solicitação/resposta, é tentador usá-los na próxima aplicação que surgir, mesmo que nenhum dos tipos forneça exatamente a semântica necessária para a aplicação. Assim, os projetistas de redes provavelmente continuarão inventando novos tipos de canais — e adicionando opções aos canais existentes — enquanto os programadores de aplicações estiverem inventando novas aplicações.

Observe também que, independentemente de *qual* funcionalidade um determinado canal fornece, há a questão de *onde* essa funcionalidade é implementada. Em muitos casos, é mais fácil visualizar a conectividade host-a-host da rede subjacente como simplesmente fornecendo um *bit pipe* , com qualquer semântica de comunicação de alto nível fornecida nos hosts finais. A vantagem dessa abordagem é que ela mantém os switches no meio da rede o mais simples possível — eles simplesmente encaminham pacotes — mas exige que os hosts finais assumam grande parte do fardo de suportar canais semanticamente ricos entre processos. A alternativa é empurrar funcionalidade adicional para os switches, permitindo assim que os hosts finais sejam dispositivos "burros" (por exemplo, aparelhos telefônicos). Veremos essa questão de

como vários serviços de rede são particionados entre os switches de pacotes e os hosts finais (dispositivos) como um problema recorrente no projeto de rede.

Entrega confiável de mensagens

Como sugerido pelos exemplos que acabamos de considerar, a entrega confiável de mensagens é uma das funções mais importantes que uma rede pode fornecer. No entanto, é difícil determinar como fornecer essa confiabilidade sem primeiro entender como as redes podem falhar. A primeira coisa a reconhecer é que redes de computadores não existem em um mundo perfeito. Máquinas travam e depois são reinicializadas, fibras são cortadas, interferências elétricas corrompem bits nos dados transmitidos, switches ficam sem espaço em buffer e, como se esses tipos de problemas físicos não fossem suficientes para se preocupar, o software que gerencia o hardware pode conter bugs e, às vezes, encaminhar pacotes para o esquecimento. Portanto, um requisito importante de uma rede é se recuperar de certos tipos de falhas, para que os programas aplicativos não precisem lidar com elas ou mesmo estar cientes delas.

Existem três classes gerais de falhas com as quais os projetistas de rede precisam se preocupar. Primeiro, conforme um pacote é transmitido por um link físico, *erros de bits* podem ser introduzidos nos dados; ou seja, um 1 é transformado em 0 ou *vice-versa* . Às vezes, bits individuais são corrompidos, mas na maioria das vezes ocorre um *erro de rajada* — vários bits consecutivos são corrompidos. Erros de bits normalmente ocorrem porque forças externas, como raios, picos de energia e fornos de micro-ondas, interferem na transmissão de dados. A boa notícia é que esses erros de bits são bastante raros, afetando, em média, apenas um em cada 106 a 107 bits em um cabo típico baseado em cobre e um em cada 1012 a 1014 bits em uma fibra óptica típica. Como veremos, existem técnicas que detectam esses erros de bits com alta probabilidade. Uma vez detectados, às vezes é possível corrigir esses erros — se soubermos qual bit ou bits estão corrompidos, podemos simplesmente invertê-los — enquanto em outros casos o dano é tão grave que é necessário descartar o pacote inteiro. Nesse caso, pode-se esperar que o remetente retransmita o pacote.

A segunda classe de falha ocorre no nível do pacote, e não no nível do bit; ou seja, um pacote completo é perdido pela rede. Um motivo para isso é que o pacote contém um erro de bit incorrigível e, portanto, precisa ser descartado. Um motivo mais provável, no entanto, é que um dos nós que precisa manipular o pacote — por exemplo, um switch que o encaminha de um enlace para outro — esteja tão sobrecarregado que não tenha onde armazenar o pacote e, portanto, seja forçado a descartá-lo. Este é o problema de congestionamento que acabamos de discutir. Menos comumente, o software em execução em um dos nós que manipula o pacote comete um erro. Por exemplo, ele pode encaminhar incorretamente um pacote pelo enlace errado, de modo que o pacote nunca chegue ao destino final. Como veremos, uma das principais dificuldades em lidar com pacotes perdidos é distinguir entre um pacote que está realmente perdido e um que apenas chega atrasado ao destino.

A terceira classe de falha ocorre no nível de nó e enlace; ou seja, um enlace físico é cortado ou o computador ao qual ele está conectado trava. Isso pode ser causado por um software que trava, uma falha de energia ou um operador de retroescavadeira imprudente. Falhas devido à configuração incorreta de um dispositivo de rede também são comuns. Embora qualquer uma dessas falhas possa eventualmente ser corrigida, elas podem ter um efeito drástico na rede por um longo período de tempo. No entanto, elas não precisam desabilitar totalmente a rede. Em uma rede comutada por pacotes, por exemplo, às vezes é possível contornar um nó ou enlace com falha. Uma das dificuldades em lidar com essa terceira classe de falha é distinguir entre um computador com falha e um que está apenas lento ou, no caso de um enlace, entre um que foi cortado e um que está muito instável e, portanto, introduzindo um alto número de erros de bits.

Conclusão principal

A ideia-chave desta discussão é que definir canais úteis envolve tanto a compreensão dos requisitos das aplicações quanto o reconhecimento das limitações da tecnologia subjacente. O desafio é preencher a lacuna entre o que a aplicação espera e o que a

tecnologia subjacente pode oferecer. Isso às vezes é chamado de *lacuna semântica.* [Próximo]

1.2.5 Gerenciabilidade

Um requisito final, que parece ser negligenciado ou deixado para o final com muita frequência (como acontece aqui), é que as redes precisam ser gerenciadas. Gerenciar uma rede inclui atualizar equipamentos à medida que a rede cresce para transportar mais tráfego ou alcançar mais usuários, solucionar problemas de rede quando algo dá errado ou o desempenho não é o desejado e adicionar novos recursos para dar suporte a novas aplicações. O gerenciamento de redes historicamente tem sido um aspecto da rede que exige muita mão de obra e, embora seja improvável que consigamos tirar as pessoas completamente do circuito, ele está sendo cada vez mais abordado por meio de projetos de automação e autorrecuperação.

Esse requisito está parcialmente relacionado à questão da escalabilidade discutida acima — à medida que a internet se expandiu para suportar bilhões de usuários e pelo menos centenas de milhões de hosts, os desafios de manter tudo funcionando corretamente e configurar corretamente novos dispositivos à medida que são adicionados tornaram-se cada vez mais complexos. Configurar um único roteador em uma rede costuma ser uma tarefa para um especialista treinado; configurar milhares de roteadores e descobrir por que uma rede desse tamanho não está se comportando como esperado pode se tornar uma tarefa que está além da capacidade de qualquer ser humano. É por isso que a automação está se tornando tão importante.

Uma maneira de tornar uma rede mais fácil de gerenciar é evitar mudanças. Uma vez que a rede esteja funcionando, simplesmente *não toque nela!* Essa mentalidade expõe a tensão fundamental entre *estabilidade* e *velocidade dos recursos* : a taxa na qual novos recursos são introduzidos na rede. Favorecer a estabilidade é a abordagem que o setor de telecomunicações (sem mencionar administradores de sistemas universitários e departamentos de TI corporativos) adotou por muitos anos, tornando-o um dos setores mais lentos e avessos a riscos que você encontrará em qualquer lugar.

Mas a recente explosão da nuvem mudou essa dinâmica, tornando necessário equilibrar melhor a estabilidade e a velocidade dos recursos. O impacto da nuvem na rede é um tópico que surge repetidamente ao longo do livro e ao qual damos atenção especial na seção *Perspectivas* ao final de cada capítulo. Por enquanto, basta dizer que gerenciar uma rede em rápida evolução é, sem dúvida, *o* desafio central nas redes hoje.

1.3 Arquitetura

A seção anterior estabeleceu um conjunto bastante substancial de requisitos para o projeto de redes — uma rede de computadores deve fornecer conectividade geral, econômica, justa e robusta entre um grande número de computadores. Como se isso não bastasse, as redes não permanecem fixas em um único ponto no tempo, mas devem evoluir para acomodar mudanças tanto nas tecnologias subjacentes nas quais se baseiam quanto nas mudanças nas demandas impostas a elas por programas aplicativos. Além disso, as redes devem ser gerenciáveis por humanos com diferentes níveis de habilidade. Projetar uma rede para atender a esses requisitos não é uma tarefa fácil.

Para ajudar a lidar com essa complexidade, os projetistas de redes desenvolveram projetos gerais — geralmente chamados de *arquiteturas de rede* — que orientam o projeto e a implementação de redes. Esta seção define com mais detalhes o que queremos dizer com arquitetura de rede, apresentando as ideias centrais comuns a todas as arquiteturas de rede. Também apresenta duas das arquiteturas mais amplamente referenciadas — a arquitetura OSI (ou de 7 camadas) e a arquitetura da internet.

1.3.1 Camadas e Protocolos

Abstração — a ocultação de detalhes de implementação por trás de uma interface bem definida — é a ferramenta fundamental usada por projetistas de sistemas para gerenciar a complexidade. A ideia de uma abstração é definir um modelo que possa capturar algum aspecto importante do sistema, encapsular esse modelo em um objeto que forneça uma interface que possa ser manipulada por outros componentes do sistema e ocultar os detalhes de como o objeto é implementado dos usuários. O desafio é identificar abstrações que forneçam simultaneamente um serviço que se mostre útil em um grande número de situações e que possa ser implementado eficientemente no sistema subjacente. É exatamente isso que estávamos fazendo quando introduzimos a ideia de um canal na seção anterior: estávamos fornecendo uma abstração para aplicações que oculta a complexidade da rede dos desenvolvedores de aplicações.

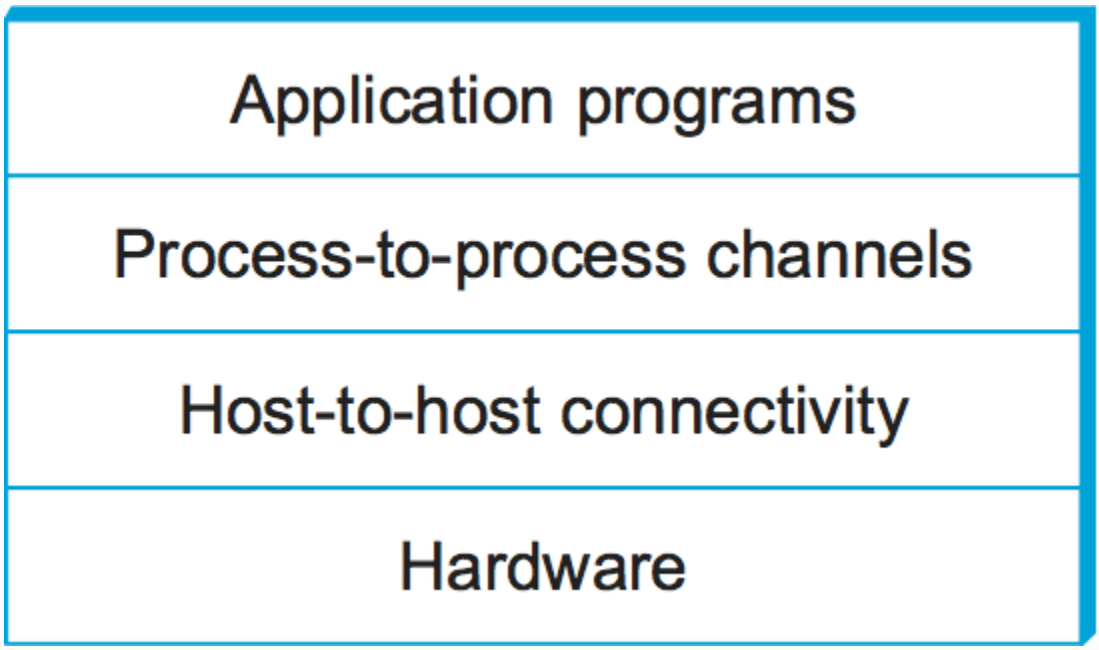

Figura 8. *Exemplo de um sistema de rede em camadas.*

Abstrações naturalmente levam à estratificação, especialmente em sistemas de rede. A ideia geral é que você comece com os serviços oferecidos pelo hardware subjacente e, em seguida, adicione uma sequência de camadas, cada uma fornecendo um nível de serviço mais alto (mais abstrato). Os serviços fornecidos nas camadas superiores são

implementados em termos dos serviços fornecidos pelas camadas inferiores. Com base na discussão de requisitos apresentada na seção anterior, por exemplo, podemos imaginar uma rede simples como tendo duas camadas de abstração intercaladas entre o programa aplicativo e o hardware subjacente, conforme ilustrado na Figura 8. A camada imediatamente acima do hardware, neste caso, pode fornecer conectividade host a host, abstraindo o fato de que pode haver uma topologia de rede arbitrariamente complexa entre quaisquer dois hosts. A próxima camada se baseia no serviço de comunicação host a host disponível e fornece suporte para canais processo a processo, abstraindo o fato de que a rede ocasionalmente perde mensagens, por exemplo.

A estrutura em camadas oferece dois recursos úteis. Primeiro, ela decompõe o problema de construir uma rede em componentes mais gerenciáveis. Em vez de implementar um software monolítico que faz tudo o que você sempre quis, você pode implementar várias camadas, cada uma resolvendo uma parte do problema. Segundo, ela oferece um design mais modular. Se você decidir adicionar algum novo serviço, talvez precise modificar apenas a funcionalidade de uma camada, reutilizando as funções fornecidas em todas as outras camadas.

No entanto, pensar em um sistema como uma sequência linear de camadas é uma simplificação exagerada. Muitas vezes, há múltiplas abstrações fornecidas em qualquer nível do sistema, cada uma fornecendo um serviço diferente para as camadas superiores, mas se baseando nas mesmas abstrações de nível inferior. Para entender isso, considere os dois tipos de canais discutidos na seção anterior. Um fornece um serviço de solicitação/resposta e o outro suporta um serviço de fluxo de mensagens. Esses dois canais podem ser ofertas alternativas em algum nível de um sistema de rede multinível, como ilustrado na Figura 9 .

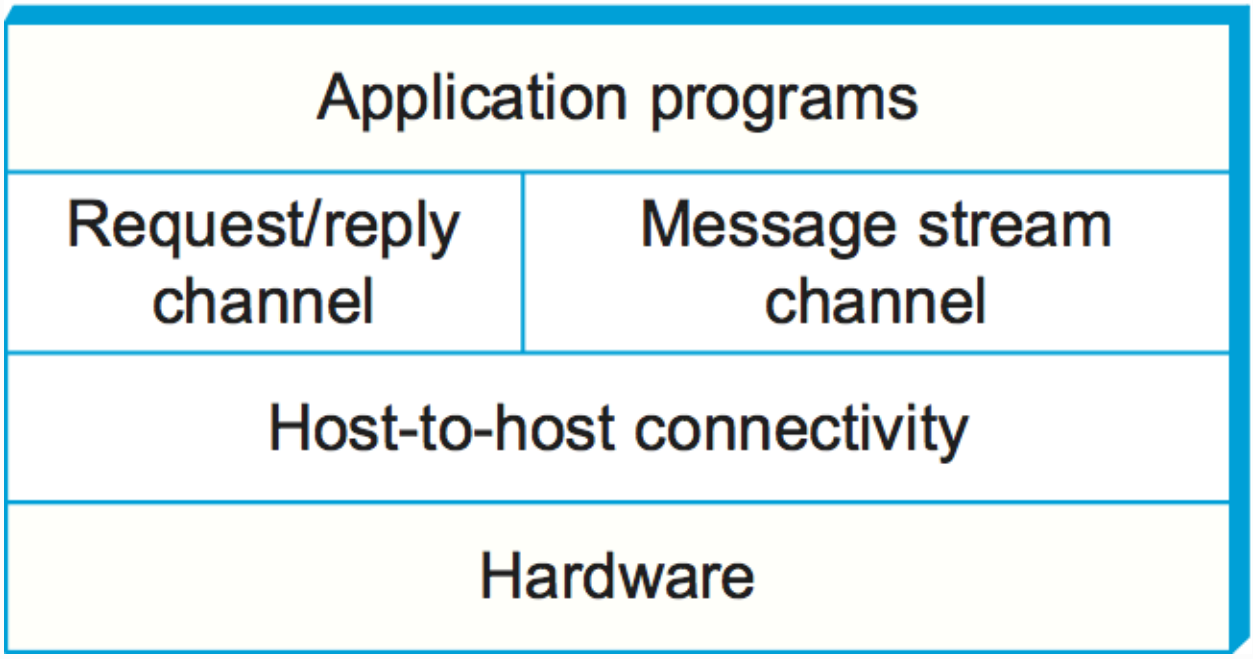

Figura 9. *Sistema em camadas com abstrações alternativas disponíveis em uma determinada camada.*

Usando esta discussão sobre camadas como base, estamos prontos para discutir a arquitetura de uma rede com mais precisão. Para começar, os objetos abstratos que compõem as camadas de um sistema de rede são chamados de *protocolos* . Ou seja, um protocolo fornece um serviço de comunicação que objetos de nível superior (como processos de aplicação, ou talvez protocolos de nível superior) usam para trocar mensagens. Por exemplo, poderíamos imaginar uma rede que suporta um protocolo de solicitação/resposta e um protocolo de fluxo de mensagens, correspondendo aos canais de solicitação/resposta e fluxo de mensagens discutidos acima.

Cada protocolo define duas interfaces diferentes. Primeiro, ele define uma *interface de serviço* para os outros objetos no mesmo computador que desejam usar seus serviços de comunicação. Essa interface de serviço define as operações que objetos locais podem realizar no protocolo. Por exemplo, um protocolo de solicitação/resposta suportaria operações pelas quais um aplicativo pode enviar e receber mensagens. Uma implementação do protocolo HTTP poderia suportar uma operação para buscar uma página de hipertexto em um servidor remoto. Um aplicativo, como um navegador web,

invocaria essa operação sempre que o navegador precisasse obter uma nova página (por exemplo, quando o usuário clicasse em um link na página exibida no momento).

Em segundo lugar, um protocolo define uma *interface ponto a ponto* para sua contraparte (ponto a ponto) em outra máquina. Essa segunda interface define a forma e o significado das mensagens trocadas entre os pontos do protocolo para implementar o serviço de comunicação. Isso determinaria a maneira como um protocolo de solicitação/resposta em uma máquina se comunica com seu ponto a ponto em outra máquina. No caso do HTTP, por exemplo, a especificação do protocolo define em detalhes como um comando *GET* é formatado, quais argumentos podem ser usados com o comando e como um servidor web deve responder ao receber tal comando.

Em resumo, um protocolo define um serviço de comunicação que exporta localmente (a interface de serviço), juntamente com um conjunto de regras que regem as mensagens que o protocolo troca com seus pares para implementar esse serviço (a interface de pares). Essa situação é ilustrada na Figura 10 .

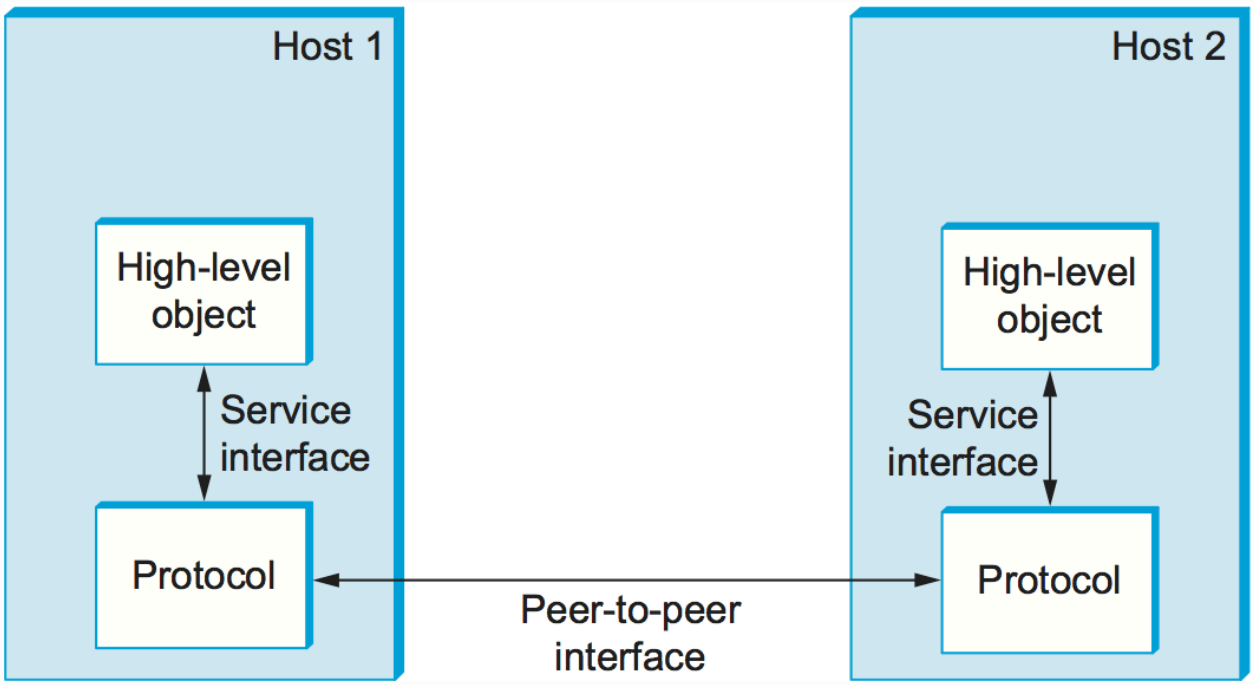

Figura 10. *Interfaces de serviço e interfaces de pares.*

Exceto no nível de hardware, onde os pares se comunicam diretamente entre si por meio de um meio físico, a comunicação ponto a ponto é indireta — cada protocolo se comunica com seu par passando mensagens para algum protocolo de nível inferior, que por sua vez entrega a mensagem ao *seu* par. Além disso, há potencialmente mais de um protocolo em qualquer nível, cada um fornecendo um serviço de comunicação diferente. Portanto, representamos o conjunto de protocolos que compõem um sistema de rede com um *grafo de protocolos* . Os nós do grafo correspondem a protocolos e as arestas representam uma relação *de dependência* . Por exemplo, a Figura 11 ilustra um grafo de protocolo para o sistema em camadas hipotético que estamos discutindo — os protocolos RRP (Request/Reply Protocol) e MSP (Message Stream Protocol) implementam dois tipos diferentes de canais processo a processo, e ambos dependem do Host-to-Host Protocol (HHP), que fornece um serviço de conectividade host a host.

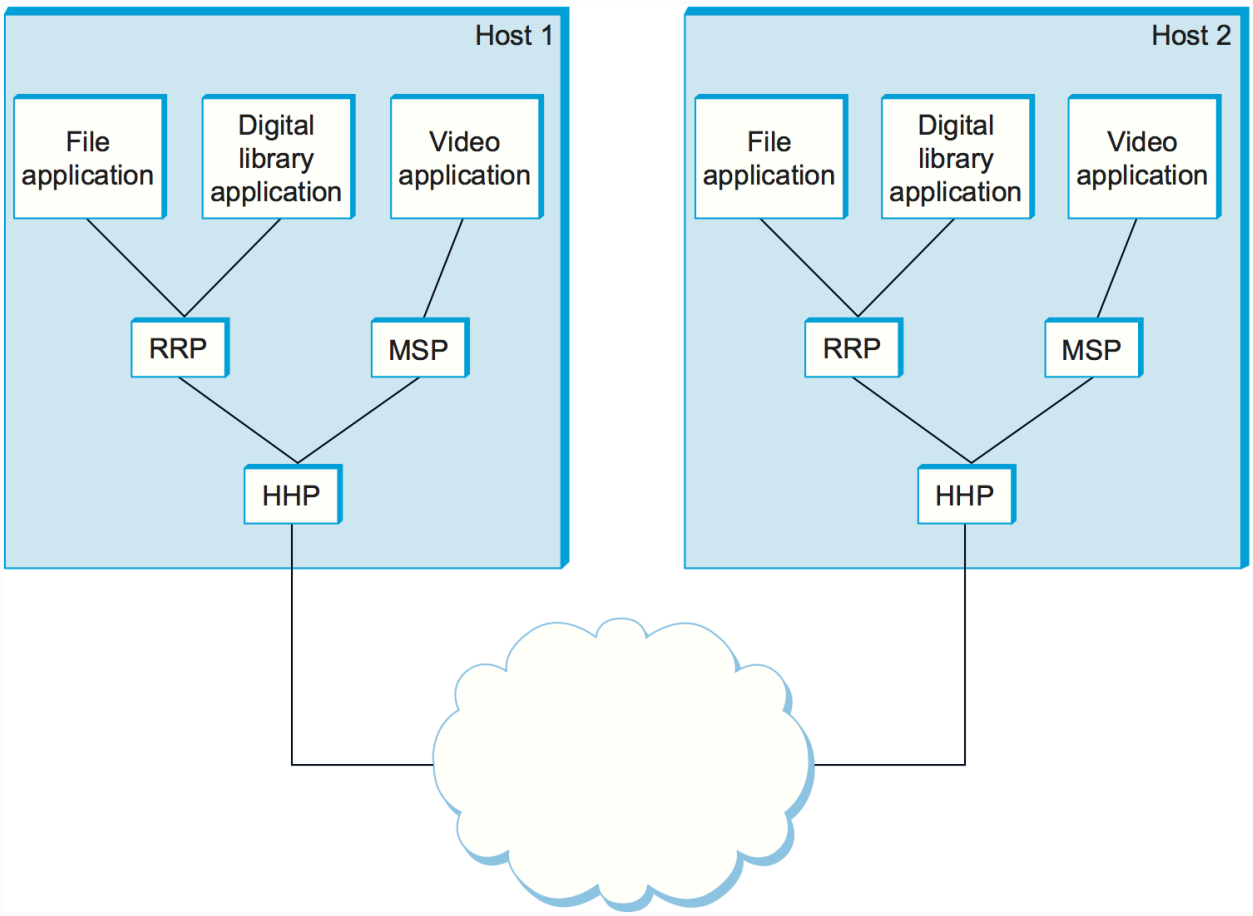

Figura 11. *Exemplo de um gráfico de protocolo.*

Neste exemplo, suponha que o programa de acesso a arquivos no host 1 queira enviar uma mensagem ao seu par no host 2 usando o serviço de comunicação oferecido pelo RRP. Nesse caso, o aplicativo de arquivos solicita ao RRP que envie a mensagem em seu nome. Para se comunicar com seu par, o RRP invoca os serviços do HHP, que, por sua vez, transmite a mensagem ao seu par na outra máquina. Assim que a mensagem chega à instância do HHP no host 2, o HHP a repassa ao RRP, que, por sua vez, a entrega ao aplicativo de arquivos. Nesse caso específico, diz-se que o aplicativo utiliza os serviços da *pilha de protocolos* RRP/HHP.

Observe que o termo *protocolo* é usado de duas maneiras diferentes. Às vezes, refere-se às interfaces abstratas — ou seja, as operações definidas pela interface de serviço e a forma e o significado das mensagens trocadas entre os pares — e, às vezes, refere-se ao módulo que realmente implementa essas duas interfaces. Para distinguir entre as interfaces e o módulo que as implementa, geralmente nos referimos ao primeiro como uma *especificação de protocolo* . As especificações são geralmente expressas usando uma combinação de prosa, pseudocódigo, diagramas de transição de estado, imagens de formatos de pacotes e outras notações abstratas. Deve ser o caso de que um determinado protocolo pode ser implementado de maneiras diferentes por diferentes programadores, desde que cada um adira à especificação. O desafio é garantir que duas implementações diferentes da mesma especificação possam trocar mensagens com sucesso. Dois ou mais módulos de protocolo que implementam com precisão uma especificação de protocolo são considerados *interoperantes* entre si.

Podemos imaginar muitos protocolos e grafos de protocolo diferentes que satisfazem os requisitos de comunicação de uma coleção de aplicações. Felizmente, existem órgãos de padronização, como a Internet Engineering Task Force (IETF) e a International Standards Organization (ISO), que estabelecem políticas para um grafo de protocolo específico. Chamamos o conjunto de regras que regem a forma e o conteúdo de um grafo de protocolo de *arquitetura de rede* . Embora esteja além do escopo deste livro, os órgãos de padronização estabeleceram procedimentos bem definidos para

introduzir, validar e, finalmente, aprovar protocolos em suas respectivas arquiteturas. Descreveremos brevemente as arquiteturas definidas pela IETF e pela ISO, mas primeiro há duas coisas adicionais que precisamos explicar sobre a mecânica da sobreposição de protocolos.

1.3.2 Encapsulamento

Considere o que acontece na Figura 11 quando um dos programas aplicativos envia uma mensagem ao seu par, passando-a para o RRP. Da perspectiva do RRP, a mensagem que ele recebe do aplicativo é uma sequência de bytes não interpretada. O RRP não se importa que esses bytes representem um conjunto de inteiros, uma mensagem de e-mail, uma imagem digital ou qualquer outra coisa; ele é simplesmente encarregado de enviá-los ao seu par. No entanto, o RRP deve comunicar informações de controle ao seu par, instruindo-o sobre como lidar com a mensagem quando ela for recebida. O RRP faz isso anexando um *cabeçalho* à mensagem. De modo geral, um cabeçalho é uma pequena estrutura de dados — de alguns bytes a algumas dezenas de bytes — que é usada entre pares para se comunicarem entre si. Como o nome sugere, os cabeçalhos geralmente são anexados ao início de uma mensagem. Em alguns casos, no entanto, essas informações de controle ponto a ponto são enviadas ao final da mensagem, sendo, nesse caso, chamadas de *trailer* . O formato exato do cabeçalho anexado pelo RRP é definido por sua especificação de protocolo. O restante da mensagem — ou seja, os dados transmitidos em nome do aplicativo — é chamado de *corpo* da mensagem ou *carga útil* . Dizemos que os dados do aplicativo são *encapsulados* na nova mensagem criada pelo RRP.

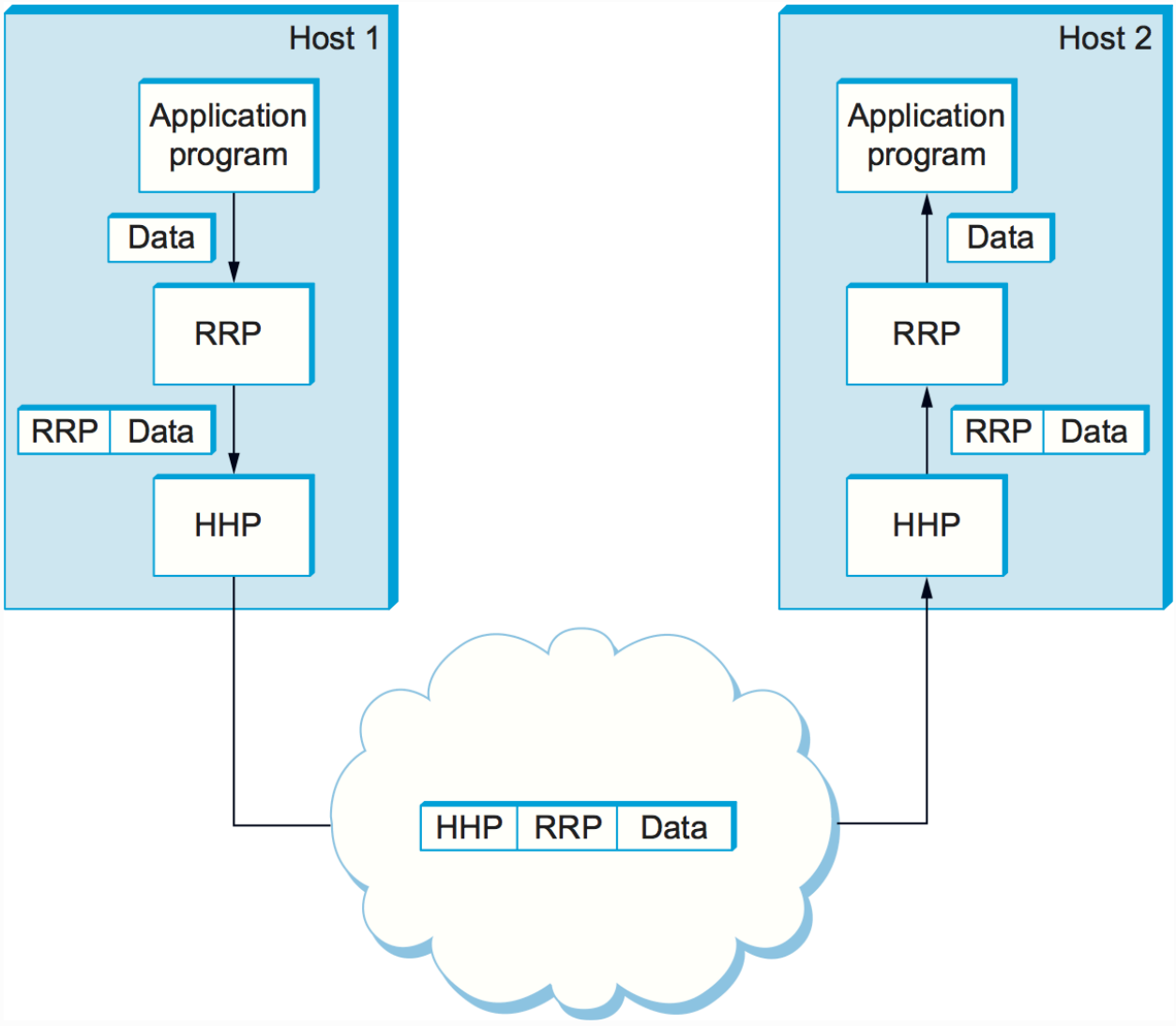

Figura 12. *Mensagens de alto nível são encapsuladas dentro de mensagens de baixo nível.*

Esse processo de encapsulamento é então repetido em cada nível do grafo do protocolo; por exemplo, o HHP encapsula a mensagem do RRP anexando um cabeçalho próprio. Se agora assumirmos que o HHP envia a mensagem ao seu par através de alguma rede, então, quando a mensagem chega ao host de destino, ela é processada na ordem oposta: o HHP primeiro interpreta o cabeçalho HHP no início da mensagem (ou seja, toma qualquer ação apropriada dado o conteúdo do cabeçalho) e passa o corpo da mensagem (mas não o cabeçalho HHP) para o RRP, que toma qualquer ação indicada pelo cabeçalho RRP que seu par anexou e passa o corpo da

mensagem (mas não o cabeçalho RRP) para o programa aplicativo. A mensagem passada do RRP para o aplicativo no host 2 é exatamente a mesma mensagem que o aplicativo passou para o RRP no host 1; o aplicativo não vê nenhum dos cabeçalhos que foram anexados a ele para implementar os serviços de comunicação de nível inferior. Todo esse processo é ilustrado na Figura 12 . Observe que, neste exemplo, os nós na rede (por exemplo, switches e roteadores) podem inspecionar o cabeçalho HHP no início da mensagem.

Observe que, quando dizemos que um protocolo de baixo nível não interpreta a mensagem que lhe é transmitida por algum protocolo de alto nível, queremos dizer que ele não sabe como extrair qualquer significado dos dados contidos na mensagem. Às vezes, porém, o protocolo de baixo nível aplica alguma transformação simples aos dados que lhe são transmitidos, como compactá-los ou criptografá-los. Nesse caso, o protocolo está transformando todo o corpo da mensagem, incluindo os dados do aplicativo original e todos os cabeçalhos anexados a esses dados por protocolos de nível superior.

1.3.3 Multiplexação e desmultiplexação

Lembre-se de que uma ideia fundamental da comutação de pacotes é multiplexar múltiplos fluxos de dados em um único link físico. Essa mesma ideia se aplica a todo o grafo do protocolo, não apenas aos nós de comutação. Na Figura 11 , por exemplo, podemos imaginar o RRP como a implementação de um canal de comunicação lógico, com mensagens de duas aplicações diferentes multiplexadas por esse canal no host de origem e, em seguida, desmultiplexadas de volta para a aplicação apropriada no host de destino.

Na prática, isso significa simplesmente que o cabeçalho que o RRP anexa às suas mensagens contém um identificador que registra o aplicativo ao qual a mensagem pertence. Chamamos esse identificador *de chave de demultiplexação* do RRP , ou *chave de demux,* para abreviar. No host de origem, o RRP inclui a chave de demux apropriada em seu cabeçalho. Quando a mensagem é entregue ao RRP no host de

destino, ele remove seu cabeçalho, examina a chave de demux e desmultiplexa a mensagem para o aplicativo correto.

O RRP não é o único a oferecer suporte à multiplexação; quase todos os protocolos implementam esse mecanismo. Por exemplo, o HHP possui sua própria chave de desmultiplexação para determinar quais mensagens passar para o RRP e quais passar para o MSP. No entanto, não há um acordo uniforme entre os protocolos — mesmo aqueles dentro de uma única arquitetura de rede — sobre o que exatamente constitui uma chave de desmultiplexação. Alguns protocolos usam um campo de 8 bits (o que significa que podem suportar apenas 256 protocolos de alto nível), e outros usam campos de 16 ou 32 bits. Além disso, alguns protocolos têm um único campo de desmultiplexação em seu cabeçalho, enquanto outros têm um par de campos de desmultiplexação. No primeiro caso, a mesma chave de desmultiplexação é usada em ambos os lados da comunicação, enquanto no segundo caso, cada lado usa uma chave diferente para identificar o protocolo de alto nível (ou programa aplicativo) ao qual a mensagem deve ser entregue.

1.3.4 Modelo OSI

A ISO foi uma das primeiras organizações a definir formalmente uma maneira comum de conectar computadores. Sua arquitetura, chamada de arquitetura *Open Systems Interconnection* (OSI) e ilustrada na Figura 13 , define um particionamento da funcionalidade de rede em sete camadas, onde um ou mais protocolos implementam a funcionalidade atribuída a uma determinada camada. Nesse sentido, o esquema dado na Figura 13 não é um grafo de protocolo, *por si só* , mas sim um *modelo de referência* para um grafo de protocolo. Ele é frequentemente chamado de modelo de 7 camadas. Embora não haja nenhuma rede baseada em OSI em execução hoje, a terminologia que ela definiu ainda é amplamente usada, então ainda vale a pena uma olhada superficial.

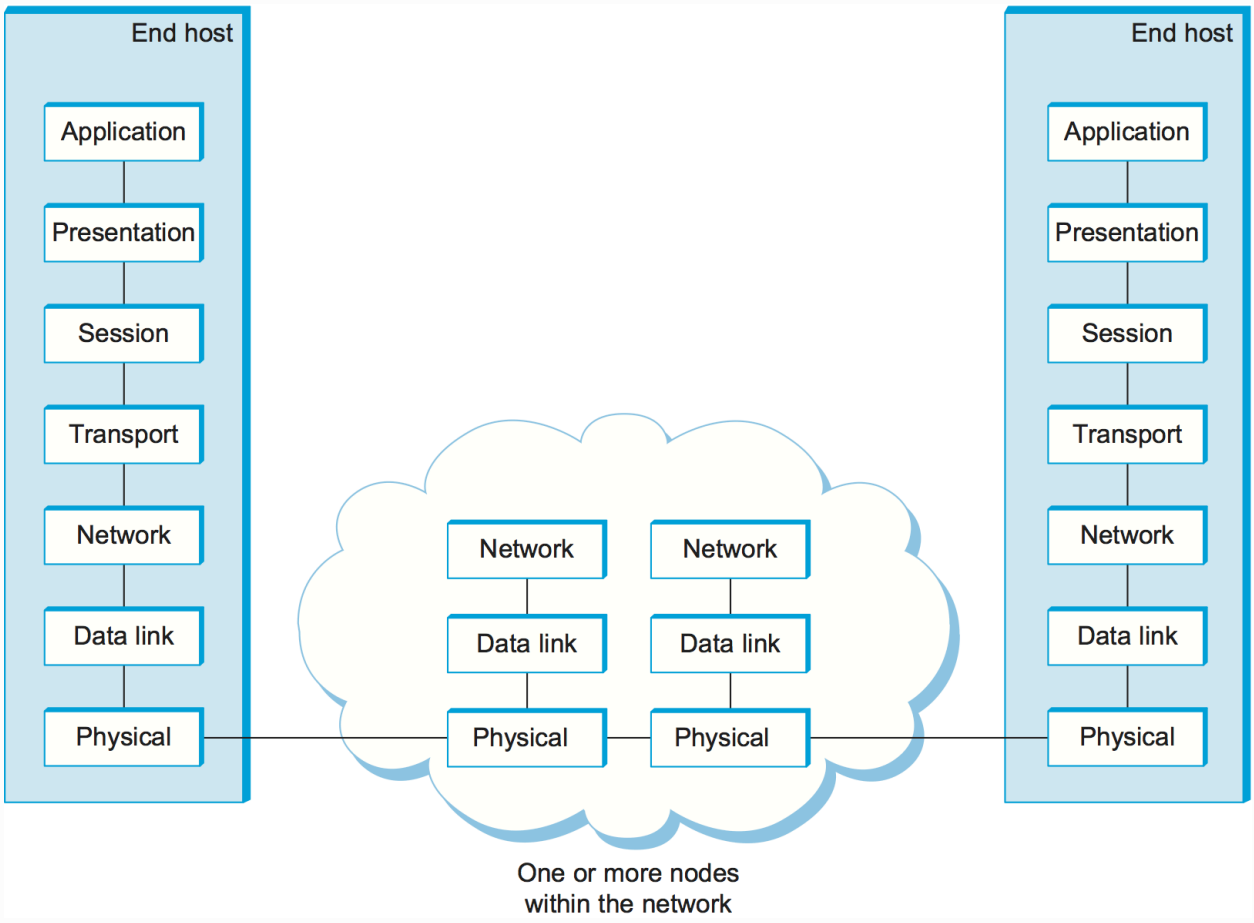

Figura 13. *O modelo OSI de 7 camadas.*

Começando de baixo para cima, a camada *física* lida com a transmissão de bits brutos por um link de comunicação. A camada *de link de dados então coleta um fluxo de bits em um agregado maior chamado quadro* . Adaptadores de rede, juntamente com drivers de dispositivo em execução no sistema operacional do nó, normalmente implementam o nível de link de dados. Isso significa que quadros, não bits brutos, são realmente entregues aos hosts. A camada *de rede* lida com o roteamento entre nós dentro de uma rede comutada por pacotes. Nessa camada, a unidade de dados trocada entre nós é normalmente chamada de *pacote* em vez de quadro, embora sejam fundamentalmente a mesma coisa. As três camadas inferiores são implementadas em todos os nós da rede, incluindo switches dentro da rede e hosts conectados ao exterior da rede. A camada *de transporte* então implementa o que até este ponto temos chamado de *canal processo a processo* . Aqui, a unidade de dados trocada é

comumente chamada de *mensagem* em vez de pacote ou quadro. A camada de transporte e as camadas superiores normalmente são executadas apenas nos hosts finais e não nos switches ou roteadores intermediários.

Pulando para a camada superior (sétima) e descendo de volta, encontramos a camada *de aplicação* . Os protocolos da camada de aplicação incluem coisas como o Protocolo de Transferência de Hipertexto (HTTP), que é a base da World Wide Web e permite que navegadores da web solicitem páginas de servidores web. Abaixo dela, a camada *de apresentação* se preocupa com o formato dos dados trocados entre pares — por exemplo, se um inteiro tem 16, 32 ou 64 bits, se o byte mais significativo é transmitido primeiro ou por último, ou como um fluxo de vídeo é formatado. Por fim, a camada *de sessão* fornece um espaço de nomes que é usado para unir os fluxos de transporte potencialmente diferentes que fazem parte de uma única aplicação. Por exemplo, ela pode gerenciar um fluxo de áudio e um fluxo de vídeo que estão sendo combinados em uma aplicação de teleconferência.

1.3.5 Arquitetura da Internet

A arquitetura da Internet, às vezes também chamada de arquitetura TCP/IP, em homenagem aos seus dois protocolos principais, é ilustrada na Figura 14. Uma representação alternativa é apresentada na Figura 15. A arquitetura da Internet evoluiu a partir de experiências com uma rede comutada por pacotes anterior, chamada ARPANET. Tanto a Internet quanto a ARPANET foram financiadas pela Agência de Projetos de Pesquisa Avançada (ARPA), uma das agências de financiamento de pesquisa e desenvolvimento do Departamento de Defesa dos EUA. A Internet e a ARPANET já existiam antes da arquitetura OSI, e a experiência adquirida com sua construção teve grande influência no modelo de referência OSI.

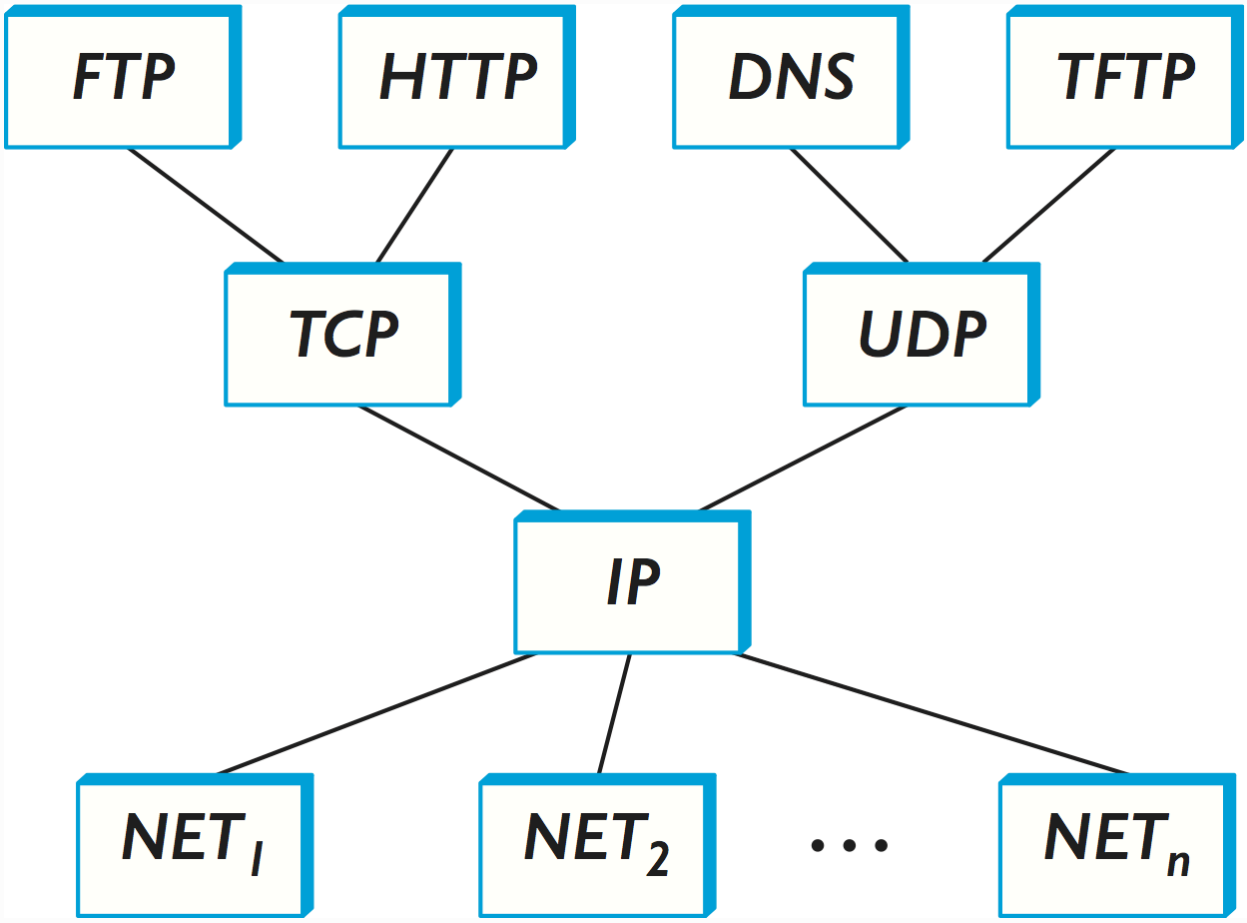

Figura 14. *Gráfico do protocolo de internet.*

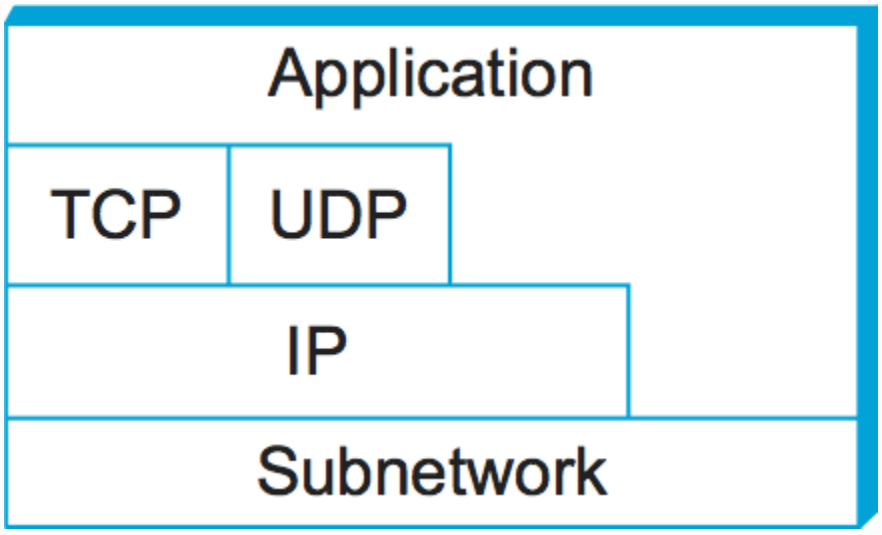

Figura 15. *Visão alternativa da arquitetura da Internet. A camada de "sub-rede" era historicamente chamada de camada de "rede" e agora é frequentemente chamada de "Camada 2" (influenciada pelo modelo OSI).*

Embora o modelo OSI de 7 camadas possa, com alguma imaginação, ser aplicado à Internet, uma pilha mais simples é frequentemente usada. No nível mais baixo, há uma grande variedade de protocolos de rede, denotados NET $_1$, NET $_2$ e assim por diante. Na prática, esses protocolos são implementados por uma combinação de hardware (por exemplo, um adaptador de rede) e software (por exemplo, um driver de dispositivo de rede). Por exemplo, você pode encontrar protocolos Ethernet ou sem fio (como os padrões Wi-Fi 802.11) nessa camada. (Esses protocolos, por sua vez, podem envolver várias subcamadas, mas a arquitetura da Internet não presume nada sobre elas.) A próxima camada consiste em um único protocolo — o *Protocolo de Internet* (IP). Este é o protocolo que suporta a interconexão de várias tecnologias de rede em uma única rede lógica. A camada sobre o IP contém dois protocolos principais — o *Protocolo de Controle de Transmissão* (TCP) e o *Protocolo de Datagrama do Usuário* (UDP). TCP e UDP fornecem canais lógicos alternativos para programas de aplicação: o TCP fornece um canal confiável de fluxo de bytes, e o UDP fornece um canal não confiável de entrega de datagramas (*datagrama* pode ser considerado sinônimo de mensagem). Na linguagem da internet, TCP e UDP são às vezes chamados de protocolos *ponta a ponta* , embora seja igualmente correto chamá-los de protocolos *de transporte* .

Acima da camada de transporte, há uma série de protocolos de aplicação, como HTTP, FTP, Telnet (login remoto) e o Protocolo Simples de Transferência de Correio (SMTP), que permitem a interoperação de aplicações populares. Para entender a diferença entre um protocolo da camada de aplicação e uma aplicação, pense em todos os diferentes navegadores da World Wide Web que estão ou estiveram disponíveis (por exemplo, Firefox, Chrome, Safari, Netscape, Mosaic, Internet Explorer). Há um número igualmente grande de diferentes implementações de servidores web. A razão pela qual você pode usar qualquer um desses programas de aplicação para acessar um site específico na Web é que todos eles estão em conformidade com o mesmo protocolo da camada de aplicação: HTTP. Confusamente, o mesmo termo às vezes se aplica tanto a

uma aplicação quanto ao protocolo da camada de aplicação que ela usa (por exemplo, FTP é frequentemente usado como o nome de uma aplicação que implementa o protocolo FTP).

A maioria das pessoas que trabalham ativamente na área de redes está familiarizada tanto com a arquitetura da Internet quanto com a arquitetura OSI de 7 camadas, e há um consenso geral sobre como as camadas se mapeiam entre as arquiteturas. A camada de aplicação da Internet é considerada a camada 7, sua camada de transporte é a camada 4, a camada IP (internetworking ou simplesmente rede) é a camada 3 e a camada de enlace ou sub-rede abaixo do IP é a camada 2.

IETF e Padronização

> Embora a chamemos de "arquitetura da Internet" em vez de "arquitetura IETF", é justo dizer que a IETF é o principal órgão de padronização responsável por sua definição, bem como pela especificação de muitos de seus protocolos, como TCP, UDP, IP, DNS e BGP. Mas a arquitetura da Internet também abrange muitos protocolos definidos por outras organizações, incluindo os padrões Ethernet e Wi-Fi 802.11 do IEEE, as especificações web HTTP/HTML do W3C, os padrões de redes celulares 4G e 5G do 3GPP e os padrões de codificação de vídeo H.232 da ITU-T, para citar alguns.

> Além de definir arquiteturas e especificar protocolos, existem outras organizações que apoiam o objetivo maior da interoperabilidade. Um exemplo é a IANA (Autoridade para Atribuição de Números na Internet), que, como o próprio nome indica, é responsável por distribuir os identificadores únicos necessários para o funcionamento dos protocolos. A IANA, por sua vez, é um departamento da ICANN (Corporação da Internet para Atribuição de Nomes e Números), uma organização sem fins lucrativos responsável pela administração geral da internet.

A arquitetura da Internet possui três características que merecem destaque. Primeiro, como melhor ilustrado pela Figura 15 , a arquitetura da Internet não implica em

camadas rígidas. A aplicação é livre para ignorar as camadas de transporte definidas e usar diretamente o IP ou uma das redes subjacentes. De fato, os programadores são livres para definir novas abstrações de canal ou aplicações que rodem sobre qualquer um dos protocolos existentes.

Em segundo lugar, se você observar atentamente o gráfico de protocolos na Figura 14 , notará um formato de ampulheta — largo na parte superior, estreito no meio e largo na parte inferior. Esse formato, na verdade, reflete a filosofia central da arquitetura. Ou seja, o IP serve como ponto focal para a arquitetura — ele define um método comum para troca de pacotes entre uma ampla coleção de redes. Acima do IP, pode haver arbitrariamente muitos protocolos de transporte, cada um oferecendo uma abstração de canal diferente para programas de aplicação. Assim, a questão da entrega de mensagens de host para host é completamente separada da questão de fornecer um serviço útil de comunicação processo a processo. Abaixo do IP, a arquitetura permite arbitrariamente muitas tecnologias de rede diferentes, variando de Ethernet a sem fio e links ponto a ponto únicos.

Um atributo final da arquitetura da Internet (ou, mais precisamente, da cultura da IETF) é que, para que um novo protocolo seja oficialmente incluído na arquitetura, é necessário que haja uma especificação de protocolo e pelo menos uma (e preferencialmente duas) implementações representativas da especificação. A existência de implementações funcionais é necessária para que os padrões sejam adotados pela IETF. Essa premissa cultural da comunidade de design ajuda a garantir que os protocolos da arquitetura possam ser implementados com eficiência. Talvez o valor que a cultura da Internet atribui ao software funcional seja melhor exemplificado por uma citação em camisetas comumente usadas nas reuniões da IETF:

> *Rejeitamos reis, presidentes e votação. Acreditamos em consenso geral e código corrente.* **(David Clark)**

Conclusão principal

Desses três atributos da arquitetura da Internet, a filosofia de design da ampulheta é importante o suficiente para ser repetida. A cintura estreita da ampulheta representa um conjunto mínimo e cuidadosamente selecionado de capacidades globais que permite que aplicações de nível superior e tecnologias de comunicação de nível inferior coexistam, compartilhem capacidades e evoluam rapidamente. O modelo de cintura estreita é crucial para a capacidade da Internet de se adaptar às novas demandas dos usuários e às tecnologias em constante mudança. [Próximo]

1.4 Software

Arquiteturas de rede e especificações de protocolo são essenciais, mas um bom projeto não basta para explicar o sucesso fenomenal da internet: o número de computadores conectados à internet cresceu exponencialmente por mais de três décadas (embora números precisos sejam difíceis de obter). O número de usuários da internet foi estimado em cerca de 4,1 bilhões no final de 2018 — aproximadamente metade da população mundial.

O que explica o sucesso da internet? Certamente, há muitos fatores contribuintes (incluindo uma boa arquitetura), mas um fator que tornou a internet um sucesso estrondoso é o fato de grande parte de sua funcionalidade ser fornecida por softwares executados em computadores de uso geral. A importância disso é que novas funcionalidades podem ser adicionadas facilmente com "apenas uma pequena questão de programação". Como resultado, novos aplicativos e serviços têm surgido em um ritmo incrível.

Um fator relacionado é o aumento massivo do poder computacional disponível em máquinas comuns. Embora as redes de computadores sempre tenham sido capazes, em princípio, de transportar qualquer tipo de informação, como amostras de voz digital,

imagens digitalizadas e assim por diante, esse potencial não era particularmente interessante se os computadores que enviavam e recebiam esses dados fossem lentos demais para fazer algo útil com as informações. Praticamente todos os computadores atuais são capazes de reproduzir áudio e vídeo digitalizados a uma velocidade e resolução bastante utilizáveis.

Nos anos desde o lançamento da primeira edição deste livro, a criação de aplicações em rede tornou-se uma atividade comum e não um trabalho exclusivo para alguns especialistas. Muitos fatores contribuíram para isso, incluindo melhores ferramentas para facilitar o trabalho e a abertura de novos mercados, como aplicativos para smartphones.

O ponto a ser observado é que saber como implementar software de rede é uma parte essencial da compreensão de redes de computadores e, embora seja provável que você não seja incumbido de implementar um protocolo de baixo nível como o IP, há uma boa chance de você encontrar motivos para implementar um protocolo de nível de aplicação — o elusivo "aplicativo matador" que levará a fama e fortuna inimagináveis. Para começar, esta seção apresenta algumas das questões envolvidas na implementação de uma aplicação de rede sobre a Internet. Normalmente, esses programas são simultaneamente uma aplicação (ou seja, projetados para interagir com os usuários) e um protocolo (ou seja, comunicam-se com pares na rede).

1.4.1 API de soquete

O ponto de partida para implementar uma aplicação de rede é a interface exportada pela rede. Como a maioria dos protocolos de rede está em software (especialmente aqueles no topo da pilha de protocolos) e quase todos os sistemas de computador implementam seus protocolos de rede como parte do sistema operacional, quando nos referimos à interface "exportada pela rede", geralmente nos referimos à interface que o

sistema operacional fornece ao seu subsistema de rede. Essa interface é frequentemente chamada de *interface de programação de aplicações* (API) de rede.

Embora cada sistema operacional seja livre para definir sua própria API de rede (e a maioria o fez), com o tempo, algumas dessas APIs passaram a ter amplo suporte; ou seja, foram portadas para sistemas operacionais diferentes do seu sistema nativo. Foi o que aconteceu com a *interface de soquete* originalmente fornecida pela distribuição Berkeley do Unix, que agora é suportada em praticamente todos os sistemas operacionais populares e é a base de interfaces específicas de linguagem, como a biblioteca de soquetes Java ou Python. Usamos Linux e C para todos os exemplos de código neste livro: Linux porque é de código aberto e C porque continua sendo a linguagem preferida para componentes internos de rede. (C também tem a vantagem de expor todos os detalhes de baixo nível, o que é útil para entender as ideias subjacentes.)

Explosão de aplicativos habilitados para soquetes

É difícil exagerar a importância da API Socket. Ela define o ponto de demarcação entre os aplicativos executados na internet e os detalhes de como a internet é implementada. Como consequência do fornecimento de uma interface bem definida e estável pelos Sockets, a criação de aplicativos para a internet se tornou uma indústria multibilionária. Partindo dos primórdios humildes do paradigma cliente/servidor e de alguns programas aplicativos simples, como e-mail, transferência de arquivos e login remoto, todos agora têm acesso a um suprimento infinito de aplicativos em nuvem a partir de seus smartphones.

Esta seção estabelece as bases revisitando a simplicidade de um programa cliente abrindo um socket para trocar mensagens com um programa servidor. No entanto, hoje, um rico ecossistema de software é construído sobre a API Socket. Essa camada inclui uma infinidade de ferramentas baseadas em nuvem que reduzem as barreiras para a implementação de aplicativos escaláveis. Retornaremos à interação entre a nuvem e a rede em cada capítulo, começando pela seção *"Perspectiva"* no final do Capítulo 1.

Antes de descrever a interface de soquete, é importante ter em mente duas preocupações distintas. Cada protocolo fornece um conjunto específico de *serviços* , e a API fornece uma *sintaxe* pela qual esses serviços podem ser invocados em um sistema computacional específico. A implementação é então responsável por mapear o conjunto tangível de operações e objetos definidos pela API no conjunto abstrato de serviços definido pelo protocolo. Se você tiver feito um bom trabalho ao definir a interface, será possível usar a sintaxe da interface para invocar os serviços de muitos protocolos diferentes. Essa generalidade certamente era um objetivo da interface de soquete, embora esteja longe de ser perfeita.

A principal abstração da interface de socket, sem surpresa, é o *socket* . Uma boa maneira de pensar em um socket é como o ponto onde um processo de aplicação local se conecta à rede. A interface define as operações para criar um socket, conectá-lo à rede, enviar/receber mensagens através do socket e fechá-lo. Para simplificar a discussão, vamos nos limitar a mostrar como os sockets são usados com o TCP.

O primeiro passo é criar um socket, o que é feito com a seguinte operação:

```
int socket(int domain, int type, int protocol);
```

O motivo pelo qual essa operação recebe três argumentos é que a interface do soquete foi projetada para ser genérica o suficiente para suportar qualquer conjunto de protocolos subjacente. Especificamente, o `domain`argumento especifica a *família* de protocolos que será usada: `PF_INET`denota a família da Internet, `PF_UNIX`denota o recurso de pipe do Unix e `PF_PACKET`denota acesso direto à interface de rede (ou seja, ignora a pilha de protocolos TCP/IP). O `type`argumento indica a semântica da comunicação. `SOCK_STREAM`é usado para denotar um fluxo de bytes. `SOCK_DGRAM`é uma alternativa que denota um serviço orientado a mensagens, como o fornecido pelo UDP. O `protocol`argumento identifica o protocolo específico que será usado. Em nosso caso, esse argumento se `UNSPEC`deve à combinação de `PF_INET`e `SOCK_STREAM`implica TCP. Por

fim, o valor de retorno de `socket` é um *identificador* para o soquete recém-criado — ou seja, um identificador pelo qual podemos nos referir ao soquete no futuro. Ele é fornecido como argumento para operações subsequentes nesse soquete.

O próximo passo depende se você é um cliente ou um servidor. Em uma máquina servidora, o processo do aplicativo executa uma abertura *passiva* — o servidor diz que está preparado para aceitar conexões, mas não estabelece uma conexão de fato. O servidor faz isso invocando as três operações a seguir:

```
int bind(int socket, struct sockaddr *address, int addr_len);

int listen(int socket, int backlog);

int accept(int socket, struct sockaddr *address, int *addr_len);
```

A `bind`operação, como o próprio nome sugere, vincula o recém-criado `socket`ao especificado `address`. Este é o endereço de rede do participante *local* — o servidor. Observe que, quando usado com os protocolos de Internet, `address`é uma estrutura de dados que inclui o endereço IP do servidor e um número de porta TCP. Portas são usadas para identificar processos indiretamente. Elas são uma forma de *chaves demux* . O número da porta geralmente é um número conhecido específico do serviço oferecido; por exemplo, servidores web geralmente aceitam conexões na porta 80.

A `listen`operação define então quantas conexões podem estar pendentes no especificado `socket`. Por fim, a `accept`operação executa a abertura passiva. É uma operação de bloqueio que não retorna até que um participante remoto tenha estabelecido uma conexão e, quando concluída, retorna um *novo* soquete que corresponde a essa conexão recém-estabelecida, e o `address`argumento contém o endereço do participante *remoto*`accept` . Observe que, quando retorna, o soquete original fornecido como argumento ainda existe e corresponde à abertura passiva; ele é usado em invocações futuras de `accept`.

Na máquina cliente, o processo do aplicativo executa uma abertura *ativa* ; ou seja, ele diz com quem deseja se comunicar invocando a seguinte operação única:

```
int connect(int socket, struct sockaddr *address, int addr_len);
```

Esta operação não retorna até que o TCP tenha estabelecido uma conexão com sucesso, momento em que a aplicação está livre para começar a enviar dados. Neste caso, `address`contém o endereço do participante remoto. Na prática, o cliente geralmente especifica apenas o endereço do participante remoto e deixa o sistema preencher as informações locais. Enquanto um servidor geralmente escuta mensagens em uma porta conhecida, um cliente normalmente não se importa com qual porta usa para si; o sistema operacional simplesmente seleciona uma não utilizada.

Depois que uma conexão é estabelecida, os processos do aplicativo invocam as duas operações a seguir para enviar e receber dados:

```
int send(int socket, char *message, int msg_len, int flags);

int recv(int socket, char *buffer, int buf_len, int flags);
```

A primeira operação envia o dado `message`para o especificado `socket`, enquanto a segunda operação recebe uma mensagem do especificado `socket`para o dado `buffer`. Ambas as operações utilizam um conjunto de `flags`funções que controlam certos detalhes da operação.

1.4.2 Exemplo de cliente/servidor

Agora, mostramos a implementação de um programa cliente/servidor simples que utiliza a interface de soquete para enviar mensagens através de uma conexão TCP. O programa também utiliza outros utilitários de rede Linux, que apresentaremos à medida

que avançamos. Nosso aplicativo permite que um usuário em uma máquina digite e envie texto para um usuário em outra máquina. Trata-se de uma versão simplificada do `talk`programa Linux, semelhante ao programa que compõe o núcleo dos aplicativos de mensagens instantâneas.

Cliente

Começamos com o lado do cliente, que recebe o nome da máquina remota como argumento. Ele chama o utilitário Linux para traduzir esse nome no endereço IP do host remoto. O próximo passo é construir a estrutura de dados de endereço (`sin`) esperada pela interface do soquete. Observe que essa estrutura de dados especifica que usaremos o soquete para conectar à Internet (`AF_INET`). Em nosso exemplo, usamos a porta TCP 5432 como a porta do servidor conhecida; esta é uma porta que não foi atribuída a nenhum outro serviço de Internet. A etapa final na configuração da conexão é chamar `socket`e `connect`. Assim que a operação retorna, a conexão é estabelecida e o programa cliente entra em seu loop principal, que lê o texto da entrada padrão e o envia pelo soquete.

```
#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>
```

```c
#define SERVER_PORT 5432
#define MAX_LINE 256

int
main(int argc, char * argv[])
{
  FILE *fp;
  struct hostent *hp;
  struct sockaddr_in sin;
  char *host;
  char buf[MAX_LINE];
  int s;
  int len;

  if (argc==2) {
    host = argv[1];
  }
  else {
    fprintf(stderr, "usage: simplex-talk host\n");
    exit(1);
```

```
  }


  /* translate host name into peer's IP address */
  hp = gethostbyname(host);
  if (!hp) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
  }


  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
  sin.sin_port = htons(SERVER_PORT);


  /* active open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
```

```
  if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
  {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
  }
  /* main loop: get and send lines of text */
  while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
  }
}
```

Servidor

O servidor é igualmente simples. Ele primeiro constrói a estrutura de dados do endereço preenchendo seu próprio número de porta (`SERVER_PORT`). Ao não especificar um endereço IP, o programa aplicativo está disposto a aceitar conexões em qualquer um dos endereços IP do host local. Em seguida, o servidor executa as etapas preliminares envolvidas em uma abertura passiva: ele cria o soquete, o vincula ao endereço local e define o número máximo de conexões pendentes permitidas. Por fim,

o loop principal aguarda que um host remoto tente se conectar e, quando isso acontece, ele recebe e imprime os caracteres que chegam na conexão.

```
#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>



#define SERVER_PORT  5432

#define MAX_PENDING  5

#define MAX_LINE     256



int

main()

{

  struct sockaddr_in sin;

  char buf[MAX_LINE];
```

```
  int buf_len;
  socklen_t addr_len;
  int s, new_s;

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = htons(SERVER_PORT);

  /* setup passive open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("simplex-talk: bind");
    exit(1);
  }
  listen(s, MAX_PENDING);
```

```
 /* wait for connection, then receive and print text */

  while(1) {

    if ((new_s = accept(s, (struct sockaddr *)&sin, &addr_len)) < 0) {

      perror("simplex-talk: accept");

      exit(1);

    }

    while (buf_len = recv(new_s, buf, sizeof(buf), 0))

      fputs(buf, stdout);

    close(new_s);

  }

}
```

1.5 Desempenho

Até aqui, focamos principalmente nos aspectos funcionais das redes. Como qualquer sistema computacional, no entanto, espera-se que as redes de computadores também tenham um bom desempenho. Isso ocorre porque a eficácia das computações distribuídas pela rede frequentemente depende diretamente da eficiência com que a rede entrega os dados computacionais. Embora o velho ditado da programação "primeiro acerte e depois acelere" permaneça verdadeiro, em redes muitas vezes é

necessário "projetar para o desempenho". Portanto, é importante compreender os vários fatores que impactam o desempenho da rede.

1.5.1 Largura de banda e latência

O desempenho da rede é medido de duas maneiras fundamentais: *largura de banda* (também chamada de *throughput*) e *latência* (também chamada de *delay*). A largura de banda de uma rede é dada pelo número de bits que podem ser transmitidos pela rede em um determinado período de tempo. Por exemplo, uma rede pode ter uma largura de banda de 10 milhões de bits/segundo (Mbps), o que significa que ela é capaz de entregar 10 milhões de bits por segundo. Às vezes, é útil pensar na largura de banda em termos de quanto tempo leva para transmitir cada bit de dados. Em uma rede de 10 Mbps, por exemplo, leva 0,1 microssegundo (μs) para transmitir cada bit.

Largura de banda e taxa de transferência são termos sutilmente diferentes. Em primeiro lugar, largura de banda é literalmente uma medida da largura de uma banda de frequência. Por exemplo, linhas telefônicas tradicionais de nível de voz suportavam uma banda de frequência que variava de 300 a 3300 Hz; dizia-se que ela tinha uma largura de banda de 3300 Hz - 300 Hz = 3000 Hz. Se você vir a palavra *largura de banda* usada em uma situação em que ela é medida em hertz, provavelmente se refere à faixa de sinais que pode ser acomodada.

Quando falamos sobre a largura de banda de um link de comunicação, normalmente nos referimos ao número de bits por segundo que podem ser transmitidos no link. Isso também é às vezes chamado de *taxa de dados.* Podemos dizer que a largura de banda de um link Ethernet é de 10 Mbps. Uma distinção útil também pode ser feita, no entanto, entre a taxa máxima de dados disponível no link e o número de bits por segundo que podemos realmente transmitir pelo link na prática. Tendemos a usar o termo *throughput* para nos referirmos ao *desempenho medido* de um sistema. Assim, devido a várias ineficiências de implementação, um par de nós conectados por um link com uma largura de banda de 10 Mbps pode atingir um throughput de apenas 2 Mbps.

Isso significaria que uma aplicação em um host poderia enviar dados para o outro host a 2 Mbps.

Por fim, frequentemente falamos sobre os *requisitos* de largura de banda de uma aplicação. Trata-se do número de bits por segundo que ela precisa transmitir pela rede para ter um desempenho aceitável. Para algumas aplicações, isso pode ser "qualquer coisa que eu conseguir"; para outras, pode ser um número fixo (de preferência, não maior que a largura de banda disponível no link); e para outras, pode ser um número que varia com o tempo. Forneceremos mais informações sobre esse tópico posteriormente nesta seção.

Embora seja possível falar sobre a largura de banda da rede como um todo, às vezes é necessário ser mais preciso, concentrando-se, por exemplo, na largura de banda de um único link físico ou de um canal lógico entre processos. No nível físico, a largura de banda está em constante aprimoramento, sem fim à vista. Intuitivamente, se você pensar em um segundo de tempo como uma distância que pode ser medida com uma régua e na largura de banda como quantos bits cabem nessa distância, poderá pensar em cada bit como um pulso de alguma largura. Por exemplo, cada bit em um link de 1 Mbps tem 1 μs de largura, enquanto cada bit em um link de 2 Mbps tem 0,5 μs de largura, conforme ilustrado na Figura 16. Quanto mais sofisticada a tecnologia de transmissão e recepção, mais estreito cada bit pode se tornar e, portanto, maior a largura de banda. Para canais lógicos entre processos, a largura de banda também é influenciada por outros fatores, incluindo quantas vezes o software que implementa o canal precisa manipular e, possivelmente, transformar cada bit de dados.

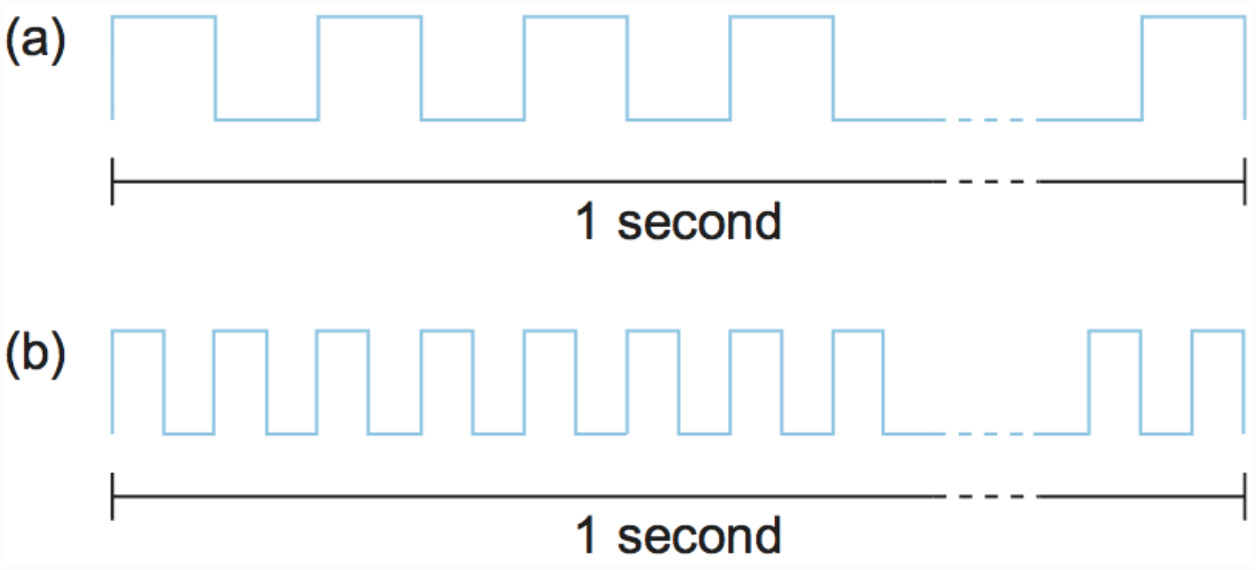

Figura 16. *Os bits transmitidos em uma largura de banda específica podem ser considerados como tendo alguma largura: (a) bits transmitidos a 1 Mbps (cada bit tem 1 microssegundo de largura); (b) bits transmitidos a 2 Mbps (cada bit tem 0,5 microssegundo de largura).*

A segunda métrica de desempenho, latência, corresponde ao tempo que uma mensagem leva para viajar de uma extremidade da rede à outra. (Assim como com a largura de banda, poderíamos nos concentrar na latência de um único link ou de um canal ponta a ponta.) A latência é medida estritamente em termos de tempo. Por exemplo, uma rede transcontinental pode ter uma latência de 24 milissegundos (ms); ou seja, uma mensagem leva 24 ms para viajar de uma costa da América do Norte à outra. Há muitas situações em que é mais importante saber quanto tempo leva para enviar uma mensagem de uma extremidade da rede à outra e vice-versa, em vez da latência unidirecional. Chamamos isso de *tempo de ida e volta* (RTT) da rede.

Frequentemente pensamos na latência como tendo três componentes. Primeiro, há o atraso de propagação da velocidade da luz. Esse atraso ocorre porque nada, incluindo um bit em um fio, pode viajar mais rápido do que a velocidade da luz. Se você conhece a distância entre dois pontos, pode calcular a latência da velocidade da luz, embora seja necessário ter cuidado, pois a luz viaja por diferentes meios em velocidades diferentes: ela viaja a $3{,}0 \times 10^{-}$ ...

```
Latency = Propagation + Transmit + Queue

Propagation =  Distance/SpeedOfLight

Transmit = Size/Bandwidth
```

onde `Distance`λ é o comprimento do fio pelo qual os dados trafegarão, `SpeedOfLight`λ é a velocidade efetiva da luz sobre esse fio, `Size`λ é o tamanho do pacote e `Bandwidth`λ é a largura de banda na qual o pacote é transmitido. Observe que, se a mensagem contiver apenas um bit e estivermos falando de um único link (em oposição a uma rede inteira), os termos `Transmit`λ e `Queue`λ não são relevantes, e a latência corresponde apenas ao atraso de propagação.

Largura de banda e latência se combinam para definir as características de desempenho de um determinado link ou canal. Sua importância relativa, no entanto, depende da aplicação. Para algumas aplicações, a latência domina a largura de banda. Por exemplo, um cliente que envia uma mensagem de 1 byte para um servidor e recebe uma mensagem de 1 byte em retorno é limitado pela latência. Assumindo que nenhuma computação séria esteja envolvida na preparação da resposta, a aplicação terá um desempenho muito diferente em um canal transcontinental com um RTT de 100 ms do que em um canal do outro lado da sala com um RTT de 1 ms. Se o canal é de 1 Mbps ou 100 Mbps é relativamente insignificante, no entanto, já que o primeiro implica que o tempo para transmitir um byte (`Transmit`) é de 8 µs e o último implica `Transmit`= 0,08 µs.

Em contraste, considere um programa de biblioteca digital que está sendo solicitado a buscar uma imagem de 25 megabytes (MB) — quanto maior a largura de banda disponível, mais rápido ele poderá retornar a imagem ao usuário. Aqui, a largura de banda do canal domina o desempenho. Para ver isso, suponha que o canal tenha uma largura de banda de 10 Mbps. Levará 20 segundos para transmitir a imagem ($25 \times 10^6 \times 8$ bits / 10×10^6 Mbps = 20 segundos), tornando relativamente irrelevante se a

imagem está do outro lado de um canal de 1 ms ou de um canal de 100 ms; a diferença entre um tempo de resposta de 20,001 segundos e um tempo de resposta de 20,1 segundos é insignificante.

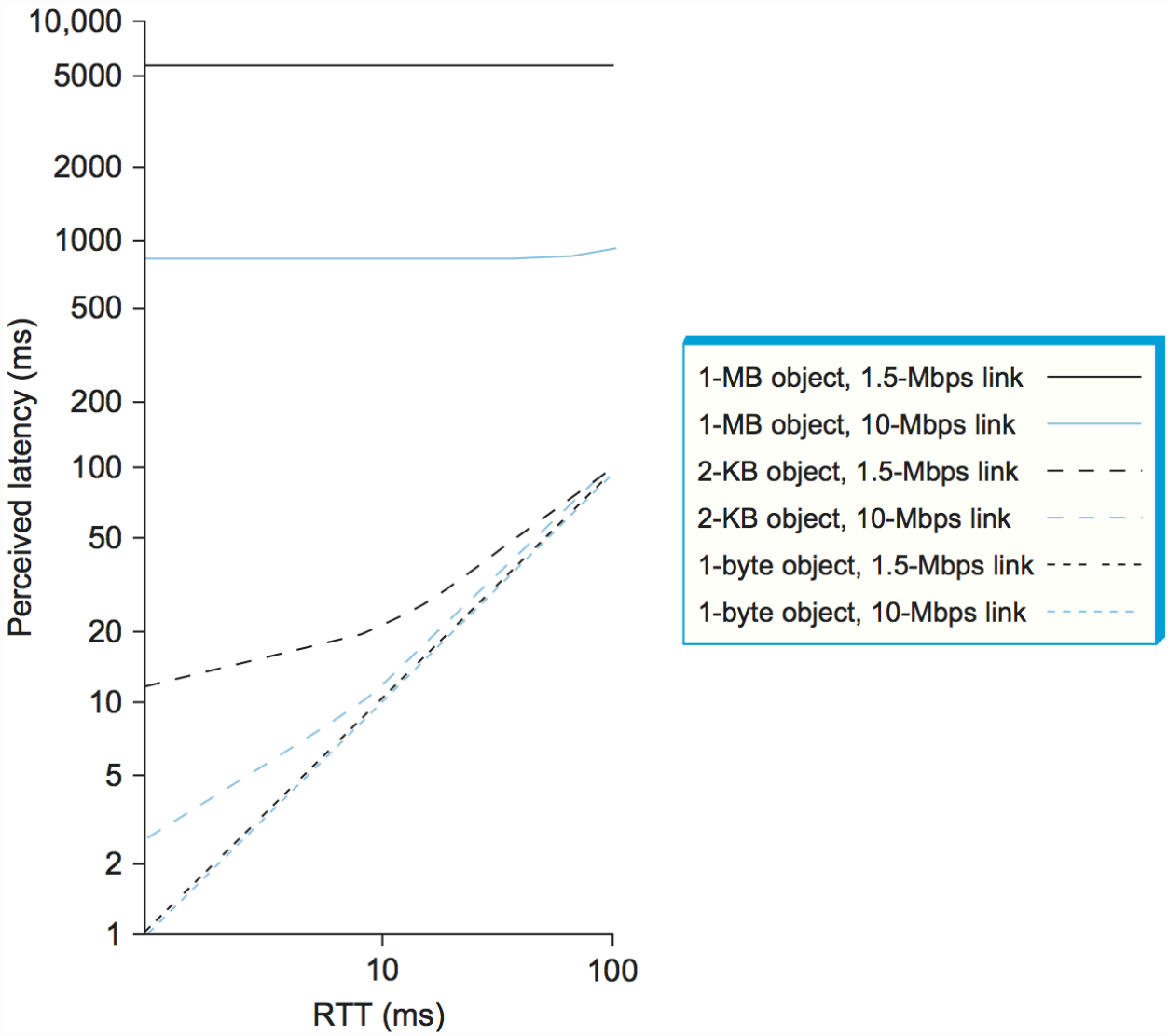

Figura 17. *Latência percebida (tempo de resposta) versus tempo de ida e volta para vários tamanhos de objetos e velocidades de link.*

A Figura 17 dá uma ideia de como a latência ou a largura de banda podem dominar o desempenho em diferentes circunstâncias. O gráfico mostra quanto tempo leva para mover objetos de vários tamanhos (1 byte, 2 KB, 1 MB) entre redes com RTTs variando de 1 a 100 ms e velocidades de link de 1,5 ou 10 Mbps. Usamos escalas logarítmicas

para mostrar o desempenho relativo. Para um objeto de 1 byte (por exemplo, uma tecla pressionada), a latência permanece quase exatamente igual ao RTT, de modo que não é possível distinguir entre uma rede de 1,5 Mbps e uma rede de 10 Mbps. Para um objeto de 2 KB (por exemplo, uma mensagem de e-mail), a velocidade do link faz uma grande diferença em uma rede RTT de 1 ms, mas uma diferença insignificante em uma rede RTT de 100 ms. E para um objeto de 1 MB (por exemplo, uma imagem digital), o RTT não faz diferença — é a velocidade do link que domina o desempenho em toda a faixa de RTT.

Observe que, ao longo deste livro, usamos os termos *latência* e *atraso* de forma genérica para denotar o tempo necessário para executar uma função específica, como entregar uma mensagem ou mover um objeto. Quando nos referimos ao tempo específico que um sinal leva para se propagar de uma extremidade a outra de um enlace, usamos o termo *atraso de propagação* . Além disso, deixamos claro, no contexto da discussão, se estamos nos referindo à latência unidirecional ou ao tempo de ida e volta.

A propósito, os computadores estão se tornando tão rápidos que, quando os conectamos a redes, às vezes é útil pensar, pelo menos figurativamente, em termos de *instruções por milha* . Considere o que acontece quando um computador capaz de executar 100 bilhões de instruções por segundo envia uma mensagem em um canal com um RTT de 100 ms. (Para facilitar a matemática, suponha que a mensagem cubra uma distância de 8.000 km.) Se esse computador ficar ocioso durante os 100 ms completos aguardando uma mensagem de resposta, ele terá perdido a capacidade de executar 10 bilhões de instruções, ou 2 milhões de instruções por milha. Seria melhor se tivesse valido a pena examinar a rede para justificar esse desperdício.

1.5.2 Produto de atraso × largura de banda

Também é útil falar sobre o produto dessas duas métricas, frequentemente chamado de *produto atraso × largura de banda* . Intuitivamente, se pensarmos em um canal entre um par de processos como um tubo oco (veja a Figura 18), onde a latência

corresponde ao comprimento do tubo e a largura de banda fornece o diâmetro do tubo, então o produto atraso × largura de banda fornece o volume do tubo — o número máximo de bits que podem estar em trânsito pelo tubo em qualquer instante. Dito de outra forma, se a latência (medida em tempo) corresponde ao comprimento do tubo, então, dada a largura de cada bit (também medida em tempo), você pode calcular quantos bits cabem no tubo. Por exemplo, um canal transcontinental com uma latência unidirecional de 50 ms e uma largura de banda de 45 Mbps é capaz de conter

$$\mathbf{50 \times 10^{-3} \times 45 \times 10^{6}\ \textit{bits/seg} = 2{,}25 \times 10^{6}\ \textit{bits}}$$

ou aproximadamente 280 KB de dados. Em outras palavras, este canal (pipe) de exemplo contém tantos bytes quanto a memória de um computador pessoal do início da década de 1980 conseguia armazenar.

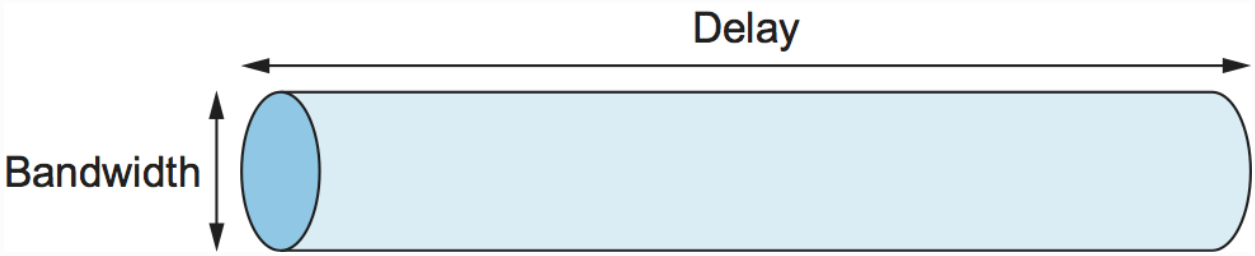

Figura 18. *Rede como um tubo.*

O produto atraso × largura de banda é importante para a construção de redes de alto desempenho, pois corresponde a quantos bits o remetente deve transmitir antes que o primeiro bit chegue ao destinatário. Se o remetente espera que o destinatário sinalize de alguma forma que os bits estão começando a chegar, e é necessária outra latência de canal para que esse sinal se propague de volta ao remetente, o remetente pode enviar dados equivalentes a um *RTT* × *largura de banda* antes de ouvir do destinatário que tudo está bem. Os bits no canal são considerados "em trânsito", o que significa que, se o destinatário disser ao remetente para parar de transmitir, ele poderá receber dados equivalentes a um RTT × largura de banda antes que o remetente consiga responder. Em nosso exemplo acima, essa quantidade corresponde a 5,5 × 10 bits (671

KB) de dados. Por outro lado, se o remetente não preencher o canal — ou seja, não enviar dados equivalentes a um RTT × largura de banda antes de parar para aguardar um sinal — o remetente não utilizará totalmente a rede.

Observe que, na maioria das vezes, estamos interessados no cenário RTT, ao qual nos referimos simplesmente como o produto atraso × largura de banda, sem dizer explicitamente que "atraso" é o RTT (ou seja, multiplicar o atraso unidirecional por dois). Normalmente, se o "atraso" em atraso × largura de banda significa latência unidirecional ou RTT fica claro pelo contexto. A Tabela 1 mostra alguns exemplos de produtos RTT × largura de banda para alguns links de rede típicos.

Tipo de link	Largura de banda	Distância unidirecional	RTT	RTT x Largura de Banda
LAN sem fio	54 Mbps	50 metros	0,33 µs	18 bits
Satélite	1 Gbps	35.000 km	230 ms	230 MB
Fibra cross-country	10 Gbps	4.000 km	40 ms	400 MB

1.5.3 Redes de alta velocidade

O aparente aumento contínuo na largura de banda faz com que os projetistas de rede comecem a pensar sobre o que acontece no limite ou, dito de outra forma, qual é o impacto no projeto de rede de ter largura de banda infinita disponível.

Embora as redes de alta velocidade tragam uma mudança drástica na largura de banda disponível para as aplicações, em muitos aspectos seu impacto na forma como pensamos sobre redes reside no que não *muda* com o aumento da largura de banda: a velocidade da luz. Para citar Scotty, de *Jornada nas Estrelas,* "Vocês não podem mudar as leis da física". Em outras palavras, "alta velocidade" não significa que a latência melhora na mesma proporção que a largura de banda; o RTT transcontinental de um link de 1 Gbps é o mesmo de 100 ms que para um link de 1 Mbps.

Para compreender a importância da largura de banda cada vez maior diante da latência fixa, considere o que é necessário para transmitir um arquivo de 1 MB em uma rede de 1 Mbps em comparação com uma rede de 1 Gbps, ambas com um RTT de 100 ms. No caso da rede de 1 Mbps, são necessários 80 tempos de ida e volta para transmitir o arquivo; durante cada RTT, 1,25% do arquivo é enviado. Em contraste, o mesmo arquivo de 1 MB nem chega perto de preencher o equivalente a 1 RTT do link de 1 Gbps, que tem um produto atraso x largura de banda de 12,5 MB.

A Figura 19 ilustra a diferença entre as duas redes. Na prática, o arquivo de 1 MB parece um fluxo de dados que precisa ser transmitido por uma rede de 1 Mbps, enquanto em uma rede de 1 Gbps parece um único pacote. Para ajudar a ilustrar esse ponto, considere que um arquivo de 1 MB representa para uma rede de 1 Gbps o que um *pacote de* 1 KB representa para uma rede de 1 Mbps.

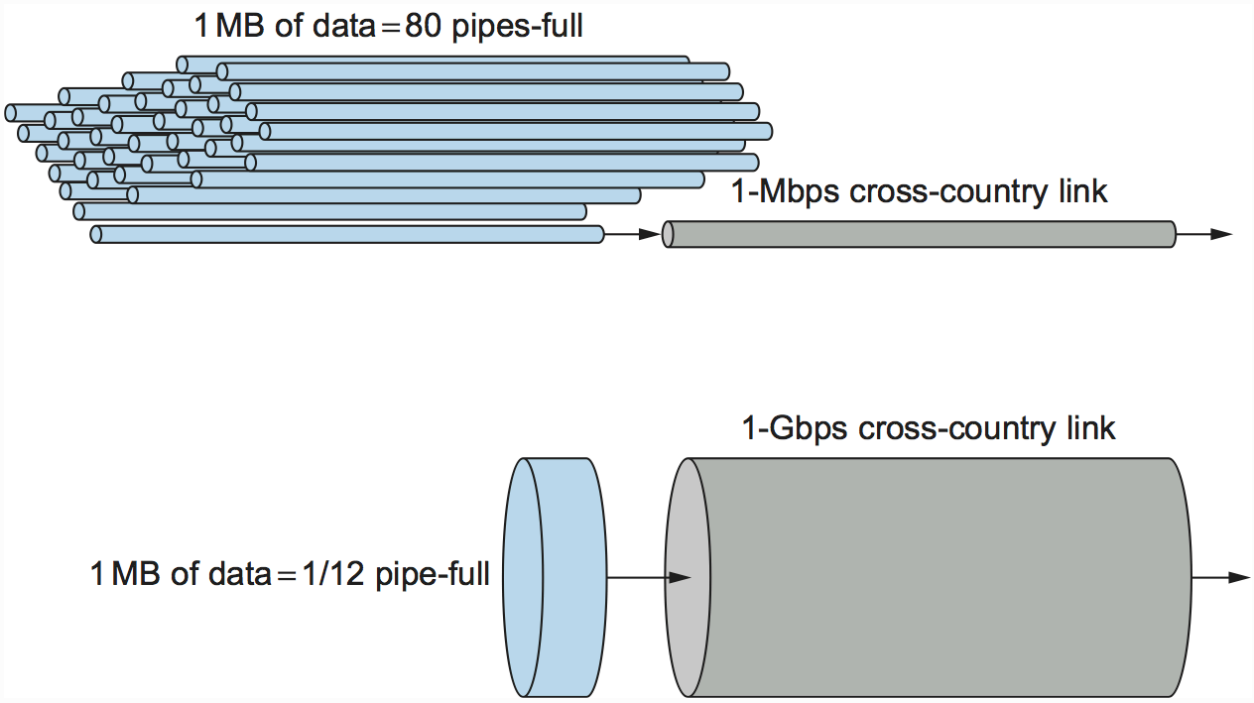

Figura 19. *Relação entre largura de banda e latência. Um arquivo de 1 MB preencheria o link de 1 Mbps 80 vezes, mas preencheria apenas 1/12 de um link de 1 Gbps.*

Outra maneira de pensar sobre a situação é que mais dados podem ser transmitidos durante cada RTT em uma rede de alta velocidade, tanto que um único RTT se torna um período de tempo significativo. Assim, embora você não pense duas vezes sobre a diferença entre uma transferência de arquivo levar 101 RTTs em vez de 100 RTTs (uma diferença relativa de apenas 1%), de repente a diferença entre 1 RTT e 2 RTTs é significativa — um aumento de 100%. Em outras palavras, a latência, em vez da taxa de transferência, começa a dominar nosso pensamento sobre o projeto de redes.

Talvez a melhor maneira de entender a relação entre taxa de transferência e latência seja retornar ao básico. A taxa de transferência efetiva de ponta a ponta que pode ser alcançada em uma rede é dada pela simples relação

Taxa de transferência = Tamanho da transferência / Tempo de transferência

onde TransferTime inclui não apenas os elementos de sentido único identificados anteriormente nesta seção, mas também qualquer tempo adicional gasto solicitando ou configurando a transferência. Geralmente, representamos esse relacionamento como

Tempo de transferência = RTT + 1/Largura de banda x Tamanho de transferência

Usamos neste cálculo para contabilizar uma mensagem de solicitação enviada pela rede e os dados enviados de volta. Por exemplo, considere uma situação em que um usuário deseja buscar um arquivo de 1 MB em uma rede de 1 Gbps com um tempo de ida e volta de 100 ms. Isso inclui tanto o tempo de transmissão para 1 MB (1/1 Gbps × 1 MB = 8 ms) quanto o RTT de 100 ms, resultando em um tempo total de transferência de 108 ms. Isso significa que a taxa de transferência efetiva será

1 MB / 108 ms = 74,1 Mbps

não 1 Gbps. Claramente, transferir uma quantidade maior de dados ajudará a melhorar a taxa de transferência efetiva, enquanto, no limite, um tamanho de transferência infinitamente grande fará com que a taxa de transferência efetiva se aproxime da largura de banda da rede. Por outro lado, ter que suportar mais de 1 RTT — por exemplo, para retransmitir pacotes perdidos — prejudicará a taxa de transferência efetiva para qualquer transferência de tamanho finito e será mais perceptível para transferências pequenas.

1.5.4 Requisitos de aplicação

A discussão nesta seção adotou uma visão de desempenho centrada na rede; ou seja, falamos em termos do que um determinado link ou canal suportará. A suposição implícita foi que os programas aplicativos têm necessidades simples — eles querem o máximo de largura de banda que a rede puder fornecer. Isso certamente se aplica ao programa de biblioteca digital mencionado anteriormente, que está recuperando uma imagem de 250 MB; quanto mais largura de banda estiver disponível, mais rápido o programa poderá retornar a imagem ao usuário.

No entanto, algumas aplicações são capazes de declarar um limite superior para a quantidade de largura de banda que precisam. Aplicações de vídeo são um excelente exemplo. Suponha que alguém queira transmitir um vídeo que tenha um quarto do tamanho de uma tela de TV padrão; ou seja, tenha uma resolução de 352 por 240 pixels. Se cada pixel for representado por 24 bits de informação, como seria o caso para cores de 24 bits, então o tamanho de cada quadro seria (352 × 240 × 24) / 8 = 247,5 KB Se o aplicativo precisar suportar uma taxa de quadros de 30 quadros por segundo, ele poderá solicitar uma taxa de transferência de 75 Mbps. A capacidade da rede de fornecer mais largura de banda não interessa a tal aplicação porque ela tem apenas uma quantidade limitada de dados para transmitir em um determinado período de tempo.

Infelizmente, a situação não é tão simples quanto este exemplo sugere. Como a diferença entre dois quadros adjacentes em um fluxo de vídeo costuma ser pequena, é possível compactar o vídeo transmitindo apenas as diferenças entre os quadros adjacentes. Cada quadro também pode ser compactado, pois nem todos os detalhes de uma imagem são facilmente percebidos pelo olho humano. O vídeo compactado não flui a uma taxa constante, mas varia com o tempo, de acordo com fatores como a quantidade de ação e detalhes na imagem e o algoritmo de compactação utilizado. Portanto, é possível prever qual será a largura de banda média necessária, mas a taxa instantânea pode ser maior ou menor.

A questão fundamental é o intervalo de tempo em que a média é calculada. Suponha que este aplicativo de vídeo de exemplo possa ser comprimido a ponto de precisar de apenas 2 Mbps, em média. Se ele transmite 1 megabit em um intervalo de 1 segundo e 3 megabits no intervalo de 1 segundo seguinte, então, no intervalo de 2 segundos, ele está transmitindo a uma taxa média de 2 Mbps; no entanto, isso será de pouca utilidade para um canal que foi projetado para suportar no máximo 2 megabits por segundo. Obviamente, apenas conhecer as necessidades médias de largura de banda de um aplicativo nem sempre será suficiente.

Geralmente, no entanto, é possível estabelecer um limite superior para a magnitude do burst que uma aplicação como esta provavelmente transmitirá. Um burst pode ser descrito por uma taxa de pico mantida por um determinado período de tempo. Alternativamente, pode ser descrito como o número de bytes que podem ser enviados na taxa de pico antes de retornar à taxa média ou a uma taxa inferior. Se essa taxa de pico for maior que a capacidade disponível do canal, os dados excedentes precisarão ser armazenados em buffer em algum lugar para serem transmitidos posteriormente. Saber a magnitude do burst que pode ser enviado permite que o projetista da rede aloque capacidade de buffer suficiente para conter o burst.

De forma análoga à forma como as necessidades de largura de banda de uma aplicação podem ser diferentes de "tudo o que ela pode obter", os requisitos de atraso de uma aplicação podem ser mais complexos do que simplesmente "o mínimo de atraso possível". No caso do atraso, às vezes não importa tanto se a latência unidirecional da rede é de 100 ms ou 500 ms, mas sim a variação da latência de pacote para pacote. A variação na latência é chamada de *jitter* .

Considere a situação em que a origem envia um pacote a cada 33 ms, como seria o caso de um aplicativo de vídeo transmitindo quadros 30 vezes por segundo. Se os pacotes chegam ao destino com um intervalo exato de 33 ms, podemos deduzir que o atraso experimentado por cada pacote na rede foi exatamente o mesmo. Se o espaçamento entre a chegada dos pacotes ao destino — às vezes chamado de *intervalo entre pacotes* — for variável, no entanto, o atraso experimentado pela sequência de pacotes também deve ter sido variável, e diz-se que a rede introduziu jitter no fluxo de pacotes, como mostrado na Figura 20. Essa variação geralmente não é introduzida em um único link físico, mas pode ocorrer quando os pacotes experimentam diferentes atrasos de enfileiramento em uma rede comutada por pacotes multi-hop. Esse atraso de enfileiramento corresponde ao componente de latência definido anteriormente nesta seção, que varia com o tempo.

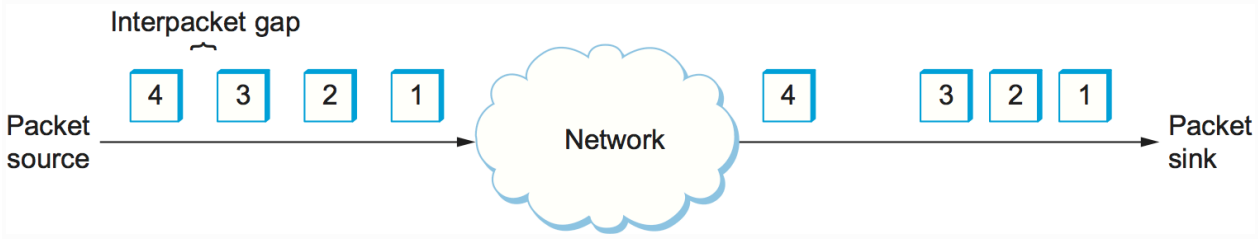

Figura 20. *Jitter induzido pela rede.*

Para entender a relevância do jitter, suponha que os pacotes transmitidos pela rede contenham quadros de vídeo e, para exibir esses quadros na tela, o receptor precise receber um novo a cada 33 ms. Se um quadro chegar antes, ele pode simplesmente ser salvo pelo receptor até o momento de exibi-lo. Infelizmente, se um quadro chegar atrasado, o receptor não terá o quadro necessário a tempo de atualizar a tela, e a qualidade do vídeo será prejudicada; não será suave. Observe que não é necessário eliminar o jitter, apenas saber o quão ruim ele é. A razão para isso é que, se o receptor conhece os limites superior e inferior da latência que um pacote pode experimentar, ele pode atrasar o momento em que inicia a reprodução do vídeo (ou seja, exibe o primeiro quadro) por tempo suficiente para garantir que, no futuro, sempre terá um quadro para exibir quando precisar. O receptor atrasa o quadro, efetivamente suavizando o jitter, armazenando-o em um buffer.

Perspectiva: Velocidade do recurso

Este capítulo apresenta algumas das partes interessadas em redes de computadores — projetistas de redes, desenvolvedores de aplicações, usuários finais e operadores de redes — para ajudar a motivar os requisitos técnicos que moldam a forma como as redes são projetadas e construídas. Isso pressupõe que todas as decisões de projeto sejam puramente técnicas, mas, claro, geralmente não é esse o caso. Muitos outros fatores, desde forças de mercado a políticas governamentais e considerações éticas, também influenciam a forma como as redes são projetadas e construídas.

Destes, o mercado é o mais influente e corresponde à interação entre operadoras de rede (por exemplo, AT&T, Comcast, Verizon, DT, NTT, China Unicom), fornecedores de equipamentos de rede (por exemplo, Cisco, Juniper, Ericsson, Nokia, Huawei, NEC), provedores de aplicativos e serviços (por exemplo, Facebook, Google, Amazon, Microsoft, Apple, Netflix, Spotify) e, claro, assinantes e clientes (ou seja, indivíduos, mas também empresas e negócios). As linhas entre esses participantes nem sempre são nítidas, com muitas empresas desempenhando múltiplos papéis. O exemplo mais notável disso são os grandes provedores de nuvem, que (a) constroem seus próprios equipamentos de rede usando componentes básicos, (b) implantam e operam suas próprias redes e (c) fornecem serviços e aplicativos para o usuário final sobre suas redes.

Ao considerar esses outros fatores no processo de design técnico, percebe-se que há algumas premissas implícitas na versão clássica da história que precisam ser reavaliadas. Uma delas é que projetar uma rede é uma atividade única. Construa-a uma vez e use-a para sempre (atualizações de hardware modulares para que os usuários possam aproveitar os benefícios das melhorias de desempenho mais recentes). A segunda é que a tarefa de construir a rede é em grande parte dissociada da tarefa de operá-la. Nenhuma dessas premissas está totalmente correta.

O design da rede está claramente evoluindo, e documentamos essas mudanças a cada nova edição do livro didático ao longo dos anos. Fazer isso em um cronograma medido em anos historicamente tem sido suficiente, mas qualquer pessoa que tenha baixado e usado o aplicativo mais recente para smartphone sabe o quão glacialmente lento qualquer coisa medida em anos é para os padrões atuais. Projetar para a evolução precisa fazer parte do processo de tomada de decisão.

Quanto ao segundo ponto, as empresas que constroem redes são quase sempre as mesmas que as operam. Elas são conhecidas coletivamente como *operadoras de rede*

e incluem as empresas listadas acima. Mas, se voltarmos a buscar inspiração na nuvem, veremos que desenvolver e operar não se aplica apenas ao nível da empresa, mas também é a forma como as empresas de nuvem que mais se desenvolvem organizam suas equipes de engenharia: em torno do modelo *DevOps* . (Se você não conhece DevOps, recomendamos a leitura de *"Engenharia de Confiabilidade de Site: Como o Google Executa Sistemas de Produção"* para ver como ele é praticado.)

Tudo isso significa que as redes de computadores estão agora no meio de uma grande transformação, com as operadoras de rede tentando simultaneamente acelerar o ritmo da inovação (às vezes conhecido como velocidade de recurso) e, ainda assim, continuar a oferecer um serviço confiável (preservando a estabilidade). E elas estão fazendo isso cada vez mais adotando as melhores práticas dos provedores de nuvem, que podem ser resumidas em dois temas principais: (1) aproveitar o hardware comum e transferir toda a inteligência para o software e (2) adotar processos de engenharia ágeis que eliminem as barreiras entre o desenvolvimento e as operações.

Essa transformação é às vezes chamada de "nuvemificação" ou "softwarização" da rede e, embora a Internet sempre tenha tido um ecossistema de software robusto, historicamente ele se limitou aos aplicativos executados *na* rede (por exemplo, usando a API Socket descrita na Seção 1.4). O que mudou é que hoje essas mesmas práticas de engenharia inspiradas na nuvem estão sendo aplicadas aos *componentes internos* da rede. Essa nova abordagem, conhecida como *Redes Definidas por Software* (SDN), é um divisor de águas, não tanto em termos de como abordamos os desafios técnicos fundamentais de enquadramento, roteamento, fragmentação/remontagem, agendamento de pacotes, controle de congestionamento, segurança e assim por diante, mas em termos de quão rapidamente a rede evolui para suportar novos recursos.

Essa transformação é tão importante que a abordaremos novamente na seção *Perspectiva* , ao final de cada capítulo. Como essas discussões explorarão, o que acontece no setor de redes tem a ver, em parte, com a tecnologia, mas também, em parte, com muitos outros fatores não técnicos, todos eles uma prova de quão profundamente a internet está inserida em nossas vidas.